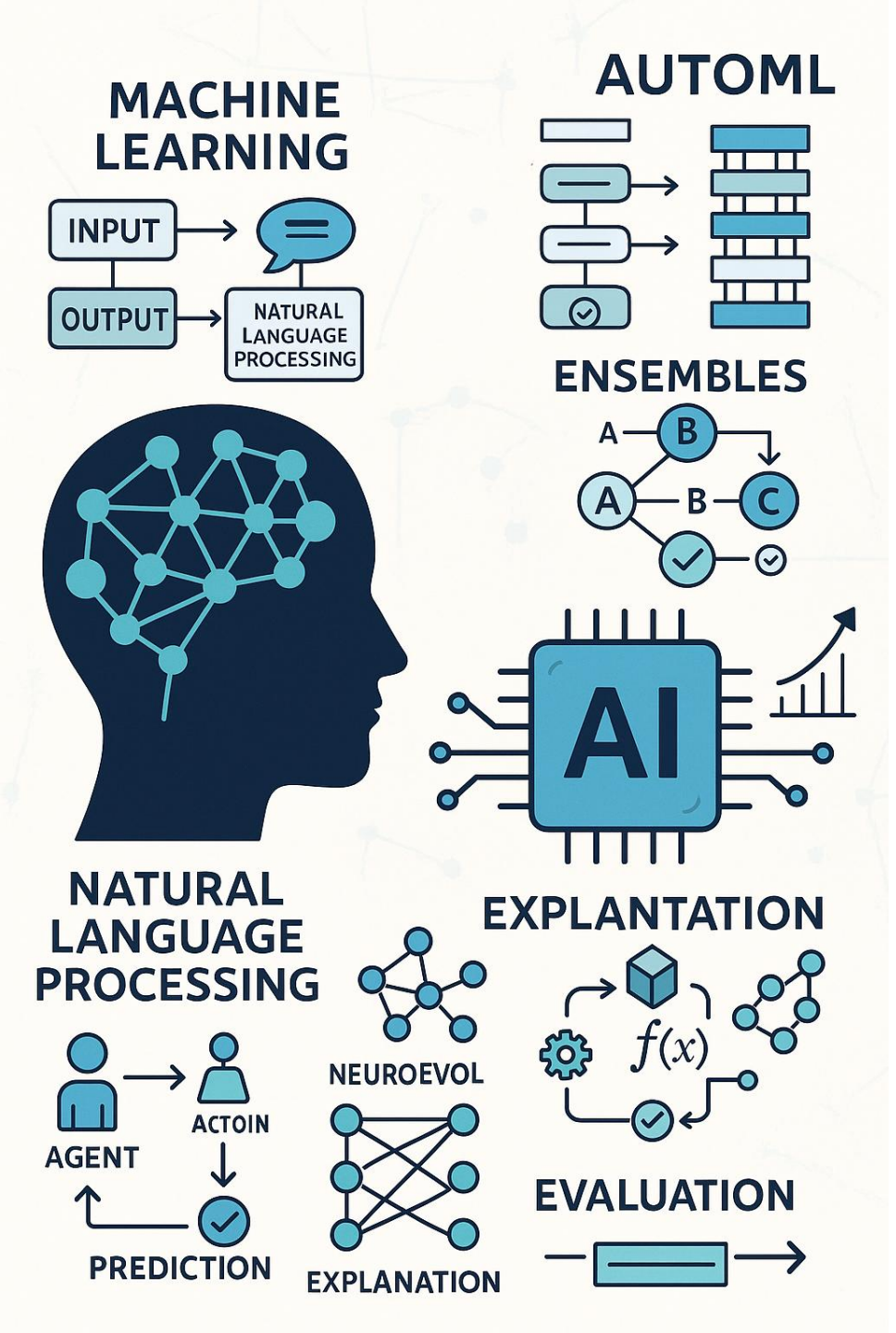
University of Ljubljana, Faculty of Computer and Information Science

Intelligent Sytems

Prof Dr Marko Robnik-Šikonja

Edition 2025



Lecturer



- Prof Dr Marko Robnik-Šikonja
- University of Ljubljana
  Faculty of Computer and Information Science
  Head of Machine Learning and Language Technlogies Lab
- FRI, Večna pot 113, 2nd floor, room 2.06, to the right from the elevator
- marko.robnik@fri.uni-lj.si
- https://fri.uni-lj.si/en/employees/marko-robnik-sikonja
- (01) 4798 241
- Contact hours (see the webpage)
  - currently, Wednesdays, 11:00 - 12:00; please, email me; other times or Zoom meeting are possible
- **Research interests**: artificial intelligence, machine learning, natural language processing, network analytics, data science
- **Teaching:** courses from areas of machine learning, natural language processing, and problem solving
- **Software and resources:** supporting open science, author of several open source ML packages, many large language models, and language resources

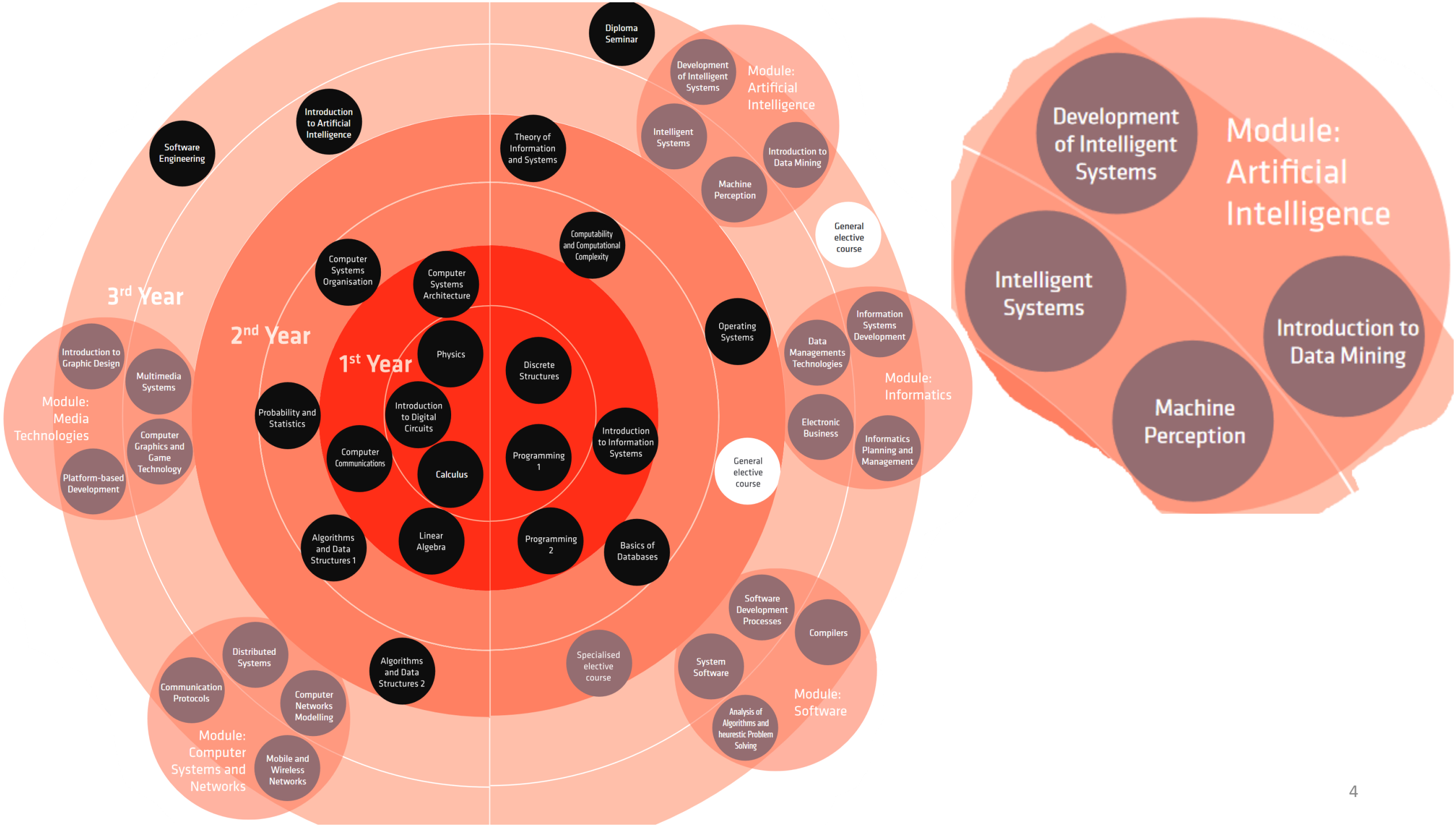
Assistants

- Dr Tadej Škvorc
- Timotej Knez, PhD student
- Boshko Koloski, PhD student





- tutorials, assignments, work in Python
- please, prepare questions!


Diploma Seminar
Development of Intelligent Systems
Module: Artificial Intelligence
Intelligent Systems
Introduction to Data Mining
Machine Perception
Introduction to Artificial Intelligence
Software Engineering
Theory of Information and Systems
General elective course
Computability and Computational Complexity
Computer Systems Organisation
Computer Systems Architecture
3rd Year
2nd Year
1st Year
Operating Systems
Information Systems Development
Data Managements Technologies
Module: Informatics
Electronic Business
Informatics Planning and Management
Introduction to Graphic Design
Multimedia Systems
Module: Media Technologies
Computer Graphics and Game Technology
Platform-based Development
Physics
Discrete Structures
Probability and Statistics
Introduction to Digital Circuits
Introduction to Information Systems
Computer Communications
Calculus
Programming 1
General elective course
Algorithms and Data Structures 1
Linear Algebra
Programming 2
Basics of Databases
Software Development Processes
Compilers
Distributed Systems
Specialised elective course
System Software
Communication Protocols
Computer Networks Modelling
Algorithms and Data Structures 2
Module: Software
Analysis of Algorithms and heurestic Problem Solving
Module: Computer Systems and Networks
Mobile and Wireless Networks

Development of Intelligent Systems
Module: Artificial Intelligence
Intelligent Systems
Introduction to Data Mining
Machine Perception

Syllabus

- nature inspired computing (genetic algorithms, genetic programming)
- basics of machine learning,
- bias, variance, generalization error, and overfitting
- representation learning and feature selection
- neural networks
- natural language processing
- ensemble methods
- kernel methods
- automated machine learning
- transformers for tabular data and time series
- model inference and explanation
- reinforcement learning

Objectives

1. students shall become acquainted with
    - nature inspired computing
    - machine learning
        - predictive modeling approaches
        - model selection and evaluation techniques
        - model comprehensibility and explanation
        - practical application of predictive modeling in python
    - natural language processing
    - reinforcement learning
2. practical use of theoretical knowledge on (almost) real-world problems ; for a given prediction problem students shall be able to
    - transform it to a form suitable for predictive modeling
    - select and train an appropriate predictive model
    - evaluate the model and present the results in a comprehensible form and language
3. awareness of domain expertise and ethical issues in data science
4. increase the (mental) problem-solving toolbox with
    - predictive modeling techniques
    - evolutionary optimization approaches
    - large language models
    - reinforcement learning
    - experiment design, result understanding, visualization, and explanation approaches

Concrete course goals

Be able to explain

- difference between different types of machine learning models
- properties of models: bias, variance, generalization, hypothesis language
- properties of the following models: kNN, decision rules, bagging, boosting, random forests, stacking, SVM, neural networks
- properties and purpose of evaluation approaches and metrics: cross-validation, bootstrapping, ROC curves, sensitivity, specificity etc.
- When and how to apply AutoML techniques
- inference methods for predictive methods and explanation of predictions
- when and why to apply reinfocement learning
- how to prepare and process text
- when and how and to optimize a problem using evolutionary algorithms

Build and evaluate models in Python

- visualize datasets and created models
- prepare data into a suitable form for modeling algorithms
- apply classification and regression models to solve a prediction task with a given data set
- build natural language classifier
- estimate error of models using statistically valid approaches
- select models and tune their parameters using cross-validation and bootstrapping
- apply AutoML techniques
- visualize models and explain their predictions
- given a new dataset, select an appropriate modeling technique and evaluate the created model

Syllabus explained

Nature inspired computing

- genetic algorithms
- genetic programming
- neuro-evolution

Introduction to statistical predictive modelling

- Learning as modelling: data, evidence, background knowledge, predictive models, hypotheses, learning as optimization, learning as search, criteria of success, inductive learning, generalization.
- Classification and regression: supervised and unsupervised learning, learning discrete and numeric functions, learning relations, learning associations.
- Simple classification models: nearest neighbor, decision rules

Model selection

- Bias and variance: error decomposition, trade-off, estimating bias and variance
- Generalization performance: training and testing set error, cross-validation, evaluation set, bootstrapping.
- Performance measures: confusion matrix, sensitivity and specificity, ROC curves, AUC, cost-based classification.
- Parameter tuning: regularization, search
- AutoML

Kernel methods

- SVM for classification and regression: kernels, support vectors, hyperplanes.
- SVM for more than two classes: one vs. one, one vs. all.

Ensemble methods

- Model averaging, why ensembles work.
- Tree based ensembles: bagging, boosting, random forests.
- MARS and AODE ensembles.
- Stacking, mixture of experts.

Neural networks

- perceptron,
- backpropagation,
- setting structure of networks
- deep neural networks
- transformer architecture
- autoencoders
- GANs
- neural embedings and representation learning

Explaining prediction models

- Model comprehensibility, visualization and knowledge discovery.
- General methodology for explaining predictive models.
- Model level and instance level explanations, methods SHAP, LIME, EXPLAIN, and IME.

Learning with special settings

- imbalanced data,
- multi-task learning,
- multi-label learning,
- Etc.

Natural language processing

- text preprocesing
- text representation
- text similarity
- text classification
- sentiment analysis
- generative models

Reinforcement learning

- basics
- Markov decision problem
- Q learning
- Deep RL

Course organisation

Obligations

- 5 quizzes
- Two projects, 50 points
- Written exam, 50 points

Grading

Obligation	% of total	subject to
Five quizzes	0%	≥ 50% alltogether
Projects	50%	≥ 50% each
Written exam	50%	≥ 50%

Learning materials

- Learning materials in Moodle
- Slides
- Quizzes
- Links and other materials

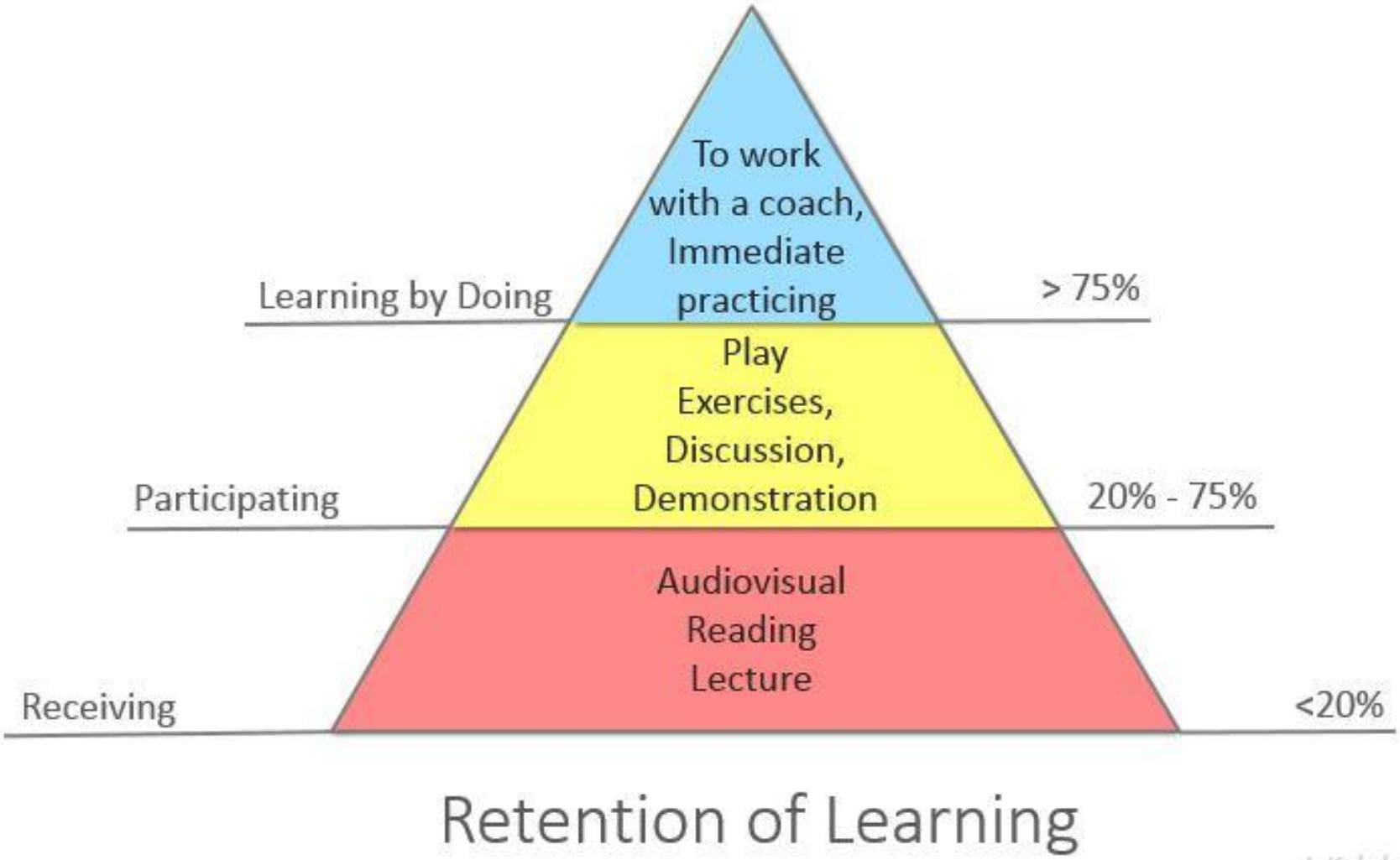
Readings (all freely available)

- James, G., Witten, D., Hastie, T., Tibshirani, R. and Taylor, J., 2023. *An Introduction to Statistical Learning: With Applications in Python*. New York: Springer. (also exists for R)
- Chollet, F. and Watson, M., 2025. *Deep Learning with Python, 3rd edition*. Manning.

Further readings:

- Jurafsky, Daniel and James, Martin (2025): Speech and Language Processing, 3rd edition in progress
- Richard S. Sutton and Andrew G. Barto: Reinforcement Learning, An Introduction, 2nd edition, MIT Press, 2018
- Kevin P. Murphy: Probabilistic Machine Learning: An Introduction. MIT Press, 2022
- Kevin P. Murphy: Probabilistic Machine Learning: Advanced Topics. MIT Press, 2023
- Friedman, J., Hastie, T., & Tibshirani, R., 2009). *The elements of statistical learning, 2nd edition*. Springer, Berlin
- scientific papers
- many excellent machine learning and data science courses on Coursera, edX etc.

Retention of learning

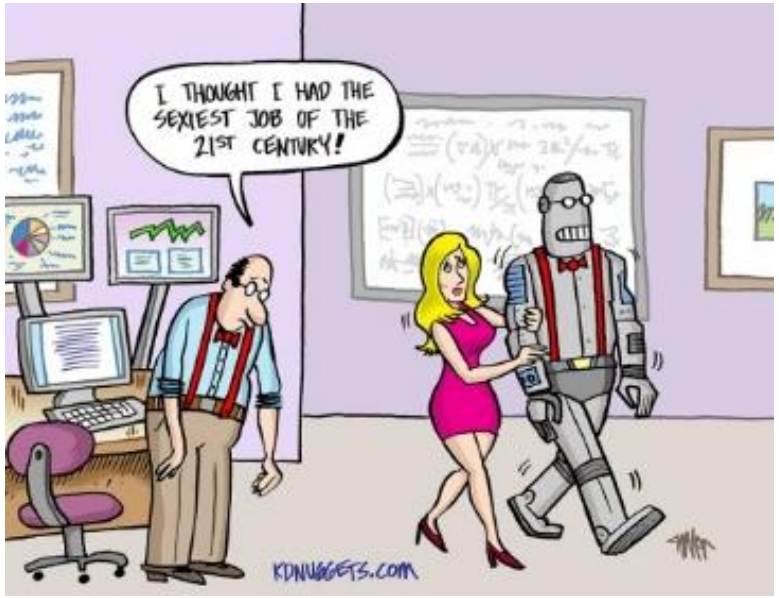


Retention of Learning

L Kokcharov © 2015

Data Science is a part of Intelligent Systems

- good job perspective
- many jobs in this area regularly occupy list of the most promising jobs
- Thomas H. Davenport, D.J. Patil: Data Scientist: The Sexiest Job of the 21st Century. *Harvard Business Review*, October 2012



MODERN DATA SCIENTIST

Data Scientist, the sexiest job of the 21th century, requires a mixture of multidisciplinary skills ranging from an intersection of mathematics, statistics, computer science, communication and business. Finding a data scientist is hard. Finding people who understand who a data scientist is, is equally hard. So here is a little cheat sheet on who the modern data scientist really is.

MATH & STATISTICS

- Machine learning
- Statistical modeling
- Experiment design
- Bayesian inference
- Supervised learning: decision trees, random forests, logistic regression
- Unsupervised learning: clustering, dimensionality reduction
- Optimization: gradient descent and variants



PROGRAMMING & DATABASE

- Computer science fundamentals
- Scripting language e.g. Python
- Statistical computing packages, e.g., R
- Databases: SQL and NoSQL
- Relational algebra
- Parallel databases and parallel query processing
- MapReduce concepts
- Hadoop and Hive/Pig
- Custom reducers
- Experience with xaaS like AWS

DOMAIN KNOWLEDGE & SOFT SKILLS

- Passionate about the business
- Curious about data
- Influence without authority
- Hacker mindset
- Problem solver
- Strategic, proactive, creative, innovative and collaborative

COMMUNICATION & VISUALIZATION

- Able to engage with senior management
- Story telling skills
- Translate data-driven insights into decisions and actions
- Visual art design
- R packages like ggplot or lattice
- Knowledge of any of visualization tools e.g. Flare, D3.js, Tableau

DATA SCIENTIST

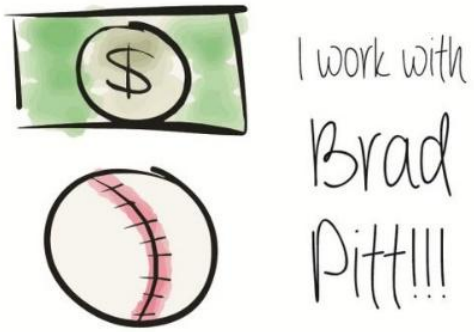
What my **CUSTOMERS** think I do



What my **MUM** thinks I do



What my **FRIENDS** think I do



What my **HUSBAND** thinks I do



What **I** think I do



What I **ACTUALLY** do



Intelligent systems and media



Will robots destroy us?
Will they take our jobs?
Will we still need a driving licence?
Will we still need doctors?
How will humanoid robots evolve?
What about cyborgs?
What is artificial general intelligence?
What is technological singularity?

New prophets of tehnological singularity

Elon Musk says humans must become cyborgs to stay relevant. Is he right?

Sophisticated artificial intelligence will make 'house cats' of humans, claims the entrepreneur, but his grand vision for mind-controlled tech may be a long way off





Some scientific opinions

- Rodney Brooks: The Seven Deadly Sins of Predicting the Future of AI. https://rodneybrooks.com/the-seven-deadly-sins-of-predicting-the-future-of-ai/ also in MIT Technology Review

- Marko Robnik-Šikonja: Is artificial intelligence a (job) killer?. The Conversation, Jul. 2017 https://theconversation.com/is-artificial-intelligence-a-job-killer-80473

- …



Short history of optimism



- starting in 1950s,
  1956 Dartmouth conference
- great expectations, enormous underestimation
  of problem difficculty
- AI winter (2 x)

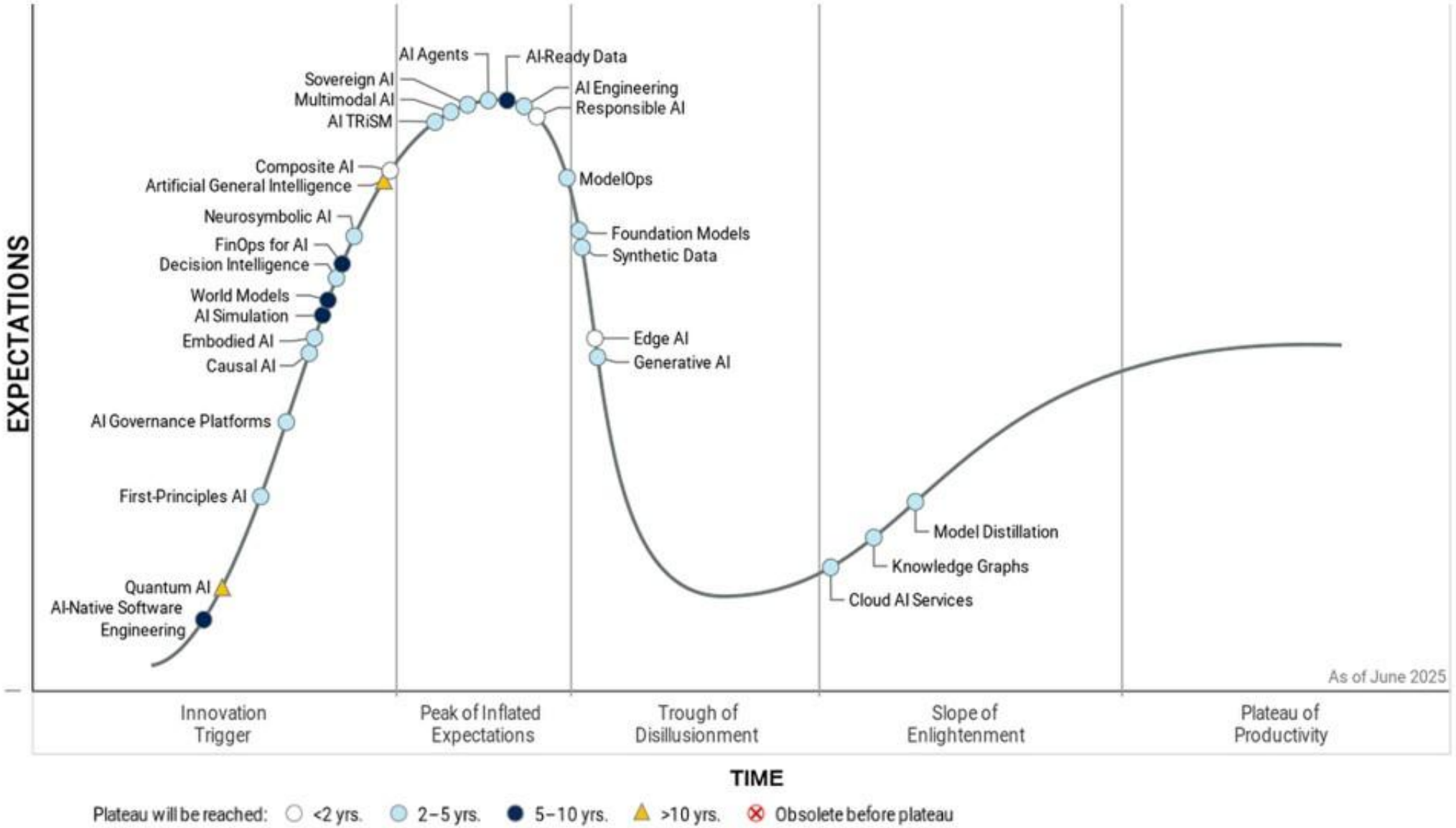
1958, H. A. Simon and Allen Newell: “… within ten years a digital computer will discover and prove an important new mathematical theorem.”

1965, H. A. Simon: “… machines will be capable, within twenty years, of doing any work a man can do.”

1967, Marvin Minsky: "Within a generation ... the problem of creating 'artificial intelligence' will substantially be solved.”

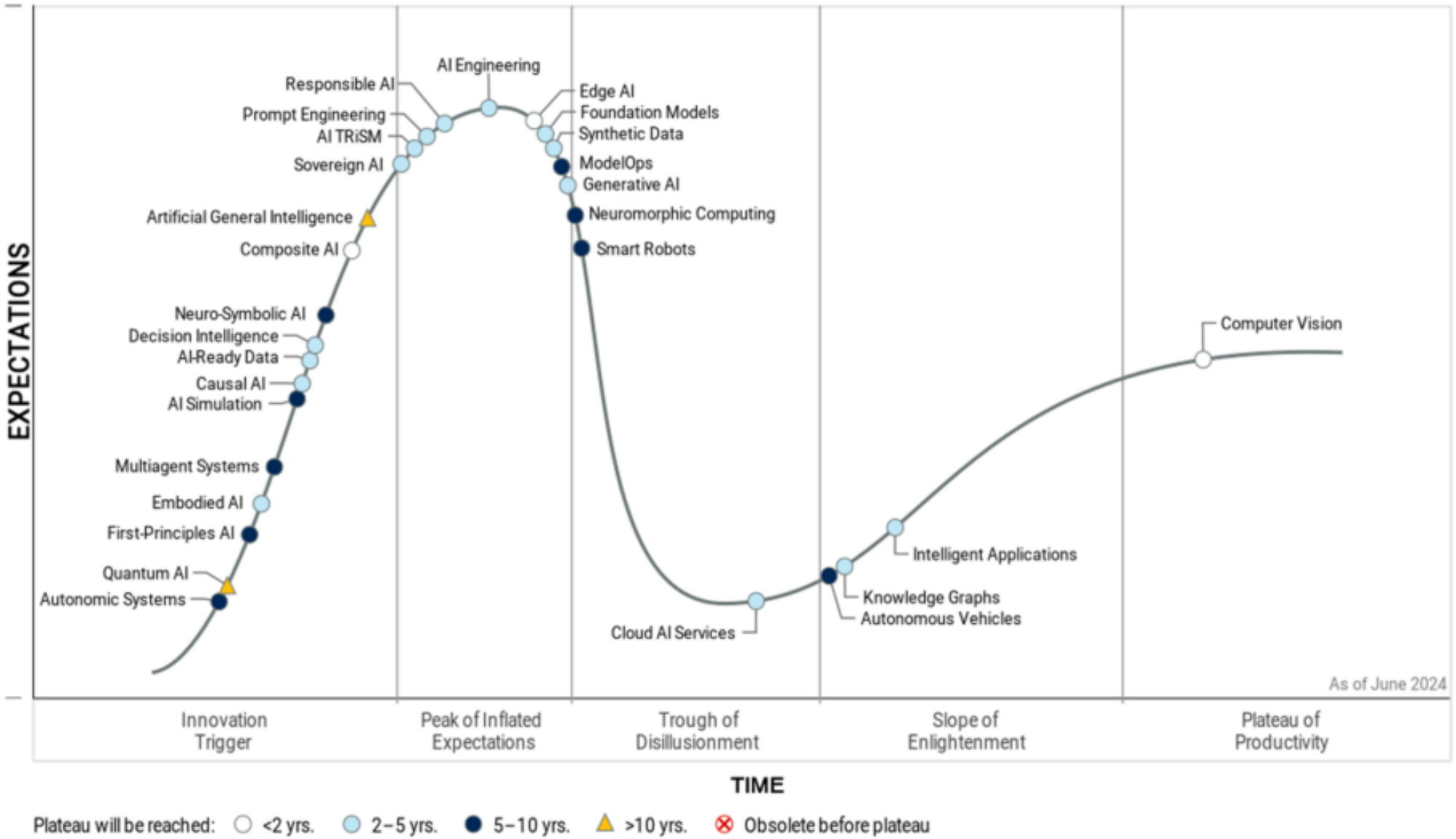
1970, Marvin Minsky: "In from three to eight years we will have a machine with the general intelligence of an average human being."

Hype Cycle for Artificial intelligence, 2025



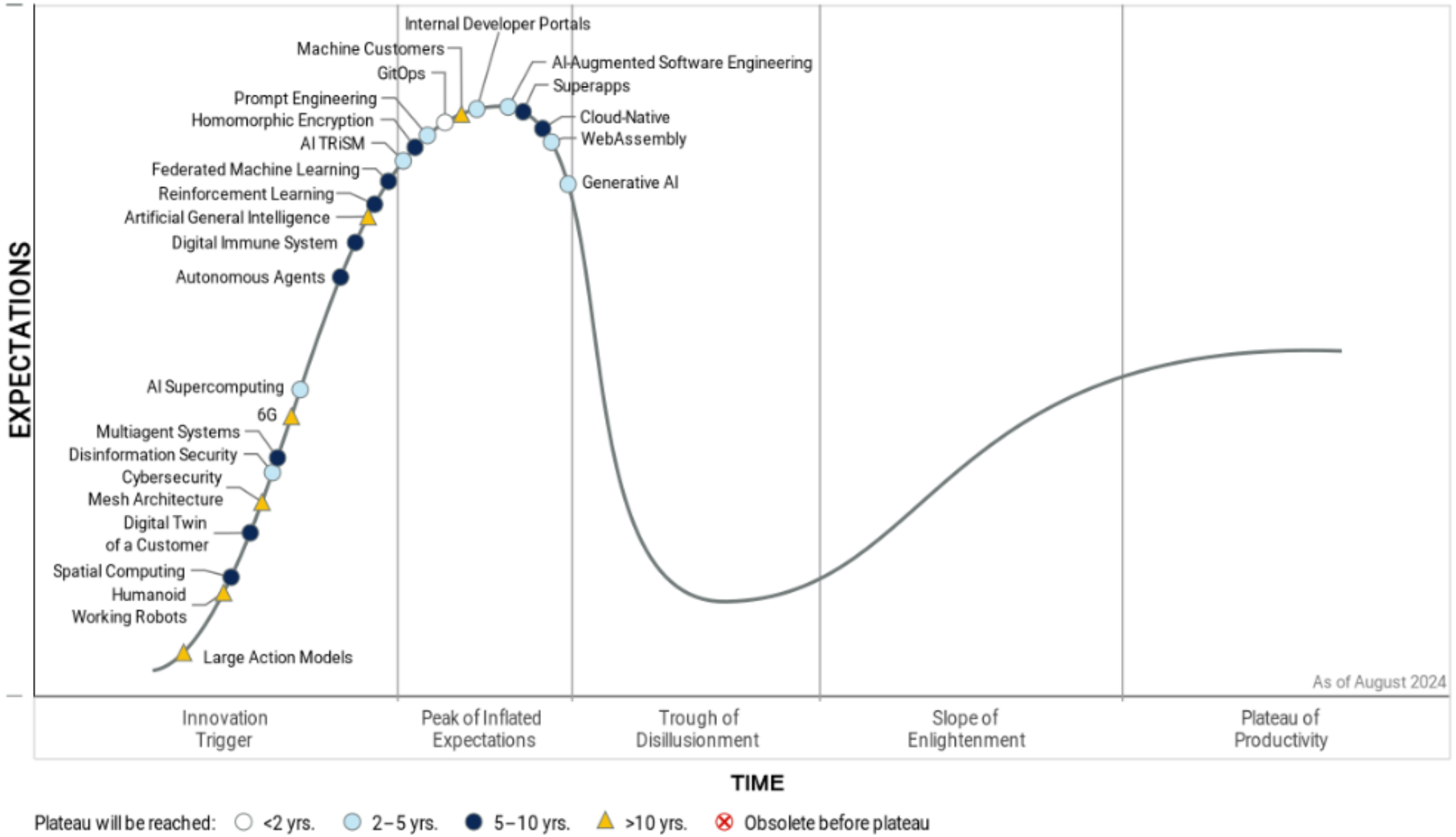
Gartner

Hype Cycle for Artificial Intelligence, 2024



Gartner

Hype Cycle for Emerging Technologies, 2024



Gartner

University of Ljubljana, Faculty of Computer and Information Science

Nature inspired computing



Prof Dr Marko Robnik Šikonja

Intelligent Systems

Edition 2025

Contents

- Introduction to evolutionary computation
- Genetic algorithms
- Genetic algorithms and automatic code generation

Evolutionary and natural computation

- Many engineering and computational ideas from nature work fantastically!
- Evolution as an algorithm
- Abstraction of the idea:
  - progress, adaptation - learning, optimization
- Survival of the fittest - competition of agents, programs, solutions
- Populations – parallelization
- (Over)specialization – local extremes
- Neuro-evolution, evolution of robots, evolution of novelty
- revival of interest

Template of evolutionary program

generate a population of agents (objects, data structures)

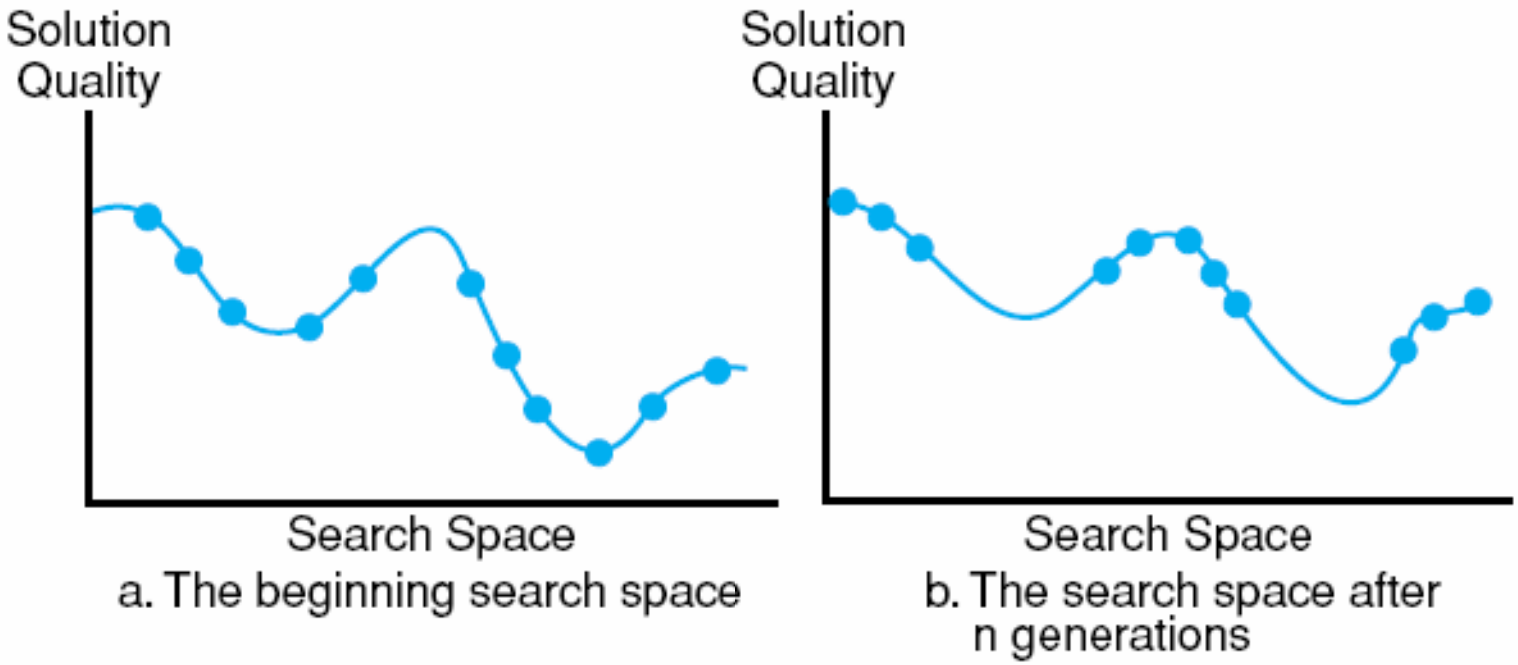
do {

compute fitness (quality) of the agents
select candidates for the reproduction using fitness
create new agents by combining the candidates
replace old agents with new ones

} while (not satisfied)

- immensely general -> many variants

A result of successful evolutionary program



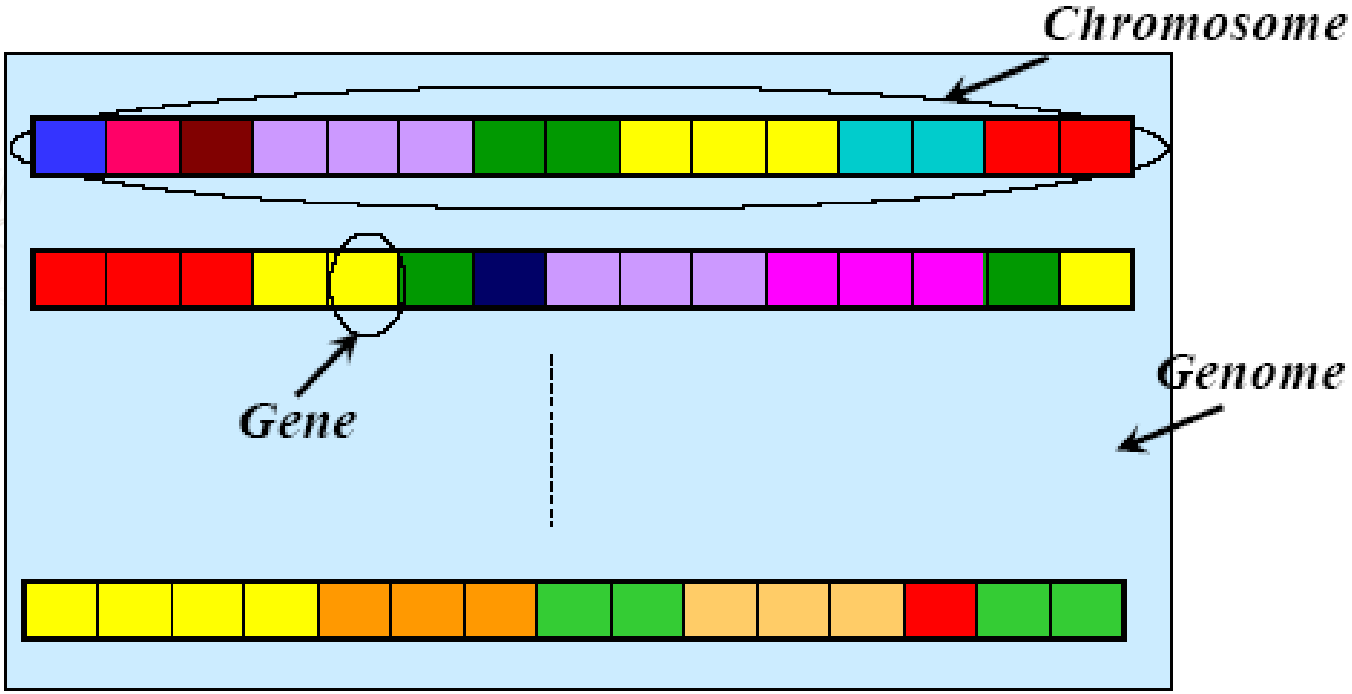
Main approaches to nature inspired computing

- Genetic algorithms
- Genetic programming
- Differential evolution
- Swarm methods (particles, ants, bees, …)
- Physics methods: simulated annealing
- etc.

Genetic Algorithms - GA

- Pioneered by John Holland in the 1970's
- Got popular in the late 1980's
- Based on ideas from Darwinian evolution
- Can be used to solve a variety of problems that are not easy to solve using other techniques
- Revival of interest in connection with neuroevolution

Chromosome, Genes and Genomes



- Only a weak analogy to GA

Genome representation

- Bit vector
- Numeric vectors
- Strings
- Permutations
- Trees: functions, expressions, programs
- ...

Crossover

- Single point/multipoint
- Shall preserve individual objects

Crossover: bit representation

Parents: 1101011100 0111000101

Children: 1101010101 0111001100

Crossover: vector representation

Simplest form

Parents: (6.13, 4.89, 17.6, 8.2) (5.3, 22.9, 28.0, 3.9)

Children: (6.13 , 22.9, 28.0, 3.9) (5.3, 4.89, 17.6, 8.2)

In reality: linear combination of parents

Linear crossover

- The linear crossover simply takes a linear combination of the two individuals.
- Let $x = (x_1, \ldots x_N)$ and $y = (y_1, \ldots y_N)$
- Select α in $(0, 1)$
- The results of the crossover is $\alpha x + (1- \alpha)y$.
- Possible variation: choose a different α for each position.

Linear crossover example

- Let $\alpha = 0.75$ and we have two individuals:

  $A = (5, 1, 2, 10)$ and $B = (2, 8, 4, 5)$

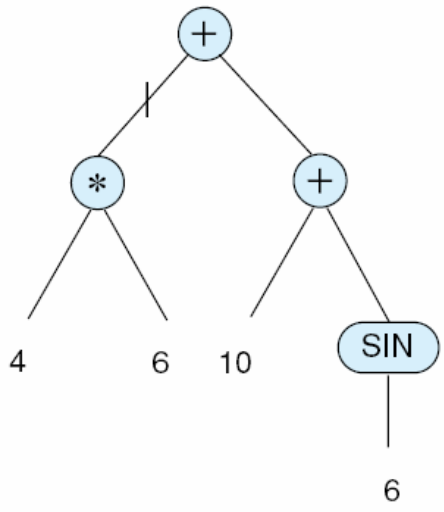
- then the result of the crossover is $\alpha A + (1- \alpha) B$

$(3.75 + 0.5, 0.75 + 2, 1.5 + 1, 7.5 + 1.25) = (4.25, 2.75, 2.5, 8.75)$

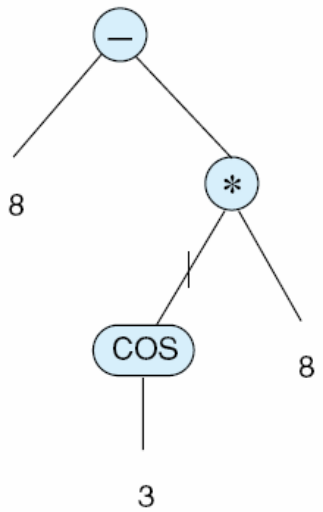
- If we use the variation and we have $\alpha = (0.5, 0.25, 0.75, 0.5)$, the result is:

$(2.5 + 1, 0.25 + 6, 1.5 + 1, 5 + 2.5) = (3.5, 6.25, 2.5, 7.5)$

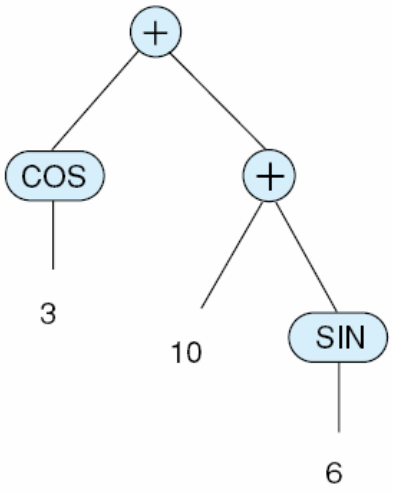
Crossover: trees



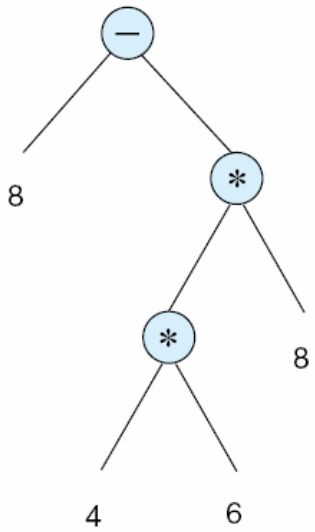
a.



b.



a.



b.

Permutations: travelling salesman problem

- 9 cities: 1,2 ..9
- bit representation using 4 bits?
  - 0001 0010 0011 0100 0101 0110 0111 1000 1001
  - crossover would give invalid genes
- permutations and ordered crossover
  - keep (part of) sequences
  - use the sequence from second cut, keep already existing

1 9 2 | 4 6 5 7 | 8 3 → x x x | 4 6 5 7 | x x ↘ 2 3 9 | 4 6 5 7 | 1 8

4 5 9 | 1 8 7 6 | 2 3 → x x x | 1 8 7 6 | x x ↗ 3 9 2 | 1 8 7 6 | 4 5

A demo: Eaters

- Plant eaters are simple organisms, moving around in a simulated world and eating plants
- Fitness function: number of plants eaten
- An eater sees one square in front of its pointed end; it sees 4 possible things: another eater, plant, empty square or the wall
- Actions: move forward, move backward, turn left, turn right
- It is not allowed to move into the wall or another eater
- Internal state: number between 0 and 15
- The behavior is determined by the 64 rules encoded in its chromosome; one rule for each of 16 states x 4 observations; one rule is a pair (action, next state)
- The chromosome therefore consists of length 64 x (4+2) bits = 384 bits
- Crossover and mutation

Gray coding of binary numbers

- Keeping similarity
- Similar object shall have similar genome

Binary	Gray
0000	0000
0001	0001
0010	0011
0011	0010
0100	0110
0101	0111
0110	0101
0111	0100
1000	1100
1001	1101
1010	1111
1011	1110
1100	1010
1101	1011
1110	1001
1111	1000

Adaptive crossover

- Different evolution phases
- Crossover templates
- 0 – first parent, 1 second parent
- Possibly different dynamics of template

	Gene	Template
Parent 1	1.2 3.4 5.6 4.5 7.9 6.8	010101
Parent 2	4.7 2.3 1.6 3.2 6.4 7.7	011100
Child 1	1.2 2.3 5.6 3.2 7.9 7.7	010100
Child 2	4.7 3.4 1.6 4.5 6.4 6.8	011101

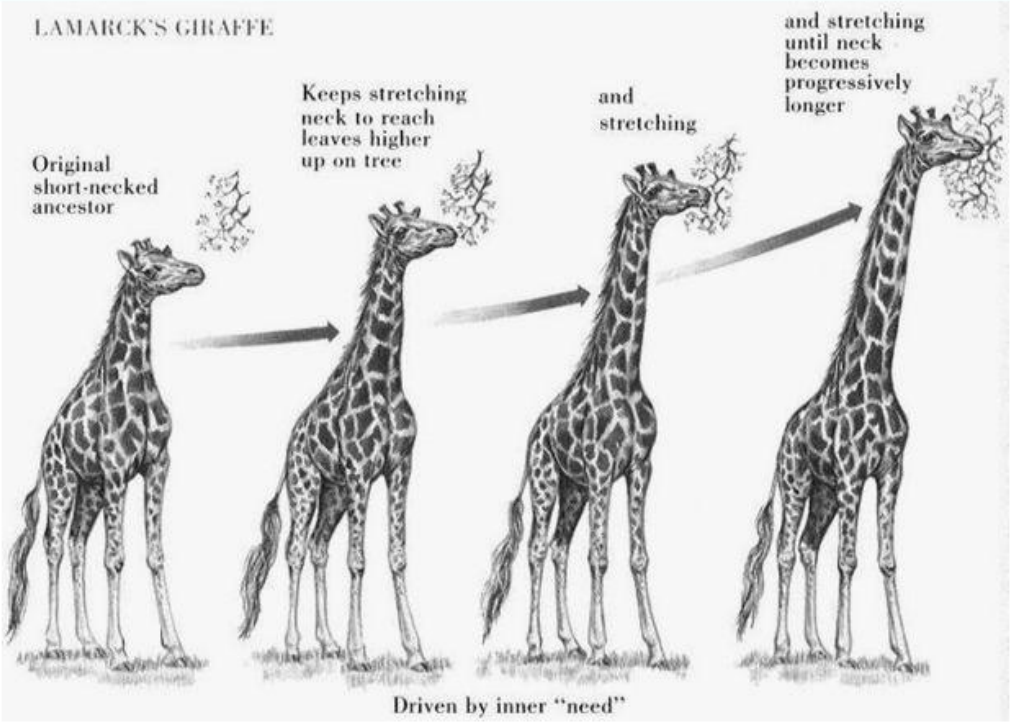
Mutation

- Adding new information
- Binary representation:
  0111001100 --> 0011001100
- Single point/multipoint
- Random search?
- Lamarckian (searching for locally best mutation)

Lamarckianism

Lamarckism is the hypothesis that an organism can pass on characteristics that it has acquired through use or disuse during its lifetime to its offspring.

An Early Proposal of Evolution: Theory of Acquired Characteristics



Jean Baptiste Lamarck (~ 1800) : Theory of Acquired Characteristics

- Use and disuse alter shape and form in an animal
- Changes wrought by use and disuse are heritable
- Explained how a horse-like animal evolved into a giraffe



Gaussian mutation

- When mutating one gene, selecting the new value by choosing uniformly among all the possible values is not the best choice (empirically).
- The mutation selects a position in the vector of floats and mutates it by adding a Gaussian error: a value extracted according to a normal distribution with the mean o and the variance depending on the problem.

Template of evolutionary program

generate a population of agents (objects, data structures)

do {

compute fitness (quality) of the agents
select candidates for the reproduction using fitness
create new agents by combining the candidates
replace old agents with new ones

} while (not satisfied)

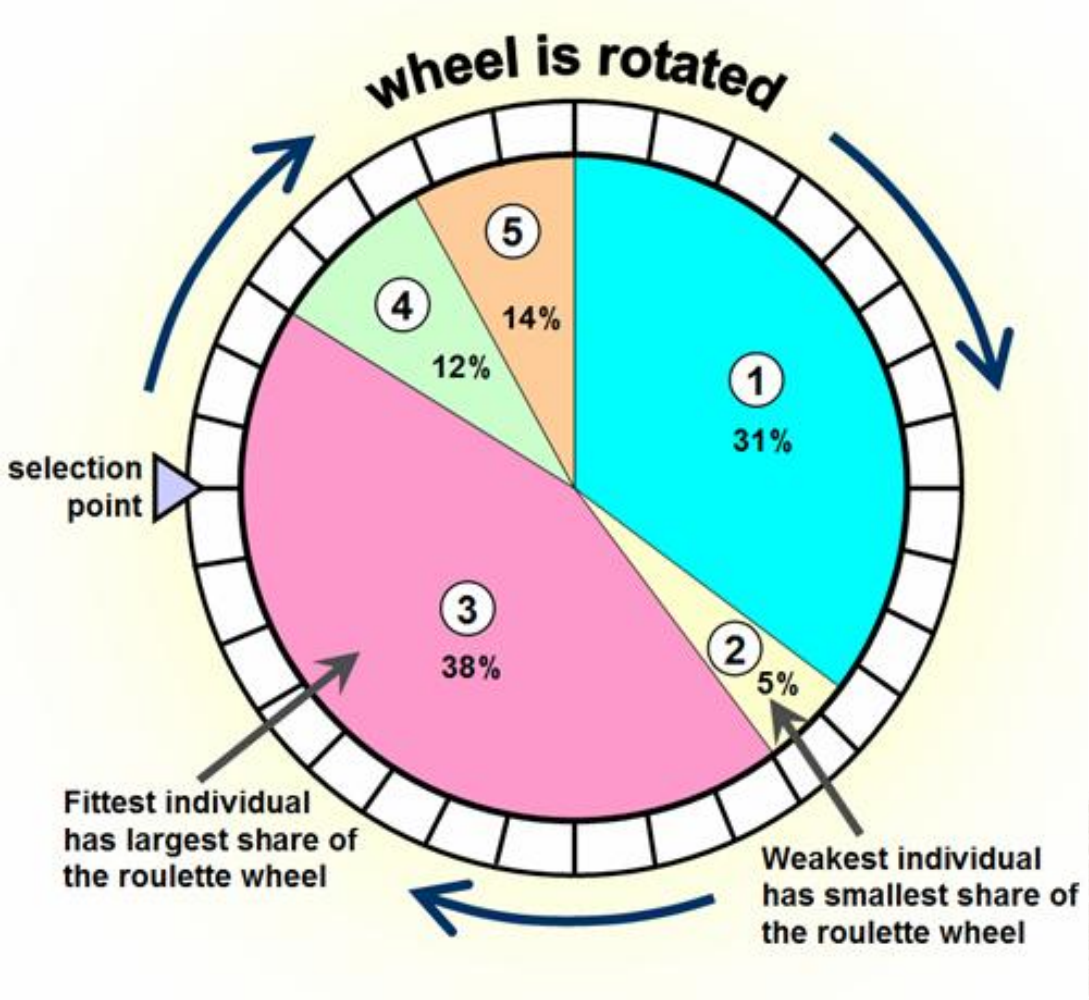
- immensely general -> many variants

Evolutional model - who will reproduce

- Keep the good
- Prevent premature convergence
- Assure heterogeneity of population

Selection

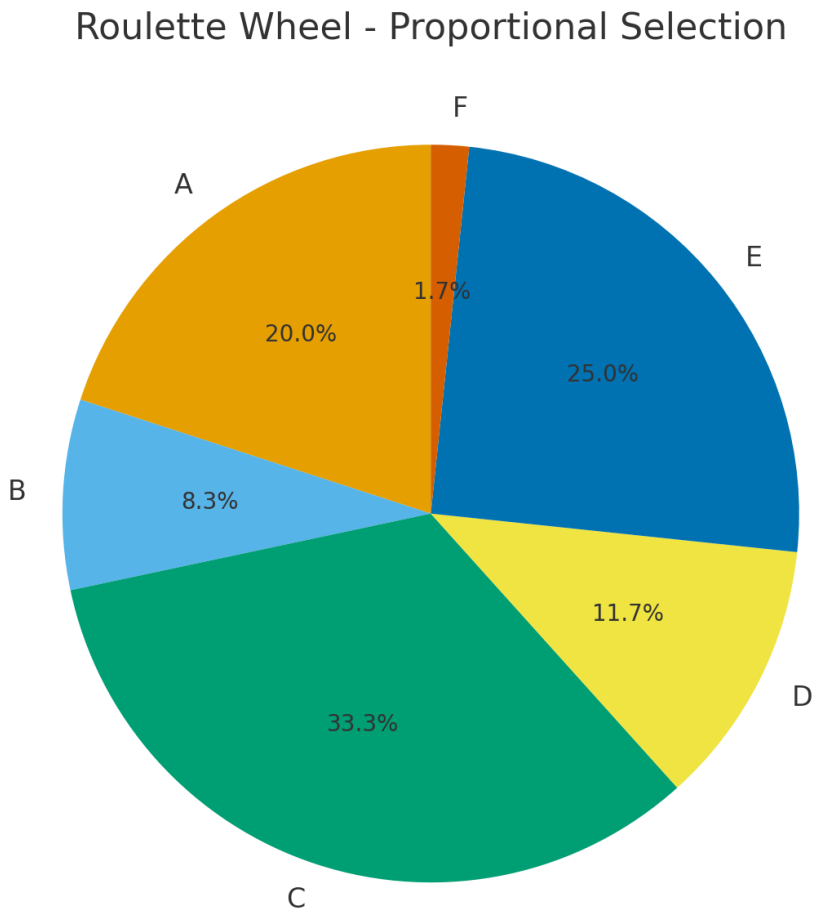
- Proportional
- Rank proportional
- Tournament
- Single tournament
- Stochastic universal sampling

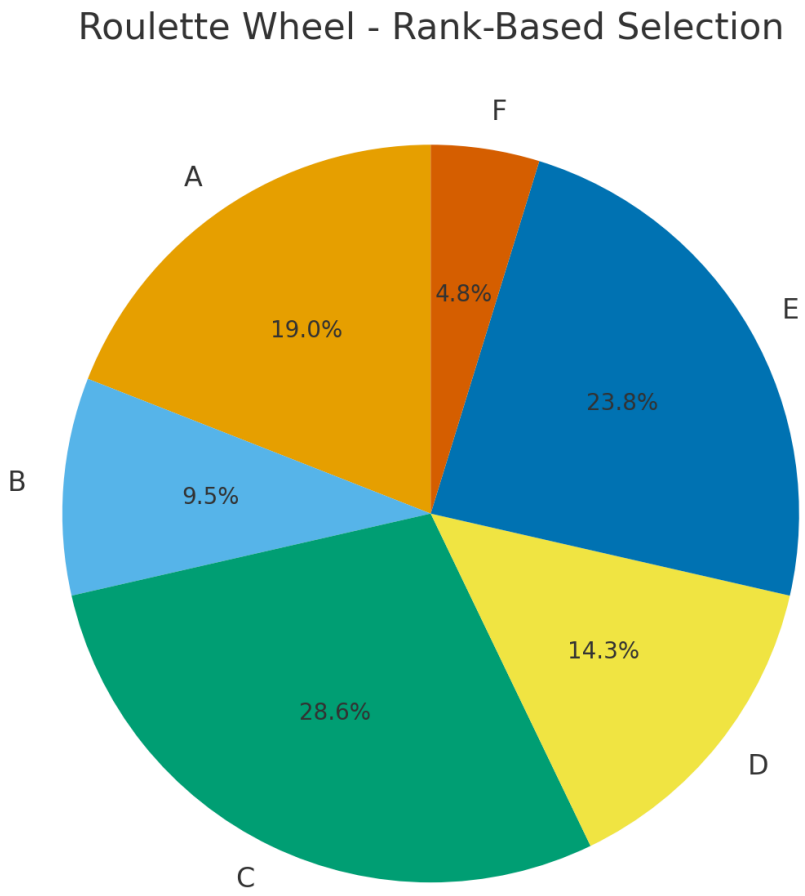


Proportional and rank based selection example

Agent	Fitness	p_{prop}	Cum_{prop}	Rank	p_{rank}	Cum_{prop}
A	12	0.200	0.200	4	0.190	0.190
B	5	0.083	0.283	2	0.095	0.286
C	20	0.333	0.617	6	0.286	0.571
D	7	0.117	0.733	3	0.143	0.714
E	15	0.250	0.983	5	0.238	0.952
F	1	0.017	1.000	1	0.049	1.000
Sum	**60**	**1.000**		**21**	**1.000**	

Roullete wheels for the proportional and rank based selection example





Tournament selection

- Several variants of tournaments

Probabilistic tournaments

1. set t=size of the tournament,
   p=probability of a choice
2. randomly sample t agents from population forming a tournament
3. select the best with probability p
4. select second best with probability $p(1-p)$
5. select third best with probability $p(1-p)^2$
6. …

End when the mating pool is large enough

Replacement

- All
- According to the fitness (roulette, rank, tournament, randomly)
- Elitism (keep a portion of the best)
- Local elitism (children replace parents if they are better)

Single tournament selection

1. randomly split the population into small groups
2. apply crossover to two best agents from each group; their offspring replace two worst agents from the group

- advantage: in groups of size *g* the best g-2 progress to next generation (we do not loose good agents, maximal quality does not decrease)
- no matter the quality even the best agents have no more than two offspring (we do not loose population diversity)
- Computational load? Speed?

Population size

- small, large?
- Considerations?

Niche specialization

- evolutionary niches are generally undesired
- punish too similar agents
- modify fitness

$f'_i = f_i / q(i)$

$q(i) = \{ 1 \quad ; sim(i) <= 4,$
$\quad sim(i)/4 \quad ; otherwise \},$

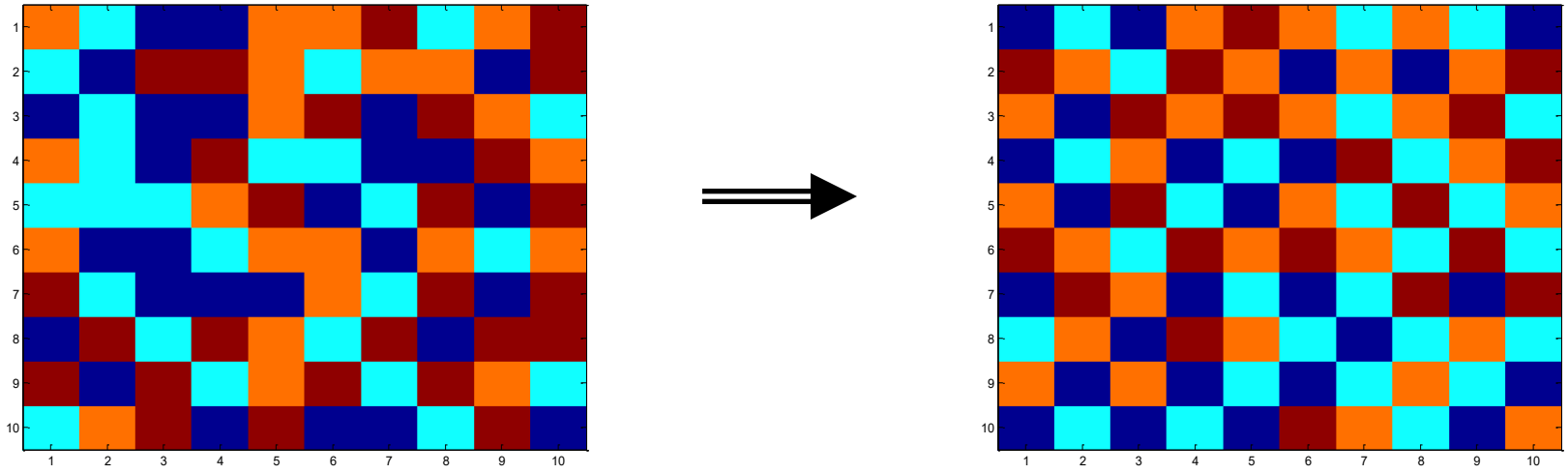
where sim(i) is the number of very similar agents to agent i

Stopping criteria

- number of generations, tracking of progress, availability of computational resources, leaderboard, mutability heuristics, etc.

Checkboard example

- We are given an ***n*** by ***n*** checkboard in which every field can have a different colour from a set of four colors.
- Goal is to achieve a checkboard in a way that there are no neighbours with the same color (not diagonal)

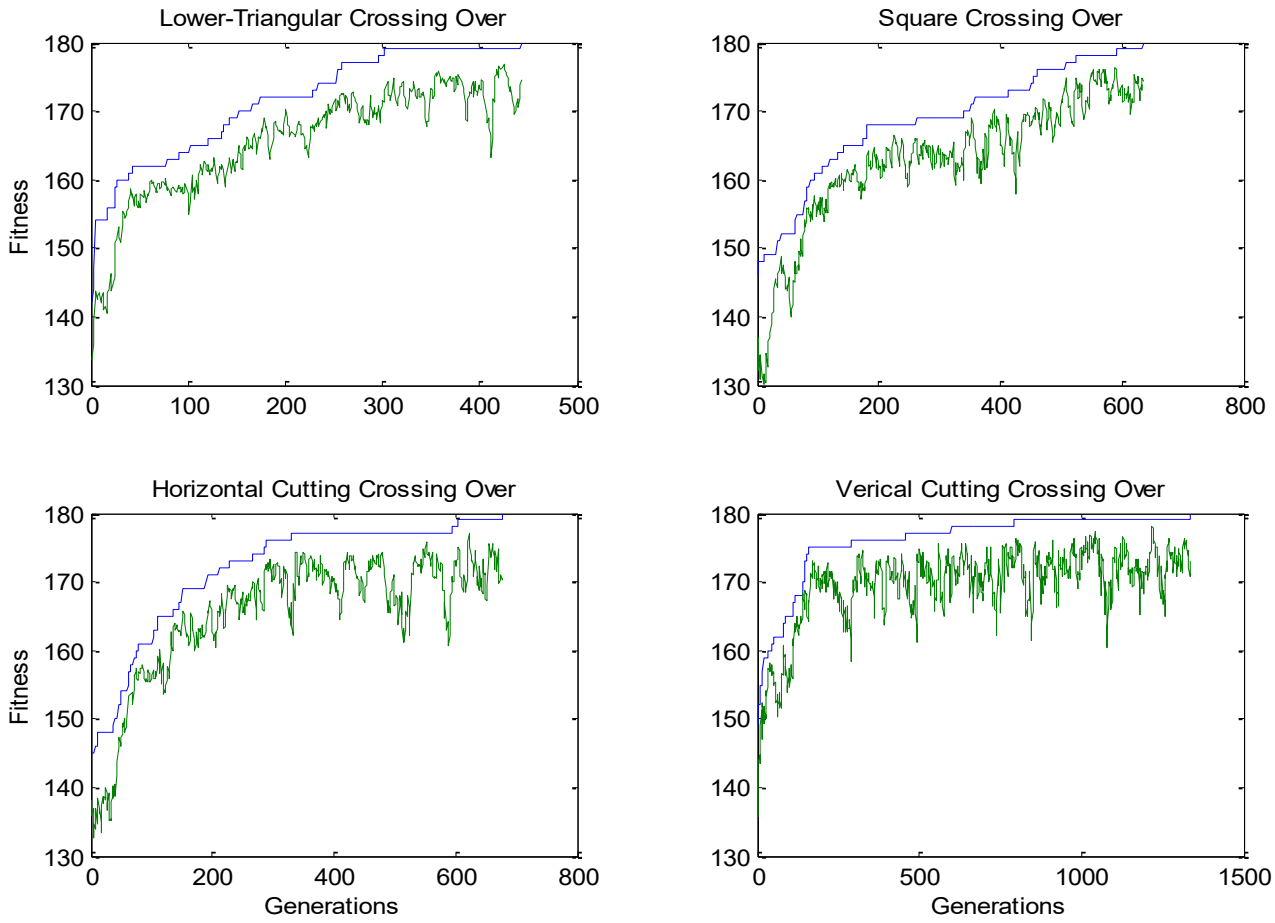


Checkboard example Cont'd

- Chromosomes represent the way the checkboard is colored.
- Chromosomes are not represented by bitstrings but by **bitmatrices**
- The bits in the bitmatrix can have one of the four values 0, 1, 2 or 3, depending on the color.
- Crossover involves matrix manipulation instead of point wise operating.
- Crossover can combine the parential matrices in a horizontal, vertical, triangular or square way.
- Mutation remains bitwise - changing bits
- Fitness function: check 2n(n-1) violations

Checkboard example Cont'd

- Fitness curves for different cross-over rules:



Why genetic algorithms work?

- building blocks hypothesis
- … is controversial (mutations)
- sampling based hypothesis

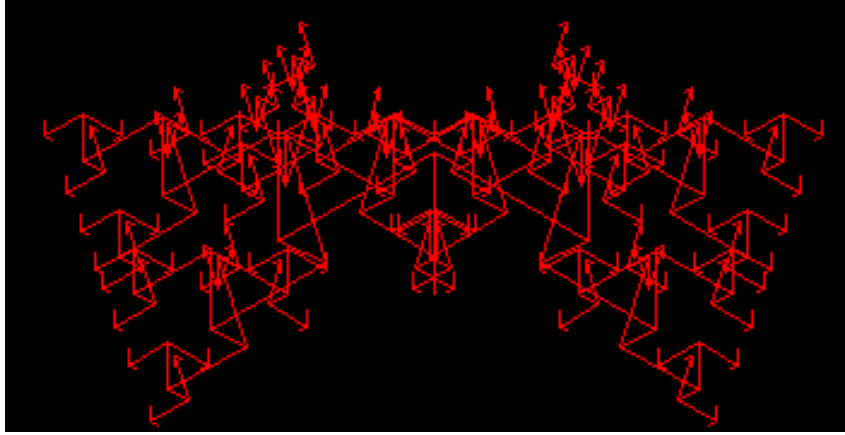
Parameters of GA

- Encoding (into fixed length strings)
- Length of the strings;
- Size of the population;
- Selection method;
- Probability of performing crossover (p_c);
- Probability of performing mutation (p_m);
- Termination criteria (e.g., a number of generations, a leaderboard mutability, a target fitness).

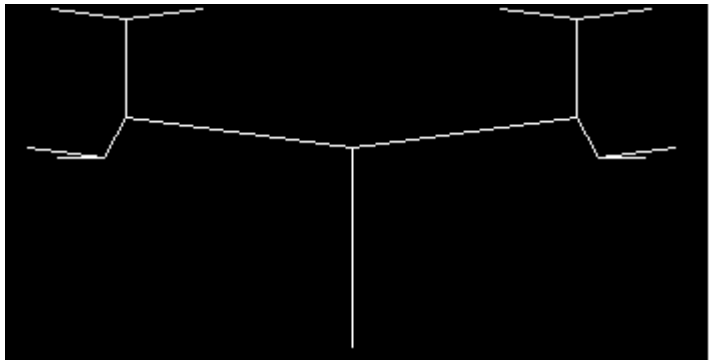
Usual settings of GA parameters

- Population size: from 20–50 to a few thousands individuals;
- Crossover probability: high (around 0.9);
- Mutation probability: low (below 0.1).

Demo: find genome of a biomorph



- A biomorph is a graphic configuration generated from nine genes.
- The first eight genes each encode a length and a direction.
- The ninth gene encodes the depth of branching.
- Each gene is encoded with five bits.
  - The four first bits represent the value, the fifth its sign.
  - Each gene can get a value from -15 to +15.
  - value of gen nine is limited to 2-9.



- There are : 8 (number of possible depths) x 2^{40} (the 8 * 5 =40 bits encoding basic genes) = 8.8 x10^{12} possible biomorphs. If we were able to test 1000 genomes every second, we would need about 280 years to complete the whole search.
- At the beginning, the drawing algorithm being known, we get the image of a biomorph. The only data directly measurable are the positions of branching points and their number. The basic algorithm simulates the collecting of these data.
- Fitness function: the distance of the generated biomorph from the target one.

Applications

- optimization
- scheduling
- bioinformatics,
- machine learning
- planning
- multicriteria optimization

Where to use evolutionary algorithms?

- Many local extremes
- Just fitness, without derivations
- No specialized methods
- Multiobjective optimization
- Robustness
- Combined with other approaches

Multiobjective optimization

- Fitness function with several objectives
- Cost, energy, environmental impact, social acceptability, human friendliness
- min $F(x)=\min (f_1(x), f_2(x), \ldots, f_n(x))$
- Pareto optimal solution: we cannot improve one criteria without getting worse on others
- GA: in reproduction, use all criteria

An example: smart buildings

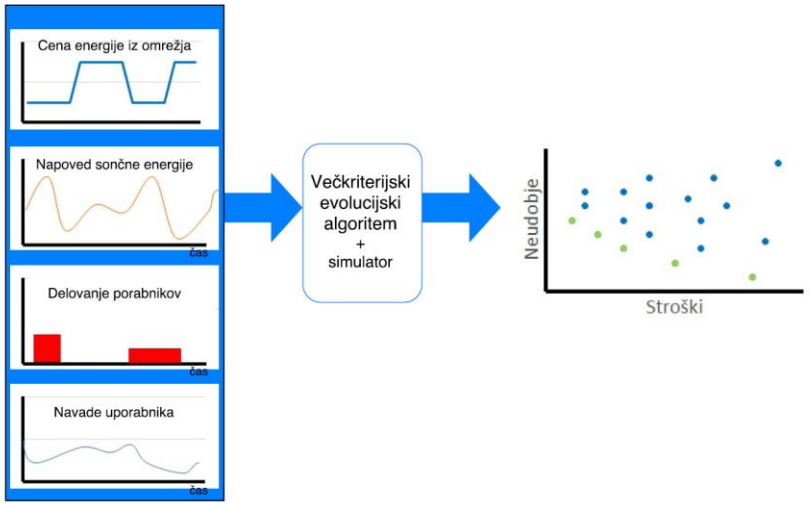


- simple scenario: heater, accumulator, solar panels, electricity from grid
- criteria: price, comfort of users (as the difference in temperature to the desired one)
- chromosome: shall encode schedule of charging and discharging the battery, heating on/off
- operational time is discretized to 15min intervals

Control problem for smart buildings

Parameters:

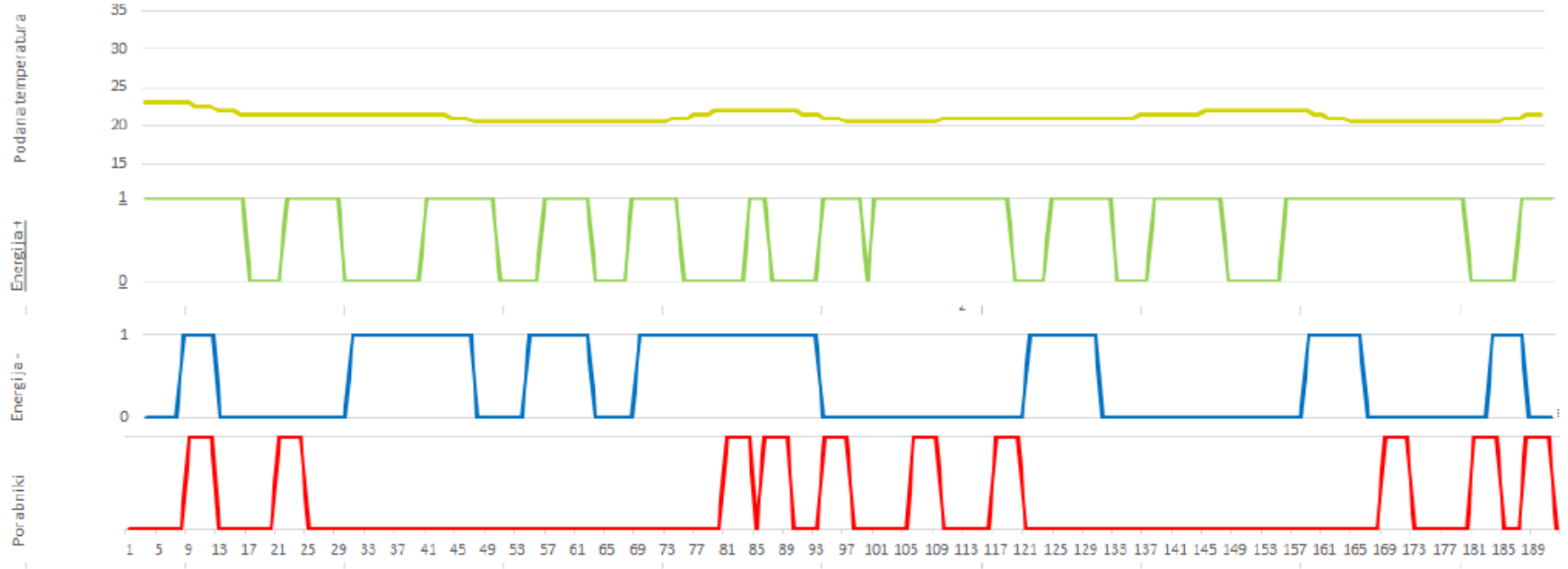
- the price of energy from the grid varies during the day
- the prediction of solar activity
- schedule of heater and battey
- usual activities of a user



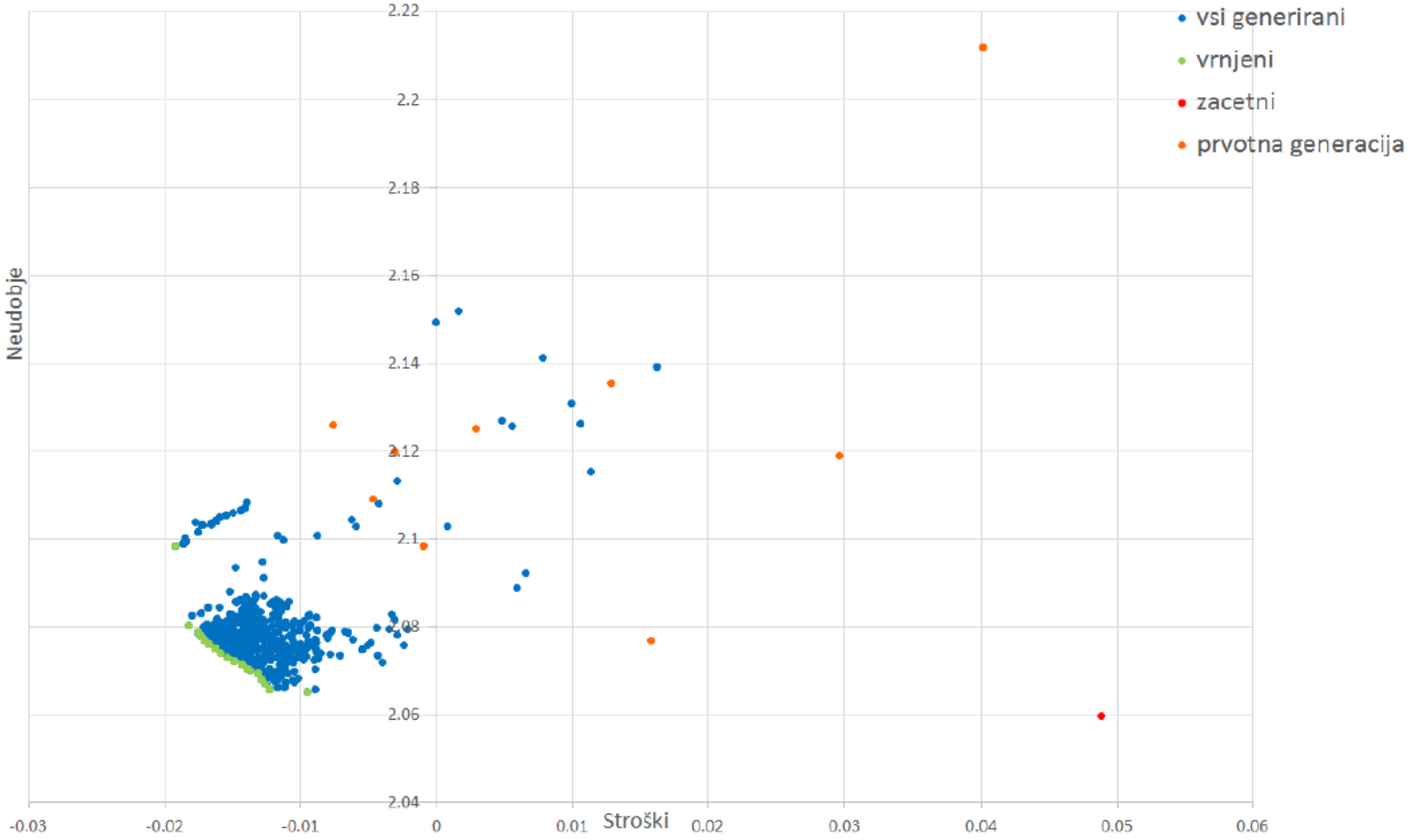
Smart building: structure of the chromosome

- temperature: for each interval we set the desired temperature between Tmin and Tmax interval
- battery+: if photovoltaic panels produce enough energy we set: 1 charging, 0 no charging
- battery-: if photovoltaic panels do not produce enough energy, we set: 1 battery shall discharge, 0 battery is not used
- appliances: each has its schedule when it is used (1) and when it is off (0)

Example of schedule



Example of solutions and optimal front



Strengths and weaknesses

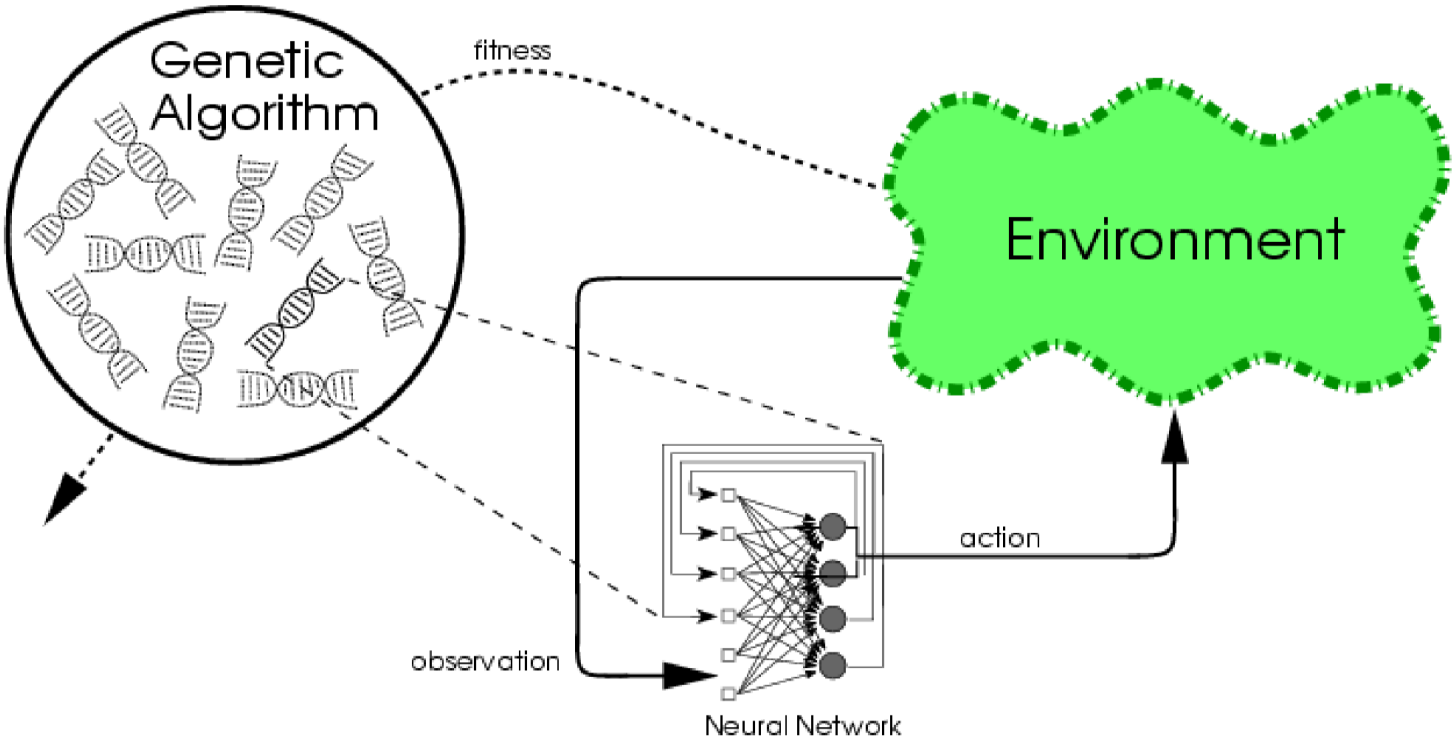
- robust, adaptable, general
- requires only weak knowledge of the problem (fitness function and representation of genes)
- several alternative solutions
- hybridization and parallelization
- faster and less memory than exhaustive or random search
- little effort to try

- suboptimal solutions
- possibly many parameters
- may be computationally expensive

- no-free-lunch theorem

Neuroevolution: evolving neural networks

- Evolving neurons and/or topologies

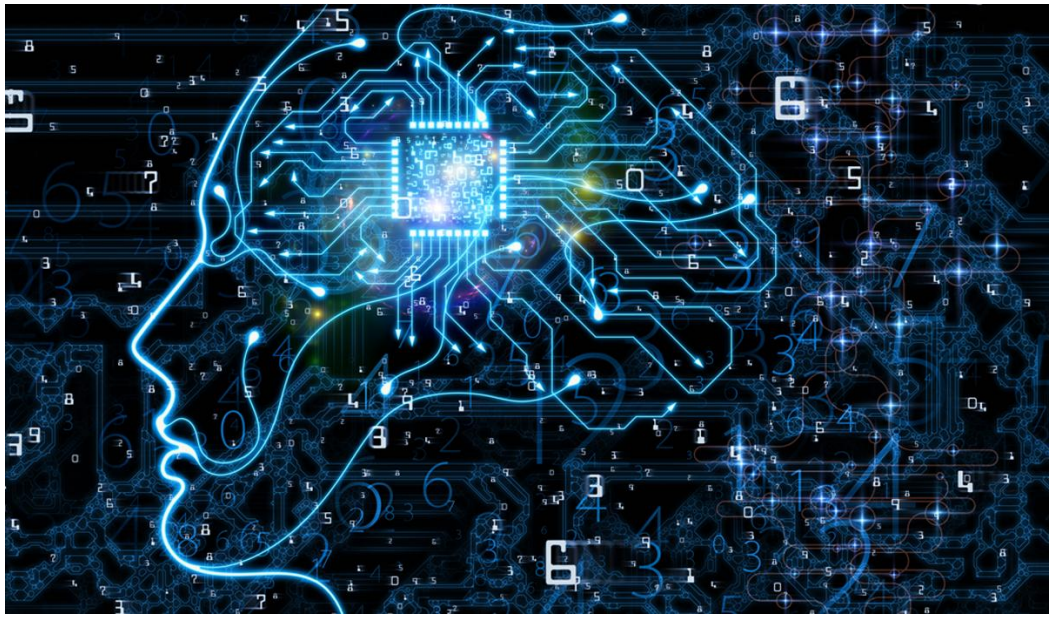


Neuroevolution

- Evolving neurons: not really necessary but attempted
- Evolving weights instead of backpropagation and gradient descent
- Evolving the architecture of neural network
  - For small nets, one uses a simple matrix representing which neuron connects which.
  - This matrix is, in turn, converted into the necessary 'genes', and various combinations of these are evolved.
- We shall review this after learning about neural networks

University of Ljubljana, Faculty of Computer and Information Science

Statistical Predictive Modeling



Prof Dr Marko Robnik-Šikonja

Intelligent Systems
Edition 2025

Learning

- **Learning** is the act of acquiring new, or modifying and reinforcing existing, knowledge, behaviors, skills, values, or preferences and may involve synthesizing different types of information.

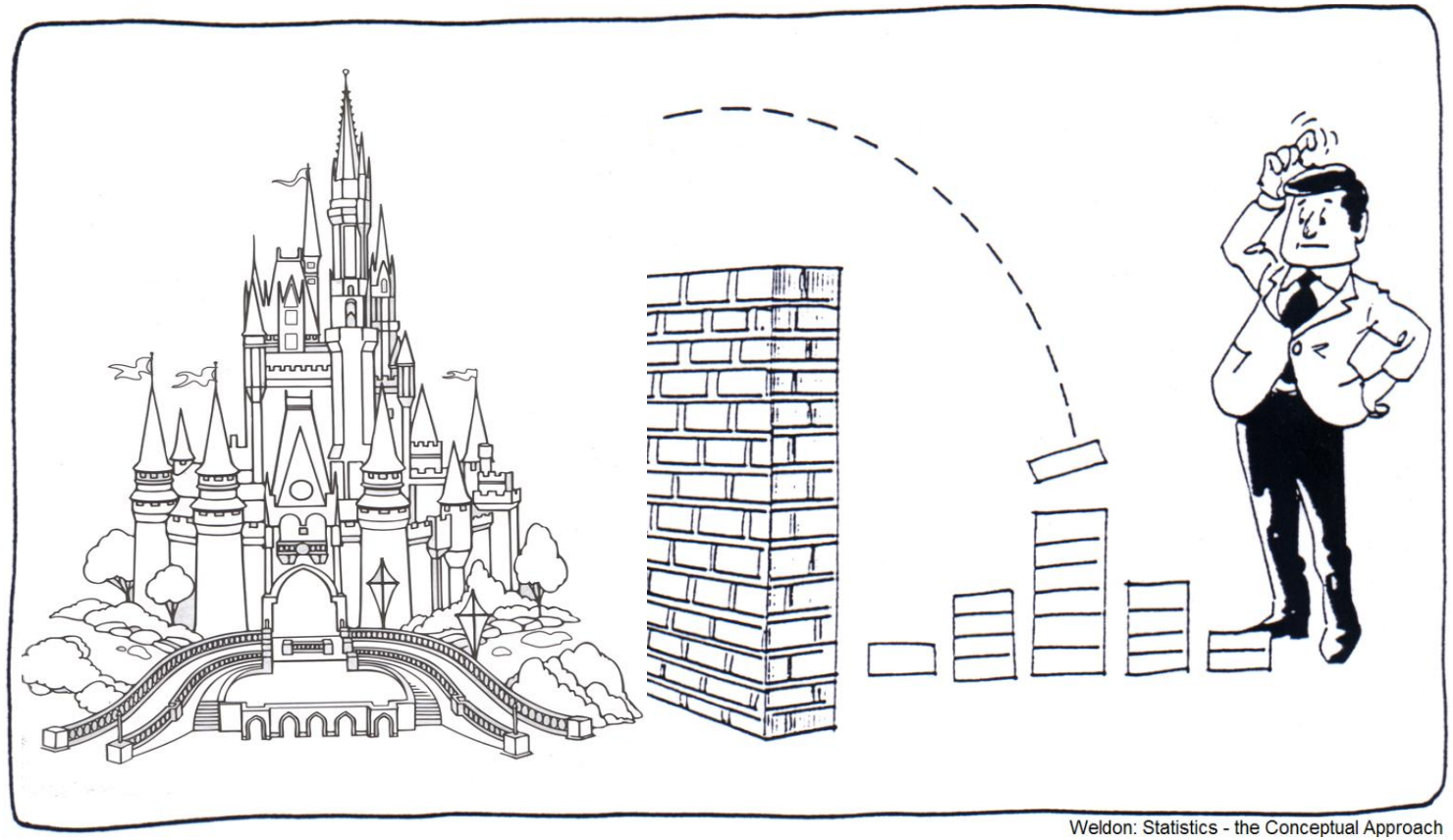
- **Statistical learning** deals with the problem of finding a predictive function based on data.

- The primary goals of statistical learning: prediction and understanding.

- Many different learning settings and data types, rapidly spreading in many areas of science, technology, and analytics, e.g., the Nobel prize in physics in 2024 to Hopfield and Hinton

Statistics and machine learning

- Definition from Wikipedia:
  ML algorithms operate by building a model from example inputs *i.e., samples.*



- ML can also be viewed as compression

The Data



Provided by the Institute of Oncology, Ljubljana



Post-surgery data for about 1000 breast cancer patients.

+

Recurrence and time of recurrence.

The Data

	class1	class2	menop	stage	grade	hType	PgR	inv	nLymph	cTh	hTh	famHist	LVI	ER	maxNode	posRatio	age
300	11.82	0	1	2	2	1	0	0	1	1	0	3	0	1	2	3	2
301	4.89	1	0	1	2	1	0	0	2	1	0	0	0	2	1	4	3
302	14.63	0	1	1	4	2	0	0	0	0	0	1	0	1	1	1	3
303	21.83	0	0	1	4	2	1	0	1	0	0	9	0	4	1	2	2
304	19.87	0	0	1	2	1	0	0	0	0	0	0	0	1	2	1	2
305	7.54	0	1	2	3	1	9	2	1	0	1	1	0	3	3	3	4
306	15.15	0	0	1	4	2	1	0	0	0	0	2	0	4	1	1	2
307	0.30	1	0	2	2	1	0	0	3	0	0	9	0	1	1	4	2
308	12.49	0	1	2	2	3	1	0	0	0	0	0	0	4	1	1	5
309	1.77	1	0	2	3	1	1	2	2	1	0	9	1	3	3	3	2

Each patient is described with 17 values:

- 15 patient's features
- 2 values, which describe the outcome

1 instance = 1 patient

	class1	class2	menop	stage	grade	hType	PgR	inv	nLymph	cTh	hTh	famHist	LVI	ER	maxNode	posRatio	age
300	11.82	0	1	2	2	1	0	0	1	1	0	3	0	1	2	3	2
301	4.89	1	0	1	2	1	0	0	2	1	0	0	0	2	1	4	3
302	14.63	0	1	1	4	2	0	0	0	0	0	1	0	1	1	1	3
303	21.83	0	0	1	4	2	1	0	1	0	0	9	0	4	1	2	2
304	19.87	0	0	1	2	1	0	0	0	0	0	0	0	1	2	1	2
305	7.54	0	1	2	3	1	9	2	1	0	1	1	0	3	3	3	4
306	15.15	0	0	1	4	2	1	0	0	0	0	2	0	4	1	1	2
307	0.30	1	0	2	2	1	0	0	3	0	0	9	0	1	1	4	2
308	12.49	0	1	2	2	3	1	0	0	0	0	0	0	4	1	1	5
309	1.77	1	0	2	3	1	1	2	2	1	0	9	1	3	3	3	2

- Menopause?
- Tumor stage
- Tumor grade
- Histological type
- Progesterone receptor lvl.
- Invasive tumor type
- Number of positive lymph nodes



- Hormonal therapy?
- Chemotherapy?
- Family medical history
- Lymphovascular invasion?
- Estrogen receptor lvl.
- Size of max. removed node
- Ratio of positive lymph nodes
- Age group

Prognostic Features

	class1	class2	menop	stage	grade	hType	PgR	inv	nLymph	cTh	hTh	famHist	LVI	ER	maxNode	posRatio	age
300	11.82	0	1	2	2	1	0	0	1	1	0	3	0	1	2	3	2
301	4.89	1	0	1	2	1	0	0	2	1	0	0	0	2	1	4	3
302	14.63	0	1	1	4	2	0	0	0	0	0	1	0	1	1	1	3
303	21.83	0	0	1	4	2	1	0	1	0	0	9	0	4	1	2	2
304	19.87	0	0	1	2	1	0	0	0	0	0	0	0	1	2	1	2
305	7.54	0	1	2	3	1	9	2	1	0	1	1	0	3	3	3	4
306	15.15	0	0	1	4	2	1	0	0	0	0	2	0	4	1	1	2
307	0.30	1	0	2	2	1	0	0	3	0	0	9	0	1	1	4	2
308	12.49	0	1	2	2	3	1	0	0	0	0	0	0	4	1	1	5
309	1.77	1	0	2	3	1	1	2	2	1	0	9	1	3	3	3	2

- Menopause?
- Tumor stage
- Tumor grade
- Histological type
- Progesterone receptor lvl.
- Invasive tumor type
- Number of positive lymph nodes

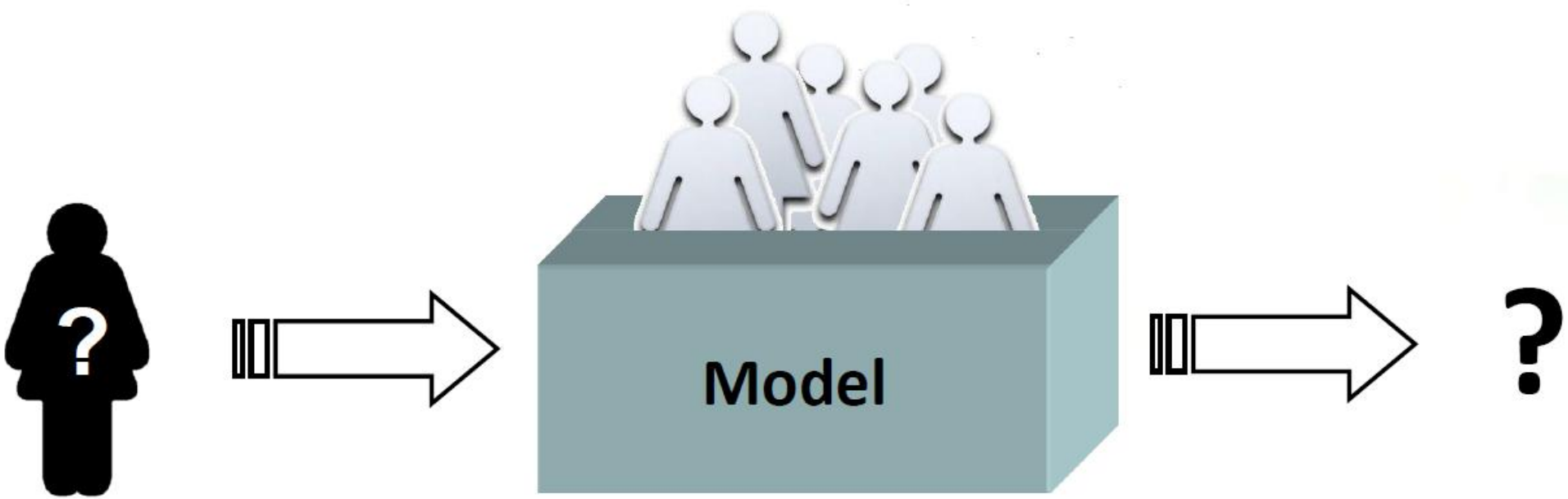


- Hormonal therapy?
- Chemotherapy?
- Family medical history
- Lymphovascular invasion?
- Estrogen receptor lvl.
- Size of max. removed node
- Ratio of positive lymph nodes
- Age group

Oncologists use these attributes for prognosis in every-day medical practice.

Basic Task in ML

We want to learn from past examples,with known outcomes.



To predict the outcome for a new patient.

Basic notation of predictive modelling

- Cancer recurrence is a statistical variable named response or target or prediction variable that we wish to predict. We usually refer to the response as Y .
- Other input variables are called attributes, features, inputs, or predictors; we name them $X_{.j}$
- One observation, called also an instance or example is denoted as $X_{i,.}$

- The input vectors form a matrix **X**

$$\mathbf{X} = \begin{pmatrix} x_{11} & x_{12} & \cdots & x_{1p} \\ x_{21} & x_{22} & \cdots & x_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n1} & x_{n2} & \cdots & x_{np} \end{pmatrix}$$

- The model we write as

$$Y = f(X) + \epsilon$$

where ∈ is independent of X, has zero mean and represents measurement errors and other discrepancies.

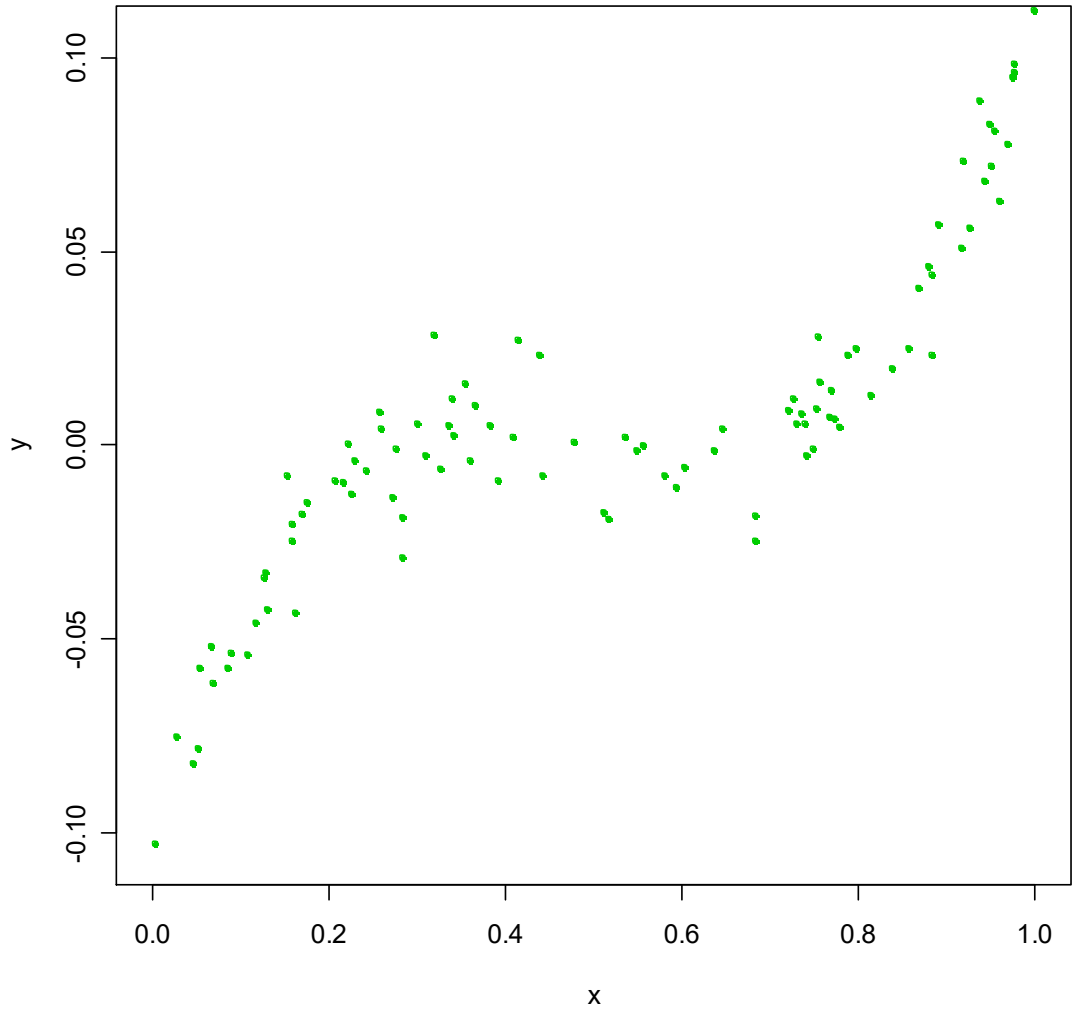
Further notation for instances and attributes

- Suppose we observe Y_i and $X_i = (x_{i,1}, x_{i,2}, \ldots x_{i,p})$for $i = 1,2, \ldots n$
- We believe that there is a relationship between Y and X.
- We can model the relationship as $$Y_i = f(\mathbf{X}_{i.}) + \varepsilon_i$$

- Where *f* is an unknown function and ε is a random error with mean zero.
- Take care, the notation may be confusing, we also use

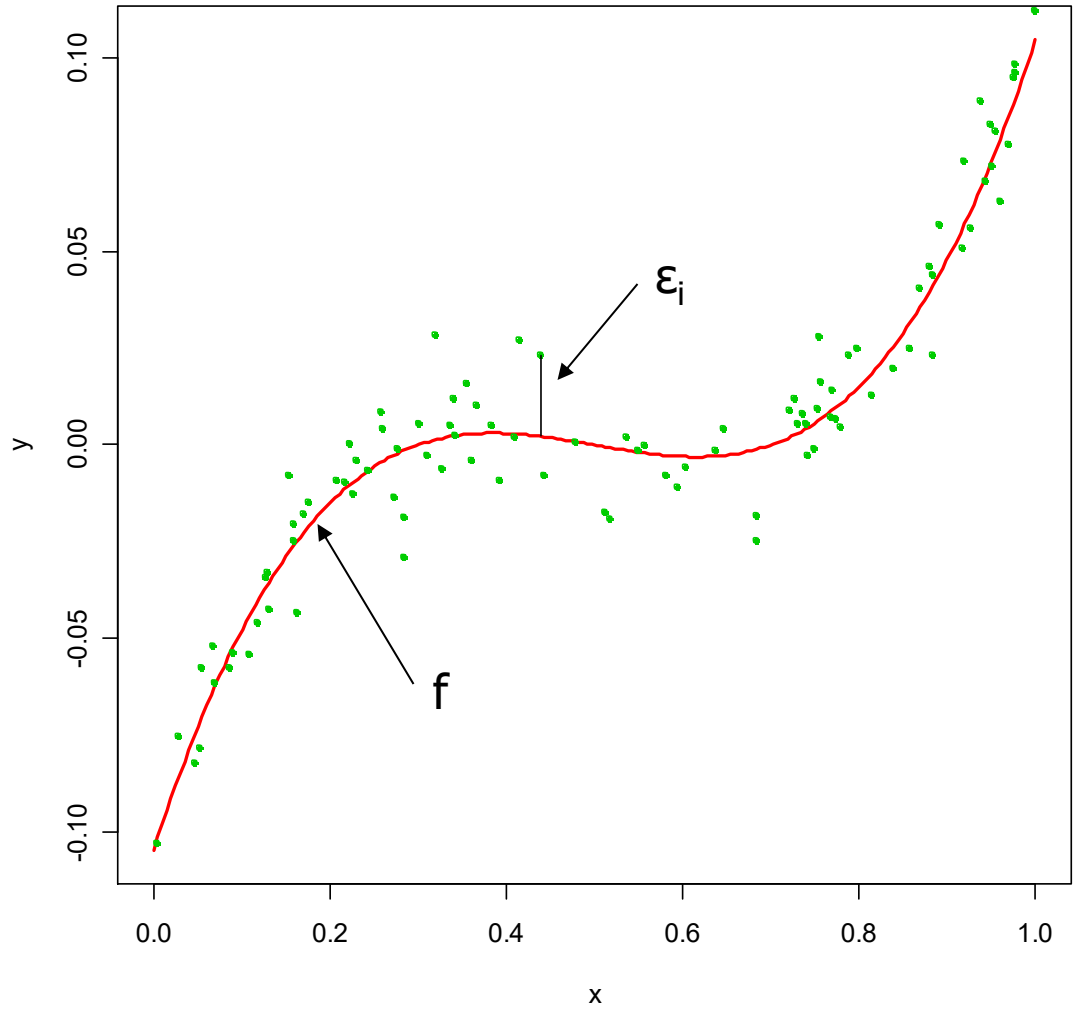
$$\mathbf{x}_j = \begin{pmatrix} x_{1j} \\ x_{2j} \\ \vdots \\ x_{nj} \end{pmatrix}$$

A simple example



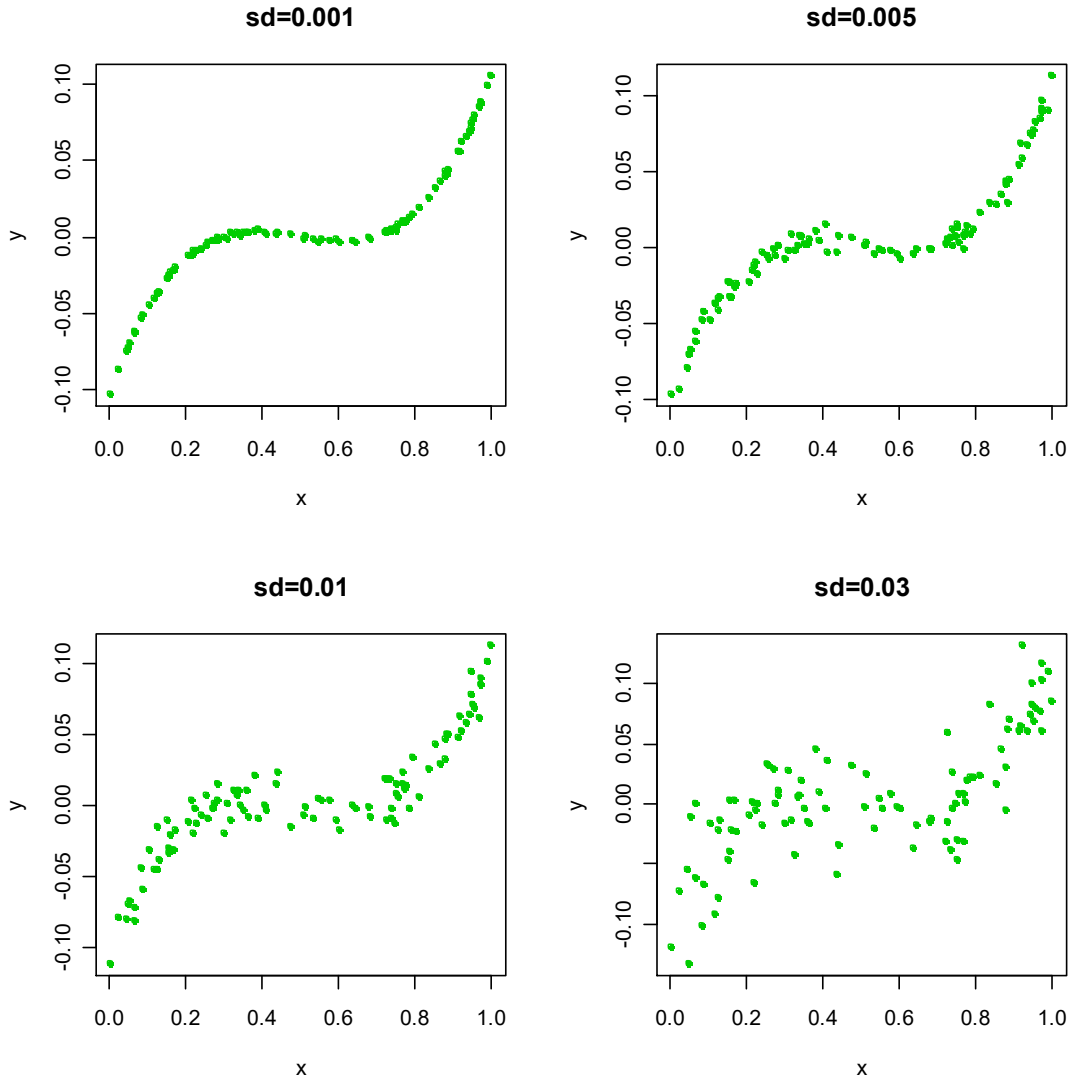
A simple example

- Assuming we know f (in red)

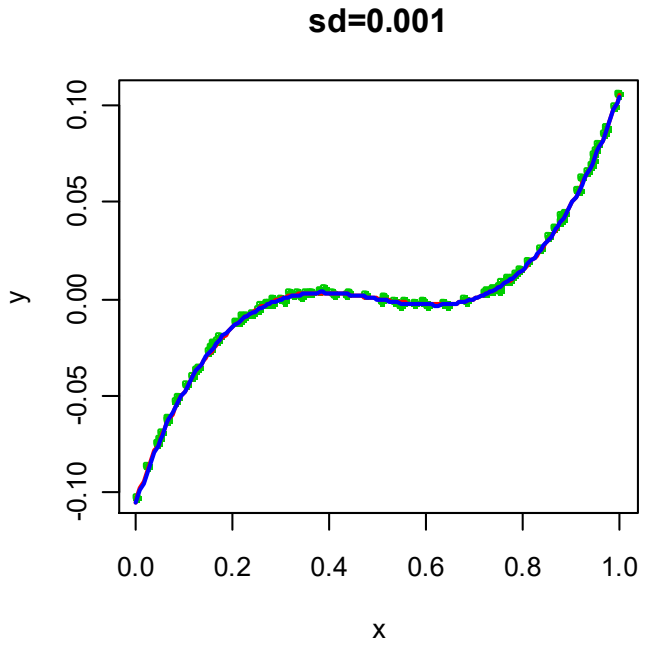


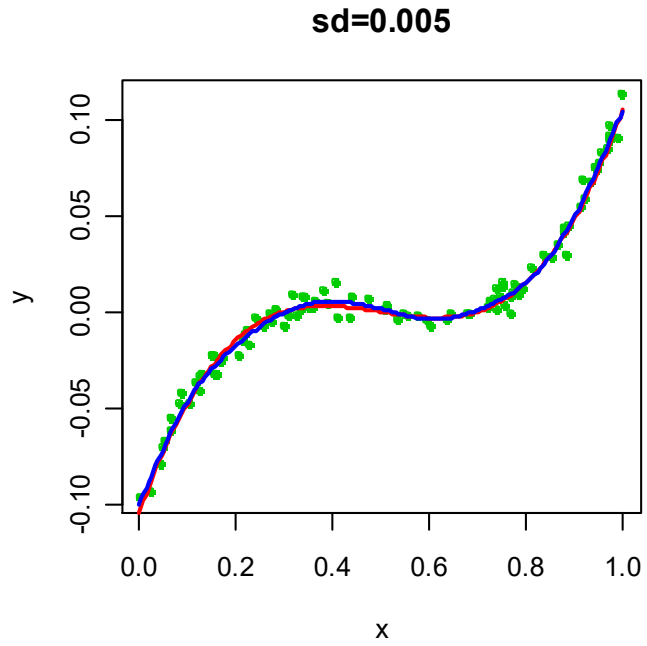
Different standard deviations of error

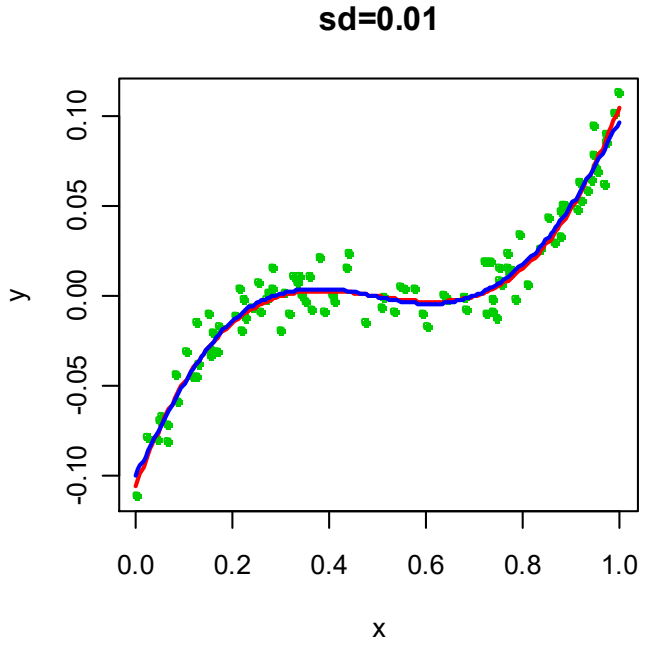
- The difficulty of estimating *f* will depend on the standard deviation of the ε's.

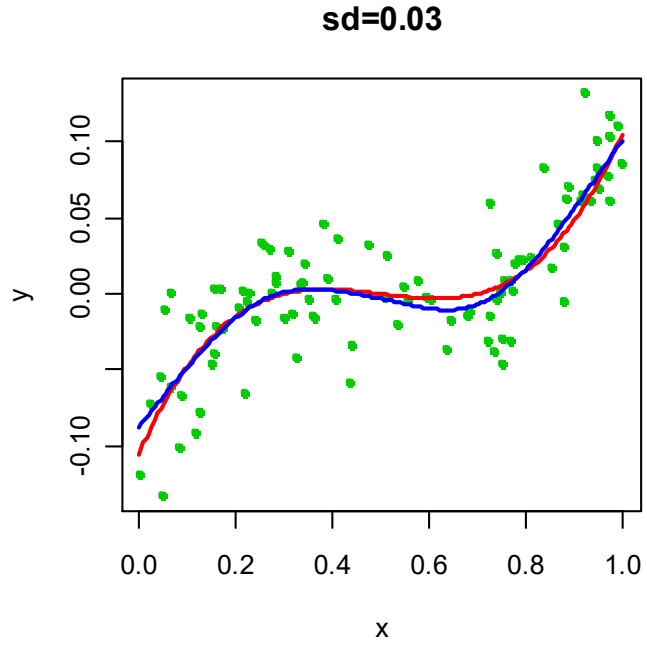


Different estimates for *f*



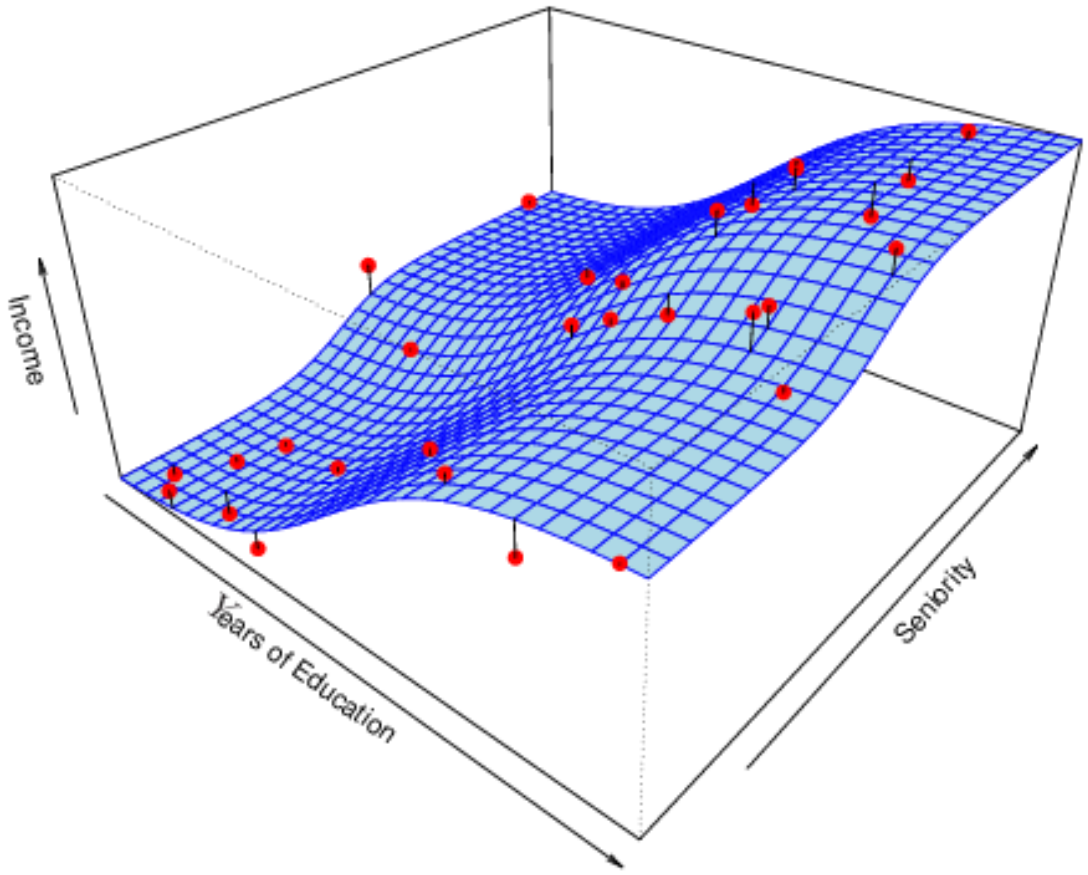






Income vs. Education and Seniority

Multidimensional **X**



1st goal of learning: prediction

- If we can produce a good estimate for f (and the variance of ε is not too large), we can make accurate predictions for the response, Y_i, based on a new value of $X_{i.}$

- Example: Direct Mailing Prediction
  - Interested in predicting how much money an individual will donate based on observations from 90,000 people on which we have recorded over 400 different characteristics.
  - Don't care too much about each individual characteristic.
  - Just want to know: For a given individual should I send out a mailing?

2nd goal of learning: inference

- often we are interested in the type of relationship between Y and all the $X_{.j}$
- For example,
  - Which particular predictors actually affect the response?
  - Is the relationship positive or negative?
  - Is the relationship a simple linear one or is it more complicated, etc.?
  - For a given $(X_{i.}, Y_i)$, which feature values x_{ij} are the most important to determine y_i?
- Sometimes more important than prediction, e.g., in medicine.
- Example: Housing Inference
  - Wish to predict median house price based on 14 variables.
  - Probably want to understand which factors have the biggest effect on the response and how big the effect is.
  - For example, how much impact does a river view have on the house value etc.

How do we estimate *f*?

- We will assume we have observed a set of **training data**

$$\{(\mathbf{X}_1, Y_1), (\mathbf{X}_2, Y_2), \ldots, (\mathbf{X}_n, Y_n)\}$$

- We must then use the training data and a statistical method to estimate f.
- Statistical Learning Methods:
  - Parametric Methods
  - Non-parametric Methods

Parametric methods

- They reduce the problem of estimating *f* down to one of estimating a set of parameters.
- They involve a two-step model-based approach

STEP 1:

Make some assumption about the functional form of *f*, i.e. come up with a model. The most common example is a linear model, i.e.

$$f(\mathbf{X}_i) = \beta_0 + \beta_1 X_{i1} + \beta_2 X_{i2} + \cdots + \beta_p X_{ip}$$

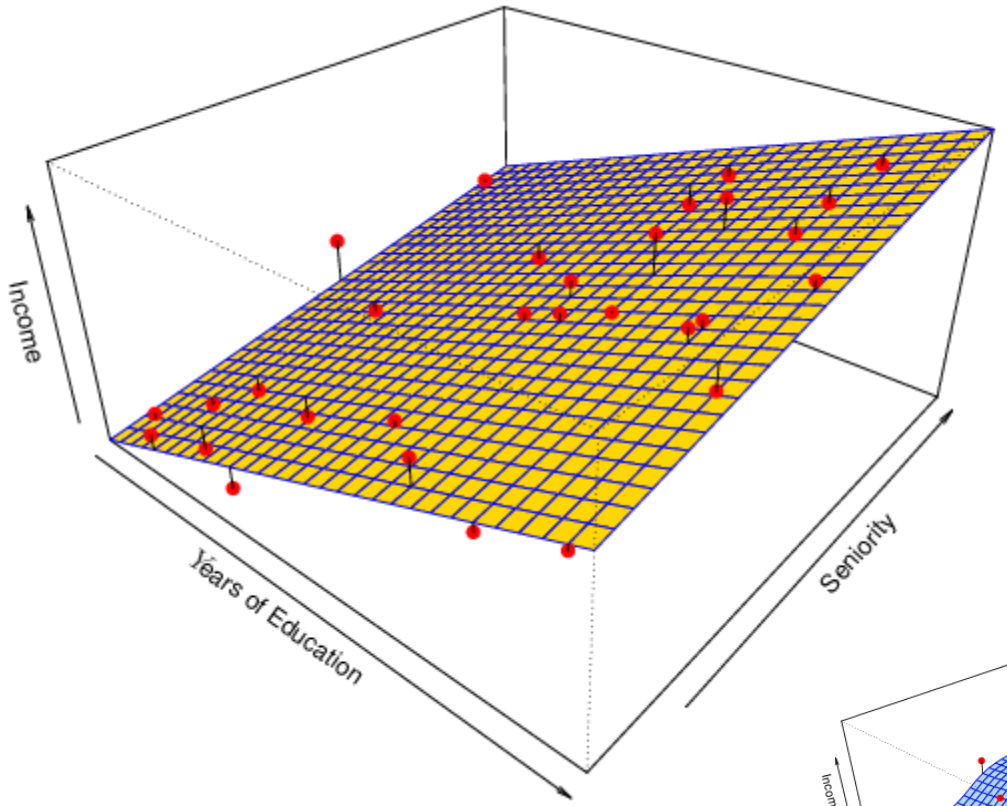
More complicated and flexible models for *f* are often more realistic.

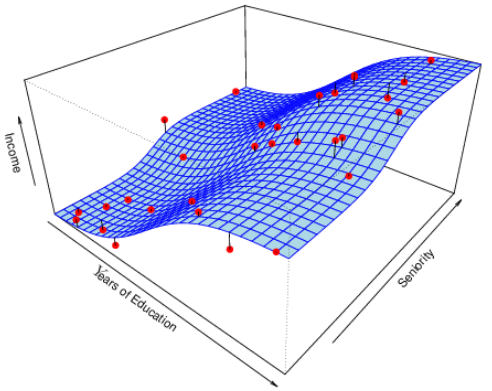
STEP 2:

Use the training data to fit the model, i.e. estimate *f* or equivalently the unknown parameters such as β_0, β_1, β_2,…β_p
For linear model the most common method uses ordinary least squares (OLS).

Example: a linear regression estimate

- Even if the standard deviation is low, we will still get a bad answer if we use the wrong model.





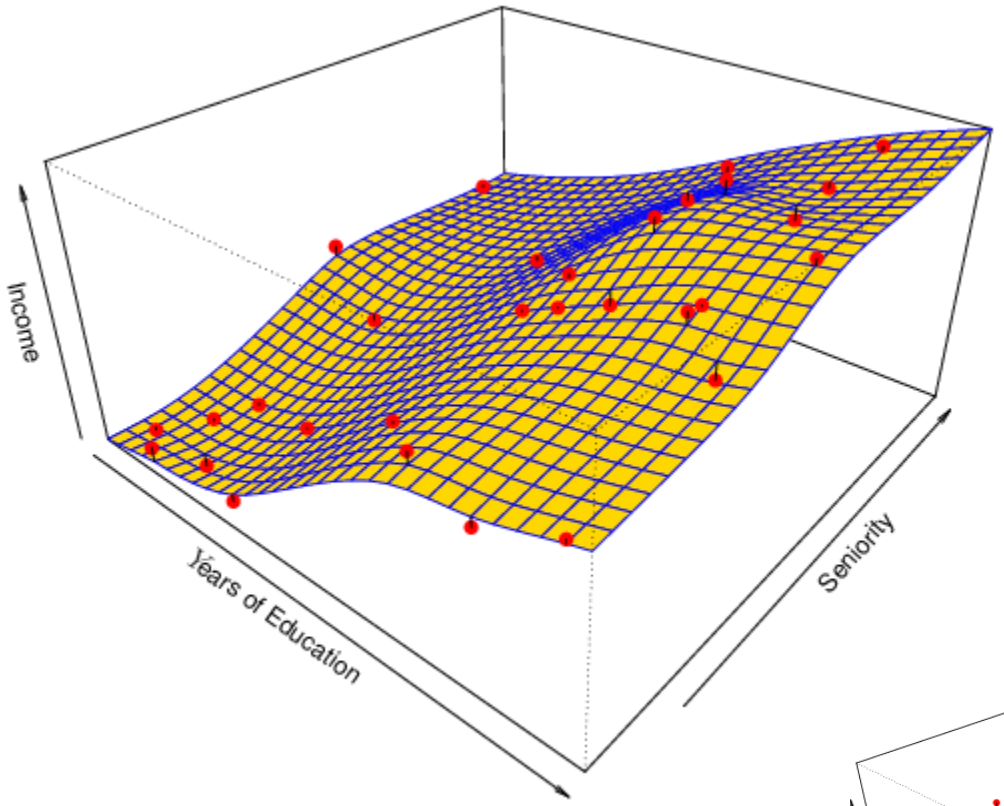
$$f = \beta_0 + \beta_1 \times Education + \beta_2 \times Seniority$$

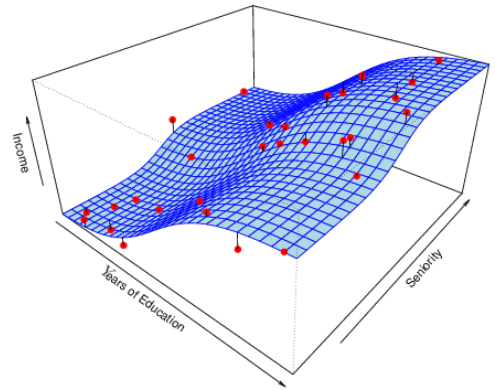
Non-parametric methods

- They do not make explicit assumptions about the functional form of f.
- Advantage: They accurately fit a wider range of possible shapes of f.
- Disadvantage: A large number of observations may be required to obtain an accurate estimate of f.

Example: a thin-plate spline estimate

- Non-linear regression methods are more flexible and can potentially provide more accurate estimates.





Trade-off between prediction accuracy and model interpretability

- Why not just use a more flexible method if it is more realistic?

Reason 1:

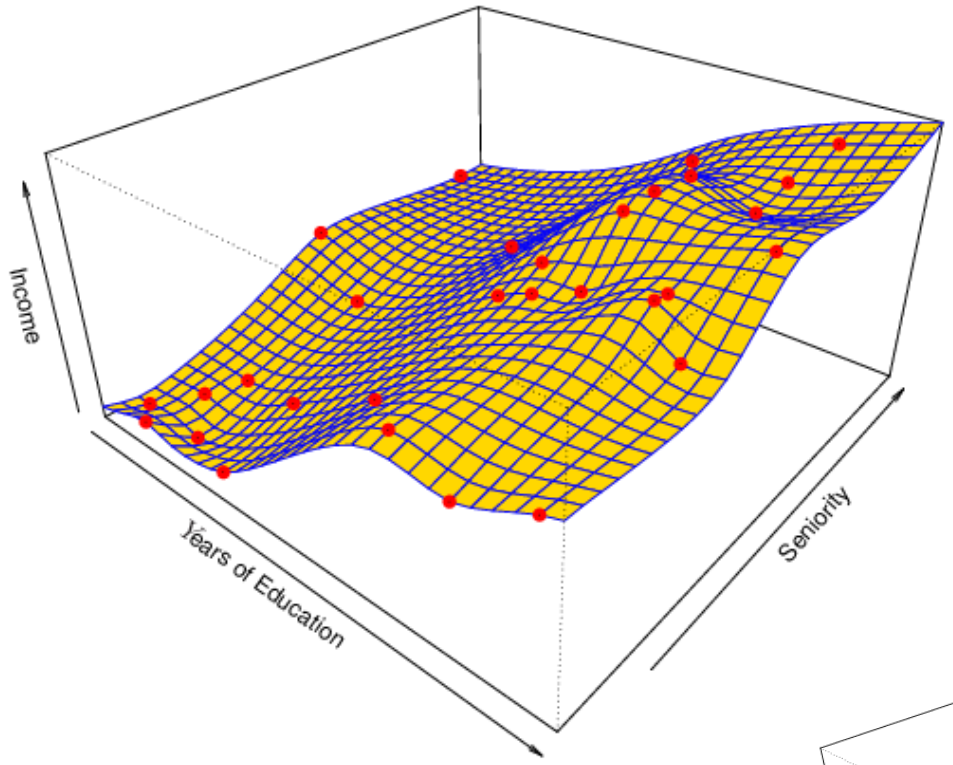
A simple method such as linear regression produces a model which is much easier to interpret (the inference part is better). For example, in a linear model, β_j is the average increase in Y for a one unit increase in X_j holding all other variables constant.

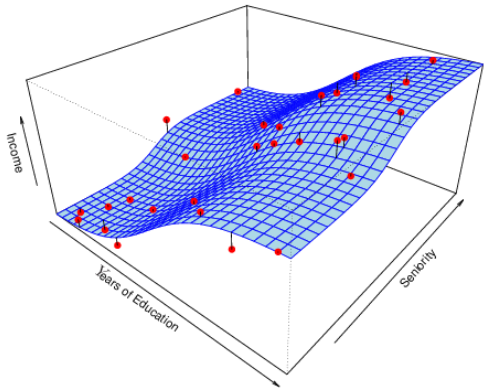
Reason 2:

Even if you are only interested in prediction, so the first reason is not relevant, it is often possible to get more accurate predictions with a simple, instead of a complicated, model. This seems counter intuitive but has to do with the fact that it is harder to fit a more flexible model.

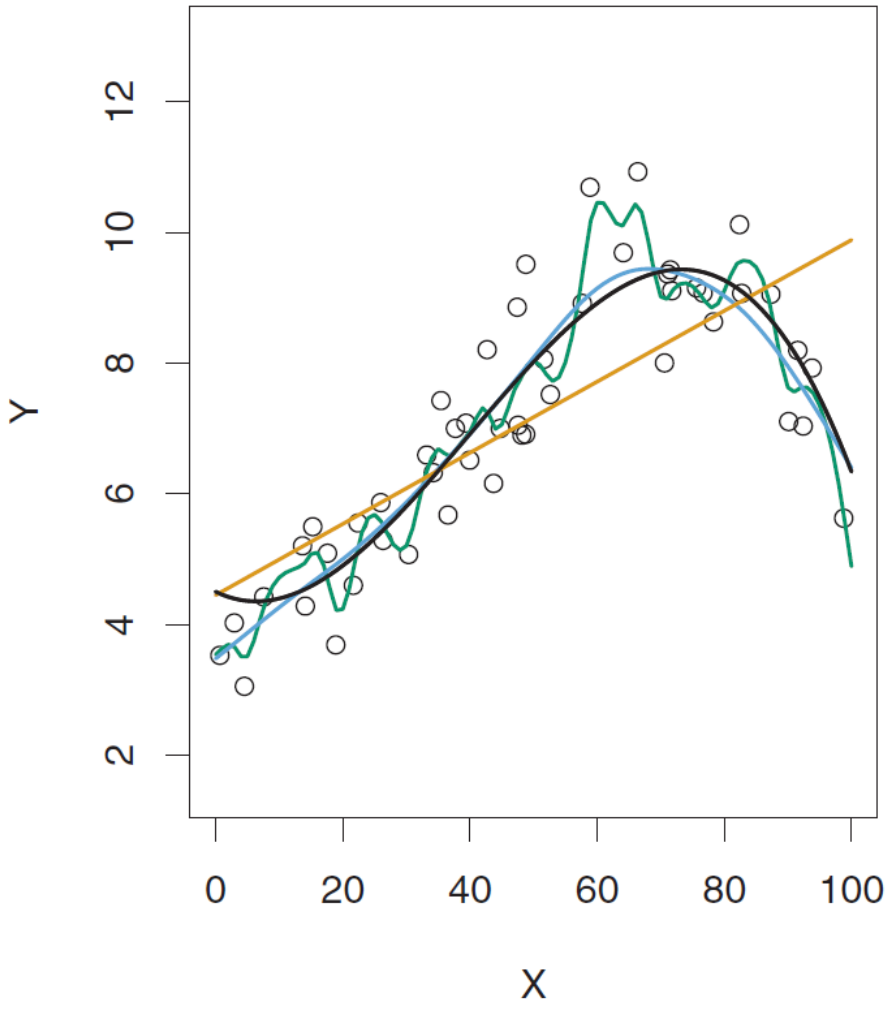
A poor estimate: overfitting

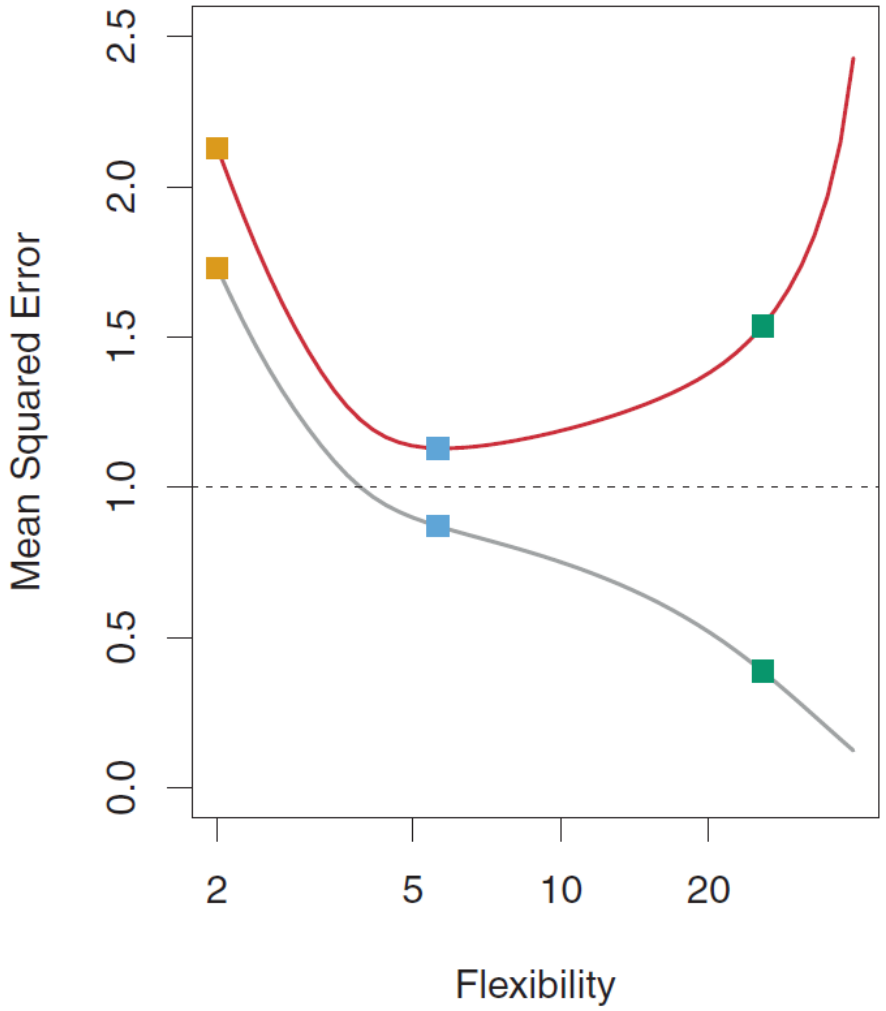
- Non-linear regression methods can also be too flexible and produce poor estimates for *f*.





Goodness of fit for three models





LEFT

Black: Truth

Orange: Linear Estimate

Blue: smoothing spline

Green: smoothing spline (more flexible)

RIGHT

RED: Test MSE

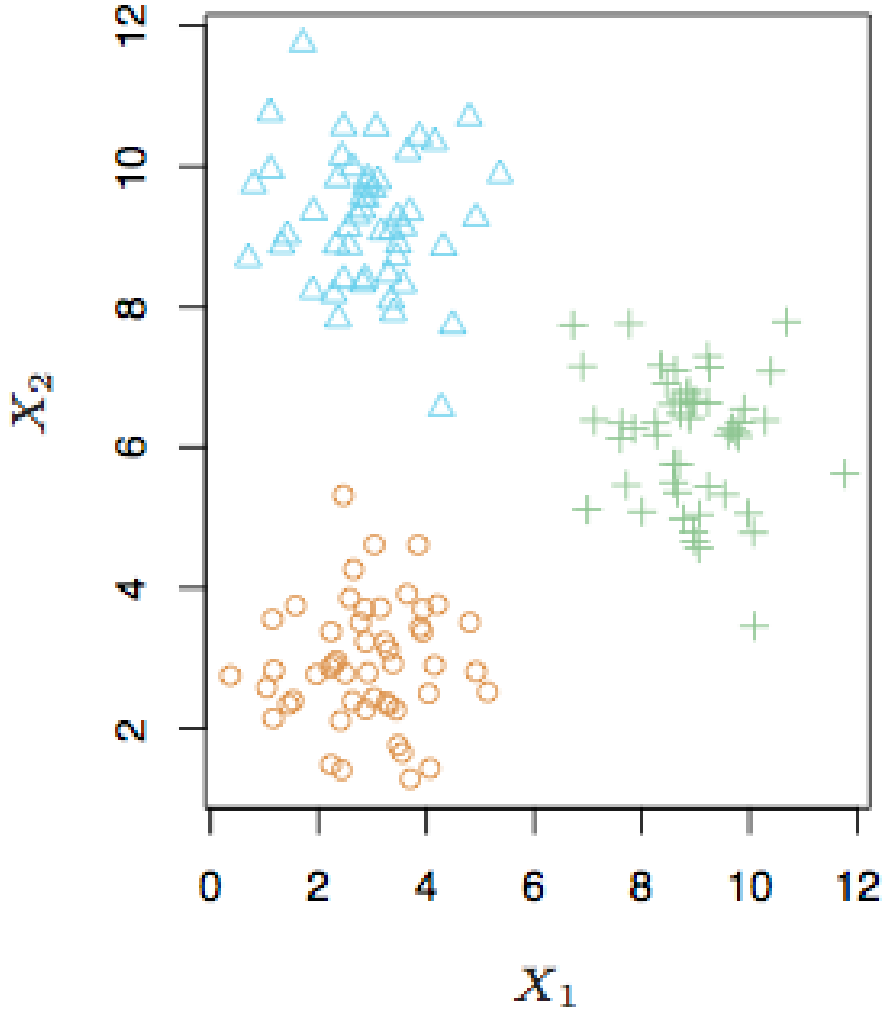
Grey: Training MSE

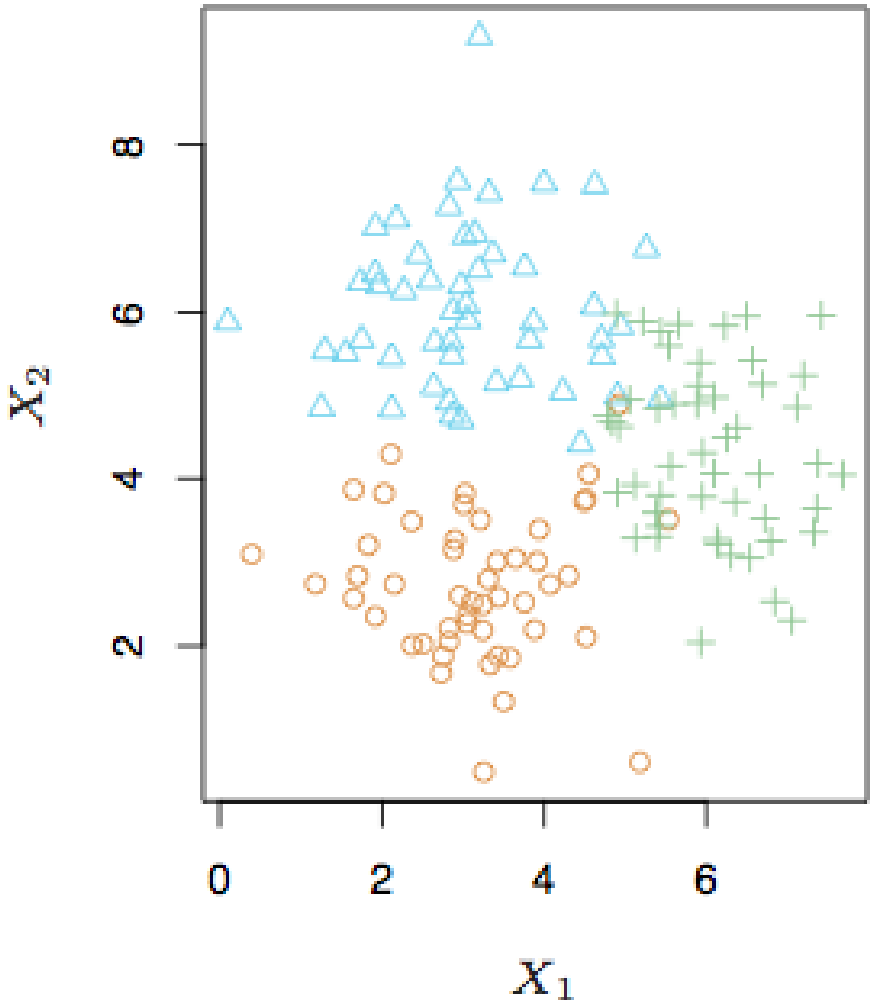
Dashed: Minimum possible test MSE (irreducible error)

Supervised, unsupervised, semi-supervised, self-supervised, weakly-supervised learning 1/2

- We can divide learning problems into Supervised and Unsupervised situations
- <u>Supervised learning:</u>
  - Supervised Learning is where both the predictors, $\mathbf{X}_i$, and the response, Y_i, are observed.
  - e.g., linear regression
- <u>Unsupervised learning:</u>
  - In this situation only the $\mathbf{X}_i$'s are observed.
  - We need to use the $\mathbf{X}_i$'s to guess what Y would have been and build a model from there.
  - A common example is market segmentation where we try to divide potential customers into groups based on their characteristics.
  - A common approach is clustering.
    - Idea: Maximizing initra-cluster similarity & minimizing inter-cluster similarity
- <u>Semi-supervised learning</u>
  - only a small sample of labelled instances are observed but a large set of unlabeled instances
  - an initial supervised model is used to label unlabeled instances
  - the most reliable predictions are added to the training set for the next iteration of supervised learning

A simple clustering example





Supervised, unsupervised, semi-supervised, self-supervised, weakly-supervised learning 2/2

- Self-supervised learning
  - a mixture of supervised and unsupervised learning
  - learns from unlabeled data
  - the labels are obtained from related properties of the data itself, often leveraging the underlying structure in the data
  - usually predicts any unobserved or hidden part (or property) of the input from any observed or unhidden part of the input.
  - e.g., in NLP, we can hide part of a sentence and predict the hidden words from the remaining words
  - e.g., in video processing, we can predict past or future frames in a video (hidden data) from current ones (observed data)
- Weakly-supervised data
  - noisy, limited, or imprecise sources are used to provide supervision signal for labeling large amounts of training data to do supervised learning
  - reduces the burden of obtaining hand-labeled data sets, which can be costly or impractical
  - e.g., using a smart electricity meter to estimate household occupancy

Regression vs. classification

- Supervised learning problems can be further divided into
- Regression problems: Y is continuous/numerical. e.g.
    - Predicting the value of certain share on stock market
    - Predicting the value of a given house based on various inputs
    - The duration in years till cancer recurrence
- Classification problems: Y is categorical, e.g.,
    - Will the price of a share go up (U) or down (D)?
    - Is this email a SPAM or not?
    - Will the cancer recur?
    - What will be an outcome of a football match (Home, Away, or Draw)?
    - Credit card fraud detection, direct marketing, classifying stars, diseases, web-pages, etc.
    - Note that we mostly predict probabilities of the categories
- Some methods work well on both types of problem, e.g., neural networks or kNN

Data mining (analytics, science): on what kinds of data?

- Database-oriented data sets and applications
  - Relational database, data warehouse, transactional database
- Advanced data sets and advanced applications
  - Data streams and sensor data
  - Time-series data, temporal data, sequence data (incl. bio-sequences)
  - Structure data, graphs, social networks and multi-linked data
  - Object-relational databases
  - Heterogeneous databases and legacy databases
  - Spatial data and spatiotemporal data
  - Multimedia database
  - Text databases
  - The World-Wide Web

Association and correlation analysis

- Frequent patterns (or frequent itemsets)
  - What items are frequently purchased together in the supermarket?
- Association, correlation vs. causality
  - A typical association rule
    - Diaper → Beer [0.5%, 75%] (support, confidence)
  - Are strongly associated items also strongly correlated?
- How to mine such patterns and rules efficiently in large datasets?
- How to use such patterns for classification, clustering, and other applications?

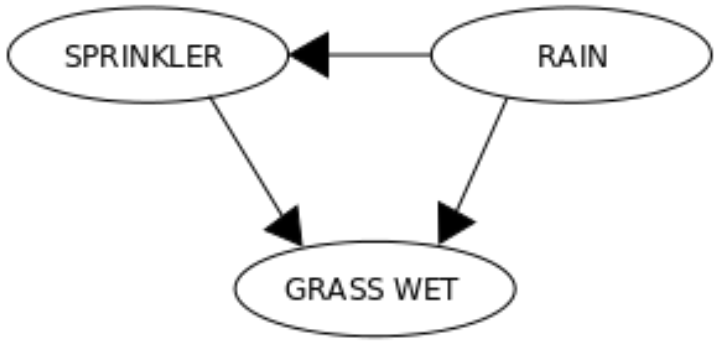
Outlier analysis

- Outlier: A data object that does not comply with the general behavior of the data
- Noise or exception? — One person's garbage could be another person's treasure
- Methods: byproduct of clustering or regression analysis, …
- Useful in fraud detection, rare events analysis

Relational learning

- Several variants:
  - Bayesian networks,
  - inductive logic programming
  - graph learning, e.g., link prediction

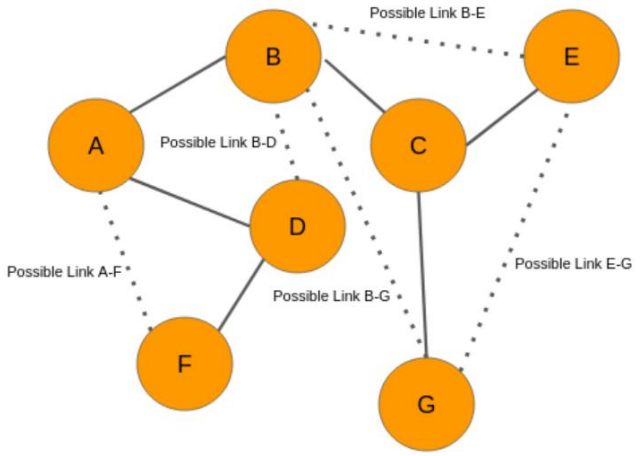
	SPRINKLER	
RAIN	T	F
F	0.4	0.6
T	0.01	0.99

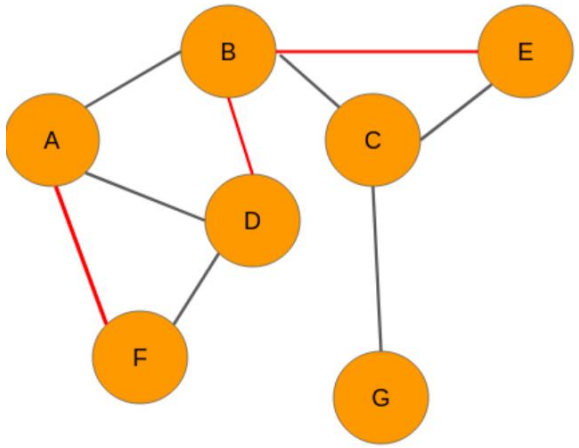


	RAIN	
T	F	
0.2	0.8	

		GRASS WET	
SPRINKLER	RAIN	T	F
F	F	0.0	1.0
F	T	0.8	0.2
T	F	0.9	0.1
T	T	0.99	0.01

Notation	Example rule
Explicit	IF $Shape = triangle \wedge Color = red \wedge Size = big$ THEN $Class = positive$
Formal	$Class = positive \leftarrow Shape = triangle \wedge Color = red \wedge Size = big$
Logical	`positive(X):-shape(X,triangle),color(X,red),size(X,big).`





Generalization as a search

- So far, we presented the “learning as an optimization” ML view
- Inductive learning: find a concept description that fits the data
- Example: rule sets as description language
  - Enormous but finite search space
- Simple solution:
  - enumerate the concept space
  - eliminate descriptions that do not fit examples
  - surviving descriptions contain target concept

Learning as optimization

- Usually the goal of classification is to minimize the test error
- Therefore, many learning algorithms solve optimization problems, e.g.,
  - linear regression minimizes squared error on the training set
  - AntMiner algorithms minimize the classification accuracy of decision rules on the training set using ACO
  - to find a good architecture of neural networks, GAs can be applied and minimize the prediction error on the validation set
  - most learning methods use optimization algorithms to minimize the implicitly or explicitly stated loss function, e.g., cross-entropy in neural network is minimized with gradient descent, where cross-entropy is a distance between two distributions, the predicted P and the true Q: $H(P, Q) = \sum_{x \in X} p(x) \log q(x)$

Criteria of success for ML

- No single best ML method (no free lunch theorem)
- How to select the best model?
  - measure the quality of fit, i.e. how well the predictions match the observed data
  - measure on previously unseen data (called test set). Why? Can we do it many times?
- In regression, the most popular measure is the mean squared error

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - f(x_i))^2$$

where y_i is the true (observed) value of instance i, and $f(x_i)$ is its predicted value

- in classification, the *classification accuracy = 1 – error rate* is the most popular criterion

$$CA = \frac{1}{n}\sum_{i=1}^{n} I(y_i = f(x_i))$$

where $f(x_i)$ is the predicted category

- We will say more about this topic later

No-Free-Lunch theorem



- In the "no free lunch" metaphor,
  each "restaurant" (problem-solving procedure)
  has a "menu" associating
  each "lunch plate" (problem)
  with a "price" (the performance of the procedure in solving the problem).
- The menus of restaurants are identical except in one regard – the prices are shuffled from one restaurant to the next.
- For an omnivore who is as likely to order each plate as any other, the average cost of lunch does not depend on the choice of restaurant.
- But a vegan who goes to lunch regularly with a carnivore who seeks economy might pay a high average cost for lunch.
- To methodically reduce the average cost, one must use advance knowledge of
  - a) what one will order and
  - b) what the order will cost at various restaurants.
- That is, improvement of performance in problem-solving hinges on using prior information to match procedures to problems.

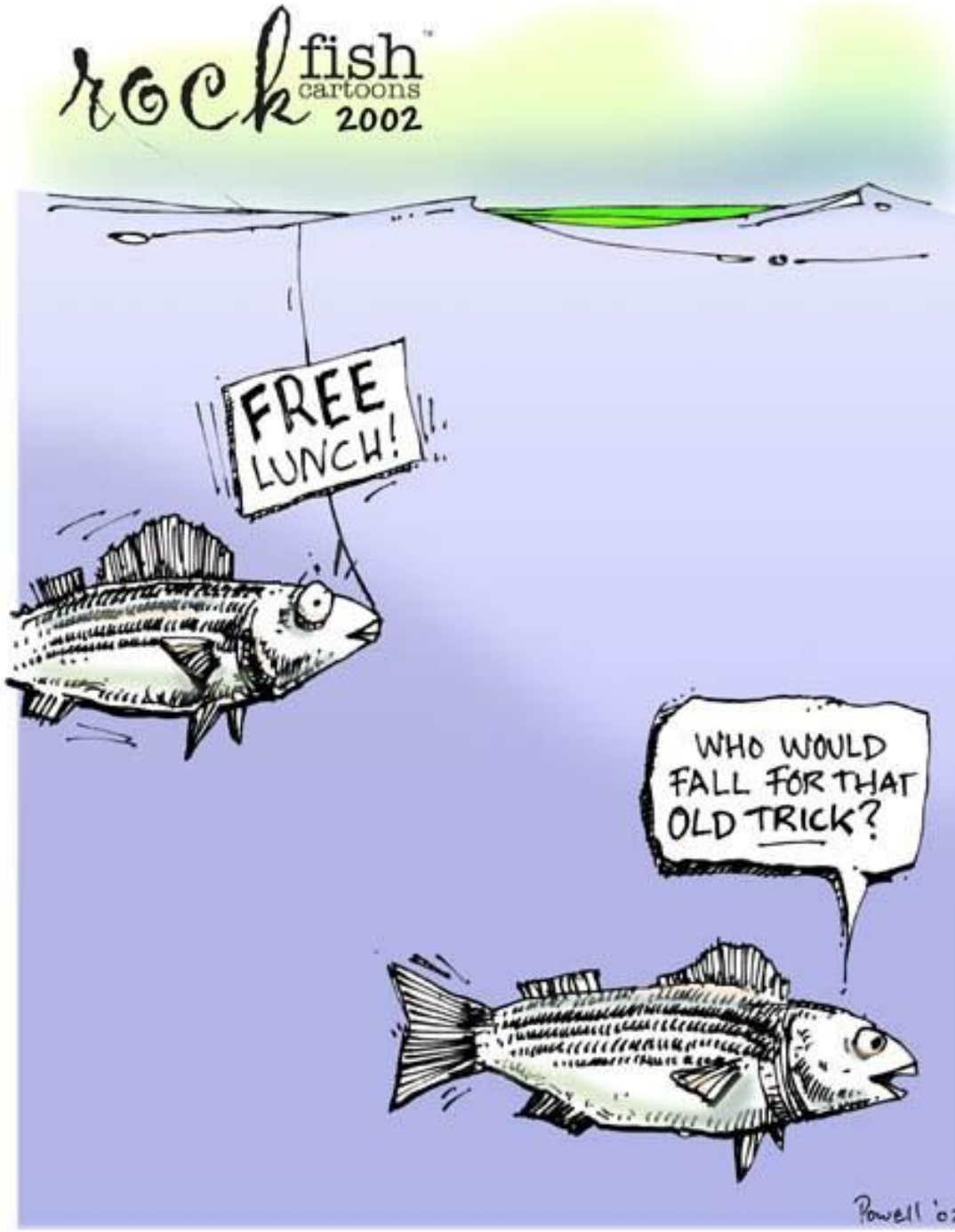
Consequences of the NFL theorem

If no information about the target function f(x) is provided:

- No classifier is better than some other in the general case.
- No classifier is better than random in the general case.

- ML practitioners possess implicit or explicit knowledge about the prices in different restaurants

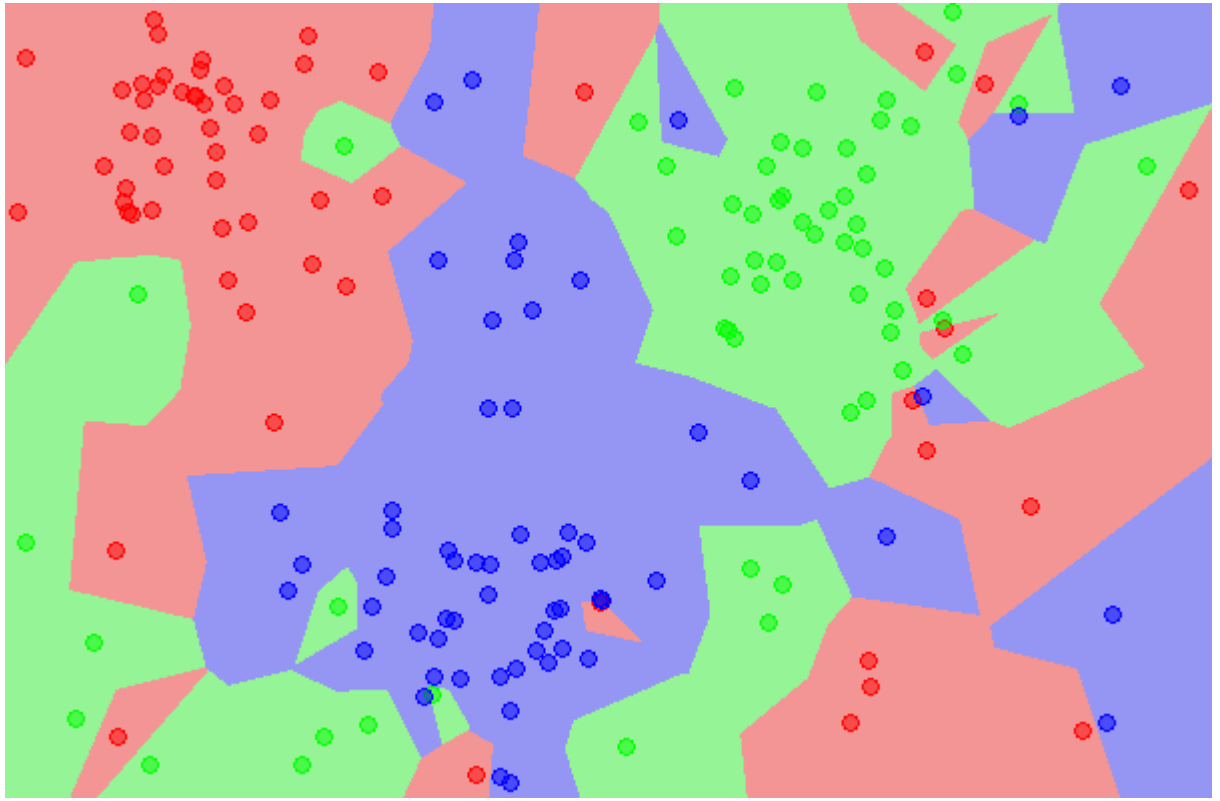
- Meta-learning
- Automatic ML (AutoML)



University of Ljubljana, Faculty of Computer and Information Science

Bias, variance and predictive models





Prof Dr Marko Robnik-Šikonja

Intelligent Systems, Edition 2024

Contents

- Bias and variance of prediction models
- Bayes optimal classifier
- Simple regression models:
  - linear models, nearest neighbor, regression trees, regression rules
- Simple classification models:
  - nearest neighbor, naïve Bayes, decision trees, decision rules, logistic regression
- Biases in data

A generalization problem

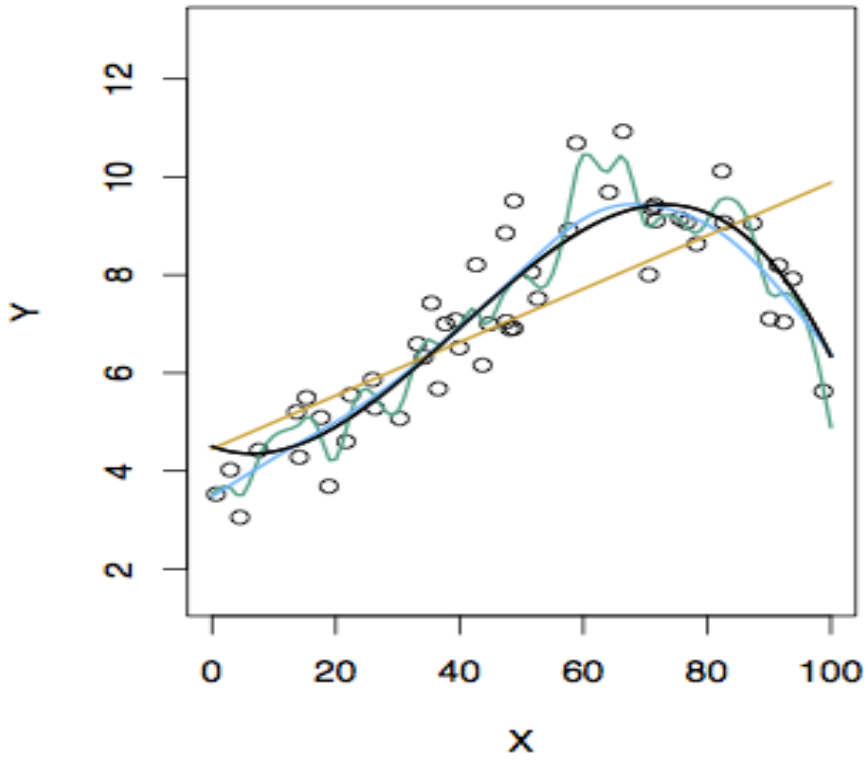
- Our ML methods have generally been designed to make error small on the training data, e.g., with linear regression, we choose the line such that MSE is minimized.
- What we really care about is how well the method generalizes i.e. how well it works on new data. We call this new data "**Test data**".
- There is no guarantee that the method with the smallest training error will have the smallest test (i.e. new data) error.
- One approach to address the problem is to reserve a portion of the training data to measure the generalization error, we this dataset "evaluation dataset". We use this dataset during training to guide the learning process, e.g., to stop it. This approach is used especially with overparametrized methods such as neural networks and in learning settings with high likelihood of overfitting such as AutoML methods.

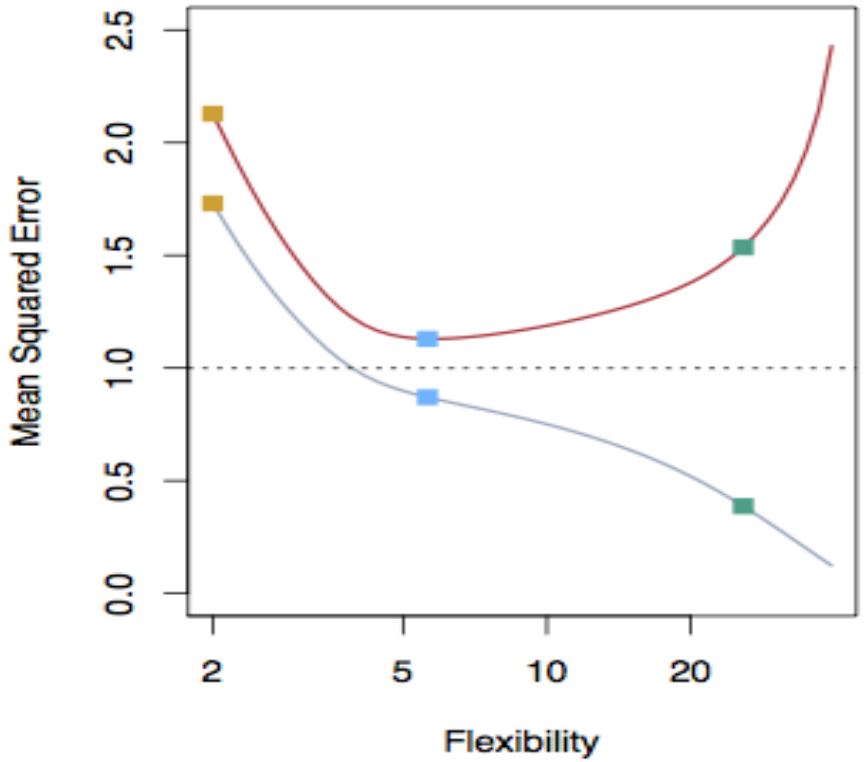
Training vs. test error

- In general the more flexible a method is the lower its training MSE will be, i.e. it will “fit” or explain the training data very well.
  - More flexible methods (such as splines) can generate a wider range of possible shapes to estimate *f* as compared to less flexible and more restrictive methods (such as linear regression). The less flexible the method, the easier to interpret the model. Thus, there is a trade-off between flexibility and model interpretability.

- However, the test MSE may in fact be higher for a more flexible method than for a simple approach like linear regression.

Different levels of flexibility: example 1





LEFT

Black: Truth

Orange: Linear Estimate

Blue: smoothing spline

Green: smoothing spline (more flexible)

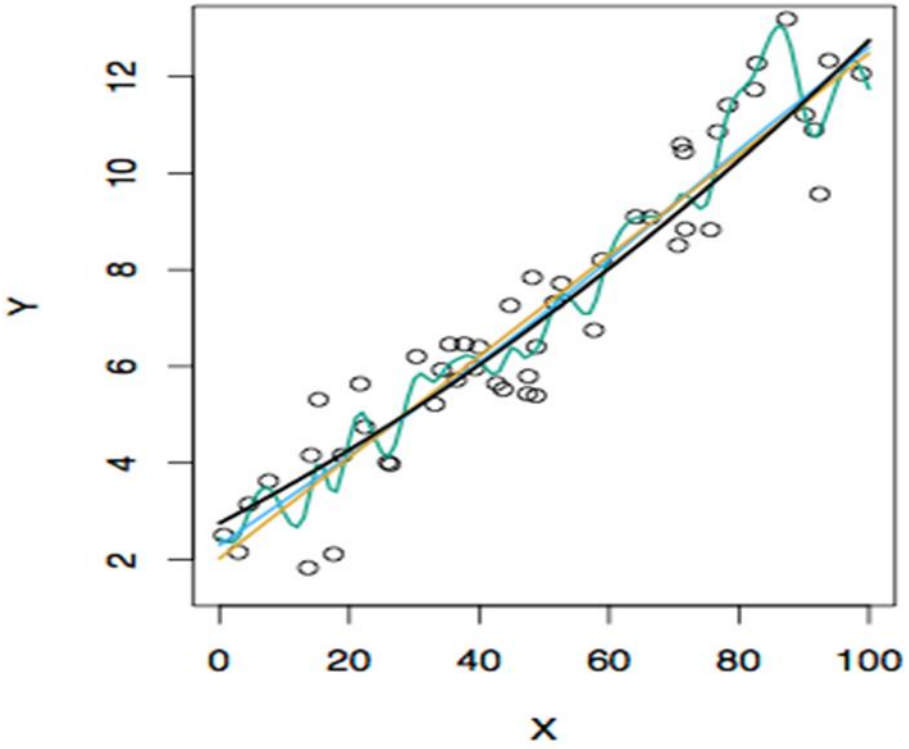
RIGHT

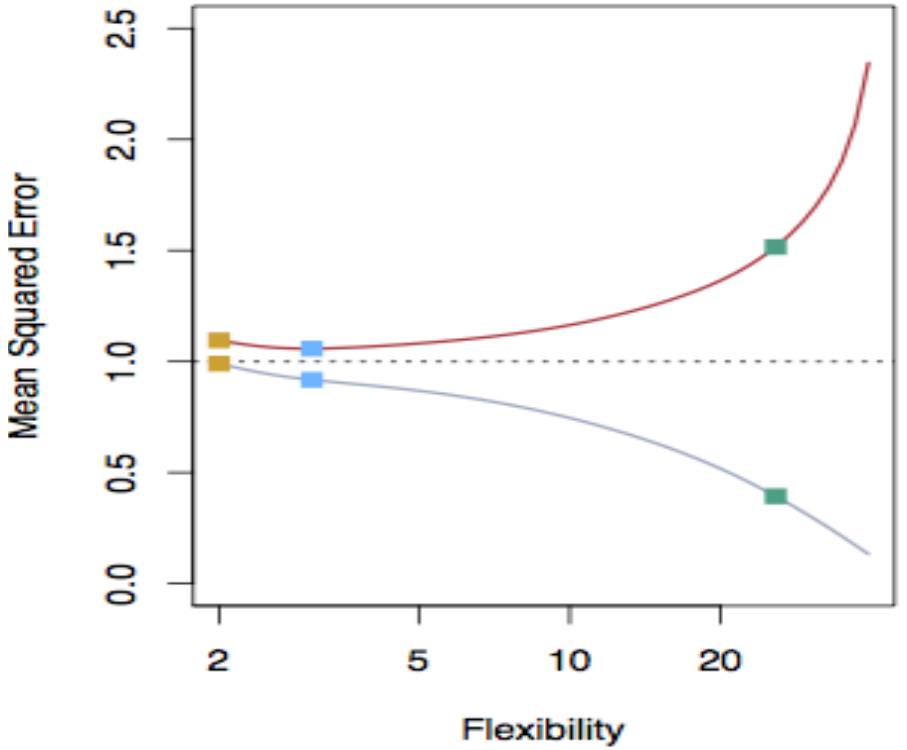
RED: Test MS

Grey: Training MSE

Dashed: Minimum possible test MSE (irreducible error)

Different levels of flexibility: example 2





LEFT

Black: Truth

Orange: Linear Estimate

Blue: smoothing spline

Green: smoothing spline (more flexible)

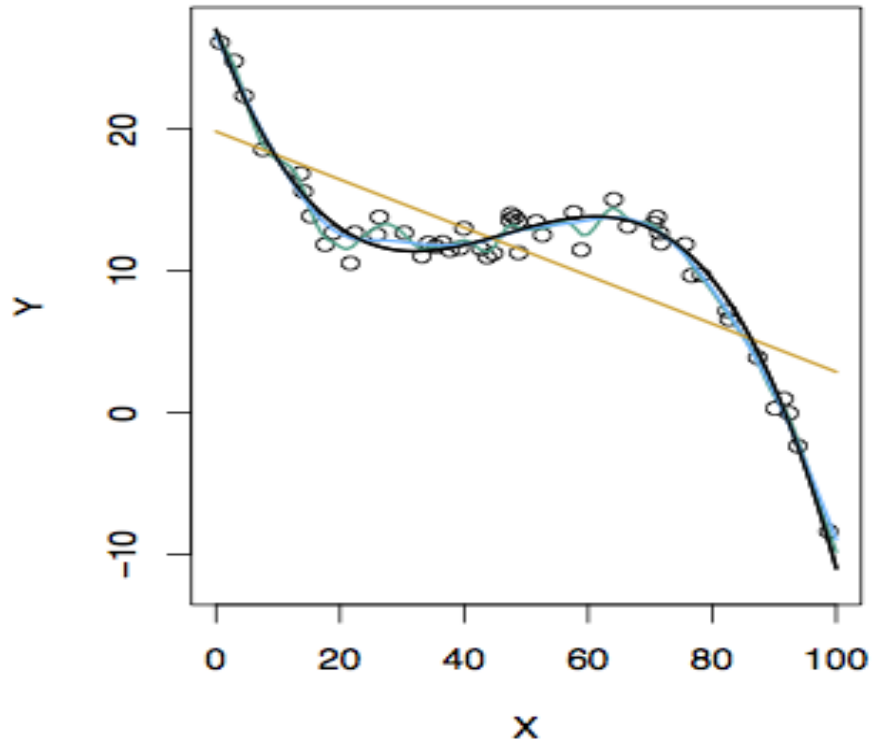
RIGHT

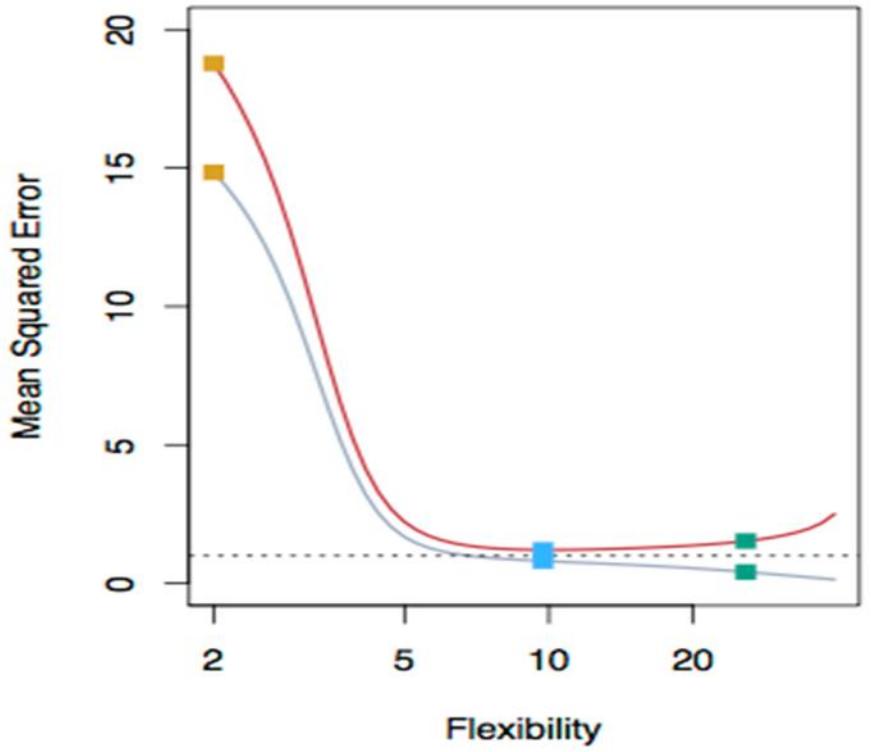
RED: Test MSE

Grey: Training MSE

Dashed: Minimum possible test MSE (irreducible error)

Different levels of flexibility: example 3





LEFT
Black: Truth
Orange: Linear Estimate
Blue: smoothing spline
Green: smoothing spline (more flexible)

RIGHT
RED: Test MSE
Grey: Training MSE
Dashed: Minimum possible test MSE (irreducible error)

Bias - variance trade-off

- The previous graphs of test versus training MSE's illustrates a very important trade-off that governs the choice of statistical learning methods.

- There are always two competing forces that govern the choice of learning method, i.e. bias and variance.

Bias of learning methods

- Bias in general: inclination or prejudice for or against one person or group, especially in a way considered to be unfair.

- Bias in ML refers to the error that is introduced by modeling a real-life problem (that is usually extremely complicated) by a much simpler problem.
- A common definition of bias:

$$\text{Bias} = E[Y]-f(x)$$

- For example, linear regression assumes that there is a linear relationship between Y and X. It is unlikely that, in real life, the relationship is exactly linear so some bias will be present.

- The more flexible/complex a method is the less bias it will generally have.

Variance of learning methods

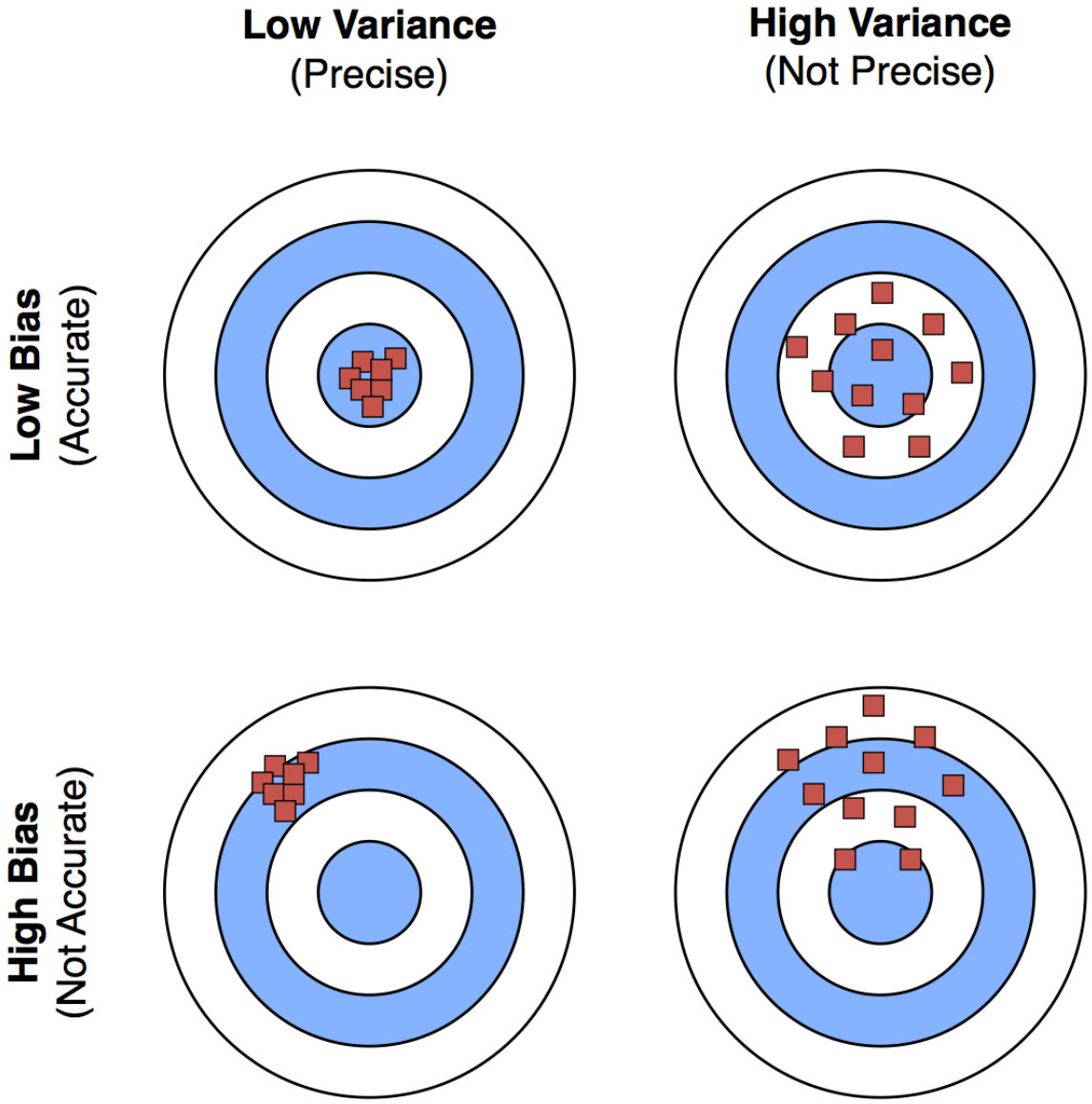
- Variance refers to how much your estimate for f would change if you had a different training data set.

- A common definition of variance:

  $\text{Var} = E[(Y - E[Y])^2]$

- Generally, the more flexible a method is, the more variance it has.

Bias-variance illustration



This work by Sebastian Raschka is licensed under a
Creative Commons Attribution 4.0 International License.

The trade-off?

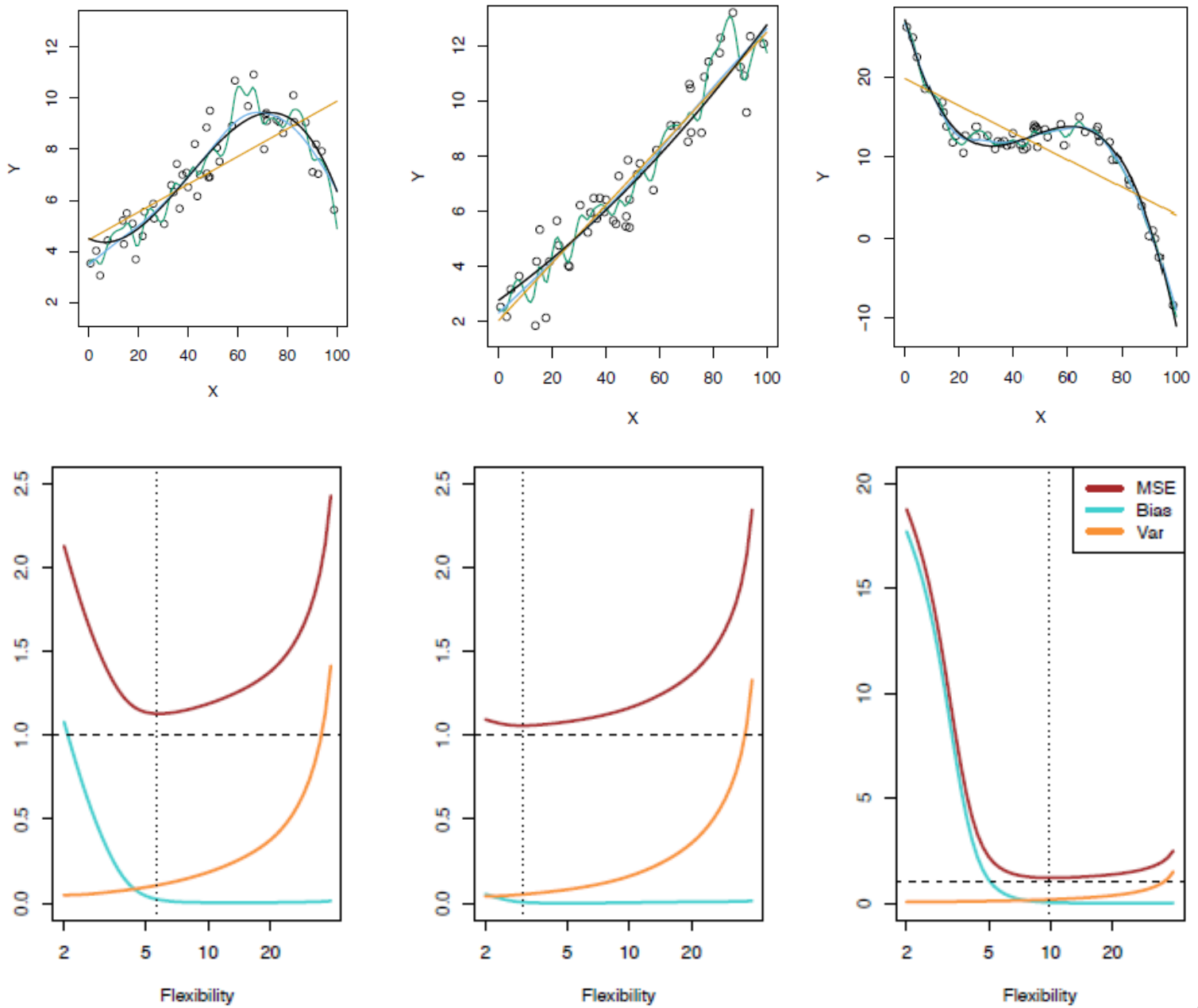
- It can be shown that for any given, $X=x_0$, the expected test MSE for a new Y at x_0 will be equal to

$$Expected\,Test\,MSE = E\left(Y - f(x_0)\right)^2 = Bias^2 + Var + \underbrace{\sigma^2}_{\text{Irreducible Error}}$$

where Bias = E[Y]-f(x) and Var = $E[(Y - E[Y])^2]$

- What this means is that as a method gets more complex
  - the bias will decrease and
  - the variance will likely increase
  - but expected test MSE may go up or down!
- The trade-off is only present if we assume fixed error!
- For some models there may be no trade-off!

Test MSE, bias and variance



Bayes classifier

- In classification, the optimal classification for an instance (x_0, y_0) can be obtained by selecting the class *j* which maximizes the probability

$$P(Y = j | X = x_0)$$

- This classifier is called the Bayes (optimal) classifier
- It implies that learning is actually an estimation of the conditional data distribution

Bayes error rate

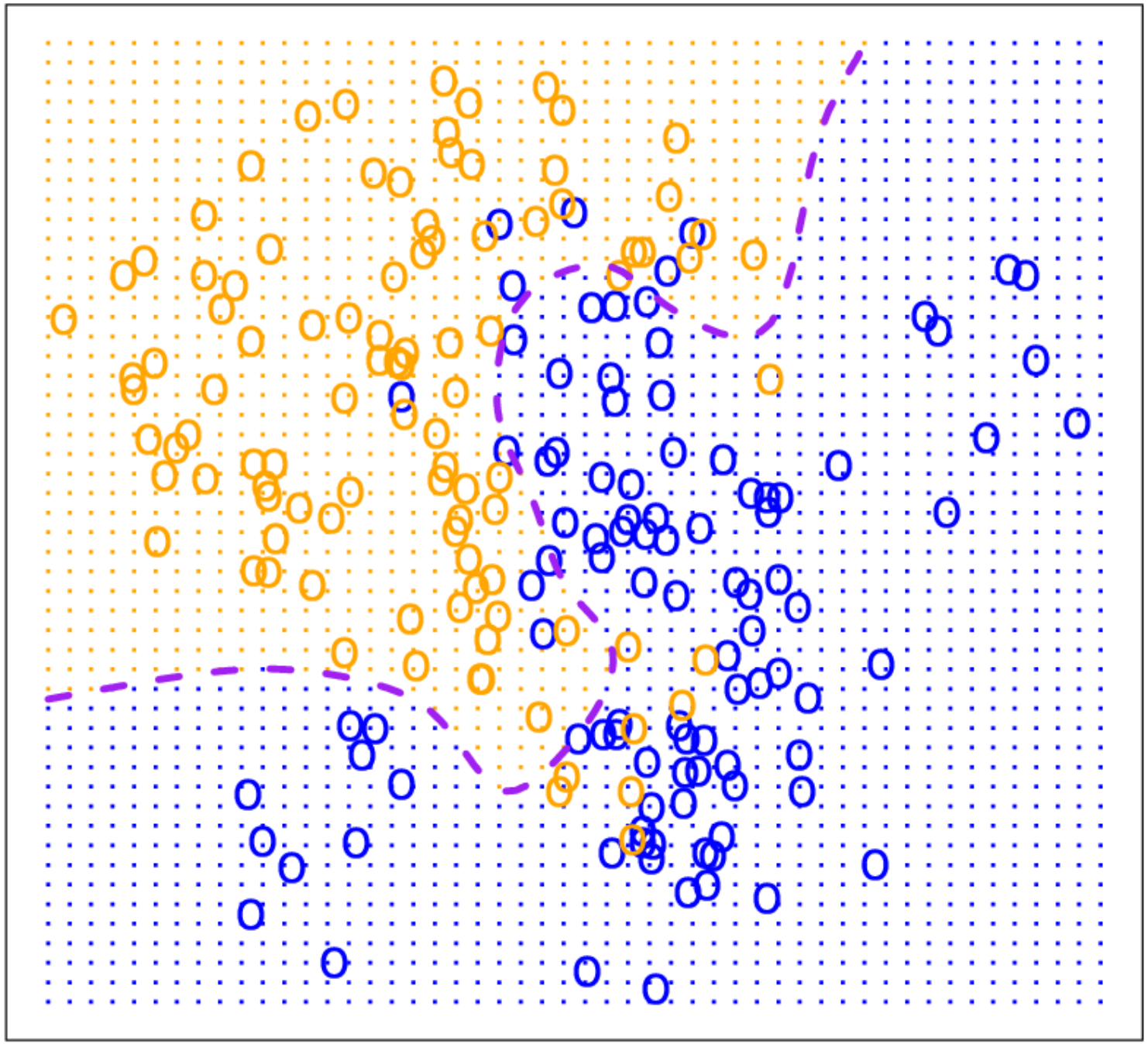
- The Bayes error rate refers to the lowest possible error rate that could be achieved if somehow we knew exactly what the “true” probability distribution of the data looked like.

- On test data, no classifier (or statistical learning method) can get lower error rates than the Bayes error rate.

- Of course, in real life problems, the Bayes error rate can’t be calculated exactly (why not?) but it is useful to think about it

Bayes optimal classifier

- for new x_0 returns the maximally probable prediction value $P(Y=y \mid X=x_0)$
- this means to select the class j with $\arg\max_j P(Y=y_j \mid X=x_0)$
- Why is this probability not easy to estimate?
- The dotted line show the Bayes decision boundary, where $P(Y=y \mid X=x_0) = 0.5$



Bayes classifier approximations

- Two models can be viewed as directly approximating the Bayes classifier $P(Y = j | X = x_0)$
- Naive Bayesian classifier
  - uses Bayesian formula to get inverse conditional probabilities
  - assuming conditional independence between features

$$P(C|X_1X_2 \dots X_n) = \frac{P(C) \cdot P(X_1X_2 \dots X_n|C)}{P(X_1X_2 \dots X_n)} \approx \frac{P(C) \cdot \prod_i P(X_i|C)}{\prod_i P(X_i)}$$

- Nearest neighbour classifier
  - directly estimates the conditional probability using instances near to x_0

K-Nearest Neighbors (KNN)

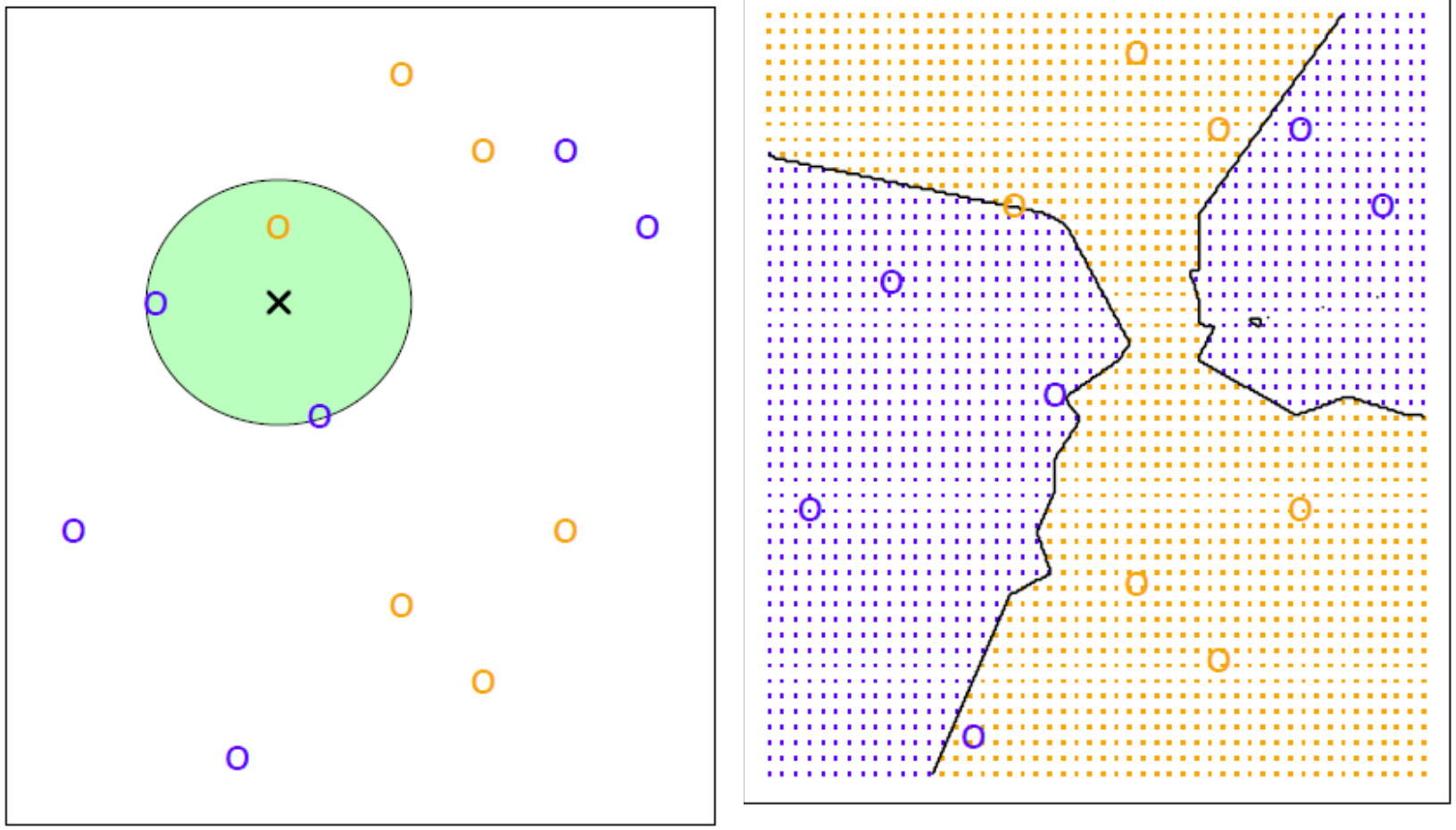
- k Nearest Neighbors is a flexible approach to estimate the Bayes classifier.

- For any given *x* we find the *k* closest neighbors to *x* in the training data, and examine their corresponding *y*.

- If the majority of the *y*'s are orange, we predict the orange label otherwise the blue label.

- The smaller that *k* is the more flexible the method will be.

KNN example with k = 3



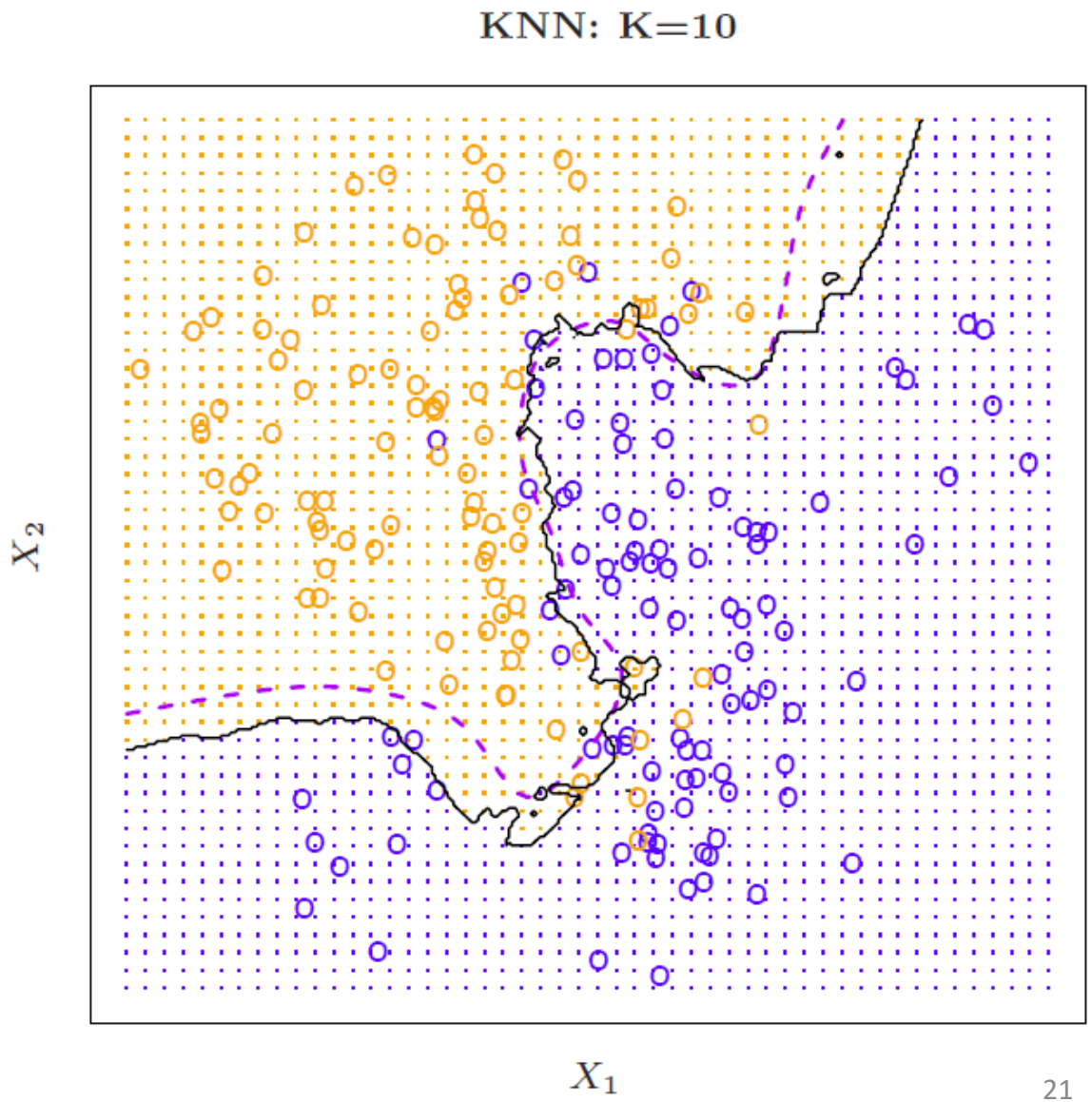
K-NN classifier

- Given a positive integer *K* and a test observation x_0, the KNN classifier first identifies the *K* points in the training data that are closest to x_0, represented by the set $\mathcal{N}_0$.
- It then estimates the conditional probability for class *j* as the fraction of points in $\mathcal{N}_0$ whose response values equal *j*:

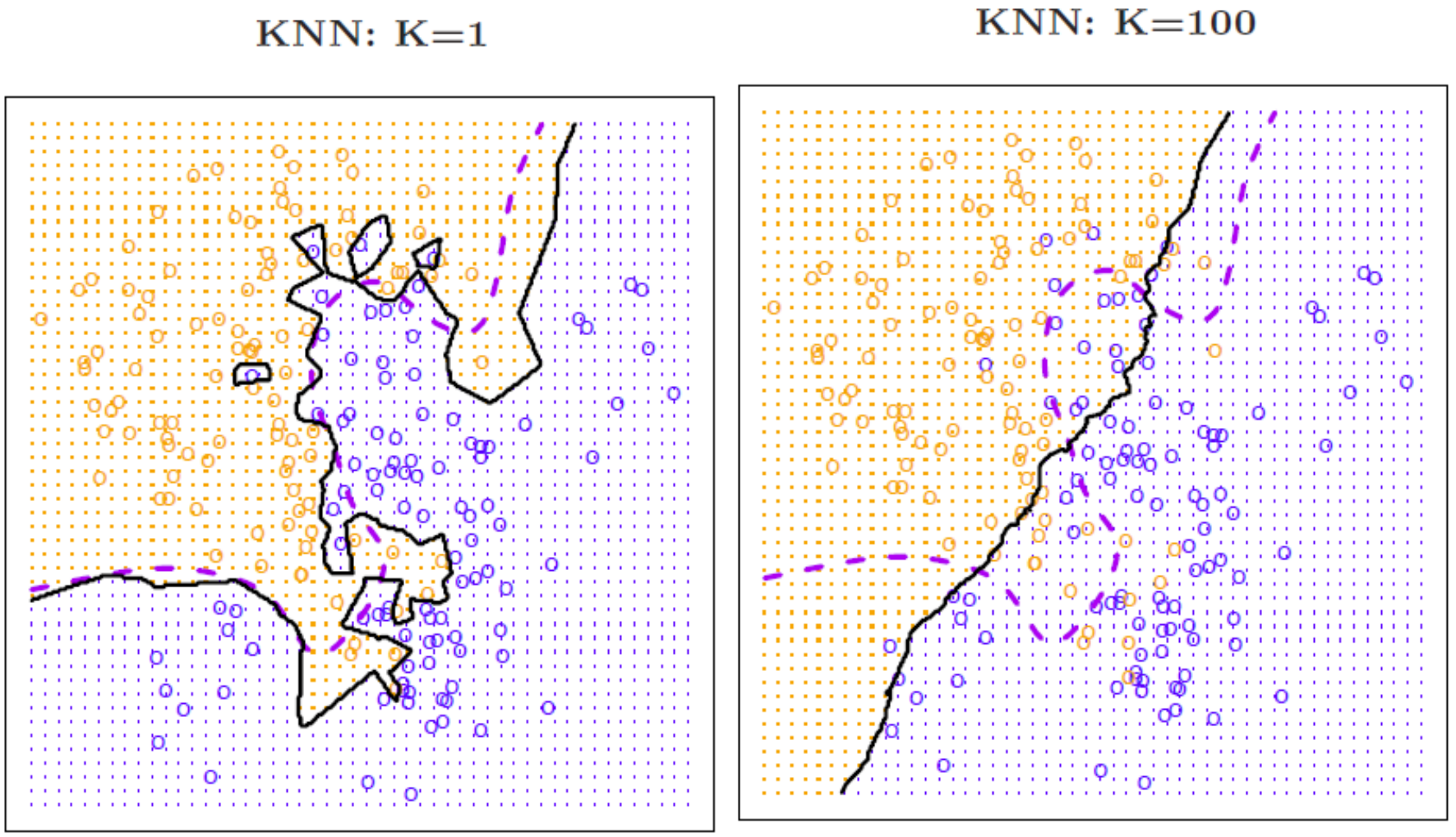
$$\Pr(Y = j|X = x_0) = \frac{1}{K} \sum_{i \in \mathcal{N}_0} I(y_i = j).$$

- applies Bayes rule and classifies the test observation x_0 to the class with the largest probability.

Simulated data:
K = 10

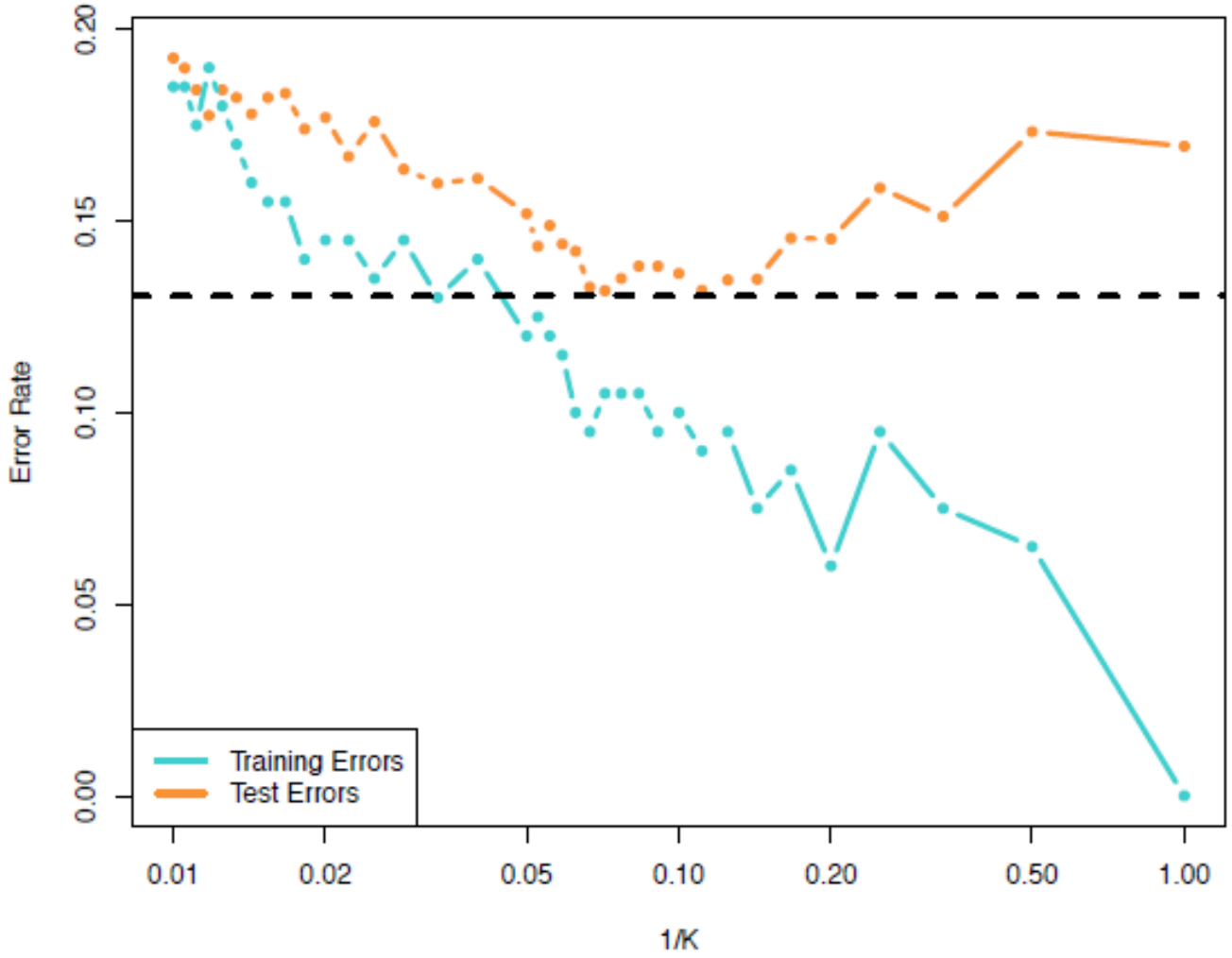


K = 1 and K = 100



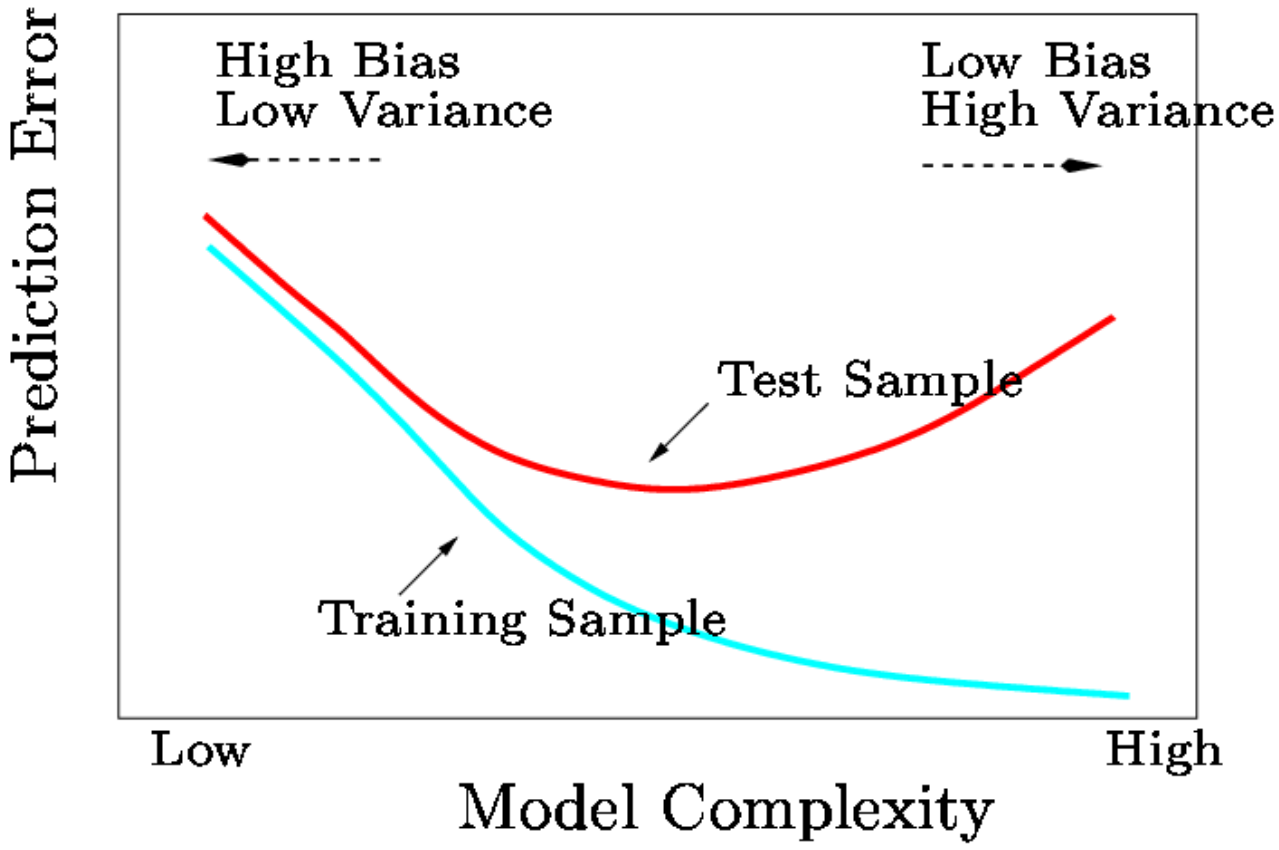
Training vs. test error rates on the simulated data

- Notice that training error rates keep going down as k decreases or equivalently as the flexibility increases.
- However, the test error rate at first decreases but then starts to increase again.



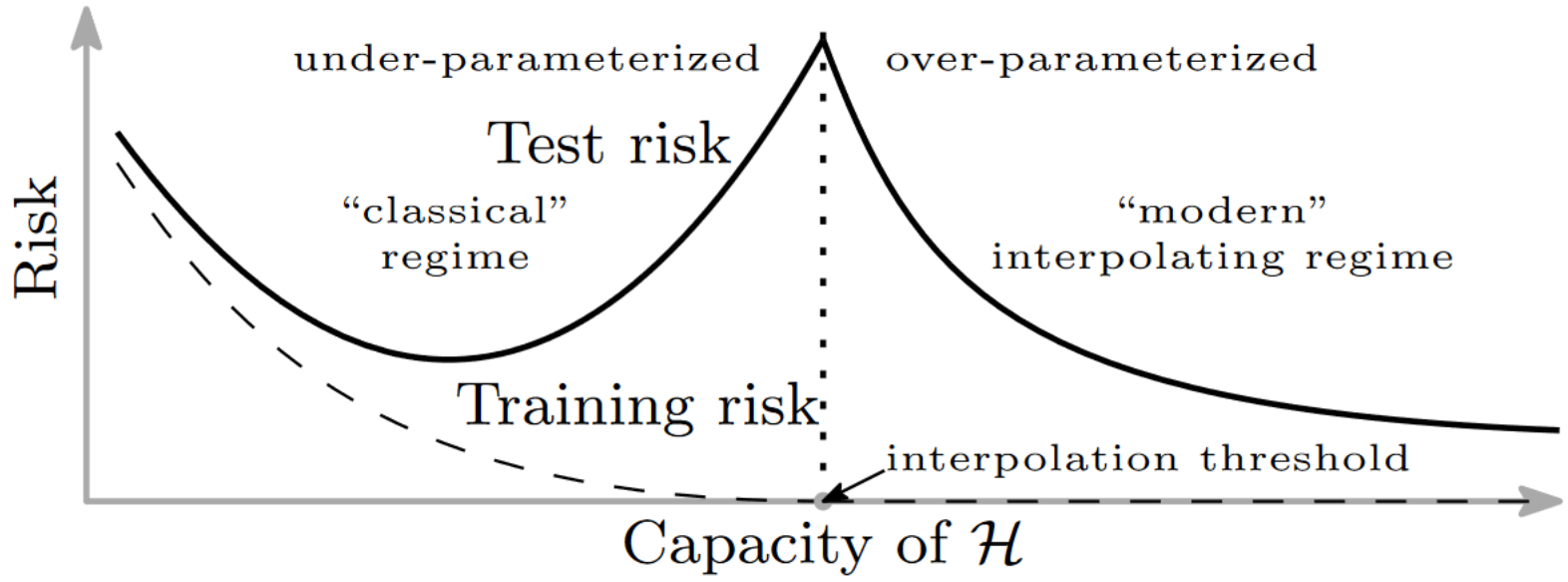
A fundamental picture

- In general training errors will always decline.
- However, test errors will decline at first (as reductions in bias dominate) but will then start to increase again (as increases in variance dominate).
- This is a conventional wisdom, but it is not true for all methods and all training regimes.



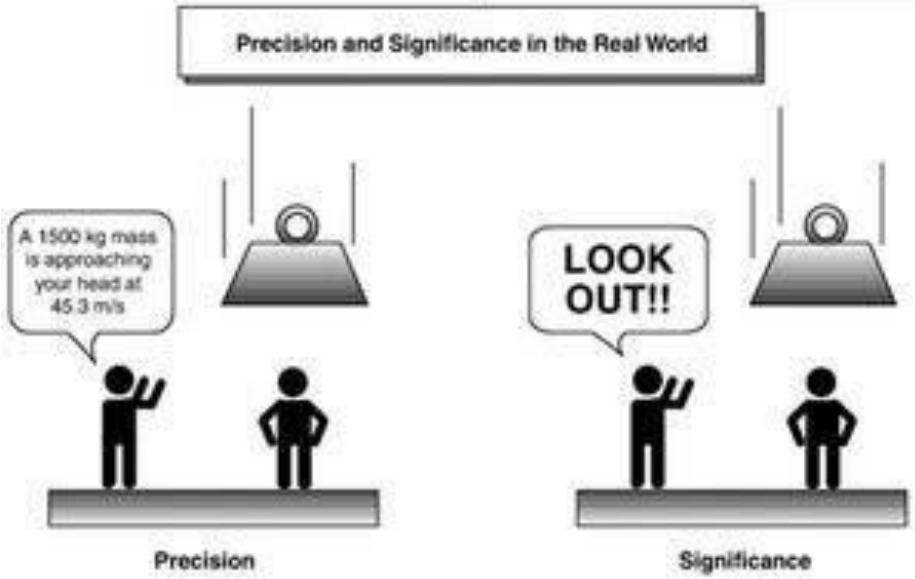
The double descent curve

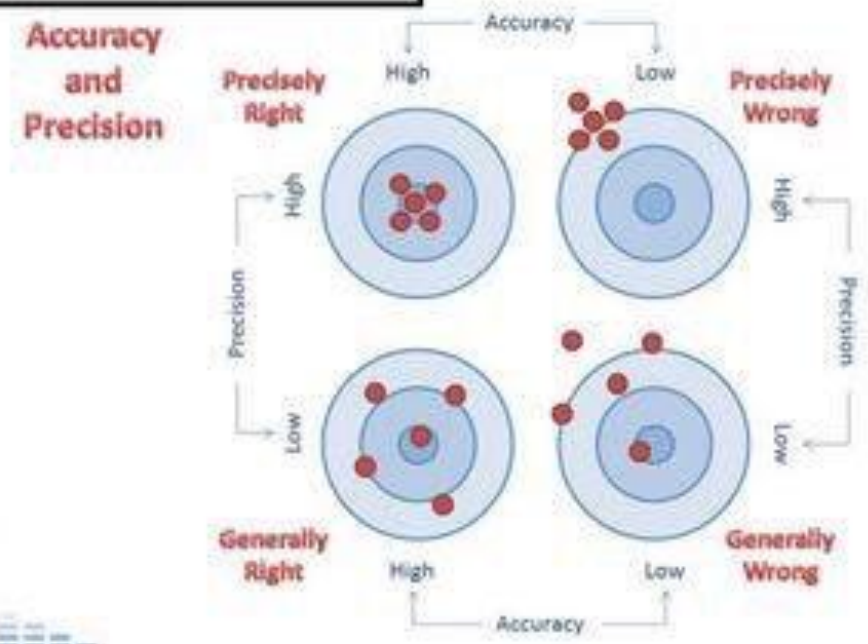
- While for some models, like kNN, there seem to be a trade-off between bias and variance, this is not a universal phenomenon
- E.g., overparametrization in neural networks produce double descent curve (similar evidence for random forests)

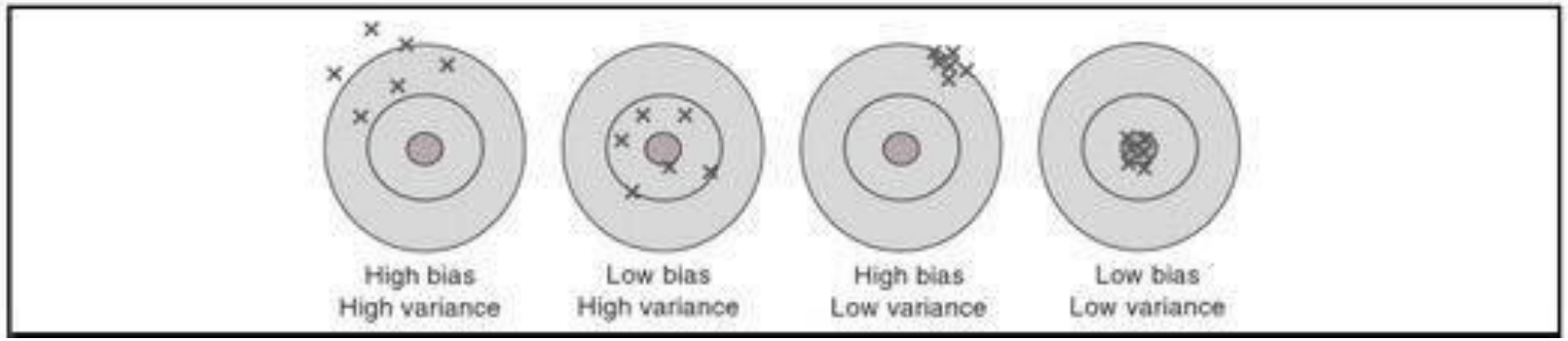


Belkin, M., Hsu, D., Ma, S. and Mandal, S., 2019. Reconciling modern machine-learning practice and the classical bias–variance trade-off. *Proceedings of the National Academy of Sciences*, *116*(32), pp.15849-15854.

Precision vs. Significance
Accuracy vs. Precision
Bias vs. Variance





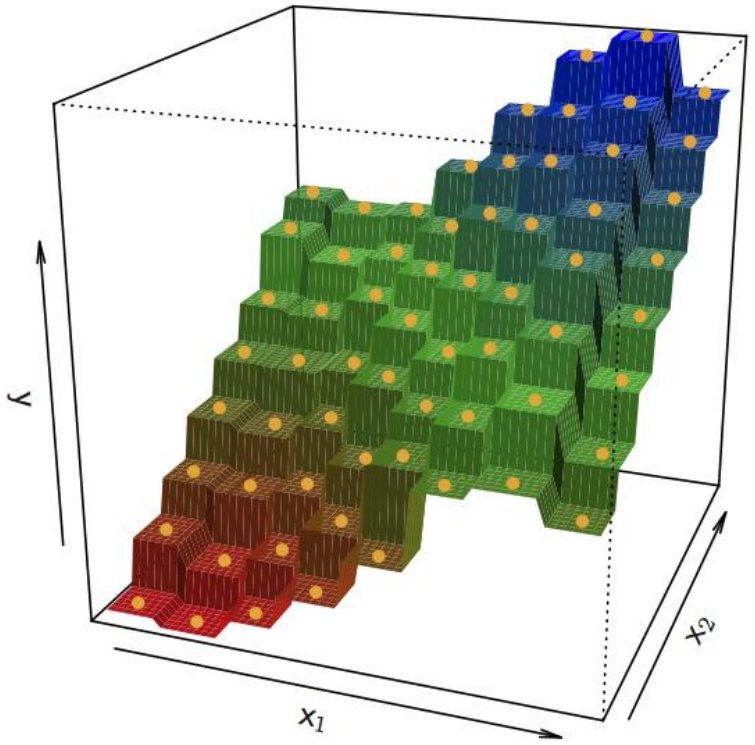


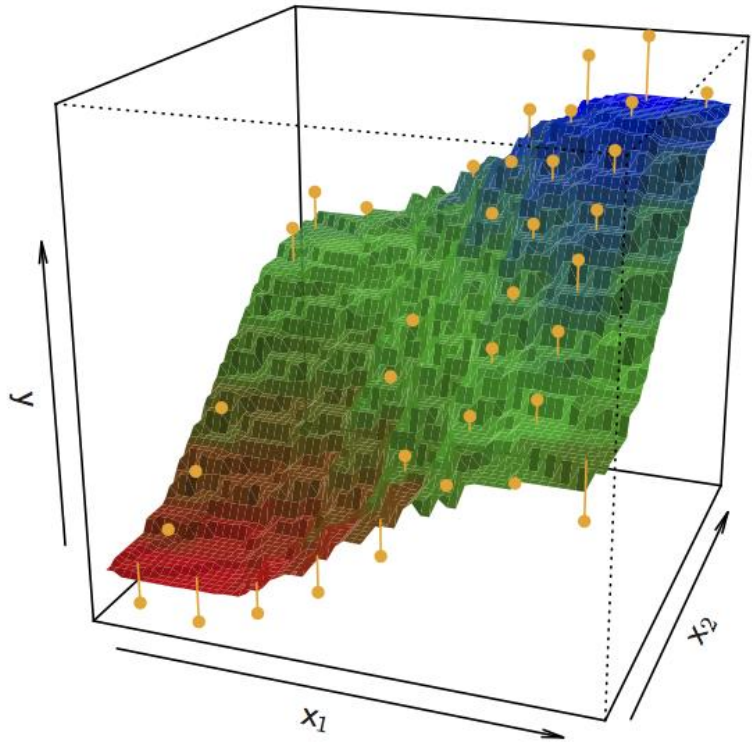
K-nearest neighbor for regression

- kNN regression is similar to the kNN classifier.
- given a set of instances (x_i, y_i)
- To predict y for a given value of x, consider k closest points to x in training data $N_k(x)$ and take the average of the responses. i.e.

$$f(x) = \frac{1}{k} \sum_{x_i \in N_k(x)} y_i$$

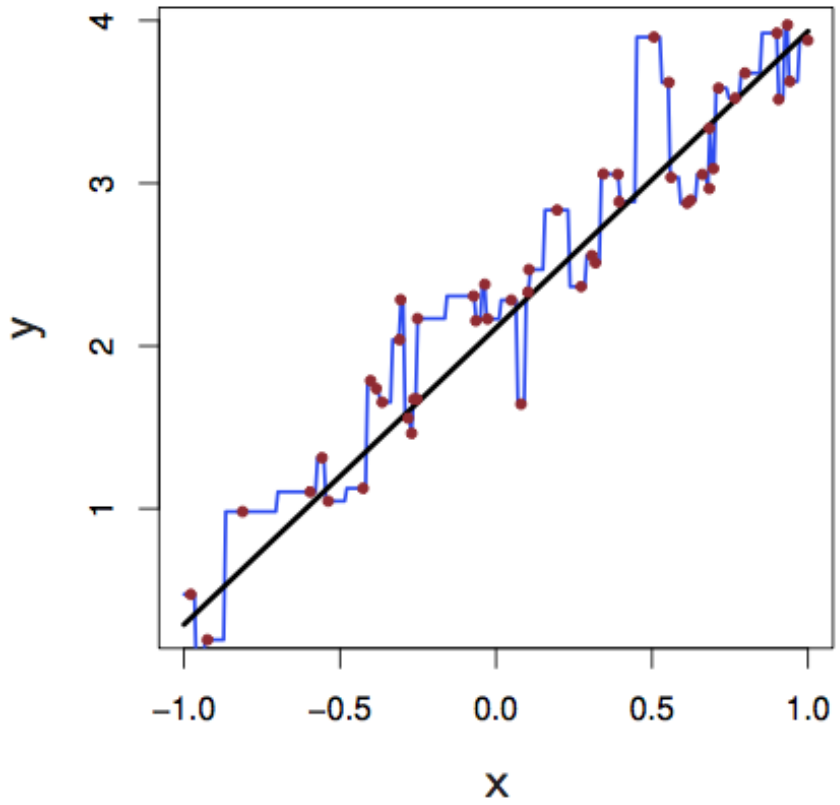
KNN Fits for k =1 and k = 9

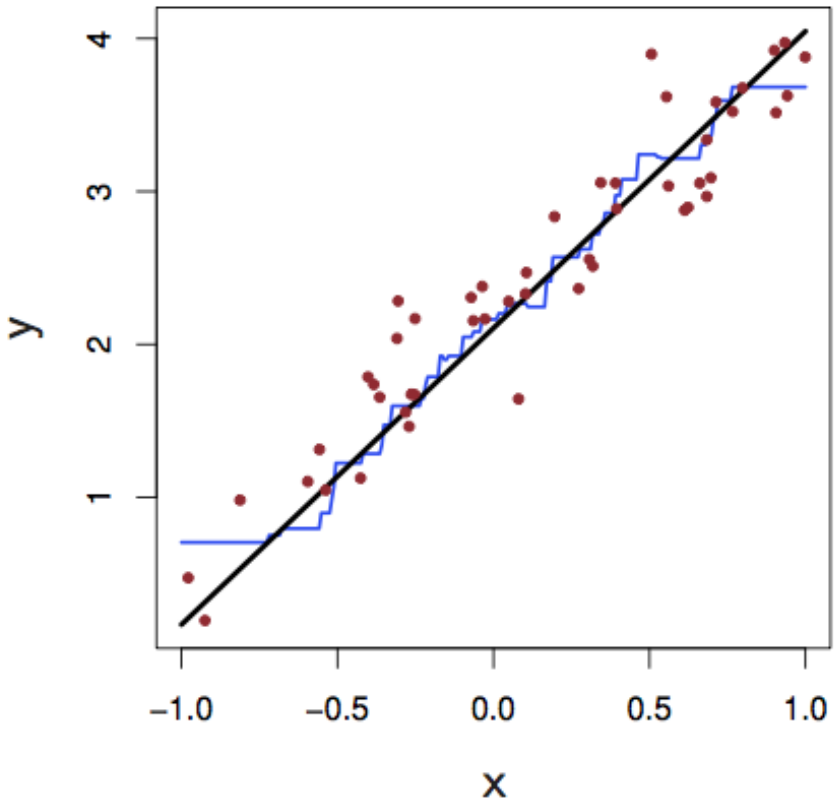




KNN fits in one dimension (k =1 and k = 9)

- black line: actual function,
- blue line: regressional kNN

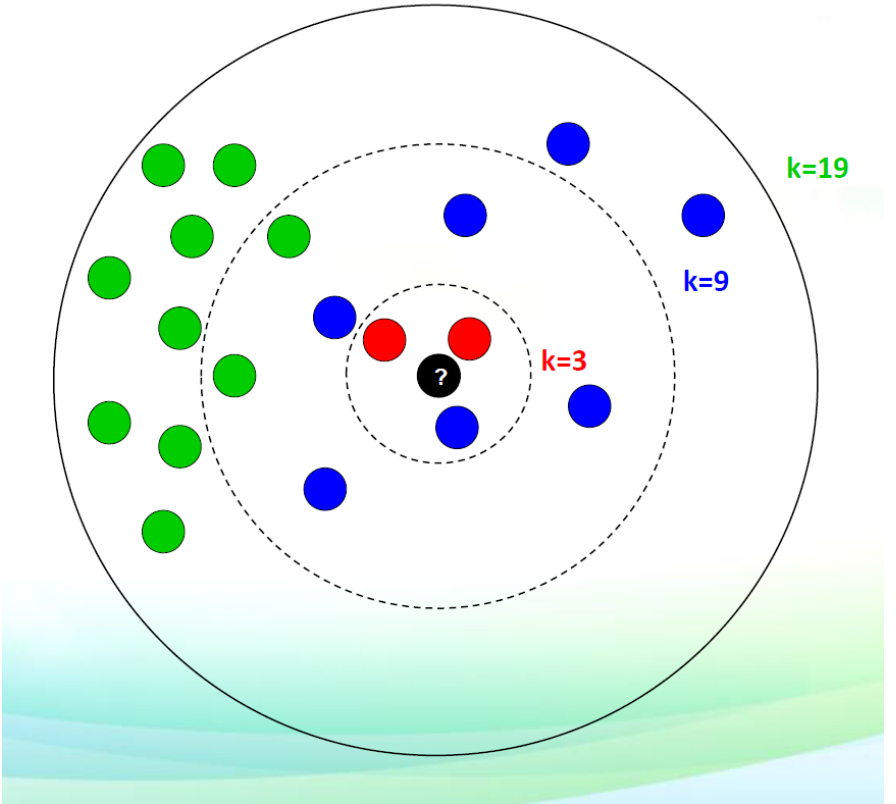




Choice of k in KNN

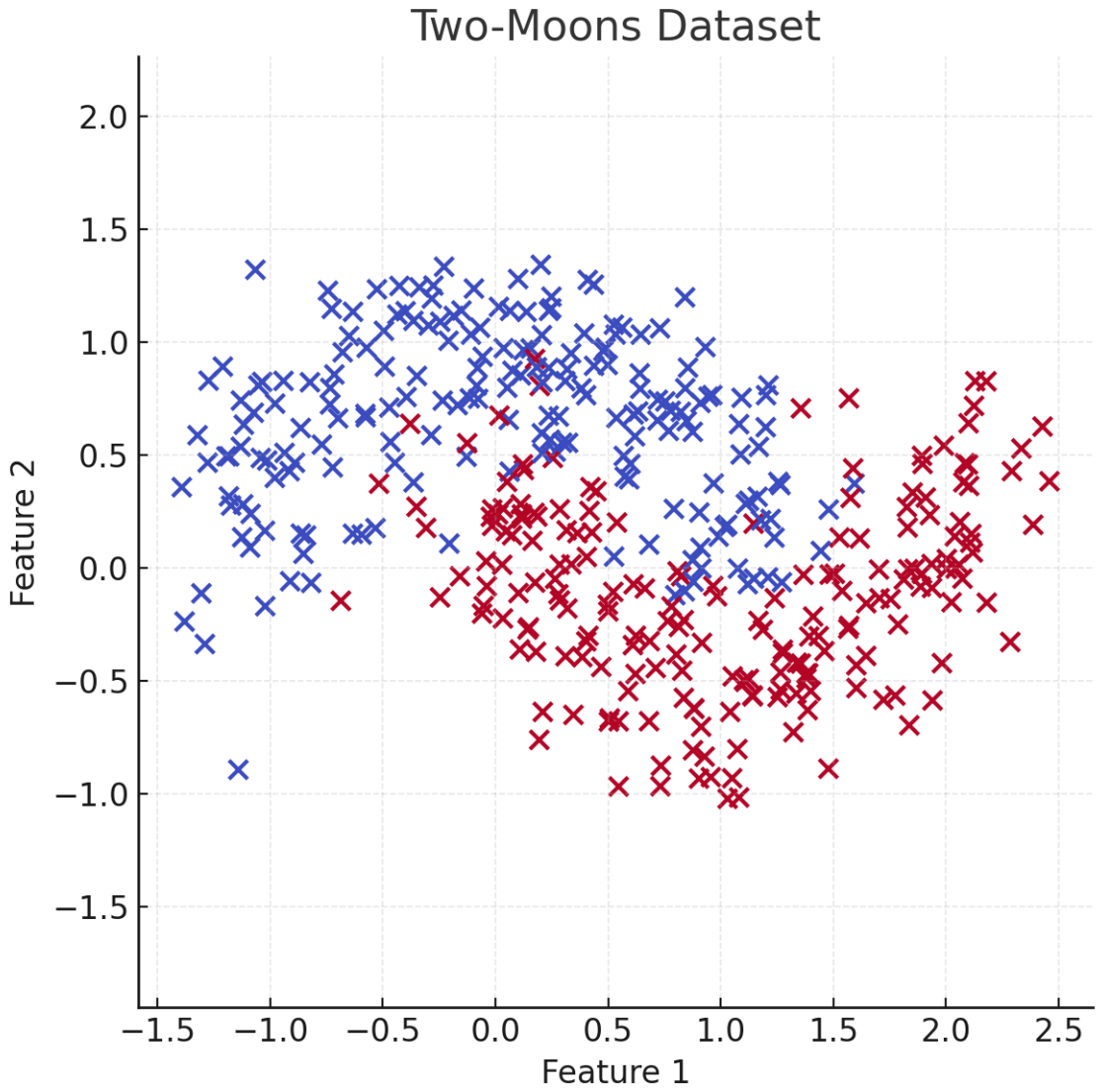
- If *k* is small, kNN is much more flexible than linear regression.
- Is that better?

- The results may be highly dependent on the choice of *k*.

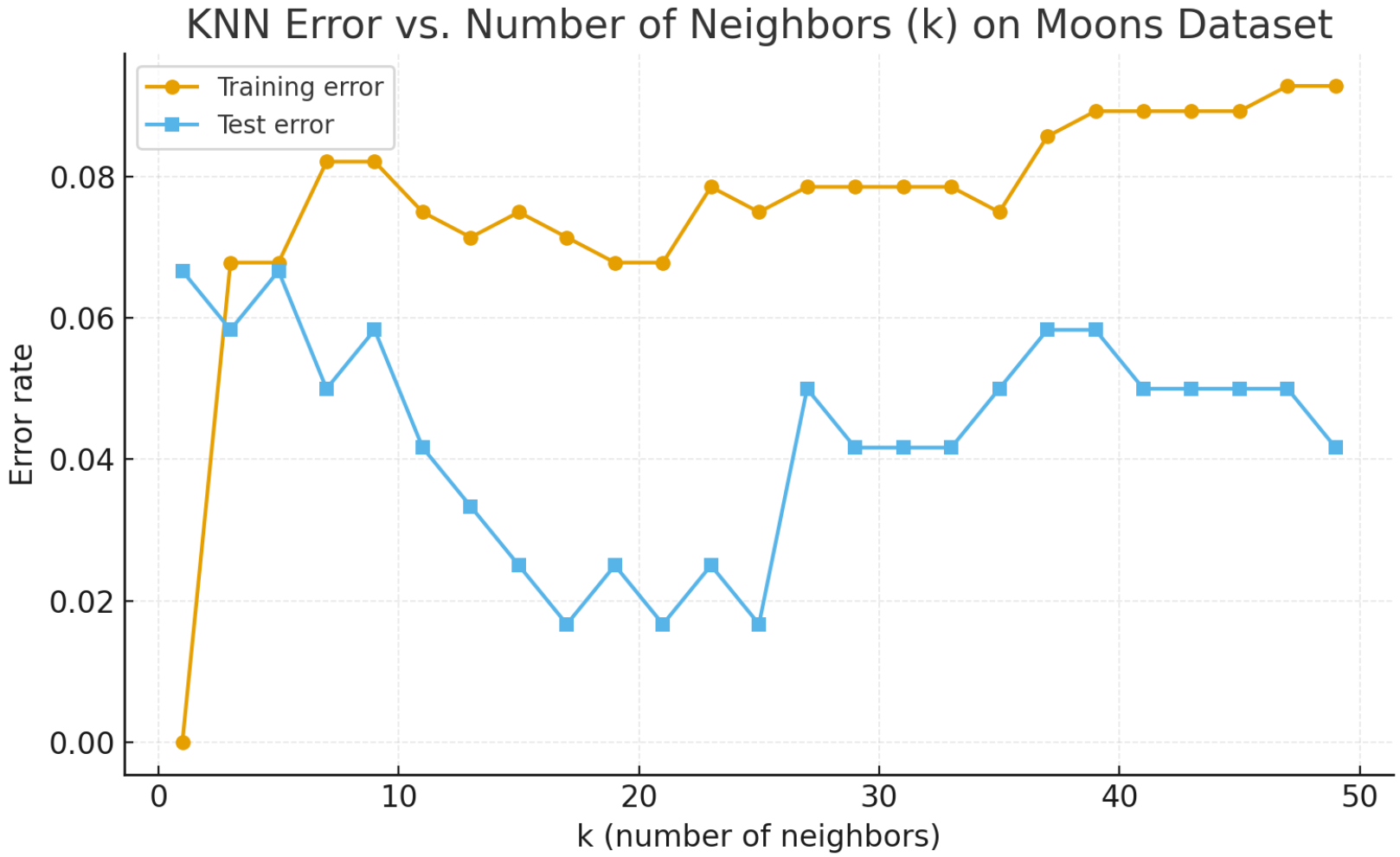


Example: two moons dataset

- Non-linear decision boundary



KNN is not so good in high dimensional situations



Speeding up KNN algorithm

- precondition: normalization of dimensions, e.g., to [0, 1]
- naive search for nearest neighbors: $O(n \cdot d \cdot t)$
  - n is number of instances
  - d is number of dimensions
  - t is number of nearest neighbors
- exact search for low dimensional spaces
  - k-d trees (d is around 10)
  - quad-trees (d=2), octrees (d=3)
  - R-tree (rectangular tree, also R+, R*, ...), d=2 or 3
- approximate search
  - RKD-tree (random k-d tree)
  - locally sensitive hashing (LSH),
  - hierarchical k-means

Word bias have several meanings

1. General / Everyday Meaning

- **Preference or prejudice** toward or against something or someone, often in an unfair way.
  - *Example:* "The judge must avoid bias in court."
  - → *Synonyms:* partiality, favoritism, prejudice.
- **Tendency** to lean in a certain direction, consciously or unconsciously.
  - *Example:* "She has a bias toward traditional art styles."

2. Social and Psychological Context

- **Cognitive bias**: A systematic pattern of deviation from rational judgment or objective standards.
  - *Example: confirmation bias* (favoring information that confirms one's beliefs).
- **Social bias**: Prejudice or discrimination based on group characteristics (e.g., gender, race, culture).
  - *Example:* "Hiring practices should be checked for gender bias."

3. Statistics & Machine Learning

- **Statistical bias**: Systematic error that leads an estimator or model to deviate from the true value.
  - *Example:* "The sample mean is an unbiased estimator of the population mean."
- **Bias–variance tradeoff**: In machine learning, bias refers to the error introduced by simplifying assumptions in a model.
  - High bias → model underfits the data.
  - Low bias → model can better capture complexity.

Ethical consideration of social and cognitive bias in ML models

- The word bias is ambiguous even within ML
- bias in models:
  - characteristic of models,
  - affects error,
  - unlikely to be ethically problematic
  - when it can be problematic?
- bias in data:
  - data unrepresentative of the true population,
  - might be ethically problematic



Biases in the data

- Machine learning models are not inherently objective. Engineers train models by feeding them a data set of training examples, and human involvement in the provision and curation of this data can make a model's predictions susceptible to bias.
- When building models, it's important to be aware of common human biases that can manifest in your data, so you can take proactive steps to mitigate their effects.
- The biases listed next provide just a small selection of biases that are often uncovered in machine learning data sets; this list *is not intended to be exhaustive*. Wikipedia's catalog of cognitive biases enumerates over 100 different types of human bias that can affect our judgment. When auditing your data, you should be on the lookout for any and all potential sources of bias that might skew your model's predictions.

Reporting bias

- **Reporting bias** occurs when the frequency of events, properties, and/or outcomes captured in a data set does not accurately reflect their real-world frequency. This bias can arise because people tend to focus on documenting circumstances that are unusual or especially memorable, assuming that the ordinary can "go without saying."
  - **EXAMPLE**: A sentiment-analysis model is trained to predict whether book reviews are positive or negative based on a corpus of user submissions to a popular website. The majority of reviews in the training data set reflect extreme opinions (reviewers who either loved or hated a book), because people were less likely to submit a review of a book if they did not respond to it strongly. As a result, the model is less able to correctly predict sentiment of reviews that use more subtle language to describe a book.

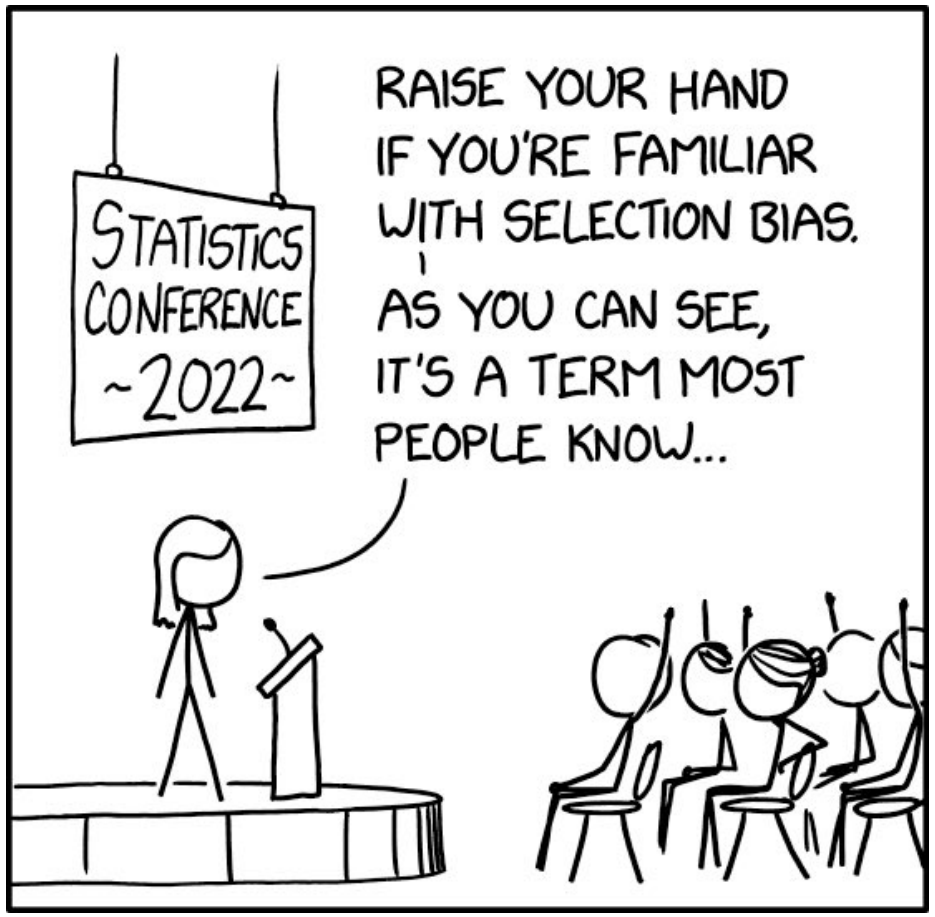
Automation bias

- **Automation bias** is a tendency to favor results generated by automated systems over those generated by non-automated systems, irrespective of the error rates of each (we also have the opposite bias)
  - **EXAMPLE**: Software engineers working for a sprocket manufacturer were eager to deploy the new "groundbreaking" model they trained to identify tooth defects, until the factory supervisor pointed out that the model's precision and recall rates were both 15% lower than those of human inspectors.



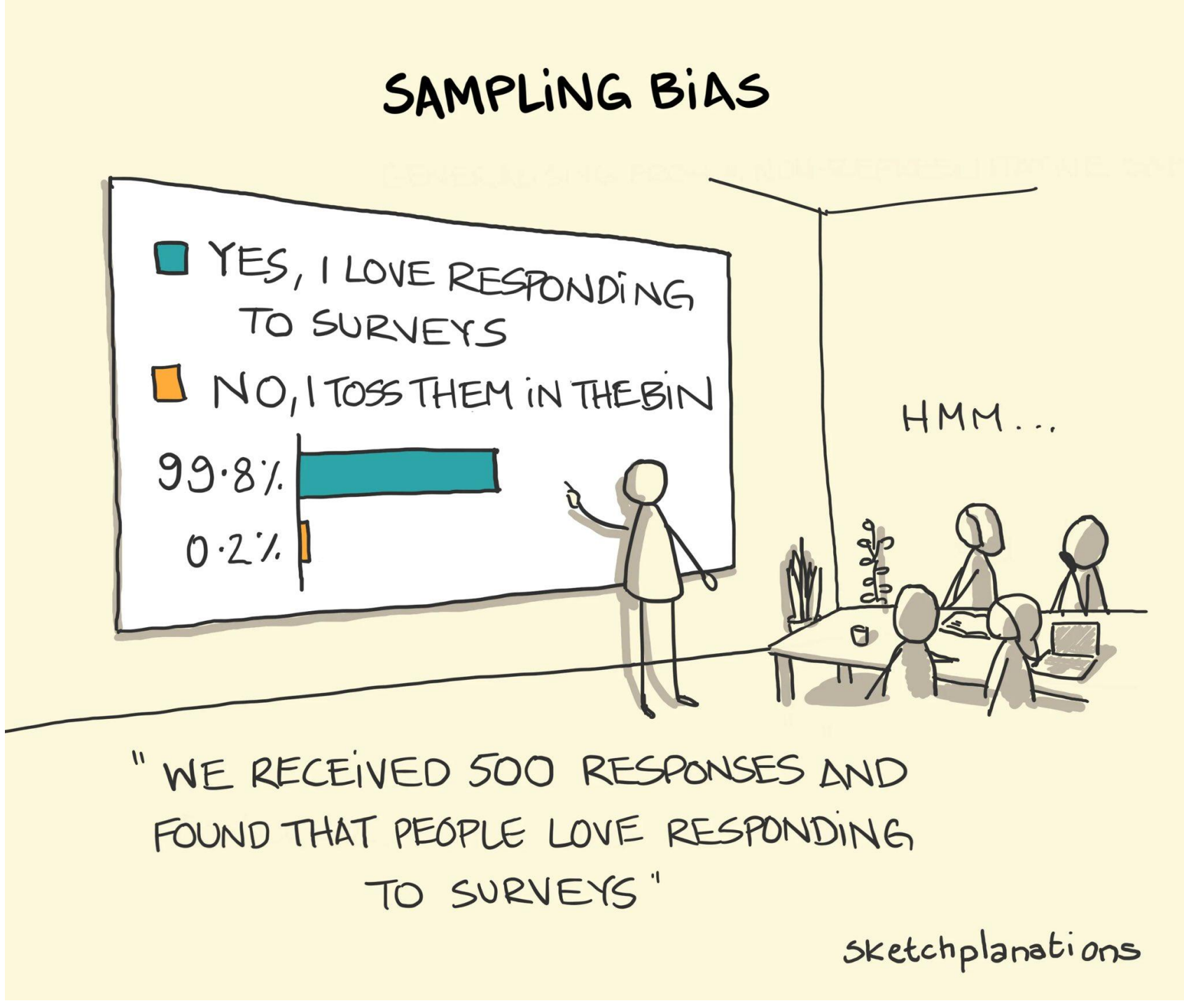
Selection bias

- **Selection bias** occurs if a data set's examples are chosen in a way that is not reflective of their real-world distribution.



Selection bias variants

- Selection bias can take many different forms:
  - **Coverage bias**: Data is not selected in a representative fashion.
    - **EXAMPLE**: A model is trained to predict future sales of a new product based on phone surveys conducted with a sample of consumers who bought the product. Consumers who instead opted to buy a competing product were not surveyed, and as a result, this group of people was not represented in the training data.
  - **Non-response bias** (or **participation bias**): Data ends up being unrepresentative due to participation gaps in the data-collection process.
    - **EXAMPLE**: A model is trained to predict future sales of a new product based on phone surveys conducted with a sample of consumers who bought the product and with a sample of consumers who bought a competing product. Consumers who bought the competing product were 80% more likely to refuse to complete the survey, and their data was underrepresented in the sample.
  - **Sampling bias**: Proper randomization is not used during data collection.
    - **EXAMPLE**: A model is trained to predict future sales of a new product based on phone surveys conducted with a sample of consumers who bought the product and with a sample of consumers who bought a competing product. Instead of randomly targeting consumers, the surveyor chose the first 200 consumers that responded to an email, who might have been more enthusiastic about the product than average purchasers.


SAMPLING BIAS
YES, I LOVE RESPONDING TO SURVEYS
NO, I TOSS THEM IN THE BIN
99.8%
0.2%
HMM...
" WE RECEIVED 500 RESPONSES AND FOUND THAT PEOPLE LOVE RESPONDING TO SURVEYS "
sketchplanations

Group attribution bias

- **Group attribution bias** is a tendency to generalize what is true of individuals to an entire group to which they belong. Two key manifestations of this bias are:
  - **In-group bias**: A preference for members of a group to which *you also belong*, or for characteristics that you also share.
    - **EXAMPLE**: Two engineers training a resume-screening model for software developers are predisposed to believe that applicants who attended the same computer-science academy as they both did are more qualified for the role.
  - **Out-group homogeneity bias**: A tendency to stereotype individual members of a group to which *you do not belong*, or to see their characteristics as more uniform.
    - **EXAMPLE**: Two engineers training a resume-screening model for software developers are predisposed to believe that all applicants who did not attend a computer-science academy do not have sufficient expertise for the role.

Implicit bias

- **Implicit bias** occurs when assumptions are made based on one's own mental models and personal experiences that do not necessarily apply more generally. We are often not aware of these biases, and some may be contrary to our conscious beliefs.
  - **EXAMPLE**: An engineer training a gesture-recognition model uses a head shake as a feature to indicate a person is communicating the word “no.” However, in some regions of the world, a head shake actually signifies “yes.” A common form of implicit bias is **confirmation bias**, where model builders unconsciously process data in ways that affirm preexisting beliefs and hypotheses. In some cases, a model builder may actually keep training a model until it produces a result that aligns with their original hypothesis; this is called an **experimenter's bias**.
  - **EXAMPLE**: An engineer is building a model that predicts aggressiveness in dogs based on a variety of features (height, weight, breed, environment). The engineer had an unpleasant encounter with a hyperactive toy poodle as a child, and ever since has associated the breed with aggression. When the trained model predicted most toy poodles to be relatively docile, the engineer retrained the model several more times until it produced a result showing smaller poodles to be more violent.

Implicit Bias is...

Attitudes, Stereotypes, & Beliefs

that can affect how we treat others

based on categorizations such as...

Race



Ability



Gender



Culture



Language



Implicit bias runs contrary to our stated beliefs. We can say that we believe in equity (and truly believe it). But then unintentionally behave in ways that are biased and discriminatory.

Adapted from @DrJRMarcelin

Good question! **Measuring bias and variance** of a machine learning method on a specific dataset is a **core diagnos
Here's a **clear and practical breakdown** of how to measure **bias** and **variance** in this context:

Where:

- **Bias** measures how far the **average prediction** of the model is from the true value.
- **Variance** measures how much **predictions fluctuate** when the model is trained on different samples of the dataset.
- **Irreducible error** is noise inherent in the data.

In practice, we approximate these quantities by **resampling the dataset**, training multiple models, and analyzing predict

- Use a **held-out test set** for evaluation to avoid optimistic bias.
- Bias and variance are typically computed per class or per example and then averaged.
- Resampling (e.g., bootstrapping) can be expensive — use smaller subsamples or Monte Carlo dropout for an a
- **Bias-variance trade-off** can be visualized by plotting bias², variance, and total error as a function of model complexity
- **Low bias, low variance** is the ideal but often hard to achieve with limited or noisy training data.
- **Low bias, high variance** (overfitting): model is too complex.
- **High bias, low variance** (underfitting): model is too simple.

Would you like me to adapt this method specifically for your use case (e.g. regression vs classification, neural nets vs tre

Revision question:
How to measure bias and variance of an ML method?

Background

- For a given data point x_0,a model's prediction can be decomposed as:
- $\mathbb{E}[(\hat{y}(x_0) - y(x_0))^2] = \mathit{Bias}^2(x_0) + \mathit{Variance}(x_0) + \mathit{Irreducible\ Error}$.

- Bias: error due to simplifying assumptions
- Variance: sensitivity to training set fluctuations
- Key idea: use resampling or cross-validation to estimate bias and variance

Practical steps 1/3

- Generate M training sets (bootstrap or k-fold)
- Train models on each set
- Collect predictions on a fixed test set for all models:

$$\hat{y}_m(x_i) = f_m(x_i),$$

for each data point x_i and model m.

Practical steps 2/3

- Compute mean prediction, bias² and variance per point

$$\bar{y}(x_i) = \frac{1}{M}\sum_{m=1}^{M} \hat{y}_m(x_i)$$

$$\text{Bias}^2(x_i) = (\bar{y}(x_i) - y(x_i))^2$$

$$\text{Variance}(x_i) = \frac{1}{M}\sum_{m=1}^{M} (\hat{y}_m(x_i) - \bar{y}(x_i))^2$$

Practical steps 3/3

- Average across all test data points

$$\text{Bias}^2 = \frac{1}{N}\sum_{i=1}^{N} Bias^2\,(x_i)$$

$$\text{Variance} = \frac{1}{N}\sum_{i=1}^{N} Variance\,(x_i)$$

-

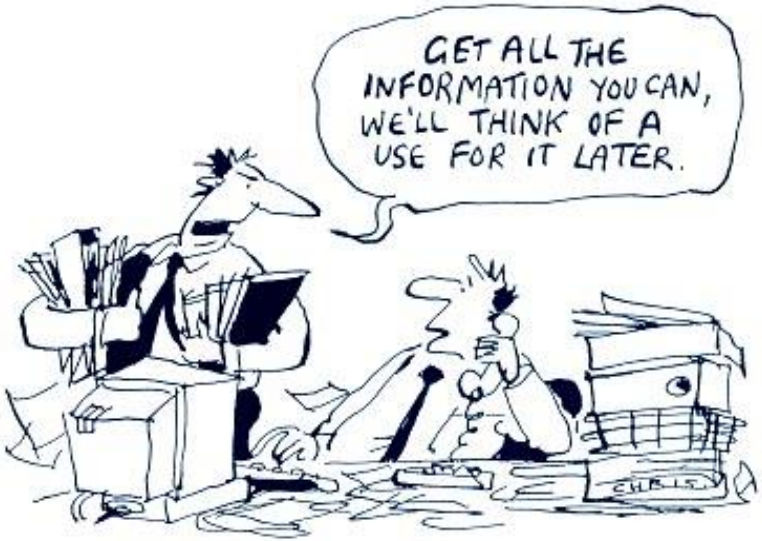
Implementation

- Use bootstrapping or multilevel cross-validation to train multiple models; compute bias and variance
- Use M = 30–100 for stable estimates

Interpretation

- High bias, low variance → underfitting
- Low bias, high variance → overfitting
- Low bias, low variance → good fit
- Plot bias^2 and variance vs model complexity for insights

Data preprocessing: feature engineering



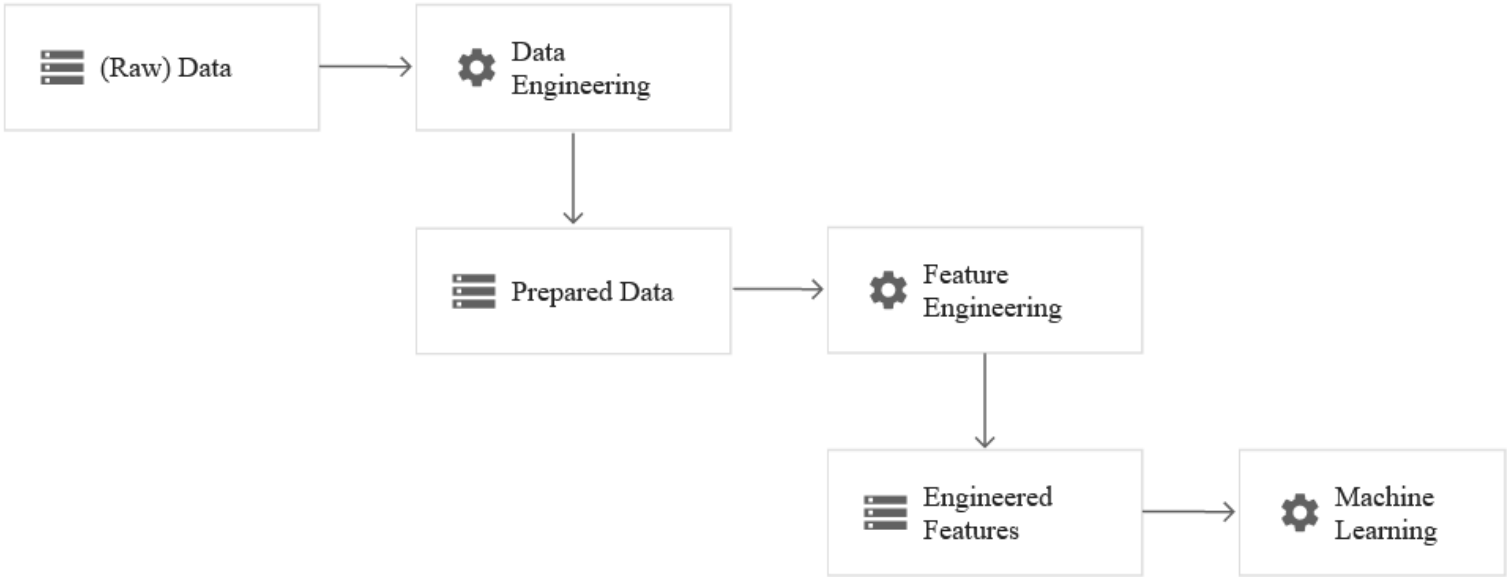
Prof Dr Marko Robnik-Šikonja
Intelligent Systems, Edition 2025

Contents

- Data preprocessing
- Feature subset selection: filter, wrapper and embedded methods
- Feature creation: constructive induction
- Feature selection extensions: unsupervised and semi-supervised learning, multi-task, multi-view, multi-label learning
- Model evaluation
- Dimensionality reduction

First steps in ML

- The data preparation step is seriously underestimated



Data preprocessing

- **Data cleansing:** removing or correcting records that have corrupted or invalid values from raw data, and removing records that are missing a large number of columns.
- **Instances selection and partitioning:** selecting data points from the input dataset to create training, evaluation (validation), and test sets. This process includes techniques for repeatable random sampling, minority class oversampling, and stratified partitioning.
- **Feature tuning:** improving the quality of a feature for ML, which includes scaling and normalizing numeric values, imputing missing values, clipping outliers, and adjusting values that have skewed distributions.
- **Feature transformation:** converting a numeric feature to a categorical feature (through discretization), or converting categorical features to a numeric representation (through one-hot encoding, sparse and dense feature embeddings). Some models work only with numeric or categorical features, while others can handle mixed-type features. Even when models handle both types, they can benefit from different representations (numeric and categorical) of the same feature.
- **Feature extraction:** reducing the number of features by creating lower-dimension, more powerful data representations using techniques such as PCA, embedding extraction, and hashing.
- **Feature selection:** selecting a subset of the input features for training the model, and ignoring the irrelevant or redundant ones, using filter or wrapper methods. Feature selection can also involve simply dropping features if the features are missing a large number of values.
- **Feature construction:** creating new features by using different operators, such as logical, arithmetical, trigonometrical, etc. Features can also be constructed by using domain knowledge, e.g., business logic from the domain of the ML use case.
- **For unstructured data:** often only modest preprocessing is needed for neural networks
  - text: casing, tokenization, embedding lookup/calculation
  - images: resizing, cropping, filters.

Why Reduce Dimensionality?

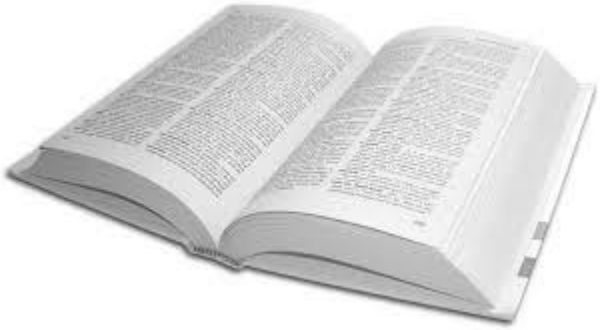
- Reduces time complexity: Less computation
- Reduces space complexity: Less parameters
- Potentially saves the cost of observing the feature
- Simpler models are more robust on small datasets
- More interpretable; simpler explanation
- Data visualization (structure, groups, outliers, etc) if plotted in 2 or 3 dimensions

Feature subset selection



- Choose a small subset of the relevant features from the original features by removing irrelevant, redundant and/or noisy features
- The aim: better learning performance, i.e. higher learning accuracy, lower computational cost, or better model interpretability

Huge number of features

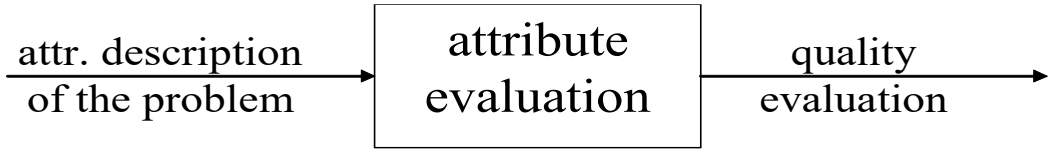


- Text classification, ≈ 100,000 words in a dictionary
- Bioinformatics, ≈ 10,000 measurements of gene expression levels
- Computer vision, ≈ 1,000,000 pixels





Evaluation of attributes



- Numerical evaluation and ranking of the attributes
- The success of the evaluation procedure depends on the role it plays in learning:
  - feature subset selection
  - building of the tree-based models
  - constructive induction
  - discretization
  - attribute weighting
  - comprehension
  - prediction
  - etc.

Attribute description

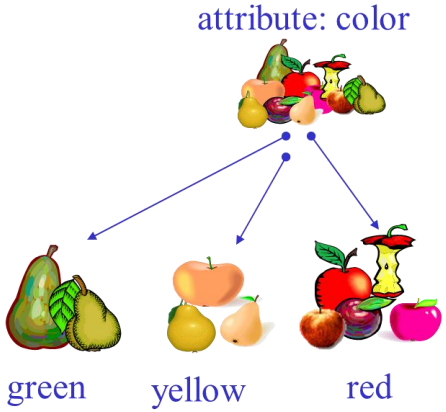


color	weight	shape	size	sort
red	12	round	middle	apple
yellow	20	conic	large	pear
red	15	round	tiny	apple
green	8	round	small	pear
yellow	22	conic	large	apple
mixed	12	conic	small	apple
green	15	round	middle	apple
mixed	8	round	tiny	apple
yellow	6	round	small	pear

- nominal attributes: ordered and unordered
- numeric attributes

Feature evaluation

- in order to select attributes, we have to evaluate (rank) them
- the success of feature evaluation is measured through the success of downstream tasks, i.e. learning
- an example: feature evaluation in decision tree building
  - in each interior node of the tree an attribute is selected which determines split of the instances
  - the attributes are evaluated to ensure useful split



Three types of feature selection methods

- Filter methods: independent on learning algorithm, select the most discriminative features through a criterion based on the character of data, e.g. information gain and ReliefF

- Wrapper methods: use the intended learning algorithm to evaluate the features, e.g., progressively add features to SVM while performance increases

- Embedded method select features in the process of learning

Feature selection: Filter methods

Heuristic measures for attribute evaluation

- Impurity based
  - information theory based (information gain, gain ratio, distance measure, J-measure)
  - probability based: Gini index, DKM, classification error on the training set
  - MDL
  - statistics G, $\aleph^2$
  - mean squared and mean absolute error (MSE, MAE)
  - assume conditional independence (upon label) between the attributes
- Context sensitive measures:
  - Relief, Contextual Merit,
  - random forests or boosting based attribute evaluation,

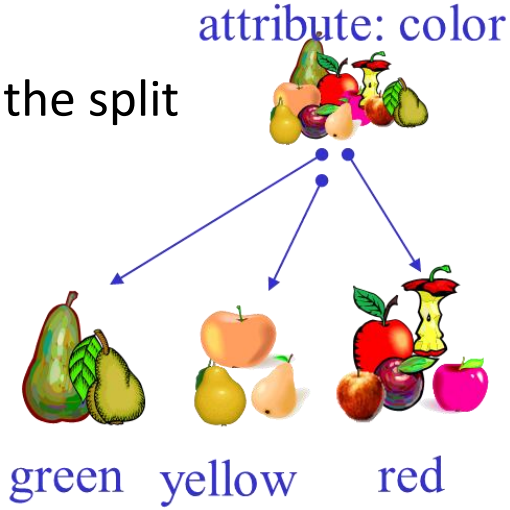
Information gain

- measure the purity of label distribution before and after the split
- impurity = entropy

$$I(\tau) = -\sum_{i=1}^{c} p(\tau_i) \log_2 p\,(\tau_i)$$

$$I(\tau|A) = -\sum_{j=1}^{v_A} p(v_j) \sum_{i=1}^{c} p(\tau_i|v_j) \log_2 p\,(\tau_i|v_j)$$

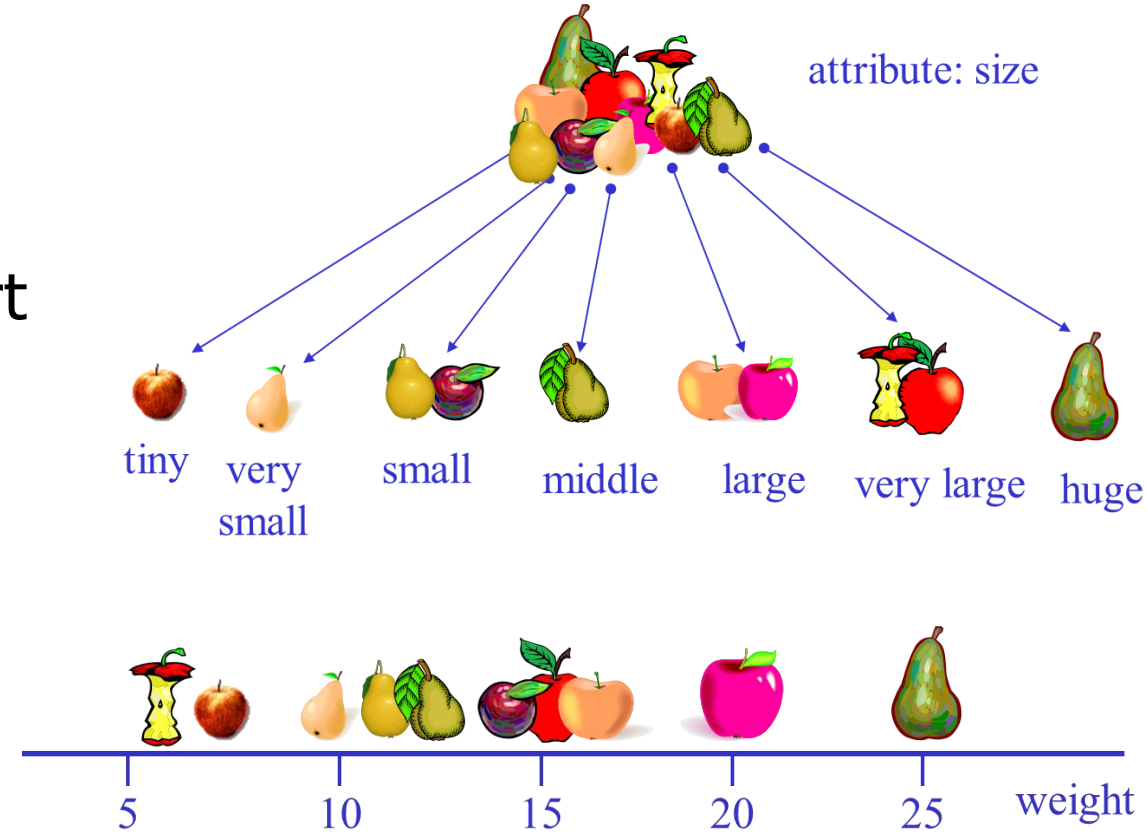
$$IG(A) = I(\tau) - I(\tau|A)$$



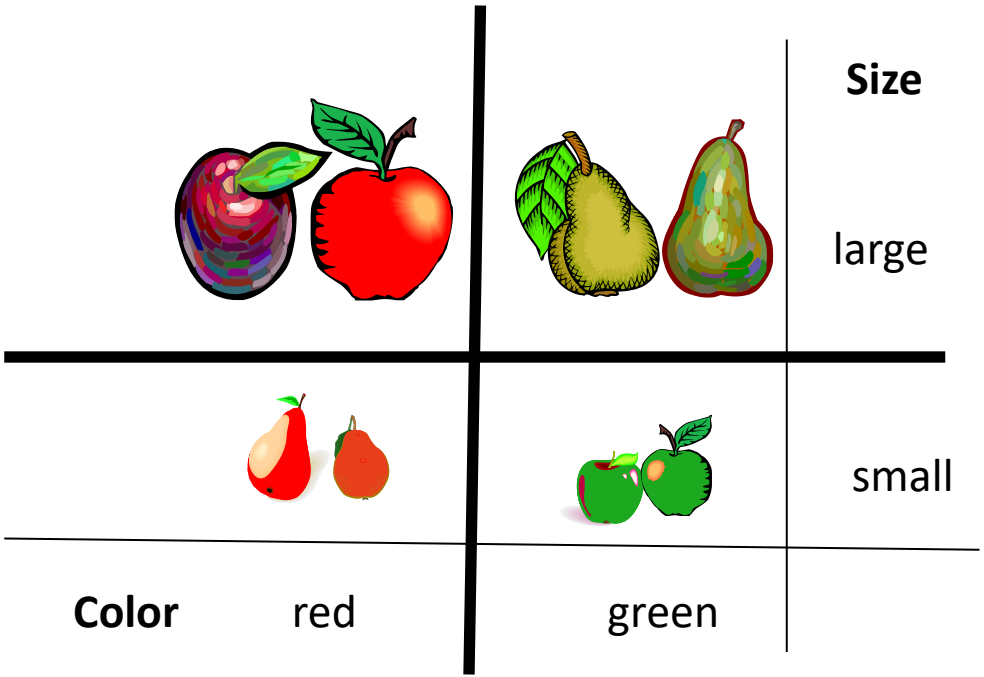
- each attribute is evaluated independently from others

Multivalued and numeric attributes

- multivalued: insufficient statistical support in certain splits
- numeric: sometimes requires prior discretization

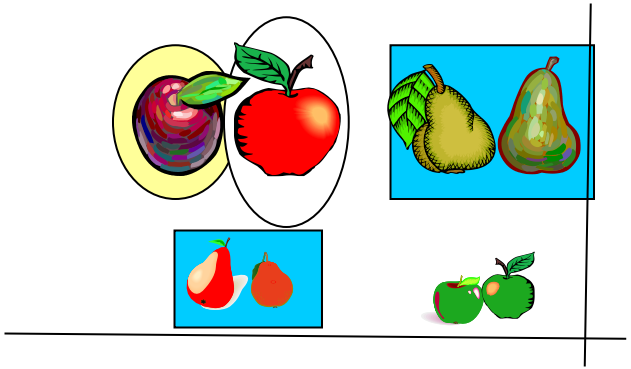


Attribute interactions



Relief algorithms

- criterion: evaluate attribute according to its power of separation between near instances



- values of good attribute should distinguish between near instances from different class and have similar values for near instances from the same class

Relief algorithms

- no assumption of conditional independence
- context sensitive
- reliable also in problems with strong conditional dependencies
- included in several machine learning systems (e.g., Weka, Orange, scikit-learn, R)
    - Relief (Kira in Rendell, 1992): two class classification
    - ReliefF (Kononenko, 1994): multi-class classification
    - RReliefF (Robnik Šikonja in Kononenko, 1997): regression

Marko Robnik-Šikonja, Igor Kononenko: Theoretical and Empirical Analysis of ReliefF and RReliefF. *Machine Learning Journal*, 53:23-69, 2003

Algorithm Relief

Input: set of instances $<x_i, \tau_i>$
Output: the vector W of attributes' evaluations

```
set all weights W[A] := 0.0;
for i := 1 to m do begin
  randomly select an instance R;
  find nearest hit H and nearest miss M;
  for A := 1 to #all_attributes do
    W[A] := W[A] - diff(A,R,H)/m + diff(A,R,M)/m;
end;
```

Function diff

- **for nominal attributes**

$$\mathrm{diff}(A, I_1, I_2) = \begin{cases} 0; \mathrm{value}(A, I_1) = \mathrm{value}(A, I_2) \\ 1; otherwise \end{cases}$$

- **for numerical attributes**

$$\mathrm{diff}(A, I_1, I_2) = \frac{|\mathrm{value}(A, I_1) - \mathrm{value}(A, I_2)|}{\max(A) - \min(A)}$$

- **distance between two instances**

$$\delta(\mathrm{I}_1, \mathrm{I}_2) = \sum_{i=1}^{a} \mathrm{diff}(i, I_1, I_2)$$

- **unknown values of attributes**

Extension: ReliefF

- For multi-class problems
- Handles incomplete and noisy data
- More robust: uses k nearest instances from all the classes

The algorithm ReliefF

Input: set of instances $<x_i, \tau_i>$
Output: the vector W of attributes' evaluations

for v:=1 to a do $W_v := 0.0$;
for i := 1 **to** m **do begin**
  randomly select an instance R_i
  find k nearest hits H
  for each class $t \neq R_{i,\tau}$ **do**
      from class t find k nearest misses M(t)
  for v := 1 to a **do**
    update W_v according to update formula
end;

Update formula

$$W_v = W_v - \frac{1}{m}\mathrm{con}(A_v, R_i, H) + \frac{1}{m}\sum_{\substack{t=1 \\ t \neq R_{i,\tau}}}^{c} \frac{p(\tau_t)\,\mathrm{con}(A_v, R_i, M(t))}{1 - p(R_{i,\tau})}$$

$$\mathrm{con}(A_v, R_i, S) = \frac{1}{k}\sum_{j=1}^{k} \mathrm{diff}(A_v, R_i, S_j)$$

In regression: RReliefF

$$\mathrm{W[A]} := \mathrm{W[A]} - \mathrm{diff(A,R,H)/m} + \mathrm{diff(A,R,M)/m};$$

$$W[A] = P(\text{different value of } A | \text{nearest instances with different prediction}) - P(\text{different value of } A | \text{nearest instances with same prediction})$$

$$\mathrm{W}[A] = P_{dA|dC} - P_{dA|\neg dC}$$

- after applying the Bayesian rule: P(A|B) = P(A)P(B|A)/P(B)

$$W[A] = \quad = \frac{P_{dC|dA}P_{dA}}{P_{dC}} - \frac{(1 - P_{dC|dA})P_{dA}}{1 - P_{dC}}$$

- we approximate this formula
- unified view on attribute evaluation in classification and regression

Marko Robnik-Šikonja, Igor Kononenko: An adaptation of Relief for attribute estimation in regression. *Machine Learning, Proceedings of ICML 1997*

Feature selection: Embedded methods

Regularization for feature selection

- feature selection as part of learning (embedded method)
- loss function is composed of two components: prediction error and number/weight of included features

$$L(X, Y, f) = \sum_{i=1}^{n} I(y_i \neq f(x_i)) + \lambda \sum_{j=1}^{a} I(A_j \in X)$$

- in regression we get similar expressions for ridge regression and lasso

Ridge regression

- Ordinary Least Squares (OLS) estimates βs by minimizing

$$\mathrm{RSS} = \sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_{j=1}^{p} \beta_j x_{ij} \right)^2 .$$

- Ridge regression minimizes a slightly different equation

$$\sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_{j=1}^{p} \beta_j x_{ij} \right)^2 + \lambda \sum_{j=1}^{p} \beta_j^2 = \mathrm{RSS} + \lambda \sum_{j=1}^{p} \beta_j^2,$$

Ridge regression adds a penalty on βs !

- The effect of this equation is to add a penalty of the form

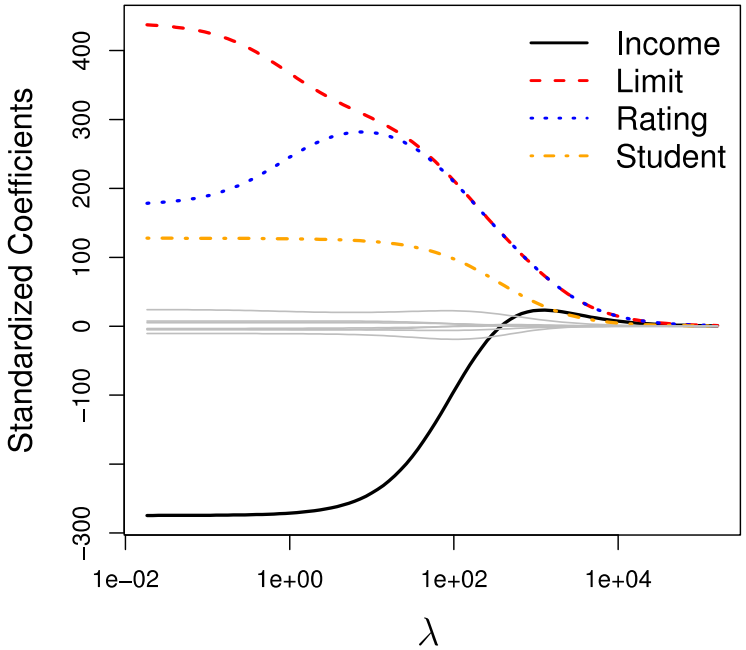
$$\lambda \sum_{j=1}^{p} \beta_j^2,$$

where the tuning parameter λ is a positive value.

- This has the effect of “shrinking” large values of βs towards zero.
- It turns out that such a constraint should improve the fit, because shrinking the coefficients can significantly reduce their variance
- Notice that when λ = 0, we get the OLS!

Credit data: ridge regression

- As λ increases, the standardized coefficients shrink towards zero.



Why can shrinking towards zero be a good thing?

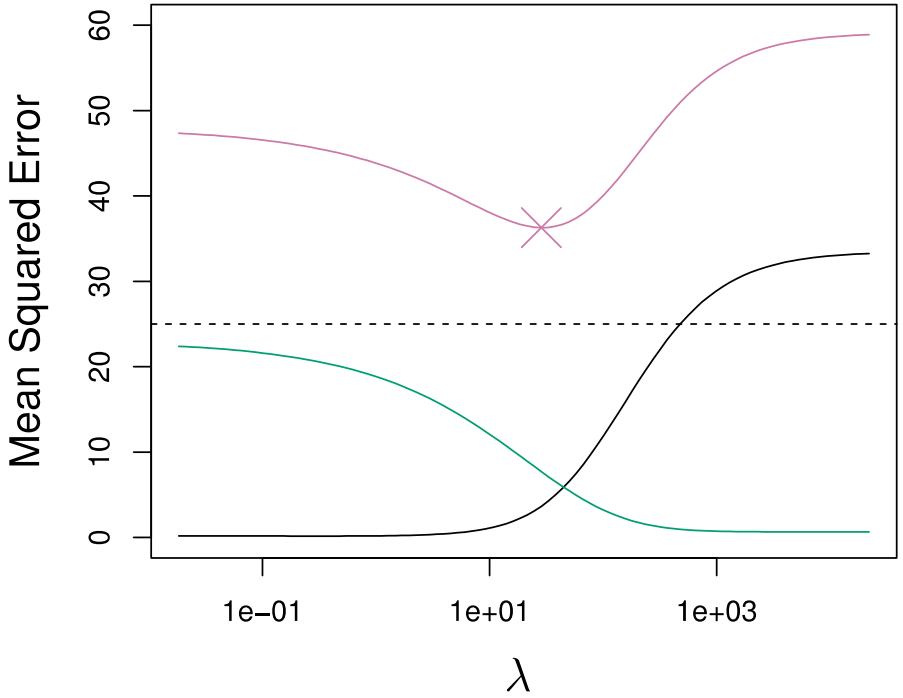
- It turns out that the OLS estimates generally have low bias but can be highly variable. In particular when n and p are of similar size or when $n < p$, then the OLS estimates will be extremely variable.

- The penalty term makes the ridge regression estimates biased but can also substantially reduce variance

- Thus, there is a bias/variance trade-off

Ridge regression bias / variance

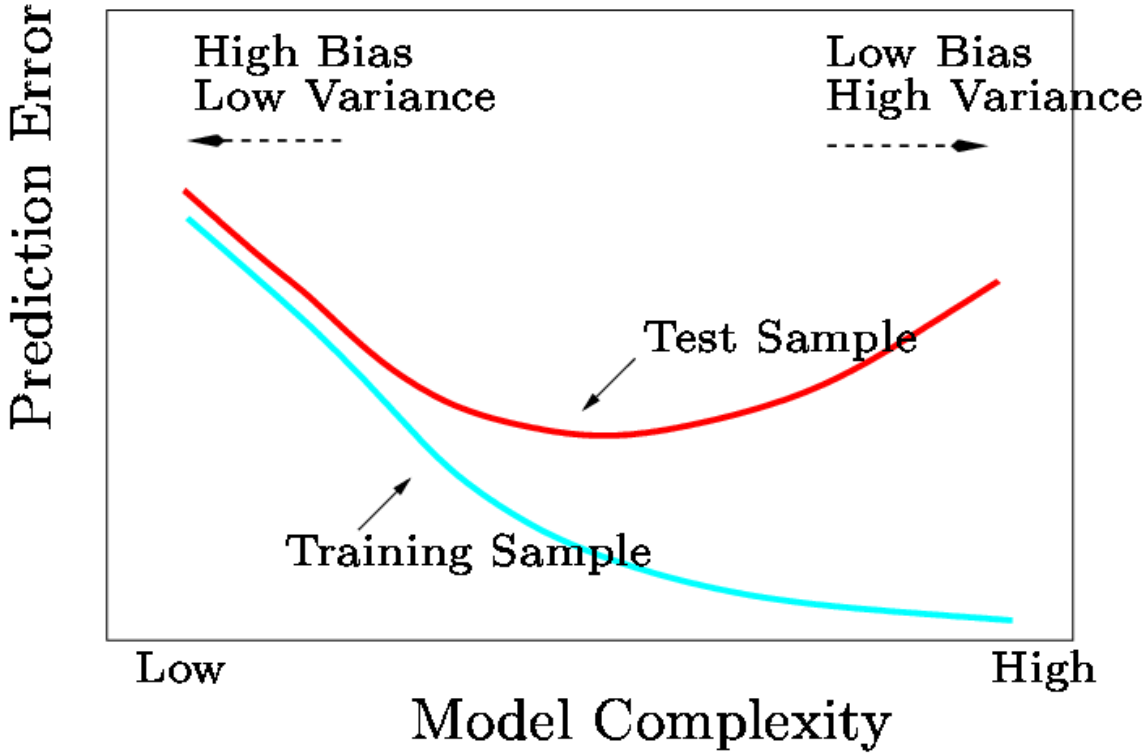
- Black: Bias
- Green: Variance
- Purple: MSE
- Increase of λ increases bias but decreases variance



Bias / variance trade-off

- In general, the ridge regression estimates will be more biased than the OLS ones but have lower variance

- Ridge regression will work best in situations where the OLS estimates have high variance



Computational advantages of ridge regression

- If number of features *p* is large, then using the best subset selection approach requires searching through enormous numbers of possible models

- With ridge regression, for any given λ, we only need to fit one model and the computations turn out to be very simple

- Ridge regression can even be used when $p > n$, a situation where OLS fails completely!

The LASSO method

- Ridge regression isn’t perfect
- One significant problem is that the penalty term will never force any of the coefficients to be exactly zero. Thus, the final model will include all variables, which makes it harder to interpret
- A more modern alternative is the LASSO
- The LASSO works in a similar way to ridge regression, except it uses a different penalty term

LASSO's Penalty Term

- Ridge Regression minimizes

$$\sum_{i=1}^{n}\left(y_i-\beta_0-\sum_{j=1}^{p}\beta_j x_{ij}\right)^2+\lambda\sum_{j=1}^{p}\beta_j^2=\mathrm{RSS}+\lambda\sum_{j=1}^{p}\beta_j^2,$$

- The LASSO estimates the βs by minimizing the

$$\sum_{i=1}^{n}\left(y_i-\beta_0-\sum_{j=1}^{p}\beta_j x_{ij}\right)^2+\lambda\sum_{j=1}^{p}|\beta_j|=\mathrm{RSS}+\lambda\sum_{j=1}^{p}|\beta_j|.$$

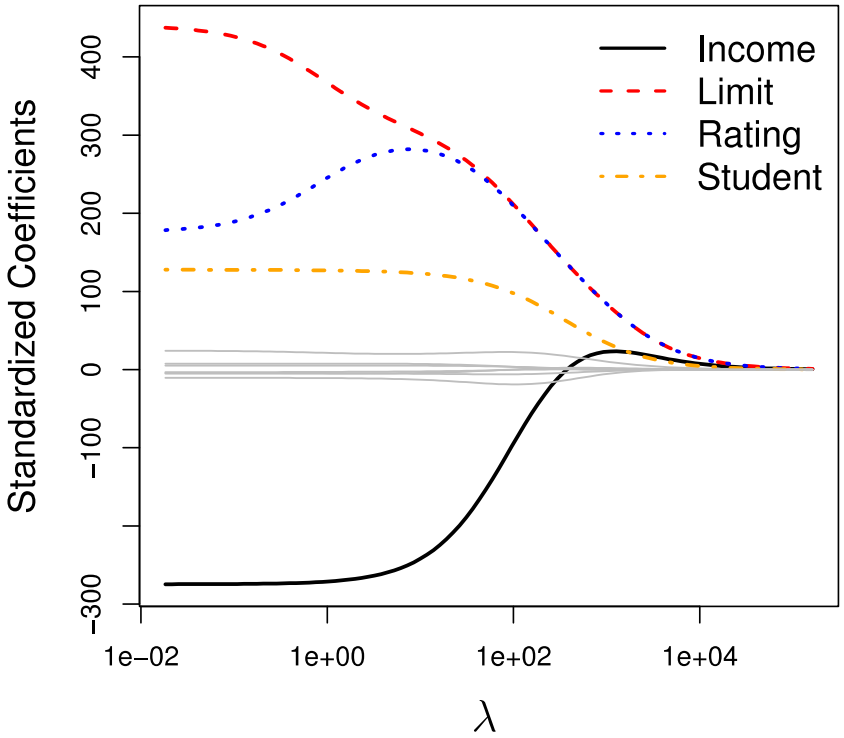
The difference between ridge regression and lasso

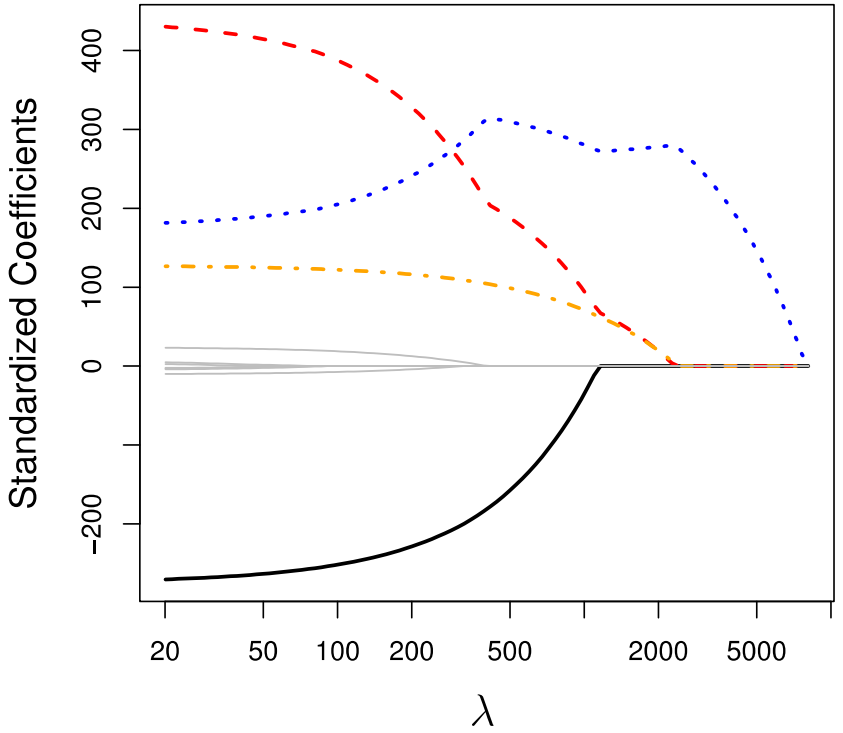
- This seems like a very similar idea but there is a big difference.

- Using LASSO penalty, it could be proven mathematically that some coefficients end up being set to exactly zero.

- With LASSO, we can produce a model that has high predictive power and it is simple to interpret.

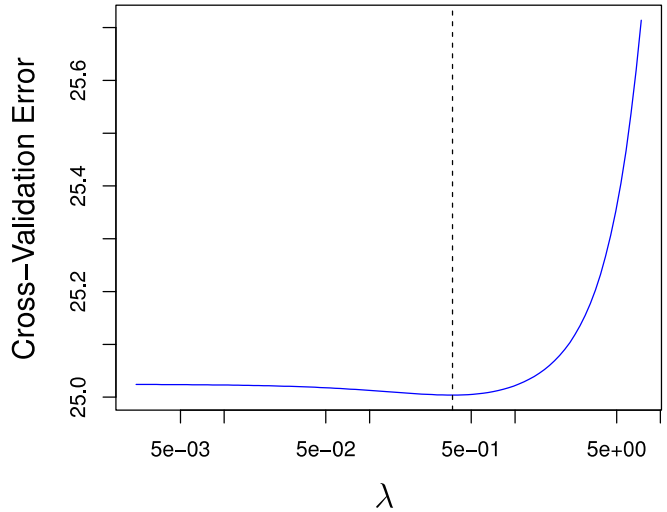
Credit data: Ridge and LASSO

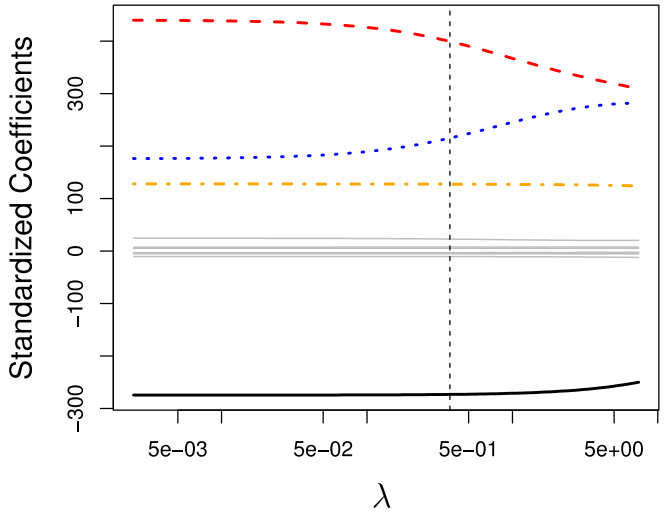




Selecting the tuning parameter λ

- We need to decide on a value for λ
- Select a grid of potential values, use cross validation to estimate the error rate on test data (for each value of λ) and select the value that gives the least error rate.





Feature selection: Wrapper methods

Wrapper approach

start with an empty set of features *S={}* // forward selection

repeat

add all unused features one by one to *S*

train a prediction model with each set *S*

evaluate each prediction model

keep the best added feature in *S*

until all features are added to *S*

return the best set of features encountered

- high computational load but effective for a given learning model; attention to data overfitting
- how would backward selection differ?

Model evaluation

Model evaluation metrics

- Evaluation metrics: How can we measure accuracy? Other metrics to consider?
- Regression: MSE, MAE
- Classification: accuracy, sensitivity, specificity, AUC, precision, recall
- Comparing classifiers:
  - Mean and confidence intervals
  - Cost-benefit analysis and ROC Curves
  - Rank-based tests (Friedman/Nemenyi)
  - Bayesian (hierarchical) tests

Classifier evaluation metrics: confusion matrix aka missclassification matrix

Confusion Matrix:

Actual class\Predicted class	C_1	¬ C_1
C_1	**True Positives (TP)**	**False Negatives (FN)**
¬ C_1	**False Positives (FP)**	**True Negatives (TN)**

Example of Confusion Matrix:

Actual class\Predicted class	buy_computer = yes	buy_computer = no	Total
buy_computer = yes	**6954**	**46**	7000
buy_computer = no	**412**	**2588**	3000
Total	7366	2634	10000

- Given *m* classes, an entry, $\boldsymbol{CM_{i,j}}$ in a confusion matrix indicates # of instances in class *i* that were labeled by the classifier as class *j*
- May have extra rows/columns to provide totals

Classification accuracy, error rate

A\P	C	¬C	
C	**TP**	**FN**	**P**
¬C	**FP**	**TN**	**N**
	P'	**N'**	**All**

- **Classifier Accuracy (CA),** or recognition rate: percentage of test set instances that are correctly classified

  Accuracy = (TP + TN)/All

- **Error rate:** *1 – accuracy*, or **Error rate = (FP + FN)/All**

Sensitivity and specificity

A\P	C	¬C	
C	**TP**	**FN**	**P**
¬C	**FP**	**TN**	**N**
	P'	**N'**	**All**

Class Imbalance Problem:

One class may be *rare*, e.g. fraud, or HIV-positive

Significant *majority of the negative class* and minority of the positive class

Sensitivity: True Positive recognition rate

Sensitivity = TP/P

Specificity: True Negative recognition rate

Specificity = TN/N

Precision, recall and F-measures

- **Precision**: exactness, i.e what % of instances the classifier labeled as positive are actually positive
  Precision = TP/P'

$$precision = \frac{TP}{TP + FP}$$

- **Recall:** completeness, i.e what % of positive instances did the classifier label as positive?
  Recall = TP / P (the same as sensitivity)

$$recall = \frac{TP}{TP + FN}$$

- Perfect score is 1.0
- Inverse relationship between precision & recall
- ***F* measure (*F_1* or *F*-score)**: harmonic mean of precision and recall,

$$F = \frac{2 \times precision \times recall}{precision + recall}$$

- ***F_β*:** weighted measure of precision and recall
  - assigns ß times as much weight to recall as to precision

$$F_\beta = \frac{(1 + \beta^2) \times precision \times recall}{\beta^2 \times precision + recall}$$

Example: precision and recall

Actual Class\Predicted class	cancer = yes	cancer = no	Total	Recognition(%)
cancer = yes	**90**	**210**	300	30.00 (*sensitivity)*
cancer = no	**140**	**9560**	9700	98.56 (*specificity)*
Total	230	9770	10000	96.40 (*accuracy*)

- *Precision* = 90/230 = 39.13% *Recall* = 90/300 = 30.00%

Multiclass evaluation

- no problems for classification accuracy
- most other measures assume binary class, e.g., precision, recall, F_1
- multiclass extensions simulate binary case
- macro average:
  - compute several one-versus-all scores and average
  - assumes balanced class distribution, gives equal weight to each class
- micro average
  - computes TP, FP, TN, FN for each class separately and then computes the measure
  - assumes all instances are of the same importance; in case of imbalanced classes this might be problematic

Multiclass example

- Let us compute precision $P = TP / (TP+FP)$.
- Let us assume multi-class classification system with four classes and the following numbers when tested:
- Class A: 1 TP and 1 FP
- Class B: 10 TP and 90 FP
- Class C: 1 TP and 1 FP
- Class D: 1 TP and 1 FP
- $P(A) = P(C) = P(D) = 0.5$, whereas $P(B)=0.1$.
- A macro-averaged precision: $P_{macro} = (0.5+0.1+0.5+0.5) / 4 = 0.4$
- A micro-averaged precision: $P_{micro} = (1+10+1+1) / (2+100+2+2) = 0.123$

Error depends on decision threshold

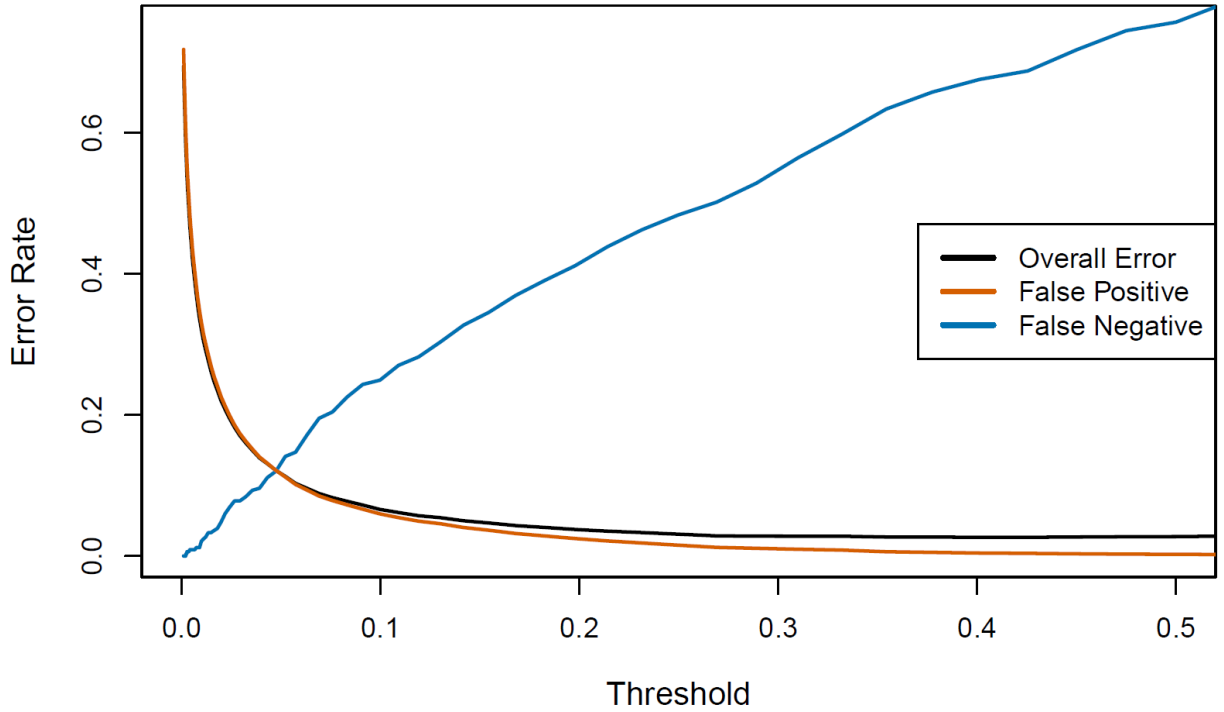
- Example: False positive and false negative rate are computed based on probabilities returned by classifier

$$P(\text{Class=True} \mid X_1, X_2, \ldots) \geq 0.5$$

- We can change the two error rates by changing the threshold from 0.5 to some other value in [0, 1]:

$$P(\text{Class=True} \mid X_1, X_2, \ldots.) \geq \text{threshold}$$

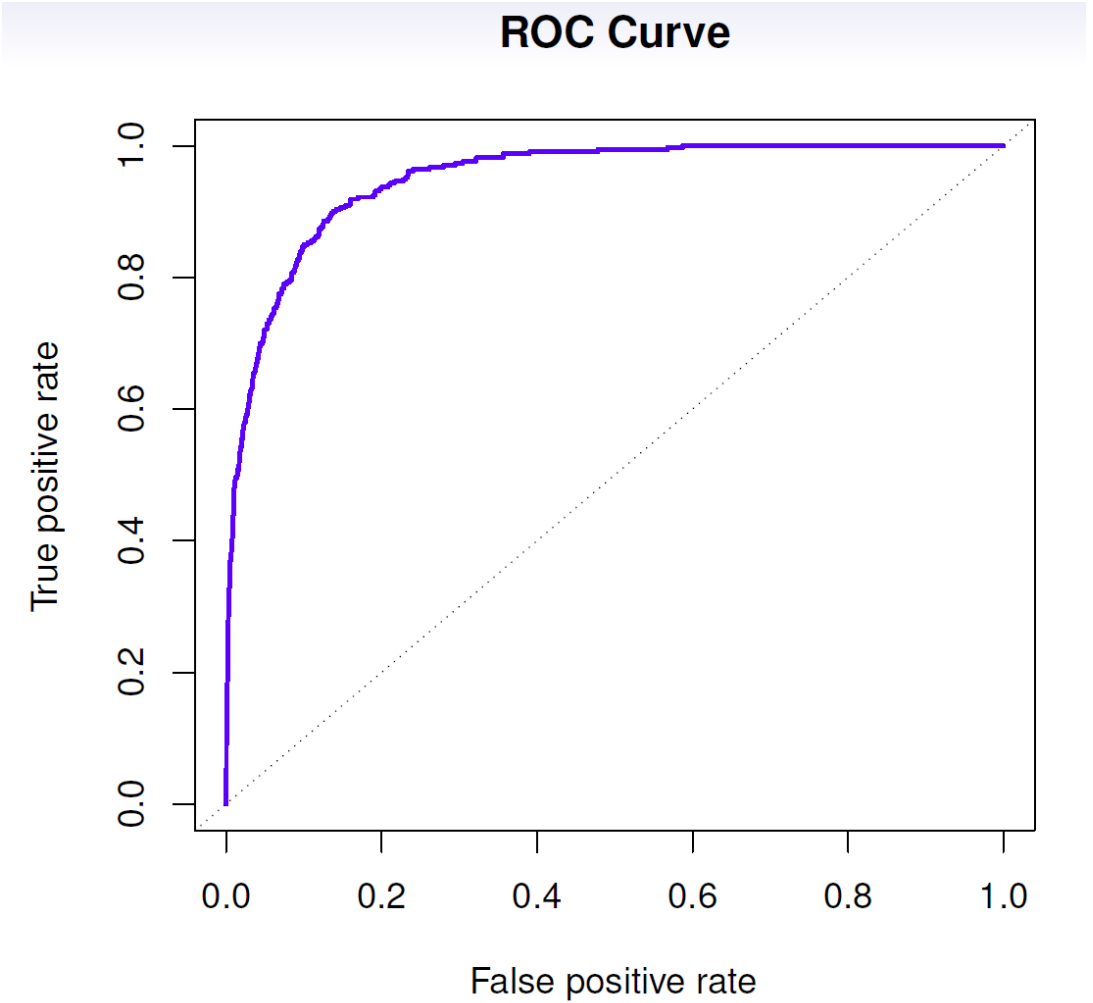
Varying the threshold



- To reduce false negative rate, we would chose threshold other than 0.5, e.g., threshold ≤ 0.1

ROC curve

- ROC curve shows both TP rate and FP rate simultaneously
- FPR = FP/(FP+TN)
- TPR = TP/(TP+FN)
- To summarize overall performance, we also use area under the ROC curve (AUC)
- The larger the AUC the better is the classifier. Why? What would be an ideal ROC curve?



Model selection

Issues affecting model selection

- **Accuracy**
  - classification: classification accuracy, AUC, F_1
  - regression: MSE, MAE
- **Speed**
  - time to construct the model (training time)
  - time to use the model (classification/prediction time)
- **Robustness**: handling noise and missing values
- **Scalability**: efficiency in disk-resident databases
- **Interpretability**
  - understanding and insight provided by the model
  - other measures, e.g., goodness of rules, such as decision tree size or compactness of classification rules

Unsupervised feature selection

- criterion: preserve similarity between instances
- Example: SPEC, spectral feature selection

Zhao Z, Liu H. Spectral feature selection for supervised and unsupervised learning. In Proceedings of ICML 2007, pp. 1151-1157.

Semi-supervised feature selection

- typically a small sample of labelled and a large sample of unlabeled data is available
- principle: use the label information of labeled data and data distribution or local structure of both labeled and unlabeled data to evaluate feature relevance

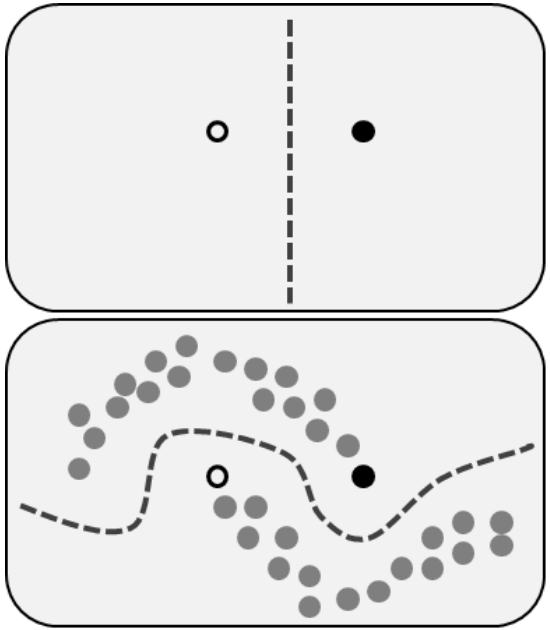
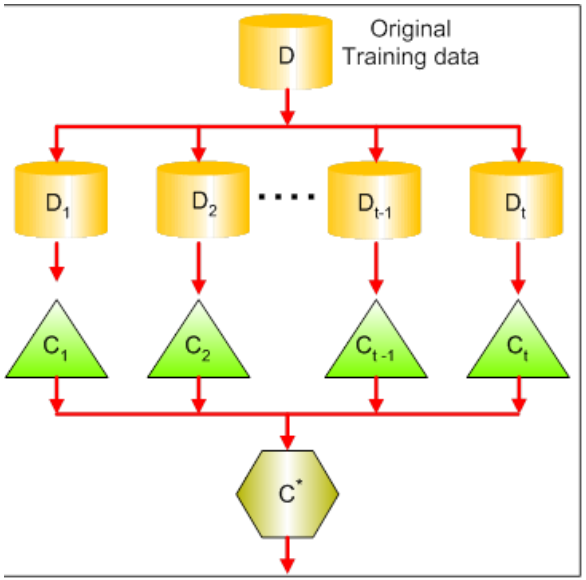


image by Techerin, Wikipedia

Cheng, H., Deng, W., Fu, C., Wang, Y. and Qin, Z., 2011. Graph-based semi-supervised feature selection with application to automatic spam image identification. In Computer Science for Environmental Engineering and EcoInformatics (pp. 259-264).

Stability of feature selection

- for high dimensional small sample data, the stability of feature selection is a pressing issue, e.g., in microarray data, we might get similar classification accuracy with different sets of features
- Solution: **ensemble approach:**
  1. produce diverse feature sets
     - different feature selection techniques,
     - instance-level perturbation
     - feature-level perturbation
     - stochasticity in the feature selector,
     - Bayesian model averaging
     - combinations of the above techniques
  2. aggregate them
     - weighted voting
     - counting



Dimensionality reduction

Feature Selection vs Feature Extraction

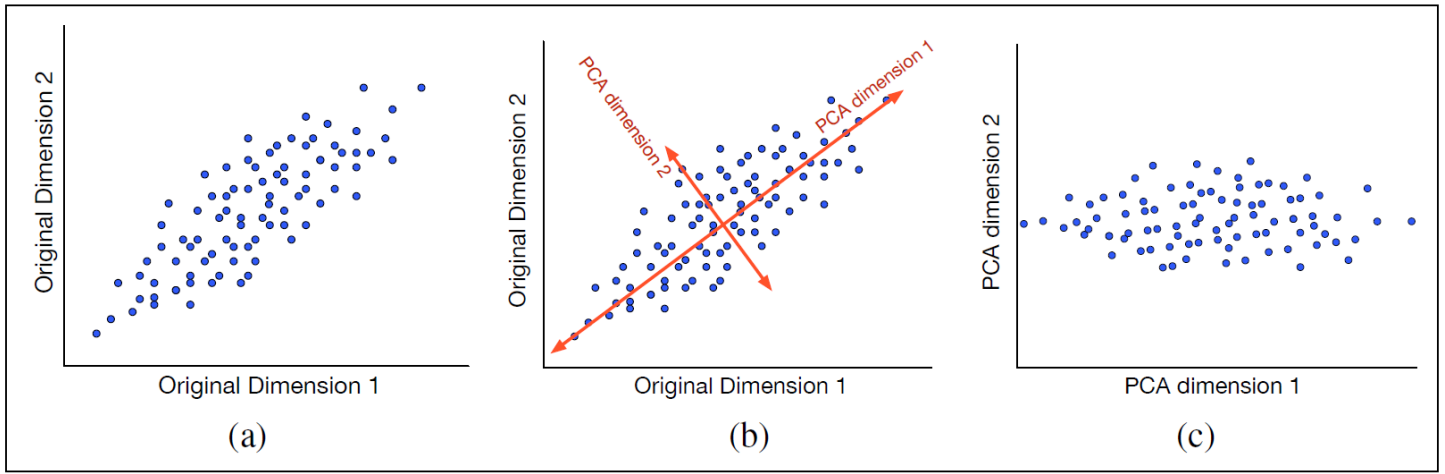
- Feature selection: Choosing $k<d$ important features, ignoring the remaining $d - k$
  - Subset selection algorithms (Filter, Wrapper, Embedded)
- Feature extraction: Project the original x_i , i =1,...,d dimensions to new $k<d$ dimensions, z_j , j =1,...,k
  - Typical statistical techniques: Principal components analysis (PCA), linear discriminant analysis (LDA), factor analysis (FA)

Feature reduction

- approximation of p-dimensional space of matrix X with lower dimensional space
- also called feature extraction
- Simplest variant: Linear transformation, i.e. rotation in the direction of the largest variance

Principle components analysis

- principle components analysis, PCA
- we iteratively find the orthogonal axes of the largest variance
- we use the new dimensions to approximate the original space



Principal Components Analysis (PCA)

- Find a low-dimensional space such that when $\mathbf{x}$ is projected there, information loss is minimized.
- The projection of $\mathbf{x}$ on the direction of $\mathbf{w}$ is: $z = \mathbf{w}^T\mathbf{x}$
- Find $\mathbf{w}$ such that Var(z) is maximized

$$\begin{aligned}\text{Var}(z) &= \text{Var}(\mathbf{w}^T\mathbf{x}) = E[(\mathbf{w}^T\mathbf{x} - \mathbf{w}^T\boldsymbol{\mu})^2] \\ &= E[(\mathbf{w}^T\mathbf{x} - \mathbf{w}^T\boldsymbol{\mu})(\mathbf{w}^T\mathbf{x} - \mathbf{w}^T\boldsymbol{\mu})] \\ &= E[\mathbf{w}^T(\mathbf{x} - \boldsymbol{\mu})(\mathbf{x} - \boldsymbol{\mu})^T\mathbf{w}] \\ &= \mathbf{w}^T E[(\mathbf{x} - \boldsymbol{\mu})(\mathbf{x} - \boldsymbol{\mu})^T]\mathbf{w} = \mathbf{w}^T \sum \mathbf{w}\end{aligned}$$

where $\text{Var}(\mathbf{x}) = E[(\mathbf{x} - \boldsymbol{\mu})(\mathbf{x} - \boldsymbol{\mu})^T] = \sum$

- Maximize Var(z) subject to $||\boldsymbol{w}||=1$

$$\max_{\mathbf{w}_1} \mathbf{w}_1^T \Sigma \mathbf{w}_1 - \alpha\left(\mathbf{w}_1^T \mathbf{w}_1 - 1\right)$$

$\sum \boldsymbol{w}_1 = a\boldsymbol{w}_1$ that is, $\boldsymbol{w}_1$ is an eigenvector of $\sum$

Choose the one with the largest eigenvalue for Var(z) to be max

- Second principal component: Max Var(z_2), s.t., $||\boldsymbol{w}_2||=1$ and orthogonal to $\boldsymbol{w}_1$ $\max_{\mathbf{w}_2} \mathbf{w}_2^T \Sigma \mathbf{w}_2 - \alpha\left(\mathbf{w}_2^T \mathbf{w}_2 - 1\right) - \beta\left(\mathbf{w}_2^T \mathbf{w}_1 - 0\right)$

$\sum \boldsymbol{w}_2 = a\,\boldsymbol{w}_2$ that is, $\boldsymbol{w}_2$ is another eigenvector of $\sum$

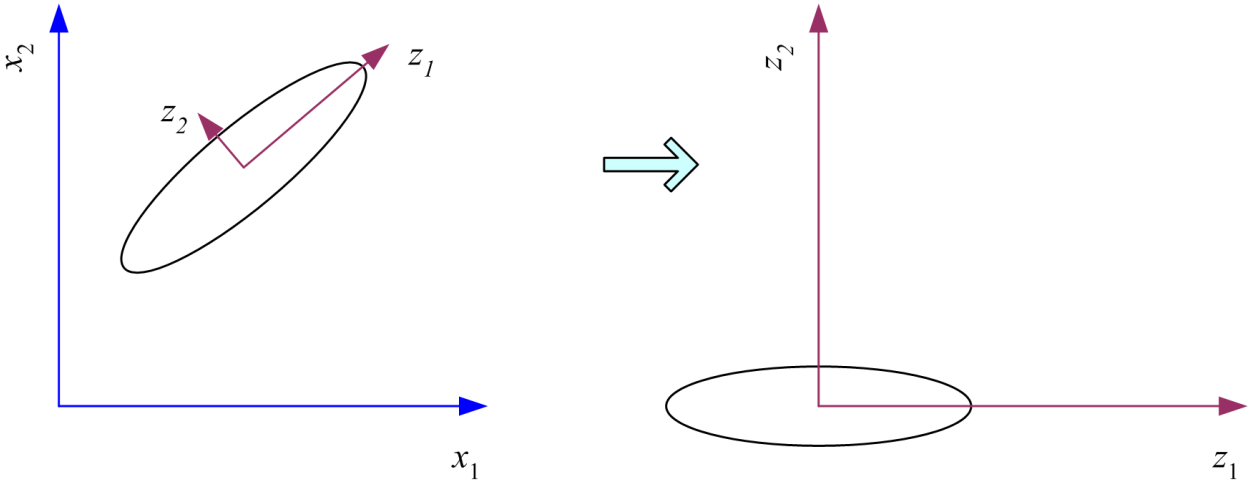
and so on.

What PCA does

$$\mathbf{z} = \mathbf{W}^T(\mathbf{x} - \boldsymbol{m})$$

where the columns of **W** are the eigenvectors of ∑, and ***m*** is the sample mean

Centers the data at the origin and rotates the axes

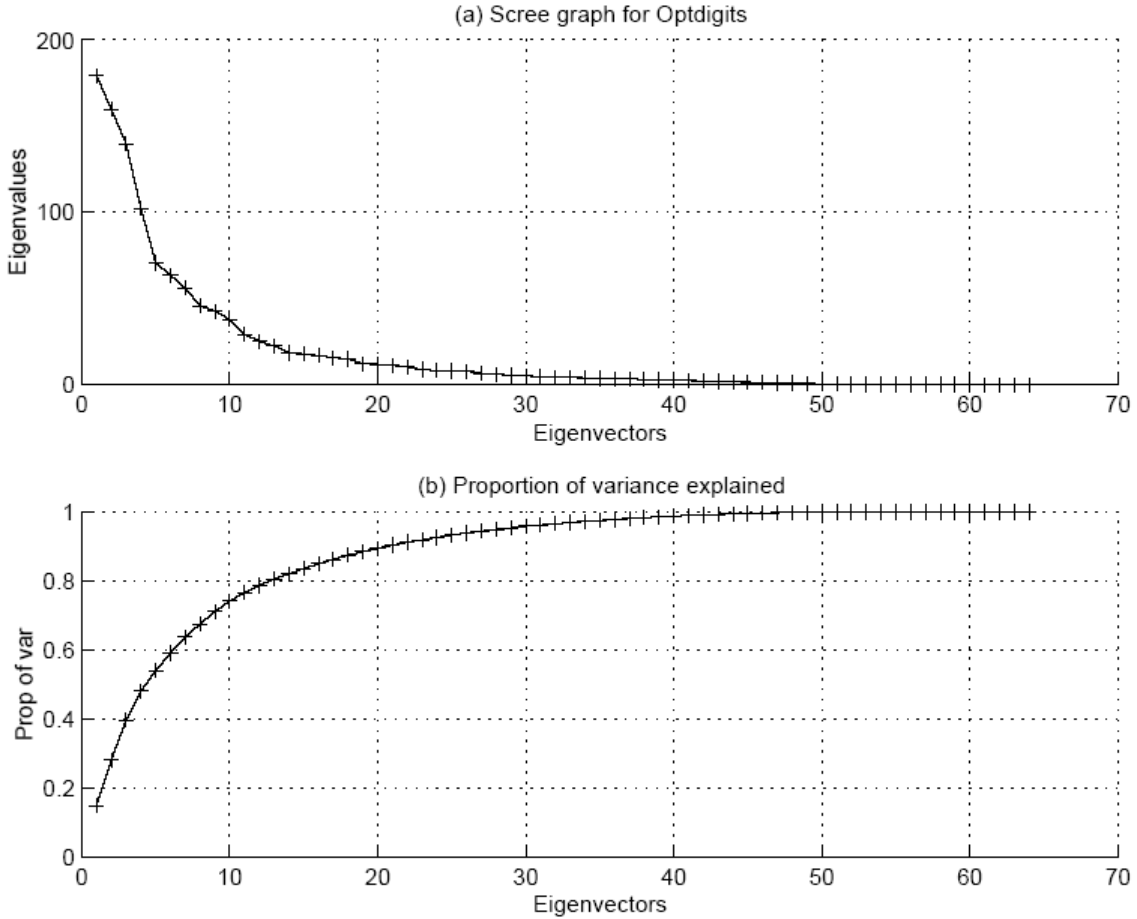


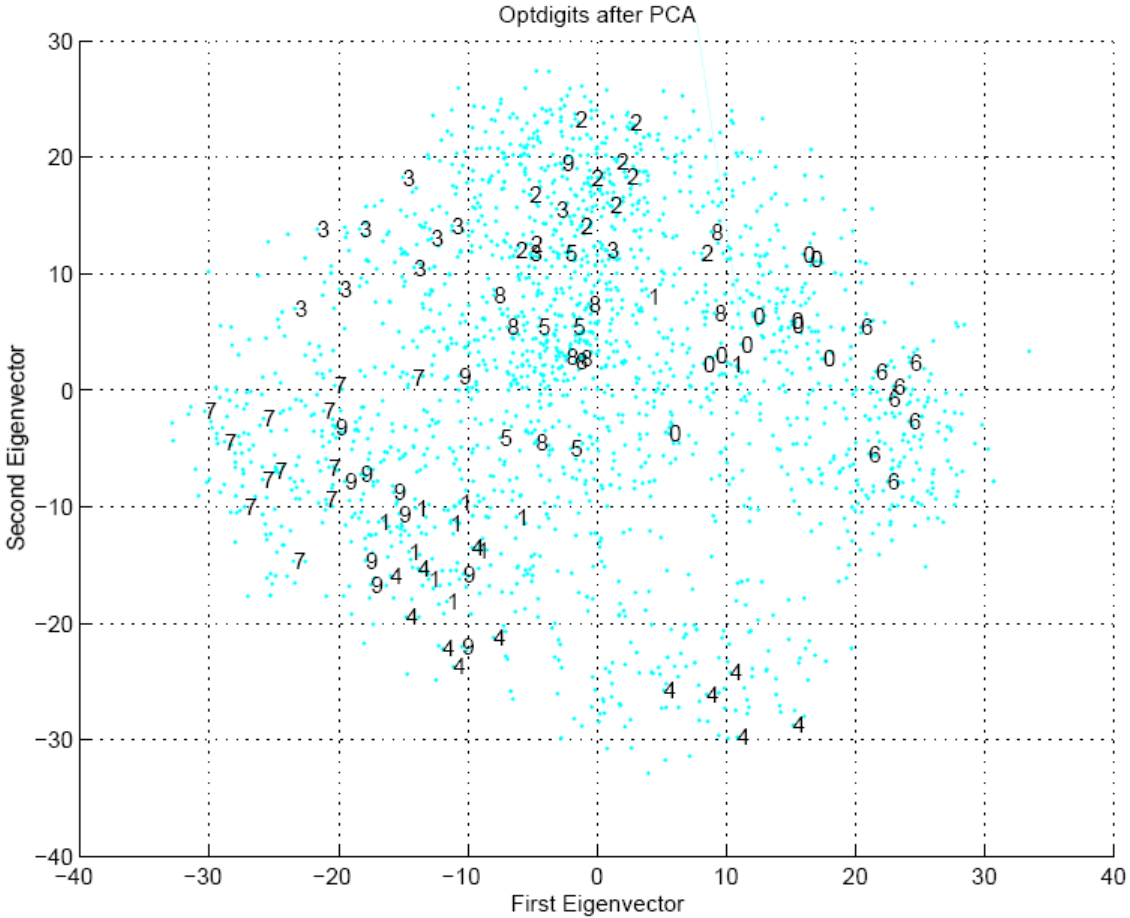
How to choose k ?

- Proportion of Variance (PoV) explained

$$\frac{\lambda_1 + \lambda_2 + \cdots + \lambda_k}{\lambda_1 + \lambda_2 + \cdots + \lambda_k + \cdots + \lambda_d}$$

  when λ_i are sorted in descending order
- Typically, stop at PoV>0.9
- Scree graph plots of PoV vs *k*, stop at "elbow"


(a) Scree graph for Optdigits
Eigenvalues
Eigenvectors
0
100
200
0
10
20
30
40
50
60
70
(b) Proportion of variance explained
Prop of var
Eigenvectors
0
0.2
0.4
0.6
0.8
1
0
10
20
30
40
50
60
70


Optdigits after PCA
Second Eigenvector
First Eigenvector
30
20
10
0
−10
−20
−30
−40
−40
−30
−20
−10
0
10
20
30
40

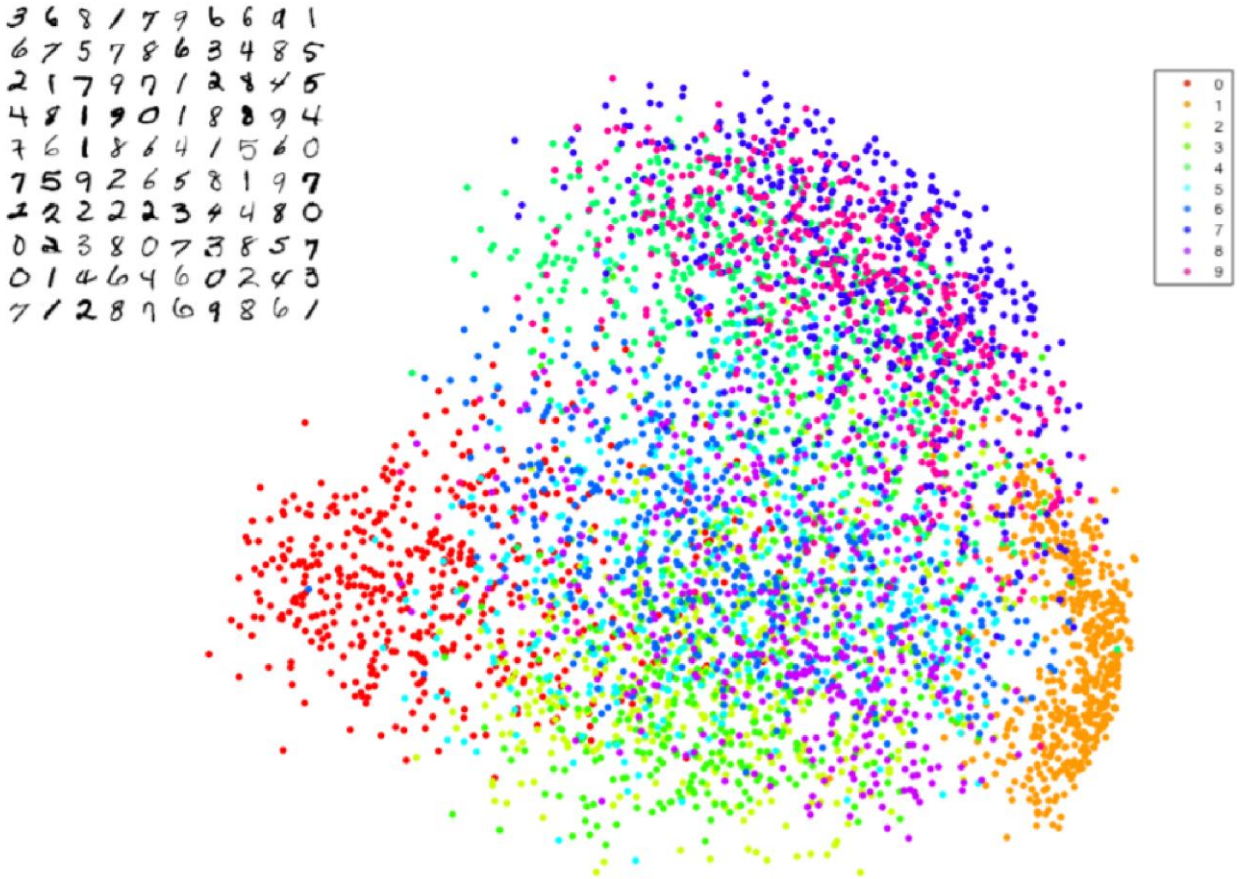
Neighbourhood preserving dimensionality reduction

- also called local embeddings
- SNE - Stochastic Neighbor Embedding
- t-SNE (t-distributed SNE)

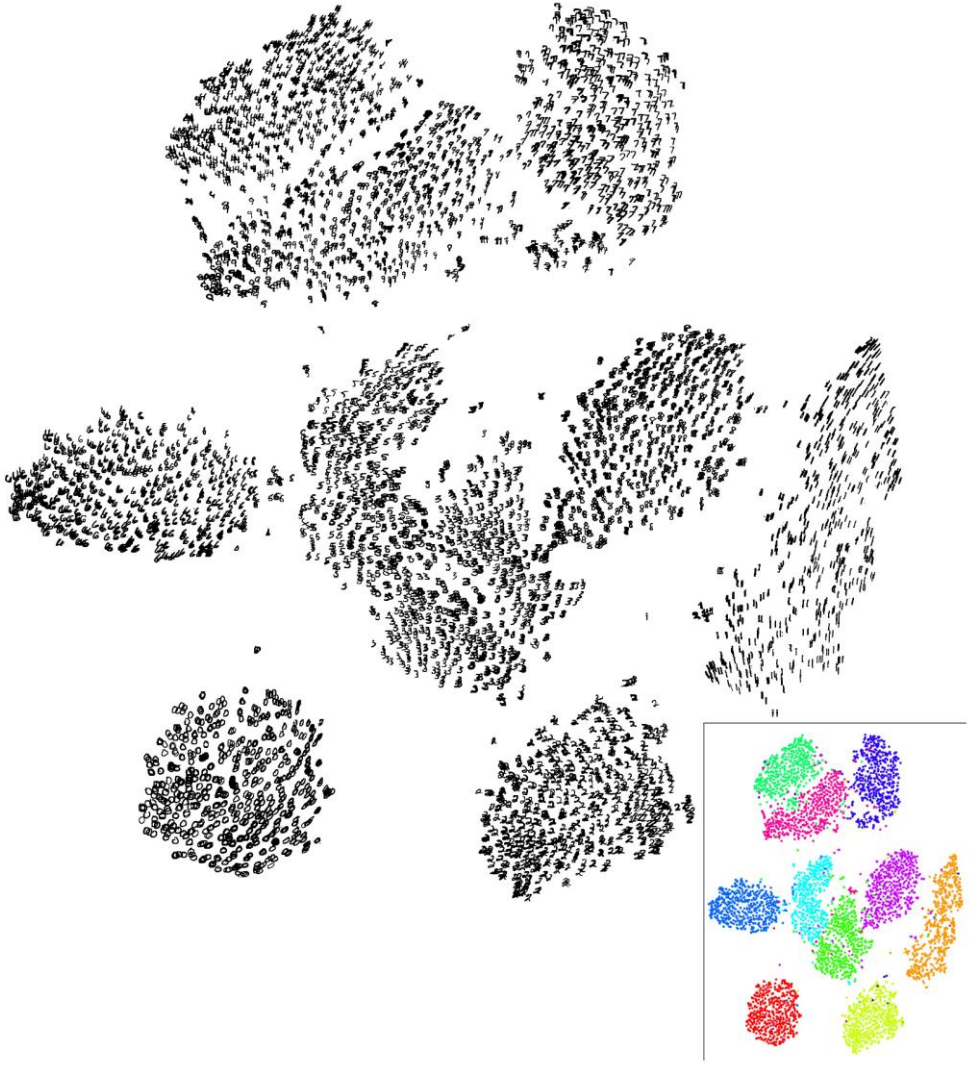
Linear and local embedding

- PCA tries to find a global structure
  - Low dimensional subspace
  - Can lead to local inconsistencies
  - Far away point can become nearest neighbors
- t-SNE is an alternative dimensionality reduction algorithm.
- t-SNE tries to preserve local structure
  - Low dimensional neighborhood should be the same as the original neighborhood.
  - Unlike PCA almost only used for visualization
  - No easy way to embed new points

PCA 2d projection of MNIST dataset



t-SNE 2d projection of MNIST dataset



Stochastic Neighbor Embedding (SNE)

- SNE basic idea:
  - ”Encode” high-dimensional neighborhood information as a distribution
  - Intuition: Random walk between data points.
    - High probability of jumping to a close point
- Find low dimensional points such that their neighborhood distribution is similar.
- How do you measure the distance between distributions?
- Most common measure: KL divergence

Neighborhood Distributions

- Consider the neighborhood around an input data point $\mathbf{x}_i \in R^d$
- Imagine that we have a Gaussian distribution centered around $\mathbf{x}_i$
- Then the probability that $\mathbf{x}_i$ chooses some other data point $\mathbf{x}_j$ as its neighbor is in proportion with the density under this Gaussian
- A point closer to $\mathbf{x}_i$ will be more likely than one further away

SNE objective

- Given $\mathbf{x_1}, .., \mathbf{x_n} \in R^D$ we define the distribution of distances between points P_{ij}
- Goal: Find good embedding $\mathbf{y_1}, .., \mathbf{y_n} \in R^d$ for some $d < D$ (normally 2 or 3)
- How do we measure an embedding quality?
- For points $\mathbf{y_1}, .., \mathbf{y_n} \in R^d$ we can define distribution Q_{ij} similarly to P_{ij}

$$Q_{ij} = \frac{\exp\left(-||\mathbf{y}^{(i)} - \mathbf{y}^{(j)}||^2\right)}{\sum_k \sum_{l \neq k} \exp\left(-||\mathbf{y}^{(l)} - \mathbf{y}^{(k)}||^2\right)}$$

- Optimize Q to be close to P
  - Minimize KL-divergence
- The embeddings $\mathbf{y_1}, .., \mathbf{y_n} \in R^d$ are the parameters we are optimizing.
  - How do you embed a new point? No embedding function, but there are ways.

