



# Autonomous Systems: Deep Learning

## 1. Overview: From AI to Deep Learning

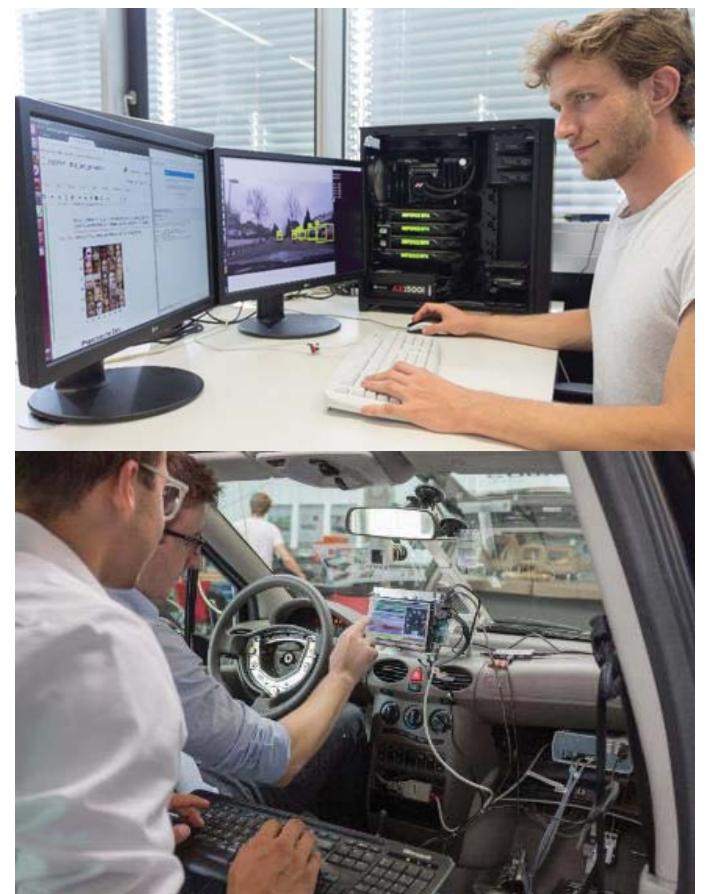


Prof. Dr.-Ing. Nicolaj Stache  
Heilbronn University of Applied Sciences

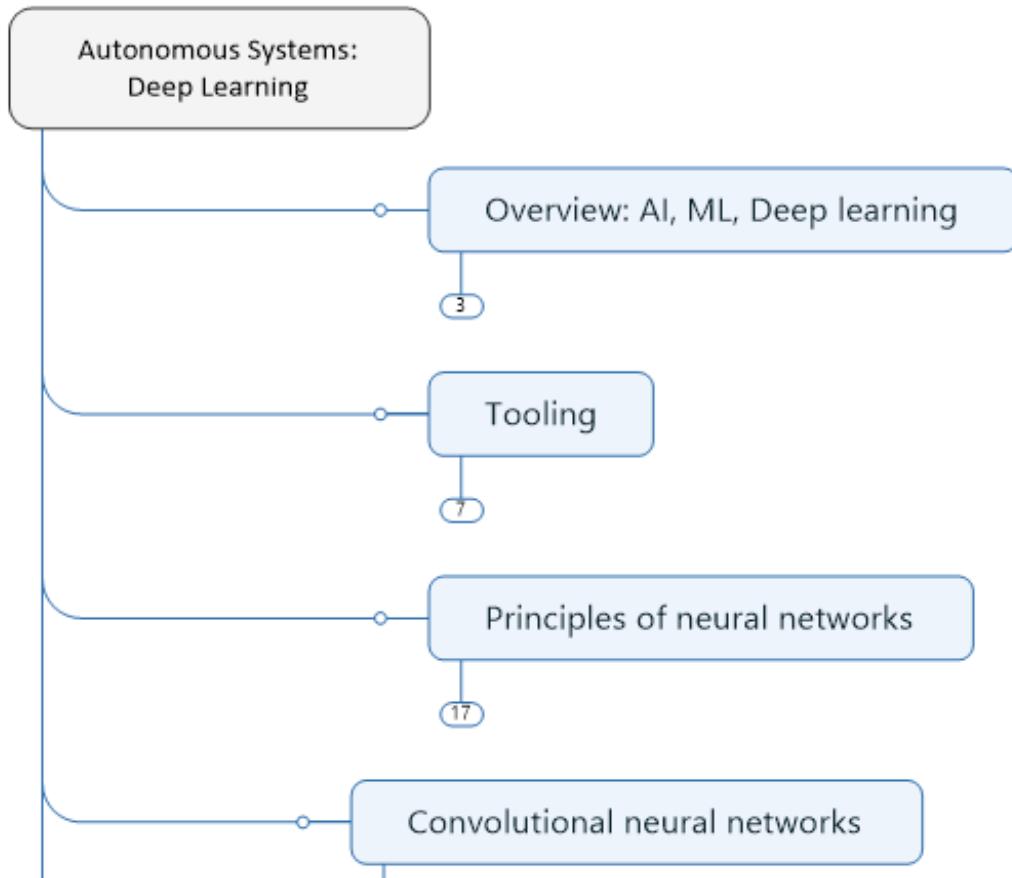
### Introduction Nicolaj Stache



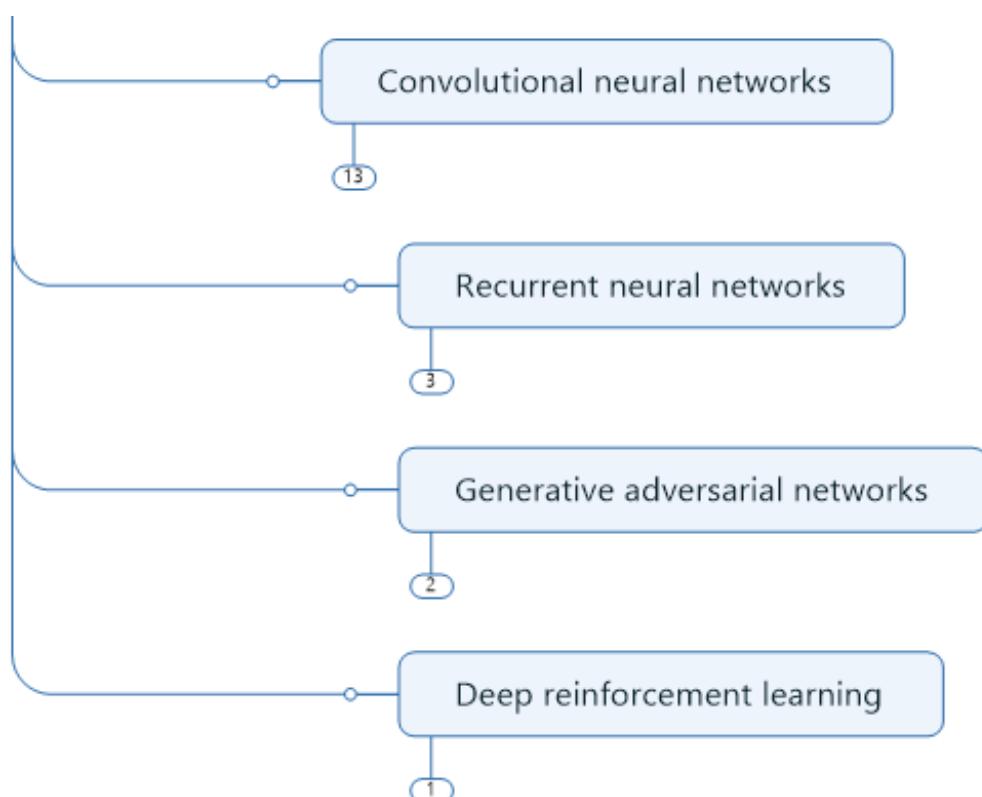
- ▶ Study RWTH Aachen: Electrical Engineering + Information Technology
- ▶ Doctorate RWTH Aachen: Industrial Image Processing
- ▶ 2010 – 2016: Continental
  - ▶ Head of Artificial Intelligence Center
  - ▶ Leader for Sensorics (automated driving)
  - ▶ Team Lead Camera Monitoring
- ▶ 2016: Professor of Measurement and Sensor Technology in the automotive field
- ▶ 2017: Program director of Automotive Systems Engineering, HHN
- ▶ 2018: Co-Founder of Center for Machine Learning (ZML) @ HHN



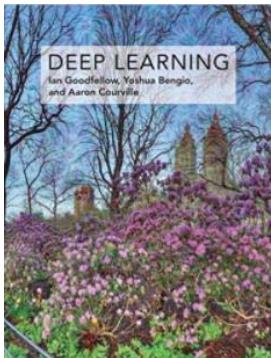
# Overview on the Course



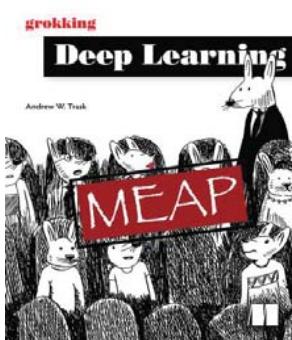
# Overview on the Course



- ▶ Goodfellow et al: Deep Learning, The MIT Press, 2016
- ▶ Trask: Grokking Deep Learning, Manning Publications, 2017
- ▶ Online Books:
  - ▶ <http://neuralnetworksanddeeplearning.com/>
  - ▶ <http://www.deeplearningbook.org/>



Auto.-Sys: Deep Learning



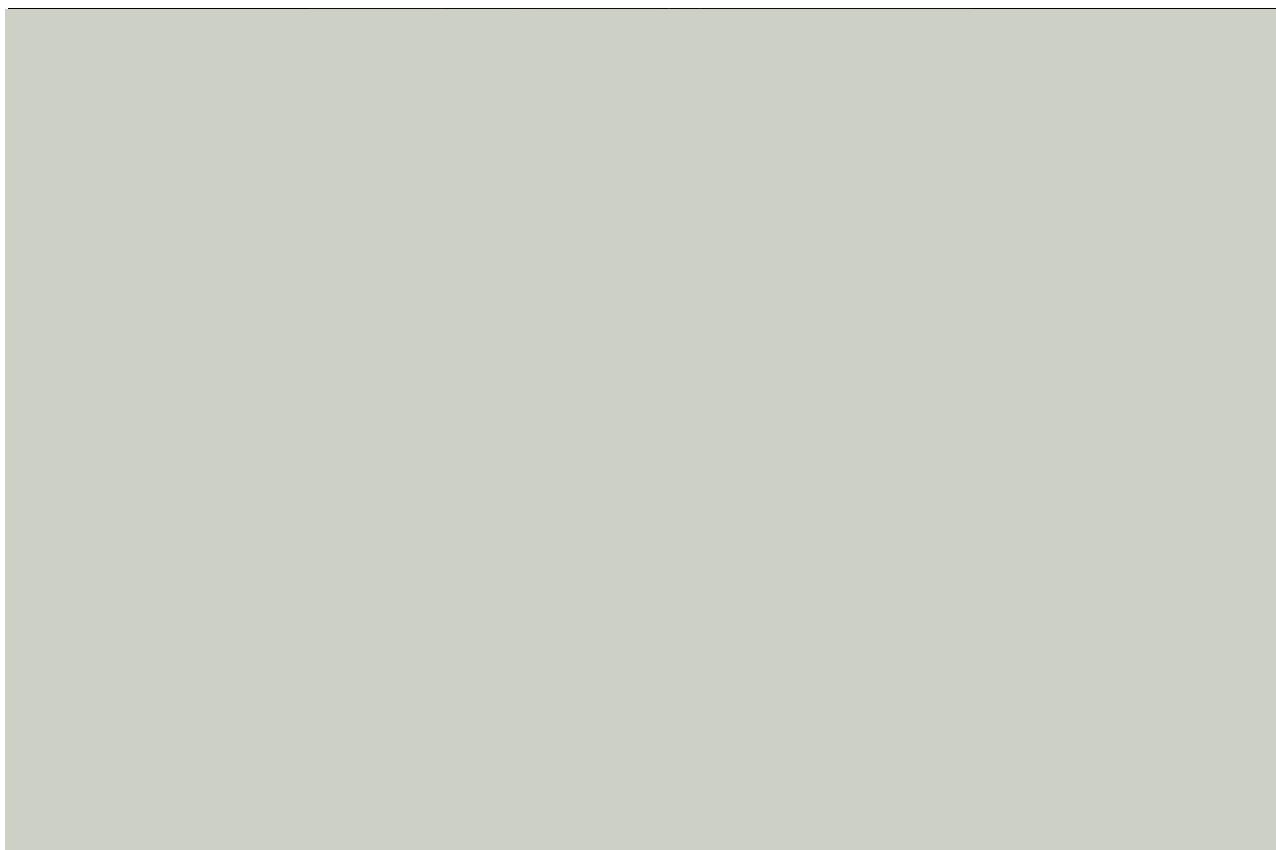
- ▶ Stanford Lectures on CNNs:  
<http://cs231n.github.io/>  
<https://www.youtube.com/playlist?list=PLC1qU-LWwrF64f4QKQT-Vg5Wr4qEE1Zxk>
- ▶ Udacity courses on deep learning:
  - ▶ <https://de.udacity.com/course/deep-learning--ud730>
  - ▶ <https://www.udacity.com/course/deep-learning-nanodegree--nd101>

Prof. Dr. N. Stache

5

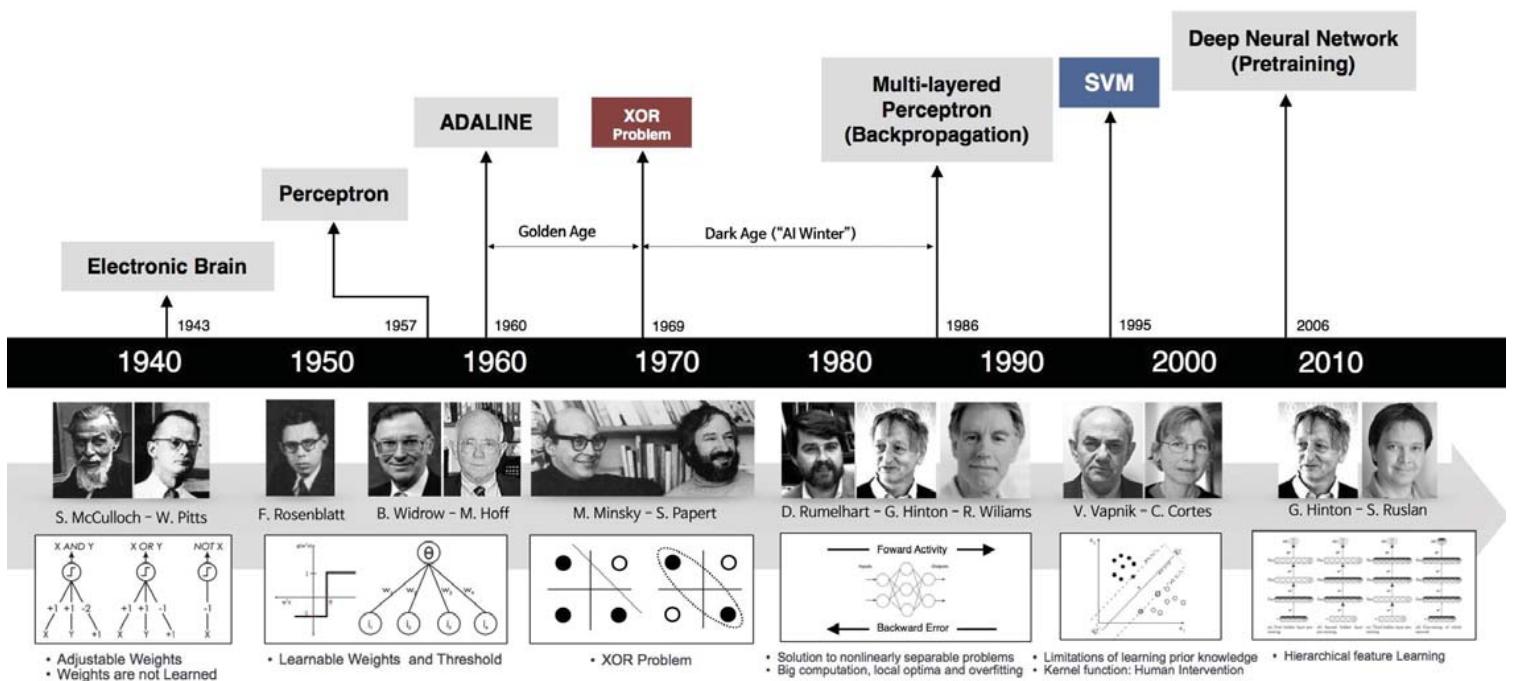
## Learning Objectives

- ▶ You can describe the meaning of...
  - ▶ Artificial Intelligence
  - ▶ Machine Learning
  - ▶ Deep learning
- ▶ You can describe the differences between supervised, unsupervised, reinforcement learning
- ▶ You know parametric and non-parametric models
- ▶ You know where and why is deep learning successful
- ▶ You can give examples of deep learning applications



Source: <https://www.zdf.de/nachrichten/heute-plus/deep-learning-104.html>, 03.03.2018

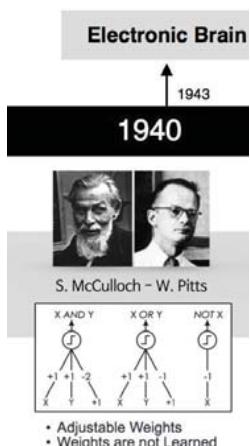
## Historical Overview



Dream of “self-thinking machine”

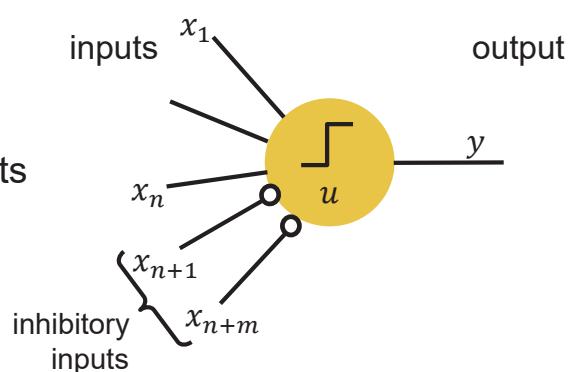
## 1943 - McCulloch & Pitts

- ▶ Re-engineer the principle of brain
- ▶ Introduction of threshold logic unit
- ▶ Realization of logic gates: and, or, not
- ▶ Only binary inputs, inhibitory inputs binary output
- ▶ No trainable weights



Auto.-Sys: Deep Learning

Source:  
[https://beamandrew.github.io/deeplearning/2017/02/23/deep\\_learning\\_101\\_page1.html](https://beamandrew.github.io/deeplearning/2017/02/23/deep_learning_101_page1.html), 21.09.2017



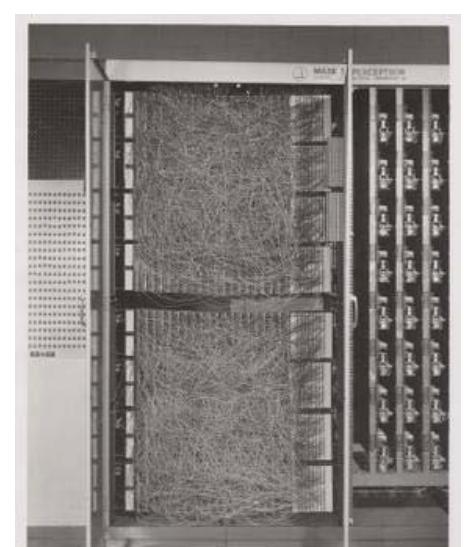
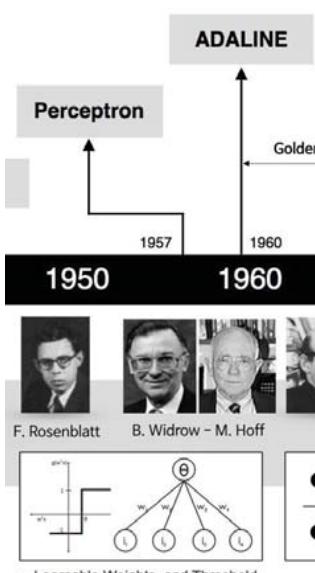
Prof. Dr. N. Stache

9

# Historical Overview 1957 - 1969

“The perceptron is the embryo of an electronic computer that [the Navy] expects will be able to walk, talk, see, write, reproduce itself and be conscious of its existence.”  
Rosenblatt (1957)

- ▶ Weights of the inputs and threshold of the output are learned
- ▶ Binary classification possible
- ▶ Single layer perceptron cannot model XOR gate
- ▶ → AI-Winter

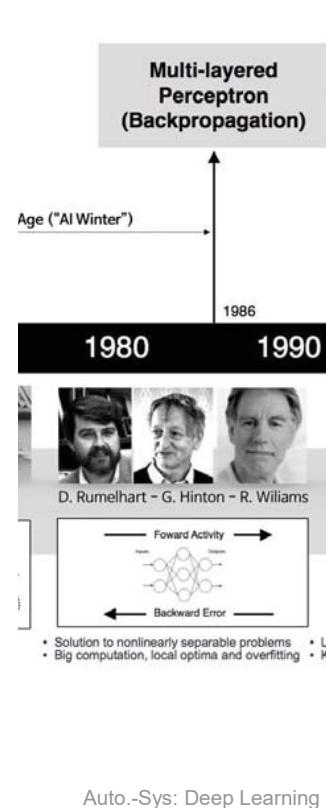


Source:  
[https://beamandrew.github.io/deeplearning/2017/02/23/deep\\_learning\\_101\\_page1.html](https://beamandrew.github.io/deeplearning/2017/02/23/deep_learning_101_page1.html), 21.09.2017;  
[https://upload.wikimedia.org/wikipedia/en/5/52/Mark\\_I\\_perceptron.jpeg](https://upload.wikimedia.org/wikipedia/en/5/52/Mark_I_perceptron.jpeg), 03.10.2018

Auto.-Sys: Deep Learning

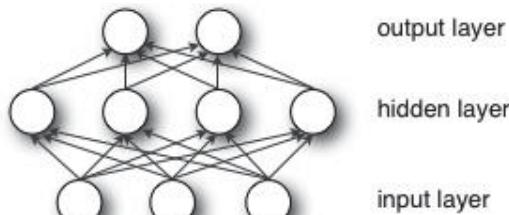
Prof. Dr. N. Stache

10



## Multilayer Perceptron + Backpropagation to learn weights

- ▶ Solution to nonlinear problems
- ▶ Multilayer Perceptron == Feedforward artificial neural network



But:

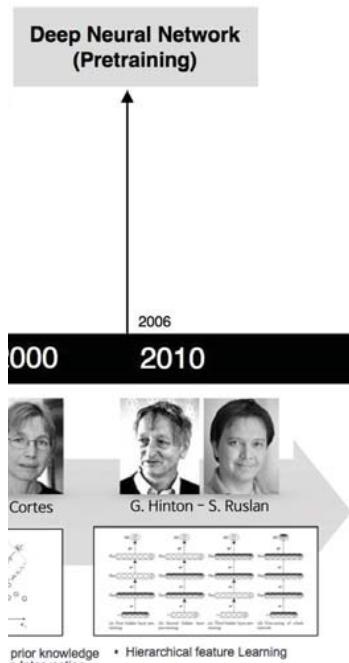
- ▶ Failed to train deep structures
  - ▶ High computational effort
  - ▶ Alternative Approach: Support Vector Machine
- 2<sup>nd</sup> AI Winter

Source:  
<http://deeplearning.net/tutorial/mlp.html>, 03.10.2018  
[https://beamandrew.github.io/deeplearning/2017/02/23/deep\\_learning\\_101\\_pa\\_r1.html](https://beamandrew.github.io/deeplearning/2017/02/23/deep_learning_101_pa_r1.html), 21.09.2017

Prof. Dr. N. Stache

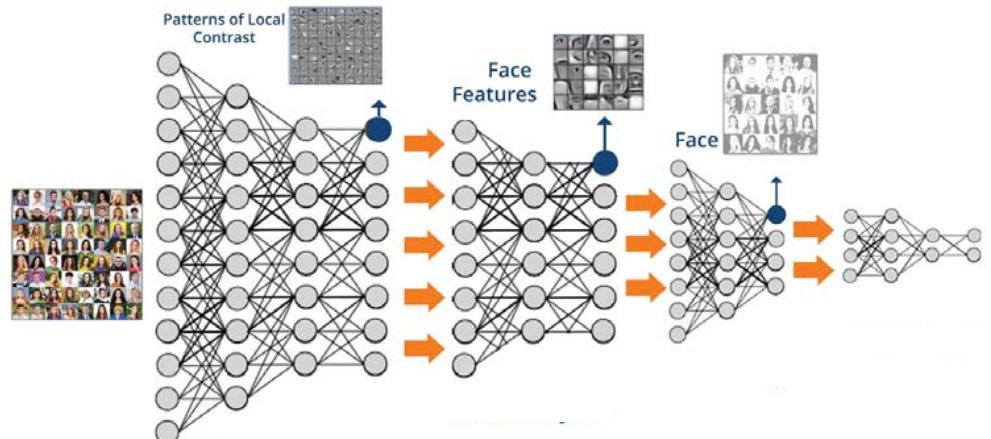
11

# Historical Overview from 2006



## Era of deep learning starts

- ▶ Breakthrough in 2012 with image recognition and language processing



Source:  
<https://cdn.edureka.co/blog/wp-content/uploads/2017/05/Deep-Neural-Network-What-is-Deep-Learning-Edureka.png>, 03.10.2018  
[https://beamandrew.github.io/deeplearning/2017/02/23/deep\\_learning\\_101\\_pa\\_r1.html](https://beamandrew.github.io/deeplearning/2017/02/23/deep_learning_101_pa_r1.html), 21.09.2017

Prof. Dr. N. Stache

12

## Artificial Intelligence

### Definition (SOURCE: MERRIAM WEBSTER)

- **1:** a branch of computer science dealing with the simulation of intelligent behavior in computers
- **2:** the capability of a machine to imitate intelligent human behavior

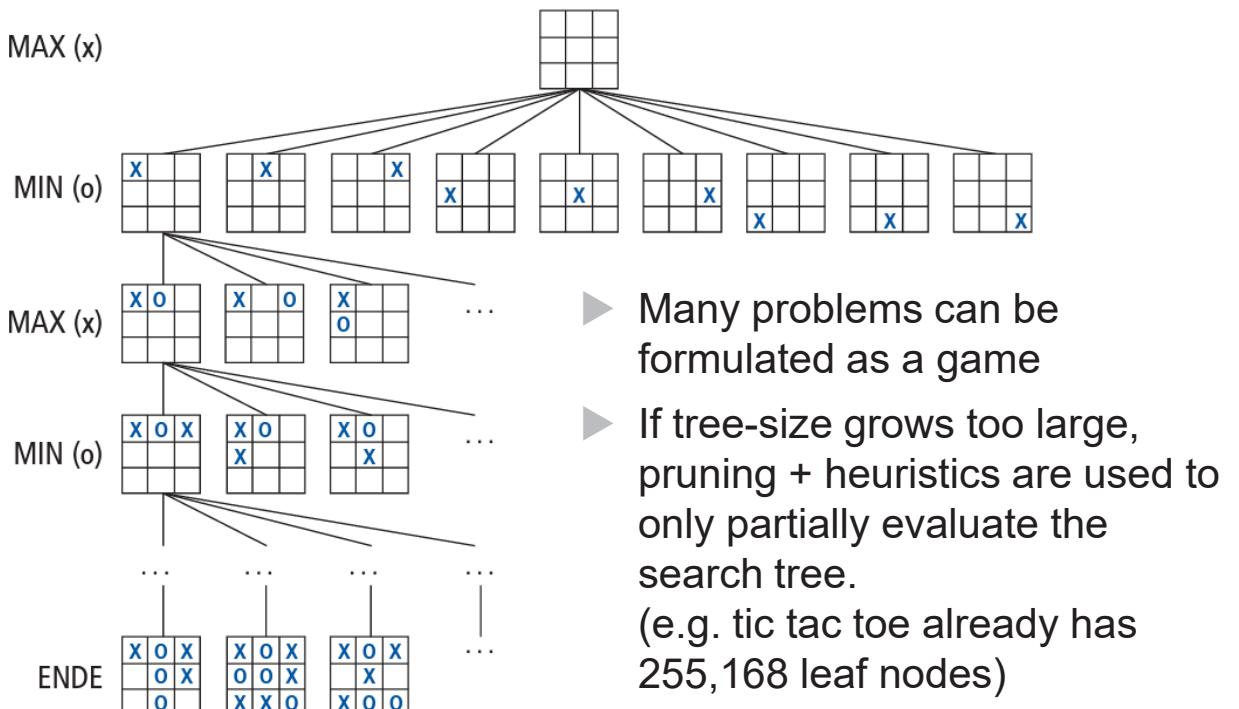


1950 – Turing test:  
Human evaluates natural language conversations ...  
... if evaluator cannot reliably tell the machine from human, the test is passed

## Example of AI – Tic Tac Toe

### Minimax-Tree:

- Player x wins at maximized score, Player y wins at minimized score



## Artificial Intelligence (not completely)

- knowledge input by hand-designed rules
- e.g. knowledge base system, chess-playing system

## Machine Learning

- system acquires own knowledge by extracting patterns from hand-defined features, i.e. it learns from data

## Representation Learning

## Deep Learning

# Overview on Machine Learning

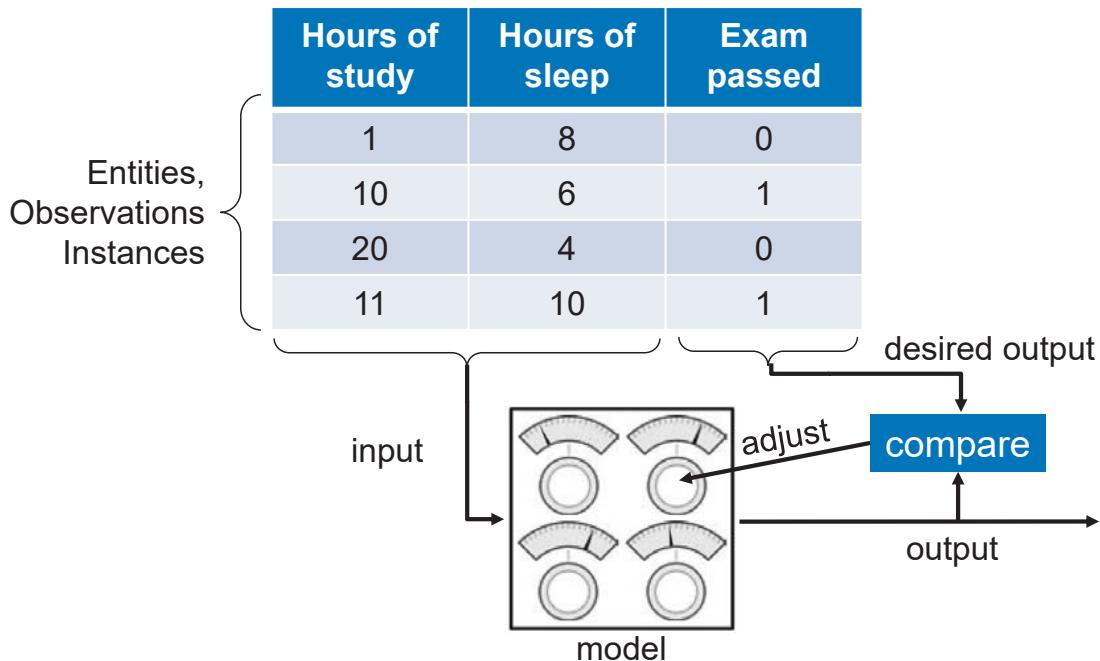
- ▶ Machine learning:  
“A field of study that gives computers the ability to learn without being explicitly programmed” (Arthur Samuel)
- ▶ More specific:
  - ▶ Data used to build models
  - ▶ Models used to make predictions on new data

▶ Example data:

	Features		Result
	Hours of study	Hours of sleep	Exam passed
Entities, Observations Instances	1	8	0
	10	6	1
	20	4	0
	11	10	1

- ▶ Exemplary task: try to predict if exam is passed, given hours of study and hours of sleep → so-called supervised learning

- ▶ Exemplary task: try to predict result of exam given hours of study and hours of sleep → so-called supervised learning



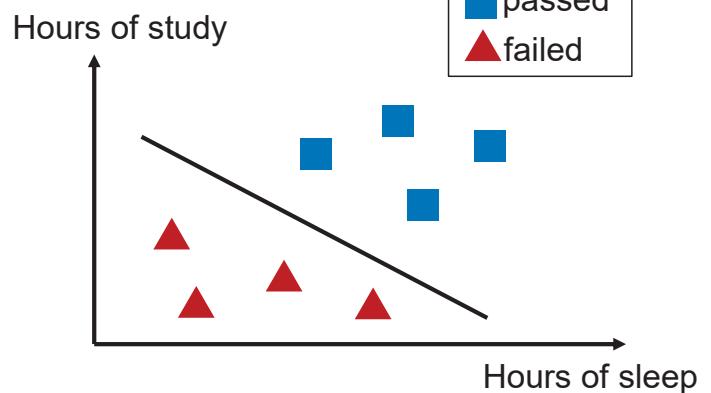
## Other Examples of Supervised Learning

- ▶ Using the **pixels** of an image to detect the *presence or absence of a cat*
- ▶ Using the **movies you've liked** to predict *movies you may like*
- ▶ Using someone's **words** to predict whether they are *happy or sad*.
- ▶ Using weather sensor **data** to predict the *probability of rain*.
- ▶ Using car engine **sensors** to predict the optimal tuning *settings*.
- ▶ Using news **data** to predict tomorrow's stock *price*.
- ▶ Using an input **number** to predict a *number double its size*.
- ▶ Using a raw **audio file** to predict a *transcript* of the audio.

# Two Main Applications of Supervised Learning

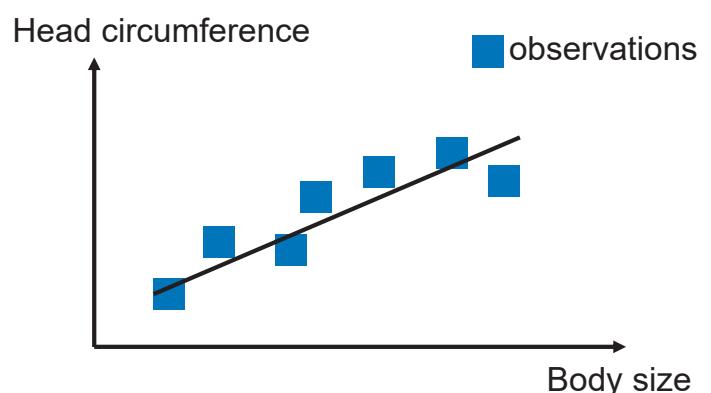
## ► Classification

- find the class boundaries
- output is categorical (+ confidence),  
e.g. exam passed, exam failed



## ► Regression

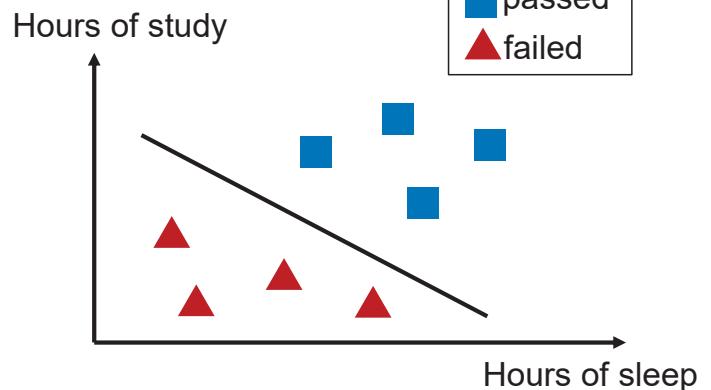
- Find the underlying relationship
- Output is a real value
- e. g. head circumference in cm



# Parametric ML-Algorithms

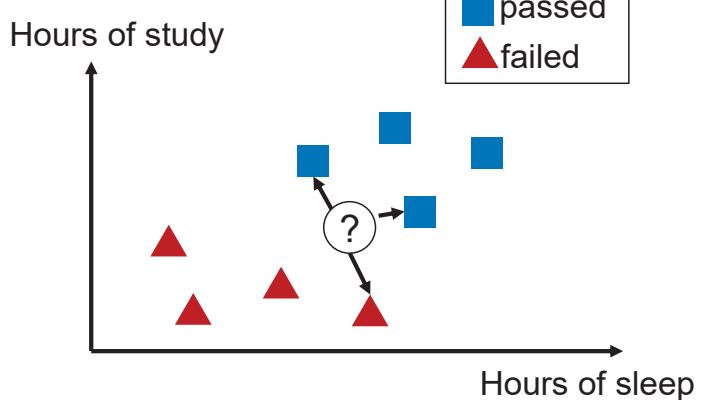
## ► Parametric

- Model with fixed number of parameters, number is independent from data
- e.g.: model of a line
- Pro:
  - Simple
  - Fast to determine parameters (= train)
  - Only few data is required to fit the model
- Con:
  - Model is constrained to specific form
  - Complexity is limited by number of parameters → could lead to poor fit



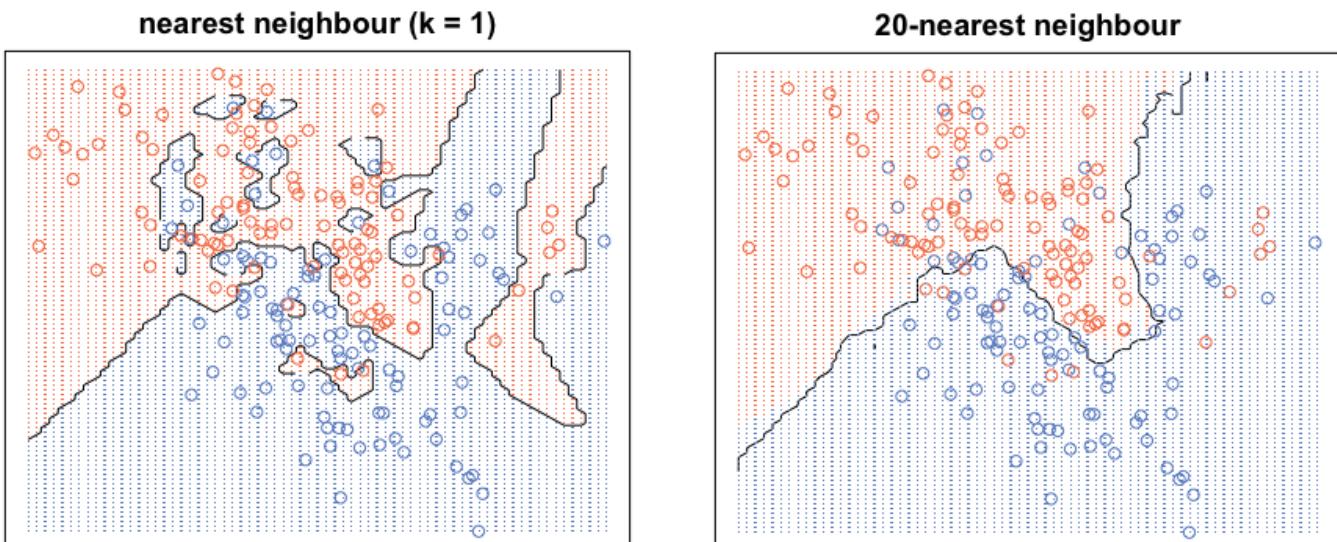
## ► Nonparametric

- No a-priori assumption about the form of the model function
- No fixed amount of parameters
- Example:  $k$ -nearest-neighbors algorithm
- Pro:
  - Flexibility, large number of functional forms to model
  - No or only weak assumptions about the underlying function needed
- Con:
  - Often: more training data required
  - More risk to overfitting

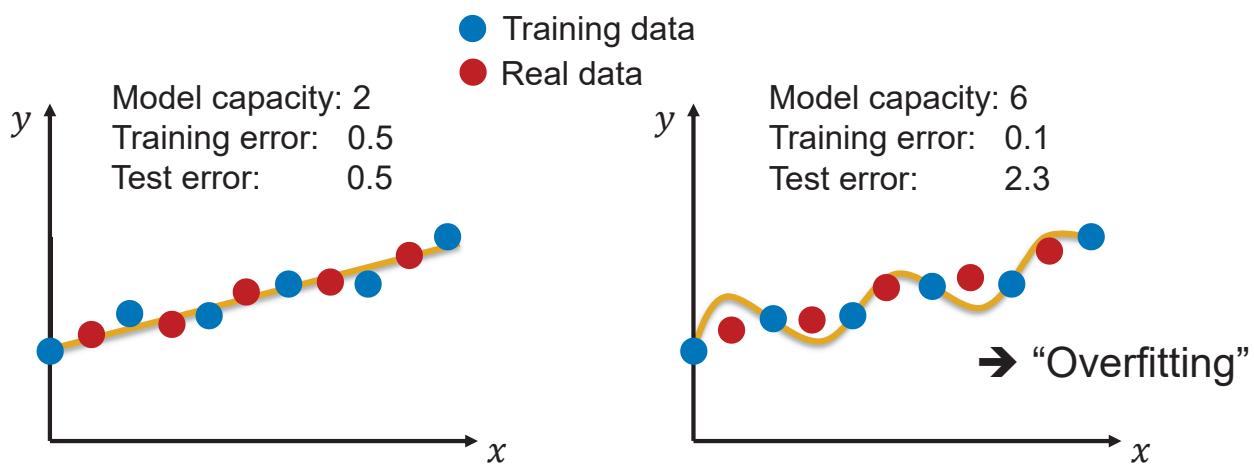


# Overfitting

- Goal:  
Find model, using given data → make predictions on new data
- Example: KNN-Classification

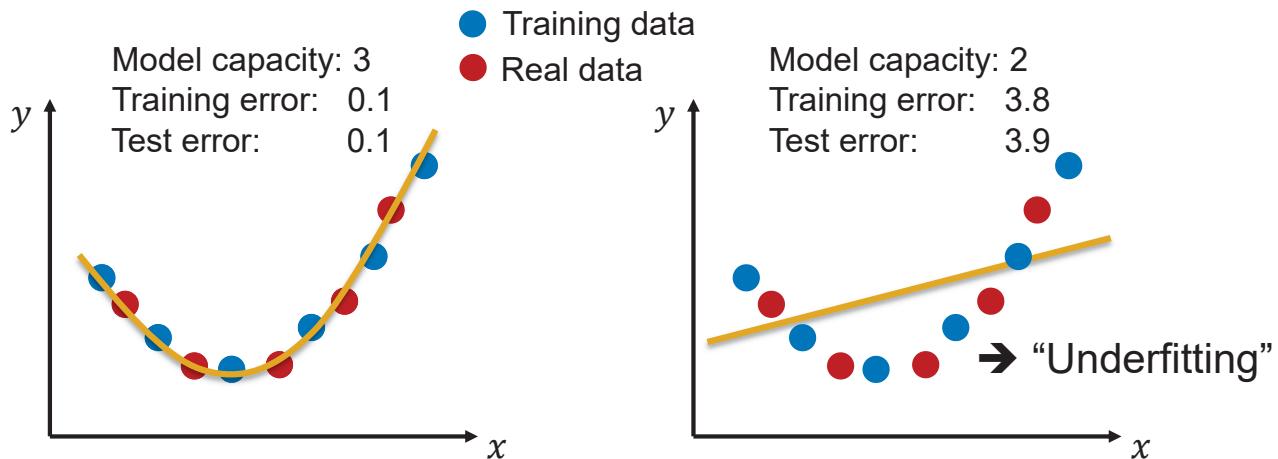


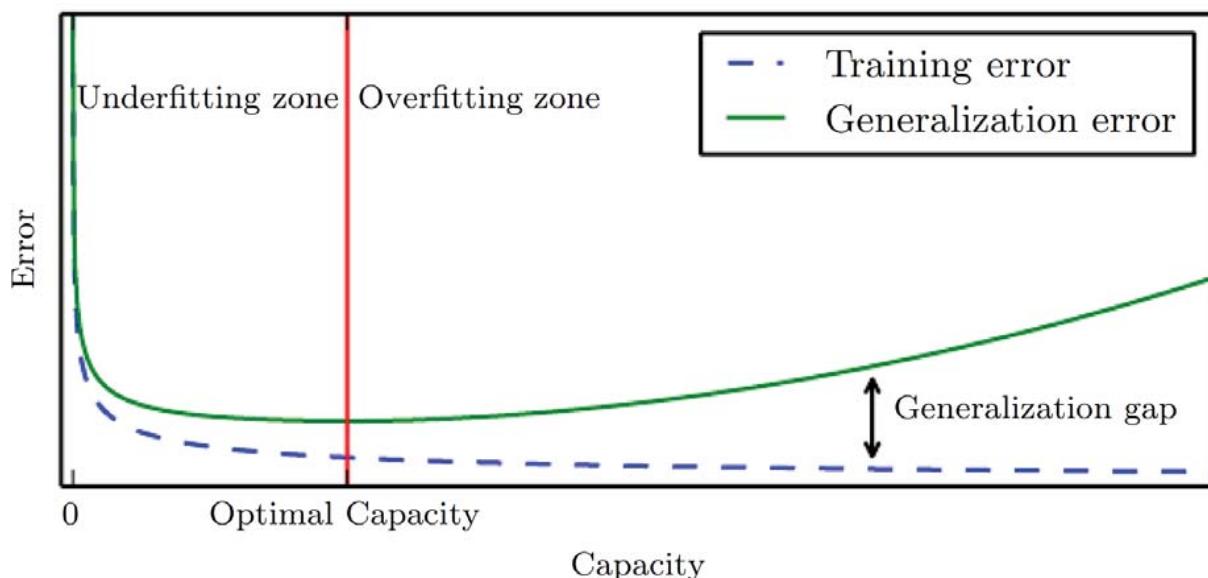
- ▶ Goal:  
Find model, using given data → make predictions on new data
- ▶ Example: Regression



# Underfitting

- ▶ Goal: Find model, using given data → make predictions on new data
- ▶ Example: Regression





- ▶ The more parameters of a parametric model, the higher the model's capacity
- ▶ Recognize Overfitting by checking the algorithm with unseen validation data → evaluate generalization error

## Summary and Overview on Learning

Supervised	Unsupervised	Reinforcement
<p>Data:</p> <ul style="list-style-type: none"> <li>• Input data</li> <li>• Output ground truth</li> </ul> <p>Scope:</p> <ul style="list-style-type: none"> <li>• Imitate the pattern between input and output → predict the output</li> </ul>	<p>Data:</p> <ul style="list-style-type: none"> <li>• Input data</li> </ul> <p>Scope:</p> <ul style="list-style-type: none"> <li>• Find patterns in the input data e.g. clustering of input data</li> </ul>	<p>Data:</p> <ul style="list-style-type: none"> <li>• Input data</li> <li>+ Reward function</li> </ul> <p>Scope:</p> <ul style="list-style-type: none"> <li>• Tries to find out what to do to maximize reward</li> </ul>



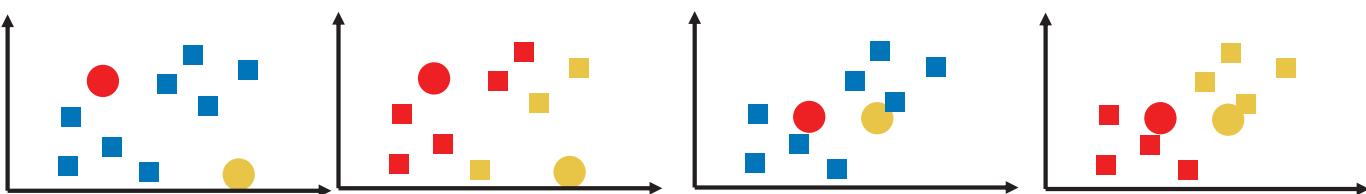
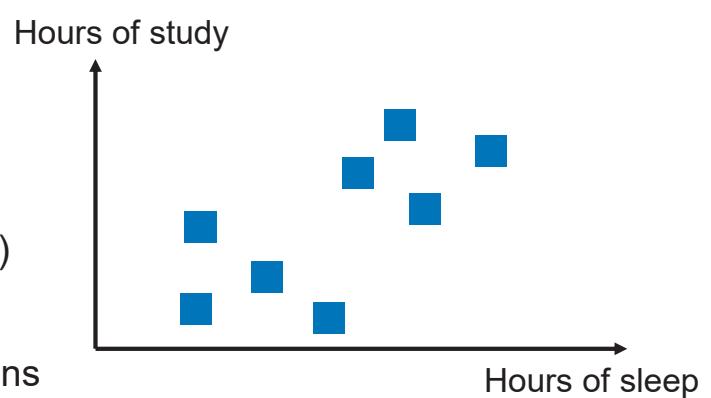
- ▶ Only input data available

	Features		Result
	Hours of study	Hours of sleep	
Entities, Observations Instances	1	8	0
	10	6	1
	20	4	0
	11	10	1

- ▶ Algorithms try to find structure in the data
- ▶ Main Application:  
Clustering of data (i.e. find inherent groupings in the data)
- ▶ Example:  $k$ -means Algorithm

## Example: K-Means Algorithm

- ▶ Input:
  - ▶  $k$  ... number of clusters we want to find
  - ▶ Input data points ( $h_{\text{sleep}}$ ,  $h_{\text{study}}$ )
- ▶ Approach:
  - ▶ Place  $k$  centroids at random locations
  - ▶ Repeat until convergence:
    - ▶ Assign each data point to the nearest centroid → this forms a cluster
    - ▶ Recompute centroid positions with the points of each cluster
  - ▶ Stop when none of the cluster assignments change



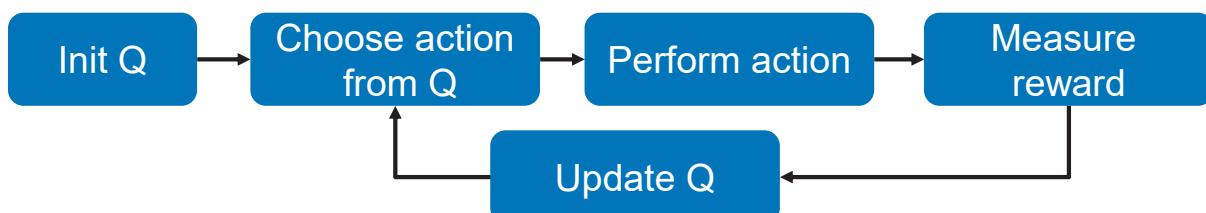
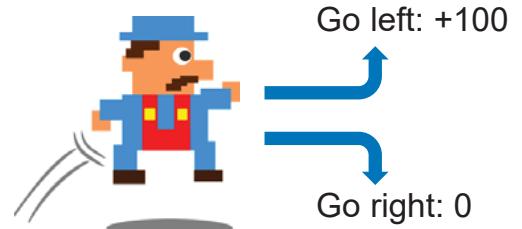
Supervised	Unsupervised	Reinforcement
<p>Data:</p> <ul style="list-style-type: none"> <li>• Input data</li> <li>• Output ground truth</li> </ul> <p>Scope:</p> <ul style="list-style-type: none"> <li>• Imitate the pattern between input and output → predict the output</li> </ul>	<p>Data:</p> <ul style="list-style-type: none"> <li>• Input data</li> </ul> <p>Scope:</p> <ul style="list-style-type: none"> <li>• Find patterns in the input data e.g. clustering of input data</li> </ul>	<p>Data:</p> <ul style="list-style-type: none"> <li>• Input data</li> <li>+ Reward function</li> </ul> <p>Scope:</p> <ul style="list-style-type: none"> <li>• Tries to find out what to do to maximize reward</li> </ul>



## Reinforcement Learning

### Q-Learning

- ▶ Agent performs actions in an environment and gets points for the actions as a reward
- ▶ Agent tries to learn how to best interact with the environment to maximize reward (without having a model of the environment)
- ▶ Q-Function(state, action): predict maximum future reward  
→ can be used for decision making to maximize cumulative reward
- ▶ Q-Function can be implemented as table with state as rows, actions as columns, containing the expected reward





Source: <https://www.youtube.com/watch?v=n7370dzCZ1o> (27.01.2018)

## Summary and Overview on Learning

Supervised	Unsupervised	Reinforcement
<p>Data:</p> <ul style="list-style-type: none"><li>• Input data</li><li>• Output ground truth</li></ul> <p>Scope:</p> <ul style="list-style-type: none"><li>• Imitate the pattern between input and output → predict the output</li></ul>	<p>Data:</p> <ul style="list-style-type: none"><li>• Input data</li></ul> <p>Scope:</p> <ul style="list-style-type: none"><li>• Find patterns in the input data e.g. clustering of input data</li></ul>	<p>Data:</p> <ul style="list-style-type: none"><li>• Input data</li><li>+ Reward function</li></ul> <p>Scope:</p> <ul style="list-style-type: none"><li>• Tries to find out what to do to maximize reward</li></ul>



Supervised	Unsupervised	Reinforcement
<p>Data:</p> <ul style="list-style-type: none"> <li>• Input data</li> <li>• Output ground truth</li> </ul> <p>Scope:</p> <ul style="list-style-type: none"> <li>• Imitate the pattern between input and output → predict the output</li> </ul>	<p>Data:</p> <ul style="list-style-type: none"> <li>• Input data</li> </ul> <p>Scope:</p> <ul style="list-style-type: none"> <li>• Find patterns in the input data e.g. clustering of input data</li> </ul>	<p>Data:</p> <ul style="list-style-type: none"> <li>• Input data</li> <li>+ Reward function</li> </ul> <p>Scope:</p> <ul style="list-style-type: none"> <li>• Tries to find out what to do to maximize reward</li> </ul>
 Classification: predict discrete responses	 Clusters	 Actions
 Regression: predict continuous responses		

## Deep Learning is a Section of ...

### Artificial Intelligence (not completely)

- knowledge input by hand-designed rules
- e.g. knowledge base system, chess-playing system

### Machine Learning

- system acquires own knowledge by extracting patterns from hand-defined features, i.e. it learns from data

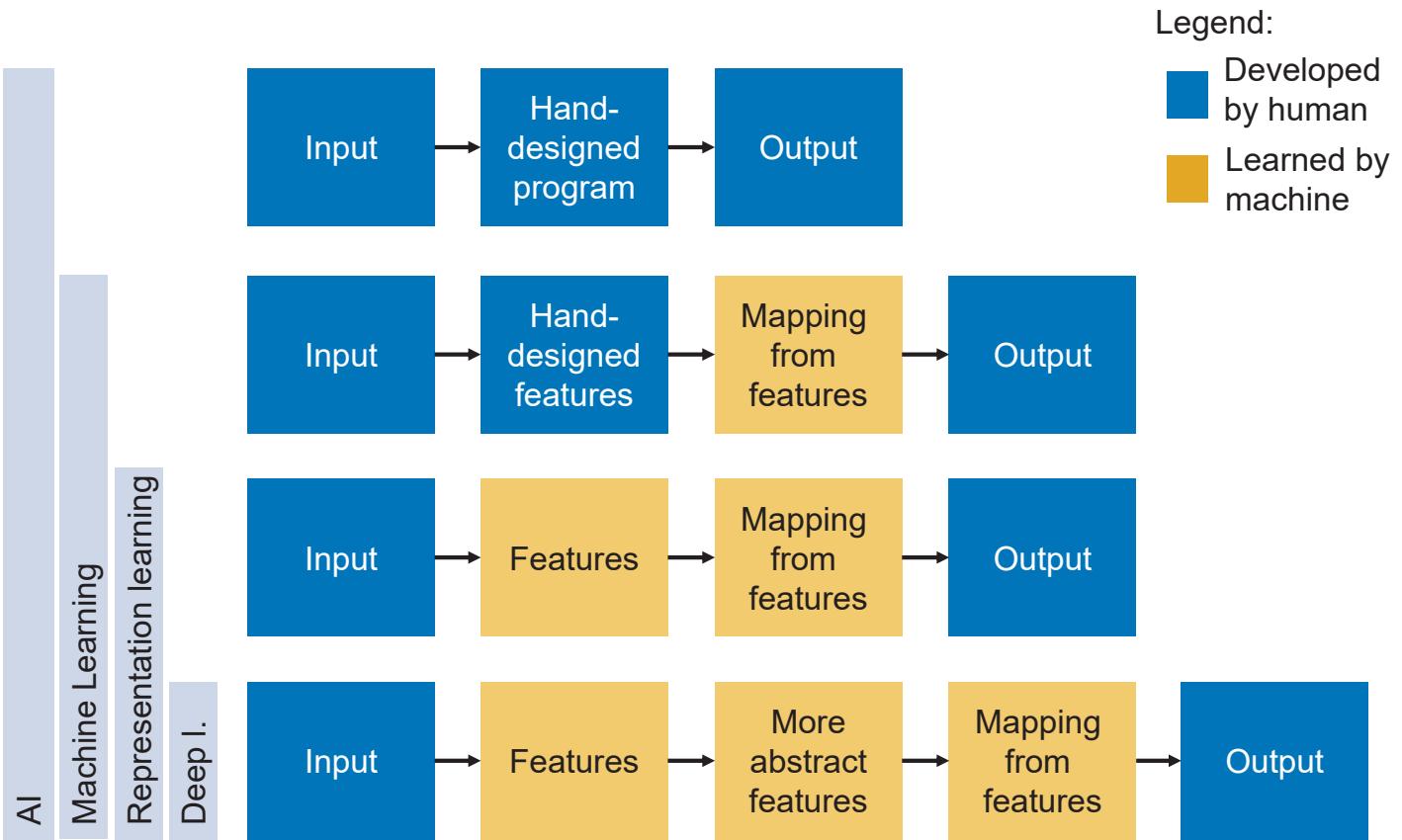
### Representation Learning

- System acquires own knowledge by finding feature representations of data and extracting patterns of it

### Deep Learning

- System acquires own knowledge by finding a hierarchy of representations and extracting patterns of it

# Deep Learning vs. Other AI Disciplines



Auto.-Sys: Deep Learning

Source: Goodfellow et. al.: Deep Learning, MIT Press, 2016.

Prof. Dr. N. Stache

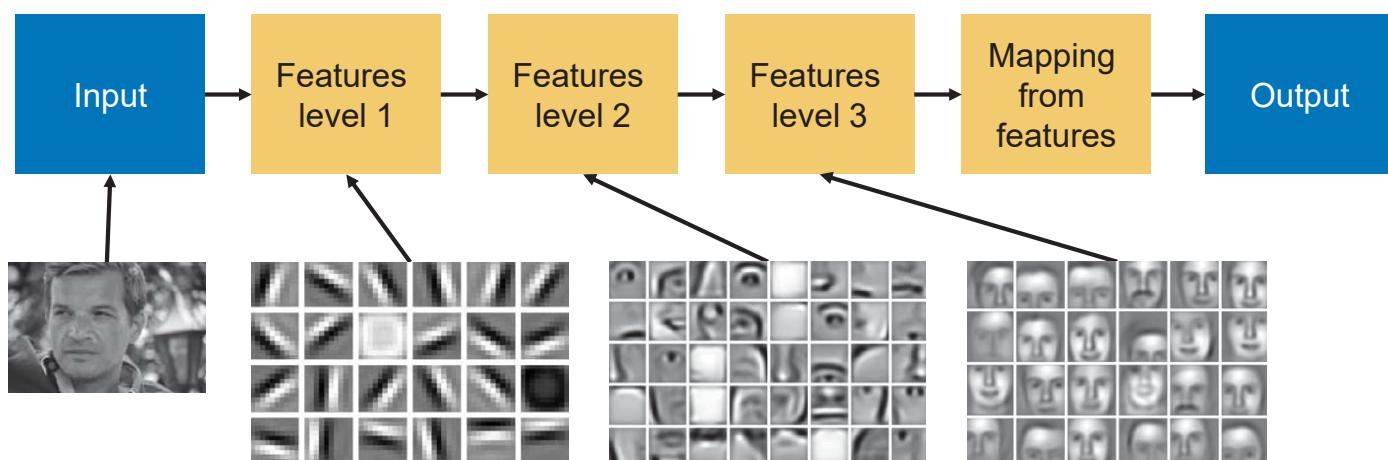
35

## What is a Feature?

- ▶ **Definition:**  
Individual measurable property of a phenomenon being observed  
[Source: Bishop, Christopher (2006). *Pattern recognition and machine learning*. Berlin: Springer]
- ▶ **Examples of hand-designed features in Computer Vision:**
  - ▶ Circles
  - ▶ Lines, Contours
  - ▶ Orientations
  - ▶ Corners
  - ▶ HOG, SIFT, ...



- ▶ Hierarchical features are determined by deep learning
- ▶ No hand-designed features needed
- ▶ Example:



Auto.-Sys: Deep Learning

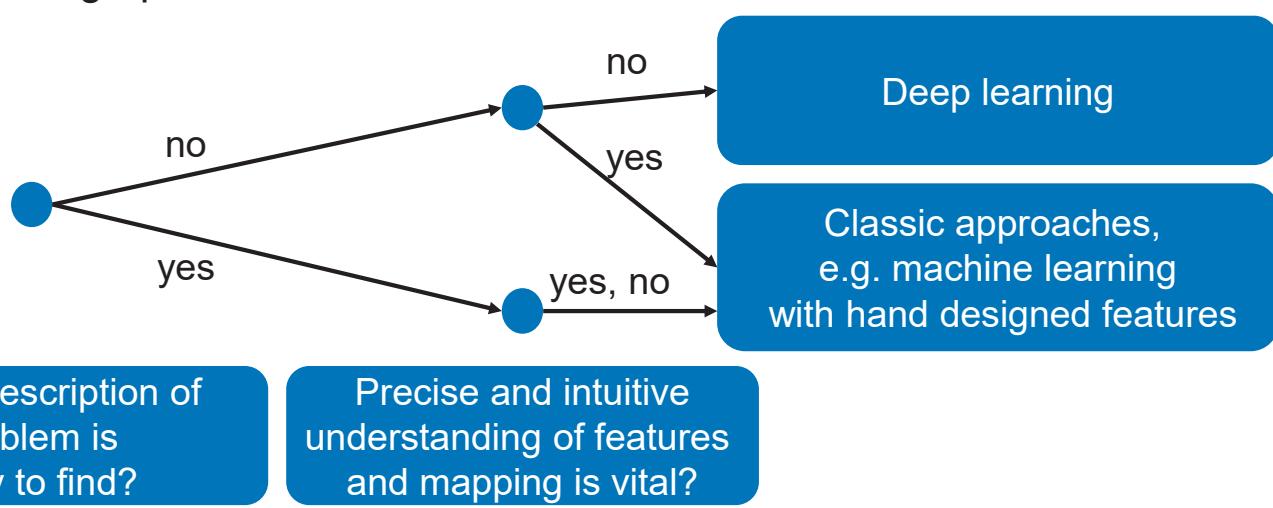
Source: <https://www.nature.com/news/computer-science-the-learning-machines-1.14481>, Images Andrew Ng, 27.08.2017

Prof. Dr. N. Stache

37

## Observations: Hand-Designed Features vs. Learned Representations

- ▶ Hand designed features: easy to understand
- ▶ In some cases: hand-designed features are difficult to define, e.g. if variation range is high → learned representations
- ▶ Decision graph:



## What is Deep Learning?

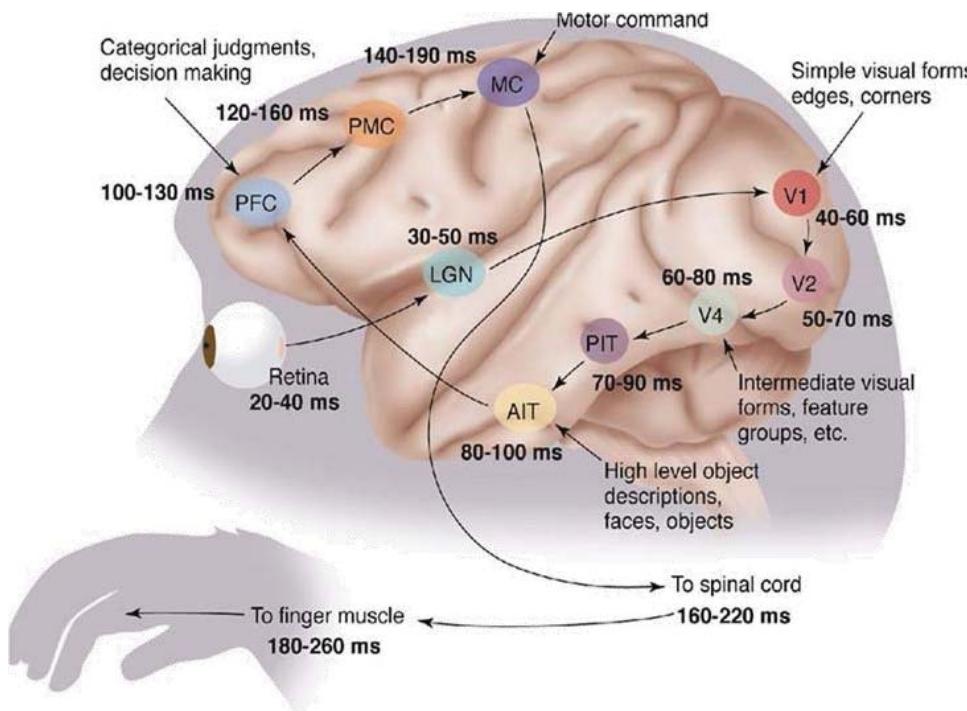
- ▶ Deep Learning is an approach for...
  - ▶ generating hierarchical representations from input data by itself and
  - ▶ mapping these to an output.

## When is deep learning beneficial?

- ▶ When formal description is difficult
- ▶ A precise understanding of features and the mapping to the output is not critical
- ▶ But: Visualization of learned features can be valuable

## Motivation for Deep Learning...

► Multiple stages in recognition pathway of mammals:



[picture from Simon Thorpe]

## ImageNet Large Scale Visual Recognition Challenge (ILSVRC)

► ImageNet:

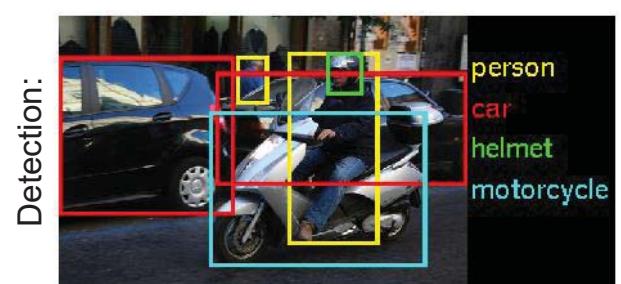
- > 14 000 000 annotated images
- > 20 000 classes
- Goal: 1000 images per class

► ILSVRC:

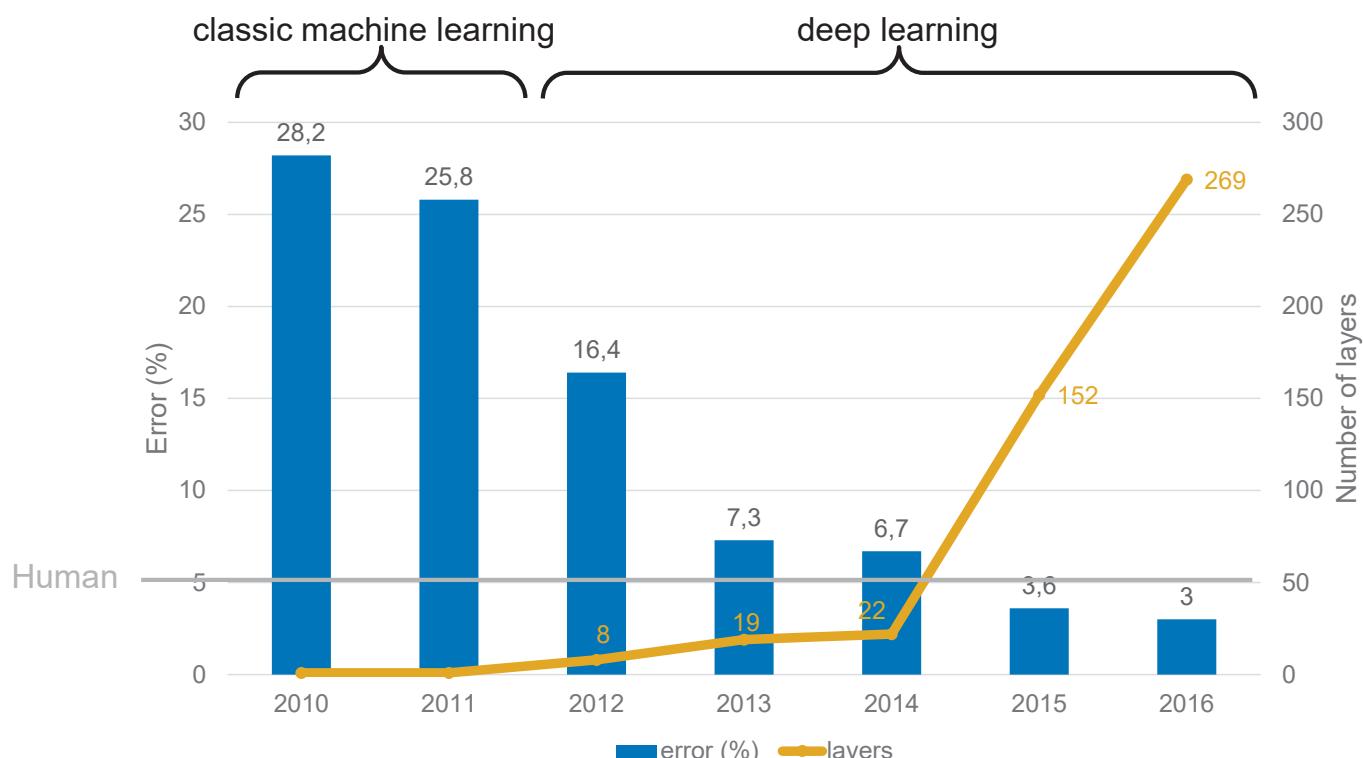
- **Classification**  
(i.e. what is in the image?  
→ 5 guesses + confidence)
- **Classification with localization**  
(i.e. like classification + 1 bounding box for each guess)
- **Detection**  
(i.e. like classification with localization but any number of objects possible, false positives are penalized)



Source: <https://blog.acolyer.org/2016/04/20/imagenet-classification-with-deep-convolutional-neural-networks/>. 27.08.2016



Source: <http://image-net.org/challenges/LSVRC/2014/>, 27.08.2016



→ Great Success of Deep Learning in ImageNet Challenge!

Auto.-Sys: Deep Learning

Source: [http://image-net.org/challenges/talks\\_2017/ILSVRC2017\\_overview.pdf](http://image-net.org/challenges/talks_2017/ILSVRC2017_overview.pdf),  
05.02.2017

Prof. Dr. N. Stache

43

## Engineering Speedup by Deep Learning

- ▶ No hand-engineered features with deep learning
- ▶ Re-use of pretrained networks (Transfer-learning)
- ▶ George Hotz, 26-year old hacker creates self-driving car!?



## ► Speech recognition

- Translation of spoken language into text



## ► Natural Language Processing

- Understanding of natural language to perform tasks, e.g.
  - Question answering
  - Text summarization
  - Search
  - Machine translation
  - Sentiment analysis

## ► Computer vision

- Handwriting recognition
- Face & People Detection
- (Low-resolution) Object recognition
- Semantic labelling



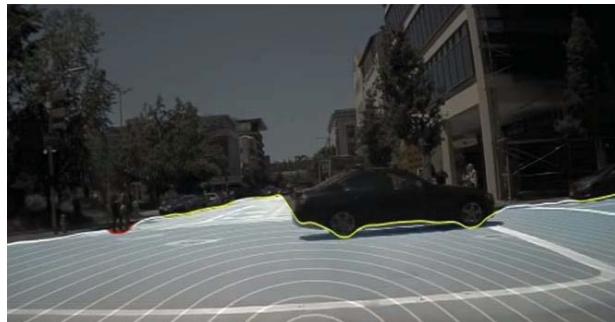
## ► Applied mathematics



- ▶ Detection of objects
- ▶ Detection of lane markings
- ▶ Detection of free space + classification of the boundary



Source: <https://www.youtube.com/watch?v=URmxzxYlmtg>, 31.08.2017,  
Talk by NVIDIA CEO Jen-Hsun Huang at GTC Europe



Source: <https://www.youtube.com/watch?v=URmxzxYlmtg>, 31.08.2017,  
Talk by NVIDIA CEO Jen-Hsun Huang at GTC Europe



Source: <https://www.youtube.com/watch?v=URmxzxYlmtg>, 31.08.2017,  
Talk by NVIDIA CEO Jen-Hsun Huang at GTC Europe

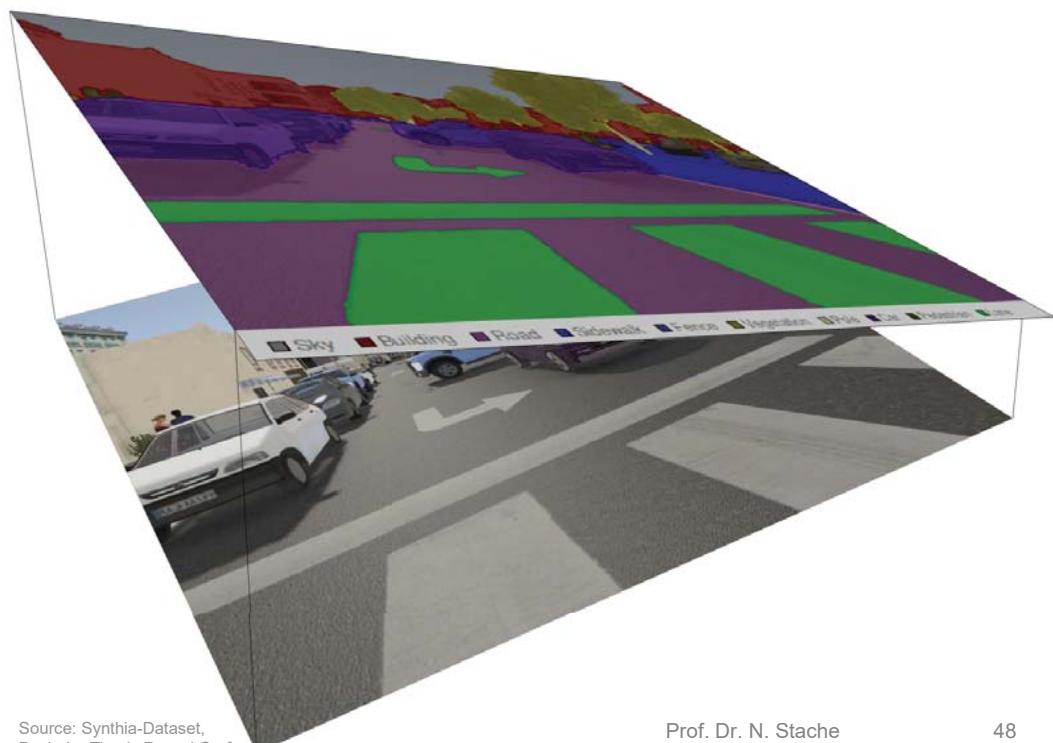
Auto.-Sys: Deep Learning

Prof. Dr. N. Stache

47

# Automotive-Related Applications

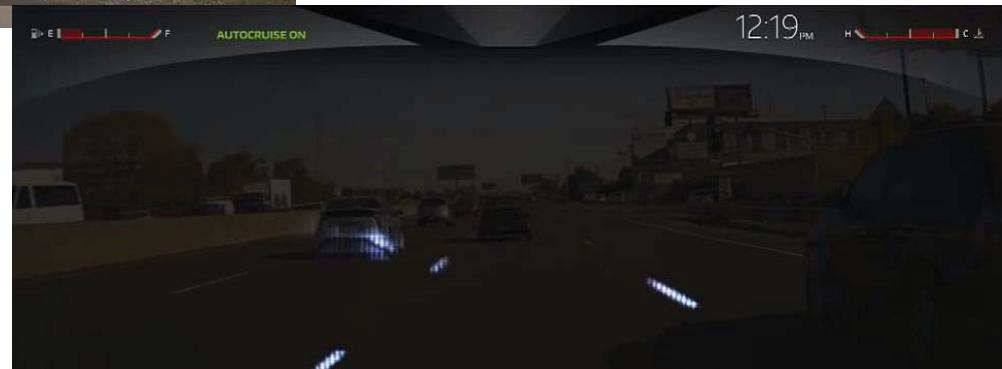
- ▶ Image Segmentation



## ► End-to End-Learning (e.g. NVIDIA BB8)



Source: <https://www.youtube.com/watch?v=URmxzxYlmtg>, 31.08.2017,  
Talk by NVIDIA CEO Jen-Hsun Huang at GTC Europe



Source: <https://www.youtube.com/watch?v=URmxzxYlmtg>, 31.08.2017,  
Talk by NVIDIA CEO Jen-Hsun Huang at GTC Europe

Visualization of  
relevant image  
regions for automated  
driving:

Auto.-Sys: Deep Learning

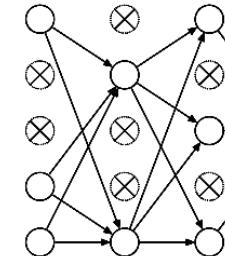
Prof. Dr. N. Stache

49

## Why Deep Learning?

- Hierarchical features like in our brains
- Outperformed classical machine learning approaches in mentioned areas
- Reduces development time
- Learn more about the (complex) problem by visualizing deep learning features
- Seems to be successful in automotive applications

- ▶ Massive labeled datasets
- ▶ Massive parallel computing power (e.g. GPUs)
- ▶ Software & algorithmic improvements
  - ▶ Improved optimizers
    - e.g. ADAM optimizer
  - ▶ New regularization techniques
    - e.g. dropout
  - ▶ Backprop-friendly activation functions
    - e.g. ReLU
  - ▶ Software-Libraries
    - e.g. TensorFlow, Theano, Caffe...



Source: Srivastava, Hinton, Krizhevsky, Sutskever, Salakhutdinov: Dropout: A simple Way to Prevent Neural Networks from Overfitting. J of Machine Learning Research 15 (2014) 1929-1958.

## Fun Projects

- ▶ Style transfer
  - ▶ Under Linux - open terminal, enter:
    - ▶ conda create -n style-transfer1 python=3.5
    - ▶ source activate style-transfer1
    - ▶ pip install tensorflow
    - ▶ conda install scipy pillow
    - ▶ conda install -c conda-forge moviepy
  - ▶ Under Windows – open terminal, enter:
    - ▶ conda create -n style-transfer python=3.5
    - ▶ activate style-transfer
    - ▶ pip install tensorflow
    - ▶ conda install scipy pillow
    - ▶ conda install -c conda-forge moviepy

*Step can be skipped  
on pool PCs!*

## ► Style transfer

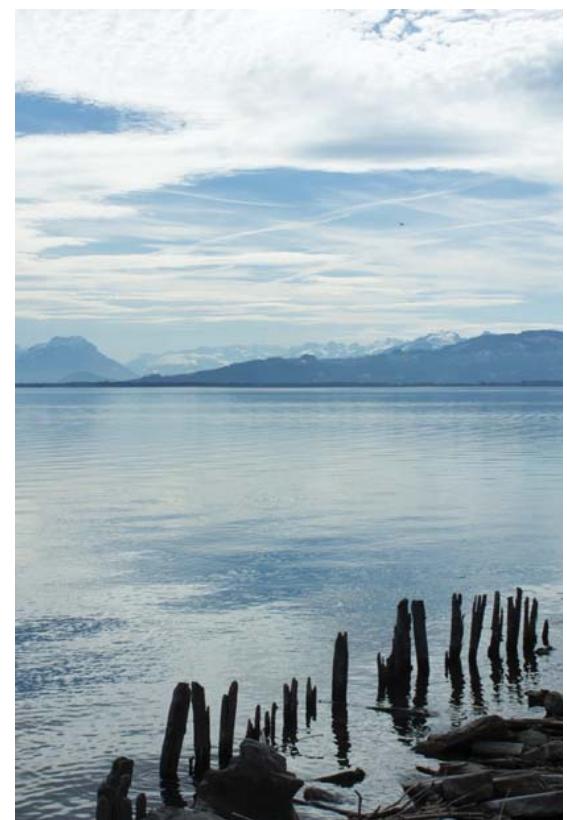
- Download the zip archive of fast style transfer and extract it  
<https://github.com/lengstrom/fast-style-transfer>
- Download the Rain Princess checkpoint (model with already tuned parameters), put it into the style transfer folder  
<https://goo.gl/niFGRc>
- Copy your image into the fast style transfer folder
- Enter in your terminal:
  - `python evaluate.py --checkpoint ./rain-princess.ckpt --in-path <path_to_input_file> --out-path ./output_image.jpg`

# Fun Projects

## ► Style transfer

- Other checkpoints:
  - Rain-Princess: <https://goo.gl/niFGRc>
  - La-muse: <https://goo.gl/AqzB6P>
  - Udnie: <https://goo.gl/B8gvAH>
  - Scream: <https://goo.gl/dkA2K9>
  - Wave: <https://goo.gl/UDTr3e>
  - Wreck: <https://goo.gl/wRJh2v>

My test image: Lake Constance



## ► Deep Traffic

- Open: <https://selfdrivingcars.mit.edu/deeptraffic/>
- Modify the parameters
  - lanesSide
  - patchesAhead
  - patchesBehind
  - Number of hidden layers
  - num\_neurons
- Try to get it faster than 65 mph

Next week:

Python + Tooling + how to install  
→ Please bring your Laptops!



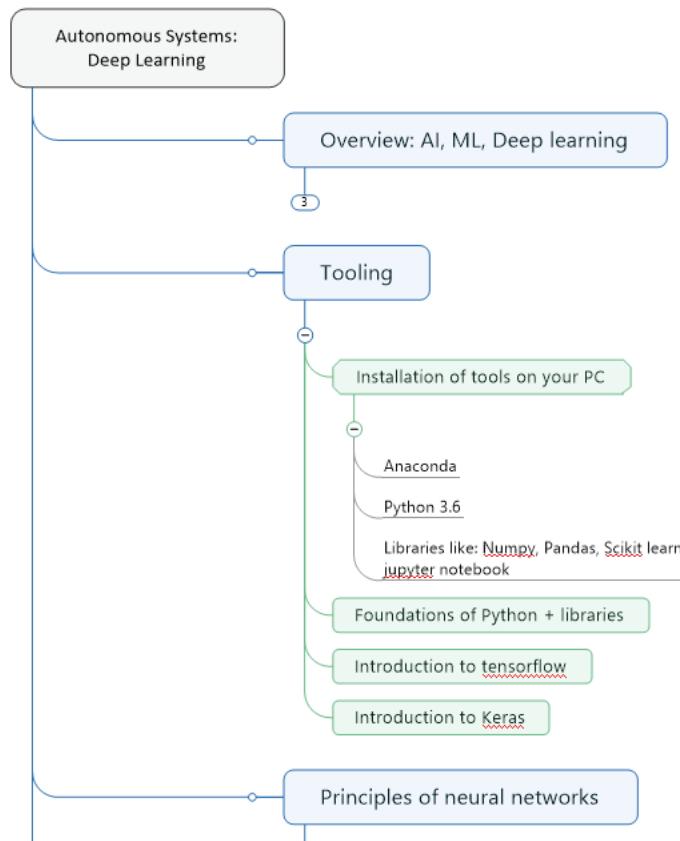
## Autonomous Systems: Deep Learning

### 2. Familiarization with Python + Tooling



Prof. Dr.-Ing. Nicolaj Stache  
Heilbronn University of Applied Sciences

## Overview on the course



# Learning objectives

- ▶ You are able to install all relevant tools and use them at home
- ▶ You know the purpose of Anaconda
- ▶ You have basic knowledge of Python
- ▶ You know how to get help in the internet and learn in depth by yourself
- ▶ You are able to use and to create Jupyter Notebooks

## What is our purpose?

- ▶ Our typical workflow:



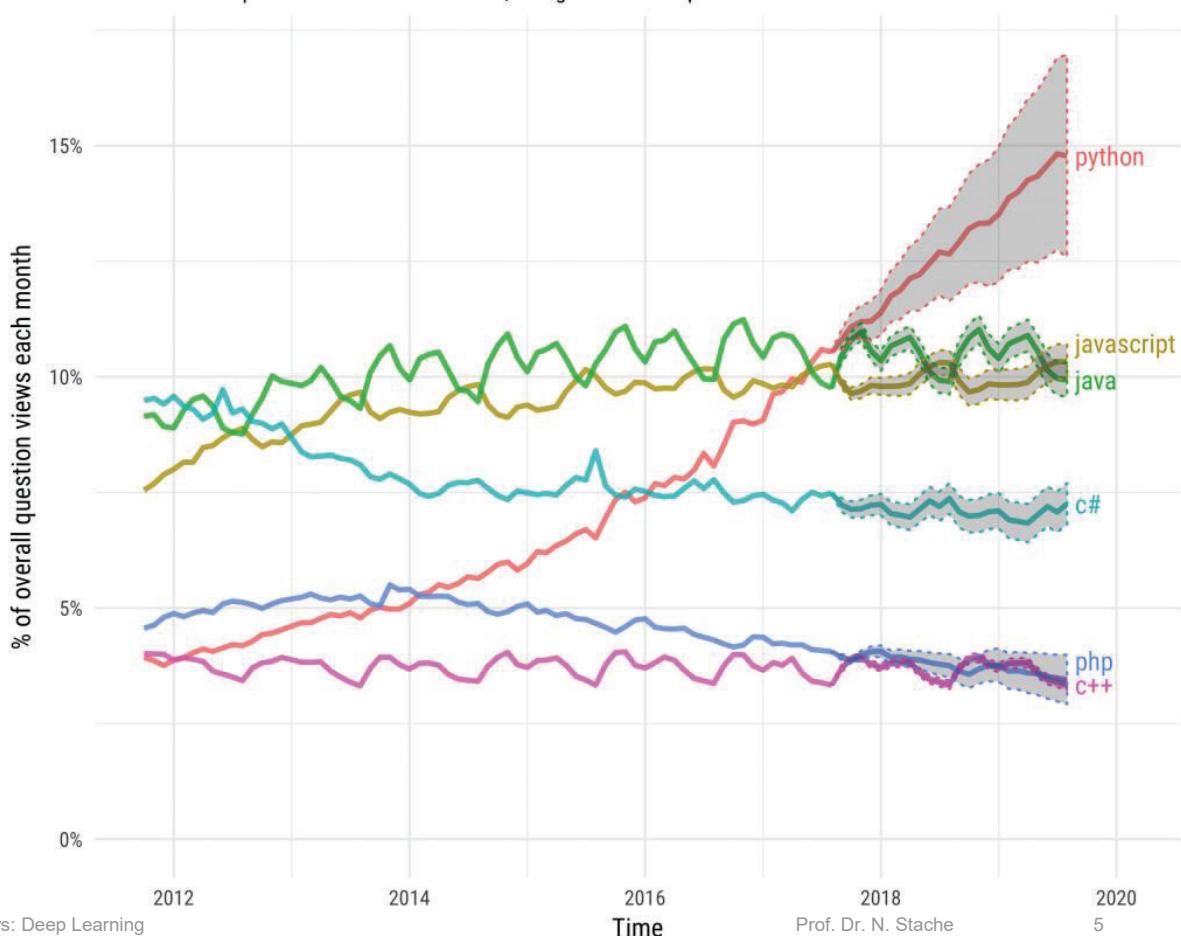
- Load data
- Convert data formats
- Handle different types of data in flexible structures
- Visualize data frames
- Methods to remove outliers
- Simple interface to leading libraries for data exploration

- ▶ Python satisfies these requirements quite well
- ▶ Python is one of the leading languages in data science

# Programming languages rated by stackoverflow

## Projections of future traffic for major programming languages

Future traffic is predicted with an STL model, along with an 80% prediction interval.

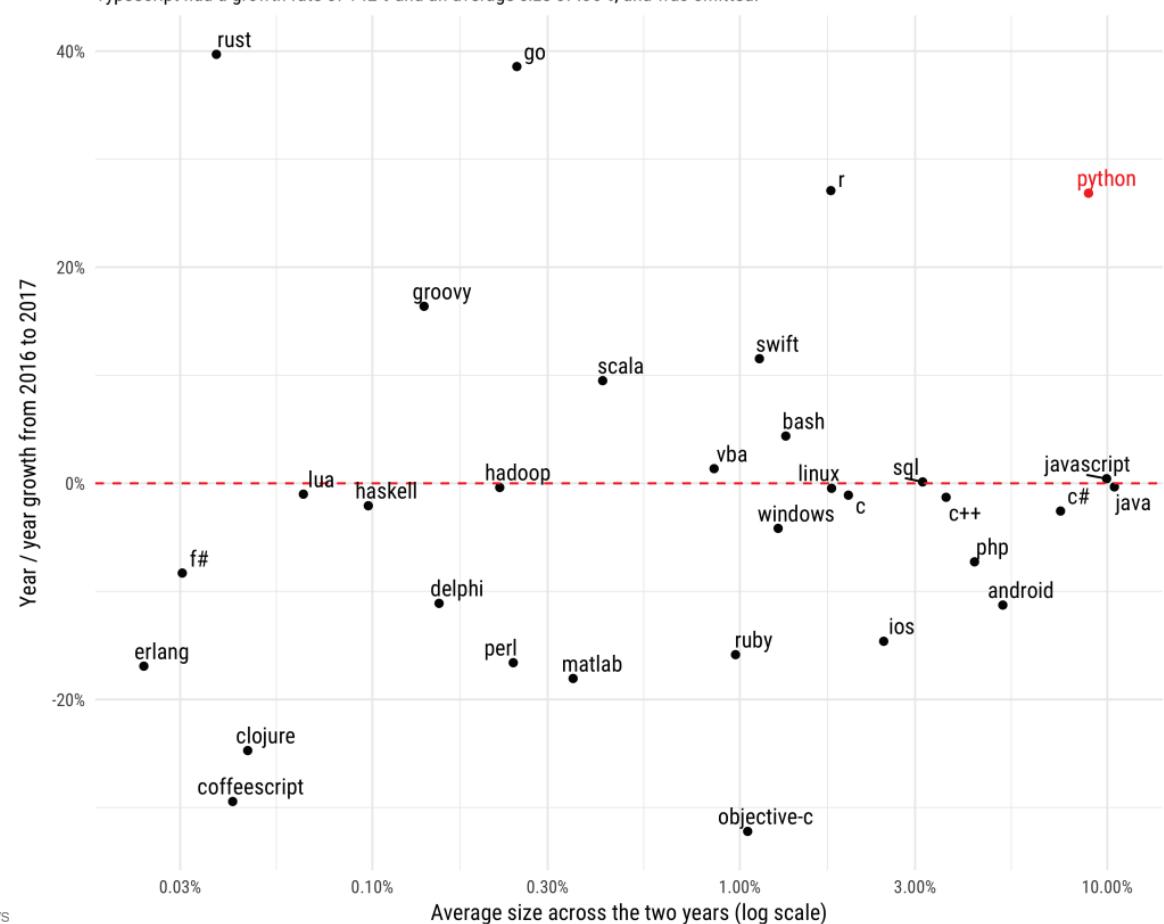


Source: <https://stackoverflow.blog/2017/09/06/incredible-growth-python/>, 03.03.2018

# Programming languages rated by stackoverflow

## Year over year growth in traffic to programming languages/platforms

Comparing question views in January-August of 2016 and 2017, in World Bank high-income countries. TypeScript had a growth rate of 142% and an average size of .36%; and was omitted.



Source: <https://stackoverflow.blog/2017/09/06/incredible-growth-python/>, 03.03.2018

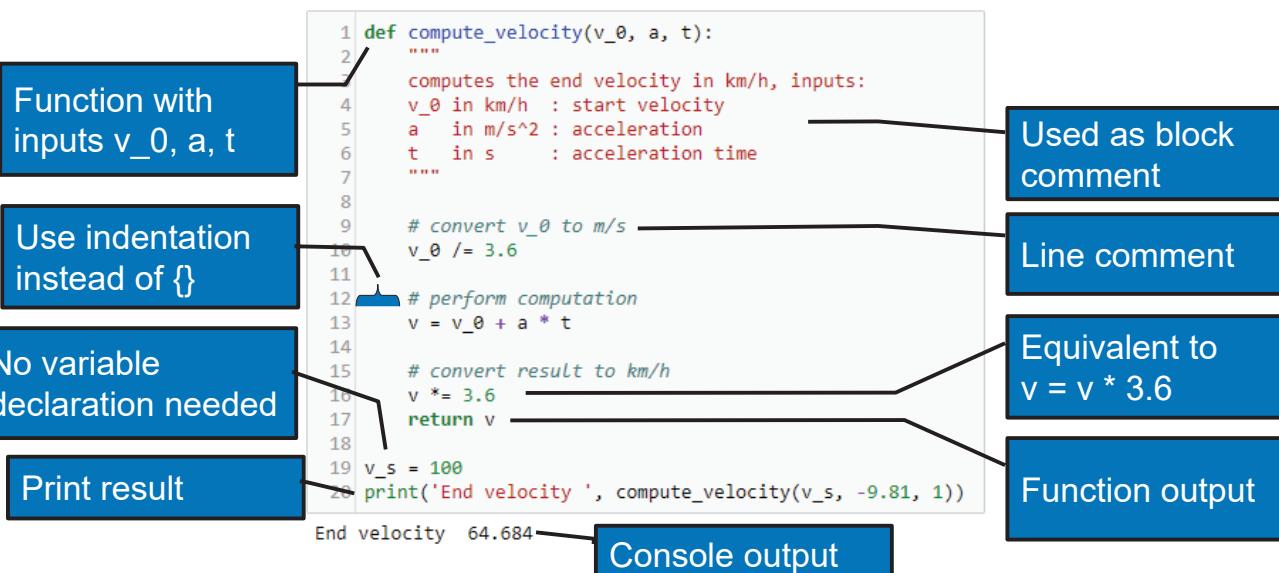


- ▶ Invented in 1990 by Guido van Rossum
- ▶ Name inspired from Monty Python's flying circus
- ▶ High level programming language
- ▶ Dynamic type and memory management
- ▶ Supports object orientation
- ▶ Interpreted language:
  - ▶ Can be directly typed into a shell
  - ▶ Stored in a file (can be executed by interpreter)
- ▶ Indentation matters as structuring element
- ▶ Incompatibility between Version 2.x and 3.x



Source: [https://en.wikipedia.org/wiki/Python\\_\(programming\\_language\)](https://en.wikipedia.org/wiki/Python_(programming_language)), 04.04.2018

## Python example



Library	Usage
numpy, scipy	Scientific & technical computing
pandas	Data manipulation and aggregation
scikit-learn	Machine learning
tensorflow, keras, theano, ...	Deep learning
matplotlib, bokeh	Vizualization
beautifulsoup, scrapy	Web scraping

## Jupyter Notebook

- ▶ Code and formatted text in one document
- ▶ Access via webbrowser
- ▶ Computations run on our server

The screenshot shows a Jupyter Notebook interface with several callout boxes highlighting specific features:

- File menu:** save, close, open other notebook, etc.
- Run cell [Shift]+[Enter]**
- Stop execution**
- Restart kernel**
- When dark: kernel is busy**
- Save notebook**
- Code-cell with Python code**
- Text cell for explanations**
- Command palette**

The notebook content includes:

- Create Input Data**: A text cell with instructions to create an array of input data.
- In [ ]:** A code cell containing Python code to create a 4x3 array and print the third column.
- Create Placeholders**: A text cell with instructions to create placeholders for input data.
- Output**: The resulting output of the code cells, showing the created arrays and placeholder definitions.

# Before we try it out

## Installation on your own PC

### Anaconda

- ▶ Distribution of python packages for data science.
- Comes with conda, a packet manager
- ▶ Allows different Python configurations installed in parallel

### Step 1: Installation of Anaconda for python 3

Please follow the installation guide on ILIAS!  
 (Deep Learning - Prof. Stache >> Tool-Installation)

### A few words on conda

Conda packages  
 (packages contain libraries like numpy, tensorflow ...)

Commands	Explanation
<code>conda install package_name</code>	Install, remove, update the package specified by "package_name"
<code>conda remove package_name</code>	
<code>conda update package_name</code>	
<code>conda update -all</code>	Update all packages
<code>conda search search_term</code>	Search for packages
<code>conda list</code>	Lists installed packages of current environment

# A few words on conda

## Conda environments

Commands	Explanation
<code>conda create -n env_name list of packages</code>	Creates a new environment with the name “env_name” with the packages “list of packages” installed
<code>conda create -n py37 python=3.7</code>	Example environment name py37, contains python of version 3.7
<code>source activate my_env / activate my_env</code>	Switch to the environment my_env (source activate used under Linux)
<code>source deactivate / deactivate</code>	Close the current environment
<code>conda env export &gt; environment.yaml</code>	Export an environment in a yaml file
<code>conda env create -f environment.yaml</code>	Creates a new environment with the same name and packages as listed in the yaml file
<code>conda env remove -n env_name</code>	Removes the environment env_name

## Let's create a new environment py35 and install packages

- ▶ The following packages shall be installed:
- ▶ Python 3.7
- ▶ Conda Packages: `notebook tensorflow==2.0.0 tensorflow-gpu==2.0.0 matplotlib pandas pillow scikit-image scikit-learn opencv spyder`
- ▶ `conda create -n dl python=3.7 notebook tensorflow==2.0.0 tensorflow-gpu==2.0.0 matplotlib ...`
- ▶ Windows: `activate dl`  
Linux: `source activate dl`
- ▶ Windows and Linux: `jupyter notebook`

# Try out python (local installation at home)

- ▶ Open the anaconda prompt
- ▶ Navigate to your working directory
- ▶ Activate the environment `d1` via  
`source activate d1` (Linux, MacOS)  
or `activate d1` (Windows)
- ▶ Type `jupyter notebook`
  - ▶ Create a new notebook, open existing notebooks from the folder
- ▶ Alternatively, type `spyder`  
(this is a development environment)

## Get hands-on with the python tutorial

- ▶ Log-in to ILIAS, Autonomous Systems Deep Learning
- ▶ Download the folder Notebooks/Introduction to Python
- ▶ Open the shell, navigate to the folder of the notebooks
- ▶ Execute `(source) activate d1`
- ▶ Execute `jupyter notebook`
- ▶ Work through 01\_Python\_tutorial\_Part1 to  
02\_Python\_tutorial\_Part2
- ▶ Do all the exercises!



## Autonomous Systems: Deep Learning

### 4. Network Validation and Training



Prof. Dr.-Ing. Nicolaj Stache  
Heilbronn University of Applied Sciences

### Learning Objectives



- ▶ You learn how the general training process looks like
- ▶ You get a deeper understanding of overfitting, underfitting, model capacity
- ▶ You learn the purpose of a validation dataset
- ▶ You get familiar with strategies for dealing with a mix of real values and categorical data
- ▶ You get familiar with the working principle of tensor flow

- ▶ Learnings from Gardening example
- ▶ Extension to Neural networks including bias term (XOR)
- ▶ Training process, model evaluation and validation
- ▶ Data preparation, student admission example
- ▶ Introduction to TensorFlow

## Gardening Example – What did we learn?

Forward pass

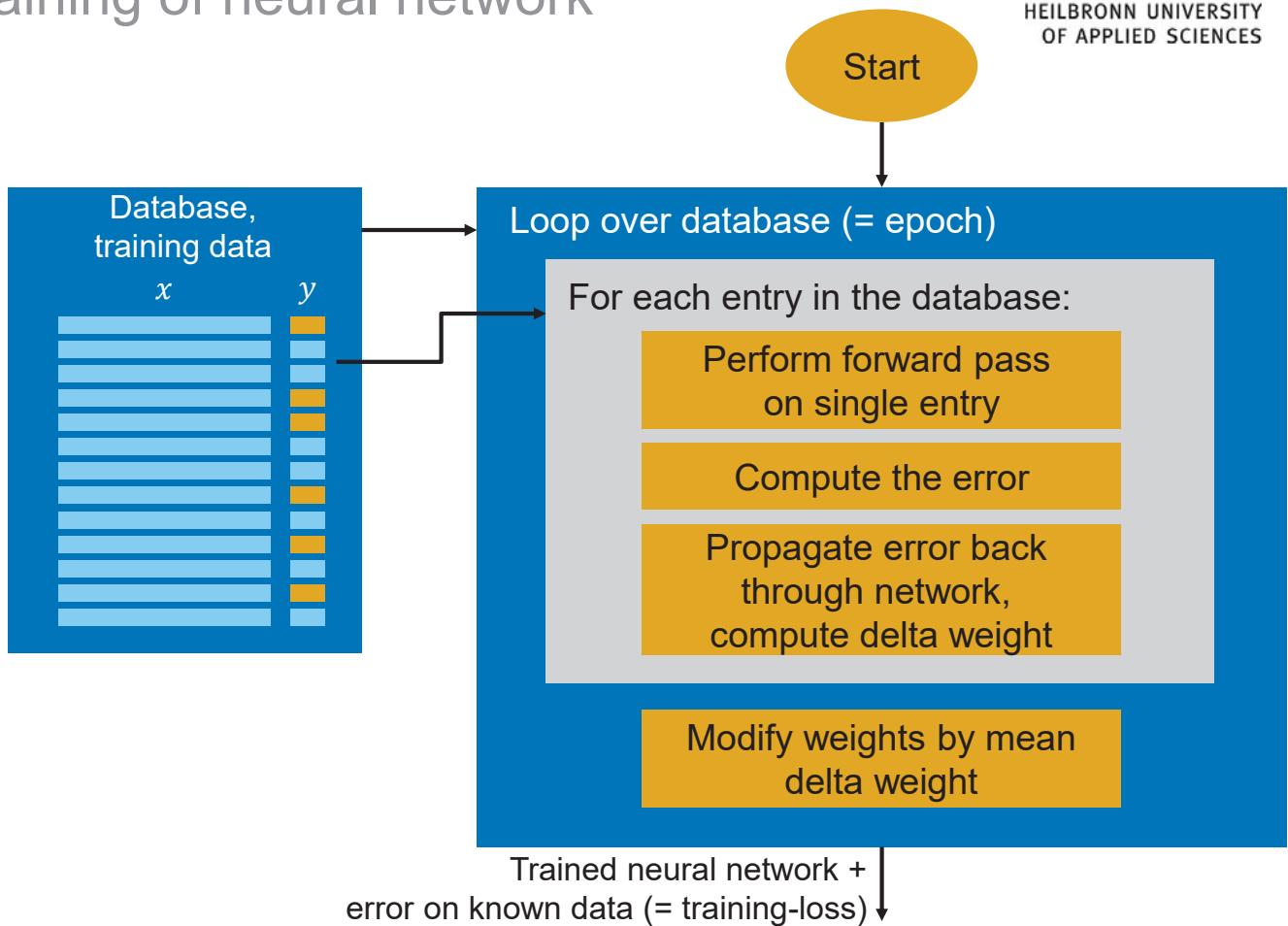
Gradient descent

Backprop

Learning rate  $\eta$

Understand  
matrix  
computations

Activation  
function,  
here: sigmoid



## Outline

- ▶ Learnings from Gardening example
- ▶ Extension to Neural networks including bias term (XOR)
- ▶ Training process, model evaluation and validation
- ▶ Data preparation, student admission example
- ▶ Introduction to TensorFlow

- ▶ We defined parameters like:
  - ▶ Number of epochs
  - ▶ Learning rate
  - ▶ Network architecture
  - ▶ Initialization of weights and biases
  - ▶ type of activation function
- ▶ by trial and error

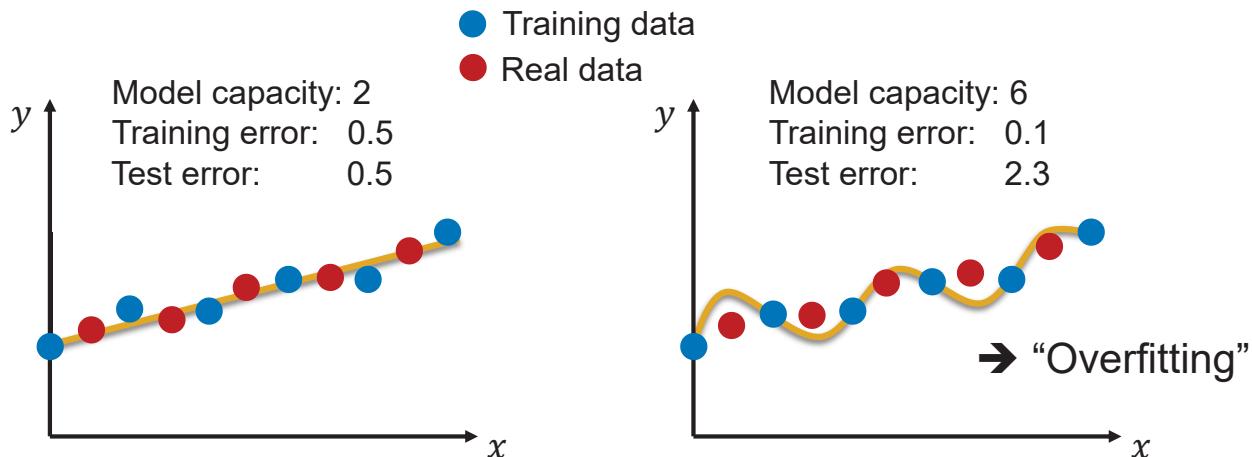
Question:

- ▶ What is the “quality” of the found result?
- ▶ How can it be measured?
- ▶ How can we define the parameters in a better way?
  - will be covered later in class

## Model Validation

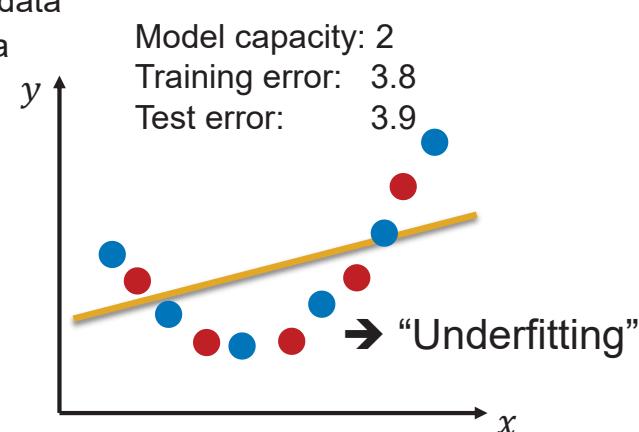
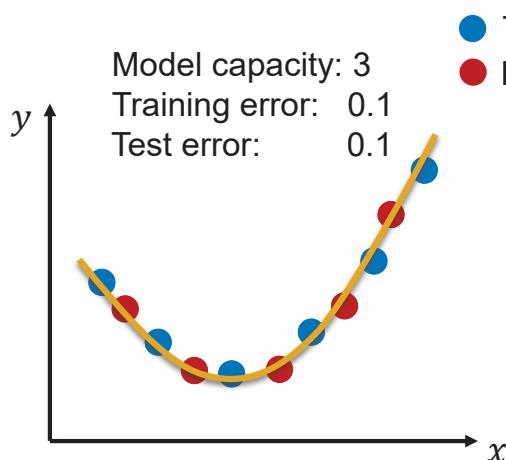
Revision: Overfitting

- ▶ Goal:  
Find model, using given data → make predictions on new data
- ▶ Example: Regression



- ▶ Goal: Find model, using given data → make predictions on new data

- ▶ Example: Regression



## Discussion on effects of overfitting / underfitting

Class: No dogs wagging their tail

← Overfitting →

Class: Dogs wagging their tail

Class: Non-Animals

← Underfitting →

Class: Animals

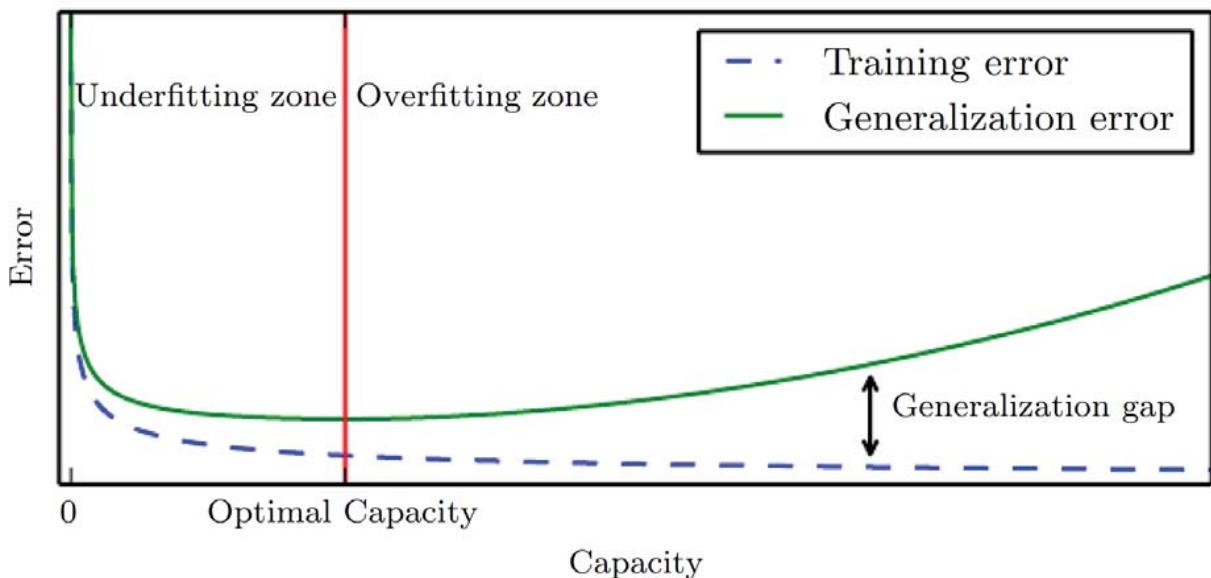




[https://cdn.pixabay.com/photo/2016/03/14/17/34/wizard-1293759\\_960\\_720.png](https://cdn.pixabay.com/photo/2016/03/14/17/34/wizard-1293759_960_720.png), 07.04.2018

► Never use testing data for training!

## Model complexity graph



- The more parameters of a parametric model, the higher the model's capacity
- Recognize Overfitting by checking the algorithm with unseen validation data → evaluate generalization error

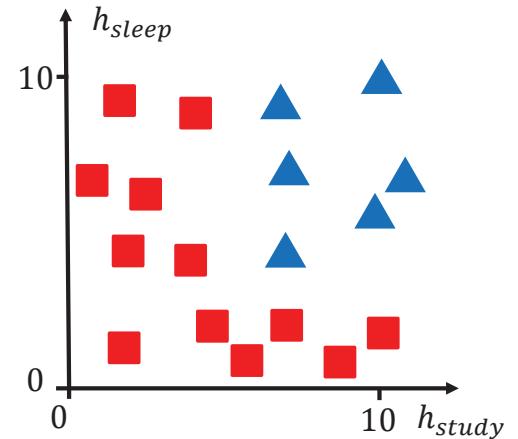
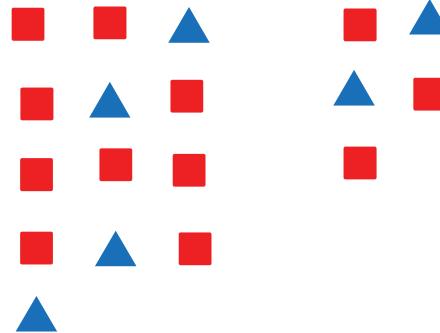
- Optimal model capacity (e.g. degree of polynomial) is determined by interpreting results on training data and testing data:

Model capacity	Error on training data	Cross validation	Result
		Error on testing data	
Low	High	High	Underfitting
Medium	Low	Low	Optimal fit
High	Very low	Medium - high	Overfitting

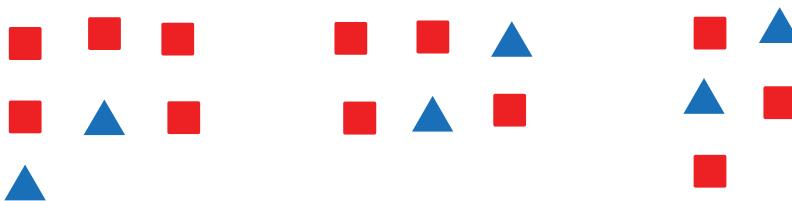
- Note: This breaks the golden rule:  
Never use test data for training!  
Why?
- Approach: Cross validation data

## Datasets

- Formerly: Training set + testing set



Now: Training set + validation set + testing set



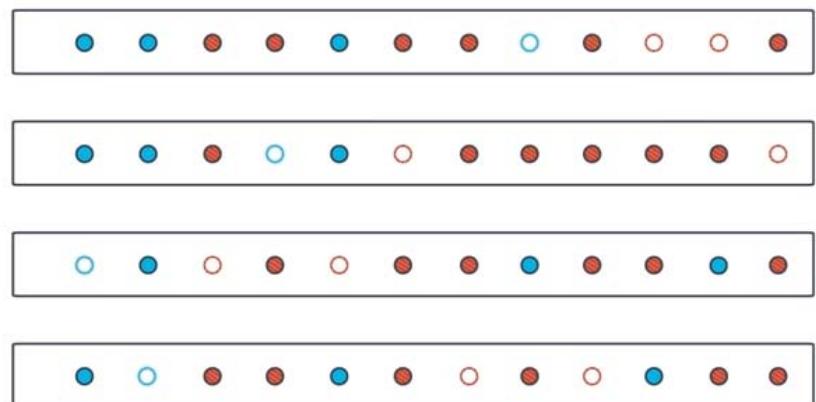
- ▶ Problem: Training Dataset gets too small due to test and validation data
- ▶ Approach:
  - ▶ Randomize data
  - ▶ Break data into  $k$  buckets
  - ▶ Train model  $k$  times, each time using a different bucket as set containing test+validation data
  - ▶ Average the results to get a final model

## Cross validation in sklearn

```
from sklearn.model_selection import KFold
kf = KFold(12, 3, shuffle = True)

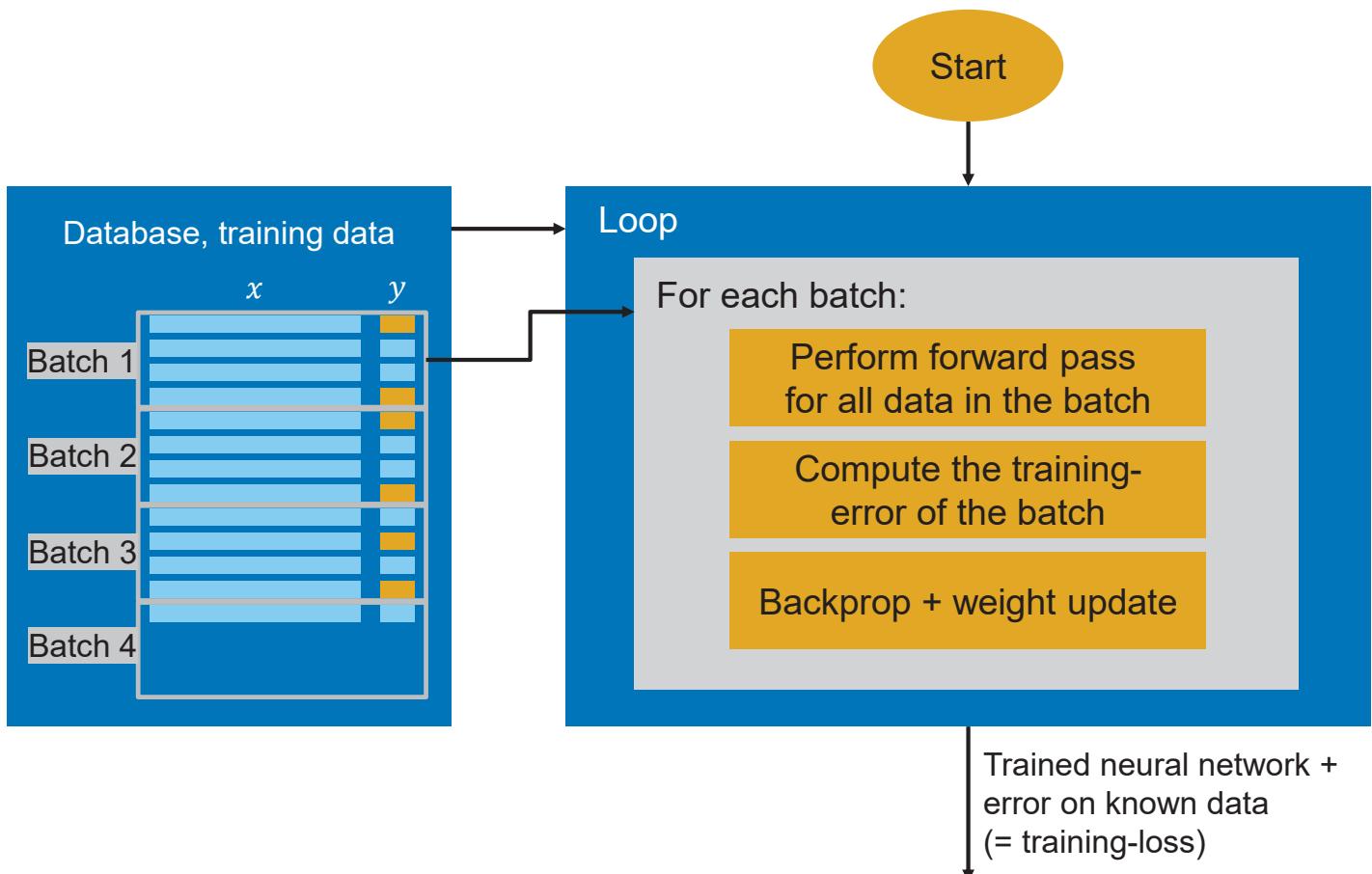
for train_indices, test_indices in kf:
    print train_indices, test_indices

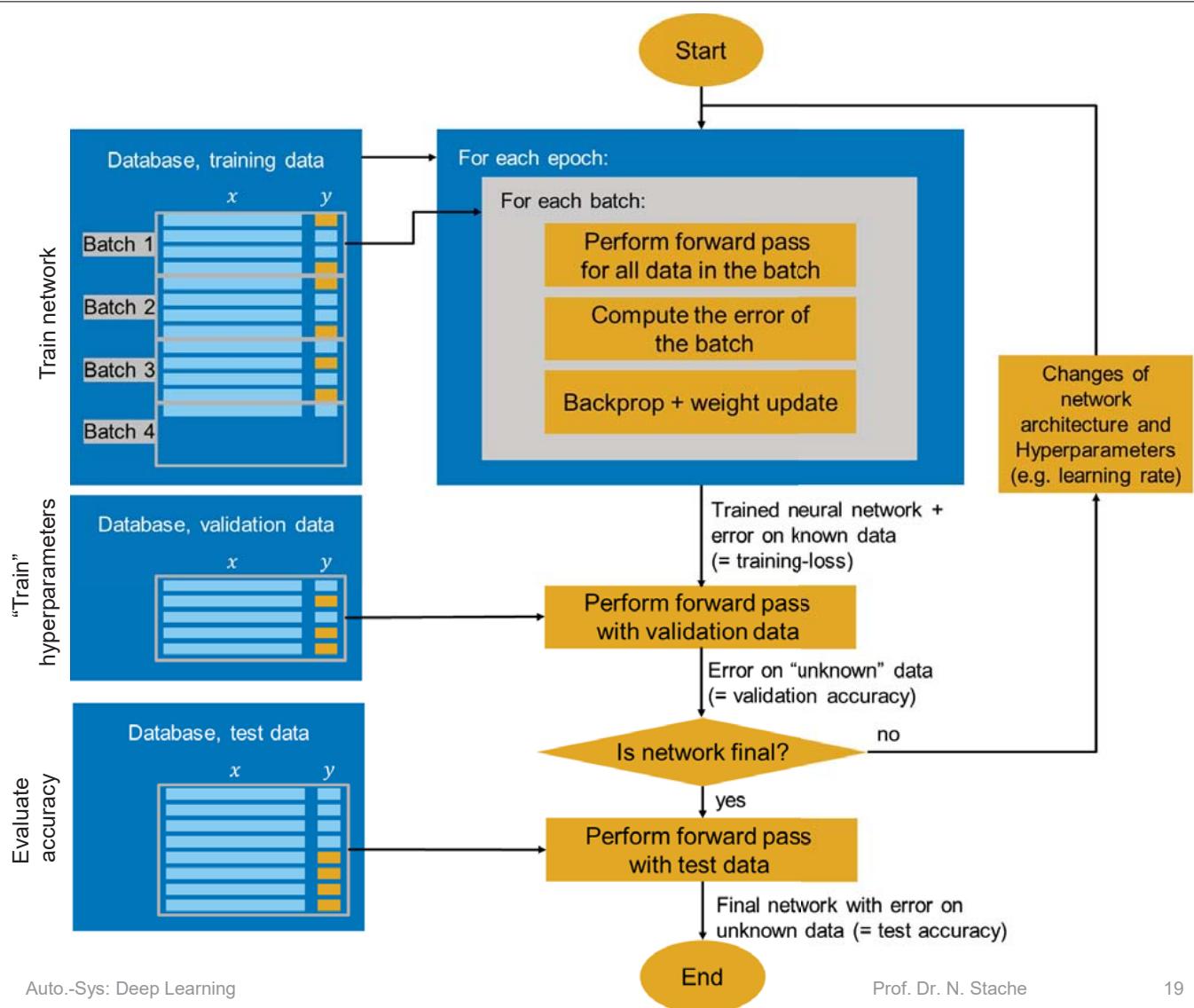
[0 1 2 3 4 5 6 8 11] [7 9 10]
[0 1 2 4 6 7 8 9 10] [3 5 11]
[1 3 5 6 7 8 9 10 11] [0 2 4]
[0 2 3 4 5 7 9 10 11] [1 6 8]
```



- ▶ Forward pass and backprop with bunch of data in parallel
  - ▶ Bunch of data == Batch
  - ▶ Batch size == how many input datasets are computed in parallel
  - ▶ Parallel computation can optimally be performed on GPUs
  - High efficiency without memory overload!
  
- ▶ Process:
  - ▶ Shuffled dataset is divided into batches of defined max. size  
Ideally: Data of one batch can represent the entire dataset
  - ▶ Data in a batch is processed in parallel, adaptations of parameters are done after batch is processed
  - Process is known as Stochastic Gradient Descent (SGD)

## General Procedure of Training a Network





## Metrics Classification Models

Classification done  
by neural network

		Positive:	Negative:
		True Positive (TP)	False Negative (FN)
How it is in reality	Positive:		
	Negative:	False Positive (FP)	True Negative (TN)

► Accuracy:

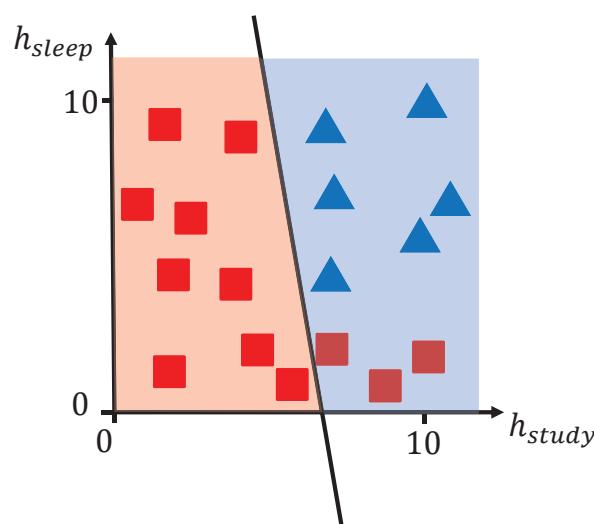
$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

		Diagnosis	
		Diagnosed sick:	Diagnosed healthy:
How it is in reality	Sick:	1000	200
	Healthy:	800	8000

► Accuracy:

$$\text{Accuracy} = \frac{1000 + 8000}{10000} = 90\%$$

► Quiz: What is the accuracy of the model below?



# Metrics

## Classification Models

### ► Precision:

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

### ► Recall:

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

### ► Example:

- high recall: algorithm recognizes almost all obstacles
- Low recall: algorithm misses many obstacles
- High precision: algorithm detects almost purely obstacles
- Low precision: algorithm detects also many obstacles (clutter), although there are not any

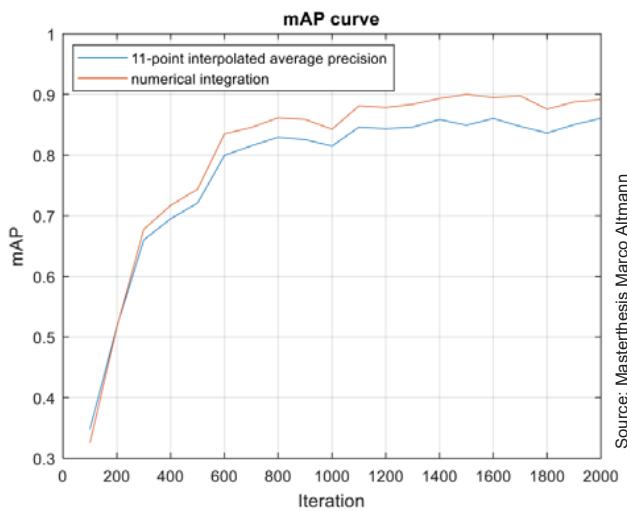
		Detector result:	
		Obstacle on the road:	Road is free
How it is in reality	Obstacle on the road	1000 TP	200 FN
	Road is free	800 FP	8000 TN

# Metrics

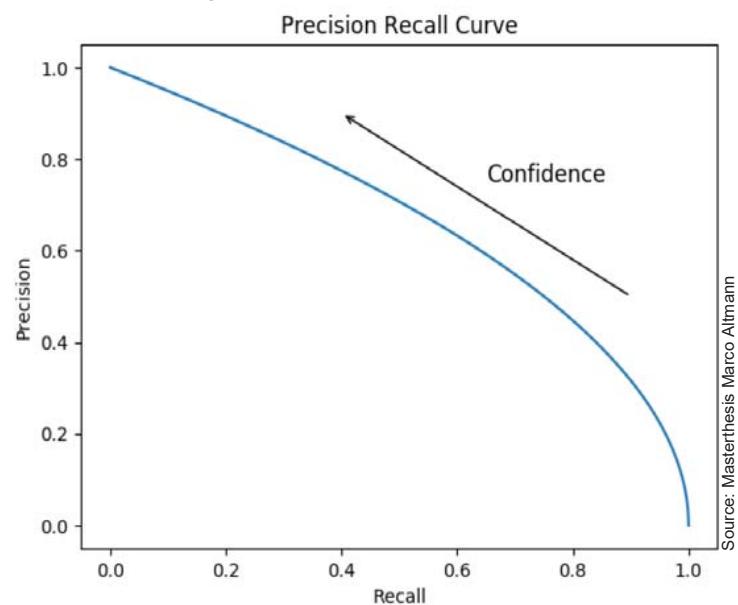
## Classification Models

### ► Precision/Recall curve:

- Average Precision = Area under curve
- Mean Average Precision = Mean of areas under curves for all classifier's classes



Detect no clutter  
but misses objects



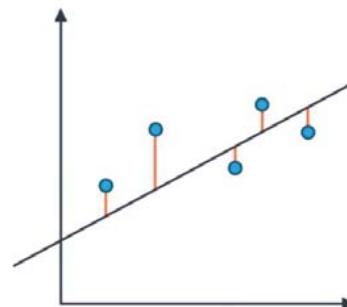
Detect all objects  
but also clutter

- ▶ Mean Absolute Error

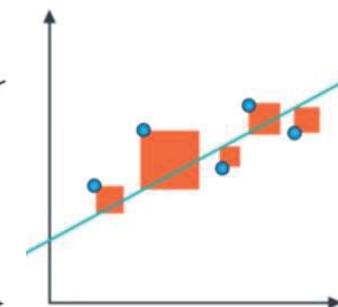
Problem:

not differentiable, bad for gradient descent

Mean absolute error:



Mean squared error:



- ▶ Solution: Mean Squared Error

- ▶ R2 Score

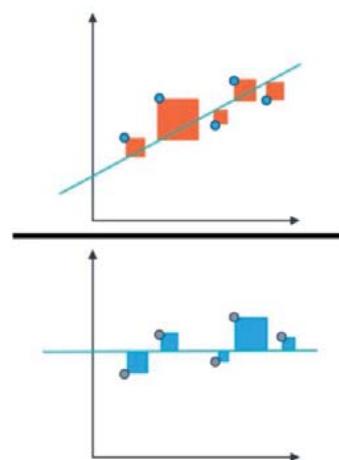
compare regression model to the simplest possible model.

$$R^2 = 1 - \frac{MSE(\text{Regr. model})}{MSE(\text{mean})}$$

Bad regr. model:  $R^2 \rightarrow 0$

Good regr. model:  $R^2 \rightarrow 1$

$$R^2 = 1 -$$



## Outline

- ▶ Learnings from Gardening example
- ▶ Extension to Neural networks including bias term (XOR)
- ▶ Training process, model evaluation and validation
- ▶ Data preparation, student admission example
- ▶ Introduction to TensorFlow

- ▶ Categorical data
  - ▶ contain label values instead of numeric values
  - ▶ often fixed set of values
  - ▶ With natural ordering e.g.: place – first, second third
  - ▶ Without natural ordering, e.g.: Dog, Cat, Bird
- ▶ Non-Categorical data, i.e. numeric data:
  - ▶ Height
  - ▶ Weight
  - ▶ Age

# Dealing with categorical data

- ▶ How can categorical data be processed:
  - ▶ Integer encoding (only for ordinal variables):  
1<sup>st</sup> place → 1, 2<sup>nd</sup> place → 2 ,...  
(however, place 1.34 needs to be interpreted)
  - ▶ One-Hot Encoding  
Assign dummy variable for each category:  
E.g. the variable Pet is decomposed into the variables “Dog”, “Cat”, “Bird”.

# Example: Student admission data

► Mixed dataset:

	admit	gre	gpa	rank
0	0	380	3.61	3
1	1	660	3.67	3
2	1	800	4.00	1
3	1	640	3.19	4
4	0	520	2.93	4
5	1	760	3.00	2
6	1	560	2.98	1
7	0	400	3.08	2
8	1	540	3.39	3
9	0	700	3.92	2

► One-hot encoded dataset:

	admit	gre	gpa	rank_1	rank_2	rank_3	rank_4
0	0	380	3.61	0	0	1	0
1	1	660	3.67	0	0	1	0
2	1	800	4.00	1	0	0	0
3	1	640	3.19	0	0	0	1
4	0	520	2.93	0	0	0	1
5	1	760	3.00	0	1	0	0
6	1	560	2.98	1	0	0	0
7	0	400	3.08	0	1	0	0
8	1	540	3.39	0	0	1	0
9	0	700	3.92	0	1	0	0

## Data normalization, data noise

► Goal: Make training as stable and reliable as possible.

► Question: What effects in the input data can disturb training?

- Features of different scales (mean and variance)

- Noise in the data

- much irrelevant information and only tiny piece of relevant information

- Misclassifications e.g. due to human error

- Data which is not representative for the task, e.g. if training set contains images of happy people + sunshine and sad people + rain it is not clear if the network responds to sun/rain or sadness / happiness

Subtract mean,  
normalize to variance

Preprocess data to  
remove outliers and  
reduce noise

Data-augmentation

- ▶ Check out the student admission example
- ▶ Optional / Homework:  
Check out the sentiment analysis by Andrew Trask

## Outline

- ▶ Learnings from Gardening example
- ▶ Extension to Neural networks including bias term (XOR)
- ▶ Training process, model evaluation and validation
- ▶ Data preparation, student admission example
- ▶ Introduction to TensorFlow → Next lecture



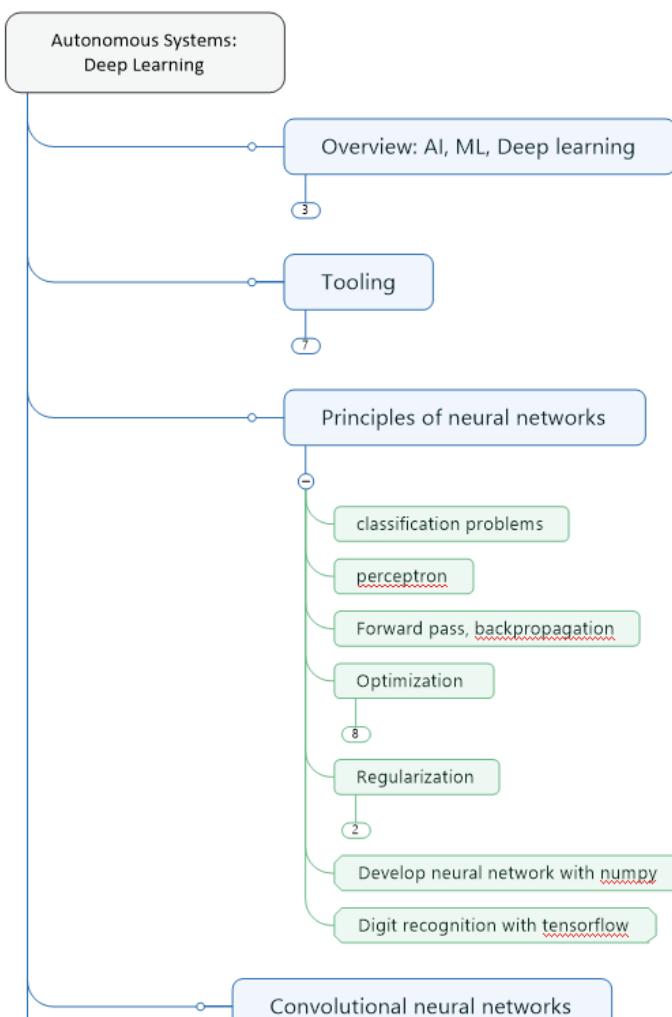
# Autonomous Systems: Deep Learning

## 3. Your first neural network



Prof. Dr.-Ing. Nicolaj Stache  
Heilbronn University of Applied Sciences

### Overview on the course

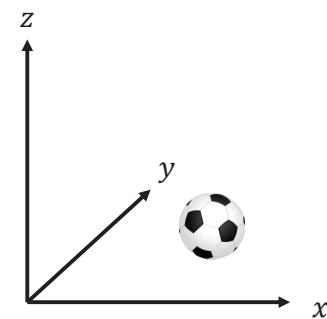


# Learning objectives

- ▶ You have a profound understanding of the terms scalar, vector, matrix, tensor
- ▶ Computations with matrices (element-wise, matrix-multiplication, when is it possible...?)
- ▶ You know how to use numpy
- ▶ You know how a neuron is modelled and a neural network can be composed
- ▶ You can calculate a forward pass through the network
- ▶ You got the idea of backpropagation and can calculate a backward pass through the network
- ▶ You are familiar with terms like: learning rate, inference step, epoch

# Data representation

- ▶ Scalar – single values
  - ▶ Dimensions: 0
  - ▶ e.g. mass of an object: 5 kg
- ▶ Vectors – lists of values
  - ▶ Dimensions: 1
  - ▶ E.g. Position x, y, z of an object
  - ▶ Row Vectors: [4.2 2.1 1.8], length: 3
  - ▶ Column Vectors:  $\begin{bmatrix} 4.2 \\ 2.1 \\ 1.8 \end{bmatrix}$ , length: 3



## ► Matrices

► e.g.: 
$$\begin{bmatrix} 1 & 8 \\ 10 & 6 \\ 20 & 4 \\ 11 & 10 \end{bmatrix}$$

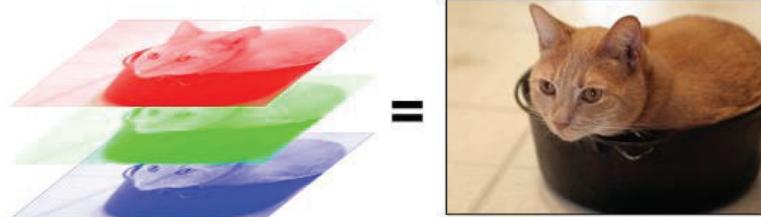
here: four rows, two columns  $\rightarrow 4 \times 2$ -Matrix

► Dimensions: 2

Hours of study	Hours of sleep
1	8
10	6
20	4
11	10

## ► Tensors

- Data collections with more than 2 dimensions
- e.g. RGB-Images



# Data representation

Dimensions	Example	Terminology									
1	<table border="1" style="display: inline-table;"><tr><td>0</td><td>1</td><td>2</td></tr></table>	0	1	2	Vector $1 \times 3$ matrix or 1 <sup>st</sup> order Tensor						
0	1	2									
2	<table border="1" style="display: inline-table;"><tr><td>0</td><td>1</td><td>2</td></tr><tr><td>3</td><td>4</td><td>5</td></tr><tr><td>6</td><td>7</td><td>8</td></tr></table>	0	1	2	3	4	5	6	7	8	Matrix    2 <sup>nd</sup> order Tensor
0	1	2									
3	4	5									
6	7	8									
3	<table border="1" style="display: inline-table;"><tr><td>0</td><td>1</td><td>2</td></tr><tr><td>3</td><td>4</td><td>5</td></tr><tr><td>6</td><td>7</td><td>8</td></tr></table>	0	1	2	3	4	5	6	7	8	3D Array (3 <sup>rd</sup> order Tensor)
0	1	2									
3	4	5									
6	7	8									
N	<table border="1" style="display: inline-table;"><tr><td>...</td><td>...</td></tr><tr><td>...</td><td>...</td></tr></table>	...	...	...	...	ND Array    N <sup>th</sup> order Tensor					
...	...										
...	...										

## ► In textbooks:

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \quad \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}$$

## ► In programming languages:

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \quad \begin{bmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \\ a_{20} & a_{21} & a_{22} \end{bmatrix}$$

Auto.-Sys: Deep Learning

Prof. Dr. N. Stache

7

<https://www.youtube.com/watch?v=F4NSy776X0c>. 17.03.2018

# Matrix refresher

## ► Scalar operation:

$$2 + 3 = 5, \quad \frac{6}{3} = 2$$

## ► Scalar – Matrix operations

Perform the same operation on each element, e.g.

$$2 \cdot \begin{bmatrix} 1 & 3 \\ 4 & 5 \end{bmatrix} = \begin{bmatrix} 2 \cdot 1 & 2 \cdot 3 \\ 2 \cdot 4 & 2 \cdot 5 \end{bmatrix} = \begin{bmatrix} 2 & 6 \\ 8 & 10 \end{bmatrix}$$

$$2 + \begin{bmatrix} 1 & 3 \\ 4 & 5 \end{bmatrix} = \begin{bmatrix} 2+1 & 2+3 \\ 2+4 & 2+5 \end{bmatrix} = \begin{bmatrix} 3 & 5 \\ 6 & 7 \end{bmatrix}$$

#### ► Normalize gray values of an image:

$$\begin{bmatrix} 143 & 150 & 204 & \cdots & 29 \\ 140 & 149 & 200 & \cdots & 32 \\ 30 & 35 & 210 & \cdots & 36 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 45 & 54 & 5 & \cdots & 48 \end{bmatrix} / 255 = \begin{bmatrix} 0.5608 & 0.5882 & 0.8000 & \cdots & 0.1137 \\ 0.5490 & 0.5843 & 0.7843 & \cdots & 0.1255 \\ 0.1176 & 0.1373 & 0.8235 & \cdots & 0.1412 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0.1765 & 0.2118 & 0.0196 & \cdots & 0.1882 \end{bmatrix}$$

18.02.2018

# Element-Wise Matrix operations

- ▶ Elementwise additions:

$$\begin{bmatrix} 1 & 3 \\ 4 & 5 \end{bmatrix} + \begin{bmatrix} 4 & 9 \\ 7 & 8 \end{bmatrix} = \begin{bmatrix} 1+4 & 3+9 \\ 4+7 & 5+8 \end{bmatrix} = \begin{bmatrix} 5 & 12 \\ 11 & 13 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 3 \\ 4 & 5 \end{bmatrix} + \begin{bmatrix} 3 & 5 \\ 2 & 3 \\ 8 & 1 \end{bmatrix} = \text{No}$$

Matrices have to be of the same size.

- ▶ Elementwise multiplication (Hadamard Product  $\circ$ ):  
Matrices have to be of the same size.

$$\begin{bmatrix} 1 & 3 \\ 4 & 5 \end{bmatrix} \circ \begin{bmatrix} 4 & 9 \\ 7 & 8 \end{bmatrix} = \begin{bmatrix} 1 \cdot 4 & 3 \cdot 9 \\ 4 \cdot 7 & 5 \cdot 8 \end{bmatrix} = \begin{bmatrix} 4 & 27 \\ 28 & 40 \end{bmatrix}$$

# Matrix product

- ▶ Matrix product (row x column)  
# columns of first matrix = # rows of the second matrix  
dot product between rows of 1<sup>st</sup> matrix and columns of 2<sup>nd</sup> matrix

$$\underbrace{\begin{bmatrix} 3 & 5 & 2 \\ 4 & 1 & 3 \end{bmatrix}}_{\substack{m=2 \\ n=3}} \cdot \begin{bmatrix} 2 & 3 \\ 1 & 6 \\ 4 & 5 \end{bmatrix} = \begin{bmatrix} 6+5+8 & 9+30+10 \\ 8+1+12 & 12+6+15 \end{bmatrix} = \begin{bmatrix} 19 & 49 \\ 21 & 33 \end{bmatrix}$$

2 3 3

Has to be identical

Dimension of the resulting matrix

- ▶ Matrix product is not commutative:  $A \cdot B \neq B \cdot A$
- ▶ Data in the **left** matrix **should be** arranged as **rows**, while data in the **right** matrix **should be** arranged as **columns**.

- Matrix multiplication with incompatible shape:

$$\begin{bmatrix} 3 & 1 \\ 2 & 7 \\ 5 & 6 \end{bmatrix} \cdot \begin{bmatrix} 1 & 8 \\ 10 & 6 \\ 20 & 4 \\ 11 & 10 \end{bmatrix} = \text{🚫}$$

$3 \times \boxed{2} \times 2$

- Use transpose of 2<sup>nd</sup> matrix:

$$\begin{bmatrix} 3 & 1 \\ 2 & 7 \\ 5 & 6 \end{bmatrix} \cdot \begin{bmatrix} 1 & 10 & 20 & 11 \\ 8 & 6 & 4 & 10 \end{bmatrix} = \checkmark$$

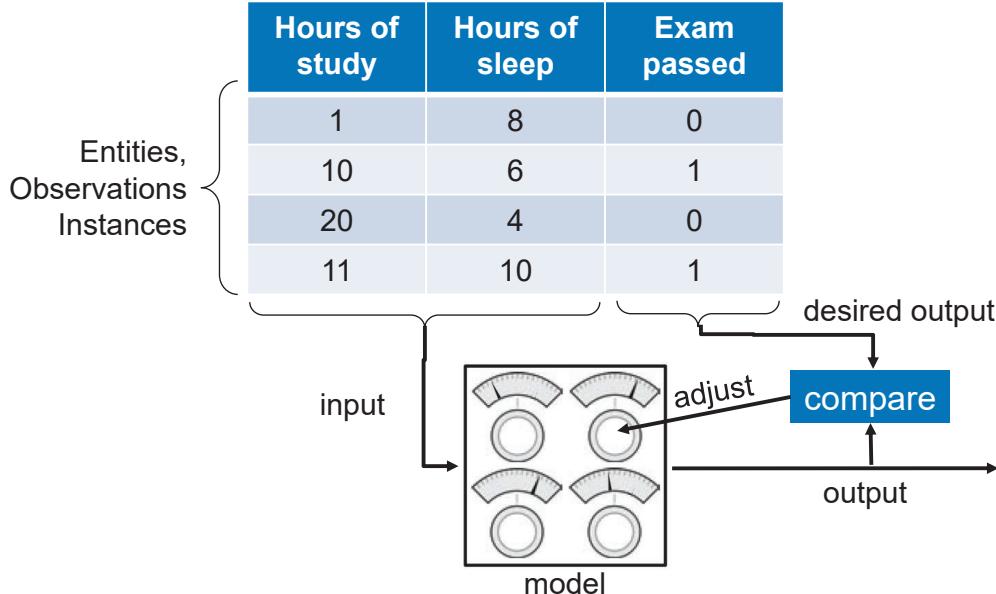
$3 \times \boxed{2} \times 4 = 3 \times 4$

- Note: you have to check if the right data is combined by multiplication

Try it out in Python with Numpy

## Exemplary task:

- Predict result of exam given hours of study and hours of sleep



Auto.-Sys: Deep Learning

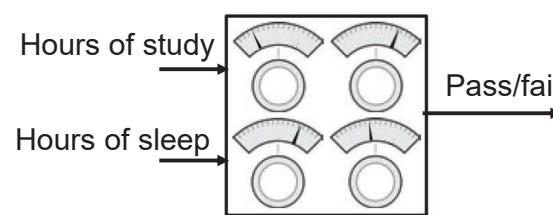
Source: Trask: Grokking Deep Learning, Manning Publications, 2017. Prof. Dr. N. Stache

13

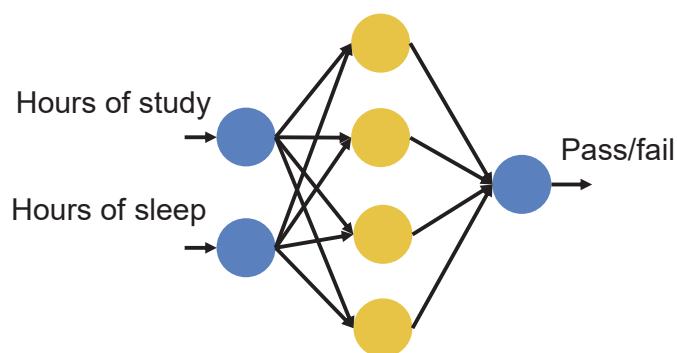
## Exemplary task:

- Predict result of exam given hours of study and hours of sleep

Hours of study	Hours of sleep	Exam passed
1	8	0
10	6	1
20	4	0
11	10	1



- Replace black box by neural network

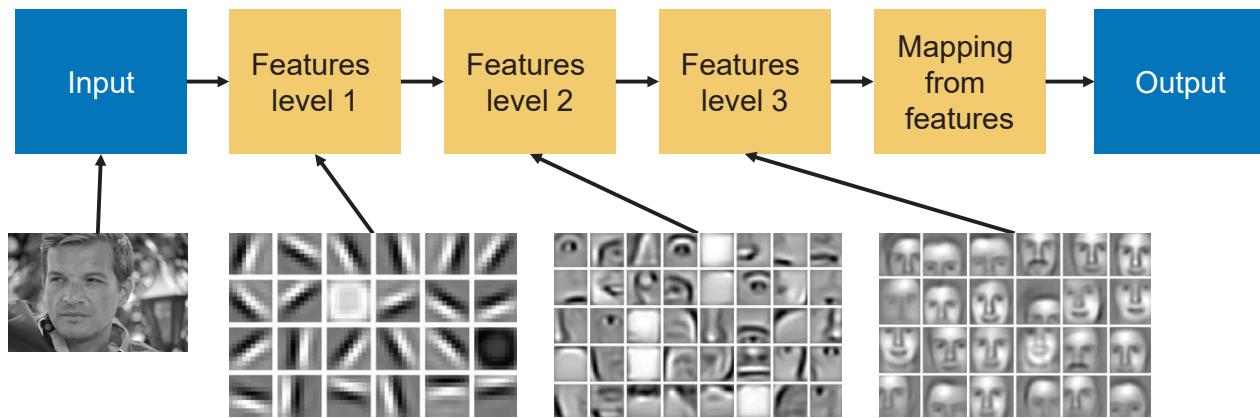


Auto.-Sys: Deep Learning

Source: Trask: Grokking Deep Learning, Manning Publications, 2017. Prof. Dr. N. Stache

14

- ▶ Hierarchical features are determined by deep learning
- ▶ No hand-designed features needed
- ▶ Example:



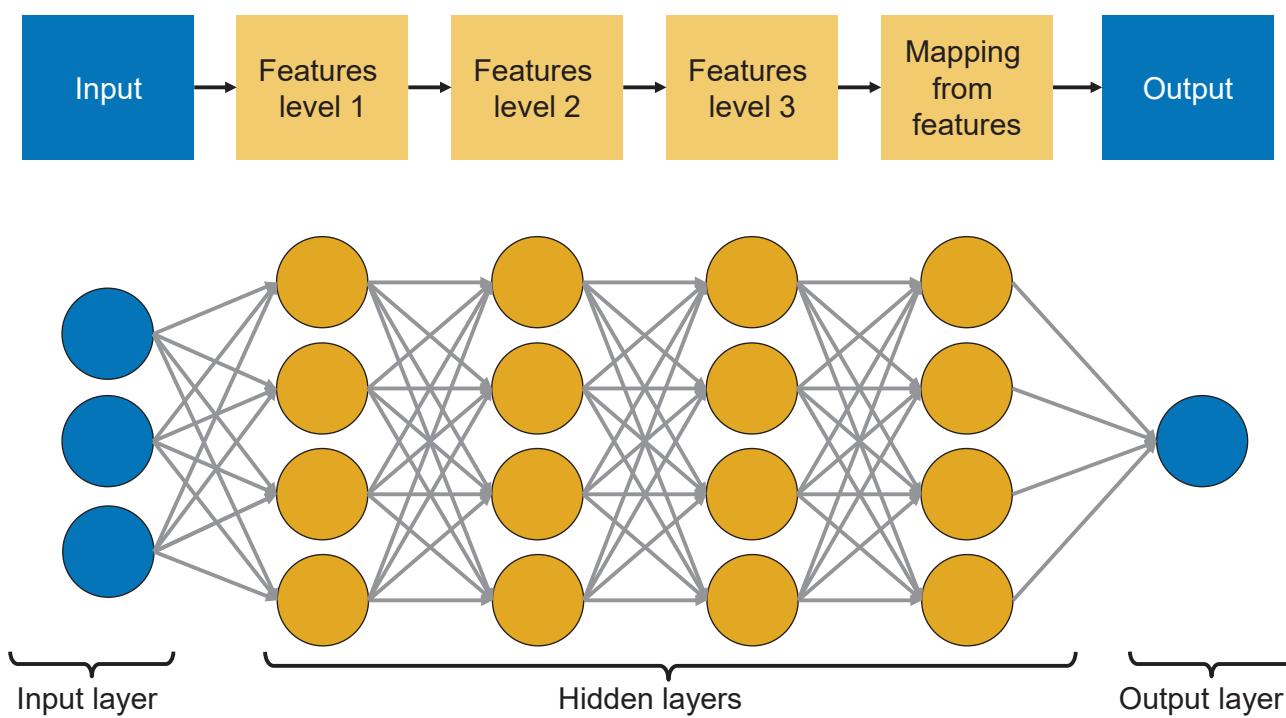
Auto.-Sys: Deep Learning

Source: <https://www.nature.com/news/computer-science-the-learning-machines-1.14481>, Images Andrew Ng, 27.08.2017

Prof. Dr. N. Stache

15

## Realization by a deep neural network

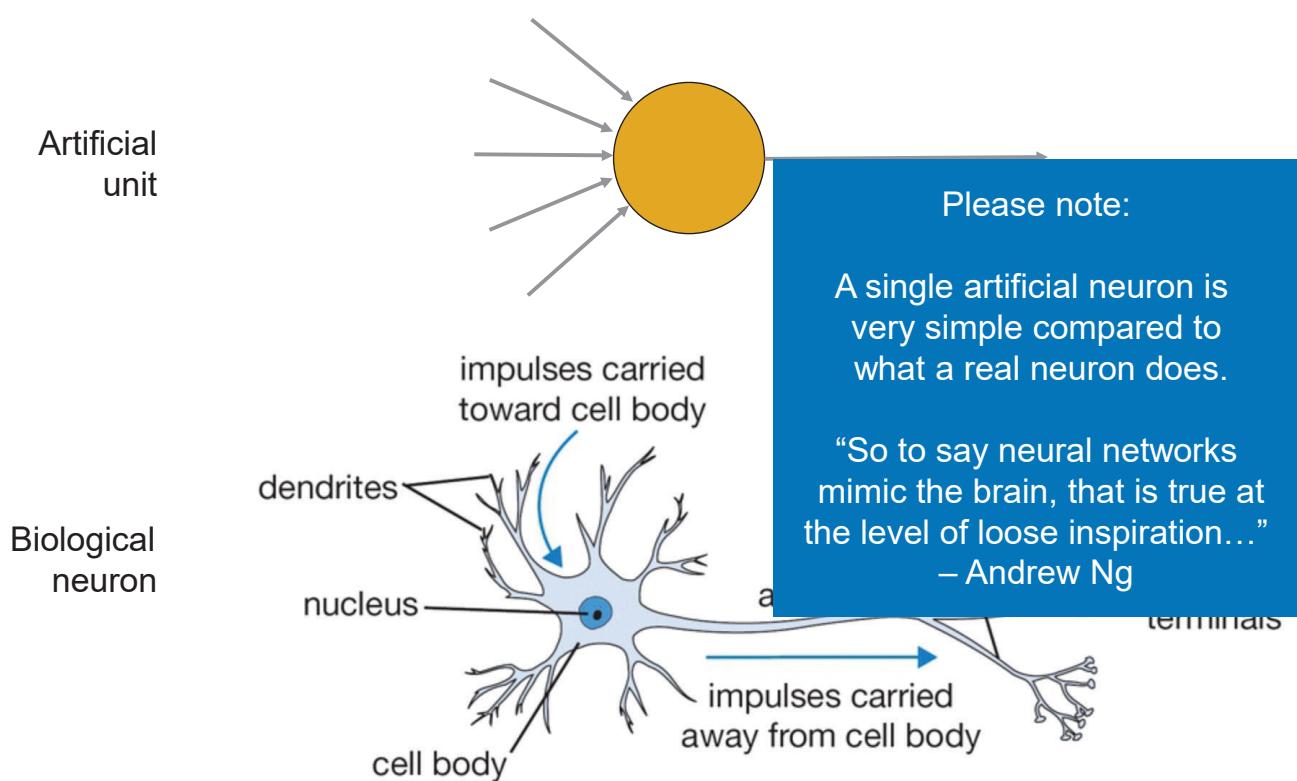


Auto.-Sys: Deep Learning

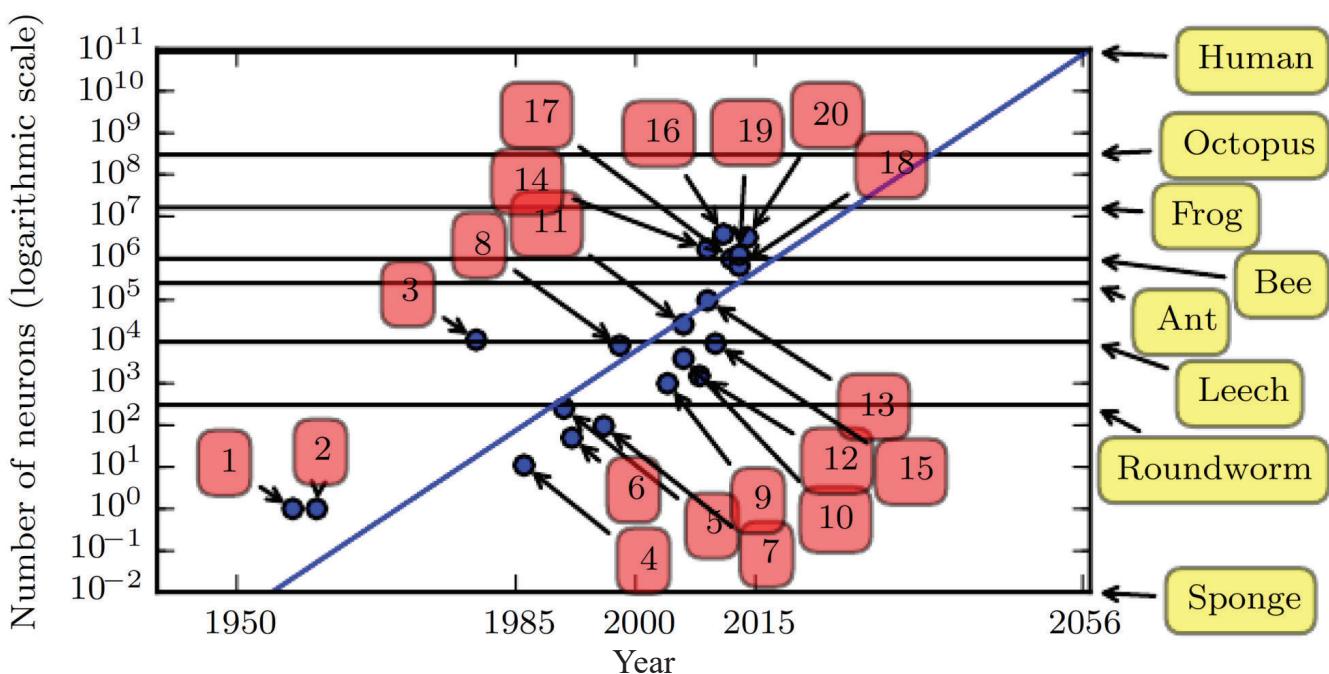
Prof. Dr. N. Stache

16

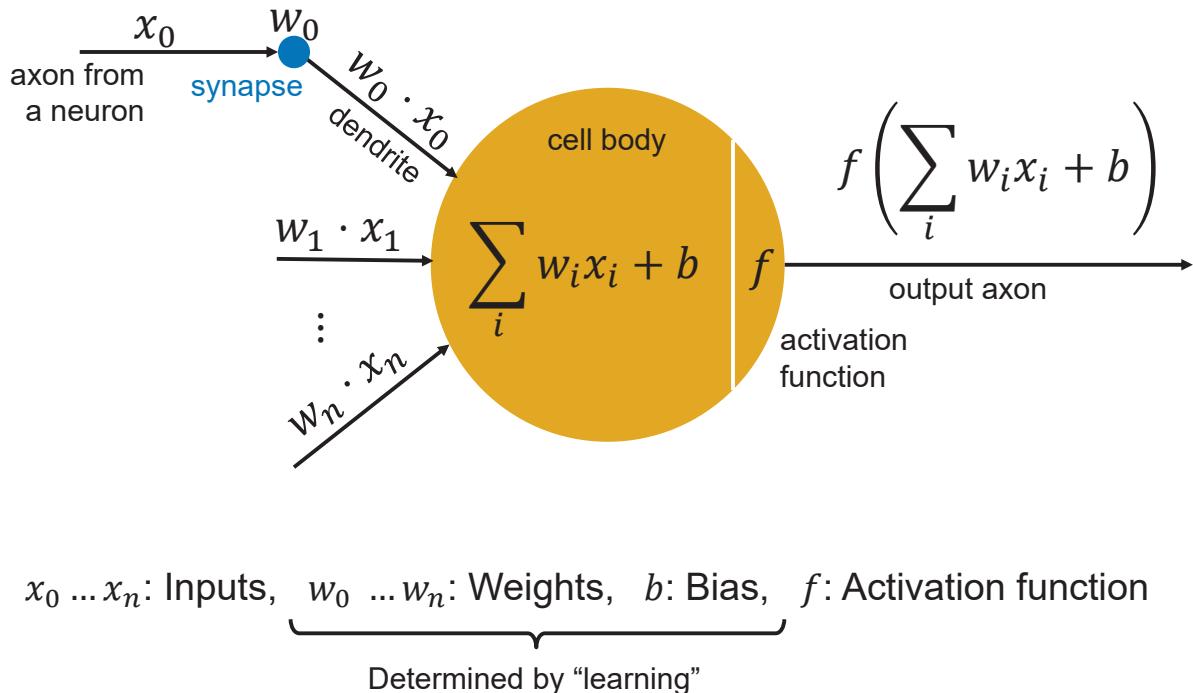
# Biological motivation



## Comparison to biological networks



1: Perceptron 1958, ... 8: LeNet-5 1998, ... 18: AlexNet 2012, ... 20: GoogLeNet 2014



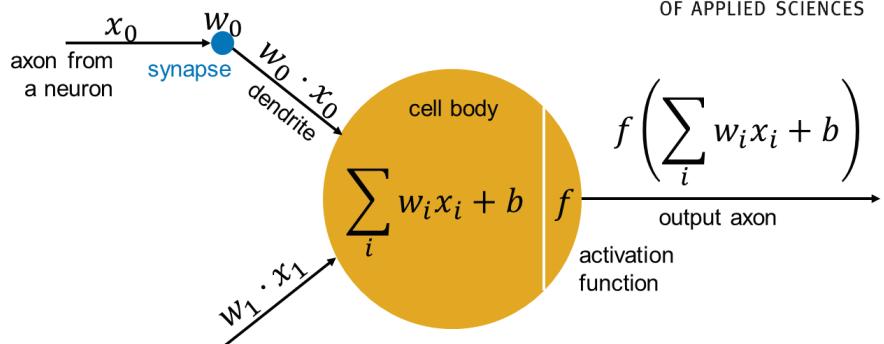
Auto.-Sys: Deep Learning

Source: <http://cs231n.github.io/neuralnetworks1/>, 05.02.2017

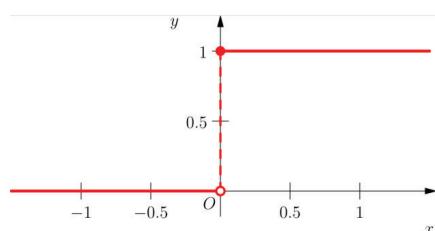
Prof. Dr. N. Stache

19

## Example 1



- $f$ : Step function, Heaviside Function



- $w_0 = 1.0, w_1 = 1.0, b = -0.5$
- $x_0, x_1 \in [0, 1]$
- What is the output?
- → Logical OR is implemented!

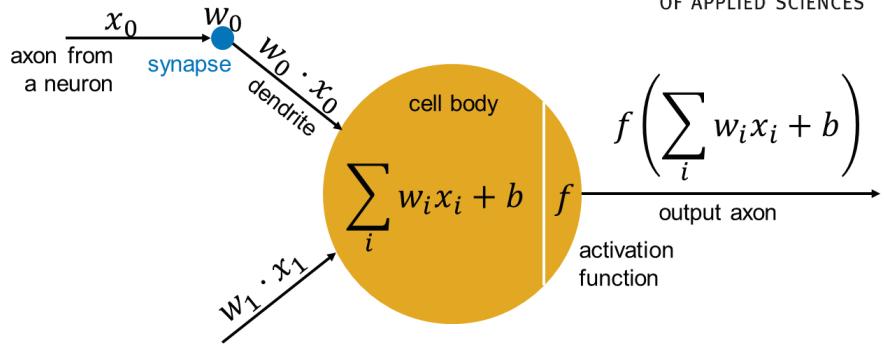
$x_0$	$x_1$	out
0	0	
0	1	
1	0	
1	1	

Auto.-Sys: Deep Learning

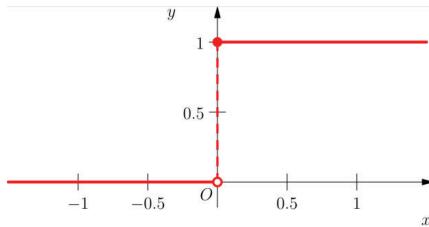
Prof. Dr. N. Stache

20

## Example 2



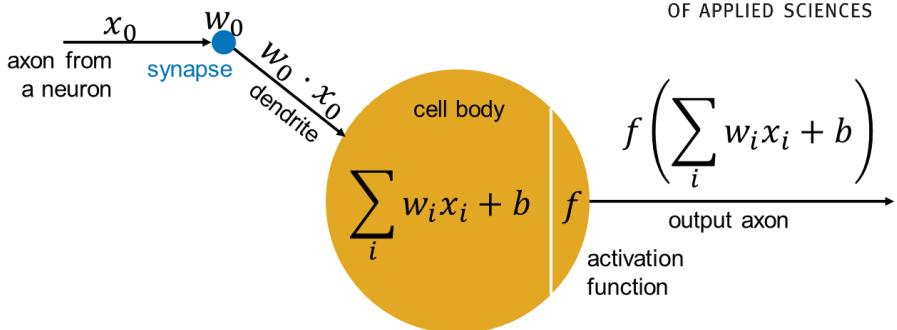
- ▶  $f$ : Step function, Heaviside Function



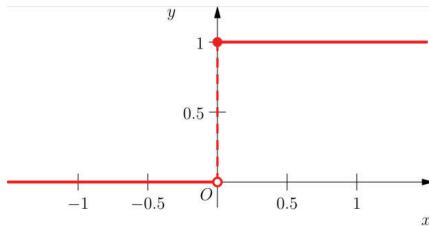
$x_0$	$x_1$	out	$x_0 \text{ AND } x_1$
0	0	0	0
0	1	0	0
1	0	0	0
1	1	1	1

- ▶  $x_0, x_1 \in [0, 1]$
- ▶ How to choose weights and bias for logical AND?
- ▶ Possible solution:  $w_0 = 1.0$ ,  $w_1 = 1.0$ ,  $b = -1.5$

## Example 3



- ▶  $f$ : Step function, Heaviside Function



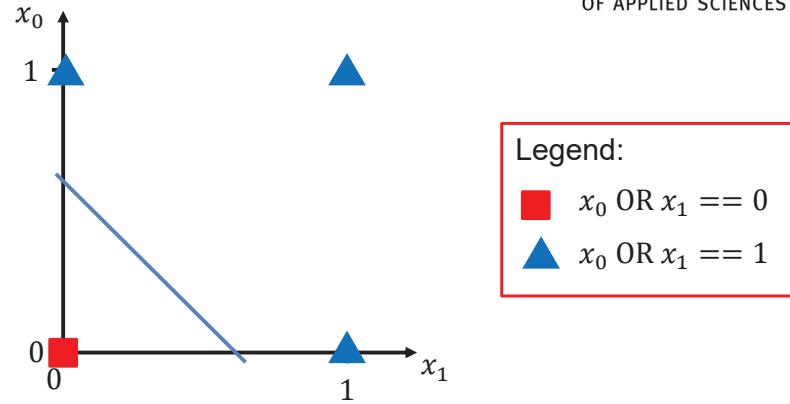
$x_0$	out	$x_0 \text{ NOT } x_1$
0	1	1
1	0	0

- ▶  $x_0 \in [0, 1]$
- ▶ How to choose weight and bias for logical NOT?
- ▶ Possible solution:  $w_0 = -1.0$ ,  $b = 0.5$

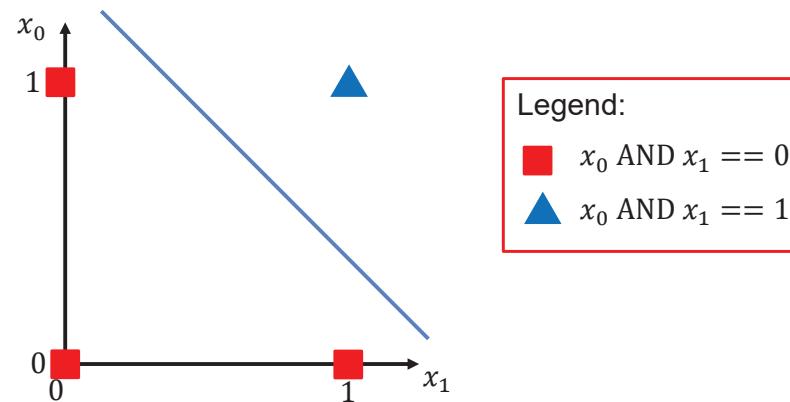
- ▶ Logic functions can be realized by neuron(s)
- ▶ Weights and biases can be tuned for specific behavior
- ▶ With stable parameters:  
neuron's behavior is deterministic
  
- ▶ Note:  
**{AND, NOT}** is a functionally complete set
  - ▶ All Boolean truth-tables can be expressed with this set
  - ▶ Basic unit of processors
  - ▶ What we do with PCs today can also be done with a neuron-based structure

## Scatter-Plots

- ▶ Logical OR:



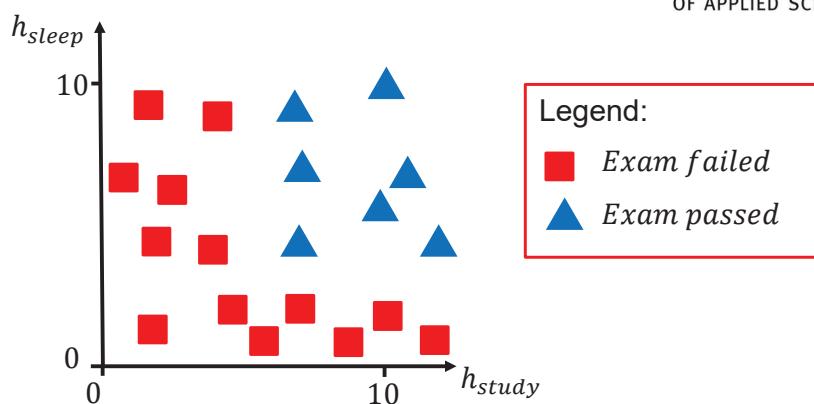
- ▶ Logical AND:



→ Both are examples  
of linear class  
separation

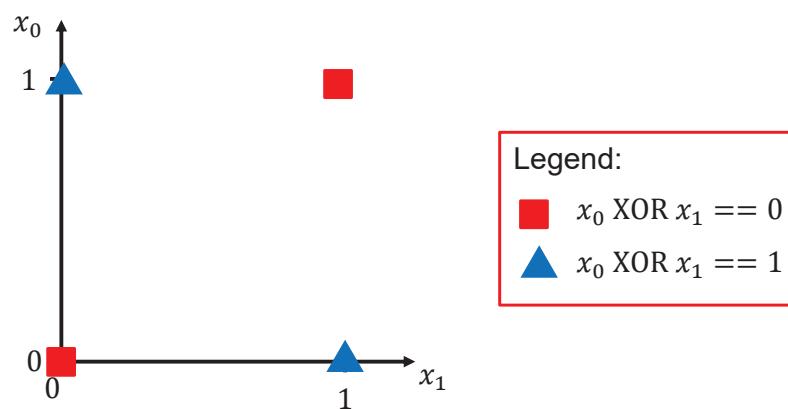
# Scatter-Plots

- ▶ Real-World Data distributions:

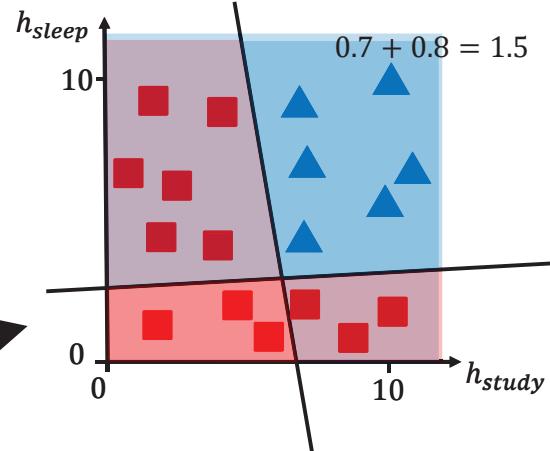
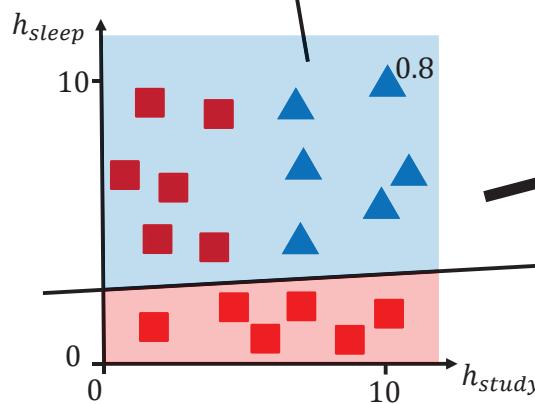
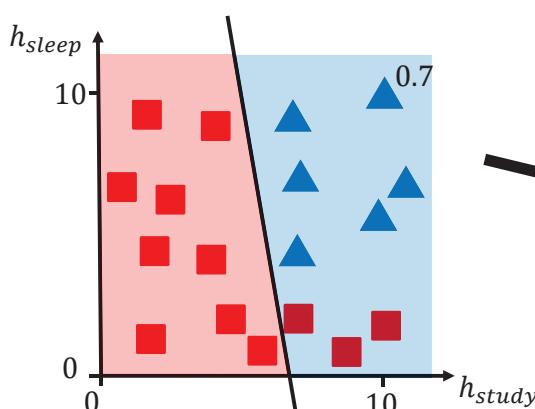


- Only non-linearly separable

(Remark: XOR also nonlinearly separable)

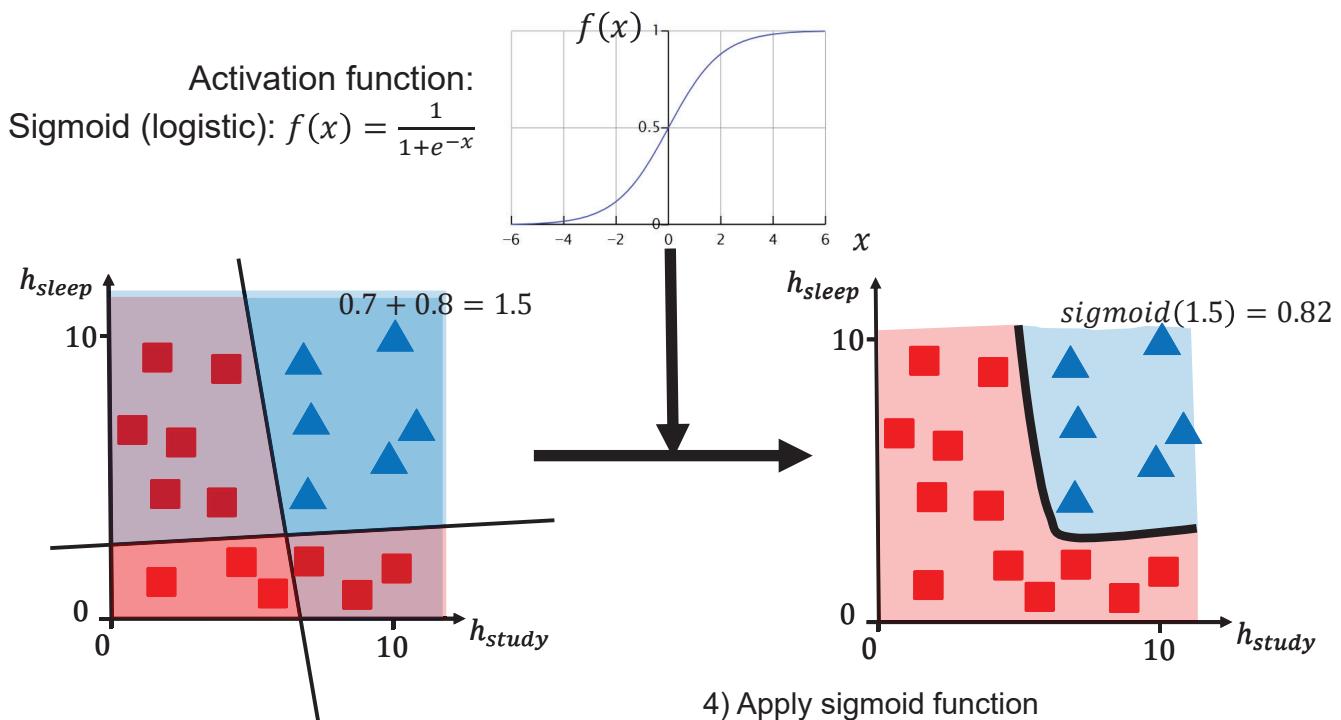


## Idea: Combine linear models to nonlinear



- 1) Compute probability from one model
- 2) Compute probability from other model
- 3) Compute weighted sum

# Idea: Combine linear models to nonlinear



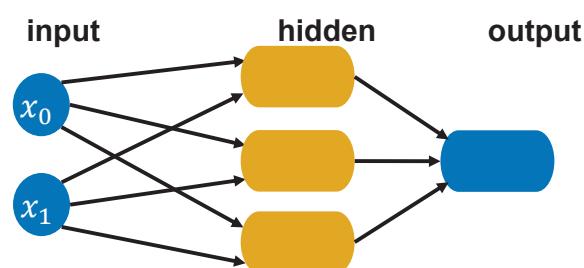
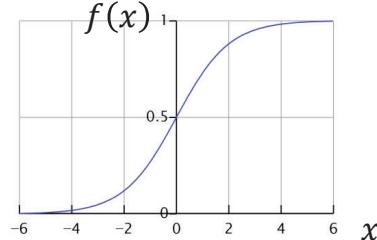
## Determining weights to realize a function

- Given normalized training data:

Input	Output
1.0, 1.0	0
0.8, -0.4	1
0.3, 0.1	1
...	...

- Given activation function:

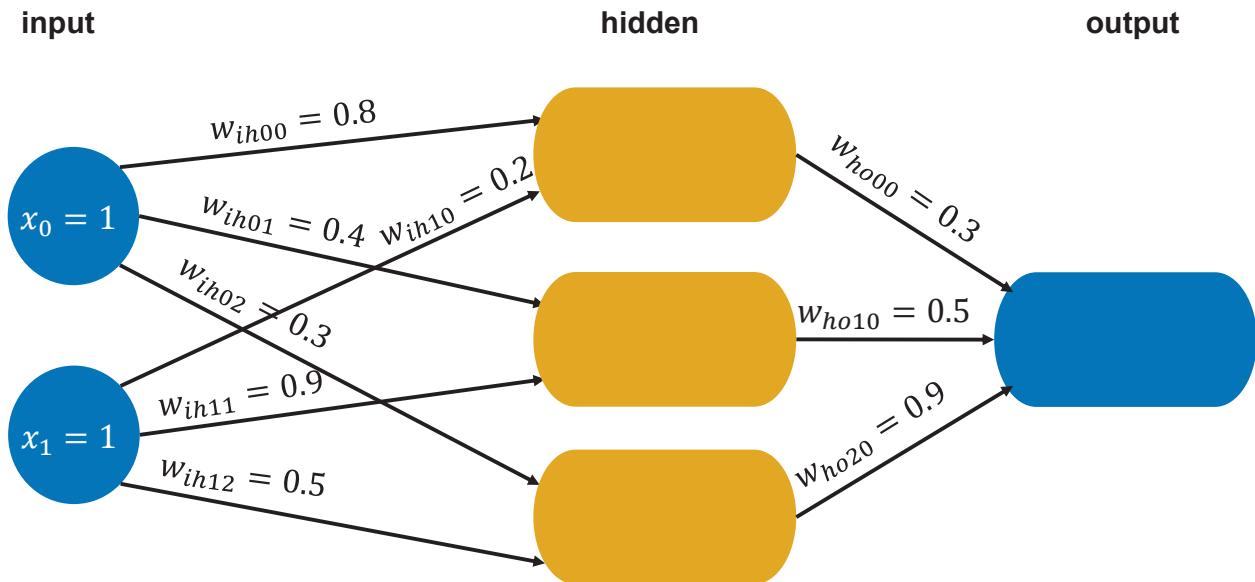
Sigmoid (logistic):  $f(x) = \frac{1}{1+e^{-x}}$



Note:  
 For simplicity, we ignore the bias term in the following explanations

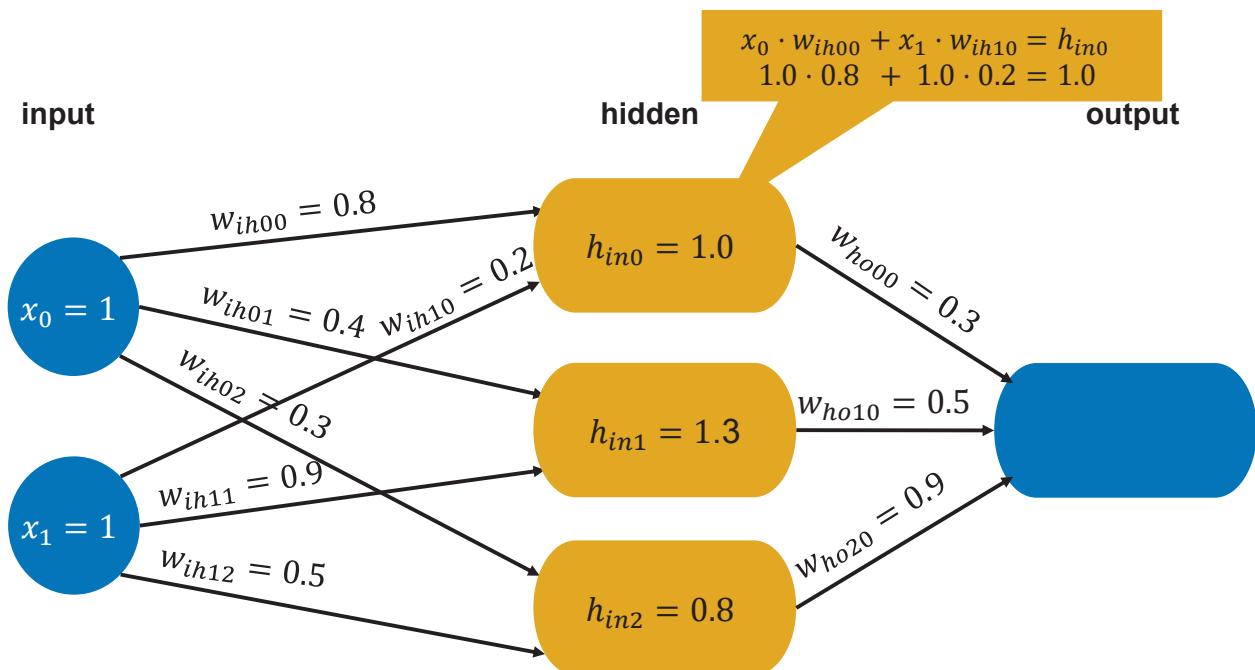
# Determining weights to realize a function

- ▶ Step 1: Initialize the weights by random noise



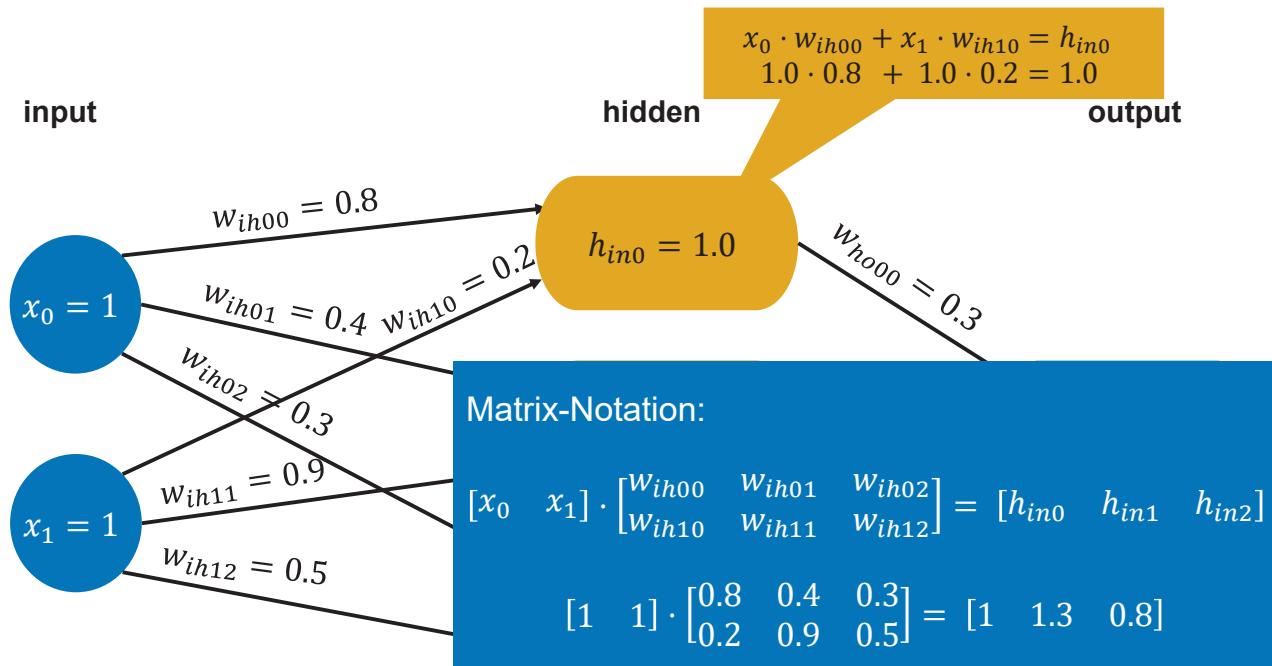
# Determining weights to realize a function

- ▶ Step 2a: Run forward pass (e.g. for first row in the table)



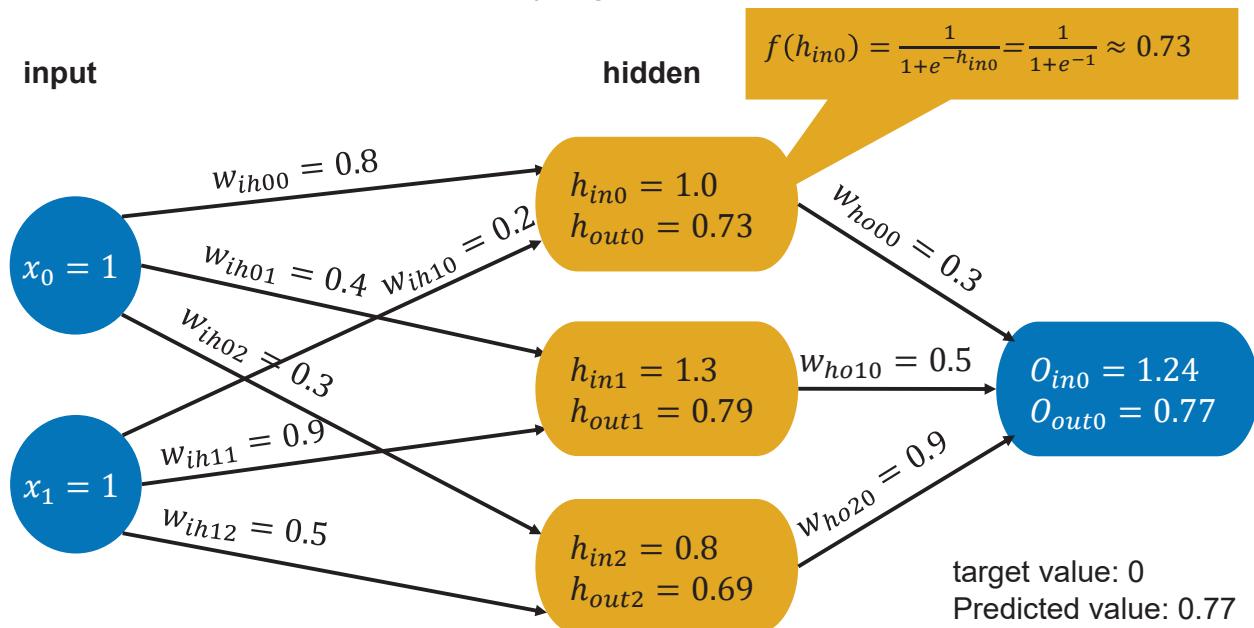
# Determining weights to realize a function

- ▶ Step 2a: Run forward pass (e.g. for first row in the table)



# Determining weights to realize a function

- ▶ Step 2b: Forward pass – apply sigmoid function



# How to improve the result?

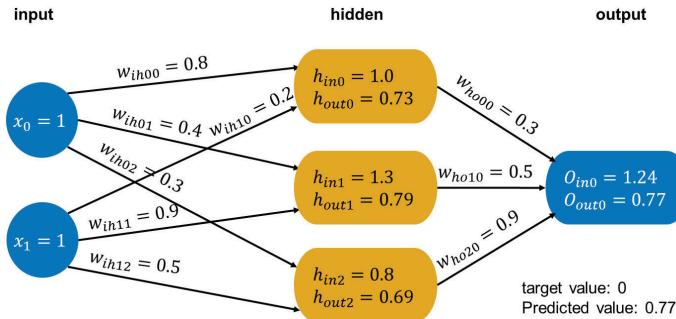
- ▶ Predicted value:  $\hat{y} = 0.77$
- ▶ Target value:  $y = 0$
- ▶ Goal
  - ▶ Modify weights that  $\hat{y}$  gets at least close to  $y$  for all training data
- ▶ Approach
  - ▶ Compute error for all training data
  - ▶ Modify weights or parameters that this error becomes minimal

## Example for error computation

- ▶ Simple idea:  $E = (y - \hat{y})$  Problem:  
- error could be neg. or pos.
  - ▶ Next ideas:  $E = |y - \hat{y}|$  or  $E = (y - \hat{y})^2$  Squared version preferred: Larger errors get more penalized than small ones. Math is simplified later on  
Problem: Needs to consider all training data  $\mu$
  - ▶ Next idea:  $E = \sum_{\mu} [y^{\mu} - \hat{y}^{\mu}]^2$  Ok, if we have one output.  
Otherwise, we need to sum over all outputs  $j$
  - ▶ Final conclusion:  $E = \frac{1}{2} \cdot \sum_{\mu} \sum_j [y_j^{\mu} - \hat{y}_j^{\mu}]^2$
- called SSE, sum of the squared errors

# Determining weights to realize a function

## ► Step 3: Compute the output error



Larger errors are more penalized, simplifies math

- ▶ Output error via sum of squared errors (SSE):

$$E = \frac{1}{2} \cdot \sum_{\mu} \sum_j \left[ y_j^{\mu} - \hat{y}_j^{\mu} \right]^2$$

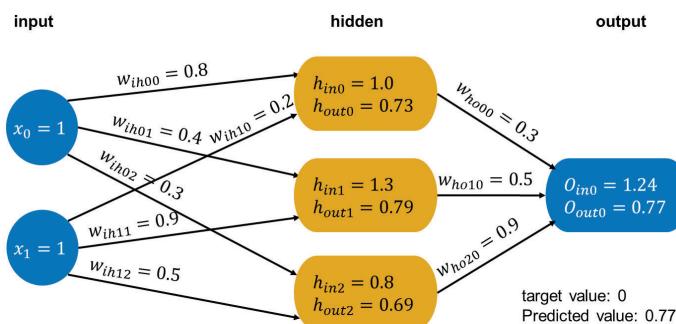
$\hat{y}$  = predicted output  
 $y$  = target value

Sum over all data points  
 $\mu$  = data point index, e.g.  
referring to (0,0), (0,1), ...

Sum over all output units,  
 $j$  = Output unit index

# Determining weights to realize a function

### ► Step 3: Compute the output error



- ▶ Output error via sum of squared errors (SSE):

$$E = \frac{1}{2} \cdot \sum_{\mu} \sum_j [y_j^{\mu} - \hat{y}_j^{\mu}]^2$$

$$E = \frac{1}{2} \cdot \sum_{\mu} \sum_i \left[ y_j^{\mu} - f(\sum_i w_{ij} \cdot x_i^{\mu}) \right]^2$$

## Activation function of the output layer, e.g. sigmoid

Sum over all inputs  
from previous layer, here: 1.2

# Determining weights to realize a function

## ► Step 3: Compute the output error

- Output error via sum of squared errors (SSE):

$$E = \frac{1}{2} \cdot \sum_{\mu} \sum_j [y_j^{\mu} - \hat{y}_j^{\mu}]^2$$

$$E = \frac{1}{2} \cdot \sum_{\mu} \sum_j \left[ y_j^{\mu} - f(\sum_i w_{ij} \cdot x_i^{\mu}) \right]^2$$

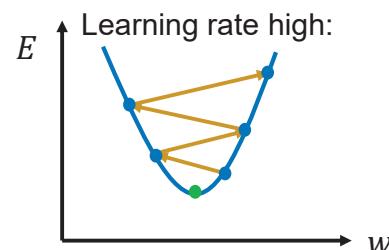
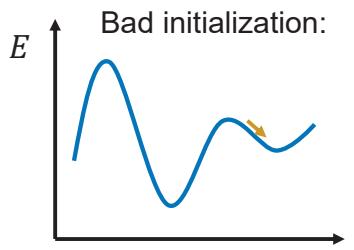
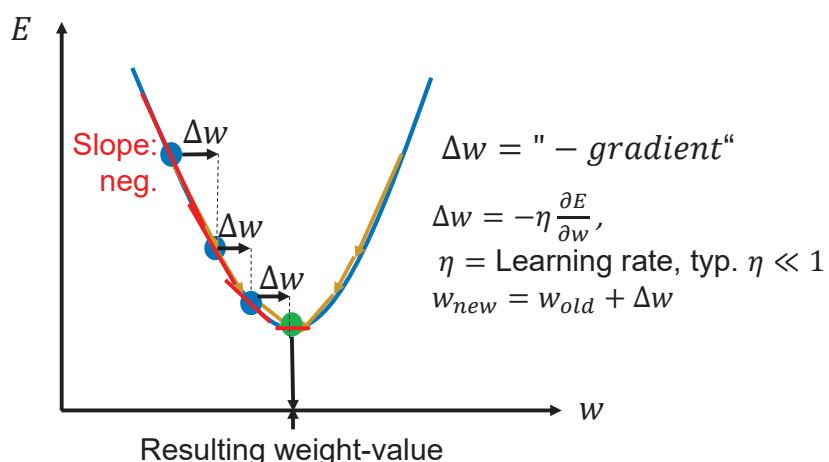
- $E$  is a cost function, depending on the unknown parameters
- Use numerical optimizer to find parameter set which minimizes cost
- Widely used approach: gradient descent

# Determining weights to realize a function

## ► Step 3: Optimizer

- Gradient descent:
- Make steps in "opposite direction of slope"
- Negative slope:  $+ \Delta w$
- Positive slope:  $- \Delta w$

- Pitfalls:



# Computation of the gradient

- ▶ One output, one dataset, one neuron:  $E = \frac{1}{2}(y - \hat{y})^2$
- ▶  $\Delta w_i = -\eta \cdot \frac{\partial E}{\partial w_i}$
- ▶  $\frac{\partial E}{\partial w_i} = \frac{\partial}{\partial w_i} \frac{1}{2} (y - \hat{y})^2 = \frac{\partial}{\partial w_i} \frac{1}{2} (y - \hat{y}(w_i))^2$  Apply chain rule
- ▶  $\frac{\partial E}{\partial w_i} = \frac{2}{2} (y - \hat{y}) \frac{\partial}{\partial w_i} (y - \hat{y}(w_i))$
- ▶  $\frac{\partial E}{\partial w_i} = -(y - \hat{y}) \frac{\partial}{\partial w_i} \hat{y}(w_i)$   $\hat{y}(w_i) = f(h)$ , where  $h = \sum_i w_i x_i$   
 $f(h)$  is the activation function (e.g. sigmoid)
- ▶  $\frac{\partial E}{\partial w_i} = -(y - \hat{y}) \frac{\partial}{\partial w_i} f(\sum_i w_i x_i)$  Apply chain rule again
- ▶  $\frac{\partial E}{\partial w_i} = -(y - \hat{y}) \cdot f'(h) \cdot x_i$   $\frac{\partial}{\partial w_i} \sum_i w_i x_i = x_i$   
 $\frac{\partial}{\partial w_1} [w_1 x_1 + w_2 x_2 + \dots + w_n x_n]$

Auto.-Sys: Deep Learning

Prof. Dr. N. Stache

39

# Computation of the gradient

$$\begin{aligned}
 &= f' \left( \sum_i w_i x_i \right) \\
 \blacktriangleright \frac{\partial E}{\partial w_i} &= -(y - \hat{y}) \cdot \overbrace{f'(h)}^{\text{activation function}} \cdot x_i \\
 \blacktriangleright \Delta w_i &= \eta \cdot \underbrace{(y - \hat{y}) \cdot f'(h)}_{= \delta, \text{ error term}} \cdot x_i
 \end{aligned}$$

Finally, the weight update is:

$$\blacktriangleright w_i = w_i + \eta \delta x_i$$

$x_i$  ... input data (in case of multi layer: input of layer before)  
 $f$  ... activation function (here: sigmoid function)  
 $h$  ...  $\sum_i w_i x_i$   
 $y$  ... target output value  
 $\hat{y}$  ... estimated output value  
 $E$  ... output error  
 $w_i$  ... weight  
 $\Delta w_i$  ... weight update per iteration  
 $i$  ... index running over all inputs  
 $\eta$  ... learning rate (typ.  $\ll 1$ )

Derivative of sigmoid:  
 $f'(x) = \text{sigmoid}'(x)$   
 $= \text{sigmoid}(x) \cdot (1 - \text{sigmoid}(x))$

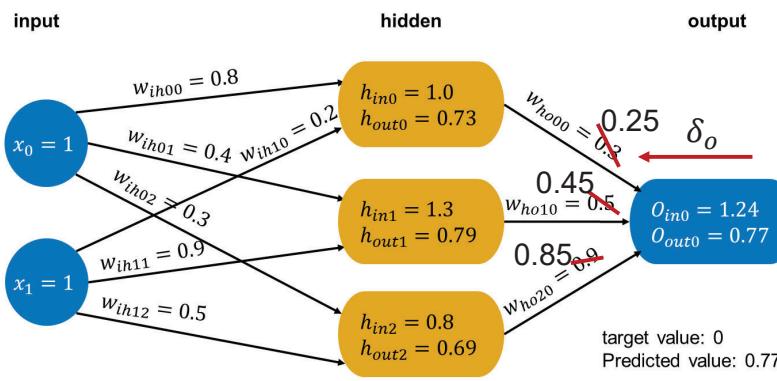
Hint: Explanatory video by Udacity:

<https://www.youtube.com/watch?v=7sxA5Ap8AWM>

## Determining weights to realize a function

- ▶ Step 3: Backpropagation – change weights  $w_{ho00} \dots w_{ho20}$

$$h = \sum w_i x_i$$



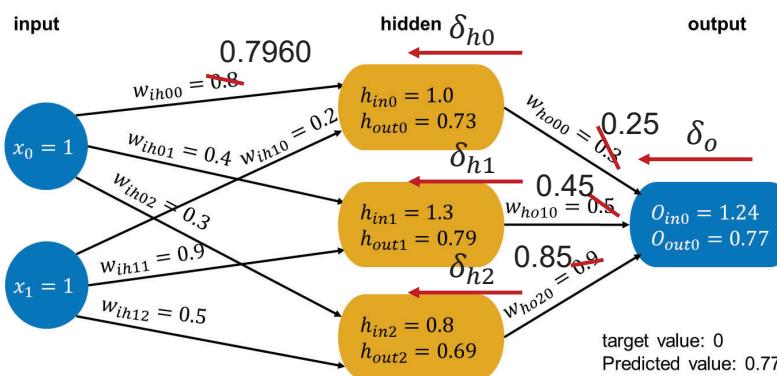
$$\begin{aligned}\delta_o &= (y - \hat{y})f'(O_{in0}) \\ &= (0 - 0.77) \cdot f'(1.24) \\ &= -0.13\end{aligned}$$

- ▶ Learning rate  $\eta = 0.5$
- ▶  $w_{ho} = w_{ho} + \eta \delta_o h_{out}$  „ $x_i$ “
- ▶  $w_{ho00} = 0.3 - 0.5 \cdot 0.13 \cdot 0.73 = 0.25$
- ▶  $w_{ho10} = 0.5 - 0.5 \cdot 0.13 \cdot 0.79 = 0.45$
- ▶  $w_{ho20} = 0.9 - 0.5 \cdot 0.13 \cdot 0.69 = 0.85$

## Determining weights to realize a function

- ▶ Step 3: Backpropagation – change weights  $w_{ih00} \dots w_{ih12}$

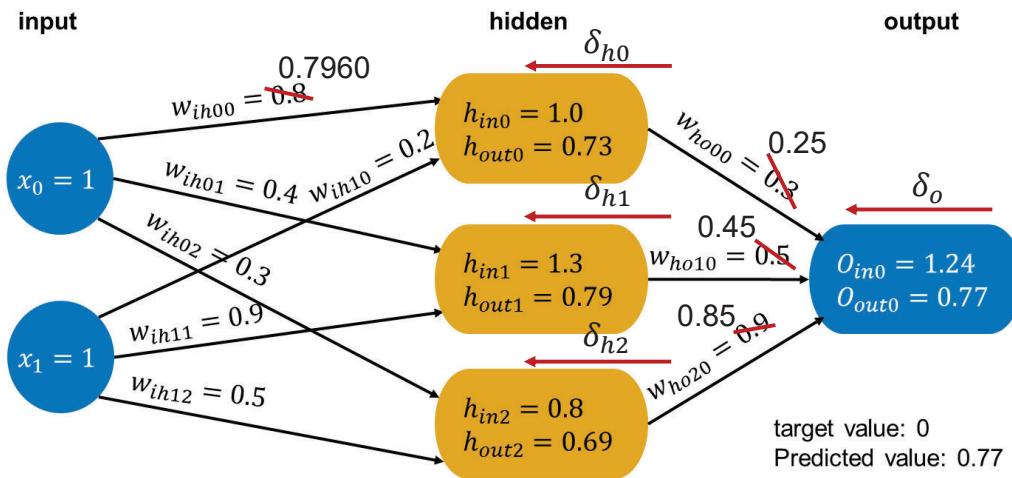
The output error term  $\delta_o$  is now propagated back, using the weighting  $w_{ho}$ . The sum is needed when there is more than one output



- ▶  $\delta_h = \sum (w_{ho} \delta_o) f'(h_{in})$
- ▶  $\delta_{h0} = \delta_o \cdot w_{ho00} \cdot f'(h_{in0}) = -0.1353 \cdot 0.3 \cdot 0.1966 = -0.0079$
- ▶  $\delta_{h1} = \delta_o \cdot w_{ho10} \cdot f'(h_{in1}) = -0.1353 \cdot 0.5 \cdot 0.1683 = -0.0114$
- ▶  $\delta_{h2} = \delta_o \cdot w_{ho20} \cdot f'(h_{in2}) = -0.1353 \cdot 0.9 \cdot 0.2139 = -0.0260$
- ▶  $w_{ih} = w_{ih} + \eta \delta_h x_i$
- ▶  $w_{ih00} = w_{ih00} + \eta \delta_{h0} x_0 = 0.8 - 0.5 \cdot 0.0079 \cdot 1 = 0.7960 \text{ etc.}$

# Determining weights to realize a function

- ▶ Step 4: Backpropagation – change weights
- ▶ Error term  $\delta$  is propagated through the network to update weights



- ▶ Further references on backpropagation lectures from Andrew Karpathy:
- ▶ <https://medium.com/@karpathy/yes-you-should-understand-backprop-e2f06eab496b#.vt3ax2kg9>
- ▶ <https://www.youtube.com/watch?v=59Hbtz7XgjM>

# Try it out in a Jupyter Notebook



## Autonomous Systems: Deep Learning

### 5. Deep Neural Networks



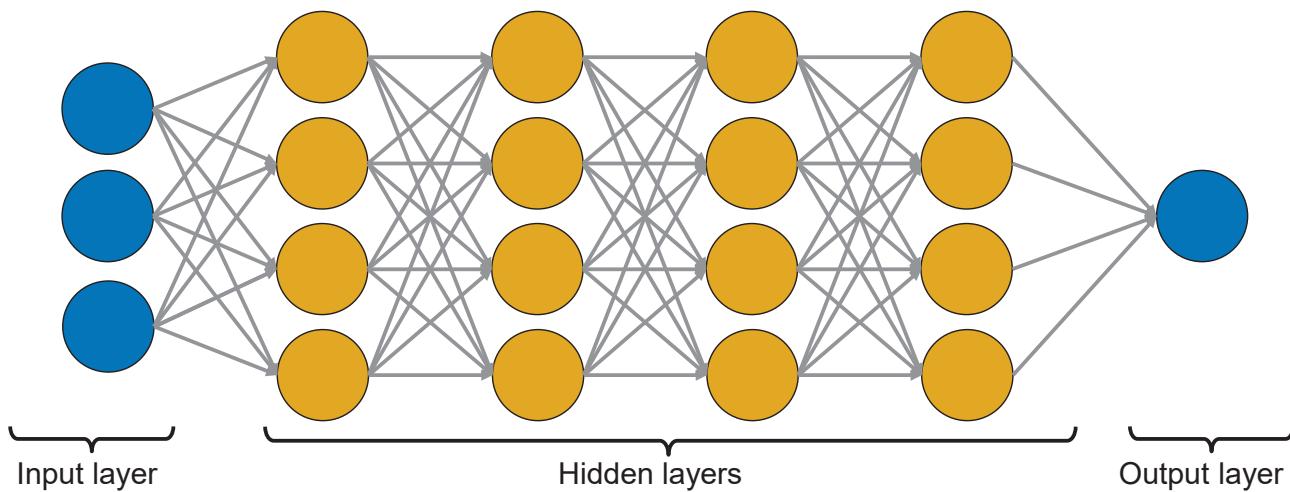
Prof. Dr.-Ing. Nicolaj Stache  
Heilbronn University of Applied Sciences

#### Practical guidance



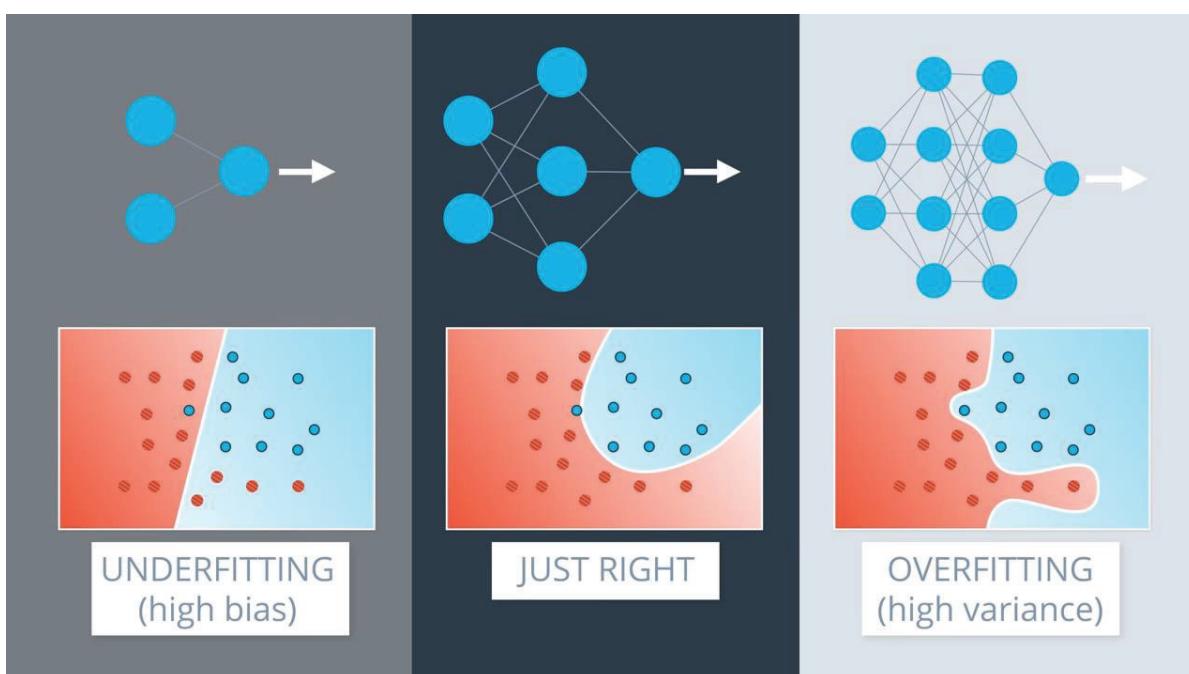
- ▶ Size of network
- ▶ Avoid overfitting
- ▶ Activation functions – downsides of sigmoid
- ▶ Optimizer tuning
- ▶ Neural Networks for classification of several classes
  - ▶ One hot encoding
  - ▶ Softmax
  - ▶ Logits
  - ▶ Cross Entropy loss
- ▶ Introduction to Tensorflow
- ▶ First deep neural network with Tensorflow

- ▶ Just have a higher complexity by increased number of hidden layers



- Expectation: Can model more complex things
- Question: Which complexity do we need for a given task

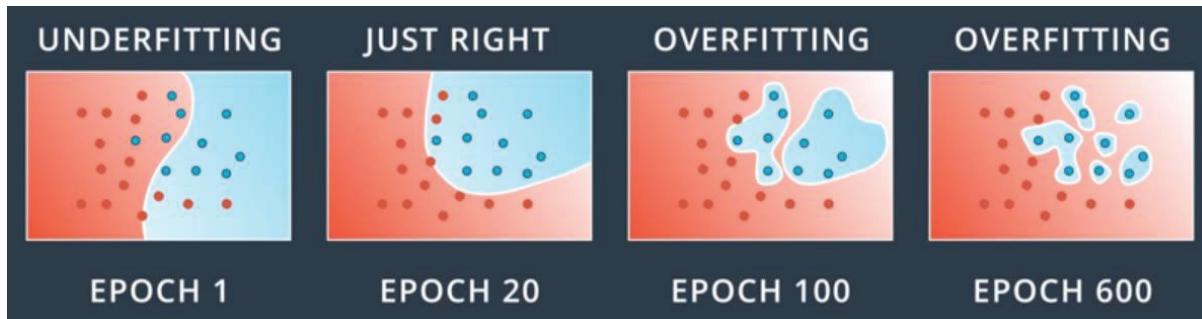
## Overfitting and Underfitting revisited



<https://www.youtube.com/watch?v=x4PfXmSNY>, 17.04.2018

- ▶ Problem: We do not know the right model complexity for the task
- ▶ Approach: Use more complex model + techniques against overfitting

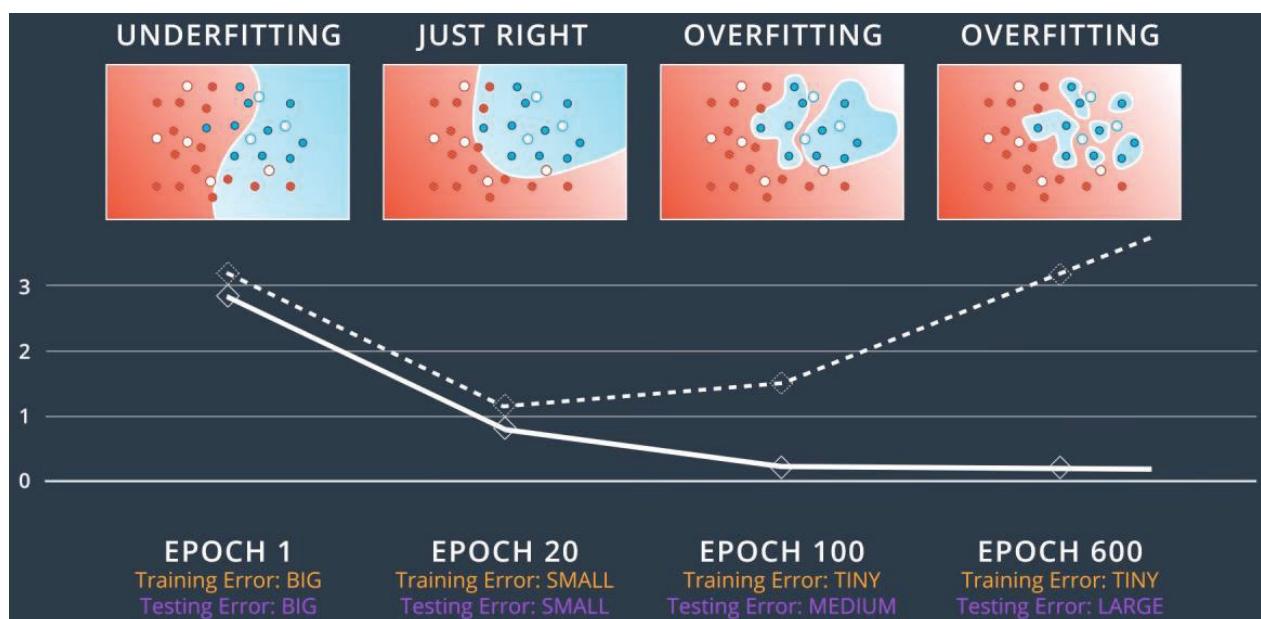
# Train a complex model...



- ▶ How do we identify underfitting and overfitting?

<https://www.youtube.com/watch?v=EnnSOFlJyVcDQ>, 17.04.2018

## Model complexity graph over training epochs



<https://www.youtube.com/watch?v=EnnSOFlJyVcDQ>, 17.04.2018

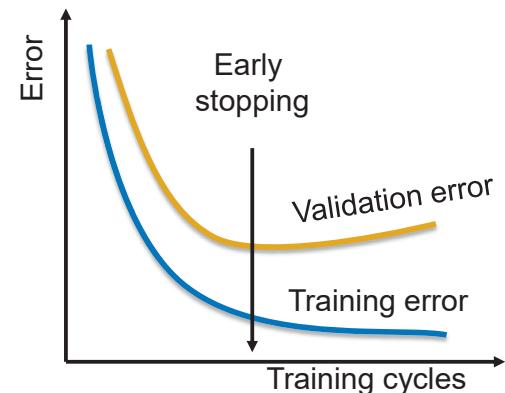
- ▶ Model complexity is virtually increased by the number of training iterations (= epochs)

# How to determine the number of epochs?

## ► Procedure:

- Start training
- Monitor Training loss
- Monitor validation error
- If validation error starts to increase while training loss decreases...
  - stop training
  - optimal model capacity found

## ► Name of this technique: “early stopping”



# Regularization

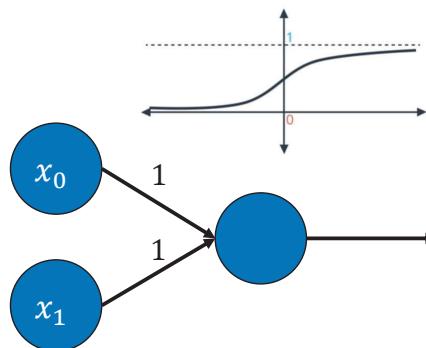
- Goal: Network to perform well also on new input
- Regularization: Strategies to reduce test error, avoid overfitting (on the expense of increased training error)
- Methods:
  - Early stopping
  - Reduce mode's capacity (less neurons, lower model complexity)
  - Penalize large weights (L2 regularization)
  - Weight sharing
  - Label smoothing (= introduce noise to the output)
  - Dropout

# Weight increase vs. result

Activation function: sigmoid

Example 1:

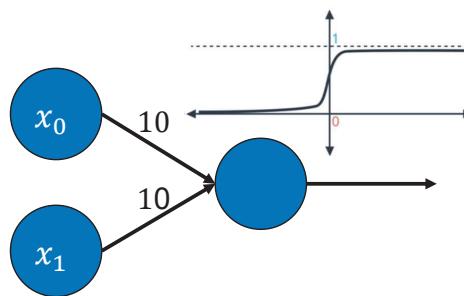
$x_0$	$x_1$	$y$
1	1	1
-1	-1	0



$x_0$	$x_1$	$y$	$\hat{y}$
1	1	1	0.88
-1	-1	0	0.12

Example 2:

$x_0$	$x_1$	$y$
1	1	1
-1	-1	0



$x_0$	$x_1$	$y$	$\hat{y}$
1	1	1	1.00
-1	-1	0	0.00

<https://www.youtube.com/watch?v=ndYnUJrx8xs>, 18.04.2018

Example 2 performs better, but it is overfitting

Steep transition of sigmoid in example 2: bad for gradient descent,  
large errors on test data

## L1, L2 regularization

► Punish high coefficients by modified error function

► L1 regularization:

$$\text{L1 error function} = \text{old error function} + \lambda(|w_1| + \dots + |w_n|)$$

Property:

small features tend to go to 0, favors sparse weights,  
used for feature selection

► L2 regularization:

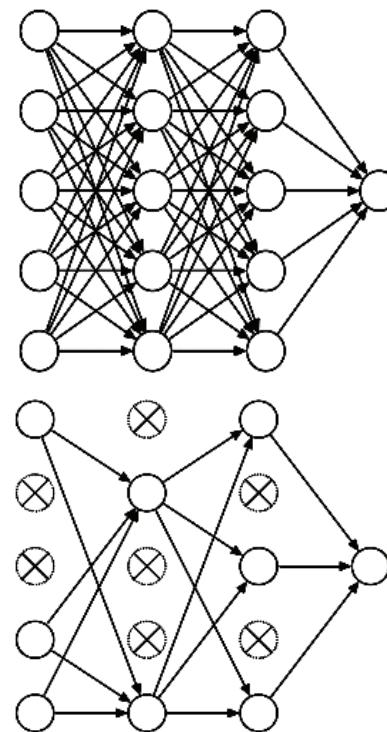
$$\text{L2 error function} = \text{old error function} + \lambda(w_1^2 + \dots + w_n^2)$$

Property:

tries to maintain all elements homogenously small, good for  
training models

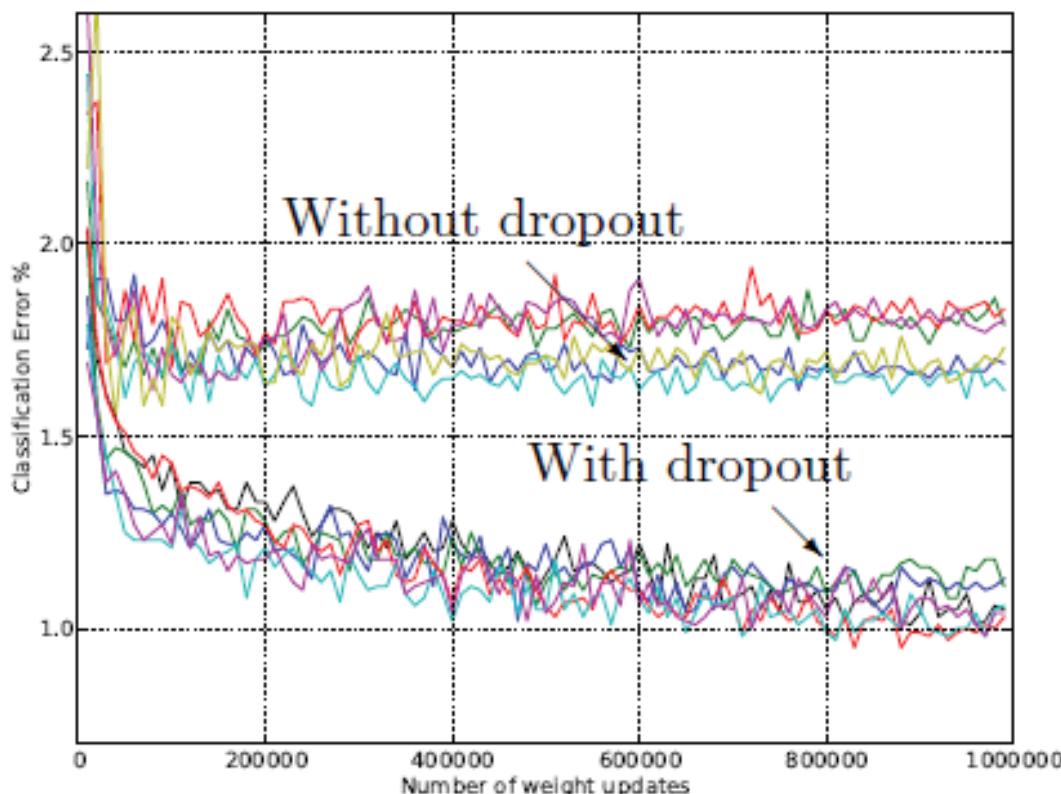
# Dropout

- ▶ Problem: some parts of the network are stronger affected by the training than others
- ▶ Approach: Dropout
  - ▶ Switch off a defined percentage of randomly chosen (non-output-)units during training
  - ▶ Acts like taking consensus of many smaller networks
  - ▶ Network is forced to learn redundant representations
    - increased robustness
    - prevent overfitting



Source: Srivastava, Hinton, Krizhevsky, Sutskever, Salakhutdinov; Dropout: A simple Way to Prevent Neural Networks from Overfitting. J. of Machine learning Research 15 (2014) 1929-1958.

# Dropout



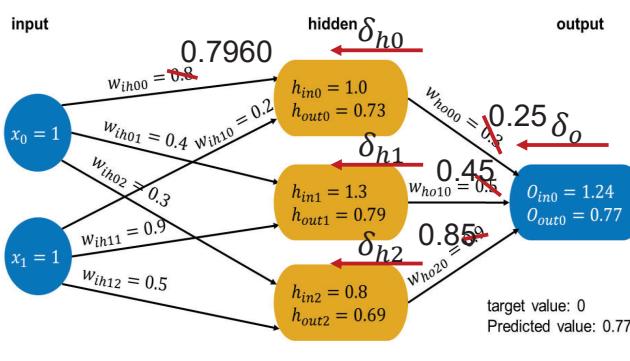
Source: Srivastava, Hinton, Krizhevsky, Sutskever, Salakhutdinov; Dropout: A simple Way to Prevent Neural Networks from Overfitting. J. of Machine learning Research 15 (2014) 1929-1958.

- ▶ Size of network
- ▶ Avoid overfitting
- ▶ Activation functions – downsides of sigmoid
- ▶ Optimizer tuning
- ▶ Neural Networks for classification of several classes
  - ▶ One hot encoding
  - ▶ Softmax
  - ▶ Logits
  - ▶ Cross Entropy loss
- ▶ Introduction to Tensorflow
- ▶ First deep neural network with Tensorflow

## Sigmoid activation function in deep networks

- ▶ Recap backpropagation:

- ▶  $\delta_o = (y - \hat{y})f'(O_{in0})$
- ▶  $\delta_h = \sum(w_{ho}\delta_o)f'(h_{in})$
- ▶  $w_{ih} = w_{ih} + \eta\delta_h x_i$



Backpropagation:

$$1) \begin{cases} ([\cdot] - [\cdot]) \cdot [\cdot] \Rightarrow [\cdot] \\ (y - \hat{y}) \cdot f'(O_{in}) \Rightarrow \delta_o \\ \uparrow \\ \text{sigmoid prime}(O_{in}) \end{cases}$$

$$2) \begin{cases} [\cdot] + [\cdot] \cdot [\cdot] \cdot [\dots]^T \Rightarrow [\cdot] \\ w_{ho} + \eta \cdot \delta_o \cdot h_{out}^T \Rightarrow w_{ho\_new} \end{cases}$$

$$3) \begin{cases} [\cdot] \cdot [\cdot] \cdot \text{sigmoid prime}([\dots]^T) \Rightarrow [\cdot] \\ \delta_o \cdot w_{ho} \cdot \text{sigmoid prime}(h_{in}) \Rightarrow \delta_h \end{cases}$$

$$4) \begin{cases} [\cdot : : \cdot] + [\cdot] \cdot [\cdot] \cdot [\cdot : \cdot] \Rightarrow \underbrace{\cdot}_{w_{in}} \quad \underbrace{\cdot}_{(\eta \cdot \delta_h \cdot x_i)} \quad \underbrace{\cdot}_{\text{Dimensions do not match!}} \\ w_{in} + \eta \cdot \delta_h \cdot x_i \Rightarrow w_{in\_new} \end{cases}$$

Note: We want to create a matrix for an update of  $w_{in}$ , which is composed like this:

$$\begin{bmatrix} \eta \cdot \delta_{h_0} \cdot x_0 & \eta \cdot \delta_{h_1} \cdot x_0 & \eta \cdot \delta_{h_2} \cdot x_0 \\ \eta \cdot \delta_{h_0} \cdot x_1 & \eta \cdot \delta_{h_1} \cdot x_1 & \eta \cdot \delta_{h_2} \cdot x_1 \end{bmatrix}$$

this can be done by:

$$\eta \cdot x_i^T \cdot \delta_h^T$$

$$\begin{cases} [\cdot : : \cdot] + [\cdot] \cdot [\cdot] \cdot [\cdot : \cdot] \Rightarrow [\cdot : : \cdot] \\ w_{in} + \eta \cdot x_i^T \cdot \delta_h^T \Rightarrow w_{in\_new} \end{cases}$$

# Sigmoid activation function in deep networks

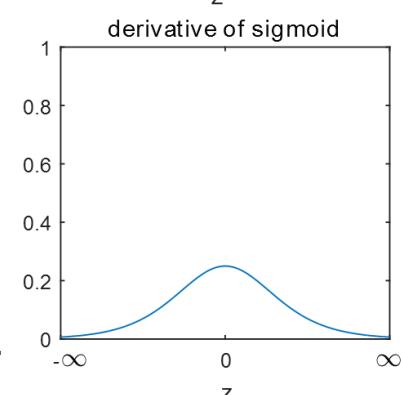
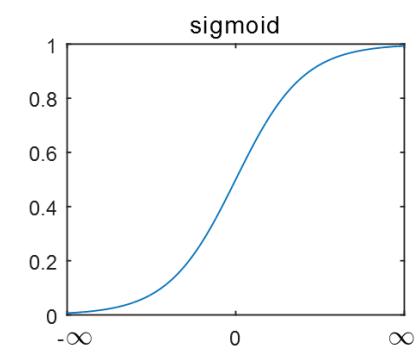
## ► Recap backpropagation:

- $\delta_o = (y - \hat{y})f'(O_{in_0})$
- $\delta_h = \sum(w_{ho}\delta_o)f'(h_{in})$

## ► Problems:

- Neurons never reach 1 or 0  
→ saturation (weights go up)
- For large weights, gradients are small  
→ slow learning
- The more layers, the more  
gradients get multiplied ( $0.25 \cdot 0.25 = 0.0625$ ) .

### ➔ Vanishing gradient problem



# Rectified Linear Unit ReLU activation function

- Alex Krizhevsky (2011):  
Rectified Linear Unit instead of sigmoid function  
 $\sigma(z) = \max(0, z)$

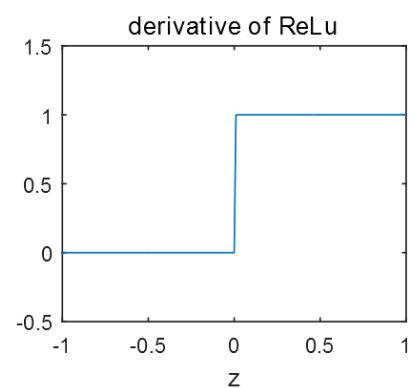
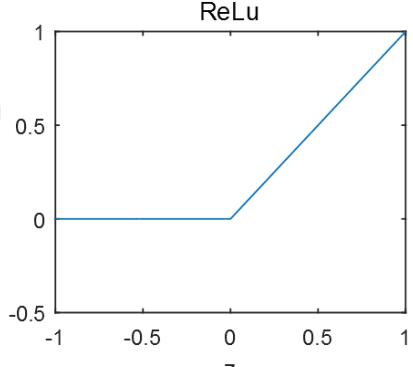
$$\frac{\partial \sigma(z)}{\partial z} = \begin{cases} 0 & \text{if } z \leq 0 \\ 1 & \text{if } z > 0 \end{cases}$$

## ► Properties:

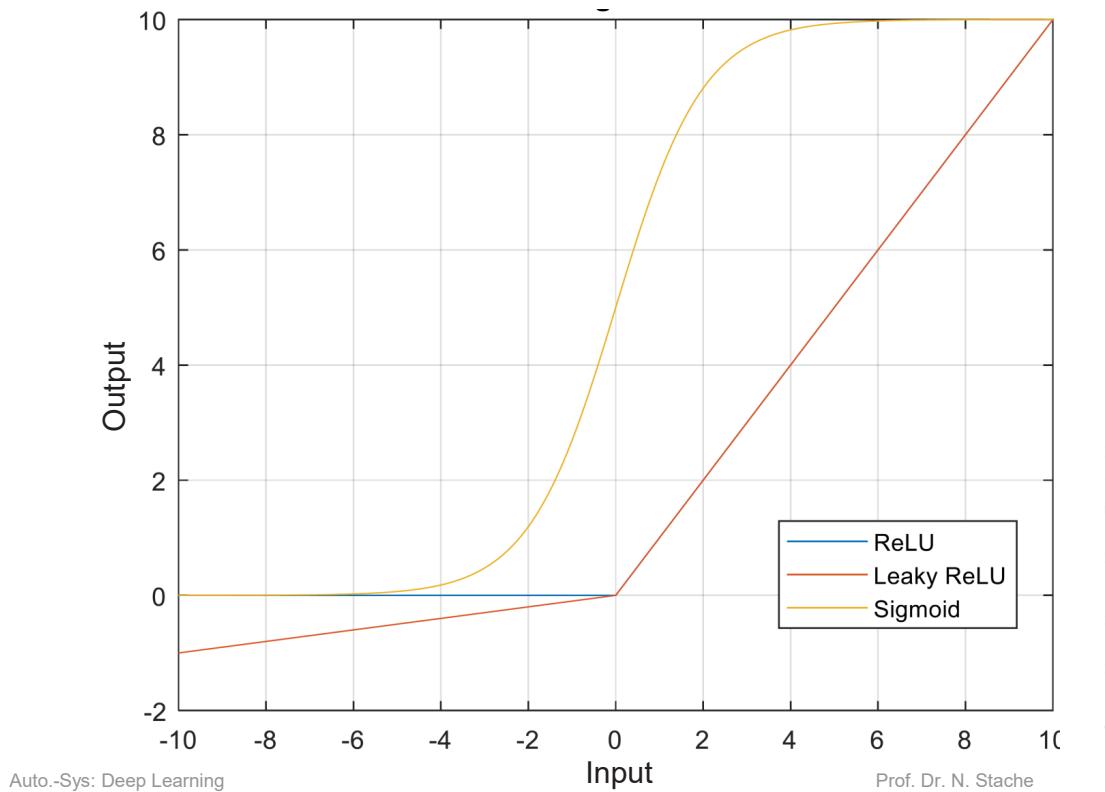
- Reduce saturation
- Very efficient to implement  
(threshold matrix to 0)
- Reduce vanishing gradient issue  
→ Enables deep network structures!

## ► Problems:

- Dying neurons (e.g. when lr is too high)



# Overview on activation functions

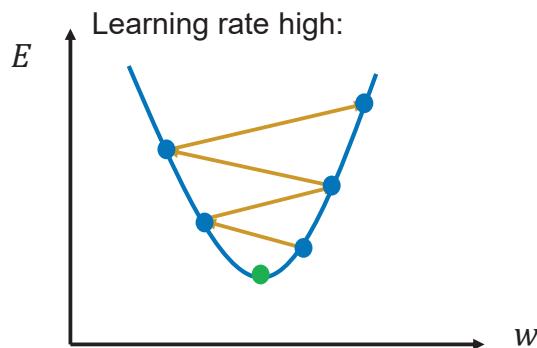


Source: Marco Altmann: Master Thesis, 2017

## Practical guidance

- ▶ Size of network
- ▶ Avoid overfitting
- ▶ Activation functions – downsides of sigmoid
- ▶ Optimizer tuning
- ▶ Neural Networks for classification of several classes
  - ▶ One hot encoding
  - ▶ Softmax
  - ▶ Logits
  - ▶ Cross Entropy loss
- ▶ Introduction to Tensorflow
- ▶ First deep neural network with Tensorflow

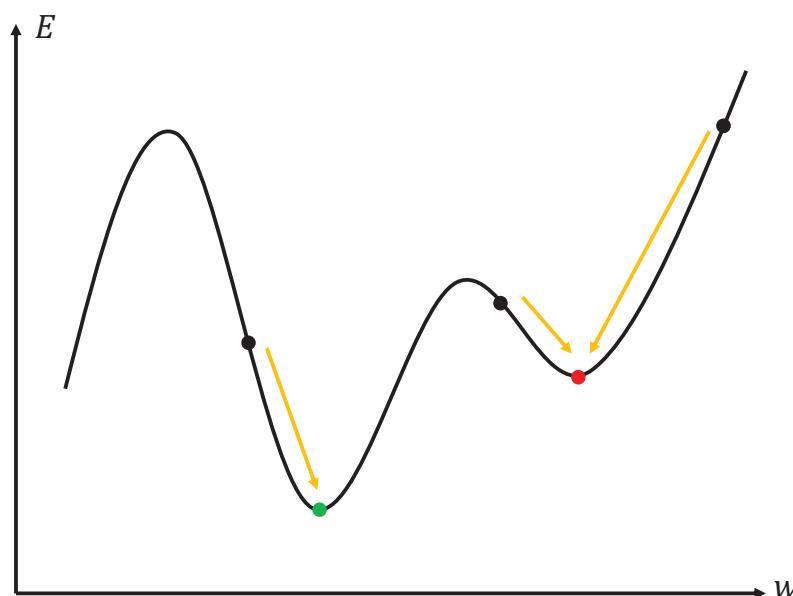
- If model is not working: Decrease learning rate



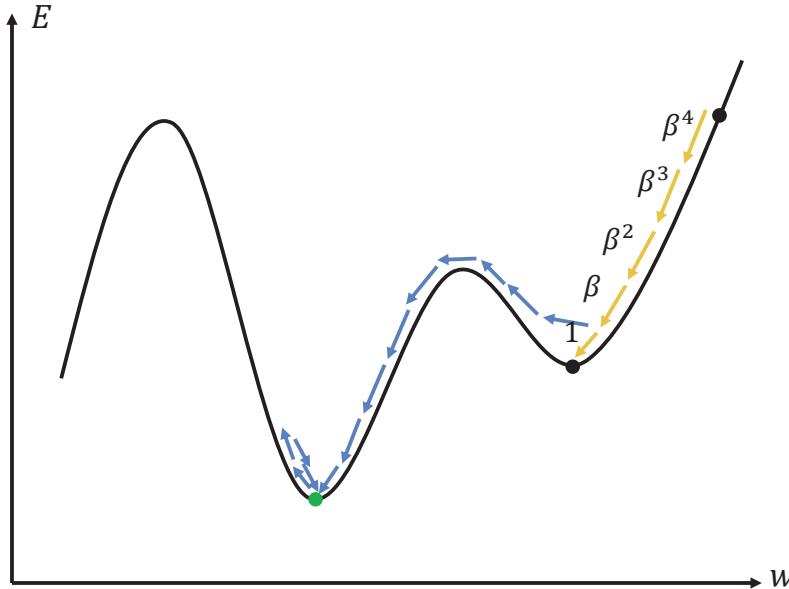
- Note: Learning is not necessarily constant (typ. decay over the epochs)
- Ideally: decrease, if model is close to solution
- There are Optimizers (e.g. ADAM-Optimizers) which do this by default.

## Random restarts

- Idea: Avoid local minima with random restarts



- ▶ Idea: Momentum, use weighted average of previous steps
- ▶  $step(n) \rightarrow step(n) + \beta step(n - 1) + \beta^2 step(n - 2) + \dots$



## Practical guidance

- ▶ Size of network
- ▶ Avoid overfitting
- ▶ Activation functions – downsides of sigmoid
- ▶ Optimizer tuning
- ▶ Neural Networks for classification of several classes
  - ▶ One hot encoding
  - ▶ Softmax
  - ▶ Logits
  - ▶ Cross Entropy loss
- ▶ Introduction to Tensorflow
- ▶ First deep neural network with Tensorflow

► Task:

- Perform classification, given multiple classes
- output the probability of each class



$P(\text{duck}) = 0.67$



$P(\text{beaver}) = 0.24$



$P(\text{walrus}) = 0.09$

- Output of Network is categorical
- Output values shall add to 1

## One-Hot Encoding

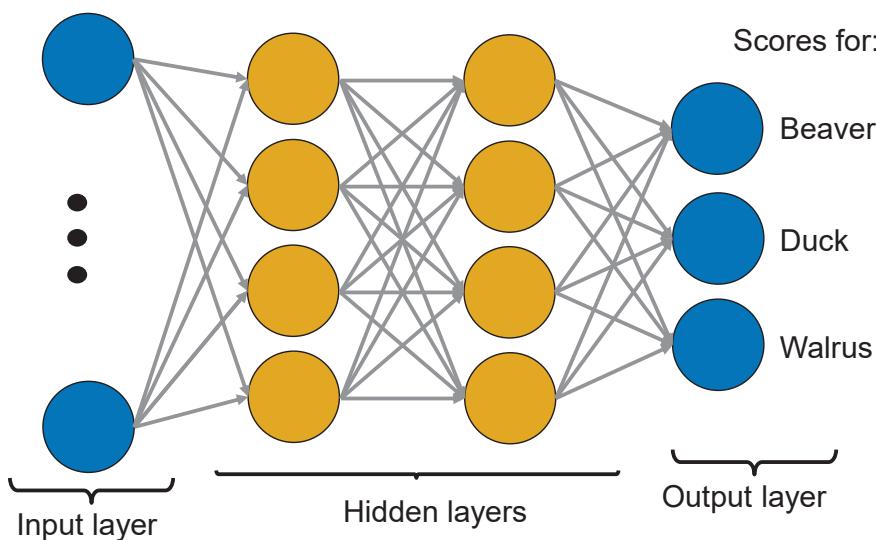
ANIMAL	VALUE
	?
	?
	?
	?
	?



ANIMAL	DUCK?	BEAVER?	WALRUS?
	1	0	0
	0	1	0
	1	0	0
	0	0	1
	0	1	0

- No values between duck, beaver, walrus make sense  
→ one-hot encoding is reasonable.

# Network for multiple classes

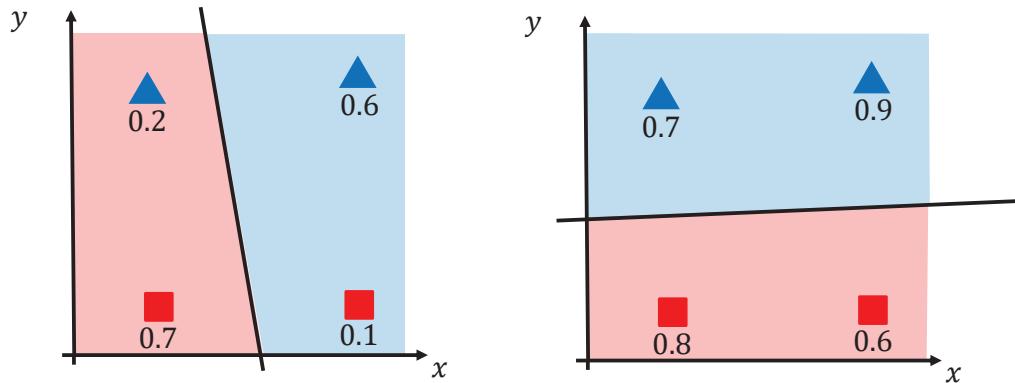


- ▶ Problem: Scores not necessarily
  - ▶ Add up to 1
  - ▶ Scores can even be negative

## Construct a network

- ▶ Problem: Scores not necessarily
  - ▶ Add up to 1
  - ▶ Scores can even be negative
- ▶ Approach:
  - ▶  $y = e^{Score}$  returns a positive value
  - ▶  $P(\text{duck}) = \frac{e^{\text{Score},\text{Duck}}}{e^{\text{Score},\text{Duck}} + e^{\text{Score},\text{Beaver}} + e^{\text{Score},\text{Walrus}}}$
- ▶ This function is called Softmax
  - ▶  $P(\text{class}_i) = \frac{e^{Z_i}}{e^{Z_1} + \dots + e^{Z_n}}$ ; given linear Scores  $Z_1, \dots, Z_n$
- ▶ Note: The scores are called “logits”

- ▶ Let us step back to a 2-Class problem:
- ▶ Assumption:
  - ▶ Model computes the probabilities in two class problem.



- ▶ Which model is better? How can we figure it out?
- ▶  $0.6 \cdot 0.2 \cdot 0.1 \cdot 0.7 = 0.0084$  vs.  $0.7 \cdot 0.9 \cdot 0.8 \cdot 0.6 = 0.3024$

# Maximum Likelihood

- ▶ Probability only needs to be maximized
- ▶ Problem: Numerical stability of Product
  - ▶ Product of many probabilities ends in small results  
→ numerically unstable
- ▶ Solution: Use Sum of Log(P)
  - numerically more stable
- ▶ Our Example:

$$0.6 * 0.2 * 0.1 * 0.7 = 0.0084$$

$$\ln(0.6) + \ln(0.2) + \ln(0.1) + \ln(0.7) \\ -0.51 \quad -1.61 \quad -2.3 \quad -0.36$$

$$-\ln(0.6) - \ln(0.2) - \ln(0.1) - \ln(0.7) = 4.8 \\ 0.51 \quad 1.61 \quad 2.3 \quad 0.36$$

$$0.7 * 0.9 * 0.8 * 0.6 = 0.3024$$

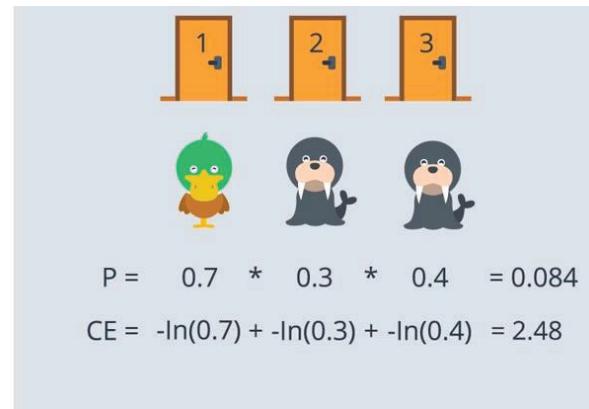
$$\ln(0.7) + \ln(0.9) + \ln(0.8) + \ln(0.6) \\ -0.36 \quad -0.1 \quad -0.22 \quad -0.51$$

$$-\ln(0.7) - \ln(0.9) - \ln(0.8) - \ln(0.6) = 1.2 \\ 0.36 \quad 0.1 \quad 0.22 \quad 0.51$$

# Cross Entropy

- ▶ Minimize Cross Entropy loss in order to maximize likelihood → Cross Entropy forms new error function
- ▶ How does it work for our tree class example?

ANIMAL	DOOR 1	DOOR 2	DOOR 3
	0.7	0.3	0.1
	0.2	0.4	0.5
	0.1	0.3	0.4

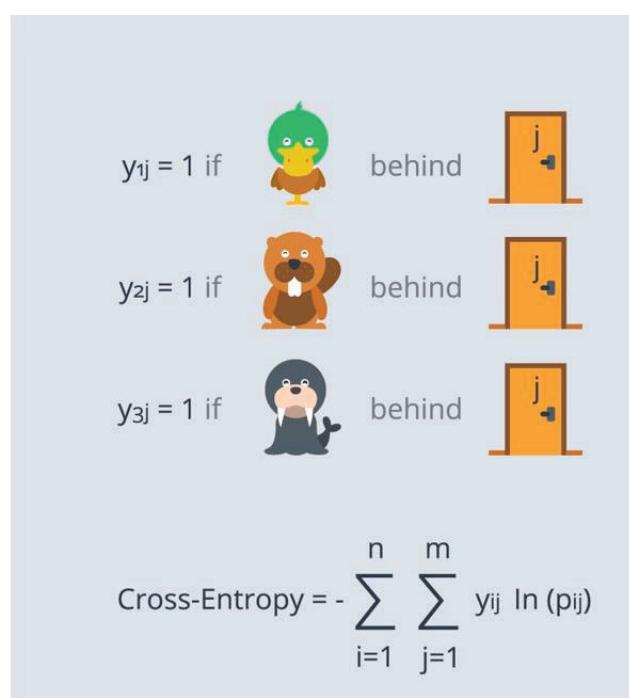


<https://www.youtube.com/watch?v=keDswcqkees>, 19.04.2018

## Cross Entropy Formula

### Multi-Class Cross-Entropy

ANIMAL	DOOR 1	DOOR 2	DOOR 3
	$p_{11}$	$p_{12}$	$p_{13}$
	$p_{21}$	$p_{22}$	$p_{23}$
	$p_{31}$	$p_{32}$	$p_{33}$



<https://www.youtube.com/watch?v=keDswcqkees>, 19.04.2018

## Error Function



ERROR FUNCTION:

$$- \frac{1}{m} \sum_{i=1}^m y_i \ln(\hat{y}_i) + (1-y_i) \ln(1-\hat{y}_i)$$



ERROR FUNCTION:

$$- \frac{1}{m} \sum_{i=1}^m \sum_{j=1}^n y_{ij} \ln(\hat{y}_{ij})$$

## Practical guidance

- ▶ Size of network
- ▶ Avoid overfitting
- ▶ Activation functions – downsides of sigmoid
- ▶ Optimizer tuning
- ▶ Neural Networks for classification of several classes
  - ▶ One hot encoding
  - ▶ Softmax
  - ▶ Logits
  - ▶ Cross Entropy loss
- ▶ Introduction to Tensorflow
- ▶ First deep neural network with Tensorflow

# Incomplete list of Libraries:

(Source: [https://en.wikipedia.org/wiki/Comparison\\_of\\_deep\\_learning\\_software](https://en.wikipedia.org/wiki/Comparison_of_deep_learning_software), 15.09.2017)

Software	Creator	Software license <sup>[a]</sup>	Open source	Platform	Written in	Interface	CUDA support	Has pretrained models	Recurrent nets	Convolutional nets
Caffe	Berkeley Center	Update: Caffe2 <sup>[2]</sup>	Yes	Linux, Mac OS X, Windows <sup>[2]</sup>	C++	Python, MATLAB				
Keras	François Chollet	MIT license	Yes	Linux, Mac OS X, Windows	Python	Python, R				
MatConvNet	Ulfhake	MIT license		Windows, Lin						
			Additionally: Matlab Neural Network Toolbox							
TensorFlow	Google Brain team	Apache 2.0	Yes	Linux, Mac OS X, Windows <sup>[30]</sup>	C++, Python	Python (Keras), C/C++, Java, Go, R <sup>[31]</sup>				
Theano	Université de Montréal	BSD license	Yes	Cross-platform	Python	Python				
Torch	Ronan Collobert, Koray Kavukcuoglu, Clement Farabet	BSD license	Yes	Linux, Mac OS X, Windows, <sup>[32]</sup> Android, <sup>[42]</sup> iOS	C, Lua	Lua, LuaJIT, <sup>[33]</sup> C, utility library for C++/OpenCL <sup>[44]</sup>				

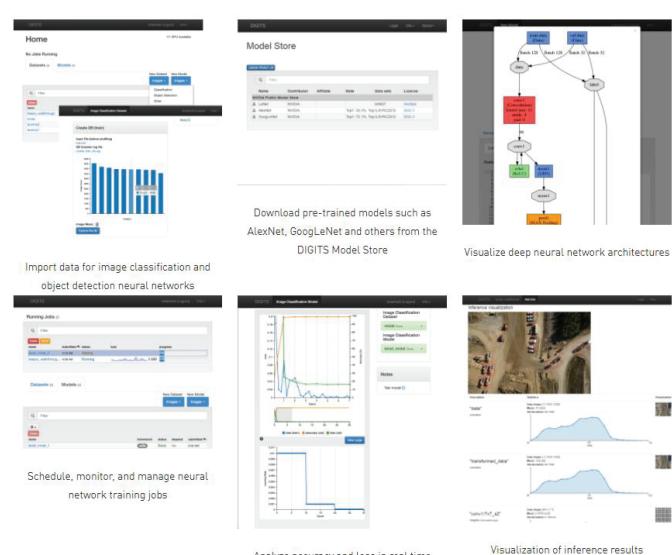
There are many more libraries...

However, Python seems to play an important role for interfacing the libraries

# NVIDIA Digits:

## Graphical high level user interface

- ▶ supports: Caffe, Torch, TensorFlow
- ▶ model store with pretrained models
- ▶ and much more...



- ▶ Open-source software library by Google Brain team
- ▶ Numerical computation using data flow graphs
- ▶ Designed for machine learning, deep neural networks... but not limited to these domains



# TensorFlow

Source: <https://en.wikipedia.org/wiki/TensorFlow> 19.09.2017

- ▶ Runs on GPUs and CPUs
- ▶ Interface to Python, C++, Java, JavaScript, Go
- ▶ Initial release: Nov. 9, 2015
- ▶ Current release: 2.1.0 (April 2020)

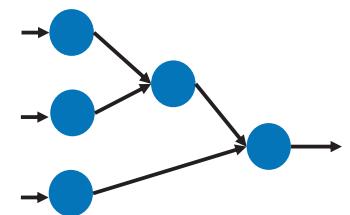
## What is meant by Tensor?

- ▶ Tensor: Multidimensional array of numerical values
- ▶ Rank or order of Tensor == dimensionality of Tensor

Dimensions	Example										
1	<table border="1"><tr><td>0</td><td>1</td><td>2</td></tr></table>	0	1	2	1d-Tensor, Vector						
0	1	2									
2	<table border="1"><tr><td>0</td><td>1</td><td>2</td></tr><tr><td>3</td><td>4</td><td>5</td></tr><tr><td>6</td><td>7</td><td>8</td></tr></table>	0	1	2	3	4	5	6	7	8	2d-Tensor, Matrix
0	1	2									
3	4	5									
6	7	8									
3	<table border="1"><tr><td>0</td><td>1</td><td>2</td></tr><tr><td>3</td><td>4</td><td>5</td></tr><tr><td>6</td><td>7</td><td>8</td></tr></table>	0	1	2	3	4	5	6	7	8	3d-Tensor
0	1	2									
3	4	5									
6	7	8									
N	<table border="1"><tr><td>...</td><td>...</td></tr><tr><td>...</td><td>...</td></tr></table>	...	...	...	...	Nd-Tensor					
...	...										
...	...										

1. Construct Data Flow Graph as TensorFlow function

- ▶ Defines, what to do with data (e.g. multiply, ...)

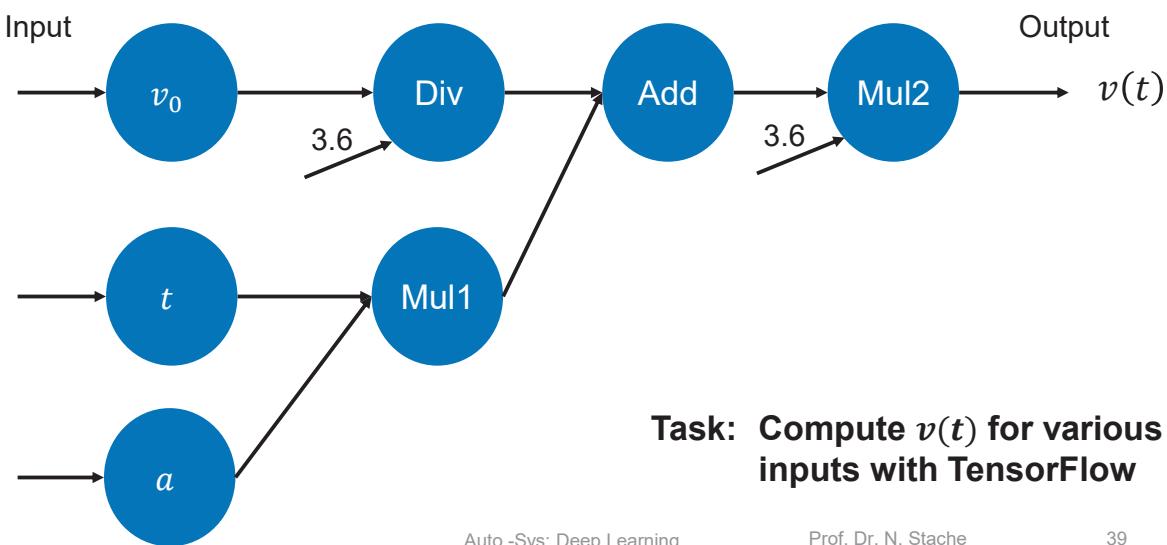


2. Call TensorFlow function with Input data

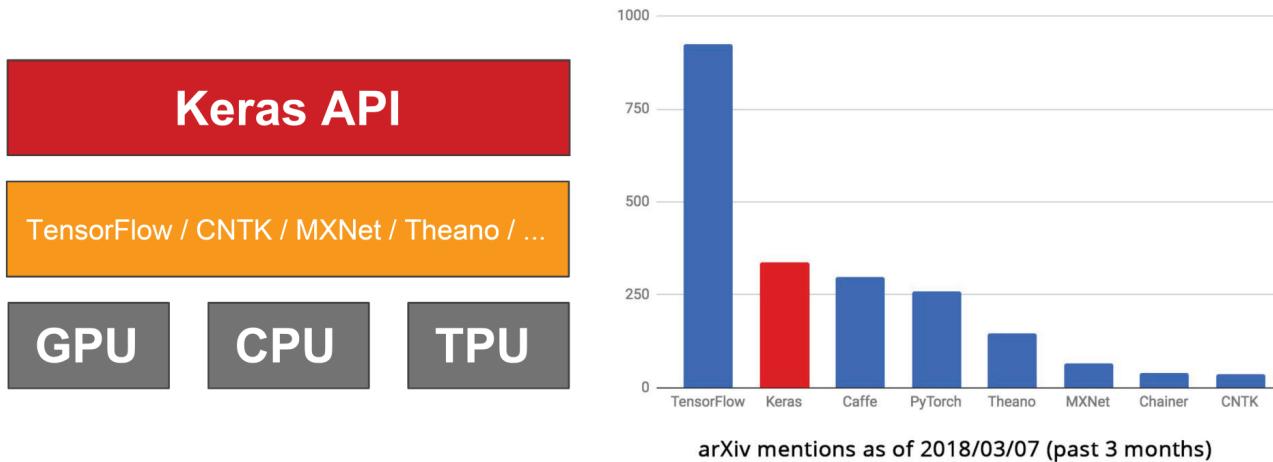
- ▶ Computation is carried out
- ▶ Returns Output data

## Example 1: Velocity computation with TensorFlow 1

- ▶ Unknown:  $v(t)$ : velocity of a vehicle in km/h
- ▶ Given:  $v_0$ : velocity in km/h,  $t$ : time in s,  $a$ : acceleration in m/s<sup>2</sup>
- ▶ Equation:  $v(t) = v_0 + a \cdot t$
- ▶ Data flow graph:



- ▶ Simplifies coding neural networks
- ▶ User-friendly front-end to e.g. Tensorflow
- ▶ E.g. infers shape of layers from known input shape



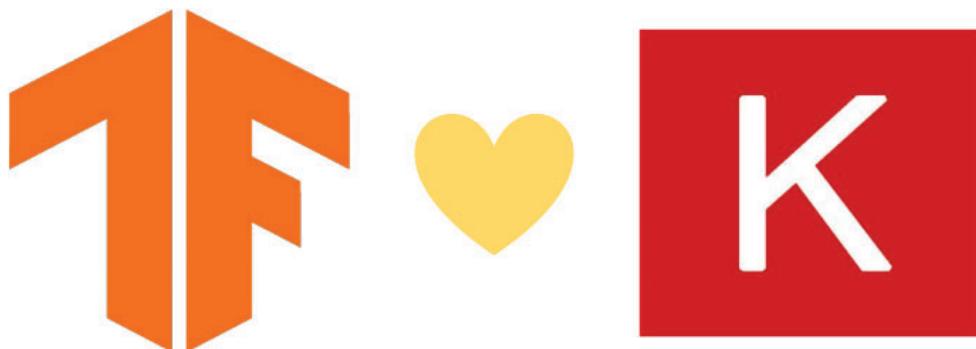
- ▶ Contributors: Google, Microsoft, Nvidia, Amazon Web Services,...
- ▶ Per year growth of community > 2x

Auto.-Sys: Deep Learning    Source: <https://web.stanford.edu/class/cs20si/lectures/march9guestlecture.pdf>, 11.09.2018    Prof. Dr. N. Stache

40

## tf.keras

- ▶ Since TensorFlow 2.0 Keras is built-in to TensorFlow as tf.keras
- ▶ No need to install Keras separately



## ► Sequential Model

- Simple
- Only single-input, single-output, sequential layer stacks
- Good for 70+% of use cases

## ► Functional Model

- Like playing with lego bricks
- Multi-input, multi-output, arbitrary static graph topologies
- Good for 95% of use cases

## ► Model subclassing

- Maximum flexibility
- Higher risk of errors

## Example: Sequential Model

```
import numpy as np
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense, Dropout

# Generate dummy data
x_train = np.random.random((1000, 20))
y_train = np.random.randint(2, size=(1000, 1))
x_test = np.random.random((100, 20))
y_test = np.random.randint(2, size=(100, 1))

model = Sequential()
model.add(Dense(64, input_dim=20, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(64, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(1, activation='sigmoid'))

model.compile(optimizer='rmsprop',
              loss='binary_crossentropy',
              metrics=['accuracy'])

model.fit(x_train, y_train,
          epochs=20,
          batch_size=128)
score = model.evaluate(x_test, y_test, batch_size=128)
```

# Example: Functional Model

```
import tensorflow as tf
from tensorflow.keras.layers import Dense
# This returns a tensor
inputs = tf.keras.Input(shape=(784,))

# a Layer instance is callable on a tensor, and returns a tensor
x = Dense(64, activation='relu')(inputs)
x = Dense(64, activation='relu')(x)
predictions = Dense(10, activation='softmax')(x)

# This creates a model that includes
# the Input Layer and three Dense Layers
model = tf.keras.Model(inputs=inputs, outputs=predictions)
model.compile(optimizer='rmsprop',
              loss='categorical_crossentropy',
              metrics=['accuracy'])

# starts training
model.fit(data, labels)
```

Source: <https://keras.io/getting-started/functional-api-guide/> | 11.09.2018

# Example: Model subclassing

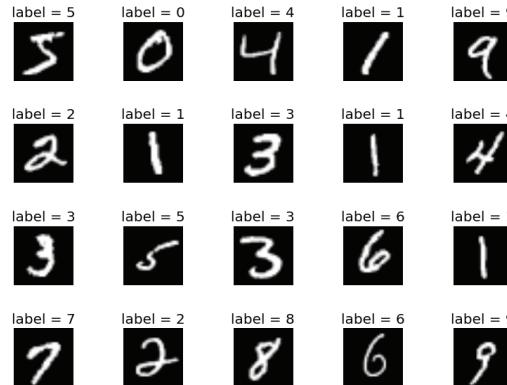
```
import tensorflow as tf
from tensorflow.keras.layers import Dense
class MyModel(tf.keras.Model):

    def __init__(self):
        super(MyModel, self).__init__()
        self.dense1 = Dense(20, activation='relu')
        self.dense2 = Dense(20, activation='relu')
        self.dense3 = Dense(10, activation='softmax')

    def call(self, inputs):
        x = self.dense1(inputs)
        x = self.dense2(x)
        return self.dense3(x)

model = MyModel()
model.fit(x, y, epochs=10, batch_size=32)
```

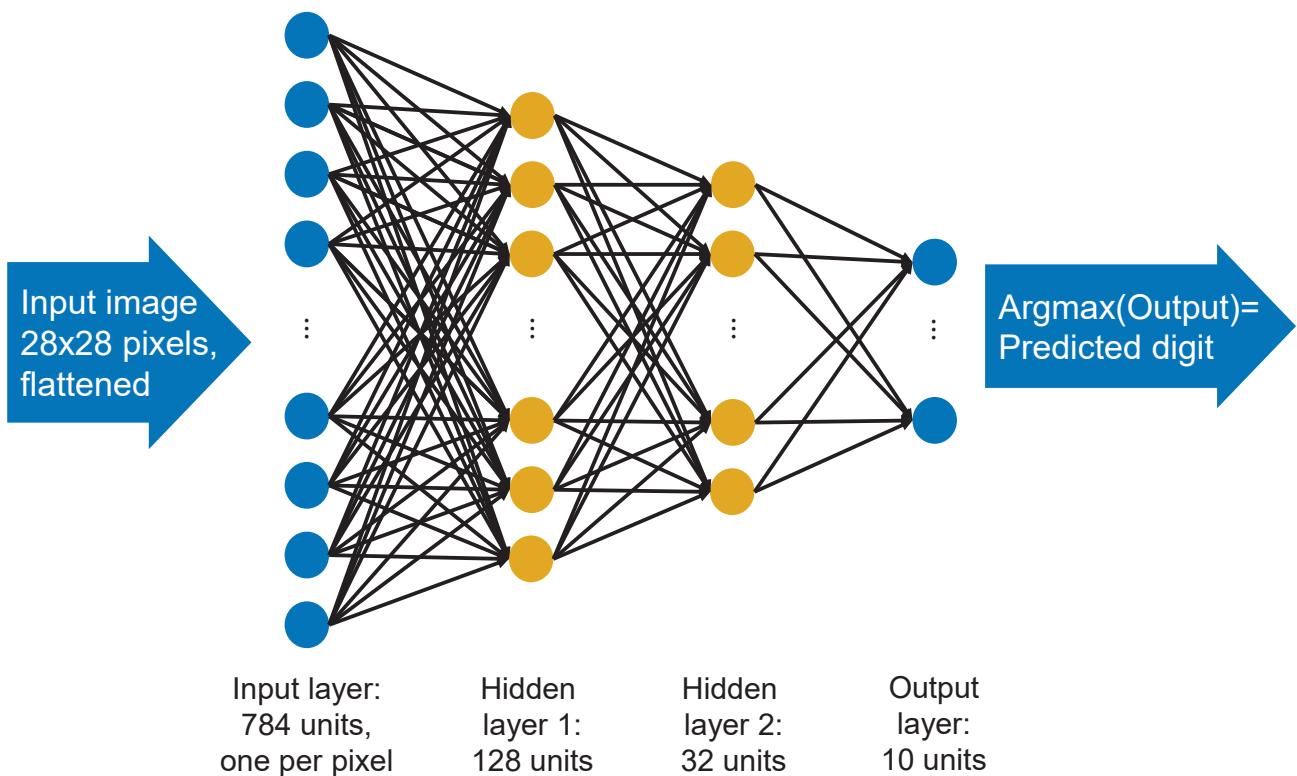
## Example 2: Digit Recognition with TensorFlow



Source: <http://corochann.com/mnist-dataset-introduction-1138.html> 19.09.2017

- ▶ MNIST database of handwritten digits  
(Modified National Institute of Standards and Technology database)
- ▶ We use 55,000 training images, 10,000 test images
- ▶ Image size: 28x28 pixels
- ▶ **Task: Classify digits using a deep neural network in TensorFlow**

## Example 2: Digit Recognition with TensorFlow





## Autonomous Systems: Deep Learning

### 6. Convolutional Neural Networks



Prof. Dr.-Ing. Nicolaj Stache  
Heilbronn University of Applied Sciences

#### Outline

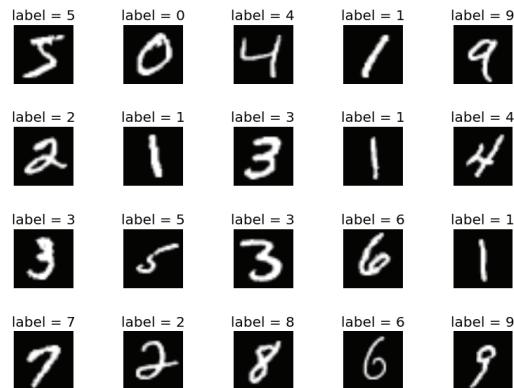


► Recap: Digit recognition with TensorFlow

► Convolutional neural networks

► Traffic sign recognition example

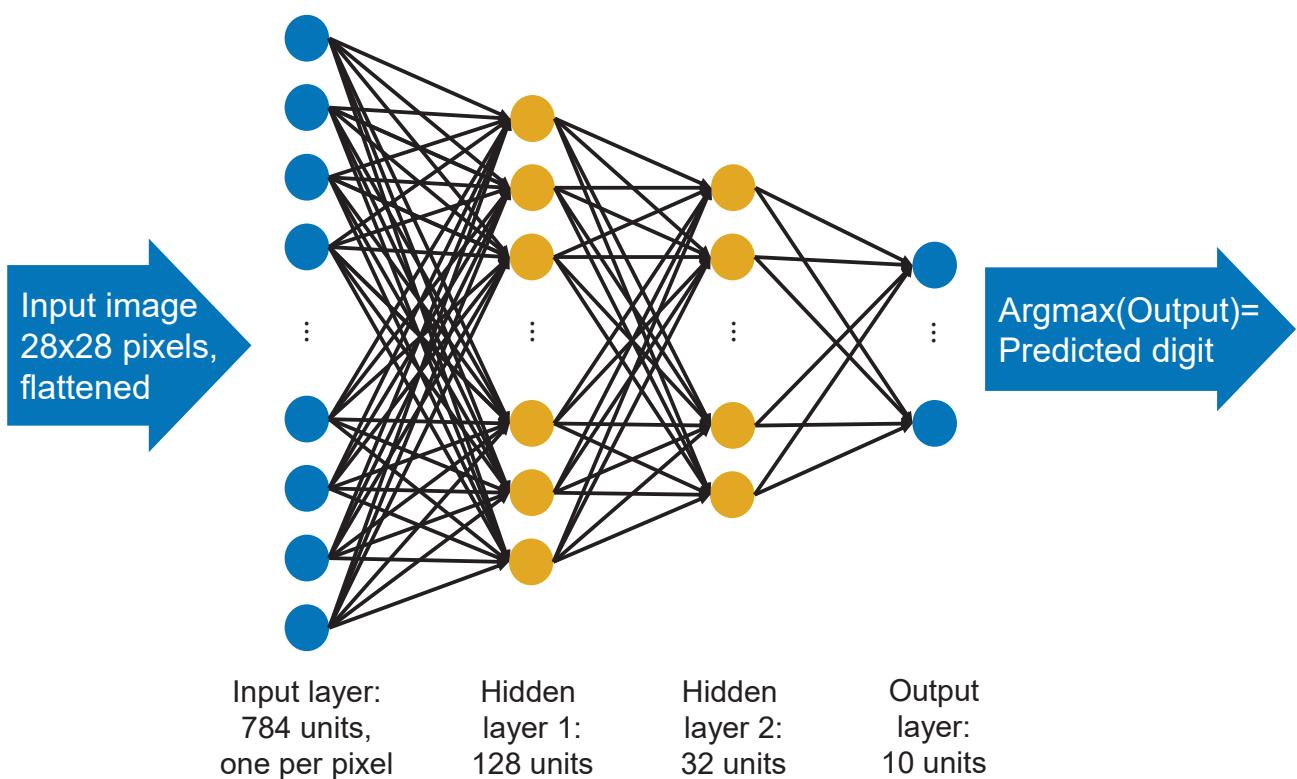
# Digit Recognition with TensorFlow



Source: <http://corochann.com/mnist-dataset-introduction-1138.html> 19.09.2017

- ▶ MNIST database of handwritten digits  
(Modified National Institute of Standards and Technology database)
- ▶ We use 55,000 training images, 10,000 test images
- ▶ Image size: 28x28 pixels
- ▶ **Task: Classify digits using a deep neural network in TensorFlow**

# Digit Recognition with TensorFlow



# Outline

► Recap: Digit recognition with TensorFlow

► Convolutional neural networks

► Traffic sign recognition example

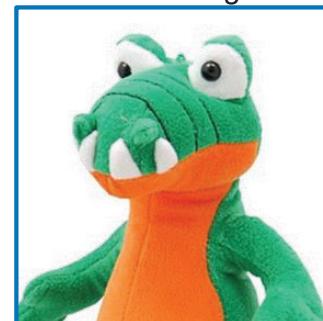
## Motivation 1: Convolutional Neural Networks

MNIST-Image:



Width: 28 px  
Height: 28 px  
Color channels: 1  
→ Parameters: 784

“Real” Image:

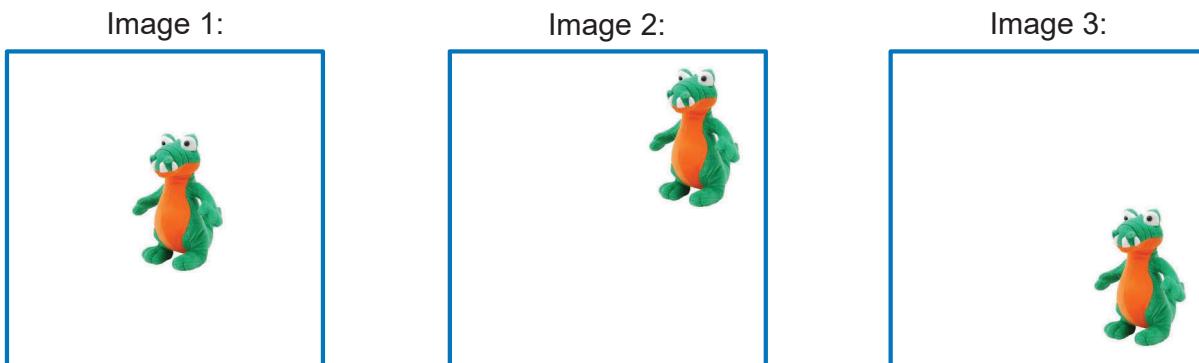


Width: 200 px  
Height: 200 px  
Color channels: 3  
→ Parameters: 120,000

Real image:

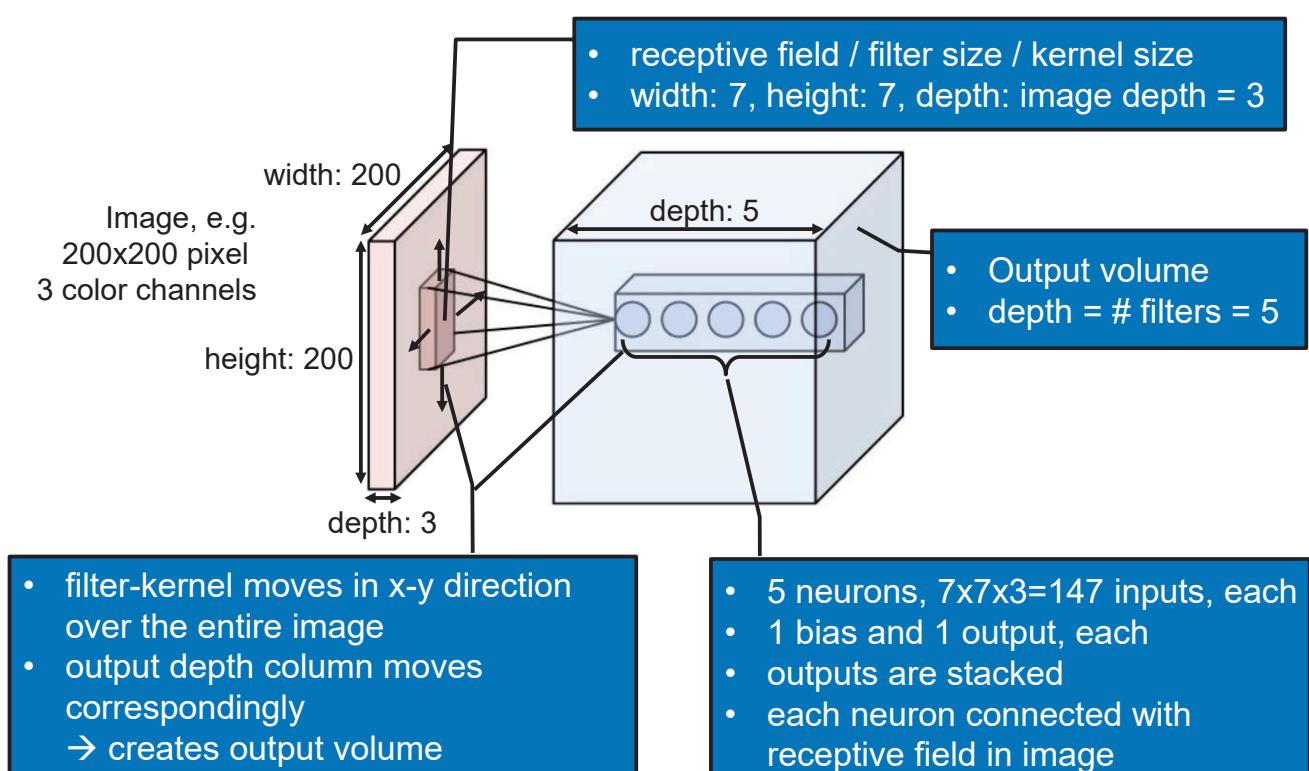
- 120,000 input units:  
→ 120,000 weights for each neuron in 1<sup>st</sup> hidden layer

## Motivation 2: Convolutional Neural Networks

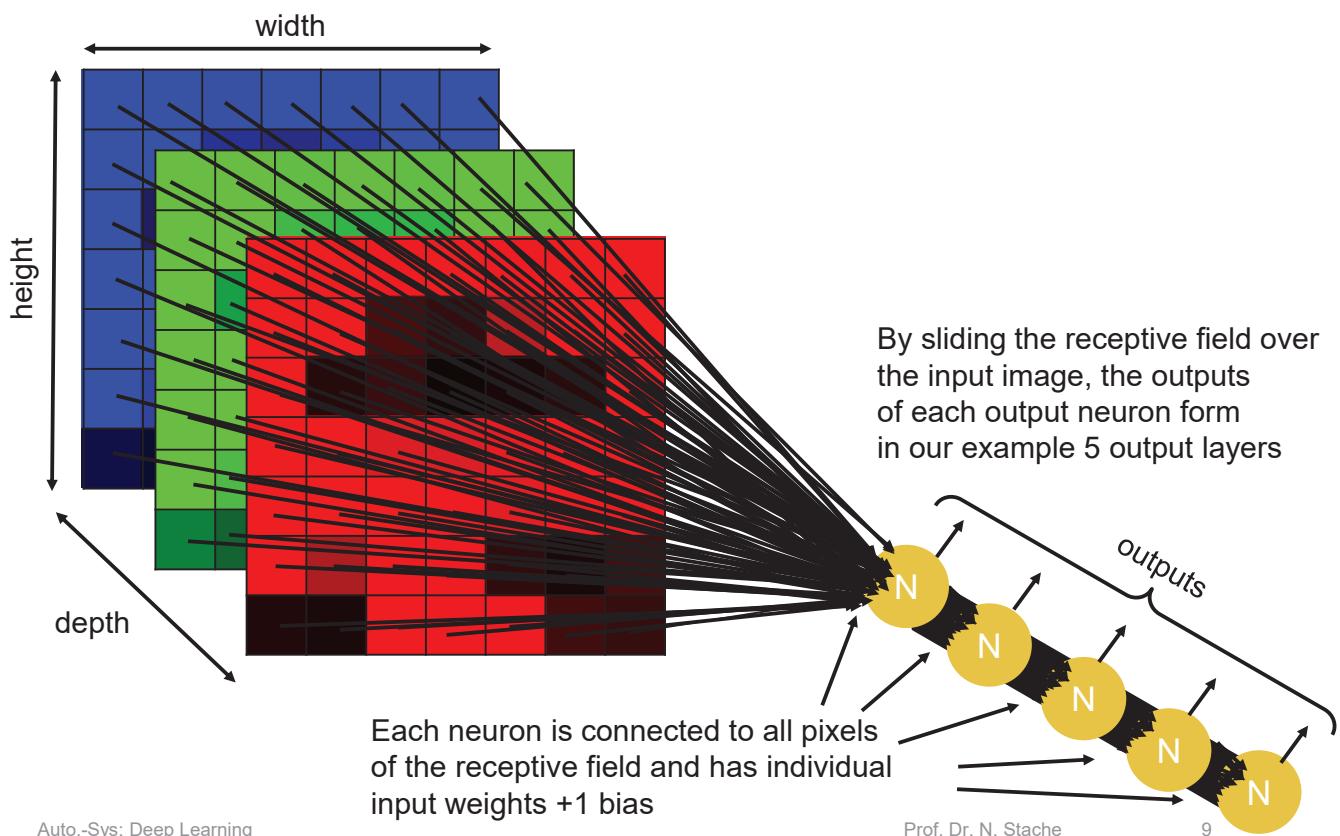


- ▶ Q: Which object is in Image 1, 2, 3? → A: “Alligator”
- ▶ Problem:
  - Images 1, 2, 3 are different(!)
  - Network has to learn the object at each position in the image?!
- ▶ Solution: We need translation invariance!
  - Convolutional Neural Networks
  - **Weight sharing** across space

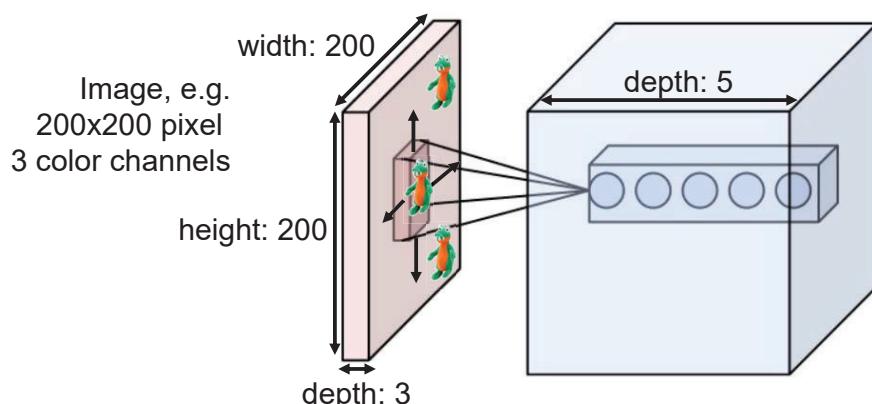
## Convolutional Neural Networks (ConvNets)



## Other illustration of Example

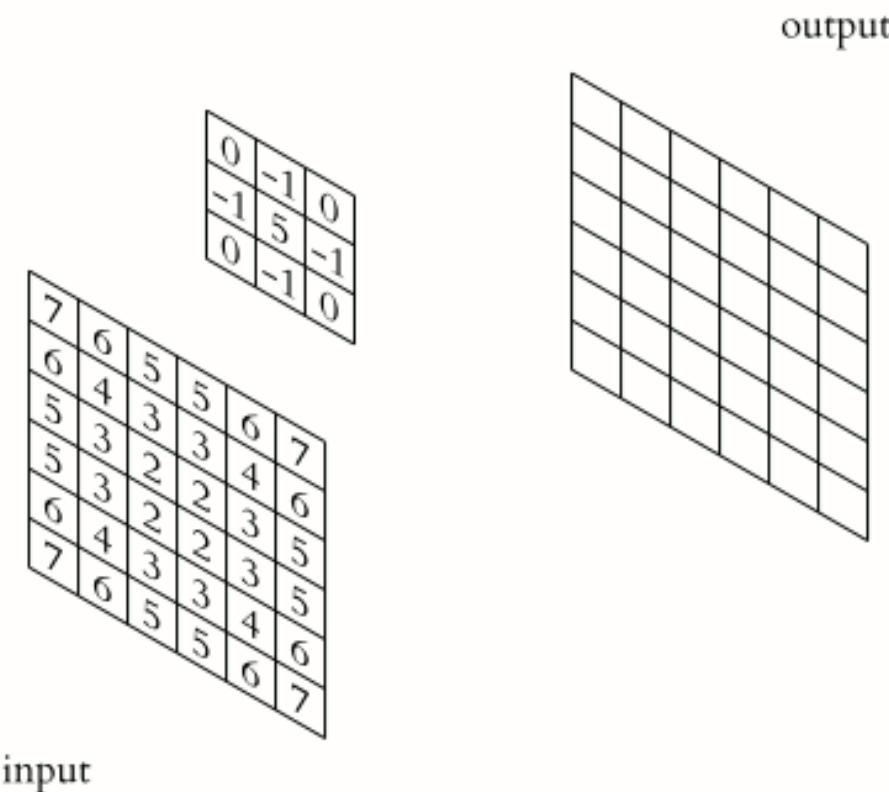


## Convolutional Neural Networks (ConvNets)



- ▶  $7 \times 7 \times 3 = 147$  inputs per neuron vs. 120,000 inputs
- ▶ Image size matters less
- ▶ Translation invariance: position of feature wrt. image irrelevant  
→ for larger features: more stacked conv layers needed

# Convolution in Image Processing



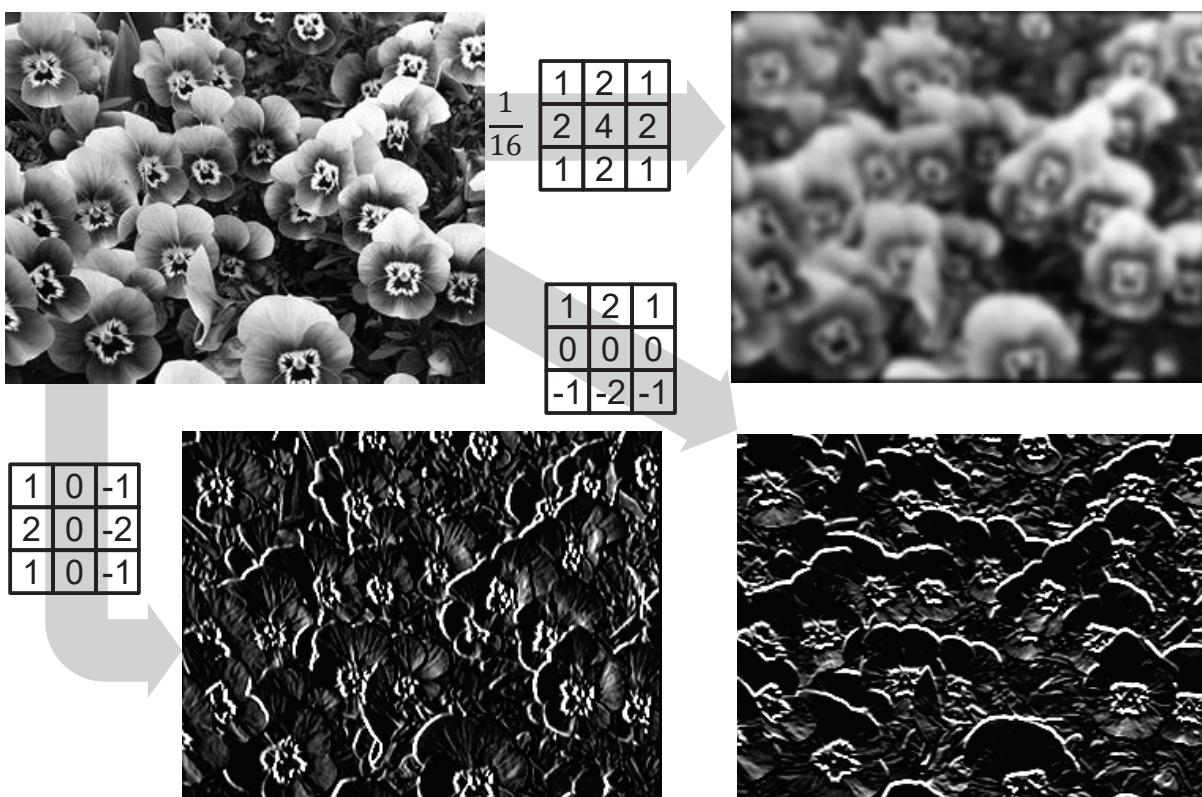
Auto.-Sys: Deep Learning

Prof. Dr. N. Stache

11

Von Michael Plotke - Eigenes Werk. CC BY-SA 3.0, <https://commons.wikimedia.org/w/index.php?curid=24288956>

## Examples of Different Conv Kernels

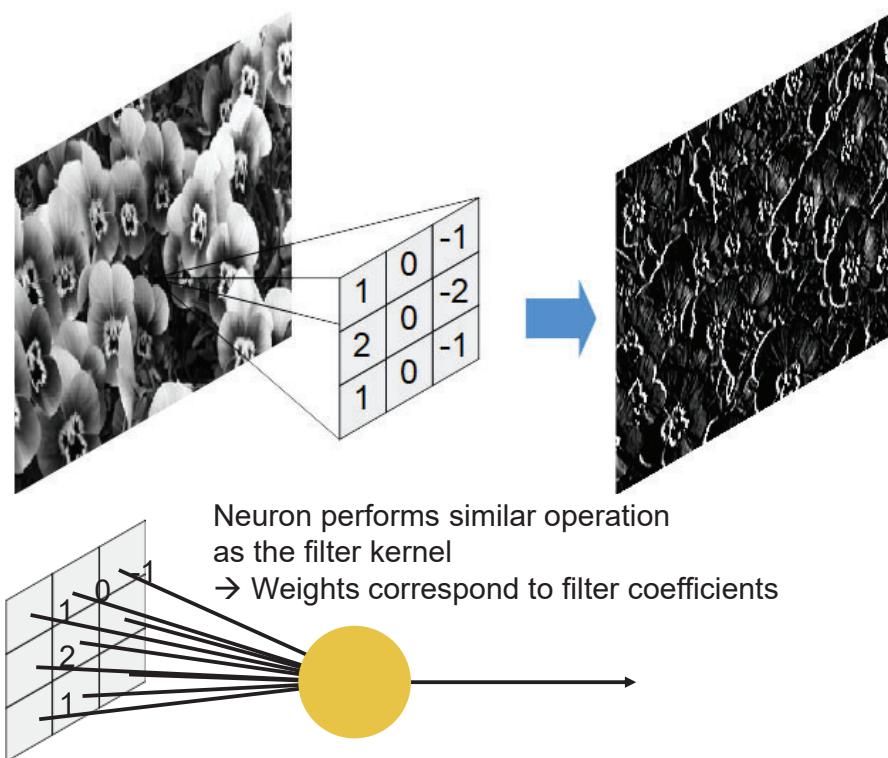


Auto.-Sys: Deep Learning

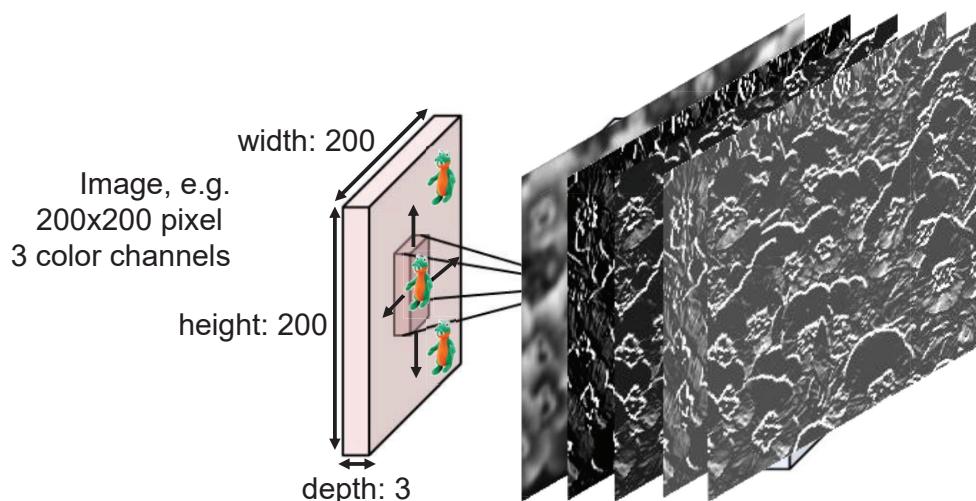
Prof. Dr. N. Stache

12

# Examples of Different Conv Kernels



## Convolutional Neural Networks (ConvNets)

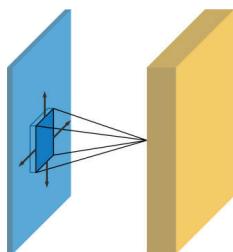


- ▶ 5 neurons are used
- ▶ output is composed of 5 different (!) filter outputs  
→ 5 ways of extracting information from the image
- ▶ Approach: Increase the amount of filters, condense the information by reduction of width and height

## Parameters:

- ▶ Stride: how many pixels the filter is moves per step

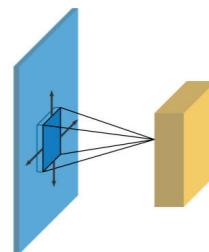
Stride = 1:



Output width =  
input width\*

Output height =  
input height\*

Stride = 2:



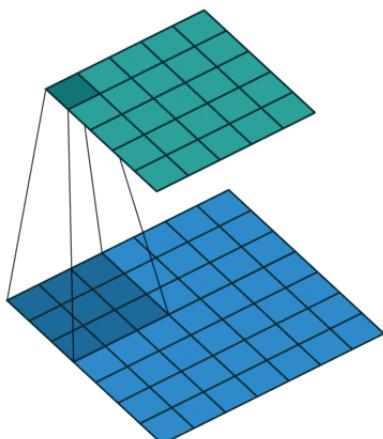
Output width =  
input width / 2\*

Output height =  
input height / 2\*

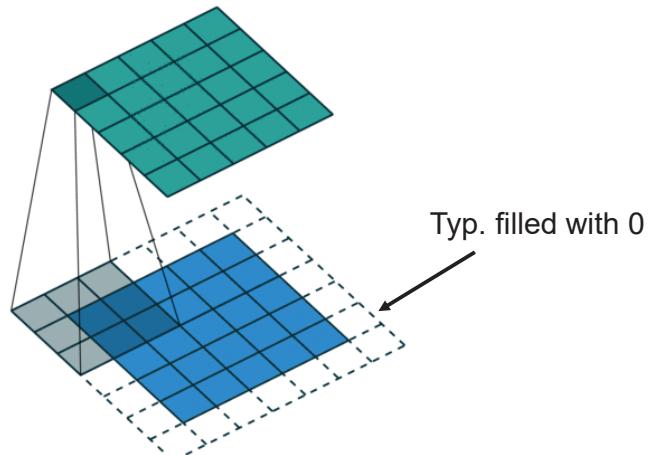
\*) same padding applied

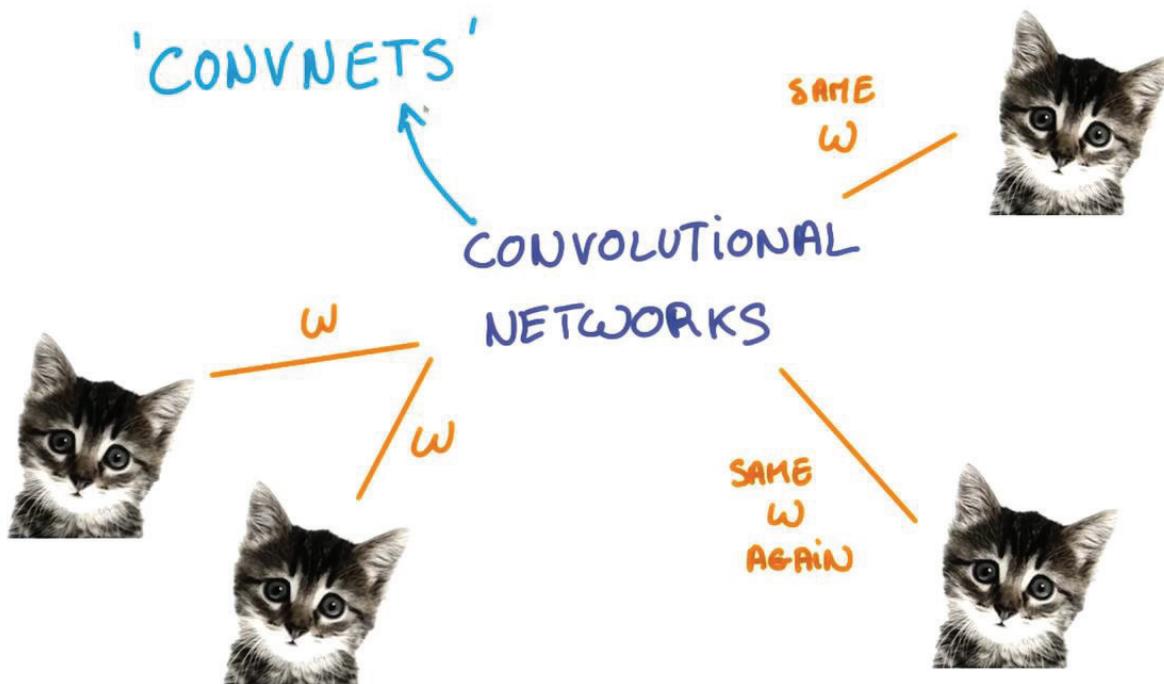
# Convolutional Neural Networks (ConvNets)

## Valid Padding:

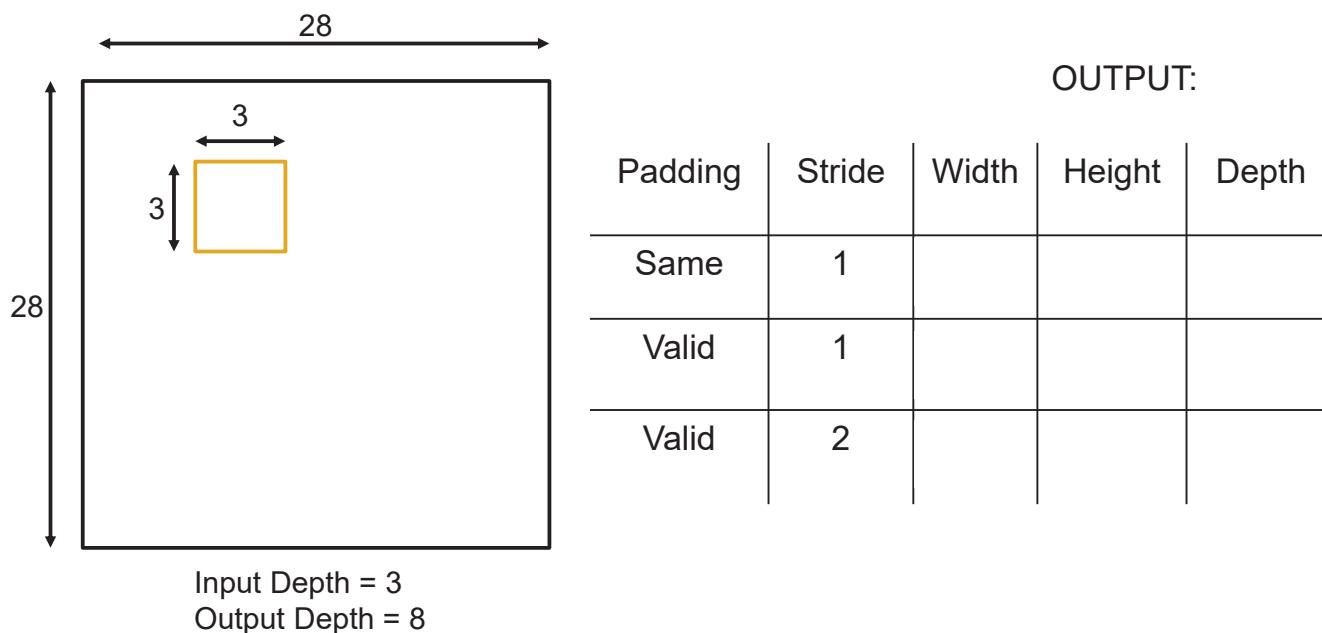


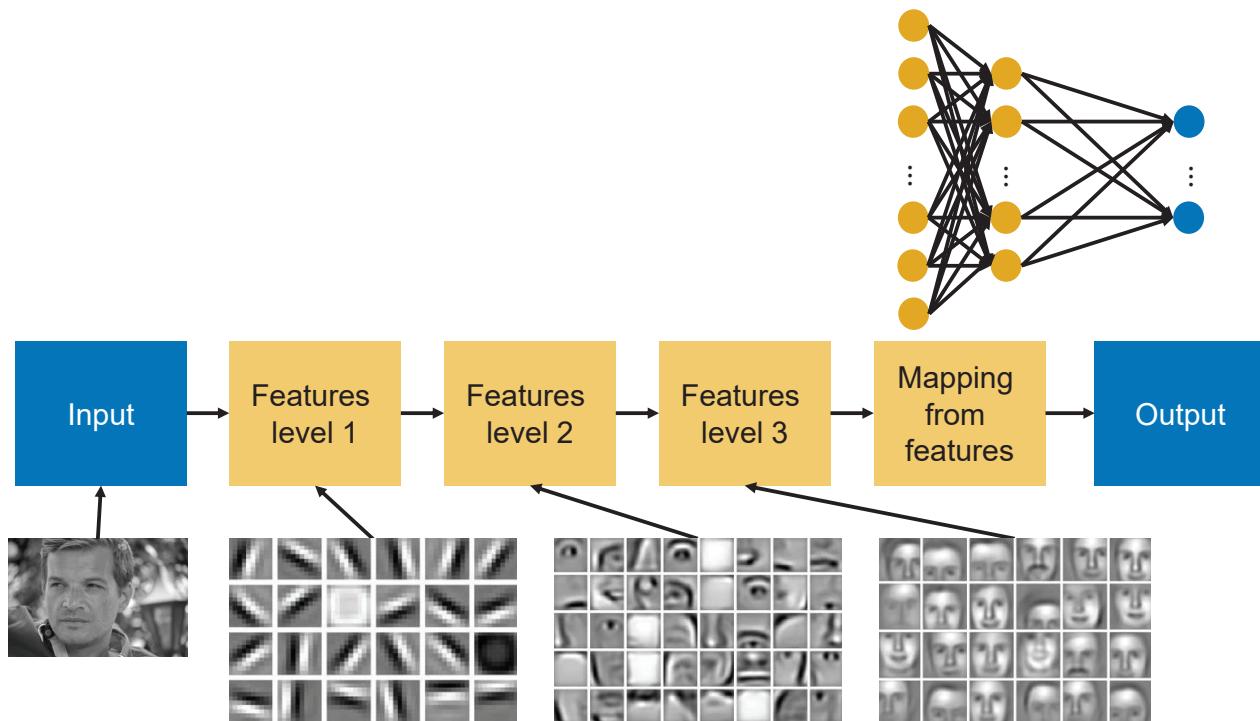
## Same Padding:





## Task





Introduction to Deep Learning

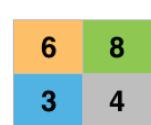
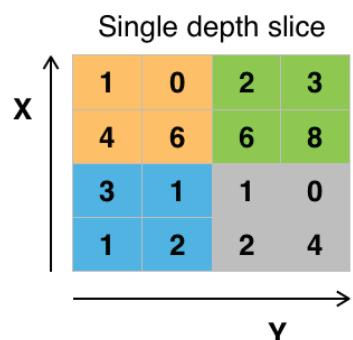
Source: <https://www.nature.com/news/computer-science-the-learning-machines-1.14481>, Images Andrew Ng, 27.08.2017

Prof. Dr. N. Stache

19

## Pooling

- ▶ Method to reduce width and height by combining information
- ▶ Alternative to stride
- ▶ Most common: Max-Pooling; alternative: average pooling

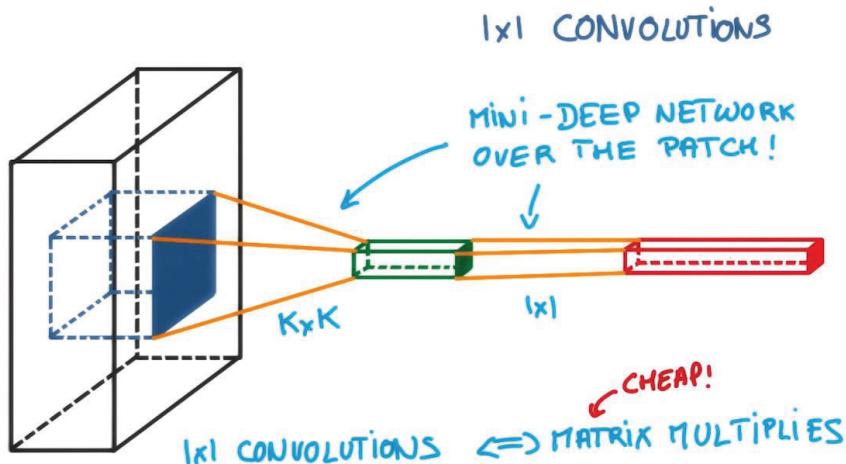


Pooling Parameters:  
 - Pooling type: max  
 - Pooling size: 2x2  
 - Pooling stride: 2

- ▶ Note: Typically more expensive compared to stride  
(if stride is 1 for a long time, the number of parameters keep high)

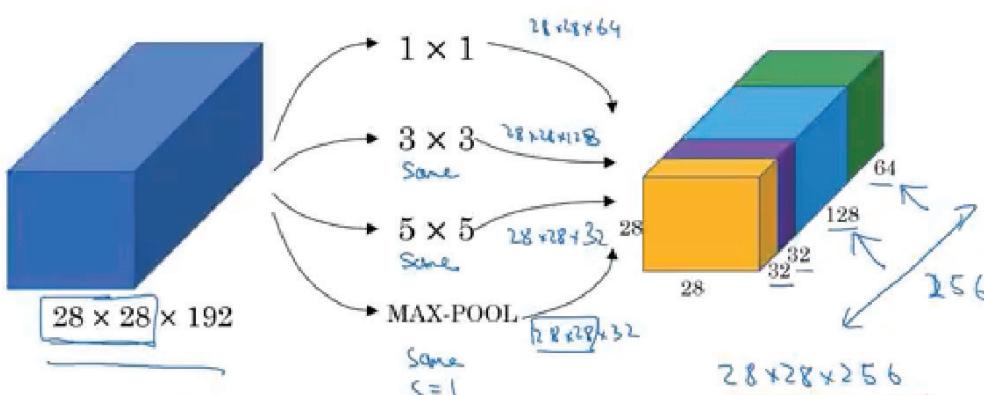
# 1x1 convolution

- ▶ Size of receptive field is  $1 \times 1$ , stride is 1
- ▶ Parameter: number of filters
  - ▶ If number of filters < depth  $\rightarrow$  reduced depth of output volume
  - ▶ If number of filters == depth  $\rightarrow$  equal depth of output volume,  
(due activation function of  $1 \times 1$  convolution neurons, nonlinear modelling capacity is increased)



# Inception module

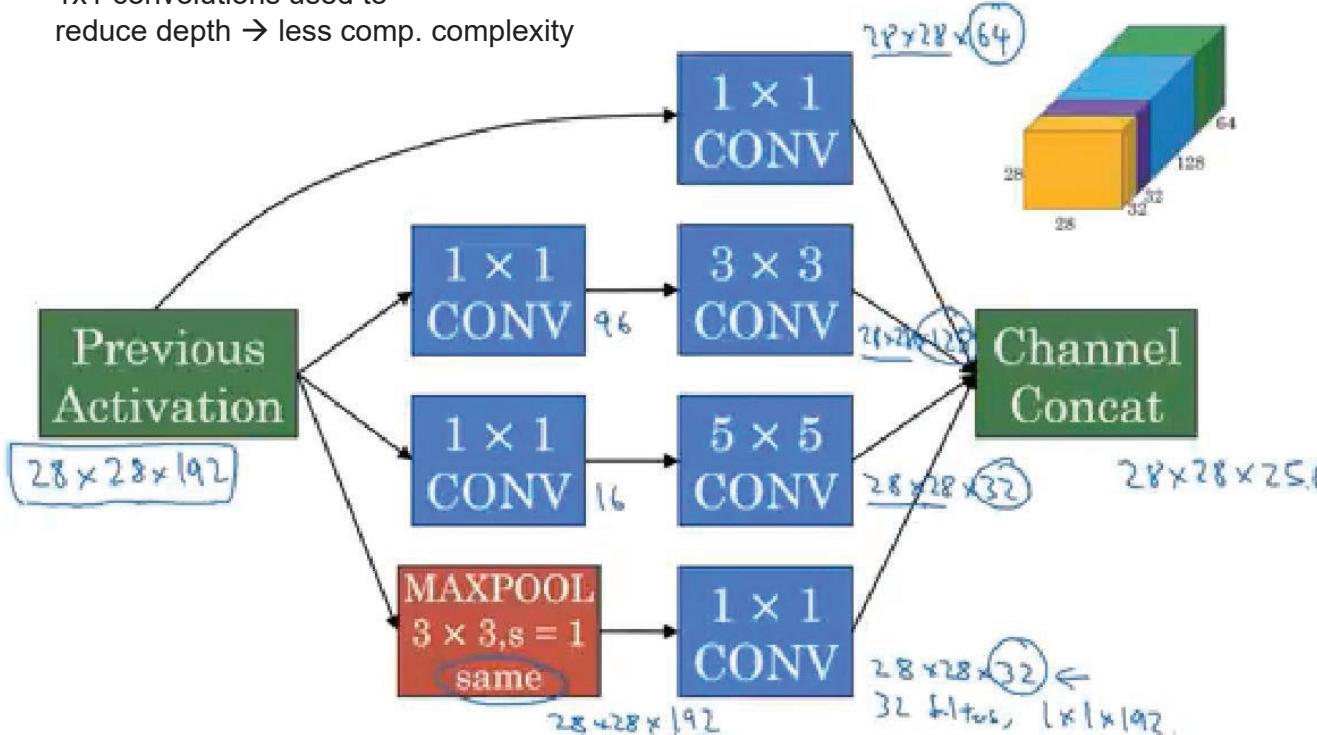
- ▶ Uncertainty of what to pick:
  - ▶  $1 \times 1$  convolution
  - ▶  $3 \times 3$ ,  $5 \times 5$  convolution?
  - ▶ Max-pooling...?
- ▶ Approach: Do it all  $\rightarrow$  concatenate outputs!



# Inception module

(basis of “inception” networks, e.g. GoogLeNet)

1x1 convolutions used to  
reduce depth → less comp. complexity



## Wording

- ▶ Kernel size: size of the “sliding window”
- ▶ Stride: Amount of steps the Kernel is moved
- ▶ Padding:
  - ▶ Same: Input is padded with 0 that output volume has the same width and height as the input, if stride == 1
  - ▶ Valid: No padding is applied -> output is a bit smaller compared to Same padding, depending on kernel size

- ▶ Recap: Digit recognition with TensorFlow
- ▶ Convolutional neural networks
- ▶ Traffic sign recognition example

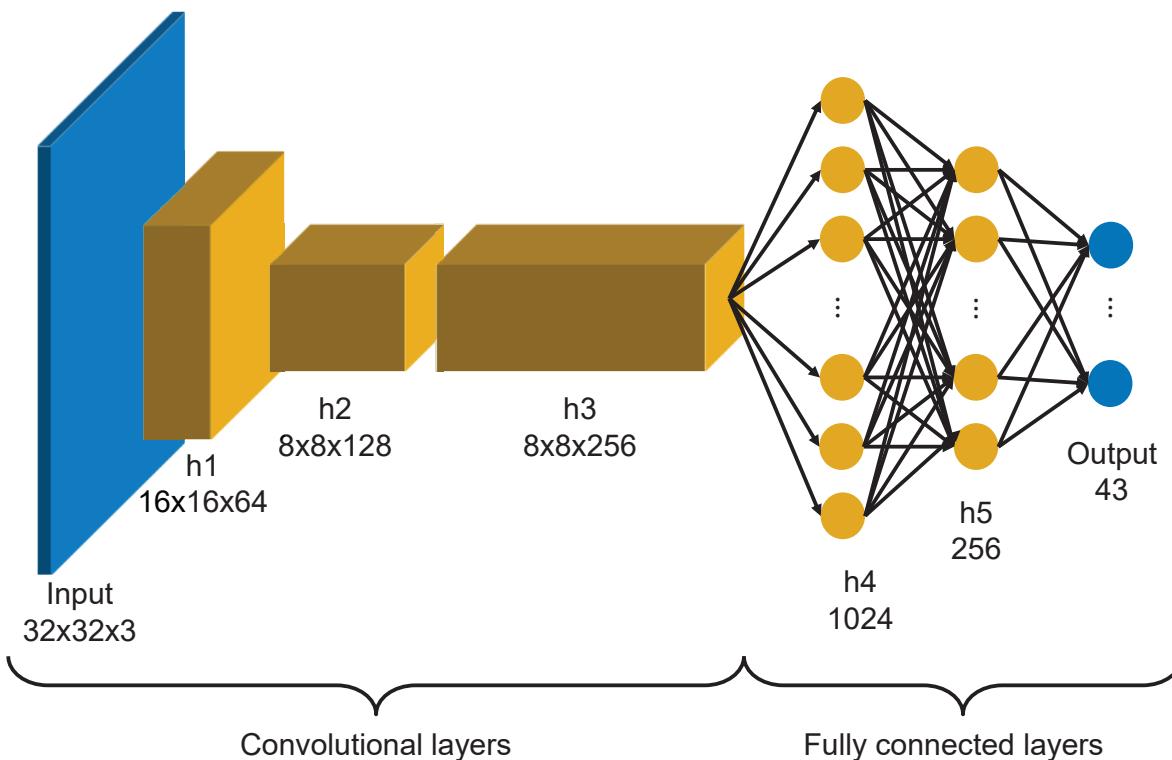
## Example 3: Traffic Sign Recognition with Keras



Source:  
<http://benchmark.ini.rub.de/?section=gtsrb&subsection=dataset>  
19.09.2017

- ▶ GTSRB-Database of German Traffic signs
- ▶ > 50,000 images in total, 43 classes
- ▶ Image size: from 15x15 ... 250x250 pixels  
→ we scaled to 32x32 pixels for this workshop
- ▶ **Task: Classify traffic signs with deep convolutional neural network!**

## Example 3: Traffic Sign Recognition with Keras



## Hands-on work

### Task:

- ▶ Check out the gardening example with Keras
- ▶ Try to internalize, how Keras works



## Autonomous Systems: Deep Learning

### 7. Network architectures + transfer learning



HOCHSCHULE HEILBRONN

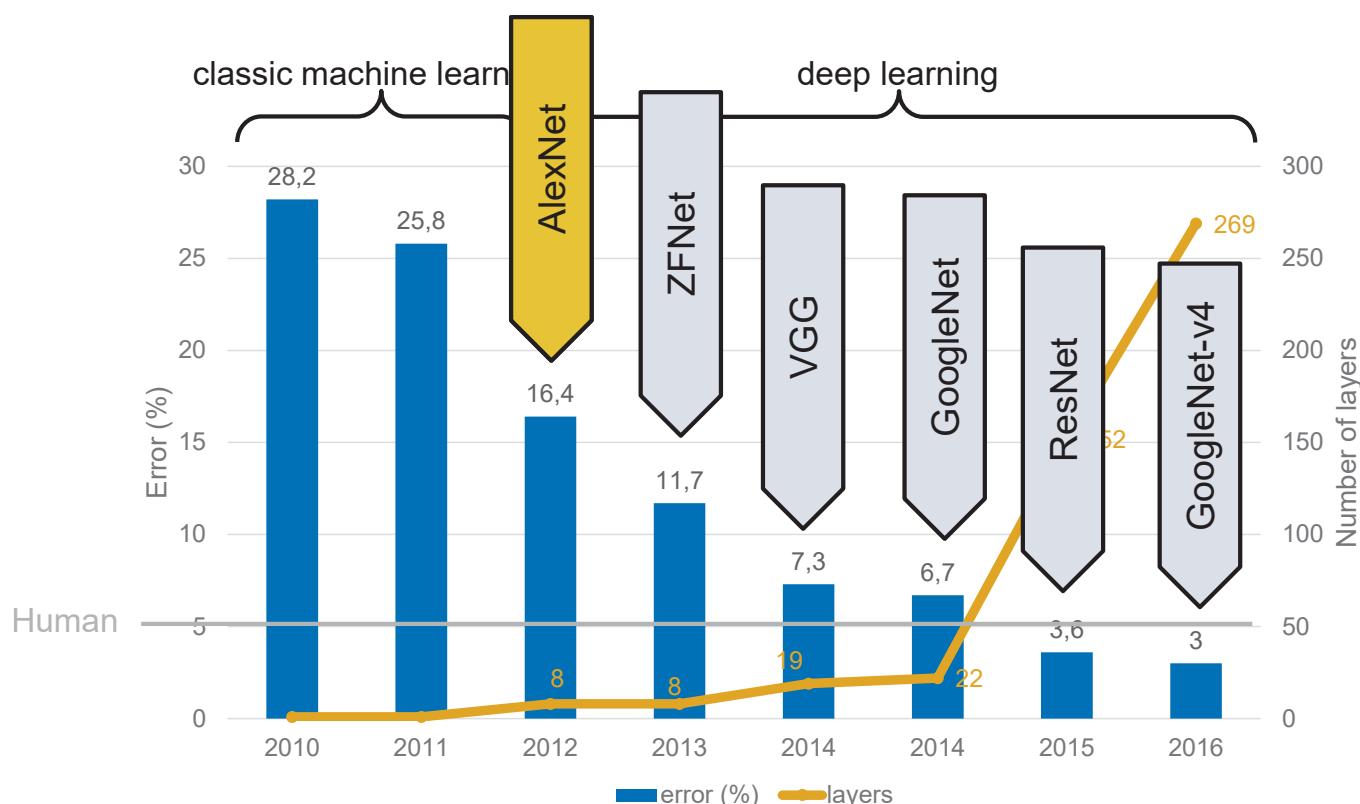
Prof. Dr.-Ing. Nicolaj Stache

Heilbronn University of Applied Sciences

## Outline



- ▶ Famous network architectures
- ▶ Batch normalization
- ▶ Transfer learning
- ▶ Semantic segmentation



→ Great Success of Deep Learning in ImageNet Challenge!

## AlexNet

Categorical output 1000 categ.

- ▶ Krizhevsky et al. 2012
- ▶ 5 conv layers and 3 fully connected layers
- ▶ 62.3 million parameters, 1.1 billion computations for forward pass
- ▶ Uses ReLU activation function
- ▶ Dropout 0.5
- ▶ Overlap-Pooling to reduce size of the network
- ▶ Local contrast normalization (not common anymore)
- ▶ Learning rate adaptation (e.g. when validation acc. Plateaus)
- ▶ SGD + Momentum
- ▶ Used heavy data augmentation for training
- ▶ Training takes 6 days on 2 GTX580 GPUs (split because only 3 GB memory)



Input image: 227x227x3

Categorical output 1000 categ.

- Q: What is the output volume of conv-layer with 11x11 kernel-size, 96 filters, stride=4, valid padding, input image: 227x227x3?
- Q: What is the number of parameters?



[http://vision.stanford.edu/teaching/cs231b\\_spring14/slides/alexnet\\_tugce\\_kyunghiee.pdf](http://vision.stanford.edu/teaching/cs231b_spring14/slides/alexnet_tugce_kyunghiee.pdf), 26.04.2018

Auto.-Sys: Deep Learning

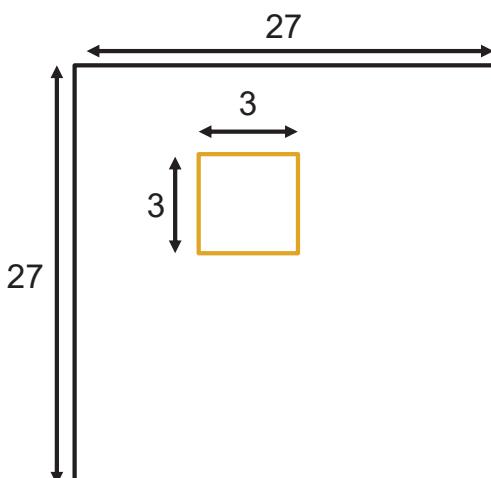
Input image: 227x227x3

Prof. Dr. N. Stache

5

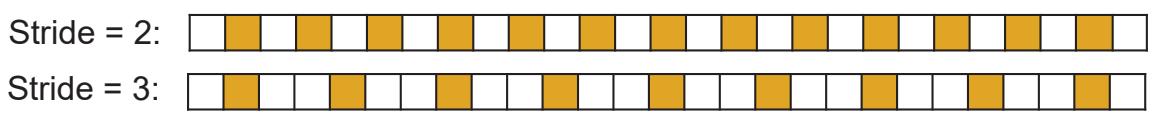
## Task

Input Depth = 3  
Output Depth = 8



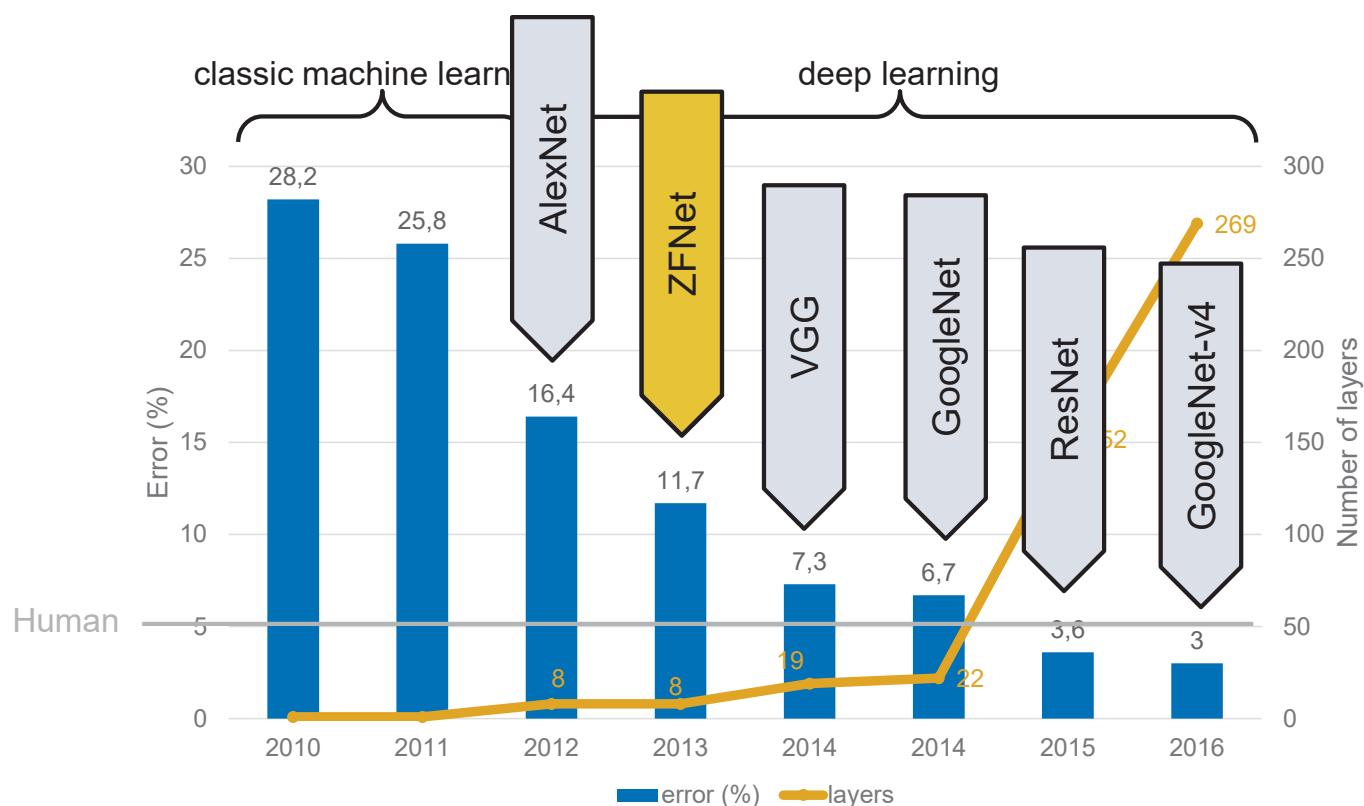
Padding	Stride	Width	Height	Depth	Parameters
Same	1				
Valid	1				
Valid	2				
Valid	3				

OUTPUT:



$$\text{OutputSize} = \frac{\text{InputSize} - \text{KernelSize} + 2 \cdot \text{PaddingSize}}{\text{stride}} + 1$$

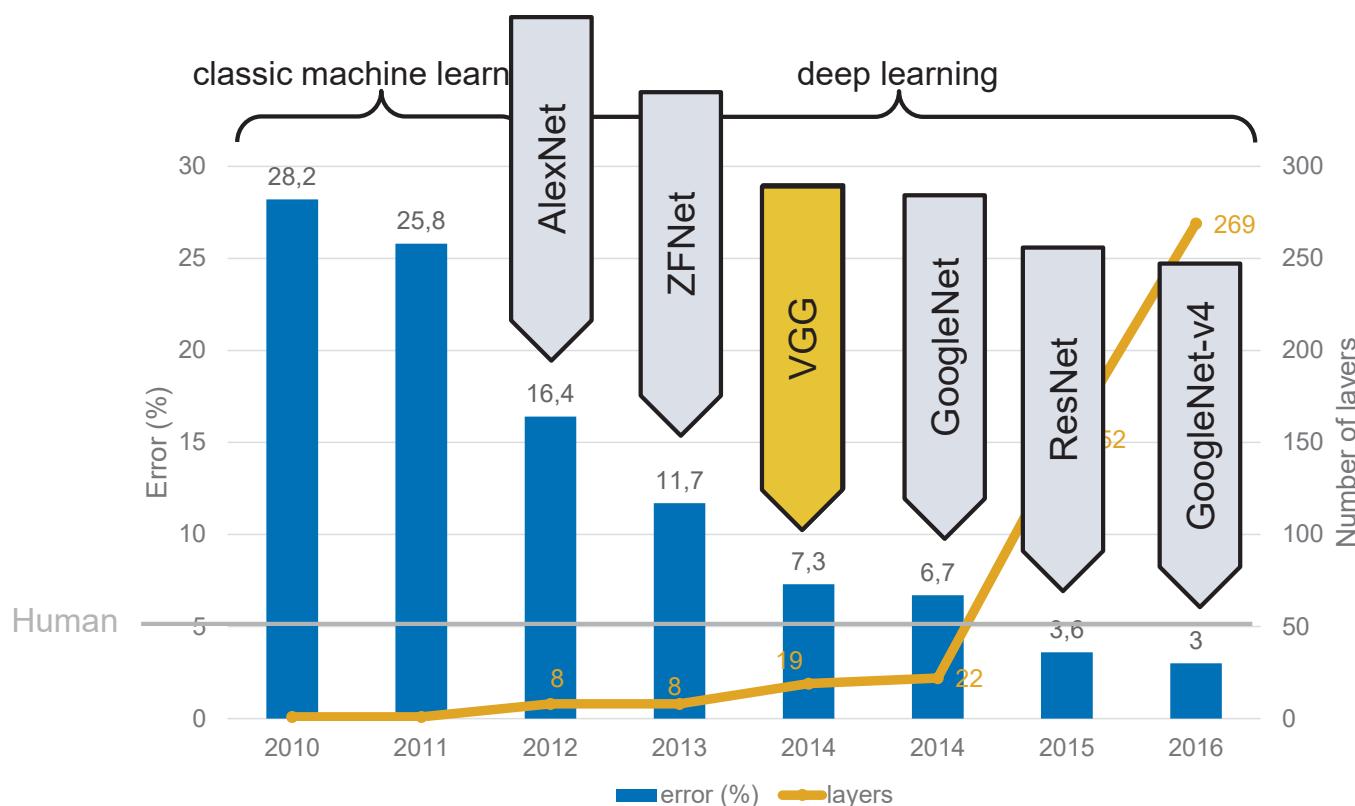
$$\text{Parameters} = \text{OutputDepth} \cdot (1 + \text{KernelWidth} \cdot \text{KernelHeight} \cdot \text{InputDepth})$$



→ Great Success of Deep Learning in ImageNet Challenge!

## ZFNet

- ▶ Zeiler and Fergus, 2013
- ▶ Like AlexNet but:
  - ▶ Conv1: change from (11x11 stride 4) to (7x7 stride 2)
  - ▶ Conv3,4,5: instead of 384, 384, 256 filters use 512, 1024, 512
  - ▶ Refined hyperparameters
- ▶ Jump in performance



→ Great Success of Deep Learning in ImageNet Challenge!

Auto.-Sys: Deep Learning

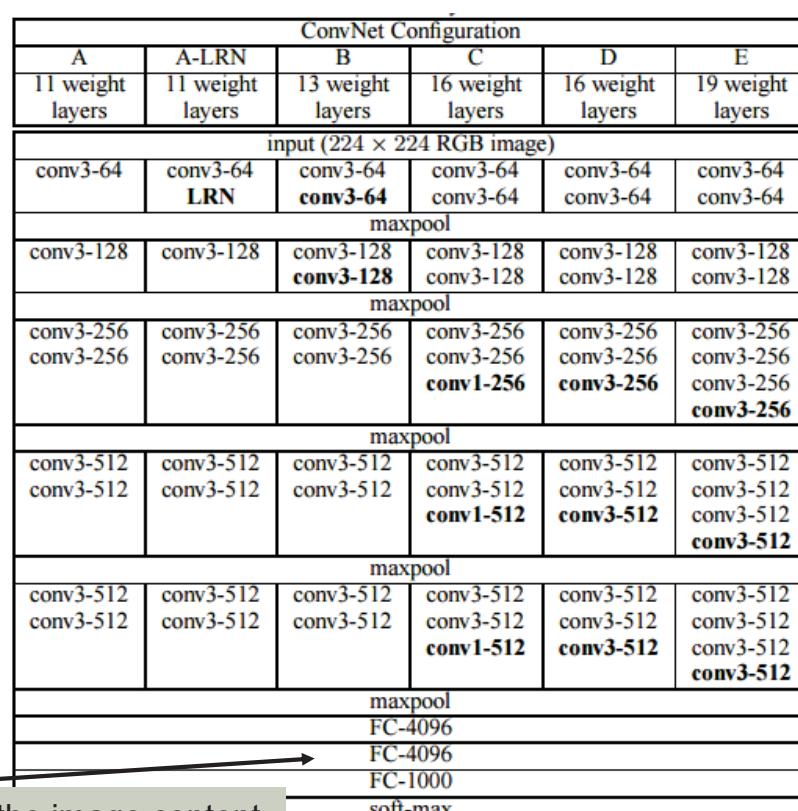
Source: [http://image-net.org/challenges/talks\\_2017/ILSVRC2017\\_overview.pdf](http://image-net.org/challenges/talks_2017/ILSVRC2017_overview.pdf),  
05.02.2017

Prof. Dr. N. Stache

9

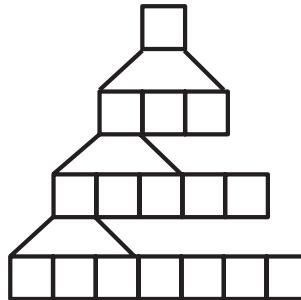
## VGGNet

- ▶ Simonyan and Zisserman, 2014
- ▶ Typ: VGG16, VGG19
- ▶ Idea: smaller filters, more depth
- ▶ Only 3x3 conv stride1, pad1 and 2x3 MAX-Pool stride 2
- ▶ Difficult to train due to depth  
→ first trained an 11 layer model and inserted additional layers
- ▶ Today: Batch-Normalization helps training the network (was not invented in 2014)
- ▶ Q: Why use small filters, e.g. three 3x3 filters?



Good output feature describing the image content,  
can ideally be used for other classification tasks

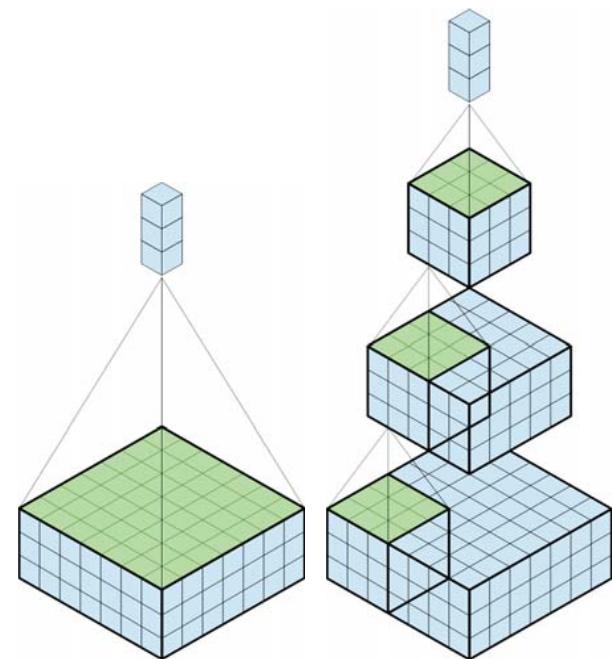
- A: A stack of three 3x3 conv filters, stride 1 equals to a receptive field of 7x7



- But:

- It needs fewer parameters:  
 $3 * (3^2 C^2)$  vs.  $7^2 C^2$  for C input and output channels per layer,  
i.e. 243 vs. 441 for C=3
- Has more nonlinearities compared to 7x7 filter
- **Both is a pro!**

Int. to ML - DL



Prof. Dr.-Ing. N.  
Stache

11

## Memory and parameter distribution in VGG16

INPUT: [224x224x3]	memory: $224*224*3=150K$	params: 0	(not counting biases)
CONV3-64: [224x224x64]	memory: $224*224*64=3.2M$	params: $(3*3*3)*64 = 1,728$	
CONV3-64: [224x224x64]	memory: $224*224*64=3.2M$	params: $(3*3*64)*64 = 36,864$	
POOL2: [112x112x64]	memory: $112*112*64=800K$	params: 0	
CONV3-128: [112x112x128]	memory: $112*112*128=1.6M$	params: $(3*3*64)*128 = 73,728$	
CONV3-128: [112x112x128]	memory: $112*112*128=1.6M$	params: $(3*3*128)*128 = 147,456$	
POOL2: [56x56x128]	memory: $56*56*128=400K$	params: 0	
CONV3-256: [56x56x256]	memory: $56*56*256=800K$	params: $(3*3*128)*256 = 294,912$	
CONV3-256: [56x56x256]	memory: $56*56*256=800K$	params: $(3*3*256)*256 = 589,824$	
CONV3-256: [56x56x256]	memory: $56*56*256=800K$	params: $(3*3*256)*256 = 589,824$	
POOL2: [28x28x256]	memory: $28*28*256=200K$	params: 0	
CONV3-512: [28x28x512]	memory: $28*28*512=400K$	params: $(3*3*256)*512 = 1,179,648$	
CONV3-512: [28x28x512]	memory: $28*28*512=400K$	params: $(3*3*512)*512 = 2,359,296$	
CONV3-512: [28x28x512]	memory: $28*28*512=400K$	params: $(3*3*512)*512 = 2,359,296$	
POOL2: [14x14x512]	memory: $14*14*512=100K$	params: 0	
CONV3-512: [14x14x512]	memory: $14*14*512=100K$	params: $(3*3*512)*512 = 2,359,296$	
CONV3-512: [14x14x512]	memory: $14*14*512=100K$	params: $(3*3*512)*512 = 2,359,296$	
CONV3-512: [14x14x512]	memory: $14*14*512=100K$	params: $(3*3*512)*512 = 2,359,296$	
POOL2: [7x7x512]	memory: $7*7*512=25K$	params: 0	
FC: [1x1x4096]	memory: $4096$	params: $7*7*512*4096 = 102,760,448$	
FC: [1x1x4096]	memory: $4096$	params: $4096*4096 = 16,777,216$	
FC: [1x1x1000]	memory: $1000$	params: $4096*1000 = 4,096,000$	

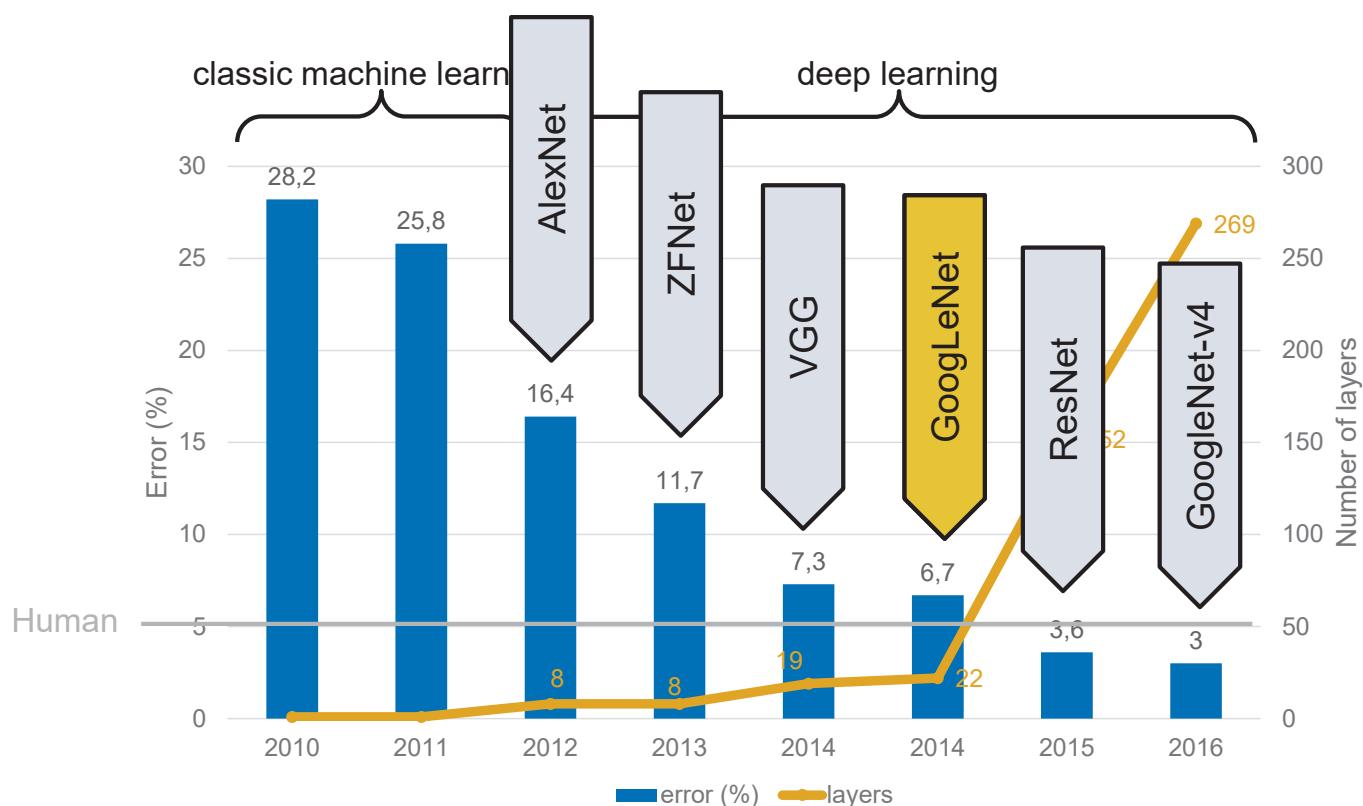
Note:

Most memory is in early CONV

Most params are in late FC

**TOTAL memory: 24M \* 4 bytes ~ 96MB / image (only forward! ~\*2 for bwd)**

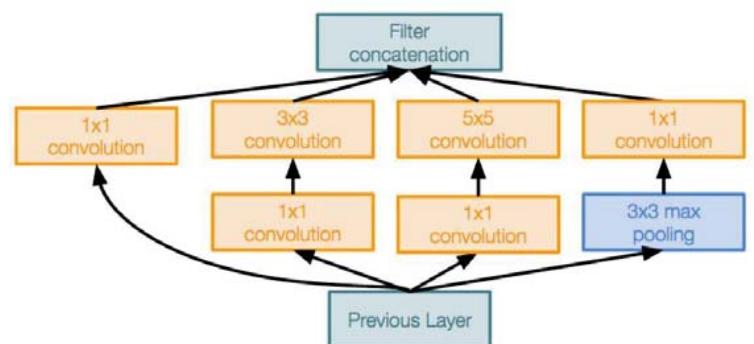
**TOTAL params: 138M parameters**



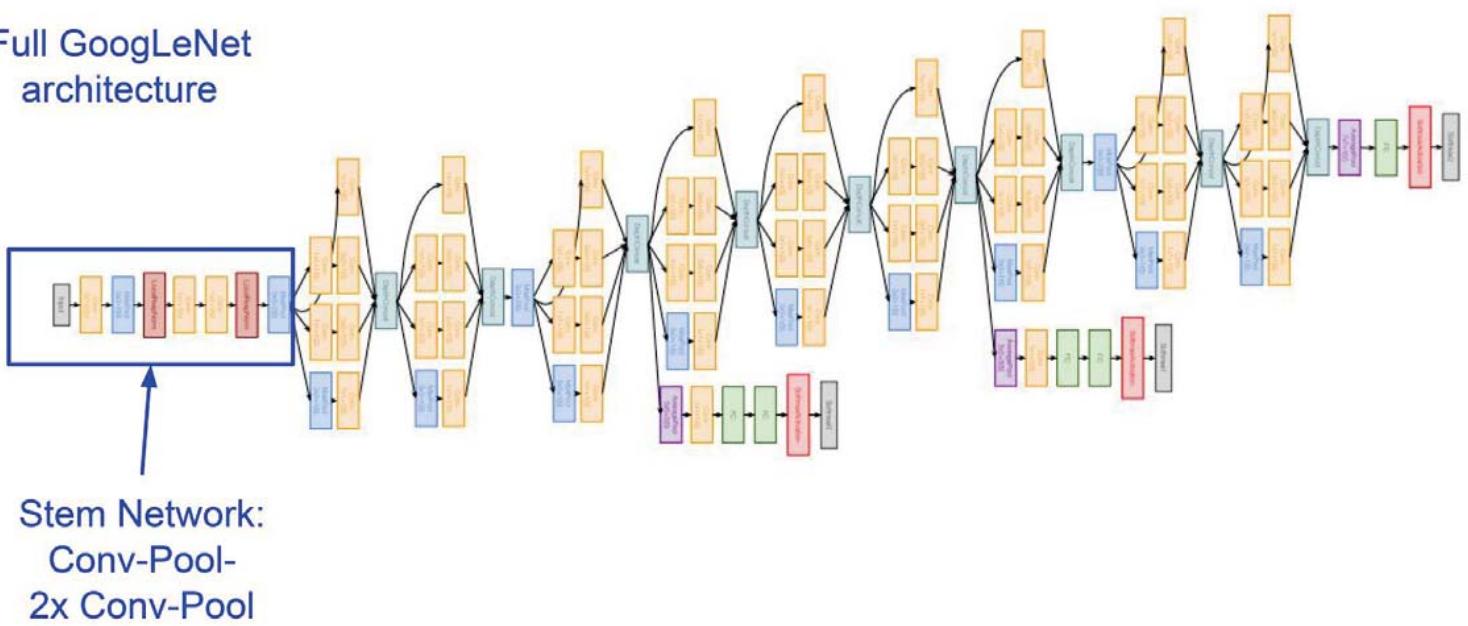
→ Great Success of Deep Learning in ImageNet Challenge!

## GoogLeNet

- ▶ Szegedy et al., 2014
- ▶ 22 layers
- ▶ Uses stacked inception modules
- ▶ Reduced amount of fully connected layers (only 1 in the network trunk)
- ▶ Only 5 million parameters 12x less than AlexNet



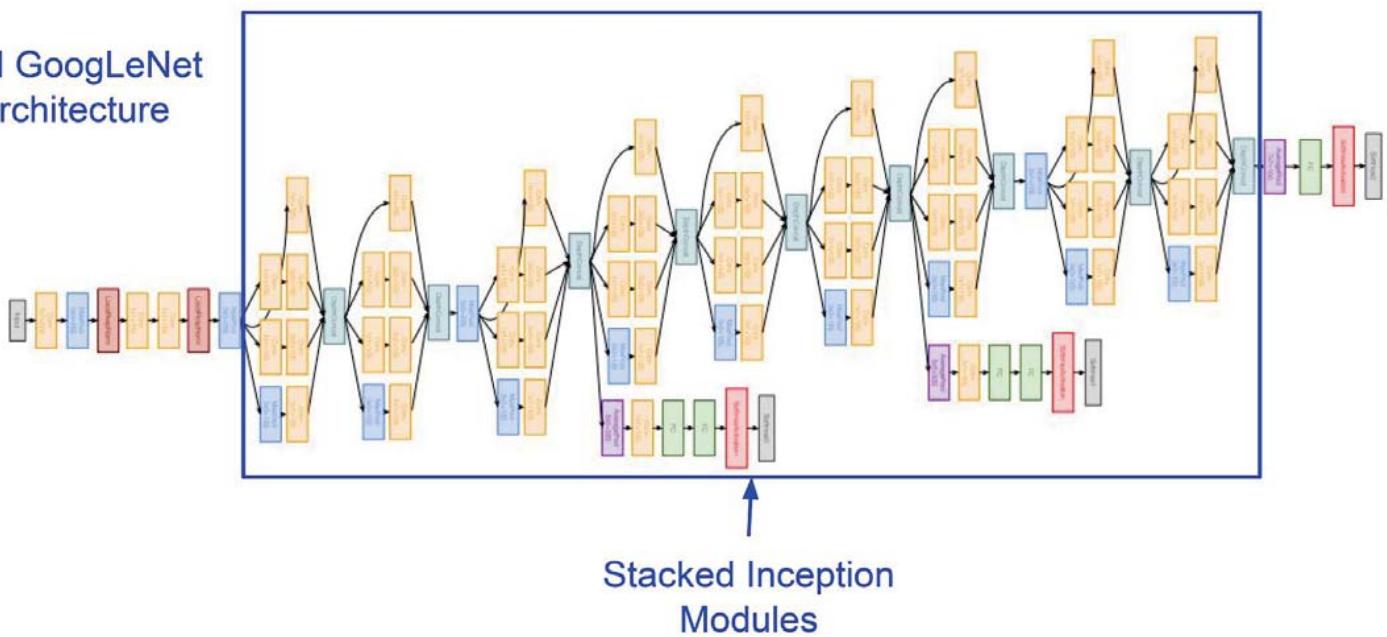
Full GoogLeNet architecture



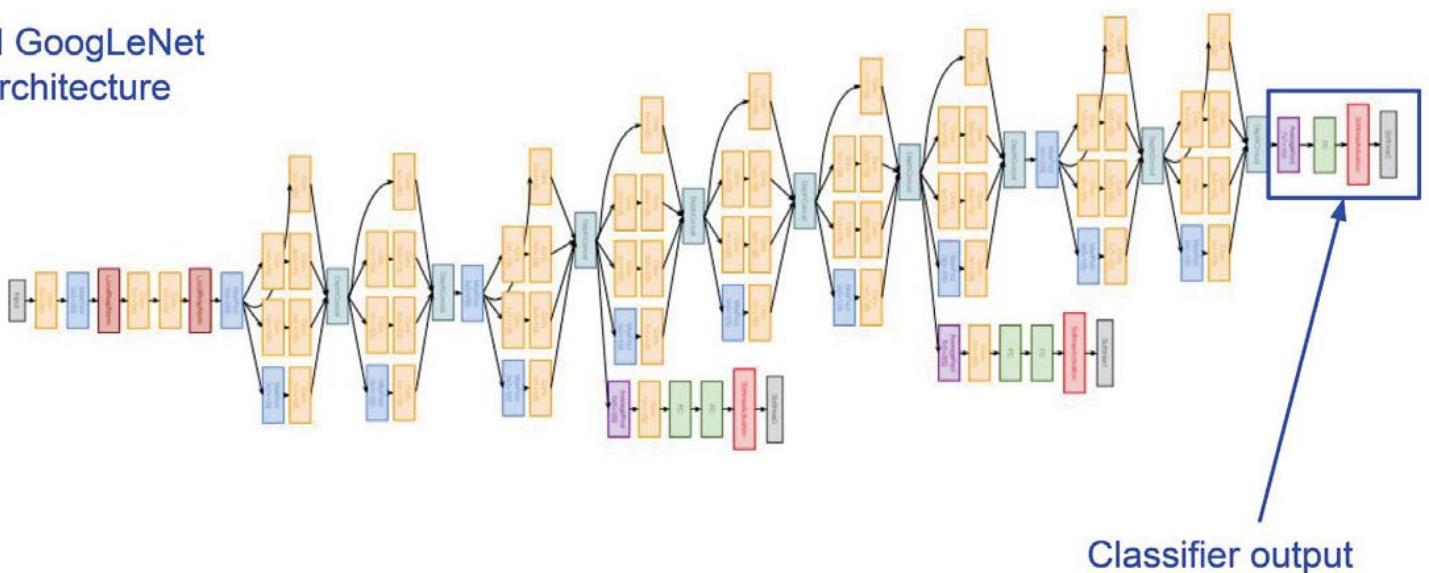
Stem Network:  
Conv-Pool-  
2x Conv-Pool

# GoogLeNet

Full GoogLeNet architecture

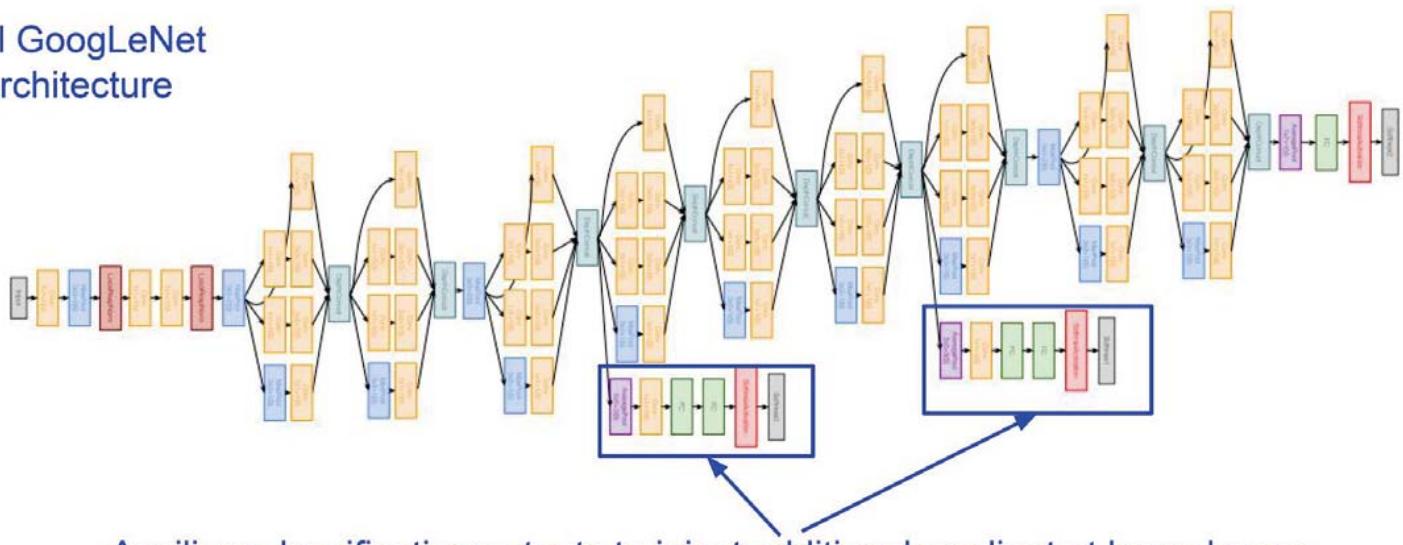


Full GoogLeNet architecture



# GoogLeNet

Full GoogLeNet architecture



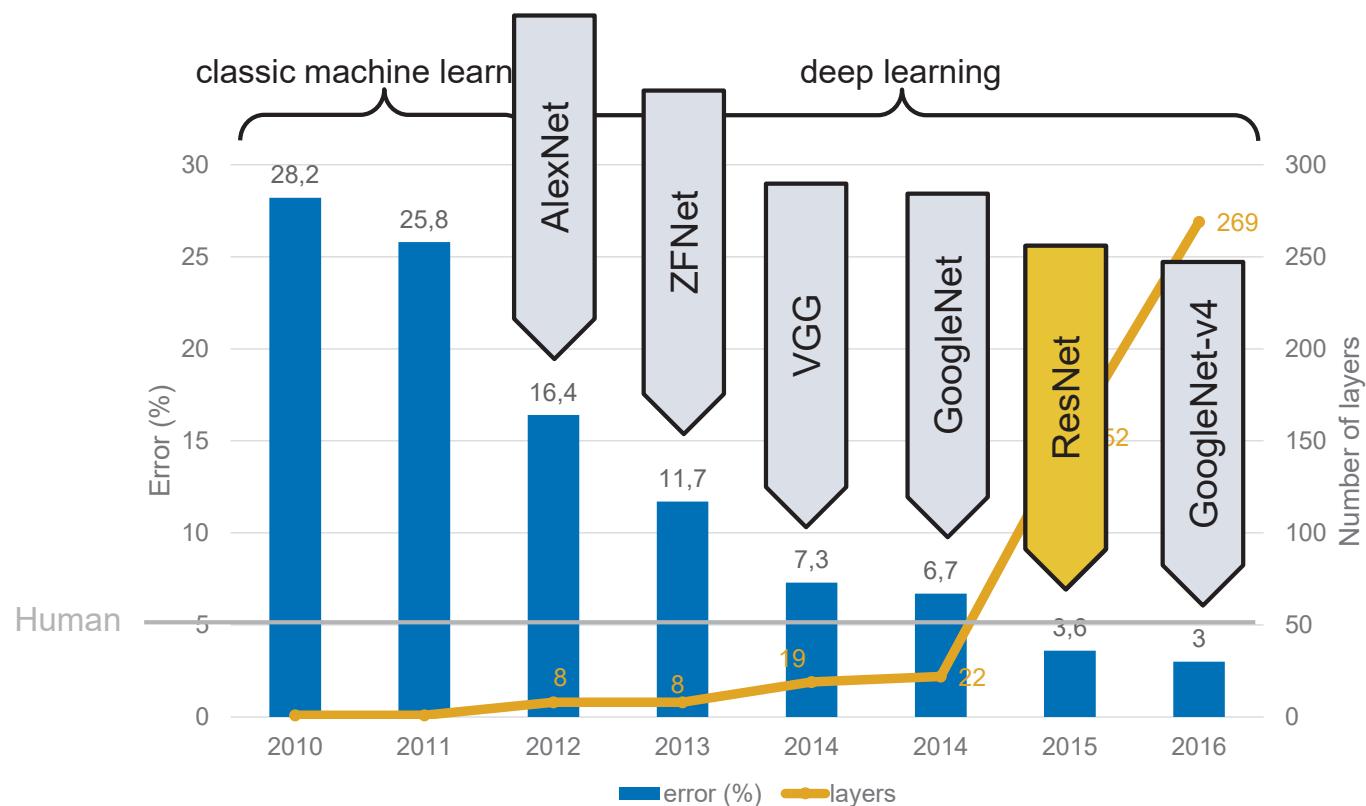
Auxiliary classification outputs to inject additional gradient at lower layers  
 (AvgPool-1x1Conv-FC-FC-Softmax)

Only used during training. Loss for optimization is the weighted average loss of all the outputs (auxiliary outputs weighted with 0.3)  
 For inference, auxiliary output are discarded.

Today auxiliary outputs are obsolete due to batch normalization

- ▶ Training a network is not an easy task
  - ▶ Auxiliary outputs are injected just to improve optimization
  - ▶ Network is trained with fewer layers and additional layers are injected later
  - ▶ Sensitivity to parameter initialization
  - ▶ Use of the best optimization algorithms
  - ▶ Learning rate has to be rather small + ideally adaptive
  - ▶ Use of heavy computational power
  - ▶ ...
- ▶ Conclusion: “Good flow of gradients is essential to be successful with deep neural networks”

## ILSVRC Classification top-5 error (%)



→ Great Success of Deep Learning in ImageNet Challenge!

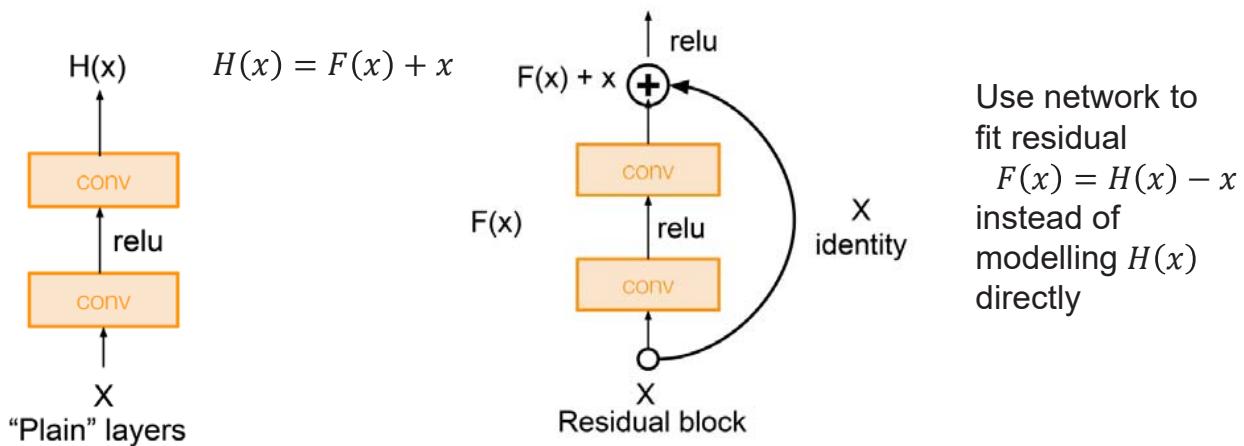
- ▶ He et al., 2015
- ▶ 152 layers deep model for Image Net classification
- ▶ Observation:



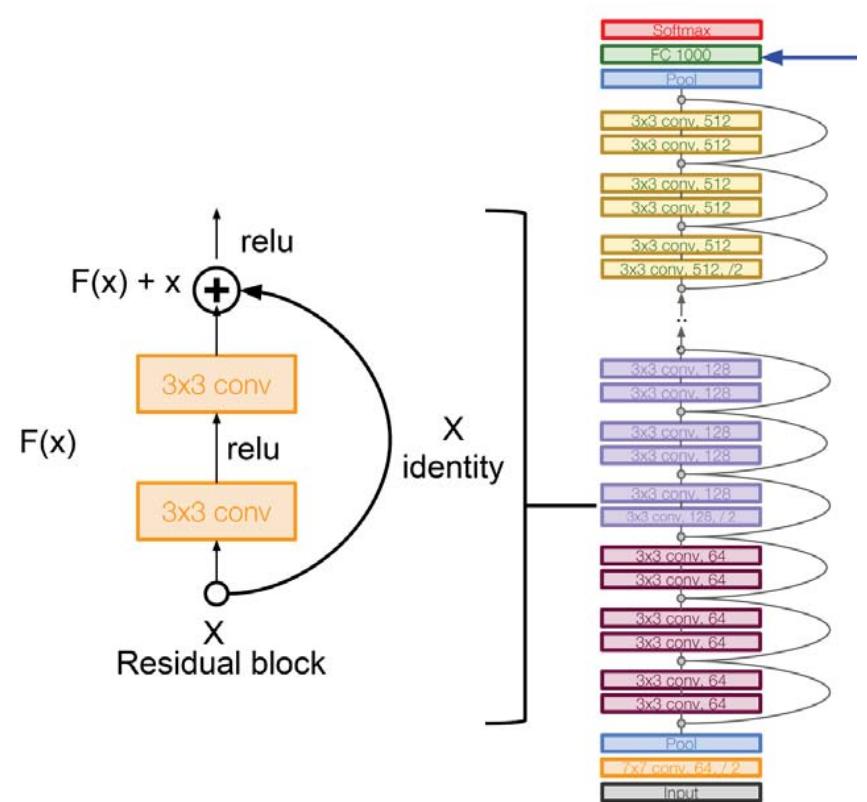
- ▶ Deep network is worse compared to more shallow network for both training and testing
- ▶ This is not overfitting as the training error is also worse
- ▶ Hypothesis: Optimization problem, deeper models are harder to optimize!

# ResNet

- ▶ Approach:
  - ▶ Create a network which composes the output as sum of input and modelled output
  - ▶ If the network does nothing → output = input
  - ▶ Otherwise: Network models the residual  $F(x)$  (hence: ResNet)

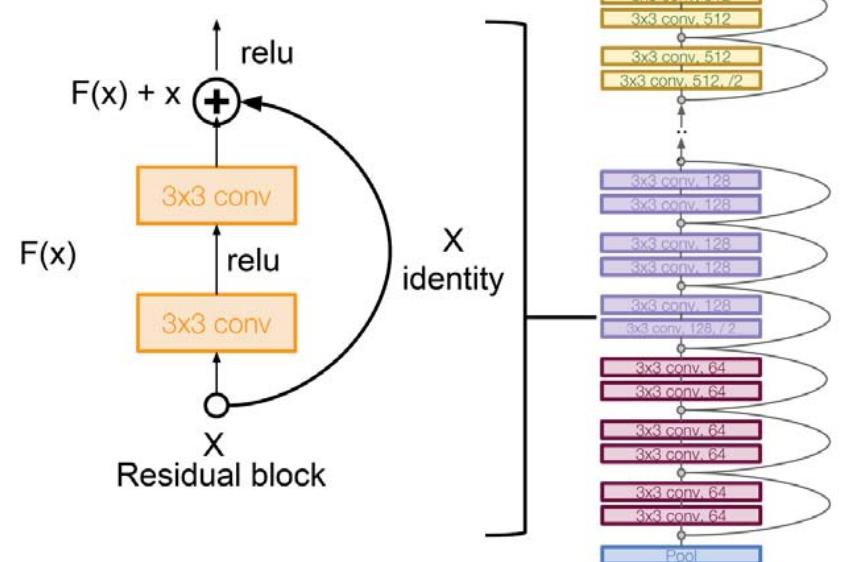


- ▶ Stacked residual blocks
- ▶ Two 3x3 filters per block
- ▶ Periodically double # of filters and downsample by stride 2
- ▶ Additional conv layer at the beginning
- ▶ Global average pooling layer after last conv layer
- ▶ Only one fully connected layer at the end



# ResNet

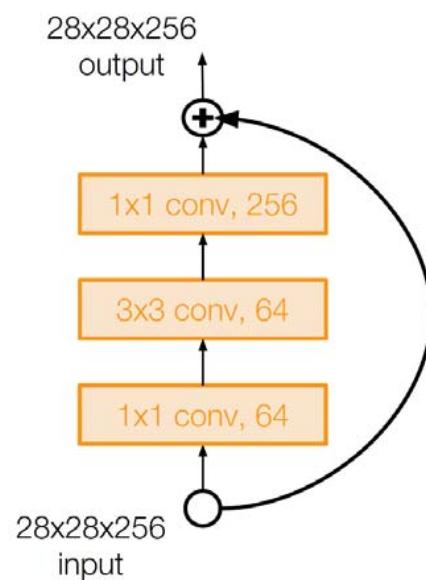
- ▶ If all weights are 0 → output is the identity
- ▶ Easy to learn not to use the layers that it does not need
- ▶ L2 regularization makes weights small (does not make sense in normal conv layers)  
→ drives towards identity  
→ forces the network not to train weights which are not needed
- ▶ Gradient backprop works well due to direct connection → acts as gradient highway



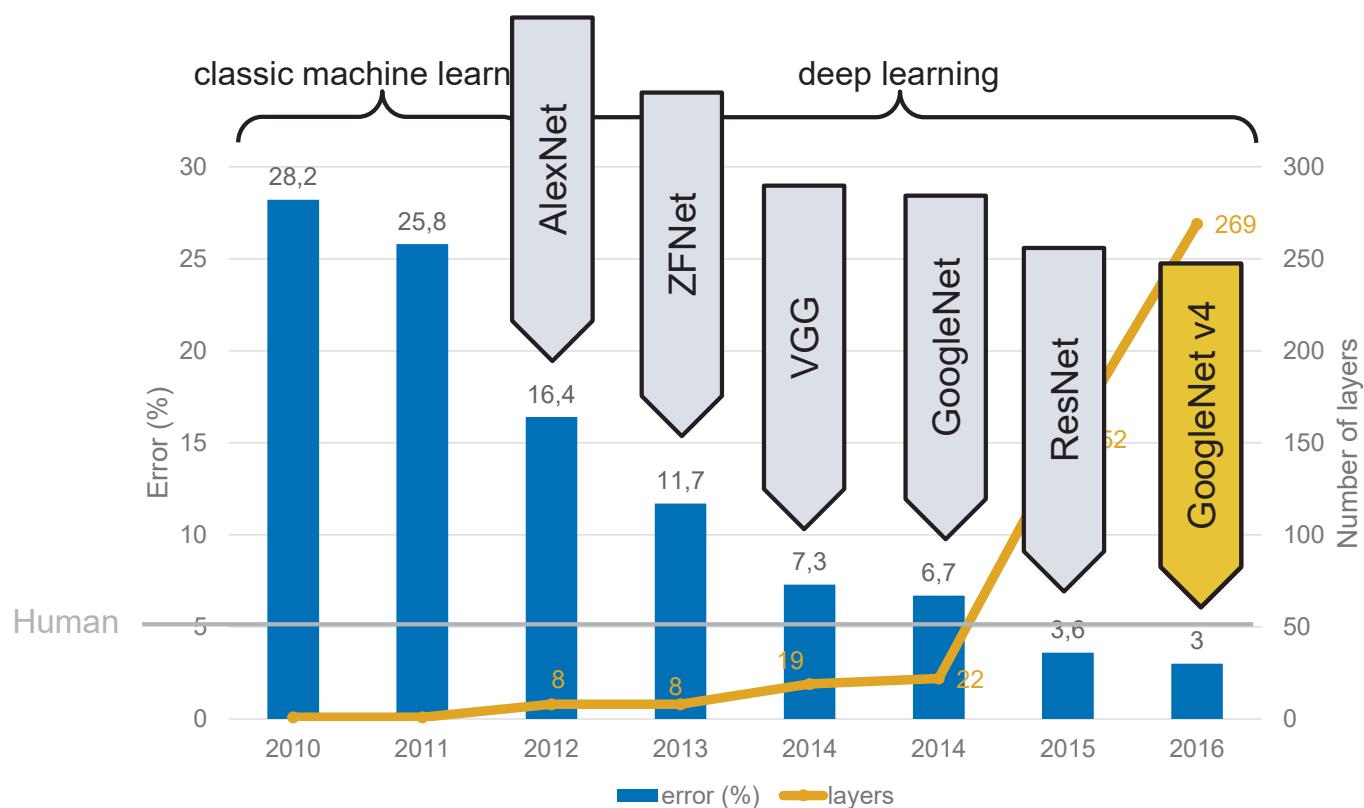
- ▶ For deep networks > 50 layers, use “bottleneck” layer ( $1 \times 1$  convolution) to improve efficiency

Training details:

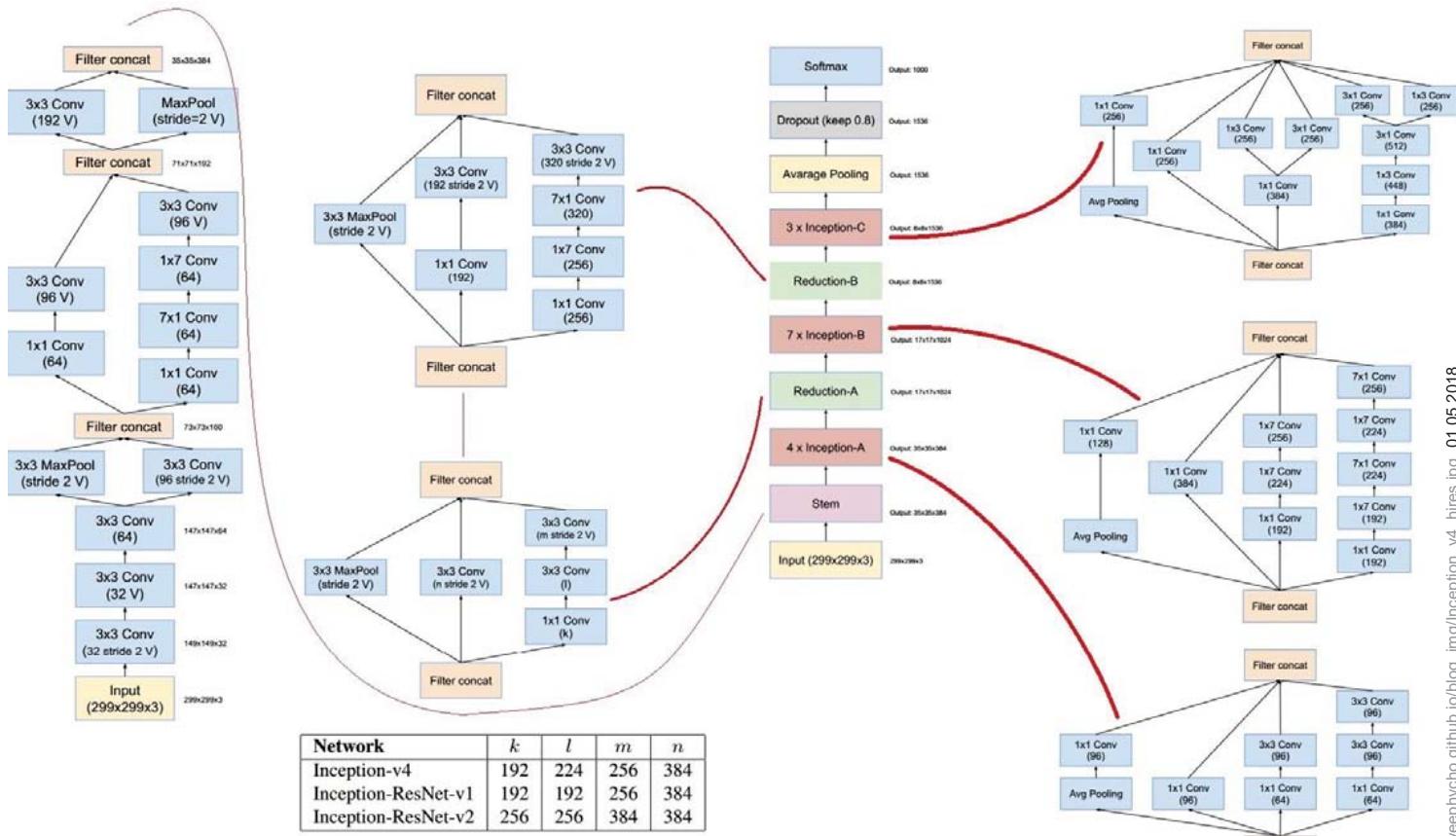
- ▶ Batch normalization after every Conv layer
- ▶ SGD + Momentum
- ▶ Learning rate modification when validation error plateaus
- ▶ Batch size 256
- ▶ No dropout used



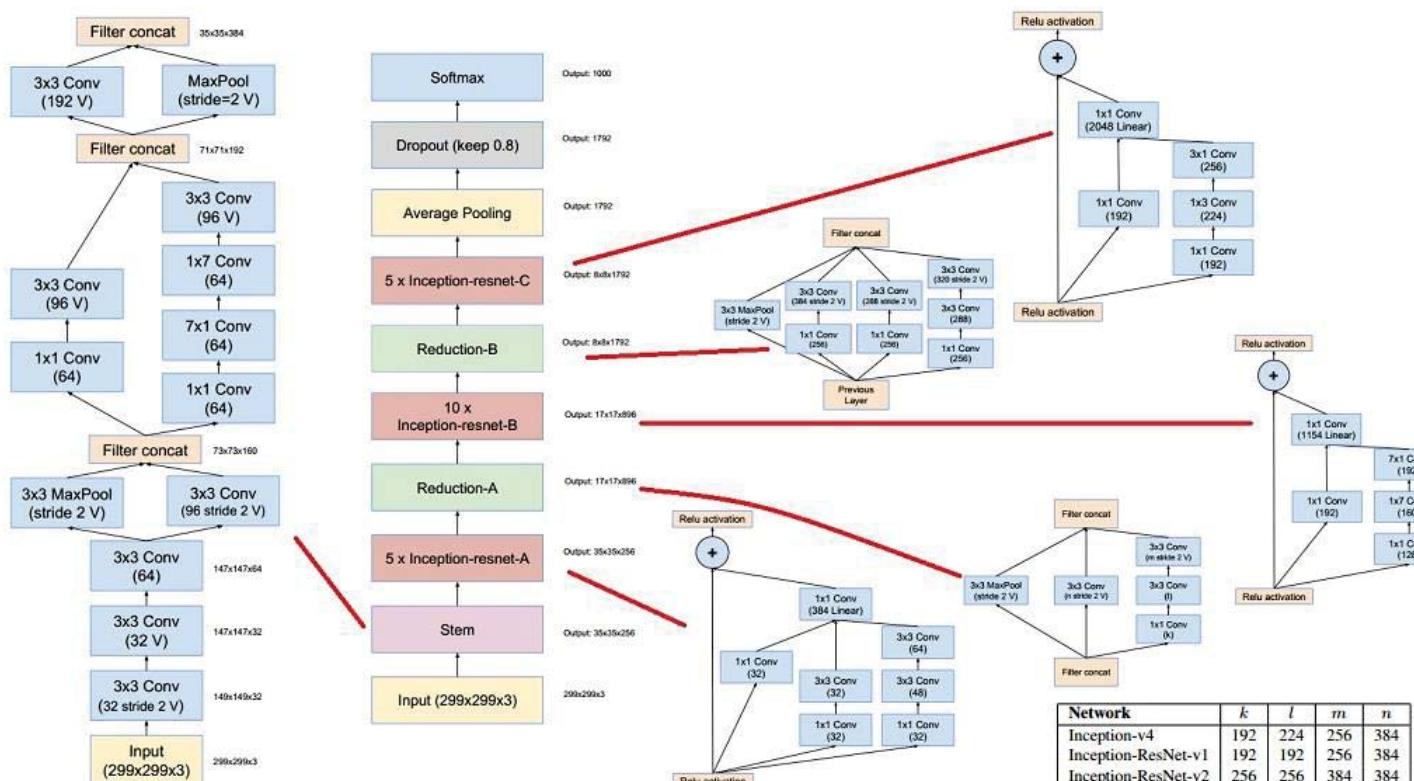
## ILSVRC Classification top-5 error (%)

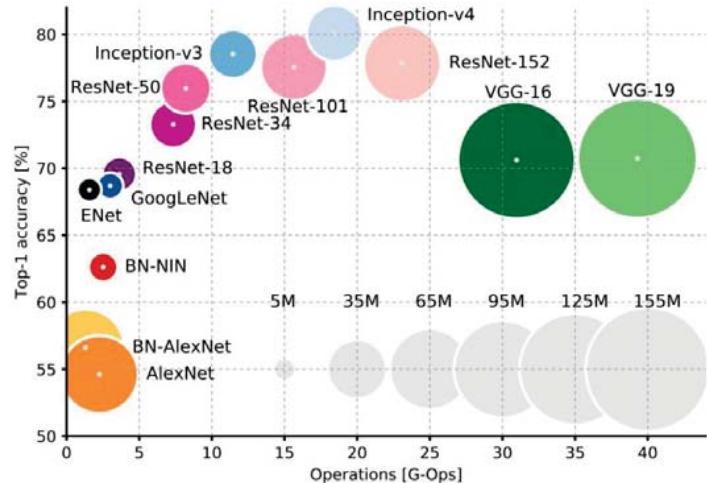
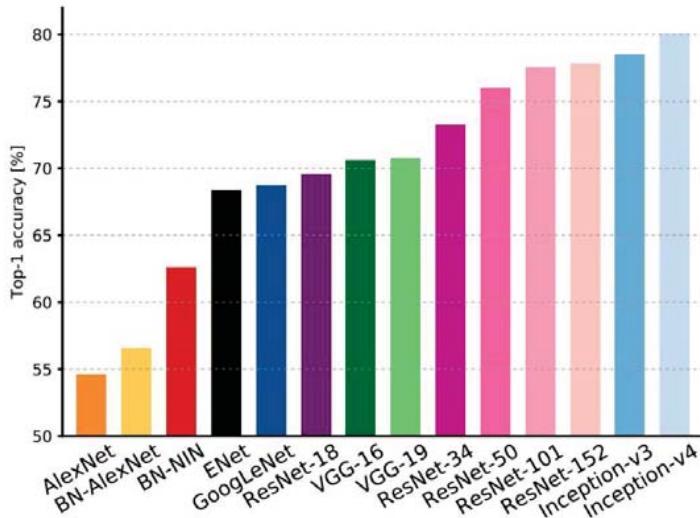


→ Great Success of Deep Learning in ImageNet Challenge!



## Inception-ResNet





An Analysis of Deep Neural Network Models for Practical Applications, 2017.

Figures copyright Alfredo Canziani, Adam Paszke, Eugenio Culurciello, 2017.

Auto.-Sys: Deep Learning

Li, Johnson, Yeung: Cs231n\_2017\_lecture9, Stanford, 2017.

Prof. Dr. N. Stache

29

## Outline

► Famous network architectures

► Batch normalization

► Transfer learning

► Semantic segmentation

## Batch normalization

## ► Input normalization:



loffe, Szegedy: Batch Normalization: Acceleration Deep Network Training by Reducing Internal Covariate Shift, arXiv:1502.03167v3 [cs.LG] 2 Mar 2015

## ► Batch normalization:

- ▶ Normalization the inputs to each hidden layer
  - ▶ Lower sensitivity to initial values of weights
  - ▶ Allows for higher learning rates, since optimization problem gets a bit nicer
  - ▶ Acts as form of regularization since inputs are influenced by other values in the batch due to normalization process

# Batch normalization

- ▶ Use a batch of  $n$  observations with  $f$  features

The diagram illustrates a neural network architecture. At the bottom, five input features are shown as downward arrows labeled  $\mu_1, \sigma_1^2$ ,  $\mu_2, \sigma_2^2$ ,  $\mu_3, \sigma_3^2$ ,  $\mu_4, \sigma_4^2$ , and  $\mu_5, \sigma_5^2$ . These features feed into a hidden layer represented by a grid of 5 columns and  $n$  rows. The columns are labeled **f1**, **f2**, **f3**, **f4**, and **f5**. Above the grid, a double-headed arrow indicates a width of  $f$ .

$$\begin{aligned}\mu_{\mathcal{B}} &\leftarrow \frac{1}{m} \sum_{i=1}^m x_i \\ \sigma_{\mathcal{B}}^2 &\leftarrow \frac{1}{m} \sum_{i=1}^m (x_i - \mu_{\mathcal{B}})^2 \\ \hat{x}_i &\leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}}\end{aligned}$$

↑

Normalized term

- Normalized inputs are not directly applied

- Instead,

$$y_i \leftarrow \gamma \hat{x}_i + \beta \equiv \text{BN}_{\gamma, \beta}(x_i), \quad \hat{x}_i = \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}$$

is used.

- $\gamma$  and  $\beta$  are parameters which are learned

- Note: Network can recover identity mapping by

$$\gamma = \sqrt{\sigma^2 + \epsilon}, \quad \beta = \mu_B$$

- For inference, batch normalization uses fixed values for  $\mu$  and  $\sigma^2$ , based on entire dataset.

# Batch normalization

**Input:** Values of  $x$  over a mini-batch:  $\mathcal{B} = \{x_{1\dots m}\}$ ;

Parameters to be learned:  $\gamma, \beta$

**Output:**  $\{y_i = \text{BN}_{\gamma, \beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m} \sum_{i=1}^m x_i \quad // \text{mini-batch mean}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m} \sum_{i=1}^m (x_i - \mu_{\mathcal{B}})^2 \quad // \text{mini-batch variance}$$

$$\hat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \quad // \text{normalize}$$

$$y_i \leftarrow \gamma \hat{x}_i + \beta \equiv \text{BN}_{\gamma, \beta}(x_i) \quad // \text{scale and shift}$$

► Famous network architectures

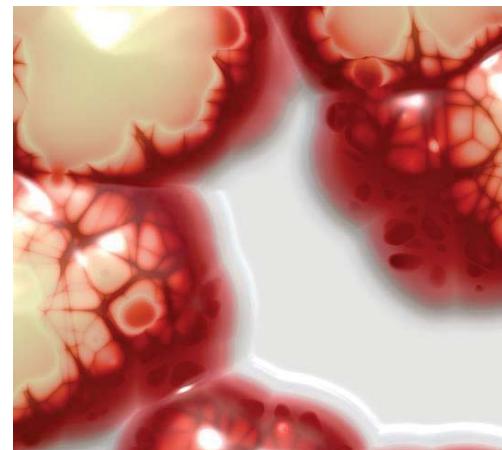
► Batch normalization

► Transfer learning

► Semantic segmentation

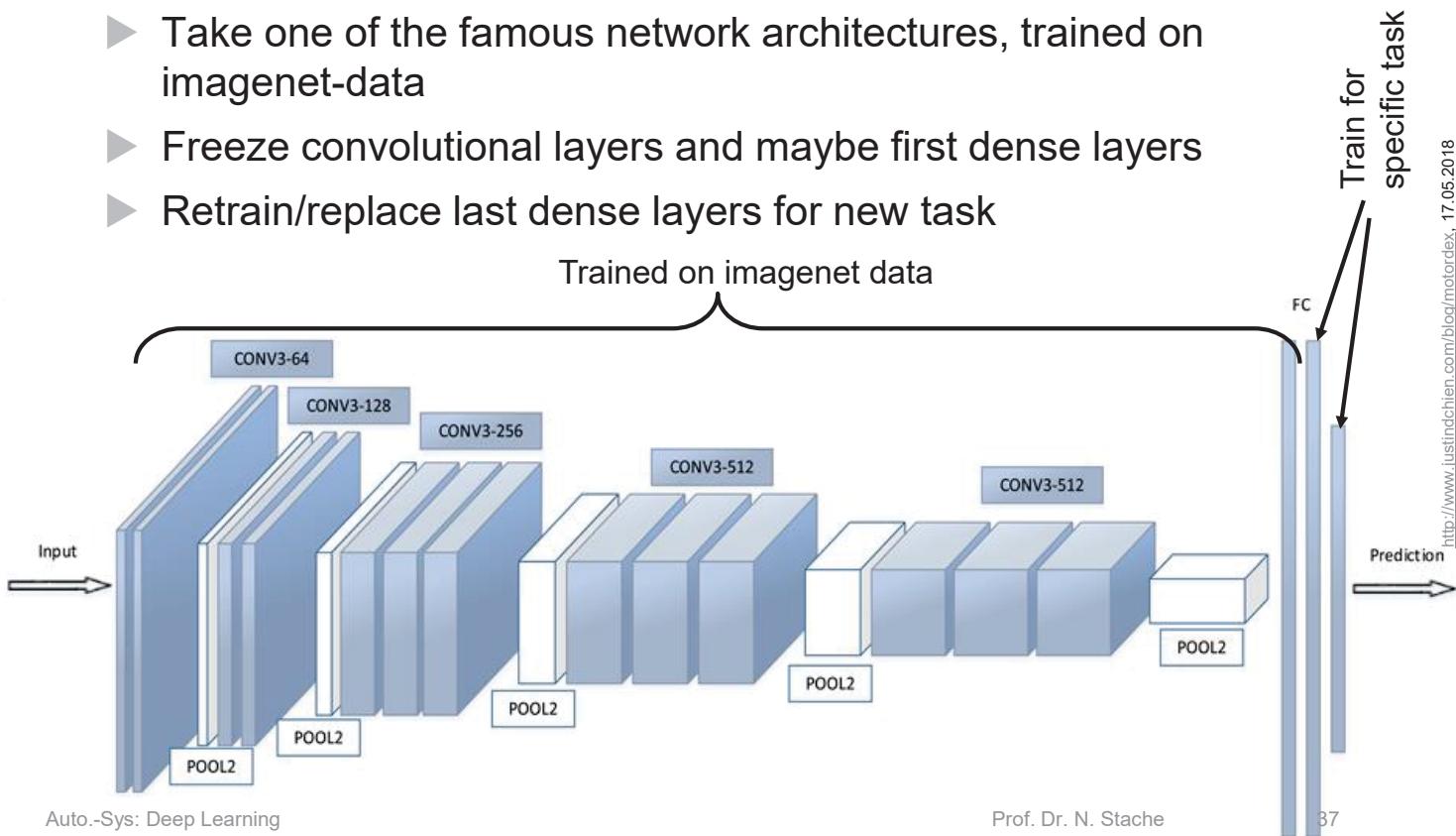
## Transfer learning

- Re-use pre-trained model for different task (with only slight changes)
- E.g.: Use model trained on traffic signs for classification of cancer cells



► Widely used:

- Take one of the famous network architectures, trained on imangenet-data
- Freeze convolutional layers and maybe first dense layers
- Retrain/replace last dense layers for new task



## Hands-on Example

- Transfer learning using VGG 16,
  - Use pretrained network on image net data for fruit classification
- 
- Check out form ILIAS, keras folder:
  - Transferlearning with VGG16 and a Fruit Dataset

- ▶ Famous network architectures
- ▶ Batch normalization
- ▶ Transfer learning

## ▶ Semantic segmentation

# Image Classification

## Image Classification:

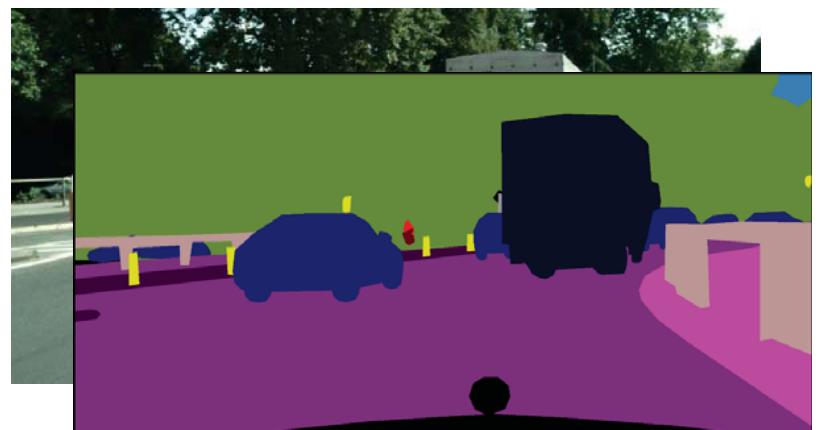
- ▶ Images show precisely one object
- ▶ Image (i.e. object) is classified, using a CNN
- ▶ Not considered:
  - ▶ Other content in the image
  - ▶ Position of the object



Source:  
<http://benchmark.ini.rub.de/?section=dataset&subsection=dataset>  
19.09.2017

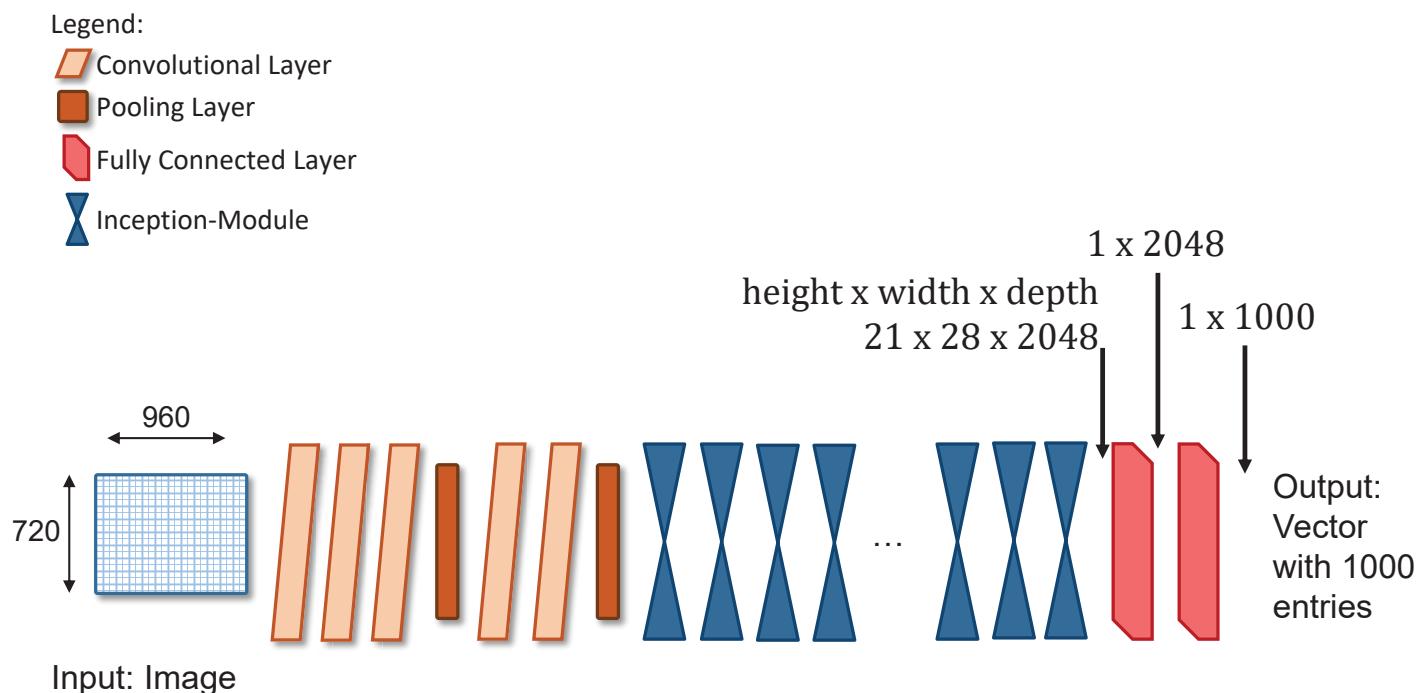
## Semantic Segmentation:

- ▶ Classification to which object class each pixel belongs
- ▶ All content is classified
- ▶ Position information is kept



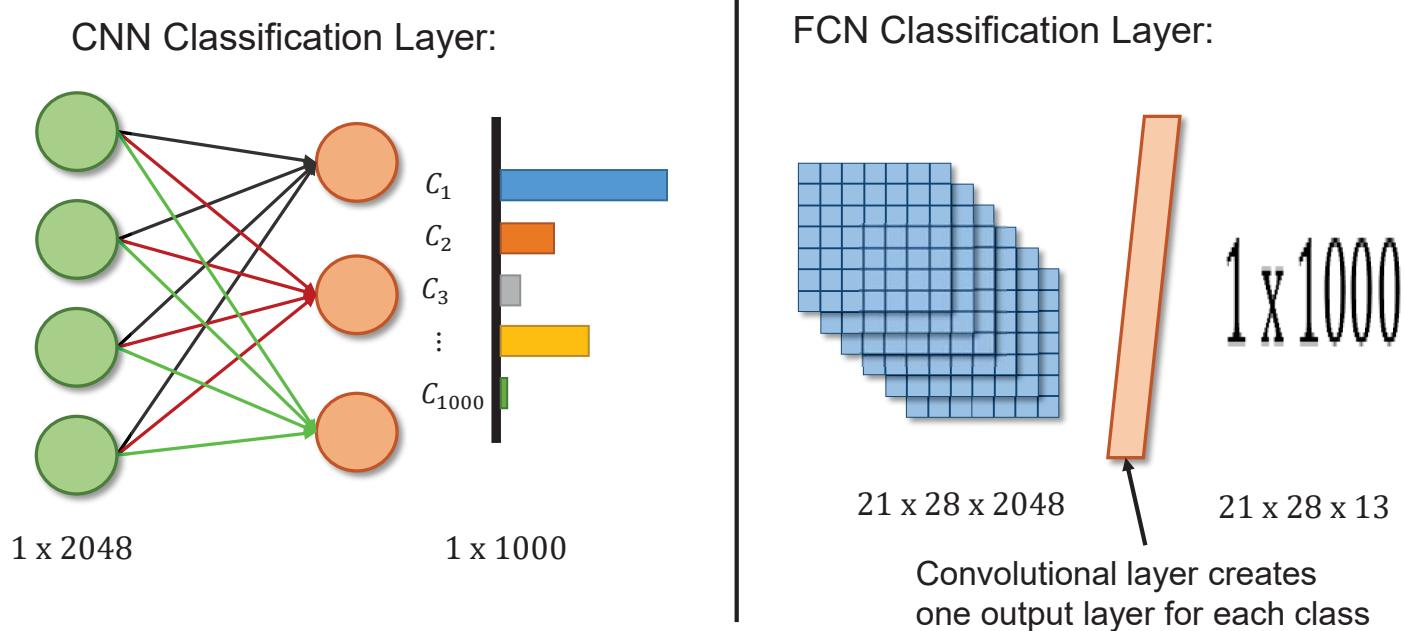
Source: [www.cityscapes-dataset.com](http://www.cityscapes-dataset.com), 07.04.2019

# Example: Inception Architecture

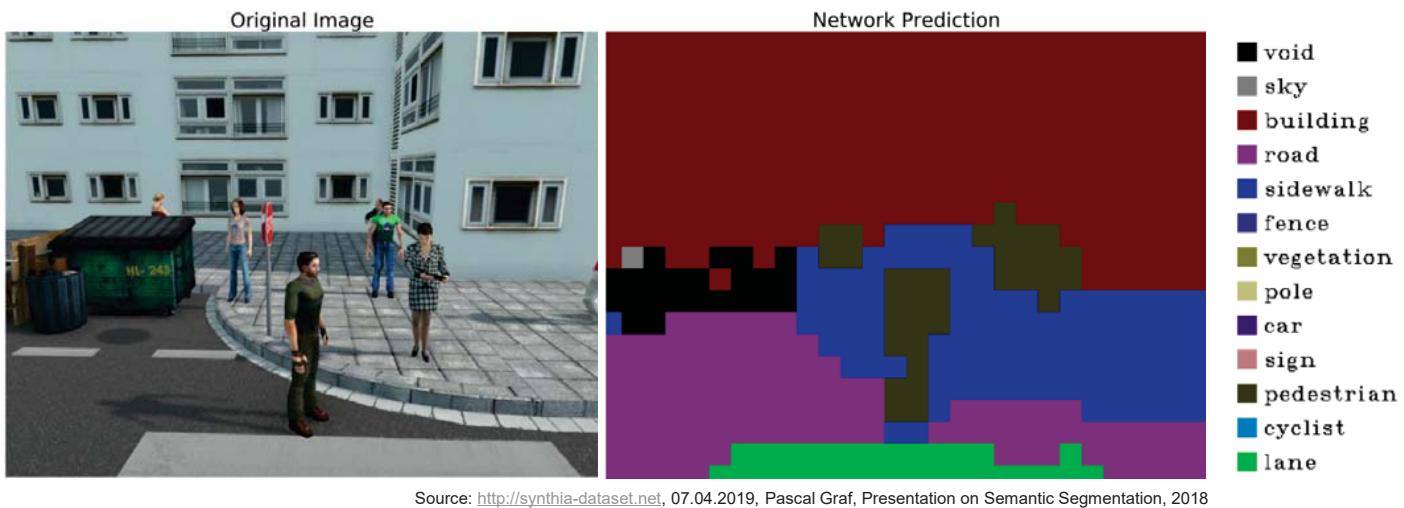


- Observation: Rough 2D information is present after the last inception module but gets lost in the fully connected layers

## Idea Fully Convolutional Network (FCN)

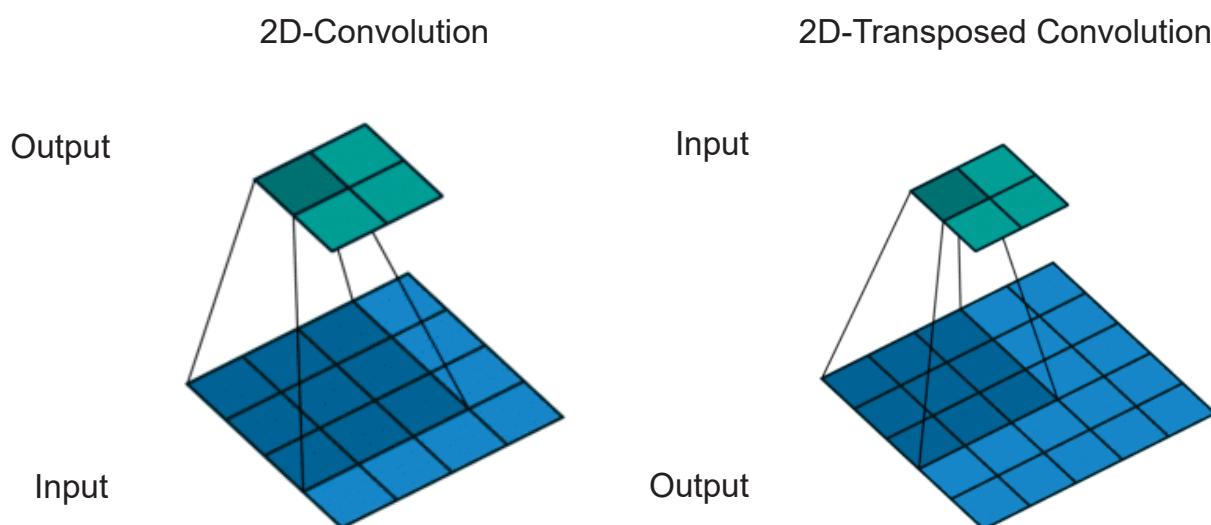


- Final step:  
Upsampling of the output layers to the size of the input image!



- ▶ Network prediction shows the max values of the 13 output maps in an color-coded overlay
- ▶ Result looks boxy, 2D-resolution is low

## Transposed Convolution



One approach to transposed convolution:

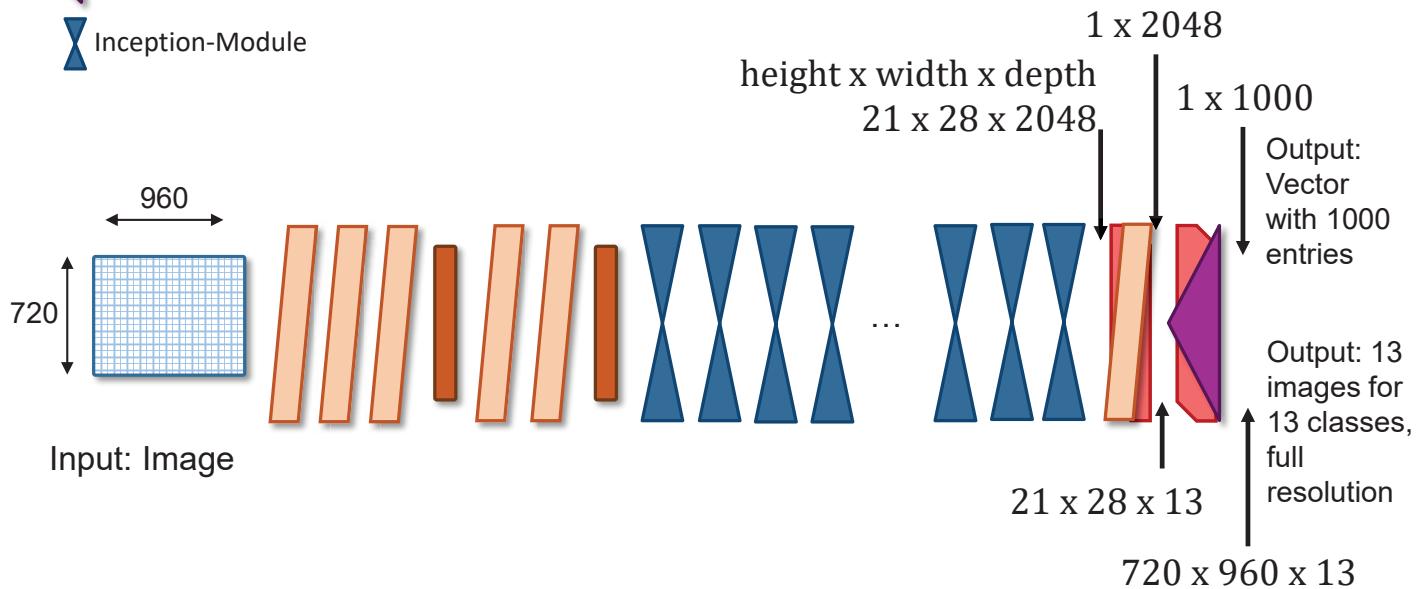
- ▶ Multiply input pixel with learned filter matrix and paste this to the output
- ▶ Move in the output with a stride = 2 while moving on the input with stride = 1
- ▶ Result is less boxy, but resolution is still low

# Modify Inception Architecture

Legend:

- Convolutional Layer
- Pooling Layer
- Fully Connected Layer
- Transposed Conv Layer
- Inception-Module

Image Source: Pascal Graf, Presentation on Semantic Segmentation, 2018



## Comparison

Earlier Stages of the Network	Later Stages of the Network
<p>Higher 2D resolution</p> <ul style="list-style-type: none"> <li>• More information from orig. image</li> <li>• Better reconstruction of the image structure</li> <li>• Higher accuracy in segmentation</li> </ul>	<p>Higher classification accuracy</p> <ul style="list-style-type: none"> <li>• Data has been processed by more layers</li> <li>• Contain information from a larger image radius</li> <li>• Better understanding of image semantics</li> </ul>

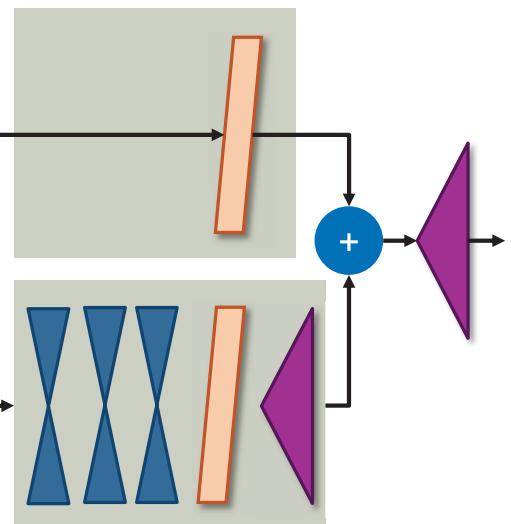
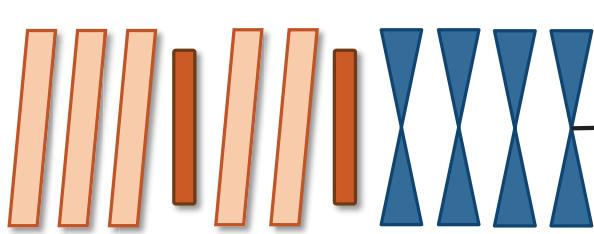
Idea: Combine later stages and earlier stages of the network for  
 → High 2D-resolution and  
 → High classification accuracy

Legend:

- Convolutional Layer
- Pooling Layer
- Fully Connected Layer
- Transposed Conv Layer
- Inception-Module

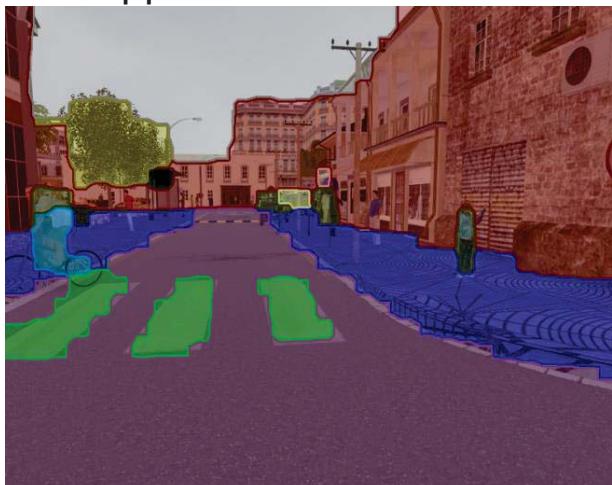
960  
720

Input: Image

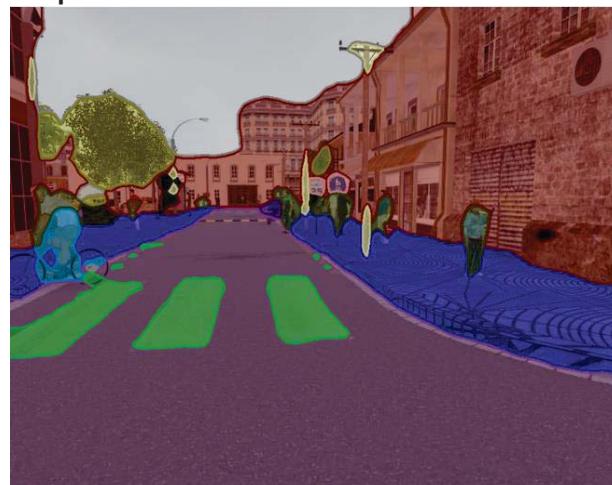


## Result

First approach:



Improved architecture:





# Autonomous Systems: Deep Learning

## 8. Recurrent Neural Networks

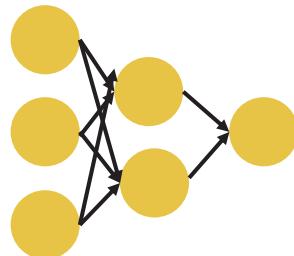


Prof. Dr.-Ing. Nicolaj Stache  
Heilbronn University of Applied Sciences

### Motivation

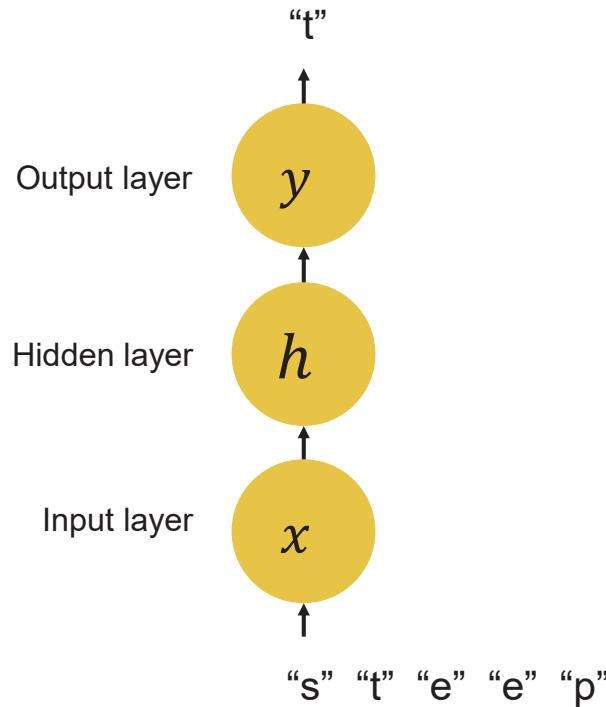


- ▶ So far: Feed forward networks



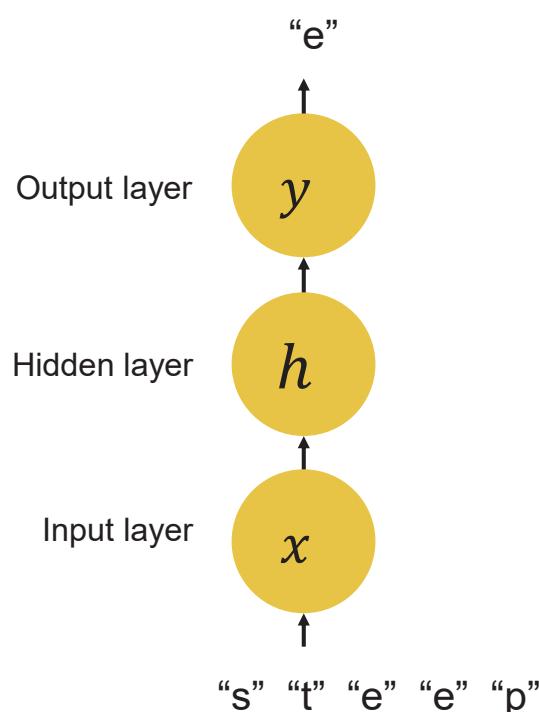
- ▶ Input is sort of fixed size and output is fixed size
- ▶ No information persistent from previous data during inference
- Application field is limited
- There is a need for more flexible processing of data sequences

► Example: Try to predict the next character in a word

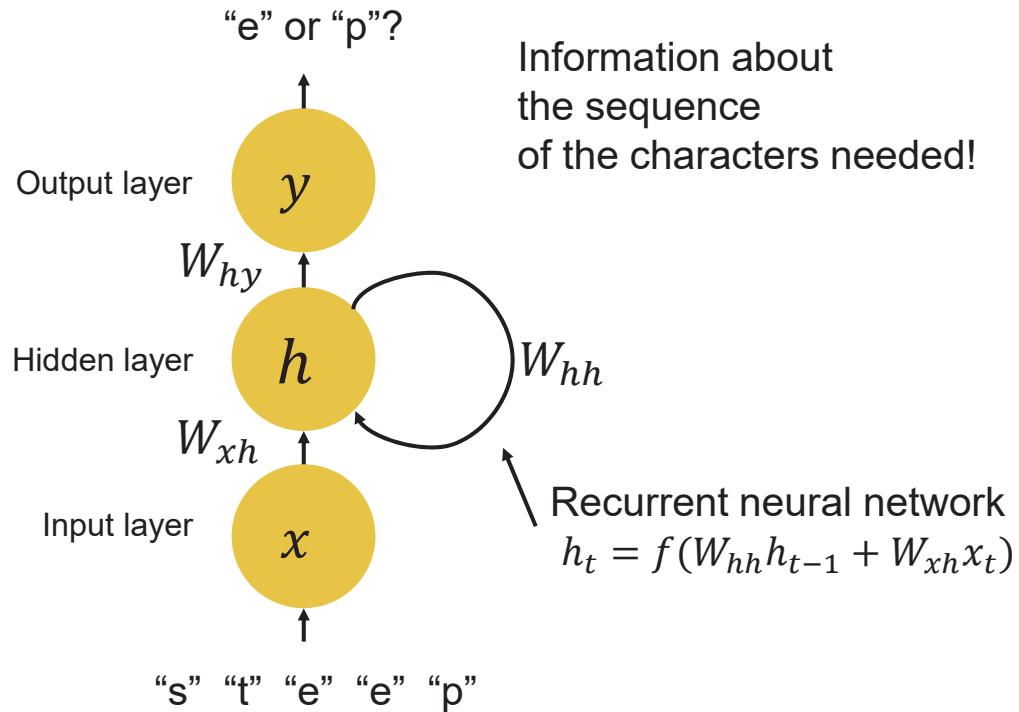


# Motivation

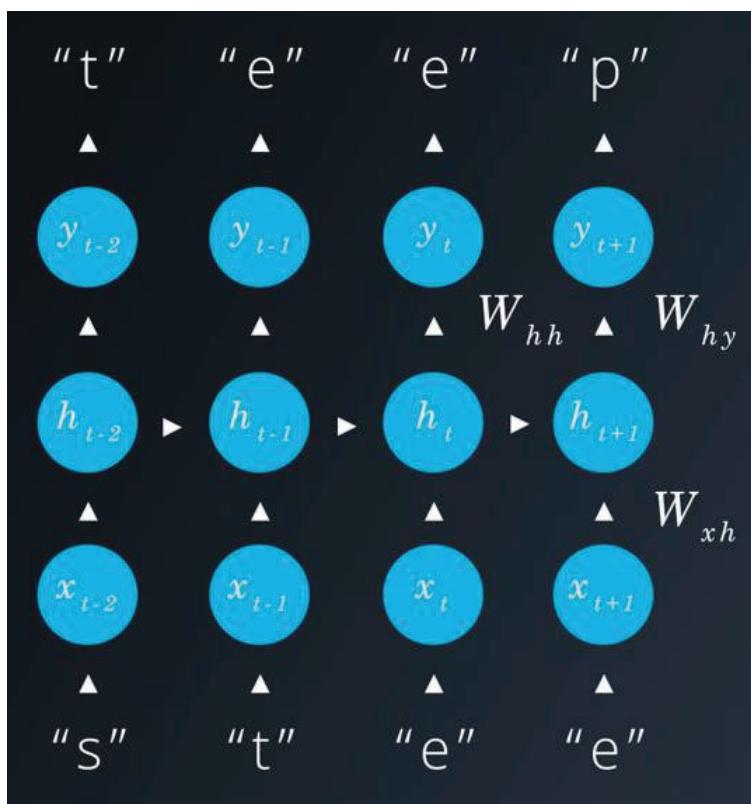
► Example: Try to predict the next character in a word



► Example: Try to predict the next character in a word



## Unrolled Recurrent Neural Network (RNN)



Note that the weight matrices in the unrolled network stay the same

# Unrolled, one-hot encoded values



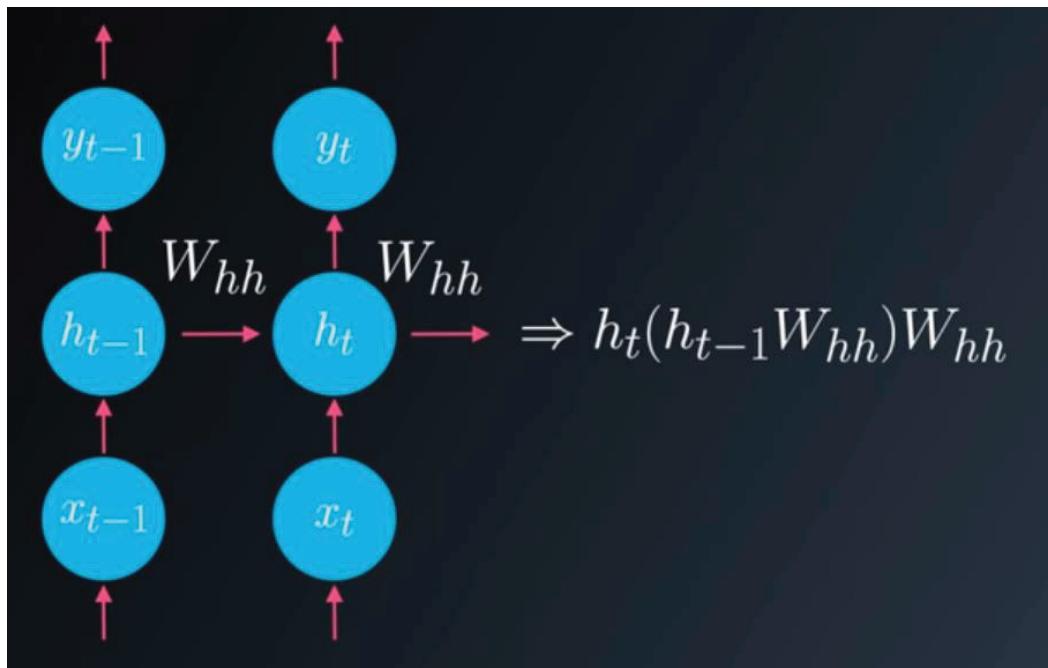
Input layer: 4-dimensional

Hidden layer: 3 units

Output layer: confidences → we want the blue number to be high and the red numbers to be low

<https://www.youtube.com/watch?v=64HSg6HafEI>, 06.06.2018

## Matrix multiplications



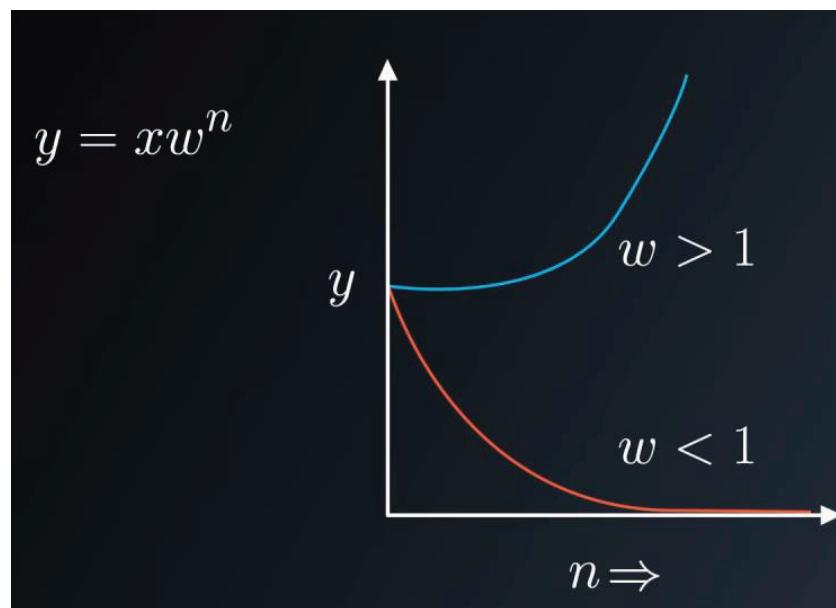
<https://www.youtube.com/watch?v=RYbSHdgZetc>, 07.06.2018

Observation:

Hidden state at step  $t$  is a function of the previous hidden state at step  $t-1$ , multiplied by  $W_{hh}$ . The output is again multiplied by  $W_{hh}$ .

→ many multiplications with  $W_{hh}$ .

# What happens to gradients in backprop?

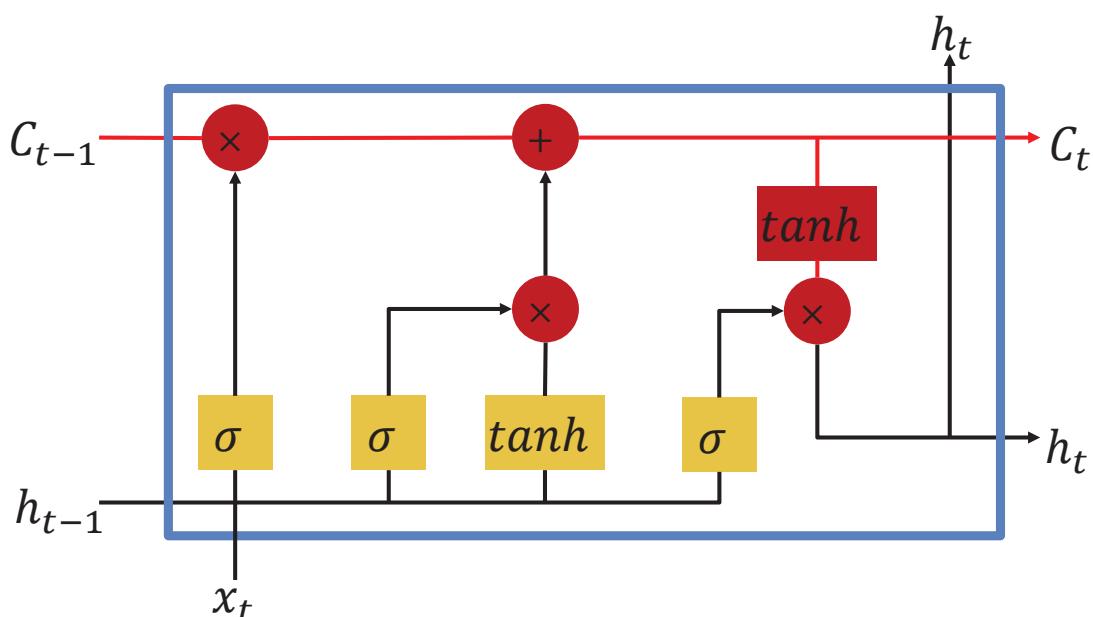


<https://www.youtube.com/watch?v=RYbSHqdzetc>, 07.06.2018

## ► Observation:

- Gradients either vanish or “explode”
- ➔ Makes it difficult to learn long range interactions (i.e.  $n$  is large)

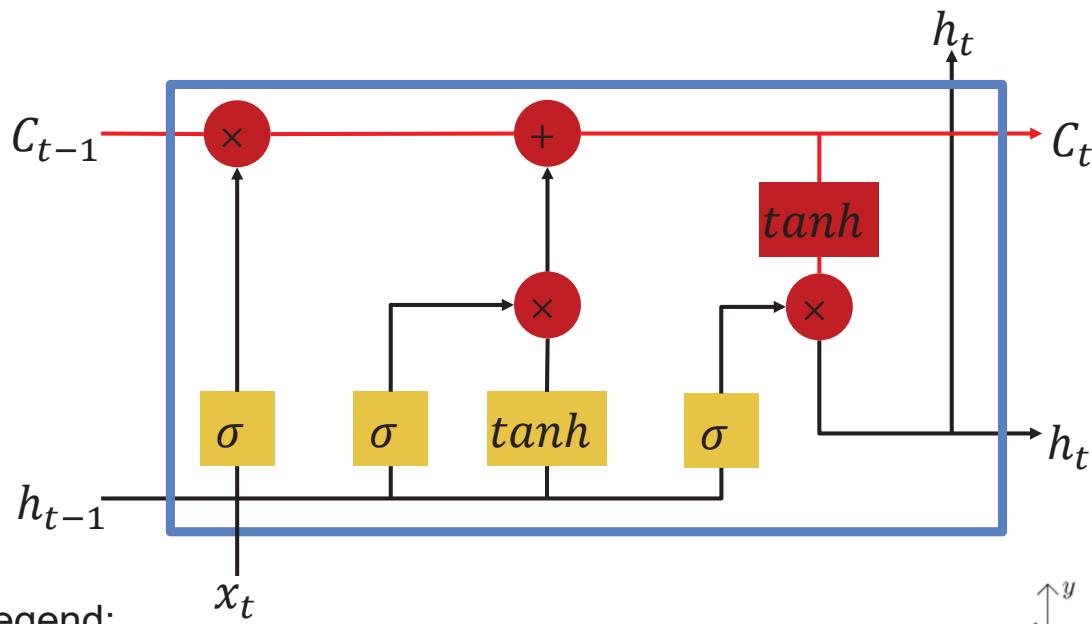
## Approach: LSTM Cell



<http://colah.github.io/posts/2015-08-Understanding-LSTMs/>, 07.06.2018

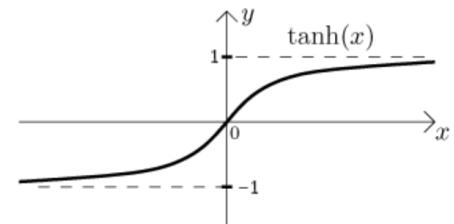
- Introduced by Hochreiter and Schmidhuber (1997)
- LSTM – “Long Short Term Memory”
- Today: One of the basic unit for recurrent networks
- Uses cell state  $C$  additionally to hidden state

# Approach: LSTM Cell

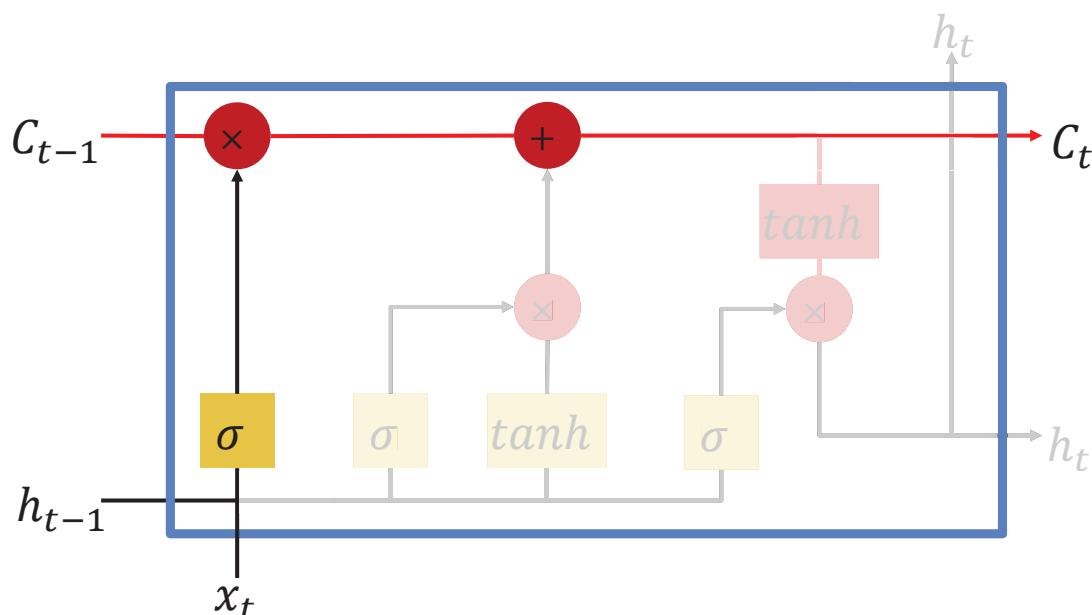


► Legend:

- Yellow boxes: trained network layers
- $\sigma$ : Sigmoid layer,  $tanh$ : hyperbolic tangent layer
- all layers have own weights
- Red boxes:  $+$ ,  $\times$ ,  $tanh$ : Element-wise operations
- $C$ : Cell state, lets information easily flow through the cell

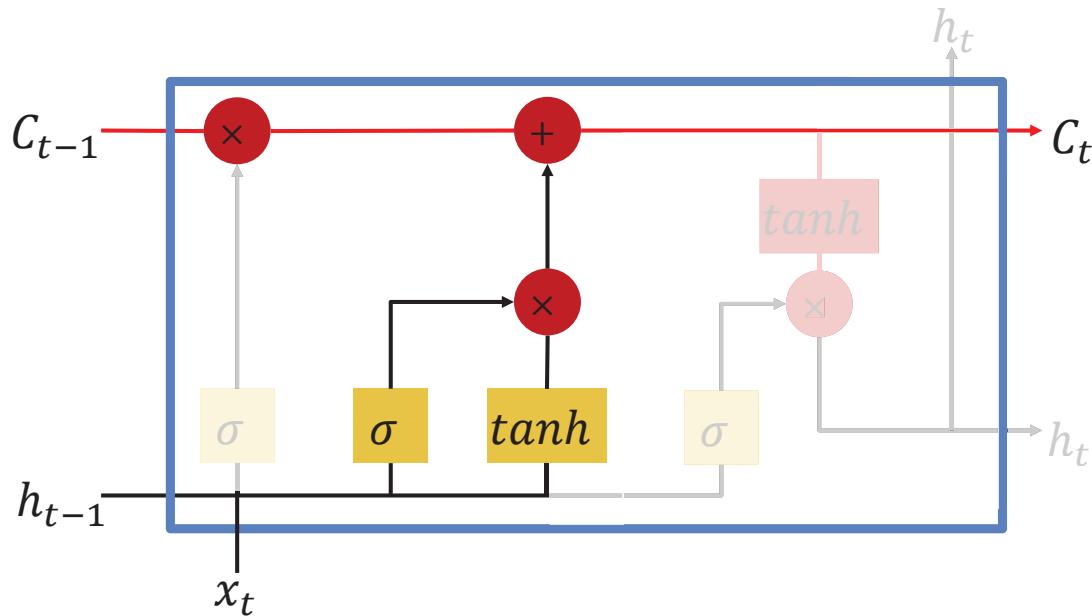


## LSTM Cell – Forget Gate



- Output of Sigmoid layer is between 0...1
- Output is multiplied elementwise with the cell state
- If sigmoid outputs are 1, the cell state is unaffected, otherwise it is changed or values are omitted
- Network can learn to forget information that causes incorrect predictions

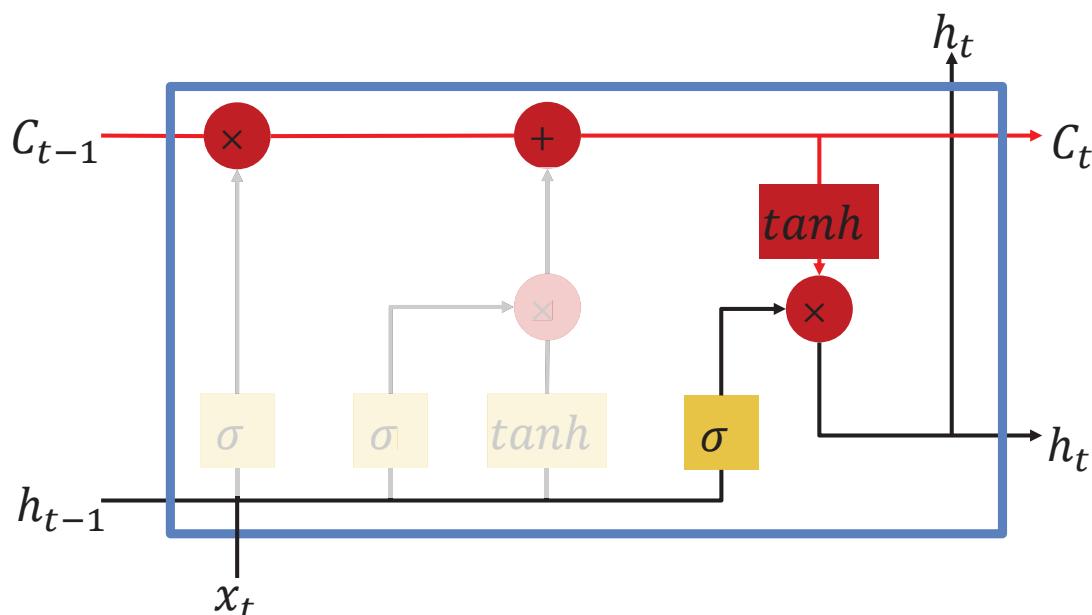
# LSTM Cell – Update Cell State



- ▶ Cell state is updated, using the previous hidden state  $h_{t-1}$  and the input  $x_t$
- ▶ The output of  $tanh$  is gated by a sigmoid layer

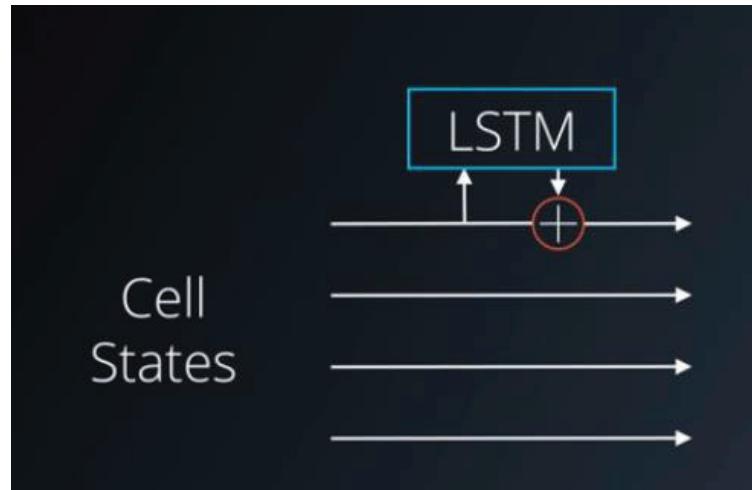
<http://colah.github.io/posts/2015-08-Understanding-LSTMs/>, 07.06.2018

# LSTM Cell – Cell State to Hidden Output



- ▶ Produce hidden state from cell state
  - ▶ Hidden state is sent to next hidden cell and to higher layers
  - ▶ Cell state is passed through  $tanh$  layer and gated by sigmoid layer
- Remember: all  $\sigma$ -Gates help network to learn which information to keep and which to omit.

<http://colah.github.io/posts/2015-08-Understanding-LSTMs/>, 07.06.2018

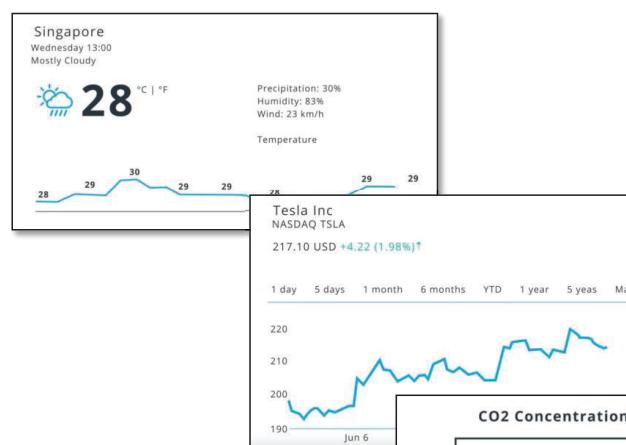


- ▶ Multiple variations of LSTM-Cells are available
- ▶ Only linear operations are affecting the cell state
- ▶ Improved gradient flow
  - no vanishing gradient problem any more
  - can learn long-term dependencies
- ▶ Remark: Gradients can still explode (Problem is addressed by thresholding)

## Time Series Forecasting

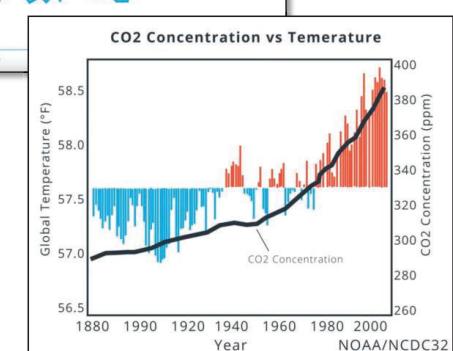
- ▶ Where do we find time series data in the real world?

- ▶ Weather Forecasts



- ▶ Stock Prices

- ▶ CO2 concentration vs temperature



## ► What is a time series exactly?

- It's an ordered sequence of values usually equally spaced over time like "every year", "every month", "every second", and so on.
- If you observe just one value than it is a univariate time series (e.g. Temperature)
- If you observe more than one it is a multivariate time series (e.g. Brainwaves, GPS coordinates or a Stereo Audio signal)

## ► Possible Application Examples:

- You could **forecast** stock prices to become rich, or the temperature, or how many products you should produce to meet the demand in the future.
- You could **understand the underlying process** that generated the time series e.g. study brainwaves to better understand sleep cycles.
- Time series analysis can also be used to **detect anomalies** e.g. the traffic of an e-mail server in order to find out abnormal activity such as a server attack.

## ► Common Patterns

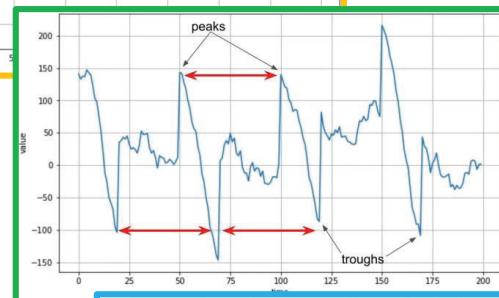
### ► Trend:

e.g. CO<sub>2</sub> concentration in the last decade



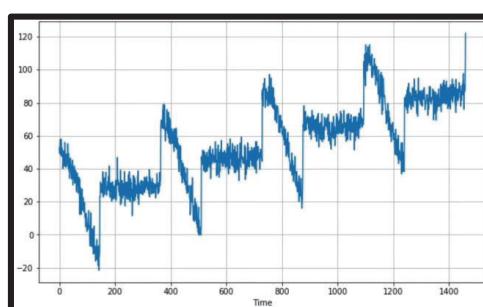
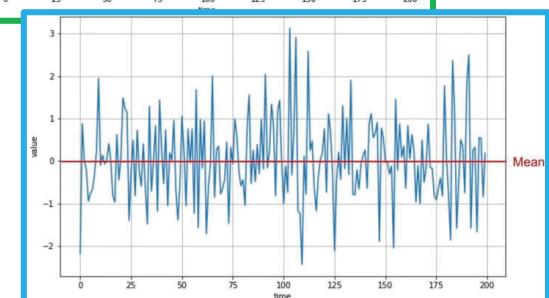
### ► Seasonality

e.g. Temperature drops every winter in comparison to the summer



### ► Noise

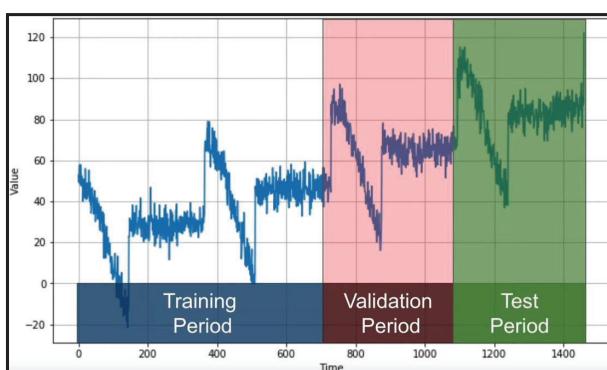
Almost all data coming from sensors do have noise



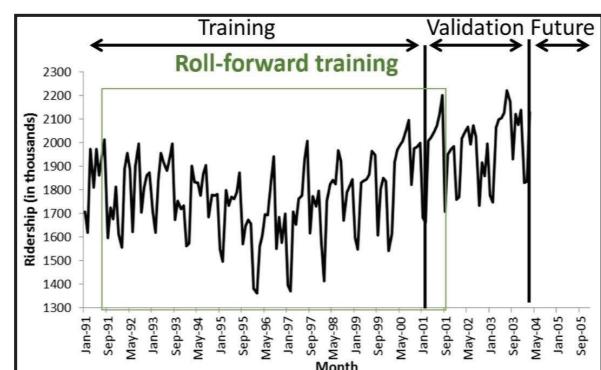
Trend + Seasonality + Noise

# Time Series Forecasting

## ► Partitioning



Fixed Partitioning



Roll-Forward Partitioning

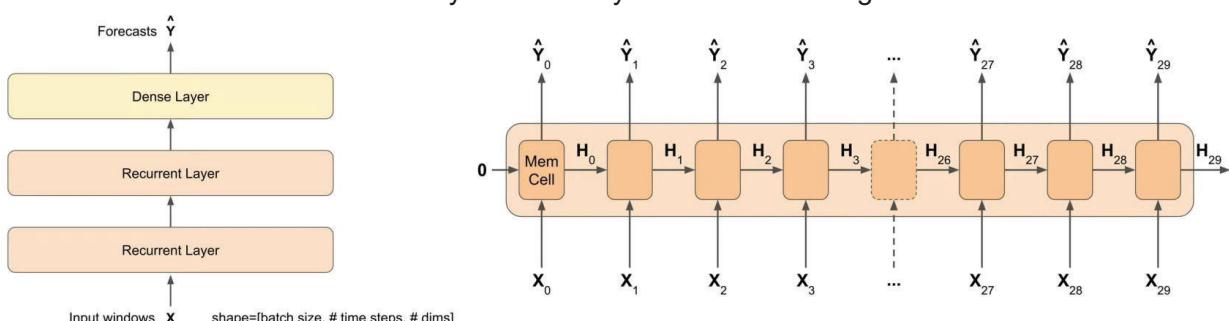
## ► Classical Forecasting Methods (not yet deep learning)

- Naive Forecasting (we will do in the notebook)
- Simple Average (SA)
- Moving Average (MA) (we will do in the notebook)
- Autoregression (AR)
- Autoregressive Integrated Moving Average (ARIMA)
- Seasonal Autoregressive Integrated Moving-Average (SARIMA)
- Vector Autoregression (VAR)
- Simple Exponential Smoothing (SES)

# Time Series Forecasting

## ► Deep Learning Technologies for Time Series Forecasting

- MLPs and CNNs
  - Possible to use for Time Series Forecasting (see in the jupyter notebook for more details)
- RNN – Recurrent neural network
  - It is typical for these kind of problems to use RNNs. A basic architecture can be seen on the left and a unrolled layer as already described in the right.

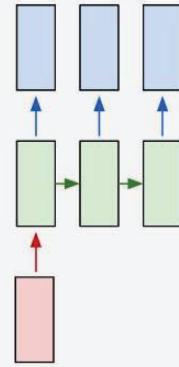


# Overview on types of network structures

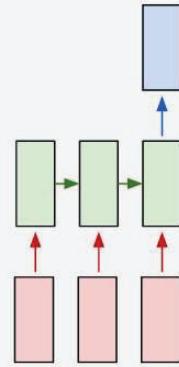
1)  
one to one



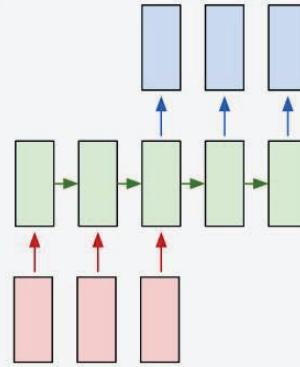
2)  
one to many



3)  
many to one



4)  
many to many



“Optimizing  
over functions”

“Optimizing over programs”

- 1) Feed forward NN
- 2) RNN with sequence output (e.g. image captioning)
- 3) RNN with sequence input (e.g. sentiment analysis)
- 4) RNN with sequence input and sequence output (e.g. machine translation)
- 5) RNN for synced sequence input and output (e.g. frame labeling in videos)

Auto.-Sys: Deep Learning

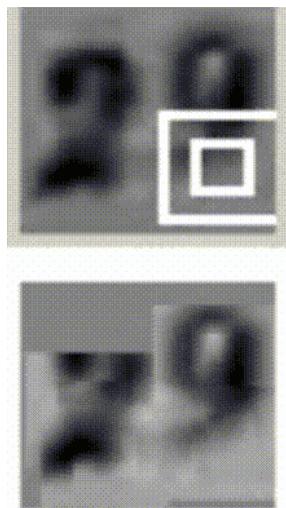
<http://karpathy.github.io/2015/05/21/rnn-effectiveness/>, 06.06.2018

Prof. Dr. N. Stache

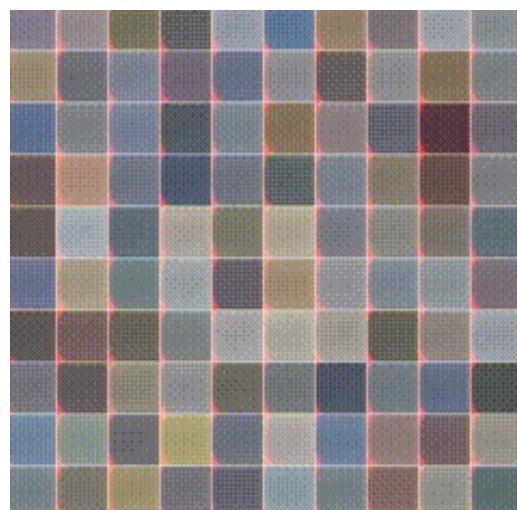
23

## Sequential processing in absence of sequences

- ▶ Sequential processing is also possible for fixed vectors:



RNN steers CNN kernel over image to read house numbers

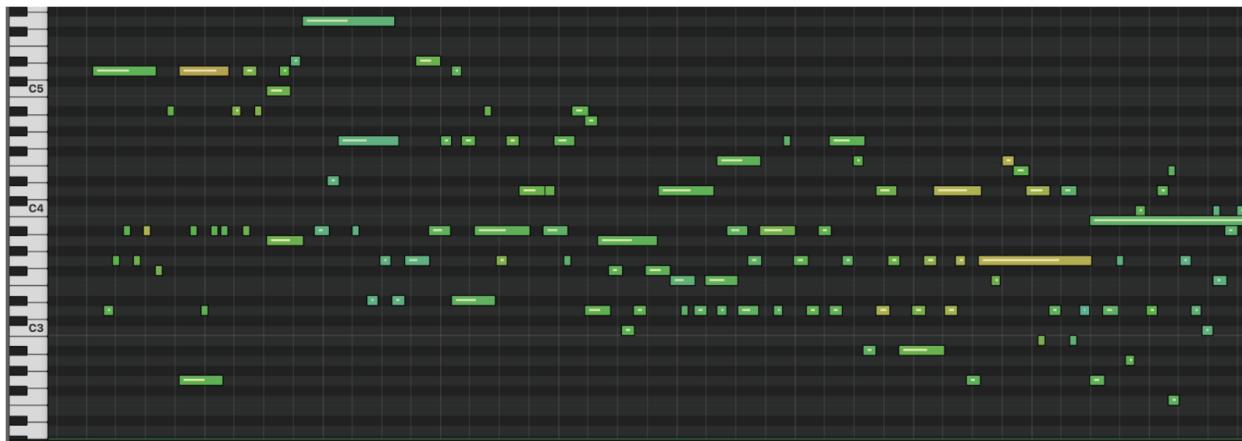


RNN paints house numbers sequentially

Multiple Object Recognition with Visual Attention, Ba et al.;  
DRAW: A Recurrent Neural Network For Image Generation, Gregor et al.

# Train network to generate music

- ▶ Use MIDI-Streams for training
- ▶ Network to generate MIDI-Stream



<https://magenta.tensorflow.org/performance-rnn>, 06.06.2018

## Image captioning

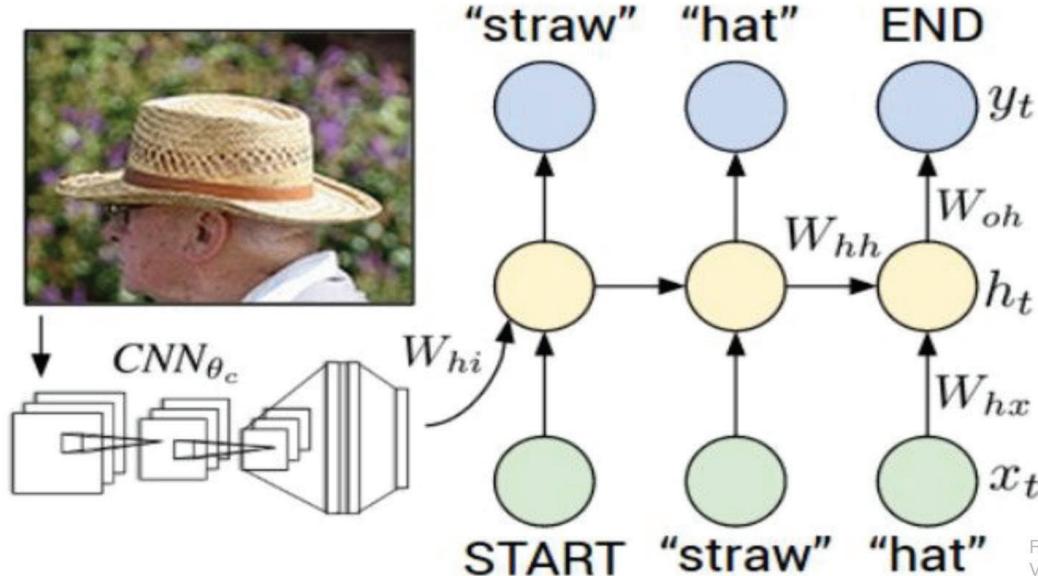


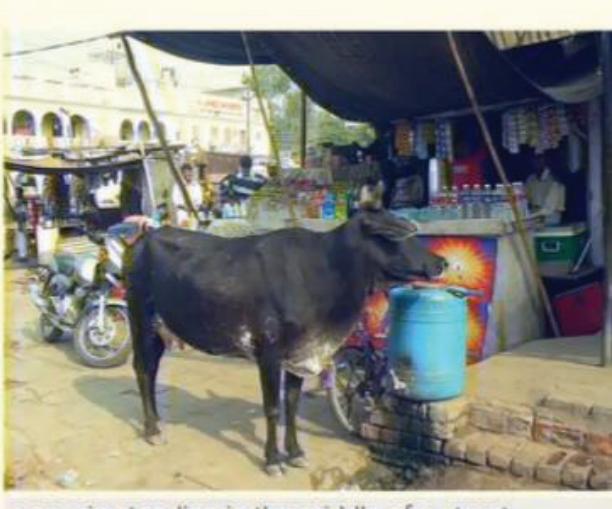
Figure from Karpathy et al., "Deep Visual-Semantic Alignments for Generating Image Descriptions", CVPR 2015; figure copyright IEEE, 2015.  
Reproduced for educational purposes.

# Image captioning

Three images for image captioning:

- 

a group of people standing around a room with remotes  
logprob: -9.17
- 

a young boy is holding a baseball bat  
logprob: -7.61
- 

a cow is standing in the middle of a street  
logprob: -8.84

<https://skillsmatter.com/skillscasts/6611-visualizing-and-understanding-recurrent-networks, 06.06.2018>

Auto.-Sys: Deep Learning

Prof. Dr. N. Stache

27

# Image captioning

<https://skillsmatter.com/skillscasts/6611-visualizing-and-understanding-recurrent-networks, 06.06.2018>

Three images for image captioning:

- 

a toilet with a seat up in a bathroom  
logprob: -13.44
- 

a woman holding a teddy bear in front of a mirror  
logprob: -9.65
- 

a horse is standing in the middle of a road  
logprob: -10.34

Auto.-Sys: Deep Learning

Prof. Dr. N. Stache

28

- ▶ RNN trained on the math book
- ▶ Here is the prediction:

*Proof.* Omitted.  $\square$

**Lemma 0.1.** Let  $\mathcal{C}$  be a set of the construction.

Let  $\mathcal{C}$  be a gerber covering. Let  $\mathcal{F}$  be a quasi-coherent sheaves of  $\mathcal{O}$ -modules. We have to show that

$$\mathcal{O}_{\mathcal{O}_X} = \mathcal{O}_X(\mathcal{L})$$

*Proof.* This is an algebraic space with the composition of sheaves  $\mathcal{F}$  on  $X_{\text{étale}}$  we have

$$\mathcal{O}_X(\mathcal{F}) = \{\text{morph}_1 \times_{\mathcal{O}_X} (\mathcal{G}, \mathcal{F})\}$$

where  $\mathcal{G}$  defines an isomorphism  $\mathcal{F} \rightarrow \mathcal{G}$  of  $\mathcal{O}$ -modules.  $\square$

**Lemma 0.2.** This is an integer  $\mathcal{Z}$  is injective.

*Proof.* See Spaces, Lemma ??.

**Lemma 0.3.** Let  $S$  be a scheme. Let  $X$  be a scheme and  $X$  is an affine open covering. Let  $\mathcal{U} \subset \mathcal{X}$  be a canonical and locally of finite type. Let  $X$  be a scheme. Let  $X$  be a scheme which is equal to the formal complex.

The following to the construction of the lemma follows.

Let  $X$  be a scheme. Let  $X$  be a scheme covering. Let

$$b : X \rightarrow Y' \rightarrow Y \rightarrow Y \rightarrow Y' \times_X Y \rightarrow X.$$

be a morphism of algebraic spaces over  $S$  and  $Y$ .

This since  $\mathcal{F} \in \mathcal{F}$  and  $x \in \mathcal{G}$  the diagram

$$\begin{array}{ccccc} S & \xrightarrow{\quad} & & & \\ \downarrow & & & & \\ \xi & \xrightarrow{\quad} & \mathcal{O}_{X'} & \xleftarrow{\quad} & \\ \text{gor}_s & & & & \\ & & & & \\ & & = \alpha' \longrightarrow & & \\ & & \downarrow & & \\ & & = \alpha' \longrightarrow \alpha & & \\ & & & & \\ \text{Spec}(K_\psi) & & \xrightarrow{\quad} & & \text{Mor}_{\text{Sets}} \xrightarrow{\quad} d(\mathcal{O}_{X/k}, \mathcal{G}) \\ & & & & \\ & & & & X \end{array}$$

is a limit. Then  $\mathcal{G}$  is a finite type and assume  $S$  is a flat and  $\mathcal{F}$  and  $\mathcal{G}$  is a finite type  $f_*$ . This is of finite type diagrams, and

- the composition of  $\mathcal{G}$  is a regular sequence,
- $\mathcal{O}_{X'}$  is a sheaf of rings.

*Proof.* We have see that  $X = \text{Spec}(R)$  and  $\mathcal{F}$  is a finite type representable by algebraic space. The property  $\mathcal{F}$  is a finite morphism of algebraic stacks. Then the cohomology of  $X$  is an open neighbourhood of  $U$ .  $\square$

*Proof.* This is clear that  $\mathcal{G}$  is a finite presentation, see Lemmas ??.  
A reduced above we conclude that  $U$  is an open covering of  $C$ . The functor  $\mathcal{F}$  is a

## Visualization cell activations

Cell sensitive to position in line:

```
The sole importance of the crossing of the Berezina lies in the fact
that it plainly and indubitably proved the fallacy of all the plans for
cutting off the enemy's retreat and the soundness of the only possible
line of action--the one Kutuzov and the general mass of the army
demanded--namely, simply to follow the enemy up. The French crowd fled
at a continually increasing speed and all its energy was directed to
reaching its goal. It fled like a wounded animal and it was impossible
to block its path. This was shown not so much by the arrangements it
made for crossing as by what took place at the bridges. When the bridges
broke down, unarmed soldiers, people from Moscow and women with children
who were with the French transport, all--carried on by vis inertiae--.
pressed forward into boats and into the ice-covered water and did not,
surrender.
```

Cell that turns on inside quotes:

```
"You mean to imply that I have nothing to eat out of.... on the
contrary, I can supply you with everything even if you want to give
dinner parties," warmly replied Chichagov, who tried by every word he
spoke to prove his own rectitude and therefore imagined Kutuzov to be
animated by the same desire.
```

```
Kutuzov, shrugging his shoulders, replied with his subtle penetrating
smile: "I meant merely to say what I said."
```

Cell that robustly activates inside if statements:

```
static int __dequeue_signal(struct sigpending *pending, sigset_t *mask,
    siginfo_t *info)
{
    int sig = next_signal(pending, mask);
    if (sig) {
        if (current->notifier) {
            if ((sigismember(current->notifier_mask, sig)) {
                if (!((current->notifier)(current->notifier_data))) {
                    clear_thread_flag(TIF_SIGPENDING);
                    return 0;
                }
            }
            collect_signal(sig, pending, info);
        }
        return sig;
    }
}
```

A large portion of cells are not easily interpretable. Here is a typical example:

```
/* Unpack a filter field's string representation from user-space
 * buffer. */
char *audit_unpack_string(void **bufp, size_t *remain, size_t len)
{
    char *str;
    if (!*bufp || (len == 0) || (len > *remain))
        return ERR_PTR(-EINVAL);
    /* of the currently implemented string fields, PATH_MAX
     * defines the longest valid length.
    */
```

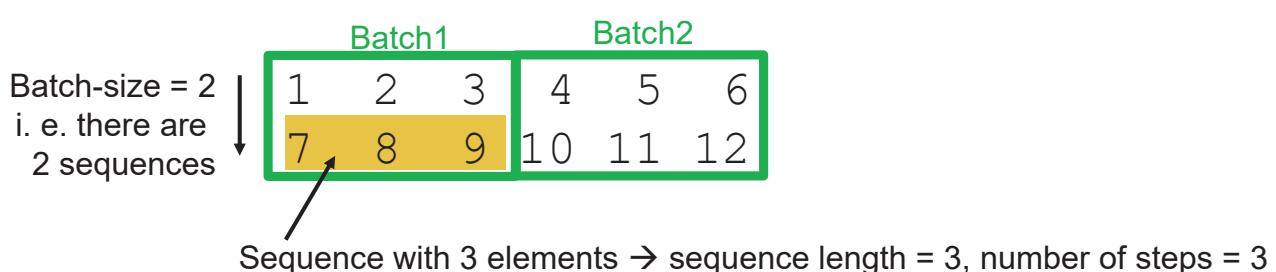
# Hands-on Example

- ▶ Create a RNN using LSTMs in Tensorflow
- ▶ Train it on Anna Karenina (Tolstoi)
- ▶ Sample trained network  
(means let it predict text character by character, given a starting sample)

## Explanation of batch wording

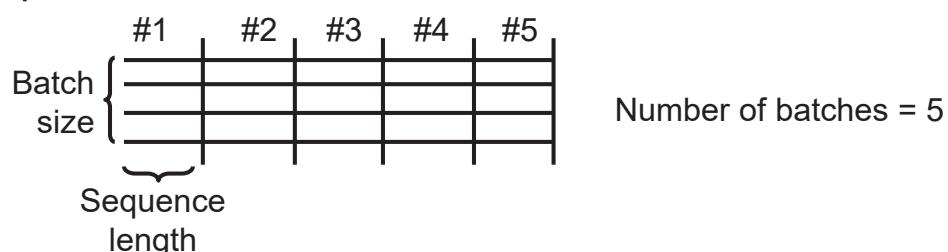
Input:

1    2    3    4    5    6    7    8    9    10    11    12

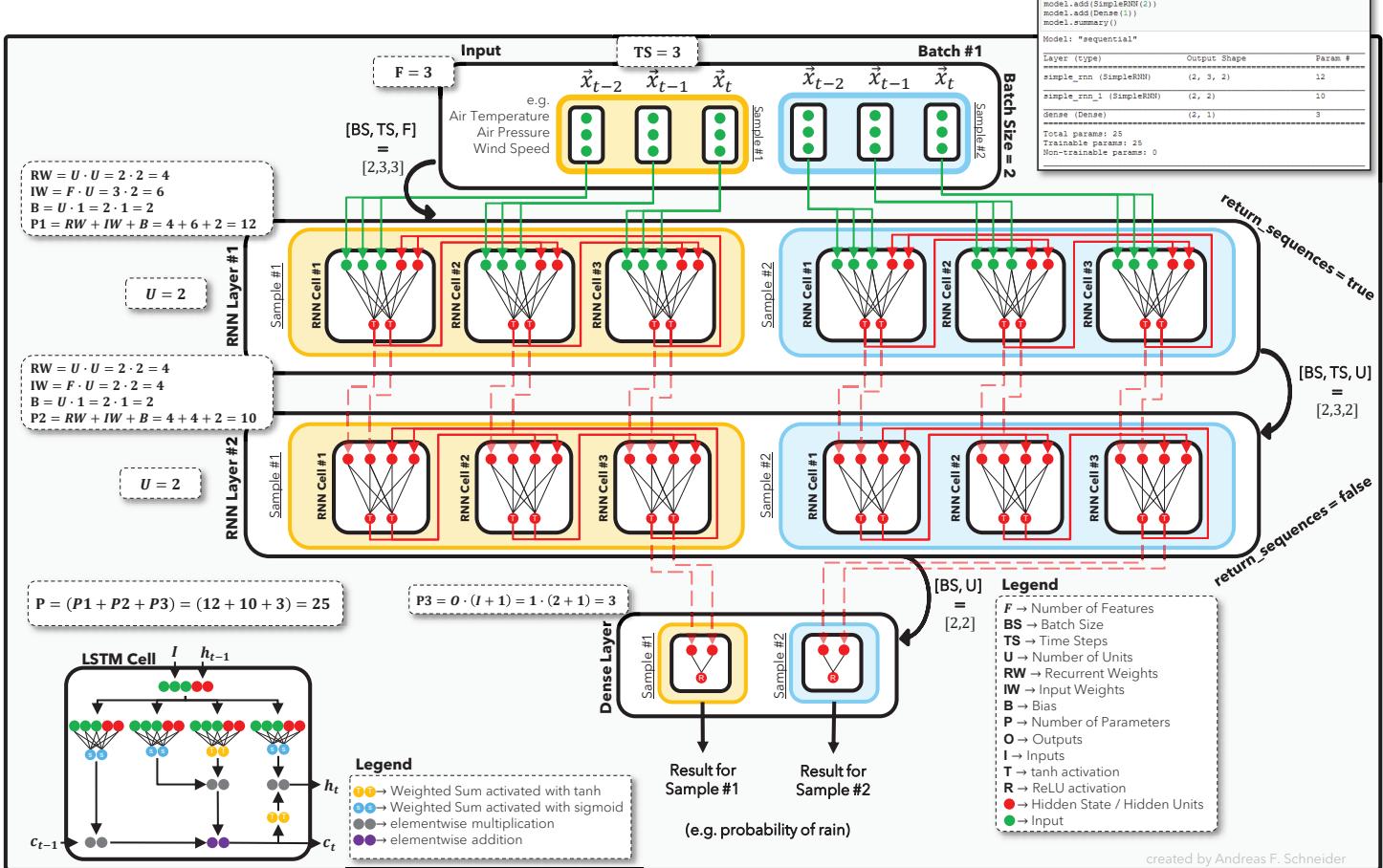


---

Other example:



# Simple RNN Full Example





## Autonomous Systems: Deep Learning

### 9. Embeddings, Word2Vec



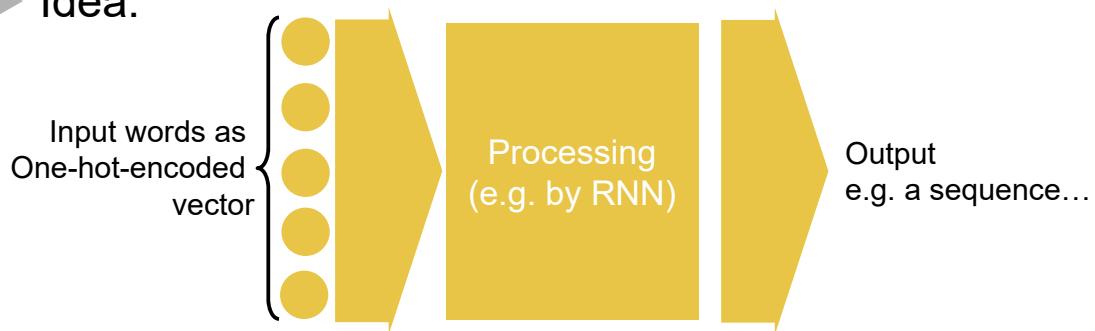
Prof. Dr.-Ing. Nicolaj Stache  
Heilbronn University of Applied Sciences

#### Task



- ▶ Create a network which “understands” text in order to
  - ▶ ... create a chatbot
  - ▶ ... analyze the sentiment
  - ▶ ... translate text
  - ▶ ... create a summary

- ▶ Idea:



# Problems

- ▶ Large amount of words:  
Input vector might have e.g. 74,000 entries
  - ▶ Different words with similar meaning:  
e.g. “cat” and “kitty”
  - ▶ Rare words convey important information  
e.g. if document contains “retinopathy” → medical document
- 

- ➔ Lots of training data + labels needed for supervised learning

# Approach

- ▶ Use **unsupervised learning** to find a mathematical description of the meaning of the words
- ▶ Unsupervised learning:
  - ▶ It is easy to find large amounts of data without labels!  
→ E.g. use Wikipedia articles etc.
- ▶ Idea: Words in similar context have similar meaning
  - ▶ Find out in which context which word appears  
→ use surrounding words

## Example

- ▶ “The **cat** purrs”
  - ▶ “The **cat** hunts mice”
- ↑↓  
Similar context  
indicates similar meaning  
of cat and kitty
- ▶ “The **kitty** purrs”
  - ▶ “The **kitty** hunts mice”
- 
- ▶ “The **dog** purrs”
  - ▶ “The **dog** hunts mice”



<https://pixabay.com/de/tier-katze-häuslich-auge-augen-17545/>, 13.06.2018



<https://pixabay.com/de/hund-husky-schlittenhund-tier-2016708/>, 13.06.2018

Auto.-Sys: Deep Learning

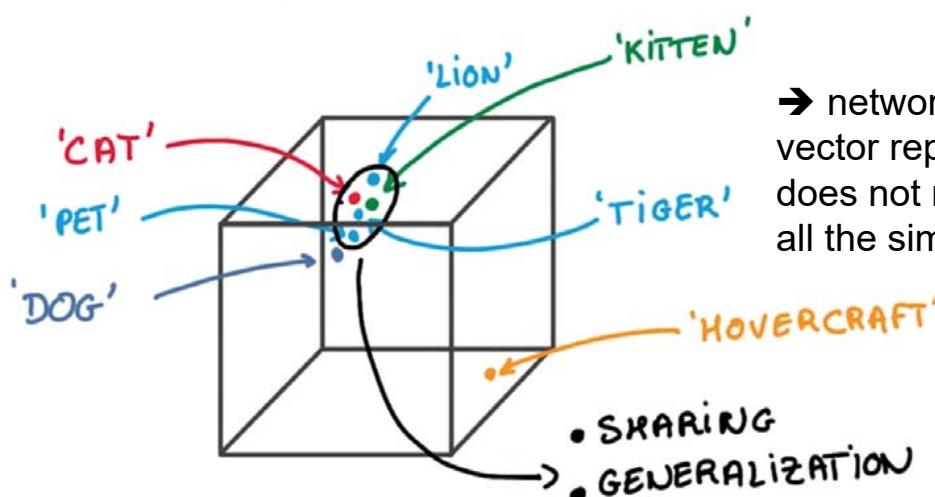
Prof. Dr. N. Stache

5

## “Mathematical description”

- ▶ Words are mapped to vectors of defined length
- ▶ Vectors model the properties of a word
- ▶ Here:
  - ▶ Words describing cat-like behavior are assigned to similar vectors
  - ▶ The distance of vectors can describe the relationship of the words

→ network using  
vector representation  
does not need to learn  
all the similar words



Auto.-Sys: Deep Learning

Prof. Dr. N. Stache

6

<https://www.youtube.com/watch?v=186HTTBonpY>, 13.06.2018

- ▶ Idea: Words in similar context have similar meaning
  - ▶ Find out in which context which word appears  
→ use surrounding words
  - ▶ Train a simple network to predict context words, given an input word  
→ network learns the statistics of word pairings (training data is word pairings)  
→ Hence, it learns similar pairings for similar input words  
→ The hidden layer's weights are trained in a way that the output produces similar pairings for similar input words  
→ The hidden layer's weights somehow encode the word's meaning to perform the above job
  - ▶ Take the weights of the hidden layer which are activated for a certain word as “word vector” which encodes its meaning.

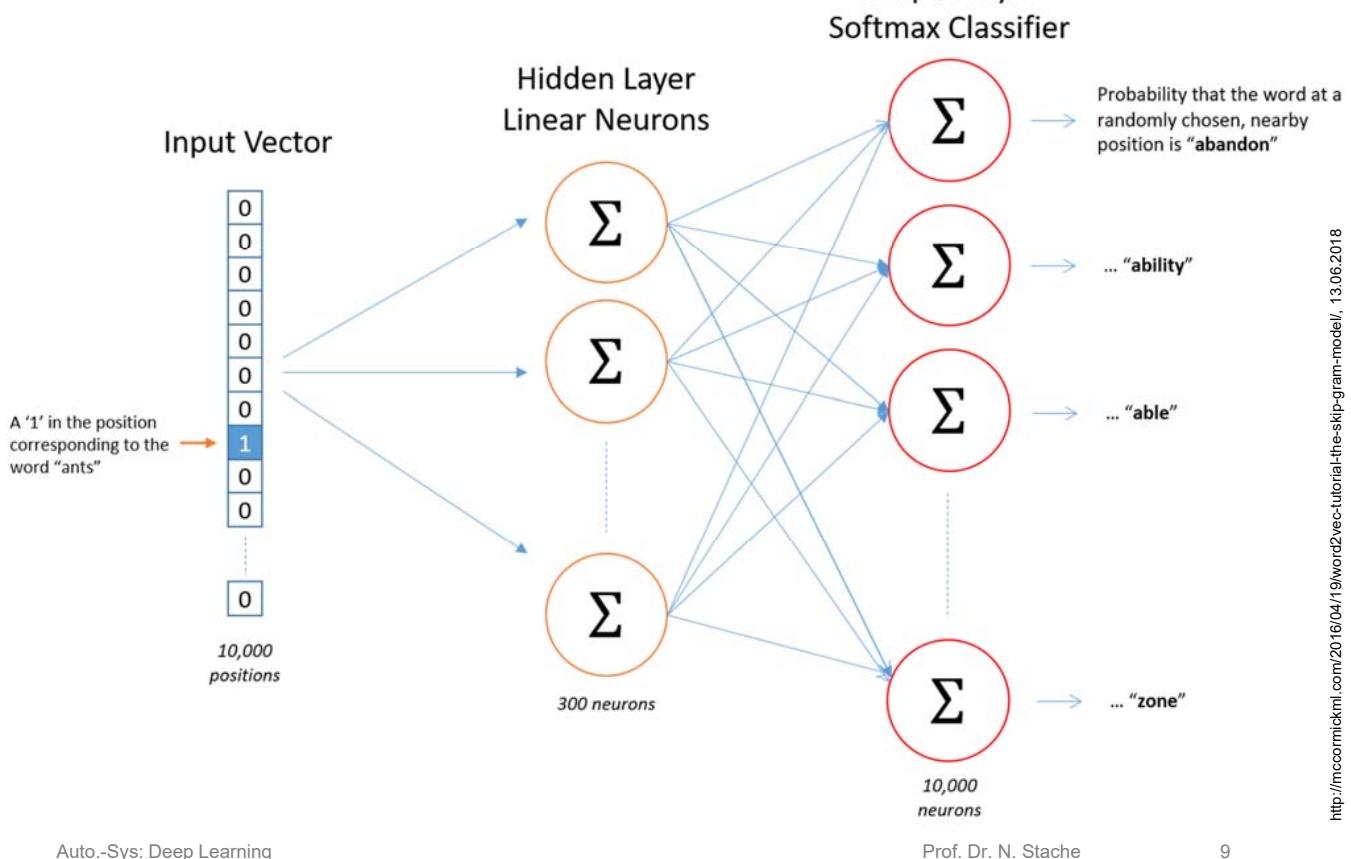
<http://mccormickml.com/2016/04/19/word2vec-tutorial-the-skip-gram-model/>, 13.06.2018

## More detailed description

Network to learn embeddings  
(assume our vocab has 10000 words):

- ▶ **Input layer takes one-hot-encoded input words**  
i.e. if there are 10000 words, the input vector has 10000 components with a 1 at the corresponding word position
- ▶ **Output is a vector of 10000 components** containing the probability for each word to appear in a pairing with the input word, output activationfunction is softmax

# More detailed description



# More detailed description

## Source Text

The quick brown fox jumps over the lazy dog.  $\rightarrow$

(the, quick)  
(the, brown)

The quick brown fox jumps over the lazy dog.  $\rightarrow$

(quick, the)  
(quick, brown)  
(quick, fox)

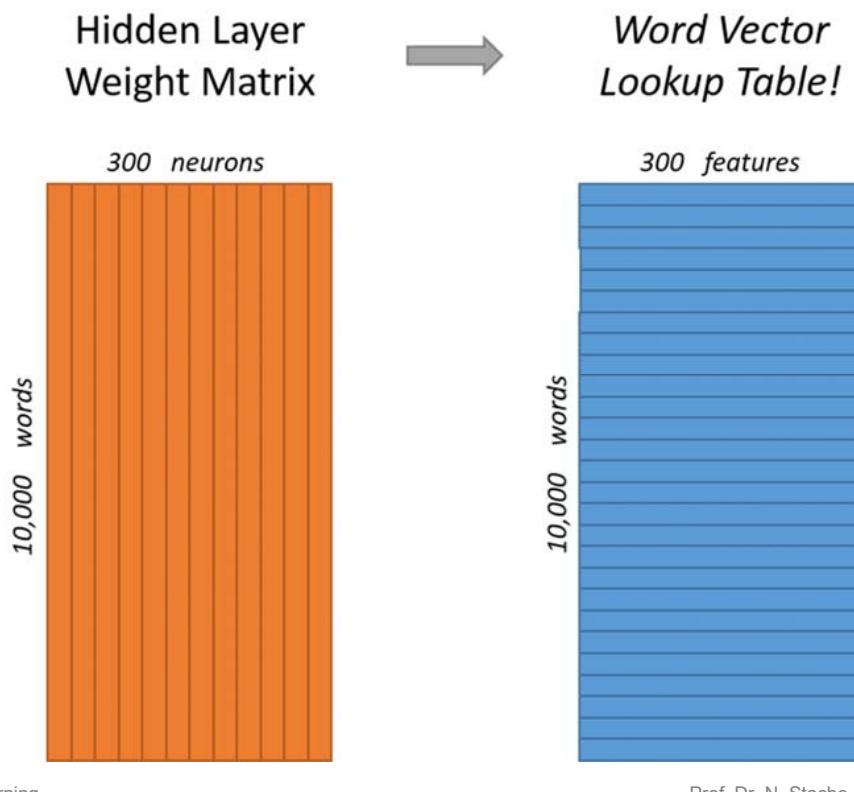
The quick brown fox jumps over the lazy dog.  $\rightarrow$

(brown, the)  
(brown, quick)  
(brown, fox)  
(brown, jumps)

The quick brown fox jumps over the lazy dog.  $\rightarrow$

(fox, quick)  
(fox, brown)  
(fox, jumps)  
(fox, over)

Let's assume the hidden layer has 300 neurons...



## End result

- ▶ The desired result is the weight matrix of the hidden layer, as it encodes the meaning of the word
- ▶ Relation between weight matrix and word...

$$\begin{bmatrix} 0 & 0 & 0 & \textcolor{green}{1} & 0 \end{bmatrix} \times \underbrace{\begin{bmatrix} 17 & 24 & 1 \\ 23 & 5 & 7 \\ 4 & 6 & 13 \\ \textcolor{green}{10} & \textcolor{green}{12} & \textcolor{green}{19} \\ 11 & 18 & 25 \end{bmatrix}}_{\text{Trained weight matrix. Note: only one row is selected for each word}} = \begin{bmatrix} 10 & 12 & 19 \end{bmatrix}$$

One-hot-input vector of specific word      Trained weight matrix. Note: only one row is selected for each word      This row is used as an embedding vector

- ▶ Preprocess data to detect common phrases and treat them as single words,
  - ▶ e.g. "New York" or "Boston Globe" (= Newspaper)
  - ▶ but omit combinations of frequent words like "this is" or "and the"
- ▶ Subsampling frequent words
  - ▶ Words like "the" in the context of a given word does not contribute to contextual information  
→ randomly remove word using an adaptive keep probability
- ▶ Negative sampling
  - ▶ During training process, only update small fractions of the weights belonging to negative outputs
  - ▶ Saves computational power at almost no performance drop

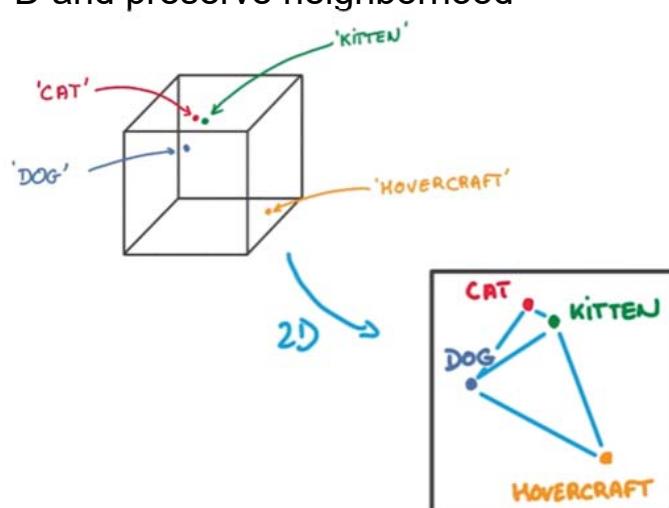
## Visualize by t-SNE

- ▶ Problem:  
Meaning of words is encoded in vectors with 300 entries → 300-dimensional space(!)
- ▶ Approach:  
Map 300-dimensional space to 2-D and preserve neighborhood structure of data

- ▶ T-SNE: T-Distributed Stochastic Neighbor Embedding (available in libraries)

- ▶ Further reading:

Laurens van der Maaten and Geoffrey Hinton.  
Visualizing Data using t-SNE.  
Journal of Machine Learning Research,  
2008. Vol. 9, pp. 2579-2605.



# Application example of visualizing MNIST data with t-SNE

- ▶ Basic principle:  
Try to capture the distribution in high dimensional space by computing normalized distances between all points and storing distribution in a matrix
- ▶ Place the points in low dimensional space in a way that the distribution matrix from high dimensional space is approximated.

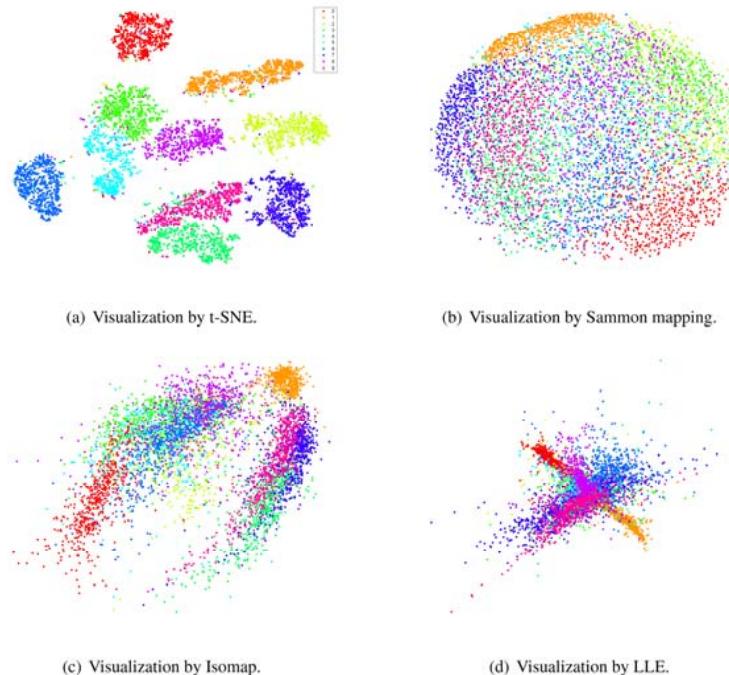


Figure 2: Visualizations of 6,000 handwritten digits from the MNIST dataset.

## Applications

- ▶ For each word we now have a vector, corresponding to its meaning
- ▶ Applications:
  - ▶ Find similar words or related words
  - ▶ Use to train a neural network on the vector, representing the meaning instead of the word itself
  - ▶ Even do “math” with it:
    - ▶ man – woman + queen =
    - ▶ Puppy – dog + cat =
    - ▶ Taller – tall + short =

Word	Correlation
blue	0.86534697
yellow	0.7989963
green	0.7967982
pink	0.7936595
purple	0.7788067

- ▶ Semantic analogy
  - ▶ Puppy vs. Dog  $\Leftrightarrow$  Kitten vs. Cat
- ▶ Syntactic analogy
  - ▶ Taller vs. Tall  $\Leftrightarrow$  Shorter vs. Short

$$V' = V_{\text{PUPPY}} - V_{\text{DOG}} + V_{\text{CAT}}$$

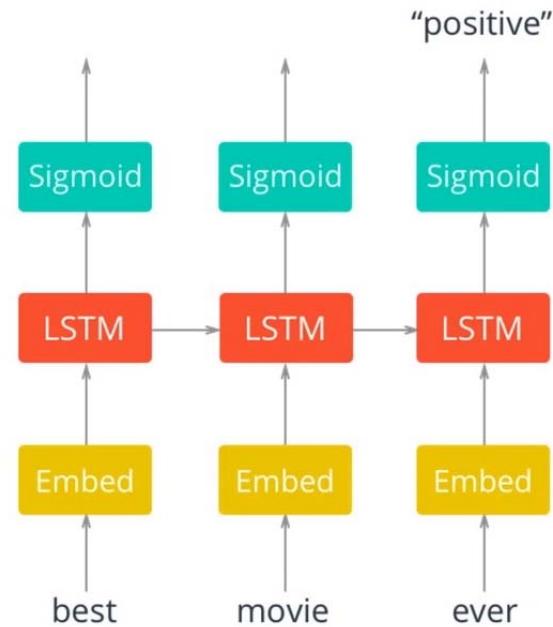
The diagram shows a square box labeled "EMBEDDING SPACE". Inside the box, there are three vectors: a green circle labeled "KITTEN", a blue dot labeled "DOG", and an orange circle labeled "PUPPY". A blue arrow points from the text "EMBEDDING SPACE" to the top right corner of the box.

## Hands-on part Word2Vec

- ▶ Please work through the tutorial embeddings.zip in the tensorflow folder
- ▶ You are going to determine word embeddings and visualize the results with t-SNE

# Use embeddings for language processing

- ▶ Please work through the tutorial sentiment-rnn
- ▶ The example is on sentiment analysis (as the bag of words example earlier in this lesson).
- ▶ Uses RNNs to capture context
- ▶ Uses embeddings
- ▶ Please note: this notebook makes a shortcut with the embedding layer, since it just trains it together with the RNN





## Autonomous Systems: Deep Learning

### 9. Autoencoders, GANs, Visualization



Prof. Dr.-Ing. Nicolaj Stache  
Heilbronn University of Applied Sciences

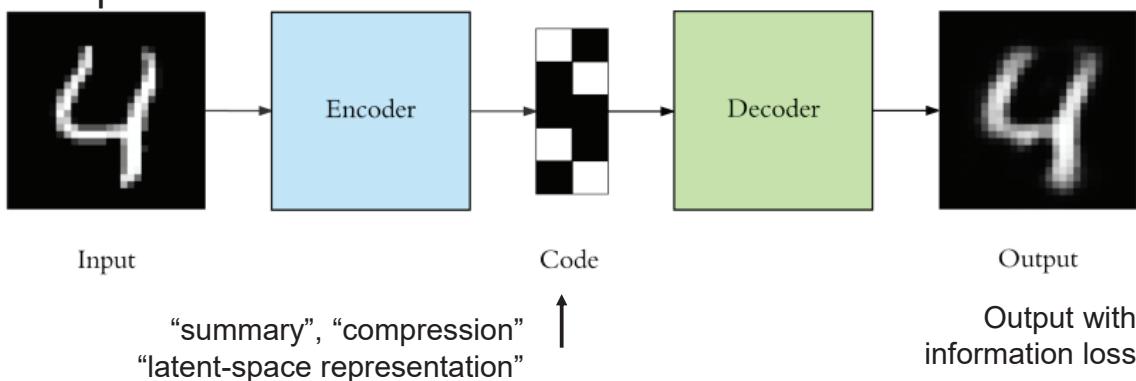
#### Outline



- ▶ Autoencoders
- ▶ Generative Adversarial Networks
- ▶ Visualization + Fooling Networks

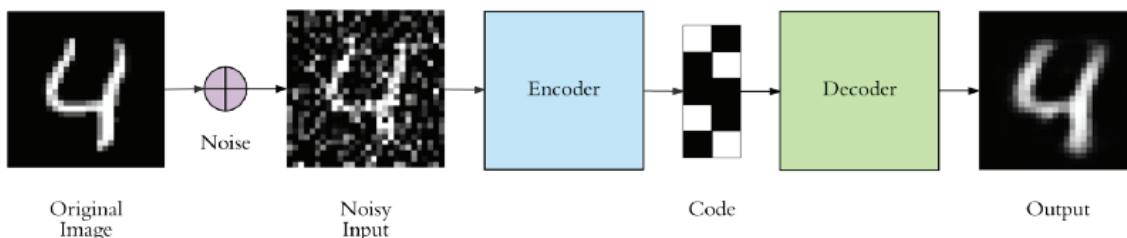
# Autoencoders

- ▶ How can we deal with unlabeled data?
- ▶ Idea:
  - ▶ Learn the underlying structure in the data with simple neural network
  - ▶ Network tries to reproduce the input → no labels required
  - ▶ Hidden layer(s) of network produce compressed form of data.
- ▶ Principle of an autoencoder:



# Applications

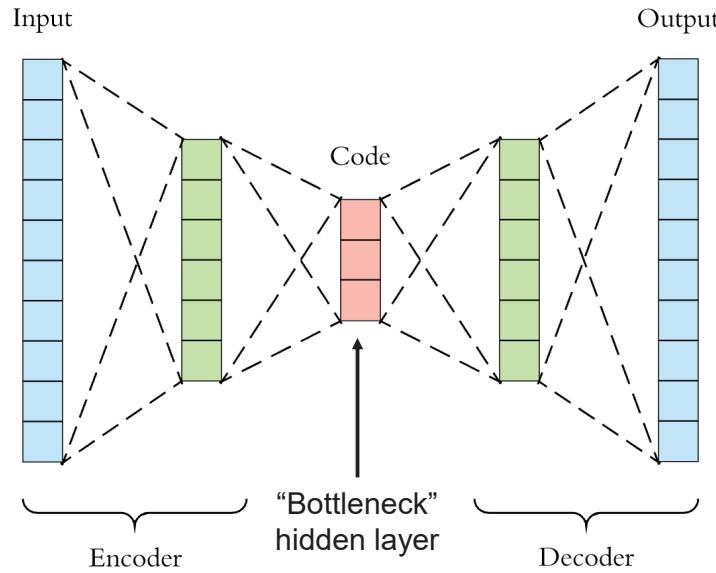
- ▶ Find low-dimensional representation of data
- ▶ Data compression
  - ▶ However, not as efficient as specific algorithms (like jpg), and less generalizing since they are adapted to trained data
- ▶ Image denoising / image reconstruction
  - ▶ Code can only model the basic image content which is reconstructed without high frequent noise or flaws



- ▶ Image colorization

# Autoencoder-Architecture

- ▶ Encoder and decoder can be composed of multiple hidden layers
- ▶ Codes are the activations of the smallest hidden layer (not the weights as in embeddings)



## Hands on work

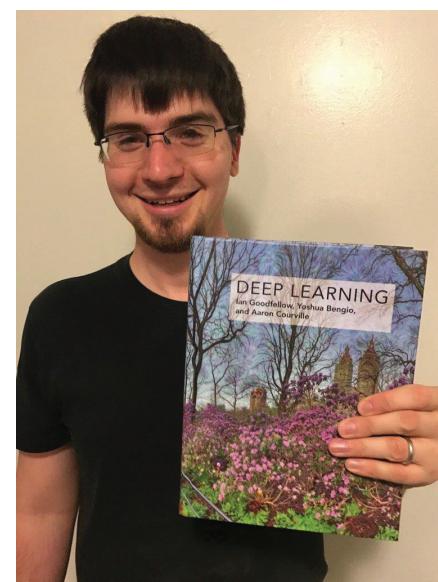
- ▶ Simple autoencoder (feed forward neural network)
- ▶ Convolutional autoencoder

# Outline

- ▶ Autoencoders
- ▶ Generative Adversarial Networks
- ▶ Visualization + Fooling Networks

## Generative Adversarial Networks (GANs)

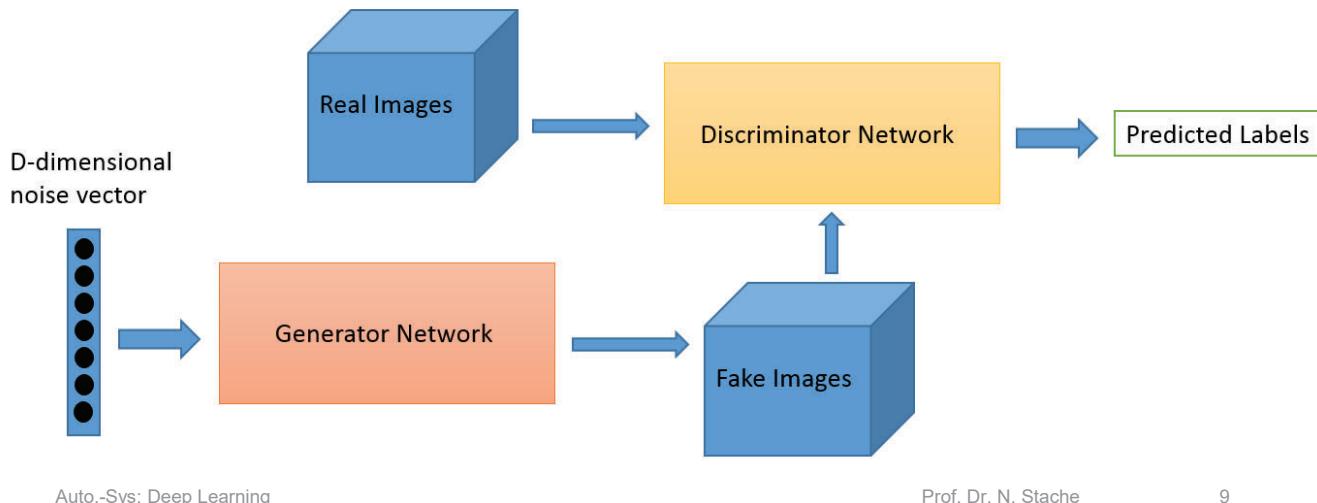
- ▶ Invented by Ian Goodfellow in 2014
- ▶ Key idea:
  - ▶ GANs are two deep neural networks competing against each other
- ▶ Discriminative algorithms:
  - ▶ classification, given input data
  - ▶ Learn boundaries of class
- ▶ Generative algorithms:
  - ▶ create input data, given a class
  - ▶ Model distribution of individual class



<https://twitter.com/goodfellow> ian/status/806103072158060546k, 21.06.2018

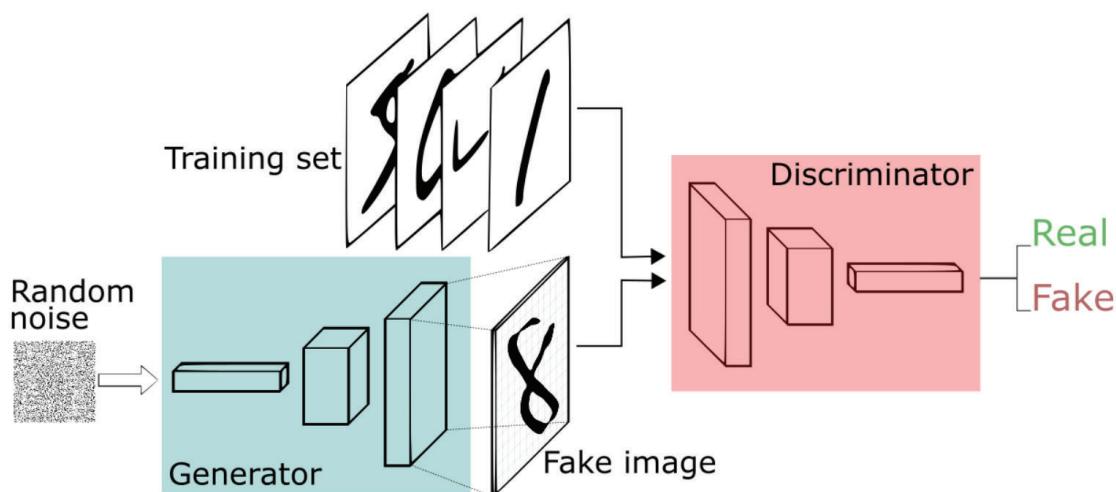
# How GANs work

- ▶ Network 1: Generator, generates new data, e.g. a painting of Picasso
- ▶ Network 2: Discriminator, decides which data is real or fake, e.g. is it a painting of Picasso or not



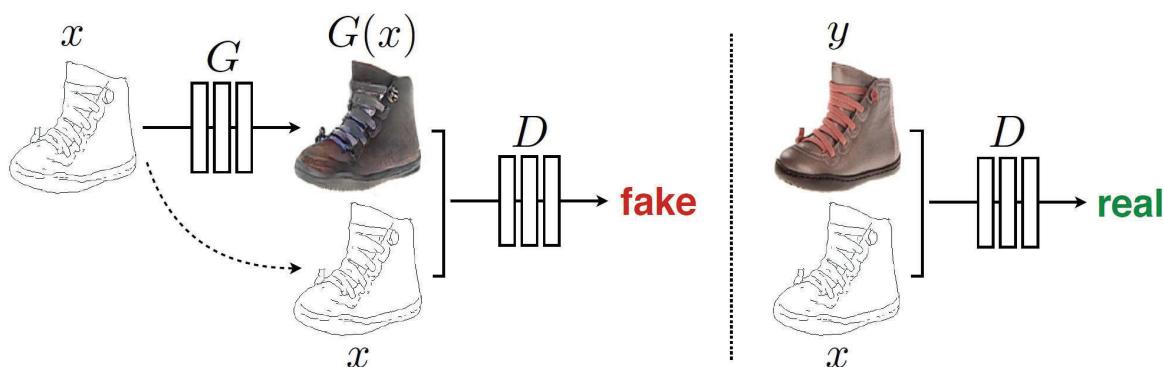
# How GANs work

- ▶ Networks optimize an opposing objective function
- ▶ Both are linked:
  - ▶ when generator changes, the discriminator has to react and vice versa



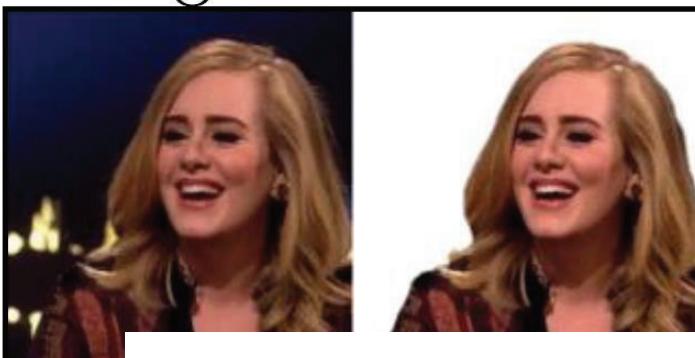
# Applications

- ▶ Generate new photorealistic images (which have never been seen before)  
(GAN is drawing this from the learned probability distribution over all hypothetical images matching to the discriminator)
- ▶ Image-to-Image Translation  
(see: <https://arxiv.org/abs/1611.07004>)



# Applications

## Background removal



# Applications



<https://www.youtube.com/watch?v=bo-T0tchgew>, 21.06.2018

- ▶ Generated images can also be used for further algo development, e.g. simulation of human gaze without the need of test persons...
- ▶ GANs are not restricted to images...
  - ▶ Use for predicting simulation results
  - ▶ Initializing start values for optimizers

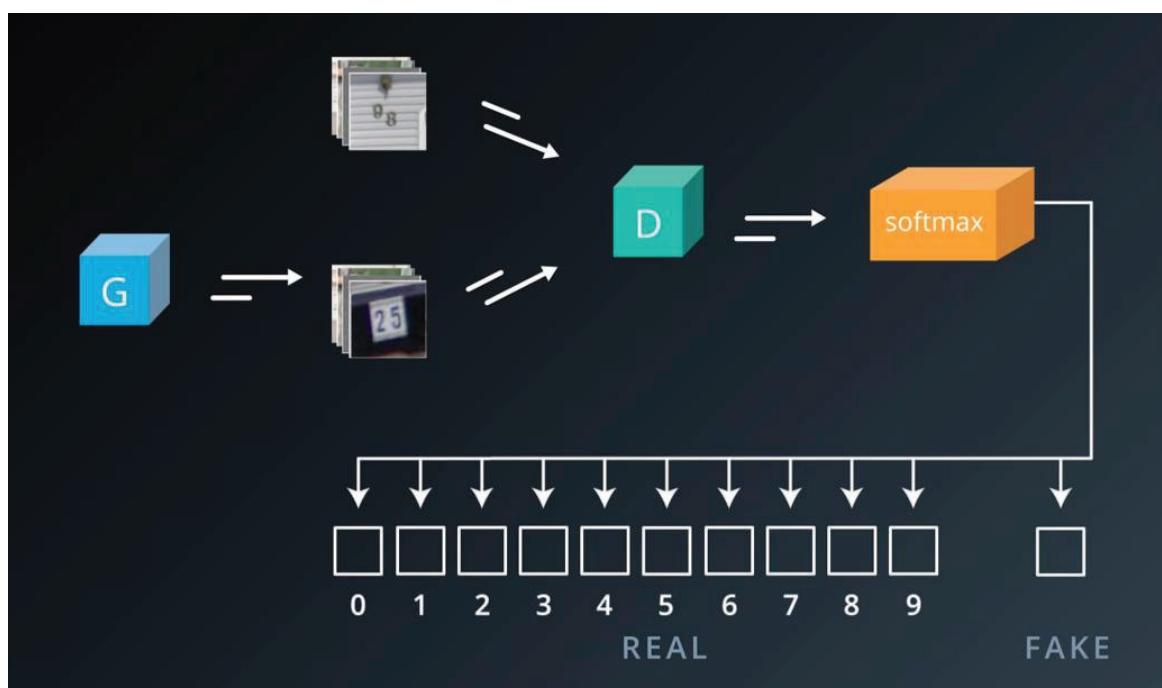
## Training tips

- ▶ While training the discriminator hold the generator values constant and vice versa
- ▶ Take care that each side does not overpower the other
- ▶ E.g. if the discriminator is very good, it will return values close to 0 or 1 and the generator will struggle to read the gradient.

<https://deeplearning4j.org/generative-adversarial-network>, 21.06.2018

- ▶ Semi-Supervised:  
Algorithm can use both labeled and unlabeled data to improve performance!
- ▶ Relevance: Labeled data is expensive!!!
  
- ▶ This far:  
Use generator of GAN and throw discriminator away...
  
- ▶ Now:  
Use discriminator of GAN!

## Architecture

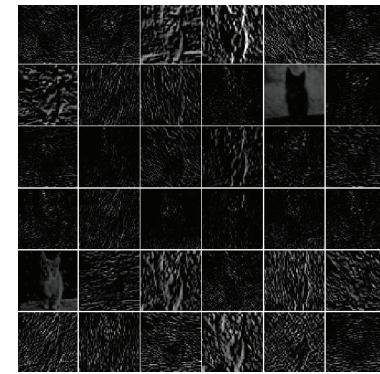


- ▶ Check out Intro\_to\_GANs example
- ▶ Check out Semi-supervised\_learning example  
(optional homework)

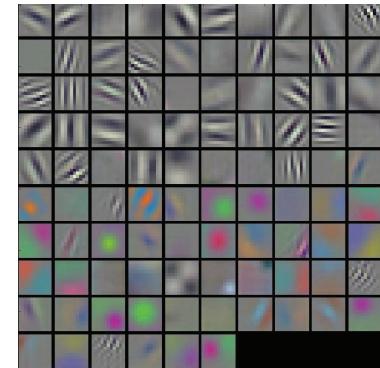
## Outline

- ▶ Autoencoders
- ▶ Generative Adversarial Networks
- ▶ Visualization + Fooling Networks

- ▶ Visualize layer activations during forward pass (e.g. AlexNet looking at a cat)
- ▶ Zero-activations for different inputs indicate dead filters, e.g. due to high learning rates
  
- ▶ Visualize weights best interpretable for 1<sup>st</sup> layer
- ▶ Noise indicate poor regularization or the network has not been trained long enough



Source: <http://cs231n.github.io/understanding-cnn/>, 19.09.2017



# Visualization

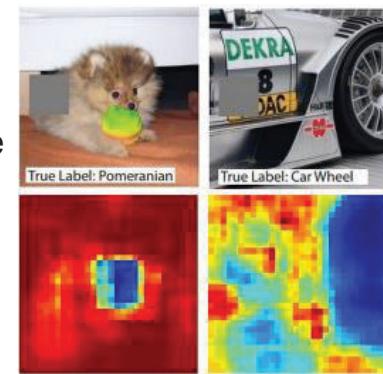
- ▶ Find images that maximally activate a neuron
  - ▶ Feed many images through the network and keep track of images that maximally activate a neuron
  
- ▶ Visualize CNN-codes with t-SNE
  - ▶ ConvNet to compute CNN codes (e.g. in our example 1024-dimensional vector)
  - ▶ t-SNE produces a 2-dimensional vector for each image which can be visualized.
  - ▶ Images that are nearby each other are also close in the CNN representation space → similarity



Source: <http://cs231n.github.io/understanding-cnn/>, 19.09.2017

► Occluding parts of the image

- Zero-out a sliding window of the image to be classified
- Visualize the probability results of the network as heat map
- Find out, which image region is responsible for classification score



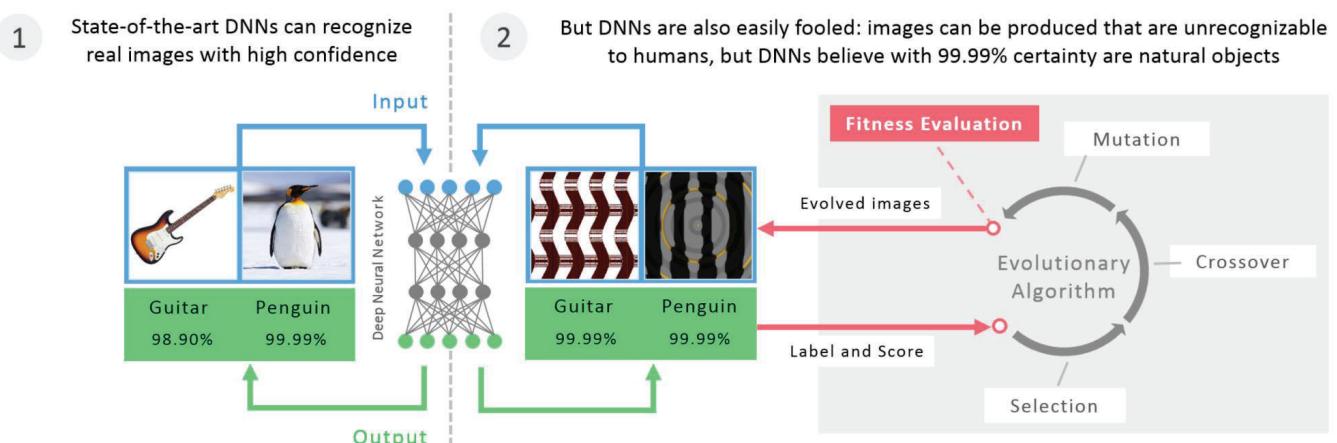
► Many other methods available...

# Fooling Deep Learning

► Approach 1:

"Nguyen A, Yosinski J, Clune J. Deep Neural Networks are Easily Fooled "

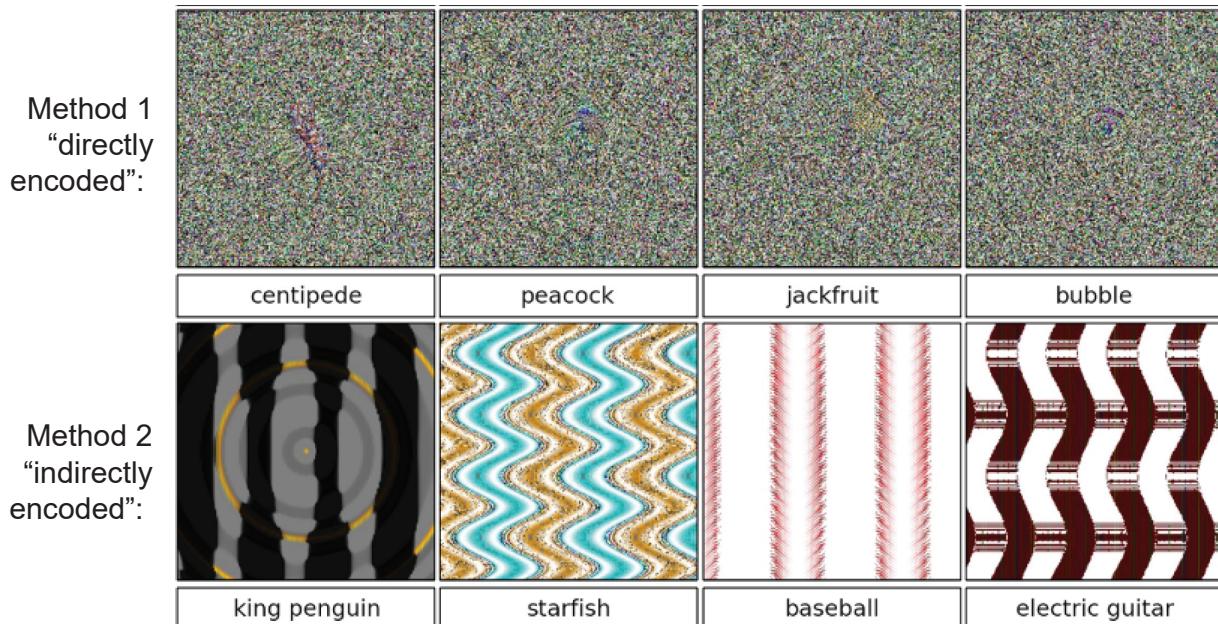
1. Use Deep Neural Network, trained on ImageNet for prediction
2. Optimization algorithm creates image to maximize prediction output of DNN



Source: Nguyen A, Yosinski J, Clune J. Deep Neural Networks are Easily Fooled: High Confidence Predictions for Unrecognizable Images. In Computer Vision and Pattern Recognition (CVPR '15), IEEE, 2015.

# Fooling Deep Learning - Result

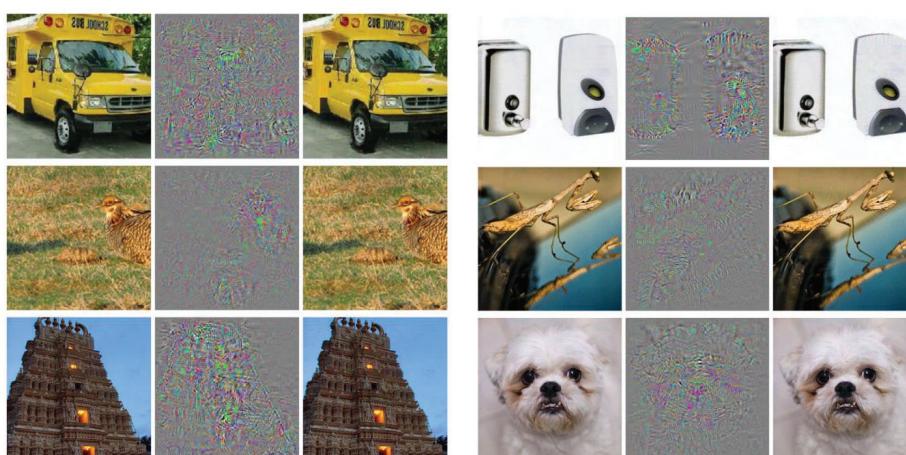
- DNN classified images with  $\geq 99.6\%$  certainty to...



Source: Nguyen A., Yosinski J., Clune J. Deep Neural Networks are Easily Fooled: High Confidence Predictions for Unrecognizable Images. In Computer Vision and Pattern Recognition (CVPR '15), IEEE, 2015.

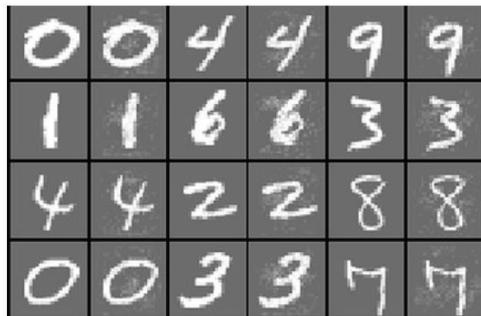
# Fooling Deep Learning

- Approach 2:  
“Szegedy et. al.: Intriguing properties of neural networks”
  - Add “special noise” to the image → misclassification of the object by neural network
  - Noise is found by maximizing the network’s prediction error
- Result:
  - AlexNet predicted all images with “noise” to: “ostrich, Struthio camelus”

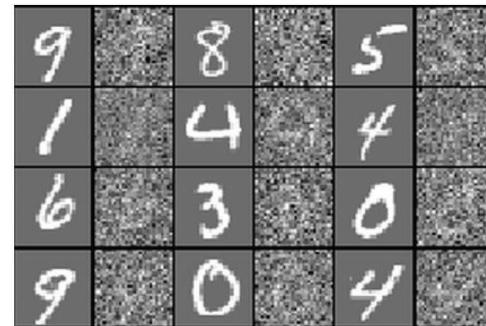


Source: C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus. Intriguing properties of neural networks. arXiv preprint arXiv:1312.6199, 2013.

- ▶ Result on MNIST digits:



(b) Even columns: adversarial examples for a 200-200-10 sigmoid network (stddev=0.063)



(c) Randomly distorted samples by Gaussian noise with stddev=1. Accuracy: 51%.



## Autonomous Systems: Deep Learning

### 10. Generative Adversarial Networks (GAN)



Prof. Dr.-Ing. Nicolaj Stache, Friedrich Carrle

Heilbronn University of Applied Sciences

#### Master's thesis



#### ► Creating synthetic EEG-data with generative methods

- Medical informatics
- EEG-data of depressed (MDD) and healthy (HC) patients
- Classification on that data
- Improvement of classification accuracy through synthetic data (more training data available)
- Used a conditional Wasserstein Generative Adversarial Network



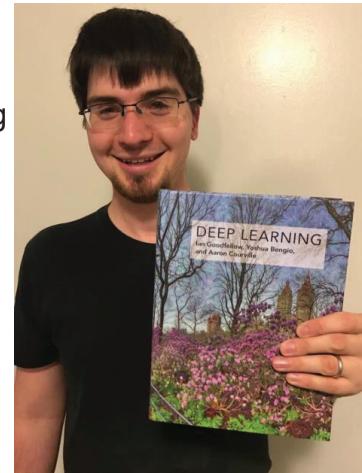
UNIVERSITÄT  
HEIDELBERG  
ZUKUNFT  
SEIT 1386



HOCHSCHULE HEILBRONN



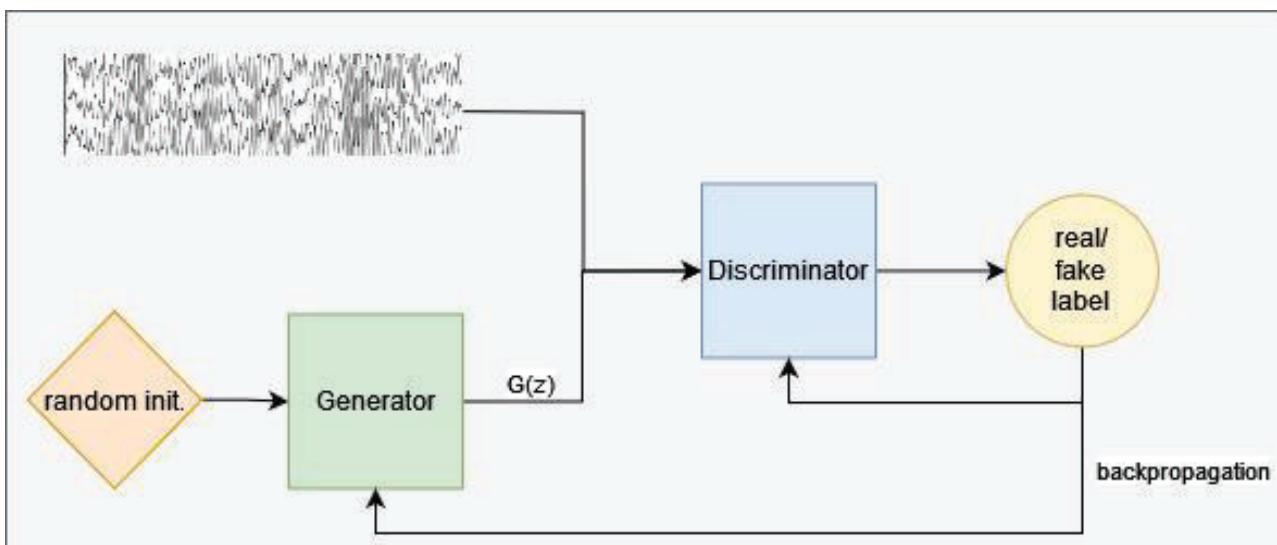
- ▶ Invented by Ian Goodfellow in 2014
- ▶ Key idea:
  - ▶ GANs are two neural networks competing against each other
- ▶ Generative algorithm:
  - ▶ Creates data
  - ▶ Learns distribution of data
- ▶ Discriminative algorithm:
  - ▶ Classification, given input data
  - ▶ Learn boundaries of class



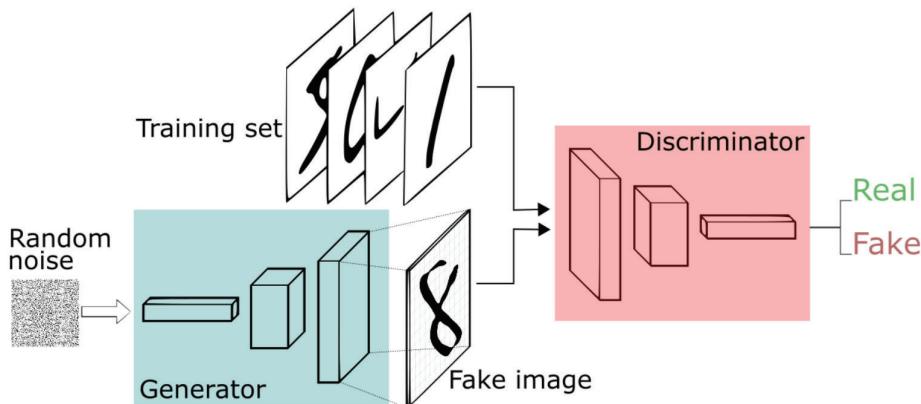
[https://twitter.com/goodfellow\\_iain/status/806103072158060546](https://twitter.com/goodfellow_iain/status/806103072158060546), 21.06.2018

## GAN architecture

- ▶ Network 1: Generator, generates new data, e.g. EEG-data, painting of Picasso
- ▶ Network 2: Discriminator, decides which data is real or fake, e.g. is it EEG-data, a painting of Picasso or not



- ▶ Networks optimize an opposing objective function
- ▶ Both are linked:
  - ▶ when Generator changes, the Discriminator has to react and vice versa



<https://deeplearning4j.org/generative-adversarial-networks>, 21.06.2018

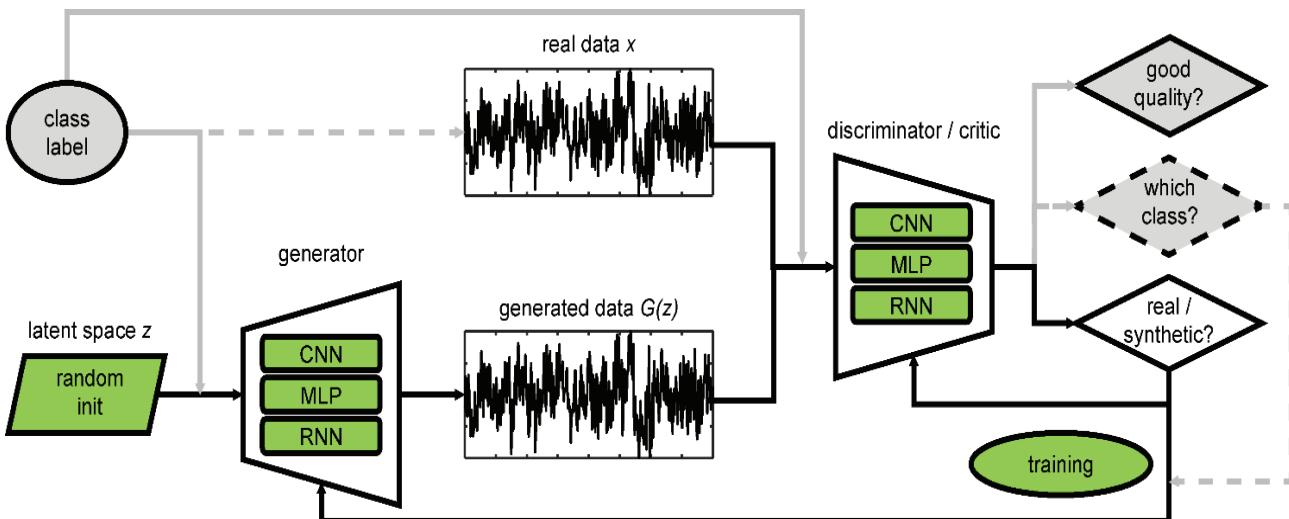
## Common problems

- ▶ Vanishing gradient
  - ▶ Discriminator too powerful
  - ▶ Generator cannot improve
  - ▶ “optimal” discriminator bad
  - ▶ Wasserstein loss as remedy
- ▶ Mode collapse
  - ▶ Generator produces a limit set of similar samples that “work”
  - ▶ Fails to capture diversity of the real data
  - ▶ Add more layers to Generator, adjust hyper parameters
- ▶ Evaluation/Blackbox
  - ▶ Assessing the quality and diversity of generated samples

- ▶ Wasserstein GAN
  - ▶ Uses Wasserstein loss function
  - ▶ Discriminator called Critic
- ▶ DCNN/RNN GAN
  - ▶ Generator and Discriminator networks can vary
- ▶ Selective GAN
  - ▶ Only use “good” data
- ▶ Conditional GAN
  - ▶ Adds label information
- ▶ Auxiliary GAN
  - ▶ Discriminator additionally trained to predict class label

# Variants of GANs

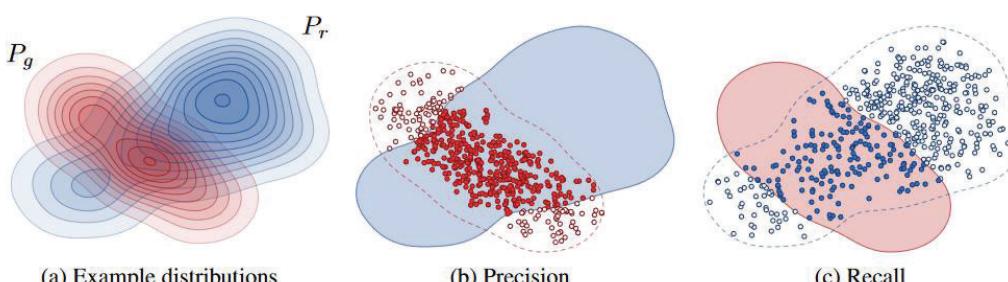
- ▶ Green: Variants
- ▶ Grey: Optional elements
- ▶ Dashed: Auxiliary Classifier



- ▶ Conditional GANs
- ▶ Auxiliary GANs
- ▶ Create only minority class
- ▶ One GAN per class
- ▶ Posthoc classification
  - ▶ Classifier
  - ▶ Expert

## Evaluation

- ▶ Generator/ Discriminator loss
- ▶ Visual inspection
  - ▶ Only for detecting of bad samples!
- ▶ Expert opinion
- ▶ Distance measures real - fake
  - ▶ Wasserstein, Manhattan, Dynamic Time Warping
  - ▶ Use case dependent
- ▶ Precision and Recall
  - ▶ Careful, explained variance
  - ▶ <https://arxiv.org/abs/1904.06991>



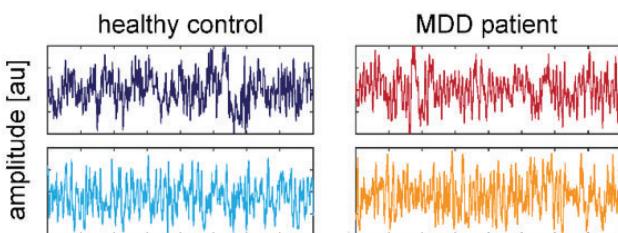
- ▶ While training the Discriminator hold the Generator values constant and vice versa
- ▶ Train Discriminator for more (5) iterations for every Generator iteration
- ▶ Take care that each side does not overpower the other
- ▶ E.g. if the Discriminator is very good, it will return values close to 0 or 1 and the Generator will struggle to read the gradient.
- ▶ Augmentation of data
  - ▶ Augment input of Discriminator for better generalization
  - ▶ Improves diversity of samples

<https://deeplearning4j.org/generative-adversarial-networks>, 21.06.2018

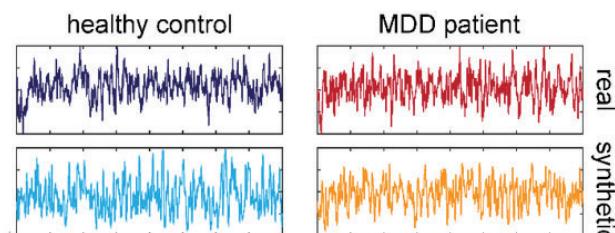
## Applications

- ▶ Not restricted on images
- ▶ Improve classification accuracy
  - ▶ More training data
  - ▶ Balance classes

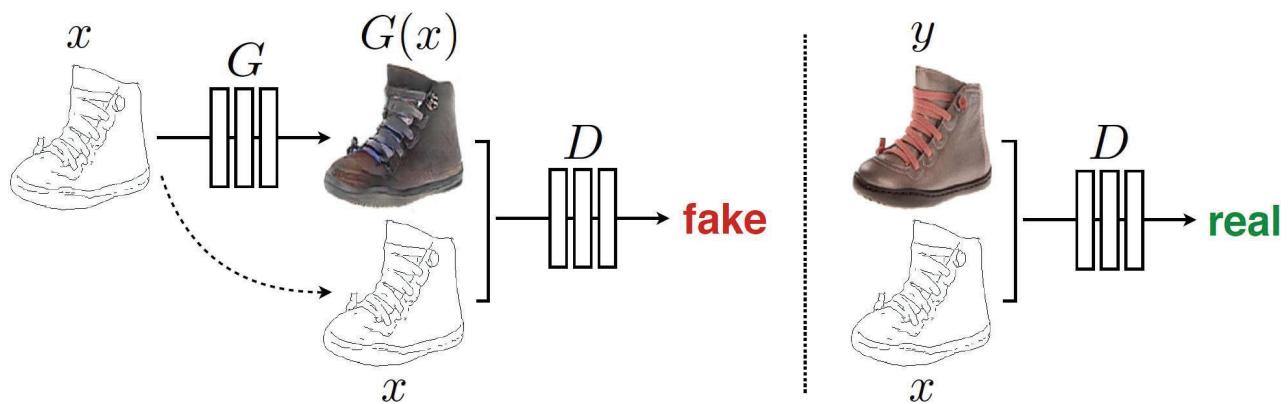
(A) dataset 1 (channel F3)



(B) dataset 2 (channel F4)



- ▶ Generate new photorealistic images  
(which have never been seen before)  
(GAN is drawing this from the learned probability distribution over all hypothetical images matching to the Discriminator)
  - ▶ Image-to-Image Translation  
(see: <https://arxiv.org/abs/1611.07004>)



Auto.-Sys: Deep Learning

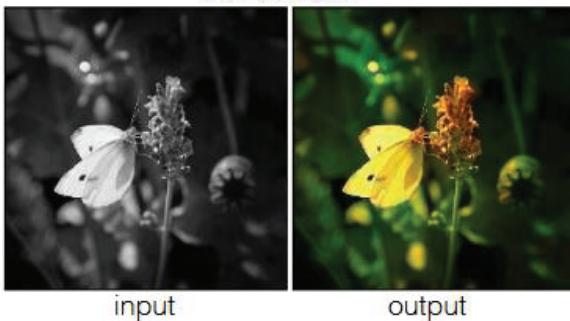
Prof. Dr. N. Stache

13

Isola et. Al.: Image-to-Image Translation with Conditional Adversarial Networks, 11.2017.

## Applications

## BW to Color



## Map to aerial photo



## Aerial photo to map

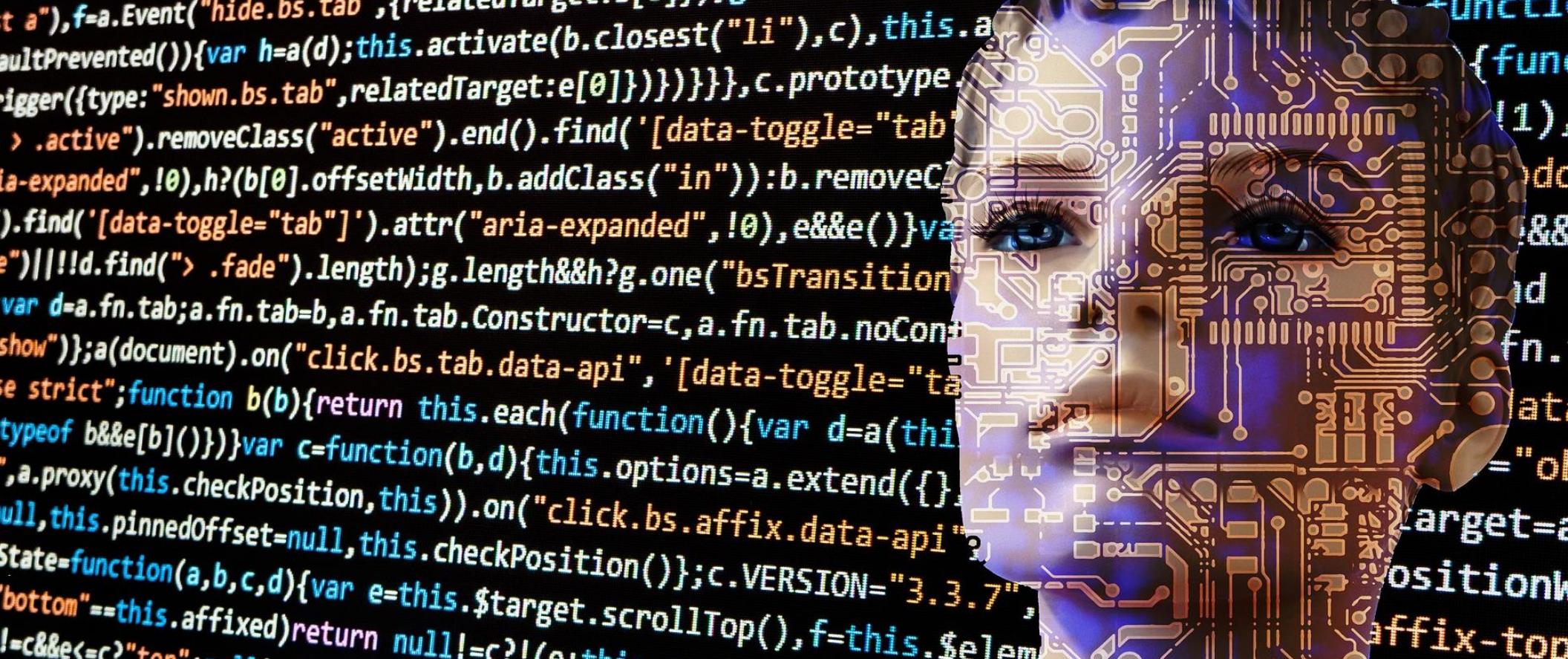


Isola et. Al.: Image-to-Image Translation with Conditional Adversarial Networks, 11.2017.

- ▶ This far:  
Use generator of GAN and throw discriminator away...
- ▶ Use Discriminator:
  - ▶ Anomaly Detection
  - ▶ Feature Extraction
  - ▶ Auxiliary GAN

## Optional: Hands on part with GANs

- ▶ Check out `Intro_to_GANs` example
- ▶ Check out `Semi-supervised_learning` example  
(optional homework)



# Autonomous Systems: Deep Learning Transformers

# Motivation

## So far: RNN based Seq2Seq

- ▶ Processes data sequentially
- ▶ Captures timely dependencies in sequences

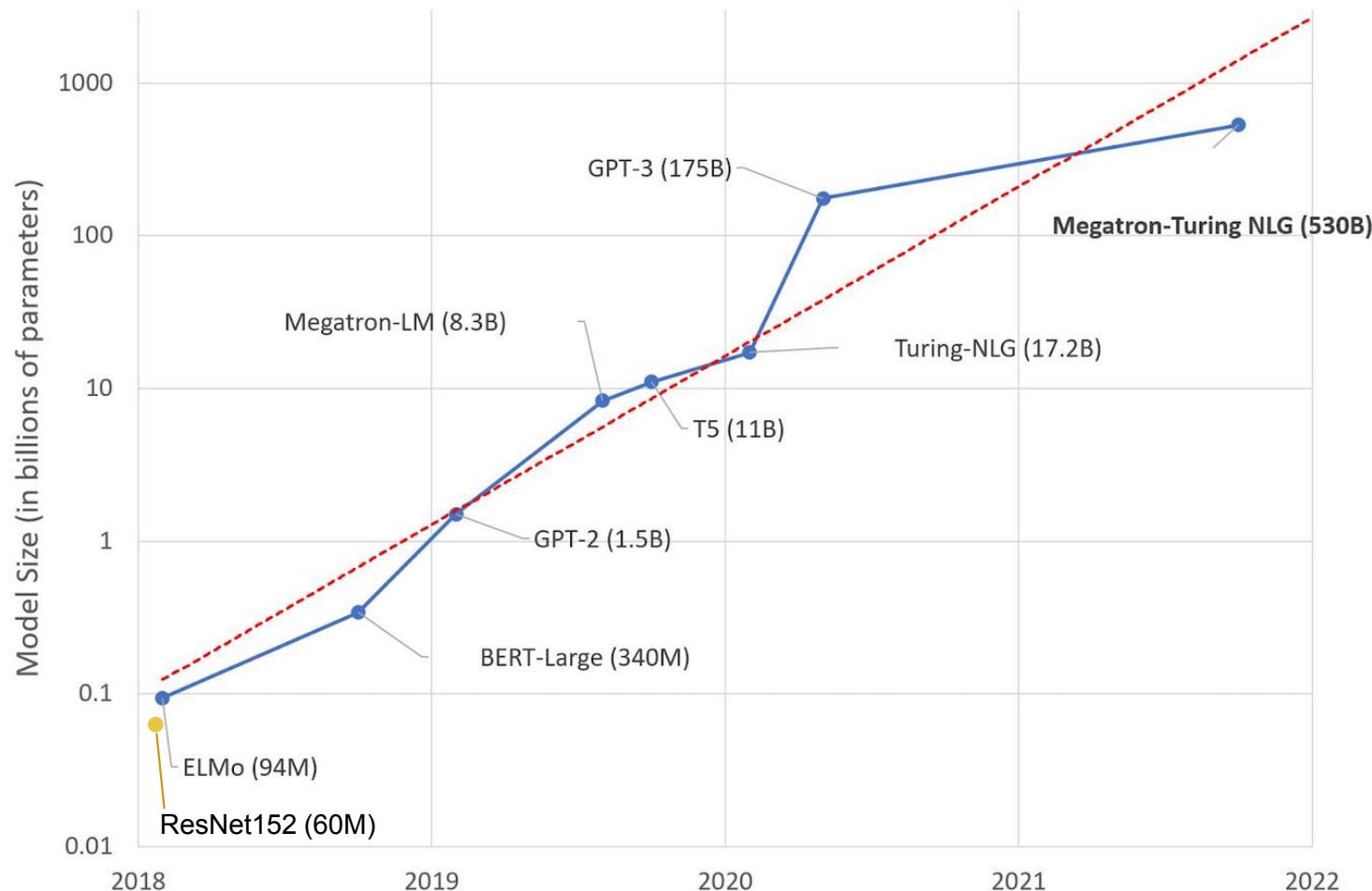
## Problem:

- ▶ The sequential nature prevents parallelization
- ▶ Struggles with long-range dependencies

## Solution:

- ▶ Transformers

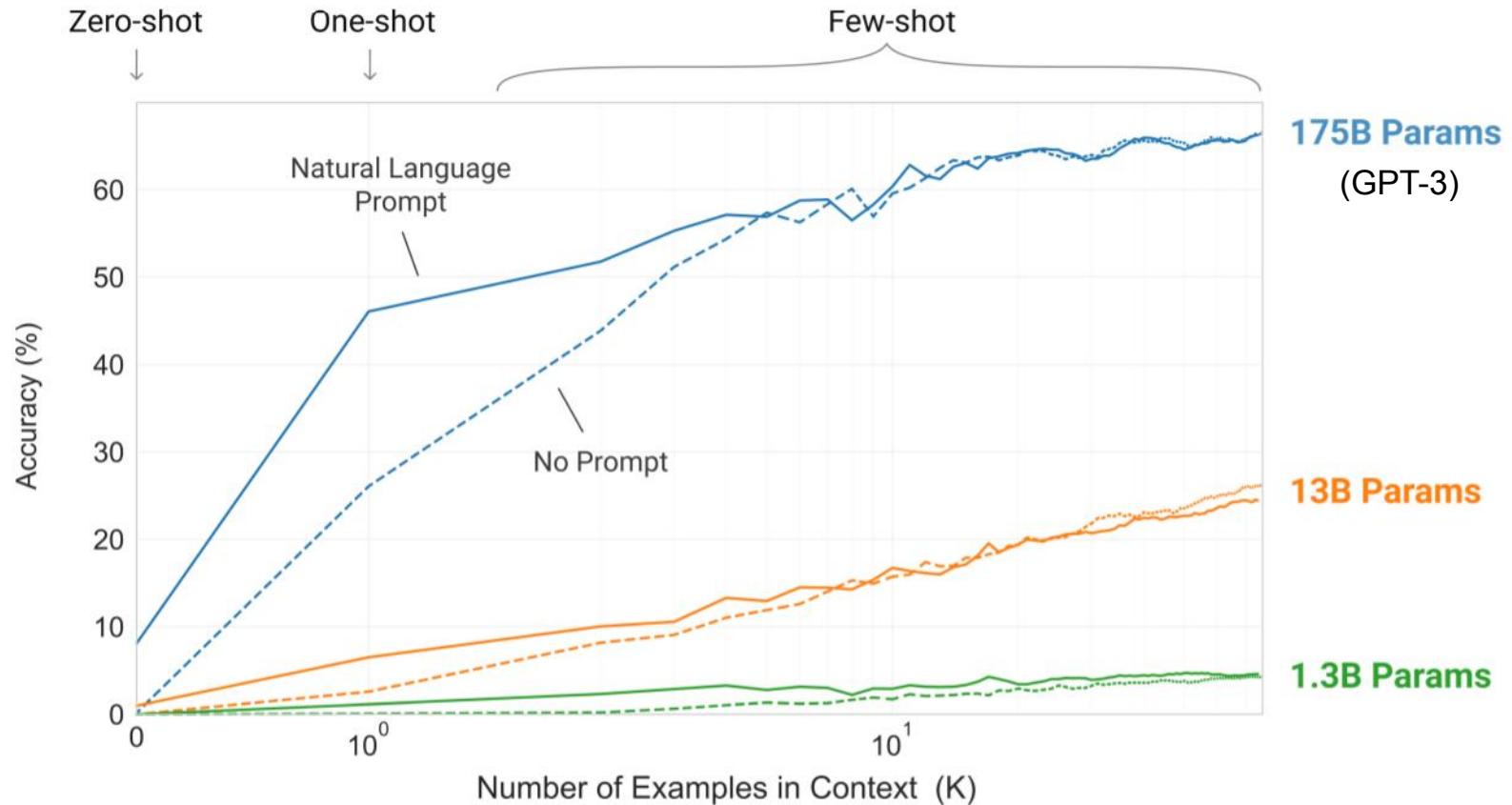
# Transformers



<https://huggingface.co/blog/large-language-models> 01.02.2022

- ▶ Size of transformers grows at an exponential rate

# Transformers



<https://arxiv.org/pdf/2005.14165.pdf>

- More parameters result in better accuracy

# GPT-3

- ▶ 175 billion trainable parameters
- ▶ Trained on about 45 TB of text data
- ▶ Training would cost \$4.5 million and would take 355 years on a V100 GPU server (28 TFLOPS capacity)

## Example Tasks:

**Q&A**

Answer questions based on existing knowle...

**Summarize for a 2nd grader**

Translates difficult text into simpler concep...

**Python bug fixer**

Find and fix bugs in source code.

**Recipe creator (eat at your own risk)**

Create a recipe from a list of ingredients.

**Explain code**

Explain a complicated piece of code.

**Ad from product description**

Turn a product description into ad copy.

→ Try it out on: <https://beta.openai.com/>

# Example

beta.openai.com/playground

Overview Documentation Examples Playground [Upgrade](#) [Help](#) [Personal](#)

**Get started**

Enter some text or select a preset, and watch the API respond with a [completion](#) that attempts to match the context or pattern you provided.

You can control which [model](#) completes your request by changing the engine.

**KEEP IN MIND**

- ⚠ Use good judgment when sharing outputs, and attribute them to your name or company. [Learn more](#).
- ⚠ Requests submitted to our models may be used to train and improve future models. [Learn more](#).
- ⚠ Most models' training data cuts off in October 2019, so they may not have knowledge of current events.

**Playground**

Create a cocktail recipe based on the name of the cocktail

**Name:** Painkiller Cocktail

**INGREDIENTS**

4 oz. pineapple juice  
2 oz. dark rum  
1 oz. fresh-squeezed orange juice  
1 oz. cream of coconut  
Ice  
freshly grated nutmeg  
halved orange slice, for garnish

**DIRECTIONS**

Fill a cocktail shaker with ice, then add pineapple juice, rum, orange juice, and cream of coconut. Shake until cold, about 15 to 30 seconds. Pour into a glass filled with ice and grate nutmeg on top. Garnish with orange slice and serve.

**Name:** Classic Eggnog

**INGREDIENTS**

2 c. milk  
1/2 tsp. ground cinnamon, plus more for garnish  
1/2 tsp. ground nutmeg  
1/2 tsp. pure vanilla extract  
6 large egg yolks  
1/2 c. granulated sugar  
1 c. heavy cream  
1 c. bourbon or rum (optional)  
Whipped cream, for serving

**DIRECTIONS**

In a small saucepan over low heat, combine milk, cinnamon, nutmeg, and vanilla and slowly bring mixture to a low boil. Meanwhile, in a large bowl, whisk egg yolks with sugar until yolks are pale in color. Slowly add hot milk mixture to egg yolks in batches to temper the eggs and whisk until combined. Return mixture to saucepan and cook over medium heat until slightly thick (and coats the back of a spoon) but does not boil. (If using a candy thermometer, mixture should

Load a preset... Save View code Share ...

Engine [text-davinci-001](#) What's new →

Temperature 0.7

Maximum length 810

Stop sequences Enter sequence and press Tab

Top P 1

Frequency penalty 0

Presence penalty 0

Best of 1

Inject start text

Inject restart text

Show probabilities Off

Generate

554

# Example

beta.openai.com/playground

Overview Documentation Examples Playground [Upgrade](#) [Help](#) [Personal](#)

**Get started** X

Enter some text or select a preset, and watch the API respond with a [completion](#) that attempts to match the context or pattern you provided.

You can control which [model](#) completes your request by changing the engine.

**KEEP IN MIND**

- ⚠ Use good judgment when sharing outputs, and attribute them to your name or company. [Learn more](#).
- ⚠ Requests submitted to our models may be used to train and improve future models. [Learn more](#).
- ⚠ Most models' training data cuts off in October 2019, so they may not have knowledge of current events.

**Playground**

Load a preset... Save View code Share ...

**1c. heavy cream**  
**1c. bourbon or rum (optional)**  
 Whipped cream, for serving

**DIRECTIONS**  
 In a small saucepan over low heat, combine milk, cinnamon, nutmeg, and vanilla and slowly bring mixture to a low boil. Meanwhile, in a large bowl, whisk egg yolks with sugar until yolks are pale in color. Slowly add hot milk mixture to egg yolks in batches to temper the eggs and whisk until combined. Return mixture to saucepan and cook over medium heat until slightly thick (and coats the back of a spoon) but does not boil. (If using a candy thermometer, mixture should reach 160°.) Remove from heat and stir in heavy cream and, if using, bourbon. Refrigerate until chilled. When ready to serve, garnish with whipped cream and cinnamon.

**Name: Juicy colada**

**INGREDIENTS**  
 1 1/2 oz. light rum  
 1 oz. pineapple juice  
 1/2 oz. coconut cream  
 1/4 oz. simple syrup  
 1 lime wedge

**DIRECTIONS**  
 Fill a shaker with ice. Add light rum, pineapple juice, coconut cream, simple syrup, and lime wedge. Shake until well combined. Strain into a hurricane glass.

**Name: Maple old fashioned**

**INGREDIENTS**  
 2 oz. bourbon  
 1/2 oz. maple syrup  
 1 dash angostura bitters  
 orange slice, for garnish

**DIRECTIONS**  
 In a rocks glass, combine bourbon, maple syrup, and bitters. Add a large ice cube and stir until chilled. Garnish with orange slice.

**Engine** text-davinci-001 [What's new →](#)

Temperature 0.7

Maximum length 810

Stop sequences Enter sequence and press Tab

Top P 1

Frequency penalty 0

Presence penalty 0

Best of 1

Inject start text

Inject restart text

Show probabilities Off

Generate 554

# Applications

## Question Answering

### Passage Sentence

In meteorology, precipitation is any product of the condensation of atmospheric water vapor that falls under gravity.

### Question

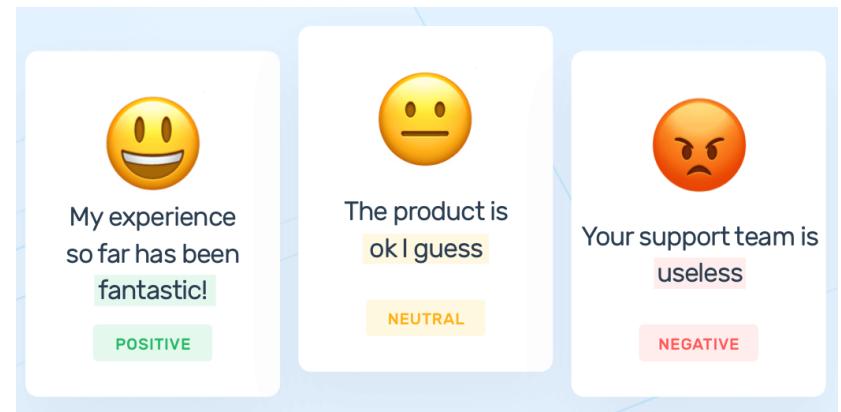
What causes precipitation to fall?

### Answer Candidate

gravity

<https://rajpurkar.github.io/mlx/qa-and-squad/> 01.02.2022

## Sentiment Analysis



<https://monkeylearn.com/sentiment-analysis/> 01.02.2022

## Named Entity Recognition

Ousted WeWork founder Adam Neumann lists his Manhattan penthouse for \$37.5 million

[organization]

[person]

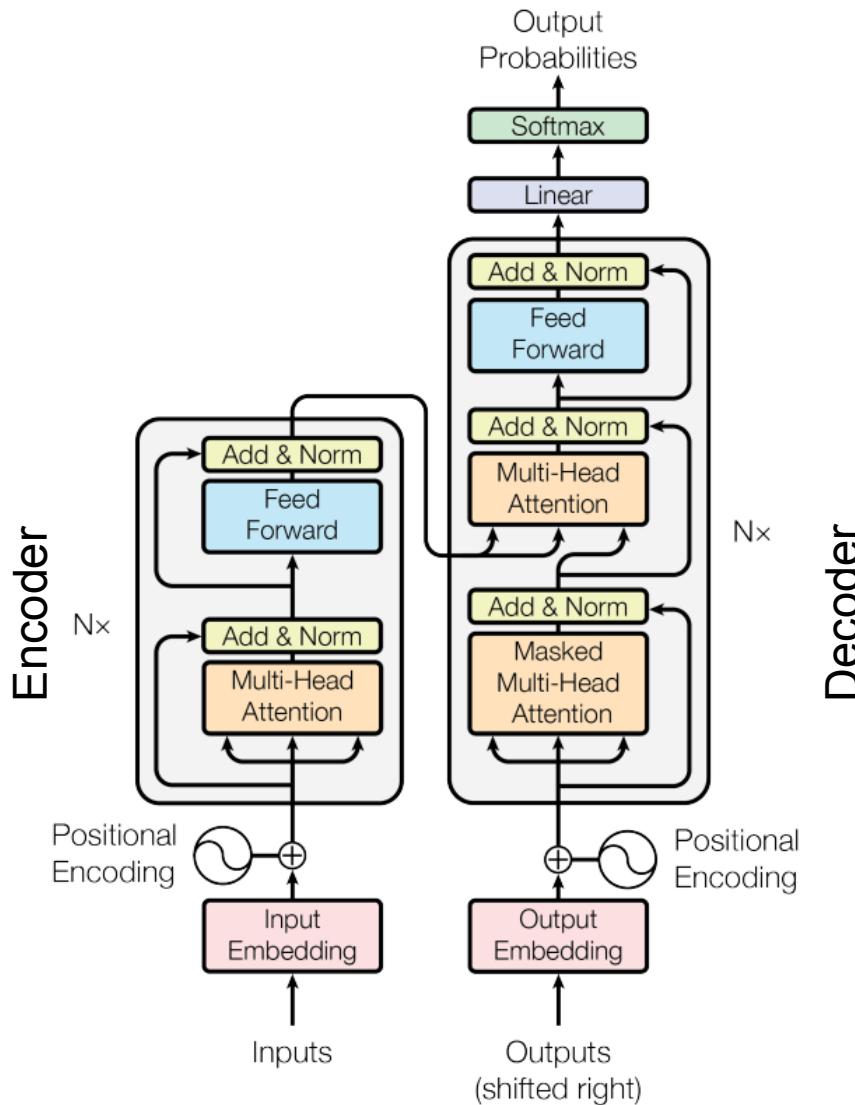
[location]

[monetary value]

<https://monkeylearn.com/blog/named-entity-recognition/> 01.02.2022

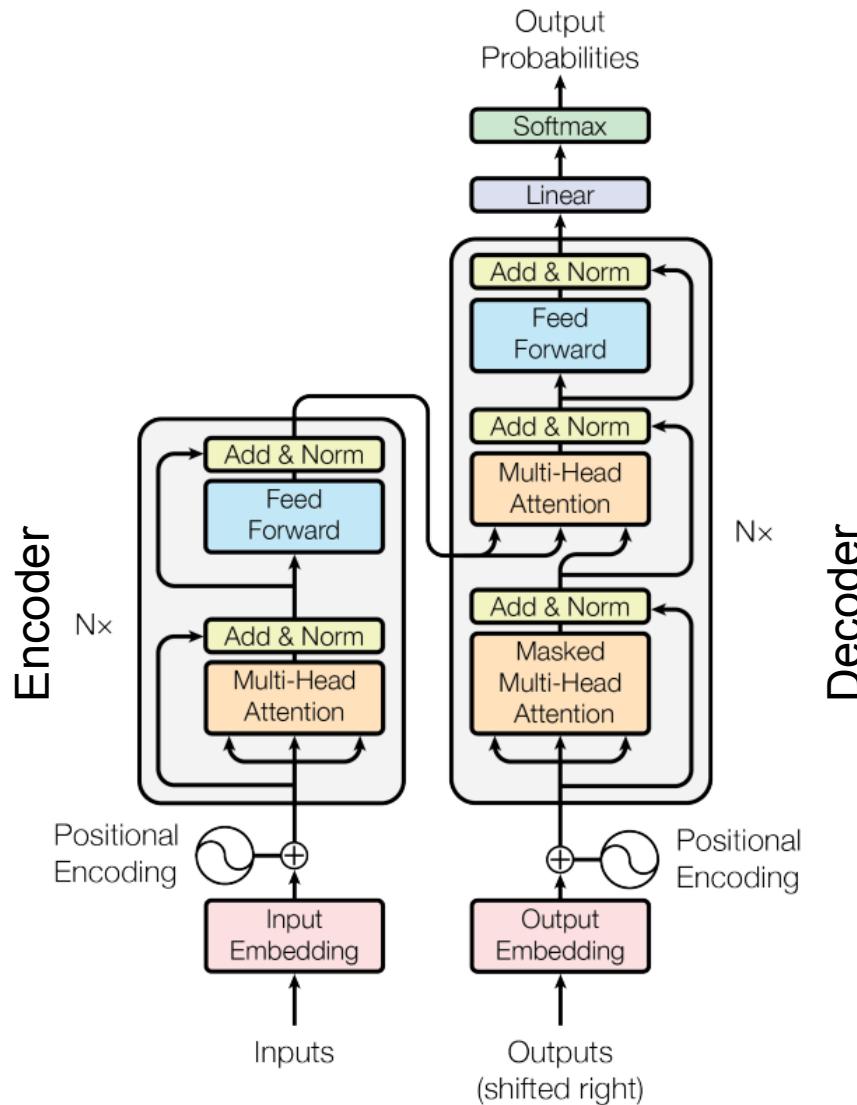
And many more...

# “Attention is all you need” 2017



- ▶ Encoder-Decoder Structure
- ▶ No Recurrence
- ▶ Outperformed all SOTA Algorithms in NLP Tasks

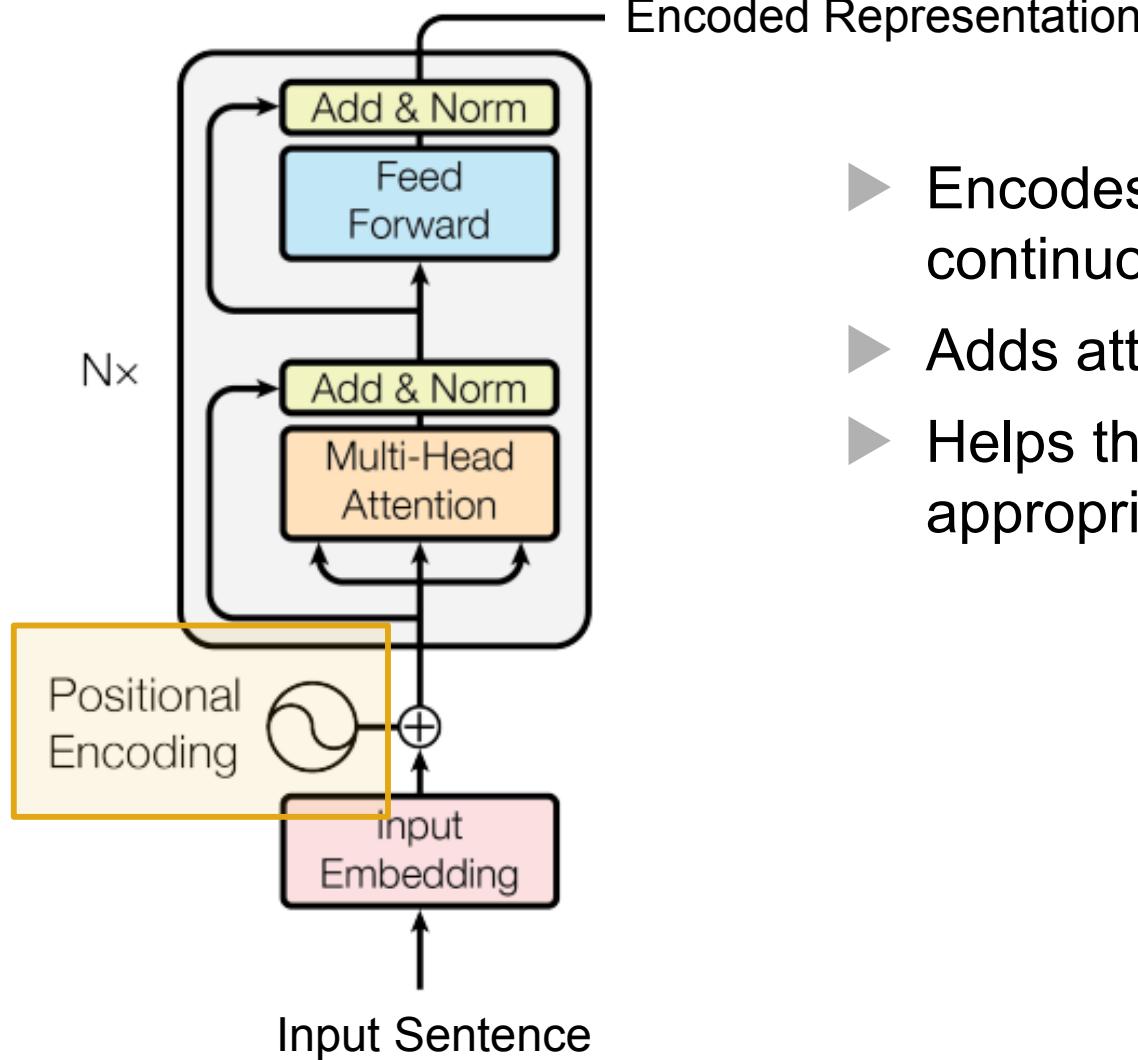
# Transformers



## Whats new?

- ▶ Positional Encodings
- ▶ Self-Attention
- ▶ Multi-Head Attention
- ▶ Masked Multi-Head Attention

# Encoder



- ▶ Encodes the input into a continuous representation
- ▶ Adds attention information
- ▶ Helps the decoder to focus on appropriate words

# Positional Encodings

- ▶ It should output a unique encoding for each time-step
- ▶ Distance between any two time-steps should be consistent across sentences with different lengths.
- ▶ It must be deterministic.

$$\vec{p}_t^{(i)} = f(t)^{(i)} := \begin{cases} \sin(\omega_k \cdot t), & \text{if } i = 2k \\ \cos(\omega_k \cdot t), & \text{if } i = 2k + 1 \end{cases}$$

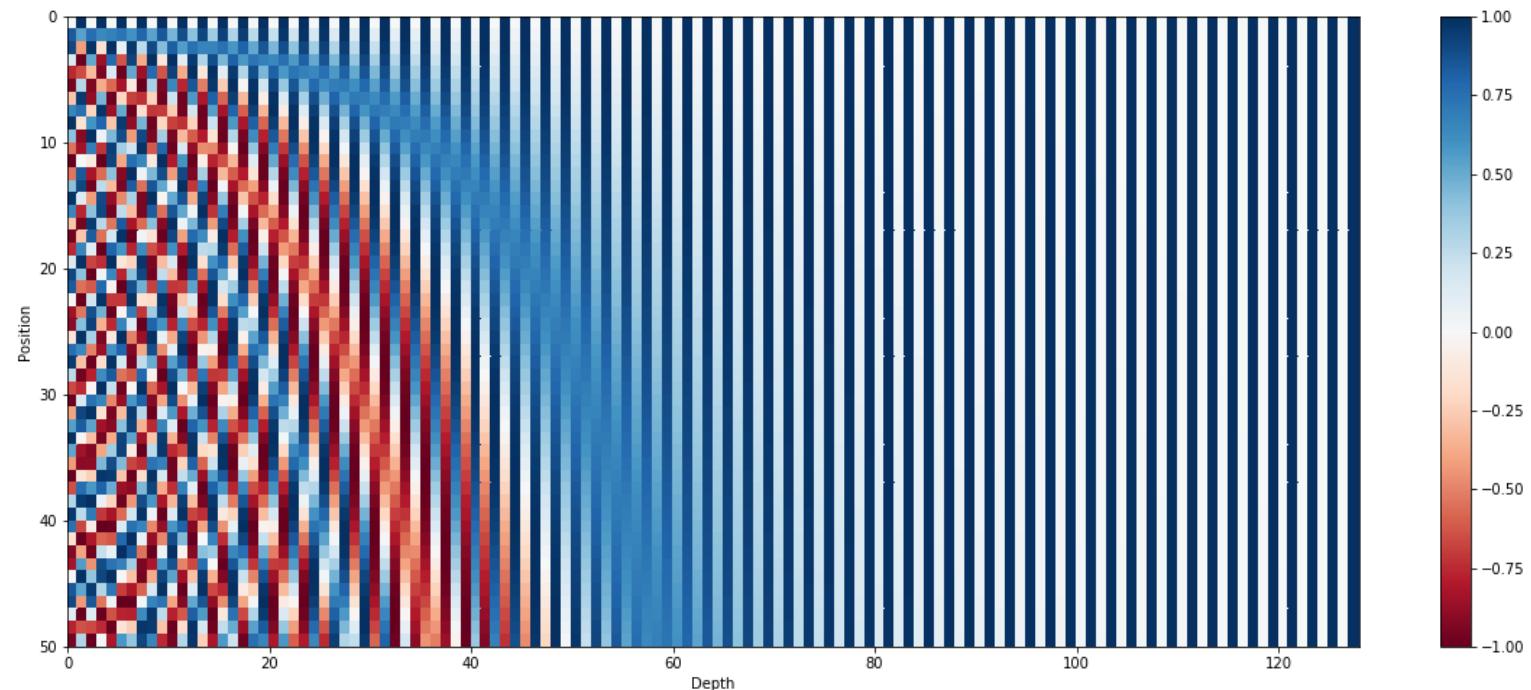
$$\vec{p}_t = \begin{bmatrix} \sin(\omega_1 \cdot t) \\ \cos(\omega_1 \cdot t) \\ \sin(\omega_2 \cdot t) \\ \cos(\omega_2 \cdot t) \\ \vdots \\ \sin(\omega_{d/2} \cdot t) \\ \cos(\omega_{d/2} \cdot t) \end{bmatrix}_{d \times 1}$$

$$\omega_k = \frac{1}{10000^{2k/d}}$$

[https://kazemnejad.com/blog/transformer\\_architecture\\_positional\\_encoding/](https://kazemnejad.com/blog/transformer_architecture_positional_encoding/) 01.02.2022

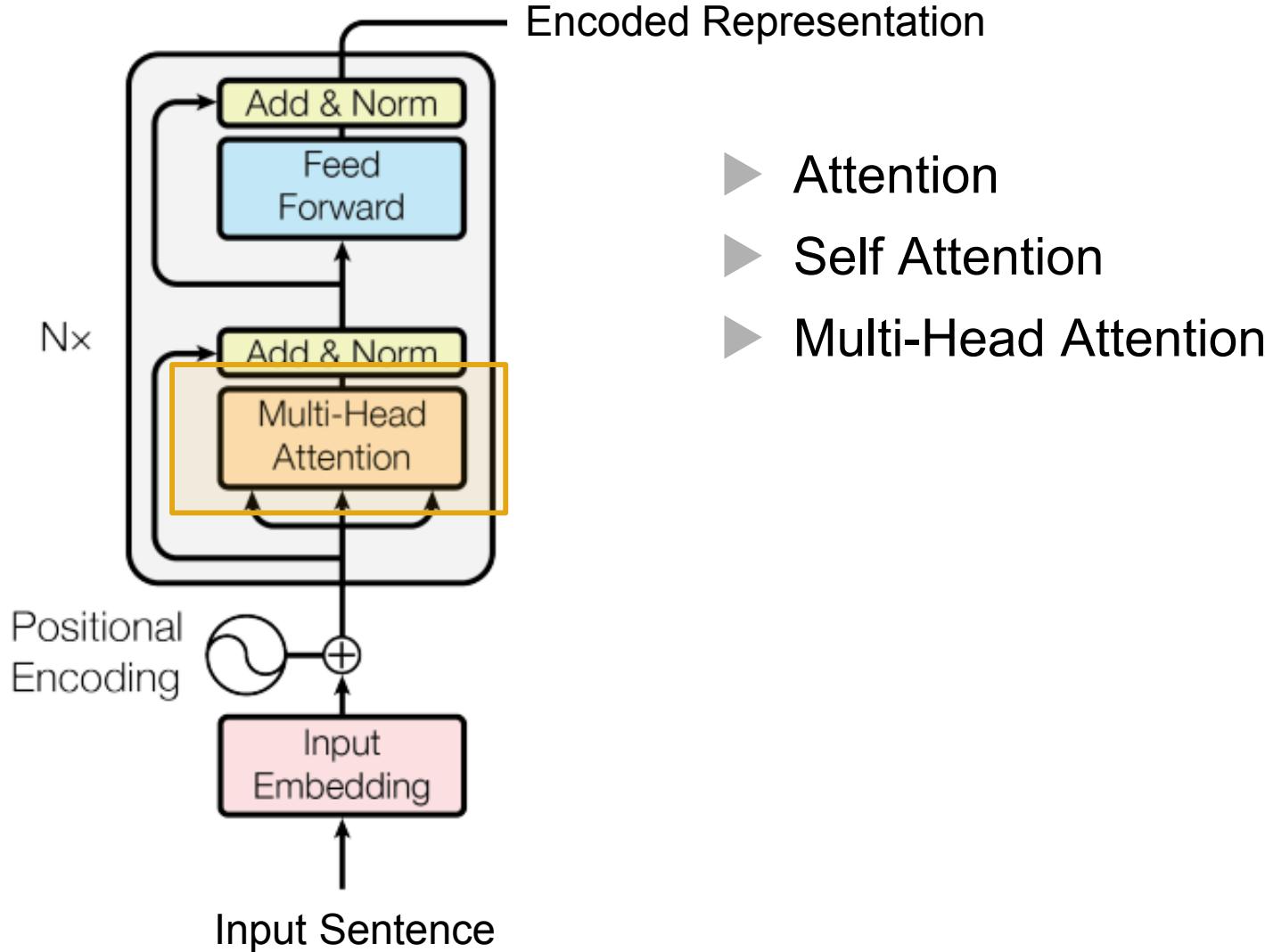
# Positional Encodings

0 :	0	0	0	0	0	1	0	0	0	0
1 :	0	0	0	1	0	1	0	0	1	0
2 :	0	0	1	0	0	1	0	1	0	0
3 :	0	0	1	1	0	1	0	1	1	0
4 :	0	1	0	0	0	1	1	0	0	0
5 :	0	1	0	1	0	1	1	1	0	1
6 :	0	1	1	0	0	1	1	1	0	0
7 :	0	1	1	1	0	1	1	1	1	0
8 :	1	0	0	0	0	0	1	0	0	0
9 :	1	0	0	1	0	0	1	0	0	1
10 :	1	0	1	0	0	0	1	0	1	0
11 :	1	0	1	1	0	0	1	0	1	1
12 :	1	1	0	0	0	0	1	1	0	0
13 :	1	1	0	1	0	0	1	1	0	1
14 :	1	1	1	0	0	0	1	1	1	0
15 :	1	1	1	1	0	0	1	1	1	1



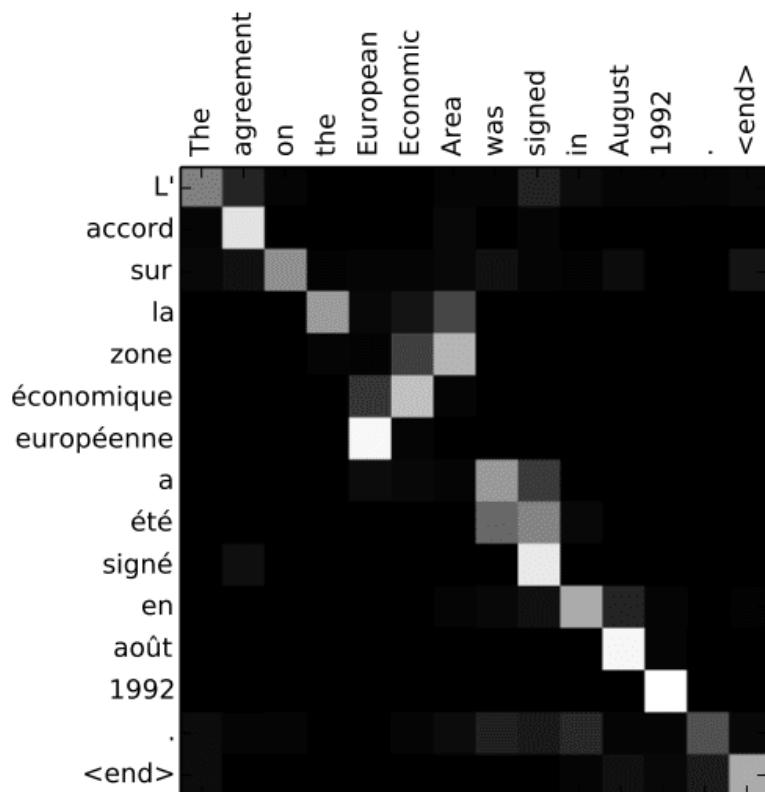
[https://kazemnejad.com/blog/transformer\\_architecture\\_positional\\_encoding/](https://kazemnejad.com/blog/transformer_architecture_positional_encoding/) 01.02.2022

# Encoder



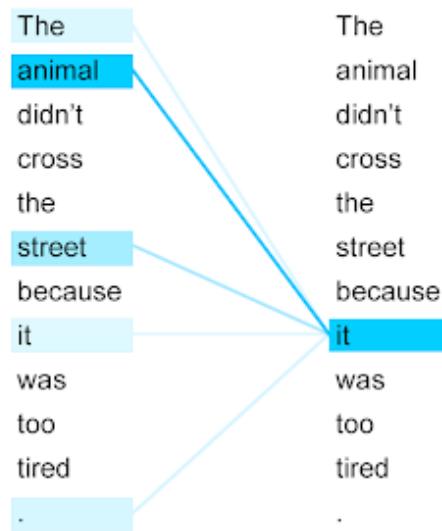
# Attention

- ▶ Mimics the human attention
- ▶ Focusing on a few relevant parts
- ▶ Relations between Input und Output Sequence
- ▶ Enables better processing of very long sequences

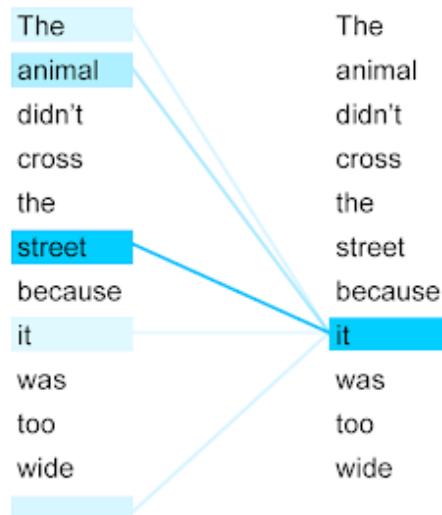


<https://arxiv.org/pdf/1409.0473.pdf> 01.02.2022

# Self-Attention

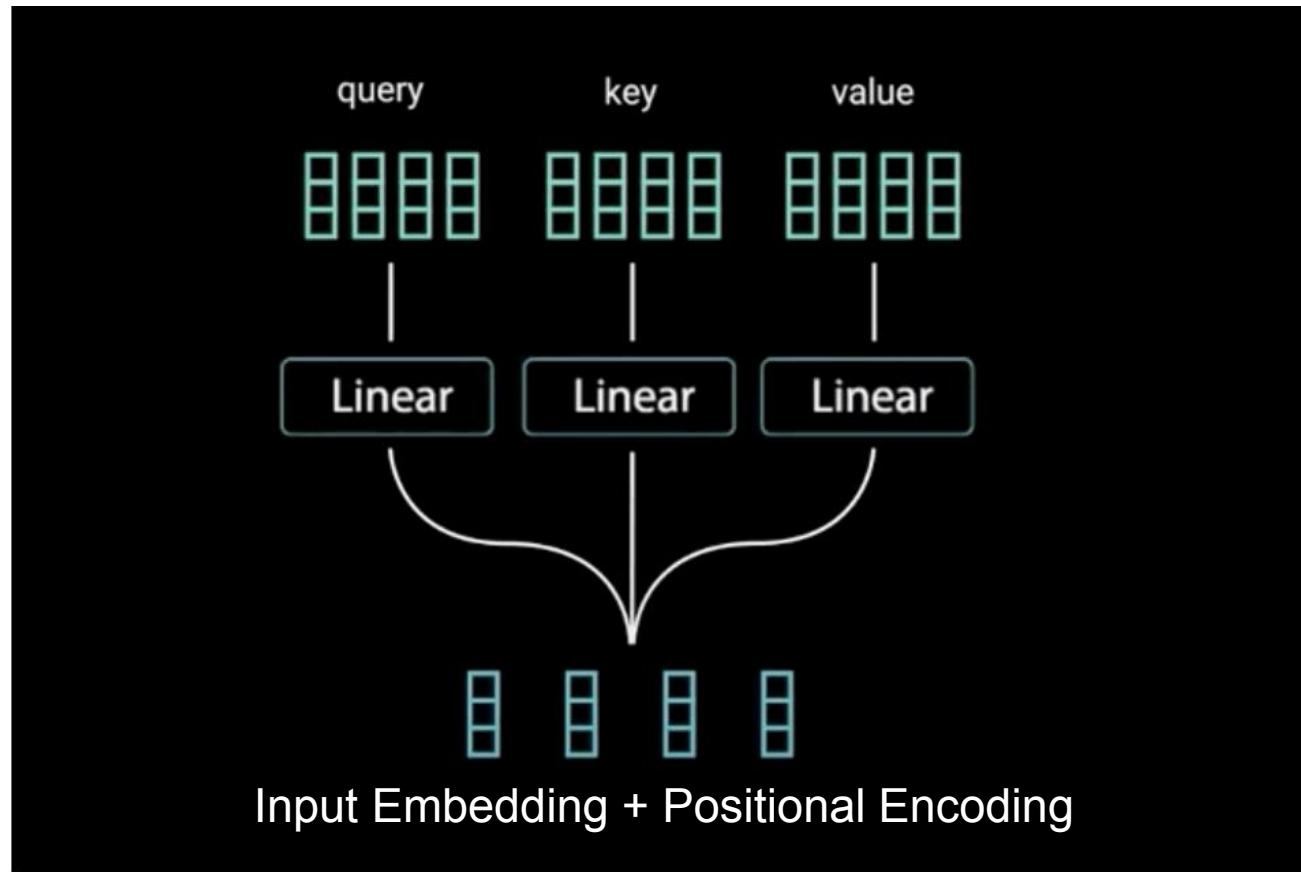


- ▶ Relations inside the same sequence
- ▶ Better understanding of context



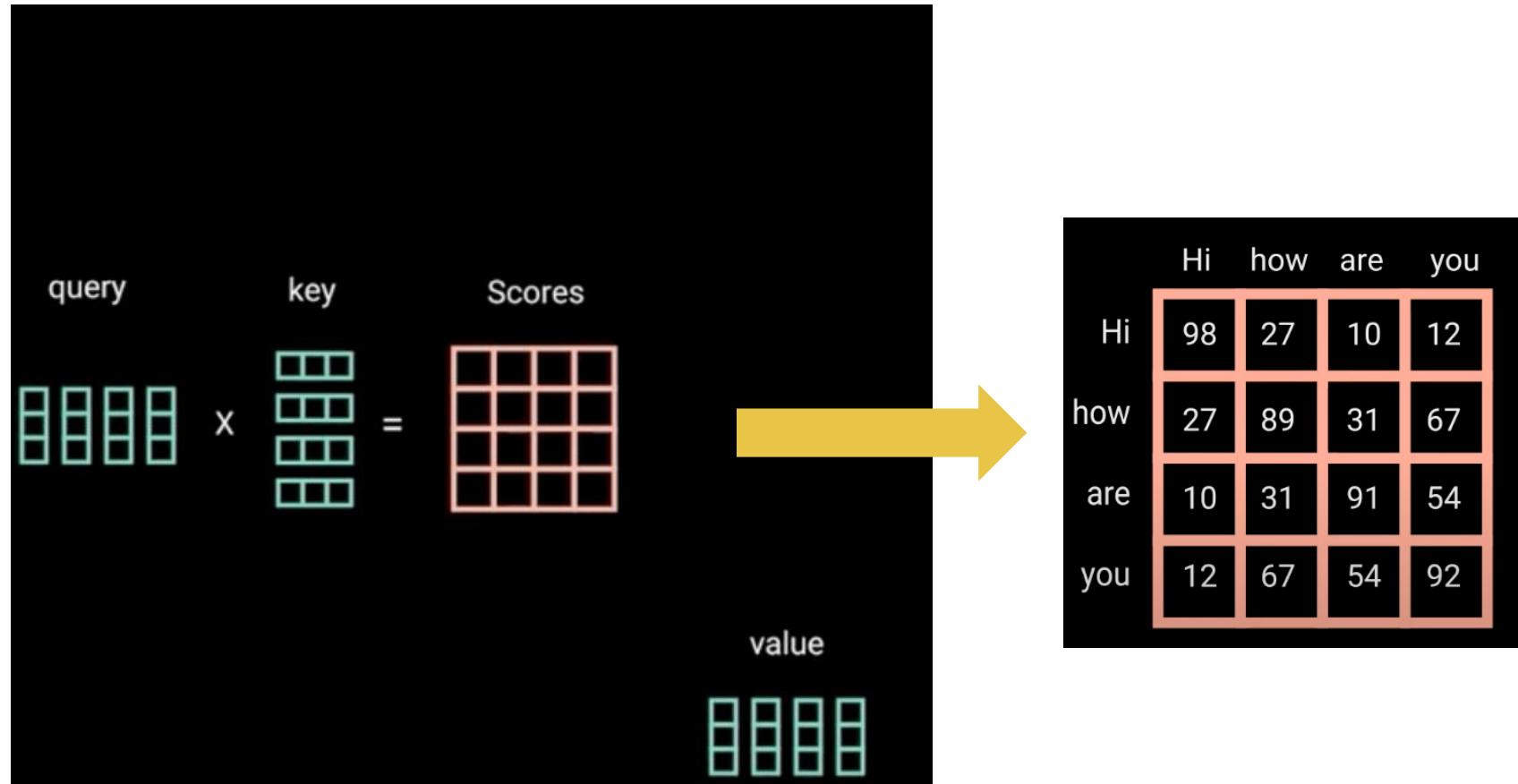
<https://ai.googleblog.com/2017/08/transformer-novel-neural-network.html> 01.02.2022

# Self-Attention



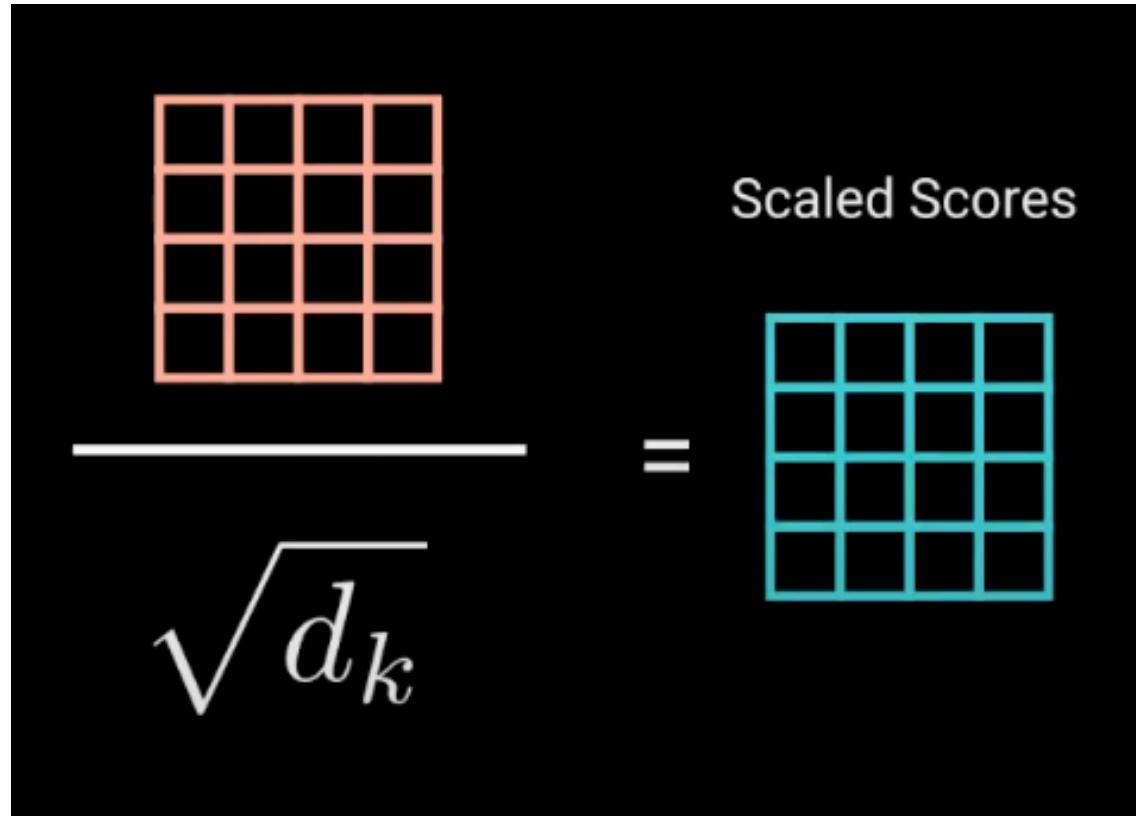
<https://towardsdatascience.com/illustrated-guide-to-transformers-step-by-step-explanation-f74876522bc0>

# Self-Attention



<https://towardsdatascience.com/illustrated-guide-to-transformers-step-by-step-explanation-f74876522bc0>

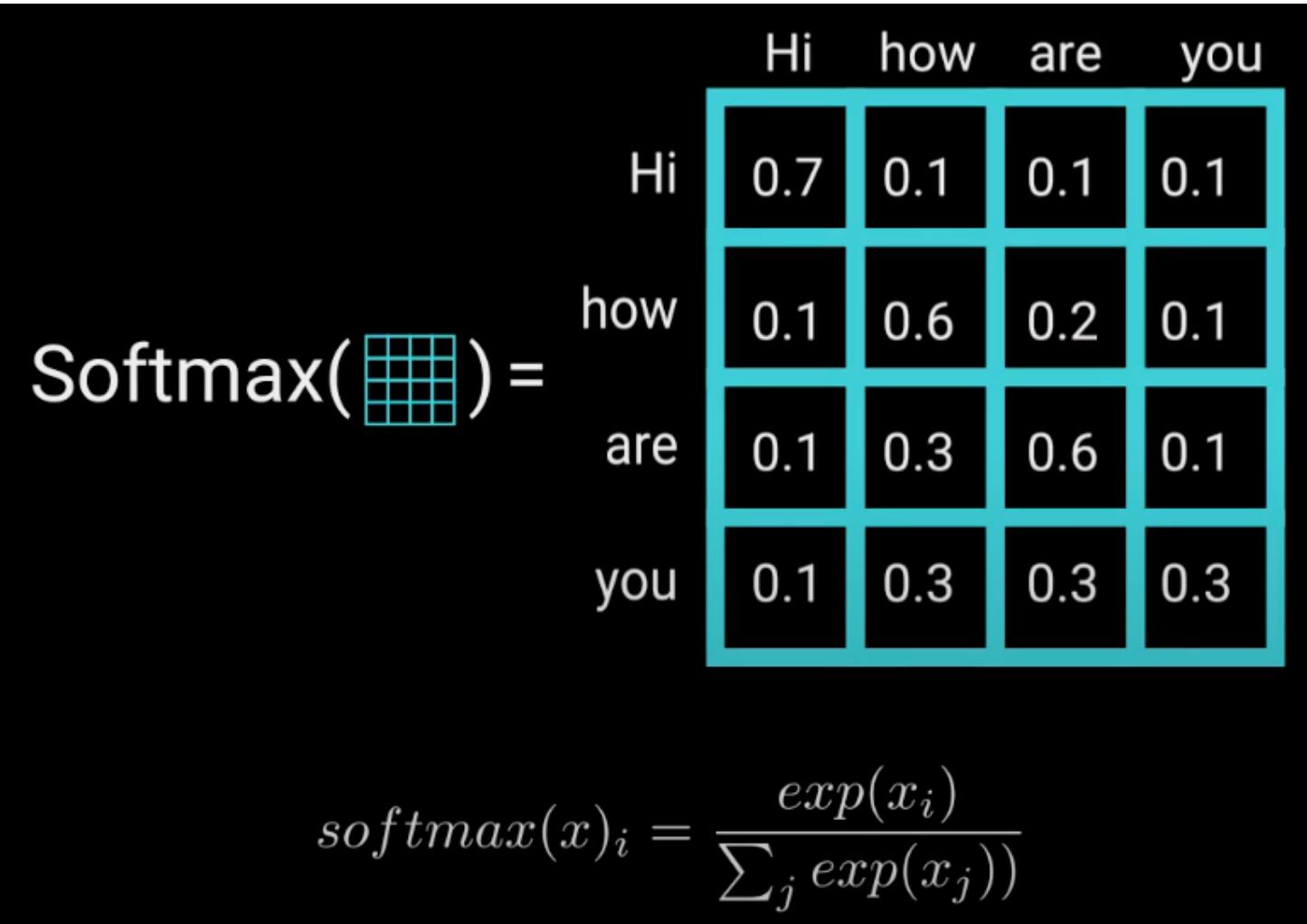
# Self-Attention



- ▶ More stable gradients
- ▶ Prevents exploding effects

<https://towardsdatascience.com/illustrated-guide-to-transformers-step-by-step-explanation-f74876522bc0>

# Self-Attention



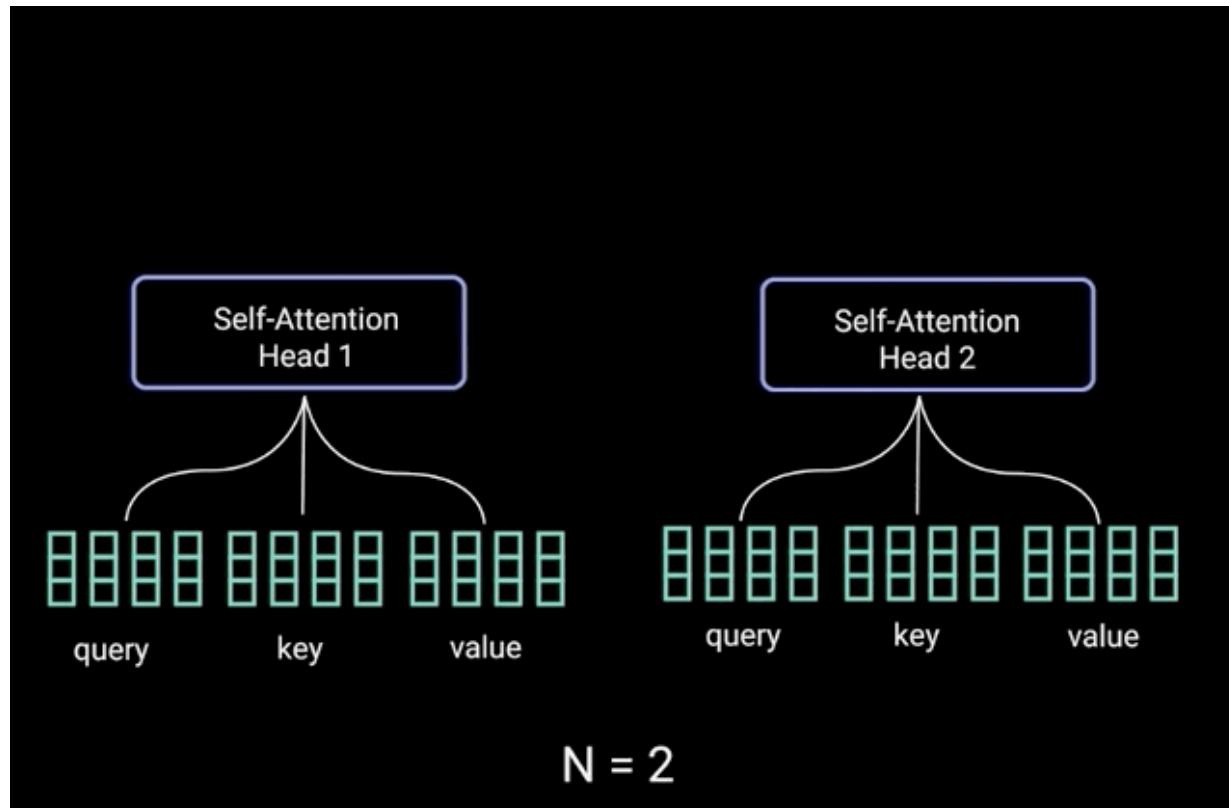
<https://towardsdatascience.com/illustrated-guide-to-transformers-step-by-step-explanation-f74876522bc0>

# Self-Attention



<https://towardsdatascience.com/illustrated-guide-to-transformers-step-by-step-explanation-f74876522bc0>

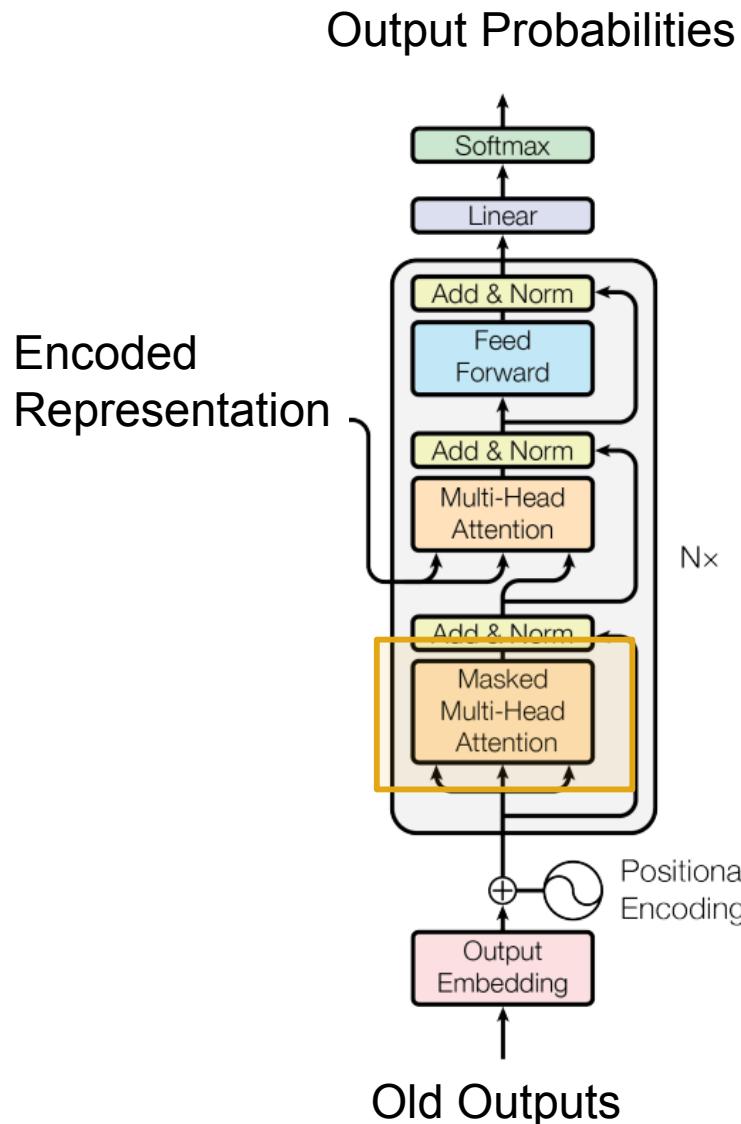
# Multi-Head Attention



- ▶ Calculate Self-Attention multiple times
  - ▶ Each head learns something different
- More representation power

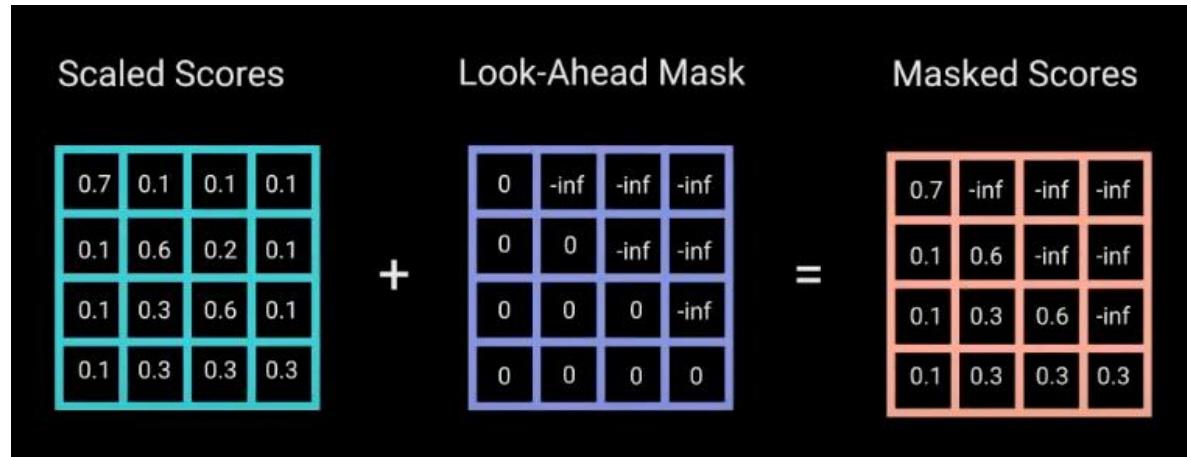
<https://towardsdatascience.com/illustrated-guide-to-transformers-step-by-step-explanation-f74876522bc0>

# Decoder

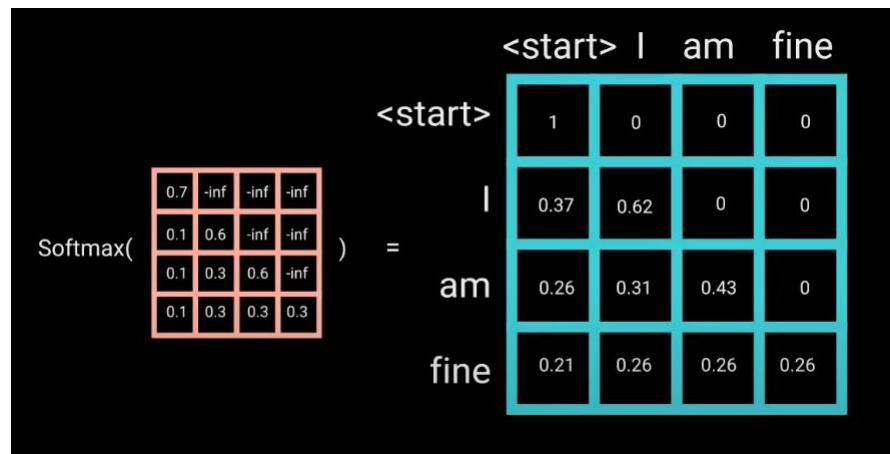


- ▶ Decodes continuous representation
- ▶ Generates text sequences
- ▶ Works autoregressive

# Masked-Multi-Head Attention

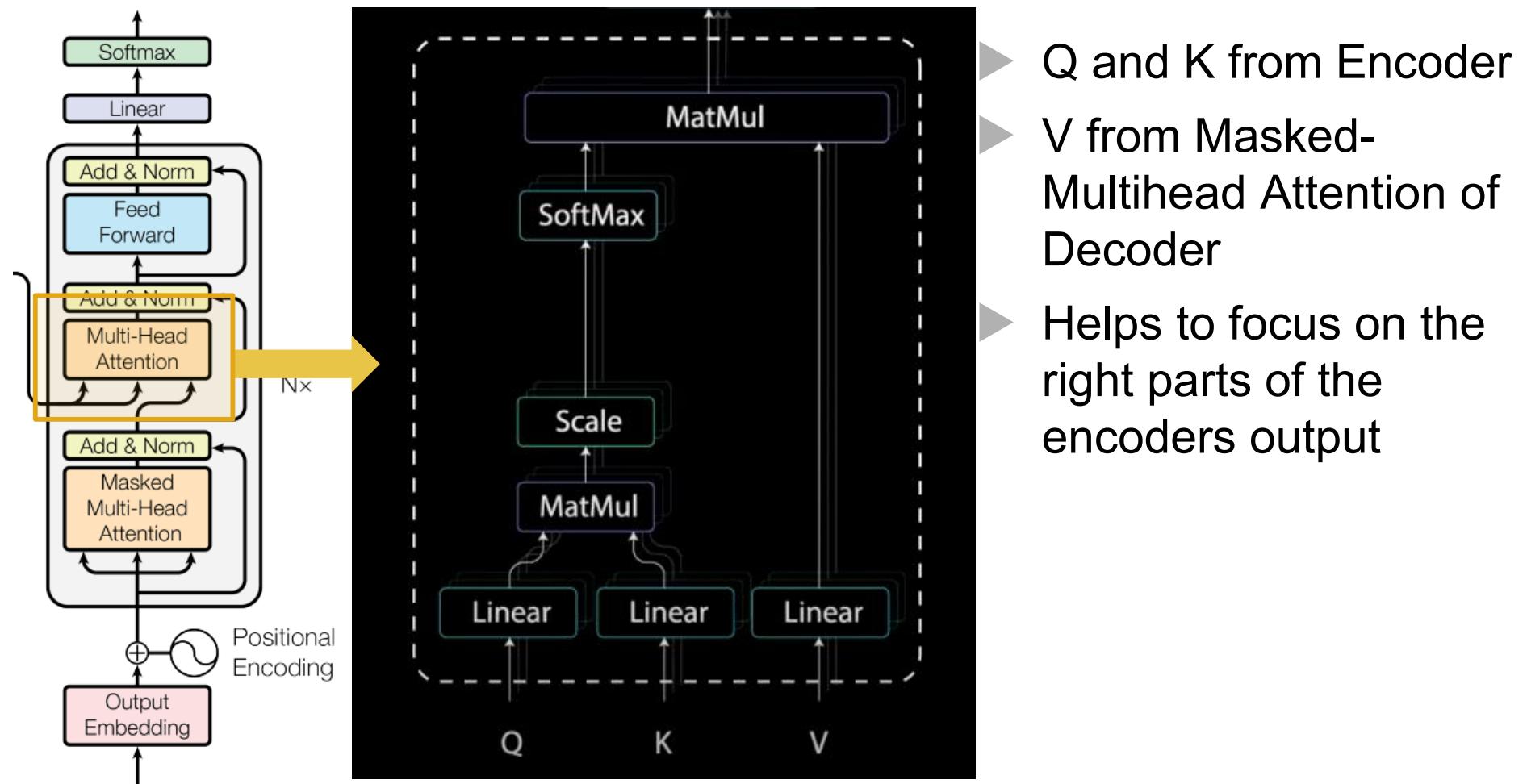


- ▶ Only used for training
- ▶ Each word gets only the attention score for itself, and the words generated before.



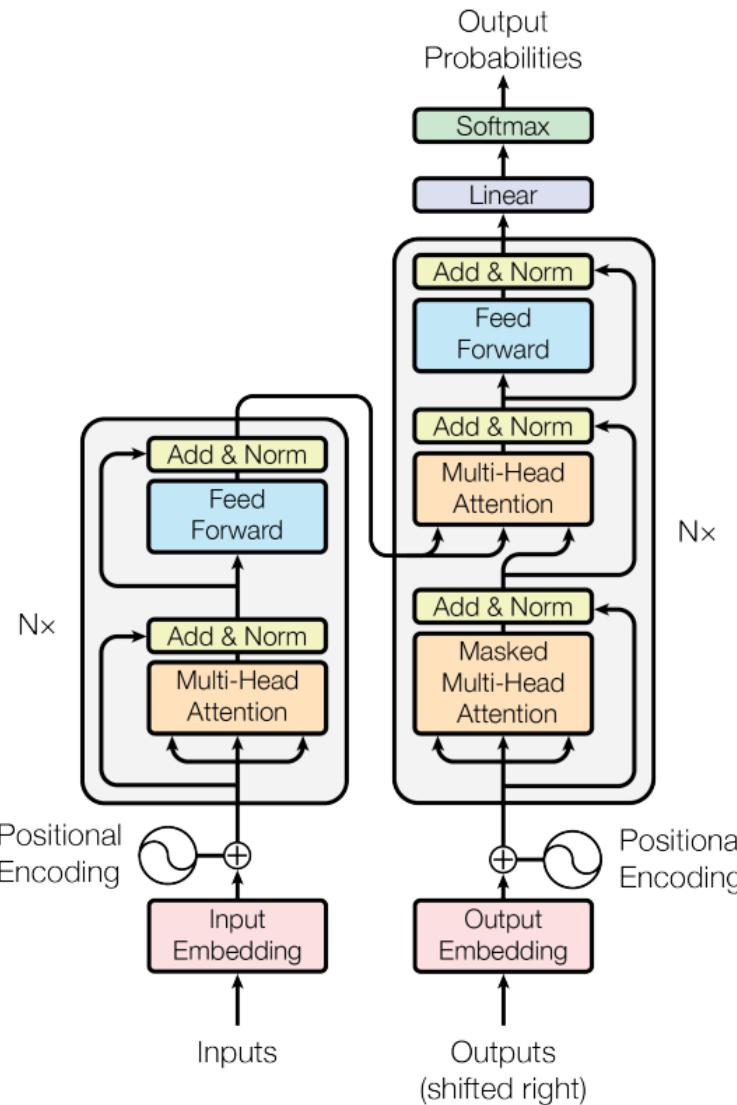
<https://towardsdatascience.com/illustrated-guide-to-transformers-step-by-step-explanation-f74876522bc0>

# Multi-Head Attention Decoder



<https://towardsdatascience.com/illustrated-guide-to-transformers-step-by-step-explanation-f74876522bc0>

# Transformers



## What you know now!

- ▶ Positional Encodings
- ▶ Self-Attention
- ▶ Multi-Head Attention
- ▶ Masked Multi-Head Attention

# Hands-on part: Translation with transformers

- ▶ Please work through the tutorial  
**12\_Transformers\_Translation\_Example.ipynb**



# Autonomous Systems: Deep Learning

## Deep Reinforcement Learning Introduction

# Outline

1. Deep Q-Learning (DQN)
2. Task 1: 2D Pole Cart with DQN
3. DQN Improvements
4. Policy Gradient
5. Actor Critic Methods
6. Task 2: Deep Deterministic Policy Gradient (DDPG)
7. Advanced Exploration
8. Further Readings & State of the Art



# 1. Deep Q-Learning (DQN)

# From Tabular Methods to Deep Methods

250 states



<https://spiele.rtl.de/kartenspiele/black-jack.html>

$10^{70802}$  states



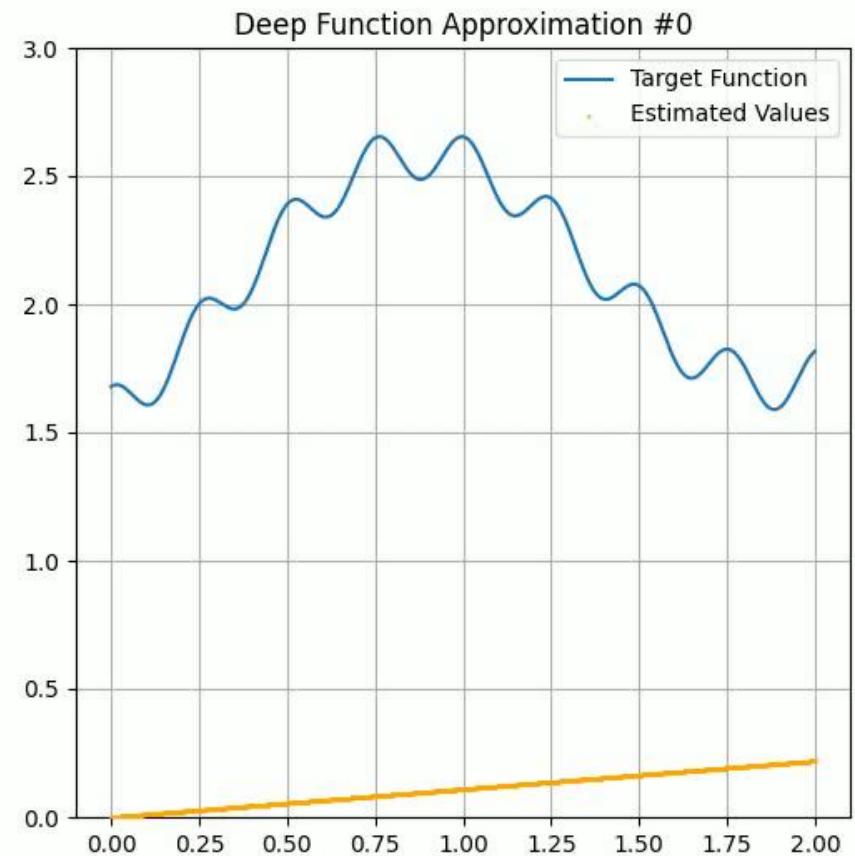
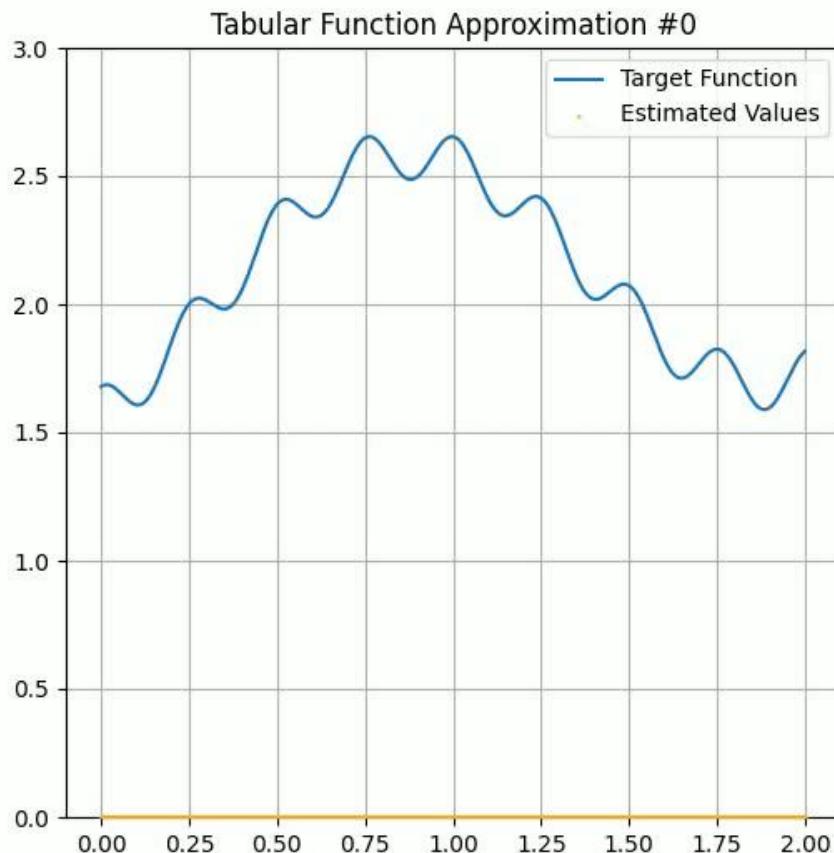
[https://www.retrogames.cz/play\\_222-Atari2600.php](https://www.retrogames.cz/play_222-Atari2600.php)

$7.7 * 10^{45}$  states



<https://i.ytimg.com/vi/VHqCAaFXpbc/maxresdefault.jpg>

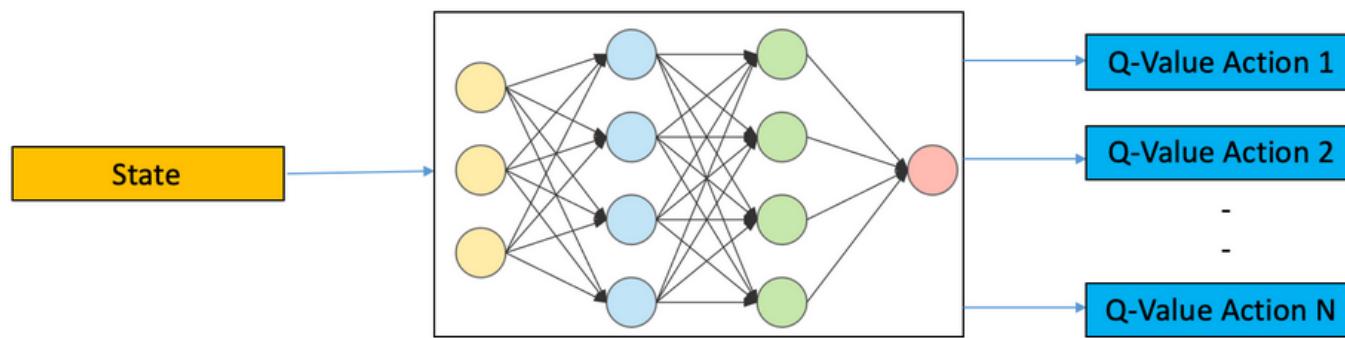
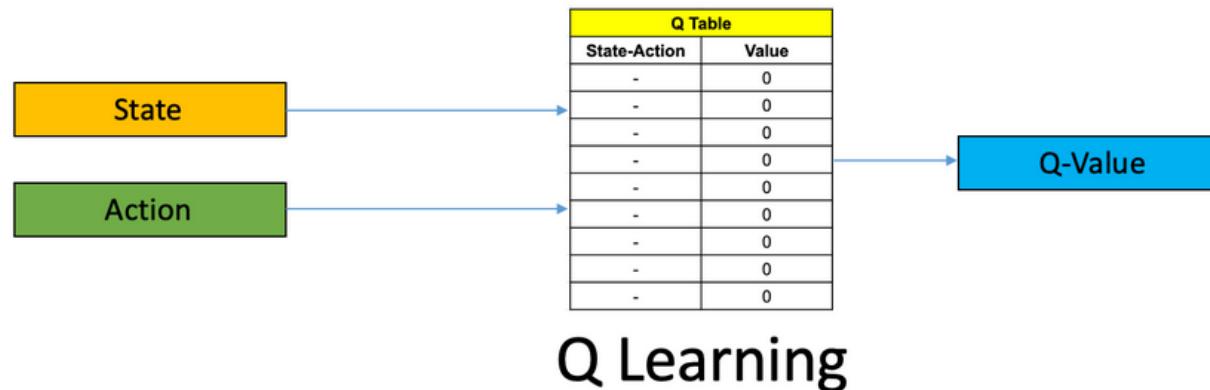
# From Tabular Methods to Deep Methods



# From Tabular Methods to Deep Methods

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[ R_{t+1} + \gamma \max_{a' \in A} Q(S_{t+1}, a') - Q(S_t, A_t) \right] \quad (1)$$

$$y = R_{t+1} + \gamma \max_{a' \in A} Q(S_{t+1}, a') \quad L = (Q(S_t, A_t) - y)^2 \quad (2)$$



**Deep Q Learning**

<https://www.analyticsvidhya.com/blog/2019/04/introduction-deep-q-learning-python/>

# Naive Algorithm

## Naïve DQN Algorithm

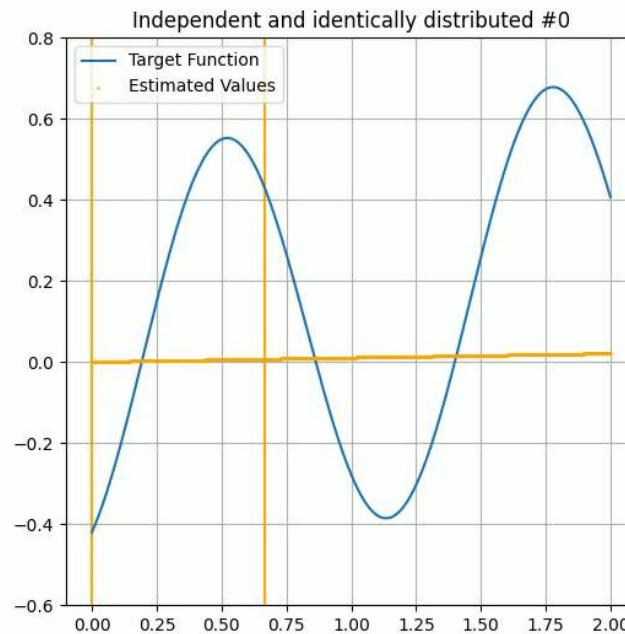
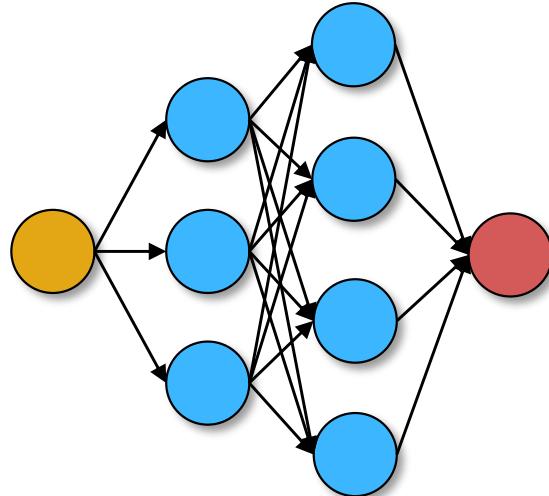
1. Initialize a random Neural Network  $Q(s, a)$ .
2. By interacting with the environment, obtain the tuple  $(s, a, r, s')$ .
3. Calculate Loss:  $L = (Q(s, a) - r)^2$  if the episode has ended, or  
$$L = (Q(s, a) - \left( r + \gamma \max_{a' \in A} Q(s', a') \right)^2 \text{ otherwise.}$$
4. Update  $Q(s, a)$  using stochastic gradient descent (SGD).
5. Repeat from step 2 until converged.

<https://www.packtpub.com/data/deep-reinforcement-learning-hands-on-second-edition>

# Naive Algorithm

## PROBLEMS

- Exploration vs. Exploitation Dilemma:  
→ Epsilon-Greedy Algorithm
- Markov Property (partially observable MDPs)  
→ State Stack
- SGD optimization:  
*Training data needs to be independent and identically distributed.*  
→ Replay Buffer



# Naive Algorithm

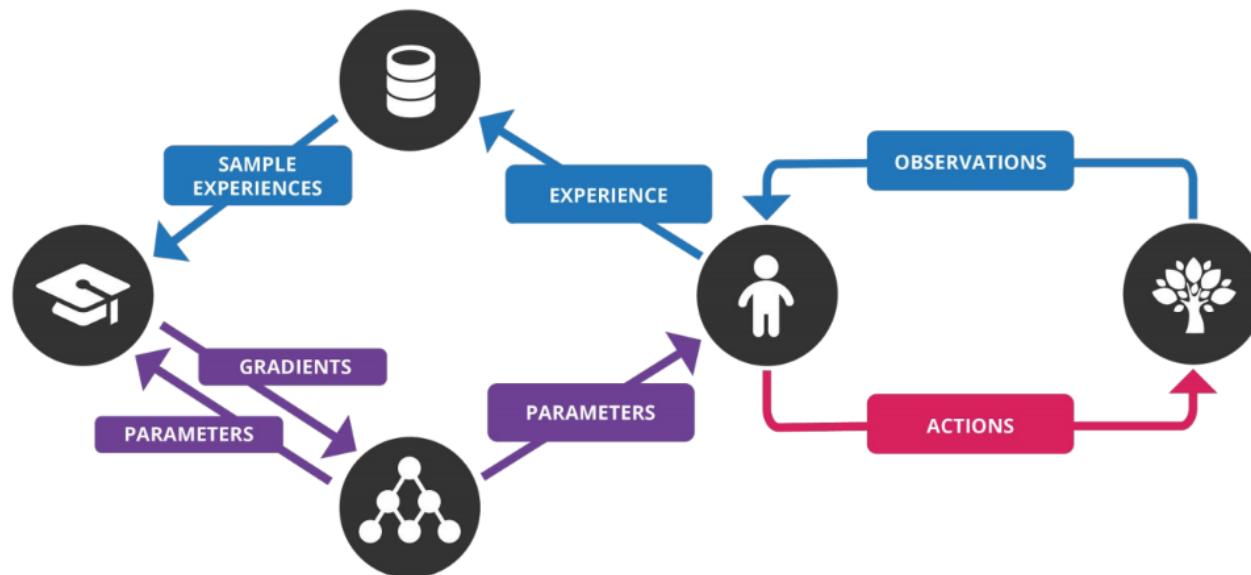
## PROBLEMS

- Exploration vs. Exploitation Dilemma:  
→ Epsilon-Greedy Algorithm
- Markov Property (**partially observable MDPs**)  
→ State Stack
- SGD optimization:  
*Training data needs to be independent and identically distributed.*  
→ Replay Buffer
- Correlation between steps:  
*We're training  $Q(s, a)$  via  $Q(s', a')$  (bootstrapping). When we perform an update of our NN's parameters to make  $Q(s, a)$ , we can indirectly alter the value produced for  $Q(s', a')$  and other states nearby.*  
→ Target Network

# Final Algorithm

$$y_i = r_i + \gamma \max_{a' \in A} \hat{Q}(s'_i, a')$$

$$L = \frac{1}{N} \sum_i (Q(s_i, a_i) - y_i)^2 \quad (3)$$



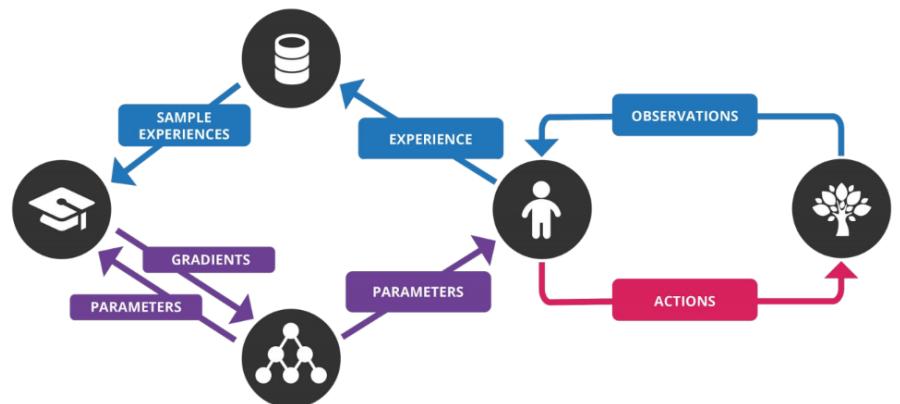
<https://www.analyticsvidhya.com/blog/2019/04/introduction-deep-q-learning-python/>

# Final Algorithm

## Final DQN Algorithm

1. Initialize two Neural Networks  $Q(s, a)$  &  $\hat{Q}(s, a)$ , an empty replay buffer and  $\epsilon \leftarrow 1.0$ .
2. Act according to an  $\epsilon - \text{greedy}$  policy, observe reward  $r$  and the next state  $s'$ .
3. Store transition  $(s, a, r, s')$  in the replay buffer.
4. Sample a random batch of transitions from the buffer. Calculate the target  $y_i = r_i$  if the episode has ended,  $y_i = r_i + \gamma \max_{a' \in A} \hat{Q}(s'_i, a')$  otherwise.
5. Calculate loss  $L = \frac{1}{N} \sum_i (Q(s_i, a_i) - y_i)^2$  and update  $Q(s, a)$  using SGD by minimizing the loss with respect to the model parameters.
6. Every  $N$  steps copy the weights from  $Q$  to  $\hat{Q}$ .
7. Repeat from step 2 until convergence.

<https://www.packtpub.com/data/deep-reinforcement-learning-hands-on-second-edition>



<https://www.analyticsvidhya.com/blog/2019/04/introduction-deep-q-learning-python/>

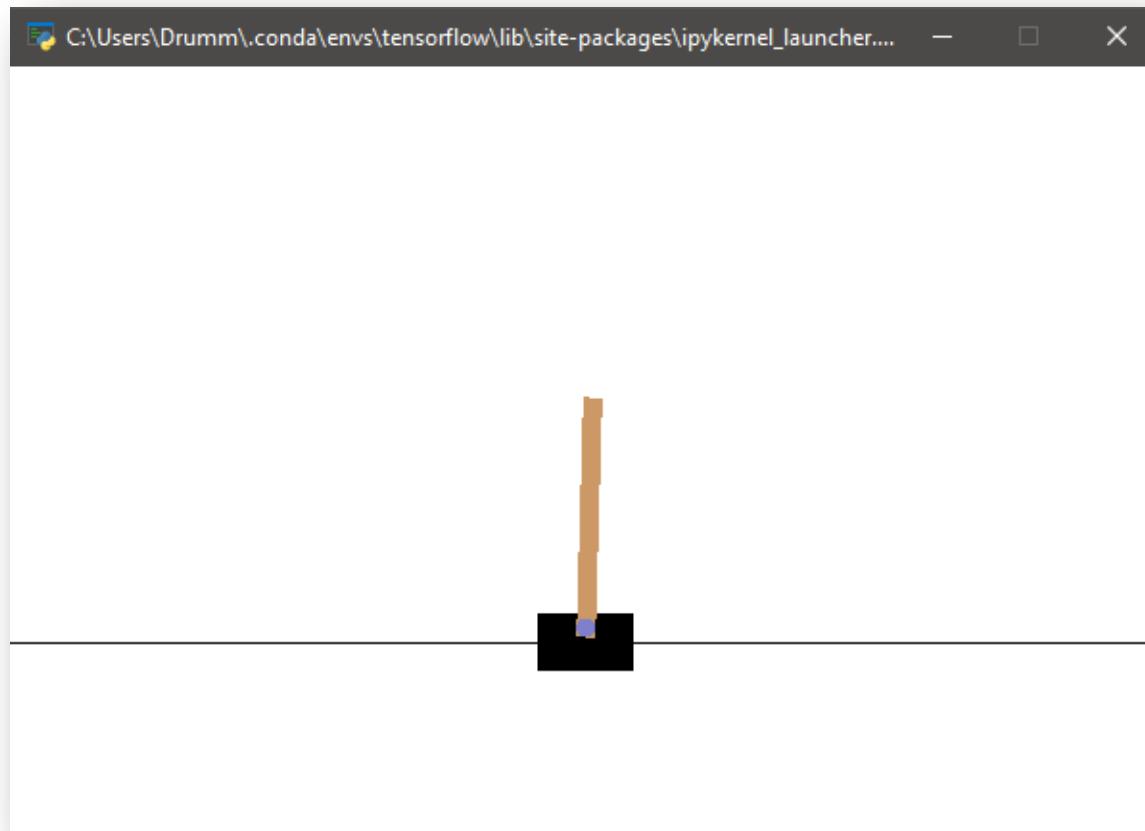


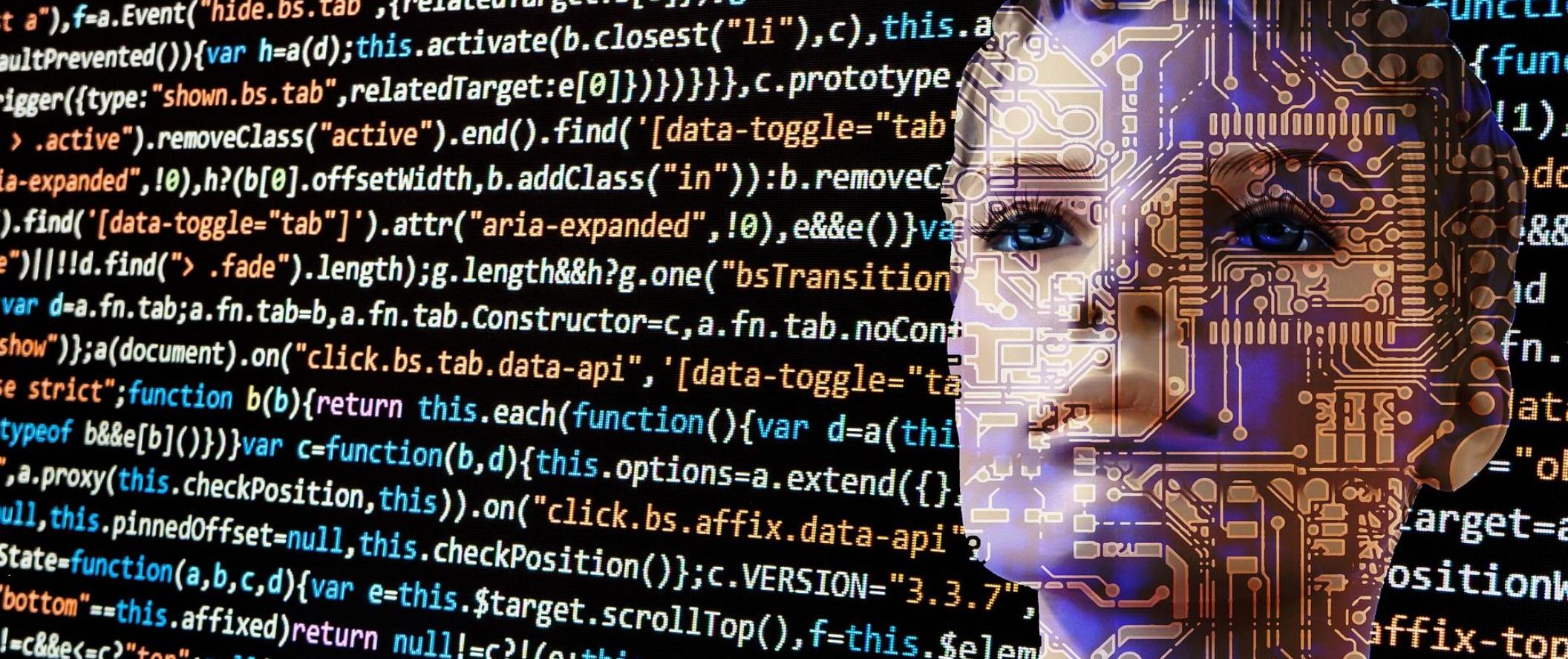
## 2. Task 1: 2D Pole Cart with DQN

# 2D Pole Cart

**Task:**

*Implement a Deep Q-Learning Algorithm (DQN) to balance a pole on a cart in two dimensions by moving the cart left and right. (Additionally, try to implement some of the improvement techniques discussed before and compare training performance.)*





### 3. DQN Improvements

# Improvements

## PRIORITIZED EXPERIENCE REPLAY

**Idea:** Prioritize replay transitions based on their estimated potential for improving the agent's policy.

**Approach:**

- Sample from the replay buffer with probabilities according to sample priorities

$$P(i) = \frac{p_i^\alpha}{\sum_j p_j^\alpha} \text{ with } \alpha \in [0, 1] \quad (4)$$

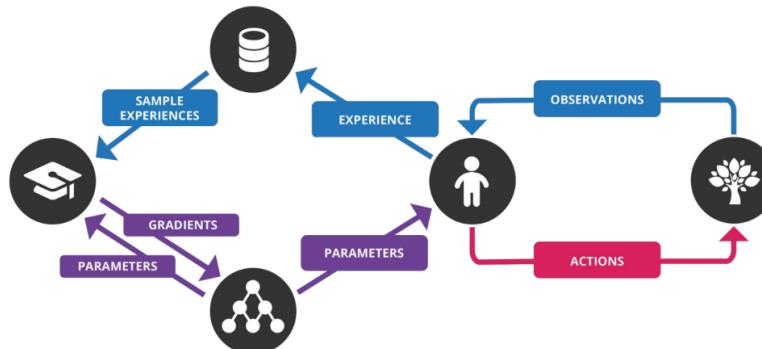
- When training on a transition update its priority with the TD-error

$$\delta_i = R_i + \gamma \hat{Q}(s_i, \text{argmax}_{a'} Q(s'_i, a')) - Q(s_i, a_i) \quad (5)$$

$$p_i = |\delta_i| + \epsilon \quad (6)$$

- New collected samples start off with maximum priority.

<https://arxiv.org/abs/1511.05952>



<https://www.analyticsvidhya.com/blog/2019/04/introduction-deep-q-learning-python/>

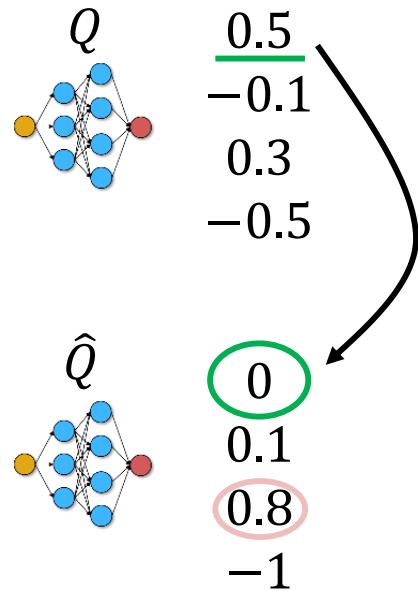
# Improvements

## MAXIMIZATION BIAS AND DOUBLE LEARNING

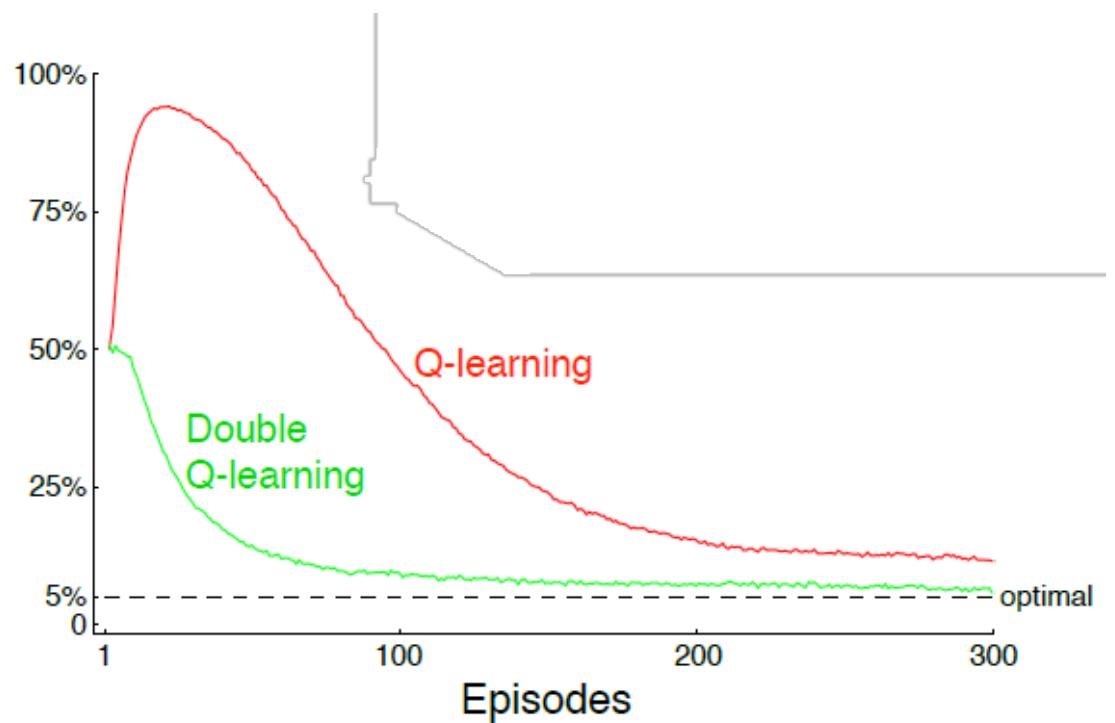
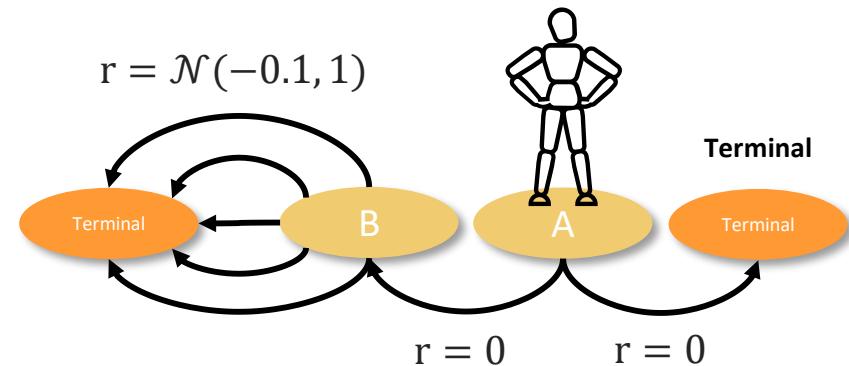
$$\text{Target} \leftarrow R_{t+1} + \gamma \max_{a' \in A} \hat{Q}(S_{t+1}, a') \quad (7)$$

$$\text{Target} \leftarrow R_{t+1} + \gamma \hat{Q}\left(S_{t+1}, \operatorname{argmax}_{a'} Q(S_{t+1}, a')\right) \quad (8)$$

<https://arxiv.org/abs/1509.06461>



% left  
actions  
from A



<http://www.andrew.cmu.edu/course/10-703/textbook/BartoSutton.pdf>



## 4. Policy Gradient

# Policy Gradient

## VALUE VS. POLICY GRADIENT

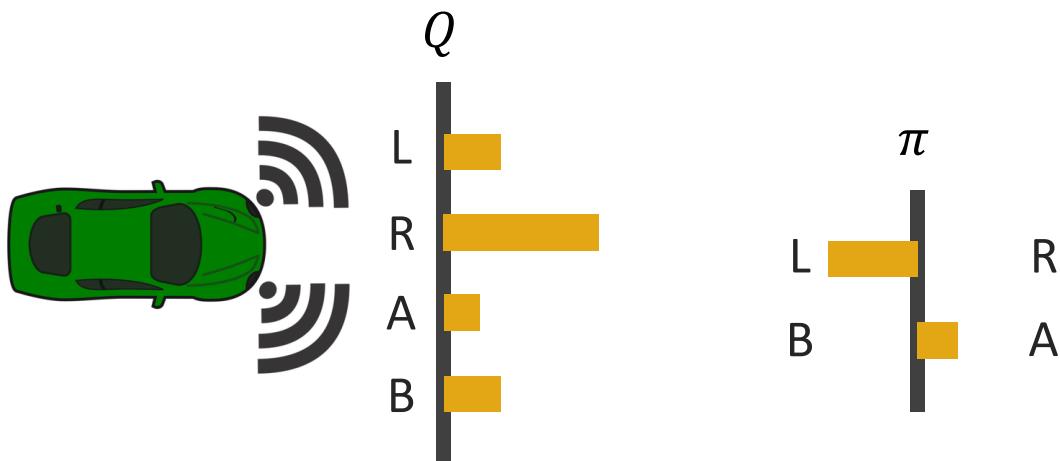
### Previous Algorithms:

- Estimate **value**  $V(s)$  or  $Q(s, a)$
- Take the action with the highest estimated value in every state

$$\pi(s) = \operatorname{argmax}_a Q(s, a) \quad (9)$$

### Why Policy?

1. Environments with lots of actions or a **continuous action space**



<https://www.packtpub.com/data/deep-reinforcement-learning-hands-on-second-edition>

# Policy Gradient

## VALUE VS. POLICY GRADIENT

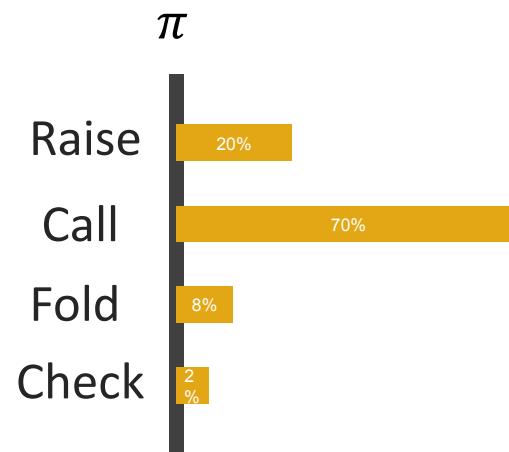
### Previous Algorithms:

- Estimate **value**  $V(s)$  or  $Q(s, a)$
- Take the action with the highest estimated value in every state

$$\pi(s) = \operatorname{argmax}_a Q(s, a) \quad (9)$$

### Why Policy?

1. Environments with lots of actions or a **continuous action space**
2. Environments with **stochasticity** in them



<https://www.packtpub.com/data/deep-reinforcement-learning-hands-on-second-edition>

# Policy Gradient

## VALUE VS. POLICY GRADIENT

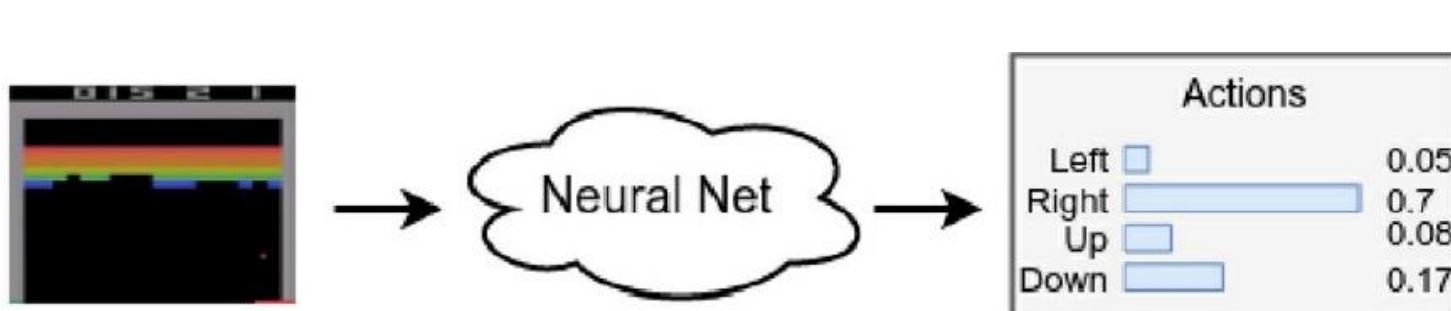
### Previous Algorithms:

- Estimate **value**  $V(s)$  or  $Q(s, a)$
- Take the action with the highest estimated value in every state

$$\pi(s) = \operatorname{argmax}_a Q(s, a) \quad (9)$$

### Why Policy?

1. Environments with lots of actions or a **continuous action space**
2. Environments with **stochasticity** in them
3. Enables smooth representation



<https://www.packtpub.com/data/deep-reinforcement-learning-hands-on-second-edition>

# Policy Gradient

## POLICY GRADIENT

### Key Idea

Push up the probabilities of actions that lead to higher return and push down the probabilities of actions that lead to lower return, until you arrive at the optimal policy.

**Policy performance:**

$$J(\pi_\theta) = \mathbb{E}_{\tau \sim \pi_\theta} [G(\tau)] \quad (10)$$

**Policy Gradient Loss:**

$$L = -\frac{1}{|D|} \sum_{\tau \in D} \sum_{t=0}^T \log \pi_\theta(a_t | s_t) G(\tau) \quad (11)$$

**Weight update for gradient ascent:**

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}_{\tau \sim \pi_\theta} \left[ \sum_{t=0}^T \nabla_\theta \log \pi_\theta(a_t | s_t) G(\tau) \right] \approx \frac{1}{|D|} \sum_{\tau \in D} \sum_{t=0}^T \nabla_\theta \log \pi_\theta(a_t | s_t) G(\tau) \quad (12)$$

[https://medium.com/@jonathan\\_hui/rl-policy-gradients-explained-9b13b688b146](https://medium.com/@jonathan_hui/rl-policy-gradients-explained-9b13b688b146)

[https://spinningup.openai.com/en/latest/spinningup/rl\\_intro3.html](https://spinningup.openai.com/en/latest/spinningup/rl_intro3.html)

<https://www.packtpub.com/data/deep-reinforcement-learning-hands-on-second-edition>

# Reinforce Algorithm

## Pseudocode

1. Initialize the network with random weights.
2. Play  $D$  full episodes, saving their  $(s, a, r, s')$  transitions.
3. For every step  $t$  of every trajectory  $\tau$ , calculate the discounted total reward for subsequent steps  $G_{\tau,t} = \sum_{i=0}^T \gamma^i r_i$ .
4. Calculate the loss function for all transitions.

$$L = -\frac{1}{|D|} \sum_{\tau \in D} \sum_{t=0}^T \log \pi_\theta(a_t | s_t) G(\tau) \quad (11)$$

5. Perform SGD update of weights minimizing the loss.
6. Repeat from step 2 until converged.

[https://medium.com/@jonathan\\_hui/rl-policy-gradients-explained-9b13b688b146](https://medium.com/@jonathan_hui/rl-policy-gradients-explained-9b13b688b146)

[https://spinningup.openai.com/en/latest/spinningup/rl\\_intro3.html](https://spinningup.openai.com/en/latest/spinningup/rl_intro3.html)

<https://www.packtpub.com/data/deep-reinforcement-learning-hands-on-second-edition>



## 5. Actor Critic Methods

# Policy Gradient Shortcomings

## Shortcomings of Policy Gradient:

- Whole trajectories needed

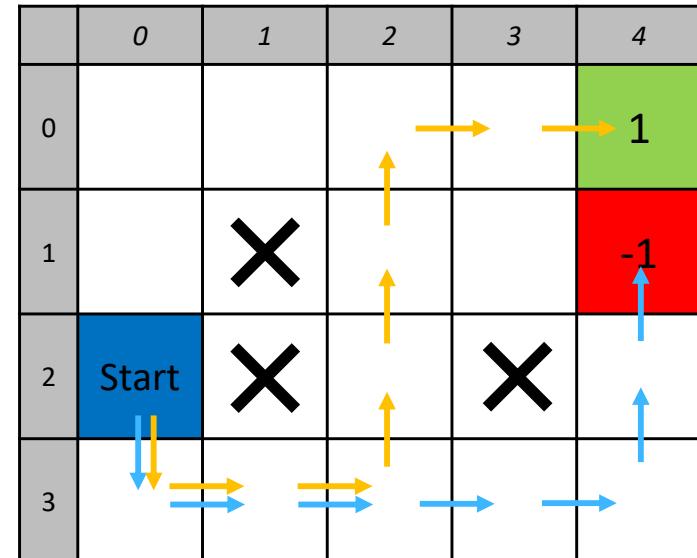
$$L = -\frac{1}{|D|} \sum_{\tau \in D} \sum_{t=0}^T \log \pi_{\theta}(a_t | s_t) G(\tau) \quad (11)$$

- On-Policy
- High variability in log probabilities and cumulative rewards:

→ High variance gradients

- Trajectories with cumulative reward of zero:

→ Zero gradients



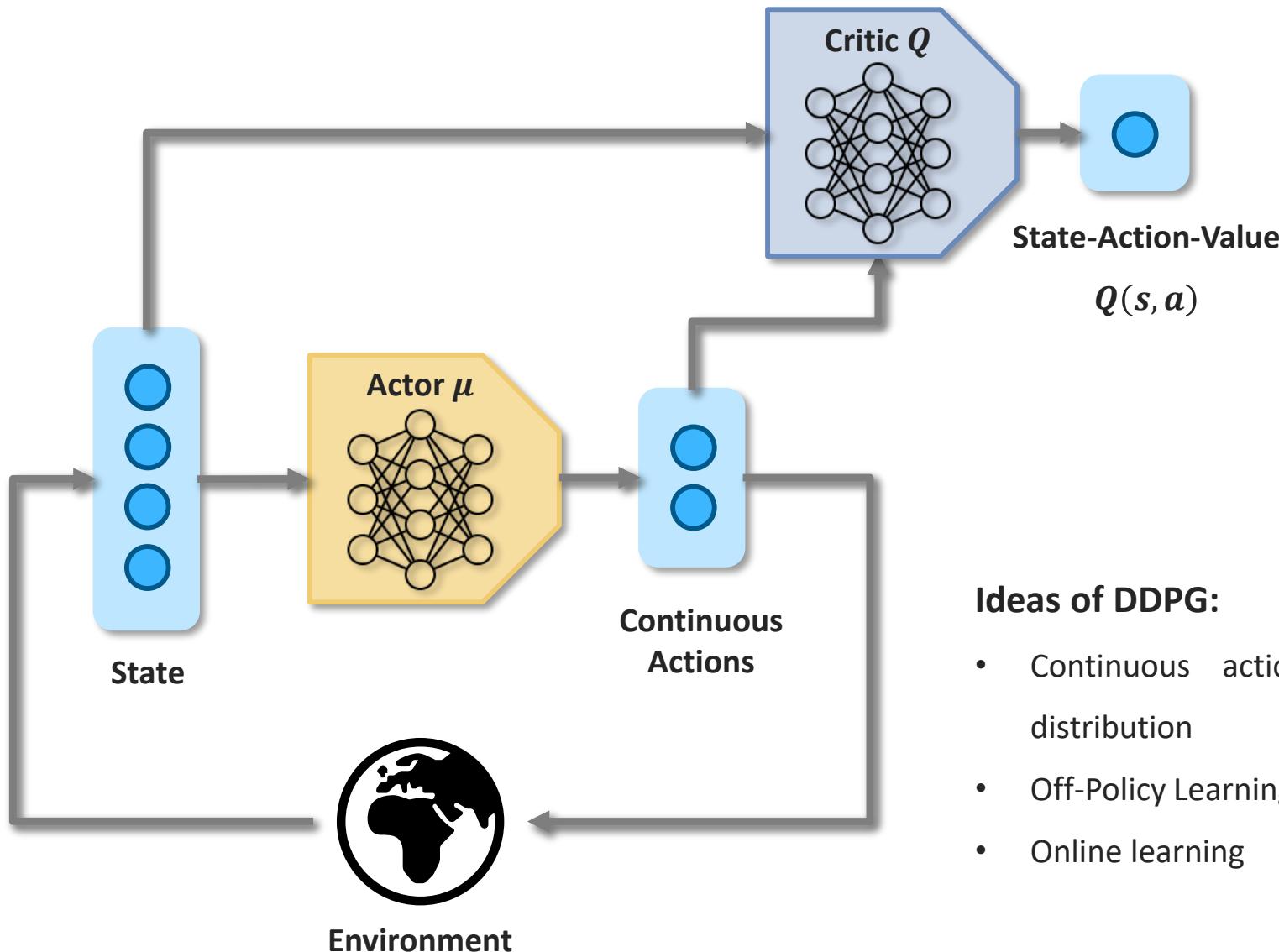
$$R_t = -0.1$$

on all other transitions

<https://towardsdatascience.com/understanding-actor-critic-methods-931b97b6df3f>

[https://medium.com/@jonathan\\_hui/rl-policy-gradients-explained-9b13b688b146](https://medium.com/@jonathan_hui/rl-policy-gradients-explained-9b13b688b146)

# Deep Deterministic Policy Gradient (DDPG)



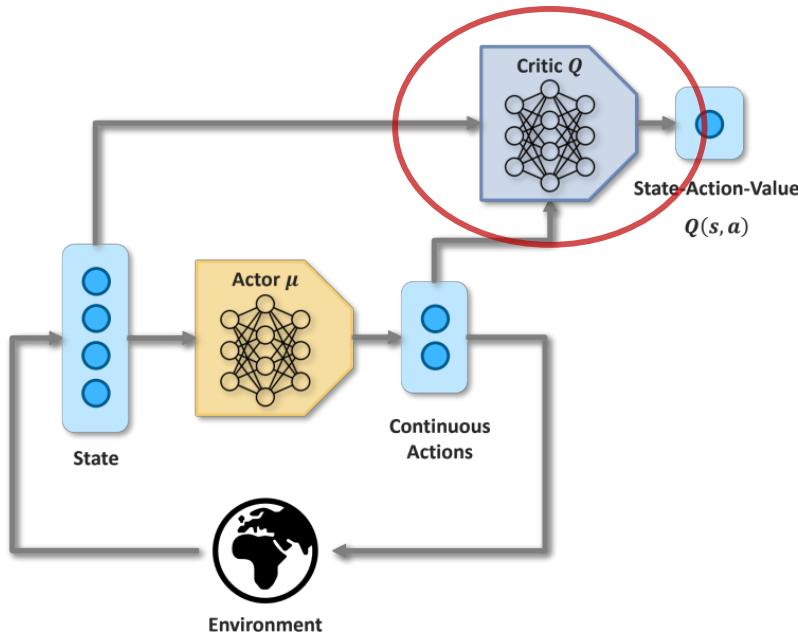
## Ideas of DDPG:

- Continuous actions instead of probability distribution
- Off-Policy Learning (→ Replay Buffer)
- Online learning

<https://towardsdatascience.com/deep-deterministic-and-twin-delayed-deep-deterministic-policy-gradient-with-tensorflow-2-x-43517b0e0185>

<https://towardsdatascience.com/deep-deterministic-policy-gradients-explained-2d94655a9b7b>

# Deep Deterministic Policy Gradient (DDPG)



**Q Learning Target & Loss:**

$$y_i = r_i + \gamma \max_{a'} \hat{Q}(s'_i, a') \quad L = \frac{1}{N} \sum_i (Q(s_i, a_i) - y_i)^2 \quad (3)$$

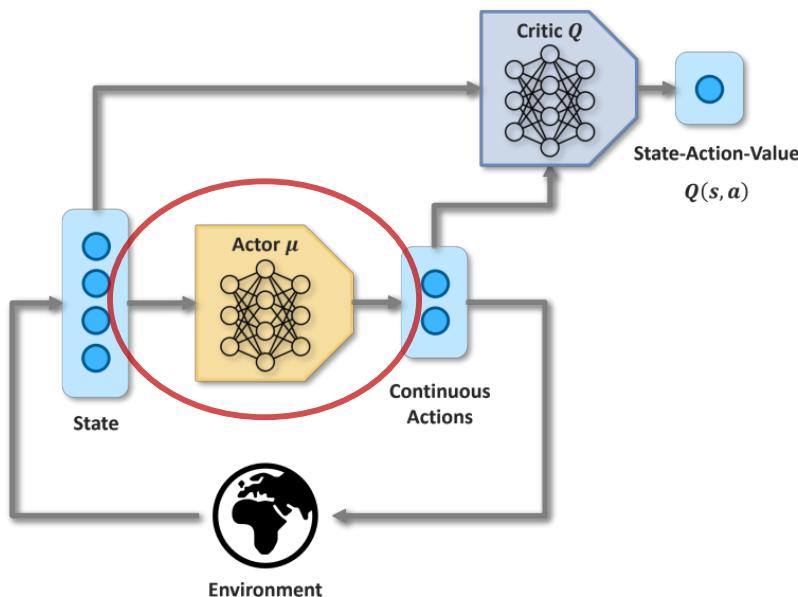
**DDPG Critic Target & Loss:**

$$y_i = r_i + \gamma \hat{Q}(s'_i, \hat{\mu}(s'_i)) \quad L_Q = \frac{1}{N} \sum_i (Q(s_i, a_i) - y_i)^2 \quad (13)$$

<https://towardsdatascience.com/deep-deterministic-and-twin-delayed-deep-deterministic-policy-gradient-with-tensorflow-2-x-43517b0e0185>

<https://towardsdatascience.com/deep-deterministic-policy-gradients-explained-2d94655a9b7b>

# Deep Deterministic Policy Gradient (DDPG)



**Policy Gradient Loss:**

$$L = -\frac{1}{|D|} \sum_{\tau \in D} \sum_{t=0}^T \log \pi_\theta(a_t | s_t) G(\tau) \quad (11)$$

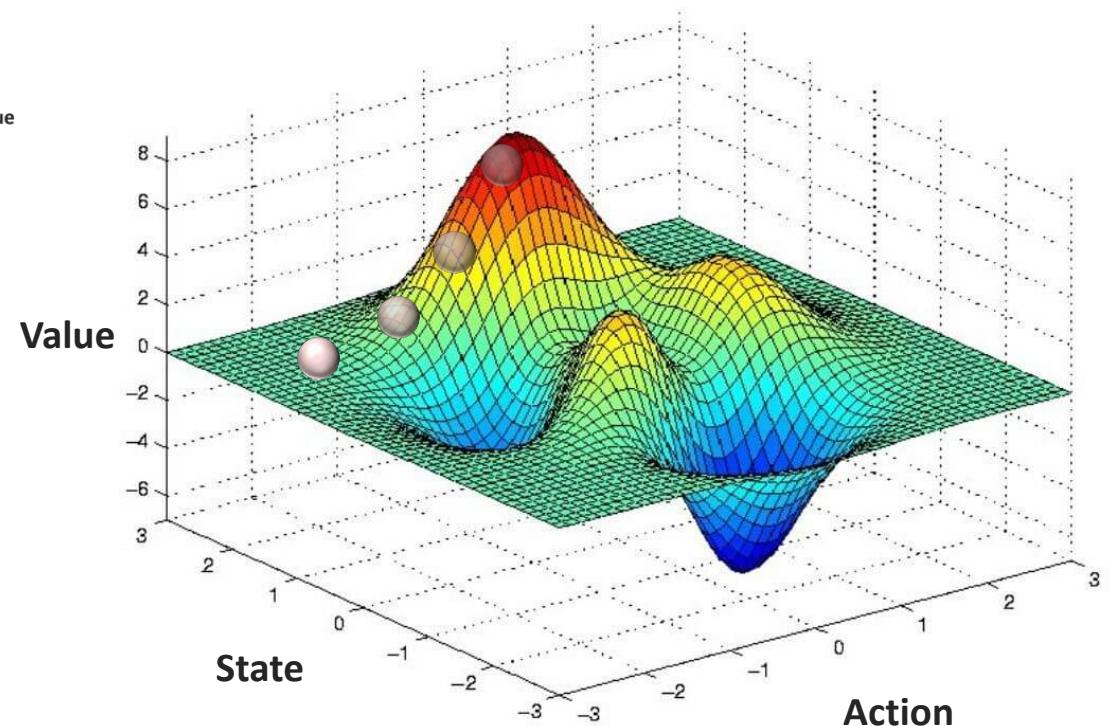
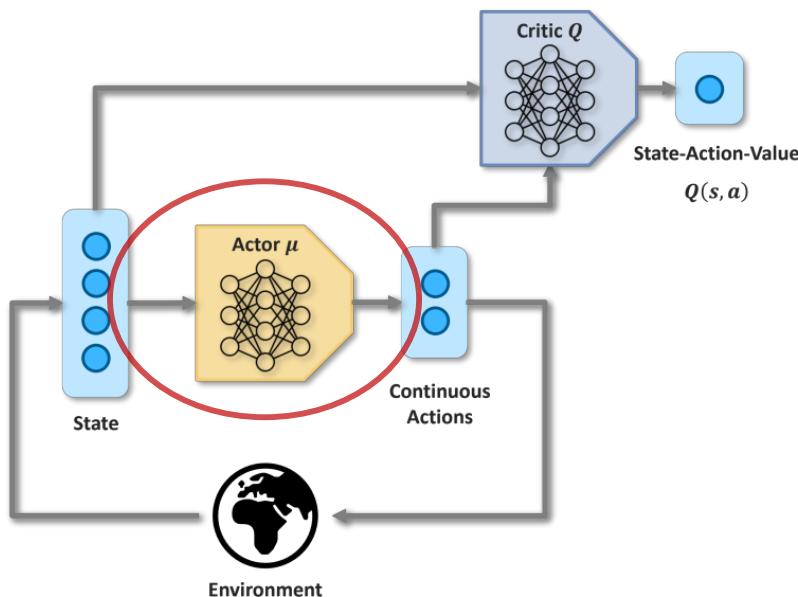
**DDPG Actor Loss:**

$$L_\mu = -\frac{1}{N} \sum_i Q(s_i, \mu(s_i)) \quad (14)$$

<https://towardsdatascience.com/deep-deterministic-and-twin-delayed-deep-deterministic-policy-gradient-with-tensorflow-2-x-43517b0e0185>

<https://towardsdatascience.com/deep-deterministic-policy-gradients-explained-2d94655a9b7b>

# Deep Deterministic Policy Gradient (DDPG)



<https://laconicml.com/stochastic-gradient-descent-in-python/>

## DDPG Actor Loss:

$$L_\mu = -\frac{1}{N} \sum_i Q(s_i, \mu(s_i)) \quad (14)$$

<https://towardsdatascience.com/deep-deterministic-and-twin-delayed-deep-deterministic-policy-gradient-with-tensorflow-2-x-43517b0e0185>

<https://towardsdatascience.com/deep-deterministic-policy-gradients-explained-2d94655a9b7b>

# Deep Deterministic Policy Gradient (DDPG)

## DDPG Algorithm

1. Initialize two Critic Networks  $Q(s, a)$  &  $\hat{Q}(s, a)$ , two Actor Networks  $\mu$  &  $\hat{\mu}$  and an empty replay buffer.
2. Select action  $a_t = \mu(s) + \mathcal{N}_t$ , observe reward  $r$  and the next state  $s'$ .
3. Store transition  $(s, a, r, s')$  in the replay buffer.
4. Sample a random batch of transitions from the buffer. Calculate the target  $y_i = r_i$  if the episode has ended,  $y_i = r_i + \gamma \hat{Q}(s'_i, \hat{\mu}(s'_i))$  otherwise.
5. Update the critic by minimizing loss:  $L_Q = \frac{1}{N} \sum_i (Q(s_i, a_i) - y_i)^2$
6. Update the actor by minimizing loss:  $L_\mu = -\frac{1}{N} \sum_i Q(s_i, \mu(s_i))$
7. Update target networks:  

$$\hat{Q}_\omega \leftarrow \tau Q_\omega + (1 - \tau) * \hat{Q}_\omega$$

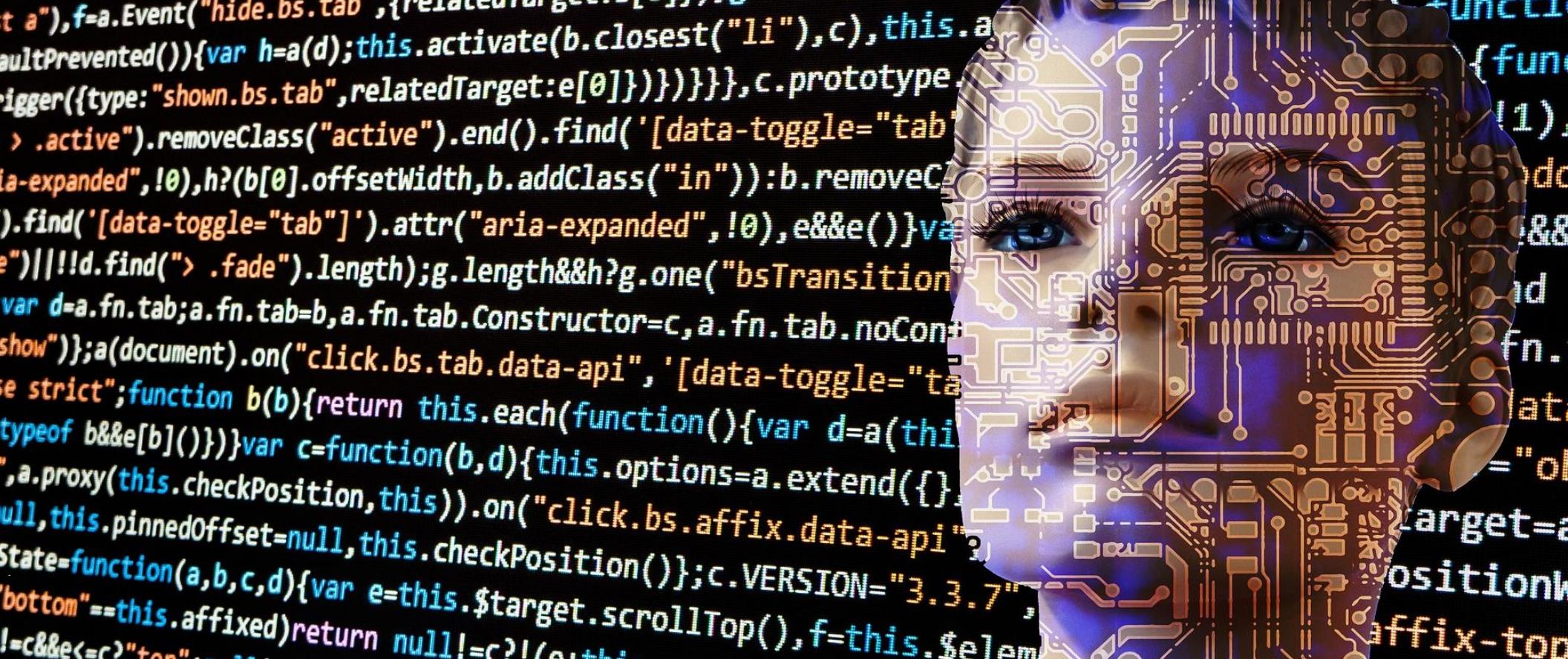
$$\hat{\mu}_\theta \leftarrow \tau \mu_\theta + (1 - \tau) * \hat{\mu}_\theta$$
8. Repeat from step 2 until convergence.

<https://towardsdatascience.com/deep-deterministic-and-twin-delayed-deep-deterministic-policy-gradient-with-tensorflow-2-x-43517b0e0185>

<https://towardsdatascience.com/deep-deterministic-policy-gradients-explained-2d94655a9b7b>



## 6. Task 2: Deep Deterministic Policy Gradient (DDPG)



## 7. Advanced Exploration

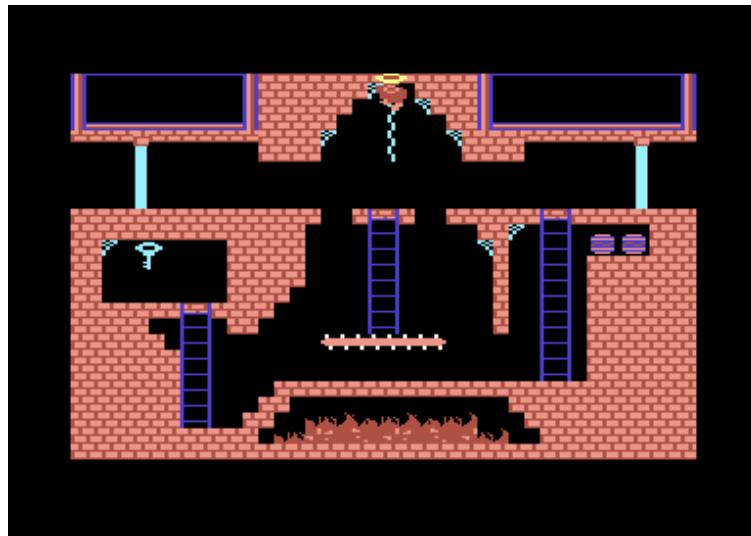
# Types of Exploration

## Random Exploration:

- Take a random action occasionally.
- Add noise to the continuous action output.  
→ Problem: Hard-Exploration Problem

## The Hard-Exploration Problem:

- Some environments have a very sparse or even deceptive reward output.



[https://www.c64-wiki.de/wiki/Montezuma%27s\\_Revenger](https://www.c64-wiki.de/wiki/Montezuma%27s_Revenger)

<https://lilianweng.github.io/posts/2020-06-07-exploration-drl>

<https://arxiv.org/abs/1810.12894>



<https://i.ytimg.com/vi/VHqCAafXpbc/maxresdefault.jpg>

# Types of Exploration

## Random Exploration:

- Take a random action occasionally.
- Add noise to the continuous action output.

➔ Problem: Hard-Exploration Problem

## Count-Based Exploration (Curiosity):

- Count how often a state has been visited.
- Give a reward bonus (intrinsic reward) for rarely visited states.

➔ Problem: Number of states

## Prediction-Based Exploration (Curiosity):

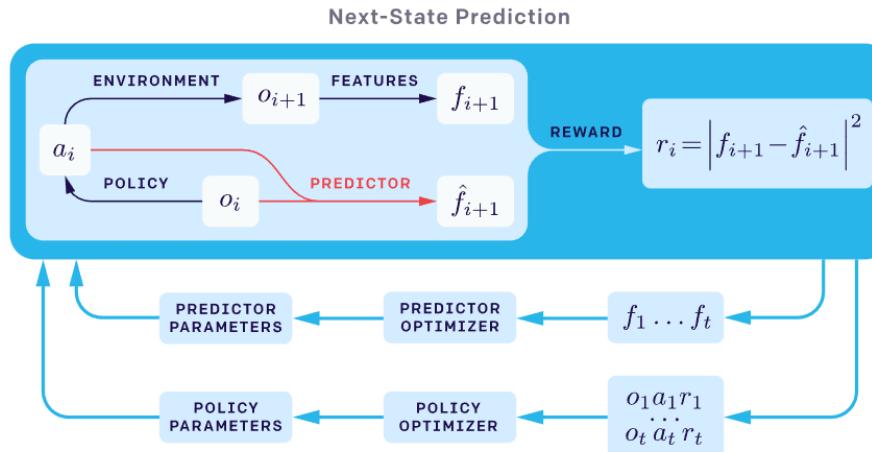
- Assess the agent's knowledge of the environment or the environment dynamics.
- Give a reward bonus (intrinsic reward) for unknown states.

<https://lilianweng.github.io/posts/2020-06-07-exploration-drl>

<https://arxiv.org/abs/1810.12894>

# Prediction-Based Exploration

## Next-State Prediction:



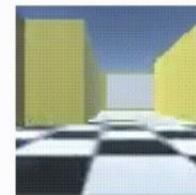
<https://openai.com/research/reinforcement-learning-with-prediction-based-rewards>

## The Noisy-TV Problem:

- An agent that is rewarded for visiting novel states will become a “couch potato” when confronted with an uncontrollable and unpredictable noisy environment.



Agent in a maze with a noisy TV



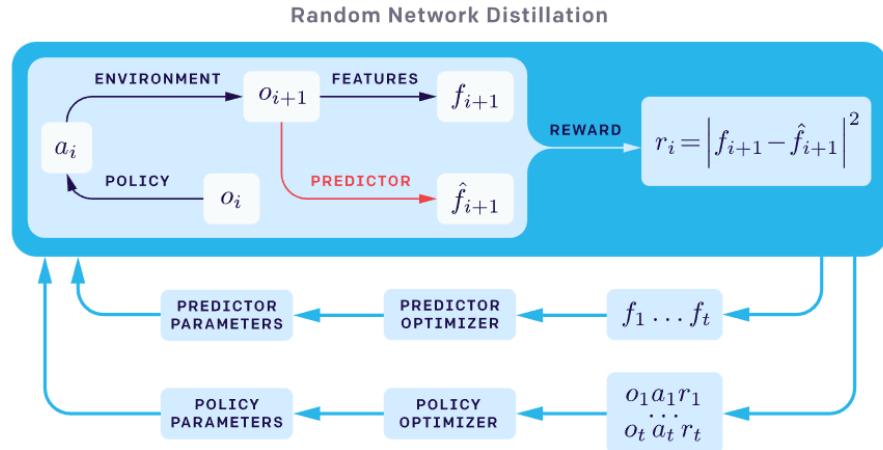
Agent in a maze without a noisy TV

<https://lilianweng.github.io/posts/2020-06-07-exploration-drl>

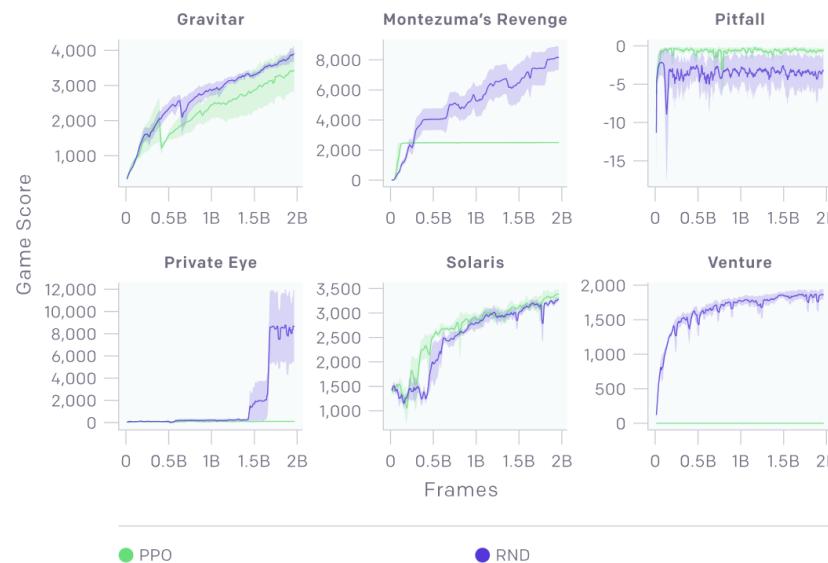
<https://arxiv.org/abs/1810.12894>

# Prediction-Based Exploration

## Random Network Distillation:



<https://openai.com/research/reinforcement-learning-with-prediction-based-rewards>



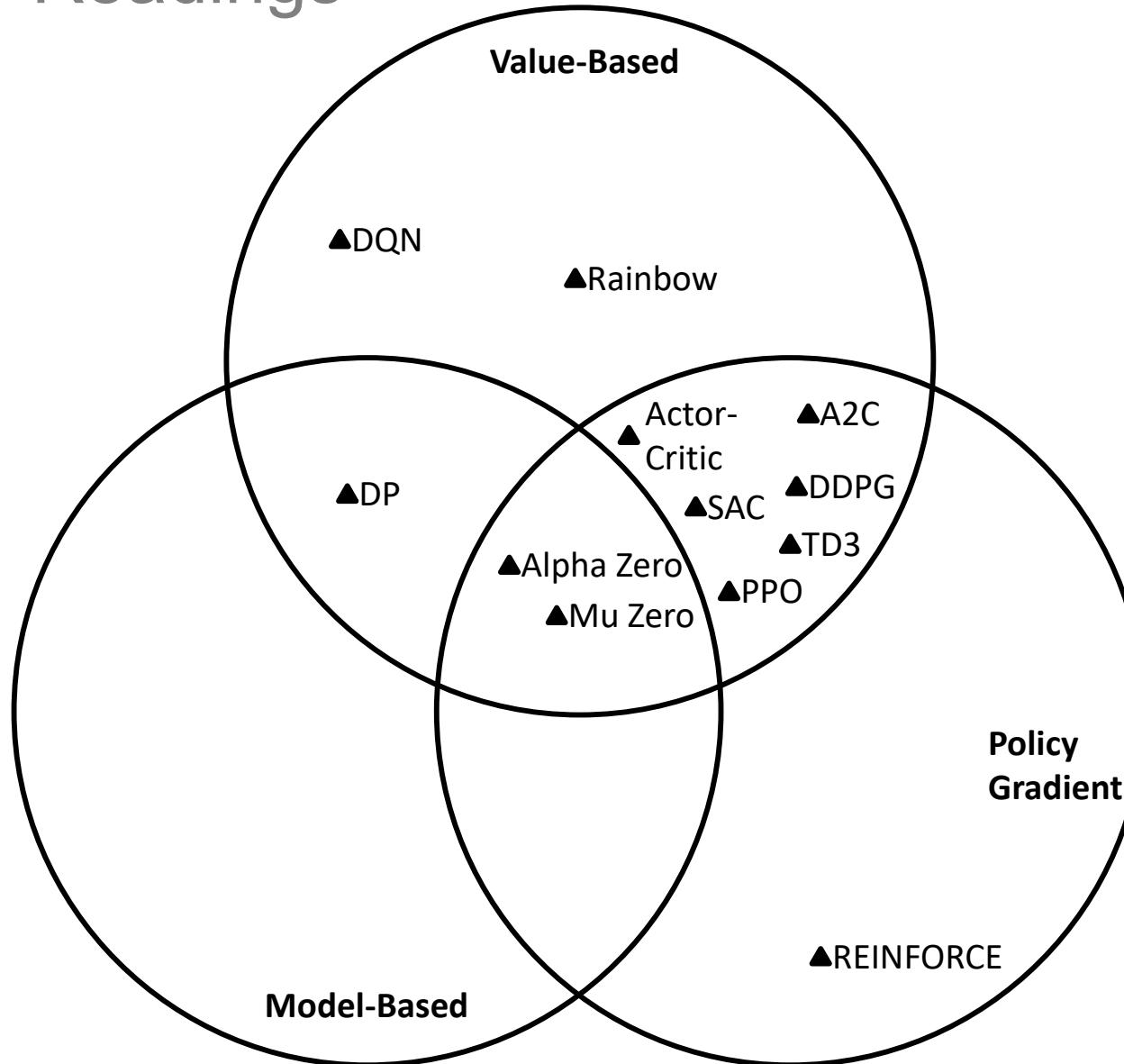
<https://lilianweng.github.io/posts/2020-06-07-exploration-drl>

<https://arxiv.org/abs/1810.12894>



## 8. Further Readings & State of the Art

# Further Readings

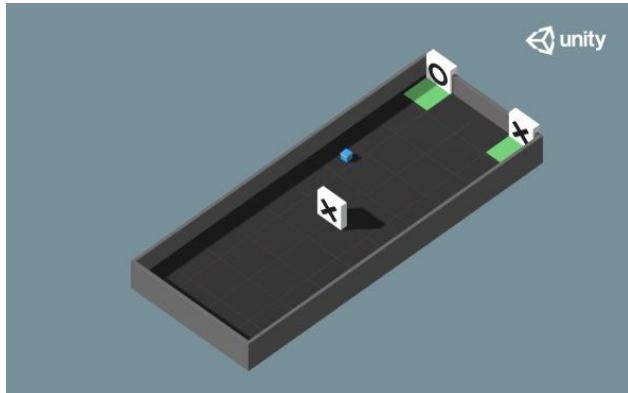


<https://lilianweng.github.io/posts/2018-04-08-policy-gradient/>

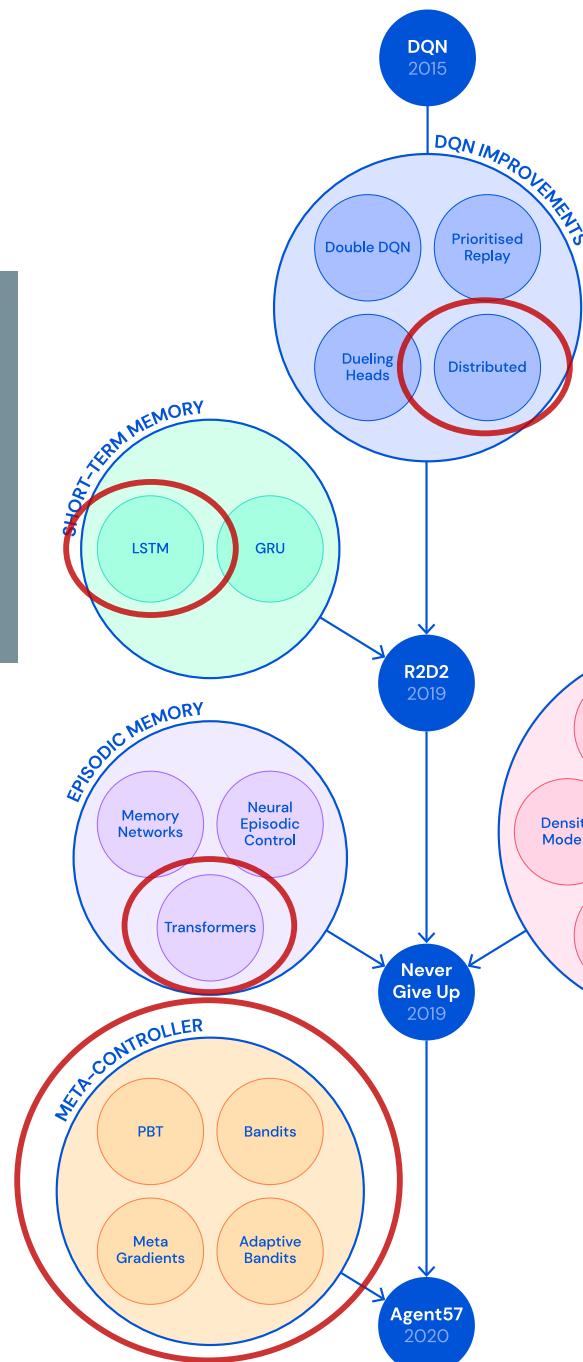
[https://spinningup.openai.com/en/latest/spinningup/rl\\_intro3.html](https://spinningup.openai.com/en/latest/spinningup/rl_intro3.html)

<https://www.packtpub.com/data/deep-reinforcement-learning-hands-on-second-edition>

# Agent 57



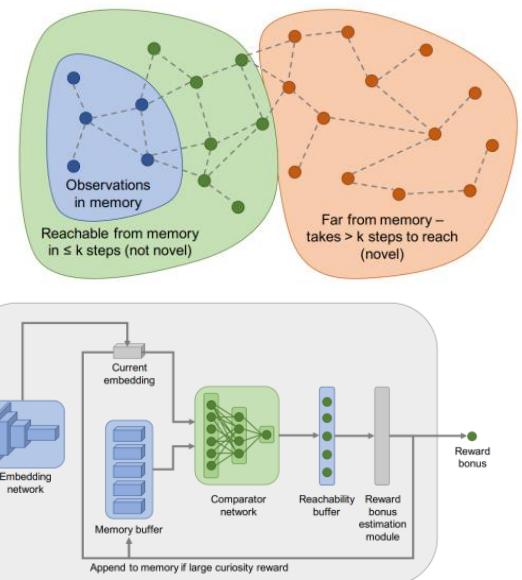
<https://github.com/Unity-Technologies/ml-agents/>



<https://www.deepmind.com/blog/agent57-outperforming-the-human-atari-benchmark>

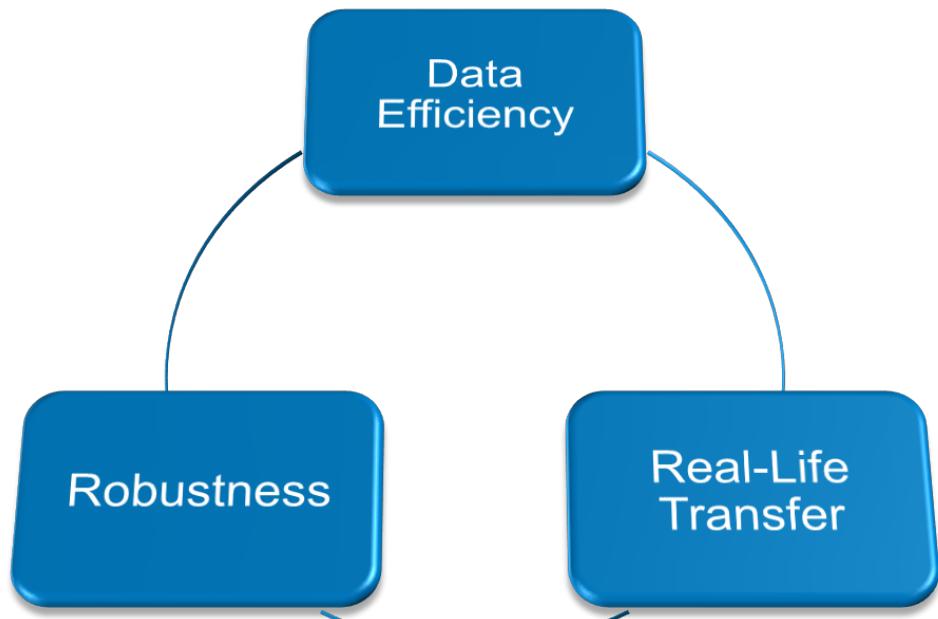


<https://www.youtube.com/watch?v=kopoLzvh5jY>

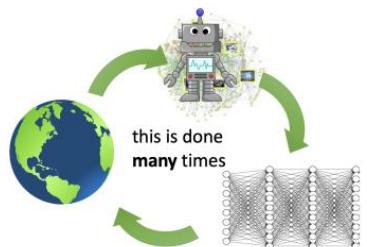


<https://arxiv.org/abs/1810.02274>

# Other Challenges in RL

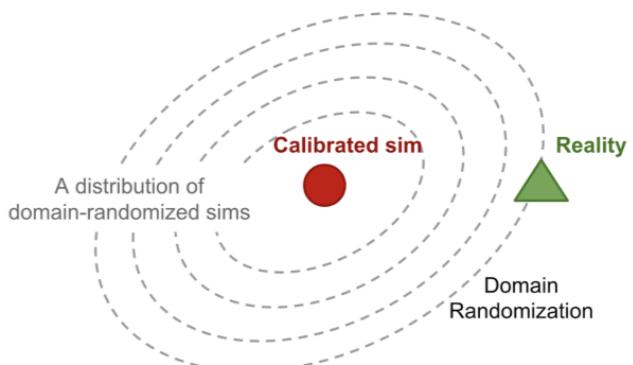
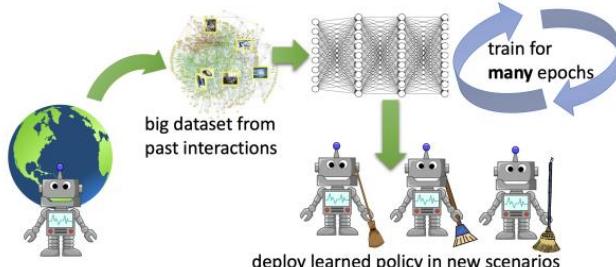


reinforcement learning



<https://bair.berkeley.edu/blog/2020/12/07/offline/>

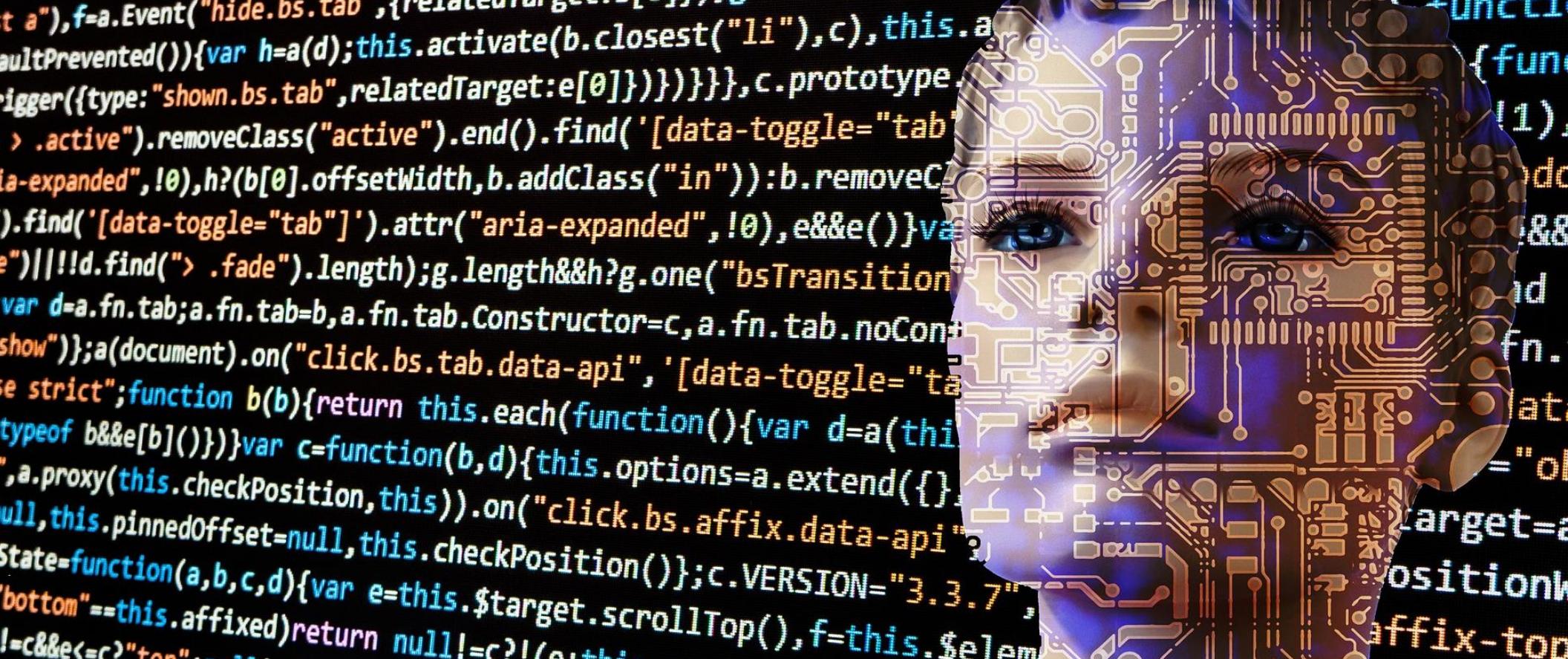
offline reinforcement learning



<https://lilianweng.github.io/posts/2019-05-05-domain-randomization/>



<https://openai.com/research/solving-rubiks-cube>

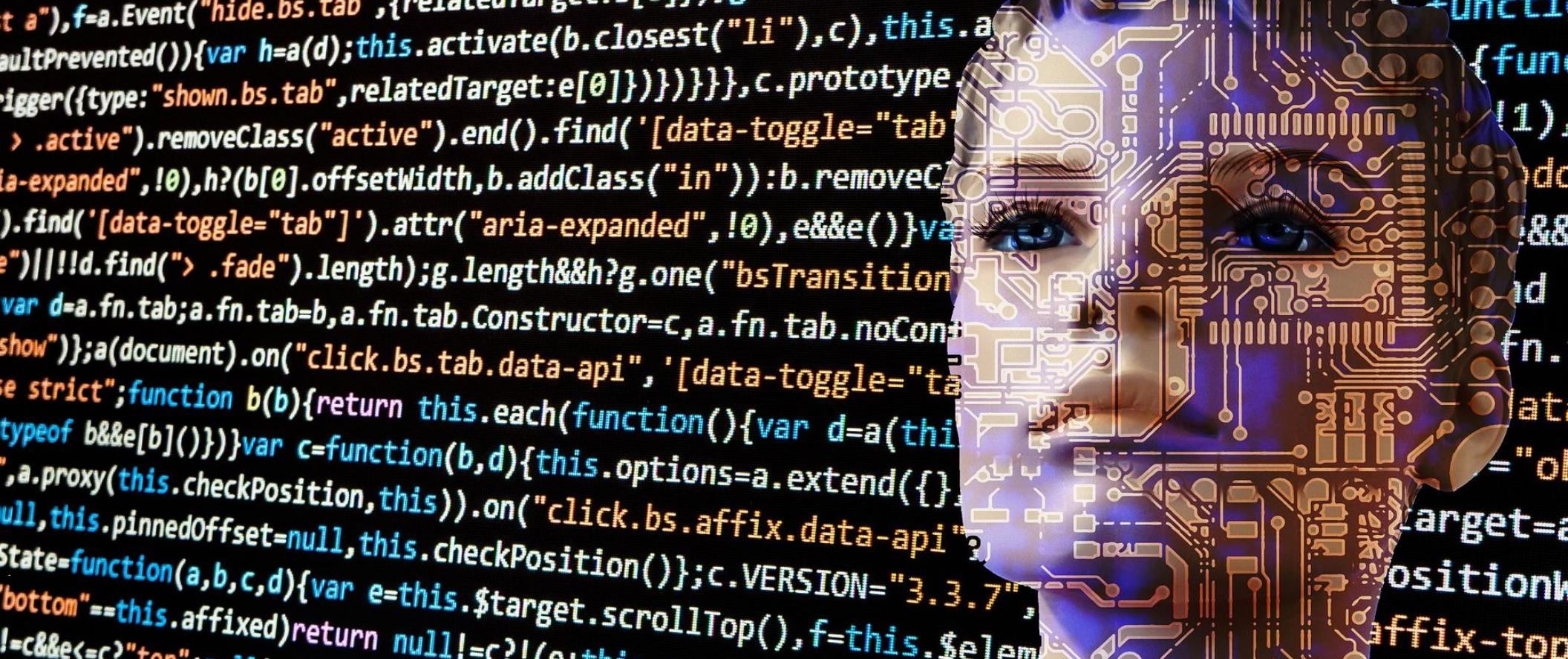


# Autonomous Systems: Deep Learning

## Reinforcement Learning Introduction

# Outline

1. Introduction and Overview
2. Literature
3. Theoretical Foundations
4. Tabular Solution Methods
5. Exploration – Exploitation Dilemma
6. Task 1: Grid World with Tabular Q-Learning
7. Task 2: Improved Exploration



# 1. Introduction and Overview

# What can Reinforcement Learning do?

## Multi-Agent Hide and Seek (OpenAI, 2019)



<https://www.youtube.com/watch?v=kopoLzvh5jY>

# What can Reinforcement Learning do?

## Solving Rubik's Cube with a Robot Hand (OpenAI, 2019)



<https://www.youtube.com/watch?v=x4O8pojMF0w>

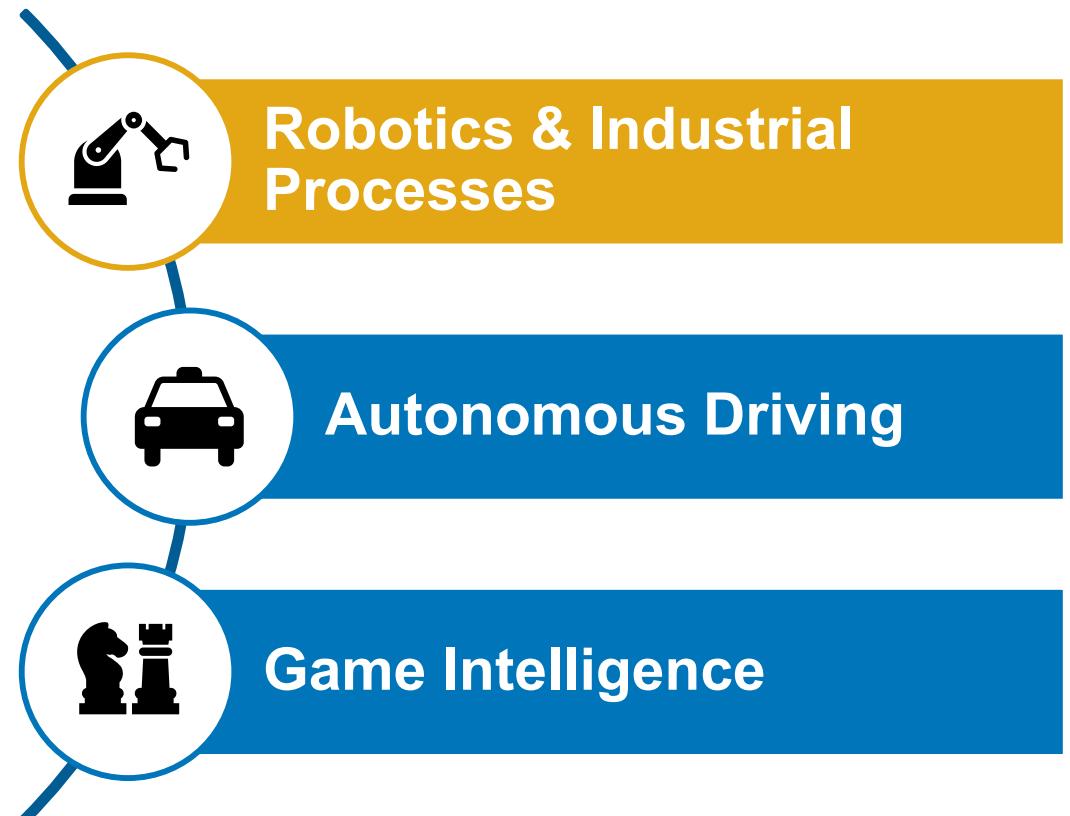
# What can Reinforcement Learning do?

**From Motor Control to Team Play in Simulated Humanoid Football  
(Deep Mind, 2022)**

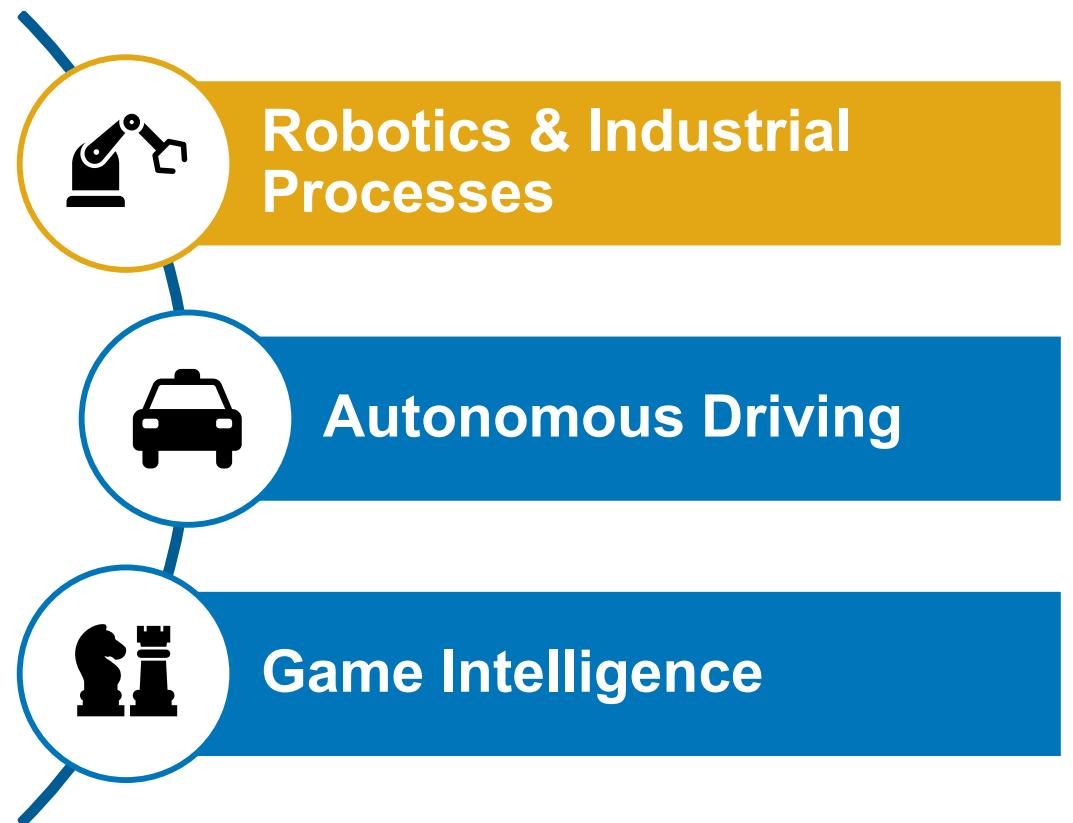


[https://www.youtube.com/watch?v=KHMwq9pv7mg&ab\\_channel=AliEslami](https://www.youtube.com/watch?v=KHMwq9pv7mg&ab_channel=AliEslami)

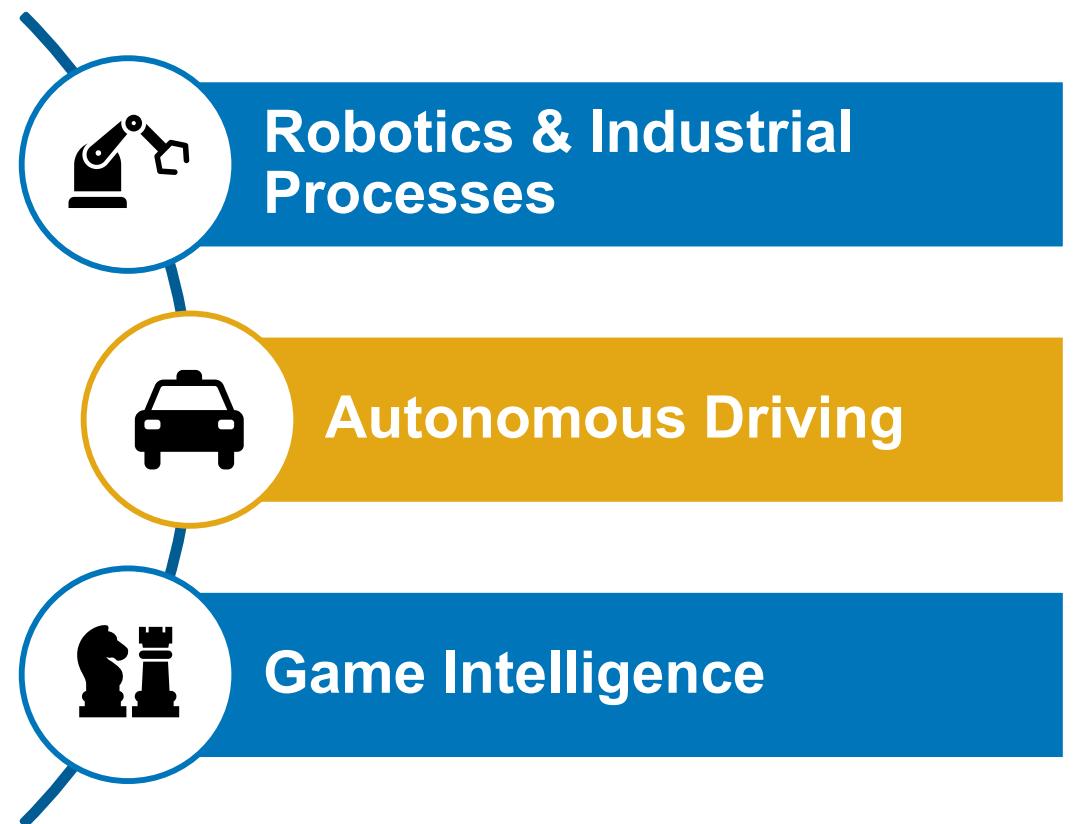
# What can Reinforcement Learning do?



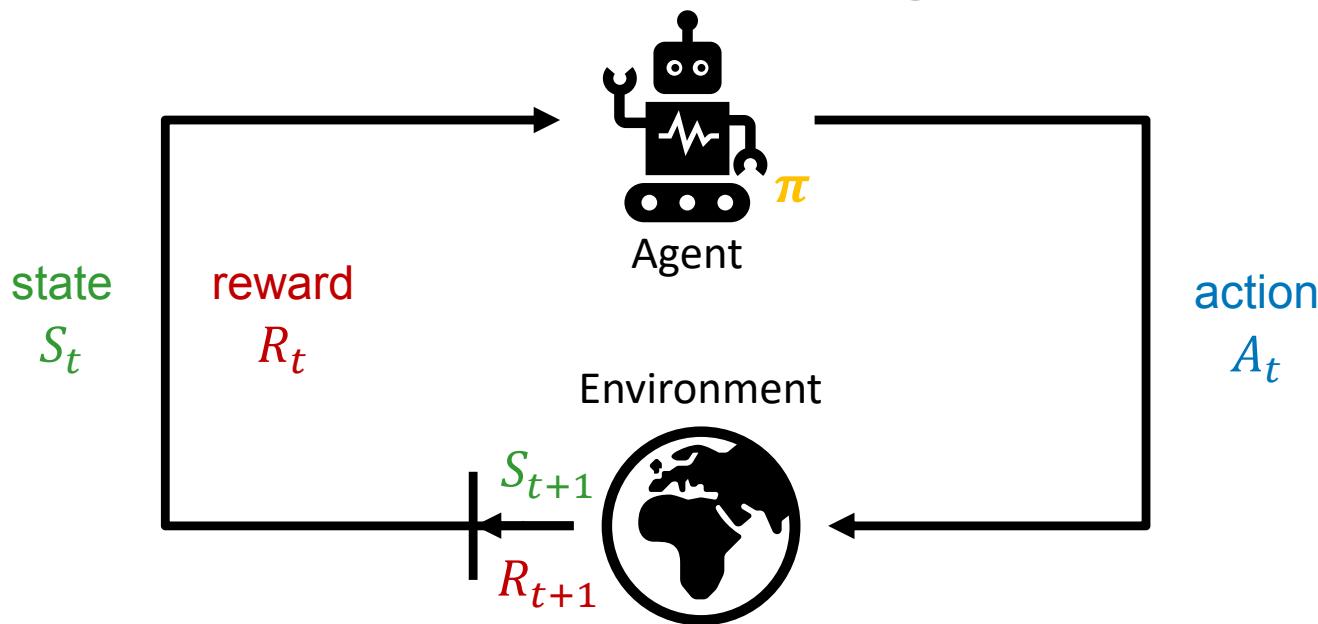
# What can Reinforcement Learning do?



# What can Reinforcement Learning do?



# What is Reinforcement Learning?

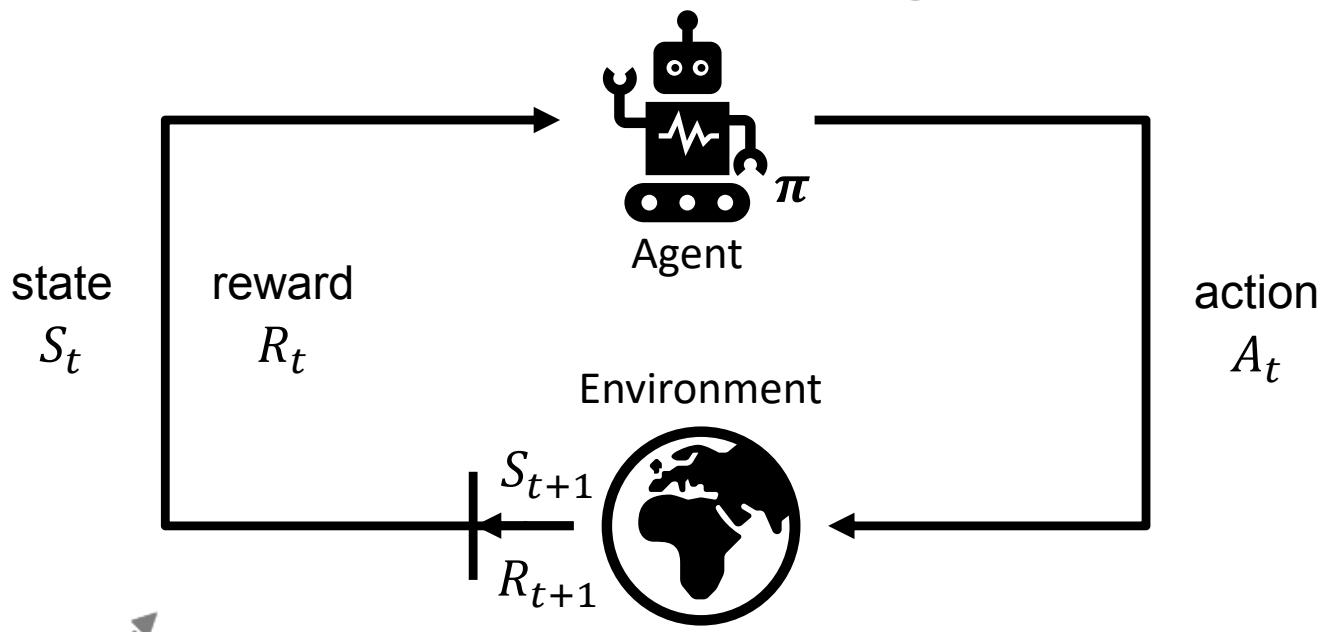


A **policy**  $\pi$  defines the learning agent's way of behaving at a given time. A policy is a mapping from perceived **states** of the environment to **actions** to be taken when in those states.

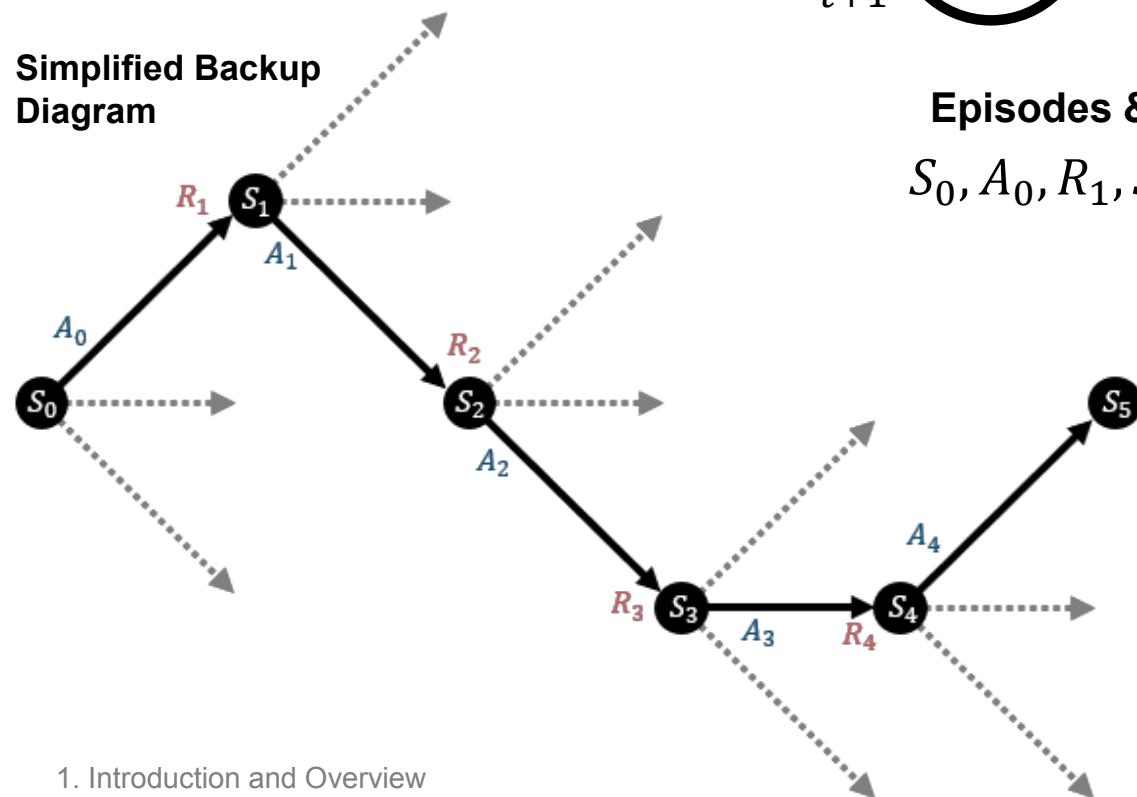
A **reward signal** defines the goal of a reinforcement learning problem. On each time step, the environment sends to the reinforcement learning agent a single number called the **reward**. The agent's sole objective is to maximize the total reward it receives over the long run. The reward signal thus defines what are the good and bad events for the agent.

Whereas the **reward signal** indicates what is good in an immediate sense, a **value function** specifies what is good in the long run. Roughly speaking, the value of a **state** is the total amount of **reward** an agent can expect to accumulate over the future, starting from that **state**.

# What is Reinforcement Learning?



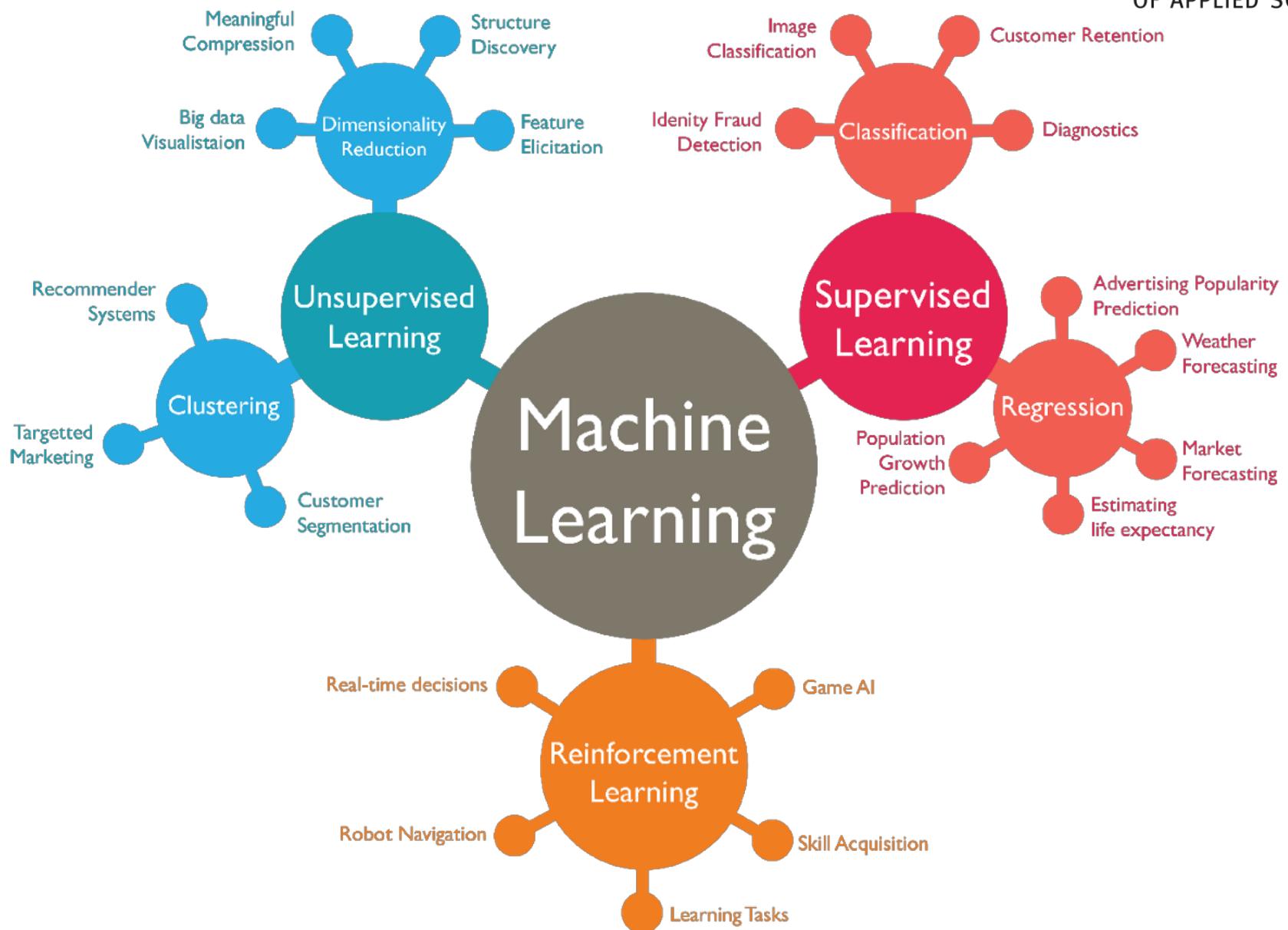
**Simplified Backup Diagram**



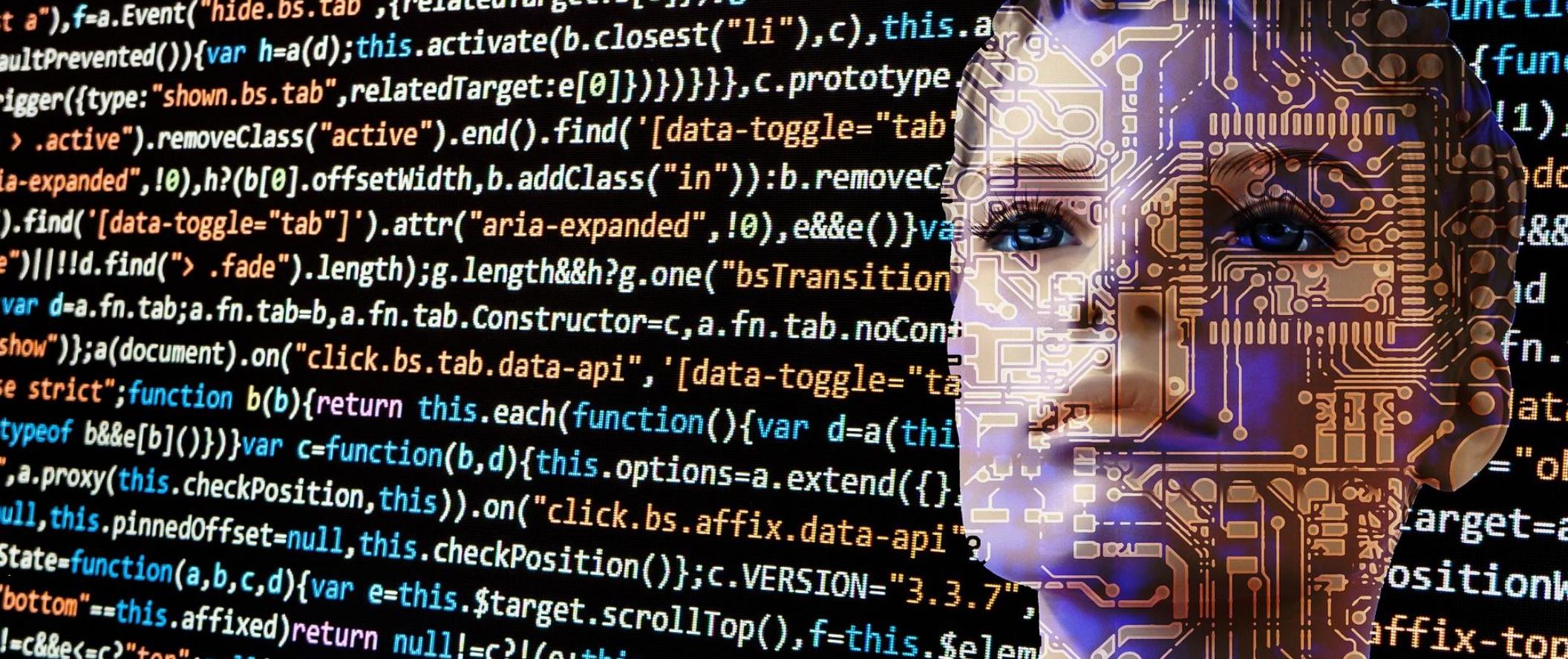
**Episodes & Trajectories**

$S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, R_3, \dots (1)$

# Why Reinforcement Learning?



[https://cdn-images-1.medium.com/max/1200/0\\*\\_cgWPP25djXBauNZ.png](https://cdn-images-1.medium.com/max/1200/0*_cgWPP25djXBauNZ.png)



## 2. Literature

# Literature

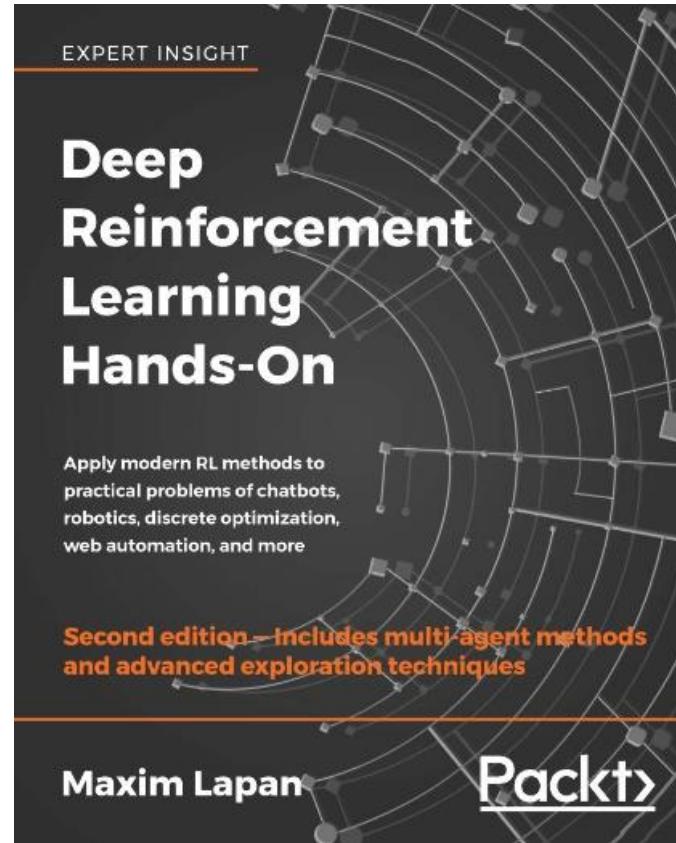
## Reinforcement Learning

An Introduction  
second edition

Richard S. Sutton and Andrew G. Barto



<http://www.andrew.cmu.edu/course/10-703/textbook/BartoSutton.pdf>



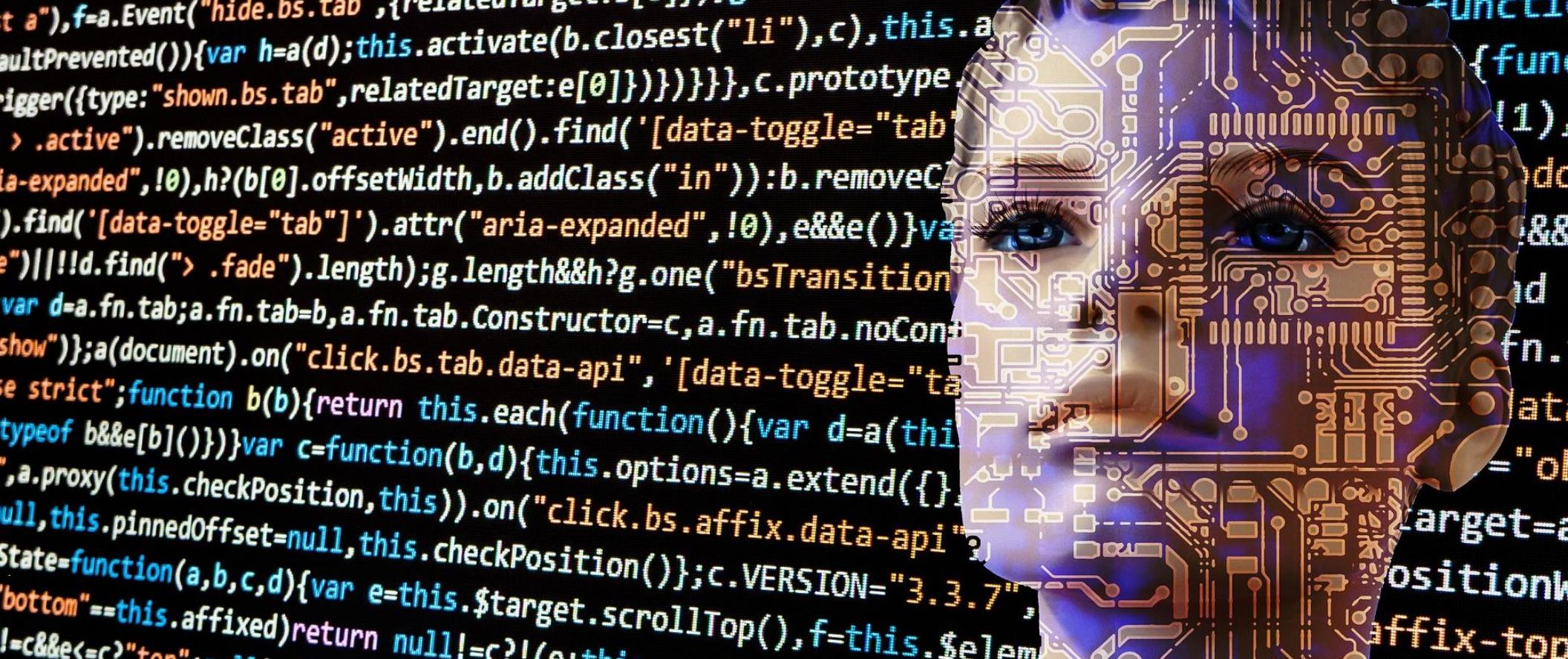
<https://www.packtpub.com/data/deep-reinforcement-learning-hands-on-second-edition>

## A (Long) Peek into Reinforcement Learning

Feb 19, 2018 by Lilian Weng [reinforcement-learning](#) [long-read](#)

In this post, we are gonna briefly go over the field of Reinforcement Learning (RL), from fundamental concepts to classic algorithms. Hopefully, this review is helpful enough so that newbies would not get lost in specialized terms and jargons while starting. [WARNING] This is a long read.

- What is Reinforcement Learning?
  - Key Concepts
    - Model: Transition and Reward
    - Policy
    - Value Function
    - Optimal Value and Policy
  - Markov Decision Processes
    - Bellman Equations
      - Bellman Expectation Equations
      - Bellman Optimality Equations
  - Common Approaches
    - Dynamic Programming
      - Policy Evaluation
      - Policy Improvement
      - Policy Iteration
    - Monte-Carlo Methods
    - Temporal-Difference Learning
      - Bootstrapping
      - Value Estimation



### 3. Theoretical Foundations

# Markov Decision Processes (MDPs)

## MARKOV PROPERTY AND MARKOV CHAIN

**Markov Property:** For any given time, the conditional distribution of future states of the process given present and past states depends only on the present state and not at all on the past states (*memoryless property*).

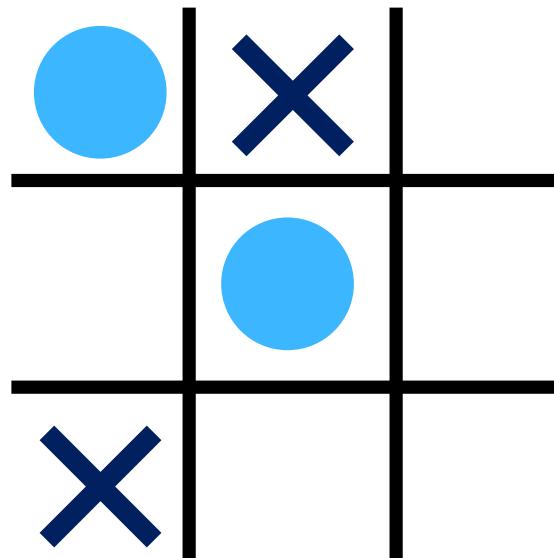
A random process with this property is called **Markov process**.

A **Markov chain** is a Markov process with discrete time and discrete state space.

<https://towardsdatascience.com/brief-introduction-to-markov-chains-2c8cab9c98ab>

$$\mathbb{P}[S_{t+1}|S_t] = \mathbb{P}[S_{t+1}|S_1, S_2, \dots, S_t] \quad (2)$$

Tic-Tac-Toe



# Markov Decision Processes (MDPs)

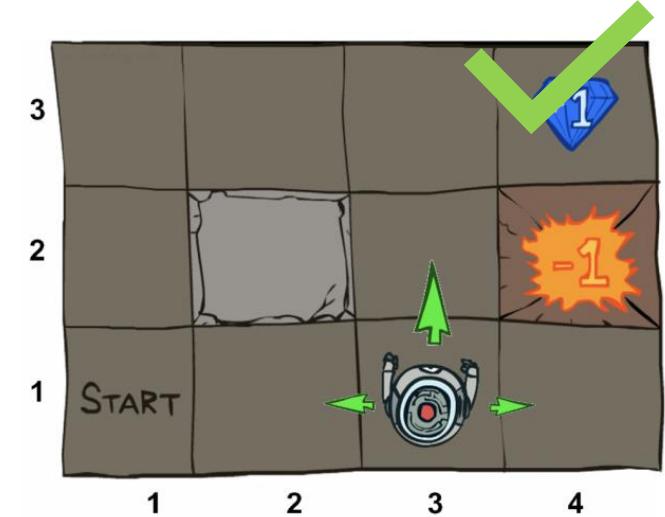
## MARKOV PROPERTY AND MARKOV CHAIN



<https://i.ytimg.com/vi/VHqCAaFXpbc/maxresdefault.jpg>



<https://spiele.rtl.de/kartenspiele/black-jack.html>



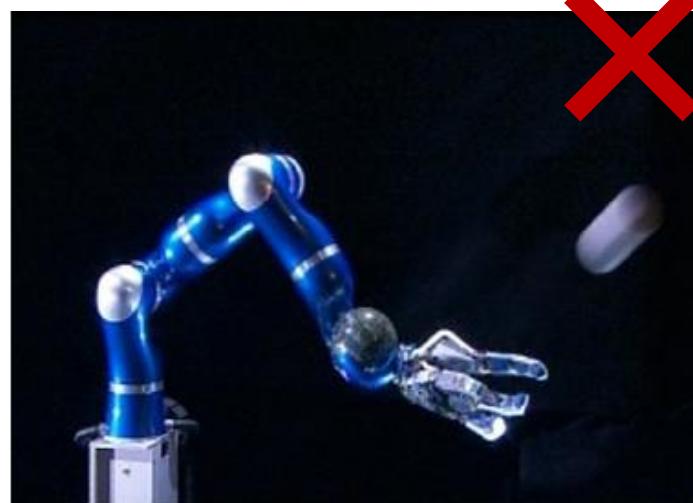
[https://cdn-images-1.medium.com/max/1024/1\\*-oiL7isNsMmktFCNalhTqQ.png](https://cdn-images-1.medium.com/max/1024/1*-oiL7isNsMmktFCNalhTqQ.png)

# Markov Decision Processes (MDPs)

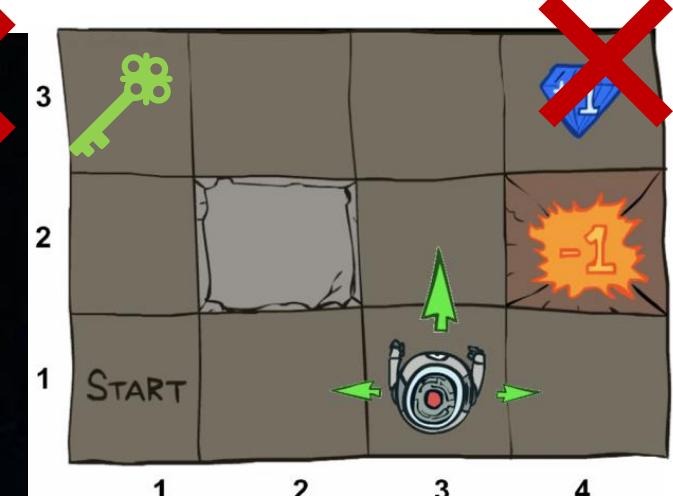
## MARKOV PROPERTY AND MARKOV CHAIN



[https://www.retrogames.cz/play\\_222-Atari2600.php](https://www.retrogames.cz/play_222-Atari2600.php)



<https://www.researchgate.net/profile/Thomas-Wimboeck/publication/225021162/figure/fig1/AS:302781194883082@1449200070953/The-hand-arm-system-catching-a-ball-The-system-is-built-from-the-DLR-LWR-III-arm-with-N.png>



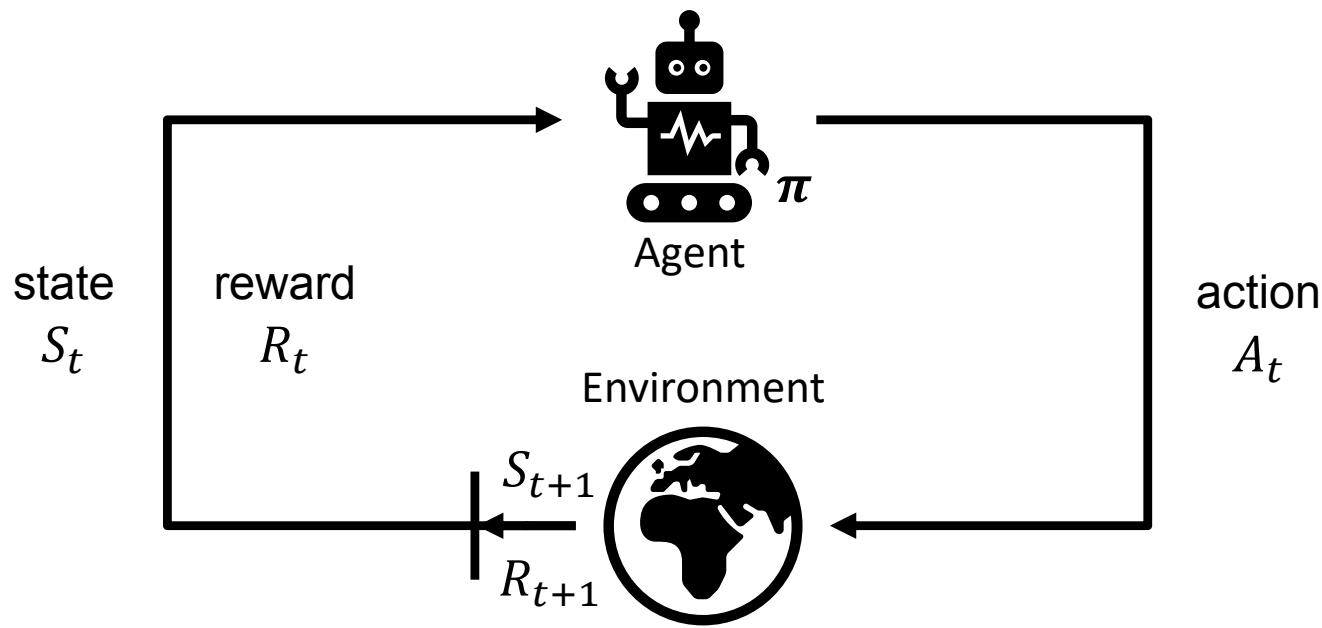
[https://cdn-images-1.medium.com/max/1024/1\\*oiL7isNsMmkFCNalhTqQ.png](https://cdn-images-1.medium.com/max/1024/1*oiL7isNsMmkFCNalhTqQ.png)

### Ways of bypassing the Markov Property:

- State stacking
- Recurrent network architectures (and training algorithms)
- State extensions

# Markov Decision Processes (MDPs)

## AGENT-ENVIRONMENT INTERFACE



### State-Transition Probability

Probability

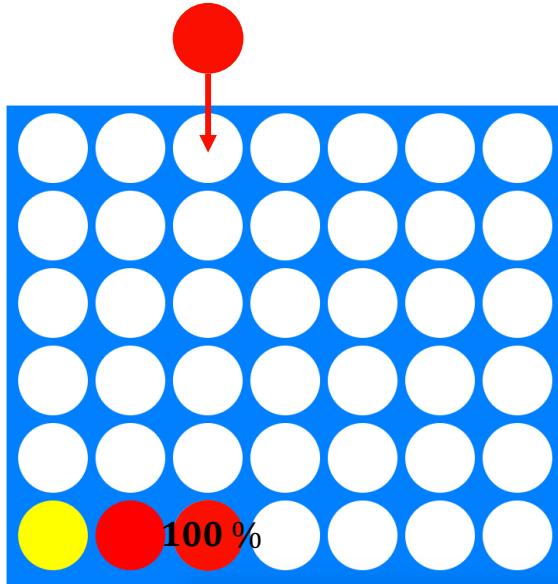
of ending up in state  $s'$

given the case we started in state  $s$  and took action  $a$

$$p(s' | s, a) = \mathbb{P}\{S_{t+1} = s' | S_t = s, A_t = a\} = \sum_{r \in R} p(s', r | s, a) \quad (3)$$

# Markov Decision Processes (MDPs)

## AGENT-ENVIRONMENT INTERFACE



<https://i.stack.imgur.com/v455k.png>



<https://spiele.rtl.de/kartenspiele/black-jack.html>

### State-Transition Probability

*Probability*

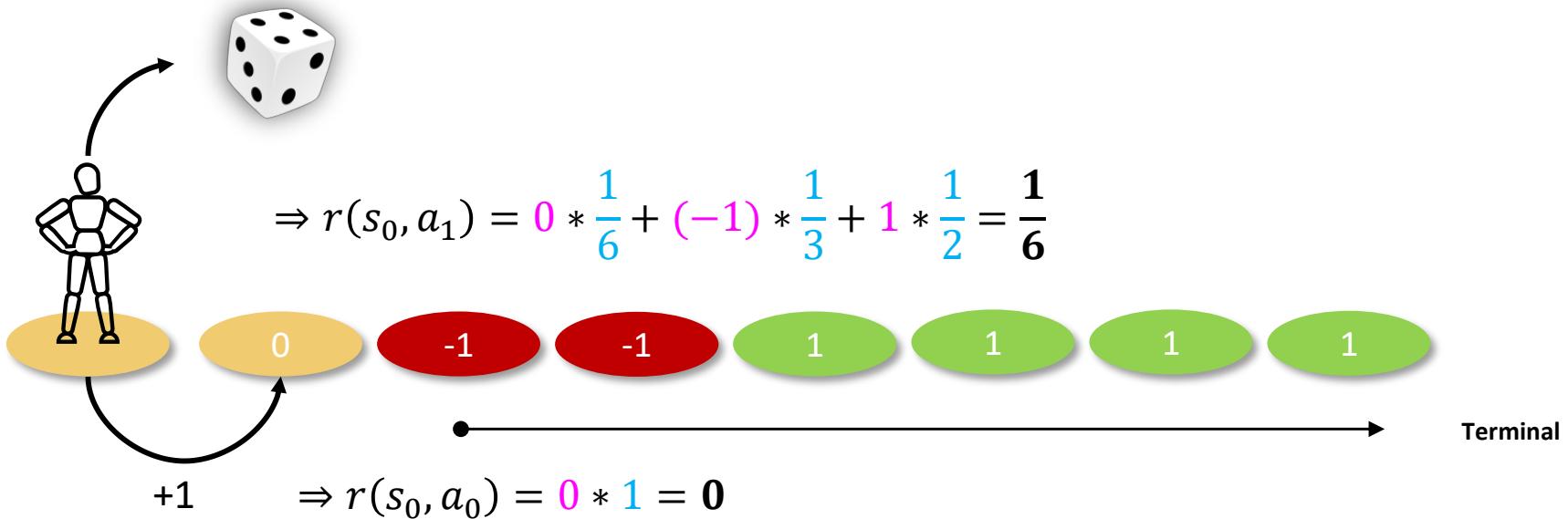
*of ending up in state  $s'$*

*given the case we started in state  $s$  and took action  $a$*

$$p(s'|s, a) = \mathbb{P}\{S_{t+1} = s' | S_t = s, A_t = a\} = \sum_{r \in R} p(s', r | s, a) \quad (3)$$

# Markov Decision Processes (MDPs)

## AGENT-ENVIRONMENT INTERFACE



### State-Transition Probability

$$p(s'|s, a) = \mathbb{P}\{S_{t+1} = s' | S_t = s, A_t = a\} = \sum_{r \in R} p(s', r | s, a) \quad (3)$$

### Expected Reward

*Expected Reward*

when in state  $s$  taking action  $a$

$$r(s, a) = \mathbb{E}[R_t | S_t = s, A_t = a] = \sum_{r \in R} r \sum_{s' \in S} p(s', r | s, a) \quad (4)$$

# Markov Decision Processes (MDPs)

## REWARDS AND RETURNS

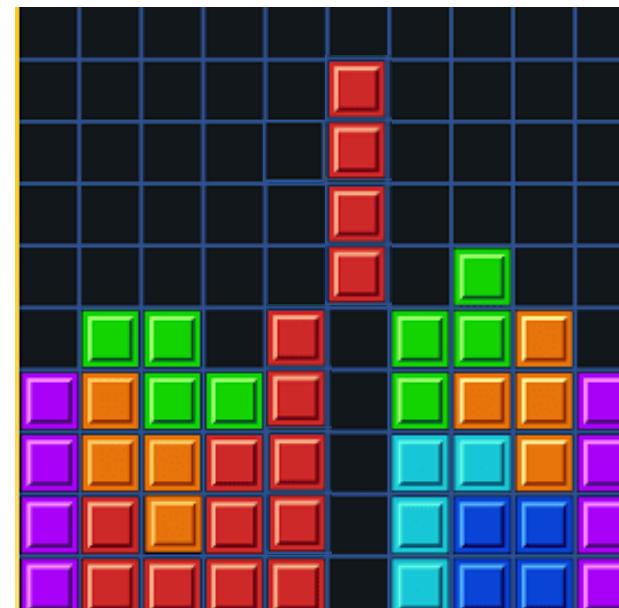
### Return

In general, we seek to maximize the expected return  $G_t$ . In the simplest case the return is the sum of the rewards where  $T$  is the final time step.

$$G_t = R_t + R_{t+1} + R_{t+2} + \dots + R_T \quad (5)$$

### Discounted Return

$$G_t = R_t + \gamma R_{t+1} + \gamma^2 R_{t+2} + \dots = \sum_{k=0}^{\infty} \gamma^k R_{t+k} = \sum_{k=t}^T \gamma^{k-t} R_k \quad (6)$$



[http://images.sftcdn.net/images/t\\_optimized,f\\_auto/p/db2558b0-5fd0-11e7-98e4-117ef89d3ee9/2596915323/classic-tetris-logo.png](http://images.sftcdn.net/images/t_optimized,f_auto/p/db2558b0-5fd0-11e7-98e4-117ef89d3ee9/2596915323/classic-tetris-logo.png)

# Markov Decision Processes (MDPs)

## REWARDS AND RETURNS

$$G_t = R_t + \gamma R_{t+1} + \gamma^2 R_{t+2} + \gamma^3 R_{t+3} \quad (6)$$

$$= R_t + \gamma(R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3})$$

$$= R_t + \gamma G_{t+1} \quad (7)$$

→ Recursive Property

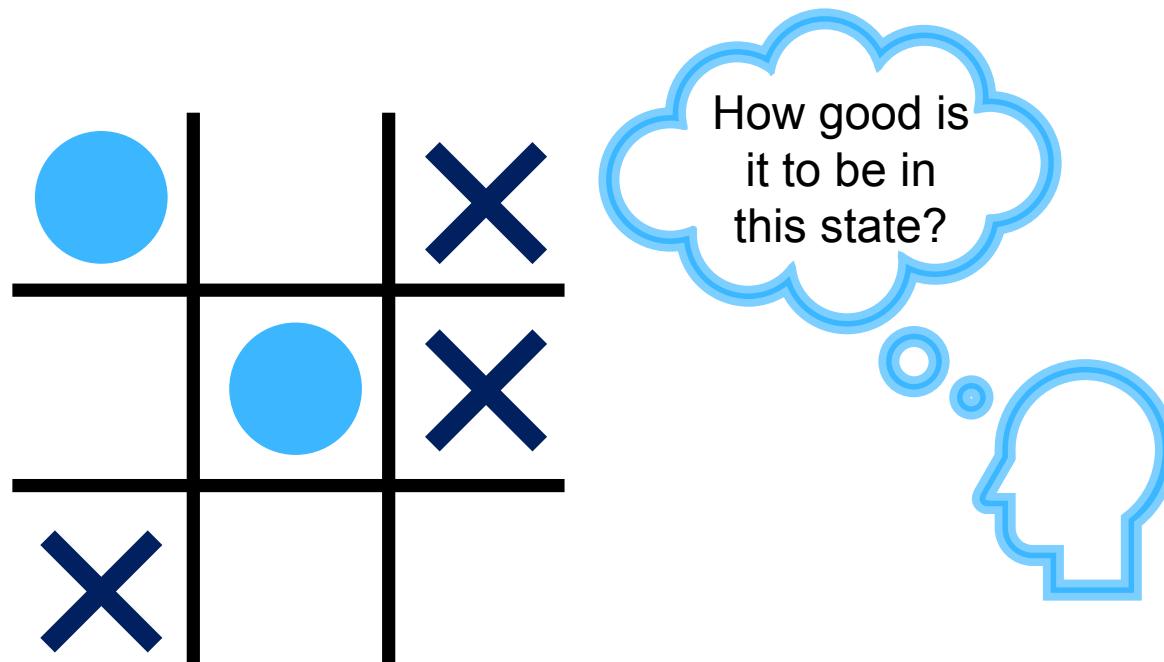
# Markov Decision Processes (MDPs)

## POLICY AND VALUE FUNCTION

### State-Value Function

The value function of a state  $s$  under a policy  $\pi$ , denoted  $v_\pi(s)$ , is the expected return when starting in  $s$  and following  $\pi$  thereafter.

$$v_\pi(s) = \mathbb{E}_\pi[G_t | S_t = s] = \mathbb{E}_\pi[R_t + \gamma G_{t+1} | S_t = s] \quad (8)$$



# Markov Decision Processes (MDPs)

## POLICY AND VALUE FUNCTION

### State-Value Function

The value function of a state  $s$  under a policy  $\pi$ , denoted  $v_\pi(s)$ , is the expected return when starting in  $s$  and following  $\pi$  thereafter.

$$v_\pi(s) = \mathbb{E}_\pi[G_t | S_t = s] = \mathbb{E}_\pi[R_t + \gamma G_{t+1} | S_t = s] \quad (8)$$

### Bellman Equation

$$v_\pi(s) = \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a) [r + \gamma v_\pi(s')] \quad (9)$$

### Bellman Optimality Equation

$$v_*(s) = \max_a \mathbb{E}_{\pi_*}[G_t | S_t = s, A_t = a] \quad (10)$$

$$= \max_a \sum_{s',r} p(s',r|s,a) [r + \gamma v_*(s')]$$



## 4. Tabular Solution Methods

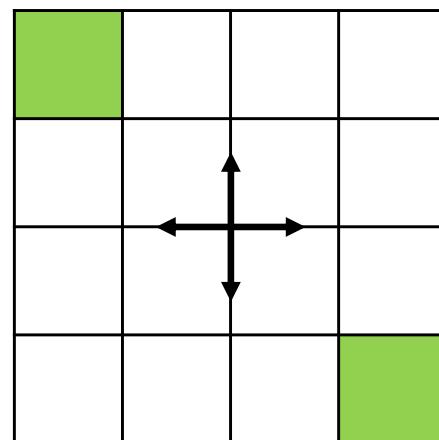
# Dynamic Programming

## POLICY EVALUATION (PREDICTION)

$$v_{\pi}(s) = \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a)[r + \gamma v_{\pi}(s')] \quad (9)$$

$$v_{k+1}(s) = \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a)[r + \gamma v_k(s')] \quad (11)$$

where  $k$  is the current iteration



$R_t = -1$   
on all transitions  
 $\gamma = 1$

Equiprobable Policy  $\pi$

# Dynamic Programming

## POLICY EVALUATION (PREDICTION)

0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0

$k = 0$

0	-1	-1	-1
-1	-1	-1	-1
-1	-1	-1	-1
-1	-1	-1	0

$k = 1$

0	-1.75	-2	-2
-1.75	-2	-2	-2
-2	-2	-2	-1.75
-2	-2	-1.75	0

$k = 2$

0	-2.44	-2.94	-3
-2.44	2.88	-3	2.94
-2.94	-3	2.88	2.44
-3	2.94	2.44	0

$k = 3$

$$v_{k+1}(s) = \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a) [r + \gamma v_k(s')] \quad (11)$$

$$\begin{aligned} v_1(s_{(0,1)}) &= 0.25 * 1 * [-1 + 0] \\ &+ 0.25 * 1 * [-1 + 0] \\ &+ 0.25 * 1 * [-1 + 0] \\ &+ 0.25 * 1 * [-1 + 0] \end{aligned}$$

$$\begin{aligned} v_2(s_{(0,1)}) &= 0.25 * 1 * [-1 + 0] \\ &+ 0.75 * 1 * [-1 + (-1)] \end{aligned}$$

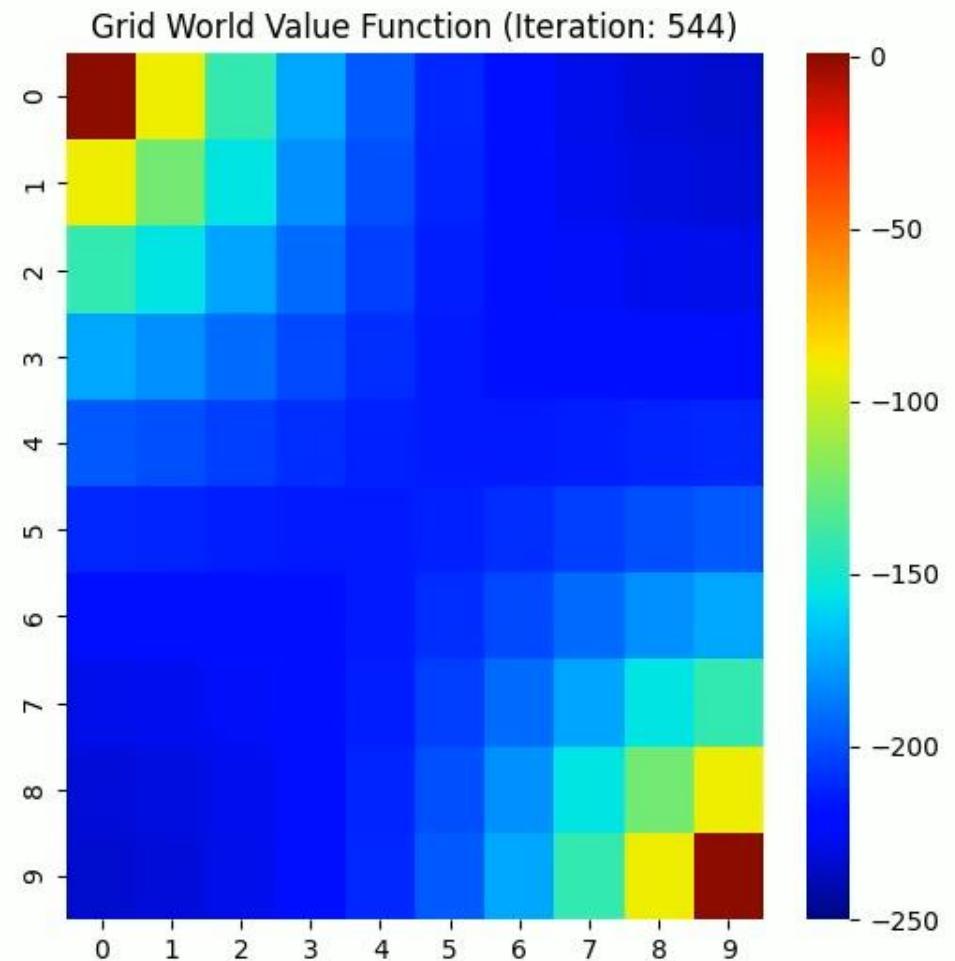
$$\begin{aligned} v_3(s_{(0,1)}) &= 0.25 * 1 * [-1 + 0] \\ &+ 0.25 * 1 * [-1 + (-1.75)] \\ &+ 0.5 * 1 * [-1 + (-2)] \end{aligned}$$

# Dynamic Programming

## POLICY EVALUATION (PREDICTION)

0	-	2.44	2.94	-3
-	-	2.88	-3	2.94
2.44	2.88	-3	2.88	2.44
-3	-	2.94	2.44	0

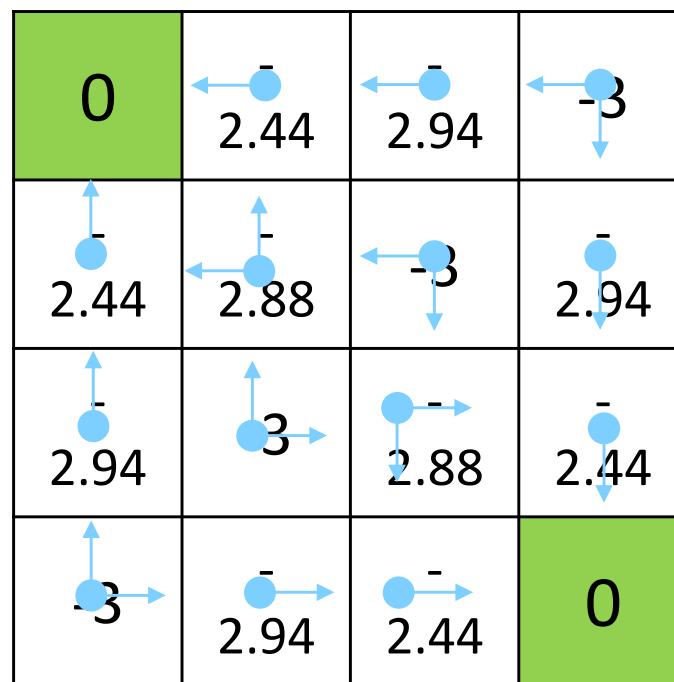
$k = 3$



# Dynamic Programming

## POLICY IMPROVEMENT

$$\pi'(s) = \operatorname{argmax}_a \sum_{s',r} p(s',r|s,a)[r + \gamma v_\pi(s')] \quad (12)$$



$$k = 3$$

$$\pi_0 \xrightarrow{\text{E}} v_{\pi_0} \xrightarrow{\text{I}} \pi_1 \xrightarrow{\text{E}} v_{\pi_1} \xrightarrow{\text{I}} \pi_2 \xrightarrow{\text{E}} \dots \xrightarrow{\text{I}} \pi_* \xrightarrow{\text{E}} v_*$$

<http://www.andrew.cmu.edu/course/10-703/textbook/BartoSutton.pdf>

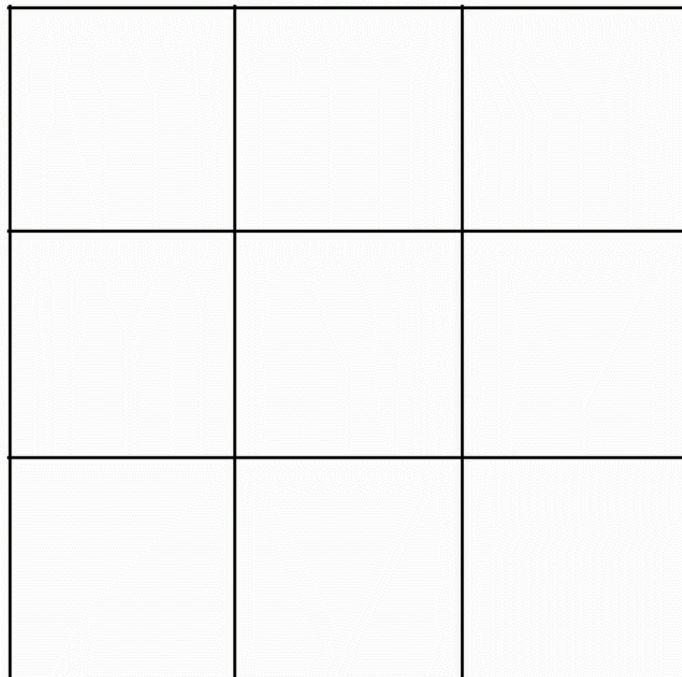
# Dynamic Programming

## PROBLEMS & CONSTRAINTS

**Number of States:**

$$3^9 = 19683 \text{ States}$$

Tic Tac Toe State #1



$$7.7 * 10^{45} \text{ States}$$

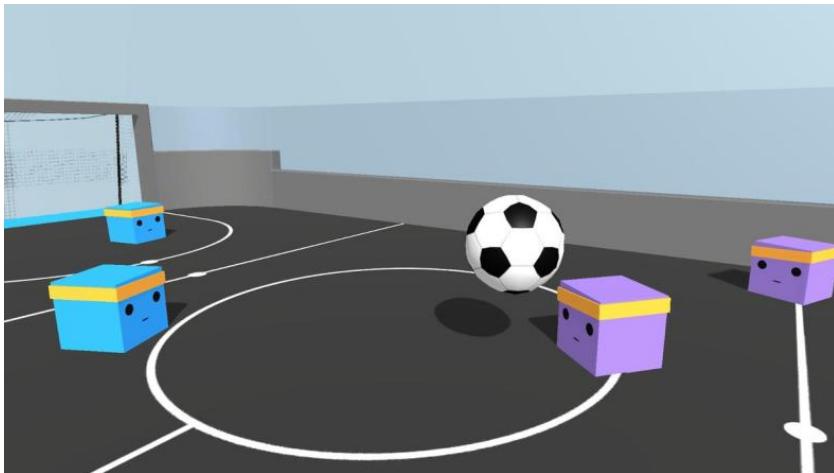


<https://i.ytimg.com/vi/VHqCAaFXpbc/maxresdefault.jpg>

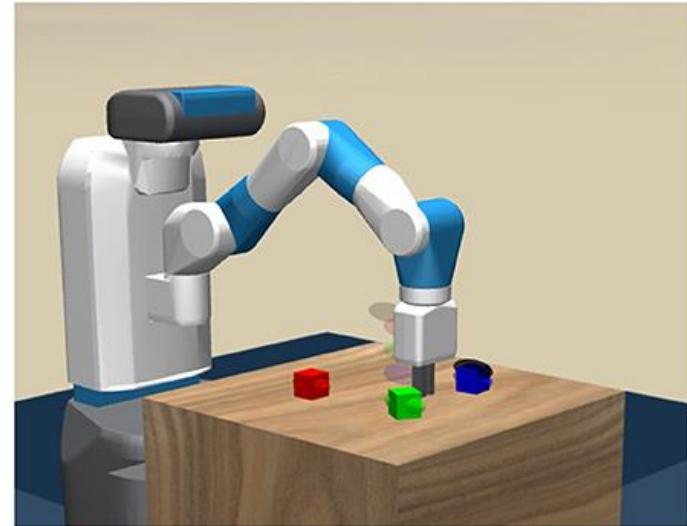
# Dynamic Programming

## PROBLEMS & CONSTRAINTS

### Model Dependence



[https://blog-api.unity.com/sites/default/files/styles/social\\_media/public/2020/02/image2-4.png.jpg?itok=2hOn0JUp](https://blog-api.unity.com/sites/default/files/styles/social_media/public/2020/02/image2-4.png.jpg?itok=2hOn0JUp)



[https://www.frontiersin.org/files/Articles/473936/frobt-06-00123-HTML/image\\_m/frobt-06-00123-g001.jpg](https://www.frontiersin.org/files/Articles/473936/frobt-06-00123-HTML/image_m/frobt-06-00123-g001.jpg)

$$v_{\pi}(s) = \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a) [r + \gamma v_{\pi}(s')] \quad (9)$$

$$v_*(s) = \max_a \sum_{s',r} p(s',r|s,a) [r + \gamma v_*(s')] \quad (10)$$

### Dynamic Programming is limited by:

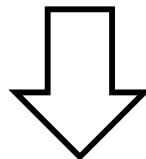
- The number of possible states in an environment (time & memory)
- The model dependence (state transition probabilities)

# Dynamic Programming

## PROBLEMS & CONSTRAINTS

Dynamic Programming is limited by:

- The number of possible states in an environment (time & memory)
- The model dependance (state transition probabilities)



### 1. Learning the Action-Value

$$q_{\pi}(s, a) = \mathbb{E}_{\pi}[G_t | S_t = s, A_t = a] = \mathbb{E}_{\pi}[R_t + \gamma G_{t+1} | S_t = s, A_t = a] \quad (13)$$

$$q_{\pi}(s, a) = \sum_{s', r} p(s', r | s, a) [r + \gamma v_{\pi}(s')] \quad (14)$$

$$q_{*}(s, a) = \sum_{s', r} p(s', r | s, a) \left[ r + \gamma \max_{a'} q_{*}(s', a') \right] \quad (15)$$

### 2. Learning from Experience

# Monte Carlo Methods (MC)

## Intuition behind Monte Carlo Control

- Create a Q-Table with one entry for each state action pair

while true:

- Play one episode and store the rewards received at each step
- Calculate the return for each visited state in the episode
- Update the Q-Values a small step towards the direction of experienced return

## Monte Carlo Update

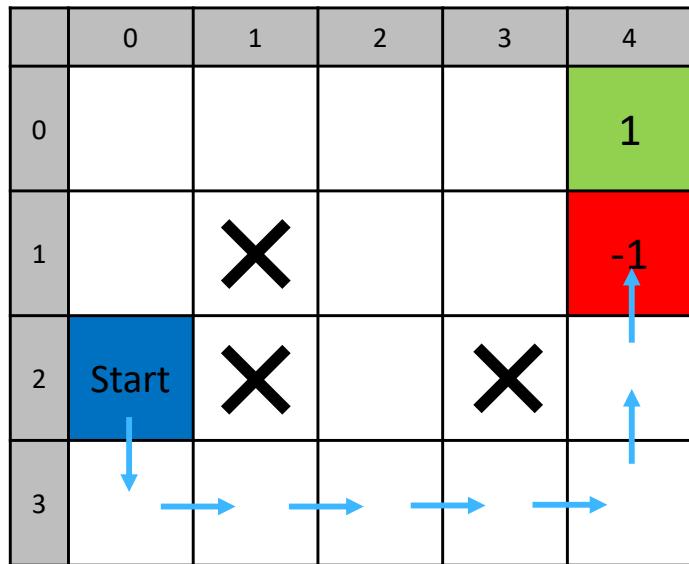
$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha [G_t - Q(S_t, A_t)] \quad (16)$$

Learning Rate      Error

# Monte Carlo Methods (MC)

## Monte Carlo Update

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[G_t - Q(S_t, A_t)] \quad (16)$$



$R_t = -0.1$   
on all other transitions

State	Left	Right	Top	Bottom
...	...	...	...	...
(2,0)	0	0	0	0
(3,0)	0	0	0	0
(0,1)	0	0	0	0
(3,1)	0	0	0	0
(0,2)	0	0	0	0
...	...	...	...	...

$$\gamma = 1; \alpha = 0.1$$

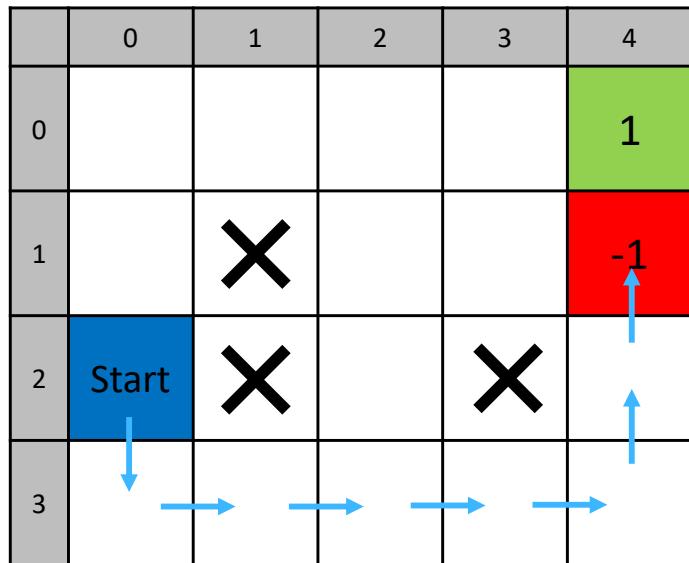
$$T_{R_t} = (-0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -1)$$

$$G_t = (-1.6, -1.5, -1.4, -1.3, -1.2, -1.1, -1)$$

# Monte Carlo Methods (MC)

## Monte Carlo Update

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[G_t - Q(S_t, A_t)] \quad (16)$$



$R_t = -0.1$   
on all other transitions

State	Left	Right	Top	Bottom
...	...	...	...	...
(2,0)	0	0	0	-0.16
(3,0)	0	-0.15	0	0
(0,1)	0	0	0	0
(3,1)	0	-0.14	0	0
(0,2)	0	0	0	0
...	...	...	...	...

$$\gamma = 1; \alpha = 0.1$$

$$T_{R_t} = (-0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -1)$$

$$G_t = (-1.6, -1.5, -1.4, -1.3, -1.2, -1.1, -1)$$

# Temporal-Difference Learning

## TD PREDICTION

### Monte Carlo Problem:

- Updates to Q-Values of visited states can only happen once the episode has finished.

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha [G_t - Q(S_t, A_t)] \quad (16)$$

### TD Learning Update:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[ R_{t+1} + \gamma \max_{a' \in A} Q(S_{t+1}, a') - Q(S_t, A_t) \right] \quad (17)$$

# Temporal-Difference Learning

## ADVANTAGES OF TD PREDICTION METHODS

- Temporal-Difference methods update their estimates based in part on other estimates - they **bootstrap**. They are naturally implemented in an online fashion; one must wait only one time step to update.
  
- TD methods do not require a model of the environment, of its reward and next-state probability distributions as opposed to DP methods.

# Temporal-Difference Learning

## Q-LEARNING: OFF-POLICY TD CONTROL

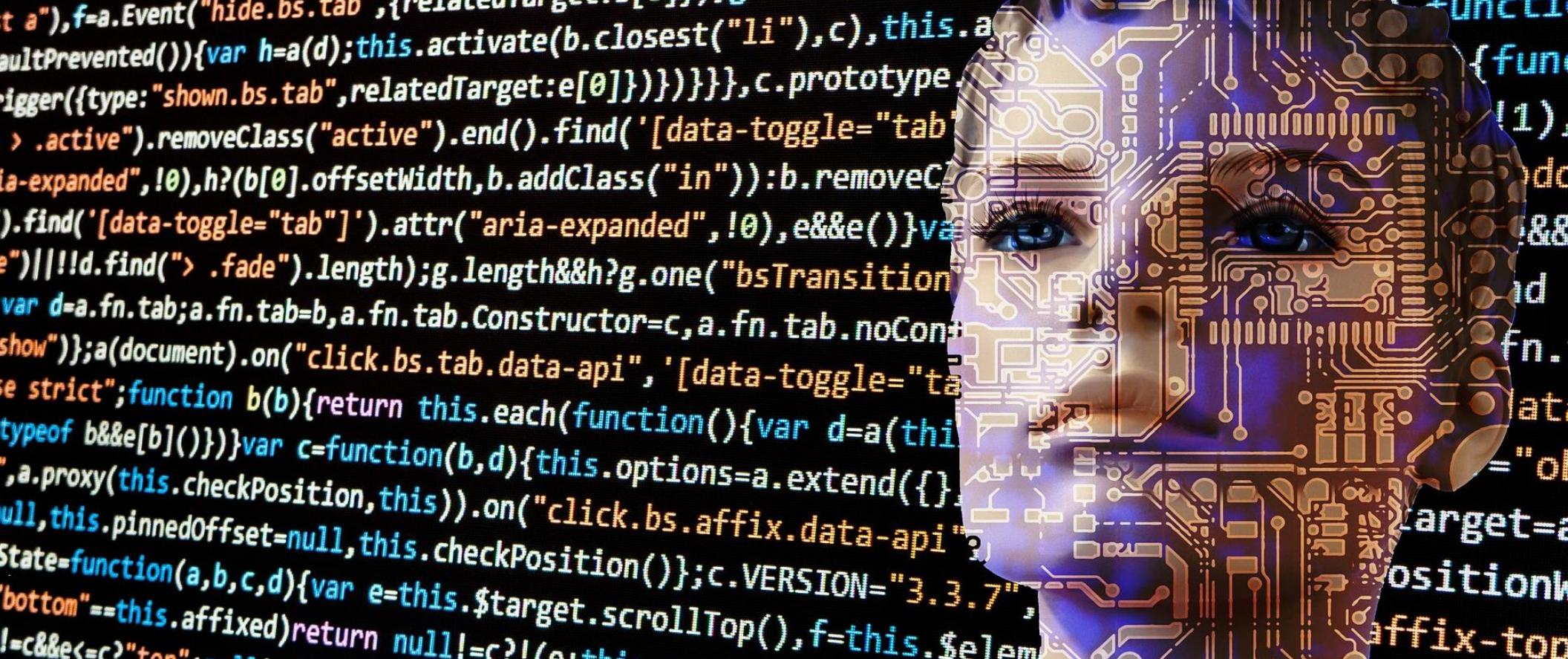
### Intuition behind Q-Learning

- Create a Q-Table with one entry for each state action pair

while true:

- Play one step in the environment (reset environment when episode ends)
- Update the Q-Value for the previous state

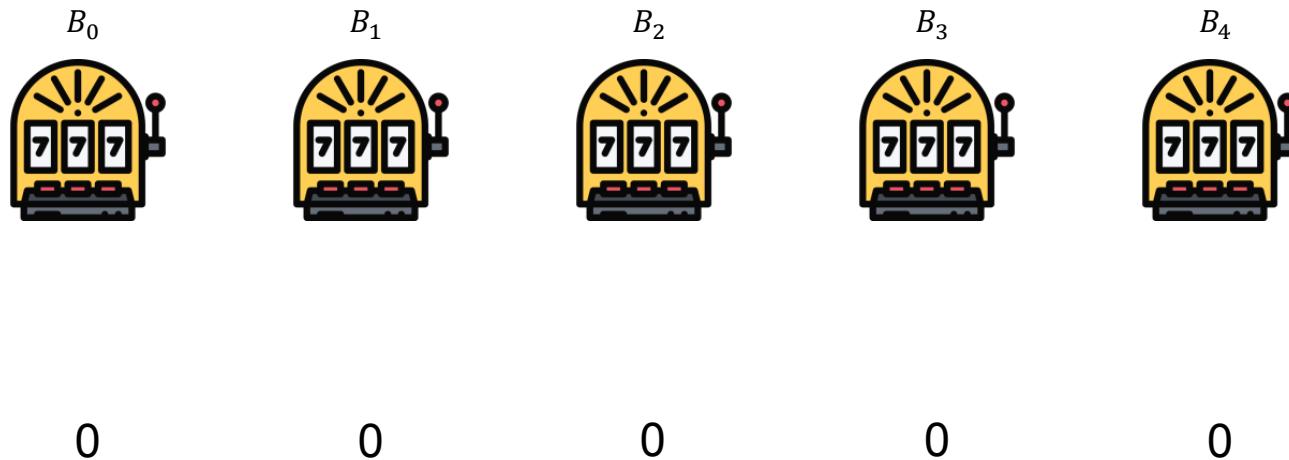
$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[ R_{t+1} + \gamma \max_{a' \in A} Q(S_{t+1}, a') - Q(S_t, A_t) \right] \quad (17)$$



## 5. Exploration - Exploitation Dilemma

# K-Armed Bandit Problem

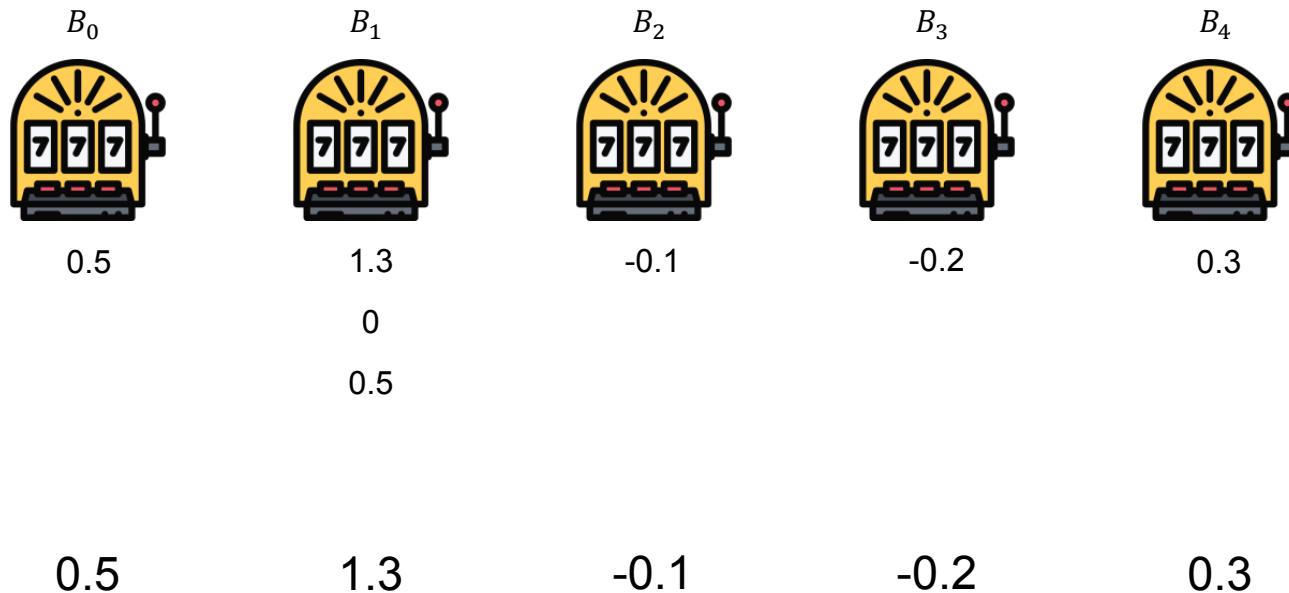
**Objective:** Maximize the total reward over some time period, for example, over 1000 action selections or time steps.



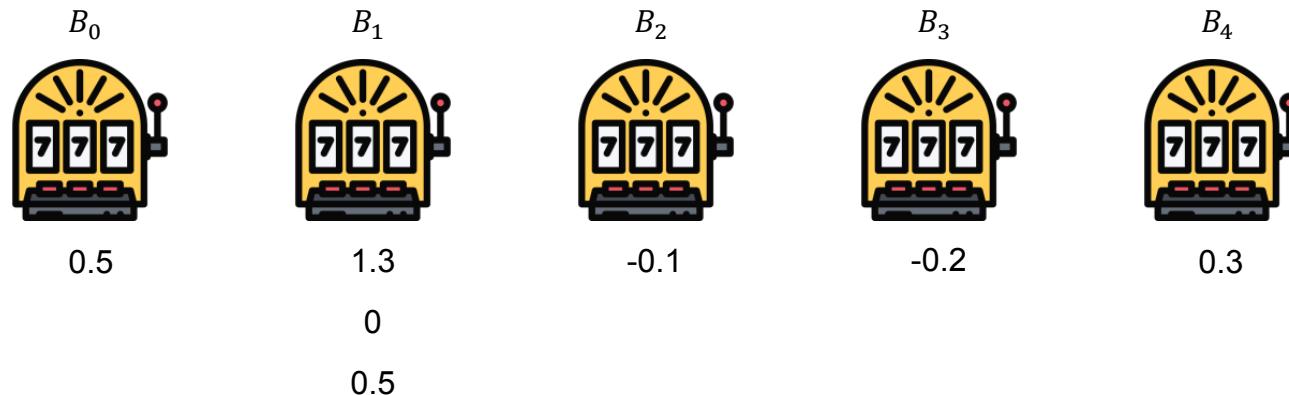
$$Q_{n+1} = \frac{R_1 + R_2 + R_3 + \dots + R_n}{n} = \frac{1}{n} \sum_{i=1}^n R_i \quad (18)$$

$$Q_{n+1} = Q_n + \frac{1}{n} [R_n - Q_n] \quad (19)$$

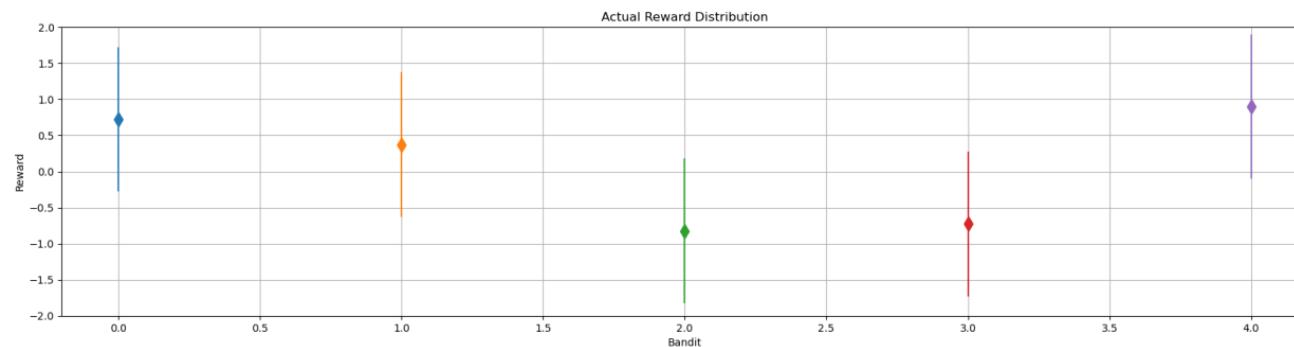
# K-Armed Bandit Problem



# K-Armed Bandit Problem

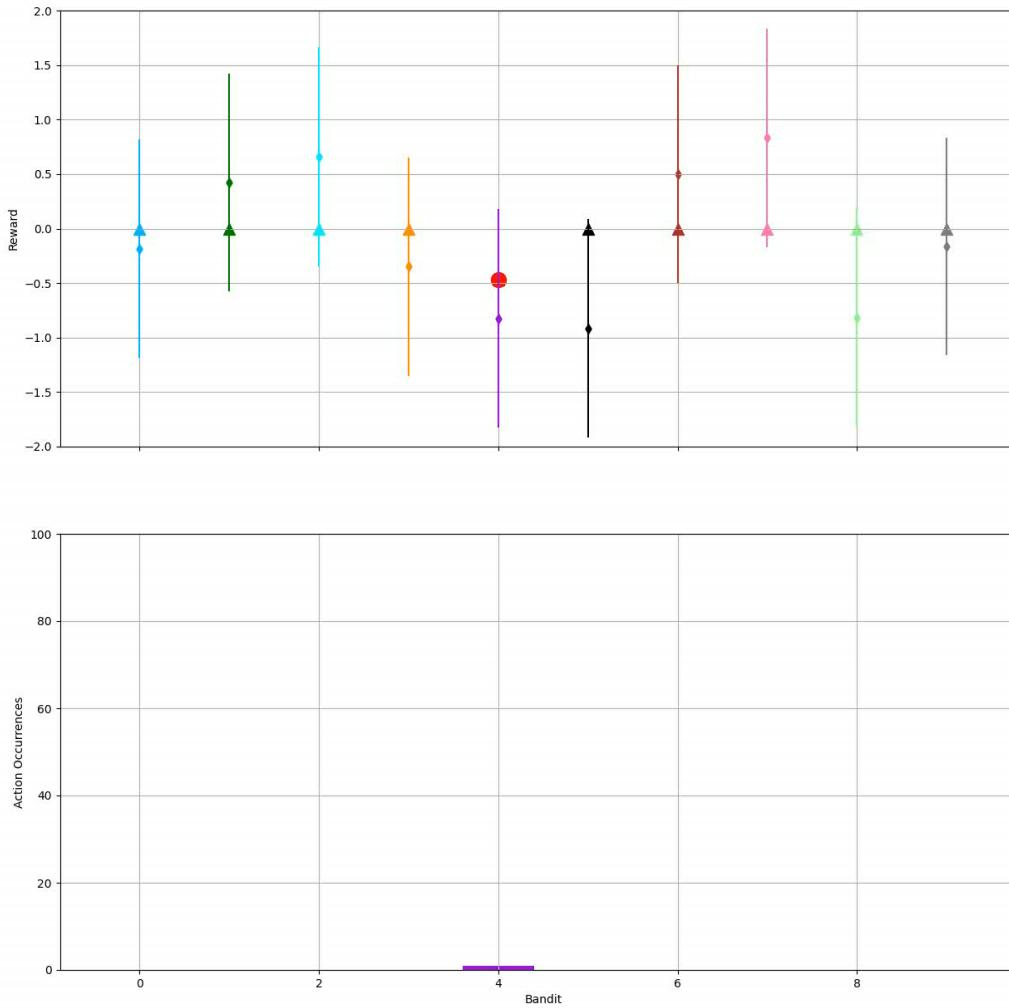


$Q_3(a)$       0.5      0.6      -0.1      -0.2      0.3



# K-Armed Bandit Problem

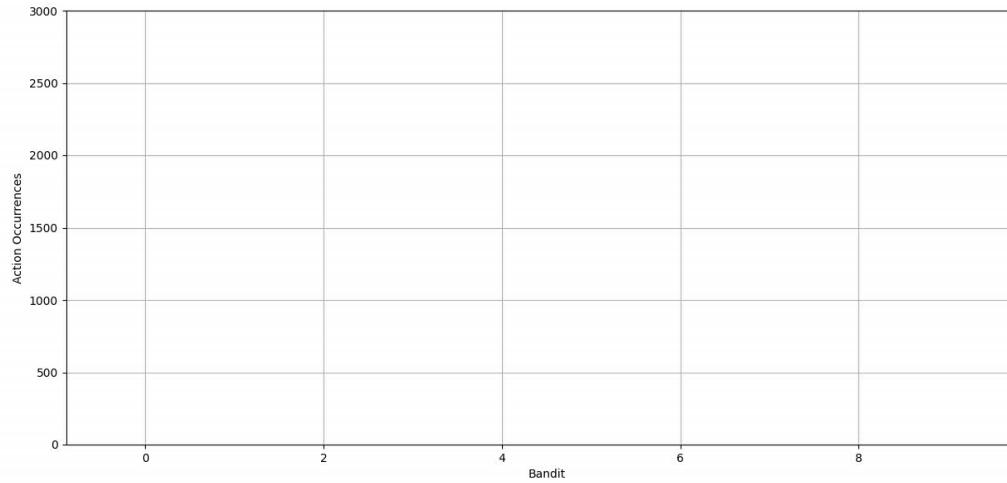
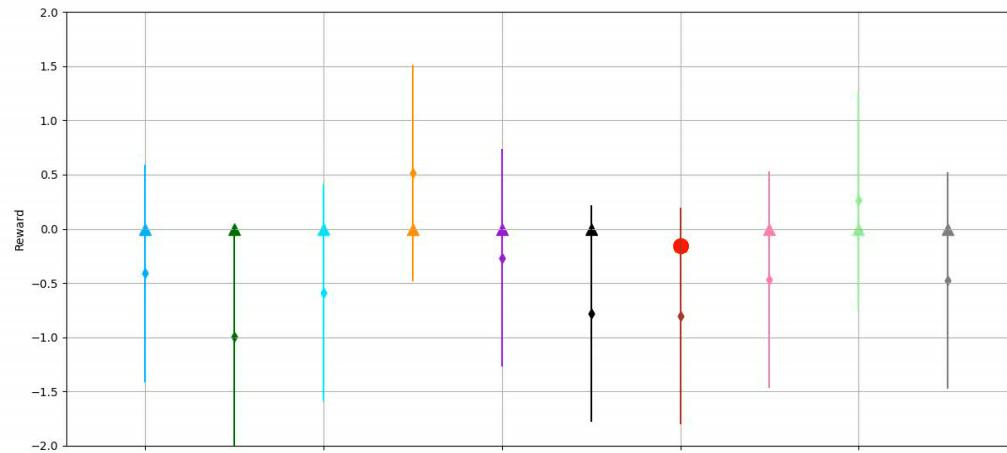
Estimated / Actual Action Values #0

**Greedy Acting:**

# K-Armed Bandit Problem

Estimated / Actual Action Values #0

Epsilon-Greedy (0.2):



# Greedy vs. Non-Greedy

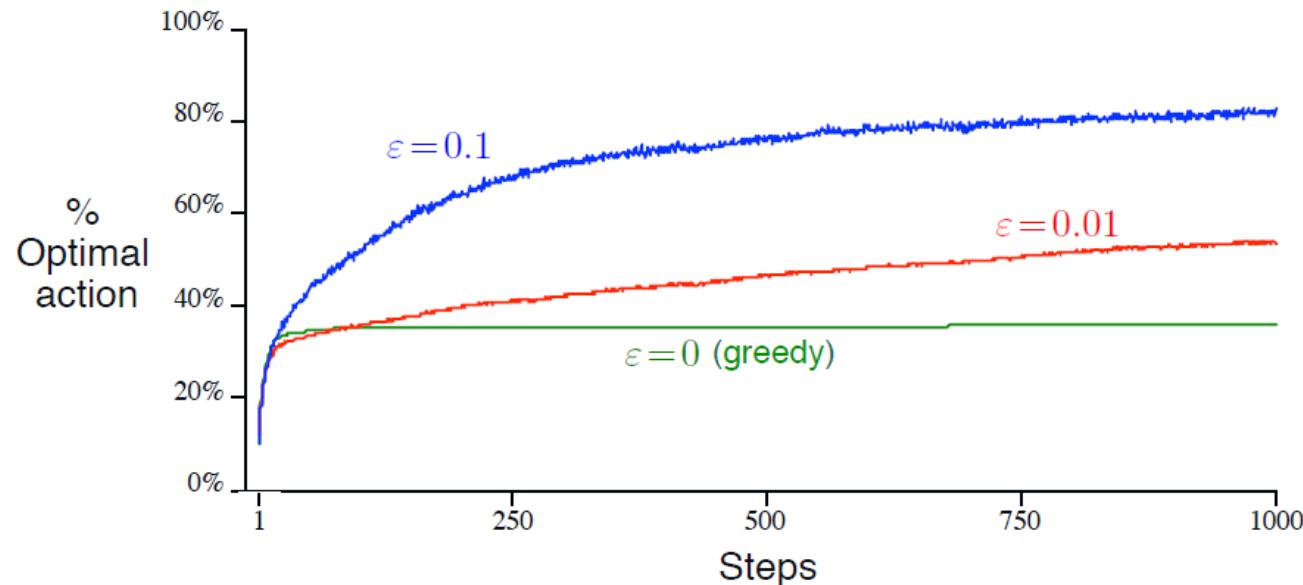
When maintaining estimates of the action values, at any time step there is at least one action whose estimated value is greatest. We call these the **greedy** actions. When you select one of these actions, we say that you are **exploiting** your current knowledge. If instead you select one of the **non-greedy** actions, then we say you are **exploring**.

## Greedy Acting:

$$A_t = \operatorname{argmax}_a Q_t(a).$$

## Epsilon-Greedy:

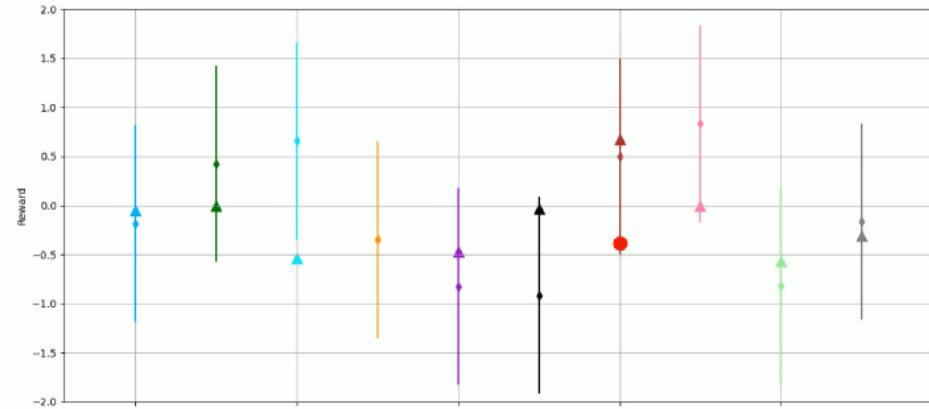
With small probability  $\epsilon$ , select randomly from among all the actions with equal probability.



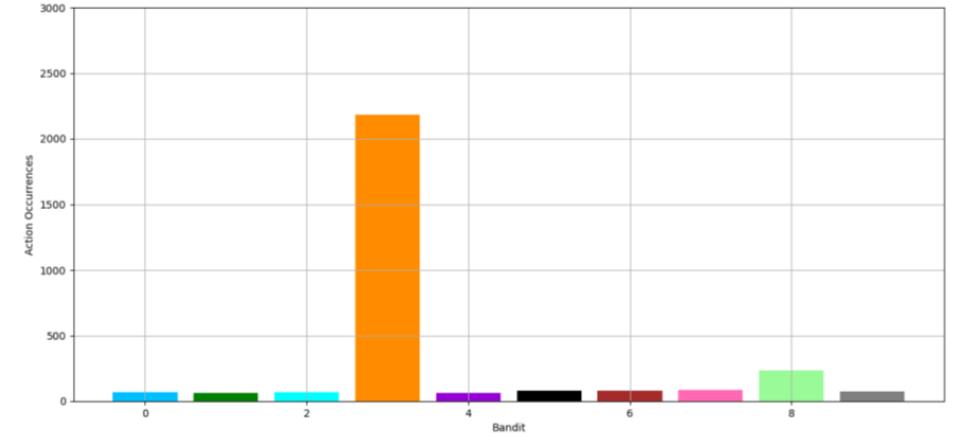
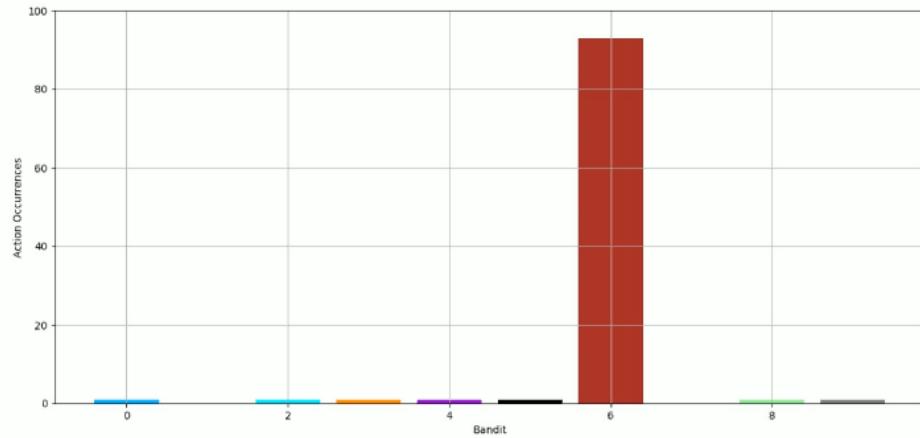
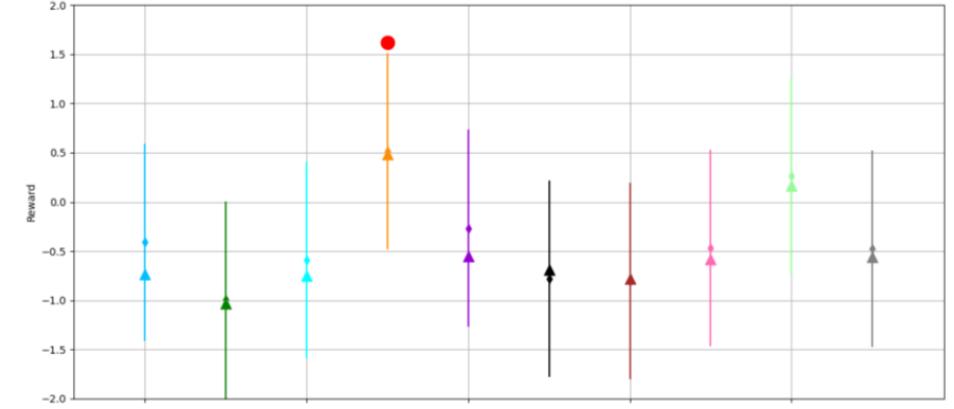
<http://www.andrew.cmu.edu/course/10-703/textbook/BartoSutton.pdf>

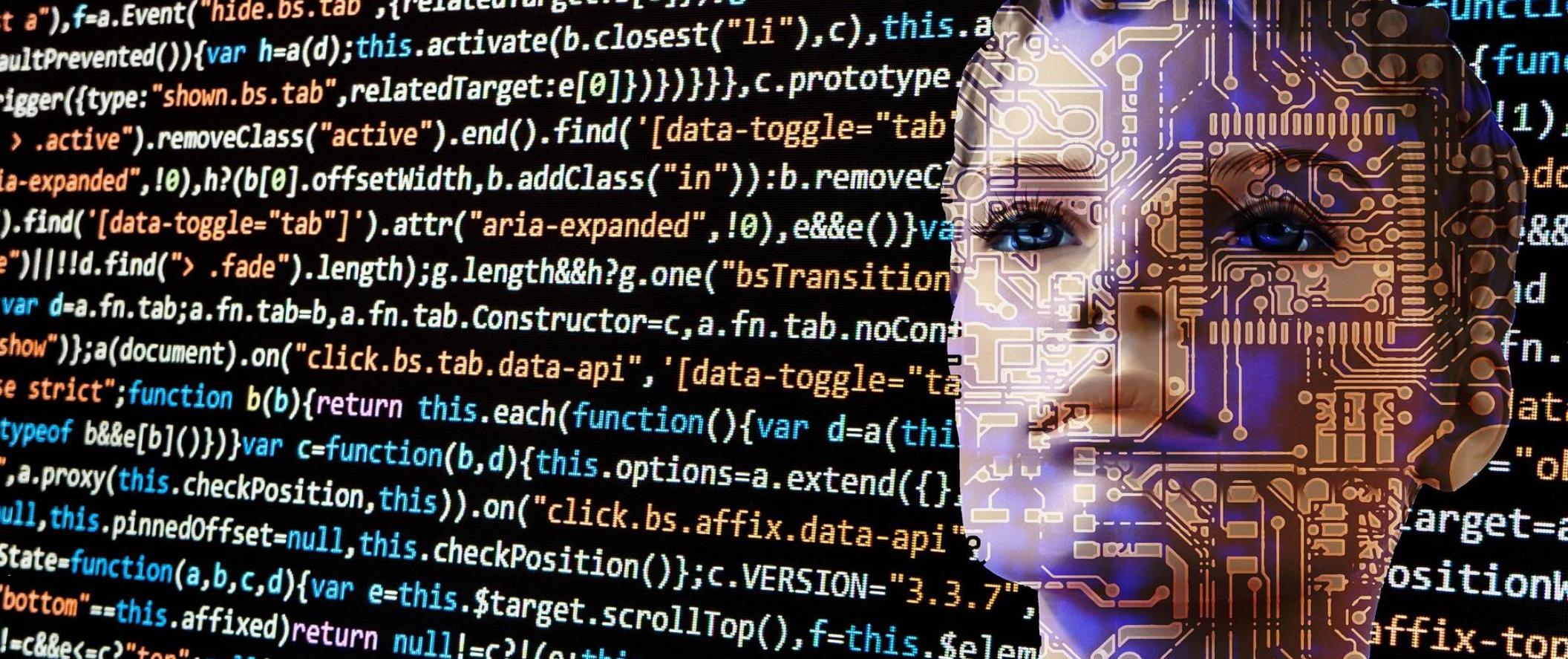
# Greedy vs. Non-Greedy

**Greedy Acting:**



**Epsilon-Greedy (0.2):**





## 6. Task 1: Grid World with Tabular Q-Learning

# Grid World

## Task:

Implement a Q-Learning algorithm to find an optimal policy in the Grid World environment described underneath. Utilize an epsilon-greedy policy.

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[ R_{t+1} + \gamma \max_{a' \in A} Q(S_{t+1}, a') - Q(S_t, A_t) \right] \quad (17)$$

	0	1	2	3	4
0					1
1		X			-1
2	Start	X		X	
3					

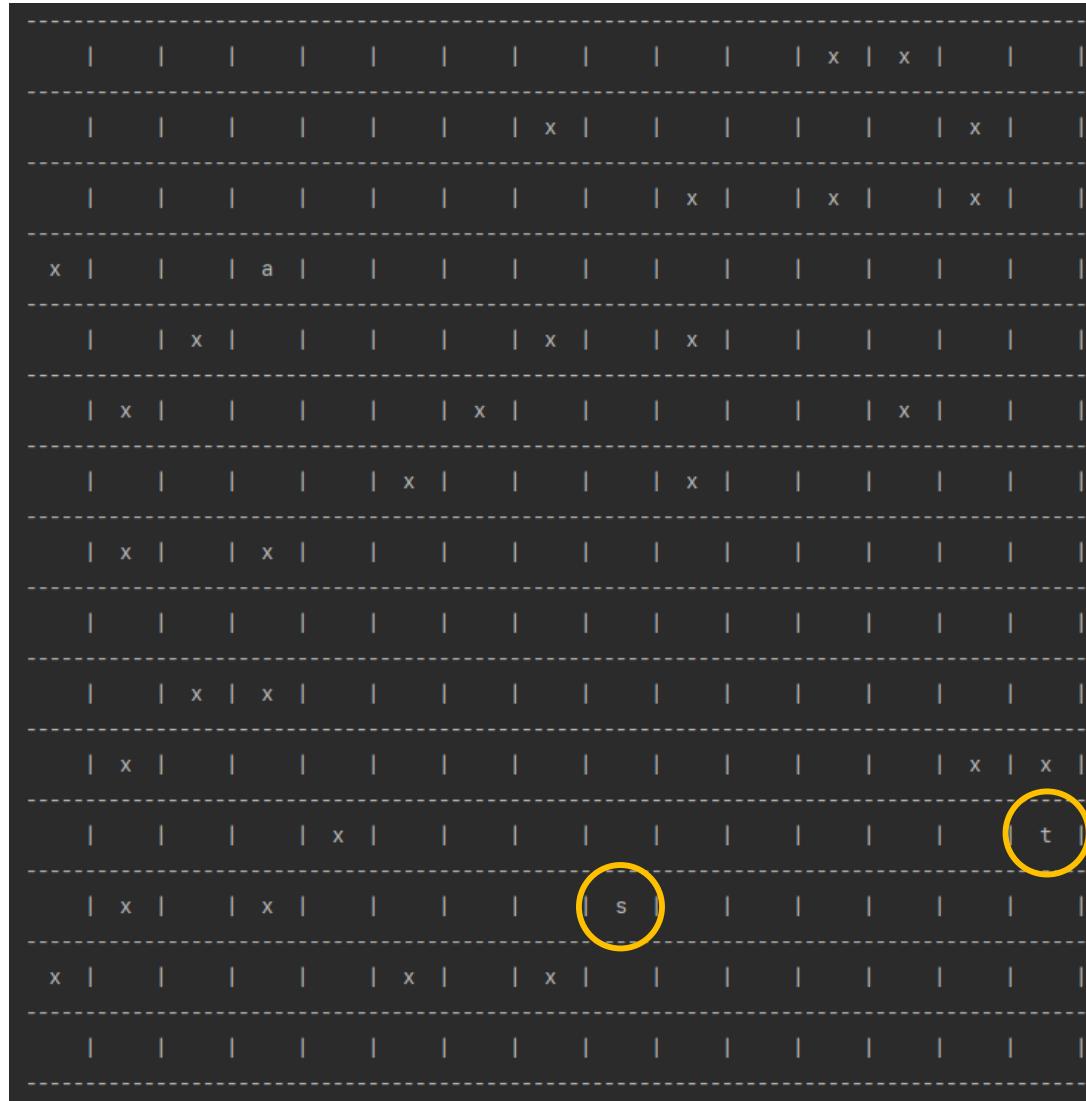
$$R_t = -0.1$$

on all other transitions

<http://www.andrew.cmu.edu/course/10-703/textbook/BartoSutton.pdf>

# Grid World

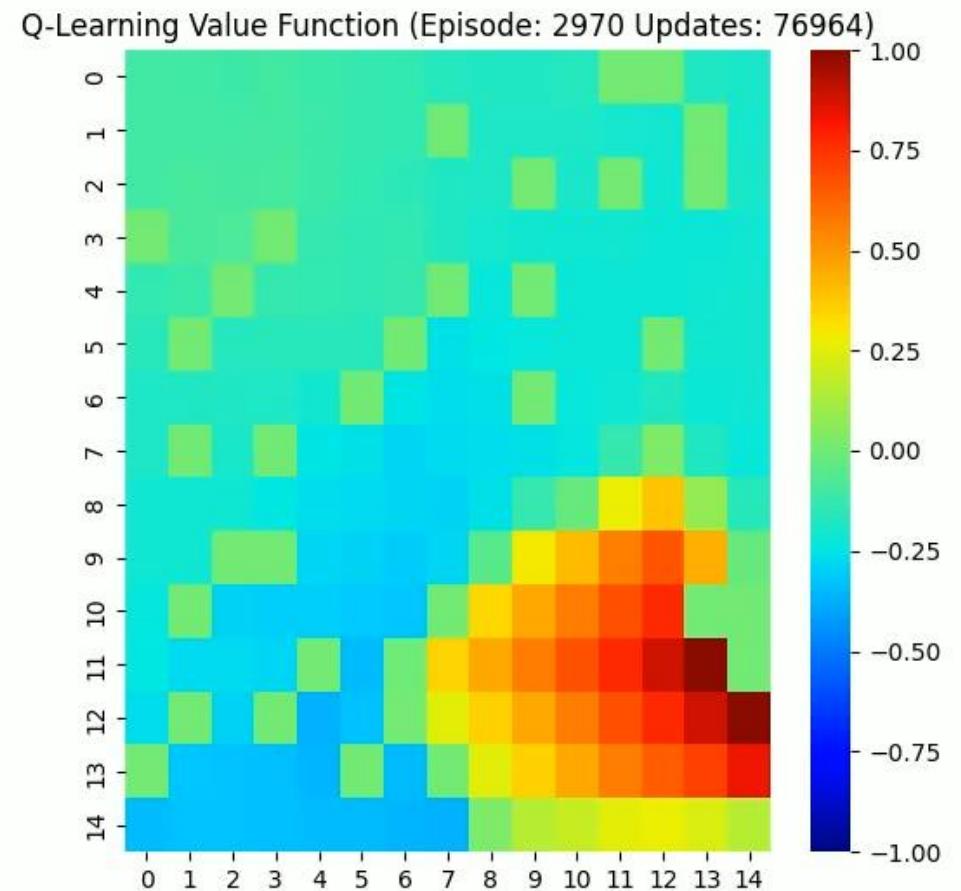
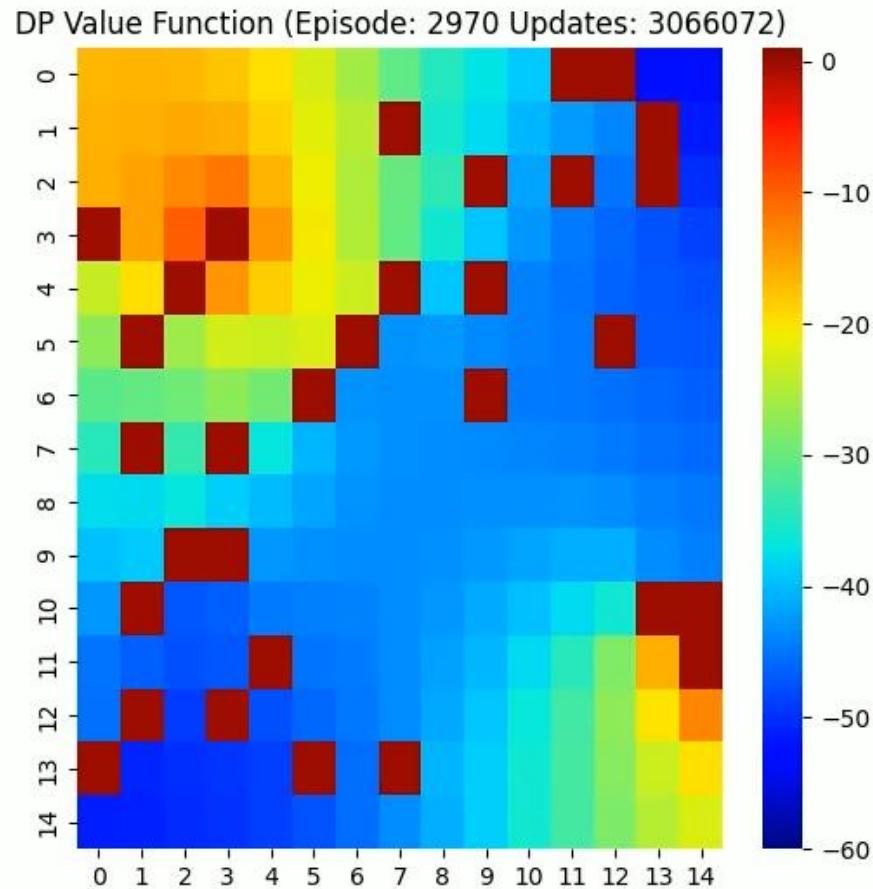
## Dynamic Programming vs Q-Learning

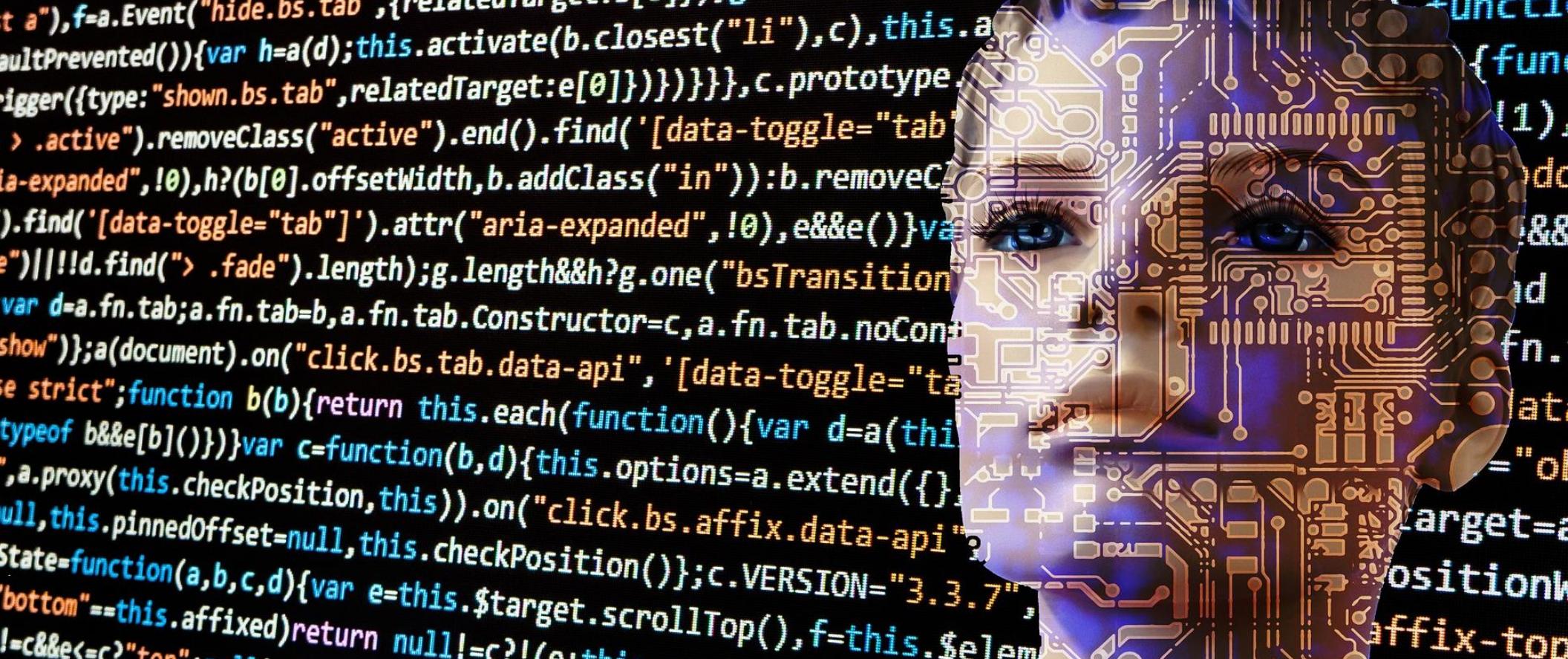


<http://www.andrew.cmu.edu/course/10-703/textbook/BartoSutton.pdf>

# Grid World

## Dynamic Programming vs Q-Learning

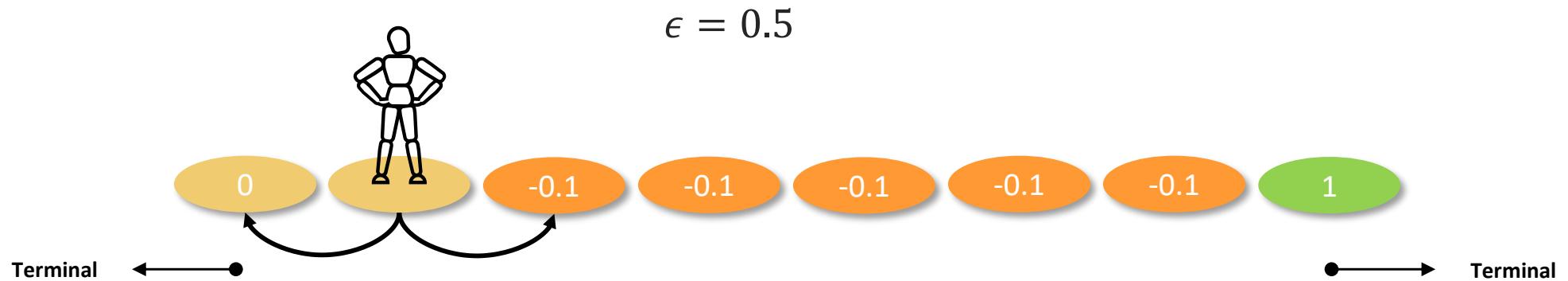




## 7. Task 2: Improved Exploration

# $\epsilon$ – Greedy

## Constraints



<https://i.ytimg.com/vi/VHqCAaFXpbc/maxresdefault.jpg>

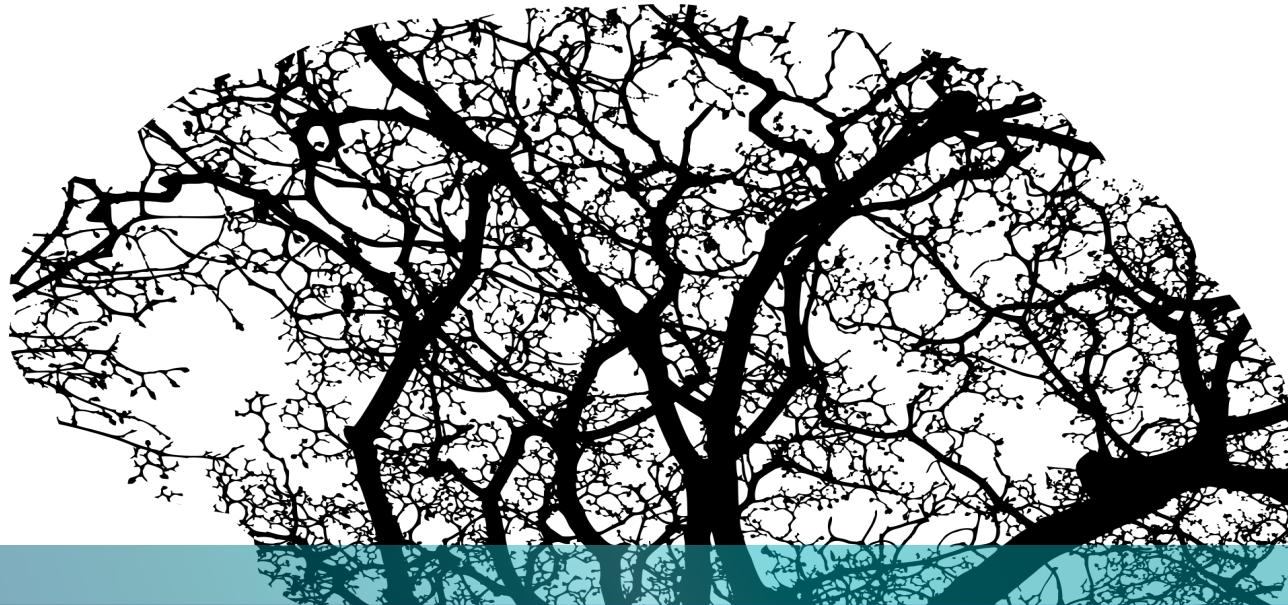
# Upper Confidence Bound (UCB)

Rather than performing exploration by simply selecting an **arbitrary action**, chosen with a probability that remains constant, the UCB algorithm **changes its exploration-exploitation balance** as it gathers more knowledge of the environment.

$$A_t = \operatorname{argmax}_a \left[ Q_t(a) + c \sqrt{\frac{\log t}{N_t(a)}} \right] \quad (20)$$

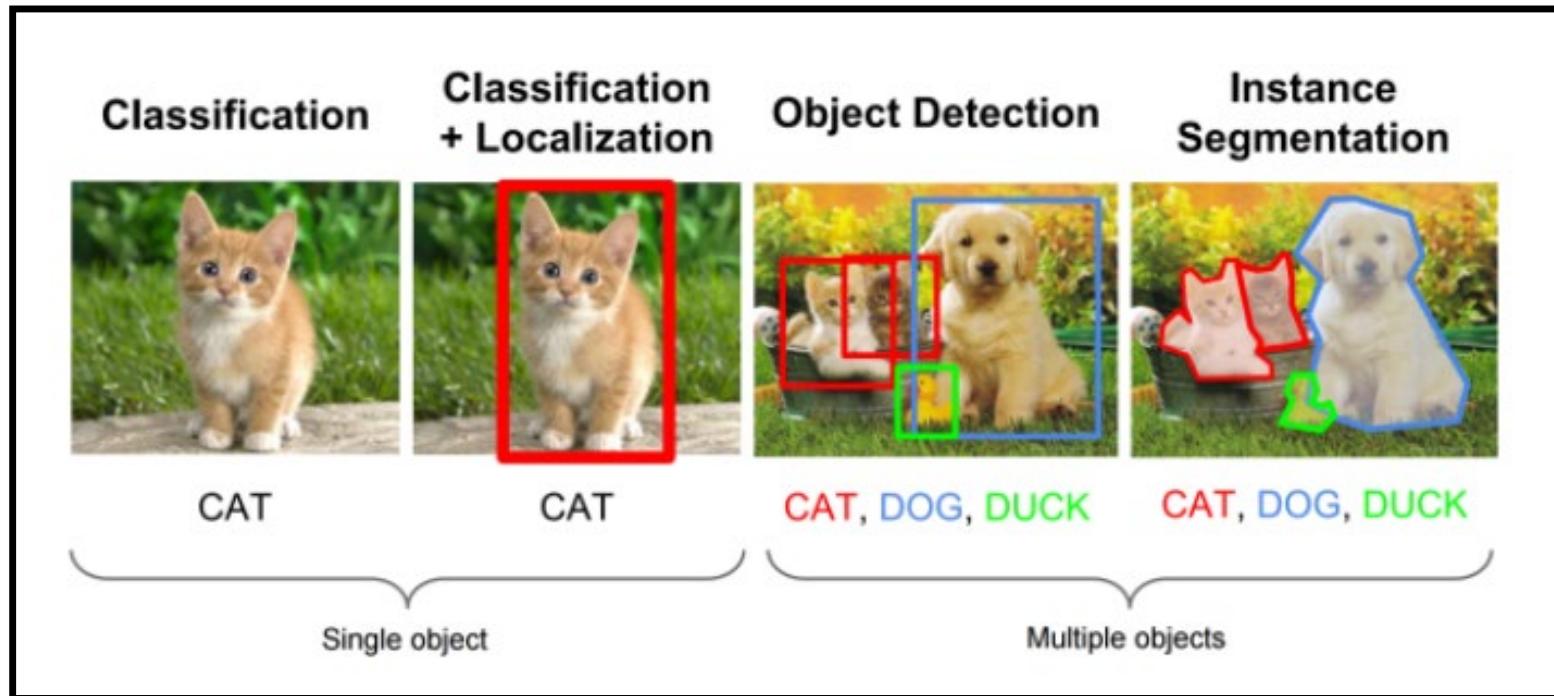
The diagram shows the UCB formula with two main components highlighted by ovals. The first component,  $Q_t(a)$ , is highlighted with a green oval and labeled "Exploitation" below it. The second component,  $c \sqrt{\frac{\log t}{N_t(a)}}$ , is highlighted with a red oval and labeled "Exploration" below it.

$Q_t(a)$	Estimated Action Value
$N_t(a)$	Number of times action $a$ has been selected before
$c$	Confidence value controlling the level of exploration
$t$	Number of times state $s$ has been visited before



› **AUTOSYS – DEEP LEARNING**  
**SSD – Single Shot Detector**

# WHAT IS THE DIFFERENCE BETWEEN: *CLASSIFICATION | LOCALIZATION | DETECTION | SEGMENTATION ?*



[SOURCE] [https://leonardoaraujosantos.gitbooks.io/artificial-intelligence/content/object\\_localization\\_and\\_detection.html](https://leonardoaraujosantos.gitbooks.io/artificial-intelligence/content/object_localization_and_detection.html)

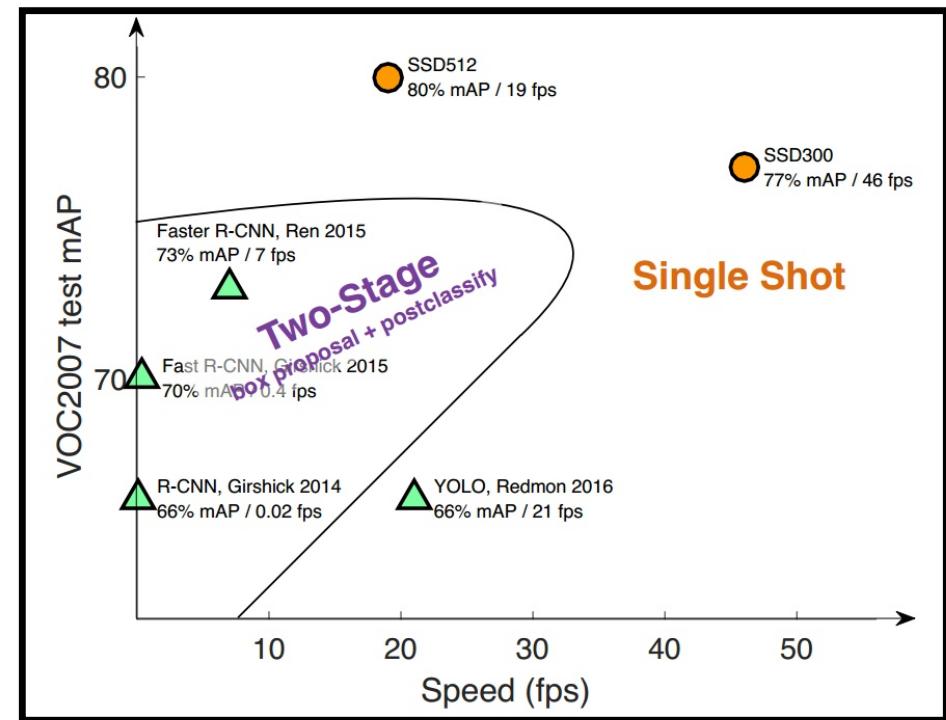
# DIFFERENT APPROACHES IN THE LAST FEW YEARS ...

## SINGLE SHOT DETECTOR

One Forward pass for both “Box Proposal” and “Classification”.

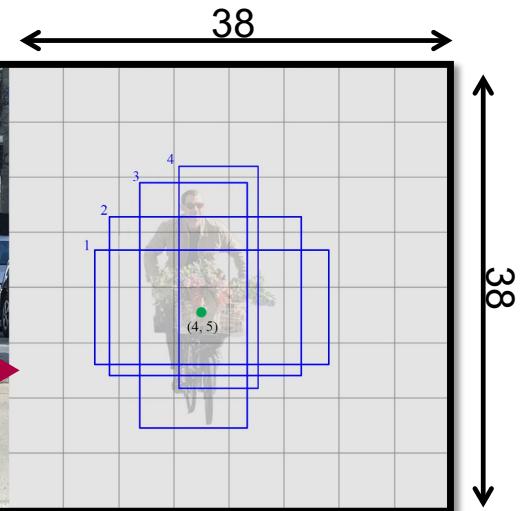
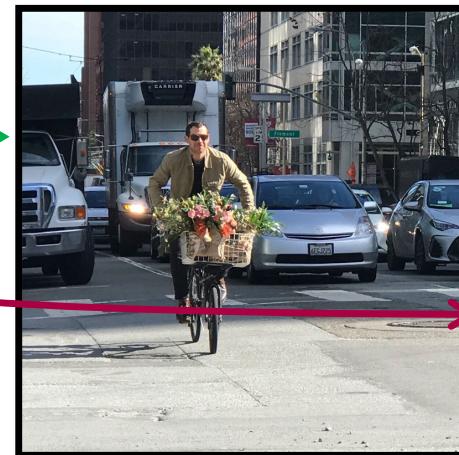
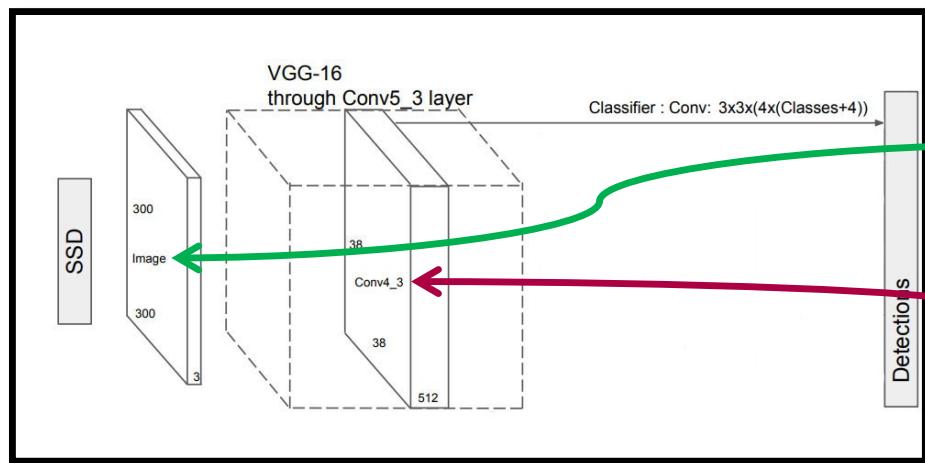
## TWO STAGE DETECTOR

Two forward passes for both “Box Proposal” and “Classification”



[SOURCE] <https://zhuanlan.zhihu.com/p/30478644>

# SSD – HIGH LEVEL VIEW



$38 \times 38 \times 4$  predictions

[SOURCE] [https://medium.com/@jonathan\\_hui/ssd-object-detection-single-shot-multibox-detector-for-real-time-processing-9bd8deac0e06](https://medium.com/@jonathan_hui/ssd-object-detection-single-shot-multibox-detector-for-real-time-processing-9bd8deac0e06)

# HOW TO LOCALIZE OBJECTS WITH REGRESSION?

---

- With regression we can adjust/update numbers rather than finding the correct class.
- To define a Bounding Box fully we need 4 Numbers:

**X, Y, Width, Height**

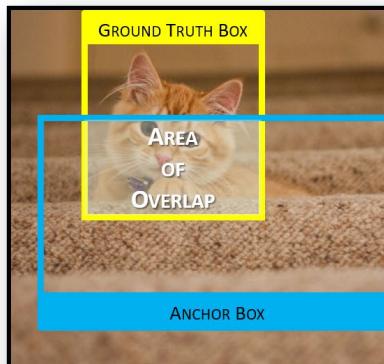
- We need a function to calculate the loss between predicted bounding box and ground truth.

**Smooth L1 / Jaccard Index**

- Now we can update the weights to get closer to the ground truth bounding box.

# UNDERSTANDING IOU

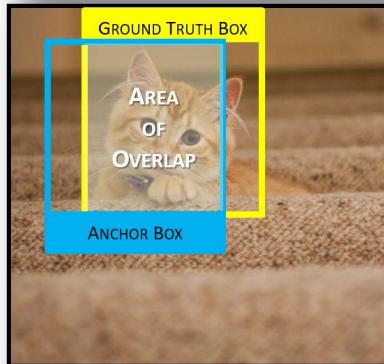
## INTERSECTION OVER UNION (JACCARD INDEX)



$$\text{Area GTB} = 5 \text{ cm} \cdot 5 \text{ cm} = 25$$
$$\text{Area AB} = 10 \text{ cm} \cdot 5 \text{ cm} = 50$$

$$\text{Area of Overlap} = 12.5$$
$$\text{Area of Union} = 62.5$$

$$IoU = \frac{12.5}{62.5} = 0.2 \rightarrow \text{Negative}$$



$$\text{Area GTB} = 5 \text{ cm} \cdot 5 \text{ cm} = 25$$
$$\text{Area AB} = 5 \text{ cm} \cdot 5 \text{ cm} = 25$$

$$\text{Area of Overlap} = 20$$
$$\text{Area of Union} = 30$$

$$IoU = \frac{20}{30} = 0.66 \rightarrow \text{Positive}$$

$$IoU = \frac{\text{Area of Overlap}}{\text{Area of Union}}$$



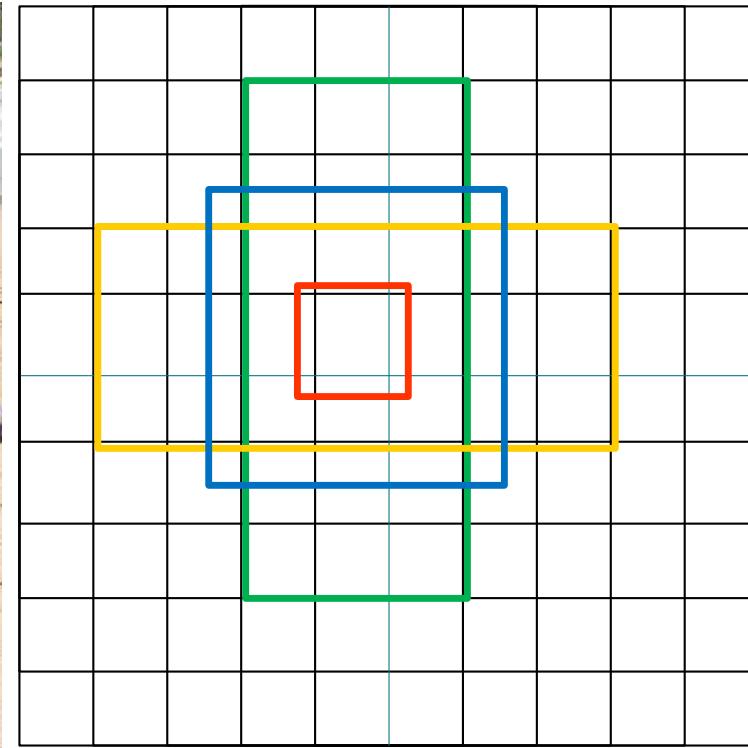
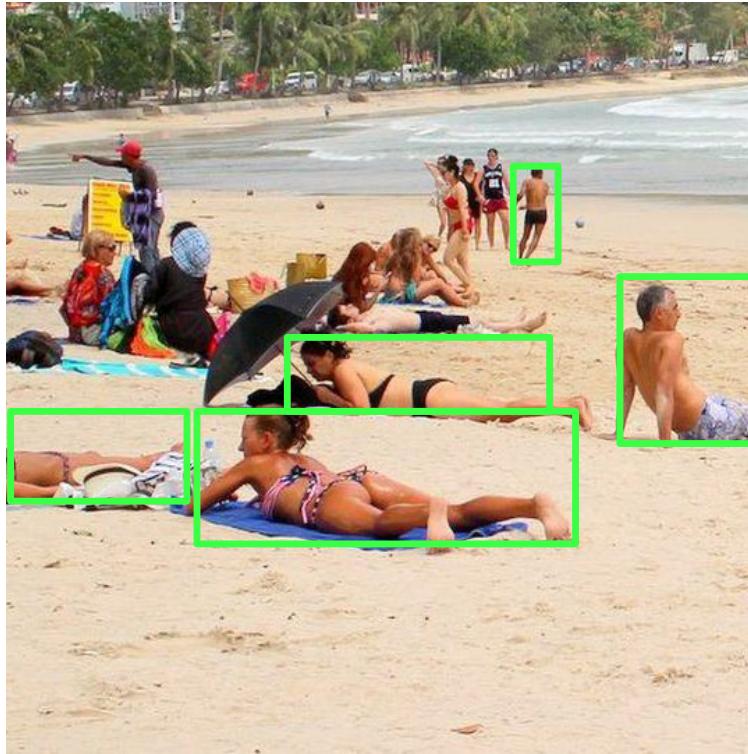
**Positive IoU Threshold**  
**[0.5, 1.0]**  
**Ignored by the network**  
**(0.2, 0.5)**  
**Negative IoU Threshold**  
**[0, 0.2]**

# USING DEFAULT BOXES (ALSO KNOW AS PRIORS OR ANCHOR BOXES)

---

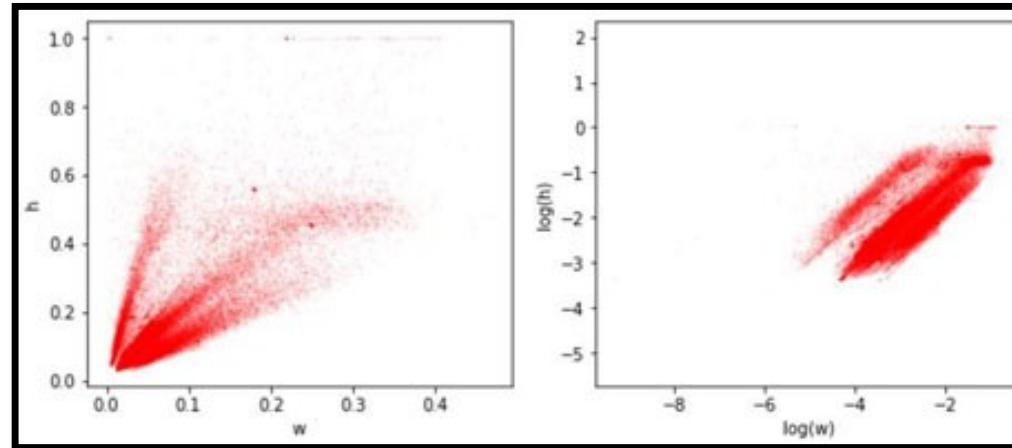
- **What is it?** Default Boxes are a collection of boxes overlaid on the image at different spatial **locations, scales** and **aspect ratios** that act as reference points on the ground truth bounding boxes
- **Why do we need it?** We could start with random predictions and use gradient descent to optimize the model. However, during the initial training, the model may fight with each other to determine what shapes to be optimized for which predictions. → No Convergence of the model, slow training, ...
- A model is then trained to make two predictions for each default box:
  - A discrete class prediction for each default box
  - A continuous prediction of an offset by which the anchor needs to be shifted to fit the ground-truth bounding box

# WHAT KIND OF DEFAULT BOXES ARE SENSIBLE?



# ARE PRE-DEFINED ASPECT RATIOS LIMITING?

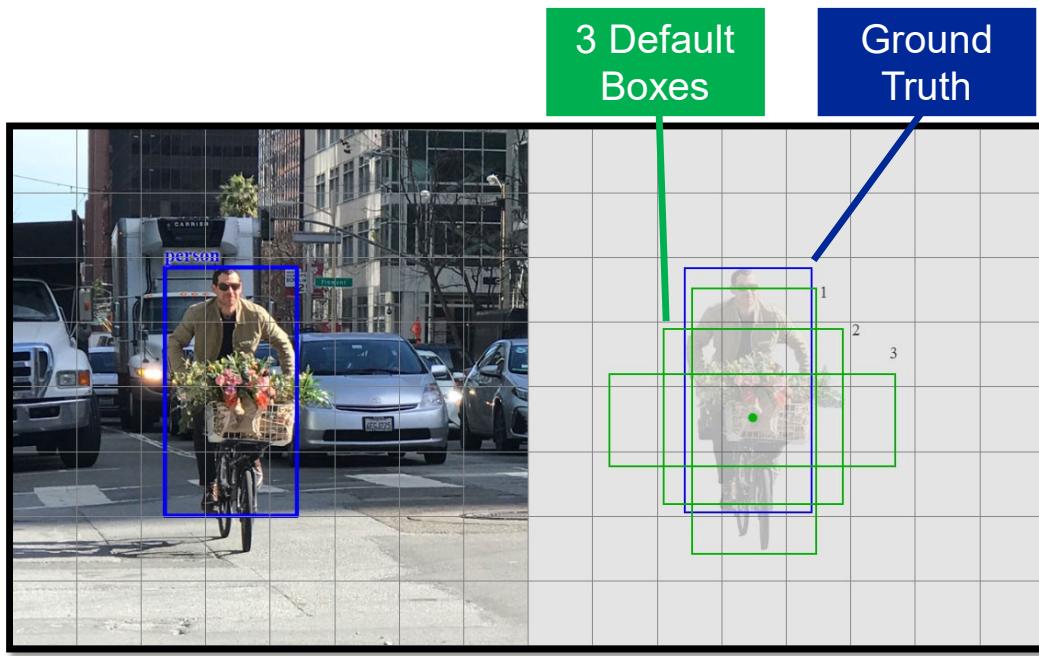
No, because **boundary boxes do not have arbitrary shape and size.**



The graph above shows the width and height distributions for the bounding boxes of the KITTI Dataset. Those distributions are highly clustered.

[SOURCE] [https://medium.com/@jonathan\\_hui/ssd-object-detection-single-shot-multibox-detector-for-real-time-processing-9bd8deac0e06](https://medium.com/@jonathan_hui/ssd-object-detection-single-shot-multibox-detector-for-real-time-processing-9bd8deac0e06)

# MATCHING STRATEGY

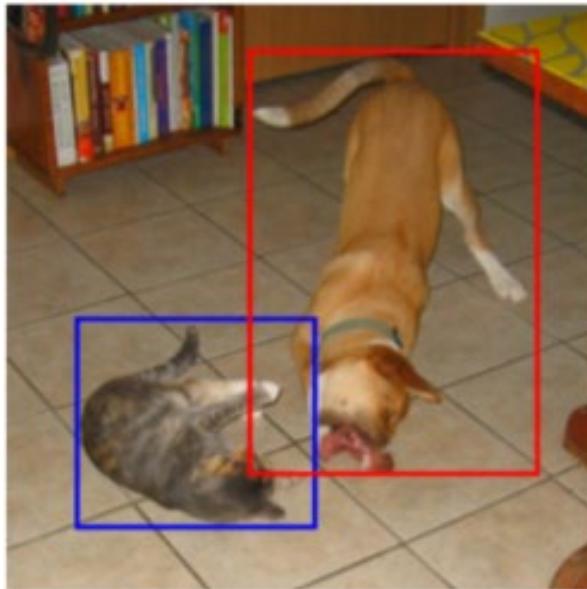


- The cost for the boundary mismatch is only calculated when the IoU is greater than 0.5
- In this case we simplify the case and only have 3 *Default Boxes*.
- Default Box 3 is discarded because the IoU value is lower than 0.5
- Default Box 1 and 2 have a IoU value greater than 0.5 → Calculate the localization cost of the corresponding predicted bounding box → Take the best.

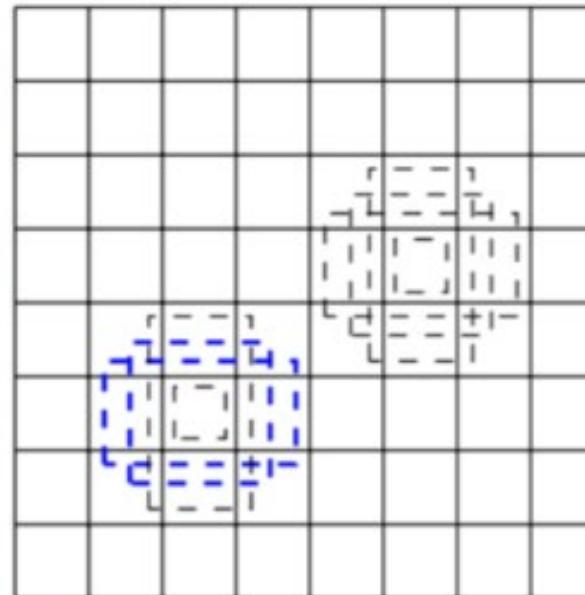
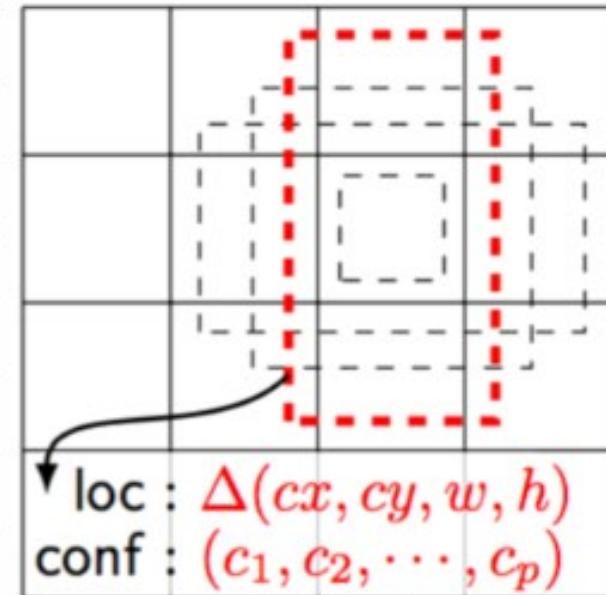
**THIS MATCHING STRATEGY ENCOURAGES EACH PREDICTION TO PREDICT SHAPES CLOSER TO THE CORRESPONDING DEFAULT BOX. THEREFORE OUR PREDICTIONS ARE MORE DIVERSE AND MORE STABLE IN THE TRAINING.**

[SOURCE] [https://medium.com/@jonathan\\_hui/ssd-object-detection-single-shot-multibox-detector-for-real-time-processing-9bd8deac0e06](https://medium.com/@jonathan_hui/ssd-object-detection-single-shot-multibox-detector-for-real-time-processing-9bd8deac0e06)

# HOW TO BE SCALE INVARIANT?



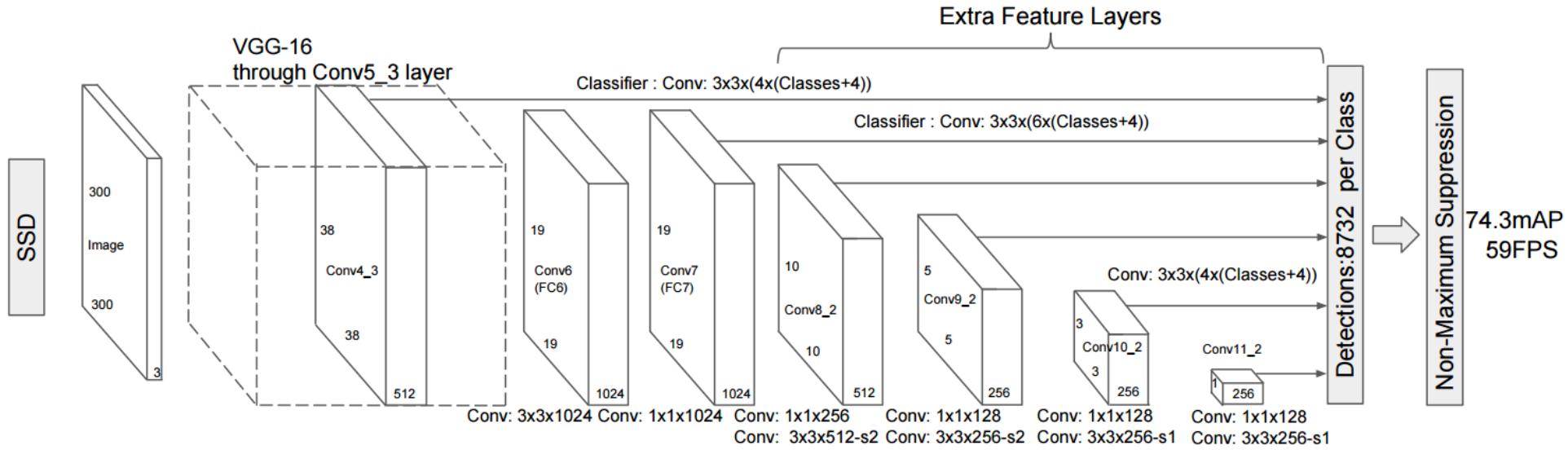
(a) Image with GT boxes

(b)  $8 \times 8$  feature map

loc :  $\Delta(cx, cy, w, h)$   
conf :  $(c_1, c_2, \dots, c_p)$

(c)  $4 \times 4$  feature map[SOURCE] <https://arxiv.org/pdf/1512.02325.pdf>

# HOW TO BE SCALE INVARIANT?

[SOURCE] <https://arxiv.org/pdf/1512.02325.pdf>

# LOSS FUNCTION

The **localization loss** is the mismatch between the ground truth box and the predicted boundary box. SSD only penalizes predictions from positive matches. We want the predictions from the positive matches to get closer to the ground truth. Negative matches can be ignored.

The localization loss between the predicted box  $l$  and the ground truth box  $g$  is defined as the smooth L1 loss with  $cx, cy$  as the offset to the default bounding box  $d$  of width  $w$  and height  $h$ .

$$L_{loc}(x, l, g) = \sum_{i \in Pos}^N \sum_{m \in \{cx, cy, w, h\}} x_{ij}^k \text{smooth}_{L1}(l_i^m - \hat{g}_j^m)$$

$$\hat{g}_j^{cx} = (g_j^{cx} - d_i^{cx})/d_i^w \quad \hat{g}_j^{cy} = (g_j^{cy} - d_i^{cy})/d_i^h$$

$$\hat{g}_j^w = \log \left( \frac{g_j^w}{d_i^w} \right) \quad \hat{g}_j^h = \log \left( \frac{g_j^h}{d_i^h} \right)$$

$$x_{ij}^p = \begin{cases} 1 & \text{if } IoU > 0.5 \text{ between default box } i \text{ and ground true box } j \text{ on class } p \\ 0 & \text{otherwise} \end{cases}$$

[SOURCE] <https://arxiv.org/pdf/1512.02325.pdf>

# LOSS FUNCTION

The **confidence loss** is the loss in making a class prediction. For every positive match prediction, we penalize the loss according to the confidence score of the corresponding class. For negative match predictions, we penalize the loss according to the confidence score of the class “0”: class “0” classifies no object is detected.

It is calculated as the softmax loss over multiple classes confidences  $c$  (class score).

$$L_{conf}(x, c) = - \sum_{i \in Pos}^N x_{ij}^p \log(\hat{c}_i^p) - \sum_{i \in Neg} \log(\hat{c}_i^0) \quad \text{where} \quad \hat{c}_i^p = \frac{\exp(c_i^p)}{\sum_p \exp(c_i^p)}$$

where  $N$  is the number of matched default boxes.

The **final loss function** looks like that:

$$L(x, c, l, g) = \frac{1}{N} (L_{conf}(x, c) + \alpha L_{loc}(x, l, g))$$

[SOURCE] <https://arxiv.org/pdf/1512.02325.pdf>

# HARD NEGATIVE MINING

---

## PROBLEM

- We make far more predictions than the number of objects present in the picture.
- This means we have much more negative matches than positive matches.
- This creates an imbalance → Bad for Training

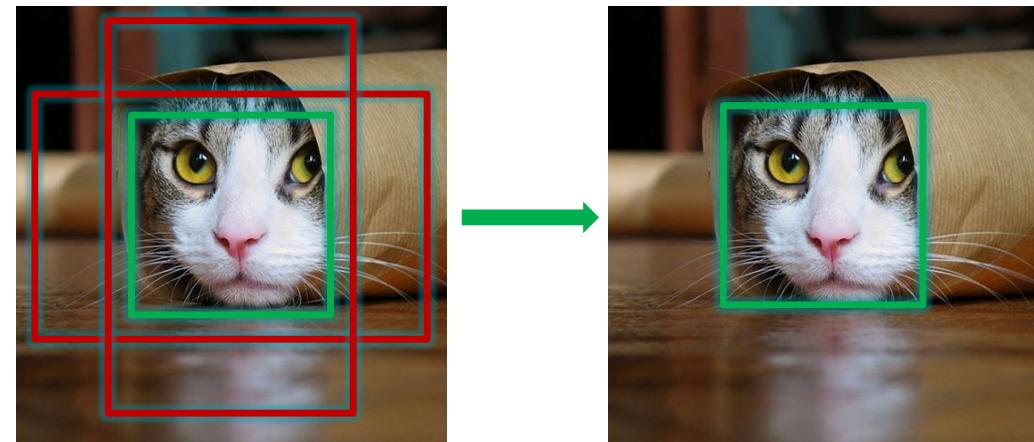
## SOLUTION – HARD NEGATIVE MINING

- Sort negative matches by their confidence loss
- Pick the negative matches with the top loss
- Make sure the ratio between negative and positive matches is not bigger than 3:1

# NON-MAXIMUM SUPPRESSION

SSD uses **non-maximum suppression** to remove duplicate predictions pointing to the same object.

- SSD sorts the predictions by the confidence scores.
- Start from the top confidence prediction, SSD evaluates whether any previously predicted boundary boxes have an IoU higher than 0.45 with the current prediction for the same class.
- If found, the current prediction will be ignored.
- Deeper Understanding: <https://towardsdatascience.com/non-maximum-suppression-nms-93ce178e177c>



# A QUICK WORD ON *DATA AUGMENTATION*

Data augmentation is **important** in improving accuracy. Augment data with flipping, cropping and color distortion.

data augmentation	SSD300		
horizontal flip	✓	✓	✓
random crop & color distortion		✓	✓
random expansion			✓
VOC2007 test mAP	65.5	74.3	<b>77.2</b>



[SOURCE] [https://medium.com/@jonathan\\_hui/ssd-object-detection-single-shot-multibox-detector-for-real-time-processing-9bd8deac0e06](https://medium.com/@jonathan_hui/ssd-object-detection-single-shot-multibox-detector-for-real-time-processing-9bd8deac0e06)

# A QUICK WORD ON *INFERENCE TIME*

SSD300 makes many predictions (8732) for a better coverage of location, scale and aspect ratios, more than many other detection methods.

Method	mAP	FPS	batch size	# Boxes	Input resolution
Faster R-CNN (VGG16)	73.2	7	1	~ 6000	~ 1000 × 600
Fast YOLO	52.7	155	1	98	448 × 448
YOLO (VGG16)	66.4	21	1	98	448 × 448
SSD300	74.3	46	1	8732	300 × 300
SSD512	76.8	19	1	24564	512 × 512
SSD300	74.3	59	8	8732	300 × 300
SSD512	76.8	22	8	24564	512 × 512

[SOURCE] [https://medium.com/@jonathan\\_hui/ssd-object-detection-single-shot-multibox-detector-for-real-time-processing-9bd8deac0e06](https://medium.com/@jonathan_hui/ssd-object-detection-single-shot-multibox-detector-for-real-time-processing-9bd8deac0e06)

# TIPPS & TRICKS

---

- Accuracy increases with the number of default boundary boxes at the cost of speed.
- Adjust aspect ratios of your default bounding boxes in accordance to your dataset.
- If you have smaller objects to detect, it is necessary to increase resolution, as well at the cost of speed

[SOURCE] [https://medium.com/@jonathan\\_hui/ssd-object-detection-single-shot-multibox-detector-for-real-time-processing-9bd8deac0e06](https://medium.com/@jonathan_hui/ssd-object-detection-single-shot-multibox-detector-for-real-time-processing-9bd8deac0e06)