

Good luck!

# cLecture: Artificial Intelligence of Automotive Engineering

## Chapter I: Introduction

.AI:

Types of intelligence:

Ways of thinking and acting:

- Artificial Intelligence → Goal: Programming a development machine **that** behaves as if it were intelligent

- Emotional, Creative, Methodical and Analytical Intelligence

- Mindsets:

- Rational (laws, logic, ...)
- Human (thoughts, human actions, cognitive science)

- Behaviors:

- Rational (Automatic Action, Adaptive, Create, Acting Agents,...)
- Human (Turing test. Natural languages, ...)

Touring test:

- Test: a human and a machine write texts to each other a test person has → to choose which of the two is **the machine** if the → test person chooses incorrectly, the test for the machine has been passed

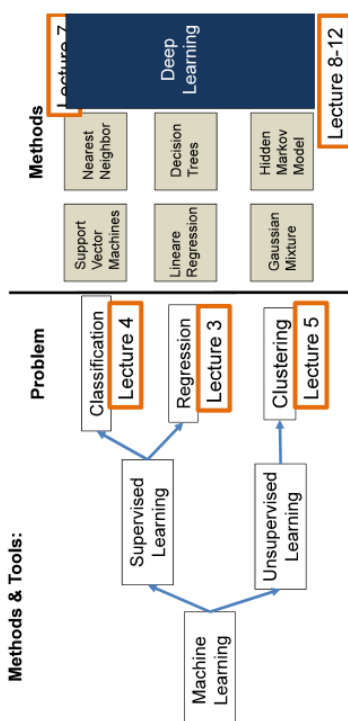
- Advantage: Test is simple and objective → Tests only the behavior of **the** machine, not the logical methods or neural methods

Cognitive Science:

- Is the interdisciplinary scientific study of the mind **and** its processes → It looks at intelligence and behavior with a focus on the nervous system
- Core competencies: language, perception, memory, attention, argumentation, emotion
- Associated with: Languages, Psychology, Artificial Intelligence, Philosophy, Anthropology

Methods of Artificial Intelligence in 9 subcategories:

1. **Logical thinking and problem solving** (Reasoning & Problem Solving):  
→ Machine gets the ability to grasp the problem piece by piece and to create logical reduction with room for manoeuvre  
→ Objective/problem descriptioning:
  - A given problem or task is to be solved
  - A machine can perceive and examine the problem piecemeal to solve the task
  - Can use formal logic to solve
  - Integration of uncertainties and probabilities  
→ Methods:
  - Intelligent search through many possible solutions (e.g. Tree Search, Dijkstra, Kruskal, Nearest Neighbour, A\*-Search)
  - Optimization → Minimization/maximization of a given problem by means of boundaries (e.g. linear or quadratic programming, heuristics, ...)  
Evolutionary computation: optimization based on adaptation search (e.g. genetic algorithms, particle swarm optimization, ant colony optimization, ...)
2. **Knowledge Representation**  
→ Represents information about the world so that a Computer can solve complex tasks  
→ Objective/problem descriptioning:
  - a computer was a standalone (autonomous) "agent"
  - The goal is to provide him with all the necessary environmental information.
  - Procedure: Creation of knowledge-based systems for a knowledge-based database
  - Knowledge is implemented with facts and rules of the environment  
→ Methods:
  - Logic: A combination of rules and sentences in logical form (e.g. propositional logic, first order logic, fuzzy logic, ...)
3. **Planning:**  
→ the machine is given the opportunity to optimize automatic planning or controlled task allocation  
→ Objective/problem descriptioning:
  - a computer acts as a standalone (autonomous) agent that acts automatically, sets goals, and achieves them
  - The task of the programmer is to add the environment and (his) future to him.
  - The agent must make its own decisions and maximize its benefits using uncertainties  
→ Methods:
  - Intelligent search through many possible solutions (e.g. Tree Search, Dijkstra, Kruskal, Nearest Neighbour, A\*-Search)
  - Agent systems: A computer works for a user or other system/agents (e.g. multi-agents, intelligent agents)
  - Evolutionary calculation: Optimized search based on adaptations (e.g. genetic algorithms, particle swarm, ant colony, ...)
  - Uncertified selection (uncertainty reasoning): action with incomplete information (e.g. Bayesian Network, Hidden Markov Model, Kalman Filter)
4. **Learning: the**  
→ machine acquires the ability to learn (based on an algorithm **that** automatically improves through experience and data without the manual execution of programs) → Machine Learning (ML)  
→ Objective/problem descriptioning:



Good luck!

- 
- The computer receives a lot of data and can process it using an algorithm
  - This algorithm gives the computer the ability to recognize patterns.
  - The computer learns from the data (→ machine learning) and can make predictions based on the data, so it does not follow a strict static program.
- Methods:
- Supervised learning: Computer receives input and Outputs and is intended to generate a calculation rule for them (e.g. regression, classification)
  - Unsupervised learning: The learning algorithm receives no regulations (label) and is left to itself in the search for a structure in Input values (e.g. clustering)
5. Natural Language Processing (NLP):
- The machine acquires the ability to read and understand human language
- Objective/problem description:
- Human languages are highly complex due to their syntax (grammar), semantics (meaning) and pragmatics (purpose).
  - Computer gets the ability to understand **these** and handwritten
- Methods:
- Logic: A combination of rules and sentences in logical form (e.g. propositional logic, first order logic, recognition-based logic, ...)
  - Classical machine learning (e.g. classification)
  - Deep Learning (e.g. LSTM network)
6. Perception:
- The machine is given the opportunity to perceive the environment by means of sensors and to create an image of it
- Objective/problem description:
- Computer is given the opportunity to perceive the environment as an agent and uses sensors as input (camera, lidar, ultrasonic, radar, microphones)
  - Machine perception: possibility to interpret the environmental data
  - Computer vision: The input of a camera is analyzed and the information is extracted
- Methods:
- Computer Vision classic (e.g. color extraction, Canny Edge, Hough lines, ...)
  - New Computer Vision (e.g. Deep Neuronal Networks, Recurrent Neuronal Networks)
7. Motion and Manipulation:
- A machine is given the ability to learn and plan how to use its movement efficiently
- Objective/problem description:
- The agent acquires the ability to move. This movement and its behavior must be planned by the developer. The type of movement must be chosen, the type of communication with the environment must be planned by the developer, and actuators must be controlled.
- Methods:
- Behavior planning (What should I do=) (e.g. logic-based (state engine), knowledge-based (Network-Graph))
  - Motion planning (how can I achieve something) (e.g. search algorithms, optimization algorithms)
  - Control (steering and control of all actuators) (e.g. classical control by PID controller, model predictive control, ...)
8. **Social intelligence:**
- A machine is given the ability to understand, interpret, process and mimic human V
- Objective/problem description:
- The agent can understand and imitate social skills (e.g. respect, sociability, ...)
  - The agent can perform emotional calculations (recognition, interpretation and human reactions process and simulate)
  - The agent can perform speech, face, body gesture recognition
- Methods:
- Database (e.g. logic-based, knowledge-based)
  - Classification (What emotion could this be?) (e.g. Support Vector Machines, k-closest neighbor, deep learning)
  - Game theory (mathematical interaction between intelligently rational decision-makers (e.g. cooperative games, evolutionary games))
- Problems: Computer-technical: Exponential time growth; Information technology: information is limited, uncertainties exist, knowledge acquisition required)
- Why Now: Existing Data (Big Data), Algorithms (Deep Learning), Computer Power (GPU)
9. General Intelligence: Imitation of the complete human ways of thinking (= General AI / strong AI / full AI)
- 
- 

Artificial Intelligence Applications:

- Big Data Analysis
  - Machine translation
  - Speech input processing
  - Self-driving cars
  - **NLP (Natural Language Processing)**: Research area that deals with the interaction of human language and computers → refers to a way in which computers can analyze, understand and implement human languages in a useful way -based typically on machine learning algorithms
  - Security (face recognition, ...)
  - Sports Analysis
-

Good luck!

Motivation for the use of artificial intelligence

Driving tasks:

Sensors:

What can be achieved with Artificial Intelligence in the vehicle?

- Safety, comfort, energy savings, traffic reduction, new mobility services, development of new software developments
- Primary: driving task (navigation (hour to minutes), wayfinding (minutes to seconds), control (seconds to milliseconds), stabilization)
- Secondary: Machine task (gear selection, light, stroke, ...)
- Tertiary: people (safety, comfort, climate, acoustics, ...), media (internet, email, telephone)
- Radar, Lidar, Ultrasonic, Kerma, GPS, IMU
- **Road detection**
  - sensors: Kamera image, HD maps, GPS position
  - algorithm: sensor fusion, computer vision, fast map comparison
- **Environmental detection** (e.g. pedestrians, cars, ...)
  - Sensors: Images, Lidar Images, Radar Scans, Ultrasonic-S cans
  - Algorithm: Sensor fusion, computer vision, classification, mapping, ...
- **Driving limitations** (e.g. signs, ...)
  - Sensors: Camera images
  - Algorithm: Computer view, classification
- - Positioning sensors: camera images, HD maps, lidar laser scans, GPS position
  - algorithms: sensor fusion, computer vision, fast map matching, particle filter
- **Wayfinding and behavior planning Sensors:**
  - vehicle data, GPS position, camera images, lidar Laserscans
  - algorithms: sensor fusion, planning analysis, computer vision, regression, classification
- **Prediction of the behavior of other road users**
  - Sensors: camera images, lidar laser scans, radar sensors, ultrasonic sensors
  - Algorithms: sensor fusion, computer vision, search
- **Vehicle Control**
  - Sensors: Vehicle Data
  - Algorithms: Sensor fusion, model adaptation, regression, ...

Maximum allowable error rate:

- $1 \cdot 10^{-6} \rightarrow 1 \text{ error per } 100 \text{ million miles (based on human behavior)}$

#### SUMMARY:

- AI = Artificial Intelligence → Ability of a computer to perform special tasks better than a human
- Biggest problem: a large amount of "labeled" data is needed + good computer performance is needed
- Intelligence classification: emotional, creative, methodical, analytical
- Subproblems of AI: Reasoning & Problem Solving, Knowledge Representation, Planning, Learning, **Natural Language Processing** (NLP = Natural Language Processing), Perception, Motion & Manipulation, Social Intelligence (Social Intelligence)
- For each subproblem of AI, mathematical tools and methods can be applied to solve it
- Focus of AI: Machine learning → gives the computer the ability to recognize patterns and learn from data
- Machine learning is divided into three sub-problems: regression, classification, clustering

## Chapter 1: Perception

### Computer View (Computer Vision – CV)

Goal:

- Giving computers the ability to perceive their environment
- Input sensors: camera, lidar, ultrasonic, radar, microphones
- Machine perception: Ability to interpret environmental data
- Computer vision: (CV): Computer vision (CV): **Computer** input is analyzed and the information extracted

Camera advantage:

- Cheapest sensor in the vehicle
- Similar to the human eye,
- Camera images contain almost all necessary information about the environment
- Images can be treated using classical computer vision algorithms and deep learning algorithms

Problems for image recognition by camera:

- Objects are very diverse and look different
- Loss of information when converting 3D objects to 2D images
- **Missing or erroneous data:** absorbed or distorted elements, shadows and images covered with "noise"

Computer View Pipeline/Procedure:

1. Select hardware (camera, lens, ...)
2. Create images/video stream
3. Integrate image into software pipeline (e.g. via CV)
4. Prepare images
5. Detect / extract elements in the image
6. Gain information from the extracted items
7. Computer view Program onsetzen

### Machine Vision (MV)

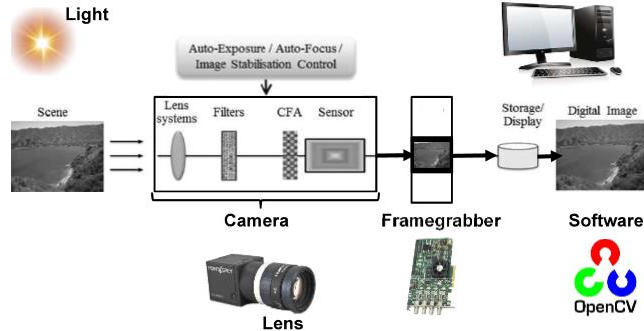
Prerequisites (see picture):

- **Light conditions** → Man can adapt to light, machine can't → Good lighting of the environment is necessary → Problem with autonomous driving
- **Optics and lens** → With the front lens, the field of view (field-of-view) and range of view → is adjusted A prior

Good luck!

knowledge of this is necessary!

- **Camera** → Camera contains sensor that converts light from the lens into electrical signals **this is → stored in a field of pixels** The resolution (precision) depends on the field of view, the viewing distance and the number of physical pixels in the camera sensor → r (standard VGA camera resolution: 640 x 480 pixels) and each physical pixel has an area of 7.4 microns
- **Frame grabber and software** → frame grabber sends the image synchronously or asynchronously over a cable/bus to the computer



## Image editing

Image processing:

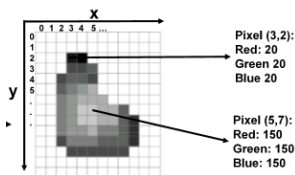
- Method to edit an image so that desired information can be extracted or the image is enhanced
- Different Computer View (CV) Algorithms / Mathematical Operations **are** used

Pipeline / Procedure:

- Object, camera and lens available
  1. Camera calibration
  2. Digital Transformation
  3. Divide the image into color ranges
  4. Image filtering
  5. Improvement of contrasts
  6. Affine Transformation
  7. Resampling / Compression
  8. Save

Camera calibration:

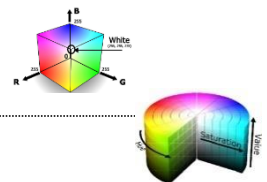
Definition Digital Image:



Color ranges:

RGB:

- Camera projects 3D object to 2D image Today's cameras are cheap and produce → **radial and tangential distortions** → The calibration should capture the internal settings of the camera and identify the parameters influencing the image (position of the camera center, image sensor format, distortion factor of the lens, ...)
- Numerical representation (usually binary) of a 2D image → Standard: digital image is a matrix of values based on colors
- The original images are obtained by using a scanner or digital camera the average of the amount of red, green and blue light the → lights hit the respective filters and are processed **accordingly**
- If we have a matrix of an RGB image named **im** (dimension: NxM), the individual pixels can be addressed as follows:  $\text{in}(y,x,b) = y$  pixels down,  $x$  pixels to the right and the whole thing in the Bten color range ( $R = 1, G = 2, B = 3$ )
- Different color ranges can be used → Best known: **RGB and HSV**
- Red-Green-Blue
- Values between 0 – 255 (8bit)
- (0,0,0) = black; (255,255,255) = white
- Theoretically  $(2 \cdot 8)^3 = \text{approx. } 16.8 \text{ million possible colours}$



HSV:

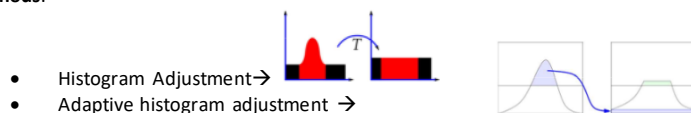
- H = Hue (hue);  $0^\circ = \text{Red}, 120^\circ = \text{Green}, 240^\circ = \text{Blue}$
- S = Saturation; 0% gray – 100% saturated color
- V = Value; 0% dark – 100% light

Filter:

- Image filters in the spatial domain: mathematical evaluation of a number grid → Methods: smoothing, sharpening, measuring texture
- Image filter in the frequency domain: is a way to change the frequency of an image → Methods: denoising, sampling, image compression
- Procedure: Multiplication of the image matrix by a filter matrix

Contrast extension:

- Kontrast = **gradation** variation between different colors of an image → Describes the difference between the darkest and lightest hue and defines the grayscale
- Low contrast: Image colors are very close to each other
- Adjustable by the histogram of the image
- Contrast Extension: Changes the image value to cover a large area
- **Methods:**



Affine Transformation:

- A function that preserves points, straight lines, and areas. → Parallel lines remain parallel
- Does not need to get the distance or angle of two lines or distance of two points
- However, it maintains the ratio of the position of a point on a line

