



Artificial Intelligence and Data Analytics for Engineers (AIDAE)

Lecture 8
June, 26th

Andrés Posada
Anas Abdelrazeq
Marco Kemmerling
Vladimir Samsonov

Today's Lecturer

Artificial Intelligence and Data Analytics for Engineers

Overview Lectures 1 – 4



1

Introduction to Data Analytics and Artificial Intelligence in Engineering: Organizational matters (e.g. exam, exercises, dates). Goals, Challenges, Obstacles, and Processes.



2

Introduction into the primary programming language of the lecture, Python: Syntax, libraries, IDEs etc. Why is Python the *lingua franca* of the Data Scientist?



3

Data Preparation: Cleansing and Transformation. How do real world data sets look like and why is cleaning and transformation an integral part of a Data Scientist's workflow?



4

Data Integration: Architectures, Challenges, and Approaches. How can you integrate various data sources into an overarching consolidating schema and why is this important?



5

Data Representation: Feature Extraction and Selection. How to pick relevant features for the task at hand. Manual vs automatic methods. What is the curse of dimensionality?



6

Data-Driven Learning: Supervised (Classification, Regression) methods and algorithms. What is an artificial neural net? What methods are there for evaluation of your model?



7

Data-Driven Learning: Unsupervised (Clustering) methods and algorithms. How can machines learn without labels? What methods are there for evaluation of your model?



8

Environment-Driven Learning: Reinforcement Learning

9

Data-Driven Learning: Artificial Neural Networks

Recap Lecture 8

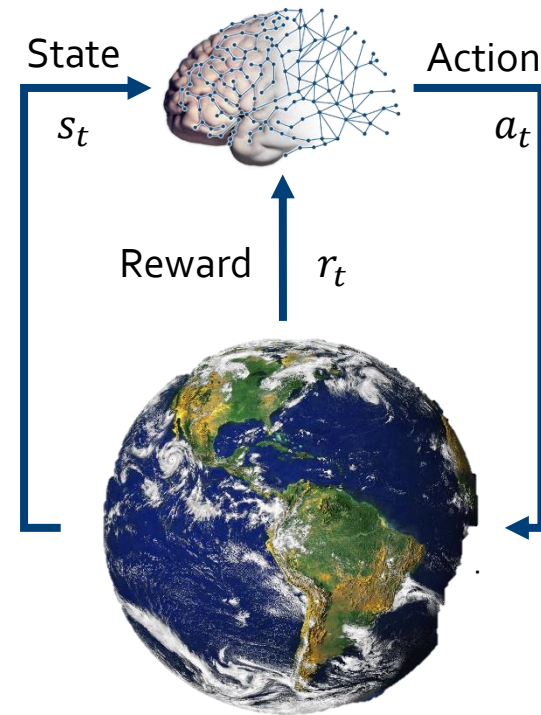
What is reinforcement learning?



Main Idea

The agent learns from success and failure through reward and punishment.

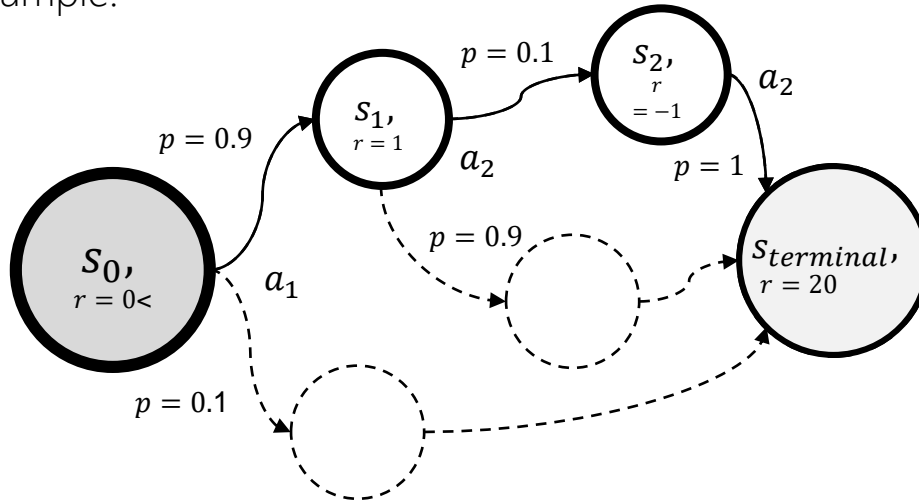
- Goal of the agent is defined by the reward function.
- Agent receives Feedback in form of a reward
With RL, the agent does not know the reward function!
- Agent must (learn) to act in such a way that the expected future reward is maximized
- All learning is based on samples of states, actions and rewards





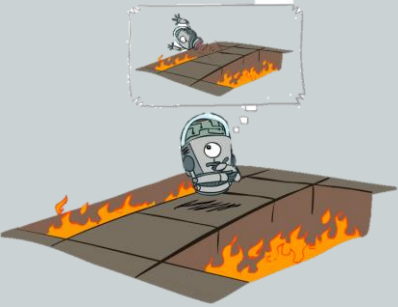

Reinforcement learning is formulated as a Markov decision-making process (MDP). This process is a tuple $\langle S, A, P, R, \gamma \rangle$

Example:

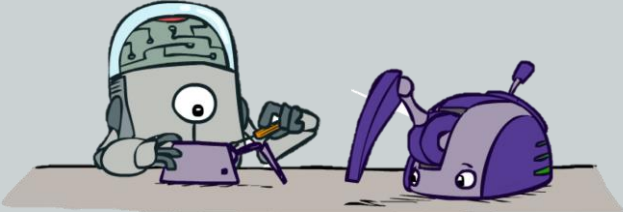
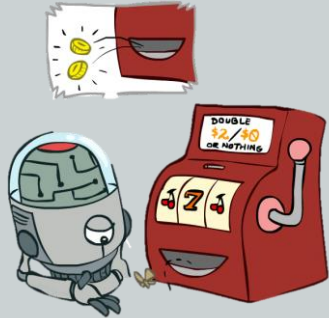


- S is a finite set of states
- A is a finite set of actions
- P is a transition matrix
 - $P_{ss'}^a = \mathbb{P}[S_{t+1} = s' | S_t = s, A_t = a]$
 - Probability that the action a leads from the state s to the state s'
- R is a reward function, $R_s^a = \mathbb{E}[R_{t+1} | S_t = s, A_t = a]$
- γ is a discount factor $\gamma \in [0, 1]$

Case Distinctions

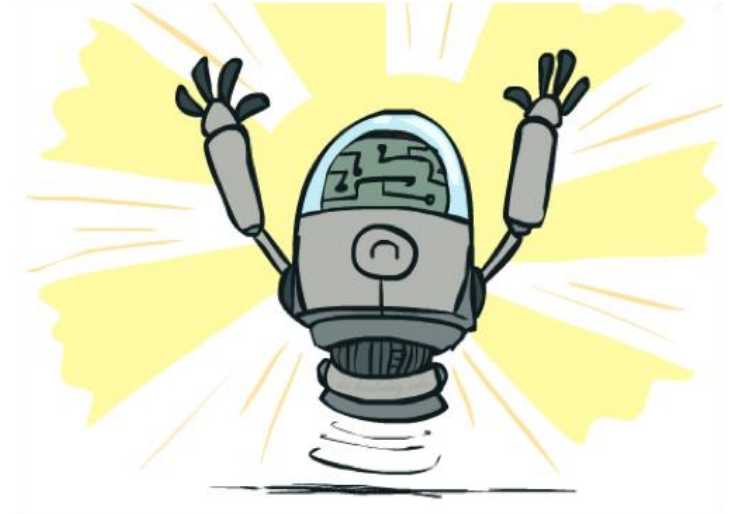
Offline Solution (MDPs)	Online Learning (RL)
	
<p>Idea:</p> <ul style="list-style-type: none">• The state space, the action model and the transition model are simulated• Definition as MDP	<p>Idea:</p> <ul style="list-style-type: none">• Agent executes actions (strategy may exist)• Rewards will be experienced• A model may be created
<p>Procedure:</p> <ul style="list-style-type: none">• e.g. Dynamic Programming• Synchronous DP• Asynchronous DP	<p>Procedure:</p> <ul style="list-style-type: none">• Model-based learning• Model-free learning• Active or passive learning

Case Distinctions

Model-based learning	Model-free learning
	
Model-based Idea: <ul style="list-style-type: none">• Learning an approximated model from experience• Solve as if the learned values were correct	Model-free idea: <ul style="list-style-type: none">• Direct learning• Strategy is derived directly• No detour via model identification
Learning of the empirical model: <ul style="list-style-type: none">• Count values s' for all s, a• Normalize to obtain an estimate of the transition model $P_{ss'}^a$• Explore any reward R_s^a	Procedure: <ul style="list-style-type: none">• Active or passive learning

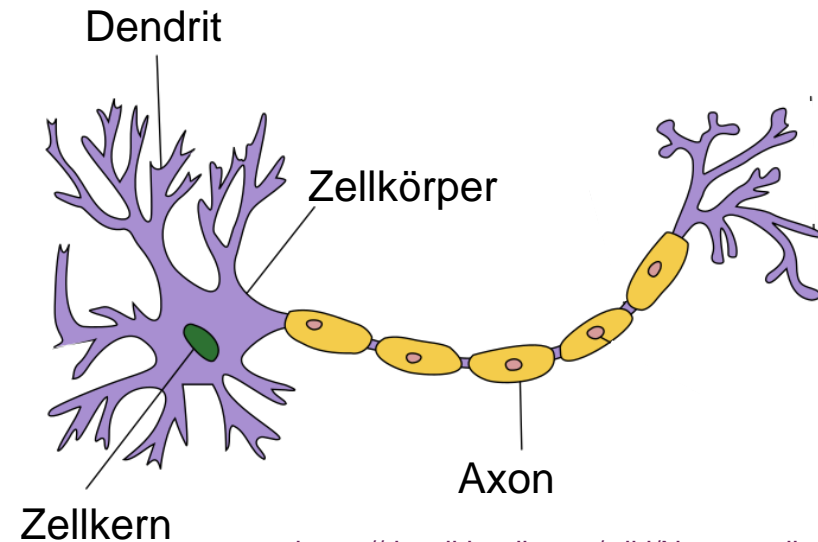
Model-free (Active) RL, Q-Learning

- Impressive results :
 - Q-Learning converges to an optimal strategy - even if the agent behaves suboptimally
- This is called **Off-Policy Learning**
- Conditional:
 - There has been enough exploration
 - The learning rate parameter may have to decrease over time.
 - ...but this must not happen too quickly.
 - In a nutshell: After enough time, it is not relevant how actions are selected for exploration (!)