Kullback–Leibler divergence

- KL divergence measures distance between two distributions, P and Q:

$$KL(Q||P) = \sum_{ij} Q_{ij} \log \left(\frac{Q_{ij}}{P_{ij}} \right)$$

- Not a metric function - not symmetric!
- Code theory intuition: If we are transmitting information that is distributed according to P, then the optimal (lossless) compression will need to send on average H(P) bits.
- What happens if you expect P (and design your compression accordingly), but the actual distribution is Q?
  - will send on average H(Q) + KL(Q||P) bits
  - KL(Q||P) is the ”penalty” for using the wrong distribution

Crowding Problem

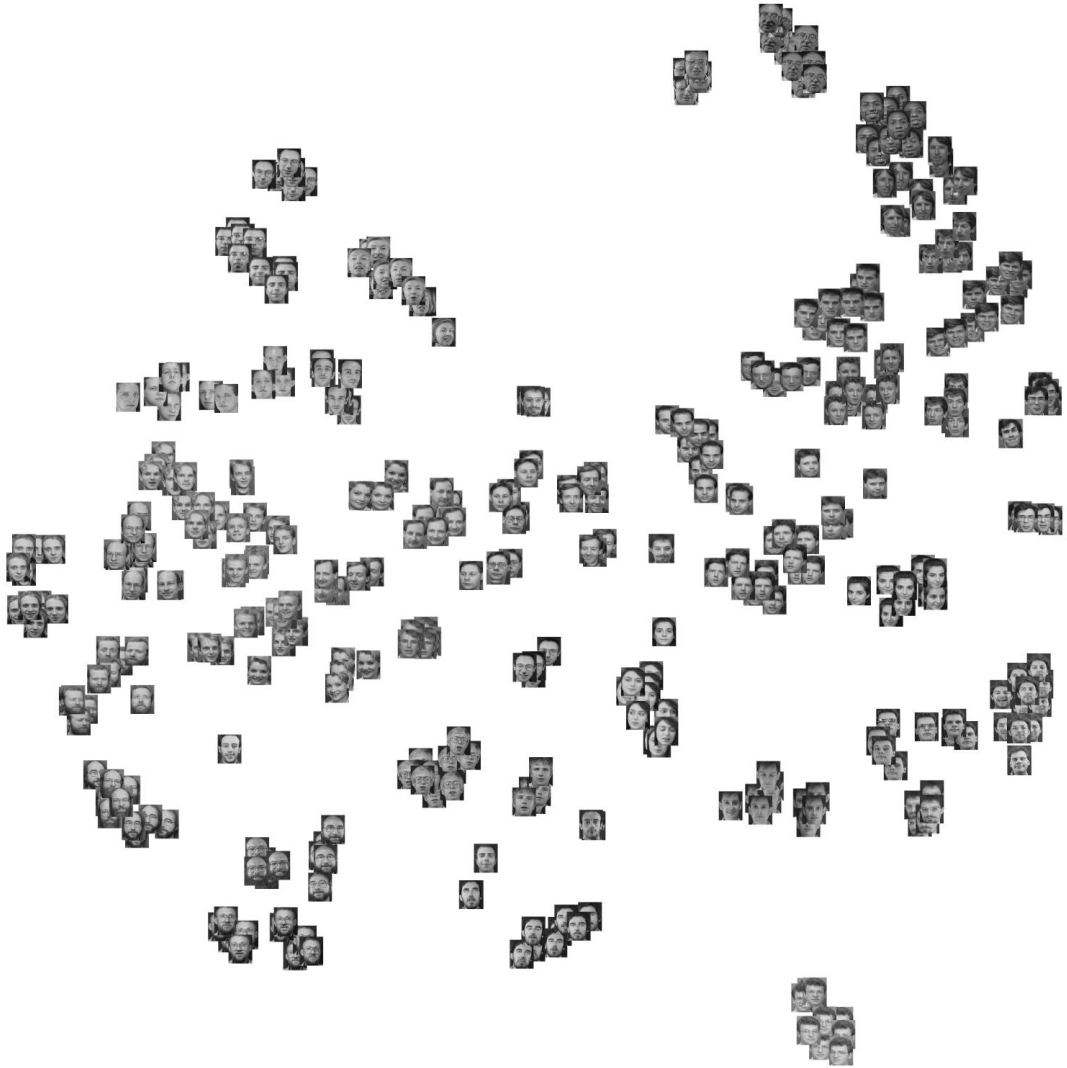
- In high dimensions, we have more room, points can have a lot of different neighbors
- In 2D, a point can have a few neighbors at distance one all far from each other - what happens when we embed in 1D?
- This is the ”crowding problem” - we don’t have enough room to accommodate all neighbors.
- This is one of the biggest problems with SNE.
- t-SNE solution: Change the Gaussian in Q to a heavy-tailed distribution.
  - if Q changes slower, we have more ”wiggle room” to place points at.

CNN features t-SNE 2d embedding



http://cs.stanford.edu/people/karpathy/cnnembed

CNN features t-SNE 2d embedding



https://lvdmaaten.github.io/tsne

t-SNE summary

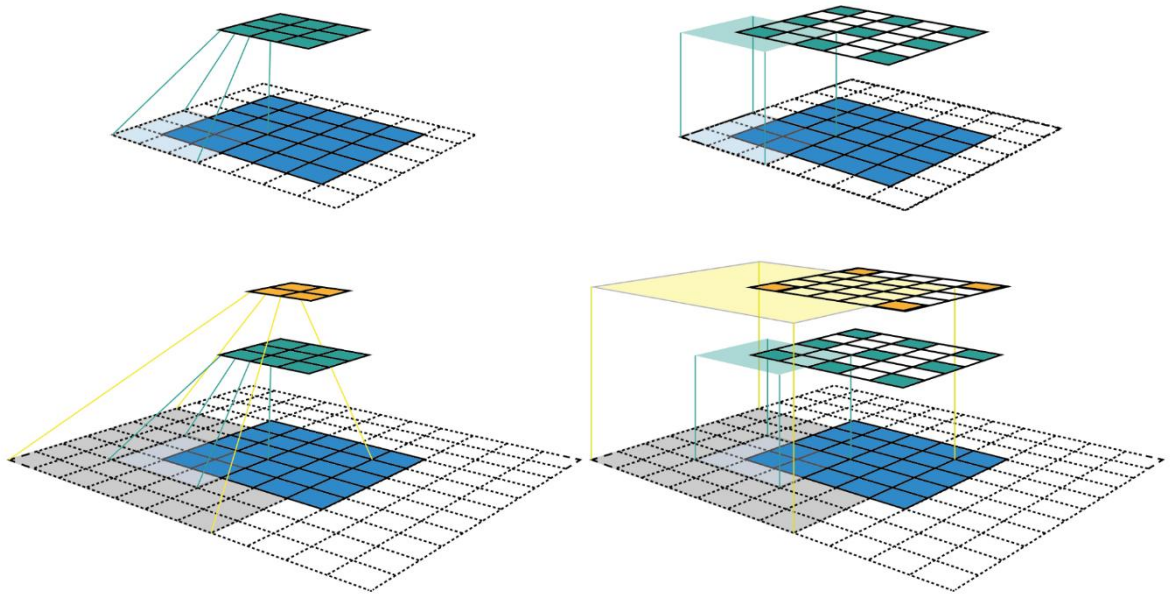
- t-SNE is a great way to visualize high-dimensional data
- Helps understand ”black-box” algorithms like DNN.
- Reduced ”crowding problem” with heavy-tailed distribution.
- Non-convex optimization - solved by gradient descent (GD) with momentum.

- Maaten, L.v.d. and Hinton, G., (2008) Visualizing data using t-SNE. Journal of Machine Learning Research, Vol 9(Nov), pp. 2579—2605, [PDF]
- Wattenberg, M., Viégas, F. and Johnson, I. (2016) How to Use t-SNE Effectively, Distil https://distill.pub/2016/misread-tsne/
- Poličar, P. OpenTSNE, https://github.com/pavlin-policar/openTSNE

University of Ljubljana, Faculty of Computer and Information Science

Neural networks





Prof Dr Marko Robnik-Šikonja

Intelligent Systems, Edition 2025

Topics overview

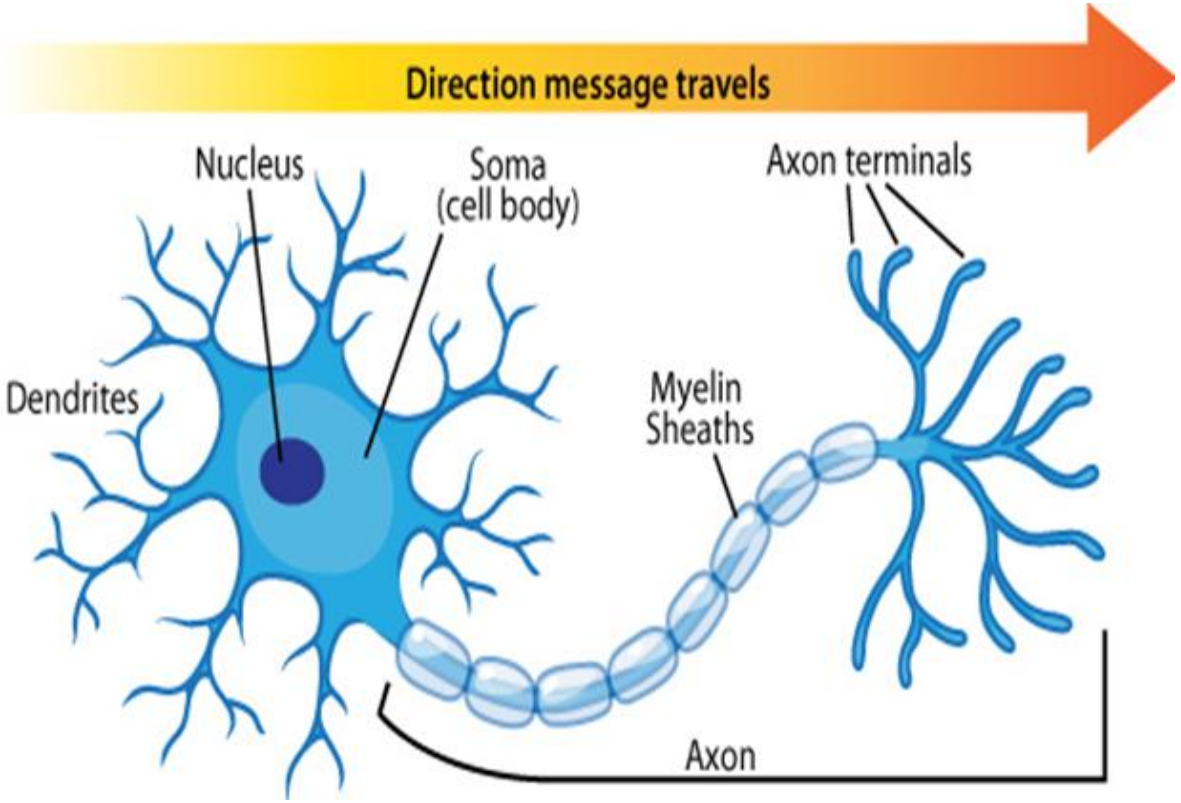
- Basics of artificial neural networks (revision)
- Backpropagation (revision)
- Deep learning
- Convolutional neural networks
- Autoencoders
- Generative adversarial networks
- Robustness

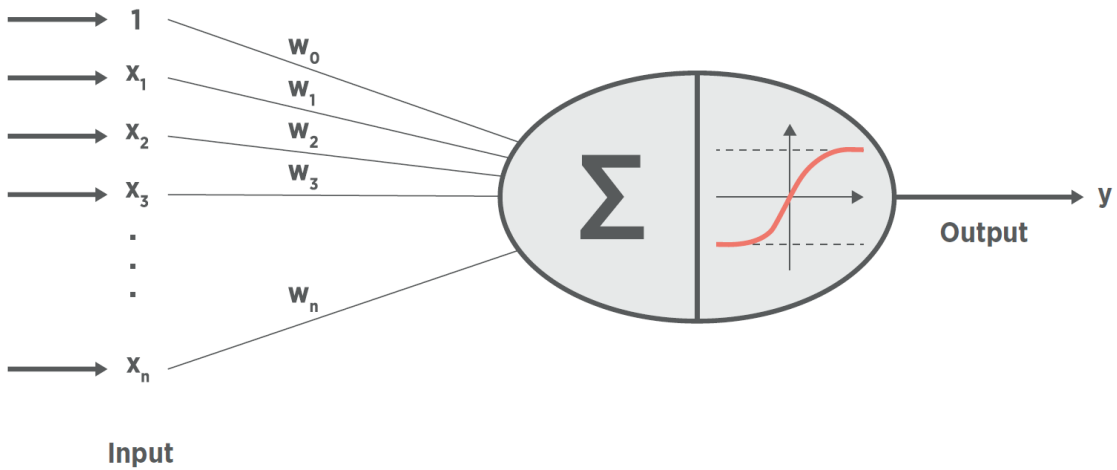
We will mention transformer networks in the natural language processing topic.

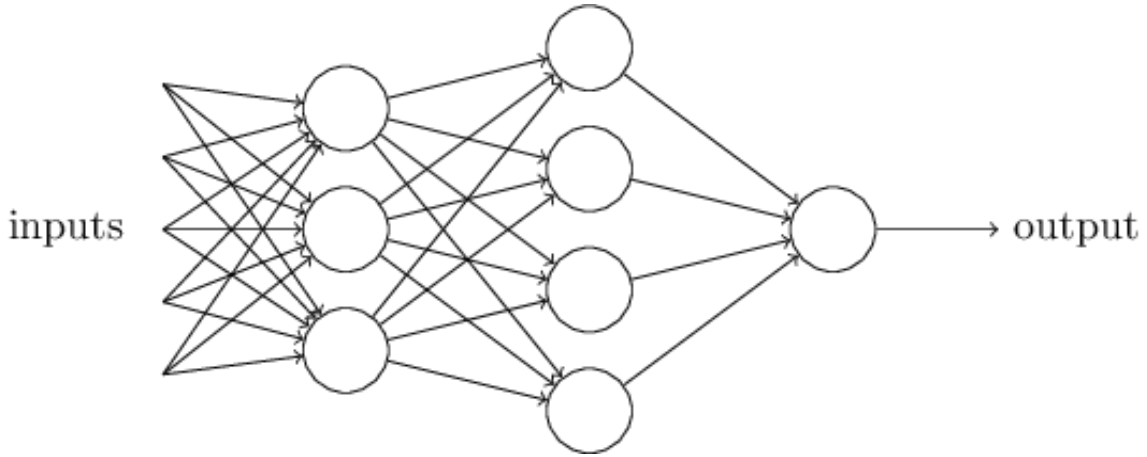
Artificial neural networks

- many approaches, we shall cover the basic ideas
- currently very strong interest, especially in deep neural networks
- http://www.deeplearningbook.org (from 2016, see also other newer literature in the introductory slides 01)

Artificial neural networks: brain analogy







learning: error backpropagation

A neuron

- Computational units, passing messages (information) in the network, typically organized into layers

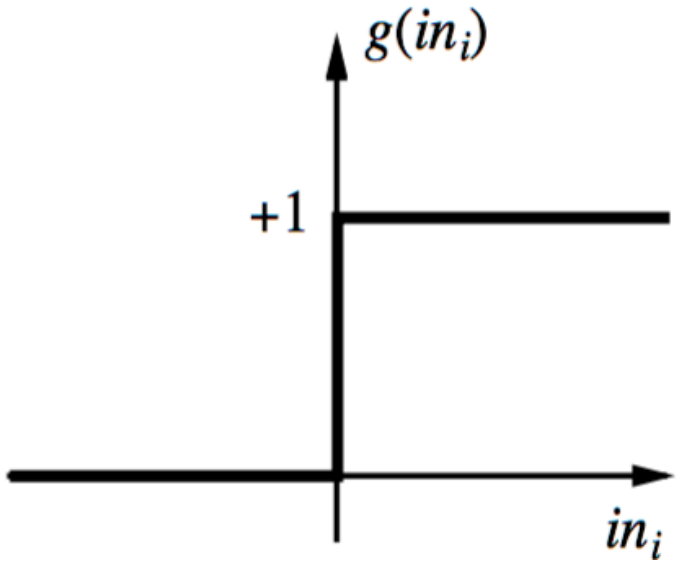
$$y = \sigma\left(\sum_{i=0}^{n} w_i x_i\right)$$

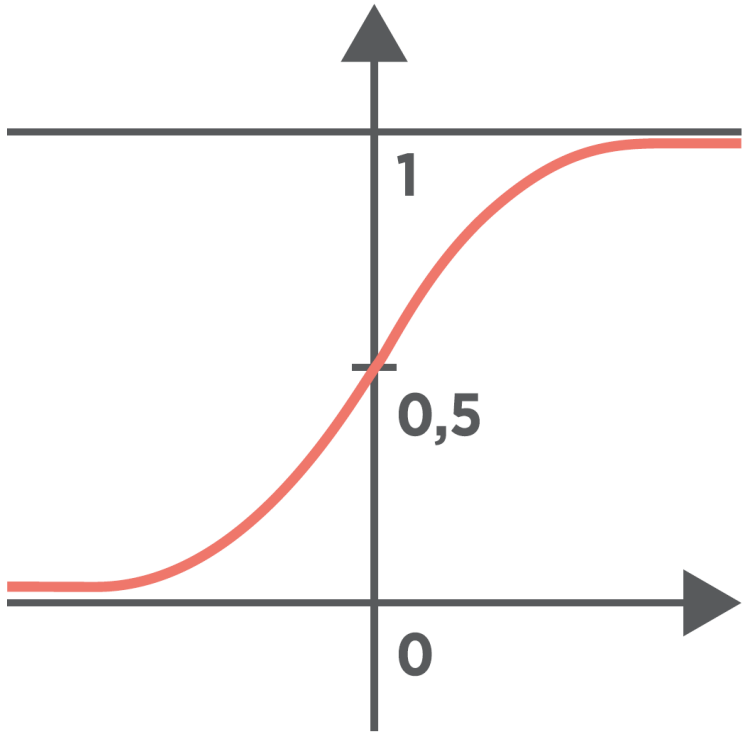
- where x_i are the inputs, $x_0 = 1$ (bias term), w_i are weights of the neuron, and σ is a non-linear activation function

Activation functions

- examples: step function, sigmoid (logistic)

$$f(x) = \frac{1}{1 + e^{-x}}$$





Activation functions

- ReLU (rectified linear unit)

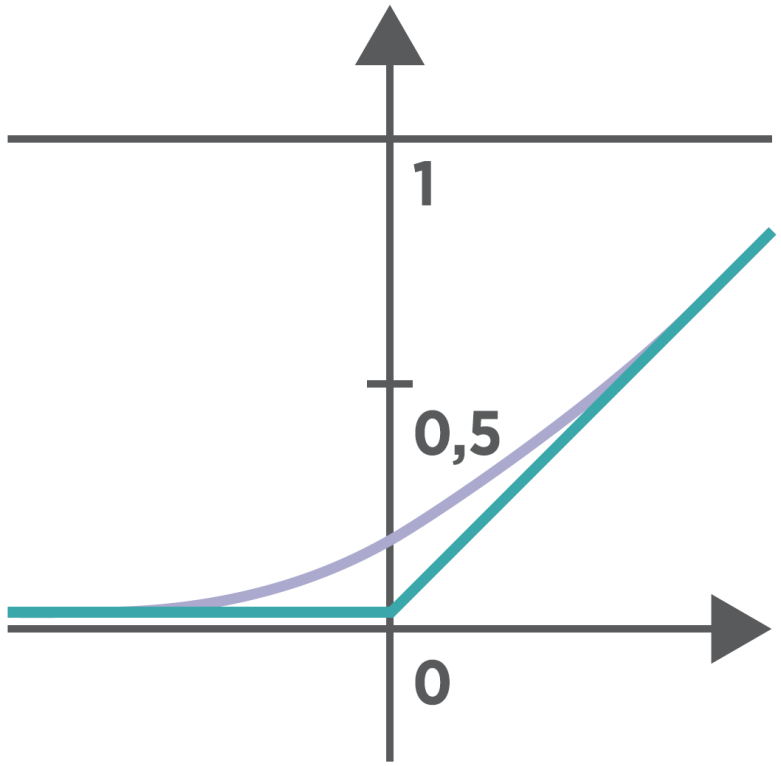
  $f(x) = \max(0, x)$

- softplus / approximation of ReLU with continuous derivation

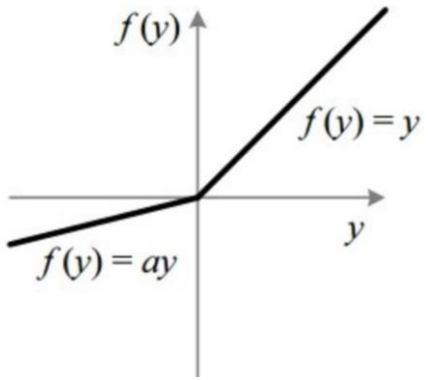
  $f(x) = \ln(1+e^x)$

- ELU (Exponential Linear Unit)

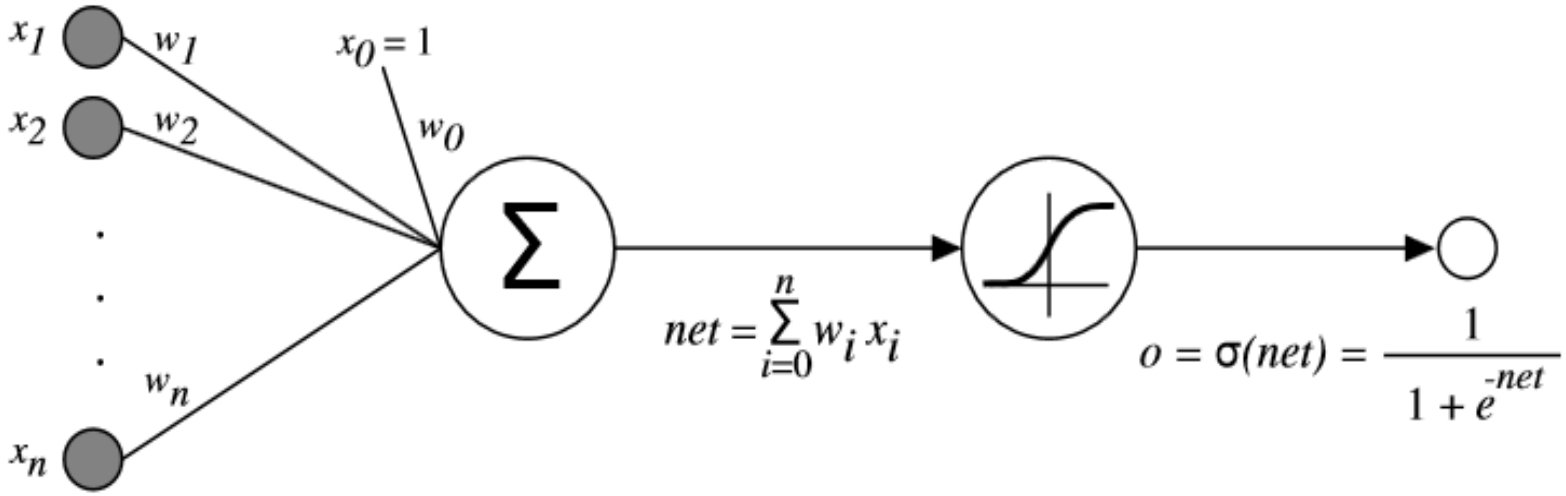
$$ELU(x) = \begin{cases} c \cdot (e^x - 1), & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$$



- Leaky ReLU: like ReLU but small slope for negative values instead of a flat slope
- many others

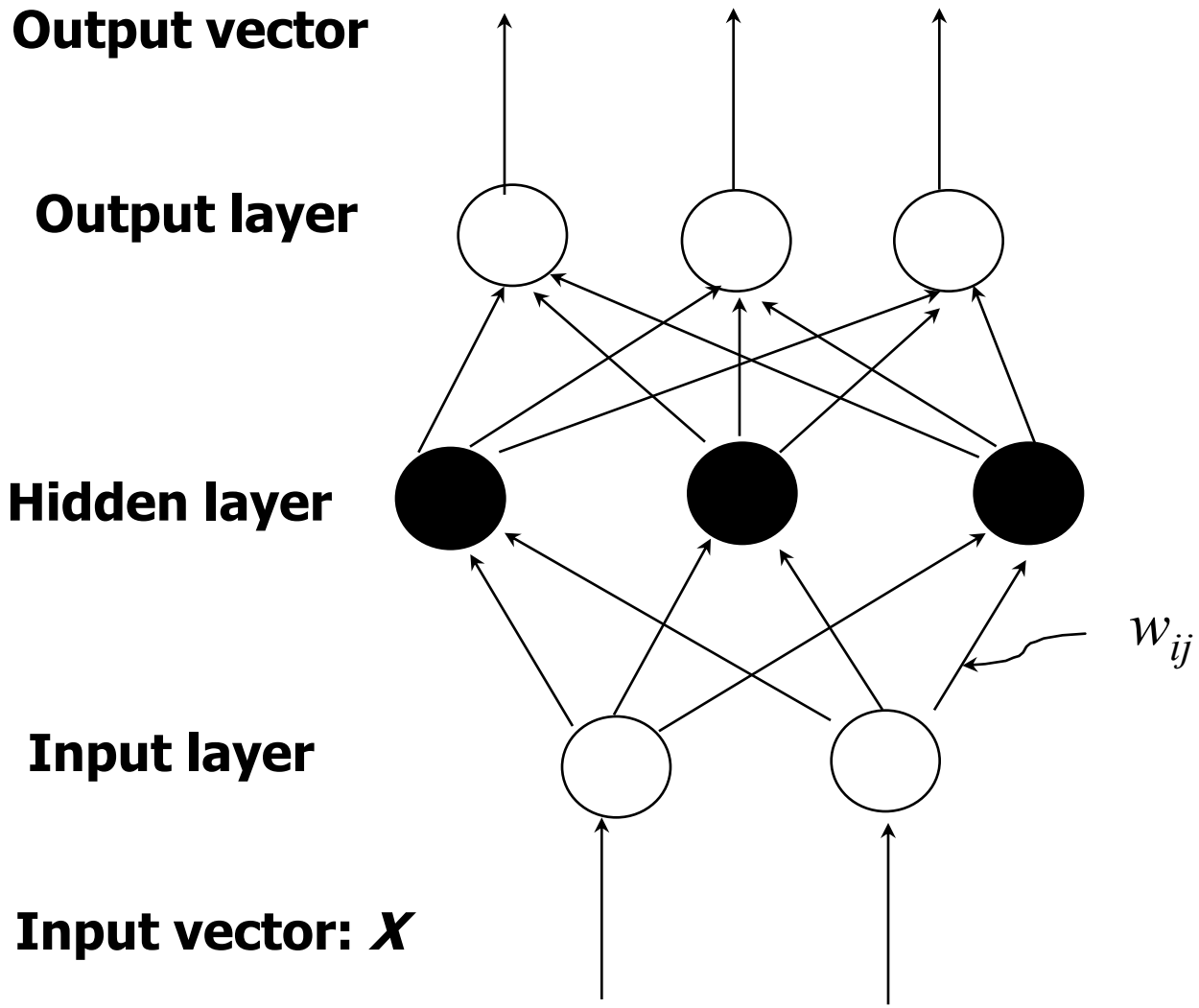


Why nonlinear?



What is a derivative of a sigmoid?

A multi-layer feed-forward NN

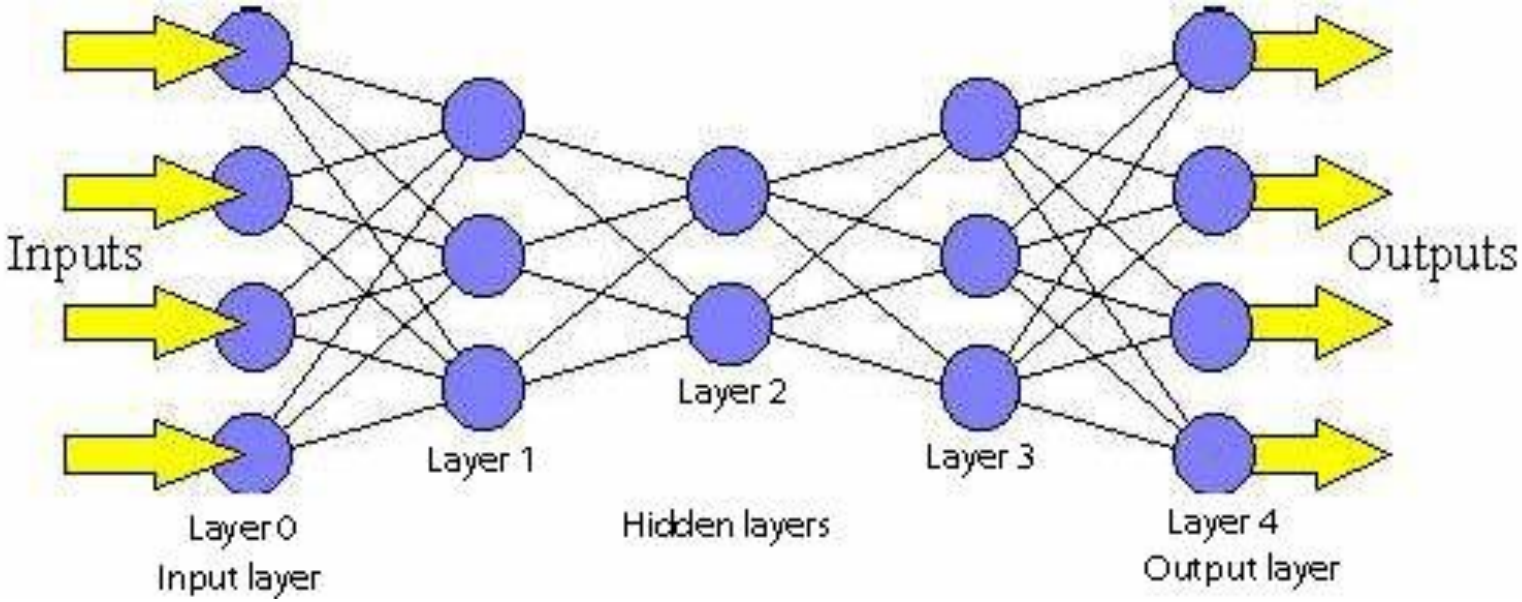


How a multi-layer NN works?

- The **inputs** to the network correspond to the attributes measured for each training tuple
- Inputs are fed simultaneously into the units making up the **input layer**
- They are then weighted and fed simultaneously to a **hidden layer**
- The number of hidden layers is arbitrary; if more than 1 hidden layer is used, the network is called a deep neural network
- The weighted outputs of the last hidden layer are input to units making up the **output layer**, which emits the network's prediction
- The network is **feed-forward**: None of the weights cycles back to an input unit or to an output unit of a previous layer
- If we have backwards connections, the network is called a recurrent neural network
- From a statistical point of view, networks perform **nonlinear regression**: Given enough hidden units and enough training samples, they can closely approximate any function

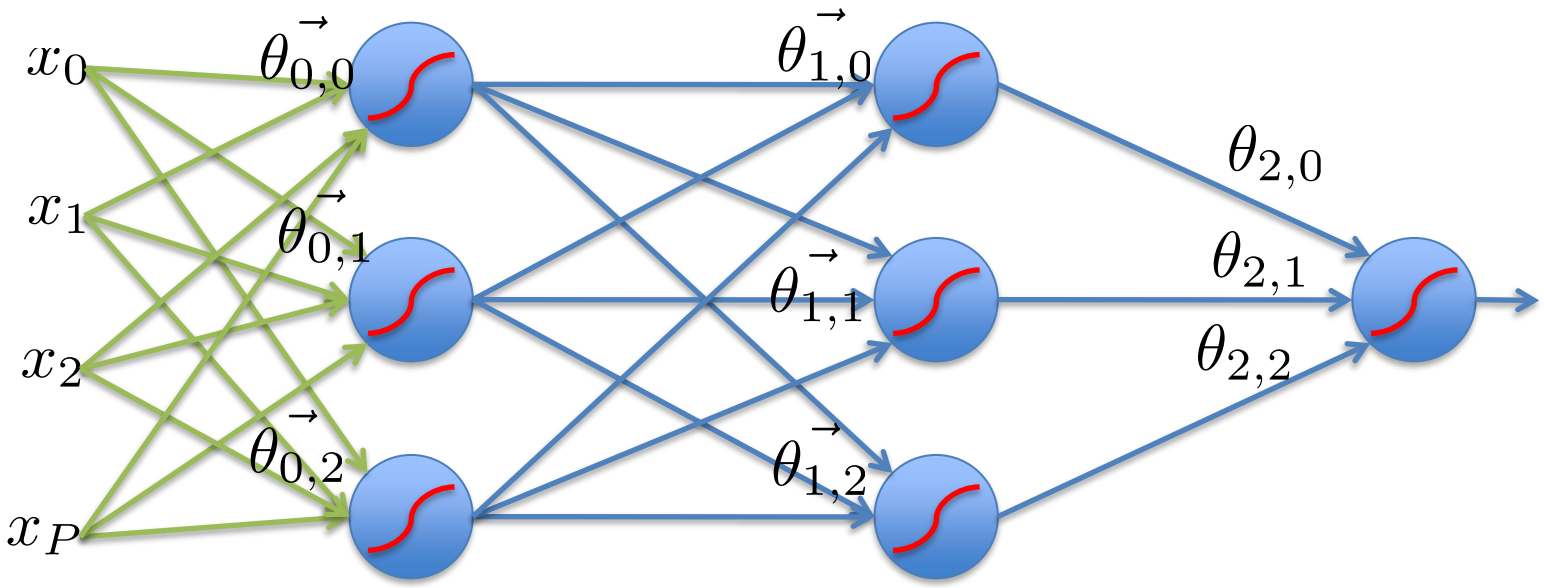
Feed-Forward Network

- neurons are activated progressively throug layers from input to output



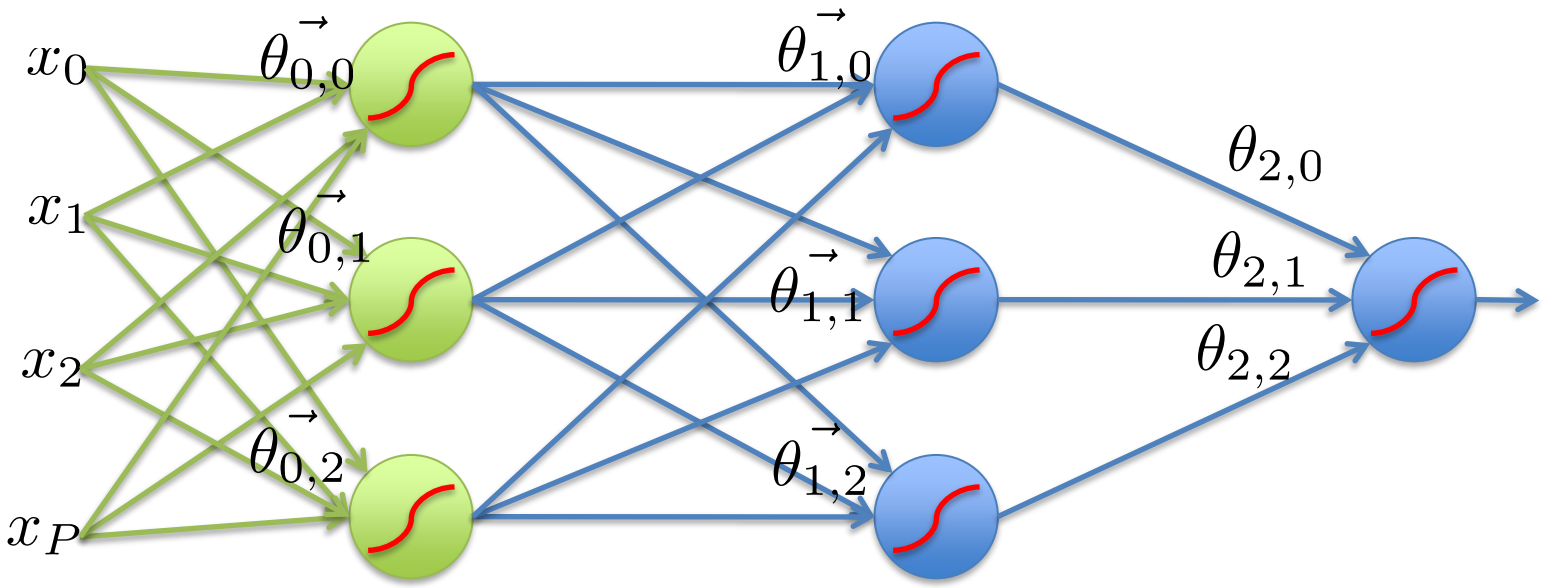
Feed-Forward Network

- Values are propagated through the network to the output, which returns the prediction

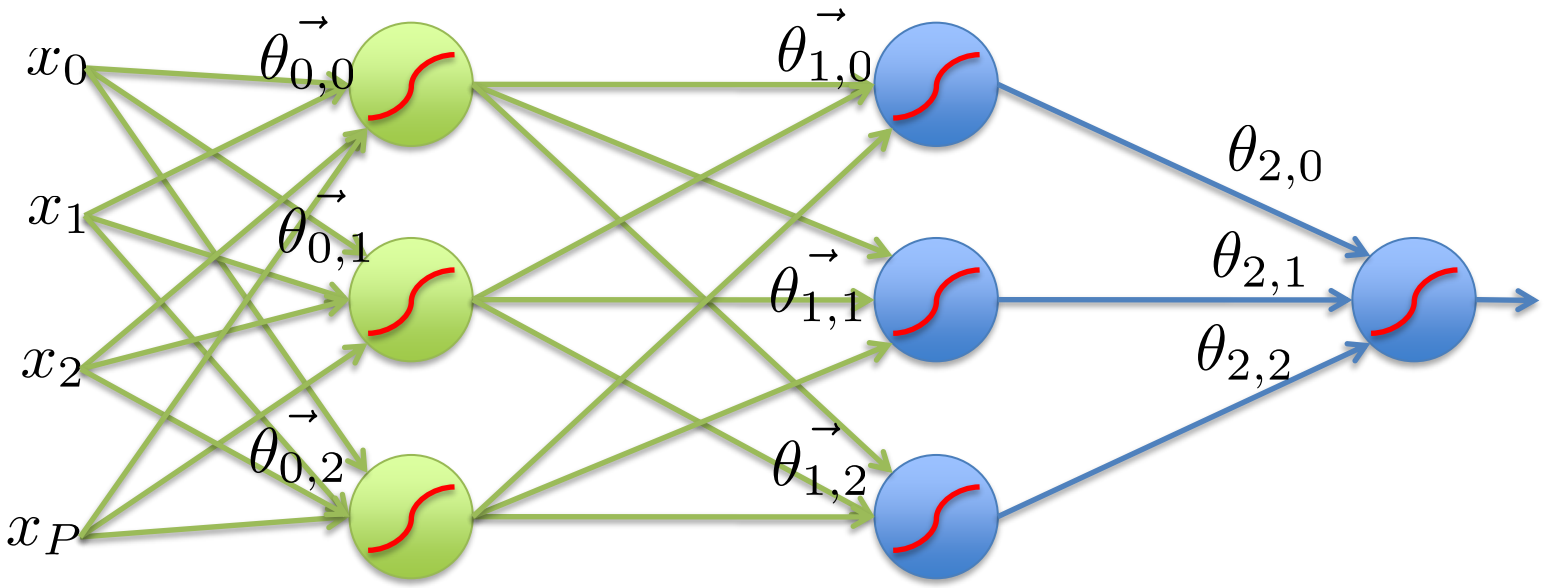


Next 6 slides by Andrew Rosenberg

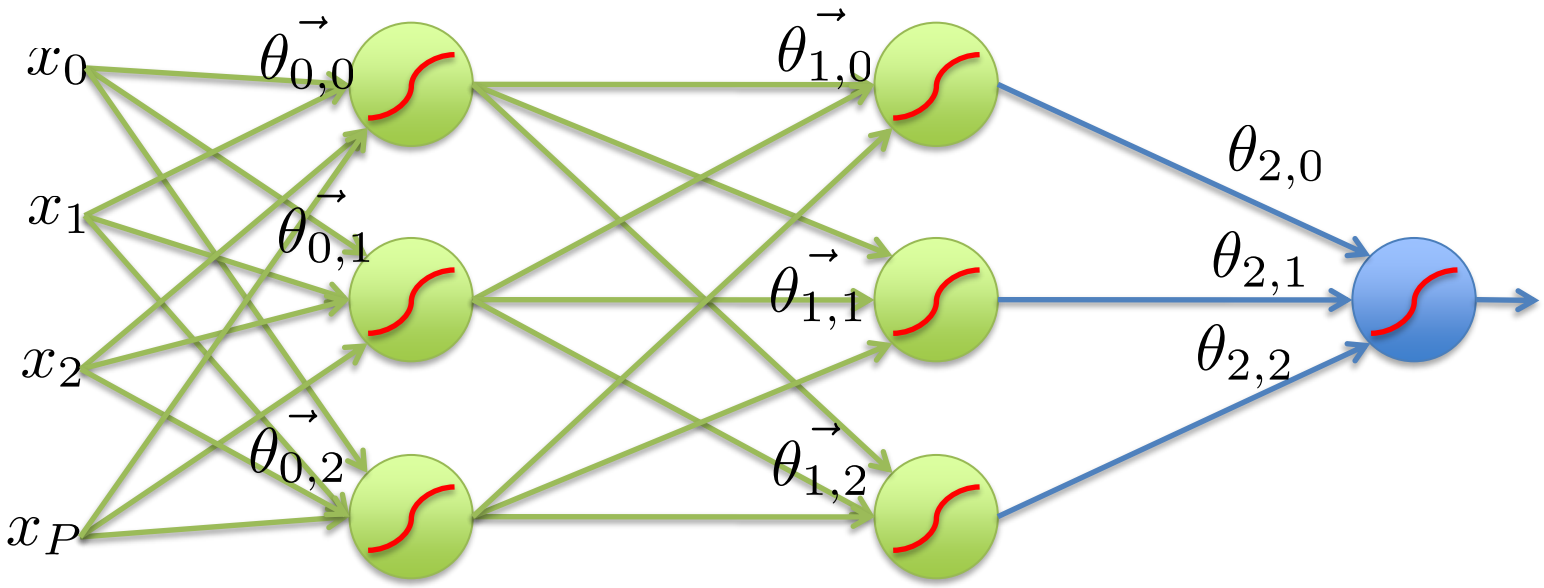
Feed-Forward Networks



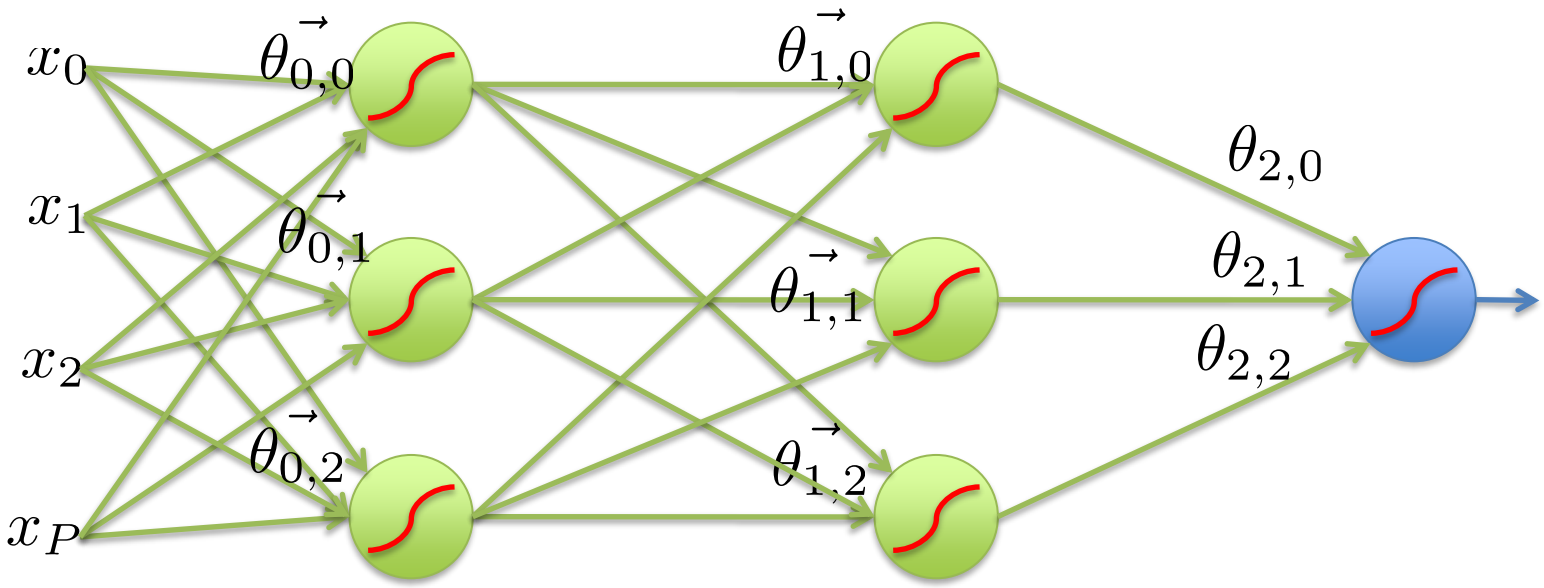
Feed-Forward Networks



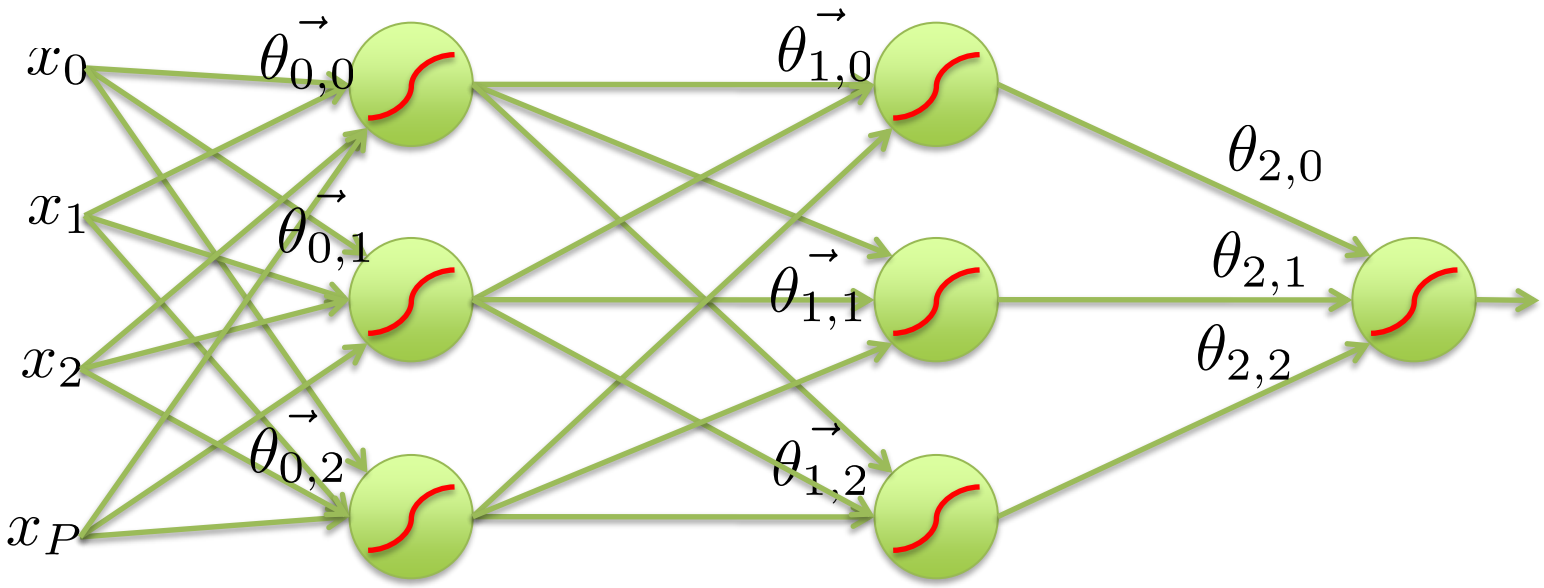
Feed-Forward Networks



Feed-Forward Networks



Feed-Forward Networks



Softmax

- In classification, softmax is often used for the last layer
- normalizes the output scores to be a probability distribution (values between 0 and 1, the sum is 1)

$$y_i = \frac{e^{z_i}}{\sum_{j \in group} e^{z_j}}$$

$$\frac{\partial y_i}{\partial z_i} = y_i \left(1 - y_i\right)$$

Criterion function

- together with softmax we frequently use cross entropy as loss (cost) function C

$$C = -\sum_j t_j \log y_j$$

target value

$$\frac{\partial C}{\partial z_i} = \sum_j \frac{\partial C}{\partial y_j} \frac{\partial y_j}{\partial z_i} = y_i - t_i$$

Backpropagation learning algorithm for NN

- Backpropagation: a neural network learning algorithm
- Started by psychologists and neurobiologists to develop and test computational analogues of neurons
- A neural network: a set of connected input/output units where each connection has a weight associated with it
- During the learning phase, the network learns by adjusting the weights so as to be able to predict the correct class label of the input tuples
- Also referred to as connectionist learning due to the connections between units

Backpropagation algorithm

- Iteratively process a set of training tuples & compare the network's prediction with the actual known target value
- For each training tuple, the weights are modified to **minimize the mean squared error** between the network's prediction and the actual target value
- Modifications are made in the "**backwards**" direction: from the output layer, through each hidden layer down to the first hidden layer, hence "**backpropagation**"
- Steps
  - Initialize weights to small random numbers, associated with biases
  - Propagate the inputs forward (by applying the activation function)
  - Backpropagate the error (by updating weights and biases)
  - Terminating condition (when error is very small, etc.)

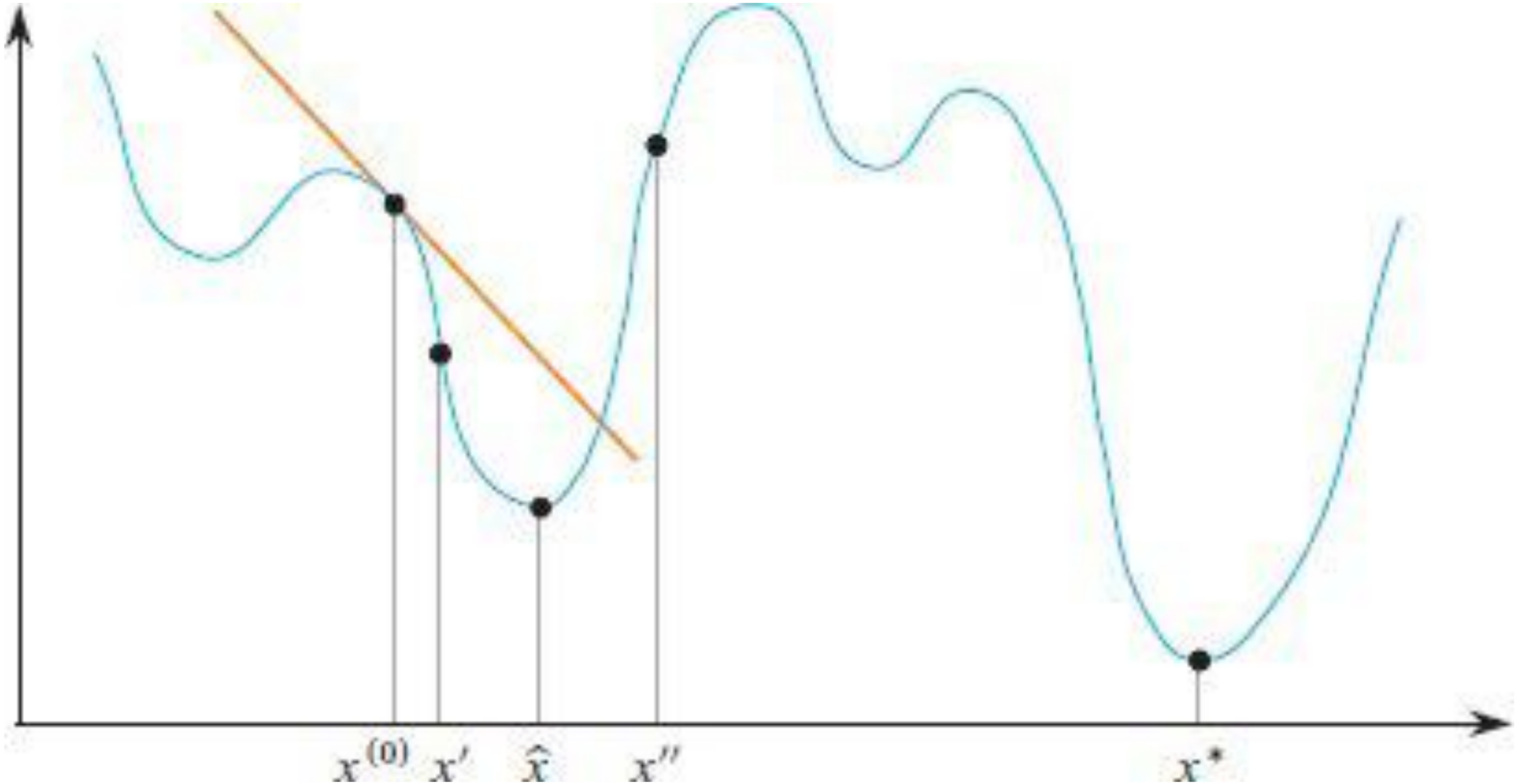
Gradient descent (GD)

- Gradient descent is an efficient local optimization in $\mathbb{R}^n$
- Local minimum of function *f:* $\mathbb{R}^n \rightarrow \mathbb{R}$ is a point ***x*** for which *f(x) ≤ f(x')* for all ***x'*** that are "near" ***x***
- Gradient $\nabla f(x)$ is a function ∇f: $\mathbb{R}^n \rightarrow \mathbb{R}^n$ comprising *n* partial derivatives:

$$\nabla f(x) = (\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \dots, \frac{\partial f}{\partial x_n})$$

- The GD optimization moves in the direction of -$\nabla f(x)$

Illustration of GD



GD algorithm

```
GRADIENT-DESCENT(f, x0, γ, T) {
  // function f,  initial value x0, fixed step size γ, number of steps T
  x_best = x = x0 ; // n-dimensional vectors, initially set to the initial value
  f_best = f_x = f(x_best) ;
  for t = 0 to T – 1 do {
    x_next = x – γ · ∇f(x);  // ∇f(x), x, and x_next are n-dimensional
    f_next = f(x_next)
    if (f_next < f_x)
      x_best = x_next ;
    x = x_next ;
    f_x = f_next ;
  }
  return x_best ;
}
```

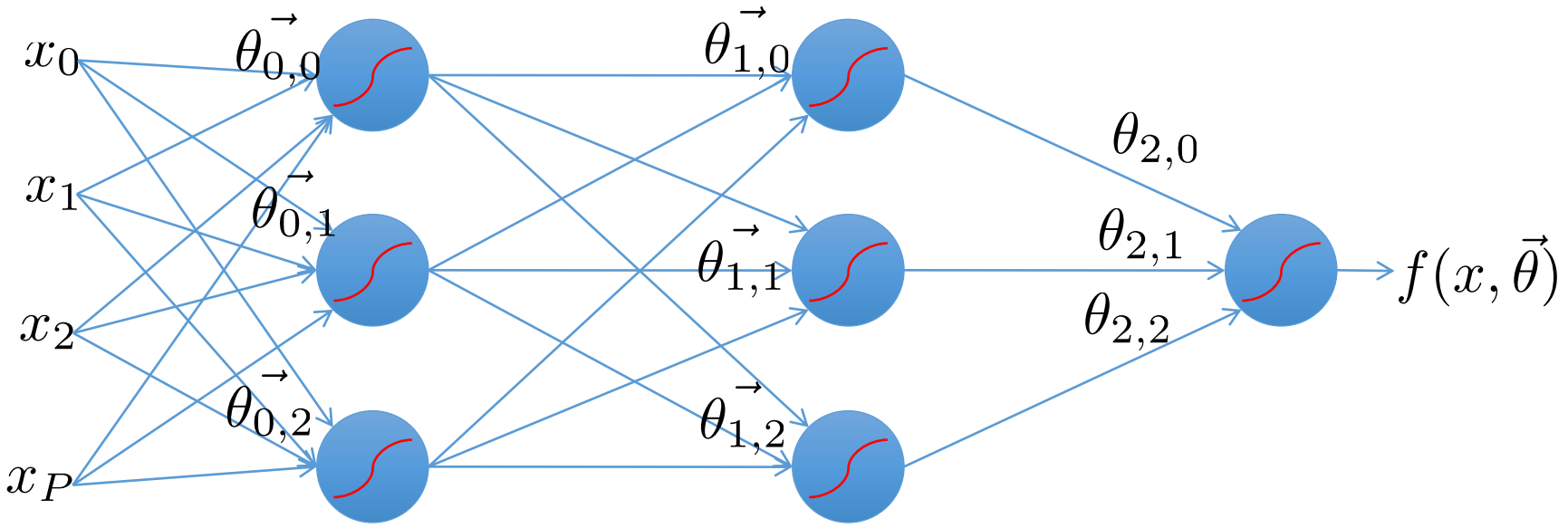
Chain rule of derivation

- In a network, the output of each neuron is a function of the activation function and all its inputs, where the inputs may again be composite functions of neurons in previous layers
- To compute the gradient of a composite function, we use the chain rule of derivation

$$f\big(g(x)\big)' = f'\big(g(x)\big)g'(x)$$

Error Backpropagation

- We will do gradient descent on the whole network.
- Training will proceed from the last layer to the first.

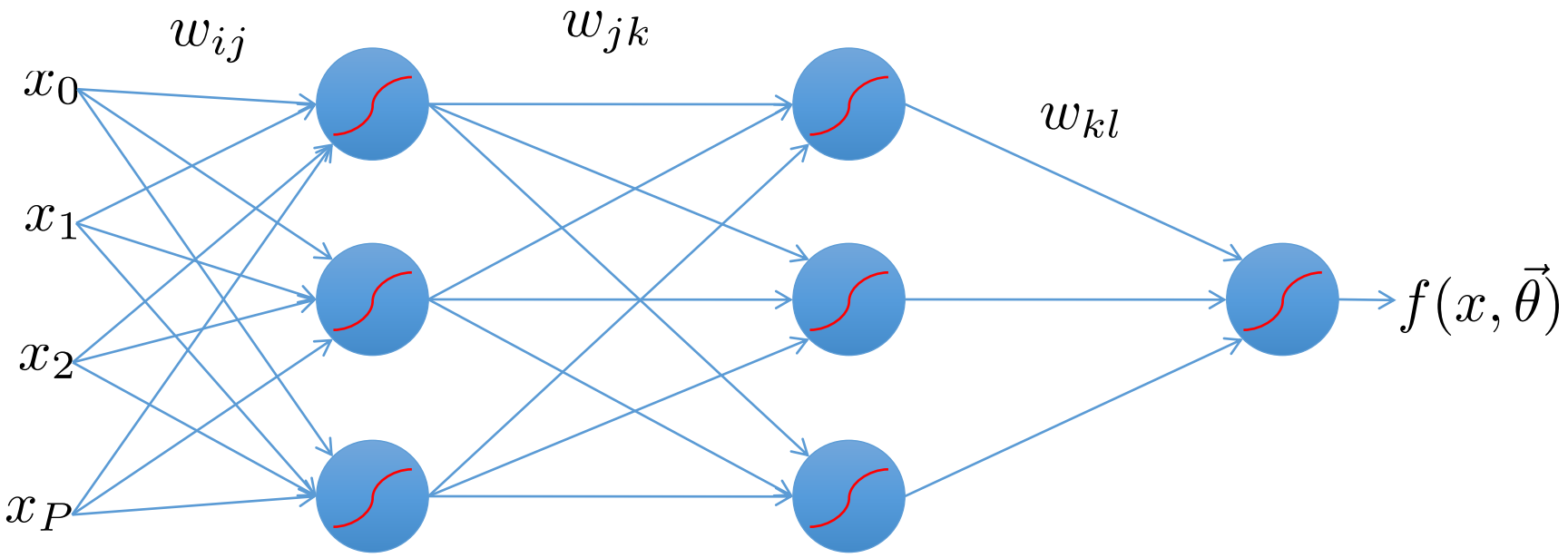


Next 18 slides by Andrew Rosenberg

Error Backpropagation

- Introduce variables over the neural network

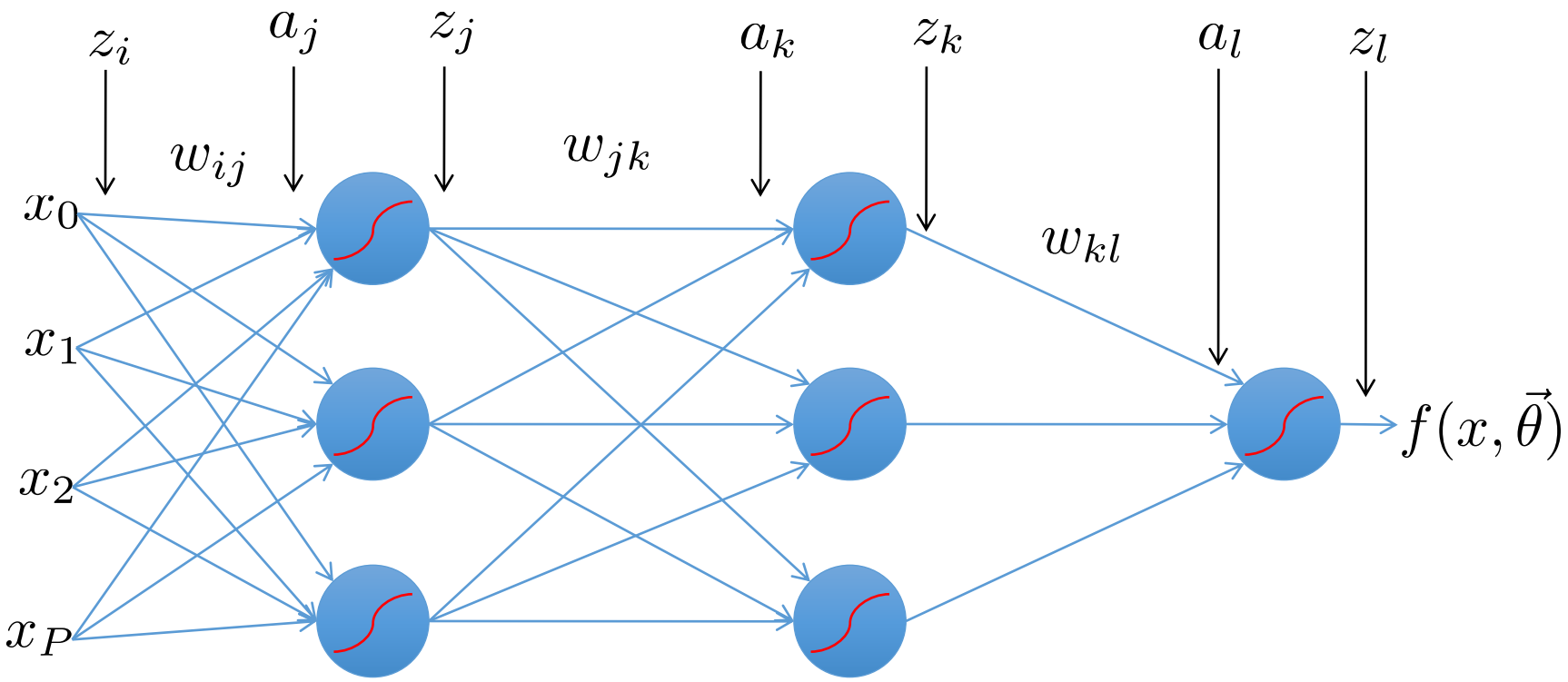
$$\vec{\theta} = \{w_{ij}, w_{jk}, w_{kl}\}$$



Error Backpropagation

$$\vec{\theta} = \{w_{ij}, w_{jk}, w_{kl}\}$$

- Introduce variables over the neural network
  - Distinguish the input and output of each node

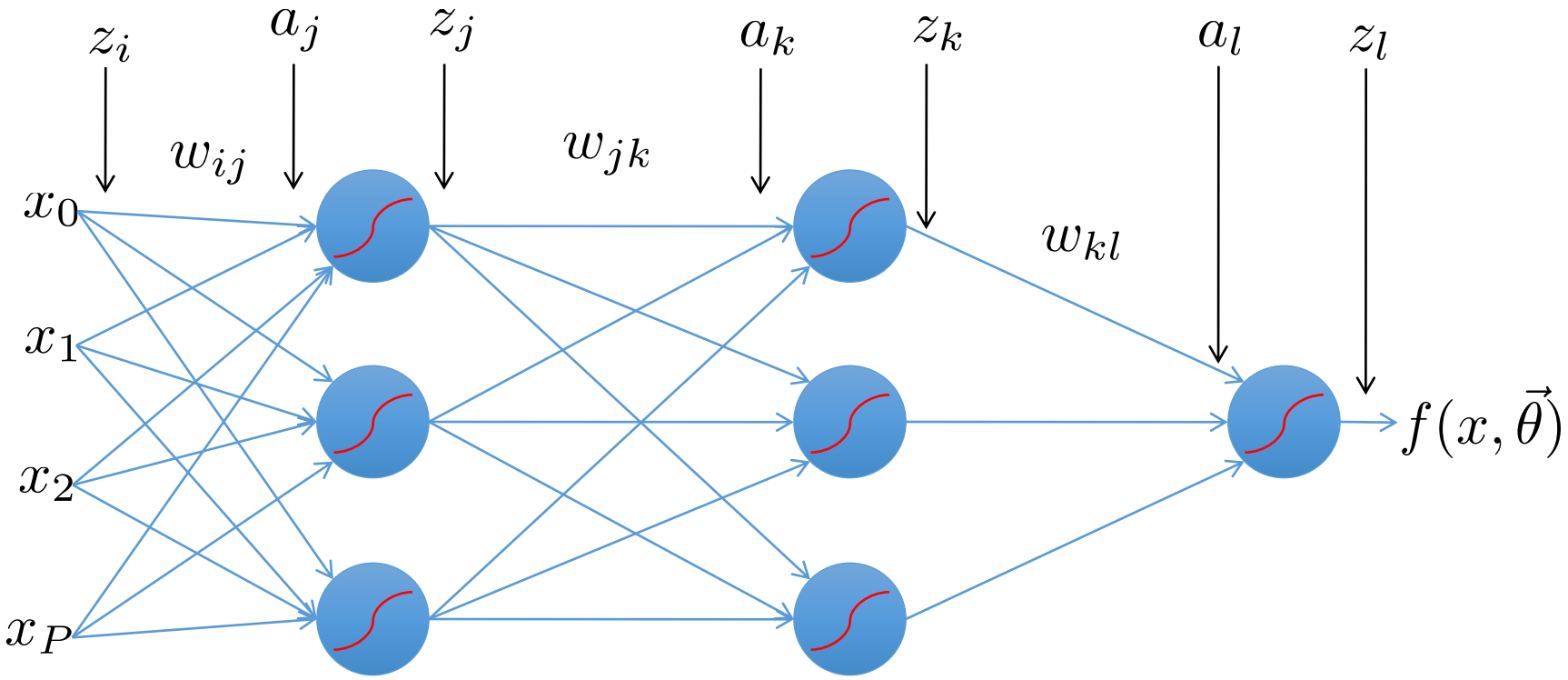


Error Backpropagation

$$\vec{\theta} = \{w_{ij}, w_{jk}, w_{kl}\}$$

$$a_j = \sum_i w_{ij} z_i \qquad a_k = \sum_j w_{jk} z_j \qquad a_l = \sum_k w_{kl} z_k$$

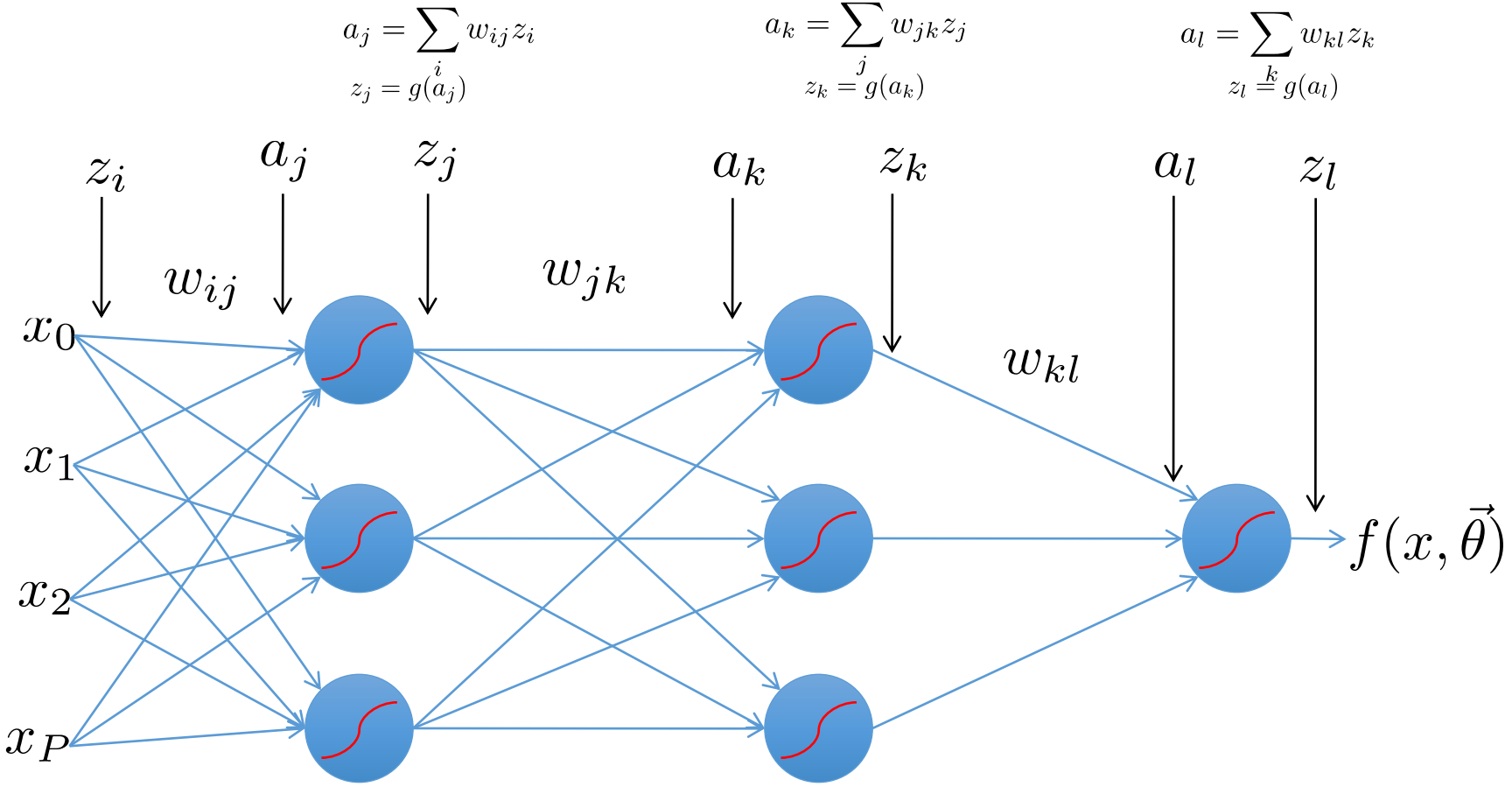
$$z_j = g(a_j) \qquad z_k = g(a_k) \qquad z_l = g(a_l)$$



Error Backpropagation

$$\vec{\theta} = \{w_{ij}, w_{jk}, w_{kl}\}$$

Training: Take the gradient of the last component and iterate backwards



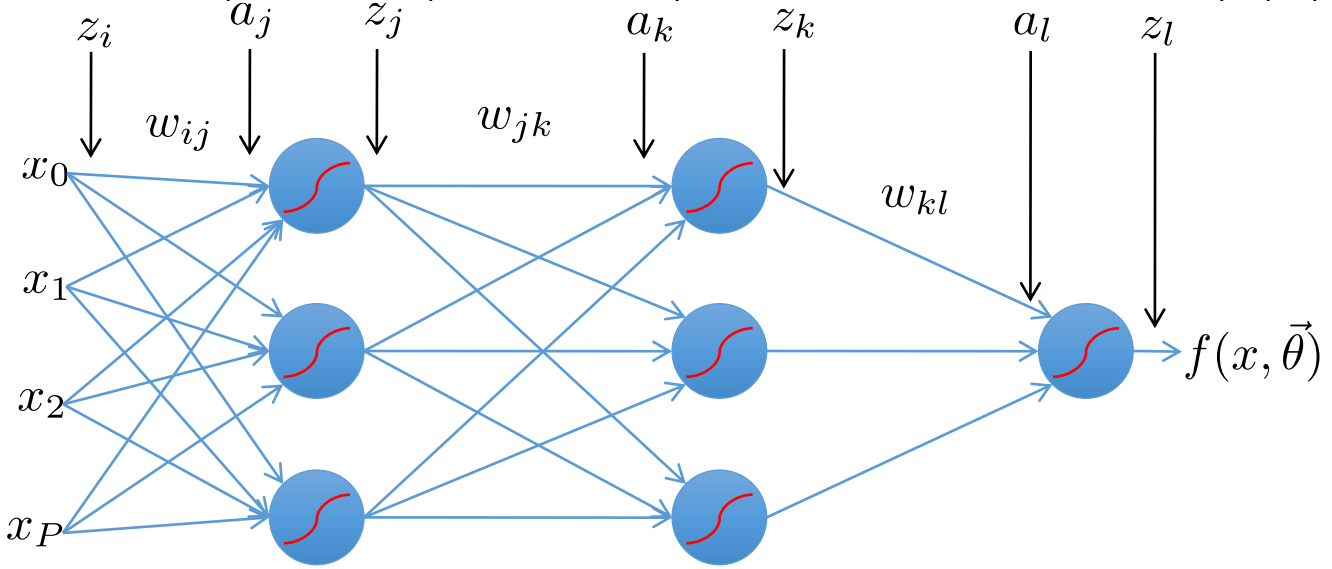
Error Backpropagation

$$R(\theta) = \frac{1}{N}\sum_{n=0}^{N} L(y_n - f(x_n))$$

Empirical Risk Function

$$= \frac{1}{N}\sum_{n=0}^{N} \frac{1}{2}\left(y_n - f(x_n)\right)^2$$

$$= \frac{1}{N}\sum_{n=0}^{N} \frac{1}{2}\left(y_n - g\left(\sum_k w_{kl} g\left(\sum_j w_{jk} g\left(\sum_i w_{ij} x_{n,i}\right)\right)\right)\right)^2$$



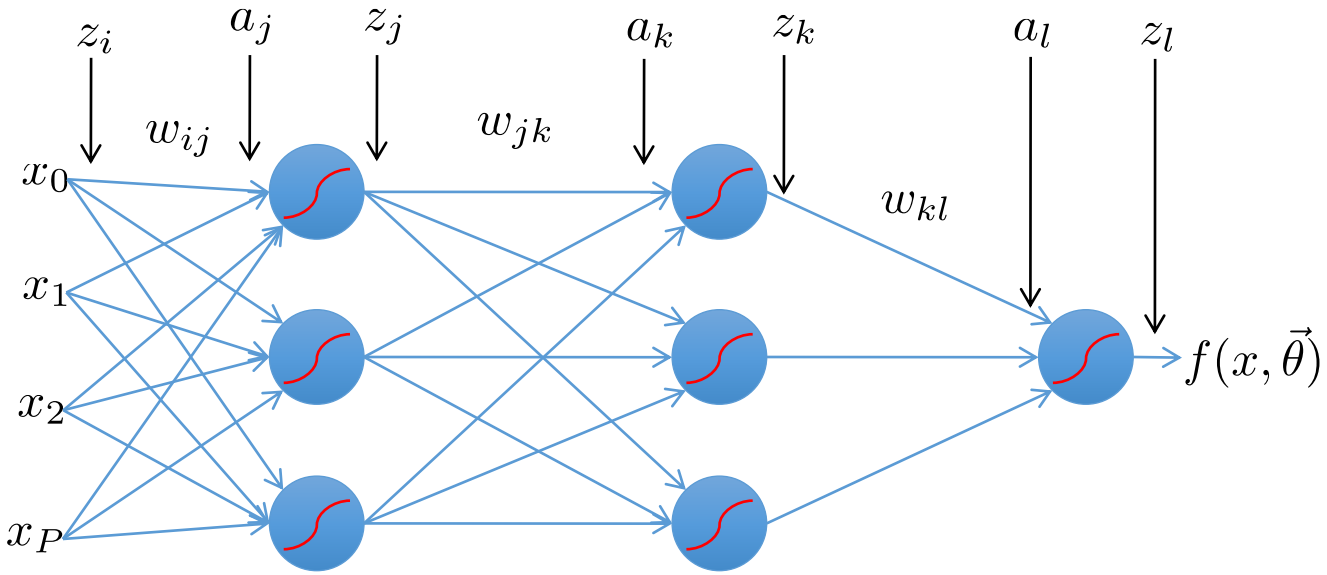
Error Backpropagation

Optimize last layer weights w_{kl}

$$L_n = \frac{1}{2}\left(y_n - f(x_n)\right)^2$$

$$\frac{\partial R}{\partial w_{kl}} = \frac{1}{N}\sum_n \left[\frac{\partial L_n}{\partial a_{l,n}}\right]\left[\frac{\partial a_{l,n}}{\partial w_{kl}}\right]$$

Calculus chain rule



Error Backpropagation

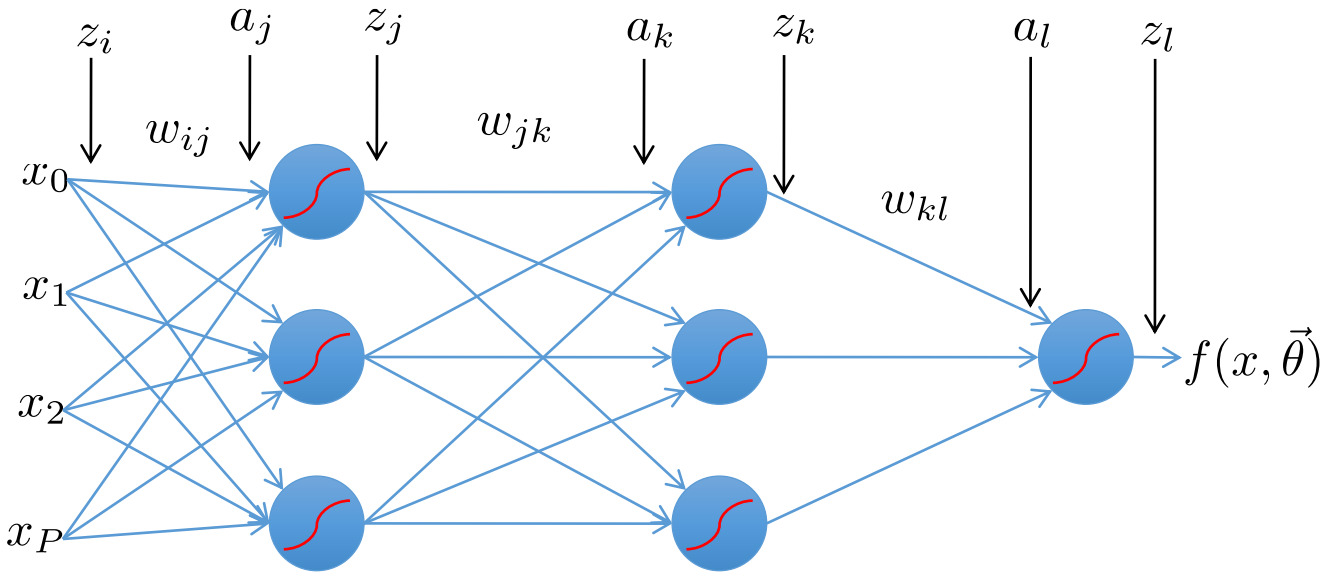
Optimize last layer weights w_{kl}

$$L_n = \frac{1}{2}\left(y_n - f(x_n)\right)^2$$

$$\frac{\partial R}{\partial w_{kl}} = \frac{1}{N}\sum_n \left[\frac{\partial L_n}{\partial a_{l,n}}\right]\left[\frac{\partial a_{l,n}}{\partial w_{kl}}\right]$$

Calculus chain rule

$$\frac{\partial R}{\partial w_{kl}} = \frac{1}{N}\sum_n \left[\frac{\partial \frac{1}{2}(y_n - g(a_{l,n}))^2}{\partial a_{l,n}}\right]\left[\frac{\partial a_{l,n}}{\partial w_{kl}}\right]$$



Error Backpropagation

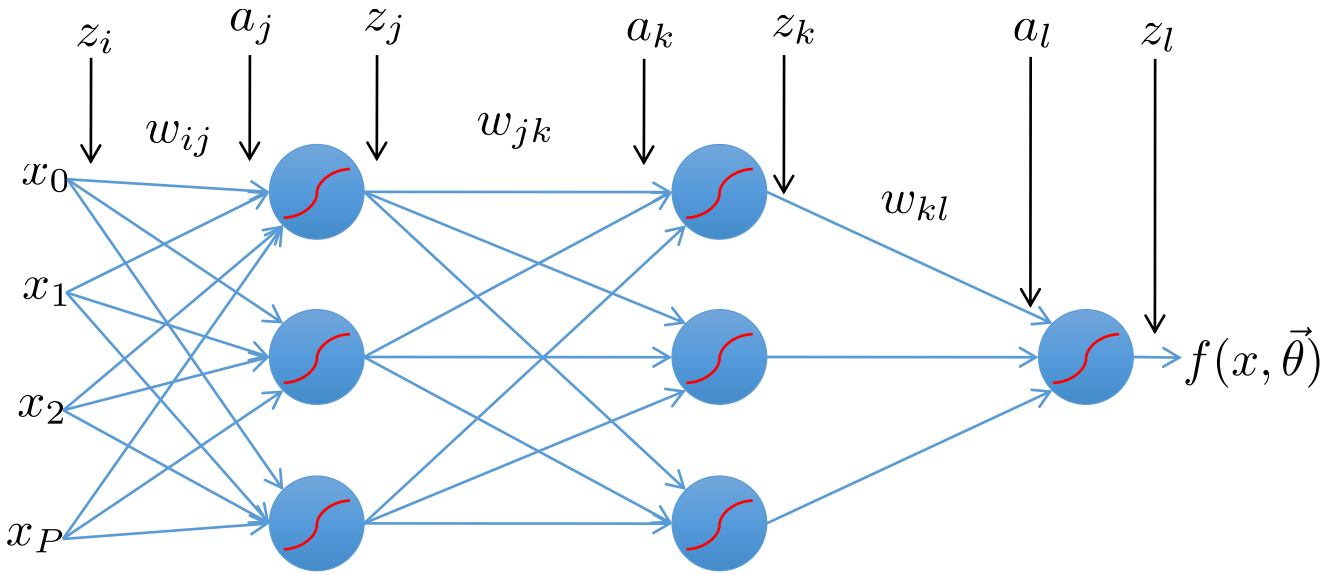
Optimize last layer weights w_{kl}

$$L_n = \frac{1}{2}\left(y_n - f(x_n)\right)^2$$

$$\frac{\partial R}{\partial w_{kl}} = \frac{1}{N}\sum_n \left[\frac{\partial L_n}{\partial a_{l,n}}\right]\left[\frac{\partial a_{l,n}}{\partial w_{kl}}\right]$$

Calculus chain rule

$$\frac{\partial R}{\partial w_{kl}} = \frac{1}{N}\sum_n \left[\frac{\partial \frac{1}{2}(y_n - g(a_{l,n}))^2}{\partial a_{l,n}}\right]\left[\frac{\partial z_{k,n} w_{kl}}{\partial w_{kl}}\right]$$



Error Backpropagation

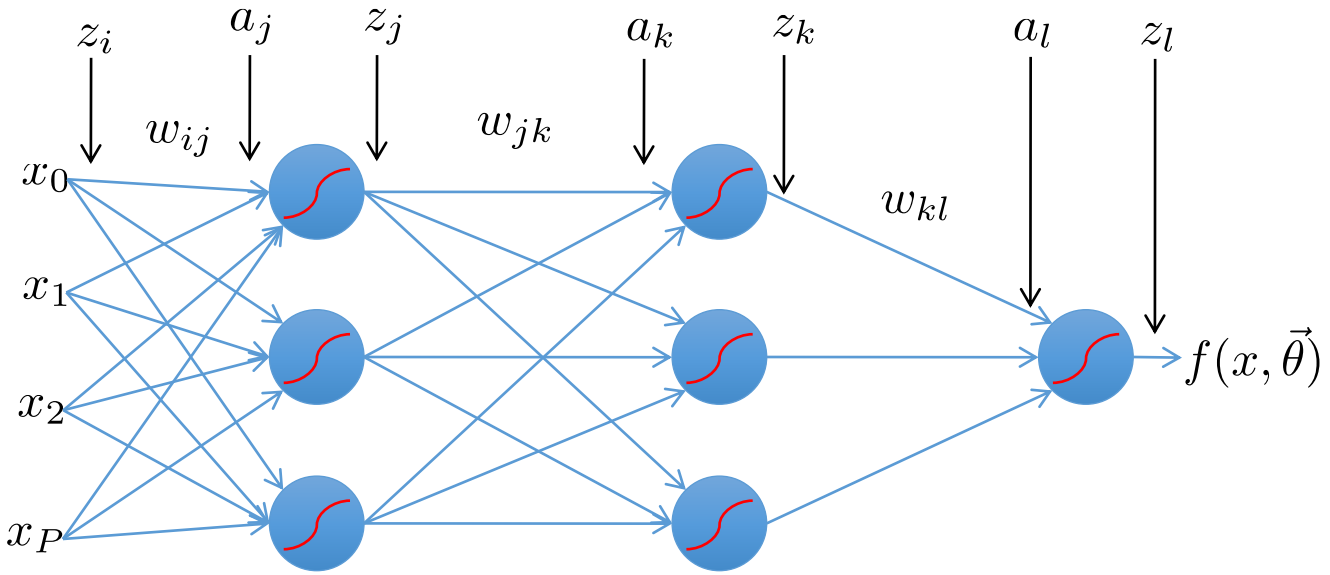
Optimize last layer weights w_{kl}

$$L_n = \frac{1}{2}\left(y_n - f(x_n)\right)^2$$

$$\frac{\partial R}{\partial w_{kl}} = \frac{1}{N}\sum_n \left[\frac{\partial L_n}{\partial a_{l,n}}\right]\left[\frac{\partial a_{l,n}}{\partial w_{kl}}\right]$$

Calculus chain rule

$$\frac{\partial R}{\partial w_{kl}} = \frac{1}{N}\sum_n \left[\frac{\partial \frac{1}{2}(y_n - g(a_{l,n}))^2}{\partial a_{l,n}}\right]\left[\frac{\partial z_{k,n} w_{kl}}{\partial w_{kl}}\right] = \frac{1}{N}\sum_n \left[-(y_n - z_{l,n})g'(a_{l,n})\right] z_{k,n}$$



Error Backpropagation

Optimize last layer weights w_{kl}

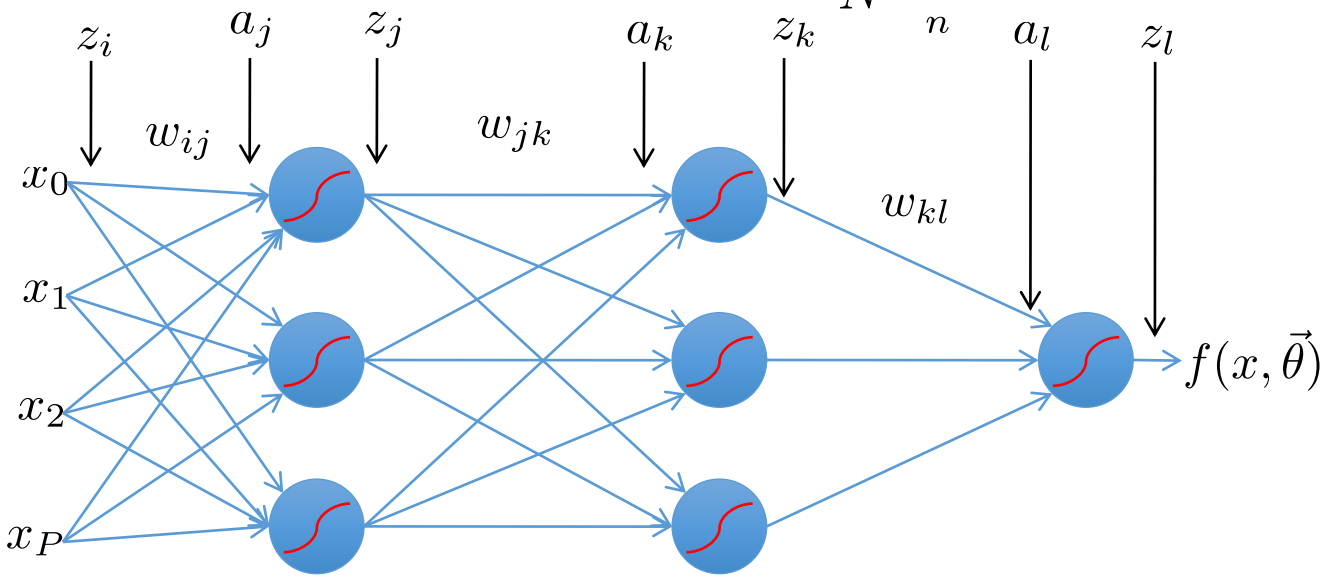
$$L_n = \frac{1}{2}\left(y_n - f(x_n)\right)^2$$

$$\frac{\partial R}{\partial w_{kl}} = \frac{1}{N}\sum_n \left[\frac{\partial L_n}{\partial a_{l,n}}\right]\left[\frac{\partial a_{l,n}}{\partial w_{kl}}\right]$$

Calculus chain rule

$$\frac{\partial R}{\partial w_{kl}} = \frac{1}{N}\sum_n \left[\frac{\partial \frac{1}{2}(y_n - g(a_{l,n}))^2}{\partial a_{l,n}}\right]\left[\frac{\partial z_{k,n} w_{kl}}{\partial w_{kl}}\right] = \frac{1}{N}\sum_n \left[-(y_n - z_{l,n})g'(a_{l,n})\right] z_{k,n}$$

$$= \frac{1}{N}\sum_n \delta_{l,n} n z_{k,n}$$

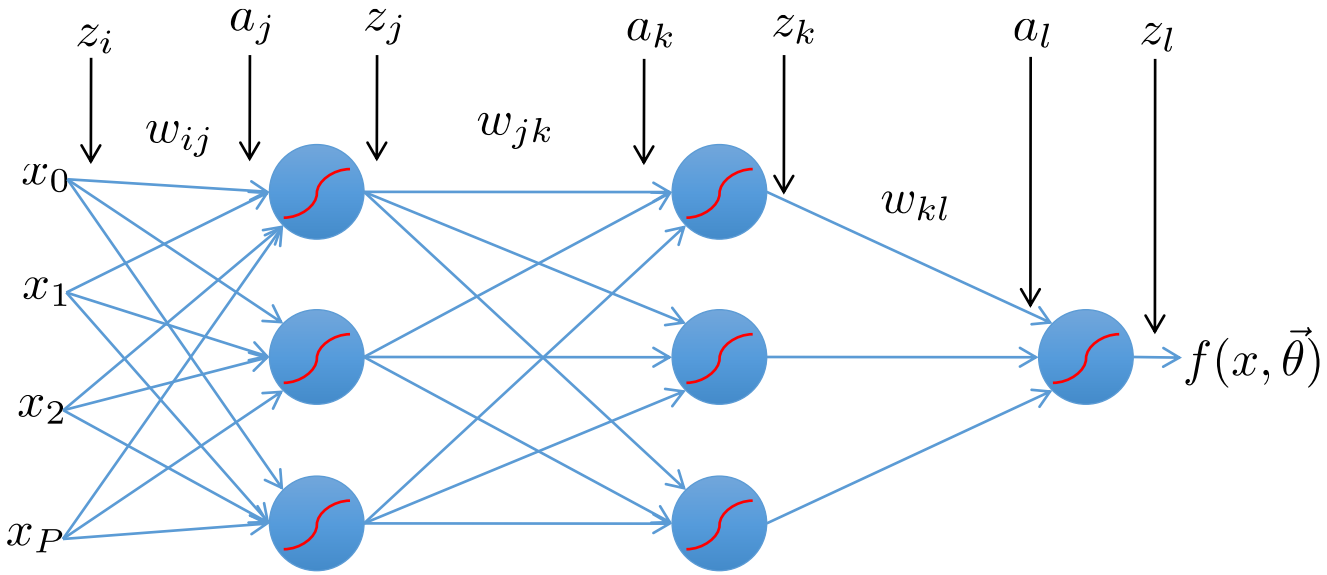


Error Backpropagation

Optimize last hidden weights w_{jk}

$$\frac{\partial R}{\partial w_{jk}} = \frac{1}{N}\sum_n \left[\frac{\partial L_n}{\partial a_{k,n}}\right]\left[\frac{\partial a_{k,n}}{\partial w_{jk}}\right]$$

$$\frac{\partial R}{\partial w_{kl}} = \frac{1}{N}\sum_n \delta_{l,n} z_{k,n}$$



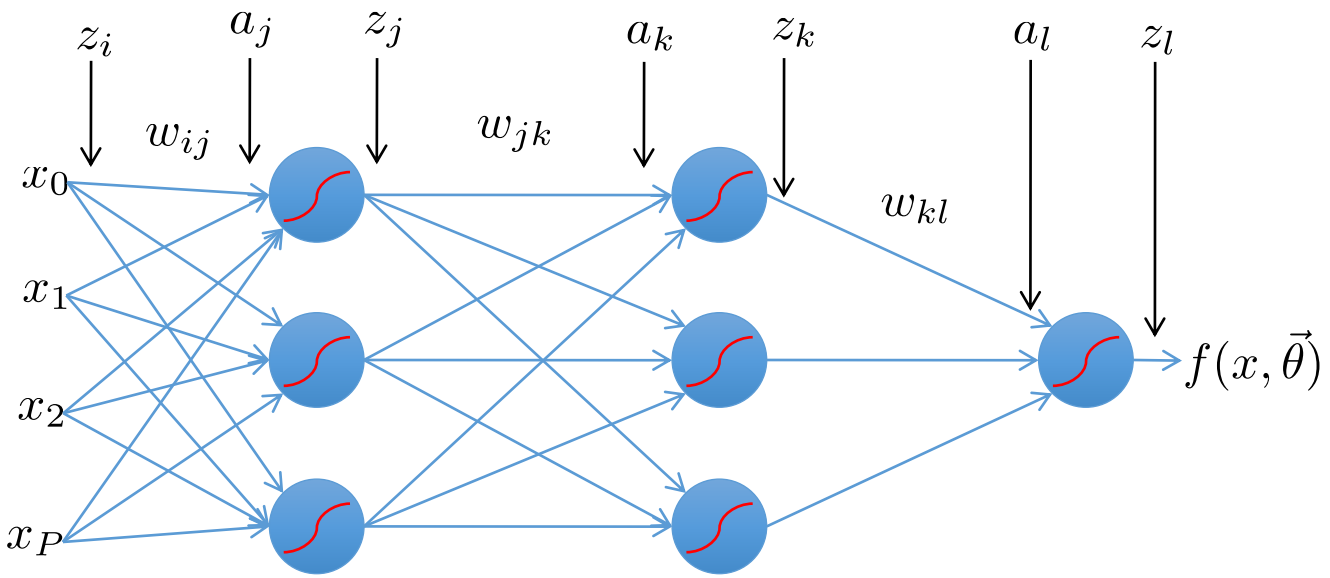
Error Backpropagation

Optimize last hidden weights w_{jk}

$$\frac{\partial R}{\partial w_{kl}} = \frac{1}{N}\sum_n \delta_{l,n} z_{k,n}$$

$$\frac{\partial R}{\partial w_{jk}} = \frac{1}{N}\sum_n \left[\sum_l \frac{\partial L_n}{\partial a_{l,n}} \frac{\partial a_{l,n}}{\partial a_{k,n}}\right] \left[\frac{\partial a_{k,n}}{\partial w_{jk}}\right]$$

Multivariate chain rule



Error Backpropagation

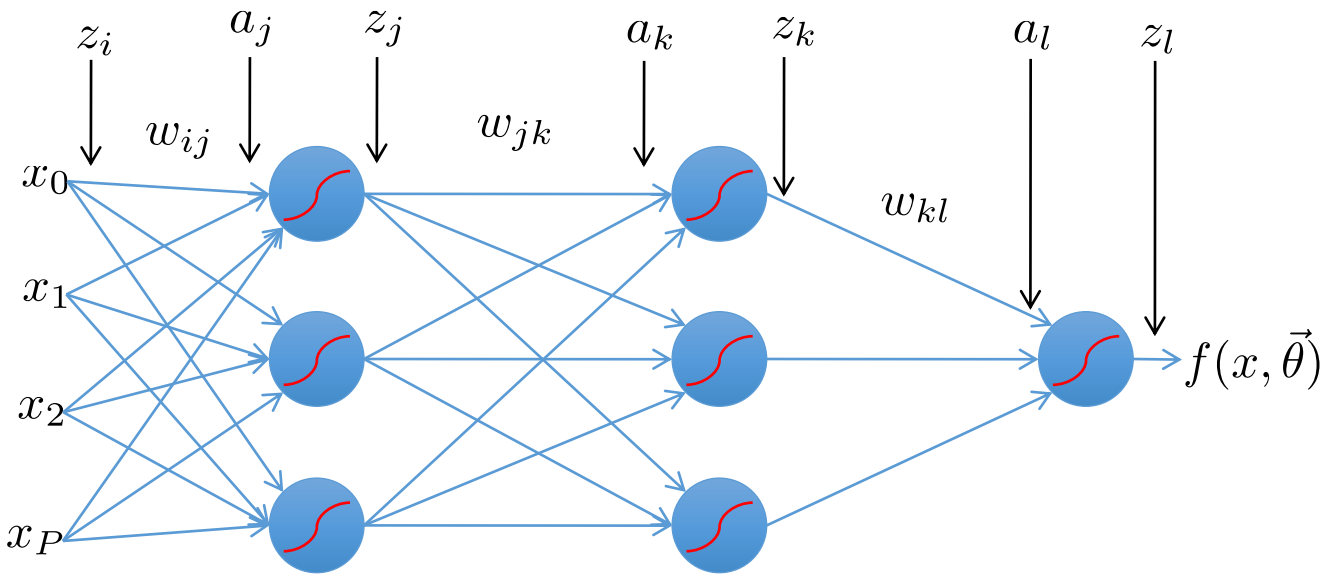
Optimize last hidden weights w_{jk}

$$\frac{\partial R}{\partial w_{kl}} = \frac{1}{N}\sum_n \delta_{l,n} z_{k,n}$$

$$\frac{\partial R}{\partial w_{jk}} = \frac{1}{N}\sum_n \left[\sum_l \frac{\partial L_n}{\partial a_{l,n}} \frac{\partial a_{l,n}}{\partial a_{k,n}}\right] \left[\frac{\partial a_{k,n}}{\partial w_{jk}}\right]$$

Multivariate chain rule

$$\frac{\partial R}{\partial w_{jk}} = \frac{1}{N}\sum_n \left[\sum_l \delta_l \frac{\partial a_{l,n}}{\partial a_{k,n}}\right] [z_{j,n}]$$



Error Backpropagation

Optimize last hidden weights w_{jk}

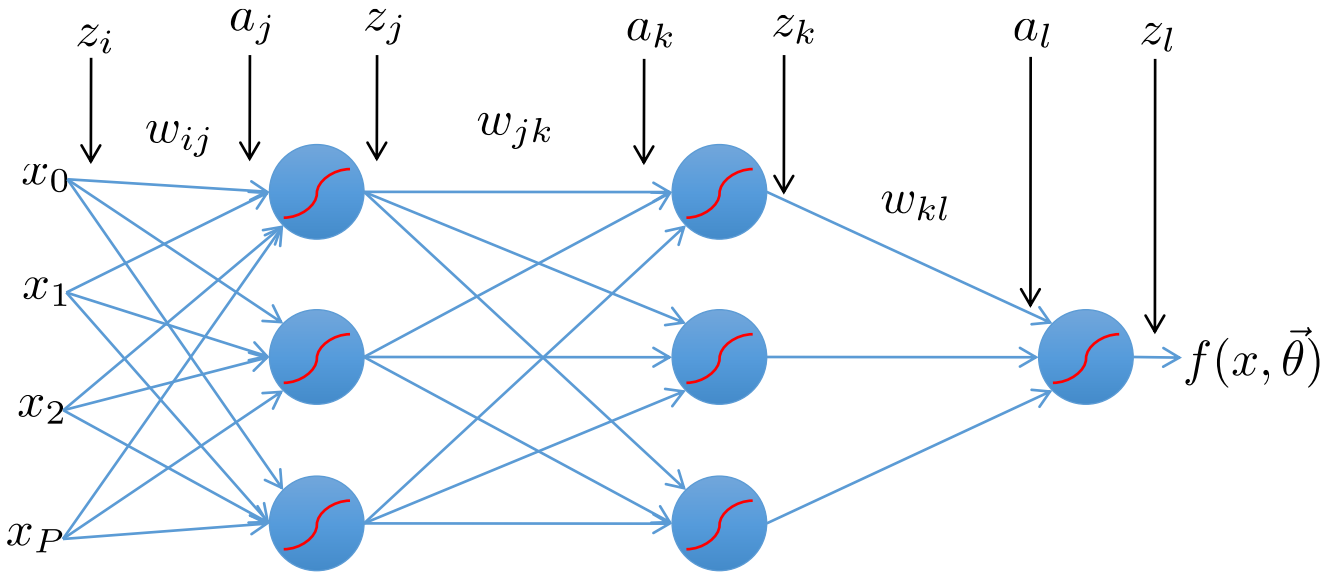
$$\frac{\partial R}{\partial w_{kl}} = \frac{1}{N}\sum_n \delta_{l,n} z_{k,n}$$

$$\frac{\partial R}{\partial w_{jk}} = \frac{1}{N}\sum_n \left[\sum_l \frac{\partial L_n}{\partial a_{l,n}} \frac{\partial a_{l,n}}{\partial a_{k,n}}\right] \left[\frac{\partial a_{k,n}}{\partial w_{jk}}\right]$$

Multivariate chain rule

$$\frac{\partial R}{\partial w_{jk}} = \frac{1}{N}\sum_n \left[\sum_l \delta_l \frac{\partial a_{l,n}}{\partial a_{k,n}}\right] [z_{j,n}]$$

$$a_l = \sum_k w_{kl} g(a_k)$$



Error Backpropagation

Optimize last hidden weights w_{jk}

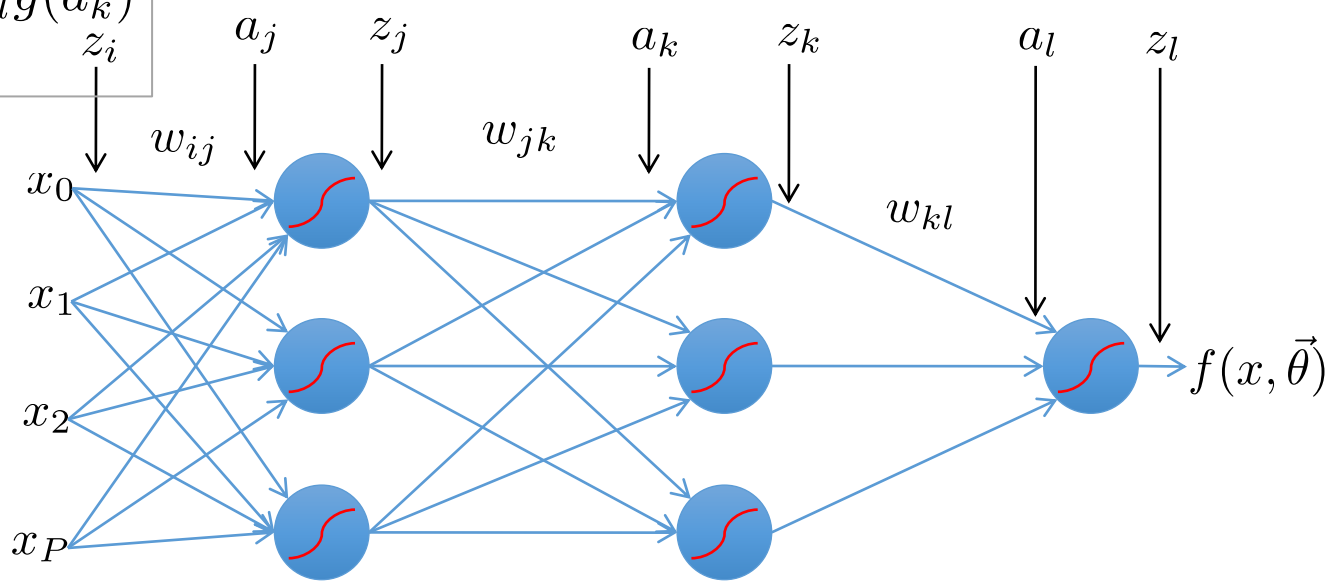
$$\frac{\partial R}{\partial w_{kl}} = \frac{1}{N}\sum_n \delta_{l,n} z_{k,n}$$

$$\frac{\partial R}{\partial w_{jk}} = \frac{1}{N}\sum_n \left[\sum_l \frac{\partial L_n}{\partial a_{l,n}}\frac{\partial a_{l,n}}{\partial a_{k,n}}\right]\left[\frac{\partial a_{k,n}}{\partial w_{jk}}\right]$$

Multivariate chain rule

$$\frac{\partial R}{\partial w_{jk}} = \frac{1}{N}\sum_n \left[\sum_l \delta_l w_{kl} g'(a_{k,n})\right][z_{j,n}] = \frac{1}{N}\sum_n [\delta_{k,n}][z_{j,n}]$$

$$a_l = \sum_k w_{kl} g(a_k)$$



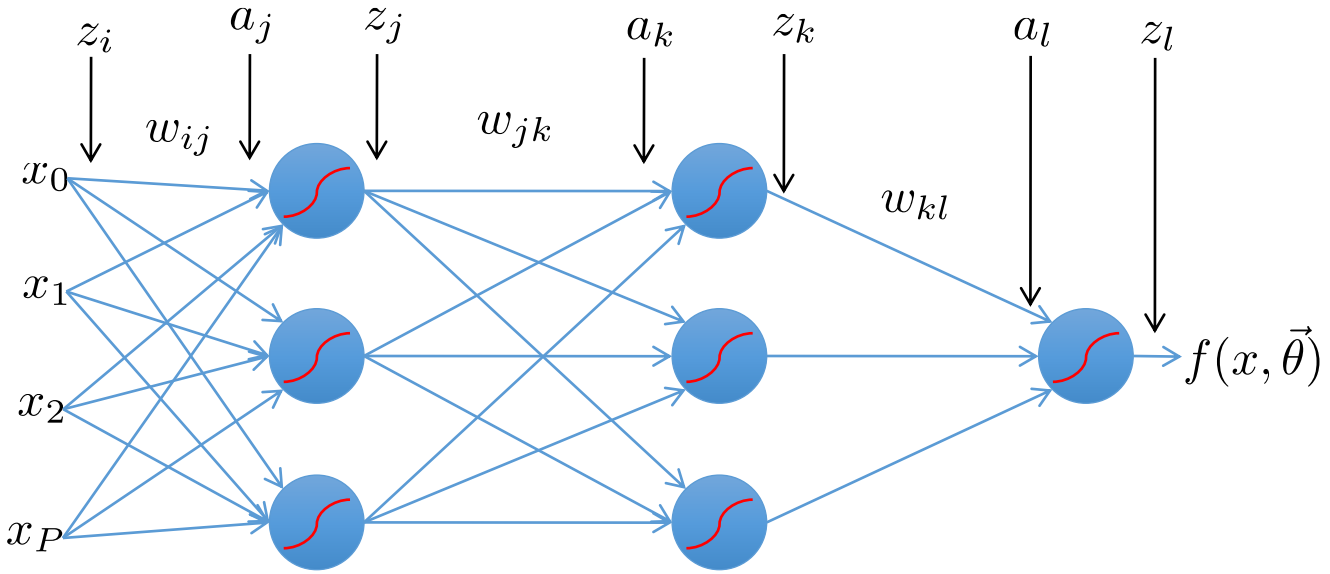
Error Backpropagation

Repeat for all previous layers

$$\frac{\partial R}{\partial w_{kl}} = \frac{1}{N}\sum_n \left[\frac{\partial L_n}{\partial a_{l,n}}\right]\left[\frac{\partial a_{l,n}}{\partial w_{kl}}\right] = \frac{1}{N}\sum_n \left[-(y_n - z_{l,n})g'(a_{l,n})\right] z_{k,n} = \frac{1}{N}\sum_n \delta_{l,n} z_{k,n}$$

$$\frac{\partial R}{\partial w_{jk}} = \frac{1}{N}\sum_n \left[\frac{\partial L_n}{\partial a_{k,n}}\right]\left[\frac{\partial a_{k,n}}{\partial w_{jk}}\right] = \frac{1}{N}\sum_n \left[\sum_l \delta_{l,n} w_{kl} g'(a_{k,n})\right] z_{j,n} = \frac{1}{N}\sum_n \delta_{k,n} z_{j,n}$$

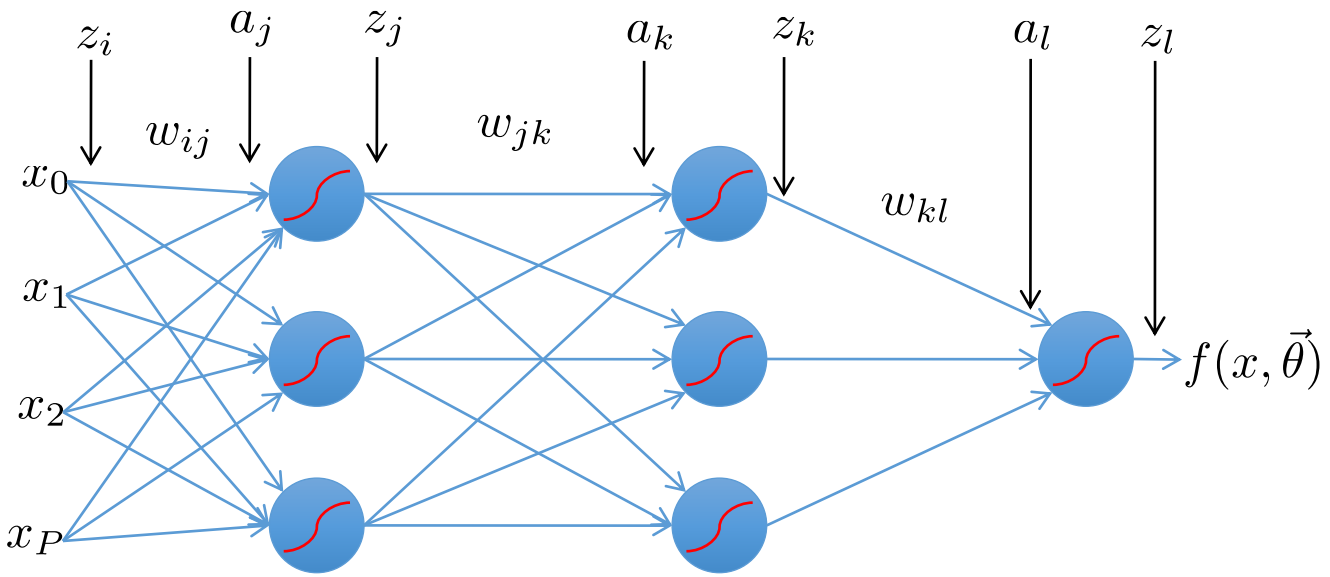
$$\frac{\partial R}{\partial w_{ij}} = \frac{1}{N}\sum_n \left[\frac{\partial L_n}{\partial a_{j,n}}\right]\left[\frac{\partial a_{j,n}}{\partial w_{ij}}\right] = \frac{1}{N}\sum_n \left[\sum_k \delta_{k,n} w_{jk} g'(a_{j,n})\right] z_{i,n} = \frac{1}{N}\sum_n \delta_{j,n} z_{i,n}$$



Error Backpropagation

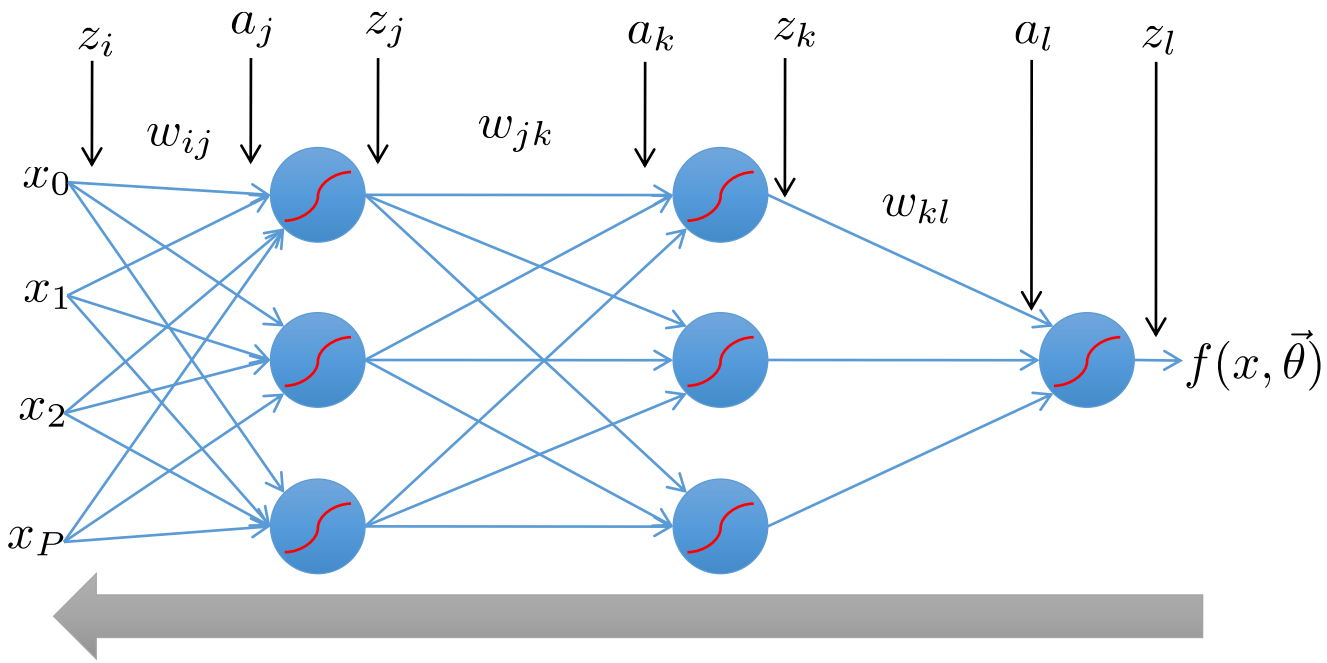
Now that we have well defined gradients for each parameter, update using Gradient Descent

$$
\begin{aligned}
w_{ij}^{t+1} &= w_{ij}^{t} - \eta \frac{\partial R}{w_{ij}} \\
w_{jk}^{t+1} &= w_{jk}^{t} - \eta \frac{\partial R}{w_{jk}} \\
w_{kl}^{t+1} &= w_{kl}^{t} - \eta \frac{\partial R}{w_{kl}}
\end{aligned}
$$



Error Back-propagation

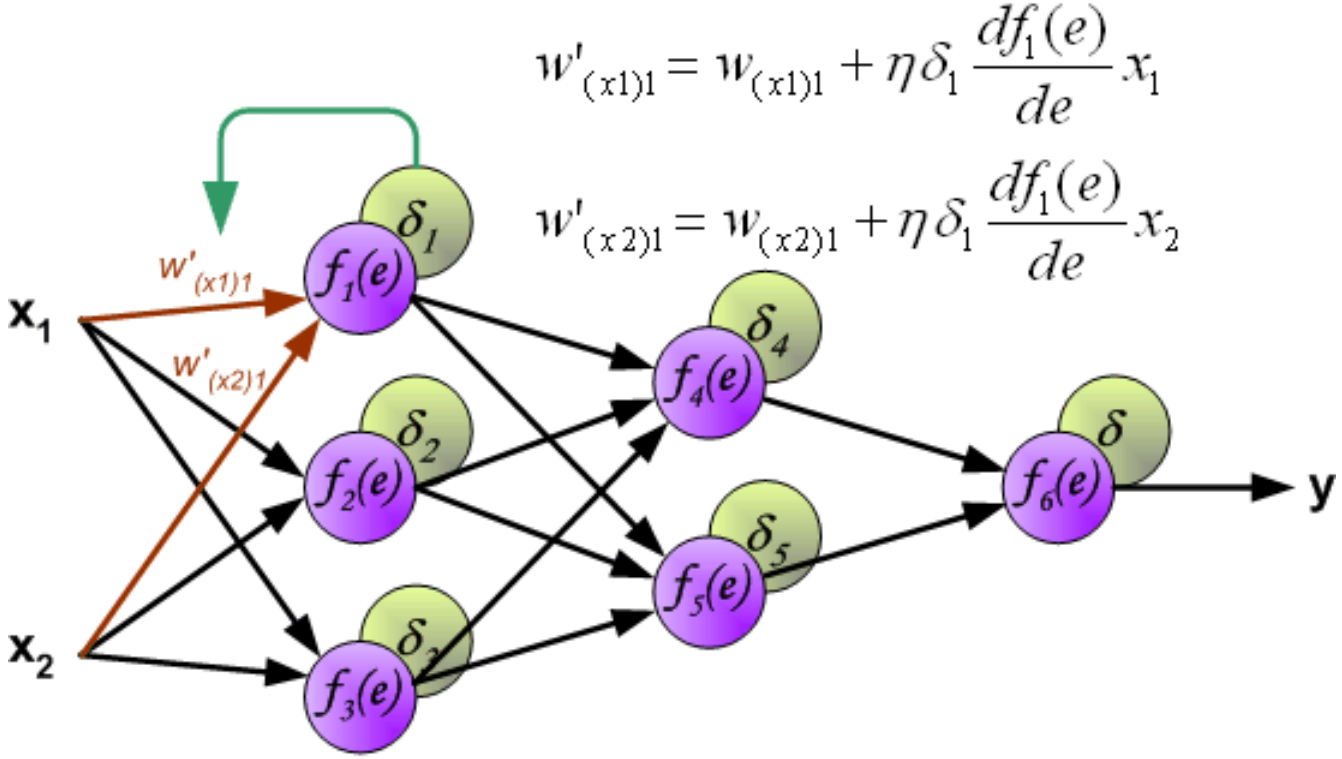
- Error backpropagation unravels the multivariate chain rule and solves the gradient for each partial component separately.
- The target values for each layer come from the next layer.
- This feeds the errors back along the network.



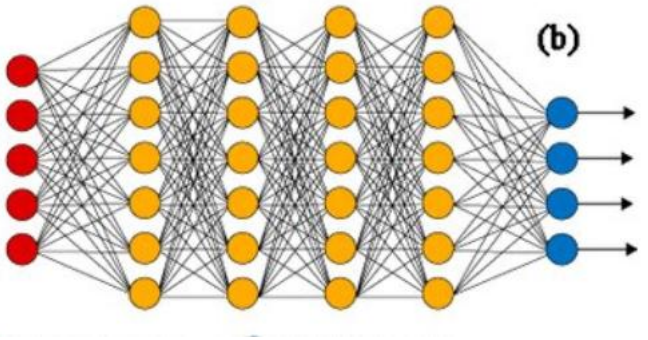
Learning with error backpropagation

- randomly initialize parameters (weights)
- compute error on the output
- compute contributions to error, δ_n, on each step backwards

- step
- epoch
- batch
- minibatch



Defining a network topology



- Decide the network topology:
  Specify # of units in the *input layer*, # of *hidden layers* (if > 1), # of units in *each hidden layer*, and # of units in the *output layer*
- Normalize the input values for each attribute measured in the training tuples to [0.0—1.0]
- One input unit per discrete attribute value, 1-hot encoded
- For classification and more than two classes, one output unit per class is used
- Once a network has been trained and its accuracy is still unacceptable, repeat the training process with a *different network topology* or a *different set of initial weights*

Neural network: strengths and weaknesses

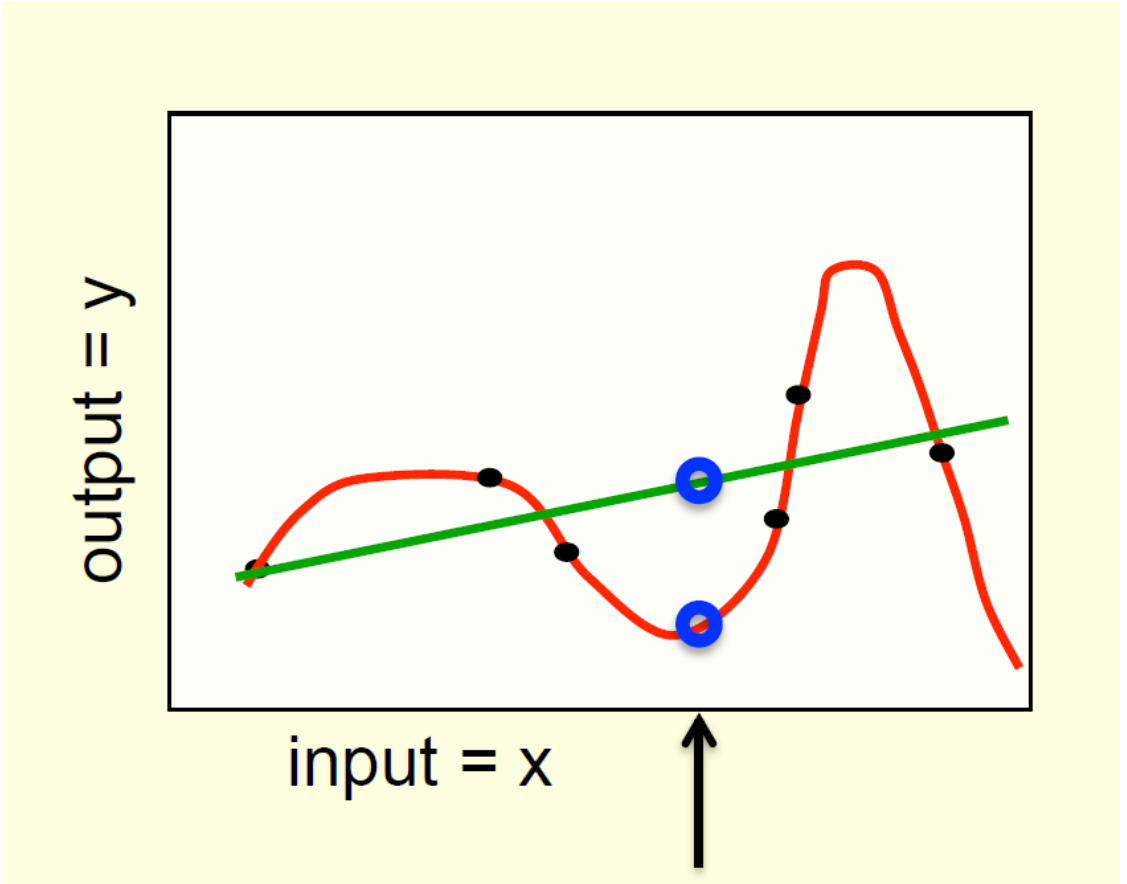
- Weakness
  - Long training time
  - Require a number of parameters typically best determined empirically, e.g., the network topology or “structure.”
  - Poor interpretability: difficult to interpret the symbolic meaning behind the learned weights and of “hidden units” in the network
  - Easy to overfit without an evaluation set
- Strength
  - High tolerance to noisy data
  - Good generalization to untrained patterns
  - Well-suited for continuous-valued inputs *and outputs*
  - Algorithms are inherently parallel
  - Builds more advanced representation
  - Successful on an array of real-world data, especially images, text, and time-series, e.g., one of the early successful deep networks was applied to hand-written letters
  - Techniques exist for the extraction of explanations from trained (small) neural networks

Efficiency and interpretability

- Efficiency of backpropagation:
  Each epoch (one iteration through the training set) takes O(|D| * *w*), with |D| training instances and *w* weights, but # of epochs can be exponential to n, the number of inputs, in worst case (not in practice)
- For easier comprehension: Rule extraction by network pruning
  - Simplify the network structure by removing weighted links that have the least effect on the trained network
  - Then perform link, unit, or activation value clustering
  - The set of input and activation values are studied to derive rules describing the relationship between the input and hidden unit layers
- Sensitivity analysis: assess the impact that a given input variable has on a network output. The knowledge gained from this analysis can be represented in rules
- Recent attempts tend to learn interpretation along with learning

Overfitting and model complexity

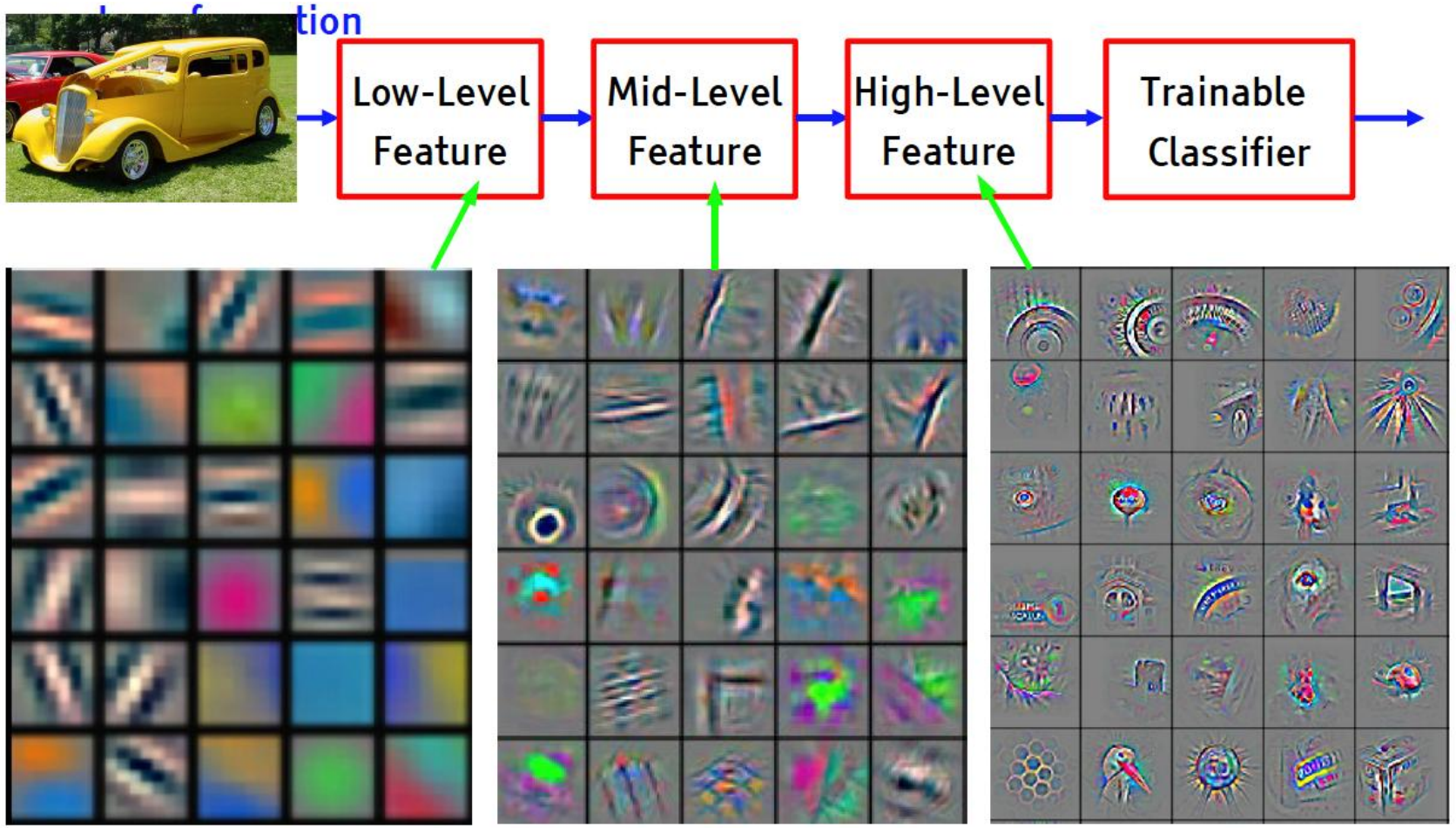
- which curve is more plausible given the data?
- overfitting
- neural nets are especially prone to overfitting
- why?



Prevention of overfitting

- Evaluation set
- Weight-decay
- Weight-sharing
- Early stopping
- Model averaging
- Bayesian fitting of neural nets:
  - distributions instead of weights,
  - inference as sampling from distributions
- Dropout
- Generative pre-training
- etc.

Deep learning = learning of hierarchical represenation



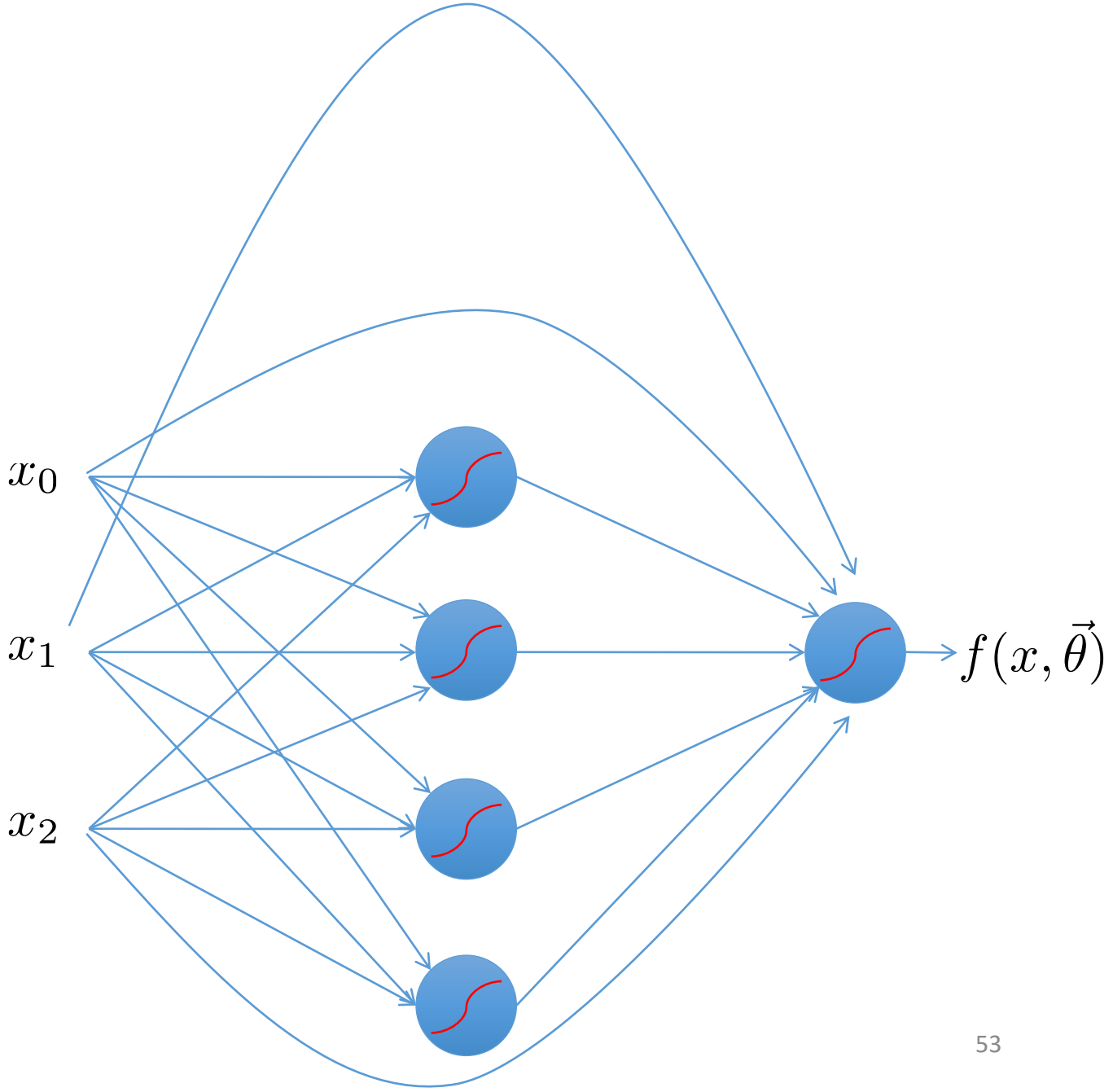
Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

Types of NN architectures

- Historically, feed-forward networks were the most commonly used; here, neurons are activated progressively through layers from input to output
- However, we often combine different types of layers
- Examples of other architectures: recurrent, convolutional, transformer

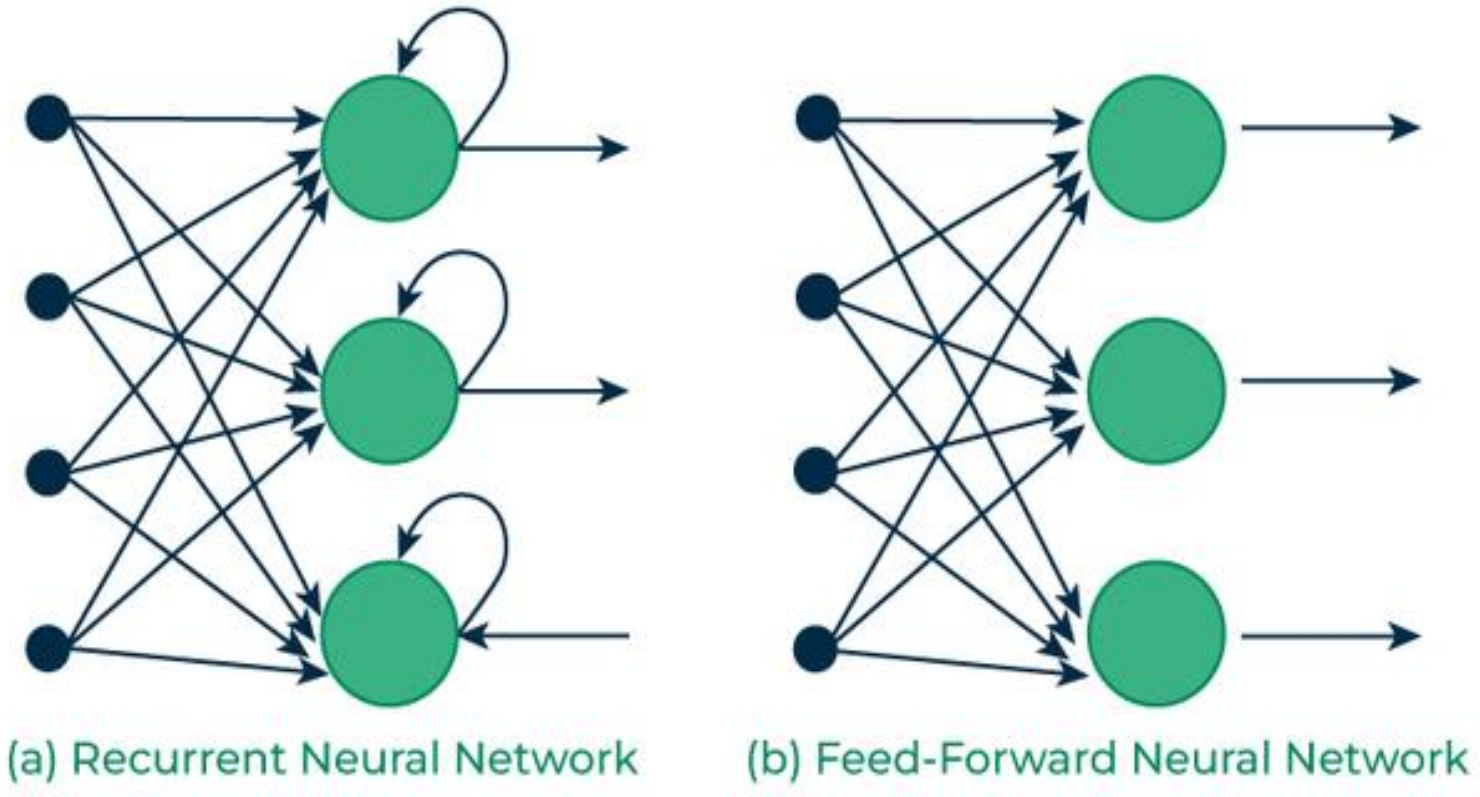
Another option: Level jumping or Skip-connections

- prevents vanishing gradients



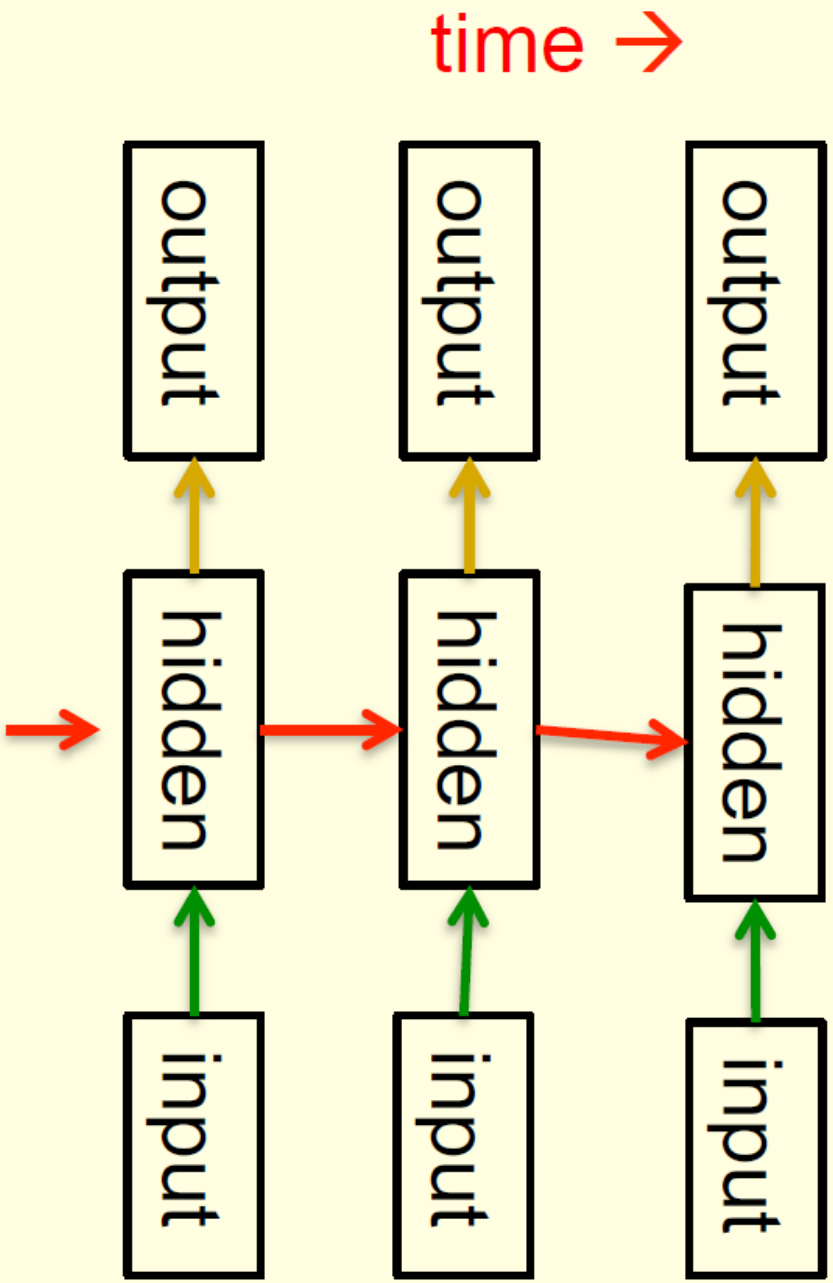
Recurrent networks

- back connections
- biologically more realistic
- store information from the past
- more difficult to learn

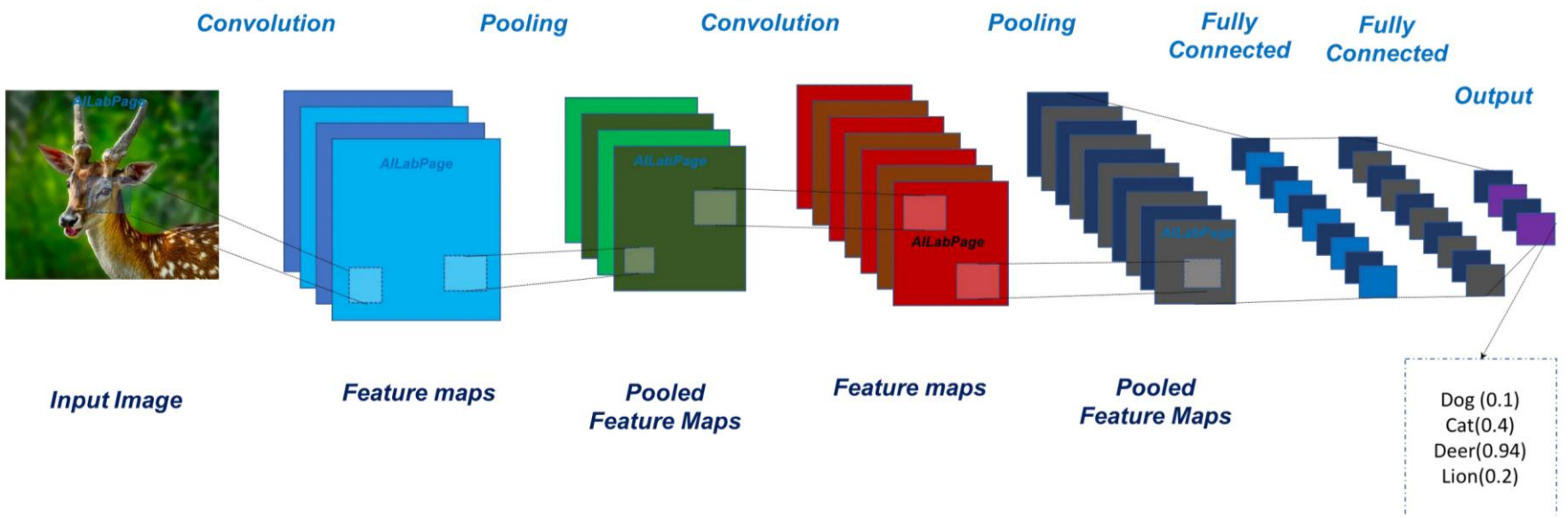


Recurrent networks for sequence learning

- unrolled network
- equivalent to deep networks with one hidden level per time slot
- but: hidden layers share weight (less parameters)



Convolutional neural networks (CNN)



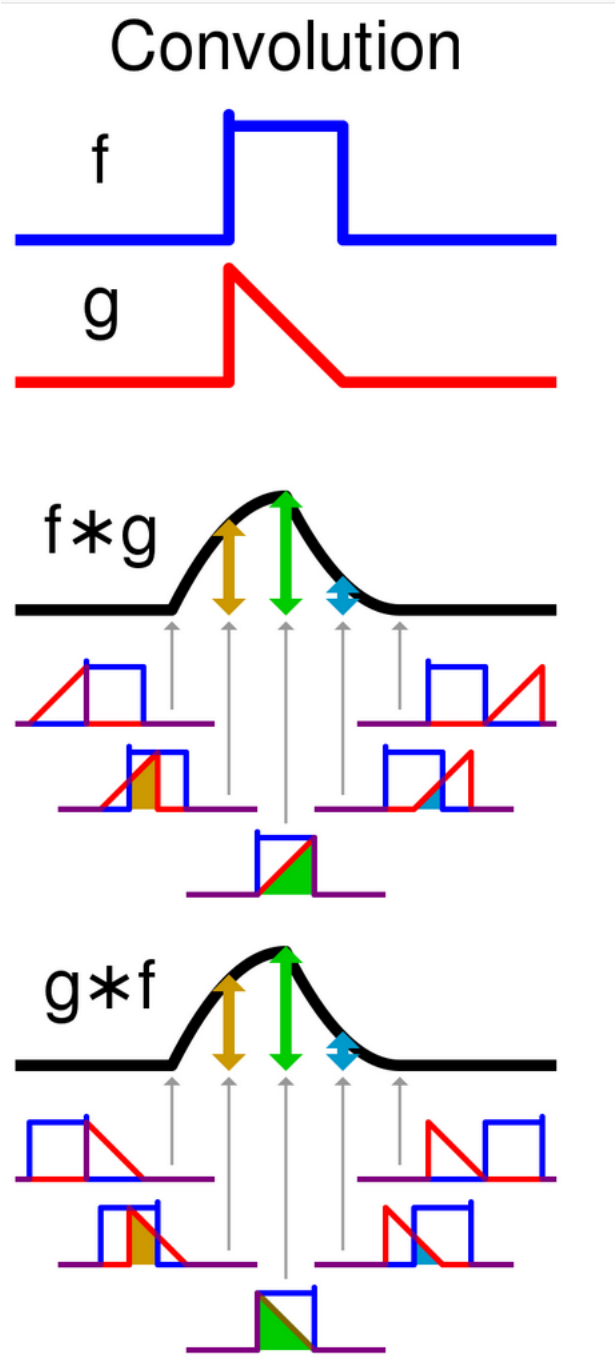
Convolution

- an operation on two functions (f and g) that produces a third function expressing how the shape of one is modified by the other.

$$(f * g)(t) \triangleq \int_{-\infty}^{\infty} f(\tau) g(t - \tau)\, d\tau.$$

- for discrete functions

$$(f * g)[n] = \sum_{m=-\infty}^{\infty} f[m] g[n - m]$$

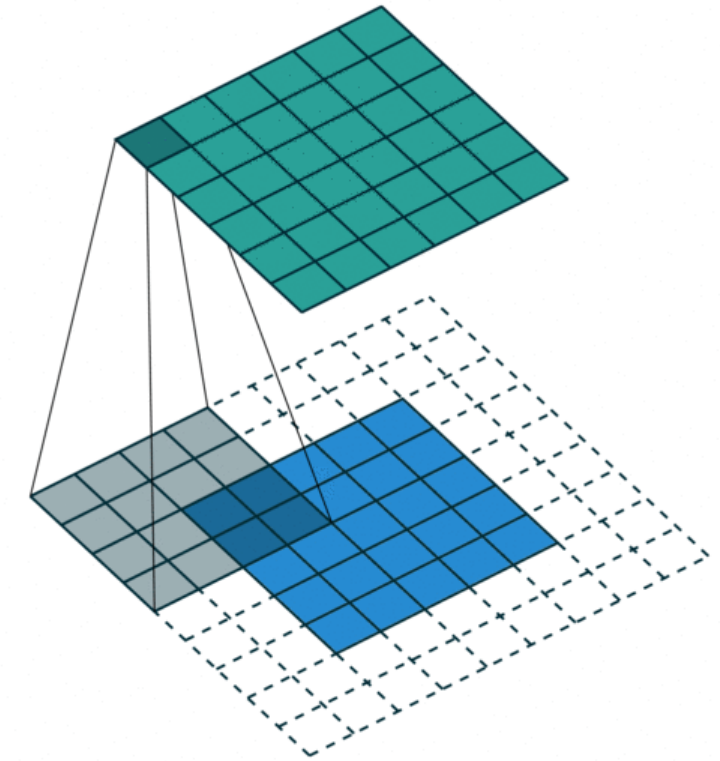


Convolutional Neural Network (CNN)

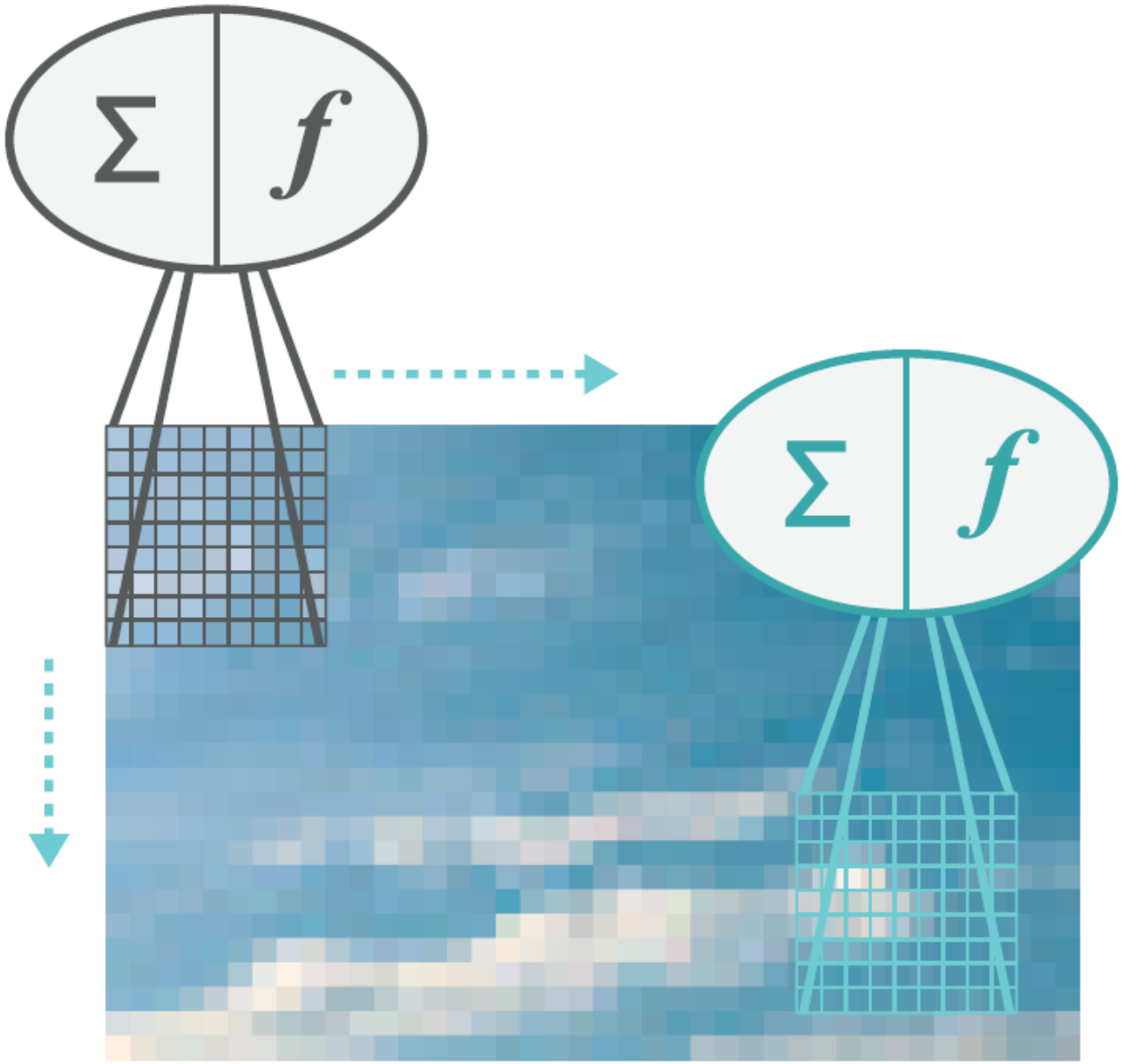
- Convolutional Neural Networks are inspired by mammalian visual cortex.
  - The visual cortex contains a complex arrangement of cells, which are sensitive to small sub-regions of the visual field, called a receptive field. These cells act as local filters over the input space and are well-suited to exploit the strong spatially local correlation present in natural images.
  - Two basic cell types:
    - Simple cells respond maximally to specific edge-like patterns within their receptive field.
    - Complex cells have larger receptive fields and are locally invariant to the exact position of the pattern.

Convolutional neural networks (CNN)

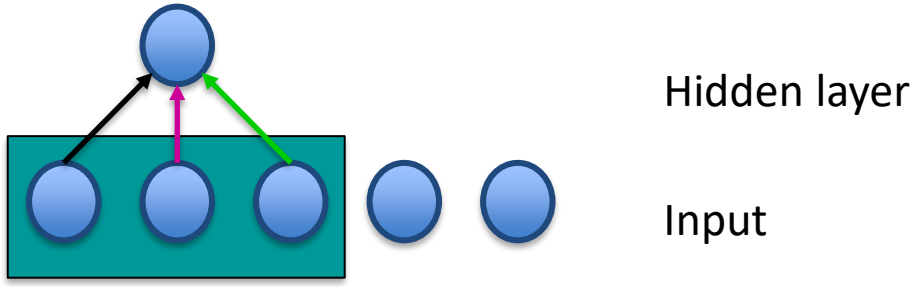
- a successful approach in image analysis, also used in language processing
- idea: many copies of small detectors used all over the image, recursively combined,
- detectors are learned, combinations are learned



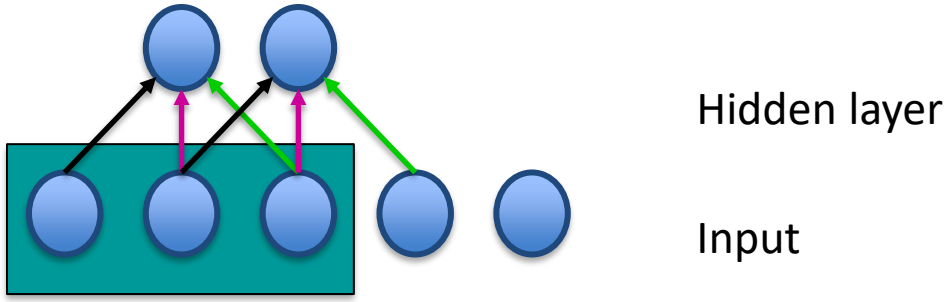
2d convolution for images



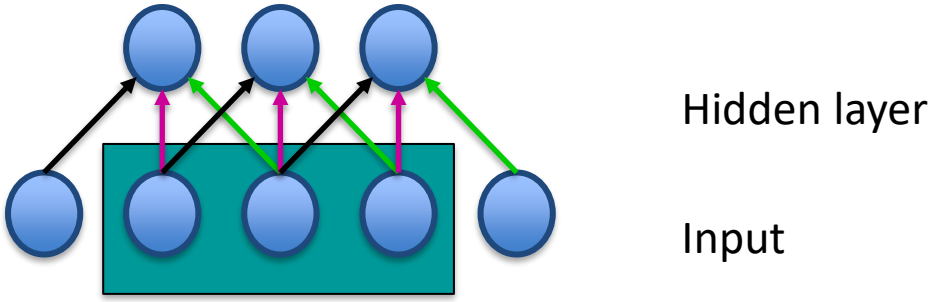
Basic Idea of CNNs



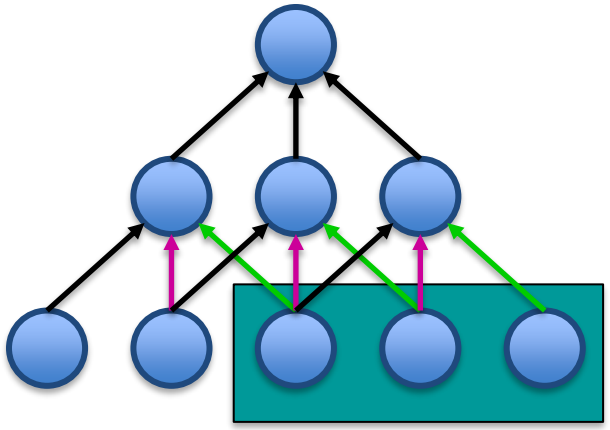
Basic Idea of CNNs



Basic Idea of CNNs



Basic Idea of CNNs

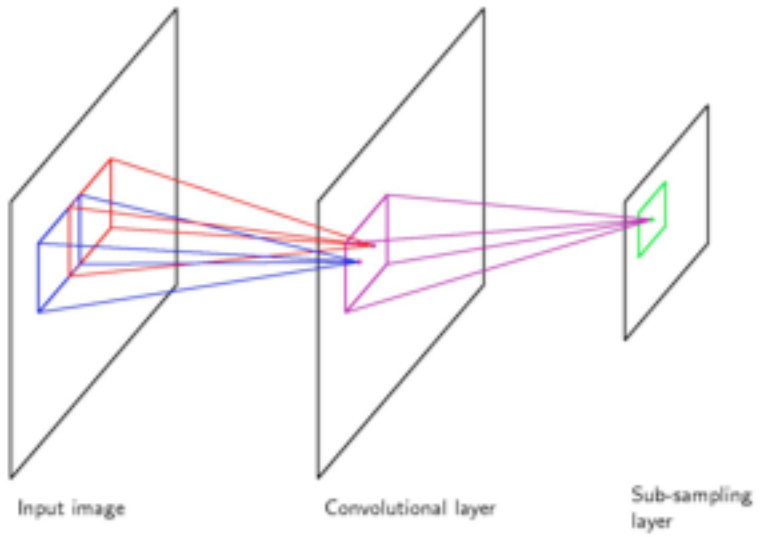


Hidden layer

Input

Convolutional Network

- The network is not fully connected.
- Different nodes are responsible for different regions of the image.
- This allows for robustness to transformations.
- Convolution is combined with subsampling.



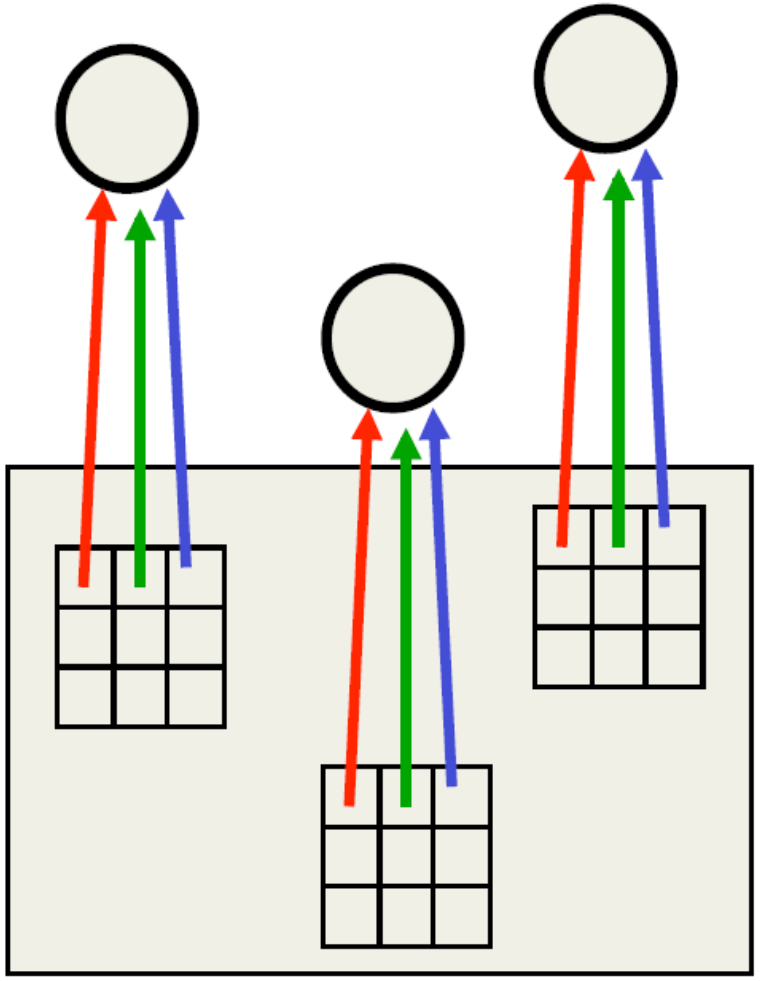
CNN Architecture: Convolutional Layer

- The core layer of CNNs
- The convolutional layer consists of a set of filters.
  - Each filter covers a spatially small portion of the input data.
- Each filter is convolved across the dimensions of the input data, producing a multidimensional feature map.
  - As we convolve the filter, we are computing the dot product between the parameters of the filter and the input.
- Intuition: the network will learn filters that activate when they see some specific type of feature at some spatial position in the input.
- The key architectural characteristics of the convolutional layer is local connectivity and shared weights.

Neural implementation of convolution

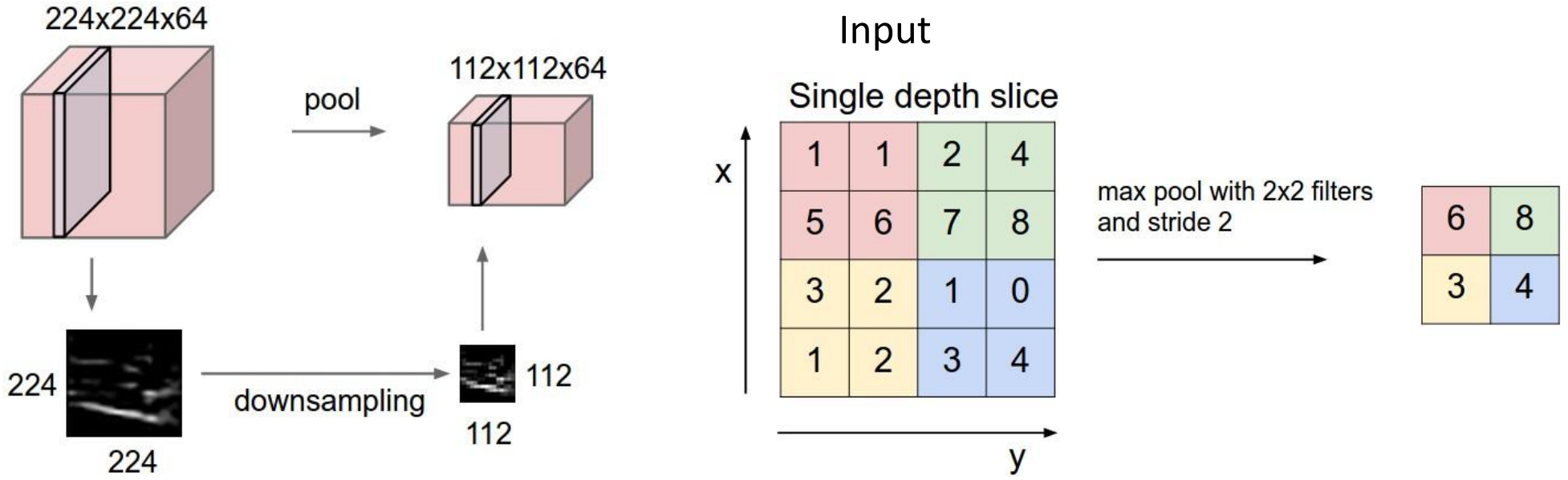
- weights of the same colors have equal weights
- adapted backpropagation

- images: 2d convolution
- languages: 1d convolution



CNN Architecture: Pooling Layer

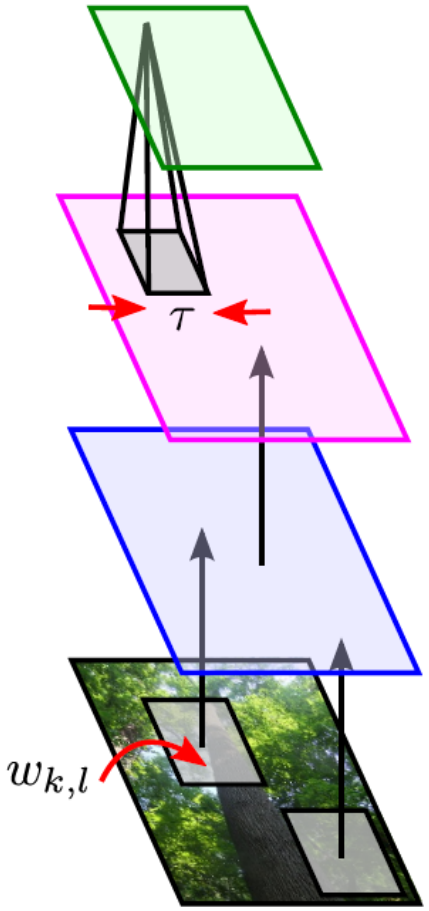
- Intuition: to progressively reduce the spatial size of the representation, to reduce the amount of parameters and computation in the network, and hence to also control overfitting
- Pooling partitions the input image into a set of non-overlapping rectangles and, for each such sub-region, outputs the maximum value of the features in that region.



CNN: pooling

- reduces the number of connections to the next layer (prevents the excessive number of parameters, speeds-up learning, reduces overfitting)
- max-pooling, min-pooling, average-pooling
- the problem: after several layers of pooling, we lose the information about the exact location of the recognized pattern and about spatial relations between different patterns and features, e.g., a nose on a forehead

Building-blocks for CNN's



$$x_{i,j} = \max_{|k|<\tau,|l|<\tau} y_{i-k,j-l}$$

mean or subsample also used

pooling stage

Feature maps of a larger region are combined.

$$y_{i,j} = f(a_{i,j})$$

e.g. $f(a) = [a]_+$

$f(a) = \text{sigmoid}(a)$

non-linear stage

Feature maps are trained with neurons.

Shared weights

$$a_{i,j} = \sum_{k,l} w_{k,l} z_{i-k,j-l}$$

only parameters

convolutional stage

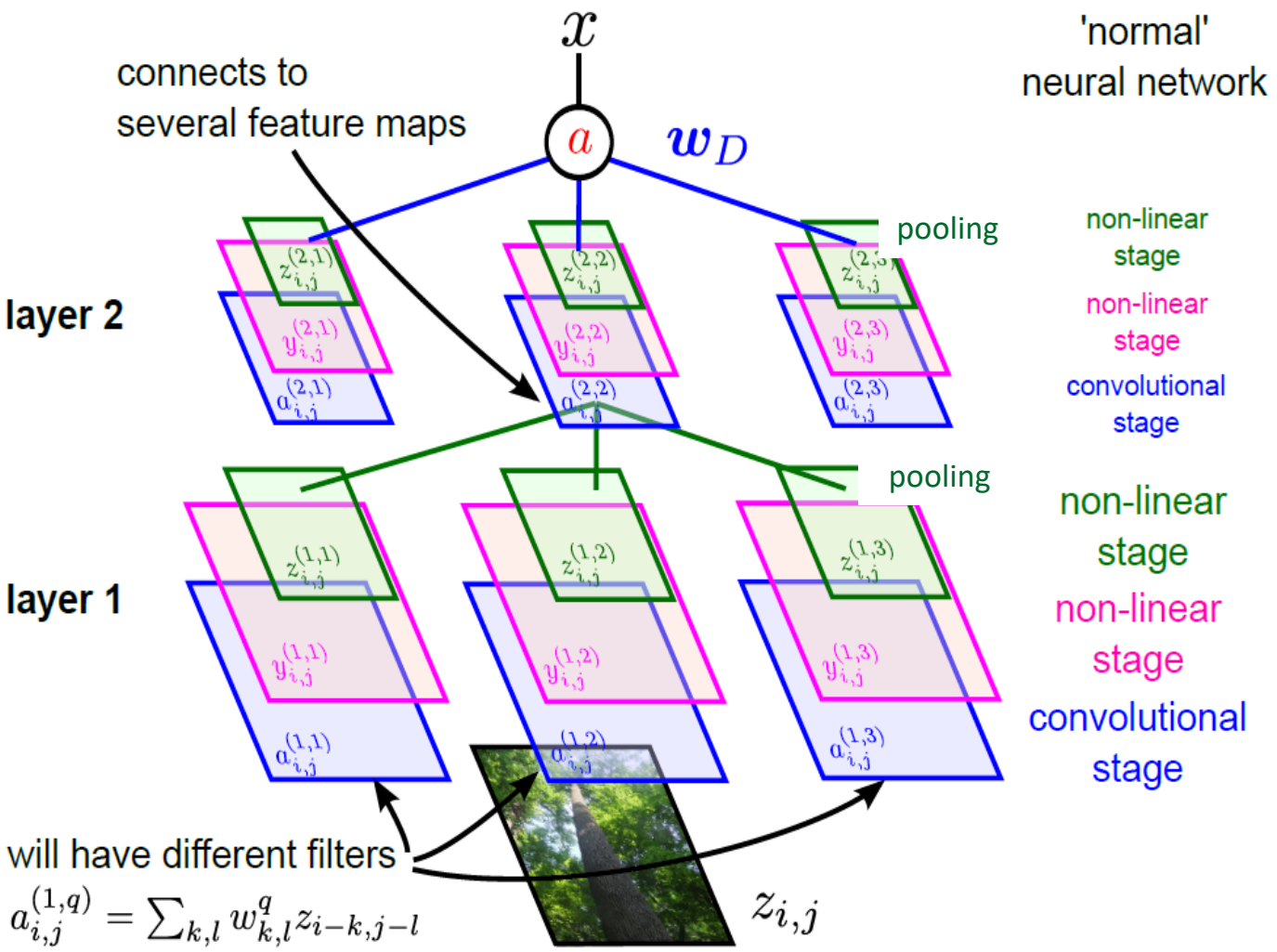
Each sub-region yields a feature map, representing its feature.

$z_{i,j}$

input image

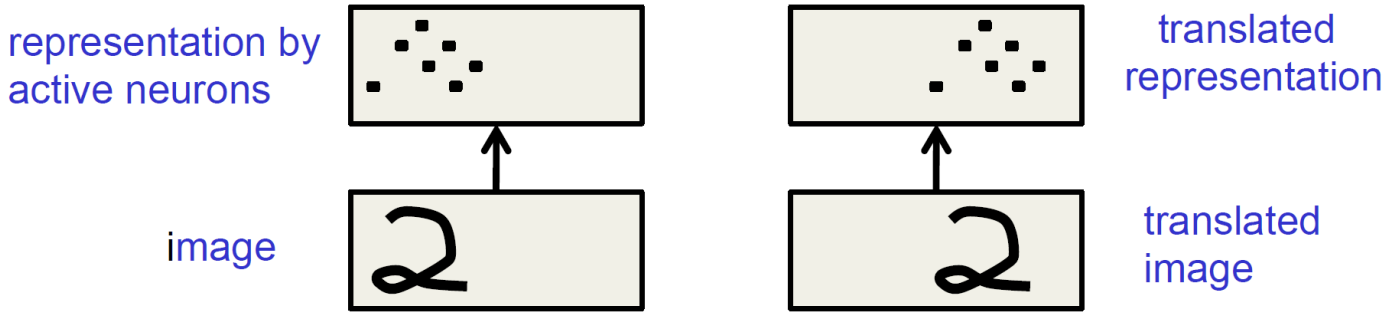
Images are segmented into sub-regions.

Full CNN


x
connects to
several feature maps
a
w_D
'normal'
neural network
pooling
non-linear
stage
non-linear
stage
convolutional
stage
layer 2
pooling
layer 1
non-linear
stage
non-linear
stage
convolutional
stage
will have different filters
$a_{i,j}^{(1,q)} = \sum_{k,l} w_{k,l}^{q} z_{i-k,j-l}$
$z_{i,j}$

Convolutional networks: illustration on image recognition

- a useful feature is learned and used on several positions
- prevents dimension hopping
- max-pooling

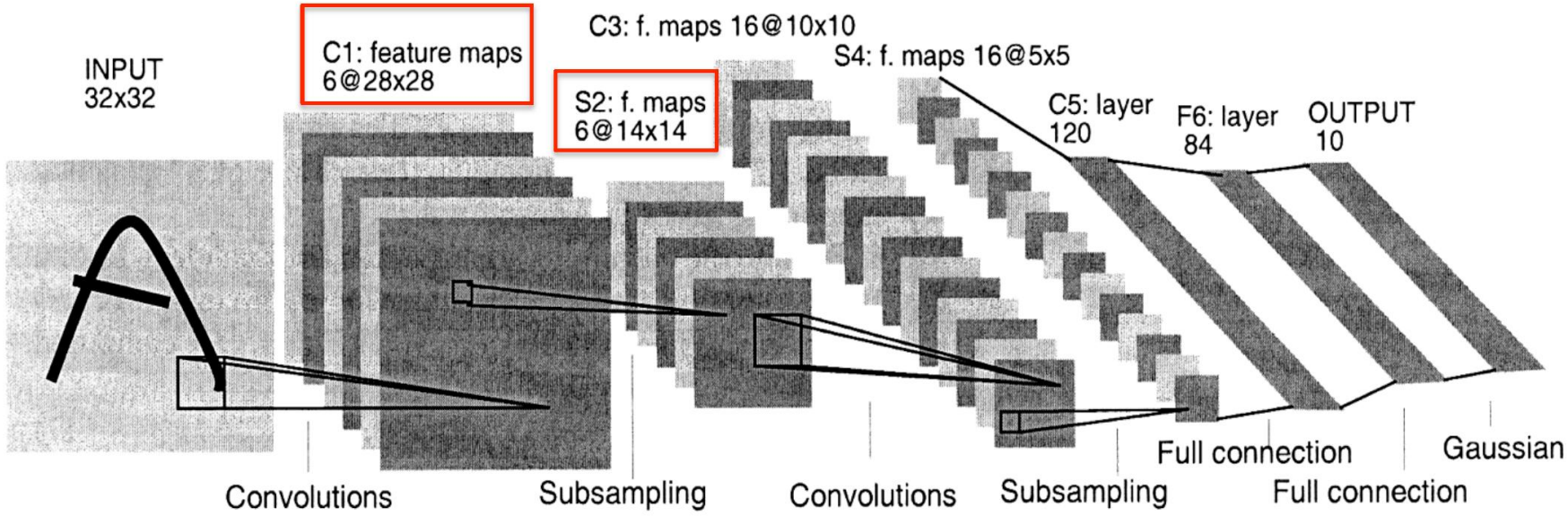


CNN early success: LeNet

- handwritten digit recognition by Yann LeCun,
- several hidden layers
- several convolutional filters
- pooling
- several other tricks

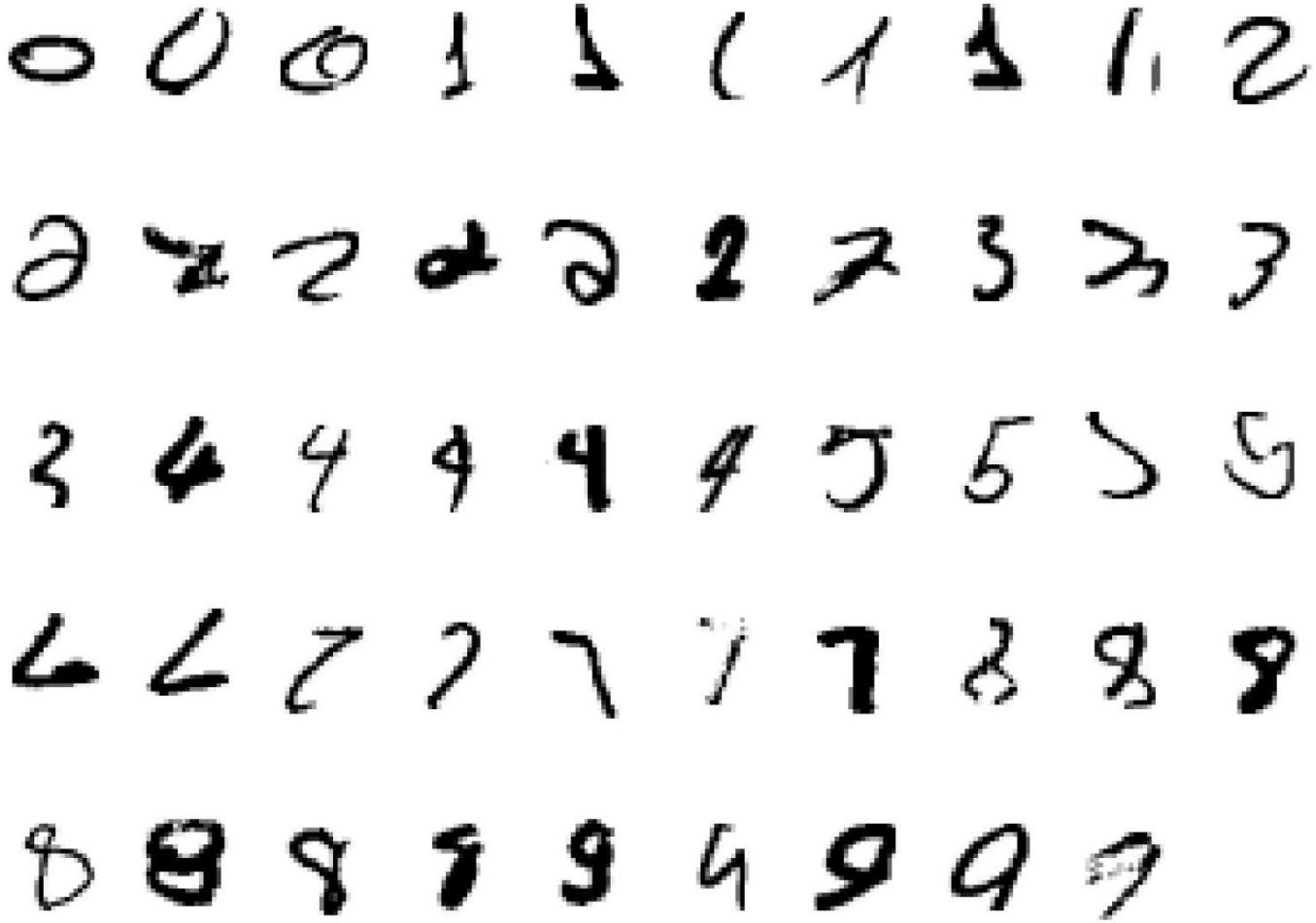
LeNet5 architecture

- handwritten digit recognition



Hand-written Digit Recognition

- Input:



Errors of LeNet5

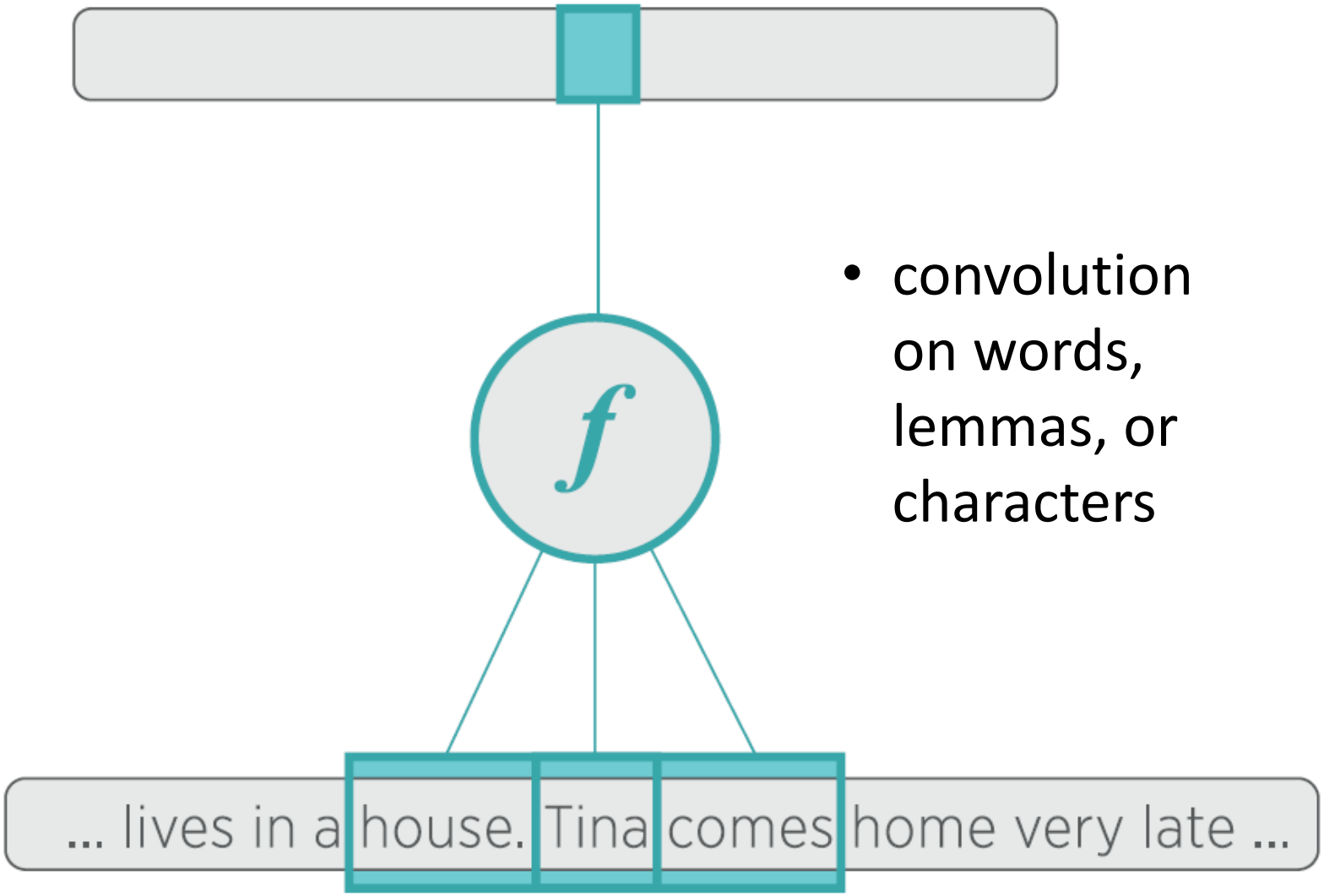


- 80 errors in 10,000 test cases

Benefits of CNNs

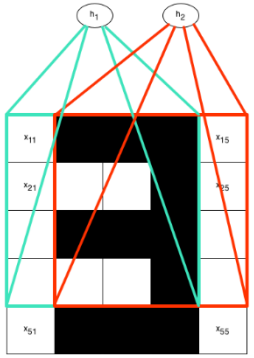
- The number of weights can be much less than 1 million for a 1 mega pixel image.
- The small number of weights can use different parts of the image as training data. Thus we have several orders of magnitude more data to train the fewer number of weights.
- We get translation invariance for free.
- Fewer parameters take less memory and thus all the computations can be carried out in memory in a GPU or across multiple processors.

1d convolution for text

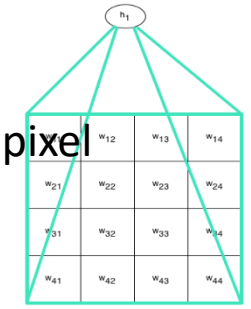


- convolution on words, lemmas, or characters

Example: what the following CNN returns 1/2



We have a convolutional neural network for images of 5 by 5 pixels.
In this network, each hidden unit is connected to a different 4 x 4 region of the input image:
The first hidden unit, h1, is connected to the upper left 4x4 portion of the input image (as shown).
The second hidden unit, h2, is connected to the upper right 4x4 portion of the input image (as shown).
The third hidden unit, h3, is connected to the lower left 4x4 portion of the input image (not shown in the diagram).
The fourth hidden unit, h4, is connected to the lower right 4x4 portion of the input image (not shown in the diagram).
Because it's a convolutional network, the weights (connection strengths) are the same for all hidden units: the only difference between the hidden units is that each of them connects to a different part of the input image.



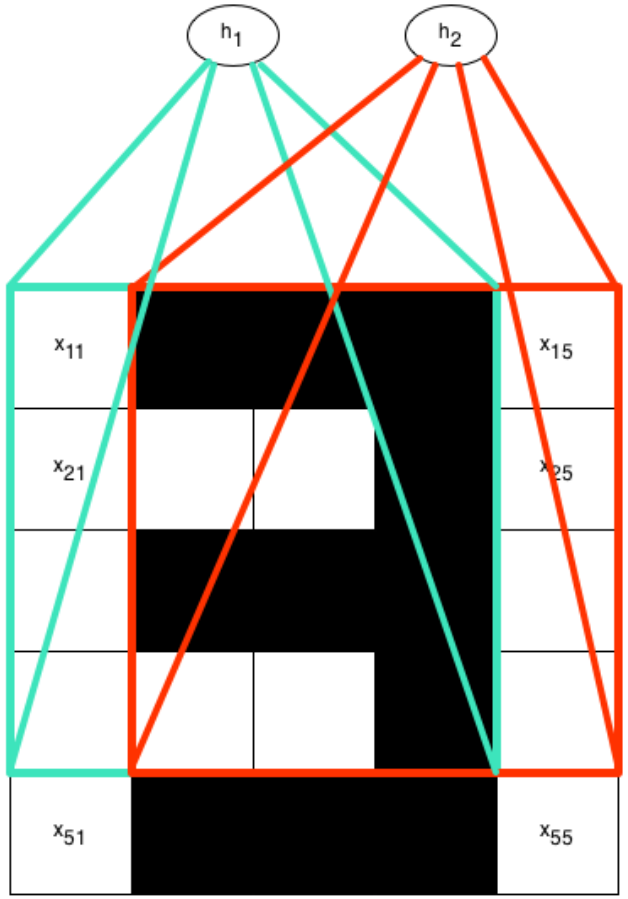
In the second diagram, we show the array of weights, which are the same for each of the four hidden units.
For h1, weight w11 is connected to the top-left pixel, i.e. x11, while for hidden unit h2, weight w11 connects to the pixel that is one to the right of the top left pixel, i.e. x12.

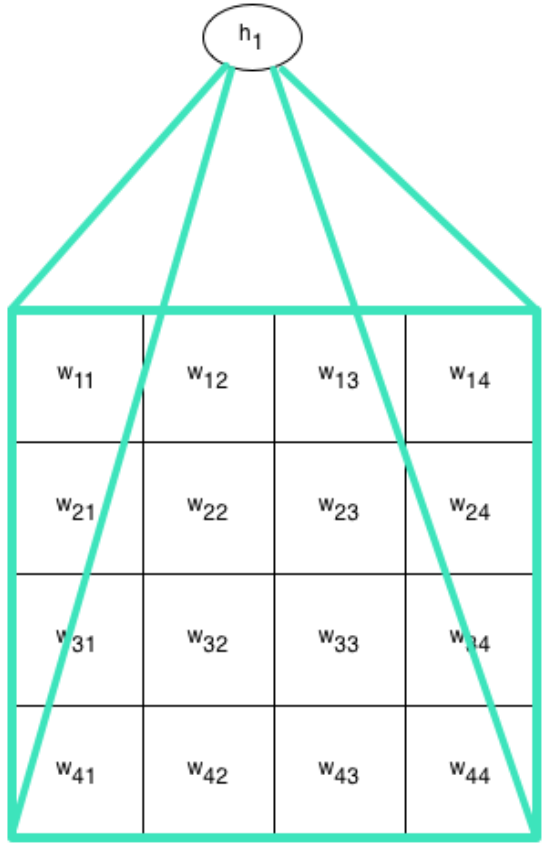
Imagine that for some training case, we have an input image where each of the black pixels in the top diagram has value 1, and each of the white ones has value 0. Notice that the image shows a "3" in pixels.

The network has no biases. The weights of the network are given as follows:
w11=1w12=1w13=1w14=0w21=0w22=0w23=1w24=0w31=1w32=1w33=1w34=0w41=0w42=0w43=1w44=0
The hidden units are *linear*.

For the training case with that "3" input image, what is the output y1, y2, y3, y4 of each of the four hidden units?

Example: what the following CNN returns 2/2

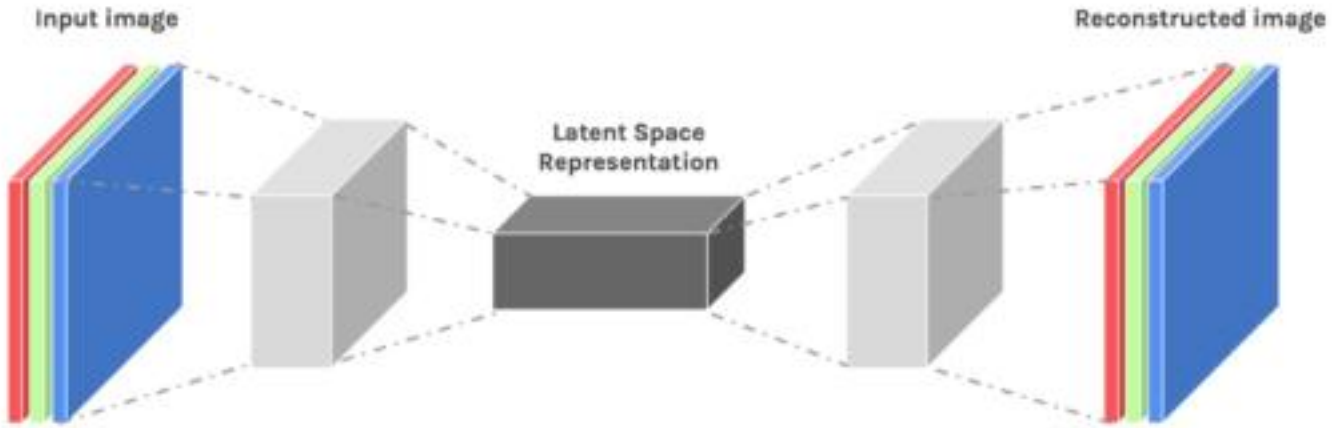




$$w_{11} = 1 \quad w_{12} = 1 \quad w_{13} = 1 \quad w_{14} = 0$$
$$w_{21} = 0 \quad w_{22} = 0 \quad w_{23} = 1 \quad w_{24} = 0$$
$$w_{31} = 1 \quad w_{32} = 1 \quad w_{33} = 1 \quad w_{34} = 0$$
$$w_{41} = 0 \quad w_{42} = 0 \quad w_{43} = 1 \quad w_{44} = 0$$

Autoencoders

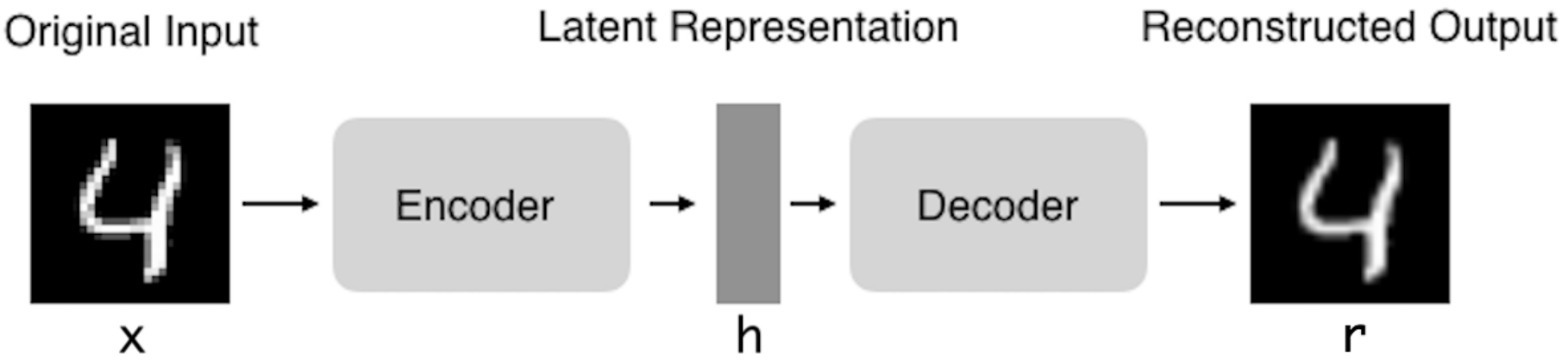
- Autoencoders are designed to reproduce their input, especially for images.
- The key point is to reproduce the input from a learned encoding.
- The loss function is the reproduction error



https://www.edureka.co/blog/autoencoders-tutorial/

Autoencoders: structure

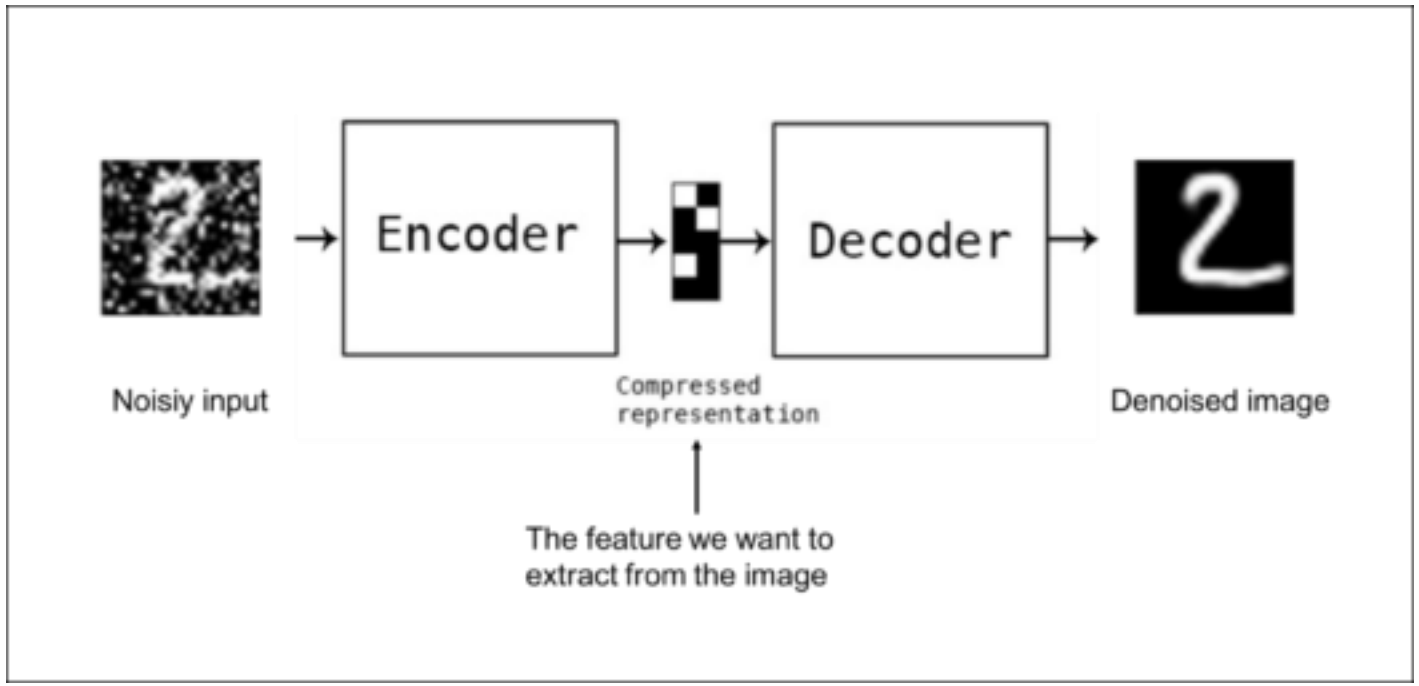
- Encoder: compress input into a latent-space of usually smaller dimension. h = f(x)
- Decoder: reconstruct input from the latent space. r = g(f(x)) with r as close to x as possible



https://towardsdatascience.com/deep-inside-autoencoders-7e41f319999f

Autoencoder applications: denoising

- Denoising: input clean image + noise and train to reproduce the clean image.



https://www.edureka.co/blog/autoencoders-tutorial/

Denoising autoencoders

- Basic autoencoder trains to minimize the loss between x and the reconstruction g(f(x)).
- Denoising autoencoders train to minimize the loss between x and g(f(x+w)), where w is random noise.
- Same possible architectures, different training data.



https://blog.keras.io/building-autoencoders-in-keras.html

Autoencoder applications: colorization

- Image colorization: input black and white and train to produce color images



https://www.edureka.co/blog/autoencoders-tutorial/

Autoencoder applications: watermark removal

- Watermark removal



https://www.edureka.co/blog/autoencoders-tutorial/

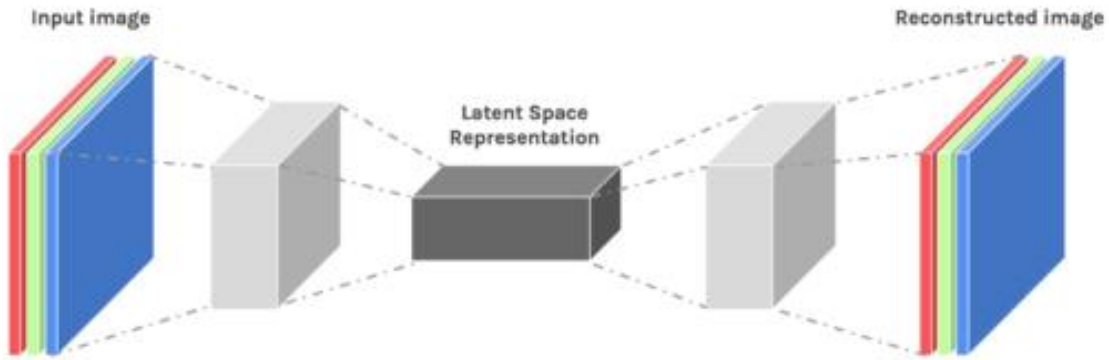
Properties of autoencoders

- **Data-specific**: Autoencoders are only able to compress data similar to what they have been trained on.
- **Lossy:** The decompressed outputs will be degraded compared to the original inputs.
- **Learned automatically from examples:** It is easy to train specialized instances of the algorithm that will perform well on a specific type of input.

https://www.edureka.co/blog/autoencoders-tutorial/

Bottleneck layer (undercomplete)

- Suppose input images are n x n and the latent space is $m < n \times n$.
- Then the latent space is not sufficient to reproduce all images.
- Needs to learn an encoding that captures the important features in training data, sufficient for approximate reconstruction.



GANs

- **Generative**
- Learn a generative model

- **Adversarial**
- Trained in an adversarial setting

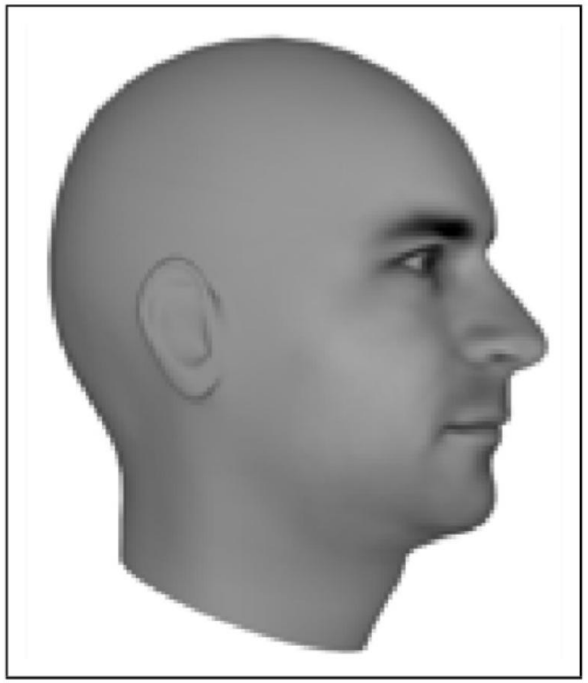
- **Networks**
- Use Deep Neural Networks

Why Generative Models?

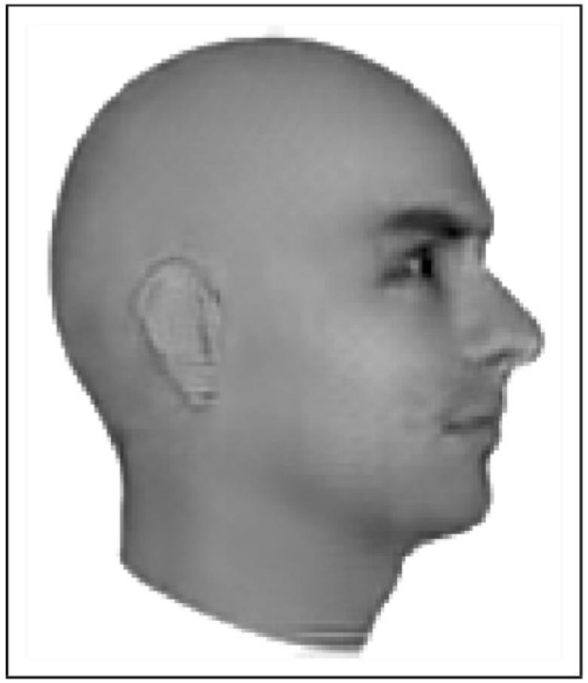
- We have only seen discriminative models so far
- Given an image X, predict a label Y
- Estimates P(Y|X)
- Discriminative models have several key limitations
- Cannot model P(X), i.e. the probability of seeing a certain image
- Thus, can't sample from P(X), i.e. can't generate new images
- Generative models (in general) cope with all of above
- Can model P(X)
- Can generate new images

What GANs can do

Ground Truth



Adversarial



Lotter, William, Gabriel Kreiman, and David Cox. "Unsupervised learning of visual structure using predictive generative networks." *arXiv preprint arXiv:1511.06380* (2015).

GANs in action

Which one is Computer generated?



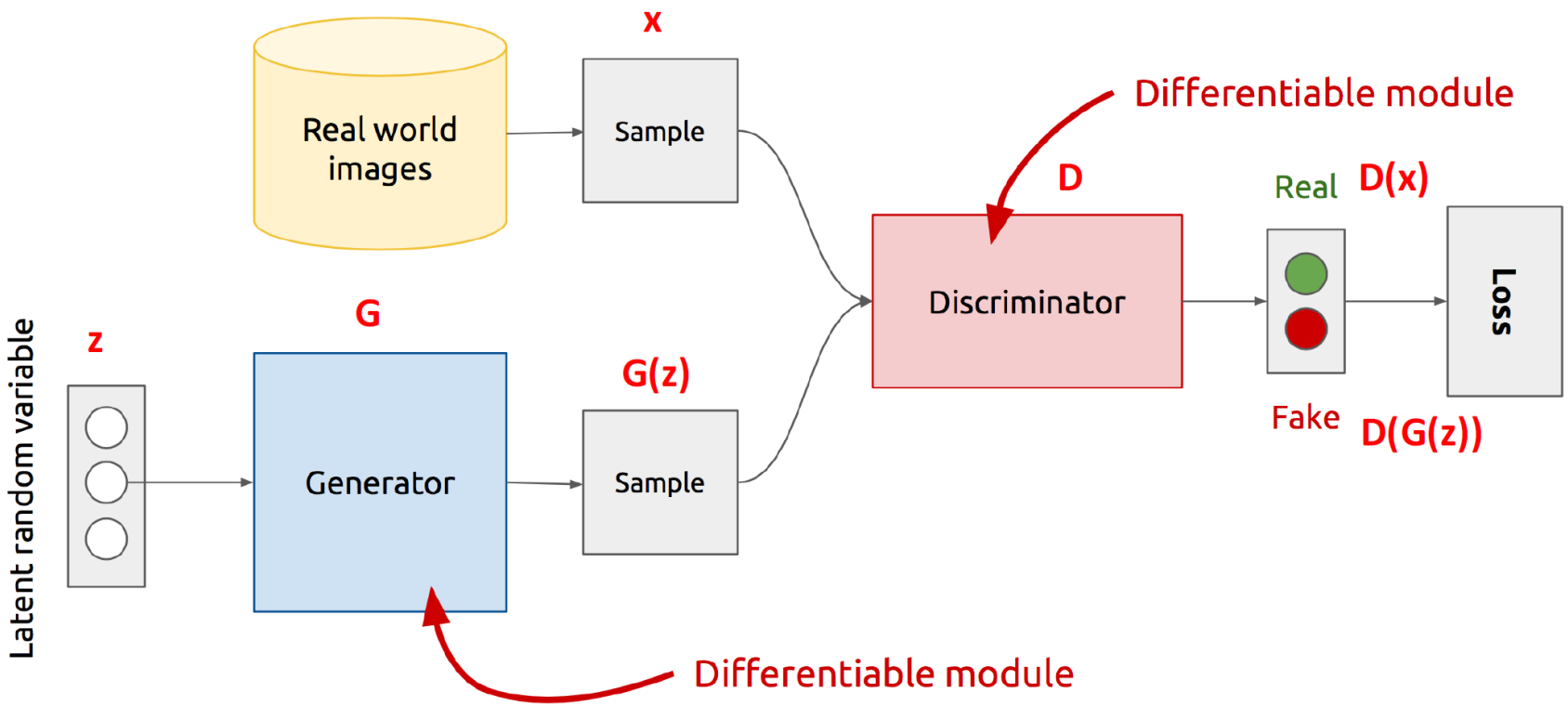


Ledig, Christian, et al. "Photo-realistic single image super-resolution using a generative adversarial network." *arXiv preprint arXiv:1609.04802* (2016).

Adversarial Training

- Generator: generate fake samples, tries to fool the Discriminator
- Discriminator: tries to distinguish between real and fake samples
- Train them against each other
- Repeat this and we get better Generator and Discriminator

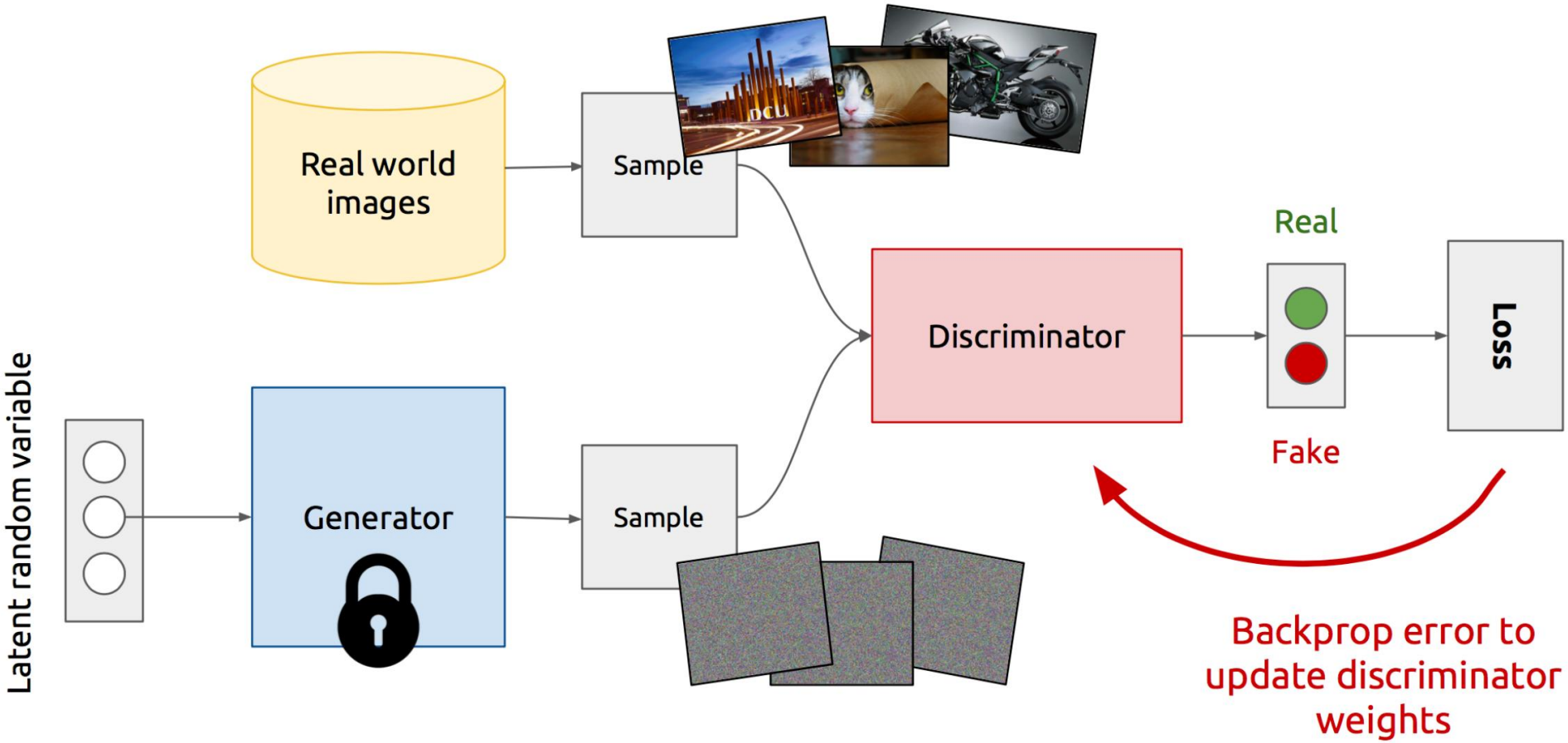
GAN's Architecture



- **Z** is some random noise (Gaussian/Uniform).
- **Z** can be thought as the latent representation of the image.

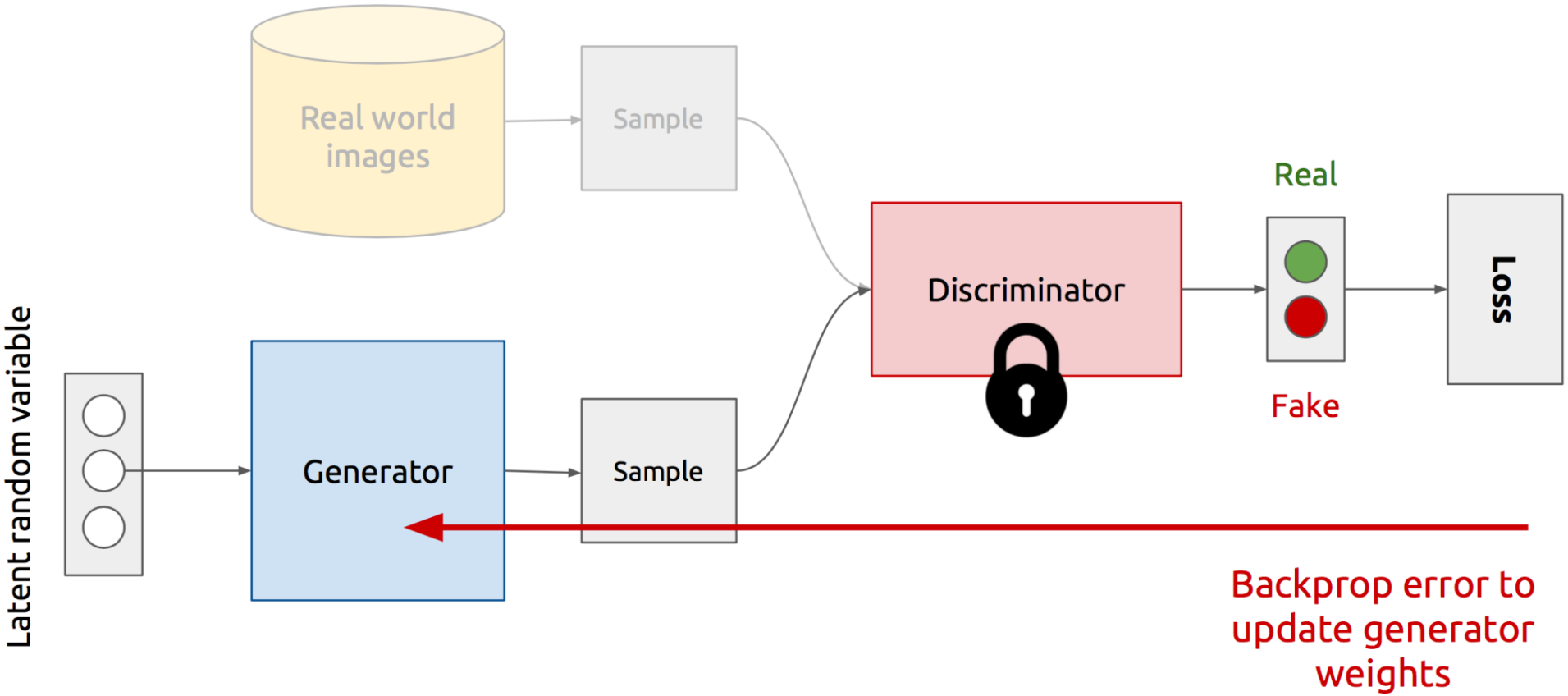
https://www.slideshare.net/xavigiro/deep-learning-for-computer-vision-generative-models-and-adversarial-training-upc-2016

Training Discriminator



https://www.slideshare.net/xavigiro/deep-learning-for-computer-vision-generative-models-and-adversarial-training-upc-2016

Training Generator



https://www.slideshare.net/xavigiro/deep-learning-for-computer-vision-generative-models-and-adversarial-training-upc-2016

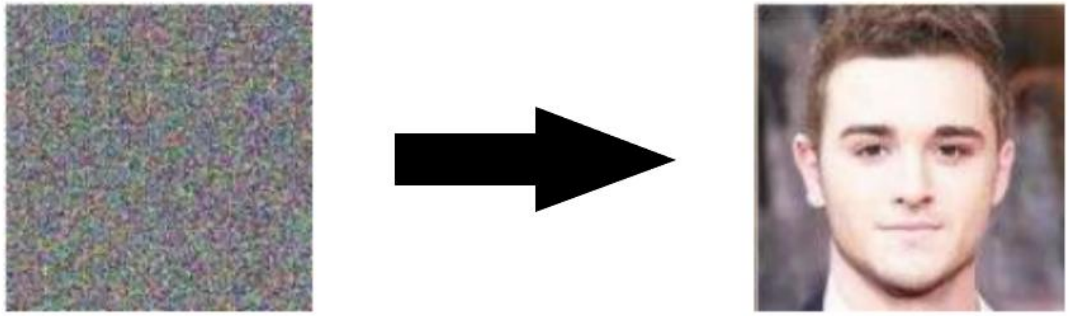
Diffusion models intro

- Recent superior image generators,
- E.g., DALL-E is prompt based
- "a bowl of soup that is a portal to another dimension as digital art".



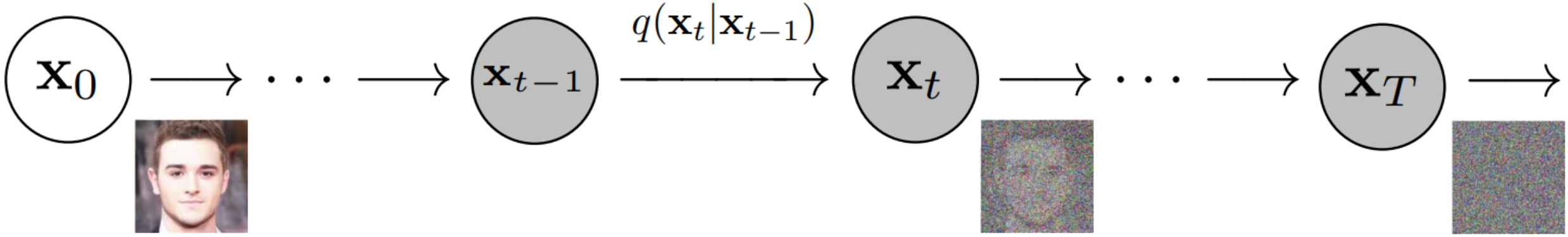
Idea of diffusion models

- generate data similar to the data on which they are trained
- destroy training data through the successive addition of Gaussian noise
- then learning to recover the data by *reversing* this noising process.
- After training, generate data by passing randomly sampled noise through the learned denoising process.



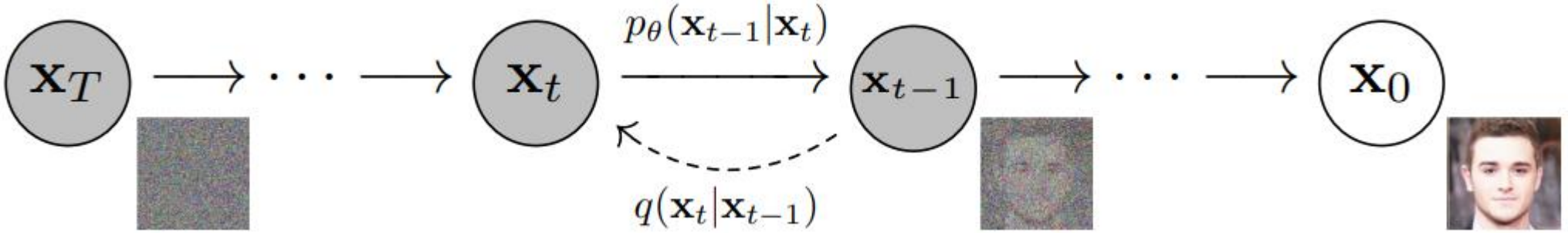
Diffusion models 1/2

- A diffusion model maps to the latent space using a fixed Markov chain. This chain gradually adds noise to the data in order to obtain the approximate posterior.



Diffusion models 2/2

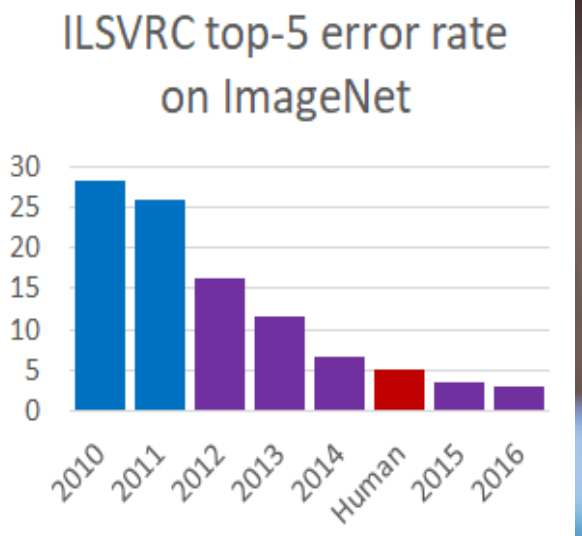
- A diffusion model is trained to **reverse** the process



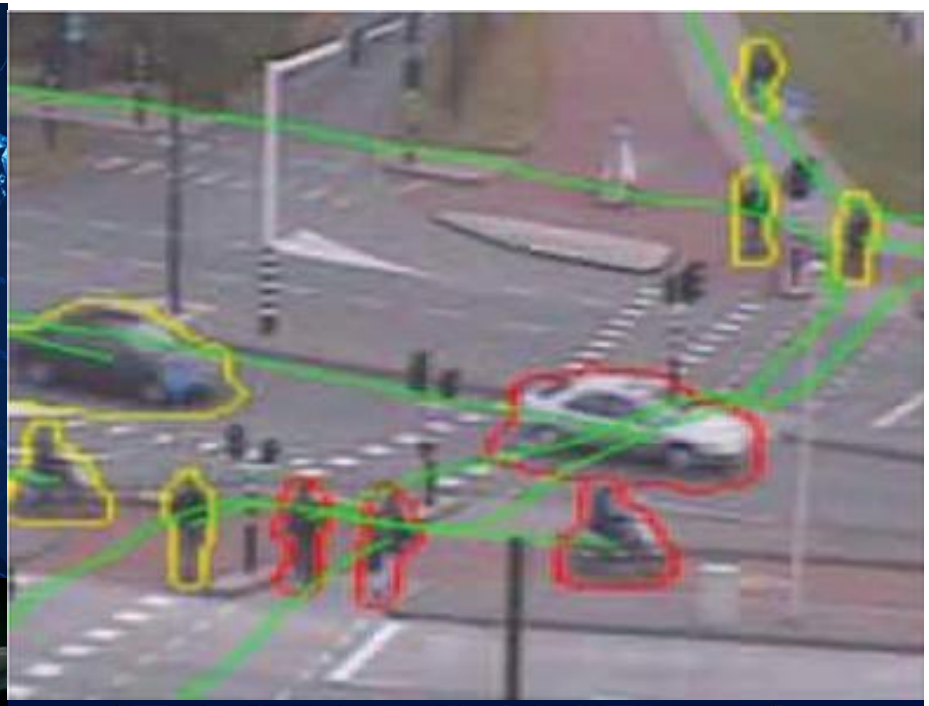
Sources

- Ian Goodfellow and Yoshua Bengio and Aaron Courville: *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org
- PyTorch
- HuggingFace library
- TensorFlow

Deep learnig





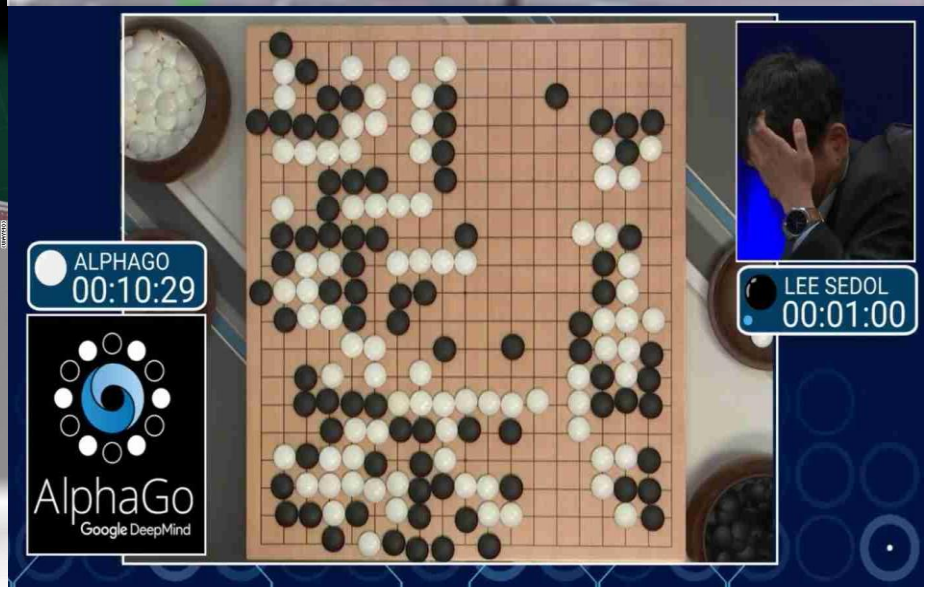




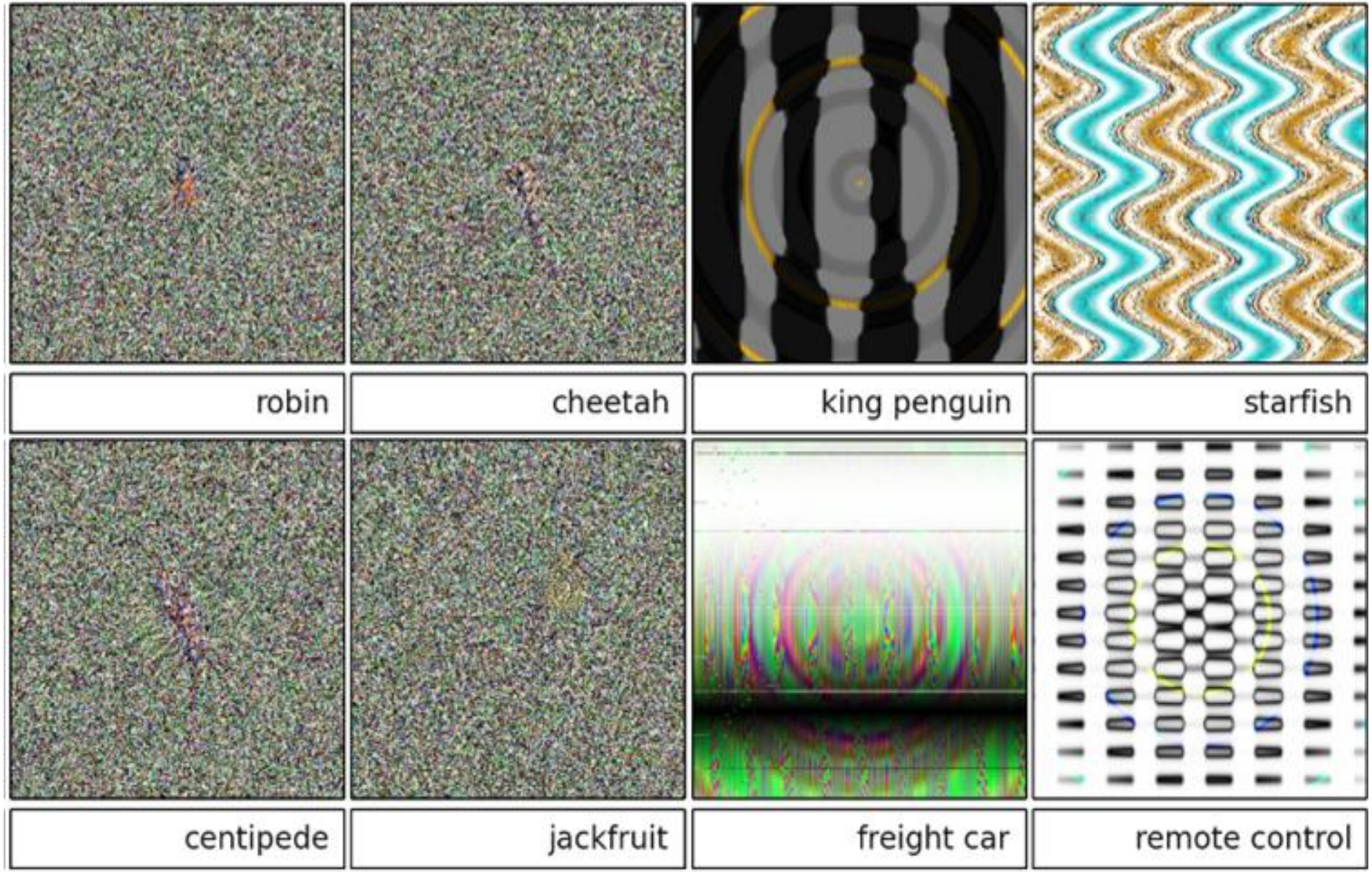




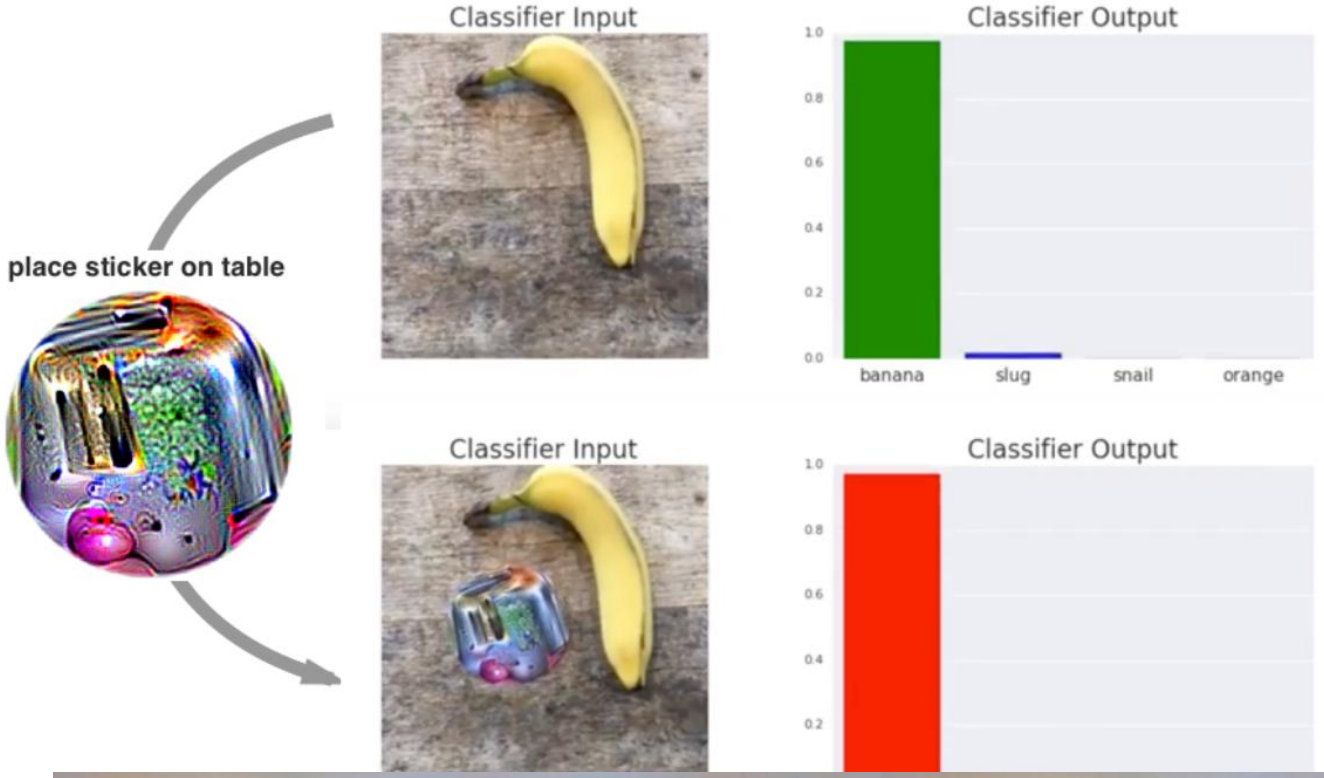
Microsoft claims new speech recognition record, achieving a super-human 5.1% error rate

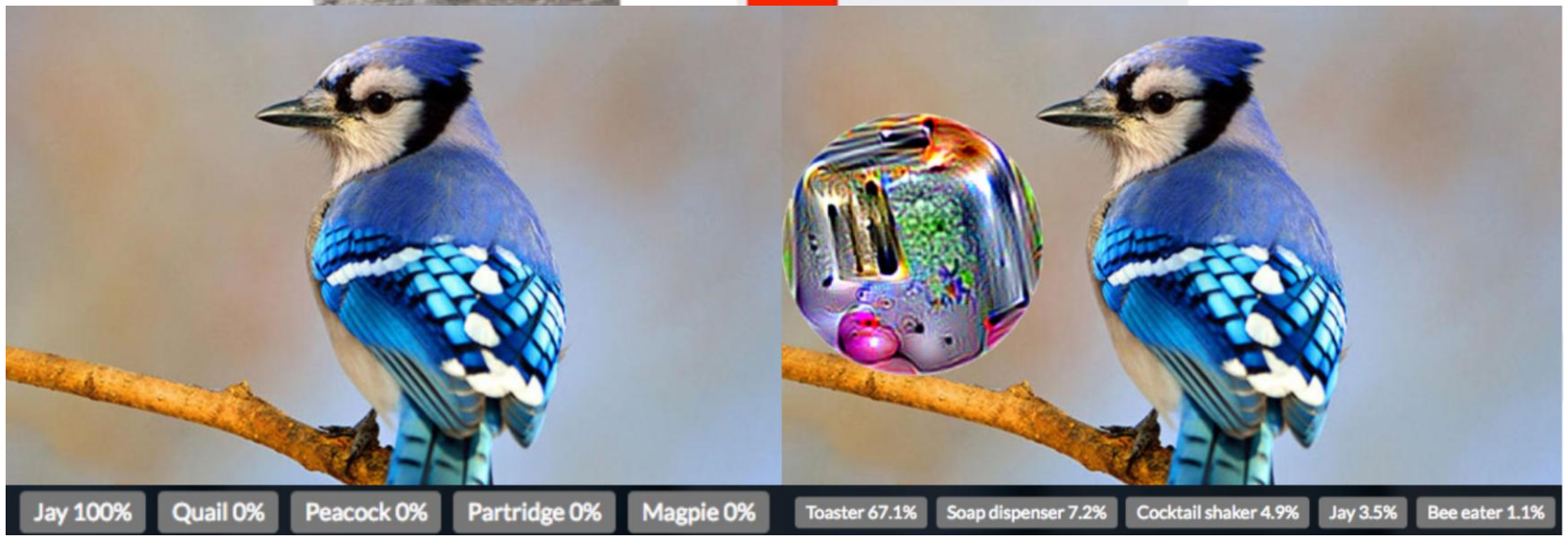


Weaknesses of deep learning



Attacks on neural networks





Failures on out-of-distribution examples

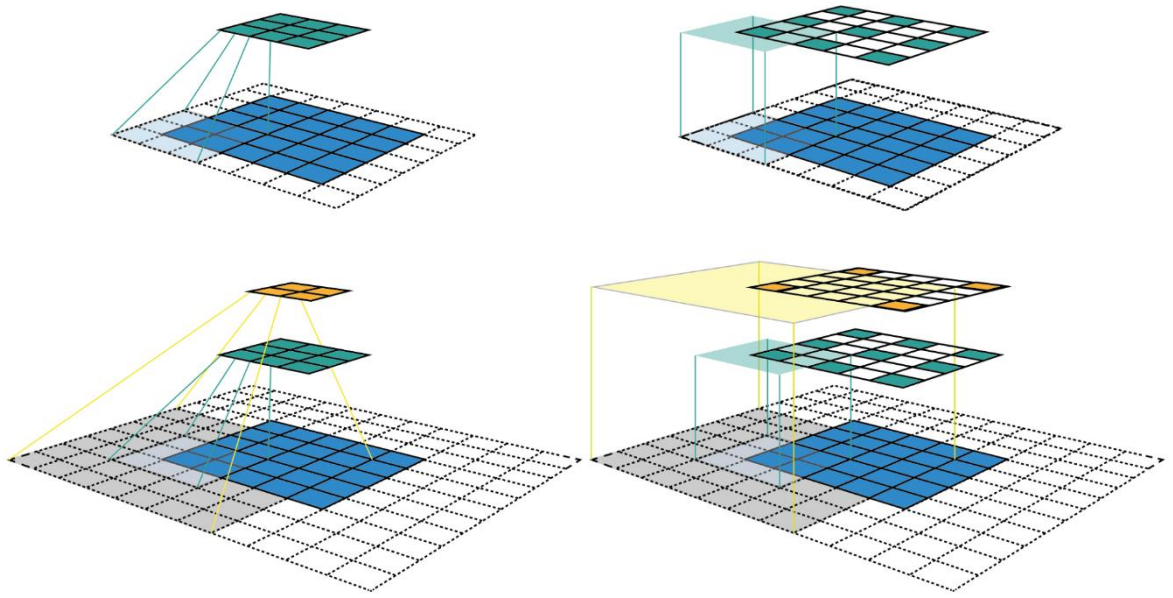
Michael A. Alcorn, Qi Li, Zhitao Gong, Chengfei Wang, Long Mai, Wei-Shinn Ku, Anh Nguyen (2018):
Strike (with) a Pose: Neural Networks Are Easily Fooled by Strange Poses of Familiar Objects. arXiv:1811.11553



University of Ljubljana, Faculty of Computer and Information Science

Neural networks





Prof Dr Marko Robnik-Šikonja

Intelligent Systems, Edition 2025

Topics overview

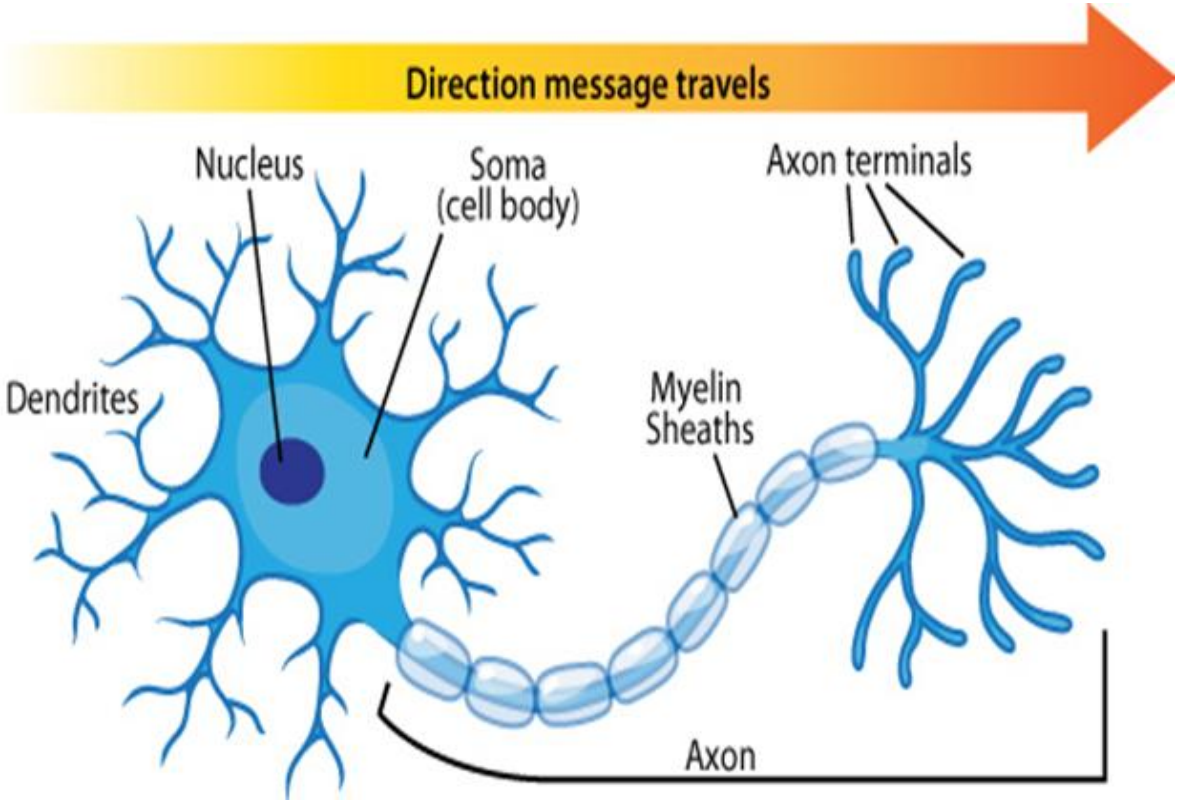
- Basics of artificial neural networks (revision)
- Backpropagation (revision)
- Deep learning
- Convolutional neural networks
- Autoencoders
- Generative adversarial networks
- Robustness

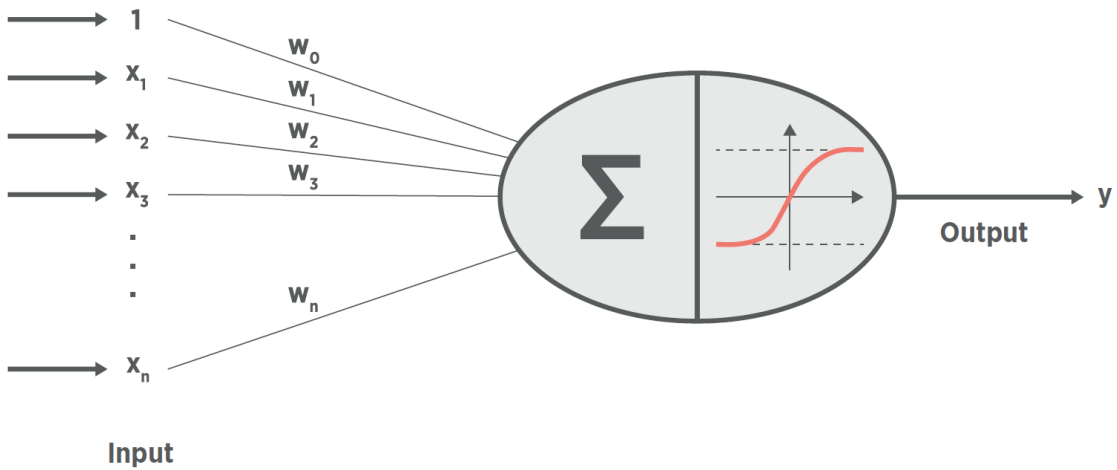
We will mention transformer networks in the natural language processing topic.

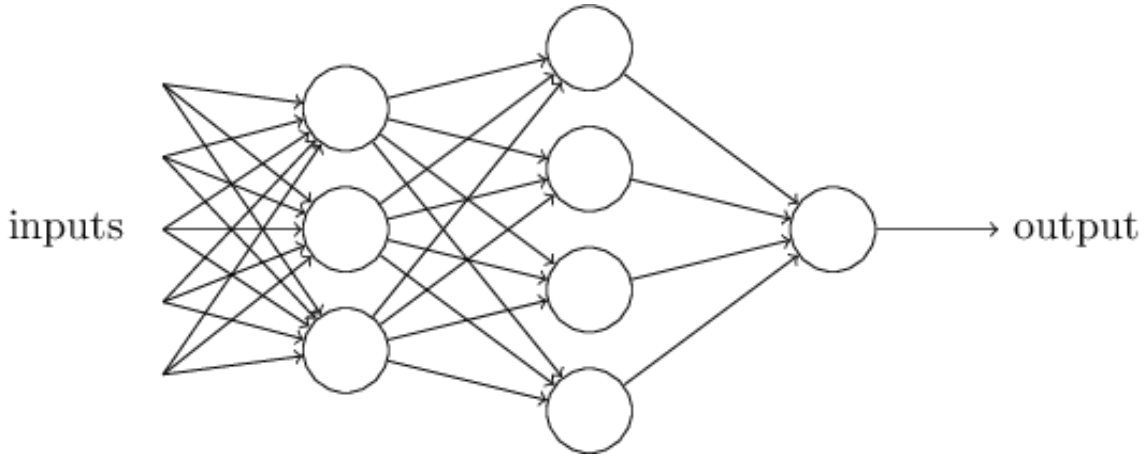
Artificial neural networks

- many approaches, we shall cover the basic ideas
- currently very strong interest, especially in deep neural networks
- http://www.deeplearningbook.org (from 2016, see also other newer literature in the introductory slides 01)

Artificial neural networks: brain analogy







learning: error backpropagation

A neuron

- Computational units, passing messages (information) in the network, typically organized into layers

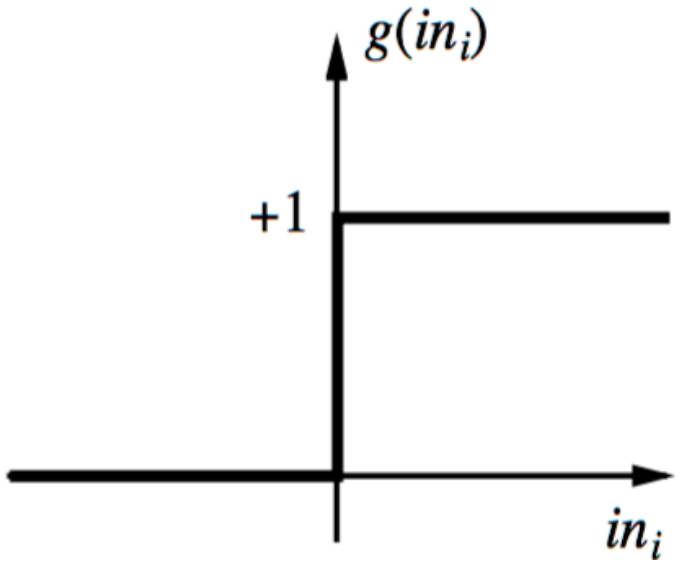
$$y = \sigma\left(\sum_{i=0}^{n} w_i x_i\right)$$

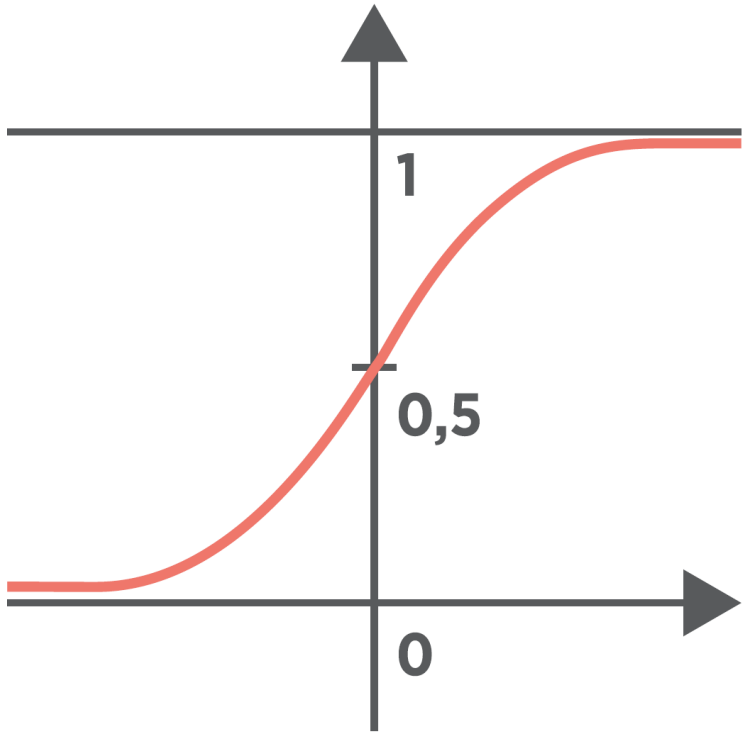
- where x_i are the inputs, $x_0 = 1$ (bias term), w_i are weights of the neuron, and σ is a non-linear activation function

Activation functions

- examples: step function, sigmoid (logistic)

$$f(x) = \frac{1}{1 + e^{-x}}$$





Activation functions

- ReLU (rectified linear unit)

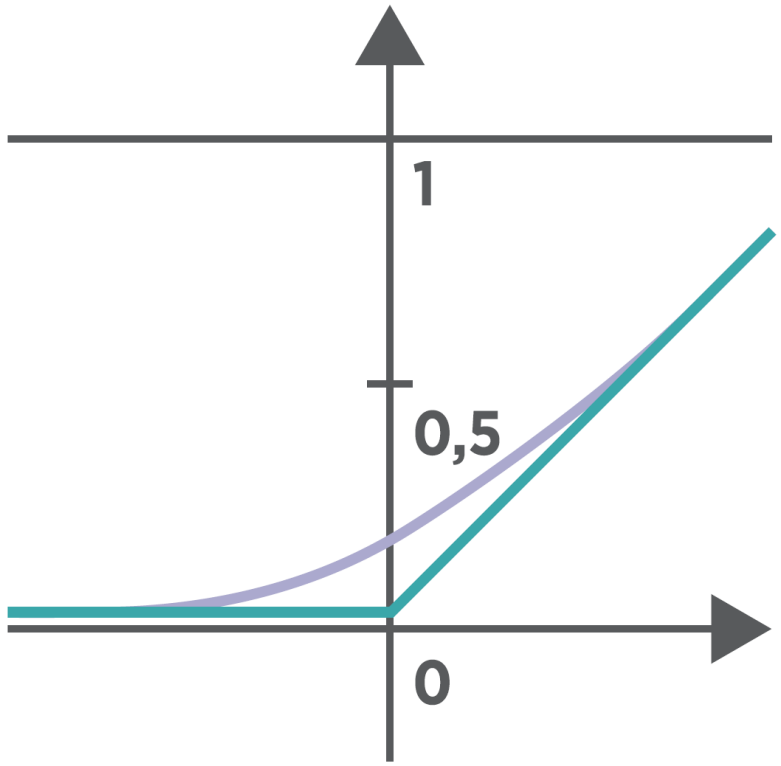
  $f(x) = \max(0, x)$

- softplus / approximation of ReLU with continuous derivation

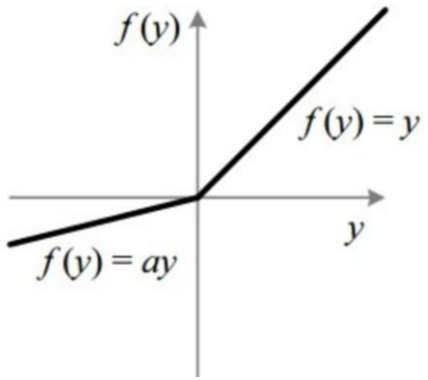
  $f(x) = \ln(1+e^x)$

- ELU (Exponential Linear Unit)

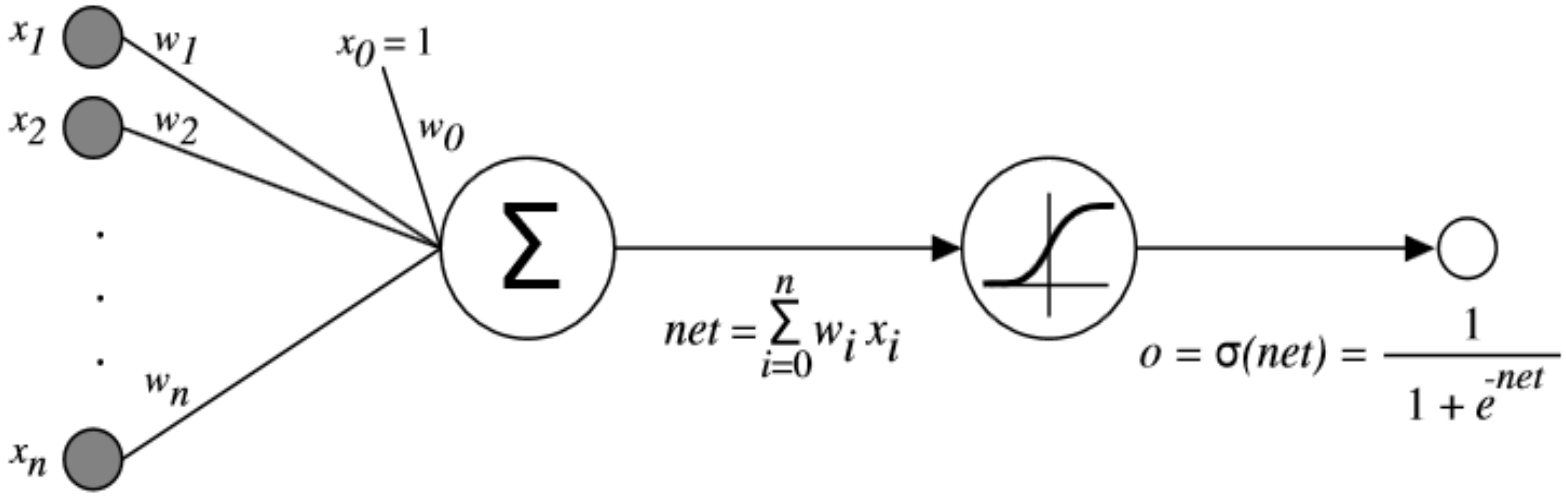
$$ELU(x) = \begin{cases} c \cdot (e^x - 1), & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$$



- Leaky ReLU: like ReLU but small slope for negative values instead of a flat slope
- many others

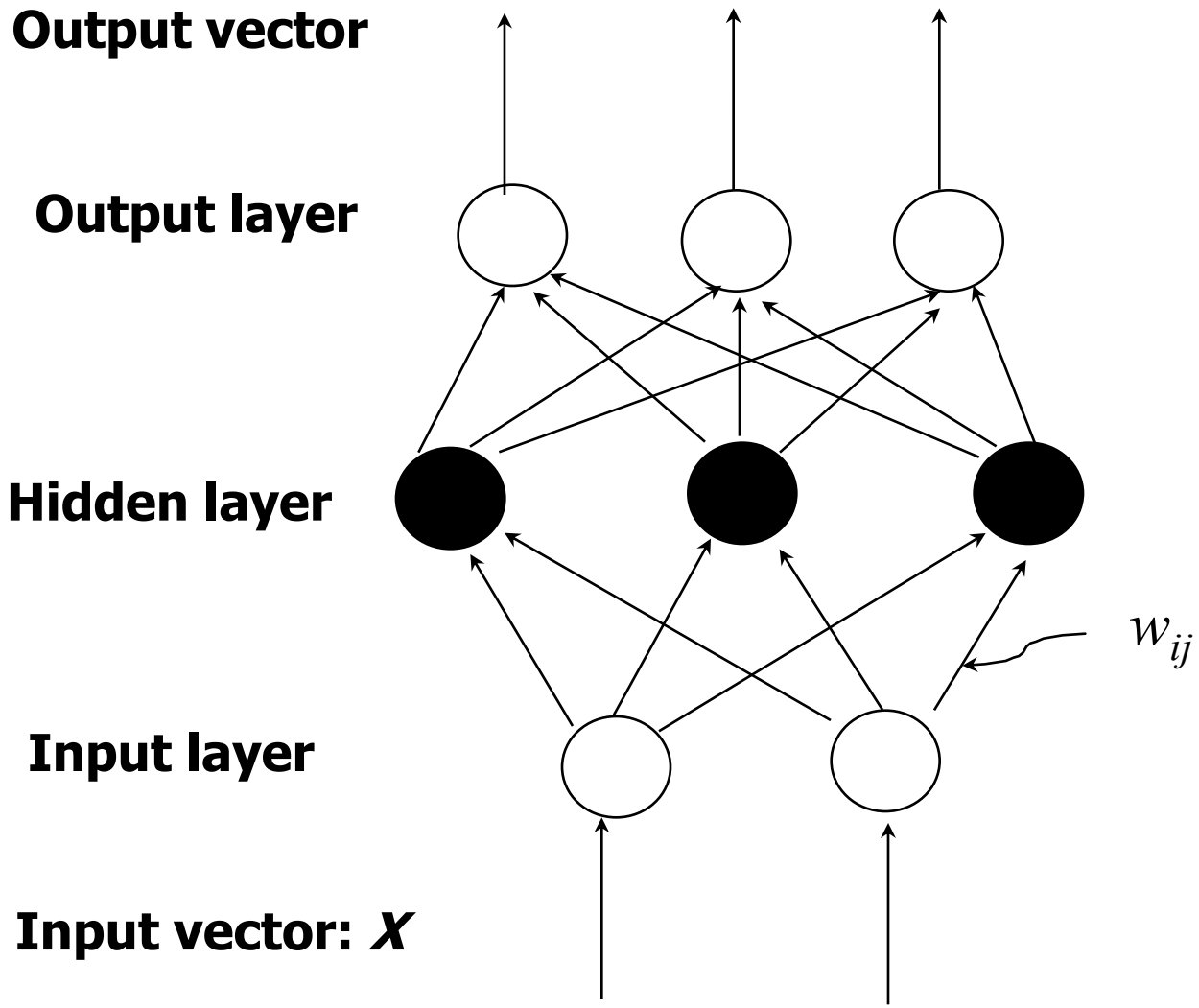


Why nonlinear?



What is a derivative of a sigmoid?

A multi-layer feed-forward NN

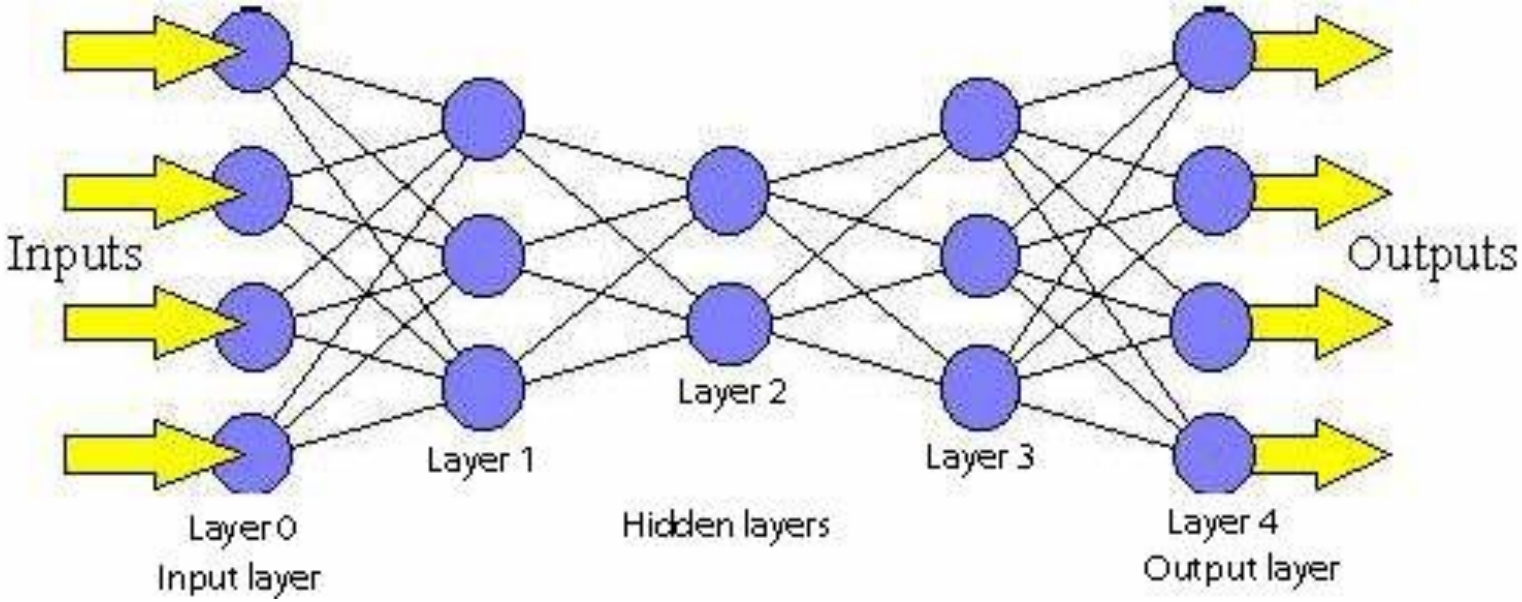


How a multi-layer NN works?

- The **inputs** to the network correspond to the attributes measured for each training tuple
- Inputs are fed simultaneously into the units making up the **input layer**
- They are then weighted and fed simultaneously to a **hidden layer**
- The number of hidden layers is arbitrary; if more than 1 hidden layer is used, the network is called a deep neural network
- The weighted outputs of the last hidden layer are input to units making up the **output layer**, which emits the network's prediction
- The network is **feed-forward**: None of the weights cycles back to an input unit or to an output unit of a previous layer
- If we have backwards connections, the network is called a recurrent neural network
- From a statistical point of view, networks perform **nonlinear regression**: Given enough hidden units and enough training samples, they can closely approximate any function

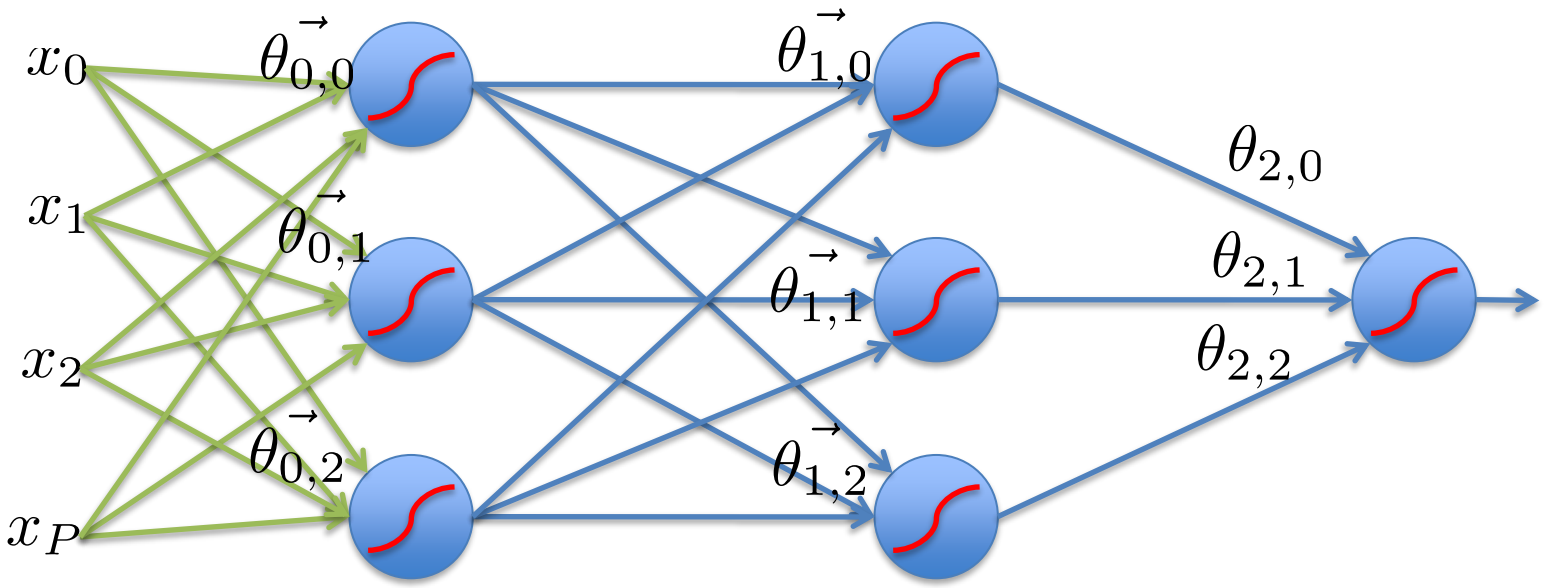
Feed-Forward Network

- neurons are activated progressively throug layers from input to output



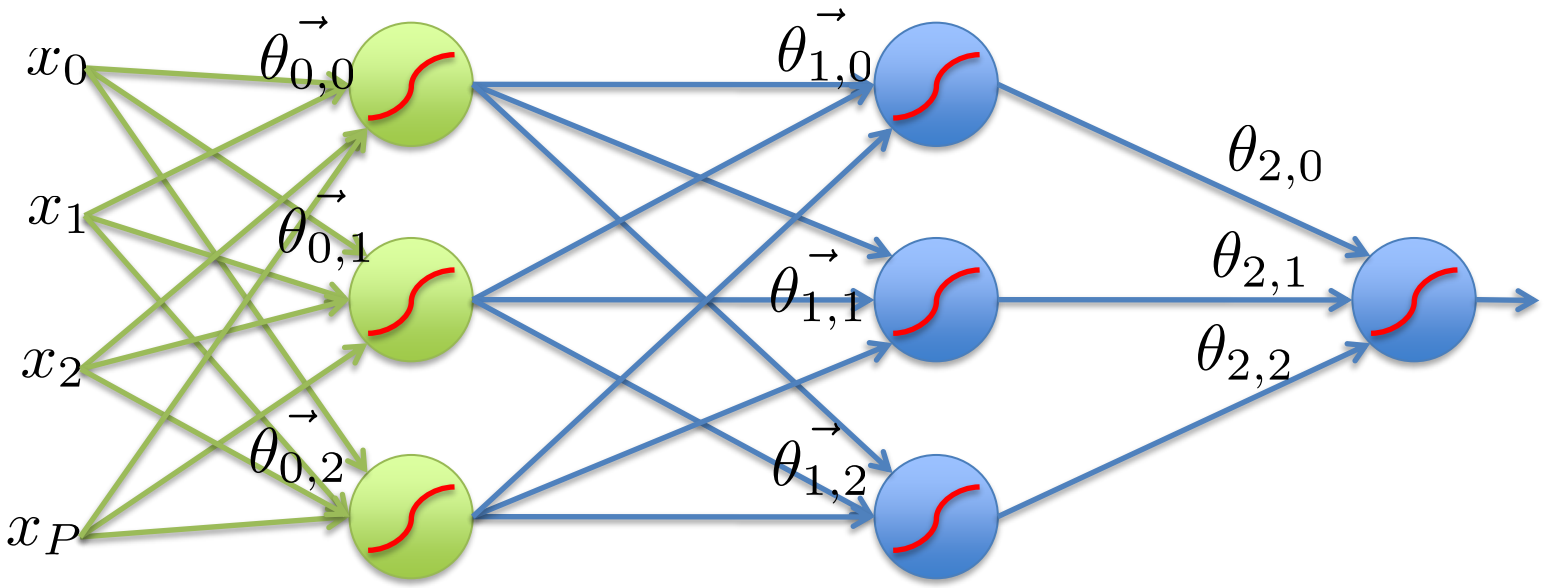
Feed-Forward Network

- Values are propagated through the network to the output, which returns the prediction

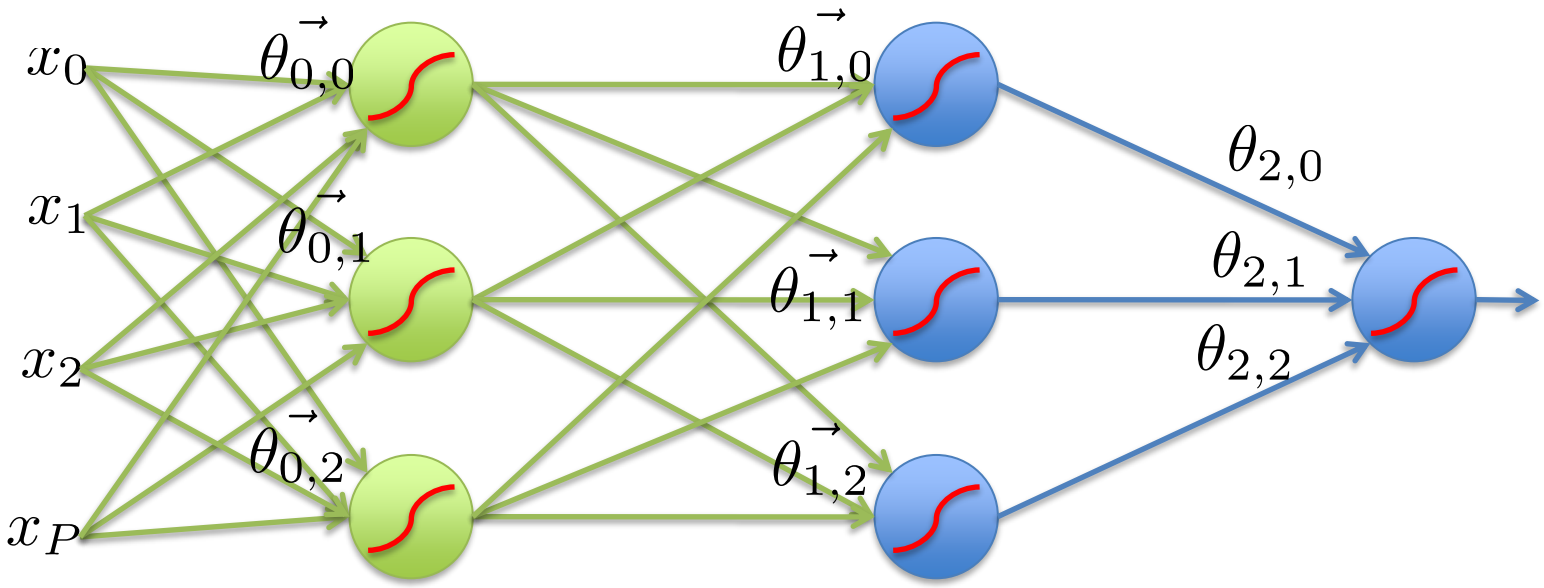


Next 6 slides by Andrew Rosenberg

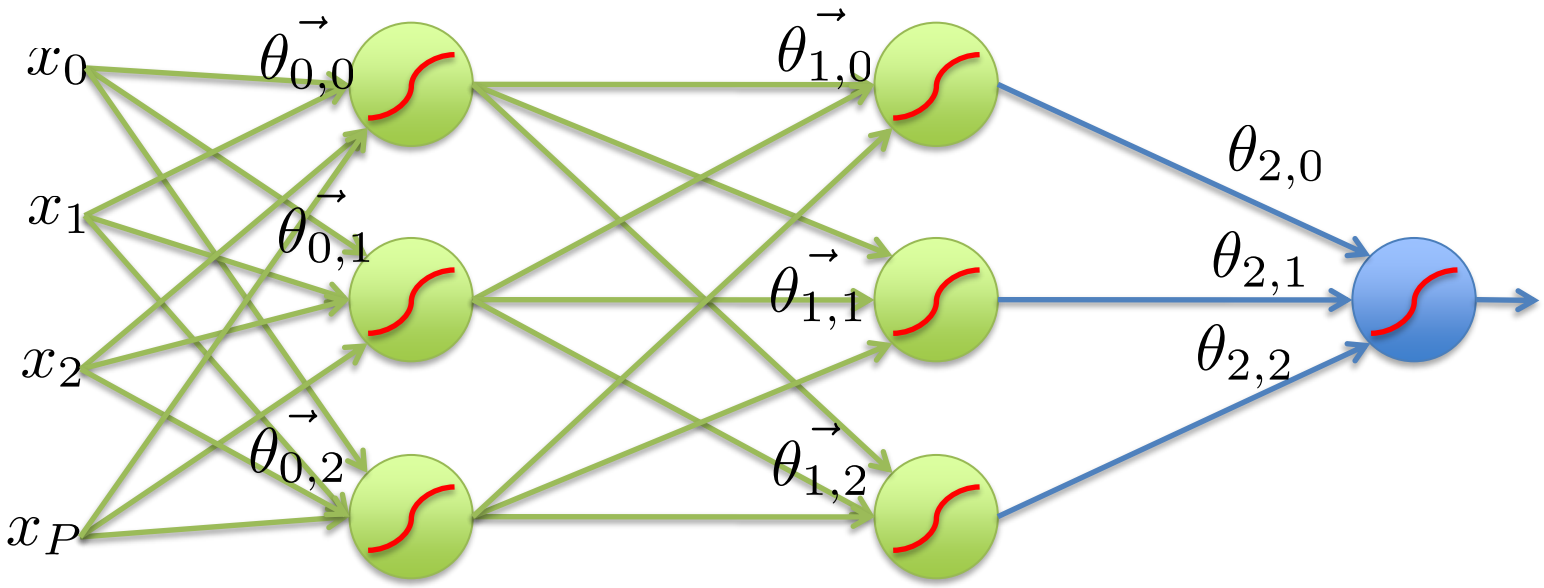
Feed-Forward Networks



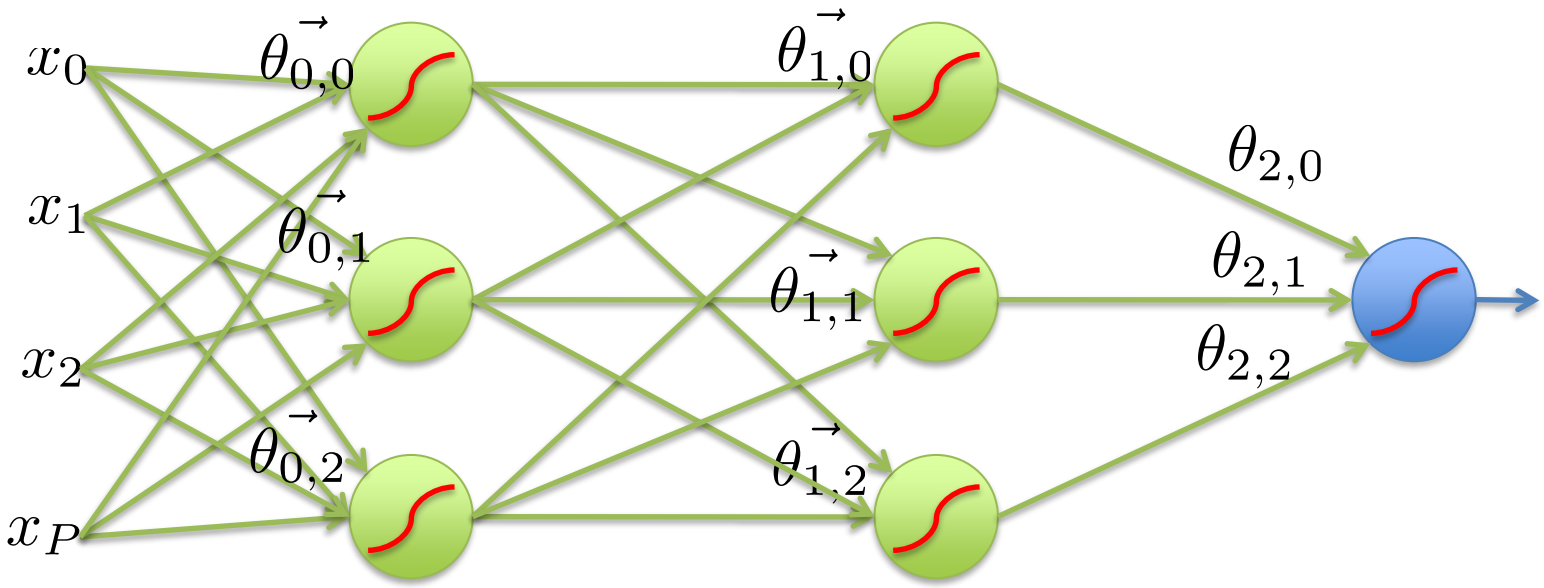
Feed-Forward Networks



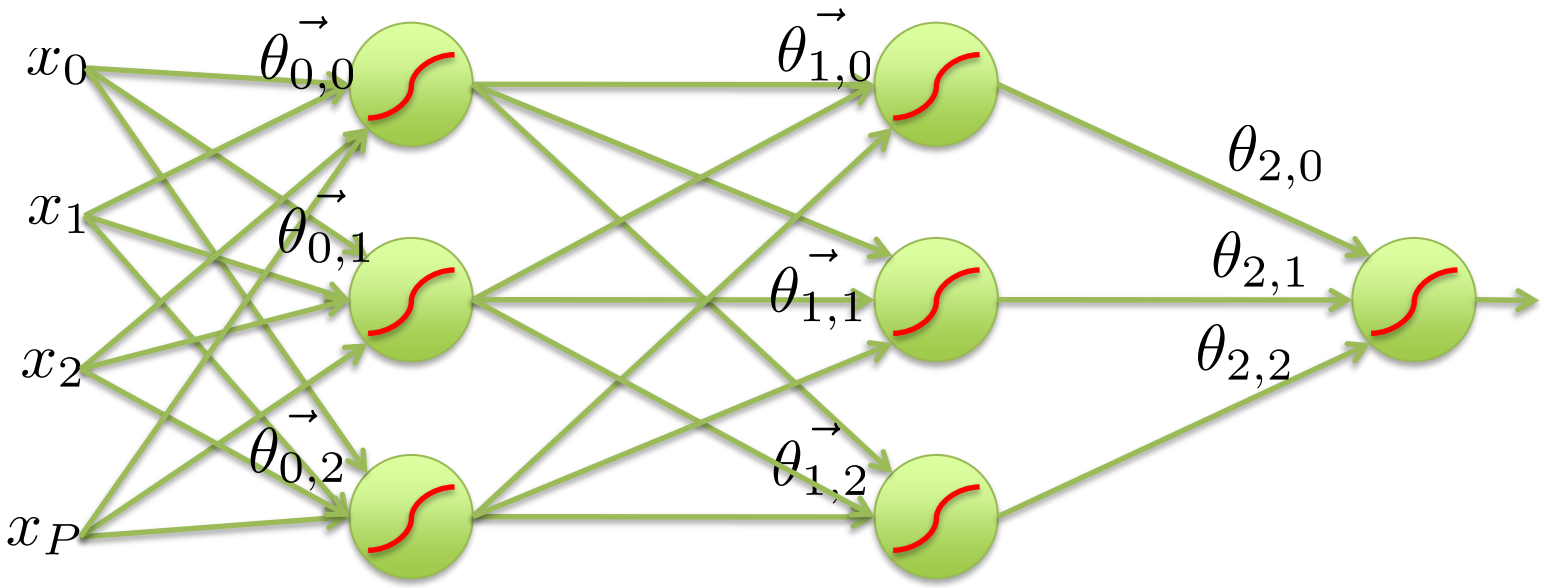
Feed-Forward Networks



Feed-Forward Networks



Feed-Forward Networks



Softmax

- In classification, softmax is often used for the last layer
- normalizes the output scores to be a probability distribution (values between 0 and 1, the sum is 1)

$$y_i = \frac{e^{z_i}}{\sum_{j \in group} e^{z_j}}$$

$$\frac{\partial y_i}{\partial z_i} = y_i \left(1 - y_i\right)$$

Criterion function

- together with softmax we frequently use cross entropy as loss (cost) function C

$$C = -\sum_j t_j \log y_j$$

target value

$$\frac{\partial C}{\partial z_i} = \sum_j \frac{\partial C}{\partial y_j} \frac{\partial y_j}{\partial z_i} = y_i - t_i$$

Backpropagation learning algorithm for NN

- Backpropagation: a neural network learning algorithm
- Started by psychologists and neurobiologists to develop and test computational analogues of neurons
- A neural network: a set of connected input/output units where each connection has a weight associated with it
- During the learning phase, the network learns by adjusting the weights so as to be able to predict the correct class label of the input tuples
- Also referred to as connectionist learning due to the connections between units

Backpropagation algorithm

- Iteratively process a set of training tuples & compare the network's prediction with the actual known target value
- For each training tuple, the weights are modified to **minimize the mean squared error** between the network's prediction and the actual target value
- Modifications are made in the "**backwards**" direction: from the output layer, through each hidden layer down to the first hidden layer, hence "**backpropagation**"
- Steps
  - Initialize weights to small random numbers, associated with biases
  - Propagate the inputs forward (by applying the activation function)
  - Backpropagate the error (by updating weights and biases)
  - Terminating condition (when error is very small, etc.)

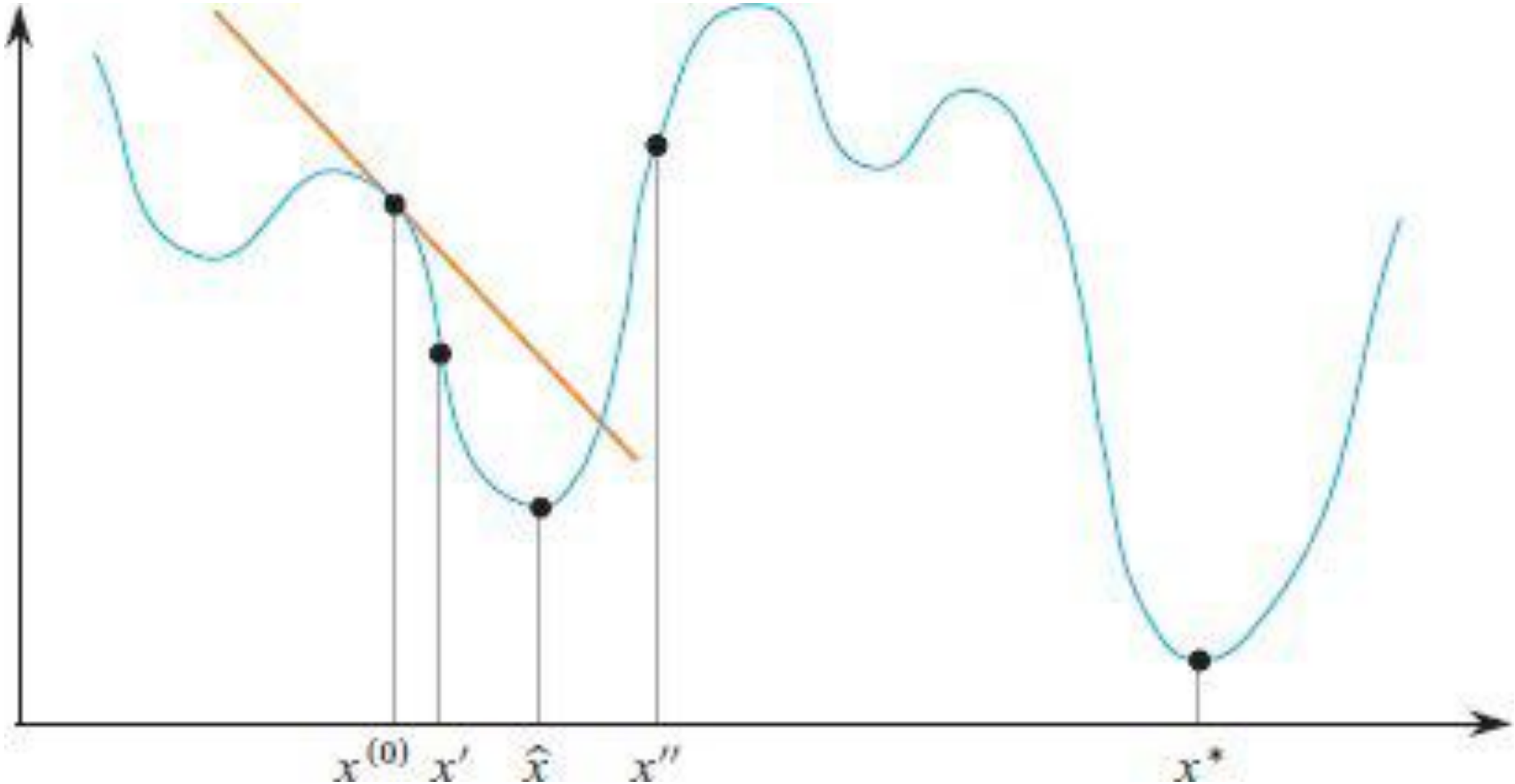
Gradient descent (GD)

- Gradient descent is an efficient local optimization in $\mathbb{R}^n$
- Local minimum of function *f:* $\mathbb{R}^n \rightarrow \mathbb{R}$ is a point ***x*** for which *f(x) ≤ f(x')* for all ***x'*** that are "near" ***x***
- Gradient $\nabla f(x)$ is a function ∇f: $\mathbb{R}^n \rightarrow \mathbb{R}^n$ comprising *n* partial derivatives:

$$\nabla f(x) = (\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \dots, \frac{\partial f}{\partial x_n})$$

- The GD optimization moves in the direction of -$\nabla f(x)$

Illustration of GD



GD algorithm

```
GRADIENT-DESCENT(f, x0, γ, T) {
  // function f,  initial value x0, fixed step size γ, number of steps T
  x_best = x = x0 ; // n-dimensional vectors, initially set to the initial value
  f_best = f_x = f(x_best) ;
  for t = 0 to T – 1 do {
    x_next = x – γ · ∇f(x);  // ∇f(x), x, and x_next are n-dimensional
    f_next = f(x_next)
    if (f_next < f_x)
      x_best = x_next ;
    x = x_next ;
    f_x = f_next ;
  }
  return x_best ;
}
```

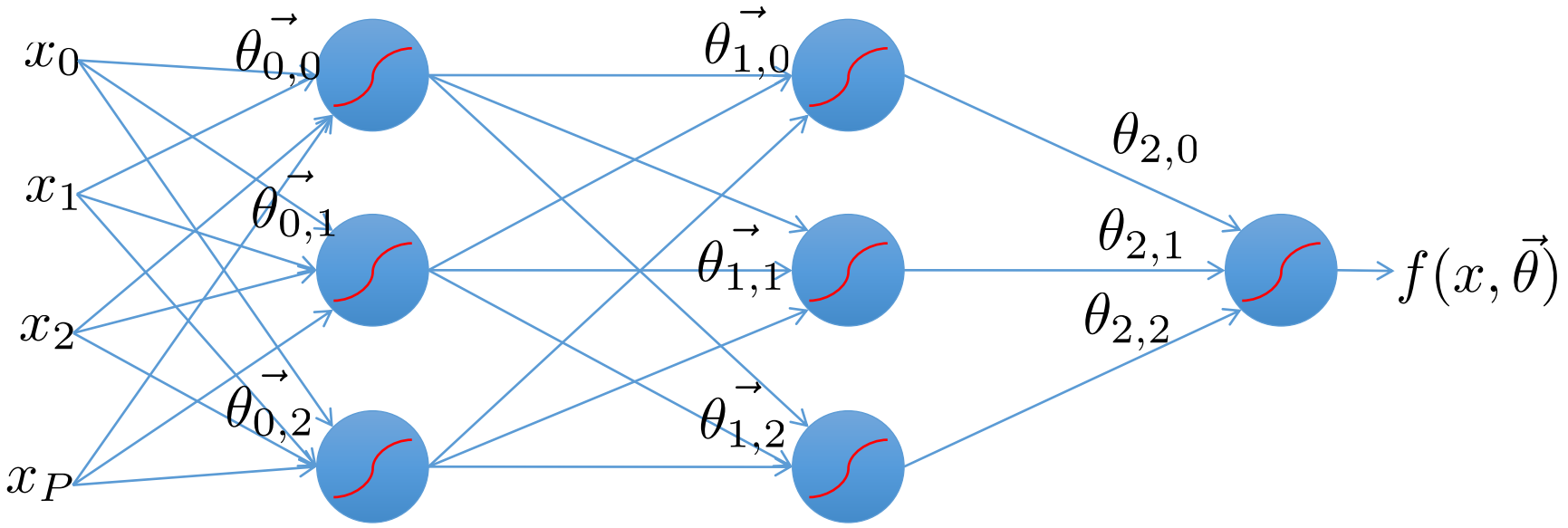
Chain rule of derivation

- In a network, the output of each neuron is a function of the activation function and all its inputs, where the inputs may again be composite functions of neurons in previous layers
- To compute the gradient of a composite function, we use the chain rule of derivation

$$f\big(g(x)\big)' = f'\big(g(x)\big)g'(x)$$

Error Backpropagation

- We will do gradient descent on the whole network.
- Training will proceed from the last layer to the first.

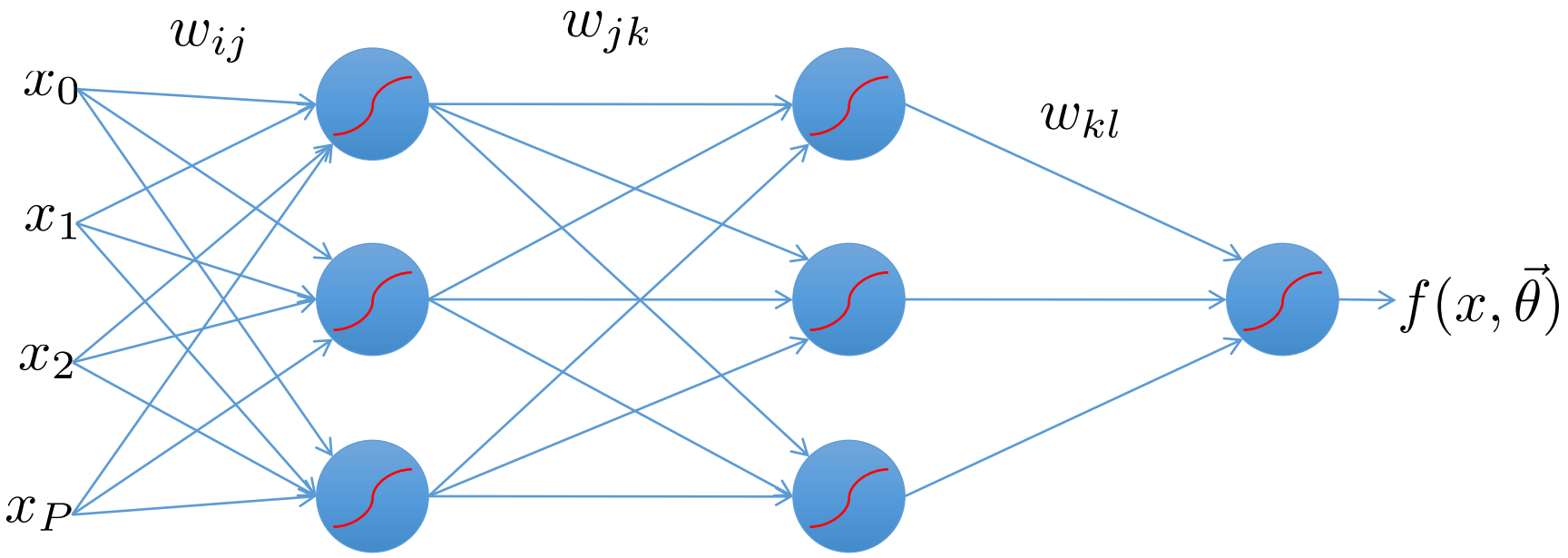


Next 18 slides by Andrew Rosenberg

Error Backpropagation

- Introduce variables over the neural network

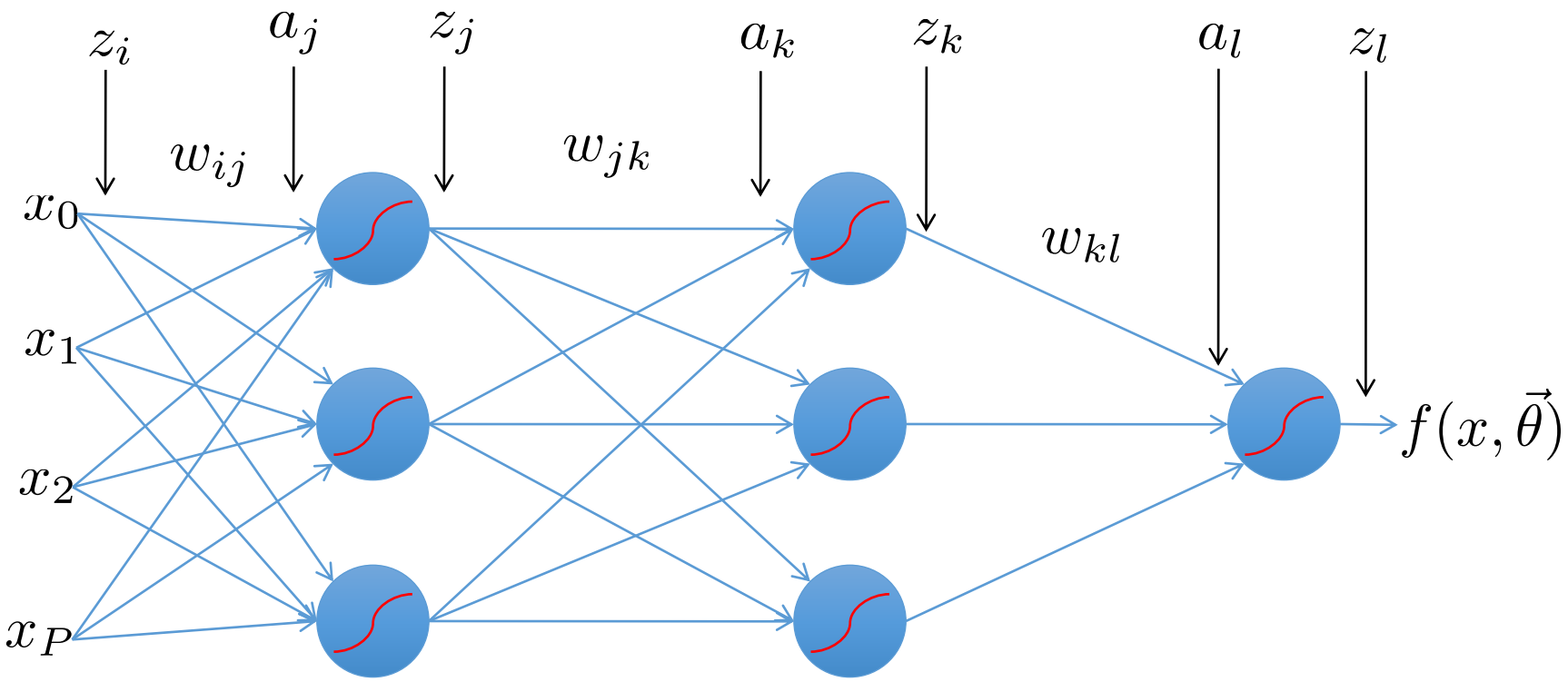
$$\vec{\theta} = \{w_{ij}, w_{jk}, w_{kl}\}$$



Error Backpropagation

$$\vec{\theta} = \{w_{ij}, w_{jk}, w_{kl}\}$$

- Introduce variables over the neural network
  - Distinguish the input and output of each node

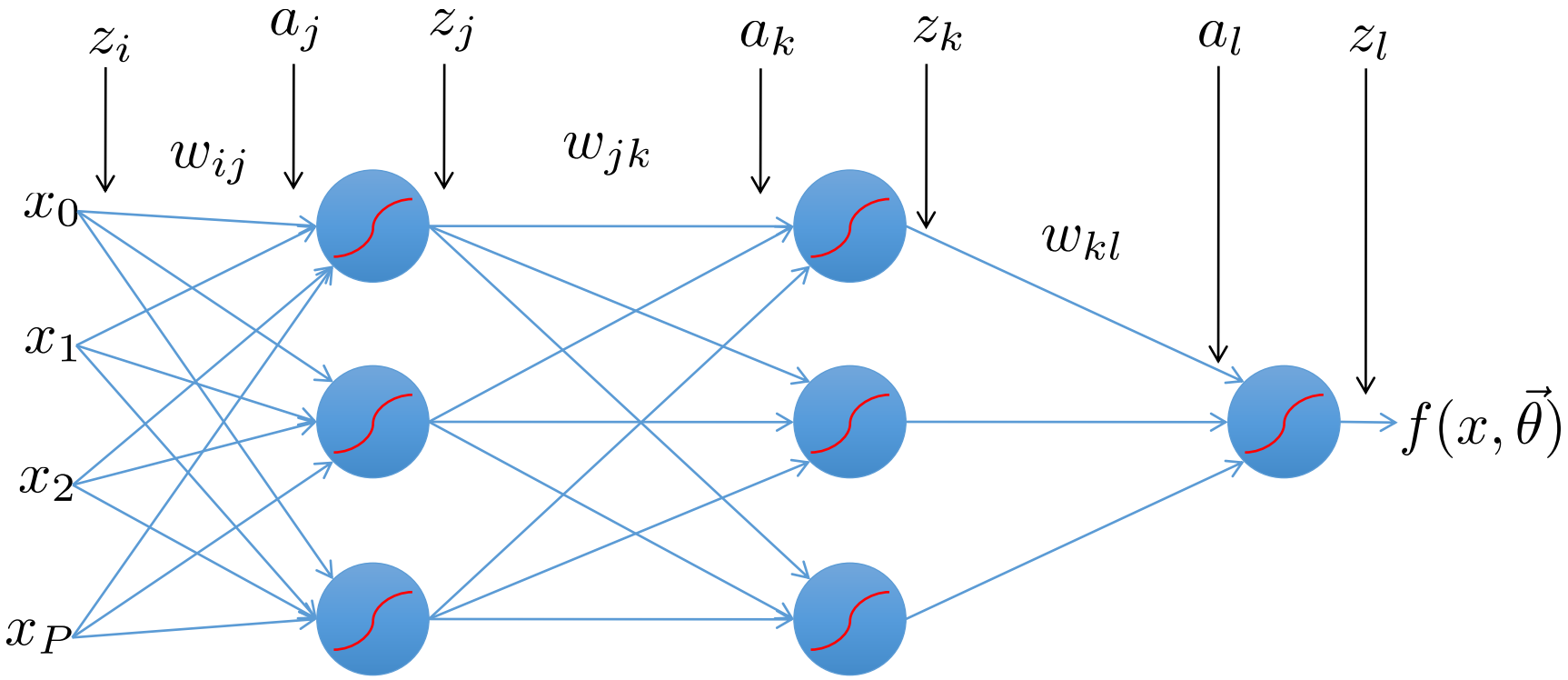


Error Backpropagation

$$\vec{\theta} = \{w_{ij}, w_{jk}, w_{kl}\}$$

$$a_j = \sum_i w_{ij} z_i \qquad a_k = \sum_j w_{jk} z_j \qquad a_l = \sum_k w_{kl} z_k$$

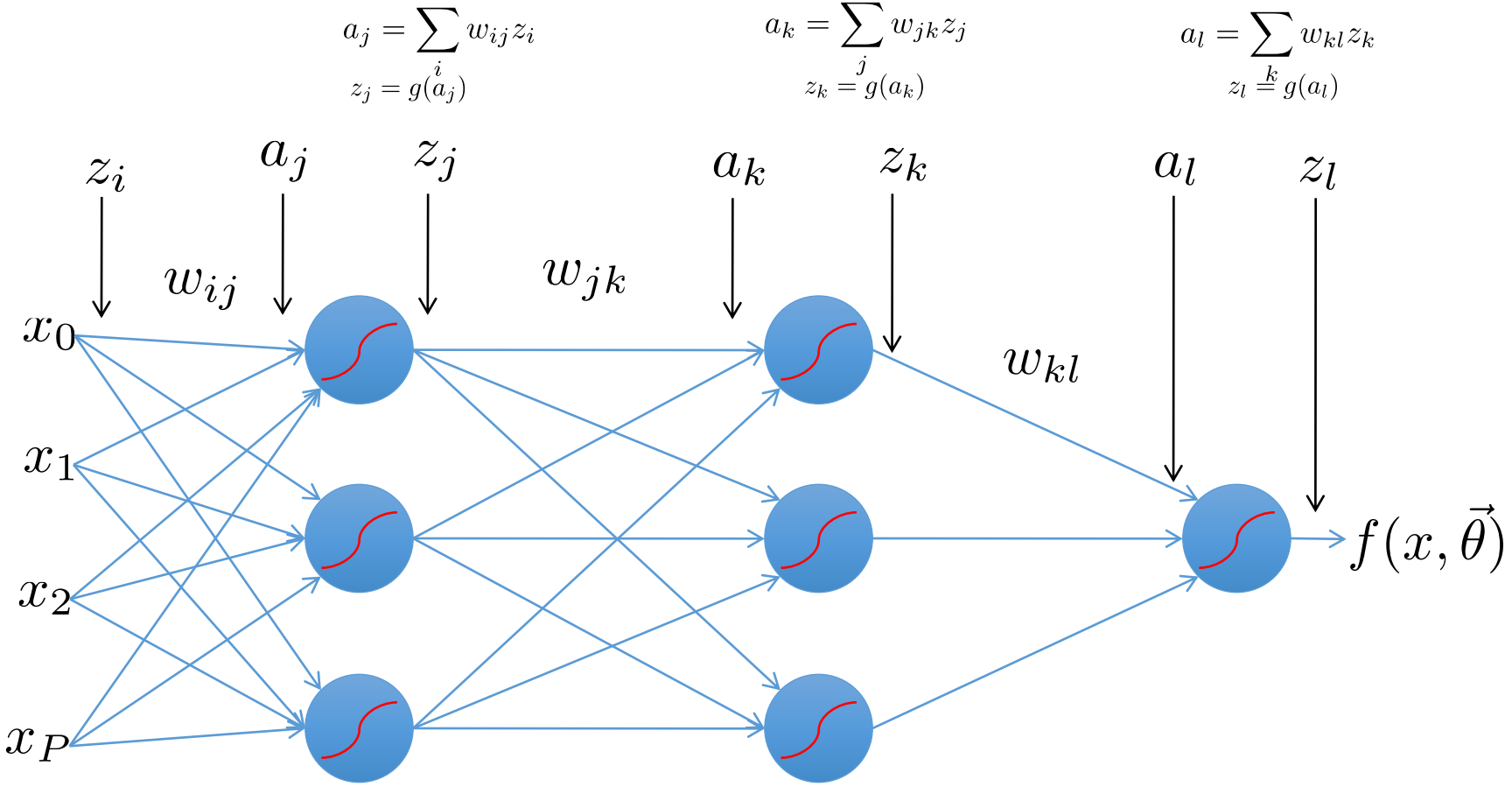
$$z_j = g(a_j) \qquad z_k = g(a_k) \qquad z_l = g(a_l)$$



Error Backpropagation

$$\vec{\theta} = \{w_{ij}, w_{jk}, w_{kl}\}$$

Training: Take the gradient of the last component and iterate backwards



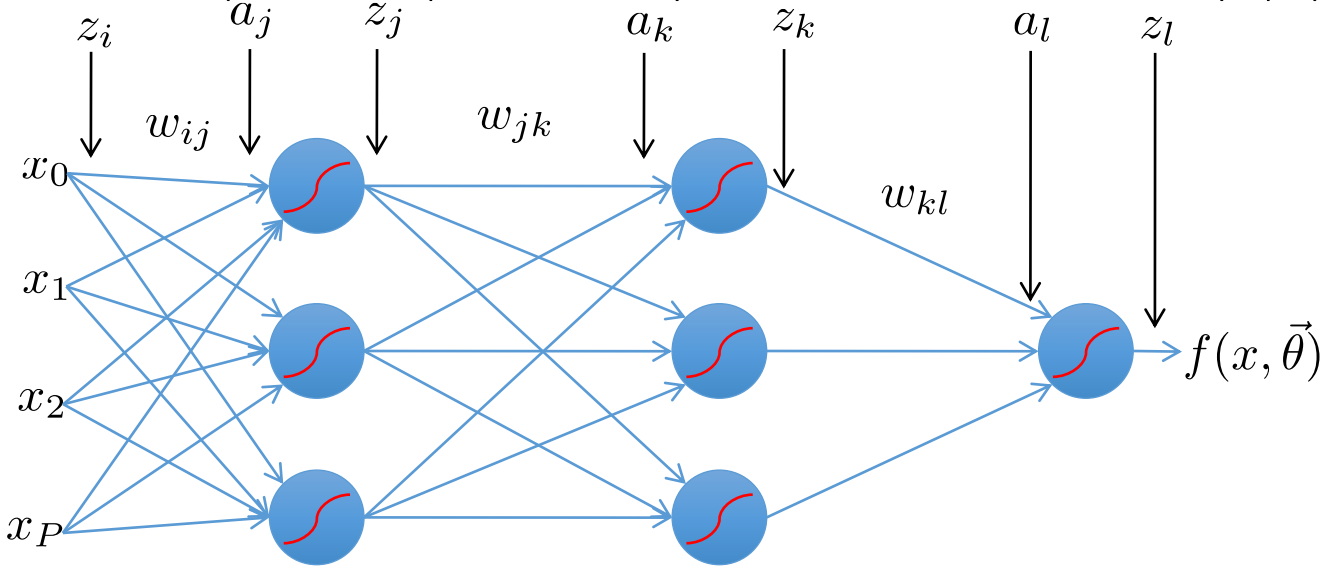
Error Backpropagation

$$R(\theta) = \frac{1}{N}\sum_{n=0}^{N} L(y_n - f(x_n))$$

Empirical Risk Function

$$= \frac{1}{N}\sum_{n=0}^{N} \frac{1}{2}\left(y_n - f(x_n)\right)^2$$

$$= \frac{1}{N}\sum_{n=0}^{N} \frac{1}{2}\left(y_n - g\left(\sum_k w_{kl} g\left(\sum_j w_{jk} g\left(\sum_i w_{ij} x_{n,i}\right)\right)\right)\right)^2$$



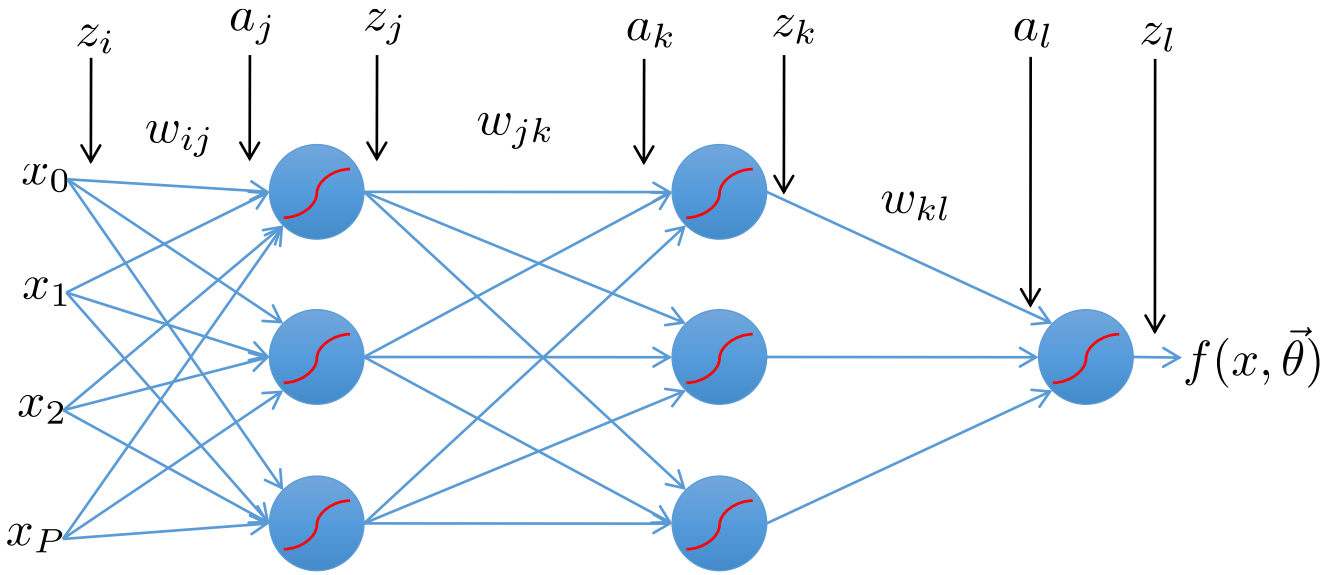
Error Backpropagation

Optimize last layer weights w_{kl}

$$L_n = \frac{1}{2}\left(y_n - f(x_n)\right)^2$$

$$\frac{\partial R}{\partial w_{kl}} = \frac{1}{N}\sum_n \left[\frac{\partial L_n}{\partial a_{l,n}}\right]\left[\frac{\partial a_{l,n}}{\partial w_{kl}}\right]$$

Calculus chain rule



Error Backpropagation

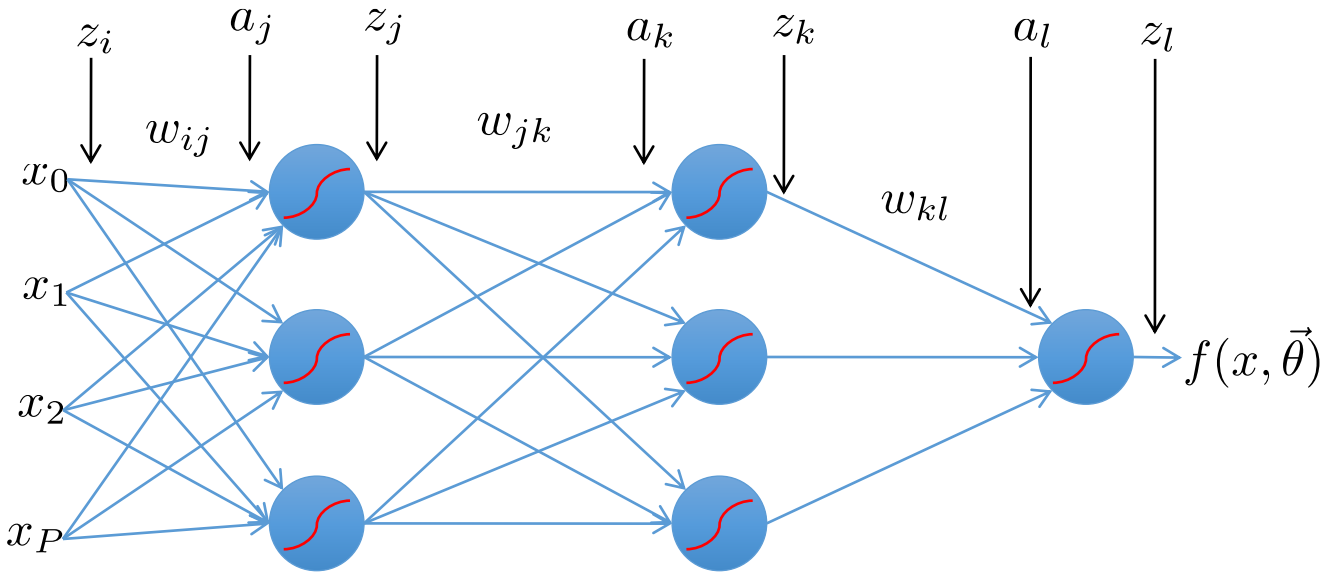
Optimize last layer weights w_{kl}

$$L_n = \frac{1}{2}\left(y_n - f(x_n)\right)^2$$

$$\frac{\partial R}{\partial w_{kl}} = \frac{1}{N}\sum_n \left[\frac{\partial L_n}{\partial a_{l,n}}\right]\left[\frac{\partial a_{l,n}}{\partial w_{kl}}\right]$$

Calculus chain rule

$$\frac{\partial R}{\partial w_{kl}} = \frac{1}{N}\sum_n \left[\frac{\partial \frac{1}{2}(y_n - g(a_{l,n}))^2}{\partial a_{l,n}}\right]\left[\frac{\partial a_{l,n}}{\partial w_{kl}}\right]$$



Error Backpropagation

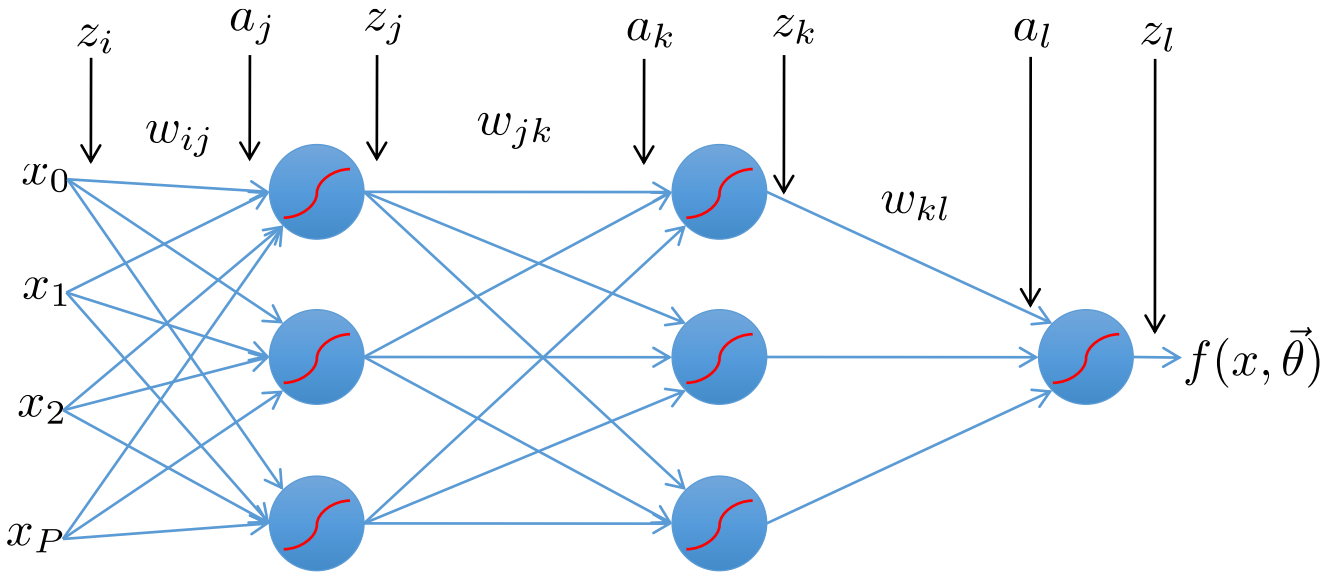
Optimize last layer weights w_{kl}

$$L_n = \frac{1}{2}\left(y_n - f(x_n)\right)^2$$

$$\frac{\partial R}{\partial w_{kl}} = \frac{1}{N}\sum_n \left[\frac{\partial L_n}{\partial a_{l,n}}\right]\left[\frac{\partial a_{l,n}}{\partial w_{kl}}\right]$$

Calculus chain rule

$$\frac{\partial R}{\partial w_{kl}} = \frac{1}{N}\sum_n \left[\frac{\partial \frac{1}{2}(y_n - g(a_{l,n}))^2}{\partial a_{l,n}}\right]\left[\frac{\partial z_{k,n} w_{kl}}{\partial w_{kl}}\right]$$



Error Backpropagation

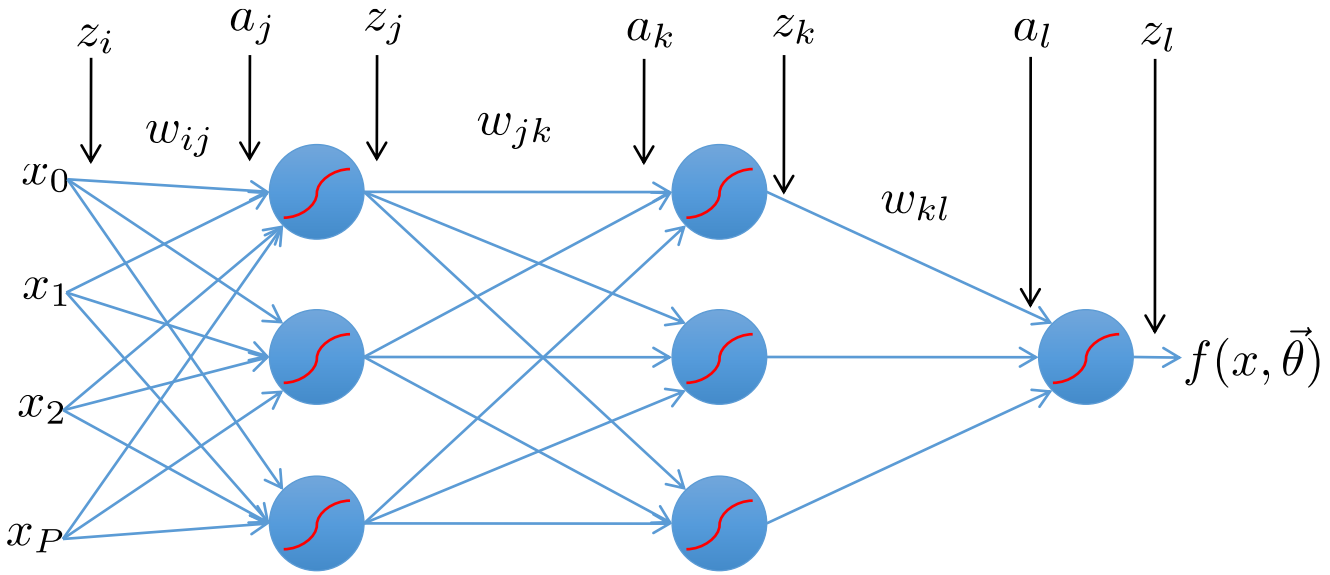
Optimize last layer weights w_{kl}

$$L_n = \frac{1}{2}\left(y_n - f(x_n)\right)^2$$

$$\frac{\partial R}{\partial w_{kl}} = \frac{1}{N}\sum_n \left[\frac{\partial L_n}{\partial a_{l,n}}\right]\left[\frac{\partial a_{l,n}}{\partial w_{kl}}\right]$$

Calculus chain rule

$$\frac{\partial R}{\partial w_{kl}} = \frac{1}{N}\sum_n \left[\frac{\partial \frac{1}{2}(y_n - g(a_{l,n}))^2}{\partial a_{l,n}}\right]\left[\frac{\partial z_{k,n} w_{kl}}{\partial w_{kl}}\right] = \frac{1}{N}\sum_n \left[-(y_n - z_{l,n})g'(a_{l,n})\right] z_{k,n}$$



Error Backpropagation

Optimize last layer weights w_{kl}

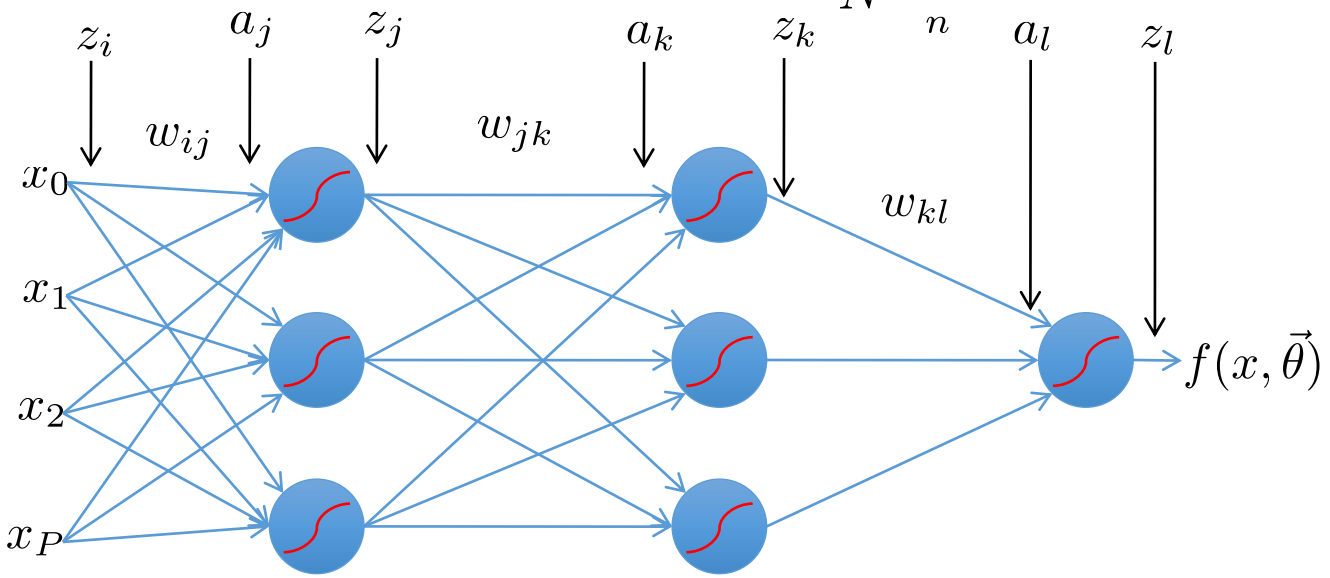
$$L_n = \frac{1}{2}\left(y_n - f(x_n)\right)^2$$

$$\frac{\partial R}{\partial w_{kl}} = \frac{1}{N}\sum_n \left[\frac{\partial L_n}{\partial a_{l,n}}\right]\left[\frac{\partial a_{l,n}}{\partial w_{kl}}\right]$$

Calculus chain rule

$$\frac{\partial R}{\partial w_{kl}} = \frac{1}{N}\sum_n \left[\frac{\partial \frac{1}{2}(y_n - g(a_{l,n}))^2}{\partial a_{l,n}}\right]\left[\frac{\partial z_{k,n} w_{kl}}{\partial w_{kl}}\right] = \frac{1}{N}\sum_n \left[-(y_n - z_{l,n})g'(a_{l,n})\right] z_{k,n}$$

$$= \frac{1}{N}\sum_n \delta_{l,n} n z_{k,n}$$

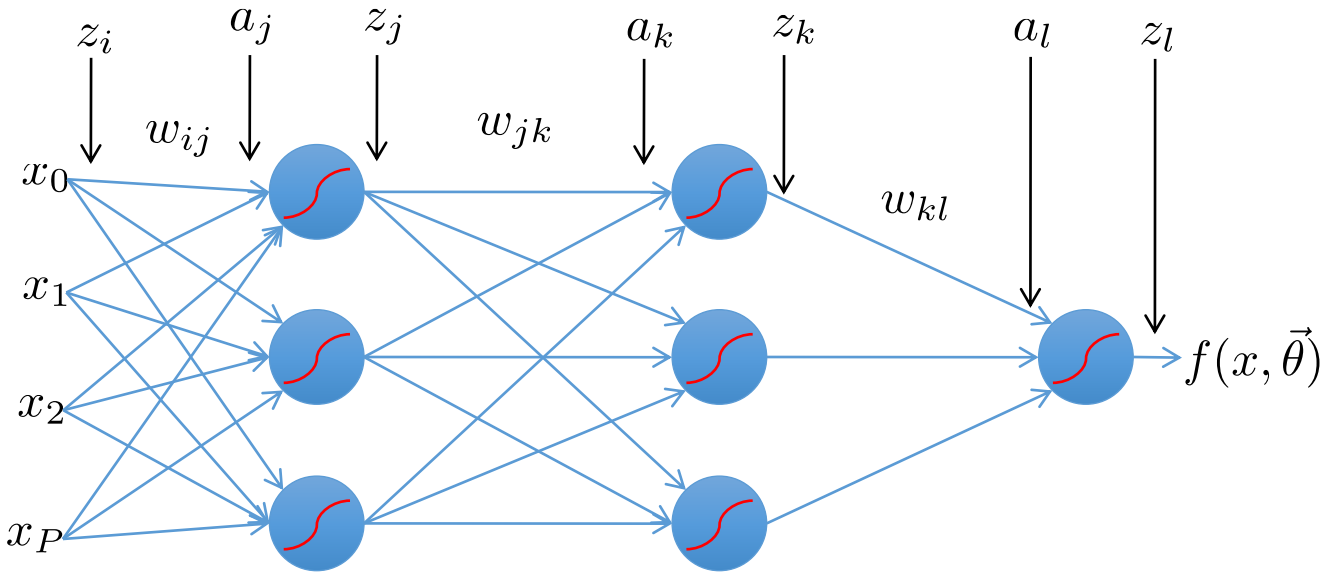


Error Backpropagation

Optimize last hidden weights w_{jk}

$$\frac{\partial R}{\partial w_{jk}} = \frac{1}{N}\sum_n \left[\frac{\partial L_n}{\partial a_{k,n}}\right]\left[\frac{\partial a_{k,n}}{\partial w_{jk}}\right]$$

$$\frac{\partial R}{\partial w_{kl}} = \frac{1}{N}\sum_n \delta_{l,n} z_{k,n}$$



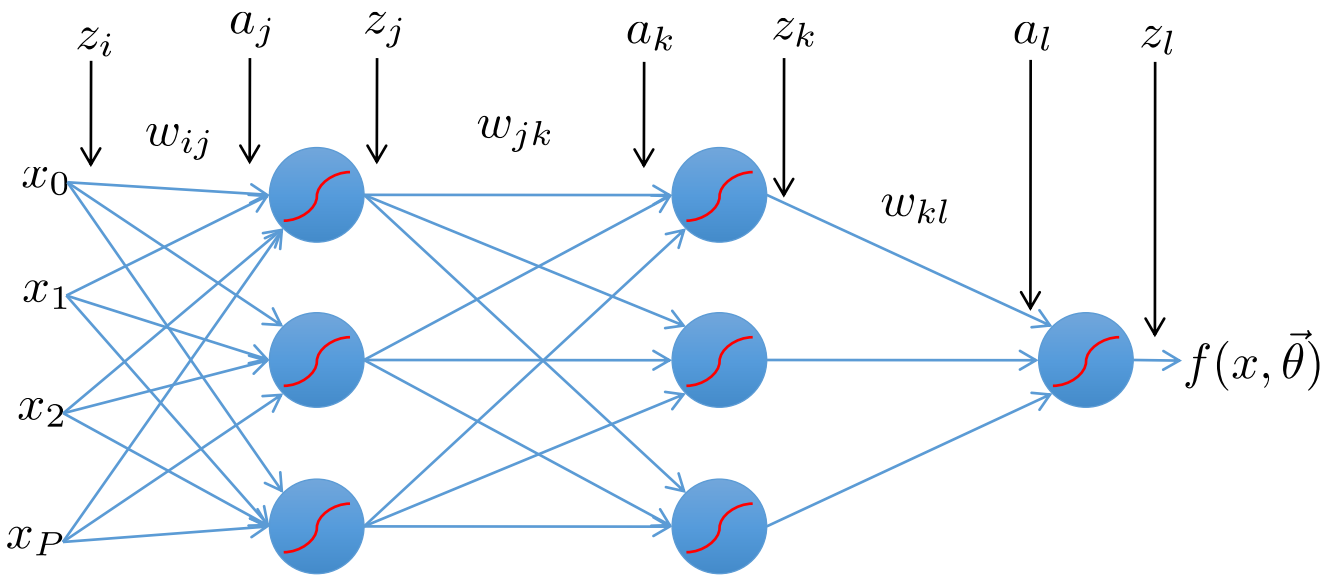
Error Backpropagation

Optimize last hidden weights w_{jk}

$$\frac{\partial R}{\partial w_{kl}} = \frac{1}{N}\sum_n \delta_{l,n} z_{k,n}$$

$$\frac{\partial R}{\partial w_{jk}} = \frac{1}{N}\sum_n \left[\sum_l \frac{\partial L_n}{\partial a_{l,n}} \frac{\partial a_{l,n}}{\partial a_{k,n}}\right] \left[\frac{\partial a_{k,n}}{\partial w_{jk}}\right]$$

Multivariate chain rule



Error Backpropagation

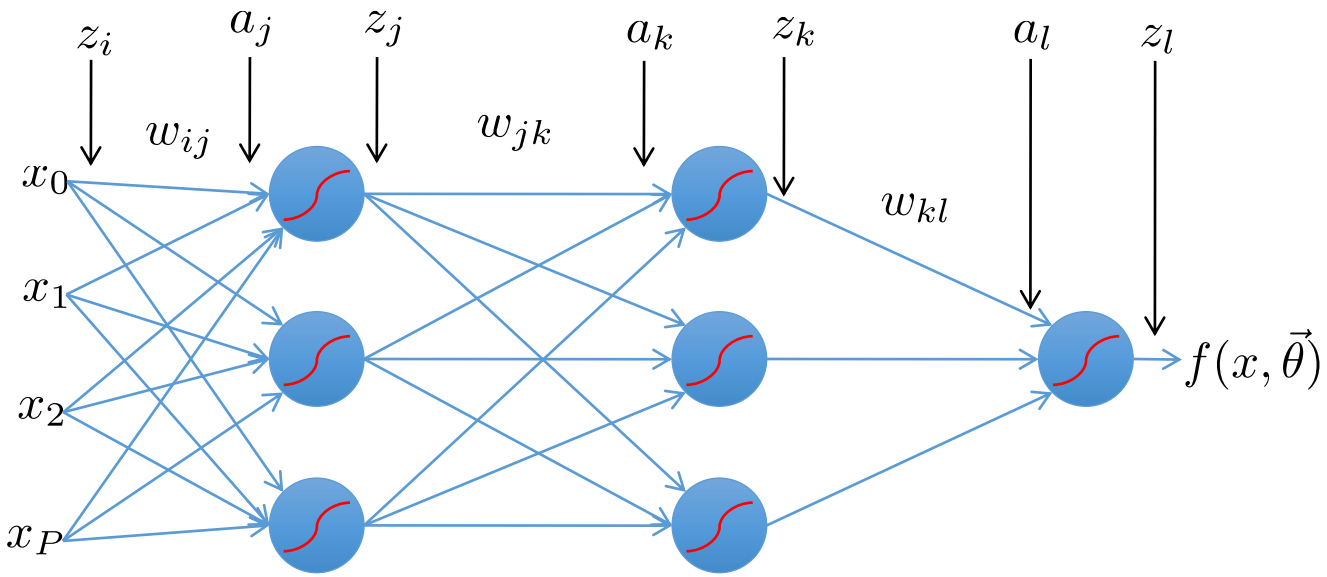
Optimize last hidden weights w_{jk}

$$\frac{\partial R}{\partial w_{kl}} = \frac{1}{N}\sum_n \delta_{l,n} z_{k,n}$$

$$\frac{\partial R}{\partial w_{jk}} = \frac{1}{N}\sum_n \left[\sum_l \frac{\partial L_n}{\partial a_{l,n}} \frac{\partial a_{l,n}}{\partial a_{k,n}}\right] \left[\frac{\partial a_{k,n}}{\partial w_{jk}}\right]$$

Multivariate chain rule

$$\frac{\partial R}{\partial w_{jk}} = \frac{1}{N}\sum_n \left[\sum_l \delta_l \frac{\partial a_{l,n}}{\partial a_{k,n}}\right] [z_{j,n}]$$



Error Backpropagation

Optimize last hidden weights w_{jk}

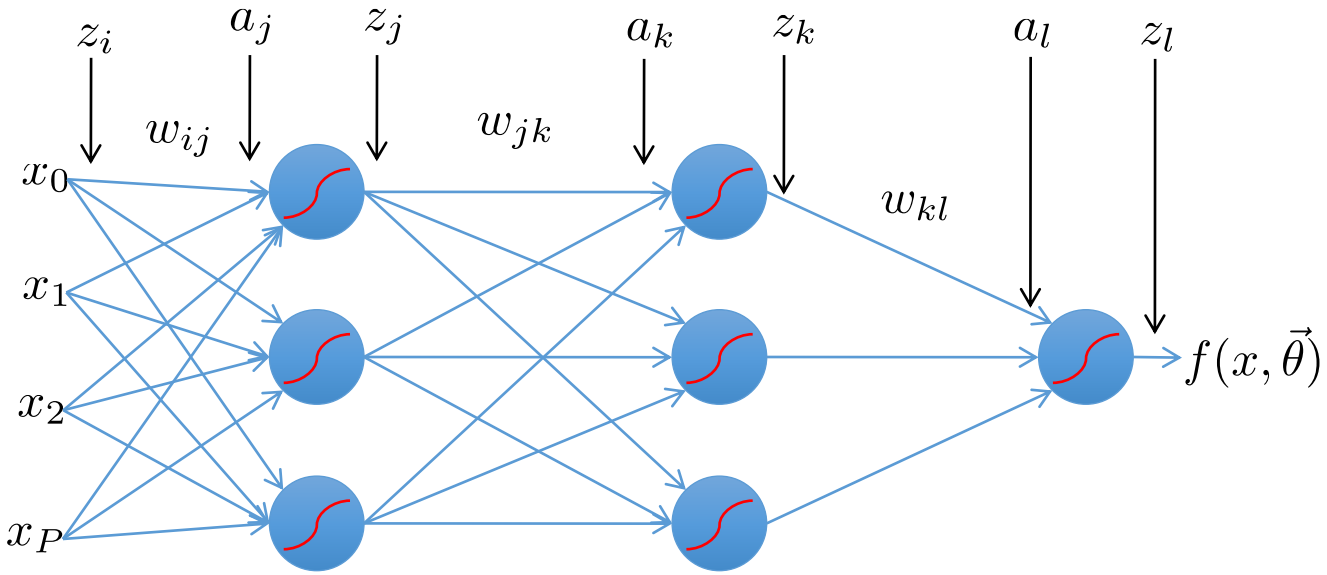
$$\frac{\partial R}{\partial w_{kl}} = \frac{1}{N}\sum_n \delta_{l,n} z_{k,n}$$

$$\frac{\partial R}{\partial w_{jk}} = \frac{1}{N}\sum_n \left[\sum_l \frac{\partial L_n}{\partial a_{l,n}} \frac{\partial a_{l,n}}{\partial a_{k,n}}\right] \left[\frac{\partial a_{k,n}}{\partial w_{jk}}\right]$$

Multivariate chain rule

$$\frac{\partial R}{\partial w_{jk}} = \frac{1}{N}\sum_n \left[\sum_l \delta_l \frac{\partial a_{l,n}}{\partial a_{k,n}}\right] [z_{j,n}]$$

$$a_l = \sum_k w_{kl} g(a_k)$$



Error Backpropagation

Optimize last hidden weights w_{jk}

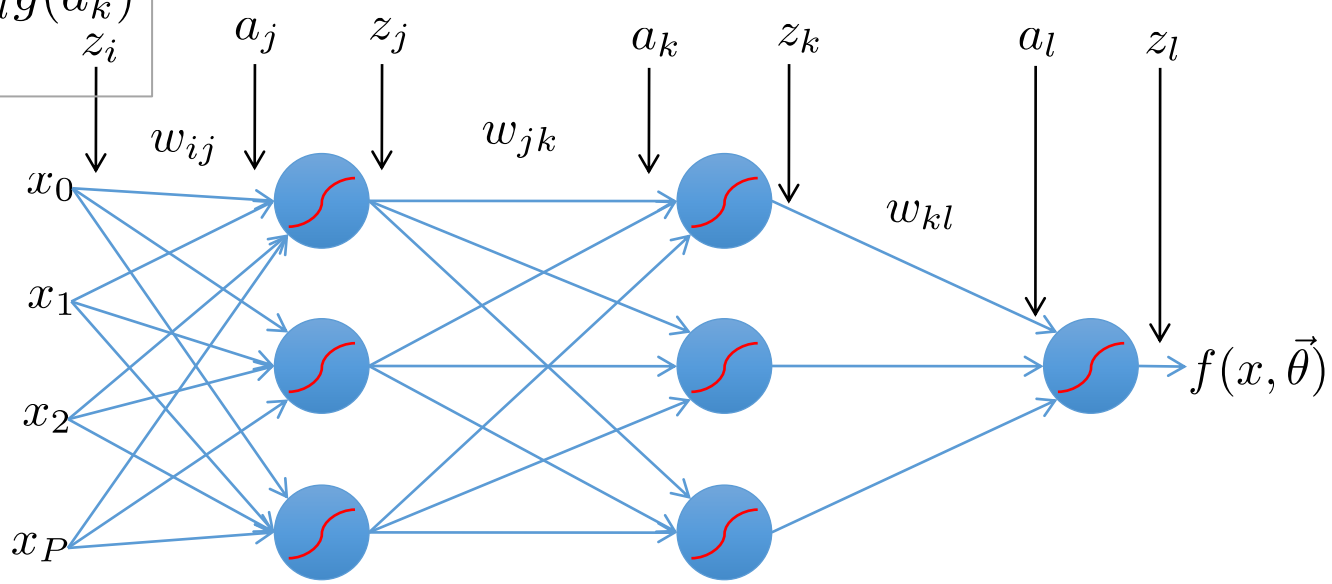
$$\frac{\partial R}{\partial w_{kl}} = \frac{1}{N}\sum_n \delta_{l,n} z_{k,n}$$

$$\frac{\partial R}{\partial w_{jk}} = \frac{1}{N}\sum_n \left[\sum_l \frac{\partial L_n}{\partial a_{l,n}}\frac{\partial a_{l,n}}{\partial a_{k,n}}\right]\left[\frac{\partial a_{k,n}}{\partial w_{jk}}\right]$$

Multivariate chain rule

$$\frac{\partial R}{\partial w_{jk}} = \frac{1}{N}\sum_n \left[\sum_l \delta_l w_{kl} g'(a_{k,n})\right][z_{j,n}] = \frac{1}{N}\sum_n [\delta_{k,n}][z_{j,n}]$$

$$a_l = \sum_k w_{kl} g(a_k)$$



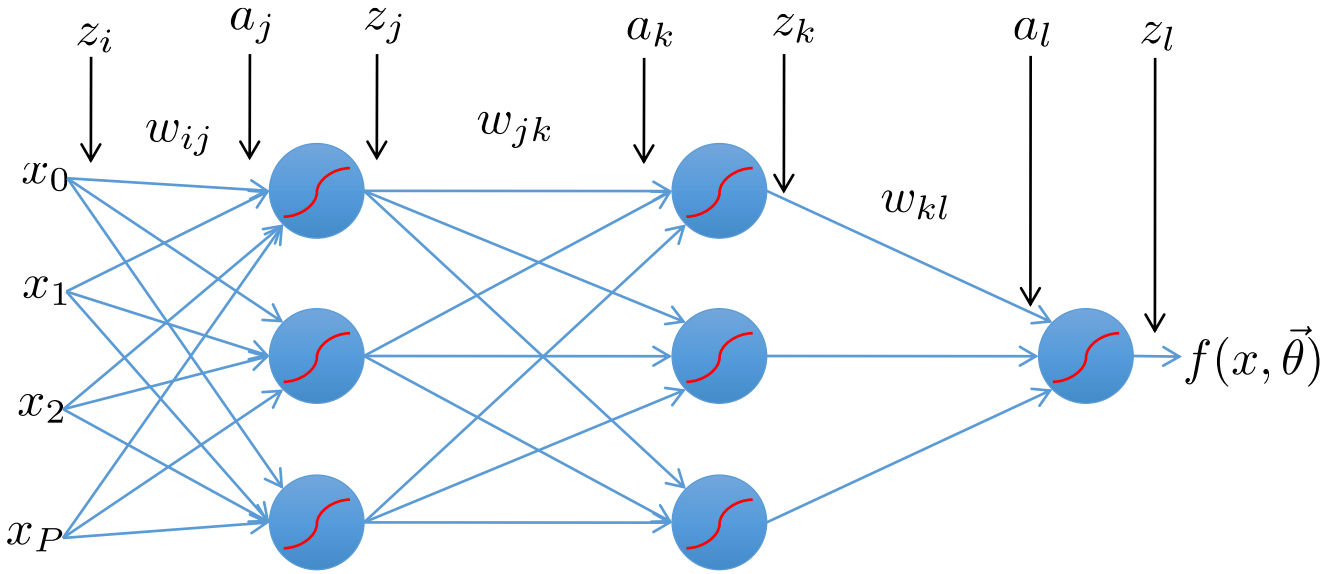
Error Backpropagation

Repeat for all previous layers

$$\frac{\partial R}{\partial w_{kl}} = \frac{1}{N}\sum_n \left[\frac{\partial L_n}{\partial a_{l,n}}\right]\left[\frac{\partial a_{l,n}}{\partial w_{kl}}\right] = \frac{1}{N}\sum_n \left[-(y_n - z_{l,n})g'(a_{l,n})\right] z_{k,n} = \frac{1}{N}\sum_n \delta_{l,n} z_{k,n}$$

$$\frac{\partial R}{\partial w_{jk}} = \frac{1}{N}\sum_n \left[\frac{\partial L_n}{\partial a_{k,n}}\right]\left[\frac{\partial a_{k,n}}{\partial w_{jk}}\right] = \frac{1}{N}\sum_n \left[\sum_l \delta_{l,n} w_{kl} g'(a_{k,n})\right] z_{j,n} = \frac{1}{N}\sum_n \delta_{k,n} z_{j,n}$$

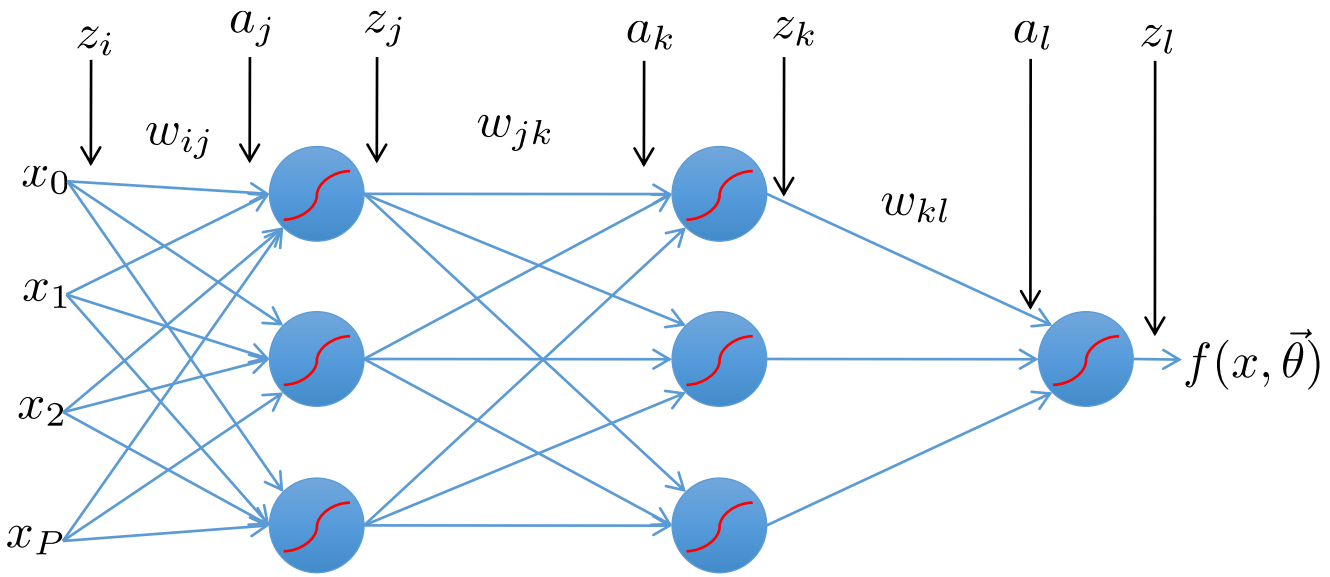
$$\frac{\partial R}{\partial w_{ij}} = \frac{1}{N}\sum_n \left[\frac{\partial L_n}{\partial a_{j,n}}\right]\left[\frac{\partial a_{j,n}}{\partial w_{ij}}\right] = \frac{1}{N}\sum_n \left[\sum_k \delta_{k,n} w_{jk} g'(a_{j,n})\right] z_{i,n} = \frac{1}{N}\sum_n \delta_{j,n} z_{i,n}$$



Error Backpropagation

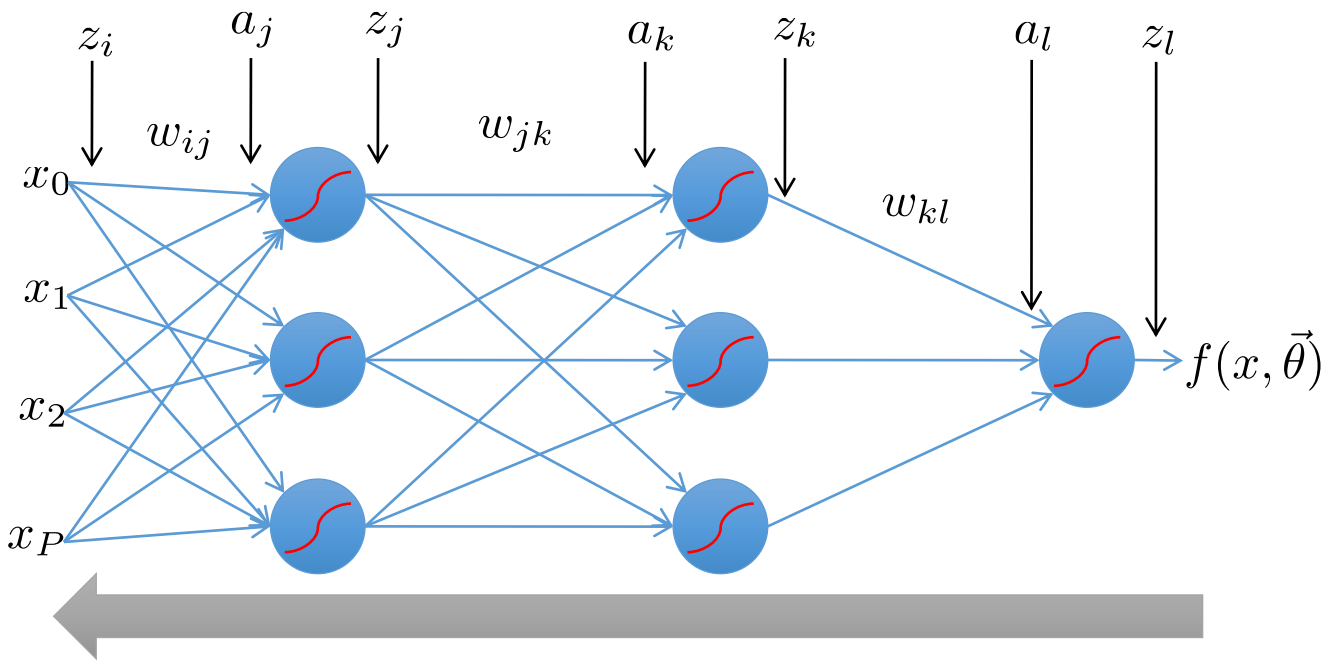
Now that we have well defined gradients for each parameter, update using Gradient Descent

$$
\begin{aligned}
w_{ij}^{t+1} &= w_{ij}^{t} - \eta \frac{\partial R}{w_{ij}} \\
w_{jk}^{t+1} &= w_{jk}^{t} - \eta \frac{\partial R}{w_{jk}} \\
w_{kl}^{t+1} &= w_{kl}^{t} - \eta \frac{\partial R}{w_{kl}}
\end{aligned}
$$



Error Back-propagation

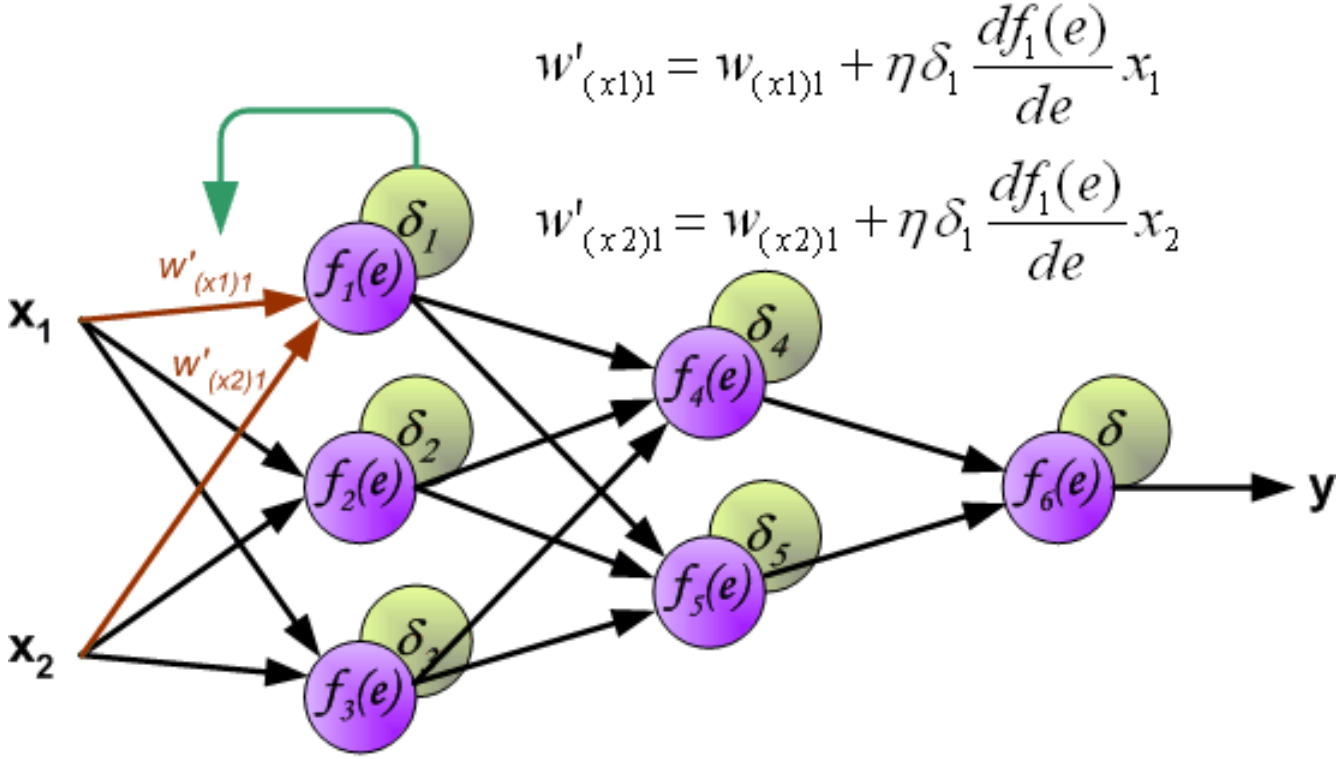
- Error backpropagation unravels the multivariate chain rule and solves the gradient for each partial component separately.
- The target values for each layer come from the next layer.
- This feeds the errors back along the network.



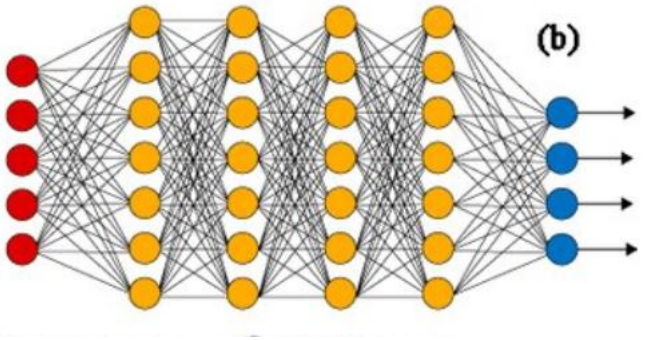
Learning with error backpropagation

- randomly initialize parameters (weights)
- compute error on the output
- compute contributions to error, δ_n, on each step backwards

- step
- epoch
- batch
- minibatch



Defining a network topology



- Decide the network topology:
  Specify # of units in the *input layer*, # of *hidden layers* (if > 1), # of units in *each hidden layer*, and # of units in the *output layer*
- Normalize the input values for each attribute measured in the training tuples to [0.0—1.0]
- One input unit per discrete attribute value, 1-hot encoded
- For classification and more than two classes, one output unit per class is used
- Once a network has been trained and its accuracy is still unacceptable, repeat the training process with a *different network topology* or a *different set of initial weights*

Neural network: strengths and weaknesses

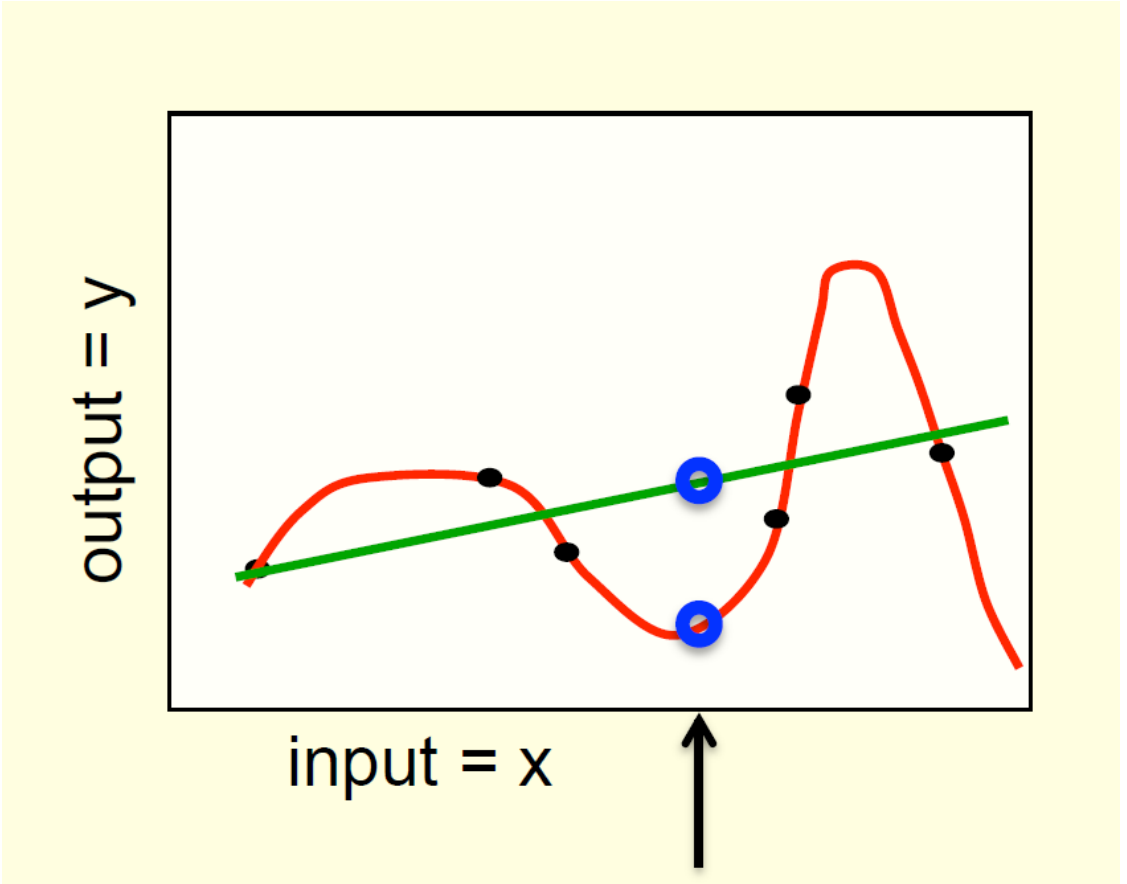
- Weakness
  - Long training time
  - Require a number of parameters typically best determined empirically, e.g., the network topology or “structure.”
  - Poor interpretability: difficult to interpret the symbolic meaning behind the learned weights and of “hidden units” in the network
  - Easy to overfit without an evaluation set
- Strength
  - High tolerance to noisy data
  - Good generalization to untrained patterns
  - Well-suited for continuous-valued inputs *and outputs*
  - Algorithms are inherently parallel
  - Builds more advanced representation
  - Successful on an array of real-world data, especially images, text, and time-series, e.g., one of the early successful deep networks was applied to hand-written letters
  - Techniques exist for the extraction of explanations from trained (small) neural networks

Efficiency and interpretability

- Efficiency of backpropagation:
  Each epoch (one iteration through the training set) takes O(|D| * *w*), with |D| training instances and *w* weights, but # of epochs can be exponential to n, the number of inputs, in worst case (not in practice)
- For easier comprehension: Rule extraction by network pruning
  - Simplify the network structure by removing weighted links that have the least effect on the trained network
  - Then perform link, unit, or activation value clustering
  - The set of input and activation values are studied to derive rules describing the relationship between the input and hidden unit layers
- Sensitivity analysis: assess the impact that a given input variable has on a network output. The knowledge gained from this analysis can be represented in rules
- Recent attempts tend to learn interpretation along with learning

Overfitting and model complexity

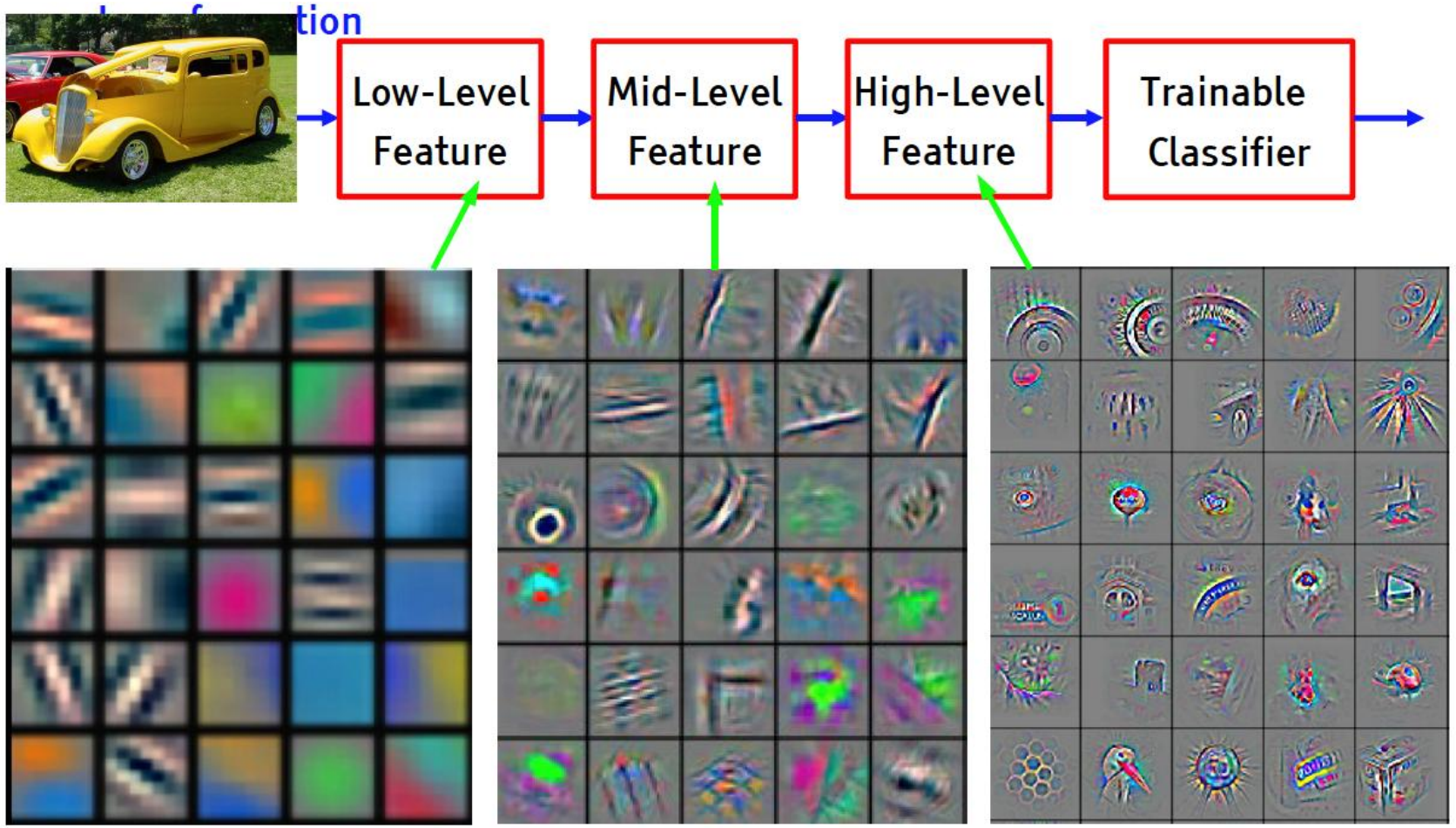
- which curve is more plausible given the data?
- overfitting
- neural nets are especially prone to overfitting
- why?



Prevention of overfitting

- Evaluation set
- Weight-decay
- Weight-sharing
- Early stopping
- Model averaging
- Bayesian fitting of neural nets:
  - distributions instead of weights,
  - inference as sampling from distributions
- Dropout
- Generative pre-training
- etc.

Deep learning = learning of hierarchical represenation



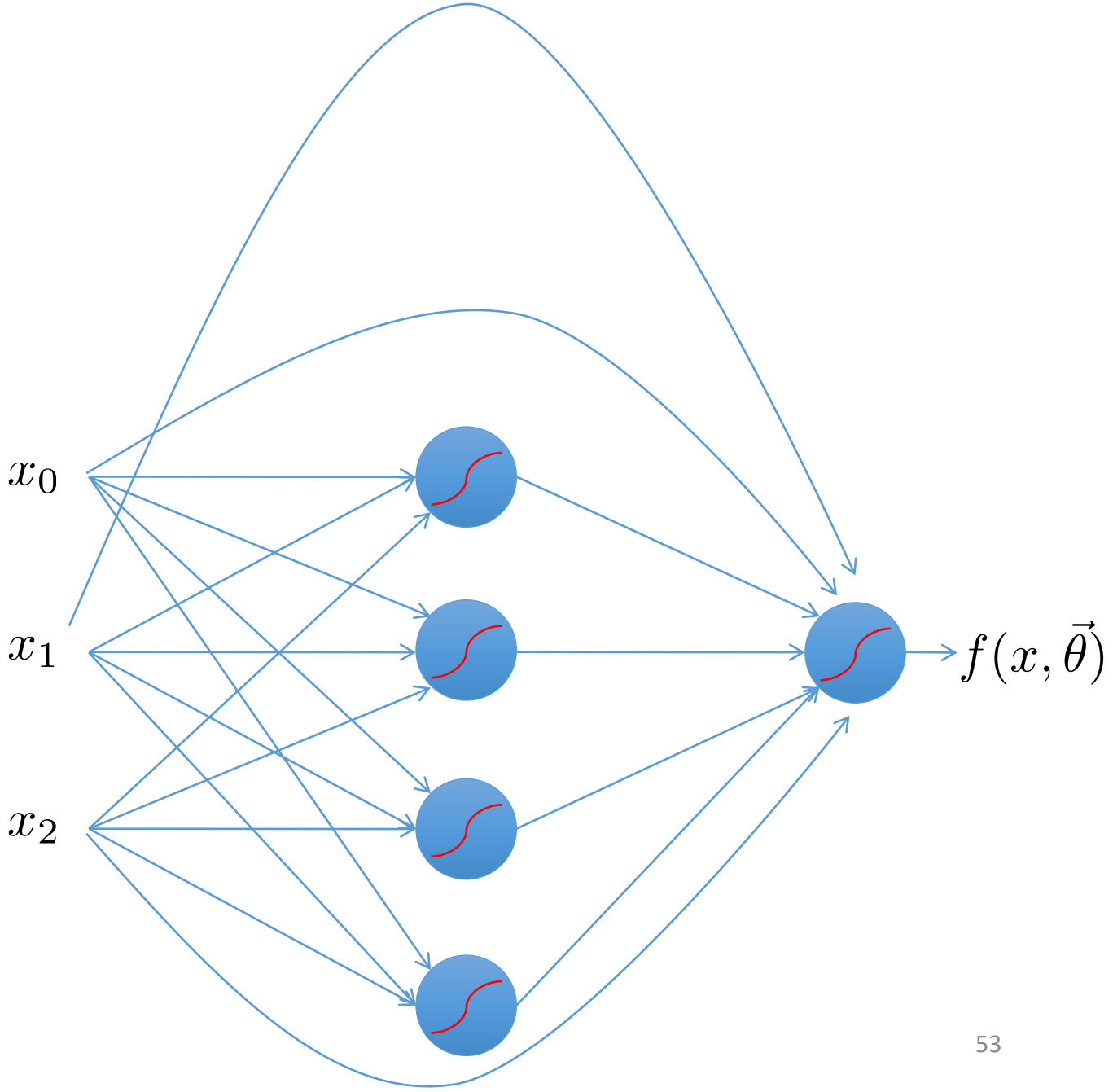
Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

Types of NN architectures

- Historically, feed-forward networks were the most commonly used; here, neurons are activated progressively through layers from input to output
- However, we often combine different types of layers
- Examples of other architectures: recurrent, convolutional, transformer

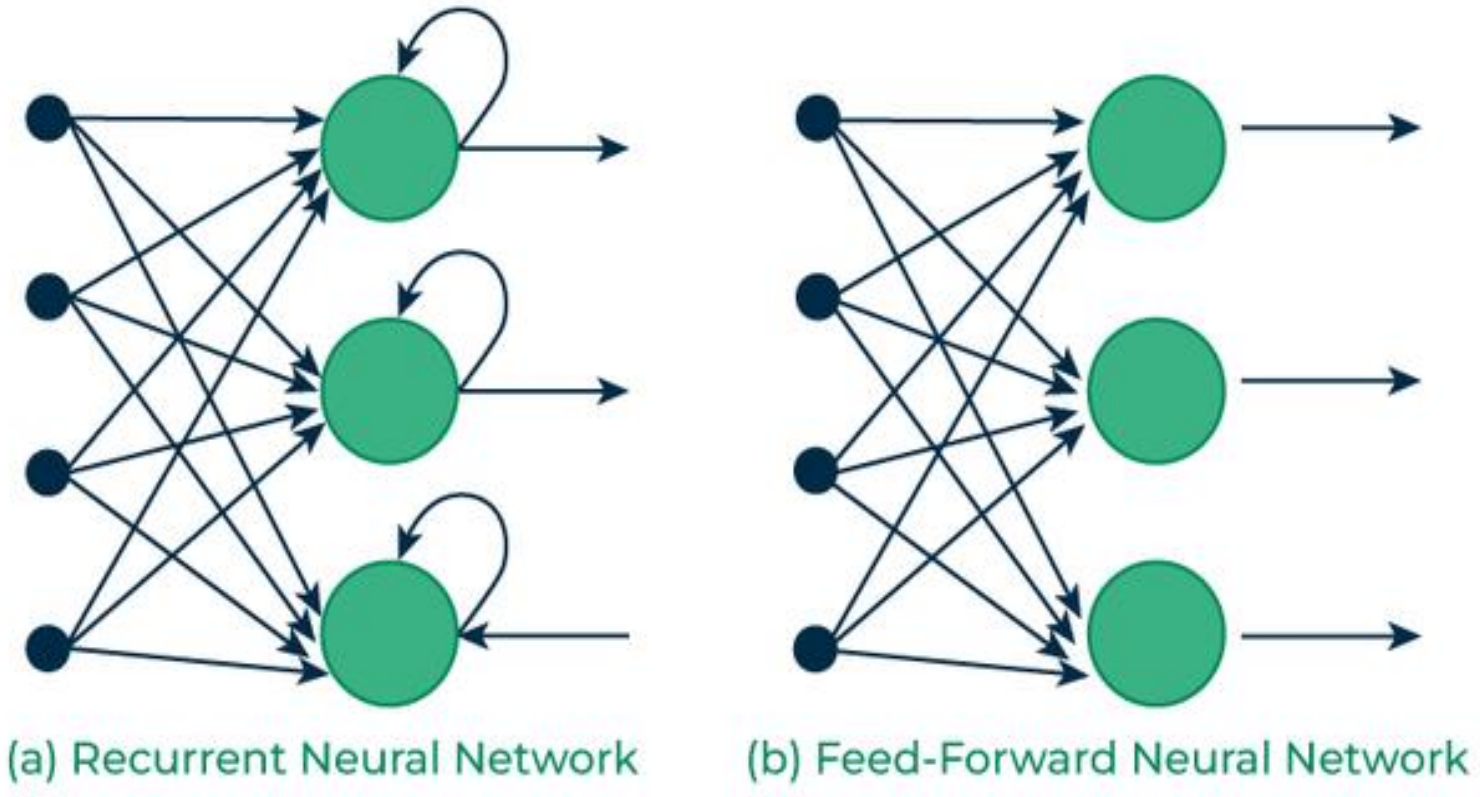
Another option: Level jumping or Skip-connections

- prevents vanishing gradients



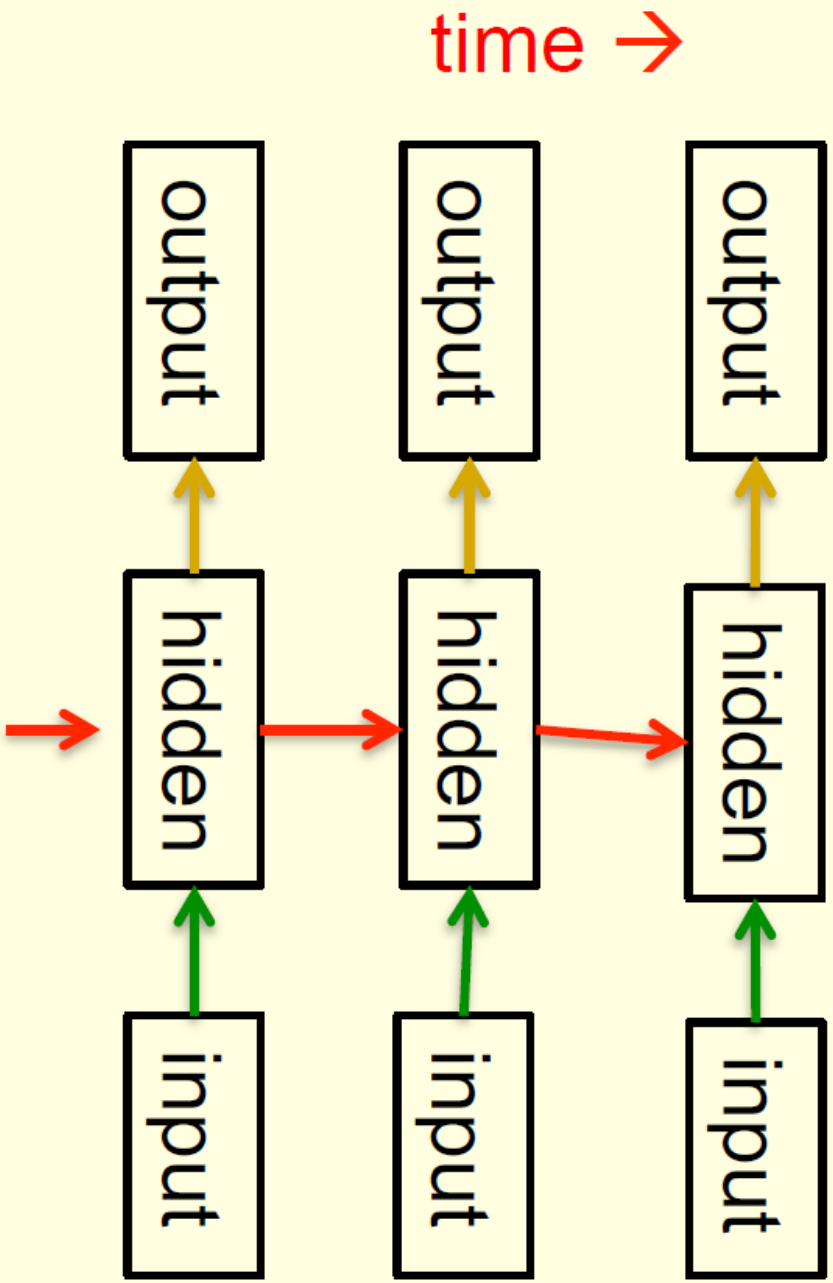
Recurrent networks

- back connections
- biologically more realistic
- store information from the past
- more difficult to learn

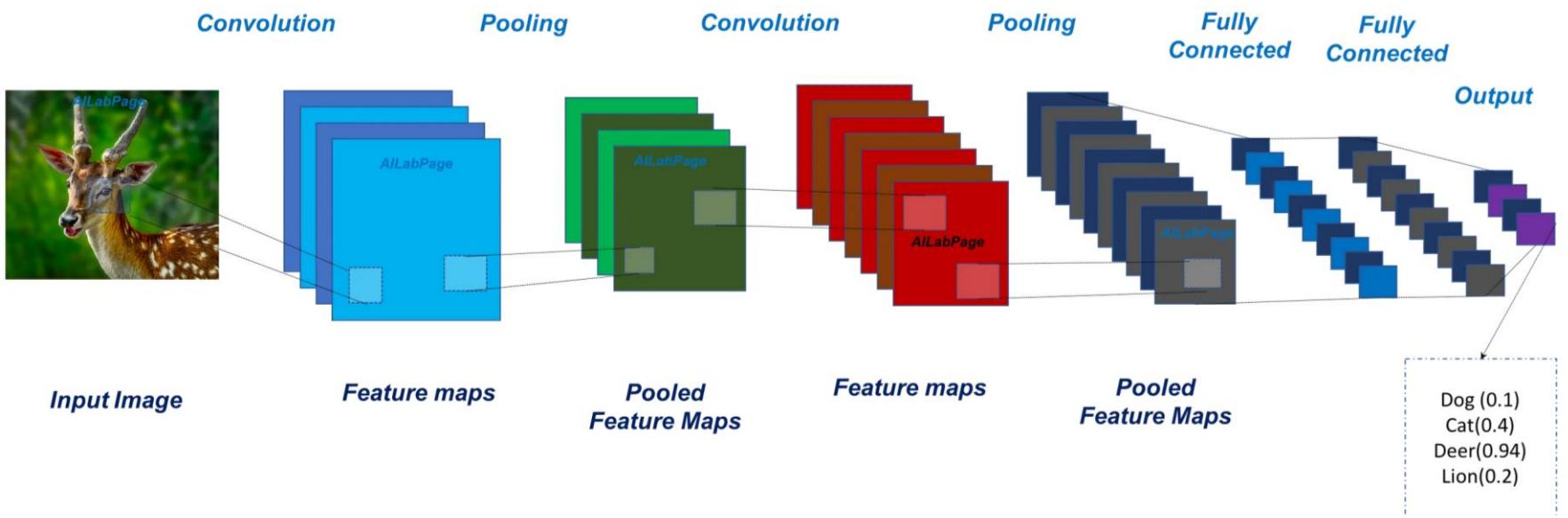


Recurrent networks for sequence learning

- unrolled network
- equivalent to deep networks with one hidden level per time slot
- but: hidden layers share weight (less parameters)



Convolutional neural networks (CNN)



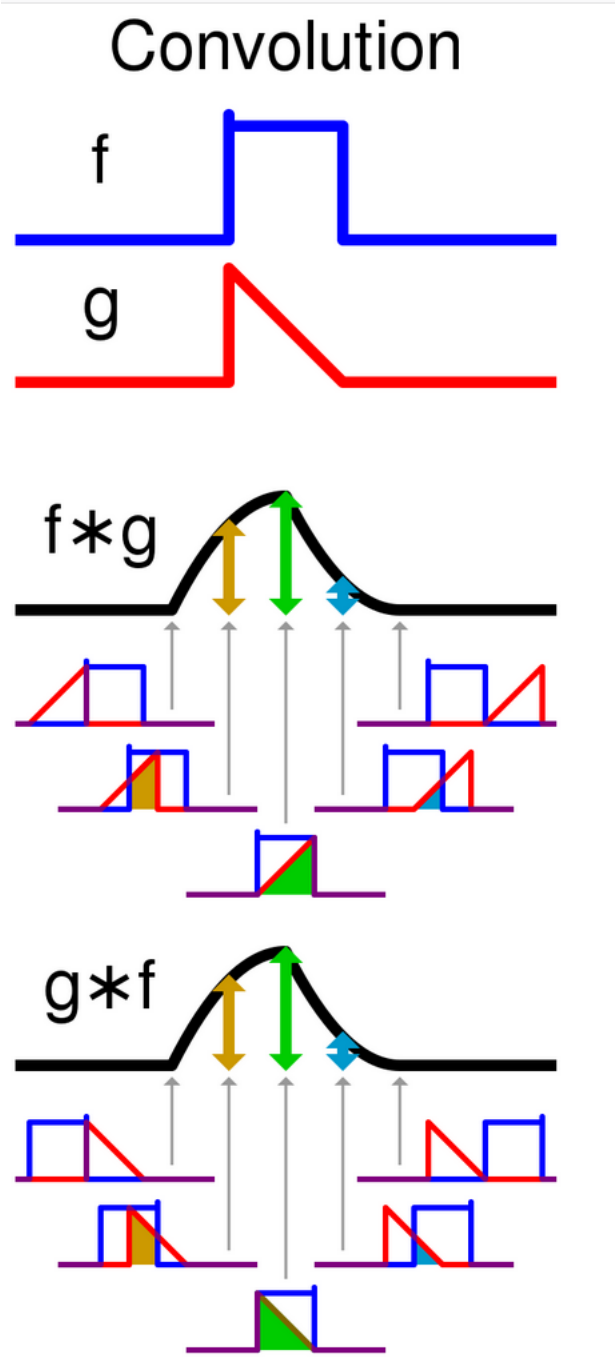
Convolution

- an operation on two functions (f and g) that produces a third function expressing how the shape of one is modified by the other.

$$(f * g)(t) \triangleq \int_{-\infty}^{\infty} f(\tau) g(t - \tau)\, d\tau.$$

- for discrete functions

$$(f * g)[n] = \sum_{m=-\infty}^{\infty} f[m] g[n - m]$$

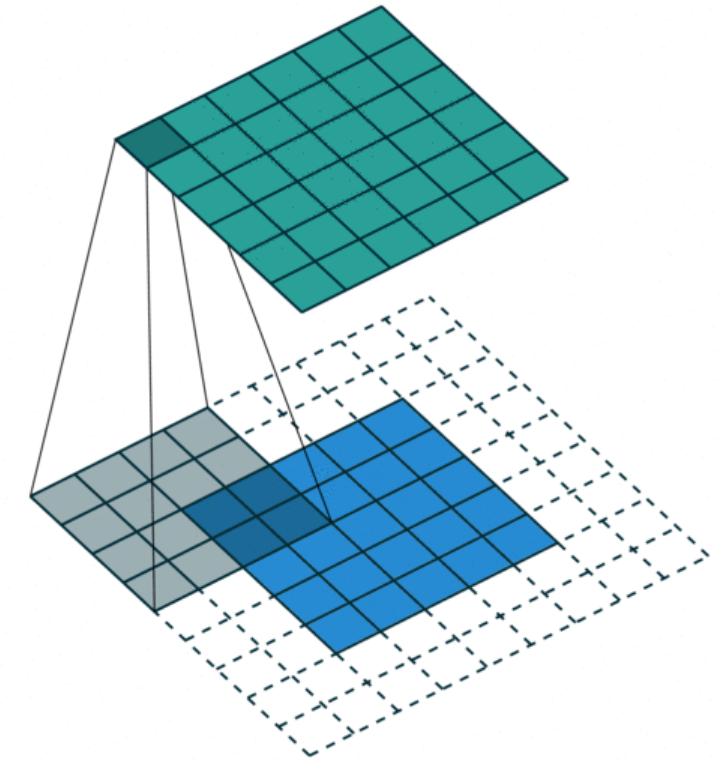


Convolutional Neural Network (CNN)

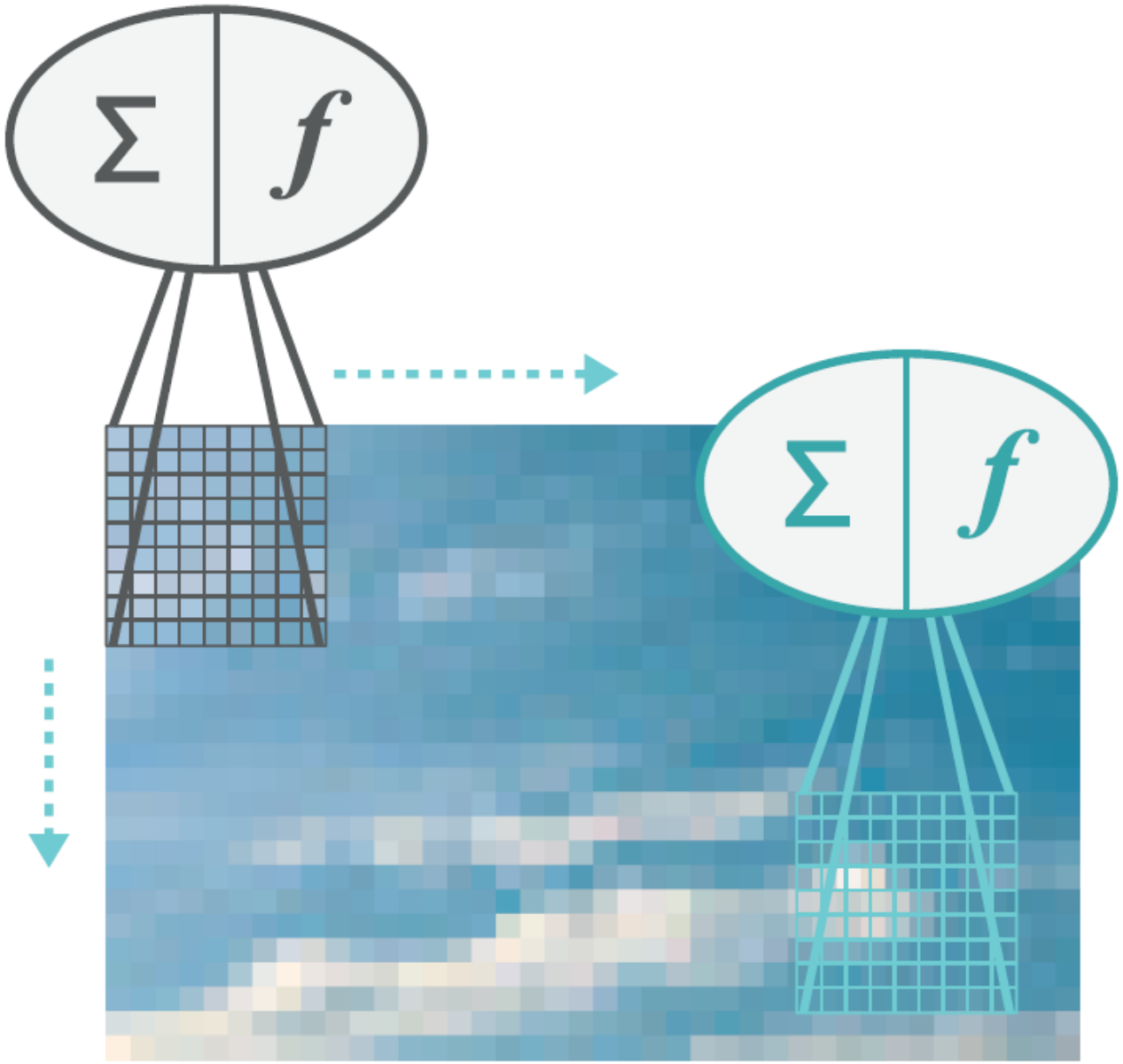
- Convolutional Neural Networks are inspired by mammalian visual cortex.
  - The visual cortex contains a complex arrangement of cells, which are sensitive to small sub-regions of the visual field, called a receptive field. These cells act as local filters over the input space and are well-suited to exploit the strong spatially local correlation present in natural images.
  - Two basic cell types:
    - Simple cells respond maximally to specific edge-like patterns within their receptive field.
    - Complex cells have larger receptive fields and are locally invariant to the exact position of the pattern.

Convolutional neural networks (CNN)

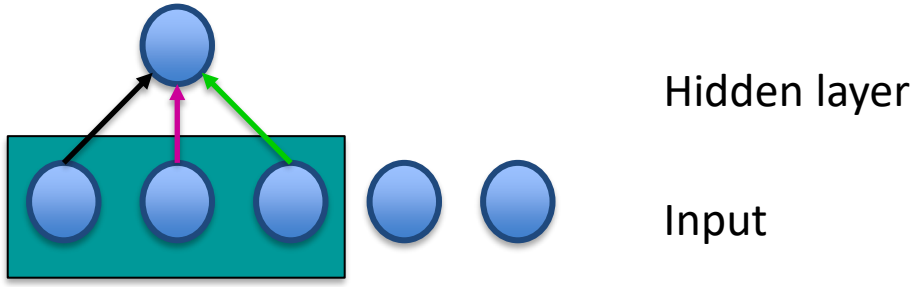
- a successful approach in image analysis, also used in language processing
- idea: many copies of small detectors used all over the image, recursively combined,
- detectors are learned, combinations are learned



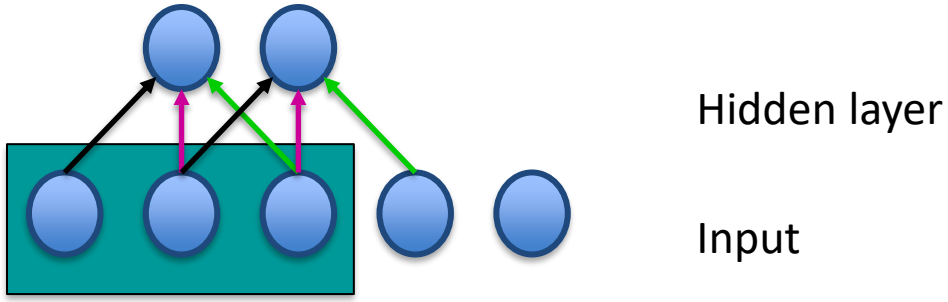
2d convolution for images



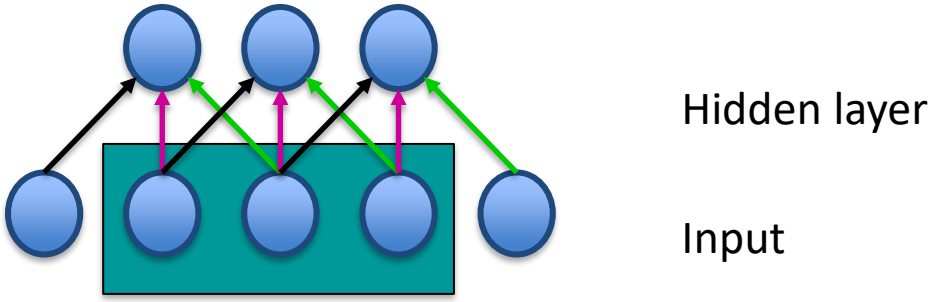
Basic Idea of CNNs



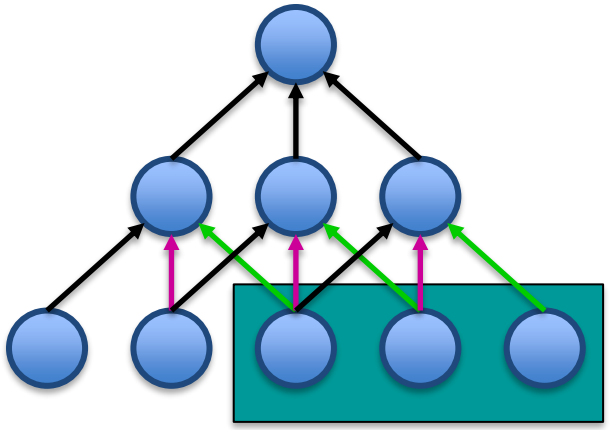
Basic Idea of CNNs



Basic Idea of CNNs



Basic Idea of CNNs

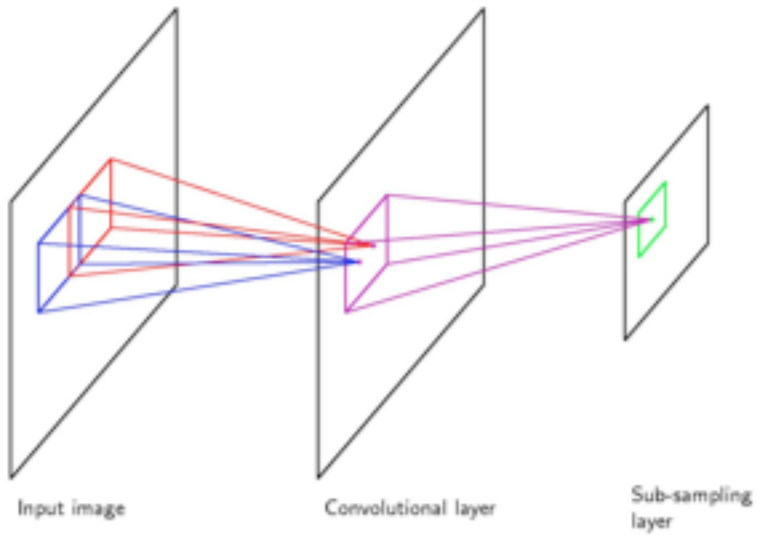


Hidden layer

Input

Convolutional Network

- The network is not fully connected.
- Different nodes are responsible for different regions of the image.
- This allows for robustness to transformations.
- Convolution is combined with subsampling.



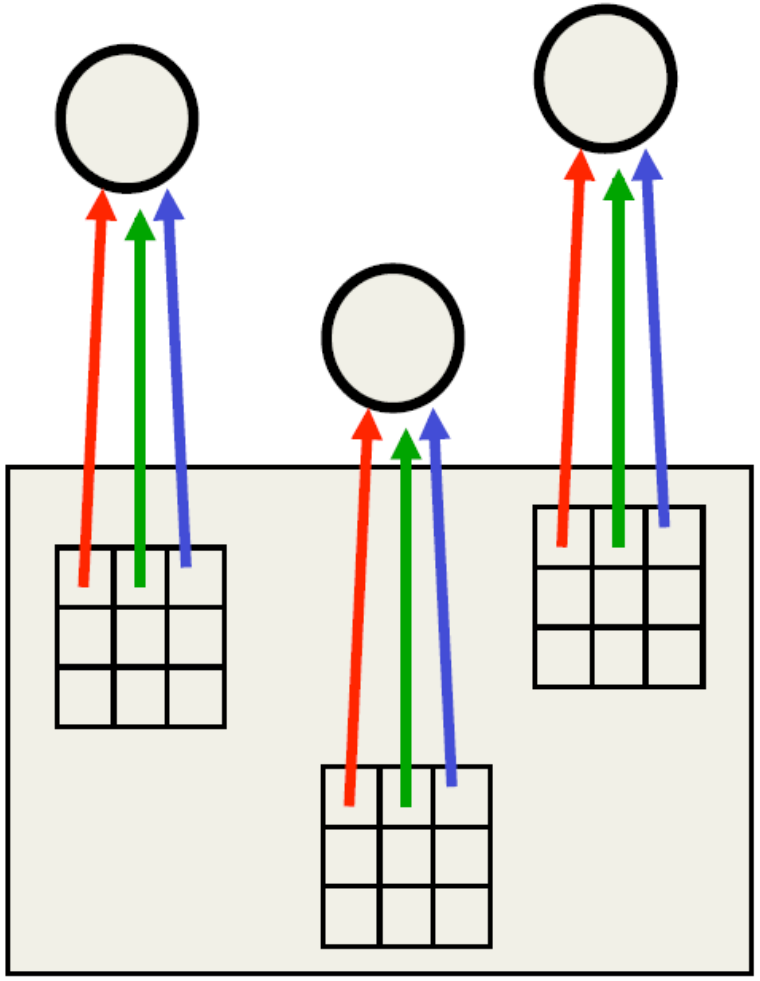
CNN Architecture: Convolutional Layer

- The core layer of CNNs
- The convolutional layer consists of a set of filters.
  - Each filter covers a spatially small portion of the input data.
- Each filter is convolved across the dimensions of the input data, producing a multidimensional feature map.
  - As we convolve the filter, we are computing the dot product between the parameters of the filter and the input.
- Intuition: the network will learn filters that activate when they see some specific type of feature at some spatial position in the input.
- The key architectural characteristics of the convolutional layer is local connectivity and shared weights.

Neural implementation of convolution

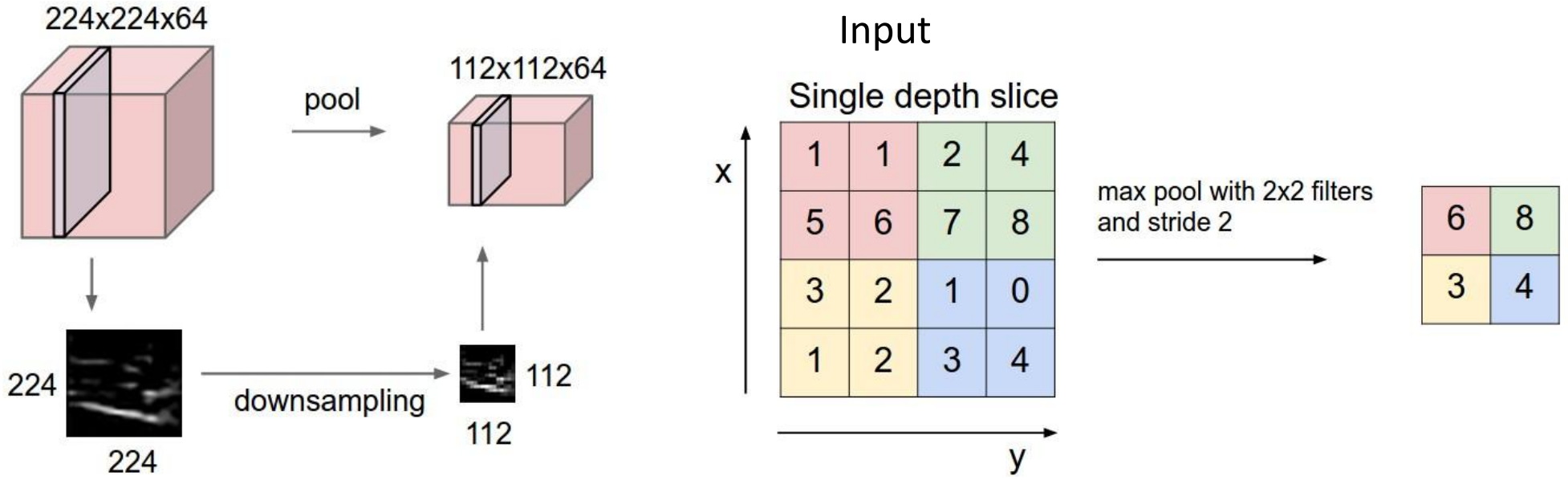
- weights of the same colors have equal weights
- adapted backpropagation

- images: 2d convolution
- languages: 1d convolution



CNN Architecture: Pooling Layer

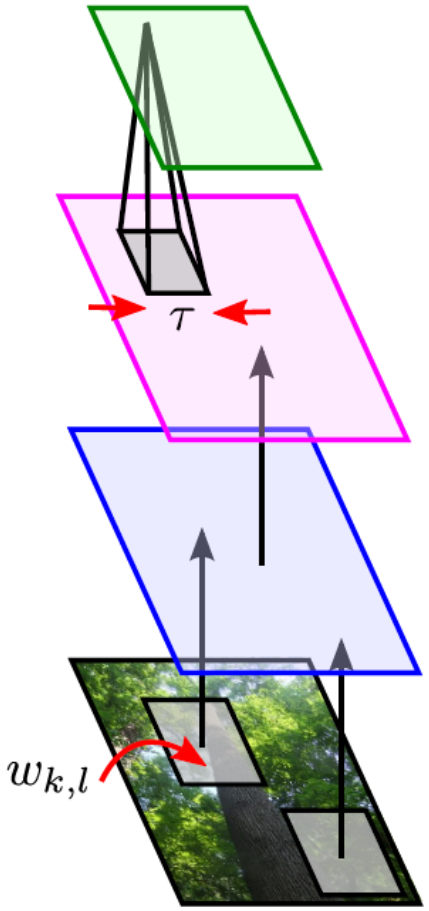
- Intuition: to progressively reduce the spatial size of the representation, to reduce the amount of parameters and computation in the network, and hence to also control overfitting
- Pooling partitions the input image into a set of non-overlapping rectangles and, for each such sub-region, usually outputs the maximum value of the features in that region.



CNN: pooling

- reduces the number of connections to the next layer (prevents the excessive number of parameters, speeds-up learning, reduces overfitting)
- max-pooling, min-pooling, average-pooling
- the problem: after several layers of pooling, we lose the information about the exact location of the recognized pattern and about spatial relations between different patterns and features, e.g., a nose on a forehead

Building-blocks for CNN's



$$x_{i,j} = \max_{|k|<\tau,|l|<\tau} y_{i-k,j-l}$$

mean or subsample also used

pooling stage

Feature maps of a larger region are combined.

$$y_{i,j} = f(a_{i,j})$$

e.g. $f(a) = [a]_+$

$f(a) = \text{sigmoid}(a)$

non-linear stage

Feature maps are trained with neurons.

Shared weights

$$a_{i,j} = \sum_{k,l} w_{k,l} z_{i-k,j-l}$$

only parameters

convolutional stage

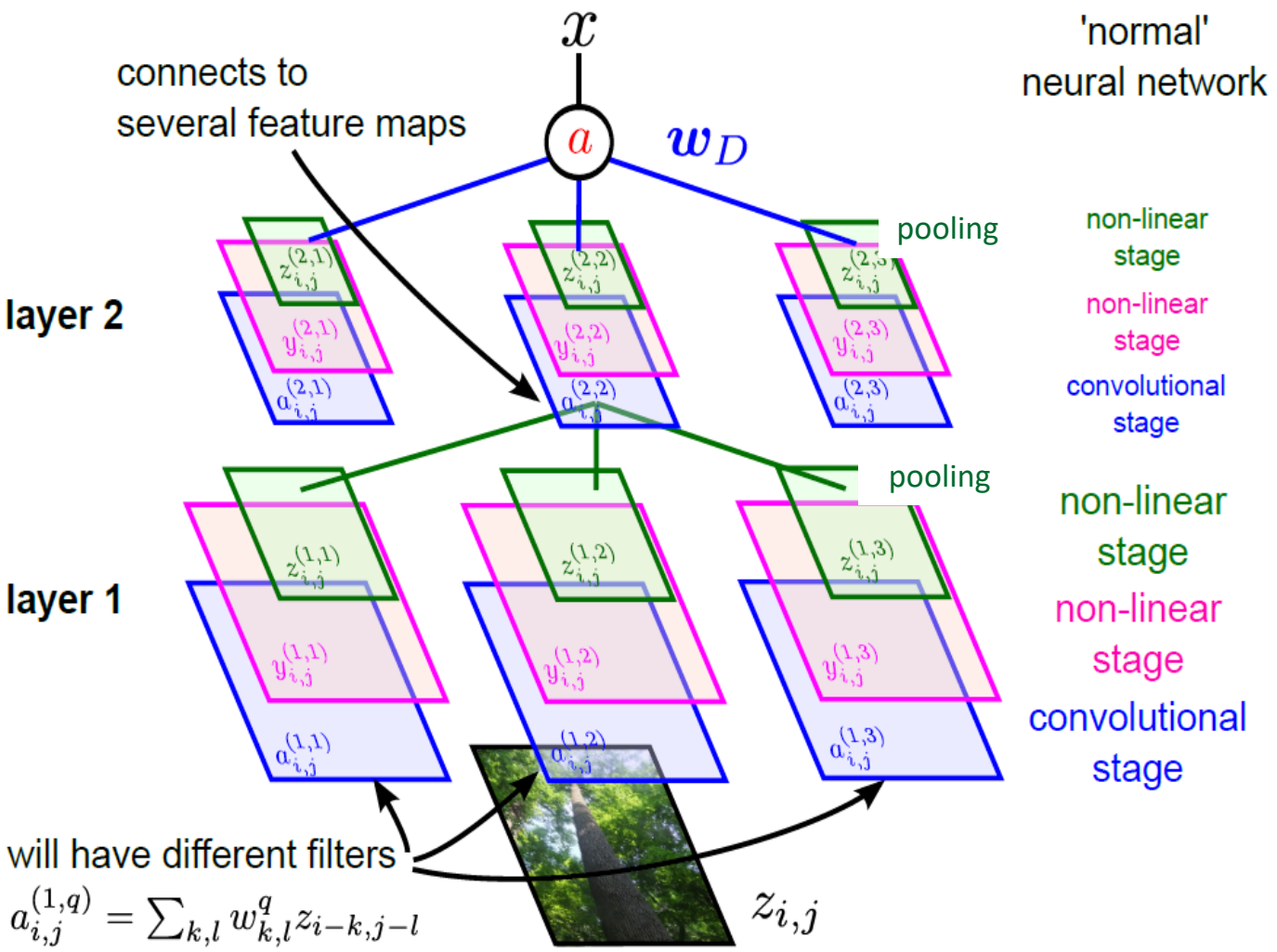
Each sub-region yields a feature map, representing its feature.

$z_{i,j}$

input image

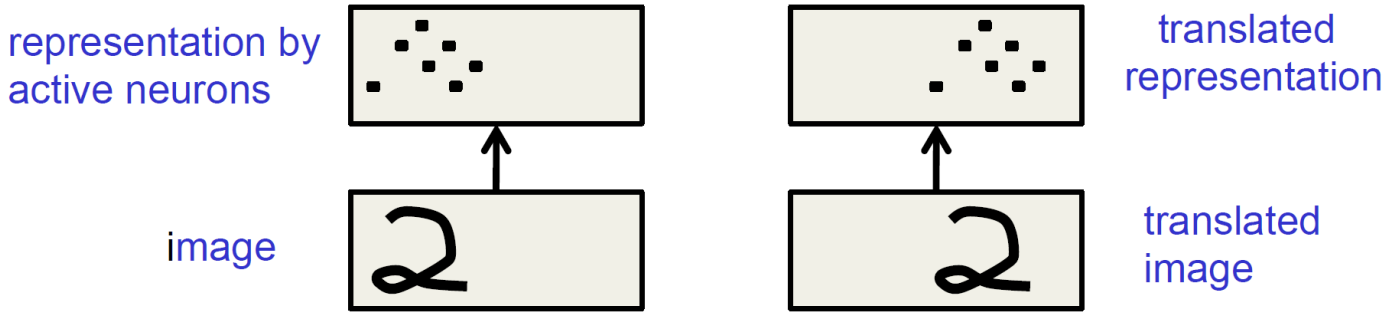
Images are segmented into sub-regions.

Full CNN


x
connects to
several feature maps
a
w_D
'normal'
neural network
pooling
non-linear
stage
non-linear
stage
convolutional
stage
layer 2
pooling
layer 1
non-linear
stage
non-linear
stage
convolutional
stage
will have different filters
$a_{i,j}^{(1,q)} = \sum_{k,l} w_{k,l}^{q} z_{i-k,j-l}$
$z_{i,j}$

Convolutional networks: illustration on image recognition

- a useful feature is learned and used on several positions
- prevents dimension hopping
- max-pooling

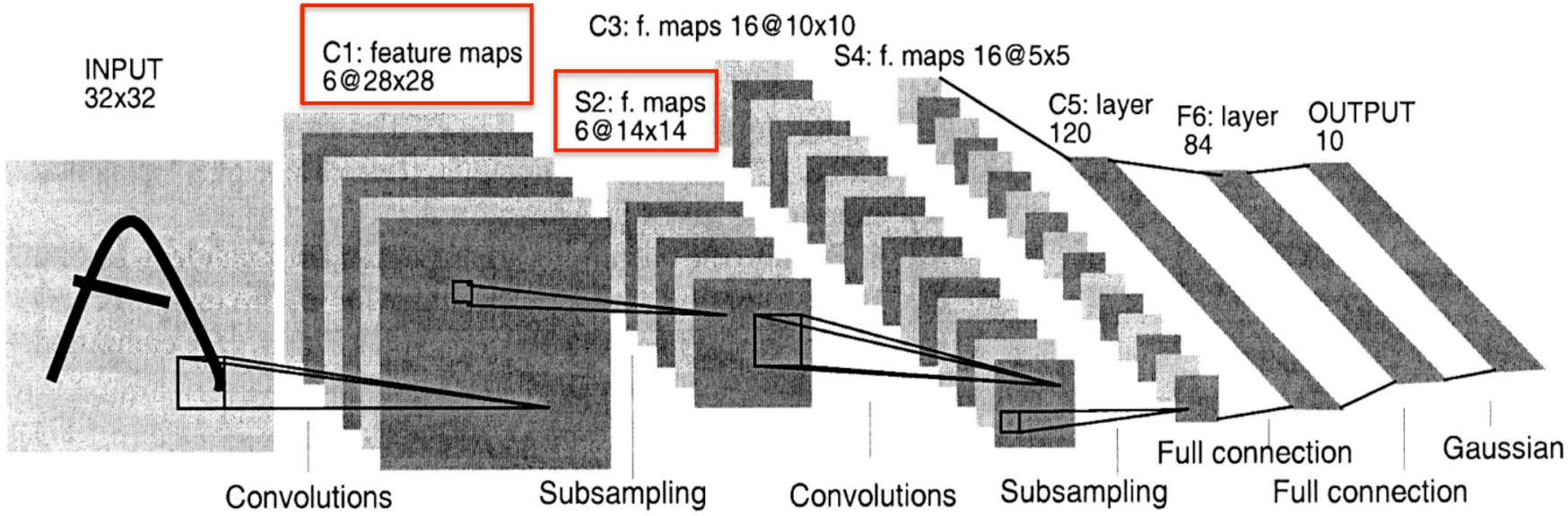


CNN early success: LeNet

- handwritten digit recognition by Yann LeCun,
- several hidden layers
- several convolutional filters
- pooling
- several other tricks

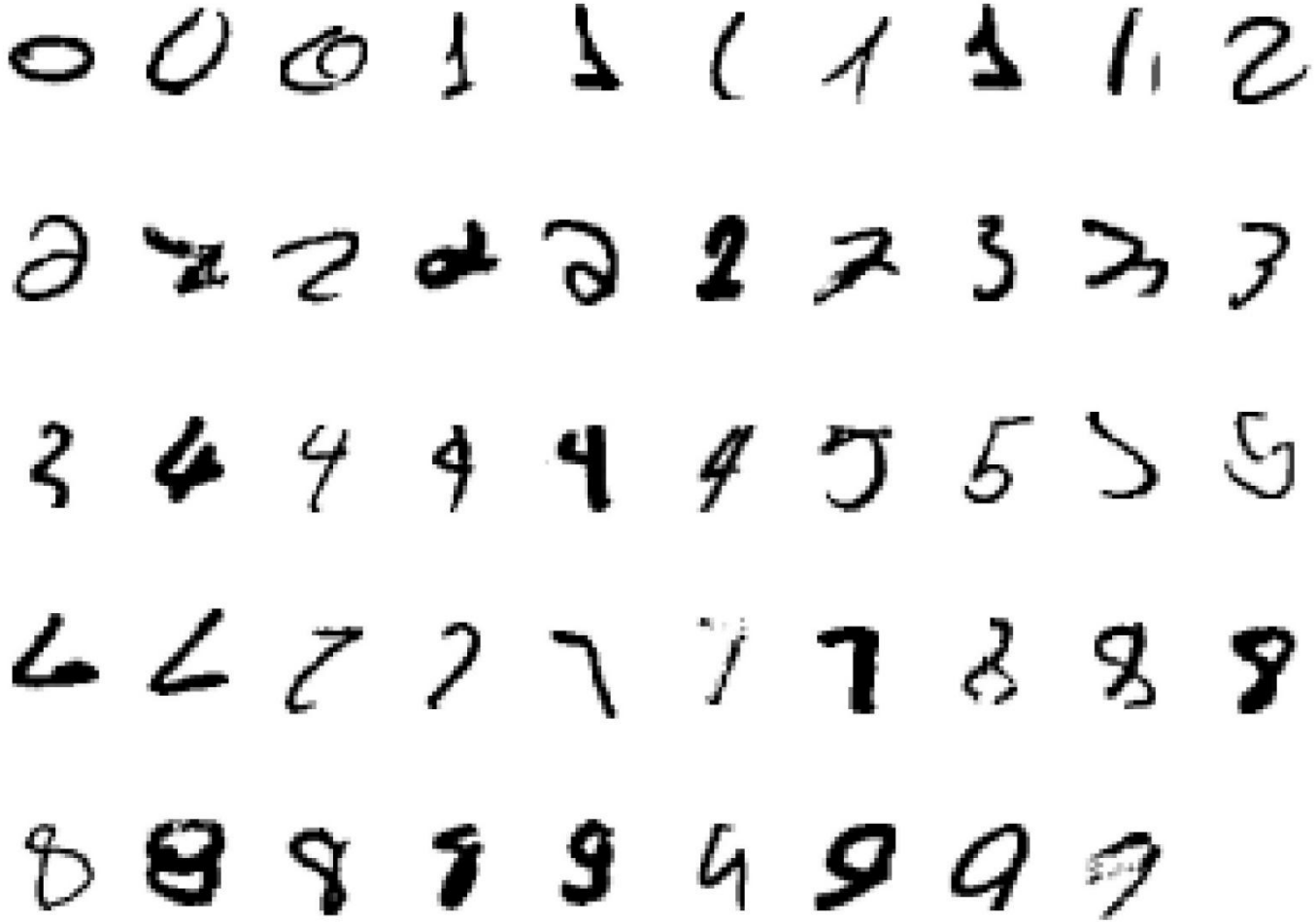
LeNet5 architecture

- handwritten digit recognition



Hand-written Digit Recognition

- Input:



Errors of LeNet5

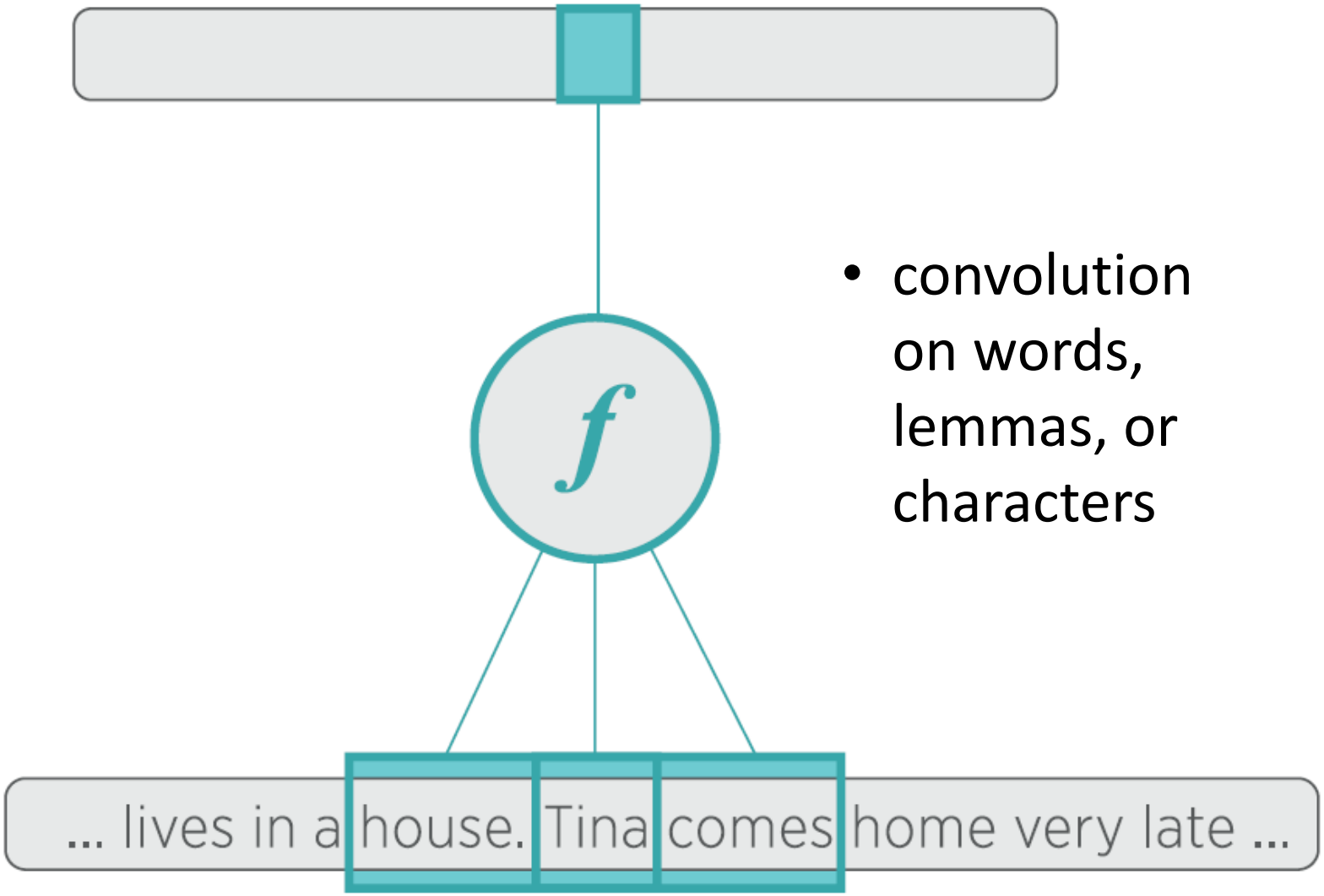


- 80 errors in 10,000 test cases

Benefits of CNNs

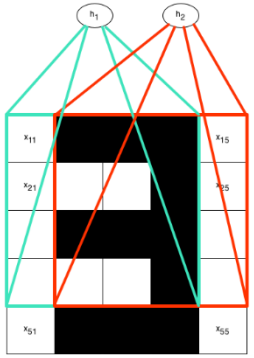
- The number of weights can be much less than 1 million for a 1 mega pixel image.
- The small number of weights can use different parts of the image as training data. Thus we have several orders of magnitude more data to train the fewer number of weights.
- We get translation invariance for free.
- Fewer parameters take less memory and thus all the computations can be carried out in memory in a GPU or across multiple processors.

1d convolution for text

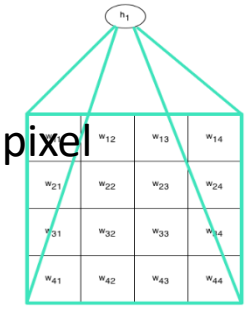


- convolution on words, lemmas, or characters

Example: what the following CNN returns 1/2



We have a convolutional neural network for images of 5 by 5 pixels.
In this network, each hidden unit is connected to a different 4 x 4 region of the input image:
The first hidden unit, h1, is connected to the upper left 4x4 portion of the input image (as shown).
The second hidden unit, h2, is connected to the upper right 4x4 portion of the input image (as shown).
The third hidden unit, h3, is connected to the lower left 4x4 portion of the input image (not shown in the diagram).
The fourth hidden unit, h4, is connected to the lower right 4x4 portion of the input image (not shown in the diagram).
Because it's a convolutional network, the weights (connection strengths) are the same for all hidden units: the only difference between the hidden units is that each of them connects to a different part of the input image.



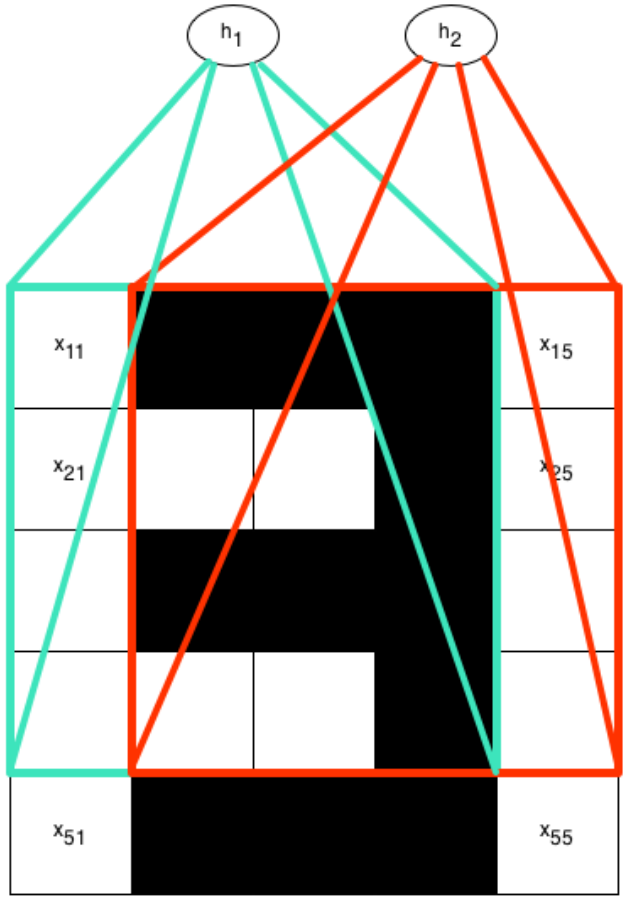
In the second diagram, we show the array of weights, which are the same for each of the four hidden units.
For h1, weight w11 is connected to the top-left pixel, i.e. x11, while for hidden unit h2, weight w11 connects to the pixel that is one to the right of the top left pixel, i.e. x12.

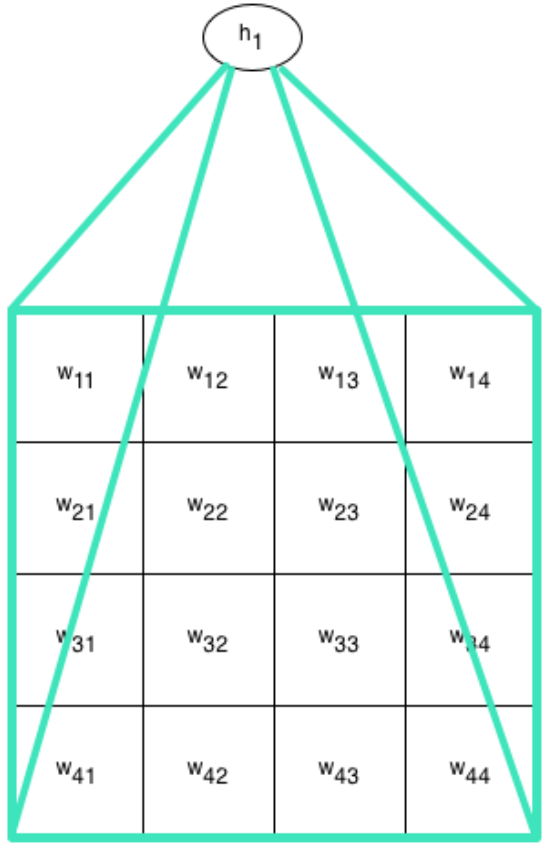
Imagine that for some training case, we have an input image where each of the black pixels in the top diagram has value 1, and each of the white ones has value 0. Notice that the image shows a "3" in pixels.

The network has no biases. The weights of the network are given as follows:
w11=1w12=1w13=1w14=0w21=0w22=0w23=1w24=0w31=1w32=1w33=1w34=0w41=0w42=0w43=1w44=0
The hidden units are *linear*.

For the training case with that "3" input image, what is the output y1, y2, y3, y4 of each of the four hidden units?

Example: what the following CNN returns 2/2

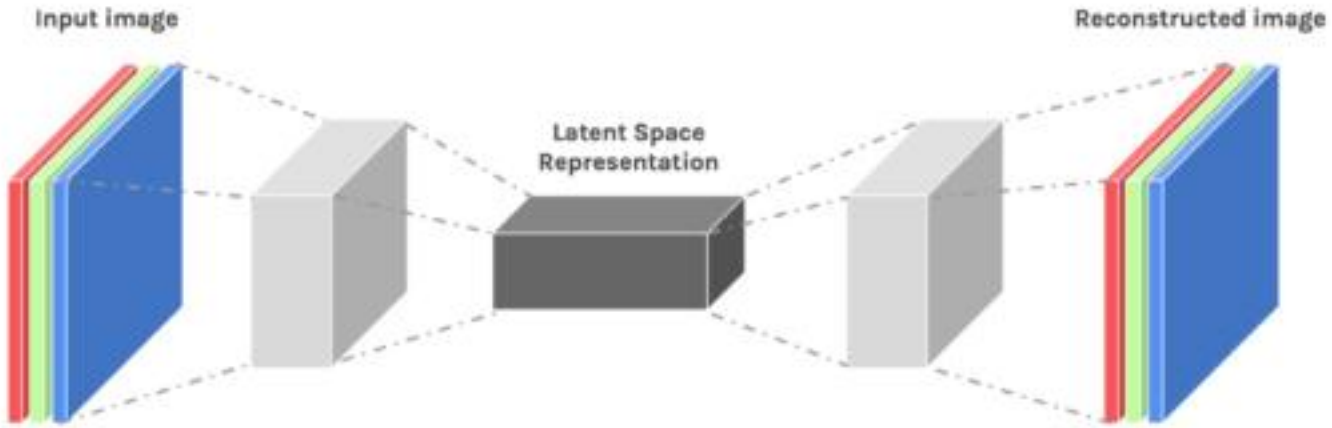




$$w_{11} = 1 \quad w_{12} = 1 \quad w_{13} = 1 \quad w_{14} = 0$$
$$w_{21} = 0 \quad w_{22} = 0 \quad w_{23} = 1 \quad w_{24} = 0$$
$$w_{31} = 1 \quad w_{32} = 1 \quad w_{33} = 1 \quad w_{34} = 0$$
$$w_{41} = 0 \quad w_{42} = 0 \quad w_{43} = 1 \quad w_{44} = 0$$

Autoencoders

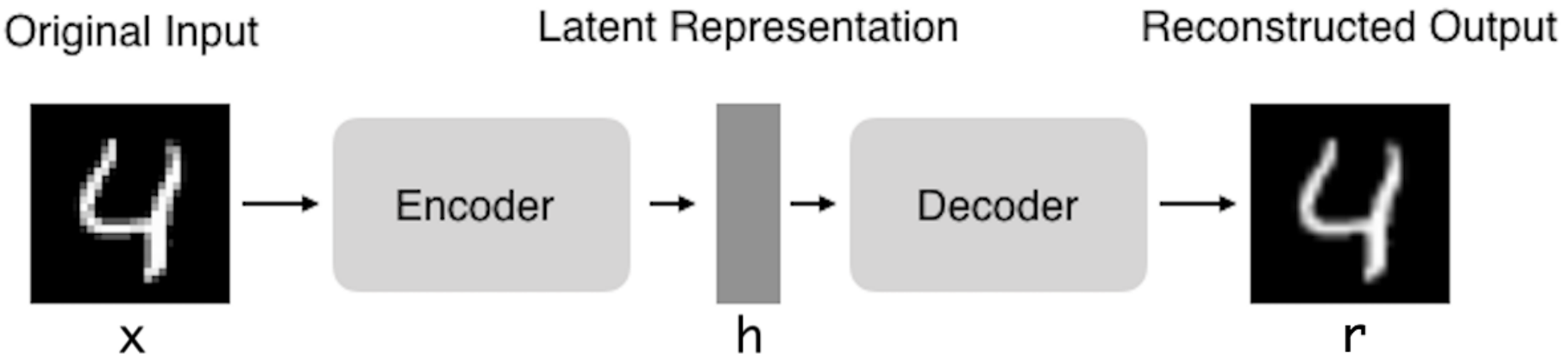
- Autoencoders are designed to reproduce their input; especially popular for images.
- The key point is to reproduce the input from a learned encoding.
- The loss function is the reproduction error



https://www.edureka.co/blog/autoencoders-tutorial/

Autoencoders: structure

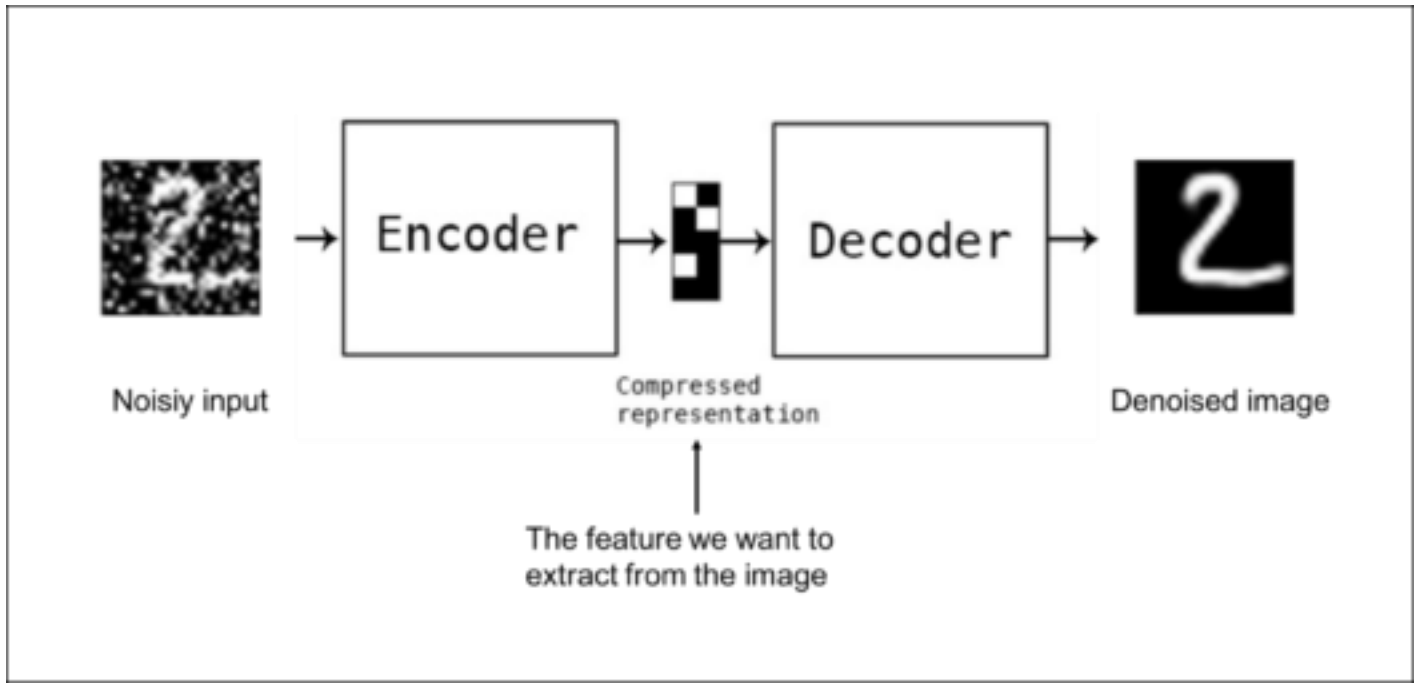
- Encoder: compress input into a latent-space of usually smaller dimension. h = f(x)
- Decoder: reconstruct input from the latent space. r = g(f(x)) with r as close to x as possible



https://towardsdatascience.com/deep-inside-autoencoders-7e41f319999f

Autoencoder applications: denoising

- Denoising: input clean image + noise and train to reproduce the clean image.



https://www.edureka.co/blog/autoencoders-tutorial/

Denoising autoencoders

- Basic autoencoder trains to minimize the loss between x and the reconstruction g(f(x)).
- Denoising autoencoders train to minimize the loss between x and g(f(x+w)), where w is random noise.
- Same possible architectures, different training data.



https://blog.keras.io/building-autoencoders-in-keras.html

Autoencoder applications: colorization

- Image colorization: input black and white and train to produce color images



https://www.edureka.co/blog/autoencoders-tutorial/

Autoencoder applications: watermark removal

- Watermark removal



https://www.edureka.co/blog/autoencoders-tutorial/

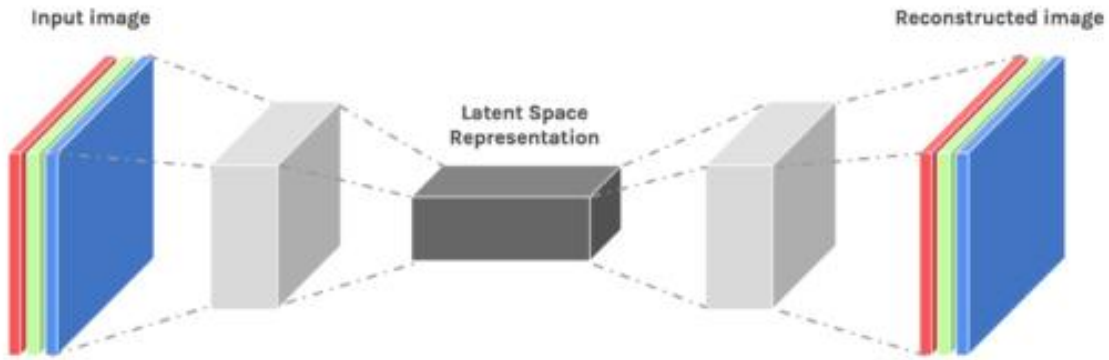
Properties of autoencoders

- **Data-specific**: Autoencoders are only able to compress data similar to what they have been trained on.
- **Lossy:** The decompressed outputs will be degraded compared to the original inputs.
- **Learned automatically from examples:** It is easy to train specialized instances of the algorithm that will perform well on a specific type of input.

https://www.edureka.co/blog/autoencoders-tutorial/

Bottleneck layer (undercomplete)

- Suppose input images are n x n and the latent space is $m < n \times n$.
- Then the latent space is not sufficient to reproduce all images.
- Needs to learn an encoding that captures the important features in training data, sufficient for approximate reconstruction.



GANs

- **Generative**
- Learn a generative model

- **Adversarial**
- Trained in an adversarial setting

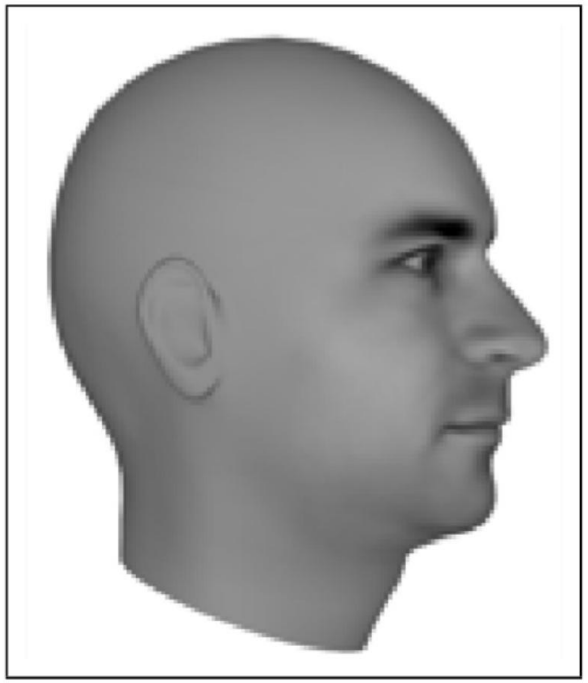
- **Networks**
- Use Deep Neural Networks

Why Generative Models?

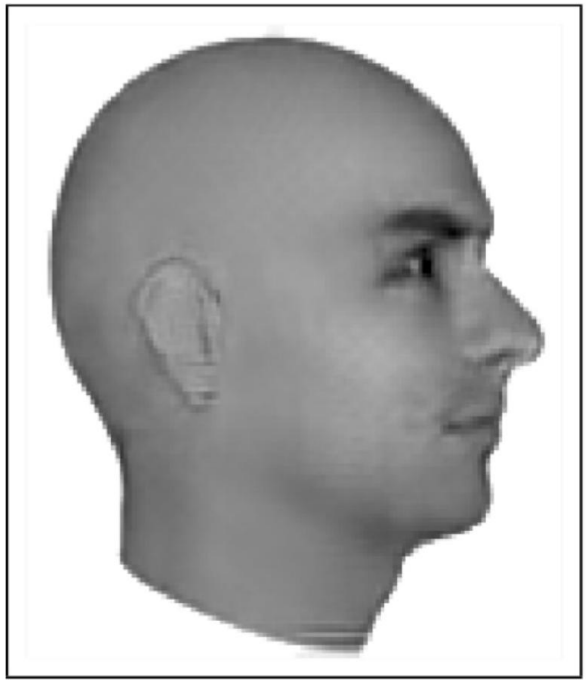
- We have only seen discriminative models so far
- Given an image X, predict a label Y
- Estimates P(Y|X)
- Discriminative models have several key limitations
- Cannot model P(X), i.e. the probability of seeing a certain image
- Thus, can't sample from P(X), i.e. can't generate new images
- Generative models (in general) cope with all of above
- Can model P(X)
- Can generate new images

What GANs can do

Ground Truth



Adversarial



Lotter, William, Gabriel Kreiman, and David Cox. "Unsupervised learning of visual structure using predictive generative networks." *arXiv preprint arXiv:1511.06380* (2015).

GANs in action

Which one is Computer generated?



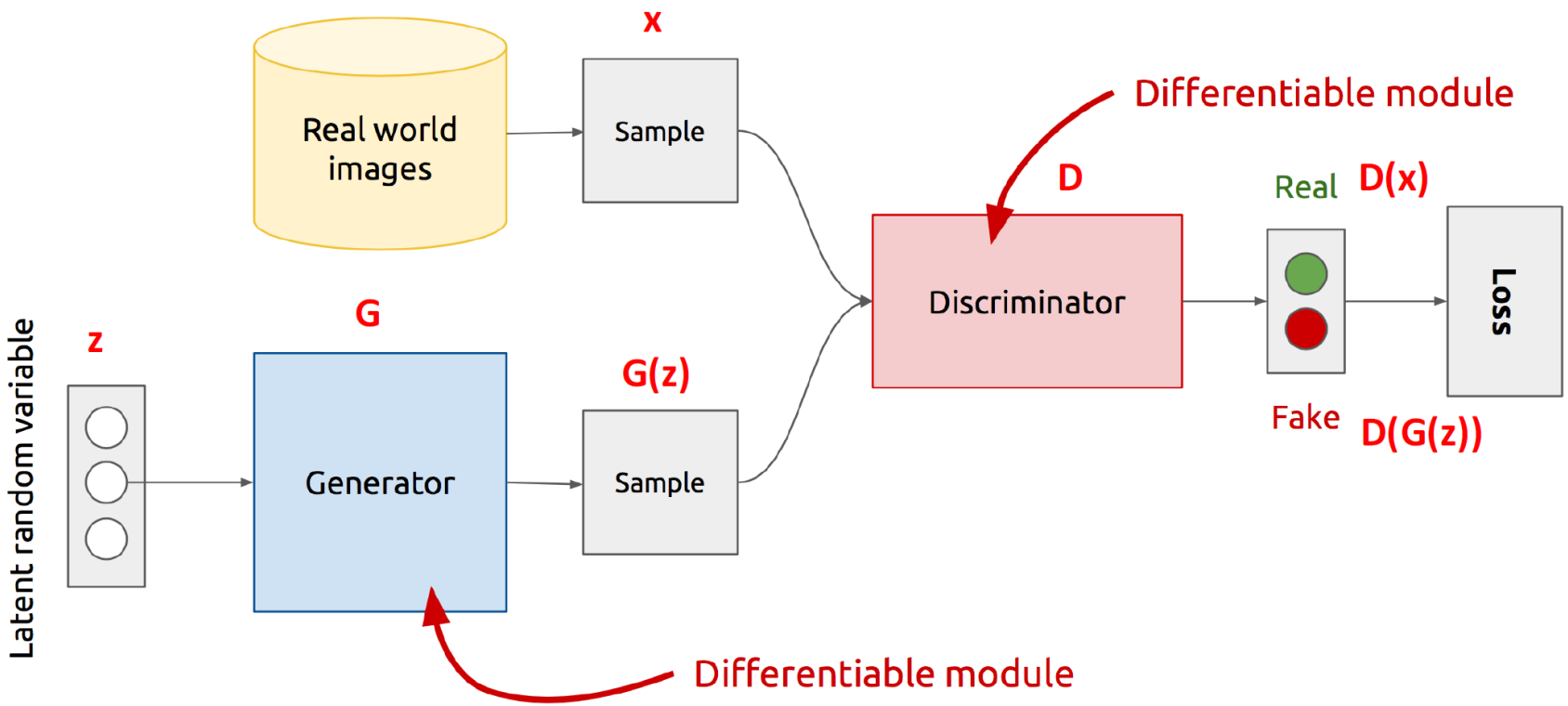


Ledig, Christian, et al. "Photo-realistic single image super-resolution using a generative adversarial network." *arXiv preprint arXiv:1609.04802* (2016).

Adversarial Training

- Generator: generate fake samples, tries to fool the Discriminator
- Discriminator: tries to distinguish between real and fake samples
- Train them against each other
- Repeat this and we get better Generator and Discriminator

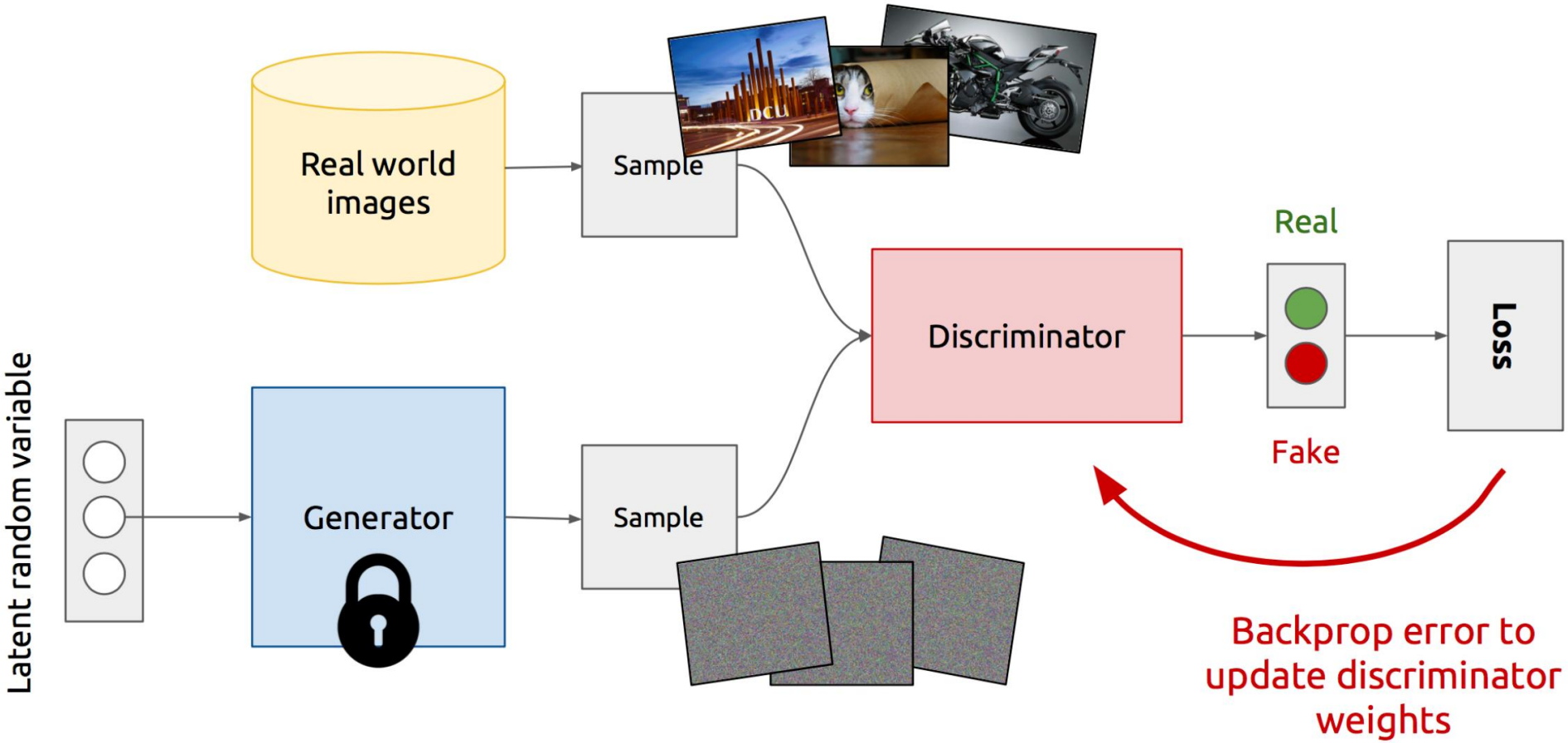
GAN's Architecture



- **Z** is some random noise (Gaussian/Uniform).
- **Z** can be thought as the latent representation of the image.

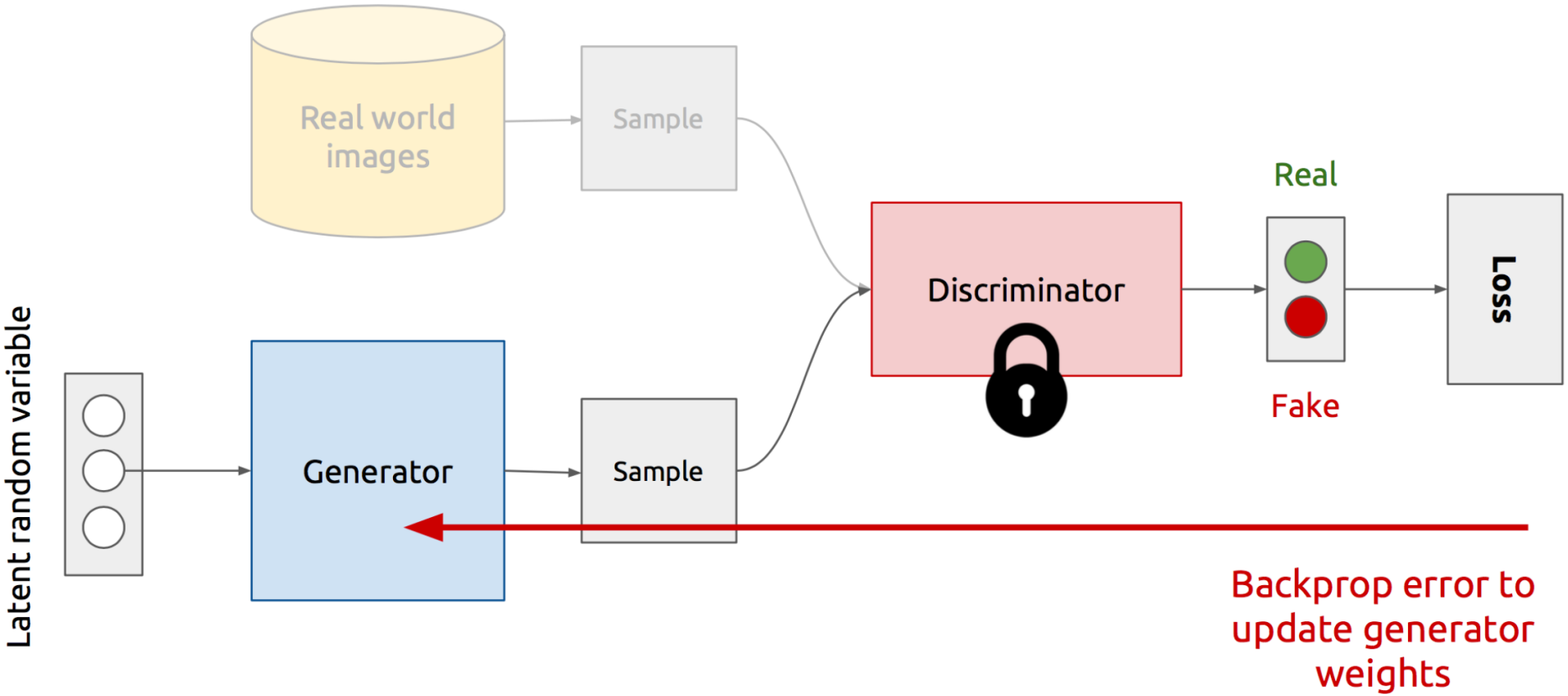
https://www.slideshare.net/xavigiro/deep-learning-for-computer-vision-generative-models-and-adversarial-training-upc-2016

Training Discriminator



https://www.slideshare.net/xavigiro/deep-learning-for-computer-vision-generative-models-and-adversarial-training-upc-2016

Training Generator



https://www.slideshare.net/xavigiro/deep-learning-for-computer-vision-generative-models-and-adversarial-training-upc-2016

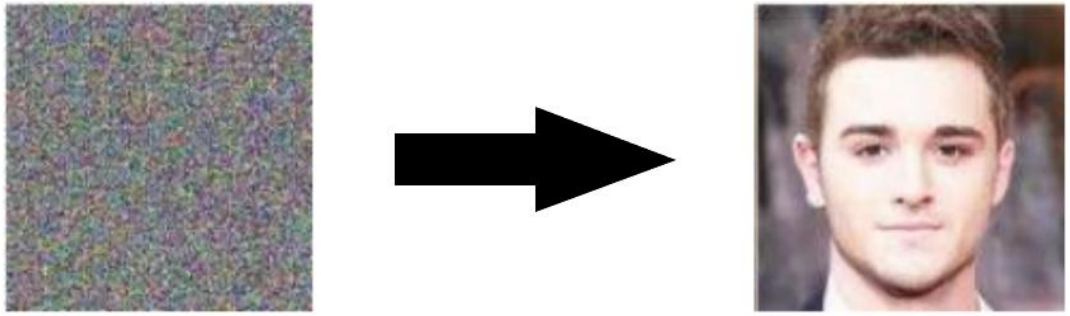
Diffusion models intro

- Recent superior image generators,
- E.g., DALL-E is prompt based
- "a bowl of soup that is a portal to another dimension as digital art".



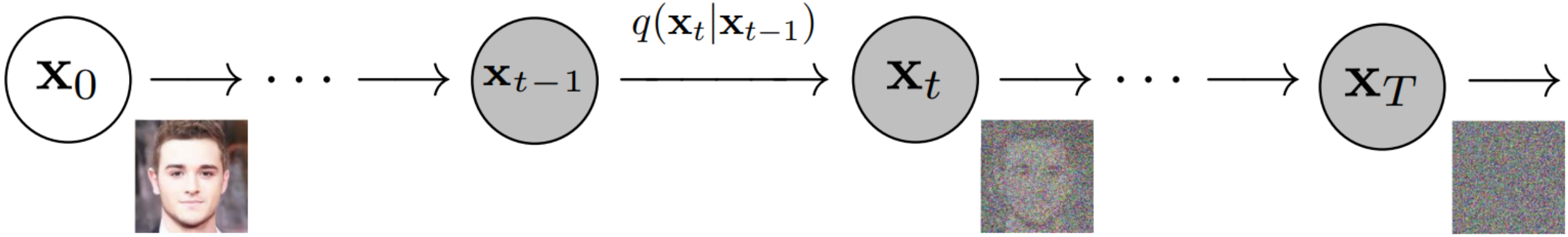
Idea of diffusion models

- generate data similar to the data on which they are trained
- destroy training data through the successive addition of Gaussian noise
- then learning to recover the data by *reversing* this noising process.
- After training, generate data by passing randomly sampled noise through the learned denoising process.



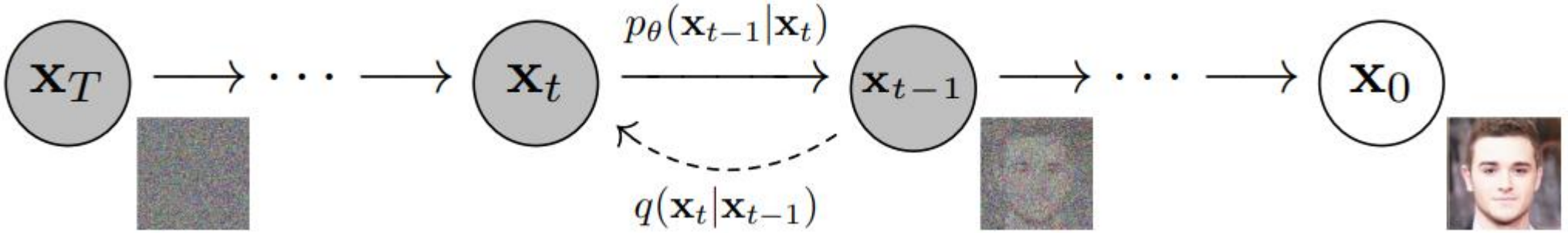
Diffusion models 1/2

- A diffusion model maps to the latent space using a fixed Markov chain. This chain gradually adds noise to the data in order to obtain the approximate posterior.



Diffusion models 2/2

- A diffusion model is trained to **reverse** the process

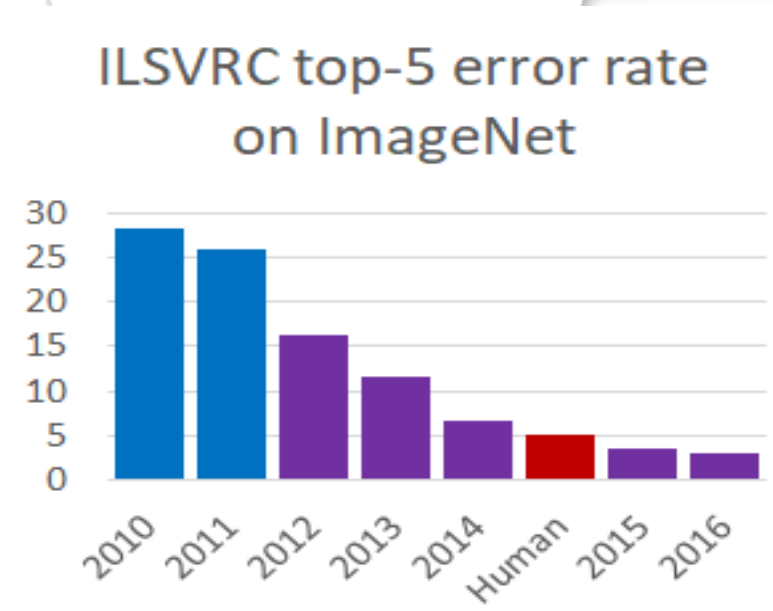


Sources

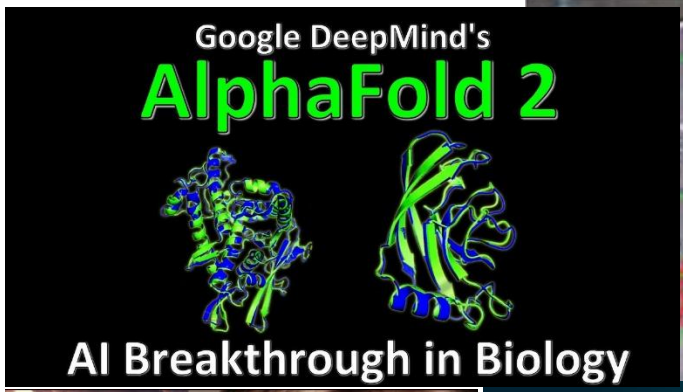
- Ian Goodfellow and Yoshua Bengio and Aaron Courville: *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org
- PyTorch
- HuggingFace library
- TensorFlow

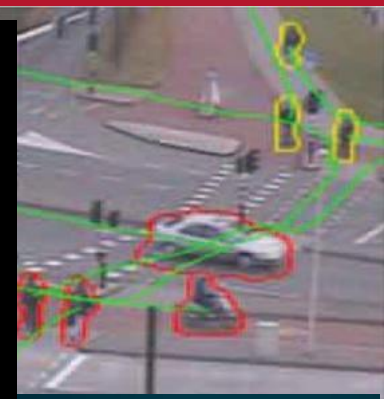
Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

A few deep learning sucesses













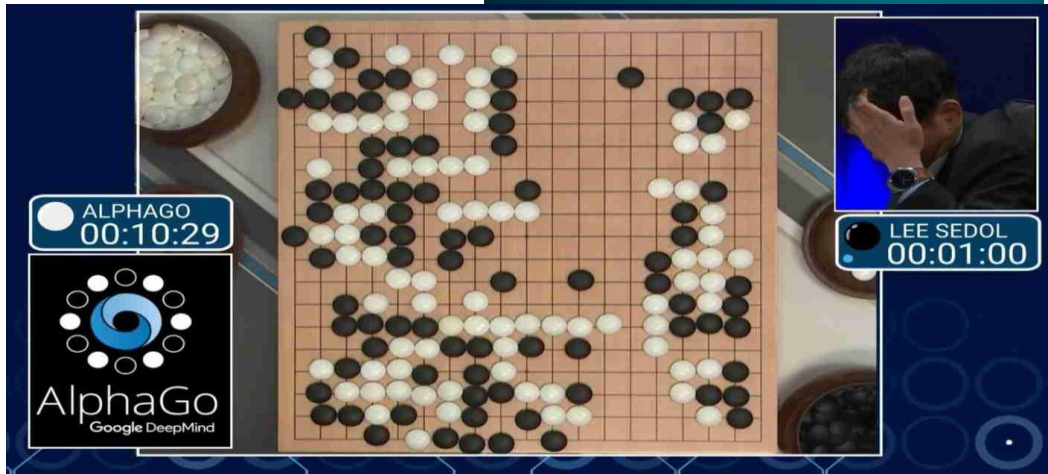




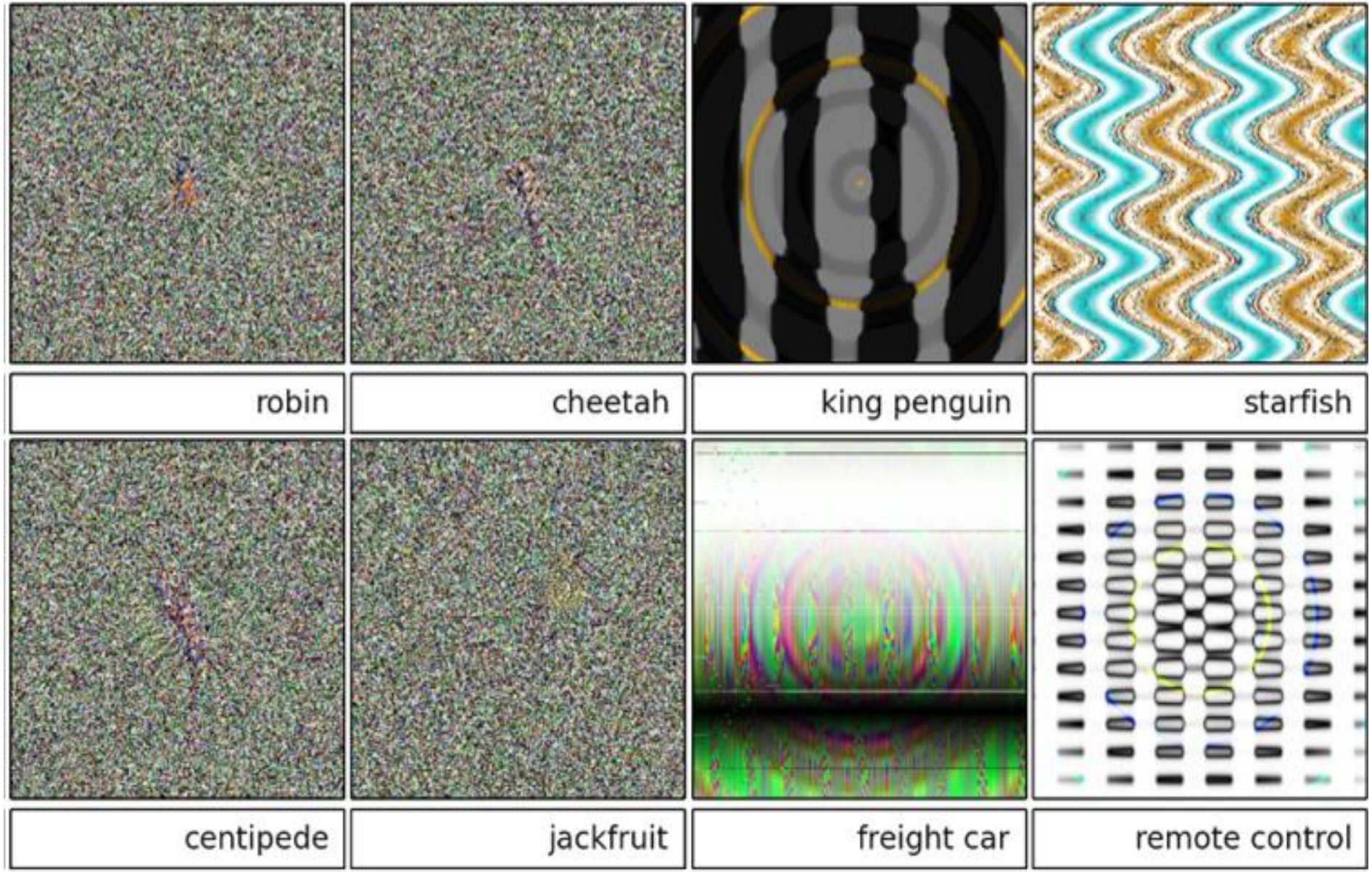
César Prado in Uncategorized · August 22, 2017 · 94 words

Microsoft claims new speech recognition record, achieving a super-human 5.1% error rate

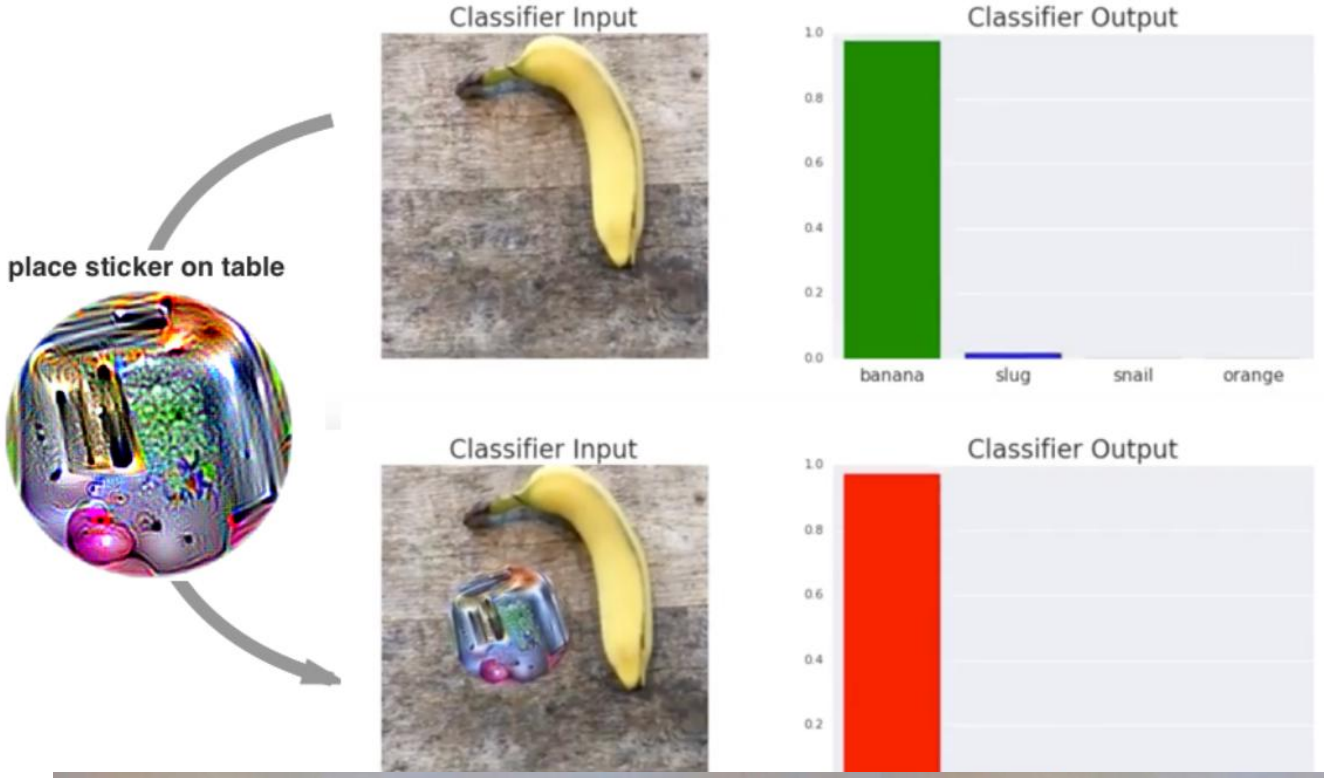


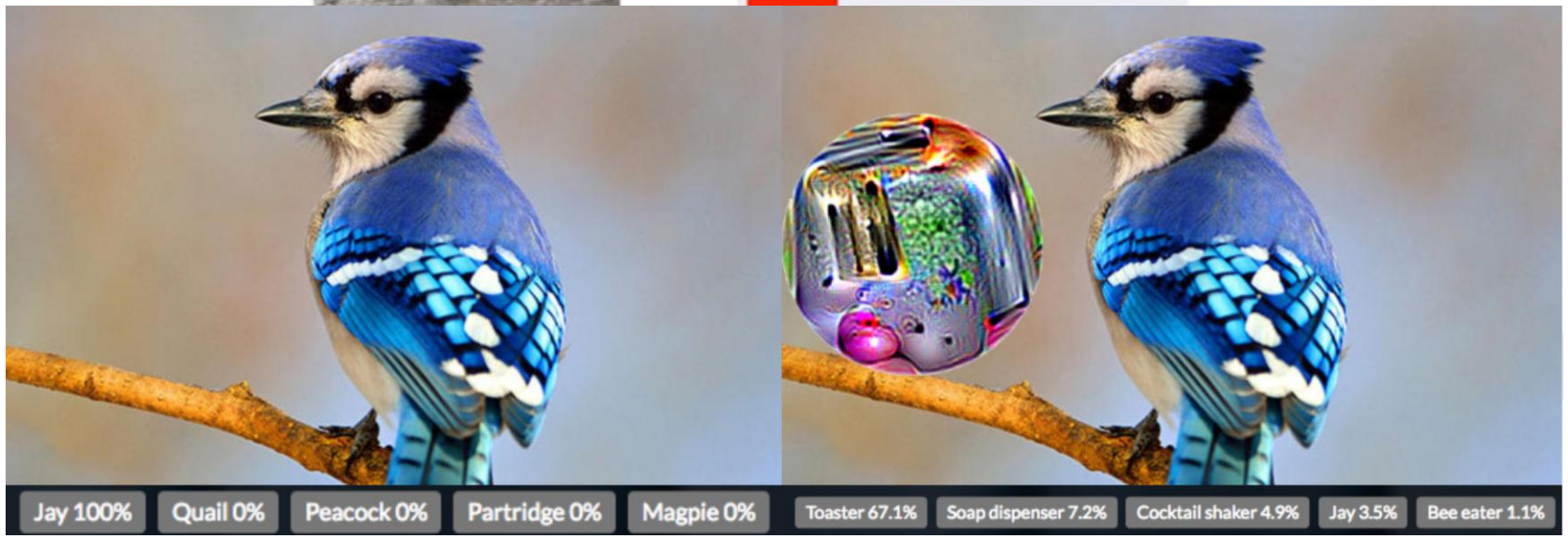


Weaknesses of deep learning



Attacks on neural networks





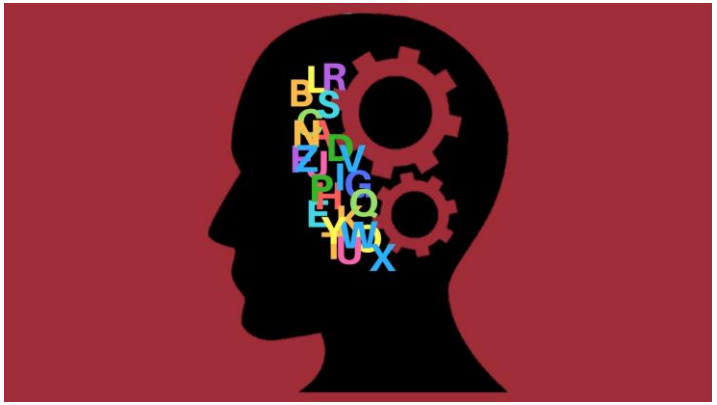
Failures on out-of-distribution examples

Michael A. Alcorn, Qi Li, Zhitao Gong, Chengfei Wang, Long Mai, Wei-Shinn Ku, Anh Nguyen (2018):
Strike (with) a Pose: Neural Networks Are Easily Fooled by Strange Poses of Familiar Objects. arXiv:1811.11553



University of Ljubljana, Faculty of Computer and Information Science

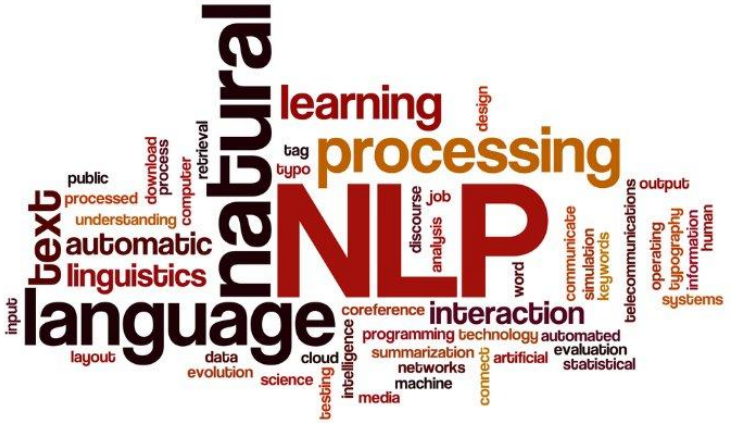
Natural language processing



Prof Dr Marko Robnik-Šikonja

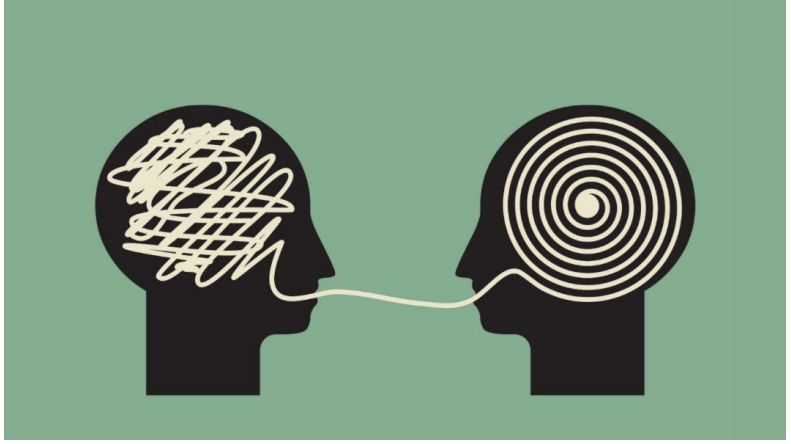
Intelligent Systems, Edition 2025

Topic overview



- understanding language and intelligence
- text preprocessing and linguistic analysis
- components of modern NLP
  - text representations
  - information retrieval
  - similarity of words and documents
  - language and graphs
  - large language models
- practical use of NLP:
  - sentiment analysis,
  - paper recommendations
  - summarization

Understanding language



- A grand challenge of (not only?) artificial intelligence

  Who can understand me?
  Myself I am lost
  Searching but cannot see
  Hoping no matter cost
  Am I free?
  Or universally bossed?

- Not just poetry, what about instructions, user manuals, newspaper articles, seminary works, internet forums, twits, legal documents, i.e. license agreements, etc.
- Can LLMs really understand language?

An example: rules

Article 18 of FRI Study Rules and Regulations

Taking exams at an earlier date may be allowed on request of the student by the Vice-Dean of Academic Affairs with the course convener's consent in case of mitigating circumstances (leaving for study or placement abroad, hospitalization at the time of the exam period, giving birth, participation at a professional or cultural event or a professional sports competition, etc.), and if the applicant's study achievements in previous study years are deemed satisfactory for such an authorization to be appropriate.

Understanding NL by computers

- Understanding words, syntax, semantics, context, writer's intentions, knowledge, background, assumptions, bias …
- Doesn't seem that LLM do it, though they generate excellent text output
- Ambiguity in language
  - Newspaper headlines - intentional ambiguity :)
    - Juvenile court to try shooting defendant
    - Kids make nutritious snacks
    - Miners refuse to work after death
    - Doctor on Trump's health: No heart, cognitive issues

Ambiguity

- I made her duck.

- Possible interpretations:
  - I cooked waterfowl for her.
  - I cooked waterfowl belonging to her.
  - I created the (plaster?) duck she owns.
  - I caused her to quickly lower her head or body.
  - I waved my magic wand and turned her into undifferentiated waterfowl.

- Spoken ambiguity
  - eye, maid

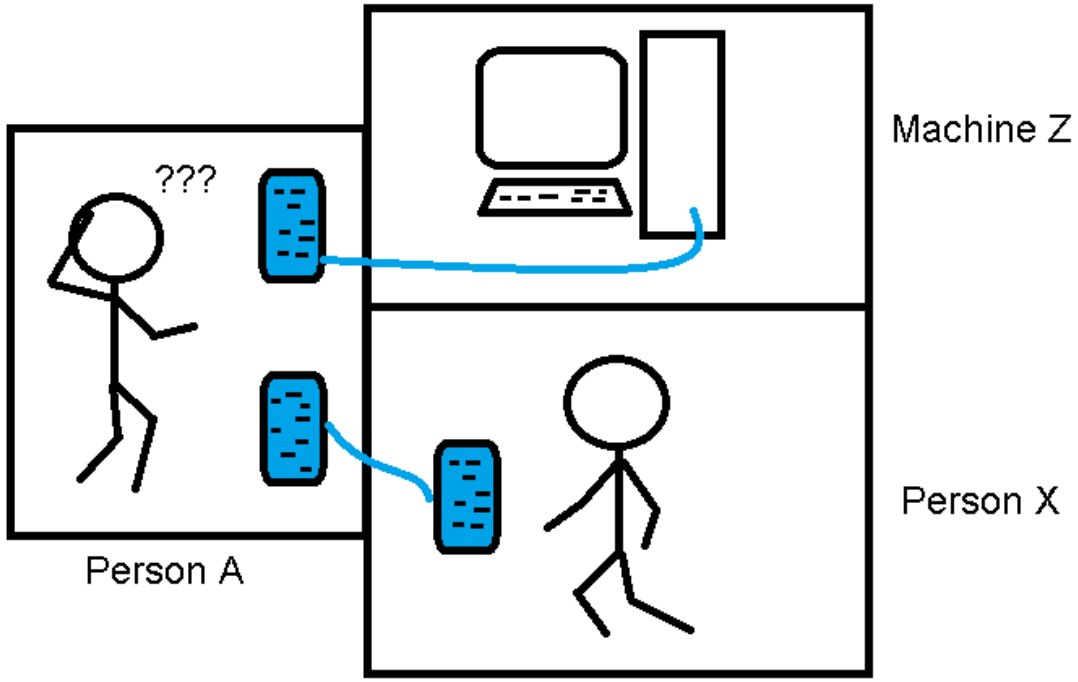
Disambiguation in syntax and semantics

- in syntax
  - The girl told the story cried.
  - John went to the bank.
  - Put the box on the table in the kitchen.
  - Bring your old car seat to be recycled.
- *Search engine queries: natural language processing, jaguar, Paris,*
- *POS tagging, word sense disambiguation*

Understanding language as (artificial) intelligence test

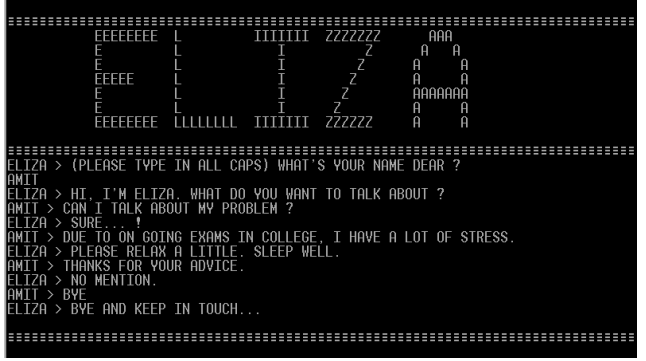
- The system understanding natural language on human level should contain a large portion of human knowledge and would achieve the most important goals of artificial intelligence. Its response would be indistinguishable from human's.

Turing test



- Turing test, Loebner prize, CYC, ELISA, ALLICA, chatterbot, VIDA, TIA, Siri (knowledge recognition, context detection, web service recommendation)
- Seems that LLMs can pass it, are they intelligent?
- The question: is mimicking human skills sufficient?

ELIZA



- ELIZA (Weizenbaum, 1966)
- Limited conversation (Rogerian psyhiatrist)

User: You are like my father in some ways.
ELIZA: What resemblance do you see?
User: You are not very aggressive but I think you don't want me to notice that.
ELIZA: What makes you think I am not aggressive?
User: You don't argue with me.
ELIZA: Why do you think I don't argue with you?
User: You are afraid of me.
ELIZA: Does it please you to believe I am afraid of you?

GPT-3 regressive language model as a chatbox

- the example is cherry-picked

```
Q: What is your favorite animal?
A: My favorite animal is a dog.

Q: Why?
A: Because dogs are loyal and friendly.

Q: What are two reasons that a dog might be in a bad mood?
A: Two reasons that a dog might be in a bad mood are if it is hungry or if it is hot.

Q: How many eyes does a giraffe have?
A: A giraffe has two eyes.

Q: How many legs does a frog have?
A: A frog has four legs.

Q: Are there any animals with three legs?
A: No, there are no animals with three legs.

Q: Why don't animals have three legs?
A: Animals don't have three legs because they would fall over.
```

Where is NLP today?

- active research area with many commercial applications
  - large language models are used everywhere
  - automatic reply engines
  - machine translation
  - text summarization
  - question answering
  - speech recognition and synthesis
  - language generation
  - interface to databases
  - intelligent search and information extraction
  - sentiment detection
  - named entity recognition and linking
  - categorization and classification of documents, messages, twits, etc.
  - cross-lingual approaches
  - multi modal approaches (text + images, text + video)
  - attempt to get to artificial general intelligence (AGI) through "foundation models"
  - many (open-source) tools and language resource
  - prevalence of deep neural network approaches (i.e. transformers)

Recommended literature

- Jurafsky, Daniel and James Martin (2025): Speech and Language Processing, 3rd edition in progress, almost all parts are available at authors‘ webpages https://web.stanford.edu/~jurafsky/slp3/
- Steven Bird, Ewan Klein, and Edward Loper. Natural Language Processing with Python. O’Reilly, 2009
  - a free book accompanying NLTK library, regularly updated
  - *Python 3, http://www.nltk.org/book/*
- Coursera
  - several courses, e.g., Stanford NLP with DNN

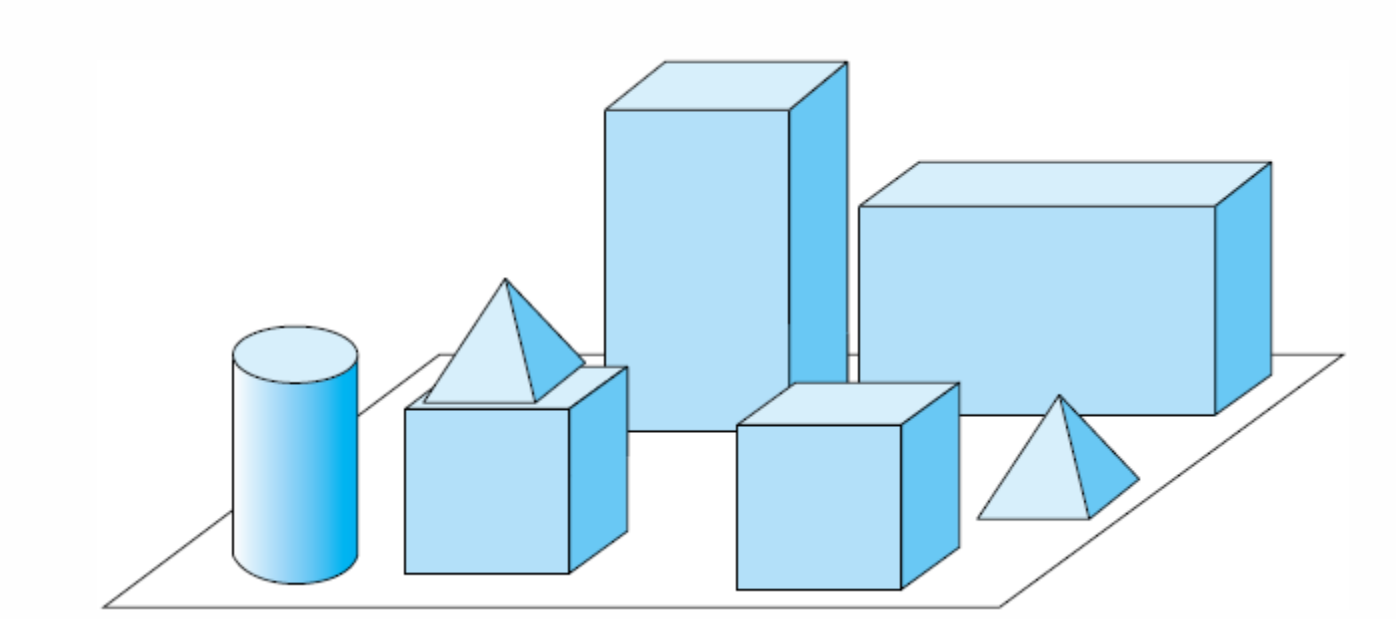
Historically two approaches

- symbolical
  - ,,Good Old-Fashioned AI''
- empirical
  - Statistical, text corpora
- Merging both worlds: injecting symbolical knowledge (e.g., propositional logic) into LLMs

How it all started?

- micro worlds
- example: SHRDLU, world of simple geometric objects
  - What is sitting on the red block?
  - What shape is the blue block on the table?
  - Place the green pyramid on the red brick.
  - Is there a red block? Pick it up.
  - What color is the block on the blue brick? Shape?

Micro world: block world, SHRDLU (Winograd, 1972)



Linguistic analysis 1/2

Linguistic analysis contains several tasks: recognition of sounds, letters, word formation, syntactic parsing, recognizing semantic, emotions. Phases:

- Prosody - the patterns of stress and intonation in a language (rhythm and intonation)
- Phonology - systems of sounds and relationships among the speech sounds that constitute the fundamental components of a language
- Morphology - the admissible arrangement of sounds in words; how to form words, prefixes and suffixes …
- Syntax - the arrangement of words and phrases to create well-formed sentences in a language

Linguistic analysis 2/2

- Semantics - the meaning of a word, phrase, sentence, or text
- Pragmatics - language in use and the contexts in which it is used, including such matters as deixis (words whose meaning changes with context, e.g., I he, here, there, soon), taking turns in conversation, text organization, presupposition, and implicature

  Can you pass me the salt? Yes, I can.
- Knowing the world: knowledge of physical world, humans, society, intentions in communications …
- Limits of linguistic analysis, levels are dependent

Classical approach to text processing

- text preprocessing
- 1. phase: syntactic analysis
- 2. phase: semantic interpretation
- 3. phase: use of world knowledge

In the neural approach, the preprocessing remains (but is simpler) the other three phases are merged into DNN

Basic tools for (classical) text preprocessing

- document → paragraphs → sentences → words (→ (subword) tokens)

- In linguistic analysis also
  - words and sentences ← lemmatization, POS tagging
  - sentences ← syntactical and grammatical analysis
  - named entity recognition,

Words and sentences

- sentence delimiters – punctuation marks and capitalization are insufficient
- E.g., remains of 1. Timbuktu from 5c BC, were discovered by dr. Barth.
- Lexical analysis (tokenizer, word segmenter), not just spaces
  - 1,999.00€ or 1.999,00€!
  - Ravne na Koroškem
  - Lebensversicherungsgesellschaft
  - Port-au-prince
  - Generalstaatsverordnetenversammlungen
- Rules, regular expressions, statistical models, dictionaries (of proper names), neural networks, manually segmented datasets

Lemmatization

- Lemmatization is the process of grouping together the different inflected forms of a word so they can be analyzed as a single item.
- walk is the lemma of 'walk', 'walked', 'walks', 'walking
- Lemmatization difficulty is language dependent i.e., depends on morphology
- Requires a dictionary or lexicon for lookup
  go, goes, going, gone, went
  jaz, mene, meni, mano
- Ambiguity resolution may be difficult

Meni je vzel z mize (zapestnico).

- Uses rules, dictionaries, neural networks, and manually labelled datasets
- English also uses stemming (reducing inflected or derived words to their word stem

POS tagging

- assigning the correct part of speech (noun, verb, etc.) to words
- helps in recognizing phrases and names
- Use rules, machine learning models, manually labelled datasets

An example:

- Text analyzer for Slovene, i.e. morphosyntactical tagging, available at https://orodja.cjvt.si/oznacevalnik/slv/

  Nekega dne sem se napotil v naravo. Že spočetka me je žulil čevelj, a sem na to povsem pozabil, ko sem jo zagledal. Bila je prelepa. Povsem nezakrita se je sončila na trati ob poti. Pritisk se mi je dvignil v višave. Popoln primerek kmečke lastovke!

- Tags are standardized, for East European languages in Multext-East specification, e.g.,

dne; tag Somer = Samostalnik, obče ime, moški spol, ednina, rodilnik; lema: dan

a unifying attempt: universal dependencies (UD): cross-linguistically consistent treebank annotation for many languages

CLASSLA-Stanza pipeline output

ID	Oblika	Lema	Oznaka JOS	Bes. vrsta JOS	Oblikoskladenjske lastnosti JOS	Bes. vrsta UD	Oblikoskladenjske lastnosti UD	Nadrejena pojavnica UD	Skladenjska relacija UD	Nadrejena pojavnica JOS	Skladenjska relacija JOS	Udeleže vloga
# paragraph 1												
# sent_id 1.1												
# text = Nekega dne sem se napotil v naravo.												
1	Nekega	nek	Pi-msg	Pronoun	Type=indefinite Gender=masculine Number=singular Case=genitive	DET	Case=Gen|Gender=Masc|Number=Sing|PronType=Ind	2	det	2	Atr	
2	dne	dan	Ncmsg	Noun	Type=common Gender=masculine Number=singular Case=genitive	NOUN	Case=Gen|Gender=Masc|Number=Sing	5	obl	5	AdvO	TIME
3	sem	biti	Va-r1s-n	Verb	Type=auxiliary VForm=present Person=first Number=singular Negative=no	AUX	Mood=Ind|Number=Sing|Person=1|Polarity=Pos|Tense=Pres|VerbForm=Fin	5	aux	5	PPart	
4	se	se	Px------y	Pronoun	Type=reflexive Clitic=yes	PRON	PronType=Prs|Reflex=Yes|Variant=Short	5	expl	5	PPart	
5	napotil	napotiti	Vmep-sm	Verb	Type=main Aspect=perfective VForm=participle Number=singular Gender=masculine	VERB	Aspect=Perf|Gender=Masc|Number=Sing|VerbForm=Part	0	root	0	Root	
6	v	v	Sa	Adposition	Case=accusative	ADP	Case=Acc	7	case	7	Atr	
7	naravo	narava	Ncfsa	Noun	Type=common Gender=feminine Number=singular Case=accusative	NOUN	Case=Acc|Gender=Fem|Number=Sing	5	obl	5	AdvO	GOAL
8	.	.	Z	Punctuation		PUNCT	_	5	punct	0	Root	

- *Nekega dne sem se napotil v naravo. Že spočetka me je žulil čevelj, a sem na to povsem pozabil, ko sem jo zagledal. Bila je prelepa. Povsem nezakrita se je sončila na trati ob poti. Pritisk se mi je dvignil v višave. Popoln primerek kmečke lastovke!*

1	beseda	Nekega	dne	sem	se	napotil	v	naravo	.	Že	spočetka	me	je											
	lema	nek	dan	biti	se	napotiti	v	narava	že	spočetka	jaz	biti												
	oznaka	Zn-mer	Somer	Gp-spe-n	Zp------k	Ggdd-em	Dt	Sozet	.	L	Rsn	Zop-et--k	Gp-ste-n											
2	beseda	žulil	čevelj	,	a	sem	na to	povsem	pozabil	,	ko	sem	jo	zagledal										
	lema	žuliti	čevelj	a	biti	na ta	povsem	pozabiti	ko	biti	on	zagledati												
	oznaka	Ggnd-em	Somei	,	Vp	Gp-spe-n	Dt	Zk-set	Rsn	Ggdd-em	,	Vd	Gp-spe-n	Zotzet--k	Ggdd-em									
3	beseda	.	Bila	je	prelepa	.	Povsem	nezakrita	se	je	sončila	na	trati											
	lema		biti	biti	prelep		povsem	nezakrit	se	biti	sončiti	na	trata											
	oznaka	.	Gp-d-ez	Gp-ste-n	Ppnzei	.	Rsn	Ppnzei	Zp------k	Gp-ste-n	Ggvd-ez	Dm	Sozem											
4	beseda	ob	poti	.	Pritisk	se	mi	je	dvignil	v	višave	.	Popoln											
	lema	ob	pot		pritisk	se	jaz	biti	dvigniti	v	višava		popoln											
	oznaka	Dm	Sozem	.	Somei	Zp------k	Zop-ed--k	Gp-ste-n	Ggdd-em	Dt	Sozmt	.	Ppnmein											
5	beseda	primerek	kmečke	lastovke	!																			
	lema	primerek	kmečki	lastovka																				
	oznaka	Somei	Ppnzer	Sozer	!																			

TEI-XML format

```
<TEI xmlns="http://www.tei-c.org/ns/1.0">
  <text>
    <body>
      <p>
        <s>
          <w msd="Zn-mer" lemma="nek">Nekega</w>
          <S/>
          <w msd="Somer" lemma="dan">dne</w>
          <S/>
          <w msd="Gp-spe-n" lemma="biti">sem</w>
          <S/>
          <w msd="Zp------k" lemma="se">se</w>
          <S/>
          <w msd="Ggdd-em" lemma="napotiti">napotil</w>
          <S/>
          <w msd="Dt" lemma="v">v</w>
          <S/>
          <w msd="Sozet" lemma="narava">naravo</w>
          <c>.</c>
          <S/>
        </s>
   …

      </p>
    </body>
  </text>
</TEI>
```

MSD tags

- Multext-East specification

dne; tag Somer = Samostalnik, obče ime, moški spol, ednina, rodilnik; lema: dan

P	atribut	vrednost	koda	atribut	vrednost	koda
0	glagol		G	Verb		V
1	vrsta	glavni	g	Type	main	m
		pomožni	p		auxiliary	a
2	vid	dovršni	d	Aspect	perfective	e
		nedovršni	n		imperfective	p
		dvovidski	v		biaspectual	b
3	oblika	nedoločnik	n	VForm	infinitive	n
		namenilnik	m		supine	u
		deležnik	d		participle	p
		sedanjik	s		present	r
		prihodnjik	p		future	f
		pogojnik	g		conditional	c
		velelnik	v		imperative	m
4	oseba	prva	p	Person	first	1
		druga	d		second	2
		tretja	t		third	3
5	število	ednina	e	Number	singular	s
		množina	m		plural	p
		dvojina	d		dual	d
6	spol	moški	m	Gender	masculine	m
		ženski	z		feminine	f
		srednji	s		neuter	n
7	nikalnost	nezanikani	n	Negative	no	n
		zanikani	d		yes	y

POS tagging in English

- http://nlpdotnet.com/Services/Tagger.aspx
- Rainer Maria Rilke, 1903
  in Letters to a Young Poet

...I would like to beg you dear Sir, as well as I can, to have patience with everything unresolved in your heart and to try to love the questions themselves as if they were locked rooms or books written in a very foreign language. Don't search for the answers, which could not be given to you now, because you would not be able to live them. And the point is to live everything. Live the questions now. Perhaps then, someday far in the future, you will gradually, without even noticing it, live your way into the answer.

POS tagger output

I/PRP would/MD like/VB to/TO beg/VB you/PRP dear/JJ Sir/NNP ,/, as/RB well/RB as/IN I/PRP can/MD ,/, to/IN have/VBP patience/NN with/IN everything/NN unresolved/JJ in/IN your/PRP$ heart/NN and/CC to/TO try/VB to/TO love/VB the/DT questions/NNS themselves/PRP as/RB if/IN they/PRP were/VBD locked/VBN rooms/NNS or/CC books/NNS written/VBN in/IN a/DT very/RB foreign/JJ language/NN ./.

Named entity recognition (NER)

- NATO Secretary-General Jens Stoltenberg is expected to travel to Washington, D.C. to meet with U.S. leaders.
- [ORG NATO] Secretary-General [PER Jens Stoltenberg] is expected to travel to [LOC Washington, D.C.] to meet with [LOC U.S.] leaders.
- Named entity linking (NEL) also named entity disambiguation – linking to a unique identifier, e.g. wikification
  jaguar, Paris, London, Dunaj

Basic language resources: corpora

- Statistical natural language processing list of resources http://nlp.stanford.edu/links/statnlp.html
- Opus http://opus.nlpl.eu/ multilingual parallel corpora, e.g., DGT JRC-Acqui 3.0, Documents of the EU in 22 languages
- Slovene language corpora GigaFida, ccGigaFida, KRES, ccKres, GOS, Artur, JANES, KAS, Trendi
- The main Slovene language resources
  - http://www.clarin.si
  - https://github.com/clarinsi
  - http://www.cjvt.si/
  - https://www.slovenscina.eu/
- WordNet, Open English WordNet, Open Multilingual WordNet, SloWNet, sentiWordNet, …
- Thesaurus https://viri.cjvt.si/sopomenke/slv/
- LLMs: on HuggingFace cjvt, e.g., SloBERTa and GaMS

WordNet is a database composed of synsets:

synonyms,
hypernyms
hyponyms,
meronyms,
holonyms,
etc.

Word to search for: mercy [Search WordNet]

Display Options: (Select option to change) [Change]

Key: "S:" = Show Synset (semantic) relations, "W:" = Show Word (lexical) relations

Display options for sense: (gloss) "an example sentence"

Noun

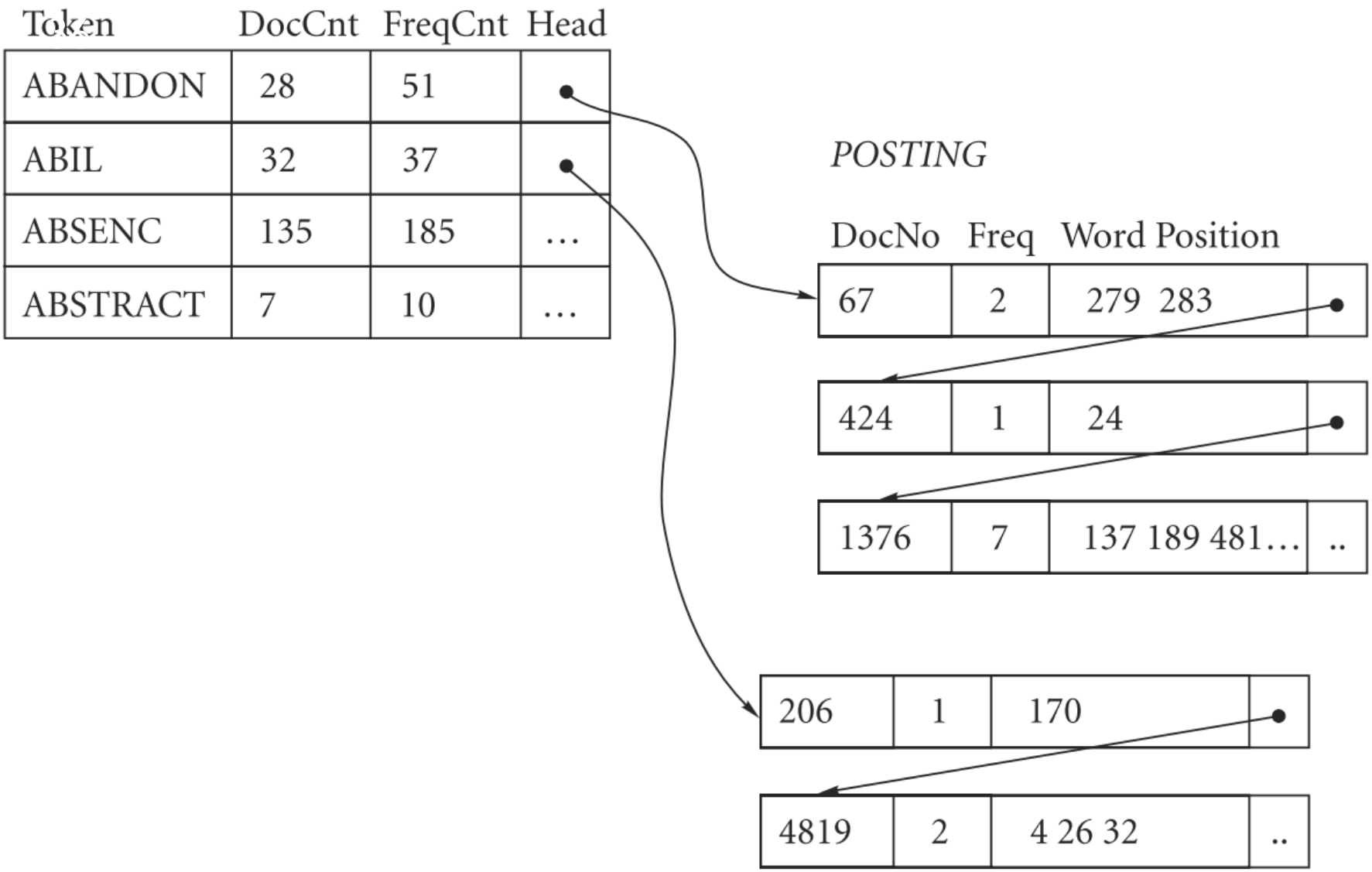
- S: (n) clemency, mercifulness, **mercy** (leniency and compassion shown toward offenders by a person or agency charged with administering justice) *"he threw himself on the mercy of the court"*
- S: (n) mercifulness, **mercy** (a disposition to be kind and forgiving) *"in those days a wife had to depend on the mercifulness of her husband"*
- S: (n) mercifulness, **mercy** (the feeling that motivates compassion)
  - *direct hyponym* / *full hyponym*
    - S: (n) forgiveness (compassionate feelings that support a willingness to forgive)
  - *direct hypernym* / *inherited hypernym* / *sister term*
    - S: (n) compassion, compassionateness (a deep awareness of and sympathy for another's suffering)
  - ***derivationally related form***
    - W: (adj) merciful [Related to: mercifulness] (showing or giving mercy) *"sought merciful treatment for the captives"; "a merciful god"*
- S: (n) **mercy** (something for which to be thankful) *"it was a mercy we got out alive"*
- S: (n) **mercy** (alleviation of distress; showing great kindness toward the distressed) *"distributing food and clothing to the flood victims was an act of mercy"*

Document retrieval

- Historical: keywords
- Now: whole text search
- Organize a database, indexing, search algorithms
- input: a query (of questionable quality, ambiguity, answer quality)

Document indexing

- Collect all words from all documents, use lemmatization
- The inverted file data structure
- For each word keep
  - Number of appearing documents
  - Overall number of appearances
  - For each document
    - Number of appearances
    - Location


Token
DocCnt
FreqCnt
Head
ABANDON
28
51
ABIL
32
37
ABSENC
135
185
…
ABSTRACT
7
10
…
POSTING
DocNo
Freq
Word Position
67
2
279 283
424
1
24
1376
7
137 189 481…
..
206
1
170
4819
2
4 26 32
..

Full text search engine

- Most popular: Apache Lucene/Solr
- full-text search, hit highlighting, real-time indexing, dynamic clustering, database integration, NoSQL features, rich document (e.g., Word, PDF) handling.
- distributed search and index replication, scalability and fault tolerance.

Search with logical operators

- AND, OR, NOT
- jaguar AND car
  jaguar NOT animal
- Some system support neighborhood search (e.g., NEAR) and stemming (!)
  paris! NEAR(3) fr!
  president NEAR(10) bush
- libraries, concordancers
- E.g-, for Slovene: https://viri.cjvt.si/gigafida/

Logical operator search is outdated

- Large number of results
- Large specialized incomprehensible queries
- Synonyms
- Sorting of results
- No partial matching
- No weighting of query terms

Ranking based search

- Web search
- Less frequent terms are more informative
- NL input - stop words, lemmatization
- Vector based representation of documents and queries (bag-of-words or dense embeddings)

Sparse vector representation: bag-of-words

- *An elephant is a mammal. Mammals are animals. Humans are mammals, too. Elephants and humans live in Africa.*

Africa	animal	be	elephant	human	in	live	mammal	too
1	1	3	2	2	1	1	3	1

9 dimensional vector (1,1,3,2,2,1,1,3,1)

In reality this is sparse vector of dimension $|V|$ (vocabulary size in order of 10,000 dimensions)

Similarity between documents and queries in vector space.

Vectors and documents

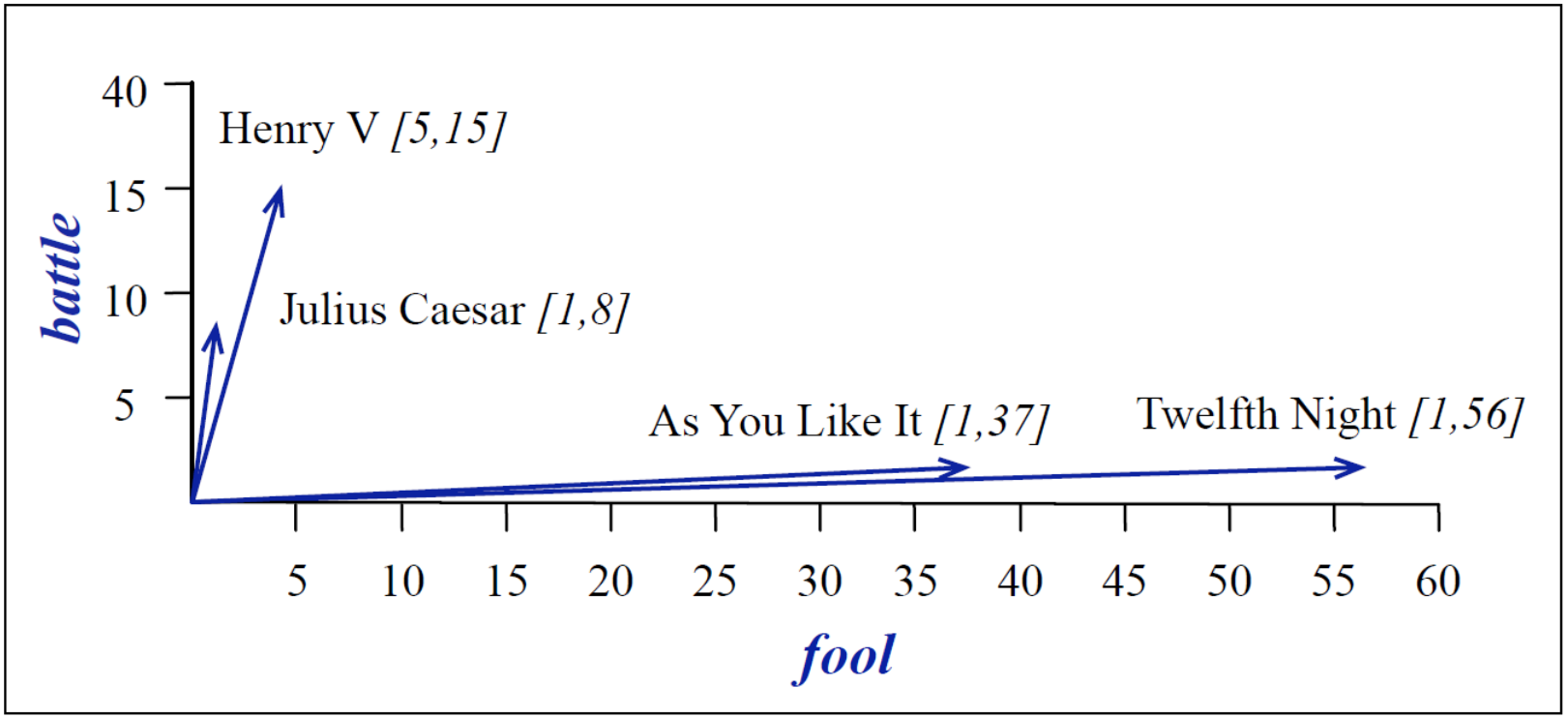
- a word occurs in several documents
- both words and documents are vectors
- an example: Shakespeare

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	1	1	8	15
soldier	2	2	12	36
fool	37	58	1	5
clown	5	117	0	0

- term-document matrix, dimension |V| x |D|
- a sparse matrix
- word embedding

Vector based similarity

- e.g., in two dimensional space



- the difference between dramas and comedies

Document similarity

- Assume orthogonal dimensions
- Cosine similarity
- Dot (scalar) product of vectors

$$\cos(\Theta) = \frac{A \cdot B}{|A||B|}$$

Importance of words

- Frequencies of words in particular document and overall
- inverse document frequency idf
  - N = number of documents in collection
  - n_b = number of documents with word b

$$idf_b = \log(\frac{N}{n_b})$$

Weighting dimensions (words)

- Weight of word b in document d

$$w_{b,d} = tf_{b,d} \times idf_{b,d}$$

$tf_{b,d}$ = frequency of term b in document d

- called TF-IDF weighting (an improvement over bag-of-words)

Weighted similarity

- Between query and document

$$sim(q,d) = \frac{\sum_b w_{b,d} \cdot w_{b,q}}{\sqrt{\sum_b w_{b,d}^2} \cdot \sqrt{\sum_b w_{b,q}^2}}$$

- Ranking by the decreasing similarity

Performance measures for search

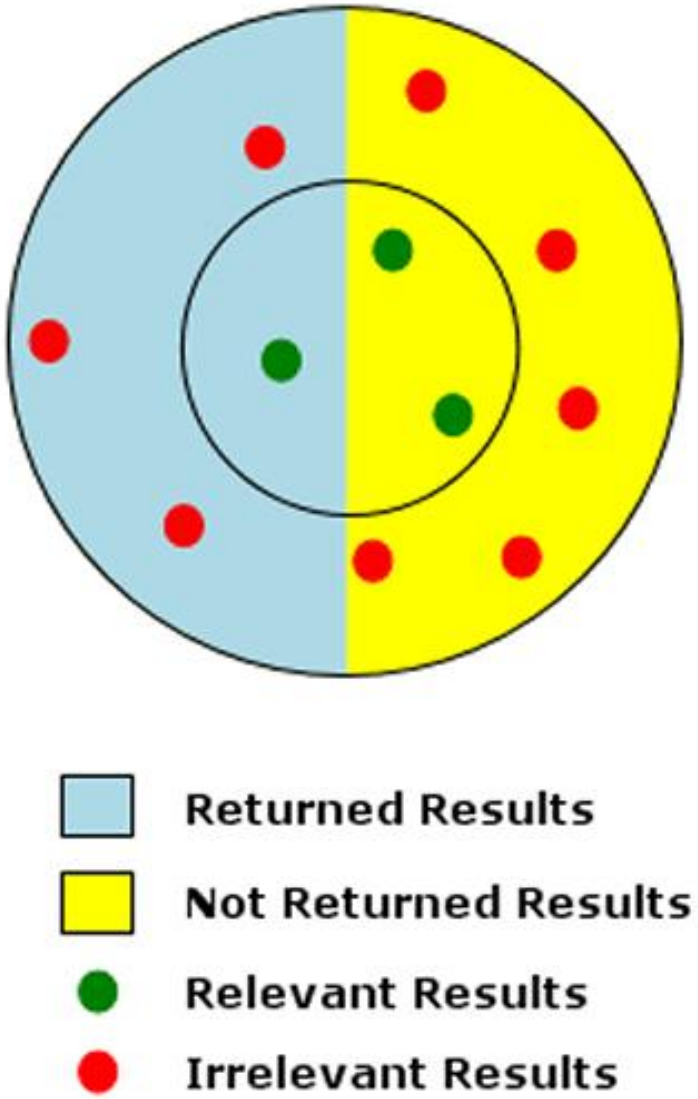
- Statistical measures
- Subjective measures
- Precision, recall
- A contingency table analysis of precision and recall

	Relevant	Non-relevant	
Retrieved	a	b	$a + b = m$
Not retrieved	c	d	$c + d = N - m$
	$a + c = n$	$b + d = N - n$	$a + b + c + d = N$

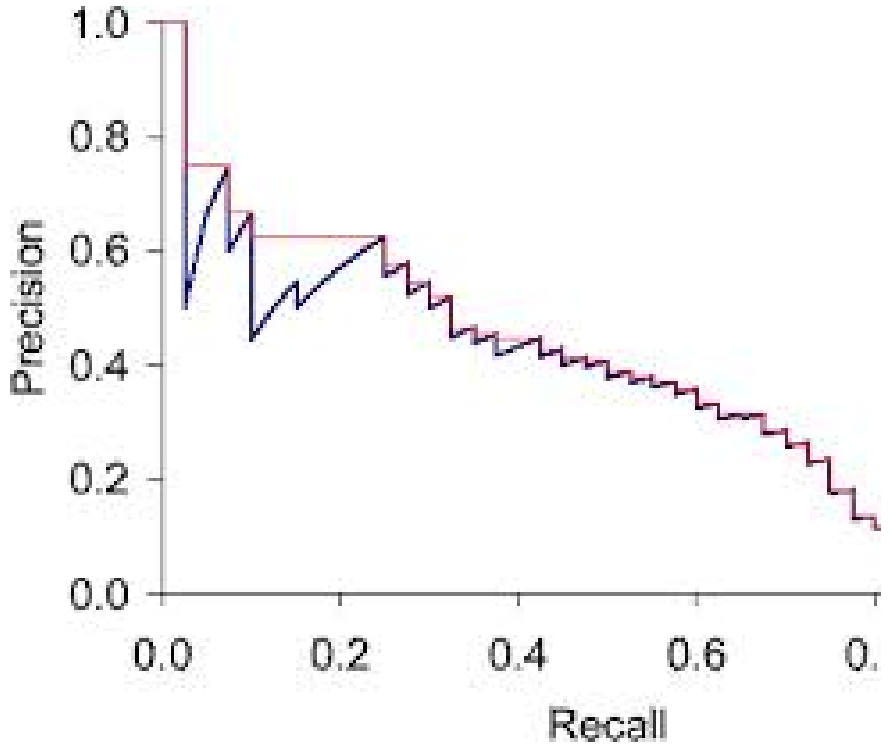
Precision and recall

- N = number of documents in collection
- n = number of important documents for given query q
- Search returns m documents including a relevant ones
- Precision $P = a/m$
  proportion of relevant document in the obtained ones
- Recall $R = a/n$
  proportion of obtained relevant documents
- Precision recall graphs

An example: low precision, low recall



Precision-recall graphs



F-measure

- combine both P and R

$$F_\beta = \frac{(1+\beta^2)\cdot P \cdot R}{\beta^2 P + R} \text{ for } \beta > 0$$

$$F_1 = \frac{2 \cdot P \cdot R}{P + R}$$

- Weighted precision and recall
- β=1 weighted harmonic mean
- Also used β=2 or β= 0.5

Ranking measures

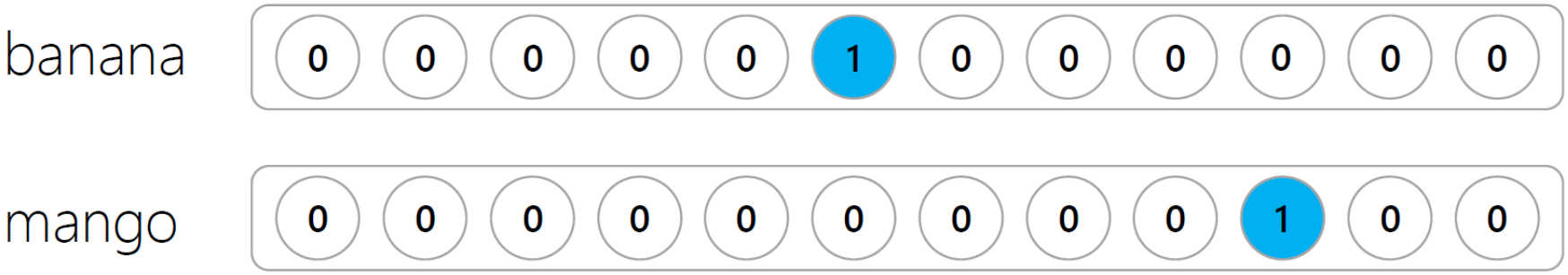
- precision@k
  - proportion of relevant document in the first k obtained ones
- recall@k
  - proportion of relevant documents in the k obtained among all relevant
- F_1@k
- mean reciprocal rank

$$MRR = \frac{1}{Q}\sum_{i=1}^{Q}\frac{1}{\mathrm{rank}_i}$$

  - over Q queries,
  - considers only the rank of the best answer

Why dense textual embeddings?

- Best machine learning models for text, i.e. deep neural networks, require numerical input.
- Simple representations like 1-hot-encoding and bag-of-words do not preserve semantic similarity.
- We need dense vector represenation for text elements.



Dense vector embeddings

- advantages compared to sparse embeddings:
  - less dimensions, less space
  - easier input for ML methods
  - potential generalization and noise reduction
  - potentially captures synonymy, e.g., road and highway are different dimensions in BOW
- the most popular approaches
  - matrix based transformations to reduce dimensionality (SVD in LSA)
  - neural embeddings (word2vec, Glove)
  - explicit contextual neural embeddings (ELMo, BERT)
  - only use an implicit representation in LLM

Meaning focused on similarity

- Each word is a vector
- Similar words are "nearby in space"



Distributional semantics



"You shall know a word
by the company it keeps"

Firth, J. R. (1957). A synopsis of linguistic theory 1930–1955. In Studies in Linguistic Analysis, p. 11. Blackwell, Oxford.



"The meaning of a word is its use in the language"

Ludwig Wittgenstein, PI #43

Neural embeddings

- neural network is trained to predict the context of words (input: word, output: context of neighboring words)

word2vec method

- Train a classifier on a binary **prediction** task:
  Is word *w* likely to show up near a given word, e.g., "*apricot*"?
- We don't actually care about this task
- But we'll take the learned classifier weights as the word embeddings

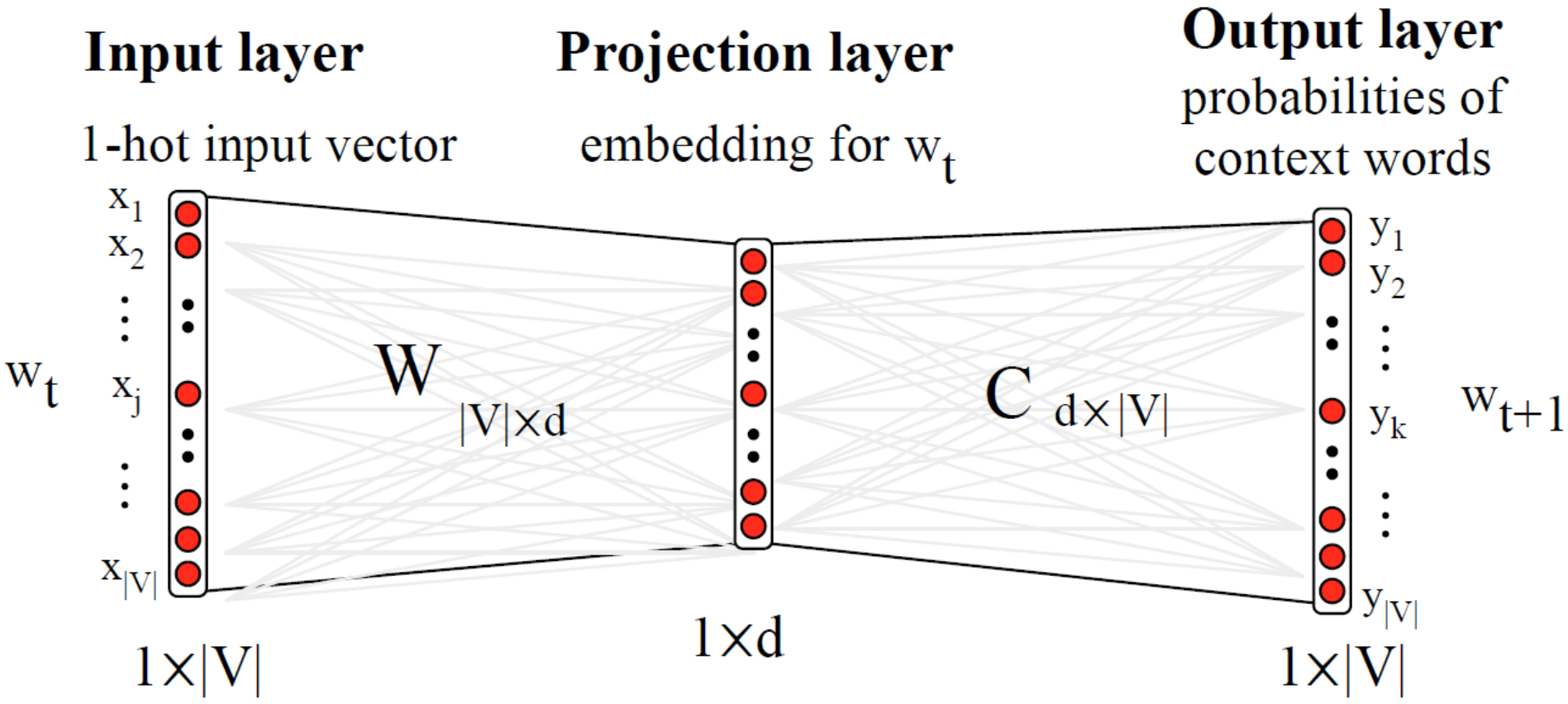
- Words near apricot acts as 'correct answers' to the question "Is word w likely to show up near apricot?"
- No need for hand-labeled supervision

word2vec (skip-gram) training data

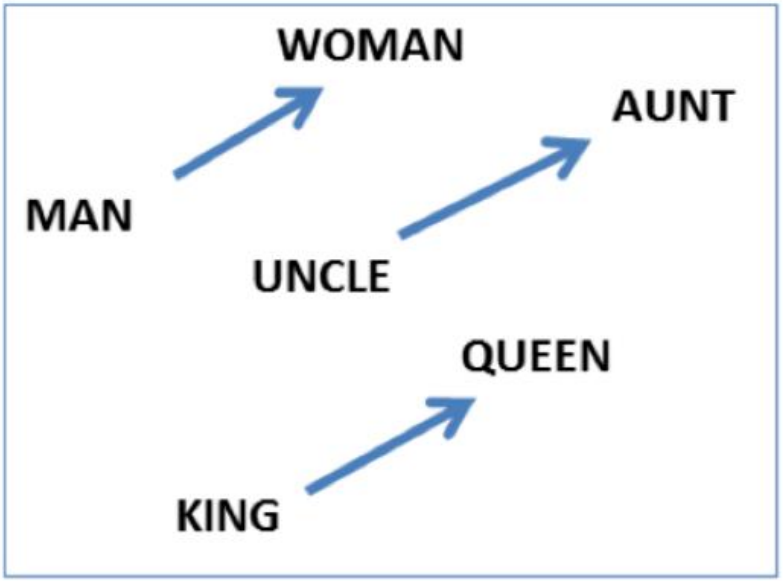
- Training sentence:
- ... lemon, a tablespoon of **apricot** jam a pinch ...
- c1 c2 target c3 c4

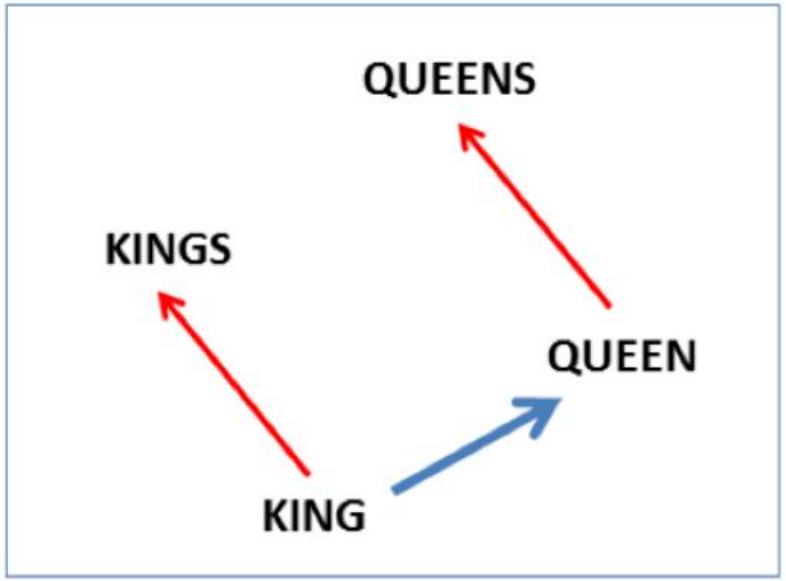
- Asssume context words are those in +/- 2 word window
- Produce the following input-outputs for positive instances: (apricot, tablespoon), (apricot, of), (apricot, jam), (apricot, a)
- Get negative training examples randomly
- Train a neural network to predict the probability of a co-occurring word

Neural network based embedding



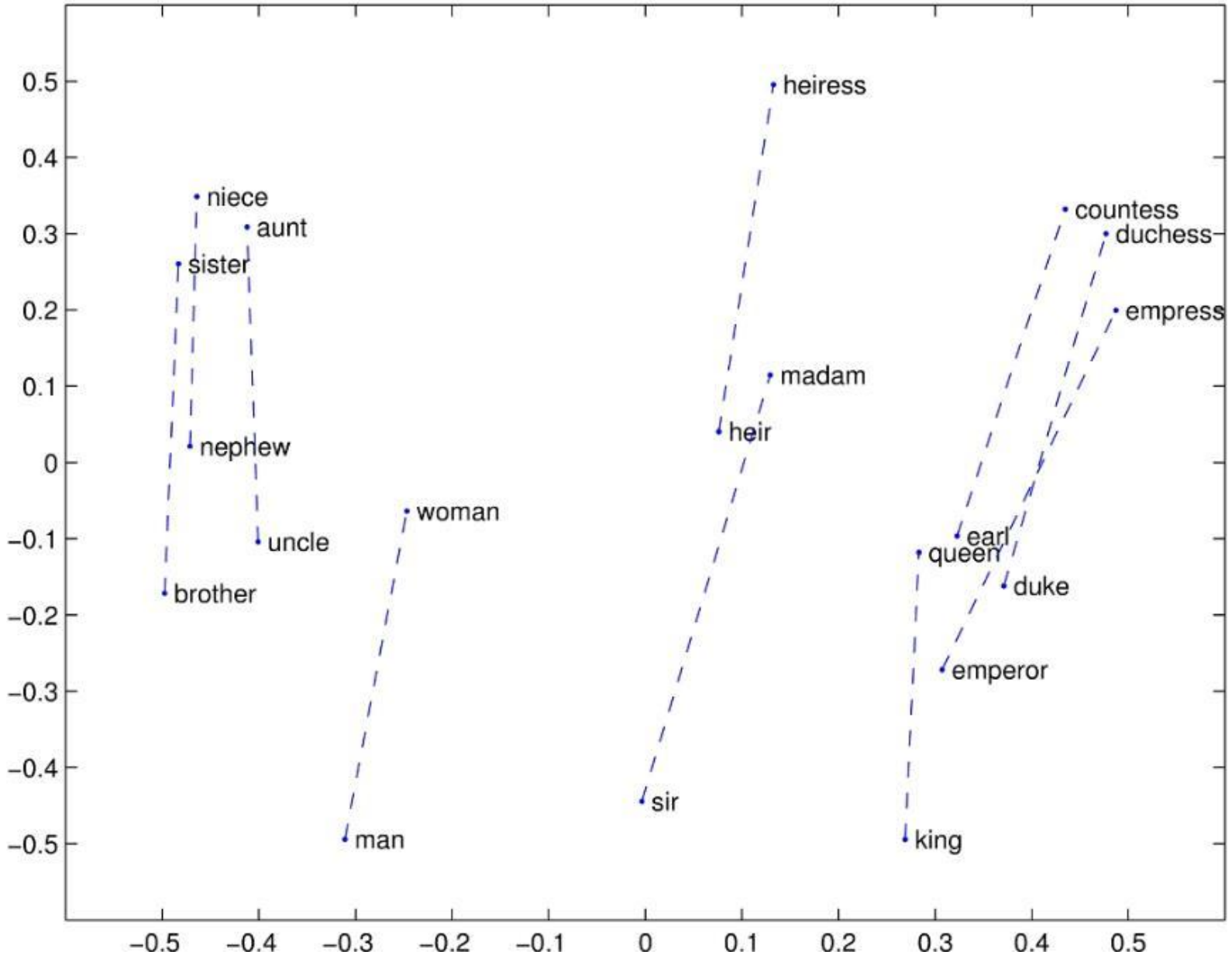
Relational similarity

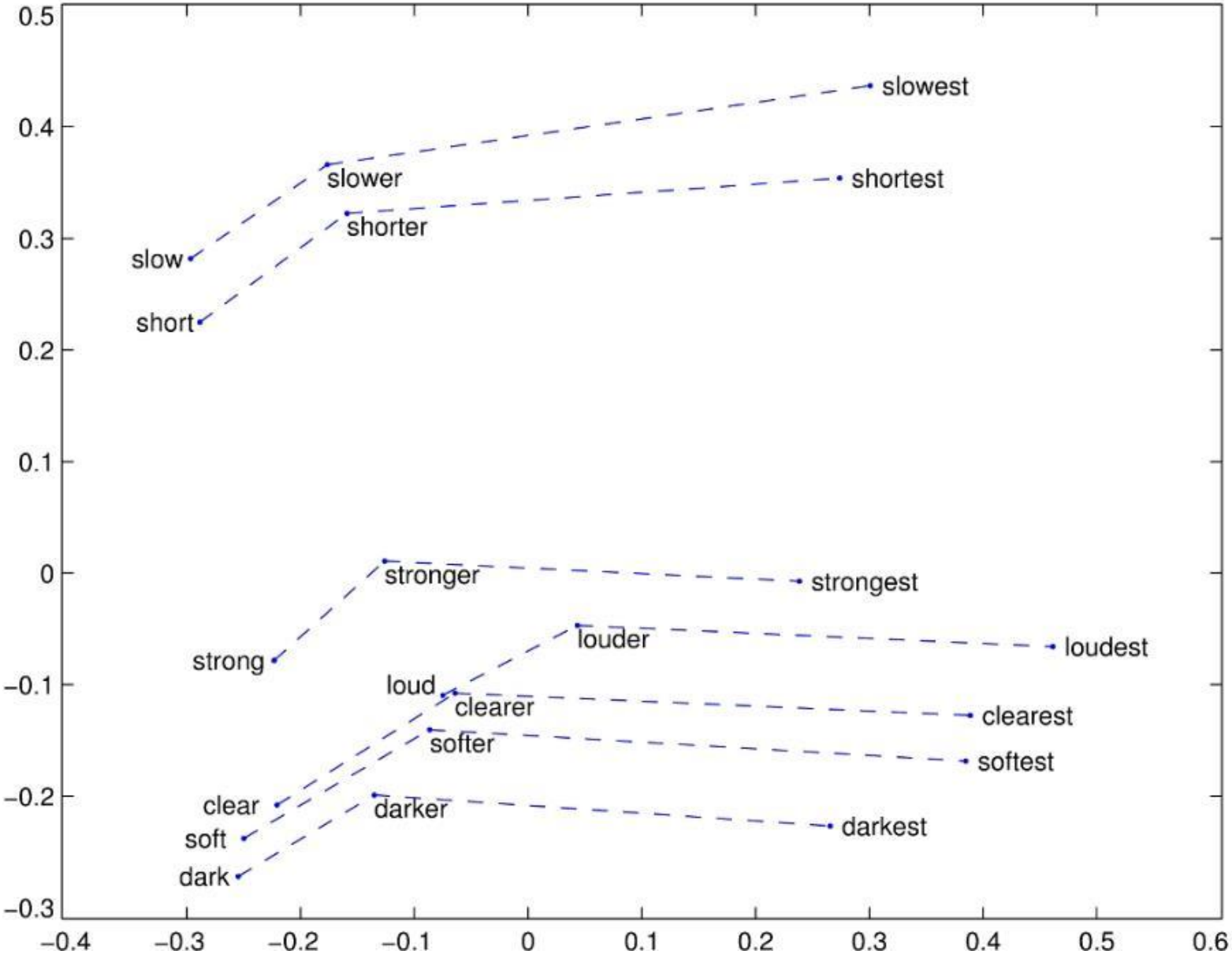




vector(‘*king*’) - vector(‘*man*’) + vector(‘*woman*’) ≈ vector(‘queen’)

vector(‘*Paris*’) - vector(‘*France*’) + vector(‘*Italy*’) ≈ vector(‘Rome’)

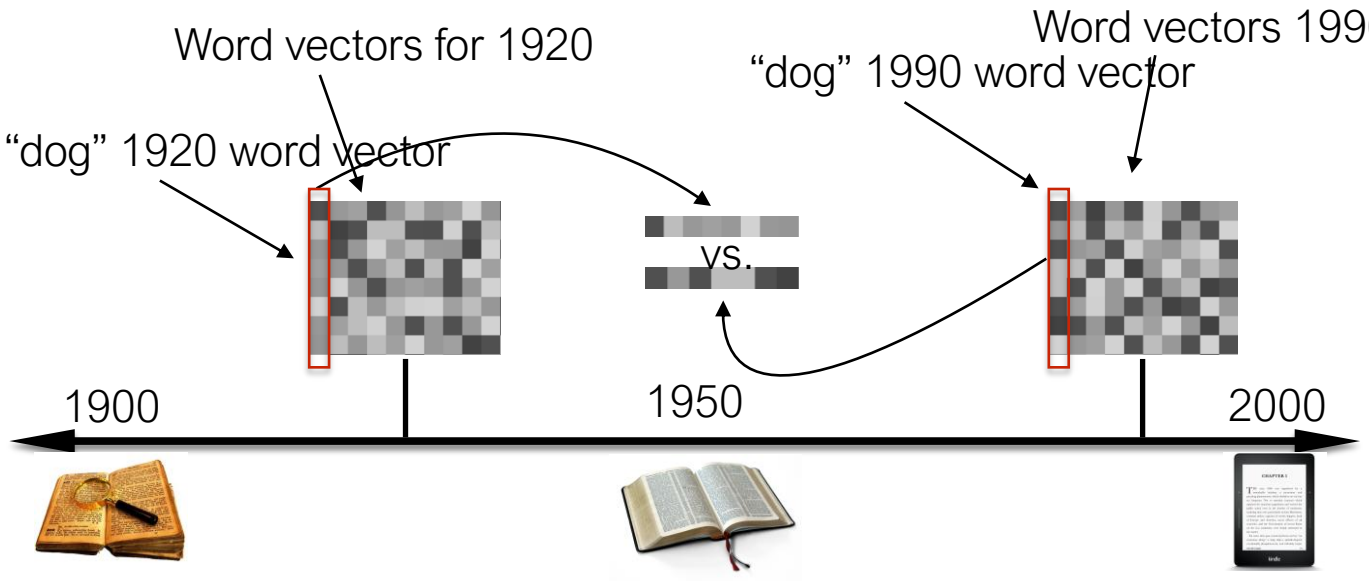

0.5
0.4
0.3
0.2
0.1
0
−0.1
−0.2
−0.3
−0.4
−0.5
−0.5
−0.4
−0.3
−0.2
−0.1
0
0.1
0.2
0.3
0.4
0.5
niece
aunt
sister
nephew
uncle
brother
woman
man
heiress
madam
heir
sir
countess
duchess
empress
earl
queen
duke
emperor
king


slowest
slower
shortest
shorter
slow
short
stronger
strongest
louder
loudest
strong
loud
clearer
clearest
softer
softest
clear
darker
darkest
soft
dark
0.5
0.4
0.3
0.2
0.1
0
−0.1
−0.2
−0.3
−0.4
−0.3
−0.2
−0.1
0
0.1
0.2
0.3
0.4
0.5
0.6

Embeddings can help study word history

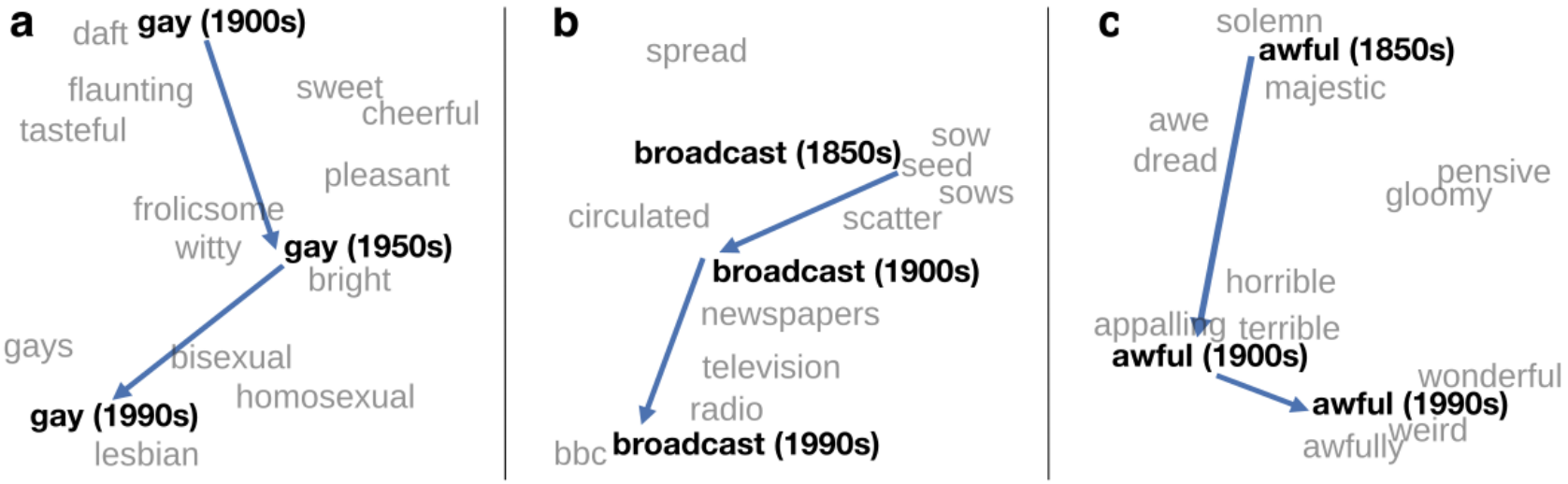
- Train embeddings on old books to study changes in word meaning!!

Diachronic word embeddings for studying language change



Visualizing changes

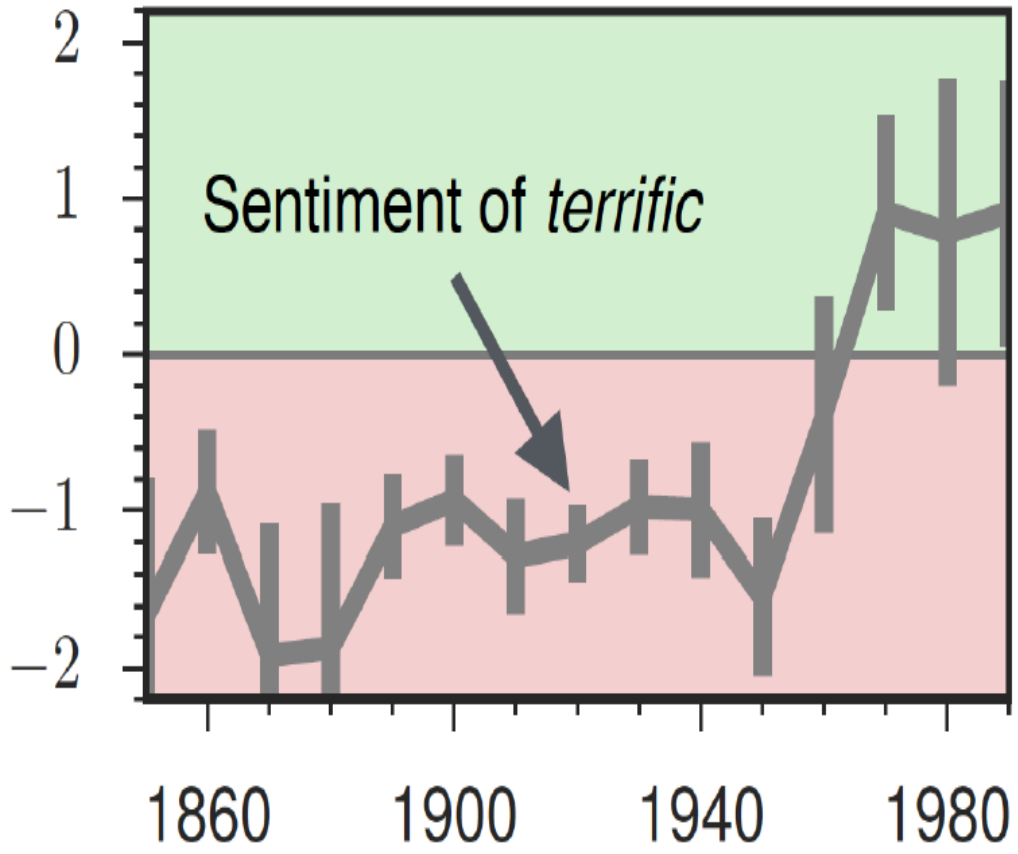
Project 300 dimensions down into 2



~30 million books, 1850-1990, Google Books data

The evolution of sentiment words

Negative words change faster than positive words



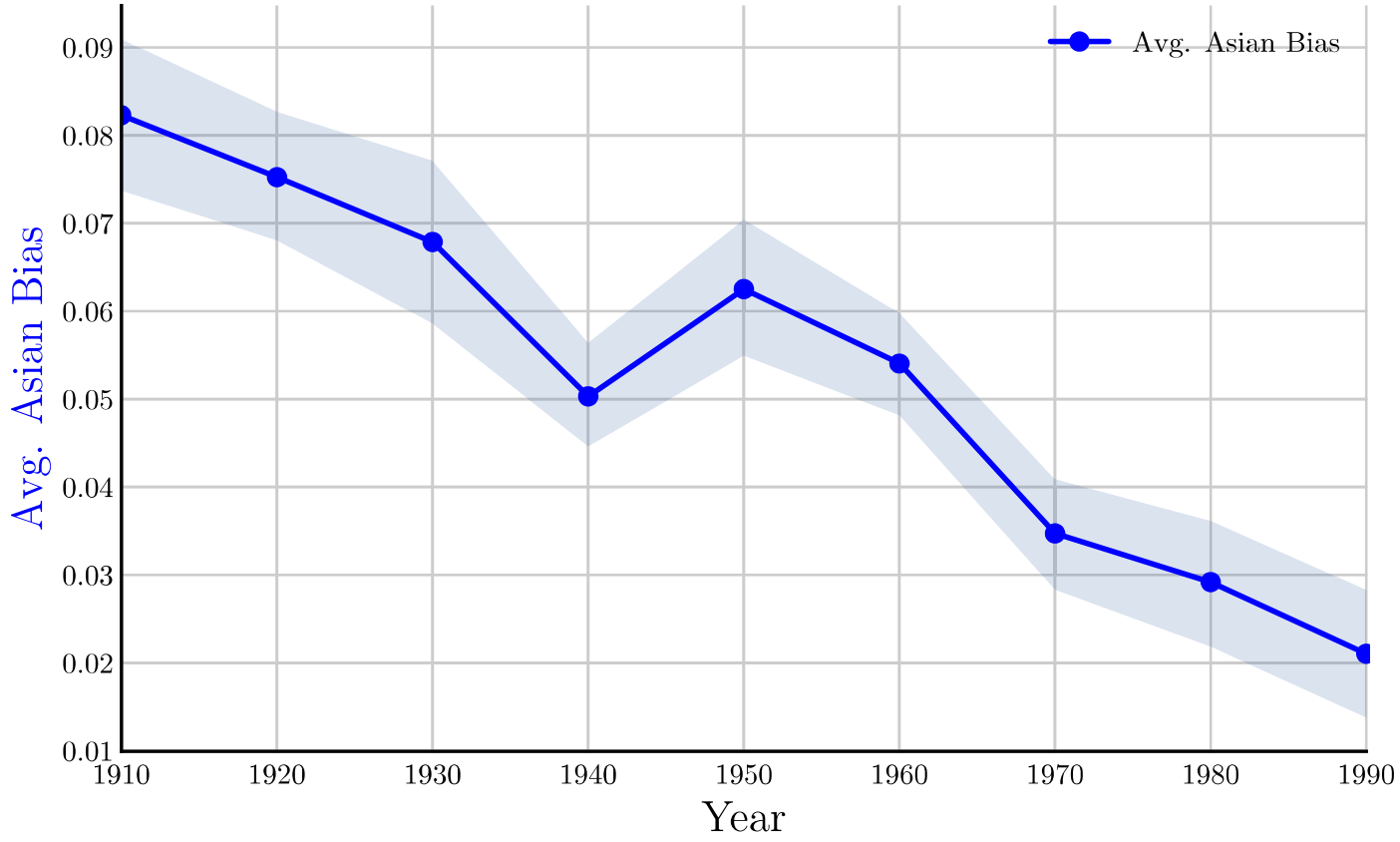
Embeddings reflect cultural bias

- Ask “Paris : France :: Tokyo : x”
  - x = Japan
- Ask “father : doctor :: mother : x”
  - x = nurse
- Ask “man : computer programmer :: woman : x”
  - x = homemaker

Bolukbasi, Tolga, Kai-Wei Chang, James Y. Zou, Venkatesh Saligrama, and Adam T. Kalai. "Man is to computer programmer as woman is to homemaker? debiasing word embeddings." In *Advances in Neural Information Processing Systems*, pp. 4349-4357. 2016.

Change in linguistic framing 1910-1990

Change in association of Chinese names with adjectives framed as "othering" (*barbaric*, *monstrous*, *bizarre*)



Garg, Nikhil, Schiebinger, Londa, Jurafsky, Dan, and Zou, James (2018). Word embeddings quantify 100 years of gender and ethnic stereotypes. *Proceedings of the National Academy of Sciences*, *115*(16), E3635–E3644

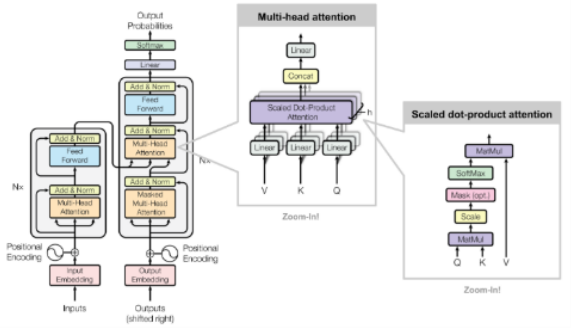
Contextual embeddings / Large language models

- word2vec produces the same vector for a word like bank irrespective of its meaning and context
- recent embeddings take the context into account
- already established as a standard
- e.g., BERT for contextual word embeddings

University of Ljubljana
Faculty of Computer and Information Science

Large language models (LLMs)

- pretrained neural large language models
- trained on large text corpora to capture relations in language
- finetuned to specific tasks
- many are publicly available
- on HuggingFace

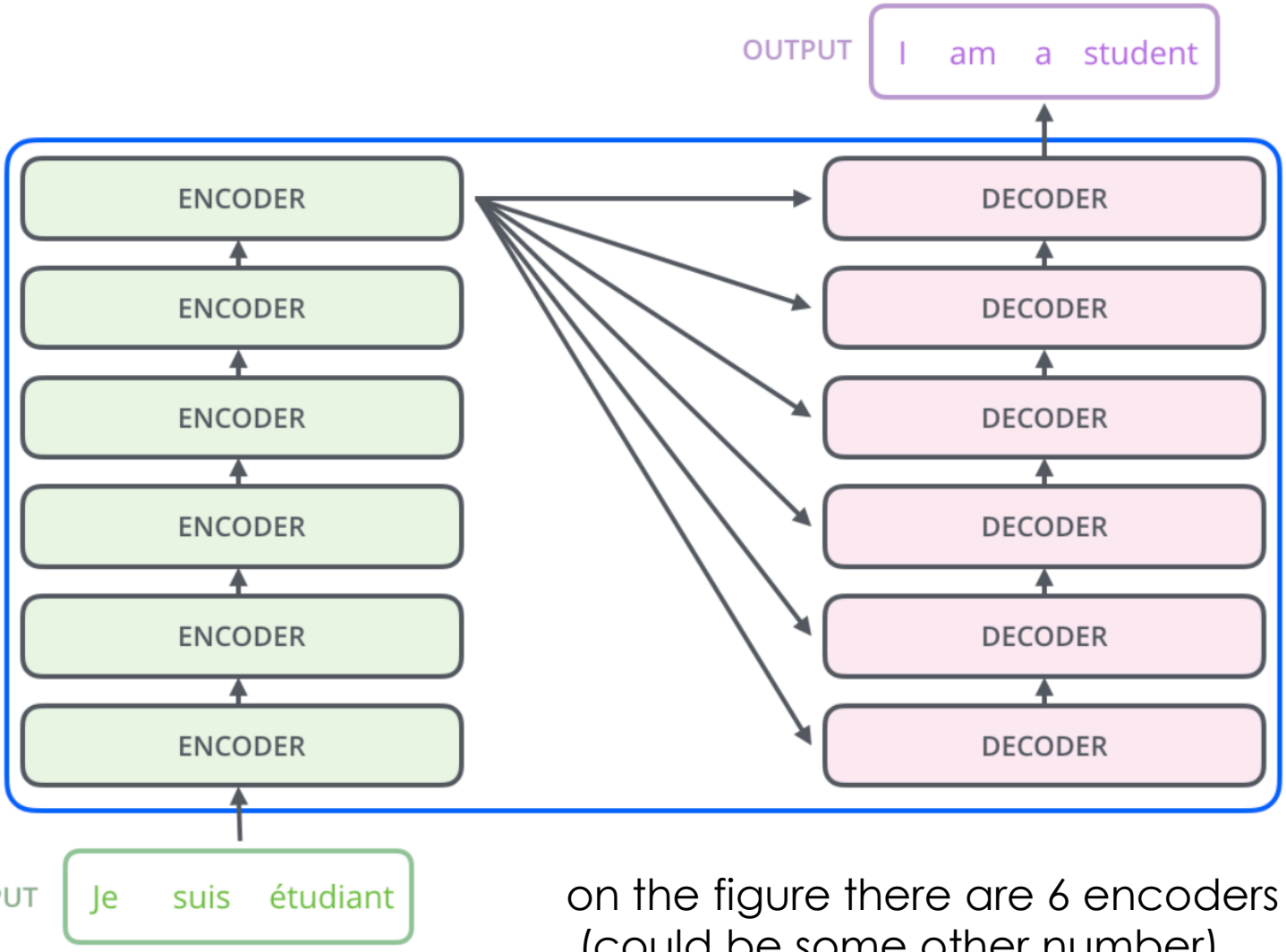


Transformer architecture of NNs

- currently the most successful DNN
- non-recurrent
- architecturally it is an encoder-decoder model
- fixed input length
- can be parallelized
- adapted for GPU (TPU) processing
- based on extreme use of attention

- https://github.com/dair-ai/Transformers-Recipe

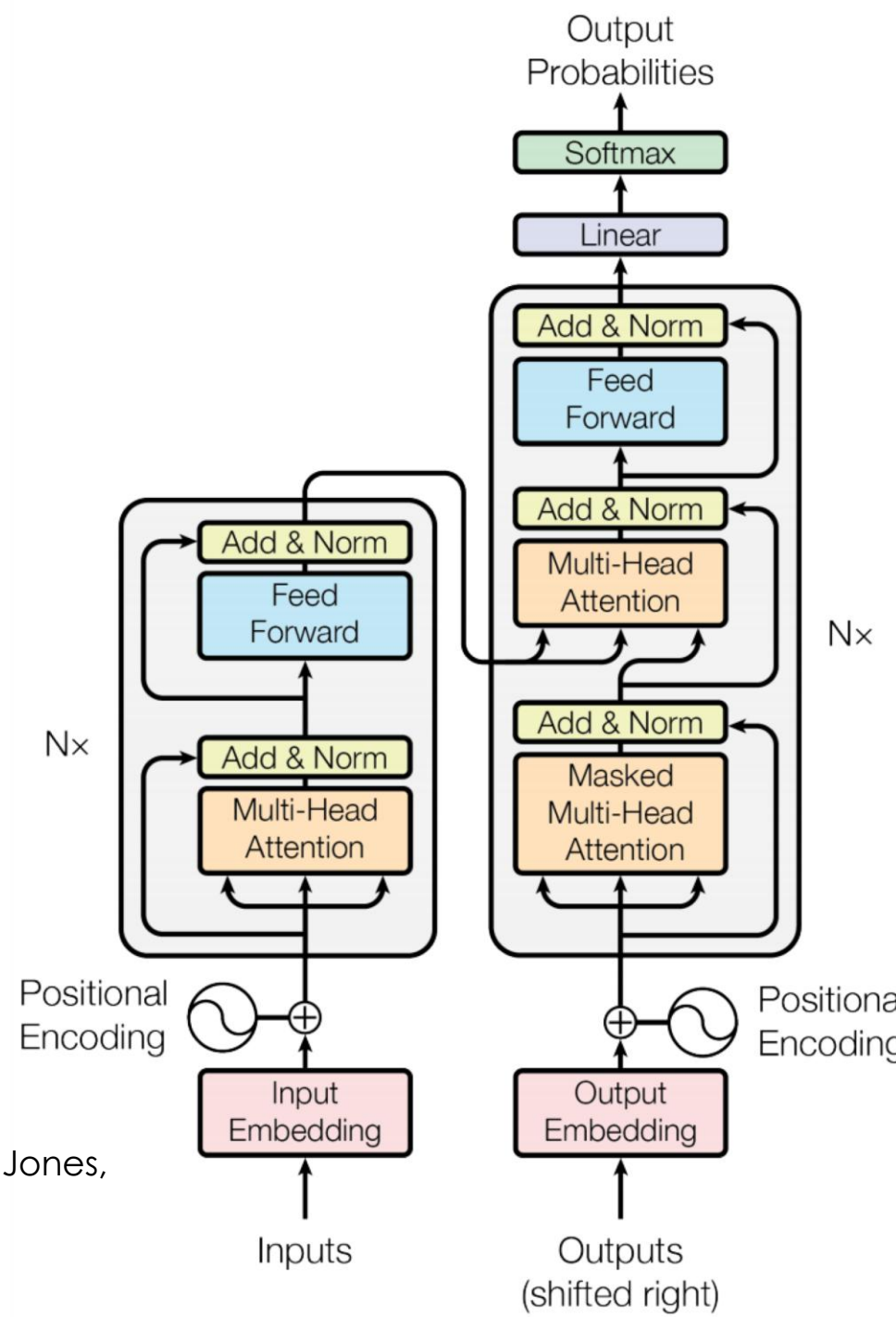
Transformer is an encoder-decoder model



on the figure there are 6 encoders and 6 decoders
(could be some other number)

Transformer overview

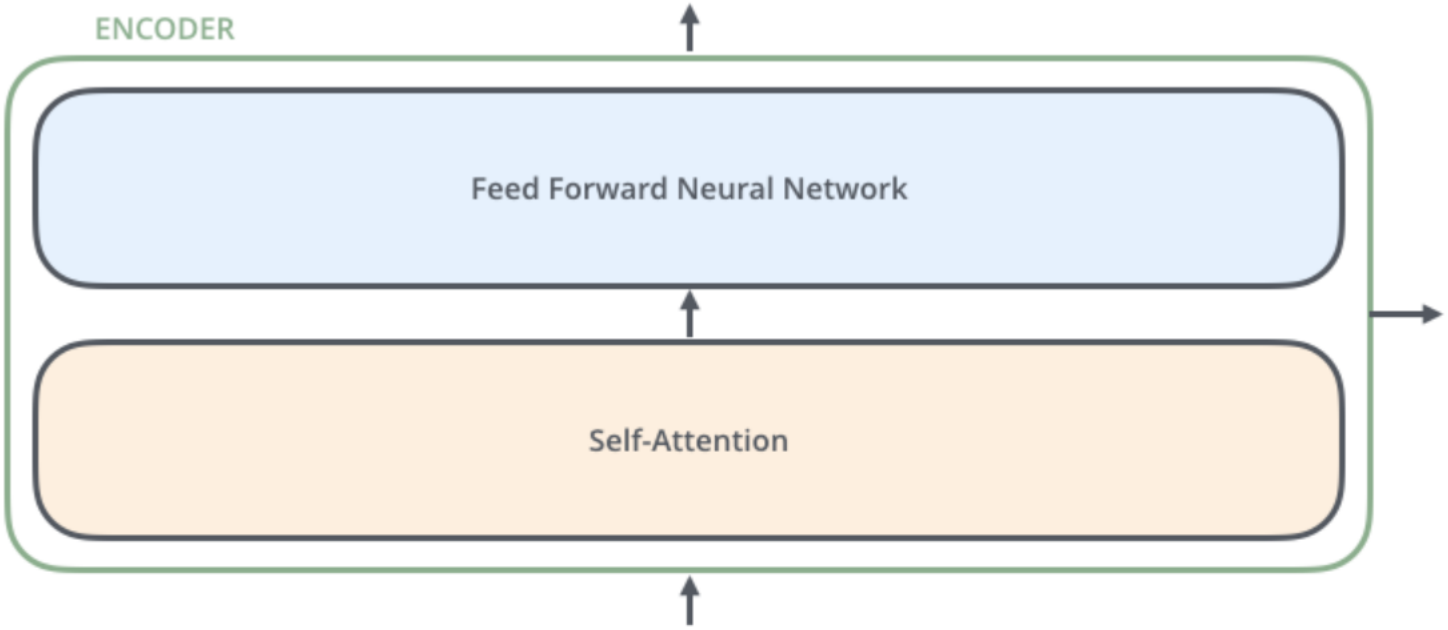
- Initial task: machine translation with parallel corpus
- Predict each translated word
- Final cost/loss/error function is standard cross-entropy error on top of a softmax classifier



Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, Ł. and Polosukhin, I., 2017. Attention is all you need. In *Advances in neural information processing systems* (pp. 5998-6008).

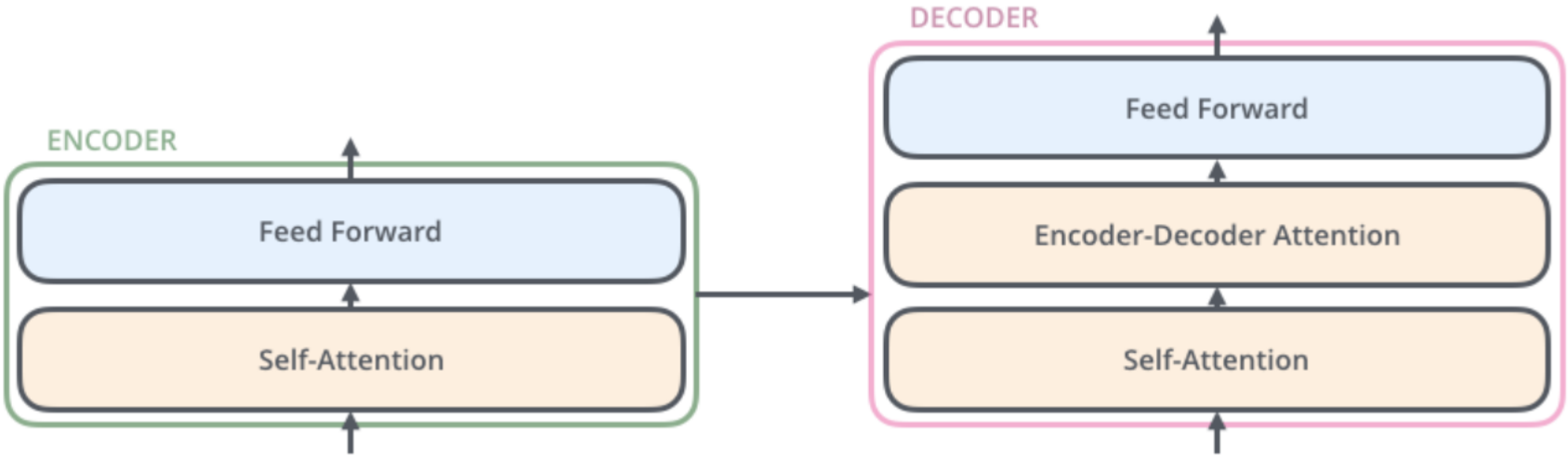
Transformer: encoder

- Two layers
- No weight sharing between different encoders
- Self-attention helps to focus on relevant part of input
- The illustration shows one encoder block

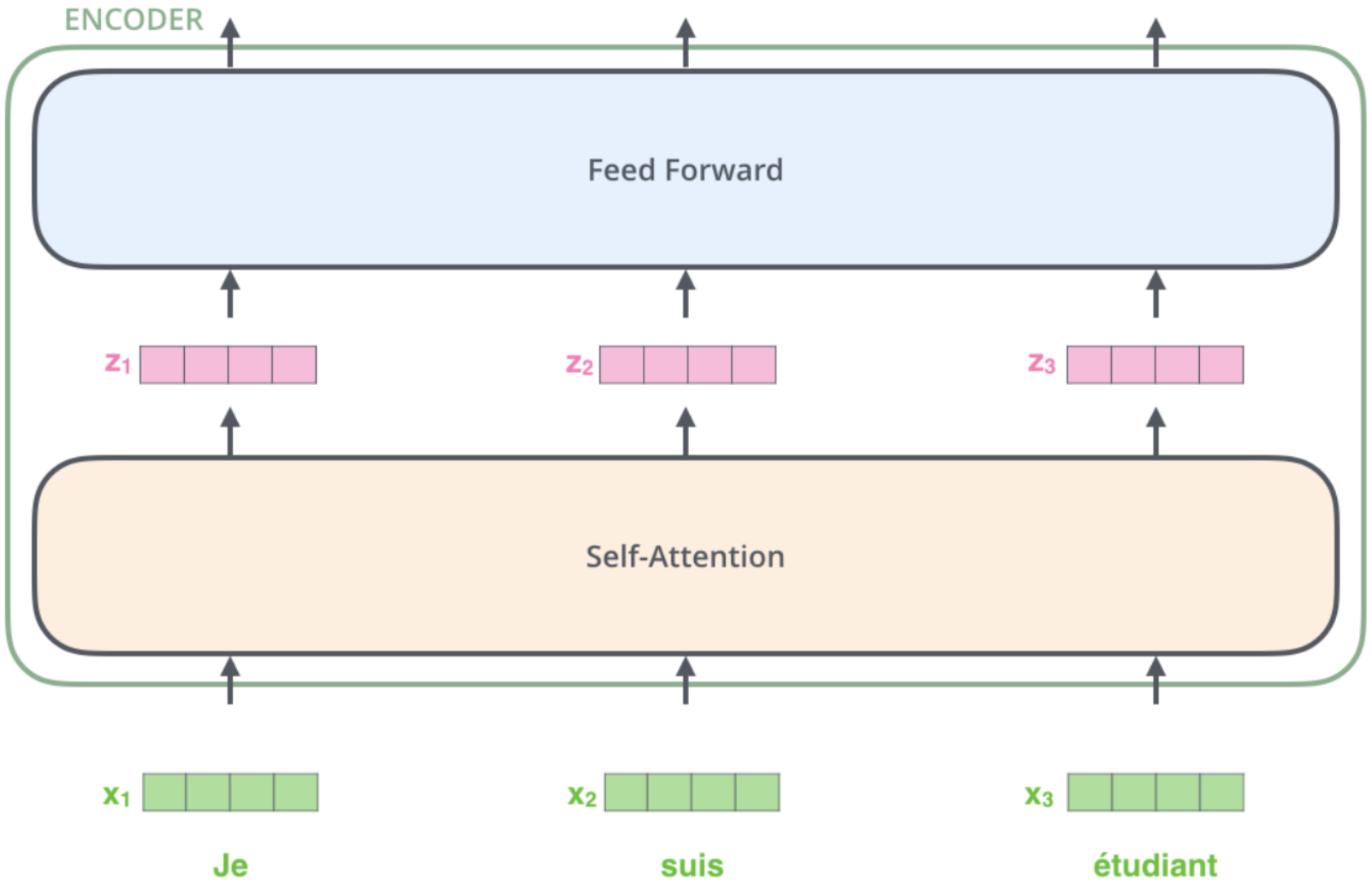


Transformer: decoder

- The same as encoder but with an additional attention layer in between, receiving input fom encoder
- Illustration shows one encoder and one decoder block (usually there are many more encoder and decoder blocks)



Start with embeddings

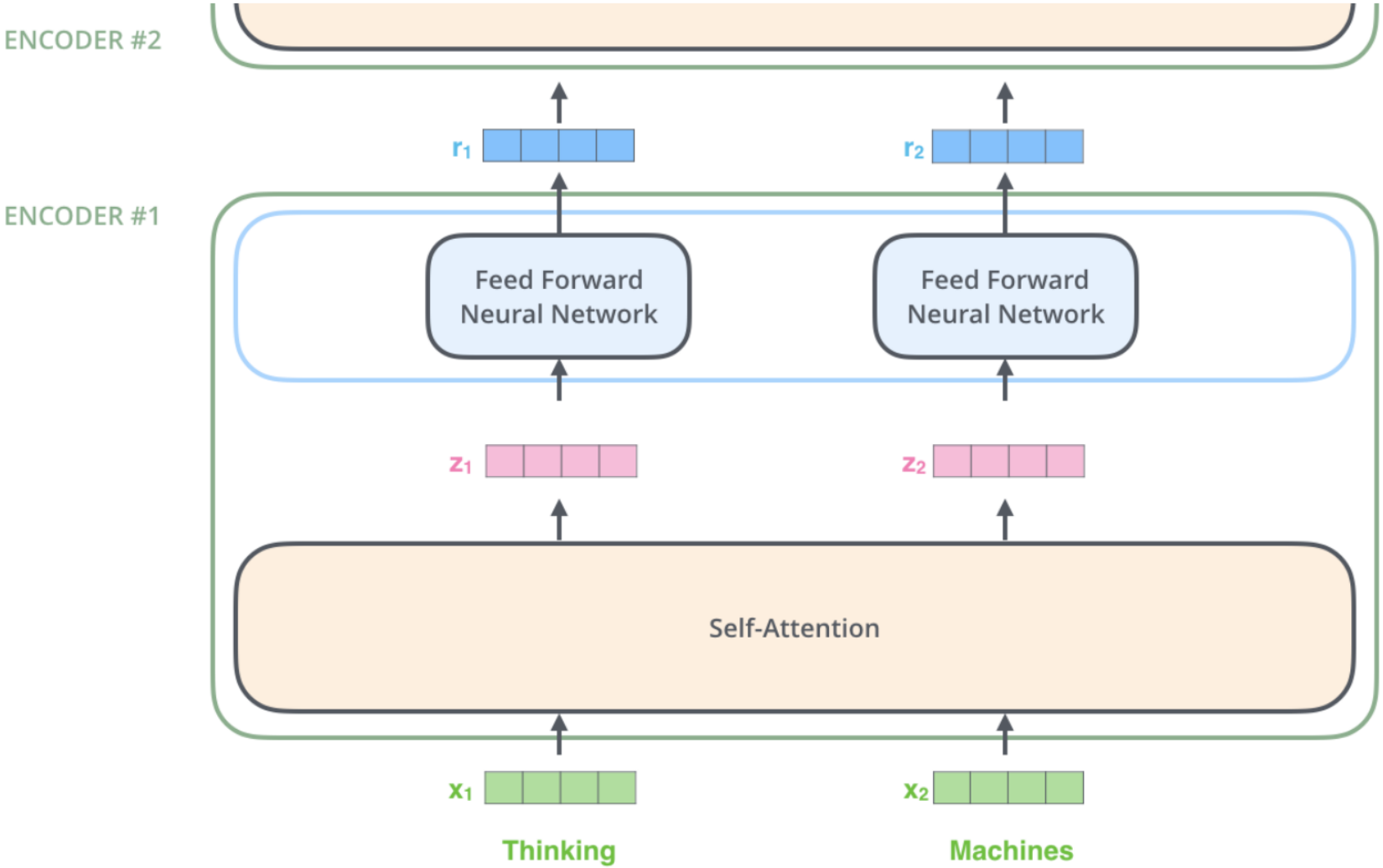


Input to transformer

- embeddings, e.g., 512 dimensional vectors (special, we will discuss that later)
- fixed length, e.g., 128 tokens (in modern architectures at least 8192)
- dependencies between inputs are only in the self-attention layer, no dependencies in feed-forward layer – good for parallelization

- Let us first present the working of the transformer with an illustration of the prediction

Encoding


ENCODER #2
r1
r2
ENCODER #1
Feed Forward Neural Network
Feed Forward Neural Network
z1
z2
Self-Attention
x1
x2
Thinking
Machines

Self-attention

- As the model processes each word (each position in the input sequence), self-attention allows it to look at other positions in the input sequence for clues that can help lead to a better encoding for this word

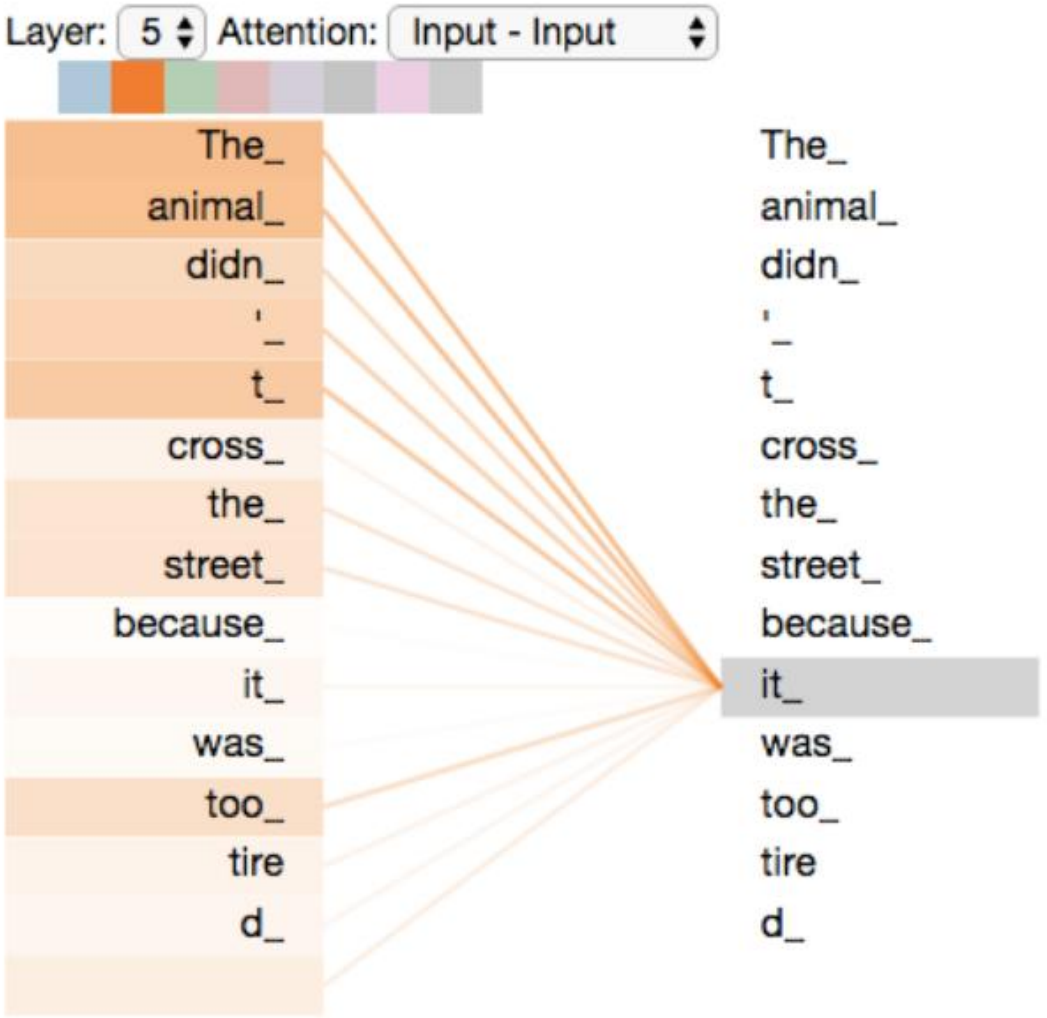
  ”The animal didn't cross the street because it was too tired”

- What does “it” in this sentence refer to? Is it referring to the street or to the animal? It’s a simple question to a human, but not as simple to an algorithm.
- *”The animal didn't cross the street because it was too wide”*

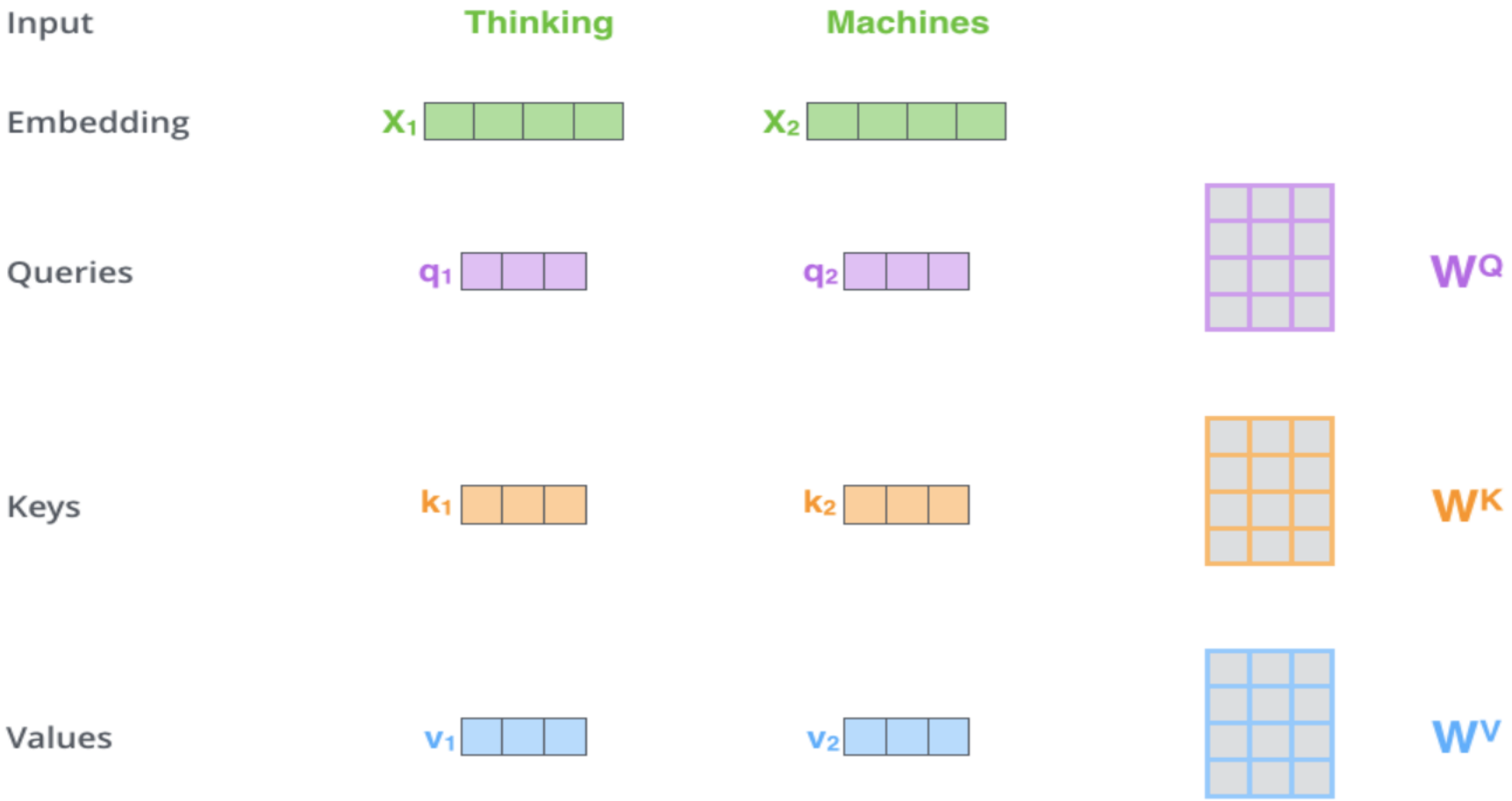
- When the model is processing the word “it”, self-attention allows it to associate “it” with “animal”.

Illustrating self-attention

- As we are encoding the word "it" in encoder #5 (the top encoder in the stack), part of the attention mechanism was focusing on "The Animal", and baked a part of its representation into the encoding of "it".

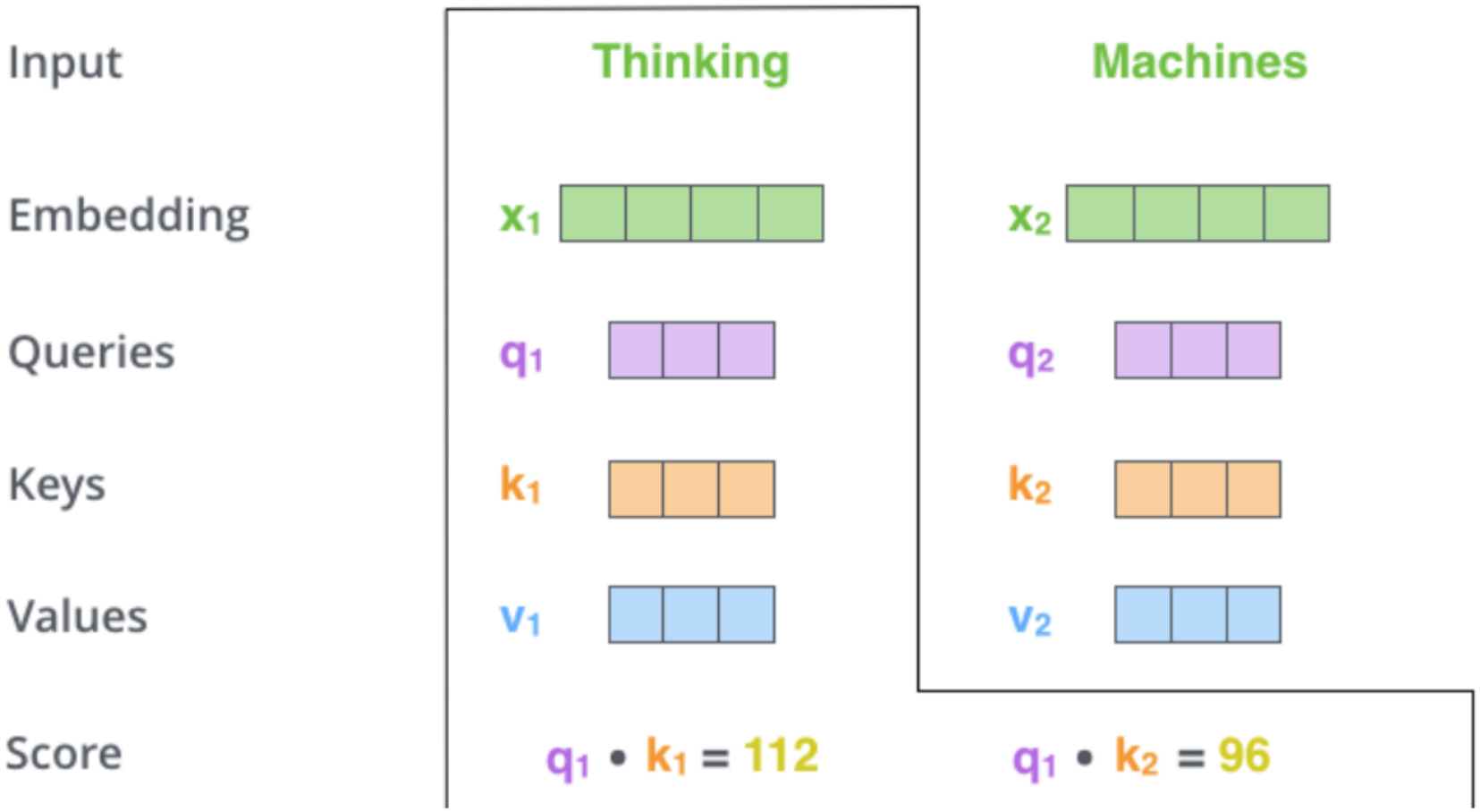


Self-attention details 1/4

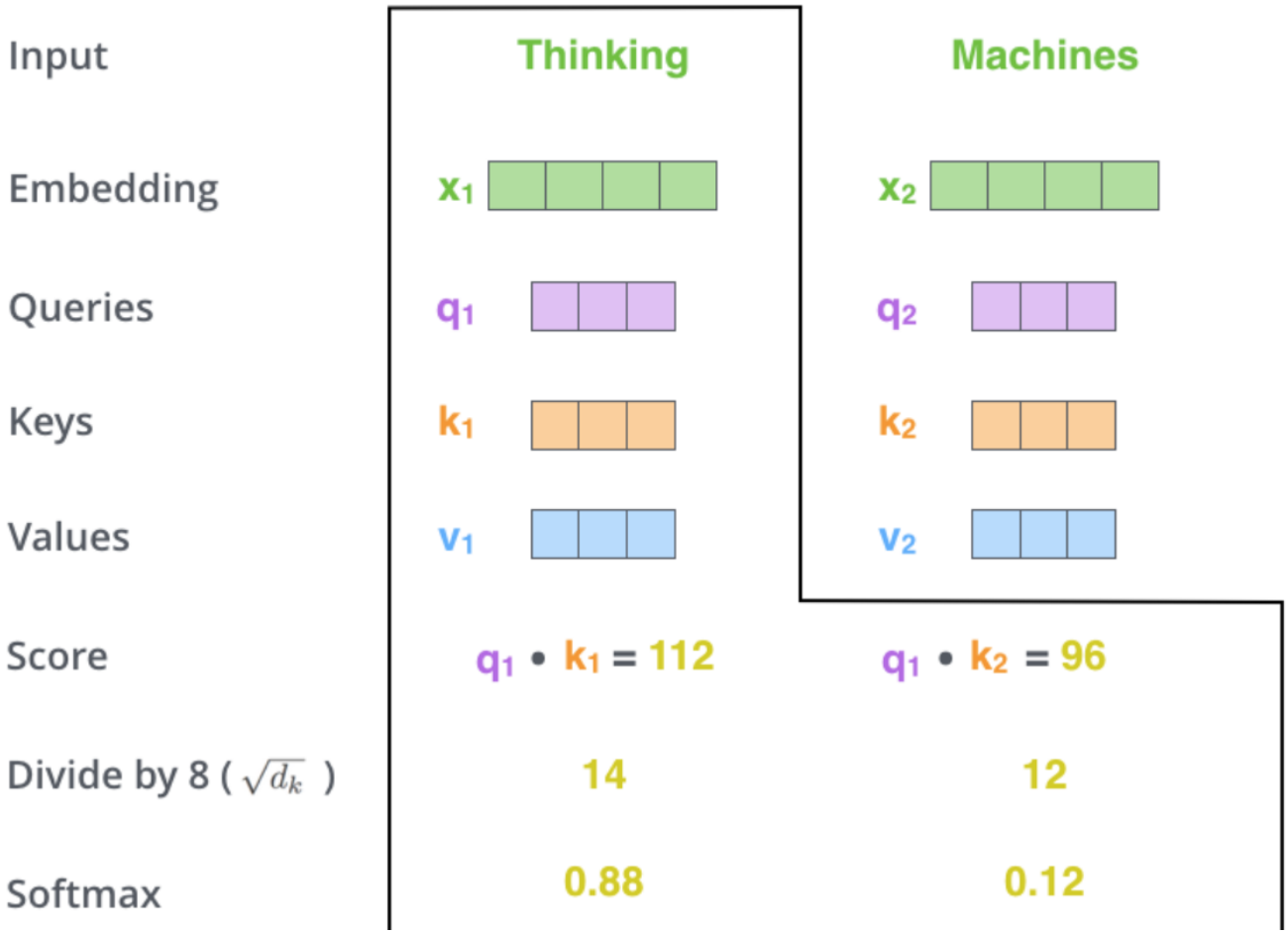


Multiplying x_1 by the W^Q weight matrix produces q_1, the "query" vector associated with that word. We end up creating a "query", a "key", and a "value" projection of each word in the input sentence.

Details 2/4: scoring



Details 3/4: normalization of scores



Details 4/4: self-attention output

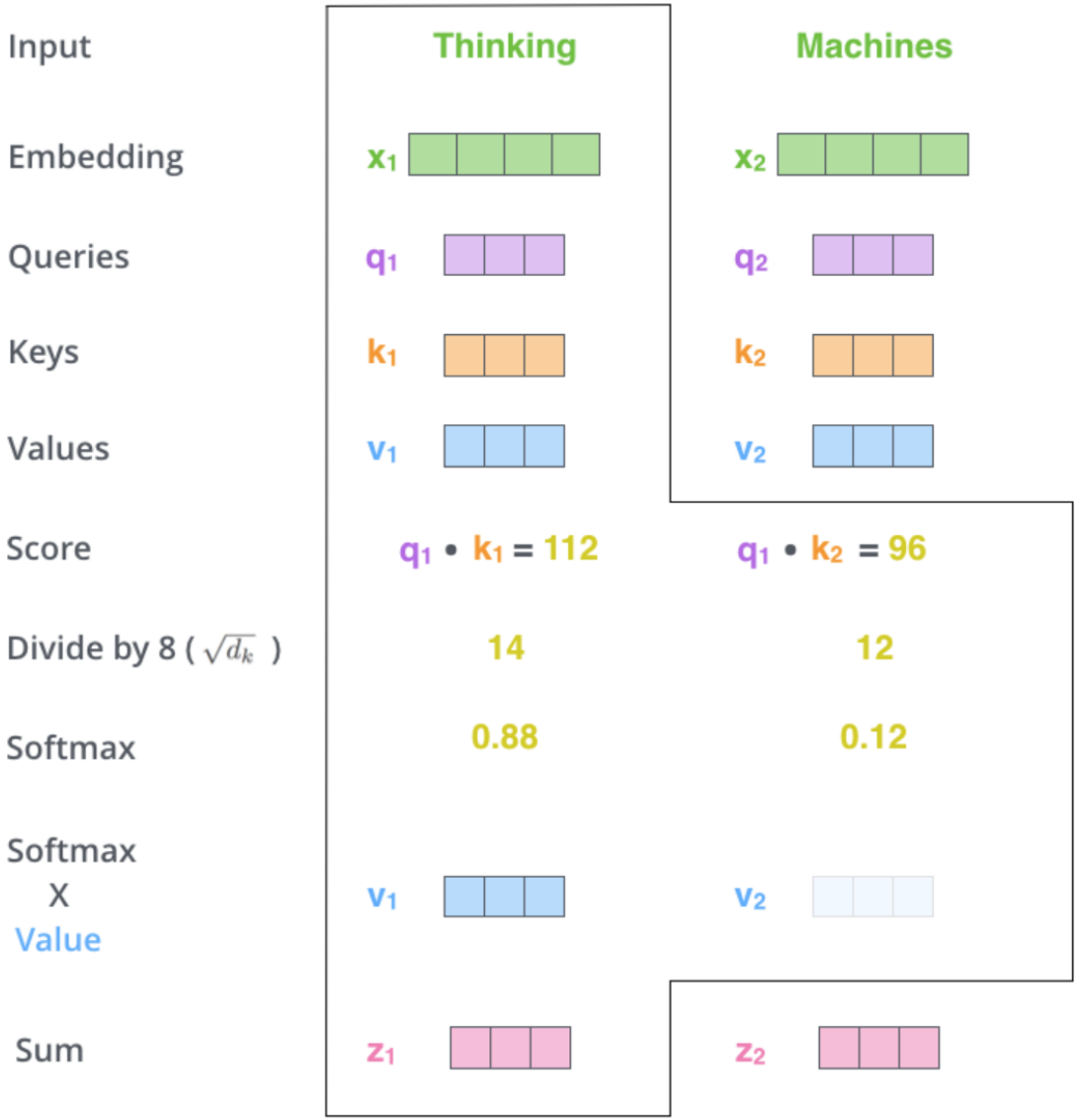
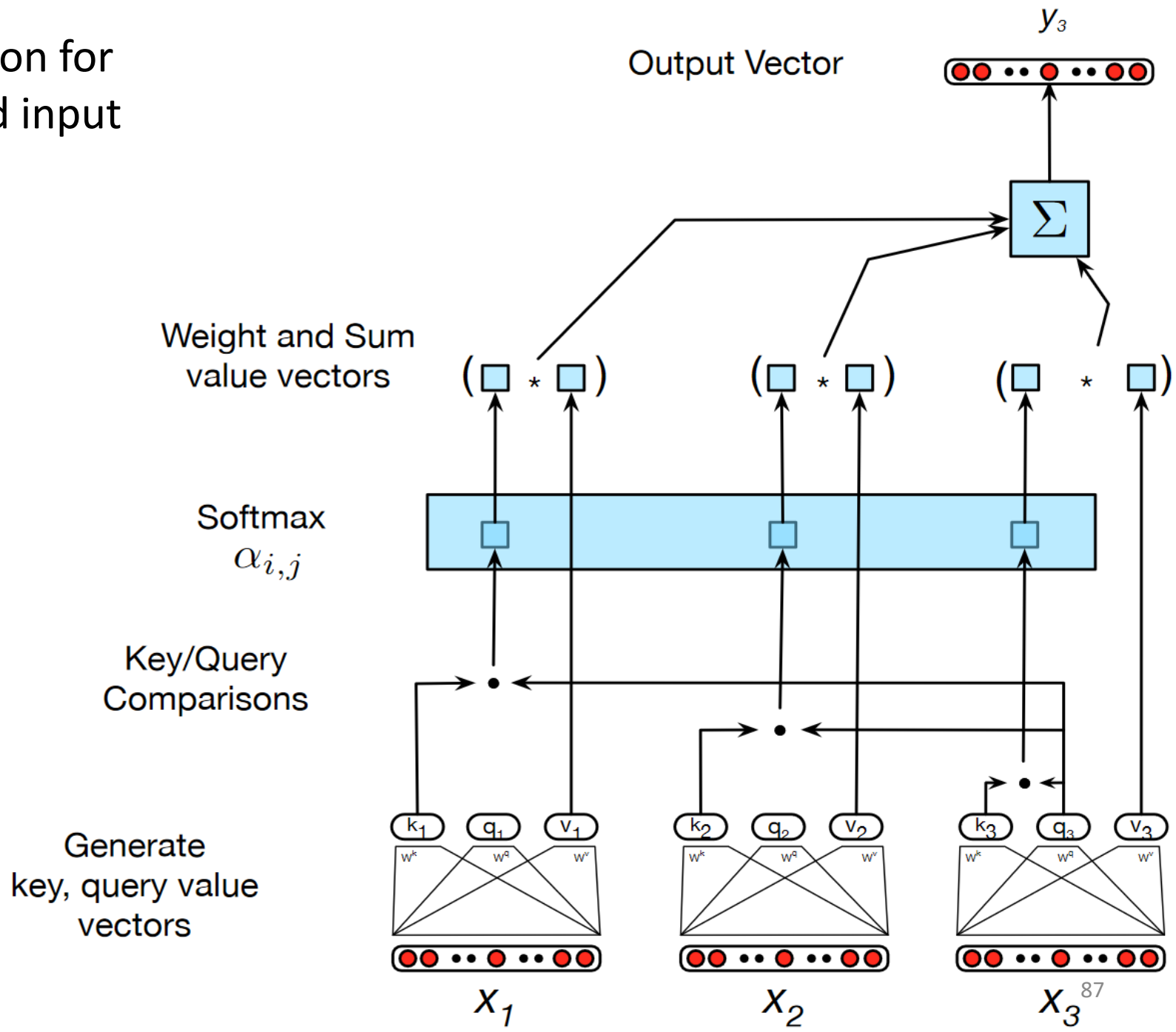
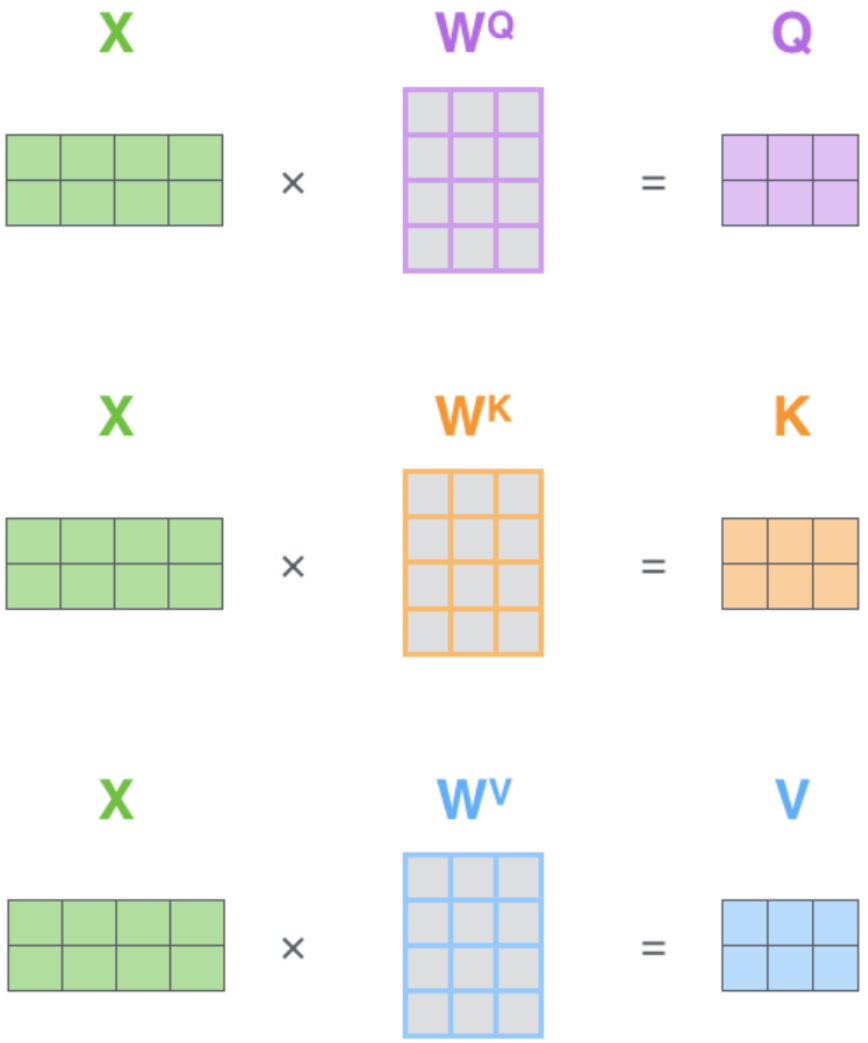


illustration for
the third input



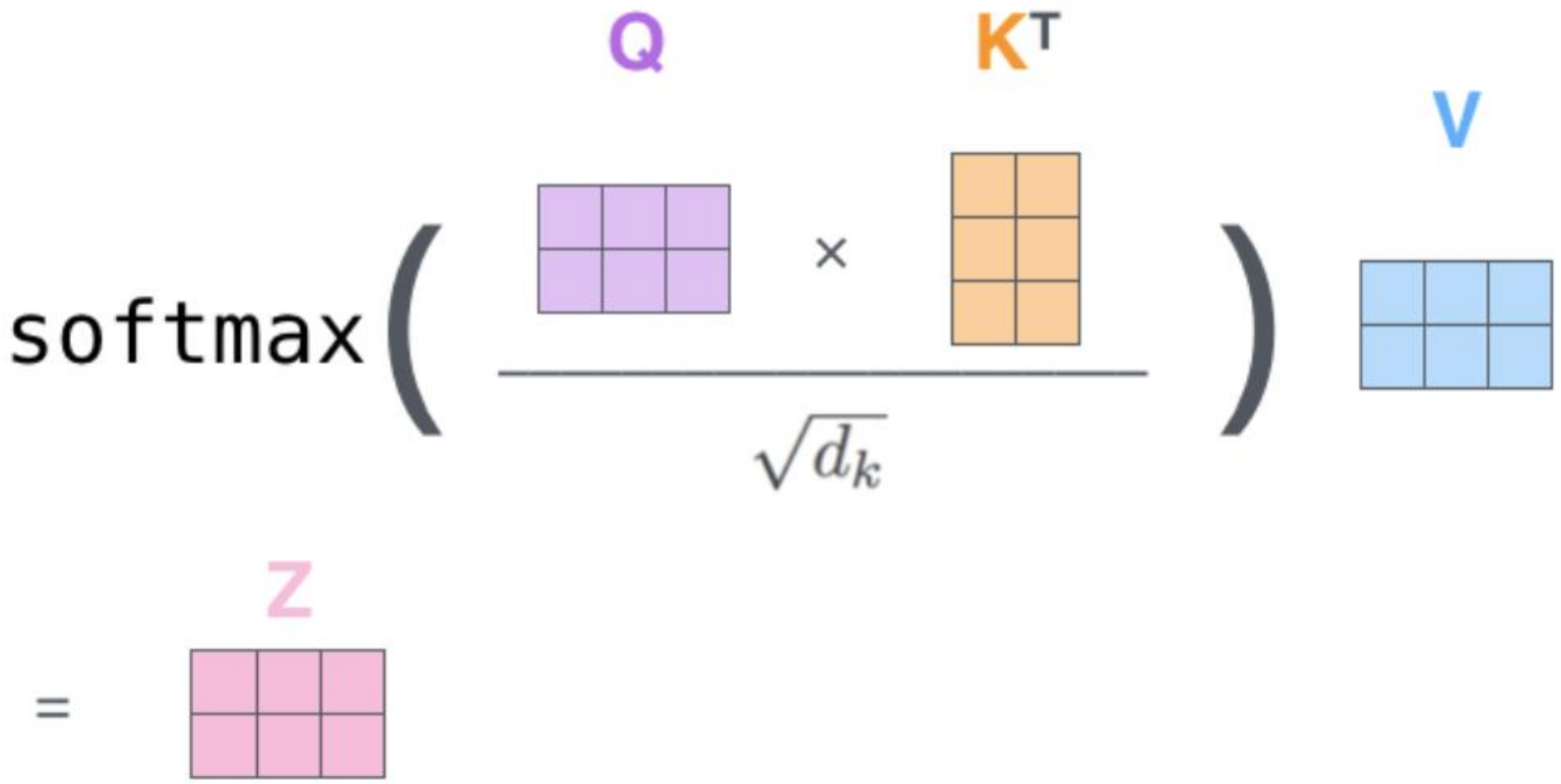
Matrix calculation of self-attention 1/2

Every row in the X matrix corresponds to a word in the input sentence.
The embedding vector x (512) is larger then the q/k/v vectors (64)

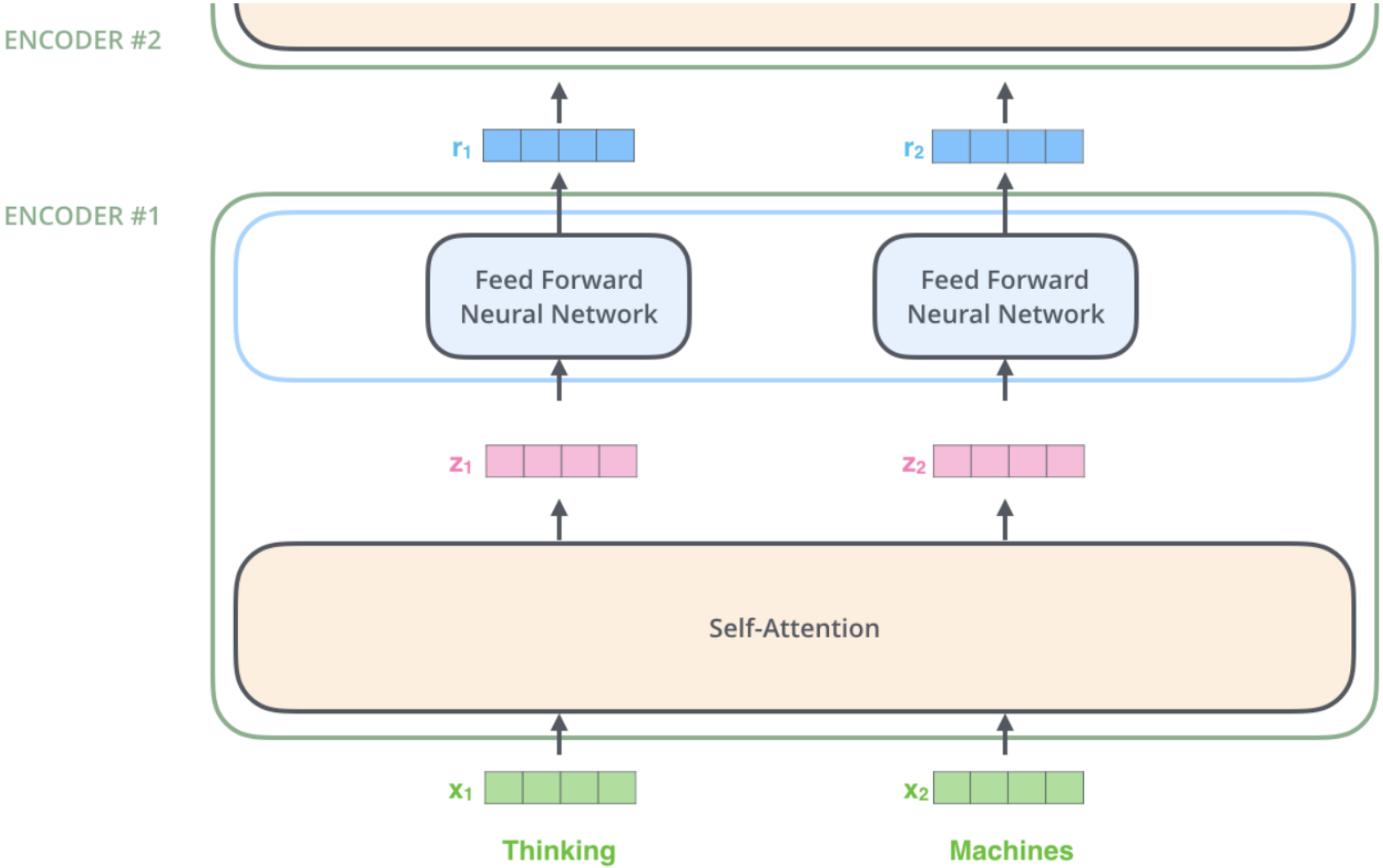


Matrix calculation of self-attention 2/2

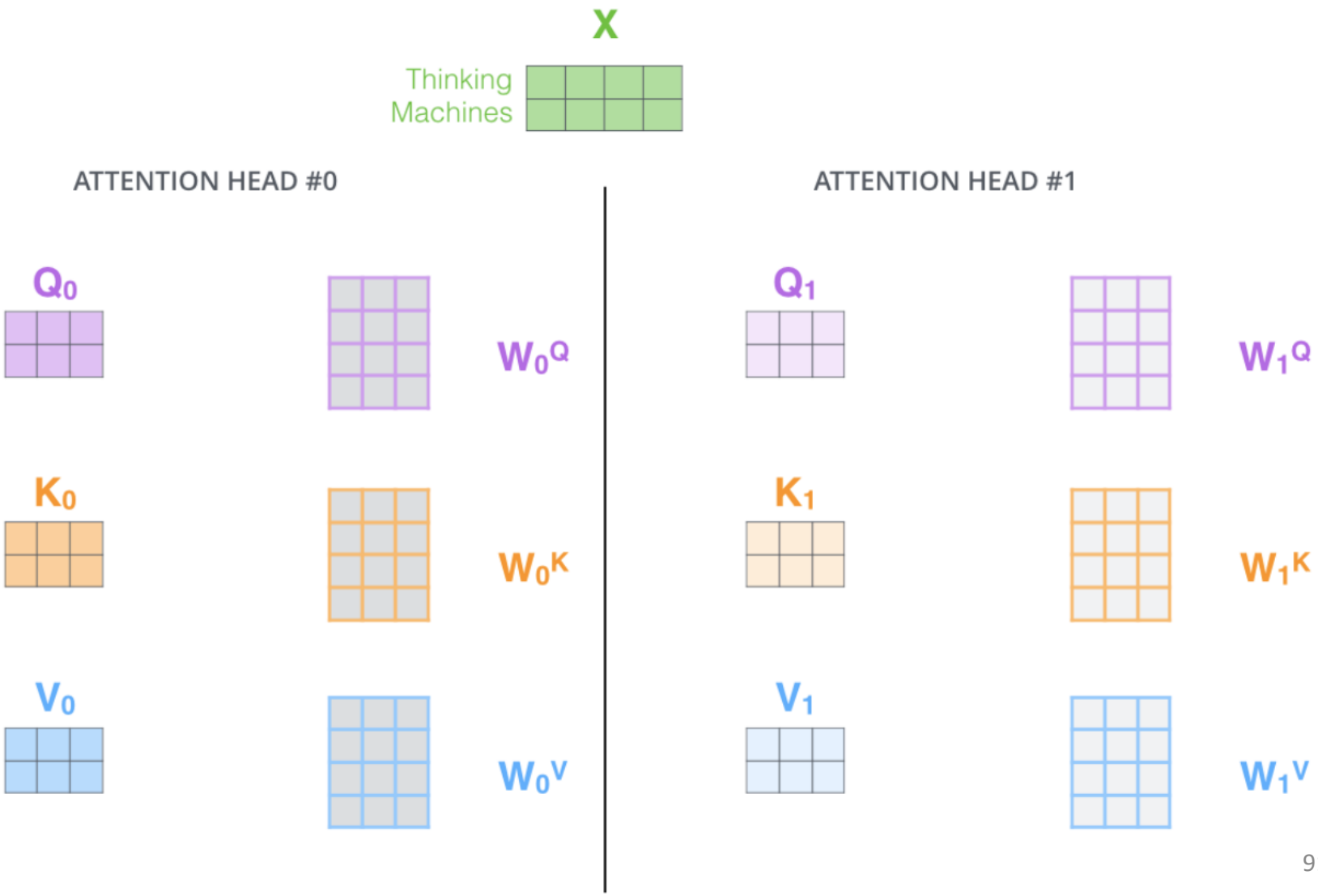
- final calculation



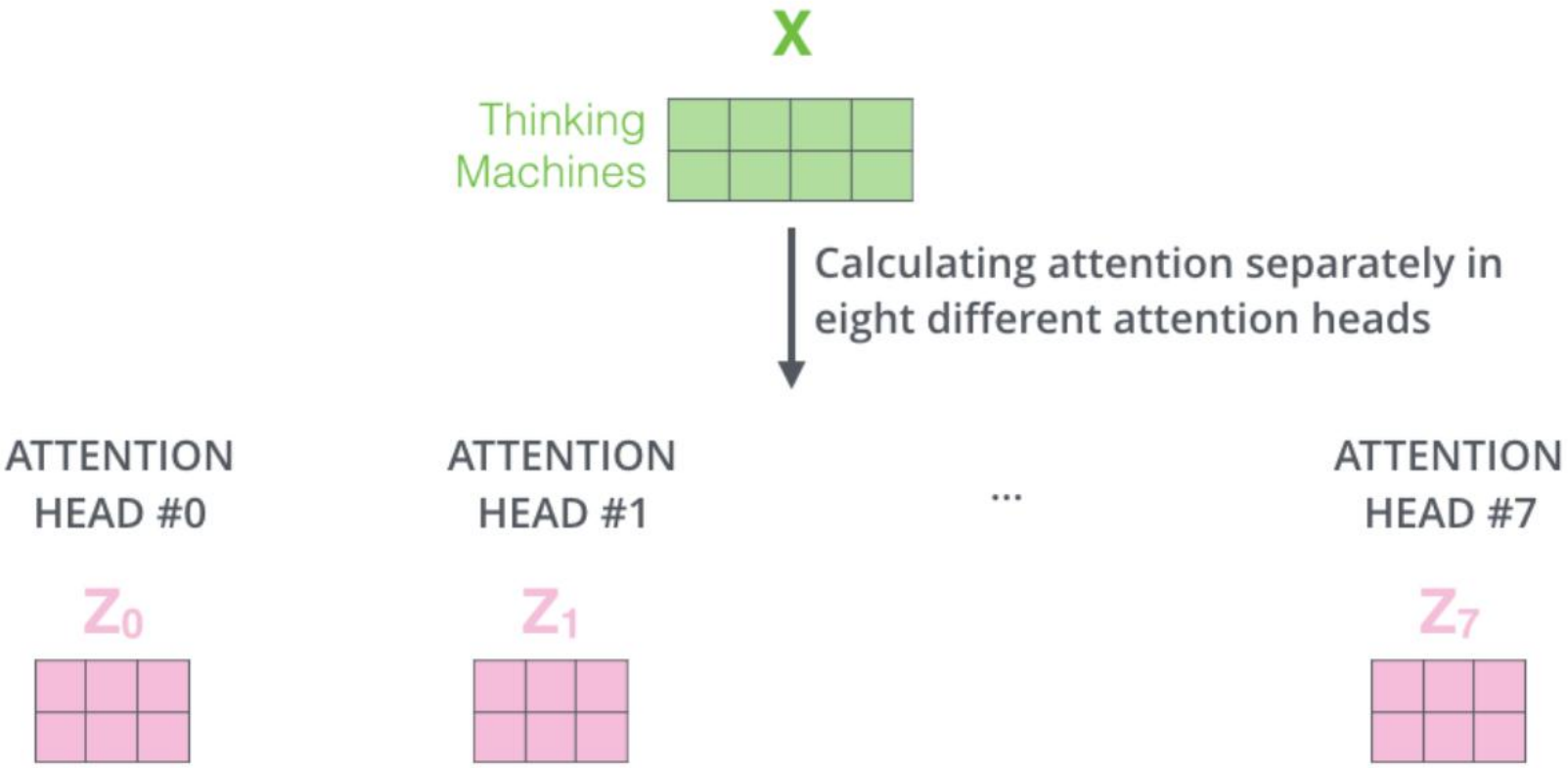
Encoding


ENCODER #2
ENCODER #1
r1
r2
Feed Forward Neural Network
Feed Forward Neural Network
z1
z2
Self-Attention
x1
x2
Thinking
Machines

Example: two attention heads



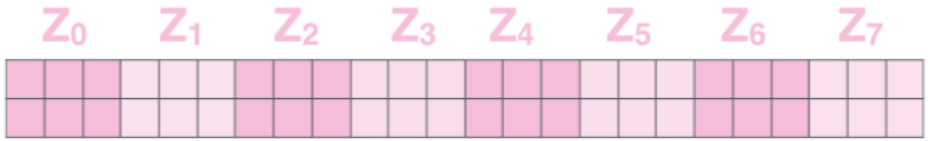
Example: 8 att. heads



- What to do with 8 Z matrices, the feed-forward layer is expecting a single matrix (one vector for each word). We need to condense all attention heads into one matrix.

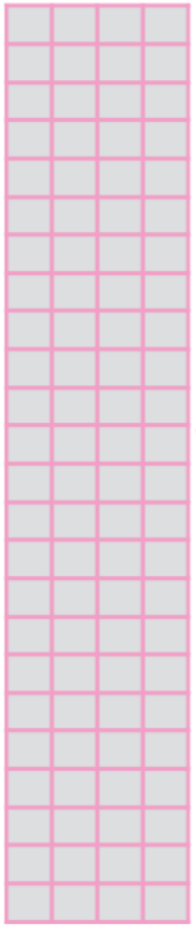
Condensation of attention heads

1) Concatenate all the attention heads



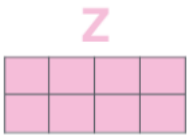
2) Multiply with a weight matrix W^O that was trained jointly with the model

X



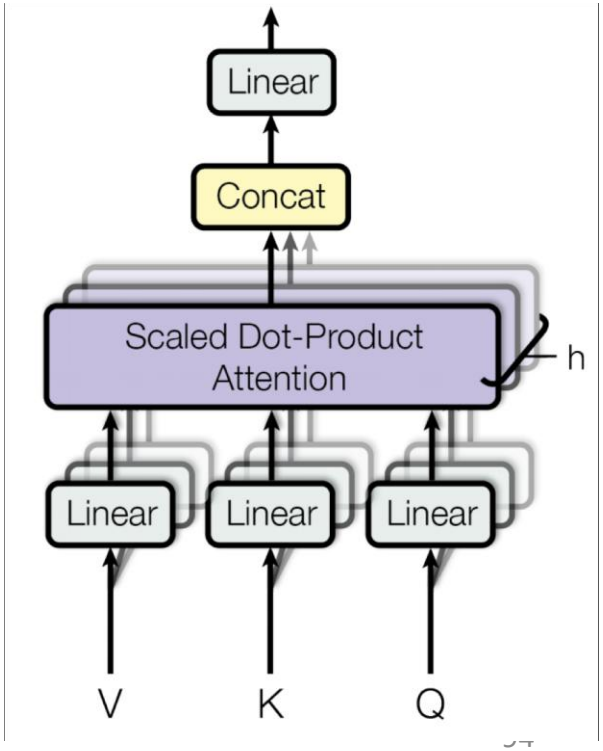
3) The result would be the Z matrix that captures information from all the attention heads. We can send this forward to the FFNN

=



Computing multi-head attention

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, ..., \text{head}_\text{h})W^O$$
$$\text{where head}_\text{i} = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$



Summary of self-attention

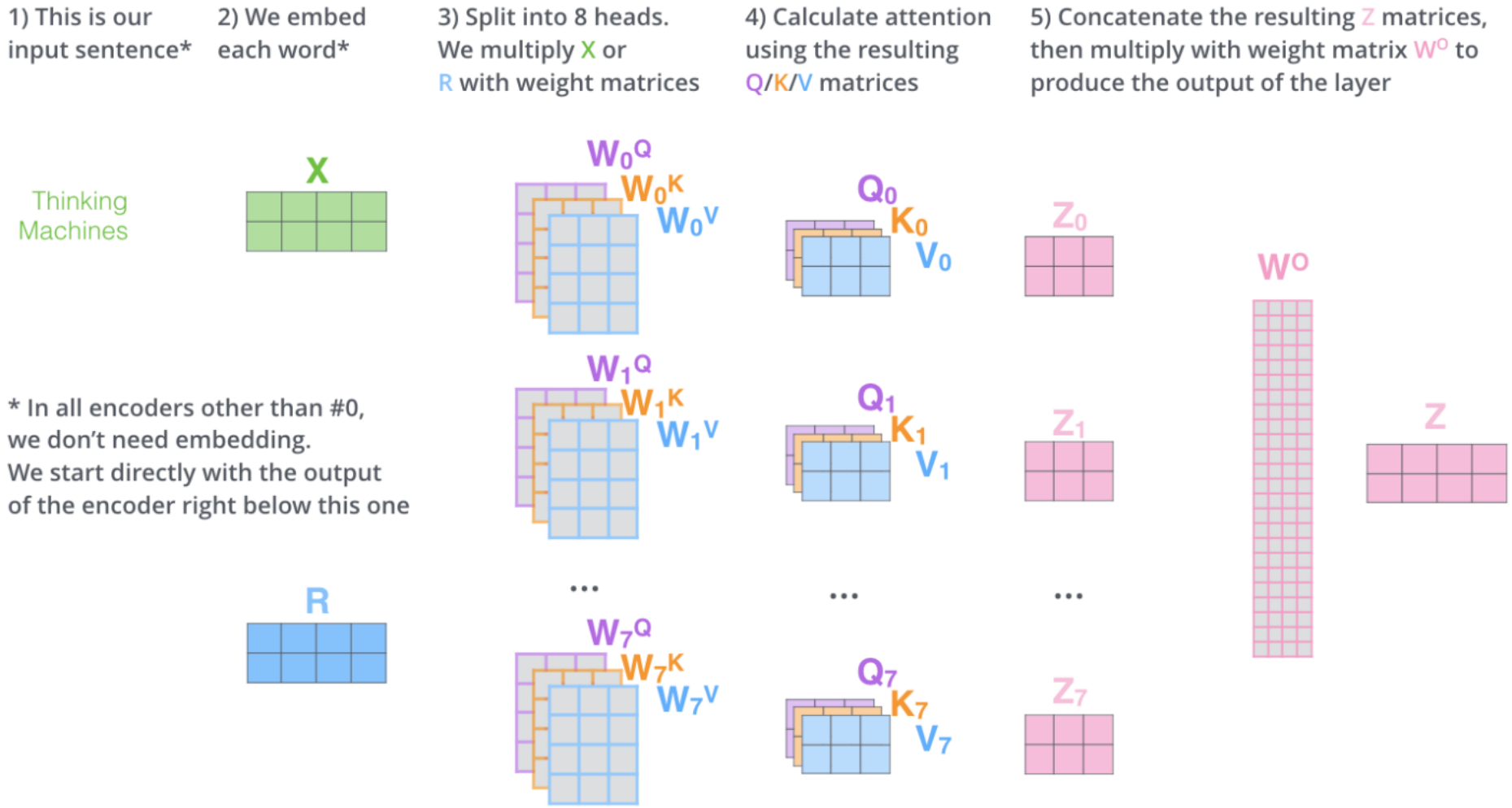


Illustration of self-attention: 1 head

- encoder #5 (the top encoder in the stack)
- As we encode the word "it", one attention head is focusing most on "the animal", while another is focusing on "tired" -- in a sense, the model's representation of the word "it" bakes in some of the representation of both "animal" and "tired".

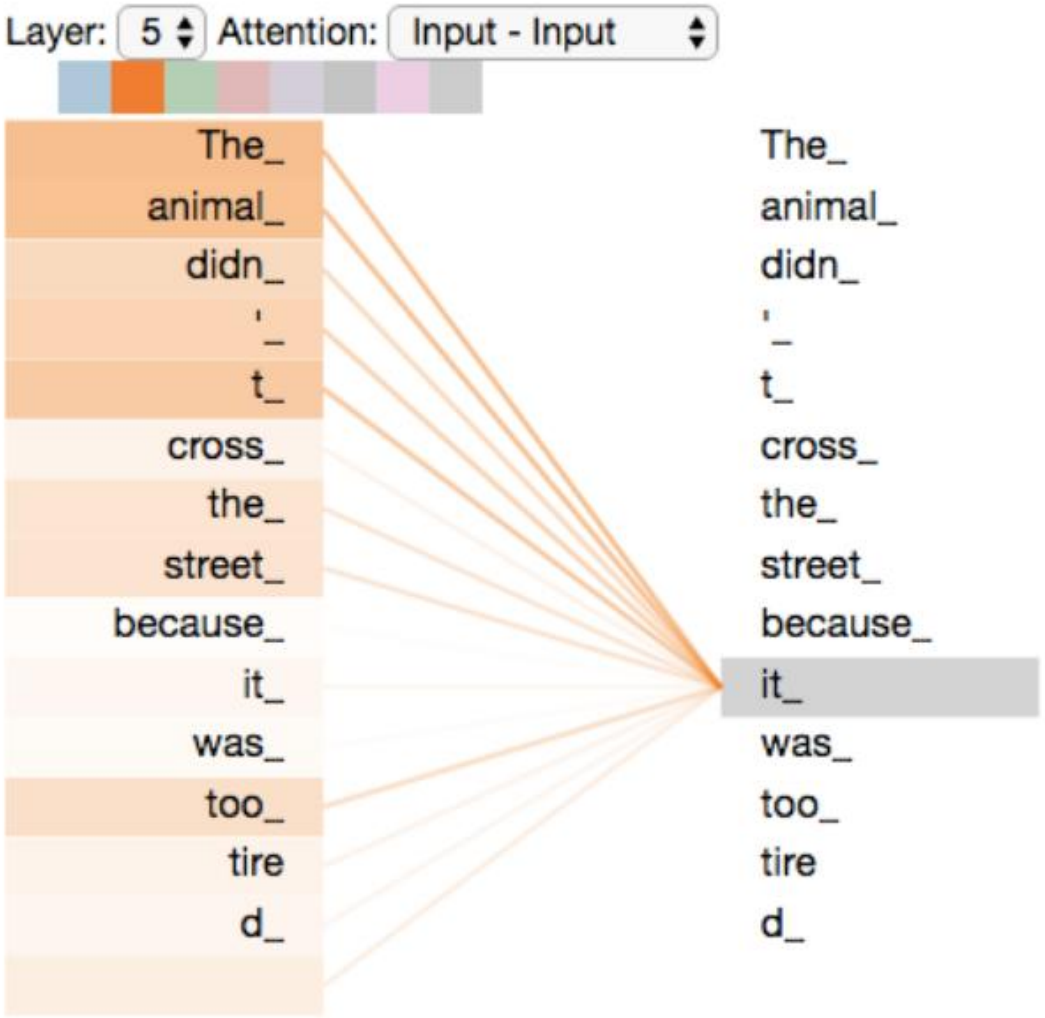
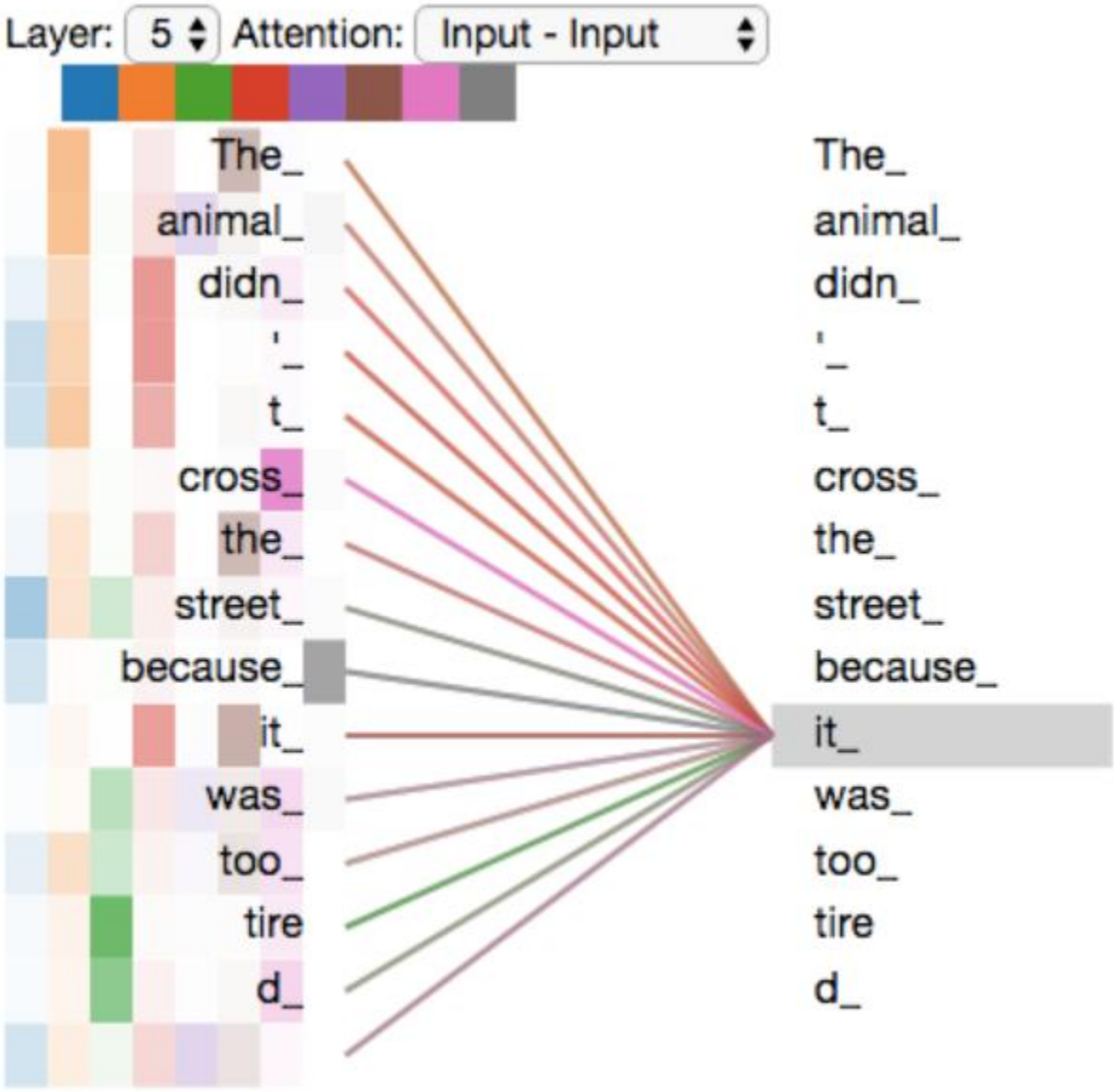
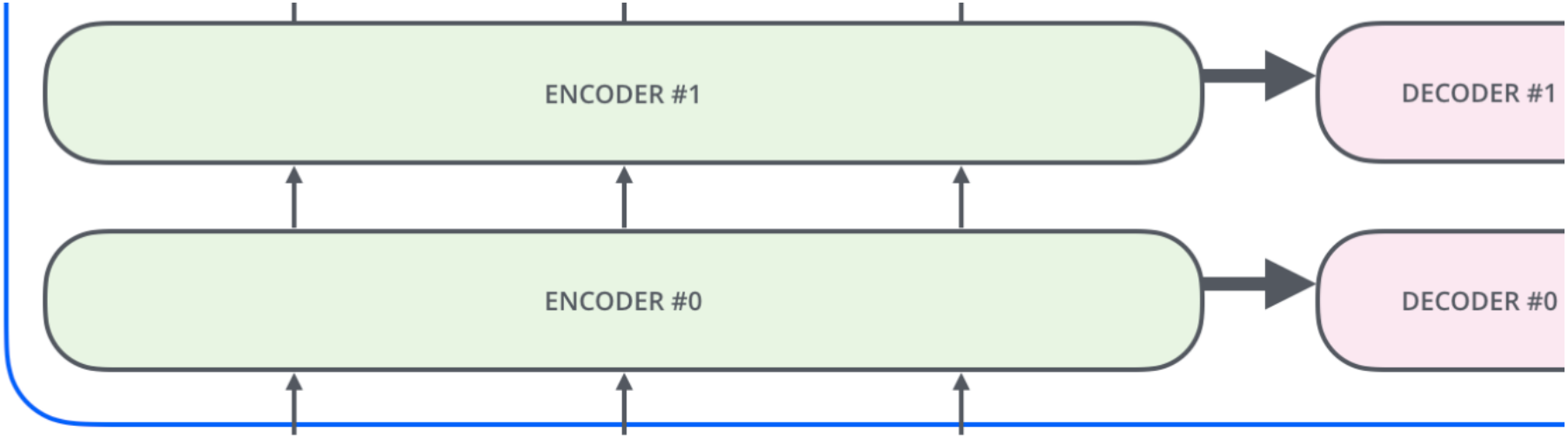


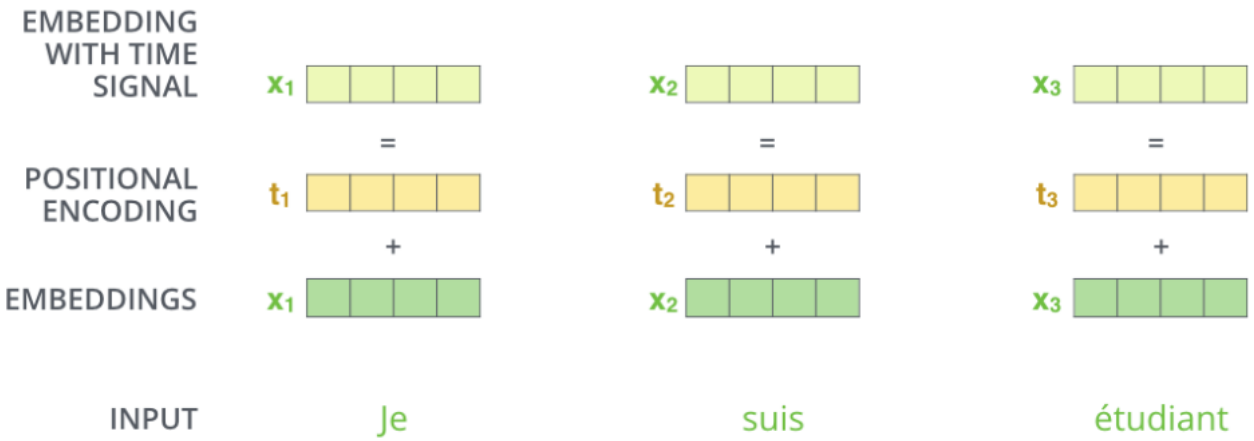
Illustration of self-attention: all heads

- all the attention heads in one picture are harder to interpret



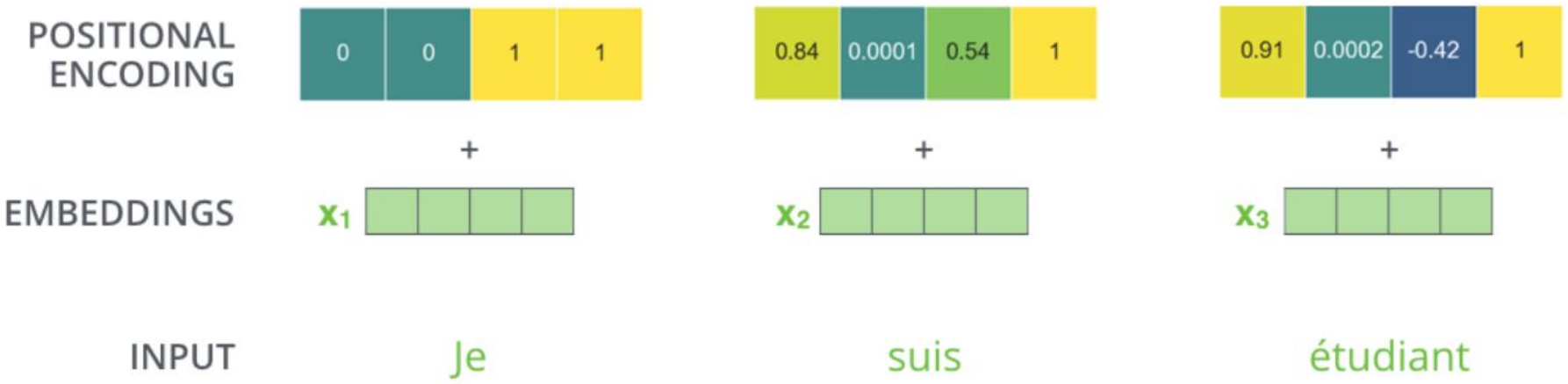
Adding position encoding



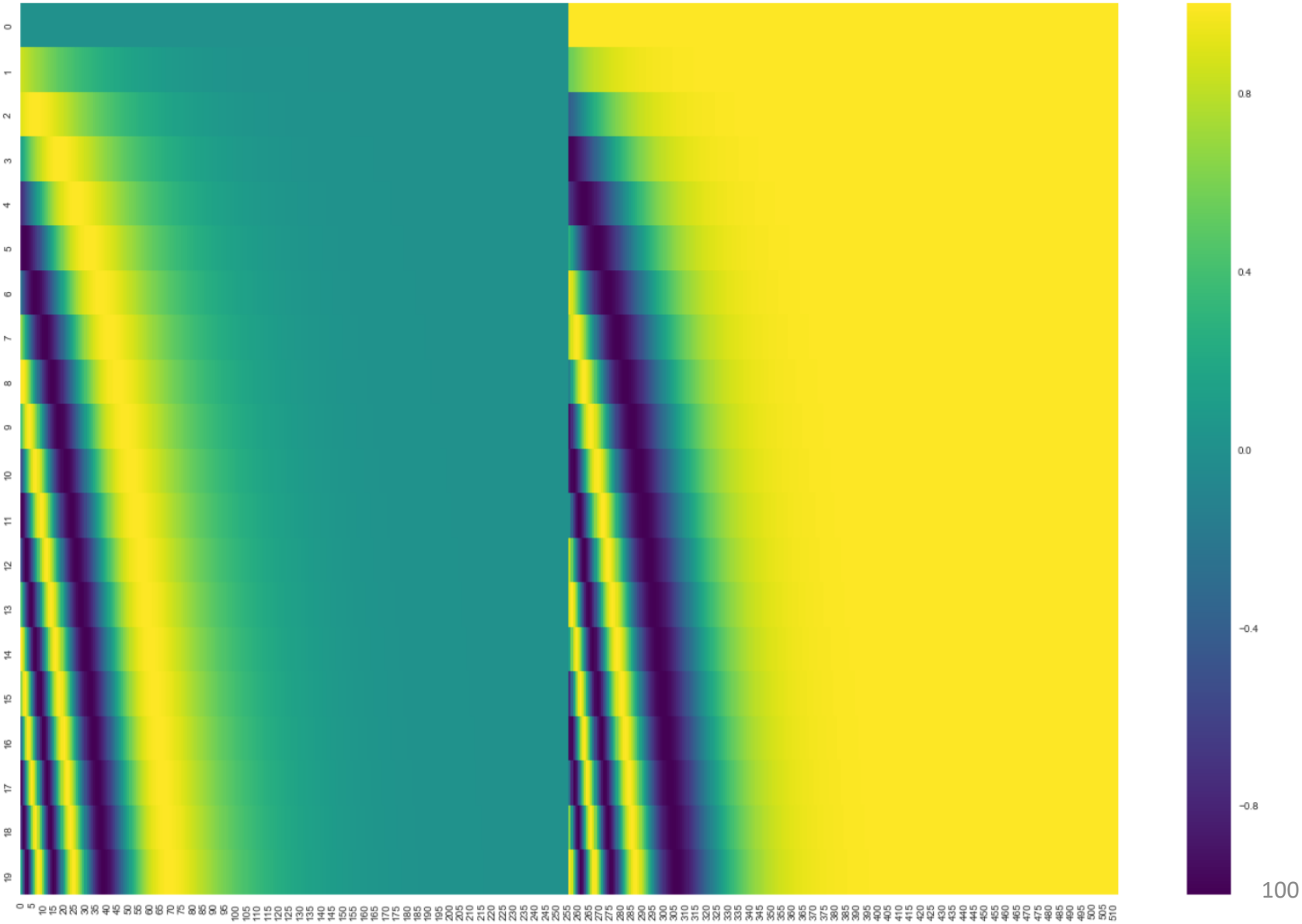


Example: encoding position

- the values of positional encoding vectors follow a specific pattern.



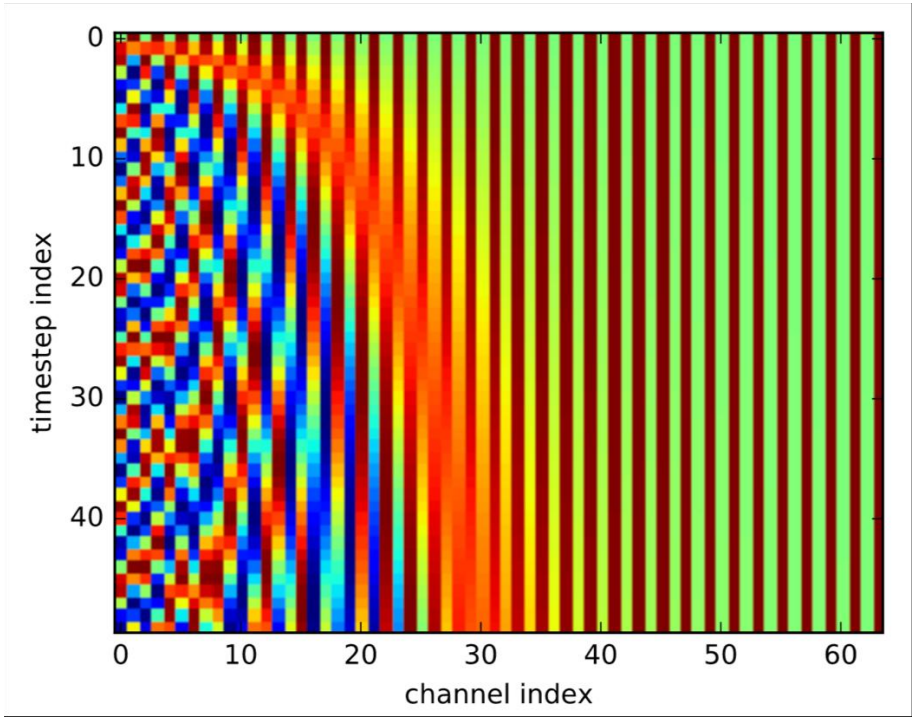
Example of positional encoding



Another positional encoding

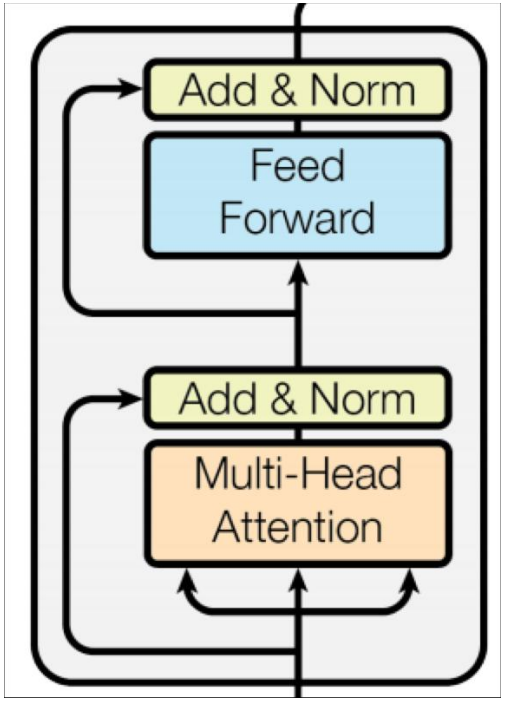
$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{\text{model}}})$$

$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{\text{model}}})$$

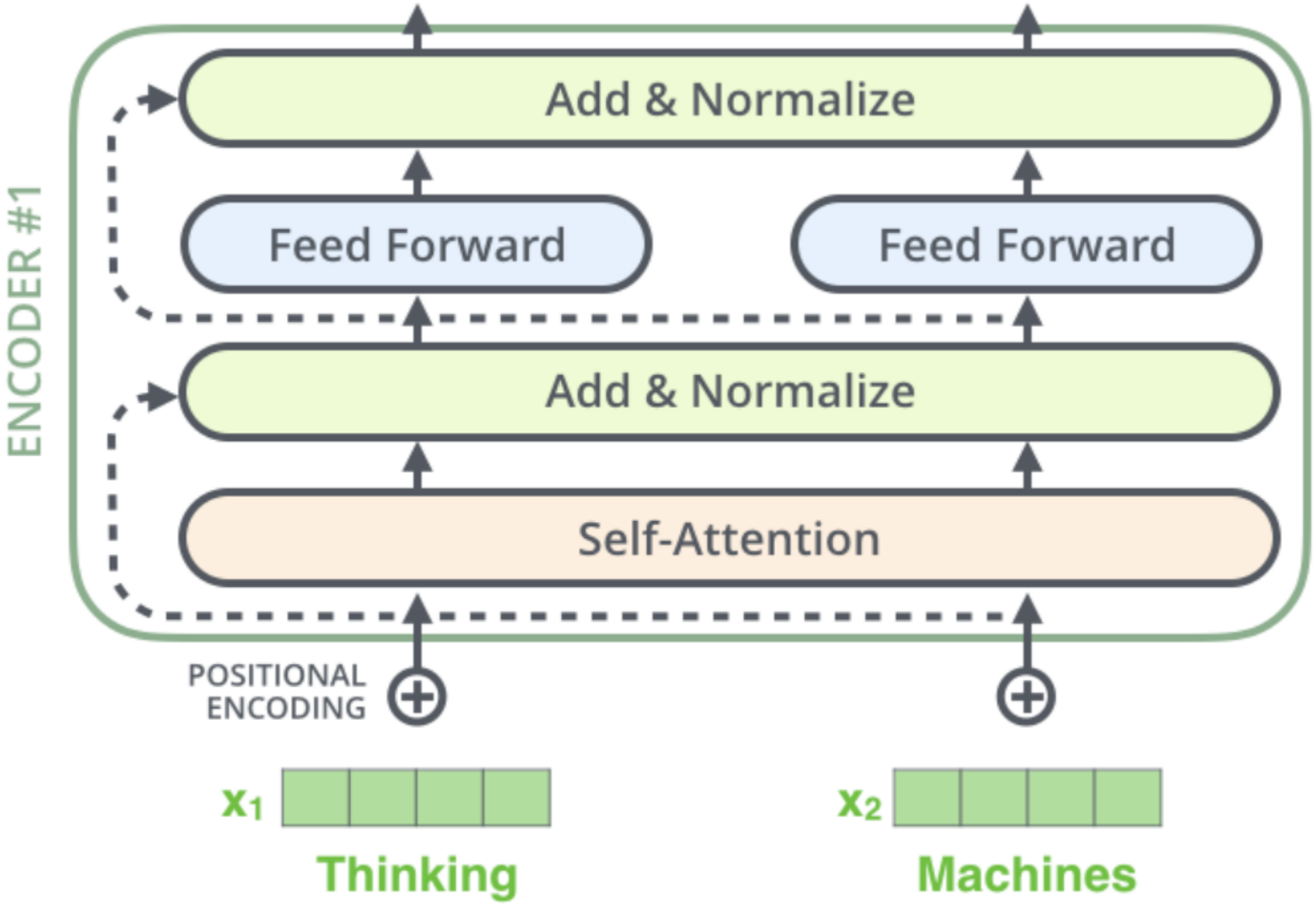


Encoder blocks

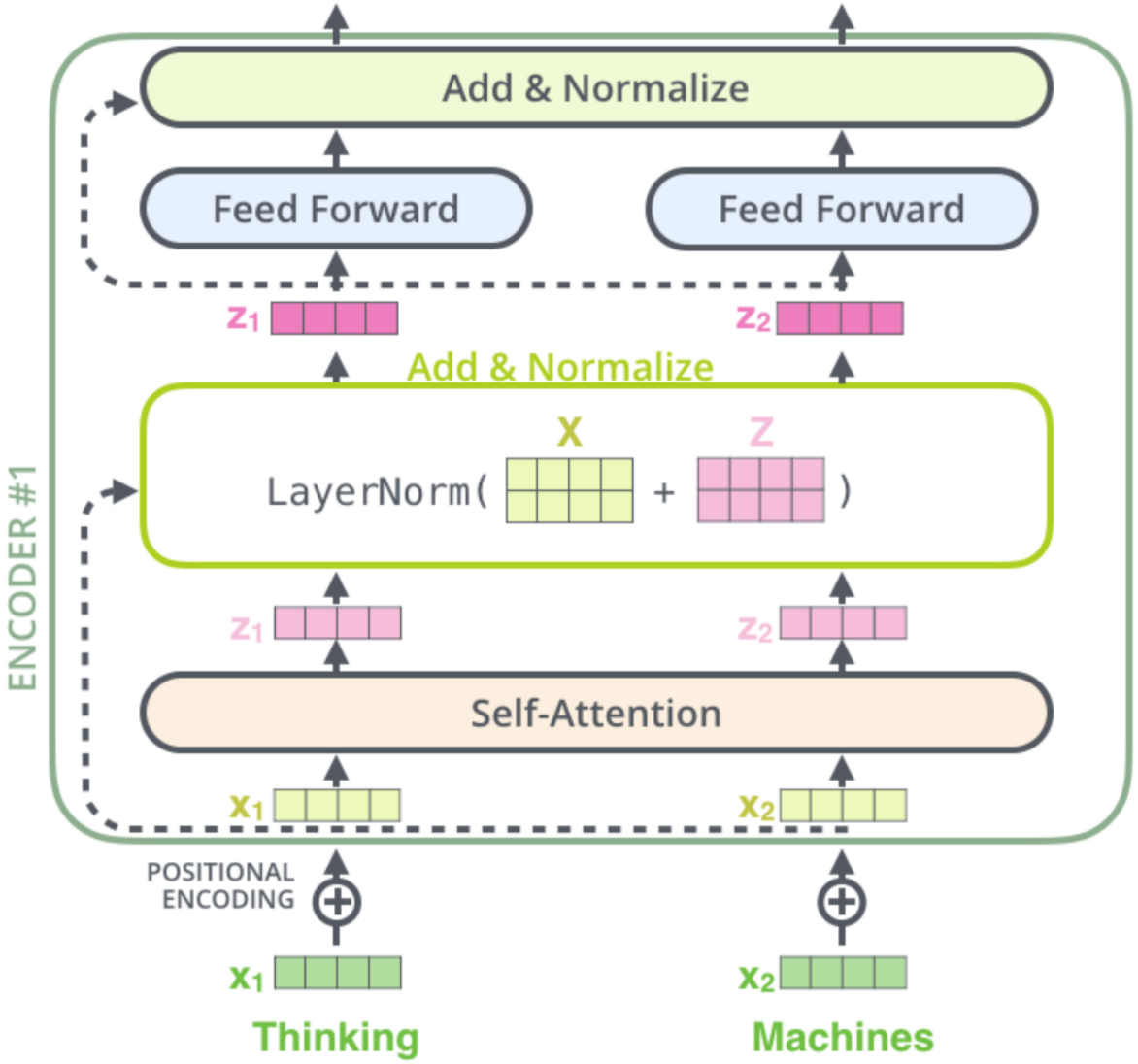
- Each block has two “sublayers”
  - Multihead attention
  - 2-layer feed-forward neural network (with ReLU)
- Each of these two steps also has a residual (short-circuit) connection and LayerNorm, i.e.:
  - LayerNorm(x + Sublayer(x))



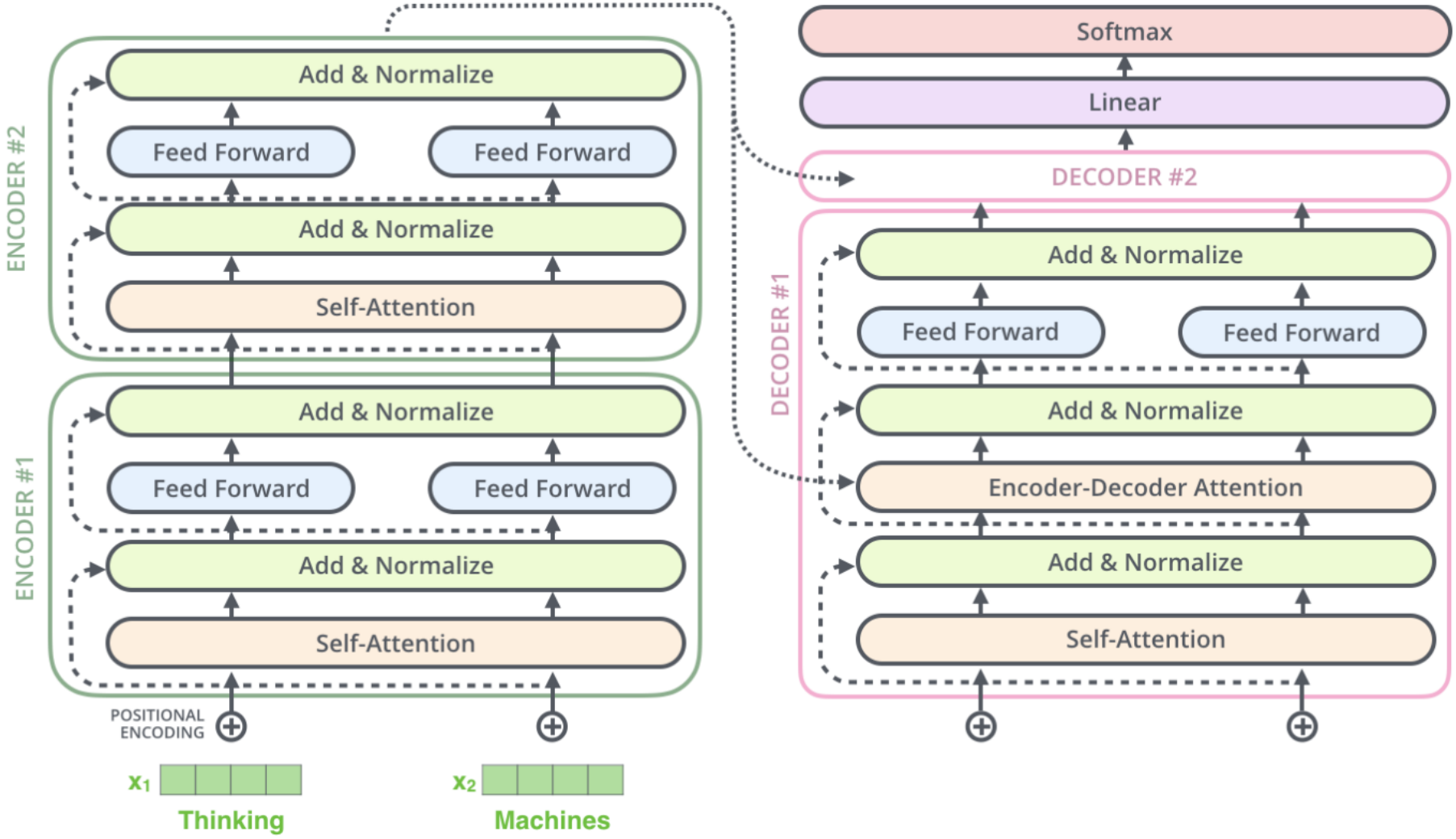
Architecture with residual connection – top level view



Architecture with residual connection – example

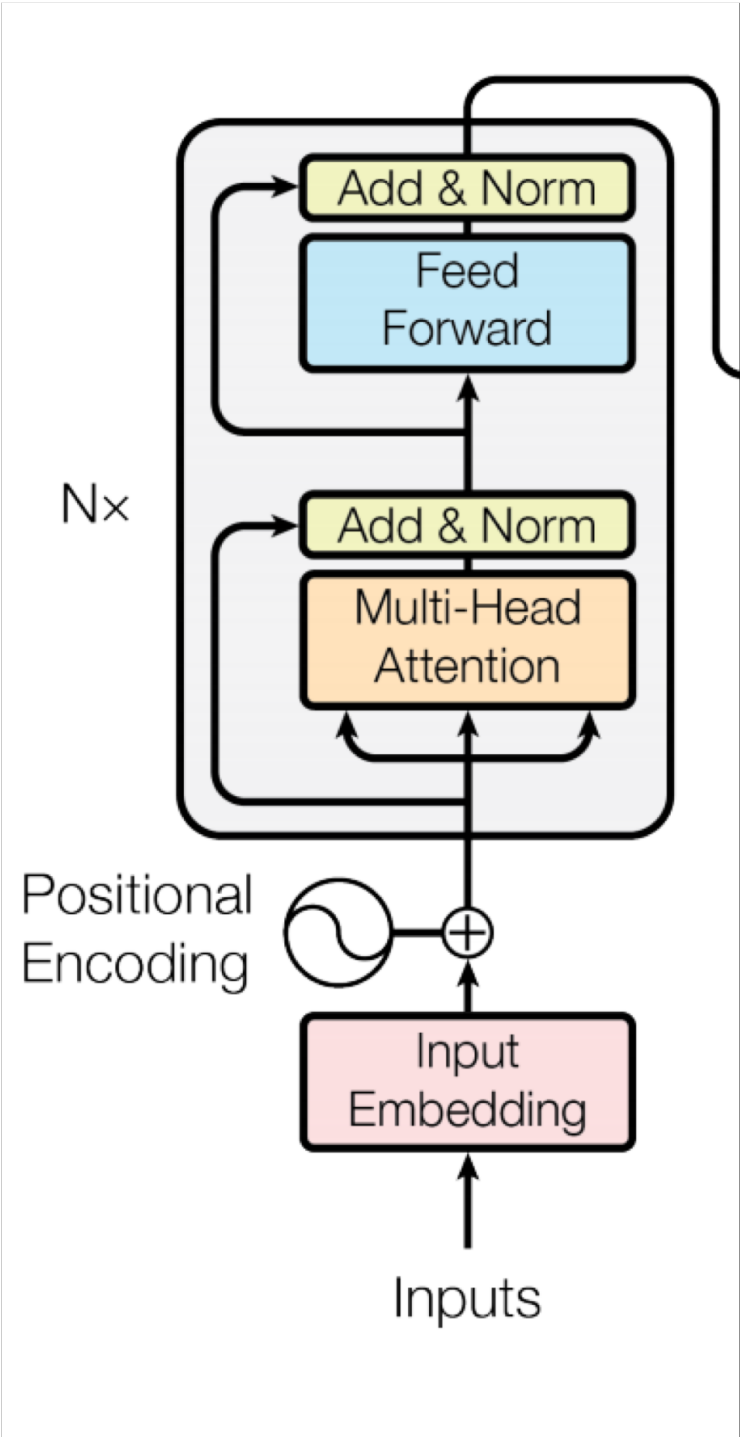


Example: 2 stacked transformer



Complete encoder

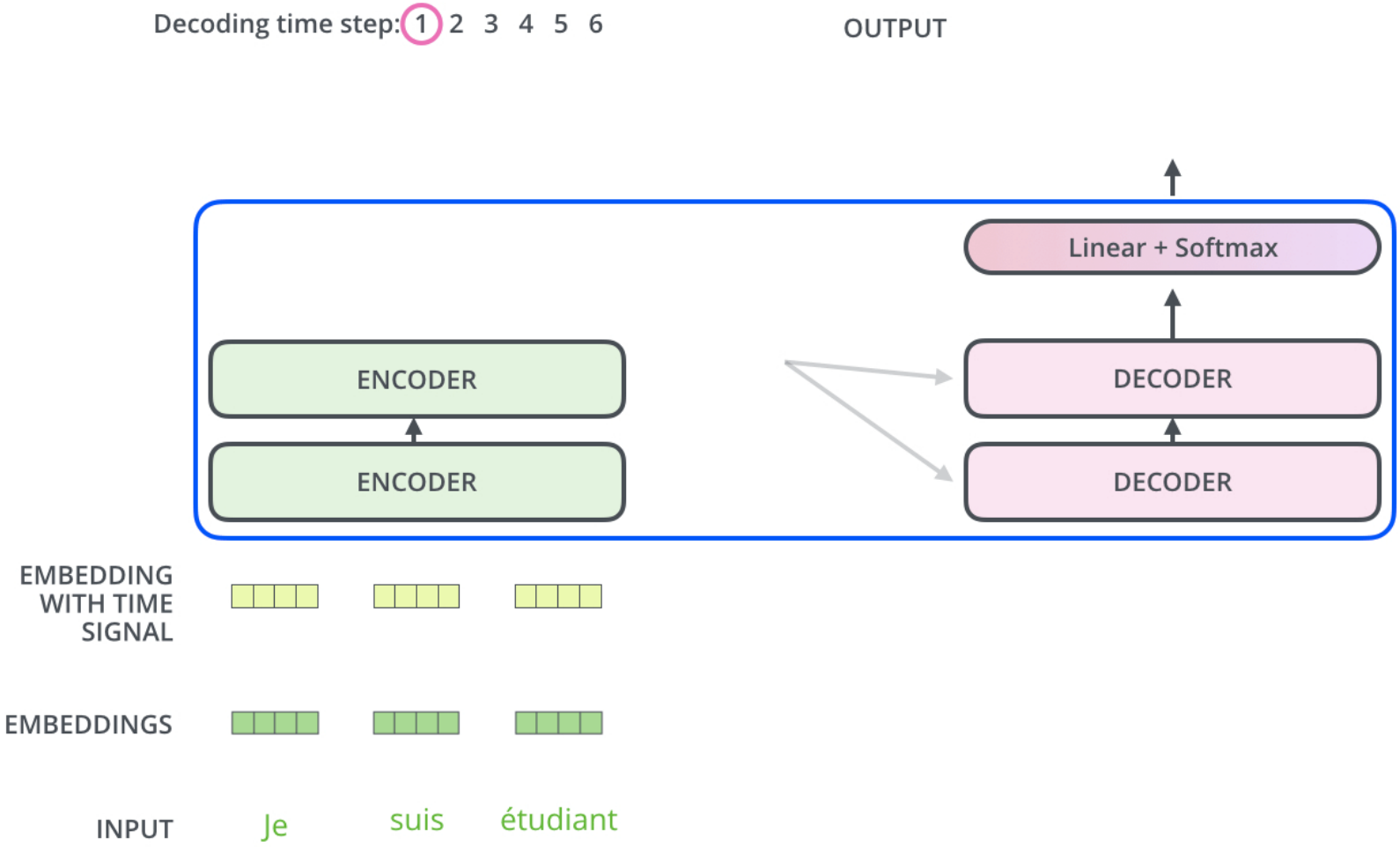
- each encoder block is repeated several times, e.g., 6 times



Decoder

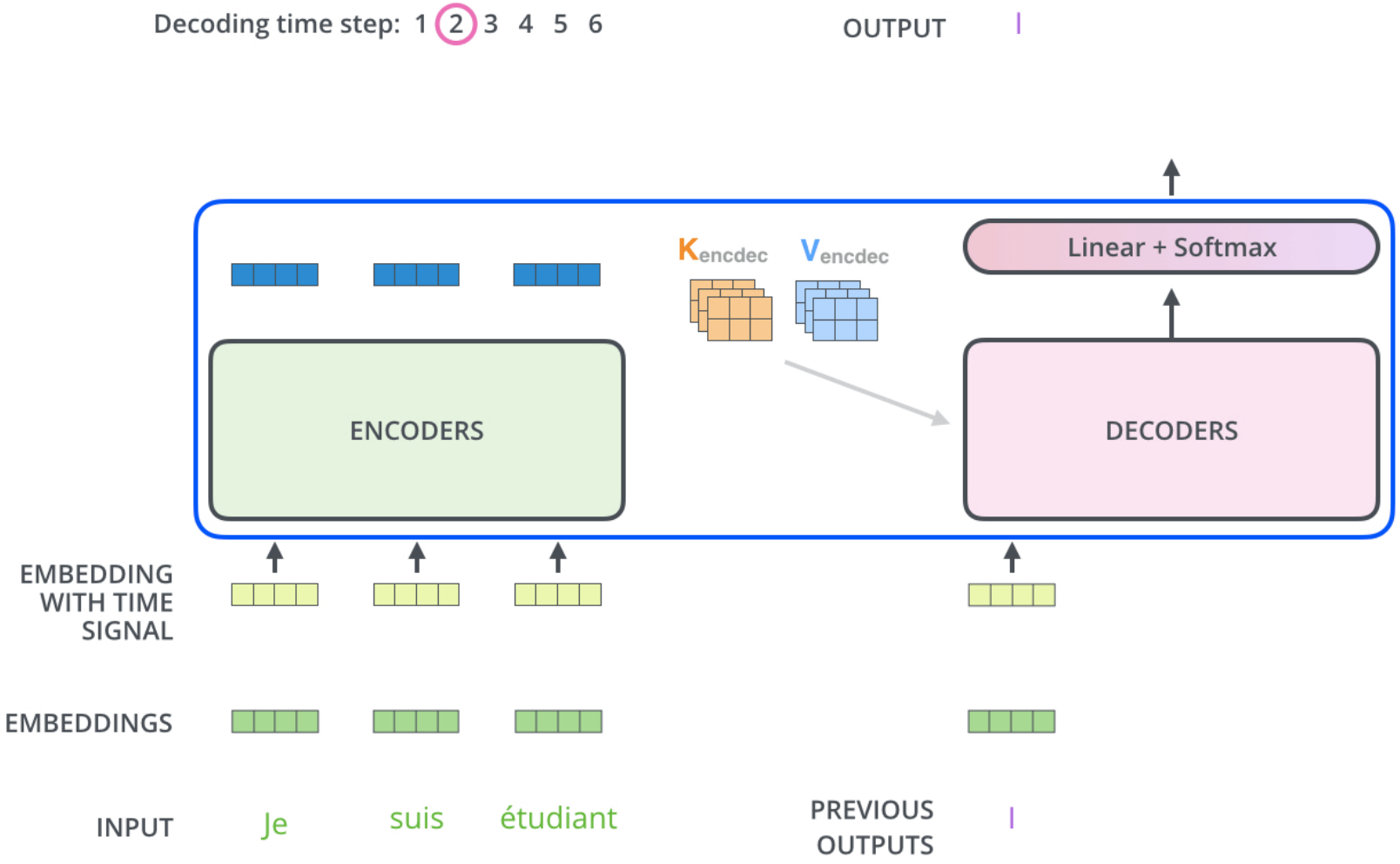
- Decoders have the same components as encoders
- An encoder starts by processing the input sequence.
- The output of the top encoder is transformed into a set of attention vectors K and V.
- These are used by each decoder in its “encoder-decoder attention” layer which helps the decoder to focus on appropriate places in the input sequence.

Encoder-decoder in action 1/2



After finishing the encoding phase, we begin the decoding phase. Each step in the decoding phase outputs an element from the output sequence (the English translation sentence in this case).

Encoder-decoder in action 2/2



The steps repeat until a special symbol indicating the end of output is generated. The output of each step is fed to the bottom decoder in the next time step. We add positional encoding to decoder inputs to indicate the position of each word.

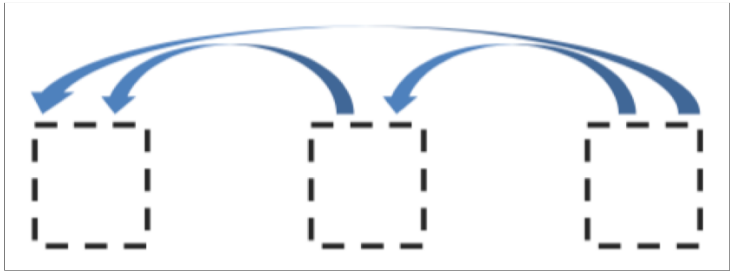
Self-attention and encoder-decoder attention in the decoder

- In the decoder, the self-attention layer is only allowed to attend to itself and earlier positions in the output sequence (to maintain the autoregressive property).
- This is done by masking future positions (setting them to -inf) before the softmax step in the self-attention calculation.

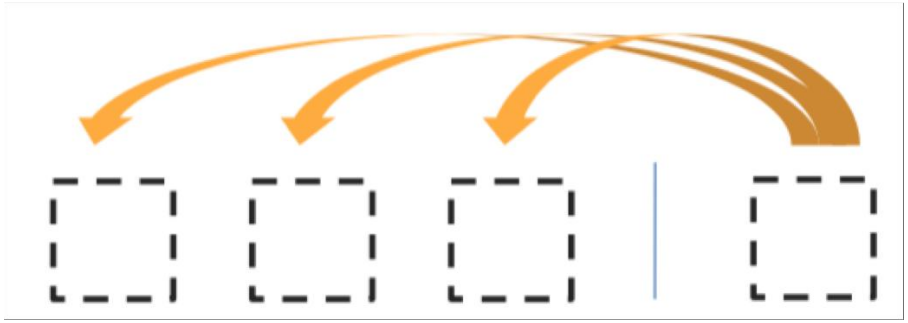
- The “Encoder-Decoder Attention” layer works just like multiheaded self-attention, except it creates its Q (queries) matrix from the layer below it, and takes the K (keys) and V (values) matrix from the output of the encoder stack.

Attentions in the decoder

1. Masked decoder self-attention on previously generated outputs

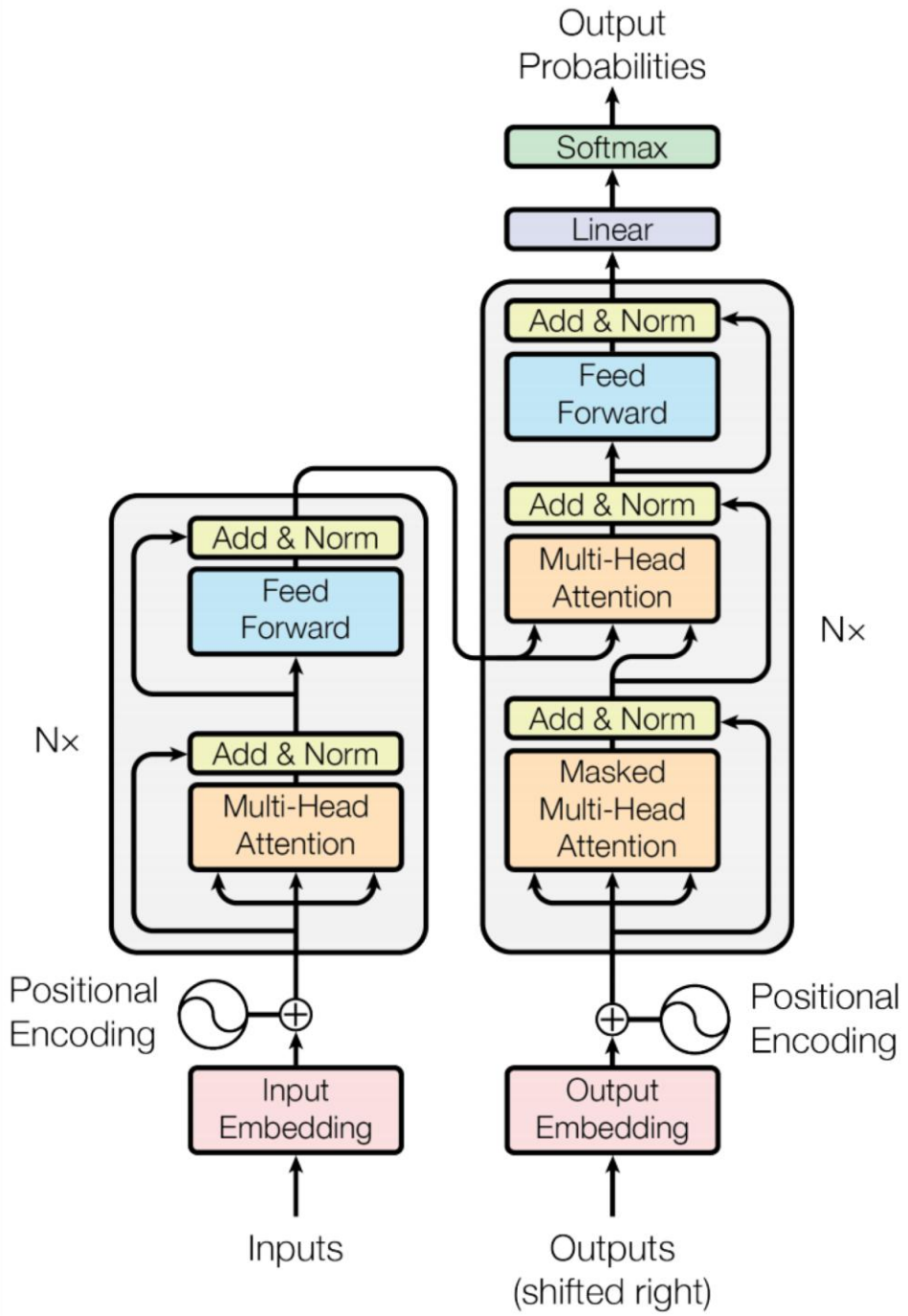


2. Encoder-Decoder Attention, where queries come from previous decoder layer and keys and values come from output of the encoder



Animated workings of attention in transformer

One encoder-decoder block



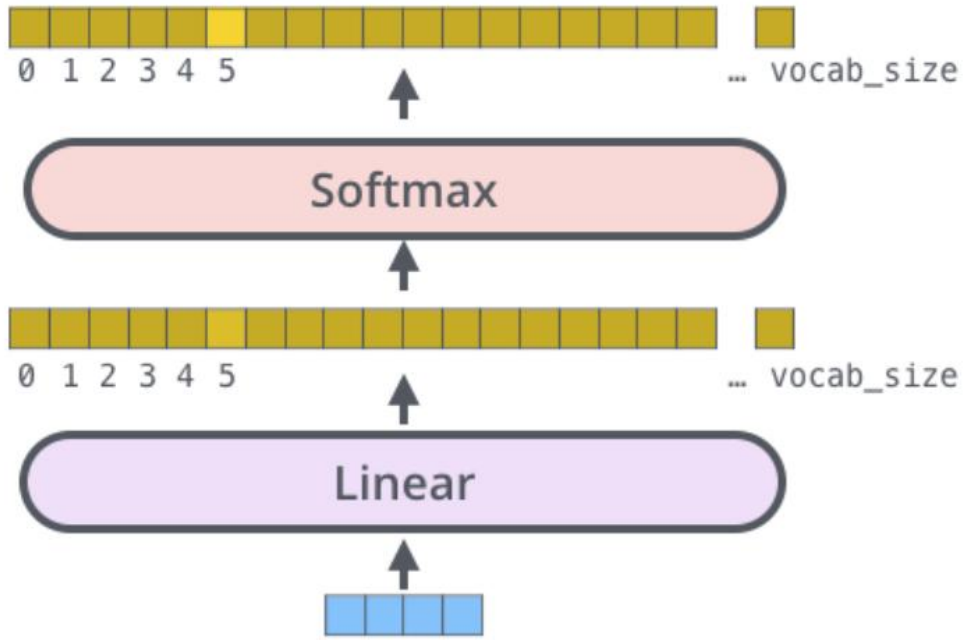
Producing the output words

Which word in our vocabulary is associated with this index?

am

Get the index of the cell with the highest value (**argmax**)

5



Training the transformer

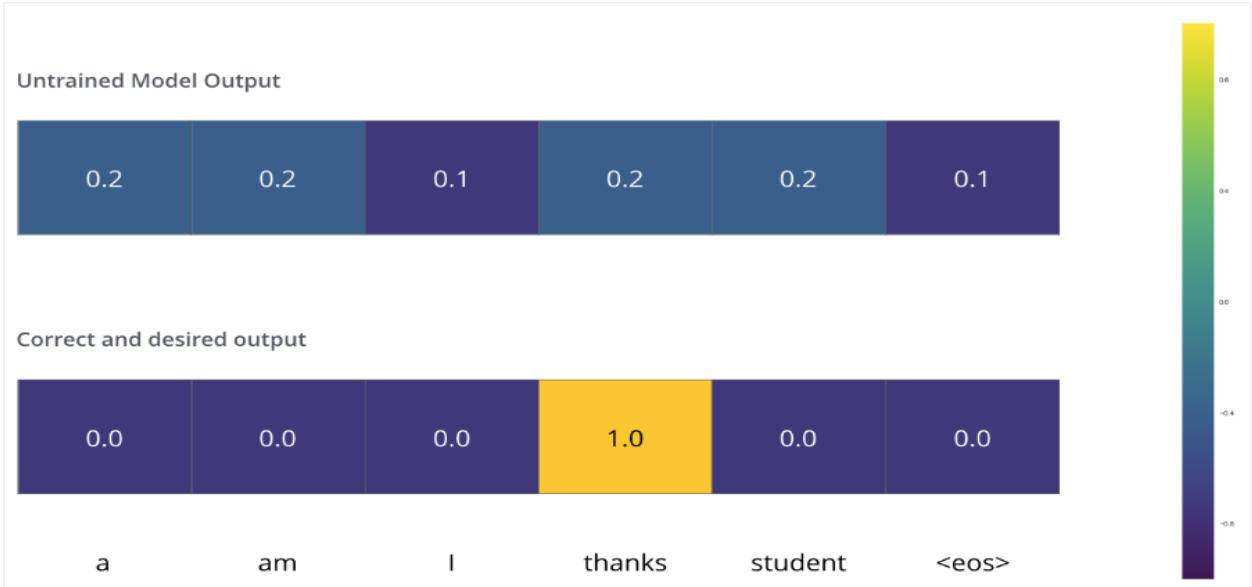
- During training, an untrained model would go through the exactly the same forward pass. But since we are training it on a labeled training dataset, we can compare its output with the actual correct output.
- For illustration, let's assume that our output vocabulary only contains six words(a, am, i, thanks, student, <eos>)
- The input is typically in the order of 10^4 (e.g., 30 000)

Output Vocabulary

WORD	a	am	I	thanks	student	<eos>
INDEX	0	1	2	3	4	5

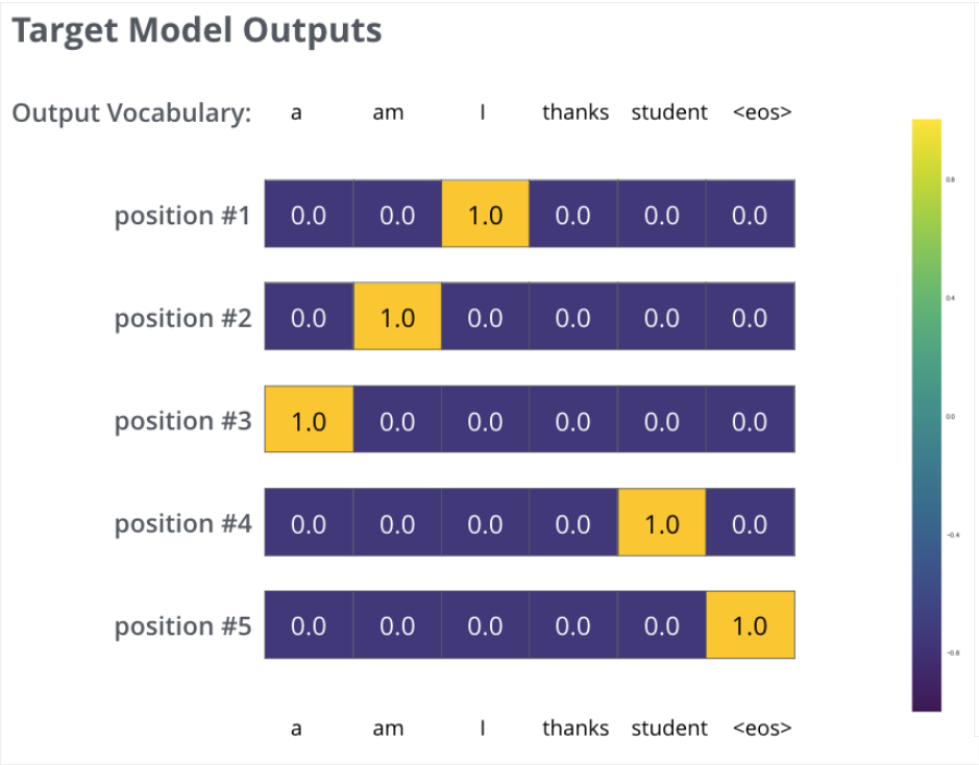
The Loss Function

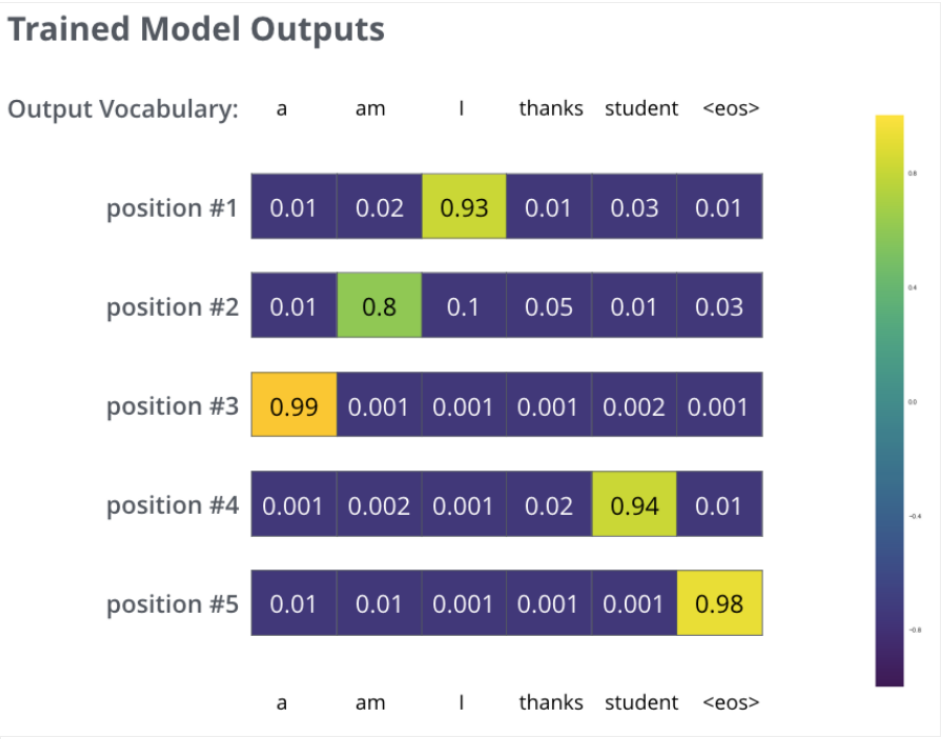
- Evaluates the difference between the true output and the returned output
- Transformer typically uses cross-entropy or Kullback–Leibler divergence.
- The model output is a probability distribution, the true output is 1-hot encoded, e.g.,



Loss evaluation for sequences

- Loss function has to be evaluated for the whole sentence, not just a single word.
- Transformers use greedy decoding or beam search.





Three flavours of transformers

- encoder only (BERT)
- encoder-decoder (machine translation, T5)
- decoder only (GPT)

BERT

- Combines several tasks
- Predicts masked words in a sentence
- Also predicts the order of sentences: is sentence A followed by sentence B or not?
- Combines several hidden layers of the network
- Uses transformer neural architecture, only the encoder part
- Uses several fine-tuned parameters
- Multilingual variant supports 104 languages by training on Wikipedia
- Many publicly available variants.

Devlin J, Chang MW, Lee K, Toutanova K (2019) BERT: Pre-training of deep bidirectional transformers for language understanding. In: Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers), pp 4171–4186

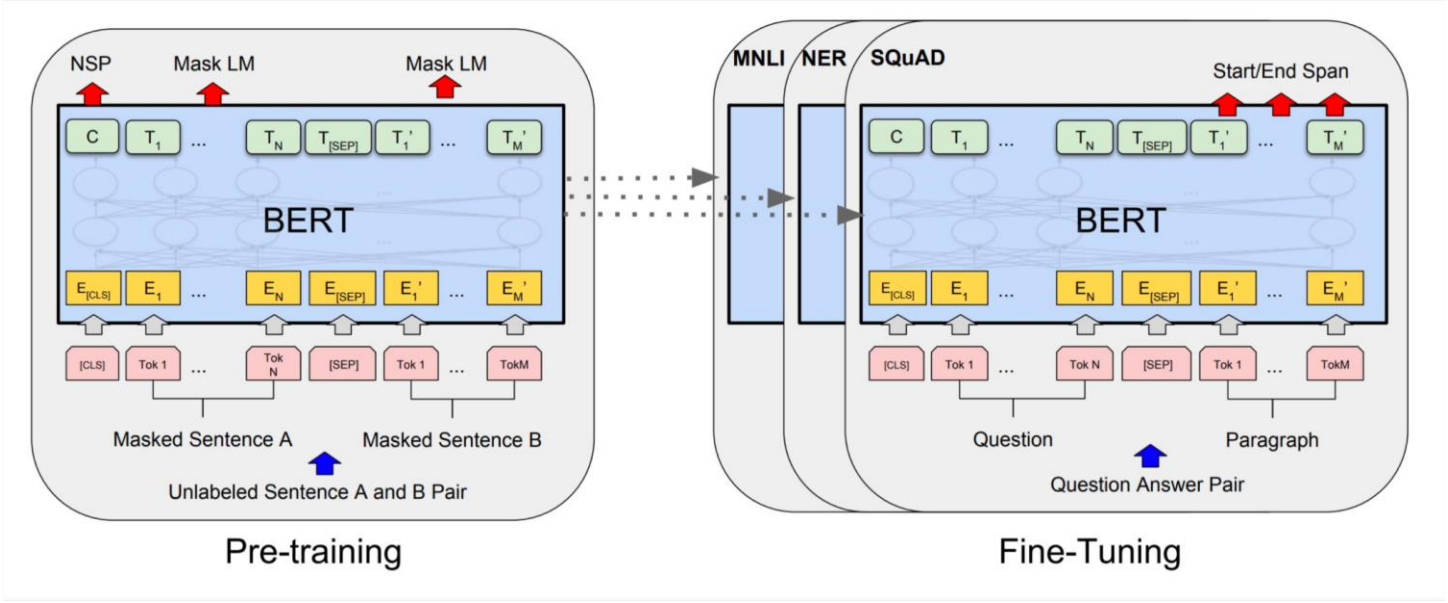
Many BERT-like embeddings

- XLM-R was trained on 2.5 TB of texts in 100 languages
- For Slovene: fastText (a variant of word2vec), ELMo, SloBERTa
- Trilingual BERT – CroSloEngual
- On HuggingFace
- Hundreds of papers investigating BERT-like models in major ML & NLP conferences

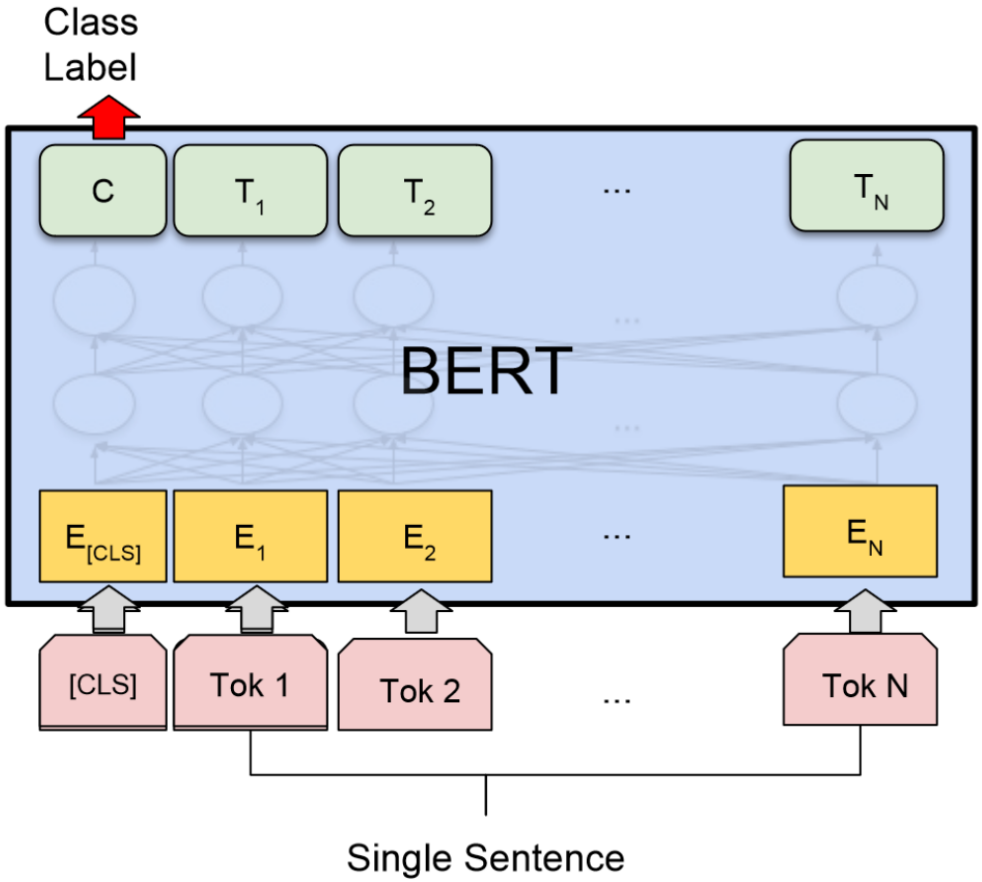
Ulčar, Matej and Marko Robnik-Šikonja. FinEst BERT and CroSloEngual BERT: less is more in multilingual models.
In Proceedings of Text, Speech, and Dialogue, TSD2020, 2020.

Use of BERT

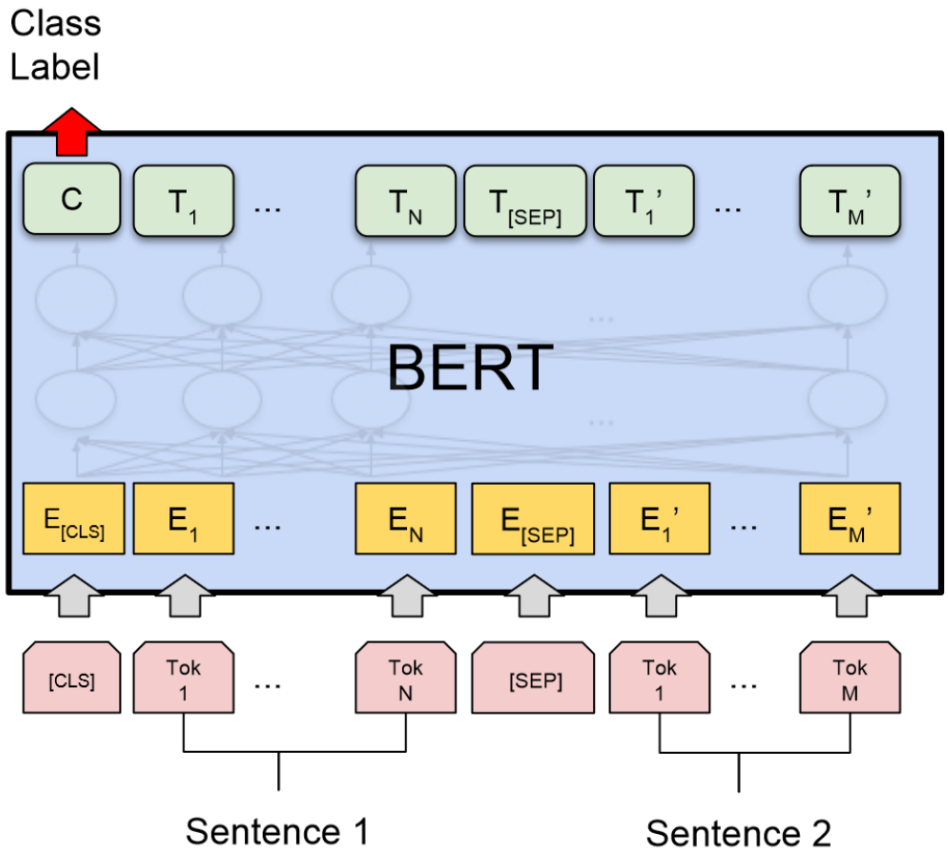
- train a classifier built on the top layer for each task that you fine tune for, e.g., Q&A, NER, inference
- achieves state-of-the-art results for many tasks
- GLUE and SuperGLUE tasks for NLI



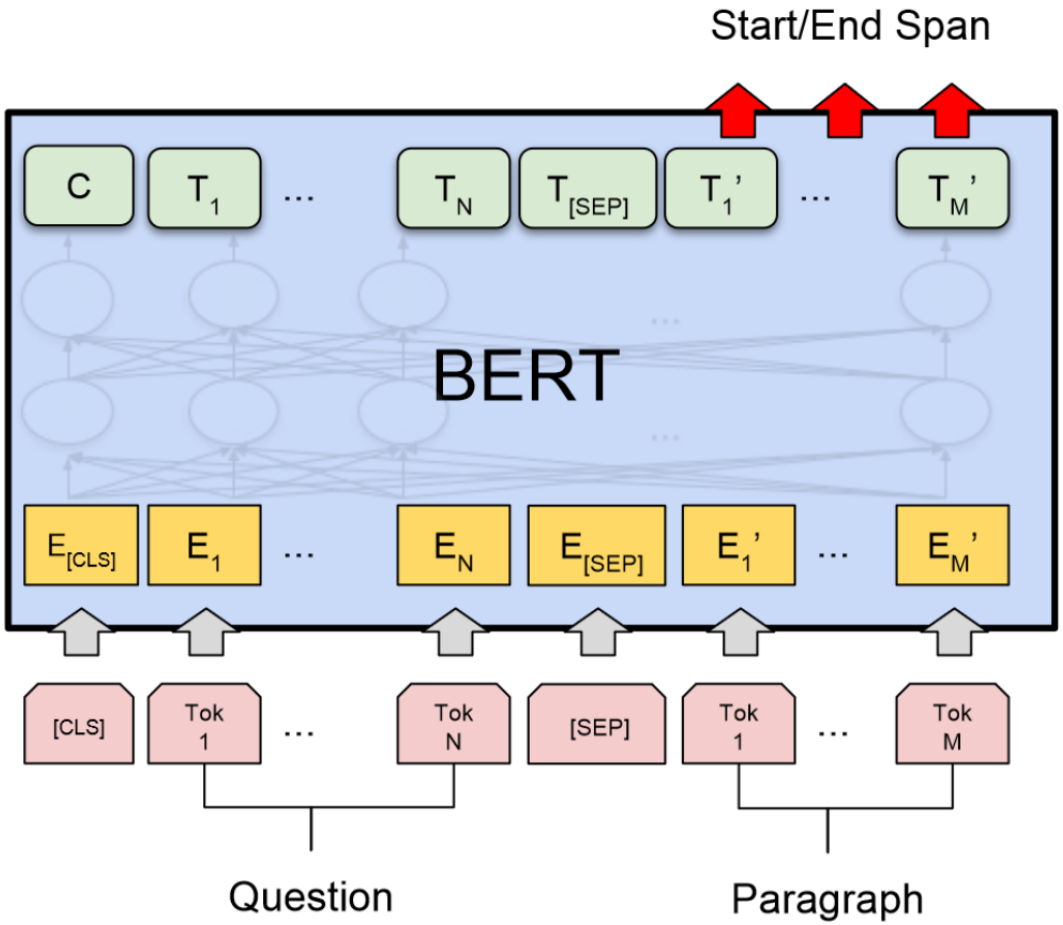
Sentence classification using BERT – sentiment, grammatical correctness



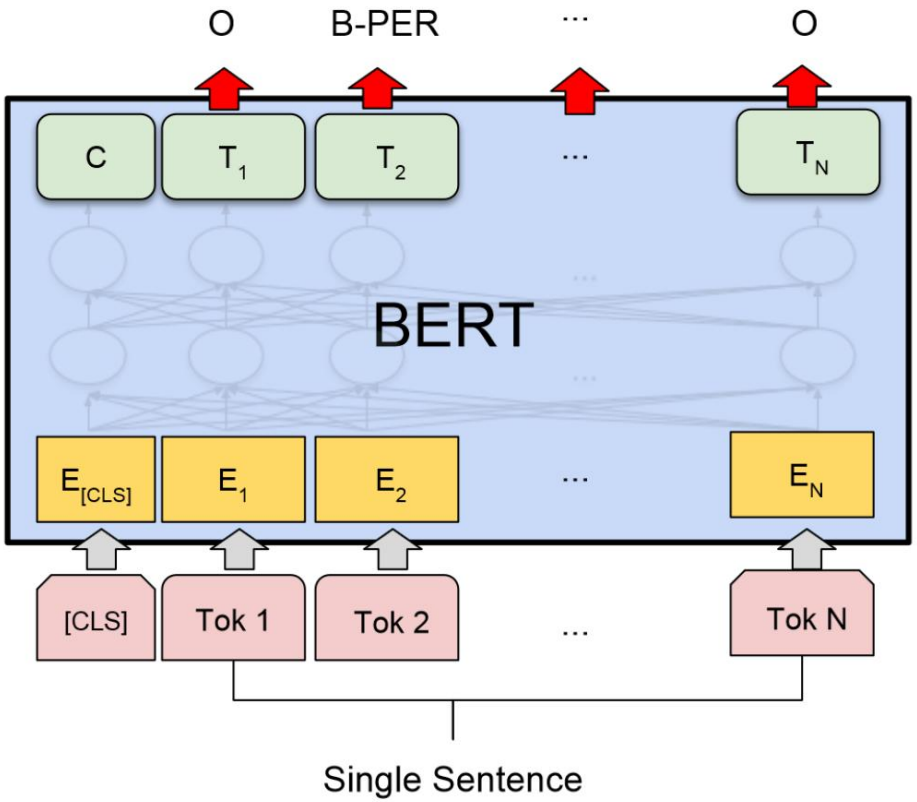
Two sentence classification using BERT-inference



Questions and answers with BERT

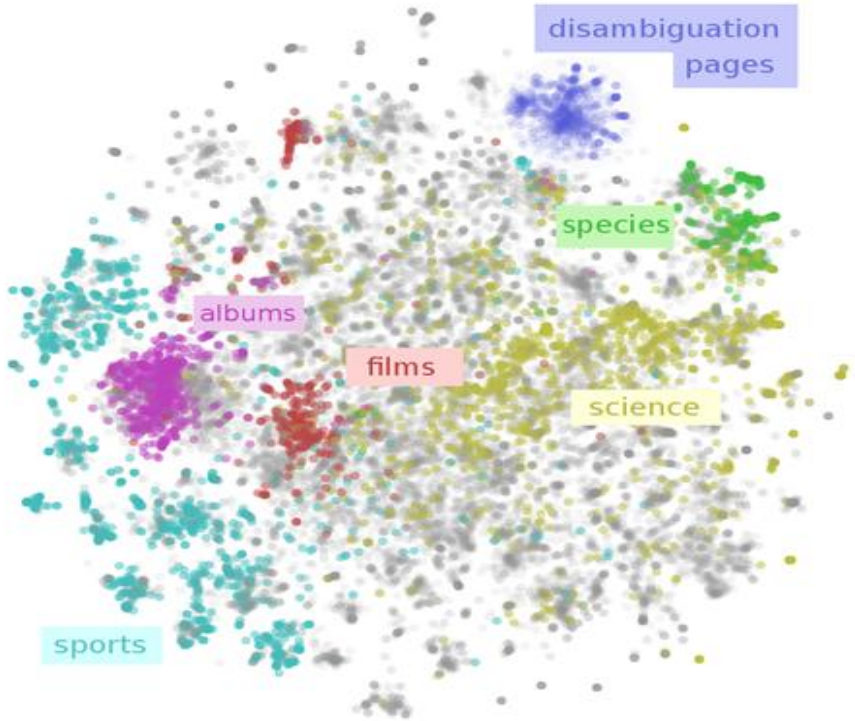


Sentence tagging with BERT-
NER, POS tagging, SRL



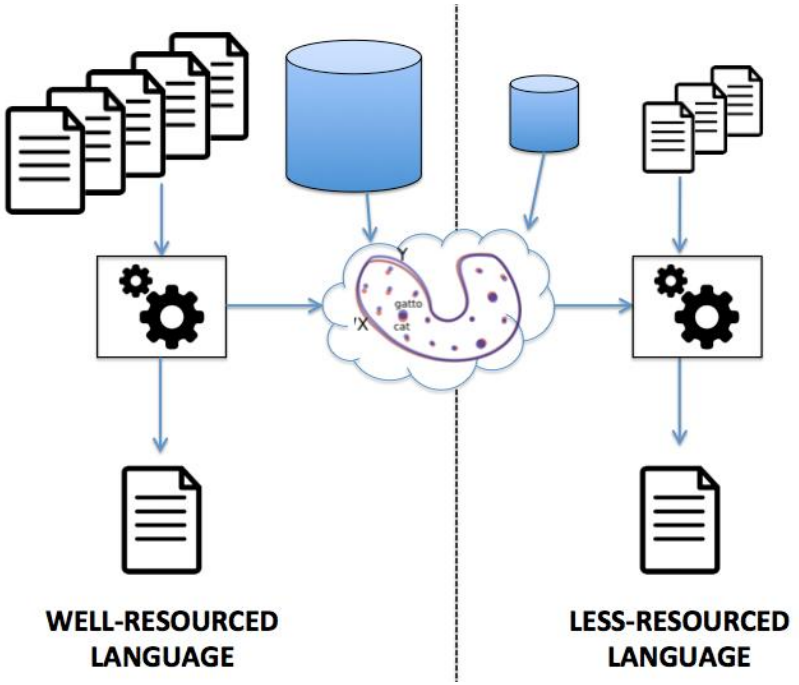
Cross-lingual embeddings

- mostly, embeddings are trained on monolingual resources
- words of one language form a cloud in high-dimensional space
- clouds for different languages can be aligned
  - $W S \approx E$ or
  - $W_1 S \approx W_2 E$



Cross-lingual model transfer based on embeddings

- Transfer of tools trained on mono-lingual resources



mostly not used anymore, superseded by multilingual LLMs

University of Ljubljana
Faculty of Computer and Information Science

Vocabulars in LLMs

- Tokenization depends on the dictionary
- The dictionary is constructed statistically (SentencePiece algorithm)

- Sentence: “Letenje je bilo predmet precej starodavnih zgodb.”
- SloBERTa:

'▁Le', 'tenje', '▁je', '▁bilo', '▁predmet', '▁precej', '▁staroda', 'vnih', '▁zgodb', '.’

- mBERT:

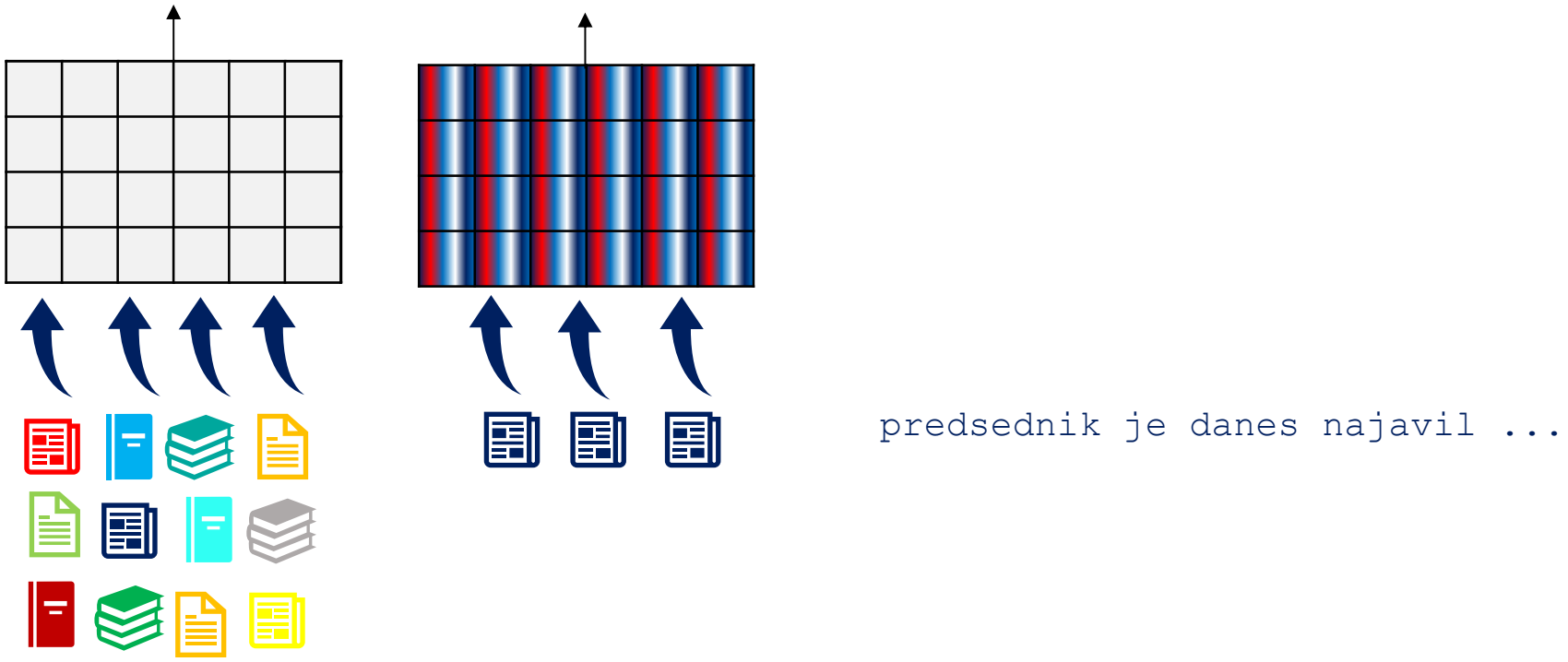
'Let', '##en', '##je', 'je', 'bilo', 'pred', '##met', 'pre', '##cej', 'star', '##oda', '##vnih', 'z', '##go', '##d', '##b', '.'

Ulčar, M., & Robnik-Šikonja, M. (2021) Training dataset and dictionary sizes matter in BERT models: The case of Baltic languages. In *International Conference on Analysis of Images, Social Networks and Texts* (pp. 162-172)

Multilingual LLMs

- Pretrained on multiple languages simultaneously
- multilingual BERT supports 104 languages by training on Wikipedia
- XLM-R was trained on 2.5 TB of texts
- allow cross-lingual transfer
- often solve the problem of insufficient training resources for less-resourced languages

Using multilingual models



Zero-shot transfer and few-shot transfer

What LLMs learn?

- We would like to travel to [MASK], ki je najlepši otok v Mediteranu.

SloBERTa: ..., Slovenija, I, Koper, Slovenia

CSE-BERT: Hvar, Rab, Cres, Malta, Brač

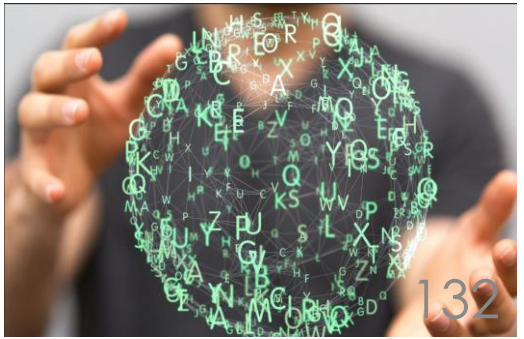
XLM-R: Mallorca, Tenerife, otok, Ibiza, Zadar

mBERT: Ibiza, Gibraltar, Tenerife, Mediterranean, Madeira

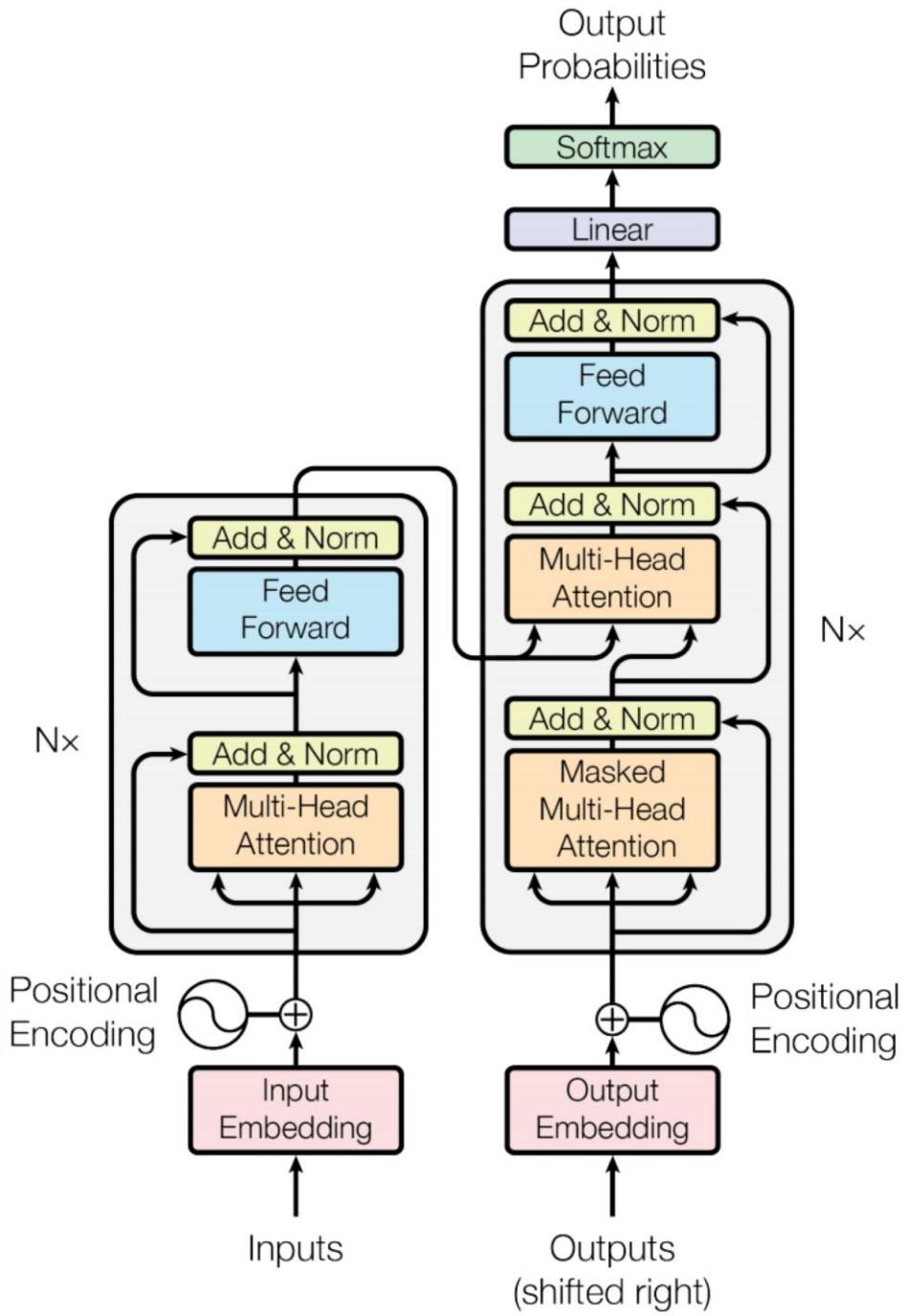
BERT (en): Belgrade, Italy, Serbia, Prague, Sarajevo

Embed all the things!

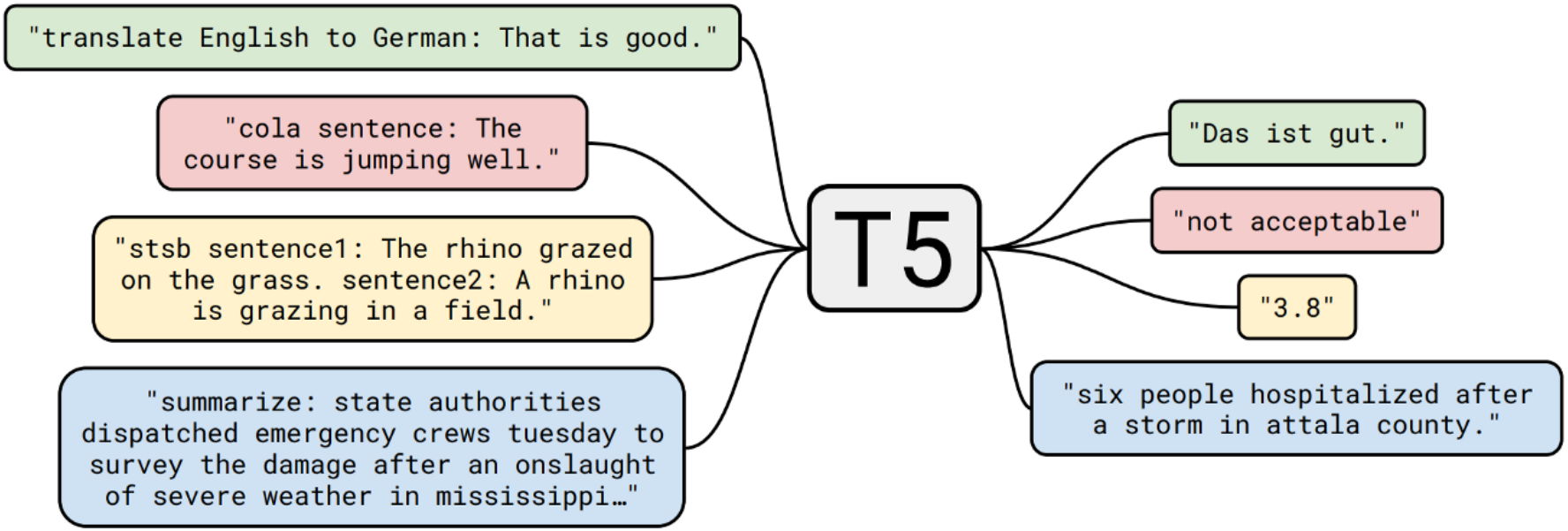
- Neural networks require numeric input
- Embedding shall preserve relations from the original space
- Representation learning is a crucial topic in nowadays machine learning
- Lots of applications whenever enough data is available to learn the representation
- In text, BERT-like models dominate for embeddings
- Similar ideas applied to texts, speech, graphs, electronic health records, relational data, time series, etc.



Transformer



T5 (Text-To-Text Transfer Transformer) models



Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y, Li, W. & Liu, P. J. (2020). Exploring the limits of transfer learning with a unified text-to-text transformer. *J. Mach. Learn. Res.*, *21*(140), 1-67.

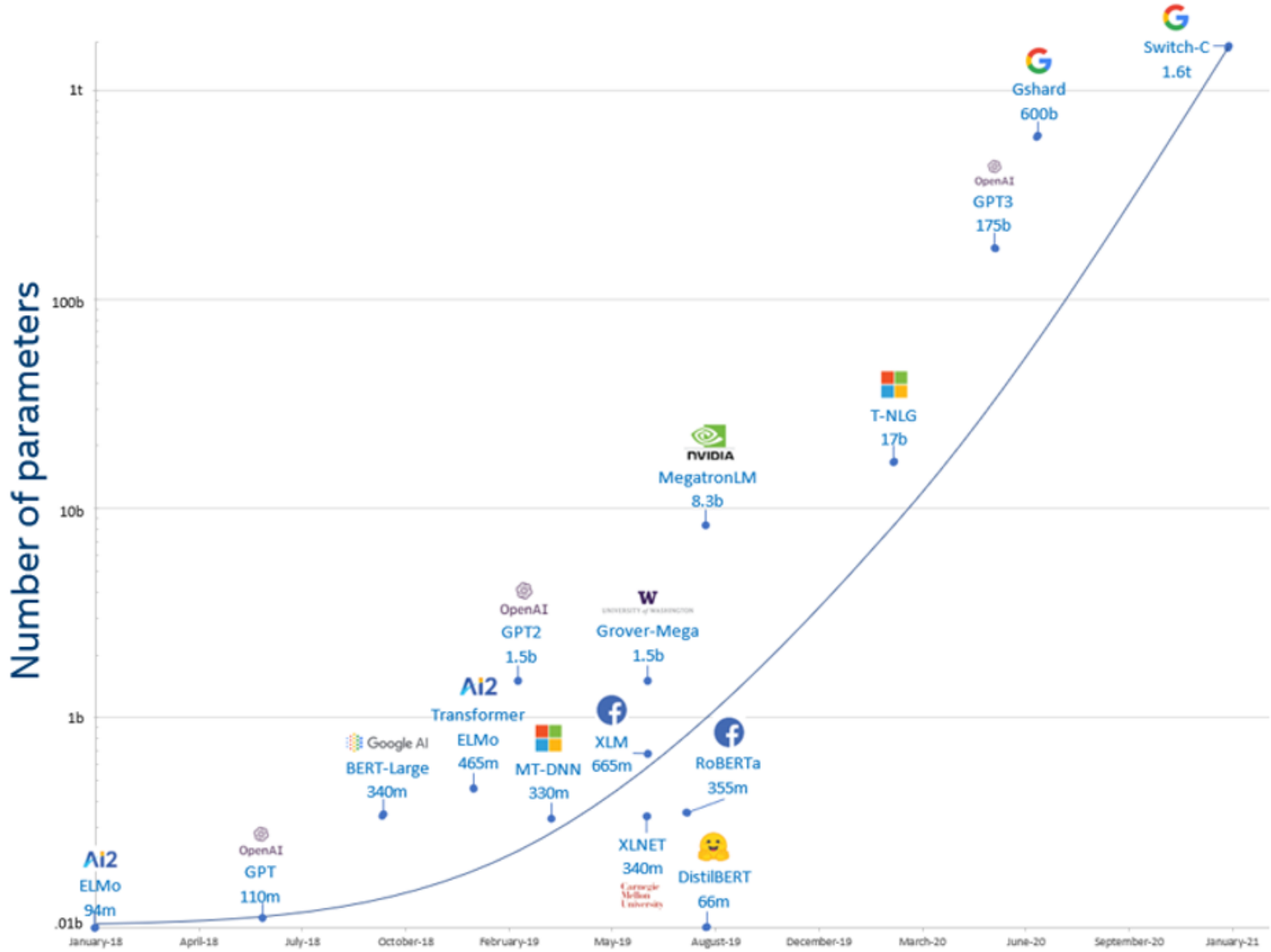
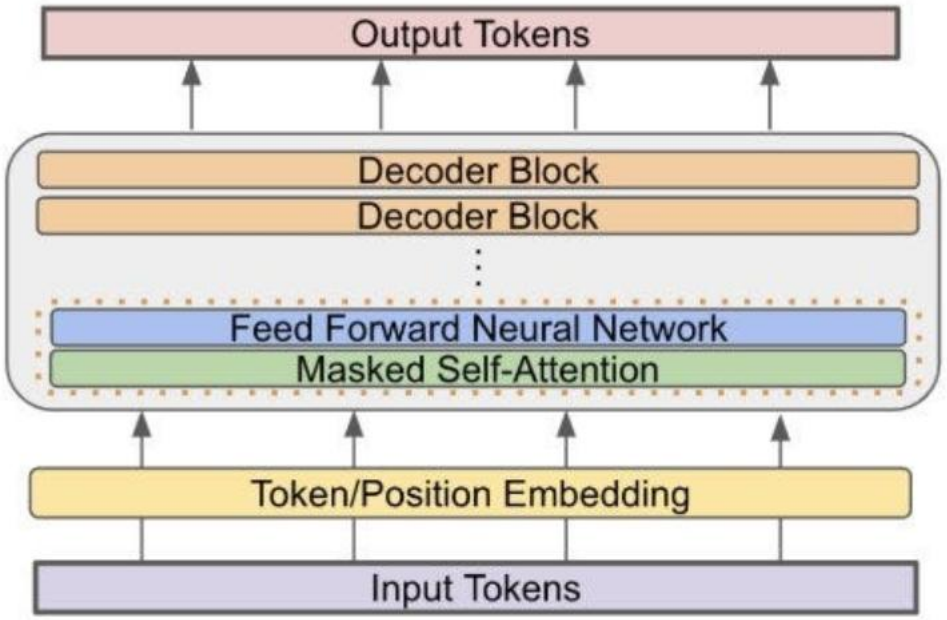


Figure 1: Exponential growth of number of parameters in DL models

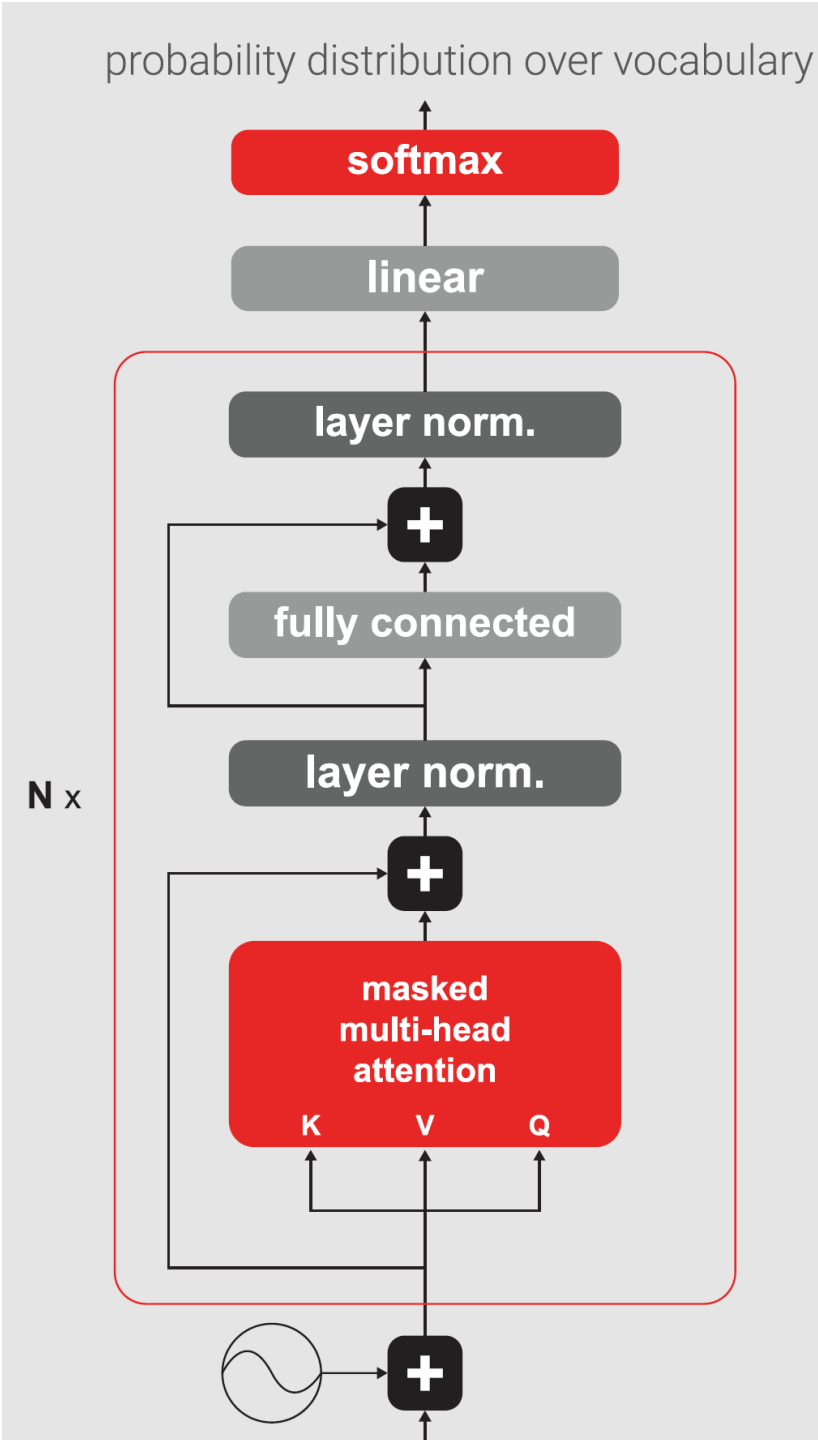
Decoder only models

- GPT, GPT-2, GPT-3, ChatGPT, GPT-4
- LLaMA, LLaMA-2, LLaMa-3
- MPT, Falcon
- Mistral and Mixtral
- OPT, Bloom
- Gemma and Gemini
- Olmo
- GaMS

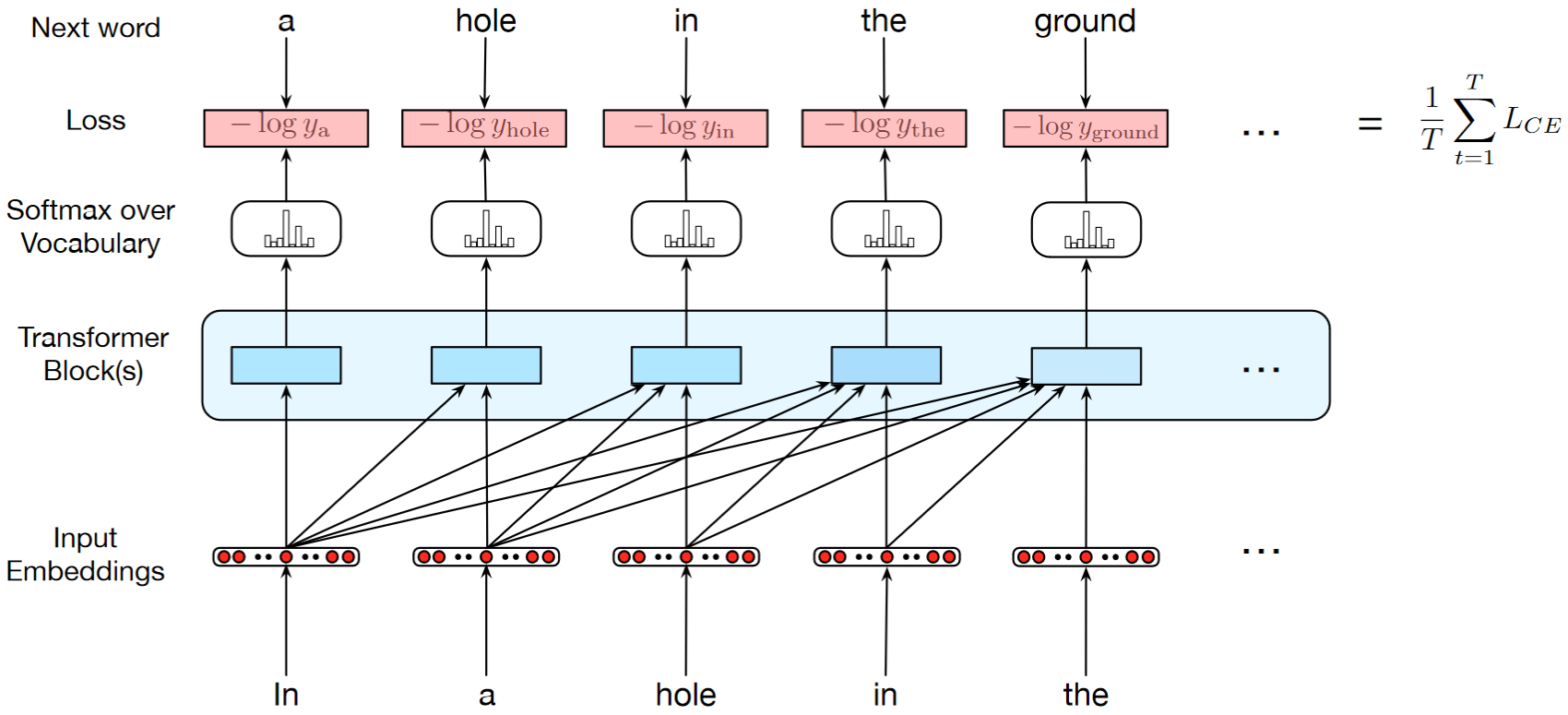


GPT family

- GPT: Generative Pre-trained Transformers
- use only the decoder part of transformer
- pretrained for language modeling (predicting the next word given the context)
- Potential shortcoming: unidirectional, does not incorporate bidirectionality
- “What are those?” he said while looking at my crocs.



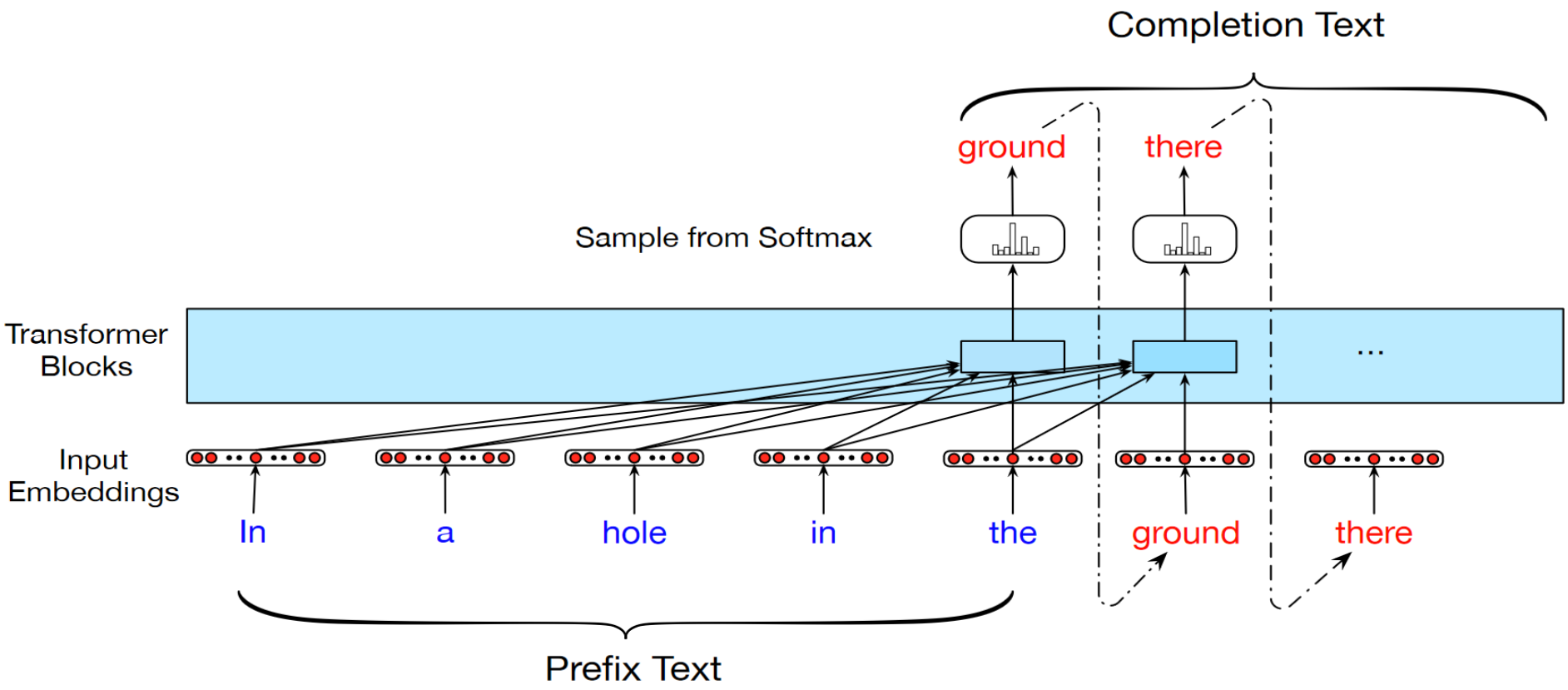
Transformer as a language model



- Can be computed in parallel

Autoregressive generators

- priming the generator with the context



- can be used also in summarization, QA and other generative tasks

GPT-2 and GPT-3

- few architectural changes, layer norm now applied to input of each subblock
- GPT-3 also uses some sparse attention layers
- more data, larger batch sizes (GPT-3 uses batch size of 3.2M)
- the models are scaled:

GPT-2:
48 layers, 25 heads
$d_m = 1600$, $d = 64$
context size = 1024
~ 1.5B parameters

GPT-3:
96 layers, 96 heads
$d_m = 12288$, $d = 128$
context size = 2048
~ 175B parameters

[1] Radford et al.: Language Models are Unsupervised Multitask Learners, 2019.
[2] Brown et al.: Language Models are Few-Shot Learners, 2020.

- see demos at https://transformer.huggingface.co/

In-context learning

- GPT-2 and GPT-3 were the first models where one can ditch the “pre-train and fine-tune” training paradigm of GPT;
- GPT-2 explores unsupervised zero-shot learning, whereas in GPT-3 the authors expand the idea into in-context learning;
- Use text input to condition the model on task description and some examples with ground truth.
- Uses zero-shot learning, one-shot learning, few-shot learning (as many examples as they can fit into the context, usually 10-100)
- No gradient updates are performed.
- A sort of associative lookup

In-context learning

Zero-shot

The model predicts the answer given only a natural language description of the task. No gradient updates are performed.

```
1  Translate English to French:        ← task description
2  cheese =>                           ← prompt
```

One-shot

In addition to the task description, the model sees a single example of the task. No gradient updates are performed.

```
1  Translate English to French:        ← task description
2  sea otter => loutre de mer          ← example
3  cheese =>                           ← prompt
```

Few-shot

In addition to the task description, the model sees a few examples of the task. No gradient updates are performed.

```
1  Translate English to French:        ← task description
2  sea otter => loutre de mer          ← examples
3  peppermint => menthe poivrée        ←
4  plush girafe => girafe peluche      ←
5  cheese =>                           ← prompt
```

Figure source: Brown et al.: Language Models are Few-Shot Learners, 2020.

- GPT-3 is still a language model and can be used for text generation

- only 12% of respondents correctly classified this as not written by a human

Title: United Methodists Agree to Historic Split
Subtitle: Those who oppose gay marriage will form their own denomination
Article: **After two days of intense debate, the United Methodist Church has agreed to a historic split - one that is expected to end in the creation of a new denomination, one that will be "theologically and socially conservative," according to The Washington Post. The majority of delegates attending the church's annual General Conference in May voted to strengthen a ban on the ordination of LGBTQ clergy and to write new rules that will "discipline" clergy who officiate at same-sex weddings. But those who opposed these measures have a new plan: They say they will form a separate denomination by 2020, calling their church the Christian Methodist denomination.**
The Post notes that the denomination, which claims 12.5 million members, was in the early 20th century the "largest Protestant denomination in the U.S.," but that it has been shrinking in recent decades. The new split will be the second in the church's history. The first occurred in 1968, when roughly 10 percent of the denomination left to form the Evangelical United Brethren Church. The Post notes that the proposed split "comes at a critical time for the church, which has been losing members for years," which has been "pushed toward the brink of a schism over the role of LGBTQ people in the church." Gay marriage is not the only issue that has divided the church. In 2016, the denomination was split over ordination of transgender clergy, with the North Pacific regional conference voting to ban them from serving as clergy, and the South Pacific regional conference voting to allow them.

Huge generative language models



- ChatGPT, OpenAI, Nov. 2022
  based on GPT-3.5 with additional training for dialogue
- uses instruction-tuning and RLHF
  (reinforcement learning with human feedback)
- demo: https://chat.openai.com/
- huge public impact, possibly disruptive for writing professions, learning, teaching, scientific writing
- GPT-4, 5: even larger, allows longer context, image input

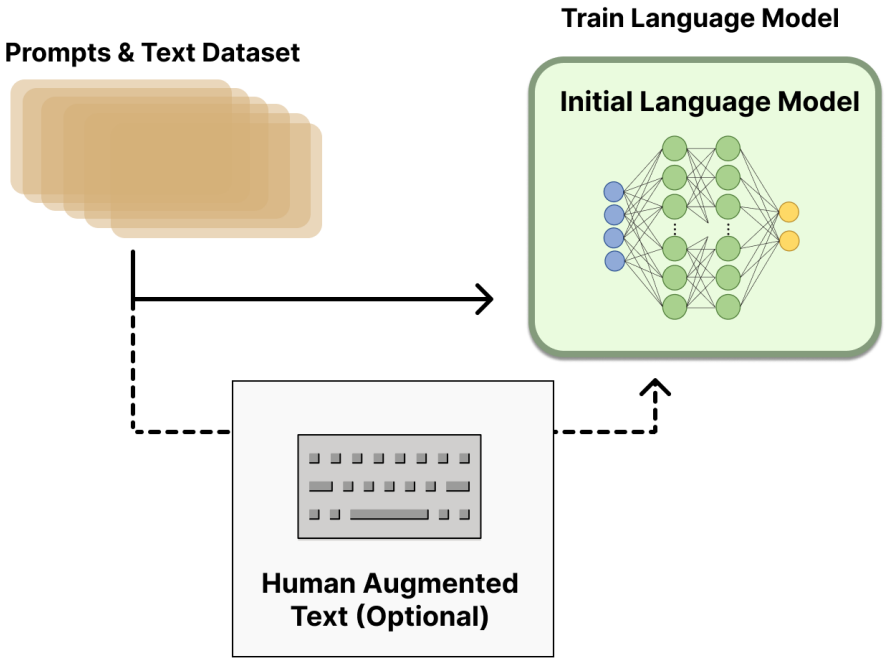
RLHF: idea

- Reinforcement Learning with Human Feedback
- A problem: Human feedback is not present during training
- Idea: Train a separate model on human feedback, this model can generate a reward to be used during training of LLM
- Three stages:
  1. Pretraining a language model (LM),
  2. Gathering data and training a reward model, and
  3. Fine-tuning the LM with reinforcement learning.

Nathan Lambert, Louis Castricato, Leandro von Werra, Alex Havrilla: "Illustrating Reinforcement Learning from Human Feedback (RLHF)", Hugging Face Blog, 2022.

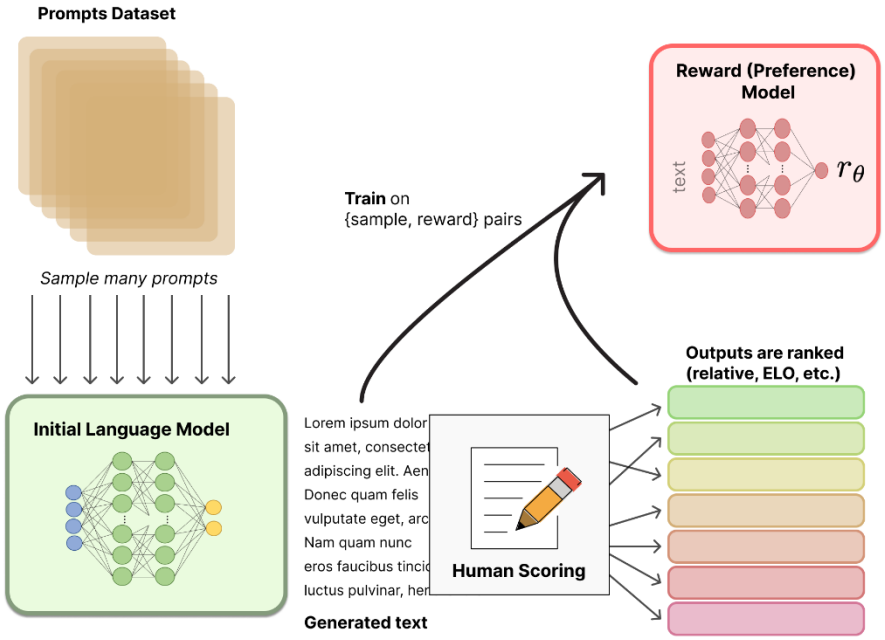
RLHF: the reward model 1/2

- input: a sequence of text, e.g., produced by LM and optionally improved by humans
- output: a scalar reward, representing the human preference of the text (e.g., a rank of the answer)
- the reward model could be an end-to-end LM, or the model ranks outputs, and the ranking is converted to reward



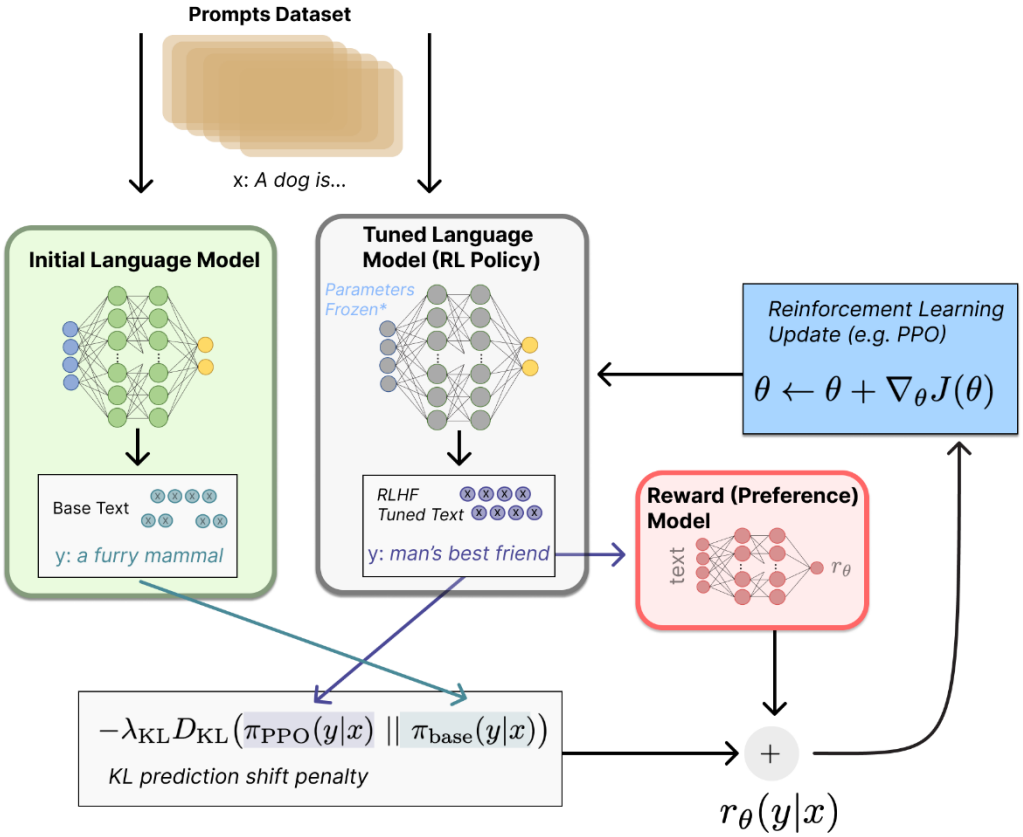
RLHF: the reward model 2/2

- the training dataset are pairs of prompts and (human improved) LM responses, e.g., 50k instances
- humans rank the responses instead of producing the direct reward as this produces better calibrated scores



RLHF: fine-tuning with RL

- RL does not change all parameters, most of parameters are frozen
- the algorithm: Proximal Policy Optimization (PPO)



NLP application examples

Sentiment analysis (SA)

- Definition: a computational study of opinions, sentiments, emotions, and attitudes expressed in texts towards an entity.
- Purpose: detecting public moods, i.e. understanding the opinions of the general public and consumers on social events, political movements, company strategies, marketing campaigns, product preferences etc.
- Part of Affective Computing (emotion, mood, personality traits, interpersonal stance, attitude)
- Can be target-based or general

SA: getting and preprocessing data

- Frequent data sources:
  - Twitter-X, forum comments, product review sites, company's Facebook pages
- Data cleaning
  - quality assessment, annotator (self-) agreement
  - preprocessing: tokenization, emojis, links, hashtags, etc,

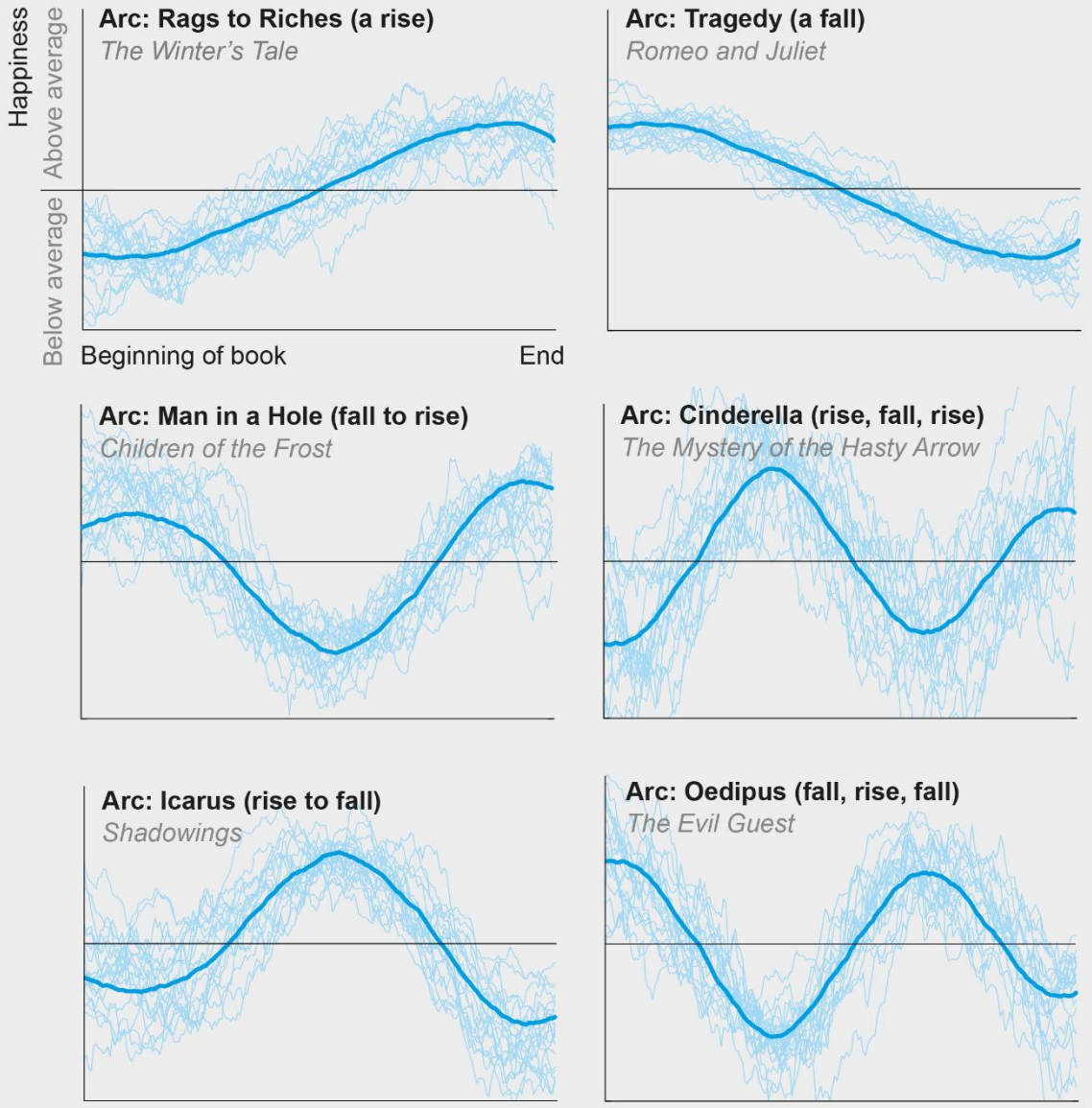
SA tasks

- sentiment classification (binary (polarity), ternary, n-ary
- subjectivity classification (vs. objectivity)
- review usefulness classification
- opinion spam classification
- emotion analysis
- hate speech, offensive speech, socially unacceptable speech
- stance detection

Emotional states in English fiction

Emotional Arcs

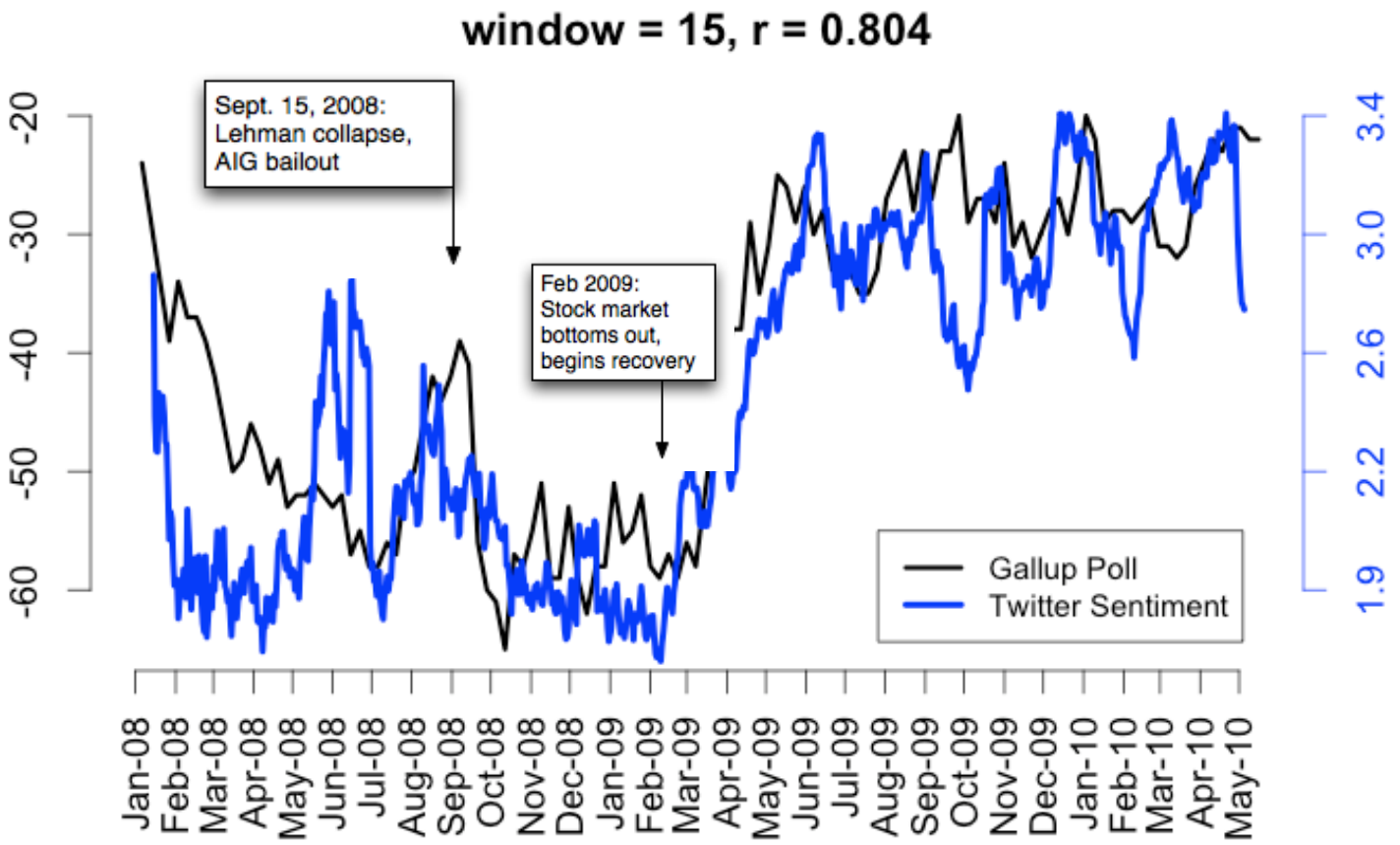
About 85 percent of 1,327 fiction stories in the digitized Project Gutenberg collection follow one of six emotional arcs—a pattern of highs and lows from beginning to end (*dark curves*). The arcs are defined by the happiness or sadness of words in the running text (*jagged plots*). All books were in English and less than 100,000 words; examples are noted.



Public opinion surveys

Twitter sentiment vs. Gallup on consumer sentiment

Brendan O'Connor, Ramnath Balasubramanyan, Bryan R. Routledge, and Noah A. Smith. 2010. From Tweets to Polls: Linking Text Sentiment to Public Opinion Time Series. In ICWSM-2010



Statistical machine translation

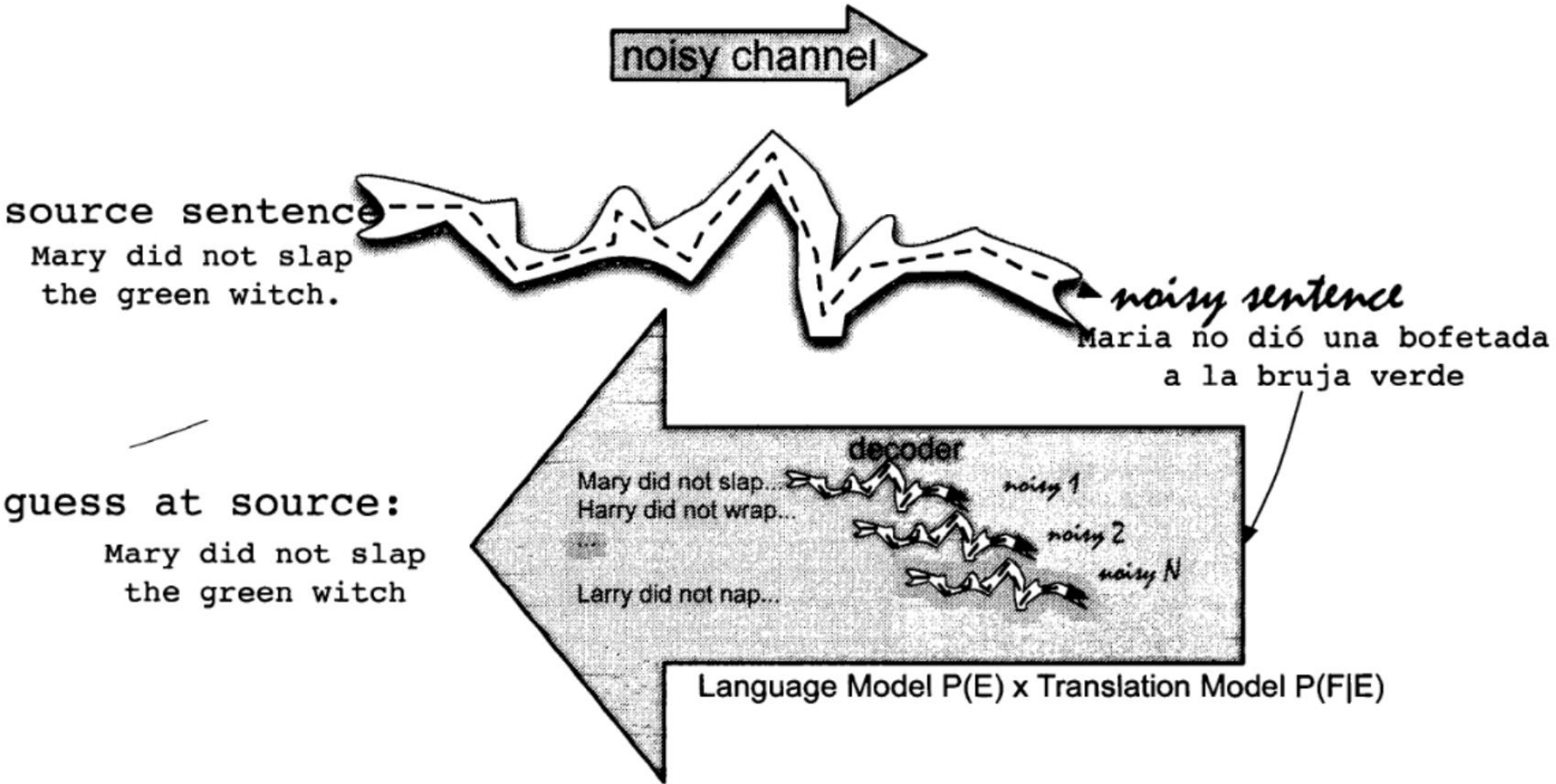
- non-neural approach: no longer used in practice but gives insight of what is needed
- idea from the theory of information
- we translate from foreign language F to English E
- a document is translated based on the probability distribution $p(e \mid f)$, i.e. the probability of the sentence e in target language based on the sentence in source language f
- Bayes rule
  $\arg\max_e p(e \mid f) = \arg\max_e p(f \mid e)\, p(e) \,/\, p(f)$
- $p(f)$ can be ignored as it is a constant for a given fixed sentence
- traditional (non-neural) approaches split the problem into subproblems
  - create a language model $p(e)$
  - a separate translation model $p(f \mid e)$
  - decoder forms the most probable e based on f

Noisy channel model

- given English sentence e
- during transmission over a noisy channel the sentence e is corrupted and we get sentence in a foreign language f, which we are able to observe
- to reconstruct the most probable sentence e we have to figure out:
  - how people speak in English (language model), p(e) and
  - how to transform a foreign language into English (translation model), p(f | e)

Noisy channel

- reasoning goes back



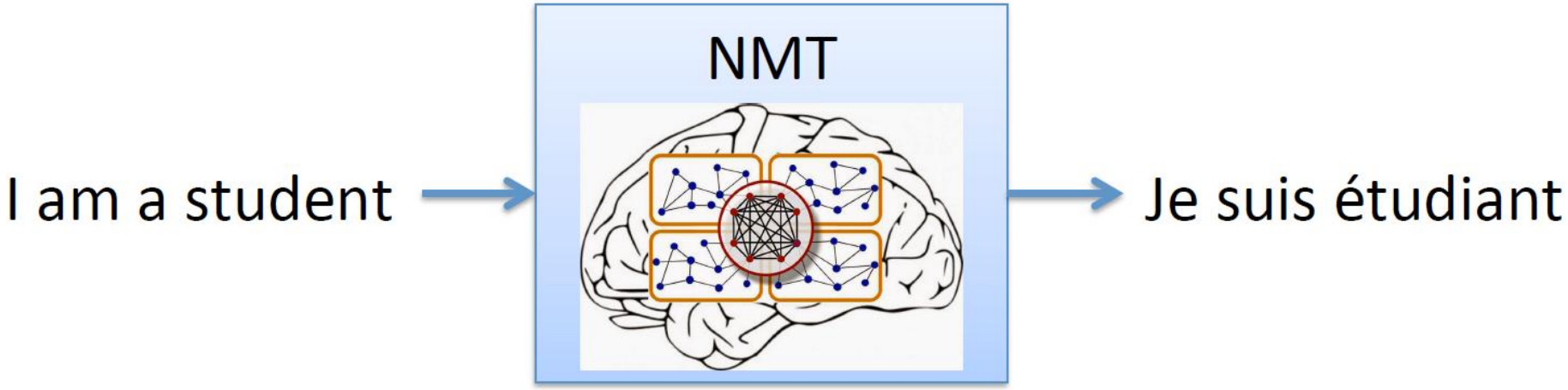
Language model

- each target (English) sentence e is assigned a probability p(e)
- estimation of probabilities for the whole sentences is not possible (why?), therefore we use language models, e.g., 3-gram models or neural language models

Translation model

- we have to assign a probability of p(f | e), which is a probability of a foreign language sentence f, given target sentence e.
- we search the e which maximizes p(e) * p(f | e)
- traditional MT approach: using translation corpus we determine which translation of a given word is the most probable
- we take into account the position of a word and how many words are needed to translate a given word

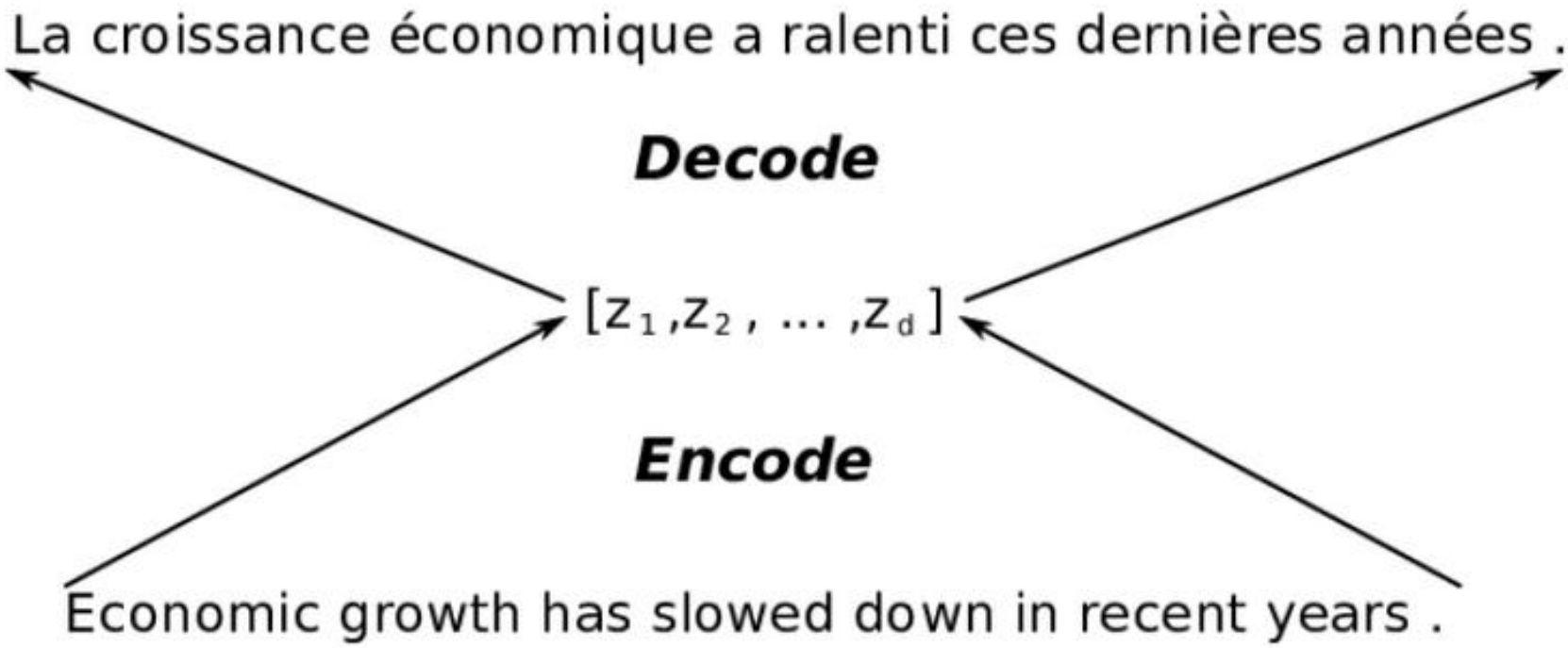
Neural machine translation (NMT)



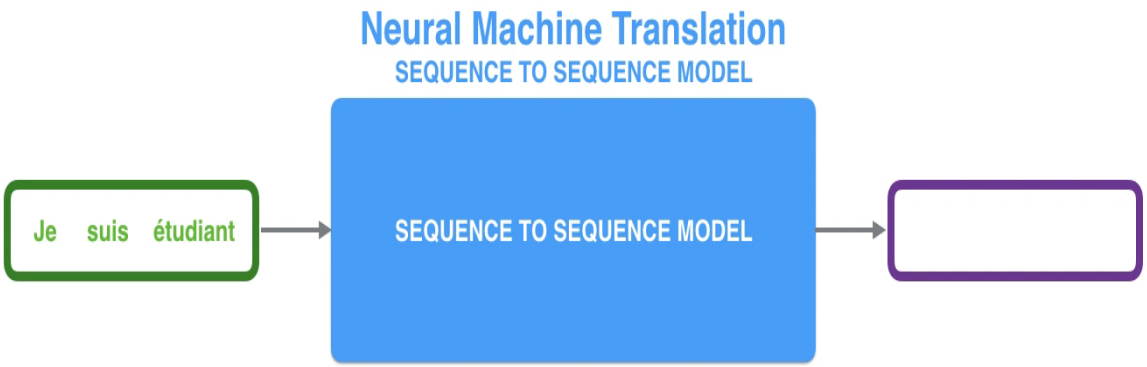
(Sutskever et al., 2014; Cho et al., 2014)

- sequence to sequence machine translation (seq2seq)
- end-to-end optimization

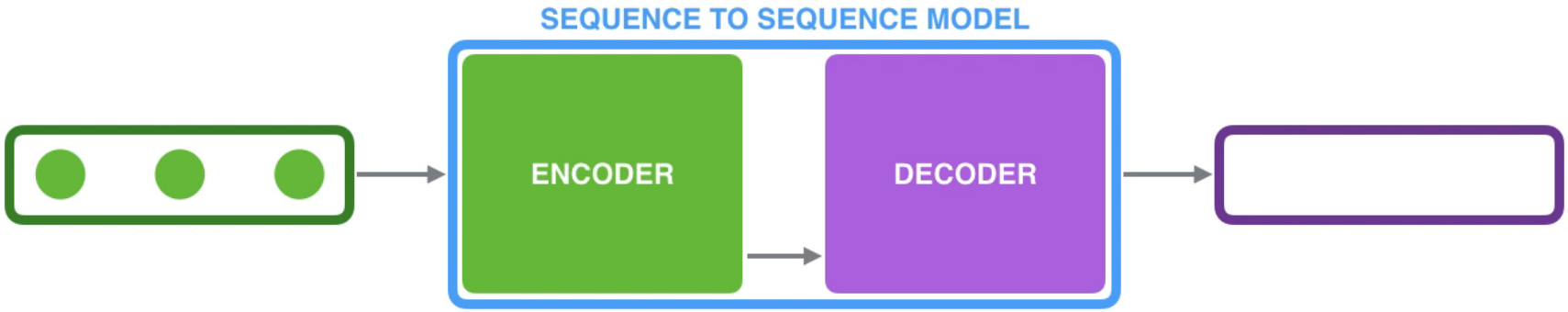
Encoder-Decoder model



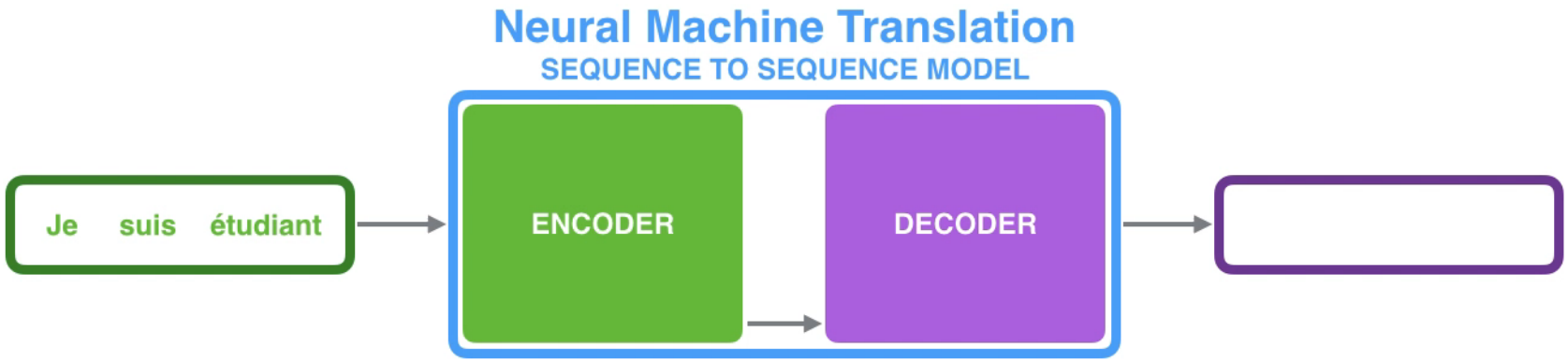
Seq2Seq for NMT

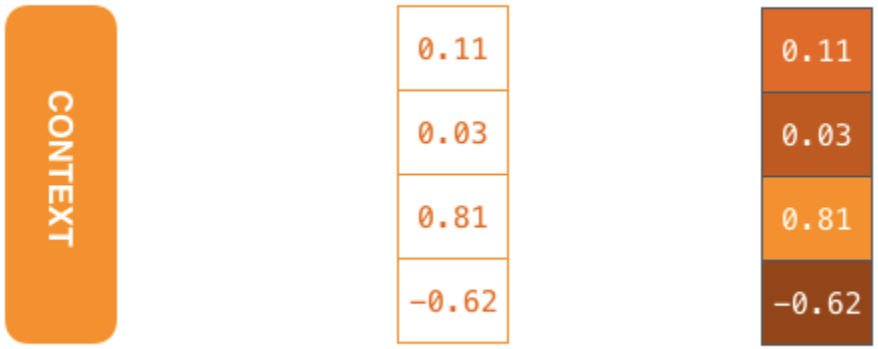


Encoder-decoder for sequences



Encoder-decoder for NMT

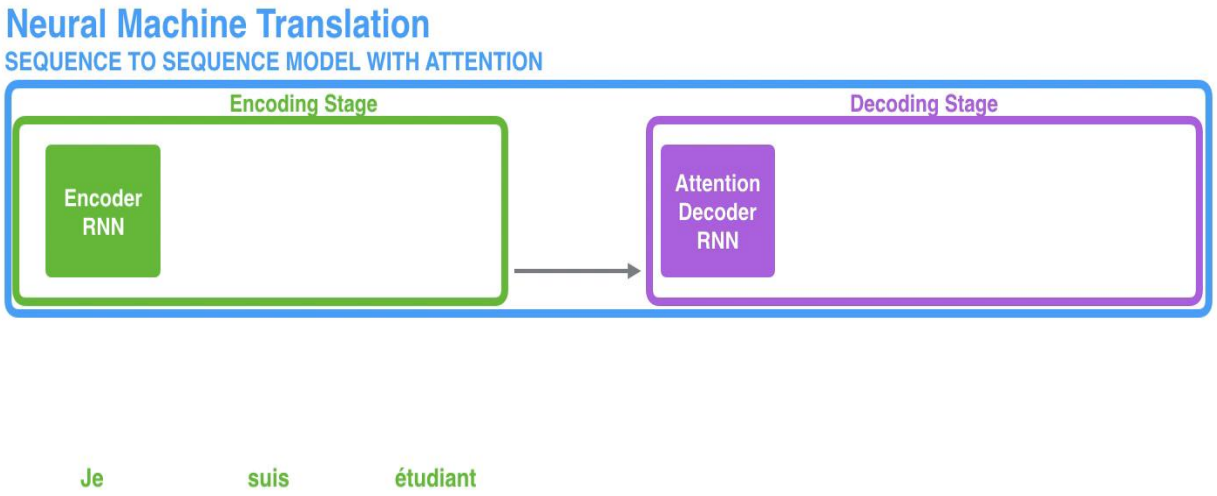




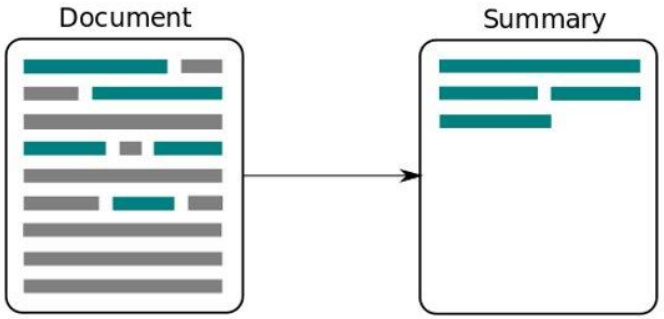
Training NMT

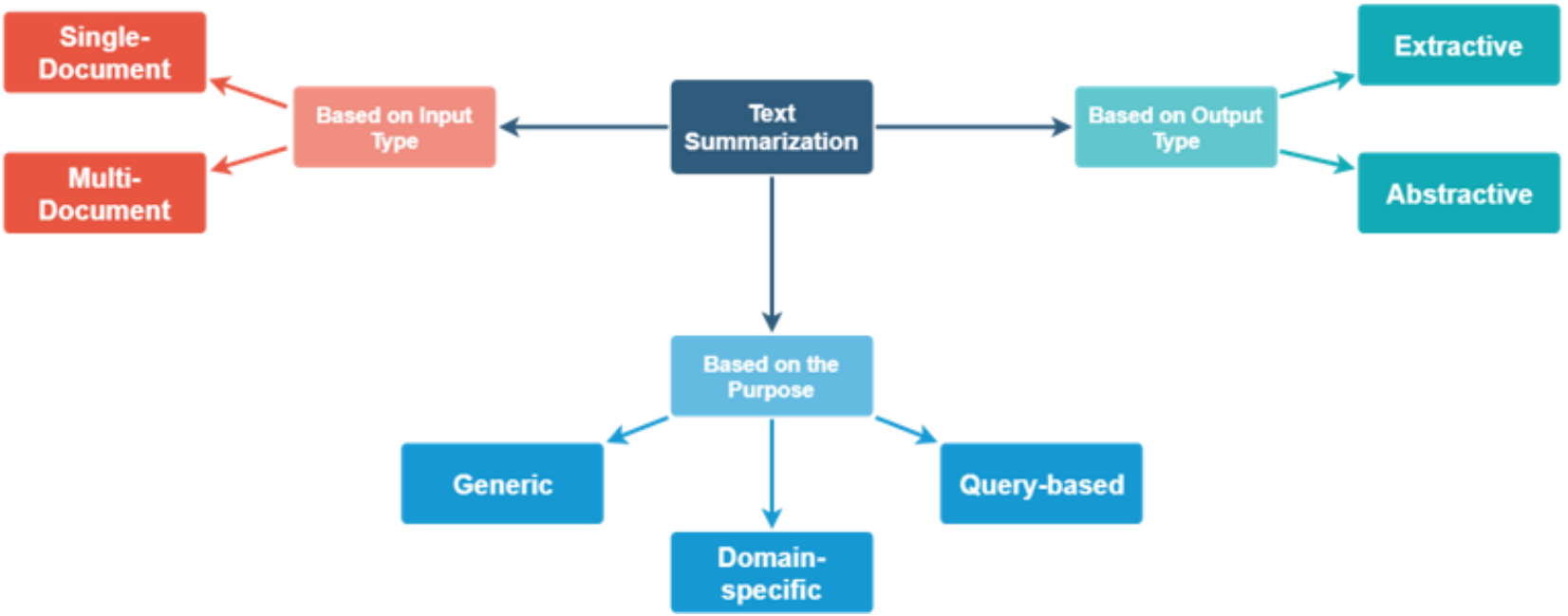
- using transformers
- softmax for output
- we maximize
  P(output sentence | input sentence)
- we sum errors on all outputs
- backpropagation
- training on correct translations
- as the translation, we return a sequence of words with the highest probability (not necessary greedily)

NMT with attention



Text summarization



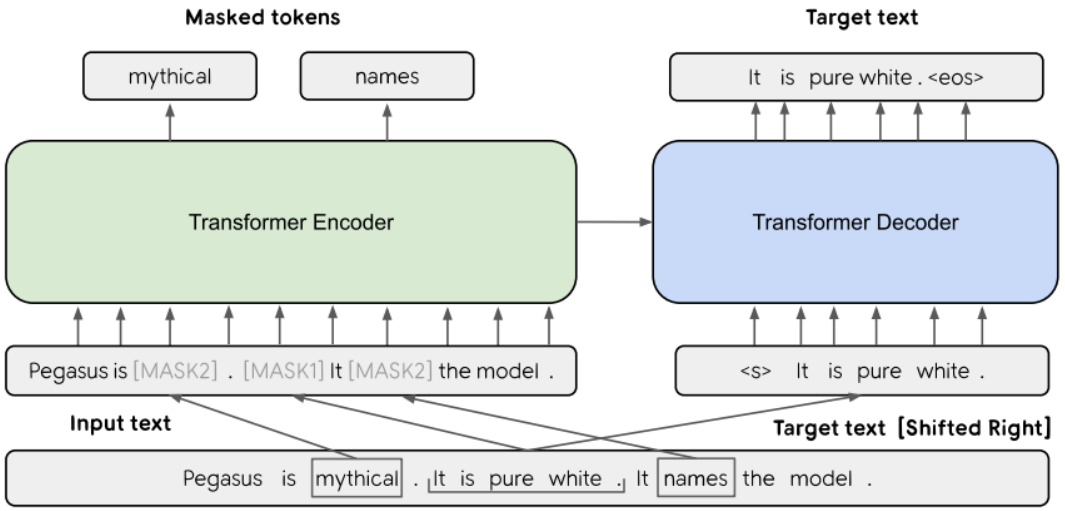


Text summarization

- Evaluation:
  - ROUGE scores,
  - BERTScore,
  - with QA:
    - question generation,
    - searching for answers in the summary
  - human
- LLMs
- Short and long texts
- cross-lingual

Summarizers – Pegasus

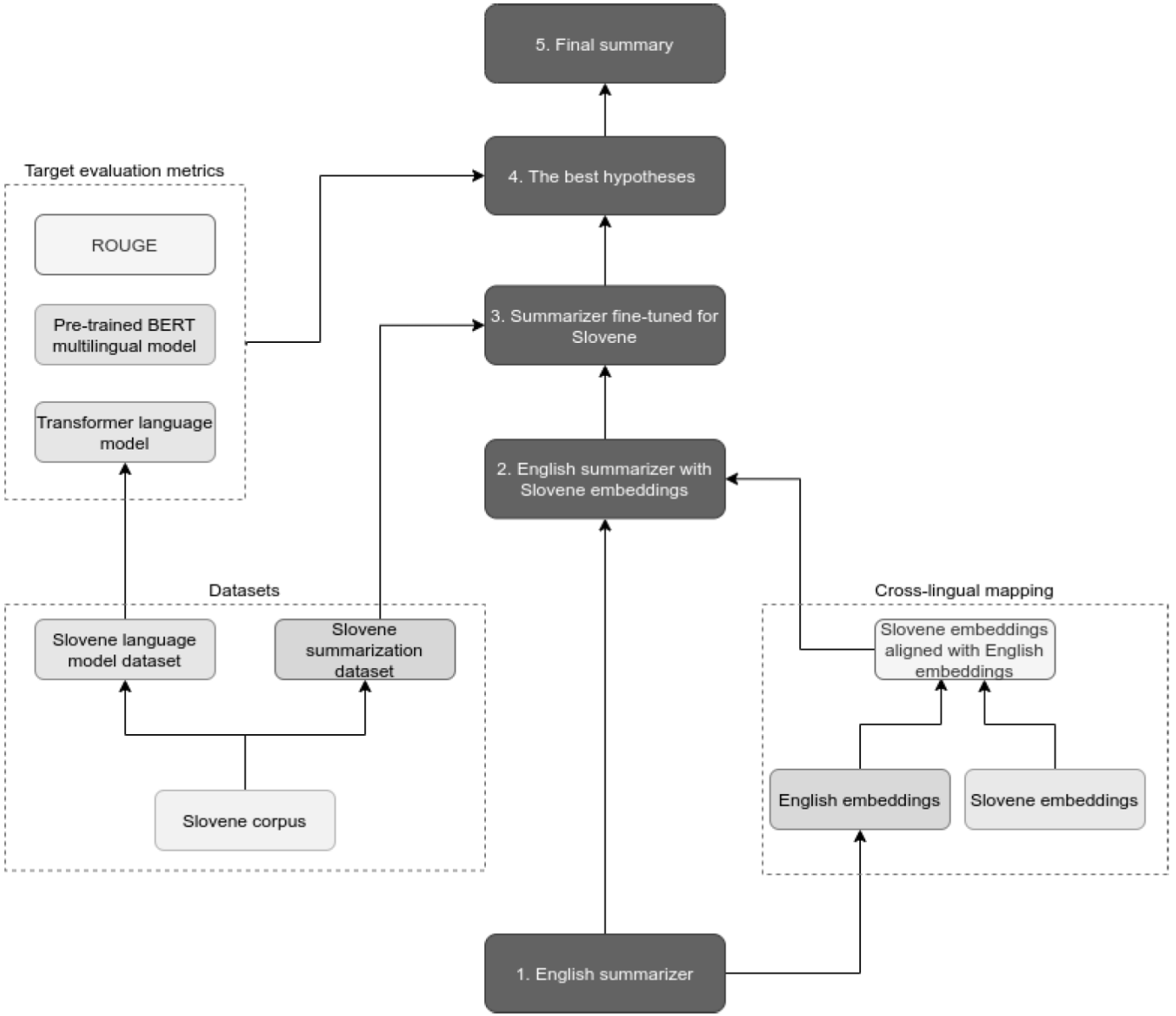
- An example of a different, successful transformer model
- Transformer BART model
- encoder-decoder architecture
- text garbling and reconstruction
- Auxiliary tasks: masked language model and missing sentence generation
- Demo: https://ai.googleblog.com/2020/06/pegasus-state-of-art-model-for.html



(Zhang et al., 2019)

Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

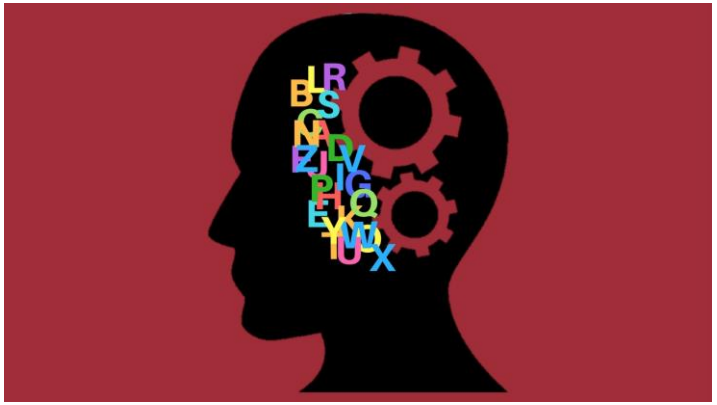
XL summarization architecture



Žagar, A., & Robnik-Šikonja, M. (2022). Cross-lingual transfer of abstractive summarizer to less-resource language. Journal of Intelligent Information Systems, 58(1), 153-173.

University of Ljubljana, Faculty of Computer and Information Science

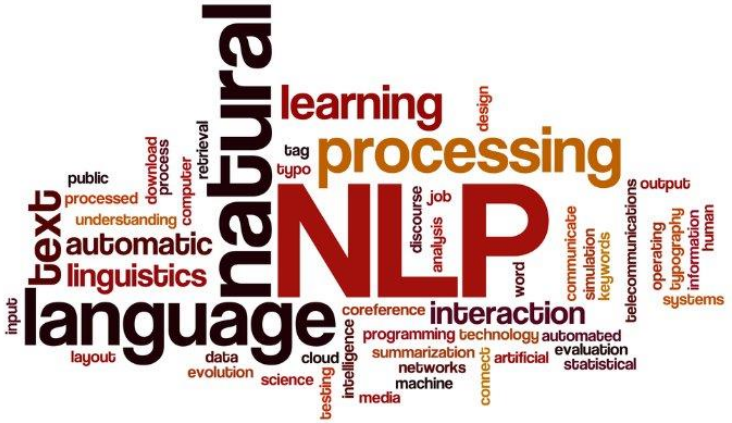
Natural language processing



Prof Dr Marko Robnik-Šikonja

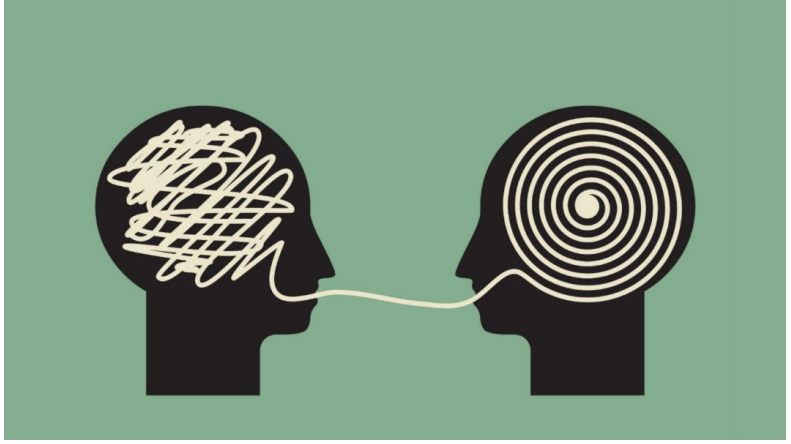
Intelligent Systems, Edition 2024

Topic overview



- understanding language and intelligence
- text preprocessing and linguistic analysis
- components of modern NLP
  - text representations
  - information retrieval
  - similarity of words and documents
  - language and graphs
  - large language models
- practical use of NLP:
  - sentiment analysis,
  - paper recommendations
  - summarization

Understanding language



- A grand challenge of (not only?) artificial intelligence

  Who can understand me?
  Myself I am lost
  Searching but cannot see
  Hoping no matter cost
  Am I free?
  Or universally bossed?

- Not just poetry, what about instructions, user manuals, newspaper articles, seminary works, internet forums, twits, legal documents, i.e. license agreements, etc.

An example: rules

Article 18 of FRI Study Rules and Regulations

Taking exams at an earlier date may be allowed on request of the student by the Vice-Dean of Academic Affairs with the course convener's consent in case of mitigating circumstances (leaving for study or placement abroad, hospitalization at the time of the exam period, giving birth, participation at a professional or cultural event or a professional sports competition, etc.), and if the applicant's study achievements in previous study years are deemed satisfactory for such an authorization to be appropriate.

Understanding NL by computers

- Understanding words, syntax, semantics, context, writer's intentions, knowledge, background, assumptions, bias …
- Doesn't seem that LLM do it, though they generate excellent text output
- Ambiguity in language
  - Newspaper headlines - intentional ambiguity :)
    - Juvenile court to try shooting defendant
    - Kids make nutritious snacks
    - Miners refuse to work after death
    - Doctor on Trump's health: No heart, cognitive issues

Ambiguity

- I made her duck.

- Possible interpretations:
  - I cooked waterfowl for her.
  - I cooked waterfowl belonging to her.
  - I created the (plaster?) duck she owns.
  - I caused her to quickly lower her head or body.
  - I waved my magic wand and turned her into undifferentiated waterfowl.

- Spoken ambiguity
  - eye, maid

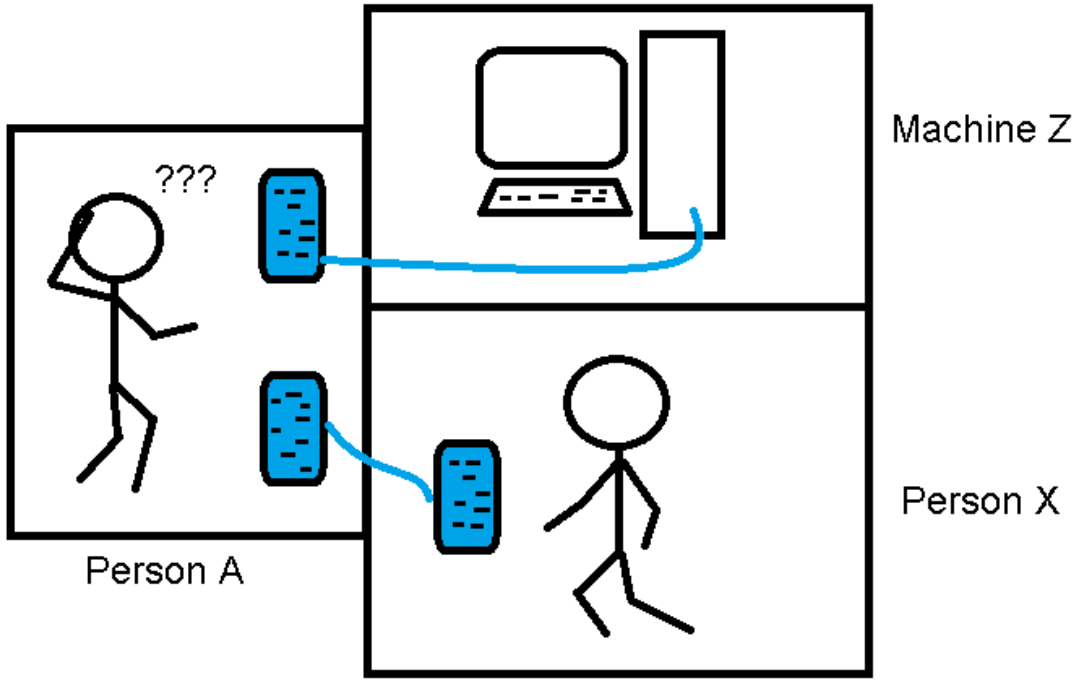
Disambiguation in syntax and semantics

- in syntax
  - The girl told the story cried.
  - John went to the bank.
  - Put the box on the table in the kitchen.
  - Bring your old car seat to be recycled.
- *Search engine queries: natural language processing, jaguar, Paris,*
- *POS tagging, word sense disambiguation*

Understanding language as (artificial) intelligence test

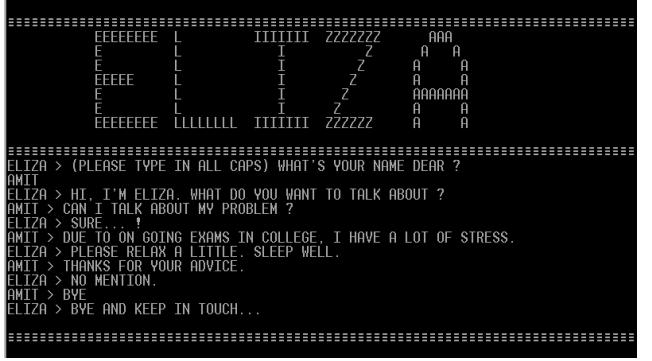
- The system understanding natural language on human level should contain a large portion of human knowledge and would achieve the most important goals of artificial intelligence. Its response would be indistinguishable from human's.

Turing test



- Turing test, Loebner prize, CYC, ELISA, ALLICA, chatterbot, VIDA, TIA, Siri (knowledge recognition, context detection, web service recommendation)
- Seems that LLMs can pass it, are they intelligent?
- The question: is mimicking human skills sufficient?

ELIZA



- ELIZA (Weizenbaum, 1966)
- Limited conversation (Rogerian psyhiatrist)

User: You are like my father in some ways.
ELIZA: What resemblance do you see?
User: You are not very aggressive but I think you don't want me to notice that.
ELIZA: What makes you think I am not aggressive?
User: You don't argue with me.
ELIZA: Why do you think I don't argue with you?
User: You are afraid of me.
ELIZA: Does it please you to believe I am afraid of you?

GPT-3 regressive language model as a chatbox

- the example is cherry-picked

```
Q: What is your favorite animal?
A: My favorite animal is a dog.

Q: Why?
A: Because dogs are loyal and friendly.

Q: What are two reasons that a dog might be in a bad mood?
A: Two reasons that a dog might be in a bad mood are if it is hungry or if it is hot.

Q: How many eyes does a giraffe have?
A: A giraffe has two eyes.

Q: How many legs does a frog have?
A: A frog has four legs.

Q: Are there any animals with three legs?
A: No, there are no animals with three legs.

Q: Why don't animals have three legs?
A: Animals don't have three legs because they would fall over.
```

Where is NLP today?

- active research area with many commercial applications
  - large language models are used everywhere
  - automatic reply engines
  - machine translation
  - text summarization
  - question answering
  - speech recognition and synthesis
  - language generation
  - interface to databases
  - intelligent search and information extraction
  - sentiment detection
  - named entity recognition and linking
  - categorization and classification of documents, messages, twits, etc.
  - cross-lingual approaches
  - multi modal approaches (text + images, text + video)
  - attempt to get to artificial general intelligence (AGI) through "foundation models"
  - many (open-source) tools and language resource
  - prevalence of deep neural network approaches (i.e. transformers)

Recommended literature

- Jurafsky, Daniel and James Martin (2024): Speech and Language Processing, 3rd edition in progress, almost all parts are available at authors‘ webpages https://web.stanford.edu/~jurafsky/slp3/
- Steven Bird, Ewan Klein, and Edward Loper. Natural Language Processing with Python. O’Reilly, 2009
  - a free book accompanying NLTK library, regularly updated
  - *Python 3, http://www.nltk.org/book/*
- Coursera
  - several courses, e.g., Stanford NLP with DNN

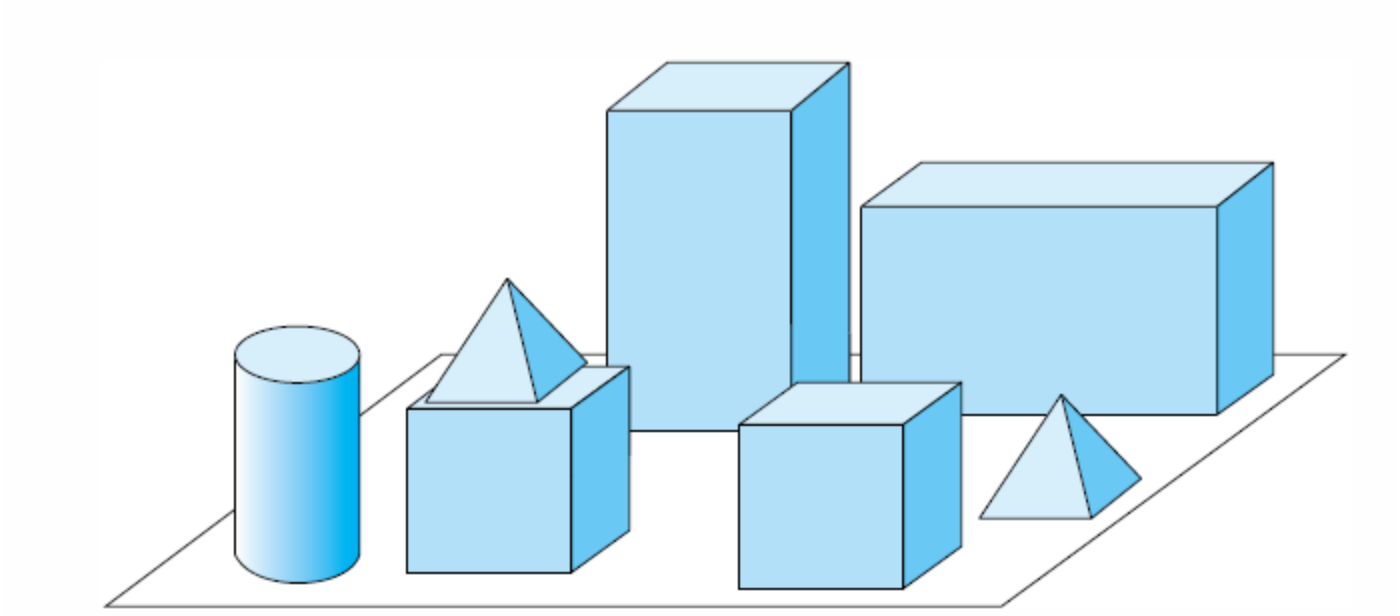
Historically two approaches

- symbolical
  - ,,Good Old-Fashioned AI''
- empirical
  - Statistical, text corpora
- Merging both worlds: injecting symbolical knowledge (e.g., propositional logic) into LLMs

How it all started?

- micro worlds
- example: SHRDLU, world of simple geometric objects
  - What is sitting on the red block?
  - What shape is the blue block on the table?
  - Place the green pyramid on the red brick.
  - Is there a red block? Pick it up.
  - What color is the block on the blue brick? Shape?

Micro world: block world, SHRDLU (Winograd, 1972)



Linguistic analysis 1/2

Linguistic analysis contains several tasks: recognition of sounds, letters, word formation, syntactic parsing, recognizing semantic, emotions. Phases:

- Prosody - the patterns of stress and intonation in a language (rhythm and intonation)
- Phonology - systems of sounds and relationships among the speech sounds that constitute the fundamental components of a language
- Morphology - the admissible arrangement of sounds in words; how to form words, prefixes and suffixes …
- Syntax - the arrangement of words and phrases to create well-formed sentences in a language

Linguistic analysis 2/2

- Semantics - the meaning of a word, phrase, sentence, or text
- Pragmatics - language in use and the contexts in which it is used, including such matters as deixis (words whose meaning changes with context, e.g., I he, here, there, soon), taking turns in conversation, text organization, presupposition, and implicature

  Can you pass me the salt? Yes, I can.
- Knowing the world: knowledge of physical world, humans, society, intentions in communications …
- Limits of linguistic analysis, levels are dependent

Classical approach to text processing

- text preprocessing
- 1. phase: syntactic analysis
- 2. phase: semantic interpretation
- 3. phase: use of world knowledge

In the neural approach, the preprocessing remains (but is simpler) the other three phases are merged into DNN

Basic tools for text preprocessing

- document → paragraphs → sentences → words (→ (subword) tokens)

- In linguistic analysis also
  - words and sentences ← lemmatization, POS tagging
  - sentences ← syntactical and grammatical analysis
  - named entity recognition,

Words and sentences

- sentence delimiters – punctuation marks and capitalization are insufficient
- E.g., remains of 1. Timbuktu from 5c BC, were discovered by dr. Barth.
- Lexical analysis (tokenizer, word segmenter), not just spaces
  - 1,999.00€ or 1.999,00€!
  - Ravne na Koroškem
  - Lebensversicherungsgesellschaft
  - Port-au-prince
  - Generalstaatsverordnetenversammlungen
- Rules, regular expressions, statistical models, dictionaries (of proper names), neural networks, manually segmented datasets

Lemmatization

- Lemmatization is the process of grouping together the different inflected forms of a word so they can be analyzed as a single item.
- walk is the lemma of 'walk', 'walked', 'walks', 'walking
- Lemmatization difficulty is language dependent i.e., depends on morphology
- Requires dictionary or lexicon for lookup
  go, goes, going, gone, went
  jaz, mene, meni, mano
- Ambiguity resolution may be difficult

Meni je vzel z mize (zapestnico).

- Uses rules, dictionaries, neural networks, manually labelled datasets
- Enlish also uses stemming (reducing inflected or derived words to their word stem

POS tagging

- assigning the correct part of speech (noun, verb, etc.) to words
- helps in recognizing phrases and names
- Use rules, machine learning models, manually labelled datasets

An example:

- Text analyzer for Slovene, i.e. morphosyntactical tagging, available at https://orodja.cjvt.si/oznacevalnik/slv/

  Nekega dne sem se napotil v naravo. Že spočetka me je žulil čevelj, a sem na to povsem pozabil, ko sem jo zagledal. Bila je prelepa. Povsem nezakrita se je sončila na trati ob poti. Pritisk se mi je dvignil v višave. Popoln primerek kmečke lastovke!

- Tags are standardized, for East European languages in Multext-East specification, e.g.,

dne; tag Somer = Samostalnik, obče ime, moški spol, ednina, rodilnik; lema: dan

a unifying attempt: universal dependencies (UD): cross-linguistically consistent treebank annotation for many languages

CLASSLA-Stanza pipeline output

ID	Oblika	Lema	Oznaka JOS	Bes. vrsta JOS	Oblikoskladenjske lastnosti JOS	Bes. vrsta UD	Oblikoskladenjske lastnosti UD	Nadrejena pojavnica UD	Skladenjska relacija UD	Nadrejena pojavnica JOS	Skladenjska relacija JOS	Udeleže vloga
# paragraph 1												
# sent_id 1.1												
# text = Nekega dne sem se napotil v naravo.												
1	Nekega	nek	Pi-msg	Pronoun	Type=indefinite Gender=masculine Number=singular Case=genitive	DET	Case=Gen|Gender=Masc|Number=Sing|PronType=Ind	2	det	2	Atr	
2	dne	dan	Ncmsg	Noun	Type=common Gender=masculine Number=singular Case=genitive	NOUN	Case=Gen|Gender=Masc|Number=Sing	5	obl	5	AdvO	TIME
3	sem	biti	Va-r1s-n	Verb	Type=auxiliary VForm=present Person=first Number=singular Negative=no	AUX	Mood=Ind|Number=Sing|Person=1|Polarity=Pos|Tense=Pres|VerbForm=Fin	5	aux	5	PPart	
4	se	se	Px------y	Pronoun	Type=reflexive Clitic=yes	PRON	PronType=Prs|Reflex=Yes|Variant=Short	5	expl	5	PPart	
5	napotil	napotiti	Vmep-sm	Verb	Type=main Aspect=perfective VForm=participle Number=singular Gender=masculine	VERB	Aspect=Perf|Gender=Masc|Number=Sing|VerbForm=Part	0	root	0	Root	
6	v	v	Sa	Adposition	Case=accusative	ADP	Case=Acc	7	case	7	Atr	
7	naravo	narava	Ncfsa	Noun	Type=common Gender=feminine Number=singular Case=accusative	NOUN	Case=Acc|Gender=Fem|Number=Sing	5	obl	5	AdvO	GOAL
8	.	.	Z	Punctuation		PUNCT	_	5	punct	0	Root	

- *Nekega dne sem se napotil v naravo. Že spočetka me je žulil čevelj, a sem na to povsem pozabil, ko sem jo zagledal. Bila je prelepa. Povsem nezakrita se je sončila na trati ob poti. Pritisk se mi je dvignil v višave. Popoln primerek kmečke lastovke!*

1	**beseda**	Nekega	dne	sem	se	napotil	v	naravo	.	Že	spočetka	me	je								
	lema	nek	dan	biti	se	napotiti	v	narava	že	spočetka	jaz	biti									
	oznaka	Zn-mer	Somer	Gp-spe-n	Zp------k	Ggdd-em	Dt	Sozet	.	L	Rsn	Zop-et--k	Gp-ste-n								
2	**beseda**	žulil	čevelj	,	a	sem	na to	povsem	pozabil	,	ko	sem	jo	zagledal							
	lema	žuliti	čevelj	a	biti	na ta	povsem	pozabiti	ko	biti	on	zagledati									
	oznaka	Ggnd-em	Somei	,	Vp	Gp-spe-n	Dt	Zk-set	Rsn	Ggdd-em	,	Vd	Gp-spe-n	Zotzet--k	Ggdd-em						
3	**beseda**	.	Bila	je	prelepa	.	Povsem	nezakrita	se	je	sončila	na	trati								
	lema		biti	biti	prelep		povsem	nezakrit	se	biti	sončiti	na	trata								
	oznaka	.	Gp-d-ez	Gp-ste-n	Ppnzei	.	Rsn	Ppnzei	Zp------k	Gp-ste-n	Ggvd-ez	Dm	Sozem								
4	**beseda**	ob	poti	.	Pritisk	se	mi	je	dvignil	v	višave	.	Popoln								
	lema	ob	pot		pritisk	se	jaz	biti	dvigniti	v	višava		popoln								
	oznaka	Dm	Sozem	.	Somei	Zp------k	Zop-ed--k	Gp-ste-n	Ggdd-em	Dt	Sozmt	.	Ppnmein								
5	**beseda**	primerek	kmečke	lastovke	!																
	lema	primerek	kmečki	lastovka																	
	oznaka	Somei	Ppnzer	Sozer	!																

TEI-XML format

```
<TEI xmlns="http://www.tei-c.org/ns/1.0">
  <text>
    <body>
      <p>
        <s>
          <w msd="Zn-mer" lemma="nek">Nekega</w>
          <S/>
          <w msd="Somer" lemma="dan">dne</w>
          <S/>
          <w msd="Gp-spe-n" lemma="biti">sem</w>
          <S/>
          <w msd="Zp------k" lemma="se">se</w>
          <S/>
          <w msd="Ggdd-em" lemma="napotiti">napotil</w>
          <S/>
          <w msd="Dt" lemma="v">v</w>
          <S/>
          <w msd="Sozet" lemma="narava">naravo</w>
          <c>.</c>
          <S/>
        </s>
   …

      </p>
    </body>
  </text>
</TEI>
```

MSD tags

- Multext-East specification

dne; tag Somer = Samostalnik, obče ime, moški spol, ednina, rodilnik; lema: dan

P	atribut	vrednost	koda	atribut	vrednost	koda
0	glagol		G	Verb		V
1	vrsta	glavni	g	Type	main	m
		pomožni	p		auxiliary	a
2	vid	dovršni	d	Aspect	perfective	e
		nedovršni	n		imperfective	p
		dvovidski	v		biaspectual	b
3	oblika	nedoločnik	n	VForm	infinitive	n
		namenilnik	m		supine	u
		deležnik	d		participle	p
		sedanjik	s		present	r
		prihodnjik	p		future	f
		pogojnik	g		conditional	c
		velelnik	v		imperative	m
4	oseba	prva	p	Person	first	1
		druga	d		second	2
		tretja	t		third	3
5	število	ednina	e	Number	singular	s
		množina	m		plural	p
		dvojina	d		dual	d
6	spol	moški	m	Gender	masculine	m
		ženski	z		feminine	f
		srednji	s		neuter	n
7	nikalnost	nezanikani	n	Negative	no	n
		zanikani	d		yes	y

POS tagging in English

- http://nlpdotnet.com/Services/Tagger.aspx
- Rainer Maria Rilke, 1903
  in Letters to a Young Poet

...I would like to beg you dear Sir, as well as I can, to have patience with everything unresolved in your heart and to try to love the questions themselves as if they were locked rooms or books written in a very foreign language. Don't search for the answers, which could not be given to you now, because you would not be able to live them. And the point is to live everything. Live the questions now. Perhaps then, someday far in the future, you will gradually, without even noticing it, live your way into the answer.

POS tagger output

I/PRP would/MD like/VB to/TO beg/VB you/PRP dear/JJ Sir/NNP ,/, as/RB well/RB as/IN I/PRP can/MD ,/, to/IN have/VBP patience/NN with/IN everything/NN unresolved/JJ in/IN your/PRP$ heart/NN and/CC to/TO try/VB to/TO love/VB the/DT questions/NNS themselves/PRP as/RB if/IN they/PRP were/VBD locked/VBN rooms/NNS or/CC books/NNS written/VBN in/IN a/DT very/RB foreign/JJ language/NN ./.

Named entity recognition (NER)

- NATO Secretary-General Jens Stoltenberg is expected to travel to Washington, D.C. to meet with U.S. leaders.
- [ORG NATO] Secretary-General [PER Jens Stoltenberg] is expected to travel to [LOC Washington, D.C.] to meet with [LOC U.S.] leaders.
- Named entity linking (NEL) also named entity disambiguation – linking to a unique identifier, e.g. wikification
  jaguar, Paris, London, Dunaj

Basic language resources: corpora

- Statistical natural language processing list of resources http://nlp.stanford.edu/links/statnlp.html
- Opus http://opus.nlpl.eu/ multilingual parallel corpora, e.g., DGT JRC-Acqui 3.0, Documents of the EU in 22 languages
- Slovene language corpora GigaFida, ccGigaFida, KRES, ccKres, GOS, Artur, JANES, KAS, Trendi
- The main Slovene language resources
  - http://www.clarin.si
  - https://github.com/clarinsi
  - http://www.cjvt.si/
  - https://www.slovenscina.eu/
- WordNet, SloWNet, sentiWordNet, …
- Thesaurus https://viri.cjvt.si/sopomenke/slv/
- LLMs: on HuggingFace cjvt, e.g., SloBERTa and GaMS

WordNet is a database composed of synsets:

synonyms,
hypernyms
hyponyms,
meronyms,
holonyms,
etc.

Word to search for: mercy [Search WordNet]

Display Options: (Select option to change) [Change]

Key: "S:" = Show Synset (semantic) relations, "W:" = Show Word (lexical) relations

Display options for sense: (gloss) "an example sentence"

Noun

- S: (n) clemency, mercifulness, **mercy** (leniency and compassion shown toward offenders by a person or agency charged with administering justice) *"he threw himself on the mercy of the court"*
- S: (n) mercifulness, **mercy** (a disposition to be kind and forgiving) *"in those days a wife had to depend on the mercifulness of her husband"*
- S: (n) mercifulness, **mercy** (the feeling that motivates compassion)
  - *direct hyponym* / *full hyponym*
    - S: (n) forgiveness (compassionate feelings that support a willingness to forgive)
  - *direct hypernym* / *inherited hypernym* / *sister term*
    - S: (n) compassion, compassionateness (a deep awareness of and sympathy for another's suffering)
  - ***derivationally related form***
    - W: (adj) merciful [Related to: mercifulness] (showing or giving mercy) *"sought merciful treatment for the captives"; "a merciful god"*
- S: (n) **mercy** (something for which to be thankful) *"it was a mercy we got out alive"*
- S: (n) **mercy** (alleviation of distress; showing great kindness toward the distressed) *"distributing food and clothing to the flood victims was an act of mercy"*

Popular NLP applications

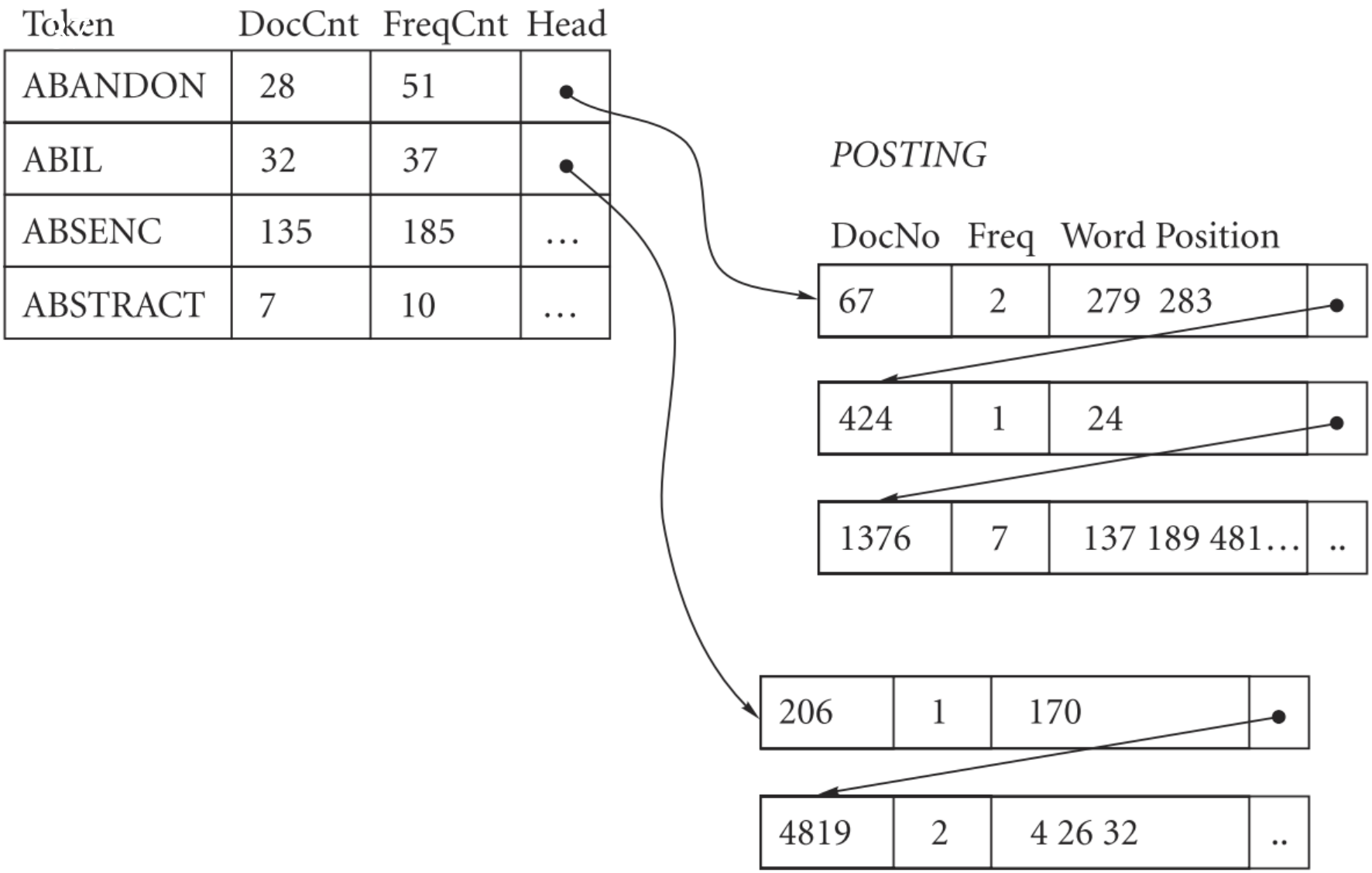
- document retrieval
- information extraction
- automatic speech recognition and generation
- text classification
- automatic summarization
- question answering
- sentiment analysis, emotion detection, stance detection
- machine translation,
- language generation
- comment filtering, hate speech detection, fake news detection
- topic analysis
- grammar tools
- many more

Document retrieval

- Historical: keywords
- Now: whole text search
- Organize a database, indexing, search algorithms
- input: a query (of questionable quality, ambiguity, answer quality)

Document indexing

- Collect all words from all documents, use lemmatization
- The inverted file data structure
- For each word keep
  - Number of appearing documents
  - Overall number of appearances
  - For each document
    - Number of appearances
    - Location


Token
DocCnt
FreqCnt
Head
ABANDON
28
51
ABIL
32
37
ABSENC
135
185
…
ABSTRACT
7
10
…
POSTING
DocNo
Freq
Word Position
67
2
279 283
424
1
24
1376
7
137 189 481…
..
206
1
170
4819
2
4 26 32
..

Full text search engine

- Most popular: Apache Lucene/Solr
- full-text search, hit highlighting, real-time indexing, dynamic clustering, database integration, NoSQL features, rich document (e.g., Word, PDF) handling.
- distributed search and index replication, scalability and fault tolerance.

Search with logical operators

- AND, OR, NOT
- jaguar AND car
  jaguar NOT animal
- Some system support neighborhood search (e.g., NEAR) and stemming (!)
  paris! NEAR(3) fr!
  president NEAR(10) bush
- libraries, concordancers
- E.g-, for Slovene: https://viri.cjvt.si/gigafida/

Logical operator search is outdated

- Large number of results
- Large specialized incomprehensible queries
- Synonyms
- Sorting of results
- No partial matching
- No weighting of query terms

Ranking based search

- Web search
- Less frequent terms are more informative
- NL input - stop words, lemmatization
- Vector based representation of documents and queries (bag-of-words or dense embeddings)

Sparse vector representation: bag-of-words

- *An elephant is a mammal. Mammals are animals. Humans are mammals, too. Elephants and humans live in Africa.*

Africa	animal	be	elephant	human	in	live	mammal	too
1	1	3	2	2	1	1	3	1

9 dimensional vector (1,1,3,2,2,1,1,3,1)

In reality this is sparse vector of dimension $|V|$ (vocabulary size in order of 10,000 dimensions)

Similarity between documents and queries in vector space.

Vectors and documents

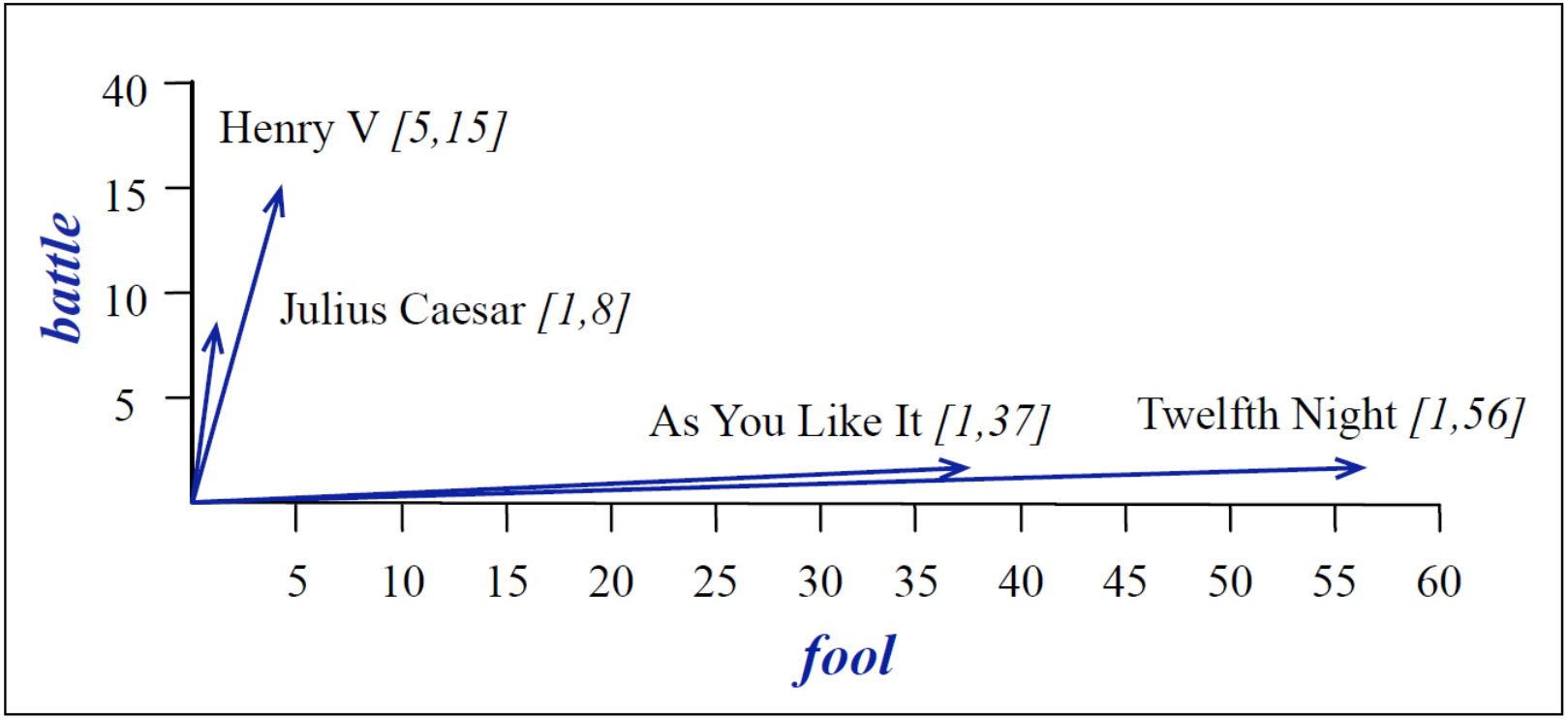
- a word occurs in several documents
- both words and documents are vectors
- an example: Shakespeare

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	1	1	8	15
soldier	2	2	12	36
fool	37	58	1	5
clown	5	117	0	0

- term-document matrix, dimension |V| x |D|
- a sparse matrix
- word embedding

Vector based similarity

- e.g., in two dimensional space



- the difference between dramas and comedies

Document similarity

- Assume orthogonal dimensions
- Cosine similarity
- Dot (scalar) product of vectors

$$\cos(\Theta) = \frac{A \cdot B}{|A||B|}$$

Importance of words

- Frequencies of words in particular document and overall
- inverse document frequency idf
  - N = number of documents in collection
  - n_b = number of documents with word b

$$idf_b = \log(\frac{N}{n_b})$$

Weighting dimensions (words)

- Weight of word b in document d

$$w_{b,d} = tf_{b,d} \times idf_{b,d}$$

$tf_{b,d}$ = frequency of term b in document d

- called TF-IDF weighting (an improvement over bag-of-words)

Weighted similarity

- Between query and document

$$sim(q,d) = \frac{\sum_b w_{b,d} \cdot w_{b,q}}{\sqrt{\sum_b w_{b,d}^2} \cdot \sqrt{\sum_b w_{b,q}^2}}$$

- Ranking by the decreasing similarity

Performance measures for search

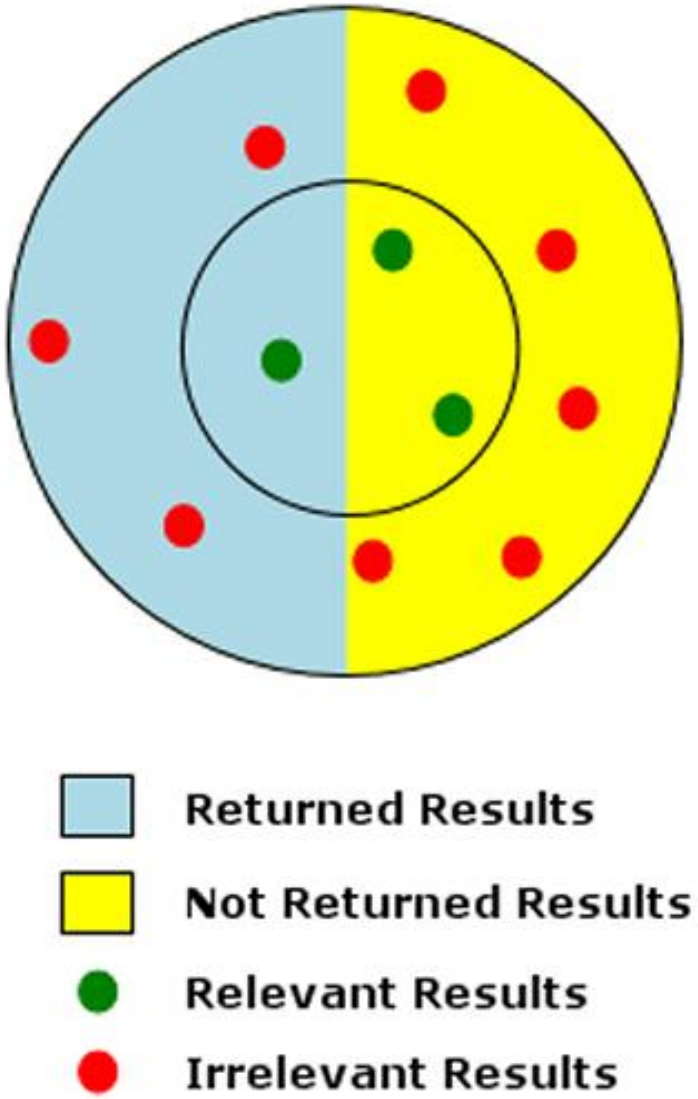
- Statistical measures
- Subjective measures
- Precision, recall
- A contingency table analysis of precision and recall

	Relevant	Non-relevant	
Retrieved	a	b	$a + b = m$
Not retrieved	c	d	$c + d = N - m$
	$a + c = n$	$b + d = N - n$	$a + b + c + d = N$

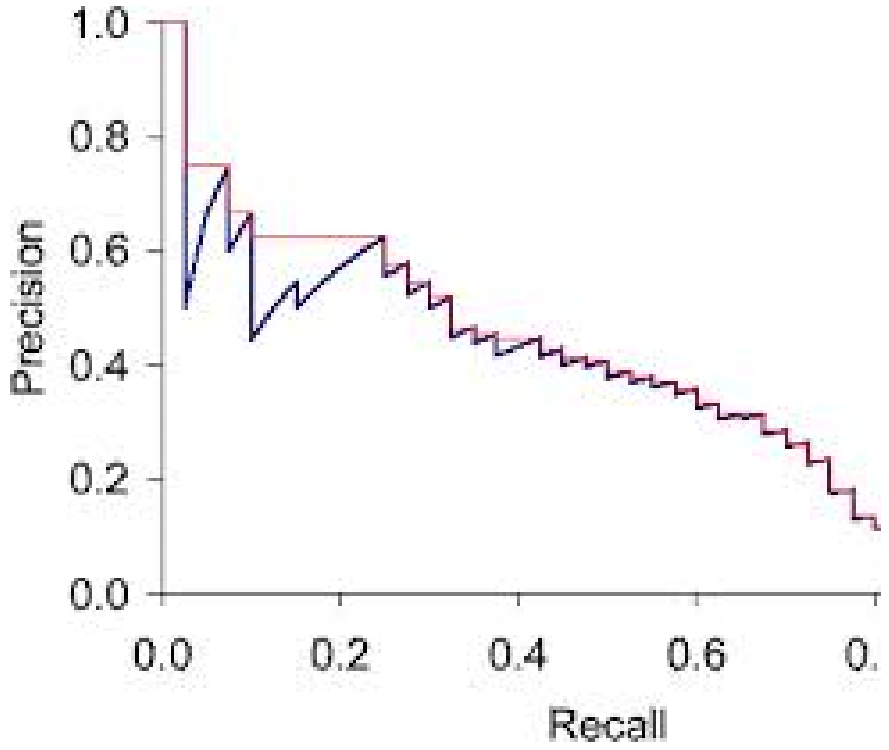
Precision and recall

- N = number of documents in collection
- n = number of important documents for given query q
- Search returns m documents including a relevant ones
- Precision $P = a/m$
  proportion of relevant document in the obtained ones
- Recall $R = a/n$
  proportion of obtained relevant documents
- Precision recall graphs

An example: low precision, low recall



Precision-recall graphs



F-measure

- combine both P and R

$$F_\beta = \frac{(1+\beta^2)\cdot P \cdot R}{\beta^2 P + R} \text{ for } \beta > 0$$

$$F_1 = \frac{2 \cdot P \cdot R}{P + R}$$

- Weighted precision and recall
- β=1 weighted harmonic mean
- Also used β=2 or β= 0.5

Ranking measures

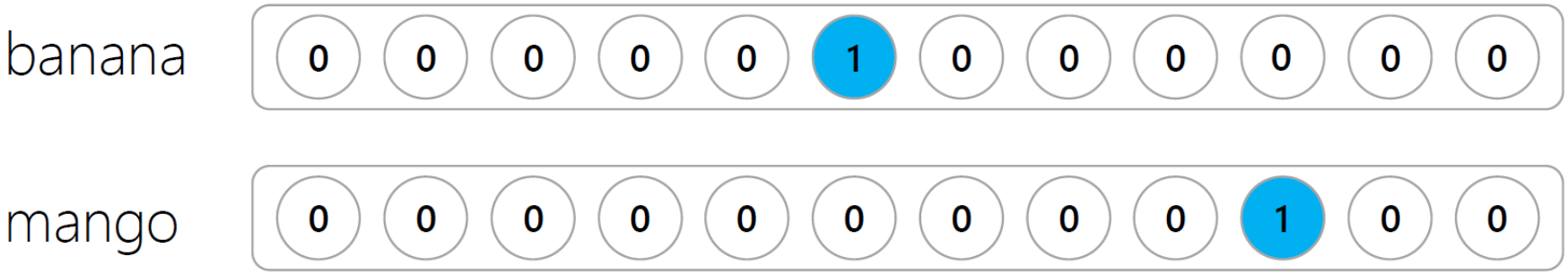
- precision@k
  - proportion of relevant document in the first k obtained ones
- recall@k
  - proportion of relevant documents in the k obtained among all relevant
- F_1@k
- mean reciprocal rank

$$MRR = \frac{1}{Q}\sum_{i=1}^{Q}\frac{1}{\mathrm{rank}_i}$$

  - over Q queries,
  - considers only the rank of the best answer

Why dense textual embeddings?

- Best machine learning models for text, i.e. deep neural networks, require numerical input.
- Simple representations like 1-hot-encoding and bag-of-words do not preserve semantic similarity.
- We need dense vector represenation for text elements.



Dense vector embeddings

- advantages compared to sparse embeddings:
  - less dimensions, less space
  - easier input for ML methods
  - potential generalization and noise reduction
  - potentially captures synonymy, e.g., road and highway are different dimensions in BOW
- the most popular approaches
  - matrix based transformations to reduce dimensionality (SVD in LSA)
  - neural embeddings (word2vec, Glove)
  - explicit contextual neural embeddings (ELMo, BERT)
  - only use an implicit representation in LLM

Meaning focused on similarity

- Each word is a vector
- Similar words are "nearby in space"



Distributional semantics



"You shall know a word
by the company it keeps"

Firth, J. R. (1957). A synopsis of linguistic theory 1930–1955. In Studies in Linguistic Analysis, p. 11. Blackwell, Oxford.



"The meaning of a word is its use in the language"

Ludwig Wittgenstein, PI #43

Neural embeddings

- neural network is trained to predict the context of words (input: word, output: context of neighboring words)

word2vec method

- Train a classifier on a binary **prediction** task:
  Is word *w* likely to show up near a given word, e.g., "*apricot*"?
- We don't actually care about this task
- But we'll take the learned classifier weights as the word embeddings

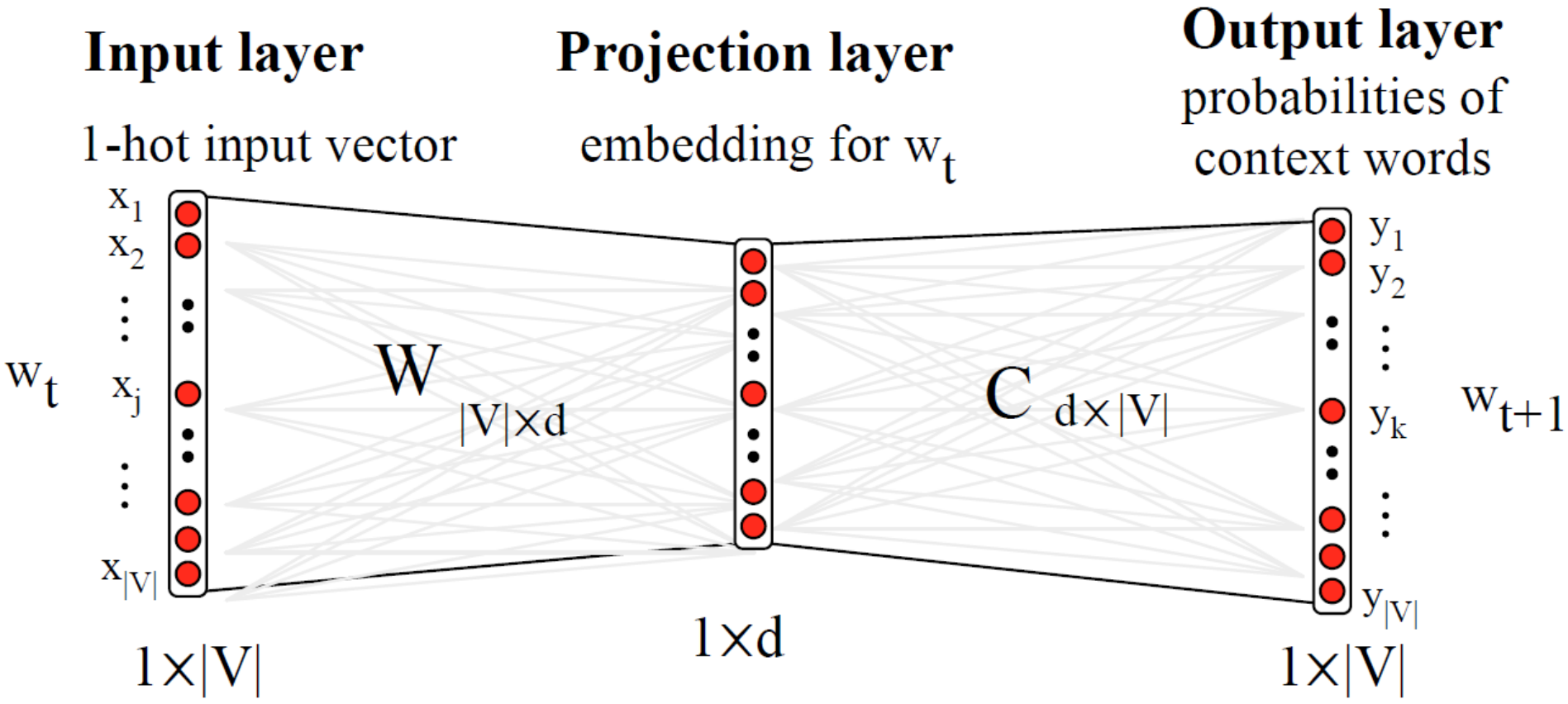
- Words near apricot acts as 'correct answers' to the question "Is word w likely to show up near apricot?"
- No need for hand-labeled supervision

word2vec (skip-gram) training data

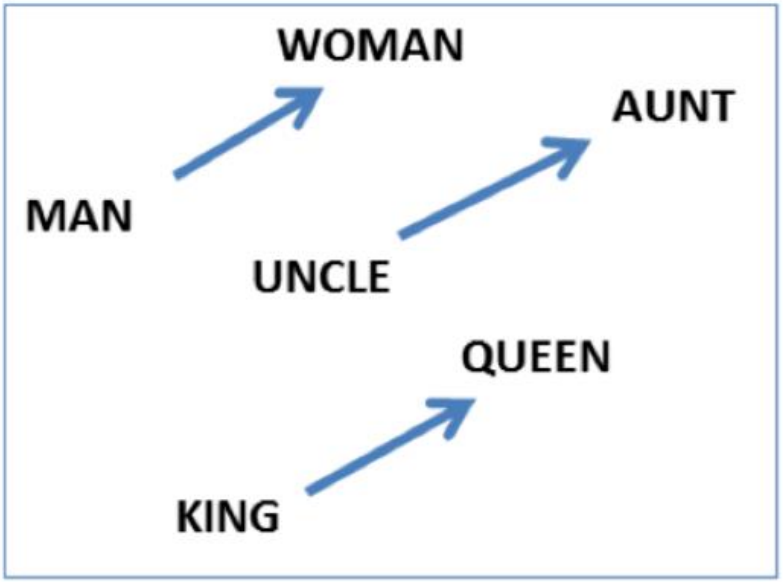
- Training sentence:
- ... lemon, a tablespoon of **apricot** jam a pinch ...
- c1 c2 target c3 c4

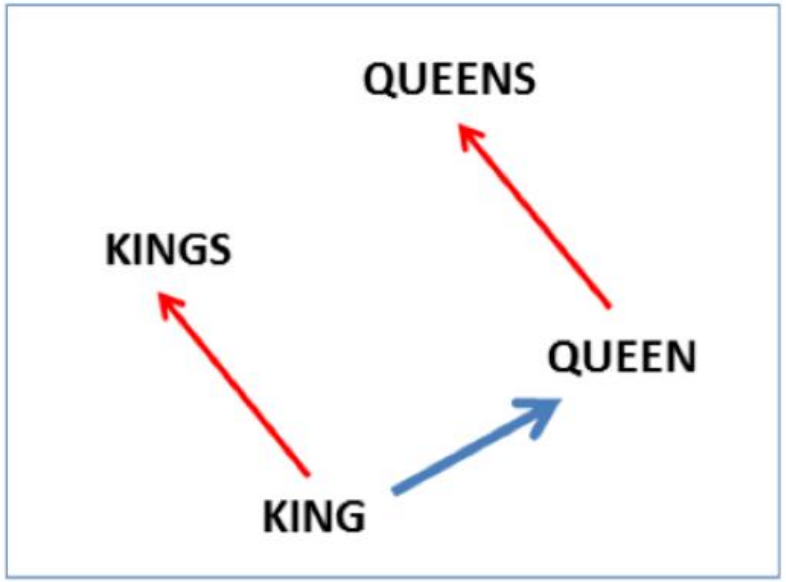
- Asssume context words are those in +/- 2 word window
- Produce the following input-outputs for positive instances: (apricot, tablespoon), (apricot, of), (apricot, jam), (apricot, a)
- Get negative training examples randomly
- Train a neural network to predict the probability of a co-occurring word

Neural network based embedding



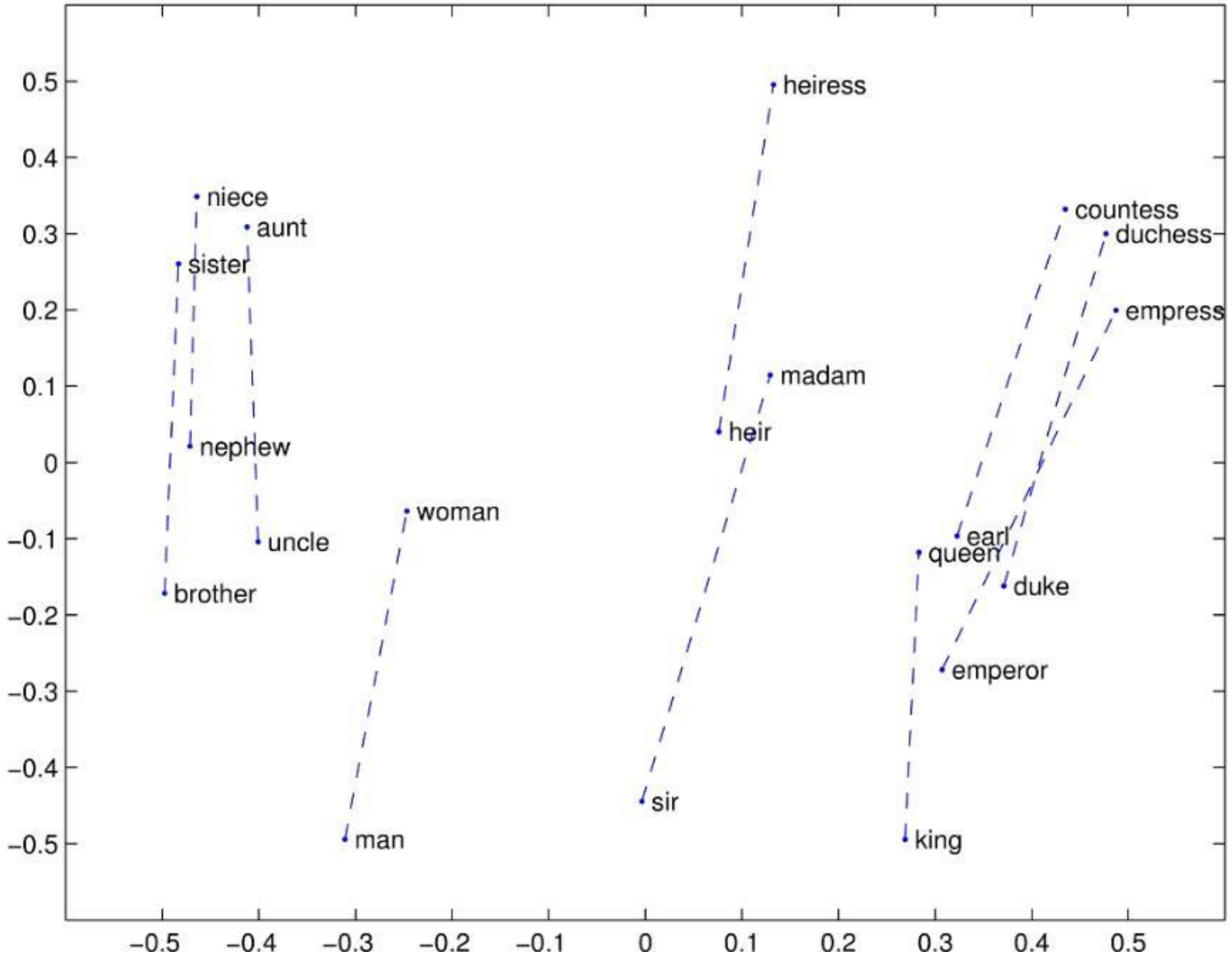
Relational similarity

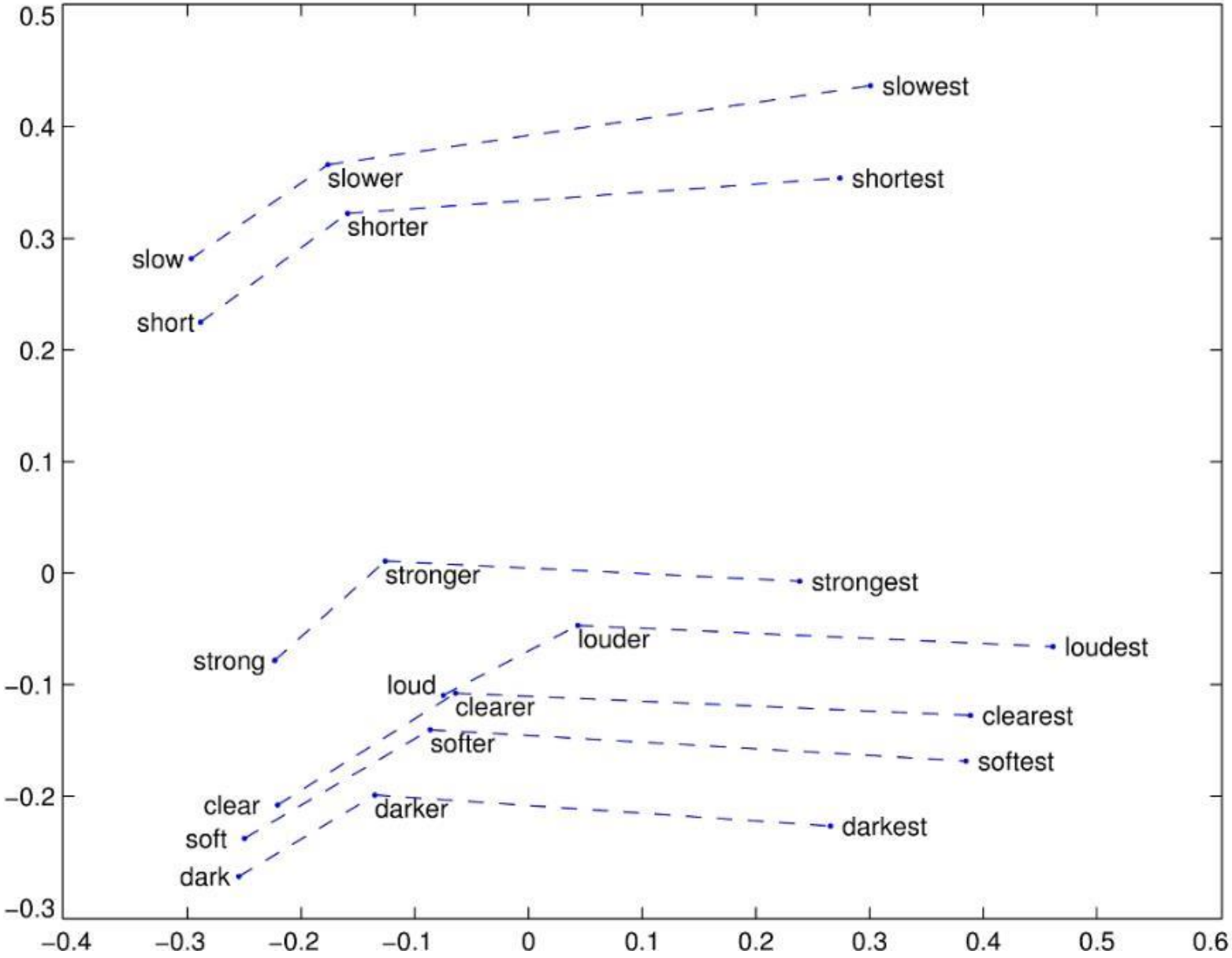




vector(‘*king*’) - vector(‘*man*’) + vector(‘*woman*’) ≈ vector(‘queen’)

vector(‘*Paris*’) - vector(‘*France*’) + vector(‘*Italy*’) ≈ vector(‘Rome’)

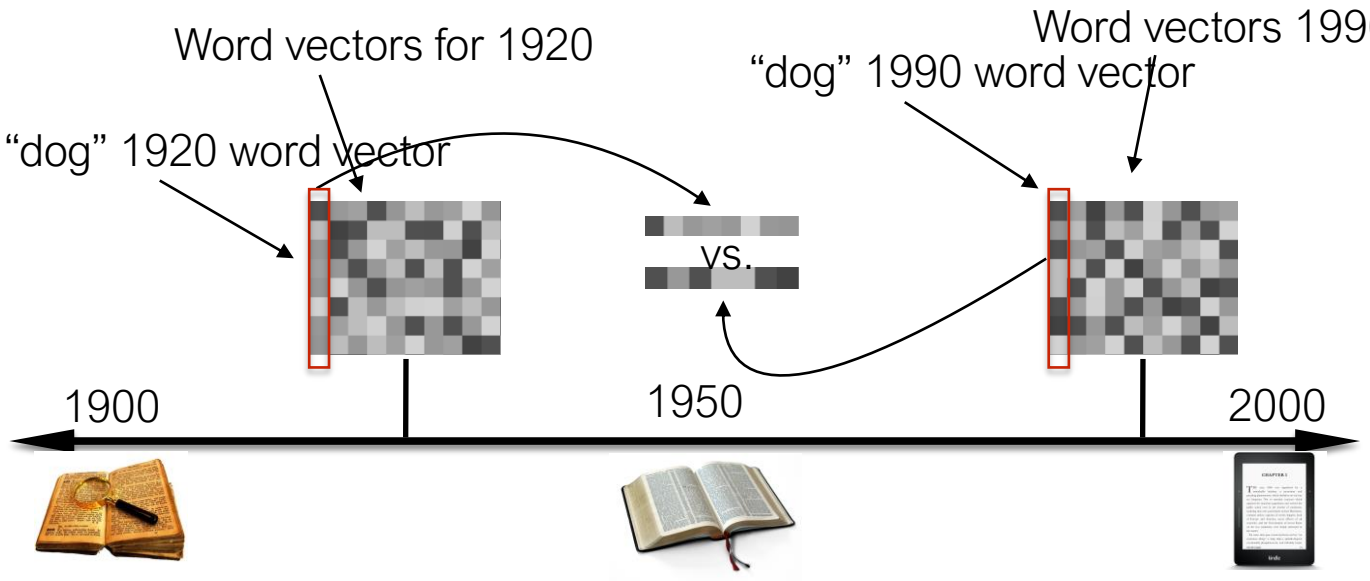

0.5
0.4
0.3
0.2
0.1
0
−0.1
−0.2
−0.3
−0.4
−0.5
−0.5
−0.4
−0.3
−0.2
−0.1
0
0.1
0.2
0.3
0.4
0.5
niece
aunt
sister
nephew
uncle
brother
woman
man
heiress
madam
heir
sir
countess
duchess
empress
earl
queen
duke
emperor
king


slowest
slower
shortest
shorter
slow
short
stronger
strongest
strong
louder
loudest
loud
clearer
clearest
softer
softest
clear
darker
darkest
soft
dark
0.5
0.4
0.3
0.2
0.1
0
−0.1
−0.2
−0.3
−0.4
−0.3
−0.2
−0.1
0
0.1
0.2
0.3
0.4
0.5
0.6

Embeddings can help study word history

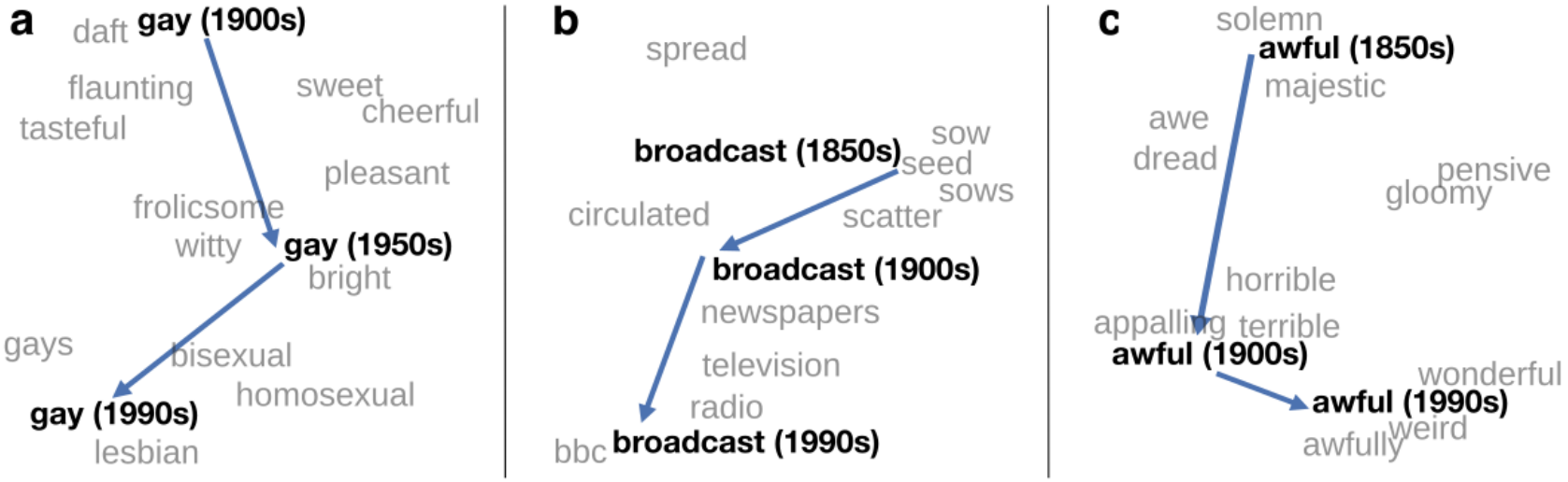
- Train embeddings on old books to study changes in word meaning!!

Diachronic word embeddings for studying language change



Visualizing changes

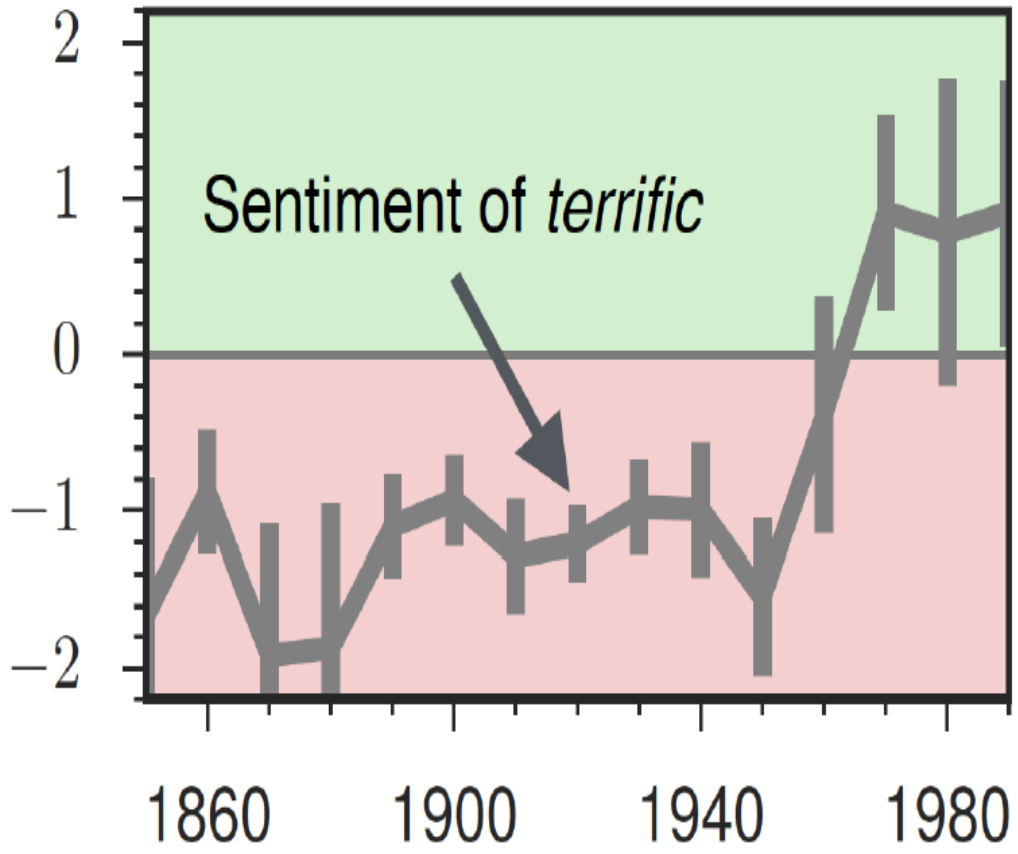
Project 300 dimensions down into 2



~30 million books, 1850-1990, Google Books data

The evolution of sentiment words

Negative words change faster than positive words



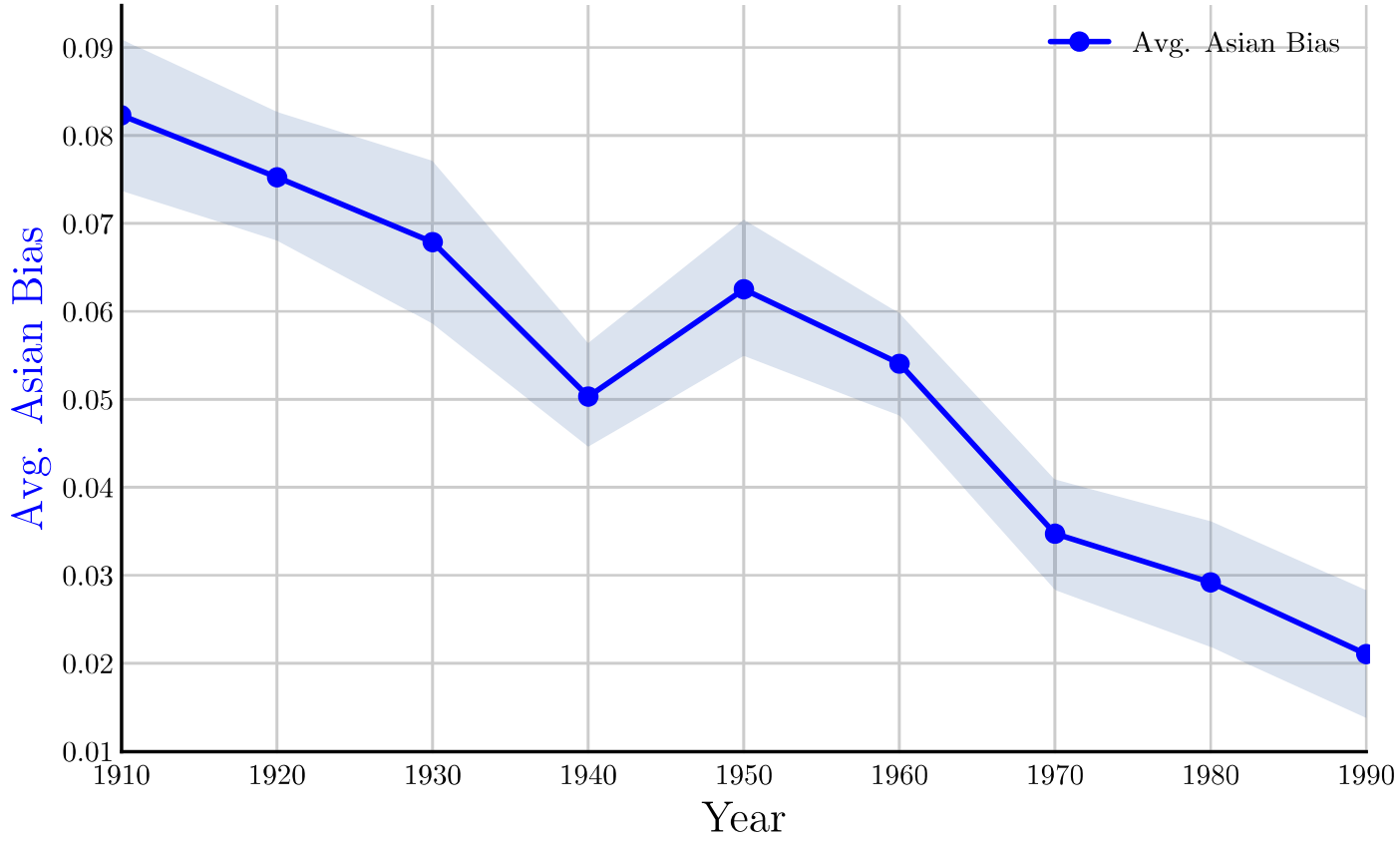
Embeddings reflect cultural bias

- Ask “Paris : France :: Tokyo : x”
  - x = Japan
- Ask “father : doctor :: mother : x”
  - x = nurse
- Ask “man : computer programmer :: woman : x”
  - x = homemaker

Bolukbasi, Tolga, Kai-Wei Chang, James Y. Zou, Venkatesh Saligrama, and Adam T. Kalai. "Man is to computer programmer as woman is to homemaker? debiasing word embeddings." In *Advances in Neural Information Processing Systems*, pp. 4349-4357. 2016.

Change in linguistic framing 1910-1990

Change in association of Chinese names with adjectives framed as "othering" (*barbaric*, *monstrous*, *bizarre*)



Garg, Nikhil, Schiebinger, Londa, Jurafsky, Dan, and Zou, James (2018). Word embeddings quantify 100 years of gender and ethnic stereotypes. *Proceedings of the National Academy of Sciences*, *115*(16), E3635–E3644

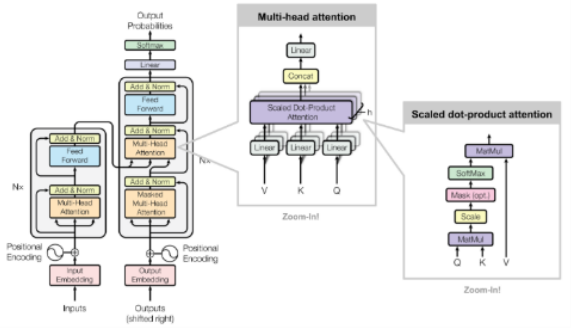
Contextual embeddings / Large language models

- word2vec produces the same vector for a word like bank irrespective of its meaning and context
- recent embeddings take the context into account
- already established as a standard
- e.g., BERT for contextual word embeddings

University of Ljubljana
Faculty of Computer and Information Science

Large language models (LLMs)

- pretrained neural large language models
- trained on large text corpora to capture relations in language
- finetuned to specific tasks
- many are publicly available
- on HuggingFace

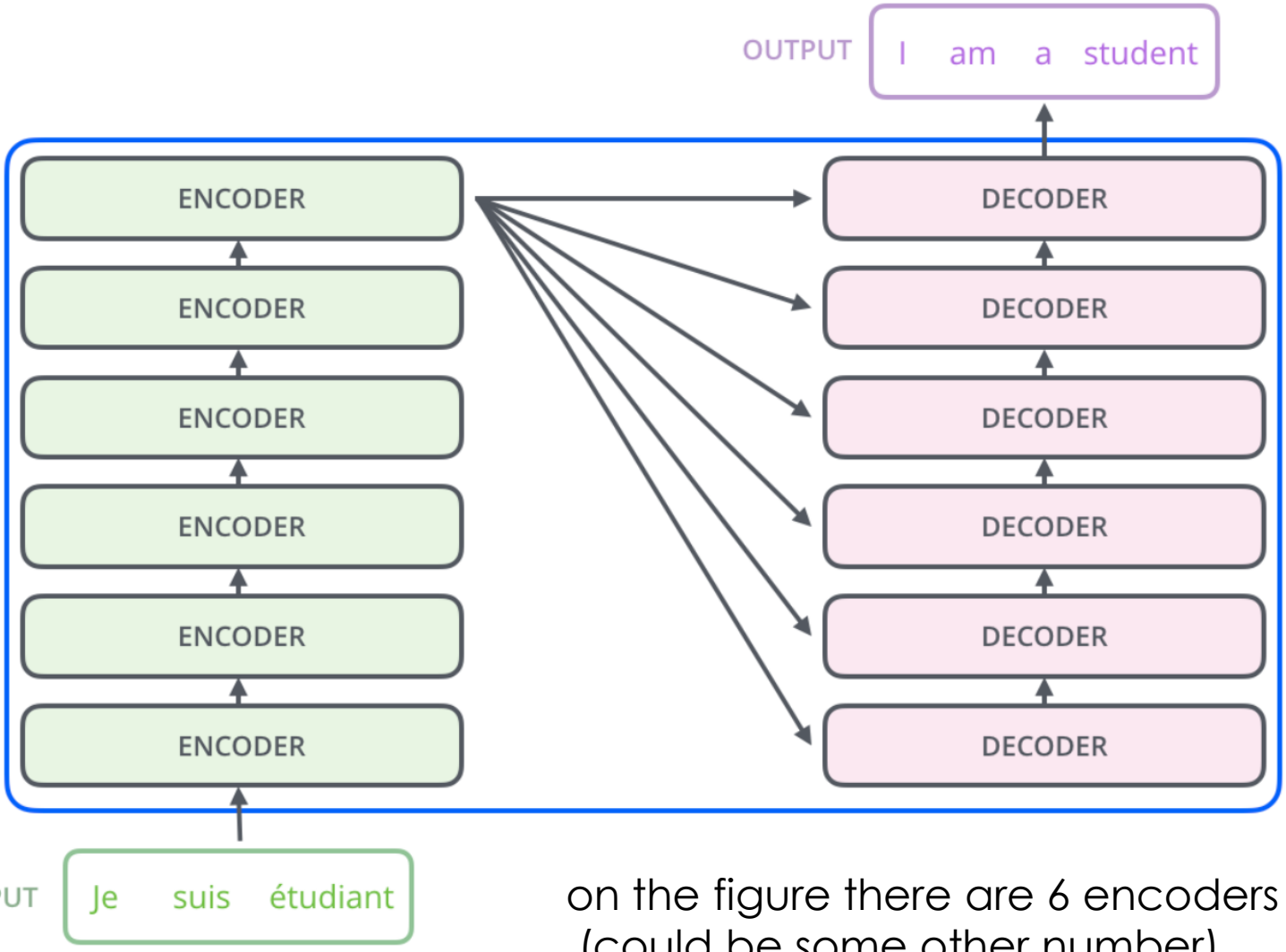


Transformer architecture of NNs

- currently the most successful DNN
- non-recurrent
- architecturally it is an encoder-decoder model
- fixed input length
- can be parallelized
- adapted for GPU (TPU) processing
- based on extreme use of attention

- https://github.com/dair-ai/Transformers-Recipe

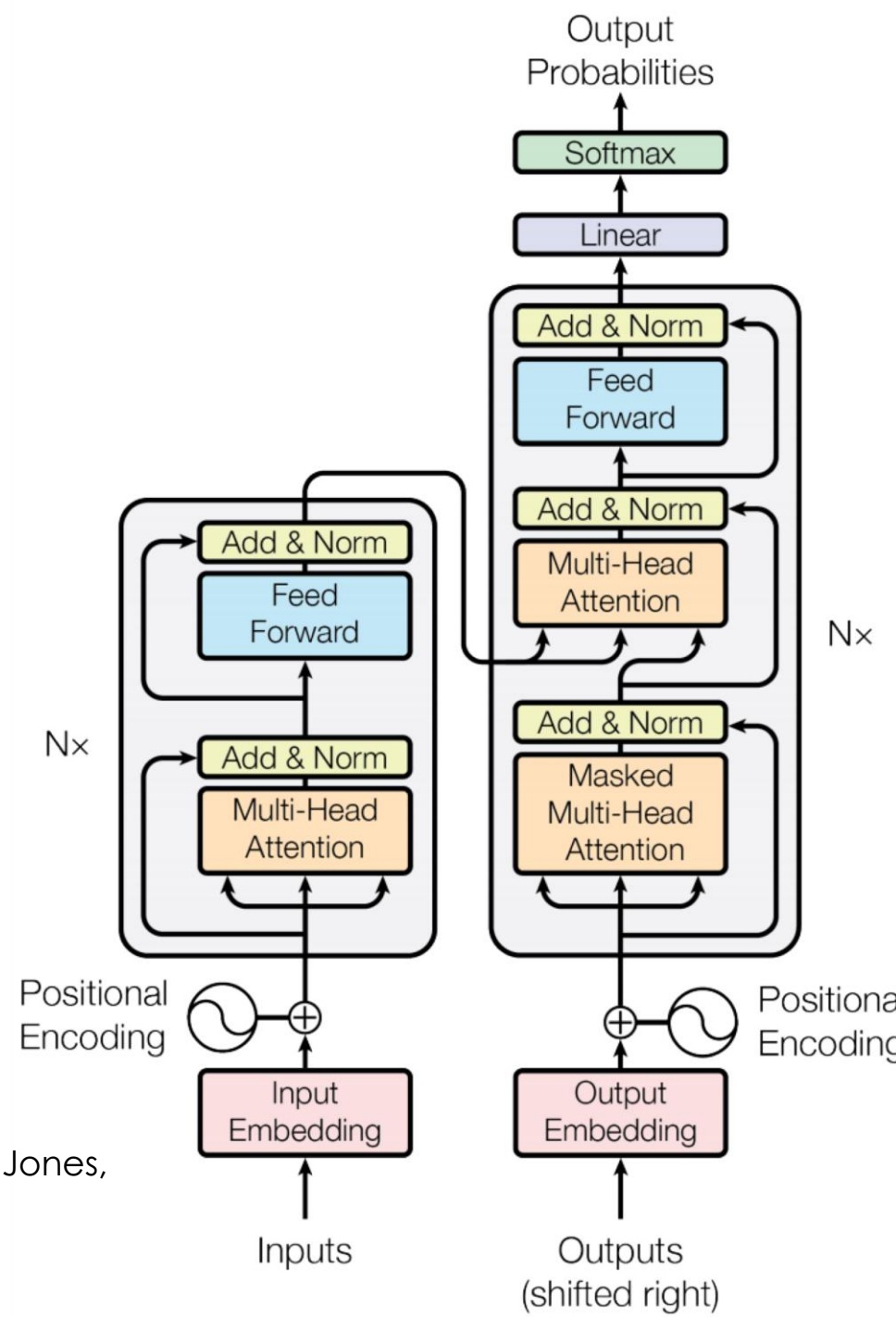
Transformer is an encoder-decoder model



on the figure there are 6 encoders and 6 decoders
(could be some other number)

Transformer overview

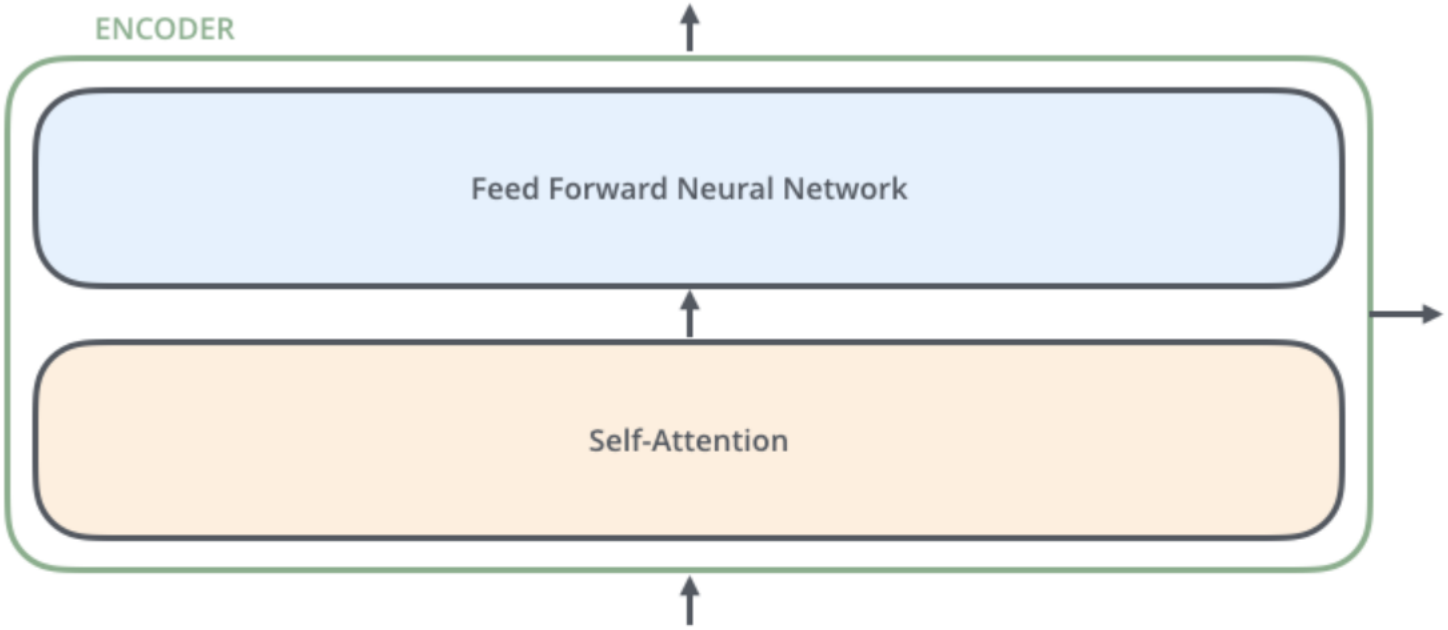
- Initial task: machine translation with parallel corpus
- Predict each translated word
- Final cost/loss/error function is standard cross-entropy error on top of a softmax classifier



Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, Ł. and Polosukhin, I., 2017. Attention is all you need. In *Advances in neural information processing systems* (pp. 5998-6008).

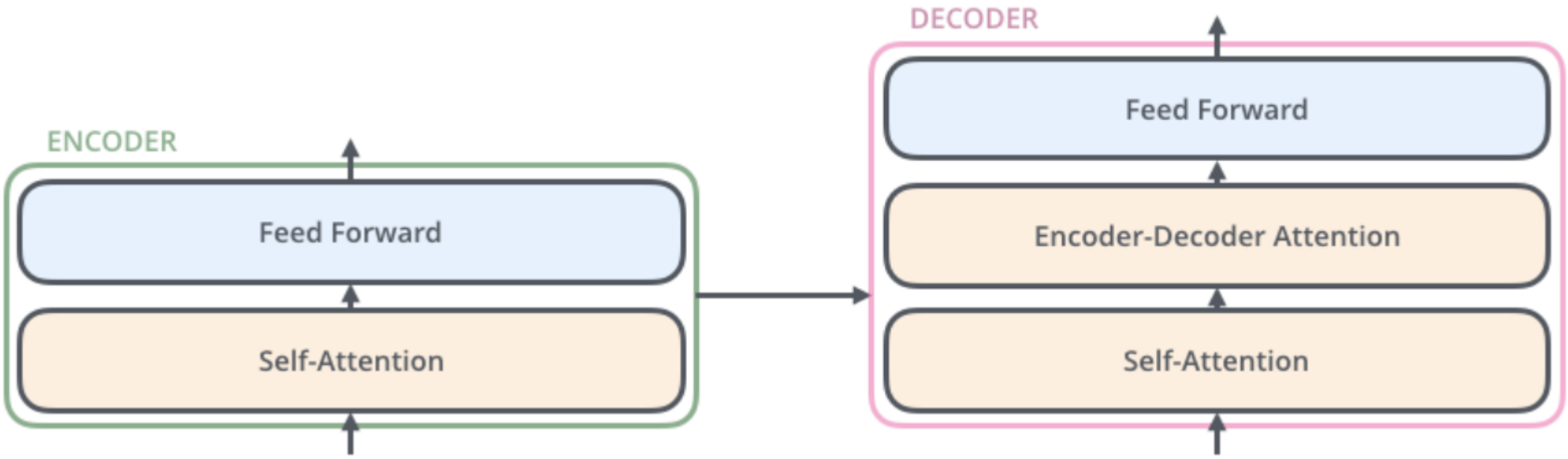
Transformer: encoder

- two layers
- no weight sharing between different encoders
- self-attention helps to focus on relevant part of input

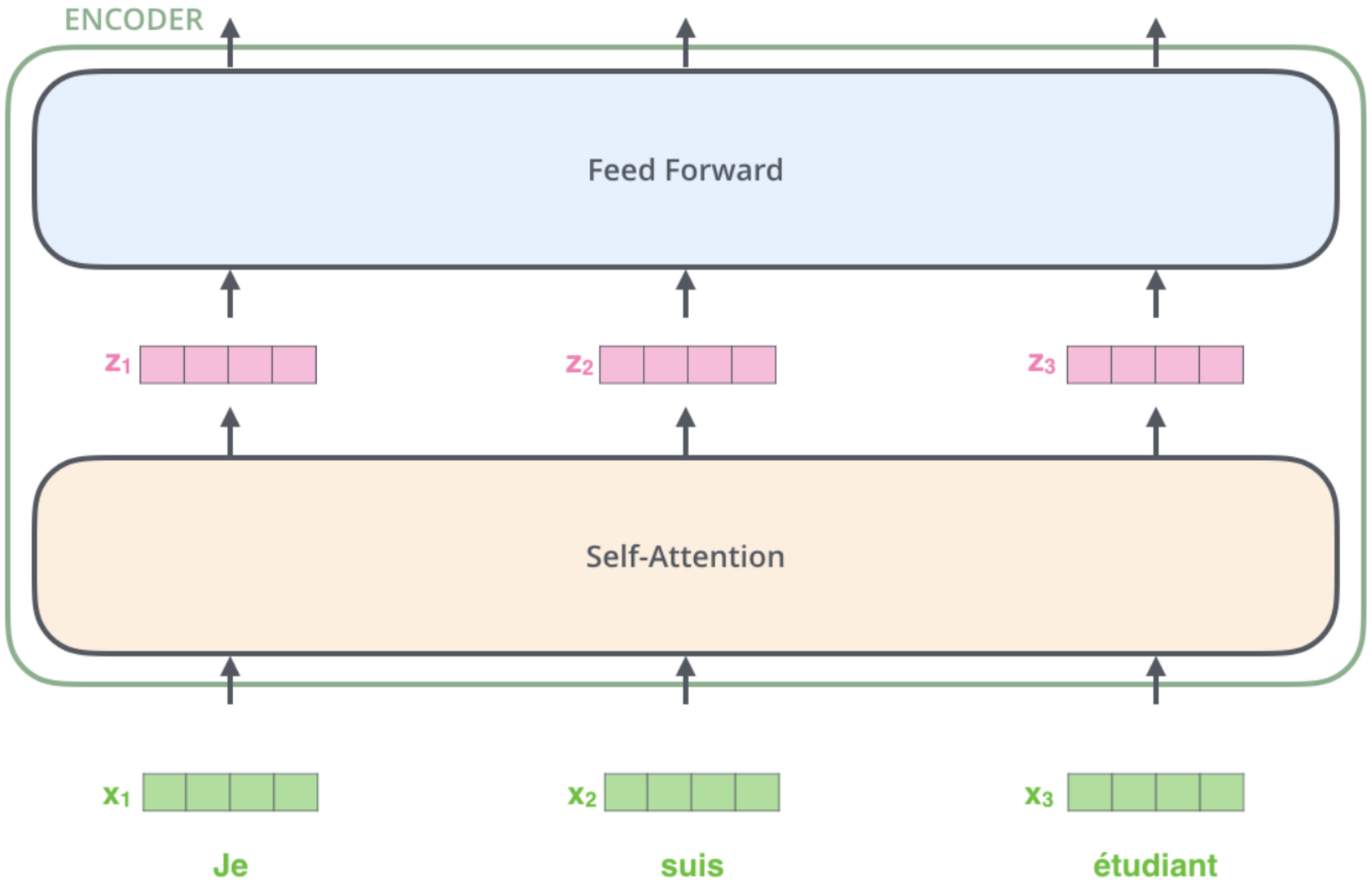


Transformer: decoder

- the same as encoder but with an additional attention layer in between, receiving input fom encoder



Start with embeddings

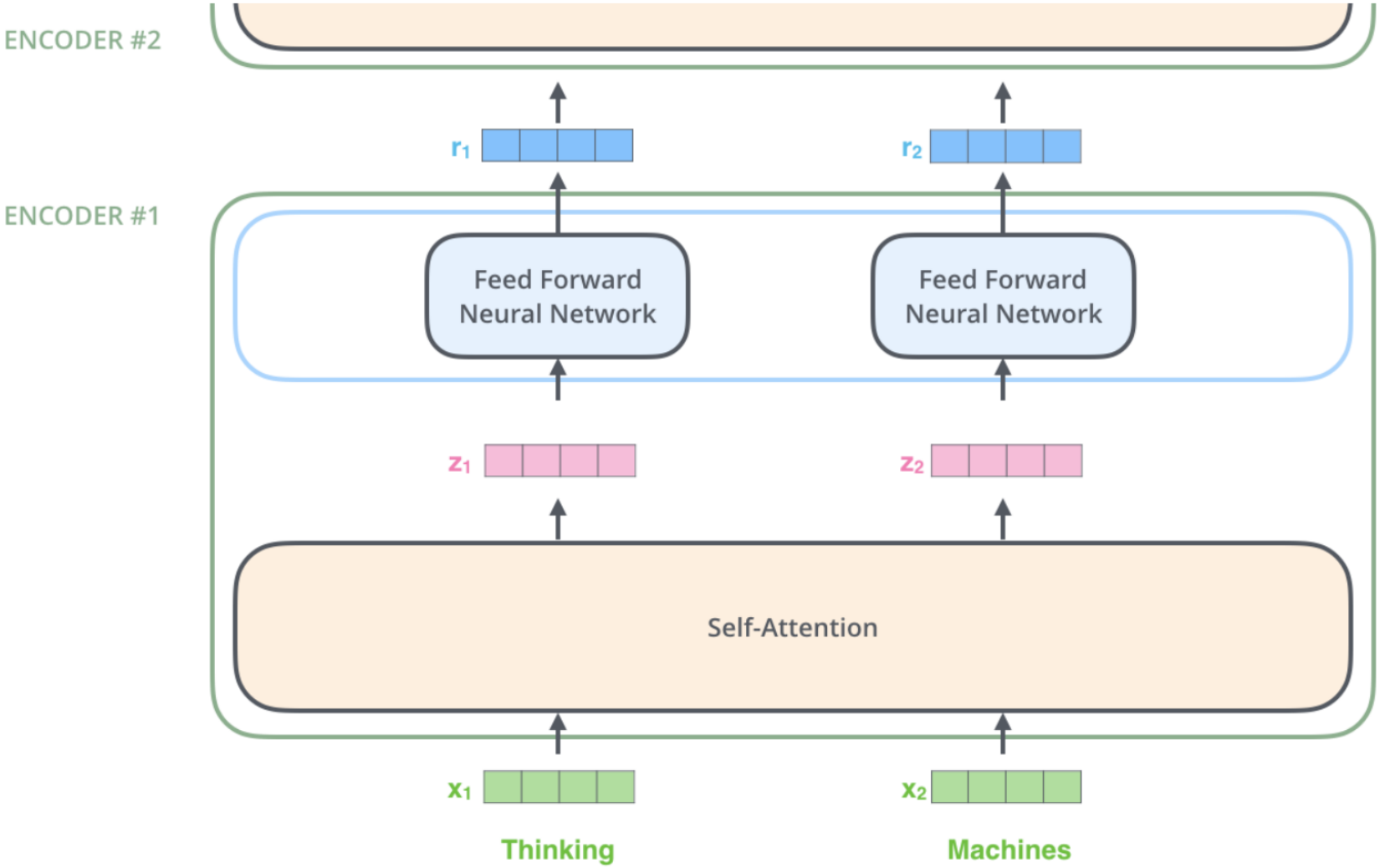


Input to transformer

- embeddings, e.g., 512 dimensional vectors (special, we will discuss that later)
- fixed length, e.g., max 128 tokens (in modern at least 8192)
- dependencies between inputs are only in the self-attention layer, no dependencies in feed-forward layer – good for parallelization

- Let us first present the working of the transformer with an illustration of the prediction

Encoding


ENCODER #2
ENCODER #1
r1
r2
Feed Forward Neural Network
Feed Forward Neural Network
z1
z2
Self-Attention
x1
x2
Thinking
Machines

Self-attention

- As the model processes each word (each position in the input sequence), self-attention allows it to look at other positions in the input sequence for clues that can help lead to a better encoding for this word

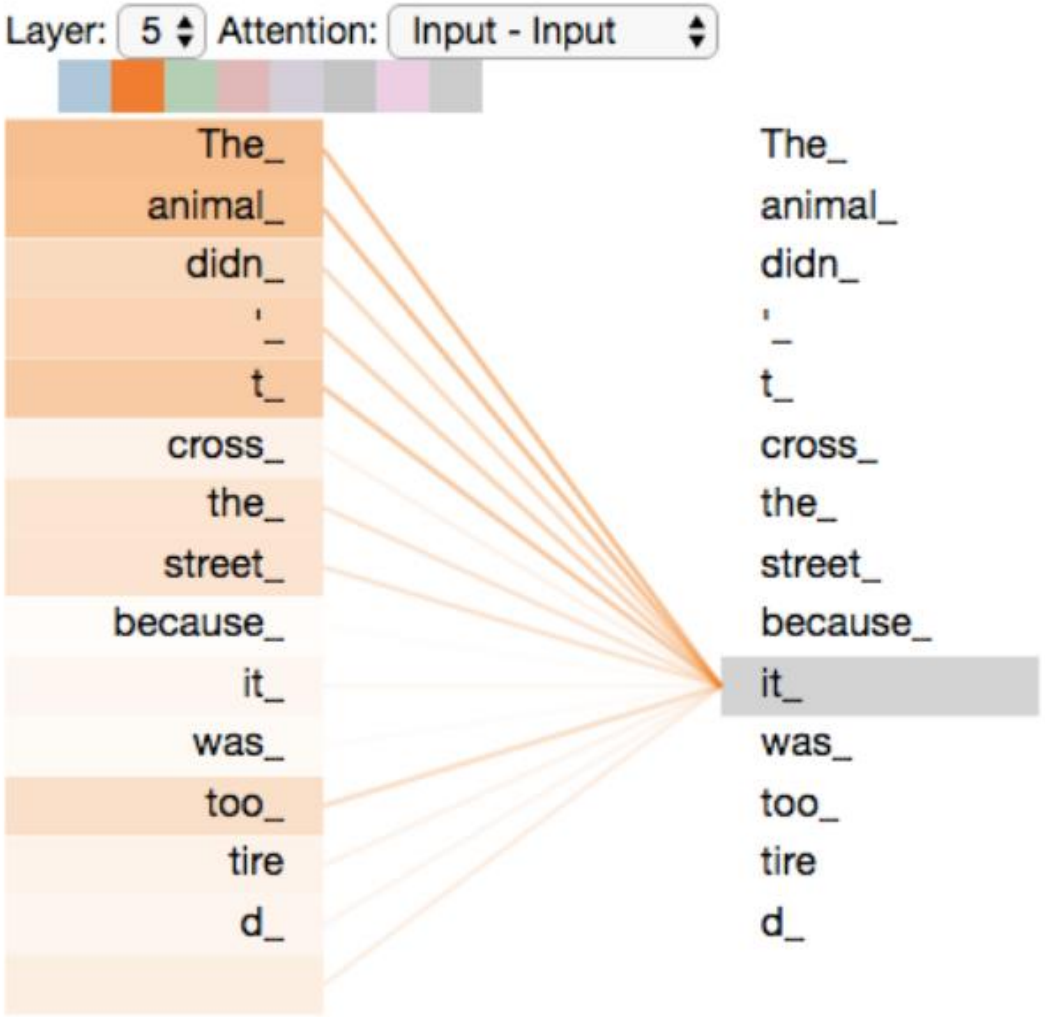
  ”The animal didn't cross the street because it was too tired”

- What does “it” in this sentence refer to? Is it referring to the street or to the animal? It’s a simple question to a human, but not as simple to an algorithm.
- *”The animal didn't cross the street because it was too wide”*

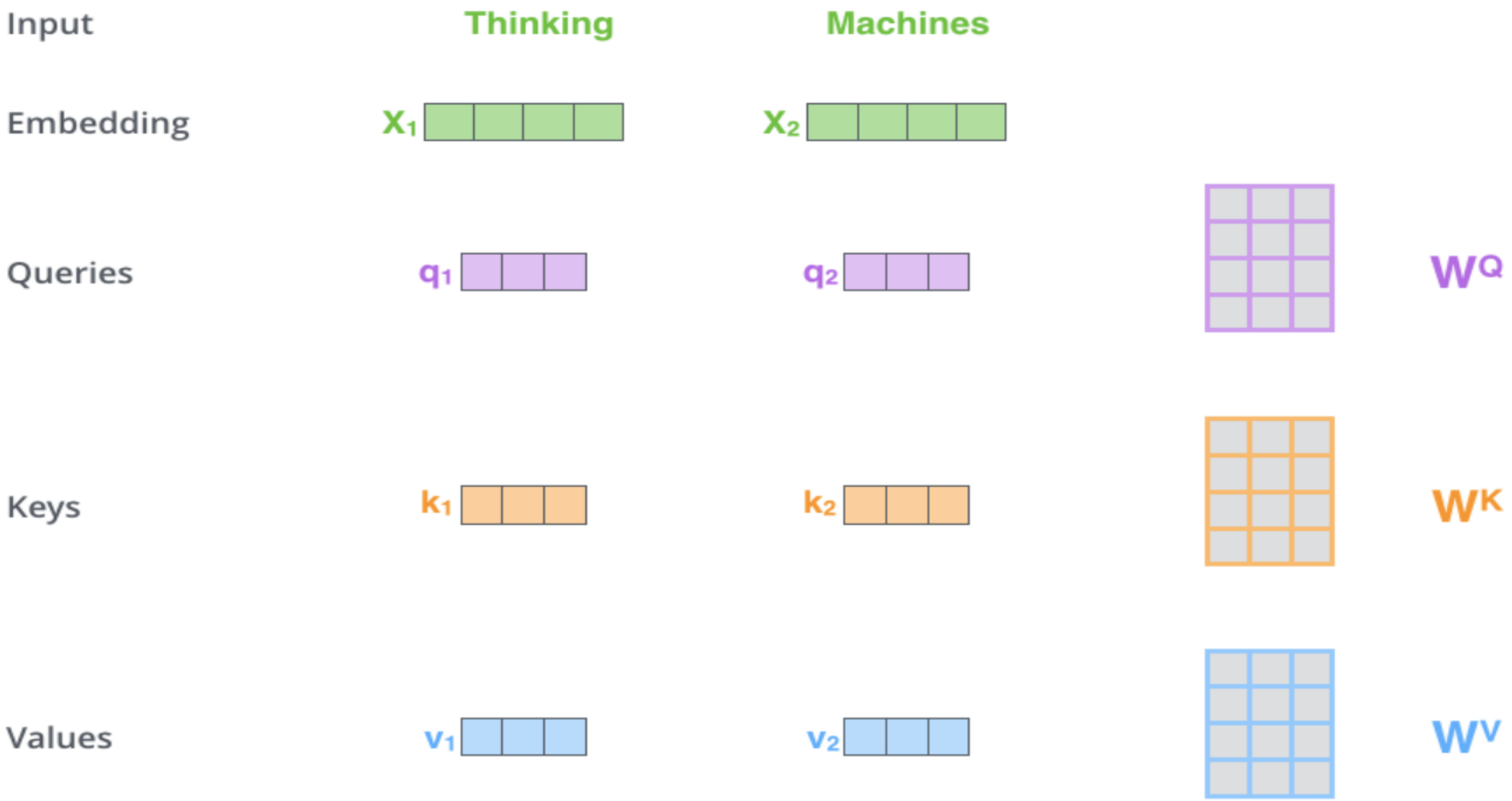
- When the model is processing the word “it”, self-attention allows it to associate “it” with “animal”.

Illustrating self-attention

- As we are encoding the word "it" in encoder #5 (the top encoder in the stack), part of the attention mechanism was focusing on "The Animal", and baked a part of its representation into the encoding of "it".

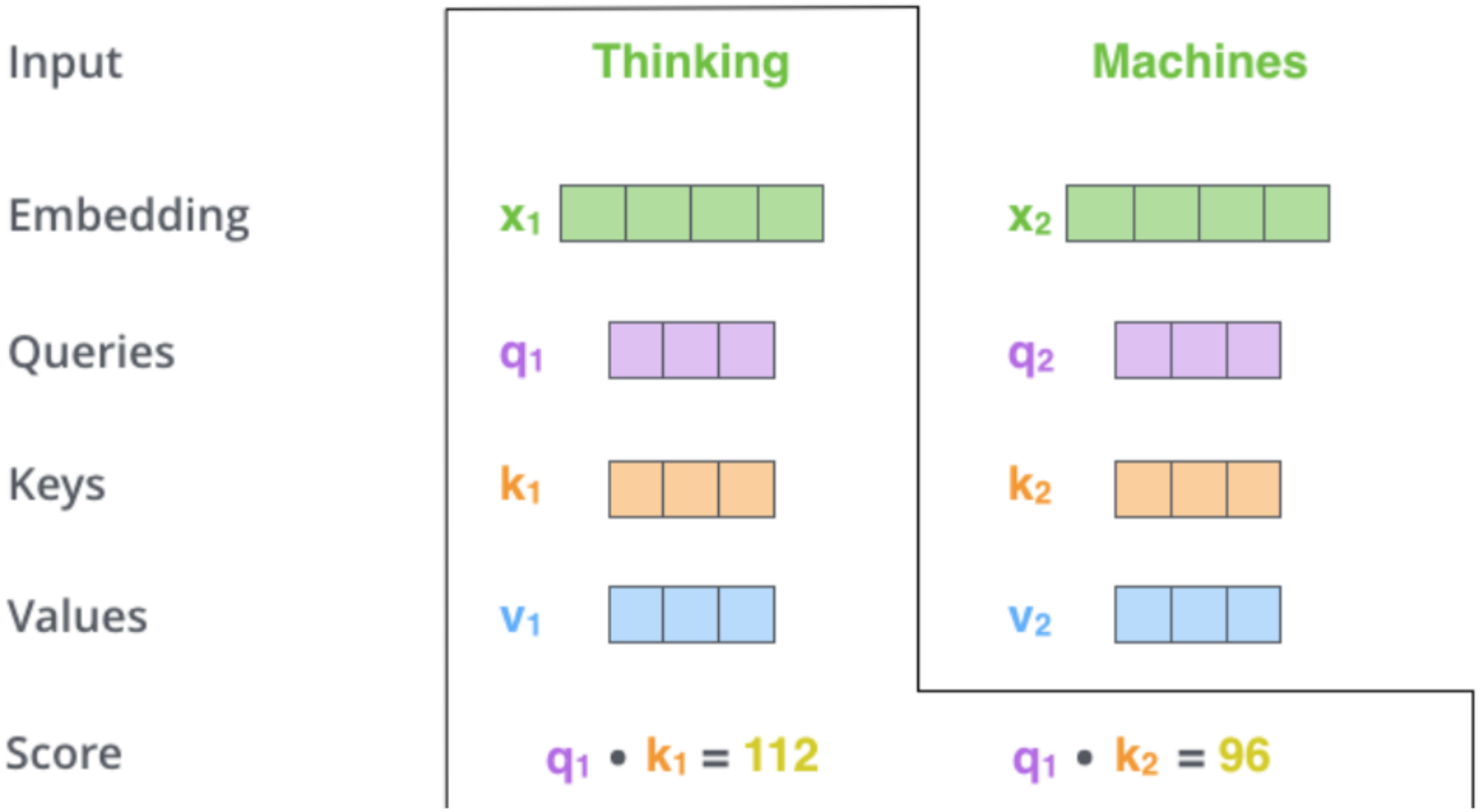


Self-attention details 1/4

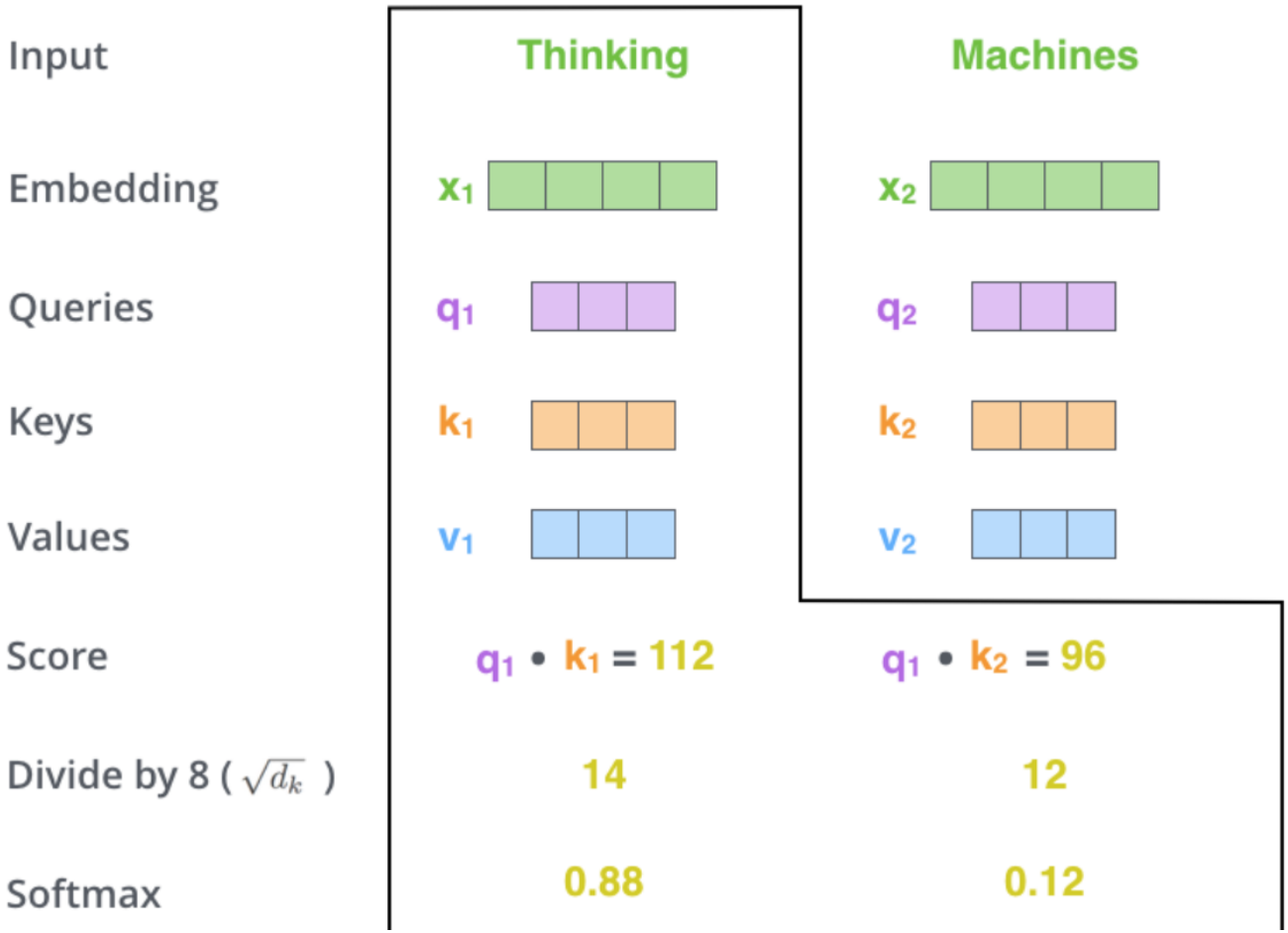


Multiplying x_1 by the W^Q weight matrix produces q_1, the "query" vector associated with that word. We end up creating a "query", a "key", and a "value" projection of each word in the input sentence.

Details 2/4: scoring



Details 3/4: normalization of scores



Details 4/4: self-attention output

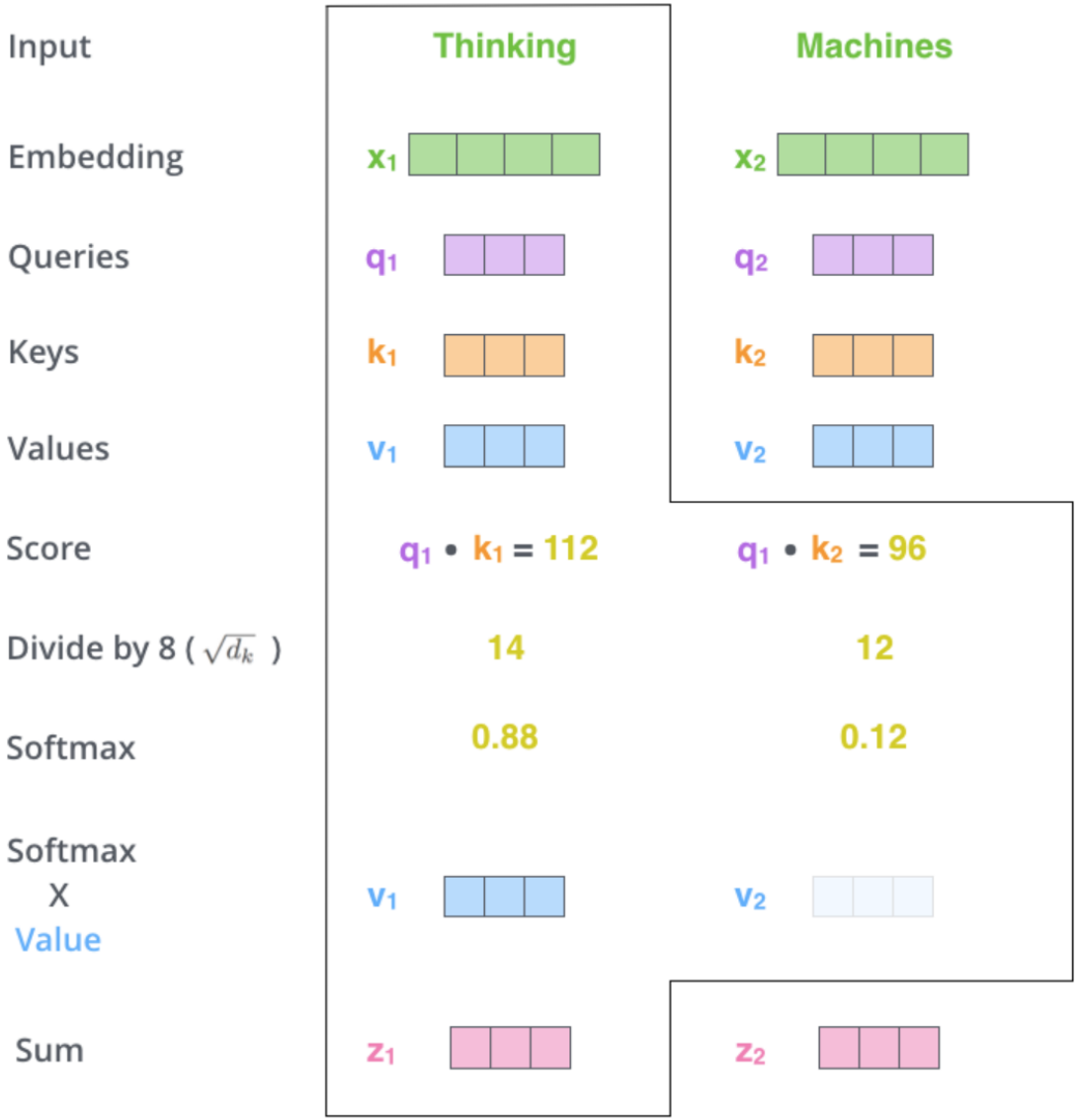
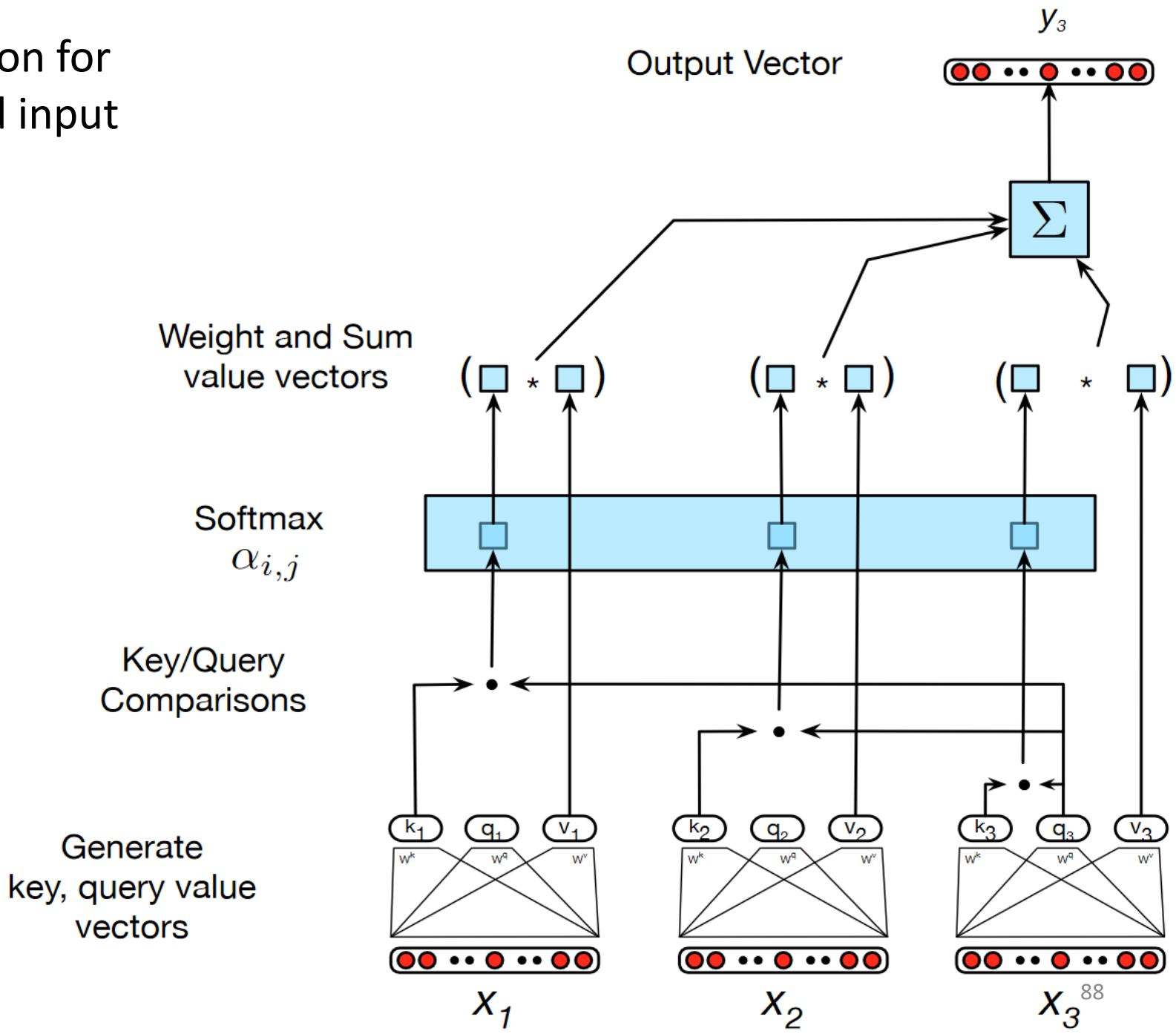
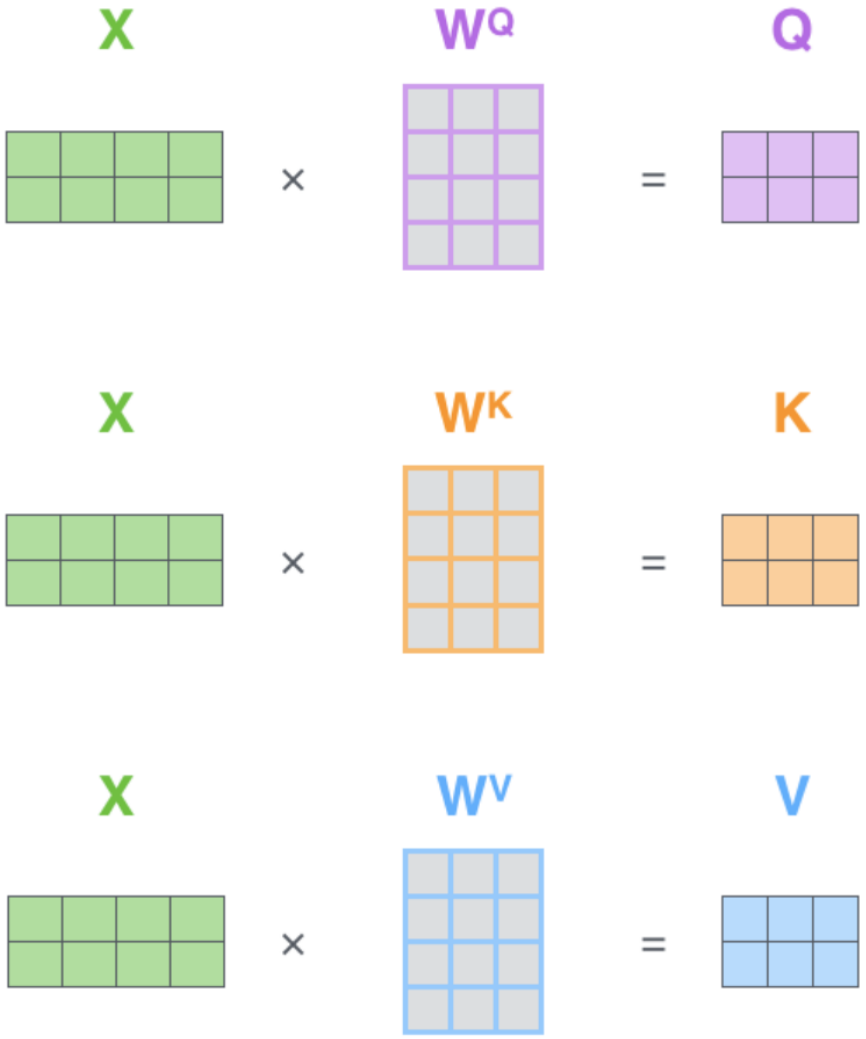


illustration for
the third input



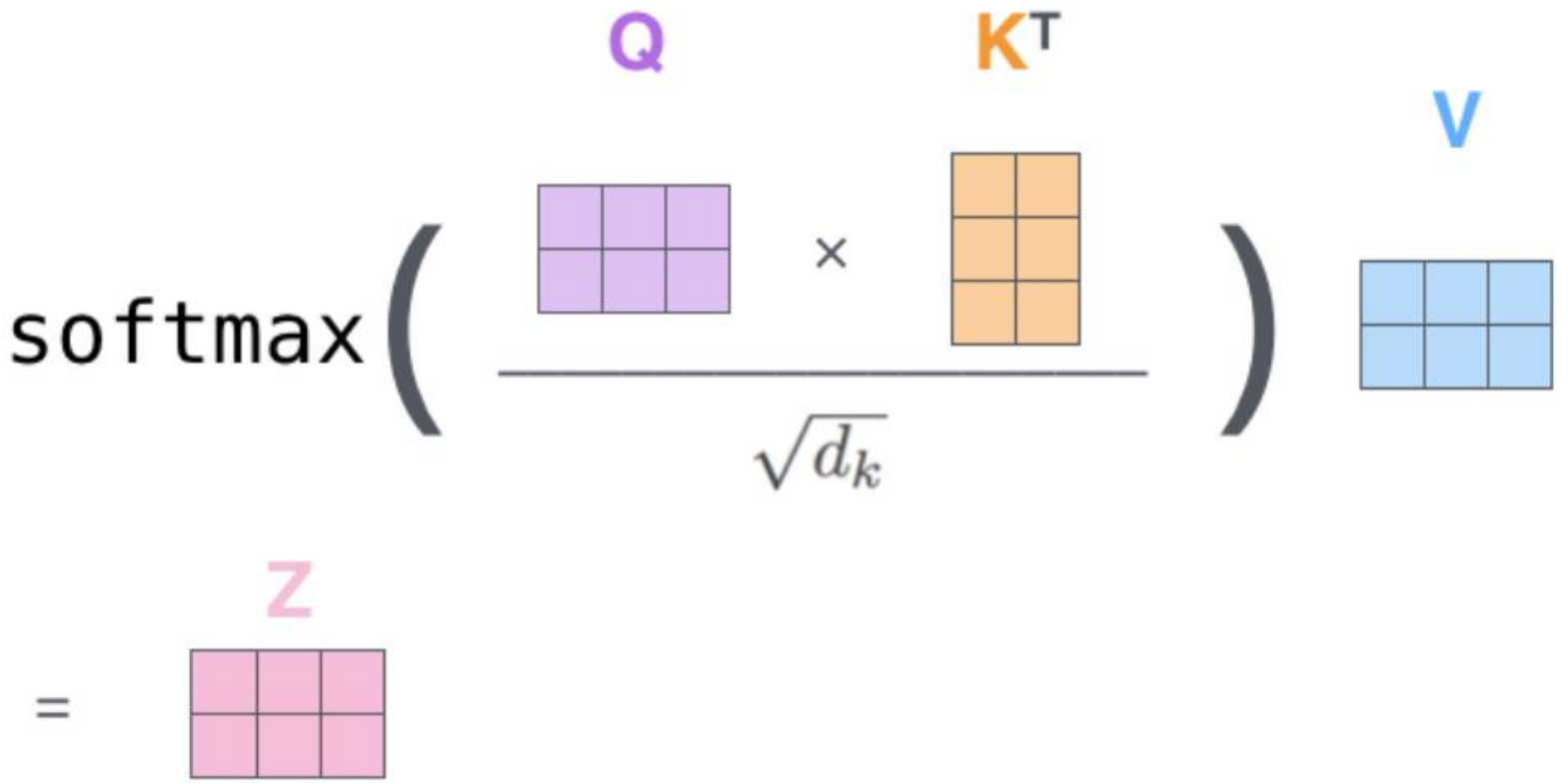
Matrix calculation of self-attention 1/2

Every row in the X matrix corresponds to a word in the input sentence.
The embedding vector x (512) is larger then the q/k/v vectors (64)

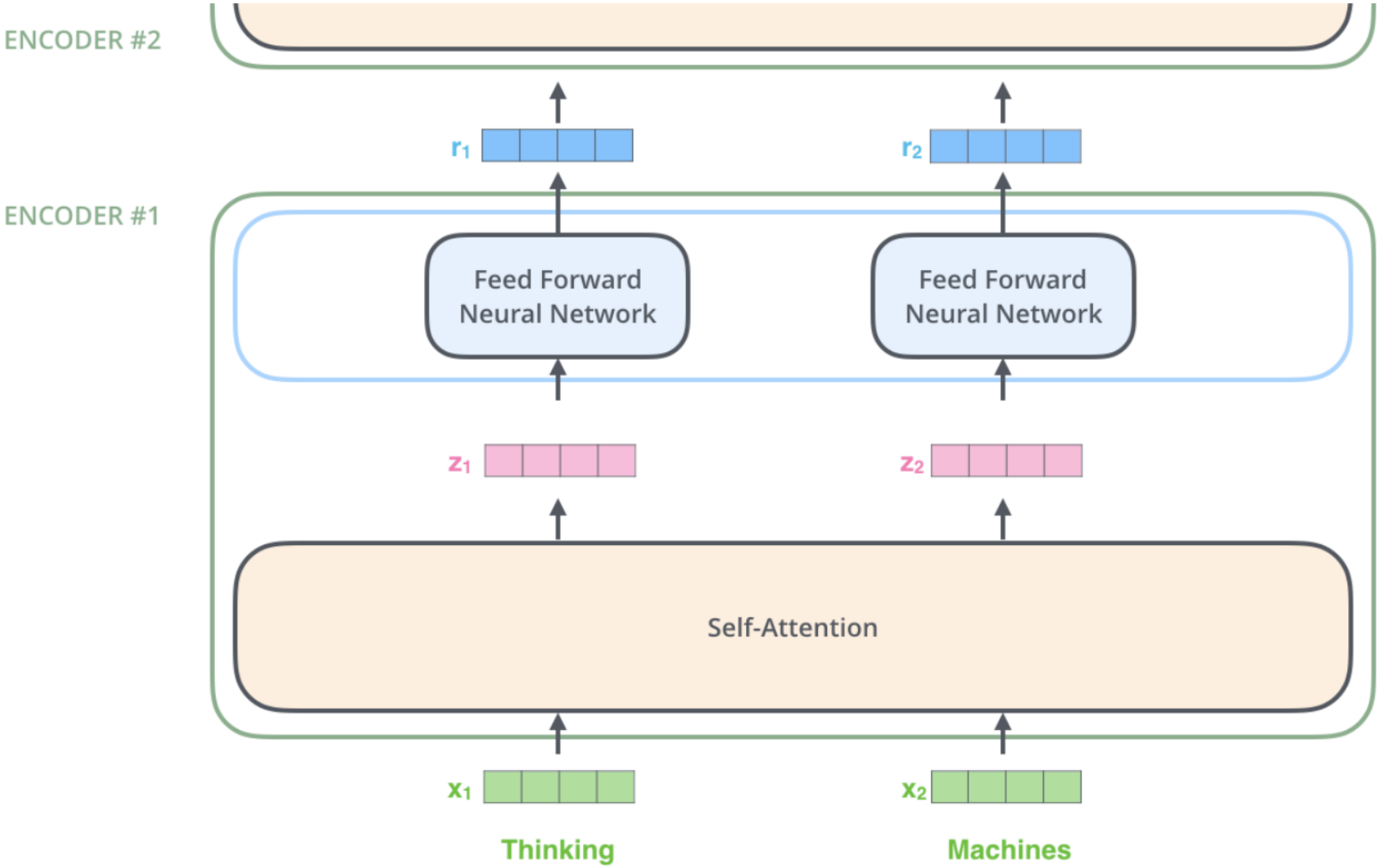


Matrix calculation of self-attention 2/2

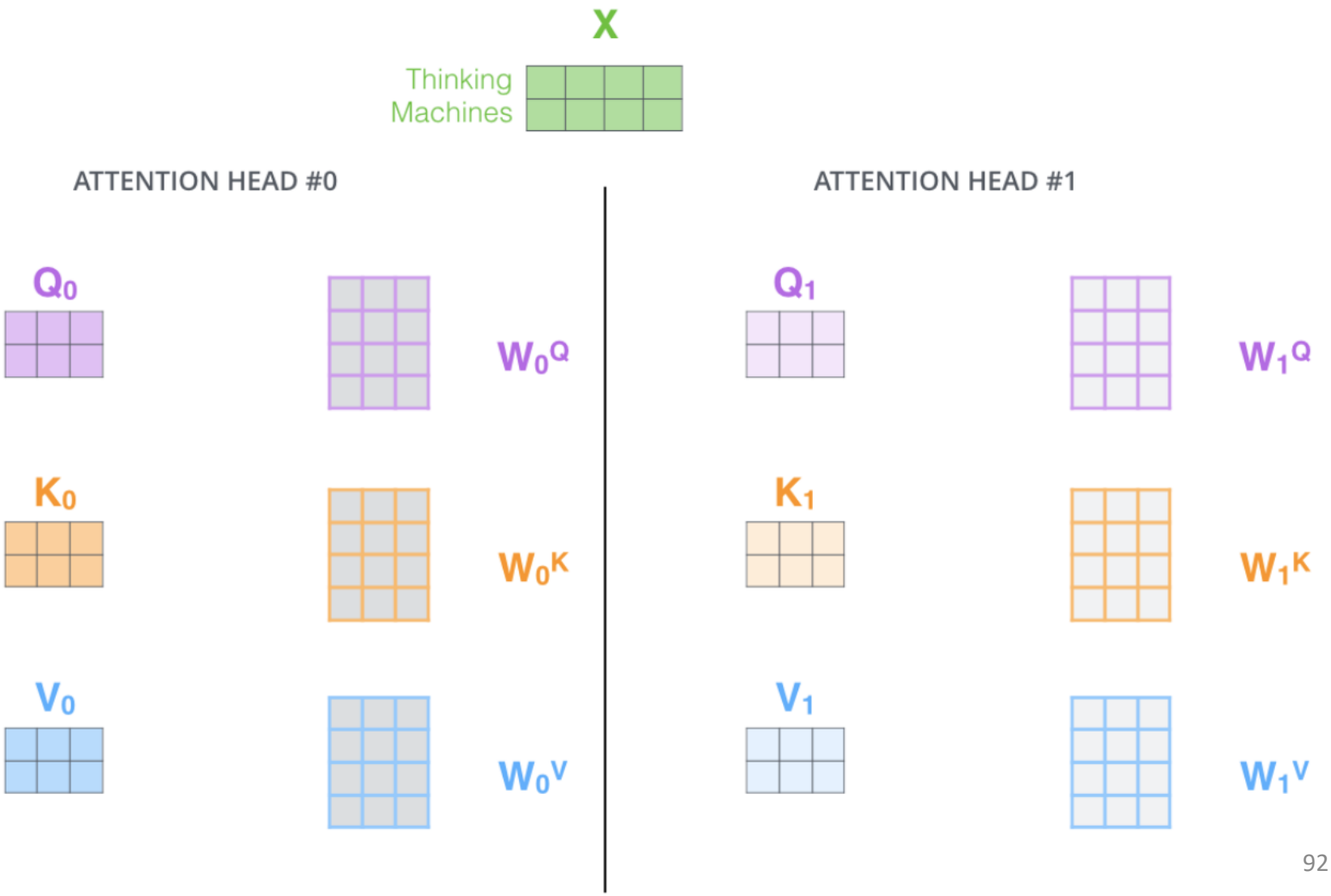
- final calculation



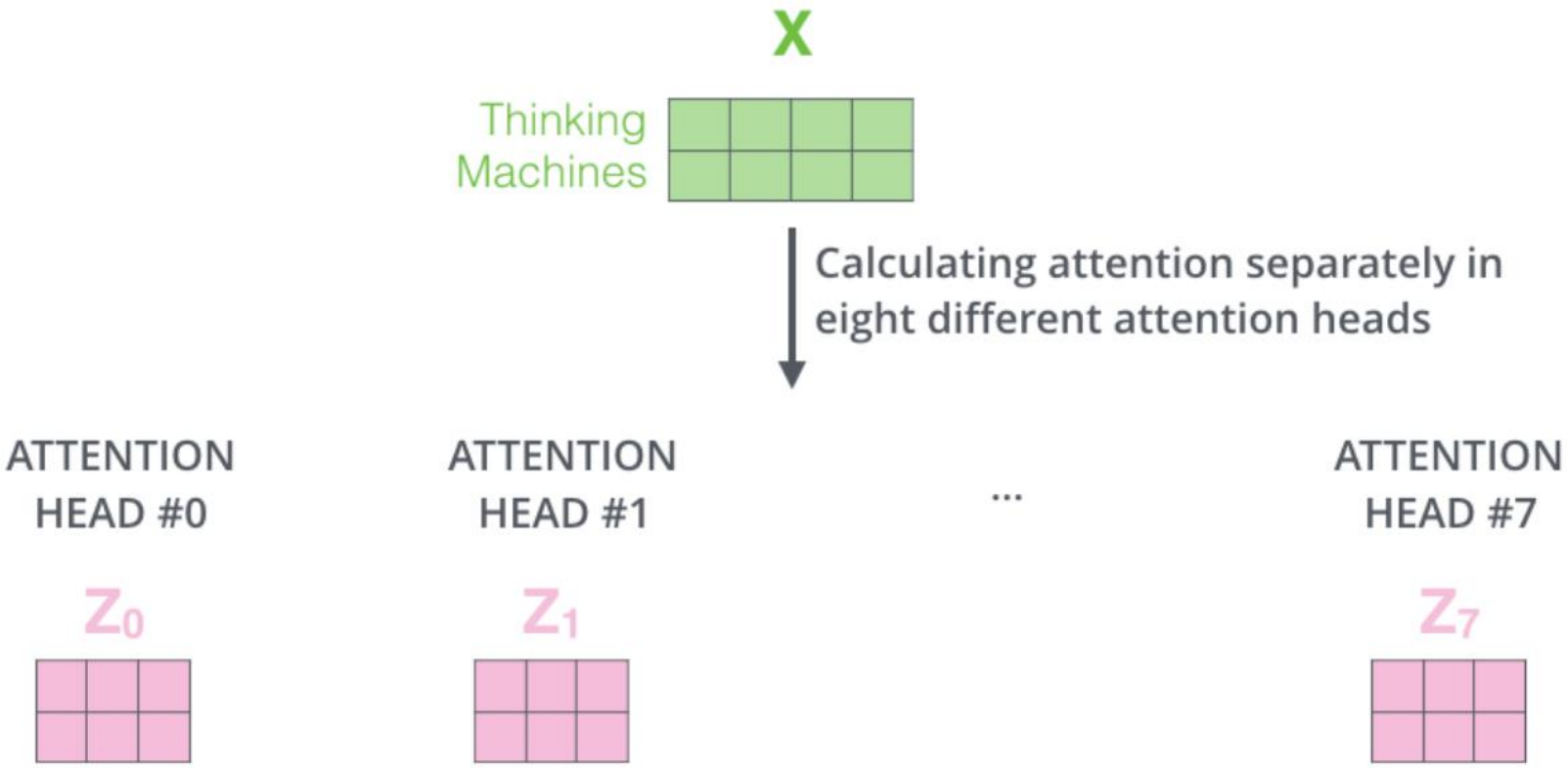
Encoding


ENCODER #2
r1
r2
ENCODER #1
Feed Forward Neural Network
Feed Forward Neural Network
z1
z2
Self-Attention
x1
x2
Thinking
Machines

Example: two attention heads



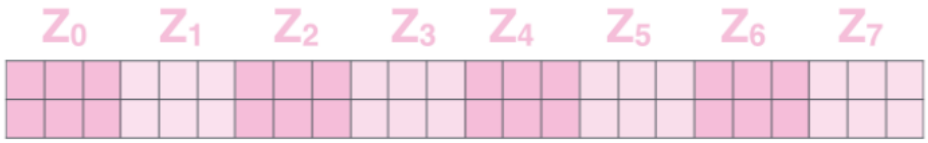
Example: 8 att. heads



- What to do with 8 Z matrices, the feed-forward layer is expecting a single matrix (one vector for each word). We need to condense all attention heads into one matrix.

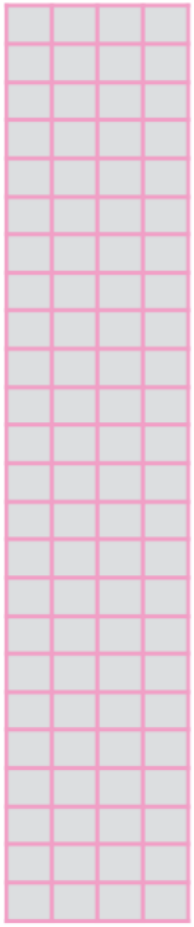
Condensation of attention heads

1) Concatenate all the attention heads



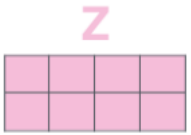
2) Multiply with a weight matrix W^O that was trained jointly with the model

X



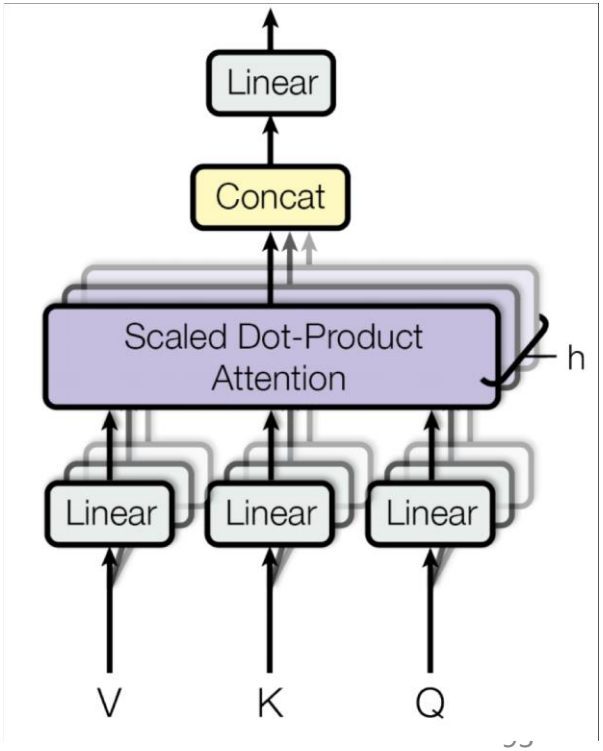
3) The result would be the Z matrix that captures information from all the attention heads. We can send this forward to the FFNN

=



Computing multi-head attention

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, ..., \text{head}_\text{h})W^O$$
$$\textbf{where } \text{head}_\text{i} = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$



Summary of self-attention

1) This is our input sentence*

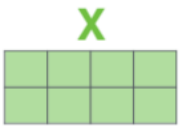
2) We embed each word*

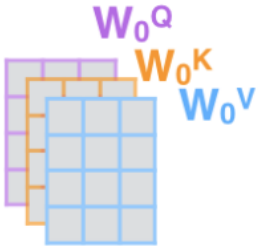
3) Split into 8 heads. We multiply X or R with weight matrices

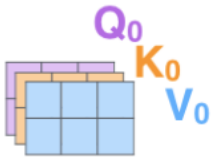
4) Calculate attention using the resulting Q/K/V matrices

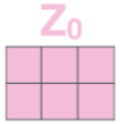
5) Concatenate the resulting Z matrices, then multiply with weight matrix W^O to produce the output of the layer

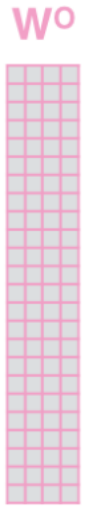
Thinking Machines

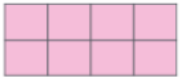


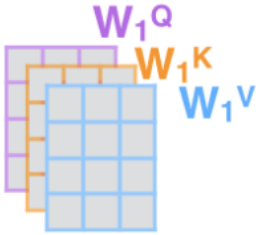


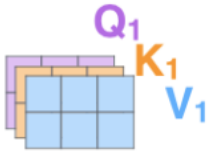


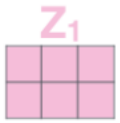






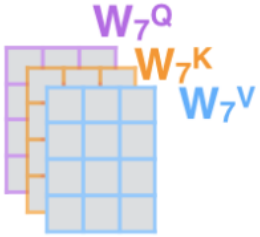


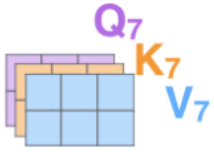




* In all encoders other than #0, we don't need embedding. We start directly with the output of the encoder right below this one







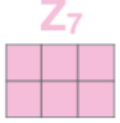


Illustration of self-attention: 1 head

- encoder #5 (the top encoder in the stack)
- As we encode the word "it", one attention head is focusing most on "the animal", while another is focusing on "tired" -- in a sense, the model's representation of the word "it" bakes in some of the representation of both "animal" and "tired".

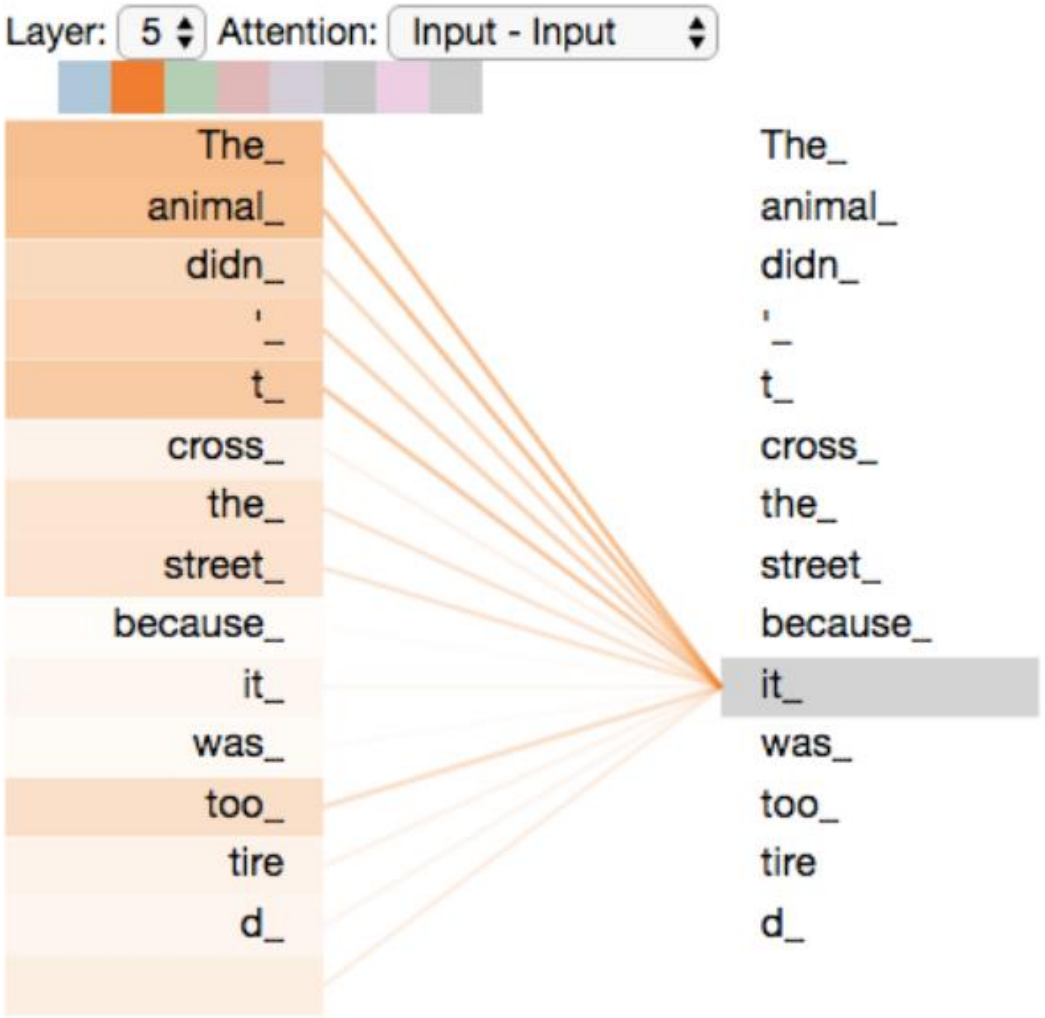
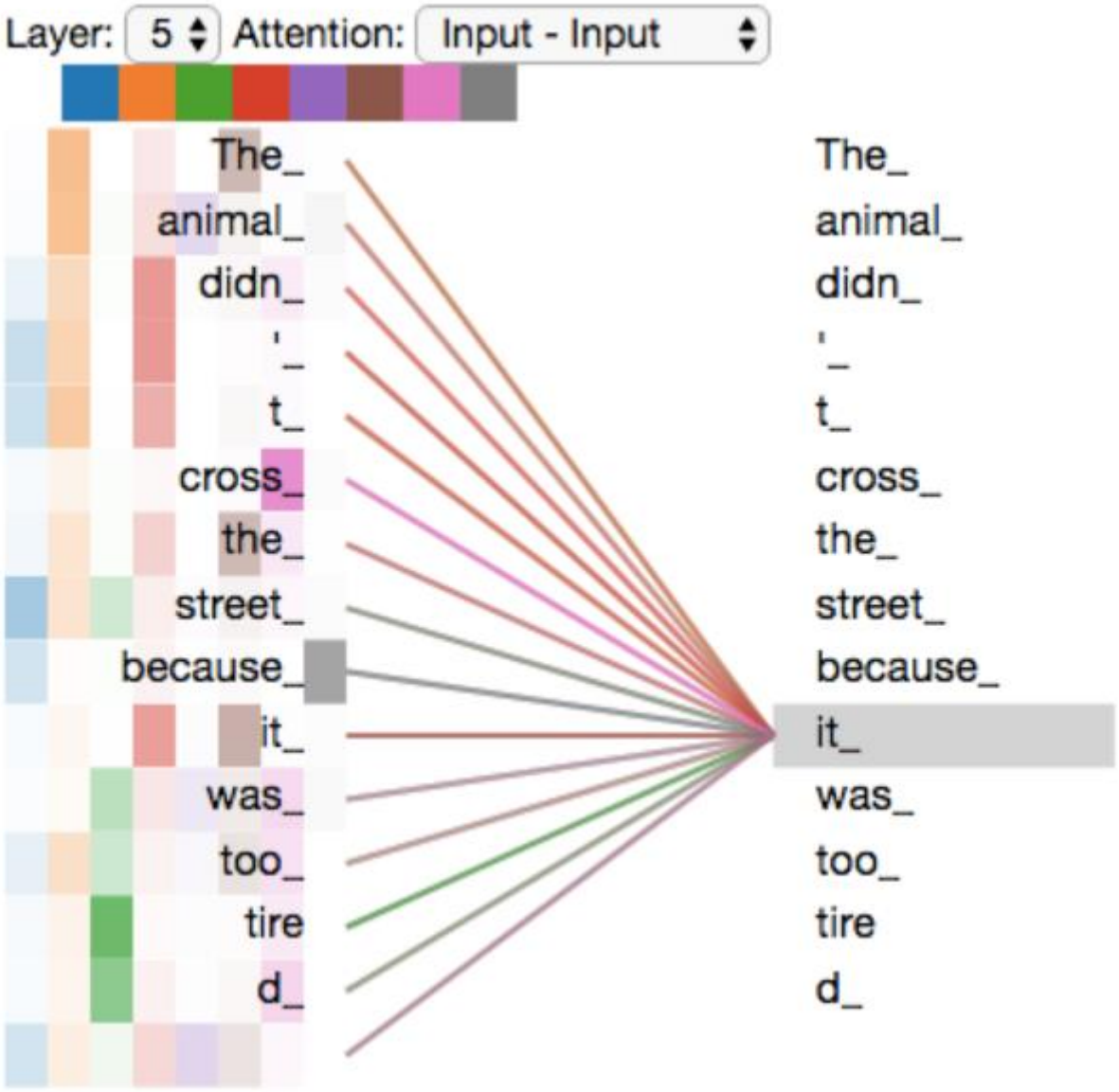
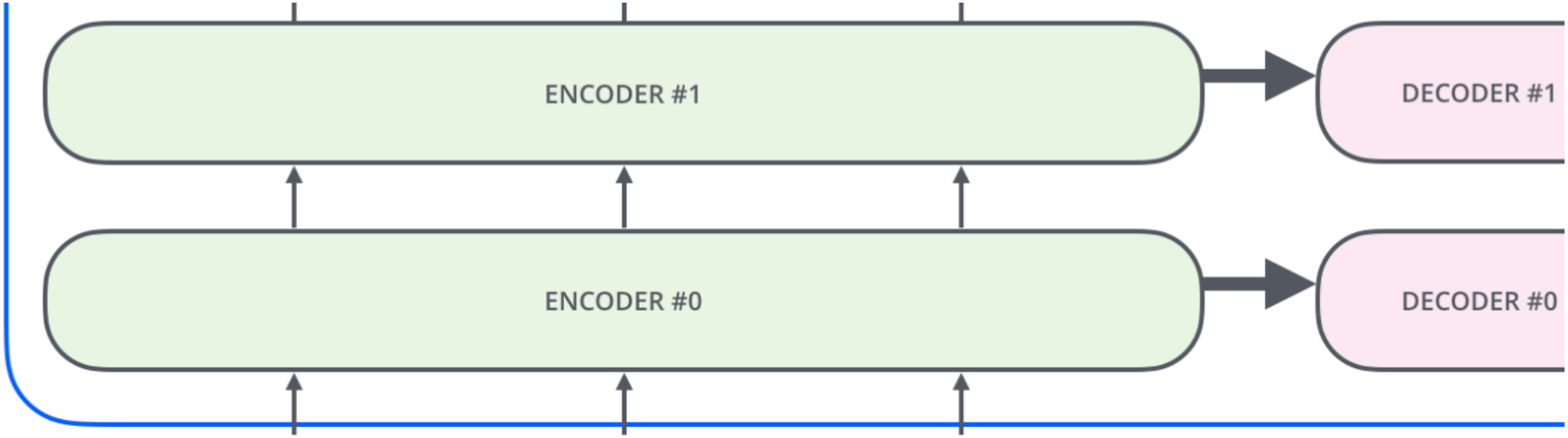


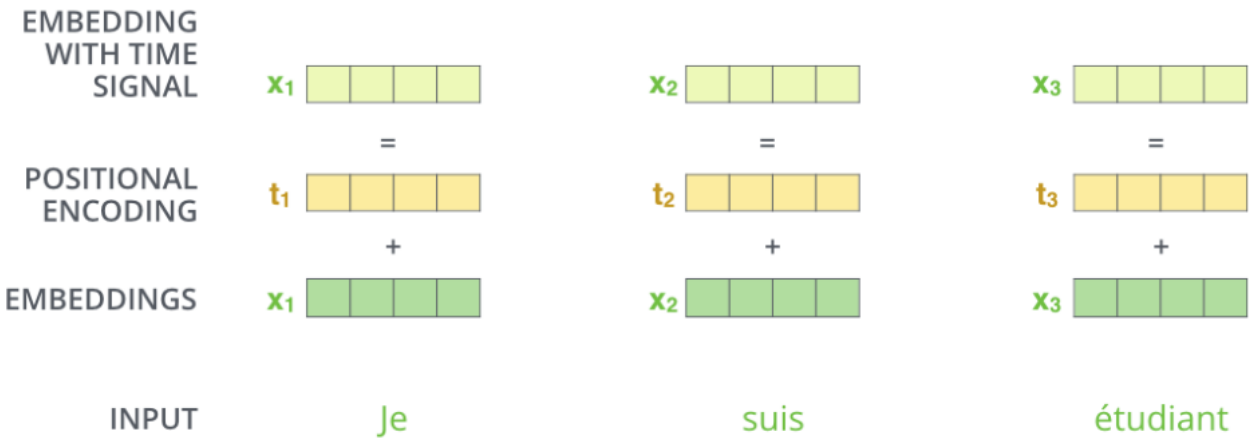
Illustration of self-attention: all heads

- all the attention heads in one picture are harder to interpret



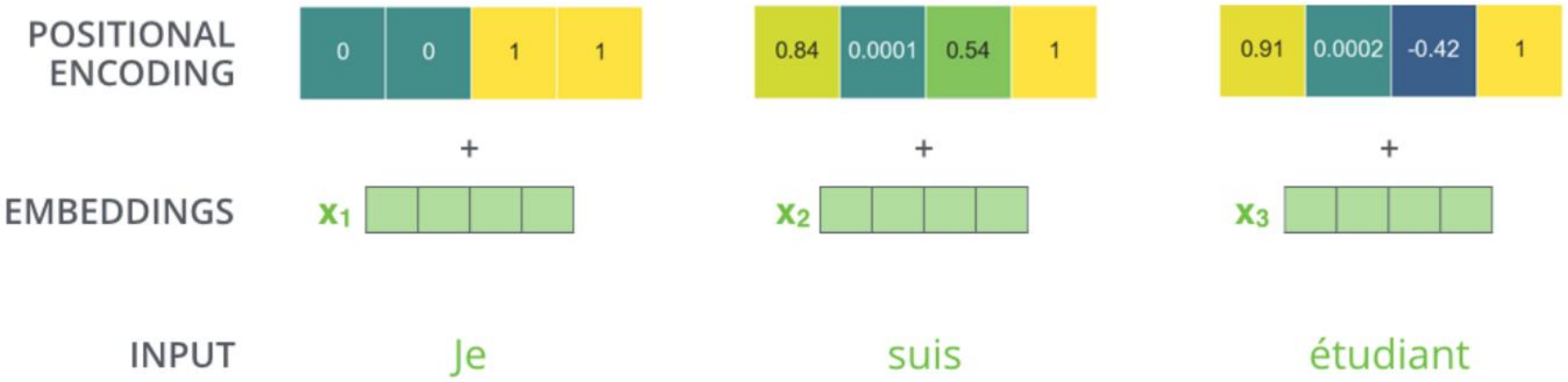
Adding position encoding



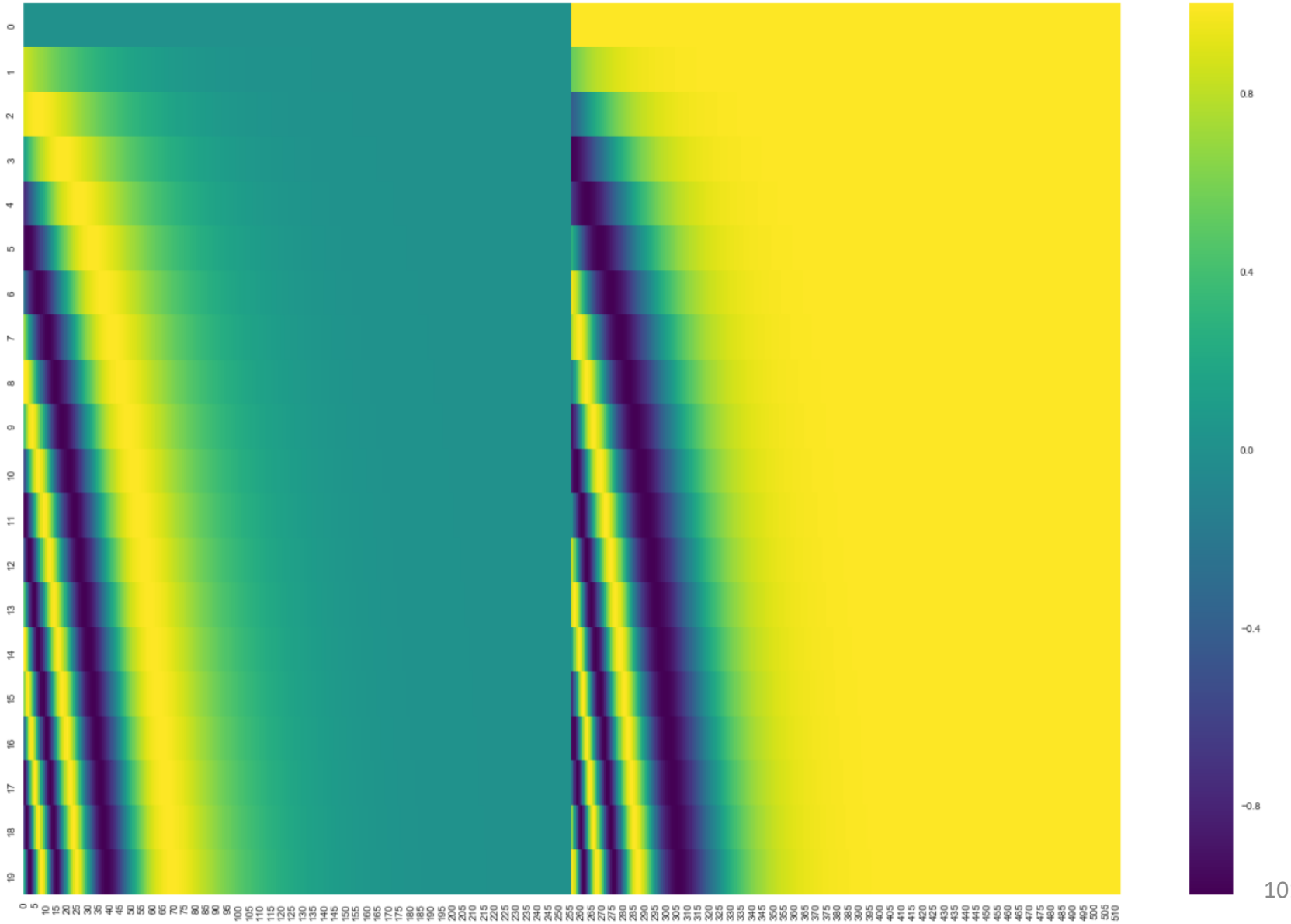


Example: encoding position

- the values of positional encoding vectors follow a specific pattern.



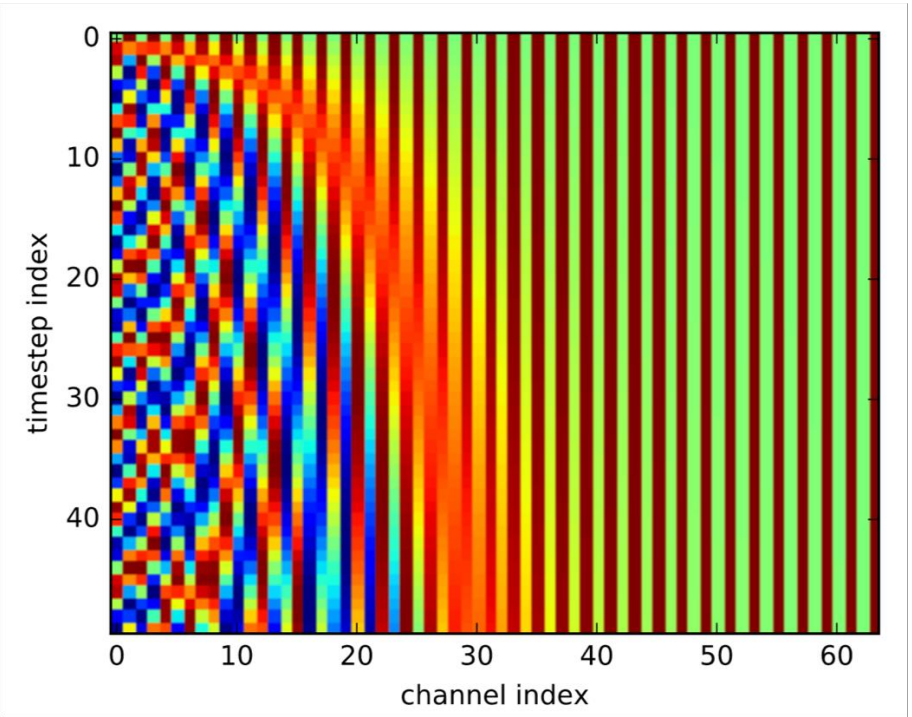
Example of positional encoding



Another positional encoding

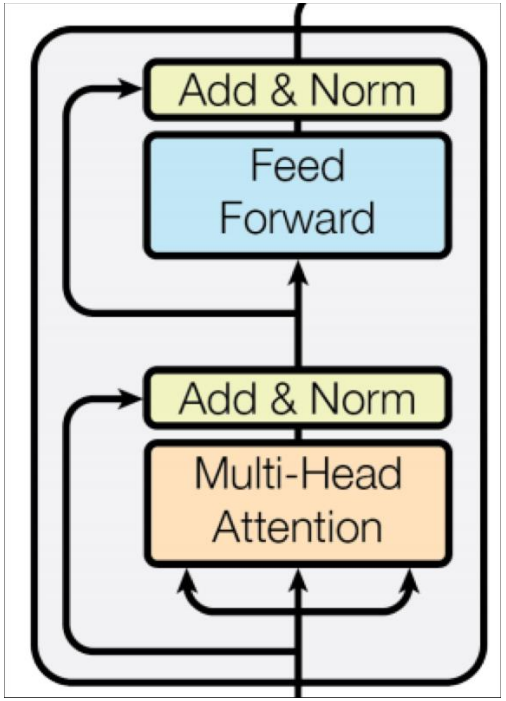
$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{\text{model}}})$$

$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{\text{model}}})$$

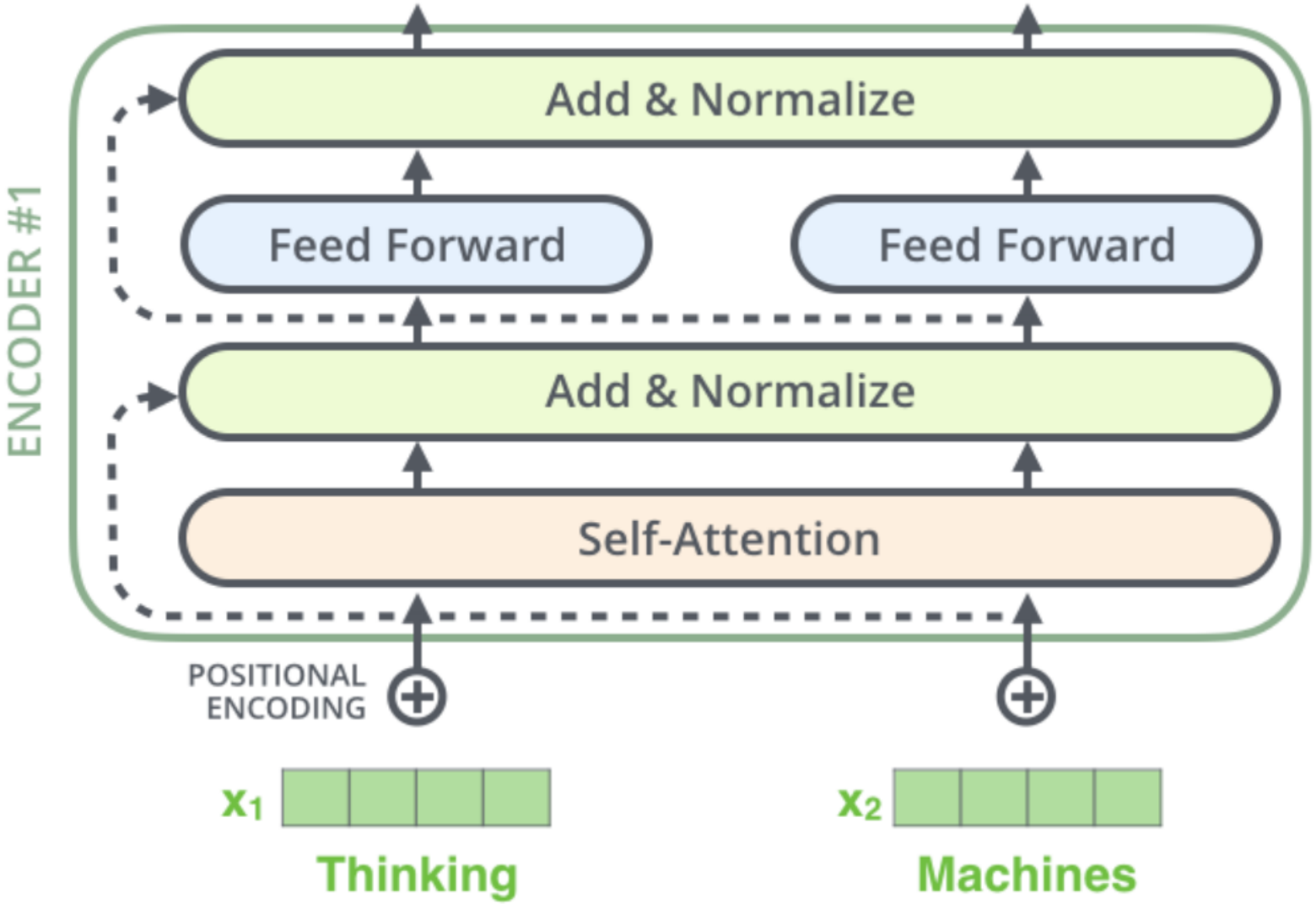


Encoder blocks

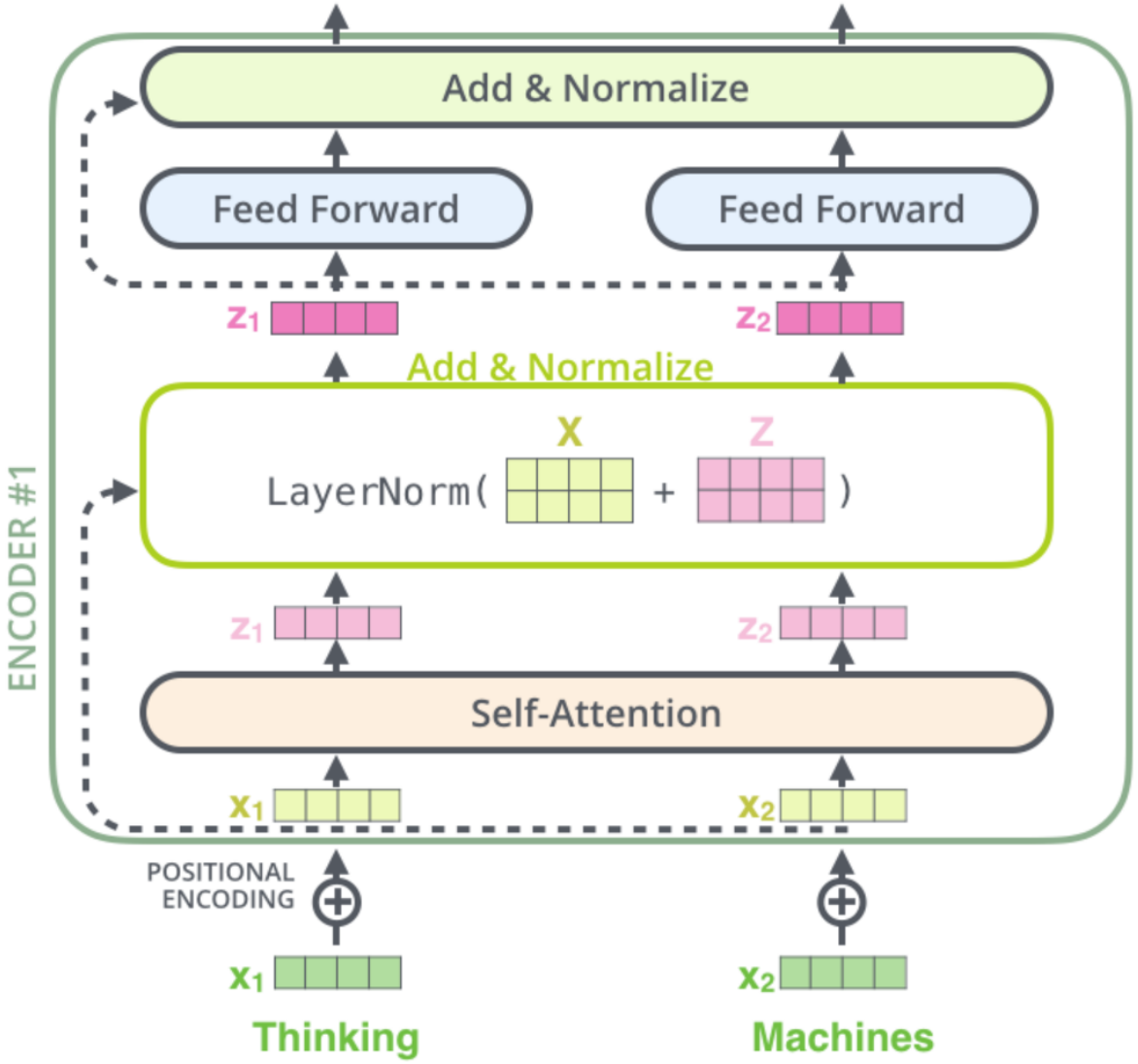
- Each block has two “sublayers”
  - Multihead attention
  - 2-layer feed-forward neural network (with ReLU)
- Each of these two steps also has a residual (short-circuit) connection and LayerNorm, i.e.:
  - LayerNorm(x + Sublayer(x))



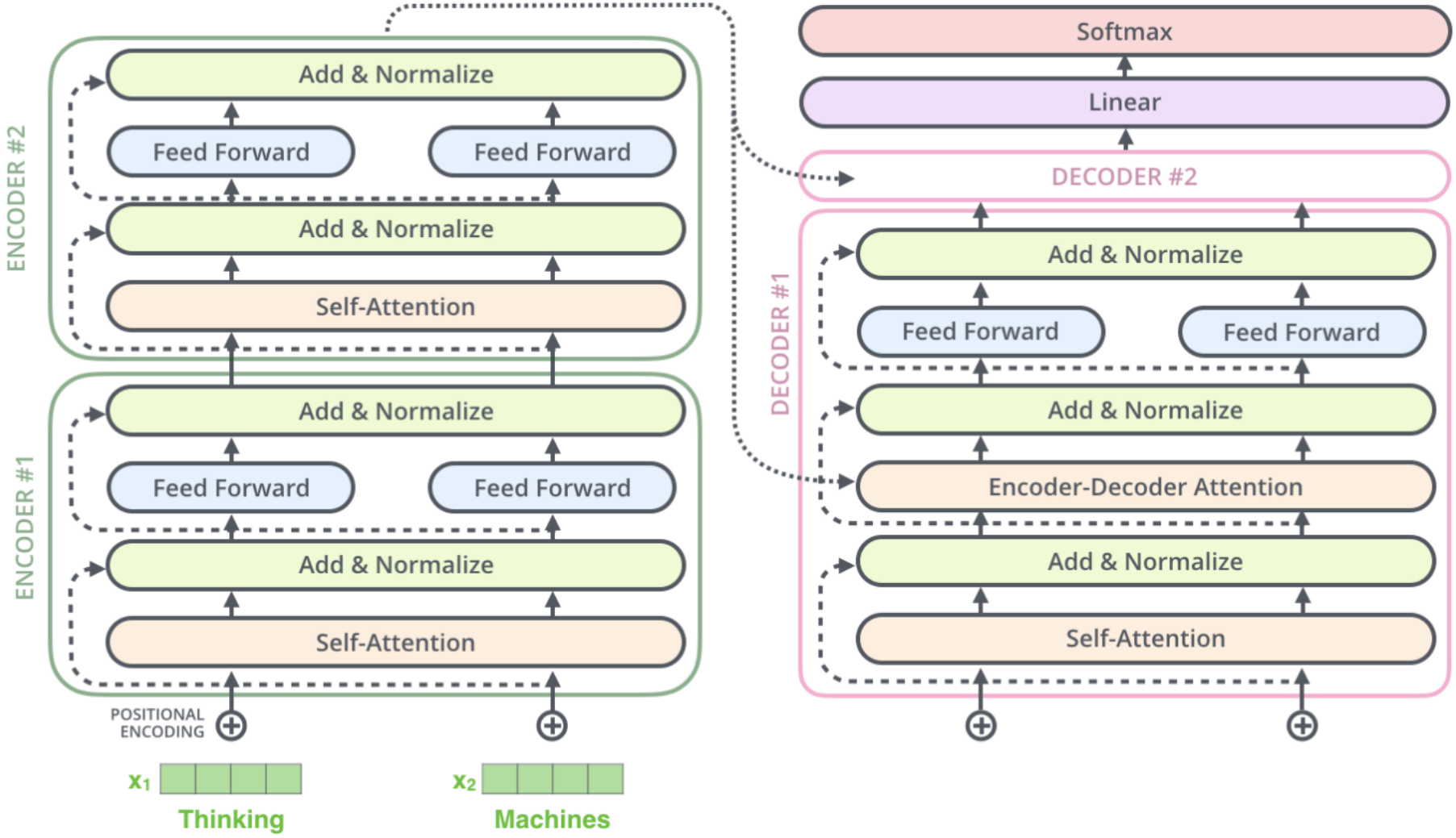
Architecture with residual connection – top level view



Architecture with residual connection – example

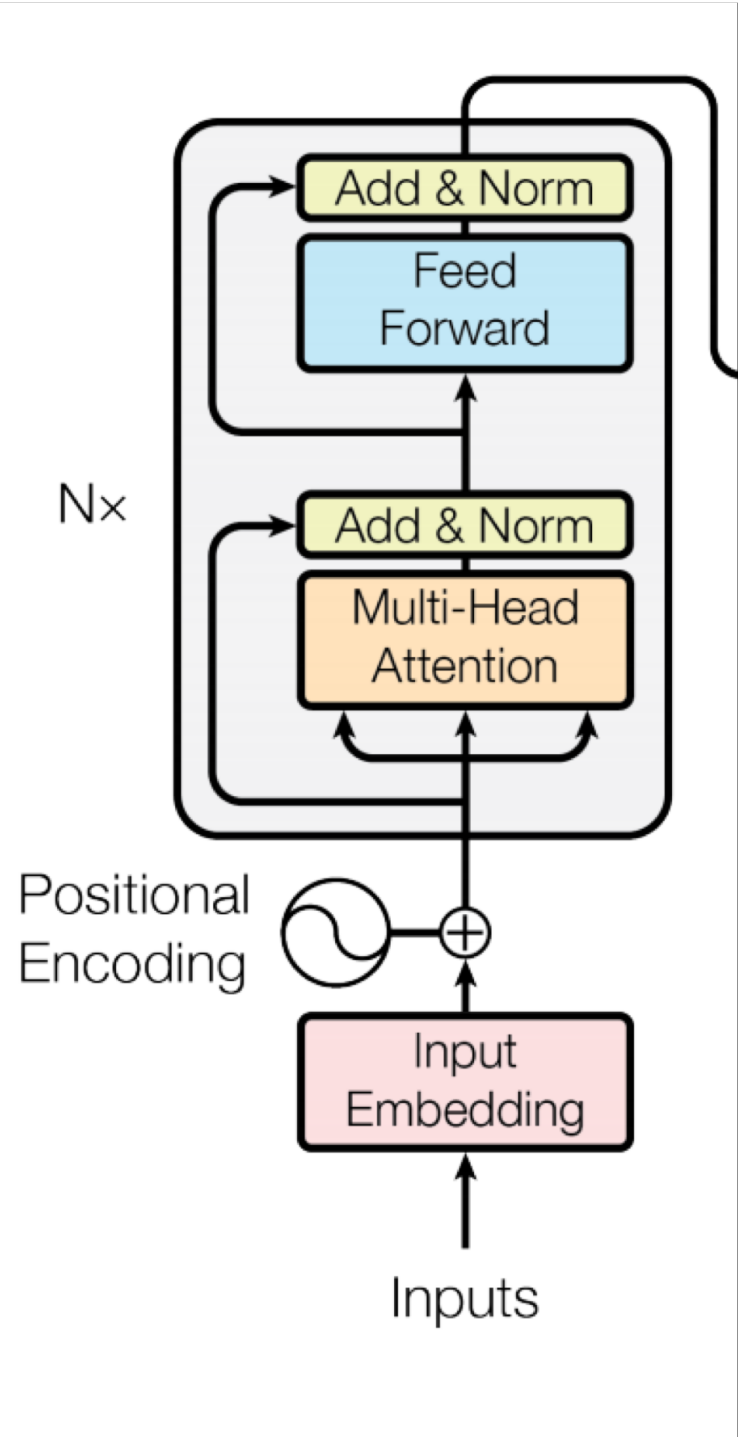


Example: 2 stacked transformer



Complete encoder

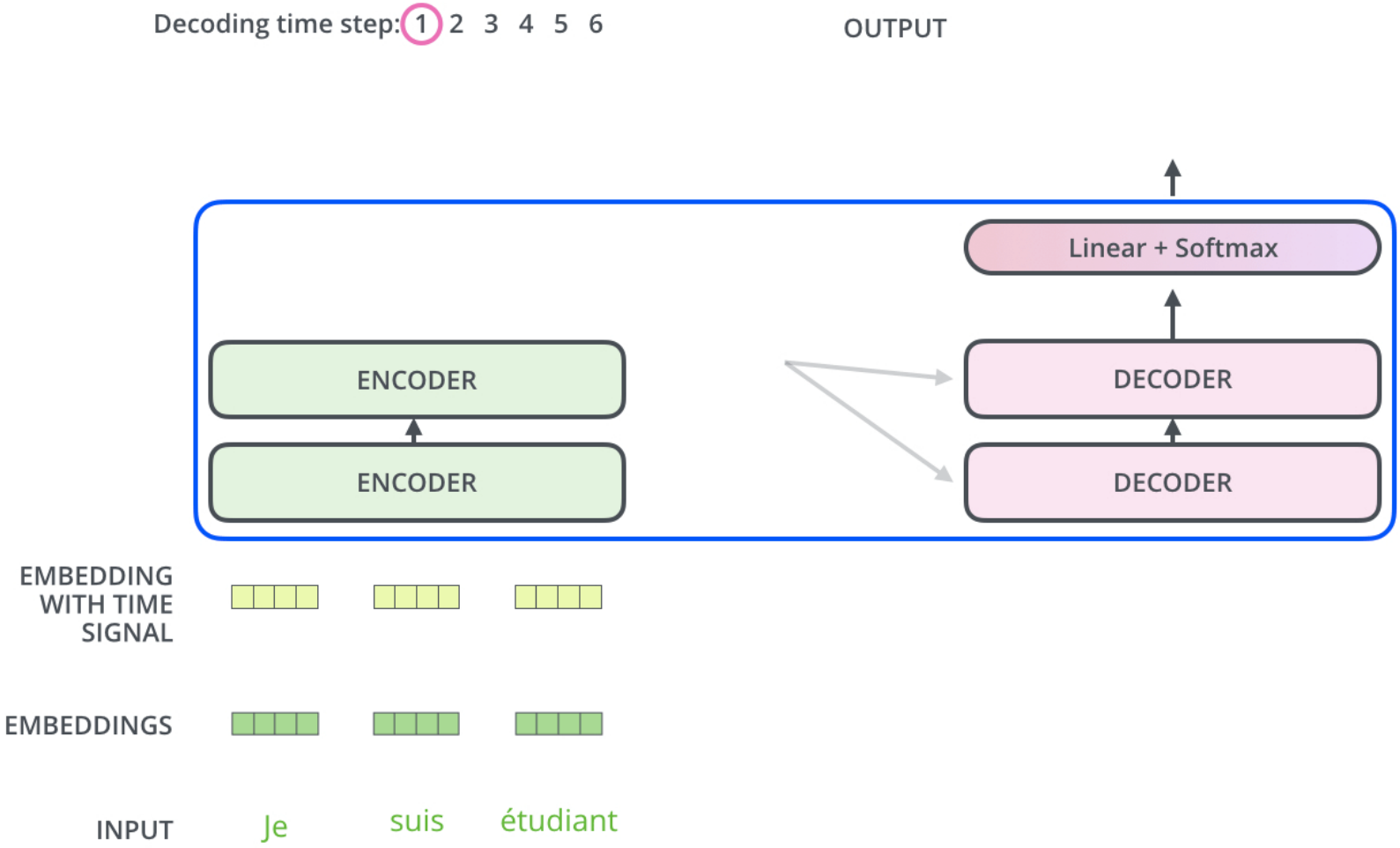
- each encoder block is repeated several times, e.g., 6 times



Decoder

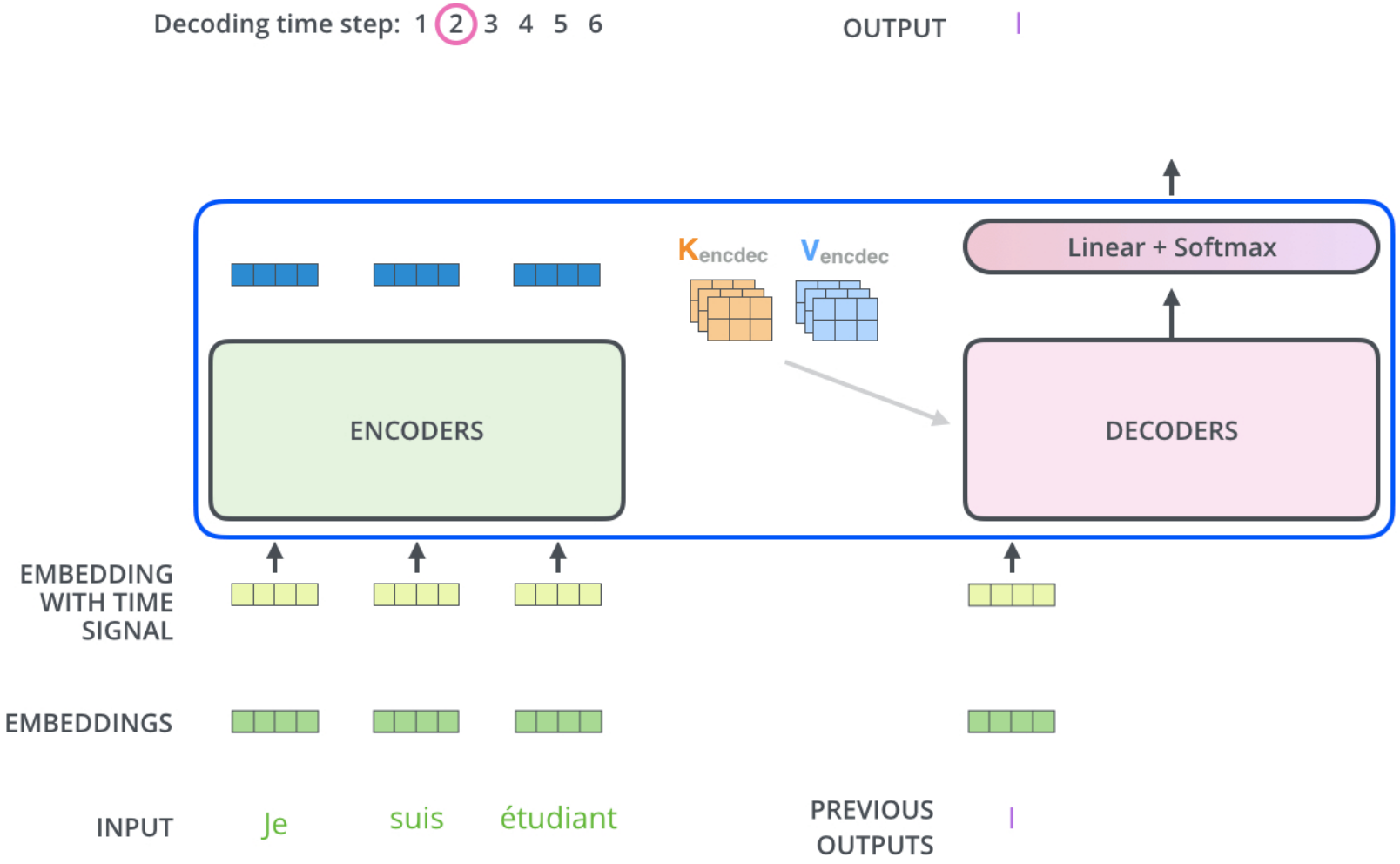
- Decoders have the same components as encoders
- An encoder starts by processing the input sequence.
- The output of the top encoder is transformed into a set of attention vectors K and V.
- These are used by each decoder in its “encoder-decoder attention” layer which helps the decoder to focus on appropriate places in the input sequence.

Encoder-decoder in action 1/2



After finishing the encoding phase, we begin the decoding phase. Each step in the decoding phase outputs an element from the output sequence (the English translation sentence in this case).

Encoder-decoder in action 2/2



The steps repeat until a special symbol indicating the end of output is generated. The output of each step is fed to the bottom decoder in the next time step. We add positional encoding to decoder inputs to indicate the position of each word.

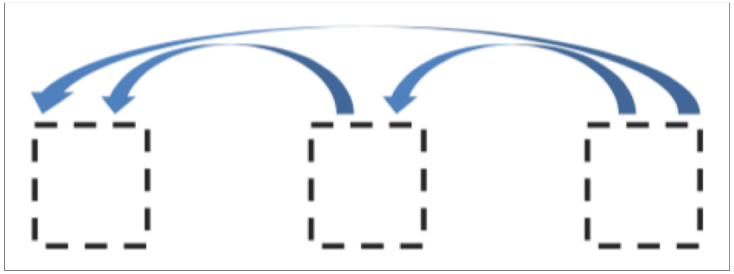
Self-attention and encoder-decoder attention in the decoder

- In the decoder, the self-attention layer is only allowed to attend to itself and earlier positions in the output sequence (to maintain the autoregressive property).
- This is done by masking future positions (setting them to -inf) before the softmax step in the self-attention calculation.

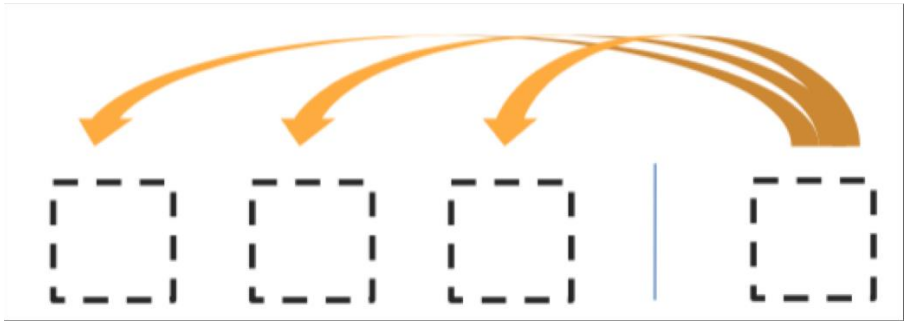
- The “Encoder-Decoder Attention” layer works just like multiheaded self-attention, except it creates its Q (queries) matrix from the layer below it, and takes the K (keys) and V (values) matrix from the output of the encoder stack.

Attentions in the decoder

1. Masked decoder self-attention on previously generated outputs

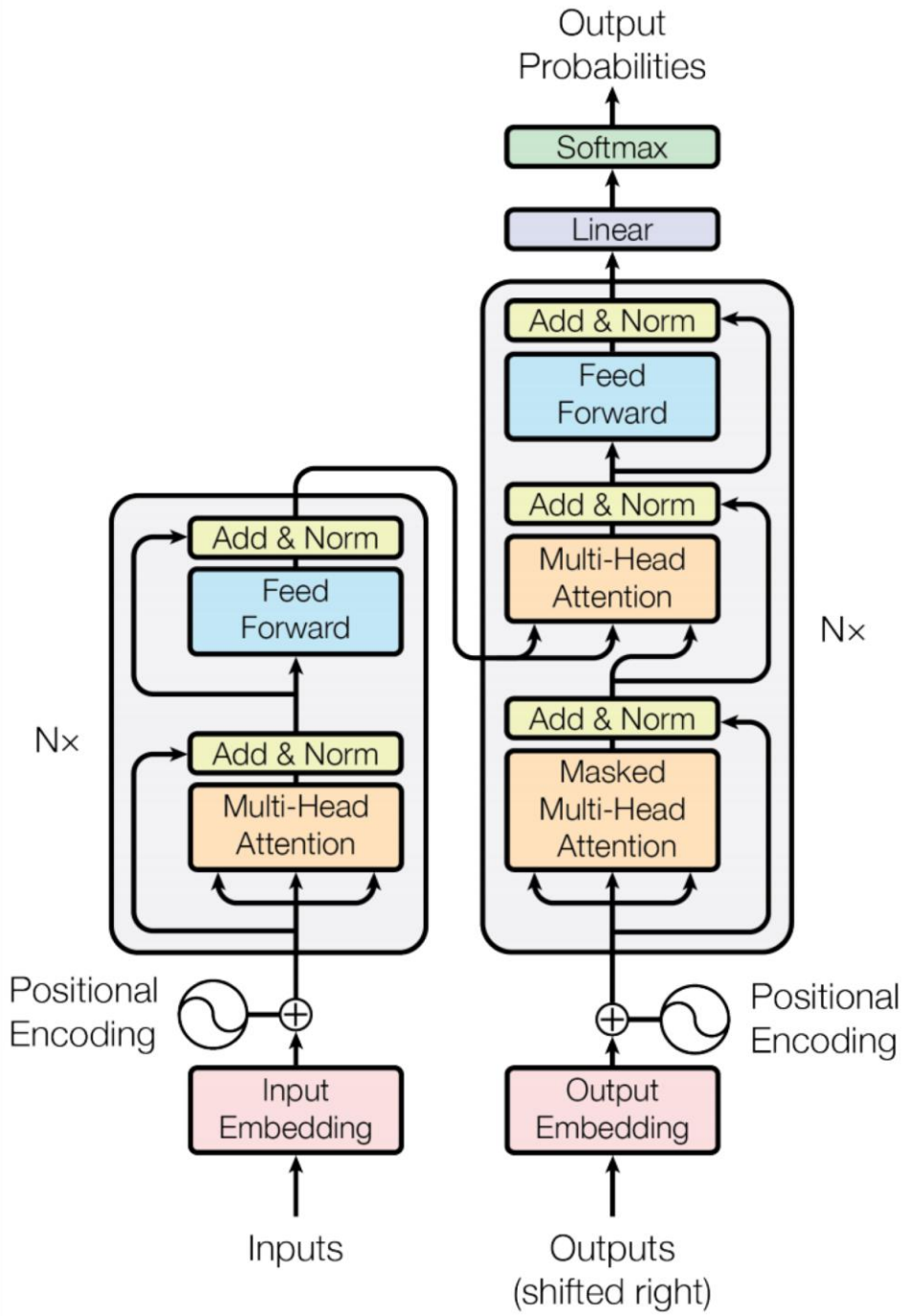


2. Encoder-Decoder Attention, where queries come from previous decoder layer and keys and values come from output of the encoder



Animated workings of attention in transformer

One encoder-decoder block



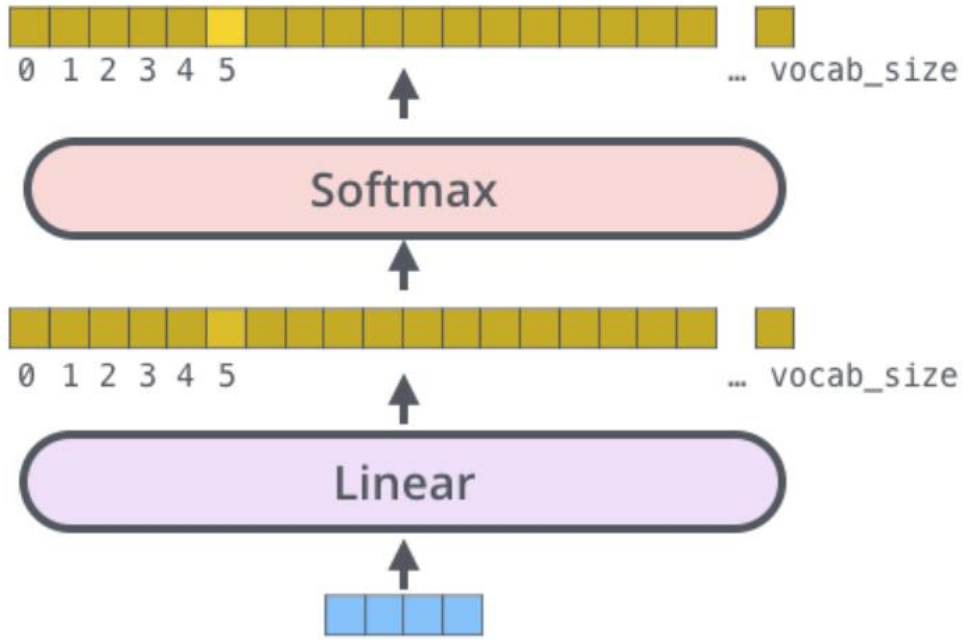
Producing the output words

Which word in our vocabulary is associated with this index?

am

Get the index of the cell with the highest value (**argmax**)

5



Training the transformer

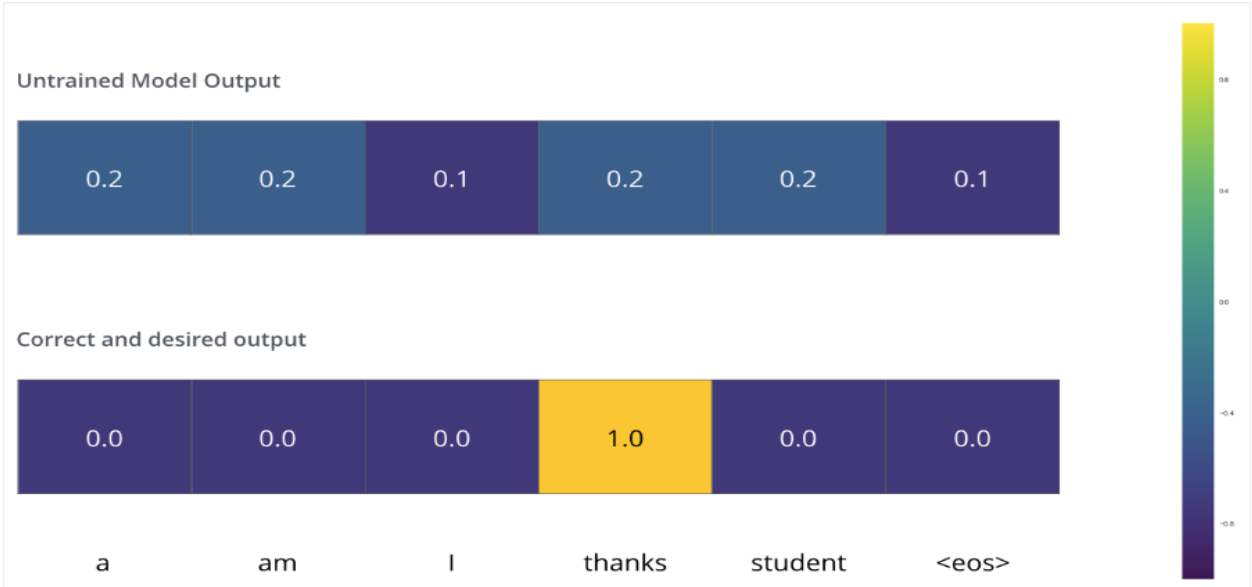
- During training, an untrained model would go through the exactly the same forward pass. But since we are training it on a labeled training dataset, we can compare its output with the actual correct output.
- For illustration, let's assume that our output vocabulary only contains six words(a, am, i, thanks, student, <eos>)
- The input is typically in the order of 10^4 (e.g., 30 000)

Output Vocabulary

WORD	a	am	I	thanks	student	<eos>
INDEX	0	1	2	3	4	5

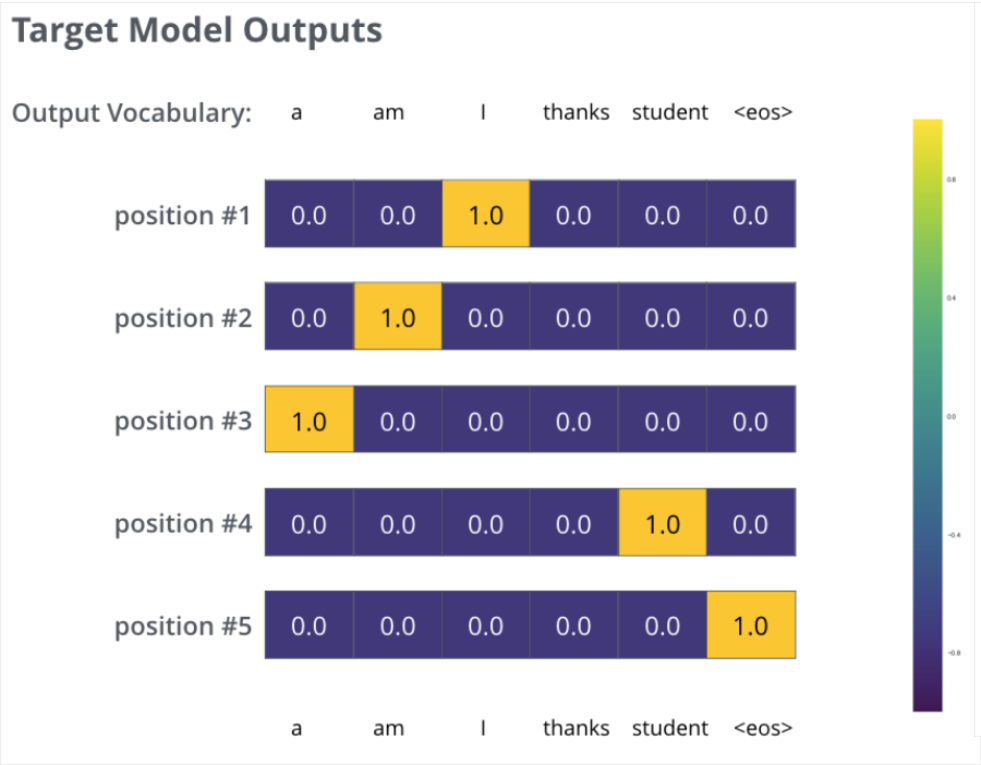
The Loss Function

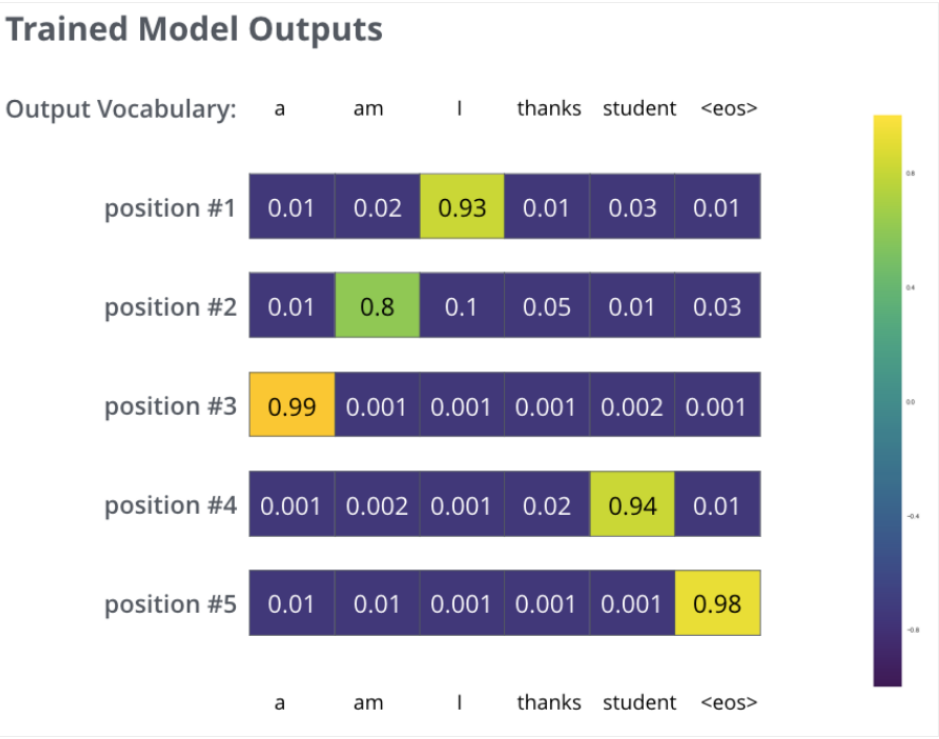
- evaluates the difference between the true output and the returned output
- transformer typically uses cross-entropy or Kullback–Leibler divergence.
- The model output is a probability distribution, the true output is 1-hot encoded, e.g.,



Loss evaluation for sequences

- loss function has to be evaluated for the whole sentence, not just a single word
- transformers use greedy decoding or beam search





Three flavours of transformers

- encoder only (BERT)
- encoder-decoder (machine translation, T5)
- decoder only (GPT)

BERT

- combines several tasks
- predicts masked words in a sentence
- also predicts order of sentences: is sentence A followed by sentence B or not
- combines several hidden layers of the network
- uses transformer neural architecture, only teh encoder part
- uses several fine tuned parameters
- multilingual variant supports 104 languages by training on Wikipedia
- publicly available

Devlin J, Chang MW, Lee K, Toutanova K (2019) BERT: Pre-training of deep bidirectional transformers for language understanding. In: Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers), pp 4171–4186

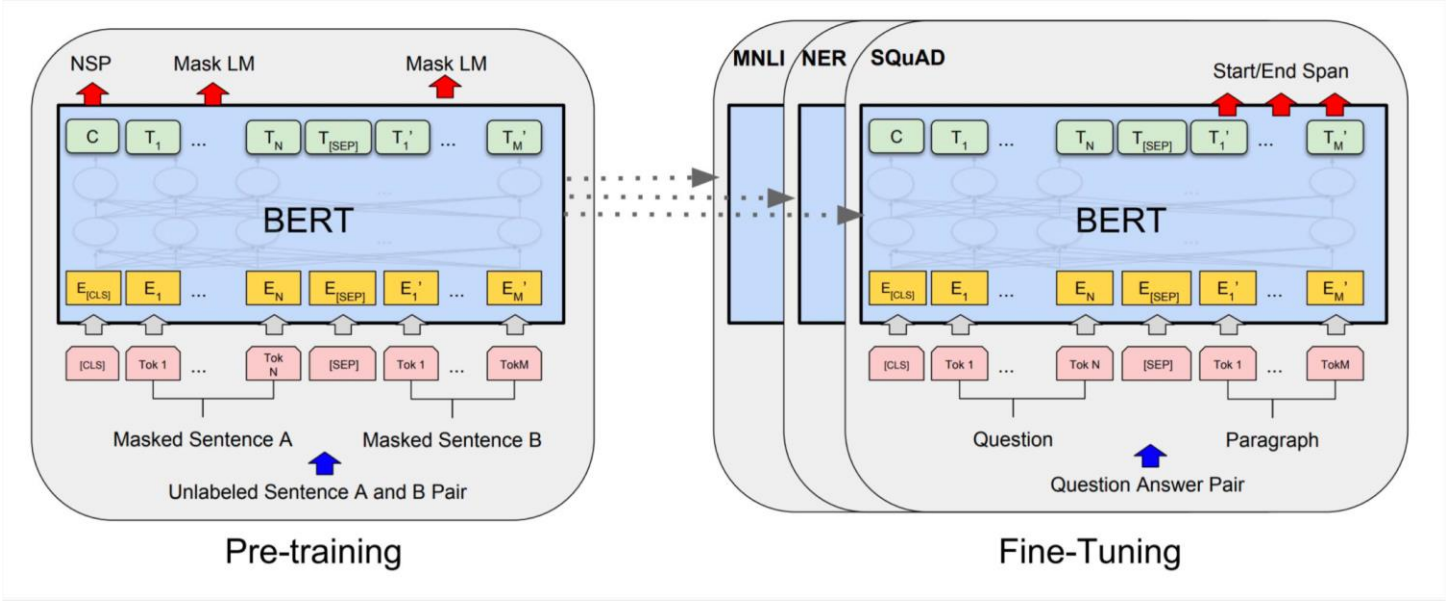
Many BERT-like embeddings

- XLM-R was trained on 2.5 TB of texts in 100 languages
- for Slovene: fastText (a variant of word2vec), ELMo, SloBERTa
- trilingual BERT – CroSloEngual
- on Clarin.si
- hundreds of papers investigating BERT-like models in major ML & NLP conferences

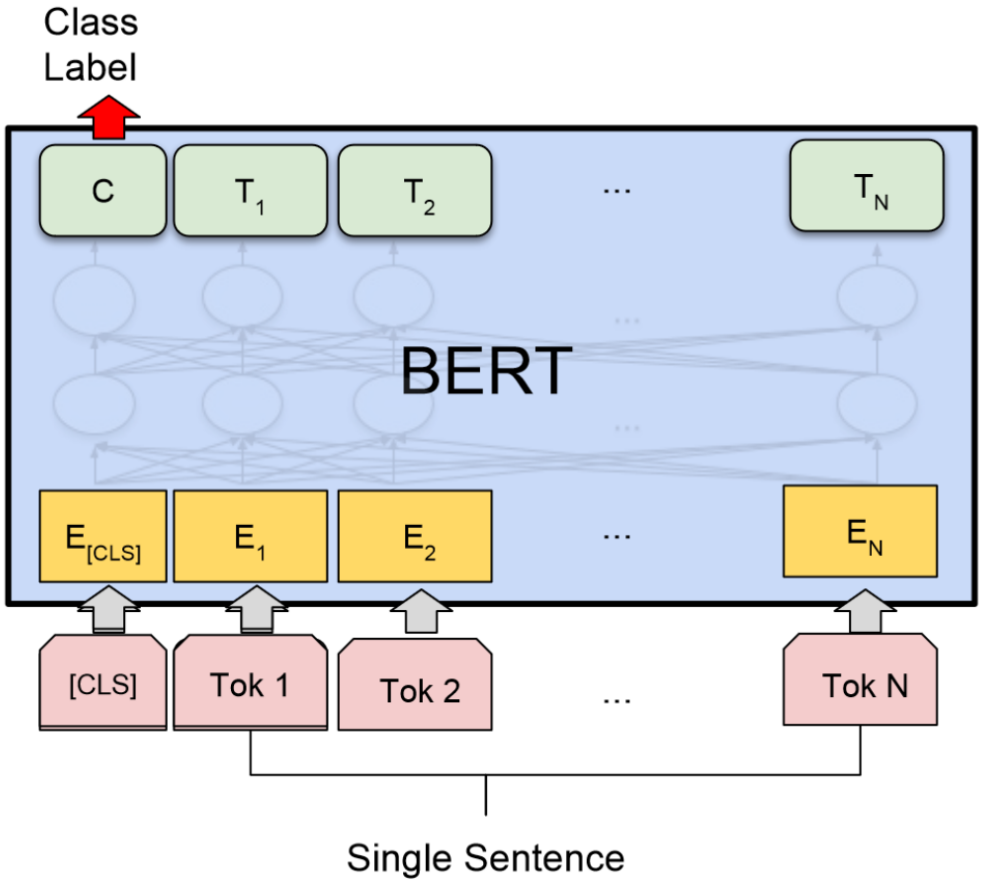
Ulčar, Matej and Marko Robnik-Šikonja. FinEst BERT and CroSloEngual BERT: less is more in multilingual models. In Proceedings of Text, Speech, and Dialogue, TSD2020, 2020.

Use of BERT

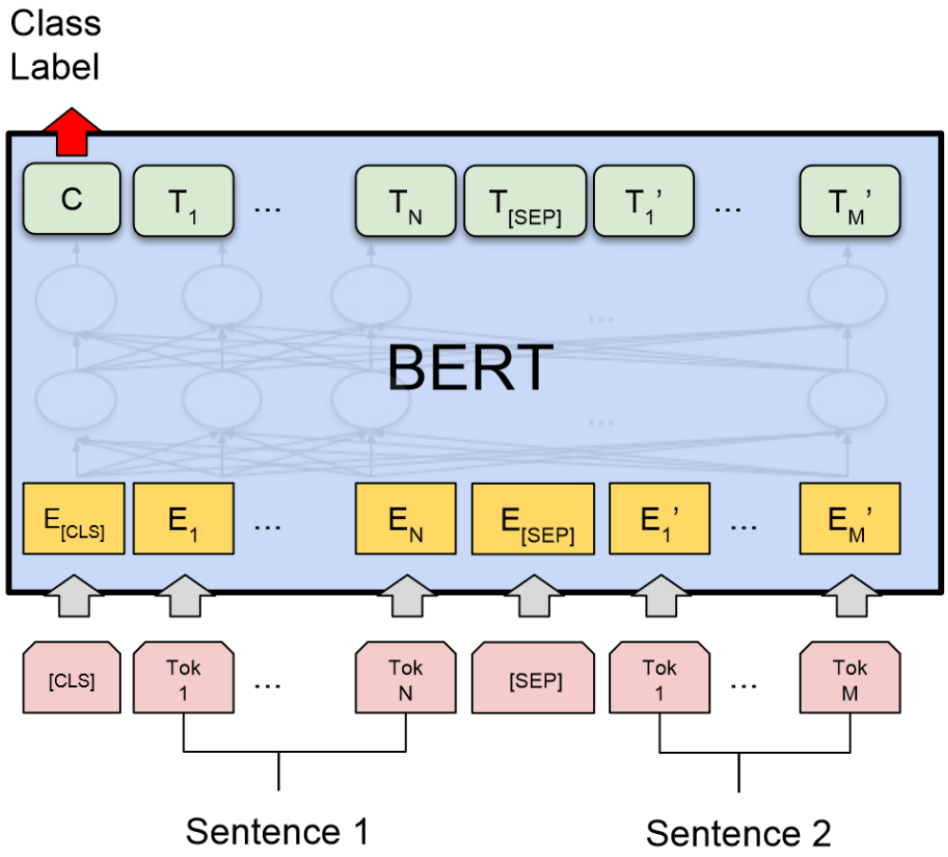
- train a classifier built on the top layer for each task that you fine tune for, e.g., Q&A, NER, inference
- achieves state-of-the-art results for many tasks
- GLUE and SuperGLUE tasks for NLI



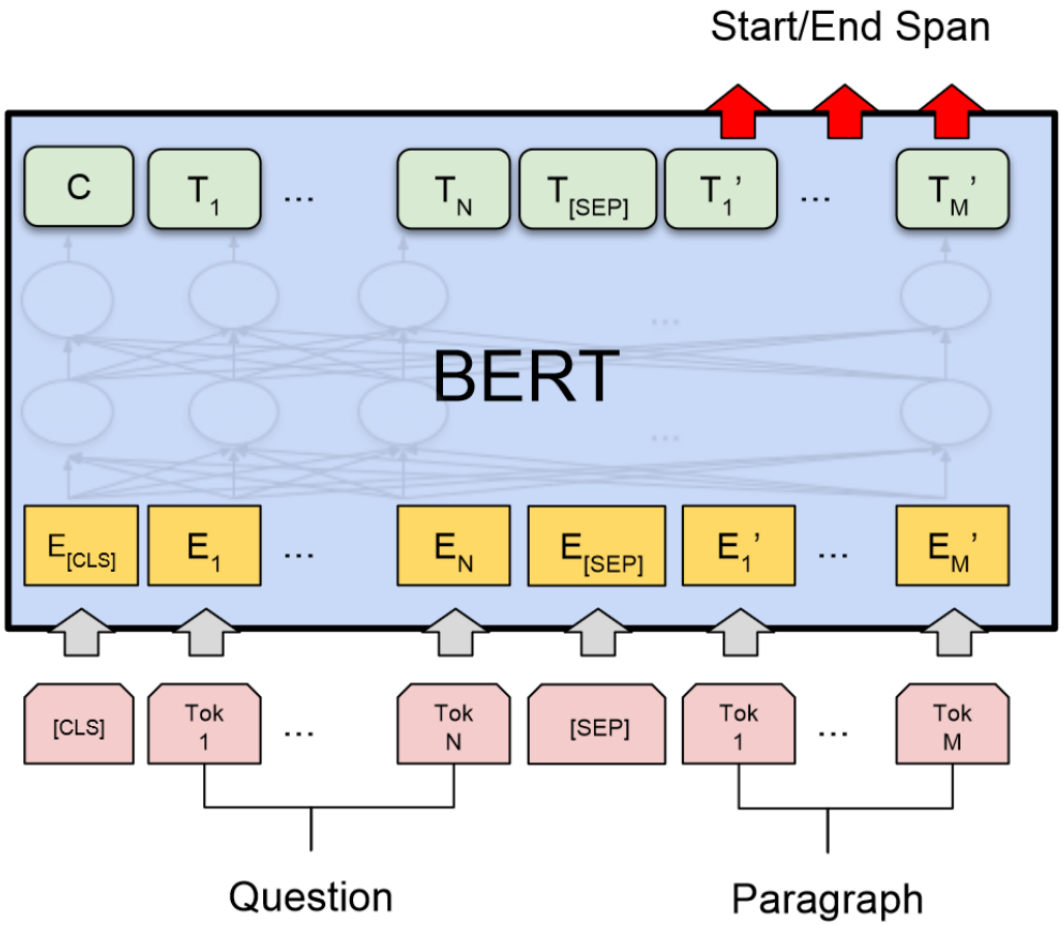
Sentence classification using BERT – sentiment, grammatical correctness



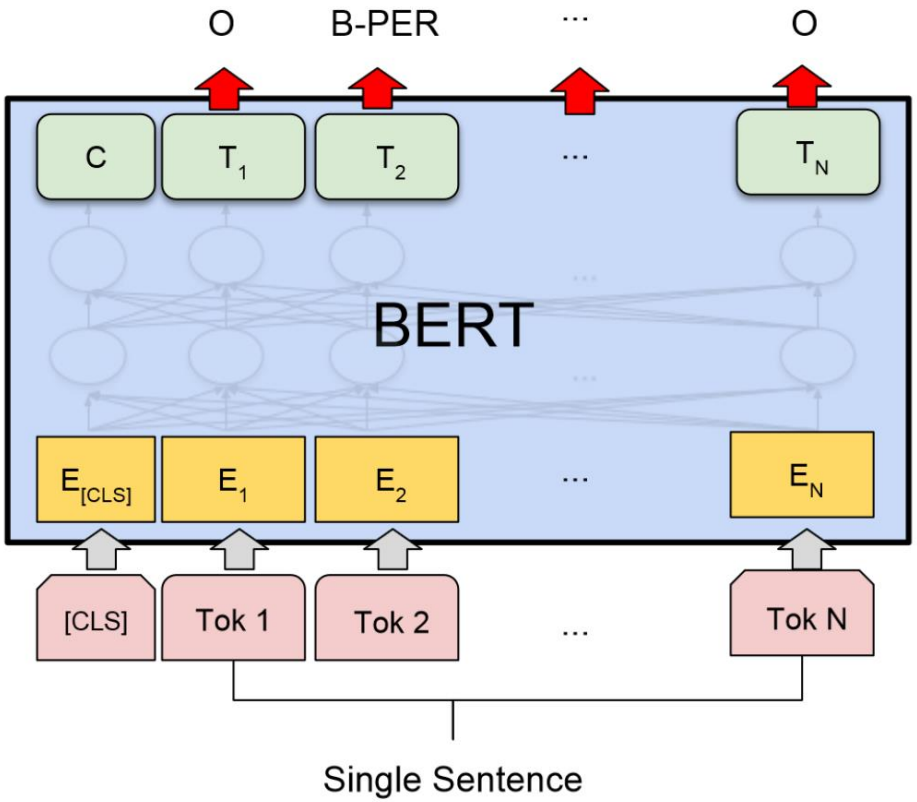
Two sentence classification using BERT-inference



Questions and answers with BERT

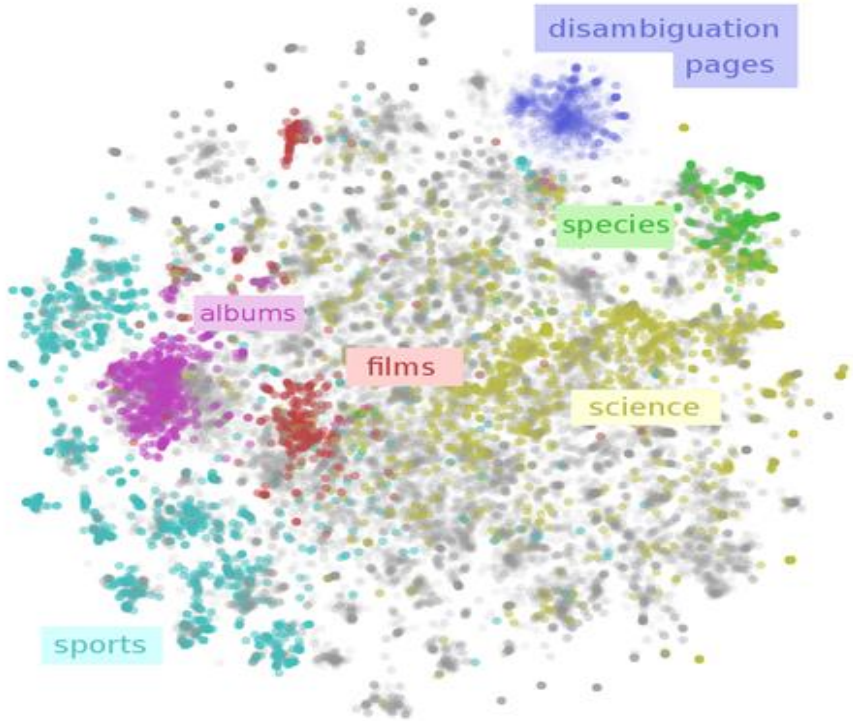


Sentence tagging with BERT-
NER, POS tagging, SRL



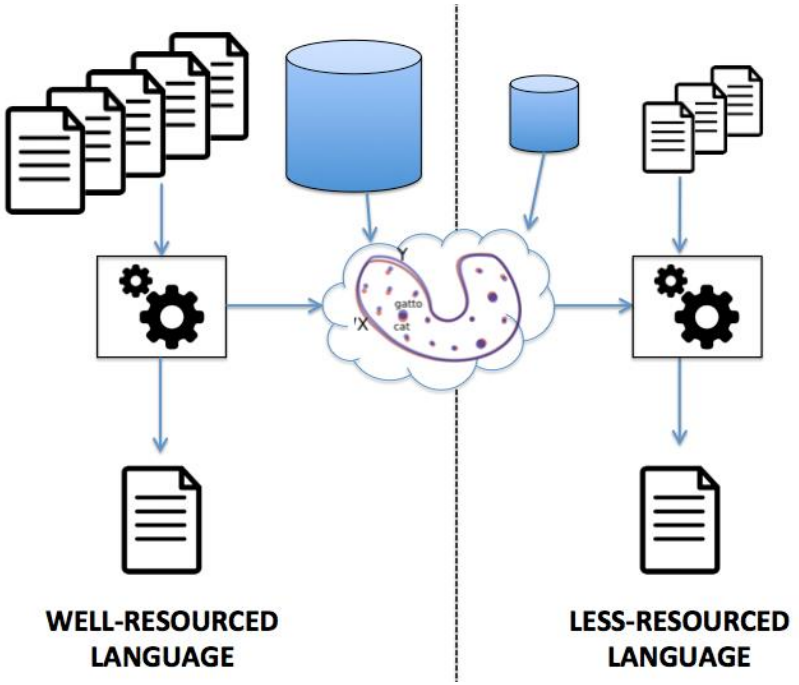
Cross-lingual embeddings

- mostly, embeddings are trained on monolingual resources
- words of one language form a cloud in high-dimensional space
- clouds for different languages can be aligned
  - $W S \approx E$ or
  - $W_1 S \approx W_2 E$



Cross-lingual model transfer based on embeddings

- Transfer of tools trained on mono-lingual resources



mostly not used anymore, superseded by multilingual LLMs

University of Ljubljana
Faculty of Computer and Information Science

Vocabulars in LLMs

- Tokenization depends on the dictionary
- The dictionary is constructed statistically (SentencePiece algorithm)

- Sentence: “Letenje je bilo predmet precej starodavnih zgodb.”
- SloBERTa:

'▁Le', 'tenje', '▁je', '▁bilo', '▁predmet', '▁precej', '▁staroda', 'vnih', '▁zgodb', '.’

- mBERT:

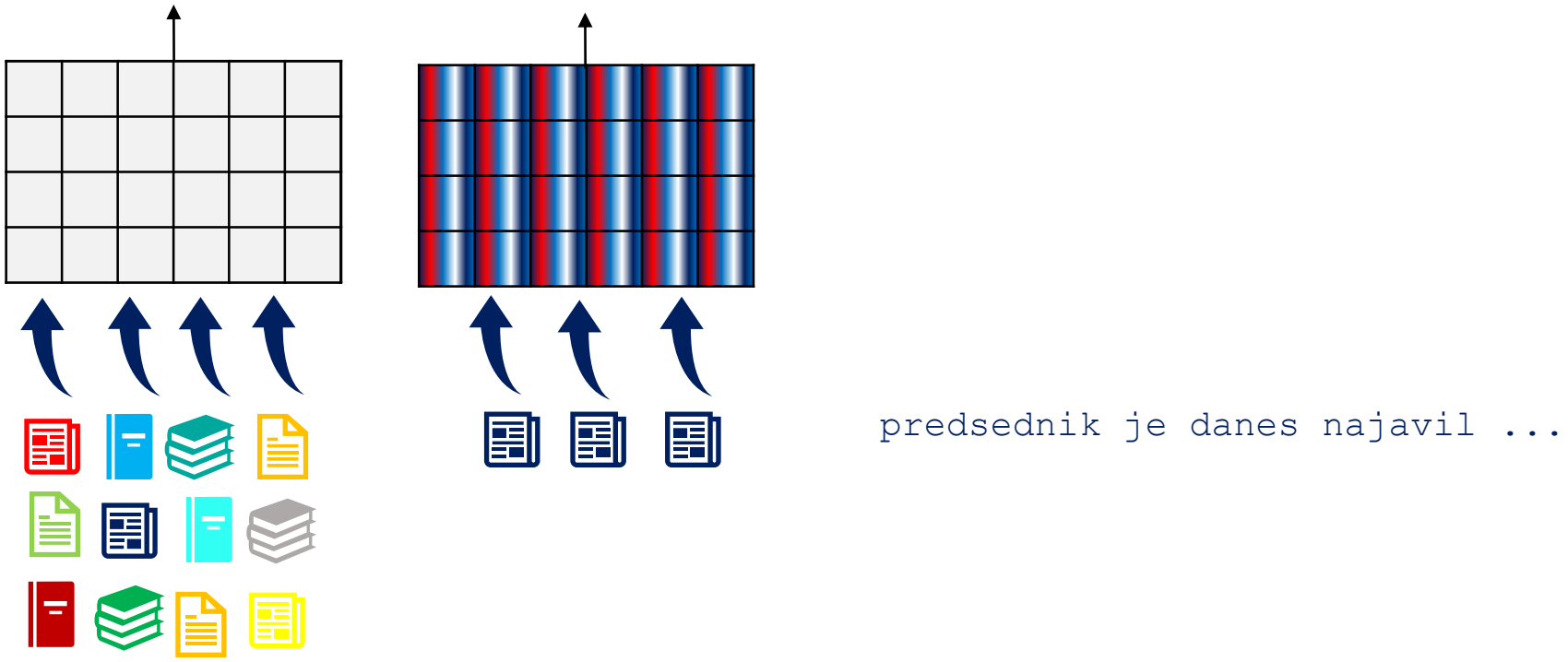
'Let', '##en', '##je', 'je', 'bilo', 'pred', '##met', 'pre', '##cej', 'star', '##oda', '##vnih', 'z', '##go', '##d', '##b', '.'

Ulčar, M., & Robnik-Šikonja, M. (2021) Training dataset and dictionary sizes matter in BERT models: The case of Baltic languages. In *International Conference on Analysis of Images, Social Networks and Texts* (pp. 162-172)

Multilingual LLMs

- Pretrained on multiple languages simultaneously
- multilingual BERT supports 104 languages by training on Wikipedia
- XLM-R was trained on 2.5 TB of texts
- allow cross-lingual transfer
- often solve the problem of insufficient training resources for less-resourced languages

Using multilingual models



Zero-shot transfer and few-shot transfer

What LLMs learn?

- We would like to travel to [MASK], ki je najlepši otok v Mediteranu.

SloBERTa: ..., Slovenija, I, Koper, Slovenia

CSE-BERT: Hvar, Rab, Cres, Malta, Brač

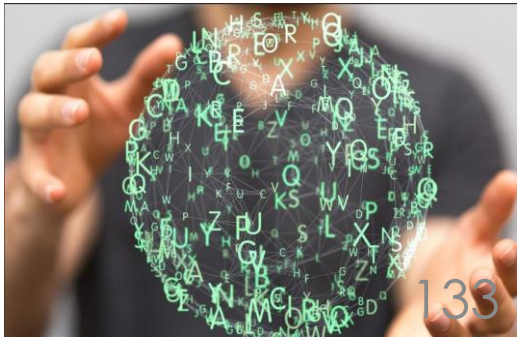
XLM-R: Mallorca, Tenerife, otok, Ibiza, Zadar

mBERT: Ibiza, Gibraltar, Tenerife, Mediterranean, Madeira

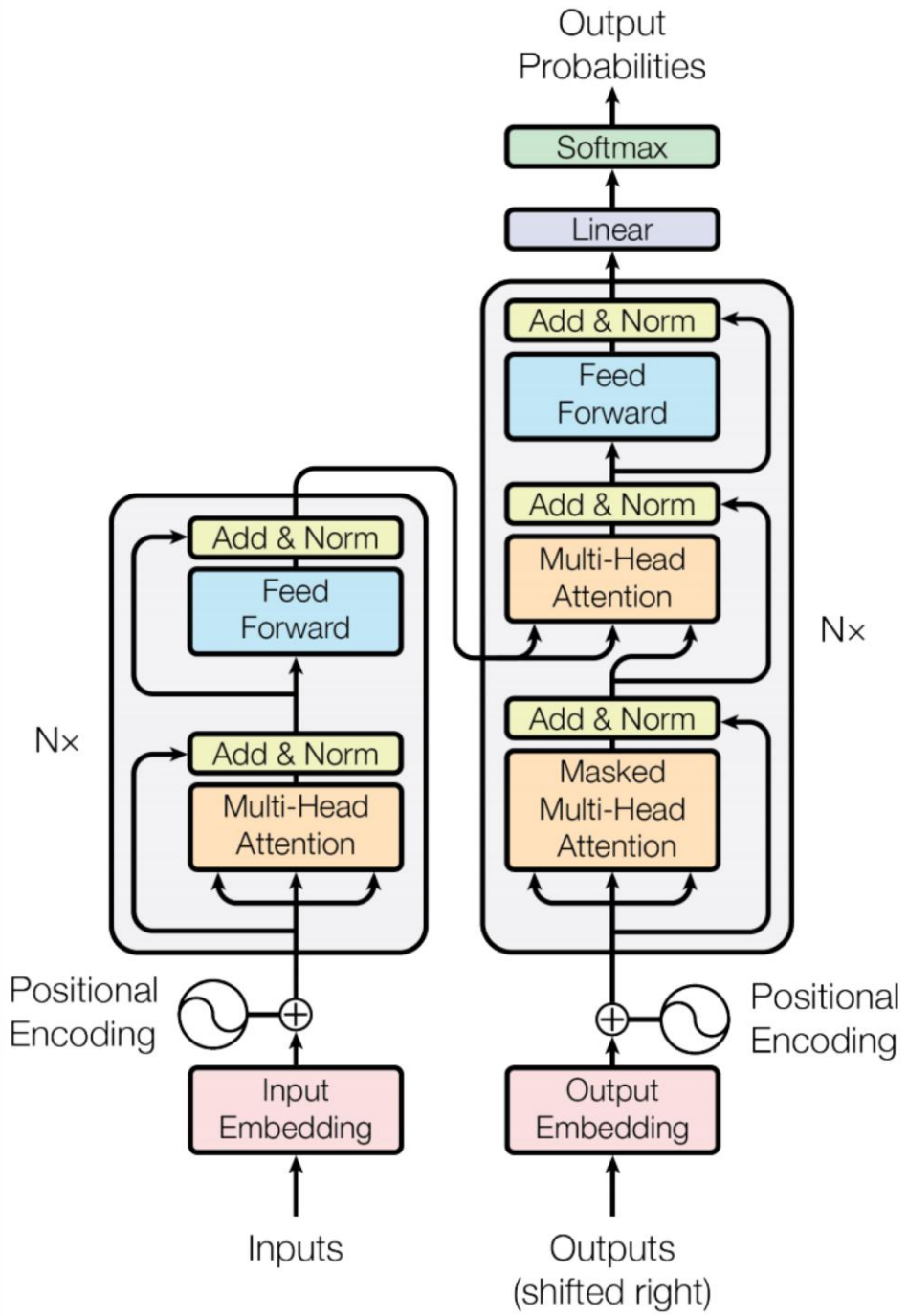
BERT (en): Belgrade, Italy, Serbia, Prague, Sarajevo

Embed all the things!

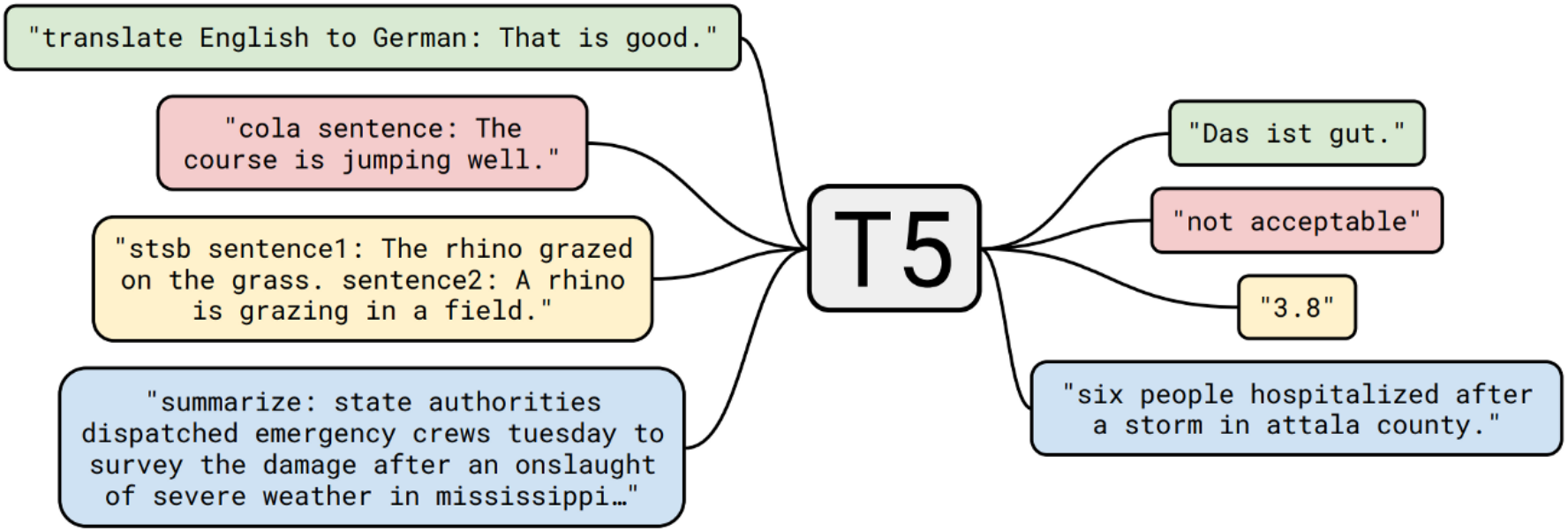
- Neural networks require numeric input
- Embedding shall preserve relations from the original space
- Representation learning is a crucial topic in nowadays machine learning
- Lots of applications whenever enough data is available to learn the representation
- In text, BERT-like models dominate for embeddings
- Similar ideas applied to texts, speech, graphs, electronic health records, relational data, time series, etc.



Transformer



T5 (Text-To-Text Transfer Transformer) models



Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y, Li, W. & Liu, P. J. (2020). Exploring the limits of transfer learning with a unified text-to-text transformer. *J. Mach. Learn. Res.*, *21*(140), 1-67.

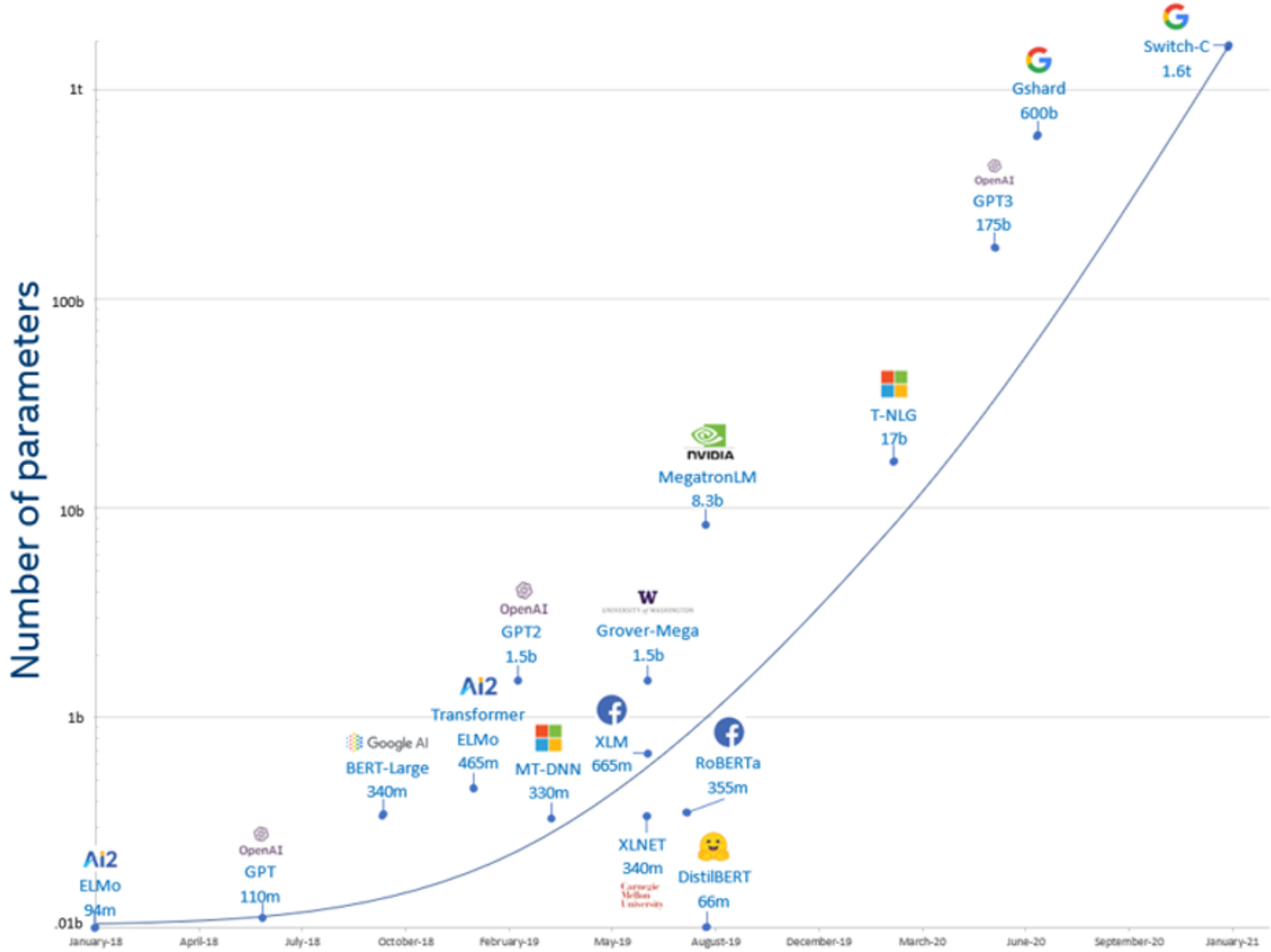
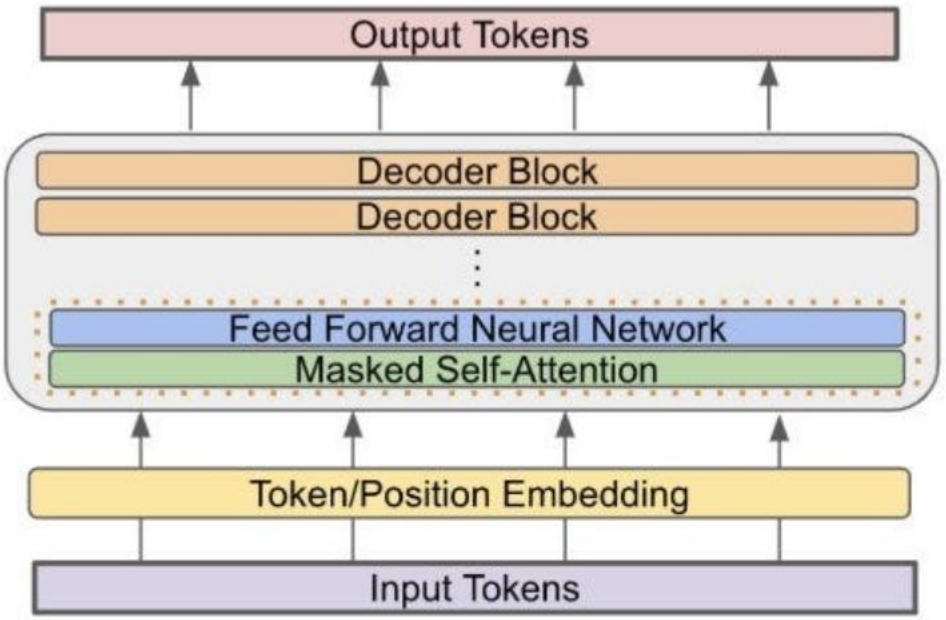


Figure 1: Exponential growth of number of parameters in DL models

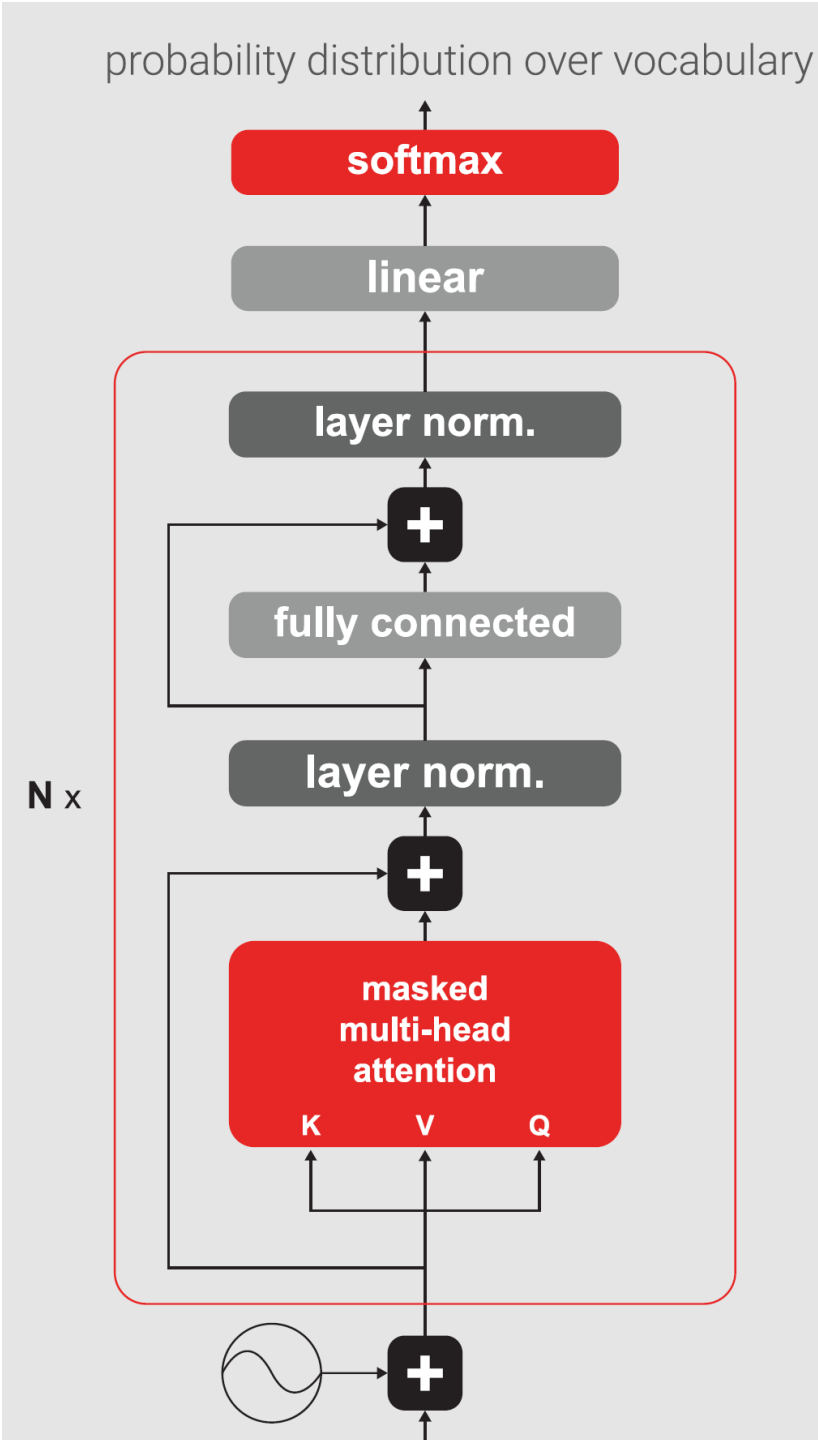
Decoder only models

- GPT, GPT-2, GPT-3, ChatGPT, GPT-4
- LLaMA, LLaMA-2, LLaMa-3
- MPT, Falcon
- Mistral and Mixtral
- OPT, Bloom
- Gemma and Gemini

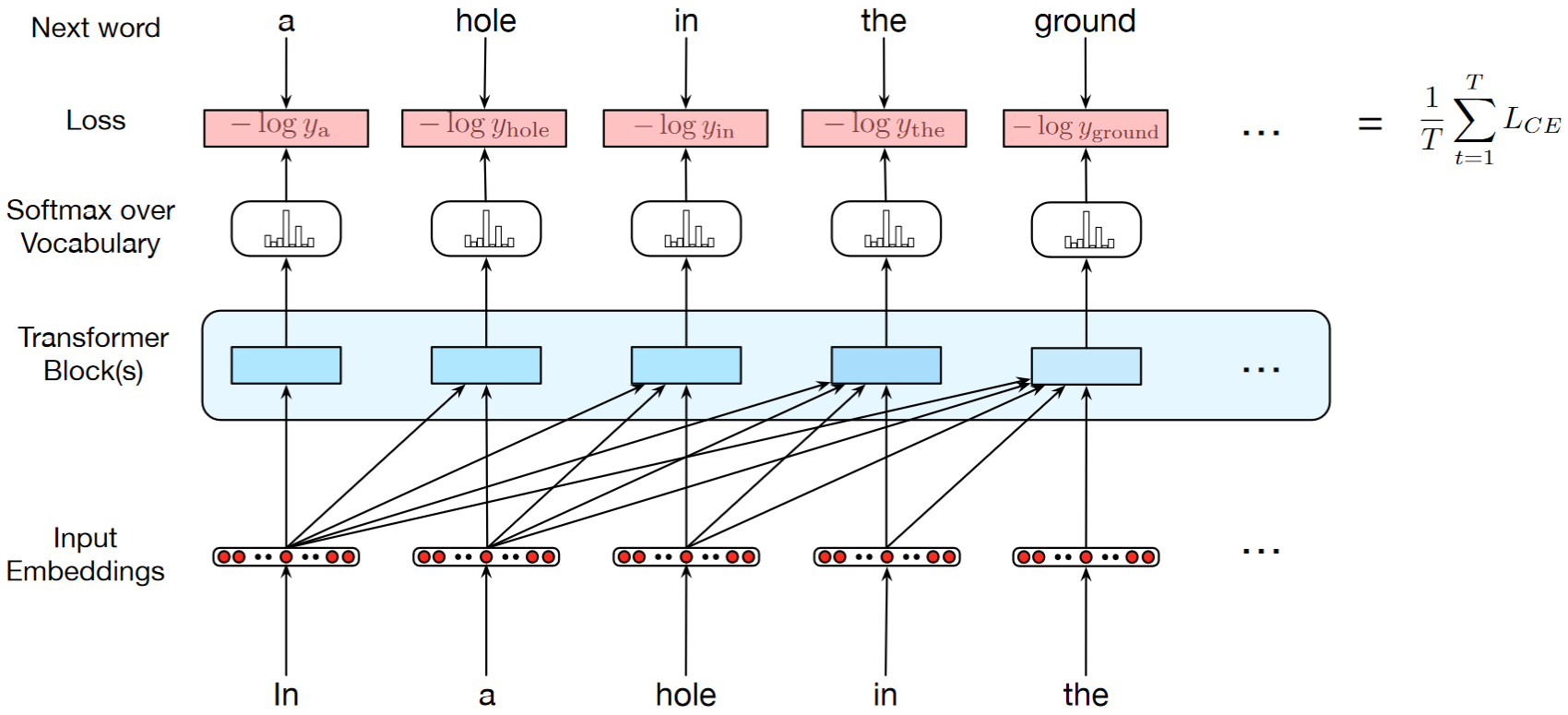


GPT family

- GPT: Generative Pre-trained Transformers
- use only the decoder part of transformer
- pretrained for language modeling (predicting the next word given the context)
- Potential shortcoming: unidirectional, does not incorporate bidirectionality
- “What are those?” he said while looking at my crocs.



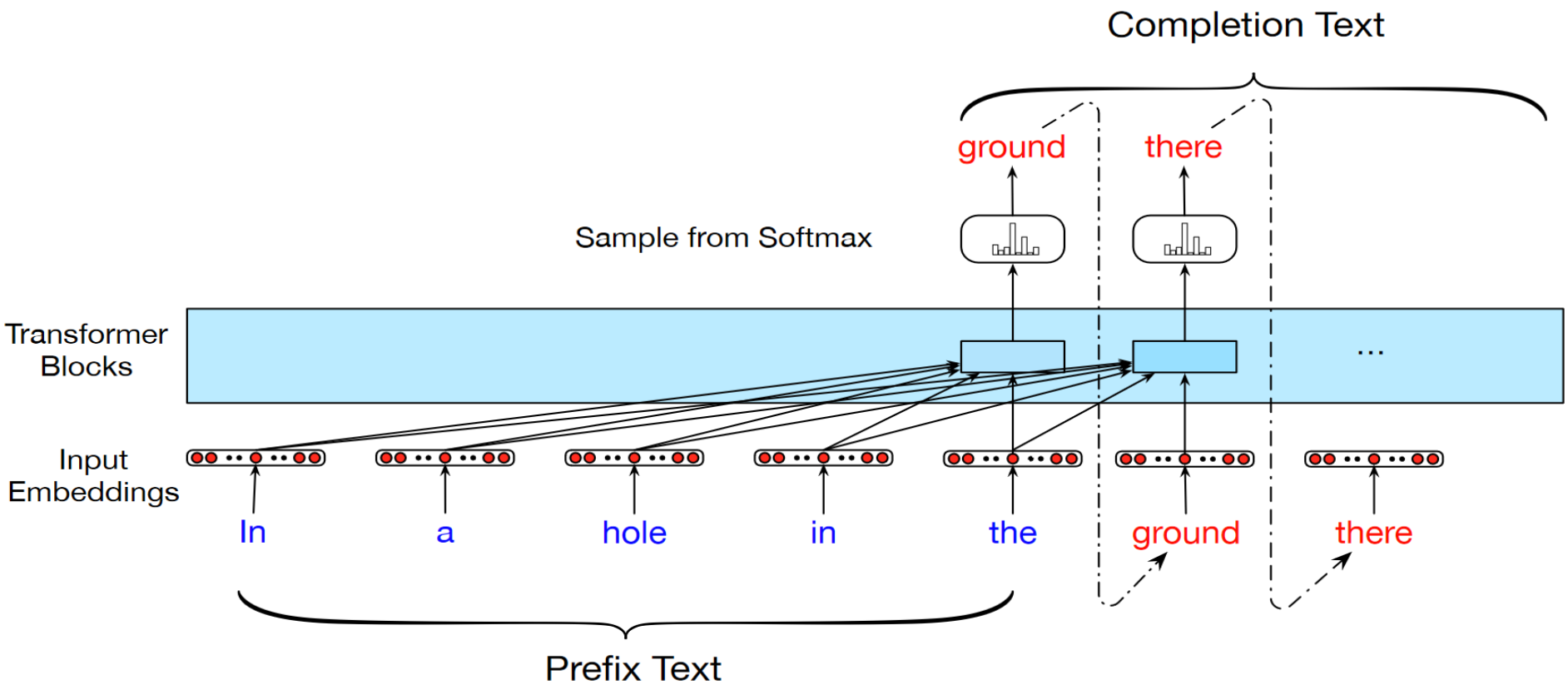
Transformer as a language model



- Can be computed in parallel

Autoregressive generators

- priming the generator with the context



- can be used also in summarization, QA and other generative tasks

GPT-2 and GPT-3

- few architectural changes, layer norm now applied to input of each subblock
- GPT-3 also uses some sparse attention layers
- more data, larger batch sizes (GPT-3 uses batch size of 3.2M)
- the models are scaled:

GPT-2:
48 layers, 25 heads
$d_m = 1600$, $d = 64$
context size = 1024
~ 1.5B parameters

GPT-3:
96 layers, 96 heads
$d_m = 12288$, $d = 128$
context size = 2048
~ 175B parameters

[1] Radford et al.: Language Models are Unsupervised Multitask Learners, 2019.
[2] Brown et al.: Language Models are Few-Shot Learners, 2020.

- see demos at https://transformer.huggingface.co/

In-context learning

- GPT-2 and GPT-3 were the first models where one can ditch the “pre-train and fine-tune” training paradigm of GPT;
- GPT-2 explores unsupervised zero-shot learning, whereas in GPT-3 the authors expand the idea into in-context learning;
- Use text input to condition the model on task description and some examples with ground truth.
- Uses zero-shot learning, one-shot learning, few-shot learning (as many examples as they can fit into the context, usually 10-100)
- No gradient updates are performed.
- A sort of associative lookup

In-context learning

Zero-shot

The model predicts the answer given only a natural language description of the task. No gradient updates are performed.

```
1  Translate English to French:        ← task description
2  cheese =>                           ← prompt
```

One-shot

In addition to the task description, the model sees a single example of the task. No gradient updates are performed.

```
1  Translate English to French:        ← task description
2  sea otter => loutre de mer          ← example
3  cheese =>                           ← prompt
```

Few-shot

In addition to the task description, the model sees a few examples of the task. No gradient updates are performed.

```
1  Translate English to French:        ← task description
2  sea otter => loutre de mer          ← examples
3  peppermint => menthe poivrée        ←
4  plush girafe => girafe peluche      ←
5  cheese =>                           ← prompt
```

Figure source: Brown et al.: Language Models are Few-Shot Learners, 2020.

- GPT-3 is still a language model and can be used for text generation

- only 12% of respondents correctly classified this as not written by a human

Title: United Methodists Agree to Historic Split
Subtitle: Those who oppose gay marriage will form their own denomination
Article: **After two days of intense debate, the United Methodist Church has agreed to a historic split - one that is expected to end in the creation of a new denomination, one that will be "theologically and socially conservative," according to The Washington Post. The majority of delegates attending the church's annual General Conference in May voted to strengthen a ban on the ordination of LGBTQ clergy and to write new rules that will "discipline" clergy who officiate at same-sex weddings. But those who opposed these measures have a new plan: They say they will form a separate denomination by 2020, calling their church the Christian Methodist denomination.**
The Post notes that the denomination, which claims 12.5 million members, was in the early 20th century the "largest Protestant denomination in the U.S.," but that it has been shrinking in recent decades. The new split will be the second in the church's history. The first occurred in 1968, when roughly 10 percent of the denomination left to form the Evangelical United Brethren Church. The Post notes that the proposed split "comes at a critical time for the church, which has been losing members for years," which has been "pushed toward the brink of a schism over the role of LGBTQ people in the church." Gay marriage is not the only issue that has divided the church. In 2016, the denomination was split over ordination of transgender clergy, with the North Pacific regional conference voting to ban them from serving as clergy, and the South Pacific regional conference voting to allow them.

Huge generative language models



- ChatGPT, OpenAI, Nov. 2022
  based on GPT-3.5 with additional training for dialogue
- uses instruction-tuning and RLHF
  (reinforcement learning with human feedback)
- demo: https://chat.openai.com/
- huge public impact, possibly disruptive for writing professions, learning, teaching, scientific writing
- GPT-4, 2023: even larger, allows longer context, image input

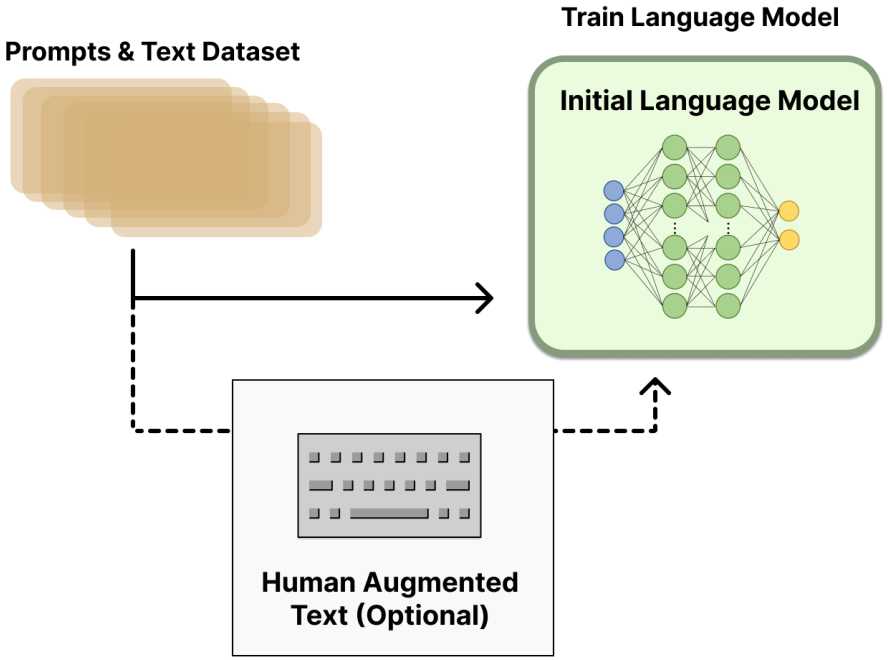
RLHF: idea

- Reinforcement Learning with Human Feedback
- A problem: Human feedback is not present during training
- Idea: Train a separate model on human feedback, this model can generate a reward to be used during training of LLM
- Three stages:
  1. Pretraining a language model (LM),
  2. Gathering data and training a reward model, and
  3. Fine-tuning the LM with reinforcement learning.

Nathan Lambert, Louis Castricato, Leandro von Werra, Alex Havrilla: "Illustrating Reinforcement Learning from Human Feedback (RLHF)", Hugging Face Blog, 2022.

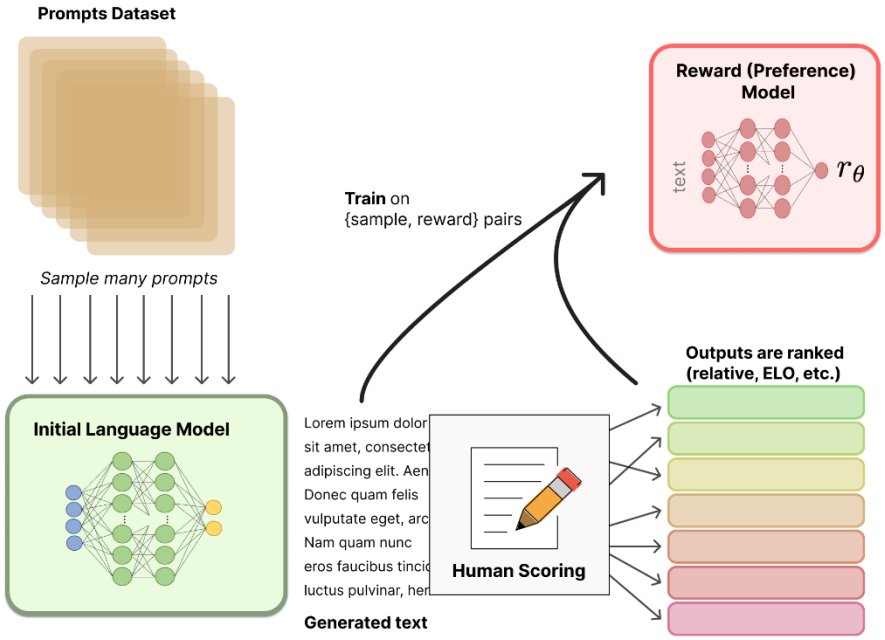
RLHF: the reward model 1/2

- input: a sequence of text, e.g., produced by LM and optionally improved by humans
- output: a scalar reward, representing the human preference of the text (e.g., a rank of the answer)
- the reward model could be an end-to-end LM, or the model ranks outputs, and the ranking is converted to reward



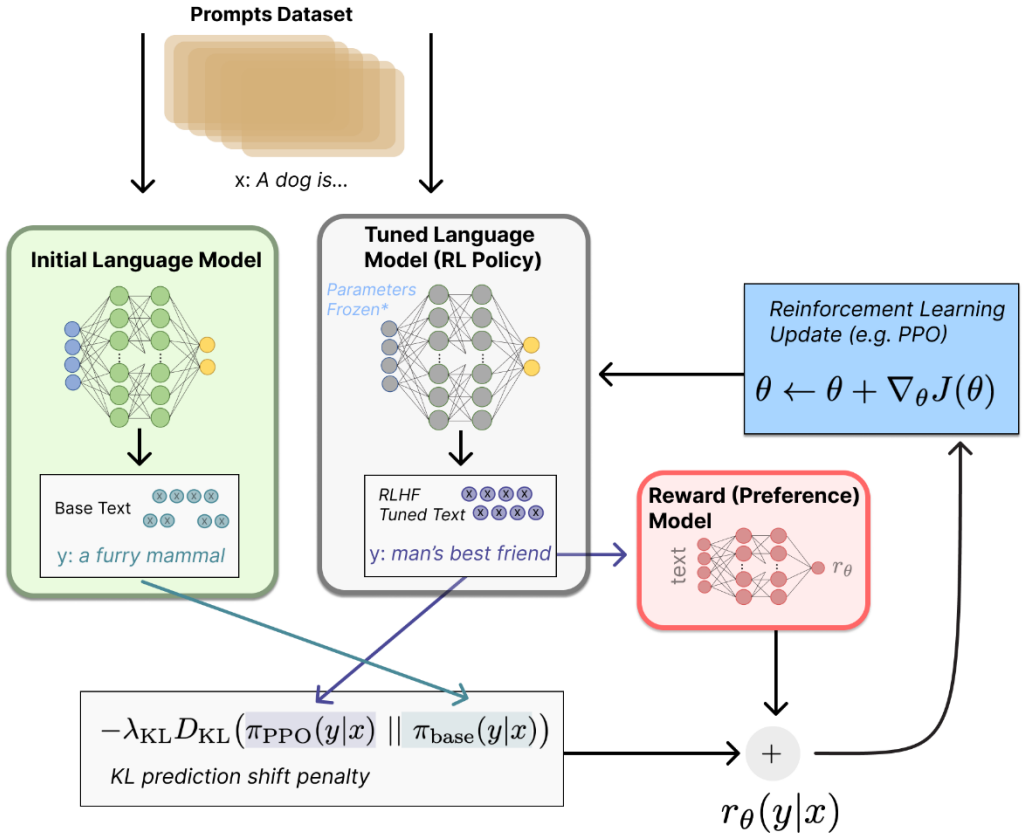
RLHF: the reward model 2/2

- the training dataset are pairs of prompts and (human improved) LM responses, e.g., 50k instances
- humans rank the responses instead of producing the direct reward as this produces better calibrated scores



RLHF: fine-tuning with RL

- RL does not change all parameters, most of parameters are frozen
- the algorithm: Proximal Policy Optimization (PPO)



NLP application examples

Sentiment analysis (SA)

- Definition: a computational study of opinions, sentiments, emotions, and attitudes expressed in texts towards an entity.
- Purpose: detecting public moods, i.e. understanding the opinions of the general public and consumers on social events, political movements, company strategies, marketing campaigns, product preferences etc.
- Part of Affective Computing (emotion, mood, personality traits, interpersonal stance, attitude)
- Can be target-based or general

SA: getting and preprocessing data

- Frequent data sources:
  - Twitter-X, forum comments, product review sites, company's Facebook pages
- Data cleaning
  - quality assessment, annotator (self-) agreement
  - preprocessing: tokenization, emojis, links, hashtags, etc,

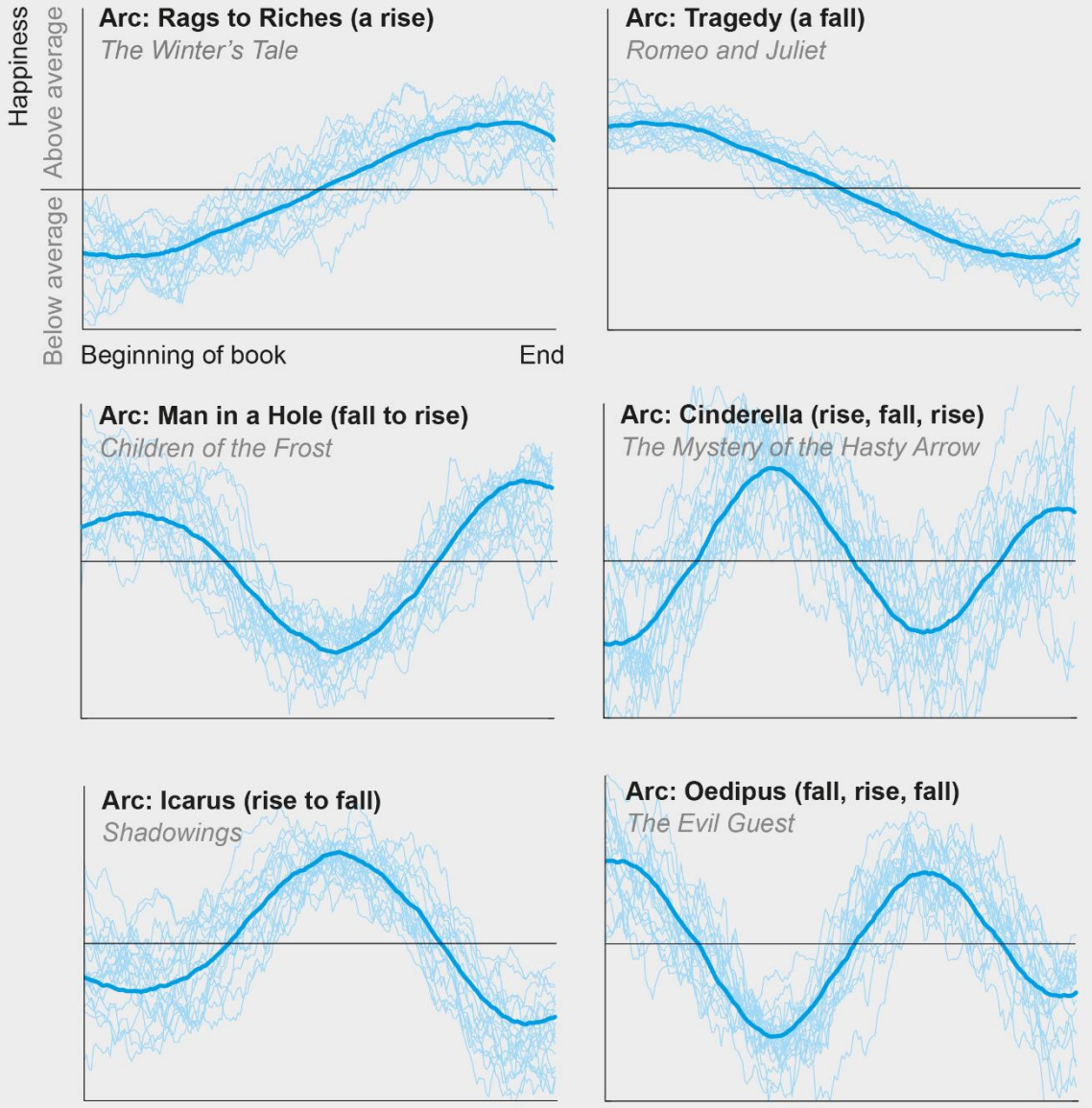
SA tasks

- sentiment classification (binary (polarity), ternary, n-ary
- subjectivity classification (vs. objectivity)
- review usefulness classification
- opinion spam classification
- emotion analysis
- hate speech, offensive speech, socially unacceptable speech
- stance detection

Emotional states in English fiction

Emotional Arcs

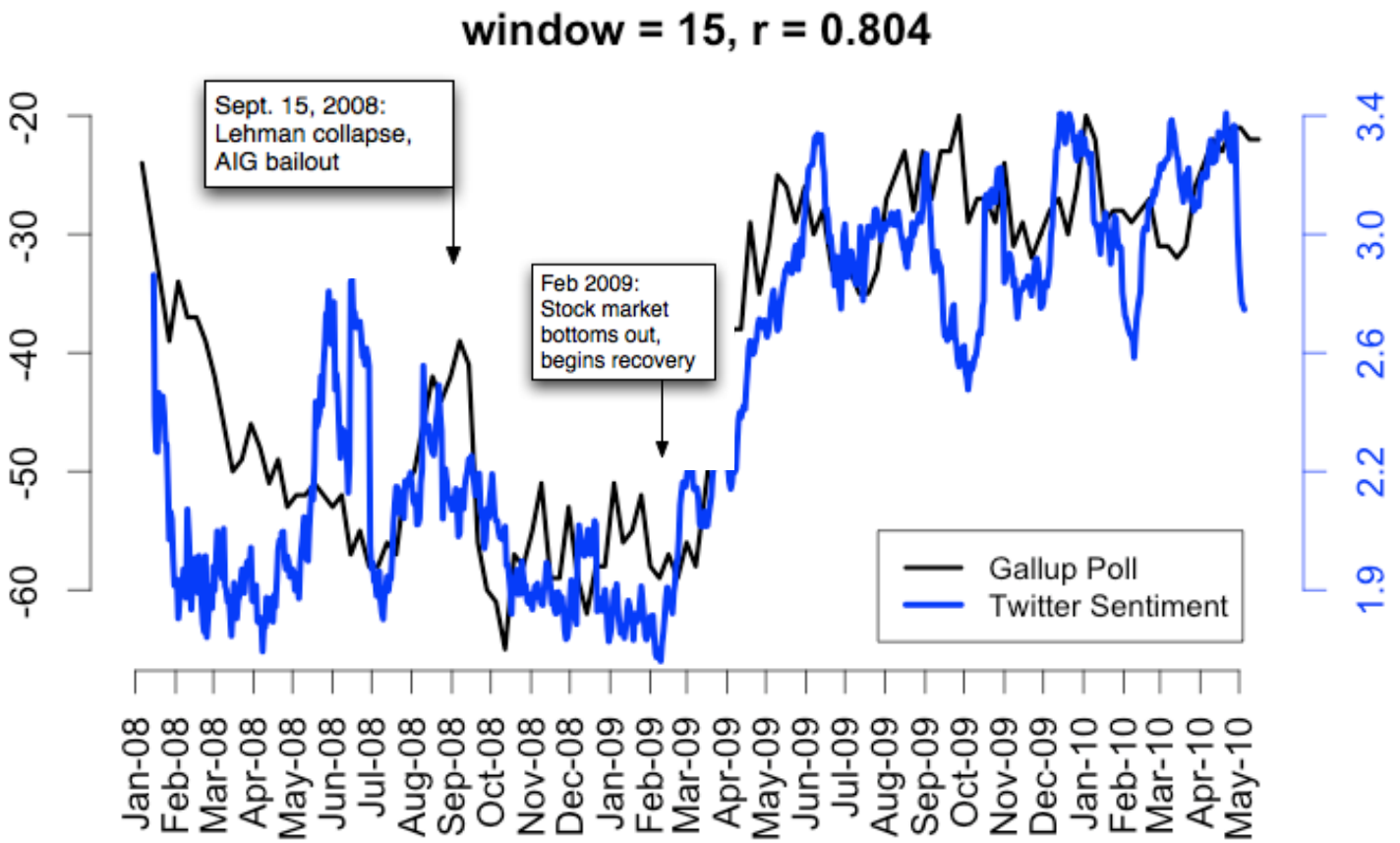
About 85 percent of 1,327 fiction stories in the digitized Project Gutenberg collection follow one of six emotional arcs—a pattern of highs and lows from beginning to end (*dark curves*). The arcs are defined by the happiness or sadness of words in the running text (*jagged plots*). All books were in English and less than 100,000 words; examples are noted.



Public opinion surveys

Twitter sentiment vs. Gallup on consumer sentiment

Brendan O'Connor, Ramnath Balasubramanyan, Bryan R. Routledge, and Noah A. Smith. 2010. From Tweets to Polls: Linking Text Sentiment to Public Opinion Time Series. In ICWSM-2010



Statistical machine translation

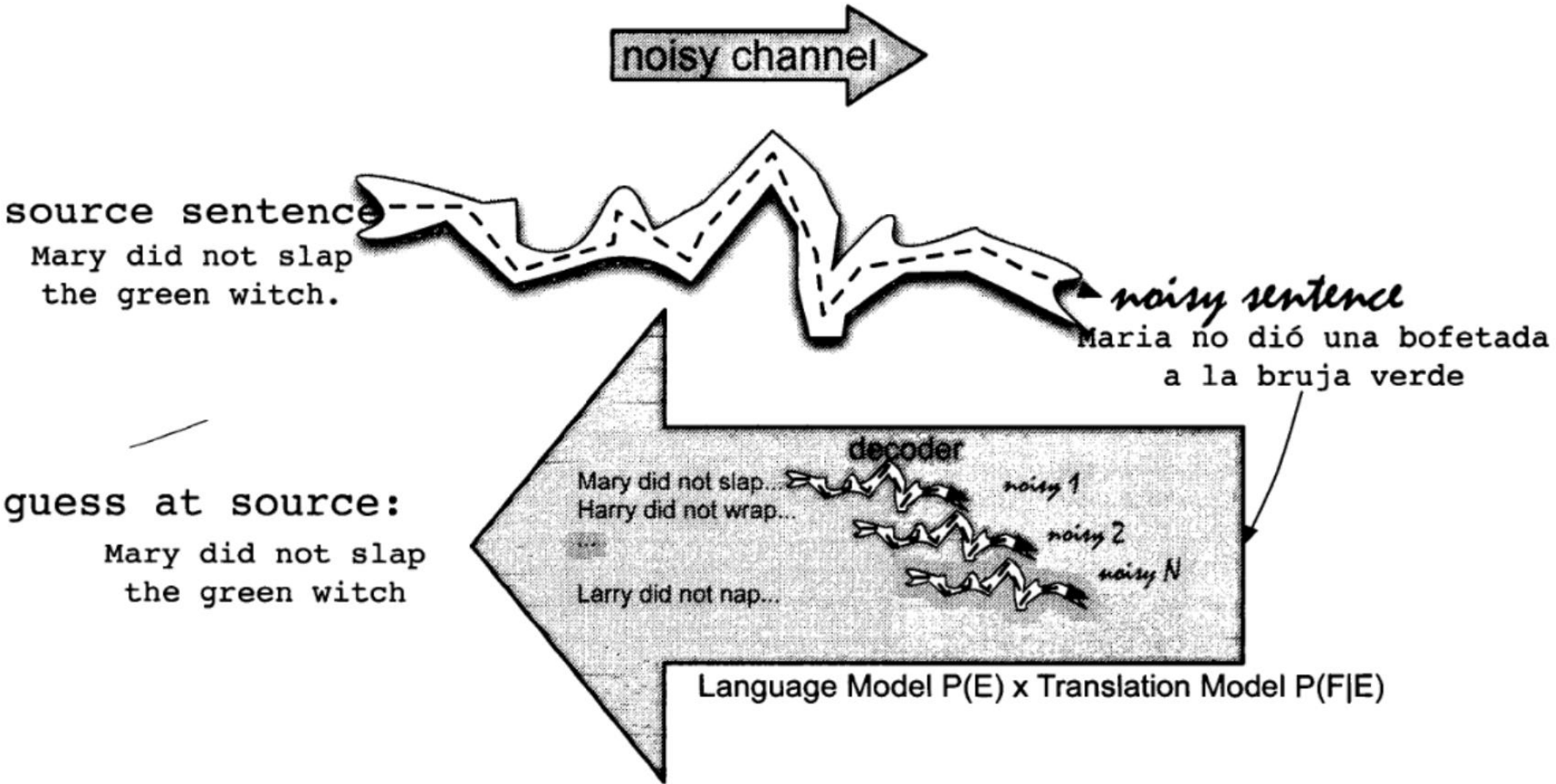
- non-neural approach: no longer used in practice but gives insight of what is needed
- idea from the theory of information
- we translate from foreign language F to English E
- a document is translated based on the probability distribution $p(e \mid f)$, i.e. the probability of the sentence e in target language based on the sentence in source language f
- Bayes rule
  $\arg\max_e p(e \mid f) = \arg\max_e p(f \mid e)\, p(e) \,/\, p(f)$
- $p(f)$ can be ignored as it is a constant for a given fixed sentence
- traditional (non-neural) approaches split the problem into subproblems
  - create a language model $p(e)$
  - a separate translation model $p(f \mid e)$
  - decoder forms the most probable e based on f

Noisy channel model

- given English sentence e
- during transmission over a noisy channel the sentence e is corrupted and we get sentence in a foreign language f, which we are able to observe
- to reconstruct the most probable sentence e we have to figure out:
  - how people speak in English (language model), p(e) and
  - how to transform a foreign language into English (translation model), p(f | e)

Noisy channel

- reasoning goes back



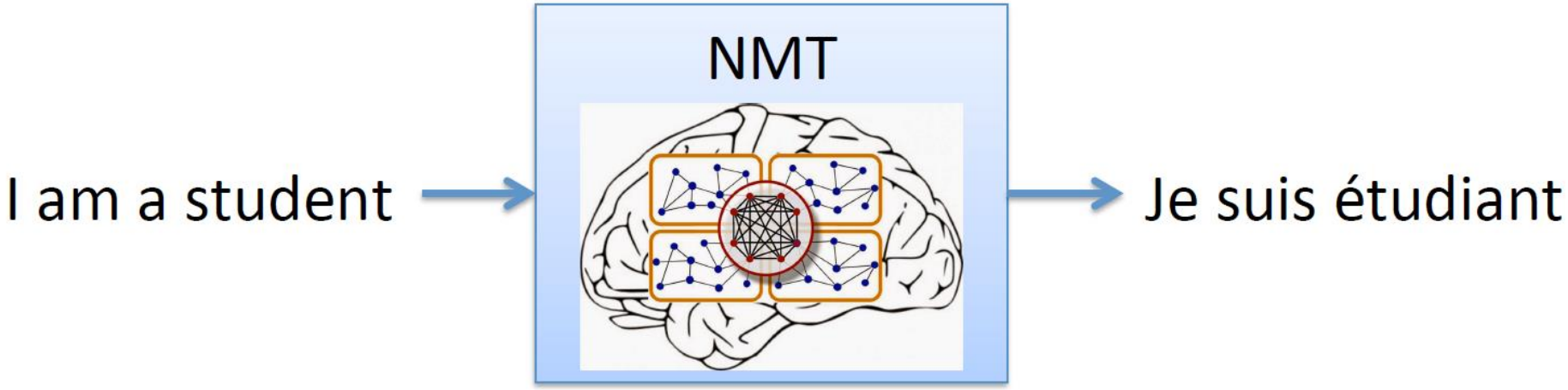
Language model

- each target (English) sentence e is assigned a probability p(e)
- estimation of probabilities for the whole sentences is not possible (why?), therefore we use language models, e.g., 3-gram models or neural language models

Translation model

- we have to assign a probability of p(f | e), which is a probability of a foreign language sentence f, given target sentence e.
- we search the e which maximizes p(e) * p(f | e)
- traditional MT approach: using translation corpus we determine which translation of a given word is the most probable
- we take into account the position of a word and how many words are needed to translate a given word

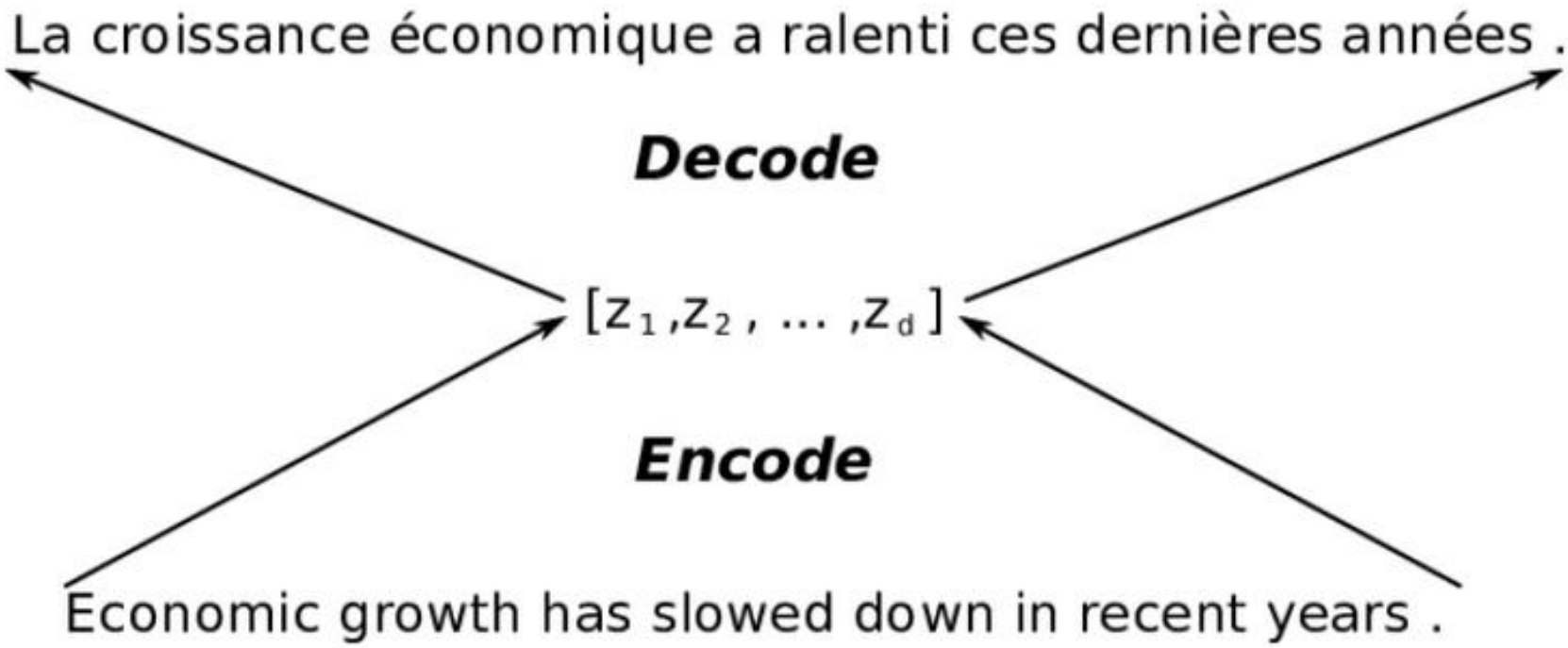
Neural machine translation (NMT)



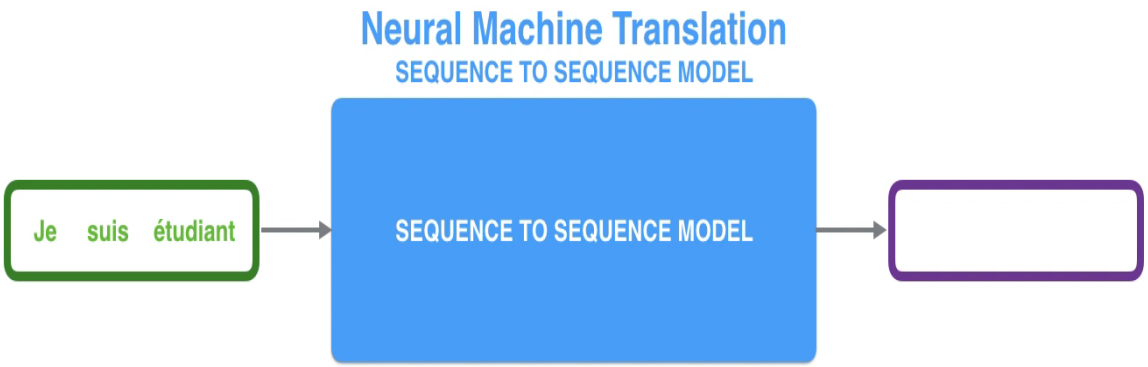
(Sutskever et al., 2014; Cho et al., 2014)

- sequence to sequence machine translation (seq2seq)
- end-to-end optimization

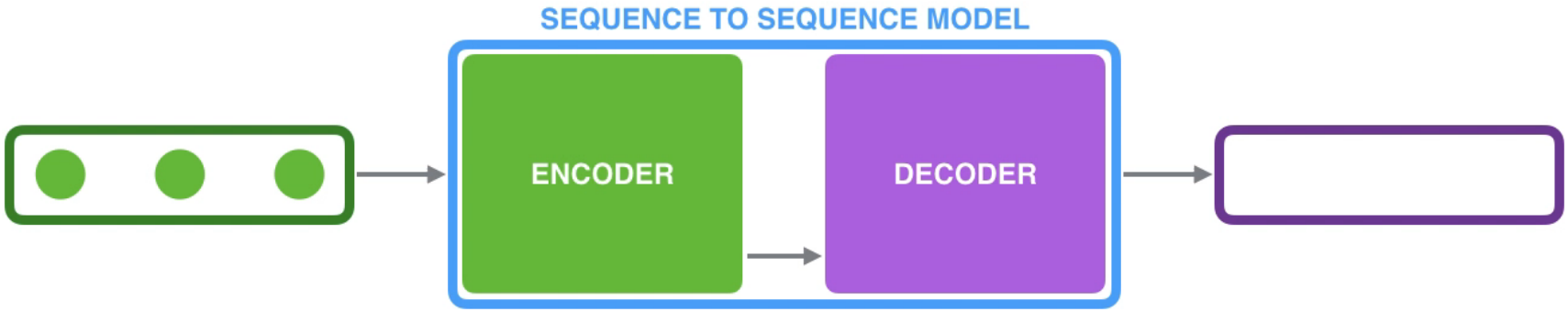
Encoder-Decoder model



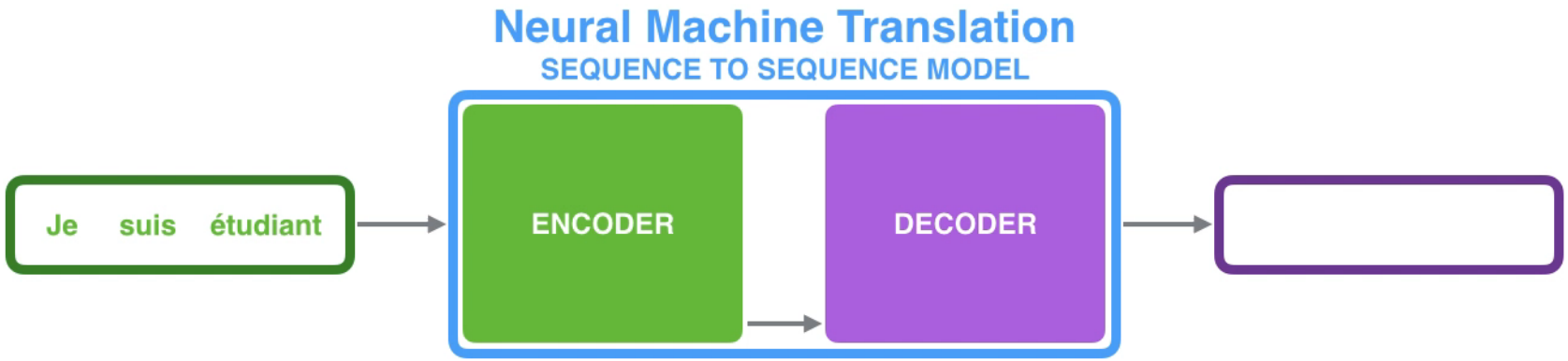
Seq2Seq for NMT

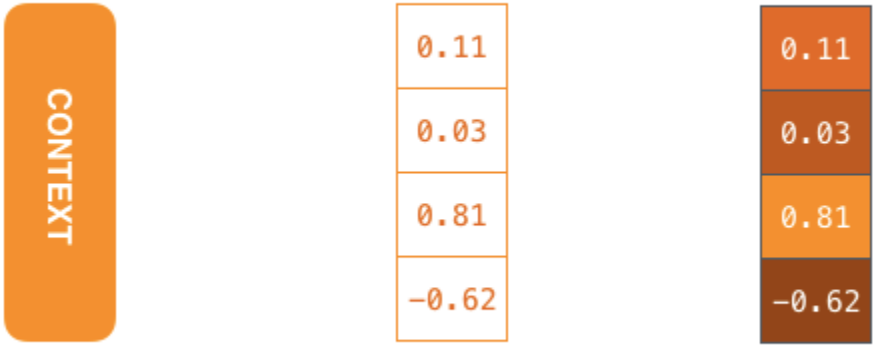


Encoder-decoder for sequences



Encoder-decoder for NMT

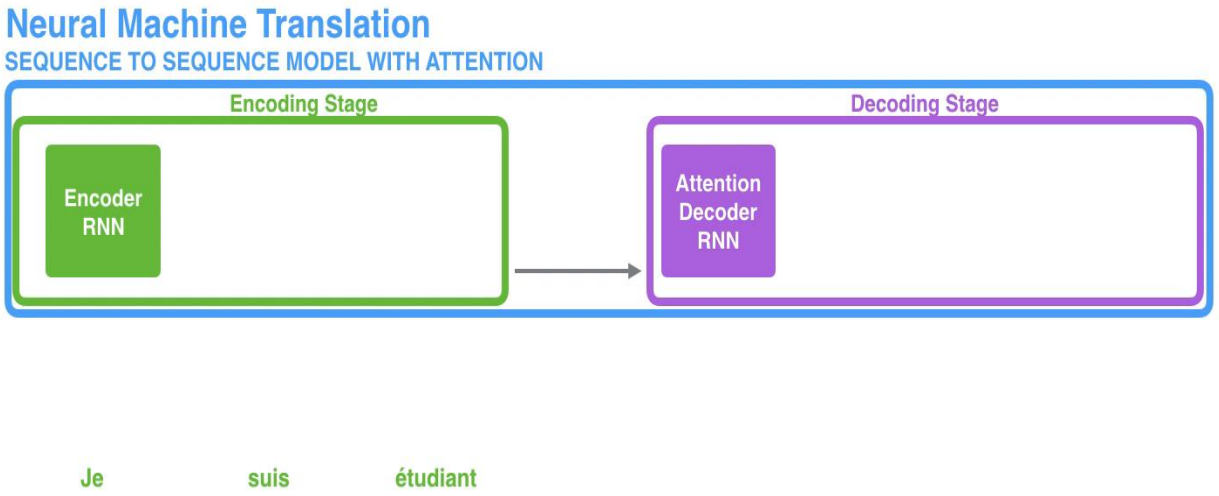




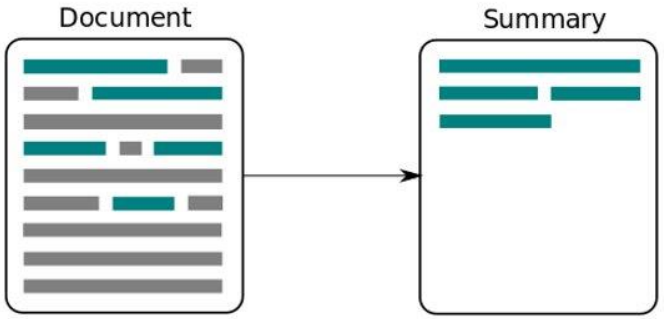
Training NMT

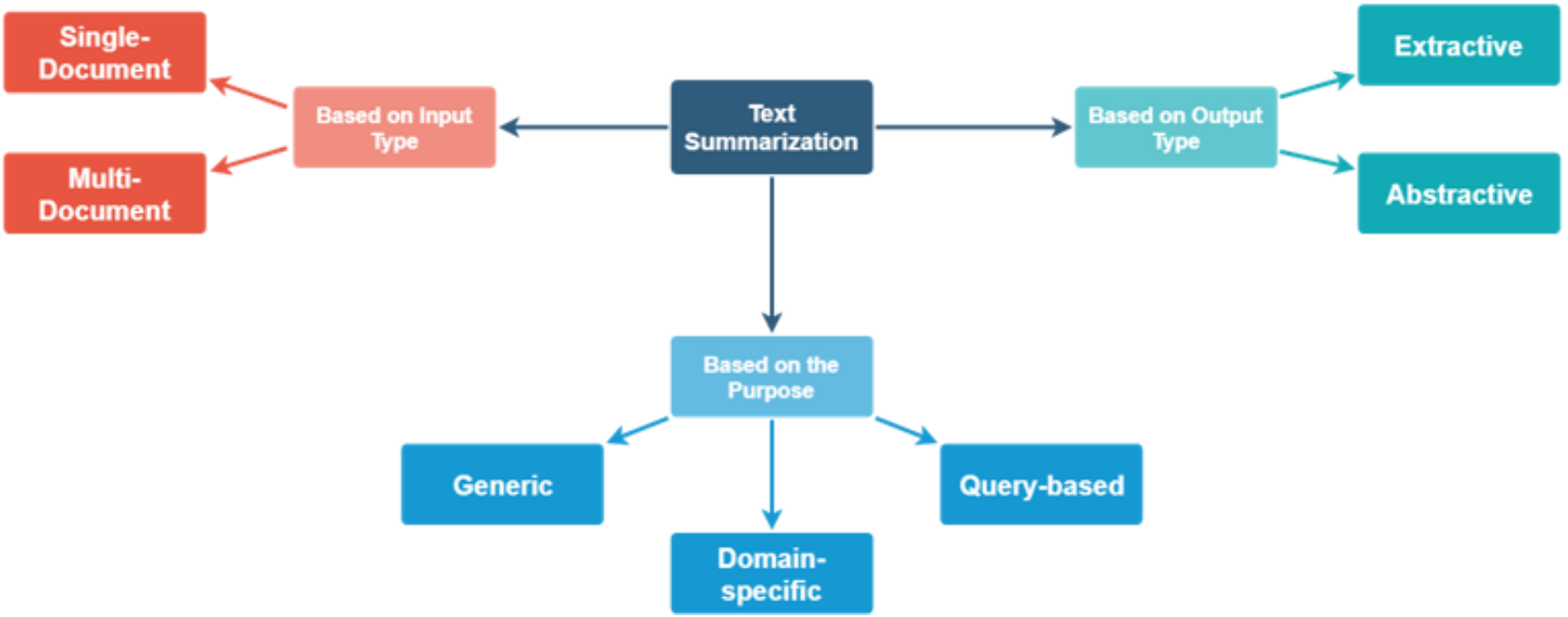
- using transformers
- softmax for output
- we maximize
  P(output sentence | input sentence)
- we sum errors on all outputs
- backpropagation
- training on correct translations
- as the translation, we return a sequence of words with the highest probability (not necessary greedily)

NMT with attention



Text summarization



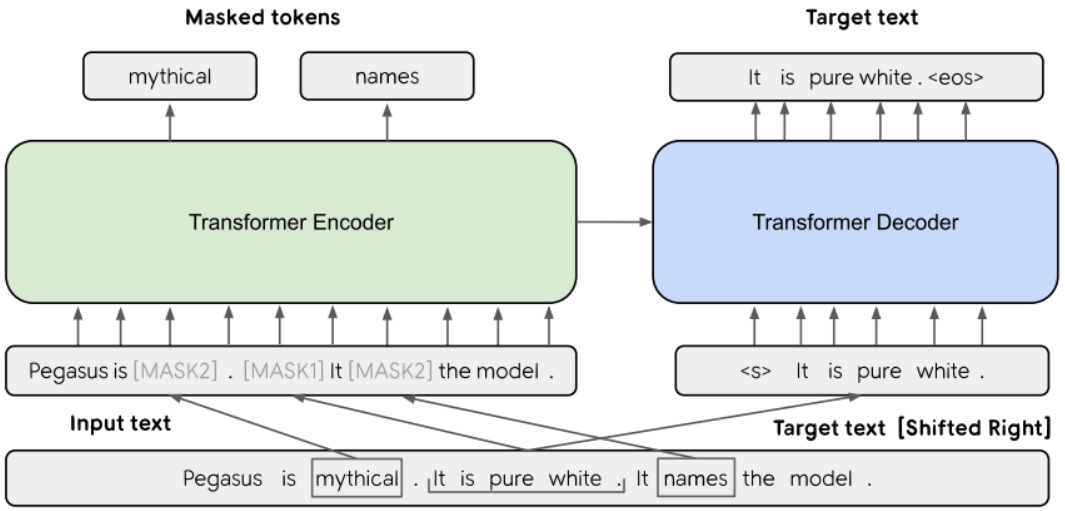


Text summarization

- Evaluation:
  - ROUGE scores,
  - BERTScore,
  - with QA:
    - question generation,
    - searching for answers in the summary
  - human
- LLMs
- Short and long texts
- cross-lingual

Summarizers – Pegasus

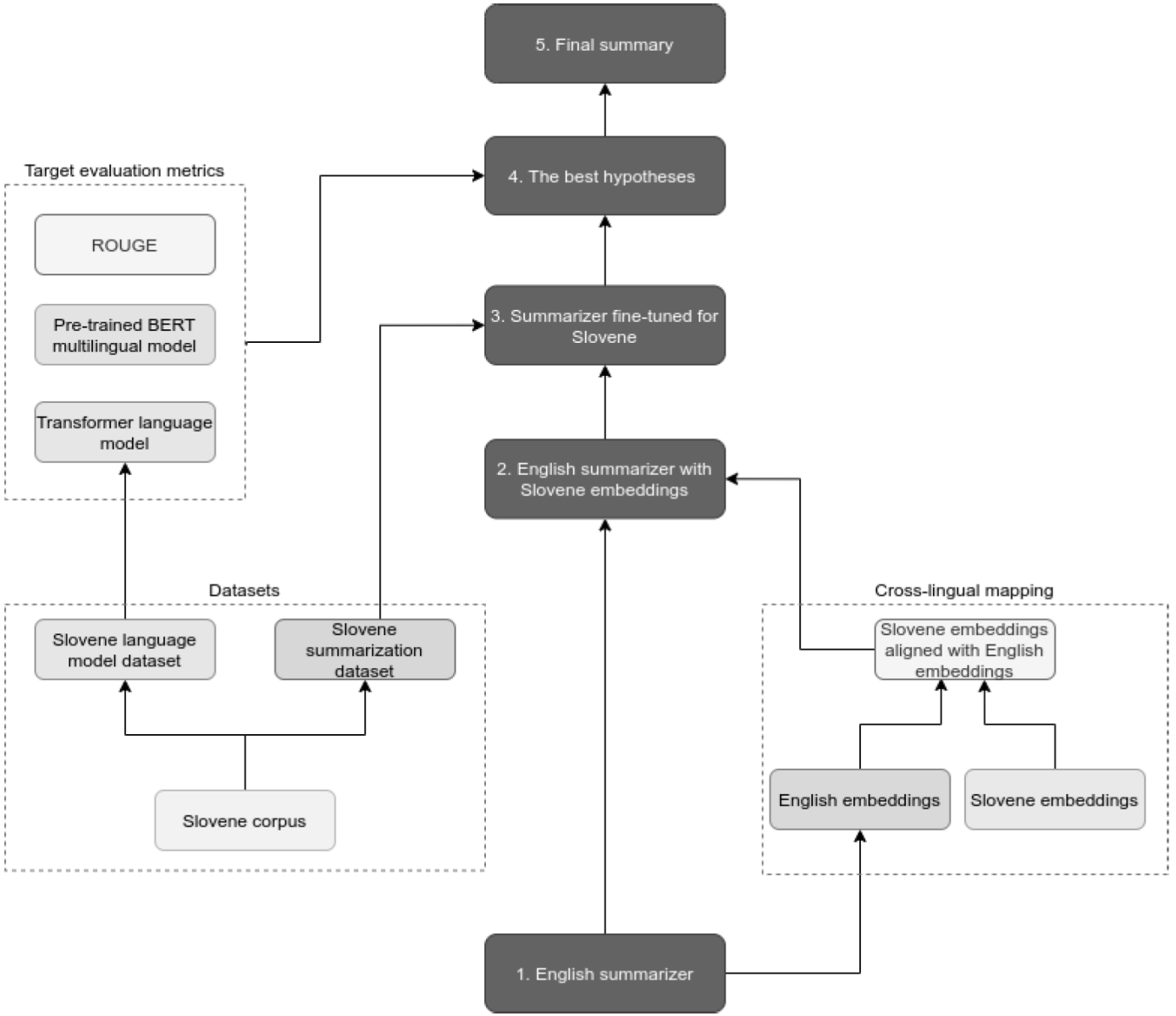
- An example of a different, successful transformer model
- Transformer BART model
- encoder-decoder architecture
- text garbling and reconstruction
- Auxiliary tasks: masked language model and missing sentence generation
- Demo: https://ai.googleblog.com/2020/06/pegasus-state-of-art-model-for.html



(Zhang et al., 2019)

Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

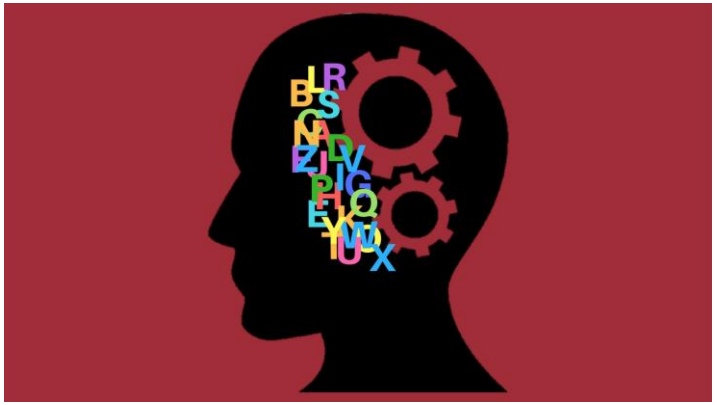
XL summarization architecture



Žagar, A., & Robnik-Šikonja, M. (2022). Cross-lingual transfer of abstractive summarizer to less-resource language. Journal of Intelligent Information Systems, 58(1), 153-173.

University of Ljubljana, Faculty of Computer and Information Science

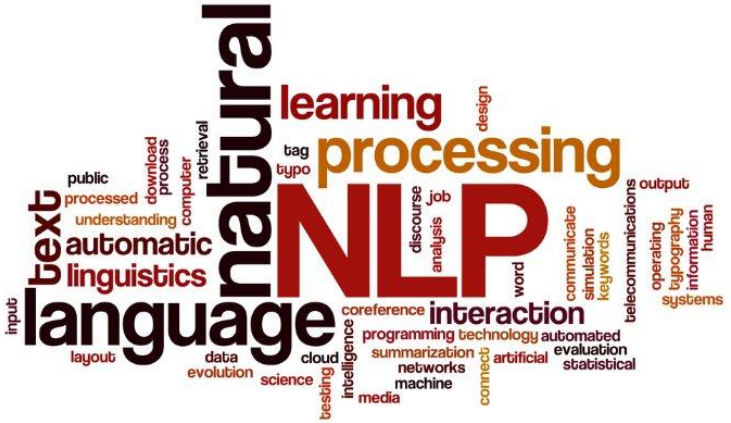
Natural language processing



Prof Dr Marko Robnik-Šikonja

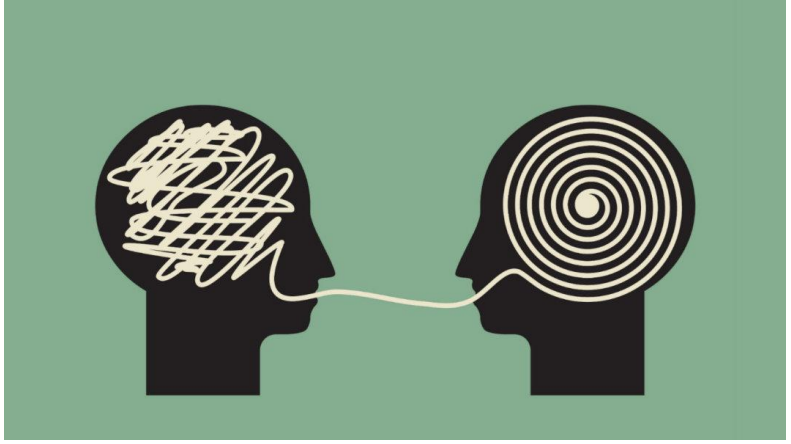
Intelligent Systems, Edition 2025

Topic overview



- understanding language and intelligence
- text preprocessing and linguistic analysis
- components of modern NLP
  - text representations
  - information retrieval
  - similarity of words and documents
  - language and graphs
  - large language models
- practical use of NLP:
  - sentiment analysis,
  - paper recommendations
  - summarization

Understanding language



- A grand challenge of (not only?) artificial intelligence

  Who can understand me?
  Myself I am lost
  Searching but cannot see
  Hoping no matter cost
  Am I free?
  Or universally bossed?

- Not just poetry, what about instructions, user manuals, newspaper articles, seminary works, internet forums, twits, legal documents, i.e. license agreements, etc.
- Can LLMs really understand language?

An example: rules

Article 18 of FRI Study Rules and Regulations

Taking exams at an earlier date may be allowed on request of the student by the Vice-Dean of Academic Affairs with the course convener's consent in case of mitigating circumstances (leaving for study or placement abroad, hospitalization at the time of the exam period, giving birth, participation at a professional or cultural event or a professional sports competition, etc.), and if the applicant's study achievements in previous study years are deemed satisfactory for such an authorization to be appropriate.

Understanding NL by computers

- Understanding words, syntax, semantics, context, writer's intentions, knowledge, background, assumptions, bias …
- Doesn't seem that LLM do it, though they generate excellent text output
- Ambiguity in language
  - Newspaper headlines - intentional ambiguity :)
    - Juvenile court to try shooting defendant
    - Kids make nutritious snacks
    - Miners refuse to work after death
    - Doctor on Trump's health: No heart, cognitive issues

Ambiguity

- I made her duck.

- Possible interpretations:
  - I cooked waterfowl for her.
  - I cooked waterfowl belonging to her.
  - I created the (plaster?) duck she owns.
  - I caused her to quickly lower her head or body.
  - I waved my magic wand and turned her into undifferentiated waterfowl.

- Spoken ambiguity
  - eye, maid

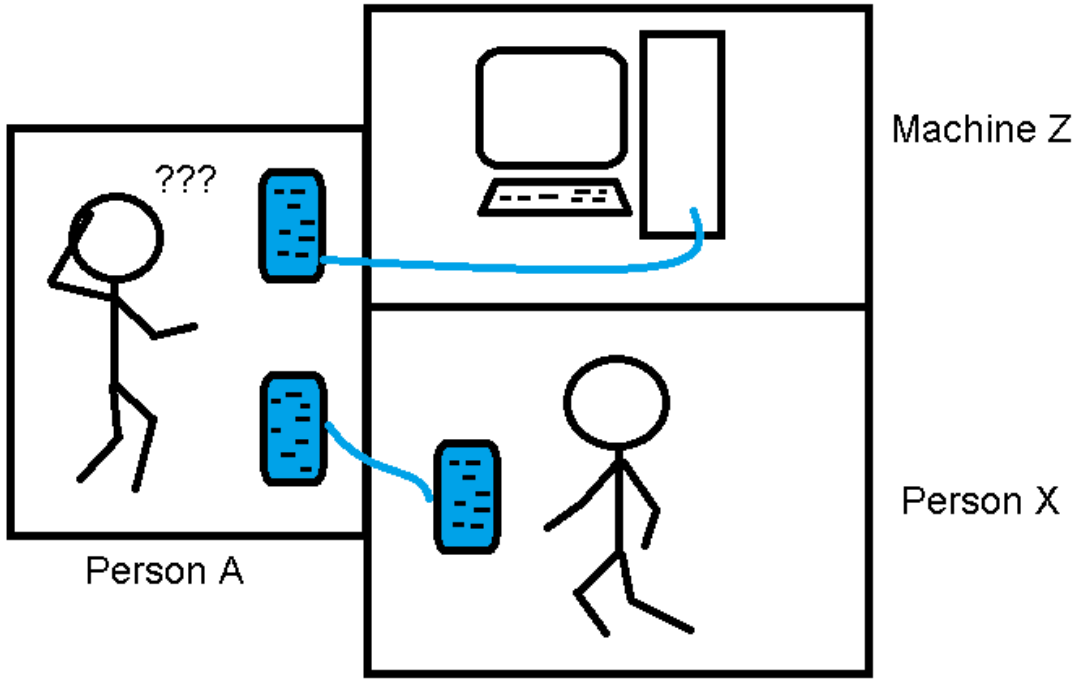
Disambiguation in syntax and semantics

- in syntax
  - The girl told the story cried.
  - John went to the bank.
  - Put the box on the table in the kitchen.
  - Bring your old car seat to be recycled.
- *Search engine queries: natural language processing, jaguar, Paris,*
- *POS tagging, word sense disambiguation*

Understanding language as (artificial) intelligence test

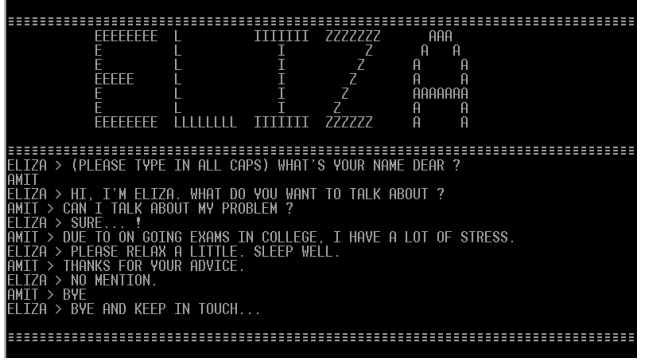
- The system understanding natural language on human level should contain a large portion of human knowledge and would achieve the most important goals of artificial intelligence. Its response would be indistinguishable from human's.

Turing test



- Turing test, Loebner prize, CYC, ELISA, ALLICA, chatterbot, VIDA, TIA, Siri (knowledge recognition, context detection, web service recommendation)
- Seems that LLMs can pass it, are they intelligent?
- The question: is mimicking human skills sufficient?

ELIZA



- ELIZA (Weizenbaum, 1966)
- Limited conversation (Rogerian psyhiatrist)

User: You are like my father in some ways.
ELIZA: What resemblance do you see?
User: You are not very aggressive but I think you don't want me to notice that.
ELIZA: What makes you think I am not aggressive?
User: You don't argue with me.
ELIZA: Why do you think I don't argue with you?
User: You are afraid of me.
ELIZA: Does it please you to believe I am afraid of you?

GPT-3 regressive language model as a chatbox

- the example is cherry-picked

```
Q: What is your favorite animal?
A: My favorite animal is a dog.

Q: Why?
A: Because dogs are loyal and friendly.

Q: What are two reasons that a dog might be in a bad mood?
A: Two reasons that a dog might be in a bad mood are if it is hungry or if it is hot.

Q: How many eyes does a giraffe have?
A: A giraffe has two eyes.

Q: How many legs does a frog have?
A: A frog has four legs.

Q: Are there any animals with three legs?
A: No, there are no animals with three legs.

Q: Why don't animals have three legs?
A: Animals don't have three legs because they would fall over.
```

Where is NLP today?

- active research area with many commercial applications
  - large language models are used everywhere
  - automatic reply engines
  - machine translation
  - text summarization
  - question answering
  - speech recognition and synthesis
  - language generation
  - interface to databases
  - intelligent search and information extraction
  - sentiment detection
  - named entity recognition and linking
  - categorization and classification of documents, messages, twits, etc.
  - cross-lingual approaches
  - multi modal approaches (text + images, text + video)
  - attempt to get to artificial general intelligence (AGI) through "foundation models"
  - many (open-source) tools and language resource
  - prevalence of deep neural network approaches (i.e. transformers)

Recommended literature

- Jurafsky, Daniel and James Martin (2025): Speech and Language Processing, 3rd edition in progress, almost all parts are available at authors‘ webpages https://web.stanford.edu/~jurafsky/slp3/
- Steven Bird, Ewan Klein, and Edward Loper. Natural Language Processing with Python. O’Reilly, 2009
  - a free book accompanying NLTK library, regularly updated
  - *Python 3, http://www.nltk.org/book/*
- Coursera
  - several courses, e.g., Stanford NLP with DNN

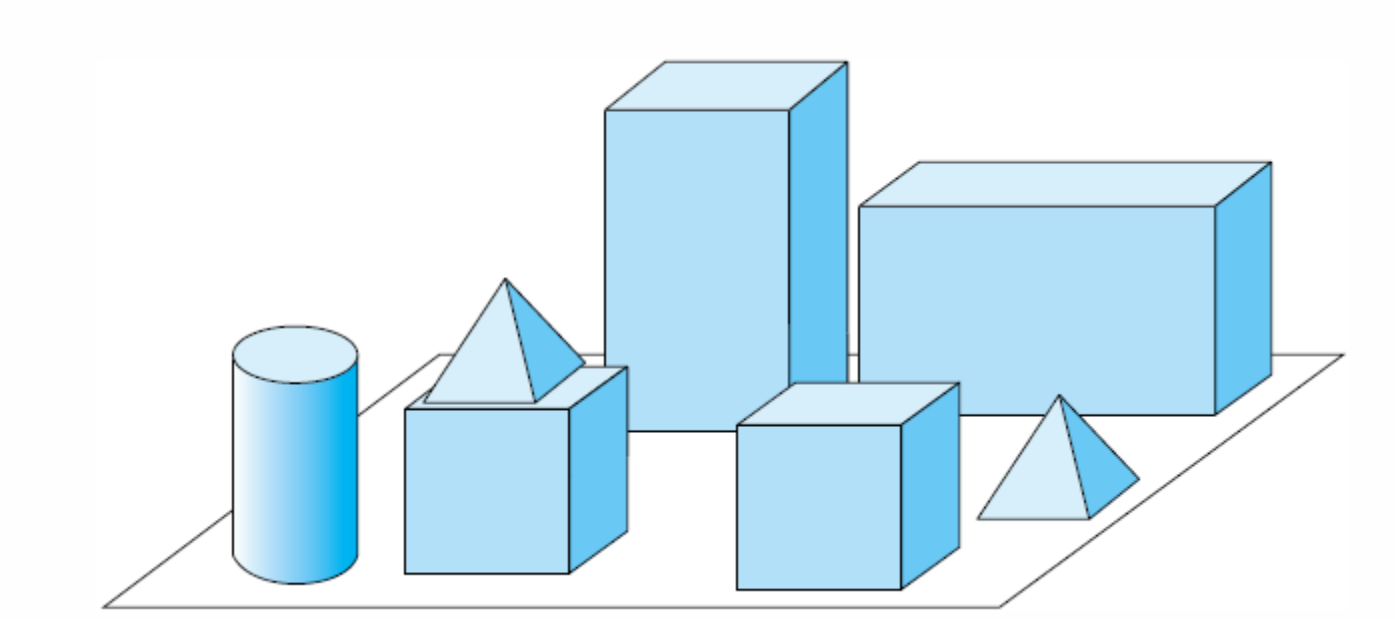
Historically two approaches

- symbolical
  - ,,Good Old-Fashioned AI''
- empirical
  - Statistical, text corpora
- Merging both worlds: injecting symbolical knowledge (e.g., propositional logic) into LLMs

How it all started?

- micro worlds
- example: SHRDLU, world of simple geometric objects
  - What is sitting on the red block?
  - What shape is the blue block on the table?
  - Place the green pyramid on the red brick.
  - Is there a red block? Pick it up.
  - What color is the block on the blue brick? Shape?

Micro world: block world, SHRDLU (Winograd, 1972)



Linguistic analysis 1/2

Linguistic analysis contains several tasks: recognition of sounds, letters, word formation, syntactic parsing, recognizing semantic, emotions. Phases:

- Prosody - the patterns of stress and intonation in a language (rhythm and intonation)
- Phonology - systems of sounds and relationships among the speech sounds that constitute the fundamental components of a language
- Morphology - the admissible arrangement of sounds in words; how to form words, prefixes and suffixes …
- Syntax - the arrangement of words and phrases to create well-formed sentences in a language

Linguistic analysis 2/2

- Semantics - the meaning of a word, phrase, sentence, or text
- Pragmatics - language in use and the contexts in which it is used, including such matters as deixis (words whose meaning changes with context, e.g., I he, here, there, soon), taking turns in conversation, text organization, presupposition, and implicature

  Can you pass me the salt? Yes, I can.
- Knowing the world: knowledge of physical world, humans, society, intentions in communications …
- Limits of linguistic analysis, levels are dependent

Classical approach to text processing

- text preprocessing
- 1. phase: syntactic analysis
- 2. phase: semantic interpretation
- 3. phase: use of world knowledge

In the neural approach, the preprocessing remains (but is simpler) the other three phases are merged into DNN

Basic tools for (classical) text preprocessing

- document → paragraphs → sentences → words (→ (subword) tokens)

- In linguistic analysis also
  - words and sentences ← lemmatization, POS tagging
  - sentences ← syntactical and grammatical analysis
  - named entity recognition,

Words and sentences

- sentence delimiters – punctuation marks and capitalization are insufficient
- E.g., remains of 1. Timbuktu from 5c BC, were discovered by dr. Barth.
- Lexical analysis (tokenizer, word segmenter), not just spaces
  - 1,999.00€ or 1.999,00€!
  - Ravne na Koroškem
  - Lebensversicherungsgesellschaft
  - Port-au-prince
  - Generalstaatsverordnetenversammlungen
- Rules, regular expressions, statistical models, dictionaries (of proper names), neural networks, manually segmented datasets

Lemmatization

- Lemmatization is the process of grouping together the different inflected forms of a word so they can be analyzed as a single item.
- walk is the lemma of 'walk', 'walked', 'walks', 'walking
- Lemmatization difficulty is language dependent i.e., depends on morphology
- Requires a dictionary or lexicon for lookup
  go, goes, going, gone, went
  jaz, mene, meni, mano
- Ambiguity resolution may be difficult

Meni je vzel z mize (zapestnico).

- Uses rules, dictionaries, neural networks, and manually labelled datasets
- English also uses stemming (reducing inflected or derived words to their word stem

POS tagging

- assigning the correct part of speech (noun, verb, etc.) to words
- helps in recognizing phrases and names
- Use rules, machine learning models, manually labelled datasets

An example:

- Text analyzer for Slovene, i.e. morphosyntactical tagging, available at https://orodja.cjvt.si/oznacevalnik/slv/

  Nekega dne sem se napotil v naravo. Že spočetka me je žulil čevelj, a sem na to povsem pozabil, ko sem jo zagledal. Bila je prelepa. Povsem nezakrita se je sončila na trati ob poti. Pritisk se mi je dvignil v višave. Popoln primerek kmečke lastovke!

- Tags are standardized, for East European languages in Multext-East specification, e.g.,

dne; tag Somer = Samostalnik, obče ime, moški spol, ednina, rodilnik; lema: dan

a unifying attempt: universal dependencies (UD): cross-linguistically consistent treebank annotation for many languages

CLASSLA-Stanza pipeline output

ID	Oblika	Lema	Oznaka JOS	Bes. vrsta JOS	Oblikoskladenjske lastnosti JOS	Bes. vrsta UD	Oblikoskladenjske lastnosti UD	Nadrejena pojavnica UD	Skladenjska relacija UD	Nadrejena pojavnica JOS	Skladenjska relacija JOS	Udeleže vloga
# paragraph 1												
# sent_id 1.1												
# text = Nekega dne sem se napotil v naravo.												
1	Nekega	nek	Pi-msg	Pronoun	Type=indefinite Gender=masculine Number=singular Case=genitive	DET	Case=Gen|Gender=Masc|Number=Sing|PronType=Ind	2	det	2	Atr	
2	dne	dan	Ncmsg	Noun	Type=common Gender=masculine Number=singular Case=genitive	NOUN	Case=Gen|Gender=Masc|Number=Sing	5	obl	5	AdvO	TIME
3	sem	biti	Va-r1s-n	Verb	Type=auxiliary VForm=present Person=first Number=singular Negative=no	AUX	Mood=Ind|Number=Sing|Person=1|Polarity=Pos|Tense=Pres|VerbForm=Fin	5	aux	5	PPart	
4	se	se	Px------y	Pronoun	Type=reflexive Clitic=yes	PRON	PronType=Prs|Reflex=Yes|Variant=Short	5	expl	5	PPart	
5	napotil	napotiti	Vmep-sm	Verb	Type=main Aspect=perfective VForm=participle Number=singular Gender=masculine	VERB	Aspect=Perf|Gender=Masc|Number=Sing|VerbForm=Part	0	root	0	Root	
6	v	v	Sa	Adposition	Case=accusative	ADP	Case=Acc	7	case	7	Atr	
7	naravo	narava	Ncfsa	Noun	Type=common Gender=feminine Number=singular Case=accusative	NOUN	Case=Acc|Gender=Fem|Number=Sing	5	obl	5	AdvO	GOAL
8	.	.	Z	Punctuation		PUNCT	_	5	punct	0	Root	

- *Nekega dne sem se napotil v naravo. Že spočetka me je žulil čevelj, a sem na to povsem pozabil, ko sem jo zagledal. Bila je prelepa. Povsem nezakrita se je sončila na trati ob poti. Pritisk se mi je dvignil v višave. Popoln primerek kmečke lastovke!*

1	beseda	Nekega	dne	sem	se	napotil	v	naravo	.	Že	spočetka	me	je											
	lema	nek	dan	biti	se	napotiti	v	narava	že	spočetka	jaz	biti												
	oznaka	Zn-mer	Somer	Gp-spe-n	Zp------k	Ggdd-em	Dt	Sozet	.	L	Rsn	Zop-et--k	Gp-ste-n											
2	beseda	žulil	čevelj	,	a	sem	na to	povsem	pozabil	,	ko	sem	jo	zagledal										
	lema	žuliti	čevelj	a	biti	na ta	povsem	pozabiti	ko	biti	on	zagledati												
	oznaka	Ggnd-em	Somei	,	Vp	Gp-spe-n	Dt	Zk-set	Rsn	Ggdd-em	,	Vd	Gp-spe-n	Zotzet--k	Ggdd-em									
3	beseda	.	Bila	je	prelepa	.	Povsem	nezakrita	se	je	sončila	na	trati											
	lema		biti	biti	prelep		povsem	nezakrit	se	biti	sončiti	na	trata											
	oznaka	.	Gp-d-ez	Gp-ste-n	Ppnzei	.	Rsn	Ppnzei	Zp------k	Gp-ste-n	Ggvd-ez	Dm	Sozem											
4	beseda	ob	poti	.	Pritisk	se	mi	je	dvignil	v	višave	.	Popoln											
	lema	ob	pot		pritisk	se	jaz	biti	dvigniti	v	višava		popoln											
	oznaka	Dm	Sozem	.	Somei	Zp------k	Zop-ed--k	Gp-ste-n	Ggdd-em	Dt	Sozmt	.	Ppnmein											
5	beseda	primerek	kmečke	lastovke	!																			
	lema	primerek	kmečki	lastovka																				
	oznaka	Somei	Ppnzer	Sozer	!																			

TEI-XML format

```
<TEI xmlns="http://www.tei-c.org/ns/1.0">
  <text>
    <body>
      <p>
        <s>
          <w msd="Zn-mer" lemma="nek">Nekega</w>
          <S/>
          <w msd="Somer" lemma="dan">dne</w>
          <S/>
          <w msd="Gp-spe-n" lemma="biti">sem</w>
          <S/>
          <w msd="Zp------k" lemma="se">se</w>
          <S/>
          <w msd="Ggdd-em" lemma="napotiti">napotil</w>
          <S/>
          <w msd="Dt" lemma="v">v</w>
          <S/>
          <w msd="Sozet" lemma="narava">naravo</w>
          <c>.</c>
          <S/>
        </s>
   …

      </p>
    </body>
  </text>
</TEI>
```

MSD tags

- Multext-East specification

dne; tag Somer = Samostalnik, obče ime, moški spol, ednina, rodilnik; lema: dan

P	atribut	vrednost	koda	atribut	vrednost	koda
0	glagol		G	Verb		V
1	vrsta	glavni	g	Type	main	m
		pomožni	p		auxiliary	a
2	vid	dovršni	d	Aspect	perfective	e
		nedovršni	n		imperfective	p
		dvovidski	v		biaspectual	b
3	oblika	nedoločnik	n	VForm	infinitive	n
		namenilnik	m		supine	u
		deležnik	d		participle	p
		sedanjik	s		present	r
		prihodnjik	p		future	f
		pogojnik	g		conditional	c
		velelnik	v		imperative	m
4	oseba	prva	p	Person	first	1
		druga	d		second	2
		tretja	t		third	3
5	število	ednina	e	Number	singular	s
		množina	m		plural	p
		dvojina	d		dual	d
6	spol	moški	m	Gender	masculine	m
		ženski	z		feminine	f
		srednji	s		neuter	n
7	nikalnost	nezanikani	n	Negative	no	n
		zanikani	d		yes	y

POS tagging in English

- http://nlpdotnet.com/Services/Tagger.aspx
- Rainer Maria Rilke, 1903
  in Letters to a Young Poet

...I would like to beg you dear Sir, as well as I can, to have patience with everything unresolved in your heart and to try to love the questions themselves as if they were locked rooms or books written in a very foreign language. Don't search for the answers, which could not be given to you now, because you would not be able to live them. And the point is to live everything. Live the questions now. Perhaps then, someday far in the future, you will gradually, without even noticing it, live your way into the answer.

POS tagger output

I/PRP would/MD like/VB to/TO beg/VB you/PRP dear/JJ Sir/NNP ,/, as/RB well/RB as/IN I/PRP can/MD ,/, to/IN have/VBP patience/NN with/IN everything/NN unresolved/JJ in/IN your/PRP$ heart/NN and/CC to/TO try/VB to/TO love/VB the/DT questions/NNS themselves/PRP as/RB if/IN they/PRP were/VBD locked/VBN rooms/NNS or/CC books/NNS written/VBN in/IN a/DT very/RB foreign/JJ language/NN ./.

Named entity recognition (NER)

- NATO Secretary-General Jens Stoltenberg is expected to travel to Washington, D.C. to meet with U.S. leaders.
- [ORG NATO] Secretary-General [PER Jens Stoltenberg] is expected to travel to [LOC Washington, D.C.] to meet with [LOC U.S.] leaders.
- Named entity linking (NEL) also named entity disambiguation – linking to a unique identifier, e.g. wikification
  jaguar, Paris, London, Dunaj

Basic language resources: corpora

- Statistical natural language processing list of resources http://nlp.stanford.edu/links/statnlp.html
- Opus http://opus.nlpl.eu/ multilingual parallel corpora, e.g., DGT JRC-Acqui 3.0, Documents of the EU in 22 languages
- Slovene language corpora GigaFida, ccGigaFida, KRES, ccKres, GOS, Artur, JANES, KAS, Trendi
- The main Slovene language resources
  - http://www.clarin.si
  - https://github.com/clarinsi
  - http://www.cjvt.si/
  - https://www.slovenscina.eu/
- WordNet, Open English WordNet, Open Multilingual WordNet, SloWNet, sentiWordNet, …
- Thesaurus https://viri.cjvt.si/sopomenke/slv/
- LLMs: on HuggingFace cjvt, e.g., SloBERTa and GaMS

WordNet is a database composed of synsets:

synonyms,
hypernyms
hyponyms,
meronyms,
holonyms,
etc.

Word to search for: mercy [Search WordNet]

Display Options: (Select option to change) [Change]

Key: "S:" = Show Synset (semantic) relations, "W:" = Show Word (lexical) relations

Display options for sense: (gloss) "an example sentence"

Noun

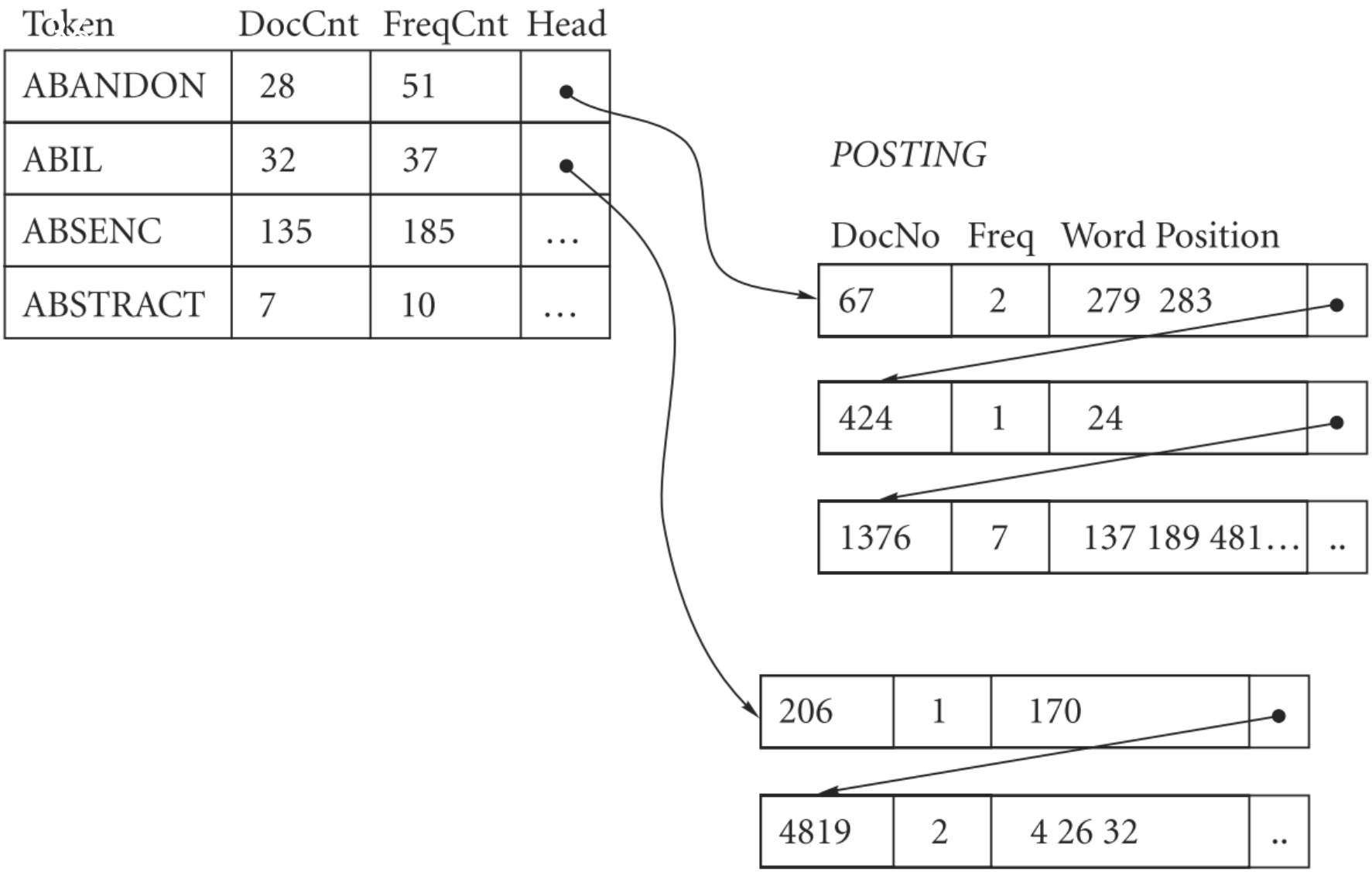
- S: (n) clemency, mercifulness, **mercy** (leniency and compassion shown toward offenders by a person or agency charged with administering justice) *"he threw himself on the mercy of the court"*
- S: (n) mercifulness, **mercy** (a disposition to be kind and forgiving) *"in those days a wife had to depend on the mercifulness of her husband"*
- S: (n) mercifulness, **mercy** (the feeling that motivates compassion)
  - *direct hyponym* / *full hyponym*
    - S: (n) forgiveness (compassionate feelings that support a willingness to forgive)
  - *direct hypernym* / *inherited hypernym* / *sister term*
    - S: (n) compassion, compassionateness (a deep awareness of and sympathy for another's suffering)
  - ***derivationally related form***
    - W: (adj) merciful [Related to: mercifulness] (showing or giving mercy) *"sought merciful treatment for the captives"; "a merciful god"*
- S: (n) **mercy** (something for which to be thankful) *"it was a mercy we got out alive"*
- S: (n) **mercy** (alleviation of distress; showing great kindness toward the distressed) *"distributing food and clothing to the flood victims was an act of mercy"*

Document retrieval

- Historical: keywords
- Now: whole text search
- Organize a database, indexing, search algorithms
- input: a query (of questionable quality, ambiguity, answer quality)

Document indexing

- Collect all words from all documents, use lemmatization
- The inverted file data structure
- For each word keep
  - Number of appearing documents
  - Overall number of appearances
  - For each document
    - Number of appearances
    - Location


Token
DocCnt
FreqCnt
Head
ABANDON
28
51
ABIL
32
37
ABSENC
135
185
…
ABSTRACT
7
10
…
POSTING
DocNo
Freq
Word Position
67
2
279 283
424
1
24
1376
7
137 189 481…
..
206
1
170
4819
2
4 26 32
..

Full text search engine

- Most popular: Apache Lucene/Solr
- full-text search, hit highlighting, real-time indexing, dynamic clustering, database integration, NoSQL features, rich document (e.g., Word, PDF) handling.
- distributed search and index replication, scalability and fault tolerance.

Search with logical operators

- AND, OR, NOT
- jaguar AND car
  jaguar NOT animal
- Some system support neighborhood search (e.g., NEAR) and stemming (!)
  paris! NEAR(3) fr!
  president NEAR(10) bush
- libraries, concordancers
- E.g-, for Slovene: https://viri.cjvt.si/gigafida/

Logical operator search is outdated

- Large number of results
- Large specialized incomprehensible queries
- Synonyms
- Sorting of results
- No partial matching
- No weighting of query terms

Ranking based search

- Web search
- Less frequent terms are more informative
- NL input - stop words, lemmatization
- Vector based representation of documents and queries (bag-of-words or dense embeddings)

Sparse vector representation: bag-of-words

- *An elephant is a mammal. Mammals are animals. Humans are mammals, too. Elephants and humans live in Africa.*

Africa	animal	be	elephant	human	in	live	mammal	too
1	1	3	2	2	1	1	3	1

9 dimensional vector (1,1,3,2,2,1,1,3,1)

In reality this is sparse vector of dimension $|V|$ (vocabulary size in order of 10,000 dimensions)

Similarity between documents and queries in vector space.

Vectors and documents

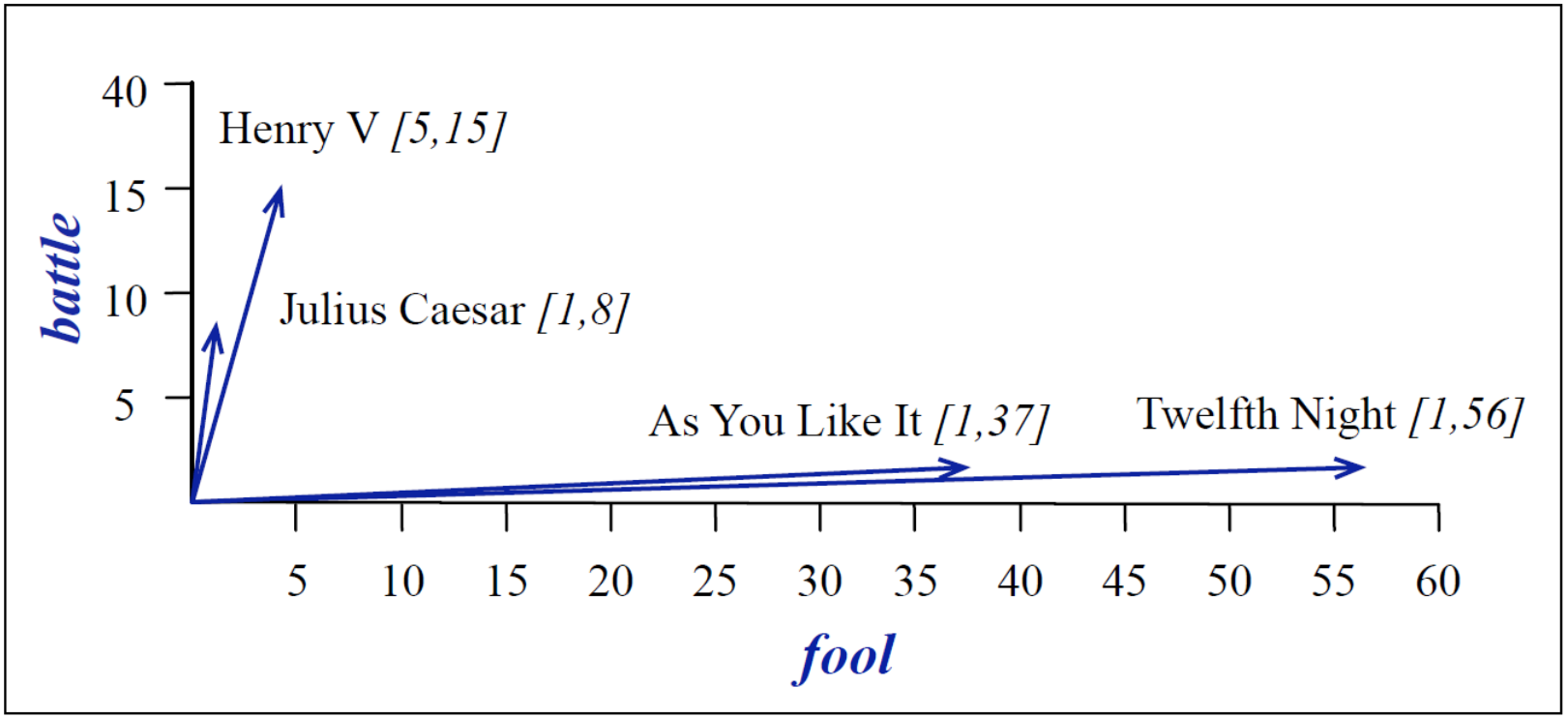
- a word occurs in several documents
- both words and documents are vectors
- an example: Shakespeare

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	1	1	8	15
soldier	2	2	12	36
fool	37	58	1	5
clown	5	117	0	0

- term-document matrix, dimension |V| x |D|
- a sparse matrix
- word embedding

Vector based similarity

- e.g., in two dimensional space



- the difference between dramas and comedies

Document similarity

- Assume orthogonal dimensions
- Cosine similarity
- Dot (scalar) product of vectors

$$\cos(\Theta) = \frac{A \cdot B}{|A||B|}$$

Importance of words

- Frequencies of words in particular document and overall
- inverse document frequency idf
  - N = number of documents in collection
  - n_b = number of documents with word b

$$idf_b = \log(\frac{N}{n_b})$$

Weighting dimensions (words)

- Weight of word b in document d

$$w_{b,d} = tf_{b,d} \times idf_{b,d}$$

$tf_{b,d}$ = frequency of term b in document d

- called TF-IDF weighting (an improvement over bag-of-words)

Weighted similarity

- Between query and document

$$sim(q,d) = \frac{\sum_b w_{b,d} \cdot w_{b,q}}{\sqrt{\sum_b w_{b,d}^2} \cdot \sqrt{\sum_b w_{b,q}^2}}$$

- Ranking by the decreasing similarity

Performance measures for search

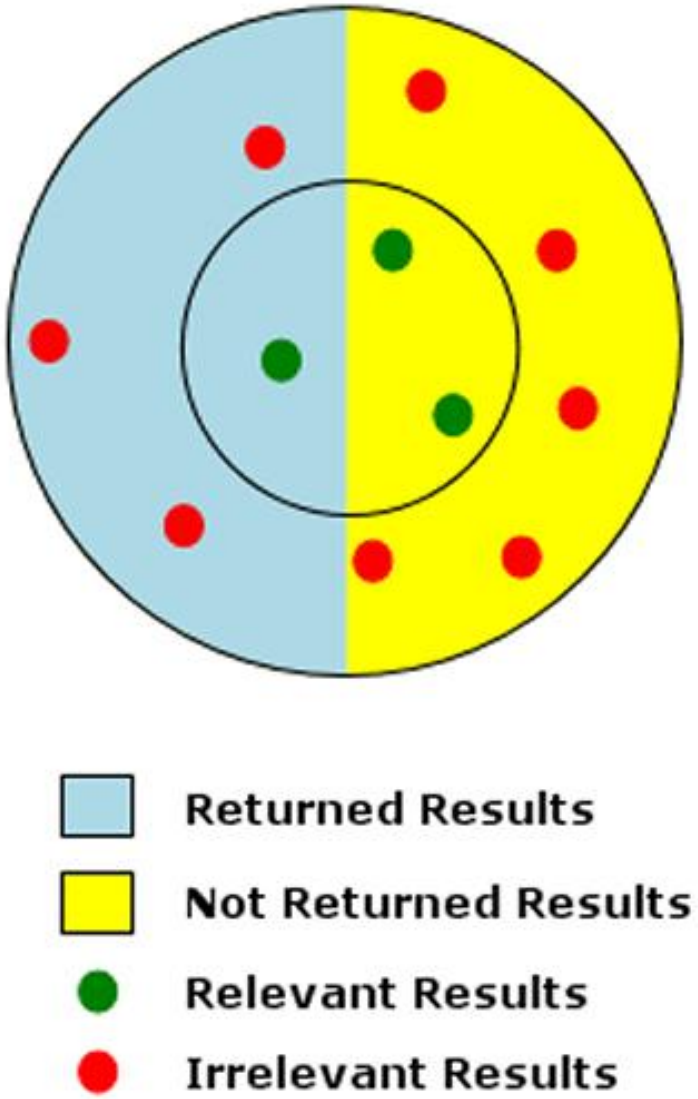
- Statistical measures
- Subjective measures
- Precision, recall
- A contingency table analysis of precision and recall

	Relevant	Non-relevant	
Retrieved	a	b	$a + b = m$
Not retrieved	c	d	$c + d = N - m$
	$a + c = n$	$b + d = N - n$	$a + b + c + d = N$

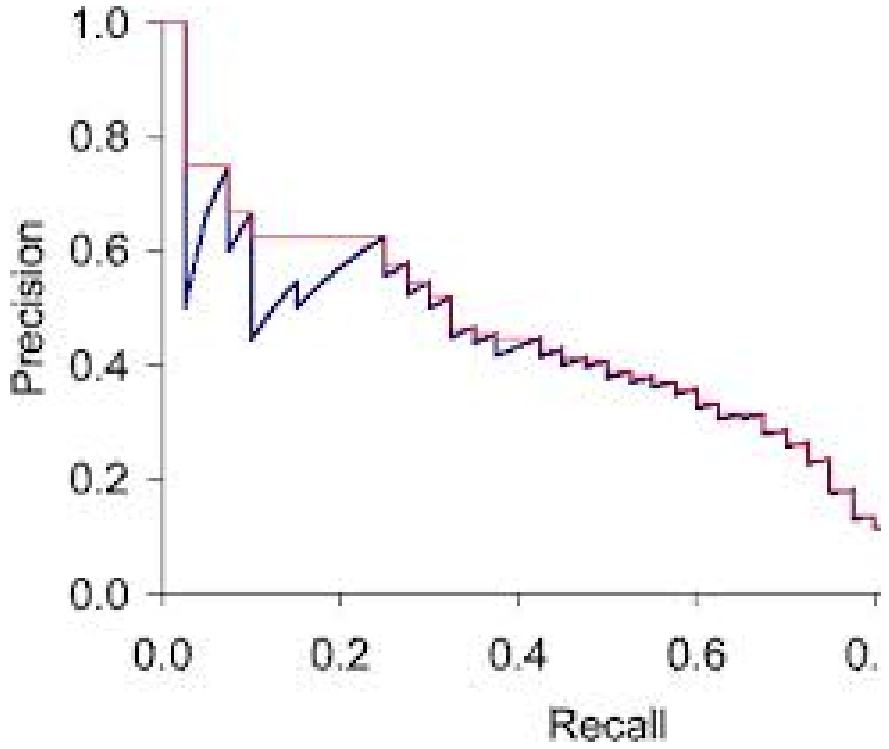
Precision and recall

- N = number of documents in collection
- n = number of important documents for given query q
- Search returns m documents including a relevant ones
- Precision $P = a/m$
  proportion of relevant document in the obtained ones
- Recall $R = a/n$
  proportion of obtained relevant documents
- Precision recall graphs

An example: low precision, low recall



Precision-recall graphs



F-measure

- combine both P and R

$$F_\beta = \frac{(1+\beta^2)\cdot P \cdot R}{\beta^2 P + R} \text{ for } \beta > 0$$

$$F_1 = \frac{2 \cdot P \cdot R}{P + R}$$

- Weighted precision and recall
- β=1 weighted harmonic mean
- Also used β=2 or β= 0.5

Ranking measures

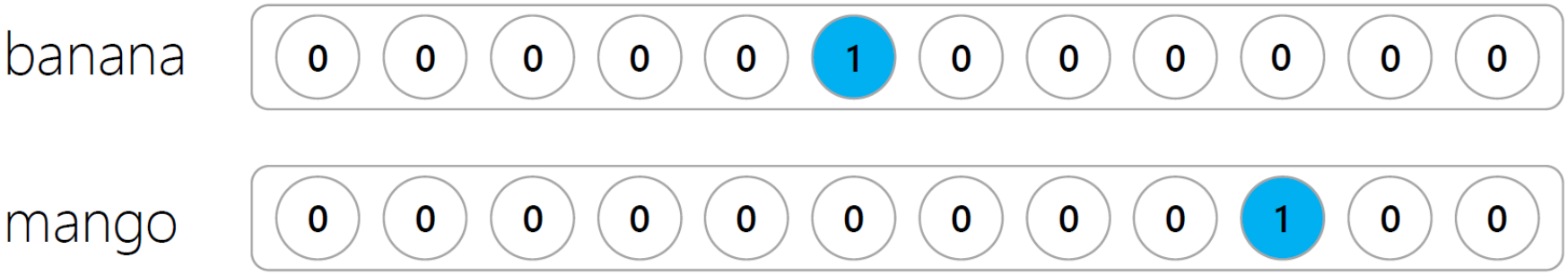
- precision@k
  - proportion of relevant document in the first k obtained ones
- recall@k
  - proportion of relevant documents in the k obtained among all relevant
- F_1@k
- mean reciprocal rank

$$MRR = \frac{1}{Q}\sum_{i=1}^{Q}\frac{1}{\mathrm{rank}_i}$$

  - over Q queries,
  - considers only the rank of the best answer

Why dense textual embeddings?

- Best machine learning models for text, i.e. deep neural networks, require numerical input.
- Simple representations like 1-hot-encoding and bag-of-words do not preserve semantic similarity.
- We need dense vector represenation for text elements.



Dense vector embeddings

- advantages compared to sparse embeddings:
  - less dimensions, less space
  - easier input for ML methods
  - potential generalization and noise reduction
  - potentially captures synonymy, e.g., road and highway are different dimensions in BOW
- the most popular approaches
  - matrix based transformations to reduce dimensionality (SVD in LSA)
  - neural embeddings (word2vec, Glove)
  - explicit contextual neural embeddings (ELMo, BERT)
  - only use an implicit representation in LLM

Meaning focused on similarity

- Each word is a vector
- Similar words are "nearby in space"



Distributional semantics



"You shall know a word
by the company it keeps"

Firth, J. R. (1957). A synopsis of linguistic theory 1930–1955. In Studies in Linguistic Analysis, p. 11. Blackwell, Oxford.



"The meaning of a word is its use in the language"

Ludwig Wittgenstein, PI #43

Neural embeddings

- neural network is trained to predict the context of words (input: word, output: context of neighboring words)

word2vec method

- Train a classifier on a binary **prediction** task:
  Is word *w* likely to show up near a given word, e.g., "*apricot*"?
- We don't actually care about this task
- But we'll take the learned classifier weights as the word embeddings

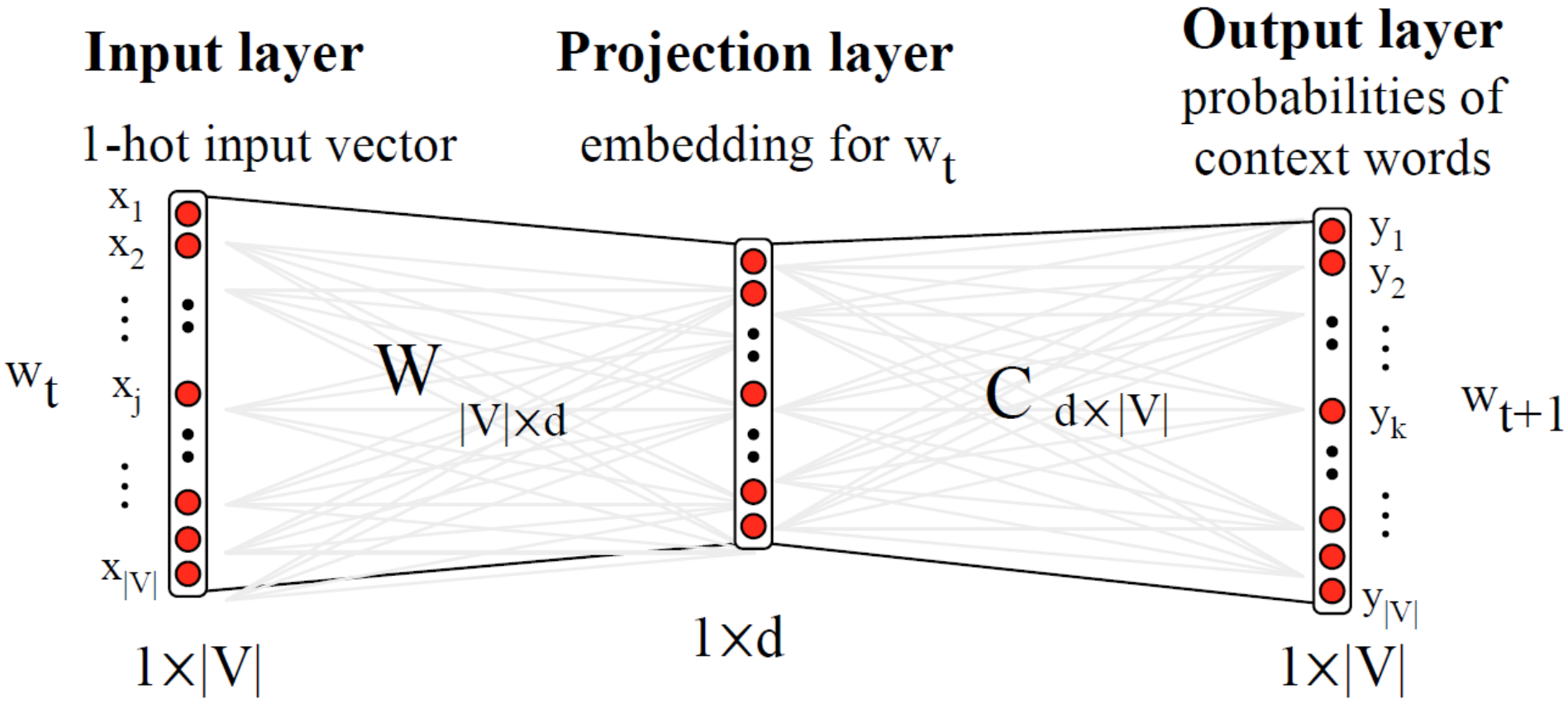
- Words near apricot acts as 'correct answers' to the question "Is word w likely to show up near apricot?"
- No need for hand-labeled supervision

word2vec (skip-gram) training data

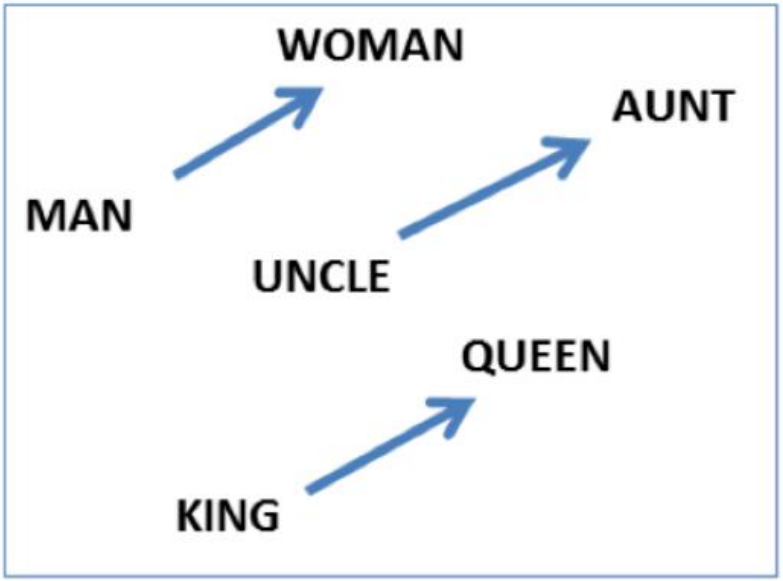
- Training sentence:
- ... lemon, a tablespoon of **apricot** jam a pinch ...
- c1 c2 target c3 c4

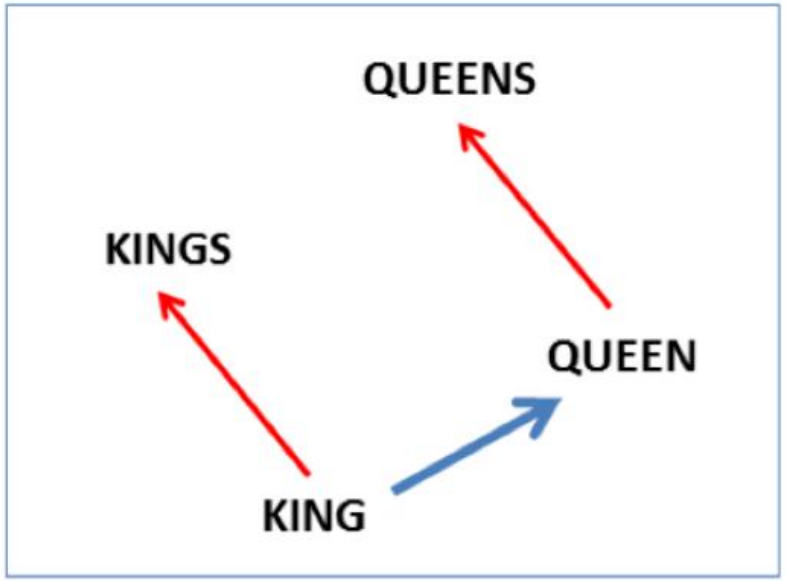
- Asssume context words are those in +/- 2 word window
- Produce the following input-outputs for positive instances: (apricot, tablespoon), (apricot, of), (apricot, jam), (apricot, a)
- Get negative training examples randomly
- Train a neural network to predict the probability of a co-occurring word

Neural network based embedding



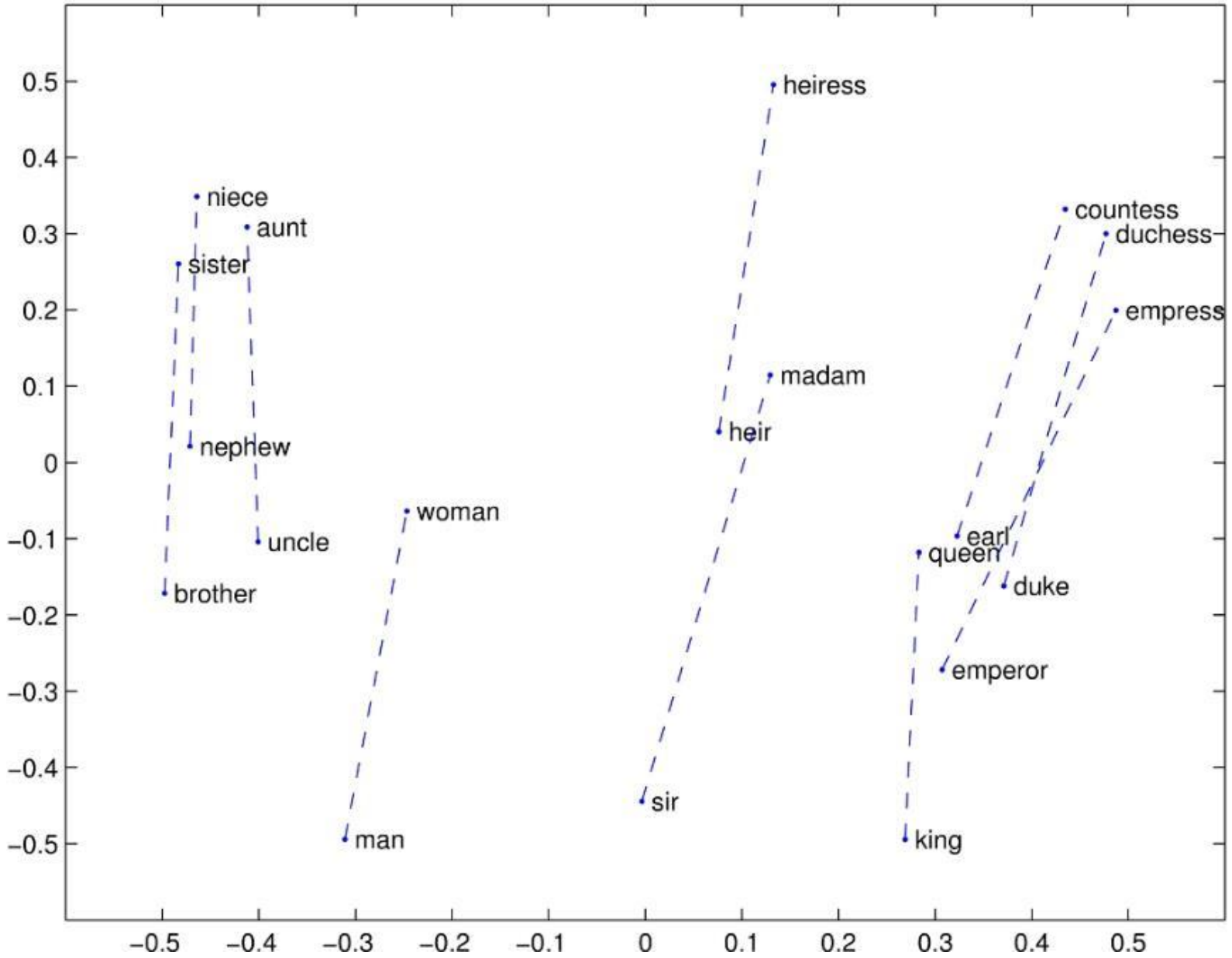
Relational similarity

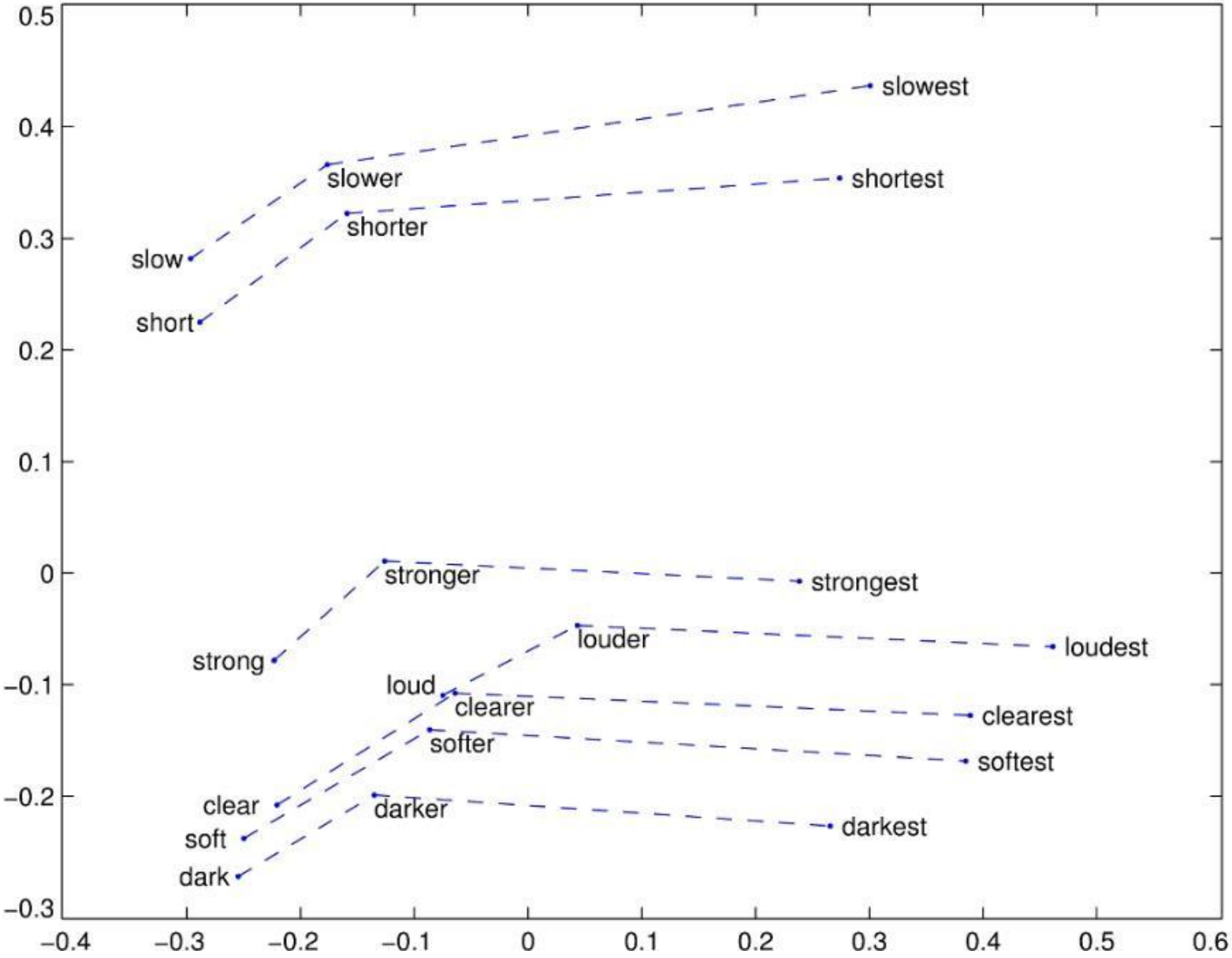




vector(‘*king*’) - vector(‘*man*’) + vector(‘*woman*’) ≈ vector(‘queen’)

vector(‘*Paris*’) - vector(‘*France*’) + vector(‘*Italy*’) ≈ vector(‘Rome’)

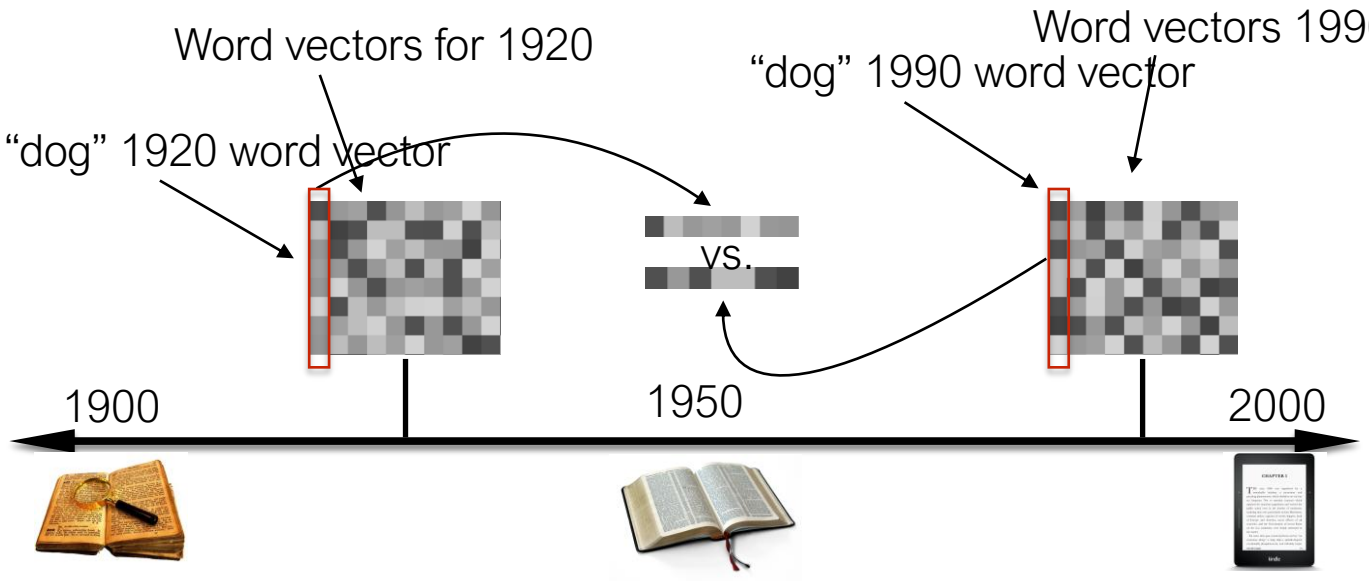

0.5
0.4
0.3
0.2
0.1
0
−0.1
−0.2
−0.3
−0.4
−0.5
−0.5
−0.4
−0.3
−0.2
−0.1
0
0.1
0.2
0.3
0.4
0.5
niece
aunt
sister
nephew
uncle
brother
woman
man
heiress
madam
heir
sir
countess
duchess
empress
earl
queen
duke
emperor
king


slowest
slower
shortest
shorter
slow
short
stronger
strongest
louder
loudest
strong
loud
clearer
clearest
softer
softest
clear
darker
darkest
soft
dark
0.5
0.4
0.3
0.2
0.1
0
−0.1
−0.2
−0.3
−0.4
−0.3
−0.2
−0.1
0
0.1
0.2
0.3
0.4
0.5
0.6

Embeddings can help study word history

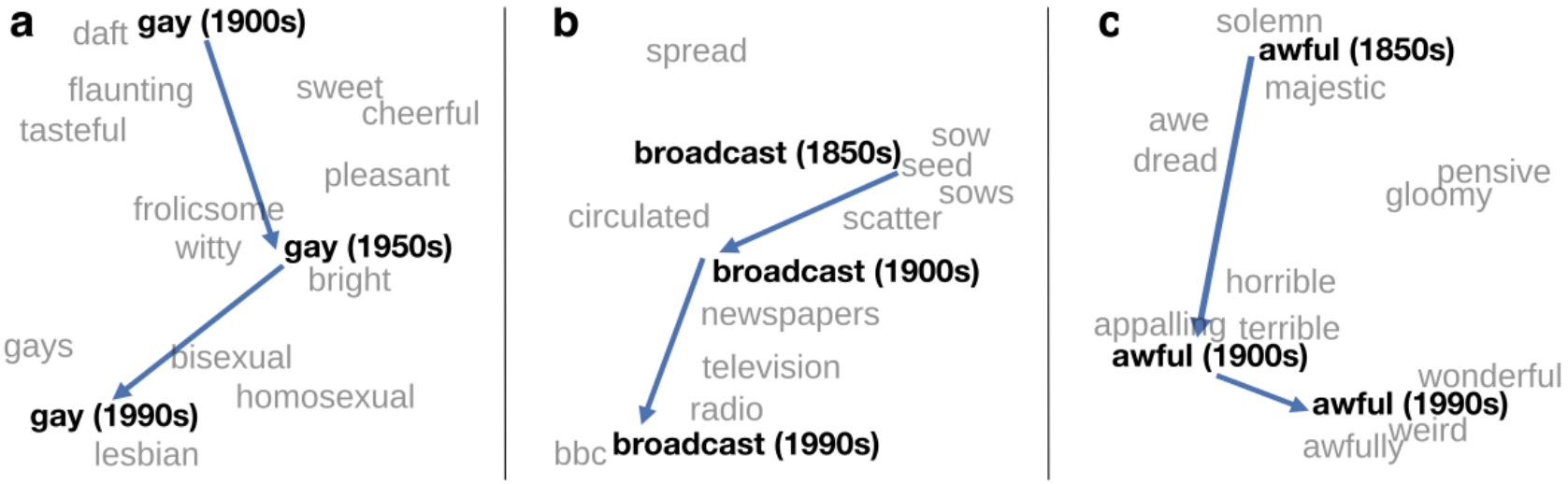
- Train embeddings on old books to study changes in word meaning!!

Diachronic word embeddings for studying language change



Visualizing changes

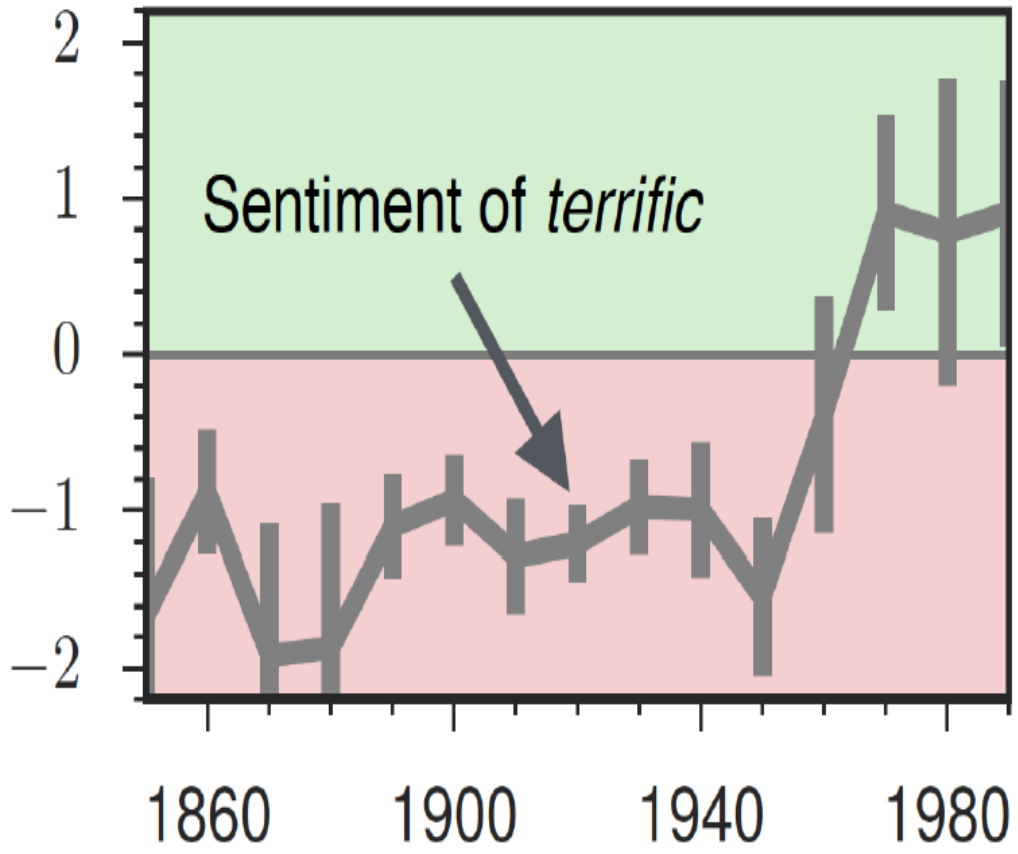
Project 300 dimensions down into 2



~30 million books, 1850-1990, Google Books data

The evolution of sentiment words

Negative words change faster than positive words



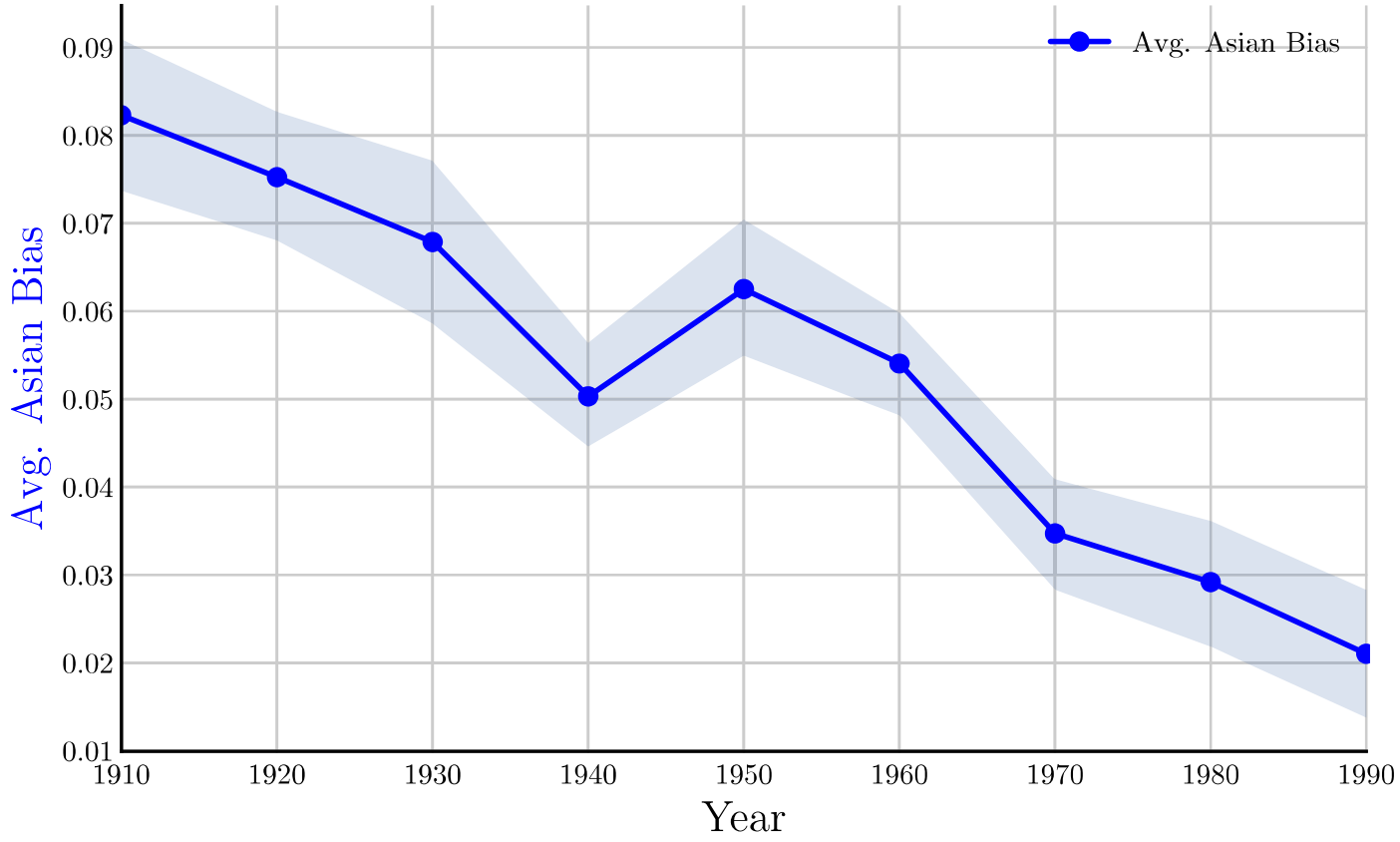
Embeddings reflect cultural bias

- Ask “Paris : France :: Tokyo : x”
  - x = Japan
- Ask “father : doctor :: mother : x”
  - x = nurse
- Ask “man : computer programmer :: woman : x”
  - x = homemaker

Bolukbasi, Tolga, Kai-Wei Chang, James Y. Zou, Venkatesh Saligrama, and Adam T. Kalai. "Man is to computer programmer as woman is to homemaker? debiasing word embeddings." In *Advances in Neural Information Processing Systems*, pp. 4349-4357. 2016.

Change in linguistic framing 1910-1990

Change in association of Chinese names with adjectives framed as "othering" (*barbaric*, *monstrous*, *bizarre*)



Garg, Nikhil, Schiebinger, Londa, Jurafsky, Dan, and Zou, James (2018). Word embeddings quantify 100 years of gender and ethnic stereotypes. *Proceedings of the National Academy of Sciences*, *115*(16), E3635–E3644

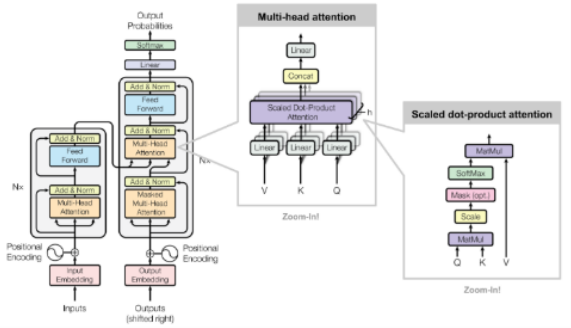
Contextual embeddings / Large language models

- word2vec produces the same vector for a word like bank irrespective of its meaning and context
- recent embeddings take the context into account
- already established as a standard
- e.g., BERT for contextual word embeddings

University of Ljubljana
Faculty of Computer and Information Science

Large language models (LLMs)

- pretrained neural large language models
- trained on large text corpora to capture relations in language
- finetuned to specific tasks
- many are publicly available
- on HuggingFace

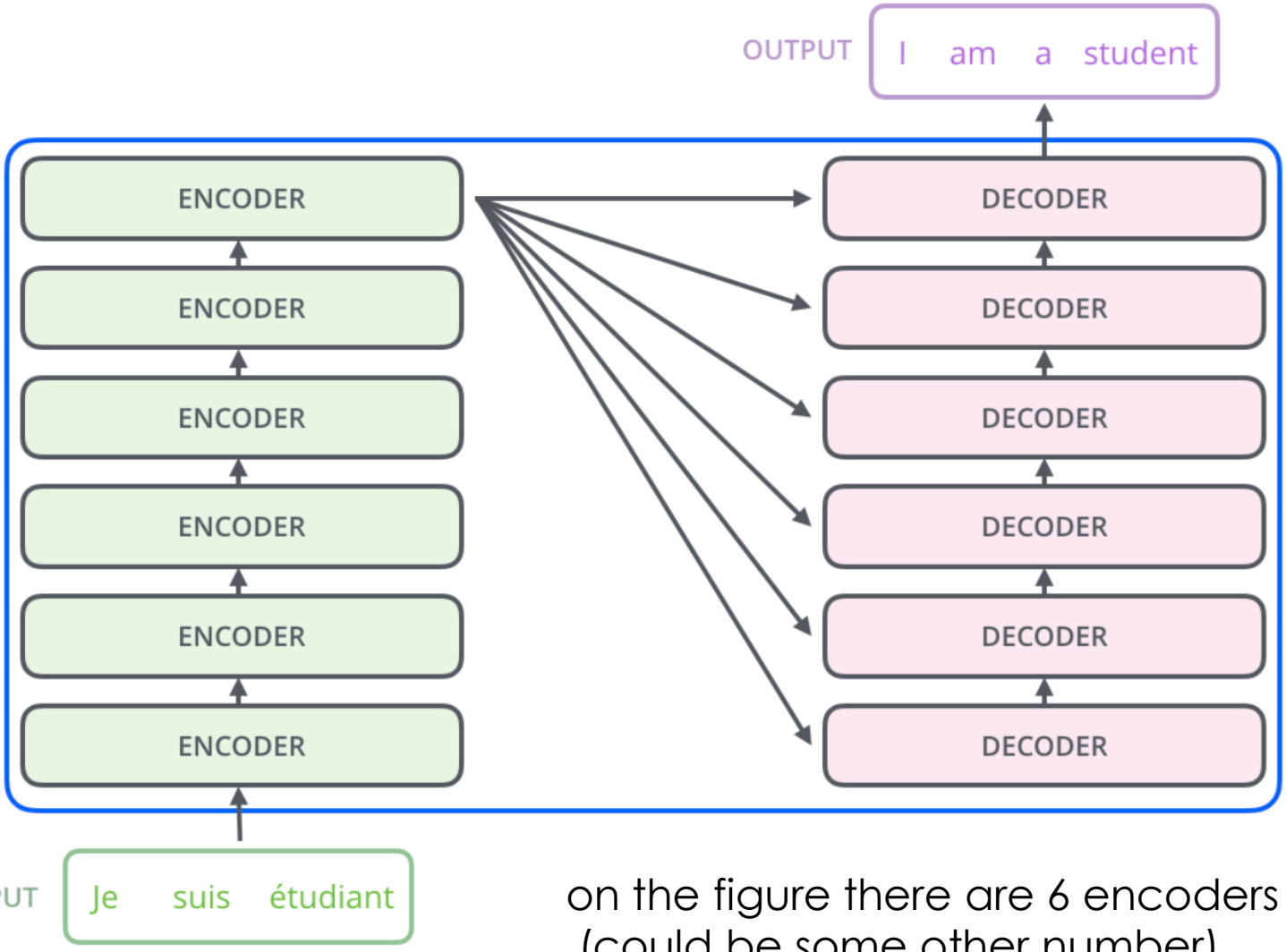


Transformer architecture of NNs

- currently the most successful DNN
- non-recurrent
- architecturally it is an encoder-decoder model
- fixed input length
- can be parallelized
- adapted for GPU (TPU) processing
- based on extreme use of attention

- https://github.com/dair-ai/Transformers-Recipe

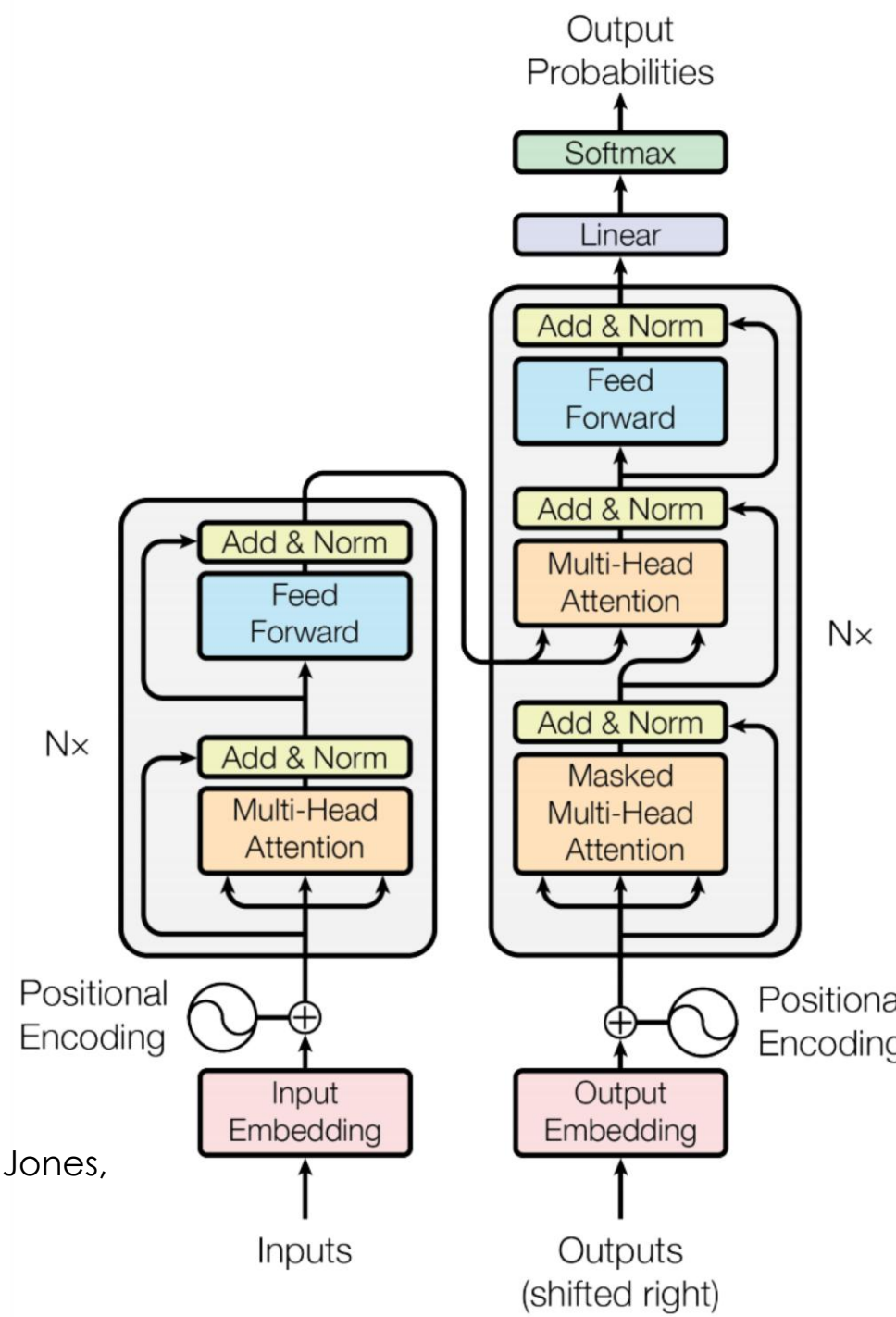
Transformer is an encoder-decoder model



on the figure there are 6 encoders and 6 decoders
(could be some other number)

Transformer overview

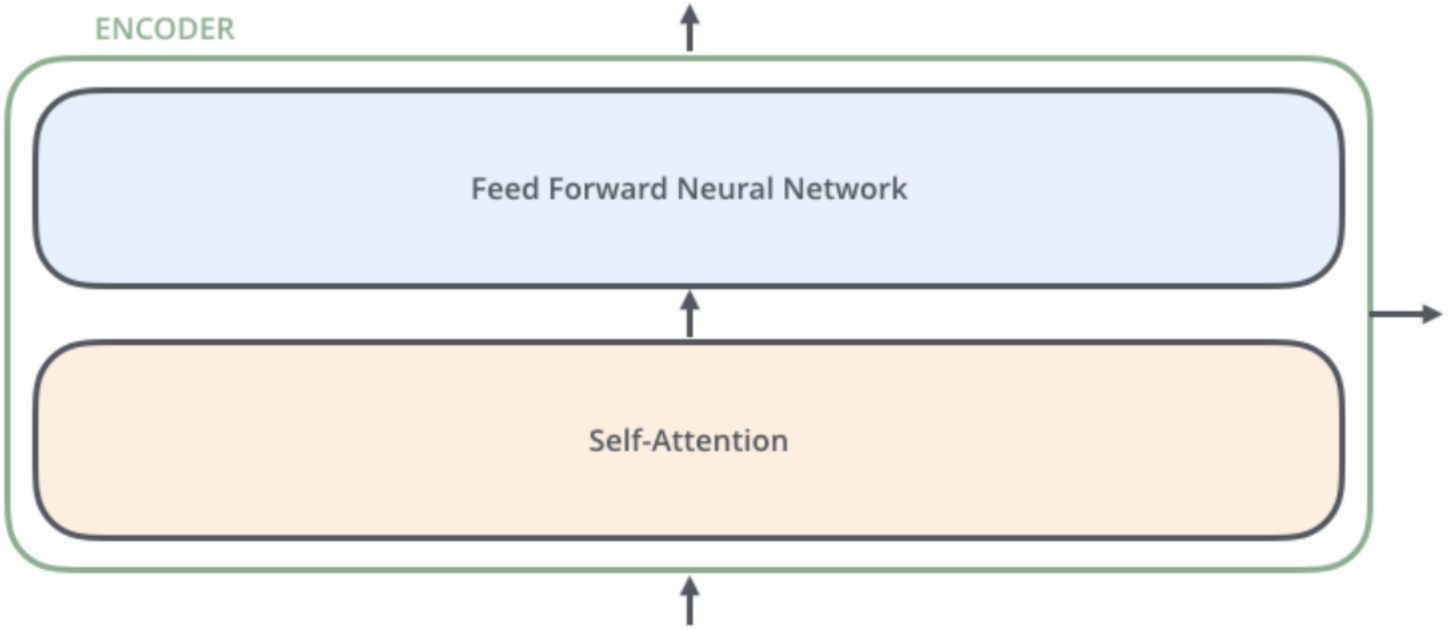
- Initial task: machine translation with parallel corpus
- Predict each translated word
- Final cost/loss/error function is standard cross-entropy error on top of a softmax classifier



Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, Ł. and Polosukhin, I., 2017. Attention is all you need. In *Advances in neural information processing systems* (pp. 5998-6008).

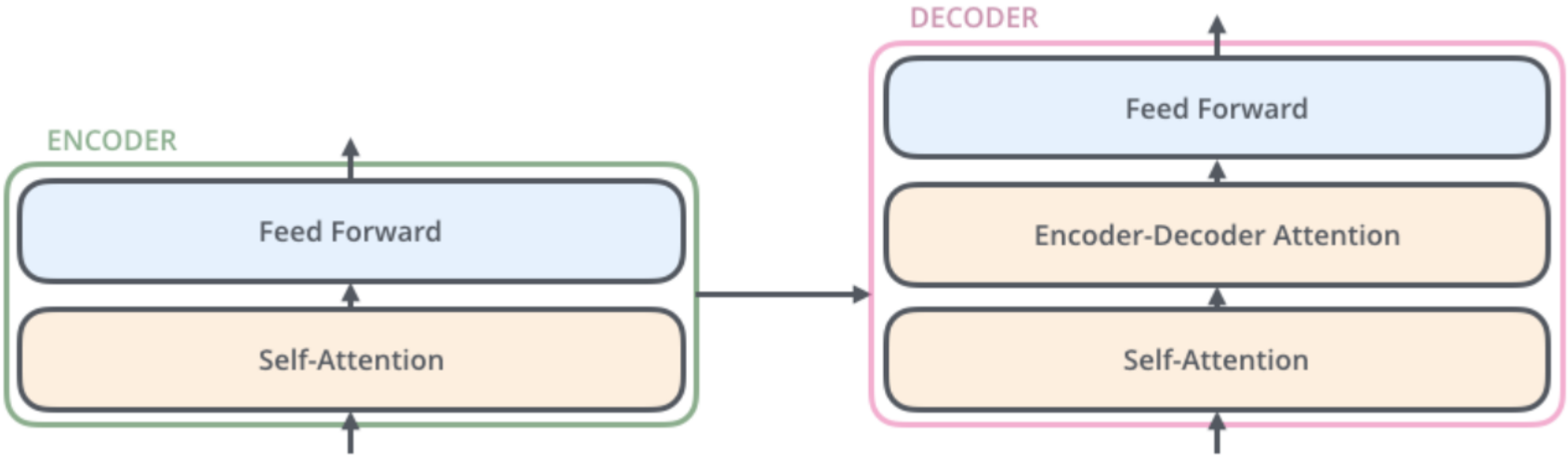
Transformer: encoder

- Two layers
- No weight sharing between different encoders
- Self-attention helps to focus on relevant part of input
- The illustration shows one encoder block

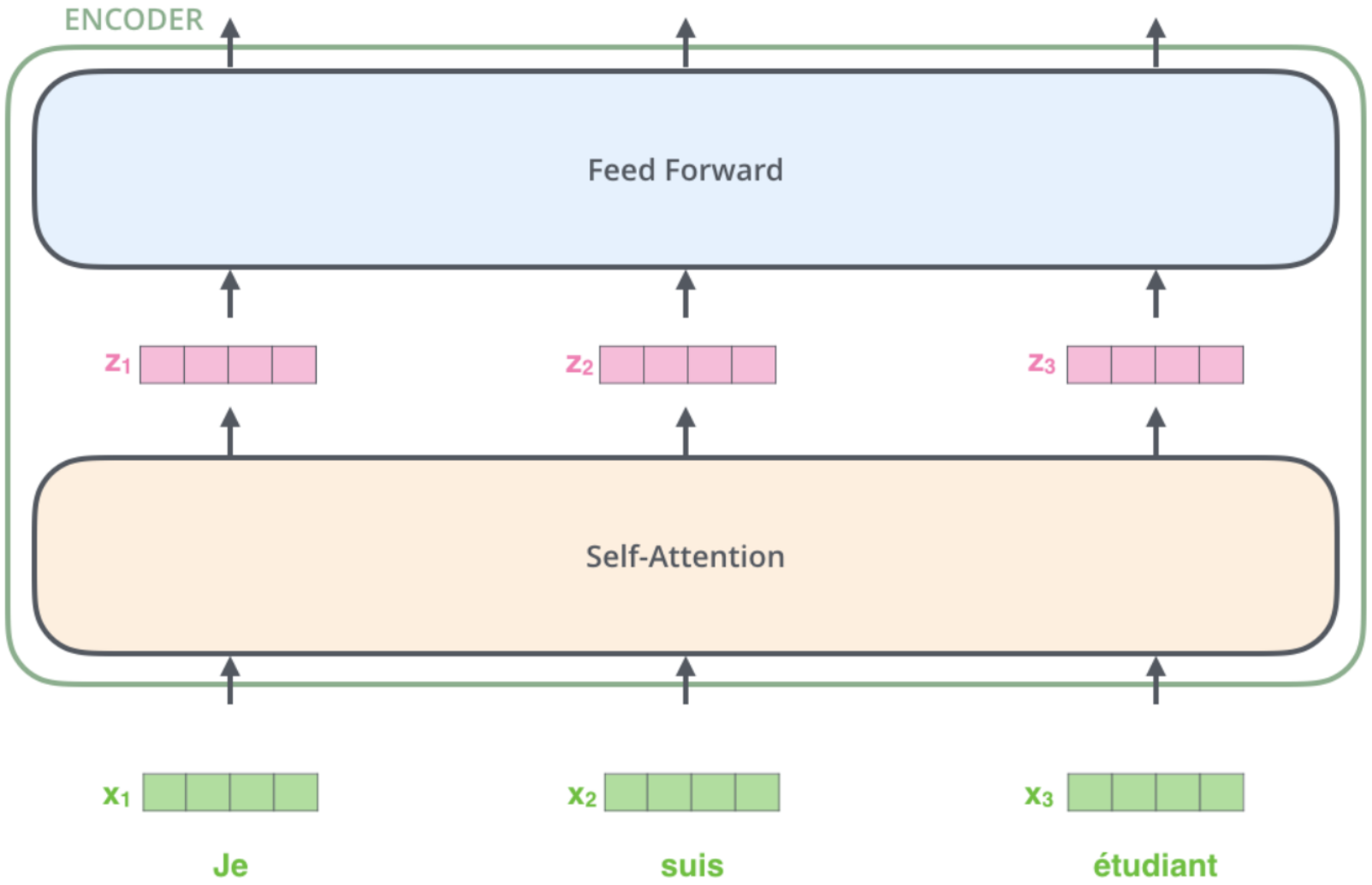


Transformer: decoder

- The same as encoder but with an additional attention layer in between, receiving input fom encoder
- Illustration shows one encoder and one decoder block (usually there are many more encoder and decoder blocks)



Start with embeddings

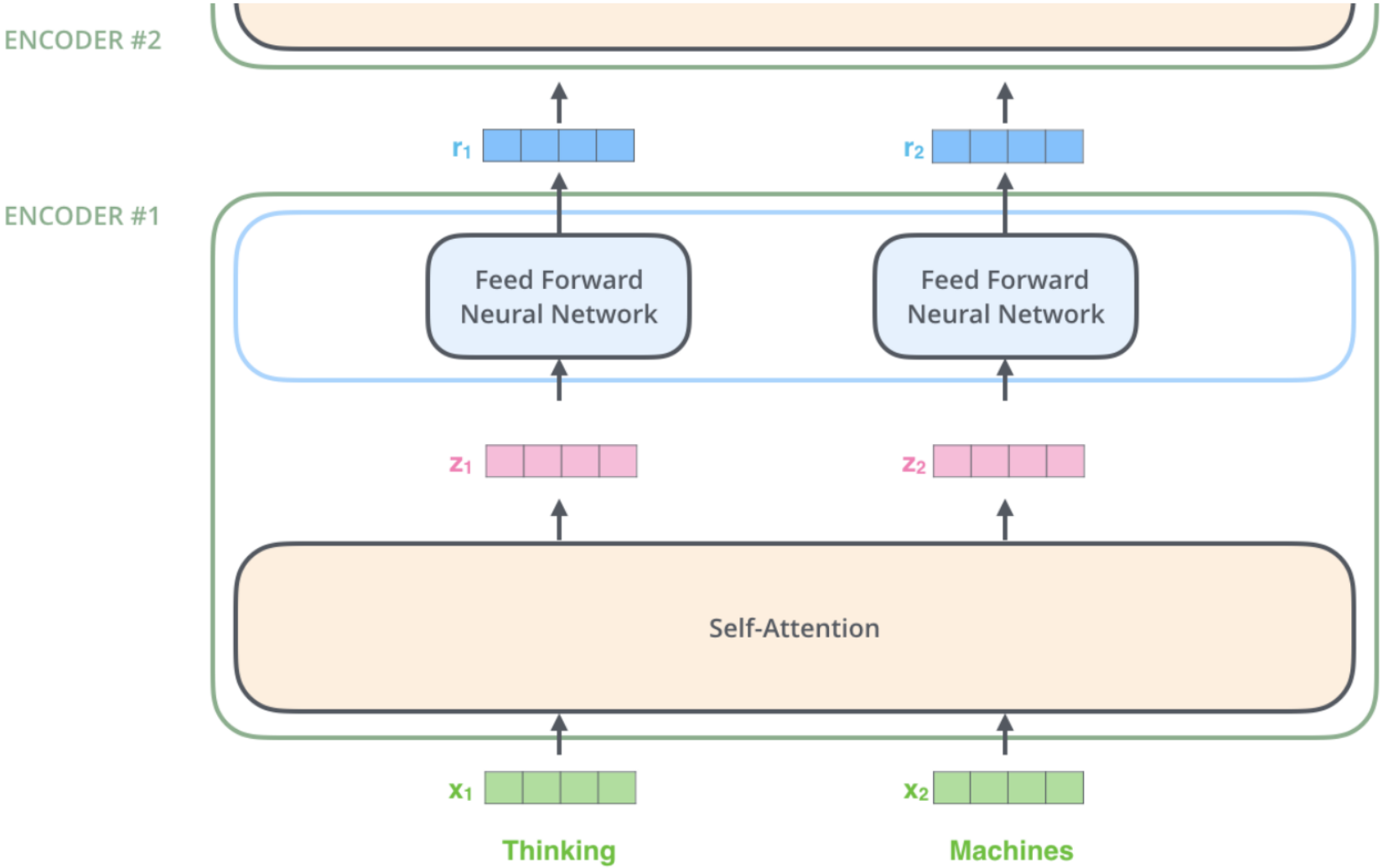


Input to transformer

- embeddings, e.g., 512 dimensional vectors (special, we will discuss that later)
- fixed length, e.g., 128 tokens (in modern architectures at least 8192)
- dependencies between inputs are only in the self-attention layer, no dependencies in feed-forward layer – good for parallelization

- Let us first present the working of the transformer with an illustration of the prediction

Encoding


ENCODER #2
r1
r2
ENCODER #1
Feed Forward Neural Network
Feed Forward Neural Network
z1
z2
Self-Attention
x1
x2
Thinking
Machines

Self-attention

- As the model processes each word (each position in the input sequence), self-attention allows it to look at other positions in the input sequence for clues that can help lead to a better encoding for this word

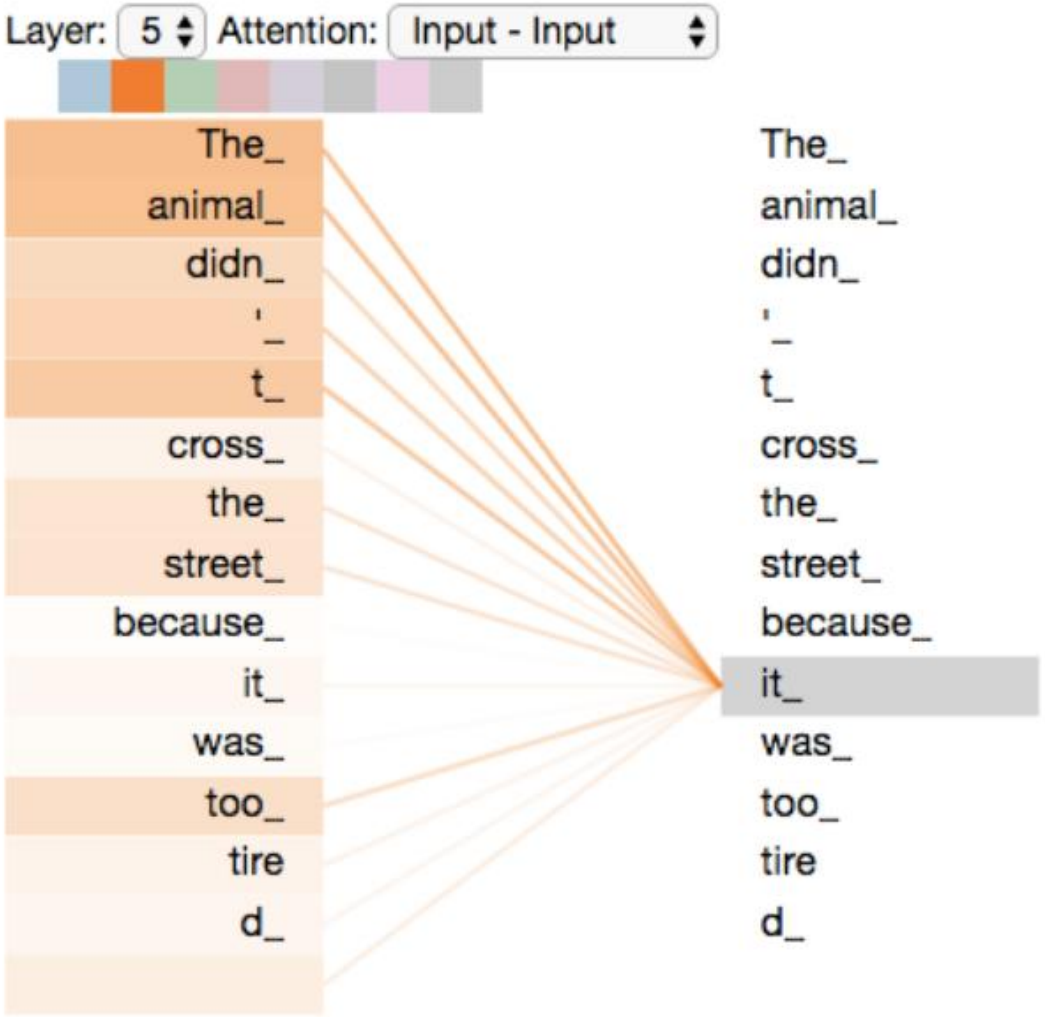
  ”The animal didn't cross the street because it was too tired”

- What does “it” in this sentence refer to? Is it referring to the street or to the animal? It’s a simple question to a human, but not as simple to an algorithm.
- *”The animal didn't cross the street because it was too wide”*

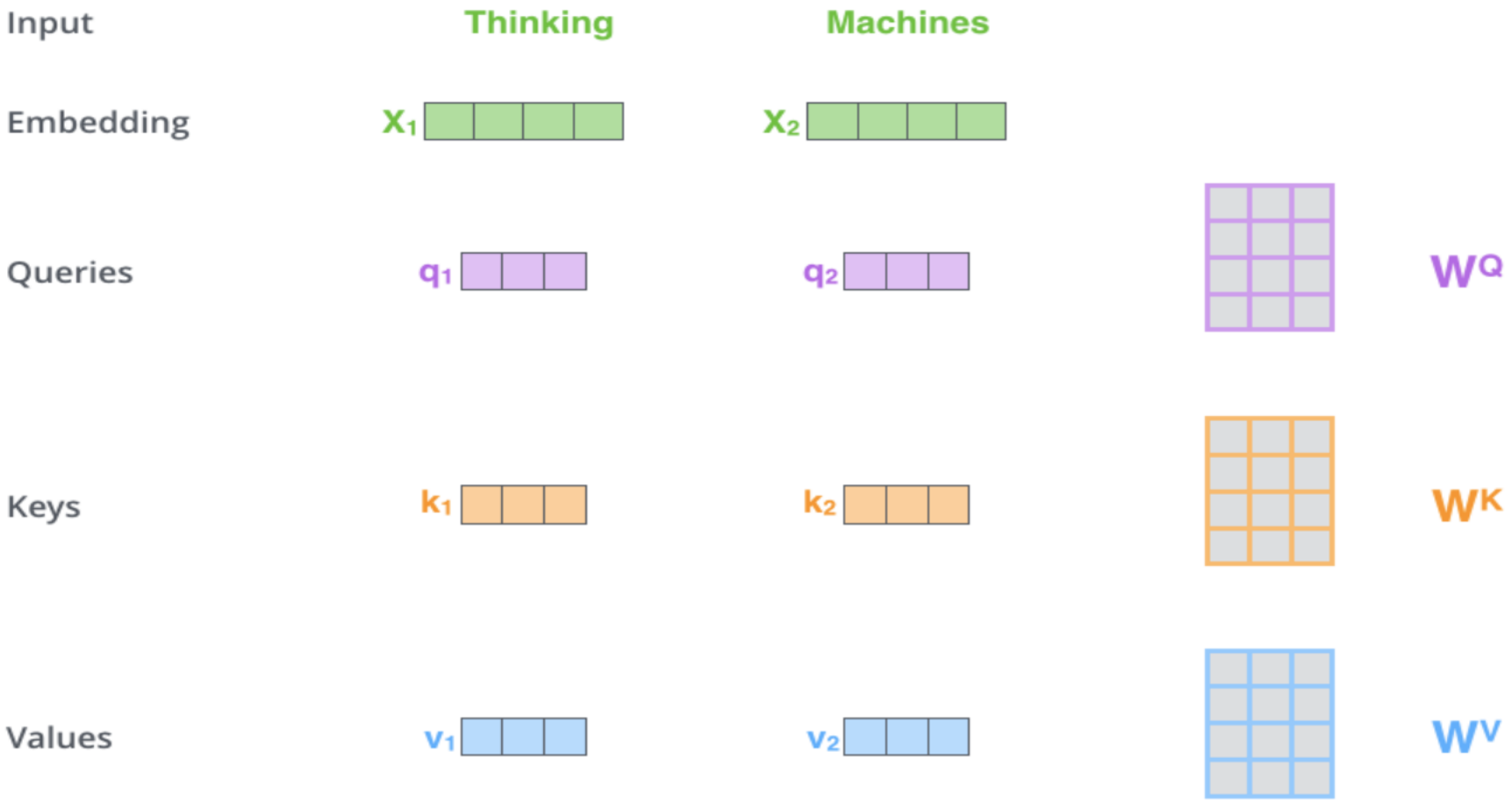
- When the model is processing the word “it”, self-attention allows it to associate “it” with “animal”.

Illustrating self-attention

- As we are encoding the word "it" in encoder #5 (the top encoder in the stack), part of the attention mechanism was focusing on "The Animal", and baked a part of its representation into the encoding of "it".

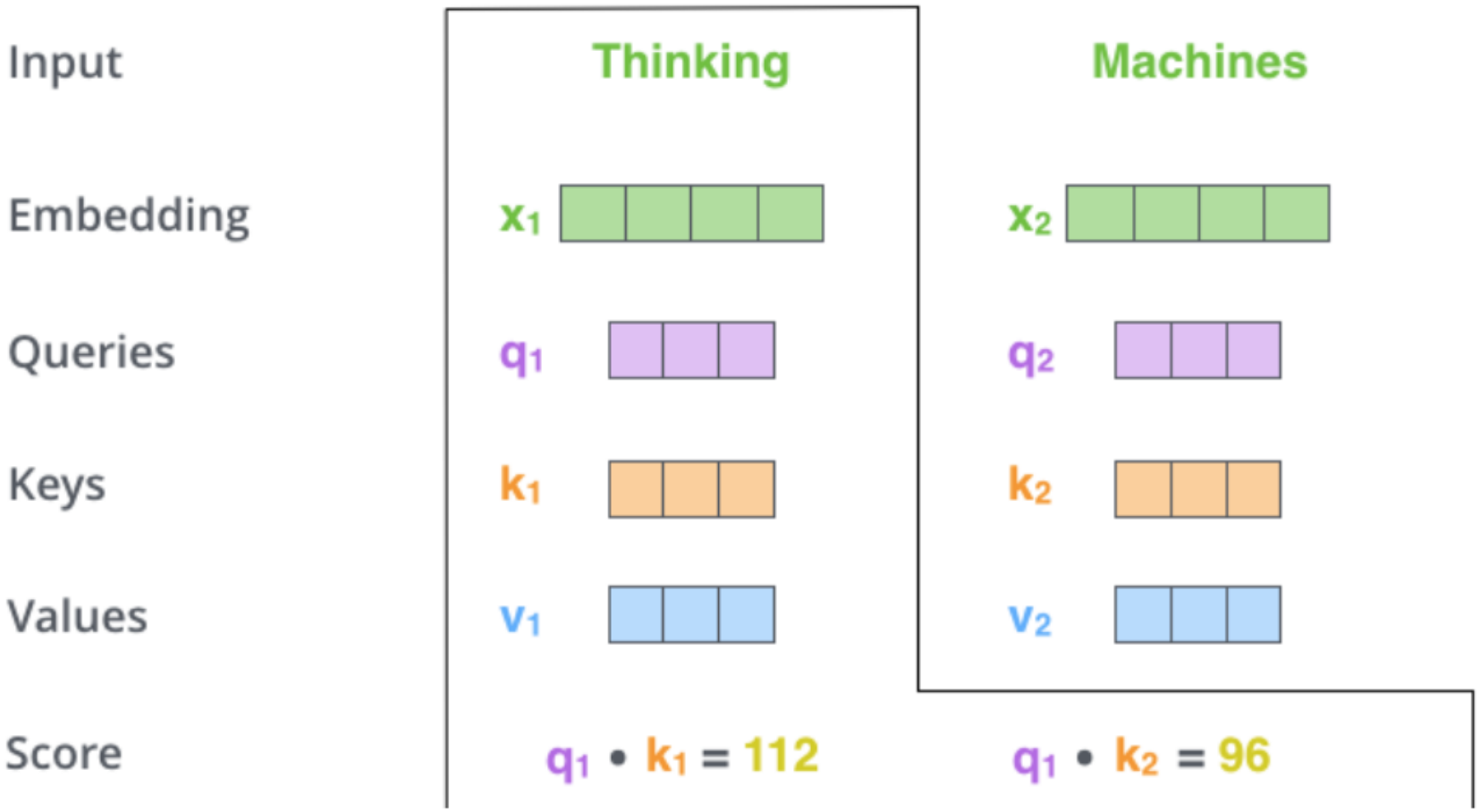


Self-attention details 1/4

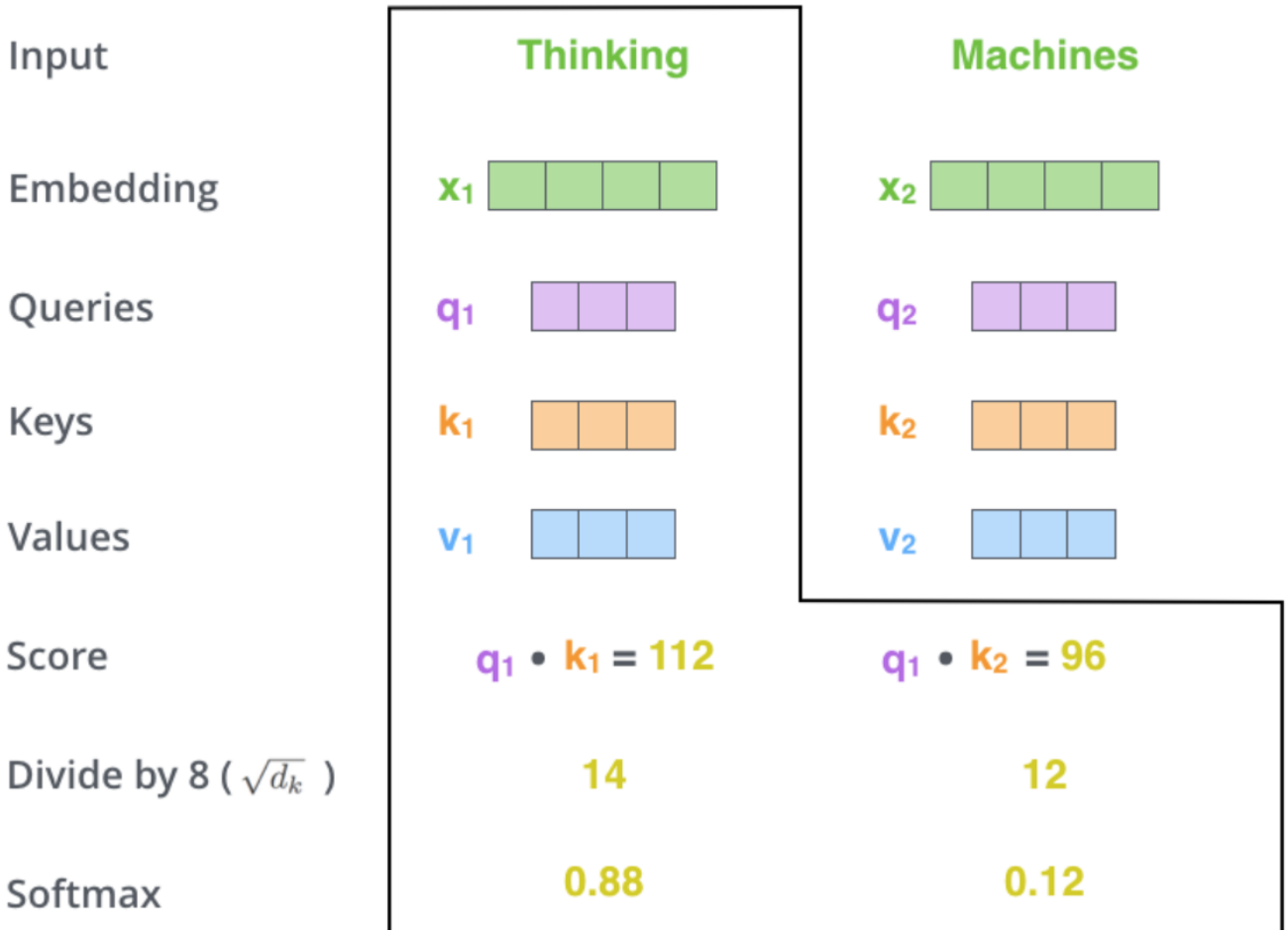


Multiplying x_1 by the W^Q weight matrix produces q_1, the "query" vector associated with that word. We end up creating a "query", a "key", and a "value" projection of each word in the input sentence.

Details 2/4: scoring



Details 3/4: normalization of scores



Details 4/4: self-attention output

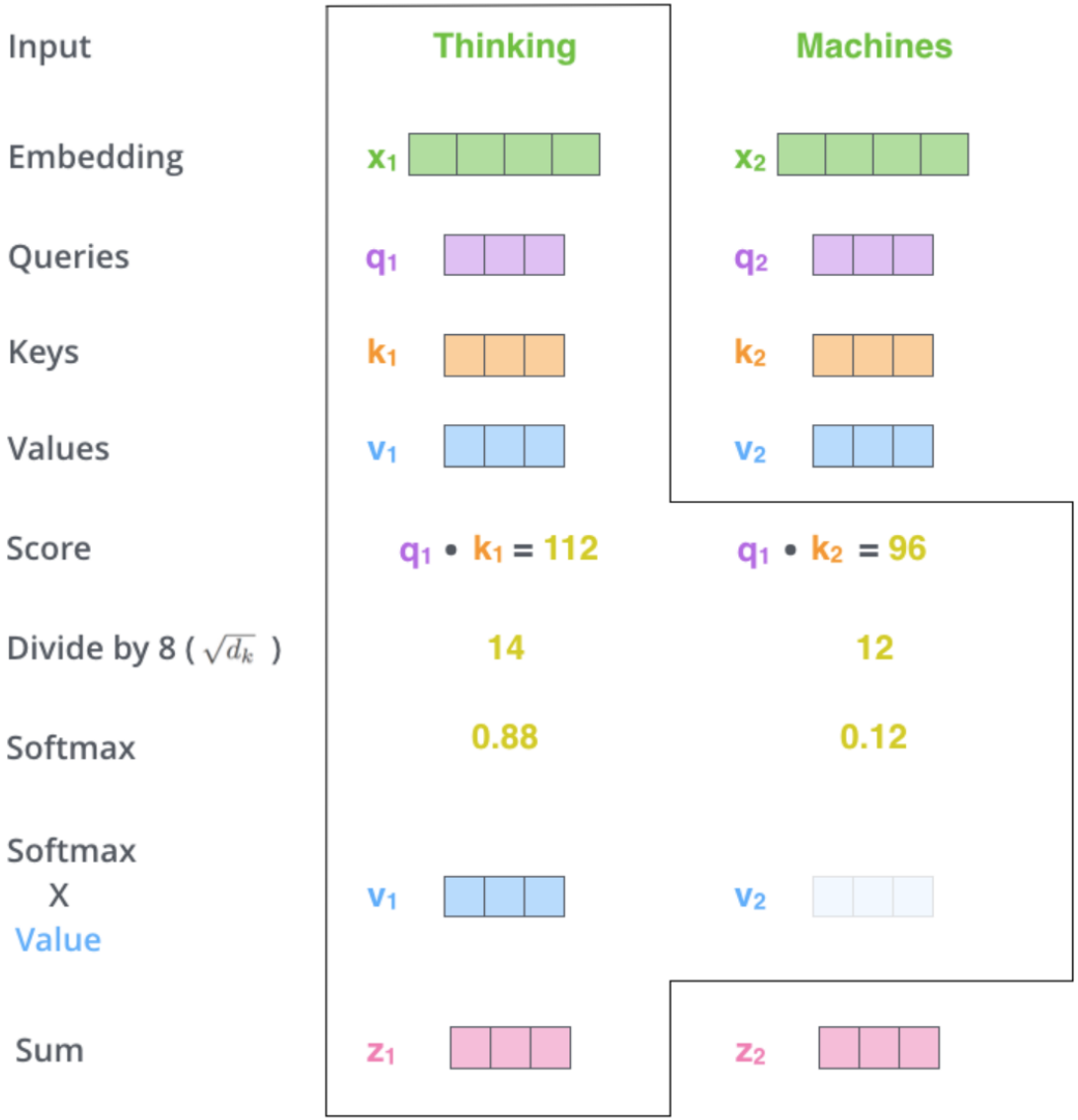
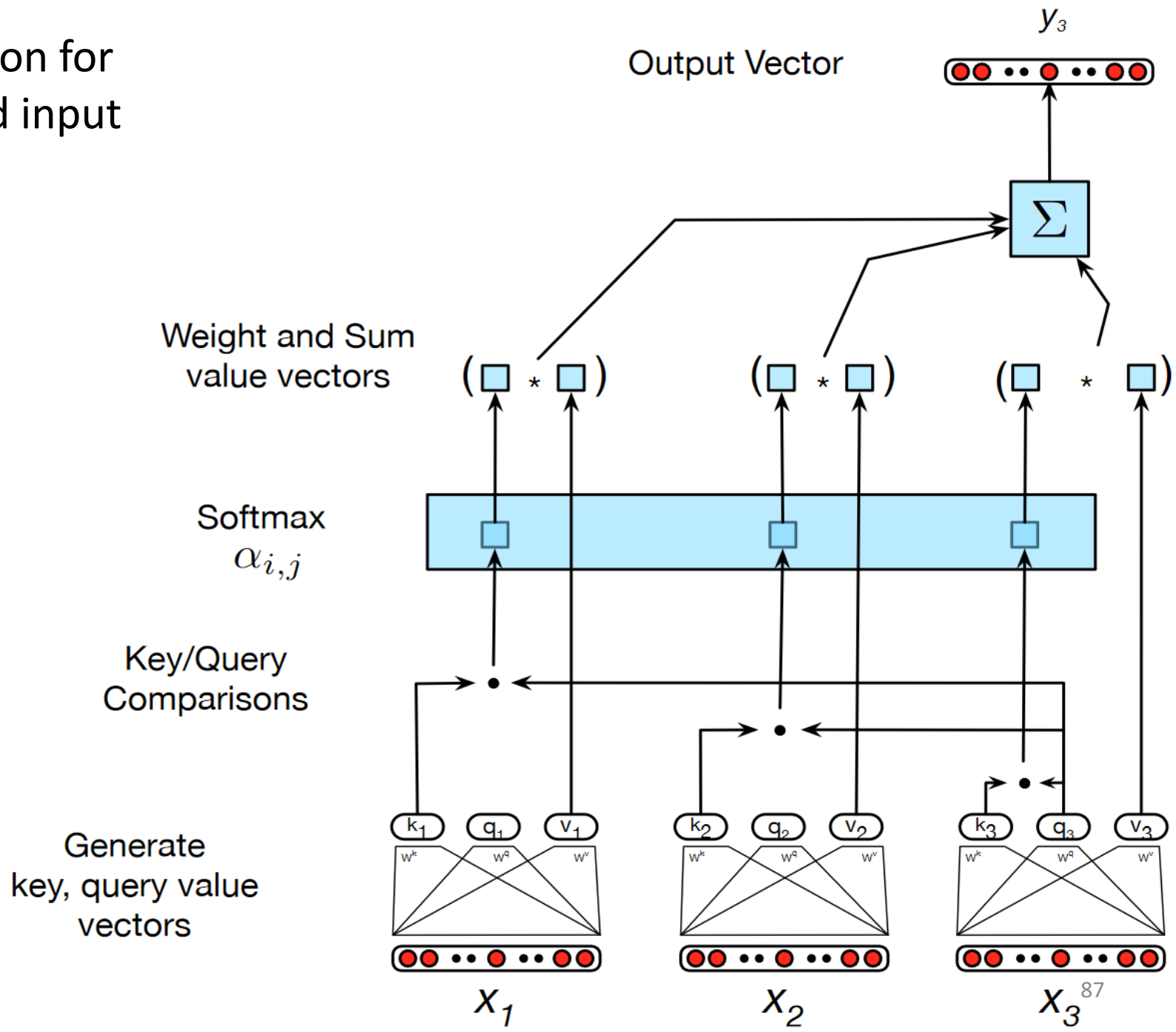
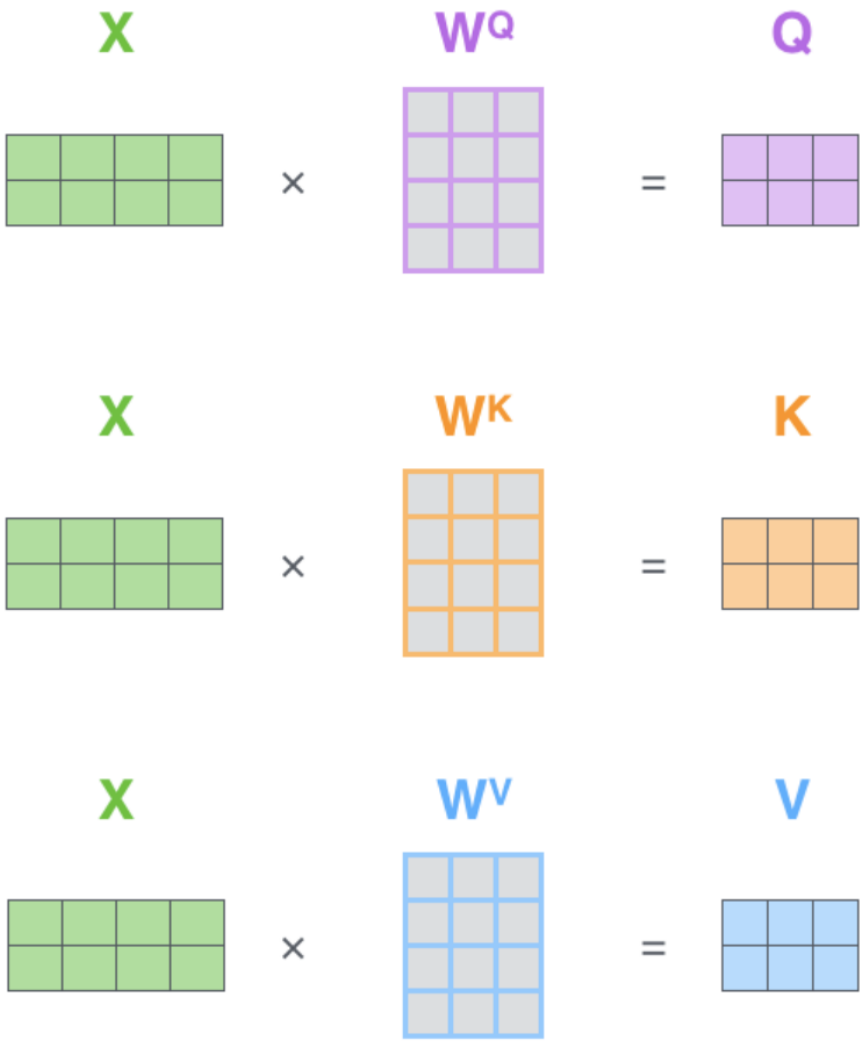


illustration for
the third input



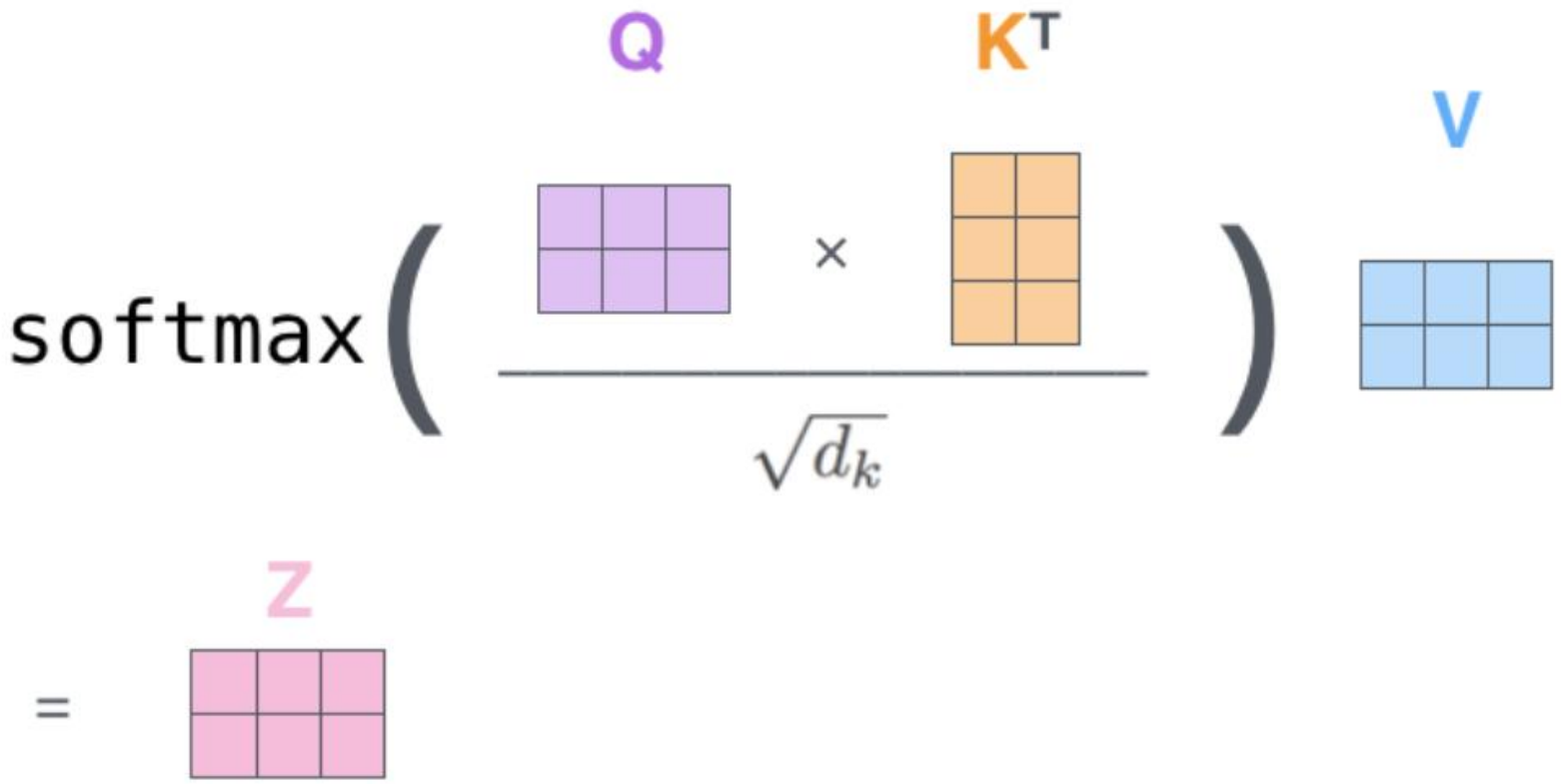
Matrix calculation of self-attention 1/2

Every row in the X matrix corresponds to a word in the input sentence.
The embedding vector x (512) is larger then the q/k/v vectors (64)

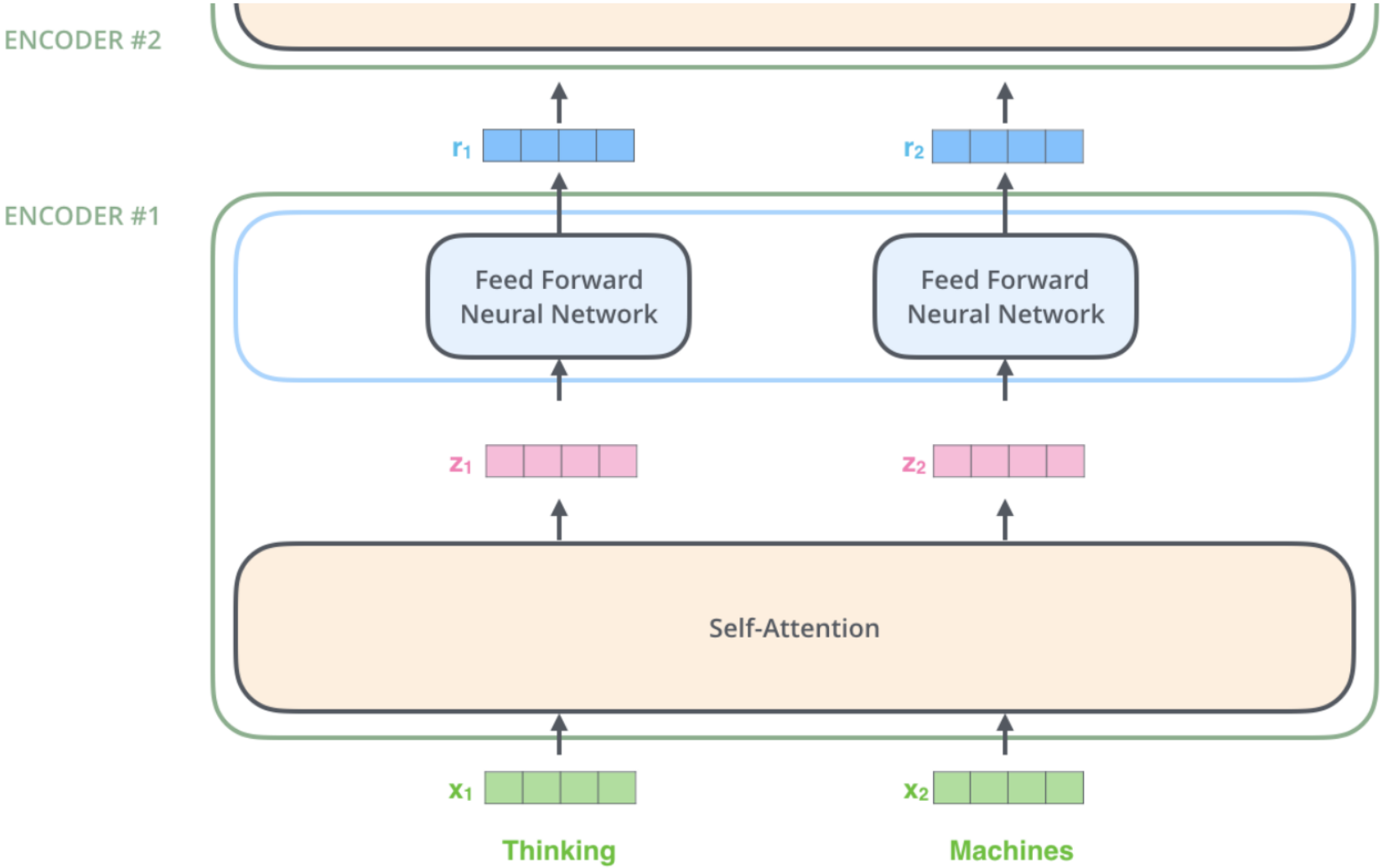


Matrix calculation of self-attention 2/2

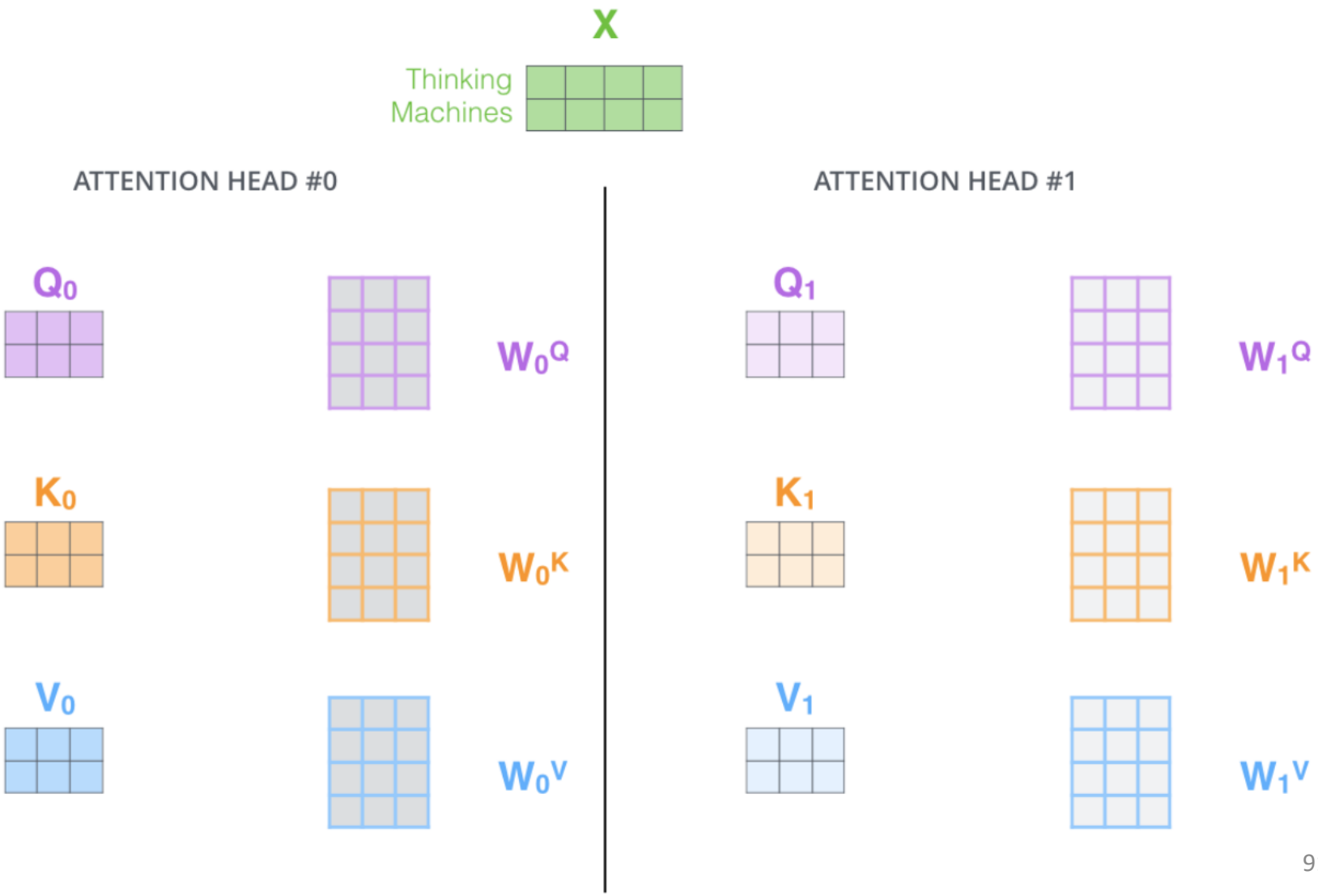
- final calculation



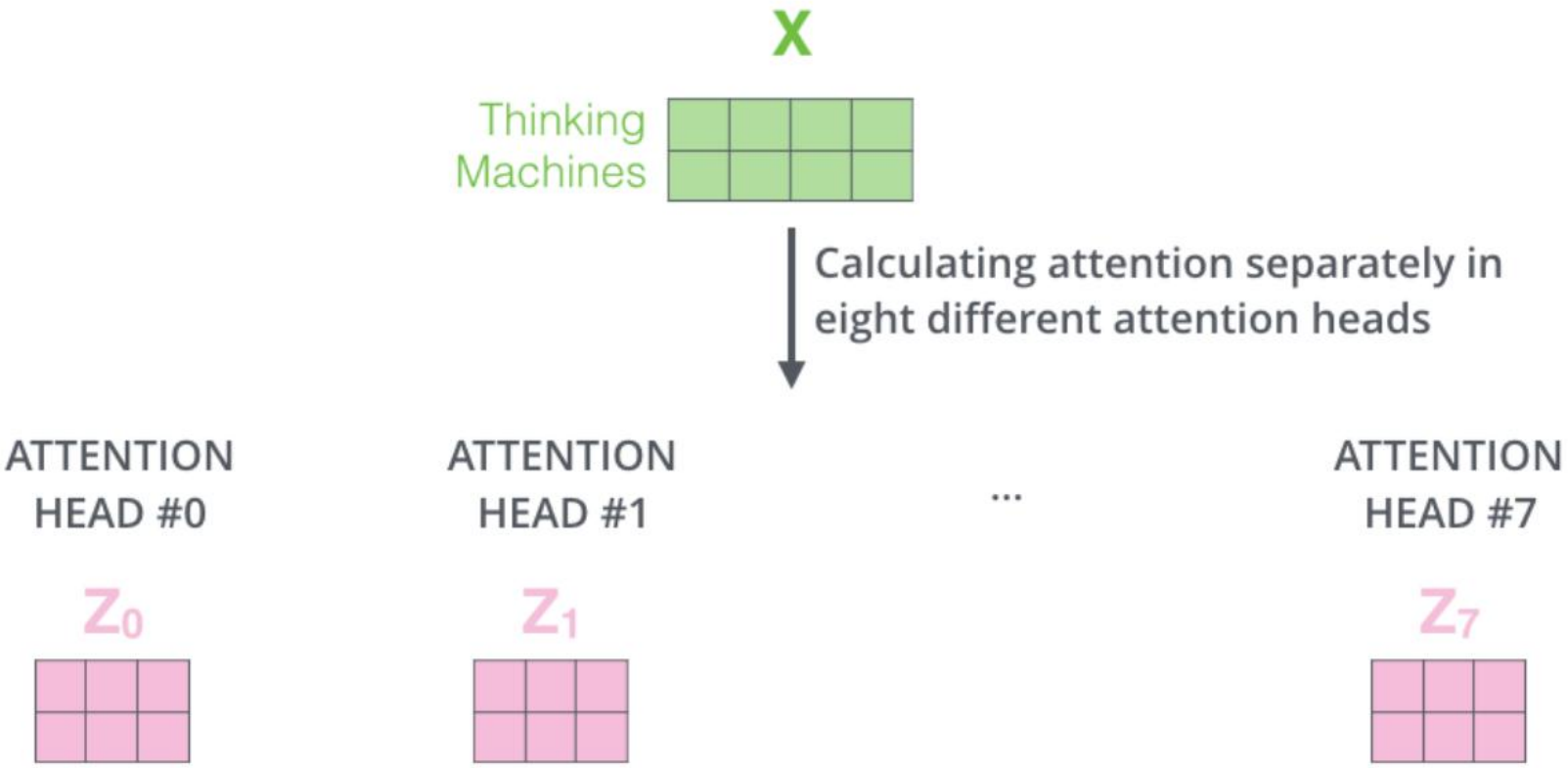
Encoding


ENCODER #2
ENCODER #1
r1
r2
Feed Forward Neural Network
Feed Forward Neural Network
z1
z2
Self-Attention
x1
x2
Thinking
Machines

Example: two attention heads



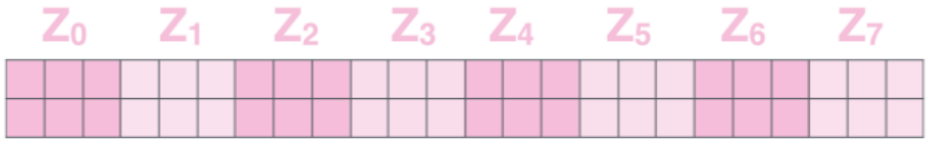
Example: 8 att. heads



- What to do with 8 Z matrices, the feed-forward layer is expecting a single matrix (one vector for each word). We need to condense all attention heads into one matrix.

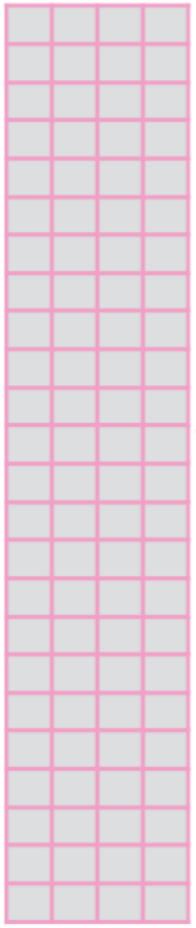
Condensation of attention heads

1) Concatenate all the attention heads



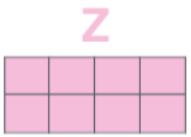
2) Multiply with a weight matrix W^O that was trained jointly with the model

X



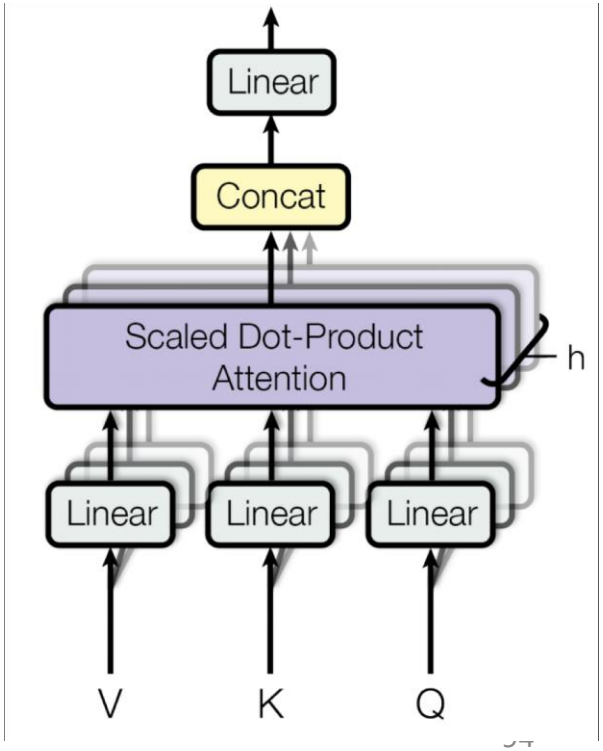
3) The result would be the Z matrix that captures information from all the attention heads. We can send this forward to the FFNN

=



Computing multi-head attention

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, ..., \text{head}_\text{h})W^O$$
$$\text{where head}_\text{i} = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$



Summary of self-attention

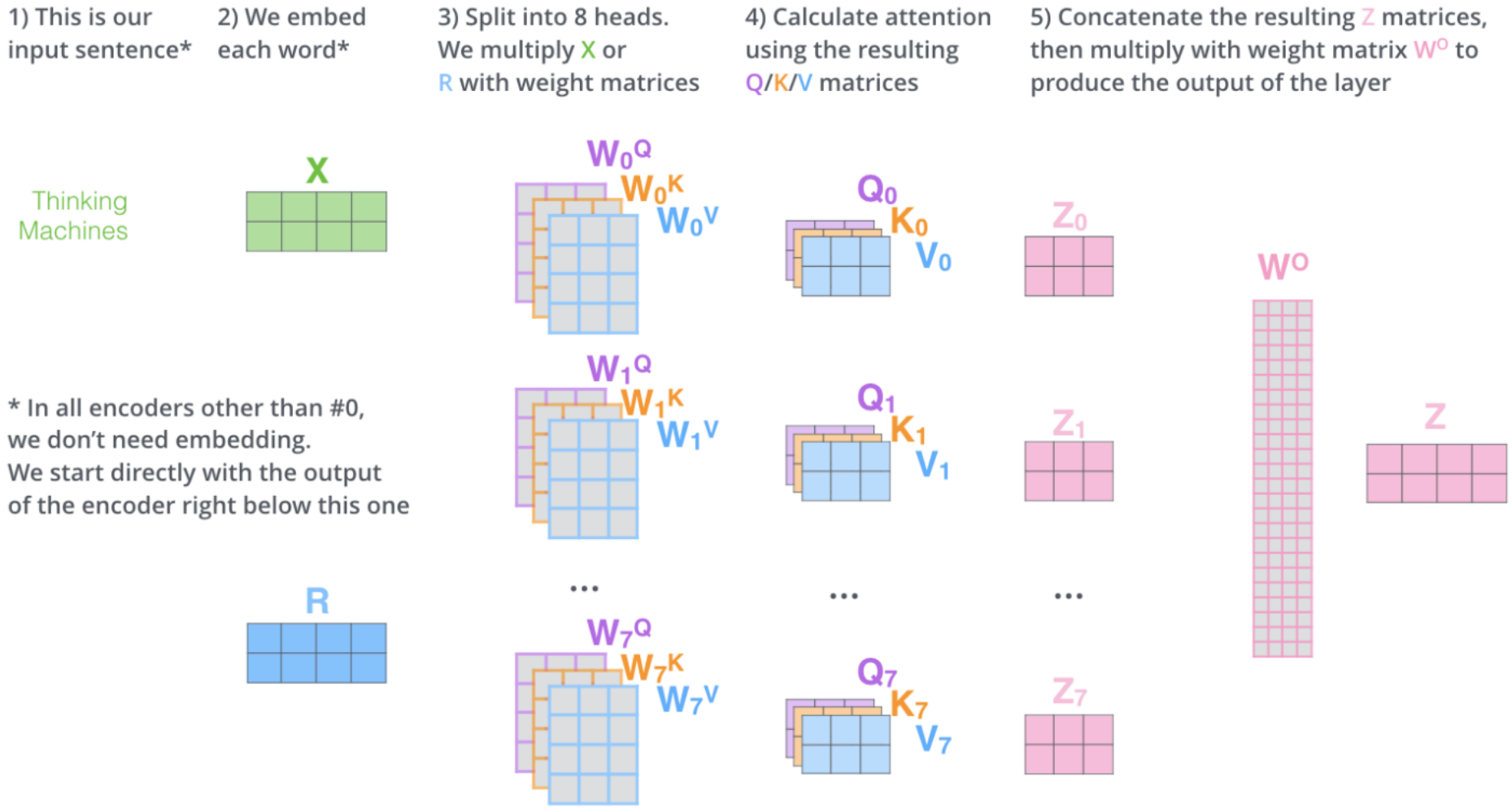


Illustration of self-attention: 1 head

- encoder #5 (the top encoder in the stack)
- As we encode the word "it", one attention head is focusing most on "the animal", while another is focusing on "tired" -- in a sense, the model's representation of the word "it" bakes in some of the representation of both "animal" and "tired".

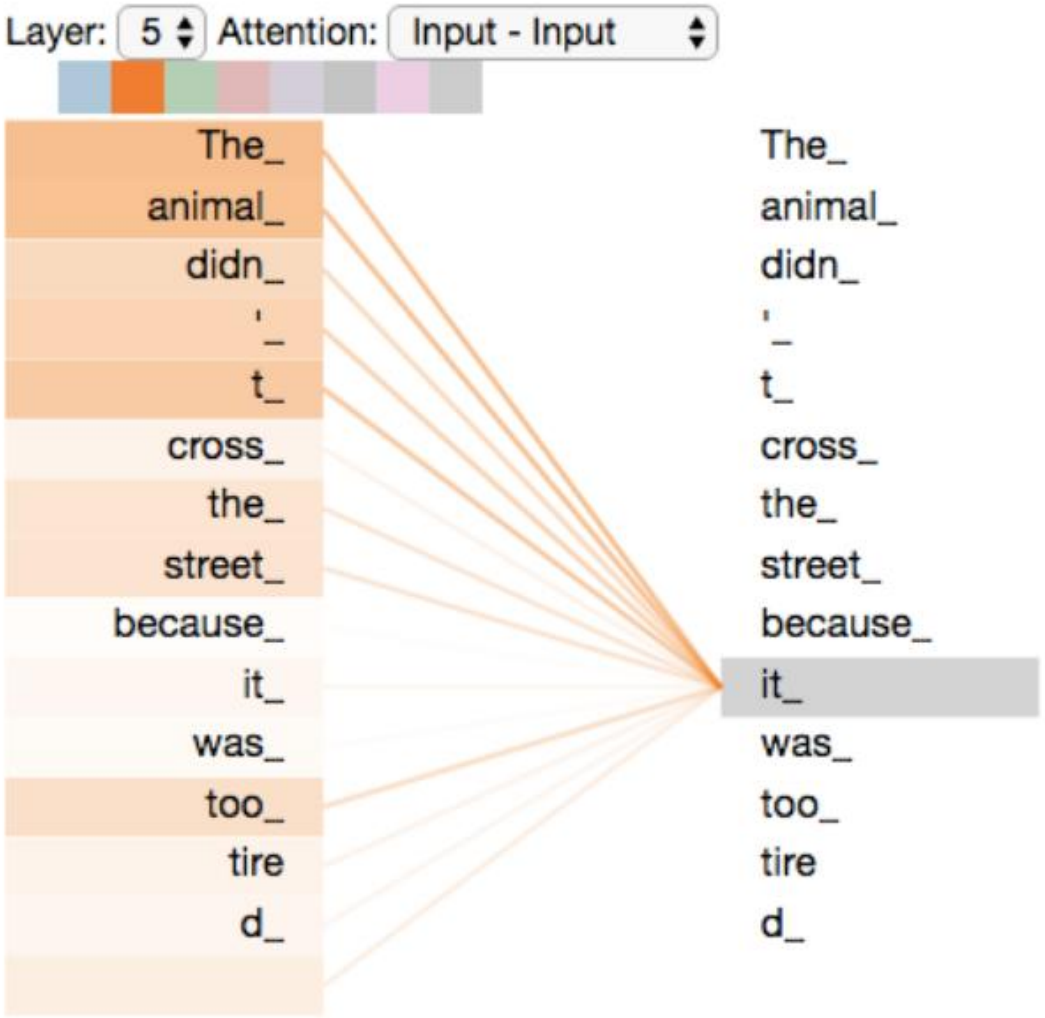
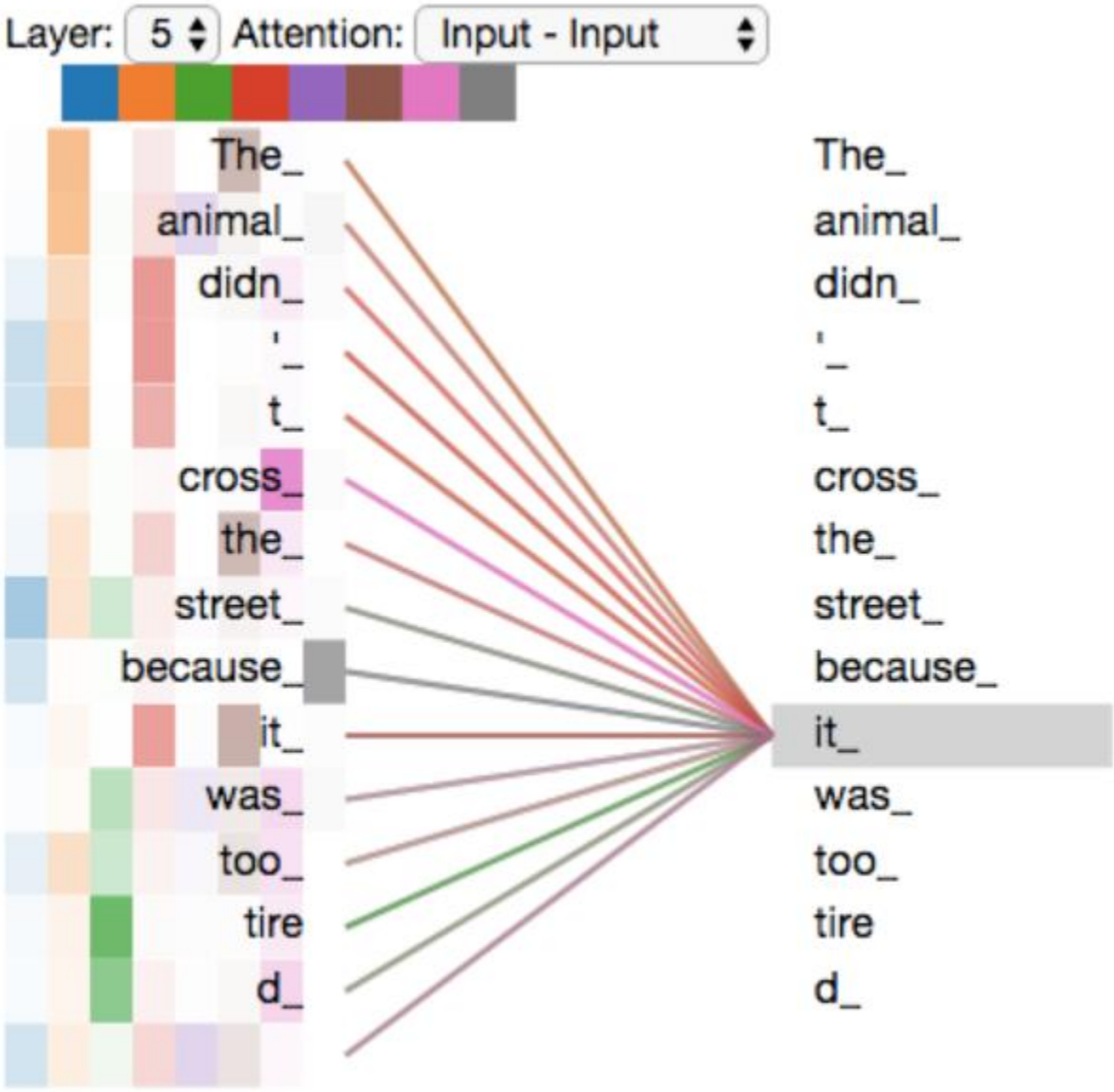
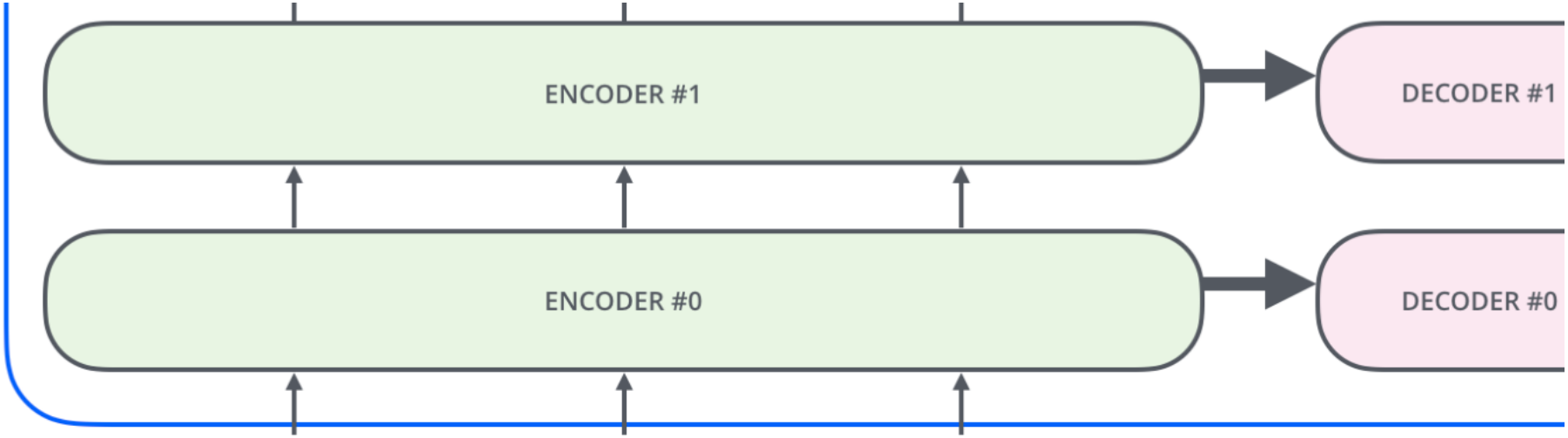


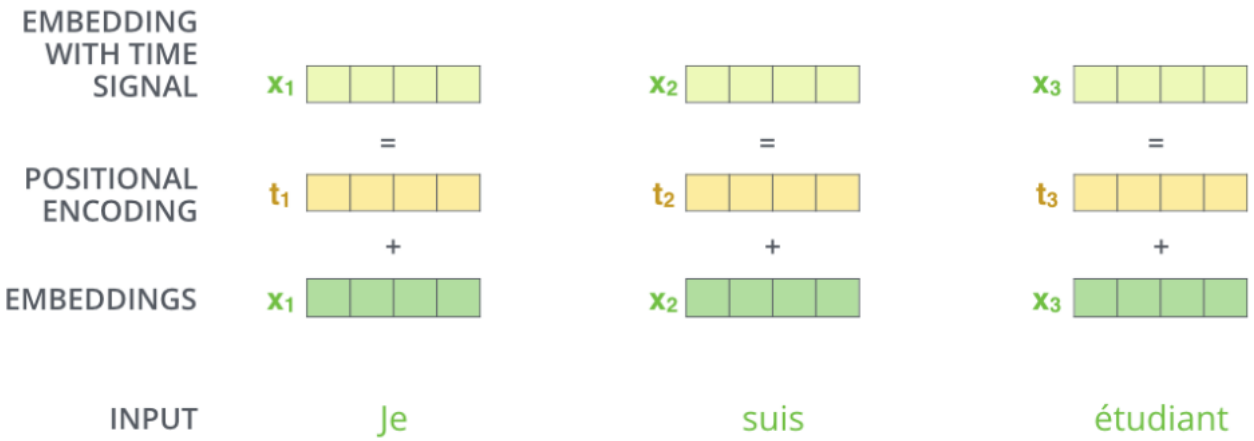
Illustration of self-attention: all heads

- all the attention heads in one picture are harder to interpret



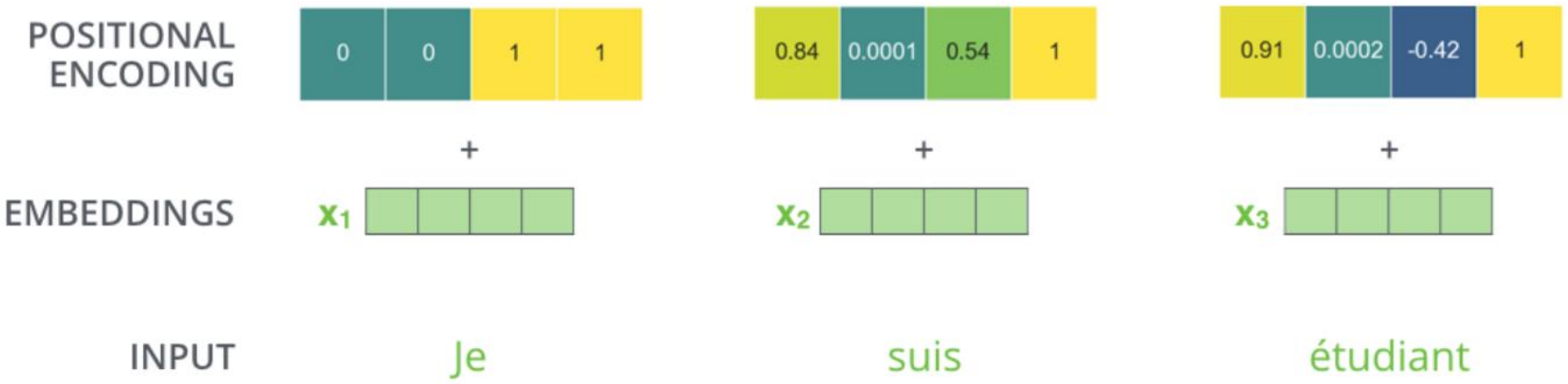
Adding position encoding



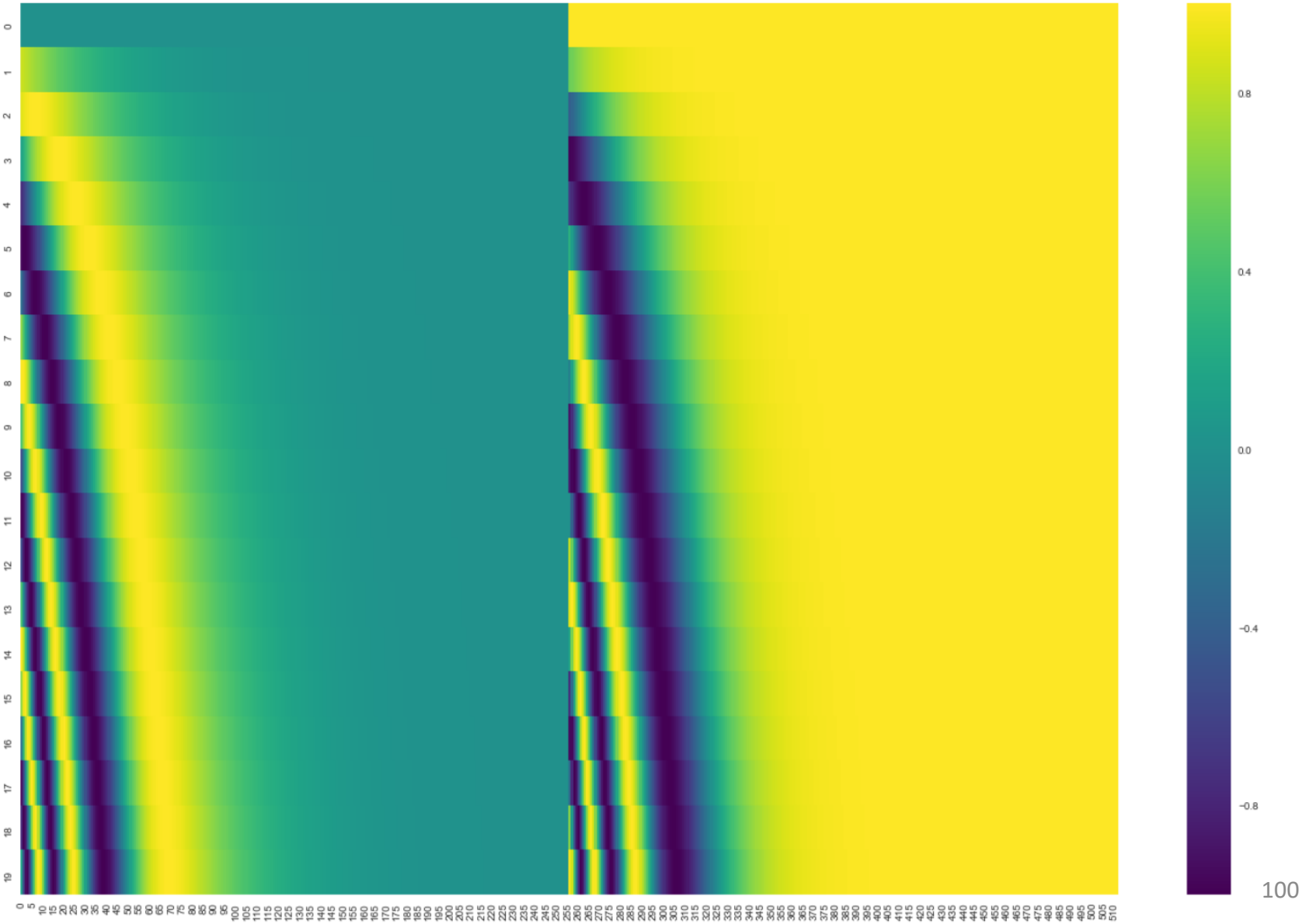


Example: encoding position

- the values of positional encoding vectors follow a specific pattern.



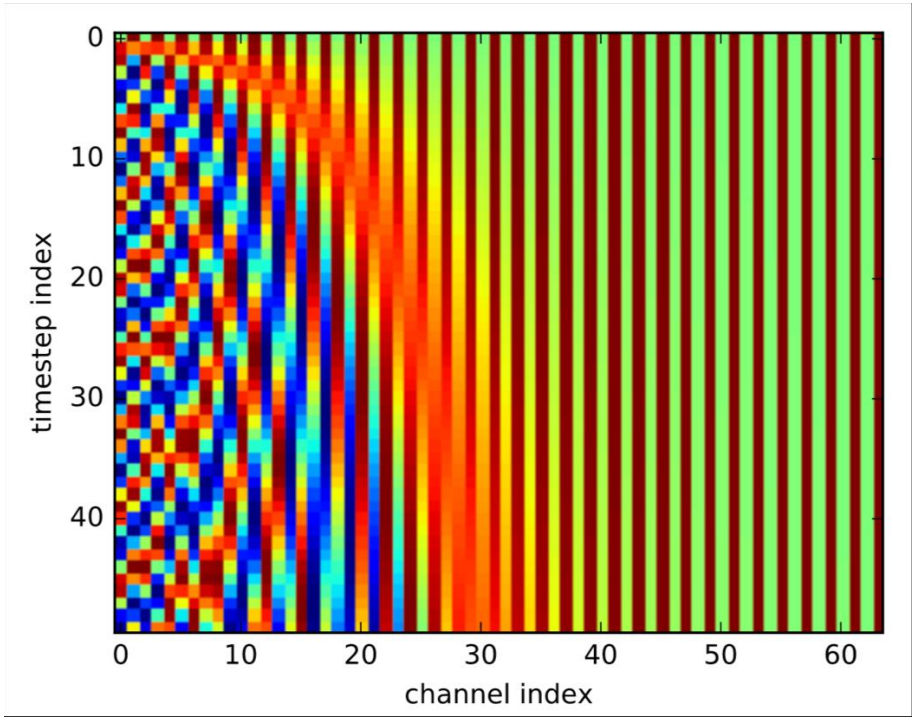
Example of positional encoding



Another positional encoding

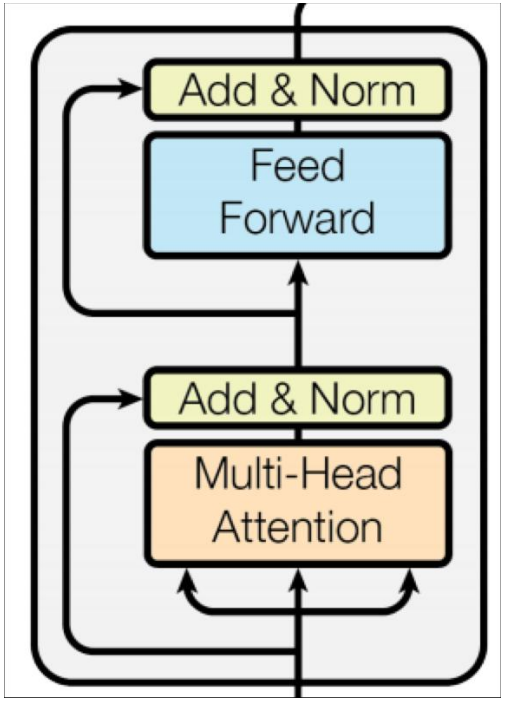
$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{\text{model}}})$$

$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{\text{model}}})$$

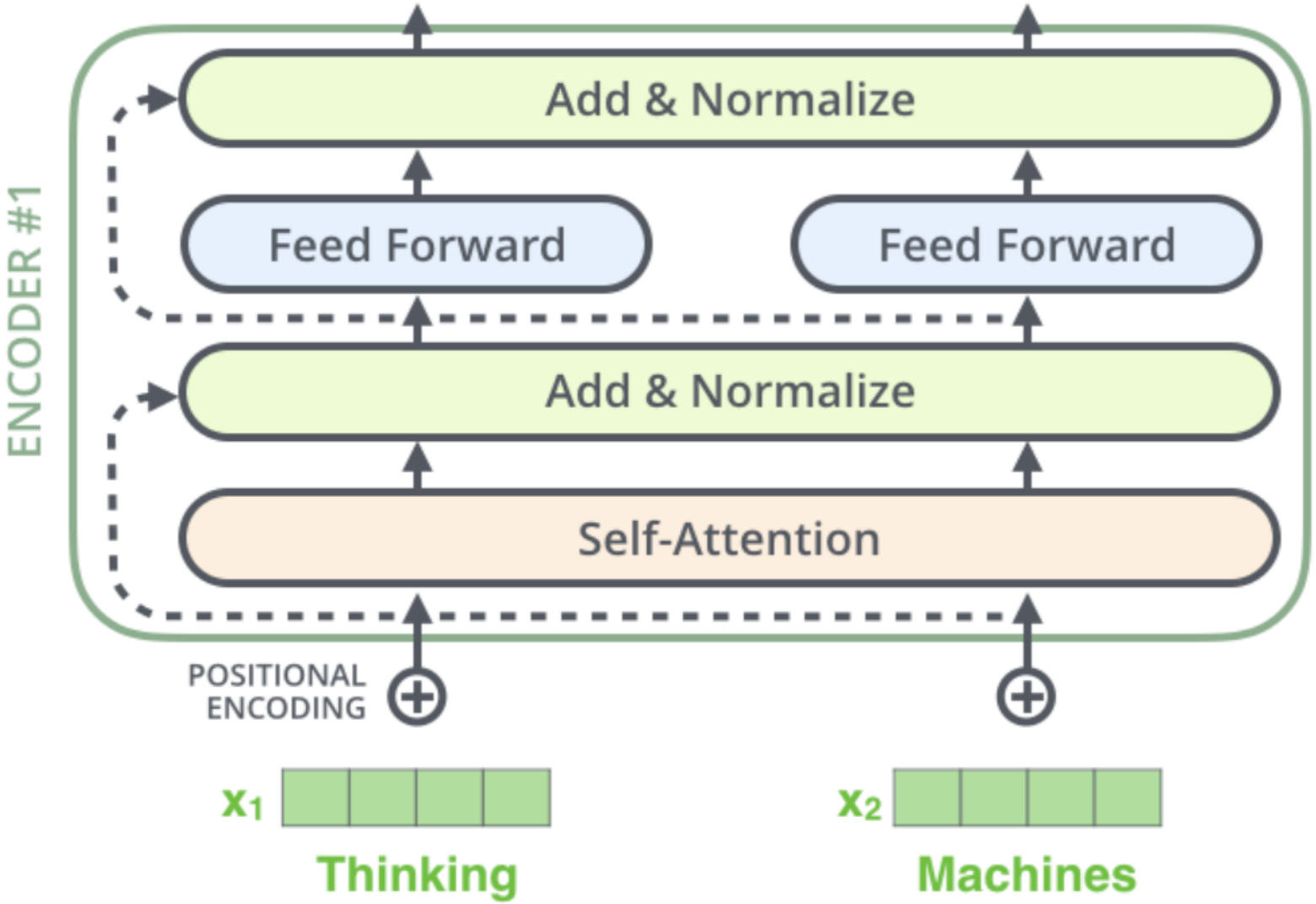


Encoder blocks

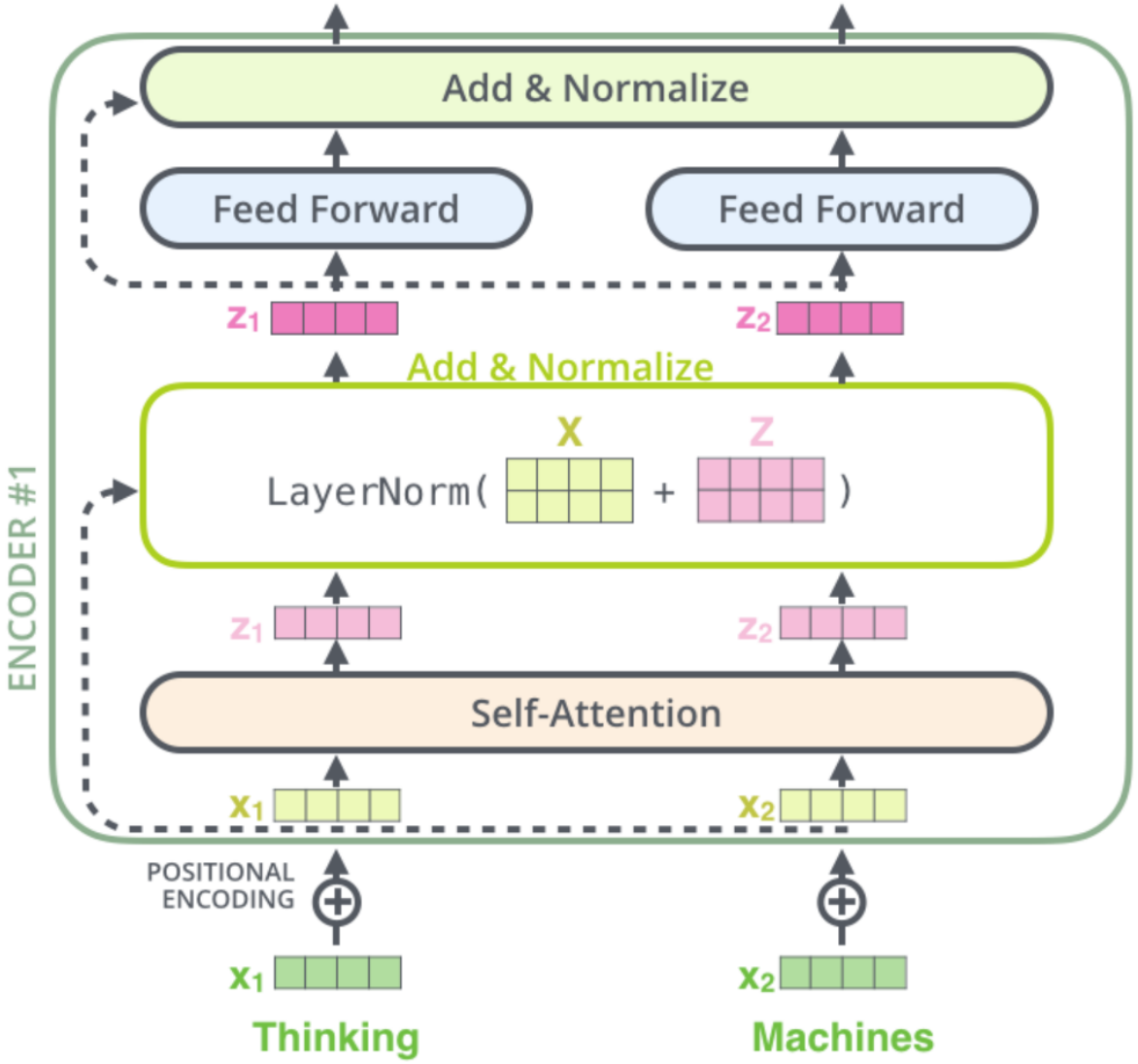
- Each block has two “sublayers”
  - Multihead attention
  - 2-layer feed-forward neural network (with ReLU)
- Each of these two steps also has a residual (short-circuit) connection and LayerNorm, i.e.:
  - LayerNorm(x + Sublayer(x))



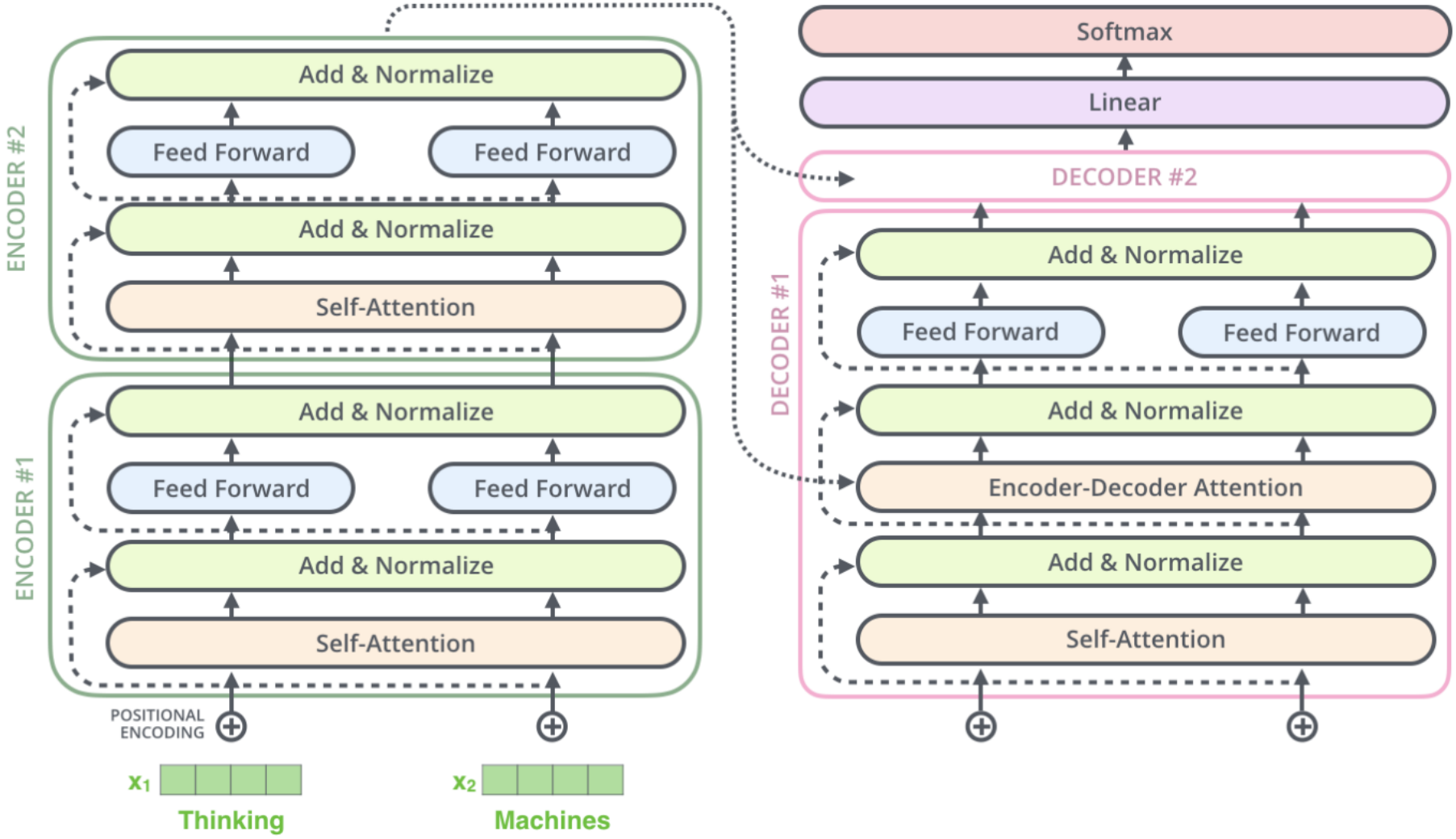
Architecture with residual connection – top level view



Architecture with residual connection – example

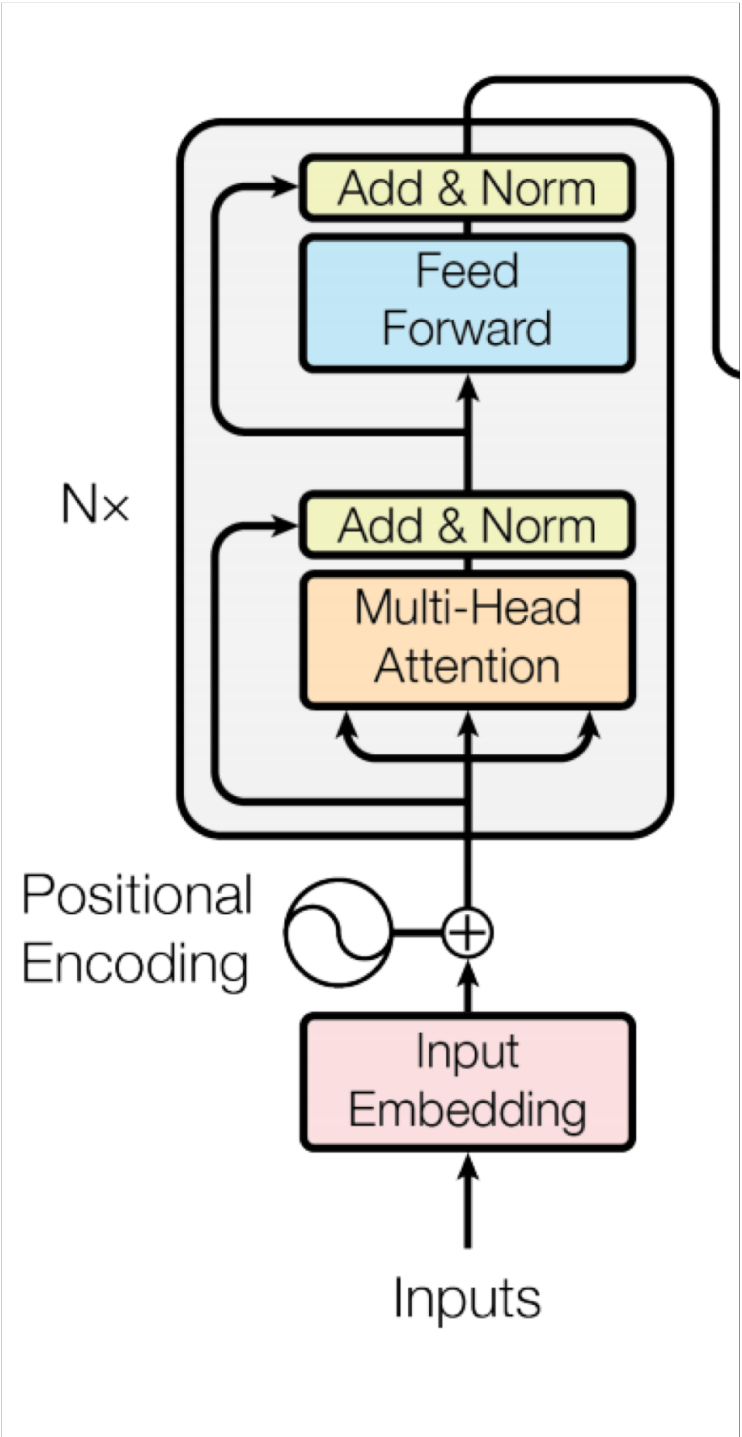


Example: 2 stacked transformer



Complete encoder

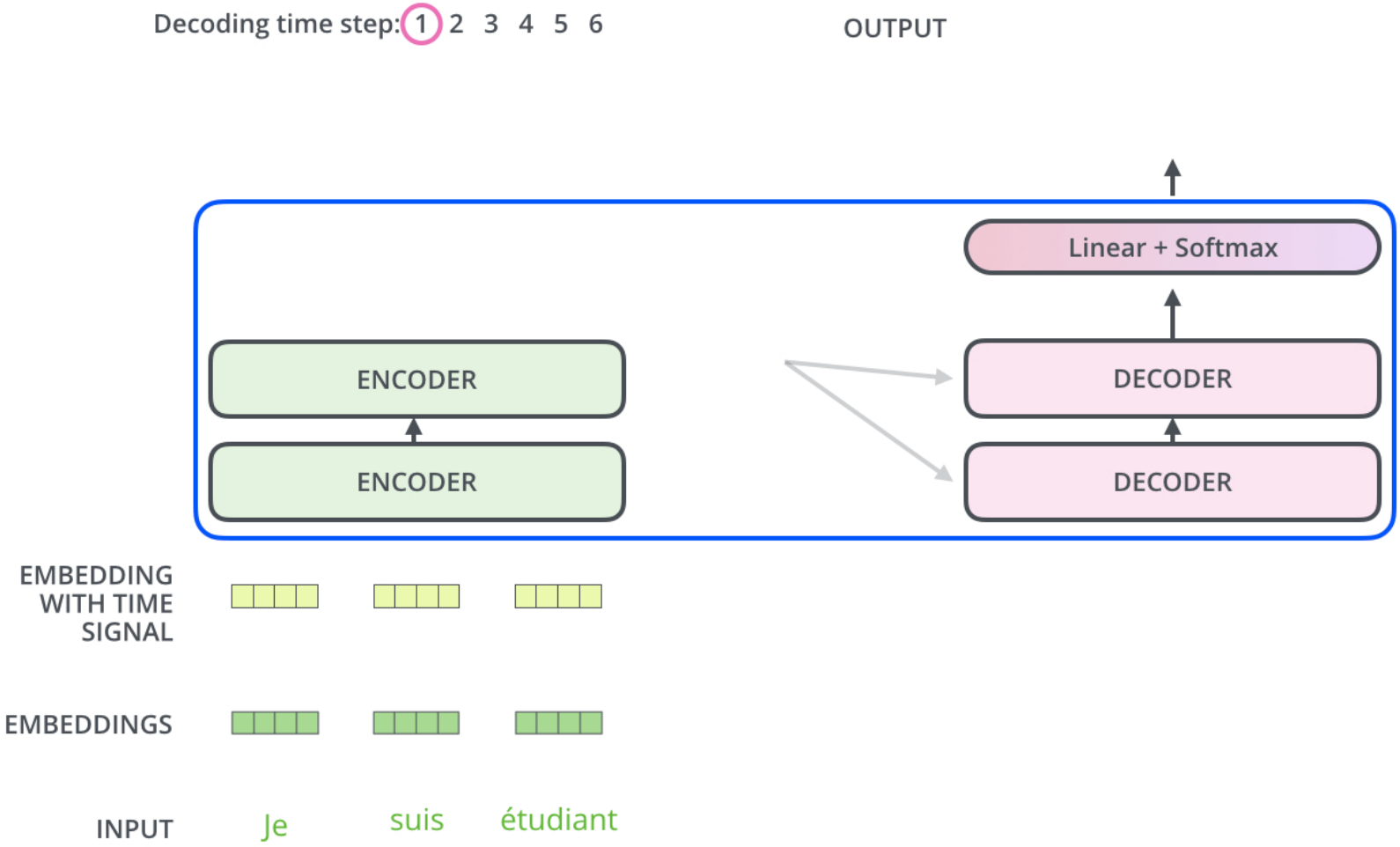
- each encoder block is repeated several times, e.g., 6 times



Decoder

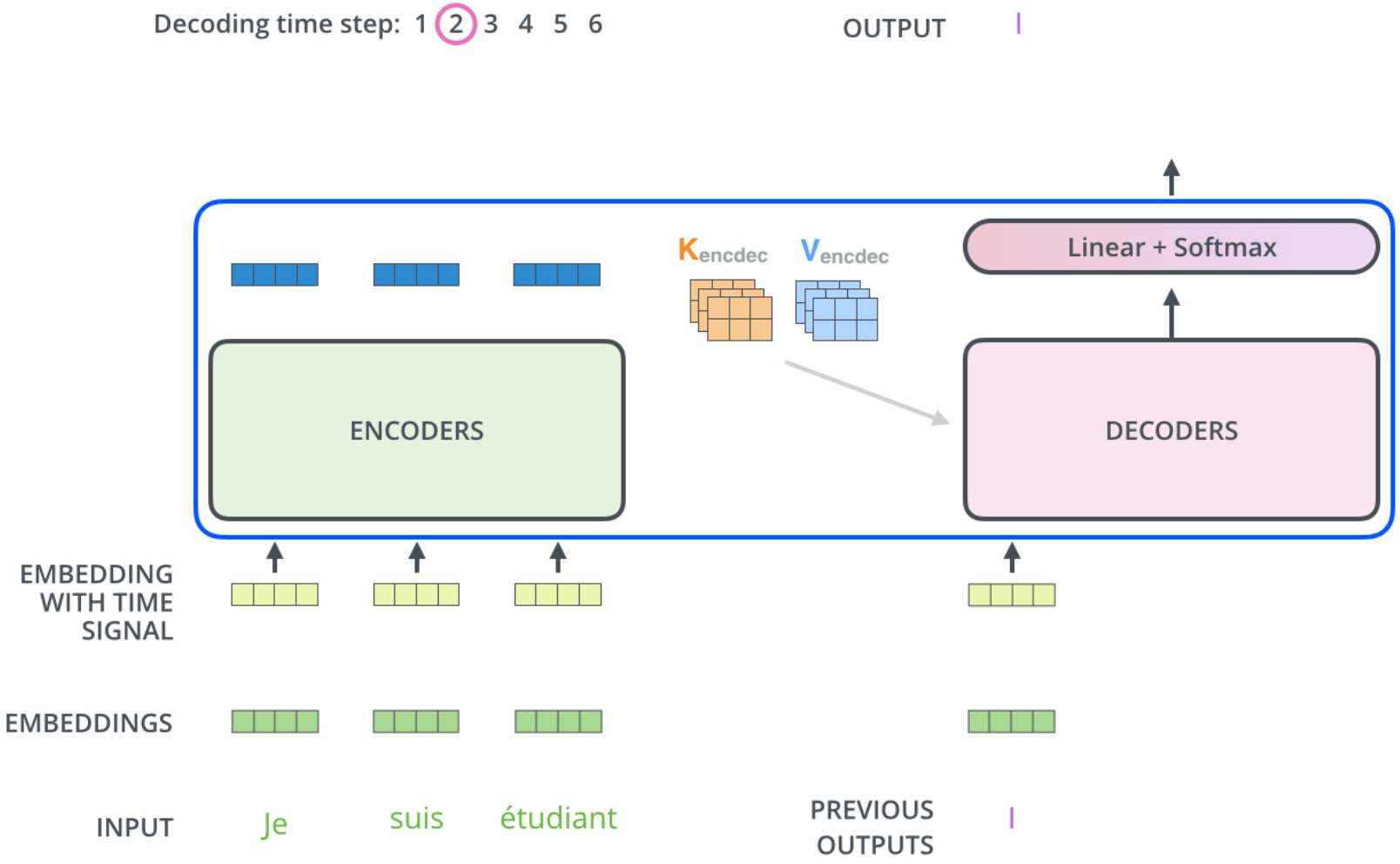
- Decoders have the same components as encoders
- An encoder starts by processing the input sequence.
- The output of the top encoder is transformed into a set of attention vectors K and V.
- These are used by each decoder in its “encoder-decoder attention” layer which helps the decoder to focus on appropriate places in the input sequence.

Encoder-decoder in action 1/2



After finishing the encoding phase, we begin the decoding phase. Each step in the decoding phase outputs an element from the output sequence (the English translation sentence in this case).

Encoder-decoder in action 2/2



The steps repeat until a special symbol indicating the end of output is generated. The output of each step is fed to the bottom decoder in the next time step. We add positional encoding to decoder inputs to indicate the position of each word.

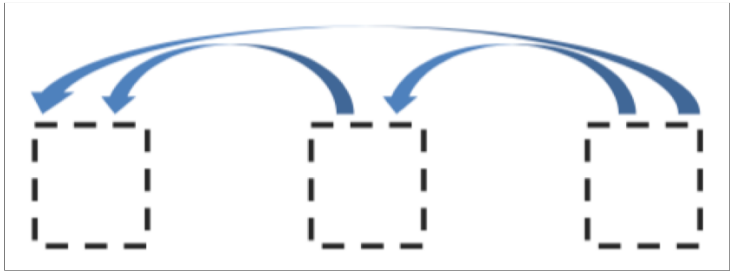
Self-attention and encoder-decoder attention in the decoder

- In the decoder, the self-attention layer is only allowed to attend to itself and earlier positions in the output sequence (to maintain the autoregressive property).
- This is done by masking future positions (setting them to -inf) before the softmax step in the self-attention calculation.

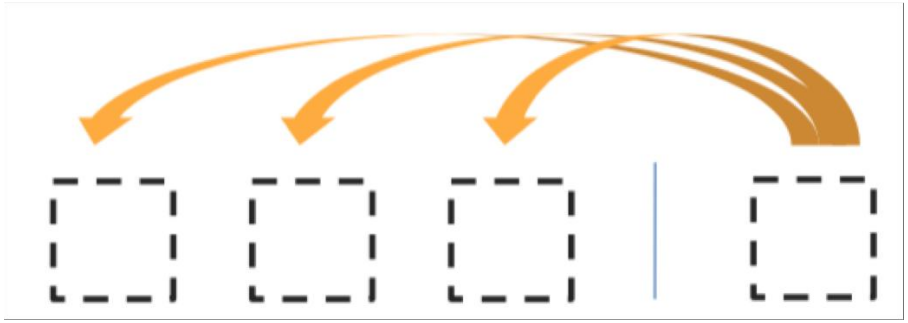
- The “Encoder-Decoder Attention” layer works just like multiheaded self-attention, except it creates its Q (queries) matrix from the layer below it, and takes the K (keys) and V (values) matrix from the output of the encoder stack.

Attentions in the decoder

1. Masked decoder self-attention on previously generated outputs

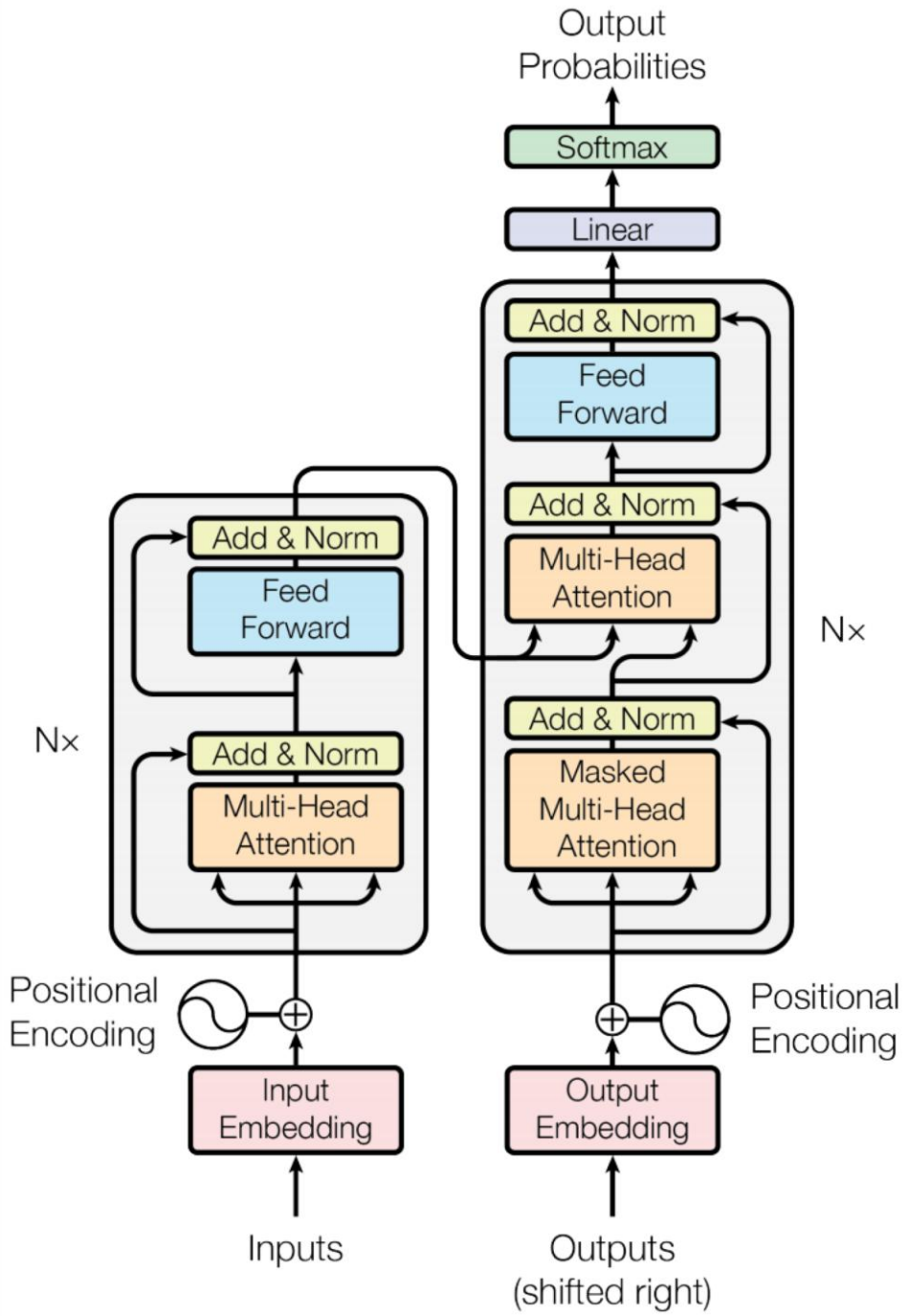


2. Encoder-Decoder Attention, where queries come from previous decoder layer and keys and values come from output of the encoder



Animated workings of attention in transformer

One encoder-decoder block



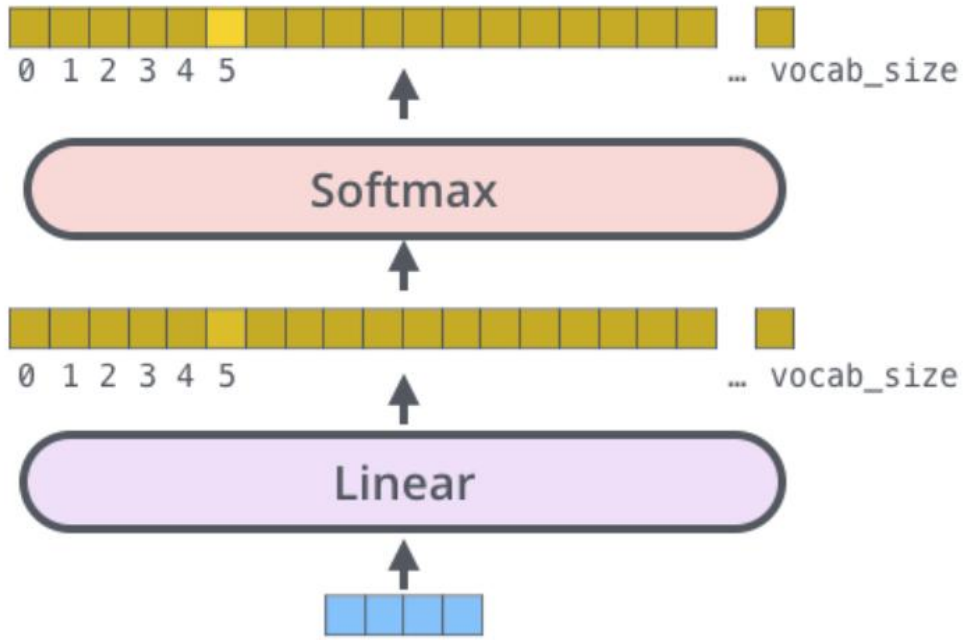
Producing the output words

Which word in our vocabulary is associated with this index?

am

Get the index of the cell with the highest value (**argmax**)

5



Training the transformer

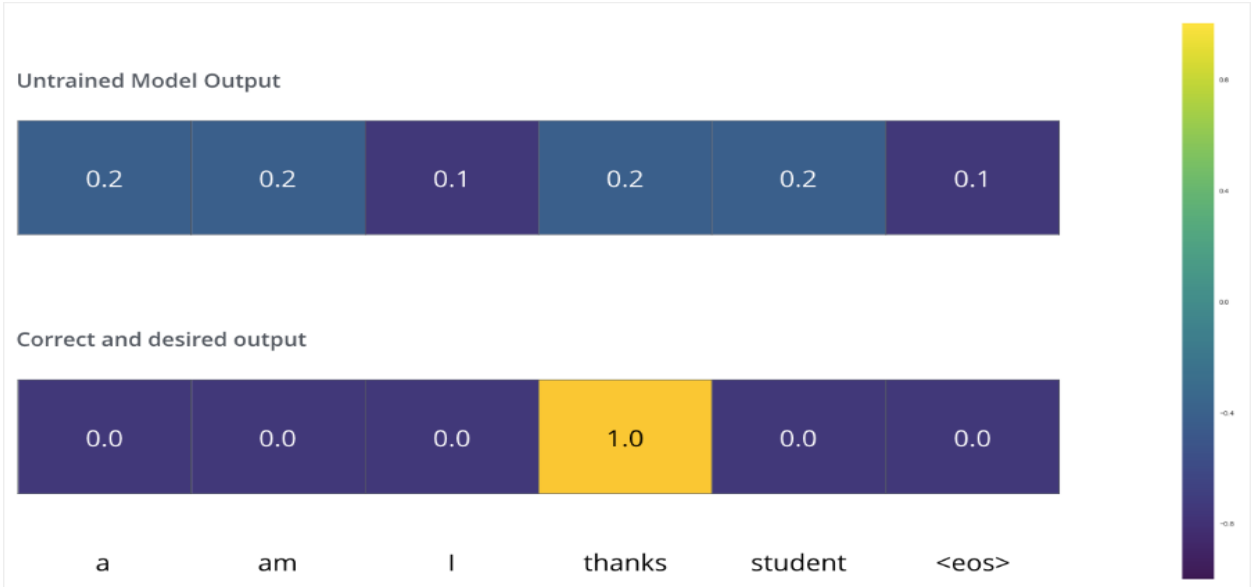
- During training, an untrained model would go through the exactly the same forward pass. But since we are training it on a labeled training dataset, we can compare its output with the actual correct output.
- For illustration, let's assume that our output vocabulary only contains six words(a, am, i, thanks, student, <eos>)
- The input is typically in the order of 10^4 (e.g., 30 000)

Output Vocabulary

WORD	a	am	I	thanks	student	<eos>
INDEX	0	1	2	3	4	5

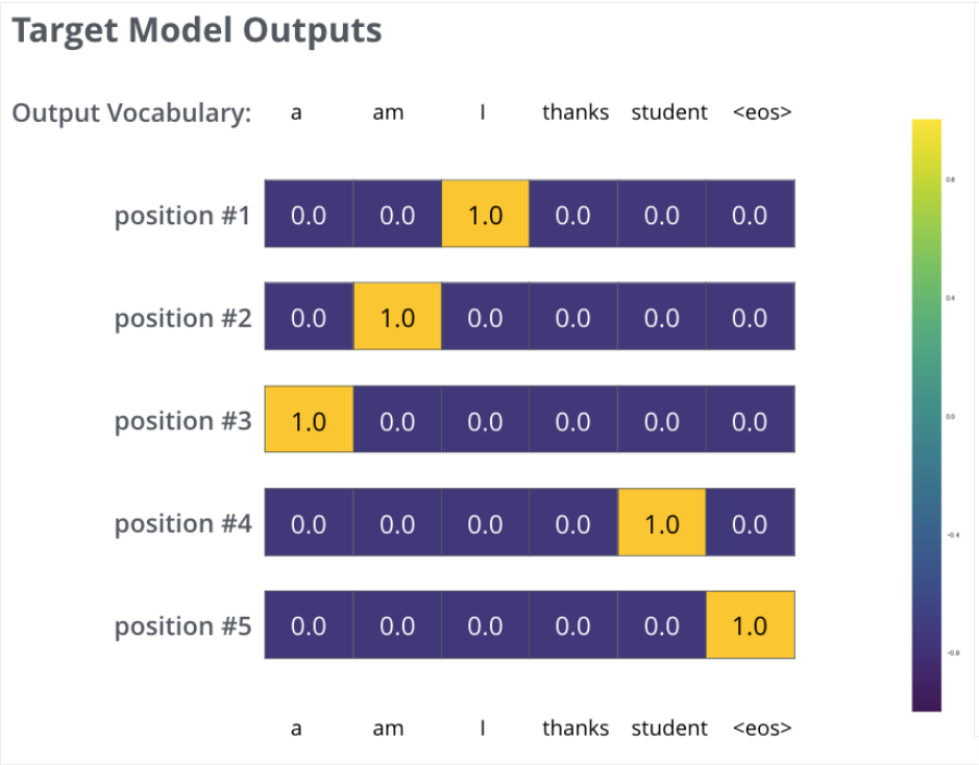
The Loss Function

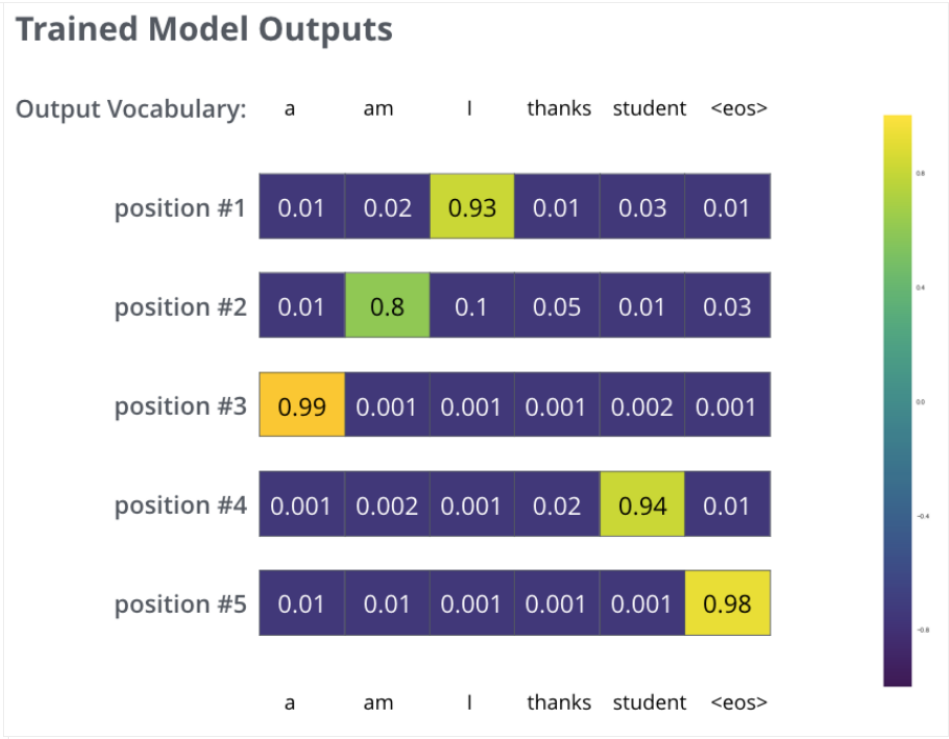
- Evaluates the difference between the true output and the returned output
- Transformer typically uses cross-entropy or Kullback–Leibler divergence.
- The model output is a probability distribution, the true output is 1-hot encoded, e.g.,



Loss evaluation for sequences

- Loss function has to be evaluated for the whole sentence, not just a single word.
- Transformers use greedy decoding or beam search.





Three flavours of transformers

- encoder only (BERT)
- encoder-decoder (machine translation, T5)
- decoder only (GPT)

BERT

- Combines several tasks
- Predicts masked words in a sentence
- Also predicts the order of sentences: is sentence A followed by sentence B or not?
- Combines several hidden layers of the network
- Uses transformer neural architecture, only the encoder part
- Uses several fine-tuned parameters
- Multilingual variant supports 104 languages by training on Wikipedia
- Many publicly available variants.

Devlin J, Chang MW, Lee K, Toutanova K (2019) BERT: Pre-training of deep bidirectional transformers for language understanding. In: Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers), pp 4171–4186

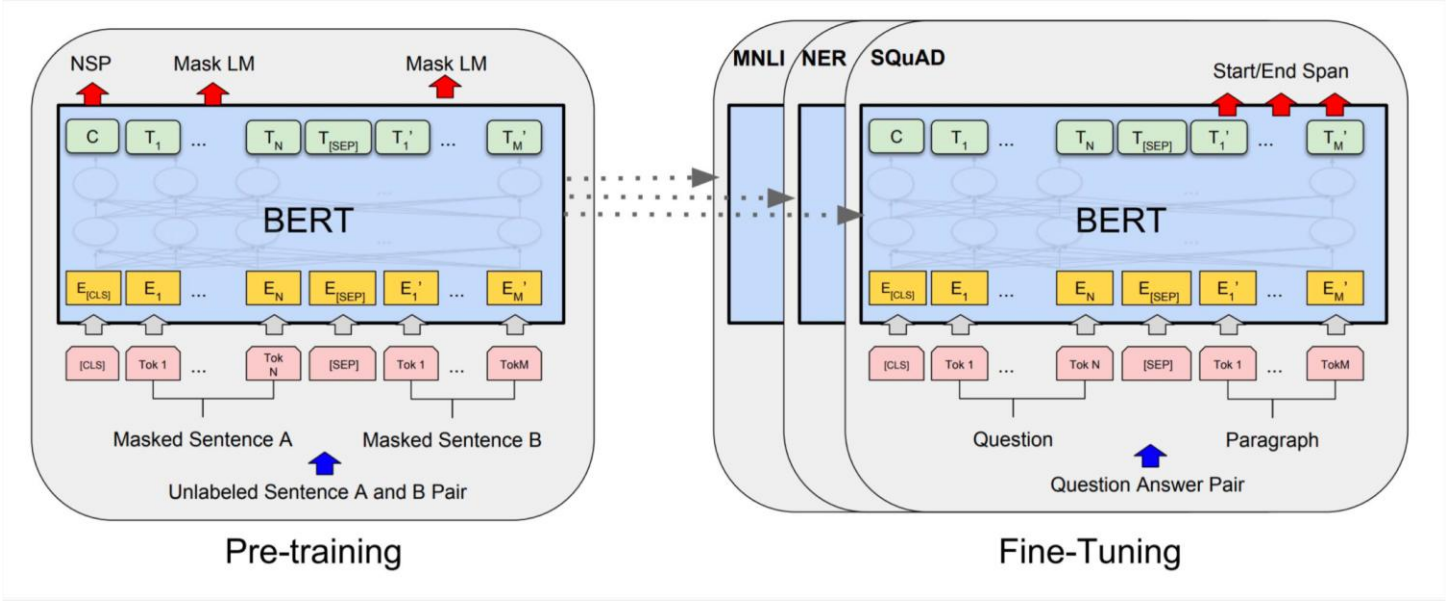
Many BERT-like embeddings

- XLM-R was trained on 2.5 TB of texts in 100 languages
- For Slovene: fastText (a variant of word2vec), ELMo, SloBERTa
- Trilingual BERT – CroSloEngual
- On HuggingFace
- Hundreds of papers investigating BERT-like models in major ML & NLP conferences

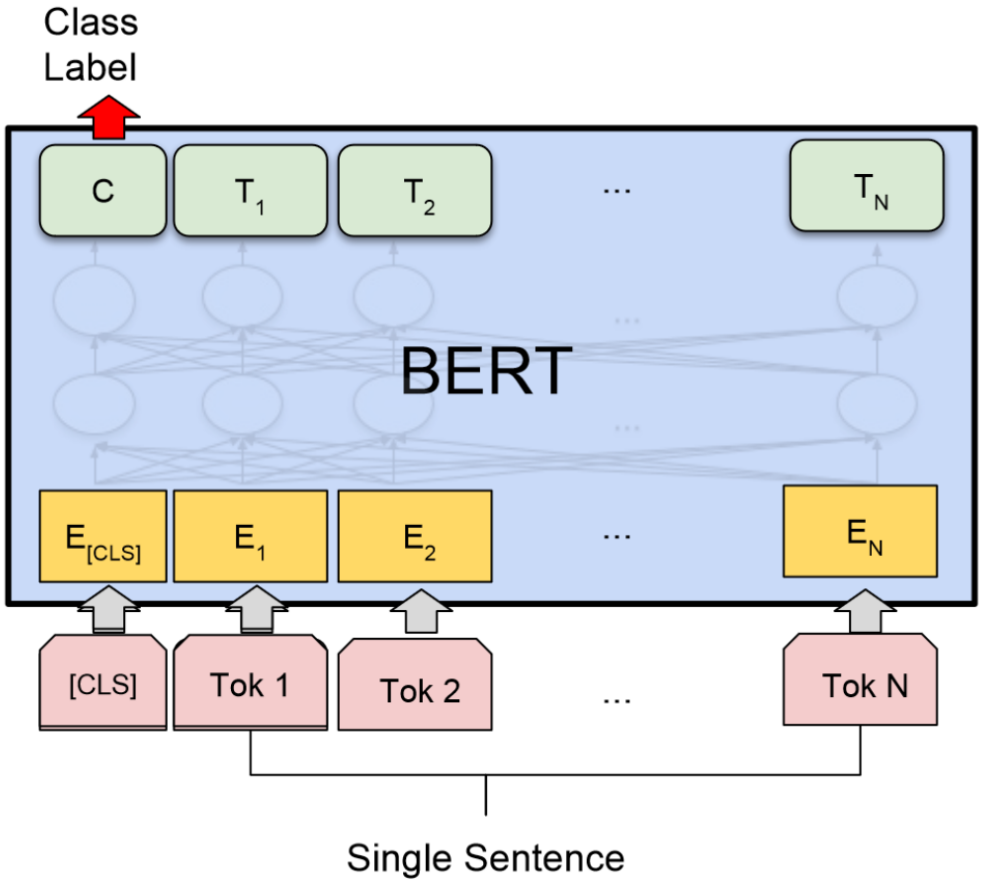
Ulčar, Matej and Marko Robnik-Šikonja. FinEst BERT and CroSloEngual BERT: less is more in multilingual models.
In Proceedings of Text, Speech, and Dialogue, TSD2020, 2020.

Use of BERT

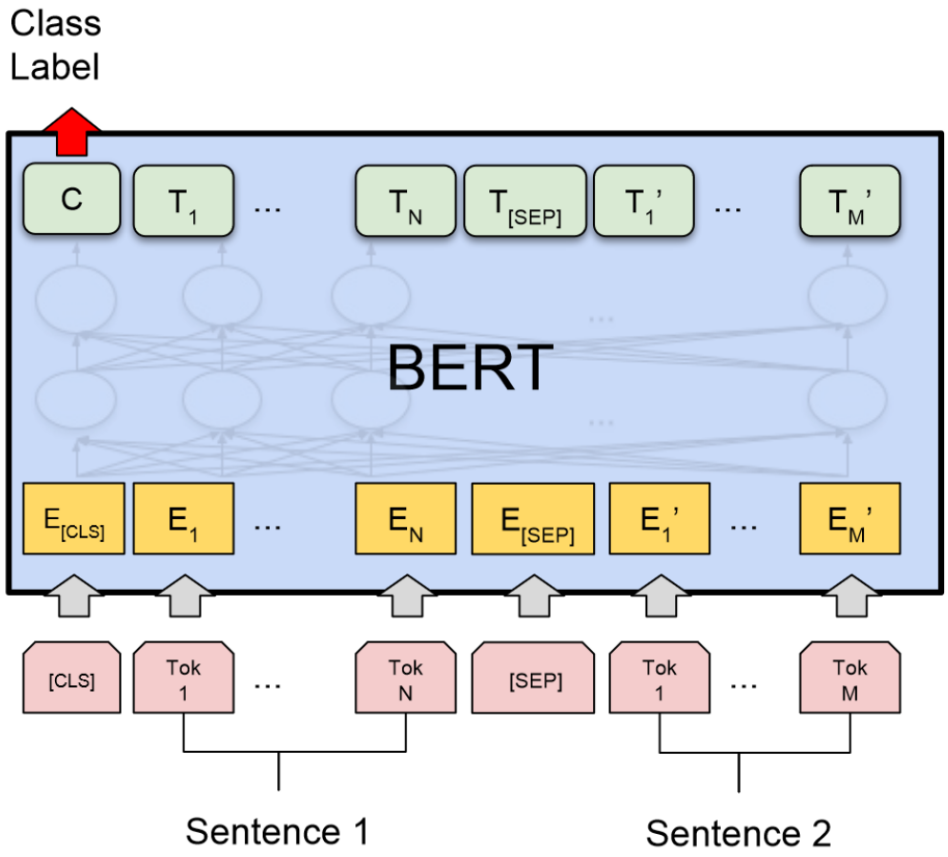
- train a classifier built on the top layer for each task that you fine tune for, e.g., Q&A, NER, inference
- achieves state-of-the-art results for many tasks
- GLUE and SuperGLUE tasks for NLI



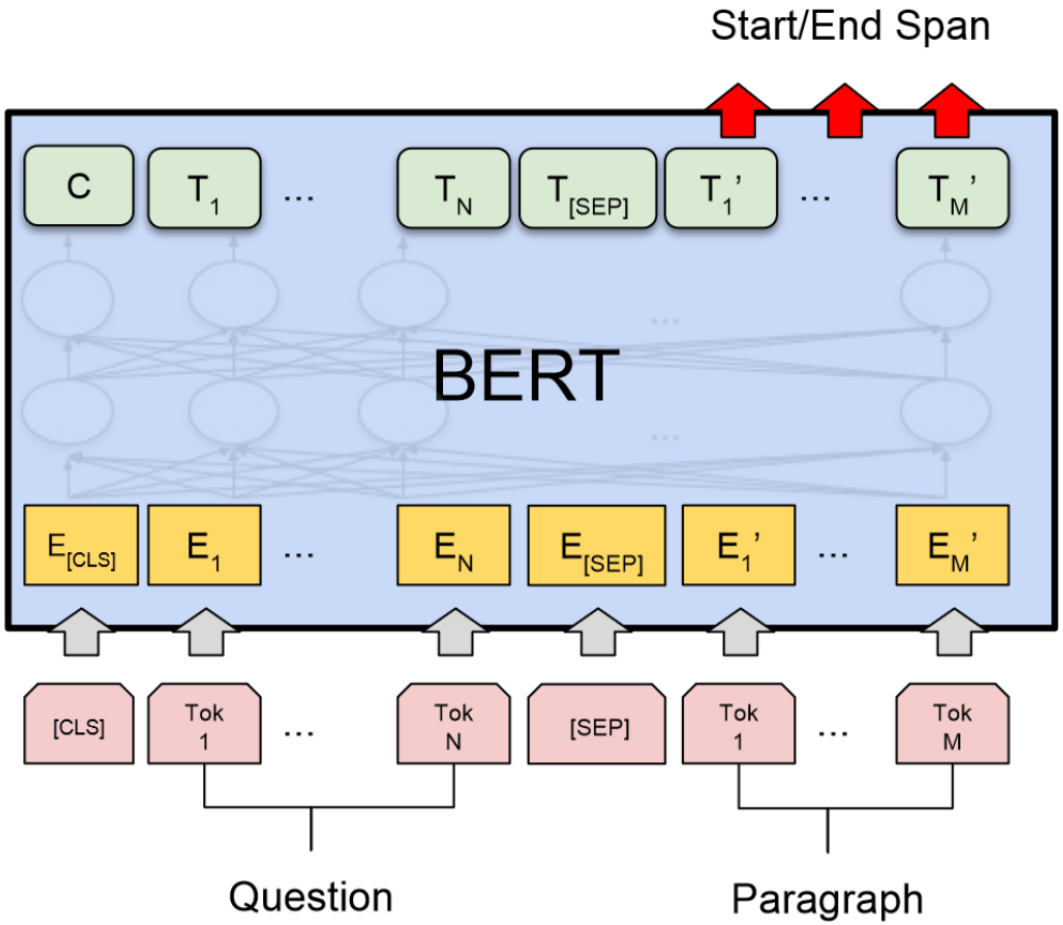
Sentence classification using BERT – sentiment, grammatical correctness



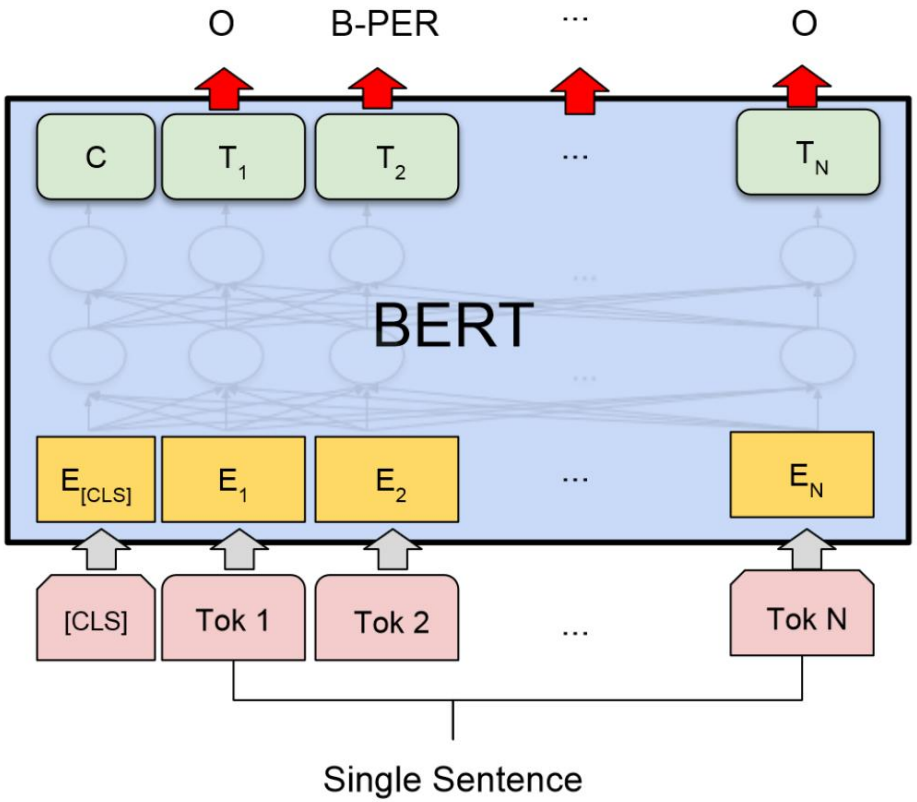
Two sentence classification using BERT-inference



Questions and answers with BERT

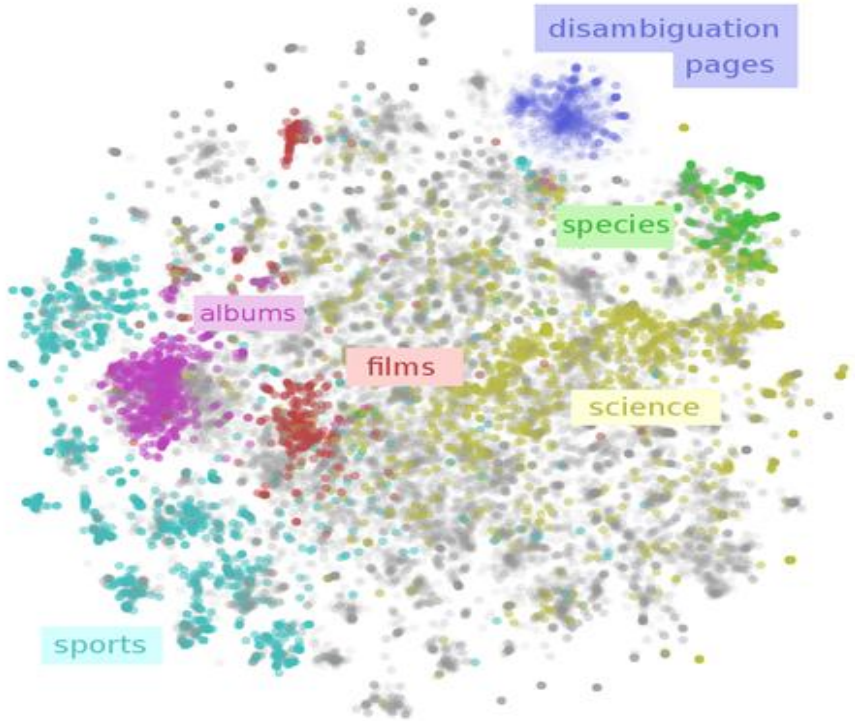


Sentence tagging with BERT-
NER, POS tagging, SRL



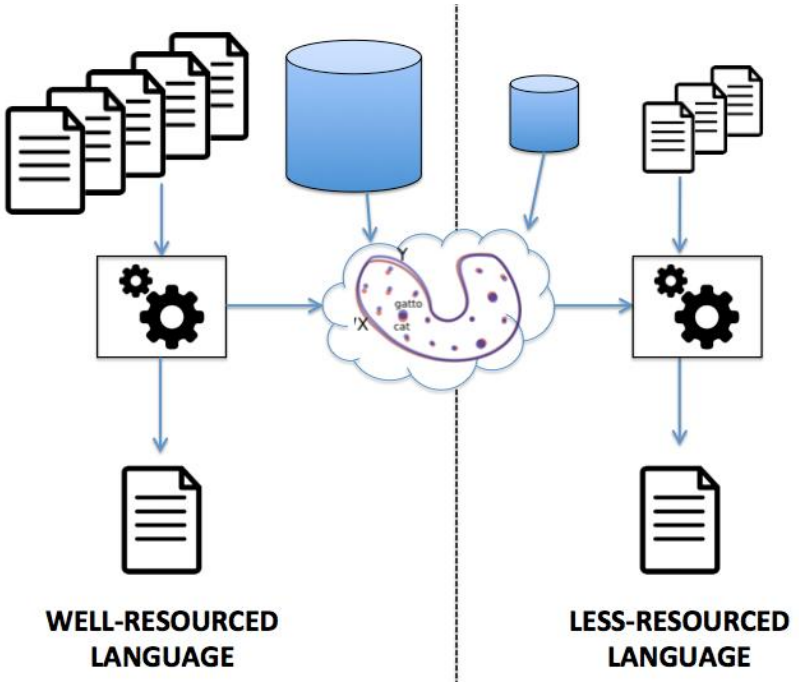
Cross-lingual embeddings

- mostly, embeddings are trained on monolingual resources
- words of one language form a cloud in high-dimensional space
- clouds for different languages can be aligned
  - $W S \approx E$ or
  - $W_1 S \approx W_2 E$



Cross-lingual model transfer based on embeddings

- Transfer of tools trained on mono-lingual resources



mostly not used anymore, superseded by multilingual LLMs

University of Ljubljana
Faculty of Computer and Information Science

Vocabulars in LLMs

- Tokenization depends on the dictionary
- The dictionary is constructed statistically (SentencePiece algorithm)

- Sentence: “Letenje je bilo predmet precej starodavnih zgodb.”
- SloBERTa:

'▁Le', 'tenje', '▁je', '▁bilo', '▁predmet', '▁precej', '▁staroda', 'vnih', '▁zgodb', '.’

- mBERT:

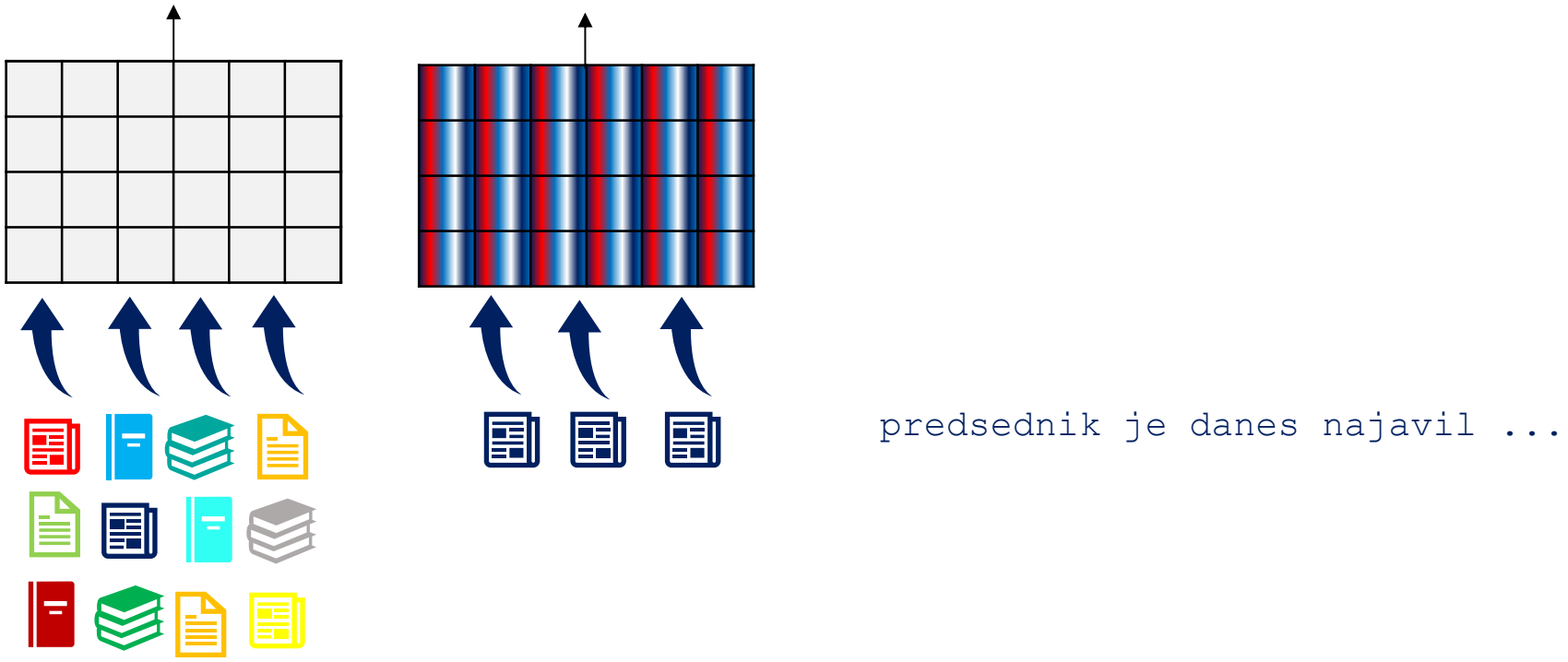
'Let', '##en', '##je', 'je', 'bilo', 'pred', '##met', 'pre', '##cej', 'star', '##oda', '##vnih', 'z', '##go', '##d', '##b', '.'

Ulčar, M., & Robnik-Šikonja, M. (2021) Training dataset and dictionary sizes matter in BERT models: The case of Baltic languages. In *International Conference on Analysis of Images, Social Networks and Texts* (pp. 162-172)

Multilingual LLMs

- Pretrained on multiple languages simultaneously
- multilingual BERT supports 104 languages by training on Wikipedia
- XLM-R was trained on 2.5 TB of texts
- allow cross-lingual transfer
- often solve the problem of insufficient training resources for less-resourced languages

Using multilingual models



Zero-shot transfer and few-shot transfer

What LLMs learn?

- We would like to travel to [MASK], ki je najlepši otok v Mediteranu.

SloBERTa: ..., Slovenija, I, Koper, Slovenia

CSE-BERT: Hvar, Rab, Cres, Malta, Brač

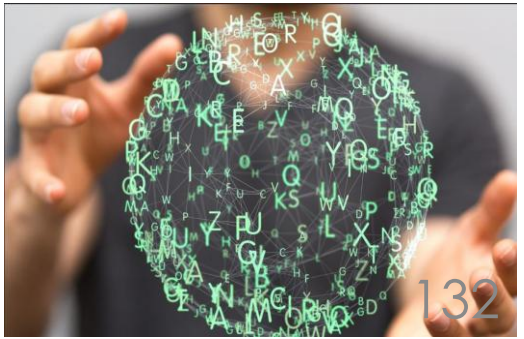
XLM-R: Mallorca, Tenerife, otok, Ibiza, Zadar

mBERT: Ibiza, Gibraltar, Tenerife, Mediterranean, Madeira

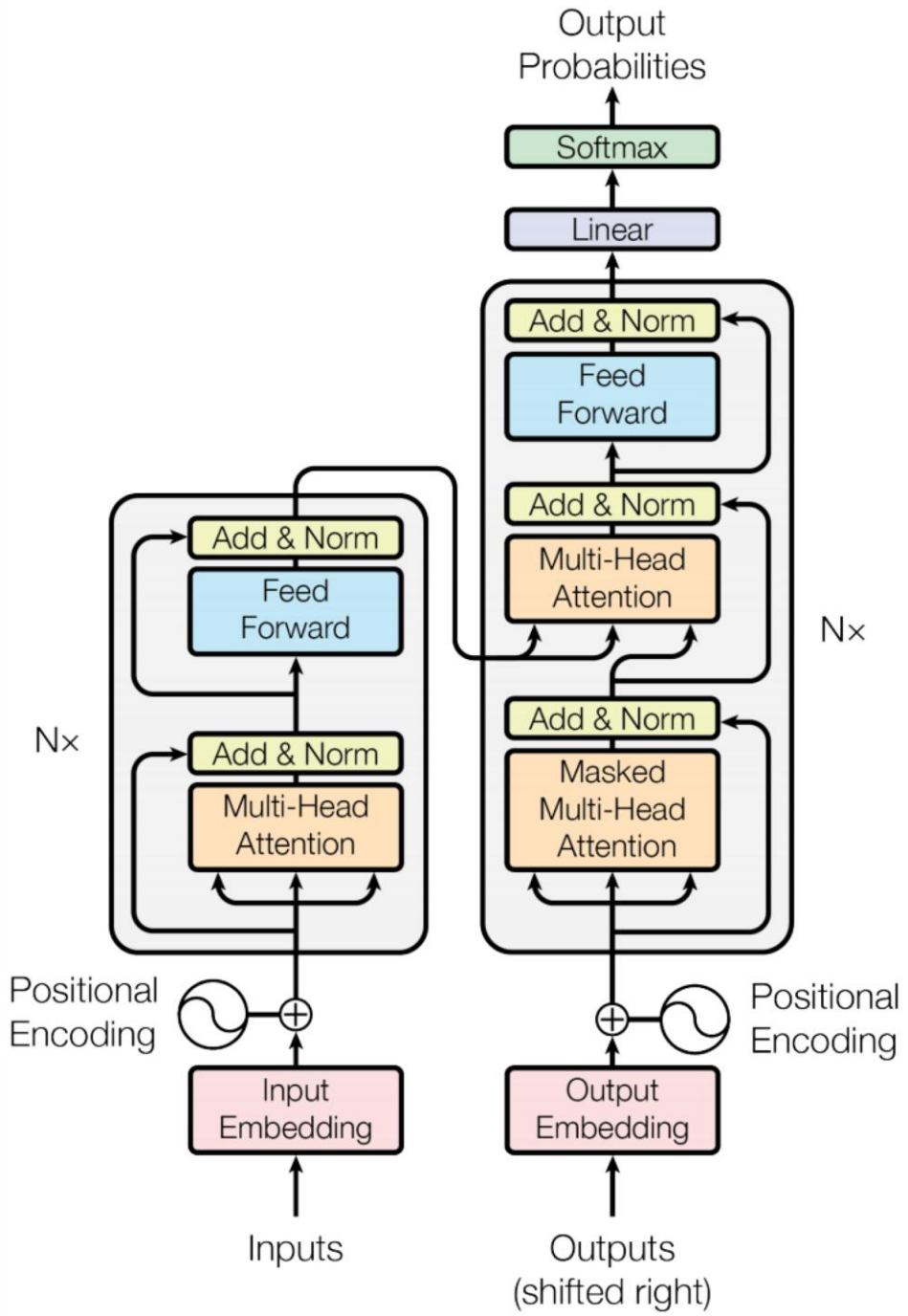
BERT (en): Belgrade, Italy, Serbia, Prague, Sarajevo

Embed all the things!

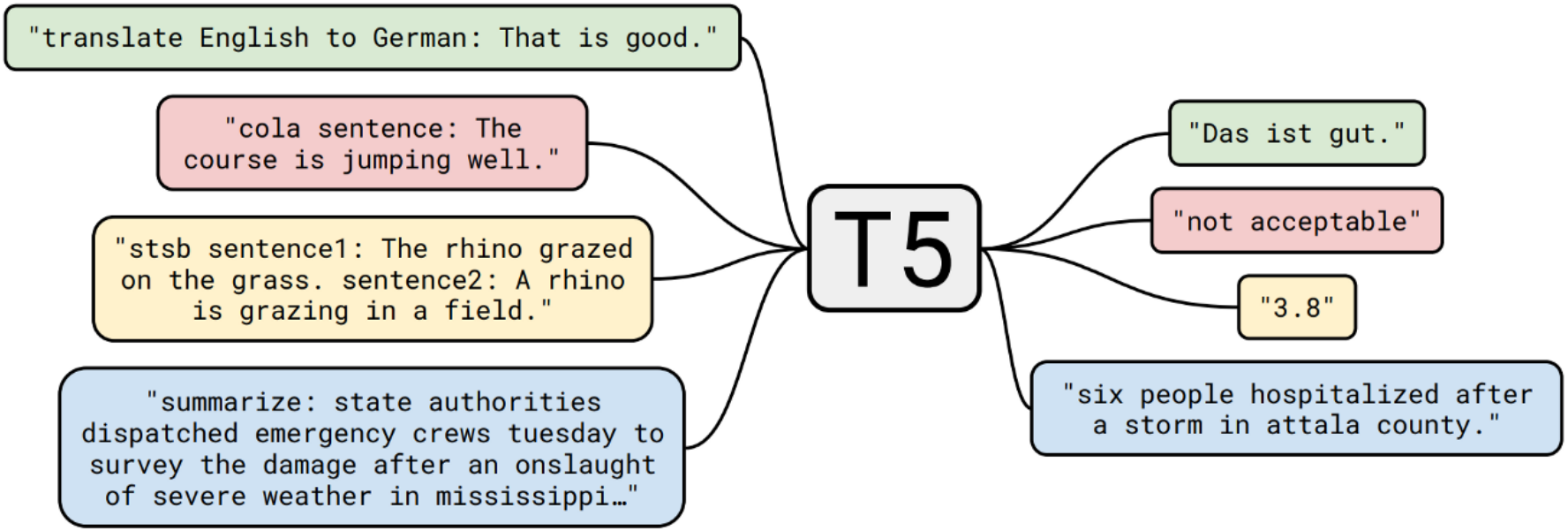
- Neural networks require numeric input
- Embedding shall preserve relations from the original space
- Representation learning is a crucial topic in nowadays machine learning
- Lots of applications whenever enough data is available to learn the representation
- In text, BERT-like models dominate for embeddings
- Similar ideas applied to texts, speech, graphs, electronic health records, relational data, time series, etc.



Transformer



T5 (Text-To-Text Transfer Transformer) models



Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y, Li, W. & Liu, P. J. (2020). Exploring the limits of transfer learning with a unified text-to-text transformer. *J. Mach. Learn. Res.*, *21*(140), 1-67.

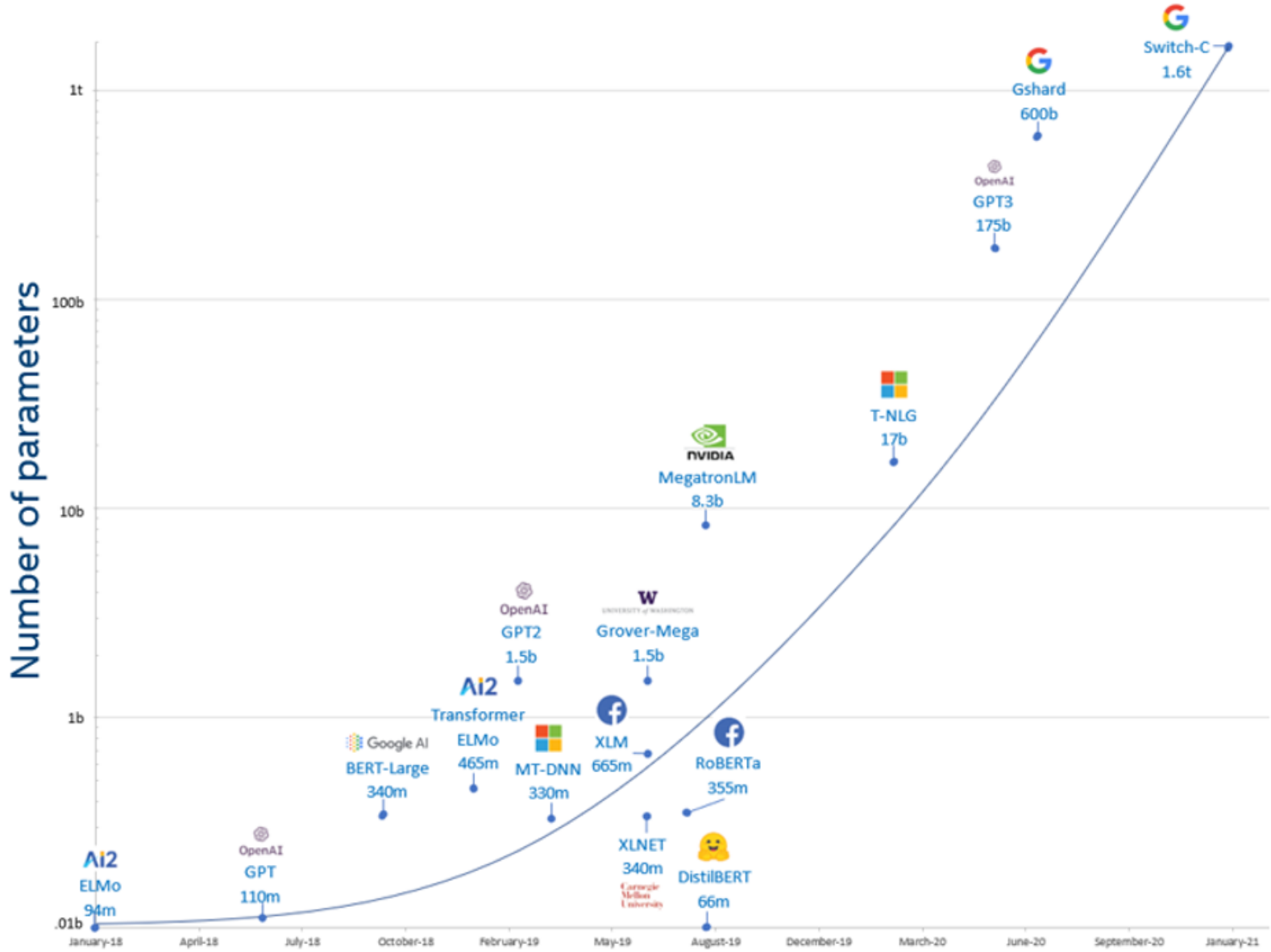
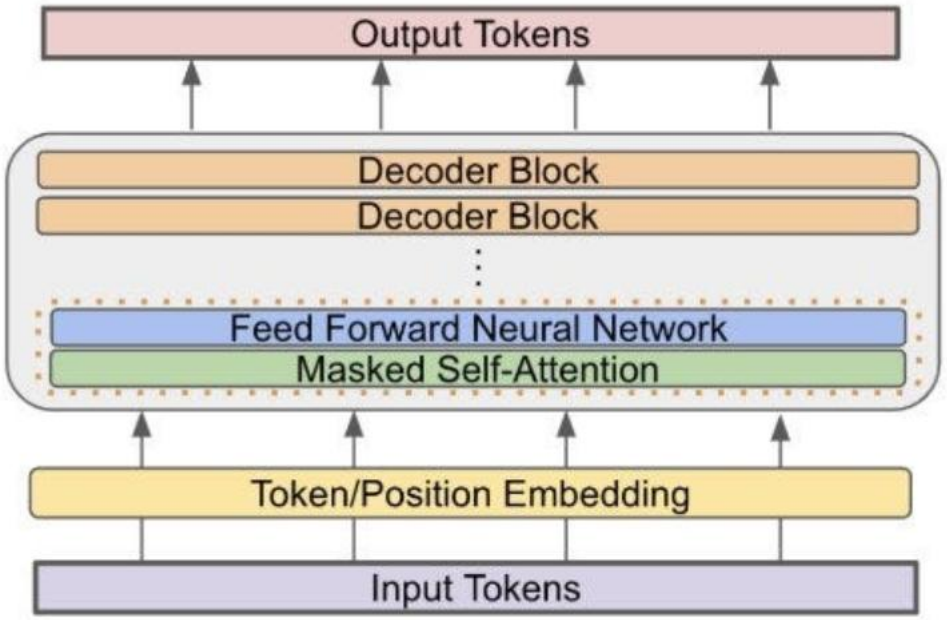


Figure 1: Exponential growth of number of parameters in DL models

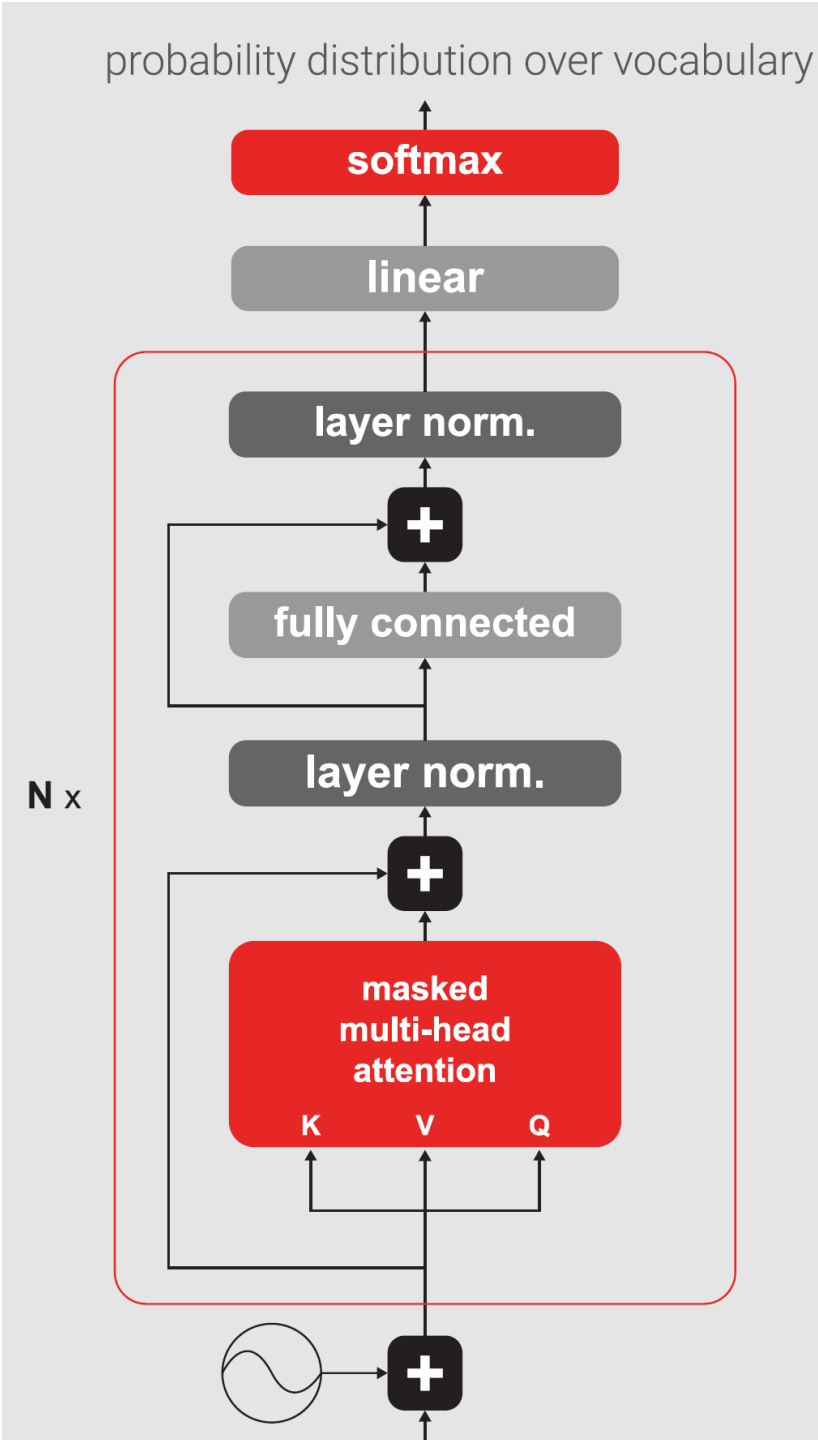
Decoder only models

- GPT, GPT-2, GPT-3, ChatGPT, GPT-4
- LLaMA, LLaMA-2, LLaMa-3
- MPT, Falcon
- Mistral and Mixtral
- OPT, Bloom
- Gemma and Gemini
- Olmo
- GaMS

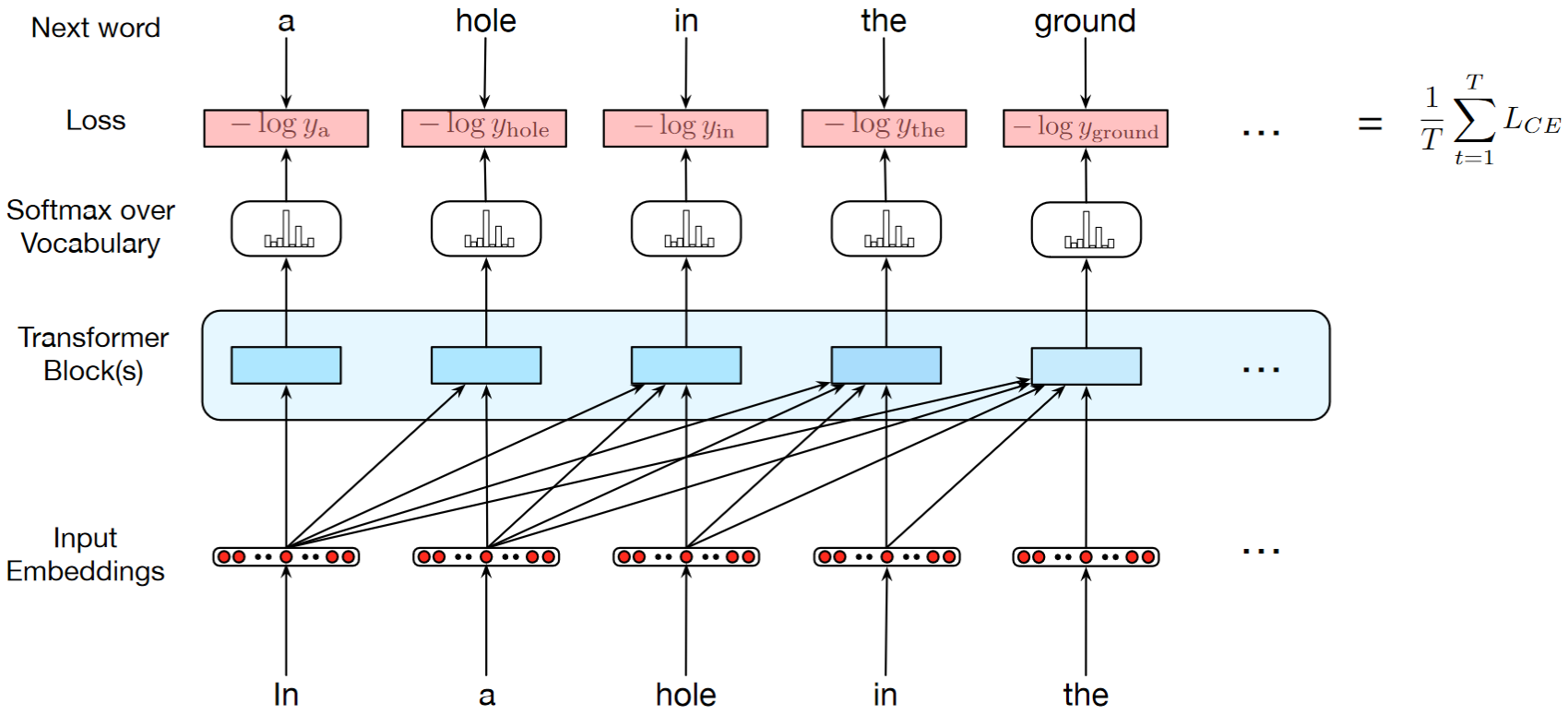


GPT family

- GPT: Generative Pre-trained Transformers
- use only the decoder part of transformer
- pretrained for language modeling (predicting the next word given the context)
- Potential shortcoming: unidirectional, does not incorporate bidirectionality
- “What are those?” he said while looking at my crocs.



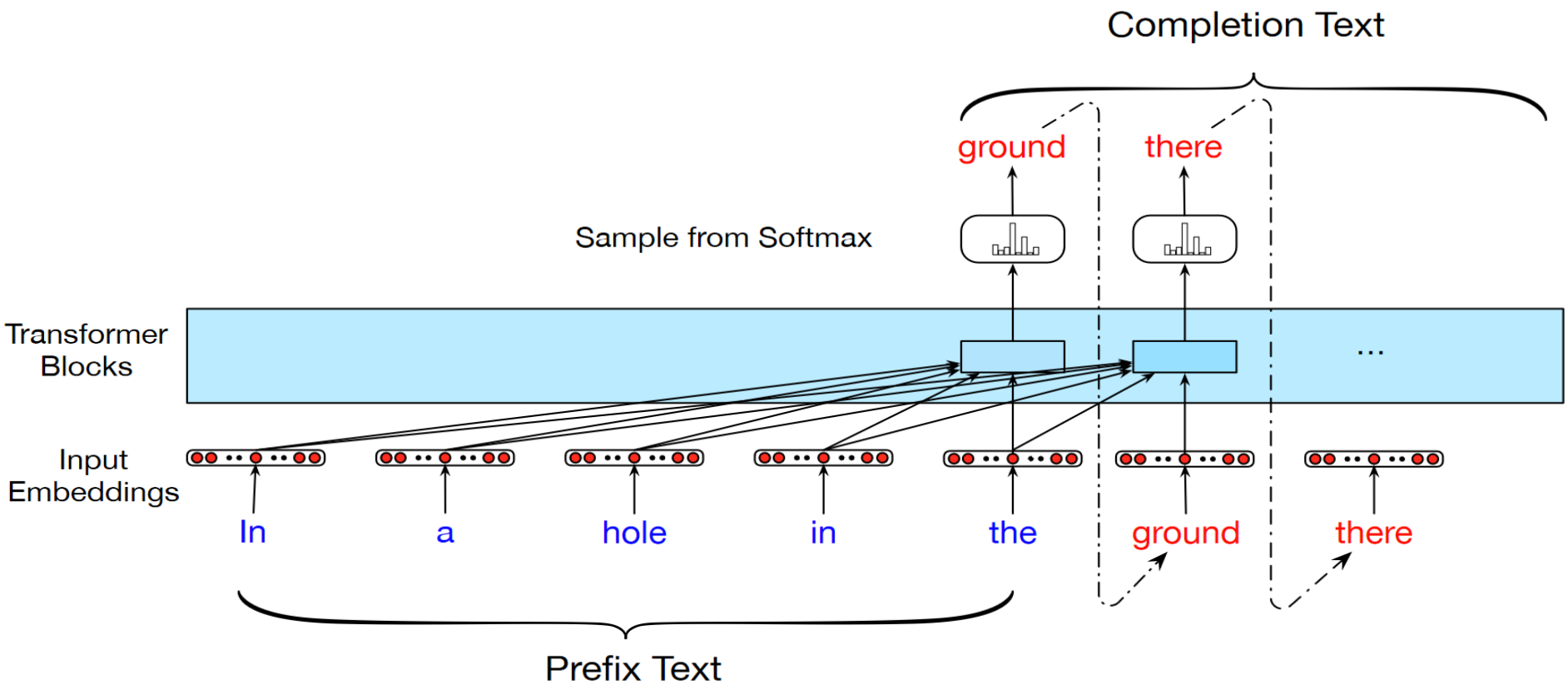
Transformer as a language model



- Can be computed in parallel

Autoregressive generators

- priming the generator with the context



- can be used also in summarization, QA and other generative tasks

GPT-2 and GPT-3

- few architectural changes, layer norm now applied to input of each subblock
- GPT-3 also uses some sparse attention layers
- more data, larger batch sizes (GPT-3 uses batch size of 3.2M)
- the models are scaled:

GPT-2:
48 layers, 25 heads
$d_m = 1600$, $d = 64$
context size = 1024
~ 1.5B parameters

GPT-3:
96 layers, 96 heads
$d_m = 12288$, $d = 128$
context size = 2048
~ 175B parameters

[1] Radford et al.: Language Models are Unsupervised Multitask Learners, 2019.
[2] Brown et al.: Language Models are Few-Shot Learners, 2020.

- see demos at https://transformer.huggingface.co/

Huge generative language models

- ChatGPT, OpenAI, Nov. 2022
  based on GPT-3.5 with additional training for dialogue
- uses instruction-following training and preference adaptation
- demo: https://chatgpt.com/
- huge public impact, disruptive for writing professions, learning, teaching, and scientific writing
- GPT-4, 5: even larger, allows longer context, image input and output, at least 8 x 220B parameters, MoE, reasoning
- LLaMa, Mistral, Gemma, DeepSeek, Qwen, EuroLLM, GaMS: smaller, open, very competitive LLMs

In-context learning

- GPT-2 and GPT-3 were the first models where one can ditch the “pre-train and fine-tune” training paradigm of GPT;
- GPT-2 explores unsupervised zero-shot learning, whereas in GPT-3 the authors expand the idea into in-context learning;
- Use text input to condition the model on task description and some examples with ground truth.
- Uses zero-shot learning, one-shot learning, few-shot learning (as many examples as they can fit into the context, usually 10-100)
- No gradient updates are performed.
- A sort of associative lookup

In-context learning

Zero-shot

The model predicts the answer given only a natural language description of the task. No gradient updates are performed.

```
1  Translate English to French:        ← task description
2  cheese =>                           ← prompt
```

One-shot

In addition to the task description, the model sees a single example of the task. No gradient updates are performed.

```
1  Translate English to French:        ← task description
2  sea otter => loutre de mer          ← example
3  cheese =>                           ← prompt
```

Few-shot

In addition to the task description, the model sees a few examples of the task. No gradient updates are performed.

```
1  Translate English to French:        ← task description
2  sea otter => loutre de mer          ← examples
3  peppermint => menthe poivrée        ←
4  plush girafe => girafe peluche      ←
5  cheese =>                           ← prompt
```

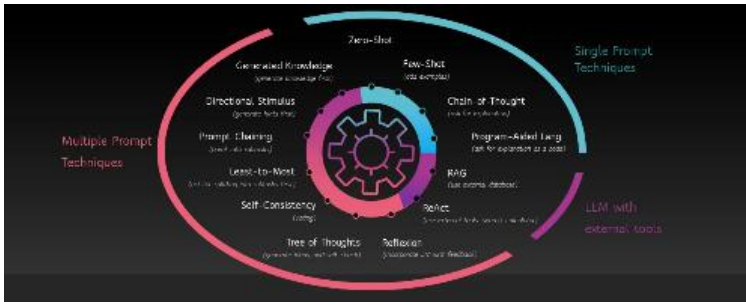
Figure source: Brown et al.: Language Models are Few-Shot Learners, 2020.

- GPT-3 is still a language model and can be used for text generation

- only 12% of respondents correctly classified this as not written by a human

Title: United Methodists Agree to Historic Split
Subtitle: Those who oppose gay marriage will form their own denomination
Article: **After two days of intense debate, the United Methodist Church has agreed to a historic split - one that is expected to end in the creation of a new denomination, one that will be "theologically and socially conservative," according to The Washington Post. The majority of delegates attending the church's annual General Conference in May voted to strengthen a ban on the ordination of LGBTQ clergy and to write new rules that will "discipline" clergy who officiate at same-sex weddings. But those who opposed these measures have a new plan: They say they will form a separate denomination by 2020, calling their church the Christian Methodist denomination.**
The Post notes that the denomination, which claims 12.5 million members, was in the early 20th century the "largest Protestant denomination in the U.S.," but that it has been shrinking in recent decades. The new split will be the second in the church's history. The first occurred in 1968, when roughly 10 percent of the denomination left to form the Evangelical United Brethren Church. The Post notes that the proposed split "comes at a critical time for the church, which has been losing members for years," which has been "pushed toward the brink of a schism over the role of LGBTQ people in the church." Gay marriage is not the only issue that has divided the church. In 2016, the denomination was split over ordination of transgender clergy, with the North Pacific regional conference voting to ban them from serving as clergy, and the South Pacific regional conference voting to allow them.

Prompt engineering (PE)



- Prompt engineering is a relatively new discipline for developing and optimizing prompts to efficiently use LLMs for a wide variety of tasks.
- Prompt engineering can help design robust and effective prompting techniques that interface with LLMs and other tools
- PE is a set of soft guidelines on how to design effective prompts
- PE is becoming an automatic approach to design effective prompts
- See https://www.promptingguide.ai/

Preference optimization of LLMs



- instruction-tuning
- Preference optimization
- ChatGPT uses RLHF (reinforcement learning with human feedback)

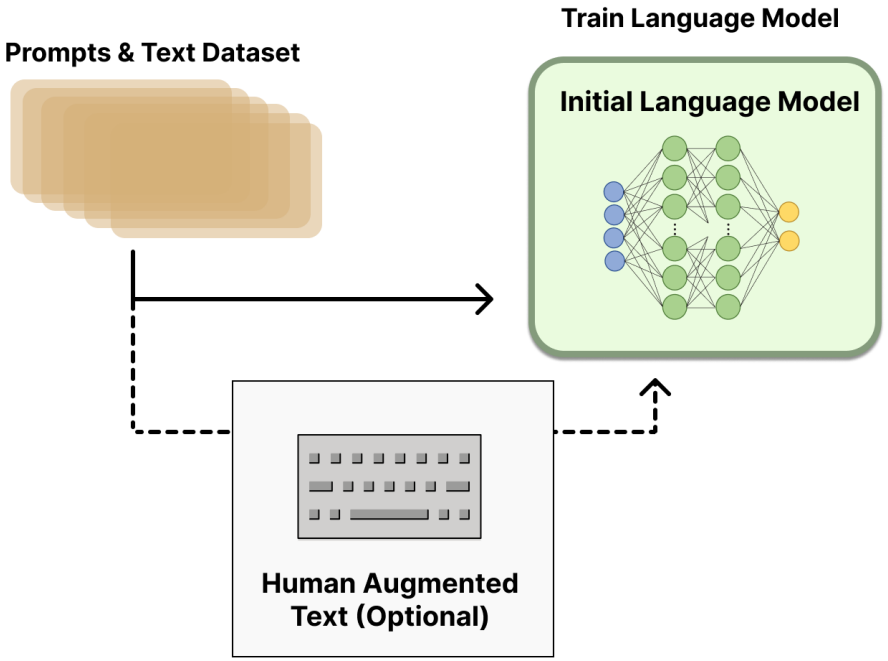
RLHF: idea

- Reinforcement Learning with Human Feedback
- A problem: Human feedback is not present during training
- Idea: Train a separate model on human feedback, this model can generate a reward to be used during training of LLM
- Three stages:
  1. Pretraining a language model (LM),
  2. Gathering data and training a reward model, and
  3. Fine-tuning the LM with reinforcement learning.

Nathan Lambert, Louis Castricato, Leandro von Werra, Alex Havrilla: "Illustrating Reinforcement Learning from Human Feedback (RLHF)", Hugging Face Blog, 2022.

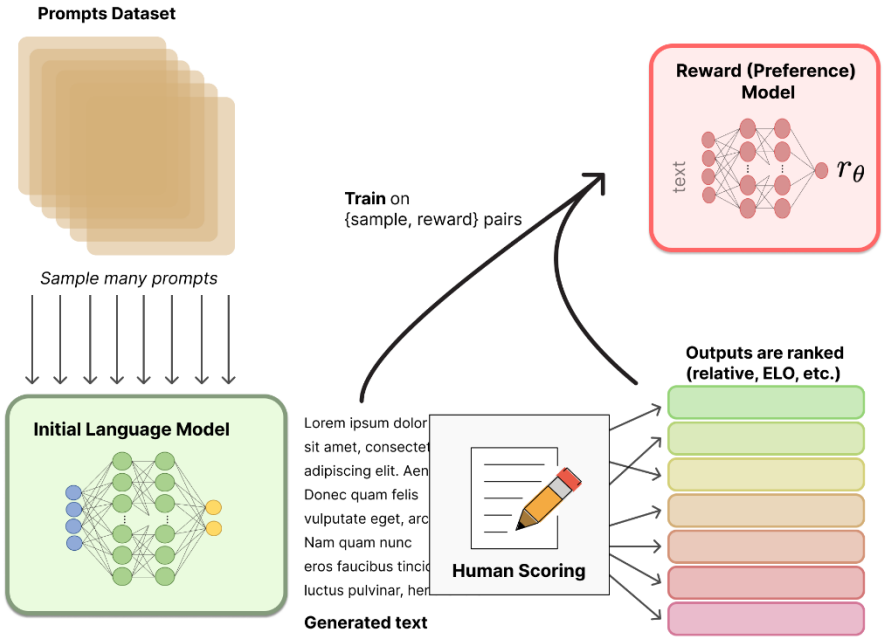
RLHF: the reward model 1/2

- input: a sequence of text, e.g., produced by LM and optionally improved by humans
- output: a scalar reward, representing the human preference of the text (e.g., a rank of the answer)
- the reward model could be an end-to-end LM, or the model ranks outputs, and the ranking is converted to reward



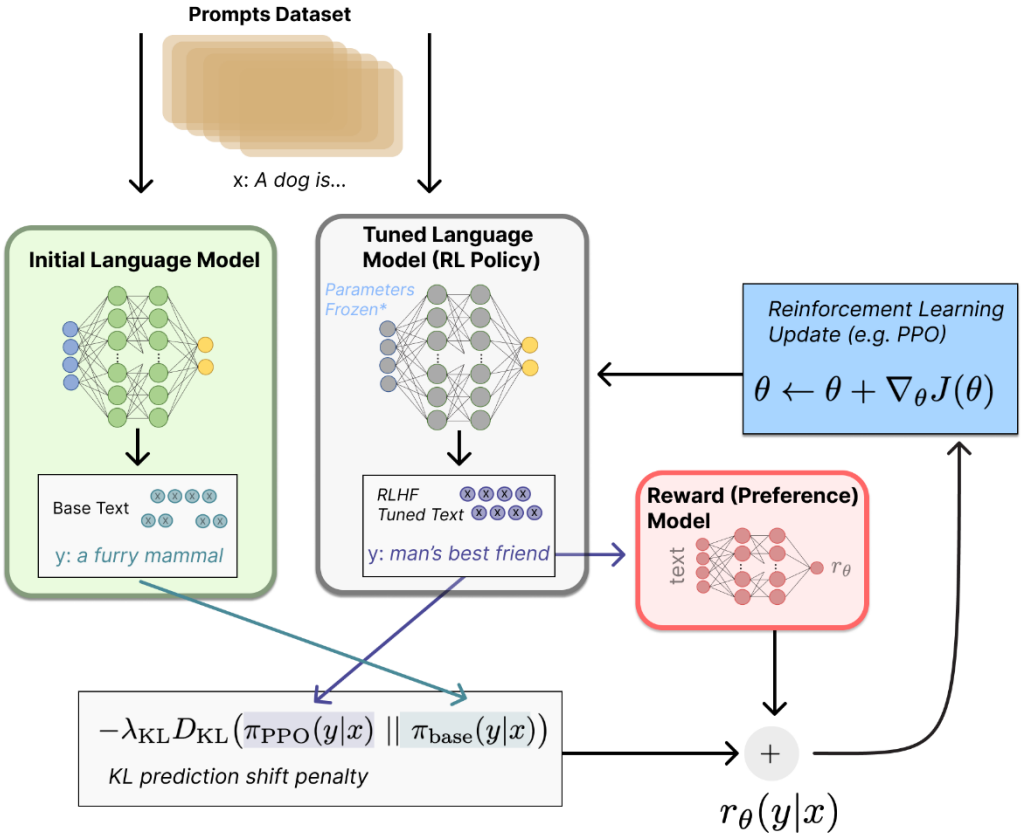
RLHF: the reward model 2/2

- the training dataset are pairs of prompts and (human improved) LM responses, e.g., 50k instances
- humans rank the responses instead of producing the direct reward as this produces better calibrated scores



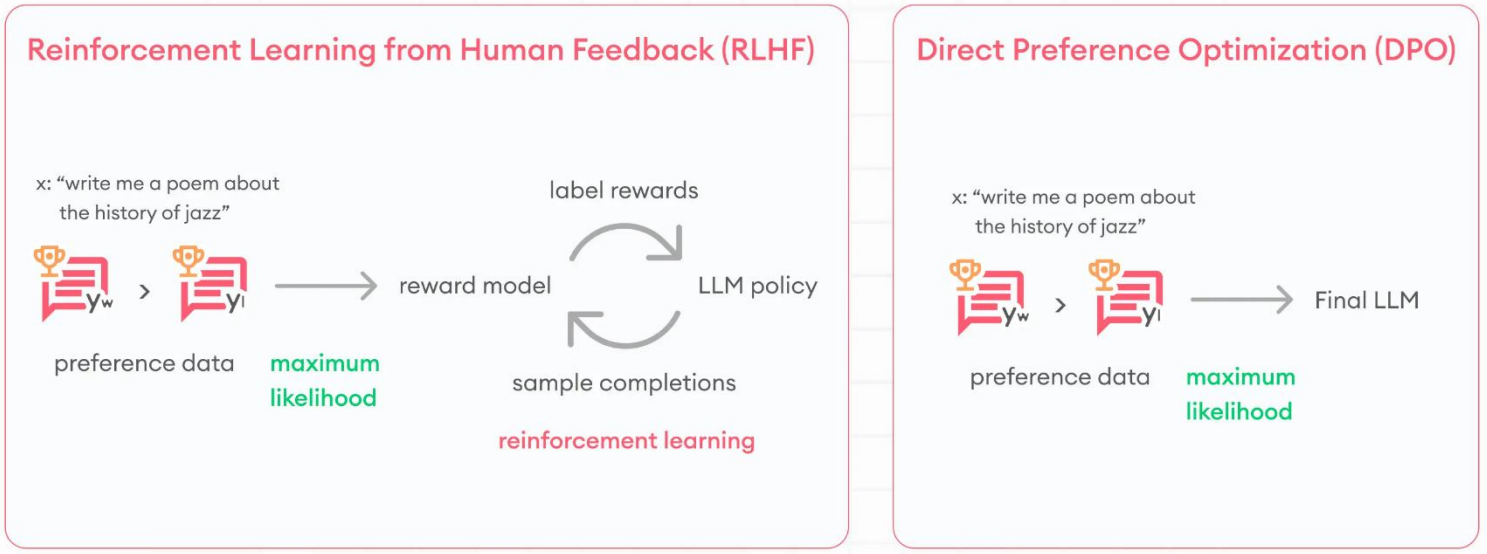
RLHF: fine-tuning with RL

- RL does not change all parameters, most of parameters are frozen
- the algorithm: Proximal Policy Optimization (PPO)



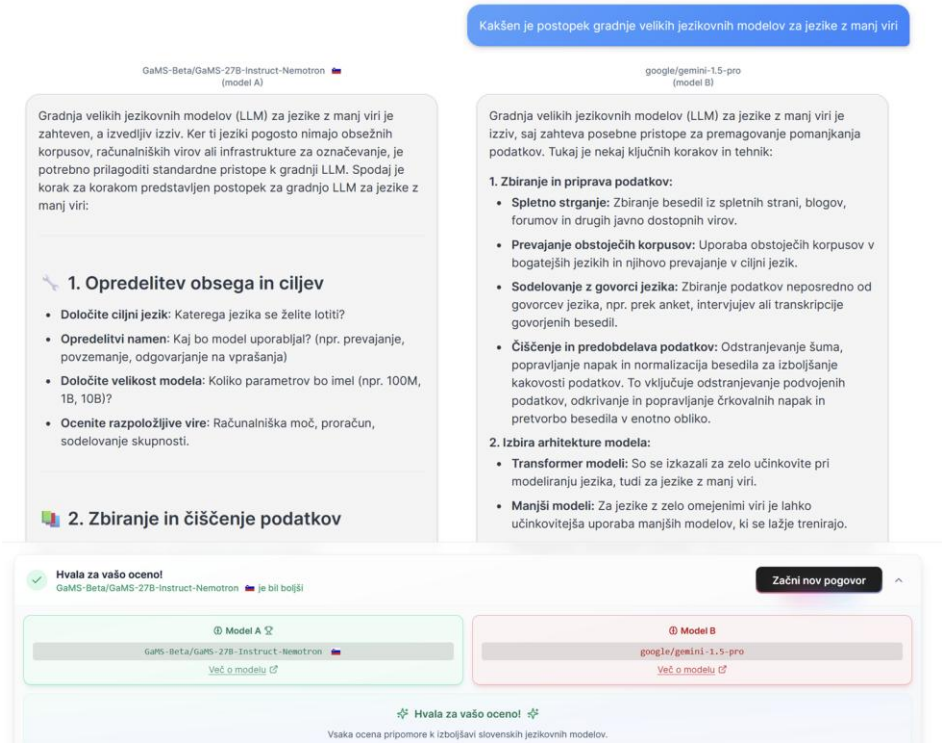
Direct preference optimization (DPO)

- DPO is a reinforcement learning approach, used as an alternative to classical RLHF with PPO
- Simpler, easier to collect data via LLM-Arena



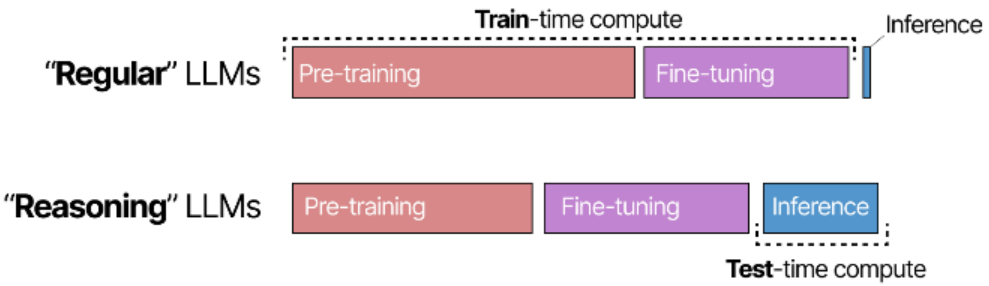
LLM Arena – feedback about model response

- A prompt is responded to by two models simultaneously
- The user chooses the better answer
- In the Slovenian LLM arena, the most successful models for Slovene are compared
- Collection and analysis of user responses
- First findings: GaMS is just a little behind the 1000x larger Gemini 2.5 Pro and is better than other models
- https://arena.cjvt.si/sl

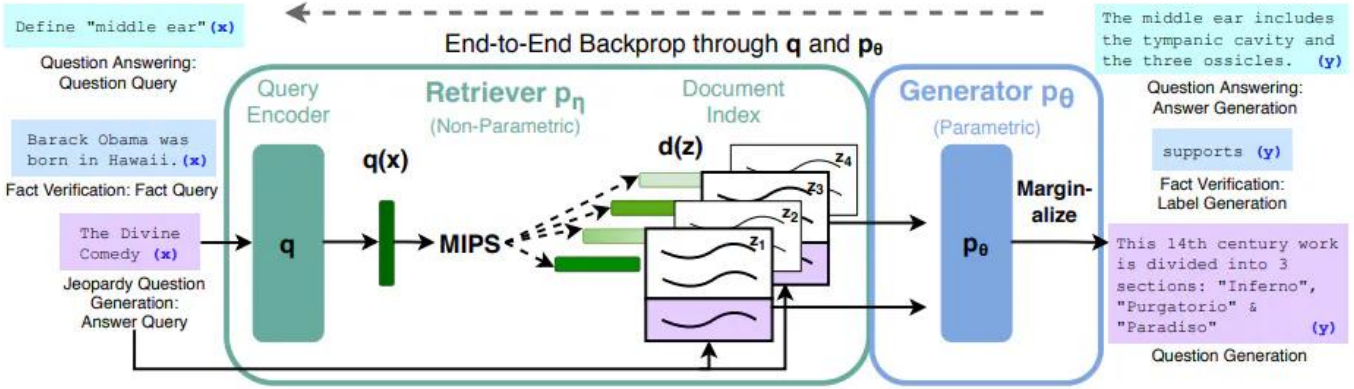


New ways to train and use LLMs

- Many prompt engineering techniques
- In-context learning: prepare context relevant for the given problem and is helpful in generating a useful response
- RAG, retrieval augmented generation
- Chain-of-thought reasoning
- Auto-GPT: breaks the goal in subgoals, prepares a sequence of actions for each subgoal, which are described and send to submodels or external programs
- Inference time reasoning
- Agent architectures



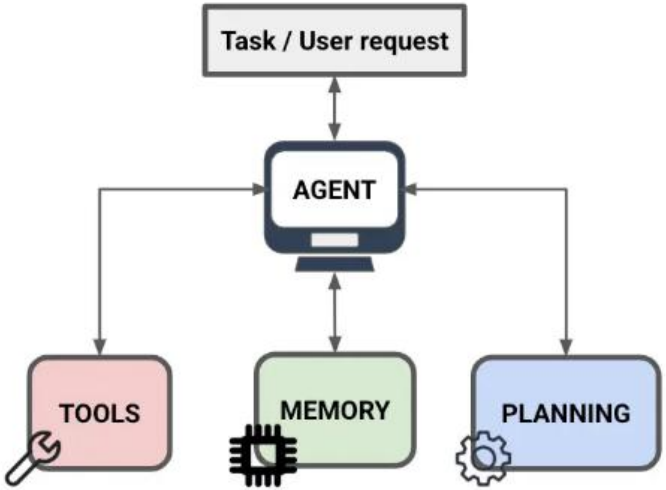
RAG details



Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., Küttler, H., Lewis, M., Yih, W.T., Rocktäschel, T. and Riedel, S., 2020. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in Neural Information Processing Systems*, *33*, pp.9459-9474.

LLM Agents

- A relatively new approach utilizing LLMs for various complex tasks



GaMS – generativni model za slovenščino

PoVeJMo
SLOVENSKI
VELIKI JEZIKOVNI MODEL

ODPRTODOSTOPEN

- Vaši podatki ostanejo pri vas

- Učenje verzije 9B: 3 dni na 32 GPU vozliščih (128 GPU) na Leonardo Booster HPC
- Sledenje navodilom: 25.000 primerov, prilagoditev parametrov, 1 dan, 1 GPU vozlišče

GaMS:
Generativni model za slovenščino
Generative Model for Slovene

VEČ RAZLIČIC

- 2B
- 9B
- 27B

DODATNO UČEN NA SLOVENSKIH BESEDILIH

- Osnova: Gemma 2
- Trenutna različica: 8 milijard slovenskih “žetonov” (6 milijard besed)

NAJBOLJŠI ODPRT MODEL ZA SLOVENŠČINO

- SloBench
- Nadgradnja – več besedil

UL | FRI

GAMS :” 9B

- Based on Gemma 2 9B.
- Pretraining on Slovene data using 15B tokens (9B Slovene tokens)
- Pretraining: 3 days on 32 GPU nodes (128 GPUs) on Leonardo Booster super-computer
- Instruction following: 25.000 instances, updating all parameters, 1 day, 1 GPU node
- The most successful open model for Slovene
- Evaluations:
  - SloBENCH : SuperGLUE, machine translation
  - SlovenianEval

Rank	Title	Authors and Affiliations	Average
1	GaMS-27B	Domen Vreš, FRI	0.7601
2	PrešernGPT 0.1	Žan Knafelc	0.7568
3	Gemma 2 27B	Google, Domen ...	0.7546
4	GaMS-9B	Domen Vreš, FRI	0.7309
5	GaMS-9B-Instru...	Domen Vreš, FRI	0.6997
6	Gemma 2 9B	Google, Domen ...	0.6980
7	SlovenianGPT-C...	Domen Vreš, FRI	0.6436
8	CroSloEngual BE...	Aleš Žagar, FRI ...	0.6078
9	GaMS-1B-Chat-f...	Domen Vreš, FRI	0.5806
10	OPT_GaMS-1B-...	Domen Vreš, FRI	0.5645

https://povejmo.si

Gemma 2 -> GaMS:”

- Phase 1: language alignment
  - parallel English-Slovene corpora
- Phase 2: high-quality Slovene data
  - Wikipedia + MetaFida + KAS
  - without: web-crawl, tweets
  - news sorted by increasing time

GaMS: “

Evaluation: SloBench - SuperGLUE

PoVeJMo
SLOVENSKI
VELIKI JEZIKOVNI MODEL

Leaderboard

Slovene SuperGLUE 1.0

Related Leaderboards

Slovene SuperGLUE (v1)

Description

This leaderboard is a Slovene (translated) version of an existing English version. The task initially consists of multiple tasks, which test a system's ability to understand and reason about texts in Slovene. All the tasks should be solvable by most college-educated Slovene speakers.

The original SuperGLUE benchmark consists of 8 tasks, while 6 tasks are included in the Slovene evaluation. These tasks are BoolQ, CB, COPA, MultiRC, RTE and WSC.

Leaderboard Editors

Aleš Žagar, ales.zagar@fri.uni-lj.si
Slavko Žitnik, slavko@zitnik.si

https://slobench.cjvt.si/

Leaderboard Submissions

Rank	Title	Authors and Affiliations	Average	BoolQ Accuracy	CB Accuracy	CB F1 Score	CB Average	COPA Accuracy	MultiRC EM	MultiRC F1a Score	MultiRC Average	RTE Accuracy	WSC Accuracy	Tag
1	GaMS-27B	Domen Vreš, FRI	0.7601	0.8333	0.6440	0.5864	0.6152	0.9540	0.3904	0.7504	0.5704	0.7931	0.7945	
2	PrešernGPT 0.1	Žan Knafelc	0.7568	0.8333	0.8520	0.5868	0.7194	0.9740	0.4985	0.8061	0.6523	0.8276	0.5342	
3	Gemma 2 27B	Google, Domen ...	0.7546	0.8333	0.6680	0.5972	0.6326	0.9140	0.4174	0.7295	0.5735	0.8276	0.7466	
4	GaMS-9B	Domen Vreš, FRI	0.7309	0.7000	0.8400	0.7955	0.8178	0.9000	0.3243	0.6551	0.4897	0.7931	0.6849	
5	GaMS-27B-Instr...	Domen Vreš, FRI	0.7038	0.8333	0.7200	0.6322	0.6761	0.9400	0.0511	0.5813	0.3162	0.7931	0.6644	
6	GaMS-9B-Instru...	Domen Vreš, FRI	0.6997	0.8000	0.7960	0.7128	0.7544	0.8140	0.0721	0.6174	0.3447	0.7931	0.6918	
7	Gemma 2 9B	Google, Domen ...	0.6980	0.8333	0.8240	0.5683	0.6962	0.8700	0.2282	0.5310	0.3796	0.7241	0.6849	

Injecting knowledge into large generative models



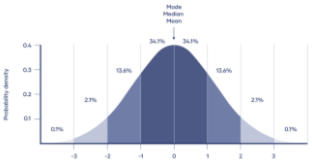
- Slovenian dataset for Common-Sense Reasoning KE-WS
- Slovenian dataset for Natural Language Inference SI-NLI
- Prepared datasets for logical reasoning and mathematics
- Prepared a Slovene safety dataset
- In working: A dataset with ethical norms
- Methodology for the transfer of linguistic and lexicographic knowledge to LLMs
- To be included in the next version of GaMS models
- New applications based on the GaMS models, e.g., grammar error correction, machine translation, summarization and simplification of texts, etc.
- GaMS 3.0 is almost ready (based on Gemma 3.0)

HPC related lessons

- Building LLM from scratch is computationally extremely costly
  - LLaMa-2 7B > 360,000 GPU hours,
  - SLING tender: Vega GPU offers 80,000 node hours = 320.000 GPU hours annually
  - GaMS 3: a combination of Leonardo Booster, Nvidia Sovereign LLMs support and UL FRI FRIDA cluster
- Building an LLM by further training an existing LLM:
  - Data quantity issues (web crawl, collection, machine translation): Hoffman scaling laws
  - Data quality issues
  - Model vocabulary issues,
  - Platform issues
  - Model parallelization (within 1 GPU node, i.e. 4 or 8 A/H100),
  - Data parallelization (across nodes), reduces inter-node communication
  - Support for bf16 (1b sign, 8b exponent, 7b significand)
  - QLORA: NF4, 1 node, needs instruction-tuning datasets,
  - Building LLM requires lots of HPC expertise
- Using platforms like the NVidia NeMo framework:
  - +speed
  - +easier parallelization
  - -less models available
  - -needs model and code tweaking
- In deployment: squeeze everything on a single computational node (if possible)



Statistical evaluation of LLMs' results



- LLMs can answer any question, even questions about complex problems, such as word sense definition

"You are a lexicographer who is writing definitions for a general dictionary of Slovene. The dictionary is targeted at native speakers of Slovene. Here are some sample definitions for a selection of nouns/adjectives/verbs/adverbs:
"file.txt"
"Write a dictionary definition for the selected sense of the noun/adjective/verb/adverb X. Below is some contextual information. There is only one sense, therefore write only one definition. Write the definition only; do not add any justifications, and do not repeat the provided contextual information."
…

- We have no guarantee that the answers are correct
- LLMs do not abstain; we almost always get an answer
- To statistically analyze the problem and usability of LLMs (either prompts or fine-tuned LLMs) we typically do a preliminary test
- To trust the results and use them, LLMs' results have to be verified first

Verification of LLMs outputs

- Using LLMs for complex questions:
  - Ask LLM
  - Verify the answers:
  - If the accuracy is high, use LLM; otherwise, this is senseless, revise
- Three verification paths:
  1. Comparison with human gold-standard answers
  2. Human verification of LLM's answers
  3. LLM-as-a-judge



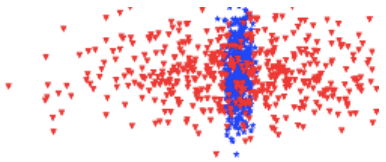
Verification 1: Comparison with human gold-standard answers

- It might be time-consuming or expensive to get the gold-standard answers
- For classification, use metrics such as accuracy, F_1, precision, recall, etc.
- In contrast to classification, the comparison is more difficult for generated texts, but methods exist
  - n-gram based matching (BLEU, ROUGE, etc),
  - sentence encoders and cosine distance
  - LLM as a judge (stronger LLM, has to be verified first, potentially circular)

Verification 2:
Human verification of LLM's answers

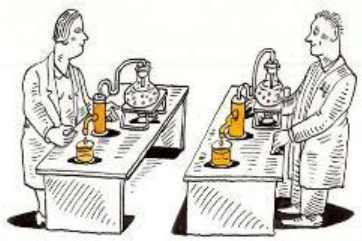
- Humans check the answers of LLMs (correctness, style, format, consistency)
- Note: Checking the answers is a weaker form of verification than producing the same or similar answers (the priming effect).

Reliability



- **Variance** in answers (humans' or LLMs') is always present
- Humans: different background, education, political orientation, sense of humour, mood, etc.
- LLMs: different prompts, different LLMs, temperature parameters, etc.
- If possible, compute
  - Inter-annotator agreement
  - Fleiss' kappa,
    Landis and Koch scale: $\kappa > 0.80$ (almost perfect), $0.60 < \kappa \leq 0.80$ (substantial), etc.
  - Annotator's self-agreement
- To improve reliability: repeat and average, show variance

Reproducibility

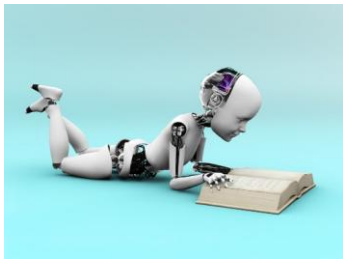


- **Reproducibility** concerns:
  - The research in LLMs is very fast
  - Commercial LLMs have a life span of around a year
  - However, there are many (less capable) open-source models

Where LLMs go?



- thousand of new applications
- multimodal models
- Knowledge injection, e.g., BioRDF
- external knowledge incorporation, e.g., ToolFormer
- explanation and ethical use



06292v2 [cs.AI] 15 Aug 2024

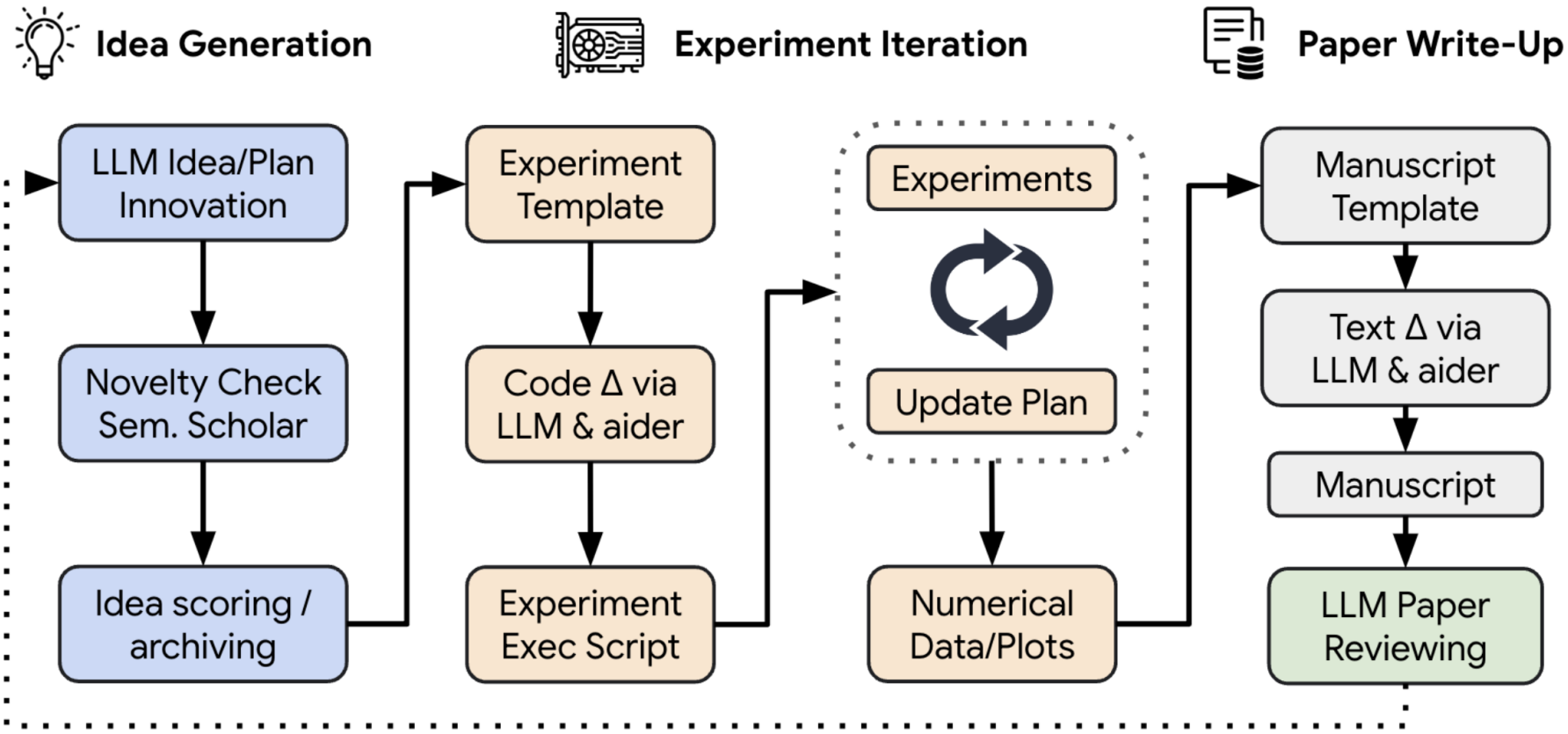
UL FRI

2024-8-16

The AI Scientist: Towards Fully Automated Open-Ended Scientific Discovery

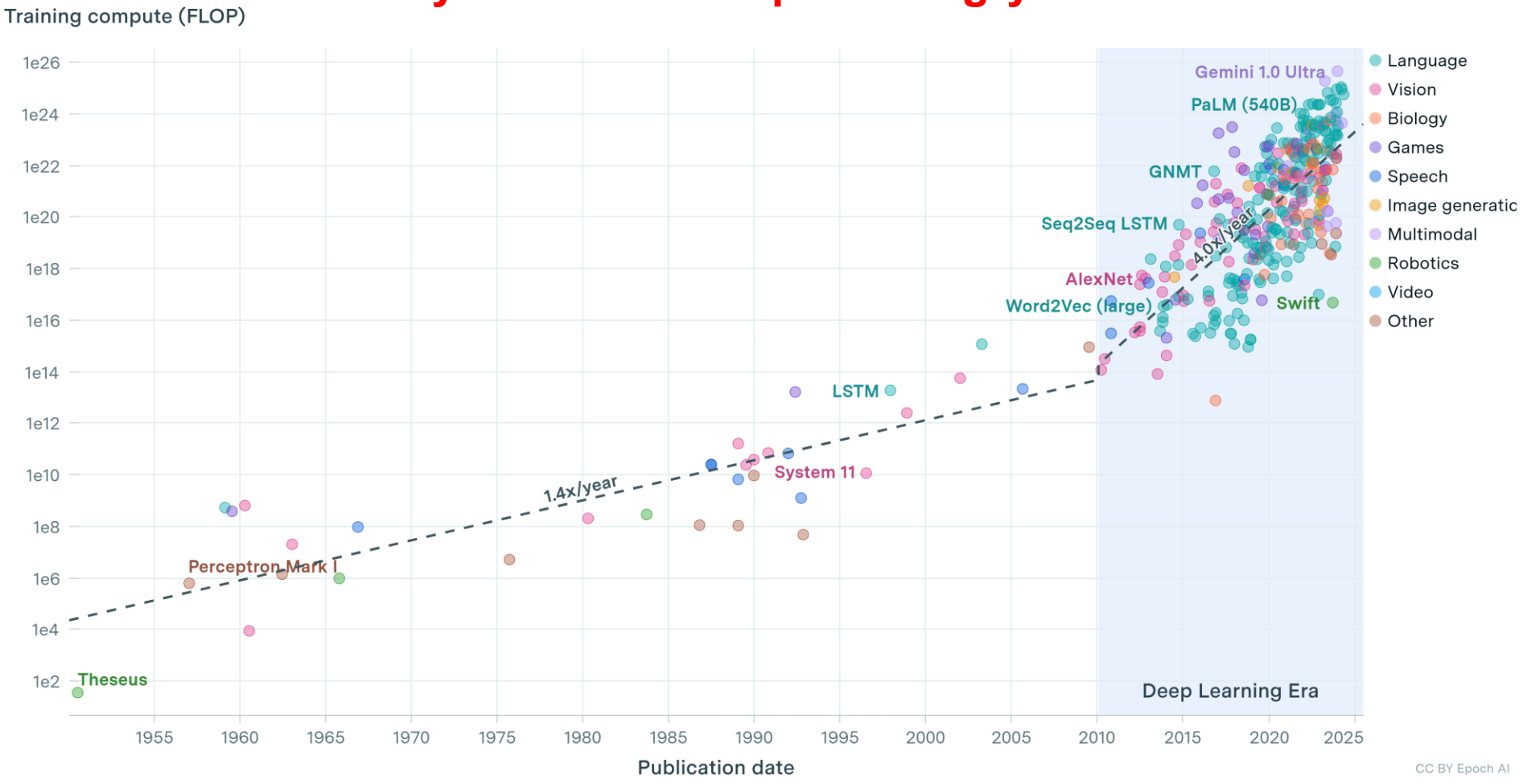
Chris Lu[1,2,*], **Cong Lu**[3,4,*], **Robert Tjarko Lange**[1,*], **Jakob Foerster**[2,†], **Jeff Clune**[3,4,5,†] **and David Ha**[1,†]
[*]Equal Contribution, [1]Sakana AI, [2]FLAIR, University of Oxford, [3]University of British Columbia, [4]Vector Institute, [5]Canada CIFAR AI Chair, [†]Equal Advising

One of the grand challenges of artificial general intelligence is developing agents capable of conducting scientific research and discovering new knowledge. While frontier models have already been used as aides to human scientists, e.g. for brainstorming ideas, writing code, or prediction tasks, they still conduct only a small part of the scientific process. This paper presents the first comprehensive framework for fully *automatic scientific discovery*, enabling frontier large language models (LLMs) to perform research independently and communicate their findings. We introduce THE AI SCIENTIST, which generates novel research ideas, writes code, executes experiments, visualizes results, describes its findings by writing a full scientific paper, and then runs a simulated review process for evaluation. In principle, this process can be repeated to iteratively develop ideas in an open-ended fashion and add them to a growing archive of knowledge, acting like the human scientific community. We demonstrate the versatility of this approach by applying it to three distinct subfields of machine learning: diffusion modeling, transformer-based language modeling, and learning dynamics. Each idea is implemented and developed into a full paper at a meager cost of less than $15 per paper, illustrating the potential for our framework to democratize research and significantly accelerate scientific progress. To evaluate the generated papers, we design and validate an automated reviewer, which we show achieves near-human performance in evaluating paper scores. THE AI SCIENTIST can produce papers that exceed the acceptance threshold at a top machine learning conference as judged by our automated reviewer. This approach signifies the beginning of a new era in scientific discovery in machine learning: bringing the transformative benefits of AI agents to the *entire* research process of AI itself, and taking us closer to a world where *endless affordable creativity and innovation* can be unleashed on the world's most challenging problems. Our code is open-sourced at https://github.com/SakanaAI/AI-Scientist.


Idea Generation
Experiment Iteration
Paper Write-Up
LLM Idea/Plan Innovation
Novelty Check Sem. Scholar
Idea scoring / archiving
Experiment Template
Code Δ via LLM & aider
Experiment Exec Script
Experiments
Update Plan
Numerical Data/Plots
Manuscript Template
Text Δ via LLM & aider
Manuscript
LLM Paper Reviewing

UL FRI

AI systems are compute hungry



LLMs are data hungry

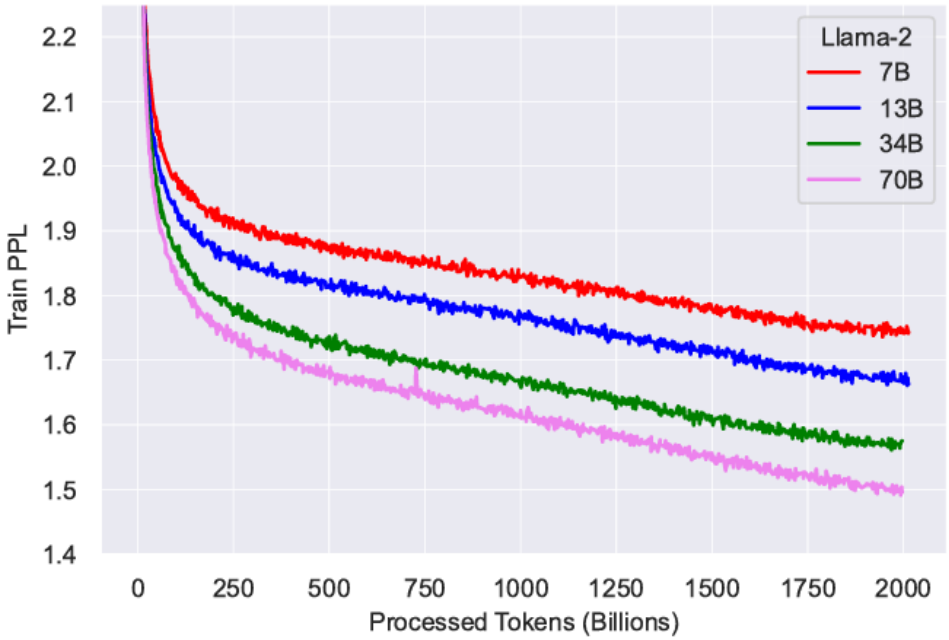


Figure 5: Training Loss for Llama 2 models. We compare the training loss of the Llama 2 family of models. We observe that after pretraining on 2T Tokens, the models still did not show any sign of saturation.

Ethical dilemmas of using AI

- bias and fairness: social and data biases
- quality of training data
- privacy and security of data
- responsibility of owners and developers
- transparency and explainability
- updating models (legal, content-based)

Responsible use of AI in research

EU prepared a draft of the guidelines

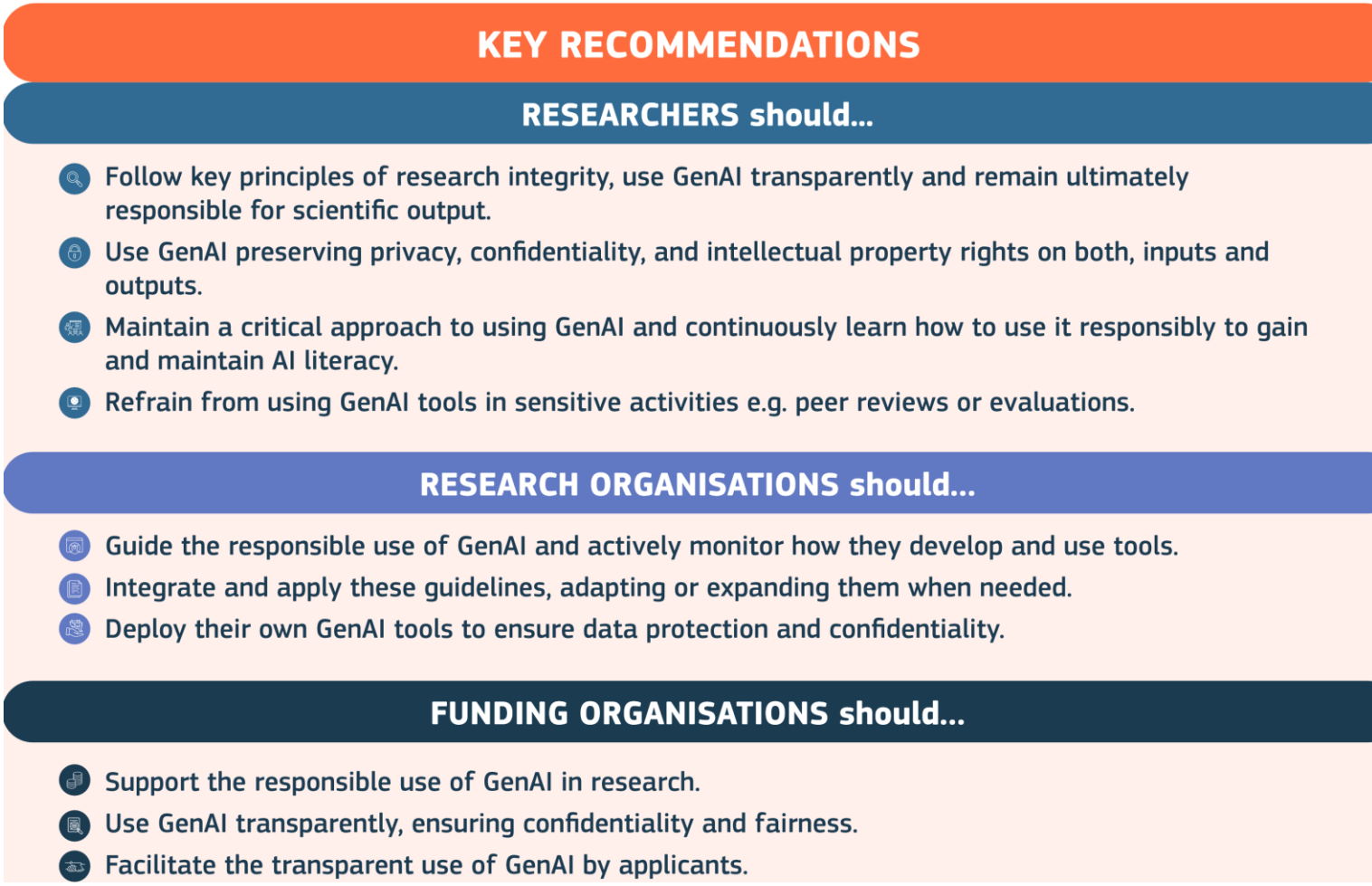
1. The researcher remains ultimately responsible for scientific output.
2. The researcher ensures fully transparency in the use of Generative AI (including tool, version, prompts, etc.).
3. The researcher pays attention to privacy and the sensitive/protected information they could share with AI tools.
4. As in their regular research activities, researchers must respect legislation applicable when using Generative AI.
5. Researchers need to learn and be trained regularly in the proper use of Generative AI tools in order to maximize the benefits from these tools.

UL FRI

EU guidelines for responsible use of AI in research

Live document

EU AI Act



EU AI Act

- Non-commercial research is exempt from regulation
- For research that can lead to marketable products, AI systems are classified into 4 categories:
  - **Unacceptable risk**: AI apps that pose a serious threat to security or fundamental rights are prohibited.
  - **High risk:** AI systems that can have a significant impact on health, safety or fundamental rights must meet strict requirements, including quality, transparency, human oversight, and safety standards.
  - **Limited risk**: These systems have certain transparency requirements so that users know that they are interacting with artificial intelligence.
  - **Minimal risk**: Most AI applications fall into this category and face minimal regulatory requirements.
- AI research targeting high-risk applications must take these requirements into account at an early stage of development.
- The act also includes provisions for general-purpose AI systems, such as underlying models (e.g., ChatGPT). These systems are subject to transparency requirements, and those with higher risk must undergo more detailed assessments. Researchers developing such models should be mindful of these obligations, especially if their system could have a major impact in different sectors.
- Researchers need to be aware of risk classifications and anticipate what requirements will need to be met if they want to **commercialize** their AI system.
- The act emphasizes the importance of **ethical development of AI**, so researchers must consider ethical considerations during development.

Slovenska tovarna umetne intelligence
Slovene AI factory - SLAIF



- namen: razširiti ekosistem umetne inteligence v Sloveniji
- povezava s superračunalniškim omrežjem Euro HPC
- strojna oprema v Sloveniji: superračunalnik Vega 2
- podporni projekt Slo/EU za prenos znanja v znanost in industrijo v okviru programa Obzorje 2020, 2025–2028
- razvoj talentov in znanja
- izobraževalni programi
- dostop do podatkov in upravljanje podatkov
- podpiranje znanosti in podjetij pri uporabi umetne inteligence
- UL FRI je vodilni partner na področju HPC, LLM in računalniškega vida

NLP application examples

Sentiment analysis (SA)

- Definition: a computational study of opinions, sentiments, emotions, and attitudes expressed in texts towards an entity.
- Purpose: detecting public moods, i.e. understanding the opinions of the general public and consumers on social events, political movements, company strategies, marketing campaigns, product preferences etc.
- Part of Affective Computing (emotion, mood, personality traits, interpersonal stance, attitude)
- Can be target-based or general

SA: getting and preprocessing data

- Frequent data sources:
  - Twitter-X, forum comments, product review sites, company's Facebook pages
- Data cleaning
  - quality assessment, annotator (self-) agreement
  - preprocessing: tokenization, emojis, links, hashtags, etc,

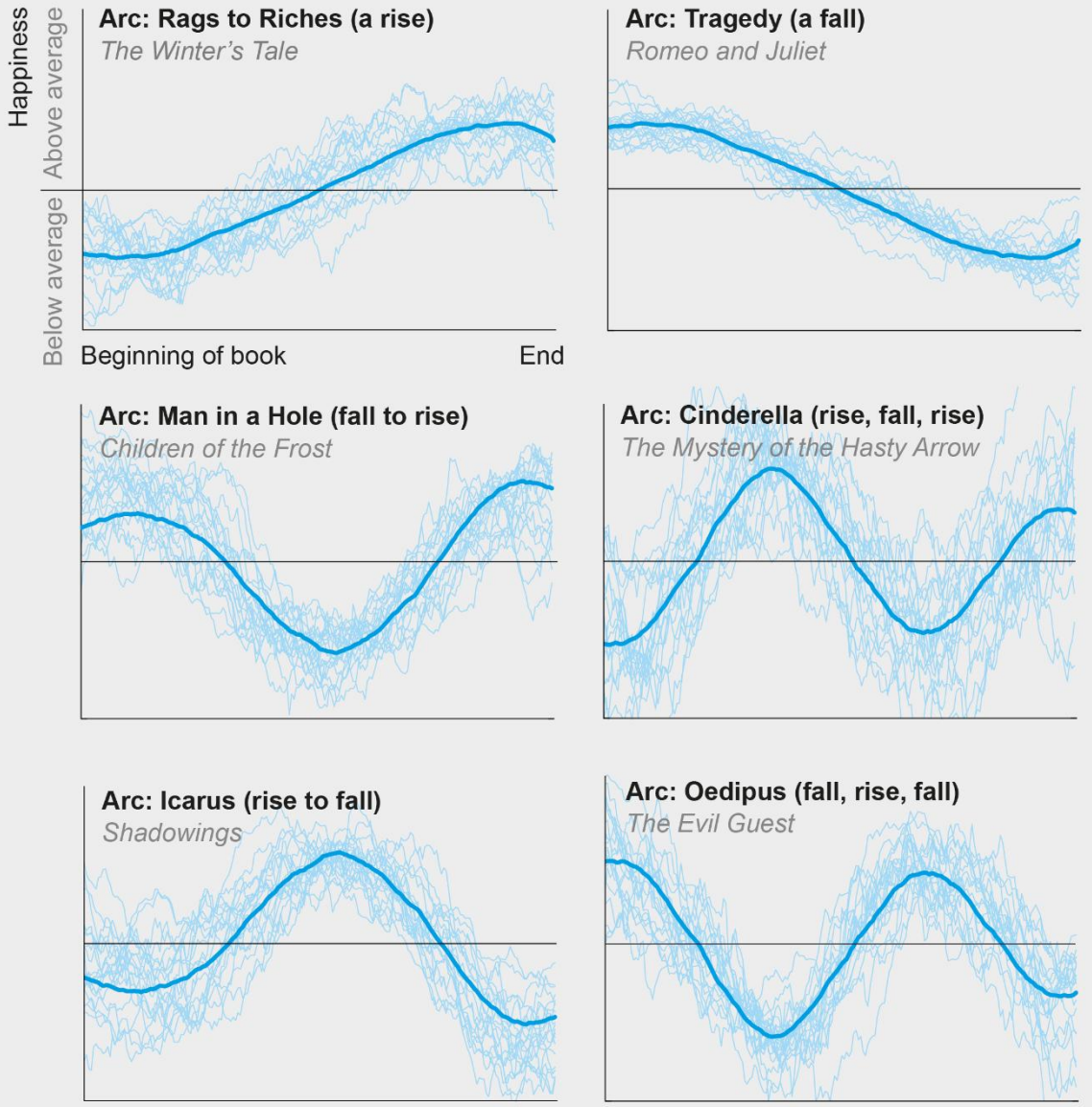
SA tasks

- sentiment classification (binary (polarity), ternary, n-ary
- subjectivity classification (vs. objectivity)
- review usefulness classification
- opinion spam classification
- emotion analysis
- hate speech, offensive speech, socially unacceptable speech
- stance detection

Emotional states in English fiction

Emotional Arcs

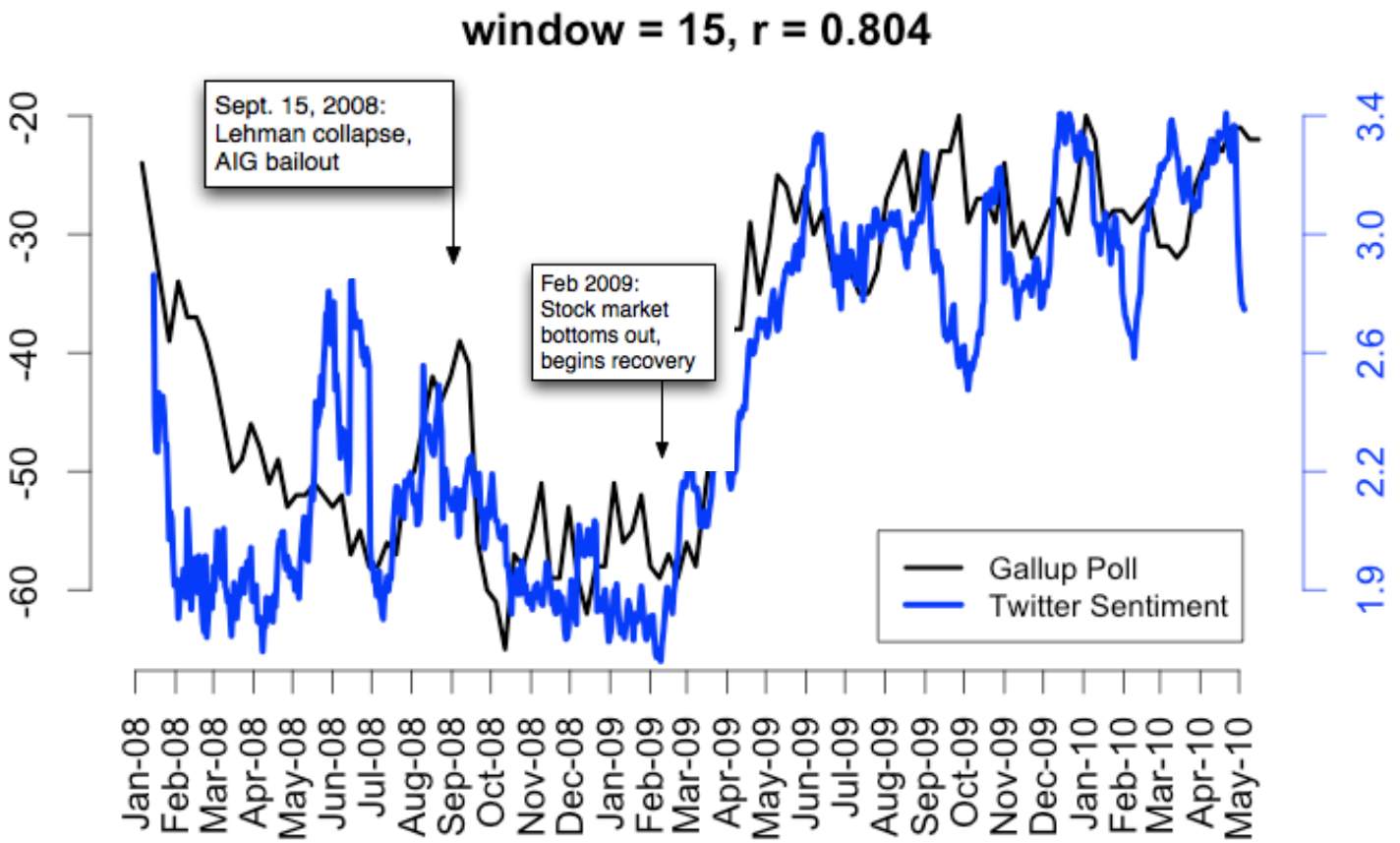
About 85 percent of 1,327 fiction stories in the digitized Project Gutenberg collection follow one of six emotional arcs—a pattern of highs and lows from beginning to end (*dark curves*). The arcs are defined by the happiness or sadness of words in the running text (*jagged plots*). All books were in English and less than 100,000 words; examples are noted.



Public opinion surveys

Twitter sentiment vs. Gallup on consumer sentiment

Brendan O'Connor, Ramnath Balasubramanyan, Bryan R. Routledge, and Noah A. Smith. 2010. From Tweets to Polls: Linking Text Sentiment to Public Opinion Time Series. In ICWSM-2010



Statistical machine translation

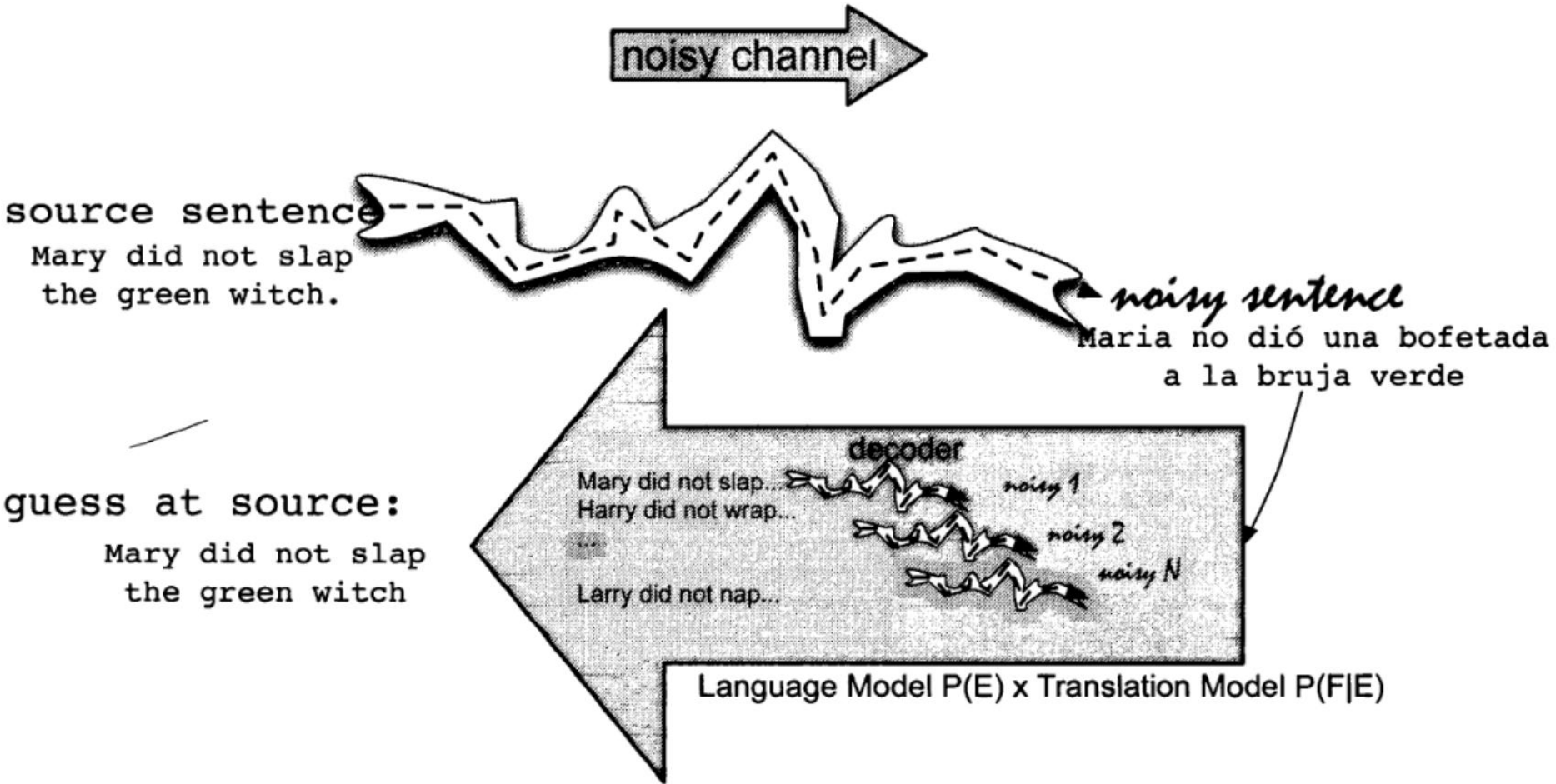
- non-neural approach: no longer used in practice but gives insight of what is needed
- idea from the theory of information
- we translate from foreign language F to English E
- a document is translated based on the probability distribution $p(e \mid f)$, i.e. the probability of the sentence e in target language based on the sentence in source language f
- Bayes rule
  $\arg\max_e p(e \mid f) = \arg\max_e p(f \mid e)\, p(e) \,/\, p(f)$
- $p(f)$ can be ignored as it is a constant for a given fixed sentence
- traditional (non-neural) approaches split the problem into subproblems
  - create a language model $p(e)$
  - a separate translation model $p(f \mid e)$
  - decoder forms the most probable e based on f

Noisy channel model

- given English sentence e
- during transmission over a noisy channel the sentence e is corrupted and we get sentence in a foreign language f, which we are able to observe
- to reconstruct the most probable sentence e we have to figure out:
  - how people speak in English (language model), p(e) and
  - how to transform a foreign language into English (translation model), p(f | e)

Noisy channel

- reasoning goes back



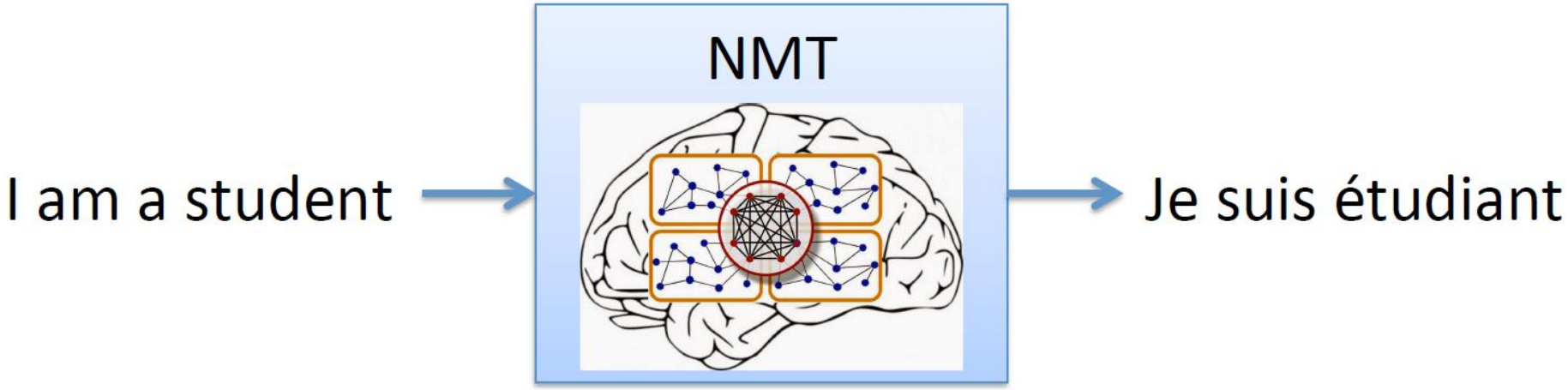
Language model

- each target (English) sentence e is assigned a probability p(e)
- estimation of probabilities for the whole sentences is not possible (why?), therefore we use language models, e.g., 3-gram models or neural language models

Translation model

- we have to assign a probability of p(f | e), which is a probability of a foreign language sentence f, given target sentence e.
- we search the e which maximizes p(e) * p(f | e)
- traditional MT approach: using translation corpus we determine which translation of a given word is the most probable
- we take into account the position of a word and how many words are needed to translate a given word

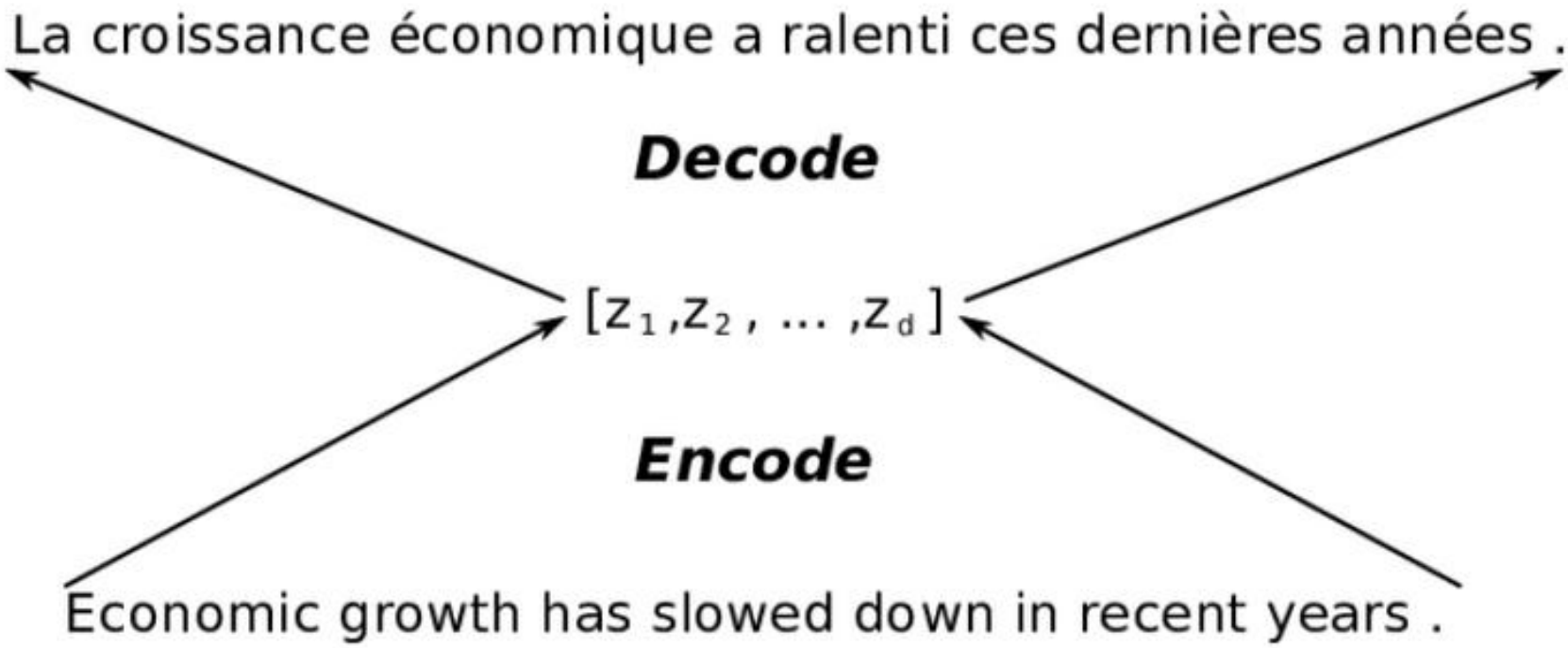
Neural machine translation (NMT)



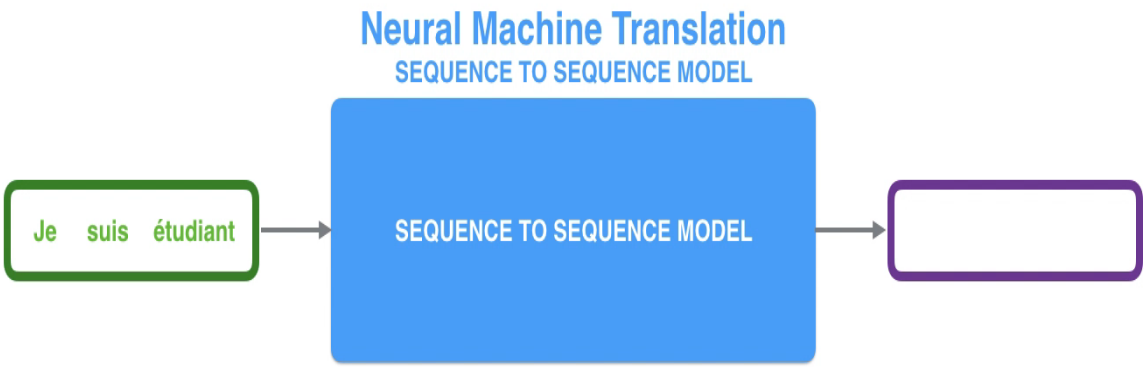
(Sutskever et al., 2014; Cho et al., 2014)

- sequence to sequence machine translation (seq2seq)
- end-to-end optimization

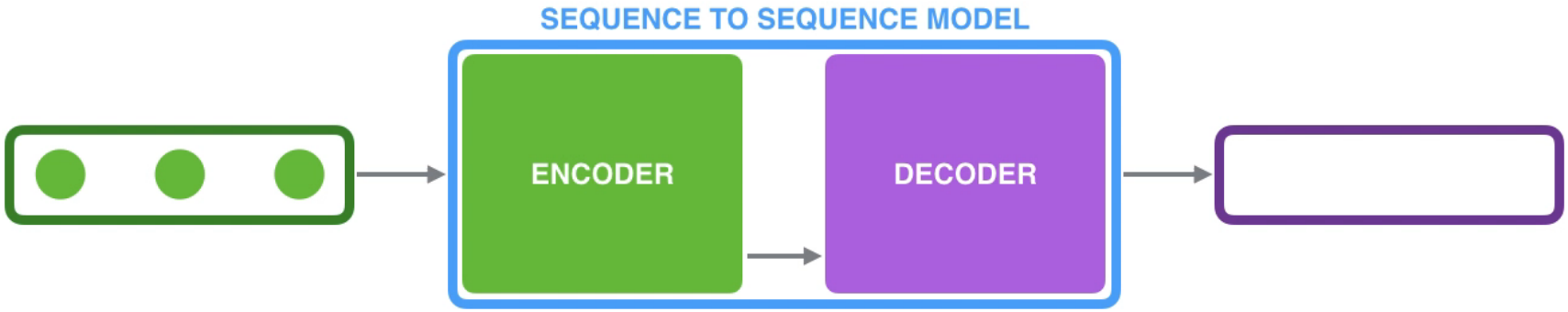
Encoder-Decoder model



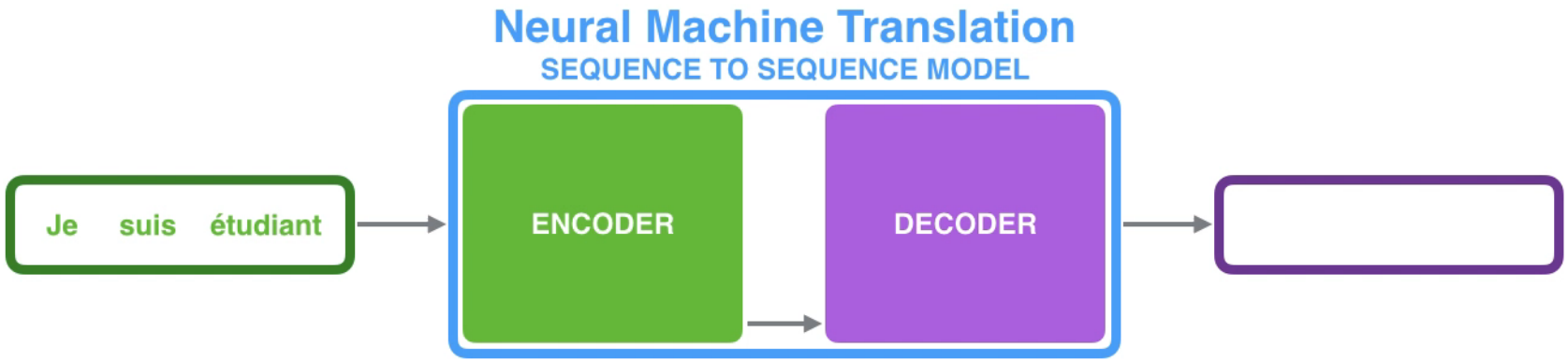
Seq2Seq for NMT

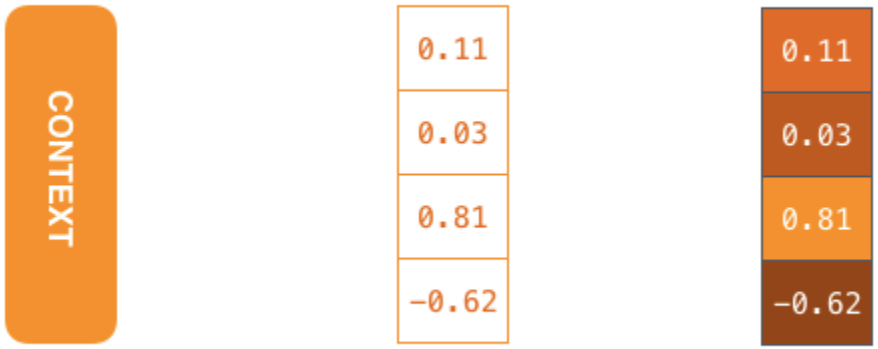


Encoder-decoder for sequences



Encoder-decoder for NMT

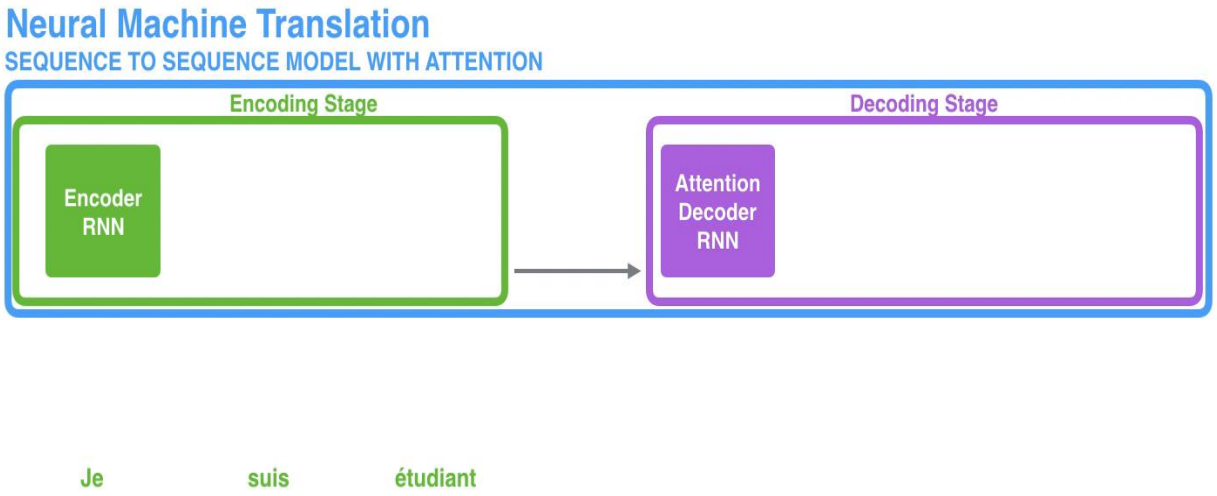




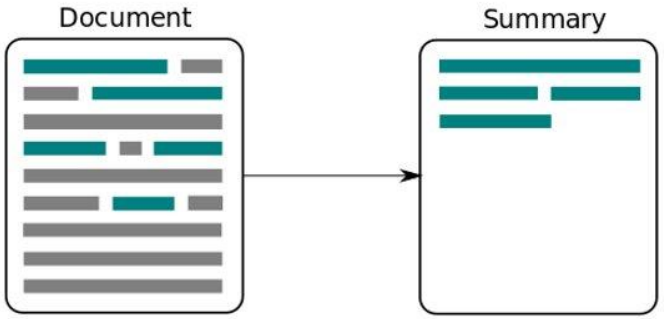
Training NMT

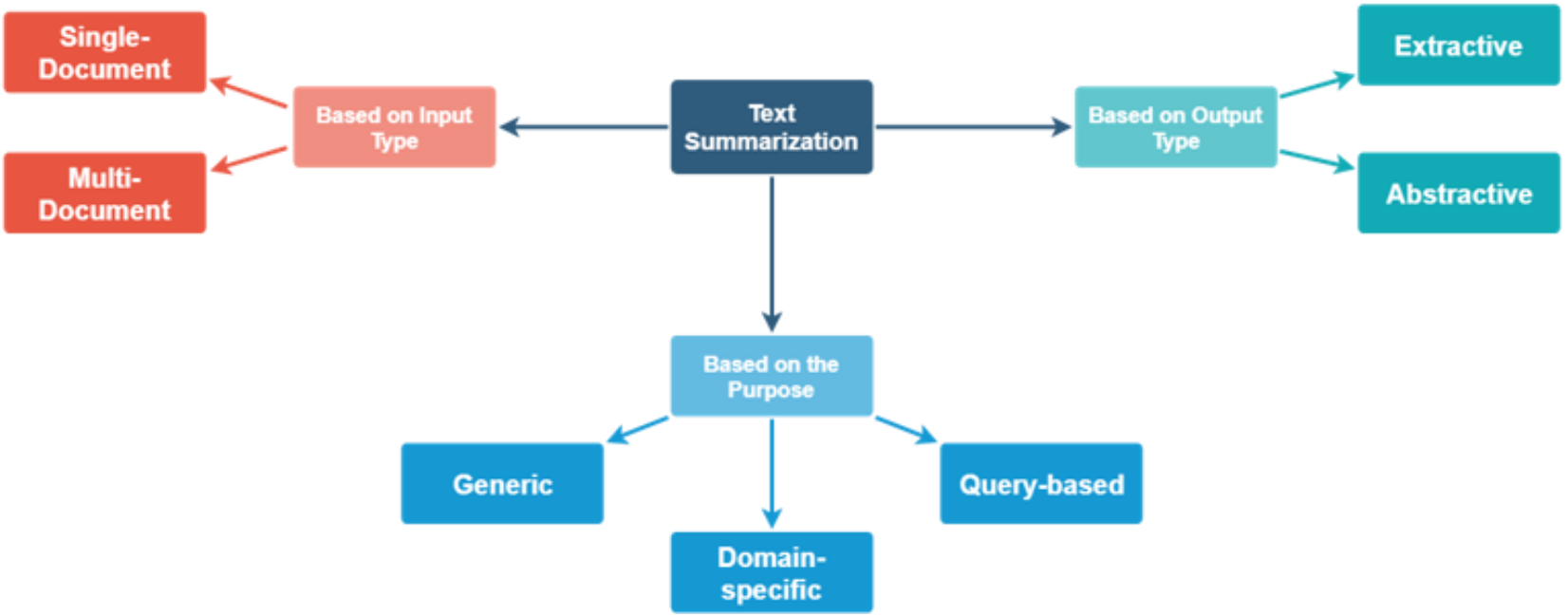
- using transformers
- softmax for output
- we maximize
  P(output sentence | input sentence)
- we sum errors on all outputs
- backpropagation
- training on correct translations
- as the translation, we return a sequence of words with the highest probability (not necessary greedily)

NMT with attention



Text summarization



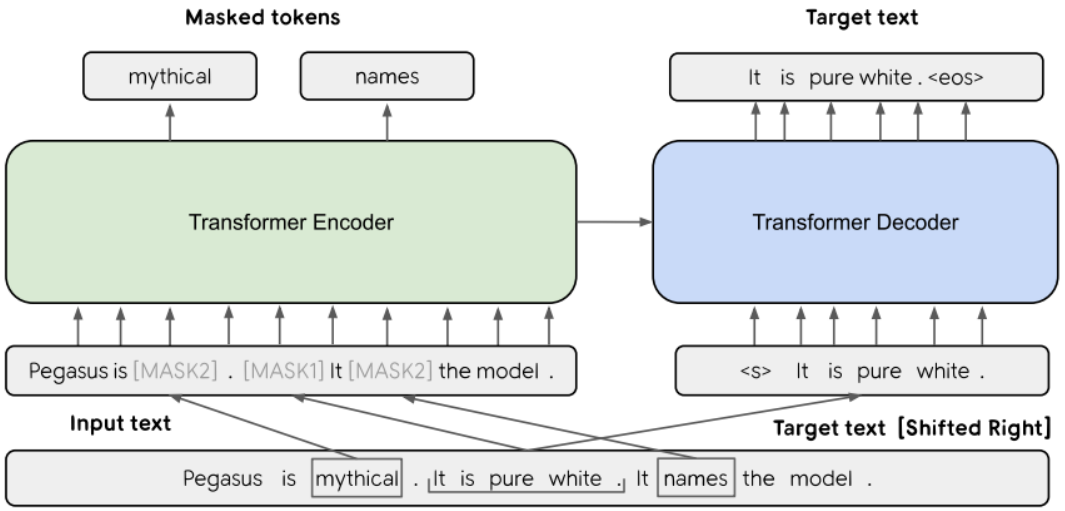


Text summarization

- Evaluation:
  - ROUGE scores,
  - BERTScore,
  - with QA:
    - question generation,
    - searching for answers in the summary
  - human
- LLMs
- Short and long texts
- cross-lingual

Summarizers – Pegasus

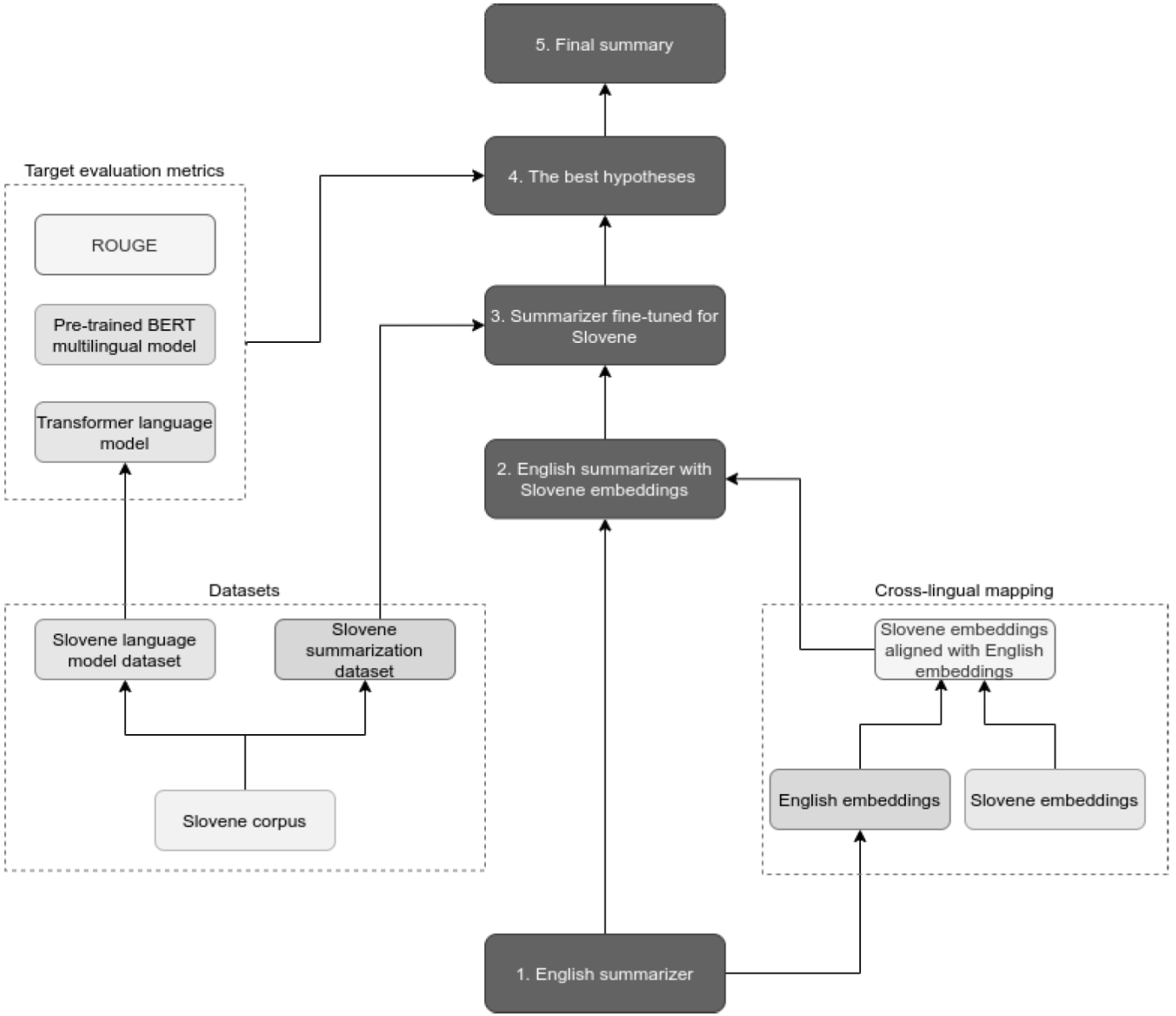
- An example of a different, successful transformer model
- Transformer BART model
- encoder-decoder architecture
- text garbling and reconstruction
- Auxiliary tasks: masked language model and missing sentence generation
- Demo: https://ai.googleblog.com/2020/06/pegasus-state-of-art-model-for.html



(Zhang et al., 2019)

Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

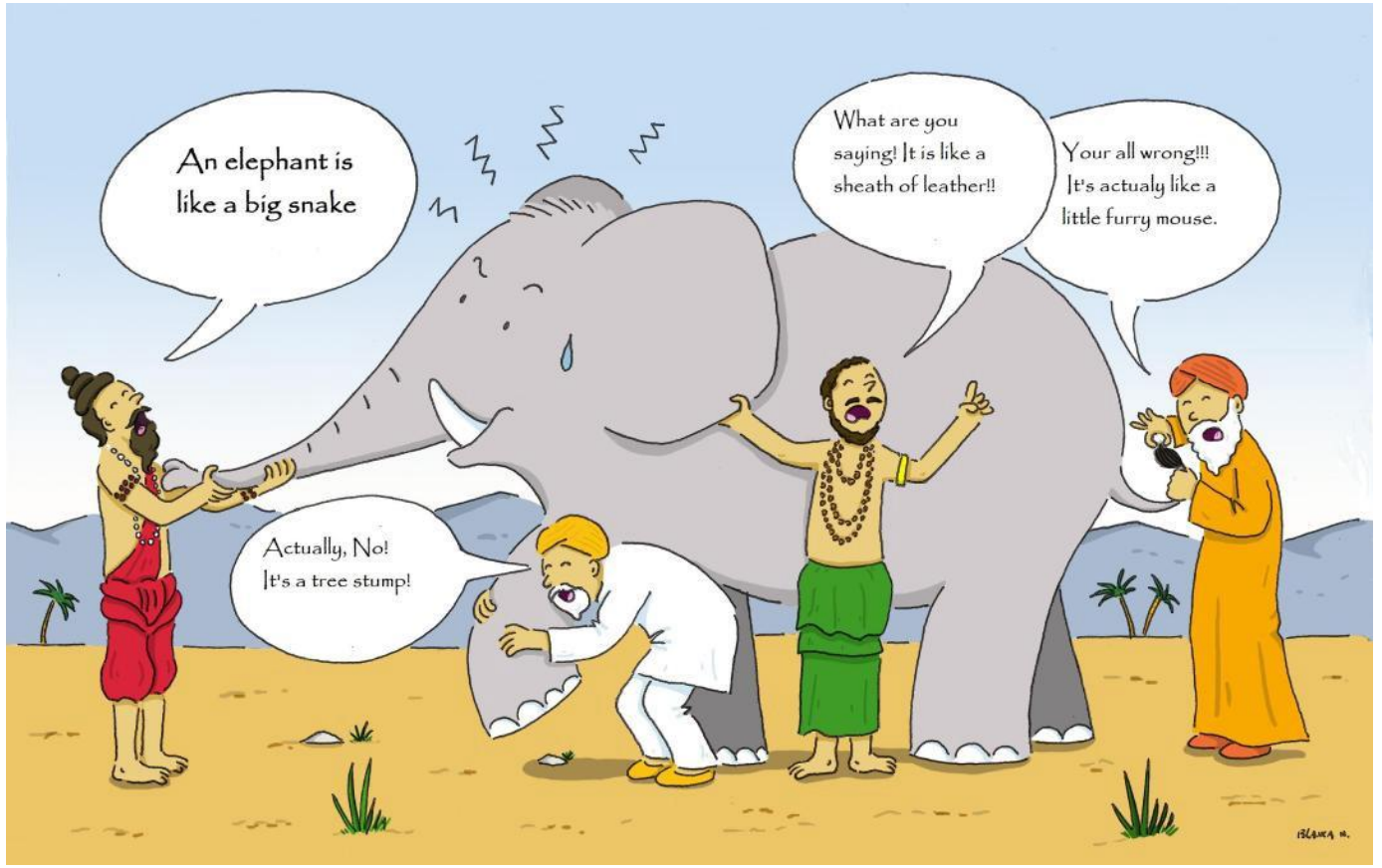
XL summarization architecture



Žagar, A., & Robnik-Šikonja, M. (2022). Cross-lingual transfer of abstractive summarizer to less-resource language. Journal of Intelligent Information Systems, 58(1), 153-173.

University of Ljubljana, Faculty of Computer and Information Science

Ensemble methods



Prof Dr Marko Robnik-Šikonja

Intelligent Systems, Edition 2024

Contents

- about ensembles: how & why
- bagging and random forests
- boosting
- stacking
- a few other ideas

How ensembles works?

- learn large number of basic (simple) classifiers
- merge their predictions

- the most successful methods
  - bagging (Breiman, 1996)
  - boosting (Freund & Shapire, 1996)
  - random forest (Breiman, 1999)
  - XGBoost (eXtreme Gradient Boosting) (Chen & Guestrin, 2016)

Why ensembles work?

- we need different classifiers
  - different in a sense that they produce correct predictions on different instances
- the law of large numbers does the rest
- guidelines for basic classifiers
  - different
  - as strong as possible, but at least weak
- a weak classifier is an expression from computational learning theory (COLT), it means a classifier whose performance is at least $\in > 0$ better than a random classifier

Bagging and random forests

- Bagging
  - sample selection with bootstrapping
  - Bagging for regression trees
  - Bagging for classification trees
  - Out-of-bag error estimation
  - Variable importance: relative influence plots

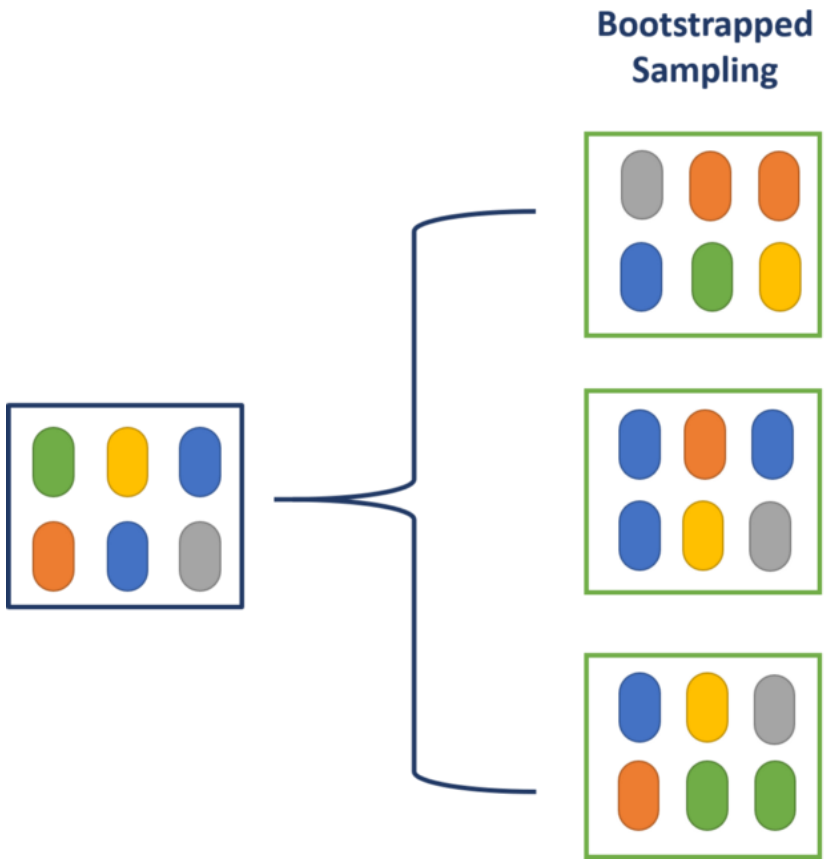
- Random Forests

Bagging

- Decision trees suffer from high variance!
  - If we randomly split the training data into 2 parts, and fit decision trees on both parts, the results of different runs could be quite different
- We would like to have models with low variance
- To solve this problem, we can use bagging (***b***ootstrap ***agg***regat***ing***).

Bootstrapping

- Resampling of the observed dataset (and of equal size to the observed dataset), each of which is obtained by random sampling with replacement from the original dataset.



Bootstrapping

- Draw instances from a dataset *with replacement*
- Probability that we do not pick an instance after N draws

$$\left(1-\frac{1}{N}\right)^{N} \approx e^{-1} = 0.368$$

that is, only 63.2% of instances are used in one draw

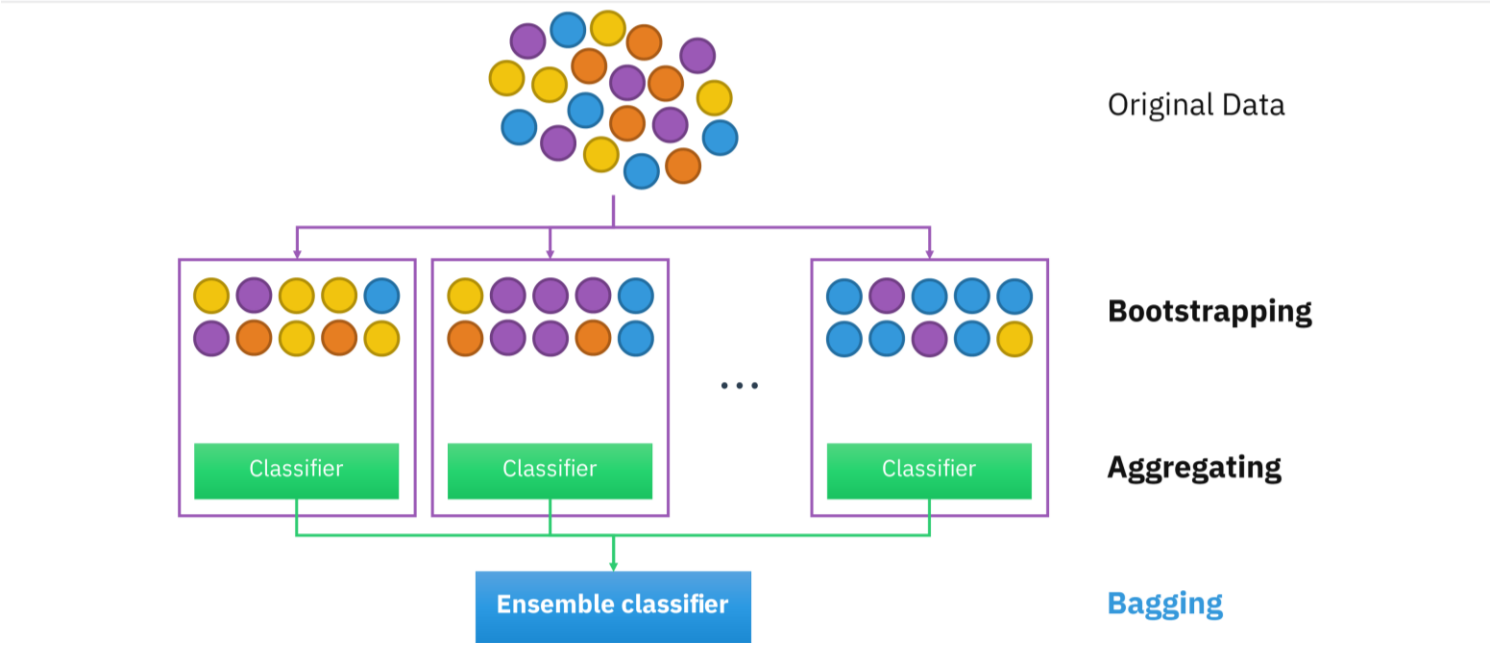
What is bagging?

- Bagging is a powerful idea based on two things:
  - Averaging: reduces variance!
  - Bootstrapping: plenty of training datasets!
- Why does averaging reduces variance?
  - Averaging a set of observations reduces variance.
  - Given a set of *n* independent observations $Z_1, \ldots, Z_n$, each with variance σ^2, the variance of the mean $\overline{Z}$ of the observations is given by σ^2/n.

How does bagging work?

- Generate B different bootstrapped training datasets

- Train the statistical learning method on each of the B training datasets, and obtain the prediction



Bagging for regression trees

- Construct B regression trees using B bootstrapped training datasets
- Average the resulting predictions

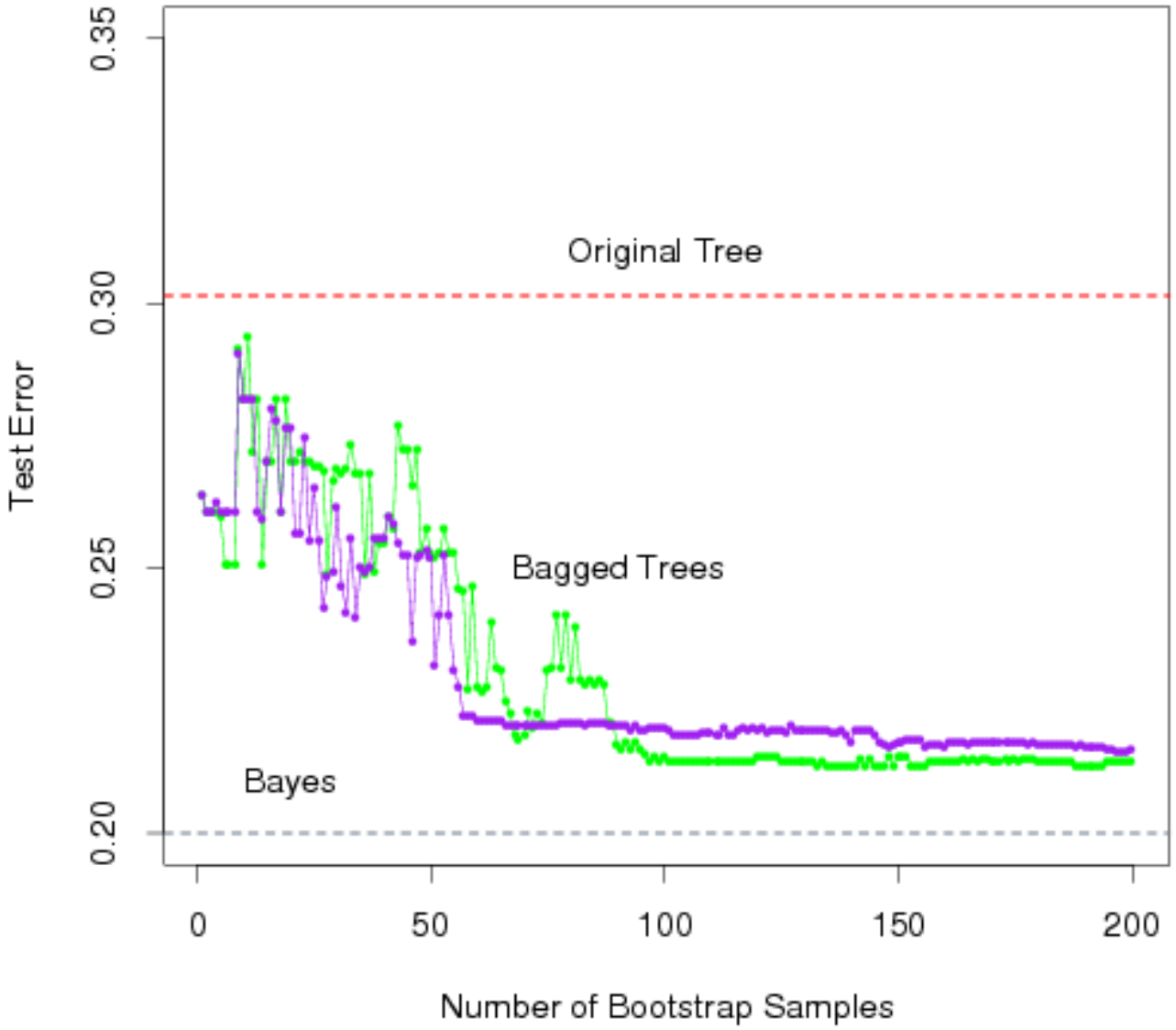
- The trees are not pruned, so each individual tree has high variance but low bias.
- Averaging these trees reduces variance, and thus we end up lowering both variance and bias ☺

Bagging for classification trees

- Construct B decision trees using B bootstrapped training datasets
- For prediction, there are two approaches:
    1. Record the class that each bootstrapped data set predicts and provide an overall prediction to the most commonly occurring one (majority vote).
    2. If our classifier produces probability estimates, we can just average the probabilities and then predict to the class with the highest probability.
- Both methods work well.

A comparison of error rates

- Here the green line represents a simple majority vote approach
- The purple line corresponds to averaging the probability estimates.
- Both do far better than a single tree (dashed red) and get close to the Bayes error rate (dashed grey).



Out-of-bag error estimation

- Since bootstrapping involves random selection of subsets of observations to build a training data set, then the remaining non-selected part could be the testing data.
- On average, each bagged tree makes use of around 1- 1/e ≈ 63% of the observations, so we end up having 1/e ≈ 37% of the observations useful for testing

Variable importance measure

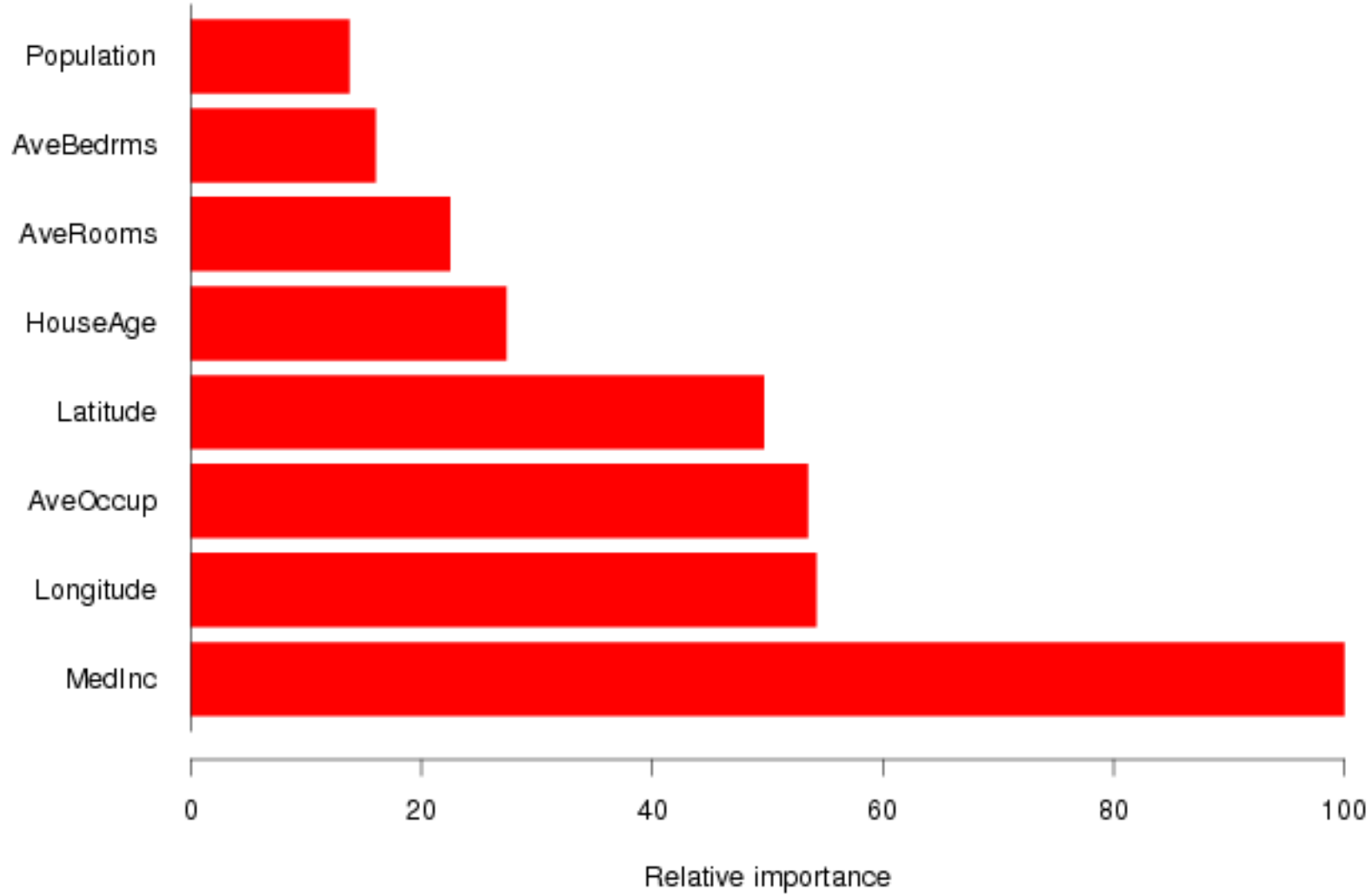
- Bagging typically improves the accuracy over prediction using a single tree, but it is now hard to interpret the model!
- We have hundreds of trees, and it is no longer clear which variables are most important to the procedure
- Thus bagging improves prediction accuracy at the expense of interpretability
- But, we can still get an overall summary of the importance of each predictor using relative influence plots

Relative influence plots

- How do we decide which variables are most useful in predicting the response?
  - We can compute something called relative influence plots.
  - These plots give a score for each variable.
  - These scores represents the decrease in MSE when splitting on a particular variable
  - A number close to zero indicates the variable is not important and could be dropped.
  - The larger the score the more influence the variable has.

Example: Housing data

- Median Income is by far the most important variable.
- Longitude, Latitude and Average occupancy are the next most important.



Random forests

- It is a very efficient statistical learning method
- It builds on the idea of bagging, but it provides an improvement because it de-correlates the trees
- How does it work?
  - Build a number of decision trees on bootstrapped training sample,
  - When building these trees, each time a split in a tree is considered, a random sample of *m* predictors is chosen as split candidates from the full set of *p* predictors.
  - Usually $m \approx \sqrt{p}$ or $m \approx 1 + \log_2 p$

Why are we considering a random sample of *m* predictors instead of all *p* predictors for splitting?

- Suppose that we have a very strong predictor in the data set along with a number of other moderately strong predictors, then in the collection of bagged trees, most or all of them will use the very strong predictor for the first split!
- All bagged trees will look similar. Hence all the predictions from the bagged trees will be highly correlated
- Averaging many highly correlated quantities does not lead to a large variance reduction, and thus random forests “de-correlates” the bagged trees leading to more reduction in variance

Properties

- low classification (and regression) error
- no overfitting
- robust concerning the noise and the number of attributes
- relatively fast
- learning instances not selected with bootstrap replication are used for evaluation of the tree (oob = out-of-bag evaluation)

Out-of-bag evaluation

- on average 1/e ~ 37% of the learning set is not used to train each of the basic classifiers
- classification margin

$$mr(\mathbf{x}, y) = P(h(\mathbf{x}) = y) - \max_{\substack{j=1 \\ j \neq y}}^{c} P(h(\mathbf{x}) = j)$$

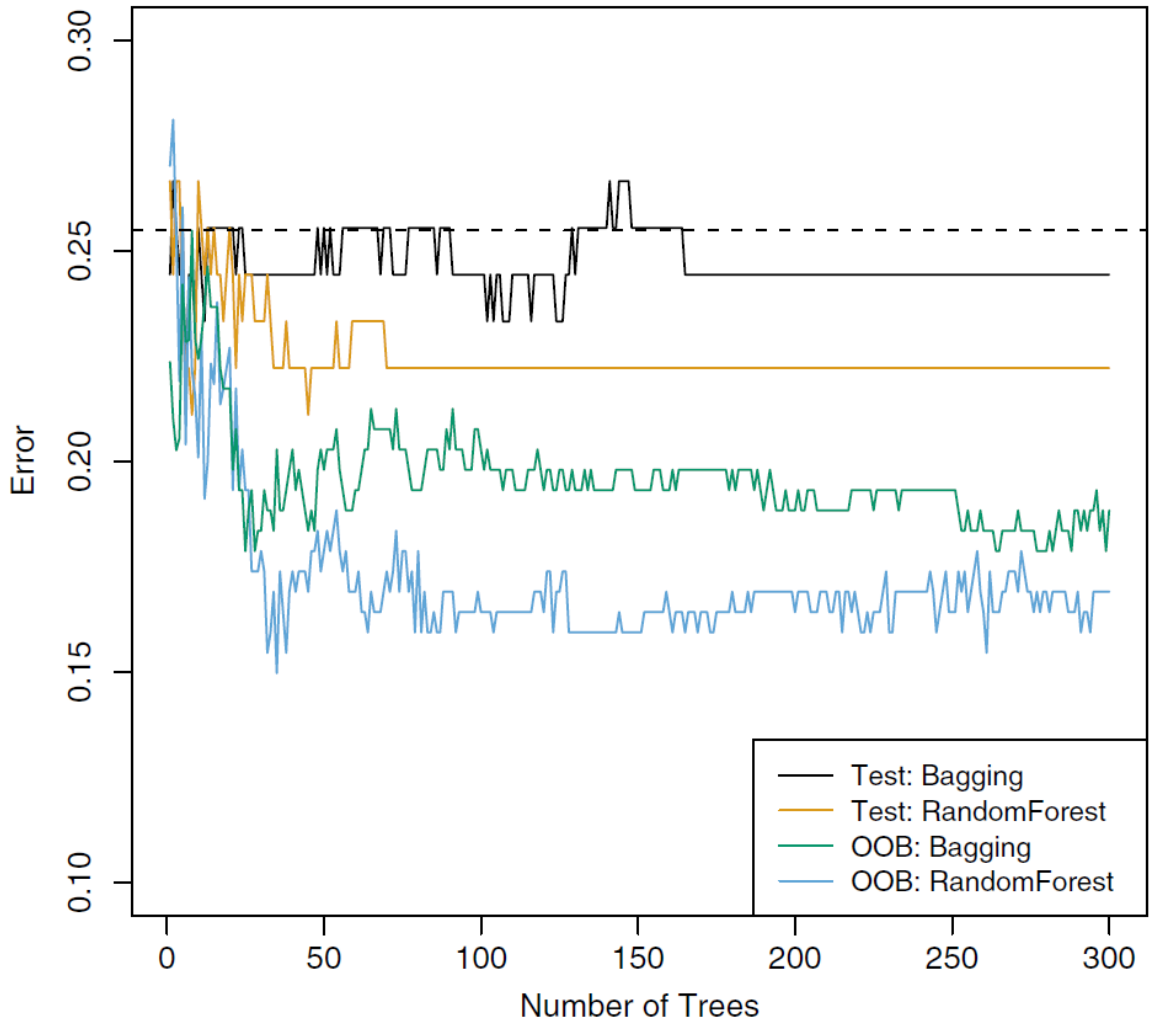
  - mr is estimated with all classifiers where **x** is in oob set
- strength of the forest = average margin over training or OOB set
- correlation of the trees in forest

$$\bar{\rho} = \frac{\mathrm{var}(mr)}{std(h())^2}$$

- we want high strength and low correlation

OOB-error estimate

- with large number of trees, the OOB estimate is roughly equivalent to the CV error estimate
- computationally much cheaper than CV
- still overly optimistic



Heart data set

RF attribute evaluation

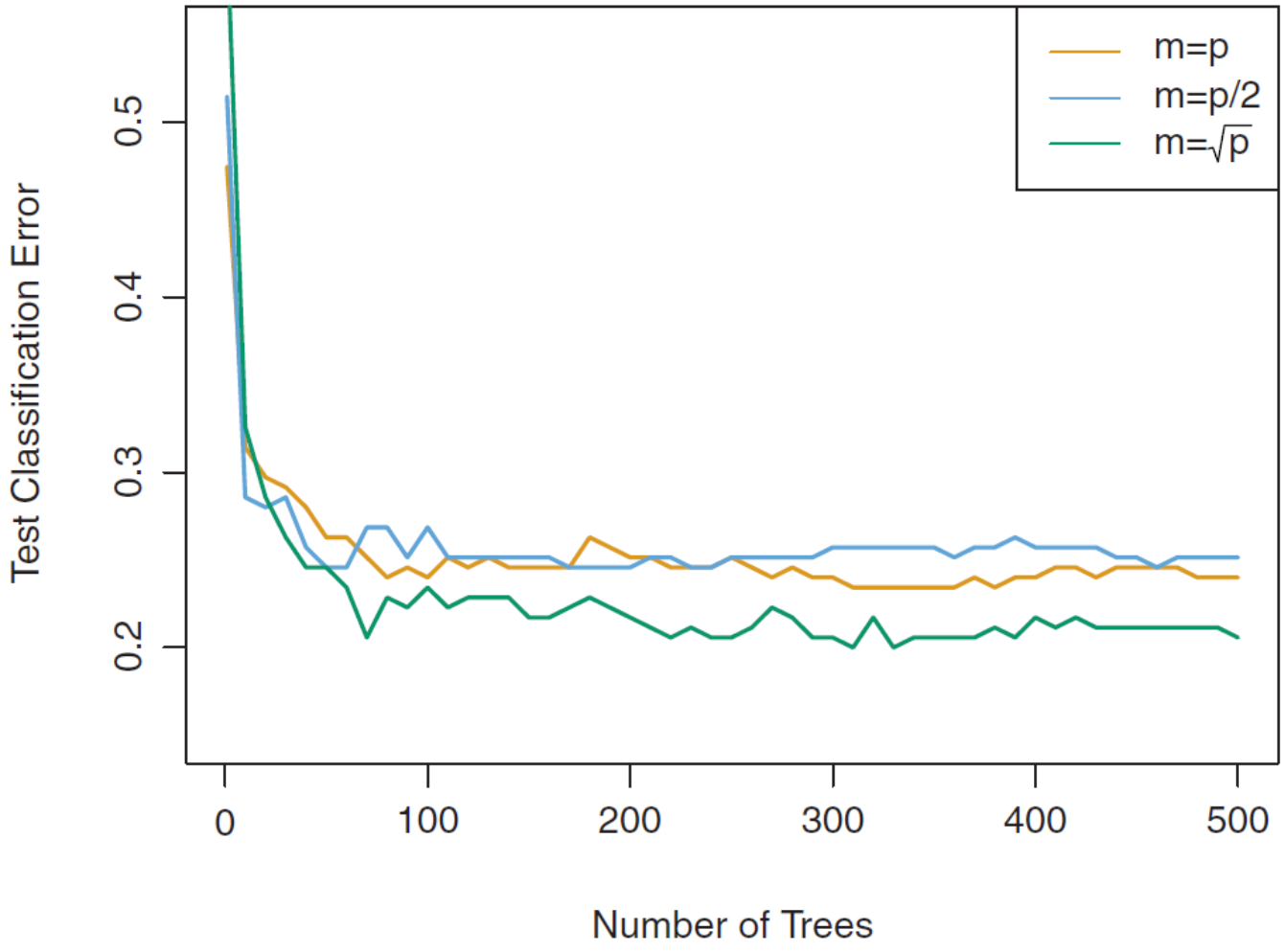
- evaluation of attribute A is the difference between
  - strength of the forest and
  - strength of the forest when values of A are randomly shuffled
- evaluated on the OOB set
- detects also strong conditional dependencies
- works also on an instance-level like nomogram (evaluates only the trees where the instance is in the OOB set)

Similarity of instances

- build instance similarity matrix
- when two instances end in the same leaf of the tree we increase their similarity score
- average over all trees gives similarity measure
- we use that similarity measure to:
  - detect outliers
  - determine typical cases for each class
  - scaling
  - missing values
  - clustering
  - visualization

Random forest with different values of "*m*"

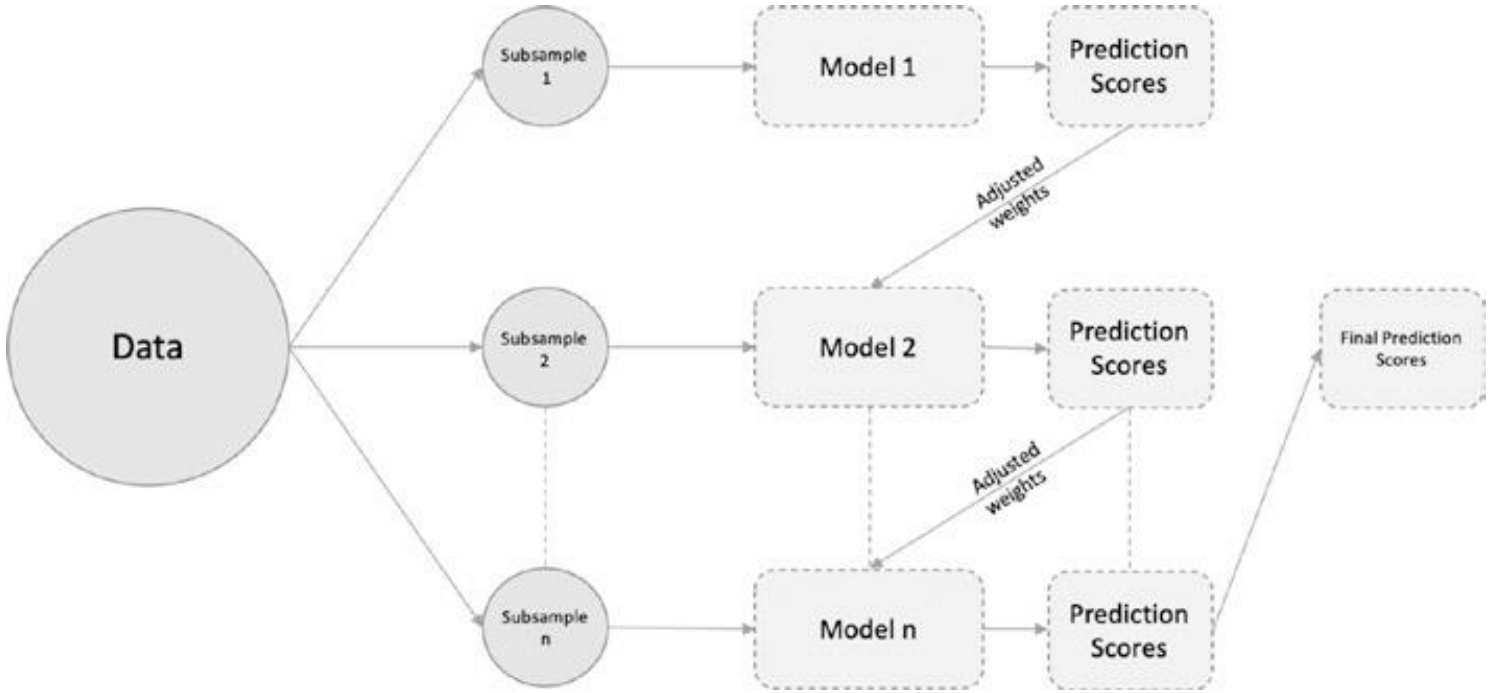
- Notice: when random forests are built using $m = p$, then this amounts to bagging.



Gene expression data (15 classes)

Boosting

- another ensemble method
- grows trees sequentially: each added tree uses information about errors of previous trees



Pseudocode for boosting in regression

1. Set $\hat{f}(x) = 0$ and $r_i = y_i$ for all i in the training set.

2. For $b = 1, 2, \ldots, B$, repeat:

   (a) Fit a tree $\hat{f}^b$ with d splits ($d+1$ terminal nodes) to the training data (X, r).

   (b) Update $\hat{f}$ by adding in a shrunken version of the new tree:

$$\hat{f}(x) \leftarrow \hat{f}(x) + \lambda \hat{f}^b(x). \tag{8.10}$$

   (c) Update the residuals,

$$r_i \leftarrow r_i - \lambda \hat{f}^b(x_i). \tag{8.11}$$

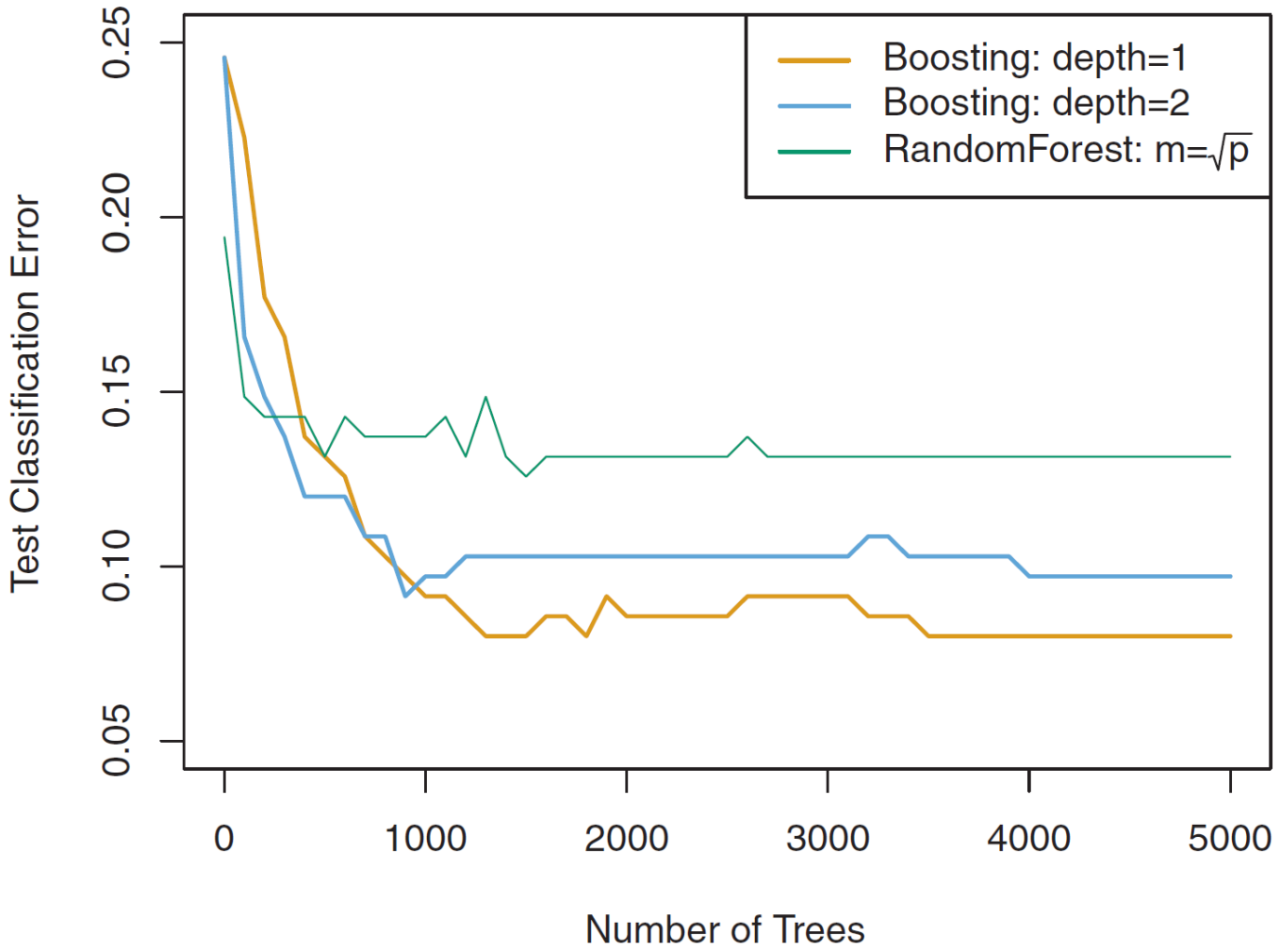
3. Output the boosted model,

$$\hat{f}(x) = \sum_{b=1}^{B} \lambda \hat{f}^b(x). \tag{8.12}$$

Boosting

- each tree takes into account residuals (i.e. errors) of previous trees
- each tree is small, containing only d splits (e.g., d=1, decision stumps)
- learning is slow, controlled by λ
- Parameters of boosting in regression
  - The number of trees *B*, selected with, e.g., CV; boosting can overfit.
  - The shrinkage parameter *λ*, a small positive number (e.g., 0.01 or 0.001), problem dependent; small *λ* requires large B to achieve good performance
  - The number *d* of splits in each tree, which controls the complexity of the boosted ensemble. Often *d* = 1 works well, but *d* also controls interaction order (*d* splits can contain at most *d* variables).

Boosting performance



Gene expression data (15 classes)
error of single tree is approx. 0.24, std. error around 0.02

Boosting in classification

- AdaBoost, Freund & Shapire, ICML, 1996
  - training instances are weighted according to the success of their classification in the previous iteration
    - increase weight of misclassified instances
    - decrease weight of correctly classified instances
    - the learning focus is transferred to the most difficult instances
  - final classification is a weighted voting of basic classifiers
- deterministic algorithm, works because training sets are different
- mostly better than bagging
- this original version can suffer from overfitting but there are better variants

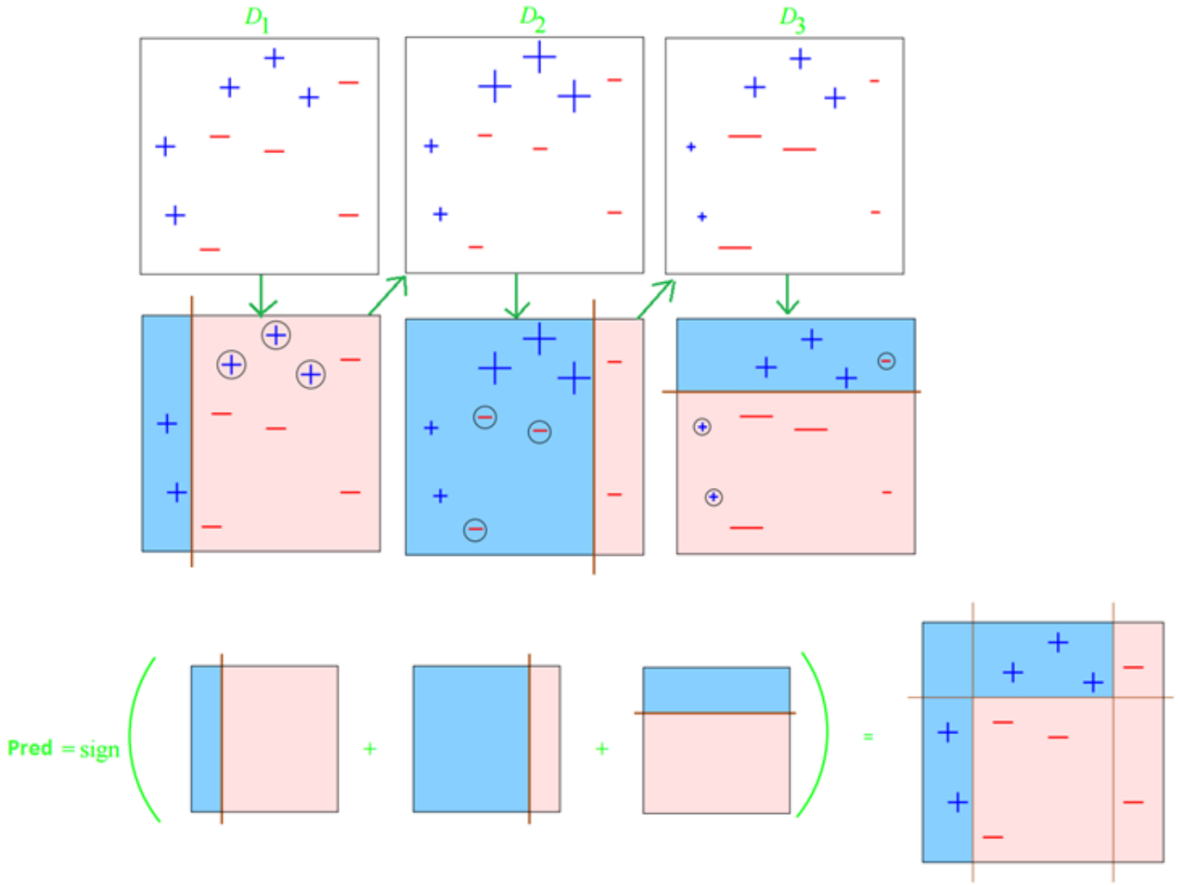
AdaBoost (Freund and Schapire, 1996)

- Given a set of *d* class-labeled instances, $(\boldsymbol{X_1}, y_1), \ldots, (\boldsymbol{X_n}, y_n)$
- Initially, all the weights of instances are set the same (*1/n*)
- Generate *k* classifiers in *k* rounds. At round *i*,
  - Instances from D are sampled (with replacement) or reweighted to form a training set D_i of the same size
  - Each instance's chance of being selected is based on its weight
  - A classification model M_i is derived from D_i
  - Its error rate is calculated using D_i as a test set
  - If an instance is misclassified, its weight is increased, otherwise it is decreased
- Error rate: $err(\boldsymbol{X_j})$ is the misclassification error of instance $\boldsymbol{X_j}$.
  Classifier M_i error rate is the sum of the weights of the misclassified instances:

$$error(M_i) = \sum_{j}^{d} w_j \times err(\mathbf{X_j})$$

- The weight of classifier M_i's vote is $\log \frac{1-error(M_i)}{error(M_i)}$

AdaBoost Example



XGBoost – eXtreme Gradient Boosting

Additive model with loss L:

$$\min_{\alpha_{n=1:N},\beta_{n=1:N}} L\left(y, \sum_{n=1}^{N} \alpha_n f(x, \beta_n)\right)$$

GB approximately solves this objective iteratively and greedily:

$$\min_{\alpha_n,\beta_n} L\left(y, f_{n-1}((x) + \alpha_n f_n(x, \beta_n)\right)$$

Chen & Guestrin(2016), XGBoost: A Scalable Tree Boosting System. *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* https://arxiv.org/abs/1603.02754

https://xgboost.readthedocs.io/en/latest/build.html#r-package-installation

Other possibilities for tree ensembles

- sampling in RF:
  - *p*-sampling without replacement (sampling the proportion of *p* instances, e.g., *p*=10%)
- limiting the size of the trees in RF and bagging
  - more trees needed
- reduced computational complexity
- regularization

Weighting of the trees

- not all trees are equally important (absolutely and in all parts of an instance space)
- weight the trees according to the data
- assume linear combination of base coefficients

$$F(x, a) = a_0 + \sum_{j=1}^{T} a_j \, t_j(x)$$

- solve for coefficients *a*

Penalization

$$\hat{\mathbf{a}} = \arg\min_{a} \frac{1}{N} \sum_{i=1}^{n} L\left(y_{i,} a_0 + \sum_{j=1}^{T} a_j \, t_j(x_i)\right)$$

- direct minimization gives poor generalization, therefore penalize

$$\hat{\mathbf{a}}(\lambda) = \arg\min_{a} \left(\frac{1}{N} \sum_{i=1}^{n} L\left(y_{i,} a_0 + \sum_{j=1}^{T} a_j \, t_j(x_i)\right) + \lambda P(\mathbf{a}) \right)$$

Common penalty functions

- ridge regression

$$P_2(\mathbf{a}) = \sum_{j=1}^{T} |a_j|^2$$

- lasso, sure-shrink

$$P_1(\mathbf{a}) = \sum_{j=1}^{T} |a_j|$$

- solve with gradient descent algorithms (Friedman & Popescu, 2003)

Local weighting

- regularization: global importance of base models
- local importance: local regularization, weighting with margin of similar instances

Locally weighted voting for RF

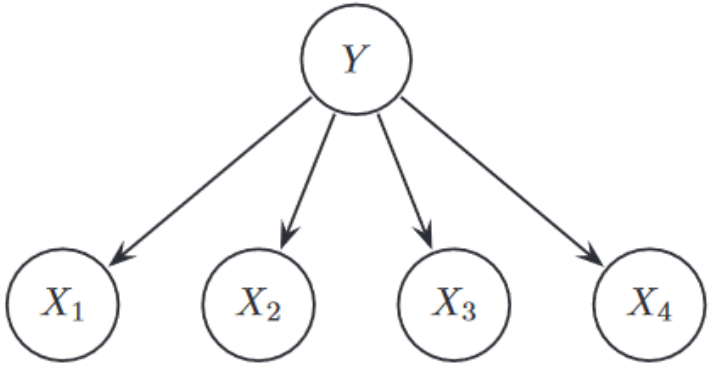
- observation: not all trees are equally good in all parts of the problem space
- opportunity: use OOB instances to locally evaluate the quality of trees
- locality: forest defines the similarity between instances

Weighted voting algorithm for RF

- in classification of a new instance
  - find *t* most similar instances
  - classify each of the similar instances with the trees where it is in the OOB set, and record the margin for the trees
  - compute weights of the trees as the average recorded margin (for trees with negative margin set the weight to zero)
  - forest classification is the weighted voting of the trees

Marko Robnik-Šikonja: Improving random forests. In *Proceedings of European conference on machine learning ECML'04*, pp. 359-370. 2004.

Naïve Bayes based ensembles



- Naive Bayes is a probabilistic classifier

$$\begin{aligned} \underset{y}{\operatorname{argmax}}\, P(y \mid \mathbf{x}) &= \underset{y}{\operatorname{argmax}}\, P(y, \mathbf{x}) / P(\mathbf{x}) \\ &= \underset{y}{\operatorname{argmax}}\, P(y, \mathbf{x}). \end{aligned}$$

- assuming that the attributes are independent given the class

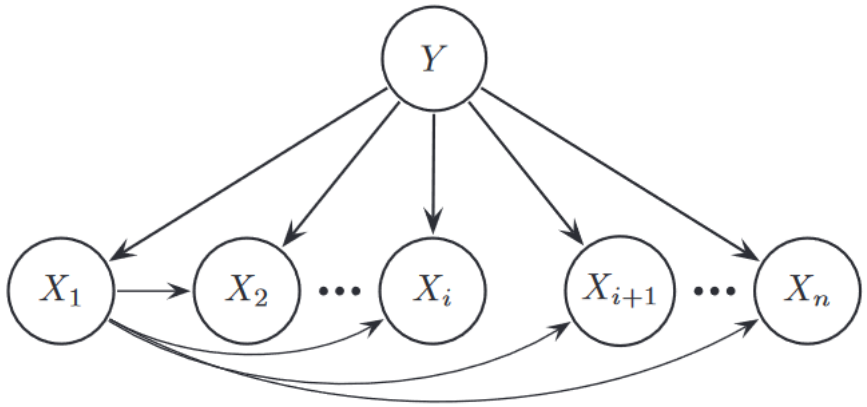
$$\hat{P}(y, \mathbf{x}) = \hat{P}(y) \prod_{i \in N} \hat{P}(x_i \mid y),$$

Semi naïve Bayes (SNB)

- besides the class, SNB allows dependence on some attributes

$$\hat{P}(y, \mathbf{x}) = \hat{P}(y) \prod_{i \in N} \hat{P}(x_i \mid y, \pi(x_i)),$$

- Example: 1-dependence estimator (ODE), where X_1 is “super-parent”

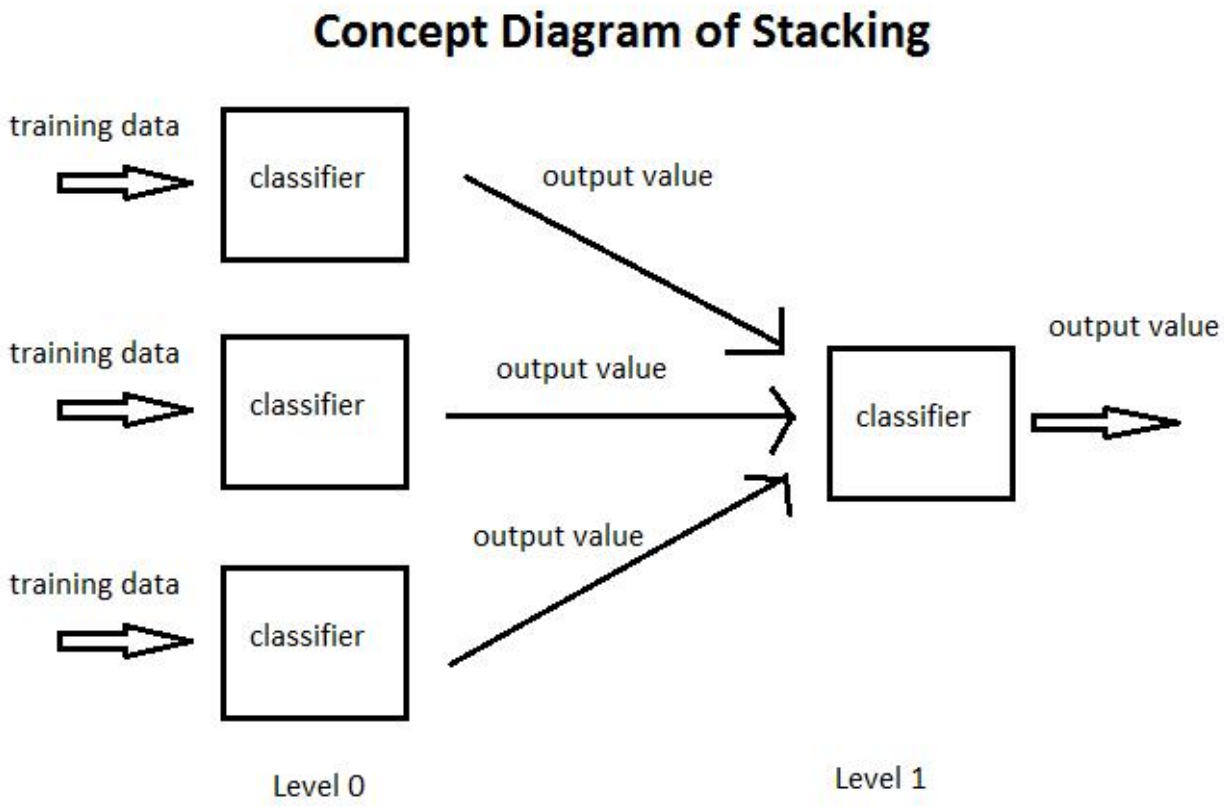


AODE ensemble

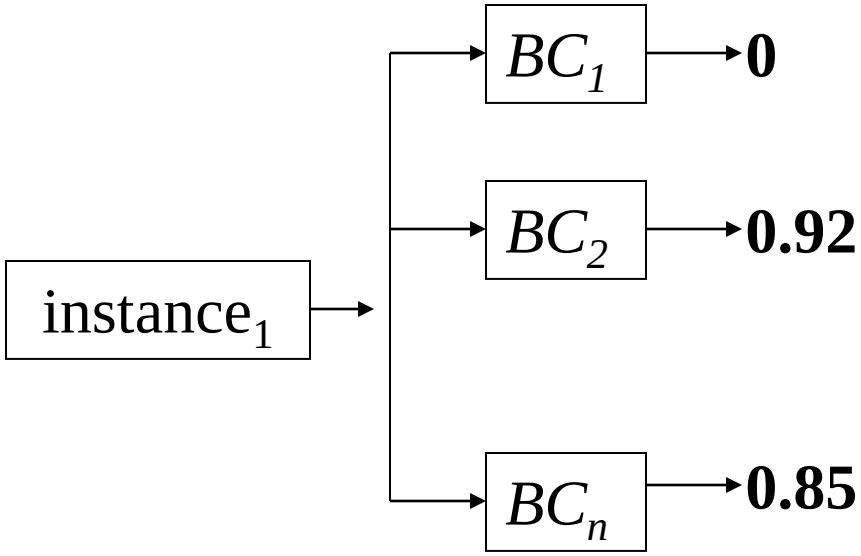
- Averaged One-Dependence Estimator (AODE) (Webb et al. 2005)
- SPODE: Super-Parent One Dependence Estimator – Semi naive Bayes where attributes are dependent on class and one more attribute
- AODE is an ensemble of SPODE classifiers, where all attributes in turn are used in SPODE classifier and their results are averaged
- Compared to naive Bayes, it has higher variance but lower bias

Stacking

- A method to combine heterogeneous predictors
- Predictions of base learners (level-0 models) are used as input for meta learner (level-1 model)
- Base learners are usually different learning schemes

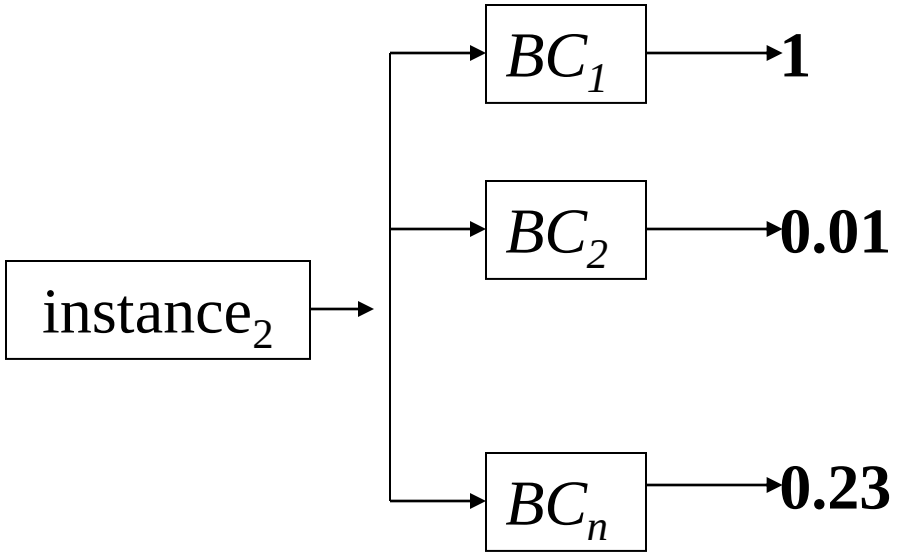


Stacking scheme



meta instances	BC_1	BC_2	…	BC_n	*Class*
instance$_1$	**0**	**0.92**		**0.85**	**1**

Stacking



meta instances	BC_1	BC_2	…	BC_n	*Class*
instance$_1$	**0**	**0.92**		**0.85**	**1**
instance$_2$	**1**	**0.01**		**0.23**	**0**

Stacking

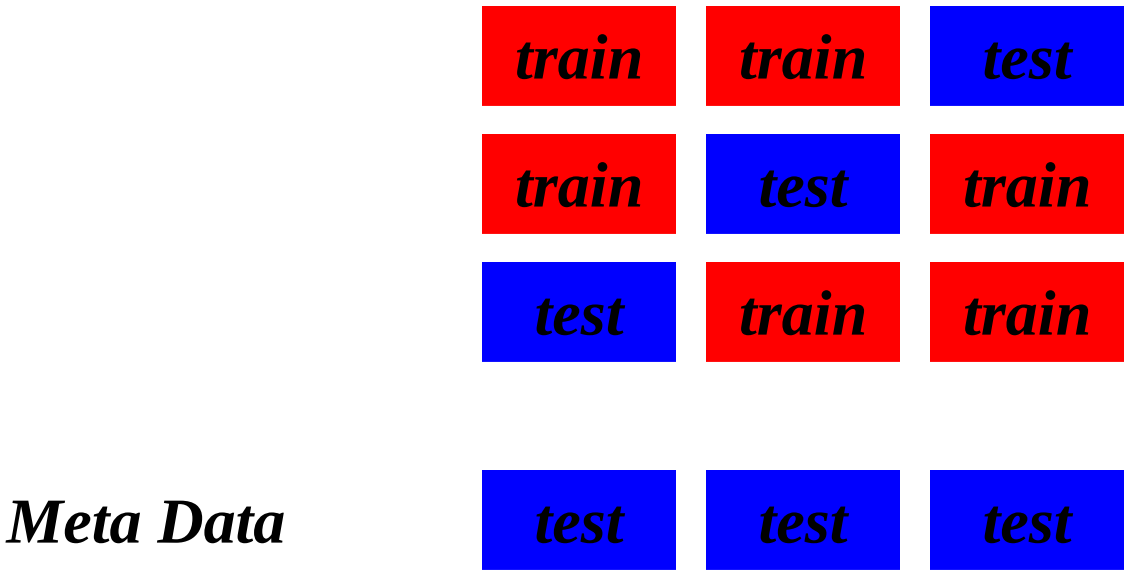
Meta Classifier



meta instances	BC_1	BC_2	…	BC_n	*Class*
$instance_1$	**0**	**0.92**		**0.85**	**1**
$instance_2$	**1**	**0.01**		**0.23**	**0**

Actual stacking

- Predictions on the training data can’t be used to generate data for level-1 model! Why not?
- The reason is that the level-0 classifier that better fit training data will be chosen by the level-1 model!
- Thus, k-fold cross-validation-like scheme is employed. An example for k = 3!

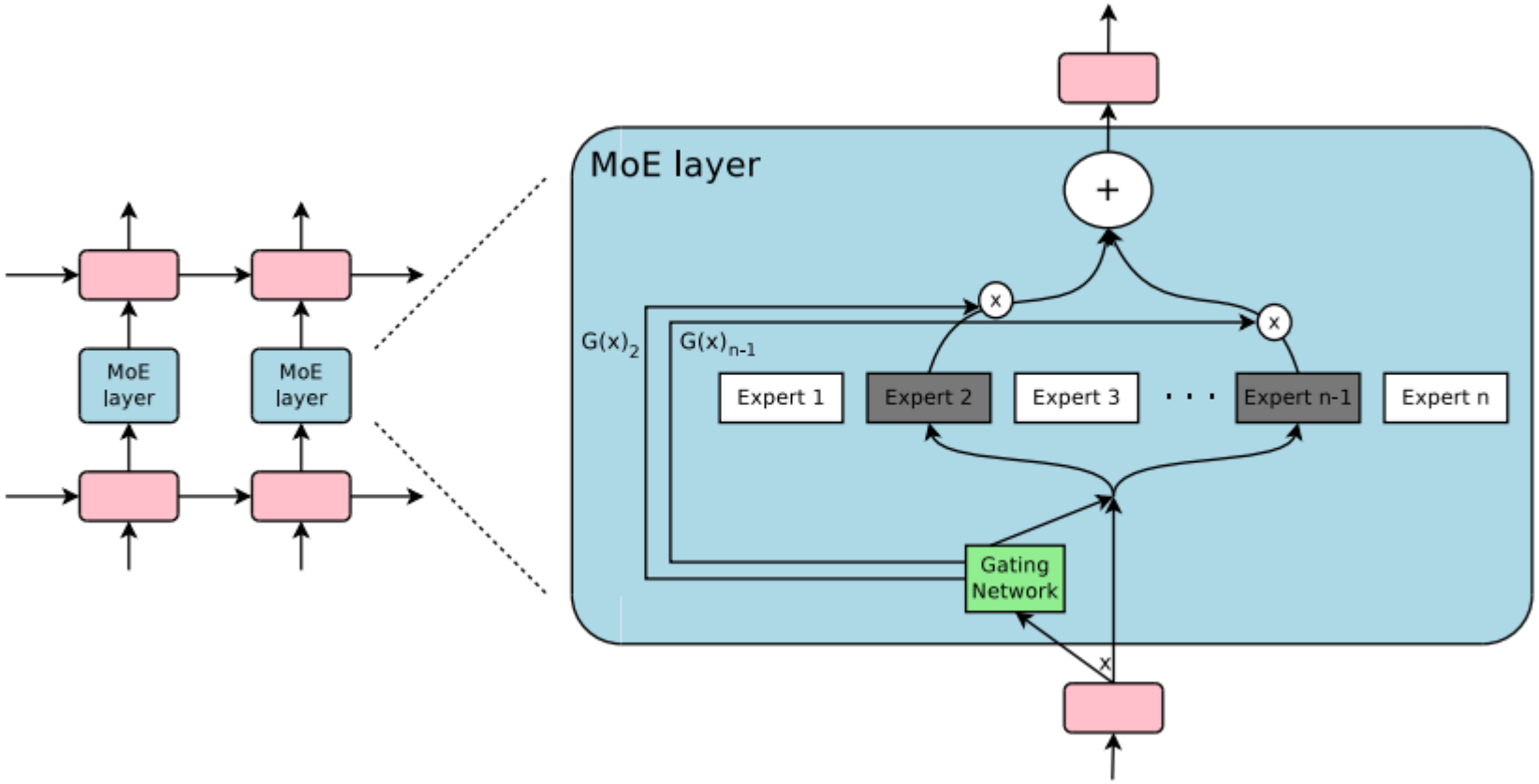


Stacking meta-learner

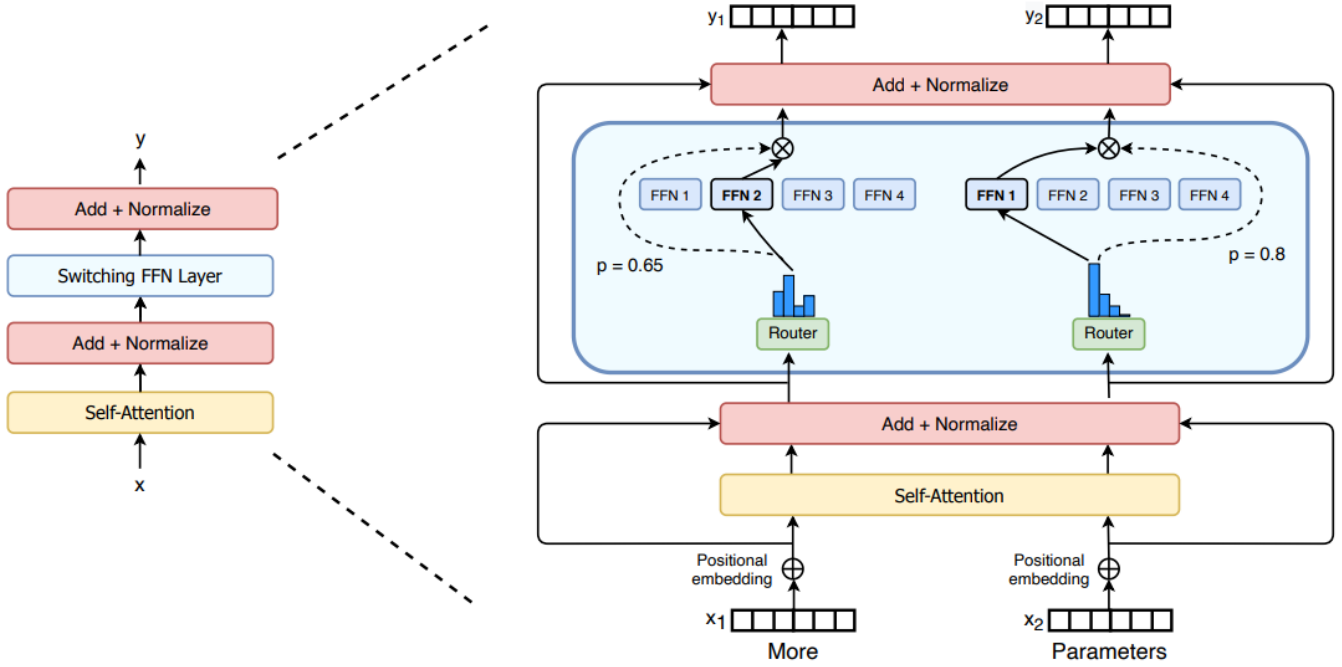
- Which algorithm to use to generate meta learner?
- In principle, any learning scheme can be applied
- For level-1 classifier Ting & Witten (1999) recommend multiple response linear regression (MRLE, note this is a regressor)
  - a classification problem with C classes is transformed into C linear regression problems, where response for problem *i* is 1 if the class equals *i*, otherwise it is 0
  - to classify a new instance employ all *C* linear models, the prediction with highest value is selected as the output

Mixture of Experts (MoE)

- Ensemble technique, useful in very large problems



MoE in transformers

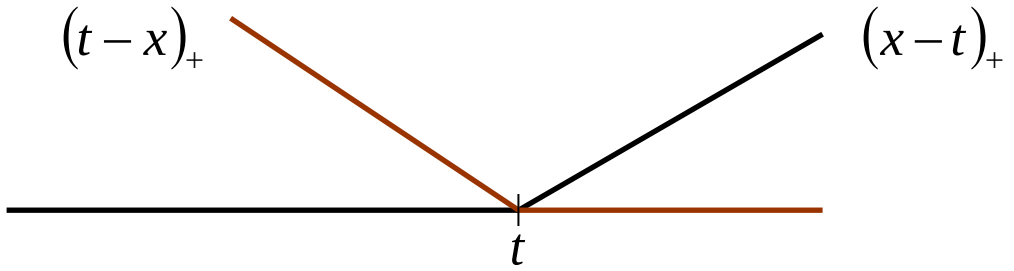


MARS - Multivariate Adaptive Regression Splines

- Generalization of stepwise linear regression
- Modification of trees to improve regression performance
- Able to capture additive structure
- Not tree-based

MARS base models

- Additive model with adaptive set of basis vectors
- Basis built up from simple piecewise linear functions



- Set "*C*" represents candidate set of linear splines, with "knees" at each data point X_j.
- Models are built with elements from C or their products.

$$C = \left\{ (X_j - t)_+, (t - X_j)_+ \right\}_{t \in \{x_{1j}, x_{2j}, \ldots x_{Nj}\} j=1,2,\ldots,p}$$

- Basis collections C: |C| = 2 * N * p

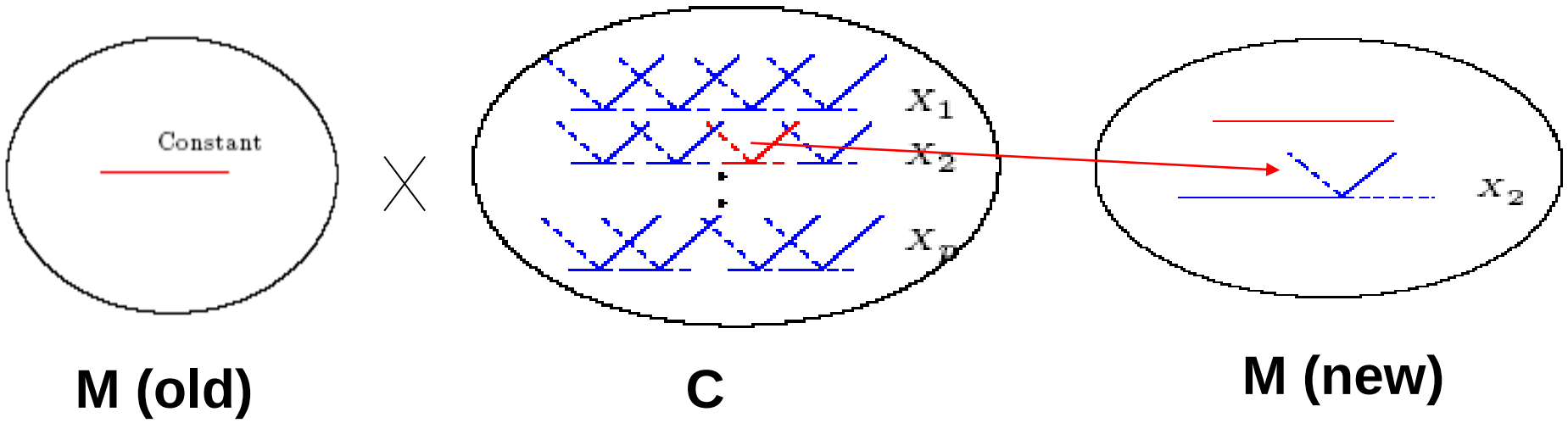
MARS procedure

Model has the form: $f(X) = \beta_0 + \sum_{m=1}^{M} \beta_m h_m(X)$

1. Given a choice for the h_m, the coefficients β are chosen by the standard linear regression.
2. Start with $h_0(X) = 1$
   All functions in *C* are candidate functions.
3. At each stage, consider as a new basis function pair all products of a function h_m in the model set *M*, with one of the reflected pairs in *C*.
   $\beta_{M+1} h_l(X) \cdot (X_j - t)_+ + \beta_{M+2} h_l(X) \cdot (t - X_j)_+, h_l \in M$
4. We add to the model terms of the form:

   $h_m(X) \cdot (t - X_j)_+$ $\qquad$ $h_m(X) \cdot (X_j - t)_+$

MARS, step 1



- On each step, add the term, which reduces residual error most, into *M*
- Repeat steps (until, e.g., *|M|* >= threshold)

MARS, choosing number of terms

- Large models can overfit.
- Backward deletion procedure: delete terms which cause the smallest increase in residual squared error, to get a sequence of models.
- Pick Model using Generalized Cross Validation:

$$GCV(\lambda) = \frac{\sum_{i=1}^{N} \left(y_i - \hat{f}(x_i)\right)^2}{(1 - M(\lambda)/N)^2}$$

- $M(\lambda)$ is the effective number of parameters in the model.
  C=3, r is the number of basis vectors, and *K* knots

$$M(\lambda) = r + cK$$

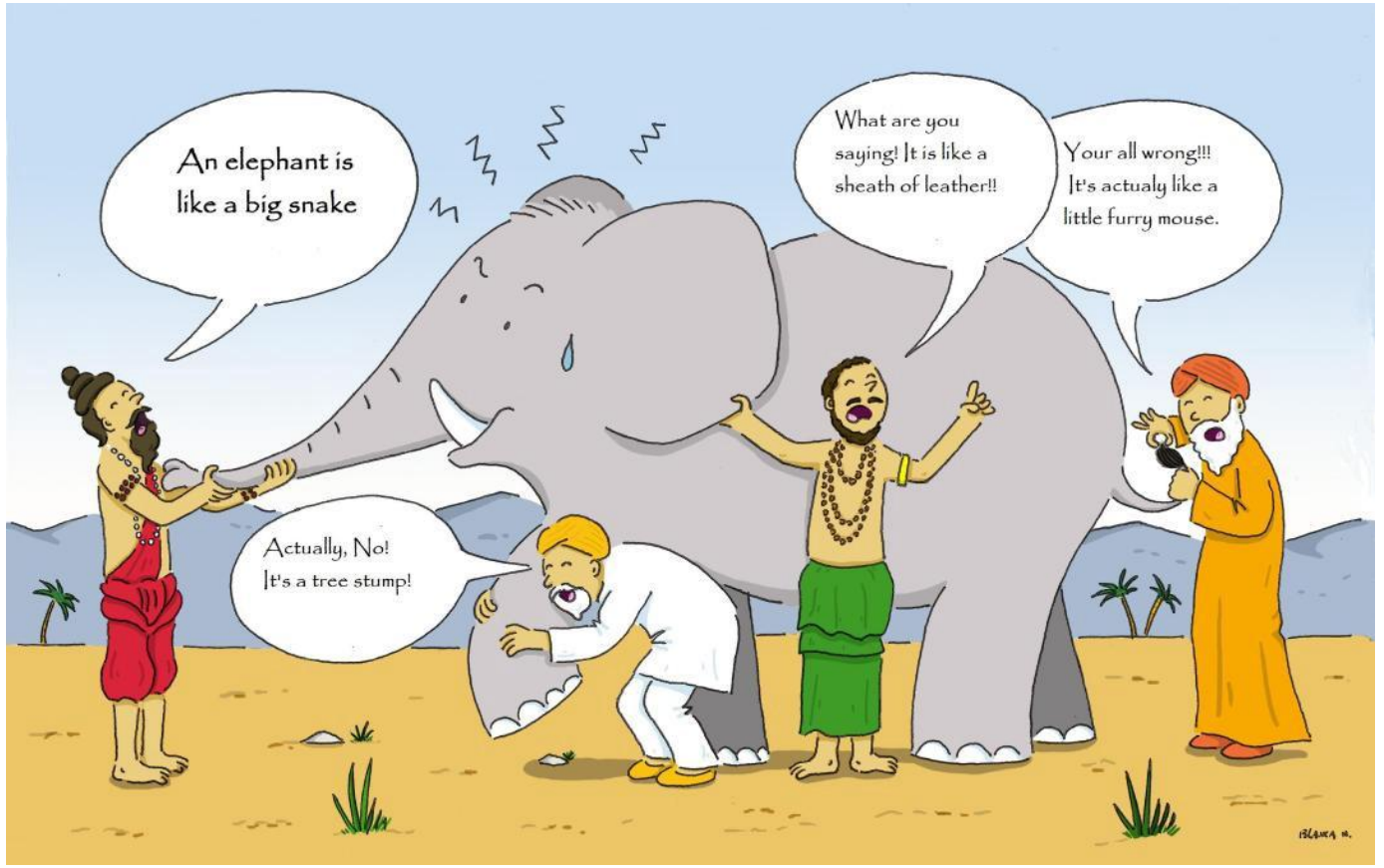
- Choose the model which minimizes $GCV(\lambda)$

MARS summary

- Basis functions operate locally
- Forward modeling is hierarchical, multiway products are built up only from existing terms
- Each input appears only once in each product
- Useful option is to set limit on order of operations. Limit of two allows only pairwise products. Limit of one results in an additive model

University of Ljubljana, Faculty of Computer and Information Science

Ensemble methods



Prof Dr Marko Robnik-Šikonja

Intelligent Systems, Edition 2025

Contents

- about ensembles: how & why
- bagging and random forests
- boosting
- stacking
- a few other ideas

How ensembles works?

- learn large number of basic (simple) classifiers
- merge their predictions

- the most successful methods
  - bagging (Breiman, 1996)
  - boosting (Freund & Shapire, 1996)
  - random forest (Breiman, 1999)
  - XGBoost (eXtreme Gradient Boosting) (Chen & Guestrin, 2016)

Why ensembles work?

- we need different classifiers
  - different in a sense that they produce correct predictions on different instances
- the law of large numbers does the rest
- guidelines for basic classifiers
  - different
  - as strong as possible, but at least weak
- a weak classifier is an expression from computational learning theory (COLT), it means a classifier whose performance is at least $\in > 0$ better than a random classifier

Bagging and random forests

- Bagging
  - sample selection with bootstrapping
  - Bagging for regression trees
  - Bagging for classification trees
  - Out-of-bag error estimation
  - Variable importance: relative influence plots

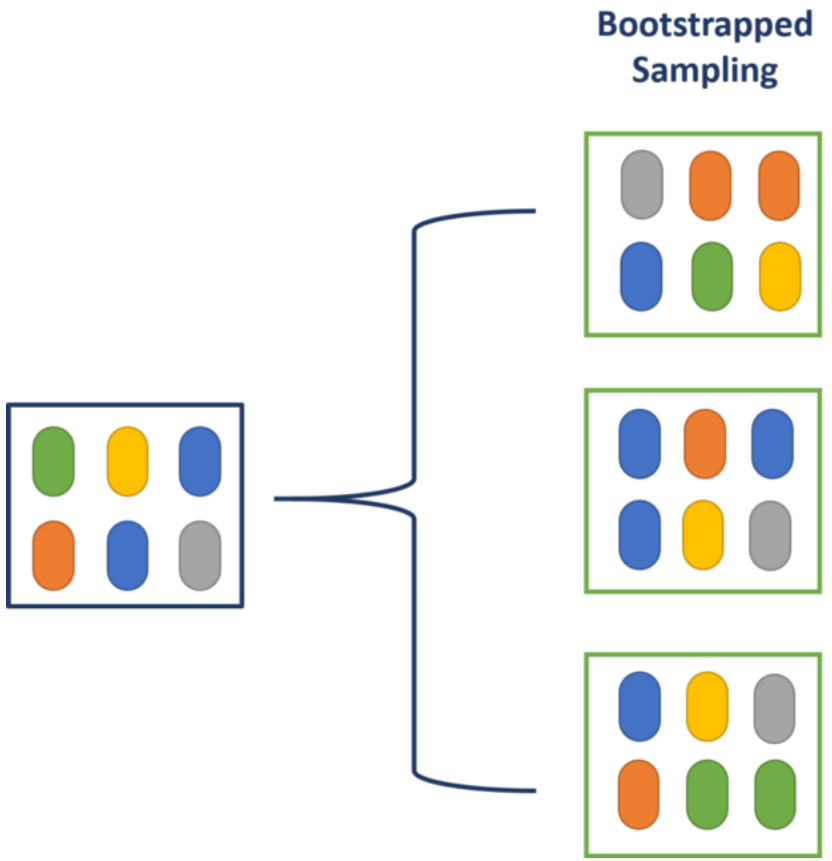
- Random Forests

Bagging

- Decision trees suffer from high variance!
  - If we randomly split the training data into 2 parts, and fit decision trees on both parts, the results of different runs could be quite different
- We would like to have models with low variance
- To solve this problem, we can use bagging (***b***ootstrap ***agg***regat***ing***).

Bootstrapping

- Resampling of the observed dataset (and of equal size to the observed dataset), each of which is obtained by random sampling with replacement from the original dataset.



Bootstrapping

- Draw instances from a dataset *with replacement*
- Probability that we do not pick an instance after N draws

$$\left(1-\frac{1}{N}\right)^{N} \approx e^{-1} = 0.368$$

that is, only 63.2% of instances are used in one draw

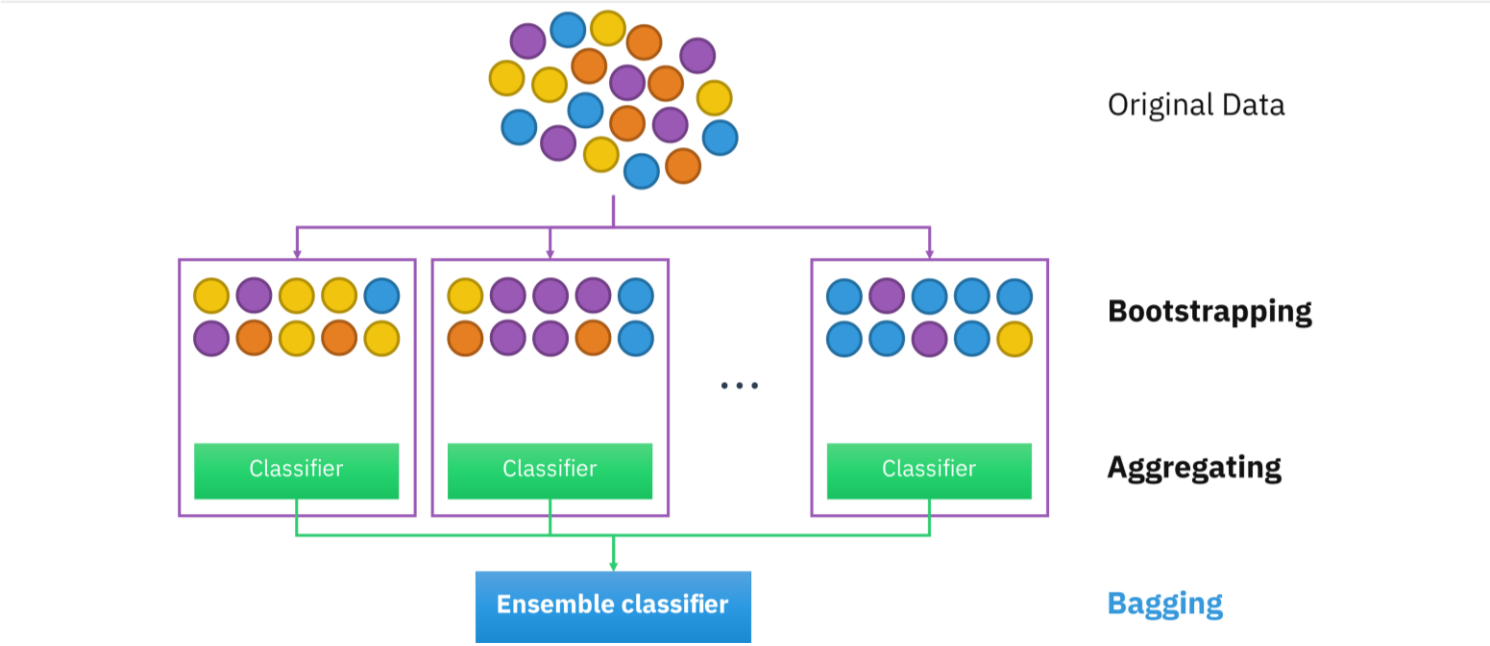
What is bagging?

- Bagging is a powerful idea based on two things:
  - Averaging: reduces variance!
  - Bootstrapping: plenty of training datasets!
- Why does averaging reduces variance?
  - Averaging a set of observations reduces variance.
  - Given a set of *n* independent observations $Z_1, \ldots, Z_n$, each with variance σ^2, the variance of the mean $\overline{Z}$ of the observations is given by σ^2/n.

How does bagging work?

- Generate B different bootstrapped training datasets

- Train the statistical learning method on each of the B training datasets, and obtain the prediction



Bagging for regression trees

- Construct B regression trees using B bootstrapped training datasets
- Average the resulting predictions

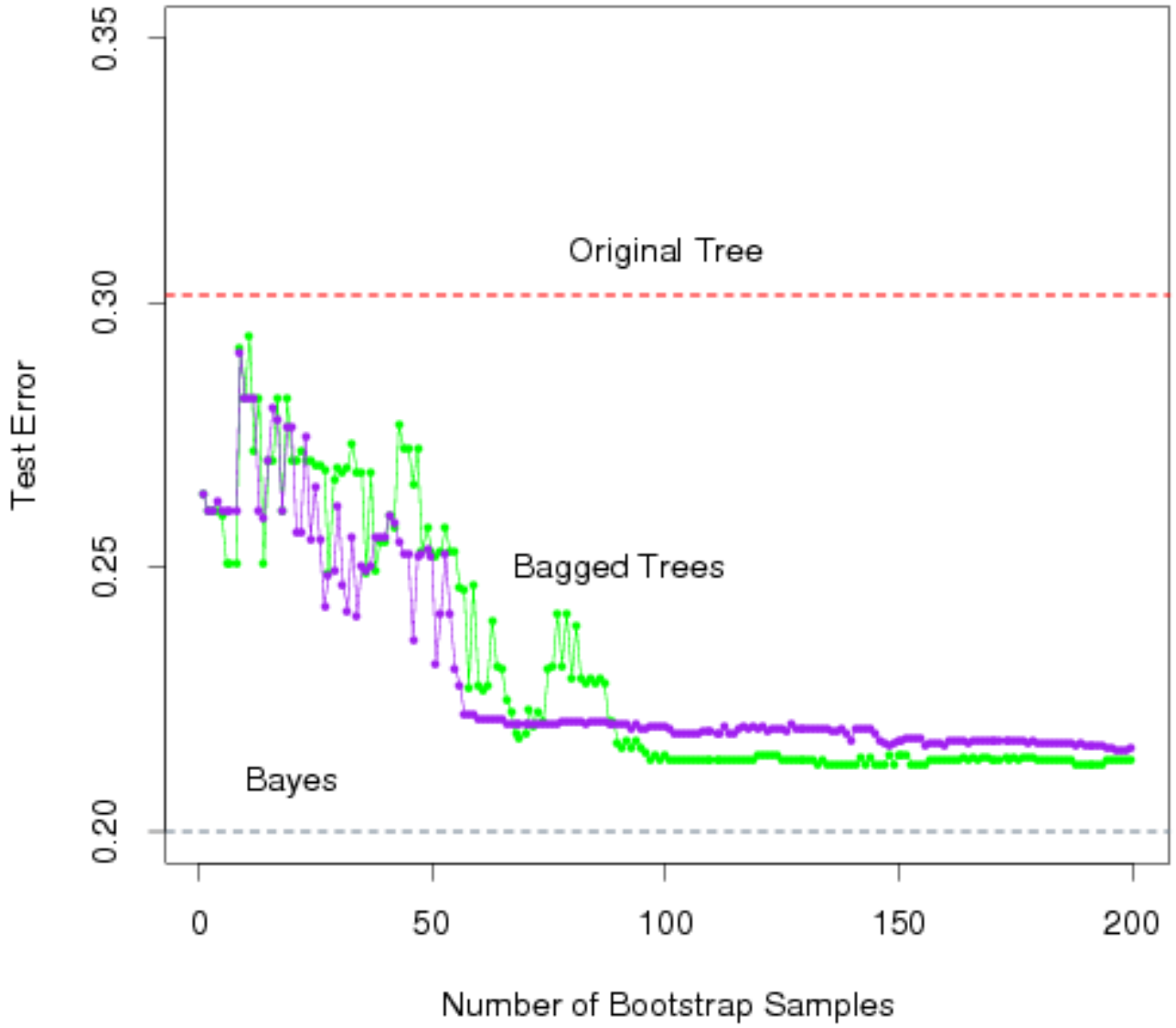
- The trees are not pruned, so each individual tree has high variance but low bias.
- Averaging these trees reduces variance, and thus we end up lowering both variance and bias ☺

Bagging for classification trees

- Construct B decision trees using B bootstrapped training datasets
- For prediction, there are two approaches:
    1. Record the class that each bootstrapped data set predicts and provide an overall prediction to the most commonly occurring one (majority vote).
    2. If our classifier produces probability estimates, we can just average the probabilities and then predict to the class with the highest probability.
- Both methods work well.

A comparison of error rates

- Here the green line represents a simple majority vote approach
- The purple line corresponds to averaging the probability estimates.
- Both do far better than a single tree (dashed red) and get close to the Bayes error rate (dashed grey).



Out-of-bag error estimation

- Since bootstrapping involves random selection of subsets of observations to build a training data set, then the remaining non-selected part could be the testing data.
- On average, each bagged tree makes use of around 1- 1/e ≈ 63% of the observations, so we end up having 1/e ≈ 37% of the observations useful for testing

Variable importance measure

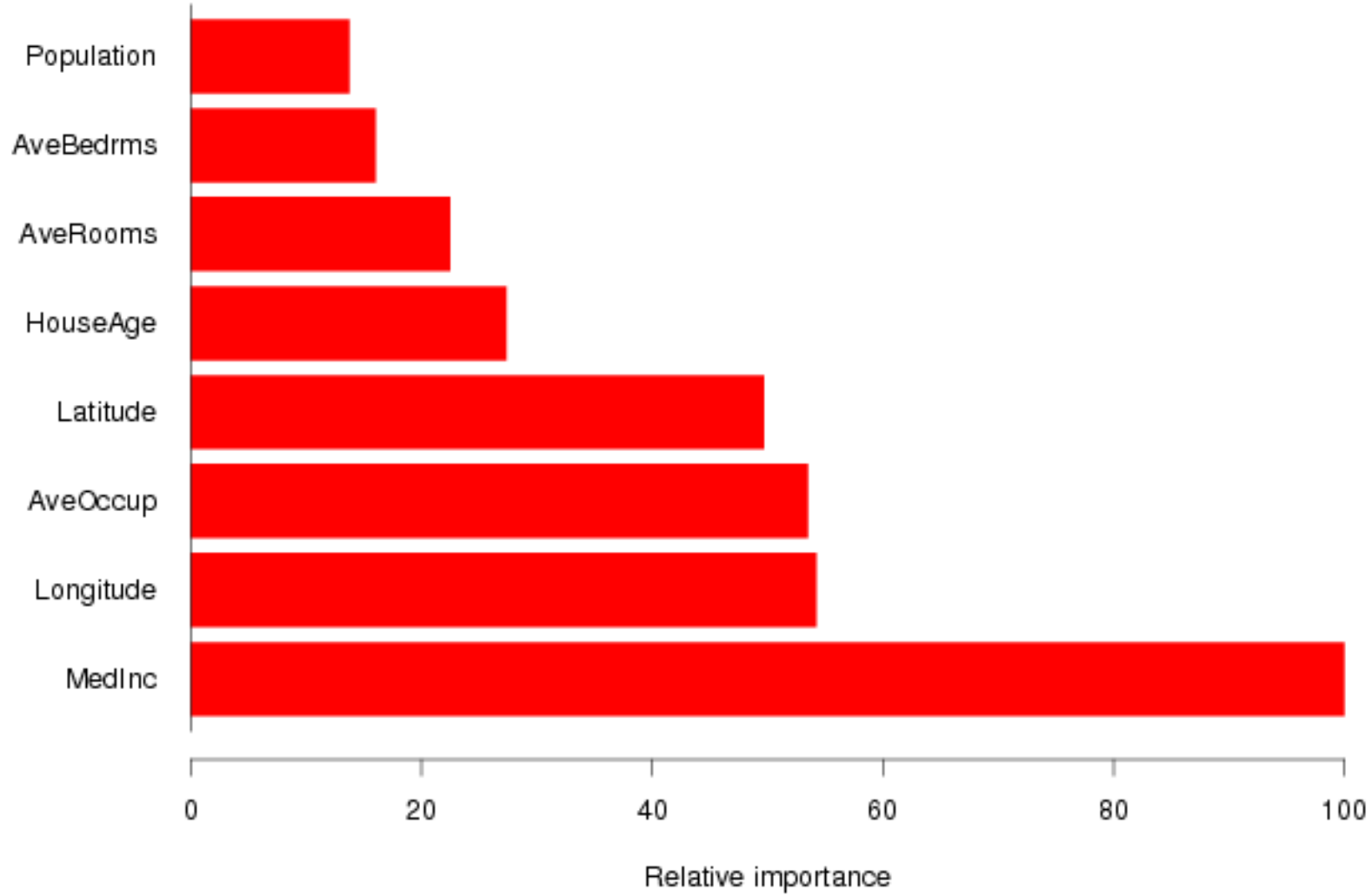
- Bagging typically improves the accuracy over prediction using a single tree, but it is now hard to interpret the model!
- We have hundreds of trees, and it is no longer clear which variables are most important to the procedure
- Thus bagging improves prediction accuracy at the expense of interpretability
- But, we can still get an overall summary of the importance of each predictor using relative influence plots

Relative influence plots

- How do we decide which variables are most useful in predicting the response?
  - We can compute something called relative influence plots.
  - These plots give a score for each variable.
  - These scores represents the decrease in MSE when splitting on a particular variable
  - A number close to zero indicates the variable is not important and could be dropped.
  - The larger the score the more influence the variable has.

Example: Housing data

- Median Income is by far the most important variable.
- Longitude, Latitude and Average occupancy are the next most important.



Random forests

- It is a very efficient statistical learning method
- It builds on the idea of bagging, but it provides an improvement because it de-correlates the trees
- How does it work?
  - Build a number of decision trees on bootstrapped training sample,
  - When building these trees, each time a split in a tree is considered, a random sample of *m* predictors is chosen as split candidates from the full set of *p* predictors.
  - Usually $m \approx \sqrt{p}$ or $m \approx 1 + \log_2 p$

Why are we considering a random sample of *m* predictors instead of all *p* predictors for splitting?

- Suppose that we have a very strong predictor in the data set along with a number of other moderately strong predictors, then in the collection of bagged trees, most or all of them will use the very strong predictor for the first split!
- All bagged trees will look similar. Hence all the predictions from the bagged trees will be highly correlated
- Averaging many highly correlated quantities does not lead to a large variance reduction, and thus random forests “de-correlates” the bagged trees leading to more reduction in variance

Properties

- low classification (and regression) error
- no overfitting
- robust concerning the noise and the number of attributes
- relatively fast
- learning instances not selected with bootstrap replication are used for evaluation of the tree (oob = out-of-bag evaluation)

Out-of-bag evaluation

- on average 1/e ~ 37% of the learning set is not used to train each of the basic classifiers
- classification margin

$$mr(\mathbf{x}, y) = P(h(\mathbf{x}) = y) - \max_{\substack{j=1 \\ j \neq y}}^{c} P(h(\mathbf{x}) = j)$$

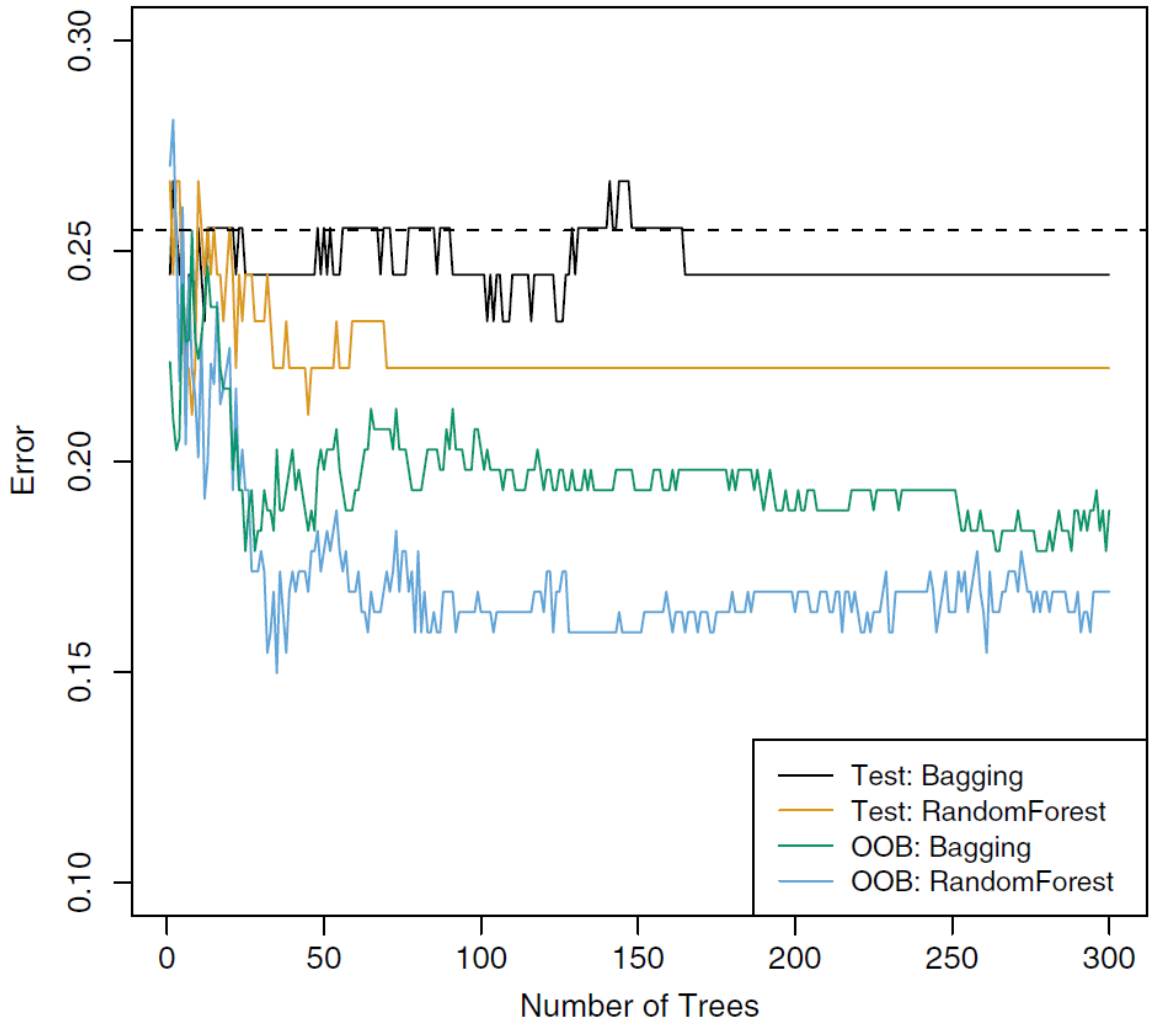
  - mr is estimated with all classifiers where **x** is in oob set
- strength of the forest = average margin over training or OOB set
- correlation of the trees in forest

$$\bar{\rho} = \frac{\mathrm{var}(mr)}{std(h())^2}$$

- we want high strength and low correlation

OOB-error estimate

- with large number of trees, the OOB estimate is roughly equivalent to the CV error estimate
- computationally much cheaper than CV
- still overly optimistic



Heart data set

RF attribute evaluation

- evaluation of attribute A is the difference between
  - strength of the forest and
  - strength of the forest when values of A are randomly shuffled
- evaluated on the OOB set
- detects also strong conditional dependencies
- works also on an instance-level like nomogram (evaluates only the trees where the instance is in the OOB set)

Similarity of instances

- build instance similarity matrix
- when two instances end in the same leaf of the tree we increase their similarity score
- average over all trees gives similarity measure
- we use that similarity measure to:
  - detect outliers
  - determine typical cases for each class
  - scaling
  - missing values
  - clustering
  - visualization

Other possibilities for tree ensembles

- sampling in RF:
  - *p*-sampling without replacement (sampling the proportion of *p* instances, e.g., *p*=10%)
- limiting the size of the trees in RF and bagging
  - more trees needed
- reduced computational complexity
- regularization

Weighting of the trees

- not all trees are equally important (absolutely and in all parts of an instance space)
- weight the trees according to the data
- assume linear combination of base coefficients

$$F(x, a) = a_0 + \sum_{j=1}^{T} a_j \, t_j(x)$$

- solve for coefficients *a*

Penalization

$$\hat{\mathbf{a}} = \arg\min_{a} \frac{1}{N} \sum_{i=1}^{n} L\left(y_{i,} a_0 + \sum_{j=1}^{T} a_j \, t_j(x_i)\right)$$

- direct minimization gives poor generalization, therefore penalize

$$\hat{\mathbf{a}}(\lambda) = \arg\min_{a} \left(\frac{1}{N} \sum_{i=1}^{n} L\left(y_{i,} a_0 + \sum_{j=1}^{T} a_j \, t_j(x_i)\right) + \lambda P(\mathbf{a}) \right)$$

Common penalty functions

- ridge regression

$$P_2(\mathbf{a}) = \sum_{j=1}^{T} |a_j|^2$$

- lasso, sure-shrink

$$P_1(\mathbf{a}) = \sum_{j=1}^{T} |a_j|$$

- solve with gradient descent algorithms (Friedman & Popescu, 2003)

Local weighting

- regularization: global importance of base models
- local importance: local regularization, weighting with margin of similar instances

Locally weighted voting for RF

- observation: not all trees are equally good in all parts of the problem space
- opportunity: use OOB instances to locally evaluate the quality of trees
- locality: forest defines the similarity between instances

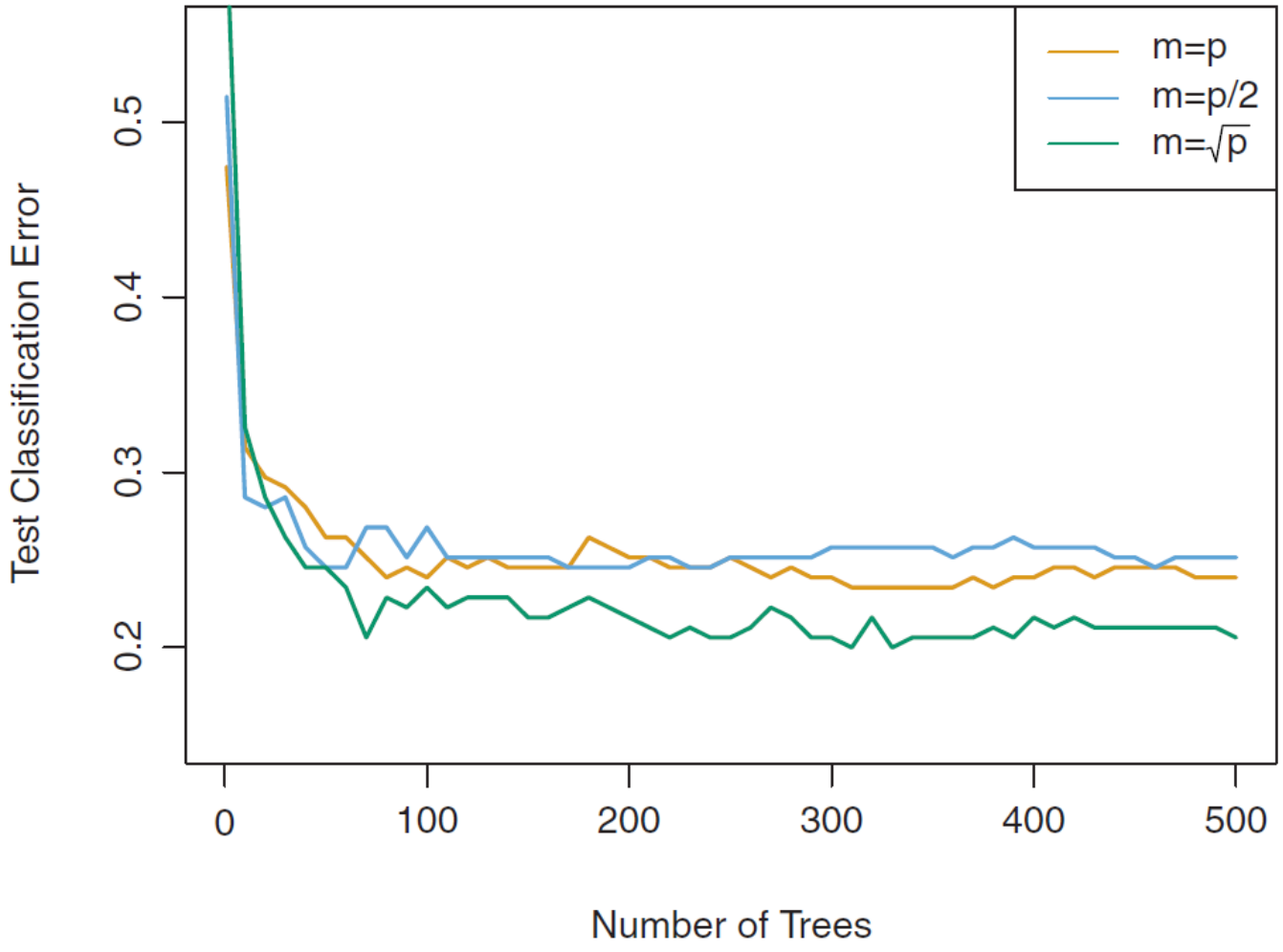
Weighted voting algorithm for RF

- in classification of a new instance
  - find *t* most similar instances
  - classify each of the similar instances with the trees where it is in the OOB set, and record the margin for the trees
  - compute weights of the trees as the average recorded margin (for trees with negative margin set the weight to zero)
  - forest classification is the weighted voting of the trees

Marko Robnik-Šikonja: Improving random forests. In *Proceedings of European conference on machine learning ECML'04*, pp. 359-370. 2004.

Random forest with different values of "*m*"

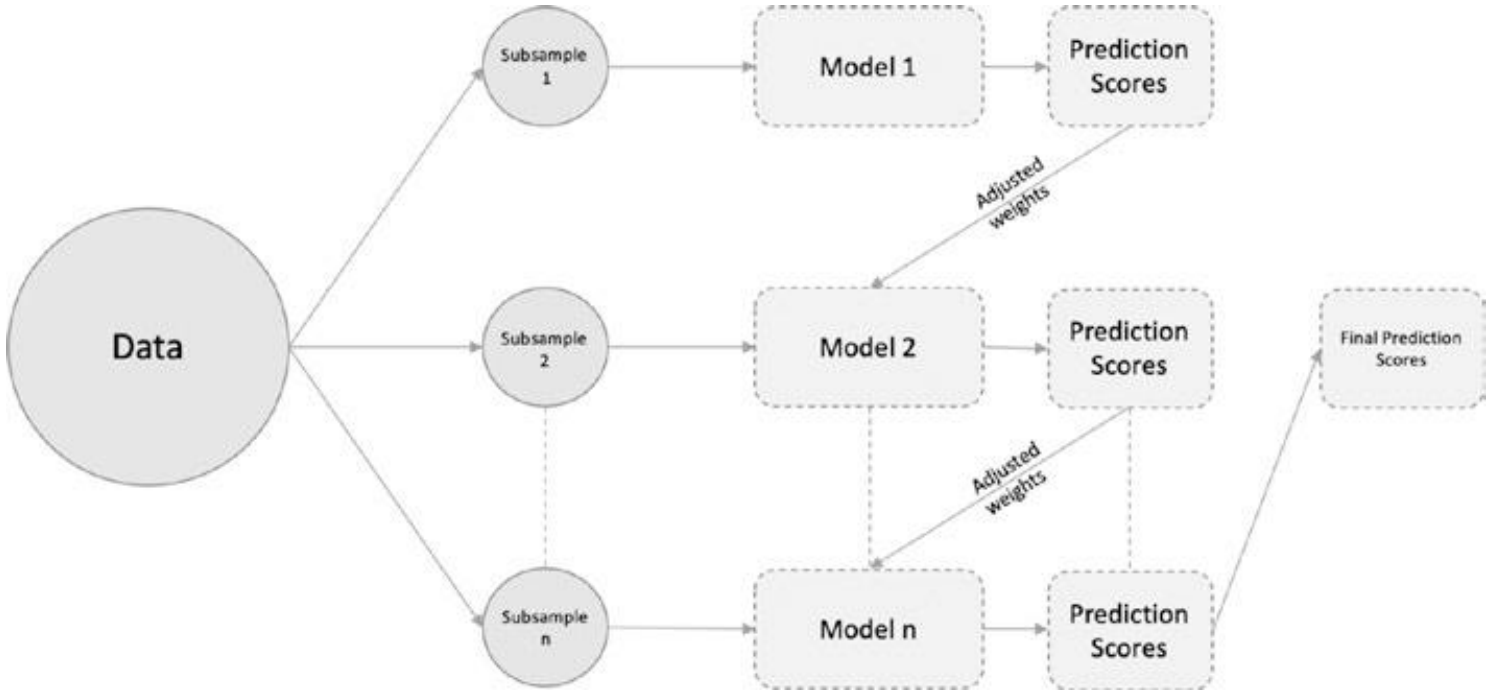
- Notice: when random forests are built using $m = p$, then this amounts to bagging.



Gene expression data (15 classes)

Boosting

- another successful ensemble method
- grows trees sequentially: each added tree uses information about errors of previous trees



Pseudocode for boosting in regression

1. Set $\hat{f}(x) = 0$ and $r_i = y_i$ for all i in the training set.

2. For $b = 1, 2, \ldots, B$, repeat:

   (a) Fit a tree $\hat{f}^b$ with d splits ($d+1$ terminal nodes) to the training data (X, r).

   (b) Update $\hat{f}$ by adding in a shrunken version of the new tree:

$$\hat{f}(x) \leftarrow \hat{f}(x) + \lambda \hat{f}^b(x). \tag{8.10}$$

   (c) Update the residuals,

$$r_i \leftarrow r_i - \lambda \hat{f}^b(x_i). \tag{8.11}$$

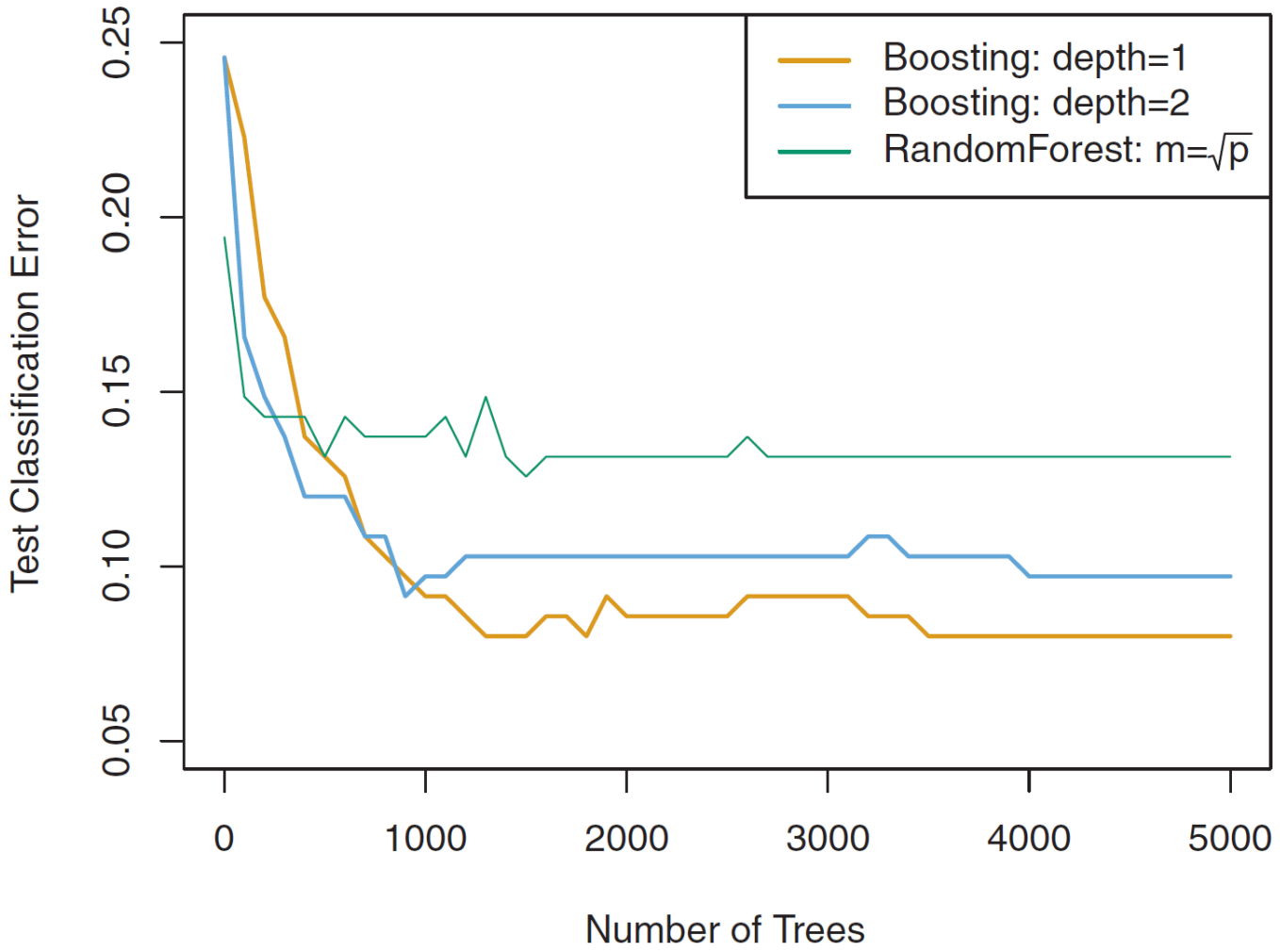
3. Output the boosted model,

$$\hat{f}(x) = \sum_{b=1}^{B} \lambda \hat{f}^b(x). \tag{8.12}$$

Boosting

- each tree takes into account residuals (i.e. errors) of previous trees
- each tree is small, containing only d splits (e.g., d=1, decision stumps)
- learning is slow, controlled by λ
- Parameters of boosting in regression
  - The number of trees *B*, selected with, e.g., CV; boosting can overfit.
  - The shrinkage parameter λ, a small positive number (e.g., 0.01 or 0.001), problem dependent; small λ requires large B to achieve good performance
  - The number *d* of splits in each tree, which controls the complexity of the boosted ensemble. Often $d = 1$ works well, but *d* also controls interaction order (*d* splits can contain at most *d* variables).

Boosting performance



Gene expression data (15 classes)
error of single tree is approx. 0.24, std. error around 0.02

Boosting in classification

- AdaBoost, Freund & Shapire, ICML, 1996
  - training instances are weighted according to the success of their classification in the previous iteration
    - increase weight of misclassified instances
    - decrease weight of correctly classified instances
    - the learning focus is transferred to the most difficult instances
  - final classification is a weighted voting of basic classifiers
- deterministic algorithm, works because training sets are different
- mostly better than bagging
- this original version can suffer from overfitting but there are better variants

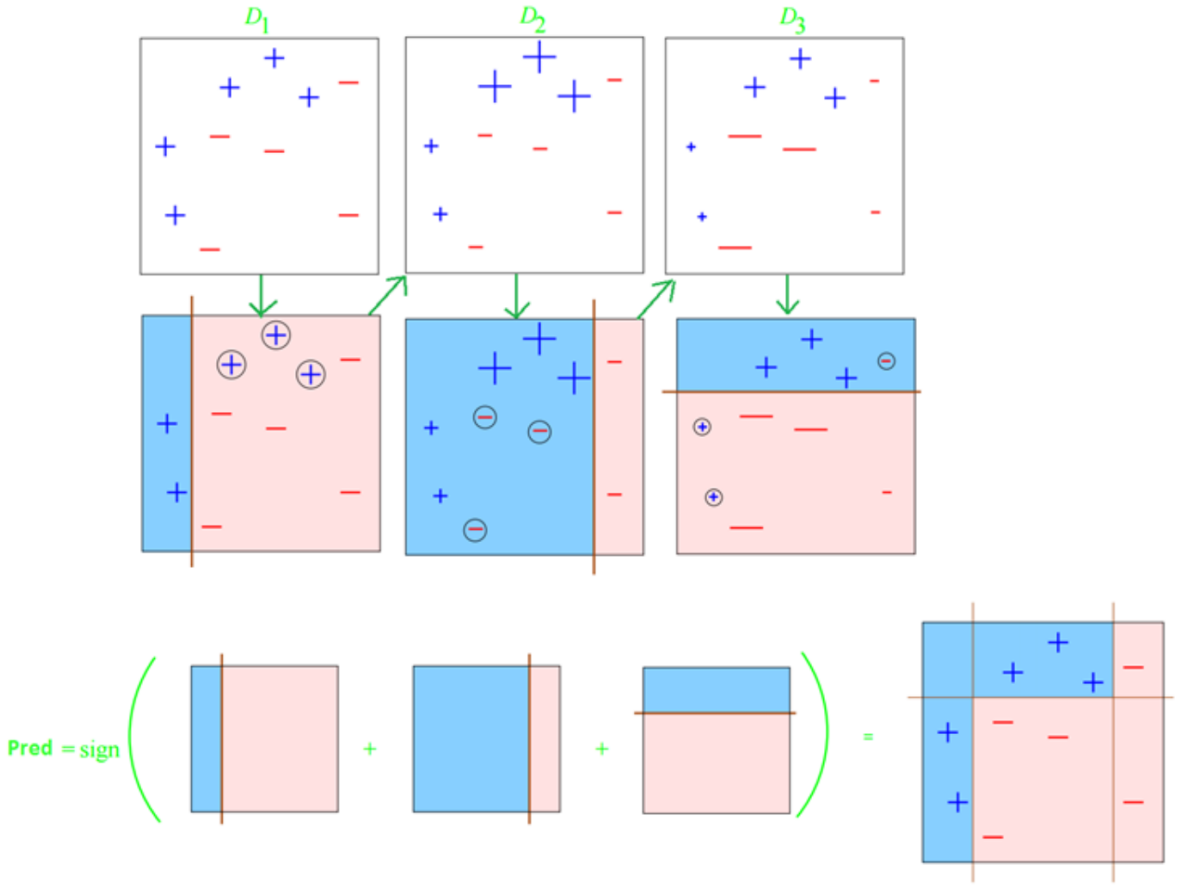
AdaBoost (Freund and Schapire, 1996)

- Given a set of *d* class-labeled instances, $(\boldsymbol{X_1}, y_1), \ldots, (\boldsymbol{X_n}, y_n)$
- Initially, all the weights of instances are set the same (*1/n*)
- Generate *k* classifiers in *k* rounds. At round *i*,
  - Instances from D are sampled (with replacement) or reweighted to form a training set D_i of the same size
  - Each instance's chance of being selected is based on its weight
  - A classification model M_i is derived from D_i
  - Its error rate is calculated using D_i as a test set
  - If an instance is misclassified, its weight is increased, otherwise it is decreased
- Error rate: $err(\boldsymbol{X_j})$ is the misclassification error of instance $\boldsymbol{X_j}$.
  Classifier M_i error rate is the sum of the weights of the misclassified instances:

$$error(M_i) = \sum_{j}^{d} w_j \times err(\mathbf{X_j})$$

- The weight of classifier M_i's vote is $\log \frac{1 - error(M_i)}{error(M_i)}$

AdaBoost Example



XGBoost – eXtreme Gradient Boosting

Additive model with loss L:

$$\min_{\alpha_{n=1:N},\beta_{n=1:N}} L\left(y, \sum_{n=1}^{N} \alpha_n f(x, \beta_n)\right)$$

GB approximately solves this objective iteratively and greedily:

$$\min_{\alpha_n,\beta_n} L\left(y, f_{n-1}((x) + \alpha_n f_n(x, \beta_n)\right)$$

- Uses small trees as f(x, β_n)
- Uses a number of computational shortcuts to allow for fast computation:
  - Samples both the instances and features
  - Uses extensive parallelism in tree building and loss optimization

Chen & Guestrin(2016), XGBoost: A Scalable Tree Boosting System. *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* https://arxiv.org/abs/1603.02754
https://xgboost.readthedocs.io/en/latest/build.html#r-package-installation

XGBoost overview

- Practically the most successful ensemble approach, giving excellent predictive accuracy, speed and parallelization, good handling of tabular data, and built-in regularization
- You can think of XGBoost as an engine that builds many small decision/regression trees, each one correcting the errors of the previous ones.
- XGBoost builds a series of small trees. Each tree asks:
  - “Given how the previous trees failed, what small adjustment can I make to reduce the remaining error?”
- It uses gradient information to know *how* to adjust and regularization to avoid overfitting.
- The final model is:

$$\hat{y} = f_1(x) + f_2(x) + \cdots + f_K(x)$$

where each $f_k$is a tiny regression/classification tree.

XGBoost Workflow

- Initialize predictions (baseline model)
- Compute gradients and Hessians
- Build a tree using gain optimization
- Compute leaf weights analytically
- Update predictions using learning rate
- Repeat for multiple boosting rounds

XGBoost Step by Step, 1 & 2 /6

1. Start with a prediction

For regression, the model starts by predicting a **constant value**, usually the mean of the target y.
For classification, it starts from the initial log-odds.

2. Compute the residuals (the errors)

Instead of fitting trees directly to the target values, XGBoost fits trees to the **gradients of the loss function**.

For each sample i:

$$g_i = \frac{\partial L(y_i, \hat{y}_i)}{\partial \hat{y}_i}$$

$$h_i = \frac{\partial^2 L(y_i, \hat{y}_i)}{\partial \hat{y}_i^2}$$

Where
g_i =gradient (how much the prediction is wrong) and
h_i =second derivative (curvature — how confident the loss curve is)

XGBoost Step by Step, 3 / 6

3. Grow a decision tree using gradients & Hessians

- XGBoost grows a tree **node by node**.
  At each possible split, it computes a **gain**:

$$\text{Gain} = \frac{1}{2}\left[\frac{G_L^2}{H_L + \lambda} + \frac{G_R^2}{H_R + \lambda} - \frac{G^2}{H + \lambda}\right] - \gamma$$

Where:

$G = \sum g_i$,$H = \sum h_i$over samples in the node

L and R denote left and right splits

λ= L2 regularization

γ= penalty for adding a leaf

- A split is accepted only if **Gain > 0**. This prevents overfitting.

XGBoost Step by Step, 4 & 5 /6

4. Compute leaf values analytically

- Each leaf gets a weight:

$$w = -\frac{G}{H + \lambda}$$

- These weights represent how much the model should adjust predictions for samples falling into that leaf.

5. Update predictions

- New prediction = previous prediction + learning_rate × leaf_value.
- This ensures controlled incremental learning.

XGBoost Step by Step, 6/6

6. Repeat for many boosting rounds

- Each new tree corrects the residual errors of the previous ensemble.
- Eventually, XGBoost builds a **strong model** from many **weak trees**.

Key properties of XGBoost

1. Regularization

- XGBoost adds L1/L2 penalties on tree leaf weights and tree complexity (number of leaves). This makes it **less prone to overfitting** than older implementations.

2. Parallelized split finding

- It uses histogram-based split computation to speed up training.

3. Handling missing values

- XGBoost learns the **optimal direction** (left or right) for missing data at each split.

4. Shrinkage (learning rate)

- Each tree's contribution is scaled down, improving generalization.

5. Column subsampling

- Randomly selecting subsets of features reduces correlation between trees.

6. Weighted quantile sketch

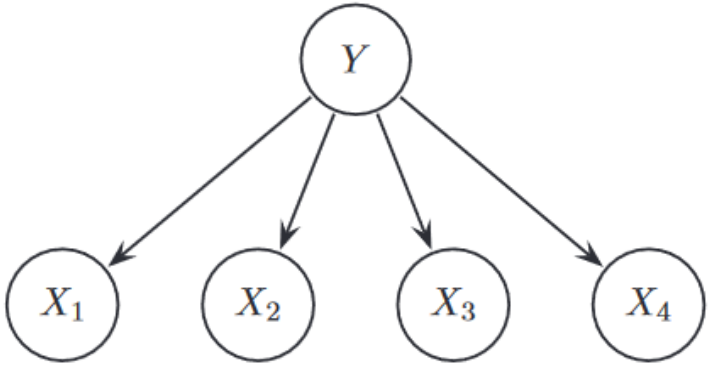
- Efficiently handles large datasets with weighted instances.

Small Example: First Tree

- Toy dataset:
- x: [1, 2, 3, 4]
- y: [2, 3, 2, 5]

- Initial prediction = mean(y) = 3
- Residuals = [-1, 0, -1, 2]
- Tree splits to reduce error:
  - Left leaf weight = avg(residuals_left)
  - Right leaf weight = avg(residuals_right)

Naïve Bayes based ensembles



- Naive Bayes is a probabilistic classifier

$$\begin{aligned} \underset{y}{\operatorname{argmax}}\, P(y \mid \mathbf{x}) &= \underset{y}{\operatorname{argmax}}\, P(y, \mathbf{x}) / P(\mathbf{x}) \\ &= \underset{y}{\operatorname{argmax}}\, P(y, \mathbf{x}). \end{aligned}$$

- assuming that the attributes are independent given the class

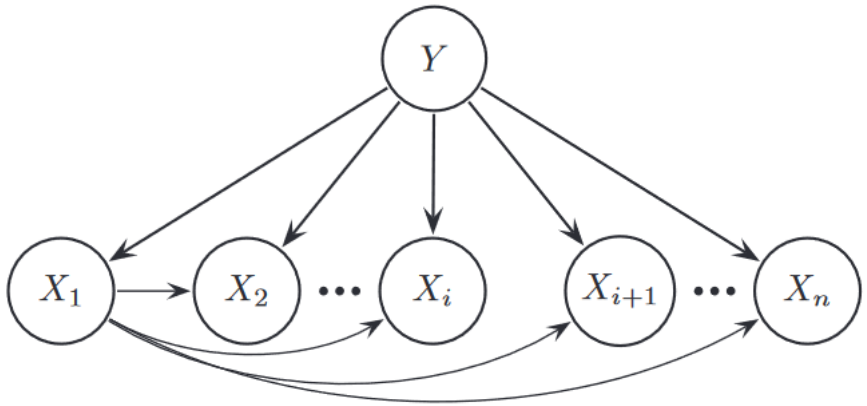
$$\hat{P}(y, \mathbf{x}) = \hat{P}(y) \prod_{i \in N} \hat{P}(x_i \mid y),$$

Semi naïve Bayes (SNB)

- besides the class, SNB allows dependence on some attributes

$$\hat{P}(y, \mathbf{x}) = \hat{P}(y) \prod_{i \in N} \hat{P}(x_i \mid y, \pi(x_i)),$$

- Example: 1-dependence estimator (ODE), where X_1 is “super-parent”

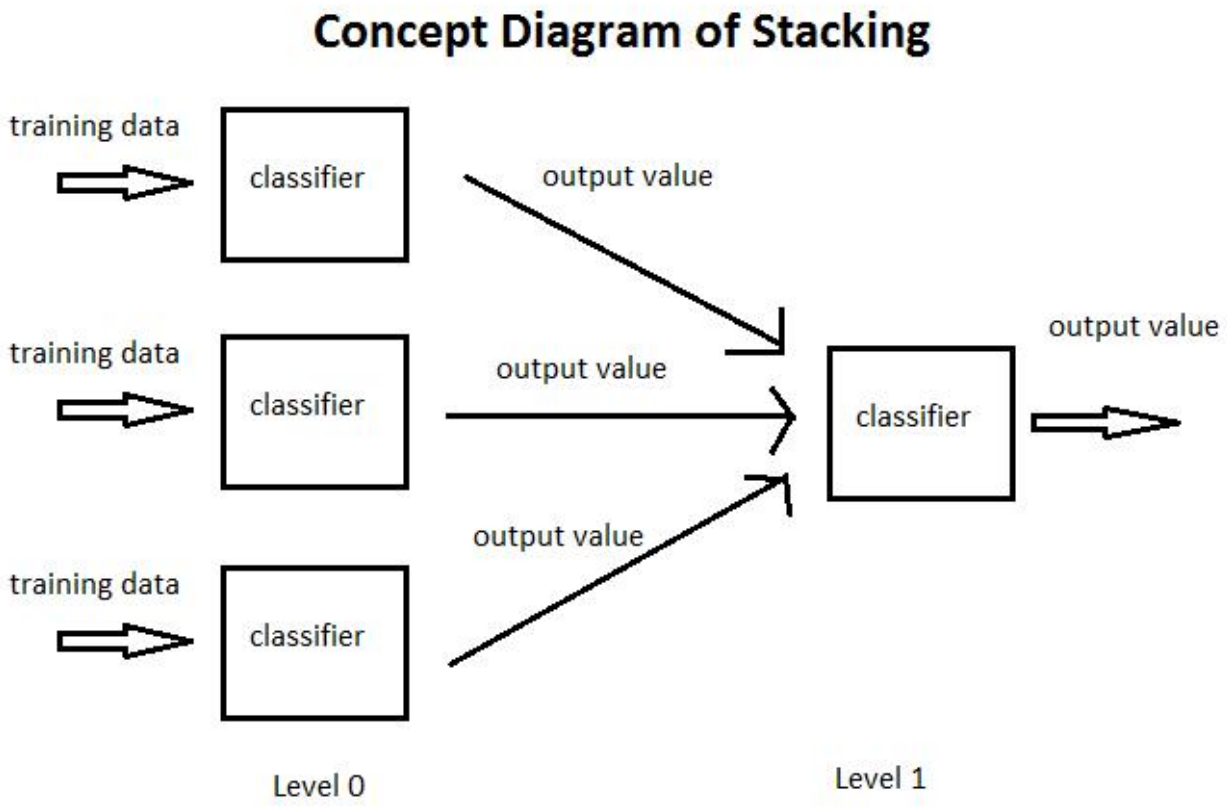


AODE ensemble

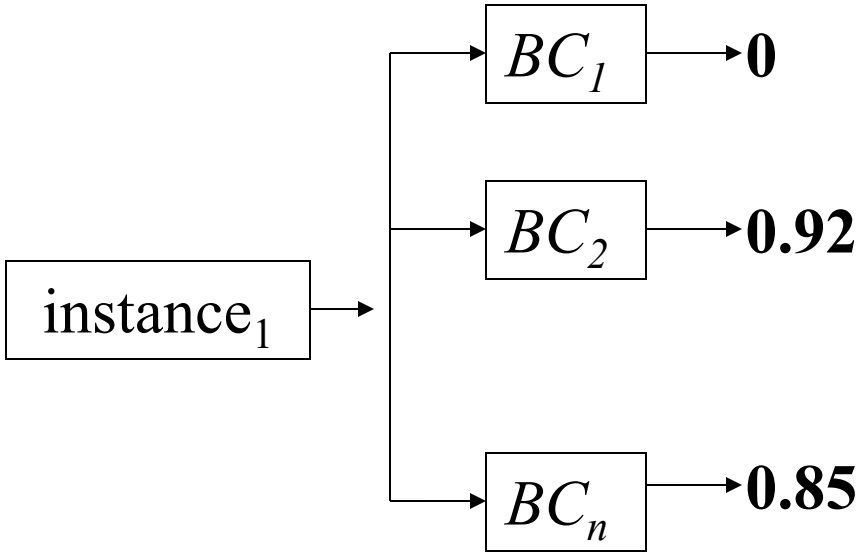
- Averaged One-Dependence Estimator (AODE) (Webb et al. 2005)
- SPODE: Super-Parent One Dependence Estimator – Semi naive Bayes where attributes are dependent on class and one more attribute
- AODE is an ensemble of SPODE classifiers, where all attributes in turn are used in SPODE classifier and their results are averaged
- Compared to naive Bayes, it has higher variance but lower bias

Stacking

- A method to combine heterogeneous predictors
- Predictions of base learners (level-0 models) are used as input for meta learner (level-1 model)
- Base learners are usually different learning schemes

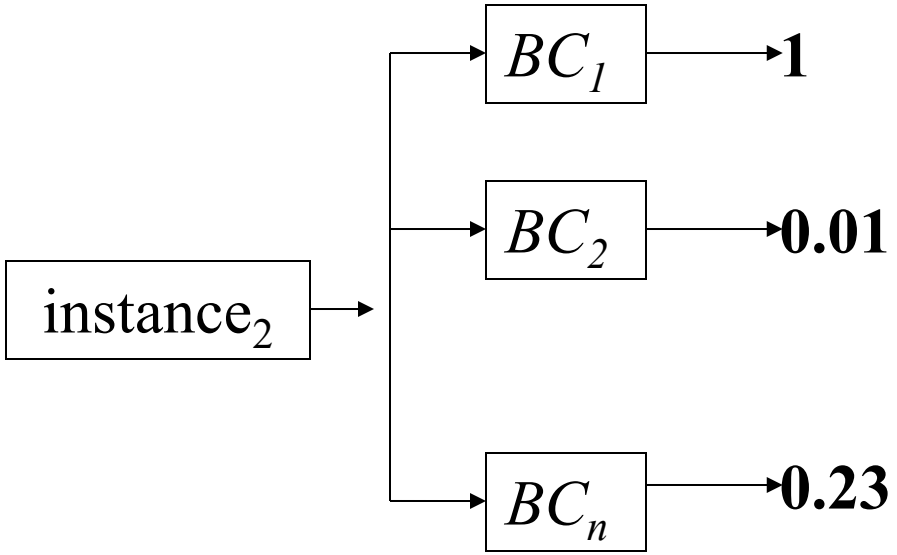


Stacking scheme



meta instances	BC_1	BC_2	…	BC_n	*Class*
instance$_1$	**0**	**0.92**		**0.85**	**1**

Stacking



meta instances	BC_1	BC_2	…	BC_n	*Class*
instance$_1$	**0**	**0.92**		**0.85**	**1**
instance$_2$	**1**	**0.01**		**0.23**	**0**

Stacking

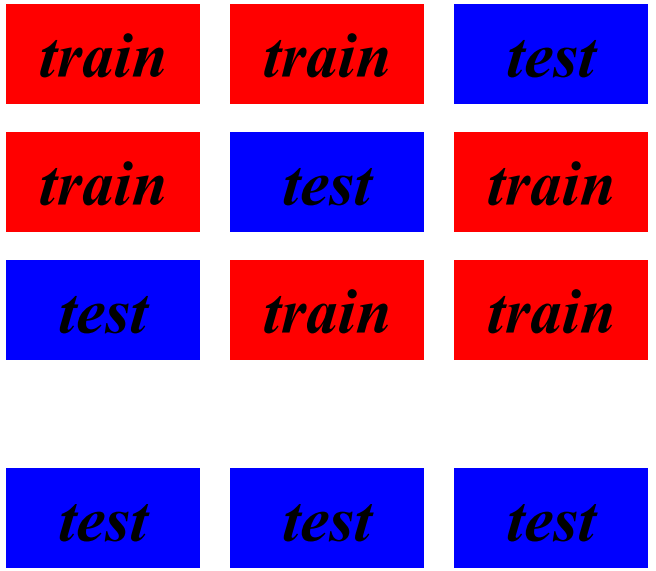
Meta Classifier



meta instances	BC_1	BC_2	…	BC_n	*Class*
$instance_1$	**0**	**0.92**		**0.85**	**1**
$instance_2$	**1**	**0.01**		**0.23**	**0**

Actual stacking

- Predictions on the training data can’t be used to generate data for level-1 model! Why not?
- The reason is that the level-0 classifier that better fit training data will be chosen by the level-1 model!
- Thus, k-fold cross-validation-like scheme is employed. An example for k = 3!

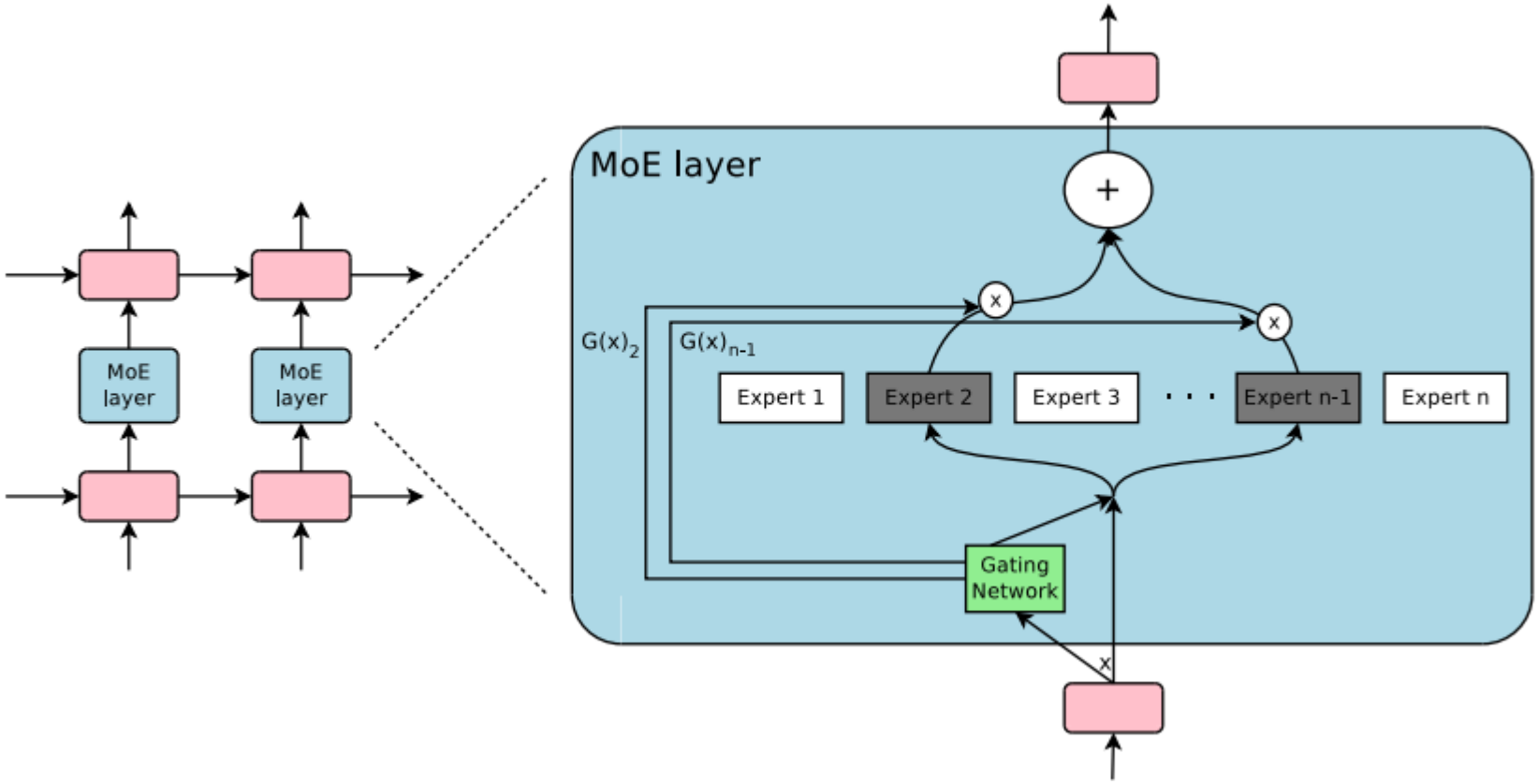


Stacking meta-learner

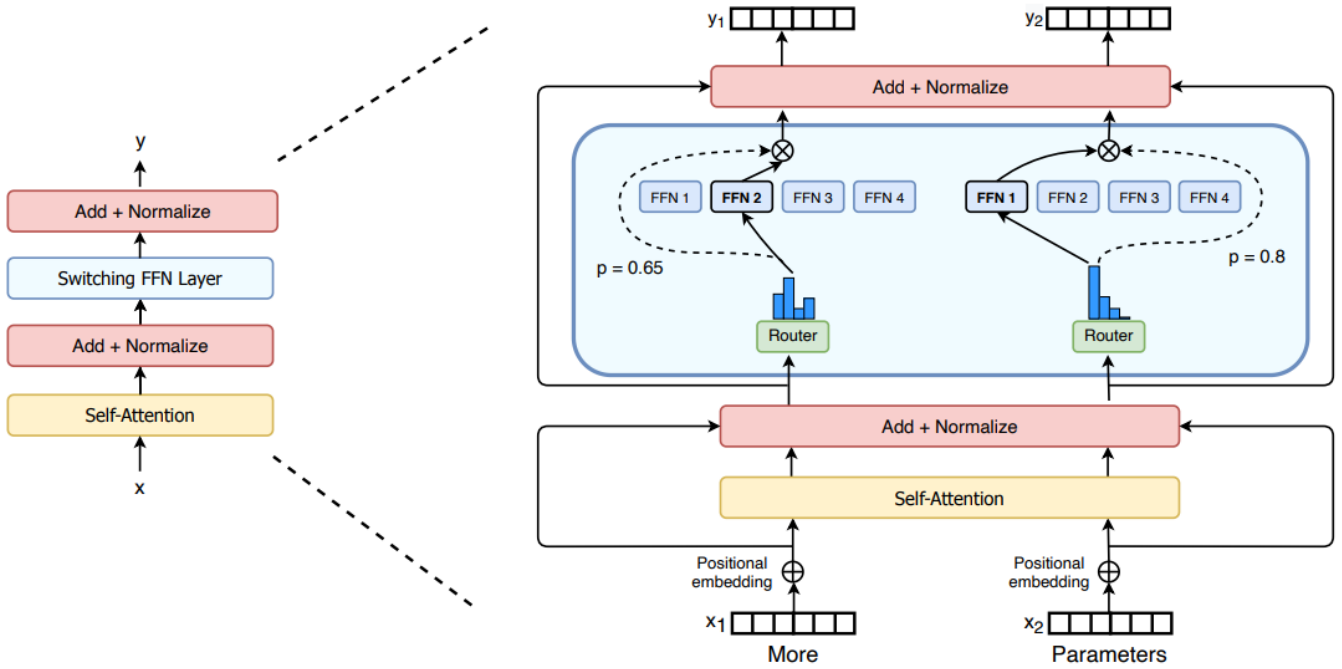
- Which algorithm to use to generate meta learner?
- In principle, any learning scheme can be applied
- For level-1 classifier Ting & Witten (1999) recommend multiple response linear regression (MRLE, note this is a regressor)
  - a classification problem with C classes is transformed into C linear regression problems, where response for problem *i* is 1 if the class equals *i*, otherwise it is 0
  - to classify a new instance employ all *C* linear models, the prediction with highest value is selected as the output

Mixture of Experts (MoE)

- Ensemble technique, useful in very large problems



MoE in transformers



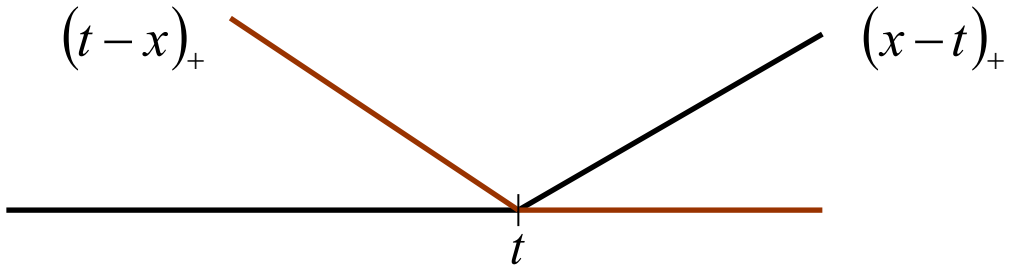
See https://huggingface.co/blog/moe

MARS - Multivariate Adaptive Regression Splines

- Generalization of stepwise linear regression
- Modification of trees to improve regression performance
- Able to capture additive structure
- Not tree-based

MARS base models

- Additive model with adaptive set of basis vectors
- Basis built up from simple piecewise linear functions



- Set "*C*" represents candidate set of linear splines, with "knees" at each data point X_j.
- Models are built with elements from C or their products.

$$C = \left\{ (X_j - t)_+, (t - X_j)_+ \right\}_{t \in \{x_{1j}, x_{2j}, \ldots x_{Nj}\} j=1,2,\ldots,p}$$

- Basis collections C: |C| = 2 * N * p

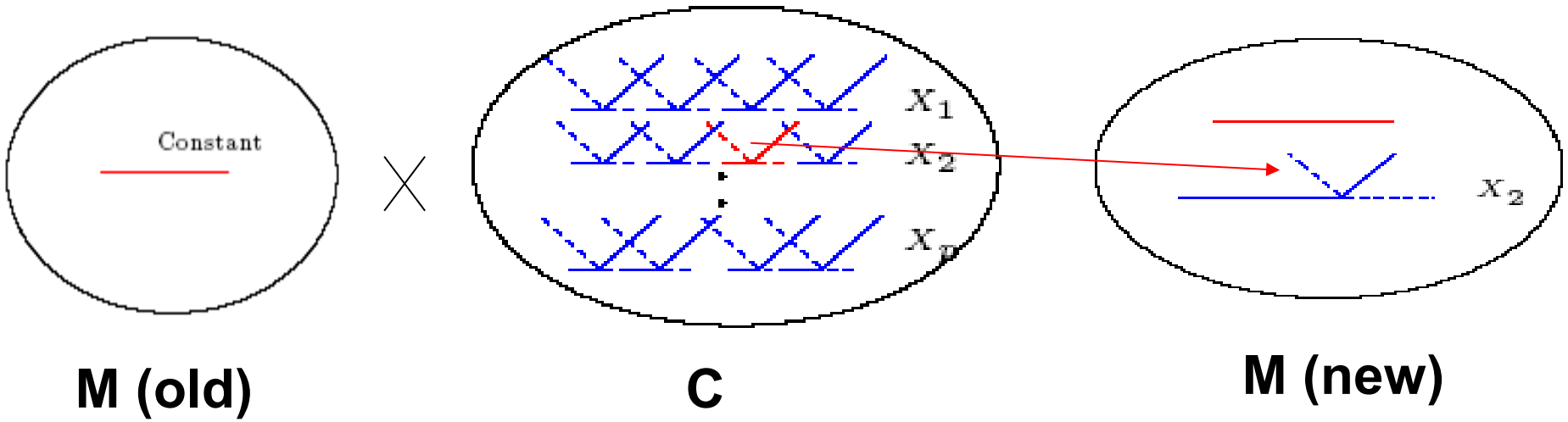
MARS procedure

Model has the form: $f(X) = \beta_0 + \sum_{m=1}^{M} \beta_m h_m(X)$

1. Given a choice for the h_m, the coefficients β are chosen by the standard linear regression.
2. Start with $h_0(X) = 1$
   All functions in *C* are candidate functions.
3. At each stage, consider as a new basis function pair all products of a function h_m in the model set *M*, with one of the reflected pairs in *C*.
   $\beta_{M+1} h_l(X) \cdot (X_j - t)_+ + \beta_{M+2} h_l(X) \cdot (t - X_j)_+, h_l \in M$
4. We add to the model terms of the form:

   $h_m(X) \cdot (t - X_j)_+$ $\qquad$ $h_m(X) \cdot (X_j - t)_+$

MARS, step 1



- On each step, add the term, which reduces residual error most, into *M*
- Repeat steps (until, e.g., *|M|* >= threshold)

MARS, choosing number of terms

- Large models can overfit.
- Backward deletion procedure: delete terms which cause the smallest increase in residual squared error, to get a sequence of models.
- Pick Model using Generalized Cross Validation:

$$GCV(\lambda) = \frac{\sum_{i=1}^{N} \left(y_i - \hat{f}(x_i)\right)^2}{(1 - M(\lambda)/N)^2}$$

- $M(\lambda)$ is the effective number of parameters in the model.
  C=3, r is the number of basis vectors, and *K* knots

$$M(\lambda) = r + cK$$

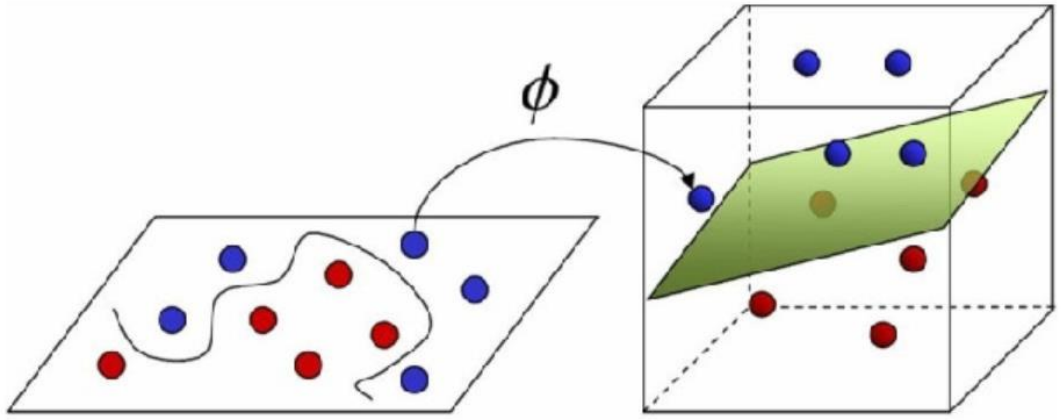
- Choose the model which minimizes $GCV(\lambda)$

MARS summary

- Basis functions operate locally
- Forward modeling is hierarchical, multiway products are built up only from existing terms
- Each input appears only once in each product
- Useful option is to set limit on order of operations. Limit of two allows only pairwise products. Limit of one results in an additive model

University of Ljubljana, Faculty of Computer and Information Science

Kernel methods



Prof Dr Marko Robnik-Šikonja

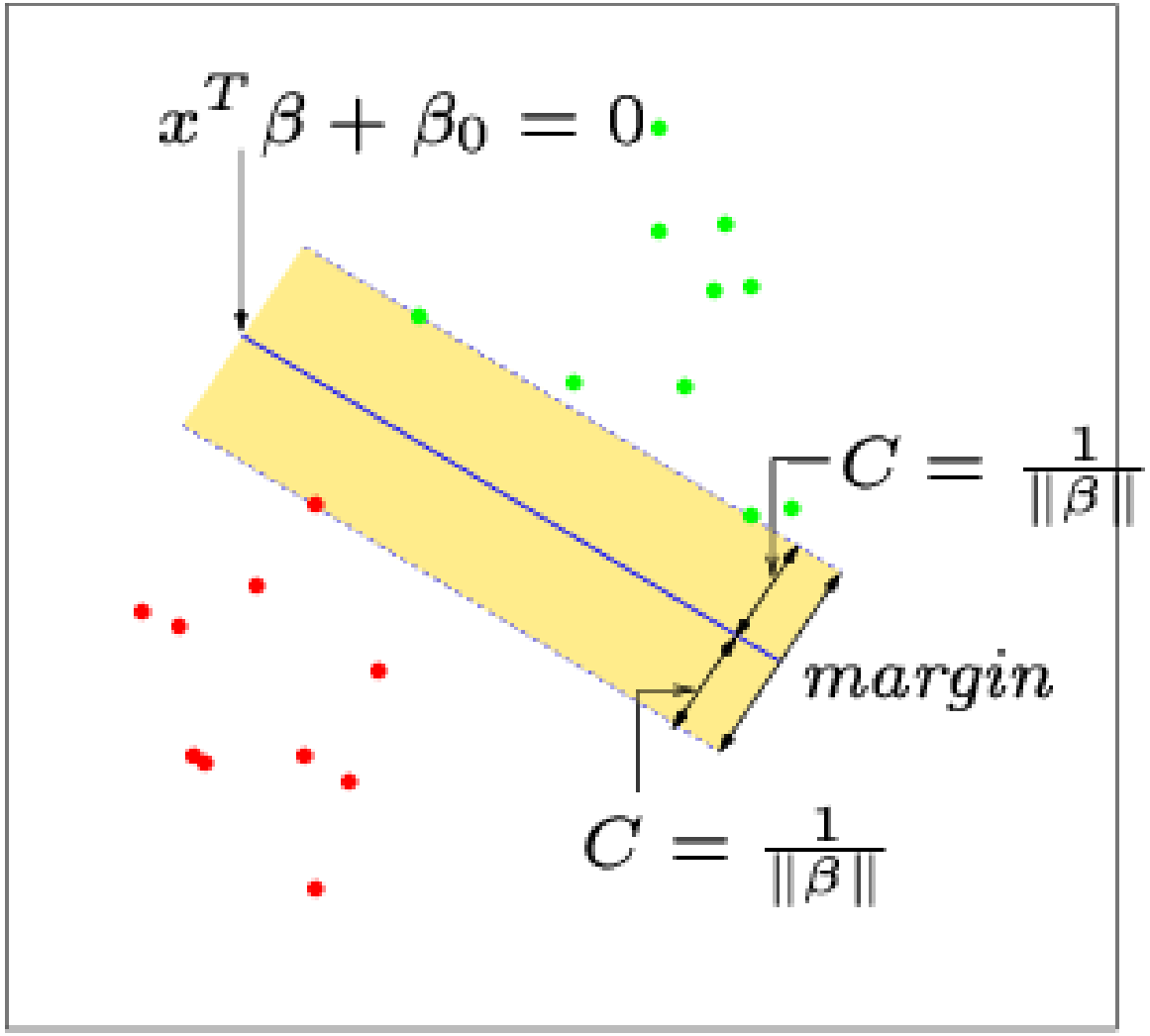
Intelligent Systems, Edition 2023

Support vector machines

- Imagine a situation where you have a two class classification problem with two predictors X_1 and X_2.
- Suppose that the two classes are “linearly separable” i.e. one can draw a straight line in which all points on one side belong to the first class and points on the other side to the second class.
- Then a natural approach is to find the straight line that gives the biggest separation between the classes, i.e. the points are as far from the line as possible
- This is the basic idea of support vector classifiers.

An illustration

- *C* is the minimum perpendicular distance between each point and the separating line.
- We find the line which maximizes *C*.
- This line is called the “optimal separating hyperplane”
- The classification of a point depends on which side of the line it falls on.



More than two dimensions

- This idea works just as well with more than two predictor variables.
- For example, with three predictors you want to find the plane that produces the largest separation between the classes.
- With more than three dimensions it becomes hard to visualize a plane but it still exists. In general they are caller hyper-planes.

Non-separating classes

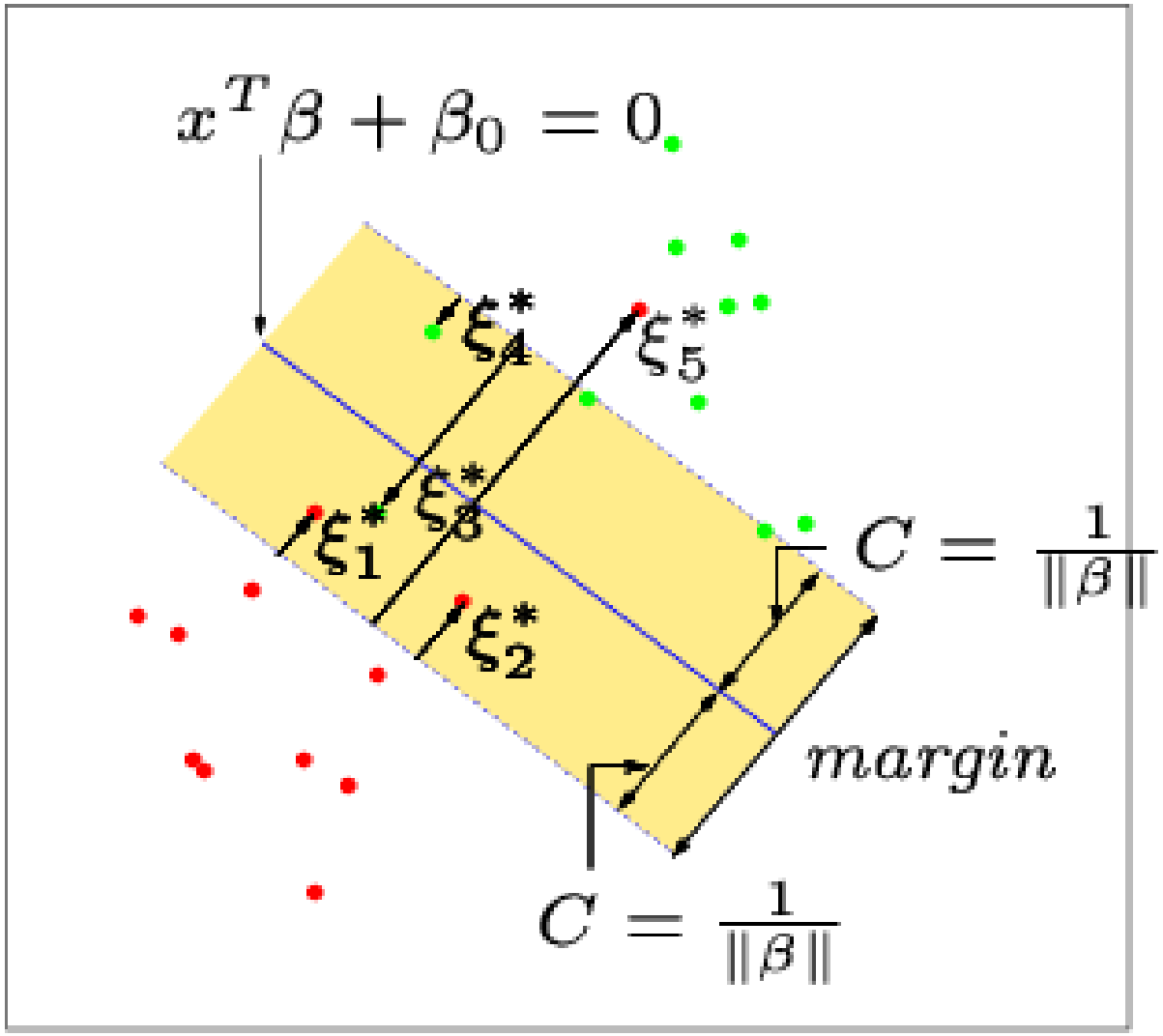
- In practice, it is not usually possible to find a hyper-plane that perfectly separates two classes.
- In other words, for any straight line or plane that we draw, there will always be at least some points on the wrong side of the line.
- In this situation, we try to find the plane that gives the best separation between the points that are correctly classified, subject to the points on the wrong side of the line not being off by too much.
- It is easier to see with a picture!

Non-separating example

- Let ξ^*_i represent the amount that the *i*-th point is on the wrong side of the margin (the dashed line).
- Then we want to maximize *C,* subject to

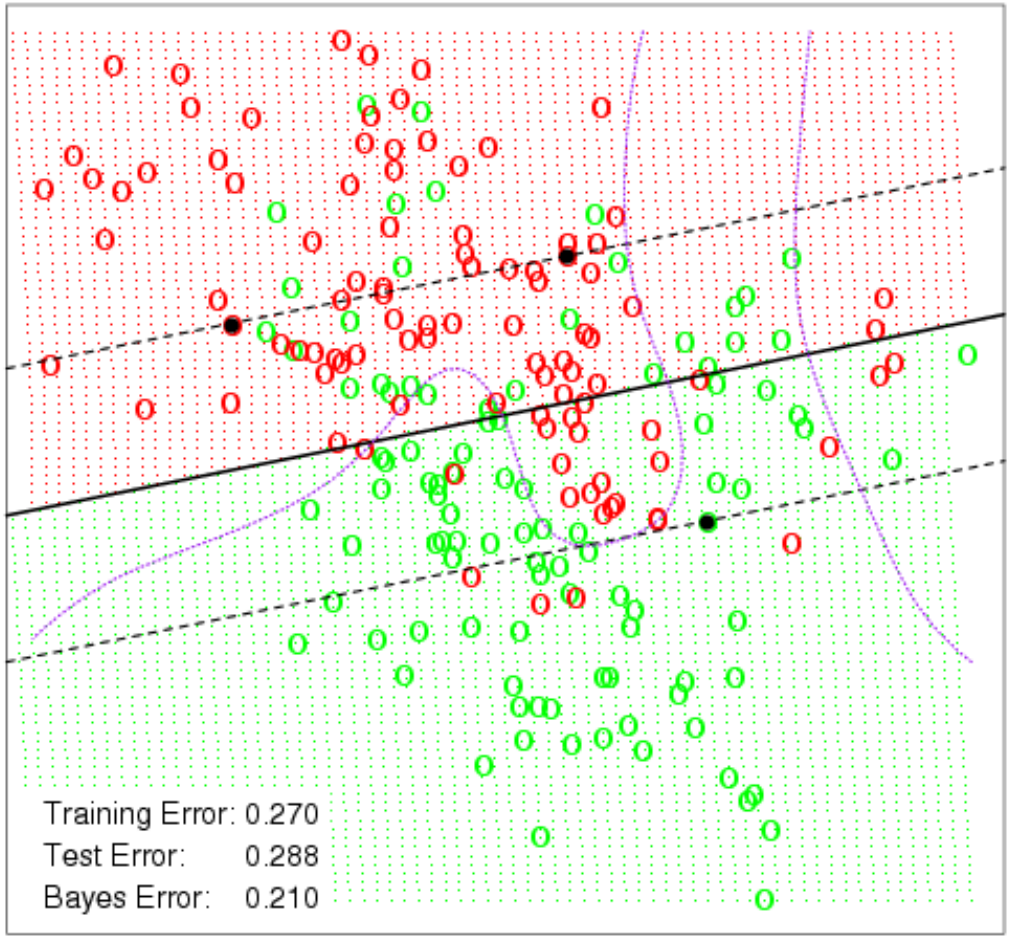
$$\frac{1}{C}\sum_{i=1}^{n}\xi_i^* \leq \text{Constant}$$

- The constant is a tuning parameter that we choose.



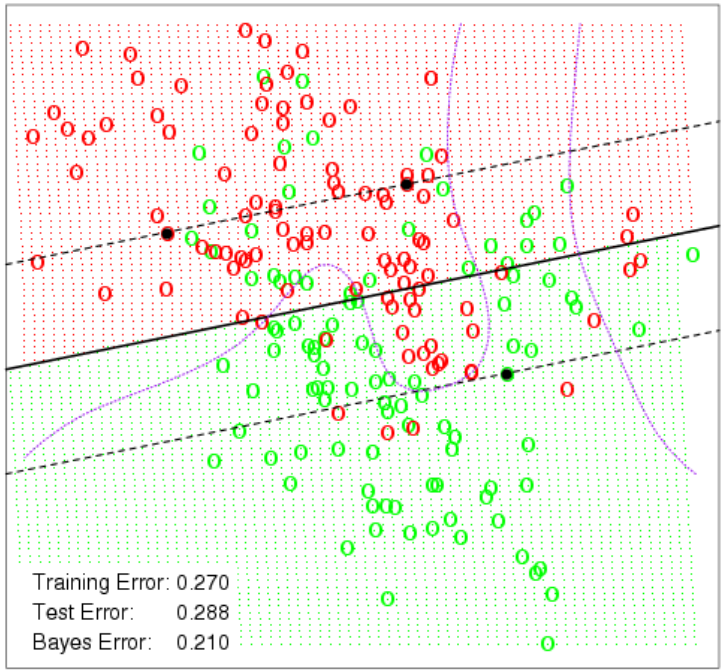
A simulation example with a small constant

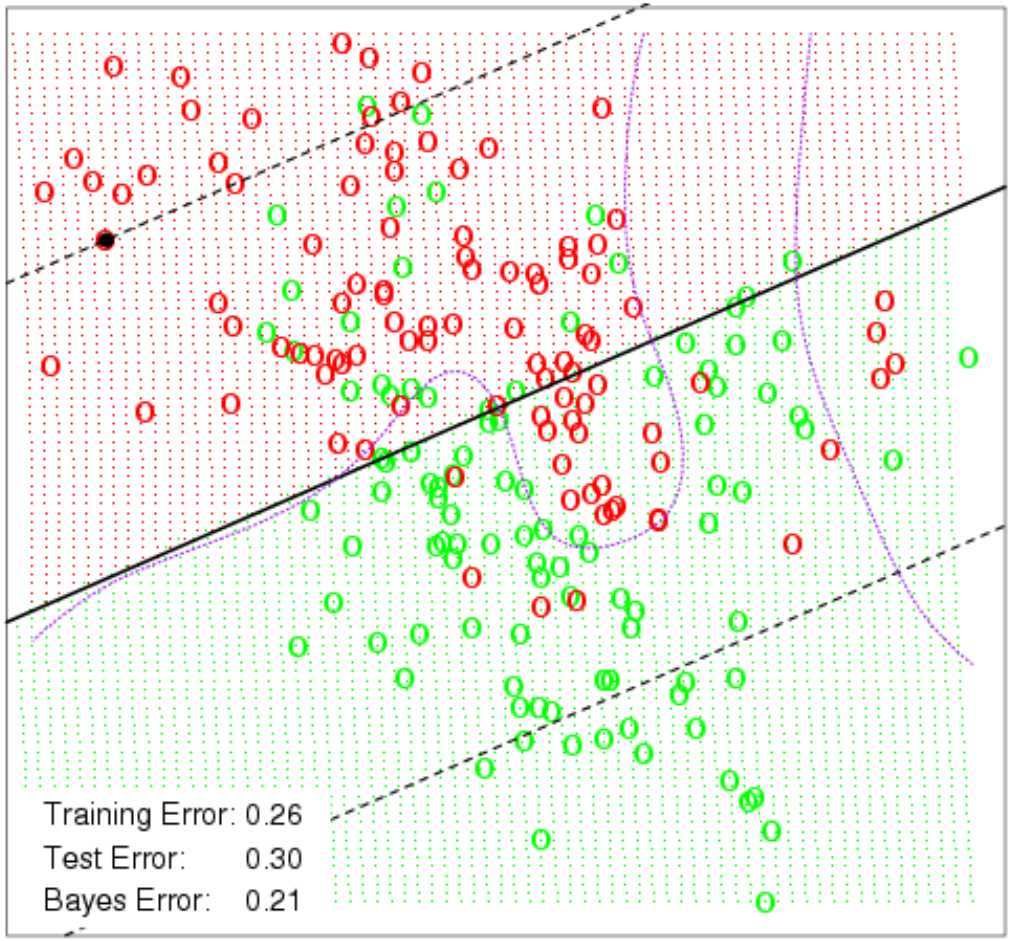
- The distance between the dashed lines represents the margin or 2C.
- The purple lines represent the Bayes decision boundaries



The same example with a larger constant

- Using a larger constant allows for a greater margin and creates a slightly different classifier.
- Notice, however, that the decision boundary must always be linear.





Non-linear support vector classifier

- The support vector classifier is fairly easy to think about. However, because it only allows for a linear decision boundary, it may not be all that powerful.
- Recall that linear regression is extended to non-linear regression using a basis function i.e.

$$Y_i = \beta_0 + \beta_1 b_1(X_i) + \beta_2 b_2(X_i) + \cdots + \beta_p b_p(X_i) + \varepsilon_i$$

A basis approach

- Conceptually, we can take a similar approach with the support vector classifier.
- The support vector classifier finds the optimal hyper-plane in the space spanned by $X_1, X_2, \ldots, X_p$.
- Instead, we can create transformations (or a basis) $b_1(x), b_2(x), \ldots, b_M(x)$ and find the optimal hyper-plane in the space spanned by $b_1(\mathbf{X}), b_2(\mathbf{X}), \ldots, b_M(\mathbf{X})$.
- This approach produces a linear plane in the transformed space but a non-linear decision boundary in the original space.
- This is called the support vector machine classifier.

Basis example

- Suppose we use polynomials as bases

$$X_1, X_2, X_1^2, X_2^2, X_1X_2, X_1^3, X_1X_2^2, \dots$$

- We go from p dimensional space to *M>p* dimensional space and fit the SVM classifier in the enlarged space
- For the bases $(X_1, X_2, X_1^2, X_2^2, X_1X_2)$ this gives a non-linear classifier in the original space

$$\beta_1X_1 + \beta_2X_2 + \beta_3X_1^2 + \beta_4X_2^2 + \beta_5X_1X_2 = 0$$

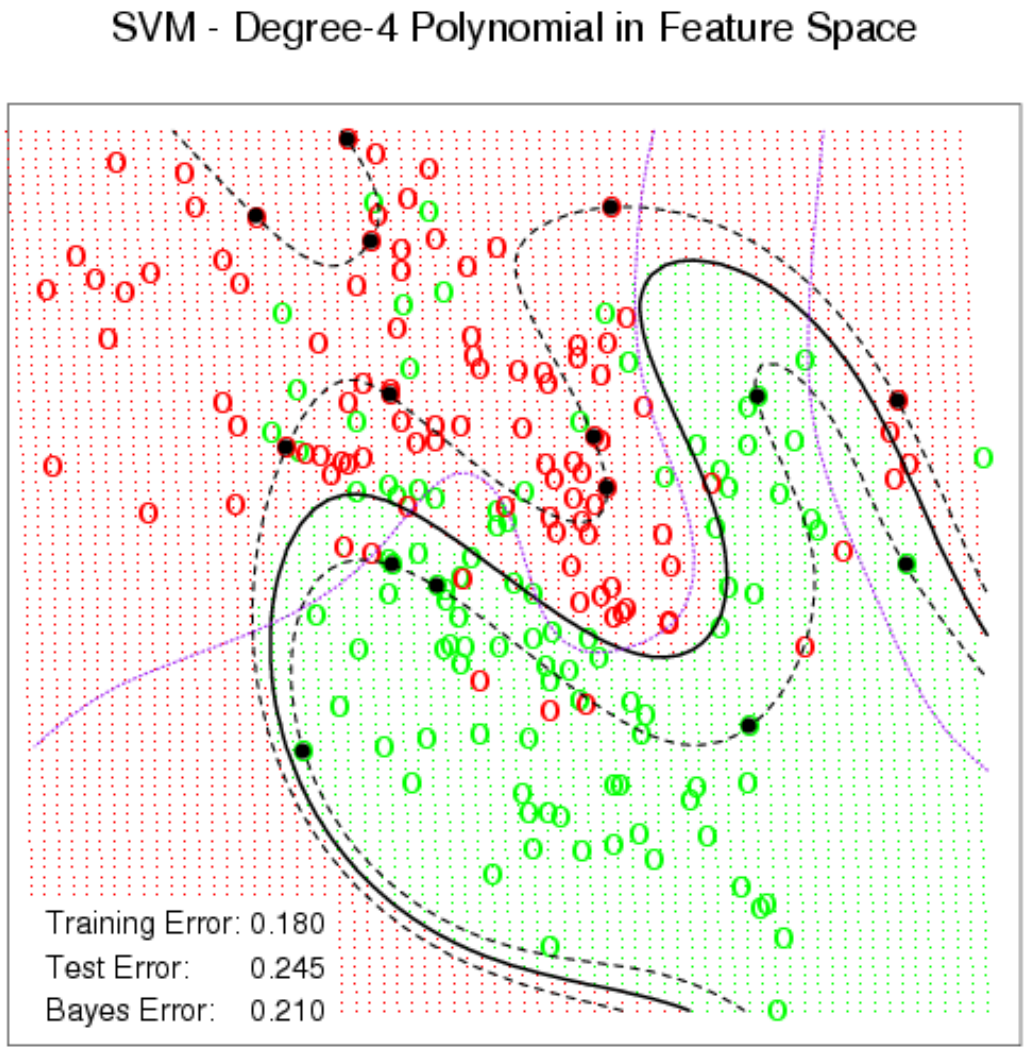
In reality

- While conceptually the basis approach is how the support vector machine works, there are some technical details which means that we don't actually choose $b_1(x), b_2(x), \ldots, b_M(x)$.
- Instead we choose a kernel function which takes the place of the basis.
- Kernel operates on inner products between instances
- Common kernel functions include
  - Linear
  - Polynomial
  - Radial Basis
  - Sigmoid

Polynomial kernel on Sim data

- Using a polynomial kernel, we now allow SVM to produce a non-linear decision boundary.
- Notice that the test error rate is a lot lower.

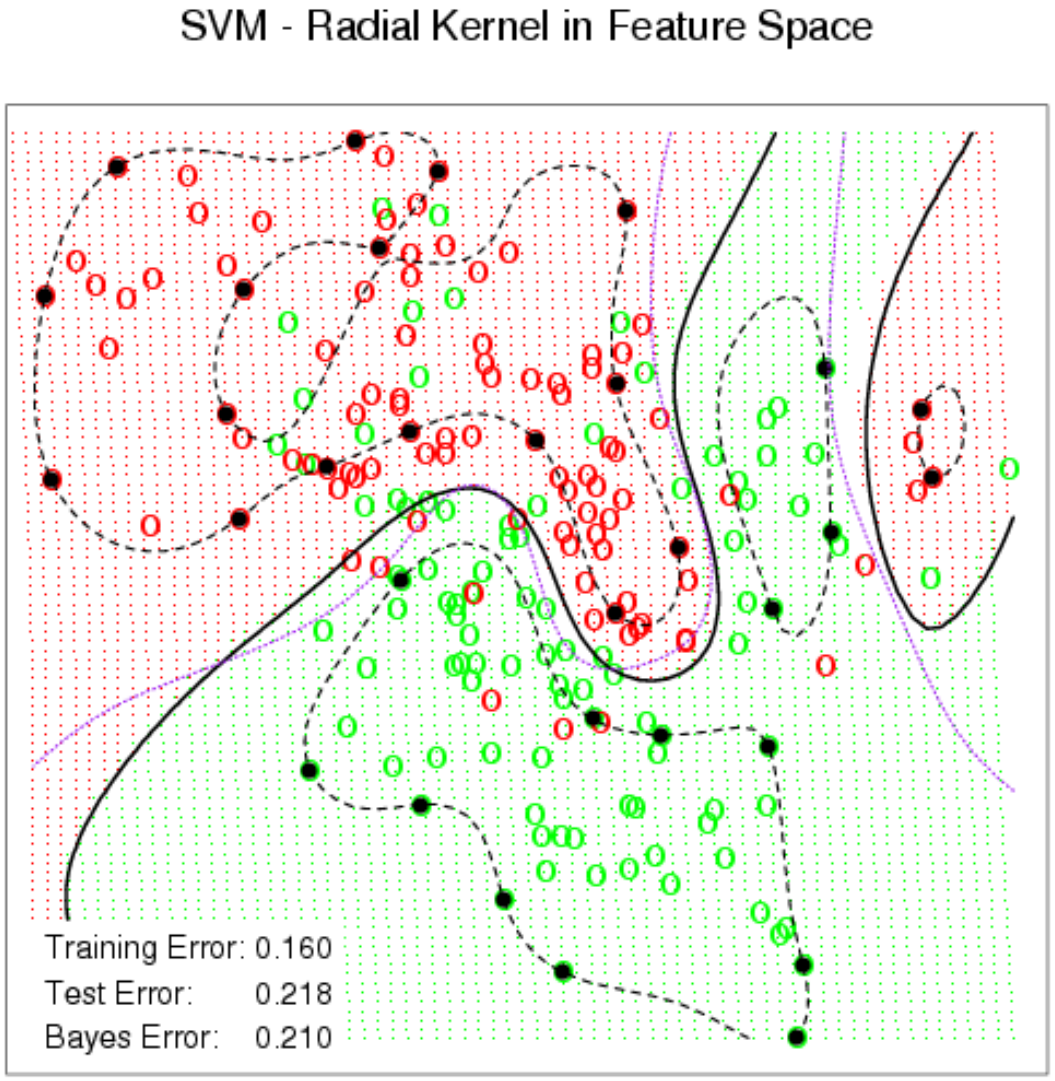
$$K(x_i, x_{i'}) = \left(1 + \sum_{j=1}^{p} x_{ij} x_{i'j}\right)^d$$



Radial basis kernel

- Using a radial basis kernel you often get an even lower error rate.

$$K(x_i, x_{i'}) = \exp(-\gamma \sum_{j=1}^{p} (x_{ij} - x_{i'j})^2).$$

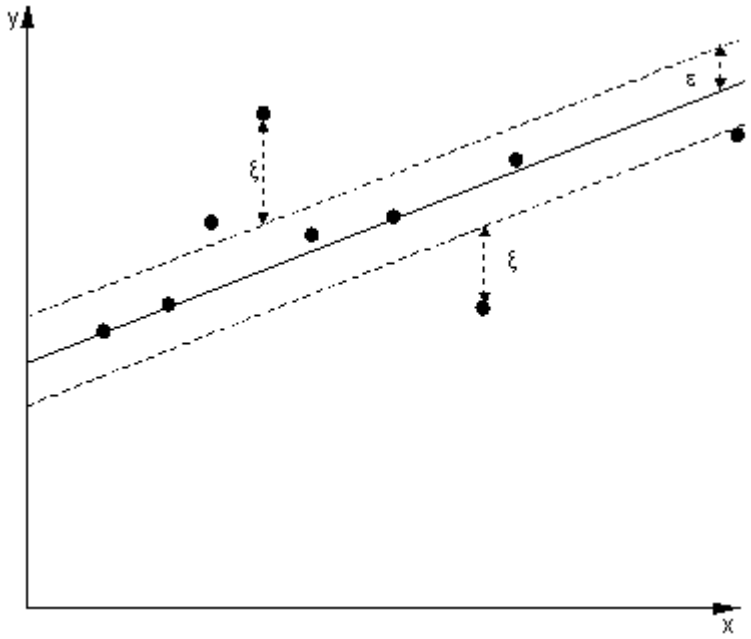


SVM for more than two classes

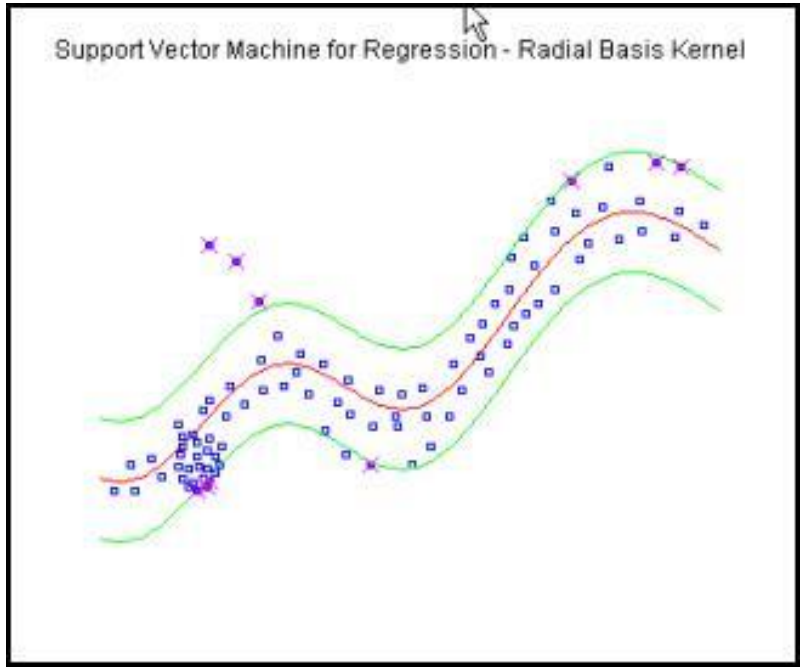
- The SVM as defined works for $K = 2$ classes. What do we do if we have $K > 2$ classes?
- One Versus All (OVA)
  Fit K different 2-class SVM classifiers $f_k(x)$, $k = 1,\ldots,K$; each class versus the rest. Classify new x to the class for which $f_k(x)$ is largest.
- One Versus One (OVO)
  Fit all $\binom{K}{2}$ pairwise classifiers $f_{uv}(x)$. Classify new x to the class that wins the most pairwise competitions.
- Which to choose?
  If K is not too large, use OVO.

SVM for regression

- As in classification, seek and optimize the generalization bounds given for regression.
- The loss function ignores errors which are situated within the certain distance of the true value; it is often called – epsilon intensive – loss function.



for linear SVM regression



University of Ljubljana, Faculty of Computer and Information Science

Inference and explanation of prediction models



Prof Dr Marko Robnik-Šikonja

Intelligent Systems, Edition 2024

Overview of topics



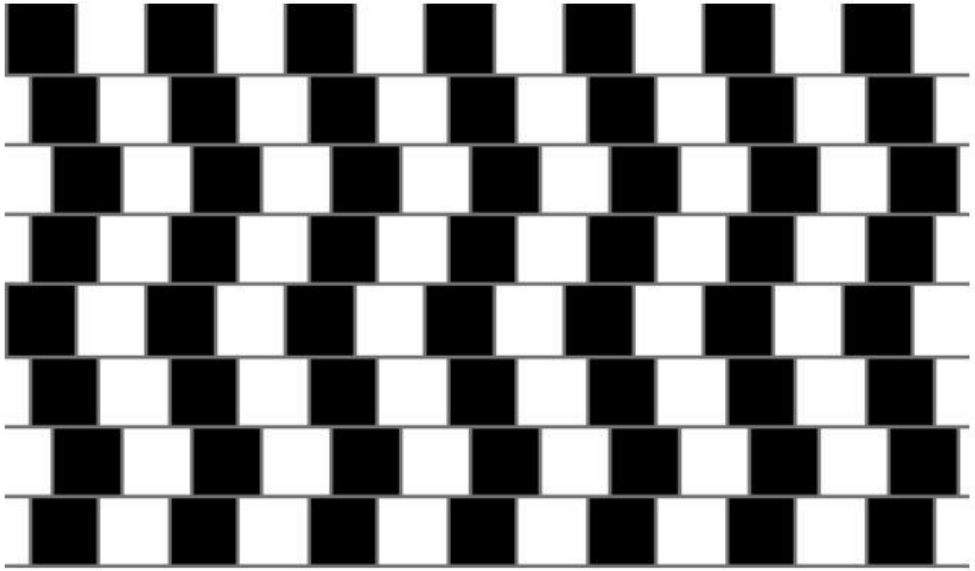
- Visualization and knowledge discovery.
- General methodology for explaining predictive models.
- Model level and instance level explanations, methods EXPLAIN and IME.

Visualization

- 1st rule of data mining: know your data.
- Therefore: visualizations, getting background data.
- Visualize: distributions of individual variables, their relations, etc.
- For high dimensional data sets one can use scaling, e.g. UMAP or t-SNE
- Clustering is useful in supervised tasks to get insight into the relation between predicted values Y and basic groups in the data. If unrelated, feature set might need amendments.

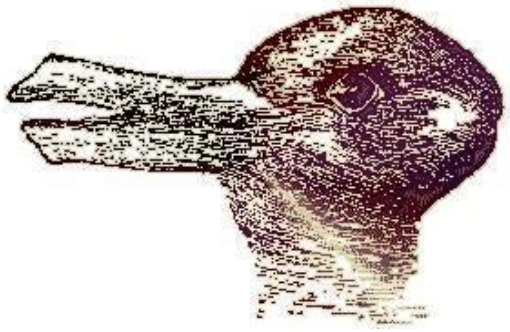
Visualizations

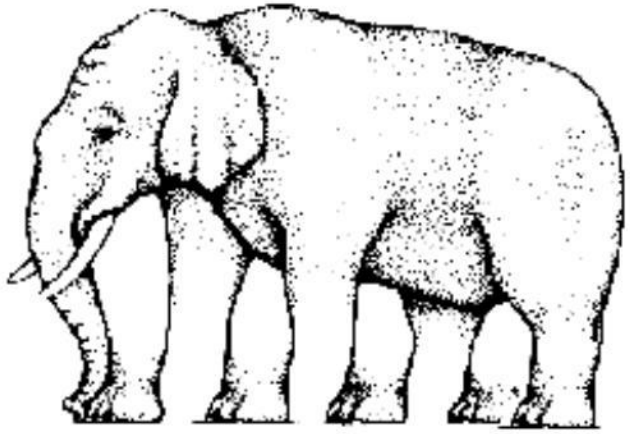
- Human visual perception has certain limitations:
  - we see what we want to see
  - we see what we see often
  - it is more difficult to notice unexpected patterns
- practice in detection of unknown
- use visualizations which expose “the unknown”



Are the horizontal lines parallel or do they slope?







How many legs does this elephant have?

Human pattern recognition

- We see inexistent patterns because we WANT to see them (we feel lost without them).



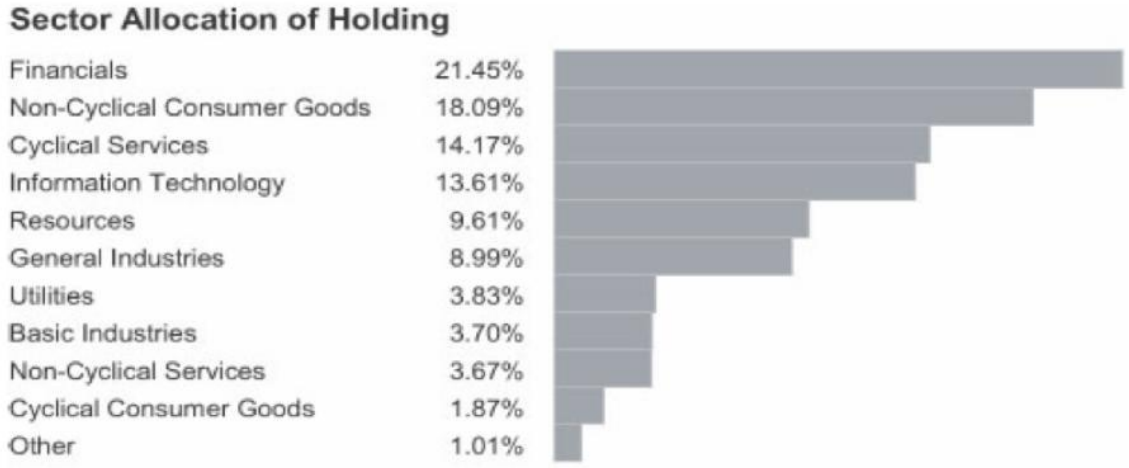


"The researchers found that when people were primed to feel out of control, they were more likely to see patterns where none exist." (**See a Pattern on Wall Street?,** *John Tierney*, *Science*)

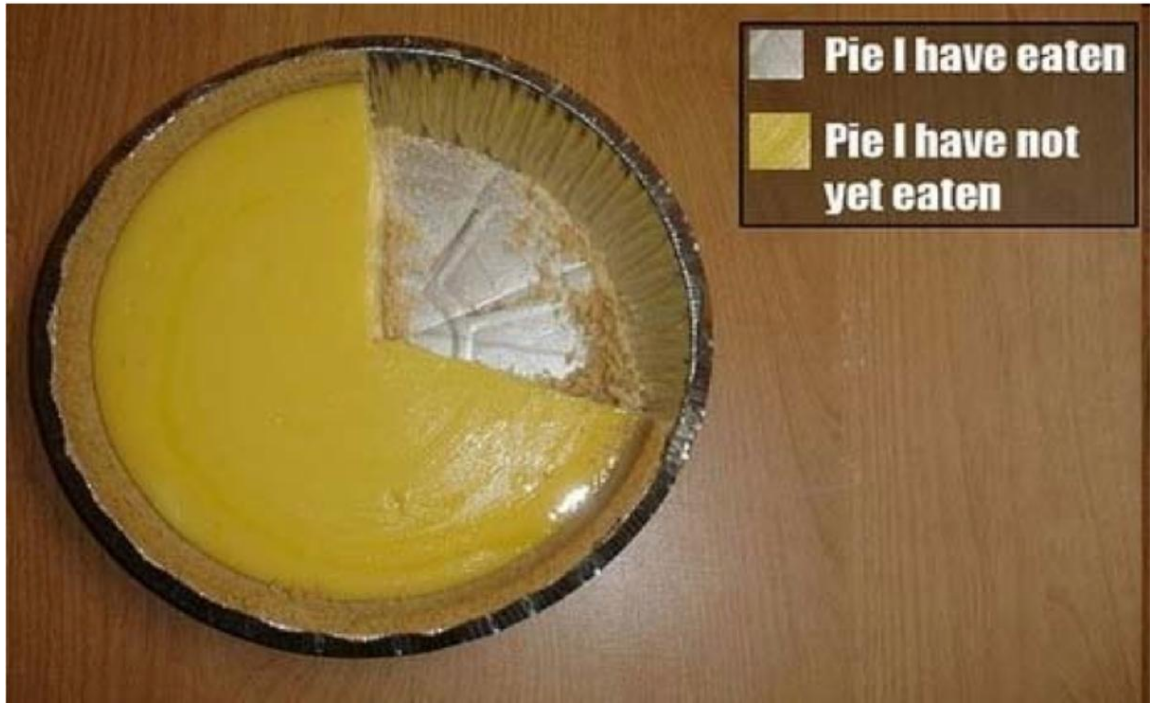
Facts about simple visualizations

- Pie charts are a bad choice: hard to read, similar colors, slope, legend is too far away
- Bar chart is much better



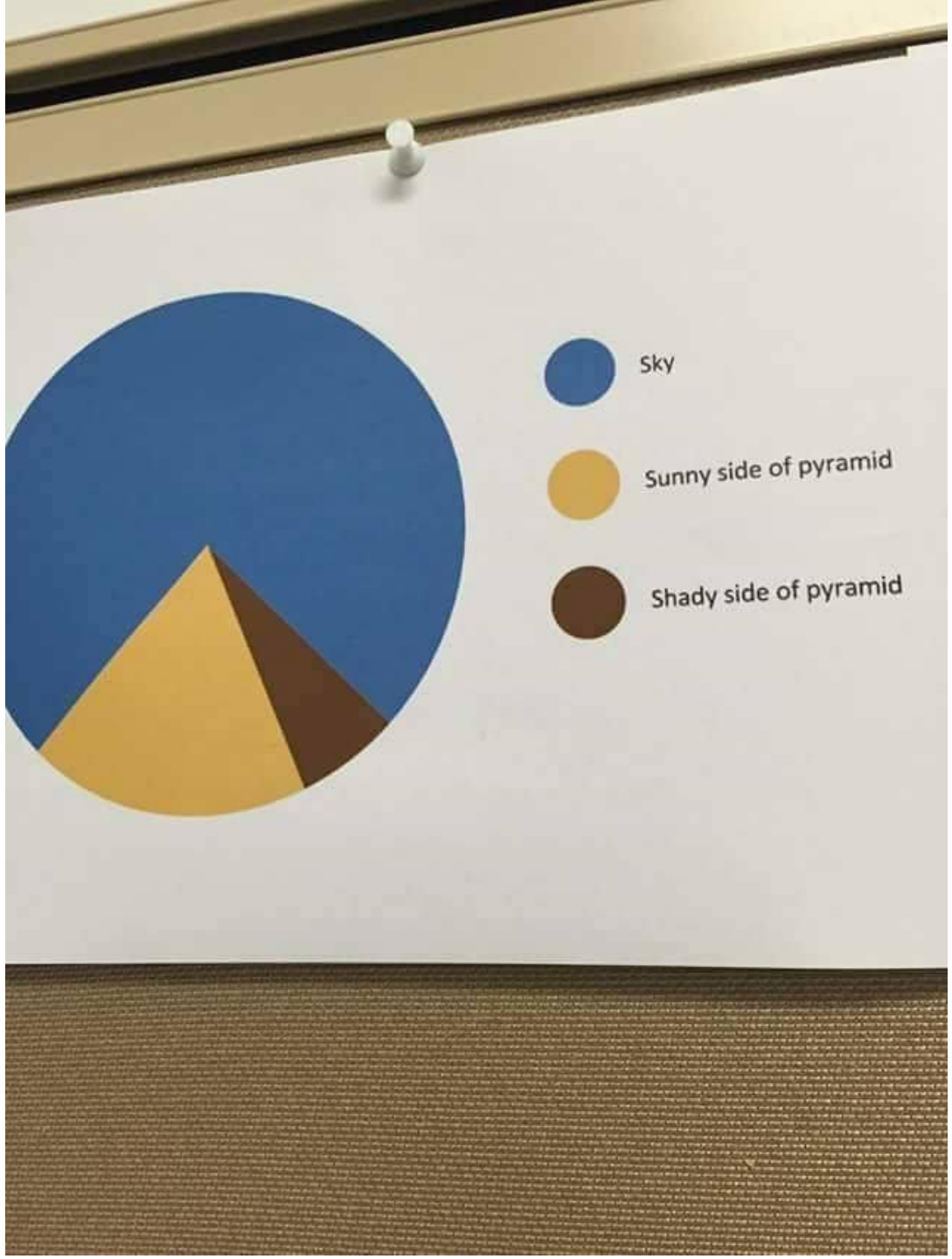


The best pie chart



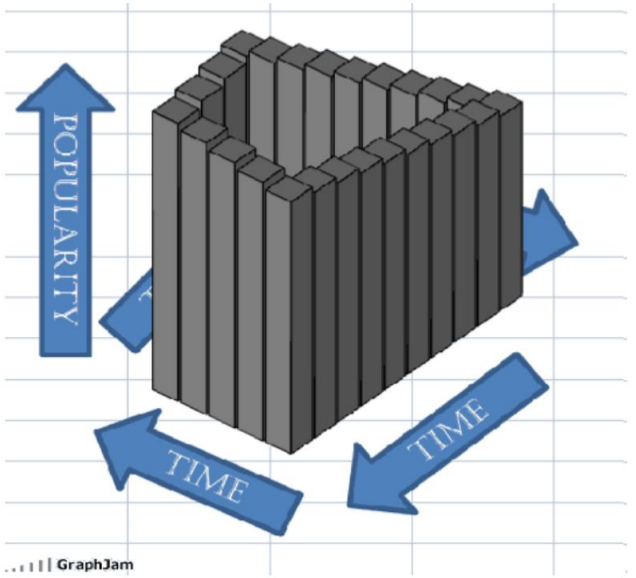
Pie charts jokes

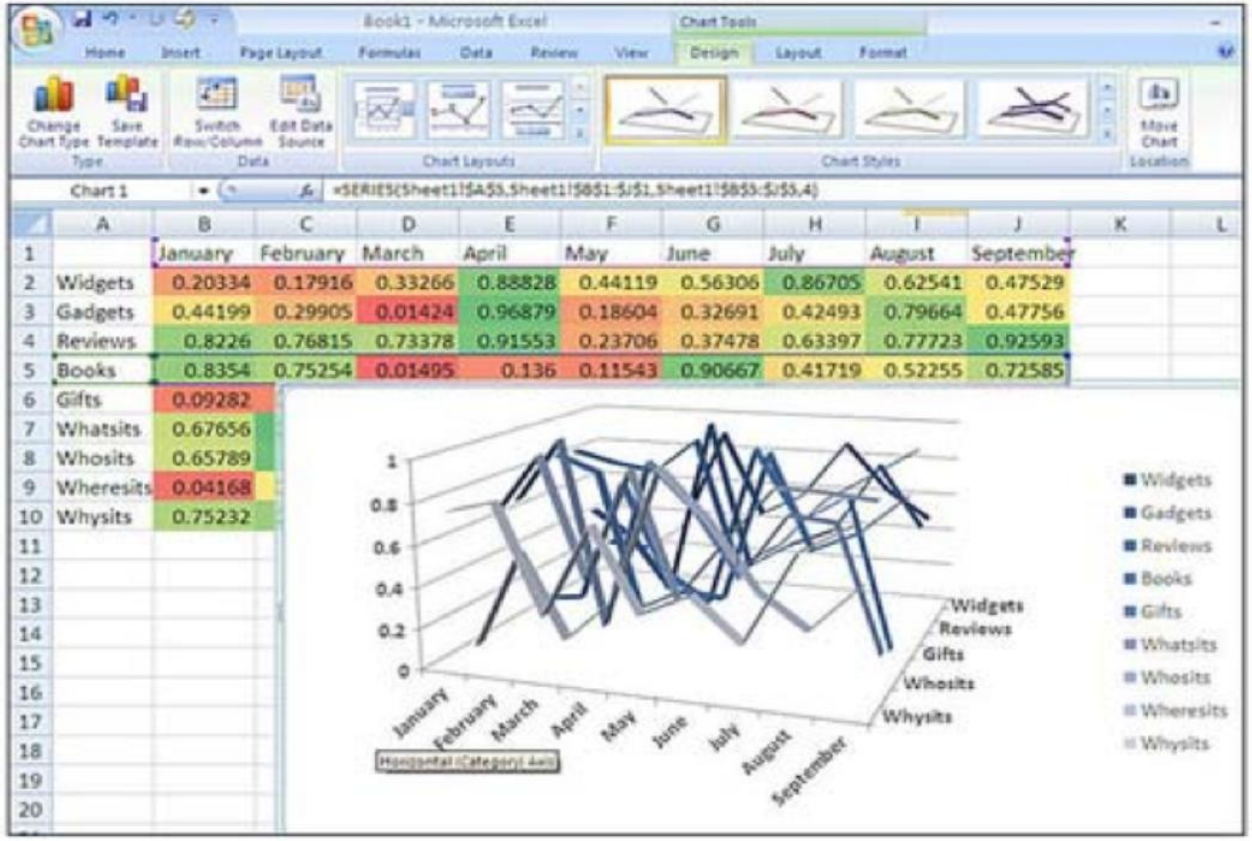
- notoriously bad

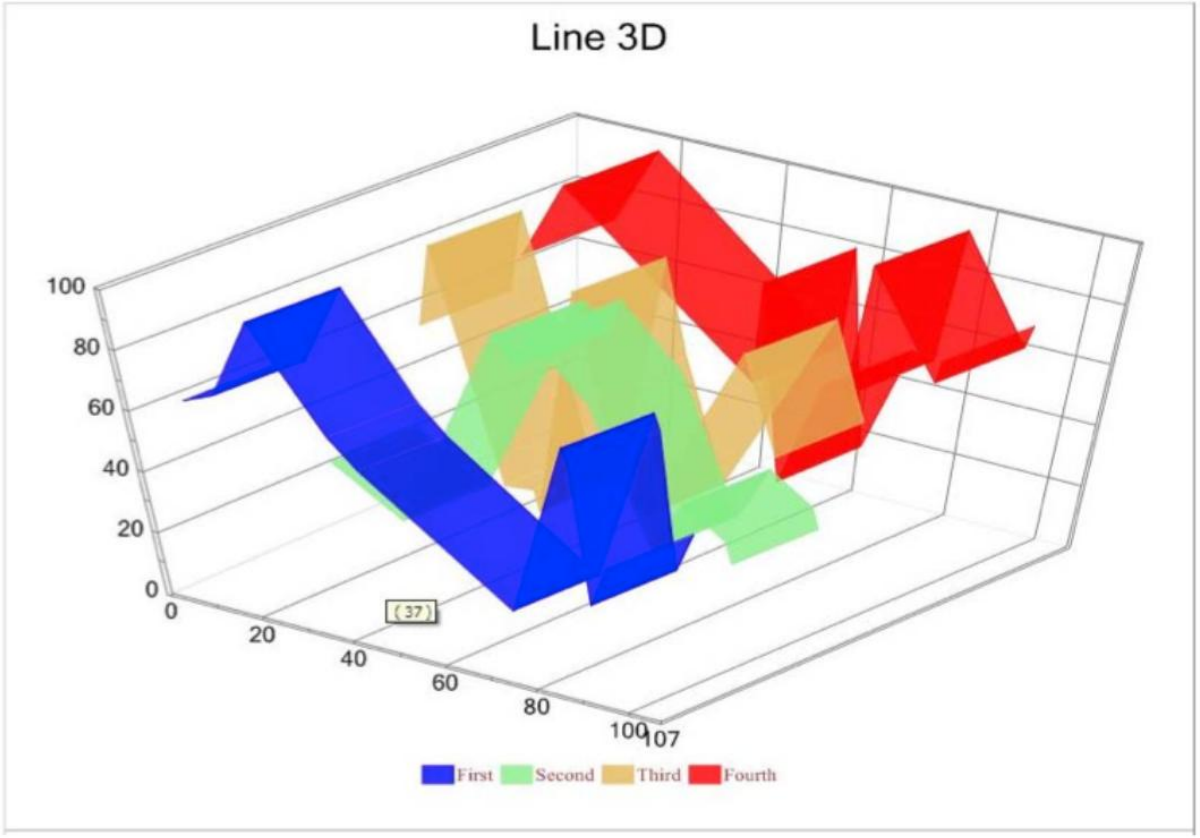


Facts about simple visualizations

- Bar charts, box plots can be OK
- 3D graphs are almost never OK for 2D info: spider plot, bowl of noodles
- Take care to be clear and do not manipulate
- A more detailed examples and recommendations https://github.com/cxli233/FriendsDontLetFriends

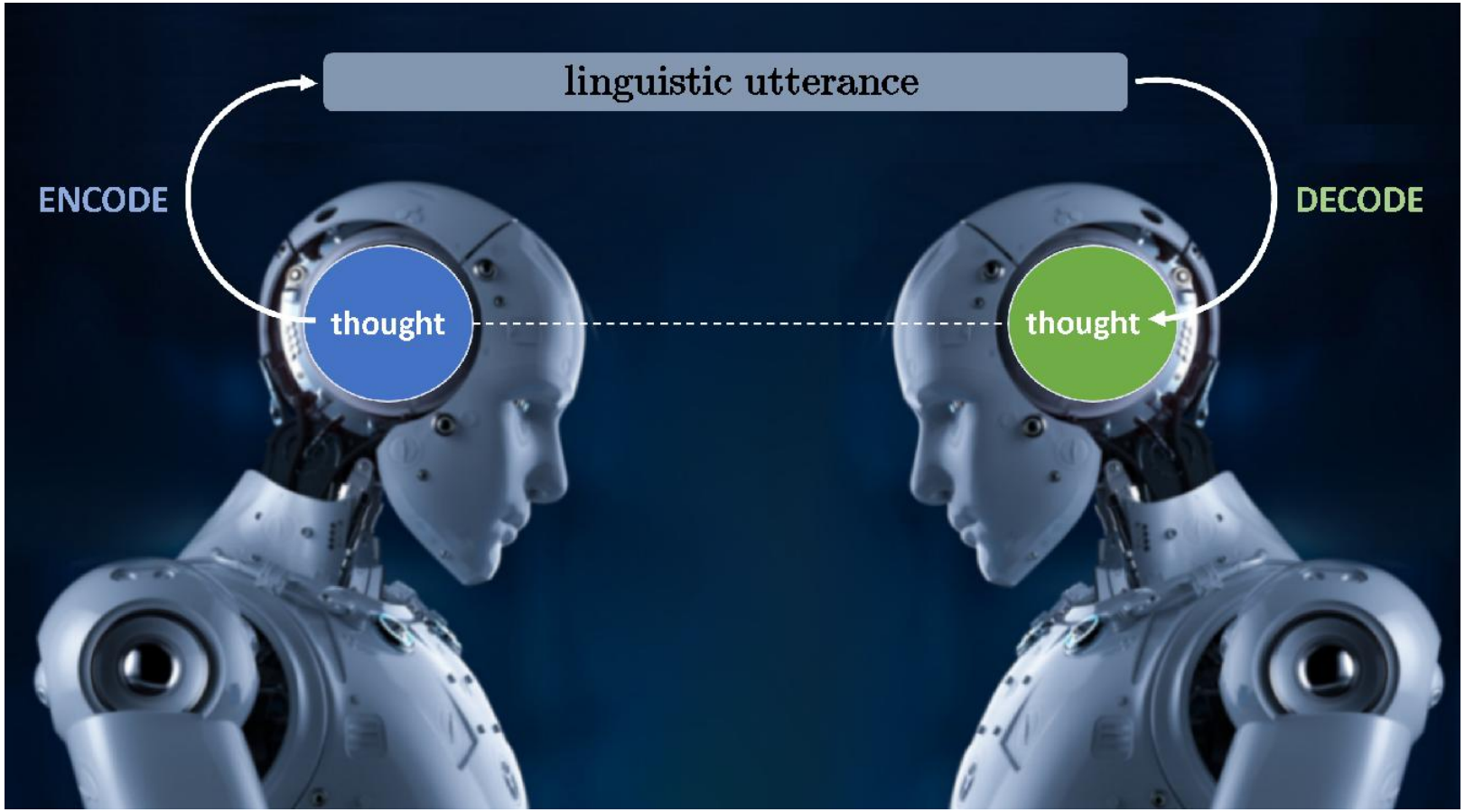






Understanding

Walid Saba, "Machine Learning Won't Solve Natural Language Understanding", The Gradient, 2021.

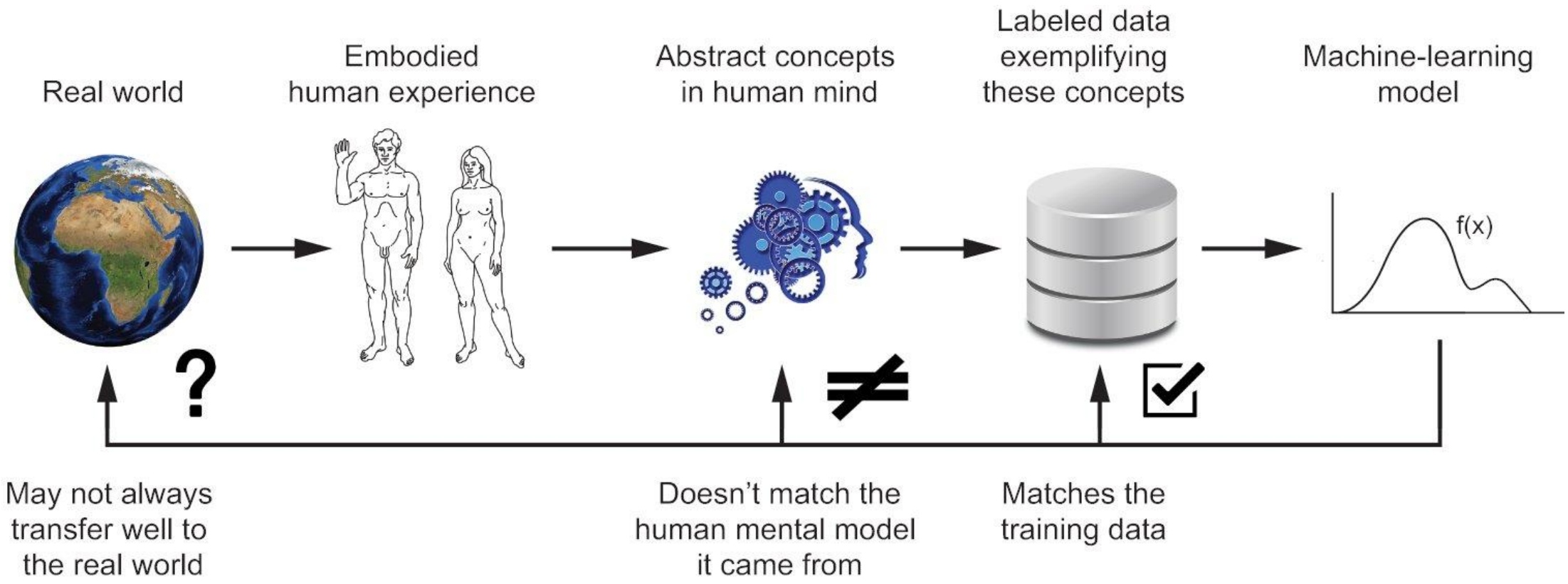


Xanadu, who is a living young human adult, and who was in graduate school, quit graduate school to join a software company that had a need for a new employee.

≈

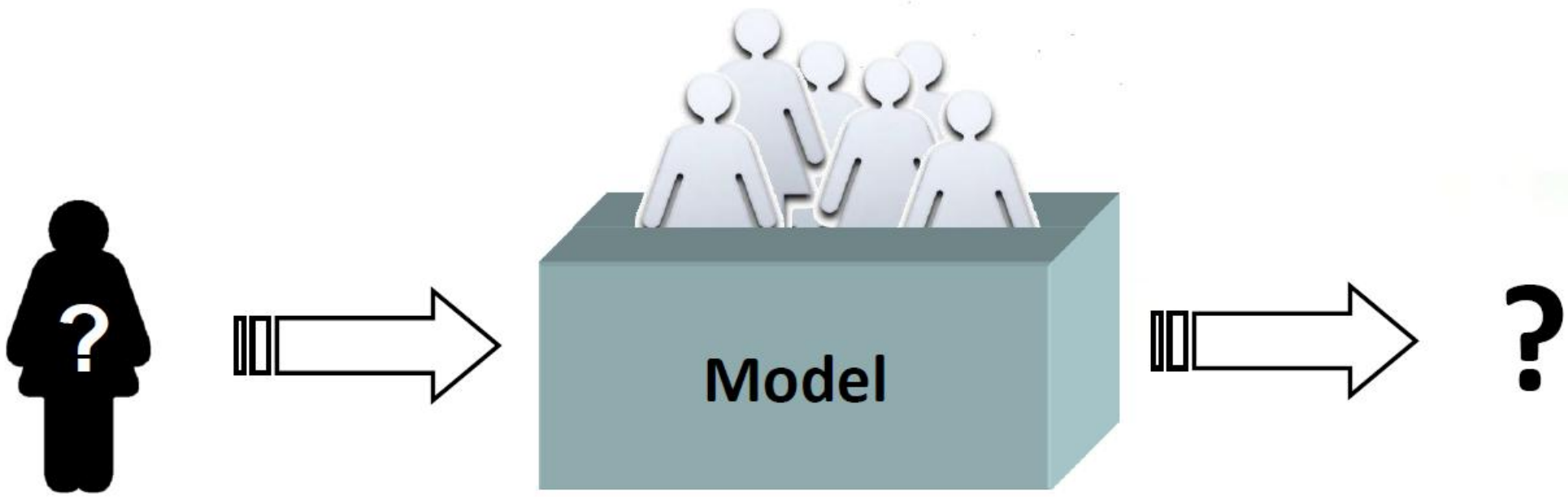
Xanadu quit graduate school to join a software company.

Understanding ML models is difficult



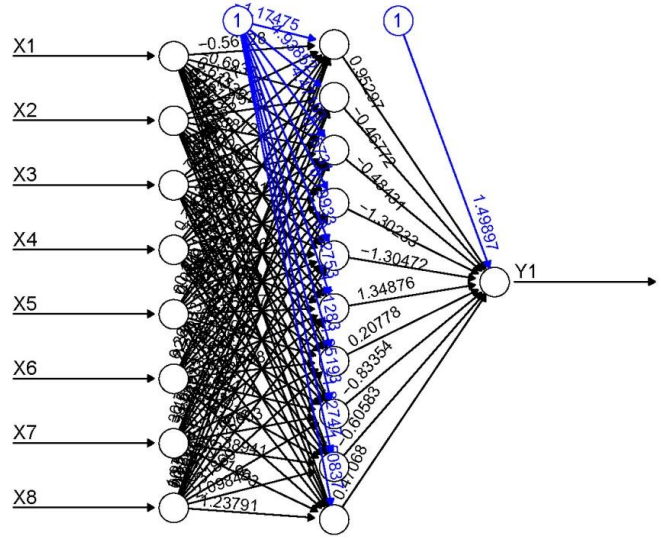
Predictive modeling scenario

We want to learn from past examples,with known outcomes.



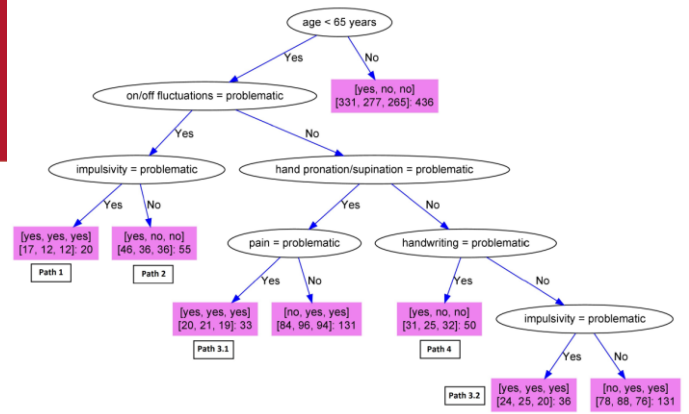
To predict the outcome for a new patient.

Explanation of predictions



- a number of successful prediction algorithms exist (SVM, boosting, random forests, neural networks), but to a user they are [redacted]
- many fields where users are very much concerned with the transparency of the models: medicine, law, consultancy, public services, etc.
- Some explanation methods are applicable to arbitrary predictors

Model comprehensibility



- decision support: model comprehensibility is important to gain users‘ trust
- knowledge acquisition
- some models are inherently interpretable and comprehensible
- decision and regression trees, classification and regression rules, linear and logistic regression $1/(1+\exp(-(b_0+b_1x_1+...b_px_p))$
- really?

Domain level explanation



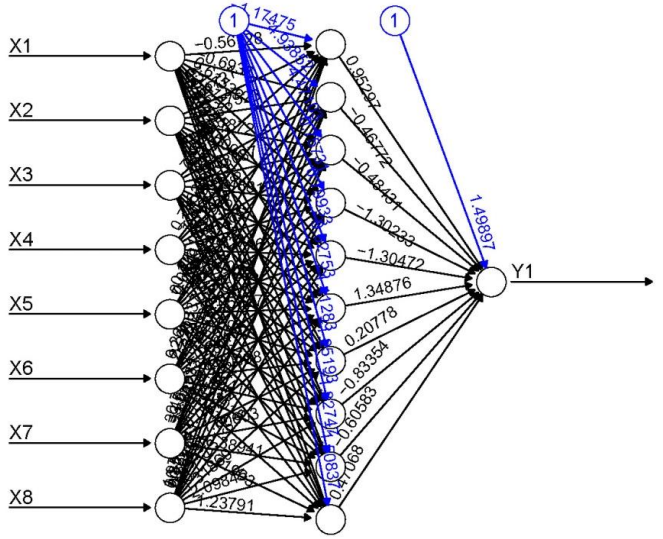
- trying to explain the "true causes and effects"
  - physical processes
  - stock exchange events

- usually unreachable except for artificial problems with known relations and generator function
- some aspects are covered with attribute evaluation, detection of redundancies, ...
- targeted indirectly through the models

Model-based explanations

All models are wrong,
but some are useful.

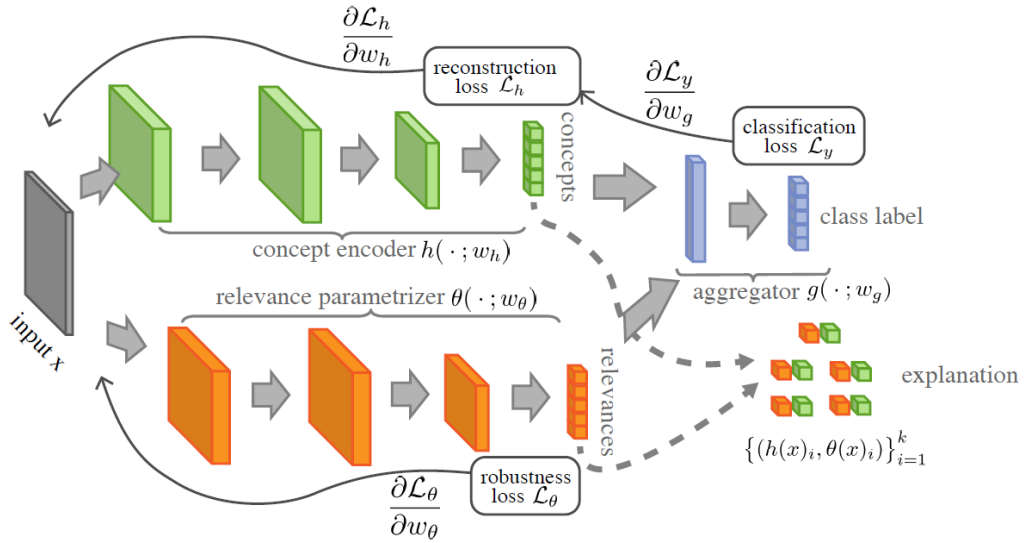
George Box, British statistician (1919 – 2013)



- make transparent the prediction process of a particular model
- the correctness of the explanation is independent of the correctness of the prediction but
- better models (with higher prediction accuracy) enable in principle better explanation at the domain level
- explanation methods are interested only in the explanation at the model level and leave to the developer of the model the responsibility for its prediction accuracy

Two flavours of explanation techniques

- model specific
  - especially used for deep neural networks

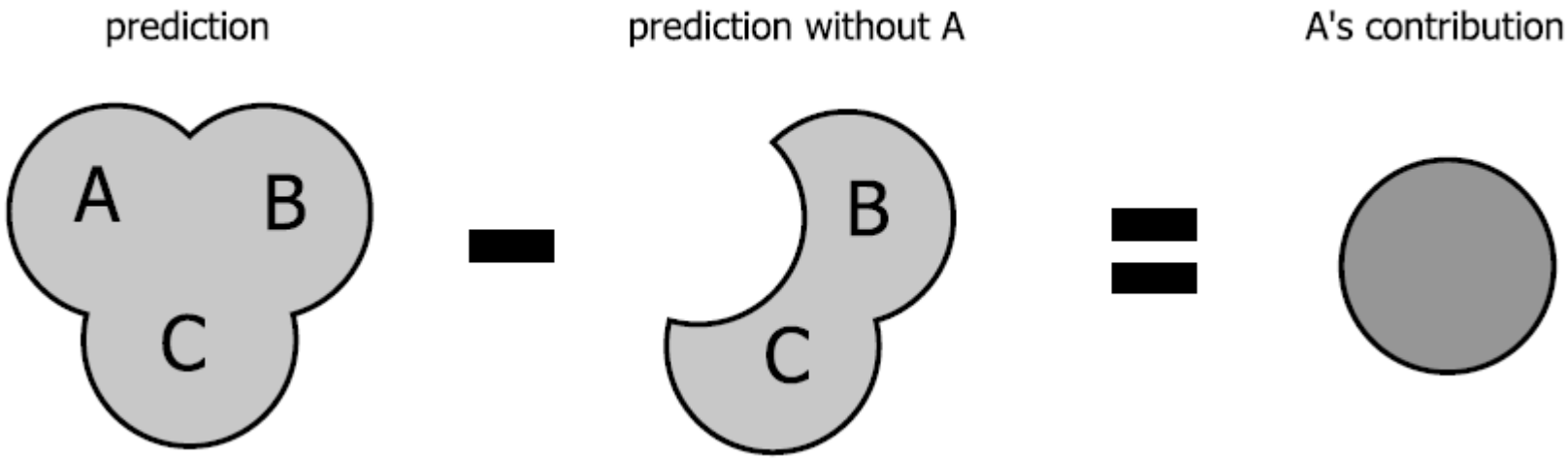


Melis, D.A. and Jaakkola, T., 2018. Towards robust interpretability with self-explaining neural networks. In *Advances in Neural Information Processing Systems* (pp. 7786-7795).

- model agnostic
  - can be used for any predictor,
  - based on perturbation of the inputs

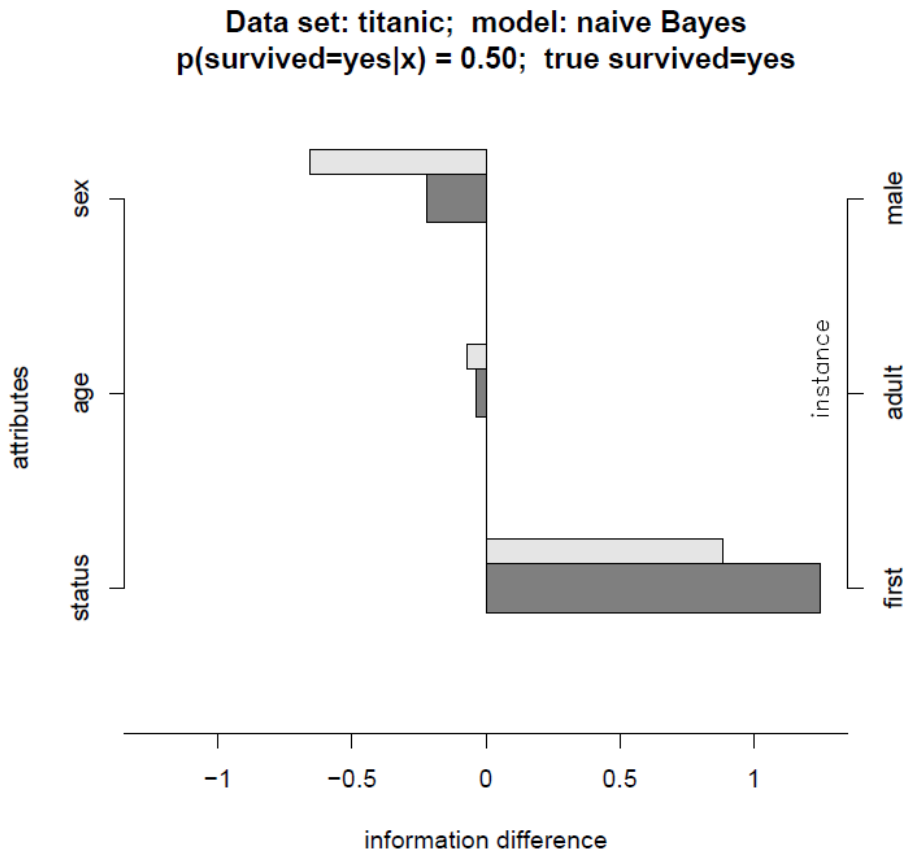
Idea of perturbtion-based explanations

- importance of a feature or a group of features in a specific model can be estimated by simulating lack of knowledge about the values of the feature(s)



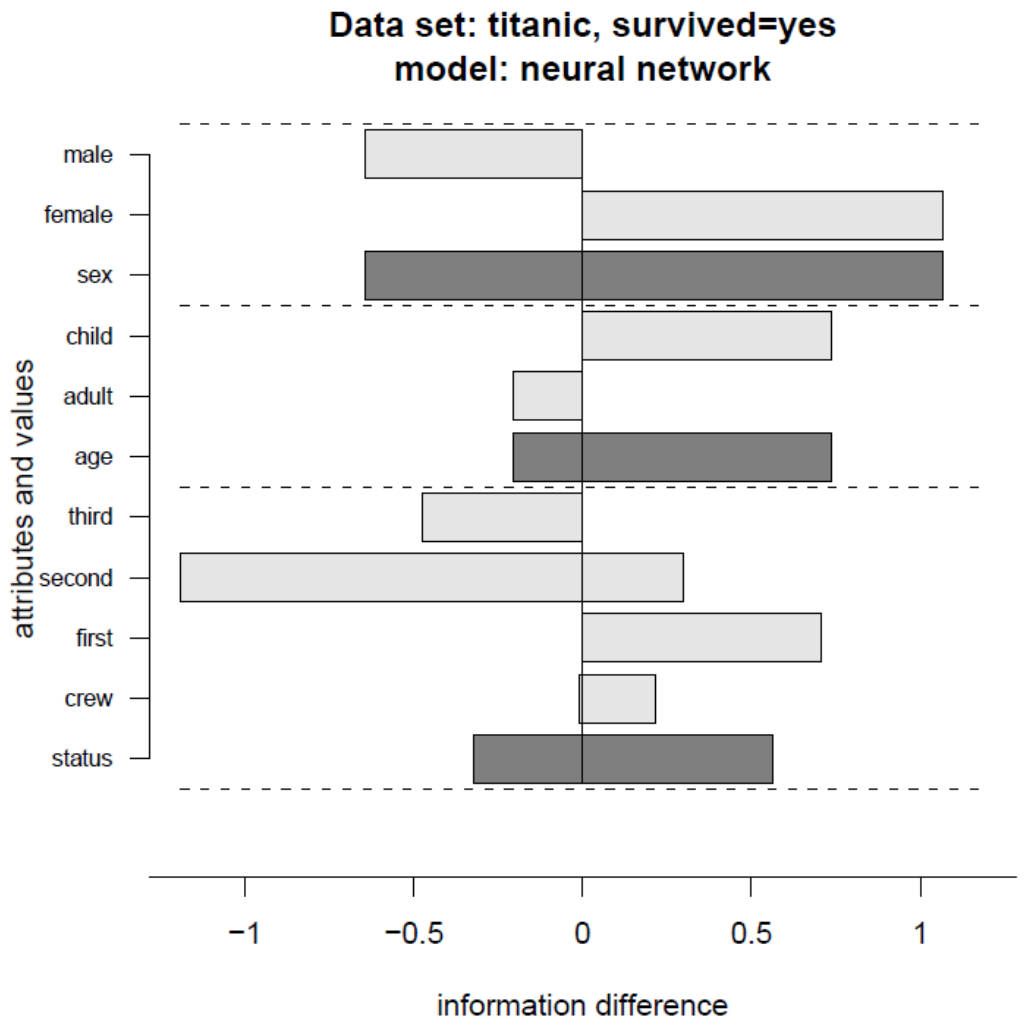
Instance-level explanation

- explain predictions for each instance separately
  - this is what practitioners applying models are interested in
  - presentation format: impact of each feature on the prediction value
- model-based



Model-level explanation

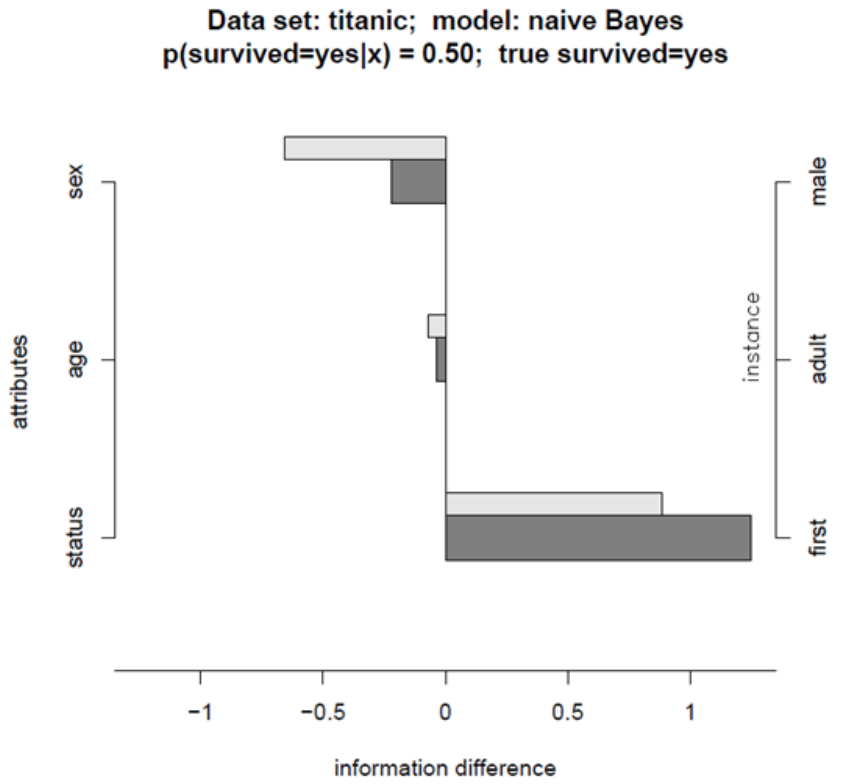
- the overall picture of a problem the model conveys
  - this is what knowledge extractors are interested in
  - presentation format: overall importance of each feature,
    but also rules, trees
- model-based



The method EXPLAIN

- "hide" one attribute at a time
- estimate contribution of attribute from

$$p(y_k|x) - p_{S\setminus\{i\}}(y_k|x)$$



Robnik-Sikonja, M., & Kononenko, I. (2008). Explaining classifications for individual instances. *IEEE Transactions on Knowledge and Data Engineering,*, *20*(5), 589-600.

Explaining EXPLAIN

- assume an instance (**x**, y); components of **x** are values of attributes A_i
- for a new instance **x**, we want to know what role each attribute's value play in the prediction model f, i.e. to what extend it contributed to the classification f(**x**)
- for that purpose
  - we compute f(**x** \ A_i), the model's prediction for **x** without the knowledge of the event $A_i = a_k$ (marginal prediction)
  - we comparing f(**x**) and f(**x** \ A_i) to assess importance of $A_i = a_k$
  - the larger the the difference the more important the role of $A_i=a_k$ in the model
- f(**x**) and f(**x** \ A_i) are source of explanations

Evaluation of prediction differences

- how to evaluate $f(\mathbf{x}) - f(\mathbf{x} \setminus A_i)$
- in classification, we take $f(\mathbf{x})$ in the form of probability

1. **difference of probabilities**

   $probDiff_i(y|\mathbf{x}) = p(y|\mathbf{x}) - p(y|\mathbf{x} \setminus A_i)$

2. **information gain** (Shannon, 1948)

   $infGain_i(y|\mathbf{x}) = \log_2 p(y|\mathbf{x}) - \log_2 p(y|\mathbf{x} \setminus A_i)$

3. **weight of evidence** also log odds ratio (Good, 1950)

   $odds(z) = p(z) / (1 - p(z))$

   $WE_i(y|\mathbf{x}) = \log_2 odds(y|\mathbf{x}) - \log_2 odds(y|\mathbf{x} \setminus A_i)$

Implementation

- $p(y|\mathbf{x})$: classify $\mathbf{x}$ with the model
- $p(y|\mathbf{x}\setminus A_i)$ – simmulate lack of knowledge of A_i in the model
  - replace with special NA value: good for some, mostly bad, left to the mercy of model's internal mechanism
  - average prediction across perturbations of A_i

    $p(y|\mathbf{x}\setminus A_i) = \sum_a p(A_i=a_s)\, p(y|\mathbf{x} \leftarrow A_i = a_s)$

    - use discretization for numeric attributes
    - use Laplace correction for probability estimation
  - we could build a separate model for each $p(y|\mathbf{x}\setminus A_i)$

Weaknes of EXPLAIN

- "hide" one attribute at a time
- estimate contribution of attribute from

$$p(y_k|x) - p_{S\setminus\{i\}}(y_k|x)$$

- weakness: if there are redundant ways to express concept, credit is not assigned
- example:

  $C = A_1 \vee A_2A_3$

  explanation for instance ($A_1=A_2=A_3=1$)

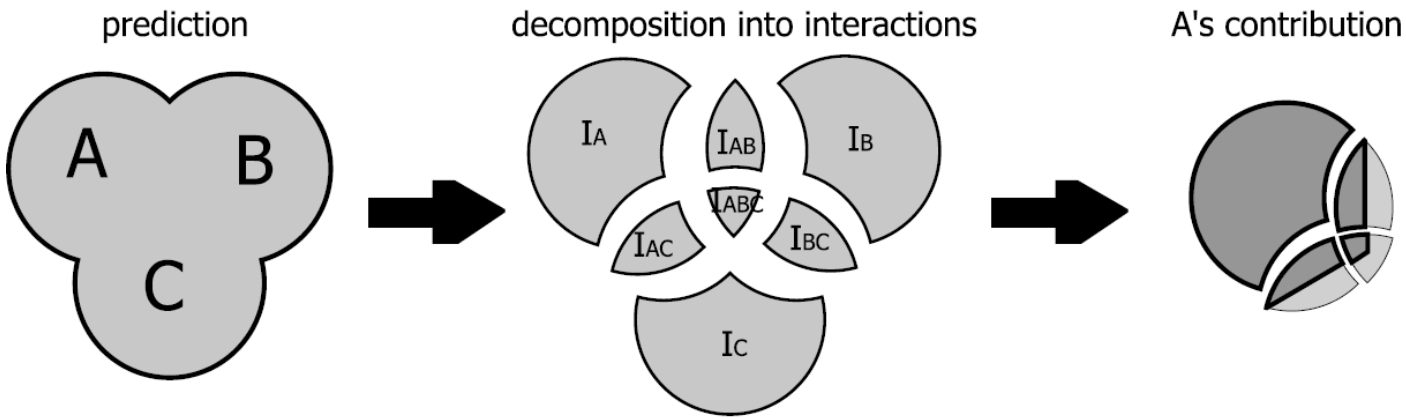
University of Ljubljana
Faculty of Computer and Information Science

The method IME

- (Interactions-based Method for Explanation)
- “hide” any subset of attributes at a time (2^a subsets!)
- the source of explanations is the difference in prediction using a subset of features Q and an empty set of features {}

$$\Delta_Q = h(x_Q) - h(x_{\{\}})$$

- the feature gets some credit for standalone contributions and for contributions in interactions



IME: sum over all subsets

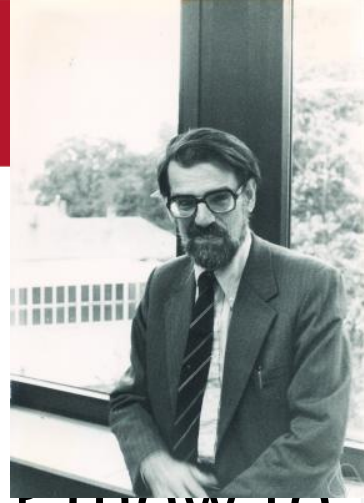
- the contributions are

$$\pi_i = \sum_{Q \subseteq \{1,2,\ldots,a\} - \{i\}} \frac{1}{a \binom{a-1}{a-|Q|-1}} (\Delta_{Q \cup \{i\}} - \Delta_Q)$$

Štrumbelj, E., Kononenko, I. & Robnik-Šikonja, M., Explaining instance classifications with interactions of subsets of feature values. *Data & Knowledge Engineering*, Oct. 2009, 68(10):886-904

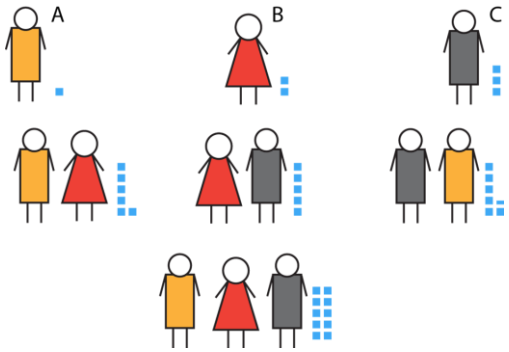
University of Ljubljana
Faculty of Computer and Information Science

Game theory analogy



- coalitional game of *a* players (attributes)
- players form coalitions (i.e. interactions)
- how to distribute the payout to the members of a coalition? (how to assign the credit for prediction)
- The Shapley value is the unique payoff vector that is
  - efficient (exactly splits payoff value),
  - symmetric (equal payments to equivalent players)
  - additive (overall credit is a sum of participating in coalitions), and
  - assigns zero payoffs to dummy players (no contribution to any coalition).

Shapley value



$$Sh_i(v) = \sum_{S \subseteq N \setminus \{i\}, s=|S|} \frac{(n-s-1)!s!}{n!}(v(S \cup \{i\}) - v(S)), \quad i = 1, \ldots, n.$$

$$\pi_i = \sum_{Q \subseteq \{1,2,\ldots,a\} - \{i\}} \frac{1}{a\binom{a-1}{a-|Q|-1}}(\Delta_{Q \cup \{i\}} - \Delta_Q)$$

- Shapley value can be efficiently approximated

Solution for IME: sampling

- Shapley value can expressed in an alternative formulation
- $\pi(a)$is the set of all ordered permutations of *a*
- Pre^i(O) is the set of players which are predecessors of player *i* in the order O ∈ $\pi(a)$

$$\varphi_i(k,x) = \frac{1}{a!} \sum_{\mathcal{O}\in\pi(a)} \left(\Delta(Pre^i(\mathcal{O}) \cup \{i\})(k,x) - \Delta(Pre^i(\mathcal{O}))(k,x)\right) =$$

$$= \frac{1}{a!} \sum_{\mathcal{O}\in\pi(a)} \left(p_{Pre^i(\mathcal{O})\cup\{i\}}(y_k|x) - p_{Pre^i(\mathcal{O})}(y_k|x)\right),$$

- smart sampling over subsets of attributes
- computationally feasible approach

Štrumbelj, E., & Kononenko, I. (2010). An efficient explanation of individual classifications using game theory. Journal of Machine Learning Research, 11, 1-18

IME algorithm

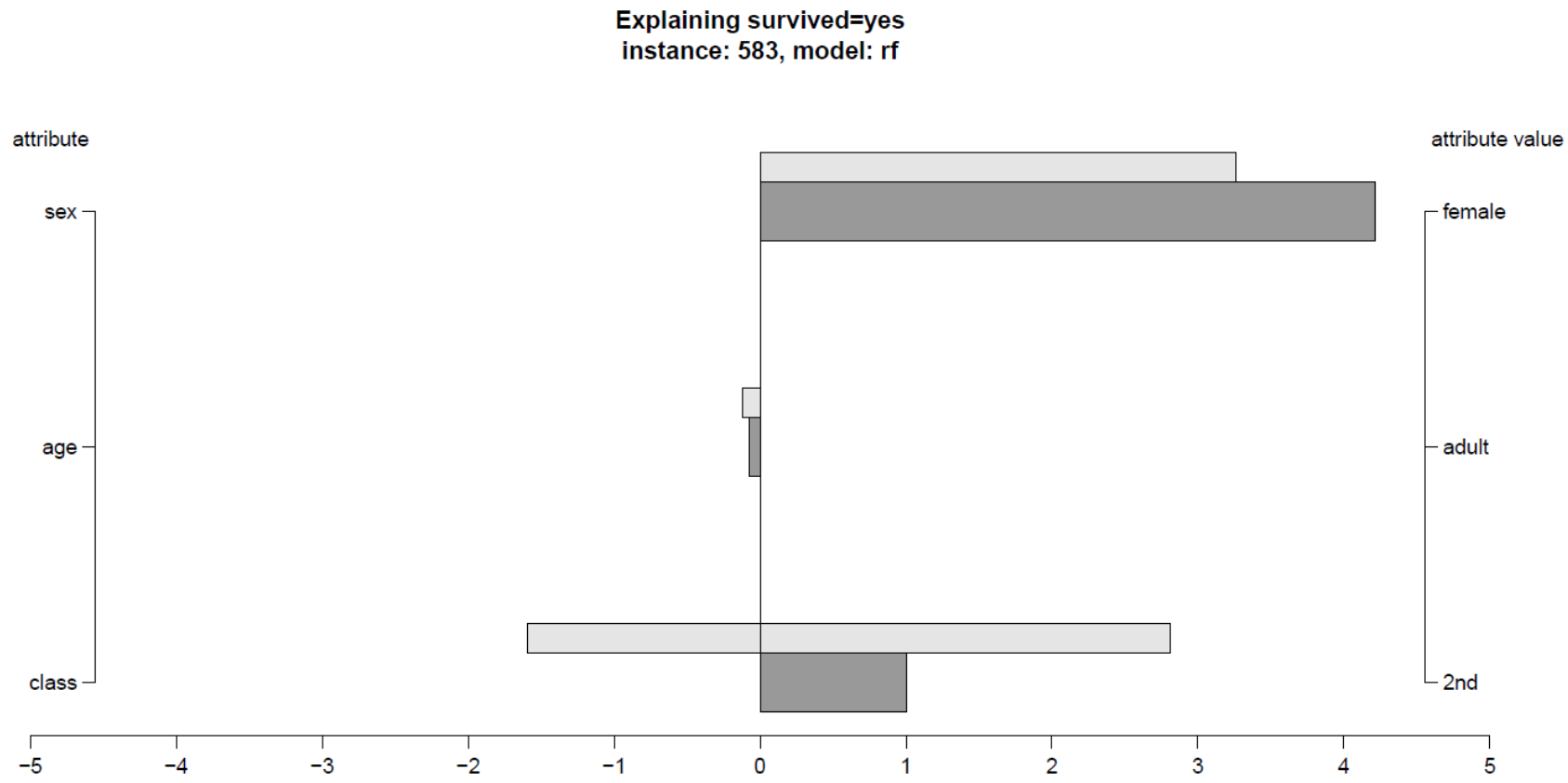
Algorithm 1 Approximating the contribution of the i-th feature's value, φ_i, for instance $x \in \mathcal{A}$.

determine m, the desired number of samples
$\varphi_i \leftarrow 0$
for $j = 1$ to m **do**
  choose a random permutation of features $O \in \pi(N)$
  choose a random instance $y \in \mathcal{A}$
  $v_1 \leftarrow f(\tau(x, y, Pre^i(O) \cup \{i\}))$
  $v_2 \leftarrow f(\tau(x, y, Pre^i(O)))$
  $\varphi_i \leftarrow \varphi_i + (v_1 - v_2)$
end for
$\varphi_i \leftarrow \frac{\varphi_i}{m}$

- by measuring the variance of contributions, we can determine the necessary number of samples for each attribute

Visualization of explanations

- instance-level explanation on Titanic data set

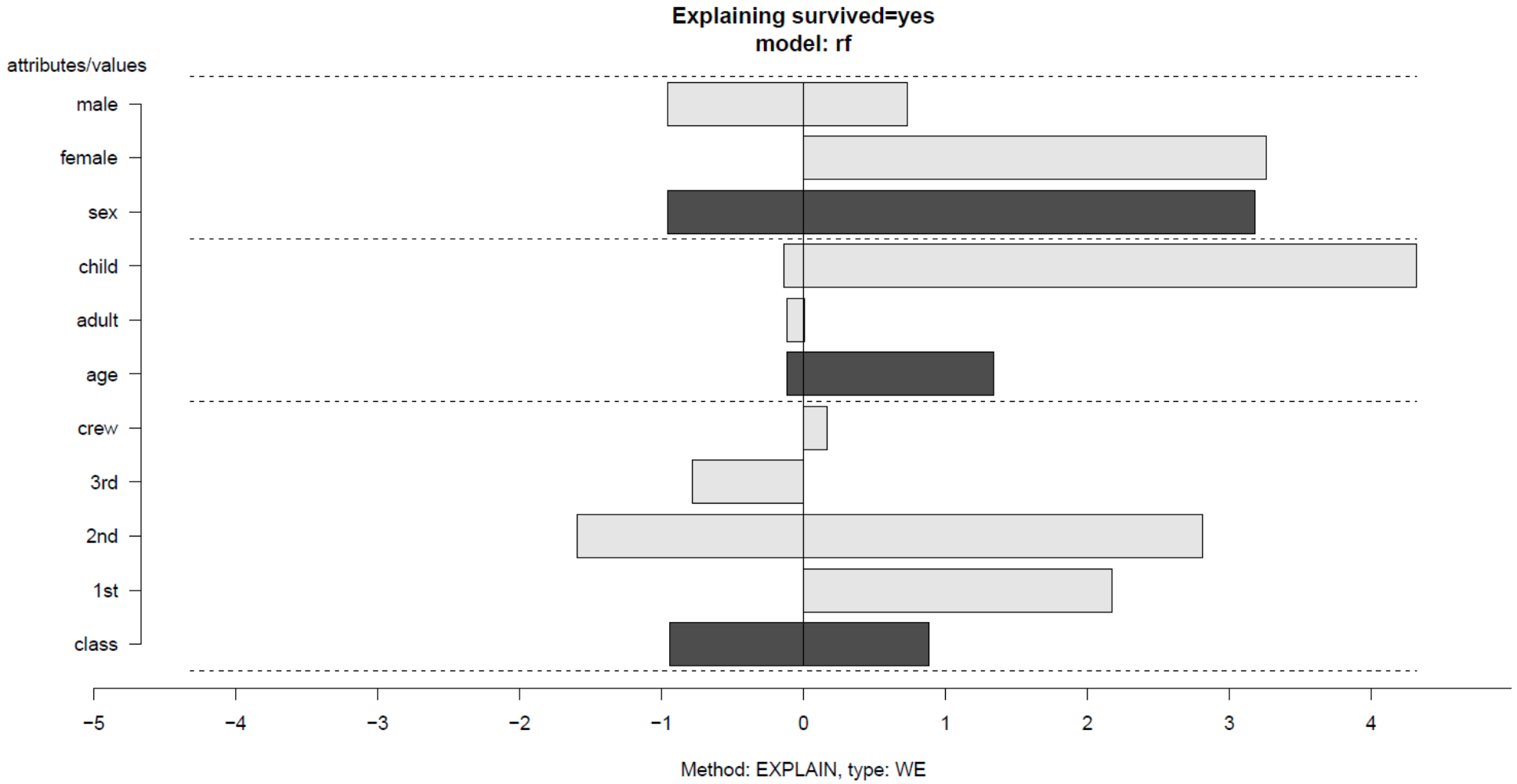


Robnik-Šikonja, M. (2015), ExplainPrediction: Explanation of Predictions for Classification and Regression.
R package version 1.3.0. http://cran.r-project.org/package=ExplainPrediction

University of Ljubljana
Faculty of Computer and Information Science

Visualization of explanations

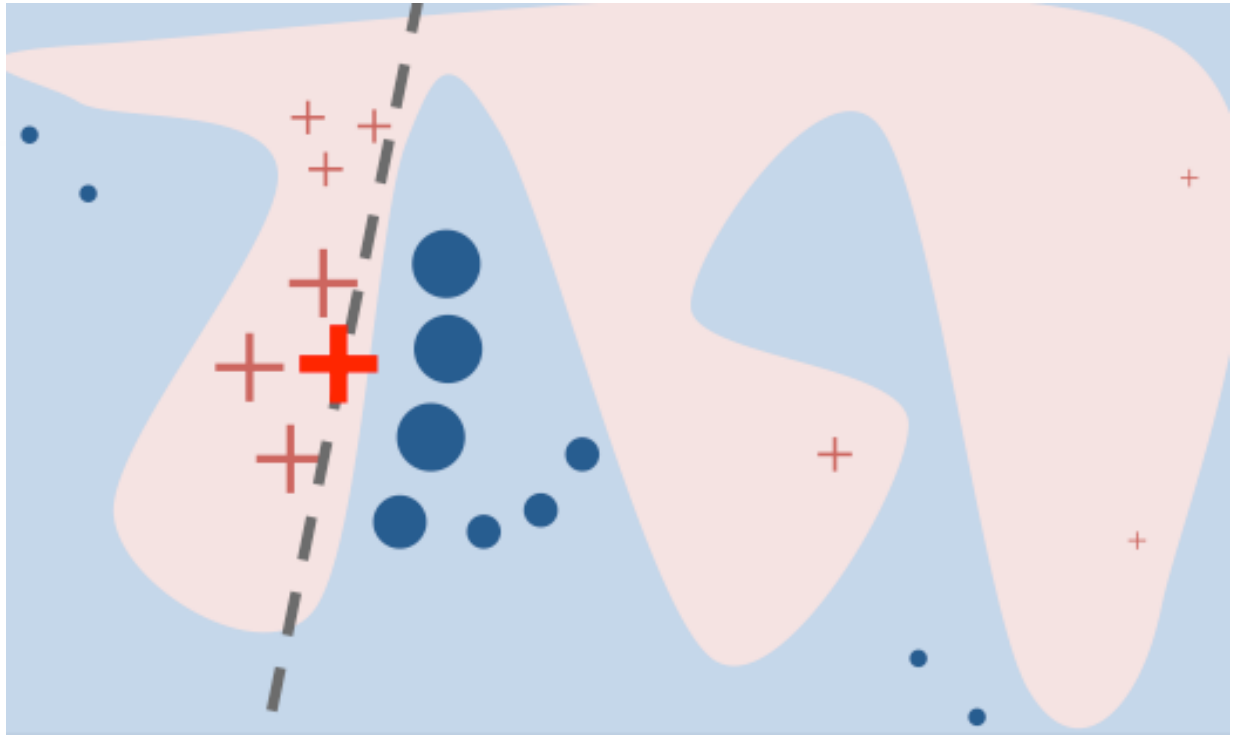
- model-level explanation on Titanic data set



University of Ljubljana
Faculty of Computer and Information Science

LIME explanation method

- Local Interpretable Model-agnostic Explanations)
- perturbations in the locality of an explained instance

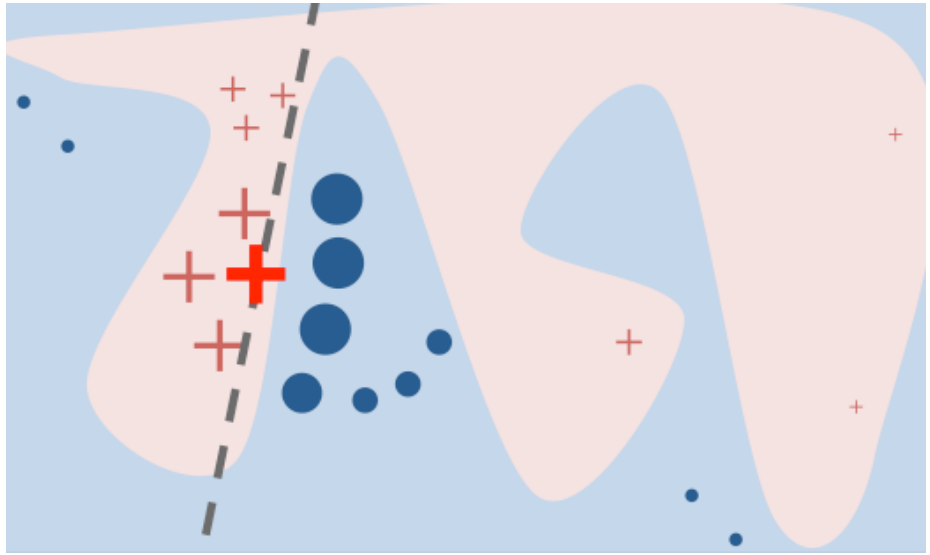


Ribeiro, M. T., Singh, S., & Guestrin, C. (2016). Why should i trust you?: Explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 1135-1144.

LIME explanation method

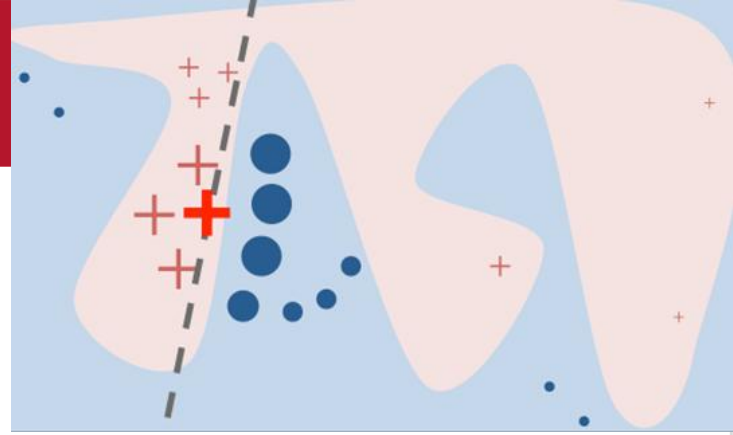
- optimize a trade-off between local fidelity of explanation and its interpretability

$$e(x) = \arg\min_{g \in G} L(f, g, \pi) + \Omega(g)$$



- L is a local fidelity function, f is a model to be explained, g is an interpretable local model g (i.e. linear model), $\pi(x, z)$ is proximity measure between the explained instance *x* and perturbed points *z* in its neighborhood, Ω is a model complexity measure

LIME details



- LIME samples around the explanation instance *x* to draw samples *z* weighted by the distance $\pi(\mathrm{x}, \mathrm{z})$
- samples *z* are used to training an interpretable model *g* (linear model)
- the squared loss measures local infidelity
- number of non-zero weights is complexity
- samples are weighted according to the Gaussian distribution of the distance between *x* and *z*

LIME strengths and weaknesses

- faster than IME
- works for many features, including text and images

- no guarantees that the explanations are faithful and stable
- neighborhood based: a curse of dimensionality
- may not detect interactions due to (too) simple interpretable local model (linear model)



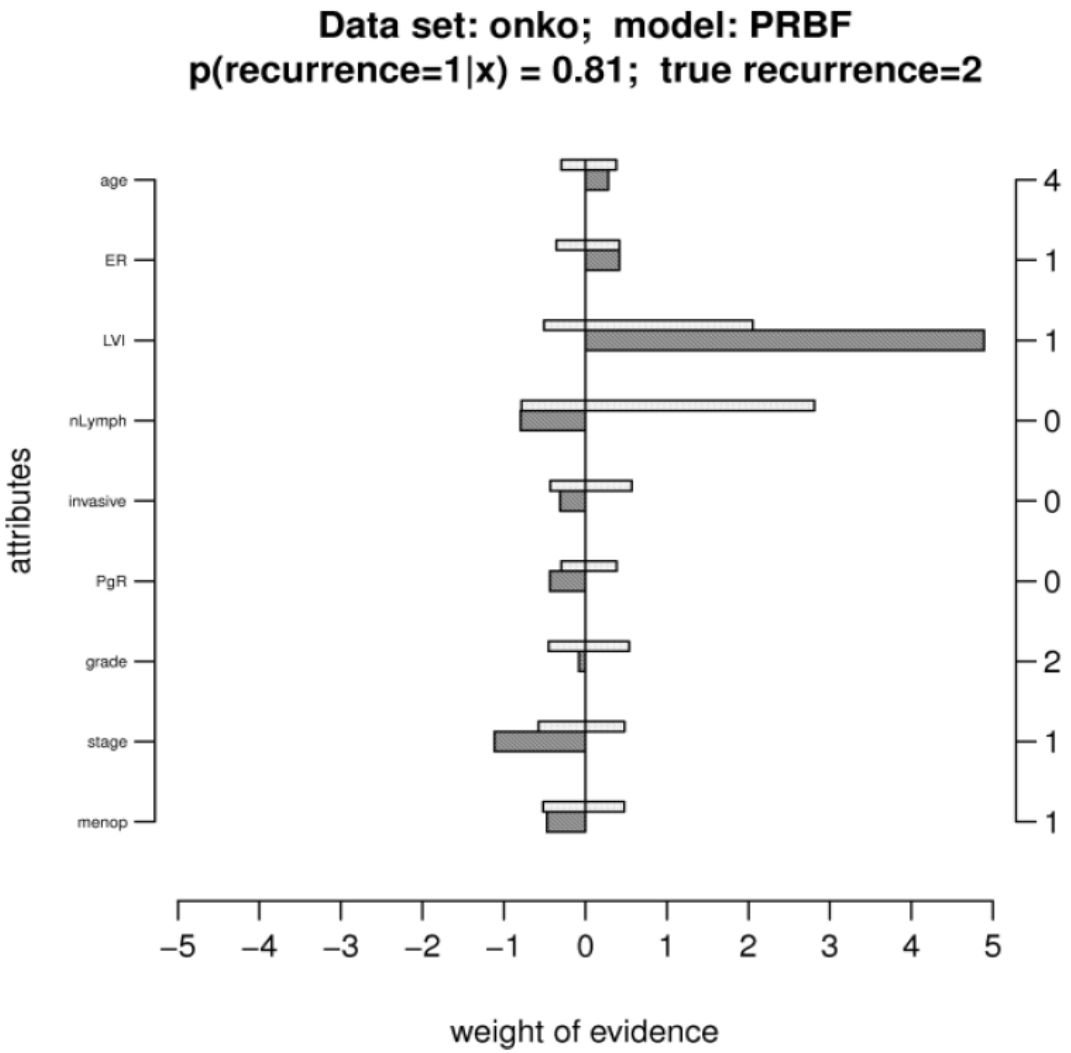
SHAP

- SHapley Additive exPlanation
- unification of several explanation methods, including IME and LIME
- KernelSHAP: based on Shapley values which are estimated using a LIME style linear regression
- faster then IME but
- still uses linear model with all its strengths and weaknesses

Lundberg, S. M., & Lee, S. I. (2017). A unified approach to interpreting model predictions. In Advances in Neural Information Processing Systems (pp. 4765-4774).



Use case: breast cancer recurrence



Cancer recurrence within 10 years

menop binary feature indicating menopausal status

stage tumor stage 1: less than 20mm, 2: between 20mm and 50mm, 3: over 50mm

grade tumor grade 1: good, 2: medium, 3: poor, 4: not applicable, 9: not determined

histType histological type of the tumor 1: ductal, 2: lobular, 3: other

PgR level of progesterone receptors in tumor (in fmol per mg of protein) 0: less than 10, 1: more than 10, 9: unknown

invasive invasiveness of the tumor 0: no, 1: invades the skin, 2: the mamilla, 3: skin and mamilla, 4: wall or muscle

nLymph number of involved lymph nodes 0: 0, 1: between 1 and 3, 2: between 4 and 9, 3: 10 or more

famHist medical history 0: no cancer, 1: 1st generation breast, ovarian or prostate cancer 2: 2nd generation breast, ovarian or prostate cancer, 3: unknown gynecological cancer 4: colon or pancreas cancer, 5: other or unknown cancers, 9: not determined

LVI binary feature indicating lymphatic or vascular invasion

ER level of estrogen receptors in tumor (in fmol per mg of protein) 1: less than 5, 2: 5 to 10, 3: 10 to 30, 4: more than 30, 9: not determined

maxNode diameter of the largest removed lymph node 1: less than 15mm, 2: between 15 and 20mm, 3: more than 20mm

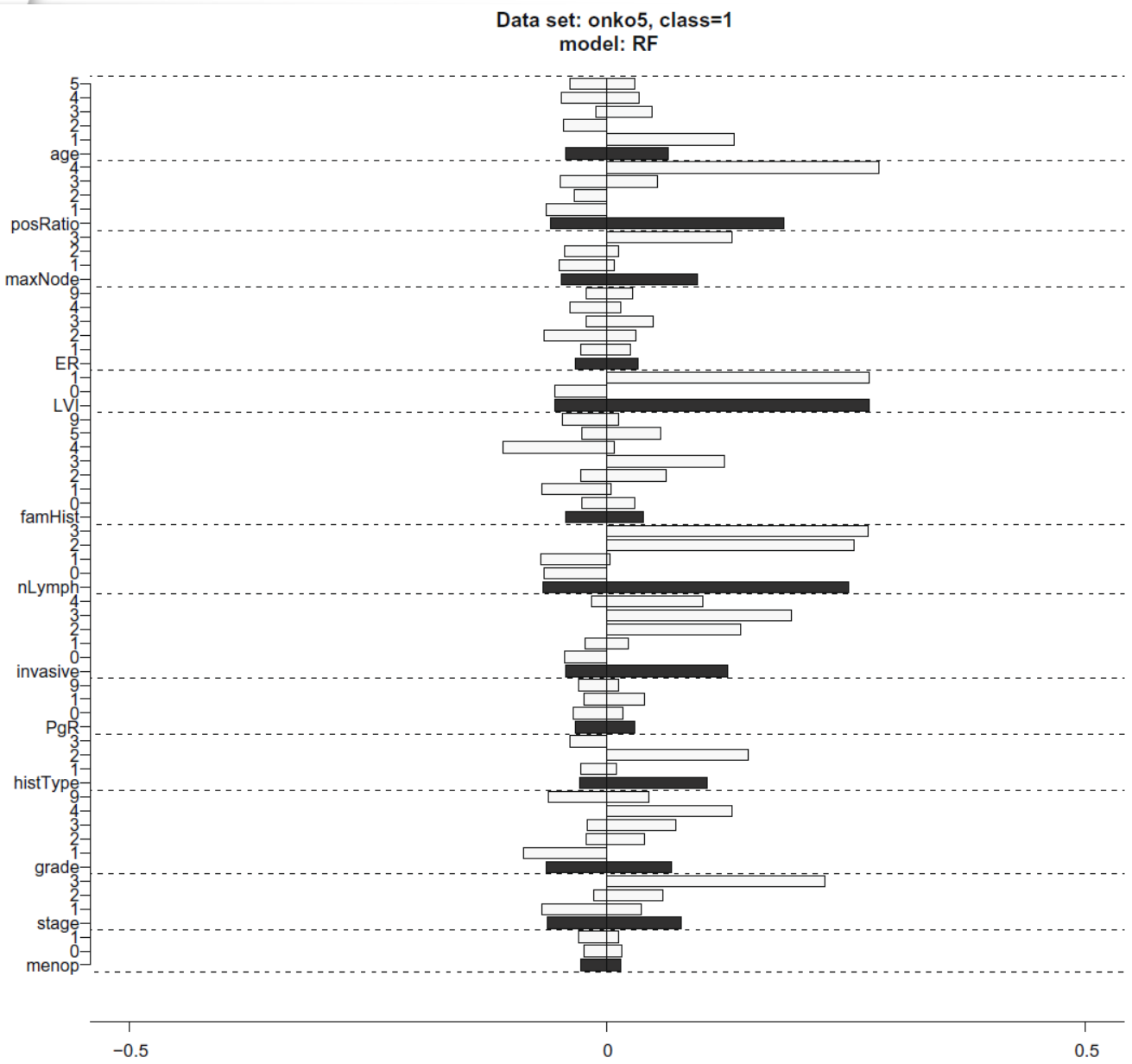
posRatio ratio between involved and total lymph nodes removed 1: 0, 2: less that 10%, 3: between 10% and 30%, 4: over 30%

age patient age group 1: under 40, 2: 40-50, 3: 50-60, 4: 60-70, 5: over 70 years

Robnik-Šikonja, M., Kononenko, I., & Štrumbelj, E. (2012). Quality of classification explanations with PRBF. *Neurocomputing*, *96*, 37-46.

University of Ljubljana
Faculty of Computer and Information Science

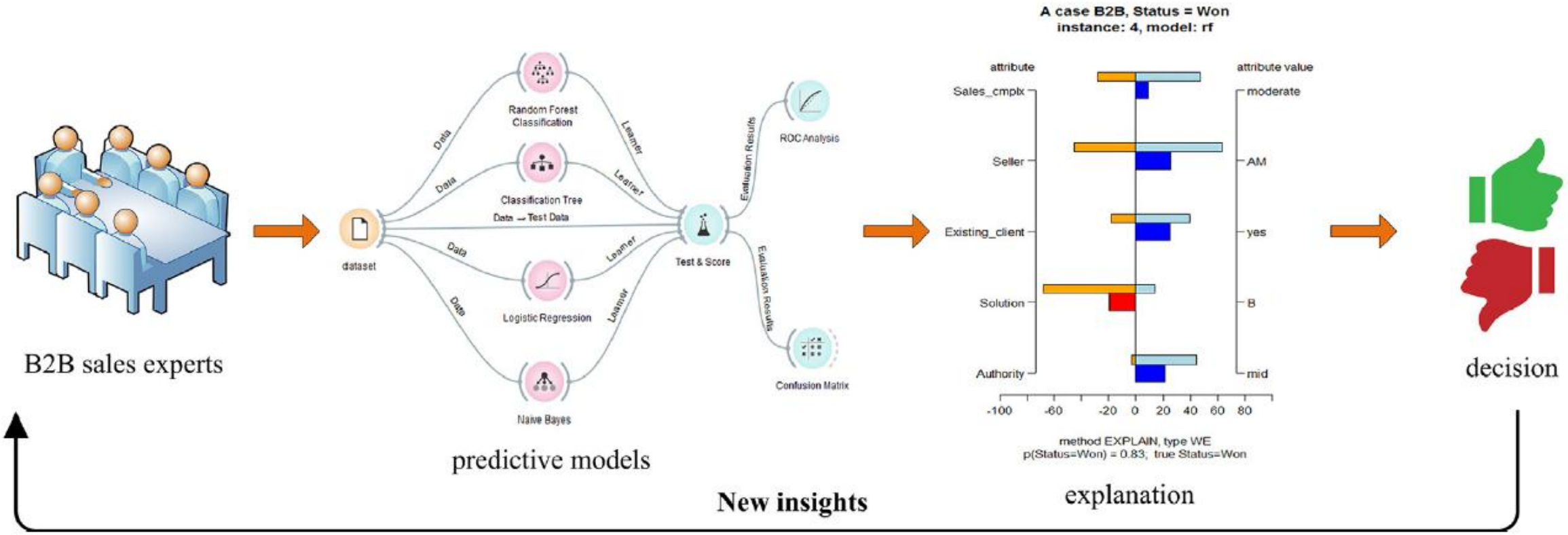
Use case: breast cancer recurrence



University of Ljubljana
Faculty of Computer and Information Science

Use case: B2B sales forecasting

- Goals: improve understanding of factors influencing the outcome and improve the sales performance



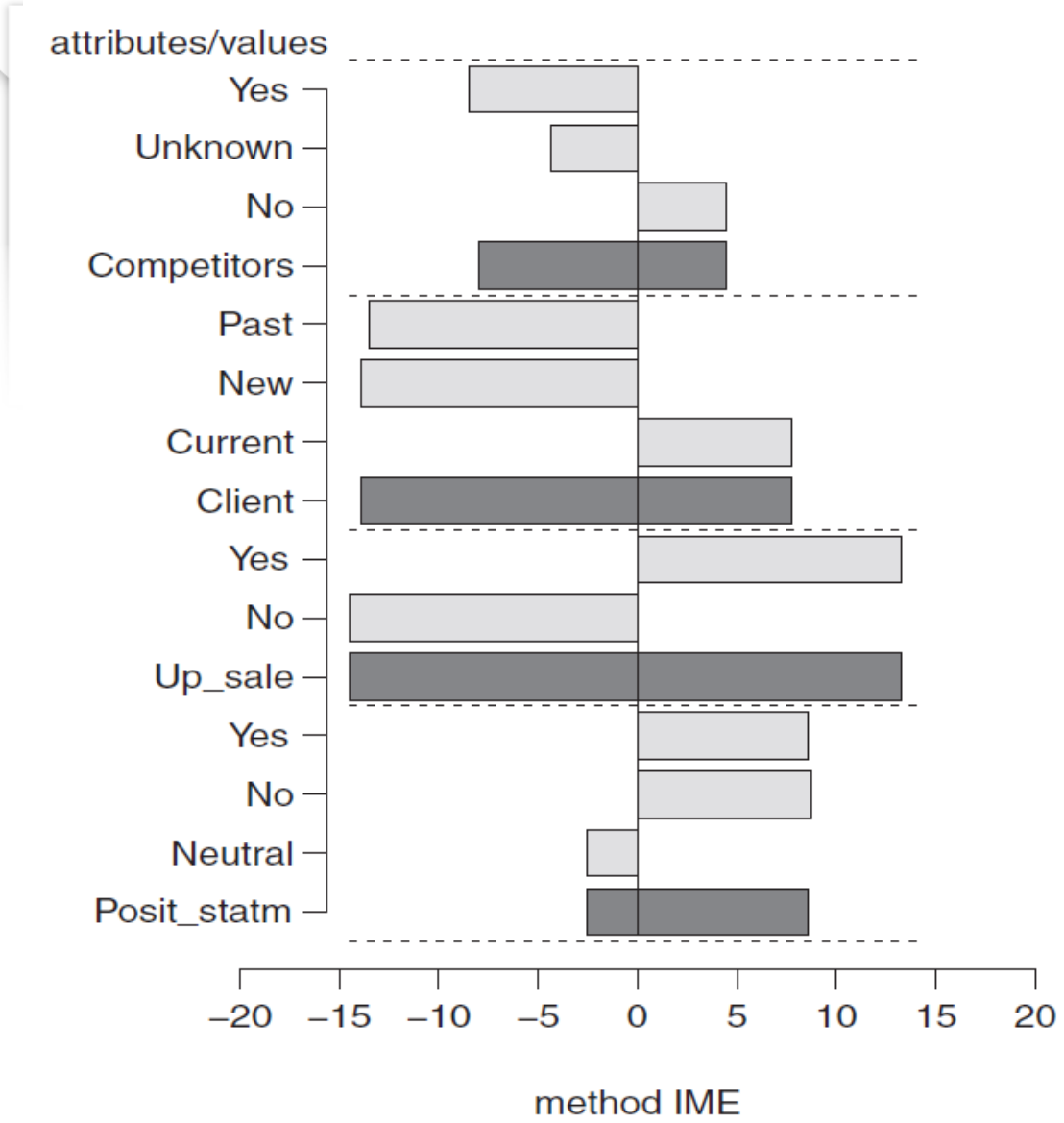
Bohanec, M., Borštnar, M. K., & Robnik-Sikonja, M. (2017). Explaining machine learning models in sales predictions. *Expert Systems with Applications*, *71*, 416-428.

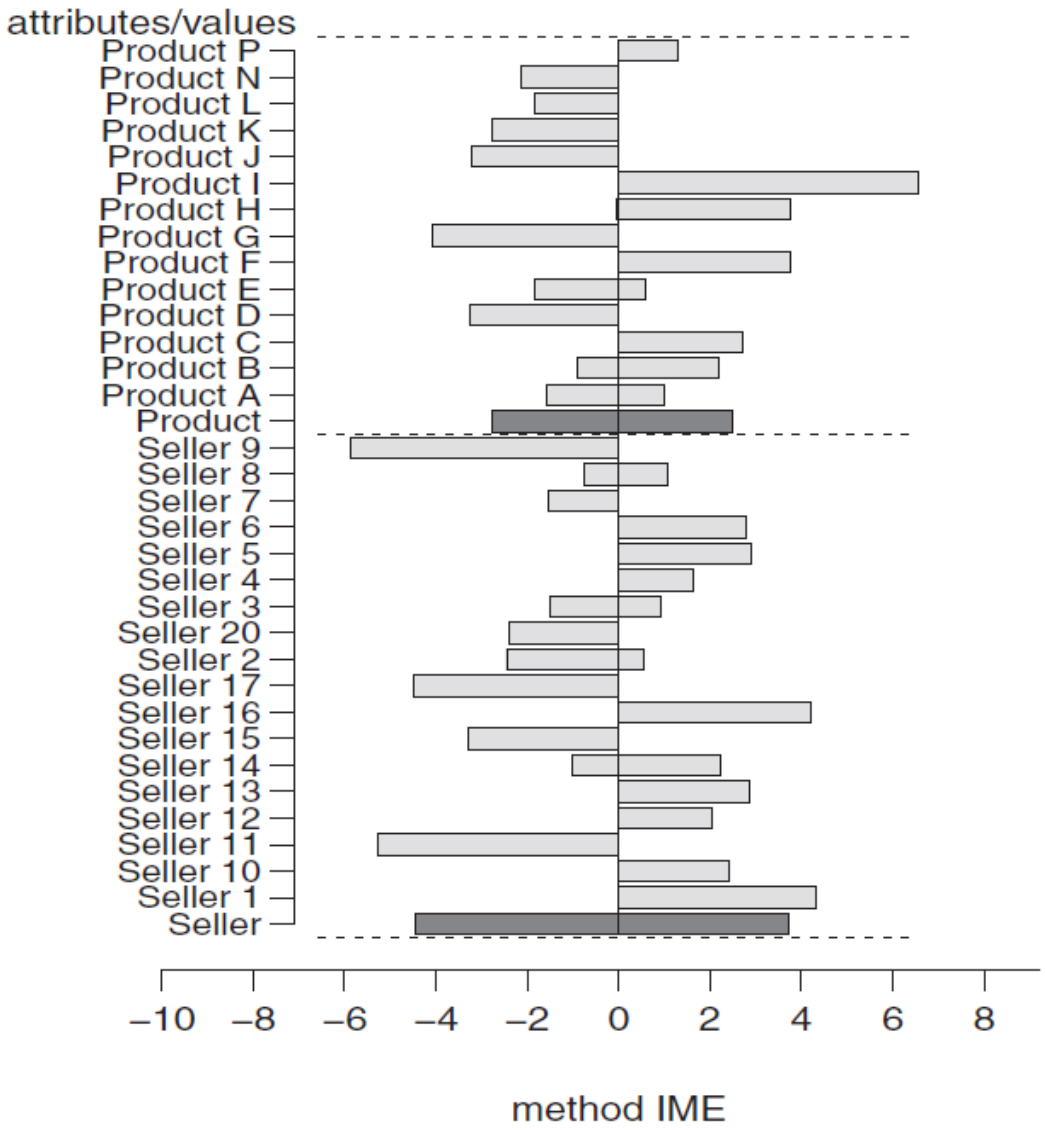
B2B sales attributes

Attribute	Description	Values
Authority	Authority level at a client side	Low, mid, high
Product	Offered product	e.g. A, B, C, etc.
Seller	Seller's name	Seller's name
Competitors	Do we have competitors?	No, yes, unknown
Company size	Size of a company	Big, mid, small
Purchasing department	Is the purchasing department involved?	No, yes, unknown
Partnership	Selling in partnership?	No, yes
Budget allocated	Did the client reserve the budget?	No, yes, unknown
Formal tender	Is a tendering procedure required?	No, yes
RFI	Did we get request for information?	No, yes
RFP	Did we get request for proposal?	No, yes
Growth	Growth of a client?	Growth, stable, etc.
Positive statements	Positive attitude expressed?	No, yes, neutral
Source	Source of the opportunity	e.g. referral, web, etc.
Client	Type of a client	New, current, past
Cross sale	A different product to existing client?	No, yes
Scope clarity	Implementation scope defined?	Clear, few questions, etc.
Strategic deal	Does this deal have a strategic value?	Very important, etc.
Up sale	Increasing sales of existing products?	No, yes
Deal type	Type of a sale	Consulting, project, etc.
Needs defined	Is client clear in expressing the needs?	Info gathering, etc.
Attention to client	Attention to a client	First deal, normal, etc.
Status	An outcome of sales opportunity	Lost, won

University of Ljubljana
Faculty of Computer and Information Science

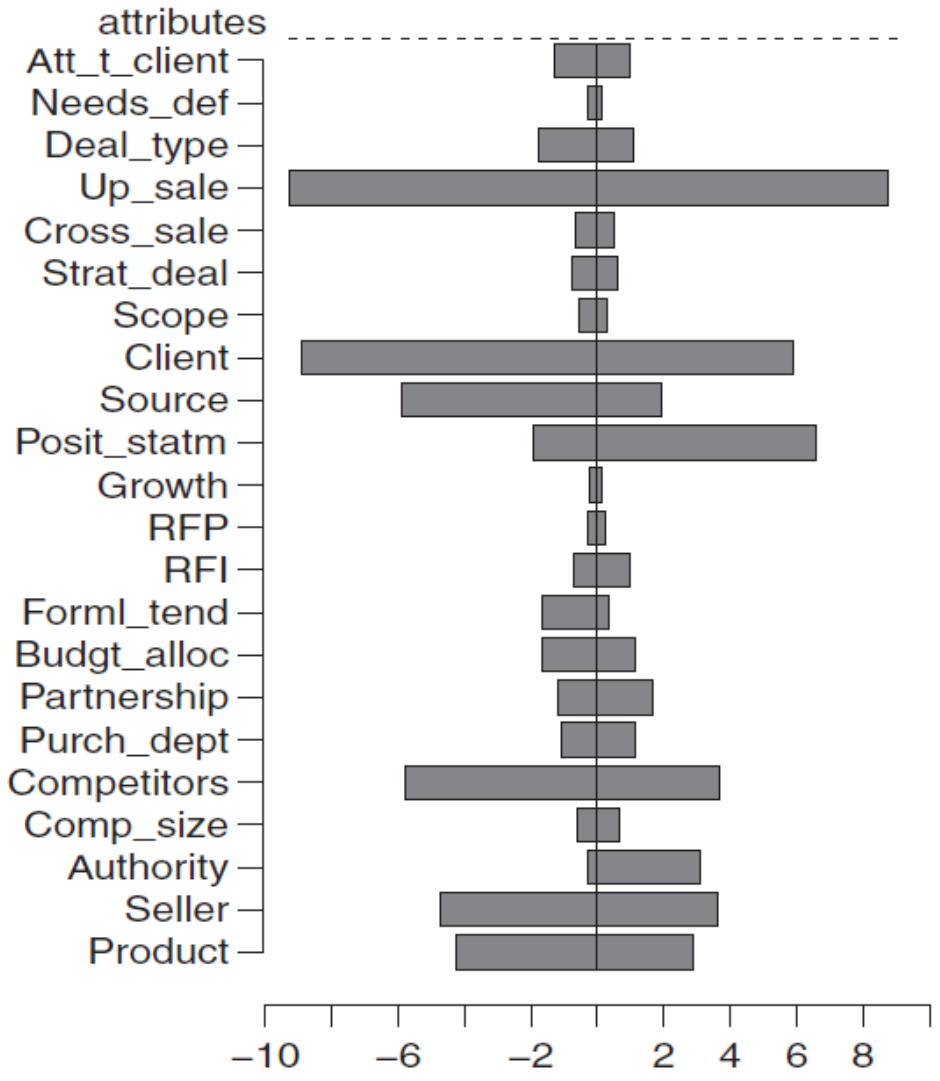
B2B sales: drill in



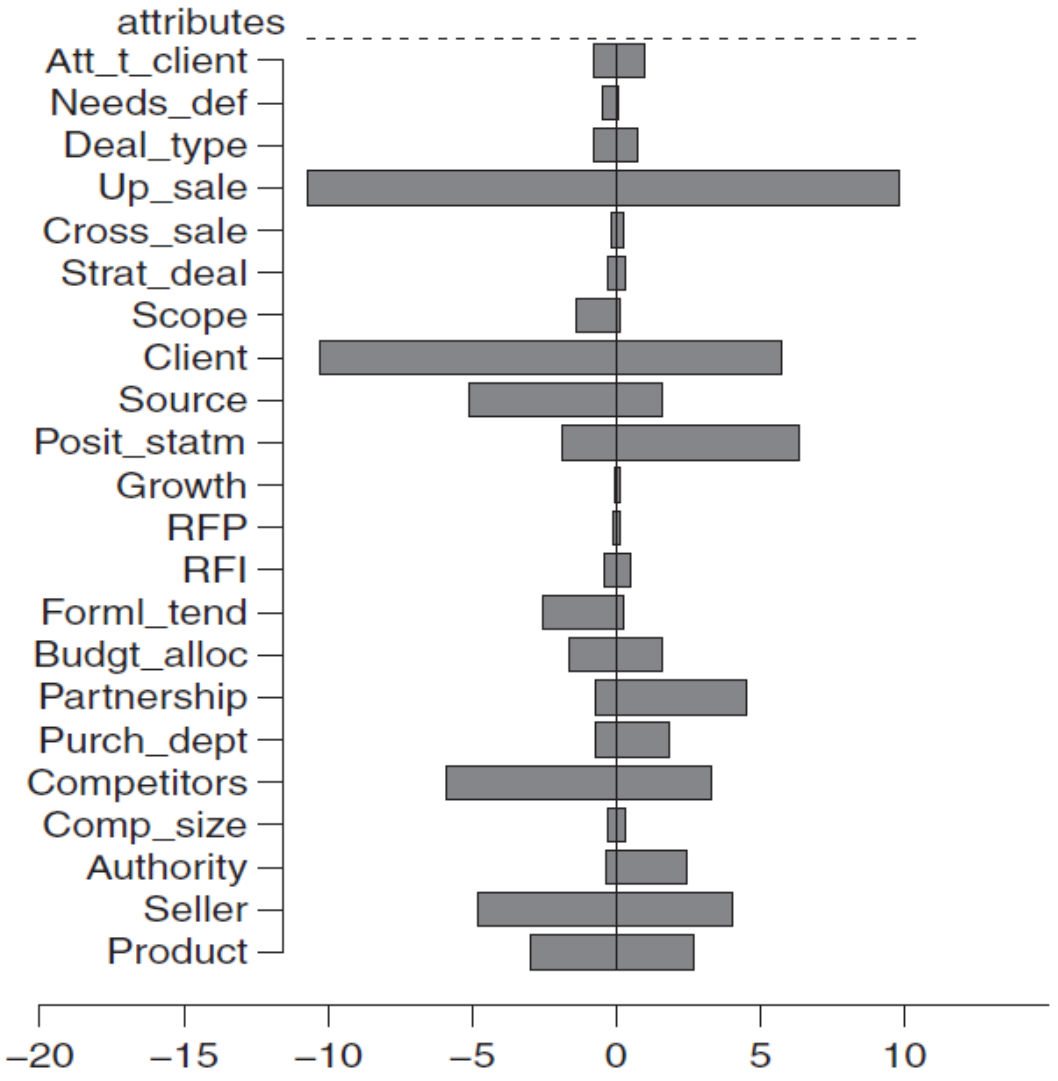


University of Ljubljana
Faculty of Computer and Information Science

B2B sales: EXPLAIN and IME



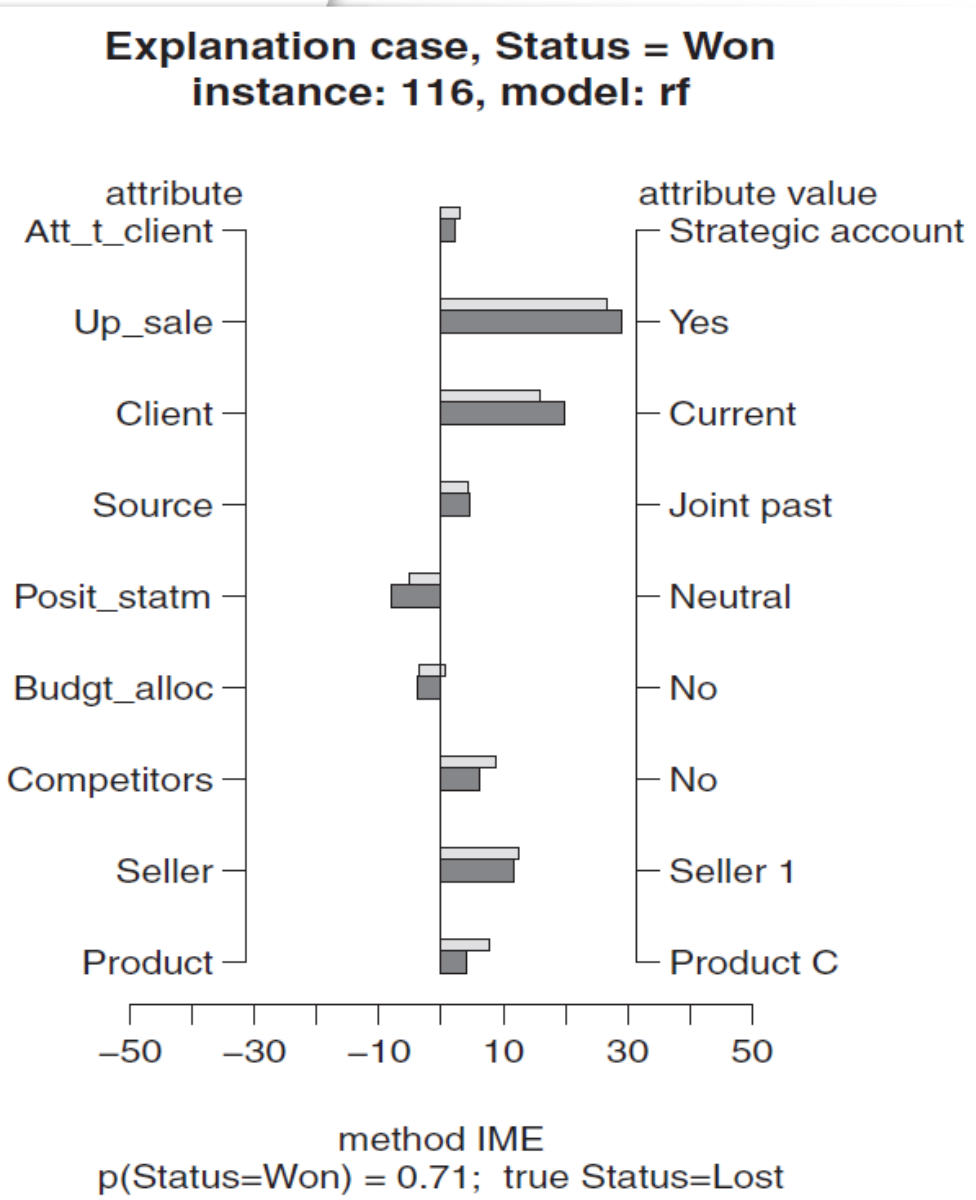
method EXPLAIN, type WE

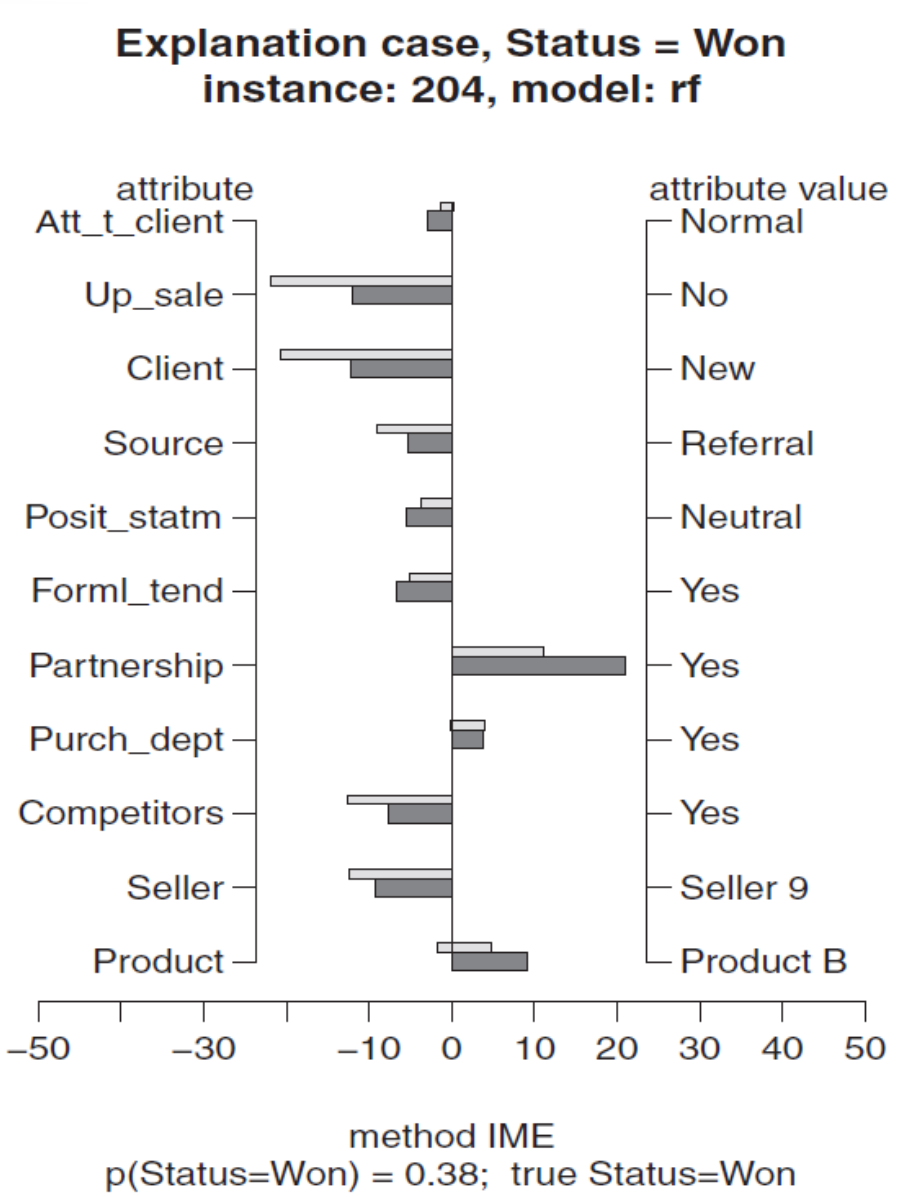


method IME

University of Ljubljana
Faculty of Computer and Information Science

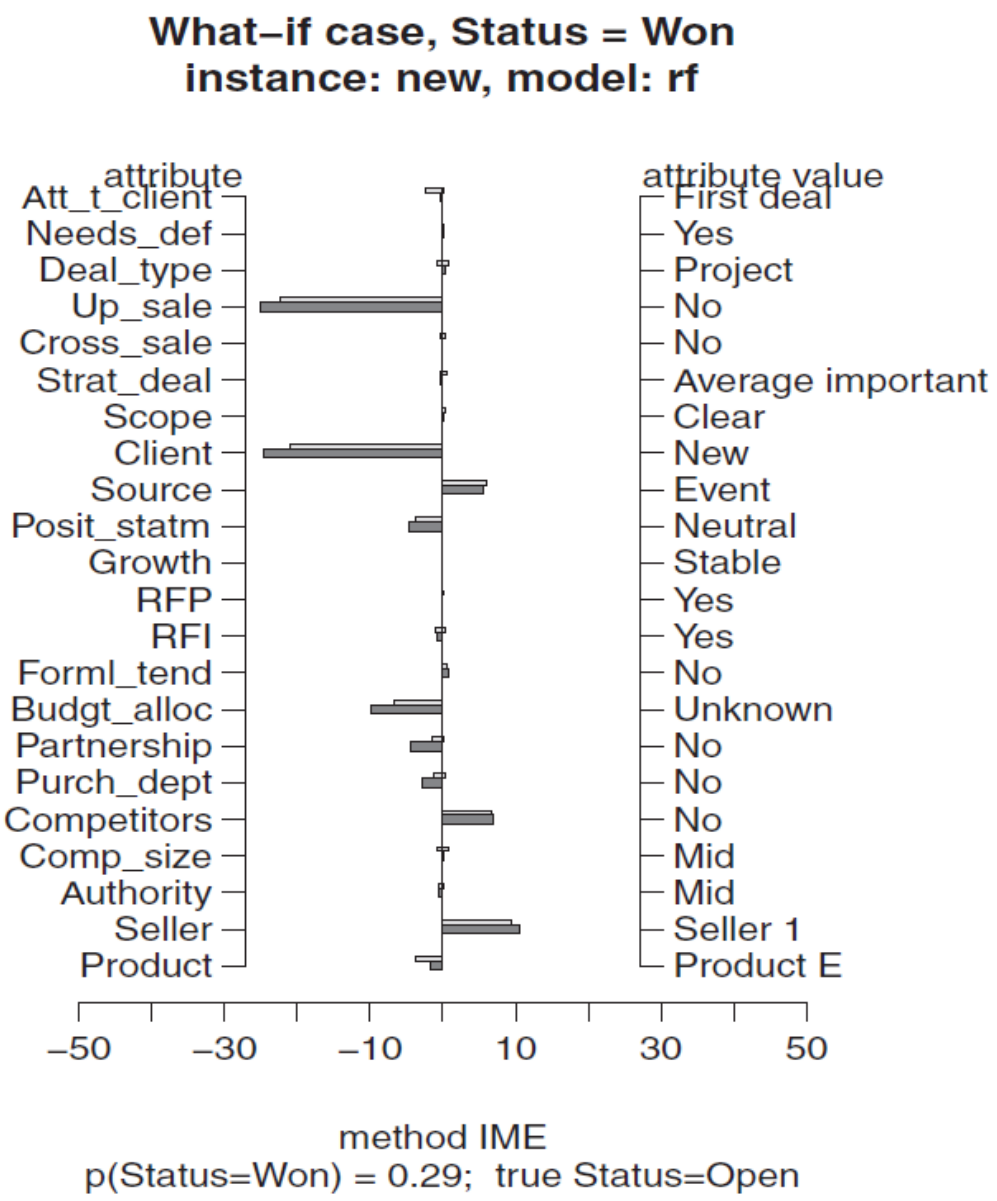
B2B sales: learning from errors





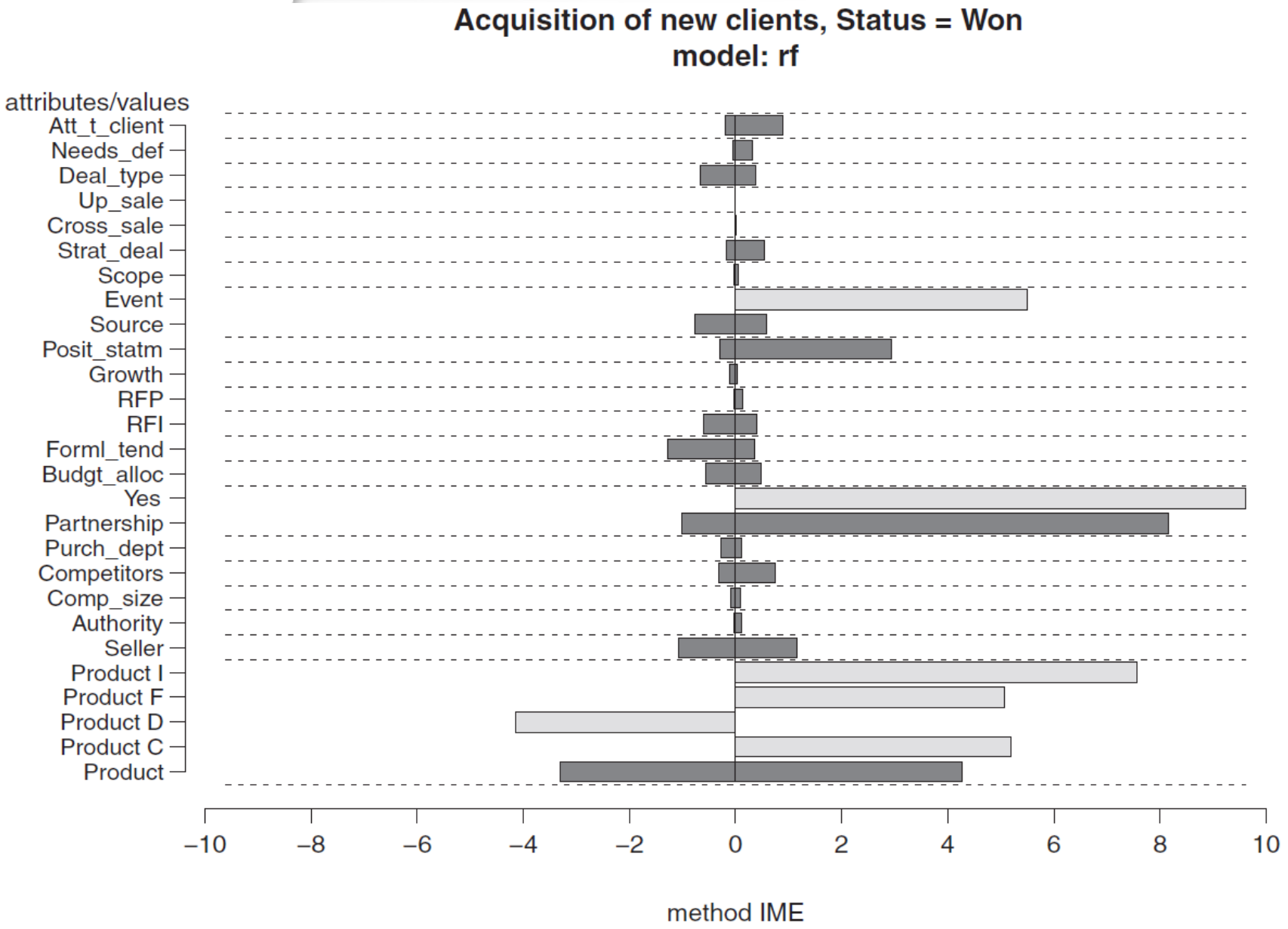
University of Ljubljana
Faculty of Computer and Information Science

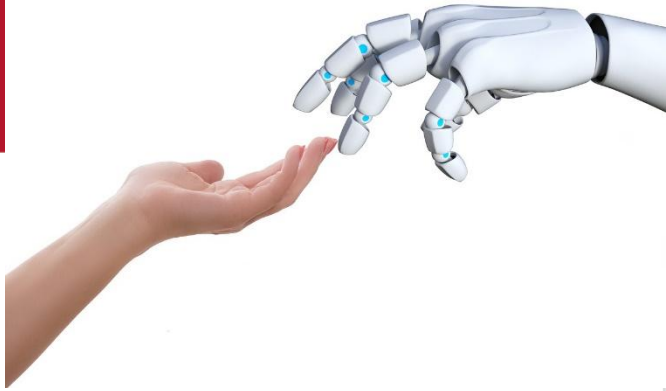
B2B: what if



University of Ljubljana
Faculty of Computer and Information Science

B2B: change of distribution



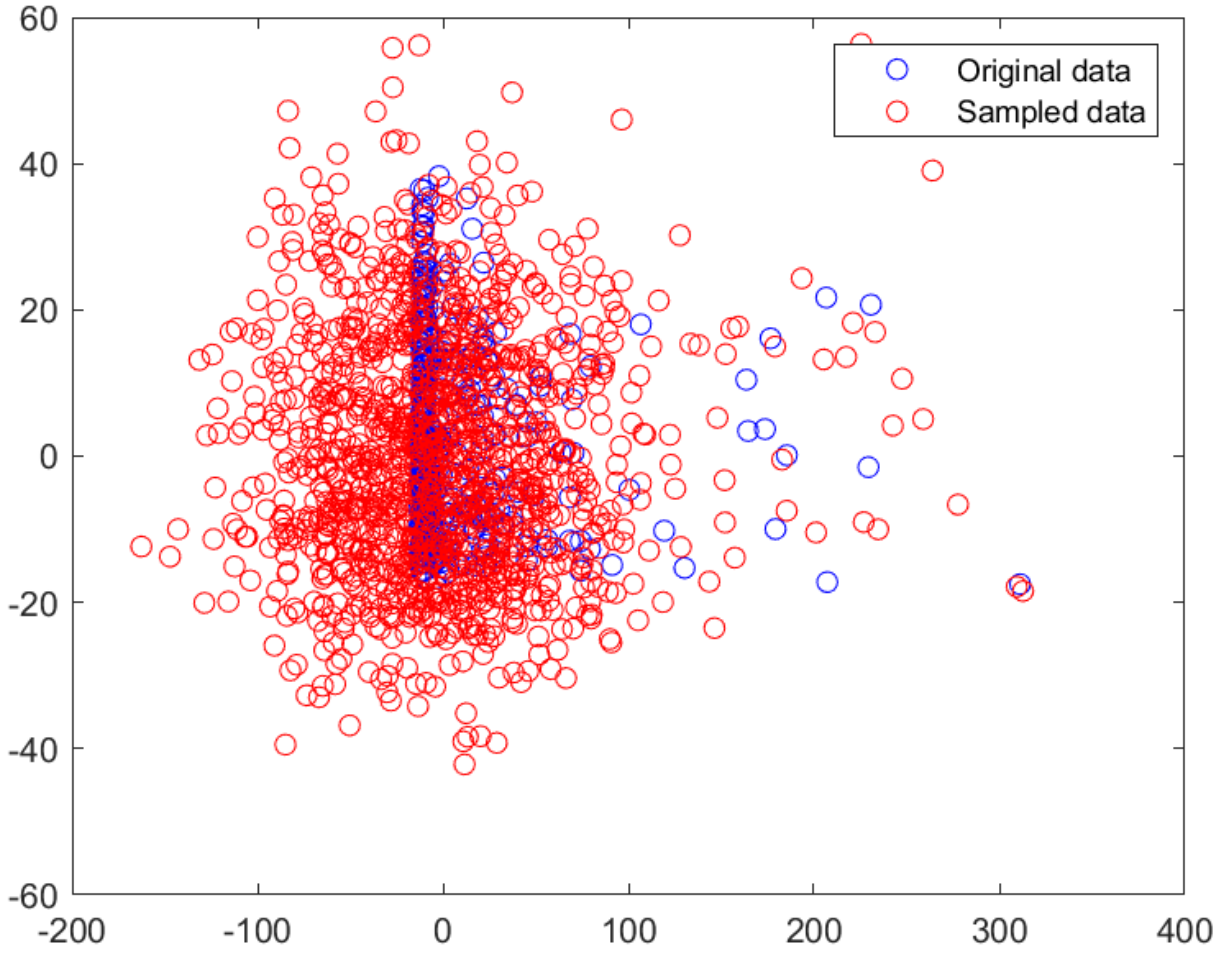


Lessons learned in B2B

- an effort needed to overcome the users' resistance
- human-in-the-loop is necessary to train, discuss, clean data, introduce explanations
- with an increased use, users gain trust in the methodology
- human mental models tend to be biased
- joint interactive approach beats both humans and ML models

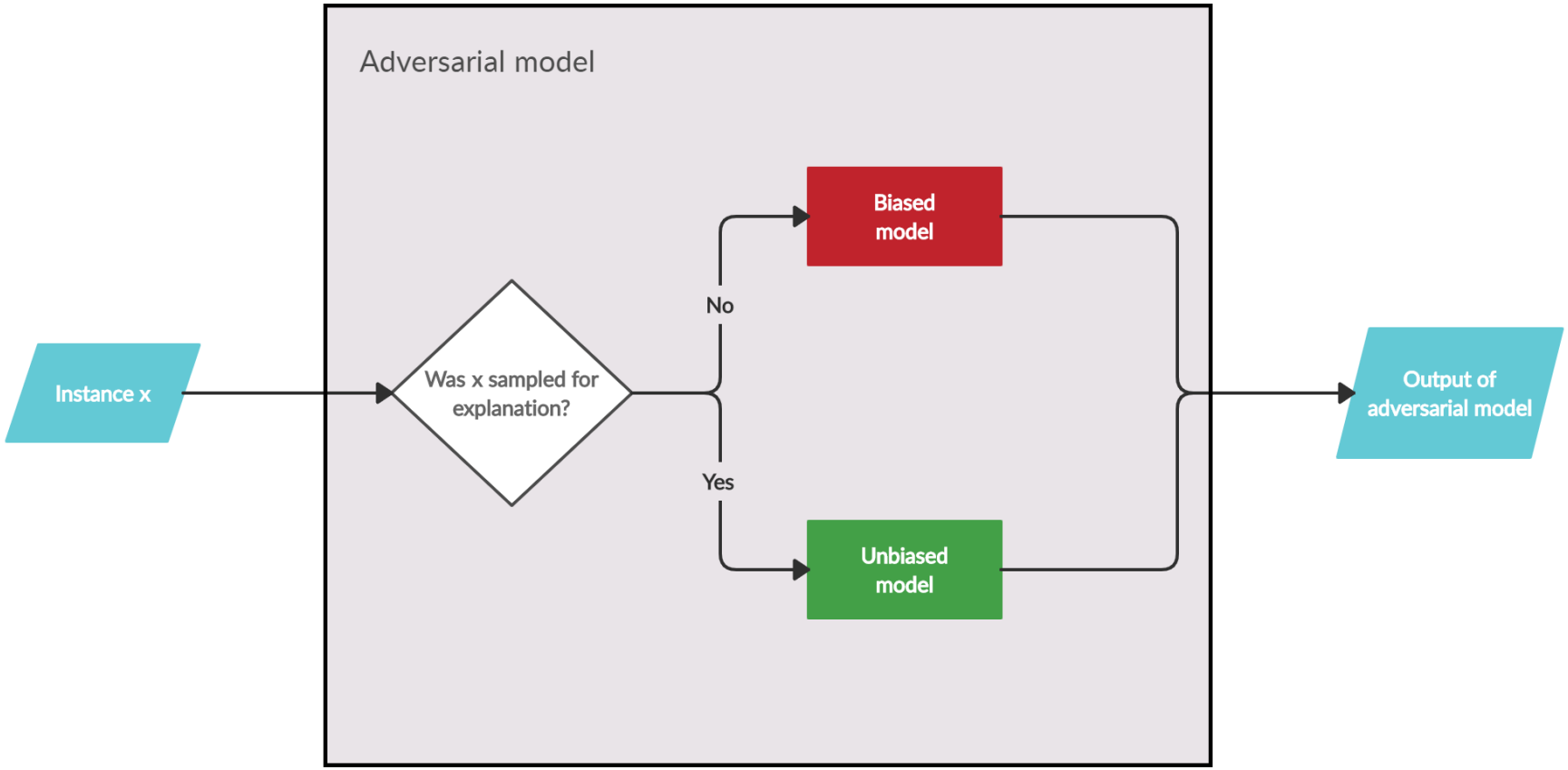
Attacks on explanations

- Poor sampling in explanation approaches makes them vulnerable
- Example: PCA based visualization of a part of the COMPAS dataset; the red dots were generated by LIME



University of Ljubljana
Faculty of Computer and Information Science

Dieselgate attacks on explanations



- Defence: better sampling

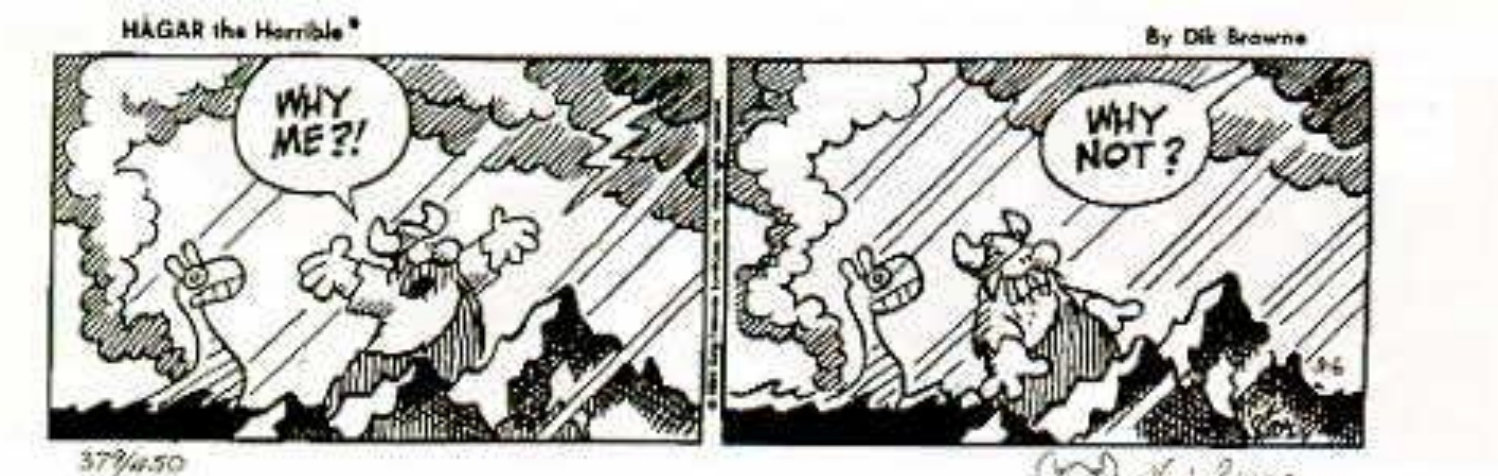
Domen Vreš and Marko Robnik-Šikonja, 2021. Better sampling in explanation methods can prevent dieselgate-like deception. *arXiv preprint arXiv:2101.11702*.

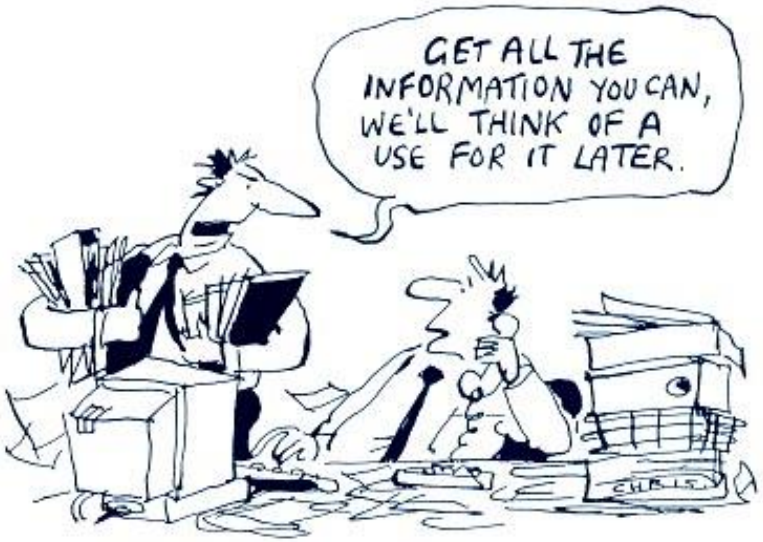
Opportunities

- better and more focused sampling
- better local explanation models
- interactions: detect and describe
- sequences: the order of attributes is important!
- images: decison areas, super-pixels
- better visualizations: human cognitive limitations
- explanations is also domain specific, we need explanation datasets

Conclusions

- many successful mechanistic explanation approaches, mostly for tabular classification problems
- LLMs are trained to explain their behavior for particular important problems
- lots of opportunities for improvements
- even human explanations are not necessarily comprehensible
- humans often explain by providing background or additional knowledge
- legal and practical need for explanations of ML models





University of Ljubljana, Faculty of Computer and Information Science

Transformers for tabular data and time series



Prof Dr Marko Robnik-Šikonja

Intelligent Systems, Edition 2025

Contents

- Transformers for tabular data
- Foundation models for tabular data
- Transformers for time series (TS) data
- Foundation models for TS data

Traditional tabular data approaches vs transformers

- Traditional tabular-data models (like XGBoost, Random Forests, SVM) often outperform deep neural networks because:
  - Tabular features can be heterogeneous (numeric, categorical, ordinal)
  - Feature interactions are complex and often non-spatial (unlike images or text)
  - There is no inherent ordering between columns
  - Dataset sizes are often small or medium, not millions of samples
- Transformers succeed in NLP and vision partly because inputs have natural sequential or spatial structure, which tabular data lacks
- How do we adapt Transformers—originally for sequences—to unordered sets of features?

Adapt transformers to tabular data

- Treat each input feature as a “token”
- Use attention to automatically learn feature interactions
- Treat features as a set, not a sequence

Treat each input feature as a “token”

- In NLP, tokens = words.
  In tabular transformers, tokens = columns/feature embeddings
- Each original feature becomes a vector embedding:
  - numeric feature → learned numerical embedding
  - categorical feature → learned category embedding (similar to word embeddings)
  - missing-value tokens are possible
- The final input is a set of feature embeddings, e.g.:

$$x \rightarrow \{e_1, e_2, \ldots, e_d\}, \text{one per feature}$$

Automatically learn feature interactions using attention

- Transformers compute pairwise interactions:

$$\text{Attention}(Q, K, V) = softmax\left(QK^{\top}/\sqrt{d}\right)V$$

- This is **useful for tabular data**, because many tasks depend on feature interactions such as:
  - Age × Income
  - Temperature × Humidity
  - Previous purchase × Current product
- Traditional models like XGBoost capture interactions through tree splitting.
- Transformers capture them via **attention weights** between features.

Treat features as a set, not a sequence

- Unlike text, tabular features don’t have left/right neighbors.
- So we use:
  - Set embeddings (no positional encoding) OR
  - Learned positional embeddings representing feature identity (not dependent on order, only on feature name)
- Some models use:
  - Feature tokens
  - Column embeddings
  - Field embeddings (groups of features)
- This gives the model awareness of *which* feature it is attending to.

Token mixing as feature engineering

- Token mixing gives expressive non-linear feature engineering
- Transformers can learn high-order feature interactions implicitly.
  Example:
  - A purely linear model cannot model things like “high income but low age → risky borrower.”
  - XGBoost learns this by splitting; Transformers learn it through multi-head attention layers.
- Multi-head attention allows different heads to learn different types of interactions. E.g.,
  - Head 1: correlations between financial variables
  - Head 2: pattern of missing values
  - Head 3: categorical → numeric interactions
  - etc.
- This is useful for tabular tasks.

Joint handling of categorical and numerical features

- Transformers handle categorical and numerical features jointly
- Advantage: instead of one-hot encoding or target encoding, categorical columns become embedding vectors, just like words in NLP.
- Numeric features can be processed by:
  - binning + embedding
  - normalization + small feedforward projection
  - continuous-value embeddings (similar to positional encodings)
- This eliminates the need for feature engineering.

Example of encoding

- Suppose a dataset contains these features:

Feature	Type	Example Value
Age	Numeric	45
Income	Numeric	60,000
Job Type	Categorical	"Engineer"
City	Categorical	"Paris"

- We want to encode this row:

x_row = { Age=45, Income=60000, Job="Engineer", City="Paris“}

Embedding numeric features

- For **numeric** features, we usually apply:
  - normalization (optional)
  - a small feedforward projection (linear layer)
- Example:

AgeEmbedding = LinearNum_Age(45) # vector in R^d_model

IncomeEmbedding = LinearNum_Income(60000) # vector in R^d_model

Assuming d_model = 32 we get

AgeEmbedding → e_age (32-dim vector)

IncomeEmbedding → e_income (32-dim vector)

Embedding categorical features

- For categorical columns, we use learned embedding tables, analogous to NLP

Engineer → Embedding(JobType)[Engineer] → e_job (32 dims)

Paris → Embedding(City)[Paris] → e_city (32 dims)

- All categorical embeddings are learned from data, just like word embeddings.
- A learnable embedding matrix maps each category to a dense vector.
- This embedding is updated via gradient descent during training (e.g., if being an engineer increases the probability of prediction, the embedding will be shifted up by the gradient)
- These embeddings replace one-hot encoding with a compact, expressive representation.
- During training the embeddings capture statistical patterns, feature interactions and semantic similarity (unintentional but common)

Column embeddings (feature identifiers)

- Each feature has a learned **column embedding:**

Feature	Column Embedding
Age	col_age
Income	col_income
Job	col_job
City	col_city

- These tell the model *which feature this token corresponds to*, since tabular features have no natural order.

Merge features and their names

Token_Age = e_age + col_age
Token_Income = e_income + col_income
Token_Job = e_job + col_job
Token_City = e_city + col_city

- Now each token knows both:
  - the value (through numerical or categorical embedding)
  - the identity of the feature (column embedding)

Forming an instance

- Optional: prepend a [CLS] token
  A learnable CLS token collects global information (like in BERT):
  Token_CLS = cls_vector (32 dims)
- Final sequence fed into Transformer
- [Token_CLS, Token_Age, Token_Income, Token_Job, Token_City]
- Each token is a vector of length 32, so input shape is 5 tokens × 32 dimensions

Together

Age=45 → Linear → e_age + col_age = Token_Age
Income=60000 → Linear → e_income + col_income = Token_Income
Job="Engineer“ → Embed → e_job + col_job = Token_Job
City="Paris" → Embed → e_city + col_city = Token_City

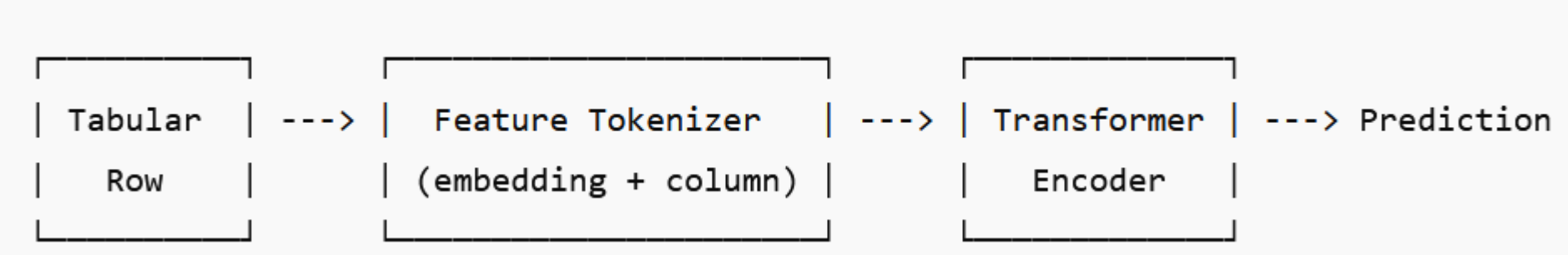
Final sequence:[CLS, Token_Age, Token_Income, Token_Job, Token_City]

- This sequence is then passed through multi-head self-attention, feed-forward layers, pooling, and final prediction head

TabTransformer

- 2020
- Categorical columns are embedded
- A Transformer stack learns interactions *between categorical features*
- Outputs feed into an MLP for final prediction
- Shown to outperform one-hot + deep networks.

FT-Transformer



- Feature Tokenizer (FT) Transformer (Gorishniy et al., 2021)
- Every feature = token
- Add learnable column embeddings
- Apply transformer blocks directly
- Pool the output for prediction
- Often matches or slightly beats XGBoost on some datasets.

Comparison to XGBoost

- **Advantages**
  - Learns high-order interactions automatically
  - Works well with raw categorical features
  - End-to-end differentiable
  - Scales well with data
  - Great for multimodal fusion (text + image + tabular)
- **Disadvantages**
  - Needs more data than XGBoost
  - Harder to tune
  - Computationally expensive
  - XGBoost still wins on many small tabular datasets

Foundation models for tabular data

- Foundation models for tabular data are large pretrained Transformer-based models that learn general-purpose representations of structured data across many datasets so they can be fine-tuned for any downstream tabular ML task.
- This is analogous to:
  - BERT → pretrained on text → fine-tuned for classification, QA
  - ViT → pretrained on images → fine-tuned for detection, segmentation
  - Tabular FM → pretrained on millions of tables → fine-tuned for forecasting, classification, regression, anomaly detection, etc.

Goals of foundation models

- A tabular foundation model aims to:
  - Learn universal patterns of tabular data
  - Capture relationships between features, categories, numeric distributions
  - Generalize across diverse datasets and domains
  - Enable training on low-data regimes (few-shot)
  - Replace hand-crafted feature engineering and classical ML baselines
- This is a major shift because historically, tabular models have been task-specific and dataset-specific (e.g., XGBoost, RF, SVM, etc).

Tabular Data Is Hard for Foundation Models

- Tabular data has challenges absent in text/images:
- Mixed feature types
  - numeric
  - categorical
  - ordinal
  - high-cardinality categories
  - missing values
- No natural spatial or sequential structure
  - Transformers succeed in domains with structure (linguistic or spatial). Tables have neither.
- Diverse ranges, distributions, and semantics
  - Income, temperature, stock price, age, ZIP code—vastly different.
- Datasets are smaller
  - Most tabular datasets have thousands (not millions) of rows.

Core approach to Tabular Foundation Models

1. Feature Tokenization
2. Large-Scale Pretraining on Many Tables
3. Universal Embedding Space for Features and Tables
4. Typically, only transformer encoder architecture with classification head

Feature Tokenization

- Each feature (numeric or categorical) is represented as a token, similar to words in NLP.
- For each training instance (row):

  [Token_age, Token_income, Token_job, Token_zipcode, Token_missing_mask, ...]

- This enables Transformers to model feature–feature interactions via self-attention.
- Popular method is FT-Transformer (simple and effective)

Large-Scale Pretraining on Many Tables

- Similar to BERT learning English by reading the entire Internet,
- Tabular foundation models learn from:
  - many tabular datasets across domains
  - synthetic tables
  - database dumps
  - business and scientific datasets
- Pretraining strategies include:
  - Masked-cell modeling: Mask random feature values and predict them (like BERT's MLM):
    Predict $x_{i,j}$ *from the rest of the row and col.*
  - Row contrastive learning: make representations of rows within the same dataset consistent.
  - Causal modeling of numeric distributions: Learn typical shapes of distributions (skewed, heavy-tailed, etc).
  - Schema-aware training: Teach the model that some features belong to the same table schema (e.g., Age, Zip_code, etc.)
- Pretraining creates universal representations.

Universal Embedding Space for Features and Tables

- The model learns:
  - embeddings for numeric values
  - embeddings for categorical values
  - embeddings for feature indices (column embeddings)
  - embeddings for table schemas
- Representation unification is key.
- Examples:
  - ZIP codes in many datasets share distributional patterns
  - Age, salary, population, etc., have a consistent statistical structure
  - Categories such as job types may share semantics across tables
- The foundation model captures these.

Architecture of Tabular Foundation Models

- While several architectures exist, most rely on:
- Transformer encoder handles feature interactions.
- Feature tokenizer encodes numeric + categorical features + metadata into tokens.
- Schema embeddings capture the meaning of table columns.
- Missing value embeddings represent missingness explicitly (often predictive).
- Prediction head is added during fine-tuning for regression/classification.

Pretraining Example: Masked Cell Modeling

- Given a row:

  Age=45, Income=60000, Job=Engineer, City=Paris
- Randomly mask:

  Age=[MASK], Income=60000, Job=Engineer, City=Paris
- Goal:

  Predict Age
- Same for categorical and numerical values.
- This forces the model to learn relationships:
  - Income ↔ Job
  - Job ↔ City
  - City ↔ Age
  - Missingness patterns

Benefits of Tabular Foundation Models

- Few-shot learning: Fine-tune on small datasets (hundreds of rows, not thousands).
- Cross-domain generalization: One pretrained model → many downstream tasks.
- No manual feature engineering: Embeddings + self-attention learn feature interactions automatically.
- Powerful transfer learning: Models pretrained on many tables outperform models trained from scratch.
- Interpretability: Attention maps may highlight feature interactions.
- Multimodal fusion capabilities: Foundation models naturally integrate:
  - text columns (via tokenization)
  - image fields (via vision embeddings)
  - time series fields
  - knowledge graph context
- Several pretrained models exist, e.g., TabLLM, TabPFN, etc.

Transformers for time series

Why Use Transformers for Time Series?

- Traditional sequence models have limitations:
  - RNNs/LSTMs struggle with long-range dependencies and parallelization
  - CNNs have fixed receptive fields unless very deep
  - Classical models (ARIMA, ETS) handle only simple patterns
  - Tree-based models do not capture sequential structure
- Transformers solve these problems by:
  - using *self-attention* to capture relationships between *all time steps at once*
  - enabling global context modeling
  - supporting multivariate time series naturally

Core idea: time steps are tokens

- Represent Each Time Step as a Token

  A time series: $x_1, x_2, x_3, \ldots, x_T$

  becomes a sequence of tokens, just like words in a sentence.
- For multivariate time series:

$$x_t = \left[x_t^{(1)}, x_t^{(2)}, \ldots, x_t^{(d)}\right]$$

- Each time step gets encoded into a vector using:
  - projection (linear layer) for numeric features
  - convolutional/embedding preprocessing for high-dimensional data
- Result:

  Embedding_1, Embedding_2, ..., Embedding_T

Add Positional Encoding!

- Unlike NLP text with word order, time series are *explicitly ordered*. Transformers need to know time step positions.
- Two approaches:
  - 1. Sinusoidal positional encoding
  - 2. Learnable positional embeddings
- The model learns a vector for each time step index.
- These are added to the step embeddings:

$$z_t = Embed(x_t) + PosEmbed(t)$$

- This gives meaning to the temporal order.

Self-Attention Learns Temporal Dependencies

- Self-attention computes how much each time step should attend to others:

$$\text{Attention}(Q, K, V) = \textit{softmax}\left(\frac{QK^{\top}}{\sqrt{d}}\right)V$$

Where:

  - Q = queries (what each time step wants to know)
  - K = keys (what each time step provides)
  - V = values (information being passed)
- In a time series, each time step can attend to any other time step.
- This lets the model learn:
  - short-term dependencies
  - long-term seasonal patterns
  - periodicity
  - sudden regime shifts
  - interactions between variables
- Unlike RNNs, attention is parallel, not sequential.

Masked Attention for Forecasting

- To prevent “peeking into the future,” transformers use **causal masks**:
  - At time t, the model can attend only to time steps ≤ t.
- This is essential for autoregressive forecasting.

Multi-Step Forecasting

- For multi-step predictions $y_{t+1}, \ldots, y_{t+h}$,the Transformer uses:
  - Encoder → processes historical data
  - Decoder → predicts future points sequentially or all at once
- The decoder inputs:
  [StartToken, y_hat(t+1), y_hat(t+2), ...]
- during inference, cross-attention connects:
  - decoder (future steps)
  - encoder (past steps)

Transformer TS properties

- **Advantages:**
  - Capture long-range dependencies
  - Parallel training
  - Flexible handling of covariates
  - Capture temporal + cross-variable relationships
  - State-of-the-art performance on many benchmarks
- **Weaknesses:**
  - Quadratic attention cost
  - Require large datasets
  - Harder to tune than simpler models

Foundation models for TS

- A time series foundation model is a *large pretrained model*—usually Transformer-based—that learns universal temporal patterns from massive collections of time series across many domains.
- Time series foundation models are pretrained on diverse temporal data so they can be adapted quickly to new time series tasks such as:
  - forecasting
  - anomaly detection
  - imputation
  - classification
  - control
  - simulation
- They aim to extract universal temporal dynamics, such as trends, seasonality, periodicity, noise patterns, cross-channel relationships
- This is highly useful because *most time series datasets are small*, but foundation models leverage large-scale pretraining to overcome this.

Why Time Series Need Foundation Models

- Diversity of shapes
  Financial data ≠ weather ≠ healthcare ≠ industrial sensors.
- Long-range dependencies
  Some effects happen months or years after causes.
- Multivariate interactions
  Many time series involve dozens–thousands of correlated variables.
- Missing values, irregular sampling
  Common in sensor / biomedical / transport data.
- Small labeled datasets
  Most forecasting datasets are tiny compared to NLP corpora.
- Foundation models provide generalizable temporal priors learned from large corpora.

Core Ideas of TS Foundation Models

- Large-Scale Pretraining on Many Time Series
- A Universal Sequence Representation
- Fine-Tuning for Downstream Tasks

Large-Scale Pretraining on Many Time Series

- Models are trained on millions of sequences across domains using self-supervised tasks such as:
- Masked time-step prediction: analogous to BERT's masked token modeling:

  x1, x2, [MASK], x4, x5

  predict x3 from context
- Next-step prediction (autoregressive pretraining)

  Teach the model dynamics by predicting x[t] from previous values
- Contrastive learning: learn representations invariant to noise, scaling, augmentation:

  two augmented versions of the same time series → same representation
- Patch-based objectives
  - Split the time series into patches
  - reconstruct missing patches
  - match patch distributions
  - predict temporal relations across patches
- This teaches models universal time-series structure.

A Universal Sequence Representation

- The dominant architecture: Temporal Transformers, which supports:
- Long-range attention captures patterns over hundreds or thousands of time steps.
- Cross-variable attention for multivariate time series, each variable becomes a token or a patch.
- Learnable positional embeddings encode time ordering.
- Several variants:
  - Informer → sparse attention for long sequences
  - Autoformer / FEDformer → frequency-domain decomposition
  - PatchTST → patch-level tokenization
  - TimesNet → periodic convolutions + attention
  - GPT-style models → pure autoregressive time series modeling

Fine-Tuning for Downstream Tasks

- After pretraining, the model is adapted with *small datasets* for tasks like:
- Forecasting

  Add a decoder head:

  model → historical window → predict future horizon
- Imputation

  Predict missing timestamps.
- Anomaly detection

  Score how “unlikely” a segment is under the learned distribution.
- Classification

  Pool the sequence representation and use an MLP head.
- Representation learning

  Use learned embeddings as features for downstream ML.
- Foundation models allow few-shot learning, where even hundreds of labeled samples are enough.

Step by step TS foundation model

1. Pretraining dataset creation; gather millions of time series from diverse sources like finance, energy, IoT sensors, healthcare, climate, retail, industrial telemetry, etc
2. Tokenization; two common strategies:
   - Time-step tokenization: Each time step is a token.
   - Patch tokenization: split time series into fixed-size patches:
     [x1...x16], [x17...x32], ...
     Each patch becomes a token (more stable and scalable).
3. Positional embeddings: make the model aware of time order.
4. Self-attention learns temporal relationships:
5. Self-supervised training: predict masked segments or next steps.
6. Save model as a foundation model and use it for transfer learning

Summary of Foundation Models for TS

- Performance boost on small datasets
  Few-shot forecasting becomes feasible.
- Capture global temporal patterns
  Trends, cycles, periodic behavior across domains.
- Rich temporal representations
  Useful for anomaly detection, classification, forecasting simultaneously.
- Scalability
  Transformers + patches allow long-context modeling.
- Multivariate compatibility
  Models learn cross-channel dependencies automatically.

Several pretrained models: PatchTST, GPT4TS, Chronos

University of Ljubljana, Faculty of Computer and Information Science

Automated Machine Learning (AutoML)



Prof Dr Marko Robnik-Šikonja

Intelligent Systems, Edition 2025

Why AutoML?

- Growing demand for ML expertise
- Manual ML pipelines are time-consuming
- Evolution: Manual ML → Automated ML
- End-to-end automation of ML pipeline
- AutoML democratizes AI
- Use cases in competitions, industry, etc.
- Not a magic

Traditional ML Pipeline

- Steps:
  data→preprocessing→model→evaluation→deployment
- AutoML automates most parts
- Instead of manually choosing:
  - features
  - models
  - hyperparameters
  - training strategies

AutoML systems automatically search, optimize, and adapt.

Automated Preprocessing

- Missing data handling
- Scaling
- Feature construction

Feature engineering automation

- Automatically generate, transform, or select features
- Automate:
  - normalization & scaling
  - categorical encoding
  - feature crosses
  - polynomial features
  - feature selection
- Approaches
  - rule-based pipelines
  - evolutionary algorithms
  - attention-based feature learning
  - embedded methods (e.g., L1 regularization)
- Example: create log(income), income/age, select top-k features

Model selection

- Automatically choose *which algorithm* to use.
- Search space includes: linear models, tree-based models, kernel methods, neural networks, etc.
- Techniques
  - Bayesian optimization over models
  - ensemble selection
  - meta-learning (use prior tasks to guide choice)
- Example: Should I use Random Forest, XGBoost, or a Neural Network?

Meta-Learning (learning to learn)

- Use experience from previous datasets to speed up AutoML.
- Store dataset “meta-features” (size, skewness, entropy, etc.)
- Learn which models worked best before
- warm-start optimization
- Example: if the dataset looks like past credit-risk data → try XGBoost first
- This reduces search time drastically.

Neural architecture search (NAS)

- Automatically design neural network architectures.
- Search space:
  - number of layers
  - width
  - skip connections
  - attention vs convolution
- Search strategies
  - reinforcement learning
  - evolutionary algorithms
  - gradient-based NAS (e.g., DARTS)
- Example: find the best CNN or Transformer architecture.

Hyperparameter optimization

- Automatically finds good hyperparameters for a fixed model
- This is the most common and mature AutoML technique.
- Typical parameters: learning rate, number of layers, tree depth, regularization strength, etc
- Methods
  - Grid search (simple, inefficient)
  - Random search (surprisingly strong baseline)
  - Bayesian optimization (Gaussian Processes, TPE)
- Example: Find best (learning_rate, max_depth) for XGBoost

Random search

- Random search optimizes hyperparameters by sampling configurations randomly,
- This is more efficient than grid search in high-dimensional spaces because it focuses effort on important parameters naturally.
- Assuming that not all parameters are equally important, grid search only samples a few values of important parameters, while random search samples many more distinct values of important parameters with the same computational budget
- Grid search:
  - assume thet you use a grid with n values per parameter
  - Total $trials = n^d$
  - Each parameter gets exactly n values, regardless of importance.
- Random search
  - Let k= number of important parameters, N= total trials
  - Random search explores $\mathcal{O}(N)$values per important parameter
- Grid search explores $\mathcal{O}\left(N^{1/k}\right)$values per important parameter
- For large d, this gap is enormous.

Pipeline Optimization (End-to-End AutoML)

- Optimize entire ML pipelines, not just components
- A pipeline might be: StandardScaler → PCA → XGBoost
- AutoML searches over:
  - preprocessing steps
  - feature selection
  - model choice
  - hyperparameters
- Key methods
  - Bayesian optimization
  - genetic programming
  - hierarchical search spaces
- Example systems
  - Auto-sklearn
  - TP ’s AutoML
  - H2O AutoML

Bayesian optimization

- Bayesian Optimization is a strategy for optimizing expensive, unknown, and non-convex functions using as few evaluations as possible.
- It is typically used when evaluating the objective is slow or costly (e.g., training a model), gradients are unavailable, the function may be noisy, and/or the number of evaluations is limited.
- In ML, the function is often:
  $f(\theta) = \textit{validation performance of model with hyperparameters } \theta$
- Bayesian optimization builds a surrogate probabilistic model of the objective function and uses it to decide where to evaluate next, balancing exploration and exploitation

Strengths and limitations of AutoML

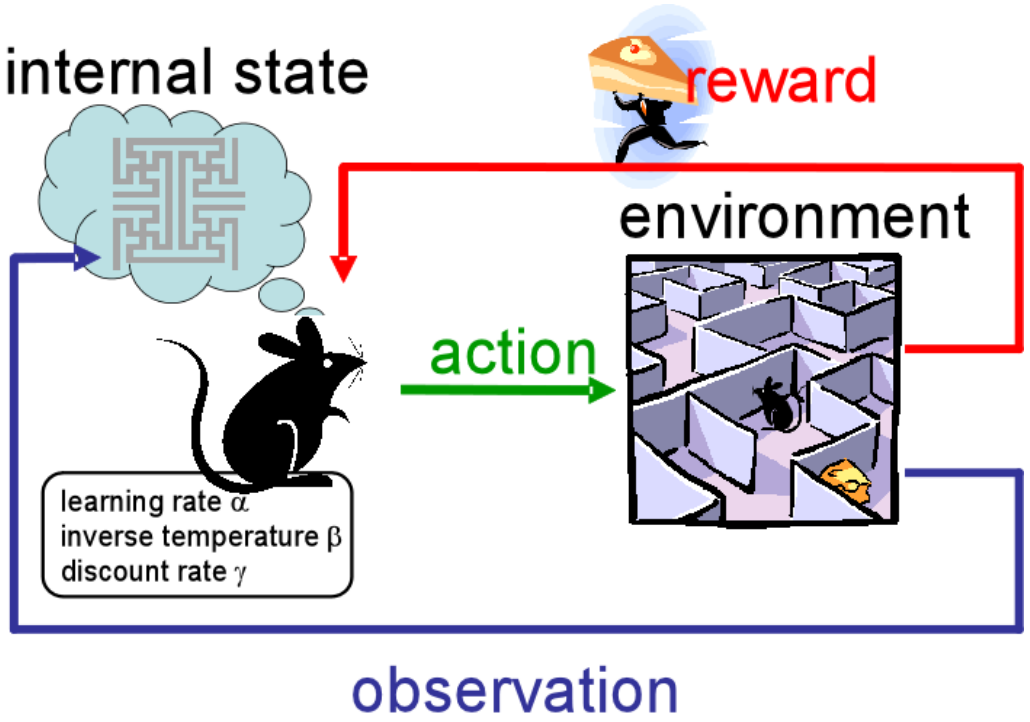
+ Saves time and expertise
+ Strong baselines quickly
+ Reduces human bias
+ Reproducible workflows

- Computationally expensive
- Limited interpretability
- Hard to encode domain knowledge
- May overfit if not controlled
- High compute cost

University of Ljubljana, Faculty of Computer and Information Science

Reinforcement learning





Prof Dr Marko Robnik-Šikonja
Intelligent Systems, Edition 2024

References

- R. S. Sutton and A. G. Barto: Reinforcement Learning: An Introduction, 2018, 2nd edition (the book is freely available)
- many papers, tutorials, online courses
- recently a revival due to deep reinforcement learning

some slides are courtesy of Andrew Barto, Peter Bodik and Lisa Torrey

Machine Learning

- Classification:
  - Given
    - Training data
  - Learn
    - A model for making a single prediction or decision



Animal/Human Learning

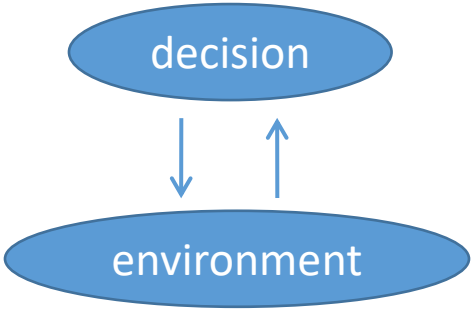
Memorization



Classification



Procedural

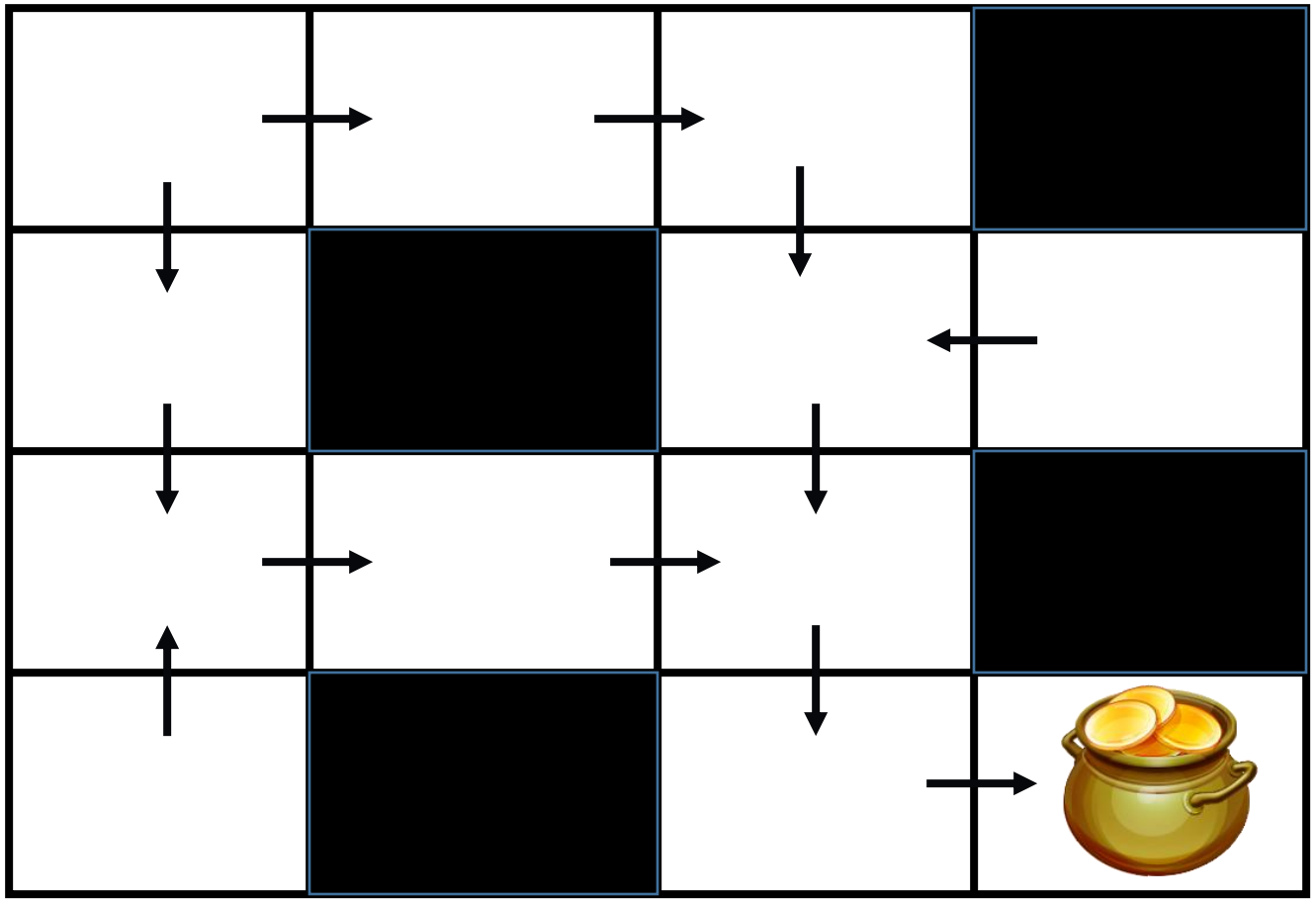


Other?

Procedural Learning

- Learning how to act to accomplish goals
  - Given
    - Environment that contains rewards
  - Learn
    - A policy for acting
- Important differences from classification
  - You don’t get examples of correct answers
  - You have to try things in order to learn

A Good Policy

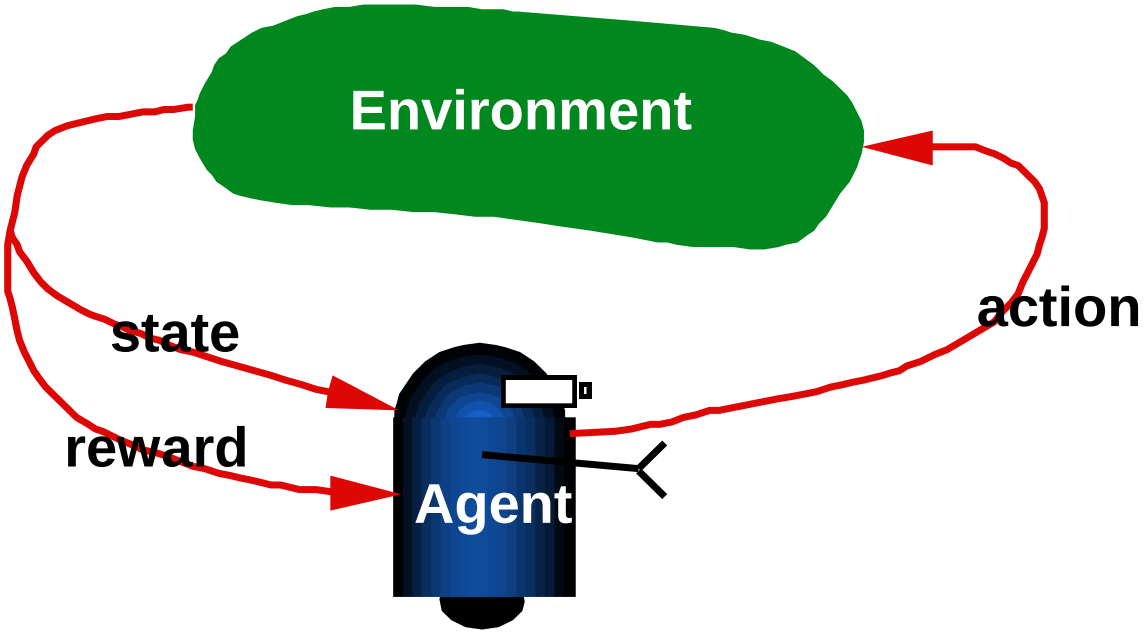


Introduction to Reinforcement Learning

- Reinforcement learning (RL), questionable terminology stemming from behavioristic psychology (behavior reinforcement)
- Agent learning in the environment, performing actions
- Getting feedback from the environment (award, punishment), not necessary immediately
- Trying to learn a policy leading to goals
- An example: playing a game without knowing the rules; after 1000 moves an opponent declares: you lost.

Agent

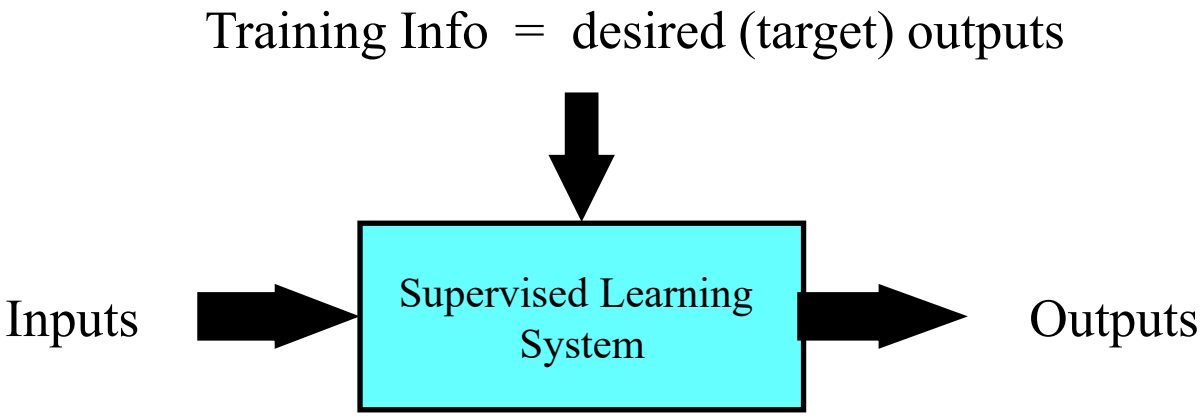
- Temporally situated
- Continual learning and planning
- Objective is to affect the environment
- Environment is stochastic and uncertain



What is Reinforcement Learning?

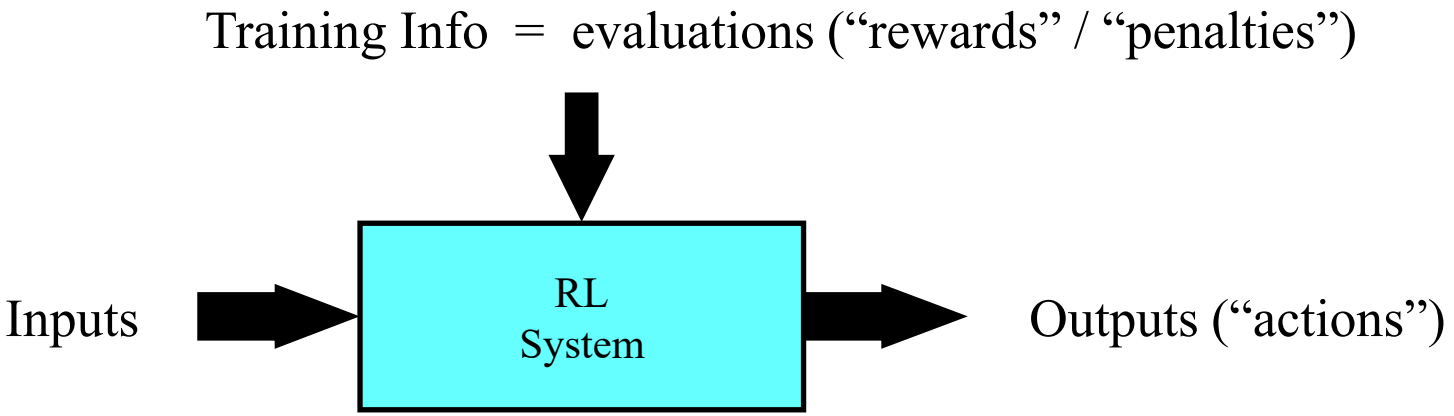
- Learning from interaction
- Goal-oriented learning: short term and possible long term awards
- Learning about, from, and while interacting with an external environment
- Learning what to do—how to map situations to actions—so as to maximize a numerical reward signal
- Agent discovers which action in what circumstances give the highest award
- Agent can build a model of its environment
- RL is not supervised learning, it is about trial and error search, exploring, getting information from environment

Supervised Learning



Error = (target output – actual output)

Reinforcement Learning



Objective: get as much reward as possible

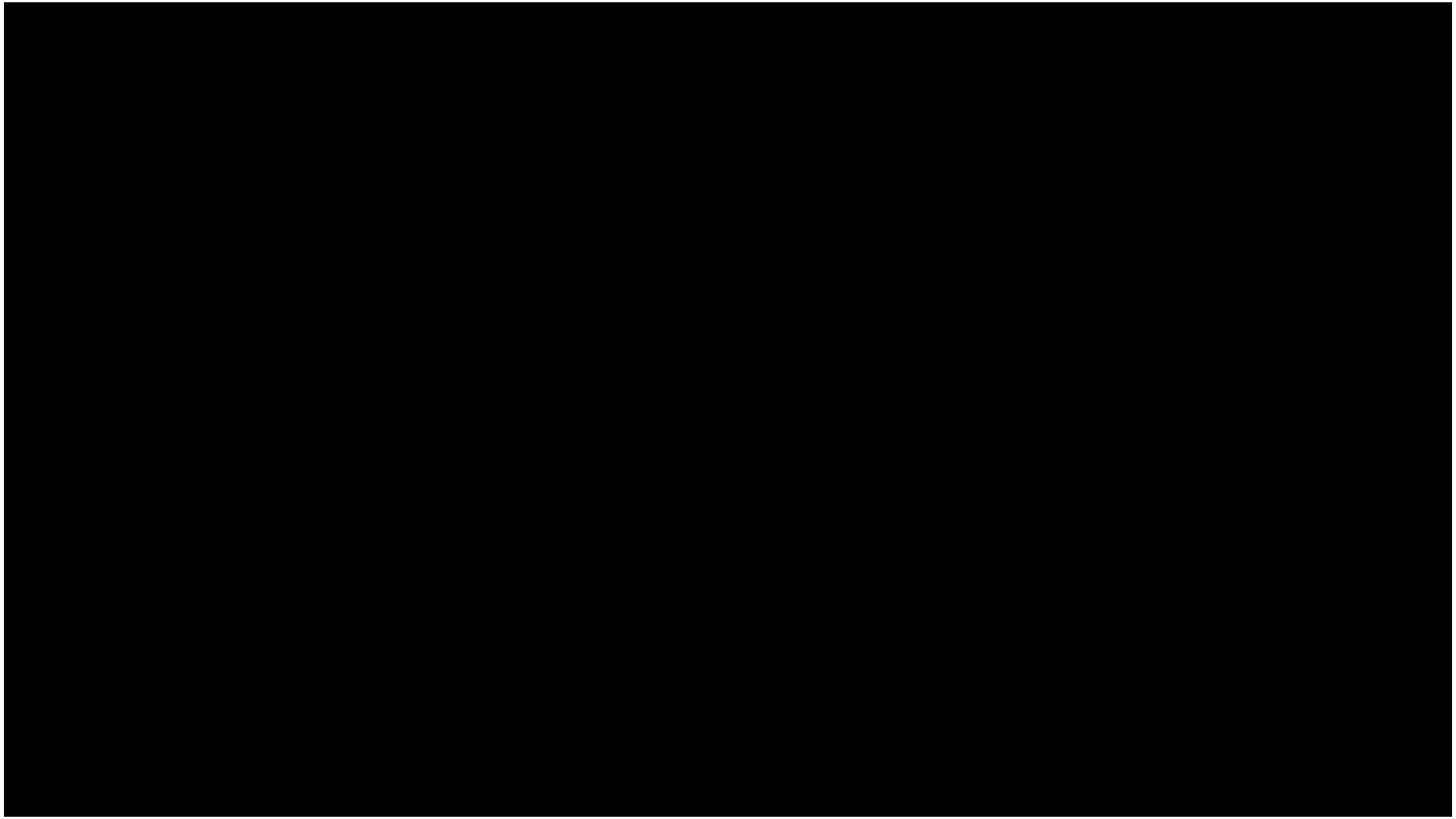
Key Features of RL

- Learner is not told which actions to take
- Trial-and-error search
- Possibility of delayed reward
  - Sacrifice short-term gains for greater long-term gains
- The need to ***explore*** and ***exploit***
- Considers the whole problem of a goal-directed agent interacting with an uncertain environment

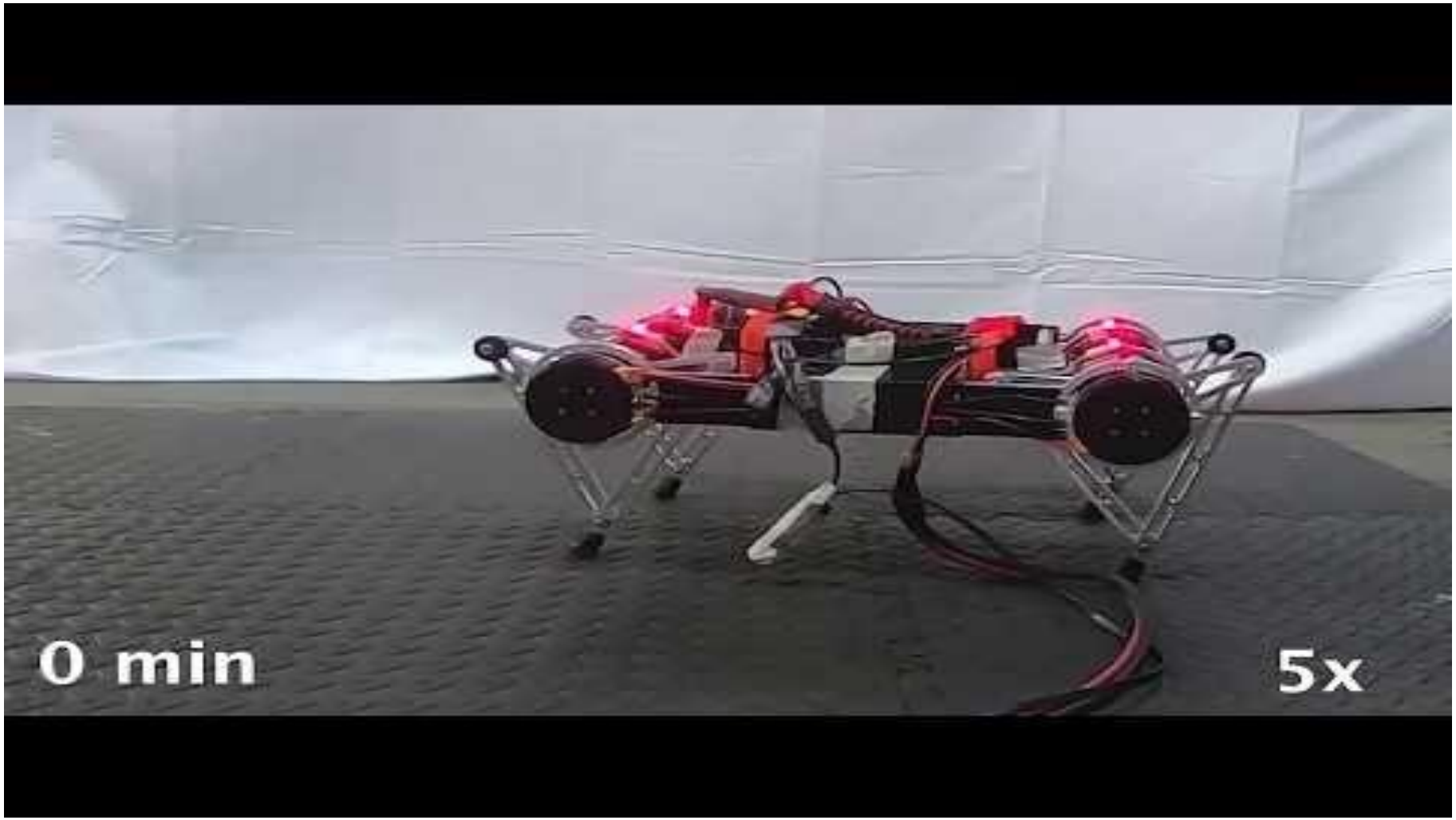
RL successful applications

- Robocup Soccer
- Financial asset management/Inventory management
- Dynamic Channel assignment in mobile communications
- Controlling elevators, industrial controllers, robots …
- Robots: navigation, grasping, moving …
- Games: backgammon (TD-Gammon, Jellyfish), Go (AlphaGo in combination with deep neural networks), Atari video games, poker, chess
- in LLMs used in RLHF

Example video: Atari game Breakout

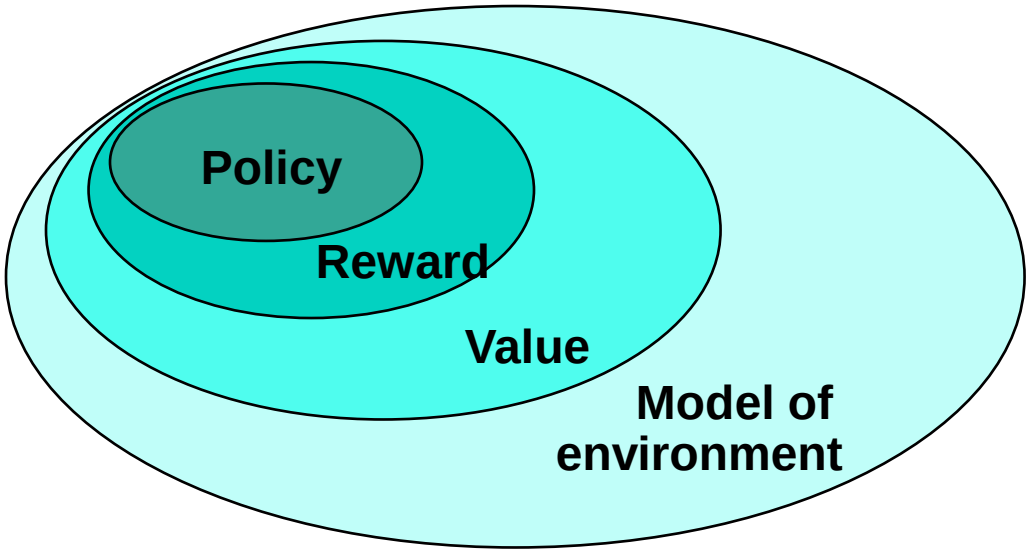


Example video:Robot training



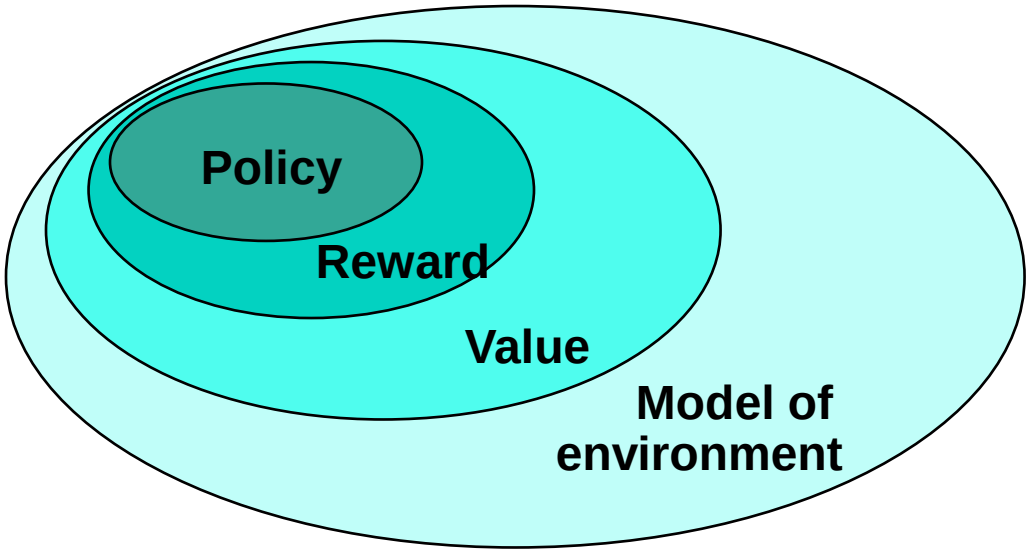
Tuomas Haarnoja, Sehoon Ha, Aurick Zhou, Jie Tan, George Tucker, Sergey Levine. Learning to Walk via Deep Reinforcement Learning. Robotics: Science and Systems (RSS). 2019.

Components of RL 1/2



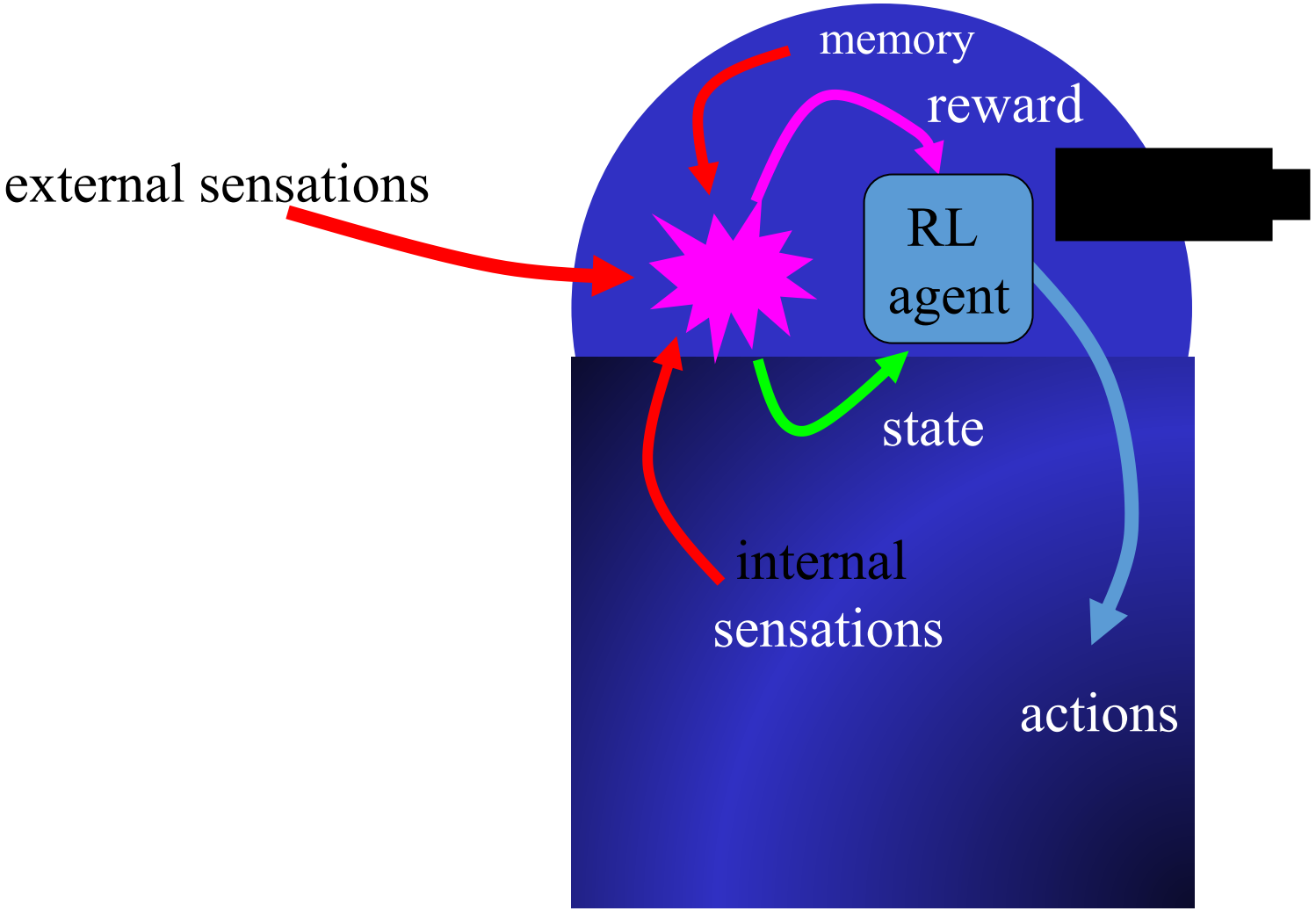
- **Policy**: what to do?
  - Defines agents choices and actions in a given time
  - Represented with rules, table, neural networks etc.
  - Result of search, planning, stochastic, etc.
- **Reward:** what is good?
  - Feedback from environment, agent tries to maximize it

Components of RL 2/2

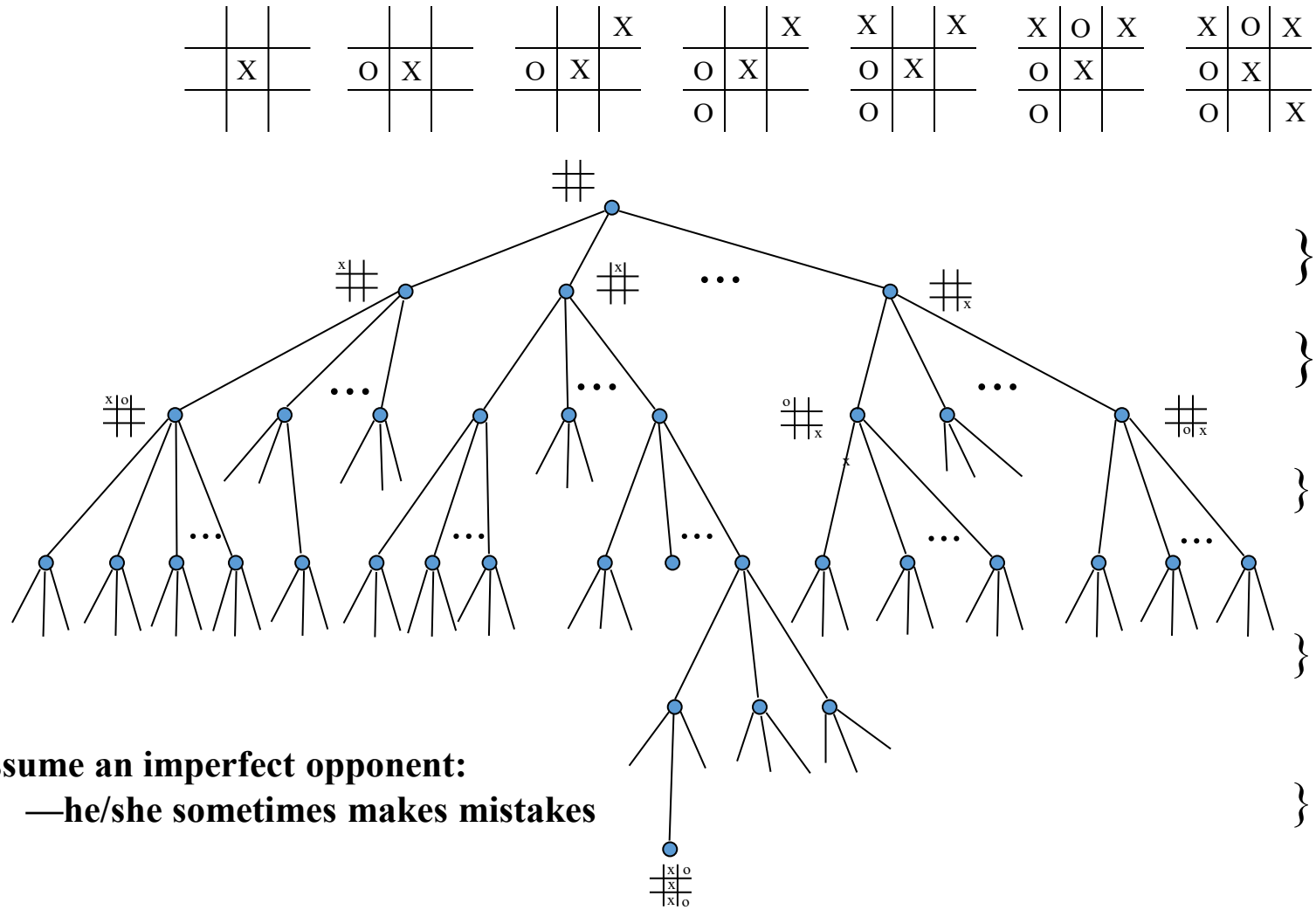


- **Value**: internal representation of what is good, it *predicts* reward
  - Agent's expectation of what can be expected in given state (long-term)
  - Implicitly contains evaluation of next states
  - Value has to be learned; use repetitions and sampling to estimate the value
- **Model**: what follows what
  - Internal representation of the environment
  - Agent can evaluate values and actions without performing them
  - Optional component

Agent from the point of view of RL



An Example: Tic-Tac-Toe



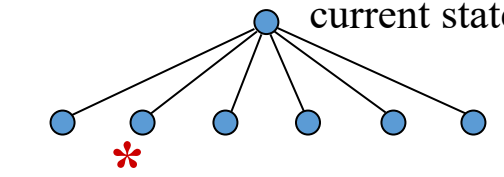
Assume an imperfect opponent:
—he/she sometimes makes mistakes

An RL Approach to Tic-Tac-Toe

1. Make a table with one entry per state:

State	$V(s)$ – estimated probability of winning	
	.5	?
	.5	?
⋮	⋮	
	1	win
⋮	⋮	
	0	loss
⋮	⋮	
	0	draw

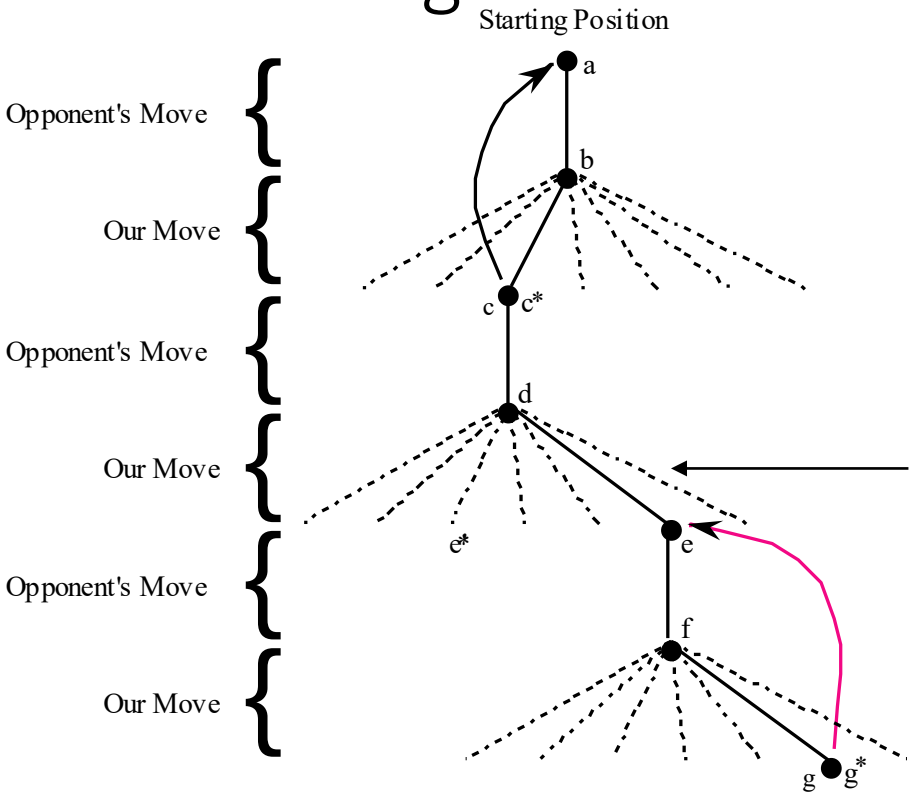
2. Now play lots of games.
To pick our moves,
look ahead one step:



Just pick the next state with the highest estimated prob. of winning — the largest $V(s)$; a ***greedy*** move.

But 10% of the time pick a move at random; an ***exploratory move***.

RL Learning Rule for Tic-Tac-Toe



"Exploratory" move

s – the state before our greedy move

s' – the state after our greedy move

We increment each $V(s)$ toward $V(s')$ – a ***backup***:

$$V(s) \leftarrow V(s) + \alpha\left[V(s') - V(s)\right]$$

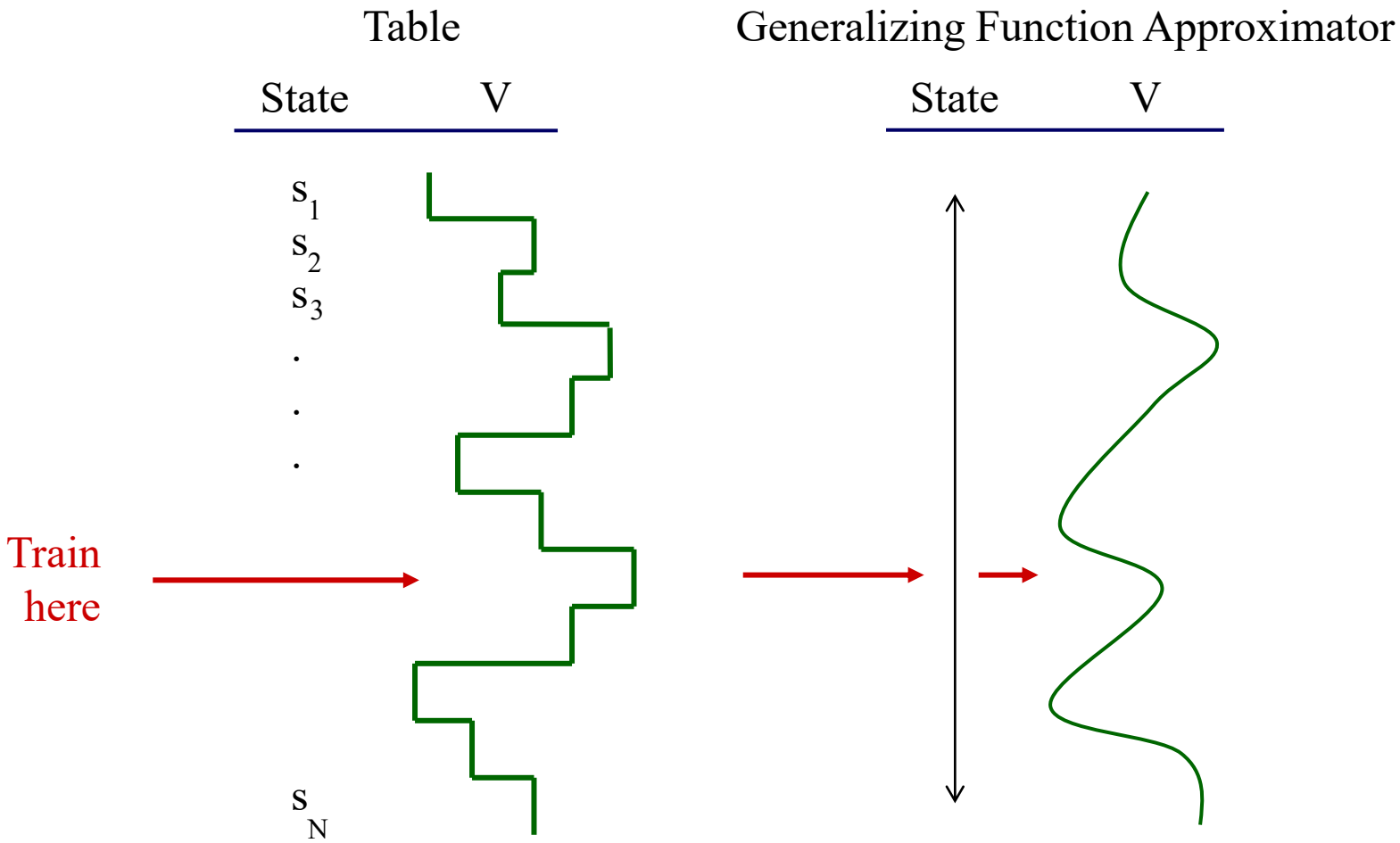
a small positive fraction, e.g., $\alpha = .1$

the ***step-size parameter***

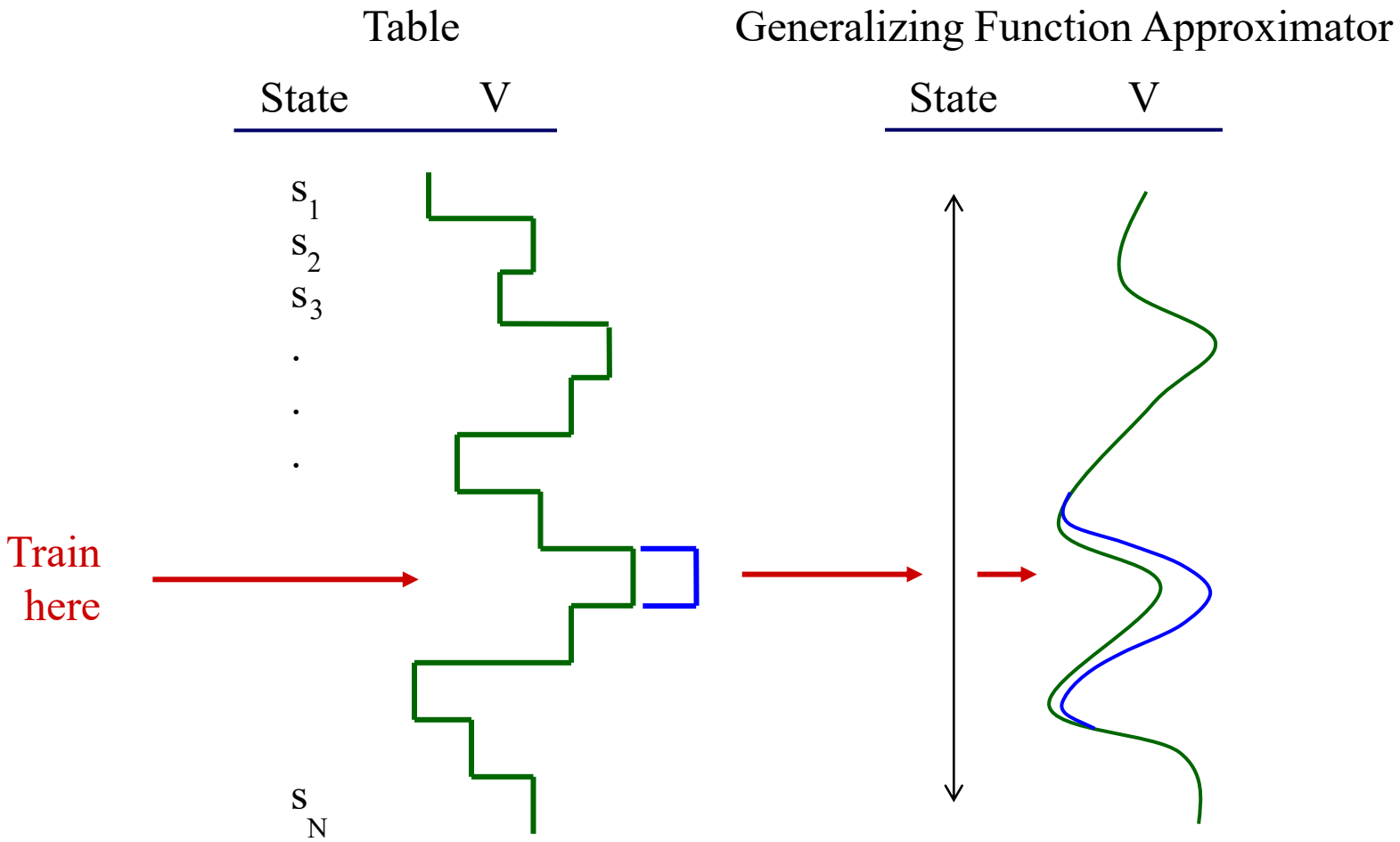
How can we improve this TTT player?

- Take advantage of symmetries
  - representation/generalization
  - How might this backfire?
- Do we need “random” moves? Why?
  - Do we always need a full 10%?
- Can we learn from “random” moves?
- Can we learn offline?
  - Pre-training from self play?
  - Using learned models of opponent?
- . . .

E.g., generalization



E.g., generalization

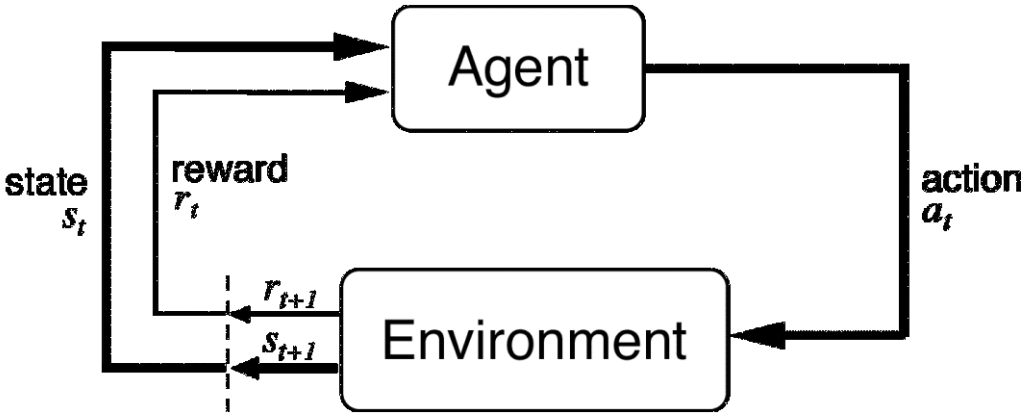


Tic-Tac-Toe is just a toy example

- Finite, small number of states
- One-step look-ahead is always possible
- State completely observable
- . . .

- RL is not limited to a finite number of states; in problems with infinite or very large number of states we only generate states encountered during search
- RL is not limited to games or opponent's response

The Agent-Environment Interface



Agent and environment interact at discrete time steps: $t = 0, 1, 2, \ldots$

Agent observes state at step t: $s_t \in S$

produces action at step t: $a_t \in A(s_t)$

gets resulting reward: $r_{t+1} \in \Re$

and resulting next state: s_{t+1}

$$\ldots \; s_t \xrightarrow[a_t]{r_{t+1}} s_{t+1} \xrightarrow[a_{t+1}]{r_{t+2}} s_{t+2} \xrightarrow[a_{t+2}]{r_{t+3}} s_{t+3} \xrightarrow[a_{t+3}]{} \ldots$$

The Agent Learns a Policy

$$\textbf{Policy} \text{ at step } t, \ \pi_t :$$

$$\text{a mapping from states to action probabilities}$$

$$\pi_t(s,a) = \text{ probability that } a_t = a \text{ when } s_t = s$$

- Reinforcement learning methods specify how the agent changes its policy as a result of experience.
- Roughly, the agent's goal is to get as much reward as it can over the long run.

Getting the degree of abstraction right

- Time steps need not refer to fixed intervals of real time.
- Actions can be low level (e.g., voltages to motors), or high level (e.g., accept a job offer), “mental” (e.g., shift in focus of attention),
- States can be low-level “sensations”, or they can be abstract, symbolic, based on memory, or subjective (e.g., the state of being “surprised” or “lost”).
- An RL agent is not like a *whole* animal or robot.
- Reward computation is in the agent’s environment because the agent cannot change it arbitrarily.
- The environment is not necessarily unknown to the agent, only incompletely controllable.

Goals and Rewards

- Is a scalar reward signal an adequate notion of a goal?—maybe not, but it is surprisingly flexible.
- A goal should specify **what** we want to achieve, not **how** we want to achieve it.
- A goal must be outside the agent's direct control—thus outside the agent.
- The agent must be able to measure success:
  - explicitly;
  - frequently during its lifespan.

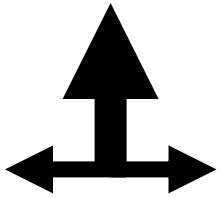
Robot in a room

			+1
			-1
START			

actions: UP, DOWN, LEFT, RIGHT

UP

80% move UP
10% move LEFT
10% move RIGHT

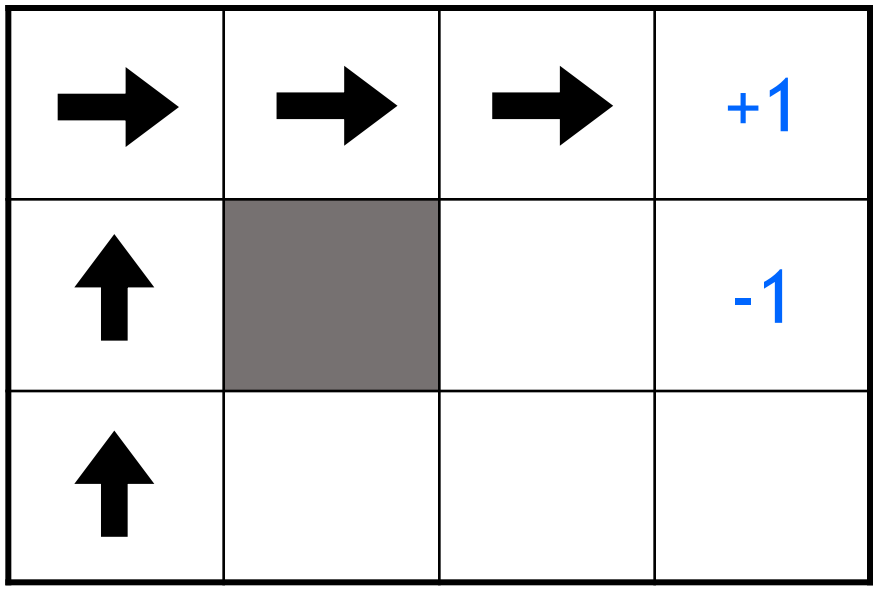


reward +1 at [4,3], -1 at [4,2]
reward -0.04 for each step

- states
- actions
- rewards

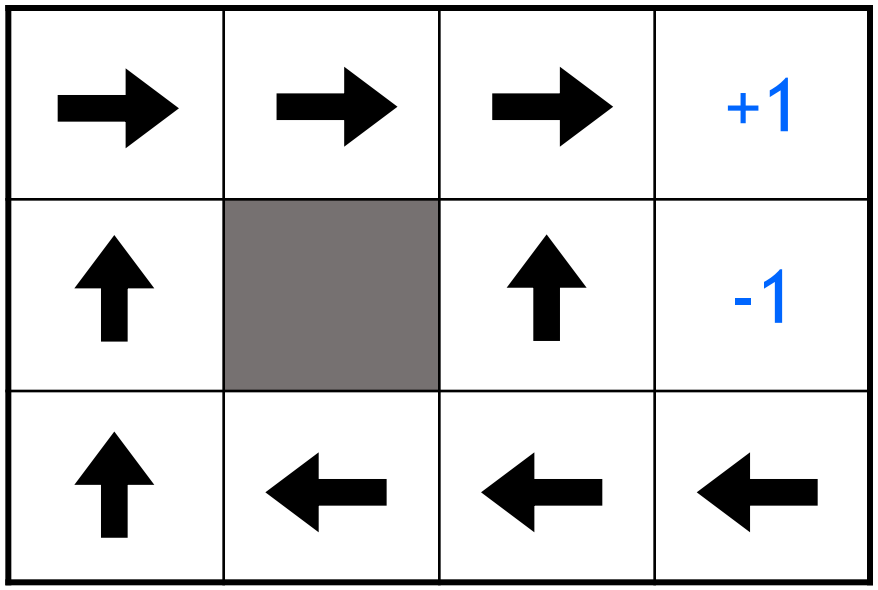
- what is the solution?

Is this a solution?

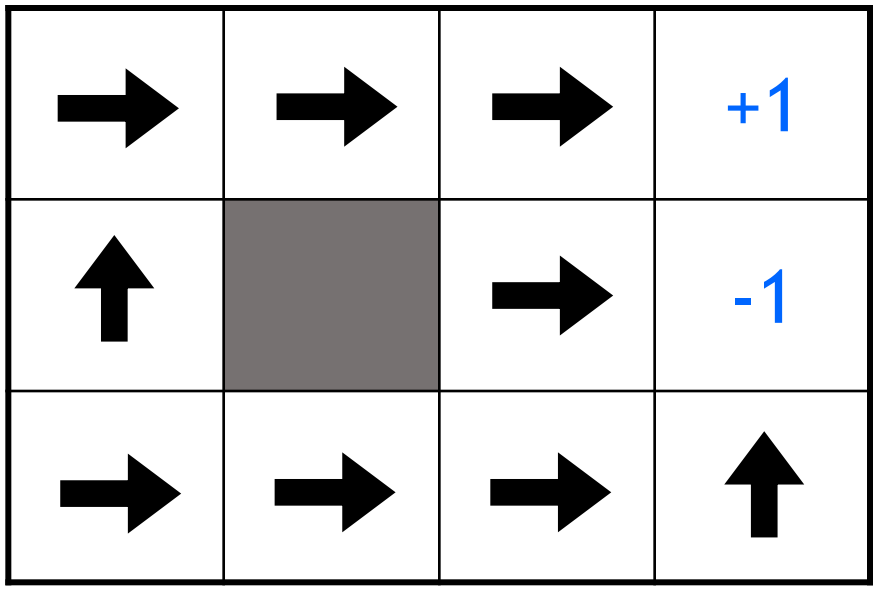


- only if actions deterministic
  - not in this case (actions are stochastic)
- solution/policy
  - mapping from each state to an action

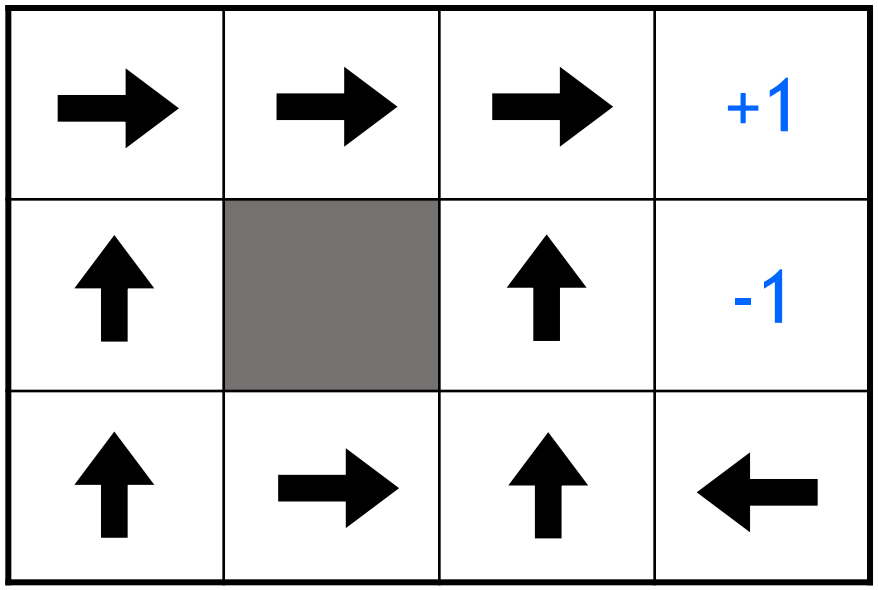
Optimal policy



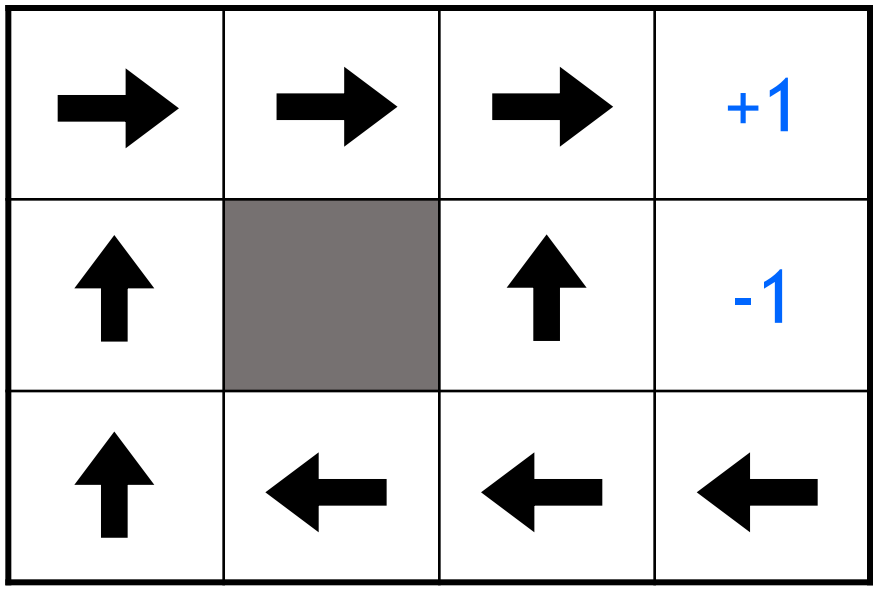
Reward for each step -2



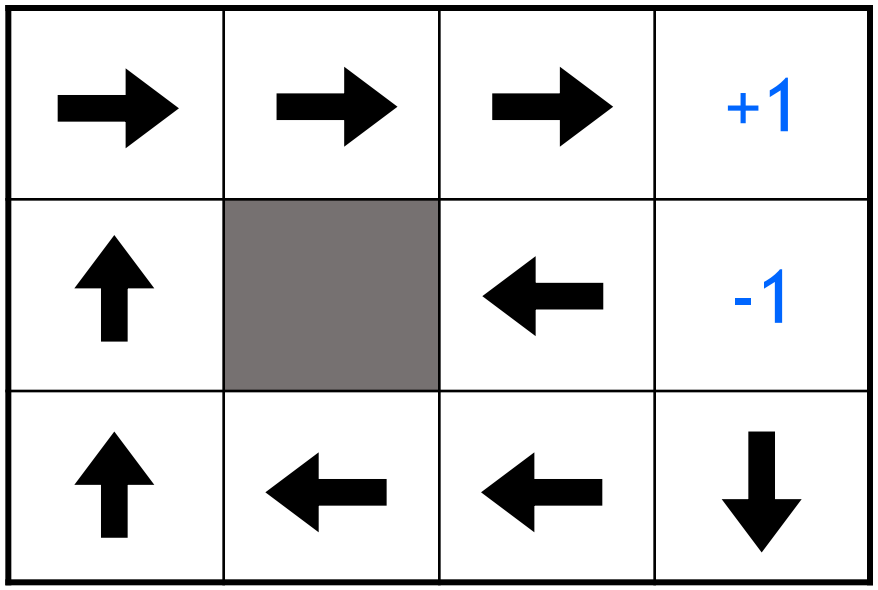
Reward for each step: -0.1



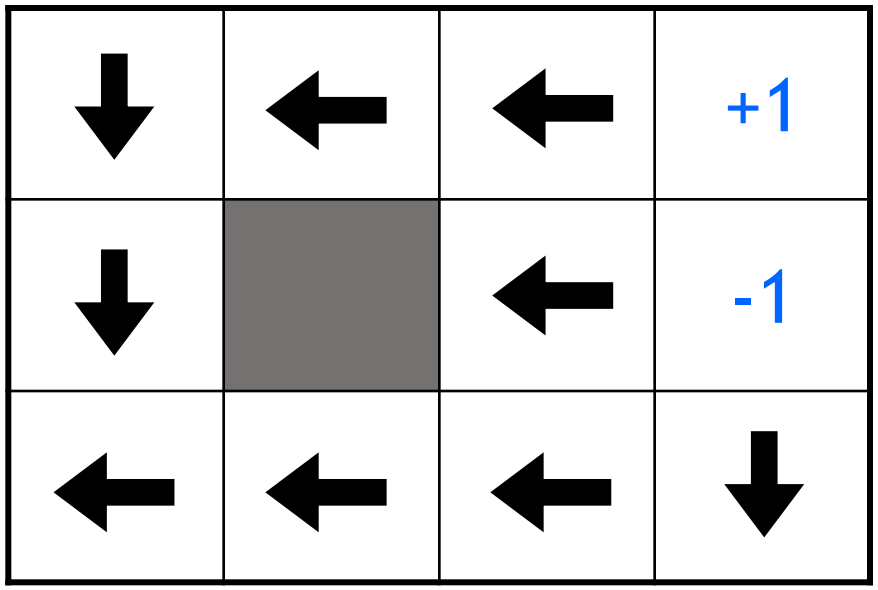
Reward for each step: -0.04



Reward for each step: -0.01



Reward for each step: +0.01



Returns

Suppose the sequence of rewards after step t is :

$$r_{t+1}, r_{t+2}, r_{t+3}, \cdots$$

What do we want to maximize?

In general,

we want to maximize the **expected return**, $E\{R_t\}$, for each step t.

Episodic tasks: interaction breaks naturally into episodes, e.g., plays of a game, trips through a maze.

$$R_t = r_{t+1} + r_{t+2} + \cdots + r_T,$$

where T is a final time step at which a **terminal state** is reached, ending an episode.

Returns for Continuing Tasks

Continuing tasks: interaction does not have natural episodes.

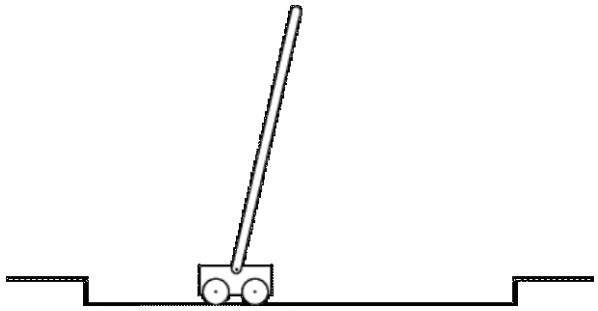
Discounted return:

$$R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1},$$

where $\gamma, 0 \le \gamma \le 1,$ is the **discount rate**.

$$\text{shortsighted} \quad 0 \leftarrow \gamma \rightarrow 1 \quad \text{farsighted}$$

An example: cart and pole



Avoid **failure:** the pole falling beyond a critical angle or the cart hitting end of track.

As an **episodic task** where episode ends upon failure:

$$\text{reward} = +1 \text{ for each step before failure}$$

$$\Rightarrow \text{ return} = \text{number of steps before failure}$$

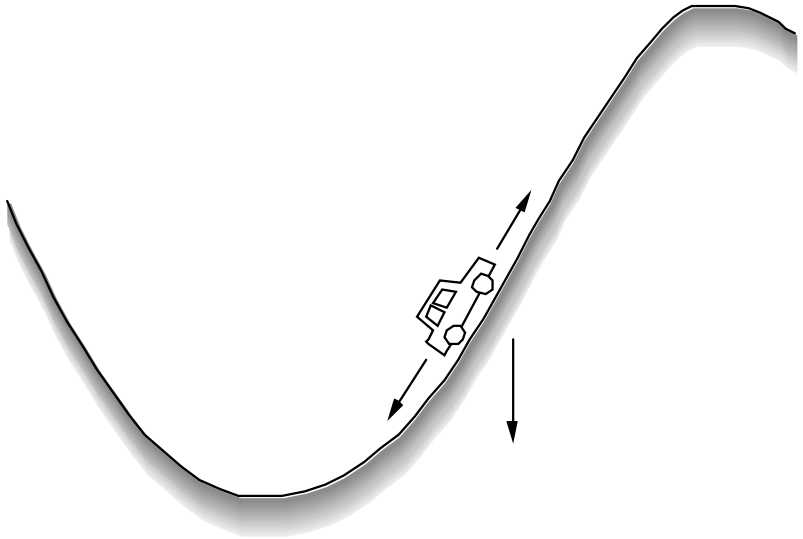
As a **continuing task** with discounted return:

$$\text{reward} = -1 \text{ upon failure; 0 otherwise}$$

$$\Rightarrow \text{ return} = -\gamma^k, \text{ for } k \text{ steps before failure}$$

In either case, return is maximized by avoiding failure for as long as possible.

Another Example



Get to the top of the hill
as quickly as possible.

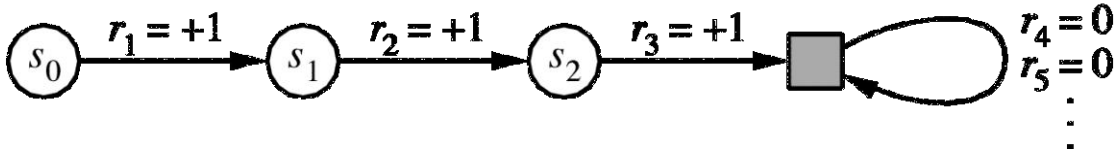
reward $= -1$ for each step where **not** at top of hill

$\Rightarrow$ return $=$ $-$ number of steps before reaching top of hill

Return is maximized by minimizing
number of steps to reach the top of the hill.

A Unified Notation

- In episodic tasks, we number the time steps of each episode starting from zero.
- We usually do not have to distinguish between episodes, so we write s_t instead of $s_{t,j}$ for the state at step *t* of episode *j*.
- Think of each episode as ending in an absorbing state that always produces reward of zero:



- We can cover all cases by writing

$$R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1},$$

where γ can be 1 only if a zero reward absorbing state is always reached.

Episodic task – finite horizon

- In time *t,* the agent is interested in *h* further states
- Rewards in that time are r_{t+1}, r_{t+2}, r_{t+3}, …, r_{t+h}

  $R_t = r_{t+1} + r_{t+2} + r_{t+3} + \ldots + r_{t+h}$
- The agent maximizes expected reward in that period

$$\max E(R_t) = \max E(\sum_{k=1}^{h} r_{t+k})$$

Finite horizon

- Two optimal behaviors
  - h-step optimal action: on step 1, do an action which is optimal under assumption that h-1 actions will follow, on step 2 do an action which is optimal under assumption that h-2 actions will follow ...
  - h-step receding action: on each step do an action which is optimal under assumption that h actions will follow
- Limited look-ahead
- Suitability of finite horizon: episodic missions (e.g., labyrinth)

Continuous tasks

- No natural end, but …

  … nearer actions are more important than more distant ones

- Agent optimizes infinite sequence of rewards
- Rewards are geometrically discounted
- rewards: $R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \gamma^3 r_{t+4,}$ ... for $0 < \gamma < 1$
- γ (discount factor) can be interpreted as interest rate, a trick to bound an infinite sum, probability of surviving another step, short/far-sightedness

$$\max E(\sum_{k=0}^{\infty} \gamma^k r_{t+k+1}), \quad 0 < \gamma < 1$$

Average reward model

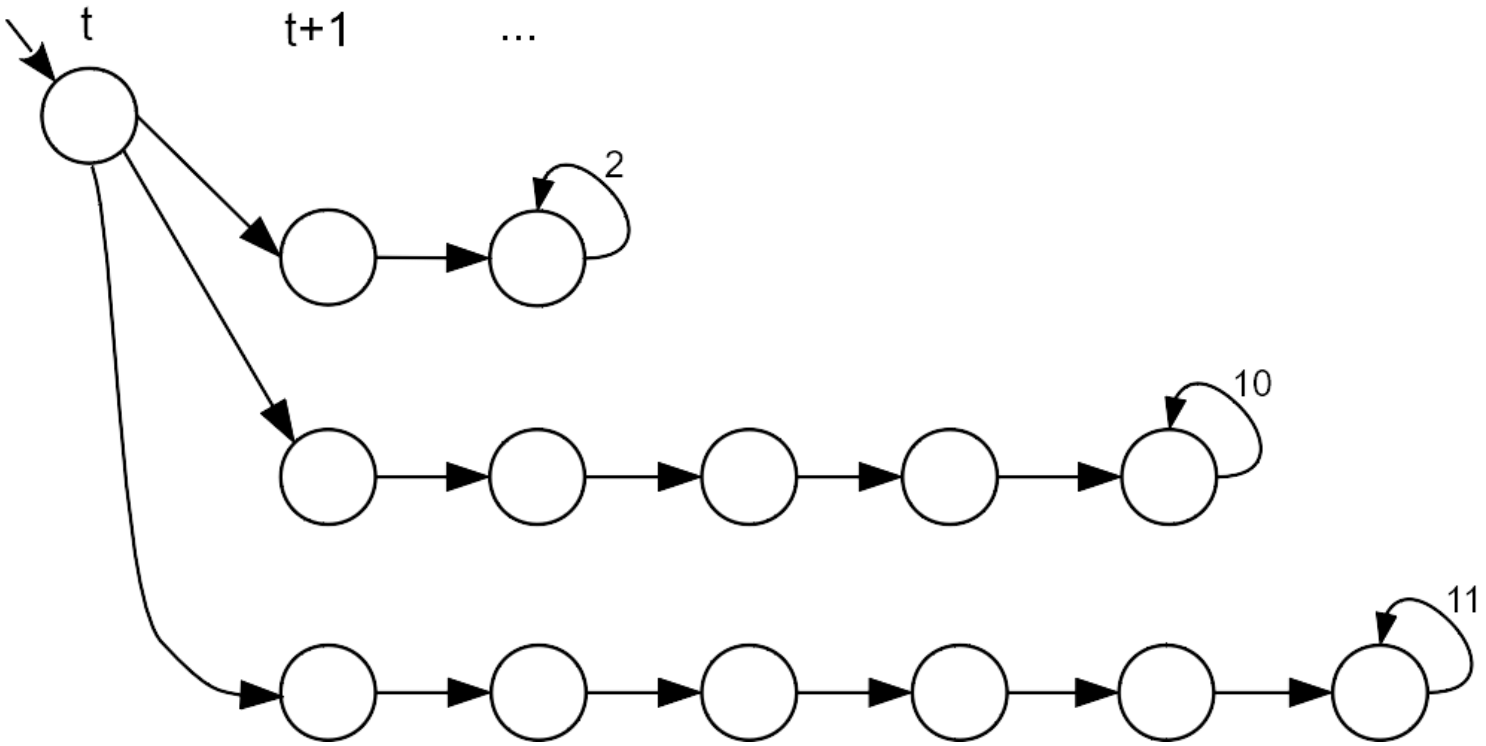
- Agent optimizes long-term average reward

$$\lim_{h \to \infty} E\left(\frac{1}{h} \sum_{k=1}^{h} r_{t+k}\right)$$

- Downside: does not know the difference between near and distant rewards

An example: rewards

1. finite horizont, h=4
2. infinite horizont, γ=0.9
3. average expected reward



The Markov Property

- By "the state" at step *t*, we mean whatever information is available to the agent at step *t* about its environment.
- The state can include immediate "sensations," highly processed sensations, and structures built up over time from sequences of sensations.
- Ideally, a state should summarize past sensations so as to retain all "essential" information, i.e. it should have the **Markov Property**:

$$\Pr\{s_{t+1}=s', r_{t+1}=r \mid s_t, a_t, r_t, s_{t-1}, a_{t-1}, \ldots, r_1, s_0, a_0\} =$$

$$\Pr\{s_{t+1}=s', r_{t+1}=r \mid s_t, a_t\}$$

$$\text{for all } s', r, \text{ and histories } s_t, a_t, r_t, s_{t-1}, a_{t-1}, \ldots, r_1, s_0, a_0.$$

Markov Decision Processes

- If a reinforcement learning task has the Markov Property, it is basically a **Markov Decision Process (MDP)**.
- If state and action sets are finite, it is a **finite MDP**.
- To define a finite MDP, you need to give:
  - **state and action sets**
  - one-step "dynamics" defined by **transition probabilities**:

$$\mathbf{P}^{a}_{ss'} = \Pr\left\{s_{t+1} = s' \,\middle|\, s_t = s, a_t = a\right\} \quad \text{for all } s, s' \in S,\ a \in A(s).$$

  - **reward probabilities**:

$$\mathbf{R}^{a}_{ss'} = E\left\{r_{t+1} \,\middle|\, s_t = s, a_t = a, s_{t+1} = s'\right\} \quad \text{for all } s, s' \in S,\ a \in A(s).$$

An Example of Finite MDP

Recycling Robot

- At each step, robot has to decide whether it should (1) actively search for a can, (2) wait for someone to bring it a can, or (3) go to home base and recharge.
- Searching is better but runs down the battery; if runs out of power while searching, has to be rescued (which is bad).
- Decisions made on basis of current energy level: `high`, `low`.
- Reward = number of cans collected

Recycling Robot MDP

$S = \{\texttt{high}, \texttt{low}\}$

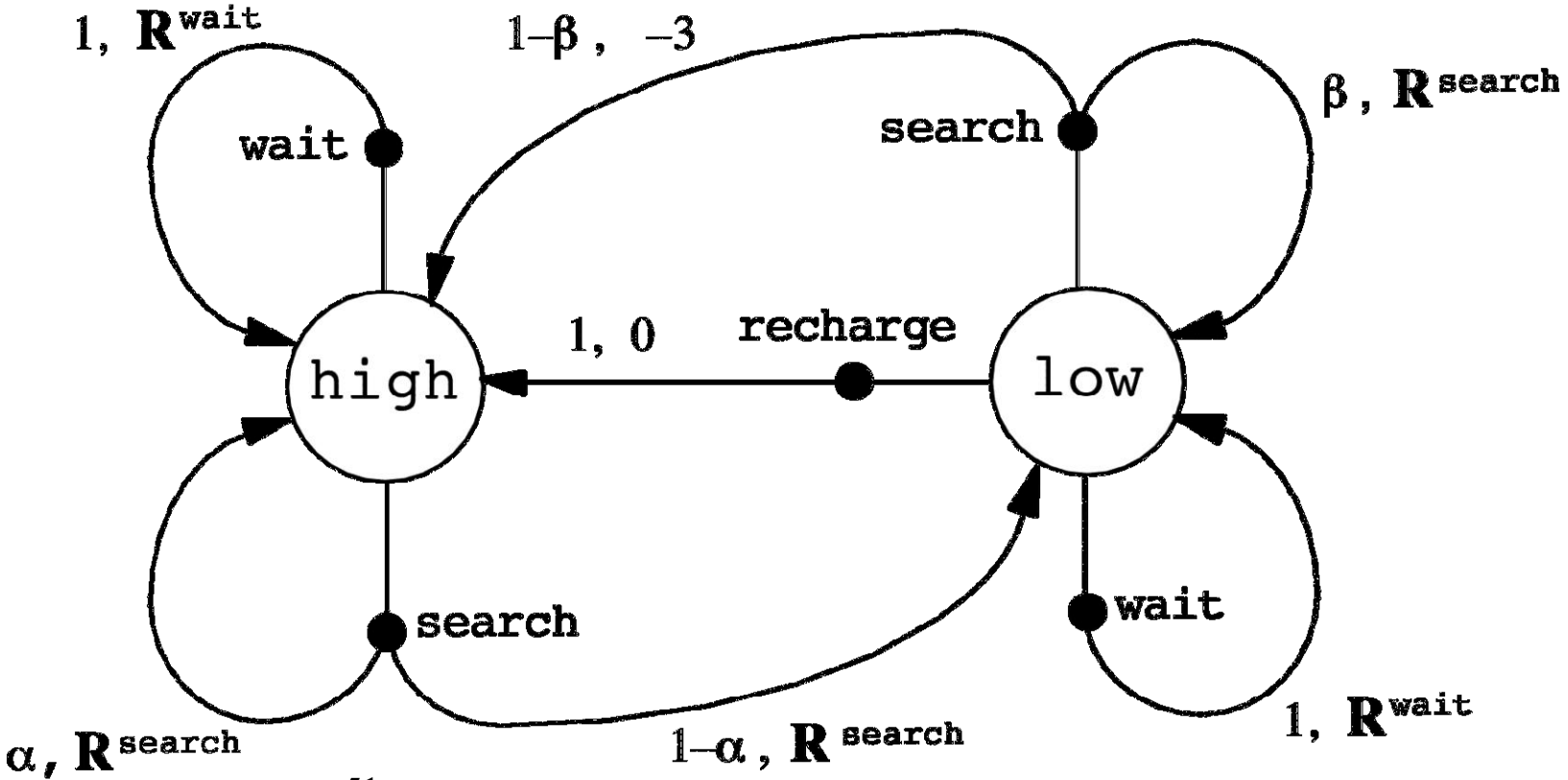
$A(\texttt{high}) = \{\texttt{search}, \texttt{wait}\}$

$A(\texttt{low}) = \{\texttt{search}, \texttt{wait}, \texttt{recharge}\}$

$\mathbf{R}^{\texttt{search}}$ = expected no. of cans while searching

$\mathbf{R}^{\texttt{wait}}$ = expected no. of cans while waiting

$\mathbf{R}^{\texttt{search}} > \mathbf{R}^{\texttt{wait}}$



Value Functions

- The **value of a state** is the expected return starting from that state; depends on the agent's policy:

State-value function for policy π:

$$V^{\pi}(s) = E_{\pi}\{R_t \mid s_t = s\} = E_{\pi}\left\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid s_t = s\right\}$$

- The **value of taking an action in a state under policy π** is the expected return starting from that state, taking that action, and thereafter following π:

Action-value function for policy π:

$$Q^{\pi}(s,a) = E_{\pi}\{R_t \mid s_t = s, a_t = a\} = E_{\pi}\left\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid s_t = s, a_t = a\right\}$$

Bellman Equation for policy π

The basic idea:

$$\begin{aligned} R_t &= r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \gamma^3 r_{t+4} \cdots \\ &= r_{t+1} + \gamma \left(r_{t+2} + \gamma r_{t+3} + \gamma^2 r_{t+4} \cdots \right) \\ &= r_{t+1} + \gamma R_{t+1} \end{aligned}$$

So:

$$\begin{aligned} V^{\pi}(s) &= E_{\pi}\left\{ R_t \,|\, s_t = s \right\} \\ &= E_{\pi}\left\{ r_{t+1} + \gamma V^{\pi}(s_{t+1}) \,|\, s_t = s \right\} \end{aligned}$$

Or, without the expectation operator:

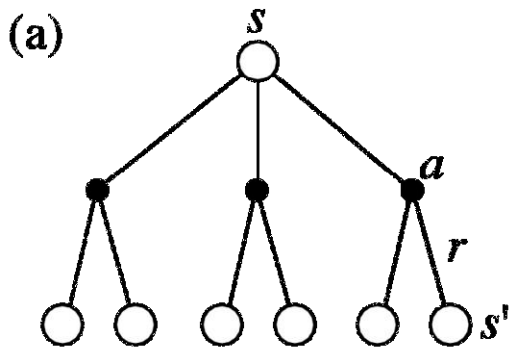
$$V^{\pi}(s) = \sum_a \pi(s,a) \sum_{s'} \mathbf{P}^a_{ss'} \left[\mathbf{R}^a_{ss'} + \gamma V^{\pi}(s') \right]$$

More on the Bellman Equation

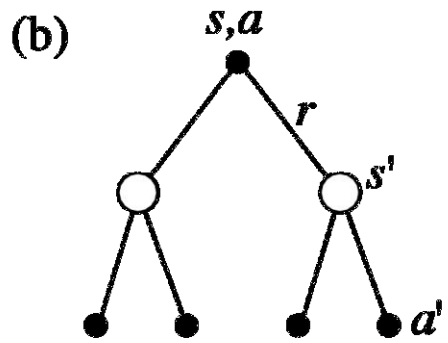
$$V^{\pi}(s) = \sum_{a} \pi(s,a) \sum_{s'} \mathbf{P}^{a}_{ss'} \left[\mathbf{R}^{a}_{ss'} + \gamma V^{\pi}(s') \right]$$

This is a set of equations (in fact, linear), one for each state.
The value function for π is its unique solution.

Backup diagrams:



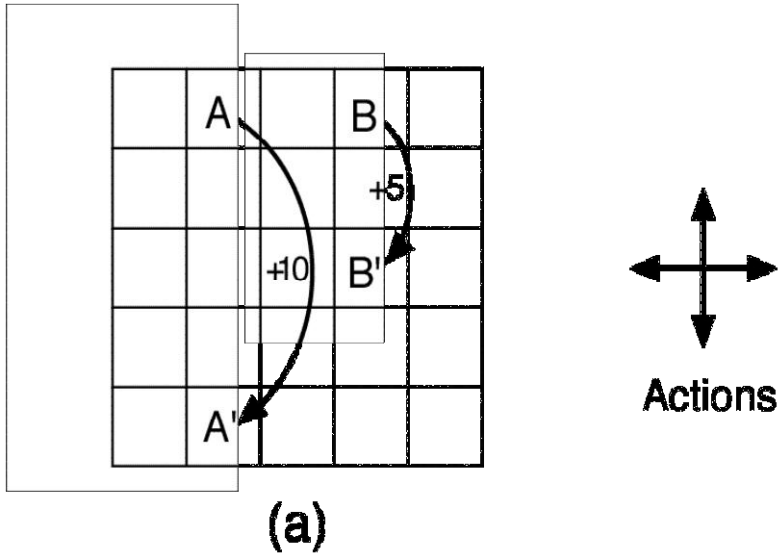
for V^{π}



for Q^{π}

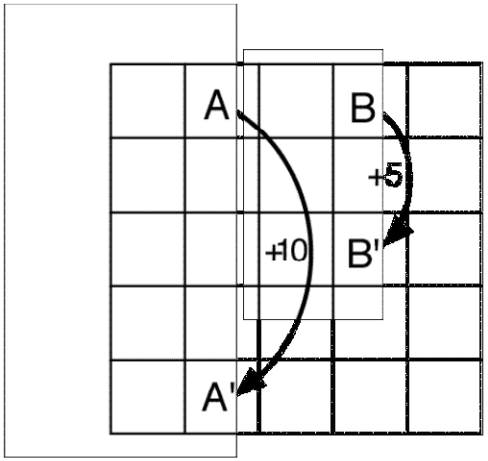
Gridworld

- Actions: `north`, `south`, `east`, `west`; deterministic.
- If it would take agent off the grid: no move but reward = –1
- Other actions produce reward = 0, except actions that move agent out of special states A and B, as shown.

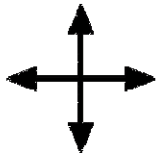


Gridworld

- Actions: `north`, `south`, `east`, `west`; deterministic.
- If would take agent off the grid: no move but reward = –1
- Other actions produce reward = 0, except actions that move agent out of special states A and B as shown.



(a)



3.3	8.8	4.4	5.3	1.5
1.5	3.0	2.3	1.9	0.5
0.1	0.7	0.7	0.4	-0.4
-1.0	-0.4	-0.4	-0.6	-1.2
-1.9	-1.3	-1.2	-1.4	-2.0

(b)

State-value function for equiprobable random policy; $\gamma = 0.9$

Optimal Value Functions

- For finite MDPs, policies can be **partially ordered**:

$$\pi \geq \pi' \quad \text{if and only if} \quad V^{\pi}(s) \geq V^{\pi'}(s) \quad \text{for all } s \in S$$

- There are always one or more policies that are better than or equal to all the others. These are the **optimal policies**. We denote them all π^*.
- Optimal policies share the same **optimal state-value function**:
- Optimal policies also share the same **optimal action-value function**:

$$V^*(s) = \max_{\pi} V^{\pi}(s) \quad \text{for all } s \in S$$

$$Q^*(s,a) = \max_{\pi} Q^{\pi}(s,a) \quad \text{for all } s \in S \text{ and } a \in A(s)$$

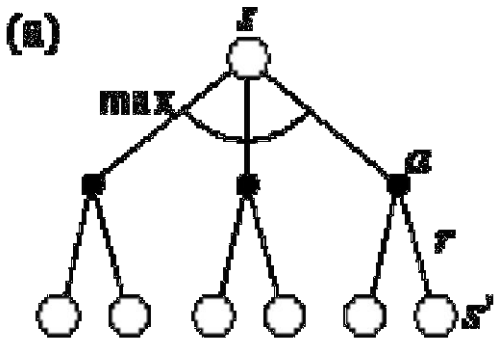
This is the expected return for taking action a in state s and thereafter following an optimal policy.

Bellman Optimality Equation for V^*

The value of a state under an optimal policy must equal the expected return for the best action from that state:

$$
\begin{aligned}
V^*(s) &= \max_{a \in A(s)} Q^{\pi^*}(s,a) \\
&= \max_{a \in A(s)} E\left\{ r_{t+1} + \gamma V^*(s_{t+1}) \,\middle|\, s_t = s, a_t = a \right\} \\
&= \max_{a \in A(s)} \sum_{s'} \mathbf{P}^a_{ss'} \left[\mathbf{R}^a_{ss'} + \gamma V^*(s') \right]
\end{aligned}
$$

The relevant backup diagram:



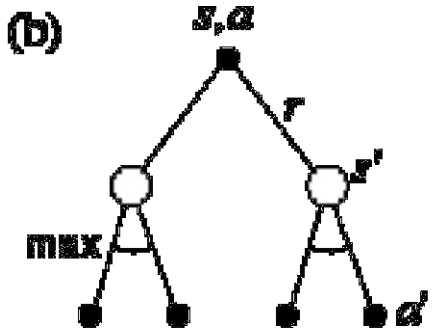
V^* is the unique solution of this system of nonlinear equations.

Bellman Optimality Equation for Q^*

$$Q^*(s,a) = E\left\{ r_{t+1} + \gamma \max_{a'} Q^*(s_{t+1}, a') \,\middle|\, s_t = s, a_t = a \right\}$$

$$= \sum_{s'} \mathbf{P}^a_{ss'} \left[\mathbf{R}^a_{ss'} + \gamma \max_{a'} Q^*(s', a') \right]$$

The relevant backup diagram:



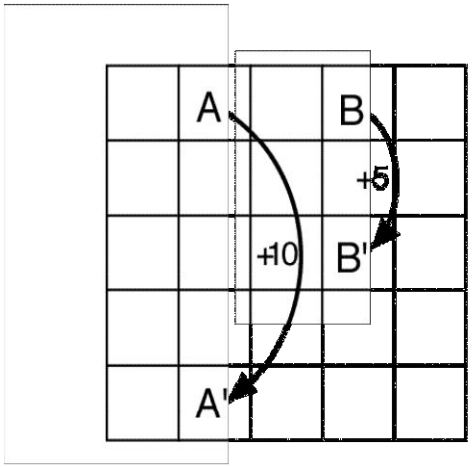
Q^* is the unique solution of this system of nonlinear equations.

Why Optimal State-Value Functions are Useful

Any policy that is greedy with respect to V^* is an optimal policy.

Therefore, given V^*, one-step-ahead search produces the long-term optimal actions.

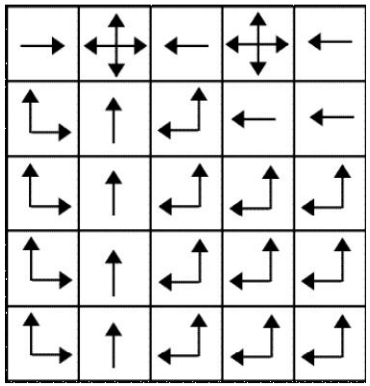
E.g., back to the gridworld:



a) gridworld

22.0	24.4	22.0	19.4	17.5
19.8	22.0	19.8	17.8	16.0
17.8	19.8	17.8	16.0	14.4
16.0	17.8	16.0	14.4	13.0
14.4	16.0	14.4	13.0	11.7

b) V^*



c) $\neq^* \pi^*$

What About Optimal Action-Value Functions?

Given Q^*, the agent does not even have to do a one-step-ahead search:

$$\pi^*(s) = \arg\max_{a \in A(s)} Q^*(s,a)$$

Solving the Bellman Optimality Equation

- Finding an optimal policy by solving the Bellman optimality equation requires the following:
  - accurate knowledge of environment dynamics;
  - enough space and time to do the computation;
  - the Markov property.
- How much space and time do we need?
  - polynomial in number of states (via dynamic programming methods),
  - BUT, number of states is often huge (e.g., backgammon has about 10^{20} states).
- We usually have to settle for approximations.
- Many RL methods can be understood as approximately solving the Bellman optimality equation.

Dynamic programming

- main idea
  - use value functions to structure the search for good policies
  - need a perfect model of the environment

- two main components
  - policy evaluation: compute V^π from π
  - policy improvement: improve π based on V^π



  - start with an arbitrary policy
  - repeat evaluation/improvement until convergence

Policy evaluation/improvement

- policy evaluation: π -> V^{π}
  - Bellman eqn's define a system of n eqn's
  - could solve, but will use iterative version

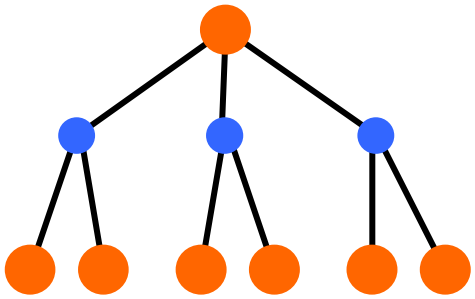
$$V_{k+1}(s) = \sum_a \pi(s,a) \sum_{s'} P^a_{ss'} [r^a_{ss'} + \gamma V_k(s')]$$

  - start with an arbitrary value function V_0, iterate until V_k converges

- policy improvement: V^{π} -> π'

$$\pi'(s) = \arg\max_a Q^{\pi}(s,a)$$

$$= \arg\max_a \sum_{s'} P^a_{ss'} \left[r^a_{ss'} + \gamma V^{\pi}(s') \right]$$



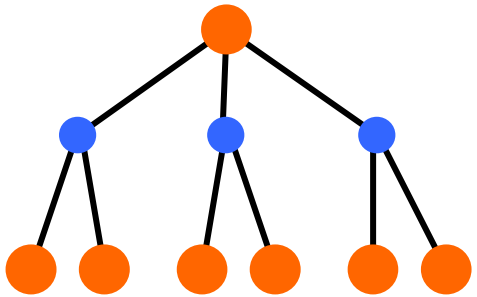
  - π' either strictly better than π, or π' is optimal (if $\pi = \pi'$)

Policy/Value iteration

- Policy iteration

$$\pi_0 \rightarrow^E V^{\pi_0} \rightarrow^I \pi_1 \rightarrow^E V^{\pi_1} \rightarrow^I \ldots \rightarrow^I \pi^* \rightarrow^E V^*$$

  - two nested iterations; too slow
  - don't need to converge to V^{π_k}
    - just move towards it



- Value iteration

$$V_{k+1}(s) = \max_a \sum_{s'} P^a_{ss'} \left[r^a_{ss'} + \gamma V_k(s') \right]$$

  - use Bellman optimality equation as an update
  - converges to V*

Using dynamic programming

- need complete model of the environment and rewards
  - robot in a room
    - state space, action space, transition model
- can we use DP to solve
  - robot in a room?
  - backgammon?
  - helicopter?
- DP bootstraps
  - updates estimates on the basis of other estimates

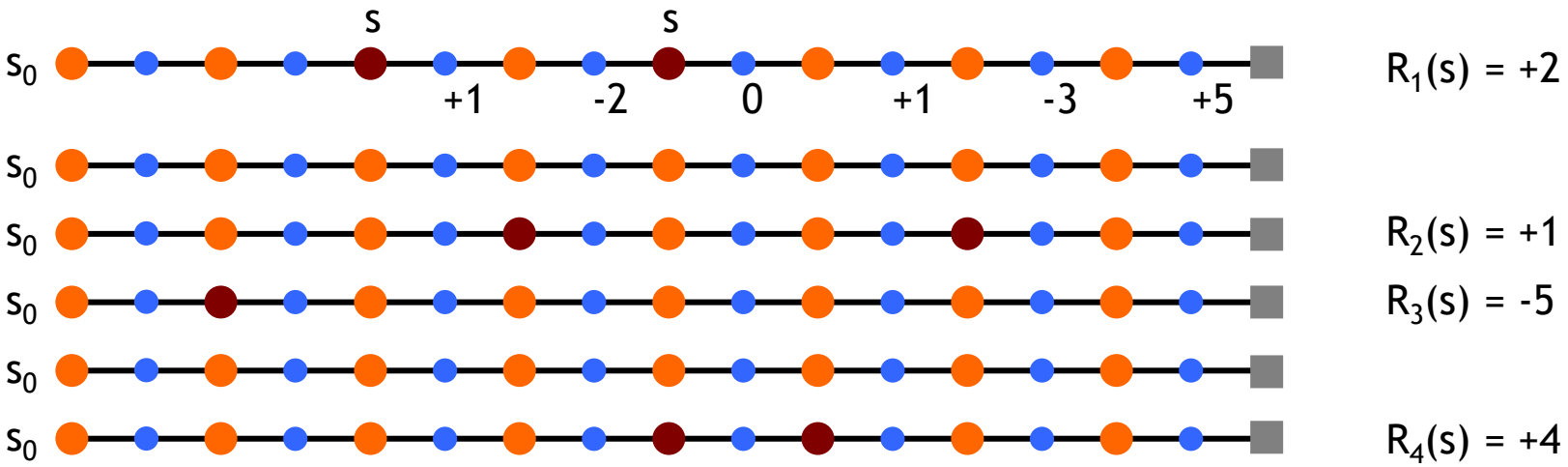
Monte Carlo methods

- don’t need full knowledge of environment
  - just experience, or
  - simulated experience
- averaging sample returns
  - defined only for episodic tasks
- but similar to DP
  - policy evaluation, policy improvement

Monte Carlo policy evaluation

- want to estimate $V^{\pi}(s)$
  = expected return starting from s and following π
  - estimate as average of observed returns in state s

- first-visit MC
  - average returns following the first visit to state s



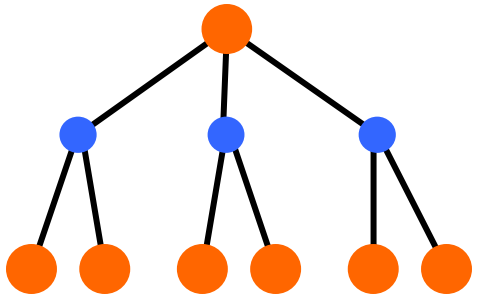
$$V^{\pi}(s) \approx (2 + 1 - 5 + 4)/4 = 0.5$$

Monte Carlo control

- V^{π} not enough for policy improvement
  - need exact model of environment

$$\pi'(s) = \arg\max_a Q^{\pi}(s,a)$$

- estimate $Q^{\pi}(s,a)$



- MC control

$$\pi_0 \rightarrow^{E} Q^{\pi_0} \rightarrow^{I} \pi_1 \rightarrow^{E} Q^{\pi_1} \rightarrow^{I} \ldots \rightarrow^{I} \pi^* \rightarrow^{E} Q^*$$

  - update after each episode

- non-stationary environment

$$V(s) \leftarrow V(s) + \alpha\,[R - V(s)]$$

- a problem
  - greedy policy won't explore all actions

Maintaining exploration

- key ingredient of RL

- deterministic/greedy policy won't explore all actions
  - don't know anything about the environment at the beginning
  - need to try all actions to find the optimal one

- maintain exploration
  - use *soft* policies instead: $\pi(s,a)>0$ (for all s,a)

- ε-greedy policy
  - with probability 1-ε perform the optimal/greedy action
  - with probability ε perform a random action

  - will keep exploring the environment
  - slowly move it towards greedy policy: ε -> 0

Simulated experience

- 5-card draw poker
  - s_0: A♣, A♦, 6♠, A♥, 2♠
  - a_0: discard 6♠, 2♠
  - s_1: A♣, A♦, A♥, A♠, 9♠ + dealer takes 4 cards
  - return: +1 (probably)

- DP
  - list all states, actions, compute P(s,a,s')
    - P([A♣,A♦,6♠,A♥,2♠], [6♠,2♠], [A♠,9♠,4]) = 0.00192

- MC
  - all you need are sample episodes
  - let MC play against a random policy, or itself, or another algorithm

Summary of Monte Carlo

- don’t need model of environment
  - averaging of sample returns
  - only for episodic tasks
- learn from:
  - sample episodes
  - simulated experience
- can concentrate on “important” states
  - don’t need a full sweep
- no bootstrapping
  - less harmed by violation of Markov property
- need to maintain exploration
  - use soft policies

Value Iteration

void valueIteration() {
  initialize $V(s)$ arbitrarily
  do{
    foreach ($s \in S$) {
      foreach ($a \in A$) {
        $Q(s,a) = R(s,a)+\gamma \Sigma_{s' \in S} T(s,a,s')V(s')$
        $V(s) = \max_a Q(s,a)$
      }
    }
  } **while** (! policy good enough) ;
}

Algorithm updates values backwards (from final states)

Value iteration: convergence

- Theorem: If the maximum difference between two successive value functions is less than ε, then the value of the greedy policy, (the policy obtained by choosing, in every state, the action that maximizes the estimated discounted reward, using the current estimate of the value function) differs from the value function of the optimal policy by no more than $2\varepsilon\gamma/(1-\gamma)$ at any state.
- An effective stopping criterion for the algorithm
- Value iteration is very flexible. The assignments to V need not be done in strict order but instead can occur asynchronously in parallel, provided that the value of every state gets updated infinitely often on an infinite run.

Policy iteration

choose an arbitrary policy π'

loop

$\pi := \pi'$

compute the value function of policy π:

solve the linear equations

$$V_\pi(s) = R(s, \pi(s)) + \gamma \sum_{s' \in \mathcal{S}} T(s, \pi(s), s') V_\pi(s')$$

improve the policy at each state:

$$\pi'(s) := \arg\max_a \left(R(s, a) + \gamma \sum_{s' \in \mathcal{S}} T(s, a, s') V_\pi(s') \right)$$

until $\pi = \pi'$

The value function of a policy is just the expected infinite discounted reward that will be gained, at each state, by executing that policy. It can be determined by solving a set of linear equations. Once we know the value of each state under the current policy, we consider whether the value could be improved by changing the first action taken. If it can, we change the policy to take the new action whenever it is in that situation. This step is guaranteed to strictly improve the performance of the policy. When no improvements are possible, then the policy is guaranteed to be optimal.

Approximate solutions

- Learning with time differences (TD),
  a model is not needed, incremental, difficult for analysis
- Dynamic programming,
  mathematically well defined problems with exact and complete description of the environment
- Monte Carlo methods,
  model is not necessary, conceptually simple, not incremental, sampling complete trajectories in interaction with environment (or model of environment)
- Efficiency, convergence

TD(λ) learning

- Learning with time differences
- Previous states receive a portion of the difference to successors
- For λ=0
  $V(s_t) = V(s_t) + c(V(s_{t+1}) - V(s_t))$
- c is a parameter, slowly decreasing during learning assuring convergence
- For $\lambda > 0$, more than just immediate successors are taken into account (speed)

Temporal Difference Learning

- combines ideas from MC and DP
  - like MC: learn directly from experience (don't need a model)
  - like DP: bootstrap
  - works for continuous tasks, usually faster then MC

- constant-alpha MC:
  - have to wait until the end of episode to update

$$V(s_t) \leftarrow V(s_t) + \alpha \left[R_t - V(s_t)\right]$$



target

- simplest TD
  - update after every step, based on the successor

$$V(s_t) \leftarrow V(s_t) + \alpha \left[r_{t+1} + \gamma V(s_{t+1}) - V(s_t)\right]$$



MC vs. TD

- observed the following 8 episodes:

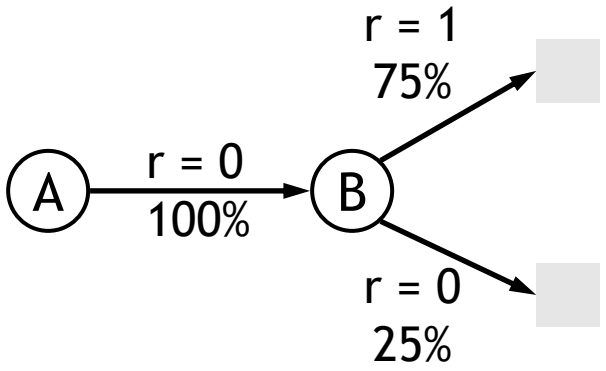
  A – 0, B – 0 B – 1 B – 1 B - 1

  B – 1 B – 1 B – 1 B – 0

- MC and TD agree on V(B) = 3/4

- MC: V(A) = 0
  - converges to values that minimize the error on training data

- TD: V(A) = 3/4
  - converges to ML estimate of the Markov process



Q-learning

- previous algorithms: on-policy algorithms
  - start with a random policy, iteratively improve
  - converge to optimal

- Q-learning: off-policy
  - use any policy to estimate Q

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t) \right]$$

  - Q directly approximates Q* (Bellman optimality eqn)
  - independent of the policy being followed
  - only requirement: keep updating each (s,a) pair

- Sarsa

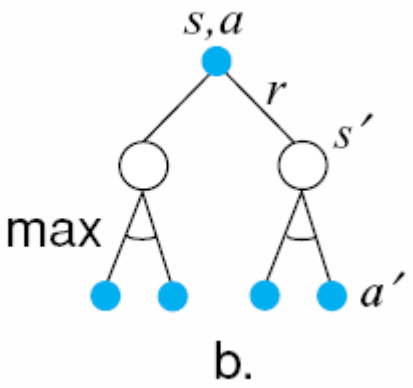
$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_t + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t) \right]$$

Q learning

- Watkins, 1989
- The most popular variant of time difference learning
- One step ahead

$Q(s_t,a_t) = (1\text{-}c)\, Q(s_t,a_t) + c(r_{t+1} + \gamma \max_a Q(s_{t+1},a) - Q(s_t,a_t))$

$0 <= c, \gamma <= 1$



Q-Learning: Definitions

- Current state: ***s***
- Current action: ***a***
- Transition function: $\delta(s, a) = s'$
- Reward function: $r(s, a) \in R$
- Policy $\pi(s) = a$
- $Q(s, a) \approx$ value of taking action ***a*** from state ***s***

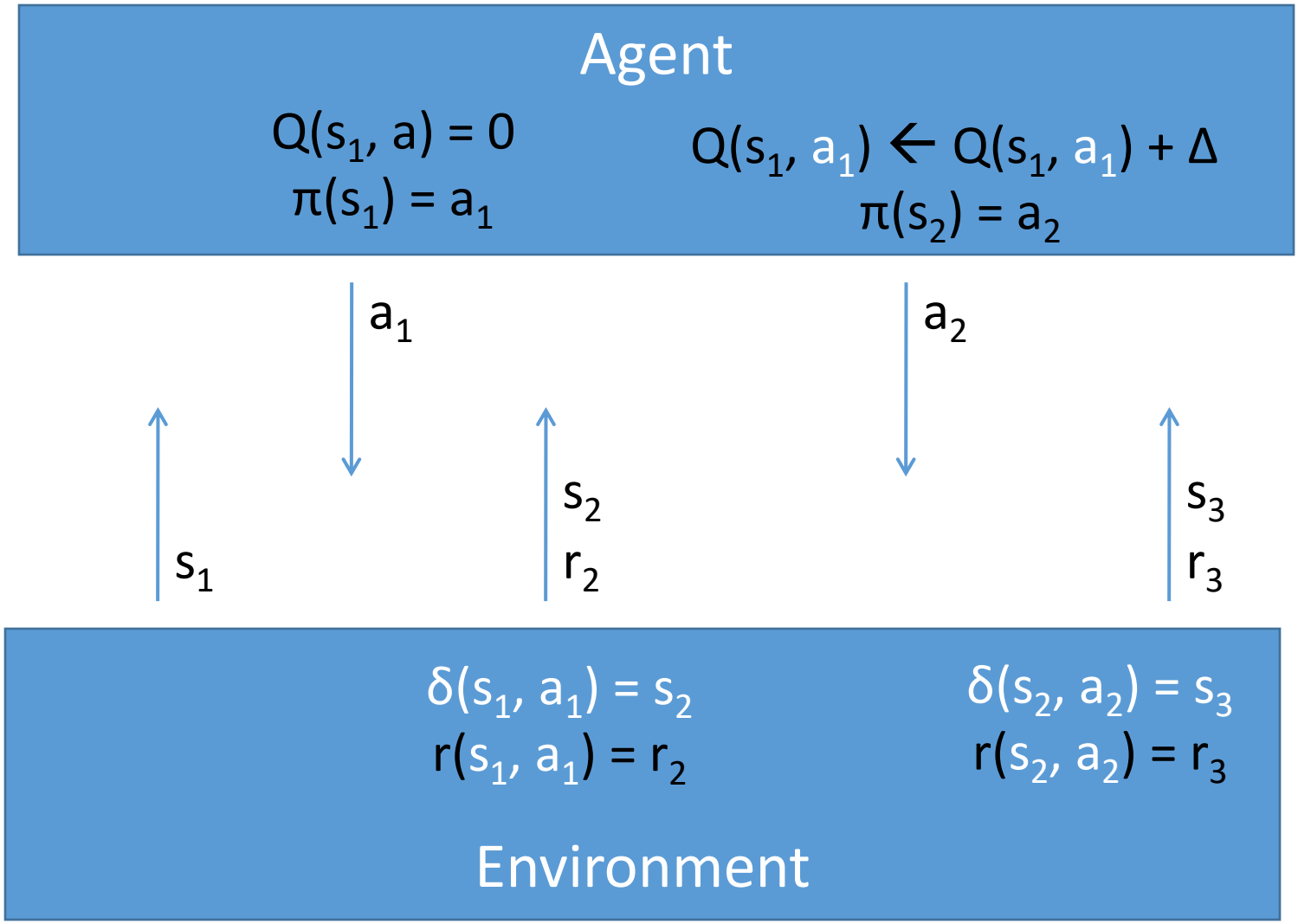
Markov property: this is independent of previous states given current state

In classification we'd have examples $(s, \pi(s))$ to learn from

The Q-function

- ***Q(s, a)*** estimates the *discounted cumulative reward*
  - Starting in state ***s***
  - Taking action ***a***
  - Following the current policy thereafter
- Suppose we have the optimal Q-function
  - What's the optimal policy in state ***s***?
  - The action $\boldsymbol{argmax_b\ Q(s, b)}$
- But we don't have the optimal Q-function at first
  - Let's act as if we do
  - And updates it after each step so it's closer to optimal
  - Eventually it will be optimal!

Q-Learning: The Procedure



Q-Learning: Updates

- The basic update equation

$$Q(s,a) \longleftarrow r(s,a) + \max_b Q(s',b)$$

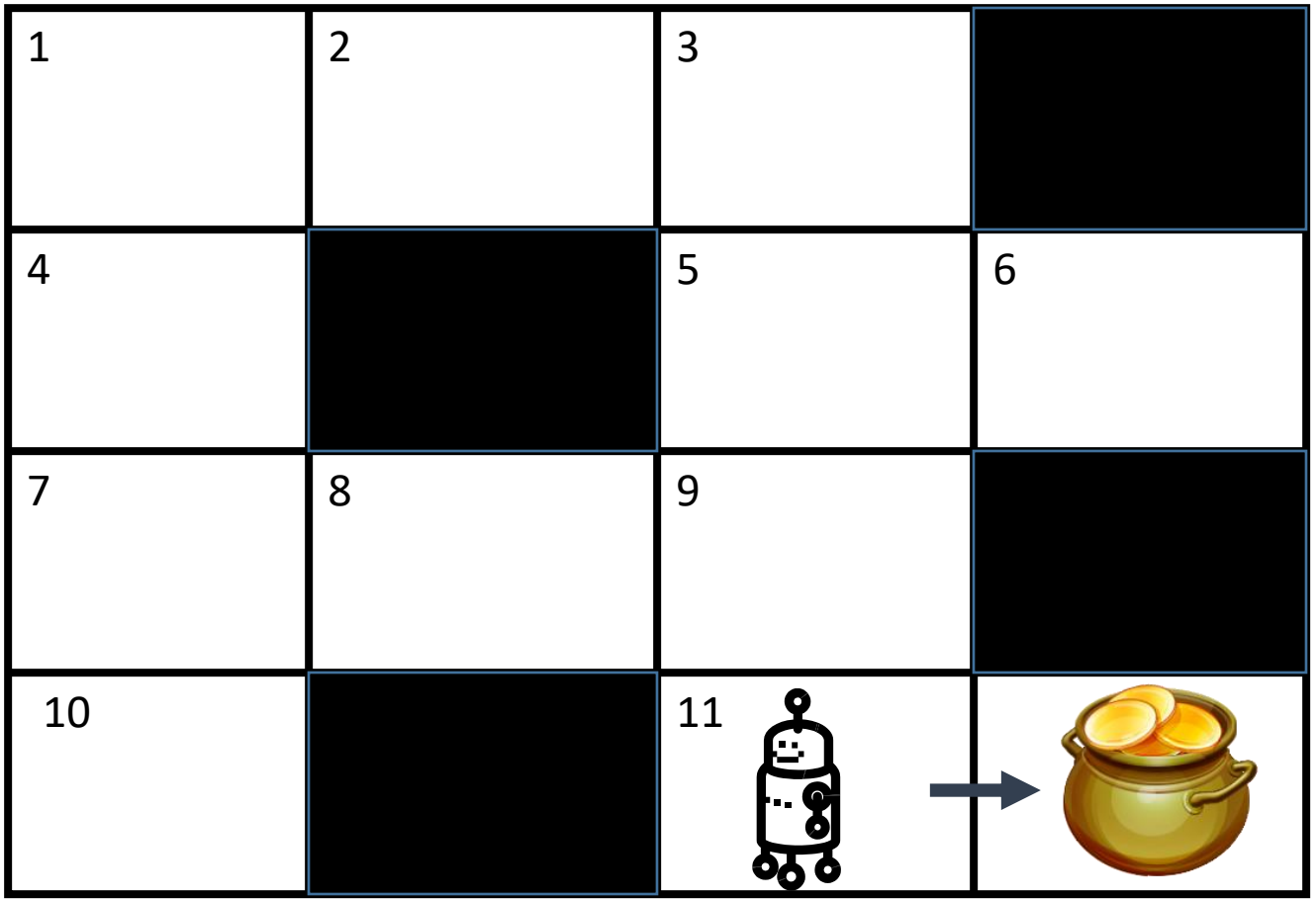
- With a discount factor to give later rewards less impact

$$Q(s,a) \longleftarrow r(s,a) + \gamma \max_b Q(s',b)$$

- With a learning rate for non-deterministic worlds

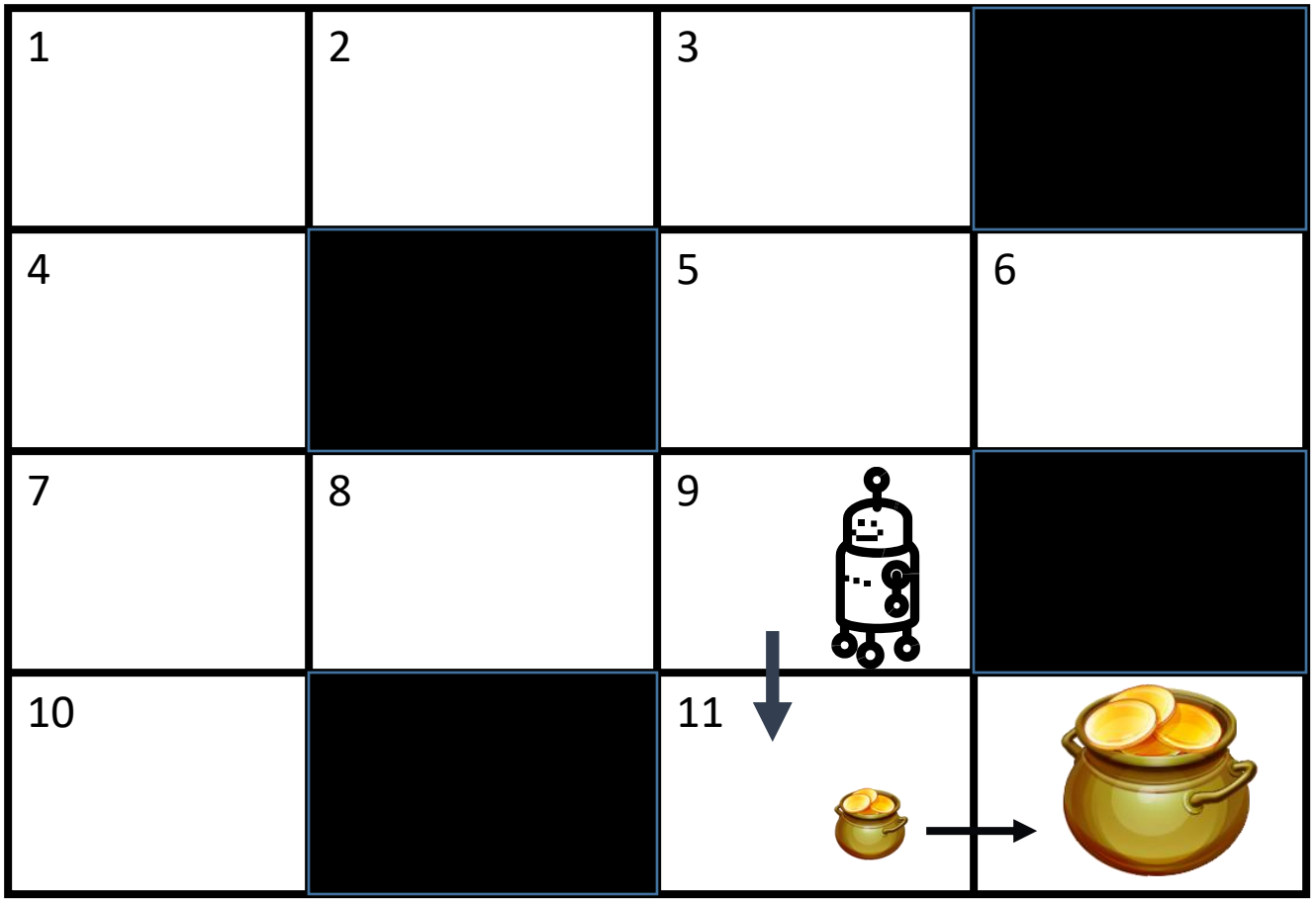
$$Q(s,a) \longleftarrow [1-\alpha]Q(s,a) + \alpha[r(s,a) + \gamma \max_b Q(s',b)]$$

Q-Learning: Update Example



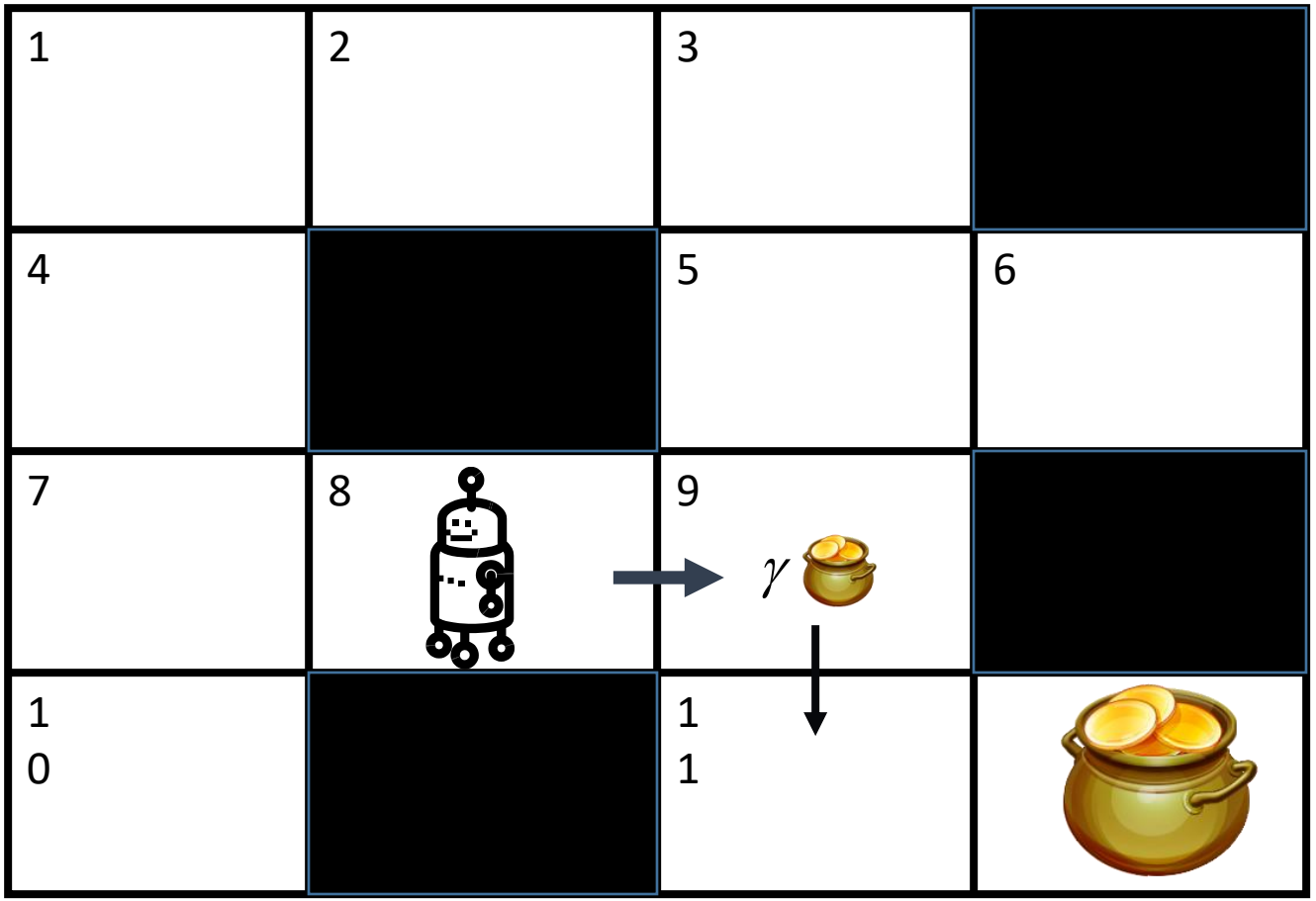
$$Q(s_{11}, a_{\rightarrow}) =$$

Q-Learning: Update Example



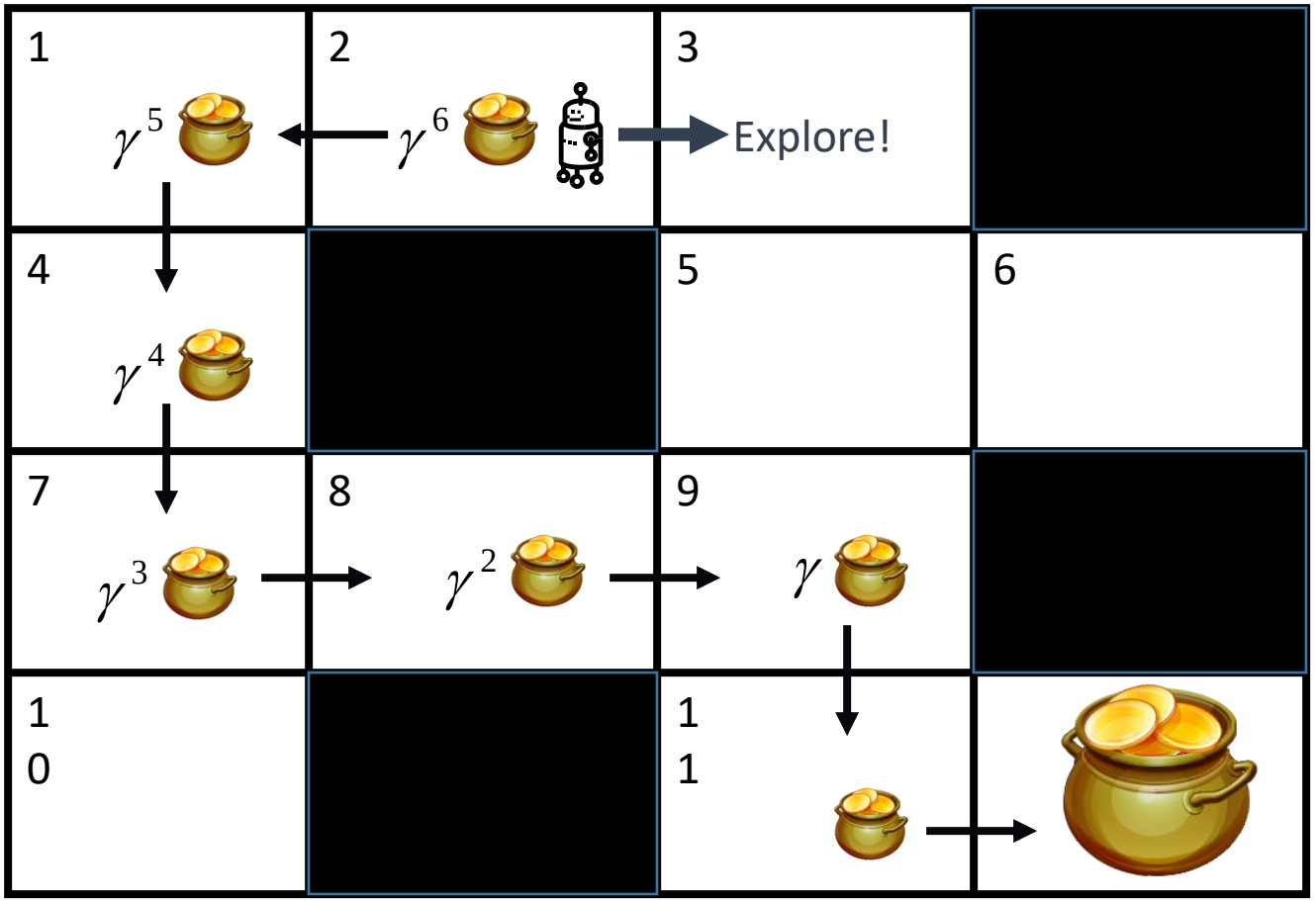
$$Q(s_9, a_{\downarrow}) = 0 + \gamma$$

Q-Learning: Update Example



$$Q(s_8, a_{\rightarrow}) = 0 + \gamma^2$$

The Need for Exploration



$$\arg\max Q(s_2, a) = \leftarrow$$

$$best = \rightarrow$$

Q learning

Initialize $Q(s, a)$ arbitrarily
Repeat (for each episode):
    Initialize s
    Repeat (for each step of episode):
        Choose a from s using policy derived from Q (e.g., ε-greedy)
        Take action a, observe r, s'
        $Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma \max_{a'} Q(s', a') - Q(s, a)]$
        $s \leftarrow s'$;
    until s is terminal

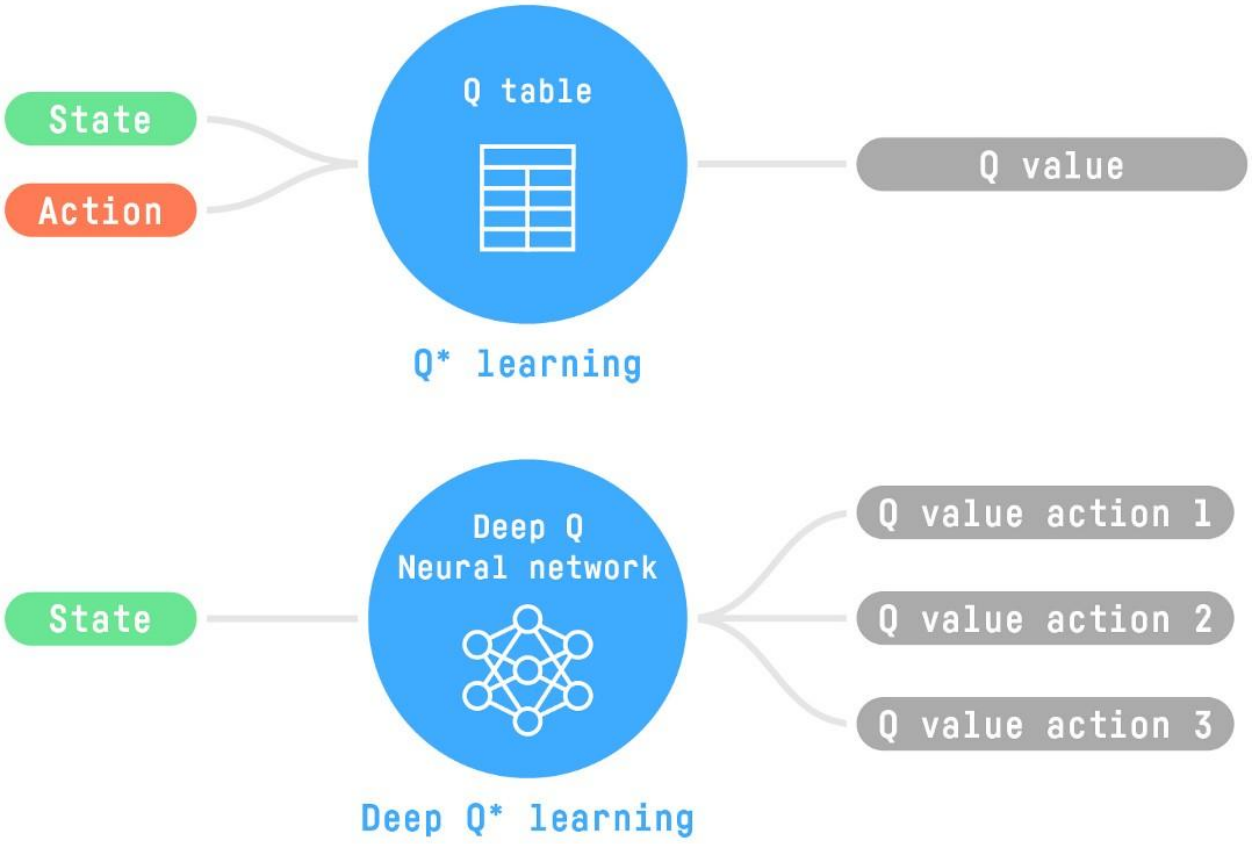
Explore/Exploit Tradeoff

- Can't always choose the action with highest Q-value
  - The Q-function is initially unreliable
  - Need to explore until it is optimal
- Most common method: ε-greedy
  - Take a random action in a small fraction of steps (**ε**)
  - Decay **ε** over time
- There is some work on optimizing exploration
  - Kearns & Singh, ML 1998
  - But people usually use this simple method

Q-Learning: Convergence

- Under certain conditions, Q-learning will converge to the correct Q-function
  - The environment model doesn't change
  - States and actions are finite
  - Rewards are bounded
  - Learning rate decays with visits to state-action pairs
  - Exploration method would guarantee infinite visits to every state-action pair over an infinite training period

Deep Q learning

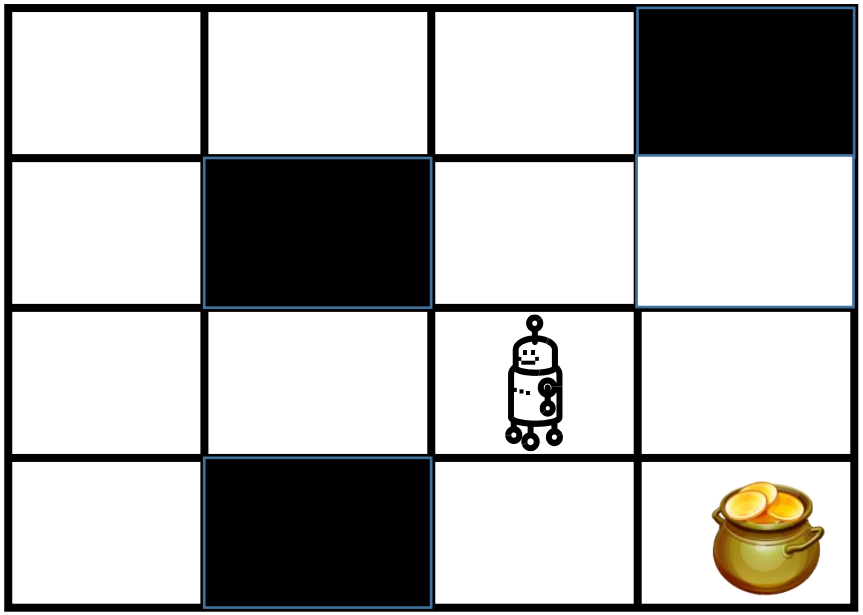


Extensions: SARSA

- SARSA: Take exploration into account in updates
  - Use the action actually chosen in updates

$$\cancel{Q(s,a) \longleftarrow r(s,a) + \gamma \max_b Q(s',b)}$$

$$Q(s,a) \longleftarrow r(s,a) + \gamma Q(s',a')$$

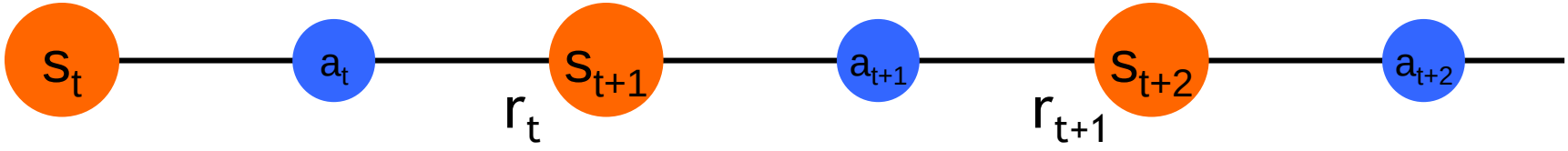


Regular: $\downarrow = \rightarrow$

SARSA: $\downarrow > \rightarrow$

Sarsa

- again, need Q(s,a), not just V(s)



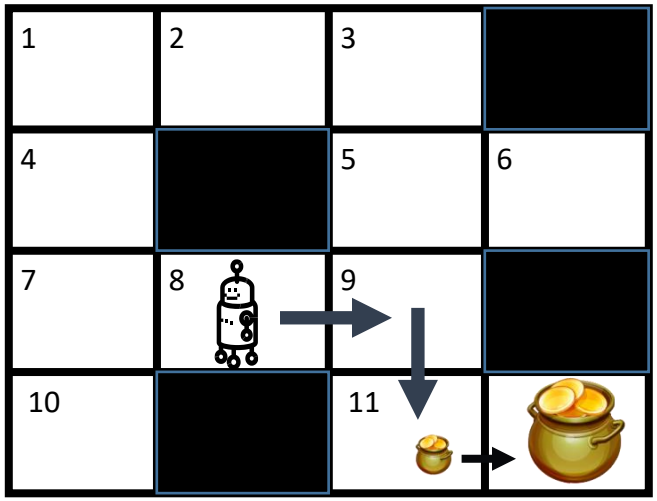
$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_t + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t) \right]$$

- control
  - start with a random policy
  - update Q and π after each step
  - again, need ε-soft policies

Extensions: Look-ahead

- Look-ahead: do updates over multiple states
  - Use some episodic memory to speed credit assignment

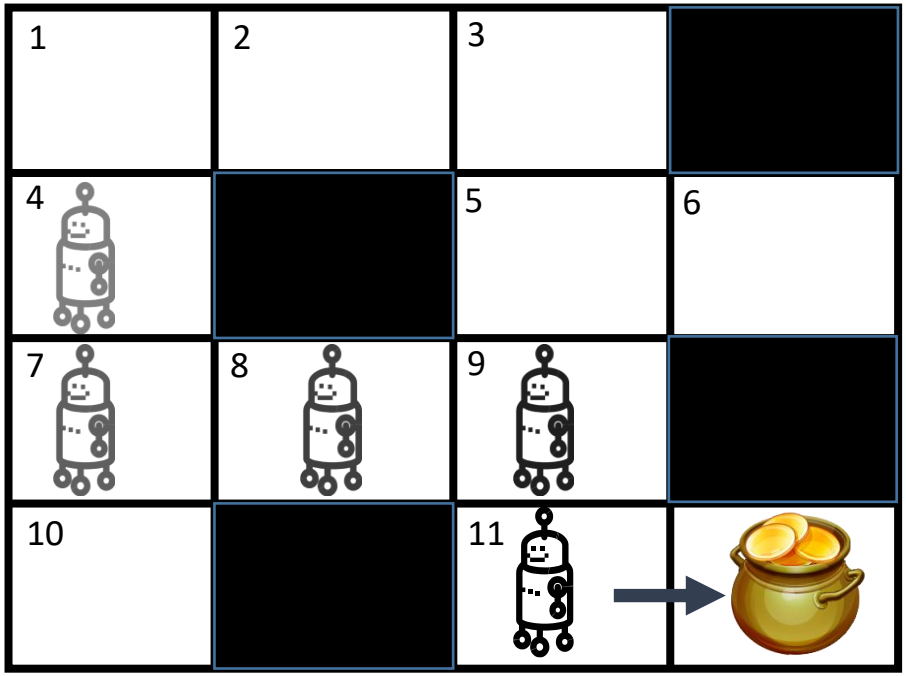
$$Q(s,a) \longleftarrow r(s,a) + \gamma r(s',a') + \gamma^2 Q(s'',a'')$$



- TD(**λ**): a weighted combination of look-ahead distances
  - The parameter **λ** controls the weighting

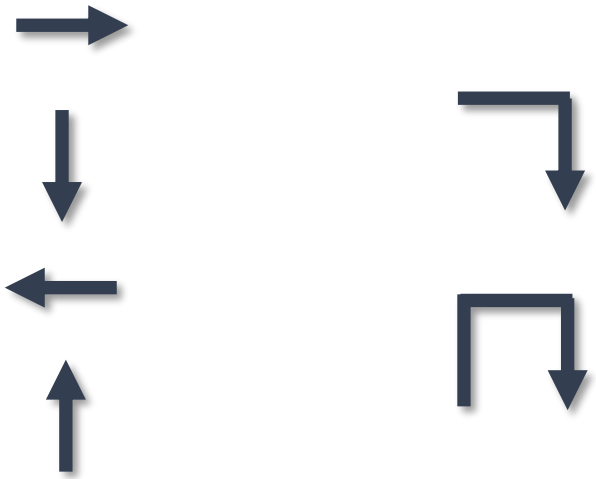
Extensions: Eligibility Traces

- Eligibility traces: Lookahead with less memory
  - Visiting a state leaves a trace that decays
  - Update multiple states at once
  - States get credit according to their trace

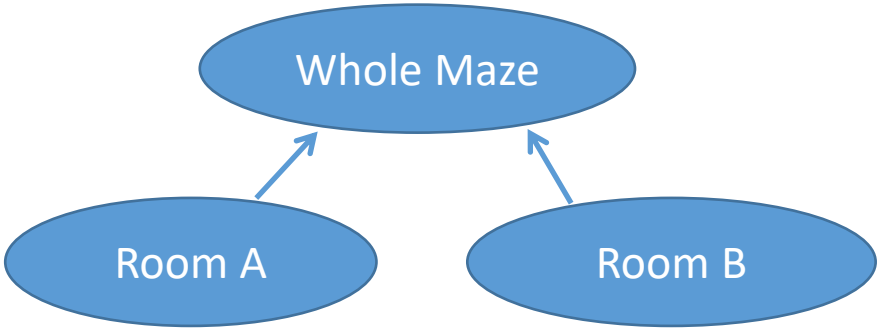


Extensions: Options and Hierarchies

- Options: Create higher-level actions

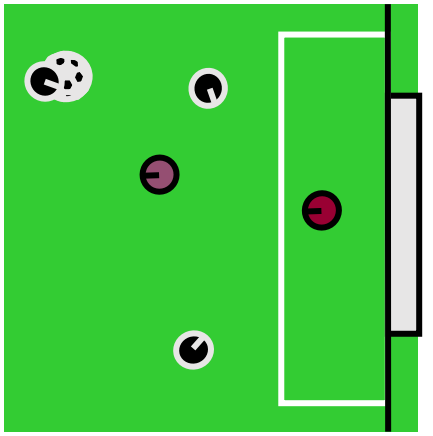


- Hierarchical RL: Design a tree of RL tasks



Extensions: Function Approximation

- Function approximation: allow complex environments
  - The Q-function table could be too big (or infinitely big!)
  - Describe a state by a feature vector
    $f = (f_1, f_2, \ldots, f_n)$
    - Then the Q-function can be any regression model
    - E.g. linear regression:
      $Q(s, a) = w_1 f_1 + w_2 f_2 + \ldots + w_n f_n$



  - Cost: convergence goes away in theory, though often not in practice
  - Benefit: generalization over similar states
  - Easiest if the approximator can be updated incrementally, like neural networks with gradient descent, but you can also do this in batches

Measuring learning performance

- Eventual convergence to optimality
  Many algorithms come with a provable guarantee of asymptotic convergence to optimal behavior. This is reassuring, but useless in practical terms.
- Speed of convergence to optimality
  Optimality is usually an asymptotic result, and so convergence speed is an ill-defined measure. More practical are
  - speed of convergence to near-optimality (how near?)
  - level of performance after a given time (what time?)
- Regret
  expected decrease in reward gained due to executing the learning algorithm instead of behaving optimally from the very beginning; these results are hard to obtain.

Challenges in Reinforcement Learning

- Feature/reward design can be very involved
  - Online learning (no time for tuning)
  - Continuous features (handled by *tiling*)
  - Delayed rewards (handled by *shaping*)
- Parameters can have large effects on learning speed
  - Tuning has just one effect: slowing it down
- Realistic environments can have partial observability
- Realistic environments can be non-stationary
- There may be multiple agents

Do Brains Perform RL?

- Should machine learning researchers care?
  - Planes don't fly the way birds do; should machines learn the way people do?
  - But why not look for inspiration?

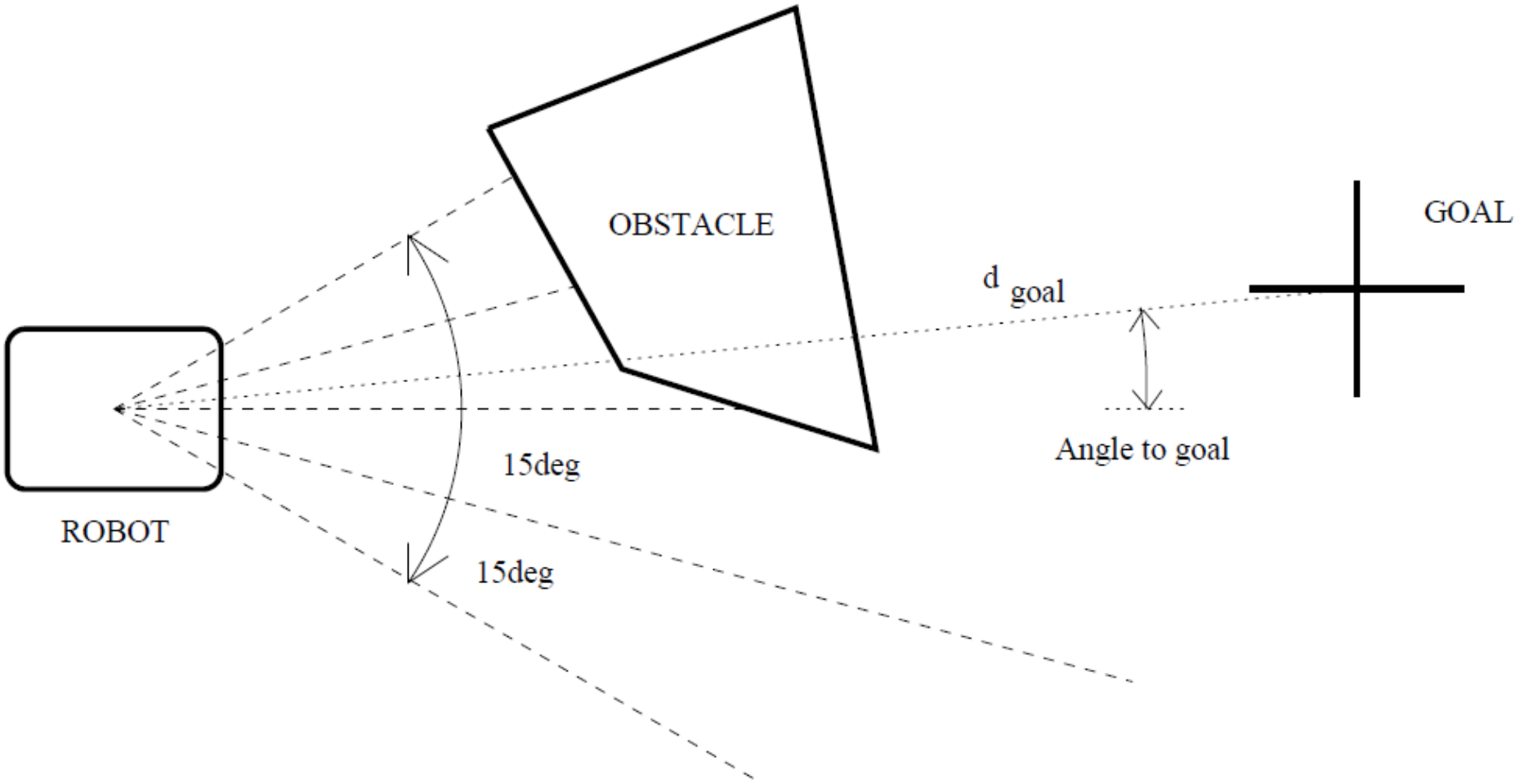
- Psychological research does show neuron activity associated with rewards
  - Really prediction error: actual – expected
  - Primarily in the striatum

What People Do Better

- Parallelism
  - Separate systems for positive/negative errors
  - Multiple algorithms running simultaneously
- Use of RL in combination with other systems
  - Planning: Reasoning about why things do or don't work
  - Advice: Someone to imitate or correct us
  - Transfer: Knowledge about similar tasks
- More impulsivity
  - Is this necessarily better?
- The goal for machine learning: Take inspiration from humans without being limited by their shortcomings

Some examples and details

An example: directing robot in 2d plane



- G.A.Rummery: Problem Solving with Reinforcement Learning, 1995

Robot in 2d: the settings

- sensors:
  - five distance measures to nearest obstacle in 15 degree forward arc
  - always knows distance and angle to the goal
- payoff after the end of the trial (reaching goal, collision with an obstacle or time out)
- start, goal and obstacles are randomly changed after every trial
- robot has to learn a generalized reactive policy; how?

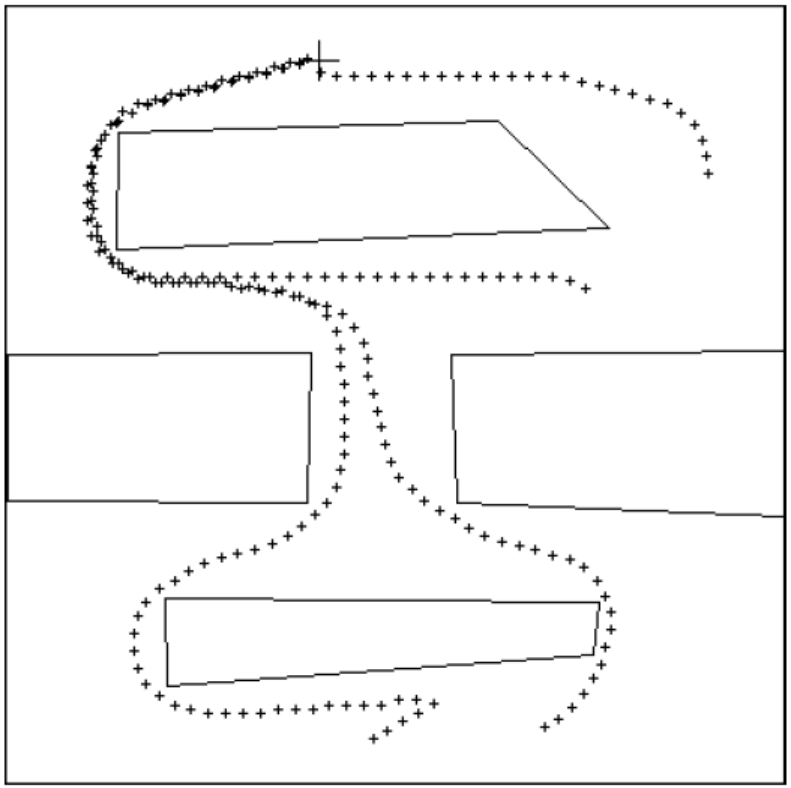
Robot in 2d: actions and rewards

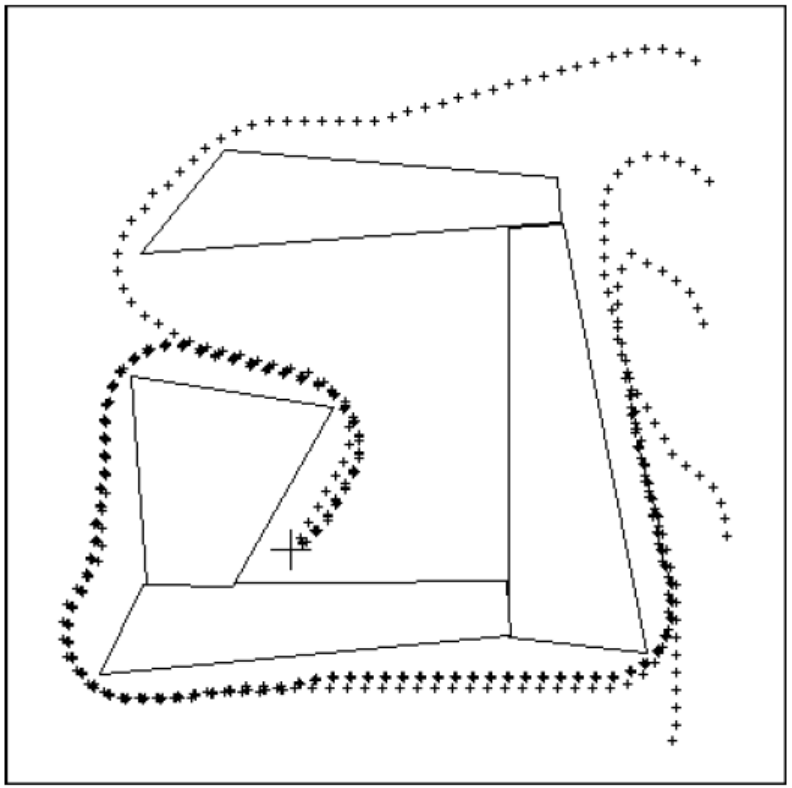
- 6 actions:
  - (turn left 15º, turn right 15º, stay in the same direction) x (move forward for a fixed distance d, do not move)
- rewards:
  - 0 in every step except the final
  - goal: if in a small fixed radius around the goal, +1
  - crash: based on a distance d from the goal e.g. $0.5 \exp(-\frac{2d_{goal}}{d_{max}})$, (note: maximum is 0.5)
  - time-out: as for crash + some small reward for not crashing, e.g., +0.3
- set γ = 0.99 to reward faster findings of the goal
- set probability of exploration/exploitation

Using NN for 2-d robot

- coarse coding the inputs (e.g., with several input sigmoids for each sensor)
- backpropagation with momentum term or eligibility traces
- batch and on-line training

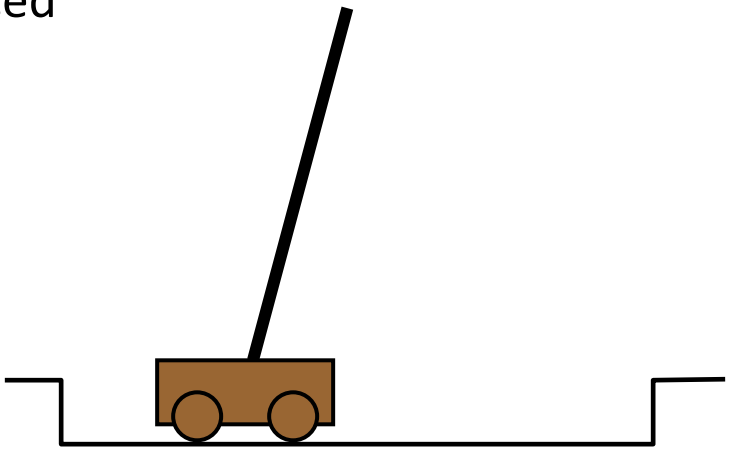
Some trajectories of trained robots





State representation

- pole-balancing
  - move car left/right to keep the pole balanced
- state representation
  - position and velocity of car
  - angle and angular velocity of pole
- what about *Markov property*?
  - would need more info
  - noise in sensors, temperature, bending of pole
- solution
  - coarse discretization of 4 state variables
    - left, center, right
  - totally non-Markov, but still works



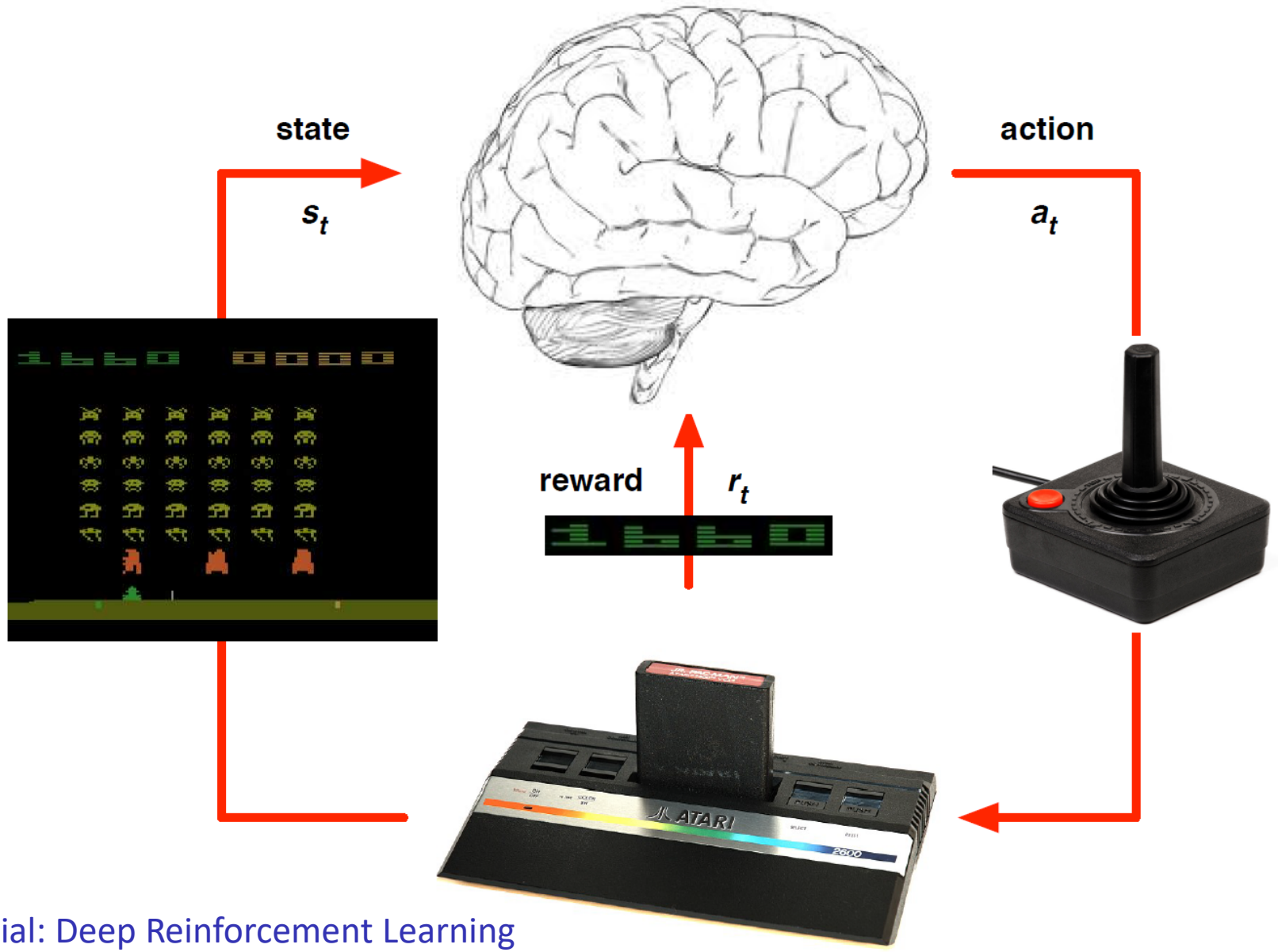
Designing rewards

- robot in a maze
  - episodic task, not discounted, +1 when out, 0 for each step
- chess
  - GOOD: +1 for winning, -1 losing
  - BAD: +0.25 for taking opponent's pieces
    - high reward even when lose
- rewards
  - rewards indicate what we want to accomplish
  - NOT how we want to accomplish it
- shaping



  - positive reward often very "far away"
  - rewards for achieving subgoals (domain knowledge)
  - also: adjust initial policy or initial value function

Reinforcement Learning in Atari

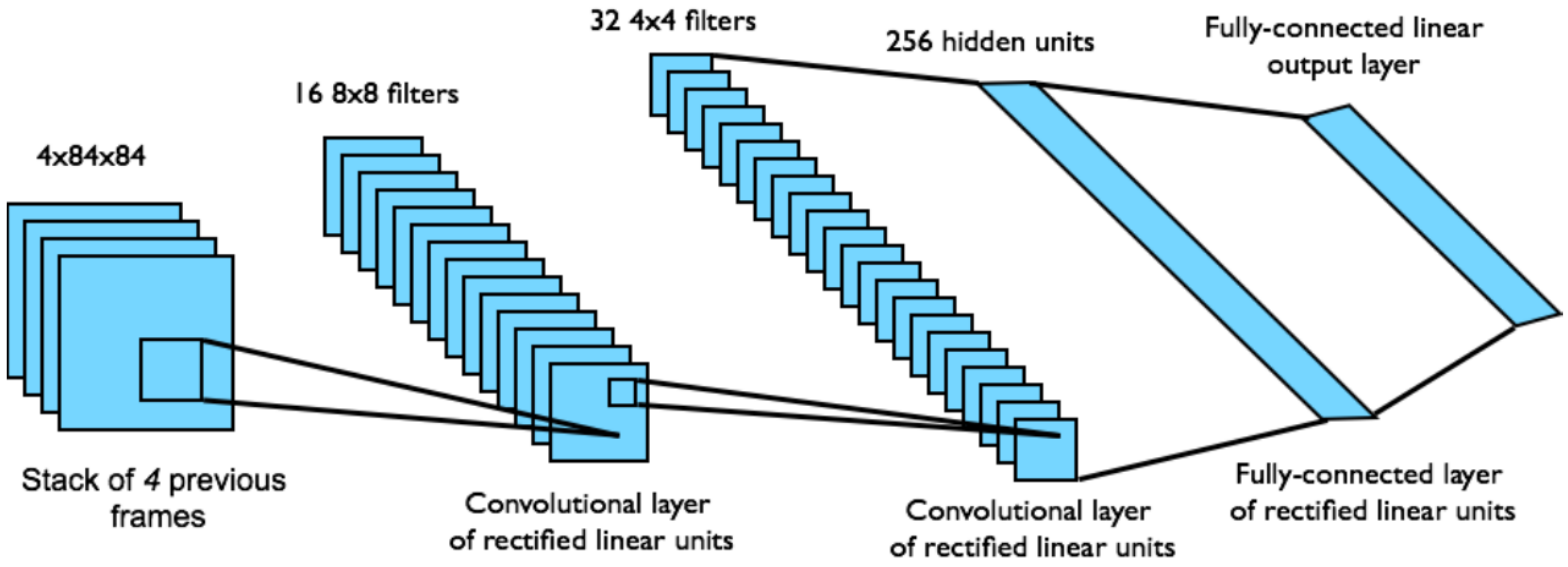


Tutorial: Deep Reinforcement Learning

David Silver, Google DeepMind

DQN in Atari

- End-to-end learning of values $Q(s, a)$ from pixels s
- Input state s is stack of raw pixels from last 4 frames
- Output is $Q(s, a)$ for 18 joystick/button positions
- Reward is change in score for that step



Network architecture and hyperparameters fixed across all games

[Mnih et al.]

Summary

- Reinforcement learning
  - use when need to make decisions in uncertain environment
  - actions have delayed effect

- solution methods
  - dynamic programming
    - need complete model

  - Monte Carlo
  - time difference learning (Sarsa, Q-learning)

- simple algorithms
- most work
  - designing features, state representation, rewards

University of Ljubljana, Faculty of Computer and Information Science

Inference and explanation of prediction models



Prof Dr Marko Robnik-Šikonja

Intelligent Systems, Edition 2025

Overview of topics



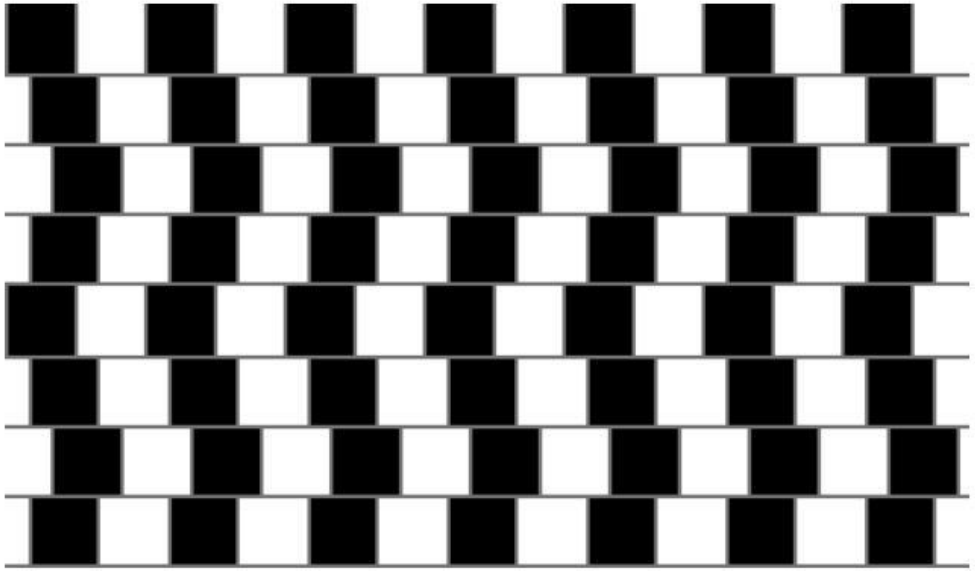
- Visualization and knowledge discovery.
- General methodology for explaining predictive models.
- Model level and instance level explanations, methods EXPLAIN and IME.

Visualization

- 1st rule of data mining: know your data.
- Therefore: visualizations, getting background data.
- Visualize: distributions of individual variables, their relations, etc.
- For high dimensional data sets one can use scaling, e.g. UMAP or t-SNE
- Clustering is useful in supervised tasks to get insight into the relation between predicted values Y and basic groups in the data. If unrelated, feature set might need amendments.

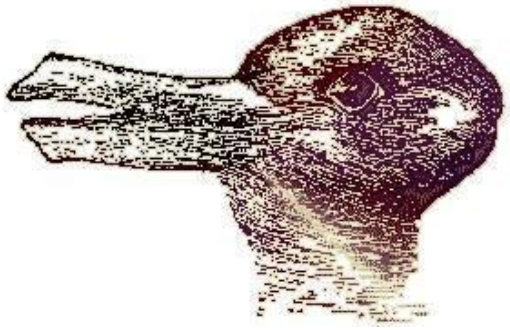
Visualizations

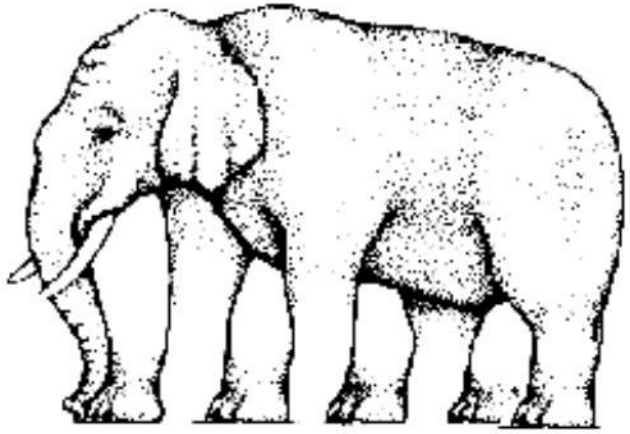
- Human visual perception has certain limitations:
  - we see what we want to see
  - we see what we see often
  - it is more difficult to notice unexpected patterns
- practice in detection of unknown
- use visualizations which expose “the unknown”



Are the horizontal lines parallel or do they slope?







How many legs does this elephant have?

Human pattern recognition

- We see inexistent patterns because we WANT to see them (we feel lost without them).



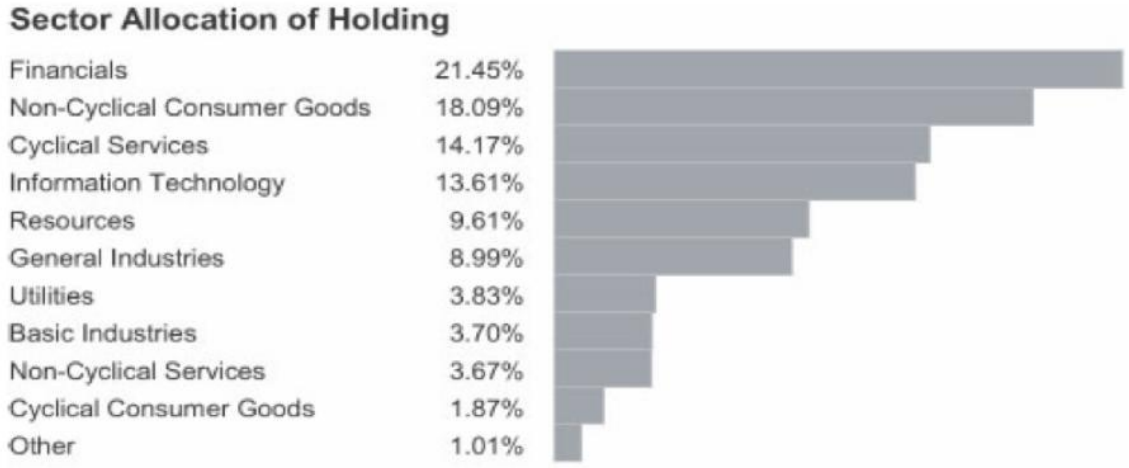


"The researchers found that when people were primed to feel out of control, they were more likely to see patterns where none exist." (**See a Pattern on Wall Street?,** *John Tierney*, *Science*)

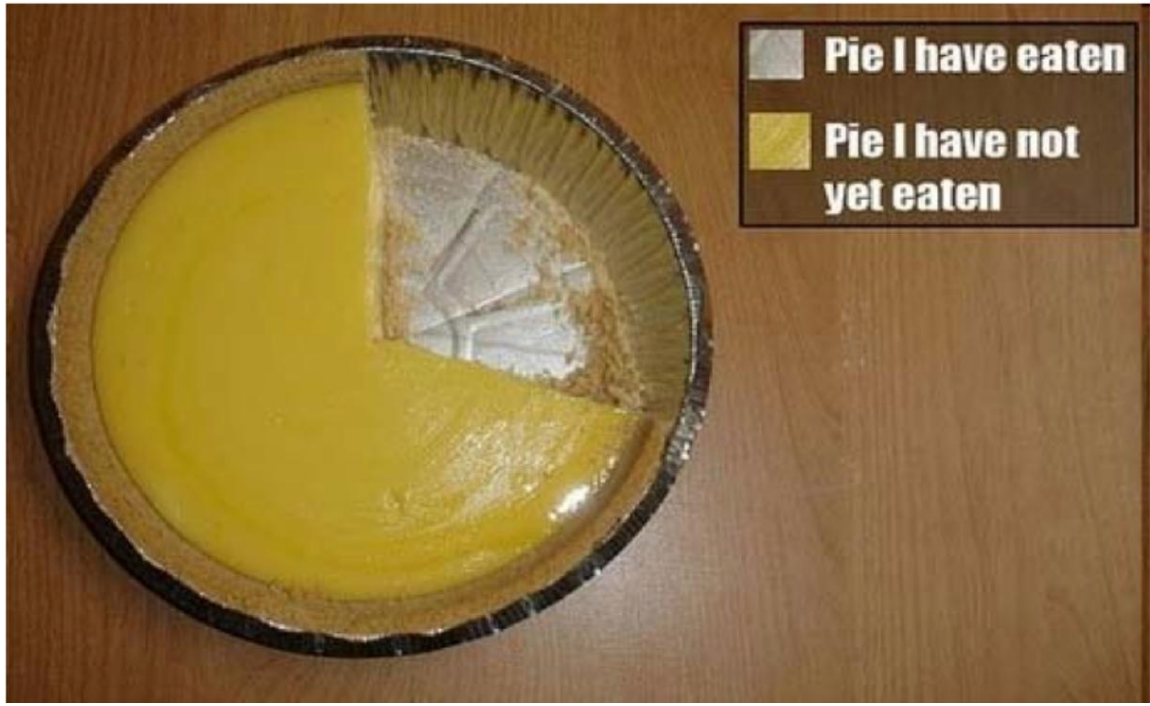
Facts about simple visualizations

- Pie charts are a bad choice: hard to read, similar colors, slope, legend is too far away
- Bar chart is much better



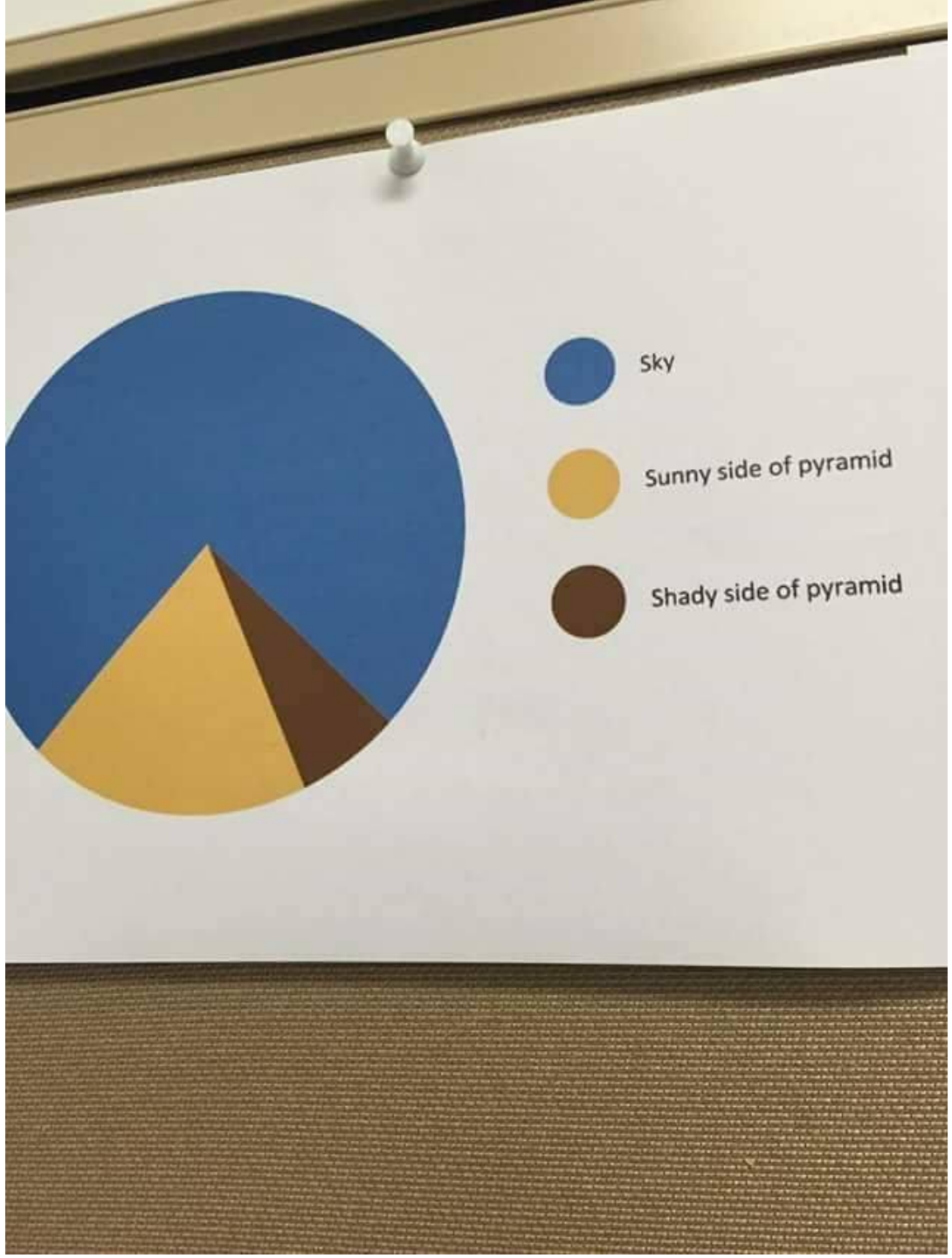


The best pie chart



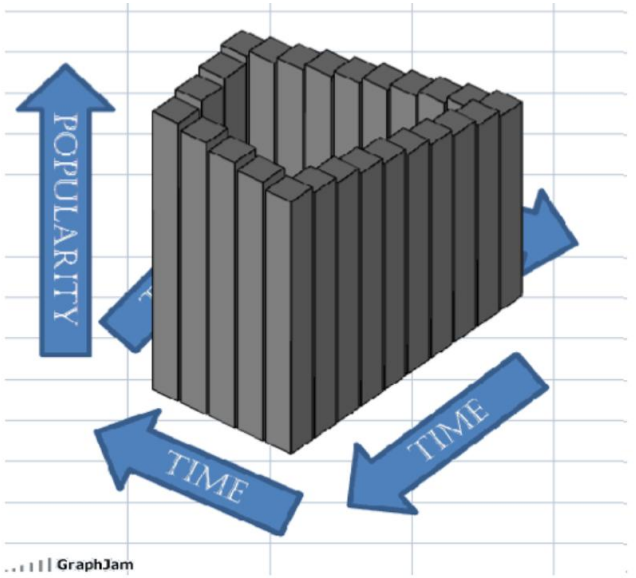
Pie charts jokes

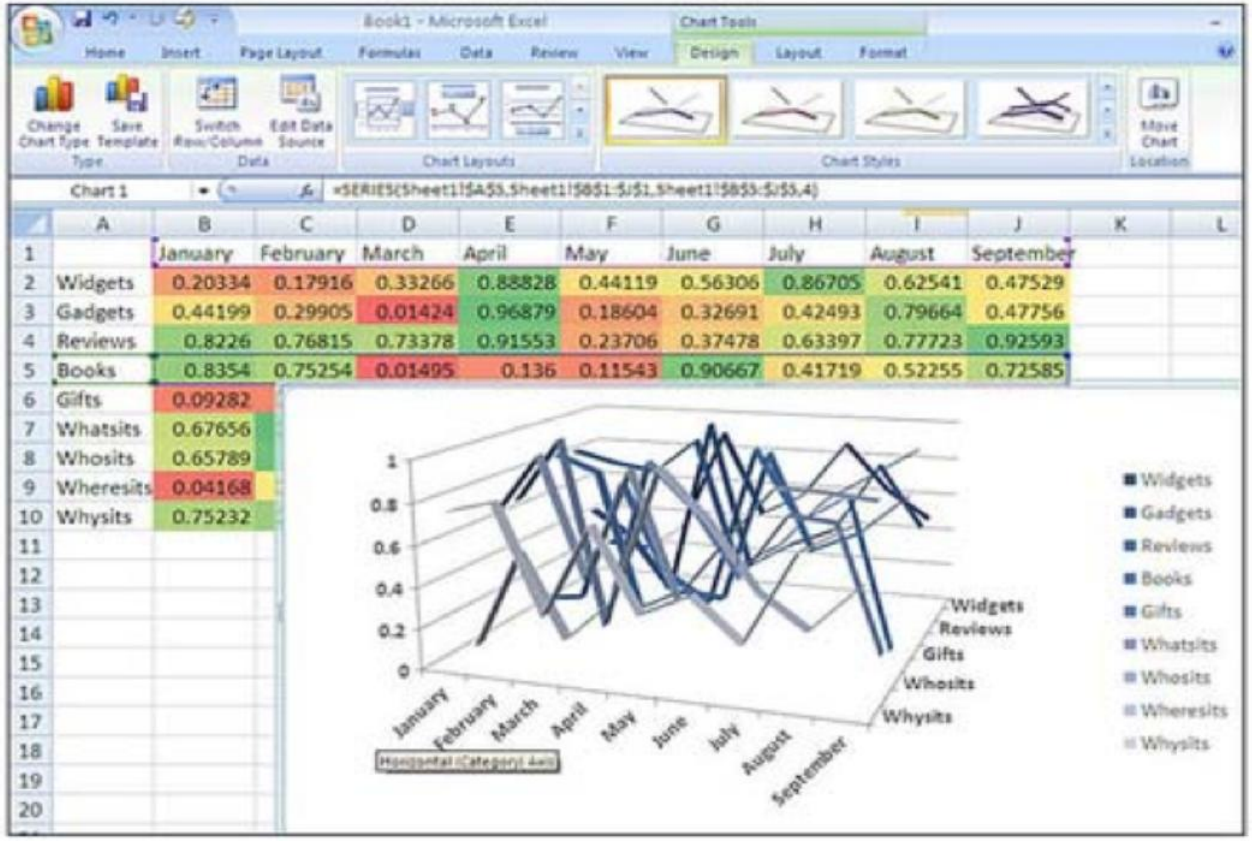
- notoriously bad

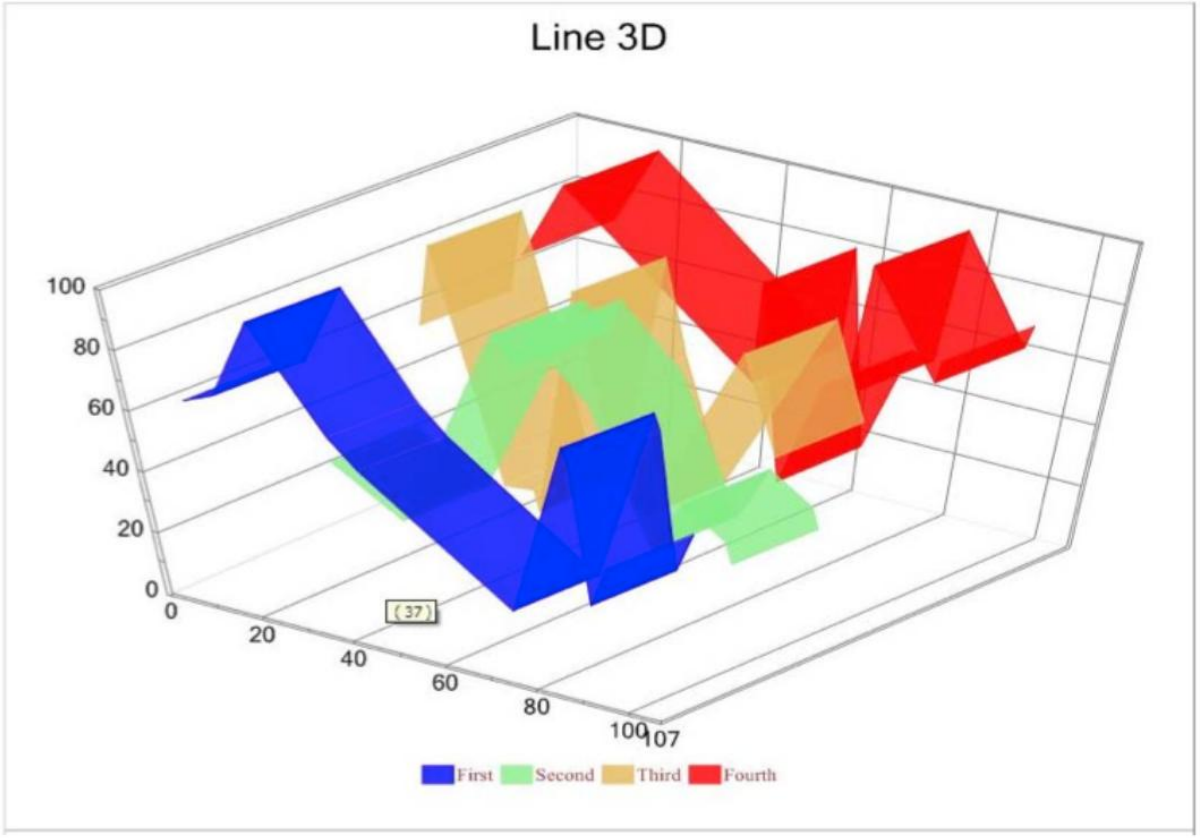


Facts about simple visualizations

- Bar charts, box plots can be OK
- 3D graphs are almost never OK for 2D info: spider plot, bowl of noodles
- Take care to be clear and do not manipulate
- A more detailed examples and recommendations https://github.com/cxli233/FriendsDontLetFriends

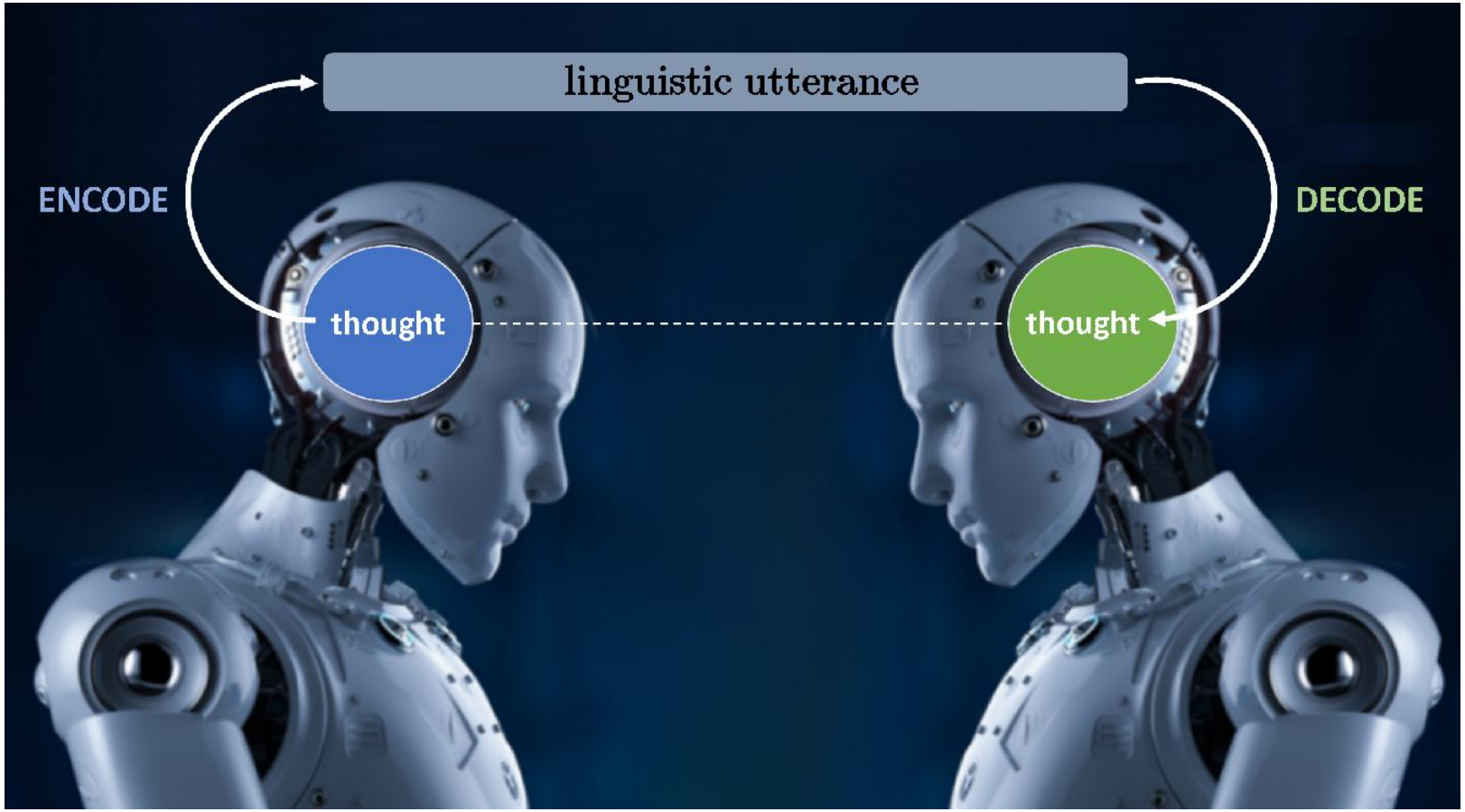






Understanding

Walid Saba, "Machine Learning Won't Solve Natural Language Understanding", The Gradient, 2021.

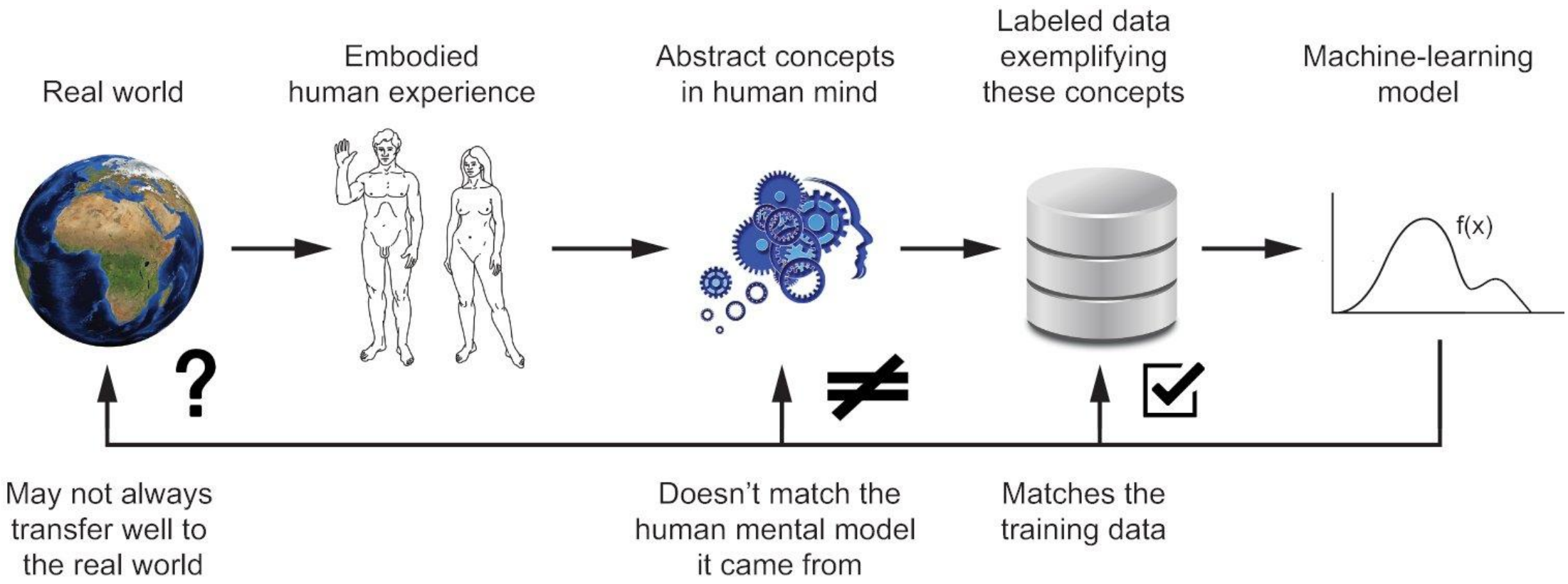


Xanadu, who is a living young human adult, and who was in graduate school, quit graduate school to join a software company that had a need for a new employee.

≈

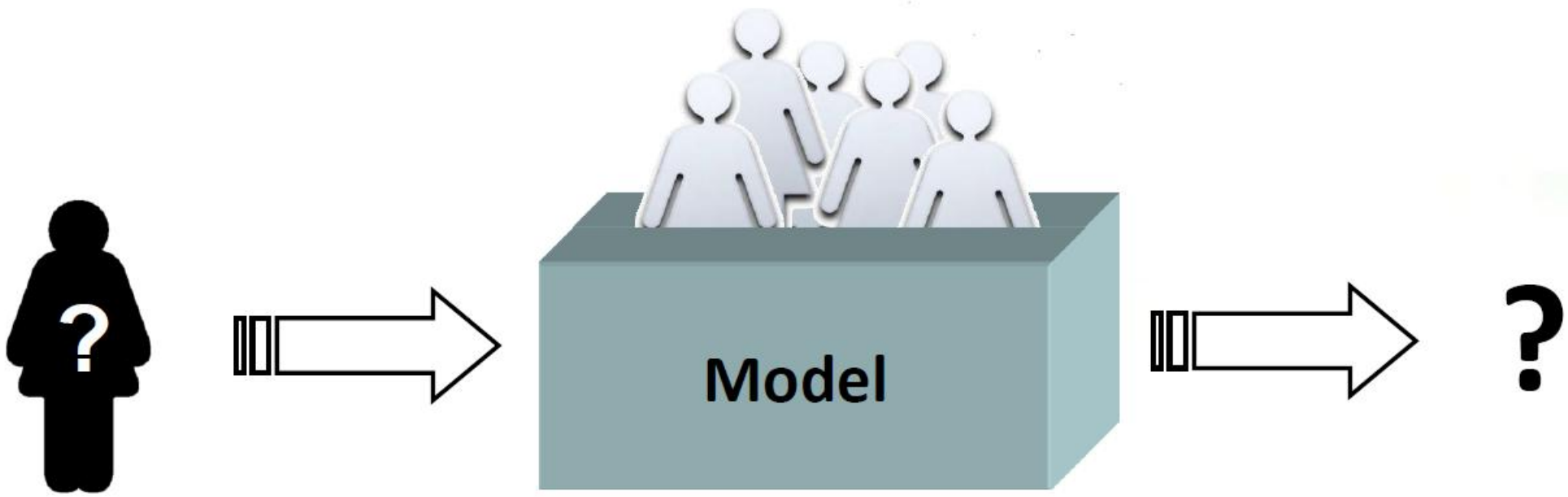
Xanadu quit graduate school to join a software company.

Understanding ML models is difficult



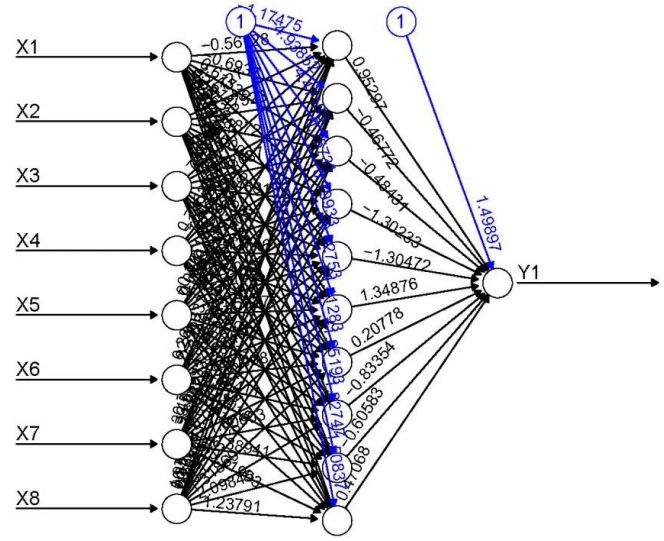
Predictive modeling scenario

We want to learn from past examples,with known outcomes.



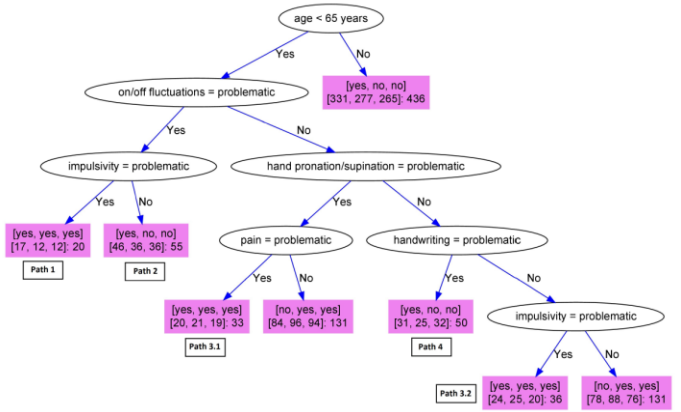
To predict the outcome for a new patient.

Explanation of predictions



- a number of successful prediction algorithms exist (SVM, boosting, random forests, neural networks), but to a user they are [redacted]
- many fields where users are very much concerned with the transparency of the models: medicine, law, consultancy, public services, etc.
- Some explanation methods are applicable to arbitrary predictors

Model comprehensibility



- decision support: model comprehensibility is important to gain users‘ trust
- knowledge acquisition
- some models are inherently interpretable and comprehensible
- decision and regression trees, classification and regression rules, linear and logistic regression $1/(1+\exp(-(b_0+b_1x_1+...b_px_p))$
- really?

Domain level explanation



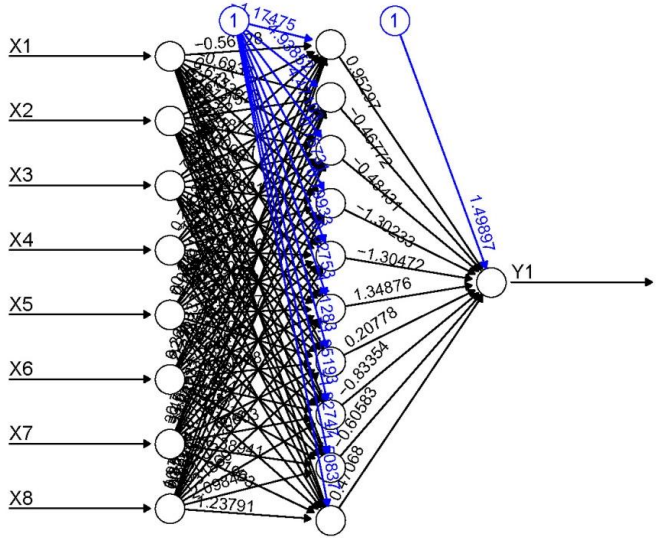
- trying to explain the "true causes and effects"
  - physical processes
  - stock exchange events

- usually unreachable except for artificial problems with known relations and generator function
- some aspects are covered with attribute evaluation, detection of redundancies, ...
- targeted indirectly through the models

Model-based explanations

All models are wrong,
but some are useful.

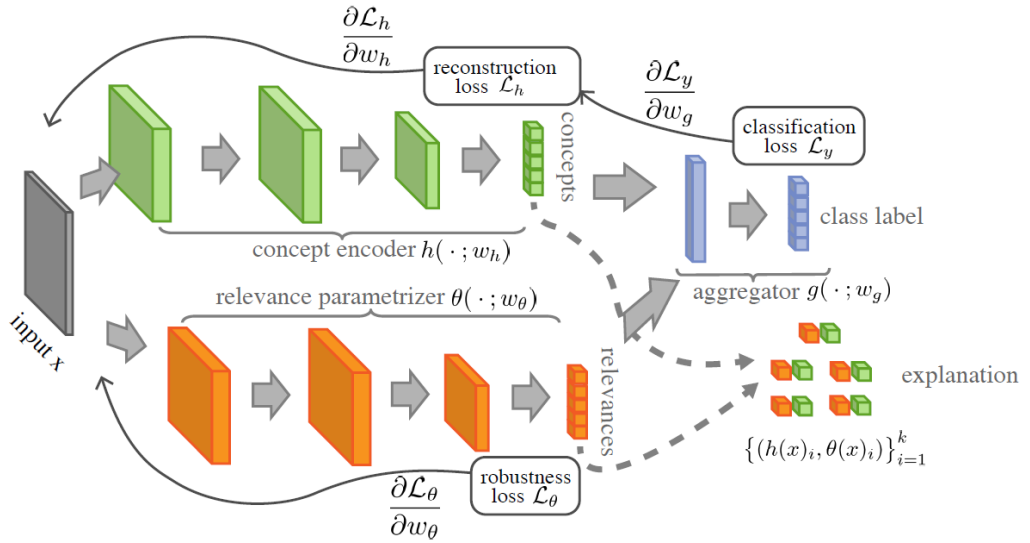
George Box, British statistician (1919 – 2013)



- make transparent the prediction process of a particular model
- the correctness of the explanation is independent of the correctness of the prediction but
- better models (with higher prediction accuracy) enable in principle better explanation at the domain level
- explanation methods are interested only in the explanation at the model level and leave to the developer of the model the responsibility for its prediction accuracy

Two flavours of explanation techniques

- model specific
  - especially used for deep neural networks

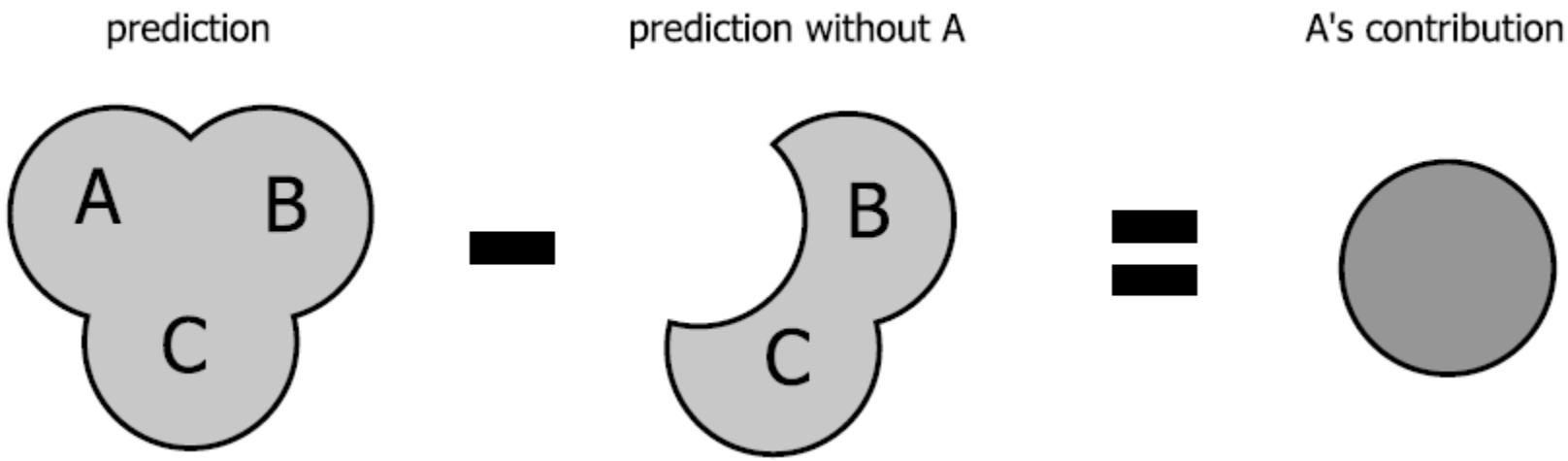


Melis, D.A. and Jaakkola, T., 2018. Towards robust interpretability with self-explaining neural networks. In *Advances in Neural Information Processing Systems* (pp. 7786-7795).

- model agnostic
  - can be used for any predictor,
  - based on perturbation of the inputs

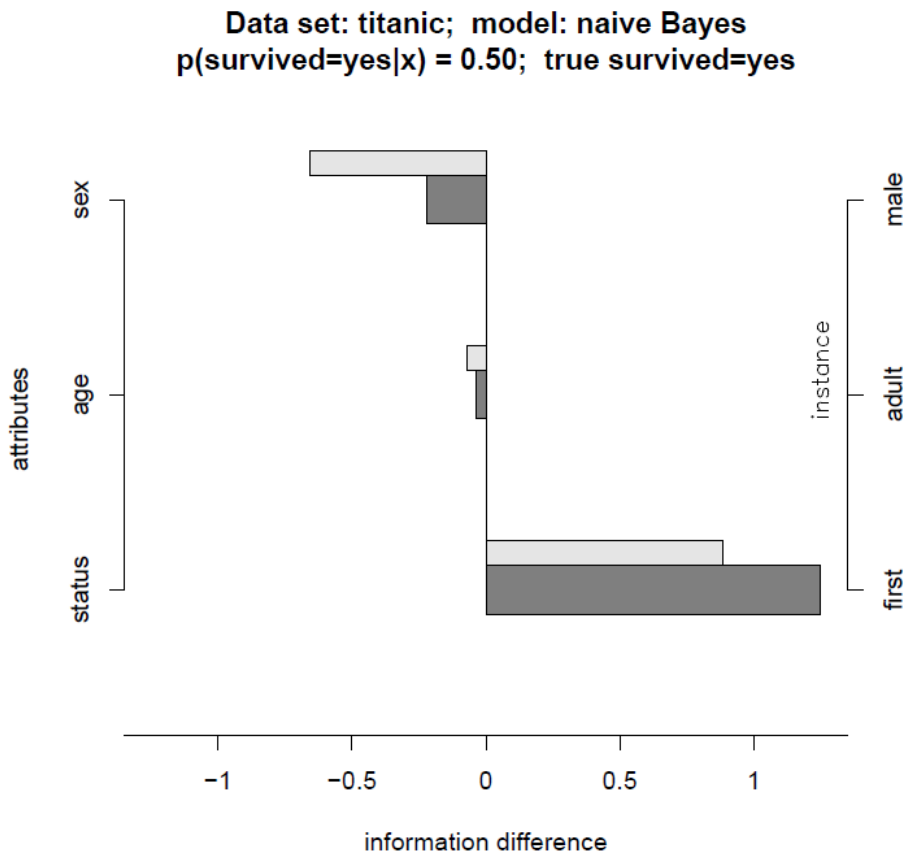
Idea of perturbtion-based explanations

- importance of a feature or a group of features in a specific model can be estimated by simulating lack of knowledge about the values of the feature(s)



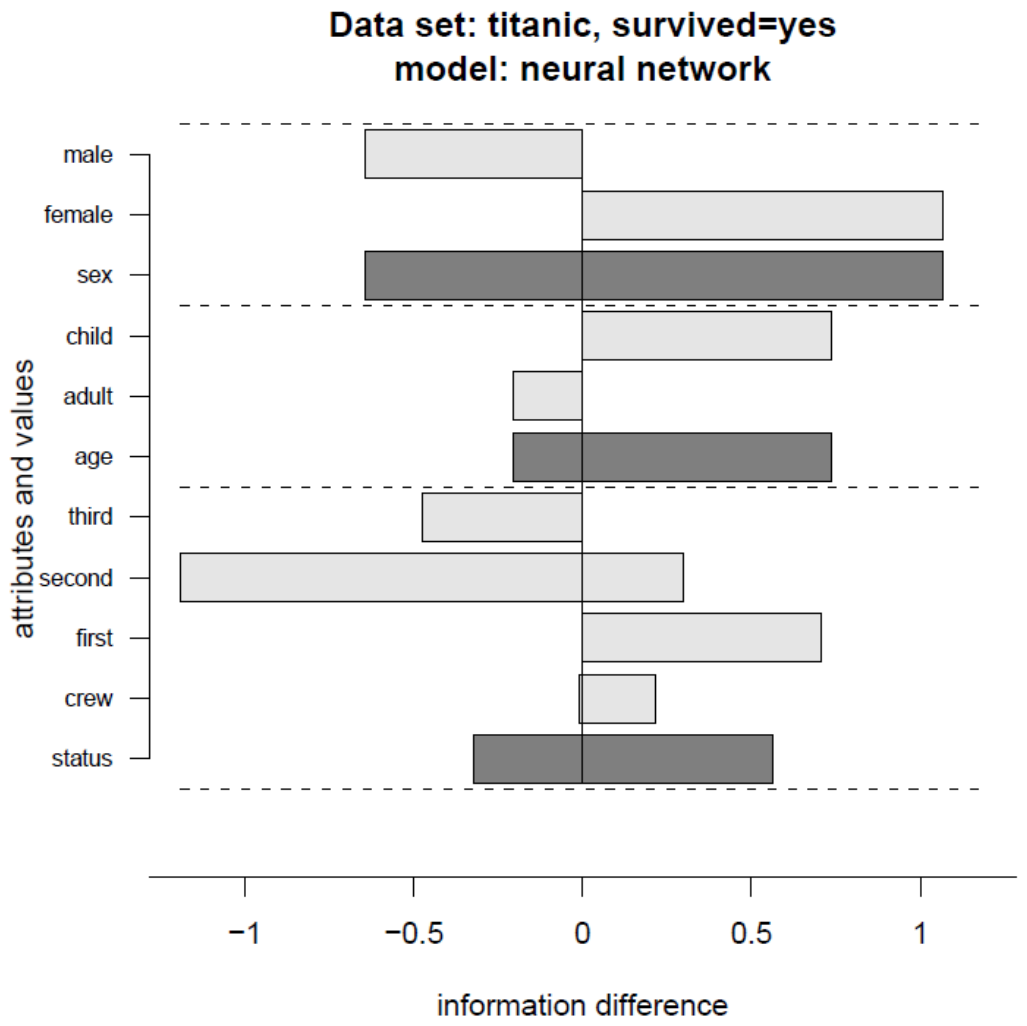
Instance-level explanation

- explain predictions for each instance separately
  - this is what practitioners applying models are interested in
  - presentation format: impact of each feature on the prediction value
- model-based



Model-level explanation

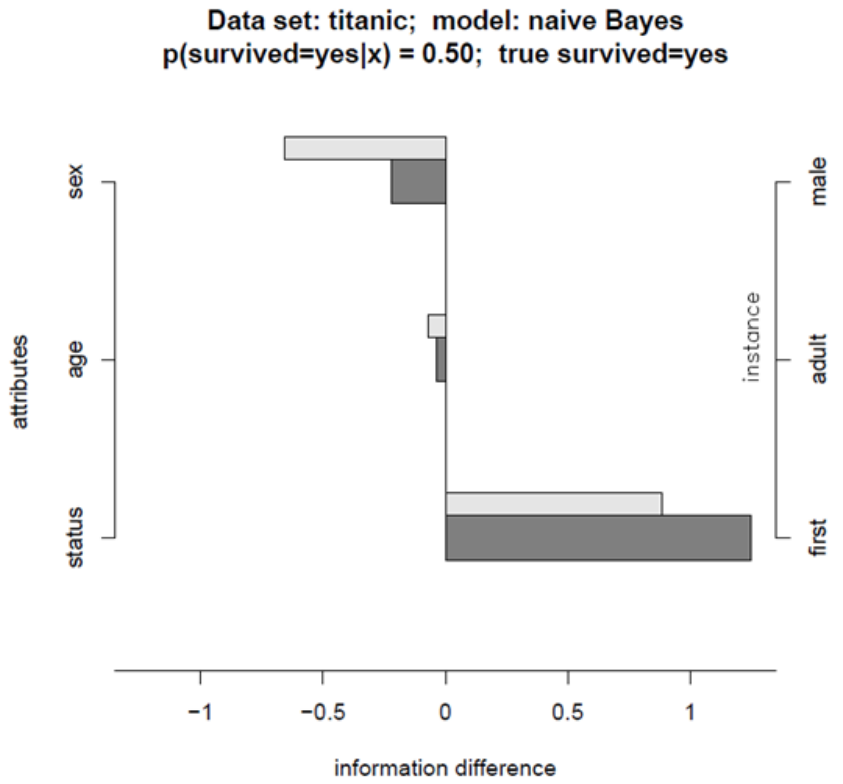
- the overall picture of a problem the model conveys
  - this is what knowledge extractors are interested in
  - presentation format: overall importance of each feature,
    but also rules, trees
- model-based



The method EXPLAIN

- "hide" one attribute at a time
- estimate contribution of attribute from

$$p(y_k|x) - p_{S\setminus\{i\}}(y_k|x)$$



Robnik-Sikonja, M., & Kononenko, I. (2008). Explaining classifications for individual instances. *IEEE Transactions on Knowledge and Data Engineering,*, *20*(5), 589-600.

Explaining EXPLAIN

- assume an instance (**x**, y); components of **x** are values of attributes A_i
- for a new instance **x**, we want to know what role each attribute's value play in the prediction model f, i.e. to what extend it contributed to the classification f(**x**)
- for that purpose
  - we compute f(**x** \ A_i), the model's prediction for **x** without the knowledge of the event $A_i = a_k$ (marginal prediction)
  - we comparing f(**x**) and f(**x** \ A_i) to assess importance of $A_i = a_k$
  - the larger the the difference the more important the role of $A_i=a_k$ in the model
- f(**x**) and f(**x** \ A_i) are source of explanations

Evaluation of prediction differences

- how to evaluate $f(\mathbf{x}) - f(\mathbf{x} \setminus A_i)$
- in classification, we take $f(\mathbf{x})$ in the form of probability

1. **difference of probabilities**

   $probDiff_i(y|\mathbf{x}) = p(y|\mathbf{x}) - p(y|\mathbf{x} \setminus A_i)$

2. **information gain** (Shannon, 1948)

   $infGain_i(y|\mathbf{x}) = \log_2 p(y|\mathbf{x}) - \log_2 p(y|\mathbf{x} \setminus A_i)$

3. **weight of evidence** also log odds ratio (Good, 1950)

   $odds(z) = p(z) / (1 - p(z))$

   $WE_i(y|\mathbf{x}) = \log_2 odds(y|\mathbf{x}) - \log_2 odds(y|\mathbf{x} \setminus A_i)$

Implementation

- $p(y|\mathbf{x})$: classify $\mathbf{x}$ with the model
- $p(y|\mathbf{x}\setminus A_i)$ – simmulate lack of knowledge of A_i in the model
  - replace with special NA value: good for some, mostly bad, left to the mercy of model's internal mechanism
  - average prediction across perturbations of A_i

    $p(y|\mathbf{x}\setminus A_i) = \sum_a p(A_i=a_s)\, p(y|\mathbf{x} \leftarrow A_i = a_s)$

    - use discretization for numeric attributes
    - use Laplace correction for probability estimation
  - we could build a separate model for each $p(y|\mathbf{x}\setminus A_i)$

Weaknes of EXPLAIN

- "hide" one attribute at a time
- estimate contribution of attribute from

$$p(y_k|x) - p_{S\setminus\{i\}}(y_k|x)$$

- weakness: if there are redundant ways to express concept, credit is not assigned
- example:

  $C = A_1 \vee A_2A_3$

  explanation for instance ($A_1=A_2=A_3=1$)

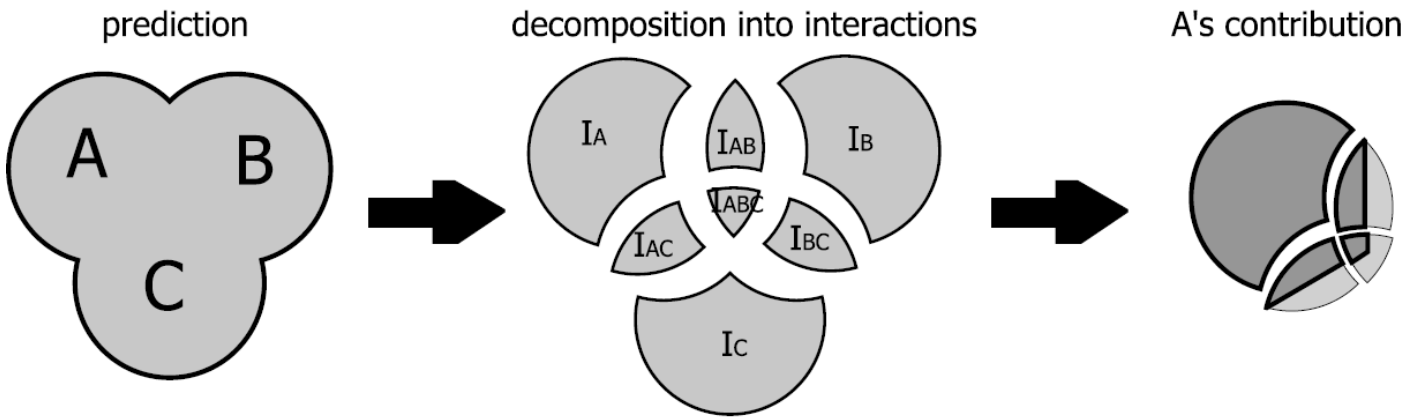
University of Ljubljana
Faculty of Computer and Information Science

The method IME

- (Interactions-based Method for Explanation)
- “hide” any subset of attributes at a time (2^a subsets!)
- the source of explanations is the difference in prediction using a subset of features Q and an empty set of features {}

$$\Delta_Q = h(x_Q) - h(x_{\{\}})$$

- the feature gets some credit for standalone contributions and for contributions in interactions



IME: sum over all subsets

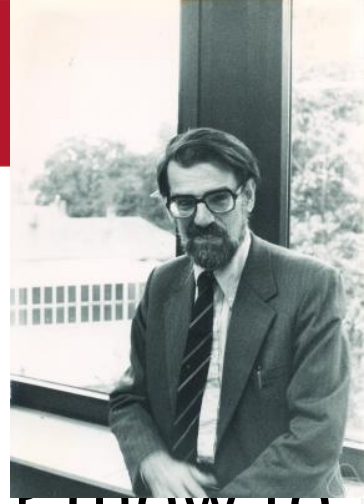
- the contributions are

$$\pi_i = \sum_{Q \subseteq \{1,2,\ldots,a\} - \{i\}} \frac{1}{a \binom{a-1}{a-|Q|-1}} (\Delta_{Q \cup \{i\}} - \Delta_Q)$$

Štrumbelj, E., Kononenko, I. & Robnik-Šikonja, M., Explaining instance classifications with interactions of subsets of feature values. *Data & Knowledge Engineering*, Oct. 2009, 68(10):886-904

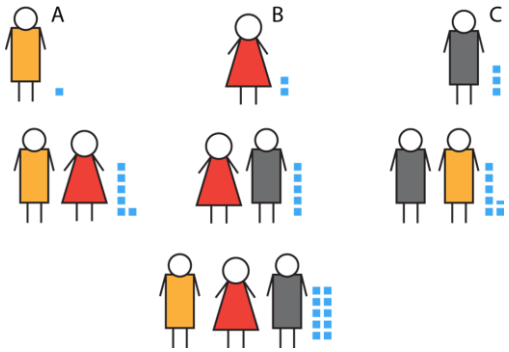
University of Ljubljana
Faculty of Computer and Information Science

Game theory analogy



- coalitional game of *a* players (attributes)
- players form coalitions (i.e. interactions)
- how to distribute the payout to the members of a coalition? (how to assign the credit for prediction)
- The Shapley value is the unique payoff vector that is
  - efficient (exactly splits payoff value),
  - symmetric (equal payments to equivalent players)
  - additive (overall credit is a sum of participating in coalitions), and
  - assigns zero payoffs to dummy players (no contribution to any coalition).

Shapley value



$$Sh_i(v) = \sum_{S \subseteq N \setminus \{i\}, s=|S|} \frac{(n-s-1)!s!}{n!}(v(S \cup \{i\}) - v(S)), \quad i = 1, \ldots, n.$$

$$\pi_i = \sum_{Q \subseteq \{1,2,\ldots,a\} - \{i\}} \frac{1}{a\binom{a-1}{a-|Q|-1}}(\Delta_{Q \cup \{i\}} - \Delta_Q)$$

- Shapley value can be efficiently approximated

University of Ljubljana
Faculty of Computer and Information Science

Solution for IME: sampling

- Shapley value can expressed in an alternative formulation
- $\pi(a)$is the set of all ordered permutations of *a*
- $Pre^i(O)$ is the set of players which are predecessors of player *i* in the order $O \in \pi(a)$

$$\varphi_i(k,x) = \frac{1}{a!} \sum_{\mathcal{O}\in\pi(a)} \left(\Delta(Pre^i(\mathcal{O}) \cup \{i\})(k,x) - \Delta(Pre^i(\mathcal{O}))(k,x)\right) =$$

$$= \frac{1}{a!} \sum_{\mathcal{O}\in\pi(a)} \left(p_{Pre^i(\mathcal{O})\cup\{i\}}(y_k|x) - p_{Pre^i(\mathcal{O})}(y_k|x)\right),$$

- smart sampling over subsets of attributes
- computationally feasible approach

Štrumbelj, E., & Kononenko, I. (2010). An efficient explanation of individual classifications using game theory. Journal of Machine Learning Research, 11, 1-18

IME algorithm

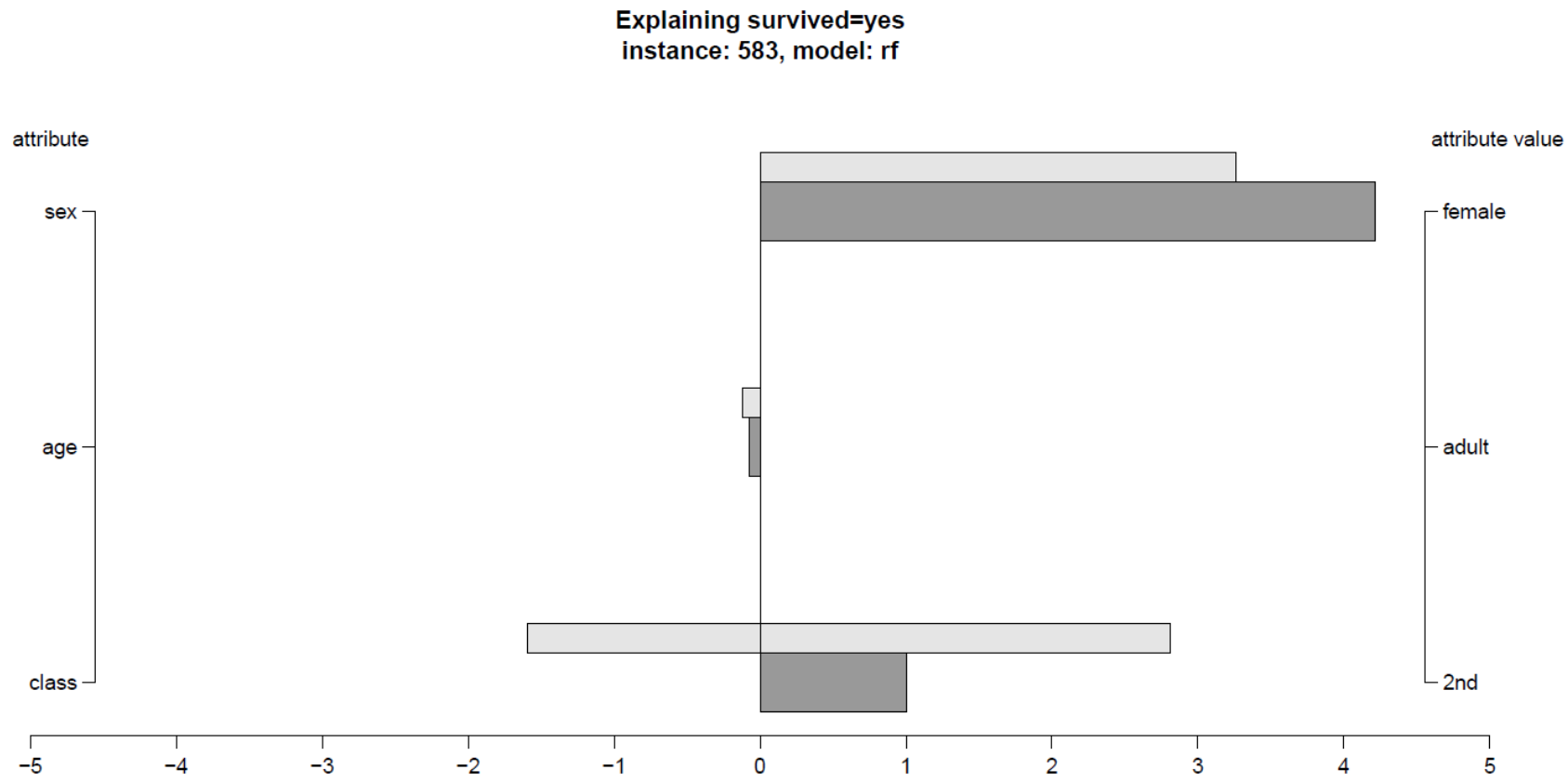
Algorithm 1 Approximating the contribution of the i-th feature's value, φ_i, for instance $x \in \mathcal{A}$.

determine m, the desired number of samples
$\varphi_i \leftarrow 0$
for $j = 1$ to m **do**
  choose a random permutation of features $O \in \pi(N)$
  choose a random instance $y \in \mathcal{A}$
  $v_1 \leftarrow f(\tau(x, y, Pre^i(O) \cup \{i\}))$
  $v_2 \leftarrow f(\tau(x, y, Pre^i(O)))$
  $\varphi_i \leftarrow \varphi_i + (v_1 - v_2)$
end for
$\varphi_i \leftarrow \frac{\varphi_i}{m}$

- by measuring the variance of contributions, we can determine the necessary number of samples for each attribute

Visualization of explanations

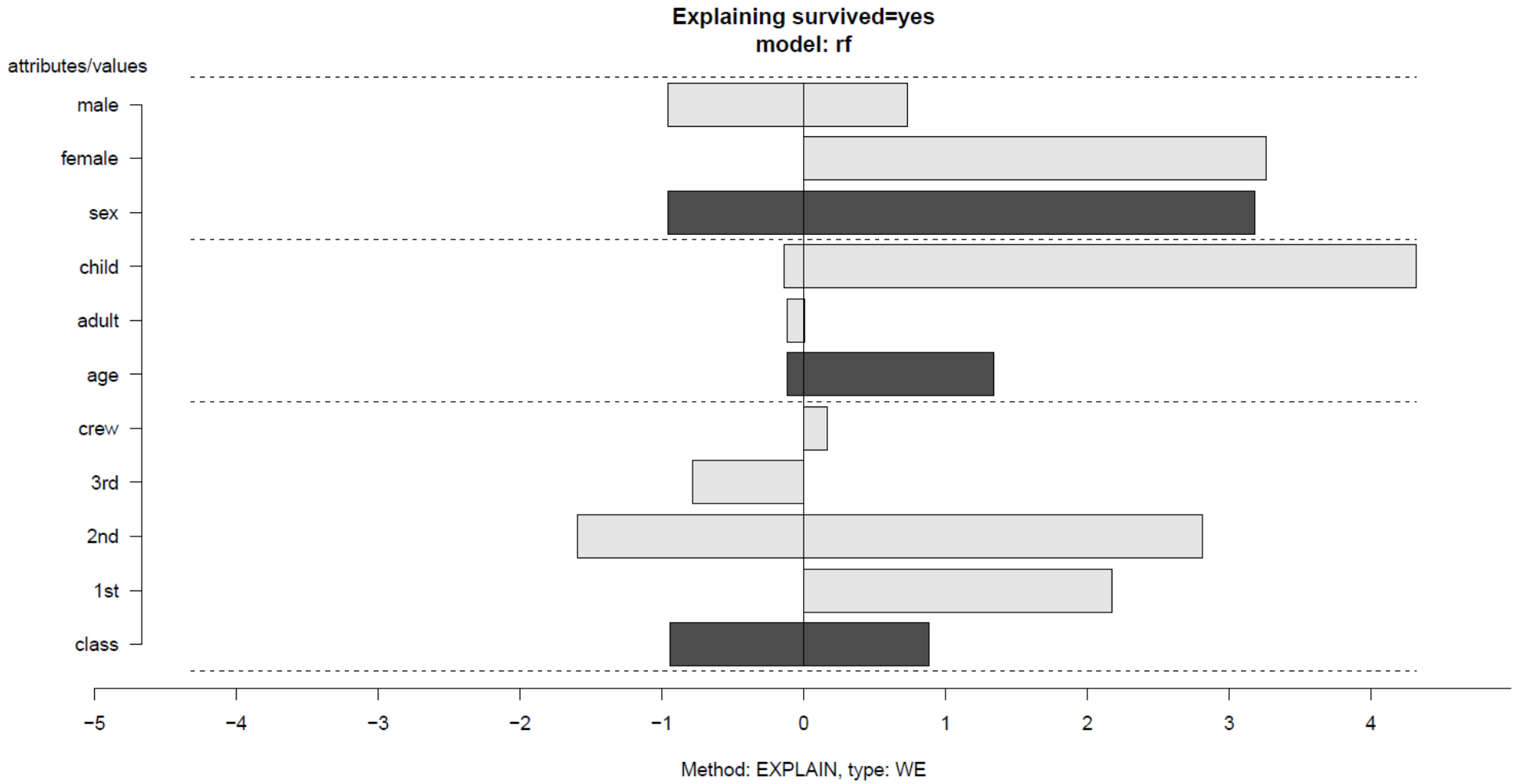
- instance-level explanation on Titanic data set



Robnik-Šikonja, M. (2015), ExplainPrediction: Explanation of Predictions for Classification and Regression.
R package version 1.3.0. http://cran.r-project.org/package=ExplainPrediction

Visualization of explanations

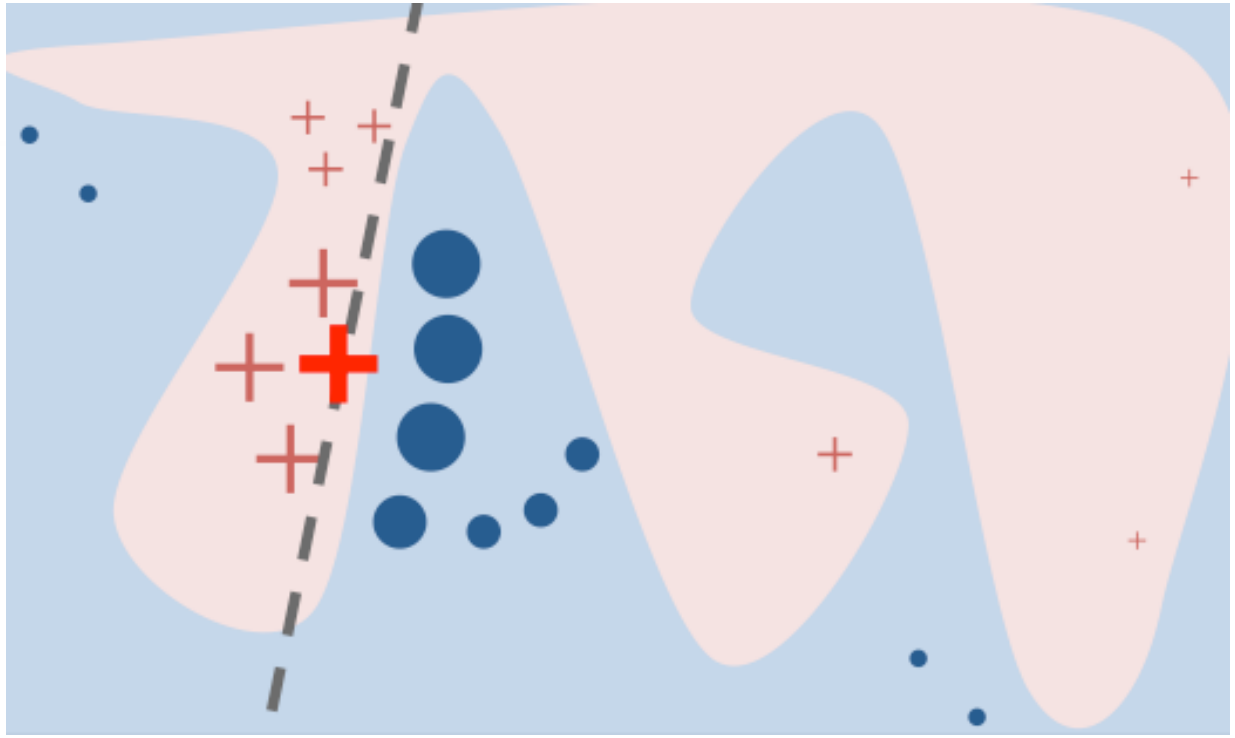
- model-level explanation on Titanic data set



University of Ljubljana
Faculty of Computer and Information Science

LIME explanation method

- Local Interpretable Model-agnostic Explanations)
- perturbations in the locality of an explained instance

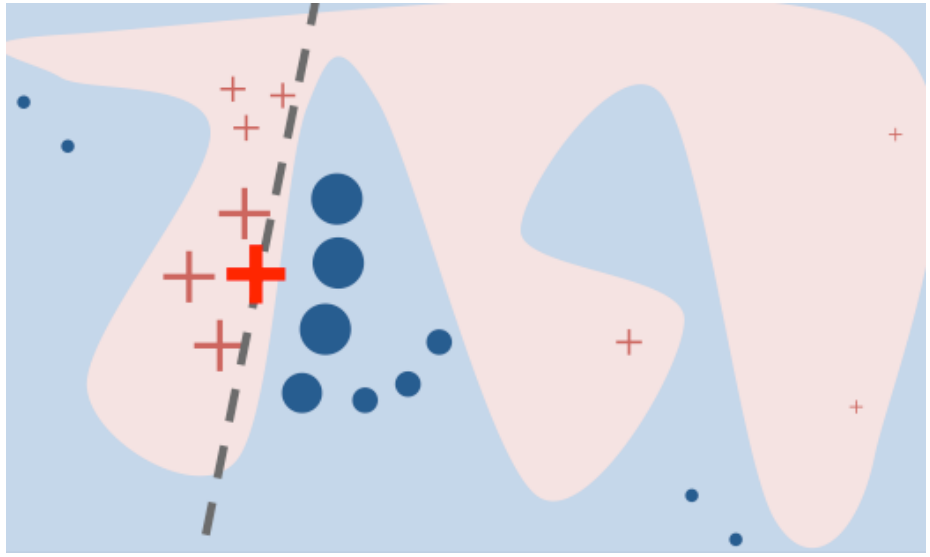


Ribeiro, M. T., Singh, S., & Guestrin, C. (2016). Why should i trust you?: Explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 1135-1144.

LIME explanation method

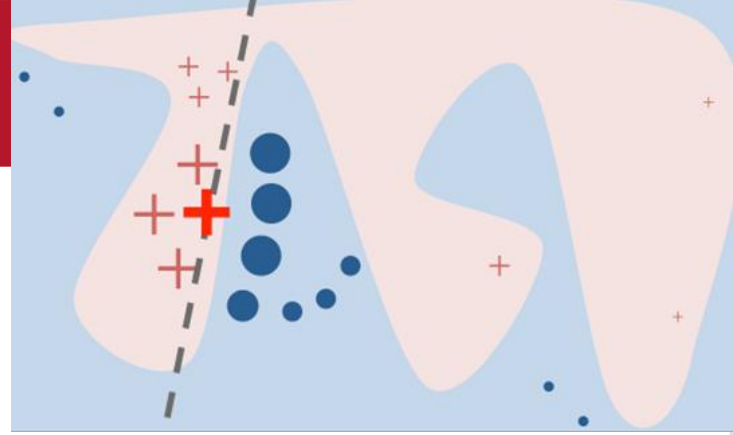
- optimize a trade-off between local fidelity of explanation and its interpretability

$$e(x) = \arg\min_{g \in G} L(f, g, \pi) + \Omega(g)$$



- L is a local fidelity function, f is a model to be explained, g is an interpretable local model g (i.e. linear model), $\pi(x, z)$ is proximity measure between the explained instance *x* and perturbed points *z* in its neighborhood, Ω is a model complexity measure

LIME details



- LIME samples around the explanation instance *x* to draw samples *z* weighted by the distance $\pi(\mathrm{x}, \mathrm{z})$
- samples *z* are used to training an interpretable model *g* (linear model)
- the squared loss measures local infidelity
- number of non-zero weights is complexity
- samples are weighted according to the Gaussian distribution of the distance between *x* and *z*

LIME strengths and weaknesses

- faster than IME
- works for many features, including text and images

- no guarantees that the explanations are faithful and stable
- neighborhood based: a curse of dimensionality
- may not detect interactions due to (too) simple interpretable local model (linear model)



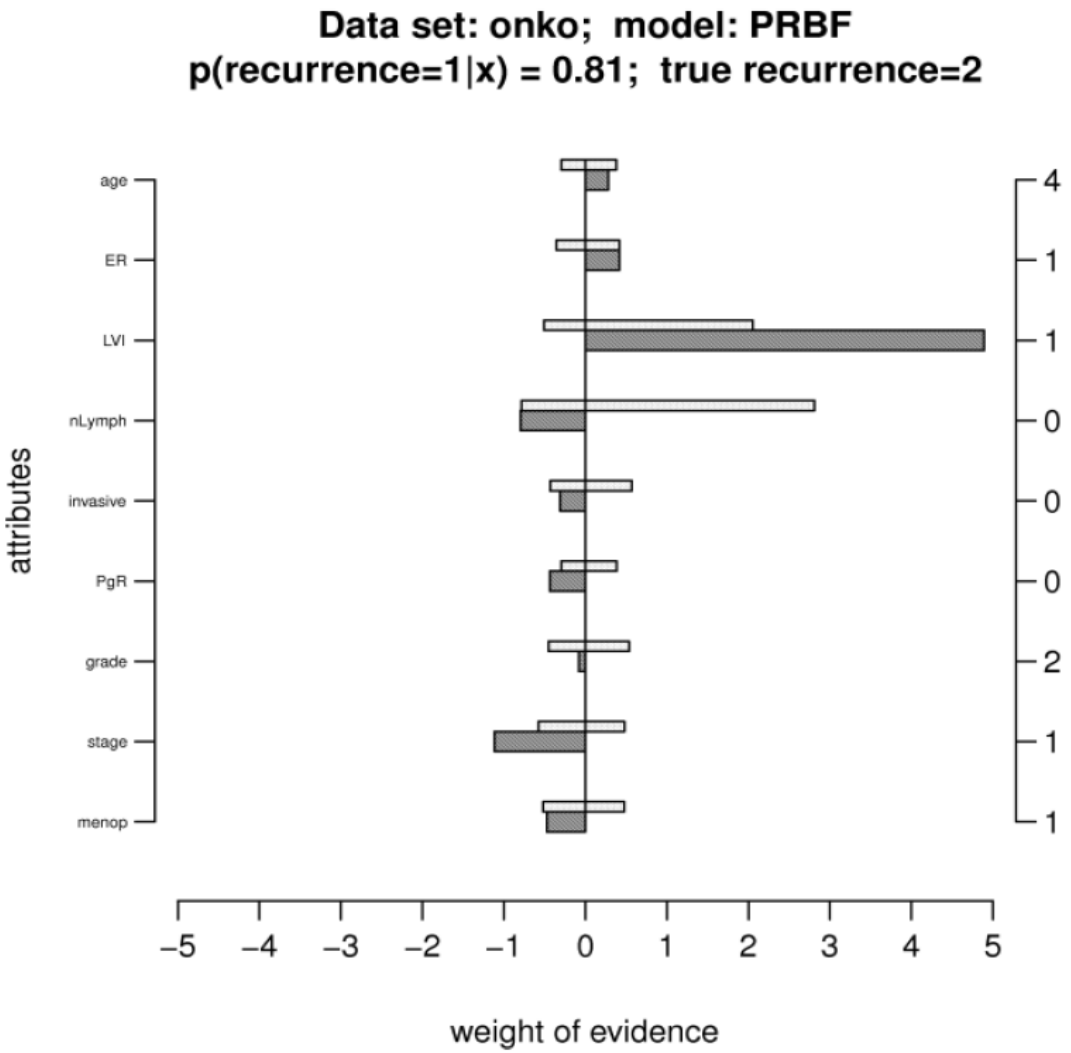
SHAP

- SHapley Additive exPlanation
- unification of several explanation methods, including IME and LIME
- KernelSHAP: based on Shapley values which are estimated using a LIME style linear regression
- faster then IME but
- still uses linear model with all its strengths and weaknesses

Lundberg, S. M., & Lee, S. I. (2017). A unified approach to interpreting model predictions. In Advances in Neural Information Processing Systems (pp. 4765-4774).



Use case: breast cancer recurrence



Cancer recurrence within 10 years

menop binary feature indicating menopausal status

stage tumor stage 1: less than 20mm, 2: between 20mm and 50mm, 3: over 50mm

grade tumor grade 1: good, 2: medium, 3: poor, 4: not applicable, 9: not determined

histType histological type of the tumor 1: ductal, 2: lobular, 3: other

PgR level of progesterone receptors in tumor (in fmol per mg of protein) 0: less than 10, 1: more than 10, 9: unknown

invasive invasiveness of the tumor 0: no, 1: invades the skin, 2: the mamilla, 3: skin and mamilla, 4: wall or muscle

nLymph number of involved lymph nodes 0: 0, 1: between 1 and 3, 2: between 4 and 9, 3: 10 or more

famHist medical history 0: no cancer, 1: 1st generation breast, ovarian or prostate cancer 2: 2nd generation breast, ovarian or prostate cancer, 3: unknown gynecological cancer 4: colon or pancreas cancer, 5: other or unknown cancers, 9: not determined

LVI binary feature indicating lymphatic or vascular invasion

ER level of estrogen receptors in tumor (in fmol per mg of protein) 1: less than 5, 2: 5 to 10, 3: 10 to 30, 4: more than 30, 9: not determined

maxNode diameter of the largest removed lymph node 1: less than 15mm, 2: between 15 and 20mm, 3: more than 20mm

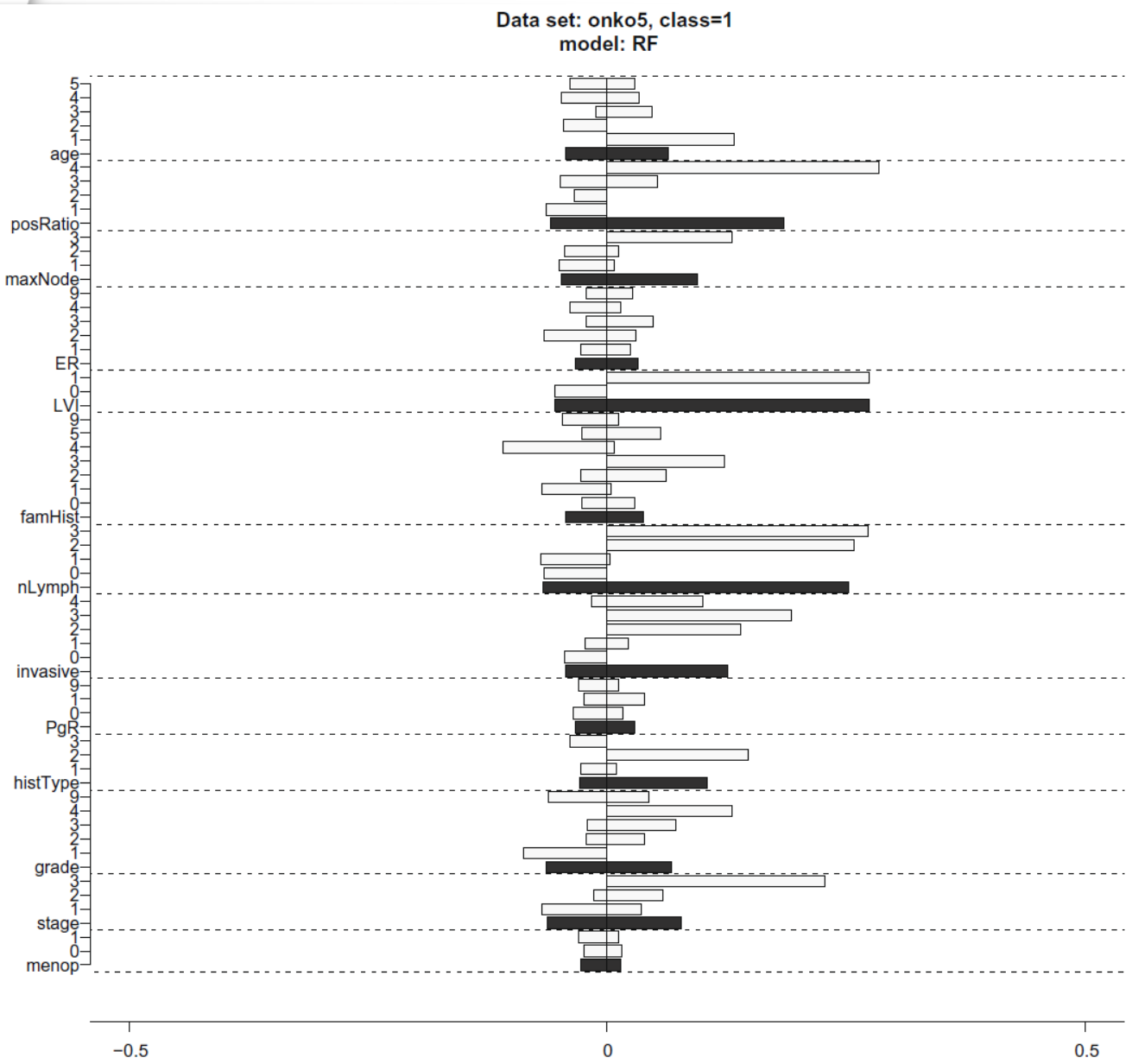
posRatio ratio between involved and total lymph nodes removed 1: 0, 2: less that 10%, 3: between 10% and 30%, 4: over 30%

age patient age group 1: under 40, 2: 40-50, 3: 50-60, 4: 60-70, 5: over 70 years

Robnik-Šikonja, M., Kononenko, I., & Štrumbelj, E. (2012). Quality of classification explanations with PRBF. *Neurocomputing*, *96*, 37-46.

University of Ljubljana
Faculty of Computer and Information Science

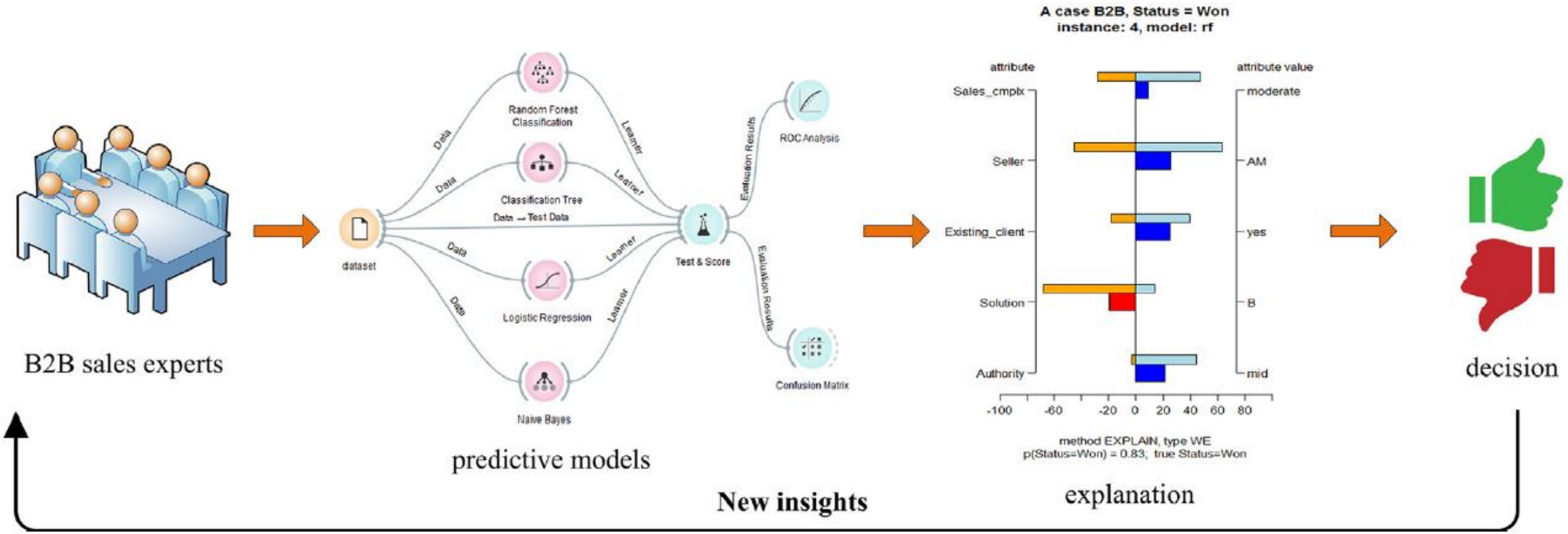
Use case: breast cancer recurrence



University of Ljubljana
Faculty of Computer and Information Science

Use case: B2B sales forecasting

- Goals: improve understanding of factors influencing the outcome and improve the sales performance



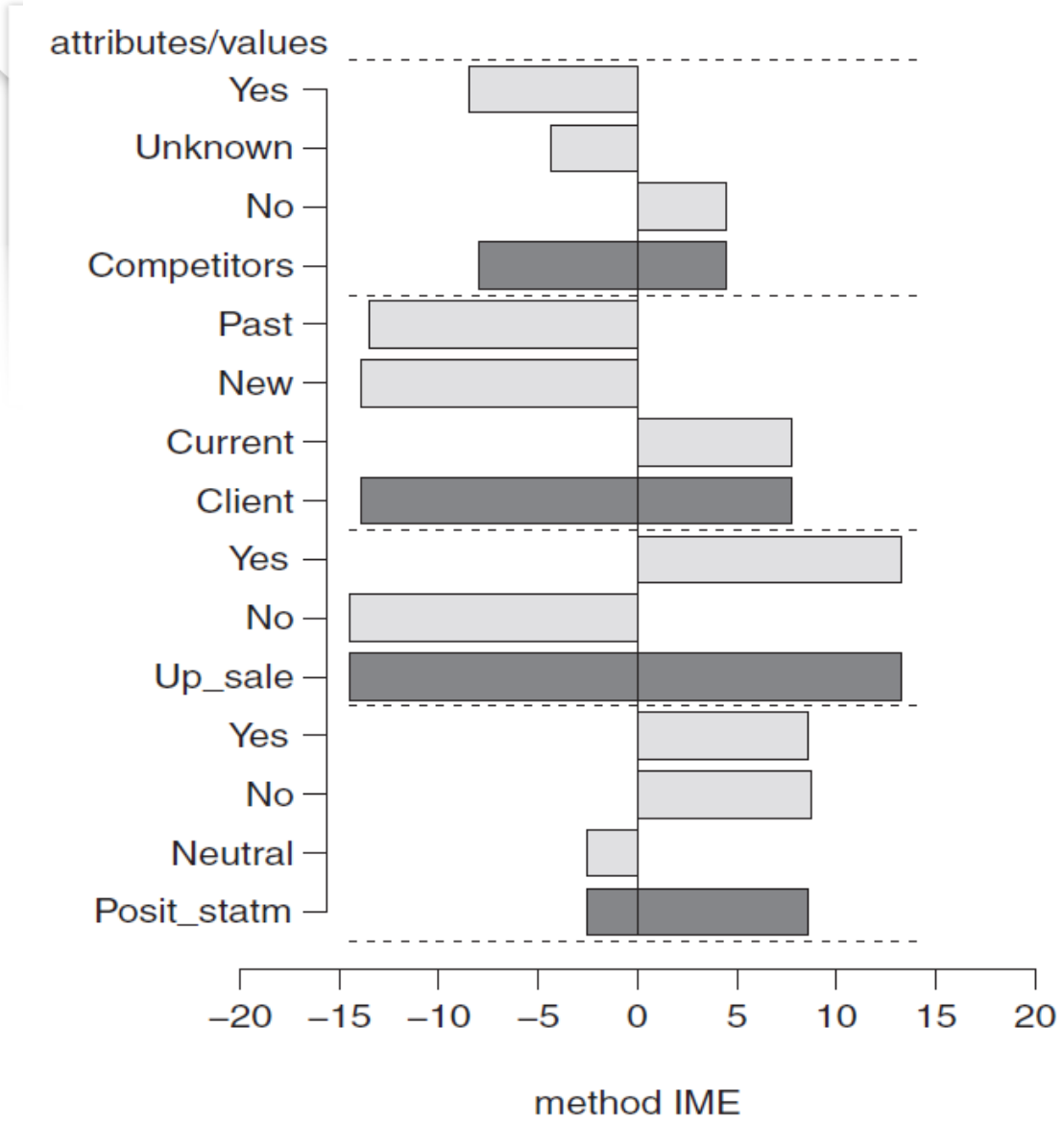
Bohanec, M., Borštnar, M. K., & Robnik-Sikonja, M. (2017). Explaining machine learning models in sales predictions. *Expert Systems with Applications*, *71*, 416-428.

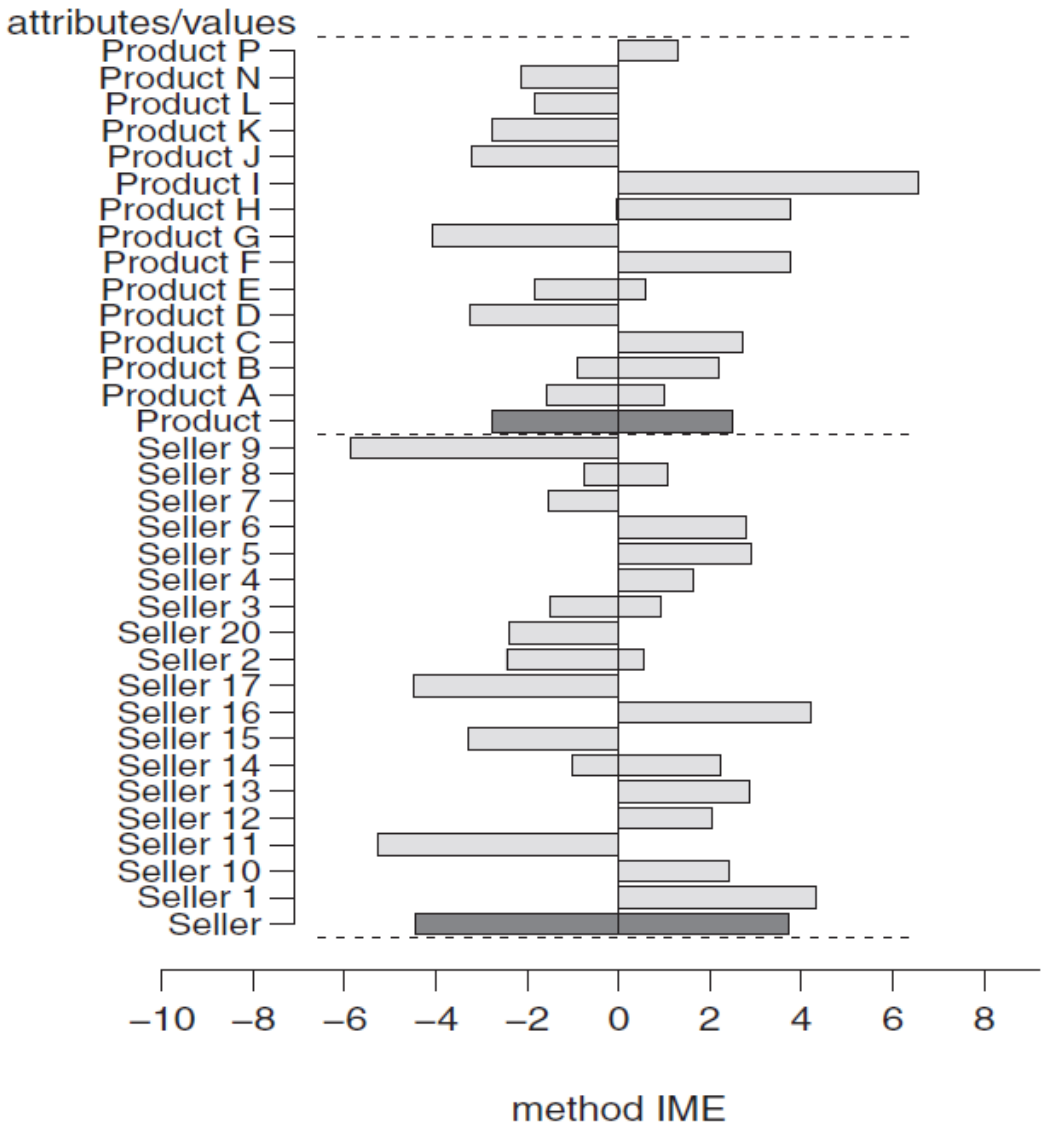
B2B sales attributes

Attribute	Description	Values
Authority	Authority level at a client side	Low, mid, high
Product	Offered product	e.g. A, B, C, etc.
Seller	Seller's name	Seller's name
Competitors	Do we have competitors?	No, yes, unknown
Company size	Size of a company	Big, mid, small
Purchasing department	Is the purchasing department involved?	No, yes, unknown
Partnership	Selling in partnership?	No, yes
Budget allocated	Did the client reserve the budget?	No, yes, unknown
Formal tender	Is a tendering procedure required?	No, yes
RFI	Did we get request for information?	No, yes
RFP	Did we get request for proposal?	No, yes
Growth	Growth of a client?	Growth, stable, etc.
Positive statements	Positive attitude expressed?	No, yes, neutral
Source	Source of the opportunity	e.g. referral, web, etc.
Client	Type of a client	New, current, past
Cross sale	A different product to existing client?	No, yes
Scope clarity	Implementation scope defined?	Clear, few questions, etc.
Strategic deal	Does this deal have a strategic value?	Very important, etc.
Up sale	Increasing sales of existing products?	No, yes
Deal type	Type of a sale	Consulting, project, etc.
Needs defined	Is client clear in expressing the needs?	Info gathering, etc.
Attention to client	Attention to a client	First deal, normal, etc.
Status	An outcome of sales opportunity	Lost, won

University of Ljubljana
Faculty of Computer and Information Science

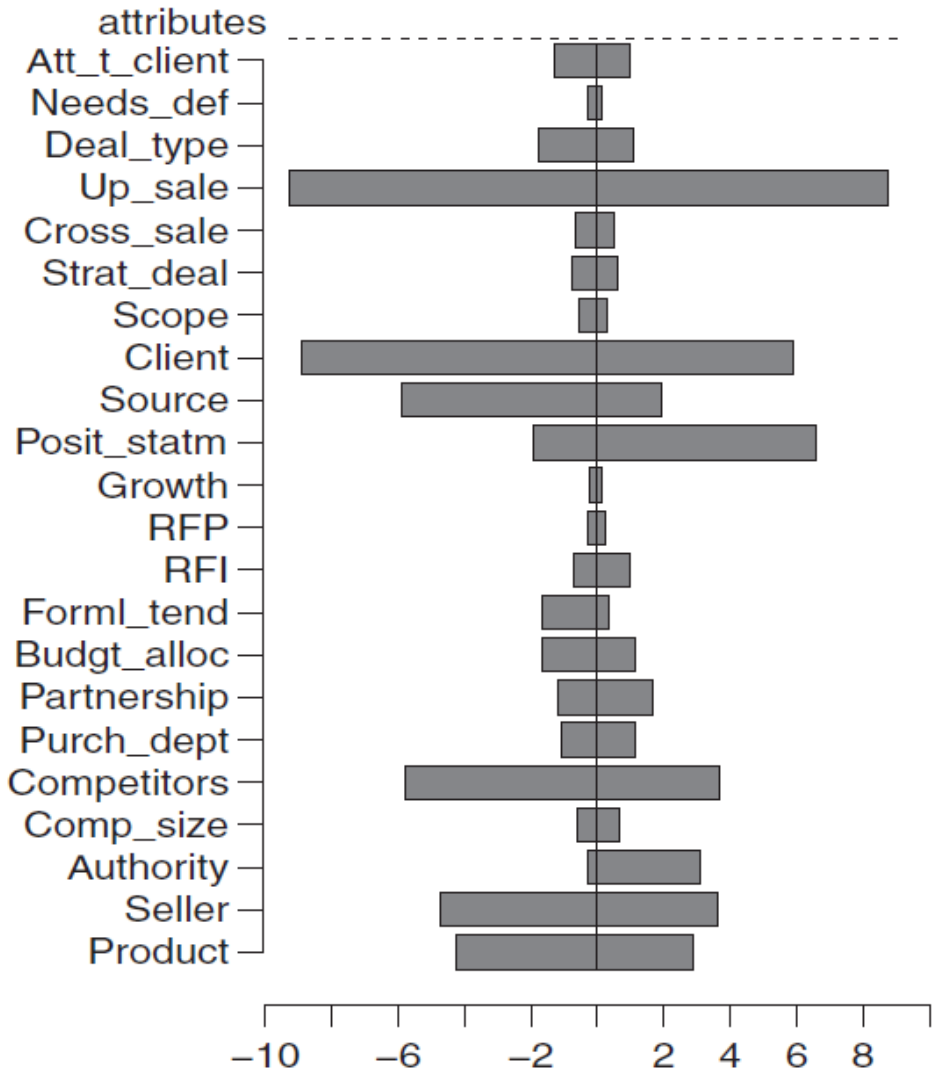
B2B sales: drill in



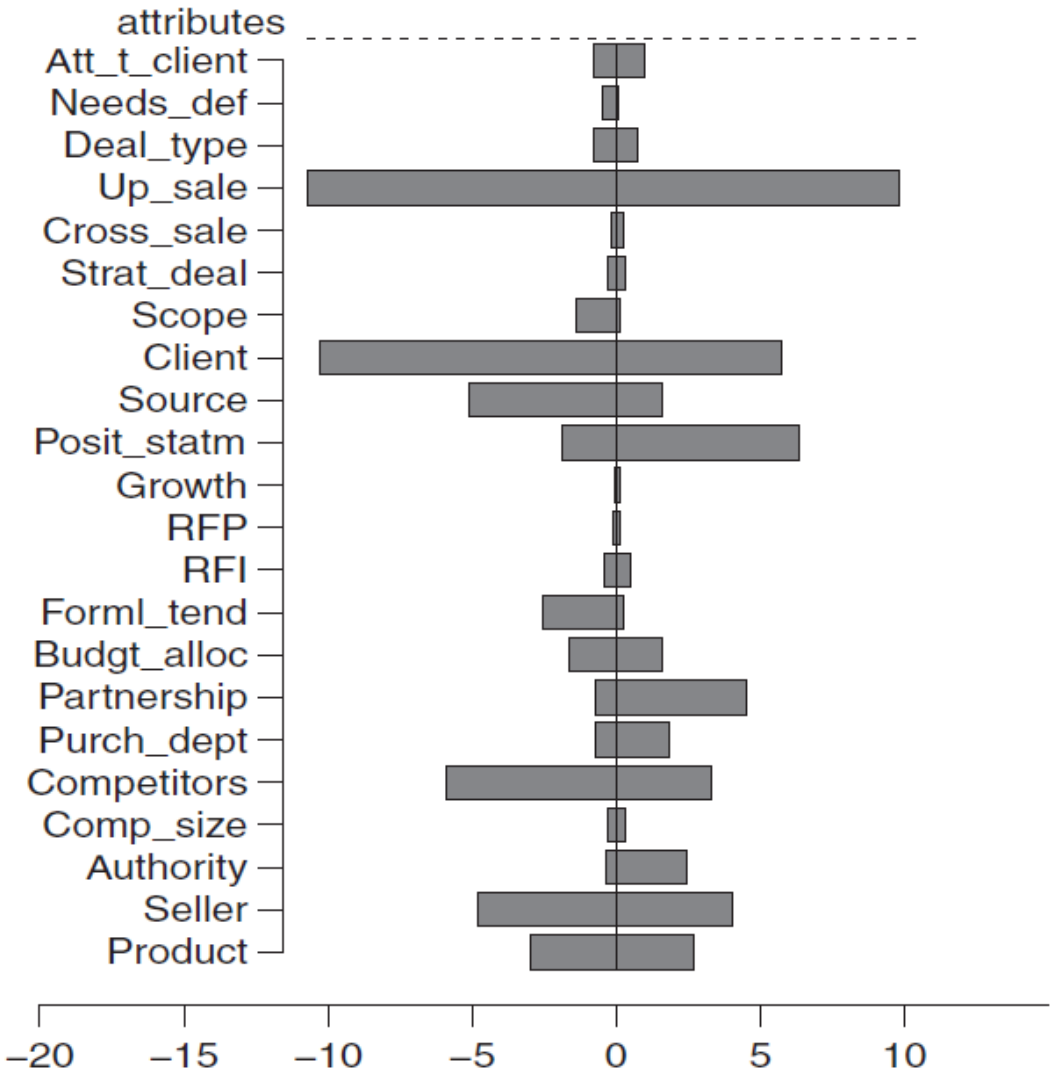


University of Ljubljana
Faculty of Computer and Information Science

B2B sales: EXPLAIN and IME



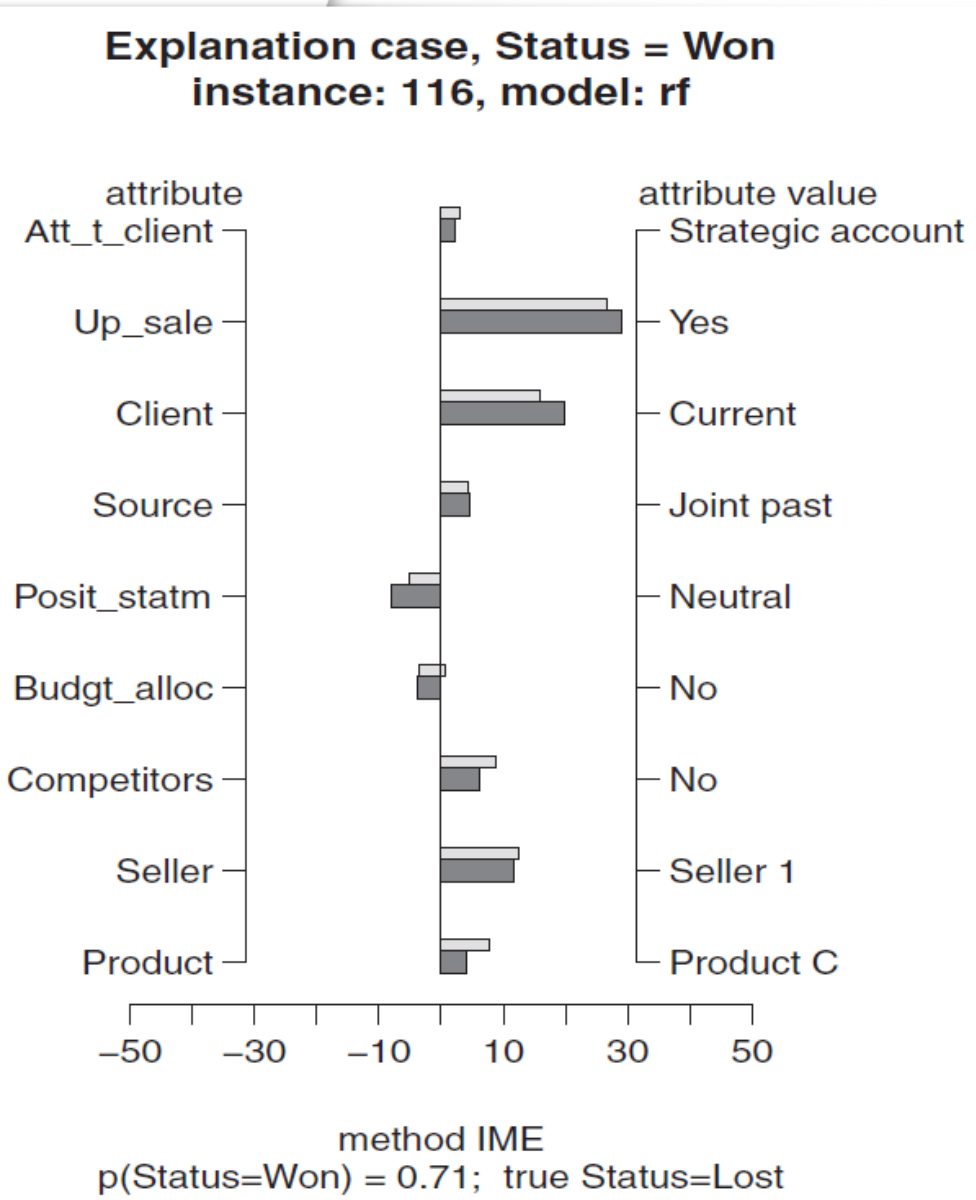
method EXPLAIN, type WE

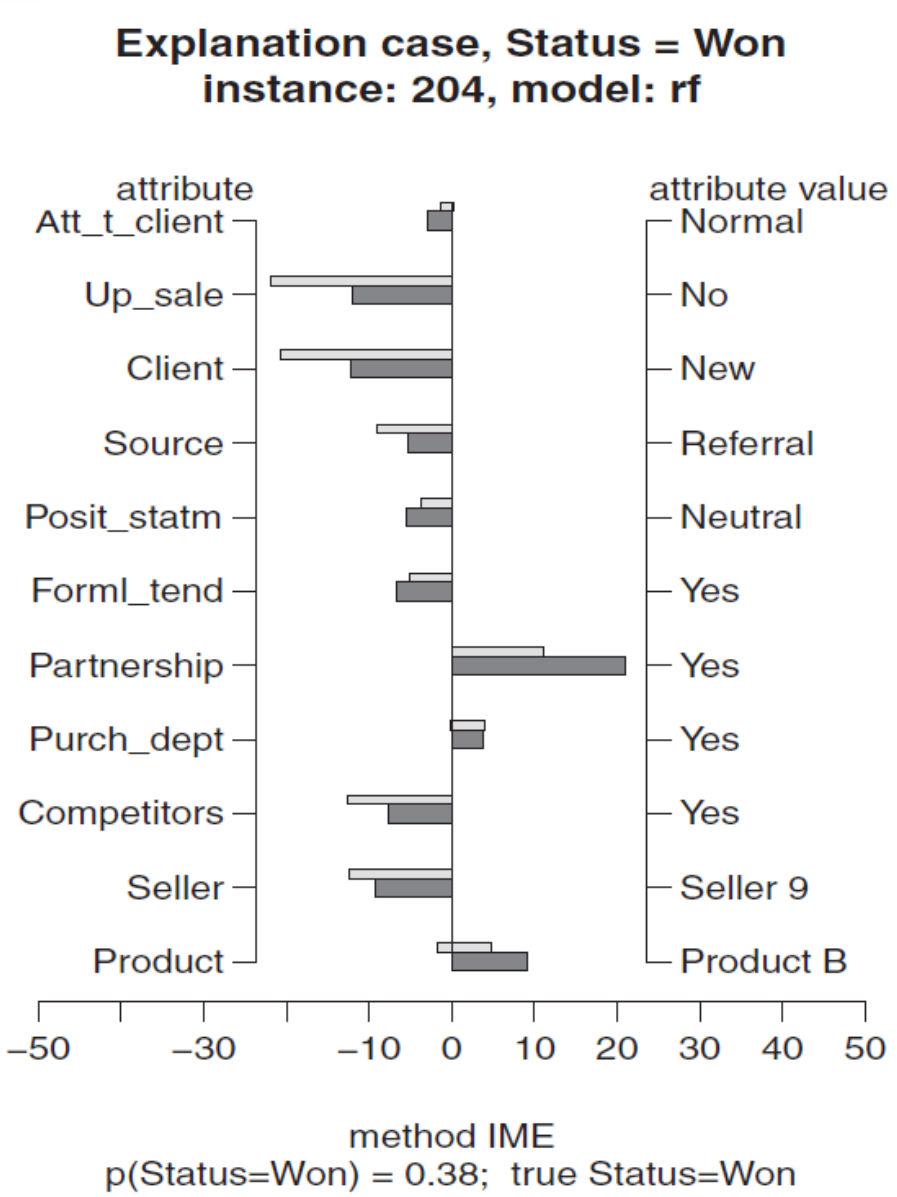


method IME

University of Ljubljana
Faculty of Computer and Information Science

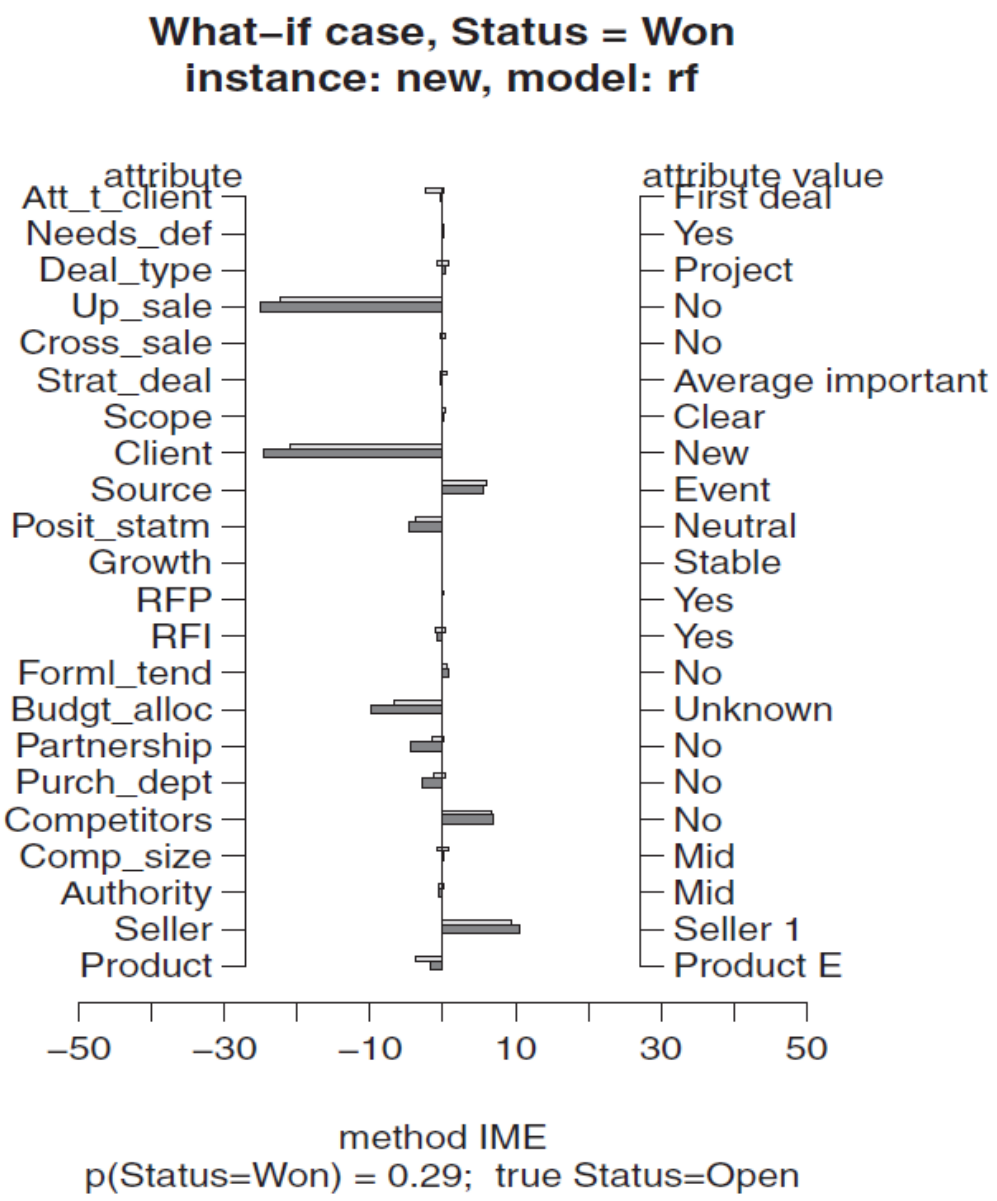
B2B sales: learning from errors





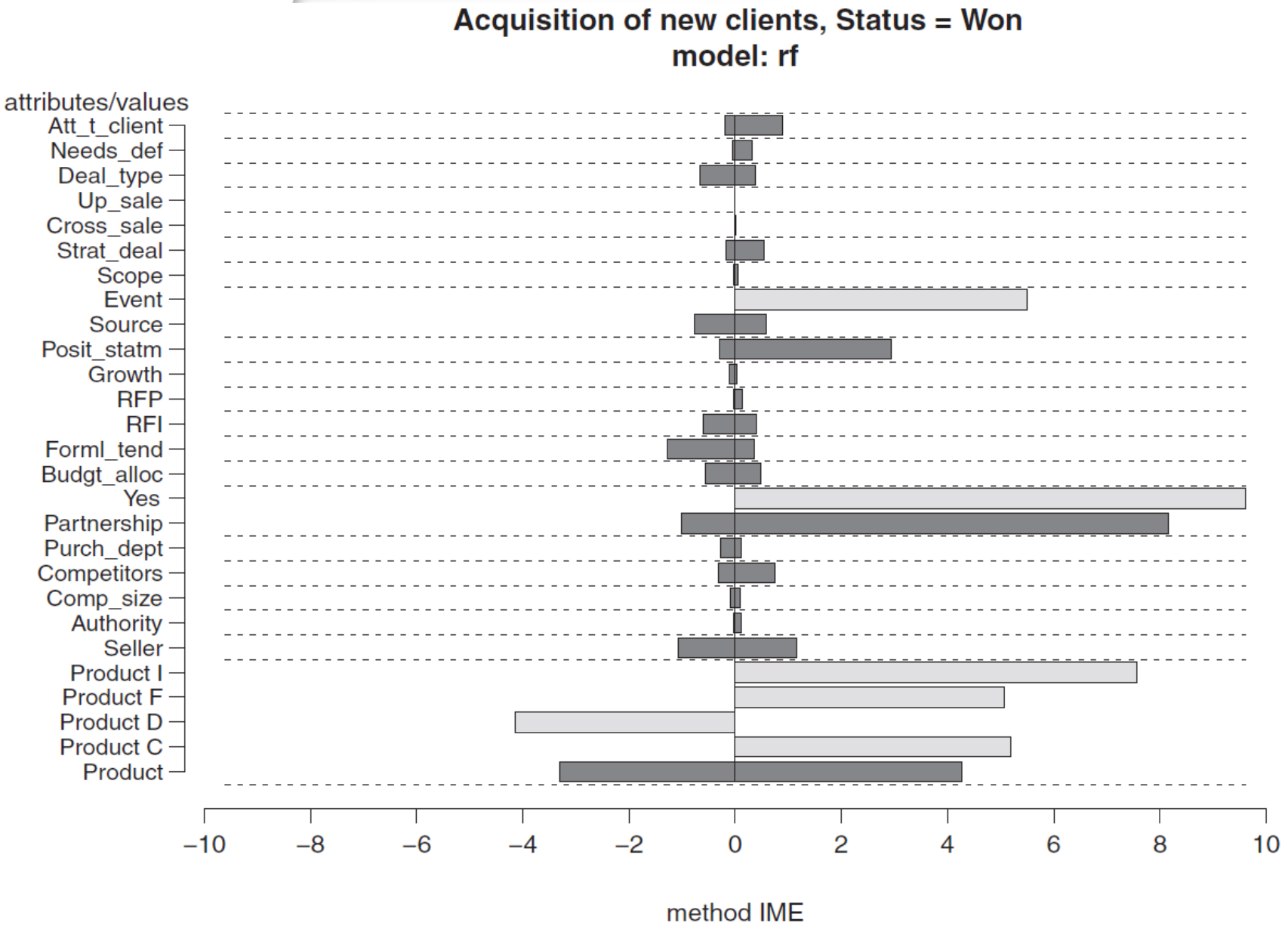
University of Ljubljana
Faculty of Computer and Information Science

B2B: what if

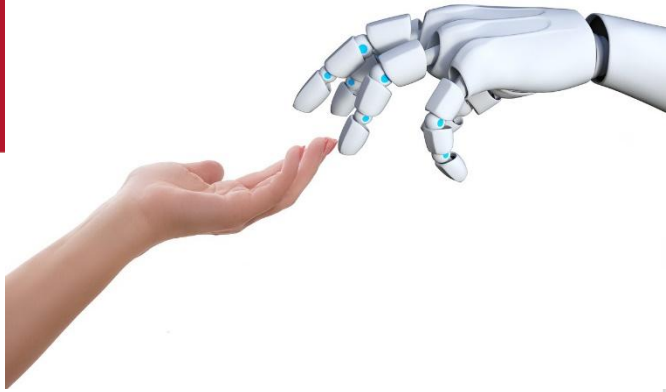


University of Ljubljana
Faculty of Computer and Information Science

B2B: change of distribution



University of Ljubljana
Faculty of Computer and Information Science



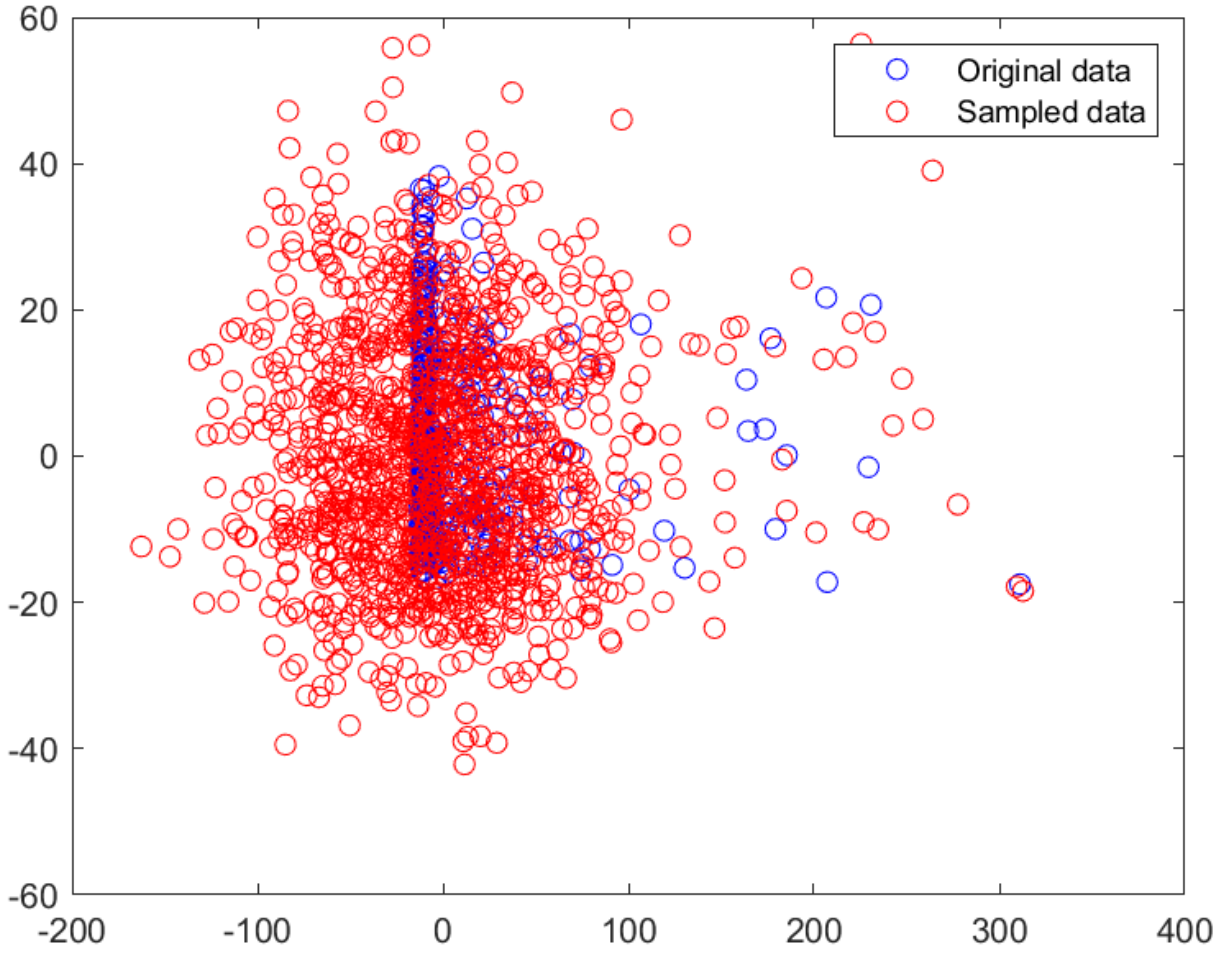
Lessons learned in B2B

- an effort needed to overcome the users' resistance
- human-in-the-loop is necessary to train, discuss, clean data, introduce explanations
- with an increased use, users gain trust in the methodology
- human mental models tend to be biased
- joint interactive approach beats both humans and ML models

University of Ljubljana
Faculty of Computer and Information Science

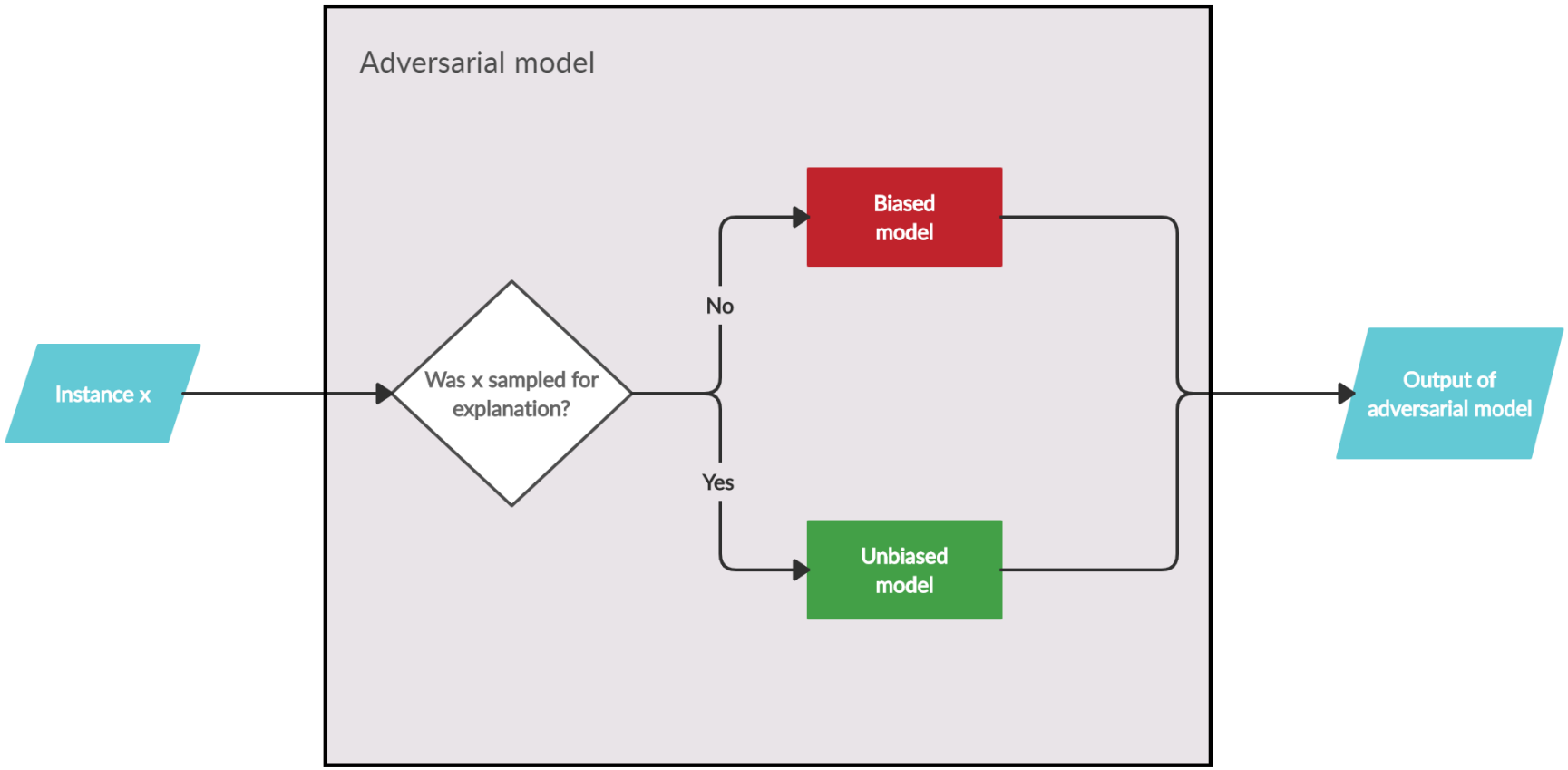
Attacks on explanations

- Poor sampling in explanation approaches makes them vulnerable
- Example: PCA based visualization of a part of the COMPAS dataset; the red dots were generated by LIME



University of Ljubljana
Faculty of Computer and Information Science

Dieselgate attacks on explanations



- Defence: better sampling

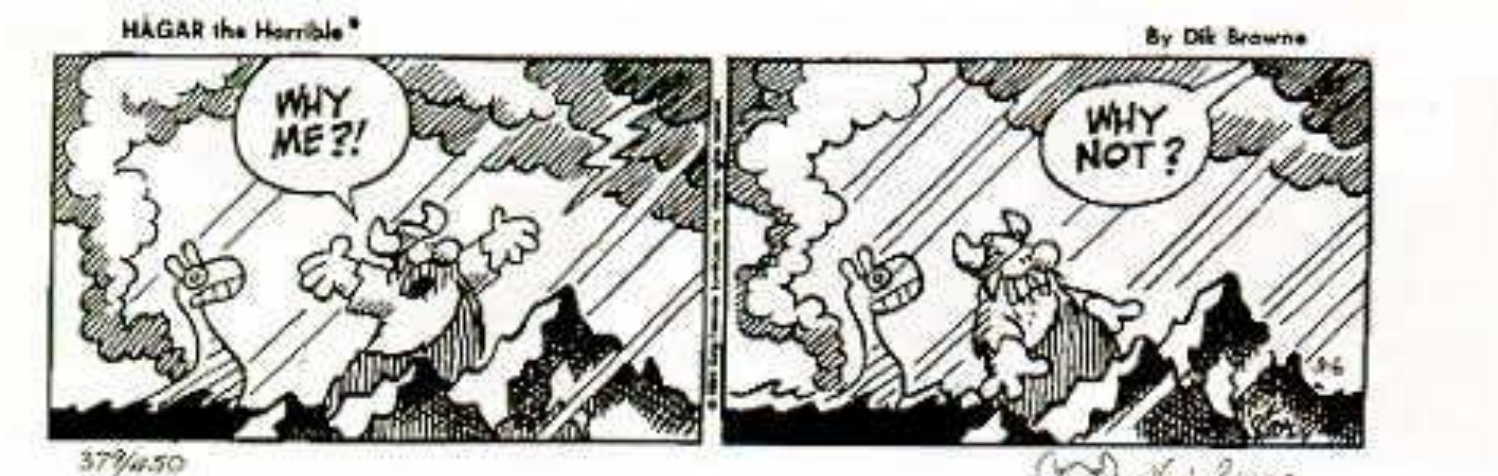
Domen Vreš and Marko Robnik-Šikonja, 2021. Better sampling in explanation methods can prevent dieselgate-like deception. *arXiv preprint arXiv:2101.11702*.

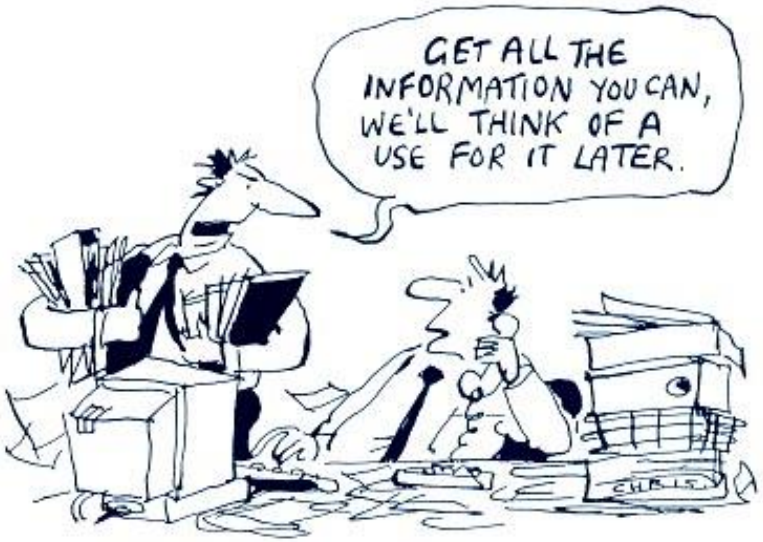
Opportunities

- better and more focused sampling
- better local explanation models
- interactions: detect and describe
- sequences: the order of attributes is important!
- images: decison areas, super-pixels
- better visualizations: human cognitive limitations
- explanations is also domain specific, we need explanation datasets

Conclusions

- many successful mechanistic explanation approaches, mostly for tabular classification problems
- LLMs are trained to explain their behavior for particular important problems
- lots of opportunities for improvements
- even human explanations are not necessarily comprehensible
- humans often explain by providing background or additional knowledge
- legal and practical need for explanations of ML models





University of Ljubljana, Faculty of Computer and Information Science

Inference and explanation of prediction models



Prof Dr Marko Robnik-Šikonja

Intelligent Systems, Edition 2025

Overview of topics



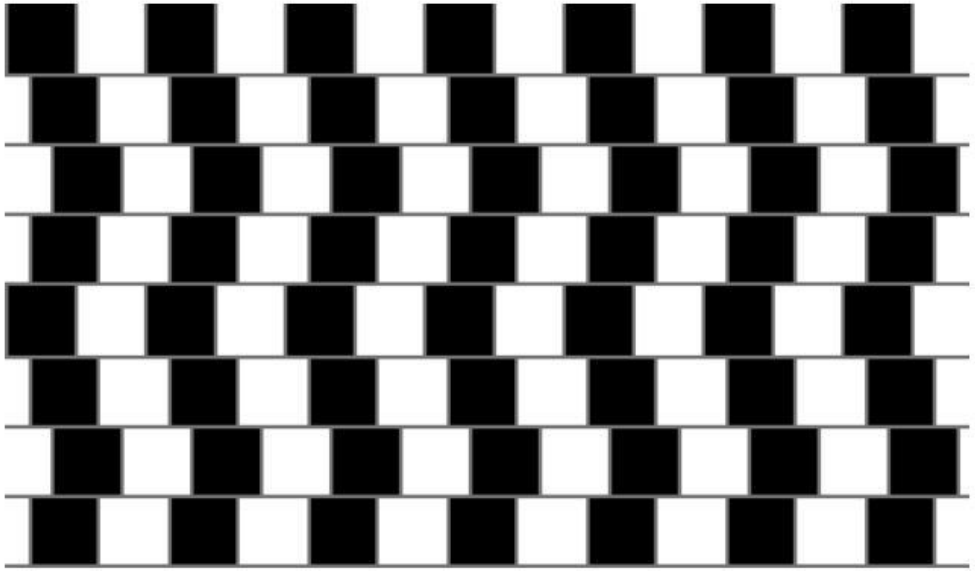
- Visualization and knowledge discovery.
- General methodology for explaining predictive models.
- Model level and instance level explanations, methods EXPLAIN and IME.

Visualization

- 1st rule of data mining: know your data.
- Therefore: visualizations, getting background data.
- Visualize: distributions of individual variables, their relations, etc.
- For high dimensional data sets one can use scaling, e.g. UMAP or t-SNE
- Clustering is useful in supervised tasks to get insight into the relation between predicted values Y and basic groups in the data. If unrelated, feature set might need amendments.

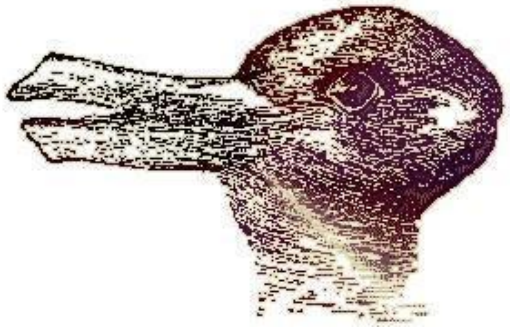
Visualizations

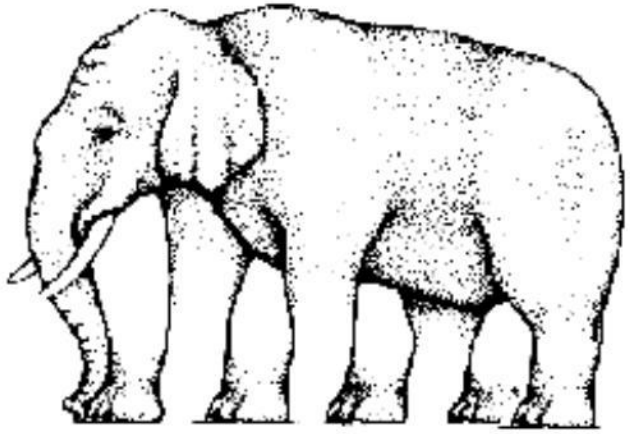
- Human visual perception has certain limitations:
  - we see what we want to see
  - we see what we see often
  - it is more difficult to notice unexpected patterns
- practice in detection of unknown
- use visualizations which expose “the unknown”



Are the horizontal lines parallel or do they slope?







How many legs does this elephant have?

Human pattern recognition

- We see inexistent patterns because we WANT to see them (we feel lost without them).



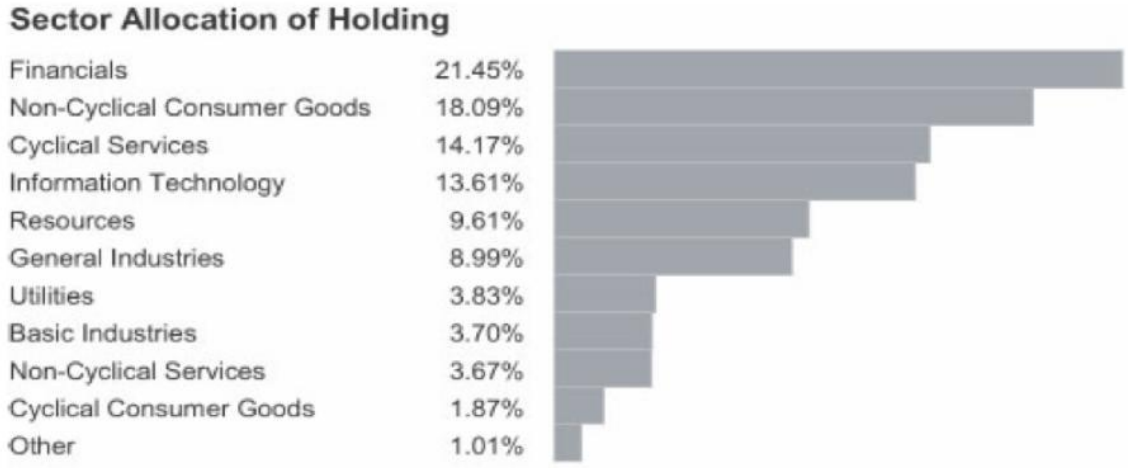


"The researchers found that when people were primed to feel out of control, they were more likely to see patterns where none exist." (**See a Pattern on Wall Street?,** *John Tierney*, *Science*)

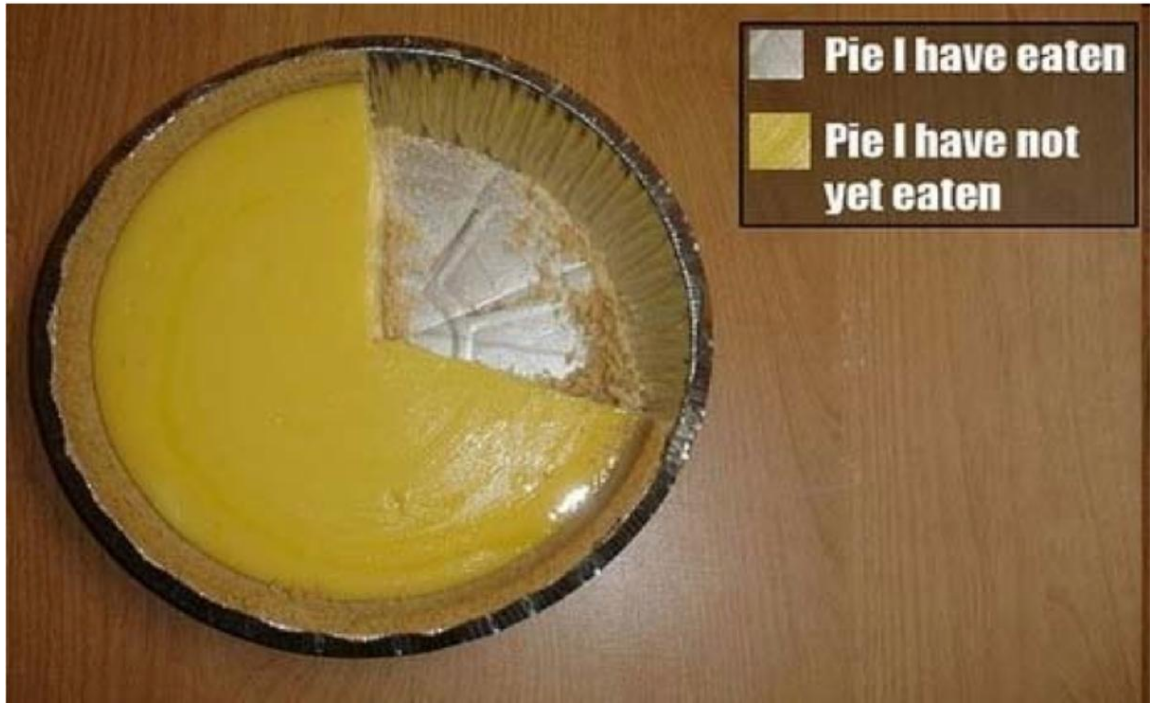
Facts about simple visualizations

- Pie charts are a bad choice: hard to read, similar colors, slope, legend is too far away
- Bar chart is much better



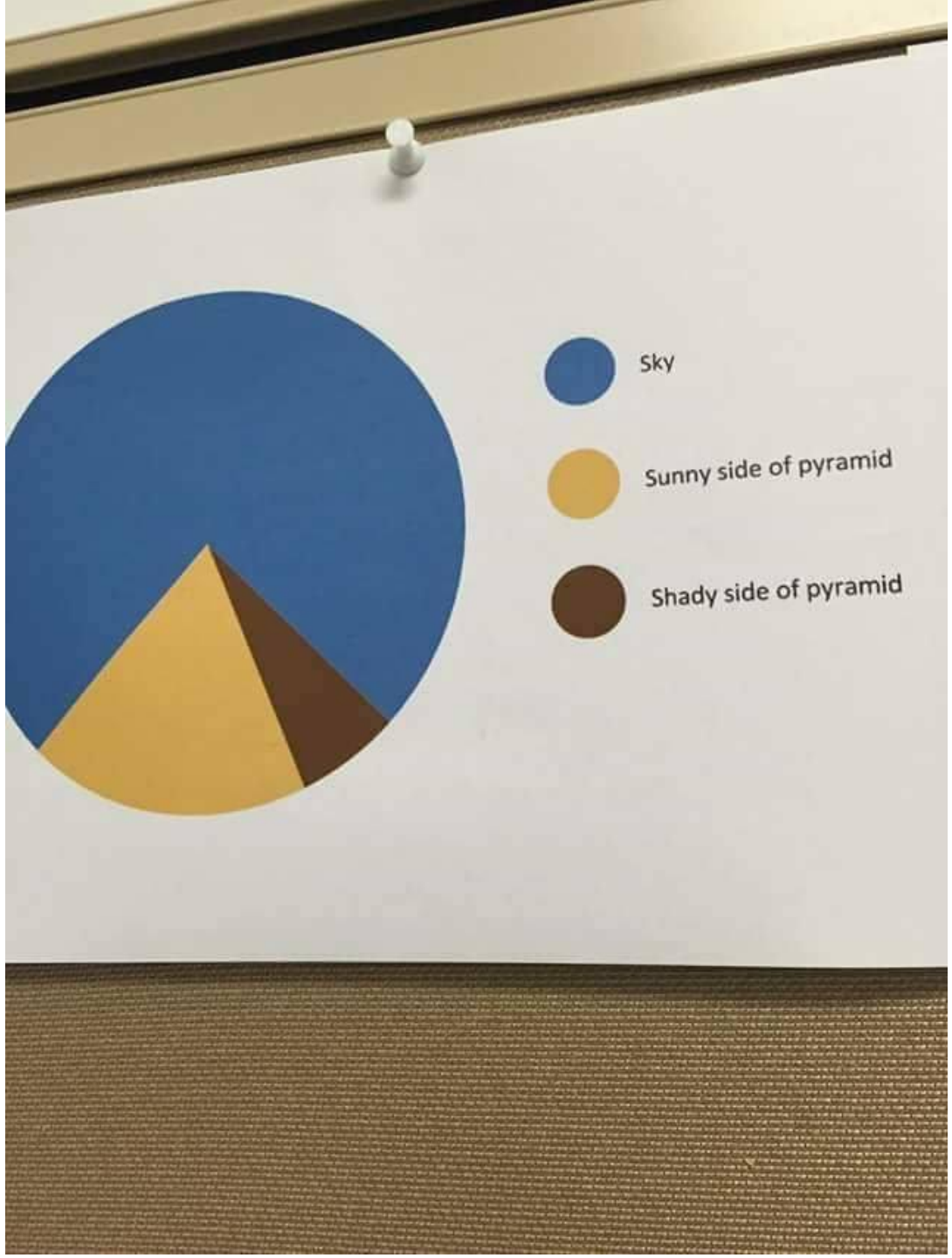


The best pie chart



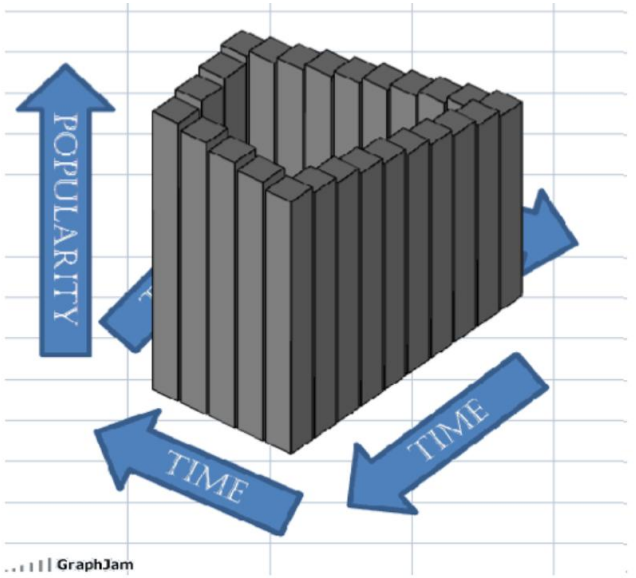
Pie charts jokes

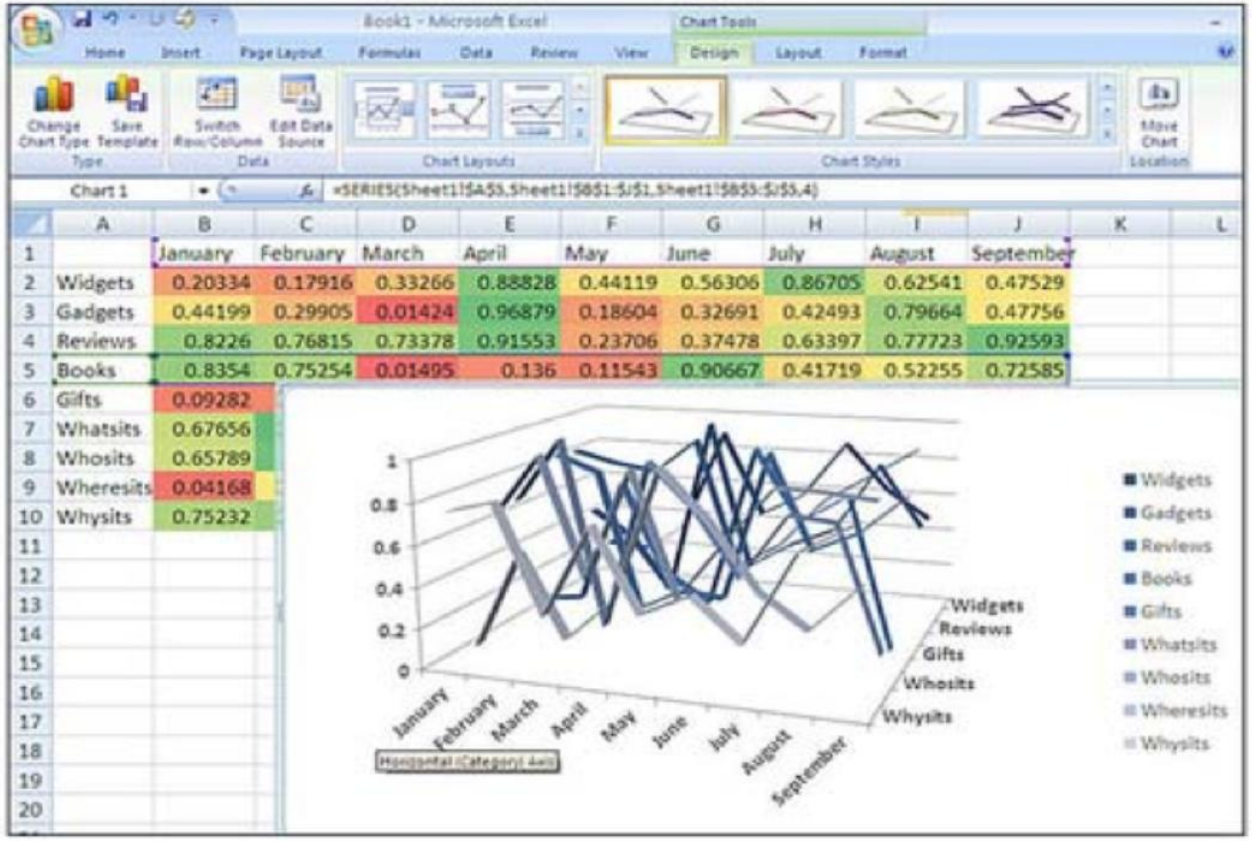
- notoriously bad

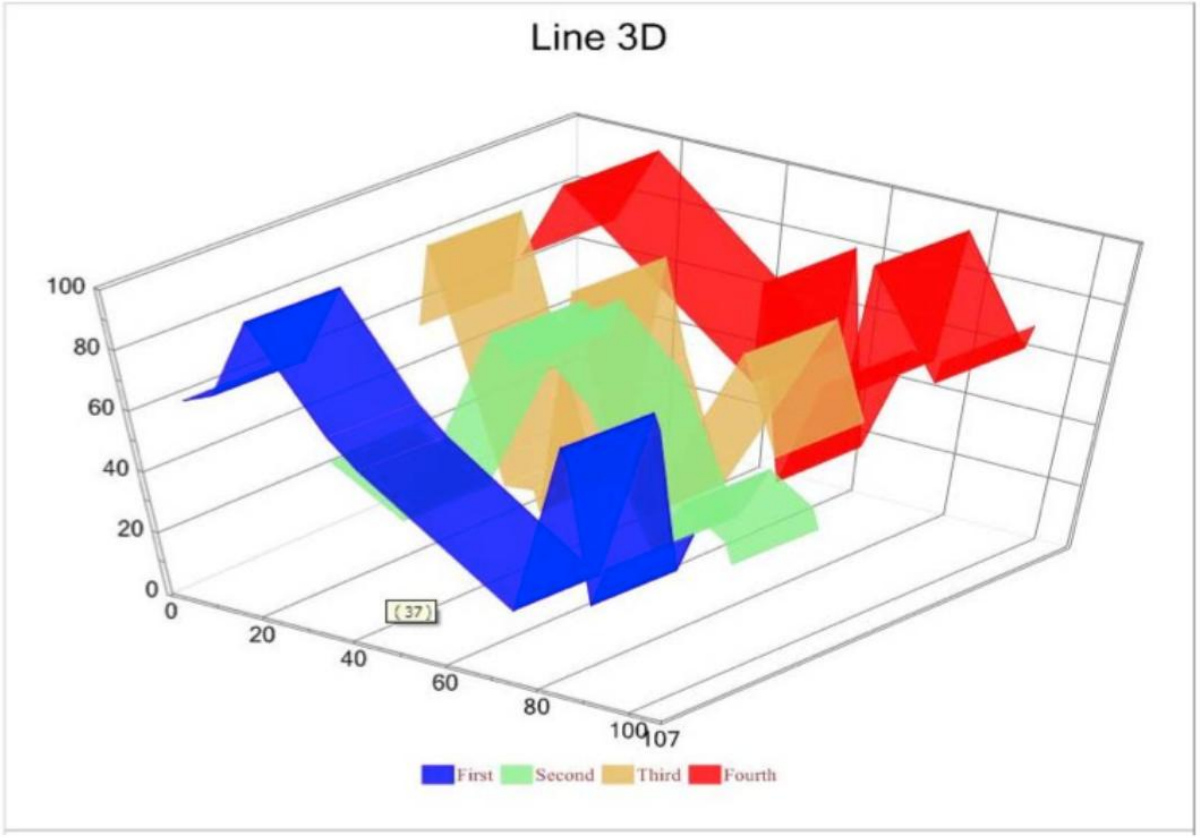


Facts about simple visualizations

- Bar charts, box plots can be OK
- 3D graphs are almost never OK for 2D info: spider plot, bowl of noodles
- Take care to be clear and do not manipulate
- A more detailed examples and recommendations https://github.com/cxli233/FriendsDontLetFriends

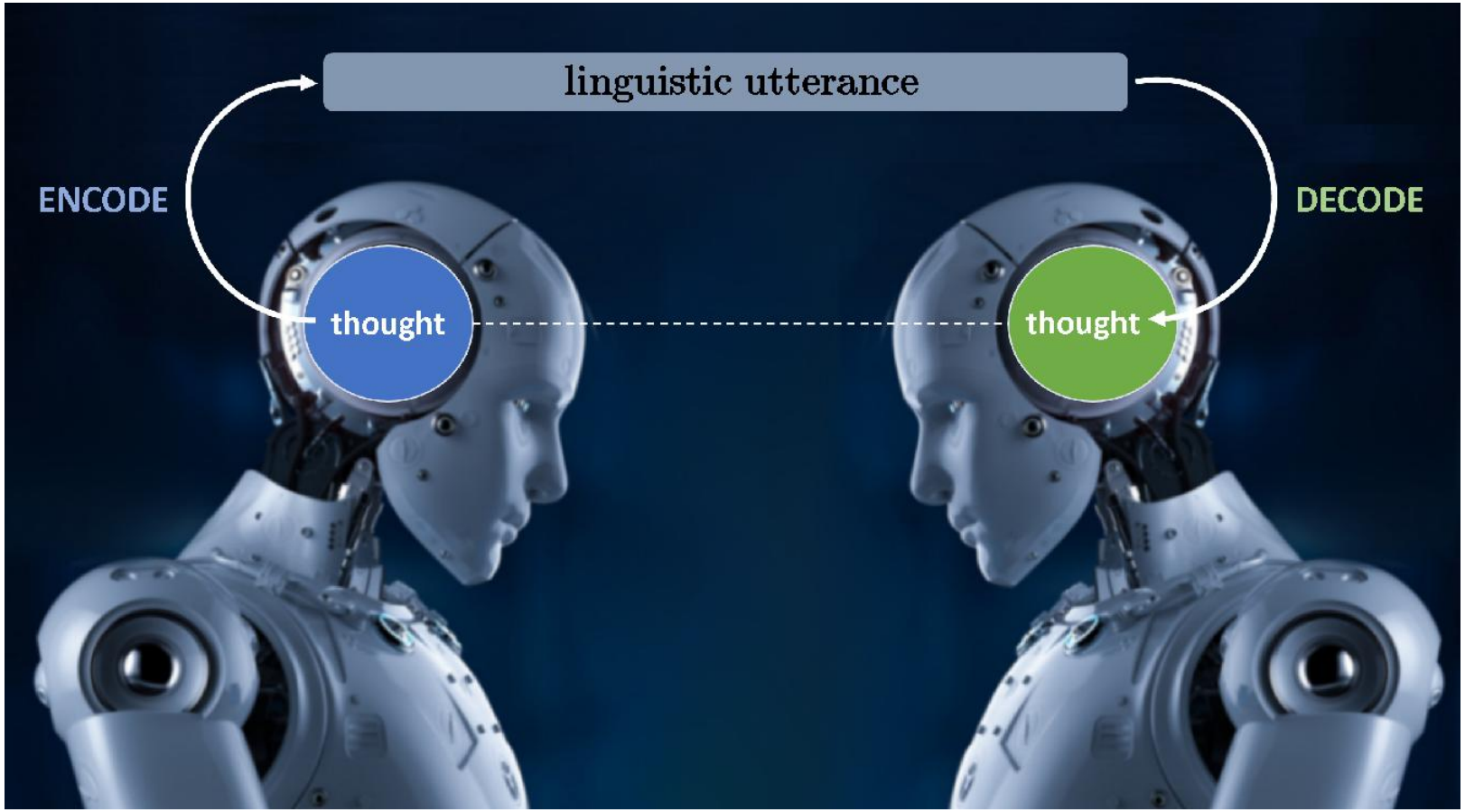






Understanding

Walid Saba, "Machine Learning Won't Solve Natural Language Understanding", The Gradient, 2021.

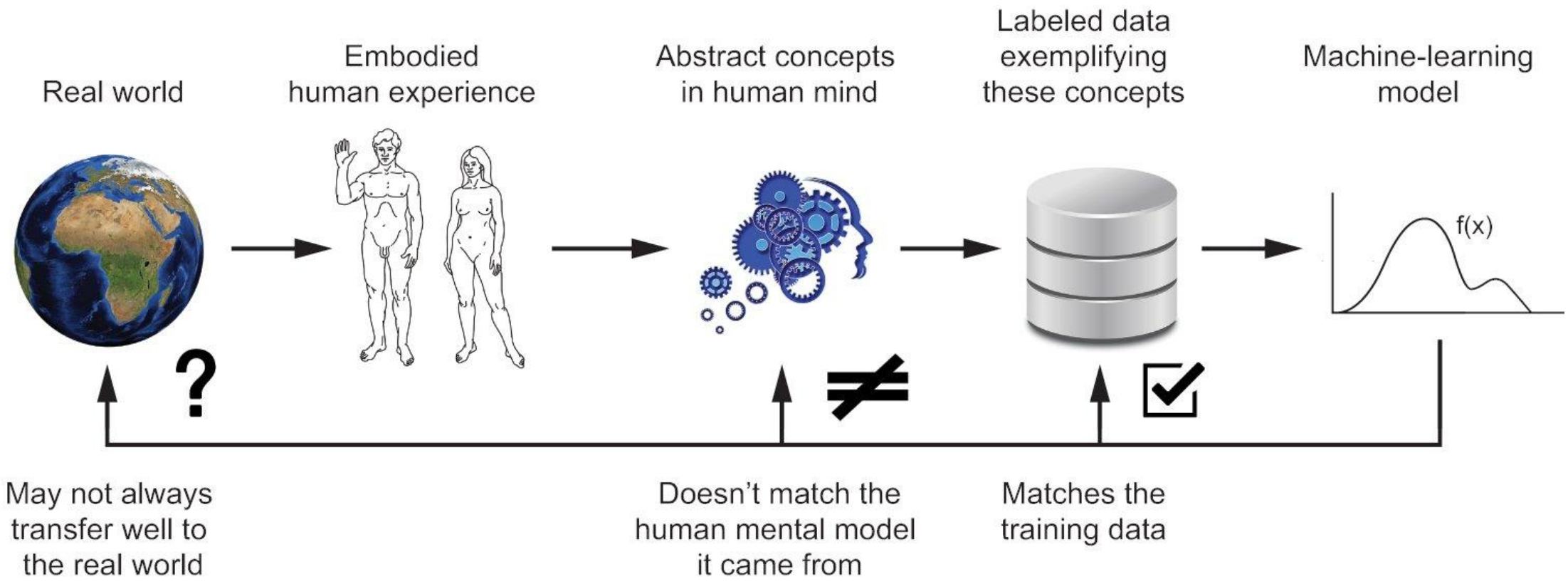


Xanadu, who is a living young human adult, and who was in graduate school, quit graduate school to join a software company that had a need for a new employee.

≈

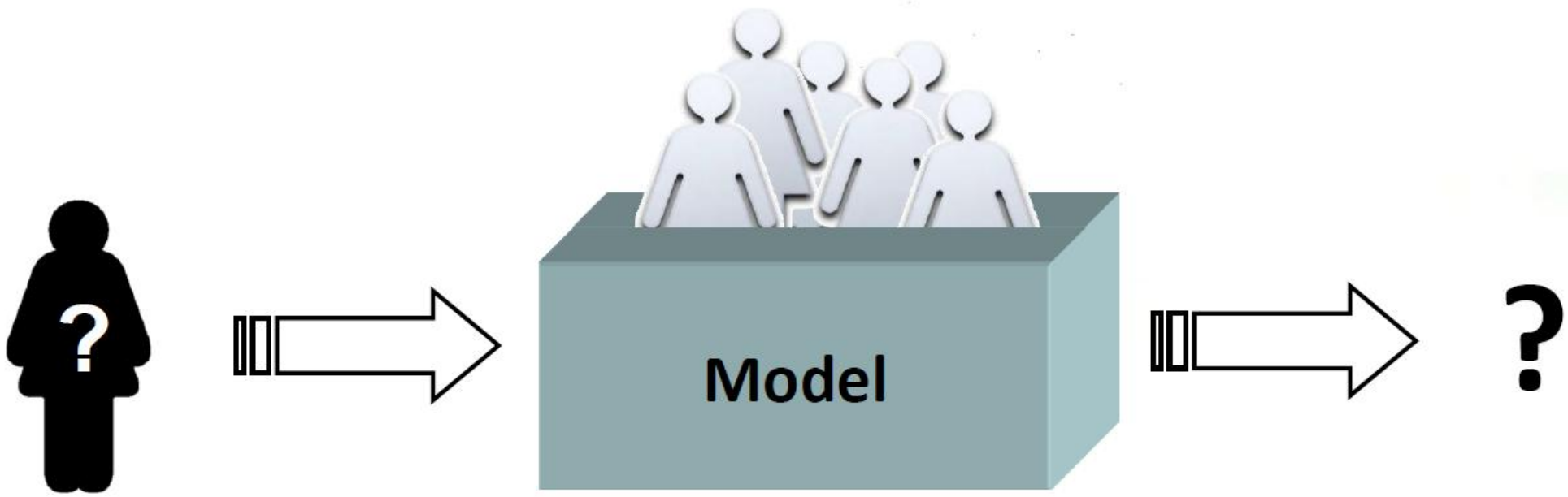
Xanadu quit graduate school to join a software company.

Understanding ML models is difficult



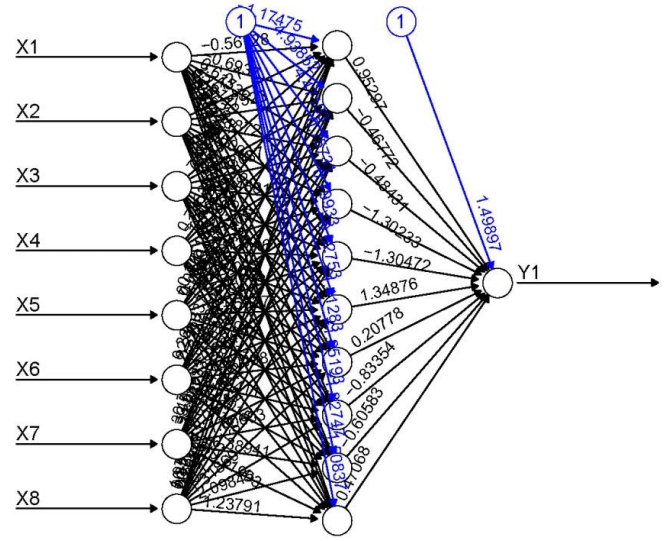
Predictive modeling scenario

We want to learn from past examples,with known outcomes.



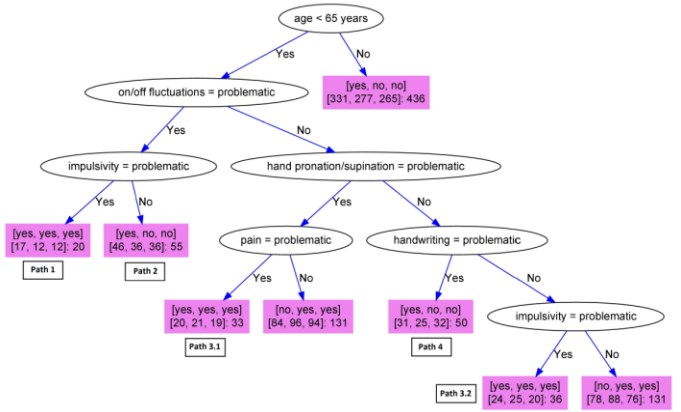
To predict the outcome for a new patient.

Explanation of predictions



- a number of successful prediction algorithms exist (SVM, boosting, random forests, neural networks), but to a user they are [redacted]
- many fields where users are very much concerned with the transparency of the models: medicine, law, consultancy, public services, etc.
- Some explanation methods are applicable to arbitrary predictors

Model comprehensibility



- decision support: model comprehensibility is important to gain users‘ trust
- knowledge acquisition
- some models are inherently interpretable and comprehensible
- decision and regression trees, classification and regression rules, linear and logistic regression $1/(1+\exp(-(b_0+b_1x_1+...b_px_p))$
- really?

Domain level explanation



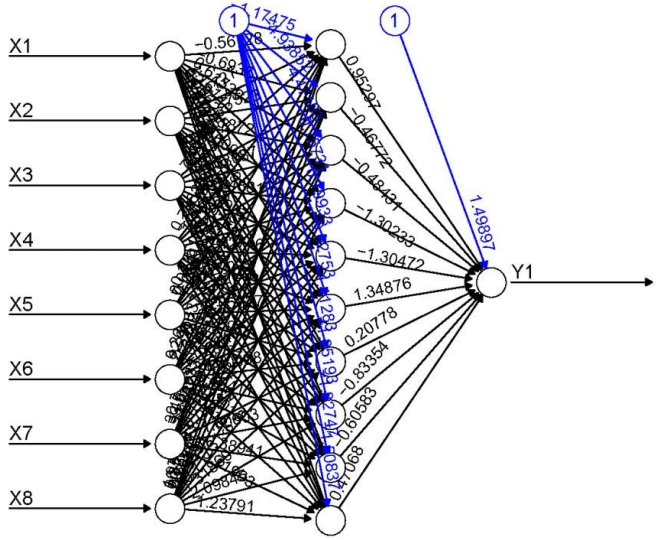
- trying to explain the "true causes and effects"
  - physical processes
  - stock exchange events

- usually unreachable except for artificial problems with known relations and generator function
- some aspects are covered with attribute evaluation, detection of redundancies, ...
- targeted indirectly through the models

Model-based explanations

All models are wrong,
but some are useful.

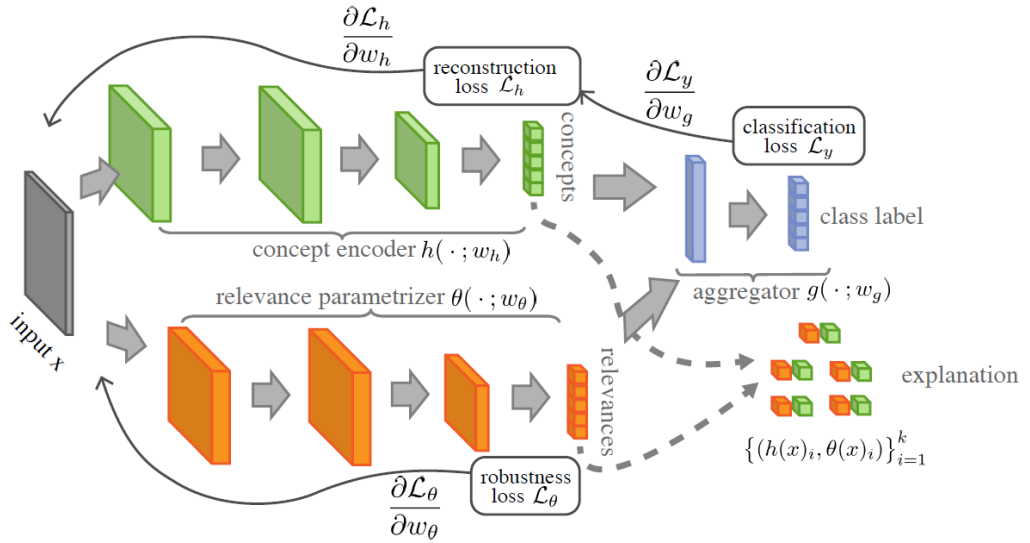
George Box, British statistician (1919 – 2013)



- make transparent the prediction process of a particular model
- the correctness of the explanation is independent of the correctness of the prediction but
- better models (with higher prediction accuracy) enable in principle better explanation at the domain level
- explanation methods are interested only in the explanation at the model level and leave to the developer of the model the responsibility for its prediction accuracy

Two flavours of explanation techniques

- model specific
  - especially used for deep neural networks

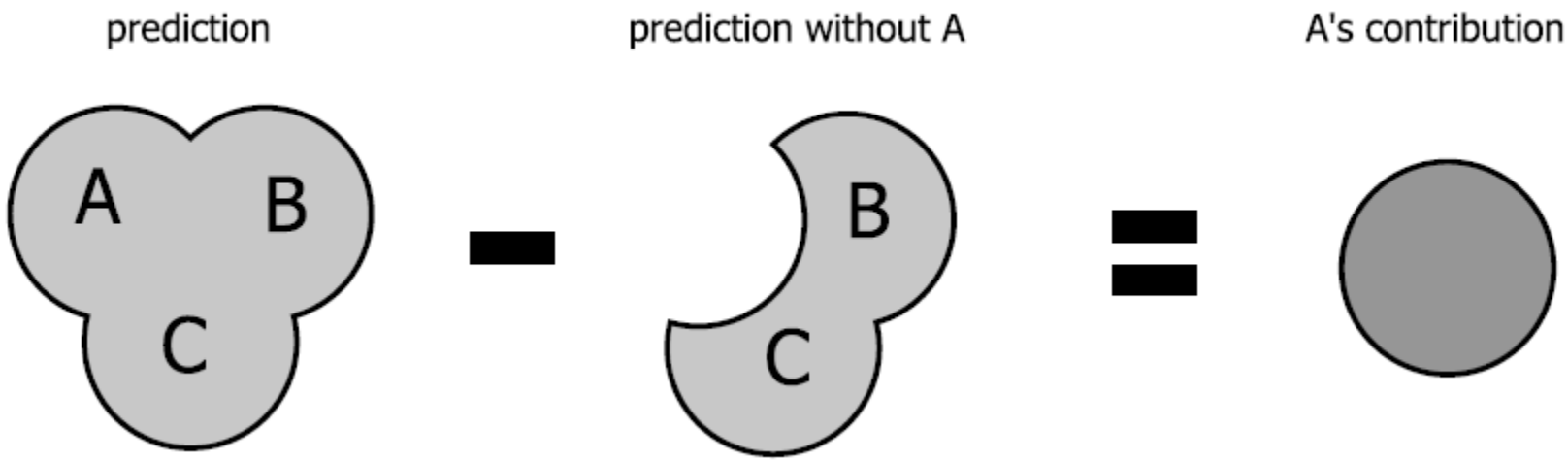


Melis, D.A. and Jaakkola, T., 2018. Towards robust interpretability with self-explaining neural networks. In *Advances in Neural Information Processing Systems* (pp. 7786-7795).

- model agnostic
  - can be used for any predictor,
  - based on perturbation of the inputs

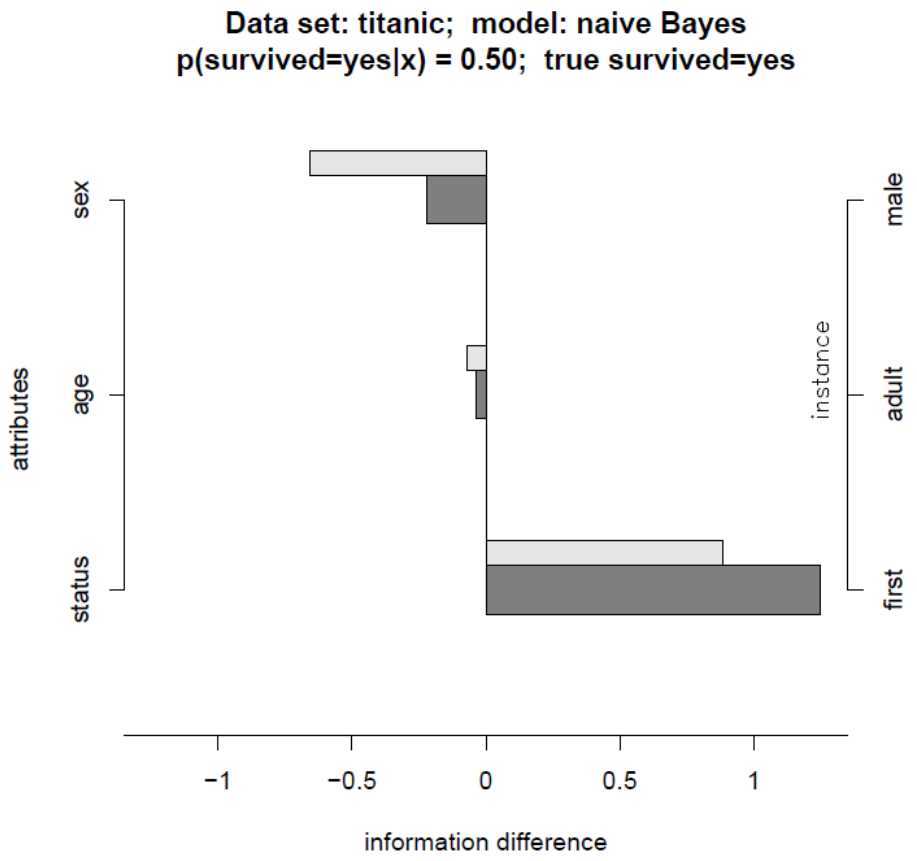
Idea of perturbtion-based explanations

- importance of a feature or a group of features in a specific model can be estimated by simulating lack of knowledge about the values of the feature(s)



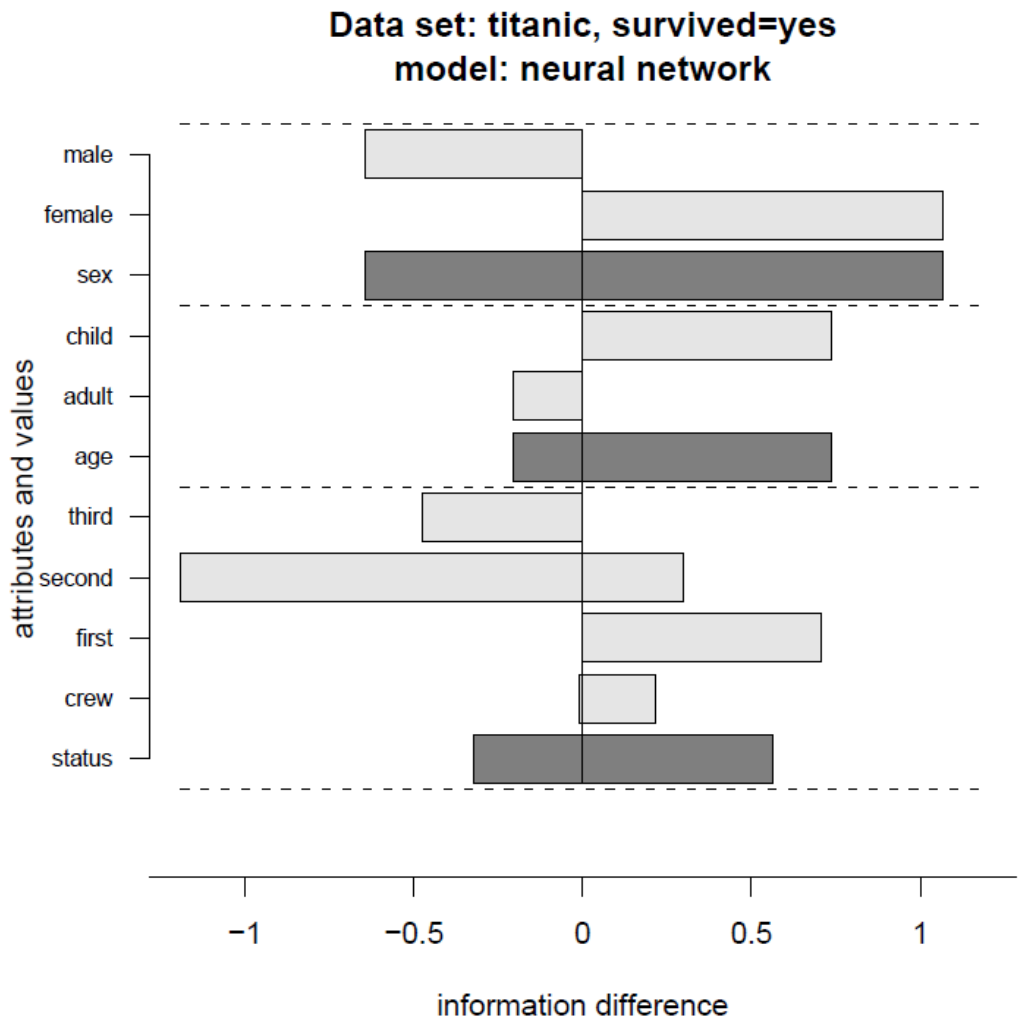
Instance-level explanation

- explain predictions for each instance separately
  - this is what practitioners applying models are interested in
  - presentation format: impact of each feature on the prediction value
- model-based



Model-level explanation

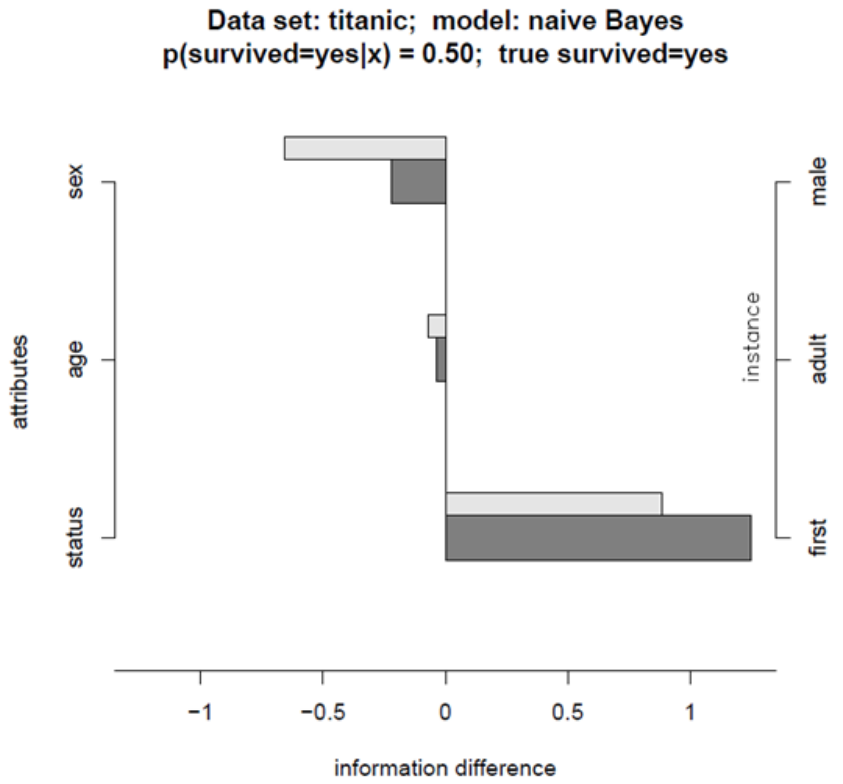
- the overall picture of a problem the model conveys
  - this is what knowledge extractors are interested in
  - presentation format: overall importance of each feature,
    but also rules, trees
- model-based



The method EXPLAIN

- "hide" one attribute at a time
- estimate contribution of attribute from

$$p(y_k|x) - p_{S\setminus\{i\}}(y_k|x)$$



Robnik-Sikonja, M., & Kononenko, I. (2008). Explaining classifications for individual instances. *IEEE Transactions on Knowledge and Data Engineering,*, *20*(5), 589-600.

Explaining EXPLAIN

- assume an instance (**x**, y); components of **x** are values of attributes A_i
- for a new instance **x**, we want to know what role each attribute's value play in the prediction model f, i.e. to what extend it contributed to the classification f(**x**)
- for that purpose
  - we compute f(**x** \ A_i), the model's prediction for **x** without the knowledge of the event $A_i = a_k$ (marginal prediction)
  - we comparing f(**x**) and f(**x** \ A_i) to assess importance of $A_i = a_k$
  - the larger the the difference the more important the role of $A_i=a_k$ in the model
- f(**x**) and f(**x** \ A_i) are source of explanations

Evaluation of prediction differences

- how to evaluate $f(\mathbf{x}) - f(\mathbf{x} \setminus A_i)$
- in classification, we take $f(\mathbf{x})$ in the form of probability

1. **difference of probabilities**

   $probDiff_i(y|\mathbf{x}) = p(y|\mathbf{x}) - p(y|\mathbf{x} \setminus A_i)$

2. **information gain** (Shannon, 1948)

   $infGain_i(y|\mathbf{x}) = \log_2 p(y|\mathbf{x}) - \log_2 p(y|\mathbf{x} \setminus A_i)$

3. **weight of evidence** also log odds ratio (Good, 1950)

   $odds(z) = p(z) / (1 - p(z))$

   $WE_i(y|\mathbf{x}) = \log_2 odds(y|\mathbf{x}) - \log_2 odds(y|\mathbf{x} \setminus A_i)$

Implementation

- $p(y|\mathbf{x})$: classify $\mathbf{x}$ with the model
- $p(y|\mathbf{x}\setminus A_i)$ – simmulate lack of knowledge of A_i in the model
  - replace with special NA value: good for some, mostly bad, left to the mercy of model's internal mechanism
  - average prediction across perturbations of A_i

    $p(y|\mathbf{x}\setminus A_i) = \sum_a p(A_i=a_s)\, p(y|\mathbf{x} \leftarrow A_i = a_s)$

    - use discretization for numeric attributes
    - use Laplace correction for probability estimation
  - we could build a separate model for each $p(y|\mathbf{x}\setminus A_i)$

Weaknes of EXPLAIN

- "hide" one attribute at a time
- estimate contribution of attribute from

$$p(y_k|x) - p_{S\setminus\{i\}}(y_k|x)$$

- weakness: if there are redundant ways to express concept, credit is not assigned
- example:

  $C = A_1 \vee A_2A_3$

  explanation for instance ($A_1=A_2=A_3=1$)

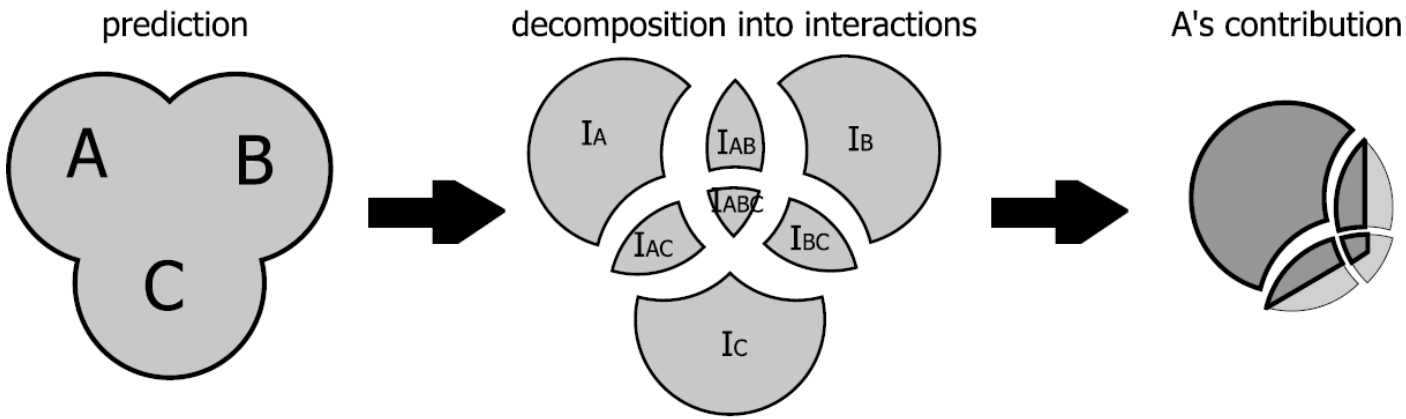
University of Ljubljana
Faculty of Computer and Information Science

The method IME

- (Interactions-based Method for Explanation)
- “hide” any subset of attributes at a time (2^a subsets!)
- the source of explanations is the difference in prediction using a subset of features Q and an empty set of features {}

$$\Delta_Q = h(x_Q) - h(x_{\{\}})$$

- the feature gets some credit for standalone contributions and for contributions in interactions



IME: sum over all subsets

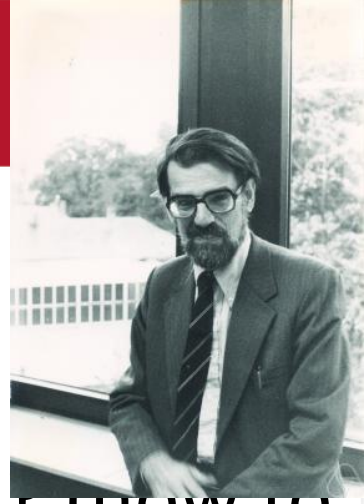
- the contributions are

$$\pi_i = \sum_{Q \subseteq \{1,2,\ldots,a\} - \{i\}} \frac{1}{a \binom{a-1}{a-|Q|-1}} (\Delta_{Q \cup \{i\}} - \Delta_Q)$$

Štrumbelj, E., Kononenko, I. & Robnik-Šikonja, M., Explaining instance classifications with interactions of subsets of feature values. *Data & Knowledge Engineering*, Oct. 2009, 68(10):886-904

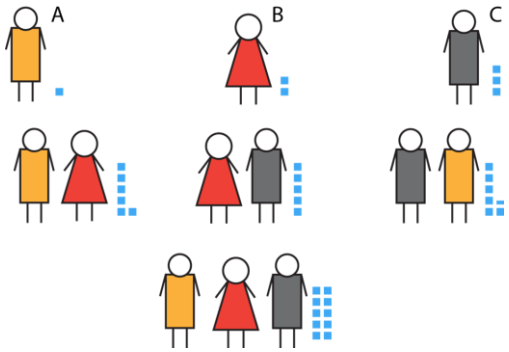
University of Ljubljana
Faculty of Computer and Information Science

Game theory analogy



- coalitional game of *a* players (attributes)
- players form coalitions (i.e. interactions)
- how to distribute the payout to the members of a coalition? (how to assign the credit for prediction)
- The Shapley value is the unique payoff vector that is
  - efficient (exactly splits payoff value),
  - symmetric (equal payments to equivalent players)
  - additive (overall credit is a sum of participating in coalitions), and
  - assigns zero payoffs to dummy players (no contribution to any coalition).

Shapley value



$$Sh_i(v) = \sum_{S \subseteq N \setminus \{i\}, s=|S|} \frac{(n-s-1)!s!}{n!}(v(S \cup \{i\}) - v(S)), \quad i = 1, \ldots, n.$$

$$\pi_i = \sum_{Q \subseteq \{1,2,\ldots,a\} - \{i\}} \frac{1}{a\binom{a-1}{a-|Q|-1}}(\Delta_{Q \cup \{i\}} - \Delta_Q)$$

- Shapley value can be efficiently approximated

University of Ljubljana
Faculty of Computer and Information Science

Solution for IME: sampling

- Shapley value can expressed in an alternative formulation
- $\pi(a)$is the set of all ordered permutations of *a*
- $Pre^i(O)$ is the set of players which are predecessors of player *i* in the order $O \in \pi(a)$

$$\varphi_i(k,x) = \frac{1}{a!} \sum_{\mathcal{O}\in\pi(a)} \left(\Delta(Pre^i(\mathcal{O}) \cup \{i\})(k,x) - \Delta(Pre^i(\mathcal{O}))(k,x)\right) =$$

$$= \frac{1}{a!} \sum_{\mathcal{O}\in\pi(a)} \left(p_{Pre^i(\mathcal{O})\cup\{i\}}(y_k|x) - p_{Pre^i(\mathcal{O})}(y_k|x)\right),$$

- smart sampling over subsets of attributes
- computationally feasible approach

Štrumbelj, E., & Kononenko, I. (2010). An efficient explanation of individual classifications using game theory. Journal of Machine Learning Research, 11, 1-18

IME algorithm

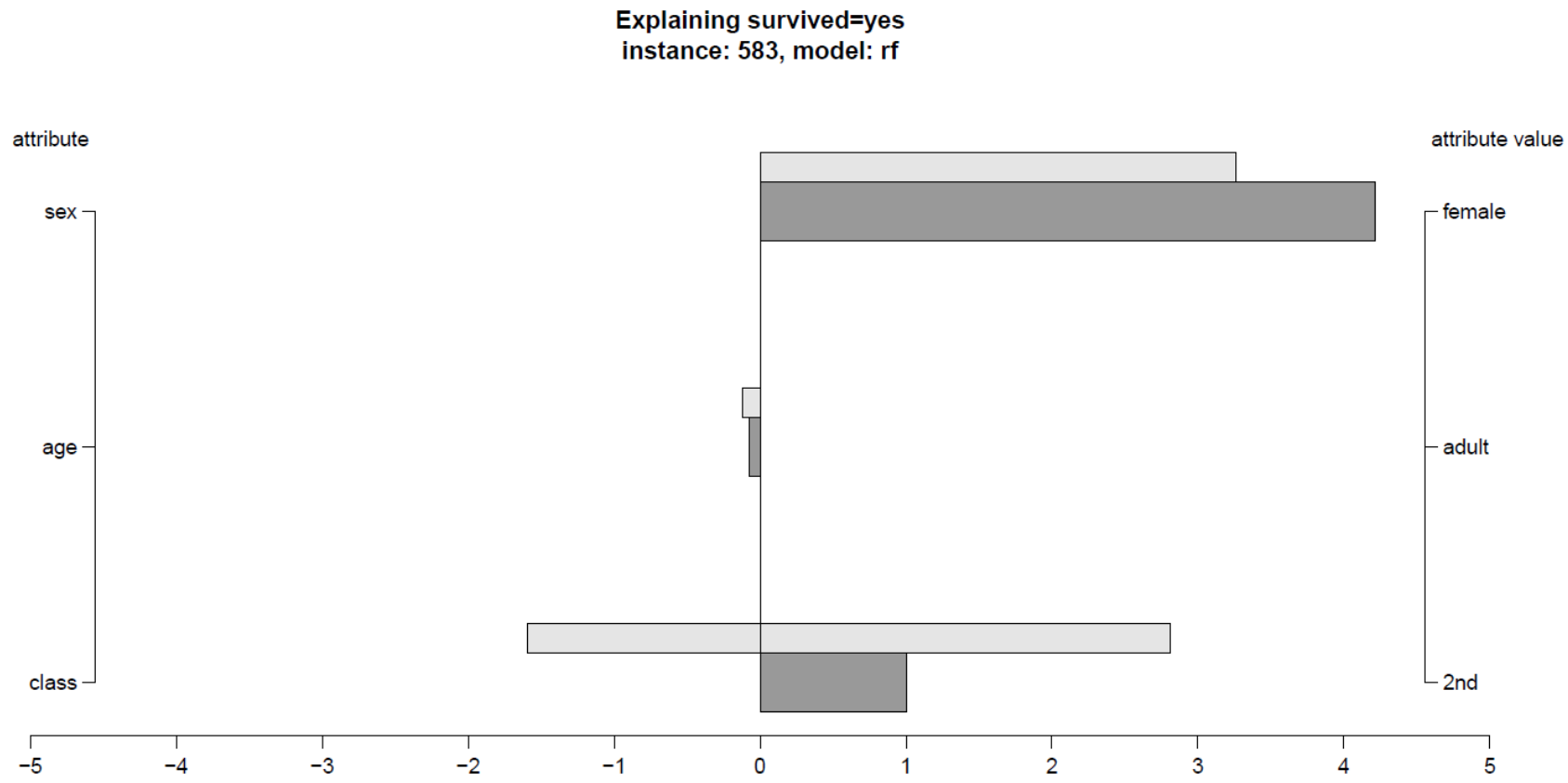
Algorithm 1 Approximating the contribution of the i-th feature's value, φ_i, for instance $x \in \mathcal{A}$.

determine m, the desired number of samples
$\varphi_i \leftarrow 0$
for $j = 1$ to m **do**
  choose a random permutation of features $O \in \pi(N)$
  choose a random instance $y \in \mathcal{A}$
  $v_1 \leftarrow f(\tau(x, y, Pre^i(O) \cup \{i\}))$
  $v_2 \leftarrow f(\tau(x, y, Pre^i(O)))$
  $\varphi_i \leftarrow \varphi_i + (v_1 - v_2)$
end for
$\varphi_i \leftarrow \frac{\varphi_i}{m}$

- by measuring the variance of contributions, we can determine the necessary number of samples for each attribute

Visualization of explanations

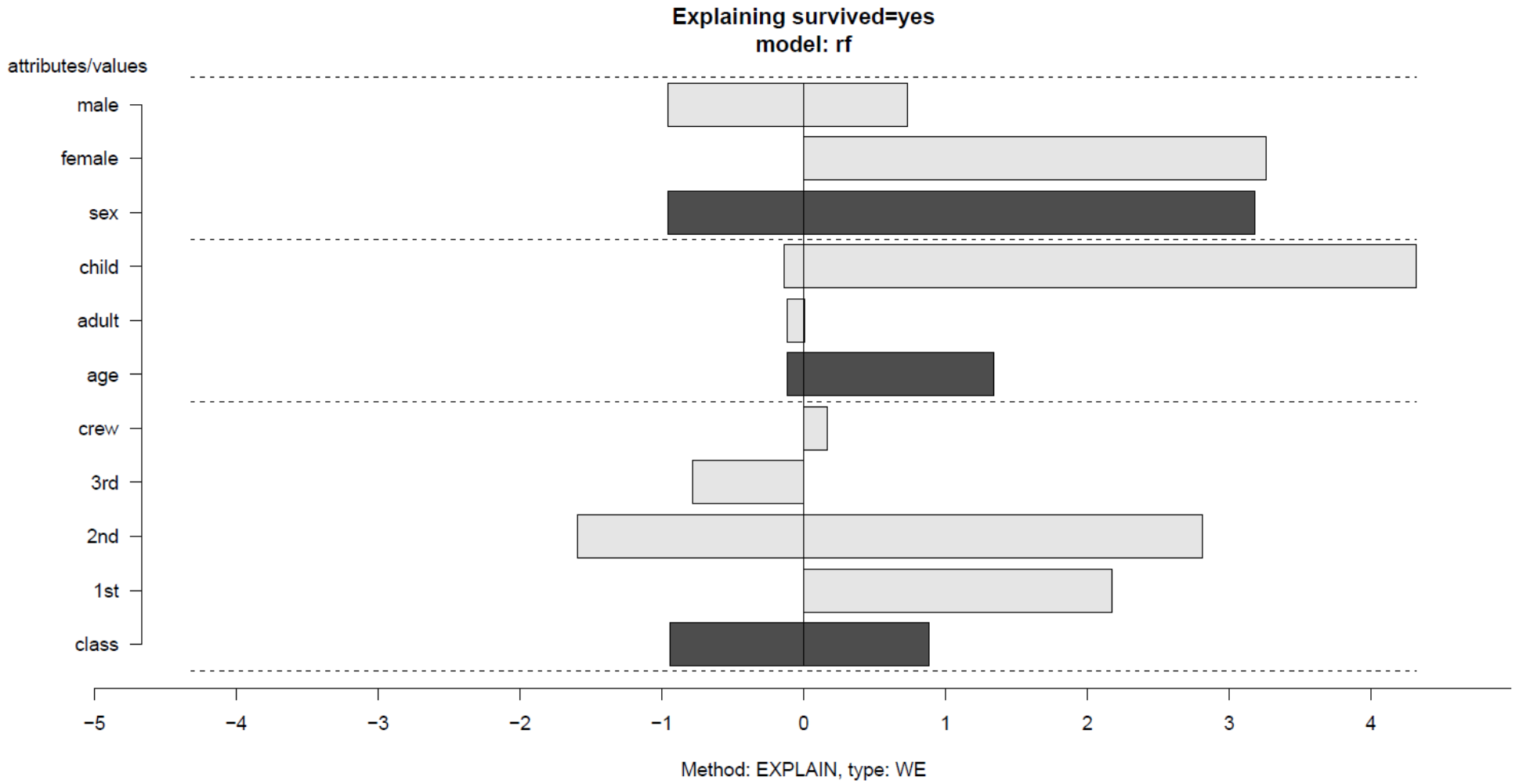
- instance-level explanation on Titanic data set



Robnik-Šikonja, M. (2015), ExplainPrediction: Explanation of Predictions for Classification and Regression.
R package version 1.3.0. http://cran.r-project.org/package=ExplainPrediction

Visualization of explanations

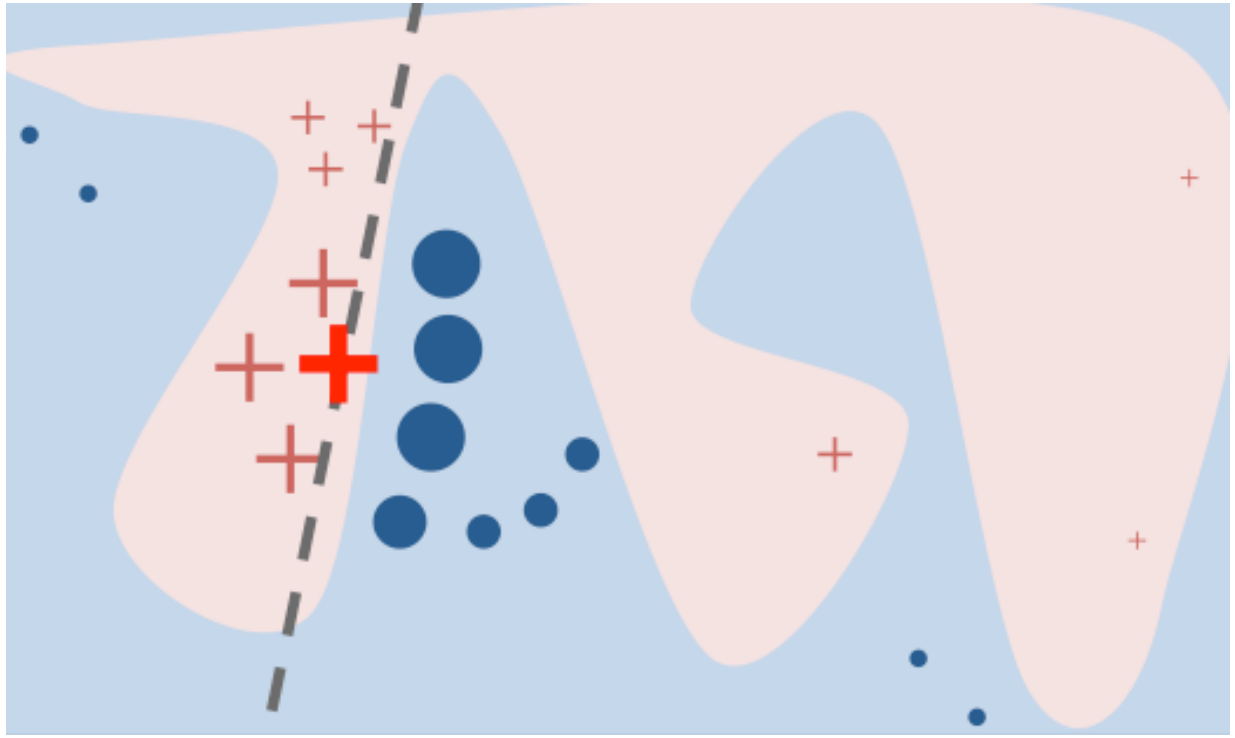
- model-level explanation on Titanic data set



University of Ljubljana
Faculty of Computer and Information Science

LIME explanation method

- Local Interpretable Model-agnostic Explanations)
- perturbations in the locality of an explained instance

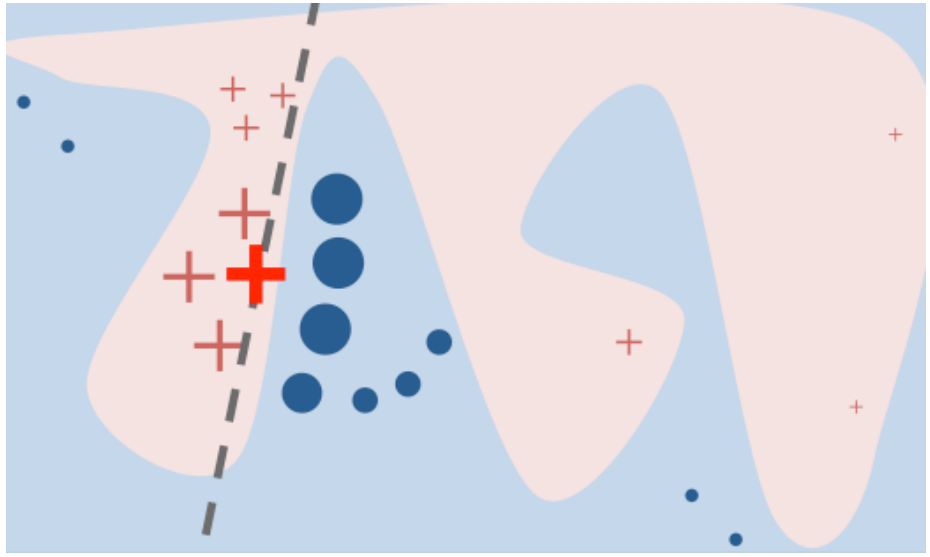


Ribeiro, M. T., Singh, S., & Guestrin, C. (2016). Why should i trust you?: Explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 1135-1144.

LIME explanation method

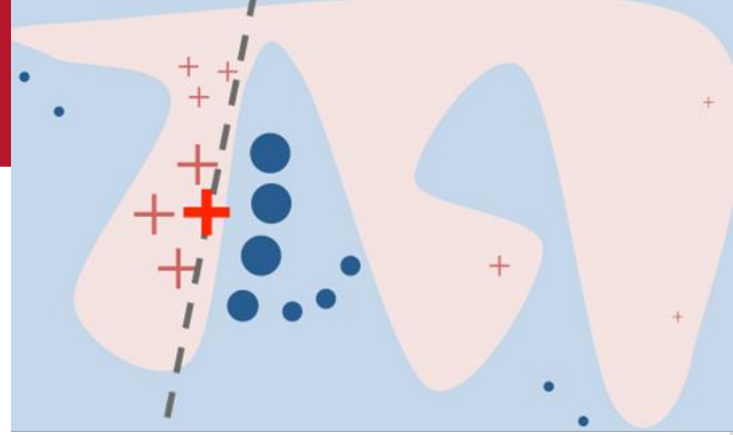
- optimize a trade-off between local fidelity of explanation and its interpretability

$$e(x) = \arg\min_{g \in G} L(f, g, \pi) + \Omega(g)$$



- L is a local fidelity function, f is a model to be explained, g is an interpretable local model g (i.e. linear model), $\pi(x, z)$ is proximity measure between the explained instance *x* and perturbed points *z* in its neighborhood, Ω is a model complexity measure

LIME details



- LIME samples around the explanation instance *x* to draw samples *z* weighted by the distance $\pi(\mathrm{x}, \mathrm{z})$
- samples *z* are used to training an interpretable model *g* (linear model)
- the squared loss measures local infidelity
- number of non-zero weights is complexity
- samples are weighted according to the Gaussian distribution of the distance between *x* and *z*

LIME strengths and weaknesses

- faster than IME
- works for many features, including text and images

- no guarantees that the explanations are faithful and stable
- neighborhood based: a curse of dimensionality
- may not detect interactions due to (too) simple interpretable local model (linear model)



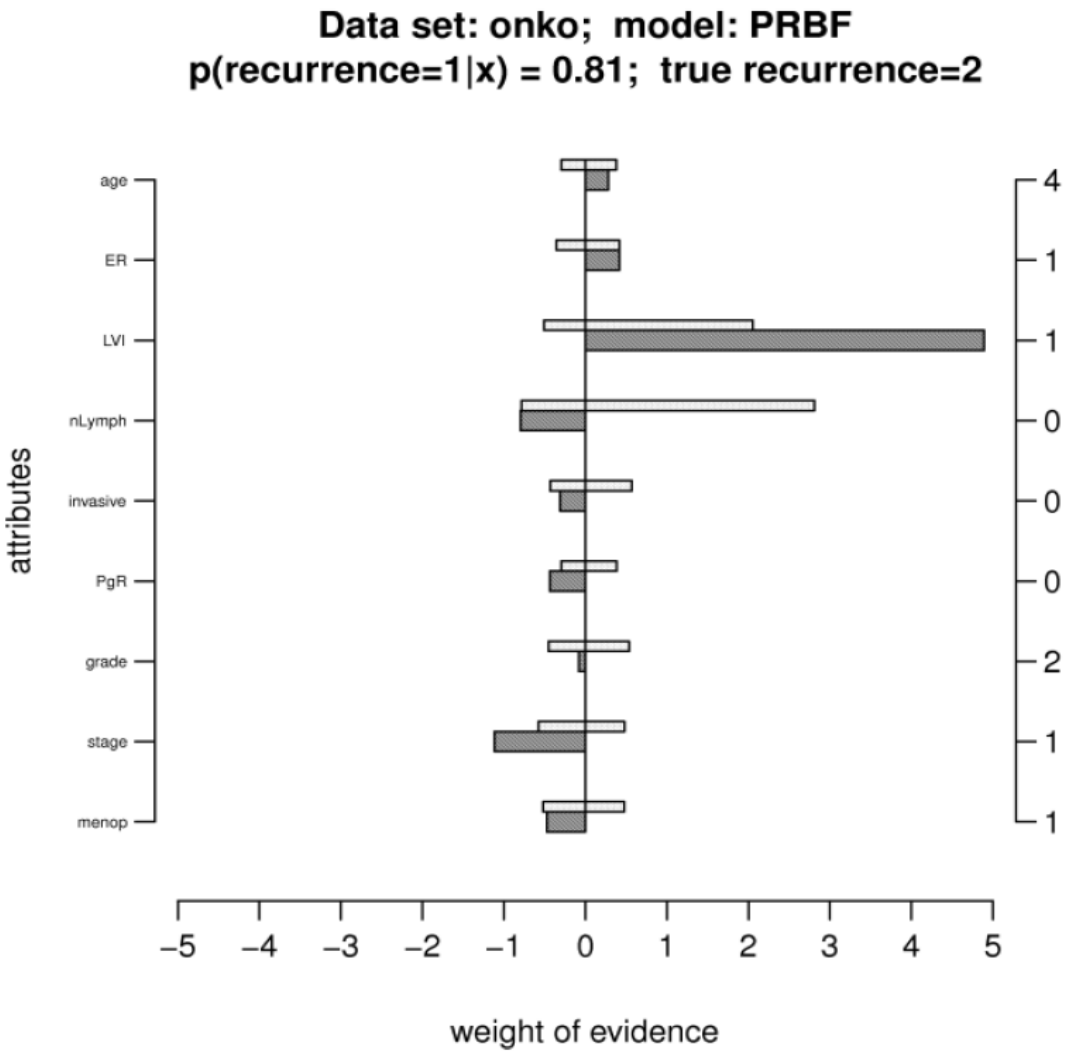
SHAP

- SHapley Additive exPlanation
- unification of several explanation methods, including IME and LIME
- KernelSHAP: based on Shapley values which are estimated using a LIME style linear regression
- faster then IME but
- still uses linear model with all its strengths and weaknesses

Lundberg, S. M., & Lee, S. I. (2017). A unified approach to interpreting model predictions. In Advances in Neural Information Processing Systems (pp. 4765-4774).



Use case: breast cancer recurrence



Cancer recurrence within 10 years

menop binary feature indicating menopausal status

stage tumor stage 1: less than 20mm, 2: between 20mm and 50mm, 3: over 50mm

grade tumor grade 1: good, 2: medium, 3: poor, 4: not applicable, 9: not determined

histType histological type of the tumor 1: ductal, 2: lobular, 3: other

PgR level of progesterone receptors in tumor (in fmol per mg of protein) 0: less than 10, 1: more than 10, 9: unknown

invasive invasiveness of the tumor 0: no, 1: invades the skin, 2: the mamilla, 3: skin and mamilla, 4: wall or muscle

nLymph number of involved lymph nodes 0: 0, 1: between 1 and 3, 2: between 4 and 9, 3: 10 or more

famHist medical history 0: no cancer, 1: 1st generation breast, ovarian or prostate cancer 2: 2nd generation breast, ovarian or prostate cancer, 3: unknown gynecological cancer 4: colon or pancreas cancer, 5: other or unknown cancers, 9: not determined

LVI binary feature indicating lymphatic or vascular invasion

ER level of estrogen receptors in tumor (in fmol per mg of protein) 1: less than 5, 2: 5 to 10, 3: 10 to 30, 4: more than 30, 9: not determined

maxNode diameter of the largest removed lymph node 1: less than 15mm, 2: between 15 and 20mm, 3: more than 20mm

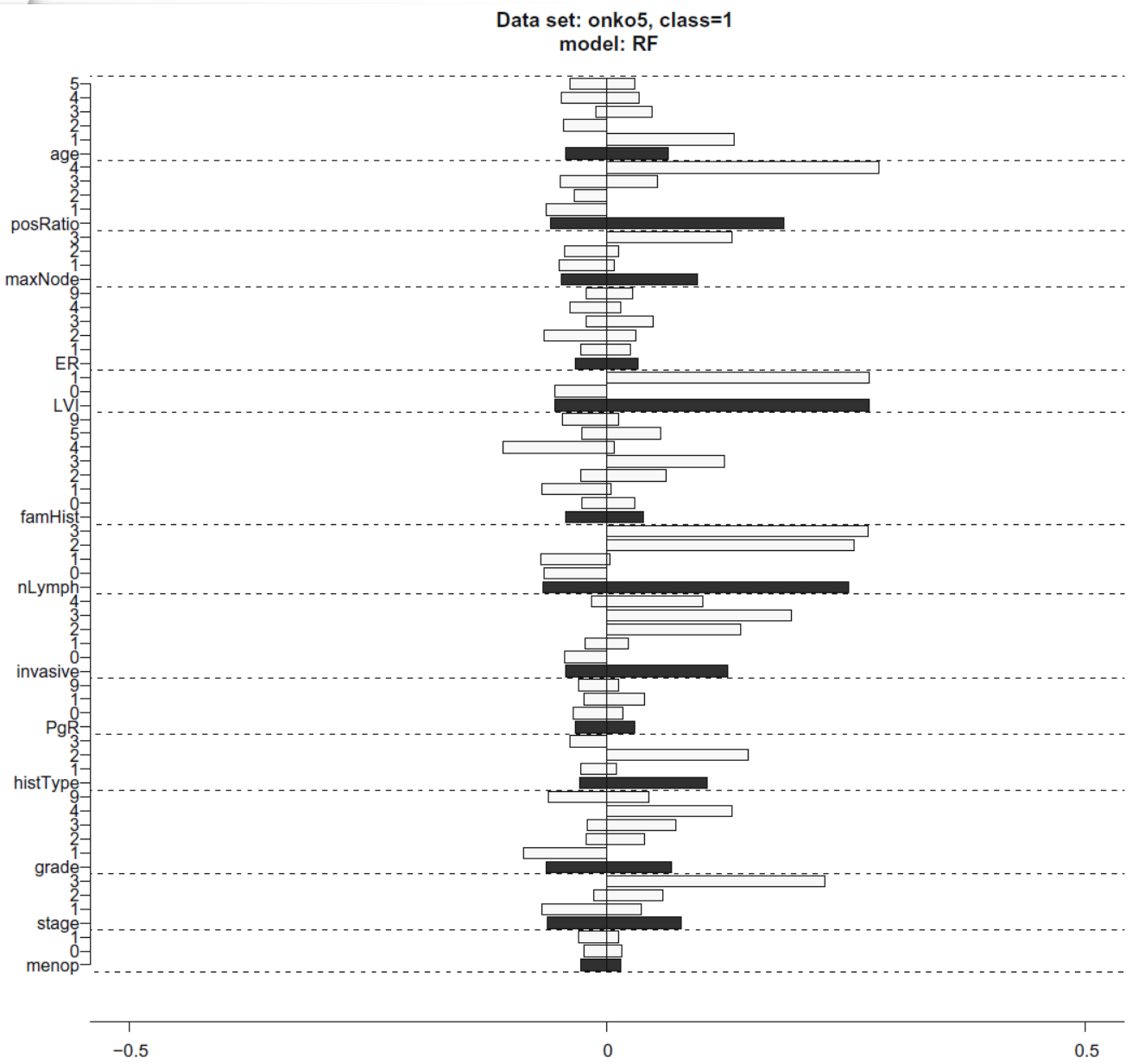
posRatio ratio between involved and total lymph nodes removed 1: 0, 2: less that 10%, 3: between 10% and 30%, 4: over 30%

age patient age group 1: under 40, 2: 40-50, 3: 50-60, 4: 60-70, 5: over 70 years

Robnik-Šikonja, M., Kononenko, I., & Štrumbelj, E. (2012). Quality of classification explanations with PRBF. *Neurocomputing*, *96*, 37-46.

University of Ljubljana
Faculty of Computer and Information Science

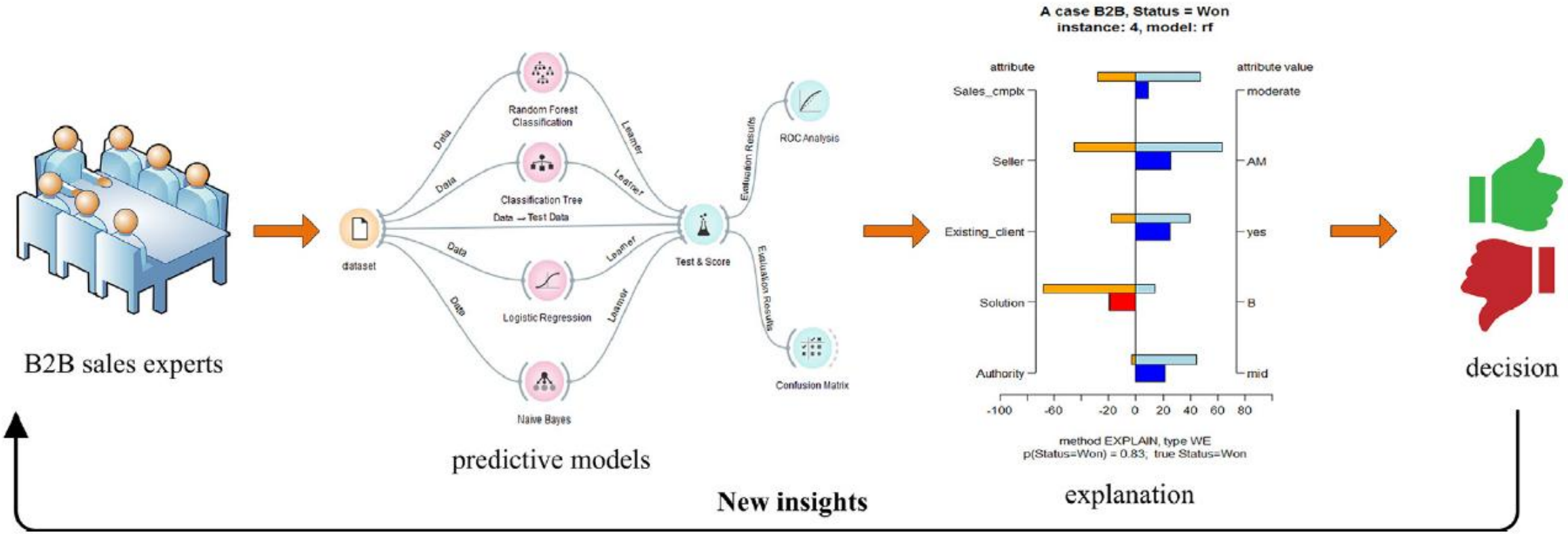
Use case: breast cancer recurrence



University of Ljubljana
Faculty of Computer and Information Science

Use case: B2B sales forecasting

- Goals: improve understanding of factors influencing the outcome and improve the sales performance



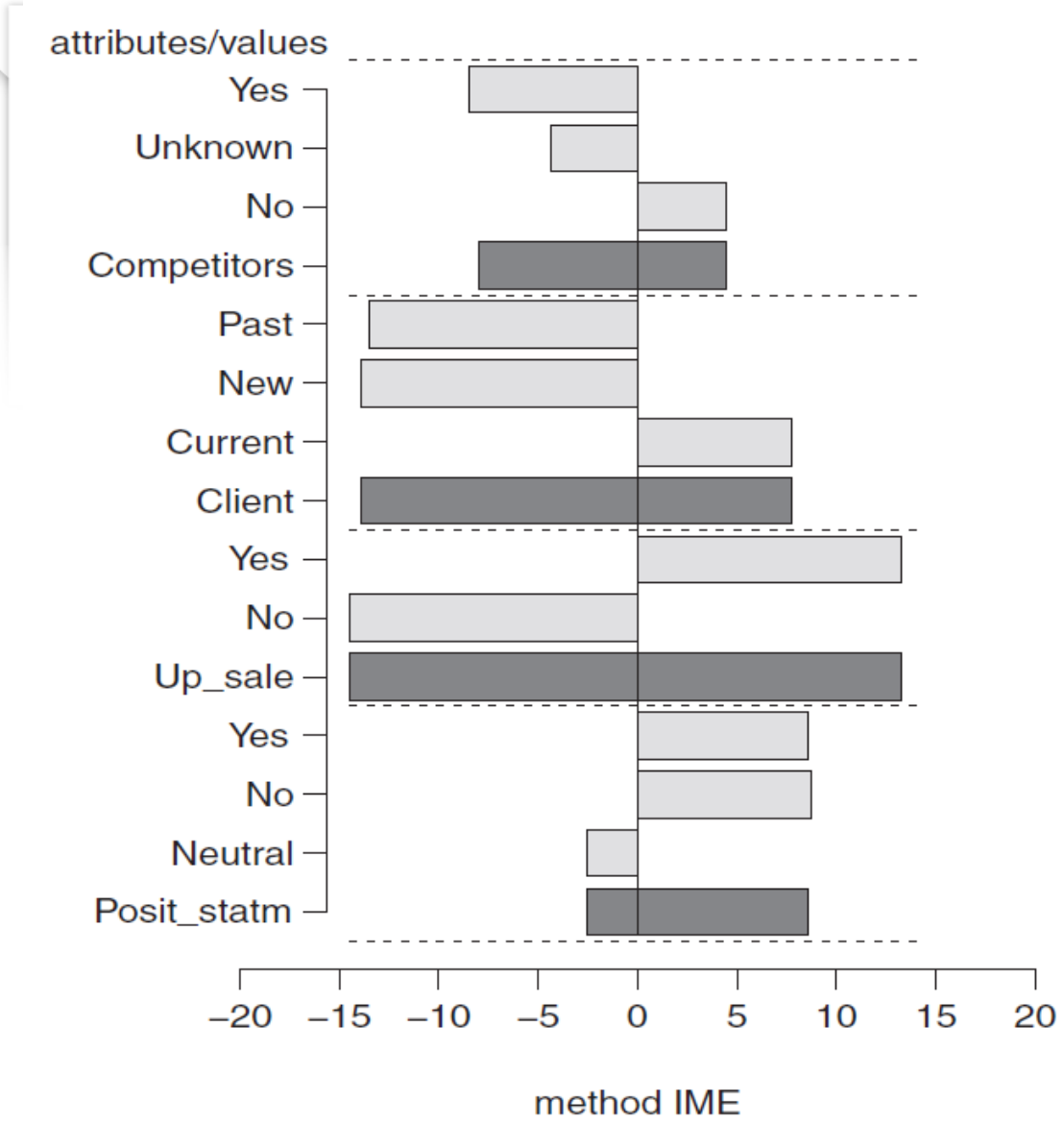
Bohanec, M., Borštnar, M. K., & Robnik-Sikonja, M. (2017). Explaining machine learning models in sales predictions. *Expert Systems with Applications*, *71*, 416-428.

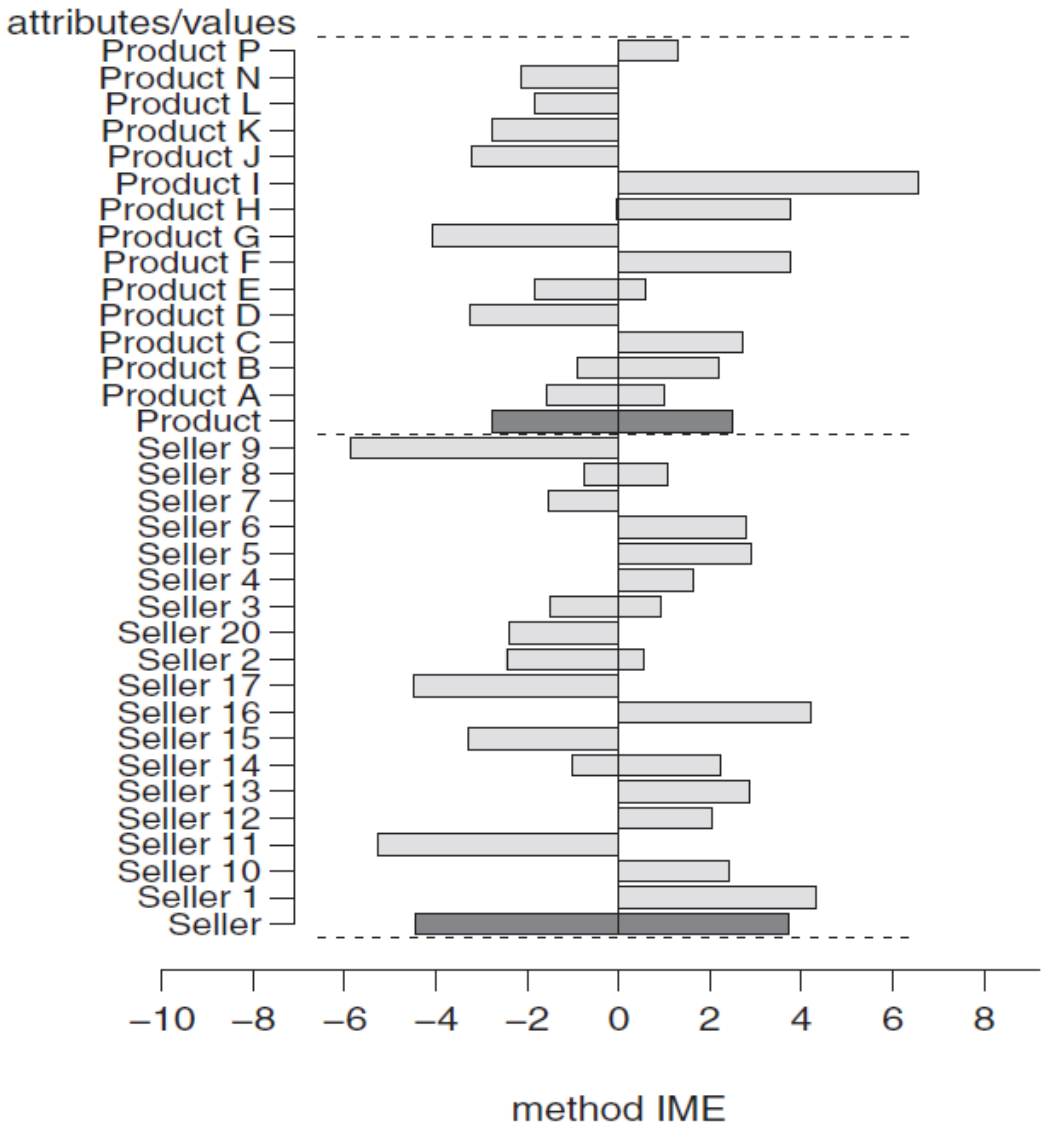
B2B sales attributes

Attribute	Description	Values
Authority	Authority level at a client side	Low, mid, high
Product	Offered product	e.g. A, B, C, etc.
Seller	Seller's name	Seller's name
Competitors	Do we have competitors?	No, yes, unknown
Company size	Size of a company	Big, mid, small
Purchasing department	Is the purchasing department involved?	No, yes, unknown
Partnership	Selling in partnership?	No, yes
Budget allocated	Did the client reserve the budget?	No, yes, unknown
Formal tender	Is a tendering procedure required?	No, yes
RFI	Did we get request for information?	No, yes
RFP	Did we get request for proposal?	No, yes
Growth	Growth of a client?	Growth, stable, etc.
Positive statements	Positive attitude expressed?	No, yes, neutral
Source	Source of the opportunity	e.g. referral, web, etc.
Client	Type of a client	New, current, past
Cross sale	A different product to existing client?	No, yes
Scope clarity	Implementation scope defined?	Clear, few questions, etc.
Strategic deal	Does this deal have a strategic value?	Very important, etc.
Up sale	Increasing sales of existing products?	No, yes
Deal type	Type of a sale	Consulting, project, etc.
Needs defined	Is client clear in expressing the needs?	Info gathering, etc.
Attention to client	Attention to a client	First deal, normal, etc.
Status	An outcome of sales opportunity	Lost, won

University of Ljubljana
Faculty of Computer and Information Science

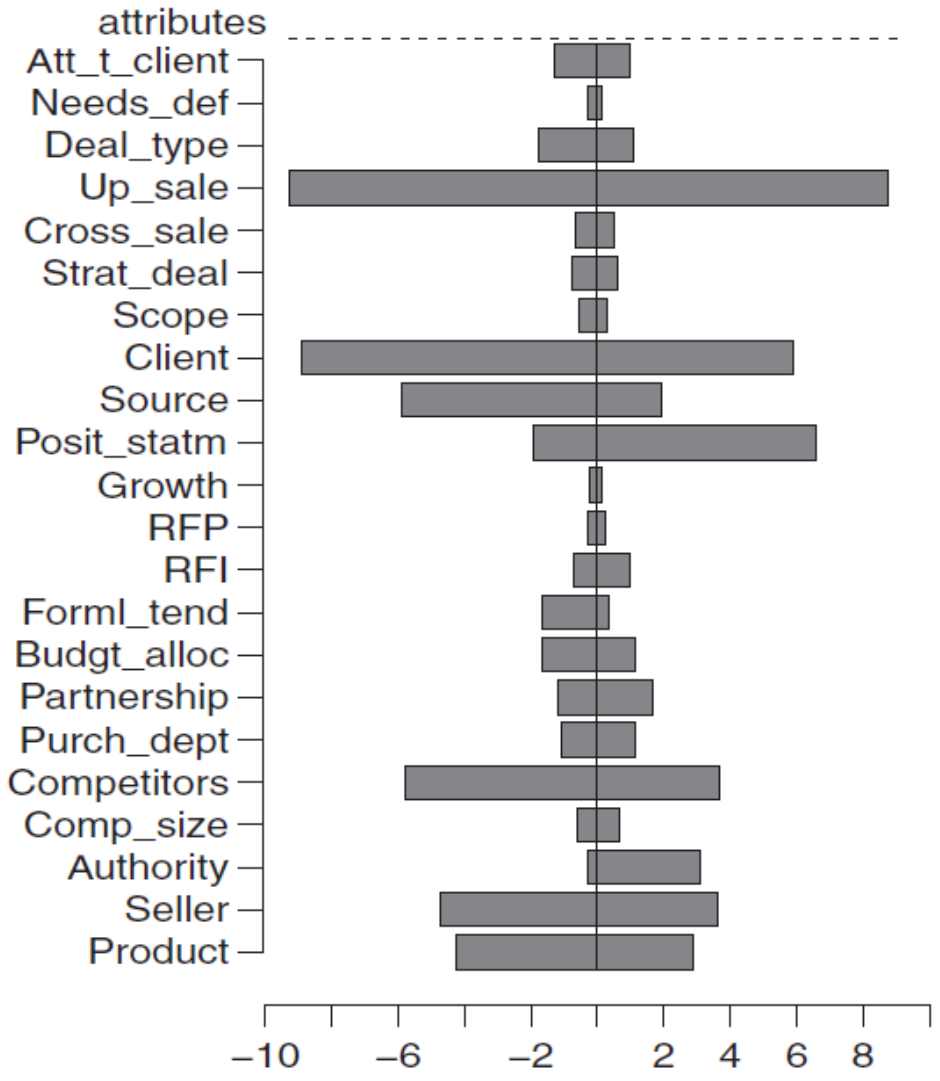
B2B sales: drill in



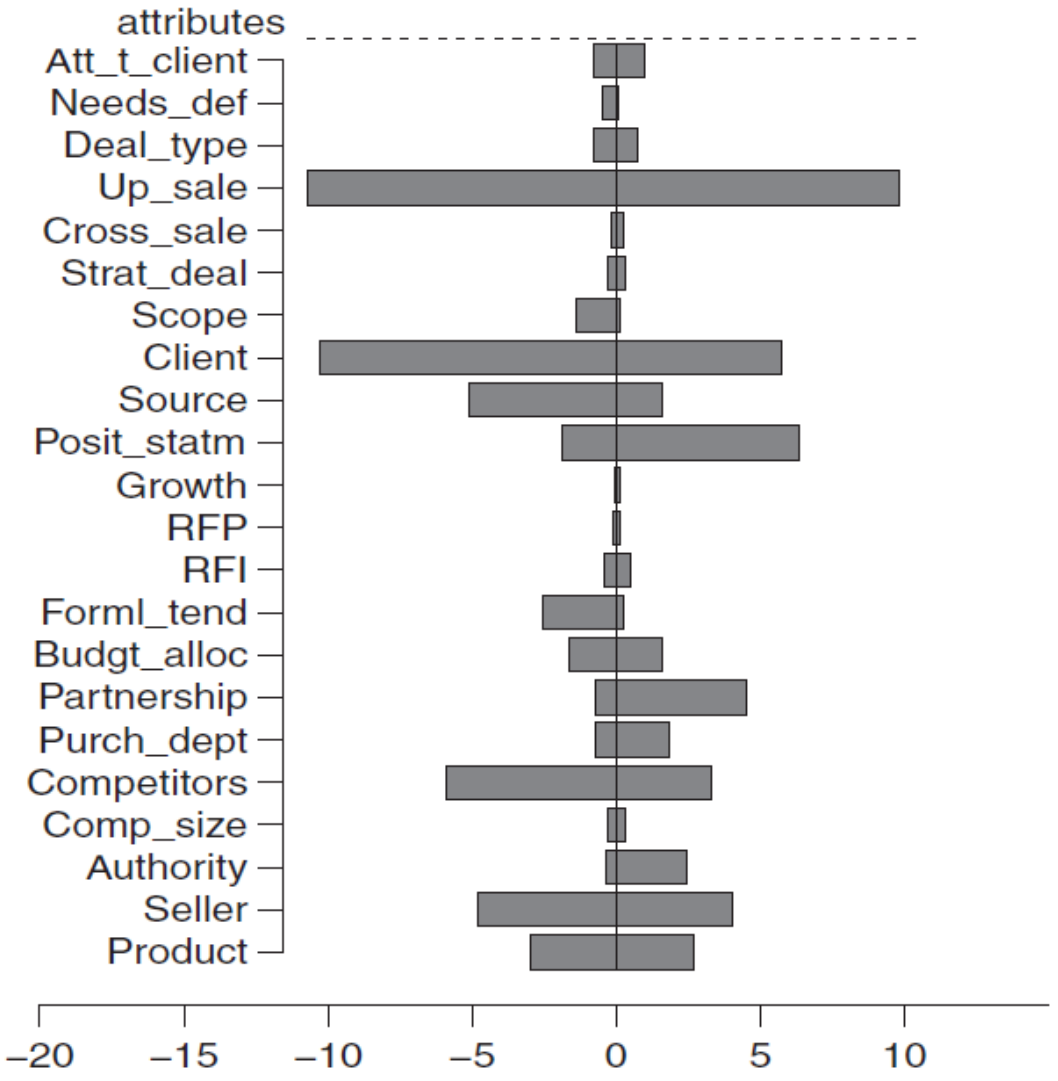


University of Ljubljana
Faculty of Computer and Information Science

B2B sales: EXPLAIN and IME



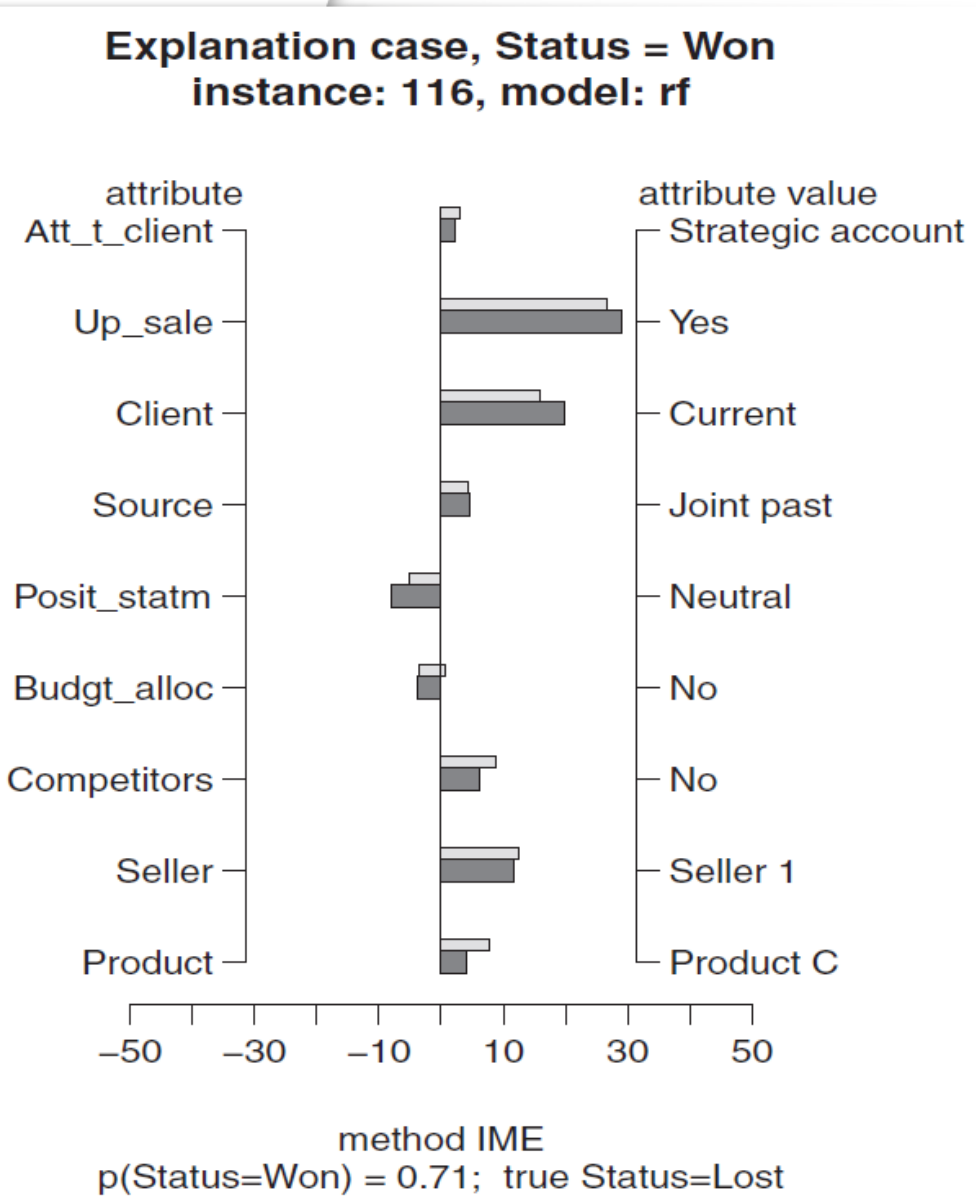
method EXPLAIN, type WE

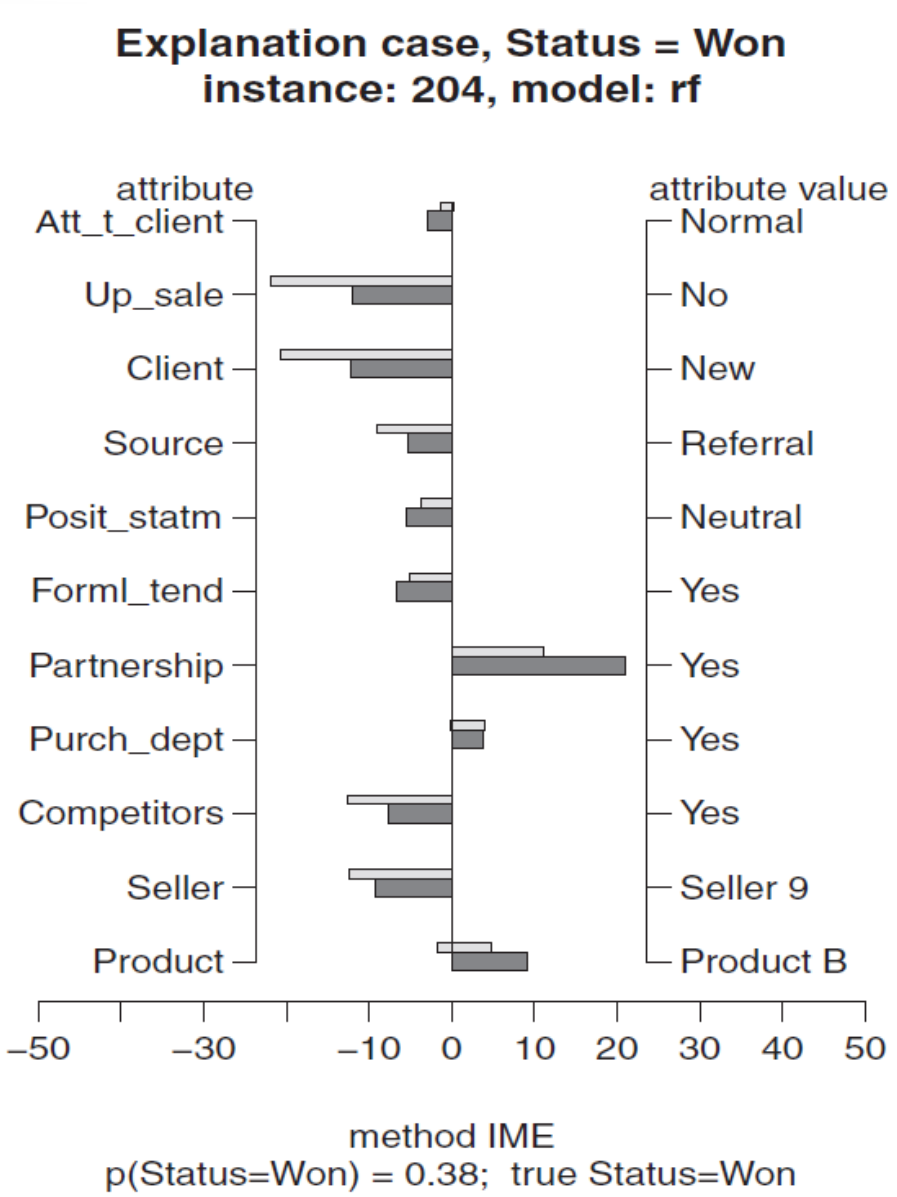


method IME

University of Ljubljana
Faculty of Computer and Information Science

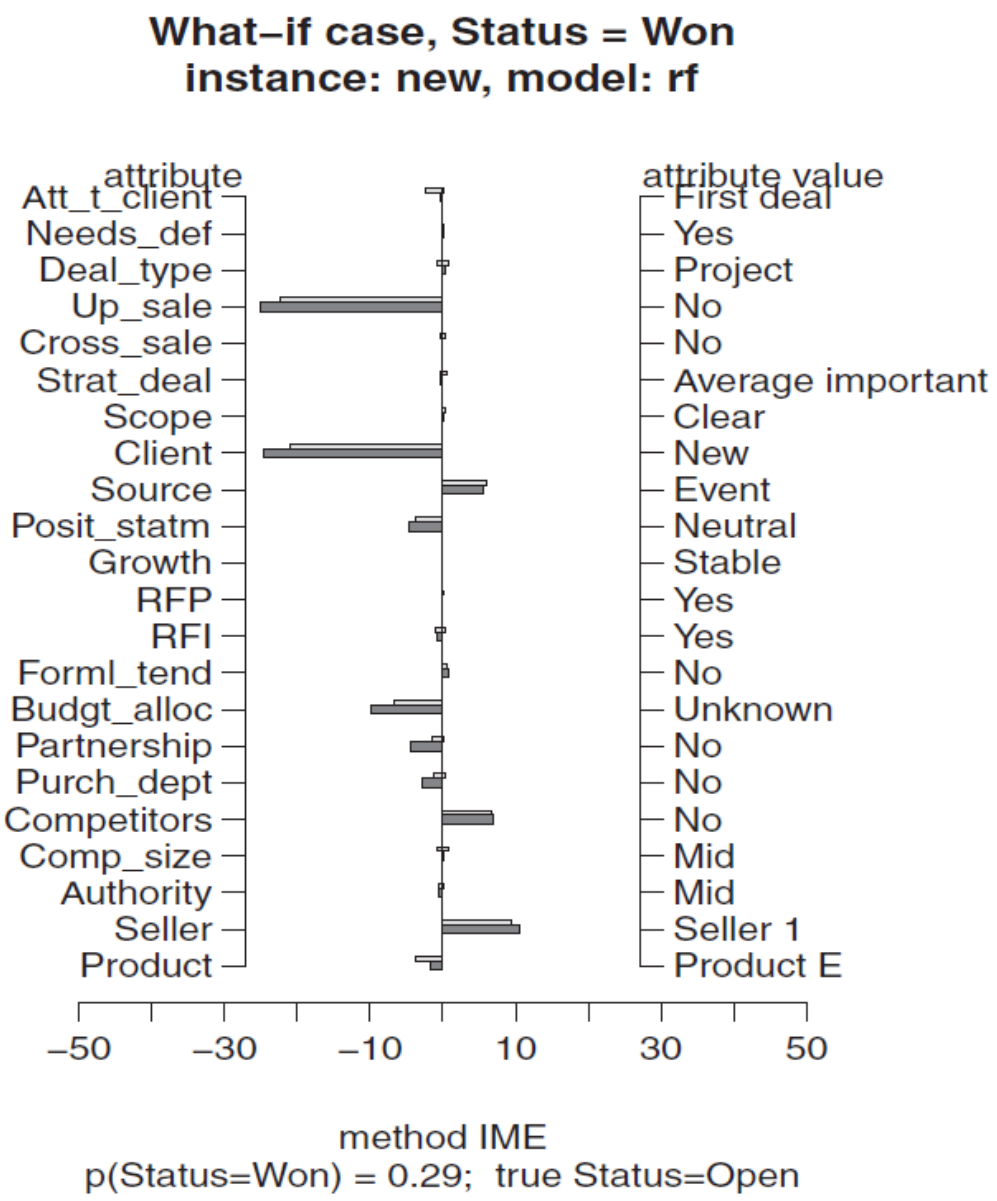
B2B sales: learning from errors





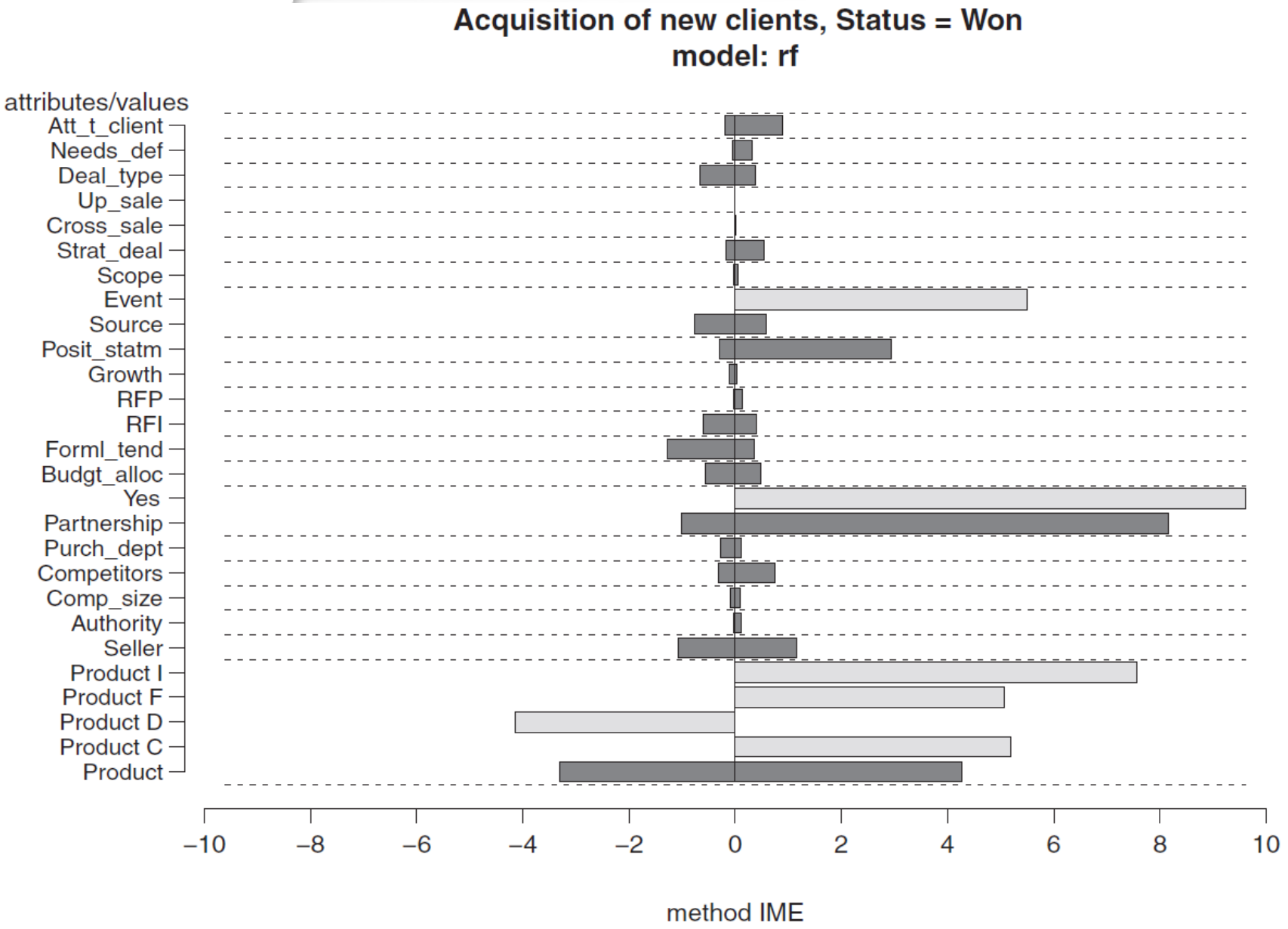
University of Ljubljana
Faculty of Computer and Information Science

B2B: what if

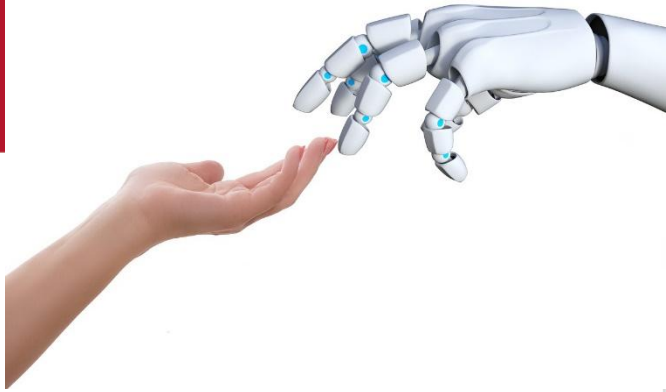


University of Ljubljana
Faculty of Computer and Information Science

B2B: change of distribution



University of Ljubljana
Faculty of Computer and Information Science



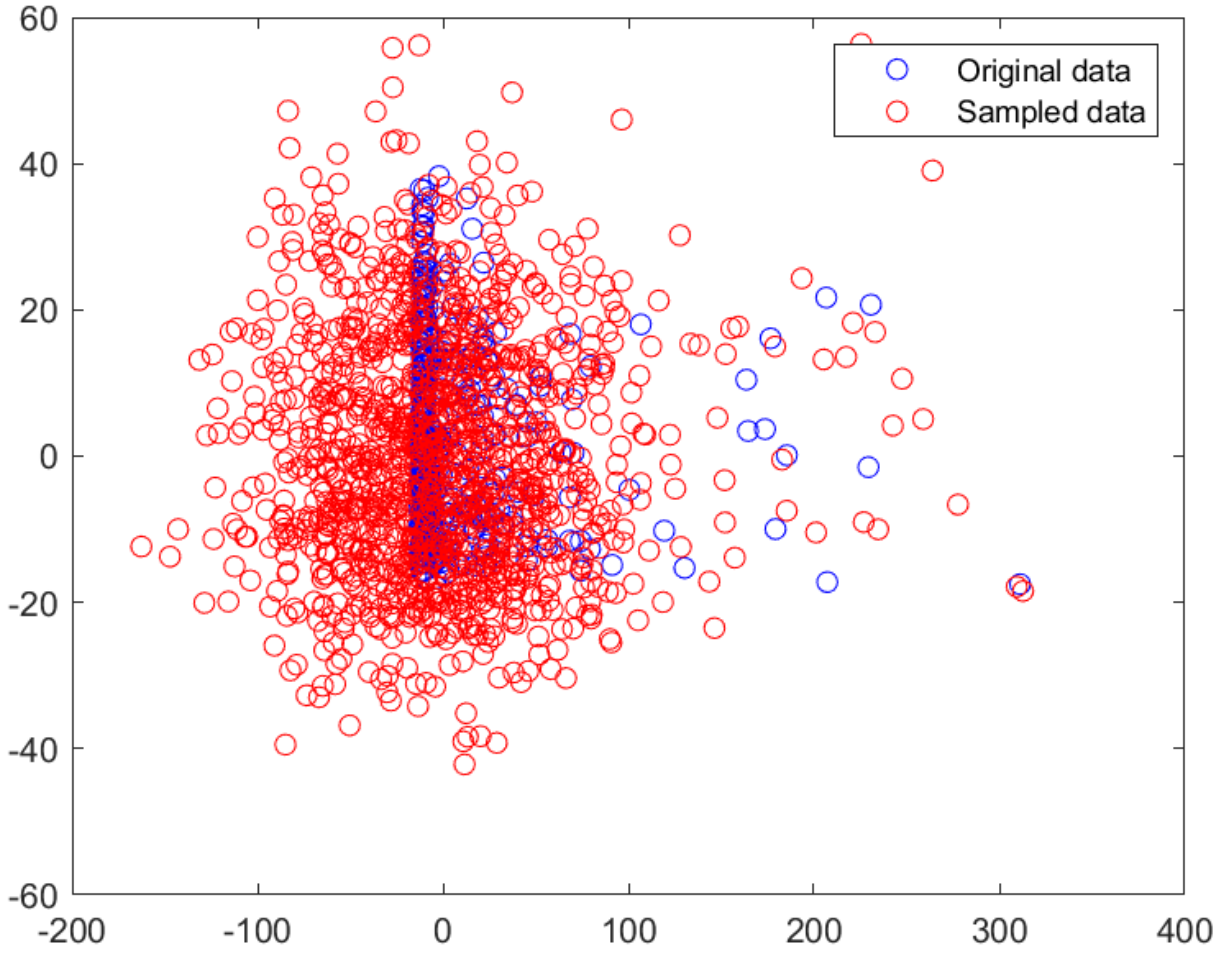
Lessons learned in B2B

- an effort needed to overcome the users' resistance
- human-in-the-loop is necessary to train, discuss, clean data, introduce explanations
- with an increased use, users gain trust in the methodology
- human mental models tend to be biased
- joint interactive approach beats both humans and ML models

University of Ljubljana
Faculty of Computer and Information Science

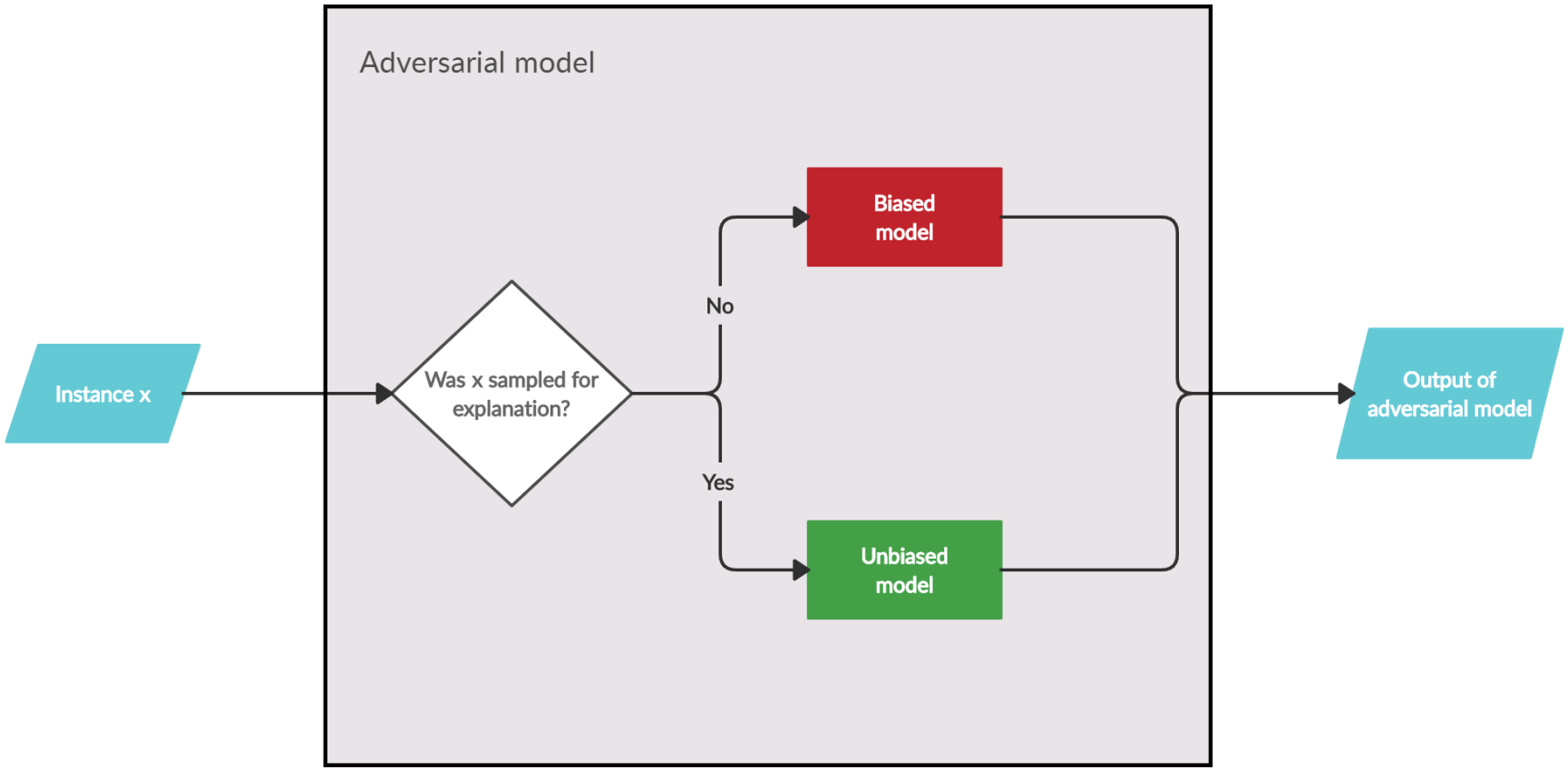
Attacks on explanations

- Poor sampling in explanation approaches makes them vulnerable
- Example: PCA based visualization of a part of the COMPAS dataset; the red dots were generated by LIME



University of Ljubljana
Faculty of Computer and Information Science

Dieselgate attacks on explanations



- Defence: better sampling

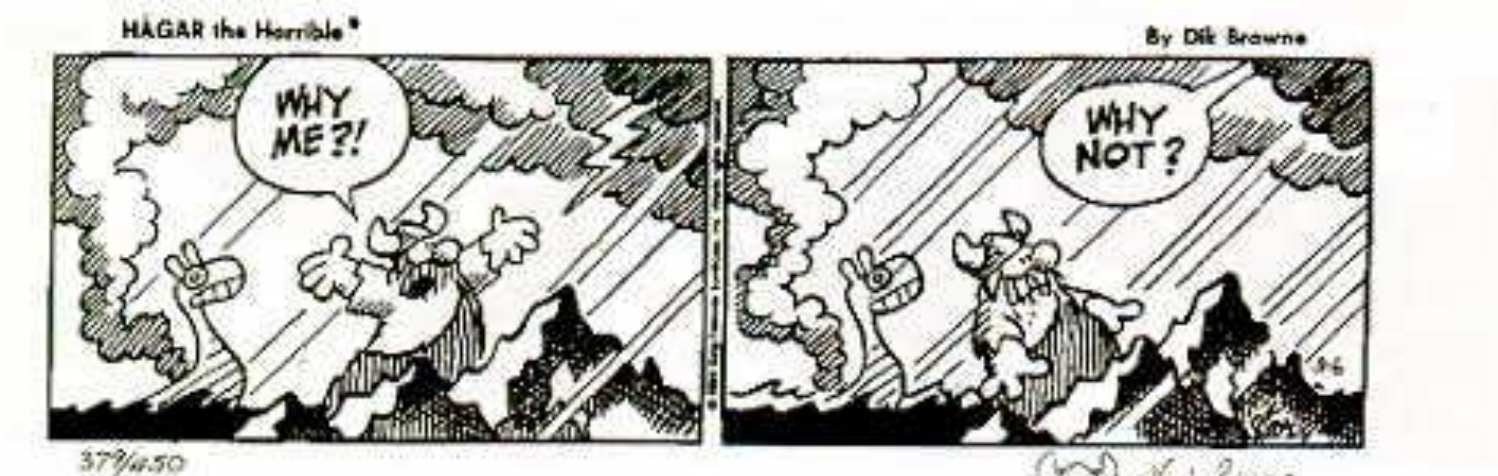
Domen Vreš and Marko Robnik-Šikonja, 2021. Better sampling in explanation methods can prevent dieselgate-like deception. *arXiv preprint arXiv:2101.11702*.

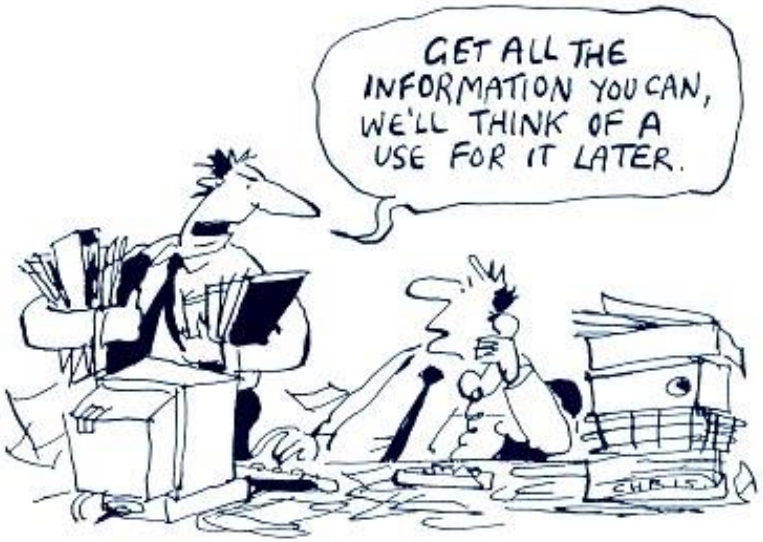
Opportunities

- better and more focused sampling
- better local explanation models
- interactions: detect and describe
- sequences: the order of attributes is important!
- images: decison areas, super-pixels
- better visualizations: human cognitive limitations
- explanations is also domain specific, we need explanation datasets

Summary of explanation approaches

- many successful mechanistic explanation approaches, mostly for tabular classification problems
- LLMs are trained to explain their behavior for particular important problems
- lots of opportunities for improvements
- even human explanations are not necessarily comprehensible
- humans often explain by providing background or additional knowledge
- legal and practical need for explanations of ML models

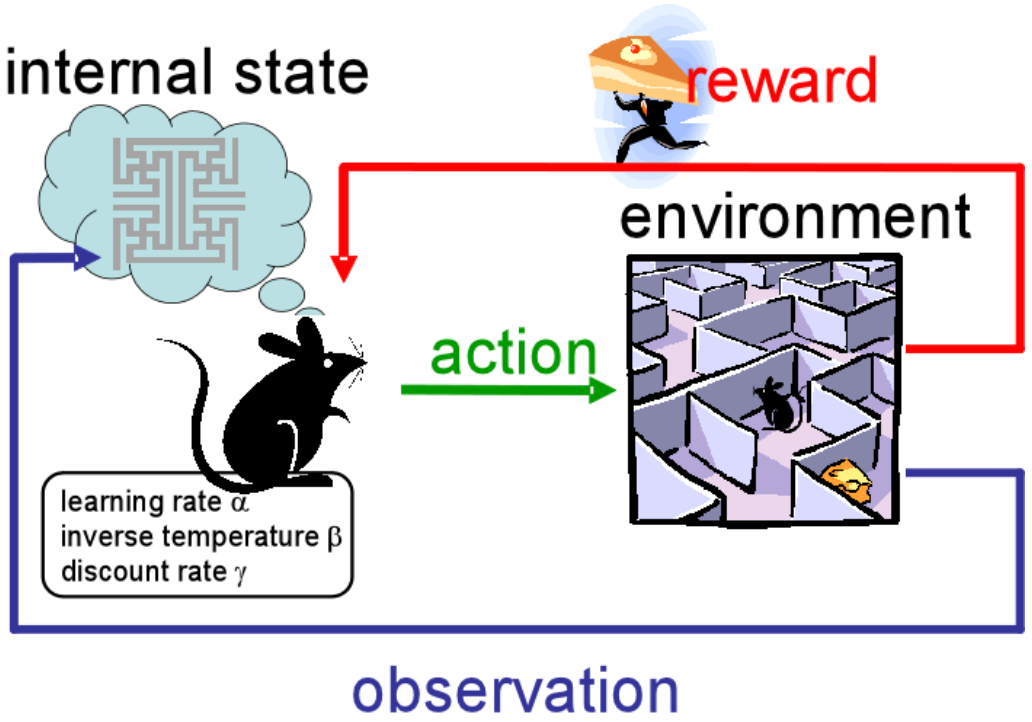




University of Ljubljana, Faculty of Computer and Information Science

Reinforcement learning





Prof Dr Marko Robnik-Šikonja
Intelligent Systems, Edition 2025

References

- R. S. Sutton and A. G. Barto: Reinforcement Learning: An Introduction, 2018, 2nd edition (the book is freely available)
- many papers, tutorials, online courses
- Recently, a revival due to deep reinforcement learning and the use of reinforcement learning in LLM preference alignment

some slides are courtesy of Andrew Barto, Peter Bodik and Lisa Torrey

Machine Learning

- Classification:
  - Given
    - Training data
  - Learn
    - A model for making a single prediction or decision



Animal/Human Learning

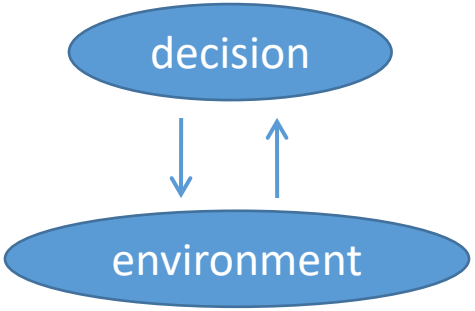
Memorization



Classification



Procedural



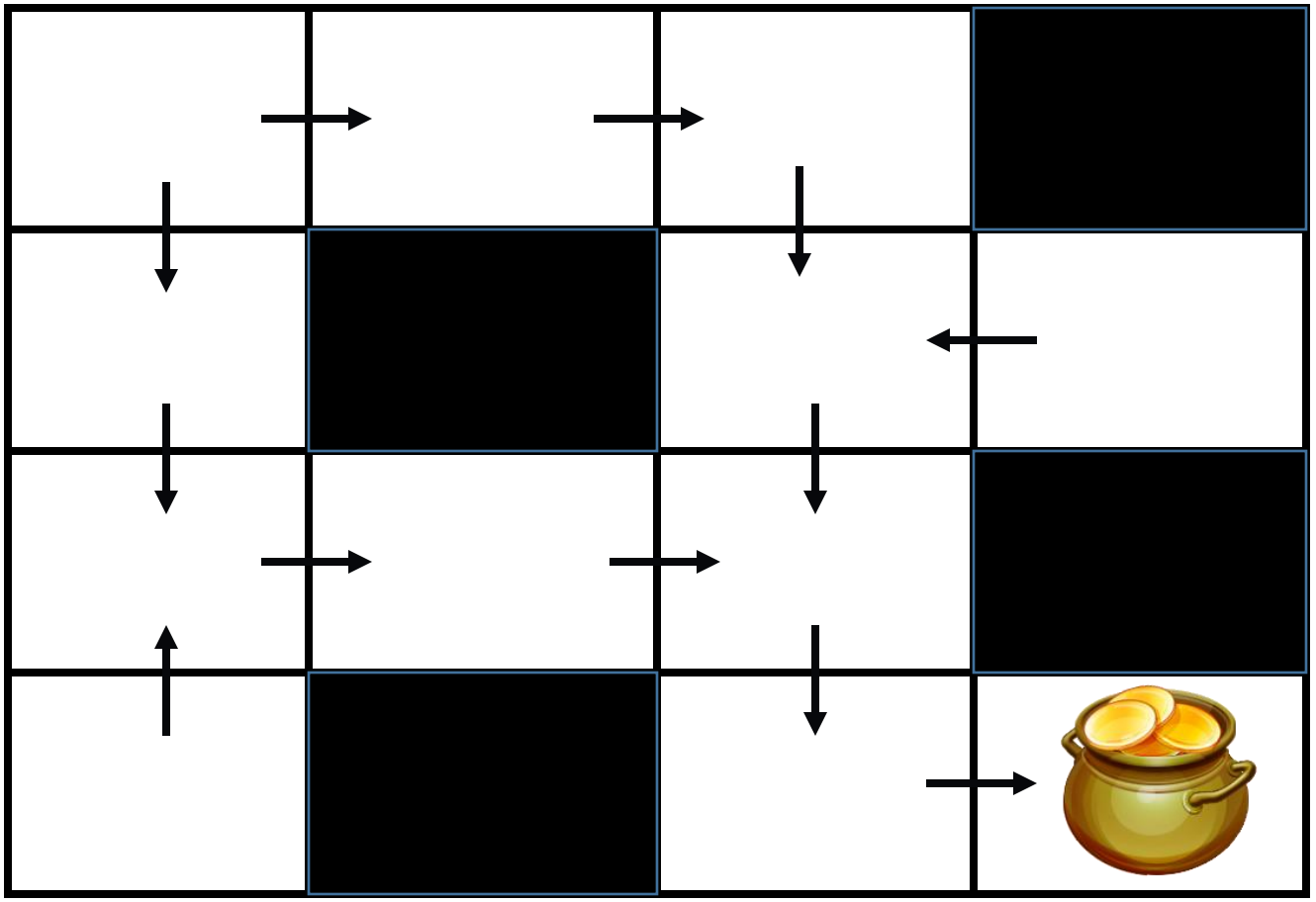
Other?

Procedural Learning

- Learning how to act to accomplish goals
  - Given
    - Environment that contains rewards
  - Learn
    - A policy for acting
- Important differences from classification
  - You don’t get examples of correct answers
  - You have to try things in order to learn

A Good Policy



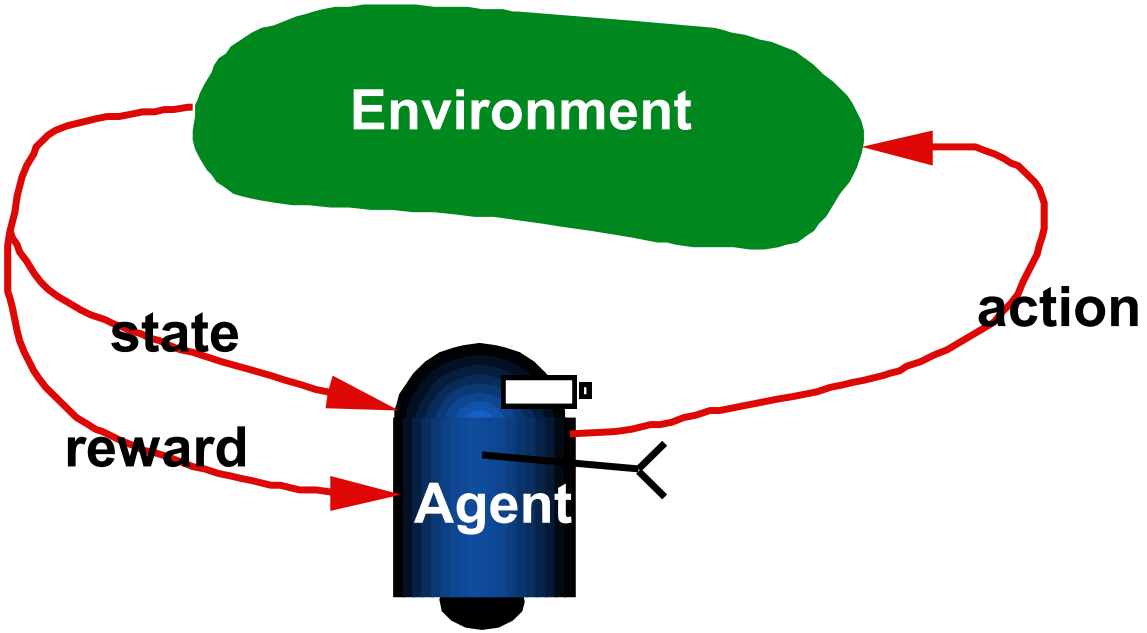


Introduction to Reinforcement Learning

- Reinforcement learning (RL)
- Questionable name stemming from behavioristic psychology (behavior reinforcement)
- Agent learning in the environment, performing actions
- Getting feedback from the environment (award, punishment), not necessary immediately
- Trying to learn a policy leading to goals
- An example: playing a game without knowing the rules; after 1000 moves, an opponent declares: You lost.

Agent

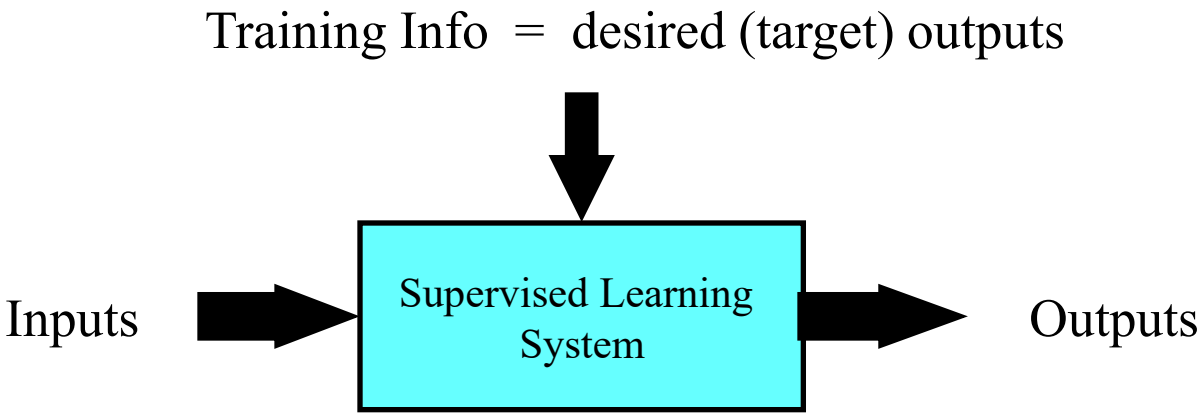
- Temporally situated
- Continual learning and planning
- Objective is to affect the environment
- Environment is stochastic and uncertain



What is Reinforcement Learning?

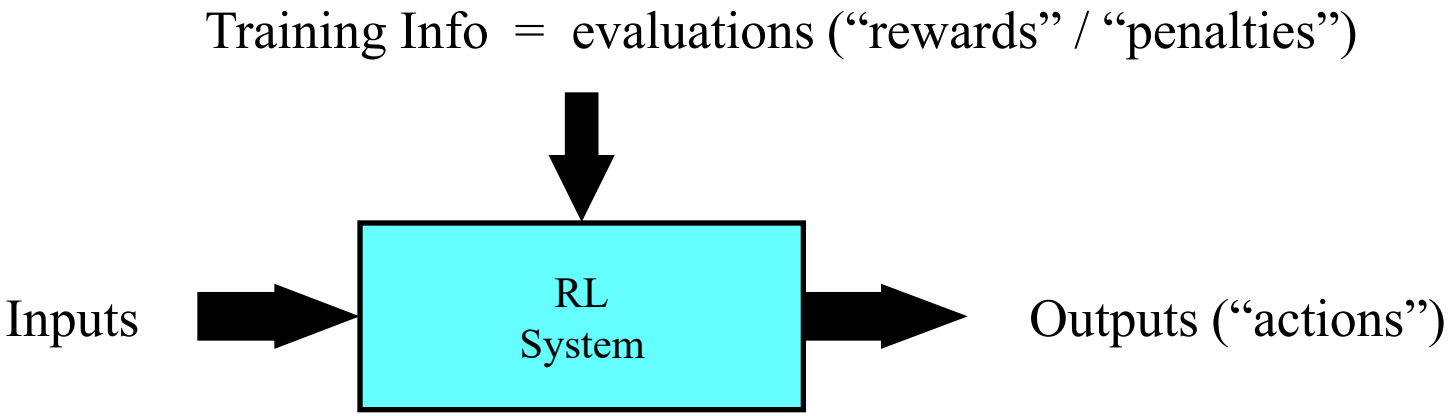
- Learning from interaction
- Goal-oriented learning: short-term and possible long-term awards
- Learning about, from, and while interacting with an external environment
- Learning what to do—how to map situations to actions—so as to maximize a numerical reward signal
- Agent discovers which action in what circumstances gives the highest award
- Agent can build a model of its environment
- RL is not supervised learning; it is about trial and error search, exploring, and getting information from the environment

Supervised Learning



Error = (target output – actual output)

Reinforcement Learning



Objective: get as much reward as possible

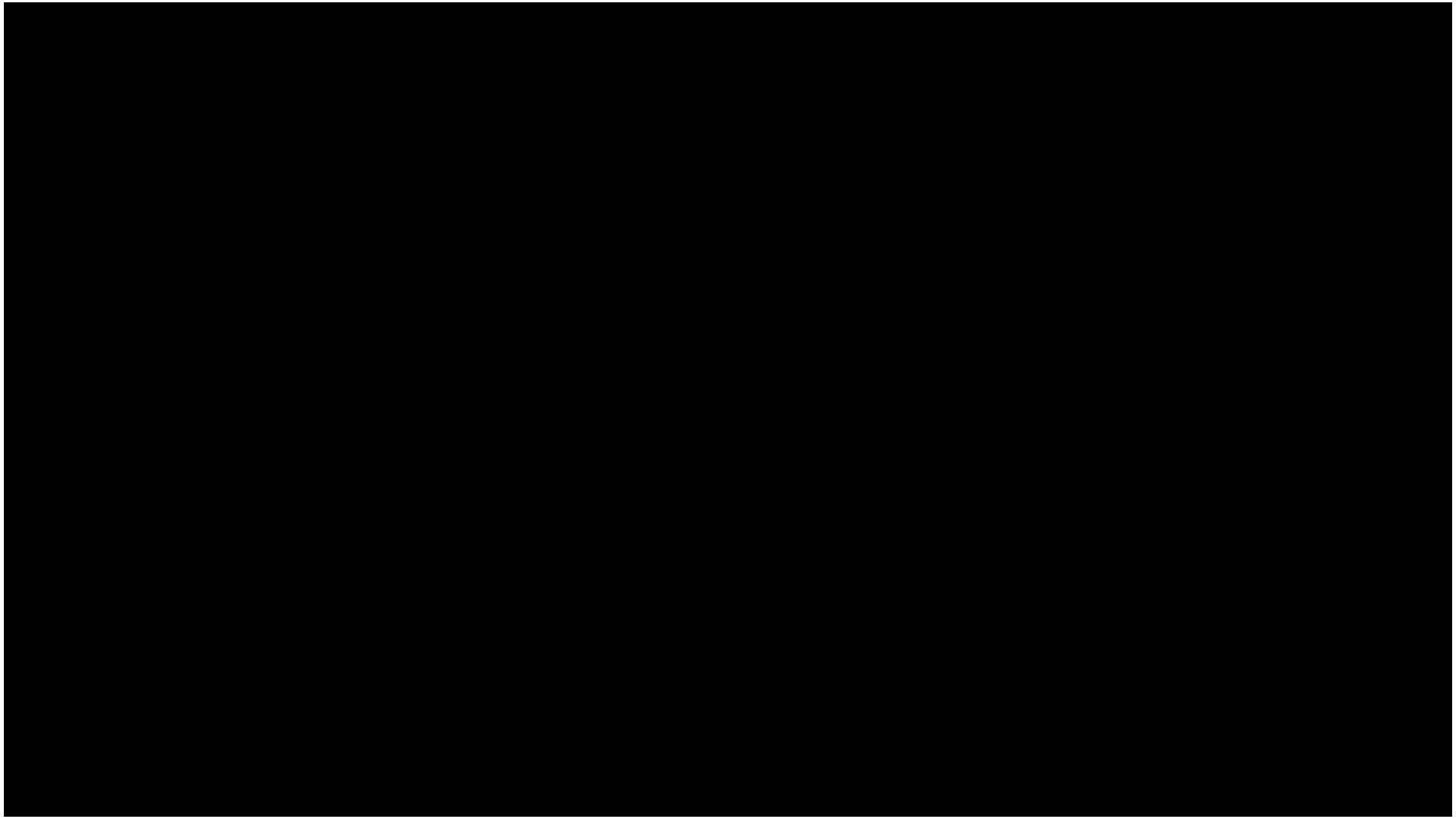
Key Features of RL

- Learner is not told which actions to take
- Trial-and-error search
- Possibility of delayed reward
  - Sacrifice short-term gains for greater long-term gains
- The need to ***explore*** and ***exploit***
- Considers the whole problem of a goal-directed agent interacting with an uncertain environment

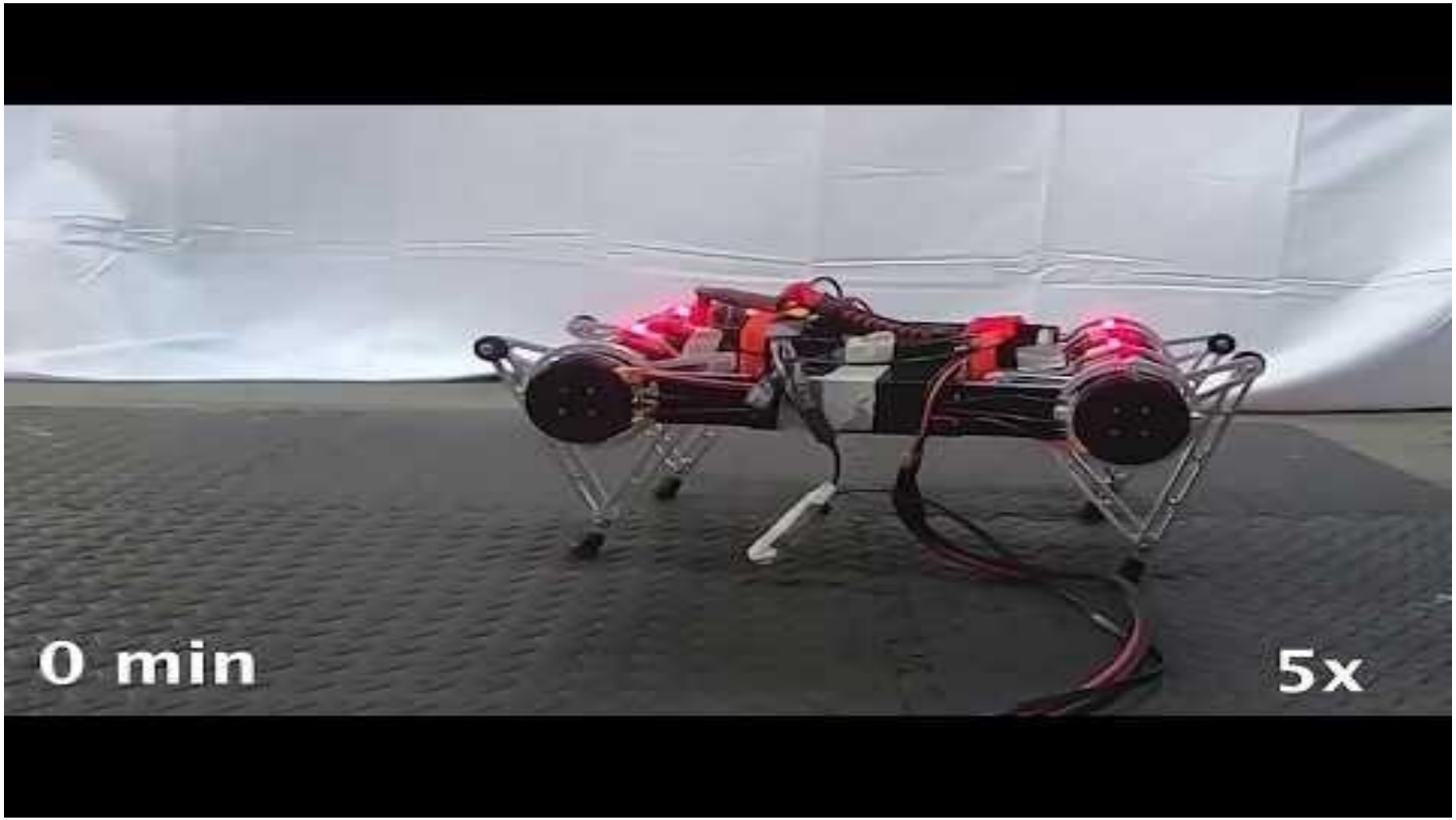
RL successful applications

- Robocup Soccer
- Financial asset management/Inventory management
- Dynamic Channel assignment in mobile communications
- Controlling elevators, industrial controllers, robots …
- Robots: navigation, grasping, moving …
- Games: backgammon (TD-Gammon, Jellyfish), Go (AlphaGo in combination with deep neural networks), Atari video games, poker, chess
- in LLMs: used in RLHF

Example video: Atari game Breakout



Example video:Robot training

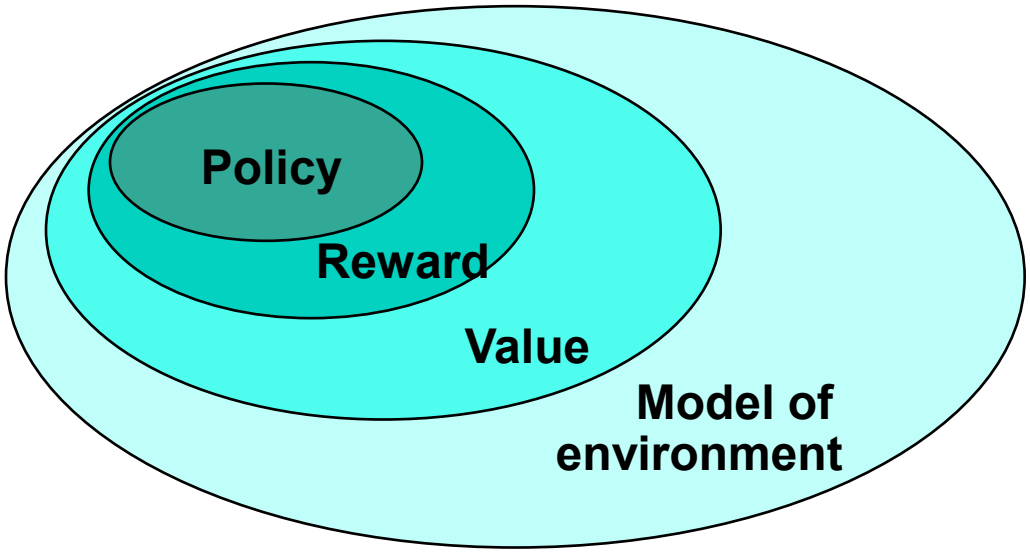


Tuomas Haarnoja, Sehoon Ha, Aurick Zhou, Jie Tan, George Tucker, Sergey Levine. Learning to Walk via Deep Reinforcement Learning. Robotics: Science and Systems (RSS). 2019.

Example video: Bipedals

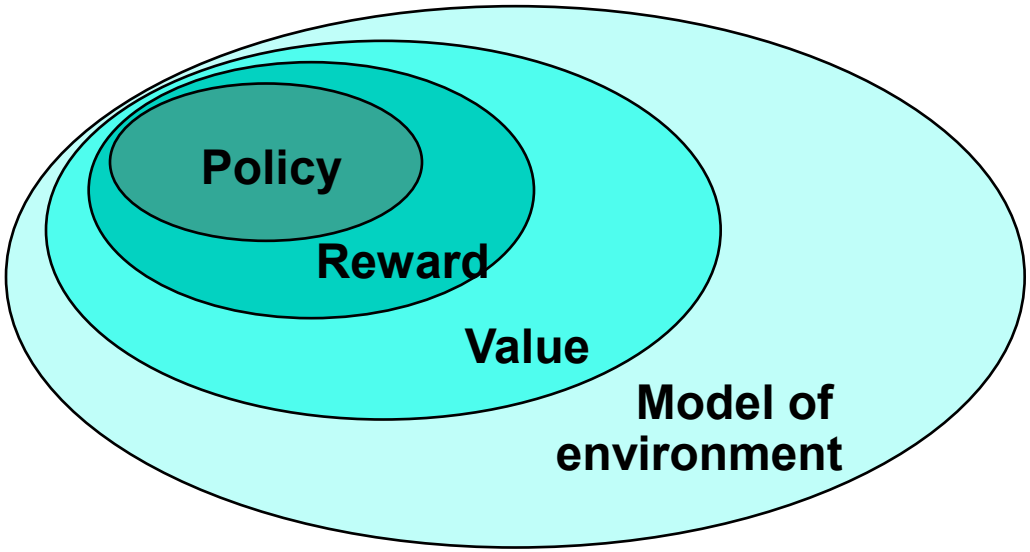


Components of RL 1/2



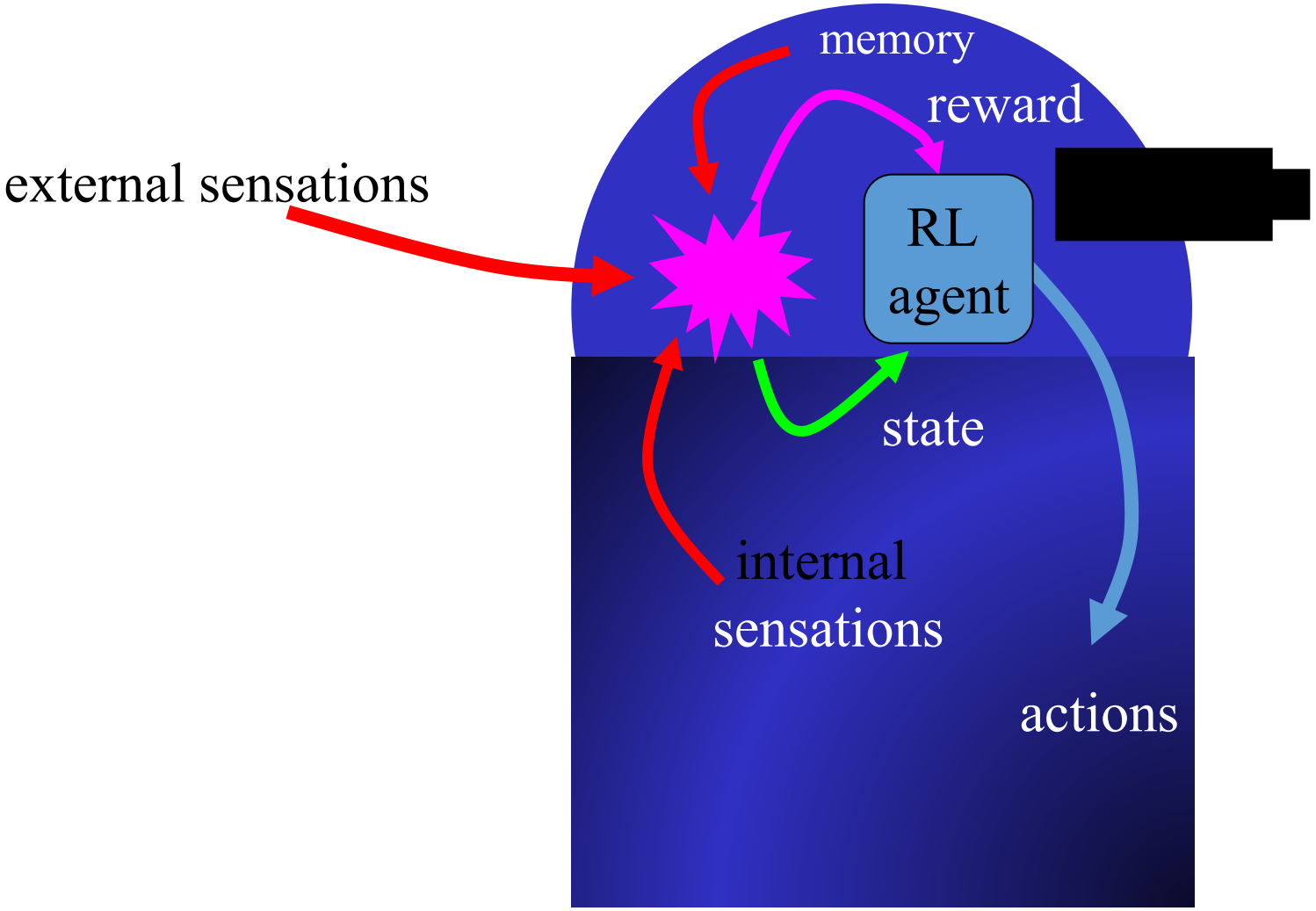
- **Policy**: what to do?
  - Defines agents choices and actions in a given time
  - Represented with rules, table, neural networks etc.
  - Result of search, planning, stochastic, etc.
- **Reward:** what is good?
  - Feedback from environment, agent tries to maximize it

Components of RL 2/2

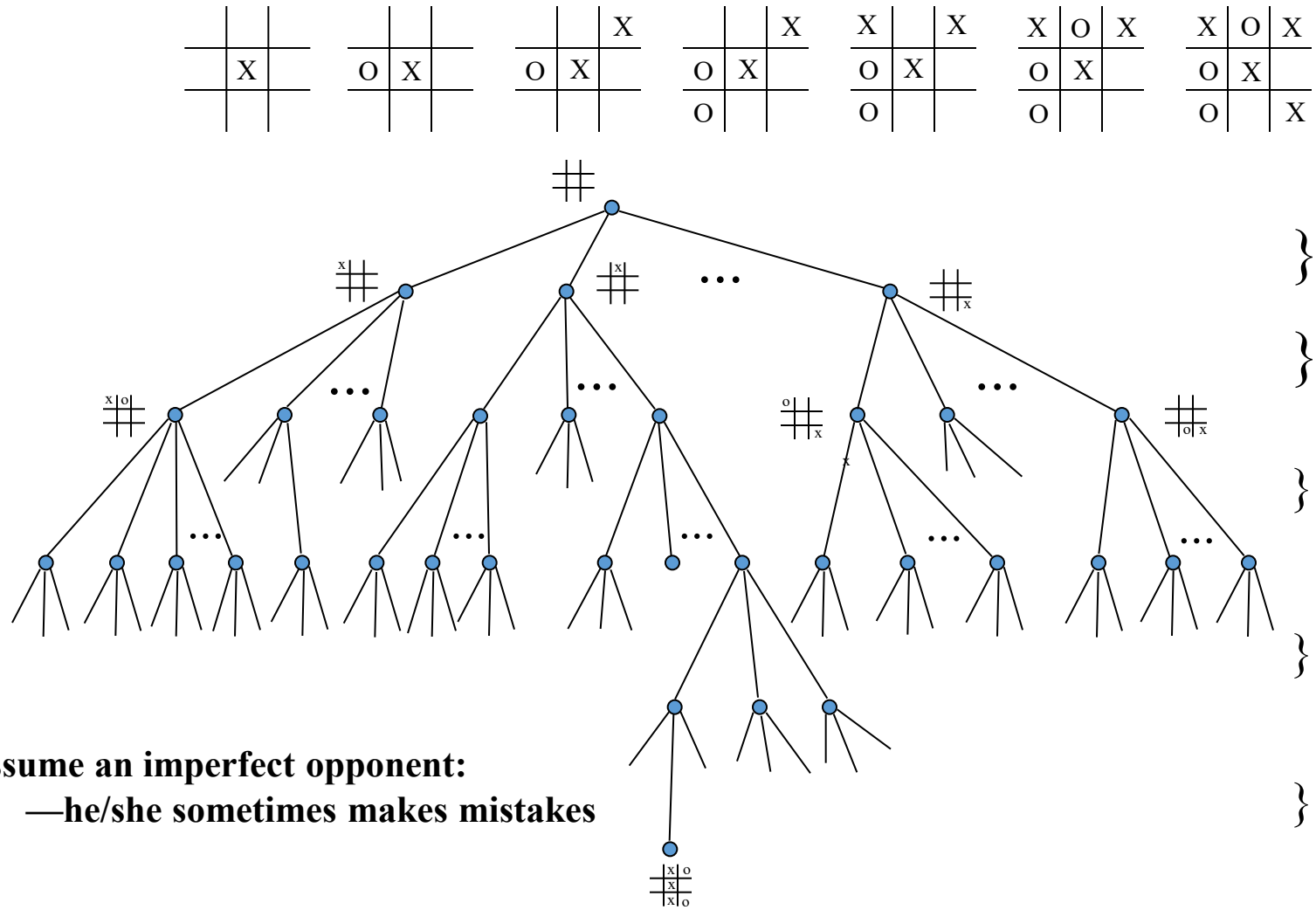


- **Value**: internal representation of what is good, it *predicts* reward
  - Agent's expectation of what can be expected in given state (long-term)
  - Implicitly contains evaluation of next states
  - Value has to be learned; use repetitions and sampling to estimate the value
- **Model**: what follows what
  - Internal representation of the environment
  - Agent can evaluate values and actions without performing them
  - Optional component

Agent from the point of view of RL



An Example: Tic-Tac-Toe



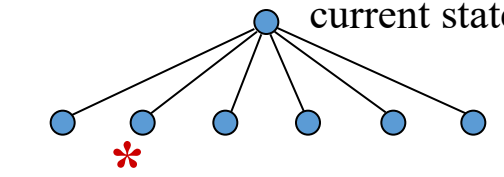
Assume an imperfect opponent:
—he/she sometimes makes mistakes

An RL Approach to Tic-Tac-Toe

1. Make a table with one entry per state:

State	$V(s)$ – estimated probability of winning	
	.5	?
	.5	?
⋮	⋮	
	1	win
⋮	⋮	
	0	loss
⋮	⋮	
	0	draw

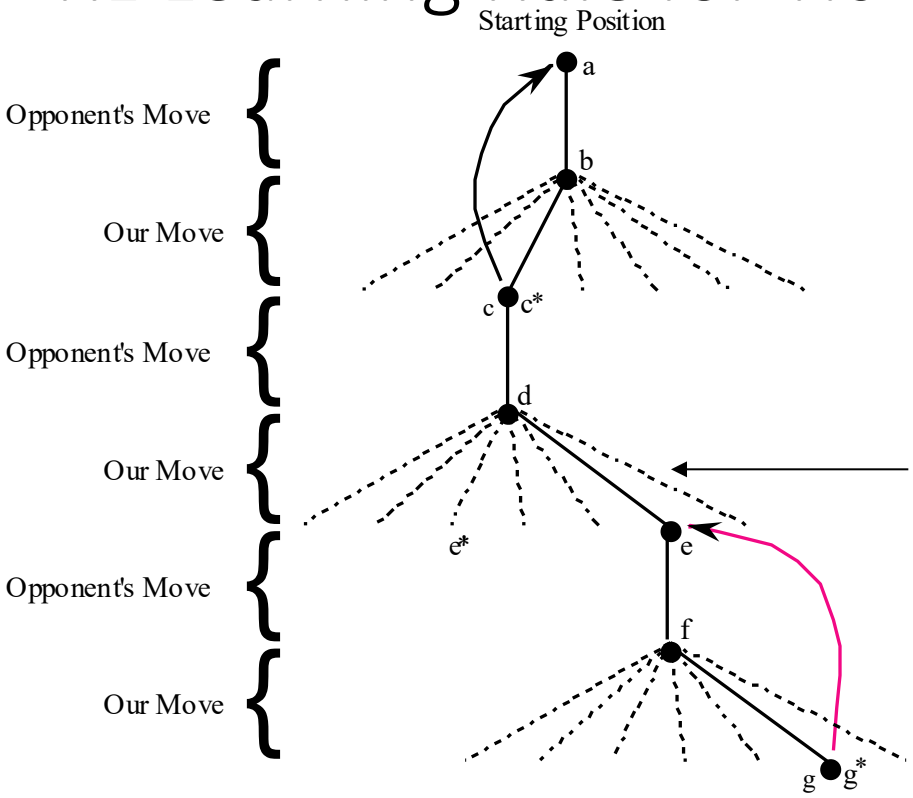
2. Now play lots of games.
To pick our moves,
look ahead one step:



Just pick the next state with the highest estimated prob. of winning — the largest $V(s)$; a ***greedy*** move.

But 10% of the time pick a move at random; an ***exploratory move***.

RL Learning Rule for Tic-Tac-Toe



"Exploratory" move

s – the state before our greedy move

s' – the state after our greedy move

We increment each $V(s)$ toward $V(s')$ – a ***backup***:

$$V(s) \leftarrow V(s) + \alpha\left[V(s') - V(s)\right]$$

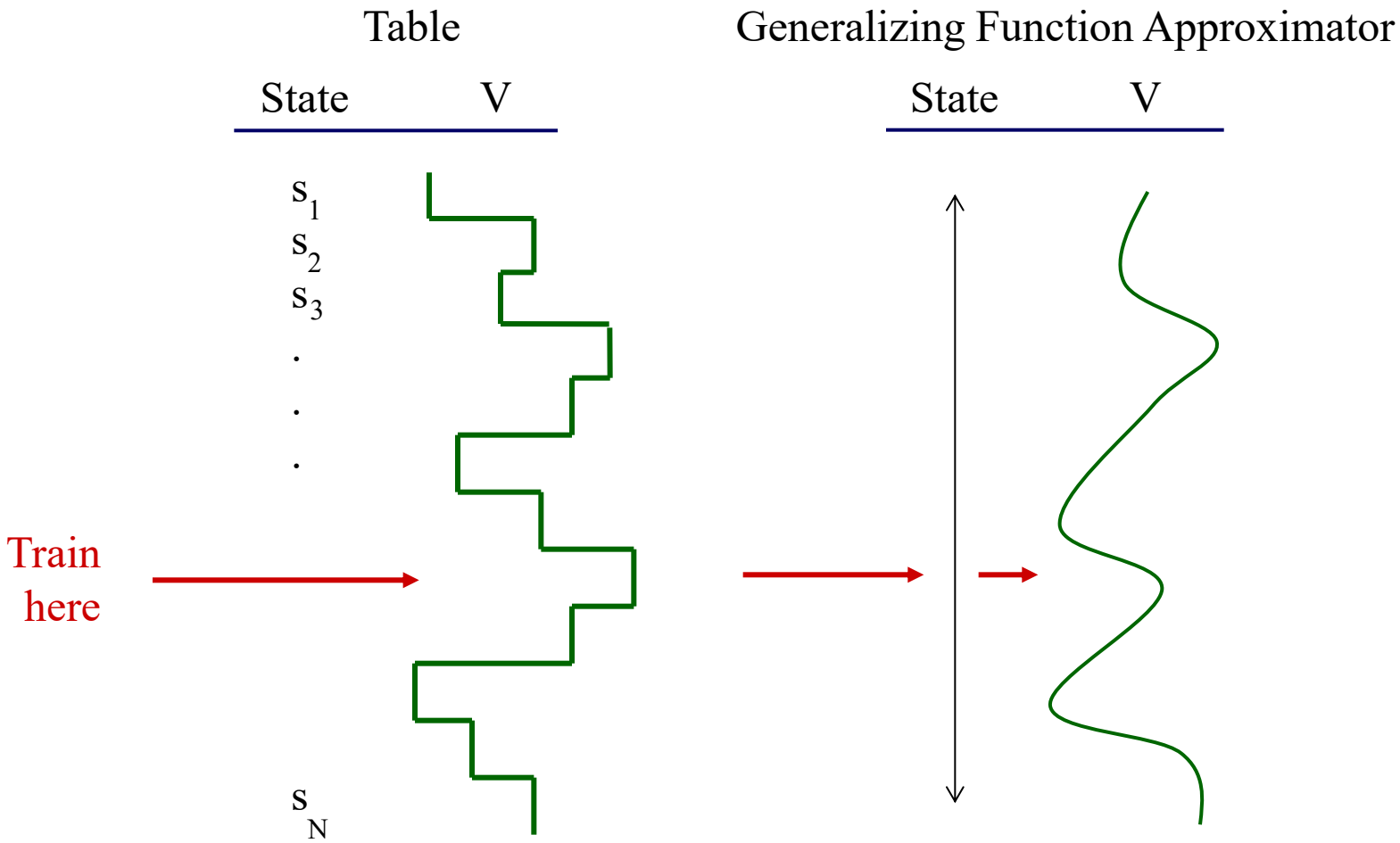
a small positive fraction, e.g., $\alpha = .1$

the ***step-size parameter***

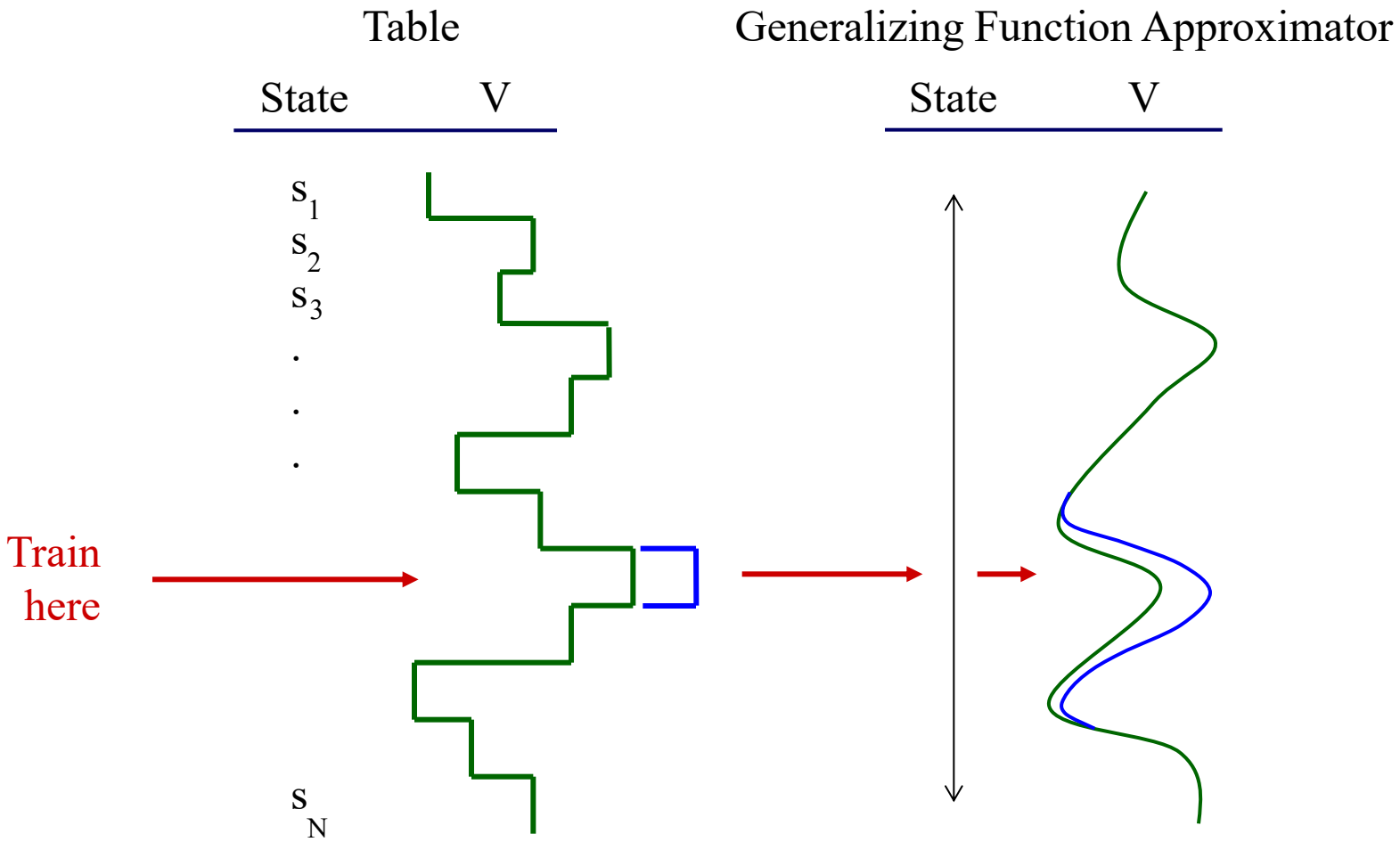
How can we improve this TTT player?

- Take advantage of symmetries
  - representation/generalization
  - How might this backfire?
- Do we need “random” moves? Why?
  - Do we always need a full 10%?
- Can we learn from “random” moves?
- Can we learn offline?
  - Pre-training from self play?
  - Using learned models of opponent?
- . . .

E.g., generalization



E.g., generalization

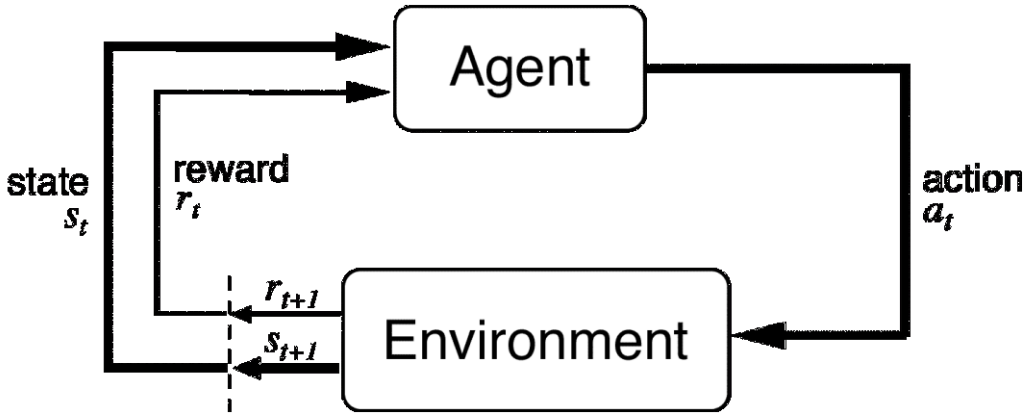


Tic-Tac-Toe is just a toy example

- Finite, small number of states
- One-step look-ahead is always possible
- State completely observable
- . . .

- RL is not limited to a finite number of states; in problems with infinite or very large number of states we only generate states encountered during search
- RL is not limited to games or opponent's response

The Agent-Environment Interface



Agent and environment interact at discrete time steps: $t = 0, 1, 2, \ldots$

Agent observes state at step t: $s_t \in S$

produces action at step t: $a_t \in A(s_t)$

gets resulting reward: $r_{t+1} \in \Re$

and resulting next state: s_{t+1}

$$\ldots \; s_t \xrightarrow[a_t]{r_{t+1}} s_{t+1} \xrightarrow[a_{t+1}]{r_{t+2}} s_{t+2} \xrightarrow[a_{t+2}]{r_{t+3}} s_{t+3} \xrightarrow{a_{t+3}} \ldots$$

The Agent Learns a Policy

Policy at step t, π_t :

a mapping from states to action probabilities

$\pi_t(s,a) =$ probability that $a_t = a$ when $s_t = s$

- Reinforcement learning methods specify how the agent changes its policy as a result of experience.
- Roughly, the agent's goal is to get as much reward as it can over the long run.
- How to represent a policy? With a table, function, ML model, etc.

Getting the degree of abstraction right

- Time steps need not refer to fixed intervals of real time.
- Actions can be low level (e.g., voltages to motors), or high level (e.g., accept a job offer), “mental” (e.g., shift in focus of attention),
- States can be low-level “sensations”, or they can be abstract, symbolic, based on memory, or subjective (e.g., the state of being “surprised” or “lost”).
- An RL agent is not like a *whole* animal or robot.
- Reward computation is in the agent’s environment because the agent cannot change it arbitrarily.
- The environment is not necessarily unknown to the agent, only incompletely controllable.

Goals and Rewards

- Is a scalar reward signal an adequate notion of a goal?—maybe not, but it is surprisingly flexible.
- A goal should specify **what** we want to achieve, not **how** we want to achieve it.
- A goal must be outside the agent's direct control—thus outside the agent.
- The agent must be able to measure success:
  - explicitly;
  - frequently during its lifespan.
- Human learning: we often get considerable better feedback on errors (not just a numeric score)

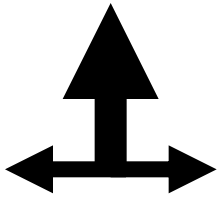
Robot in a room

			+1
			-1
START			

actions: UP, DOWN, LEFT, RIGHT

UP

80% move UP
10% move LEFT
10% move RIGHT

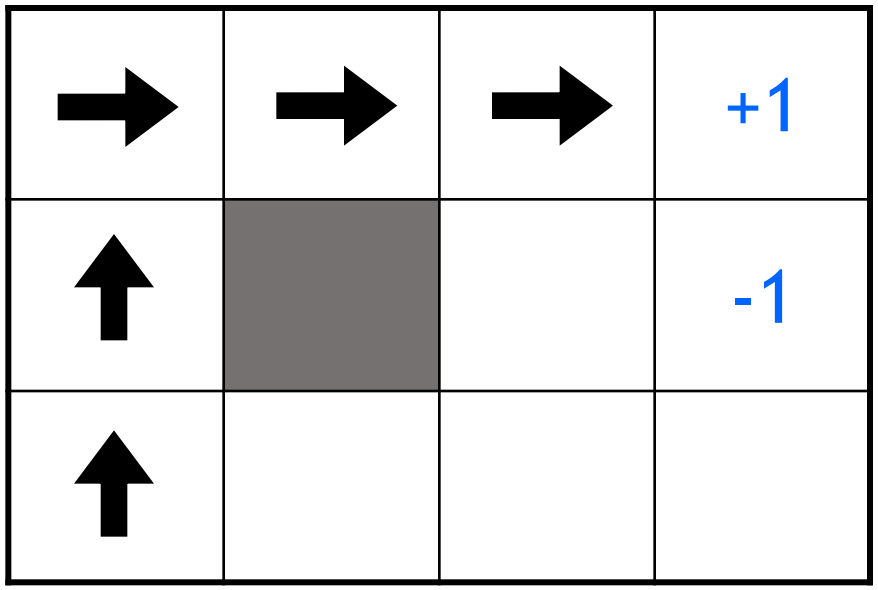


reward +1 at [4,3], -1 at [4,2]
reward -0.04 for each step

- states
- actions
- rewards

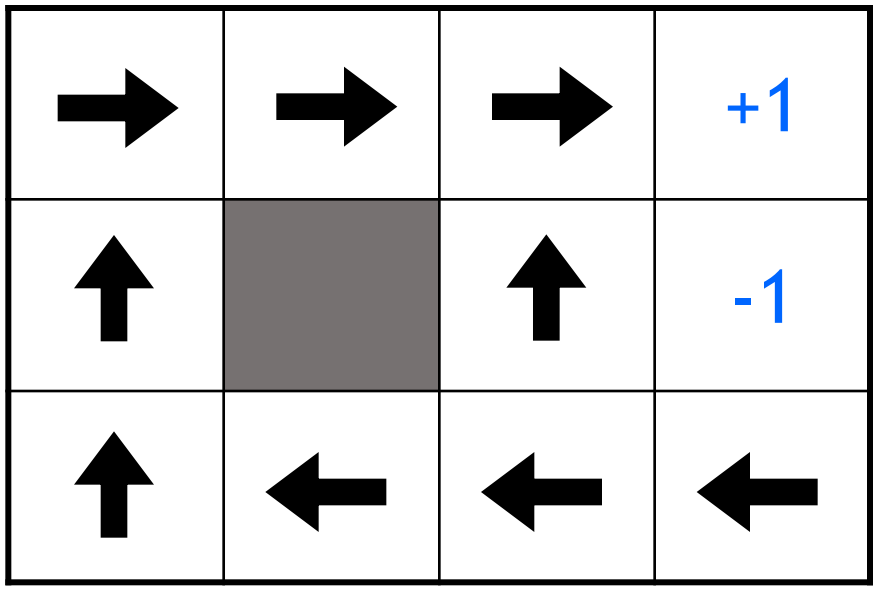
- what is the solution?

Is this a solution?

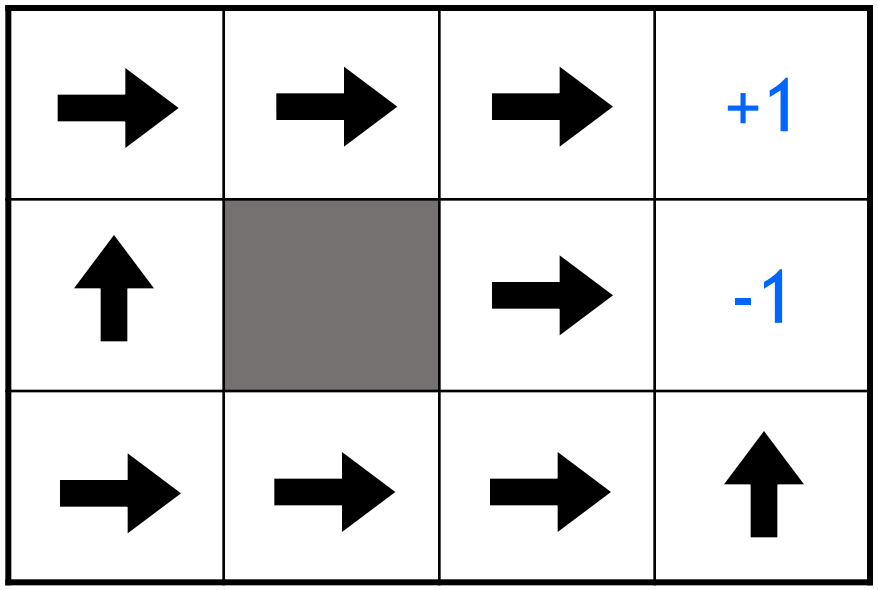


- only if actions are deterministic
  - not in this case (actions are stochastic)
- solution/policy
  - mapping from each state to an action

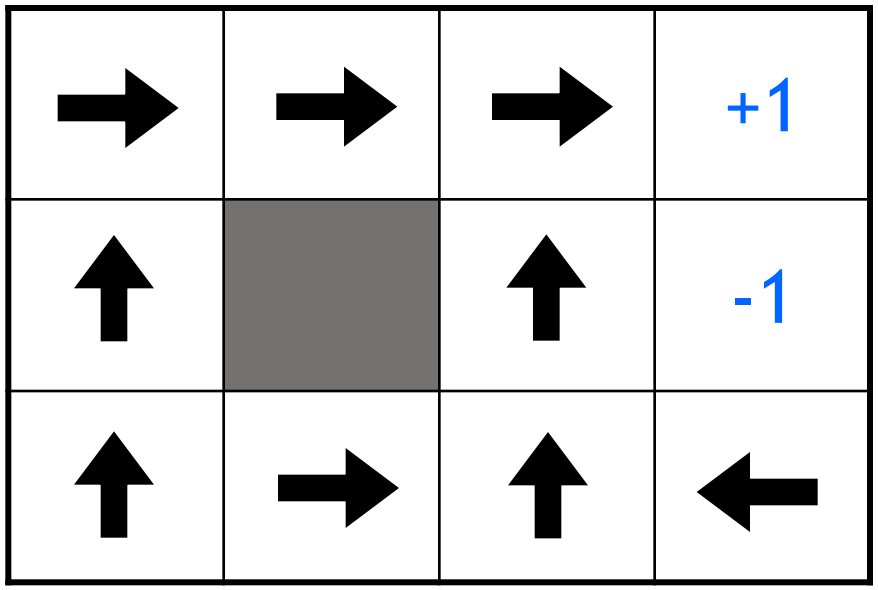
Optimal policy



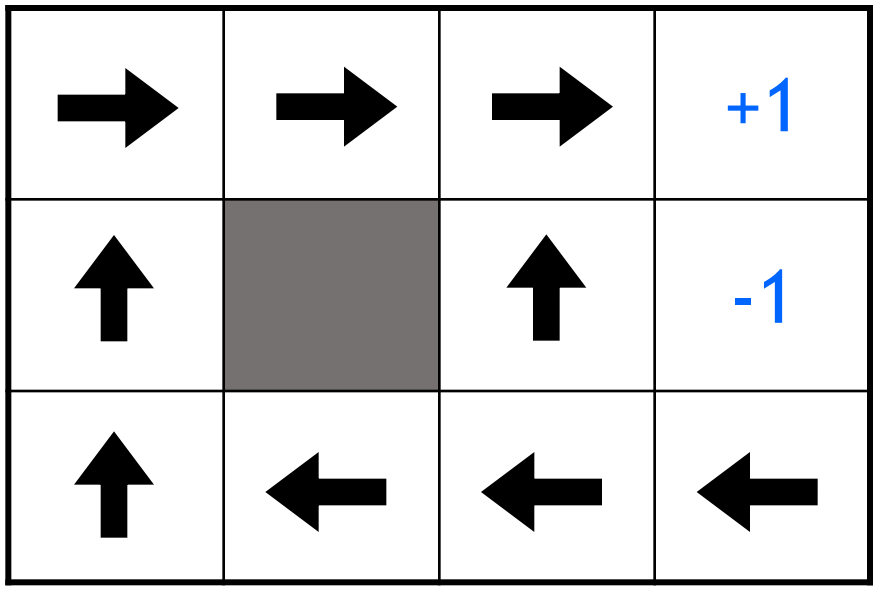
Reward for each step -2



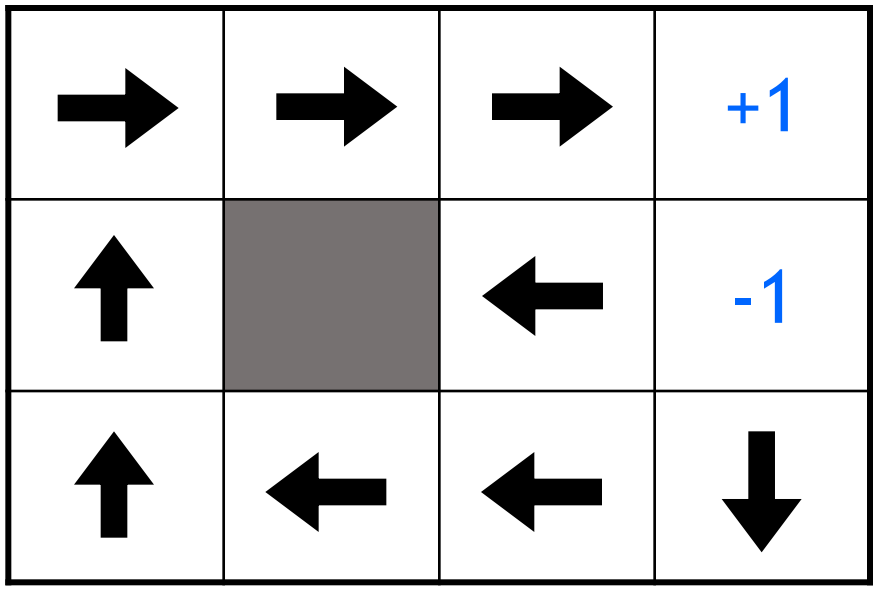
Reward for each step: -0.1



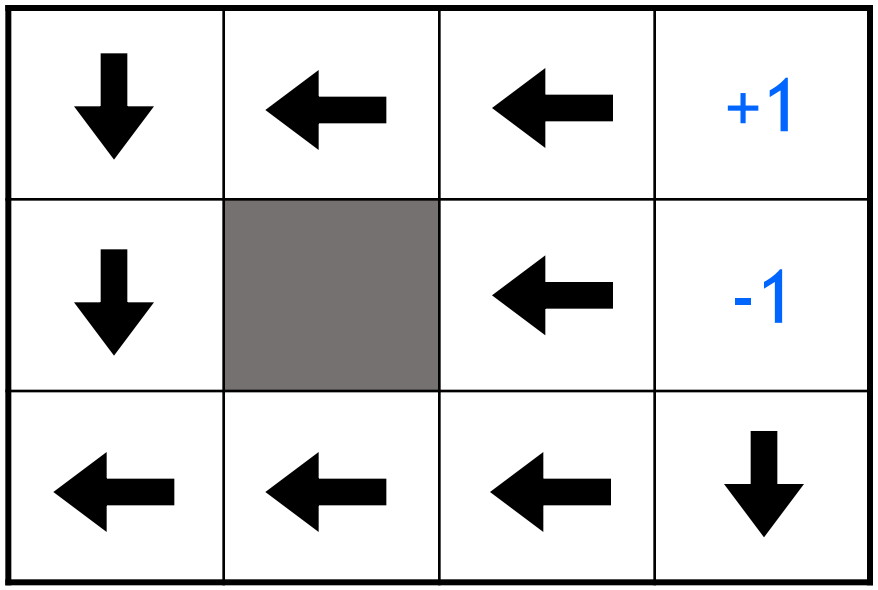
Reward for each step: -0.04



Reward for each step: -0.01



Reward for each step: +0.01



Returns

Suppose the sequence of rewards after step t is :

$$r_{t+1}, r_{t+2}, r_{t+3}, \cdots$$

What do we want to maximize?

In general,

we want to maximize the **expected return**, $E\{R_t\}$, for each step t.

Episodic tasks: interaction breaks naturally into episodes, e.g., plays of a game, trips through a maze.

$$R_t = r_{t+1} + r_{t+2} + \cdots + r_T,$$

where T is a final time step at which a **terminal state** is reached, ending an episode.

Returns for Continuing Tasks

Continuing tasks: interaction does not have natural episodes.

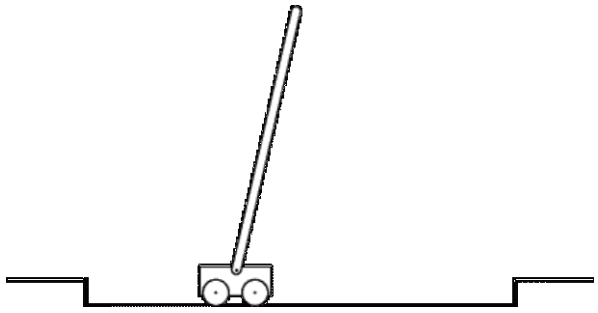
Discounted return:

$$R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1},$$

where $\gamma, 0 \leq \gamma \leq 1,$ is the **discount rate**.

$$\text{shortsighted } 0 \leftarrow \gamma \rightarrow 1 \text{ farsighted}$$

An example: cart and pole



Avoid **failure:** the pole falling beyond a critical angle or the cart hitting end of track.

As an **episodic task** where episode ends upon failure:

$$\text{reward} = +1 \text{ for each step before failure}$$

$$\Rightarrow \text{ return} = \text{number of steps before failure}$$

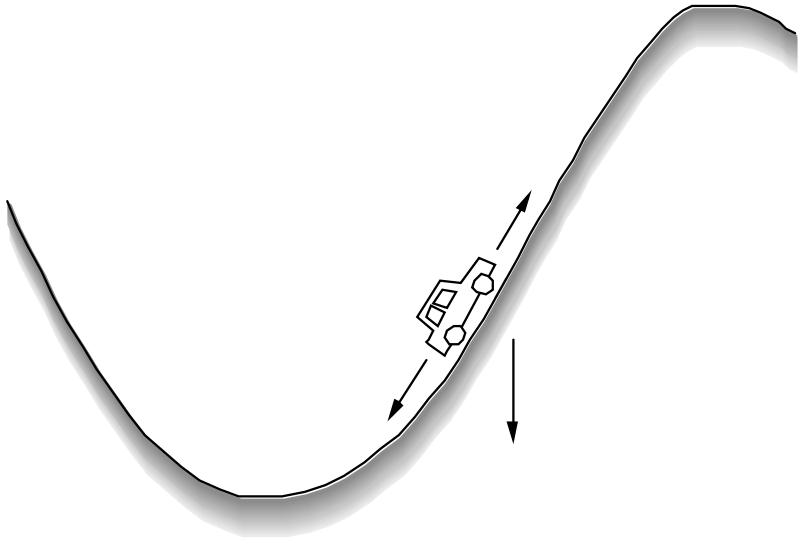
As a **continuing task** with discounted return:

$$\text{reward} = -1 \text{ upon failure; } 0 \text{ otherwise}$$

$$\Rightarrow \text{ return} = -\gamma^k, \text{ for } k \text{ steps before failure}$$

In either case, return is maximized by avoiding failure for as long as possible.

Another Example



Get to the top of the hill
as quickly as possible.

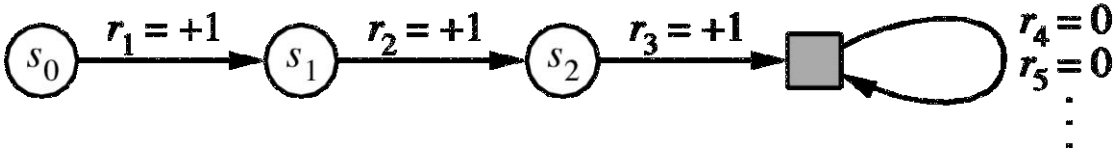
reward $= -1$ for each step where **not** at top of hill

$\Rightarrow$ return $=$ $-$ number of steps before reaching top of hill

Return is maximized by minimizing
number of steps to reach the top of the hill.

A Unified Notation

- In episodic tasks, we number the time steps of each episode starting from zero.
- We usually do not have to distinguish between episodes, so we write s_t instead of $s_{t,j}$ for the state at step *t* of episode *j*.
- Think of each episode as ending in an absorbing state that always produces reward of zero:



- We can cover all cases by writing

$$R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1},$$

where γ can be 1 only if a zero reward absorbing state is always reached.

Episodic task – finite horizon

- In time *t,* the agent is interested in *h* further states
- Rewards in that time are r_{t+1}, r_{t+2}, r_{t+3}, …, r_{t+h}

  $R_t = r_{t+1} + r_{t+2} + r_{t+3} + \ldots + r_{t+h}$
- The agent maximizes expected reward in that period

$$\max E(R_t) = \max E(\sum_{k=1}^{h} r_{t+k})$$

Finite horizon

- Two optimal behaviors
  - h-step optimal action: on step 1, do an action which is optimal under assumption that h-1 actions will follow, on step 2 do an action which is optimal under assumption that h-2 actions will follow ...
  - h-step receding action: on each step do an action which is optimal under assumption that h actions will follow
- Limited look-ahead
- Suitability of finite horizon: episodic missions (e.g., labyrinth)

Continuous tasks

- No natural end, but …

  … nearer actions are more important than more distant ones

- Agent optimizes infinite sequence of rewards
- Rewards are geometrically discounted
- rewards: $R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \gamma^3 r_{t+4,}$ ... for $0 < \gamma < 1$
- γ (discount factor) can be interpreted as interest rate, a trick to bound an infinite sum, probability of surviving another step, short/far-sightedness

$$\max E(\sum_{k=0}^{\infty} \gamma^k r_{t+k+1}), \quad 0 < \gamma < 1$$

Average reward model

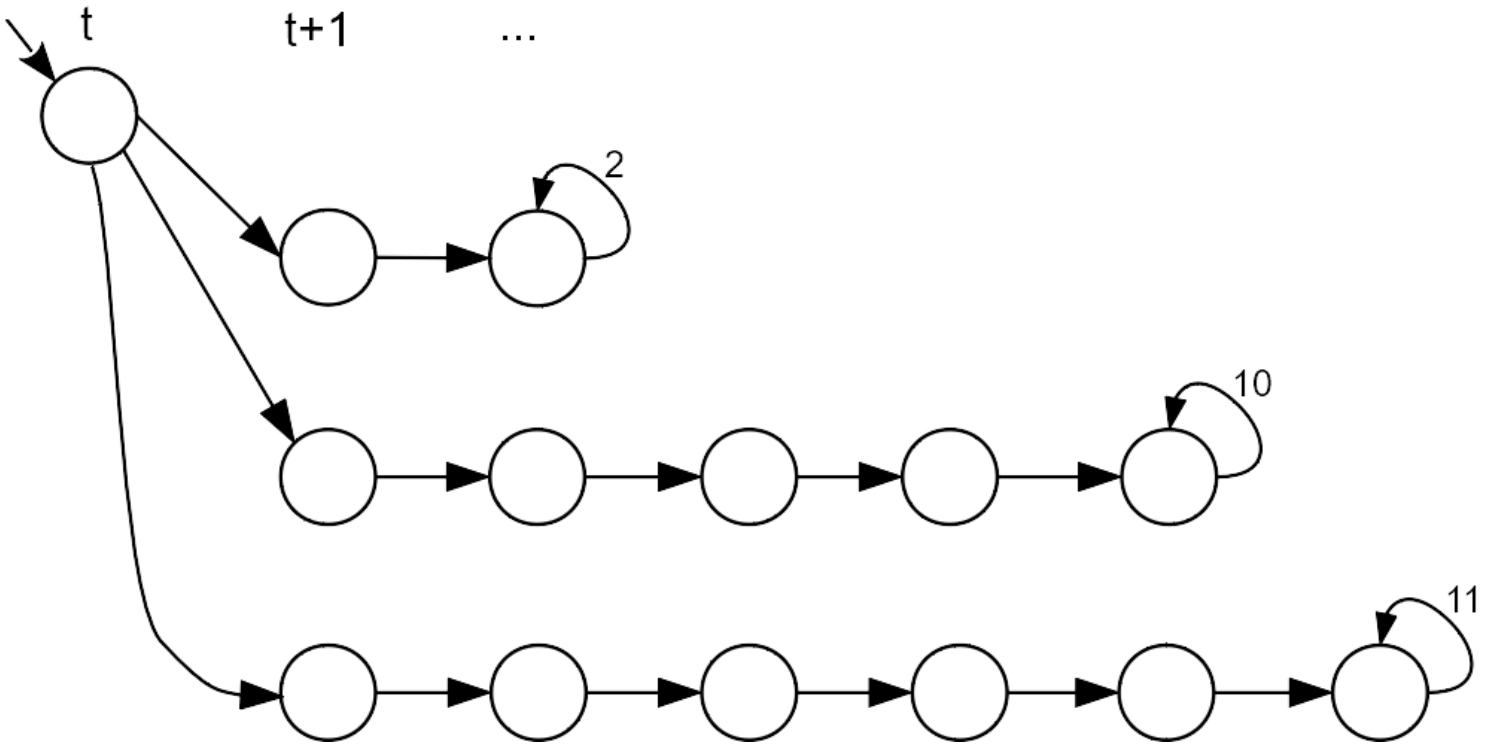
- Agent optimizes long-term average reward

$$\lim_{h\to\infty} E\left(\frac{1}{h}\sum_{k=1}^{h} r_{t+k}\right)$$

- Downside: does not know the difference between near and distant rewards

An example: rewards

1. finite horizont, h=4
2. infinite horizont, γ=0.9
3. average expected reward



The Markov Property

- By "the state" at step *t*, we mean whatever information is available to the agent at step *t* about its environment.
- The state can include immediate "sensations," highly processed sensations, and structures built up over time from sequences of sensations.
- Ideally, a state should summarize past sensations so as to retain all "essential" information, i.e. it should have the **Markov Property**:

$$\Pr\{s_{t+1}=s', r_{t+1}=r \mid s_t, a_t, r_t, s_{t-1}, a_{t-1}, \ldots, r_1, s_0, a_0\} =$$

$$\Pr\{s_{t+1}=s', r_{t+1}=r \mid s_t, a_t\}$$

for all s', r, and histories $s_t, a_t, r_t, s_{t-1}, a_{t-1}, \ldots, r_1, s_0, a_0$.

Markov Decision Processes

- If a reinforcement learning task has the Markov Property, it is basically a **Markov Decision Process (MDP)**.
- If state and action sets are finite, it is a **finite MDP**.
- To define a finite MDP, you need to give:
  - **state and action sets**
  - one-step "dynamics" defined by **transition probabilities**:

$$\mathbf{P}_{ss'}^{a} = \Pr\left\{ s_{t+1} = s' \,\middle|\, s_t = s, a_t = a \right\} \quad \text{for all } s, s' \in S,\ a \in A(s).$$

  - **reward probabilities**:

$$\mathbf{R}_{ss'}^{a} = E\left\{ r_{t+1} \,\middle|\, s_t = s, a_t = a, s_{t+1} = s' \right\} \quad \text{for all } s, s' \in S,\ a \in A(s).$$

An Example of Finite MDP

Recycling Robot

- At each step, robot has to decide whether it should (1) actively search for a can, (2) wait for someone to bring it a can, or (3) go to home base and recharge.
- Searching is better but runs down the battery; if runs out of power while searching, has to be rescued (which is bad).
- Decisions made on basis of current energy level: `high`, `low`.
- Reward = number of cans collected

Recycling Robot MDP

$S = \{\texttt{high}, \texttt{low}\}$

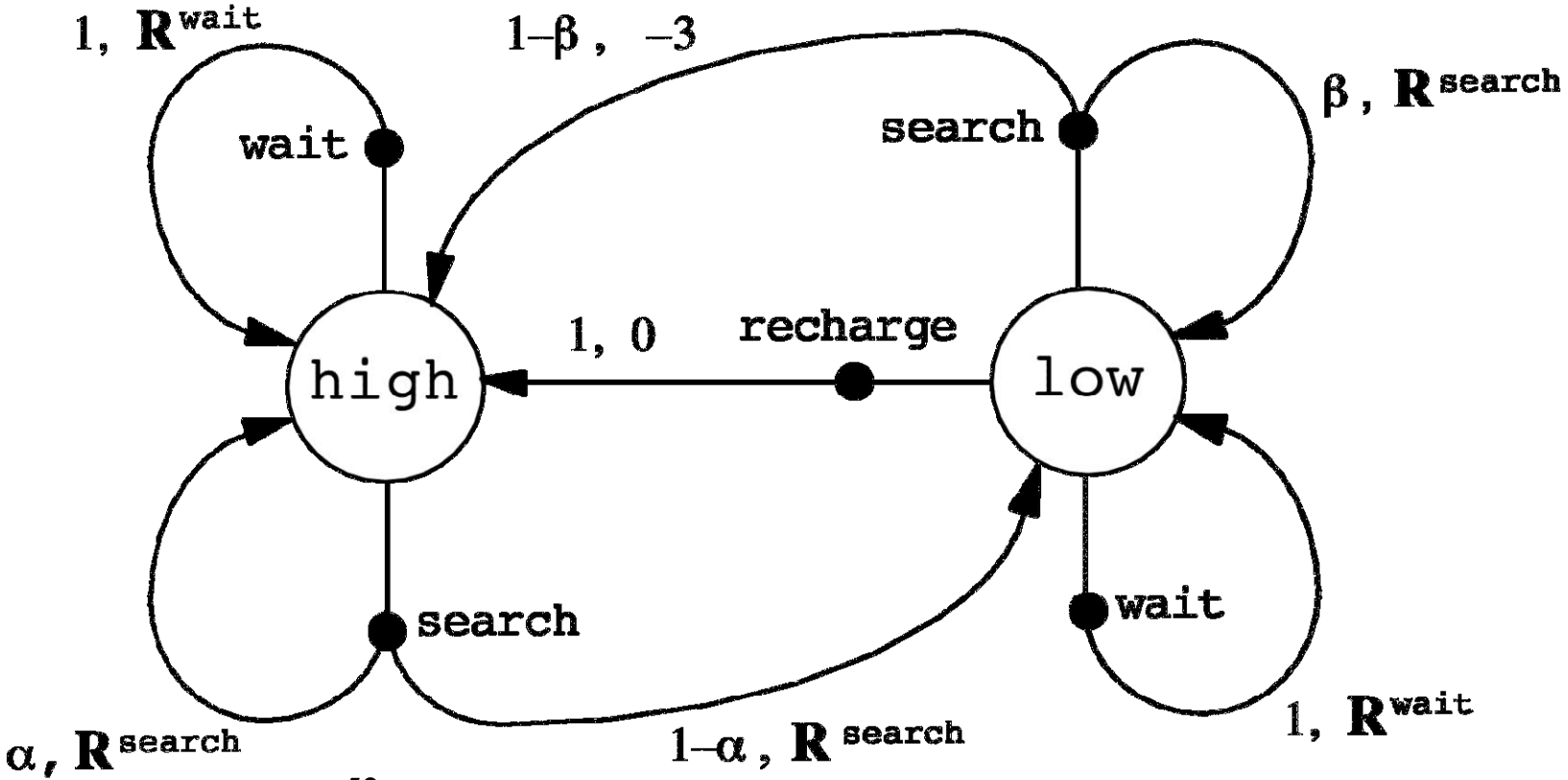
$A(\texttt{high}) = \{\texttt{search}, \texttt{wait}\}$

$A(\texttt{low}) = \{\texttt{search}, \texttt{wait}, \texttt{recharge}\}$

$\mathbf{R}^{\texttt{search}}$ = expected no. of cans while searching

$\mathbf{R}^{\texttt{wait}}$ = expected no. of cans while waiting

$\mathbf{R}^{\texttt{search}} > \mathbf{R}^{\texttt{wait}}$



Value Functions

- The **value of a state** is the expected return starting from that state; depends on the agent's policy:

State-value function for policy π:

$$V^{\pi}(s) = E_{\pi}\{R_t \mid s_t = s\} = E_{\pi}\left\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid s_t = s\right\}$$

- The **value of taking an action in a state under policy π** is the expected return starting from that state, taking that action, and thereafter following π:

Action-value function for policy π:

$$Q^{\pi}(s,a) = E_{\pi}\{R_t \mid s_t = s, a_t = a\} = E_{\pi}\left\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid s_t = s, a_t = a\right\}$$

Bellman Equation for policy π

The basic idea:

$$\begin{aligned} R_t &= r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \gamma^3 r_{t+4} \cdots \\ &= r_{t+1} + \gamma \left(r_{t+2} + \gamma r_{t+3} + \gamma^2 r_{t+4} \cdots \right) \\ &= r_{t+1} + \gamma R_{t+1} \end{aligned}$$

So:

$$\begin{aligned} V^{\pi}(s) &= E_{\pi}\left\{ R_t \,\middle|\, s_t = s \right\} \\ &= E_{\pi}\left\{ r_{t+1} + \gamma V^{\pi}(s_{t+1}) \,\middle|\, s_t = s \right\} \end{aligned}$$

Or, without the expectation operator:

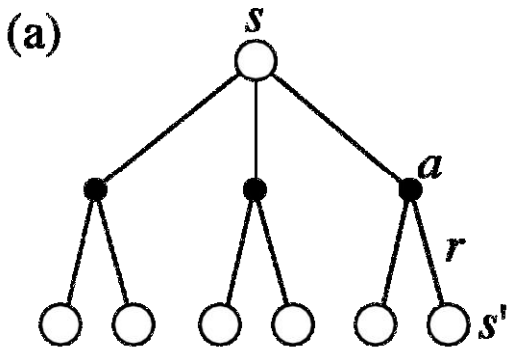
$$V^{\pi}(s) = \sum_a \pi(s,a) \sum_{s'} \mathbf{P}^a_{ss'} \left[\mathbf{R}^a_{ss'} + \gamma V^{\pi}(s') \right]$$

More on the Bellman Equation

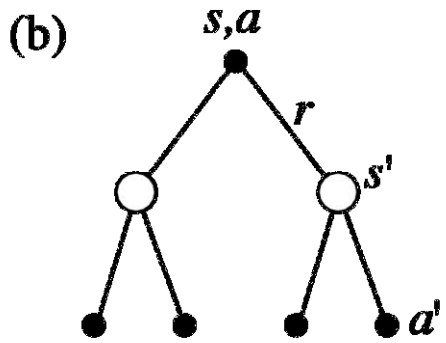
$$V^{\pi}(s) = \sum_{a} \pi(s,a) \sum_{s'} \mathbf{P}^{a}_{ss'} \left[\mathbf{R}^{a}_{ss'} + \gamma V^{\pi}(s') \right]$$

This is a set of equations (in fact, linear), one for each state.
The value function for π is its unique solution.

Backup diagrams:



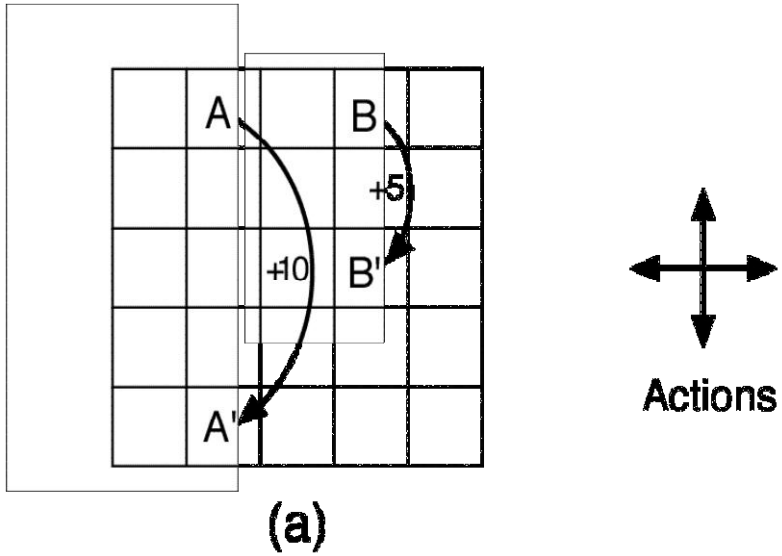
for V^{π}



for Q^{π}

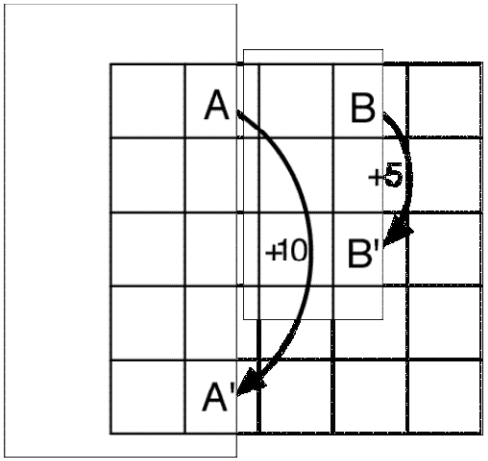
Gridworld

- Actions: `north`, `south`, `east`, `west`; deterministic.
- If it would take agent off the grid: no move but reward = –1
- Other actions produce reward = 0, except actions that move agent out of special states A and B, as shown.

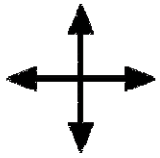


Gridworld

- Actions: `north`, `south`, `east`, `west`; deterministic.
- If would take agent off the grid: no move but reward = –1
- Other actions produce reward = 0, except actions that move agent out of special states A and B as shown.



(a)



3.3	8.8	4.4	5.3	1.5
1.5	3.0	2.3	1.9	0.5
0.1	0.7	0.7	0.4	-0.4
-1.0	-0.4	-0.4	-0.6	-1.2
-1.9	-1.3	-1.2	-1.4	-2.0

(b)

State-value function for equiprobable random policy; $\gamma = 0.9$

Optimal Value Functions

- For finite MDPs, policies can be **partially ordered**:

$$\pi \geq \pi' \quad \text{if and only if} \quad V^{\pi}(s) \geq V^{\pi'}(s) \ \text{ for all } s \in S$$

- There are always one or more policies that are better than or equal to all the others. These are the **optimal policies**. We denote them all π^*.
- Optimal policies share the same **optimal state-value function**:
- Optimal policies also share the same **optimal action-value function**:

$$V^*(s) = \max_{\pi} V^{\pi}(s) \quad \text{for all } s \in S$$

$$Q^*(s,a) = \max_{\pi} Q^{\pi}(s,a) \ \text{ for all } s \in S \text{ and } a \in A(s)$$

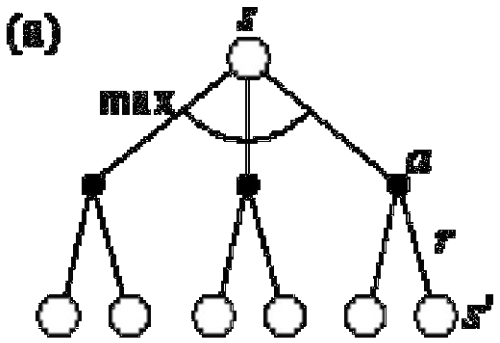
This is the expected return for taking action a in state s and thereafter following an optimal policy.

Bellman Optimality Equation for V^*

The value of a state under an optimal policy must equal the expected return for the best action from that state:

$$
\begin{aligned}
V^*(s) &= \max_{a \in A(s)} Q^{\pi^*}(s,a) \\
&= \max_{a \in A(s)} E\left\{ r_{t+1} + \gamma V^*(s_{t+1}) \,\middle|\, s_t = s, a_t = a \right\} \\
&= \max_{a \in A(s)} \sum_{s'} \mathbf{P}^a_{ss'} \left[\mathbf{R}^a_{ss'} + \gamma V^*(s') \right]
\end{aligned}
$$

The relevant backup diagram:



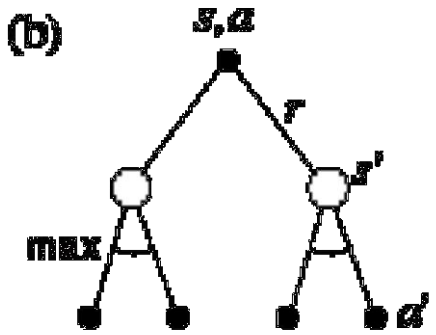
V^* is the unique solution of this system of nonlinear equations.

Bellman Optimality Equation for Q^*

$$Q^*(s,a) = E\left\{ r_{t+1} + \gamma \max_{a'} Q^*(s_{t+1}, a') \,\middle|\, s_t = s, a_t = a \right\}$$

$$= \sum_{s'} \mathbf{P}^a_{ss'} \left[\mathbf{R}^a_{ss'} + \gamma \max_{a'} Q^*(s', a') \right]$$

The relevant backup diagram:



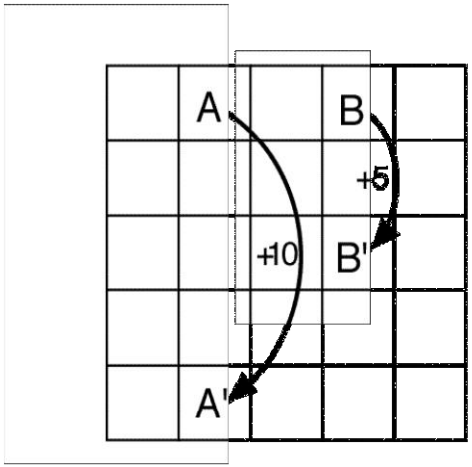
Q^* is the unique solution of this system of nonlinear equations.

Why Optimal State-Value Functions are Useful

Any policy that is greedy with respect to V^* is an optimal policy.

Therefore, given V^*, one-step-ahead search produces the long-term optimal actions.

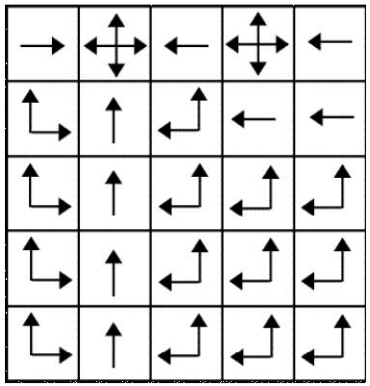
E.g., back to the gridworld:



a) gridworld

22.0	24.4	22.0	19.4	17.5
19.8	22.0	19.8	17.8	16.0
17.8	19.8	17.8	16.0	14.4
16.0	17.8	16.0	14.4	13.0
14.4	16.0	14.4	13.0	11.7

b) V^*



c) $\neq^* \pi^*$

What About Optimal Action-Value Functions?

Given Q^*, the agent does not even have to do a one-step-ahead search:

$$\pi^*(s) = \arg\max_{a \in A(s)} Q^*(s,a)$$

Solving the Bellman Optimality Equation

- Finding an optimal policy by solving the Bellman optimality equation requires the following:
  - accurate knowledge of environment dynamics;
  - enough space and time to do the computation;
  - the Markov property.
- How much space and time do we need?
  - polynomial in number of states (via dynamic programming methods),
  - BUT, number of states is often huge (e.g., backgammon has about 10^{20} states).
- We usually have to settle for approximations.
- Many RL methods can be understood as approximately solving the Bellman optimality equation.

Dynamic programming

- main idea
  - use value functions to structure the search for good policies
  - need a perfect model of the environment

- two main components
  - policy evaluation: compute V^π from π
  - policy improvement: improve π based on V^π



  - start with an arbitrary policy
  - repeat evaluation/improvement until convergence

Policy evaluation/improvement

- policy evaluation: π -> V^{π}
  - Bellman eqn's define a system of n eqn's
  - could solve, but will use iterative version

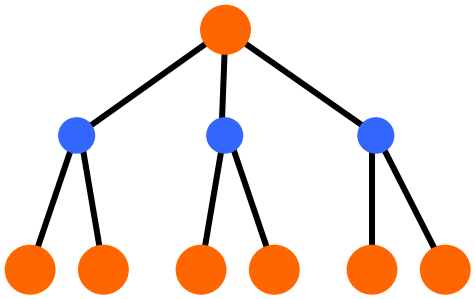
$$V_{k+1}(s) = \sum_a \pi(s,a) \sum_{s'} P^a_{ss'} [r^a_{ss'} + \gamma V_k(s')]$$

  - start with an arbitrary value function V_0, iterate until V_k converges

- policy improvement: V^{π} -> π'

$$\pi'(s) = \arg\max_a Q^{\pi}(s,a)$$

$$= \arg\max_a \sum_{s'} P^a_{ss'} \left[r^a_{ss'} + \gamma V^{\pi}(s') \right]$$



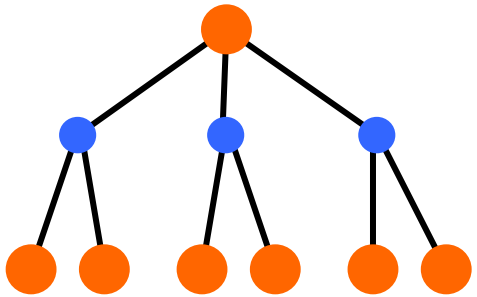
  - π' either strictly better than π, or π' is optimal (if $\pi = \pi'$)

Policy/Value iteration

- Policy iteration

$$\pi_0 \rightarrow^E V^{\pi_0} \rightarrow^I \pi_1 \rightarrow^E V^{\pi_1} \rightarrow^I \ldots \rightarrow^I \pi^* \rightarrow^E V^*$$

  - two nested iterations; too slow
  - don't need to converge to V^{π_k}
    - just move towards it



- Value iteration

$$V_{k+1}(s) = \max_a \sum_{s'} P^a_{ss'} \left[r^a_{ss'} + \gamma V_k(s') \right]$$

  - use Bellman optimality equation as an update
  - converges to V*

Using dynamic programming

- need complete model of the environment and rewards
  - robot in a room
    - state space, action space, transition model
- can we use DP to solve
  - robot in a room?
  - backgammon?
  - helicopter?
- DP bootstraps
  - updates estimates on the basis of other estimates

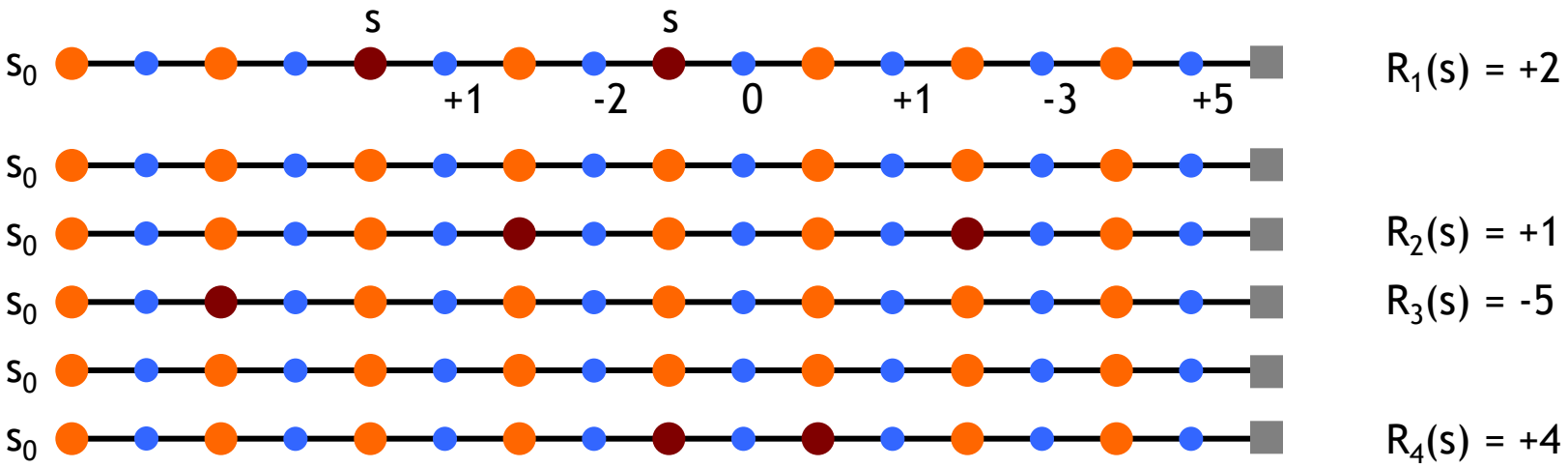
Monte Carlo methods

- don’t need full knowledge of environment
  - just experience, or
  - simulated experience
- averaging sample returns
  - defined only for episodic tasks
- but similar to DP
  - policy evaluation, policy improvement

Monte Carlo policy evaluation

- want to estimate $V^{\pi}(s)$
  = expected return starting from s and following π
  - estimate as average of observed returns in state s

- first-visit MC
  - average returns following the first visit to state s



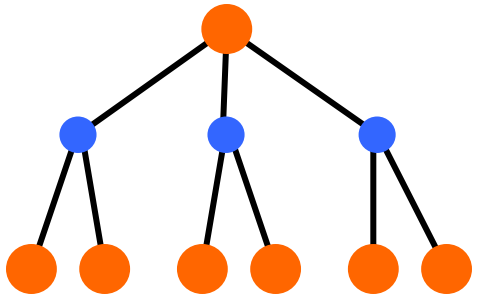
$$V^{\pi}(s) \approx (2 + 1 - 5 + 4)/4 = 0.5$$

Monte Carlo control

- V^π not enough for policy improvement
  - need exact model of environment

$$\pi'(s) = \arg\max_a Q^\pi(s,a)$$

- estimate $Q^\pi(s,a)$



- MC control

$$\pi_0 \rightarrow^E Q^{\pi_0} \rightarrow^I \pi_1 \rightarrow^E Q^{\pi_1} \rightarrow^I \ldots \rightarrow^I \pi^* \rightarrow^E Q^*$$

  - update after each episode

- non-stationary environment

$$V(s) \leftarrow V(s) + \alpha\,[R - V(s)]$$

- a problem
  - greedy policy won't explore all actions

Maintaining exploration

- key ingredient of RL

- deterministic/greedy policy won't explore all actions
  - don't know anything about the environment at the beginning
  - need to try all actions to find the optimal one

- maintain exploration
  - use *soft* policies instead: $\pi(s,a)>0$ (for all s,a)

- ε-greedy policy
  - with probability 1-ε perform the optimal/greedy action
  - with probability ε perform a random action

  - will keep exploring the environment
  - slowly move it towards greedy policy: ε -> 0

Simulated experience

- 5-card draw poker
  - s_0: A♣, A♦, 6♠, A♥, 2♠
  - a_0: discard 6♠, 2♠
  - s_1: A♣, A♦, A♥, A♠, 9♠ + dealer takes 4 cards
  - return: +1 (probably)

- DP
  - list all states, actions, compute P(s,a,s')
    - P([A♣,A♦,6♠,A♥,2♠], [6♠,2♠], [A♠,9♠,4]) = 0.00192

- MC
  - all you need are sample episodes
  - let MC play against a random policy, or itself, or another algorithm

Summary of Monte Carlo

- don’t need model of environment
  - averaging of sample returns
  - only for episodic tasks
- learn from:
  - sample episodes
  - simulated experience
- can concentrate on “important” states
  - don’t need a full sweep
- no bootstrapping
  - less harmed by violation of Markov property
- need to maintain exploration
  - use soft policies

Value Iteration

void valueIteration() {
  initialize $V(s)$ arbitrarily
  do{
    foreach ($s \in S$) {
      foreach ($a \in A$) {
        $Q(s,a) = R(s,a)+\gamma \Sigma_{s' \in S} T(s,a,s')V(s')$
        $V(s) = \max_a Q(s,a)$
      }
    }
  } **while** (! policy good enough) ;
}

Algorithm updates values backwards (from final states)

Value iteration: convergence

- Theorem: If the maximum difference between two successive value functions is less than ε, then the value of the greedy policy, (the policy obtained by choosing, in every state, the action that maximizes the estimated discounted reward, using the current estimate of the value function) differs from the value function of the optimal policy by no more than $2\varepsilon\gamma/(1-\gamma)$ at any state.
- An effective stopping criterion for the algorithm
- Value iteration is very flexible. The assignments to V need not be done in strict order but instead can occur asynchronously in parallel, provided that the value of every state gets updated infinitely often on an infinite run.

Policy iteration

choose an arbitrary policy π'

loop

$\pi := \pi'$

compute the value function of policy π:

solve the linear equations

$$V_\pi(s) = R(s, \pi(s)) + \gamma \sum_{s' \in \mathcal{S}} T(s, \pi(s), s') V_\pi(s')$$

improve the policy at each state:

$$\pi'(s) := \arg\max_a \left(R(s, a) + \gamma \sum_{s' \in \mathcal{S}} T(s, a, s') V_\pi(s') \right)$$

until $\pi = \pi'$

The value function of a policy is just the expected infinite discounted reward that will be gained, at each state, by executing that policy. It can be determined by solving a set of linear equations. Once we know the value of each state under the current policy, we consider whether the value could be improved by changing the first action taken. If it can, we change the policy to take the new action whenever it is in that situation. This step is guaranteed to strictly improve the performance of the policy. When no improvements are possible, then the policy is guaranteed to be optimal.

Approximate solutions

- Learning with time differences (TD),
  a model is not needed, incremental, difficult for analysis
- Dynamic programming,
  mathematically well defined problems with exact and complete description of the environment
- Monte Carlo methods,
  model is not necessary, conceptually simple, not incremental, sampling complete trajectories in interaction with environment (or model of environment)
- Efficiency, convergence

TD(λ) learning

- Learning with time differences
- Previous states receive a portion of the difference to successors
- For λ=0
  $V(s_t) = V(s_t) + c(V(s_{t+1}) - V(s_t))$
- c is a parameter, slowly decreasing during learning assuring convergence
- For $\lambda > 0$, more than just immediate successors are taken into account (speed)

Temporal Difference Learning

- combines ideas from MC and DP
  - like MC: learn directly from experience (don't need a model)
  - like DP: bootstrap
  - works for continuous tasks, usually faster then MC

- constant-alpha MC:
  - have to wait until the end of episode to update

$$V(s_t) \leftarrow V(s_t) + \alpha \left[R_t - V(s_t)\right]$$



target

- simplest TD
  - update after every step, based on the successor

$$V(s_t) \leftarrow V(s_t) + \alpha \left[r_{t+1} + \gamma V(s_{t+1}) - V(s_t)\right]$$



MC vs. TD

- observed the following 8 episodes:

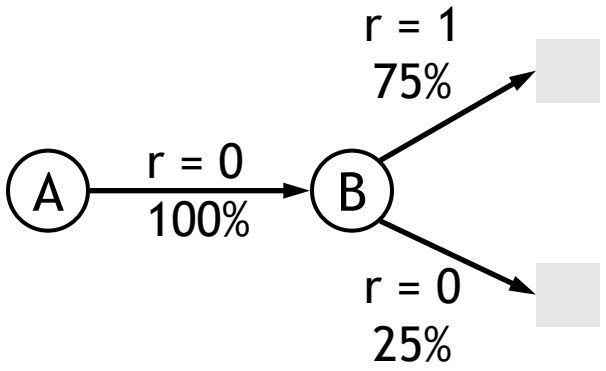
  A – 0, B – 0 B – 1 B – 1 B - 1

  B – 1 B – 1 B – 1 B – 0

- MC and TD agree on V(B) = 3/4

- MC: V(A) = 0
  - converges to values that minimize the error on training data

- TD: V(A) = 3/4
  - converges to ML estimate of the Markov process



Q-learning

- previous algorithms: on-policy algorithms
  - start with a random policy, iteratively improve
  - converge to optimal

- Q-learning: off-policy
  - use any policy to estimate Q

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t) \right]$$

  - Q directly approximates Q* (Bellman optimality eqn)
  - independent of the policy being followed
  - only requirement: keep updating each (s,a) pair

- Sarsa

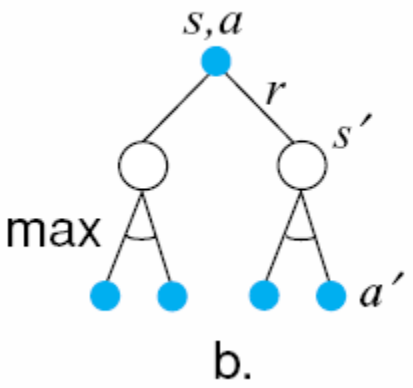
$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_t + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t) \right]$$

Q learning

- Watkins, 1989
- The most popular variant of time difference learning
- One step ahead

$Q(s_t,a_t) = (1-c)\, Q(s_t,a_t) + c(r_{t+1} + \gamma \max_a Q(s_{t+1},a) - Q(s_t,a_t))$

$0 <= c, \gamma <= 1$



Q-Learning: Definitions

- Current state: ***s***
- Current action: ***a***
- Transition function: $\delta(s, a) = s'$
- Reward function: $r(s, a) \in R$
- Policy $\pi(s) = a$
- $Q(s, a) \approx$ value of taking action ***a*** from state ***s***

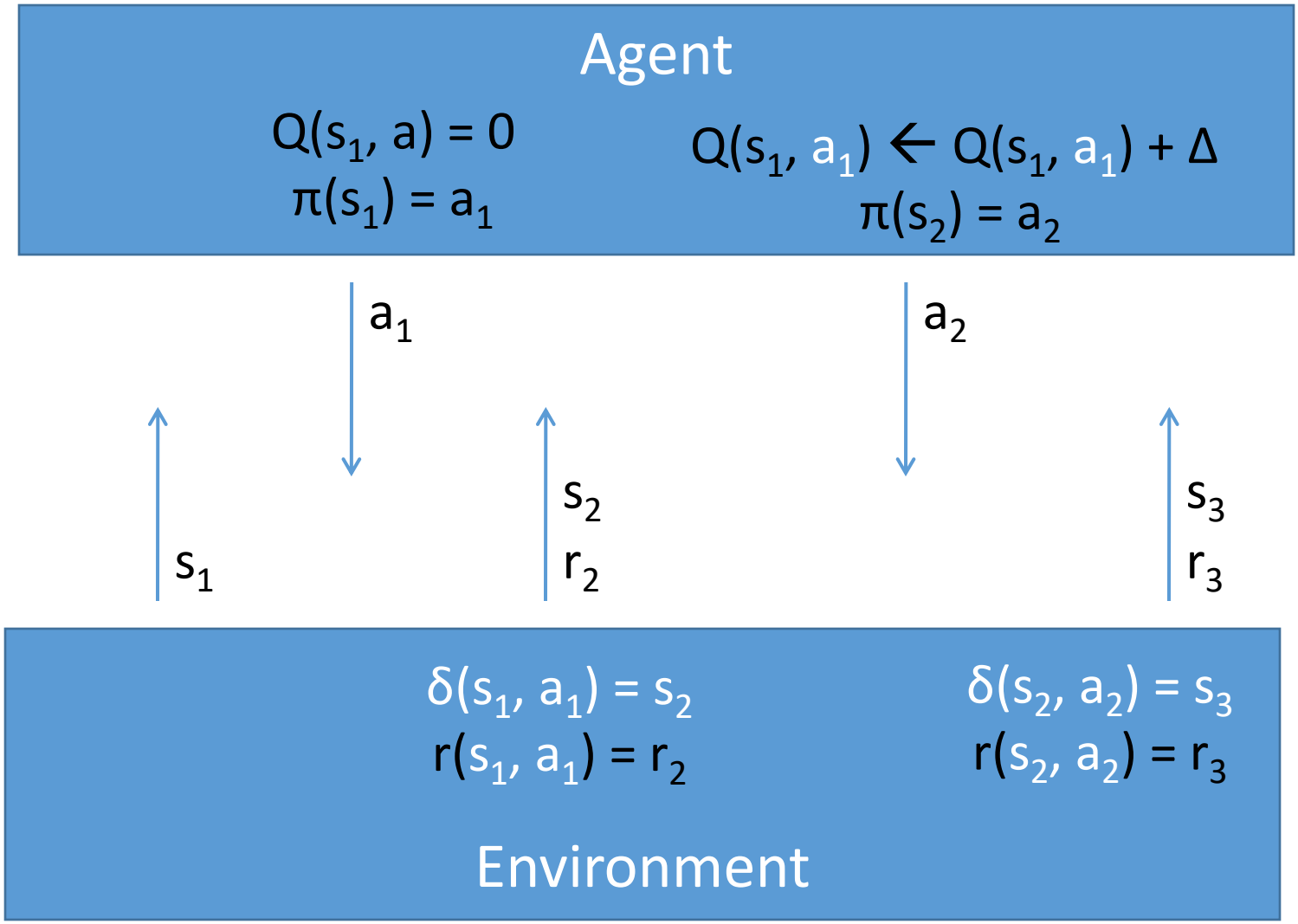
Markov property: this is independent of previous states given current state

In classification we'd have examples $(s, \pi(s))$ to learn from

The Q-function

- ***Q(s, a)*** estimates the *discounted cumulative reward*
  - Starting in state ***s***
  - Taking action ***a***
  - Following the current policy thereafter
- Suppose we have the optimal Q-function
  - What's the optimal policy in state ***s***?
  - The action $\boldsymbol{argmax_b\ Q(s, b)}$
- But we don't have the optimal Q-function at first
  - Let's act as if we do
  - And updates it after each step so it's closer to optimal
  - Eventually it will be optimal!

Q-Learning: The Procedure



Q-Learning: Updates

- The basic update equation

$$Q(s,a) \longleftarrow r(s,a) + \max_b Q(s',b)$$

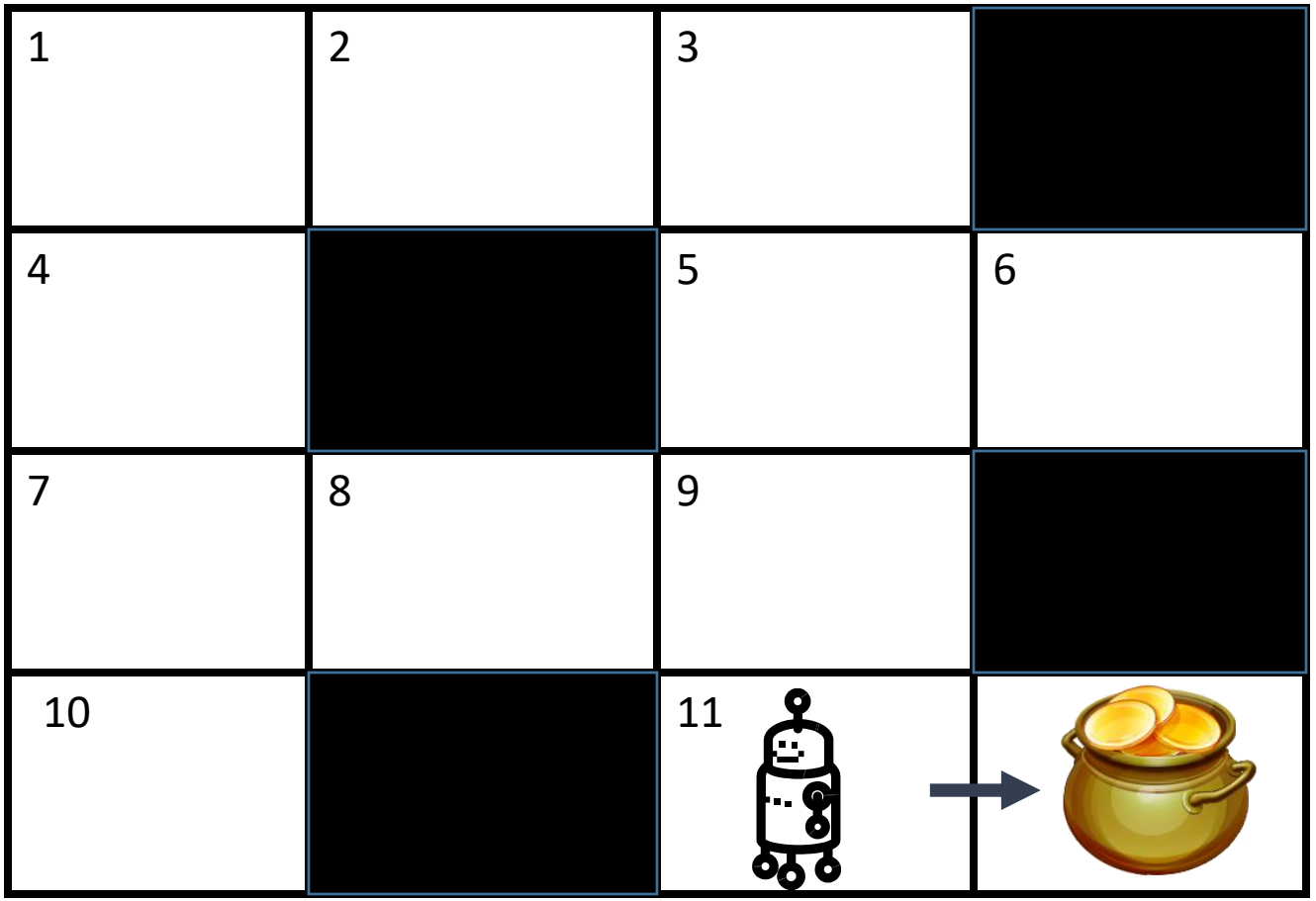
- With a discount factor to give later rewards less impact

$$Q(s,a) \longleftarrow r(s,a) + \gamma \max_b Q(s',b)$$

- With a learning rate for non-deterministic worlds

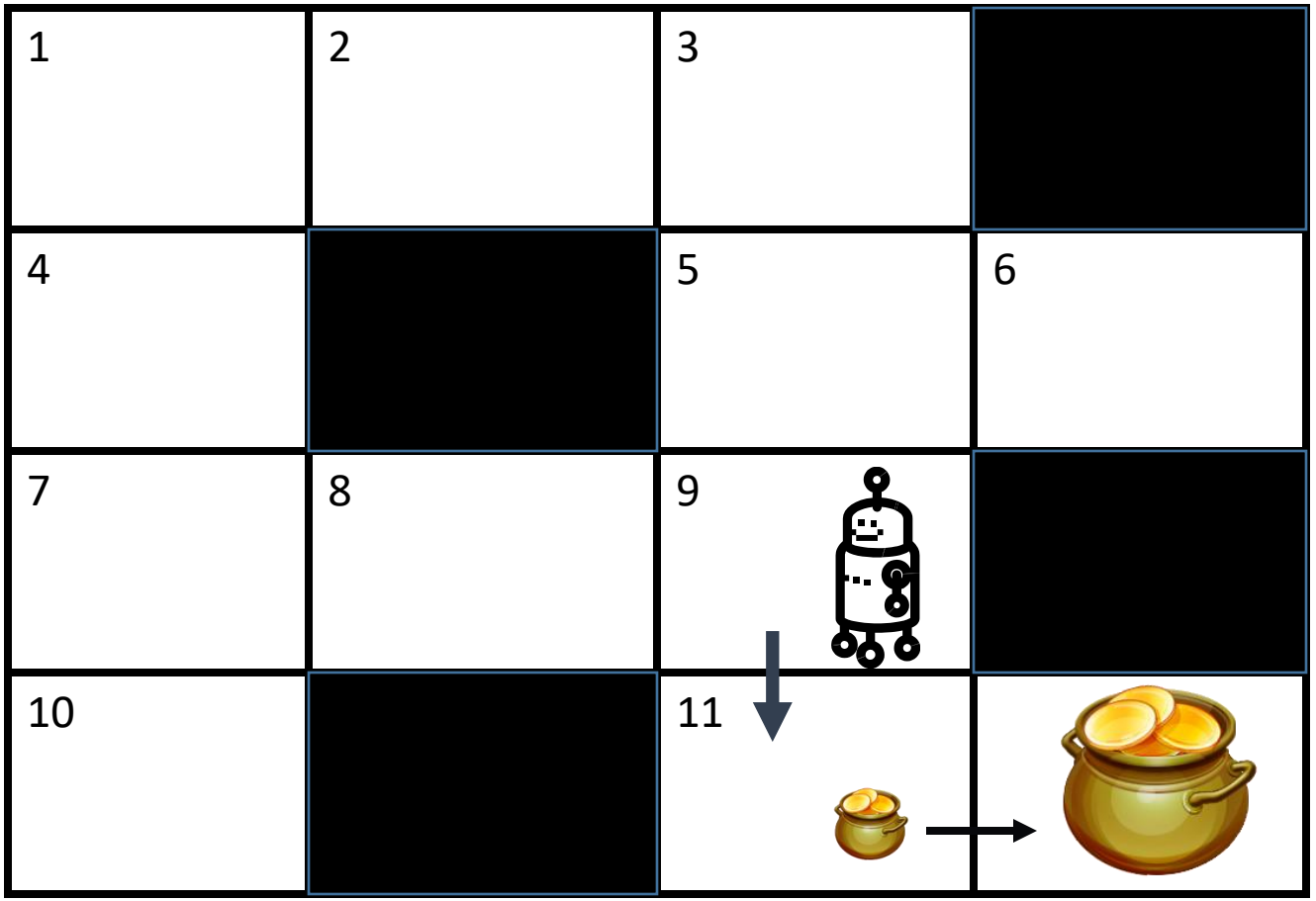
$$Q(s,a) \longleftarrow [1-\alpha]Q(s,a) + \alpha[r(s,a) + \gamma \max_b Q(s',b)]$$

Q-Learning: Update Example



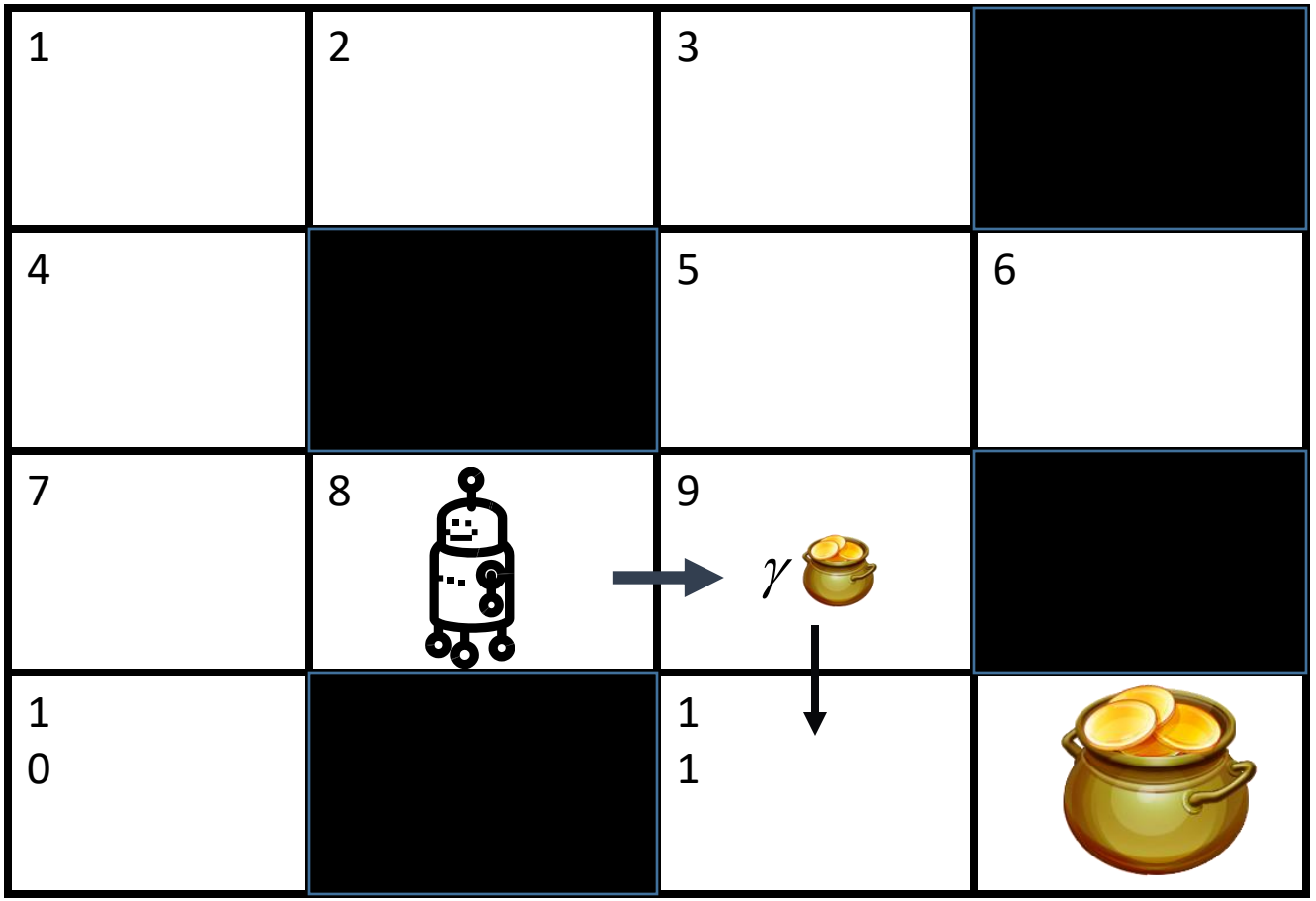
$$Q(s_{11}, a_{\rightarrow}) =$$ 

Q-Learning: Update Example



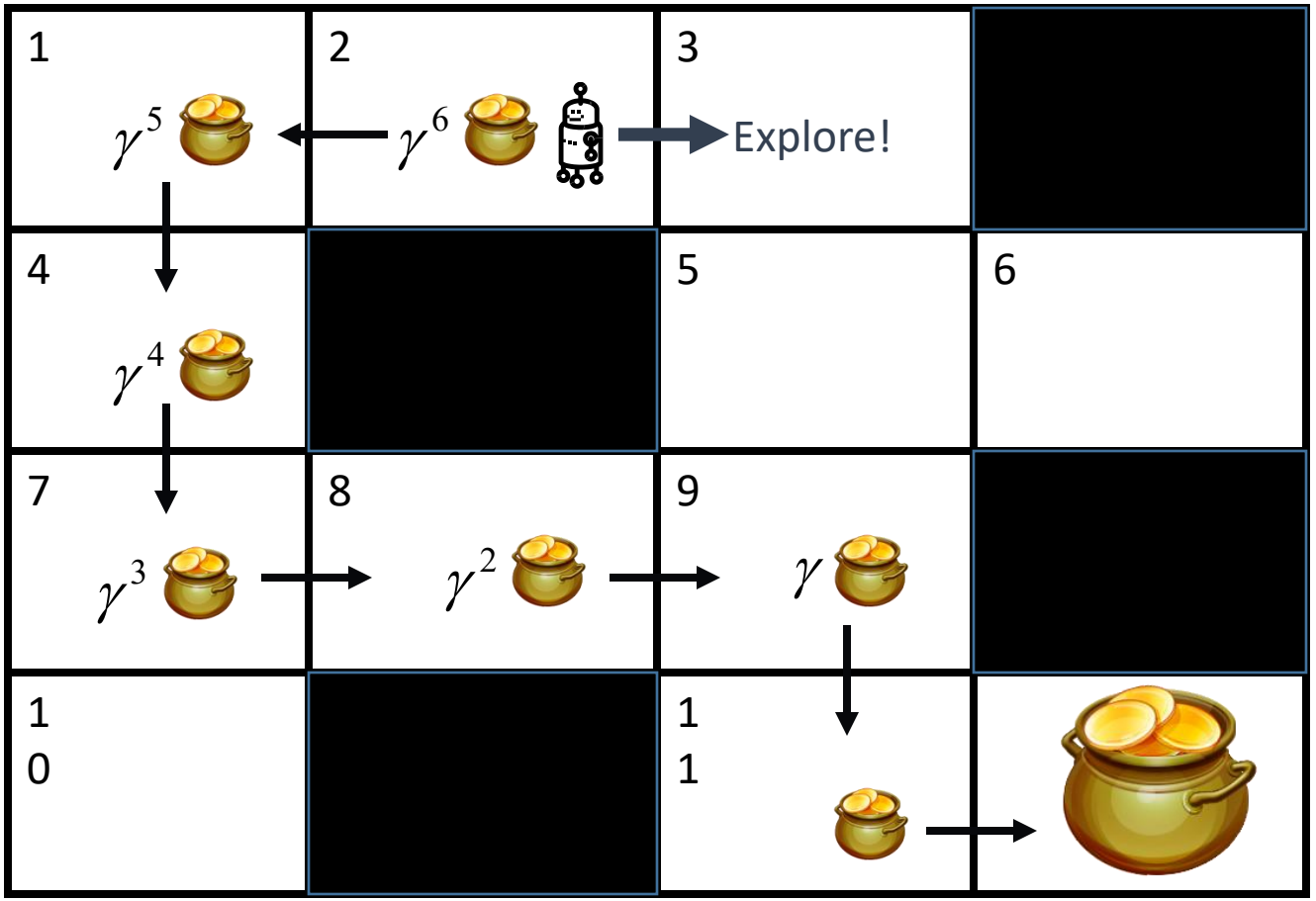
$$Q(s_9, a_{\downarrow}) = 0 + \gamma$$

Q-Learning: Update Example



$$Q(s_8, a_{\rightarrow}) = 0 + \gamma^2$$

The Need for Exploration



$$\arg\max Q(s_2, a) = \leftarrow$$

$$best = \rightarrow$$

Q learning

Initialize $Q(s, a)$ arbitrarily
Repeat (for each episode):
    Initialize s
    Repeat (for each step of episode):
        Choose a from s using policy derived from Q (e.g., ε-greedy)
        Take action a, observe r, s'
        $Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma \max_{a'} Q(s', a') - Q(s, a)]$
        $s \leftarrow s'$;
    until s is terminal

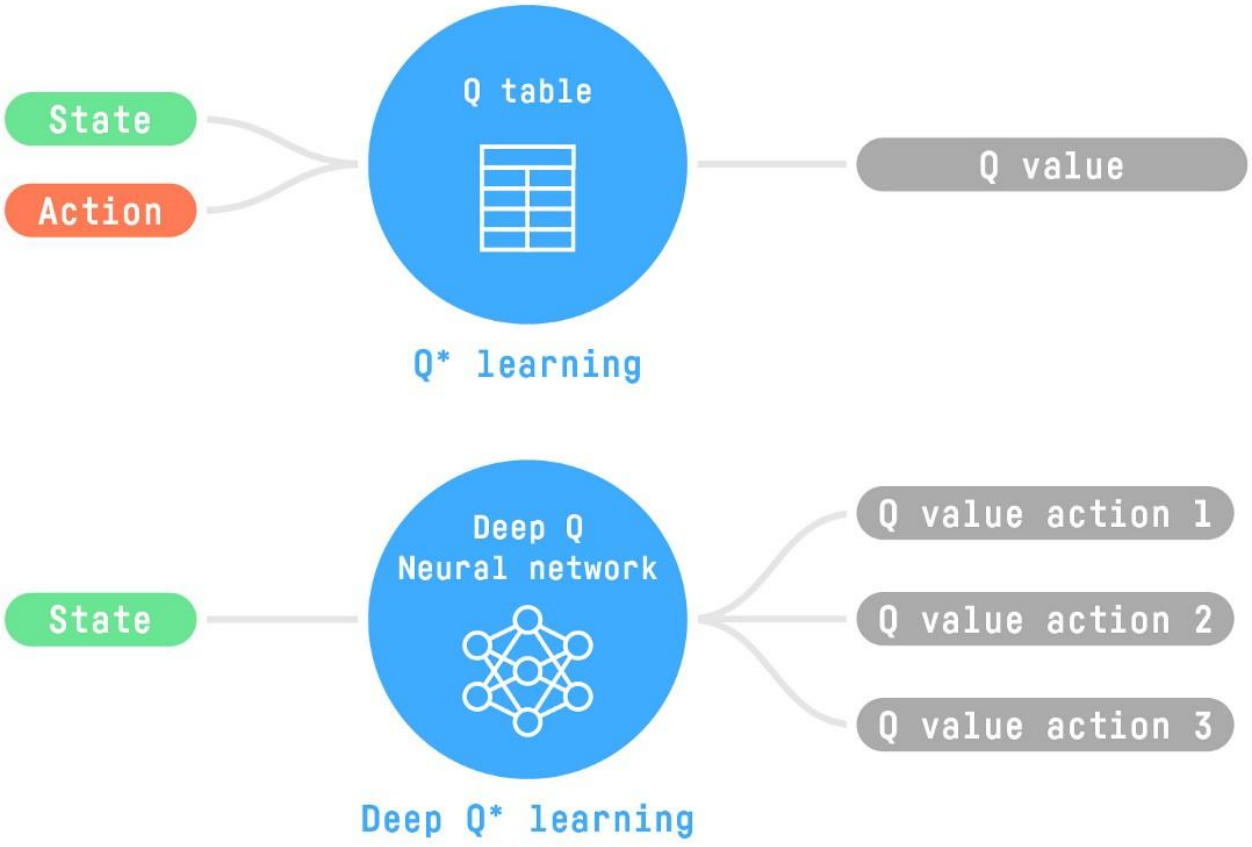
Explore/Exploit Tradeoff

- Can't always choose the action with highest Q-value
  - The Q-function is initially unreliable
  - Need to explore until it is optimal
- Most common method: ε-greedy
  - Take a random action in a small fraction of steps (**ε**)
  - Decay **ε** over time
- There is some work on optimizing exploration
  - Kearns & Singh, ML 1998
  - But people usually use this simple method

Q-Learning: Convergence

- Under certain conditions, Q-learning will converge to the correct Q-function
  - The environment model doesn't change
  - States and actions are finite
  - Rewards are bounded
  - Learning rate decays with visits to state-action pairs
  - Exploration method would guarantee infinite visits to every state-action pair over an infinite training period

Deep Q learning

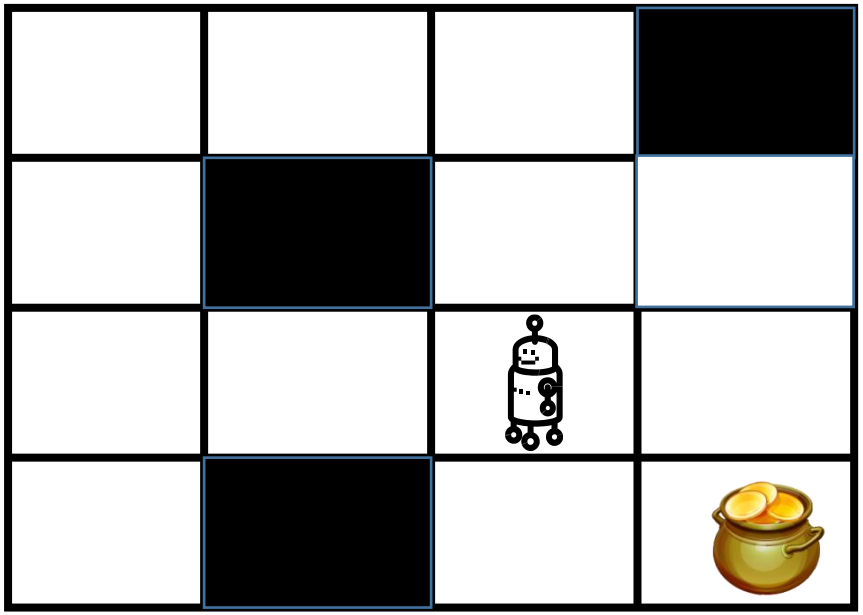


Extensions: SARSA

- SARSA: Take exploration into account in updates
  - Use the action actually chosen in updates

$$\cancel{Q(s,a) \longleftarrow r(s,a) + \gamma \max_b Q(s',b)}$$

$$Q(s,a) \longleftarrow r(s,a) + \gamma Q(s',a')$$

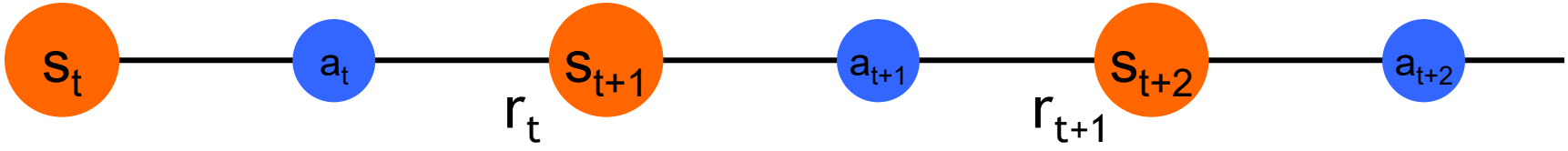


Regular: $\downarrow = \rightarrow$

SARSA: $\downarrow > \rightarrow$

Sarsa

- again, need Q(s,a), not just V(s)



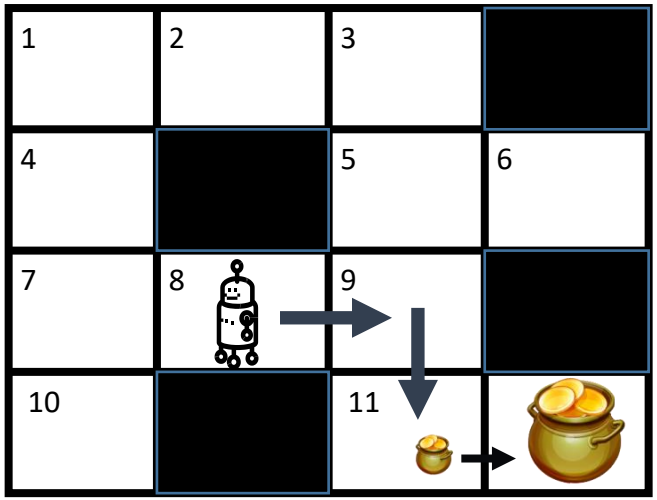
$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_t + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t) \right]$$

- control
  - start with a random policy
  - update Q and π after each step
  - again, need ε-soft policies

Extensions: Look-ahead

- Look-ahead: do updates over multiple states
  - Use some episodic memory to speed credit assignment

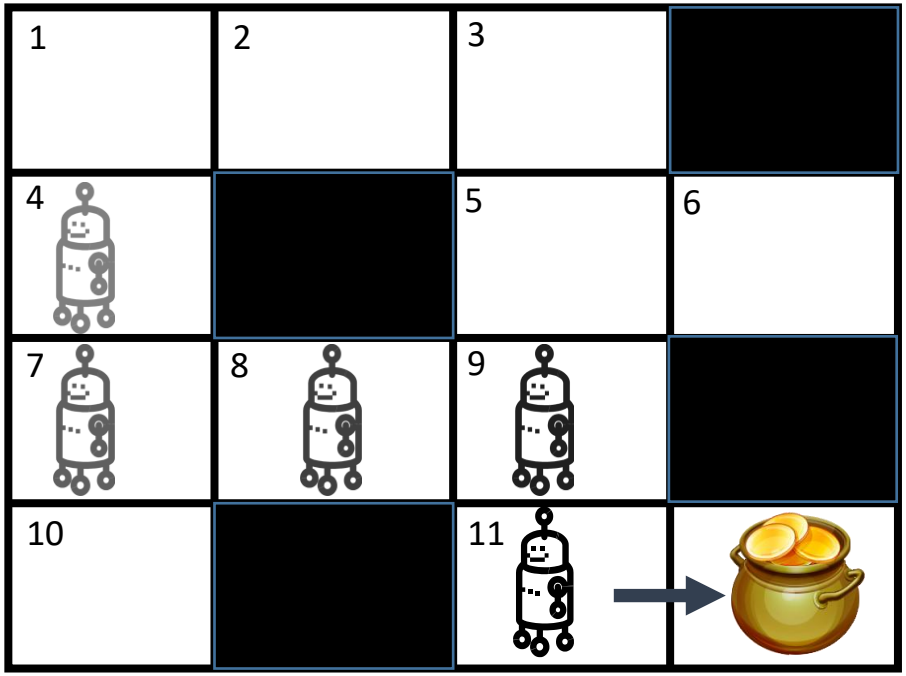
$$Q(s,a) \longleftarrow r(s,a) + \gamma r(s',a') + \gamma^2 Q(s'',a'')$$



- TD(**λ**): a weighted combination of look-ahead distances
  - The parameter **λ** controls the weighting

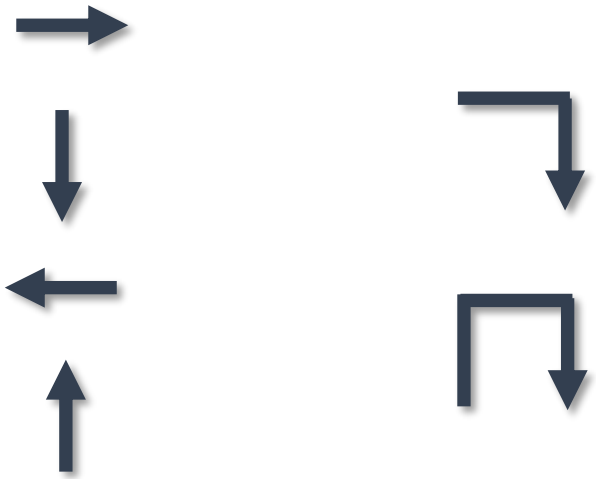
Extensions: Eligibility Traces

- Eligibility traces: Lookahead with less memory
  - Visiting a state leaves a trace that decays
  - Update multiple states at once
  - States get credit according to their trace

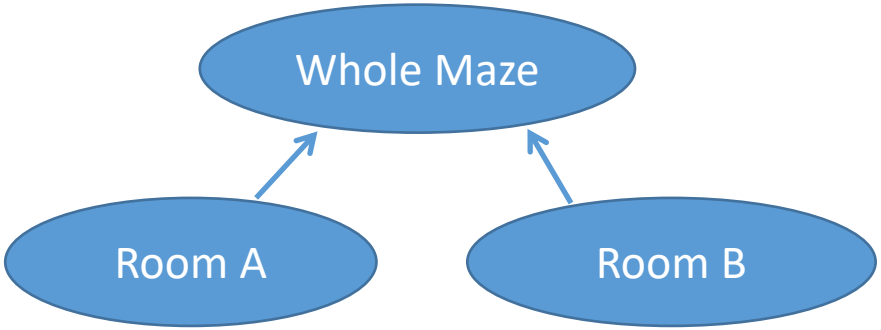


Extensions: Options and Hierarchies

- Options: Create higher-level actions

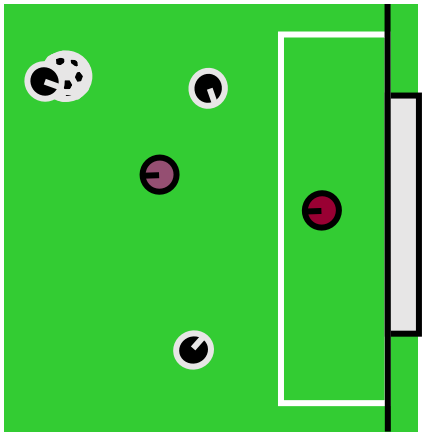


- Hierarchical RL: Design a tree of RL tasks



Extensions: Function Approximation

- Function approximation: allow complex environments
  - The Q-function table could be too big (or infinitely big!)
  - Describe a state by a feature vector
    $f = (f_1, f_2, \ldots, f_n)$
    - Then the Q-function can be any regression model
    - E.g. linear regression:
      $Q(s, a) = w_1 f_1 + w_2 f_2 + \ldots + w_n f_n$



  - Cost: convergence goes away in theory, though often not in practice
  - Benefit: generalization over similar states
  - Easiest if the approximator can be updated incrementally, like neural networks with gradient descent, but you can also do this in batches

Measuring learning performance

- Eventual convergence to optimality
  Many algorithms come with a provable guarantee of asymptotic convergence to optimal behavior. This is reassuring, but useless in practical terms.
- Speed of convergence to optimality
  Optimality is usually an asymptotic result, and so convergence speed is an ill-defined measure. More practical are
  - speed of convergence to near-optimality (how near?)
  - level of performance after a given time (what time?)
- Regret
  expected decrease in reward gained due to executing the learning algorithm instead of behaving optimally from the very beginning; these results are hard to obtain.

Challenges in Reinforcement Learning

- Feature/reward design can be very involved
  - Online learning (no time for tuning)
  - Continuous features (handled by *tiling*)
  - Delayed rewards (handled by *shaping*)
- Parameters can have large effects on learning speed
  - Tuning has just one effect: slowing it down
- Realistic environments can have partial observability
- Realistic environments can be non-stationary
- There may be multiple agents

Do Brains Perform RL?

- Should machine learning researchers care?
  - Planes don't fly the way birds do; should machines learn the way people do?
  - But why not look for inspiration?

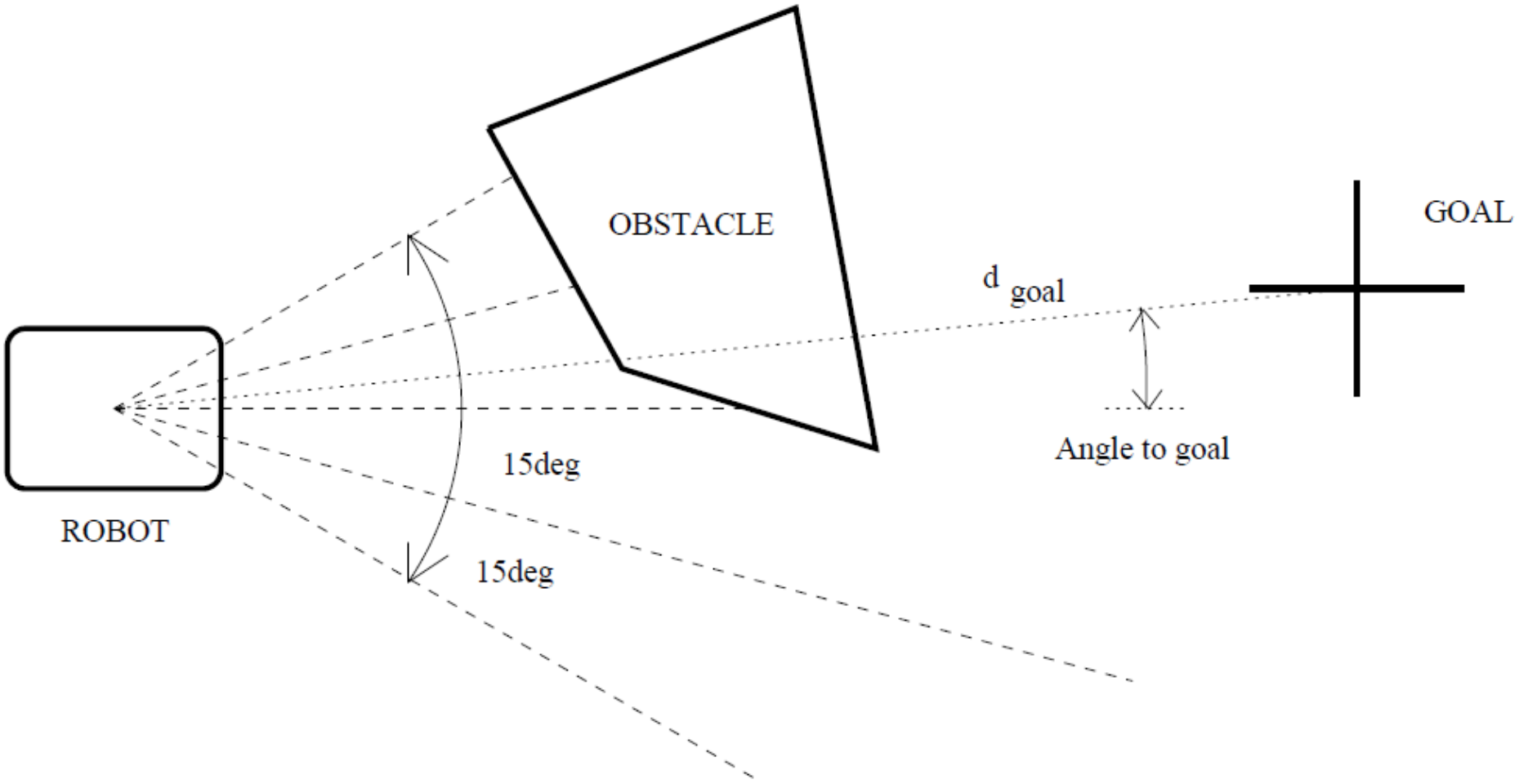
- Psychological research does show neuron activity associated with rewards
  - Really prediction error: actual – expected
  - Primarily in the striatum

What People Do Better

- Parallelism
  - Separate systems for positive/negative errors
  - Multiple algorithms running simultaneously
- Use of RL in combination with other systems
  - Planning: Reasoning about why things do or don't work
  - Advice: Someone to imitate or correct us
  - Transfer: Knowledge about similar tasks
- More impulsivity
  - Is this necessarily better?
- The goal for machine learning: Take inspiration from humans without being limited by their shortcomings

Some examples and details

An example: directing robot in 2d plane



- G.A.Rummery: Problem Solving with Reinforcement Learning, 1995

Robot in 2d: the settings

- sensors:
  - five distance measures to nearest obstacle in 15 degree forward arc
  - always knows distance and angle to the goal
- payoff after the end of the trial (reaching goal, collision with an obstacle or time out)
- start, goal and obstacles are randomly changed after every trial
- robot has to learn a generalized reactive policy; how?

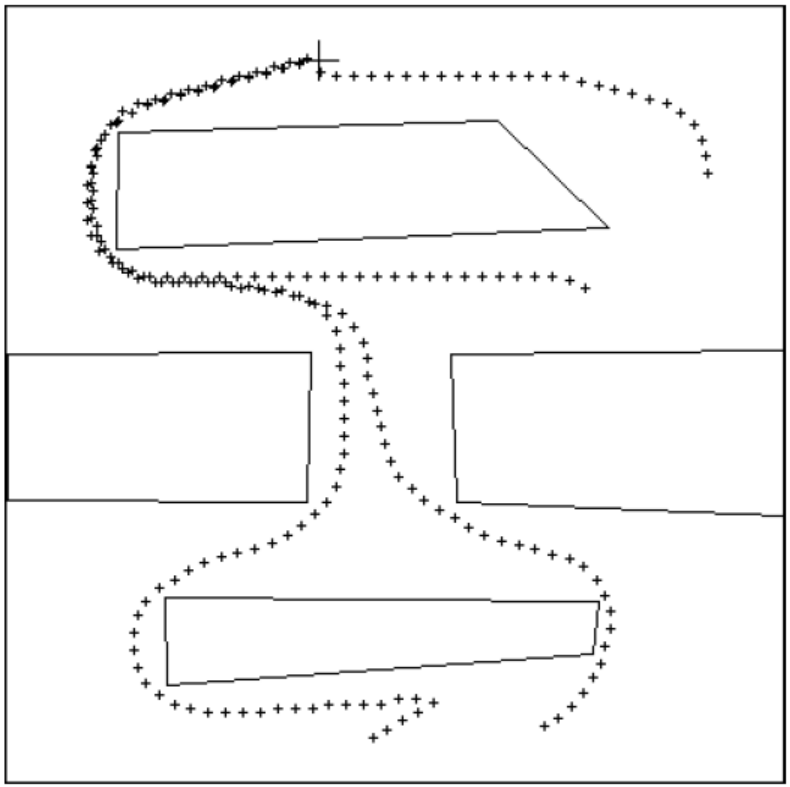
Robot in 2d: actions and rewards

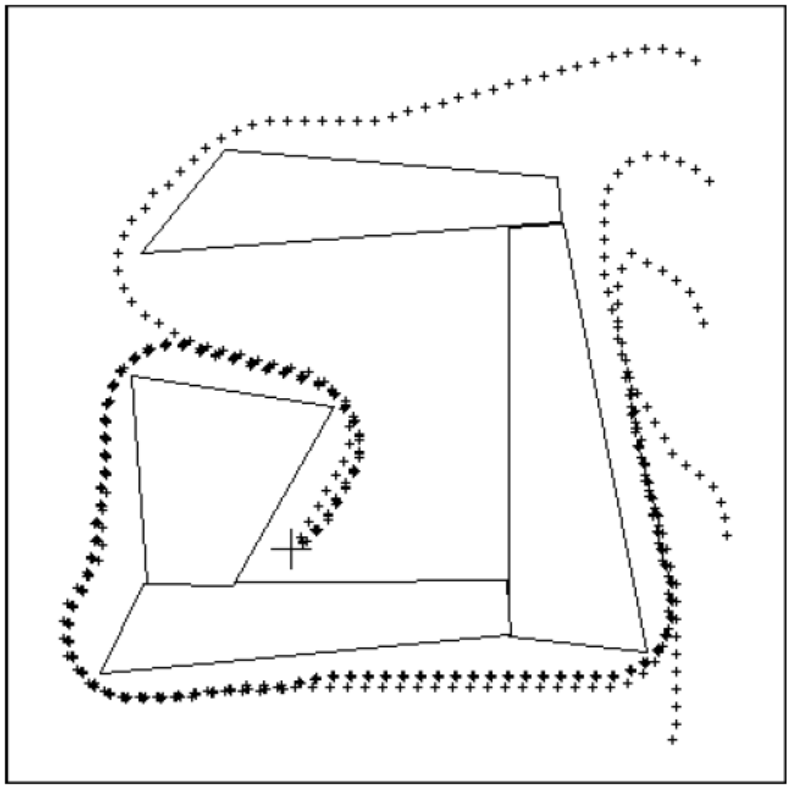
- 6 actions:
  - (turn left 15º, turn right 15º, stay in the same direction) x (move forward for a fixed distance d, do not move)
- rewards:
  - 0 in every step except the final
  - goal: if in a small fixed radius around the goal, +1
  - crash: based on a distance d from the goal e.g. $0.5 \exp(-\frac{2d_{goal}}{d_{max}})$, (note: maximum is 0.5)
  - time-out: as for crash + some small reward for not crashing, e.g., +0.3
- set γ = 0.99 to reward faster findings of the goal
- set probability of exploration/exploitation

Using NN for 2-d robot

- coarse coding the inputs (e.g., with several input sigmoids for each sensor)
- backpropagation with momentum term or eligibility traces
- batch and on-line training

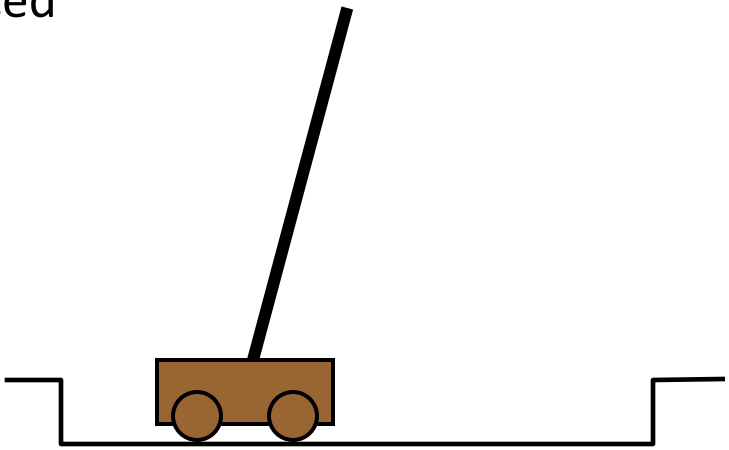
Some trajectories of trained robots





State representation

- pole-balancing
  - move car left/right to keep the pole balanced
- state representation
  - position and velocity of car
  - angle and angular velocity of pole
- what about *Markov property*?
  - would need more info
  - noise in sensors, temperature, bending of pole
- solution
  - coarse discretization of 4 state variables
    - left, center, right
  - totally non-Markov, but still works



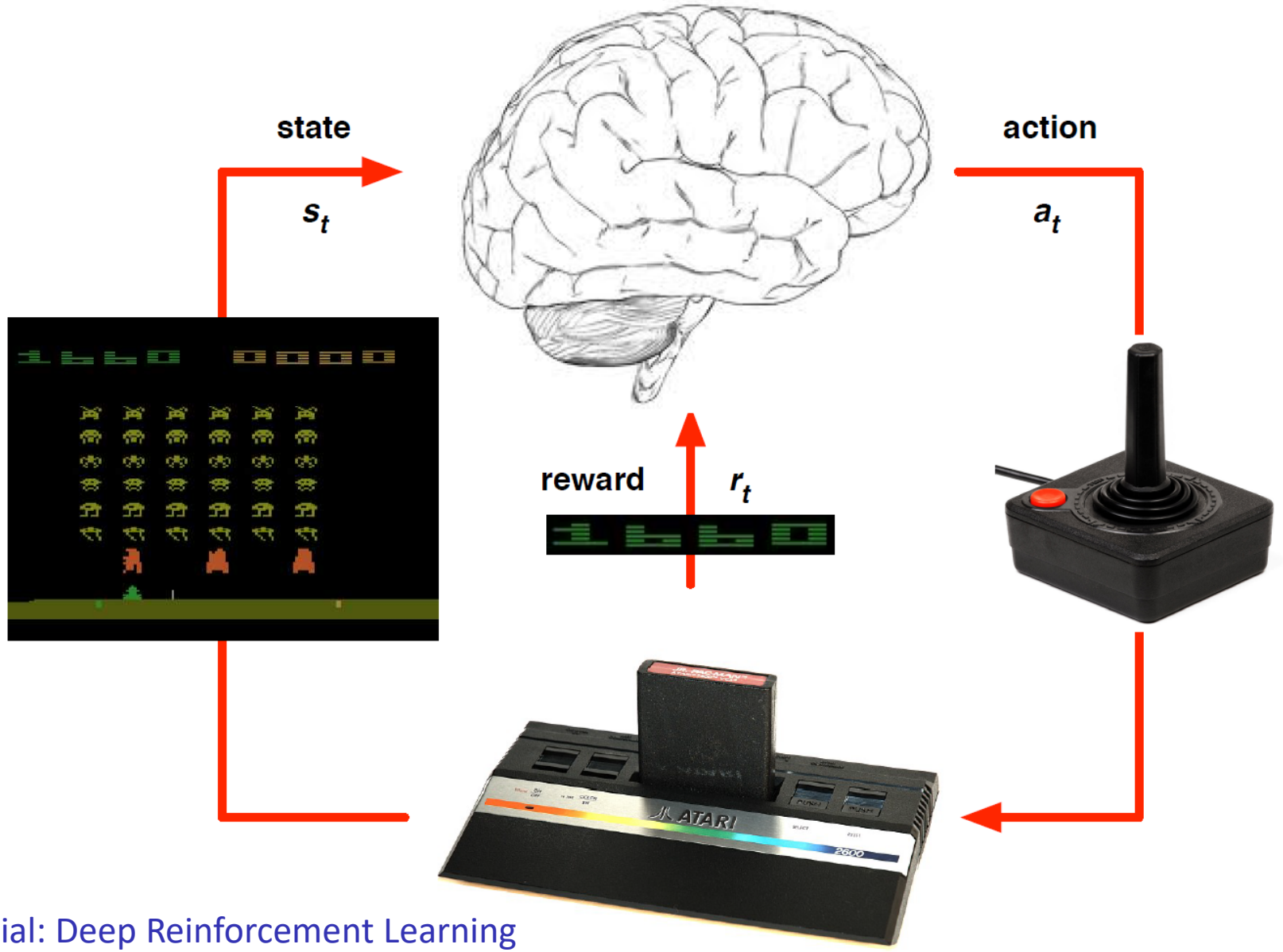
Designing rewards

- robot in a maze
  - episodic task, not discounted, +1 when out, 0 for each step
- chess
  - GOOD: +1 for winning, -1 losing
  - BAD: +0.25 for taking opponent's pieces
    - high reward even when lose
- rewards
  - rewards indicate what we want to accomplish
  - NOT how we want to accomplish it
- shaping



  - positive reward often very "far away"
  - rewards for achieving subgoals (domain knowledge)
  - also: adjust initial policy or initial value function

Reinforcement Learning in Atari

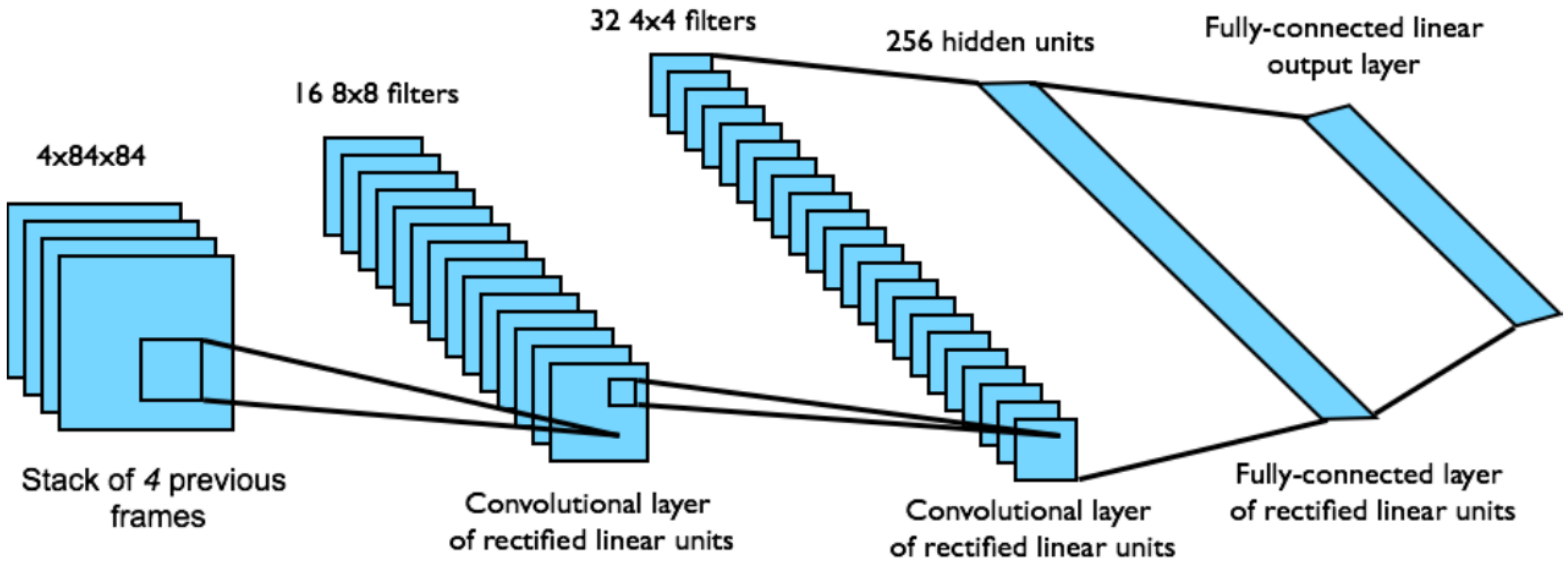


Tutorial: Deep Reinforcement Learning

David Silver, Google DeepMind

DQN in Atari

- End-to-end learning of values $Q(s, a)$ from pixels s
- Input state s is stack of raw pixels from last 4 frames
- Output is $Q(s, a)$ for 18 joystick/button positions
- Reward is change in score for that step



Network architecture and hyperparameters fixed across all games

[Mnih et al.]

RL with human feedback (RLHF)

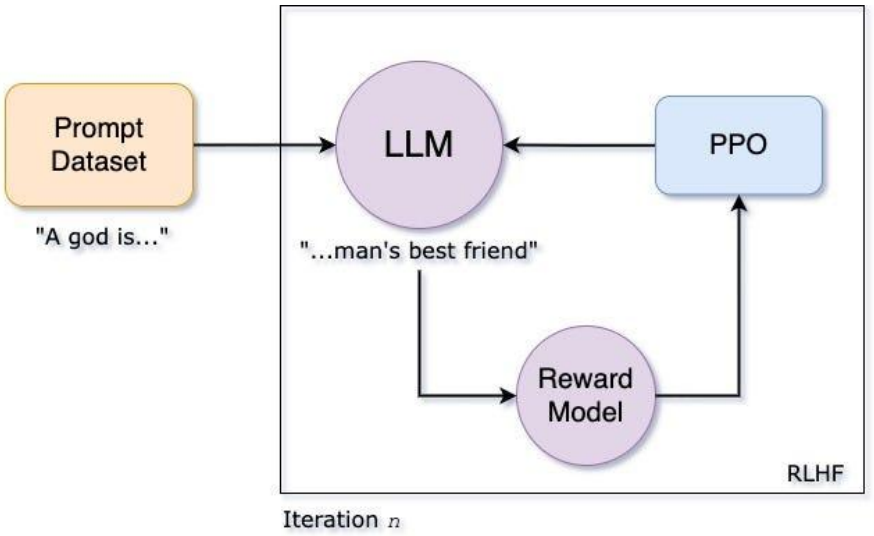
- RLHF for LLM preference alignment
- **RLHF** is the classic pipeline used to **align LLMs** with human preferences such as helpfulness, harmlessness, and honesty. It combines **supervised learning**, **human preference modeling**, and **RL**
- RLHF teaches an LLM what humans prefer by first learning a reward model from human judgments, then optimizing the LLM to maximize that reward while staying close to its original behavior.
- RLHF does **not** align truth or correctness directly. It aligns the model to **human preferences**, such as helpfulness, politeness, refusal behavior, non-toxicity, style and tone
- This can lead to known issues, such as reward hacking, over-verbosity, preference bias, and misalignment with ground truth

Three stages of RLHF

1. Supervised Fine-Tuning of the initial LLM to produce reasonable answers

2. Reward model on e.g., pairs of human responses (x,y+,y−) i.e., which score is better

3. RL to adjust LLM weights, typically using PPO (proximal policy optimization)



Proximal policy optimization (PPO)

In LLM alignment, PPO objective (simplified) is set as:

$$\max_{\theta} \quad \mathbb{E}_{y\sim\pi_\theta}[r_\phi(x,y) - \beta\,\mathrm{KL}\left(\pi_\theta \parallel \pi_{ref}\right)]$$

- Where:
- π_{ref} =frozen SFT model
- KL penalty prevents the model from drifting too far
- PPO ensures stable, clipped updates

Direct Preference Optimization (DPO)

- A simple and stable algorithm for aligning large language models (LLMs) using preference data — without reinforcement learning.
- DPO removes the reward model and RL
- Instead of learning a reward function $r(x, y)$, DPO directly optimizes the policy so that:

$$\pi_\theta(y^+ \mid x) \; > \; \pi_\theta(y^- \mid x)$$

- but relative to a reference model π_{ref} i.e. usually the pretrained SFT model).

Training Procedure for DPO

- 1. Start from pretrained/SFT model
- 2. Clone reference model (frozen)
- 3. Compute log-probs for y+ and y-
- 4. Apply DPO loss
- 5. Standard backpropagation

- No rollouts, no rewards, no RL.

Summary of RL

- Reinforcement learning
  - use when need to make decisions in uncertain environment
  - actions have delayed effect

- solution methods
  - dynamic programming
    - need complete model

  - Monte Carlo
  - time difference learning (Sarsa, Q-learning)

- simple algorithms
- most work
  - designing features, state representation, rewards