Good luck!

$$\text{Rotate} \begin{pmatrix} \cos(\theta) & \sin(\theta) & 0 \\ -\sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \text{Reflect} \begin{pmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Resampling:

### Feature extraction

Feature recognition:

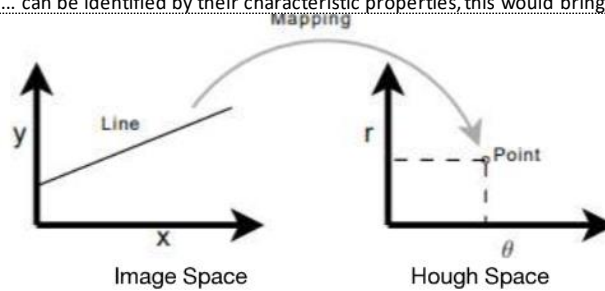
Remember extraction:

Used for:

Edge detection:

- E.g.
- Mathematical technique to create a new image that has a different pixel height and/or width → Make bigger: Upsampling; Make smaller: Downsampling
- Examination of the image for the desired feature → The feature found would be stored in a subset of the image (often in an isolated form, continuous curves or connected regions)
- If a feature is found, it can be extracted
- 3D reconstruction, motion detection, robot navigation, ...
- Identifies sudden changes in intensities in an image a lot of → information in the image can be detected → more compact than images
- Role model: Artist drawing → Problem: Artists use object-based prior knowledge)
- Possibilities:  
By means of horizontal line → detection of deviations from this;  
Canny Edge Detection → Pre-built edge detection tool
- Problem: Reduces the image information, but it remains an image that is composed of pixels → If lines, ellipses, ... can be identified by their characteristic properties, this would bring more → remedy: Hough Line algorithm

Hough Line algorithm:



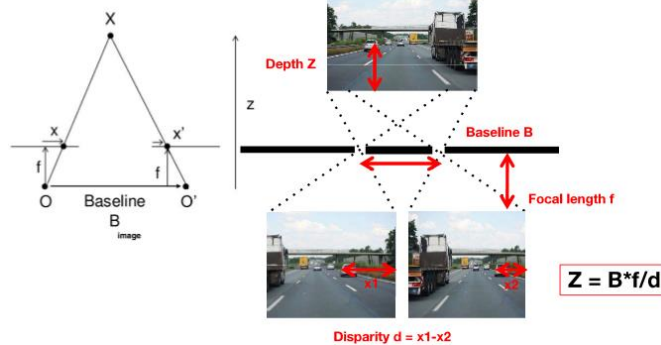
1. Edge detection (e.g. by canny edge detection)
2. Storage of edge points in the Hough area
3. Interpretation of the accumulator to obtain lines of infinite length
4. Conversion of infinite lines to finite lines

Corner detection:



Depth detection:

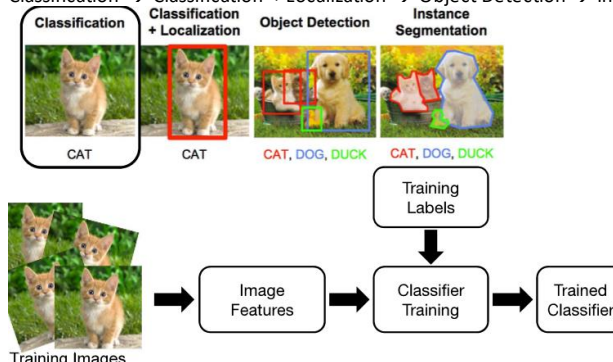
- By means of a small window, corners can be easily detected → by moving the window and detecting significant changes in intensity
- Possible regions: flat (no change in intensity in all directions), edge (no change along the edges direction), corner (significant change in all directions)
- Inequality between 2 images → Result: Inequality Map



### Characteristic Analysis

General:

- Classification → Classification + Localization → Object Detection → Instance Segmentation

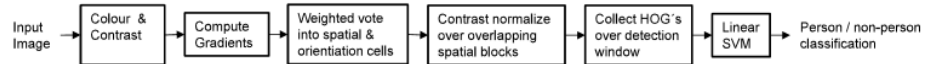


Good luck!

Classification methods:

- Rule-based, logical, database
- Machine learning
- Deep Learning

Procedure:



### Information analysis

Optical flow:

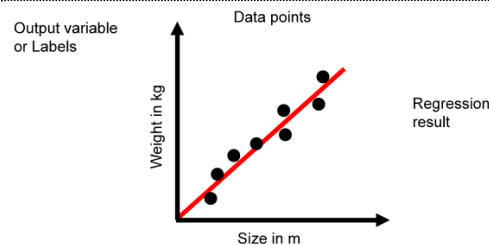
- Video is a sequence of images over time
- Optical flow is a pattern of apparent movement of objects, surfaces, and edges in a visual scene created by the relative movement between the viewer and the scene.

### SUMMARY:

- Perception is an AI problem
- Division of perception into computer vision (computer vision → ability to receive information from a camera) and machine vision (machine vision) possible
- Reasons for using cameras: cheap and easy to use → provide a large amount of data → similar to the human eye
- Machine Vision Setup Challenges: Big Differences That Need to Be Known: Camera, Lenses, Computer Setup, Operating Environment
- CV pipeline is important
  - Images need to be edited to work better: filters, contrast adjustment, color range changes, ...
  - Features such as edges and corners must be extracted
  - With the features, information can be obtained from the image (object recognition & classification, ...)
  - With machine learning algorithms, **this classification can be performed based on features and the images that have been previously edited.**

## Chapter III: Regression

General and motivation:



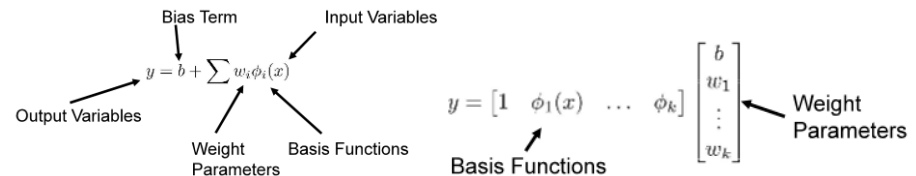
- Produces continuous output
- supervised
- Determining Compensation → Values Filtering Out Outliers
- Objective: To enable the prediction of data and model structures based on the total outputs → Noise reduction, since the data are mostly only instantaneous → values Allows the data to be used in simulations, optimizations, ...
- Examples: house prices, number of sales, person weight
- Application in the automotive industry: **sensor calibration** (accelerometers, gyroscopes, ...), **parameter estimation** (many parameters only partially known), **pricing**, ...

### Model variants:

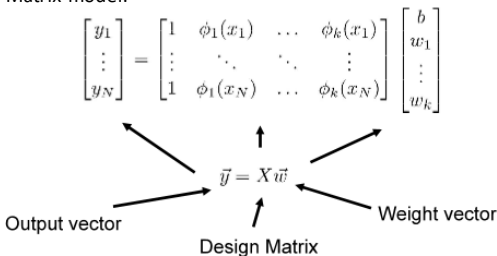
Linearmodel:

General:

- Function model:



- Matrix model:



- Nonlinear regression:

- **Nonlinear optimization:** solves the nonlinear optimization problem, specifies the parameter set (left)
- **Gaussian process:** no parameters, Bayesian interpretation (middle)
- **Vector machine support:** No parameters, Uses kernel methods, Mainly used for classification (right)

Good luck!

Expiration:

Possible basic functions:

1. Select model (type and number of basic functions)
2. Find/select parameters: Definition of a loss function to find out how well the model reproduces the data
3. Validate model: A data set that was not used for training is used to control and test the obtained model → Ensure that unviewed data is not forgotten

- Top left: Linear function
- Bottom left: Sine function
- Top right: Polynomial function
- Bottom right: Gaussian base function

- Polynomial functions:

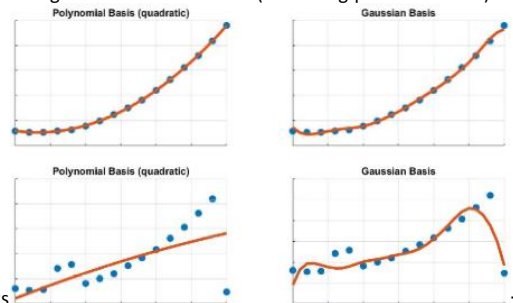
- Global defined on an independently defined variable domain
- Design matrix is poorly conditioned for a large set of input variables

- Gaussian basic function:

- Locally defined on an independent variable domain
- Sparse Design Matrix
- Infinitely differentiable
- Hyperparameters: number of Gaussian functions, width, mean of each  $\mu$  basis function

$$\exp\left(-\frac{(x - \mu)^2}{2s^2}\right)$$

- Comparison: left: Global base function n right: local base function (scattering parameter: 0.3) Graphs show the

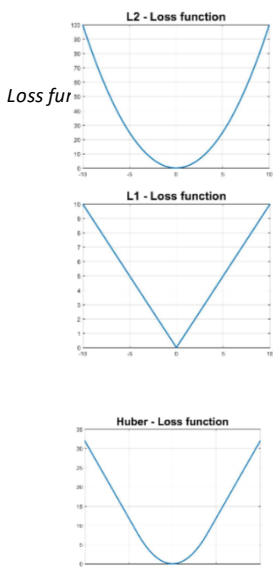
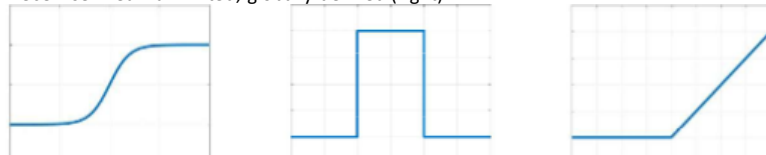


sensitivity of the functions to outliers

→

- Other basic functions:

- Sigmoid function: Limited and globally defined (left)
- Binary function: bounded, Locally defined, Non-continuous (middle)
- Piecewise linear: unlimited, globally defined (right)



- Measures the correctness of the model based on the training data → Best Model: Minimum Loss Function
- $\min_{\vec{w}} (\vec{L}(\vec{x}, \vec{y}, \vec{w}))$  with L = loss function; x = independent variables, y = target variables, w = weights

- Mean Squared Error (L2) →  $L = \frac{1}{N} \sum (y - \hat{y})^2$ 
  - Advantages: important for practical applications, a solution can be easily obtained analytically, differentiable
  - Disadvantages: Not robust against outliers
  - Examples: basic regression, energy optimization, control applications

- Mean Absolute Error (L1) →  $L = \frac{1}{N} \sum |y - \hat{y}|$ 
  - Advantages: Robust against outliers, more intuitive
  - Disadvantages: No analytical solution, originally not differentiable
  - Examples: Financial Applications

- Huber Loss →  $L = \sum \begin{cases} 0.5(y - \hat{y})^2, & \text{for } |y - \hat{y}| \leq \delta \\ \delta |y - \hat{y}| - 0.5\delta^2, & \text{otherwise} \end{cases}$ 
  - Advantages: Combines the strengths of L1 and L2, robust and differentiable



Good luck!

Analytical solutions of mean square error (MSE):

- Cons: More hyperparameters, No analytical solution
- In practice: start with L2 if possible

Example low-dimensional example solution → of the optimization problem  $\min_{\vec{w}} \frac{1}{2} \sum (y - \hat{y})^2$  with  $\hat{y} = w_1 x + b$  and  $\{x_1, y_1\}, \{x_2, y_2\}, \{x_3, y_3\} \rightarrow$

$$\min_{w_1, b} \frac{1}{2} \left[ (y_1 - w_1 x_1 - b)^2 + (y_2 - w_1 x_2 - b)^2 + (y_3 - w_1 x_3 - b)^2 \right]$$

$$\Delta L(x, y, w) = \begin{bmatrix} \frac{\partial L(x, y, w)}{\partial w_1} \\ \frac{\partial L(x, y, w)}{\partial b} \end{bmatrix} = 0$$

→ In general, the optimal solution is where the derivative = 0 :

The most important advantage of this feature is that there is an analytical solution and can be calculated for small and large data sets.

Sequential analytical solution:

- → Recursive Least Square LRLS → method is difficult to operate while a product does not have enough memory to store all data points.

$$\vec{w}(k+1) = \underbrace{\vec{w}(k)}_{\text{Old parameter estimate}} + \underbrace{P(k) \vec{x}^T}_{\text{Correction gain}} \underbrace{(I + \vec{x}^T P(k) \vec{x})^{-1}}_{\text{Residual}} (\underbrace{\bar{y} - \vec{x}^T \vec{w}(k)}_{\text{Prediction based on old parameters}})$$

- With  $P$  = memory matrix →  $P(k+1) = (I - P(k) \vec{x}^T (I + \vec{x}^T P(k) \vec{x})^{-1} \vec{x}) P(k)$
- With  $I$  = identification matrix of the corresponding dimension

- Forgotten error: Compensation of parameter changes after a long time → multiplication by  $\gamma$  →

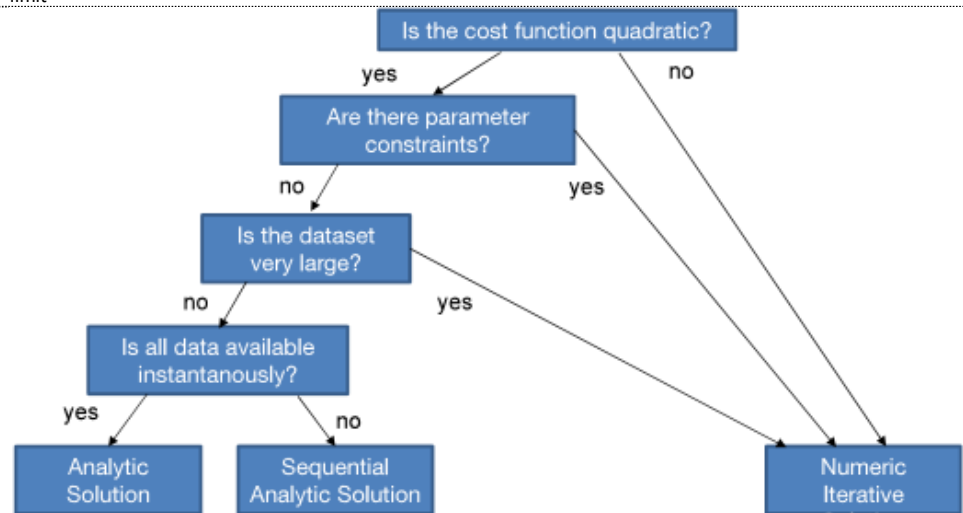
$$\vec{w}(k+1) = \vec{w}(k) + P(k) \vec{x}^T (\gamma I + \vec{x}^T P(k) \vec{x})^{-1} (\bar{y} - \vec{x}^T \vec{w}(k))$$

$$P(k+1) = \gamma^{-1} (I - P(k) \vec{x}^T (\gamma I + \vec{x}^T P(k) \vec{x})^{-1} \vec{x}) P(k)$$

Numerically iterative solution:

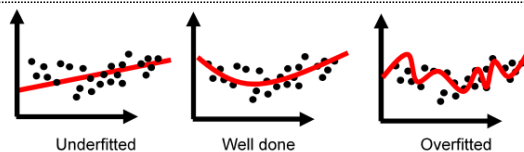
- Regression numerically soluble important for large-scale problems and for non-quadratic loss functions →
- Methods: gradient lineage, Gauss-Newton, Levenberg-Marquardt
- Advantage: Very generic
- Disadvantage: Knowledge of numerical optimization required
- Limitations of weights: Weights can be seen as physical quantities (temperature, mass, ...) that have a defined limit

Selection of the appropriate calculation type:



Validation of the solution:

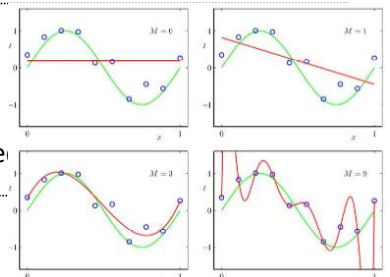
Choosing the right model:



- Not enough features
- Wrong structure
- Too many features
- Unrelevant features

Overfitting:

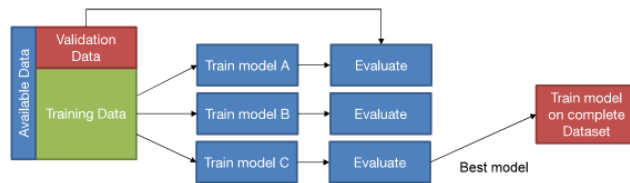
- Choice of hyperparameters
- Overfitting = error to properly generalize the data points
- Cost function decreases with increasing model complexity
- Interference and noise become too important
- Overfitting occurs when:
  - The data points are too sparse or the model is too complicated
- Sparseness increases rapidly with increasing input dimension (1D, 2D, 3D)





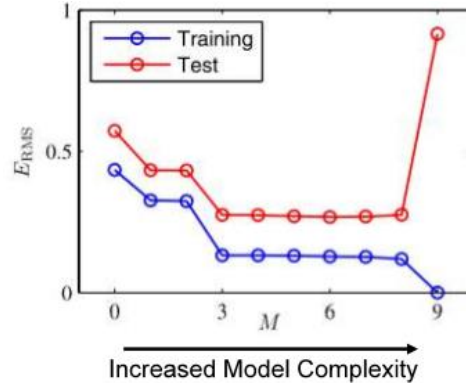
Good luck!