Lecture 8: Neural Networks

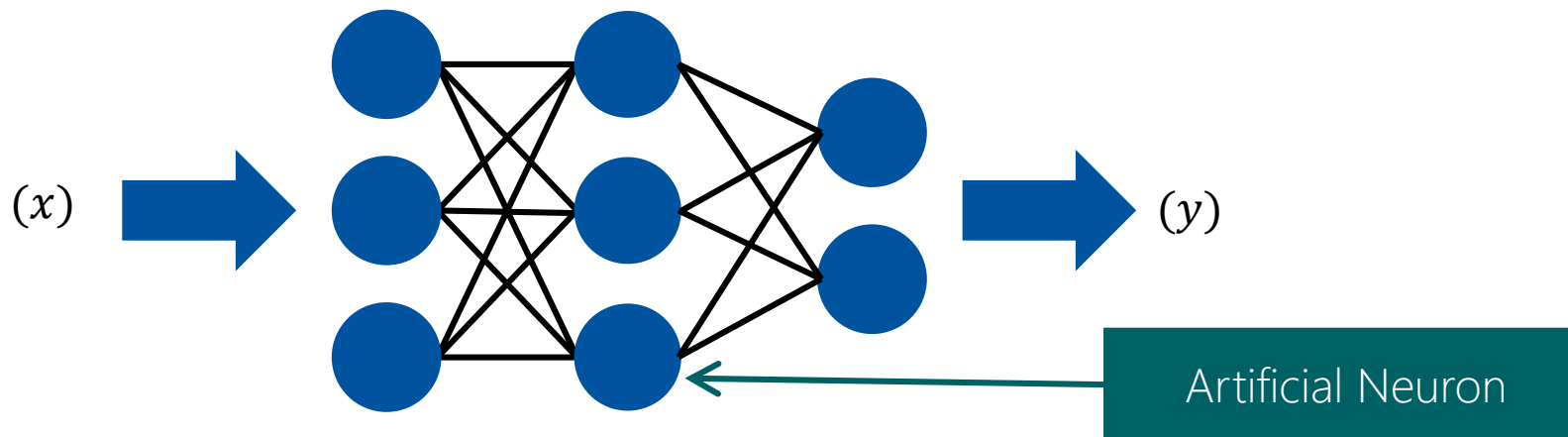
- Biology as a model
- The human brain consists of approximately 100 billion interconnected nerve cells, so-called neurons
- A neuron reacts via the dendrite to external stimuli and transmits electrical impulses to other neurons via the axon
- Neurons are specialized in recognizing individual properties in our sensory data (visual, auditory,...). Together they form our perception of the world around us.



<https://de.wikipedia.org/wiki/Nervenzelle>

Artificial Neural Networks

- Origins: Algorithms that try to imitate the functioning of the brain.
- First achievements in the 1940s and 1950s (McCulloch-Pitts-Neuron, 1943 and Perceptron, 1957). Popularity declined in the late 90s
- Recent revival: State-of-the-art for many applications (e.g. image processing, speech processing).
- Can be used for regression and classification
- A network consists of computation units: artificial **neurons** or **units**.

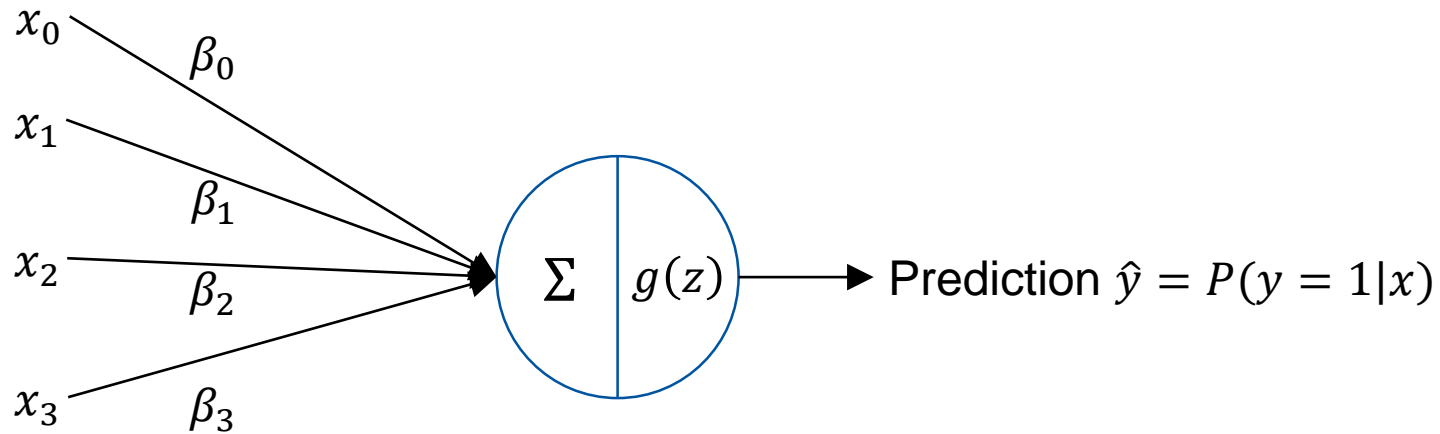




Sounds familiar?

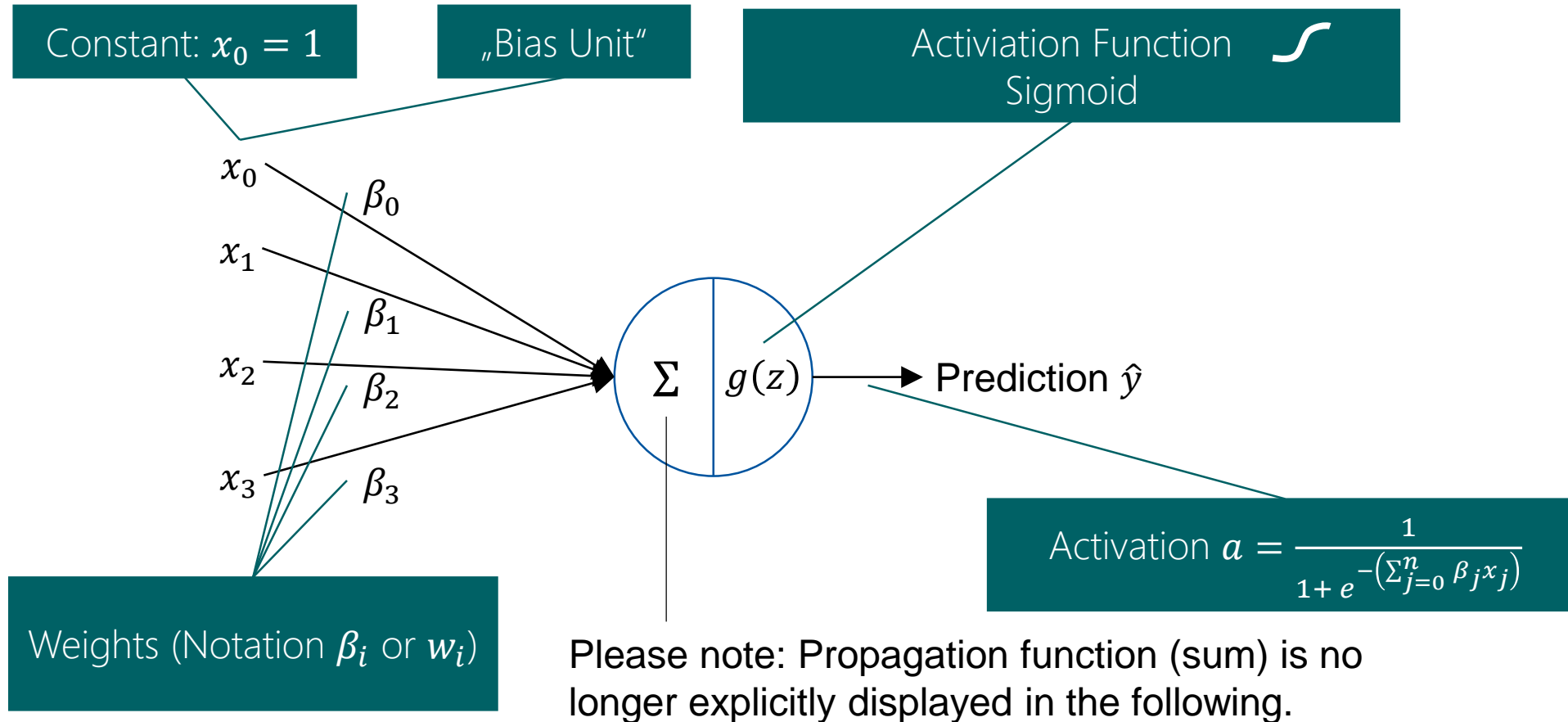
The linear model of logistic regression has a very similar structure!

- Logistic Regression: $f(x) = g(\sum_{j=0}^n \beta_j x_j)$ mit $g(z) = \frac{1}{1 + e^{-z}}$ und $x_0 = 1$
- $g(z)$: logistic function or sigmoid function
- Logistic regression model can also be represented as neuron!



Single Neurons

Logistic regression model can also be represented as neuron!

