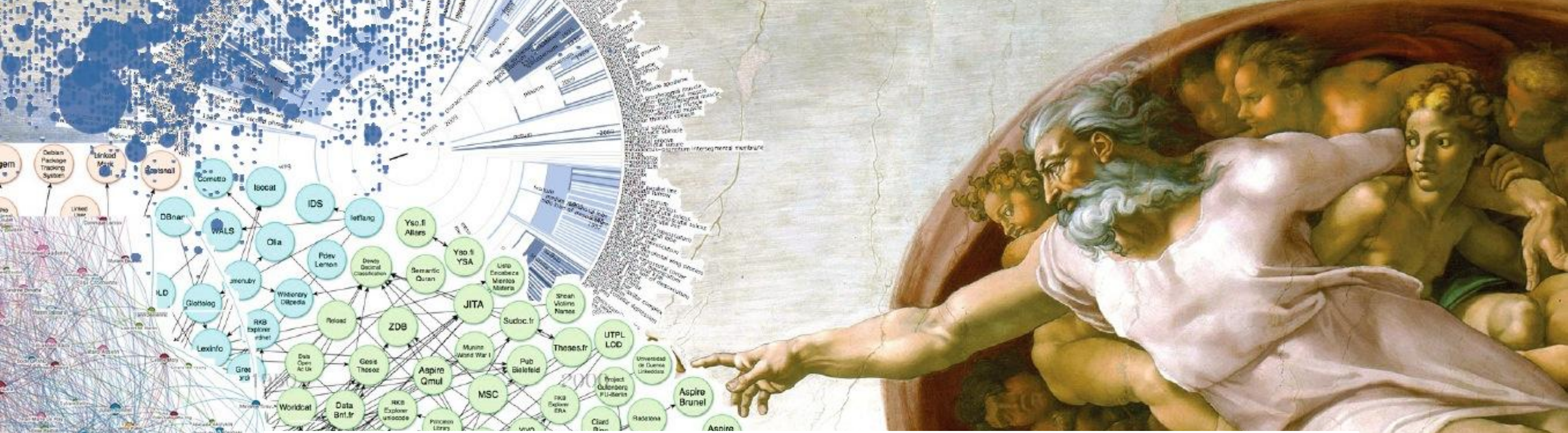


Information Service Engineering

Lecture 12: Basic Machine Learning - 3

Karlsruher Institut für Technologie

FIZ Karlsruhe
Leibniz Institute for Information Infrastructure

Prof. Dr. Harald Sack
FIZ Karlsruhe - Leibniz Institute for Information Infrastructure
AIFB - Karlsruhe Institute of Technology

Summer Semester 2021

Information Service Engineering

Last Lecture: Machine Learning - 2

- Unsupervised vs. supervised learning
- k-Means Clustering
- Linear Regression
- Decision Trees

4.1 A Brief History of AI

4.2 Introduction to Machine Learning

4.3 Main Challenges of Machine Learning

4.4 Machine Learning Workflow

4.5 Basic ML Algorithms 1 - k-Means Clustering

4.6 Basic ML Algorithms 2 - Linear Regression

4.7 Basic ML Algorithms 3 - Decision Trees

4.8 Neural Networks and Deep Learning

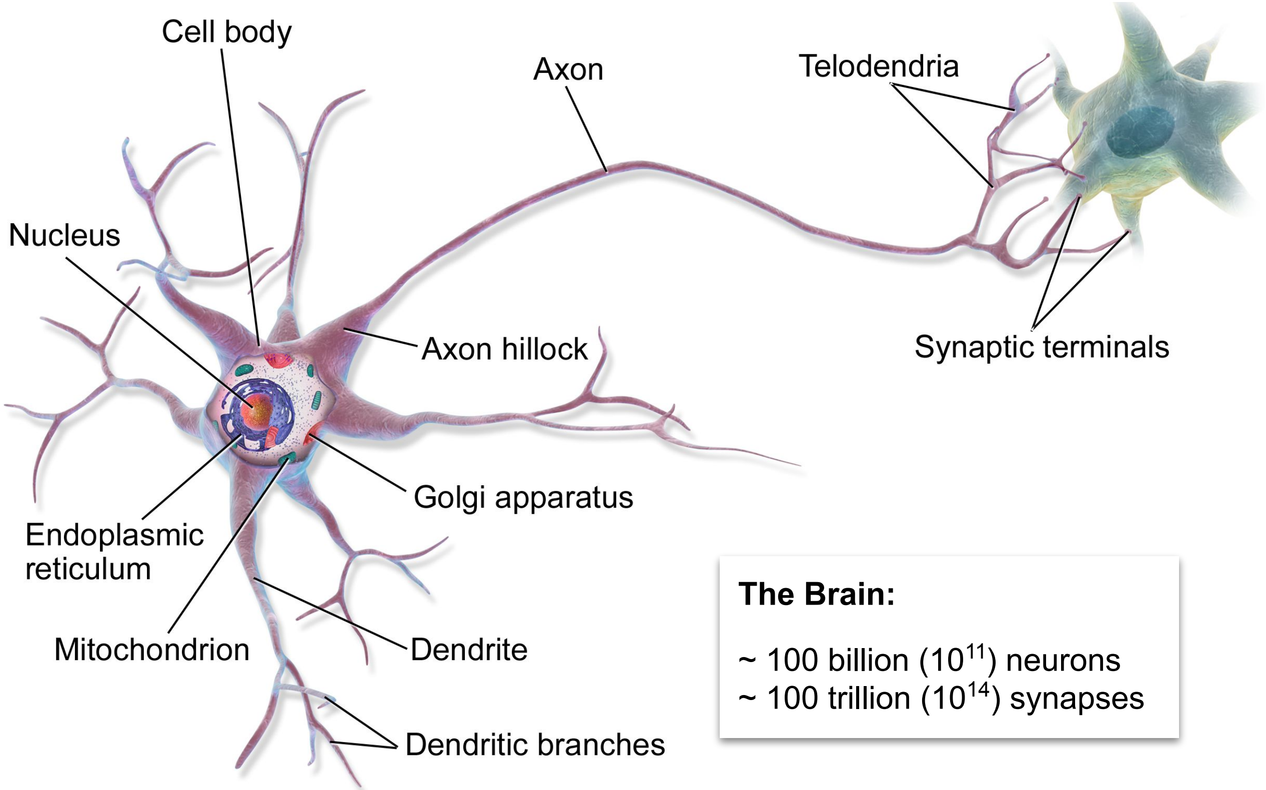
4.9 Knowledge Graph Embeddings

4.10 Knowledge Graph Completion

Biological Neural Networks

Karlsruher Institut für Technologie
FIZ Karlsruhe
Leibniz Institute for Information Infrastructure

Donald O. Hebb, *The Organization of Behavior* (1949)



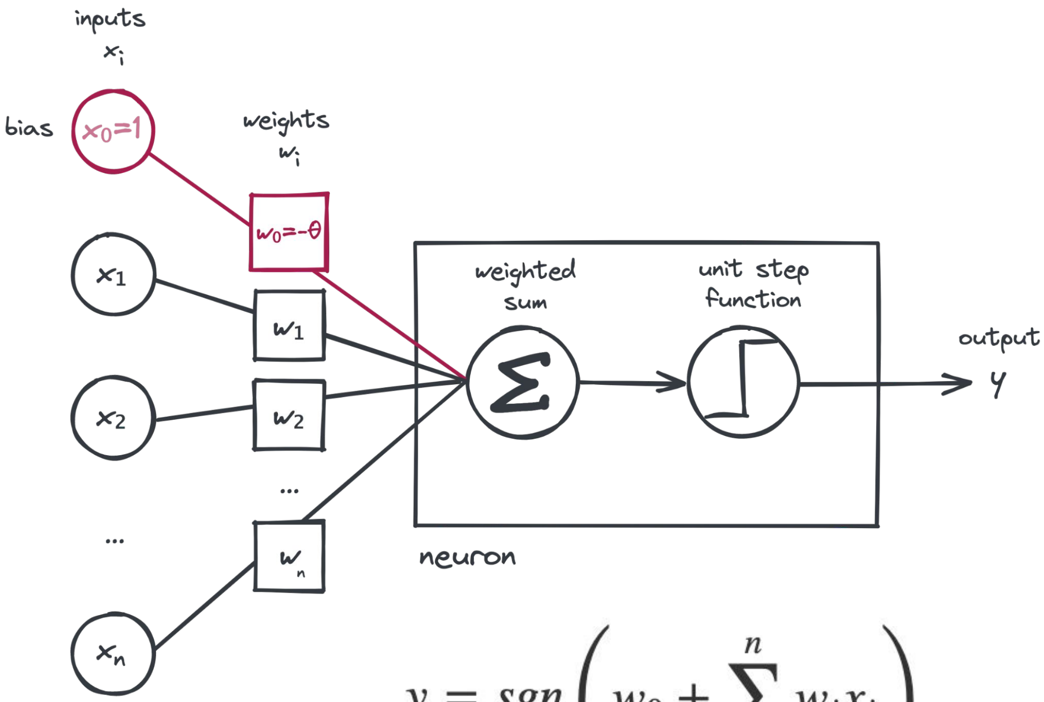
The Brain:

~ 100 billion (10^{11}) neurons
~ 100 trillion (10^{14}) synapses

- The majority of **neurons** encode their outputs or activations as a series of brief electrical pulses.
- **Dendrites** are the receptive zones that receive activation from other neurons.
- The **cell body** (**soma**) of the neuron's processes the incoming activations (excitatory and inhibitory) and converts them into output activations.
- **Axons** are transmission lines that send activation to other neurons.
- **Synapses** allow weighted transmission of signals (via **neurotransmitters**) between axons and dendrites to build up large neural networks.
- **All-or-one response:** A higher stimulus does **not** cause a higher response. → "binary classifier"

McCulloch-Pitts Neurons

McCulloch & Pitts, *A Logical Calculus of the Ideas Immanent in Nervous Activity* (1943)



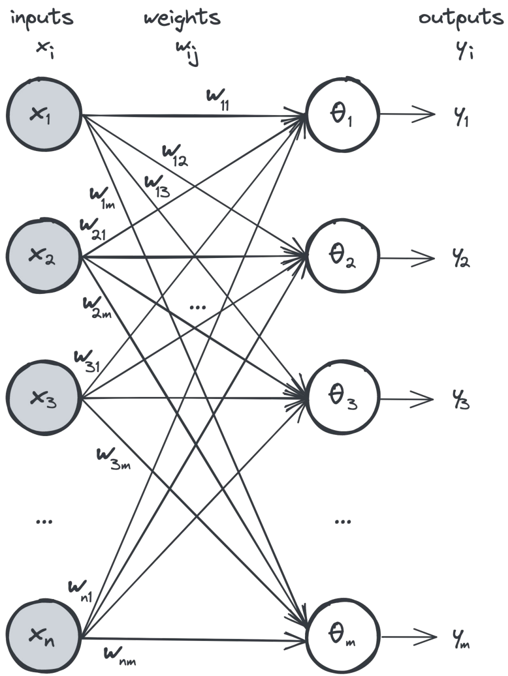
$$y = sgn\left(w_0 + \sum_{i=1}^{n} w_i x_i\right)$$

- **Artificial neurons** have the same basic components as biological neurons.
- The simplest Artificial Neural Networks (ANN) consist of a set of **McCulloch-Pitts neurons**.
- There is a **threshold bias** $x_0=1$, $w_0=-\theta$
  - Allows bias to be captured from input neurons.
  - Allows normalization of output thresholds without loss of generality.

Perceptron

Frank Rosenblatt, *The perceptron. A probabilistic model for information storage and organization in the brain* (1958)

An arrangement of **one input layer of activations feeding forward to one output layer** of McCulloch-Pitts neurons is known as a **simple Perceptron**.

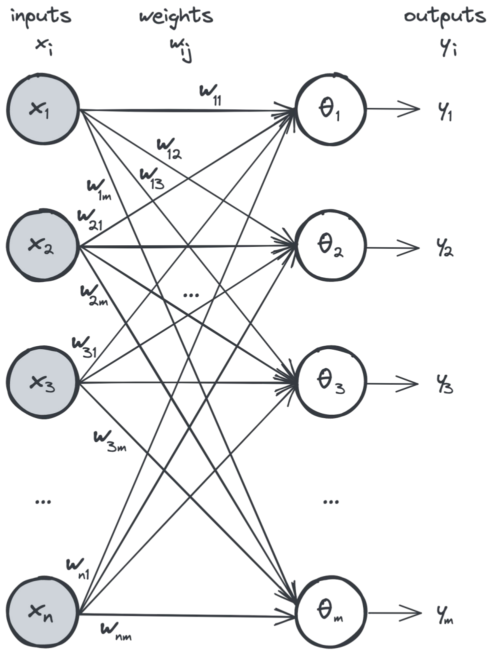


- **Input layer** with n inputs $x_1,\ldots,x_n$.
- **Output layer** of m McCulloch-Pitts neurons $y_1,\ldots,y_m$ with associated thresholds $\theta_1,\ldots,\theta_m$.
- Each input x_i is connected to each output neuron y_j with an associated **weight** $w_{i,j}$.
- For the **training**, each output y_j computed by the perceptron will be compared with a **desired output** $d_1,\ldots,d_m$.
- **Training a perceptron** means **adapting the weights** w_{ij} until they fit input-output relationships of the given ‘training data‘.