- Difficult to control in high-dimensional domains or autonomous systems
- A statistical technique is to separate the data into training and validation data



Validation data set:

- Different hyperparameters can be used to influence the model
- Validation is always the same



- Graphic: from  $M = 3$  the hyperparameters fit, from 8, no longer
- Prerequisite:
  - Test data set must represent the future properties of the underlying physical system
  - The test data must not be a repetition → If it is used again and again, the test data is no longer independent data and thus no longer represents the original system
  - Division of test data necessary before design → min. 2/3 of the data as training data is recommended
- K-Fold cross-validation: If there is little data and it is not to be split, a smaller data set can be used, which is divided into several "folds" → Variance of estimation error is an indicator for model stability

- Model structure should not be chosen according to characteristics of the data set, but according to the underlying physical principles
- Problem with polynomial basis function: tends to have large coefficients

→ for few data Problem with

→ Gaussian basis function: tends to overfit locally, resulting in individual large coefficients

- One workaround is regularization:
  - "Punishment" of high coefficients in optimization prevent these effects
  - Weighting of punishment terms yields intuitive hyperparameters to control model complexity

- Ridge Regression: (L2 regularization)

- Prevents overfitting well → Analytical solutions are possible as an extension to the MSE
- Difficult to use and adjust for high dimensions

$$\min_{\vec{w}} L(\vec{x}, \vec{y}, \vec{w}) + \lambda \vec{w}^T \vec{w}$$

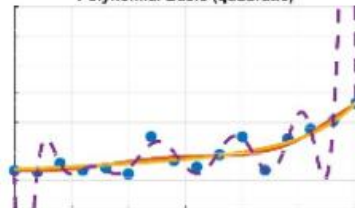
- Lasso Regression: (L1 regularization)

- Tends to provide sparse solutions and can therefore be applied to feature selection
- Sparse solution means that several coefficients become zero

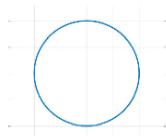
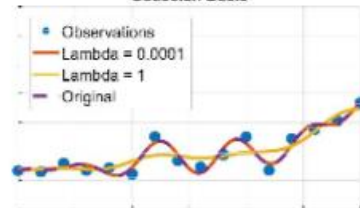
$$\vec{w} = [0 \quad w_1 \quad 0 \quad 0 \quad 0 \quad w_2]$$

$$\min_{\vec{w}} L(\vec{x}, \vec{y}, \vec{w}) + \lambda \sum_i |w_i|$$

Polynomial Basis (quadratic)



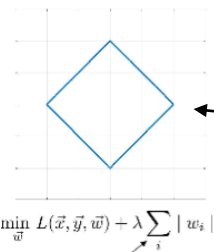
Gaussian Basis



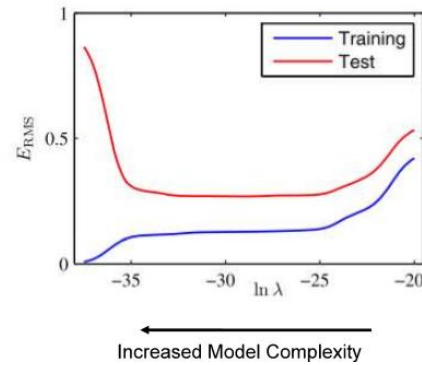
$$\min_{\vec{w}} L(\vec{x}, \vec{y}, \vec{w}) + \lambda \vec{w}^T \vec{w}$$

Regularization Term

Regularization:



Good luck!



Normalization:

- Adjustment of regularization with the training data
- Input data:
  - Regression requires inversion of design matrix
  - Numerically complex and can lead to instabilities
  - In general, all matrix entry variables must be of the same order of magnitude
- Output data:
  - weighting weights to highlight important data; Applying regression to multiple tasks → which may not have the same scale

Learning summary:

- What is regression and what is the difference to clustering and classification
- Difference linear and nonlinear regression
- Possible applications and applications in the automotive industry
- Various application possibilities for local and global basic functions
- Types of loss functions
- Iterative and analytical solution method for training models
- Regularisation and validation techniques

## Chapter IV: Classification

Definition:

- Systematic organization of events into groups or categories according to the criteria applied

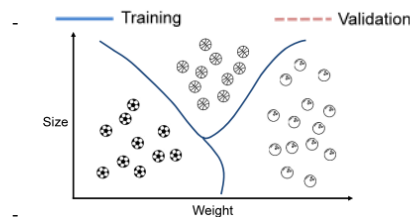
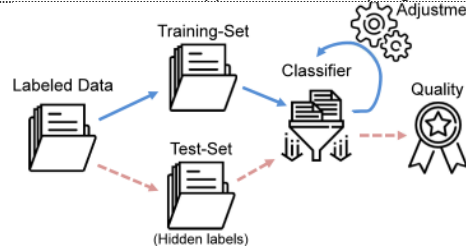
General:

- Generates discrete, estimated output data → Supervised
- Examples: Object, Spam Detection

Machine learning:

- Requires prior knowledge for correct classification + must automatically create complex classification rules → should be able to be applied to extremely large data sets

Expiration:



Quality requirements for classification:

- **Scalability** (efficient in disk-resistant data set), **Compact** (decision tree size, number of decision rules), **Correct** (loss function), **Interpretable** (comprehensibility of the data set and decisions), **Efficient** (time to generate the model (training time) and time to use the model (predict time)), **Robust** (robust against variables and missing variables)

Validation options of the classification:

- k-fold Cross Validation:
  - Split dataset into k equal subsets and use k-1 partitions as training data and the remaining partition as dataset
  - Additional requirement: stratified folds → The class distribution in the training and test data must correspond to the classification of the total set
  - Standard: 10x folded validation

Good luck!

Confusion Matrix:

		Classified as			
Correct Label		Class 1	Class 2	Class 3	Class 4
	Class 1	45	0	2	1
	Class 2	3	44	0	1
	Class 3	0	0	67	0
	Class 4	8	5	6	37
		Person ist krank ( $r_p + f_n$ )		Person ist gesund ( $f_p + r_n$ )	
Test positiv ( $r_p + f_p$ )		richtig positiv ( $r_p$ )		falsch positiv ( $f_p$ )	
Test negativ ( $f_n + r_n$ )		falsch negativ ( $f_n$ )		richtig negativ ( $r_n$ )	

With:

- Recall:  $\frac{TP}{TP+FN}$
- Precision:  $\frac{TP}{TP+FP}$
- Specificity:  $\frac{TN}{TN+FP}$

True Positives: TP  
False Positives: FP  
True Negatives: TN  
False Negatives: FN

## Methods

More:

Logistic Regression:

- Decision construction, neural networks
- $$g(z) = \frac{1}{1 + e^{-z}}$$
- 
- Sigmoid function:
  - Advantage: Easy to use, probability statement about whether an element is in a class, fast training phase, generates comprehensible models
  - Disadvantages: linearity (difficult to apply to nonlinear problems), overfitting (training data must be chosen skillfully)

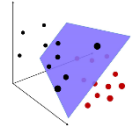
Nearest Neighbors:

- Classification by looking for the nearest neighbor
  - **Example-based learning:**
    - No training or testing phase
    - Stores tagged training data
    - Processes training data when a new object needs to be classified
    - Game between complexity and time → Difficult to create a model based on a large data set, but it is easy to use; Easy to store a large data set, but difficult to search
  - Other methods:
    - NN classification: Considers only the nearest neighbor (see top left)
    - K-NN classification: Considers closest neighbors (see bottom left)
    - Weighted k-NN classification: Considers the weighted distance to the nearest neighbors (see bottom center)
    - Averaged NN classification: Looks at the closest mean of the positionsen **Of** the classes (see bottom right)
- 
- Determine the number k:
    - Generalization vs. Overfitting
    - Capital K: Many objects from different classes
    - Small K: Sensitive to outliers
    - Practice:  $1 \ll k \ll 10$
  - Weighting of neighbors of the weighted k-NN method:
    - Frequency of neighbouring classes:  $w_i = \frac{1}{\text{Häufigkeit}_i}$
    - Distance to neighbour:  $w_i = \frac{1}{\text{Distant}_i^2}$
  - **Advantages** closest neighbor:
    - Easy to calculate/calculate, Good results for many applications, Easy adaptation to new training data, Robust against noise by averaging (k-NN)
  - **Disadvantages:**
    - Efficiency (decreases with increasing training data) (can be reduced by → logarithmization, dimensionality (not every dimension is relevant) (→ weighting dimensioning)
  - **Neutral:** Does not create explicit prediction capabilities for classes

Good luck!

## Support Vector Machine:

- **Linear separation:** objects in  $\mathbb{R}^d$ ;  $\rightarrow 2$  classes divided by a hypersurface  $\mathbb{R}^d$   
 $\rightarrow$  Training: calculation of the hypersurface; Classification: Distance to hypersurface



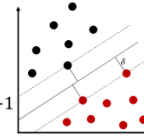
- SVM - Maximum **Margin Hyperplane**: Maximum distance to hyperplane; High generalization, supports vector distance formal definition:  $\rightarrow \delta$

Training data:  $(x_1, y_1) \dots (x_n, y_n)$   
with  $x \in \mathbb{R}^d, y \in \{-1, 1\}$

Training: Minimize  $\|w\|$   
with  $y_i(w \cdot x_i - b) \geq 1$  for  $i = 1 \dots n$

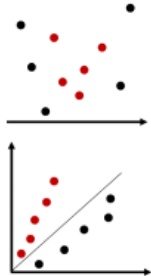
Hyperplane:  $w \cdot x - b = 0$   
with  $w$  normal vector,  $\frac{b}{\|w\|}$  offset from origin,

Classification:  
if  $(w \cdot x - b) \geq 0$ ,  $y = 1$ ; else  $y = -1$   
with Data  $x \in \mathbb{R}^d$

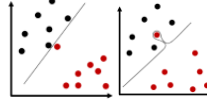


$\rightarrow$  Margin:  $\delta = \frac{1}{\|w\|}$

- SVM - **Soft Margin**: Linear separation: Not always possible, Not always optimal tradeoff between error and spread: Allow error



$\rightarrow$  to increase margin



- SVM - **Space Transformation**:

- Non-linear data  $\rightarrow$  Too many errors with soft margin
- Use of higher-dimensional spaces: Increase dimension until linear separation is possible  $\rightarrow$  Transform hyperplane back Hyperplane  $\rightarrow$  becomes nonlinear
- For example, quadratic transformation  $\rightarrow$  hyperplane becomes a 2nd order polynomial

- SVM - **Kernel Machines**:

- Dimension transformation: from lower to higher dimension (complex calculation)
- Hyperplane transformation: higher to lower dimension  $\rightarrow$  Feasibility not guaranteed (complex calculation)
- Kernel: elegant computation; Calculates point product without complete space transformation

$\rightarrow$  Calculation: Polynomial:  $k(x_i, x_j) = (x_i \cdot x_j)^d$ ; Gaussian radial bias function:  
 $k(x_i, x_j) = \exp(-\gamma \|x_i - x_j\|^2)$  for  $\gamma > 0$ ; Other possibilities: Linear, Sigmoid, Hyperbolic

•  $f: \mathbb{R}^3 \rightarrow \mathbb{R}^9$

$f(x) = (x_1x_1, x_1x_2, x_1x_3, x_2x_1, x_2x_2, x_2x_3, x_3x_1, x_3x_2, x_3x_3)$   
 $k(x, y) = (x \cdot y)^2$

•  $x = (1, 2, 3), y = (4, 5, 6)$

$f(x) = (1, 2, 3, 2, 4, 6, 3, 6, 9)$

$f(y) = (16, 20, 24, 20, 25, 30, 24, 30, 36)$

$f(x) \cdot f(y) = 16 + 40 + 72 + 40 + 100 + 180 + 72 + 180 + 324 = 1024$

•  $k(x, y) = (4 + 10 + 18)^2 = 32^2 = 1024$

$\rightarrow$  Example:  $\rightarrow$  no transformation to  $\mathbb{R}^9$  required

- Pros: High classification rate, effective when dimension  $>$  number of examples; Robust to overfitting, Compact (level in area), Versatile (different kernels functions)
- Disadvantages: Efficiency (large training phase), complexity (high implementation effort), black box (models are difficult to interpret)

## Uses

### Examples:

- Big data, image classification, music classification, language classification, error detection (quality control during production)
- For automotive:
  - $\rightarrow$  Perception (camera output pixel fields  $\rightarrow$  classification adds values to each pixel pixel  $\rightarrow$  splitting  $\rightarrow$  object recognition)
  - $\rightarrow$  Vehicle detection and tracking: training data  $\rightarrow$  Extracts image features  $\rightarrow$  Model creation based on features  $\rightarrow$  Recognition of features on each frame of a video stream
- How to obtain labeled data?  $\rightarrow$  Create yourself, pay someone for it, use database
- Feature recognition: histogram of orientations by derivatives

Example program Python SVM  
Classifier: scikit-learn:

```
>>> from sklearn import svm
>>> clf = svm.SVC()
>>> clf.fit(training_features, training_labels)
>>> clf.score(test_features, test_labels)
>>> clf.predict(new_feature)
```

```
o Training: 1.44 Seconds
o Test: Accuracy = 0.9848
o Prediction: 0 or 1
```

## Summary:

- Classification = assignment of given classes to data  $\rightarrow$  for this a high number of evaluated (labeled) training data is required  $\rightarrow$  Is a supervised procedure
- Machine learning can extract knowledge from a large data set
- Several criteria are used to measure the quality of a classifier
- Methods of classification: Logistic regression
  - $\rightarrow$  Use of linear regression together with sigmoid Function as classification method possible  $\rightarrow$  Closest neighbor
  - $\rightarrow$  Example-based learning method, no training is needed  $\rightarrow$  SVM linear

Good luck!