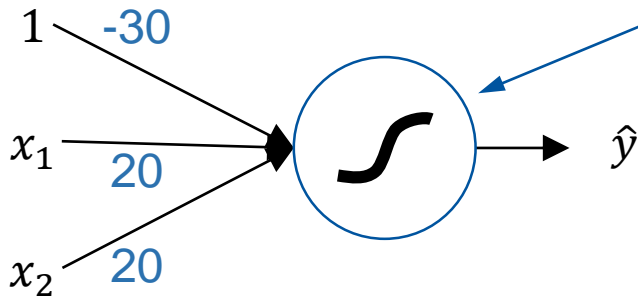


Single Neurons – The Linear Case

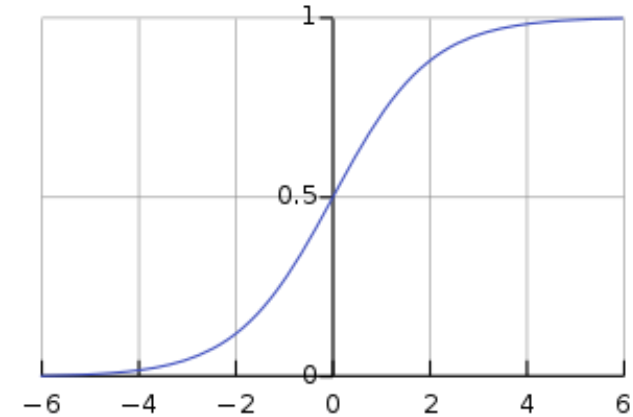
! **Linear separability**
A single neuron is suitable for classification if the data can be separated linearly!

Simple Example: the AND-Function

- $x_1, x_2 \in \{0,1\}$
- $y = x_1 \text{ AND } x_2$



$f(x) = g(-30 + 20 \times x_1 + 20 \times x_2)$ with $g(z)$ Sigmoid-Function



https://en.wikipedia.org/wiki/Sigmoid_function

x_1	x_2	$f(x)$
0	0	$g(-30) \approx 0$
0	1	$g(-10) \approx 0$
1	0	$g(-10) \approx 0$
1	1	$g(10) \approx 1$

Single Neurons – The Linear Case

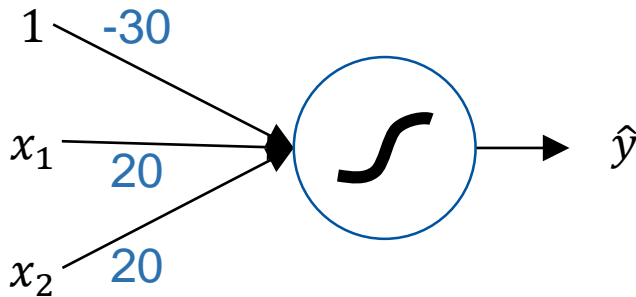


Linear separability

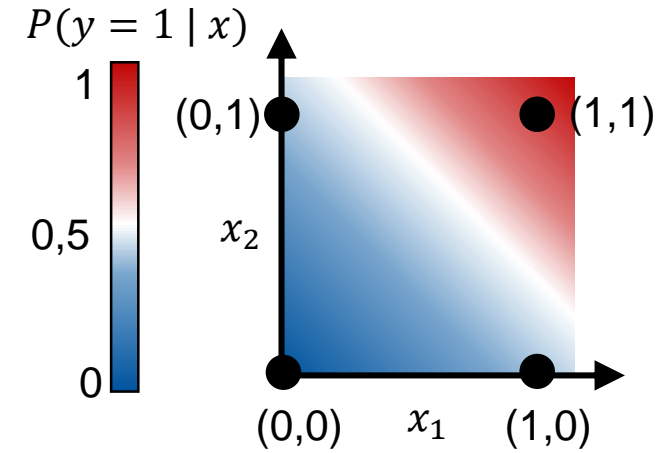
A single neuron is suitable for classification if the data can be separated linearly!

Simple Example: the AND-Function

- $x_1, x_2 \in \{0,1\}$
- $y = x_1 \text{ AND } x_2$



$f(x) = g(-30 + 20 \times x_1 + 20 \times x_2)$ with $g(z)$ Sigmoid-Function



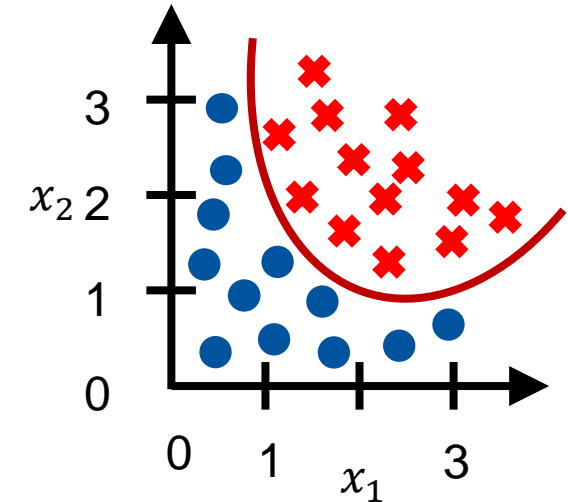
x_1	x_2	$f(x)$
0	0	$g(-30) \approx 0$
0	1	$g(-10) \approx 0$
1	0	$g(-10) \approx 0$
1	1	$g(10) \approx 1$



What to do with non-linearity?

A single neuron cannot make a nonlinear separation between two classes!

- A well-known possibility from logistic regression: transformation of input variables
- Example (linear and non-linear)
 - $f_A(x) = g(\beta_0 + \beta_1 x_1 + \beta_2 x_2)$
 - $f_B(x) = g(\beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_1^2 + \beta_4 x_2^2 + \beta_5 x_1 x_2)$



2 major disadvantages of transformation

- 1) Extreme inefficiency: Number of input variables and complexity increase enormously
- 2) Optimal degree of transformation is mostly not known before.

The Non-Linear Case – Transformation is Inefficient

Example 1: Training data with 100 input values x_1, \dots, x_{100}

- Transformation with degree 2 results in approx. 5000 new input variables
($x_1^2, x_2^2, x_1x_2, x_3^2, x_1x_3, x_2x_3, \dots$)
- Transformation with degree 3 results in approx. 170.000 new input variables

Example 2: Classification of coloured images with 50x50 pixels



$$IMG = \begin{pmatrix} x_{1,1} & x_{1,2} & \cdots & x_{1,50} \\ x_{2,1} & x_{2,2} & \cdots & x_{2,50} \\ \vdots & \vdots & \ddots & \vdots \\ x_{50,1} & x_{50,2} & \cdots & x_{50,50} \end{pmatrix} \in \mathbb{N}^{50 \times 50 \times 3}$$

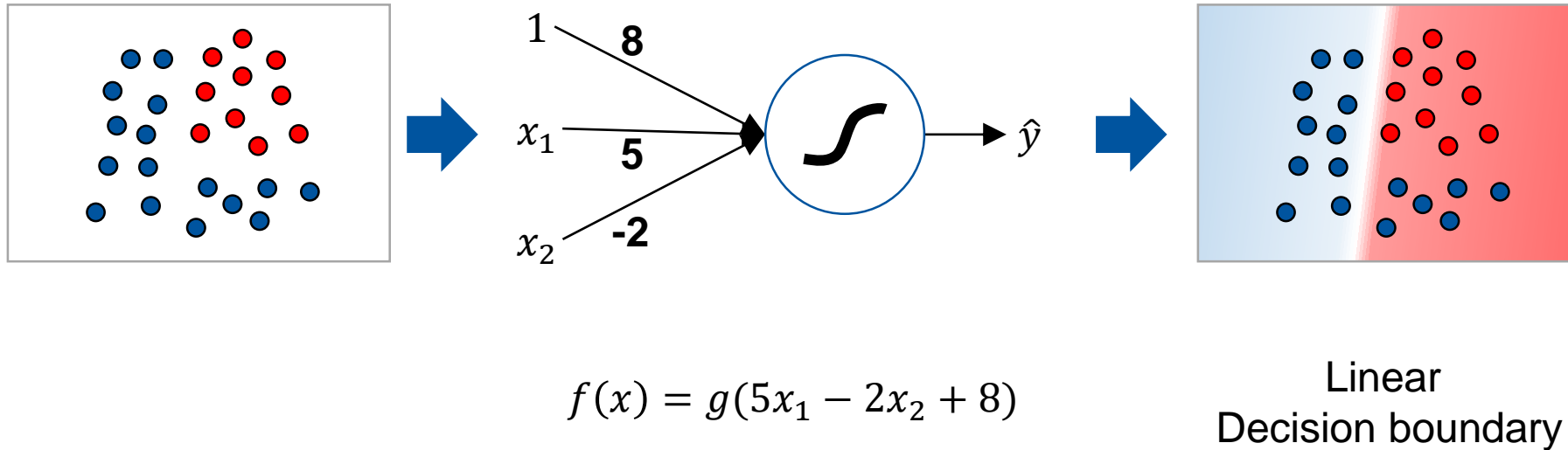
$$x_{i,j} \in \mathbb{N}^3$$

- At 50x50 pixels and 3 RGB values → 7500 Input values in the model
- Transformation with degree 2 results in approx. 3 million input variables!