What the Perceptron can Do

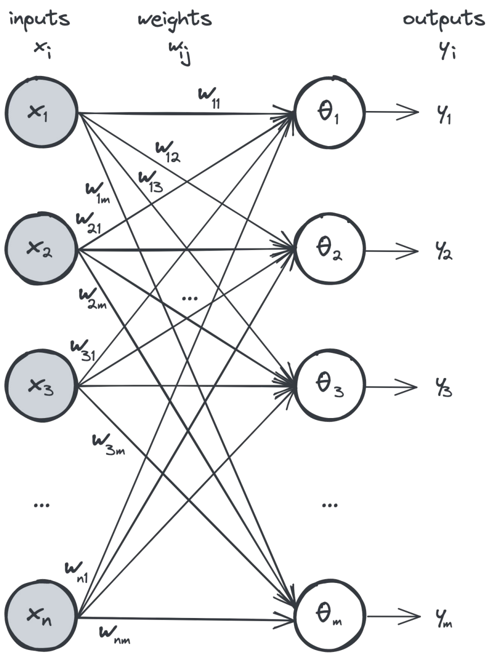
Learning with a Perceptron

- Initialize weights randomly
- Take one sample x_i and predict y_i
- For erroneous predictions update weights
    - If the output was $y_i = 0$ and desired output $d_i = 1$, **increase weights**.
    - If the output was $y_i = 1$ and desired output $d_i = 0$, **decrease weights**.
- Repeat until no errors are made.



Perceptron (2)

An arrangement of **one input layer of activations feeding forward to one output layer** of McCulloch-Pitts neurons is known as a simple **Perceptron**.



Network activation: for each neuron in output layer $j=1,\ldots,m$

$$y_j = sgn\left(\sum_{i=1}^{n} w_{ij}\, x_i - \Theta_j\right)$$

Perceptron learning function:
for each feature weight w_{ij}, $i=1,..,n$ of neuron in output layer $j=1,..,m$

$$w_{ij}(t+1) = w_{ij}(t) + \Delta w_{ij}(t)$$
$$\Delta w_{ij} = \eta\left(d_j - y_j(t)\right) x_i$$

learning rate coefficient

desired output

actual output

Perceptron Learning Algorithm

Input vector	$\mathbf{x}(t) = [+1, x_1(t), x_2(t), ..., x_n(t)]^T$
Weight vector	$\mathbf{w}(t) = [-\theta, w_1(t), w_2(t), ..., w_n(t)]^T$, θ = bias
Actual response	$y(t)$ (quantized)
Desired response	$d(t)$
Learning-rate	η, $0 < \eta \leq 1$

1. **Initialization.** Set $w(0) = 0$. Then perform the following computations for time-step $t = 1, 2, ...$
2. **Activation.** At time-step t, activate the perceptron by applying continuous-valued input vector $\boldsymbol{x}(t)$ and desired response $d(t)$.
3. **Computation of Actual Response.** Compute the actual response of the perceptron as $y(t) = sgn[\mathbf{w}^\top(t)\,\mathbf{x}(t)]$
4. **Adaptation of Weight Vector.** Update the weight vector of the perceptron to obtain, where

$$\mathbf{w}(t+1) = \mathbf{w}(t) + \eta\,[d(t) - y(t)]\,\mathbf{x}(t) \qquad d(t) = \begin{cases} +1 \text{ if } x(t) \text{ belongs to class } c_1 \\ -1 \text{ if } x(t) \text{ belongs to class } c_2 \end{cases}$$

5. **Continuation.** Increment time step $t = t+1$ by one and go back to step 2.

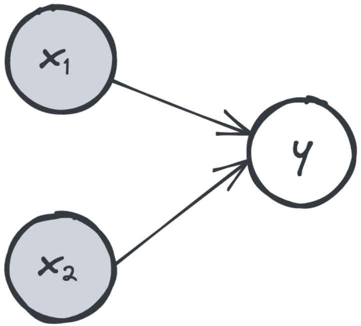
Building an Artificial Neural Network

Using artificial neural networks to solve real problems is a multi-stage process:

1. **Understand** and **specify the problem** in terms of inputs and required outputs.
2. Take the **simplest form of network** that might be able to solve the problem.
3. Try to find appropriate connection **weights** and neuron **thresholds** so that the network produces appropriate outputs for each input in its training data.
4. **Test** the network on its **training data**, and also on **new** (validation/testing) **data**.
5. If the network doesn't perform well enough, go back to 3 and work harder.
6. If the network still doesn't perform well enough, go back to 2 and work harder.
7. If the network still doesn't perform well enough, go back to 1 and work harder.
8. Problem solved – move on to next problem.

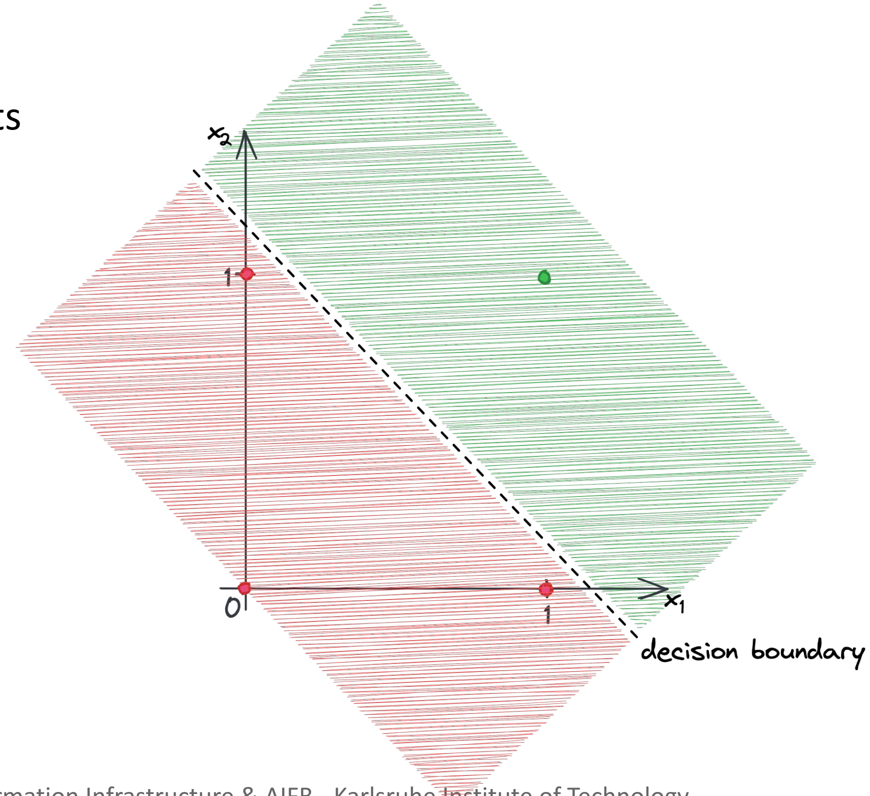
Decision Boundaries

- We can use McCulloch-Pitts neurons to **implement the basic logic gates** (e.g. AND, OR, NOT).
- Train network to calculate the appropriate weights and thresholds in order to correctly classify the different classes (i.e. form **decision boundaries** between classes).



AND

x_1	x_2	y
0	0	**0**
1	0	**0**
0	1	**0**
1	1	**1**

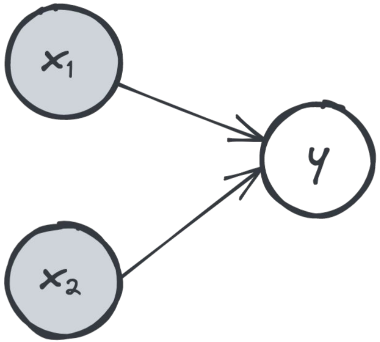


The Problem with the XOR

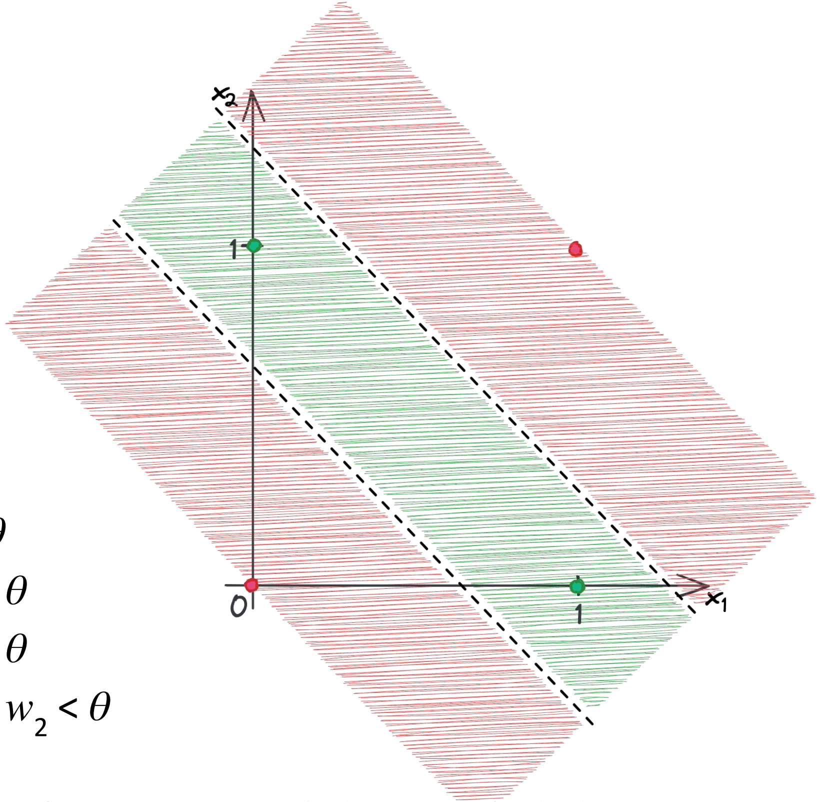
- However, the simple **exclusive or (XOR)** cannot be solved by perceptrons.
  [Minsky and Papert, "*Perceptrons*", 1969]

XOR

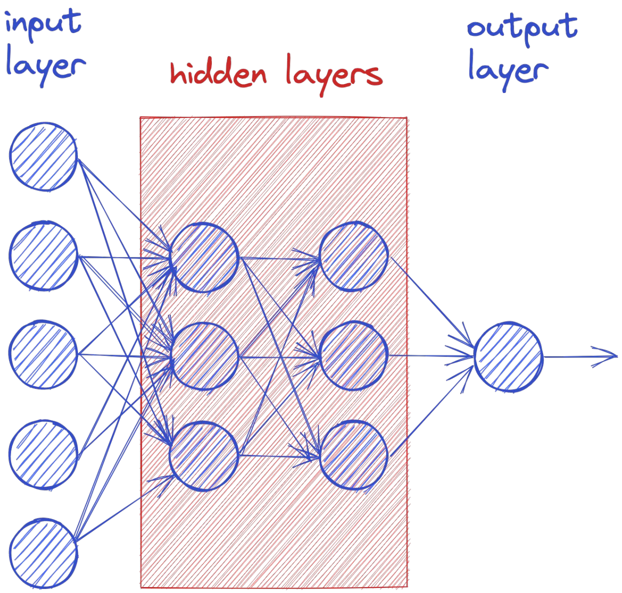
x_1	x_2	y_1
0	0	**0**
0	1	**1**
1	0	**1**
1	1	**0**



- $0\,w_1 + 0\,w_2 < \theta \quad \rightarrow \quad 0 < \theta$
- $0\,w_1 + 1\,w_2 > \theta \quad \rightarrow \quad w_2 > \theta$
- $1\,w_1 + 0\,w_2 > \theta \quad \rightarrow \quad w_1 > \theta$
- $1\,w_1 + 1\,w_2 < \theta \quad \rightarrow \quad w_1 + w_2 < \theta$



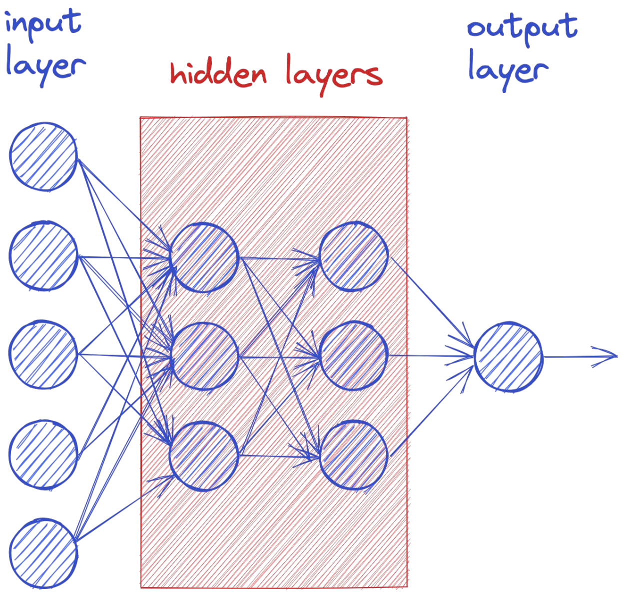
Multilayer Neural Networks



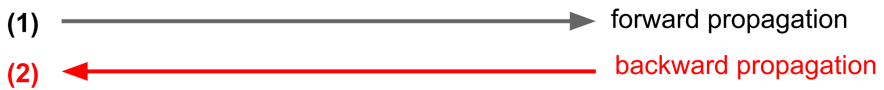
- The perceptron is limited to the classification of linearly separable patterns. How can we make it more flexible?
- **Solution**:
  - Use **non-monotonic activation functions**.
  - Use multilayer neural networks, which include at least **one hidden layer** of neurons with non-linear activations functions.
- **Problem**: Desired output for hidden nodes is not known
- **New Learning Algorithm**: Backpropagation Algorithm

Backpropagation Algorithm

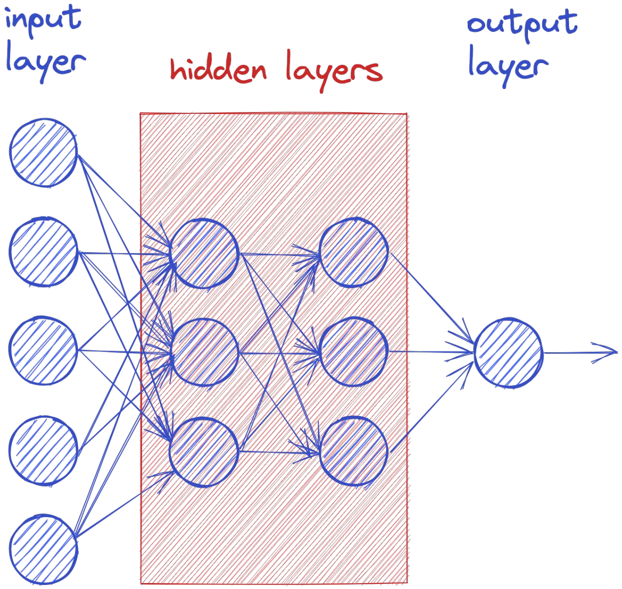
Rumelhart, Hinton & Williams, *Learning representations by back-propagating errors* (1986)



- **Forward phase**
  - Synaptic weights of the network are fixed.
  - Input signal is propagated through the network layer by layer, until it reaches the output.
  - Changes only in activation potentials and outputs.
- **Backward phase**
  - An error signal is produced by comparing network output with desired response.
  - Resulting error signal is propagated through the network layer by layer in the backward direction.
  - Successive adjustments are made to the synaptic weights of the network.

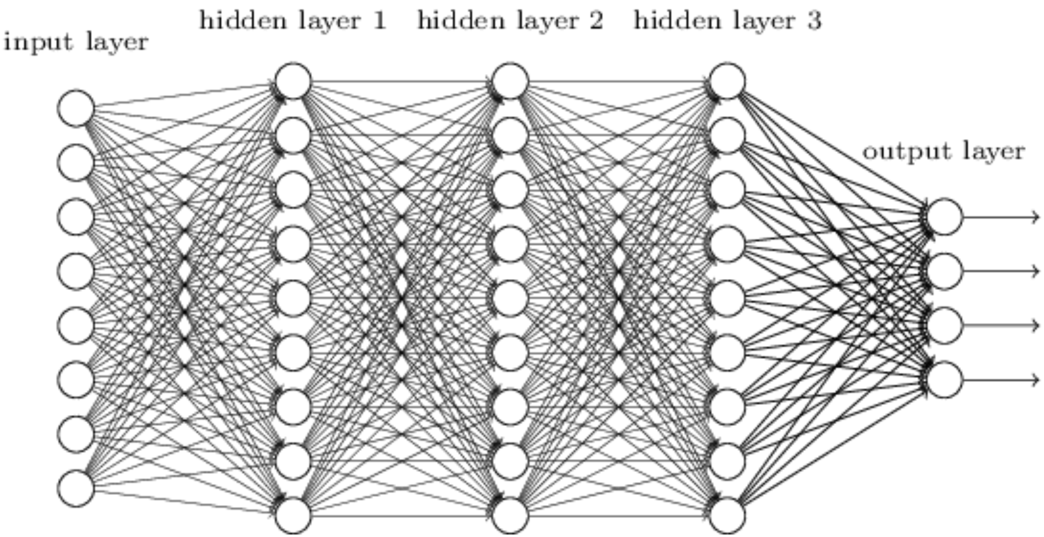


Feature Engineering



- The quality of your Machine Learning approach depends on the appropriate assembly of the **features** used for learning.
- **Feature engineering** is the process of using domain knowledge of the data to create features that make machine learning algorithms work.
- When working on a machine learning problem, feature engineering is manually designing what the input x_i should be.
- Coming up with appropriate features is
  - difficult,
  - time consuming, and
  - requires expert knowledge.

Towards Deep Learning



- **Deep Learning** uses **deep** (multi layer) **neural networks (DNN)**.
- Assume hidden layers with **1000 nodes each**.
- In the given example, we need **2 Mio parameters** to optimize between the hidden layers only.
- To classify images or for object detection, there would be no spatial invariance, i.e. **lack of generalization**.

About the Term “Deep Learning”

*“**Representation learning** is a set of methods that allows a machine to be fed with raw data and to automatically discover the representations needed for detection or classification. Deep-learning methods are representation-learning methods with multiple levels of representation [...]”*

-- LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. Nature, 521(7553), 436.

Visual Analysis - Image Classification Tasks

→ Cat

Challenges:

- Viewpoint variation
- Background clutter
- Different Illumination
- Occlusion
- Deformation
- Intra class variation

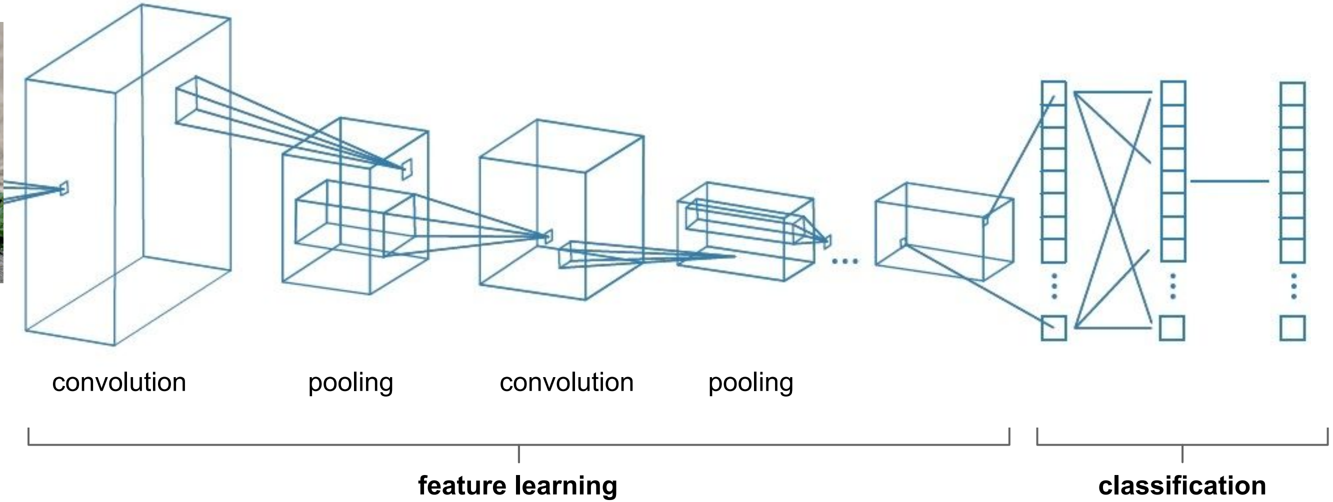
- An image is a tensor of integers between [0, 255], as e.g. 800 x 600 x 3 (3 channels RGB)

```
[[105 112 108 111 104  99 106  99  96 103 112 119 104  97  93  87]
 [ 91  98 102 106 104  79  98 103  99 105 123 136 110 105  94  85]
 [ 76  85  90 105 128 105  87  96  95  99 115 112 106 103  99  85]
 [ 99  81  81  93 120 131 127 100  95  98 102  99  96  93 101  94]
 [106  91  61  64  69  91  88  85 101 107 109  98  75  84  96  95]
 [114 108  85  55  55  69  64  54  64  87 112 129  98  74  84  91]
 [133 137 147 103  65  81  80  65  52  54  74  84 102  93  85  82]
 [128 137 144 140 109  95  86  70  62  65  63  63  60  73  86 101]
 [125 133 148 137 119 121 117  94  65  79  80  65  54  64  72  98]
 [127 125 131 147 133 127 126 131 111  96  89  75  61  64  72  84]
 [115 114 109 123 150 148 131 118 113 109 100  92  74  65  72  78]
 [ 89  93  90  97 108 147 131 118 113 114 113 109 106  95  77  80]
 [ 63  77  86  81  77  79 102 123 117 115 117 125 125 130 115  87]
 [ 62  65  82  89  78  71  80 101 124 126 119 101 107 114 131 119]
 [ 63  65  75  88  89  71  62  81 120 138 135 105  81  98 110 118]
 [ 87  65  71  87 106  95  69  45  76 130 126 107  92  94 105 112]
 [118  97  82  86 117 123 116  66  41  51  95  93  89  95 102 107]
 [164 146 112  80  82 120 124 104  76  48  45  66  88 101 102 109]
 [157 170 157 120  93  86 114 132 112  97  69  55  70  82  99  94]
 [130 128 134 161 139 100 109 118 121 134 114  87  65  53  69  86]
 [128 112  96 117 150 144 120 115 104 107 102  93  87  81  72  79]
 [123 107  96  86  83 112 153 149 122 109 104  75  80 107 112  99]
 [122 121 102  80  82  86  94 117 145 148 153 102  58  78  92 107]
 [122 164 148 103  71  56  78  83  93 103 119 139 102  61  69  84]]
```

What the computer really "sees"

Convolutional Neural Networks

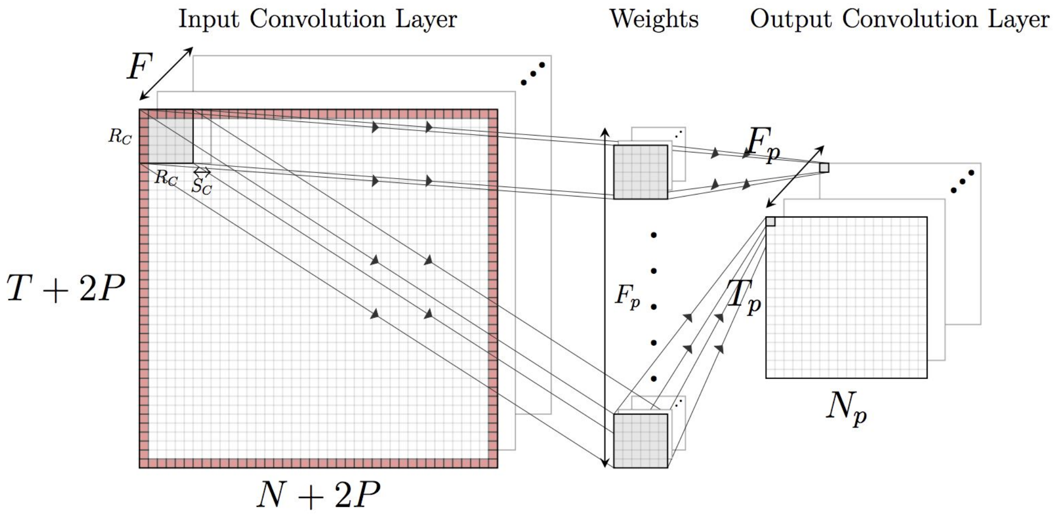
- CNNs are a special type of neural networks for processing **spatially arranged data**.
- CNNs are particularly adapted for **visual analysis tasks**.
- Fundamental building blocks of CNNs: **convolution** and **pooling**.

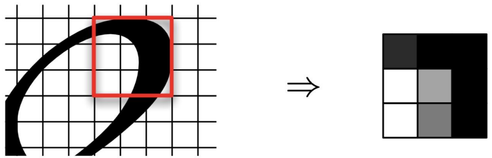


- Convolutional layer: feature extraction
- Pooling layer: compress and aggregate information, save parameters

Convolution Layer

- **Purpose:** examine (filter) the input from different perspectives.
- Each neuron checks a specific area of the input field using a **filter kernel**.
- A filter examines the image for a certain **feature**, e.g. color, edges or brightness.
- The result of a filter is the weighted input of a range and is stored in the **convolutional layer**.
- The depth of the convolutional layer is defined by the number of filters.

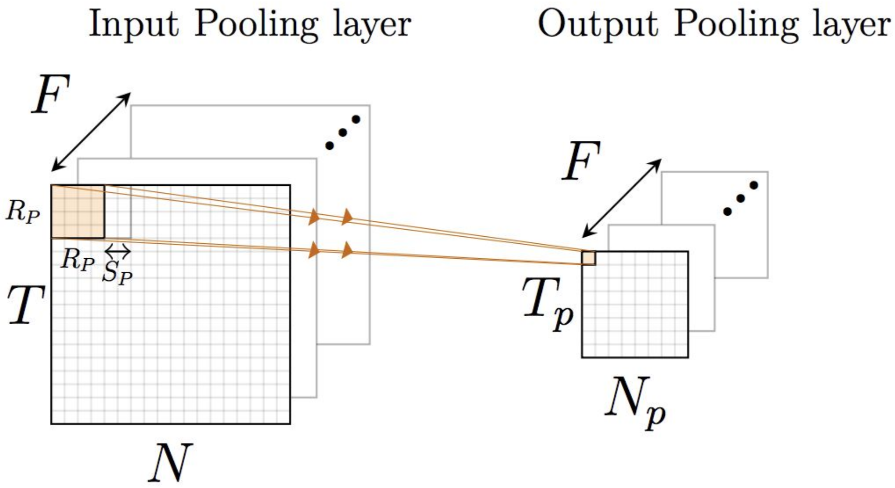




Convolutional filter acts as a **classifier for local features**

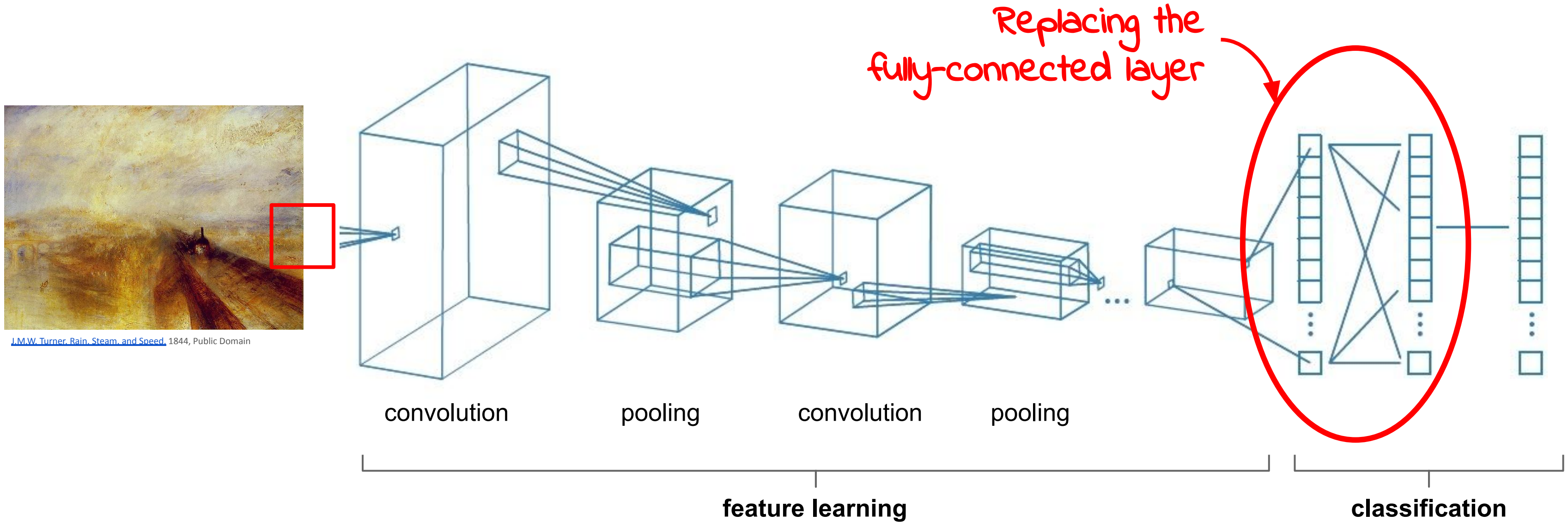
Pooling Layer

- **Purpose**: subsampling
- Information is compressed between the individual convolutional layers via **pooling layers** (dimensionality reduction), as e.g. by simply taking the maximum.
- Pooling layers run through the **feature maps** created by the filters and compress them.



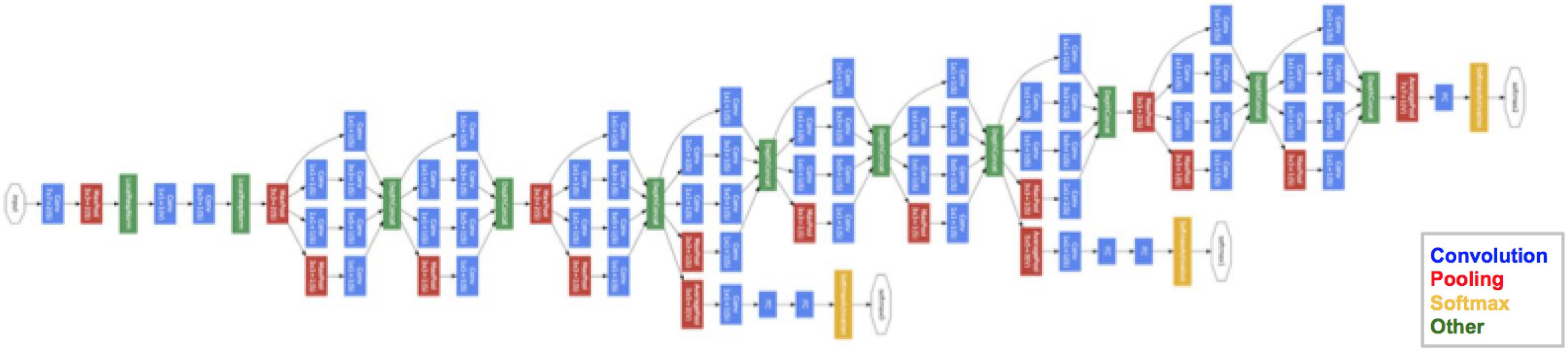
Transfer Learning

- **Transfer Learning (TL)**: learning a new task depends on the previously learned task
- **TL for Deep Learning**: off-the-shelf pre-trained models as feature extractors
    - Fine-tuning off-the-shelf pre-trained models via supervised domain adaption



Deep Learning

- Most **deep learning architectures** combine and recombine a limited set of architectural primitives
    - Fully connected layers
    - Convolutional layers
    - Pooling Layers
    - Recurrent neural network layers
    - Long Short-Term Memory Cells



From Classification to Generation

Style Adaption

Leon A. Gatys, Alexander S. Ecker, Matthias Bethge: [A Neural Algorithm of Artistic Style](). CoRR abs/1508.06576 (2015)

C

D

https://junyanz.github.io/CycleGAN/images/teaser_high_res.jpg

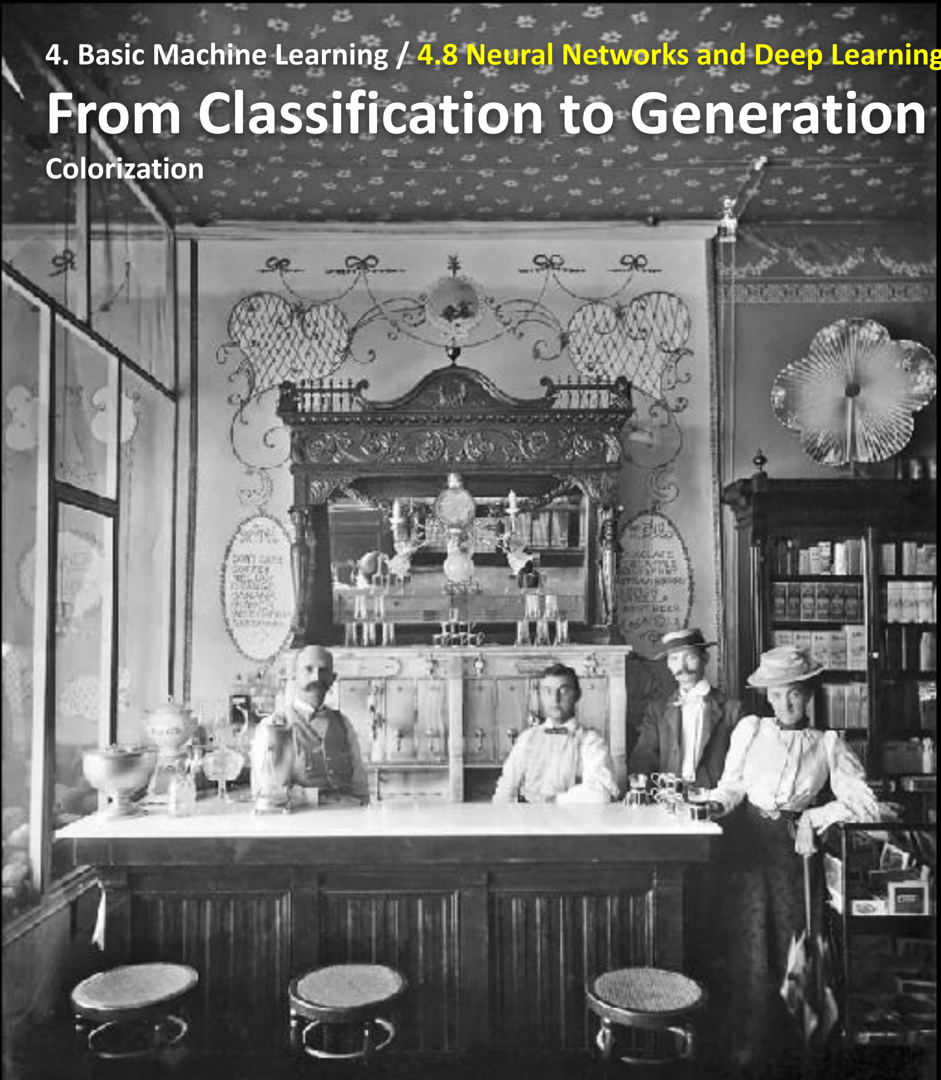
From Classification to Generation

Style Adaption

Monet → Photograph

From Classification to Generation

Colorization



https://github.com/jantic/DeOldify

From Classification to Generation

Cross Domain Transfer

https://junyanz.github.io/CycleGAN/

From Classification to Generation

Super Resolution

bicubic (21.59dB/0.6423)	SRResNet (23.53dB/0.7832)	SRGAN (21.15dB/0.6868)	original

https://arxiv.org/pdf/1609.04802.pdf

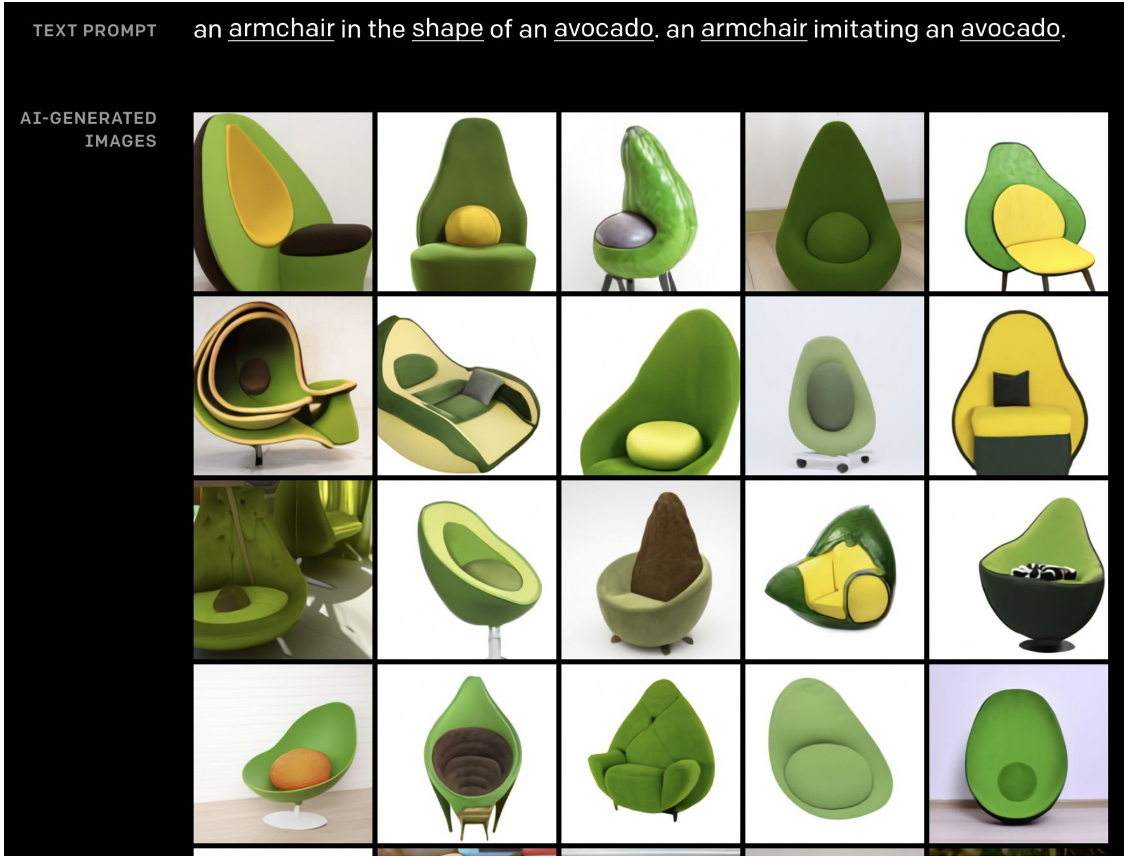
From Classification to Generation

Image Completion

https://openai.com/blog/image-gpt/

From Classification to Generation

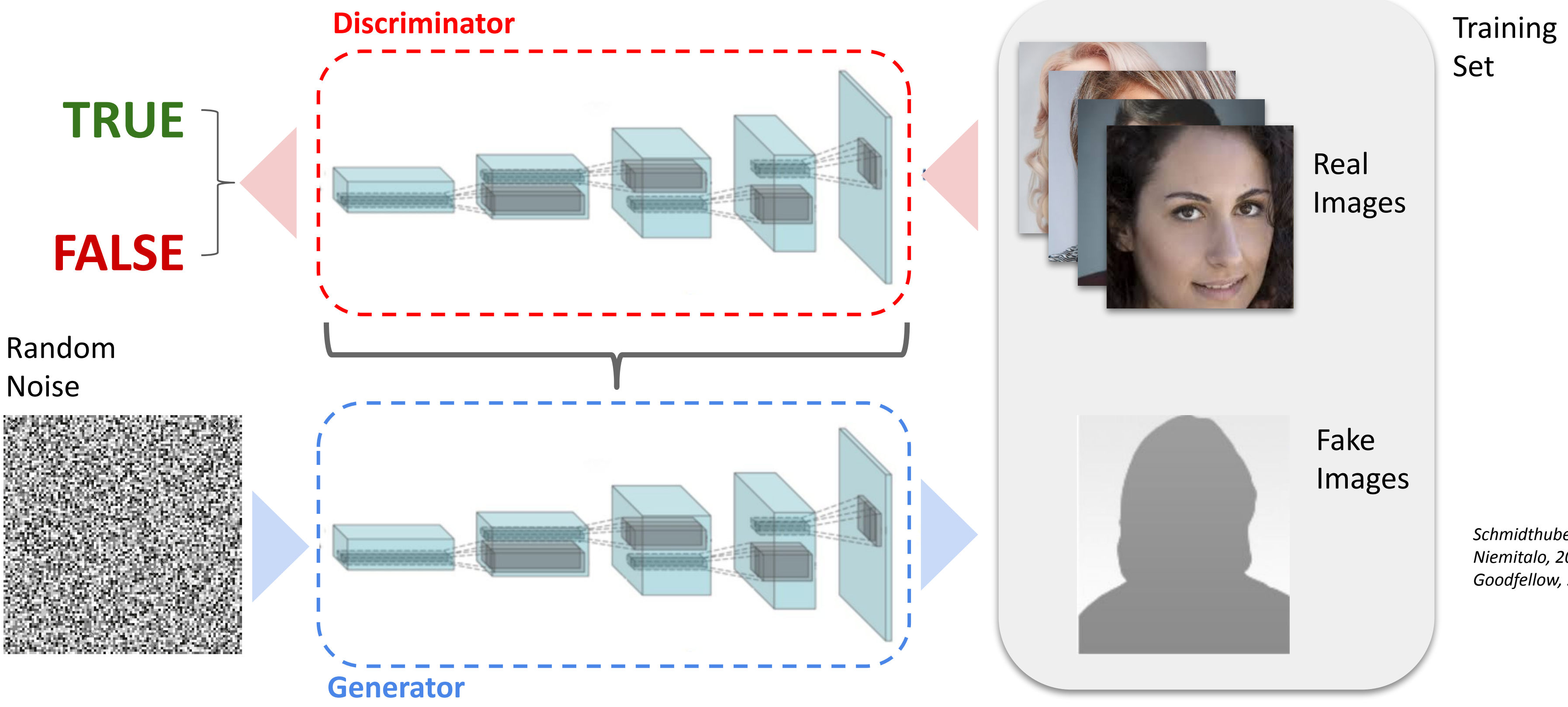
Text-to-Image



OpenAI, DALL·E: Creating Images from Text (2021)

Comparative Learning

Generative Adversarial Networks (GANs)



Schmidthuber, 1992
Niemitalo, 2010
Goodfellow, 2014

https://towardsdatascience.com/generative-adversarial-networks-explained-34472718707a

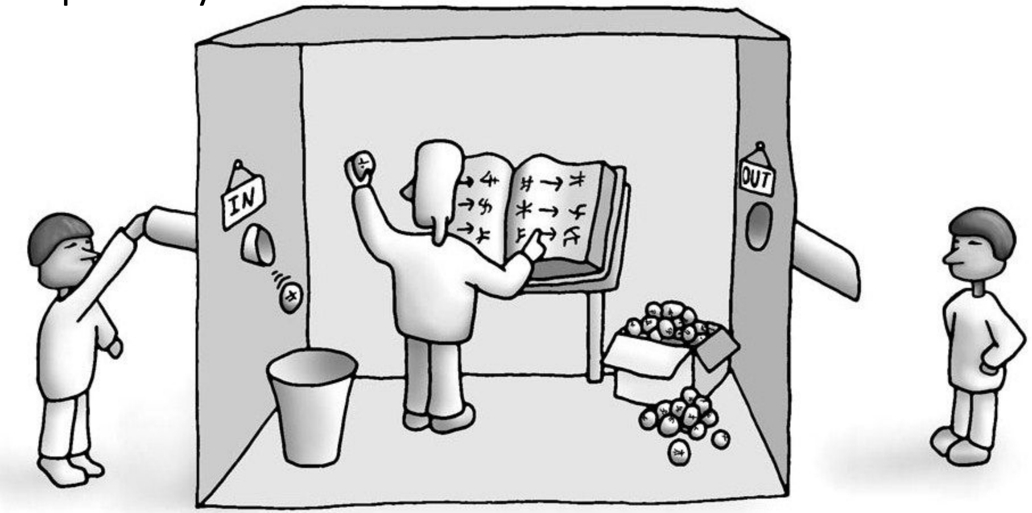
The Clever Hans Effect

or Why we shouldn't always trust ML

https://commons.wikimedia.org/wiki/File:Osten_und_Hans.jpg

The Chinese Room Problem

- Suppose you are placed in a room with a **book of symbols and instructions**.
- When a symbol appears, the book tells you what symbols to produce.
- To any outside observer, the room is able to perfectly answer questions in Chinese, but. . .
  - Does the room know Chinese?
  - Do you know Chinese?
  - Does the book know Chinese?
- The same dilemma occurs
  when we talk about machine learning...



Neural Networks Notebook

Neural Networks at work

12 - ISE2021 -Neural Networks.ipynb

File Edit View Insert Runtime Tools Help Last edited on 28 June

Comment Share

+ Code + Text Connect Editing

Basic Machine Learning - Basic Neural Network Applications

This is the colab notebook example for *lecture 12: Basic Machine Learning 3, chapter 4.8 Neural Networks and Deep Learning*, of AIFB/KIT Lecture "Information Service Engineering" Summer Semester 2021.

In this colab notebook you will learn how to make use of the [keras] library as well as the [SciKit Learn] library for applying machine learning algorithms, in particular for the **multilayer neural networks**, and [mathplotlib] to draw diagrams.

Please make a copy of this notebook to try out your own adaptions via "File -> Save Copy in Drive".

Please also don't forget to turn on GPU support for this notebook via "Edit -> Notebook Settings".

In the last lecture, we were working on an example of decision trees machine learning applied on weather data. We were trying to **predict the weather**, i.e. will it be raining or not, for the next day from today's weather observations.

By then, we achieved the best results for weather prediction with *Random Forests* with an **F1-score of 64.45%**. Today, we want to find out, by how far neural networks are able to challenge this result.

```
# SciKit Learn helper functions
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score,confusion_matrix

# for data analysis and manipulation
import pandas as pd

# Keras neural network model
from keras.models import Sequential
from keras.layers import Dense

# To plot pretty figures
%matplotlib inline
import matplotlib
import matplotlib.pyplot as plt
import seaborn as sns
```

Neural Networks Notebook

Distributional Semantics

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

- A word's meaning is given by the words that frequently appear close-by.
- When a word w appears in a text, its **context** is the set of words that appear nearby (within a fixed-size window).
- Use the different contexts of w to build up a representation of w.

Though quite agile on land, **capybaras** are equally at home in the water.
A giant cavy rodent native to South America, the **capybara** actually is the largest living rodent.

These context words will represent "capybara"

John Rupert Firth
(1890-1960)

J.R. Firth (1957) *A synopsis of linguistic theory, Studies in linguistic analysis*, Blackwell, Oxford

Word Vectors

- We will build a **dense vector** for each word, so that it is similar to vectors of words that appear in similar contexts.

$$\text{capybara} = \begin{pmatrix} 0.286 \\ 0.792 \\ -0.177 \\ -0.107 \\ 0.109 \\ -0.542 \\ 0.349 \\ 0.271 \end{pmatrix}$$

- **Word vectors** are a distributed representation. They are also referred to as **word embeddings** or word representations.

Word2Vec

Overview

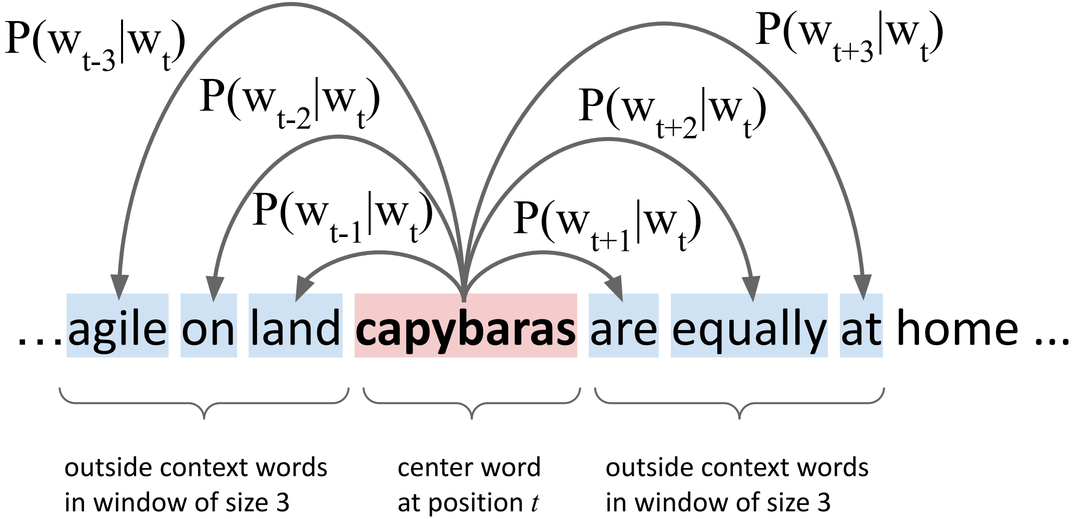
- **Word2vec** (Mikolov et al. 2013) is a framework for learning word vectors.
- Operating Principle:
  - We need to have a large corpus of text.
  - Every word in a fixed vocabulary is represented by a vector.
  - Go through **each position t** in the text, which has a **center word c** and **context ("outside") words o**.
  - Use the **similarity** of the word vectors for c and o to **calculate the probability of o given c** (or vice versa).
  - Keep adjusting the word vectors to maximize this probability.

Mikolov, Tomas; et al. (2013). "*Efficient Estimation of Word Representations in Vector Space*". arXiv:1301.3781

Word2Vec

Overview

- Example windows and process for computing $P(w_{t+j}|w_t)$

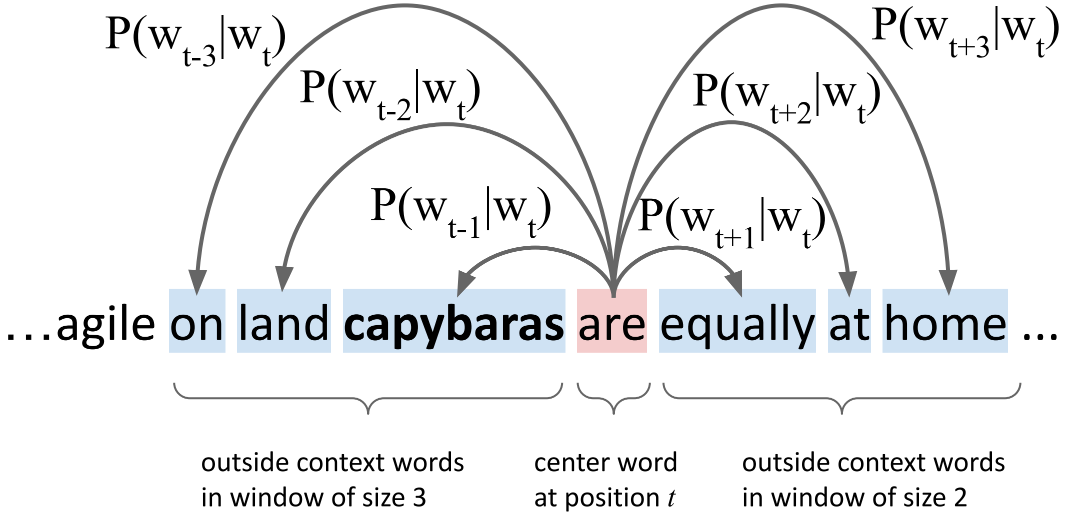


Mikolov, Tomas; et al. (2013). "*Efficient Estimation of Word Representations in Vector Space*". arXiv:1301.3781

Word2Vec

Overview

- Example windows and process for computing $P(w_{t+j}|w_t)$



Mikolov, Tomas; et al. (2013). "*Efficient Estimation of Word Representations in Vector Space*". arXiv:1301.3781

Word2Vec

Objective Function

- For each position $t = 1, \ldots, T$, predict context words within a window of fixed size m, given center word w_t

$$\text{Likelihood} = L(\theta) = \prod_{t=1}^{T} \prod_{\substack{-j \leq m \leq j \\ j \neq 0}} P(w_{t+j} | w_t; \theta)$$

θ is all variables to be optimized

also referred to as cost or loss function

- The **objective function** $J(\theta)$ is the (average) negative log likelihood:

$$J(\theta) = -\frac{1}{T} \log L(\theta) = -\frac{1}{T} \sum_{t=1}^{T} \sum_{\substack{-j \leq m \leq j \\ j \neq 0}} \log P(w_{t+j} | w_t; \theta)$$

- Minimizing objective function == Maximizing predictive accuracy

Word2Vec

Objective Function

- Minimizing the objective function $J(\theta) = -\frac{1}{T}\sum_{t=1}^{T}\sum_{\substack{-j\leq m\leq j\\ j\neq 0}} \log P(w_{t+j}|w_t;\theta)$
- How to calculate $P(w_{t+j}|w_t;\boldsymbol{\theta})$?
- Idea: Use two vectors per word:
  - v_w when w is a center word
  - u_w when w is a context word
- The probability that a **context word** $\boldsymbol{o}$ co-occurs for a **center word** $\boldsymbol{c}$:

$$P(o|c) = \frac{\exp(u_o^T v_c)}{\sum_{w\in V}\exp u_w^T v_c}$$

Mikolov, Tomas; et al. (2013). "*Efficient Estimation of Word Representations in Vector Space*". arXiv:1301.3781

Word2Vec

Objective Function

exponentiation to make everything positive

Dot product compares similarity larger dot product = larger probability

$$u^T v = u \cdot v = \sum_{i=1}^{n} u_i v_i$$

$$P(o|c) = \frac{\exp(u_o^T v_c)}{\sum_{w \in V} \exp u_w^T v_c}$$

normalize over the entire vocabulary V

- This is a **softmax function** $\mathbb{R}^n \to (0,1)^n$

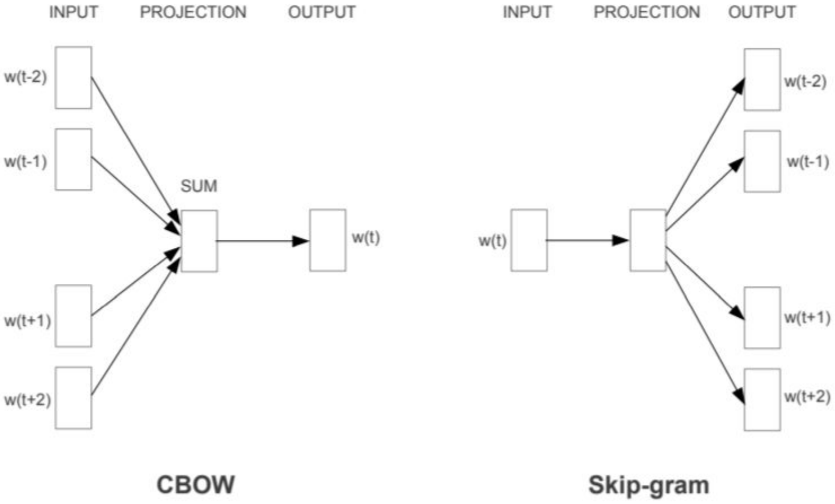
$$\mathrm{softmax}(x_i) = \frac{\exp(x_i)}{\sum_{j=1}^{n} \exp(x_j)} = p_i$$

- The softmax function maps arbitrary values x_i to a probability distribution p_i
  - "**max**": amplifies probability of the largest x_i
  - "**soft**": also assigns (some) probability to smaller x_i

Word2Vec

Variants

- Word2Vec maximizes the objective function by putting similar words nearby in vector space.
- Two model variants:
    - **Skip-gram (SG):**
      Predict (sequence independent) context words given the center word.
    - **Continuous Bag of Words (CBOW):**
      Predict center word from (bag of) context words.

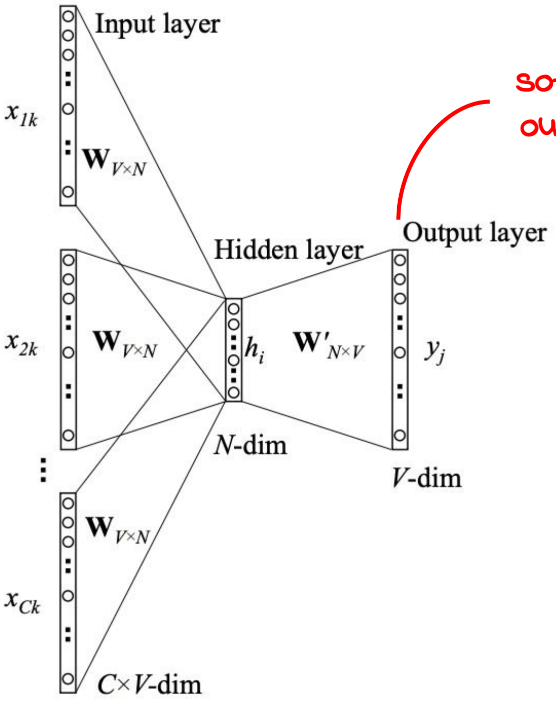


Mikolov, Tomas; et al. (2013). "*Efficient Estimation of Word Representations in Vector Space*". arXiv:1301.3781

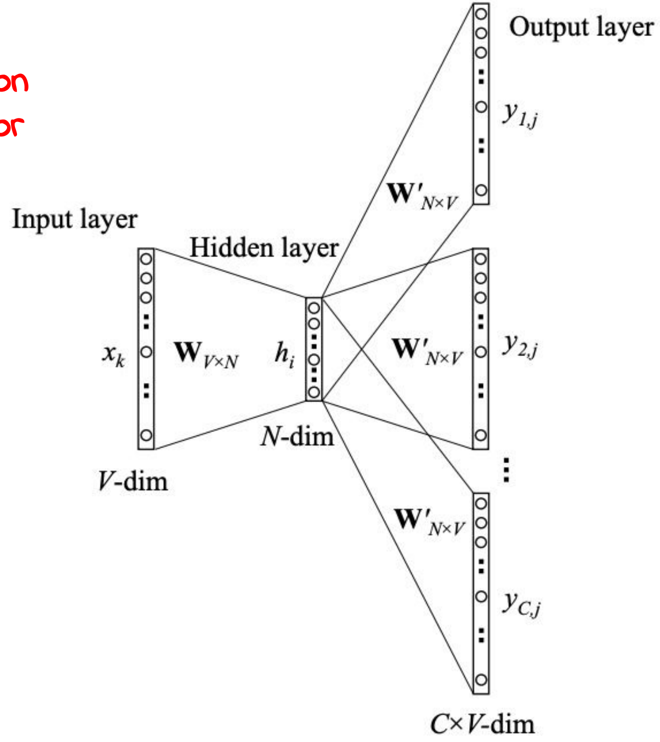
Word2Vec

Skip-gram & CBOW

softmax is on output vector



CBOW



Skip-gram

Rong, X.(2014). *word2vec parameter learning explained*. CoRR, abs/1411.2738

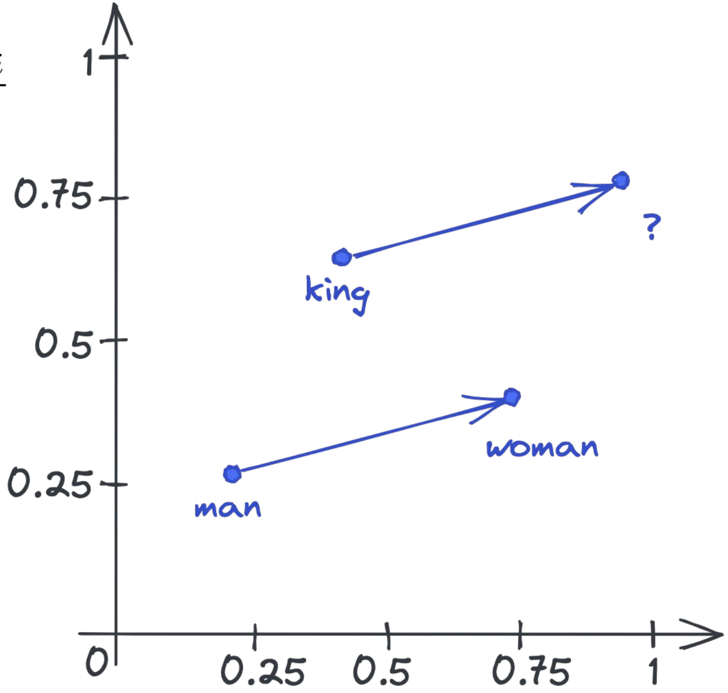
How to Evaluate?

Intrinsic Evaluation of Word Vectors

- Word Vector Analogies

$$a{:}b :: c{:}d \longrightarrow d = \arg\max_i \frac{(x_b - x_a + x_c)^T x_i}{||x_b - x_a + x_c||}$$

man : woman = king : ?



- Evaluate word vectors by how well their cosine distance after addition captures intuitive semantic and syntactic analogy questions.
- Discarding the input words from the search!
- Problem: What if the information is there but not linear?

Word Embeddings Notebook

Word2Vec at work

12 - ISE2021 -Word Embeddings.ipynb

File Edit View Insert Runtime Tools Help Last edited on 18 June

+ Code + Text

Basic Machine Learning - Word Embeddings

This is the colab notebook example for lecture 12: Basic Machine Learning 3, chapter 4.8 Word Embeddings, of AIFB/KIT Lecture "Information Service Engineering" Summer Semester 2021.

In this colab notebook you will learn how to make use of the SciKit Learn and the gensim library for applying machine learning algorithms, in particular for the **word2vec** (and other) **word embeddings**, and mathplotlib to draw diagrams.

Please make a copy of this notebook to try out your own adaptions via "File -> Save Copy in Drive"

```
#First loading all necessary libraries
%matplotlib inline
import urllib.request
import gensim
import gensim.downloader as api

# to visualize word vectors
from sklearn.manifold import TSNE
import matplotlib.pyplot as plt
```

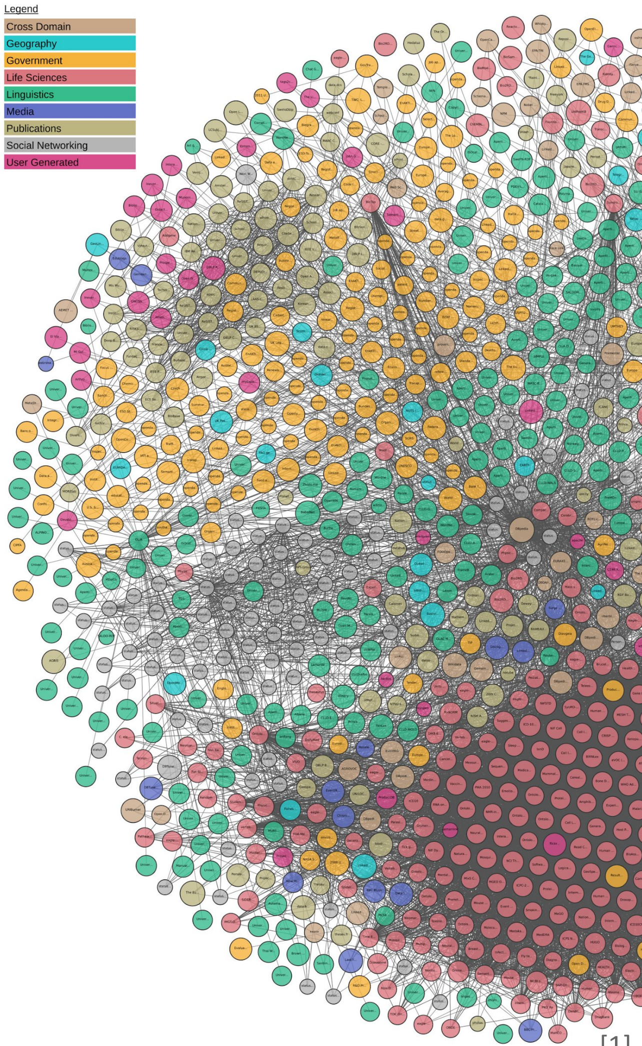
```
#helper function to draw a TSNE diagram of a word vector space
def plot_word_embeddings(model, search_list):
    words = []
    for term in search_list:
        words += [w[0] for w in model.most_similar([term], topn=5)]
    words += search_list

    vectors = model[words]

    tsne = TSNE(n_components=2, random_state=0, n_iter=10000, perplexity=7)
    T = tsne.fit_transform(vectors)
```

Word Embeddings Notebook

Semantic Similarity

From Words to Entities

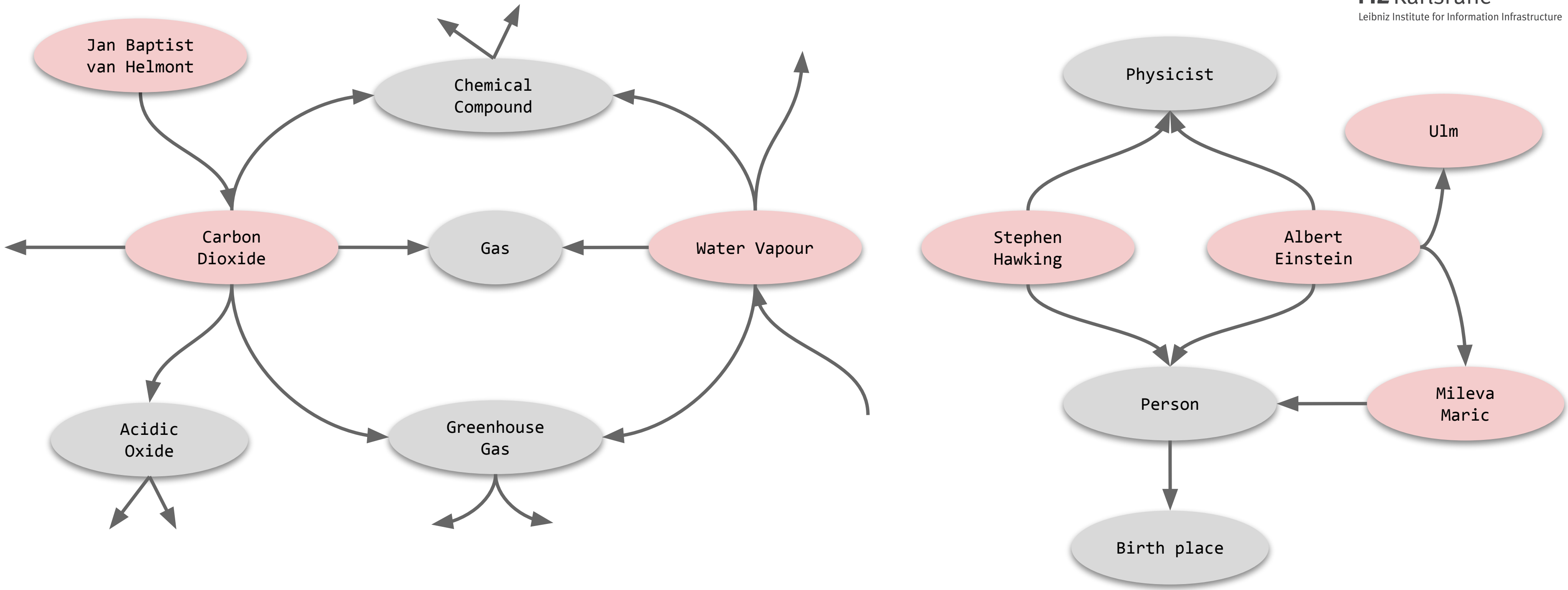
- For **Word Embeddings**, words with similar meanings are mapped to close vectors in a vector space.
- How can we map this concept to Knowledge Graphs?
- **When are two nodes (entities) semantically similar?**
  - If they can be described by the same/similar facts, as e.g.
    - Carbon Dioxide is a Greenhouse Gas and water vapour is a greenhouse gas.
    - Albert Einstein is a Physicist and Stephen Hawking is a Physicist.
    - **Is Stephen Hawking more similar to Albert Einstein or to Carbon Dioxide?**



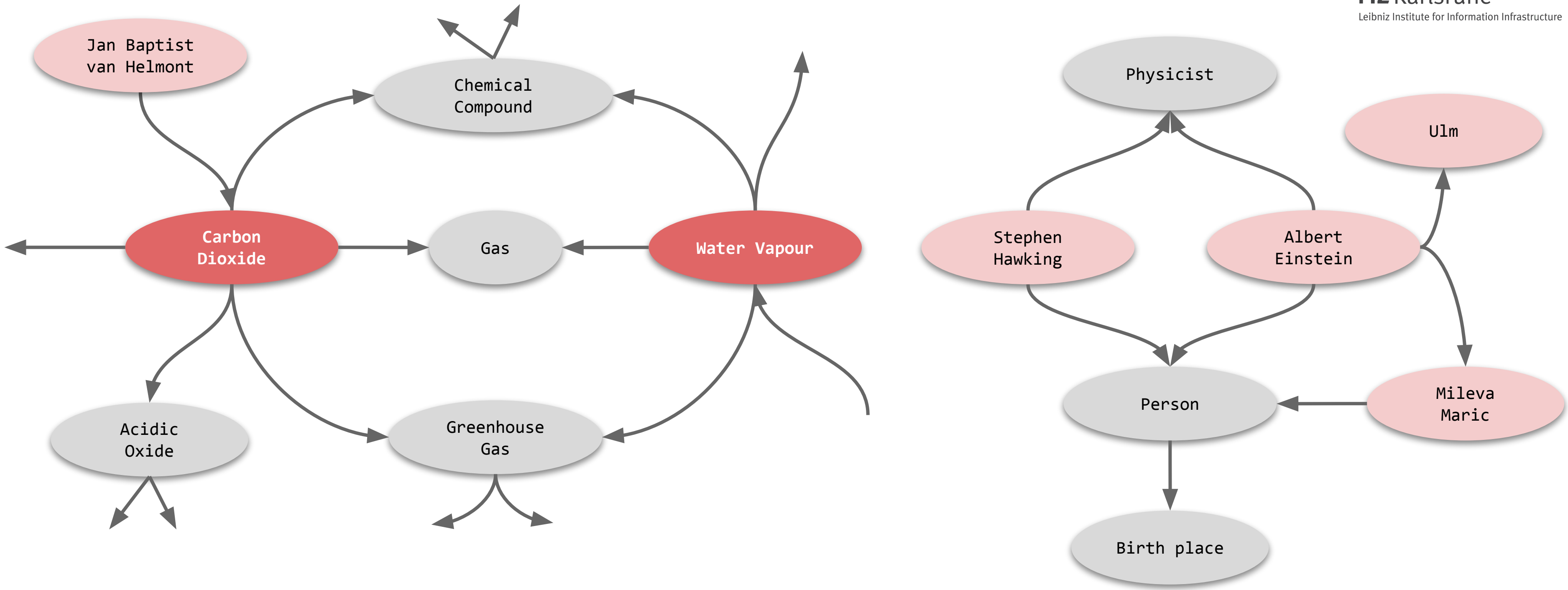
The Linked Open Data Cloud from lod-cloud.net

[1]

Semantic Similarity

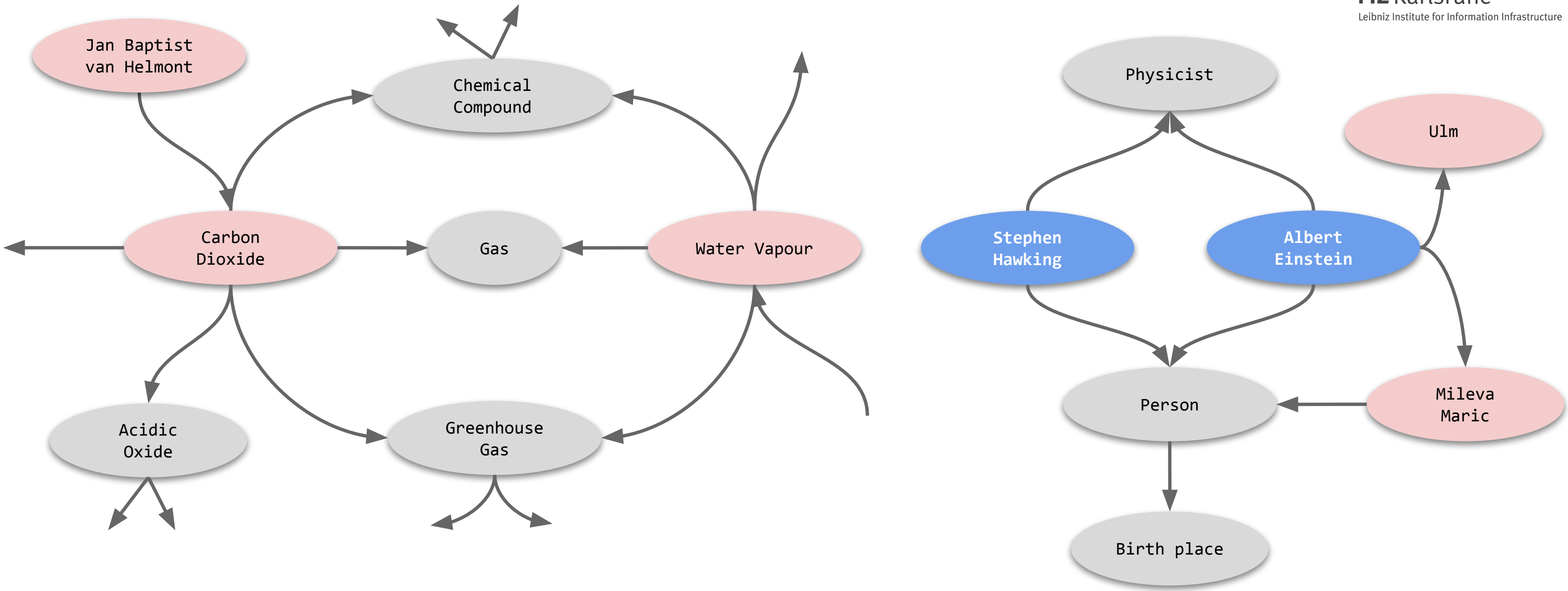


Semantic Similarity



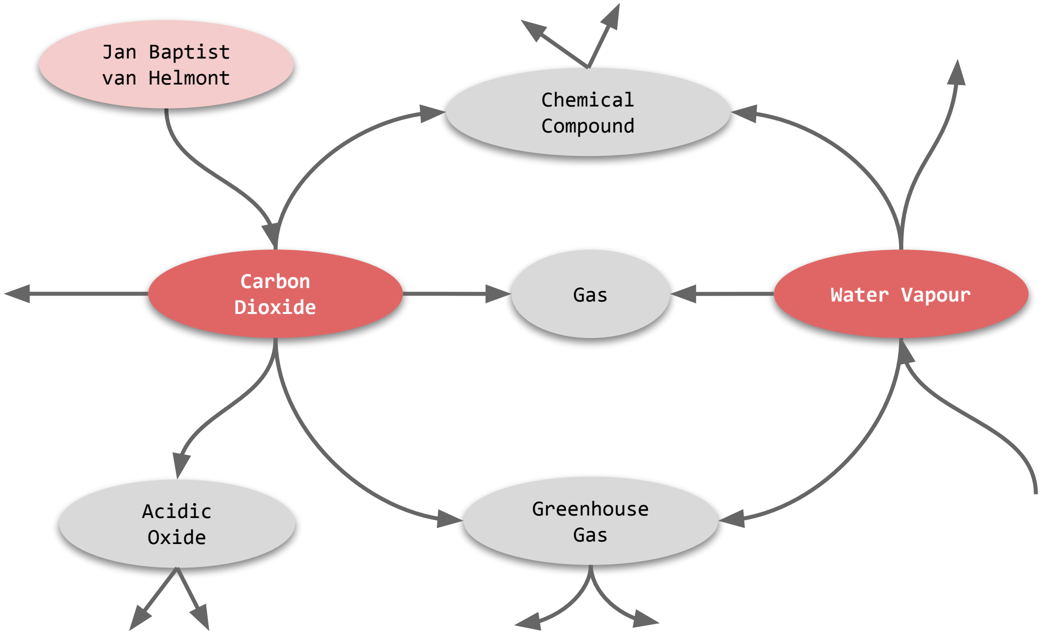
- **Carbon Dioxide** and **Water Vapour** share similar (structural) context in the graph.

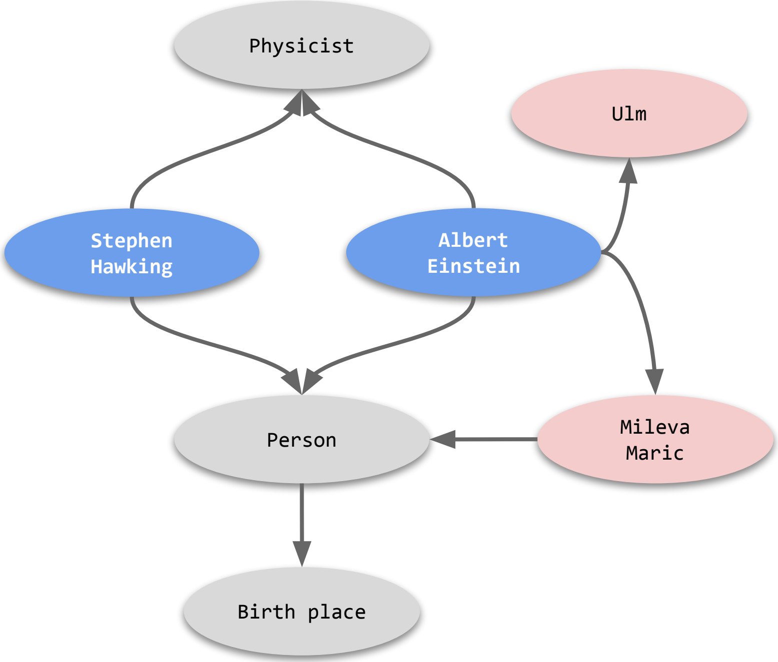
Semantic Similarity



- **Stephen Hawking** and **Albert Einstein** share similar (structural) context in the graph.

Semantic Similarity

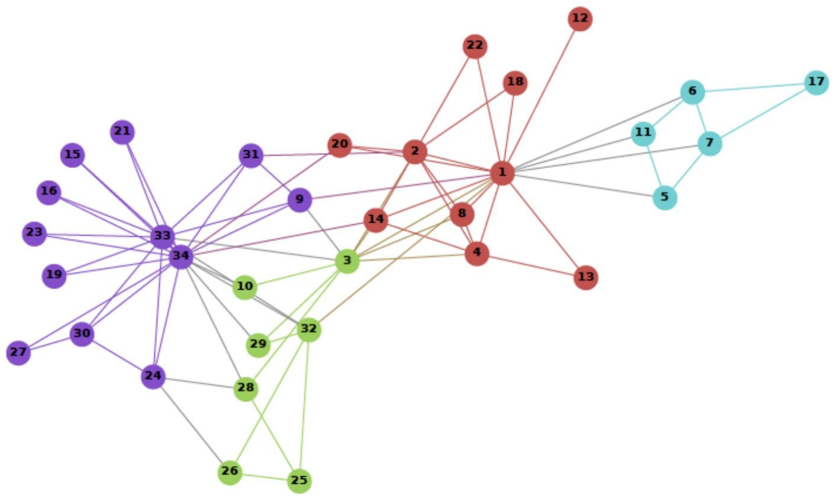




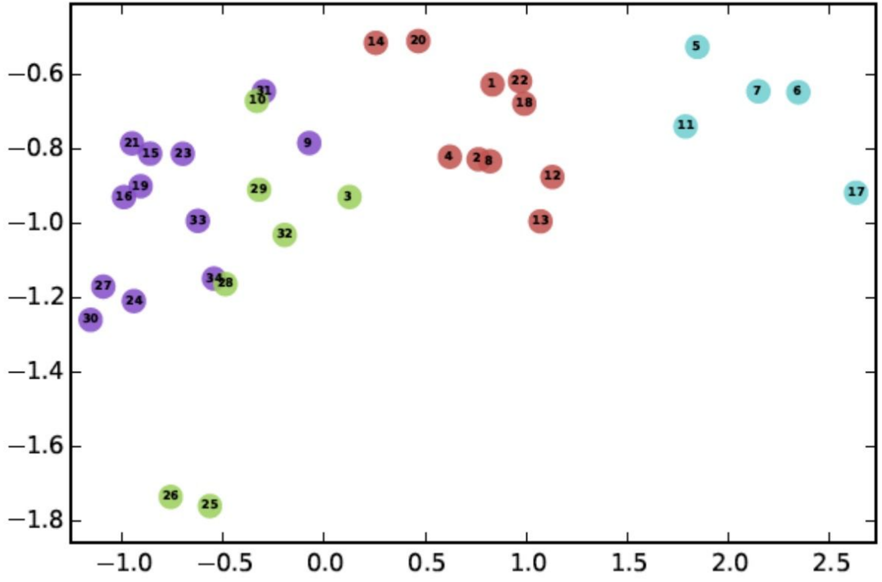
- "You shall know a node by the company it keeps",
  i.e. similar nodes can be identified by having the same/similar neighborhood (context)
  ▸ **Distributional Semantics**

Graph Embeddings

(Knowledge) Graph Representation



Vector Space Representation



Idea: Find embedding of nodes in a low-dimensional vector space so that **"similar" nodes in the graph** have **vector embeddings that are close together**.

Knowledge Graphs are more than just plain Graphs

Karlsruher Institut für Technologie

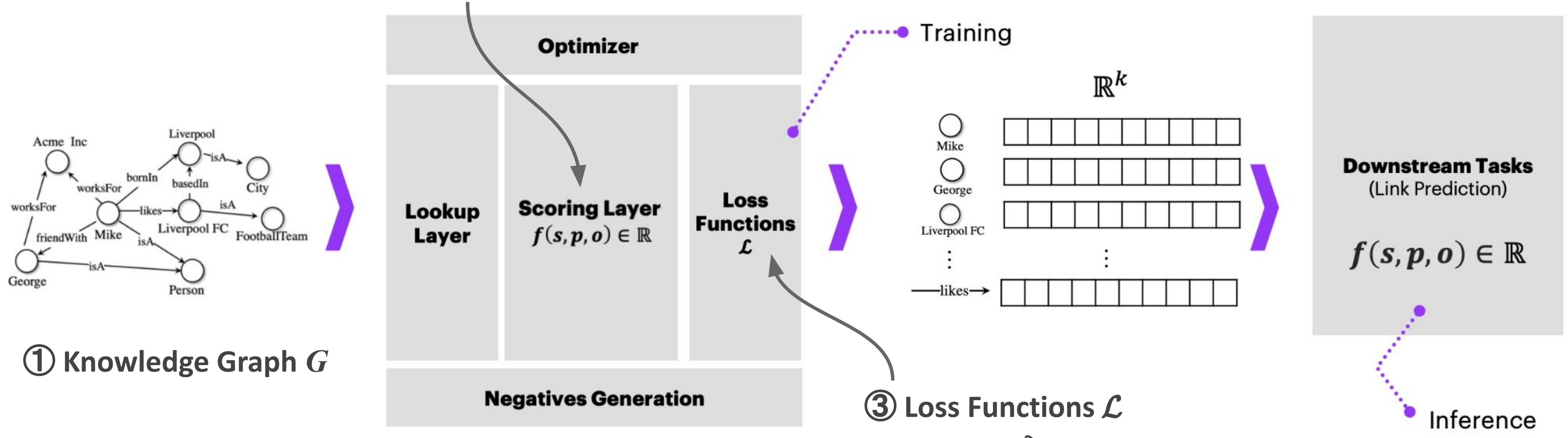
FIZ Karlsruhe
Leibniz Institute for Information Infrastructure

- Besides entities (nodes), we also want to represent
  - Properties and property relations
    - Hierarchies and inverse properties
    - Symmetry and antisymmetry
    - Reflexivity and irreflexivity
    - Functionality and inverse functionality
    - Transitivity
  - Literals
    - Multimodality (text, numbers, images, etc.)
    - Datatype semantics

Knowledge Graph Embed… …truction Kit

High score = high chances for the fact (s,p,o) to be true

② Scoring Layer $f(s,p,o) \in \mathbb{R}$



④ Negatives Generation

Create "corrupted" versions of existing triples as negative triples

Pays a penalty if score of positive triple less than score of synthetic negative

Costabello, L. et al, *ECAI-20 Tutorial: Knowledge Graph Embeddings: From Theory to Practice*, 2020

Knowledge Graph Embeddings

Many ways to generate Knowledge Graph Embeddings:

- **Translational Methods**: TransE, TransH, TransR, TransEdge, …
- **Semantic Matching**: RESCAL, DistMult, HolE, ComplEx
- **Graph Convolutional Networks**: R-GCN, TransGCN, ConvE, ConvR, ConvKB
- **RelationPaths**: PTransE, DeepWalk, RDF2Vec
- ...

Knowledge Graph Completion - Link Prediction

	Task	Example	Result
Link Prediction	Triple Classification	`(JosephFourier, occupation, physicist)?`	(yes, 95%)
	Tail Prediction	`(JosephFourier, occupation, ?)`	(1, physicist, 0.95), (2, chemist, 0.93) ...
	Head Prediction	`(?, occupation, physicist)`	(1, AlbertEinstein, 0.91) (2, StephenHawking, 0.90)
	Relation Prediction	`(JosephFourier, ?, physicist)`	(1, occupation, 0.95)
	Entity Classification (Type Prediction)	`(JosephFourier, isA, ?)`	(1, Person, 0.99) (2, Human, 0.99),...

Information Service Engineering

4. Machine Learning

Information Service Engineering
Lecture Overview

1. Information, Natural Language and the Web
2. Natural Language Processing
3. Linked Data Engineering
4. Basic Machine Learning
5. ISE Applications

4. Basic Machine Learning - 3
Bibliography

- S. Marsland, ***Machine Learning, An Algorithmic Perspective***, 2nd. ed., Chapman & Hall / CRC Press, 2015.
    - Chap. 3 (Neural Networks)

  (The book should also be available on the Web as pdf, just keep looking…)

- Word Embeddings:
  Mikolov, Tomas; et al. (2013). ***Efficient Estimation of Word Representations in Vector Space***.
  arXiv:1301.3781

- Knowledge Graph Embeddings:
  Wang et al., ***Knowledge graph embedding: A survey of approaches and applications***. IEEE Transactions on Knowledge and Data Engineering (TKDE), 2017.
  (if you want to deepen your knowledge beyond the scope of the lecture…)

4. Basic Machine Learning - 3
Syllabus Questions

- How does a Biological Neural Network work?
- Explain the McCulloch Pitts Neuron model.
- Explain a Perceptron.
- Why can’t a Perceptron solve the XOR problem?
- What are the limitations of Deep Neural Networks?
- How to Convolutional Neural Networks overcome (some of) the problems/limitations of Deep Neural Networks?
- Explain Word Embeddings.
- Explain Graph Embeddings.
- What is Knowledge Graph Completion?