- classifiers that use a maximum margin hyperplane → With Kerneltrick, SVMs can be used as linear classification →
- Classification is very important for perception of a system → Requires a lot of ranked data
  - There are ready-made Python libraries and open-source datasets for classification processes
  - Training with large data sets takes a lot of time
  - Procedure: Classify → data Connect → data
  - For classification, features must be extracted from the images

## Chapter V: Clustering

Definition:

General:

- Grouping of similar elements that are close to each other, sometimes also those that surround things
- Generates discrete, estimated values → Unsupervised
- Examples: Google News, point cloud processing (lidar)
- Clustering vs. Segmentation: both are interchangeable Clustering:
  - Statistical background → Clustering creates segments and vice versa
  - Segment: Business Background
- Cluster = A collection of items → one element of a cluster is similar to the others of the same cluster and dissimilar to the others of the same cluster
- Differences to classification: No pre-assigned classes, Unsupervised learning
- Applications: Provide insights into large data sets, prepare the data for other algorithms

Training and validation:

- Elements  $e \in E$
- Cluster  $c \in C$ , with  $c \subseteq E$  and  $\bigcup_{c \in C} c = E$  and  $\bigcap_{c \in C} c = \emptyset$
- Representative  $r_c = \text{mean}(c)$
- $\text{variability}(c) = \sum_{e \in c} \text{distance}(r_c, e)^2 \rightarrow \text{Clustering: Minimize } \sum_{c \in C} \text{variability}(c)$

Euclidian

$$\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

- Euclidean distance:
- Chebyshev distance:  $\max(|x_1 - x_2|, |y_1 - y_2|)$



- Unsupervised learning: — Training — Validation
- Quality check of a cluster:

- Distances to representatives of the cluster depend on  $k$ :  $k = 2$ : very large distances,  $k = n - 1$ : very small distances
- Similarity: Within a cluster:  $o \in a \in C \rightarrow$  Mean distance to all elements in the same cluster:  
$$\text{sim}(o) = \frac{1}{|a|} \sum_{e \in a} \text{distance}(o, e)$$
- Dissimilarity: to other  $e \notin b \in C \rightarrow$  clusters: Mean distance to all elements of the second closest cluster:  
$$\text{dsim}(o) = \min_{c \neq a} \left( \frac{1}{|c|} \sum_{e \in c} \text{distance}(o, e) \right)$$
- Silhouette coefficient: 
$$s(o) = \frac{\text{dsim}(o) - \text{sim}(o)}{\max\{\text{sim}(o), \text{dsim}(o)\}}$$
 if  $\text{sim}(o) = \text{dsim}(o) = 0$ , then  $s(o) = 0$   
$$\text{silh}(c) = \frac{1}{|c|} \sum_{o \in c} s(o)$$
  
$$\rightarrow \text{silh}(E) = \frac{1}{|E|} \sum_{o \in E} s(o)$$

## Methods

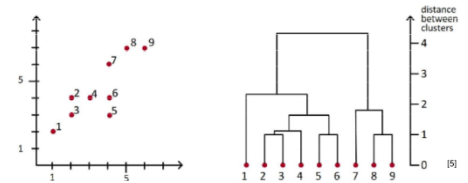
Hierarchical clustering:

- Procedure:
  1. Start with one cluster per element
  2. Combine the two closest clusters
  3. Repeat step 2 until all items are in a cluster
- **divisive clustering method**: in which first all objects are considered to belong to a cluster and then gradually the already formed clusters are divided into smaller and smaller clusters until each cluster consists of only one object. (Also referred to as "top-down procedure")
- **agglomerative clustering methods**: in which each object first forms a cluster and then gradually the already formed clusters are combined into larger and larger ones until all objects belong to a cluster. (Also referred to as "bottom-up approach")
- Distance types:
  - Single connection: Smallest connection of two elements of different clusters
  - Complete connection: Largest distance between two elements of different clusters
  - Middle link: Mean connection between all elements of one cluster and those of another cluster

	BOS	NY	CHI	DEN	SF	SEA
BOS	0	206	963	1949	3095	2979
NY		0	802	1771	2934	2815
CHI			0	966	2142	2013
DEN				0	1235	1307
SF					0	808
SEA						0

Good luck!

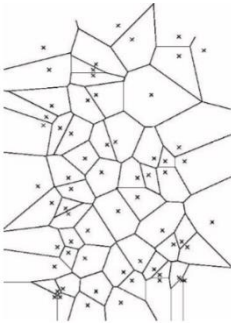
- Dendrogram (see right):
  - root: Cluster with all points
  - leaf: Cluster with a point
  - Edges: Combines two clusters
  - Depth: Distance between two combined Clusters



- Example see left
- Advantages: Generic (no cluster number or cluster parameters need to be defined), Visualization (dendrogram shows hierarchy), Hierarchy (ratio of the individual clusters to each other), Deterministic (always creates the same clusters)
- **Disadvantages:** Scalability (runtime:  $O(n^3)$ ), Selection (The final valid cluster must be selected based on the hierarchy)

*k-means:*

- **Basic idea:** Minimization of the square distance to the cluster mean value (variability), minimization of the summed variability of all clusters (large sum: bad cluster, minimized cluster: optimal cluster)
- **k choose:** Preliminary expert knowledge (5 different types of bacteria:  $k = 5$ )
  - looking for good  $k$ :
  - 
  - Naive approach: brute force with  $k = 2 \dots n-1$
  - Pre-run hierarchical clustering with a subset of data
- **Lloyd algorithm:** Given the number of desired clusters  $k$  + record Initialization: Selection of  $k$  arbitrarily Repeat
  - until stable: assign objects to the nearest representative + calculate the center of each cluster as a new representative value
  - 
  -
- Dealing with randomness:
  - Naive approach: Use a small subset of the dataset and use the representations found to initialize the entire dataset
  - Improved approach: Select a random small subset → Save Clusters + Representation → Cluster the merged sets and save them as representations → Use Best Clustering as Initialization of the Total Data
    - Get  $m$  small random subsets  $A \dots M \subset E$
    - Cluster  $A$  to  $M$  and save representatives  $R_A \dots R_M$
    - Cluster the merged set  $AM = A \cup \dots \cup M$ ,  $m$  times with  $R_A \dots R_M$  as initial representatives
    - Use the representation ( $R_A \dots R_M$ ) of the best clustering of  $AM$  as initial representation for  $E$
- **Advantages:** Efficient ( $O(tkn)$  with mostly  $k, t \ll n$  (with:  $n$  = number of objects,  $k$  = number of clusters,  $t$  = iterations)), implementation easy to implement
- **Disadvantages:** mean must exist (applicability), sensitive to outliers (noise),  $k$  must be defined (specification), can be stuck in a local optimum (initialization), convex space partitions (cluster shape)
- Other methods:



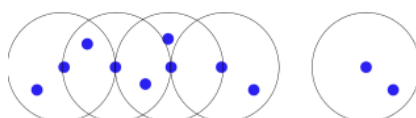
- Voronoi Model: Voronoi diagram divides space into Voronoi cells for each point → Voronoi cell of point  $p$  covers the area where the nearest data point  $p$  is
- K-Medoids, K-Median Clustering:
  - Represents the mean (not immer present)
  - Distance: squared distance → normal distance (reduction of interference by outliers)
  - 2 types: Medoid: Object in the middle, Median: Artificial object in the middle
  - Basic idea: Minimum distance between objects from a cluster to its representation

	K-means	K-medoid	K-median
data	Numerical data (mean)	metric	ordered attributed data
efficiency	High $O(tkn)$	Low $O(tk(n-k)^2)$	High $O(tkn)$
Sensitivity to outliers	High	Low	Low

- Pros: Easy to implement/use
- Disadvantages:  $k$  must be specified (specification), can be stuck in a local optimum (initialization), convex space partitions (cluster shape)

*DBSCAN:*

- Density Based Clustering (Dichtebasiertes Clustering)
- Two parameters: minimum number of points,  $\epsilon$  radius of the environment
- Three point classes: core, edge, outlier
- $p_n$  is reachable from  $p_1$  if there is a path  $p_1$  to  $p_n$  where every point  $p_i$  on the path must be a core point, except for  $p_{n0}$ 
  - $p_n$  is „reachable“ from  $p_1$ , if there is a path  $p_1 \dots p_n$  where each  $p_i$  on the path must be a core point, except for  $p_n$



- **Advantages:** any room division (cluster shape),  $k$  is determined automatically (specification), separates cluster

Good luck!

from interference (noise), efficiency DBSCAN  $O(n^2)$

- **Disadvantages:** Parameters are difficult to determine (specification), Very susceptible to parameter changes (sensitivity)

#### Uses

Examples:

For the automotive industry:

- Google News, genome patterns, social networks, market aggregation, buyer clustering (Amazon, Netflix)
- Traffic analysis: collects mobility data from cars or density of specific regions → clusters to filter interesting information; High level sensor fusion; Low-level sensor fusion

Summary:

- Clustering is used to find groups in records → is an optimization problem → unsupervised problem, no labeling required
- objects in a cluster are similar to each other and dissimilar to other clusters
- Distance can be used to express similarity
- The silhouette can be used to determine the quality of a cluster
- Segmentation and clustering can be exchanged
- Hierarchical clustering → forms hierarchical dendrogram, the number of desired clusters can be determined subsequently
- k-mean: fast, greedy, non-deterministic; Number of clusters must be determined beforehand, only convex clusters can be created
- DBSCAN: is density-based, and can handle noise; The elements are classified as cores, edges and outliers
- Complex shapes can be grouped as clusters
- Clustering is used as a pre-process or to find coherence
- Many applications, but rarely used alone → Experts or classification methods give importance to clusters afterwards

## Chapter VI: Pathfinding

Problem:

- People are looking for a way very visually
- They exclude paths from the outset that can already be classified as visually pointless
- Human pathfinding mechanism not yet 100% known → Difficult to design an algorithm according to it → Machine pathfinding takes place differently and iteratively

Required elements for car navigation:

- HMI → User Interaction
- Navigation mechanism → routing, positioning, guidance
- Data → Map, Transport, Point of Interest

What needs to be modeled?

- **Road geometry** (starting point, end, length, position)
- **Road connections** (connections, as the crow flies)
- **Road attributes** (passage restrictions, speed limits, type)

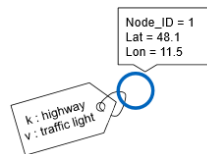
Road modelling:

- Self-creation using GPS, satellites, ...
- Open Source Maps

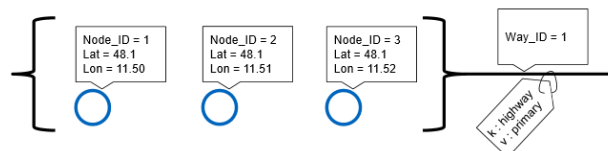
#### User HMI

Definitions:

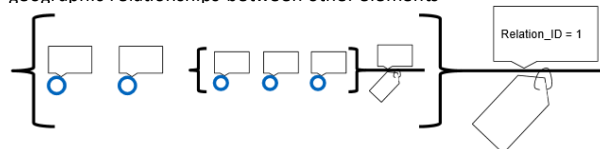
- **Tags:** Tags describe characteristics of the map elements → Key and values are free text based, but subject to certain guidelines
- **Node:** ID, length and width → attributes are defined by tags a → tag consists of a key and values



- **Paths:** Defined by an ordered number of nodes Nodes → are grouped into edges → Attributes are specified by tags



- **Relationships:** Defined by an ordered number of nodes, paths, and relationships → Define logical or geographic relationships between other elements



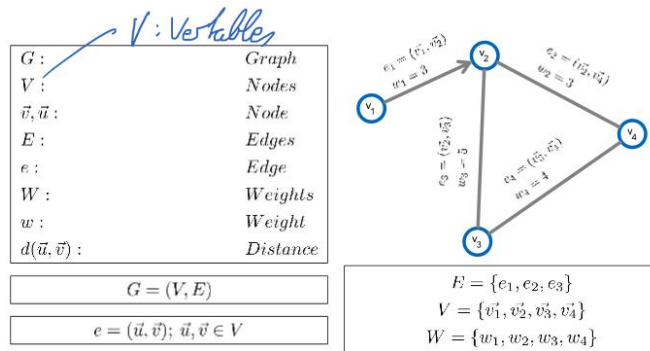
Element Graph:

- Nodes: , Direct edges: , Indirect/unweighted edges: , Weighted edges (cost time, distance →): ,



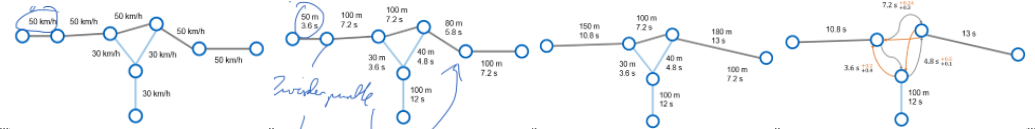
Good luck!

Convention:



Steps to create roads using counts:

1. Definition of properties (geometry, speed limits, ...) (left)
2. Calculate the cost of each route (middle left)
3. Remove unnecessary intermediate nodes (center right)
4. Drawing of turns (adjustment of the required time) (right)

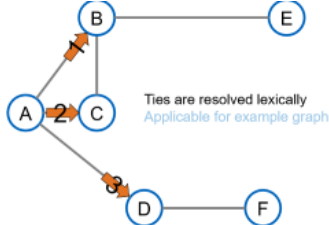


Assumptions:

- Only non-negative weights
  - Edges are unoriented (not a one-way street)
  - Nodes are given by their coordinates in a two-dimensional space
  - Weighted edges are defined by a distance function  $\rightarrow$  The Euclidean distance is used
- $$\|\vec{v} - \vec{u}\|_2 = \sqrt{(v_x - u_x)^2 + (v_y - u_y)^2}$$

#### Navigation mechanism

Search trees:



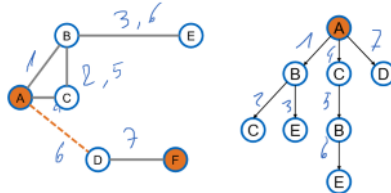
- **Definition:**
  - A search tree can be obtained from a graph
  - Each child node denotes a path that is a one-step extension of the path designated by its parent
  - Converting a graph into a search tree: writing out all possible ways until there is no way left
  - Only partial trees are used in the chapter: Only the edges that are suitable for the route are shown
- Problem formulation: Find a way (Depth First,  $\rightarrow$  Breadth First, Best First)  
 $\rightarrow$  Find an optimal way (Dijkstra, A\*)
- **Completeness:** A search algorithm is complete when it is guaranteed that an existing solution will be found in a limited time.
- **Informedness:** A graph is informed if it contains additional information that is interesting for the problem to be solved and has the potential to guide the search algorithm to its goal.
- **Convention:** Paths never bite their own tail

#### Algorithms: Find a Way

British Museum:

Depth First:

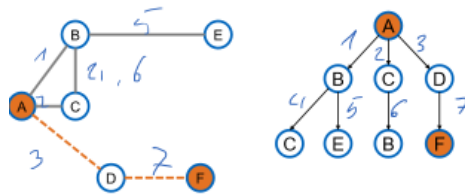
- Random nodes are chosen as long as the algorithm is not stuck  $\rightarrow$  When goal is reached: Done  $\rightarrow$  If stuck: Start all over again
- Uninformed graphs
- Deep dive into the search tree and examine path by path
- Procedure:
  1. No interest in costs at first
  2. Search for possible paths
  3. At the end of a path that is not the target  $\rightarrow$  Backtracking to previous point  $\rightarrow$  Iterative



Breadth First:

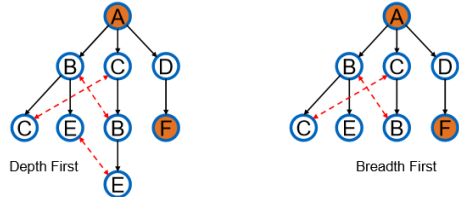
- Uninformed graphs
- Examination of the tree level by level
- Procedure:
  1. Review of each node of a level (alphabetical order)
  2. If not achieved, 2. Level follows

Good luck!



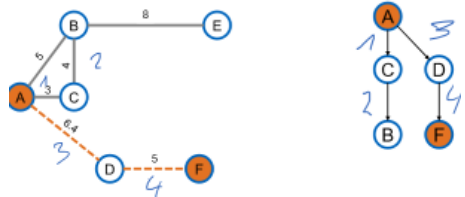
Avoidance of repeat visits:

- Avoidance of repeat visits that did not lead to the goal (e.g. B or C)



Best First:

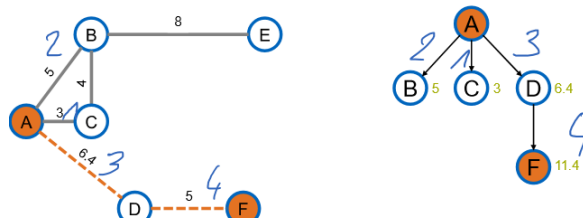
- Informed graphs
- Do immer whatis most in the current n step Always take the **probably** best step → Algorithm does not take the best global step but the apparent best step →
- No revisiting of already visited nodes!!!!