The Non-Linear Case – Combination of Neurons

Data point $(x_1^{(i)}, x_2^{(i)})$ of 2 classes, which are not linear sepearable

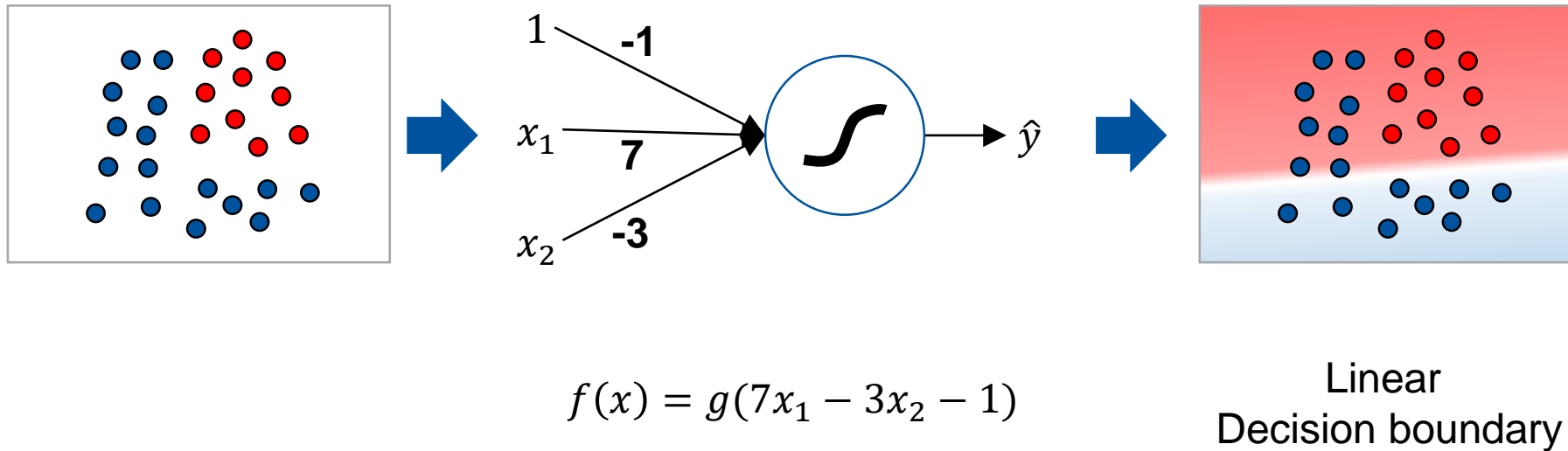
- Neuron with Sigmoid activation function:



The Non-Linear Case – Combination of Neurons

Data point $(x_1^{(i)}, x_2^{(i)})$ of 2 classes, which are not linear sepearable

- It's also possible to use the following neuron with different weights:



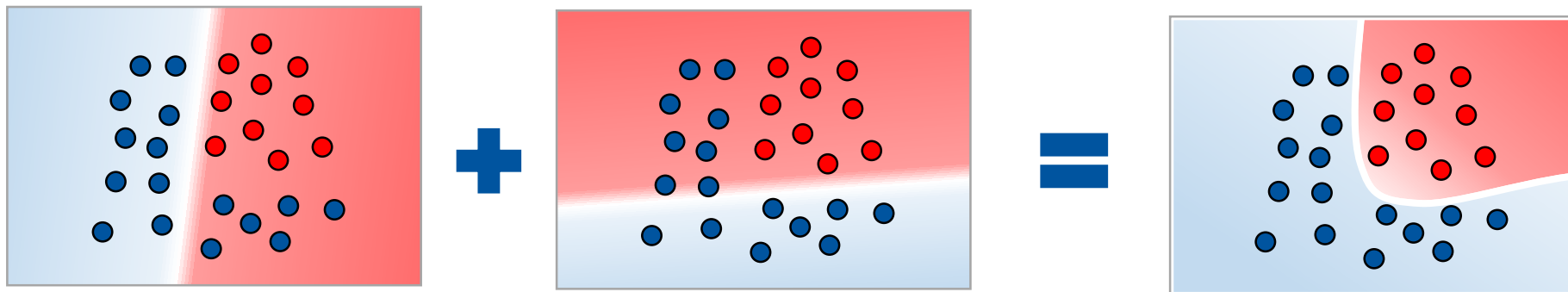
$$f(x) = g(7x_1 - 3x_2 - 1)$$



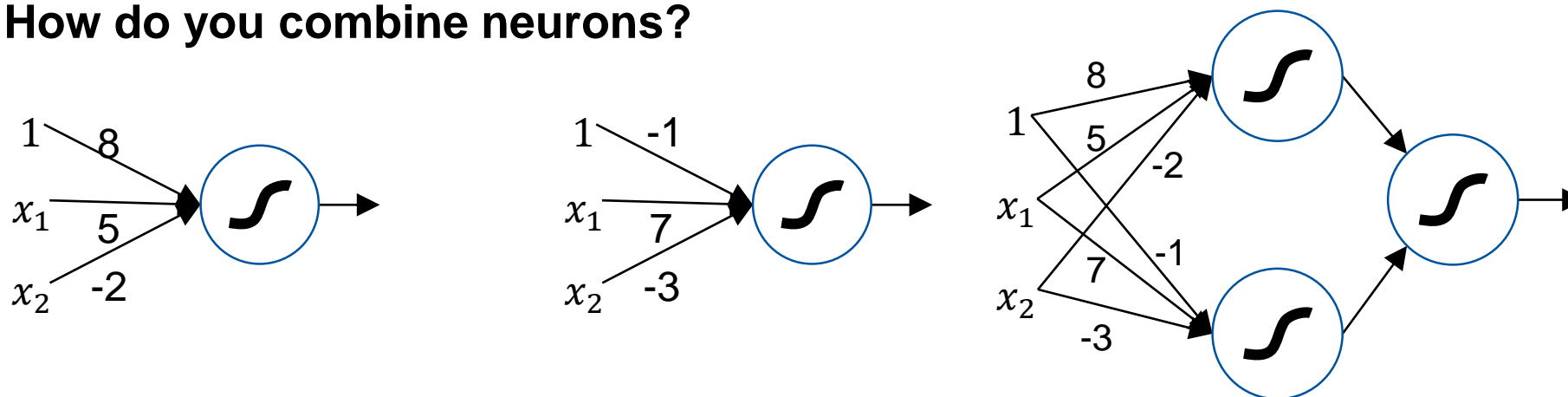
We can combine both neurons into a simple net!

The Non-Linear Case – Combination of Neurons

Illustration: Combination of regions



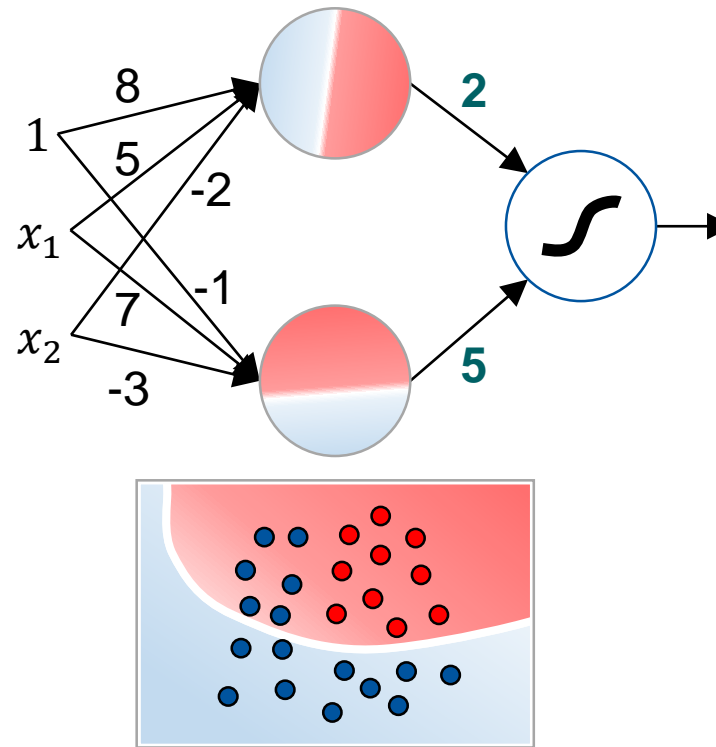
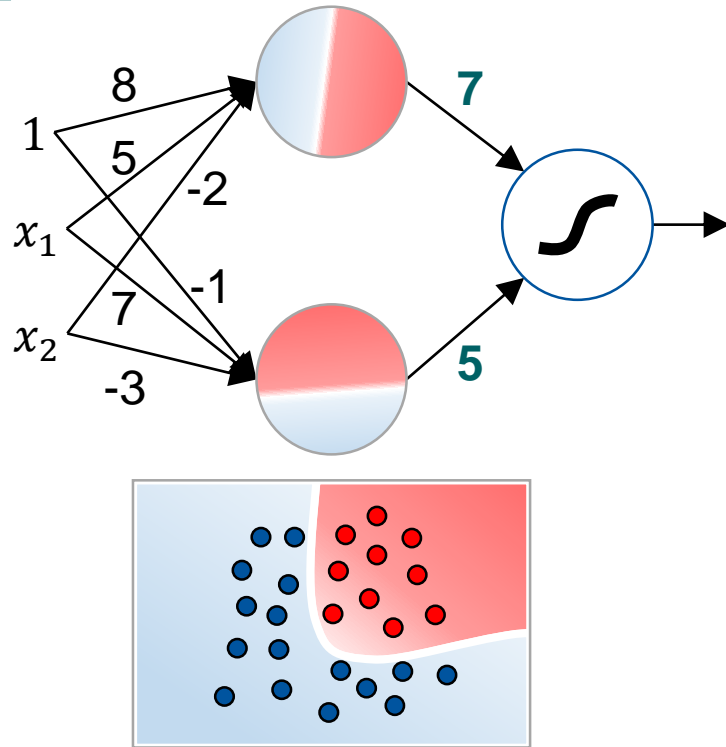
How do you combine neurons?





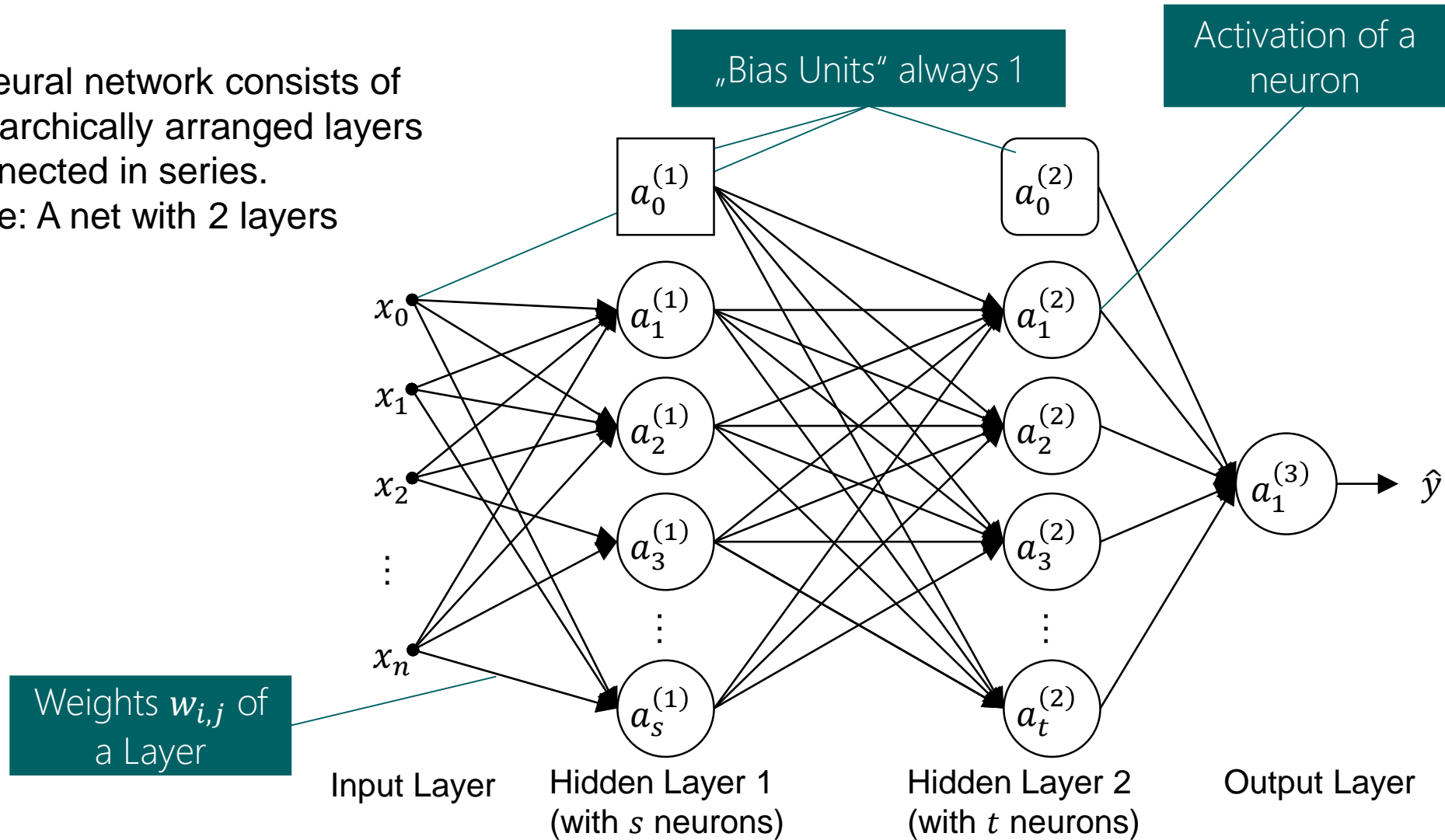
Combination of neurons by layered arrangement

- Output values (activations) of the first two neurons are input values for the third neuron.
- The inputs of the third neuron are also **weighted**!



The Artificial Neural Net

- A neural network consists of hierarchically arranged layers connected in series.
- Here: A net with 2 layers



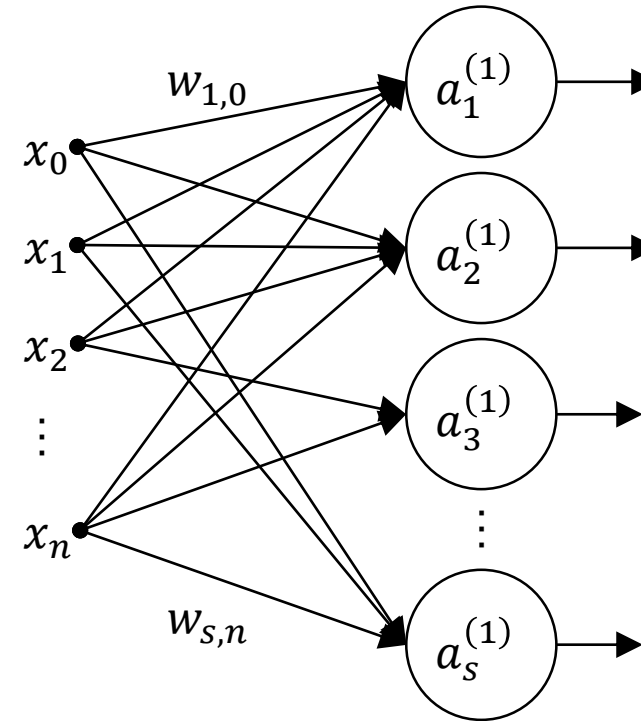
The Artificial Neural Net

- A layer in a net consists of individual neurons.
- The neurons within a layer are not linked.
- The weights of a layer can be displayed in a weight matrix.

$$\mathbf{W} \in \mathbb{R}^{s \times (n+1)} = \begin{pmatrix} w_{1,0} & w_{1,1} & \cdots & w_{1,n} \\ w_{2,0} & w_{2,1} & \cdots & w_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ w_{s,0} & w_{s,1} & \cdots & w_{s,n} \end{pmatrix}$$

- When propagating the input, the input vector $\mathbf{x} \in \mathbb{R}^n$ is multiplied from the left by the weight matrix \mathbf{W} and then transformed by the activation function.

$$\mathbf{a} \in \mathbb{R}^s = \mathbf{g}(\mathbf{W}\mathbf{x})$$



Computation of activations

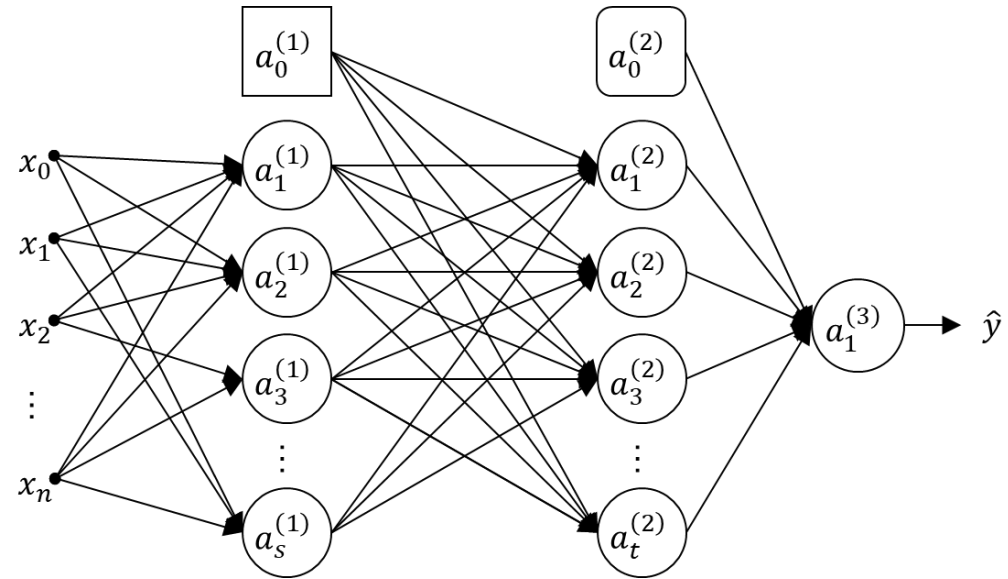
- $a_i^{(j)}$ = activation of neuron i in layer j
- $W^{(j)}$ = weight matrix of layer j

Activations of hidden layer 1

- $a_1^{(1)} = g(w_{1,0}^{(1)}x_0 + w_{1,1}^{(1)}x_1 + \dots + w_{1,n}^{(1)}x_n)$
- $a_2^{(1)} = g(w_{2,0}^{(1)}x_0 + w_{2,1}^{(1)}x_1 + \dots + w_{2,n}^{(1)}x_n)$
- ...
- $a_s^{(1)} = g(w_{s,0}^{(1)}x_0 + w_{s,1}^{(1)}x_1 + \dots + w_{s,n}^{(1)}x_n)$

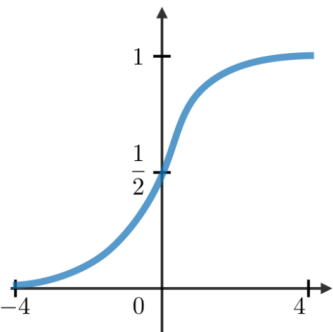
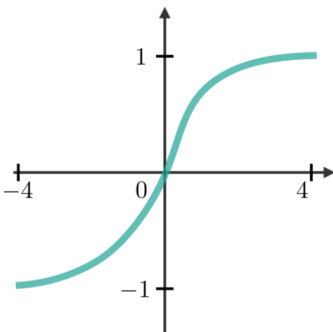
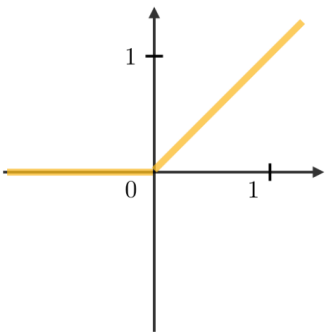
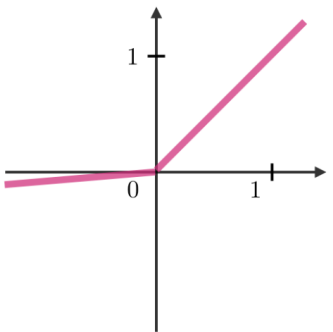
Analog: Activations of hidden layer 2 (here: example for $a_1^{(2)}$)

- $a_1^{(2)} = g(w_{1,0}^{(2)}a_0^{(1)} + w_{1,1}^{(2)}a_1^{(1)} + \dots + w_{1,s}^{(2)}a_s^{(1)})$
- If the net has s_j neurons in layer j and s_{j+1} neurons in layer $j + 1$ hat, then the weight matrix $W^{(j+1)}$ has dimensions $s_{j+1} \times (s_j + 1)$



Activation Functions

- Various activation functions are possible. Important: they must be differentiable in order to be usable for learning (gradient method)
- Tangens Hyperbolicus (Tanh) behaves similar to the logistic function (Sigmoid)
- Leaky ReLU addresses the problem of vanishing gradients.
- Simple gradient calculation without numerical approximation.
- Not constantly differentiable in $x=0$ and unlimited.

Sigmoid	Tanh	ReLU	Leaky ReLU
$g(z) = \frac{1}{1 + e^{-z}}$	$g(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$	$g(z) = \max(0, z)$	$g(z) = \max(\epsilon z, z)$ with $\epsilon \ll 1$
			

Multiclass Classification – One vs All Classification

- A simple neural net with an output neuron predicts a class (or regression value).
- For multiple classes: Add one output neuron per class.



Person



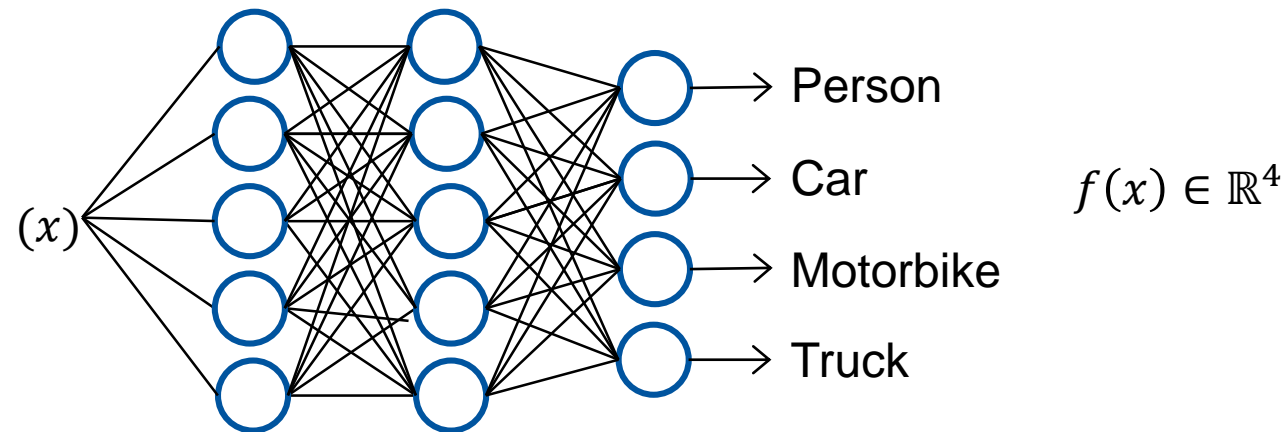
Car



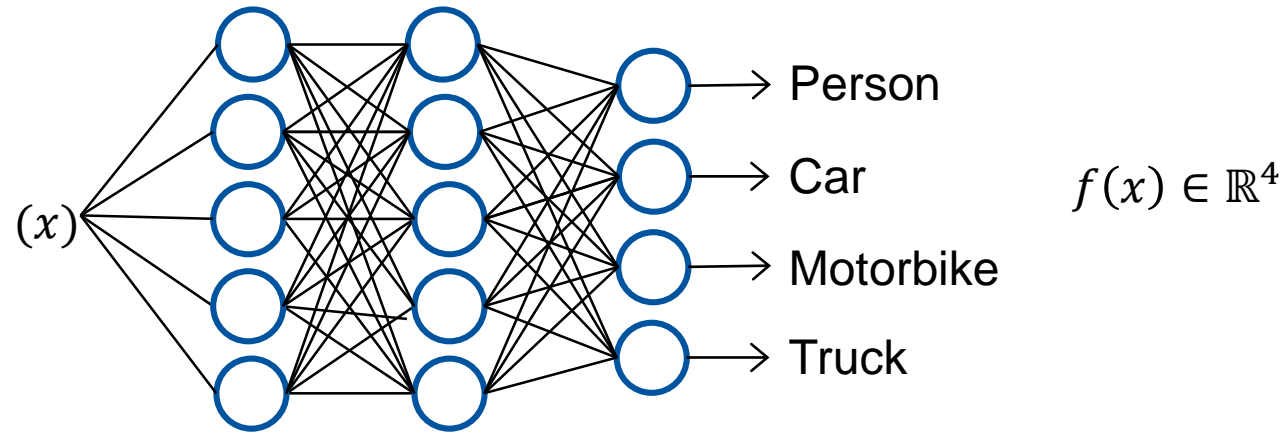
Motorbike



Truck



Multiclass Classification – One vs All Classification



Goal: $f(x) \approx \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$

If Person

$$f(x) \approx \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$$

If Car

$$f(x) \approx \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}$$

If Motorbike

$$f(x) \approx \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

If Truck

Use of one-hot encoding for the classes. For a training dataset with data points:

$$(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \dots, (x^{(m)}, y^{(m)})$$

$$y \in \mathbb{R}^4 \quad \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

Person

$$\begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$$

Car

$$\begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}$$

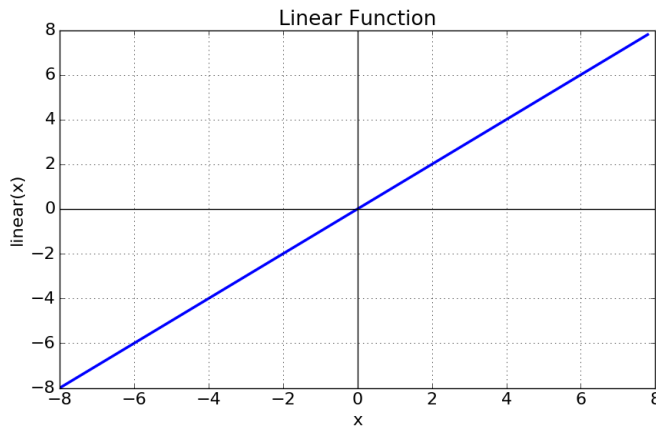
Motorbike

$$\begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

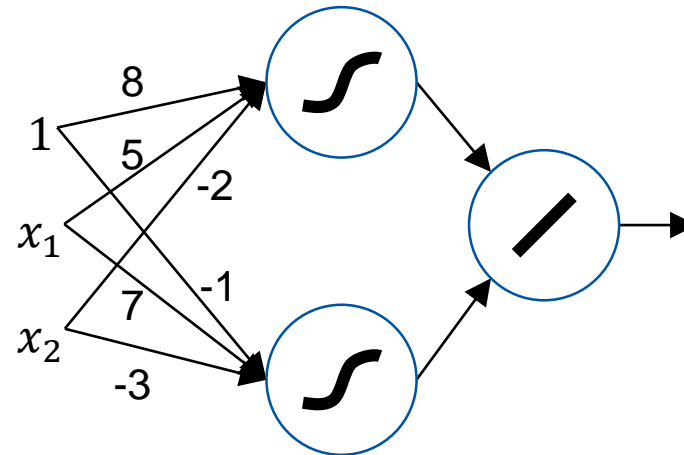
Truck

Linear activation function

- Neural networks can be used for both classification and regression.
- Difference in the Output Layer:
 - Classification: one neuron per class
 - For regression: a single neuron with a linear activation function, so the output value is not changed.



<https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6>



How do you pick the weights?

- Once you have selected the topology of a neural network (number of layers, number of neurons per layer, activation function), the model is "fitted" to the training data.
- During training, all previously unknown weights $W^{(j)}$ are adjusted for of layers j of the model, so that the model prediction \hat{y} is close to the real values.

- What is the cost function for that?

- Regression: **mean squared error**

$$L(f(x)) = \frac{1}{2m} \sum_{i=1}^m (\hat{y}^{(i)} - y^{(i)})^2 \text{ for } m \text{ data points in the training set}$$

- For binary classification: **Binary Cross-Entropy Loss**

$$L(f(x)) = -\frac{1}{m} [\sum_{i=1}^m y^{(i)} \log \hat{y}^{(i)} + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)})]$$

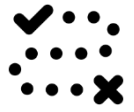
- For multiclass classification (with softmax output): **mean negativ logarithm**

Neural Network Training

Model training using gradients

- Given is an artificial neural network with newly initialized weights and a training set with i data points $(x^{(i)}, y^{(i)})$
- Goal of backpropagation: Adjusting all weights $W^{(j)}$ for all layers j in order to minimize the cost function.
- Backpropagation is based on the gradient descent method: The gradient of the error is propagated back through the net.
- In back propagation, the gradients of the cost function are calculated for each weight of the neural network and then used to adjust the weight.

Gradient Descent



Minimization of
Cost Function



Starting point
Initialization of
weights



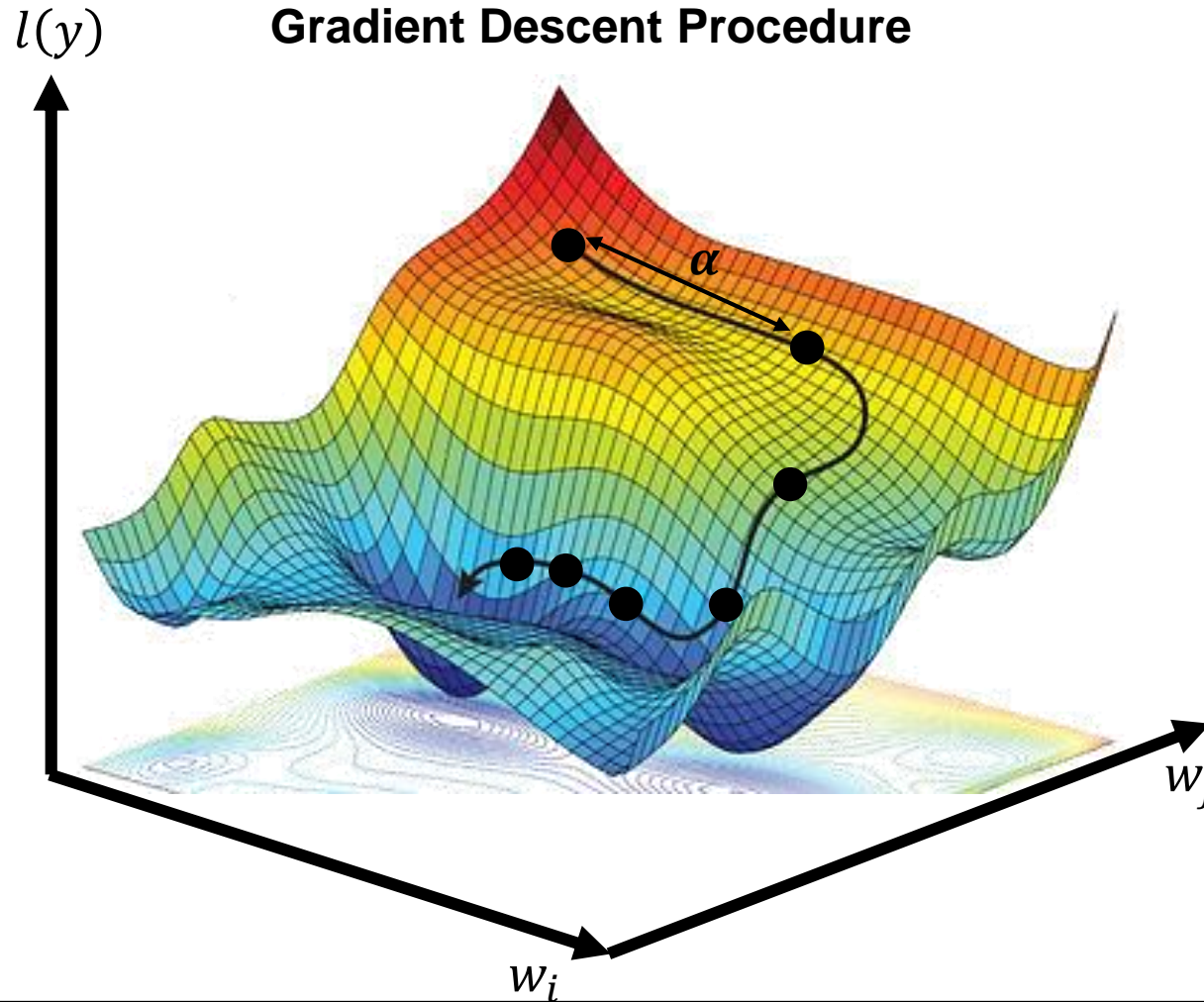
Descent along the
largest negative
gradient



Weight adjustment
by increment α



Reaching a local
minimum

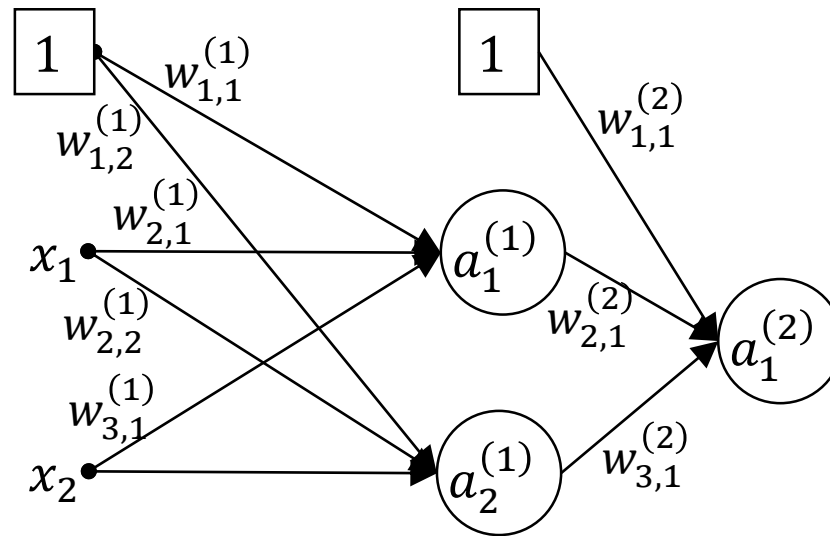


Procedure (Example for Binary Classification)

Given a training set (x, y) with data points $(x^{(i)}, y^{(i)})$ $i = 1, \dots, m$ and the following neural net with weights $W^{(j)}$, $j \in \{1, 2\}$ and the cost function

$$L(f(x)) = -\frac{1}{m} \left[\sum_{i=1}^m y^{(i)} \log \hat{y}^{(i)} + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)}) \right]$$

$$f(x) = \hat{y} = g \left(W^{(2)} \times g(W^{(1)} \times x) \right)$$



$$W^{(1)} = \begin{pmatrix} w_{1,1}^{(1)} & w_{1,2}^{(1)} \\ w_{2,1}^{(1)} & w_{2,2}^{(1)} \\ w_{3,1}^{(1)} & w_{3,2}^{(1)} \end{pmatrix}$$

$$W^{(2)} = \begin{pmatrix} w_{1,1}^{(2)} \\ w_{2,1}^{(2)} \\ w_{3,1}^{(2)} \end{pmatrix}$$

Procedure (Example for Binary Classification)

Given a training set (x, y) with data points $(x^{(i)}, y^{(i)})$ $i = 1, \dots, m$ and the following neural net with weights $W^{(j)}, j \in \{1, 2\}$ and the cost function $L(f(x)) = -\frac{1}{m} [\sum_{i=1}^m y^{(i)} \log \hat{y}^{(i)} + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)})]$

1. Compute the output values (predictions) \hat{y} of the model for x (Feedforward)
2. Compute the costs $L(f(x))$ through the cost function (Please note: $L(f(x))$ generates a scalar).
3. Using the cost function: update the weights through the **gradient of the cost function** using the specific weight. For each layer j :

$$W^{(j)} = W^{(j)} - \alpha \frac{\partial L(f(x))}{\partial W^{(j)}} \quad \text{with learning rate } \alpha$$

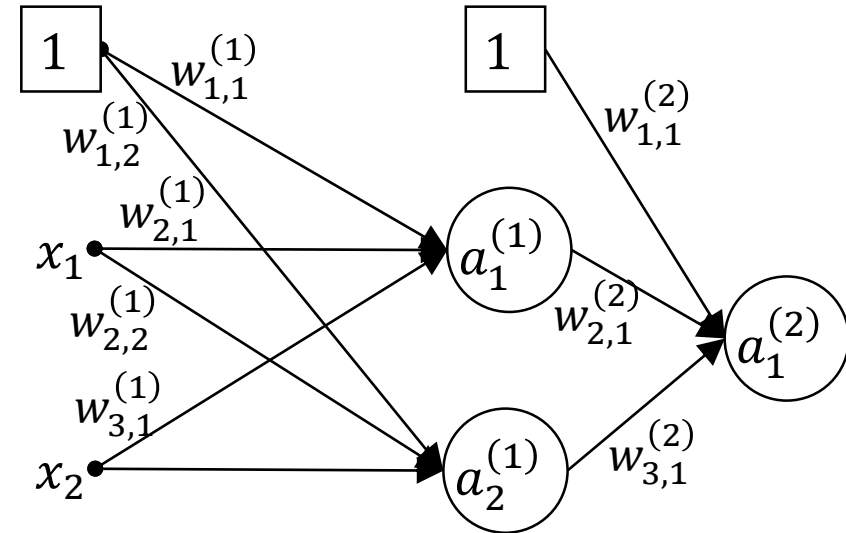
How are the gradients $\frac{\partial L(f(x))}{\partial W^{(j)}}$ calculated?

Gradient of the cost function

- Using the chain rule you can compute the gradient
- Example: Gradient w.r.t. weight $w_{1,1}^{(1)}$:

$$\frac{\partial L(f(x))}{\partial w_{1,1}^{(1)}} = \frac{\partial L(f(x))}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z^{(2)}} \frac{\partial z^{(2)}}{\partial z^{(1)}} \frac{\partial z^{(1)}}{\partial w_{1,1}^{(1)}}$$

- The partial derivative $\frac{\partial L(f(x))}{\partial w_{1,1}^{(1)}}$ is composed of the product of above derivatives.



Please note: $g(z)$ is a sigmoid-function

Overview

Given a training set i with data points $(x^{(i)}, y^{(i)})$

1. Selection of a network topology

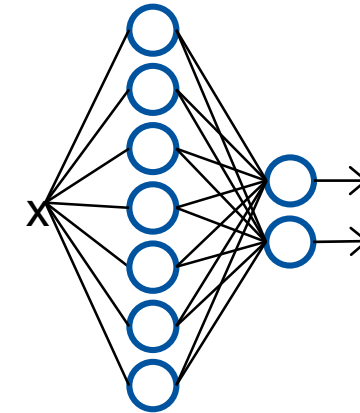
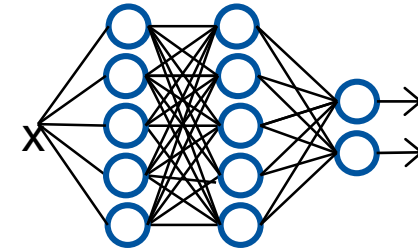
- Definition of the number of layers and neurons (recommendation: start with a hidden layer)
- Definition of the activation function for the hidden layers

2. Initialize weights $W^{(j)}$ for each layer j

- Important for learning efficiency: randomized values instead of fixed value for all weights (e.g. 0)

3. Calculate the output values (predictions) $\hat{y}^{(i)}$ of the model for $x^{(i)}$

4. Calculate gradients and optimize weights through backpropagation



Neural Network Libraries in Python

Scikit learn (of course 😊)

```
from sklearn.neural_network import MLPClassifier
X = [[0., 0.], [1., 1.]]
y = [0, 1]
clf = MLPClassifier(solver='sgd', alpha=1e-5,
                    hidden_layer_sizes=(5, 2), random_state=1)

clf.fit(X, y)
MLPClassifier(activation='relu', alpha=1e-05, batch_size='auto',
              beta_1=0.9, beta_2=0.999, early_stopping=False,
              epsilon=1e-08, hidden_layer_sizes=(5, 2),
              learning_rate='constant', learning_rate_init=0.001,
              max_iter=200, momentum=0.9, n_iter_no_change=10,
              nesterovs_momentum=True, power_t=0.5, random_state=1,
              shuffle=True, solver='sgd', tol=0.0001,
              validation_fraction=0.1, verbose=False, warm_start=False)

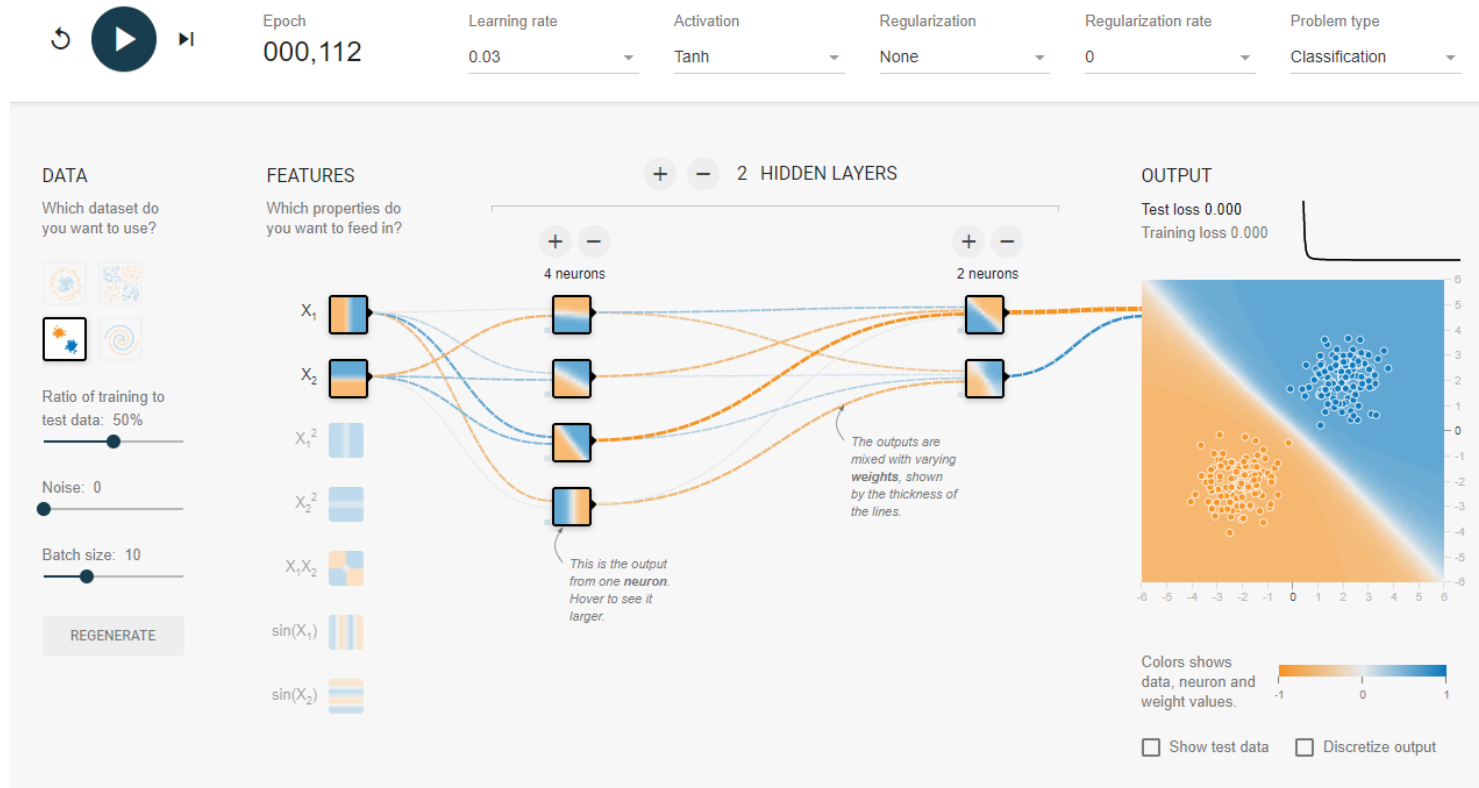
clf.predict([[2., 2.], [-1., -2.]])
```

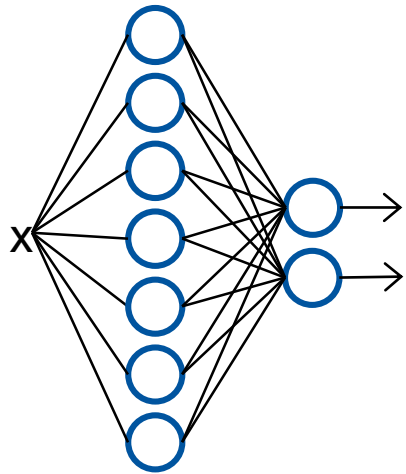



TensorFlow

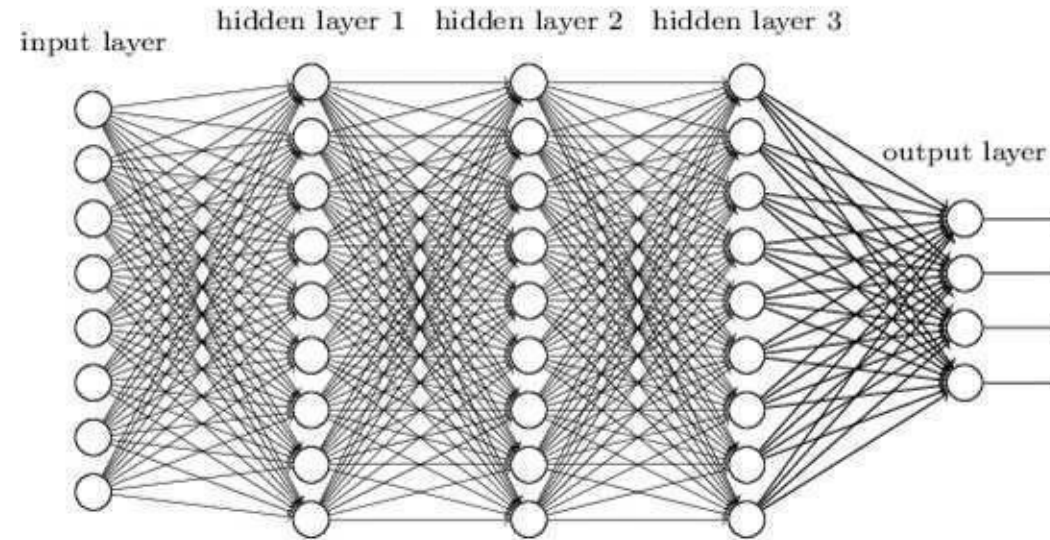
Checkout <https://playground.tensorflow.org>

Live Demo



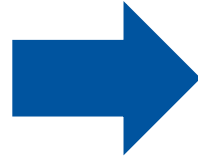
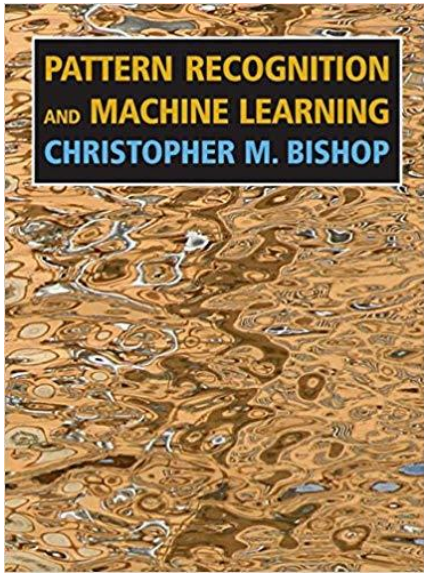


Today's topic



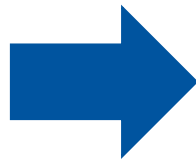
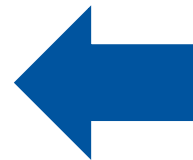
Deep Neural Networks

Further Reading Material

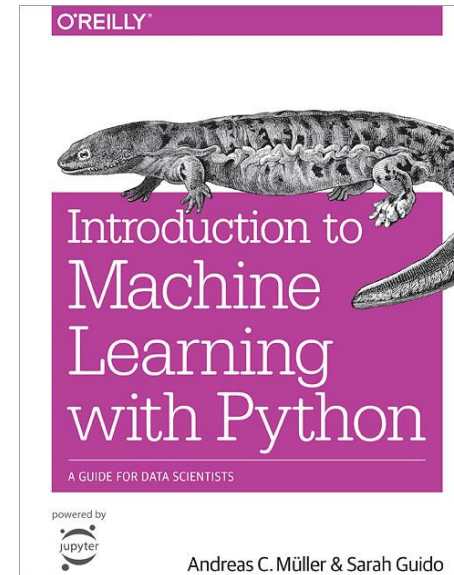


**Whole Chapter on Neural Nets
(deep dive into the theoretical foundations)**

**Whole Chapter on Neural Nets
(applied, zero mathematics)**



**Check out the YouTube
lectures of Prof. Andrew Ng**



Thank you for your attention!

Lecture Team AIDAE
aidae@ima-ifu.rwth-aachen.de