GenAI for Software Engineering 2025

Introduction

Jorge Melegati

About me

- Bachelor in Computer Engineering (2010) @ITA
- Master in Computer Science (2017) @IME – USP
- PhD in Computer Science (2021) @Free University of Bozen-Bolzano

- >8 years experience in software development in industry

About the course

- When?
  - Lectures: Tue/Wed 10:00-12:00 & 14:00-16:00, Fri 10:00-12:00
  - Wednesday afternoons will be reserved to project work
  - But on 19/02 afternoon will have an hour talk from a Guest Speaker

- We will use edisciplinas:
  - https://edisciplinas.usp.br/course/view.php?id=126019
  - Please, register to it if you haven't done so yet

About the course

- Goals:
  - Give an overview of **how the current models work**
  - Presents their **advantages and disadvantages**
  - Describe **how they could be used for SE activities**

About the course – main topics

- Introduction to AI, ML and the transformers architecture
- Use and limitations of transformers
- Adapting models to downstream tasks
  - Prompt engineering, CoT, RAG, and ReAct
  - Examples of uses of GenAI for software engineering

Grading

- **Final project** to be presented at the last lecture (21/02)
  - Groups of **3 or 4** people (to be set in edisciplinas)
  - Task: to **employ GenAI for a SE activity** and discuss strengths and limitations
    - It could be the replication of an existent study, the exploration of a idea from you, etc.

- **Poster session**
  - You should visit other posters and discuss the studies of the others
  - You have to choose a poster from another group and write a review of it

- **Final grade** = average(Final project, Review)

About you

Let’s enter edisciplinas and answer the poll!

Introduction to AI

First of all, what is artificial intelligence?

The use of technologies to build machines and computers that have the ability to **mimic cognitive functions associated with human intelligence**, such as analyzing, reasoaning, and learning.

AI4SE vs SE4AI

- AI-based systems are software systems!

- How to better design and implement them?
  - Software Engineering problem
  - Independent of the use context

- SE4AI: Software Engineering processes for systems that employ Artificial Intelligence

AI4SE vs SE4AI

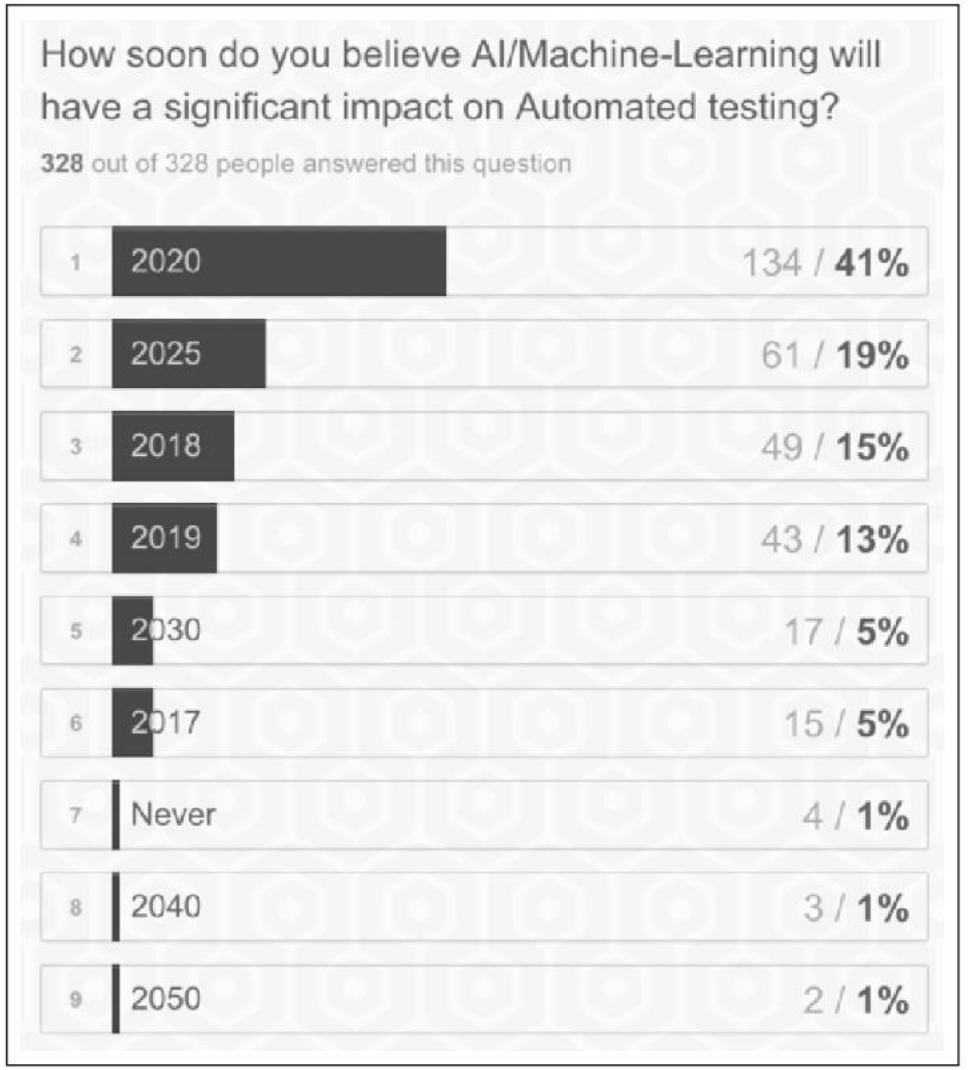
- Artificial Intelligence for Software Engineering (AI4SE)
  - The use of AI-based tools for several steps of SE processes
- Requirements Engineering
  - Example: to automatically identify the priority
- Coding
  - Example: generating coding according to the specification
- Testing
  - Example: automatically create test code

AI4SE

- Last papers in the International Conference on Software Engineering
  - https://conf.researchr.org/track/icse-2024/icse-2024-research-track

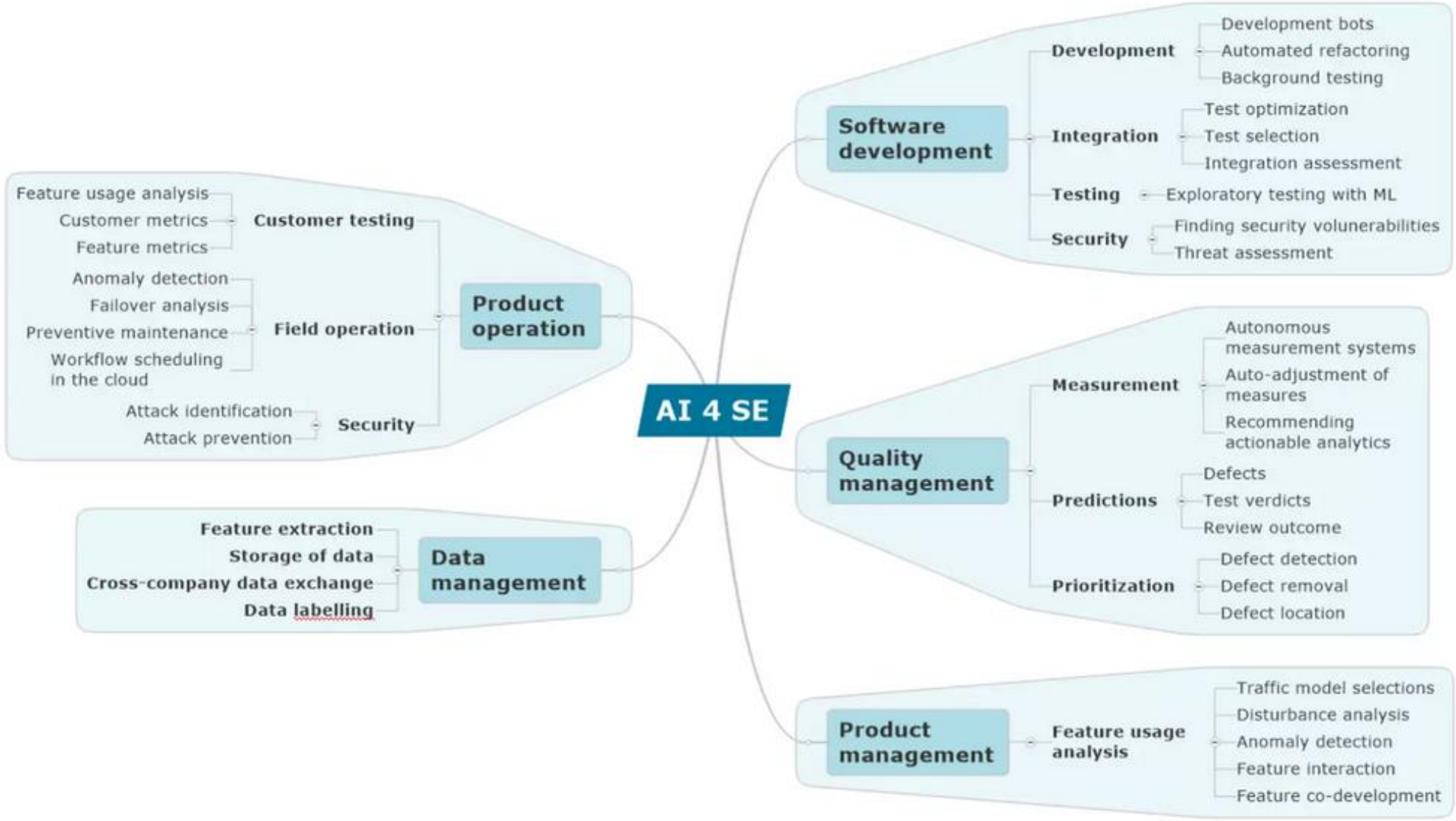
- Cutting-edge studies

And that’s how practitioners perceive the future….



Source: King, T. M., Arbon, J., Santiago, D., Adamo, D., Chin, W., & Shanmugam, R. (2019). AI for Testing Today and Tomorrow: Industry Perspectives. *2019 IEEE International Conference On Artificial Intelligence Testing (AITest)*, 81–88.

Several applications…



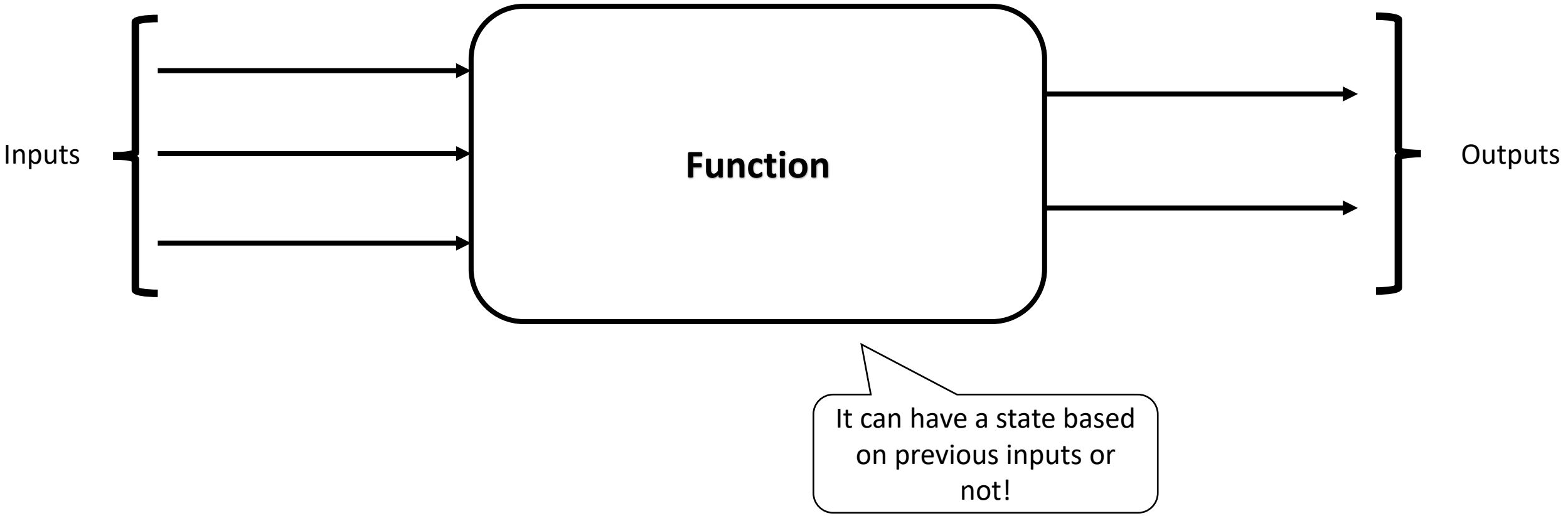
And machine learning?

A subset of AI that uses algorithms that extract knowledge from data (trained) to produce models that can perform complex tasks.

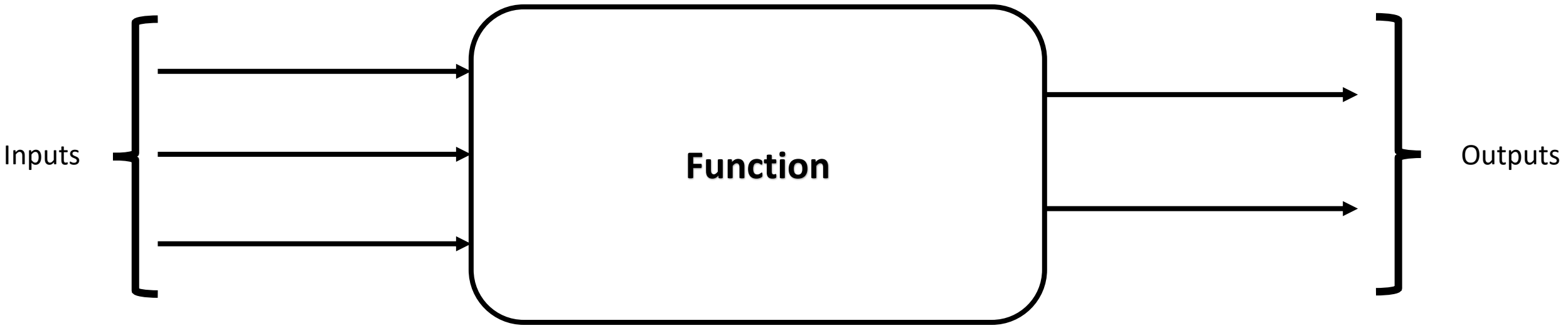
Other fields in AI:

- Natural language processing
- Computer Vision
- Robotics

An intuition on machine learning



An intuition on machine learning



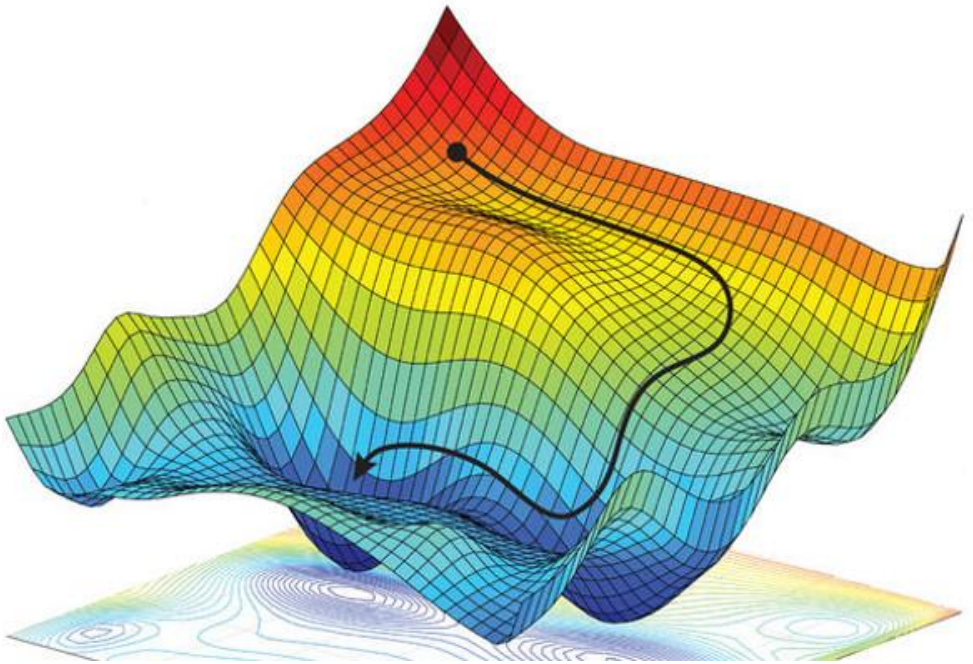
Question: how to define the parameters of the function?

It can have a state based on previous inputs or not!

Training process

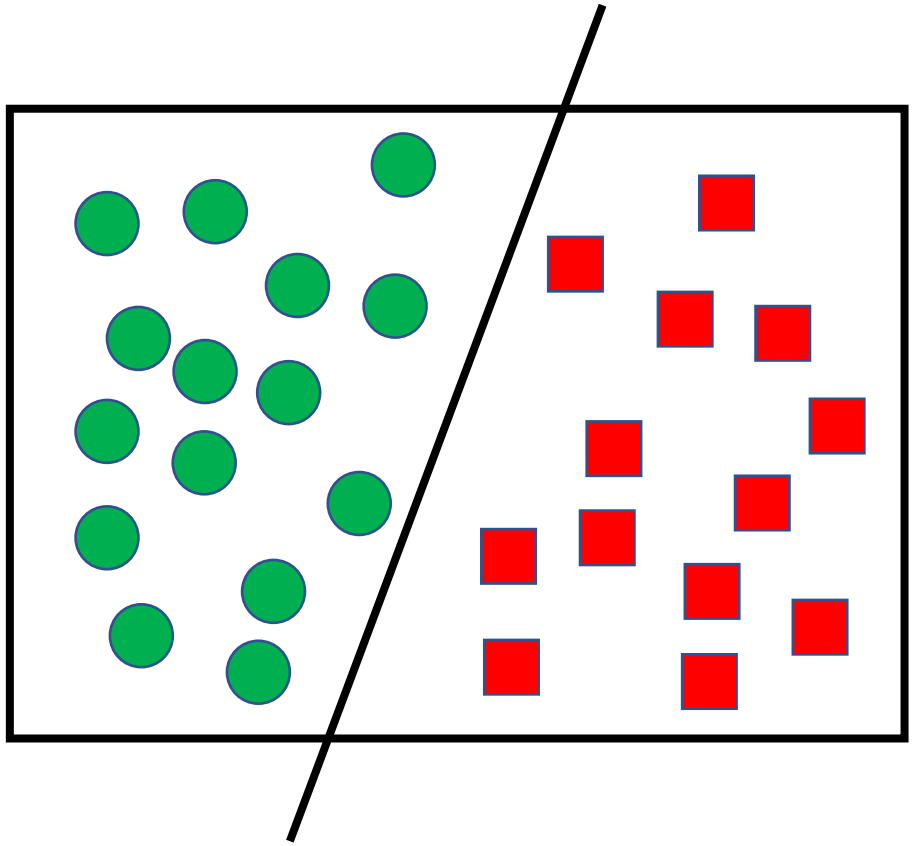
- Adding points to find the best function that fit the desired output

- Represented by a loss function
  - Goal: minimize the loss function



Source: Amini, A., Soleimany, A., Karaman, S., & Rus, D. (2018). Spatial uncertainty sampling for end-to-end control. *arXiv preprint arXiv:1805.04829*.

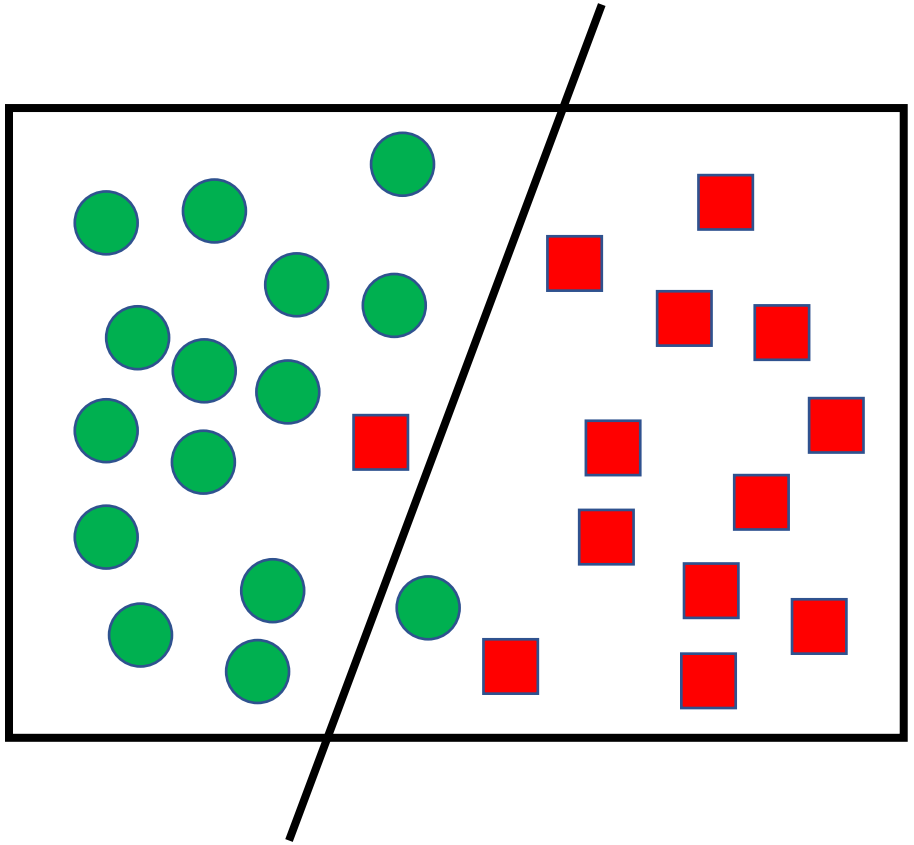
Simplified view in 2 dimensions



Task: separating these two geometrical forms distributed in this two-dimensional space.

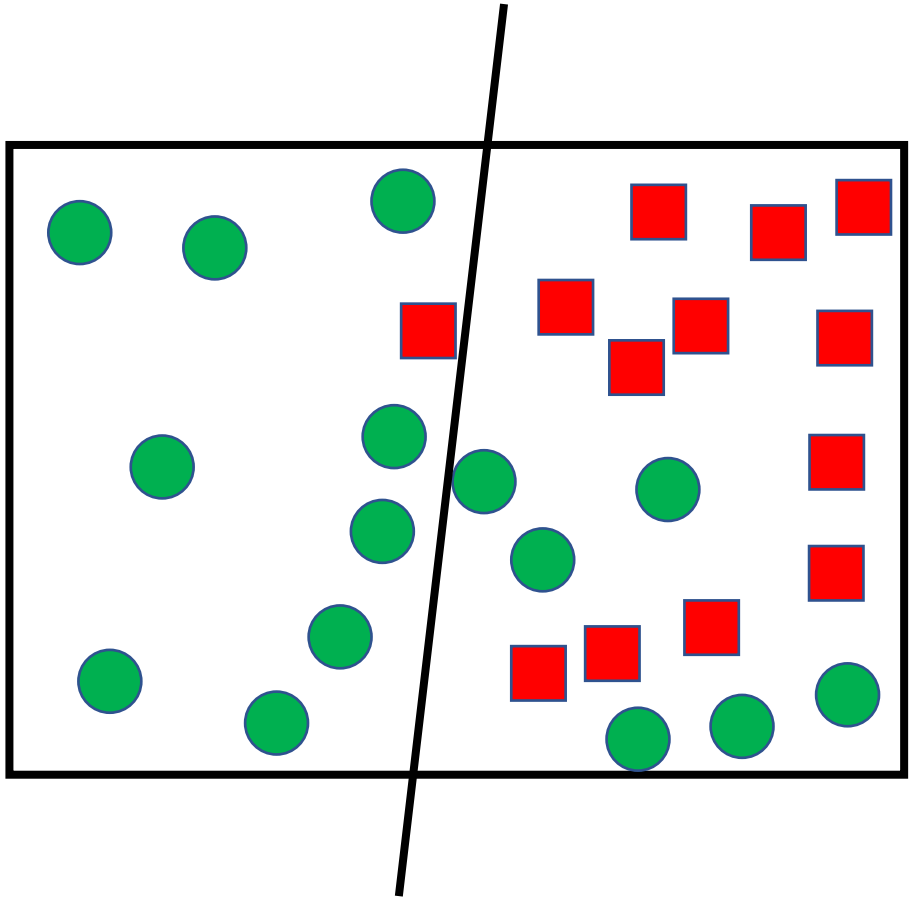
Let’s try to do it in a linear fashion.

Simplified view in 2 dimensions



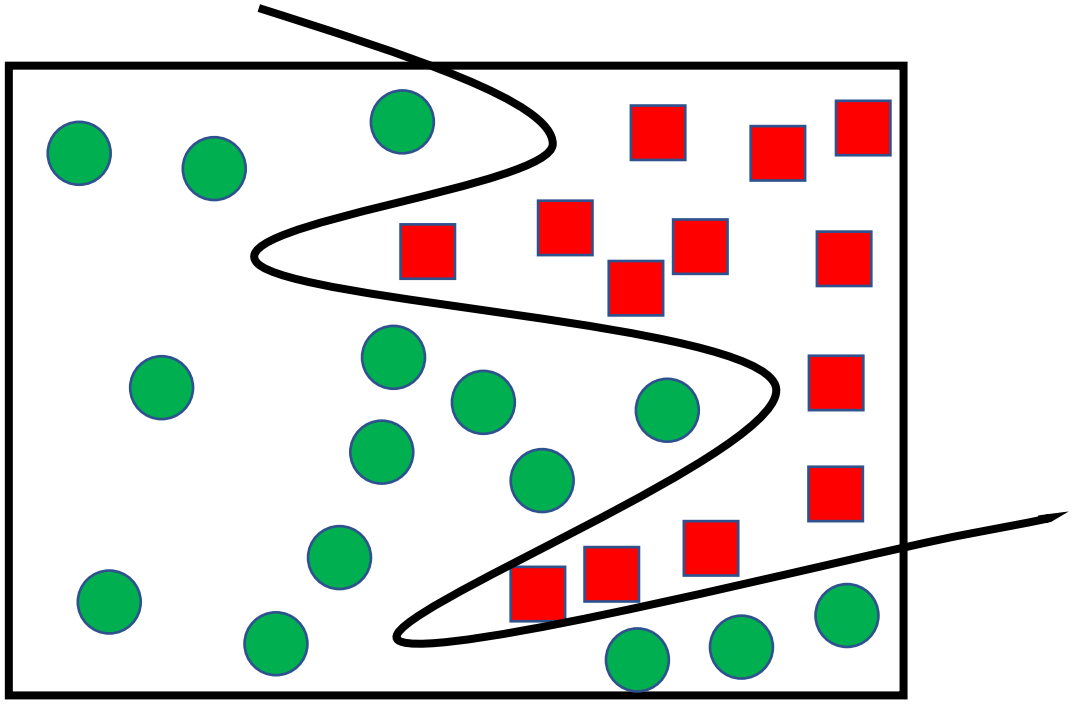
In some cases, it is not possible to be perfect but good enough…

Simplified view in 2 dimensions



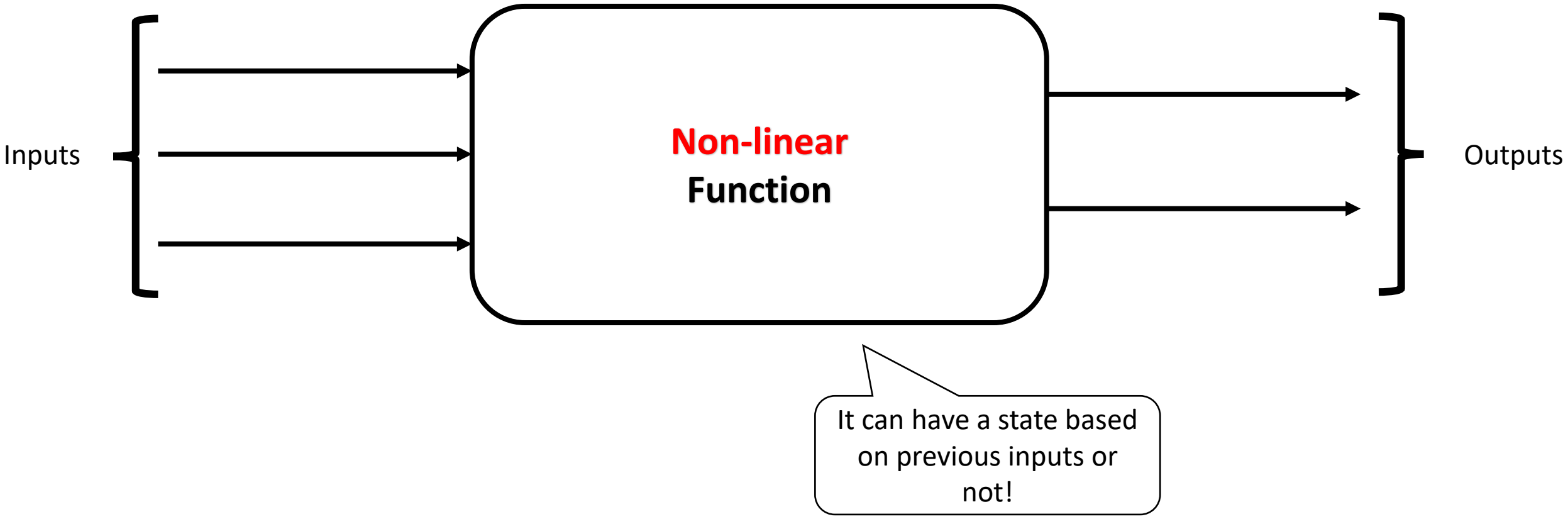
… but for others the error becomes large.

Simplified view in 2 dimensions



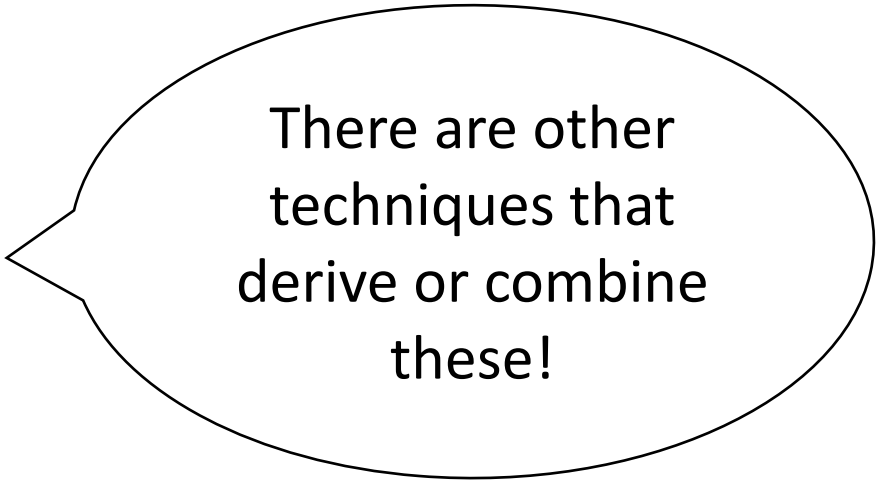
Then, a non-linear approach leads to better results!

An intuition on machine learning



Broad categories of training

- Supervised learning
  - Training data has expected output values given for the inputs (labelled data)
- Unsupervised learning
  - The model should identify patterns in unlabelled data
- Reinforcement learning
  - The model interacts with an environment to perform a goal receiving a feedback

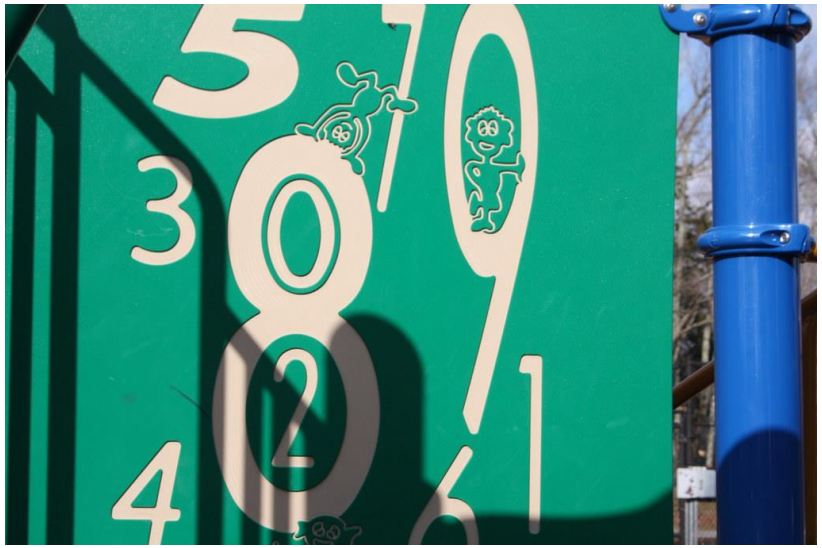


Supervised learning – Example: classification

- Classify an input in 1 of N classes

- Examples:
  - Given a picture, identify the digit between 0 and 9
    - 10 classes

  - Given a sentence, tell if it is gramatically correct or not
    - 2 classes => binary classification



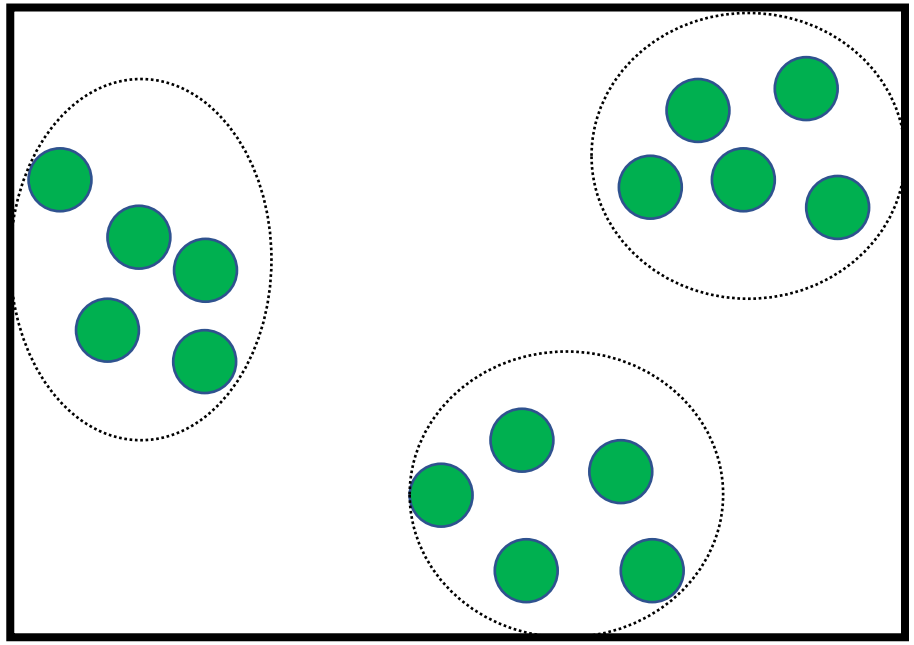
Challenge of supervised learning

- Need of labelled data
  - Time and effort required

Unsupervised example: clustering

- Identify groups of similar instances

- Examples:
  - Classify customers based on their previous purchases
  - Recommender systems



Supervised vs unsupervised learning

Supervised

- Deals with inputs and expected outputs
- Goal: to predict the value given other inputs
- Example: classification

Unsupervised

- Deals only with the inputs
- Goal: build a data model to
  - Reveal structures
  - Detect anomalies
- Example: clustering

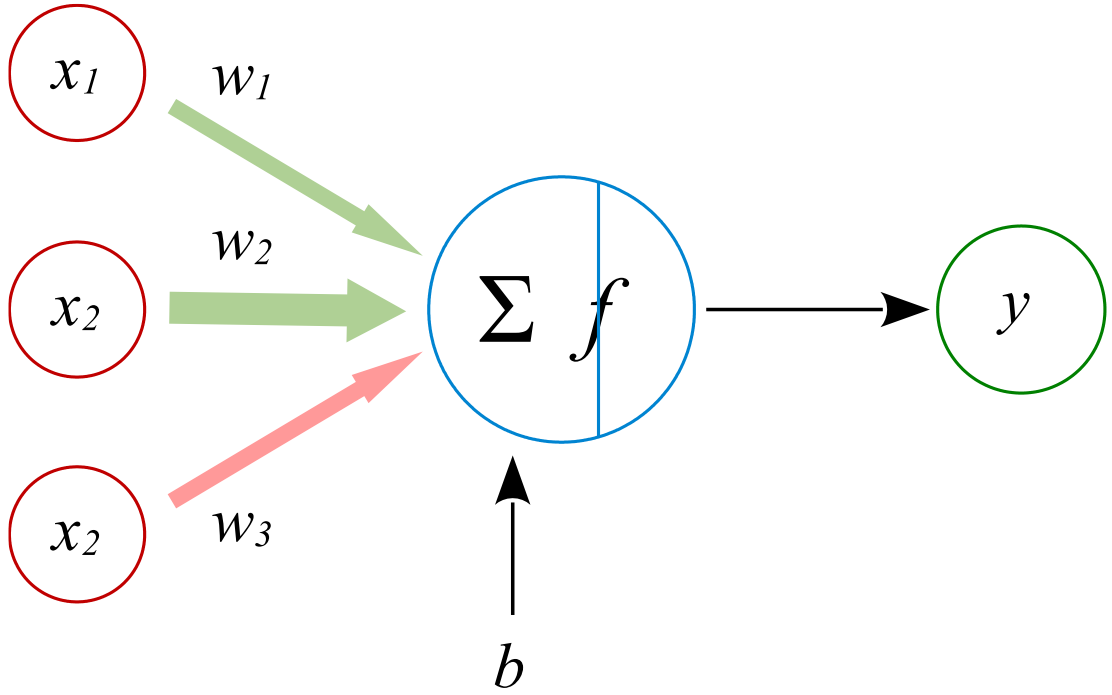
Machine learning vs neural networks

- Neural networks (NN) are ML-models based on artificial neurons, mathematical functions inspired by biological neurons.

- Besides NN, ML can be based on, for example:
  - Genetic algorithms
  - Support Vector Machines (SVMs)

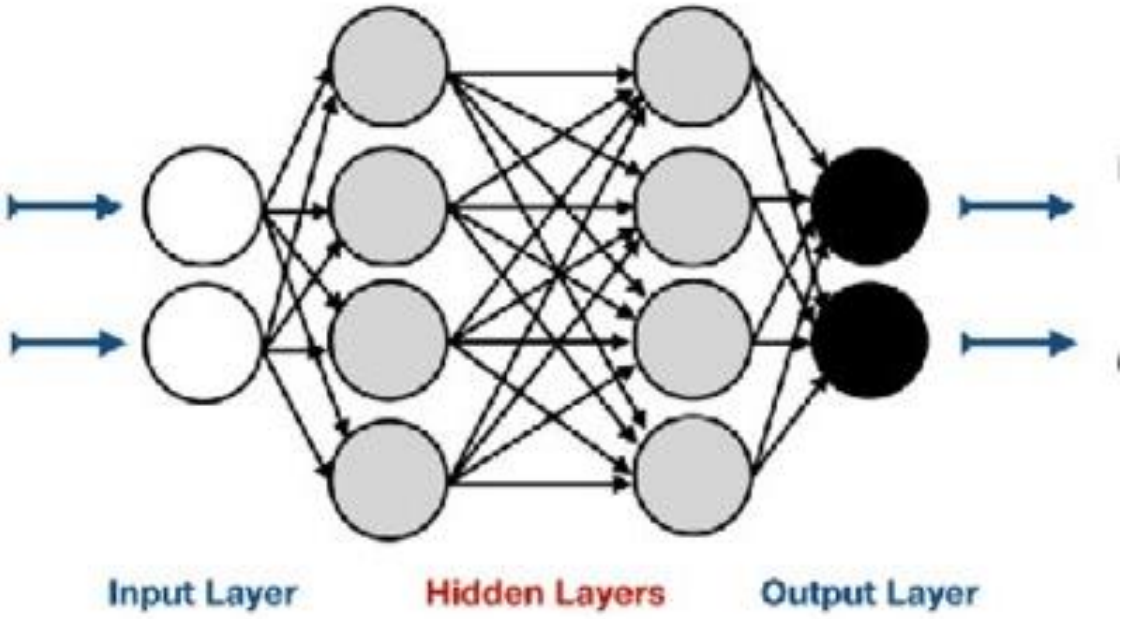
Artificial neuron

- The neuron receives one or more inputs…
- … that are generally weighted and summed (often with an extra constant bias included)…
- … which is passed through a non-linear function (activation function)



Layers of neural networks

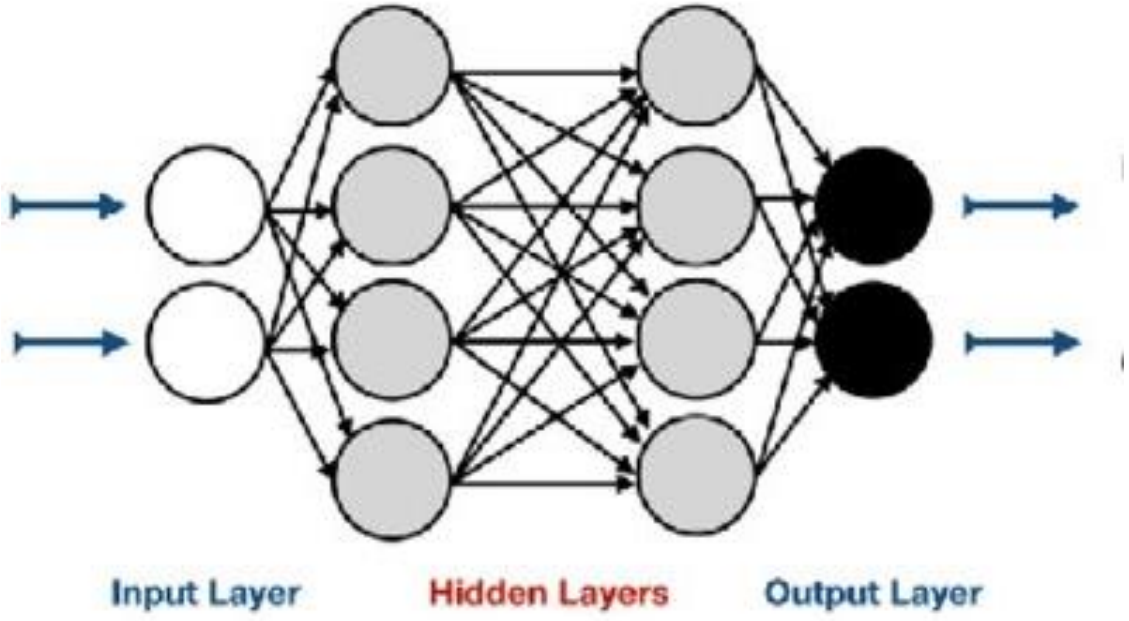
- Neurons could be organized in layers such that:
  - The outputs of a layer are used as inputs for the next neurons



Feed forward networks

- When the outputs of a layer are used as inputs of the next layer (only one direction), they are called feed forward

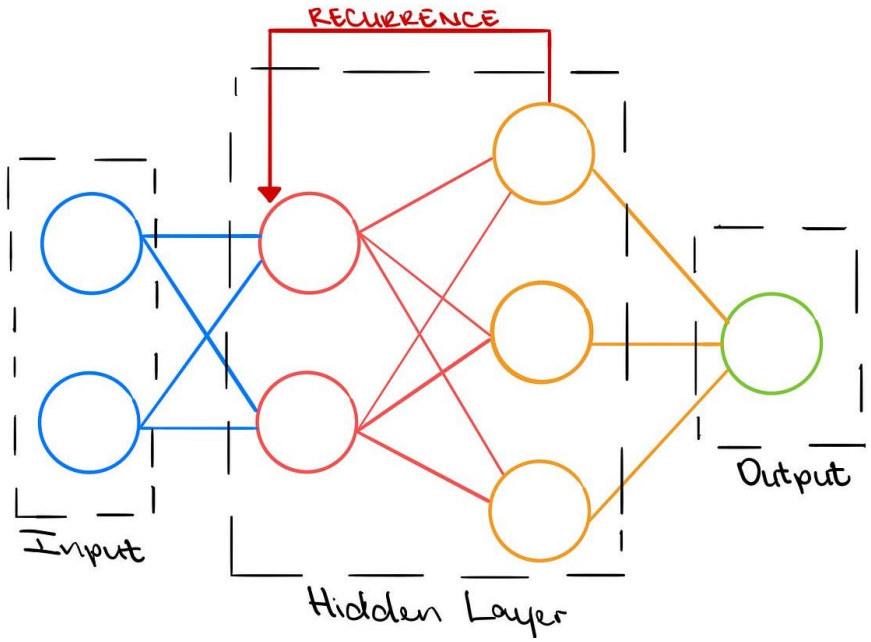
- The output is a function of the inputs



Recurrent neural networks (RNN)

- When some outputs of a layer are used as input of previous layers, the network is called recurrent

- The output is a function of the input and a hidden state (memory)



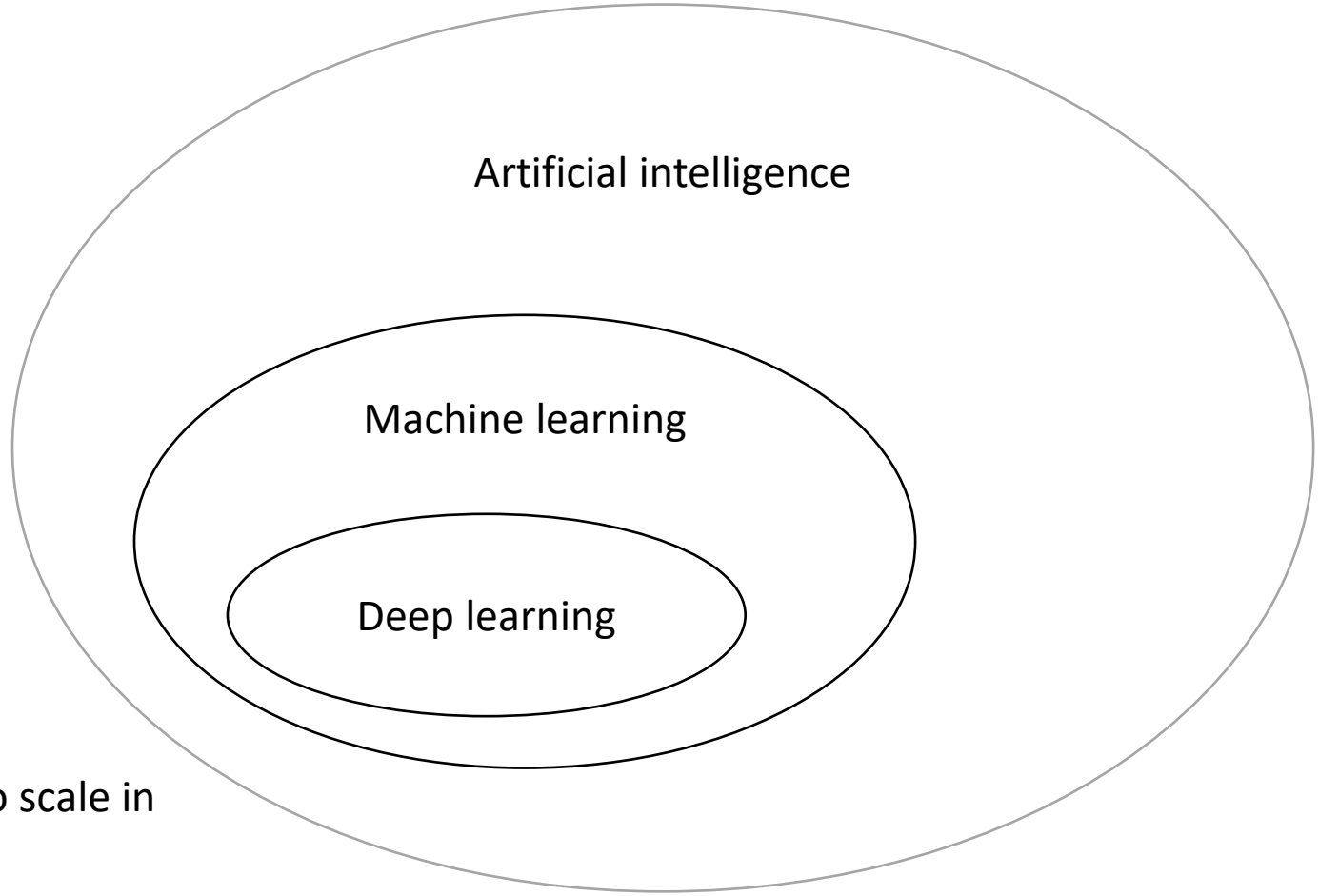
Advantages and disadvantages of RNNs

- Effective with sequential data
- Consider the context ("memory")

- Limited memory
  - Problems with long sentences
- Bias to recent data
- Sequential nature
  - Each element of the input is provided at a time
  - Precludes parallelization

Deep learning

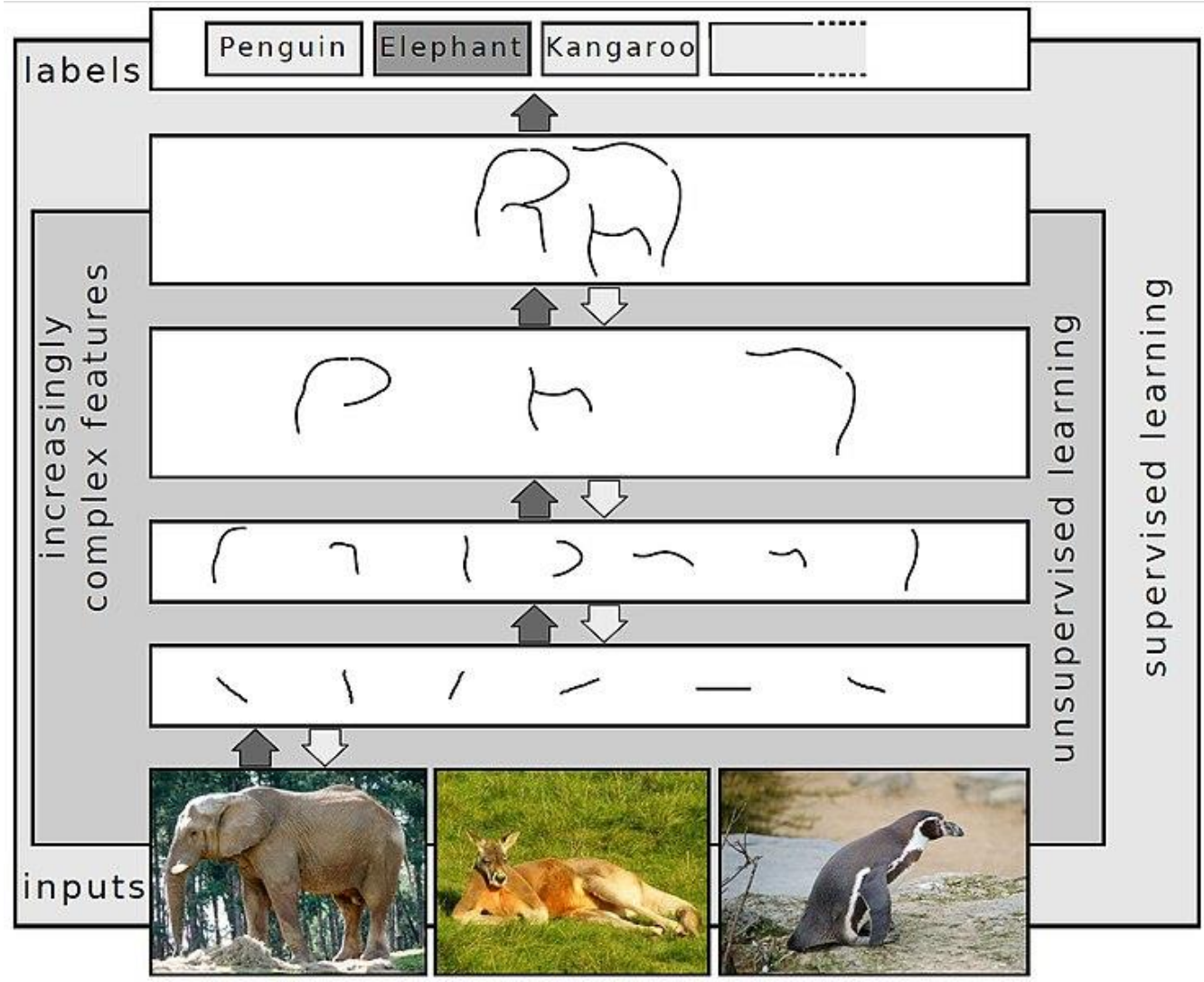
- A subset of ML that employs neural networks with several layers to do complex tasks.



Important: there is no scale in this picture!

Deep learning

- Intuition:
  - Different layers represent different levels of abstraction



Representing Inages on Multiple Levels of Abstraction in Deep Learning by Sven Behnke. Published under CC BY-SA 4.0. Available at: https://commons.wikimedia.org/wiki/File:Deep_Learning.jpg

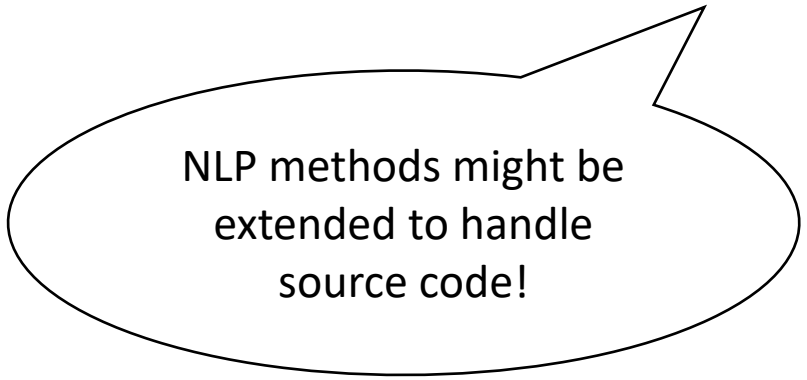
Natural Language Processing (NLP)

- The branch of AI concerned with giving the ability to handle natural language (what humans speak or write)

- NLP tasks:
  - Speech recognition
  - Natural-language understanding
  - Natural-language generation

The naturalness hypothesis

"Software is a form of human communication; software corpora have similar statistical properties to natural language corpora; and these properties can be exploited to build better software engineering tools."



A Survey of Machine Learning for Big Code and Naturalness

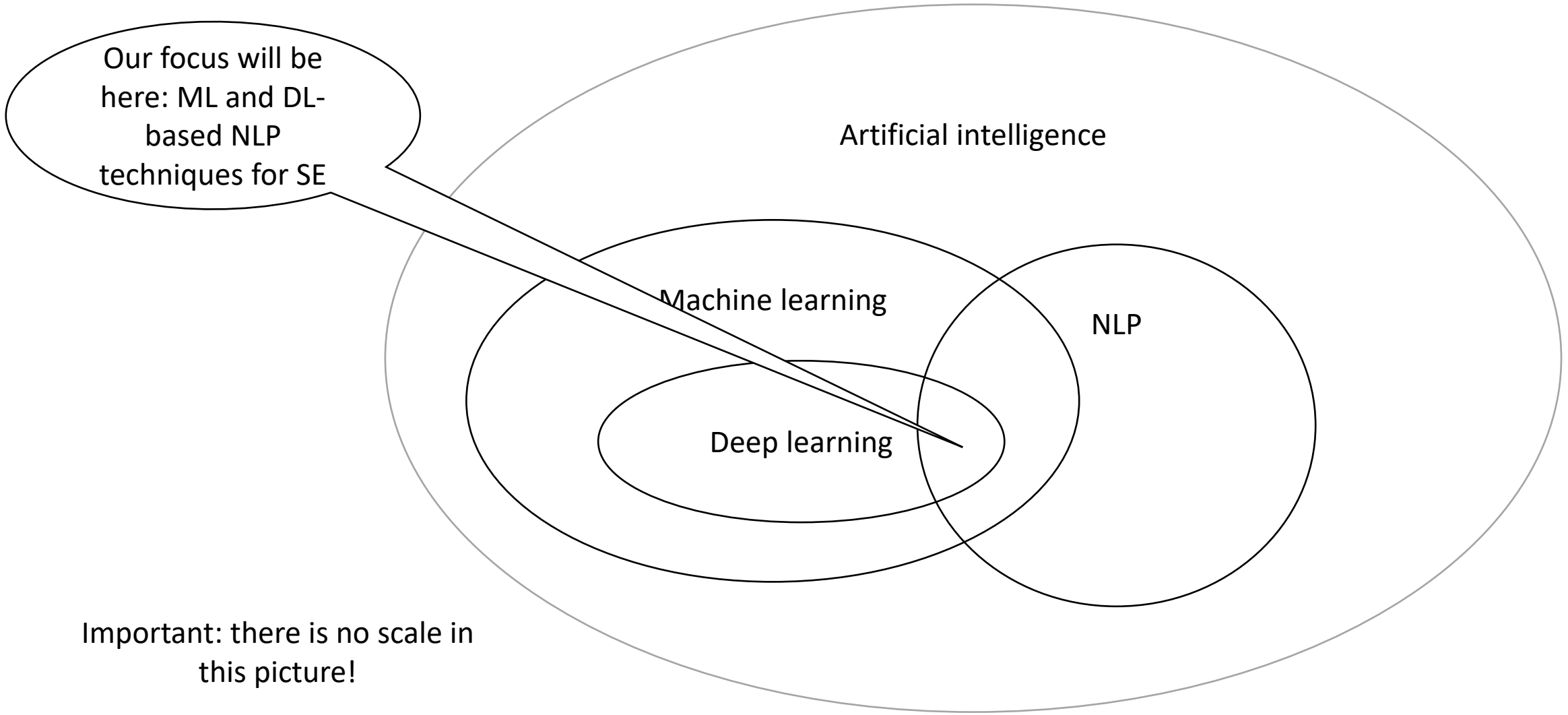
MILTIADIS ALLAMANIS, Microsoft Research
EARL T. BARR, University College London
PREMKUMAR DEVANBU, University of California, Davis
CHARLES SUTTON, University of Edinburgh and The Alan Turing Institute

Research at the intersection of machine learning, programming languages, and software engineering has recently taken important steps in proposing learnable probabilistic models of source code that exploit the abundance of patterns of code. In this article, we survey this work. We contrast programming languages against natural languages and discuss how these similarities and differences drive the design of probabilistic models. We present a taxonomy based on the underlying design principles of each model and use it to navigate the literature. Then, we review how researchers have adapted these models to application areas and discuss cross-cutting and application-specific challenges and opportunities.

CCS Concepts: • **Computing methodologies** → **Machine learning**; Natural language processing; • **Software and its engineering** → **Software notations and tools**; • **General and reference** → *Surveys and overviews*;

Additional Key Words and Phrases: Big code, code naturalness, software engineering tools, machine learning

ML vs NLP



Important: there is no scale in this picture!

Code encoding or embedding

- Neural networks are mathematical functions
  - Inputs should be numbers!
- Problem: how to represent code (text) as numbers?
- Encoding is transforming input data is something machine-digestable (not a formal definition!)

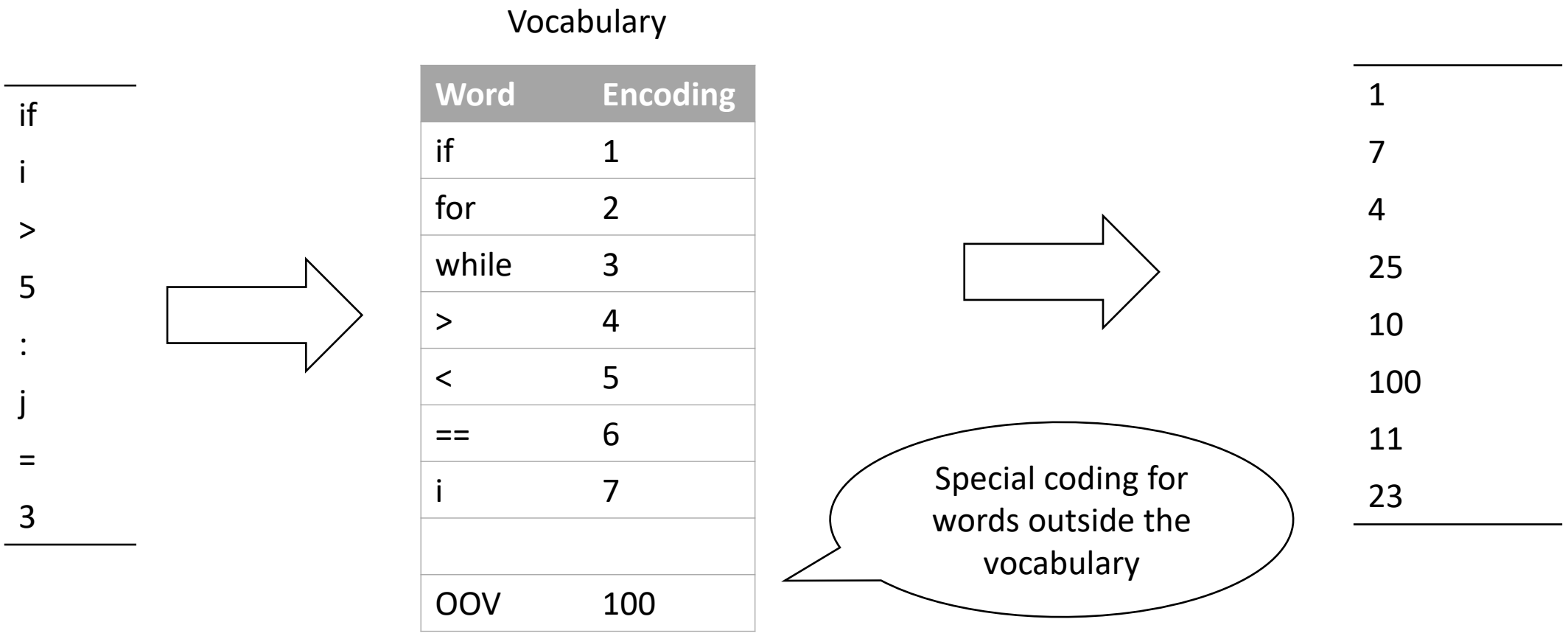
Distance preservation

- Distance between two values might be an important feature in the data

- We might want to maintain this distance in the representation!

Single value encoding

- Each value is converted into a single integer number



Single value encoding – Positive and negative

- Each value is converted into a single integer number
- Positive aspects: easy to compute, small in memory
- Negative aspects: often a complete dictionary cannot be built

Vectorization

- Each value is converted into an array of numbers
- Positive: information gain
- Negative: more preprocessing needed

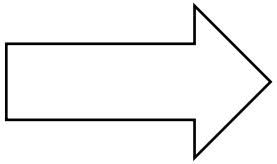
One hot encoding

- A dictionary containing all (or most of) possible values is created and sorted

- Each value is converted to an array consisting of zeros except for the position of the value in the dictionary

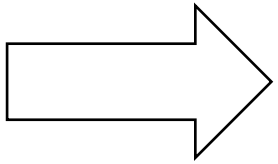
One-hot encoding

if
i
>
5
:
j
=
3



Vocabulary

Word	Position
if	1
for	2
while	3
>	4
<	5
==	6
i	7
OOV	100



[1 0 0 0 0 0 0 … 0]
[0 0 0 0 0 0 1 … 0]
[0 0 0 1 0 0 0 … 0]
[0 0 0 … 0 1 0 …0]
[0 0 0 … 0 1 0 …0]
[0 0 0 … 0 0 0 0 1]
[0 0 0 … 0 1 0 …0]
[0 0 0 … 0 1 0 …0]

One hot encoding – positive and negative

- Positive: easy to compute, might preserve distance

- Negative: large use of memory, often a complete dictionary could not be buit

Word2Vec

- Created by researchers at Google
- Idea: create a vector to represent semantic
- A model for learning vector representations of words, called “word embeddings”

Word2Vec goals

- (Water – wet) + fire = flames
- (Paris – France) + Italy = Rome

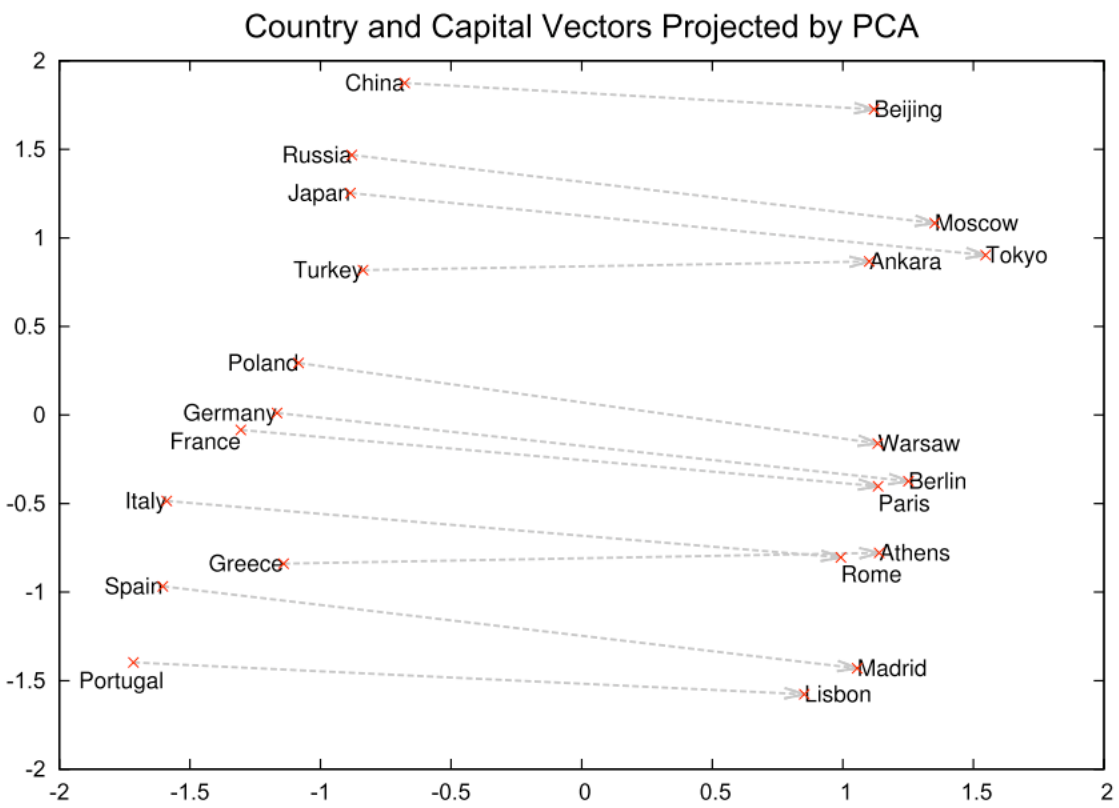


Figure 2: Two-dimensional PCA projection of the 1000-dimensional Skip-gram vectors of countries and their capital cities. The figure illustrates ability of the model to automatically organize concepts and learn implicitly the relationships between them, as during the training we did not provide any supervised information about what a capital city means.

Source: Mikolov, Tomas, et al. "Distributed representations of words and phrases and their compositionality." *Advances in neural information processing systems* 26 (2013).

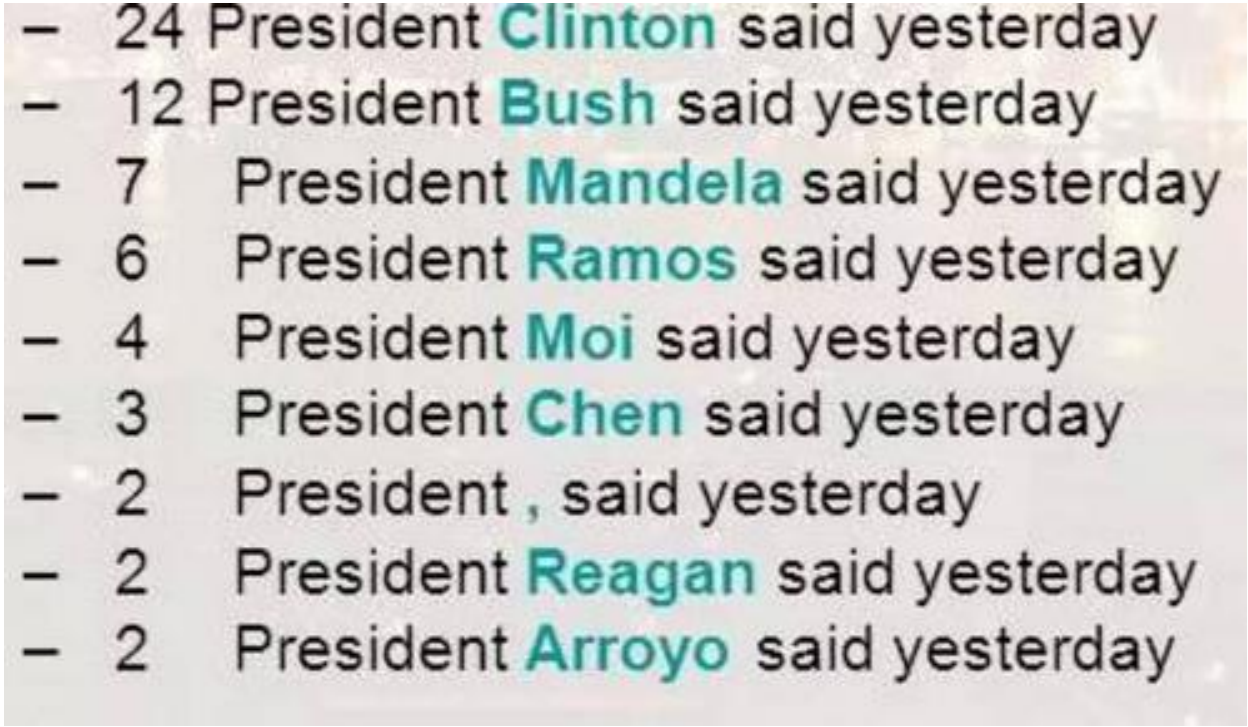
Word2Vec

- Represents words in a continuous vector space
- The vector space is relatively low-dimensional (100-300 dimensions)
- Semantically and syntactically similar words are mapped to nearby points

Word2Vec implementation

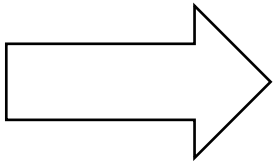
- Learning word embeddings from raw text

- Words that occur in similar contexts tend to be similar



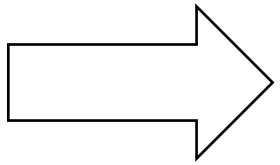
Word2Vec

if
i
>
5
:
j
=
3



Vocabulary (vectors learned)

Word	Position
if	[0.2 0.1 0.7 … 0 0.3]
for	[0.4 0.2 0.5 … 0.1 0.2]
while	[0.6 0.2 0.1 … 0.9 0.9]
>	[0 0.3 0.4 … 0.5 0.2]
<	[0.1 0.2 0.5 … 0.1 0.3]
==	[0.2 0.2 0.6 … 0 0.3]
i	[0.8 0.6 0.2 … 0.8 0.7]
OOV	[0.9 0 0.2 … 0.6 0.5]



[0.2 0.1 0.7 … 0 0.3]
[0.8 0.6 0.2 … 0.8 0.7]
[0 0.3 0.4 … 0.5 0.2]
[0.8 0.3 0.1 … 0.3 0.2]
[0.4 0.3 0.2 … 0.9 1.0]
[0.6 0.1 0.1 … 0.1 1.0]
[0.7 0.5 0.2 … 0.1 0]
[0.7 0.4 0.1 … 0.2 0.3]

Advantages of Word2Vec

- Scalability
  - Trained on large corpora
- Reusability
  - Pre-trained word embeddings can be used for other tasks
- Flexibility of the size of the embedding
  - Compare it to one hot encoding
- No need of human labelling (unsupervised learning)

Limitations of Word2Vec

- Inability to handle out-of-vocabulary (OOV) words
- No shared representations of sub-word levels
- Polysemic words
  - The bird uses the wings to fly. (wing = part of the airplane)
  - The university is building a new wing. (wing = part of the building)
  - The word “wing” would have the same representation on Word2Vec

Byte-Pair Encoding (BPE) for tokenization

- A form of subword tokenization
  - Middle ground between word-level tokenization and character-level tokenization

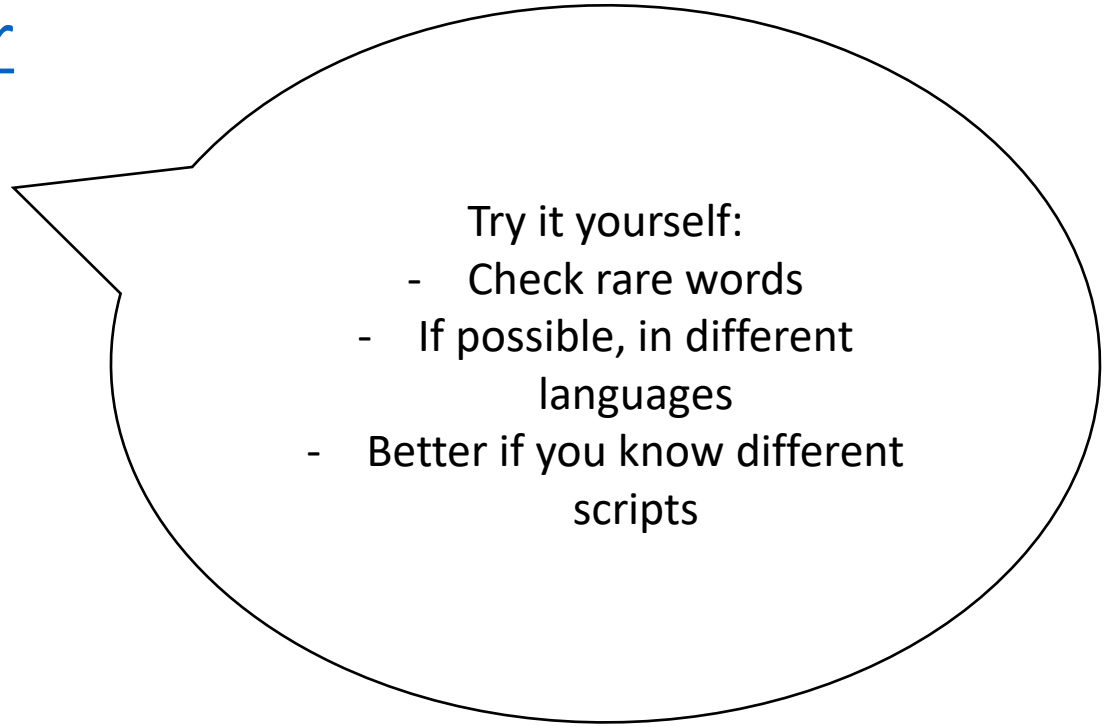
- Most (common) words are encoded as a single token
- Rare words and/or long are enconded a sequence of the tokens

- Example: order = 10, unordered = un – order – ed = 2 – 10 – 1

Example: OpenAI’s tokenizer tool

- https://platform.openai.com/tokenizer

- https://github.com/openai/tiktoken



Byte-Pair Encoding (BPE) for tokenization

- Advantages:
  - Reduced size of the vocabulary
  - Related words have similar encodings

- Still, it does not solve the problem of context

ELMo (Embeddings from Language Model)

- Character-level tokens (whole sentence) are input to a bidirectional language model (biLM)
- Output: word-level embeddings
- Solve the issues:
  - Context (syntax and semantics)
  - Polysemy