#### Algorithms: Find an optimal way

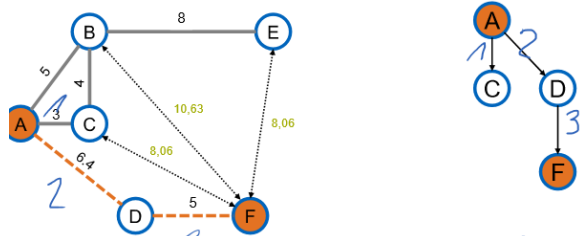
Dijkstra:

- Examine all the shortest paths until the destination is reached
  1. Shortest path → check is goal achieved
  2. Second shortest path → check is goal achieved
  3. ...
- Shortest path defined by route length
- No revisiting!



A\*:

- Compares lower estimate → as the crow flies
- Adds to the air line of a node the distance it takes to reach the node



- $A \rightarrow C: 3 + 8.06 < 6.4 + 5$  but  $C \rightarrow B: 7 + 10.63 > 6.4 + 5$
- $A \rightarrow B: 5 + 10.63 > A \rightarrow C$  or  $A \rightarrow D$

Summarizing:

- Informed graphs: Best First, Dijkstra, A\*
- Uninformed graphs: Breadth First, Depth First
- Completed graphs: Breadth First, Depth First, Best First, Dijkstra, A\*
- Optimal graphene: Dijkstra, A\*

#### Data:

General:

- Large amounts of data, different types of static weightings (distance, CO2, costs, ...) , different types of dynamic weightings (road jam → becomes "more expensive", ...)

Summary:

- Highway geometry, spatial information, traffic rules can be represented by digital data
- Mathematical graphs can be used to represent roads and find paths
- Different route algorithms can be used for route finding
- Real wayfinding brings further problems with it in the wayfinding → not covered in lecture
- Ideas and terms

Good luck!

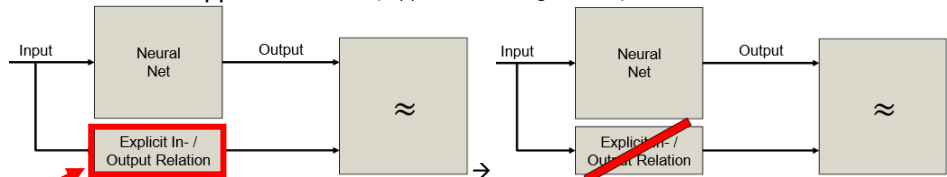
## Chapter VII: Artificial Neural Networks

The aim of this chapter is:

Tasks:

General

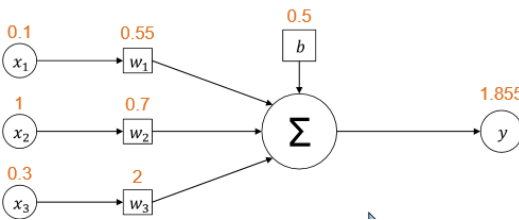
- Understand what a neuron is, how is it defined and is it used?
- Images to Text Converter, Voice Control, Image Coloring, Autonomous Driving
- Neural networks as **approximators** ("approximation algorithms")



- Often no explicit in-/output relationship known
- In most cases, in-/output values can be observed/viewed → Neural networks can learn from them
- Supervised machine learning (**supervised**) (also possible unsupervised)

### First direction to artificial neurons


Simplest approximation:

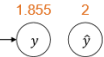


- Linear regression:  $y = f(\vec{w} \cdot \vec{x}, b) = \sum_i w_i x_i + b = w_1 x_1 + w_2 x_2 + \dots + w_n x_n + b$

$\vec{x}$ : Input Vector  
 $\vec{w}$ : Weight Vector  $\vec{y}$ : Output  
 $b$ : Bias  $\hat{y}$ : Training Data

- with  $b$ :
- The basic idea of linear regression is to find the bias  $b$  and the weights  $w$ , so that the resulting function fits well to the data set with input  $x$  and output  $y$  values
- Exemplary representation (left): With 3 inputs and respective values for input, output, bias and weightings

- Forward pass: Passage through neural network:  (Backward passage → part of the backtracking algorithm → Next chapter)

- Loss function:  $L = \frac{1}{2} \cdot (y - \hat{y})^2$  with  $\hat{y}$  = training data (performed according to )

→ **Minimize losses by adjusting bias and weights**

Gradient Descent (Gradient Descent):

- Gradient defines steepest rise → Update weights by one step in the opposite direction → Stops when optimization criterion is reached or after N iterations

$$\nabla L = \begin{pmatrix} \frac{\partial L}{\partial w_1} \\ \frac{\partial L}{\partial w_2} \\ \vdots \\ \frac{\partial L}{\partial w_n} \\ \frac{\partial L}{\partial b} \end{pmatrix}$$

$$\vec{w}_{new} = \vec{w}_{old} - \alpha \cdot \nabla L$$

(with  $\alpha$  = length)

$$\frac{\partial L}{\partial w_i} = \frac{dL}{dy} \cdot \frac{\partial y}{\partial w_i} \quad \frac{\partial L}{\partial b} = \frac{dL}{dy} \cdot \frac{\partial y}{\partial b}$$

$$\frac{dL}{dy} = \frac{d}{dy} \frac{1}{2} \cdot (y - \hat{y})^2 = y - \hat{y} = \Delta y$$

$$\frac{\partial y}{\partial w_i} = \frac{\partial}{\partial w_i} \sum_i w_i \cdot x_i + b = x_i$$

$$\frac{\partial y}{\partial b} = \frac{\partial}{\partial b} \sum_i w_i \cdot x_i + b = 1$$

$$\rightarrow \vec{w}_{new} = \vec{w}_{old} - 0.5 \cdot \nabla L$$

→ Insert new values and then possibly better approximation of  $y$  (see left)

- Problems: Stuck in **local minima** or in **plates**, **oscillations e.g. with symmetrical profiles**, **jumping out of global minimum into a local minimum**

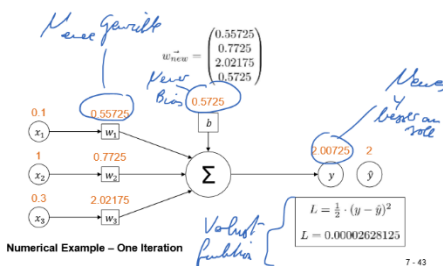
The Neuron:

- Has  $n$  inputs with  $n$  weights
- Has exactly one bias
- Has exactly one activation function

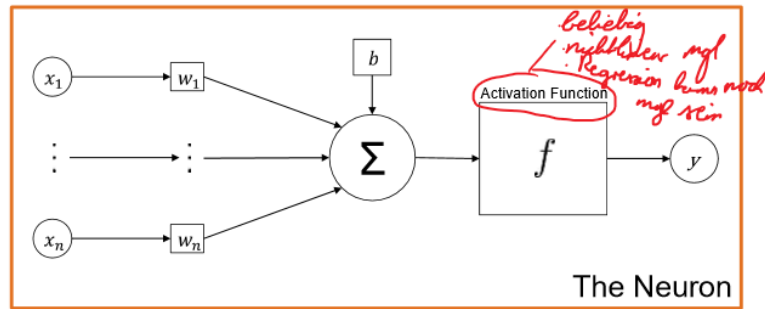
$$y = f(\sum_i w_i \cdot x_i + b)$$

- **Output:**

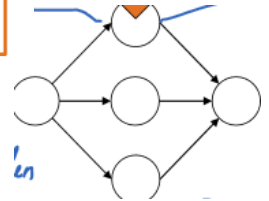
- **Artificial neuron** = Simple extension of linear regression without basic function



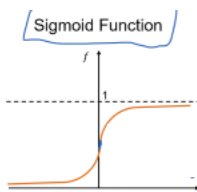
Good luck!



- Multiple neurons can be put together → is called a "layer"
- Each artificial neural network consists of an input layer that reads the data, several **hidden** layers consisting of artificial neurons and an **output layer** that provides the results of the forward pass.



Activation features:



- It is possible to generate nonlinear outputs from purely linear inputs

- Simplest: identity function → simplest proof of linear regression (left); Binary step function → Binary

$$f(x) = \begin{cases} 1, & x \geq 0 \\ 0, & x < 0 \end{cases}$$

classification (right):

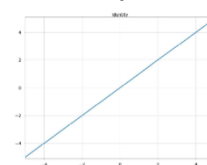
→ derivative vertical → Not imageable

- **In order for a neuron to learn something, the derivation must be imageable!!!**

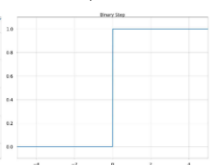
- Solution for binary step: Sigmoid Funktion 0 and 1

separation → continuously derivable →  $f = \frac{1}{1+e^{-y}} = \frac{e^y}{1+e^y}$  → by strong slope:  $f'(x) = f(x) \cdot (1 - f(x))$

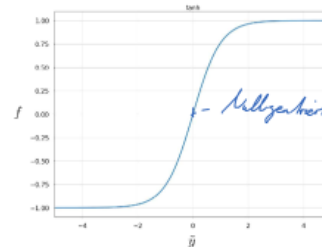
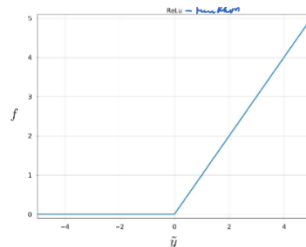
Linear Regression



Binary Classification



- For larger neural networks: ReLu function:  $f = \begin{cases} \tilde{y} & \tilde{y} \geq 0 \\ 0 & \tilde{y} < 0 \end{cases}$  bottom left) or ( $f = \tanh(\tilde{y})$  right)

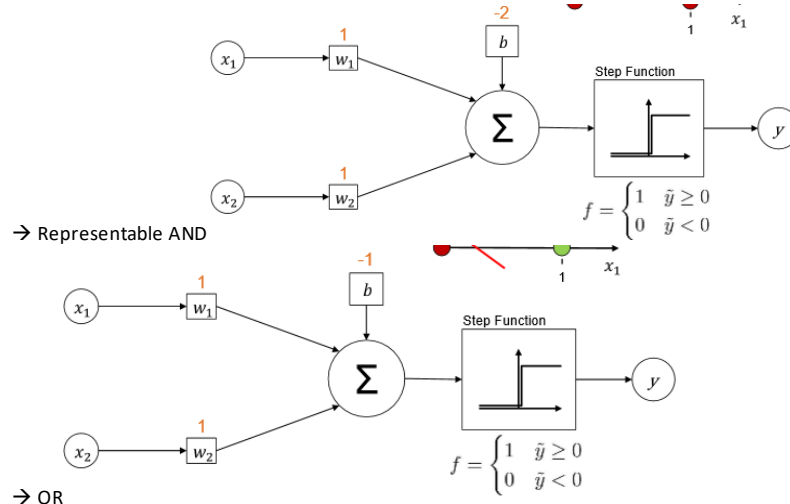


- Advantage Sigmoid (or similar) against via binary: There is never exactly 0 or 1 → distances to the transition area given & Extra statements possible, such as Close to border: New vehicle reads?, Information about truth content of the statement

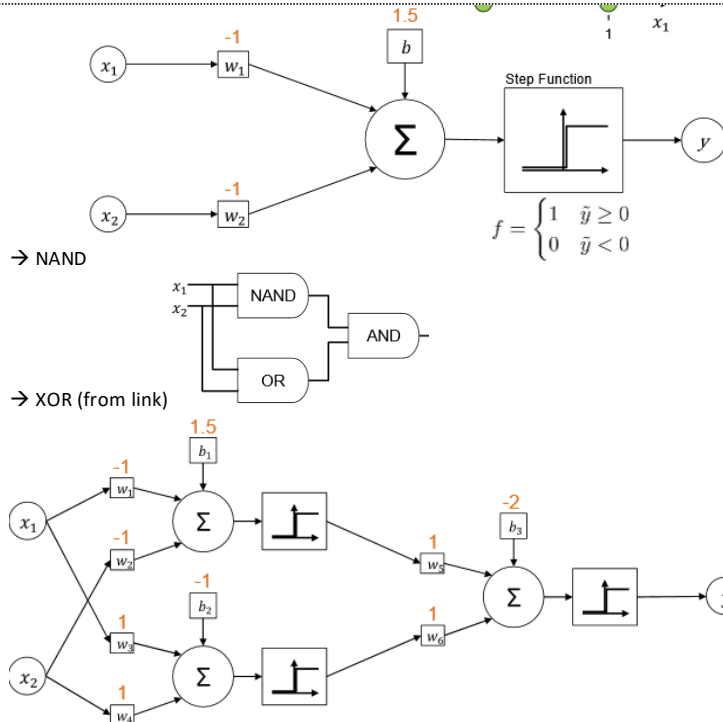
- Epoch: An epoch was implemented when all training vectors were used once to update the weights

### Multilayer networks

Boolean operators:



Good luck!



Summary:

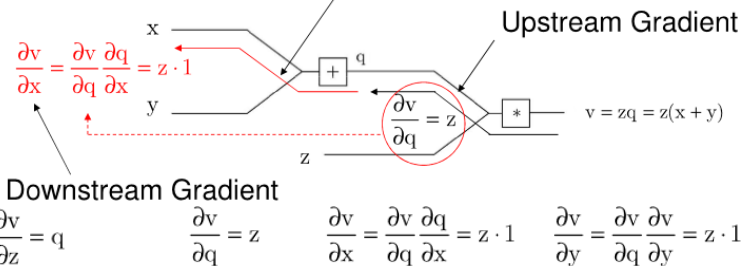
- Neural networks are mathematical tools that can be used to approximate any mathematical function
- Gradient descent is a suitable approach for adjusting the weights
- A single neuron can perform linear regression and binary classifications
- Non-linear problems, multi-layered classifications and regression are best performed with neural networks
- Definitions and ideas

## Chapter VIII: Deep Neural Networks

Backpropagation:

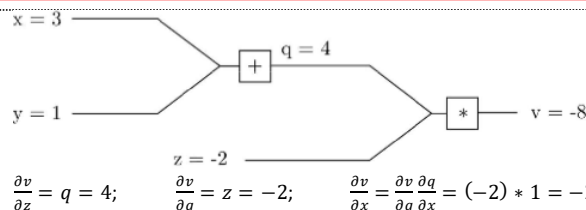
Simple example

Gradient „flows“

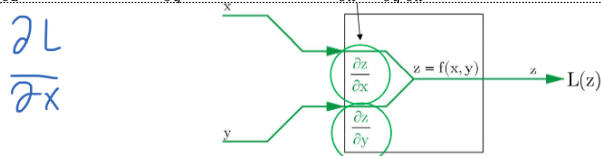


- **Addition:** Downstream gradient factor remains the same:  $\frac{\partial v}{\partial y} = \frac{\partial v}{\partial q} \frac{\partial q}{\partial y} = \frac{\partial v}{\partial q}$
- **Multiplication:** Downstream gradient changes the factor: e.g.:  $\frac{\partial v}{\partial q} = z$

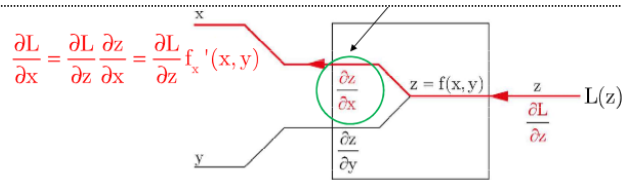
Backpropagation with numbers:



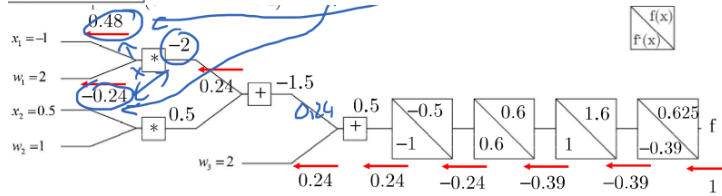
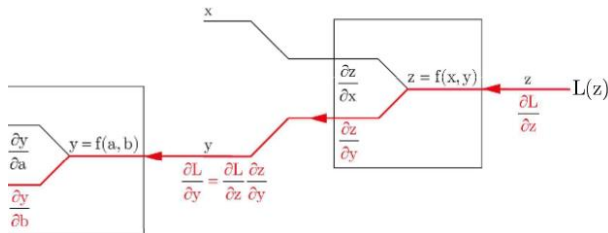
Difference forward and backward:



Good luck!

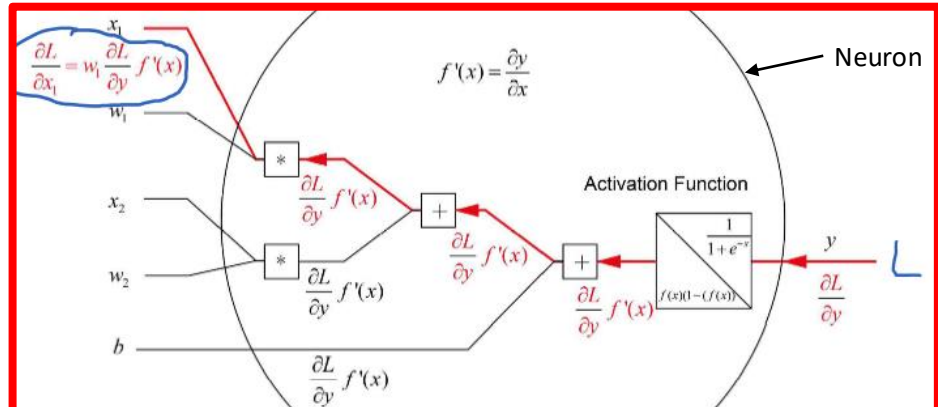


Connection of several neural networks:

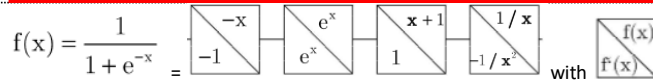


$$\frac{\partial f}{\partial x_1} = 1 * (-0.39) * 1 * 0.6 * (-1) * 1 * 2 = 0.48$$

General:



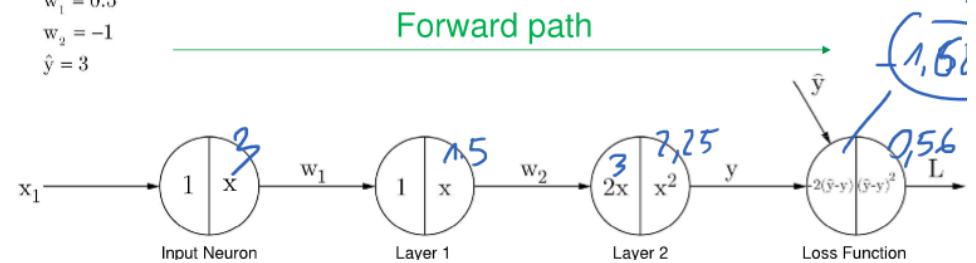
Graphical representation of the Sigmoid function:



Neural chain:

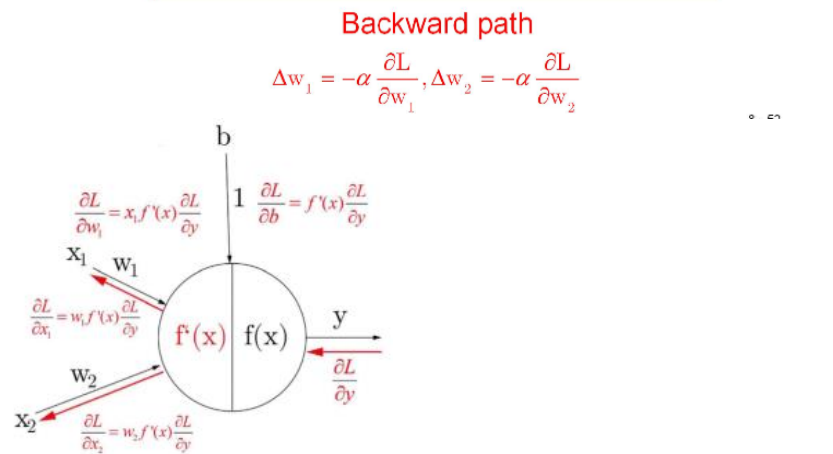
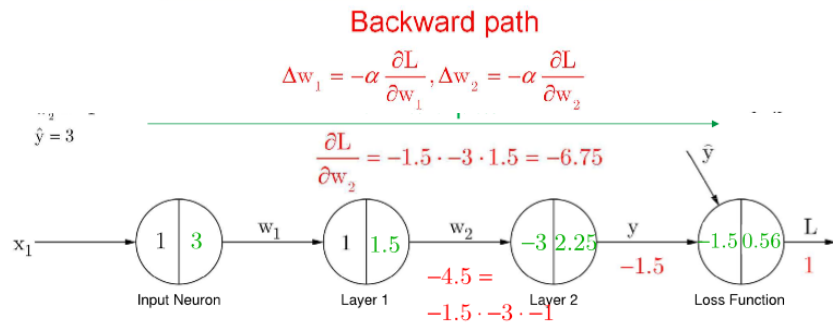
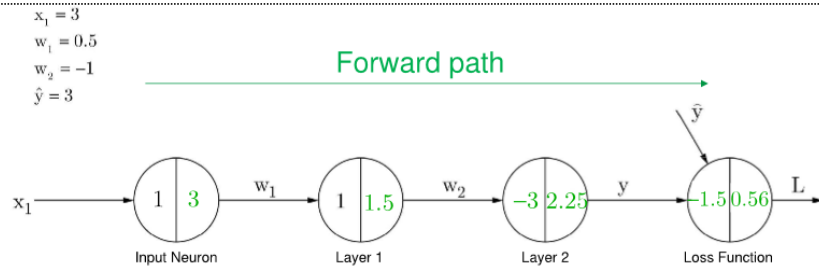
$$\begin{aligned} x_1 &= 3 \\ w_1 &= 0.5 \\ w_2 &= -1 \\ \hat{y} &= 3 \end{aligned}$$

Forward path

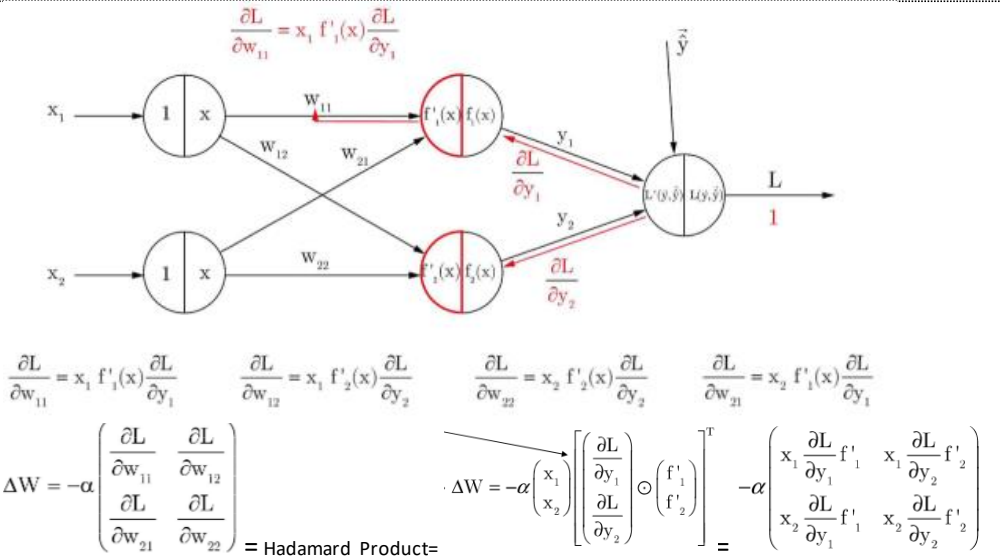


→ Solution

Good luck!

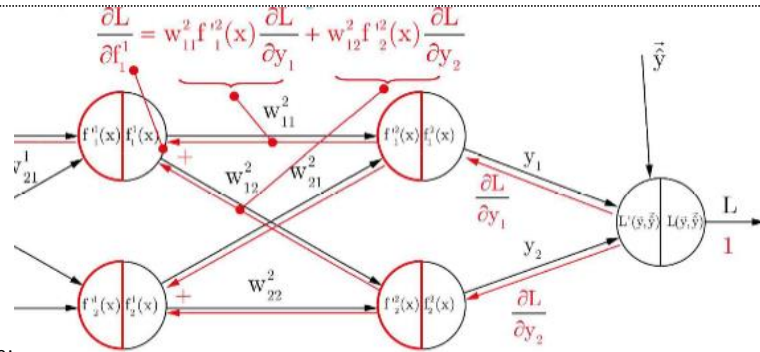


Neural network:





Good luck!



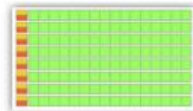
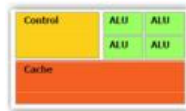
- Note:

- CPU: Small Computational Density, Complex Logical Control Mechanisms, Large Cache, Low Latency Tolerance
- GPU: High computational density, Many calculations per memory access, Parallel Computation Optimizer, Low Latency Tolerance

CPU

vs

GPU



Calculation hardware:

## Neural Networks

Activation functions:

leaky/parametric Relu

Sigmoid

ReLU

tanh

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

- Sigmoid function:

Two main problems: Causes disappearing gradient  $\rightarrow$  Gradient about 0 stops at very large or very small  $x$ -values Network  $\rightarrow$  output is not zero centered  $\rightarrow$  All gradients positive or all negative  $\rightarrow$  Inefficient weight updates

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

$$f'(x) = \frac{4}{(e^x + e^{-x})^2}$$

- Tangent hyperbolicus:

Better than Sigmoid: output is zero-centered, but also causes a vanishing gradient

$$f(x) = \begin{cases} x, & x \geq 0 \\ 0, & x < 0 \end{cases}$$

$$f'(x) = \begin{cases} 1, & x \geq 0 \\ 0, & x < 0 \end{cases}$$

- ReLU (Rectified Linear Unit):

Most used activation function (efficient computation, converges very quickly, does not activate every neuron at the same time);

**Problem:** Gradient = 0 for  $x < 0 \rightarrow$  can cause disappearing gradient & not zero centered

**Application:** Most used in hidden layers, positive bias during initialization to obtain an Active ReLU

$$f(x) = \begin{cases} x, & x \geq 0 \\ ax, & x < 0 \end{cases}$$

$$f'(x) = \begin{cases} 1, & x \geq 0 \\ a, & x < 0 \end{cases}$$

- Leaky ReLU:

Removes 0 portion of ReLU by inserting a small slope  $\rightarrow$  more stable than ReLU, but extends the function by another parameter

Features: efficient computation, converges very quickly, does not "die", parameters can also be learned through the network

$$f(x) = \begin{cases} x, & x \geq 0 \\ a(e^x - 1), & x < 0 \end{cases}$$

$$f'(x) = \begin{cases} 1, & x \geq 0 \\ ae^x, & x < 0 \end{cases}$$

- Exponential Linea Unit (ELU):

Take advantage of ReLU and Leaky ReLU; Calculation requires  $e^x$

- Softmax: Type of Sigmoid function  $\rightarrow$  Divides by the buzzer of all outputs

$\rightarrow$  Allows percentage representation

$$f(x) = \frac{e^{y_i}}{\sum_{k=1}^K e^{y_k}}$$

Rule of thumb:

- Sigmoid/Softmax: Classifications
- Sigmoid / tanh: danger of disappearing gradients
- ReLU: is the most used nowadays, but should only be used in hidden layers
- Start with ReLU if no optimal results use Leaky ReLU or ELU

Fully meshed layers:

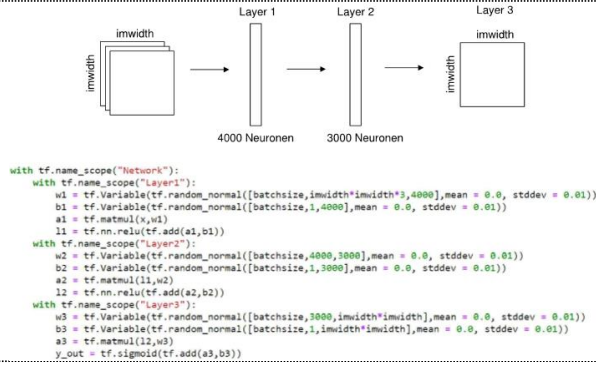
$$\vec{y}^T = (y_1 \ y_2 \ y_3) = f(W^1 \vec{x} + \vec{b})^T = f(\vec{x}^T W^{1T} + \vec{b}^T)$$

Stochastics vs. Batch:

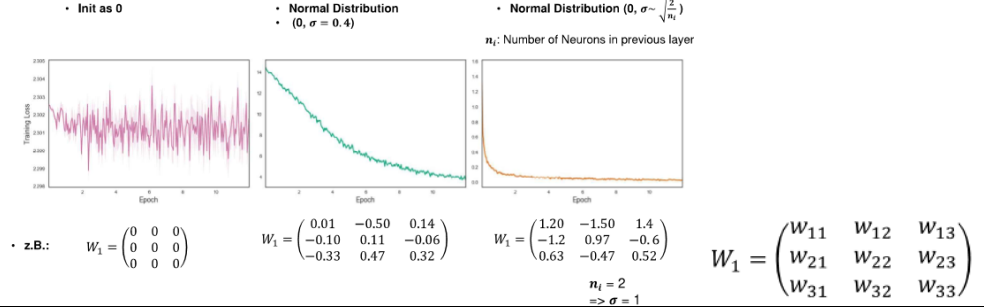
- $\rightarrow$  Read Evnt in lecture / Internet
- Stochastic training can miss local minima all inputs due to random selection
- Generally takes longer
- Small batches can find at least faster, but may require more computing power

Good luck!

Python example with tensorflow:



Weight initialization:

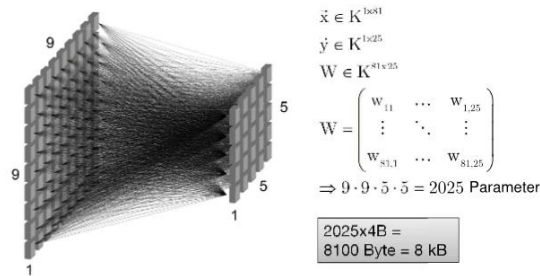


Summary:

- Training of a neural network:
  1. Preparing input data
  2. Prepare the data pipeline to get it into the neural network
  3. Define the network
  4. Initialize all variables
  5. Train the network
  6. Monitor training
- Use of computational graphs to calculate local gradients in all mathematical functions
- Calculate weight updates through backpropagating
- The activation functions and their possible applications
- Using batches to train neural networks
- Correct initialization of weights
- Steps to train a neural network

## Chapter IX: Convolutional Neural Networks

Fully Connected Layer:



→ Double dimension:

$18 \times 18 \times 10 \times 10 \times 4B =$   
 $129600 \text{ Byte} = 0.1296 \text{ MB}$

FullHD x FullHD?  
1920x1080x3 x 1920x1080x3x 4B =  
1.54729341e+14 B =  
154.293 GB =  
154 TB =  
Internet Traffic of 12800 people in 2018  
 $\Rightarrow$  For images we need something else

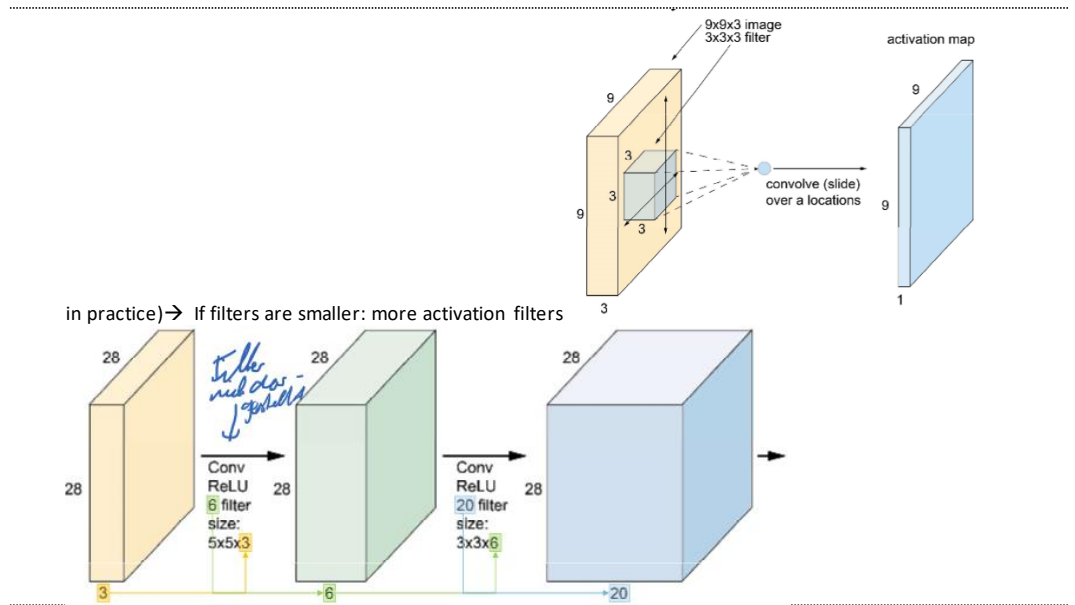
Mathematical calculation:

$(f * g)(x) = \int_{\mathbb{R}^n} f(\tau)g(x - \tau)d\tau$

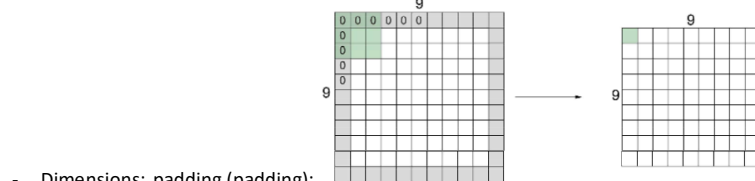
General:

- Number of input layers Must be the same as filter Anumber (or filter number less, but this rarely or not happens)

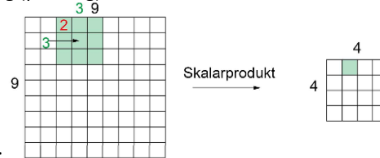
Good luck!



Terms → Very cryptically look up the script → on the Internet:

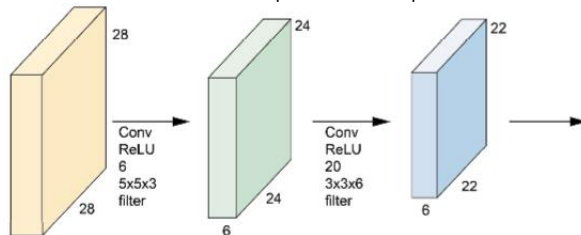


- Dimensions: padding (padding):



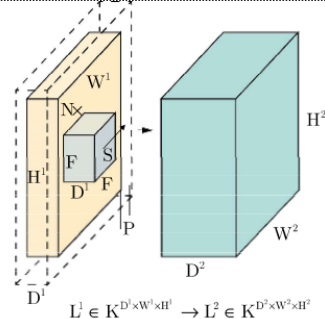
- Dimensions: stride:

- Without padding, layer volume would shrink with each step → Can cause problems in the very deep neural



network → Using padding:

Formulas:



N : Number of Filters

F : Filtersize

S : Stride

P : Padding

New Dimensions:

$$W^2 = \frac{W^1 + 2P - F}{S} + 1$$

$$H^2 = \frac{H^1 + 2P - F}{S} + 1$$

$$D^2 = N$$

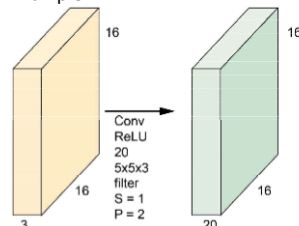
Keep Dimensions:

$$F = 3 \Rightarrow P = 1$$

$$F = 5 \Rightarrow P = 2$$

$$F = 7 \Rightarrow P = 3$$

- Example:



$$W^2 = \frac{16 + 2 \cdot 2 - 5}{1} + 1 = 16$$

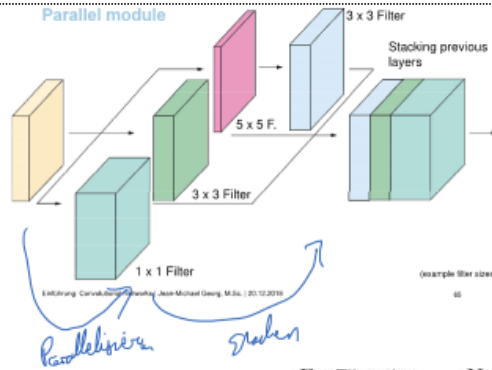
$$H^2 = \frac{16 + 2 \cdot 2 - 5}{1} + 1 = 16$$

$$D^2 = 20$$

→ Number of parameters in layer: 5x5x3+1 = 76 per filter → 20\*76=1520 parameters

Good luck!

Parallel layers:



Python example:

```
with tf.name_scope("ConvLayer"):
    w = tf.Variable(tf.random_normal([size,size,ind,count],mean=0,stddev = 0.01))
    b = tf.Variable(tf.zeros([batchsize,imwidth,imwidth,count]))
    conv = tf.nn.conv2d(inlayer,w,strides=[1, stride, stride, 1],padding='SAME',b))
```

F : Filtersize

N : Number of filters

S : Stride

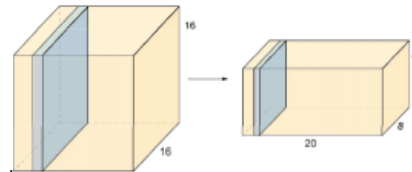
P : Padding

„Same“ keeps dimensions

### Unification (pooling)

General:

- In principle = Reduces spatial output dimensions



- Operates independently on each activation card

Max union (pooling):

1	4	0	6
-3	2	5	9
-8	3	-8	-1
2	1	-5	-4

Stride = 1

4	9
3	-1

- Find maximum value in each area

1	4	0	6
-3	2	5	9
-8	3	-8	-1
2	1	-5	-4

Stride = 1

4	5

Average Pooling:

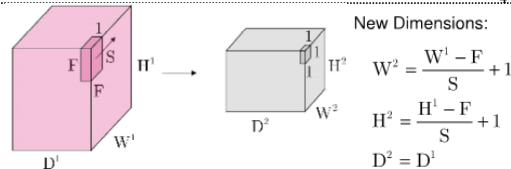
1	4	0	6
-3	2	5	9
-8	3	-8	-1
2	1	-5	-4

Stride = 1

1	5
-0.5	-4.5

$$\text{with } \frac{1+4+(-3)+2}{4} = 1$$

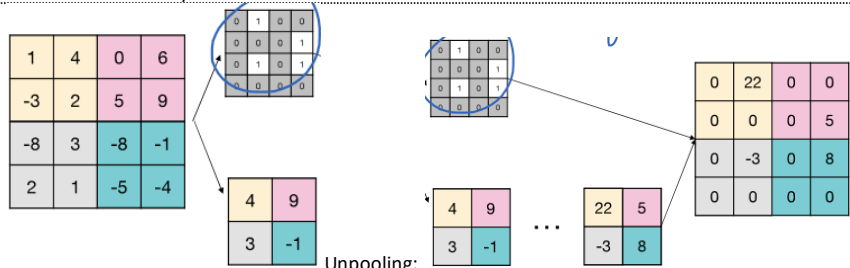
Dimensions:



Advantage:

- Large areas have to be used for large images → would be computationally expensive → Pooling reduced Image → of smaller filters necessary

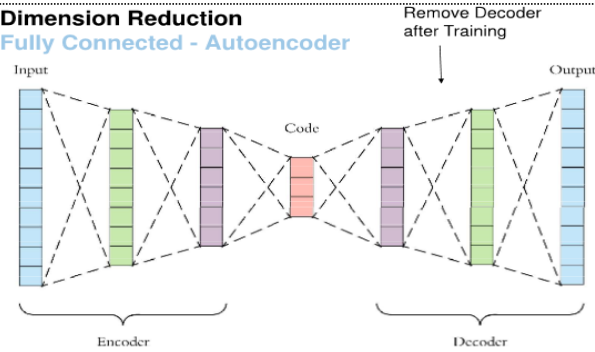
Unpooling:



- Pooling:

Good luck!

### Dimension Reduction Fully Connected - Autoencoder



CNN in action:

- Weather detection
- Autonomous driving

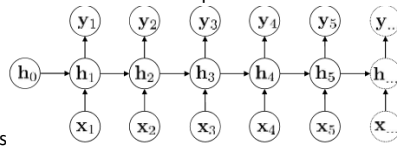
Optimisation:

- Stochastic gradient descent → Problem: Loss may change quickly to one dimension and slowly to the other; local minima and saddle points
- Nesterov + Momentum
- AdaGrad and RMDProp
- Adam

## Chapter X: Recurrent Neural Networks

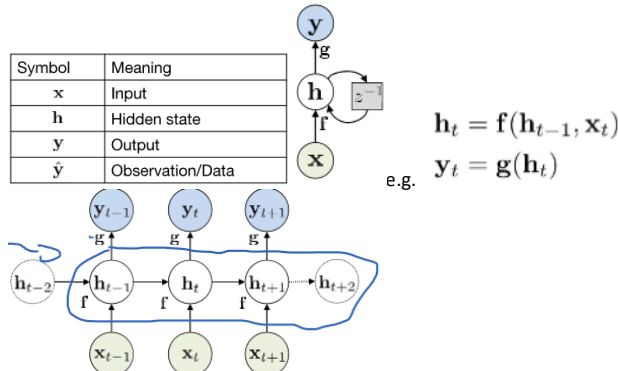
Motivation:

- Often not all information is immediately visible → Details are often hidden in the scene, but reading all data would require too many **parameters** → Add parameters and memory to take advantage of important

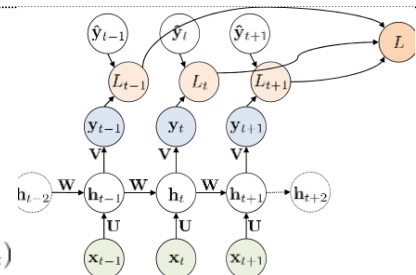


features of past sequences

Terms:



Loss function:



- Buzzer across all time steps  $L = \sum_t L_t = \sum_t L(y_t, \hat{y}_t)$

$$L_t = \frac{1}{2} (y_t - \mu(x_t, \dots))^2$$

- For regression:
- Supervised learning → Feedback from the output can be used for learning
- Regulates only non-split parameters

Teacher forcing:

Regulation with dropout:

Summary of simple RNN and backpropagation:

- Meaning of the slides up to this point were difficult for me to understand
- An RNN is just a state-space model with **free parameters** that we want to learn
- Gradient descent usually uses backpropagation, but: Computation graph is deep (high computational and memory overhead) → Uses truncated sequences (leads to bias in the gradient)
- When output is used as input, backpropagation can be avoided → **Use of teacher forcing**

Vanishing or "exploding" gradient:

$$h_{t+1} = a \cdot h_t$$

$$h_{t+k} = a^k \cdot h_t \rightarrow \frac{\partial h_{t+k}}{\partial h_t} = a^k$$

- Adopted:

Good luck!

$$\lim_{k \rightarrow \infty} \frac{\partial h_{t+k}}{\partial h_t} = \begin{cases} \infty & \text{if } a > 1 \\ -\infty & \text{if } a < -1 \\ 0 & \text{if } |a| < 1 \\ 1 & \text{if } a = 1 \\ \text{undefined} & \text{if } a = -1 \end{cases}$$

*this leads to the gradient vanishing or exploding problem*

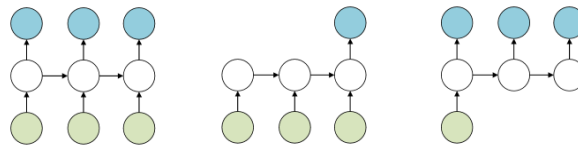
- Follow:

Summary: Challenge of large dependencies:

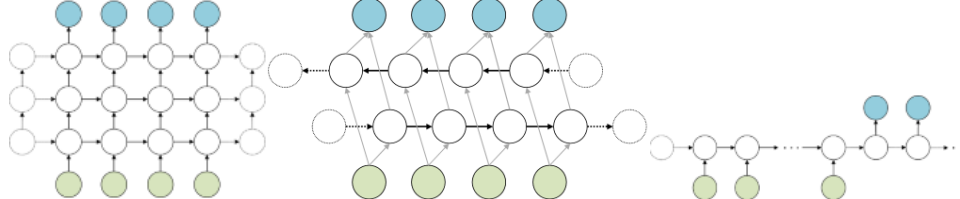
- Reusing the same weights can result in a very large or very small gradient  $\rightarrow$  slowing down training
- Large gradients must be reduced  $\rightarrow$ , e.g. by cutting standards or by cutting out individual entries
- Small gradients can be processed by using inputs from several steps in the past.
- A good initialization of the repetitive weights can avoid small or large gradients at the beginning of the training

Different RNN structures:

- Many to Many (left)  $\rightarrow$  For the approximation of the dynamics of physical systems
- Many to One (middle)  $\rightarrow$  classification of a video
- One to Many (right)  $\rightarrow$  Description of a picture with sentences



- Multilayer RNN (left)
- Bidirectional RNN  $\rightarrow$  Integration of future information may be useful (e.g. handwritten knowledge) (middle)
- Sequence to sequence (left)  $\rightarrow$  for example: translation into different languages

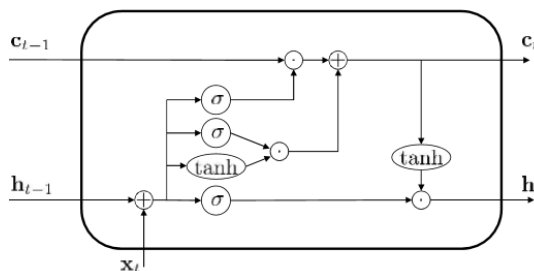


Long short-term memory:

- Idea of not updating the whole hidden state each time
- Protect the state from being overwritten by useless information
- Be selective in
  - What to write (input gate)  $i_t = \sigma(W_i h_{t-1} + U_i x_t + b_i)$
  - What to read (output gate)  $o_t = \sigma(W_o h_{t-1} + U_o x_t + b_o)$
  - What to forget (forget gate)  $f_t = \sigma(W_f h_{t-1} + U_f x_t + b_f)$

$$c_t = f_t \odot c_{t-1} + i_t \odot \tanh(W_h h_{t-1} + U_h x_t + b_h)$$

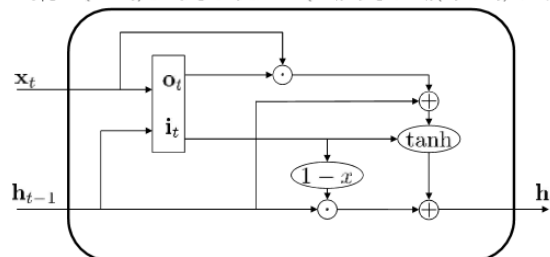
$$h_t = o_t \odot c_t$$



Gated Recurrent Unit:

- Same idea using gates, here
  - Input/write  $i_t = \sigma(W_i h_{t-1} + U_i x_t + b_i)$
  - Forget  $f_t = 1 - i_t$
  - Output/read  $o_t = \sigma(W_o h_{t-1} + U_o x_t + b_o)$

$$h_{t+1} = (1 - i_t) \odot h_{t-1} + i_t \odot \tanh(W_h h_{t-1} + U_h (o_t \odot x_t) + b_h)$$



Summary:

- Many structures and activations possible: Deep RNN, Bidirectional RNN, ...
- Previous knowledge difficult to identify which variant works best: **currently LSTM and GRU** most widely used

Good luck!

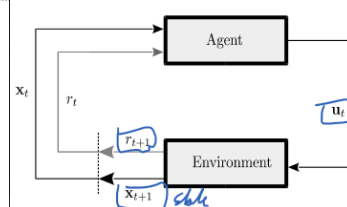
- RNN = Dynamic Systems
- Application: Whenever data occurs in sequences (e.g. audio, video, text, speech, ...)
- Timely unfolding is similar to training a deep neural network, but difficulty occurs: biased gradient, large and deep computational graphs, small or large gradients
- Different RNN structures can be favored for different situations

## Chapter XI: Reinforcement Learning

Basic ideas:

- Supervised learning: Learning with **weighted data**, → e.g. image classifications using weighted data sets and deep neural networks (input and output data known → Search near relations in between)
- Unsupervised learning: Learning with unweighted **data**, → e.g. clustering with K-means (only one record exists → Search for structure in the data)
- Reinforcement Learning: "Solving to Get Rewards"

General:



- Agent examines environment and makes decisions  $As \rightarrow$  a result, the environment he perceives changes  $x_t$  and he receives reward or consequence  $r_t$
- **Agent**: Who acts (e.g. computer)
- **Environment**: Where does he act
- **Reward**: a scalar signal that the agent receives → Based on how well the agent performs its task → In our case, we determine what the *belohnung* looks like
- **State**: A signal that describes the environment (e.g. speed or position of a robot arm)
- **Action**: The agent determines his action → depending on his specifications (In chess: What the computer has to "see", position of all pieces on the chessboard)
- **Goal of the RL**: To train the agent to collect as much reward as possible

Motivation:

- Problems can become very complex → Difficult model to reduce, so reality is well described
- Generally rather nonlinear models with high state-space models → often difficult and require a team of experts

Summary:

- RL describes a high level idea of learning to make good decisions, repeat a task and receive a reward. **It's not an algorithm!**
- Elements of an RL: agent, environment, reward, states

Static example for preparation:

Probability mass function (PMF), 2 dice example  $P(x, y)$

$x$  random variable: 1 if (sum of eyes) > 5, else 0

$y$  random variable: 1 if at least one dice rolled a 5, else 0

Dice1 \ Dice2	1	2	3	4	5	6
1	2	3	4	5	6	7
2	3	4	5	6	7	8
3	4	5	6	7	8	9
4	5	6	7	8	9	10
5	6	7	8	9	10	11
6	7	8	9	10	11	12

$$P(x=0, y=0) = \frac{10}{36} = \frac{5}{18}$$

$$P(x=0, y=1) = 0$$

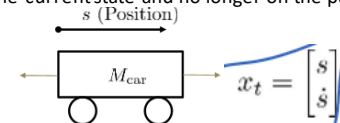
$$P(x=1, y=0) = \frac{15}{36}$$

$$P(x=1, y=1) = \frac{11}{36}$$

$$\mathbb{E}^P[x] = \sum_i P(x_i) x_i = \frac{1}{6} \cdot 1 + \frac{1}{6} \cdot 2 + \dots = \frac{21}{6} = 3.5$$

Markov's decision:

- Expected value:
- Markov, when the transition to the next state depends only on the current state and no longer on the past!



- Markov condition: All necessary information available: Example:

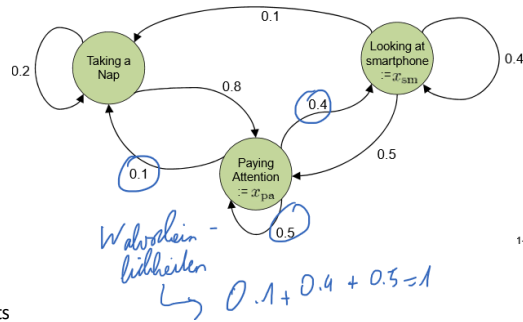
A state is Markov, if

$$P(x_{t+1}|x_t) = P(x_{t+1}|x_t, x_{t-1}, \dots, x_0)$$

- Markov process: random process until there is a Markov state for each state

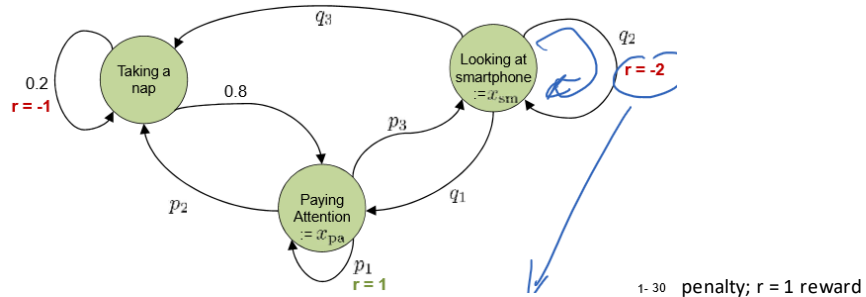


Good luck!



Example: Sleeping students

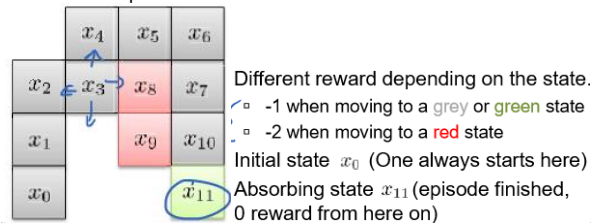
- Markov decision process: A Markov process with additional rewards and the possibility to influence the probabilities with:  $r = -2$  bad penalty,  $r = -1 \Rightarrow$



- Strategy: maximum future reward  $\rightarrow$  The example process is bad because there is only one reward and no goal/end state

$$\sum_{t=0}^{\infty} \gamma^t \cdot r_t ; \gamma \in [0, 1]$$

- Maximization: with  $\gamma^t$  = serves as a weight that weights the next state
- Another example:



### Value Function, Q-Learning

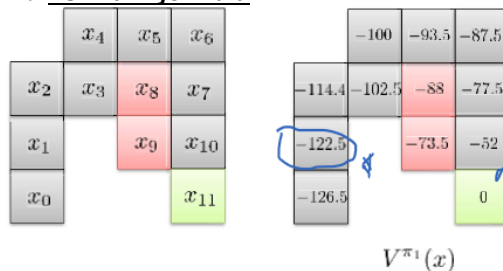
Value function:

- Depending on condition and rules; Returns the expected future reward; Starts in state  $x$  and is always followed by a rule  $\pi$

$$V^{\pi}(x) = \mathbb{E}^{\pi} \left[ \sum_{\tau=t}^{\infty} \gamma^{\tau-t} r_{\tau+1} | x_t = x \right]$$

$0 < \gamma \leq 1$  discount factor

- With **Bellmann formula**:  $\mathbb{E}^{\pi} [r_{t+1} + \gamma \cdot V(x_{t+1}) | x_t = x]$



- E.g. -122.5: Expected value if you contest the field completely with random steps  $\rightarrow$  Amount of the final reward value
- Can be used when all probabilities are known (motion and rule) by iteration of the Bellmann formula for all states:

Good luck!

$$V(x_7) = \frac{1}{4} \cdot (-1 + (-77,5)) + \frac{1}{4} \cdot (-1 + (-87,5)) + \frac{1}{4} \cdot (-1 + (-88)) + \frac{1}{4} \cdot (-1 + (-52))$$

1- 43

schritt-  
wahrscheinlichkeit      schritt       $V(x_1)$        $V(x_t)$       ...

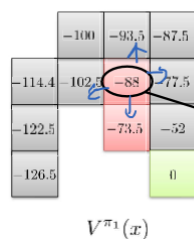
$$V(x_0) = \frac{3}{4}(-1 + V(x_0)) + \frac{1}{4}(-1 + V(x_1))$$

$$V(x_1) = \frac{1}{4}(-1 + V(x_0)) + \frac{2}{4}(-1 + V(x_1)) + \frac{1}{4}(-1 + V(x_2))$$

⋮

- Rule improvement:

$$u_{\text{greedy}} = \arg \max_u \mathbb{E}[r_{t+1} + \gamma V^\pi(x_{t+1}) | x_t = x, u_t = u]$$



$$\text{für } \rightarrow: -1 - 77,5 = -78,5$$

$$\text{für } \downarrow: -1 - 73,5 = -74,5 \rightarrow \text{am Meiste Belohnung}$$

$$\text{für } \leftarrow: -1 - 102,5 = -103,5$$

$$\text{für } \uparrow: -1 - 93,5 = -94,5$$

neue Strategie determiniert  
immer das Beste

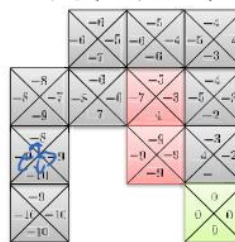
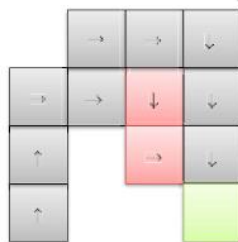
Current value function:

- Depending on the state, the next action and rules; Returns the expected future reward; Starts in state  $x$ , then selects an action  $u$  and is followed by a rule  $\pi$

$$Q^\pi(x, \pi(x)) = V^\pi(x)$$

$$Q^\pi(x, u) = \mathbb{E}^\pi \left[ \sum_{\tau=t}^{\infty} \gamma^{\tau-t} r_{\tau+1} \mid x_t = x, u_t = u \right]$$

$$= \mathbb{E}^\pi [r_{t+1} + \gamma Q(x_{t+1}, \pi(x_{t+1})) \mid x_t = x, u_t = u]$$



Q(x, u) in PC Memory:

	↑	←	↓	→
$x_0$	-9	-10	-10	-10
$x_1$	-8	-9	-10	-9
⋮	⋮			
$x_{10}$	-3	-4	-1	-2
$x_{11}$	0	0	0	0

- Disadvantage value function: Must know next state

- Advantage Q function: Knows all the information needed to execute the rules, but does not need to know how big the movement probabilities are

$$u_{\text{greedy}}(x) = \arg \max_u \mathbb{E}[r_{t+1} + \gamma V^\pi(x_{t+1}) \mid x_t = x, u_t = u]$$

$$u_{\text{greedy}}(x) = \arg \max_u Q(x, u)$$

- Disadvantage: More memory and time to train

Summary:

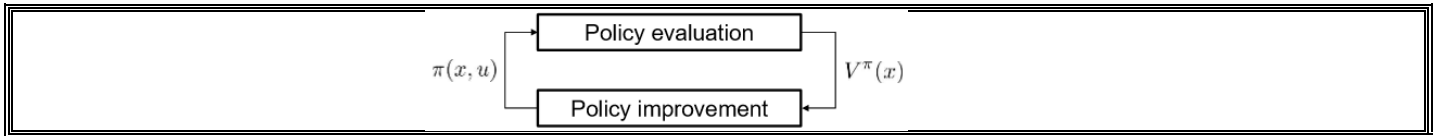
- With Bellmann formula, the value function or the action value function can be learned

- If the function is present, the rules can be improved If the value function is used, the

→ knowledge of the motion dynamics is still **needed**

→ With action value function, the best values can be easily read, but more memory is needed

Good luck!

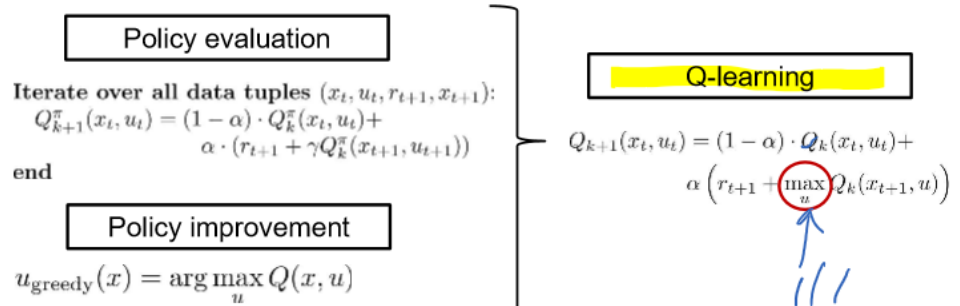


Model-free learning:

- Prerequisites: All states and actions have a probability of  $>0$  that they can be visited  $\rightarrow$  Problem: In greedy rule, the other actions and states must also be considered; The learning rate decreases; We learn infinitely long
- You have to look at new ways to learn  $\rightarrow$  Greedy Bad, as deterministic

Q function:

- Solution: Use Greedy approach, but at the same time give the others a low probability  $\rightarrow$  called  **$\epsilon$ -Greedy**
- Combination of rule evaluation and rule improvement



- Learning without a model

Summary:

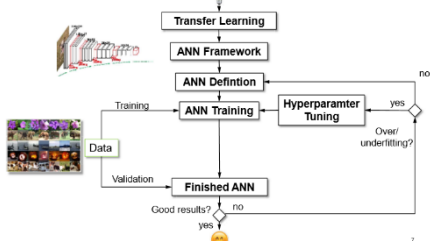
- Procedure: 1. Learn value function or state value function to the current rule (use Bellmann equation)
- 2. Use point 1 to improve
- 3. Repeat
- Deterministic rule problem when learning the value function or action value function  $\rightarrow$   **$\epsilon$ -Greedy**
- No need to train the value function for full convergence previous  $\rightarrow$  updates occurs possible
- Q-Learning can be used to learn the optimal greedy rule  $\rightarrow$  Voting algorithm in discrete Markovs decision-making processes

Expectation horizon for exam of this chapter:

- Bellmann equation
- How the calculation of the value function works in discrete Markovs decision-making processes
- How the calculation of the action value function works in discrete Markovs decision-making processes
- How to get a new greedy rule for a given value or action value function
- Calculation of the Q-Learning Update Step
- Understand why a deterministic rule does not work in a deterministic learning environment

## Chapter XII: AI Development

AI Development Pipeline:



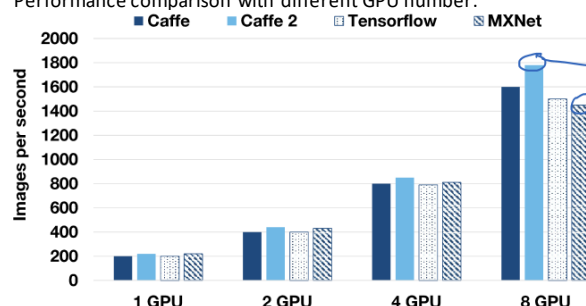
Transfer what you learn:

Frameworks:

1. Type of problem
2. Machine learning method usable: regression, classification, clustering
3. Need deep learning?
4. Choice of programming language
5. Enough data available?
6. Is the data variable?
7. A little GPU power available to get first results?
8. Do the hyperparameters need to be varied?
9. Need multiple GPUs? (If problem too big)
10. What kind of hardware is required for the final evaluation?
11. Hard connectable with other hardware?

- Vary layers of existing networks
- Finetuning: Train the entered network  $\rightarrow$  Start with pre-tested weights
- Training without preset: Start with random weights

- Benefits  $\rightarrow$  a lot of work can be saved
- Scikit-learn Library, Matlab Tensorflow, Keras, Caffe and Caffe 2, mxnet,
- Performance comparison with different GPU number:

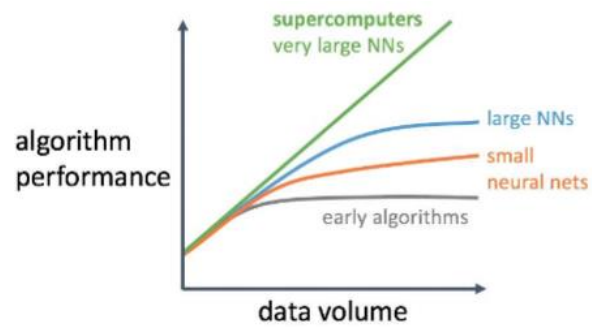


Data:

- Used for training the algorithm

Good luck!

- 
- More and different data is needed for non-overfitting of the training
  - More data needed for good regression, clustering and classification



- Labeled Data: What does the data contain?
  - How can machine learning be applied to little data?
    - Use data → customization Mirroring, rotating, scaling, moving, changing (noise, ...)
  - **Split data: 60% training, 20% validation, 20% test database**
  - **Advantages over CPU: Faster training, faster completion, better results**
  - Focus: Not overfitting the training, Good evaluation at the end → to achieve this, various parameters can be adjusted → Hyperparameters "→magic" behind the networks
- 

GPU for Deep Learning:

Hyperparameter Training: