

Big Data & Data Science Neuronale Netze

WS2025/26

Prof. Dr. Klemens Waldhör

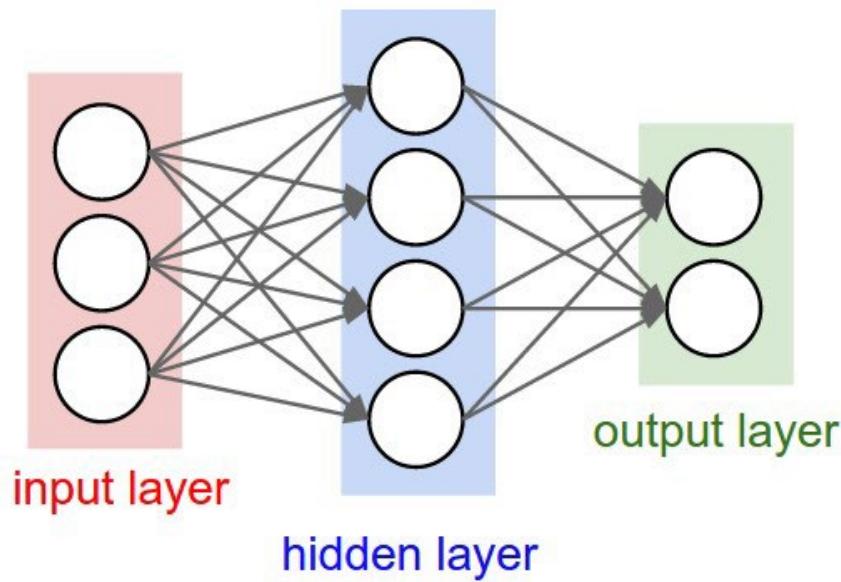
**© FOM Hochschule für Oekonomie & Management
gemeinnützige Gesellschaft mbH (FOM), Leimkugelstraße 6, 45141 Essen**

Dieses Werk ist urheberrechtlich geschützt und nur für den persönlichen Gebrauch im Rahmen der Veranstaltungen der FOM bestimmt.

Die durch die Urheberschaft begründeten Rechte (u.a. Vervielfältigung, Verbreitung, Übersetzung, Nachdruck) bleiben dem Urheber vorbehalten.

Das Werk oder Teile daraus dürfen nicht ohne schriftliche Genehmigung der FOM reproduziert oder unter Verwendung elektronischer Systeme verarbeitet, vervielfältigt oder verbreitet werden.

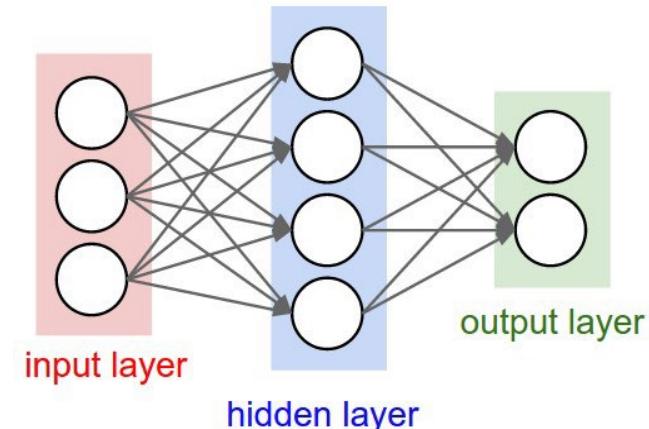
- Theorie
- CNNs Convolutional Neural Networks / Bilderkennung / Klassifizierung am Beispiel Mikrometeoriten
- Optimierung neuronaler Netzwerke
- Varianten und Beispiele für Neurale Netze
- Probleme und Herausforderungen
- Exkurs: Delta Rule und Back Propagation in NN



Artificial Neural Networks Künstliche Neuronale Netze

- Künstliche neuronale Netze (ANN) sind Berechnungsmodelle, die sich an der Struktur des menschlichen Gehirns und seinen grundlegenden Bausteinen orientieren: **Neuronen**
- Künstliche Neuronen (auch "Knoten" genannt) stellen die grundlegende Recheneinheit eines künstlichen neuronalen Netzes dar
- Jedes Neuron empfängt mehrere Eingabewerte x_i , die mit einer Reihe von Parametern w multipliziert und dann mit Hilfe einer Übertragungsfunktion (in der Regel die Summe, gefolgt von einer Verzerrung (Bias) b aggregiert werden
- Anschließend werden die aggregierten Eingaben mit einer so genannten Aktivierungsfunktion f verarbeitet, was zu einer Ausgabe y führt.

$$output = f \left(\sum (weights * input) + bias \right)$$



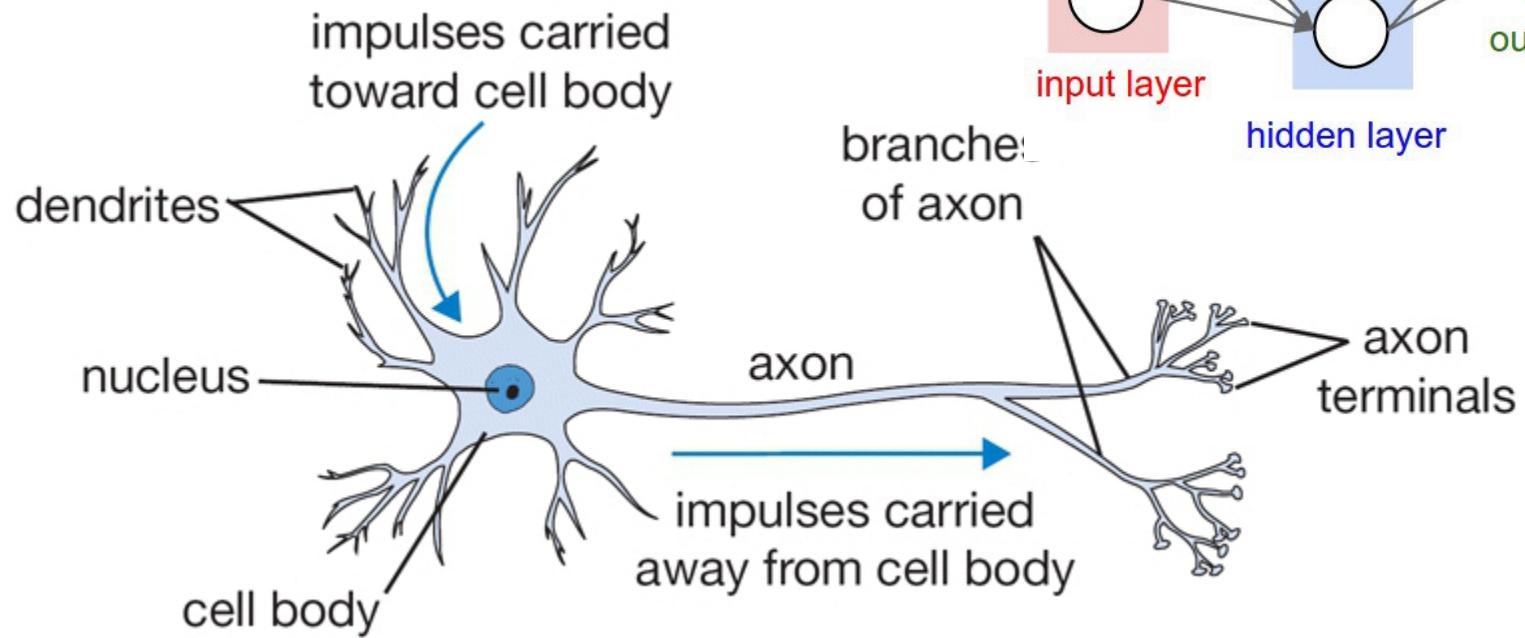
Terminologie

- Datensatz = Sample
- Attribute eines Samples = Features
- Anzahl der Features = Anzahl der Dimensionen im NN = Hypothesenraum

Visualisierung, Playgrounds

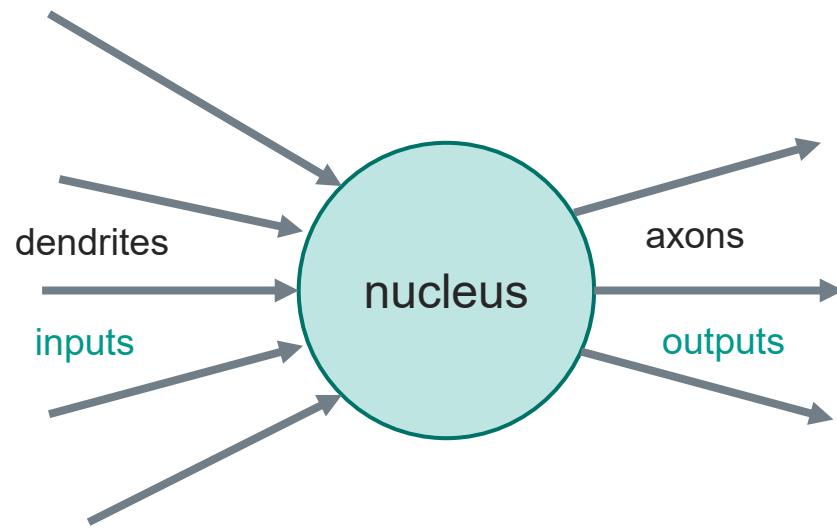
- <https://playground.tensorflow.org/>
- <https://cs.stanford.edu/people/karpathy/convnetjs/demo/mnist.html>

- Struktur eines Neurons im menschlichen Gehirn



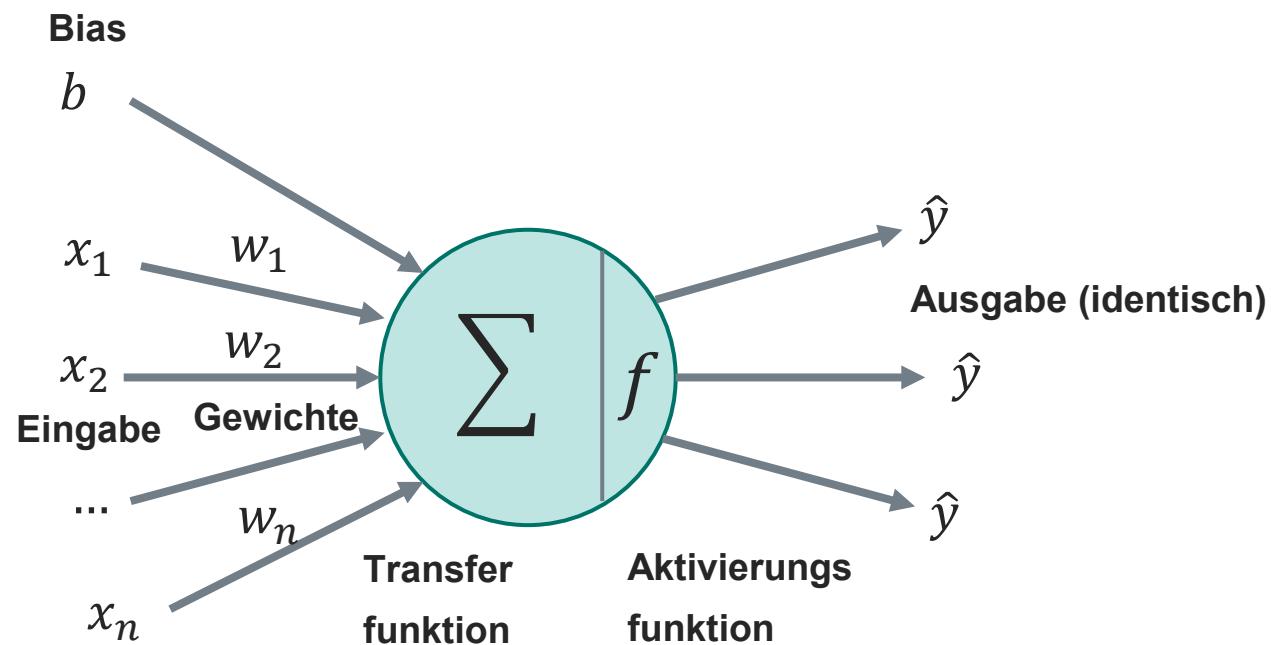
<https://towardsdatascience.com/a-gentle-introduction-to-neural-networks-series-part-1-2b90b87795bc>

- Die Idee: Übertragen der Struktur eines echten Neurons in ein mathematisches Modell



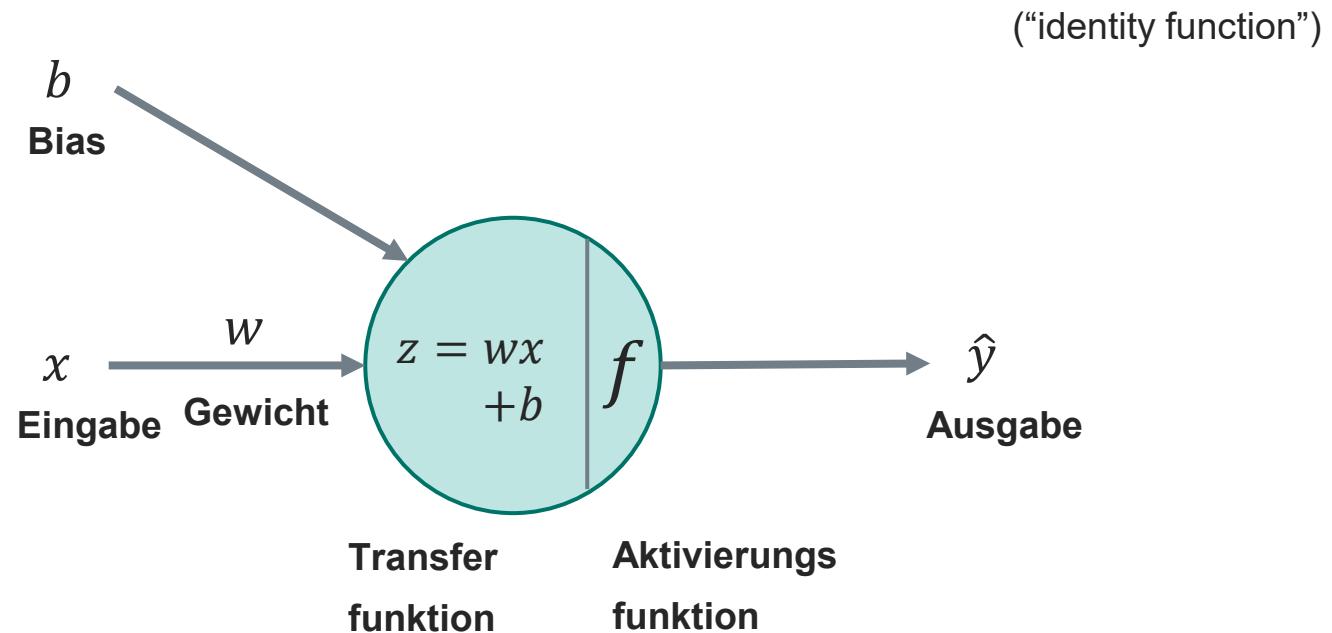
Artificial Neuron (Simple Perceptron)

Transferfunktion:
$$z = \sum_i w_i x_i + b$$
 Aktivierungsfunktion: $\hat{y} = f(z)$



- Beispiel: Lineare Regression dargestellt als künstliches Neuron

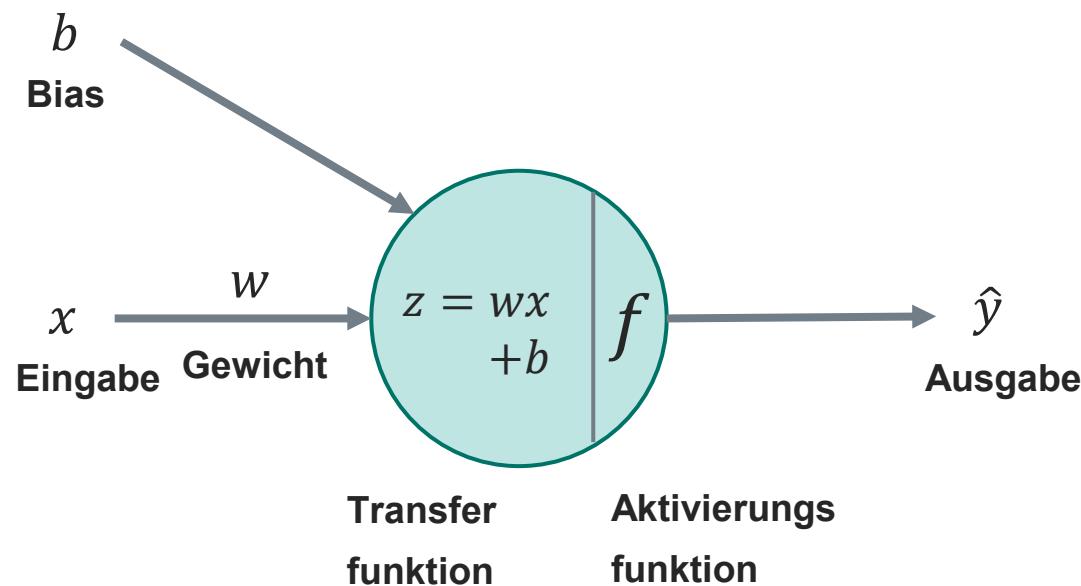
Transferfunktion: $z = wx + b$ **Aktivierungsfunktion:** $\hat{y} = f(z) = z$



- Beispiel: Logistische Regression dargestellt als künstliches Neuron

$$\text{Transferfunktion : } z = wx + b \quad \text{Aktivierungsfunktion : } \hat{y} = f(z) = \frac{1}{1+e^{-z}}$$

(Sigmoid-Funktion)

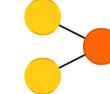


Neural Networks

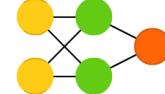
©2016 Fjodor van Veen - asimovinstitute.org

- Backfed Input Cell
- Input Cell
- △ Noisy Input Cell
- Hidden Cell
- Probabilistic Hidden Cell
- △ Spiking Hidden Cell
- Output Cell
- Match Input Output Cell
- Recurrent Cell
- Memory Cell
- △ Different Memory Cell
- Kernel
- Convolution or Pool

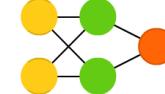
Perceptron (P)



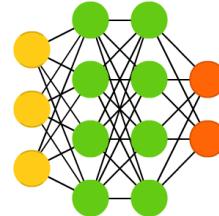
Feed Forward (FF)



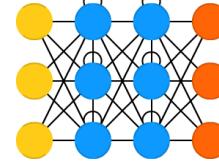
Radial Basis Network (RBF)



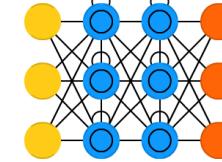
Deep Feed Forward (DFF)



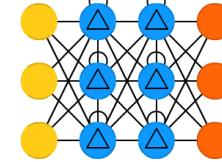
Recurrent Neural Network (RNN)



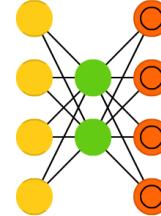
Long / Short Term Memory (LSTM)



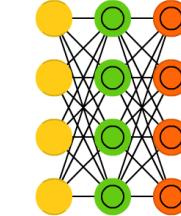
Gated Recurrent Unit (GRU)



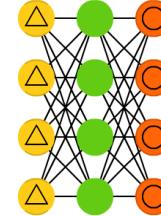
Auto Encoder (AE)



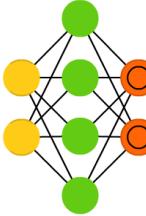
Variational AE (VAE)



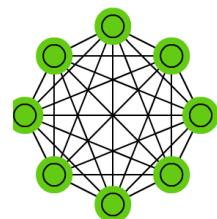
Denoising AE (DAE)



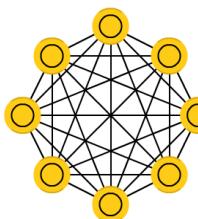
Sparse AE (SAE)



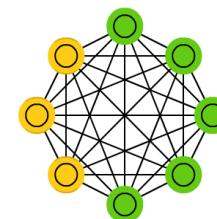
Markov Chain (MC)



Hopfield Network (HN)



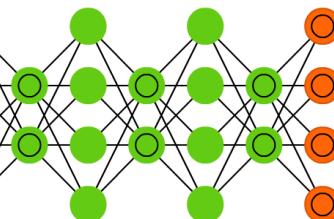
Boltzmann Machine (BM)

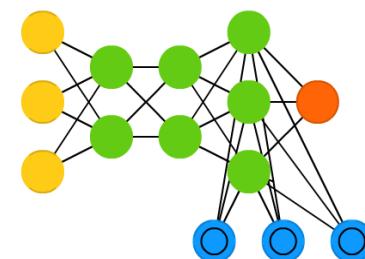
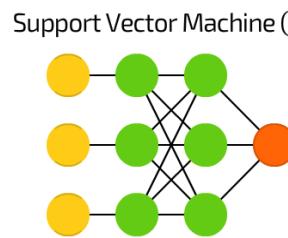
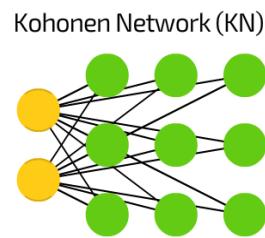
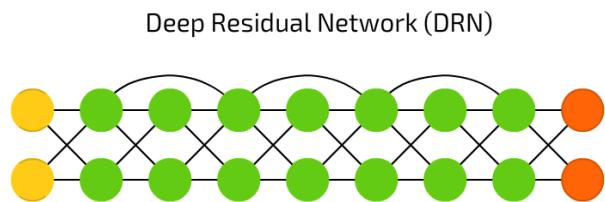
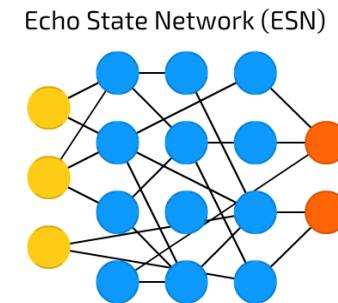
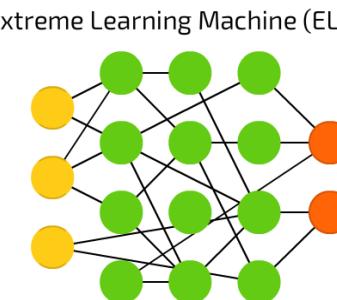
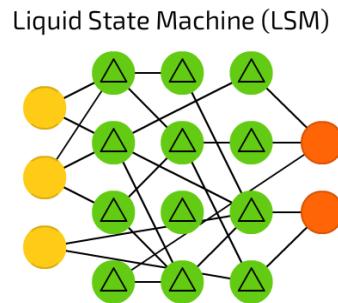
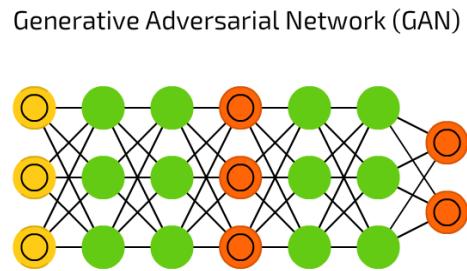
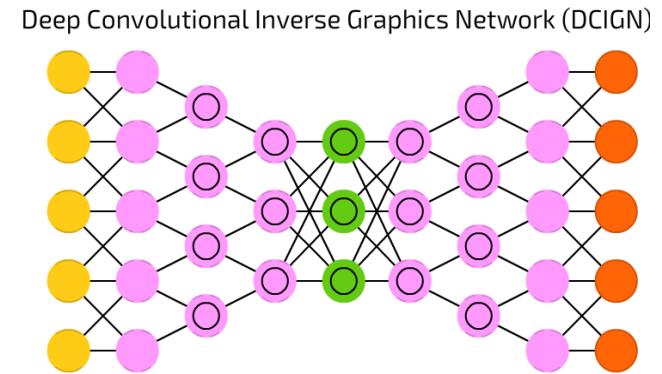
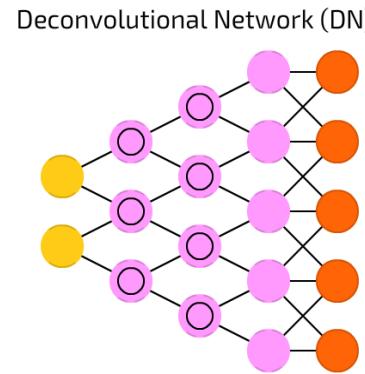
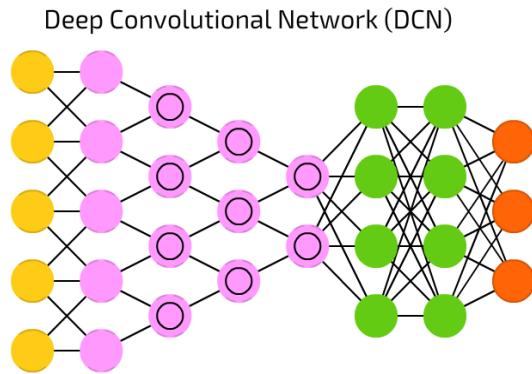


Restricted BM (RBM)

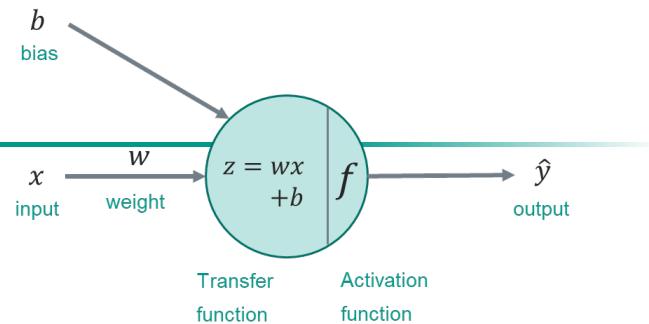


Deep Belief Network (DBN)





Aktivierungsfunktionen (activation function)



Lineare Funktion

- Nicht nützlich für neuronale Netze, da die meisten Probleme nichtlinear sind (ein Menge linearer Funktionen ist auch eine lineare Funktion)

$$f(x) = mx$$



Sigmoid

- Nicht-linear
- Neigt zu verschwindenden Gradienten (Lernen hört auf, wenn die Werte von x sehr klein oder sehr groß werden)

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



tanh

- Nicht-linear
- Ähnlich dem Sigmoid
- Weniger anfällig für verschwindende Gradienten (vanishing gradients)

$$f(x) = \tanh(x) = \frac{2}{1 + e^{-2x}} - 1$$



Rectified Linear Unit (ReLU)

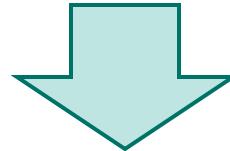
- Nicht-lineare
- Meistgenutzte Aktivierungsfunktion
- Vereinfacht den Gradientenabstieg in tiefen Netzen (Deep Networks)

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



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

- Das menschliche Gehirn besteht aus $\sim 10^{11}$ Neuronen
- Jedes Neuron hat etwa 10^4 bis 10^5 Verbindungen
- Berechnungen in Neuronen dauern $\sim 0,001$ Sekunden
- Kognitive Aufgaben (z. B. Gesichtserkennung) dauern $\sim 0,1$ Sekunden



- Daher werden während einer kognitiven Aufgabe nur ~ 100 Berechnungen in einem einzigen Neuron durchgeführt.
- Daraus folgt: Berechnungen müssen über viele Neuronen hinweg parallelisiert werden
- Künstliche neuronale Netze: Parallelere Berechnungen werden durch die Gruppierung mehrerer Neuronen in sogenannten Schichten erreicht

Schichten eines einfachen künstlichen neuronalen Netzes (Multilayer Perceptron):

Eingabeschicht

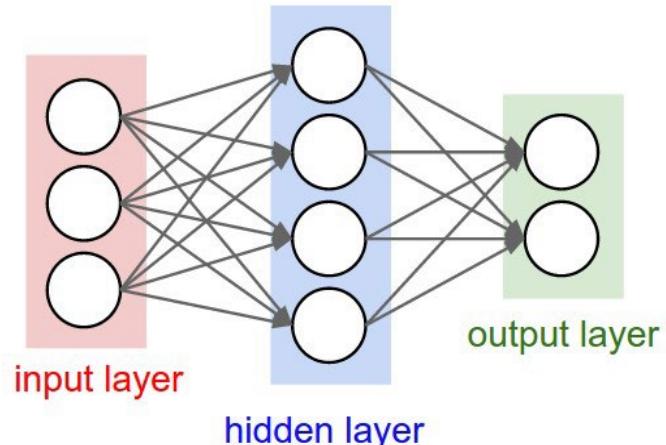
- Liefert die Eingabedaten, führt keine Berechnungen durch

Verborgene Schicht (Hidden Layer)

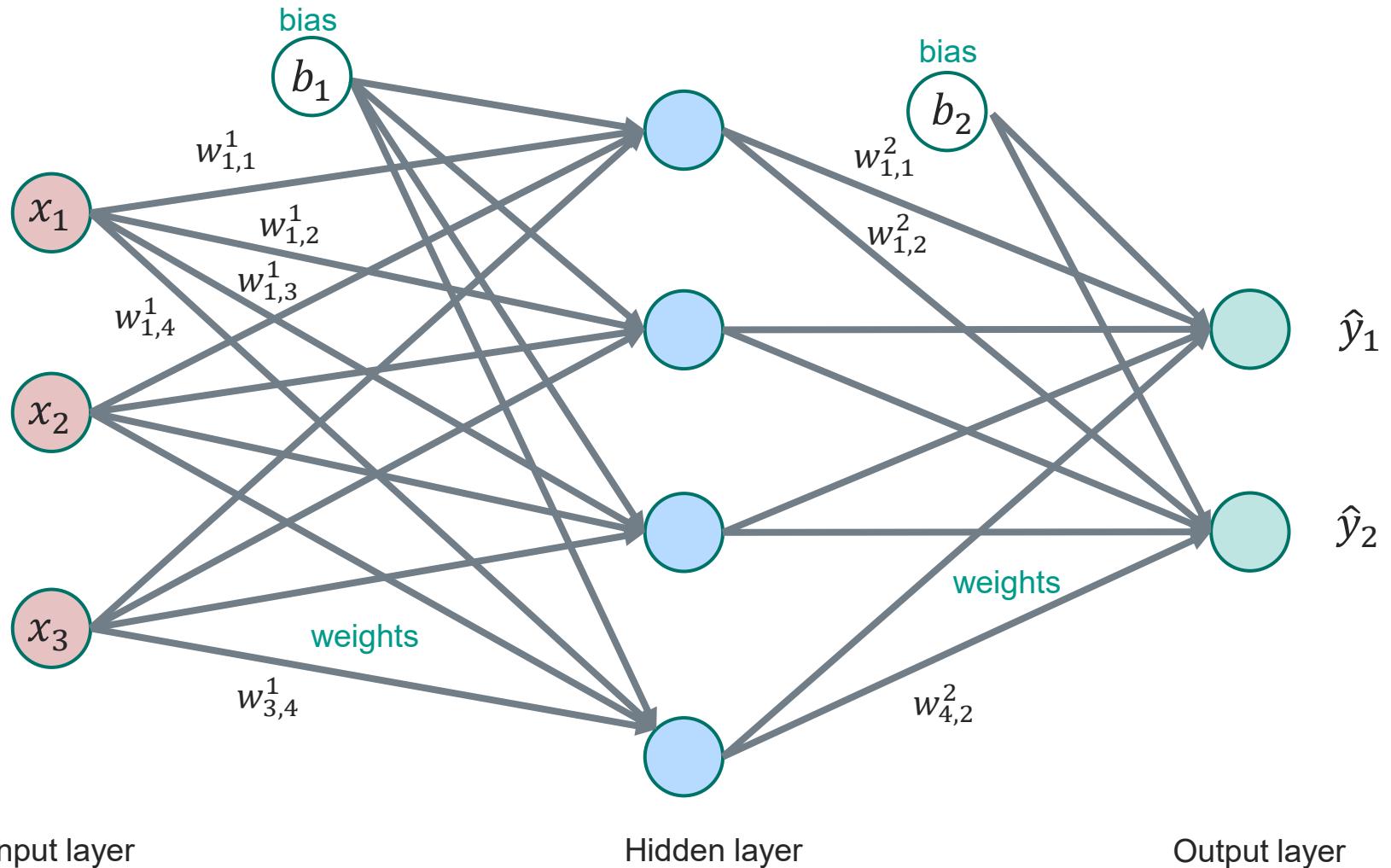
- Transformiert die Eingabedaten durch Anwendung der Übertragungs- und Aktivierungsfunktionen (unter Verwendung der Gewichte)
- Übergabe der Ergebnisse an die nächste Schicht (versteckte Schicht oder Ausgabeschicht)
- Die Komplexität eines neuronalen Netzmodells nimmt mit der Anzahl der verborgenen Schichten zu

Ausgabeschicht

- Umwandlung der Ergebnisse aus der vorherigen versteckten Schicht in die gewünschte Ausgabe (z. B. Klassenbezeichnungen)



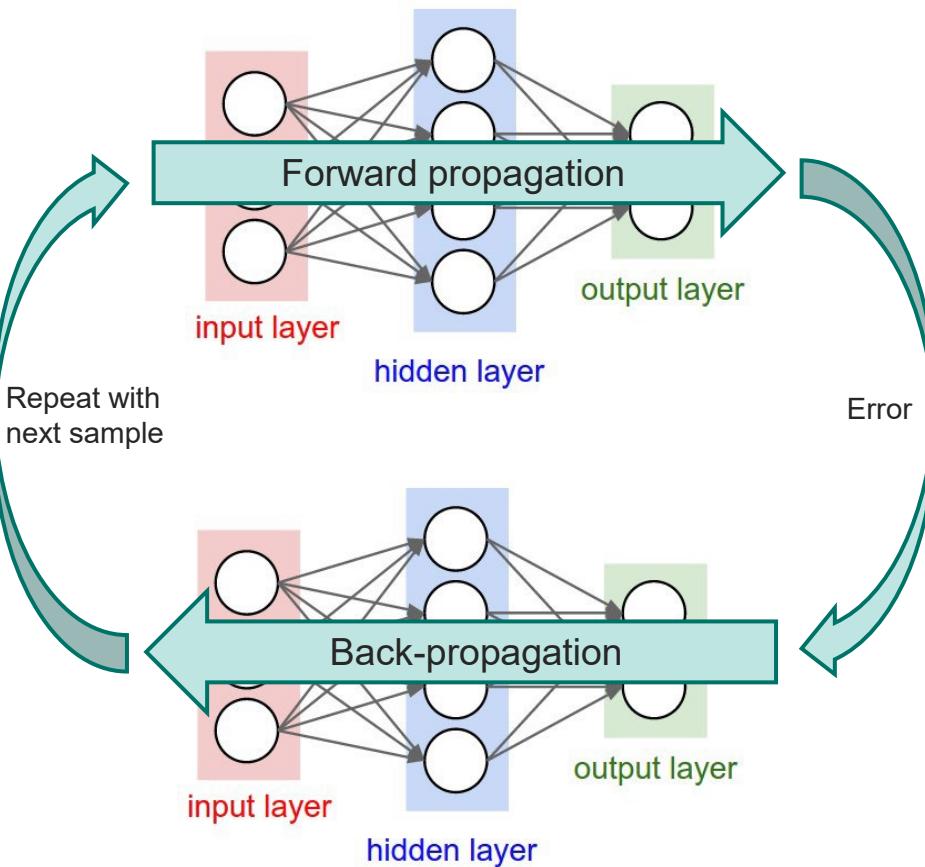
Multilayer Perceptron (MLP)



Multilayer Perceptron (MLP)

- Mehrschichtiges Perzepron (MLP)
- Jedes Neuron einer Schicht ist mit allen Neuronen der vorhergehenden Schicht "vollständig verbunden".
- Jede Verbindung (grauer Pfeil) hat ein Gewicht $w_{i,j}^l$, jede Schicht hat einen Biasterm b_l
- Die Eingangswerte eines Neurons werden mit den Gewichten $w_{i,j}^l$ der entsprechenden Verbindungen multipliziert, aufsummiert, zum Bias b_l , addiert und dann mit einer Aktivierungsfunktion transformiert.
- Auf diese Weise werden die Eingabewerte durch das Netz weitergeleitet, wobei sie von den verschiedenen Neuronenschichten transformiert werden, bis die letzte Ausgangsschicht eine Vorhersage liefert (Vorwärtspropagation, forward propagation)
- Die Vorhersagen des Netzes können anhand einer Verlustfunktion (z. B. Cross-Entropy-Loss) bewertet werden.
- Die Gewichte $w_{i,j}^l$ und Verzerrungen b_l des Netzes werden mit Hilfe des Gradientenabstiegs in Bezug auf die Fehlerfunktion aktualisiert (Rückwärtspropagation, backward propagation)

Multilayer Perceptron (MLP)

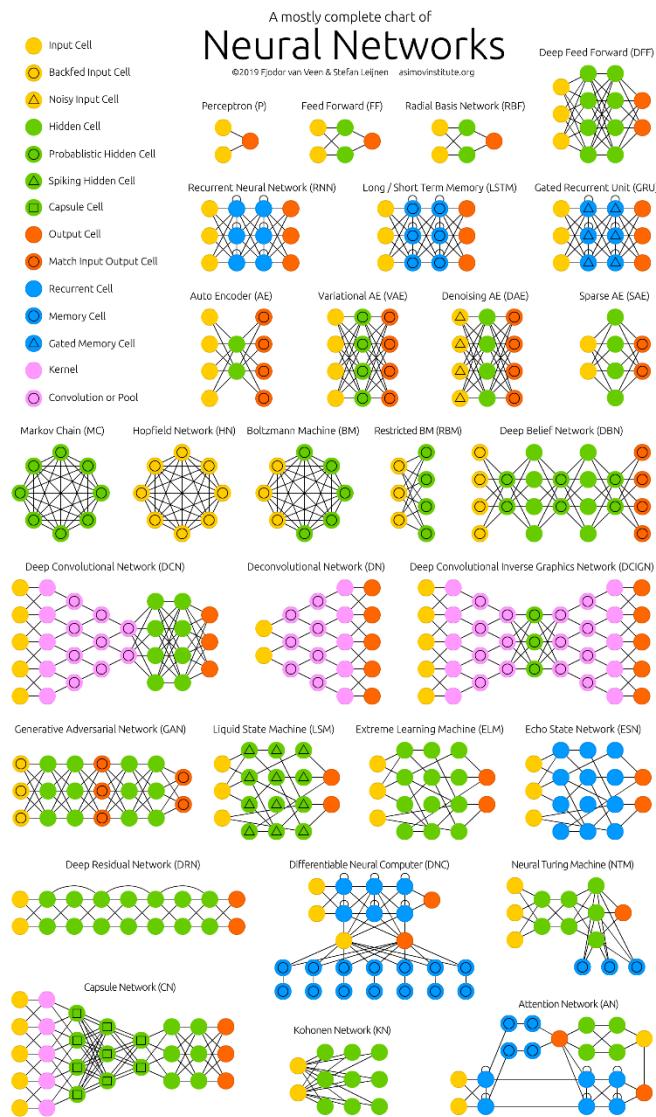


Vorwärtspropagation

1. Die Eingabedaten werden sequentiell durch die Schichten der Neuronen geleitet.
2. Verwenden Sie die Parameter (Gewichte/Biases), um die Aktivierungen in jedem Neuron zu berechnen.
3. Die Ergebnisse der Ausgabeschicht stellen die Vorhersagen dar.
4. Vergleichen Sie die Vorhersagen mit den tatsächlichen Bezeichnungen und berechnen Sie den Fehler.

Rückwärtspropagation

1. Verwenden Sie den Fehler, um die Gradienten in Bezug auf jeden Parameter im Netz zu berechnen (rückwärts vom Ausgang zum Eingang)
2. Verwenden Sie die Gradienten, um die Gewichte zu aktualisieren (ähnlich wie beim Gradientenabstieg für lineare Regression)
3. Wiederholen Sie den Vorwärts-Rückwärts-Zyklus für die nächste Probe (bis zur Konvergenz).



Delta Rule und Back Propagation in NN

Gewichtsänderung nach der Delta-Regel (einfaches Perzeptron)

w' = neues Gewicht

w = altes Gewicht

λ (lambda) = Lernrate

t_x = Trainingslabel (Soll- bzw. Lehrwert)

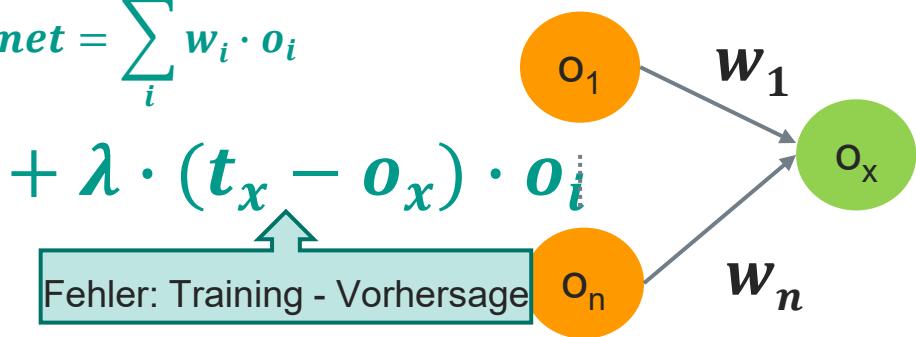
o_x = Ausgabewert (berechneter Wert)

o_i = Eingabewert

$$net = \sum_i^n w_i \cdot o_i$$

$$w_i' = w_i + \lambda \cdot (t_x - o_x) \cdot o_i$$

Fehler: Training - Vorhersage



Die **Delta-Regel** passt die Gewichte so an, dass die Differenz zwischen dem **Soll-Wert** (t_x) und dem **tatsächlichen Ausgabewert** (o_x) kleiner wird. Das Gewicht wird in Richtung des Fehlers korrigiert, skaliert durch die Lernrate und den Eingabewert.

LISTING 6.1 Algorithmus der Delta-Regel

```

PROCEDURE Delta-Regel
REPEAT
    FOR m:=1 TO Anzahl Muster DO
        Musterm an Eingabeschicht anlegen;
        Bestimme Ausgabewerte der Ausgabe-Neuronen;
        FOR j:=1 TO Anzahl Ausgabe-Neuronen DO
            (*) IF teachoutputj ≠ outputj THEN
                FOR i:=1 TO Anzahl Eingabe-Neuronen DO
                    (***)  $w_{ij} = w_{ij} + \text{Lernfaktor} * \text{output}_i * (\text{teachoutput}_j - \text{output}_j);$ 
                END IF
            END FOR
        END FOR
        Test aller Muster;
    UNTIL gewünschtes Verhalten erreicht;
END Delta-Regel

```

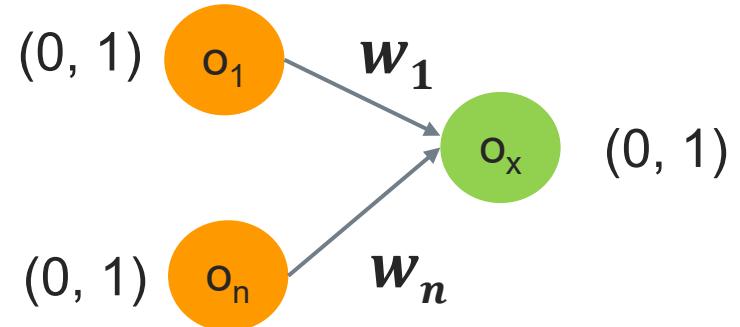
Lämmel, Uwe; Cleve, Jürgen
(2012): Künstliche Intelligenz. Mit
51 Tabellen, 43 Beispielen, 118
Aufgaben, 89 Kontrollfragen und
Referatsthemen. 4. Aufl.
München: Hanser.

- Aussagenlogik und Verknüpfungen

A	$\neg A$	A	B	A	B	$A \rightarrow B$	$A \leftrightarrow B$
T	F	T	T	T	T	T	T
F	T	T	F	F	T	F	F
		F	T	F	T	T	F
		F	F	F	F	T	T

A	B	A	B	$A \mid B$	A	B
T	T	F	F	F	F	F
T	F	T	T	T	F	F
F	T	T	T	T	F	F
F	F	F	T	T	T	T

- Als neuronales Netz (Perceptron)



$$w'_i = wi + \lambda \cdot (t_x - o_x) \cdot o_i$$

Anpassung der Gewichte nach der Regel: $w' = w + \lambda \cdot (t_x - o_x) \cdot o_i$

Lernrate Lambda	Delta Regel: Logisches And, Activation ox: Aktivierungsfunktion = 0 wenn Net < 0.5, sonst 1							
	oi	Input-neuronen-Schicht	w	Output-neuron	tx training label	Net = wi*oi	ox	Delta tx-ox
Startphase - Gewicht 0	1	1	0,000	1	1	0,000	0	1,000
	1	2	0,000					
Gewichte nach 1. Trainingphase	1	1	0,200	1	1	0,400	0	1,000
	1	2	0,200					
Gewichte nach 2. Trainingsatz	1	1	0,400	1	1	0,800	1	0,000
	1	2	0,400					
Test mit (0,0), (0,1), (1,0) und Startgewichten	0	1	0,000	1	0	0,000	0	0,000
	0	2	0,000					
	0	1	0,000	1	0	0,000	0	0,000
	1	2	0,000					
	1	1	0,000	1	0	0,000	0	0,000
Test mit (0,0), (0,1), (1,0) und Endgewichten	0	1	0,400	1	0	0,000	0	0,000
	0	2	0,400					
	0	1	0,400	1	0	0,400	0	-0,400
	1	2	0,400					
	1	1	0,400	1	0	0,400	0	-0,400
	0	2	0,400					

w' = neues Gewicht, w = altes Gewicht, λ (lambda) = Lernrate, t_x = Trainingslabel (Soll- bzw. Lehrwert)

o_x = Ausgabewert (berechneter Wert), o_i = Eingabewert

$$w'_i = wi + \lambda \cdot (t_x - o_x) \cdot o_i$$

Anpassung der Gewichte nach der Regel: $w' = w + \lambda \cdot (t_x - o_x) \cdot o_i$

Lernrate Lambda	Delta Regel: Logisches OR, Activation ox: Aktivierungsfunktion = 0 wenn Net < 0,5, sonst 1							
	oi	Input-neuronen-Schicht	w	Output-neuron	tx training label	Net = wi*oi	ox	Delta tx-ox
Startphase - Gewicht 0	1 1	1 2	0,000 0,000	1 1	1 1	0,000 0,400	0 0	1,000 1,000
Gewichte nach 1. Trainingphase	1 1	1 2	0,200 0,200	1 1	1 1	0,400 0,800	0 1	1,000 0,000
Gewichte nach 2. Trainingsatz	1 1	1 2	0,400 0,400	1 1	1 1	0,400 0,400	0 1	1,000 0,000
Gewichte nach 3. Trainingsatz	1 0	1 2	0,400 0,400	1 1	1 1	0,400 0,000	0 0	1,000 0,000
Test mit (0,0), (0,1), (1,0) und Startgewichten	0 0	1 2	0,000 0,000	1 0	0 1	0,000 0,000	0 0	0,000 1,000
	0 1	1 2	0,000 0,000	1 1	1 1	0,000 0,000	0 0	0,000 1,000
	1 0	1 2	0,000 0,000	1 1	1 1	0,000 0,000	0 0	0,000 1,000
	0 0	1 2	0,400 0,400	1 0	0 1	0,000 0,400	0 0	0,000 1,000
	0 1	1 2	0,400 0,400	1 1	1 1	0,400 0,800	0 1	0,000 0,000
Test mit (0,0), (0,1), (1,0) und Endgewichten	0 0	1 2	0,400 0,400	1 0	0 1	0,000 0,000	0 0	0,000 0,000
	0 1	1 2	0,400 0,400	1 1	1 1	0,400 0,800	0 1	0,000 0,000
	1 0	1 2	0,400 0,400	1 1	1 1	0,000 0,000	0 0	0,000 0,000
	1 1	1 2	0,400 0,400	1 1	1 1	0,800 0,800	1 1	0,000 0,000
	1 0	1 2	0,400 0,400	1 1	1 1	0,000 0,000	0 0	0,000 0,000

w' = neues Gewicht, w = altes Gewicht, λ (lambda) = Lernrate, t_x = Trainingslabel (Soll- bzw. Lehrwert)

o_x = Ausgabewert (berechneter Wert), o_i = Eingabewert

Wende Deltaregel auf xor an

A	$\neg A$	A	B	A	B	$A \rightarrow B$	$A \leftrightarrow B$
T	F	T	T	T	T	T	T
F	T	T	F	F	T	F	F
		F	T	F	T	T	F
		F	F	F	F	T	T

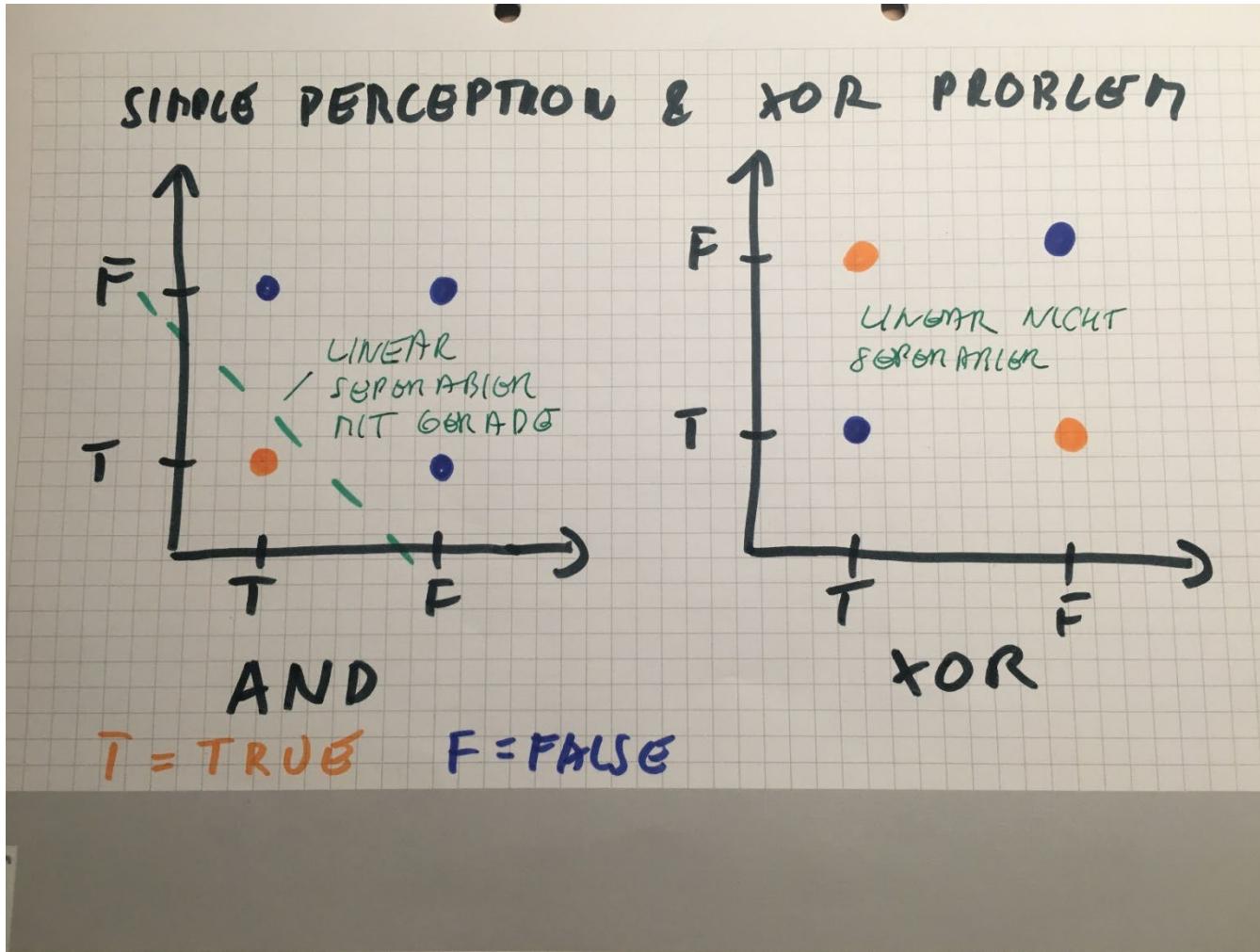
A	B	A	B	$A \mid B$	A	B
T	T		F	F	F	F
T	F		T	T	F	F
F	T		T	T	F	F
F	F		F	T	T	T

$$w_i' = w_i + \lambda \cdot (t_x - o_x) \cdot o_i$$

Implementiere in Excel, Python, ...

w' = neues Gewicht, w = altes Gewicht, λ (lambda) = Lernrate, t_x = Trainingslabel (Soll- bzw. Lehrwert)
 o_x = Ausgabewert (berechneter Wert), o_i = Eingabewert

Aufgabe: Grenzen (Limitierungen) einfacher Perzeptrons



Einfache Perzeptrons sind auf linear trennbare Probleme beschränkt und können keine komplexen Muster oder nichtlinearen Zusammenhänge erfassen.

Nur linear trennbare Probleme

- Es kann nur Daten korrekt klassifizieren, die durch eine gerade Linie (bzw. Hyperebene) trennbar sind.
- Beispiel: Das XOR-Problem kann ein einfaches Perzepron nicht lösen.

Keine versteckten Schichten

- Das Modell besteht nur aus Eingabe- und Ausgabeschicht → keine Hierarchie oder Abstraktion.
- Es kann keine komplexen Muster oder nichtlinearen Zusammenhänge lernen.

Einfache Lernregel (Delta-Regel)

- Die Anpassung erfolgt nur lokal und linear; es gibt keine Rückpropagation von Fehlern durch mehrere Schichten.

Empfindlich gegenüber Rauschen und Skalierung

- Kleine Änderungen in den Eingabedaten können das Ergebnis stark beeinflussen.
- Eingaben müssen meist normalisiert oder skaliert werden.

Keine Gedächtnis- oder Kontextfähigkeit

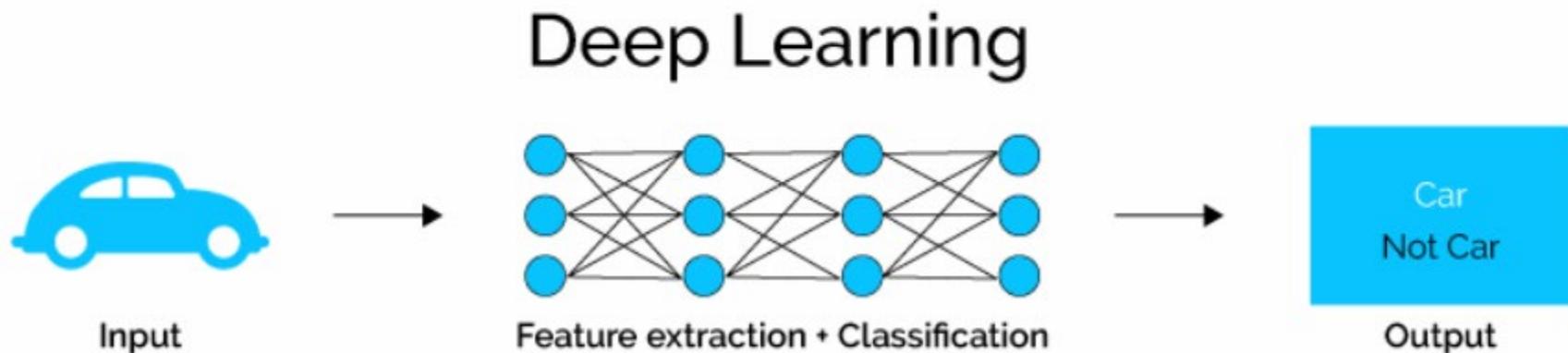
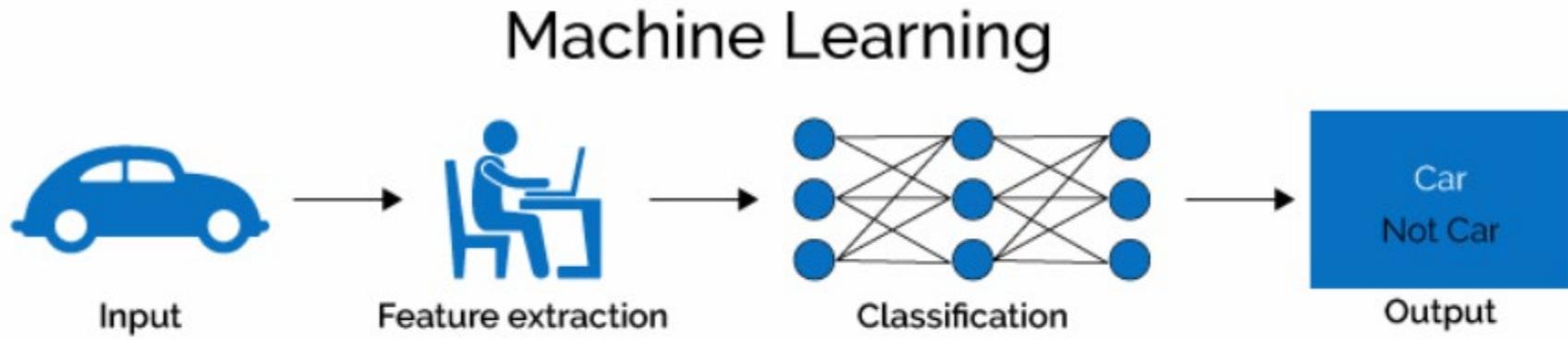
- Es berücksichtigt keine zeitlichen oder sequentiellen Abhängigkeiten (im Gegensatz zu rekurrenten Netzen).



CNNs Convolutional Neural Networks Bilderkennung / Klassifizierung am Beispiel Mikrometeoriten

Tiefe neuronale Netze – deep neural networks

- Anstatt Merkmale manuell zu definieren und ein neuronales Netz darauf zu trainieren, fügen wir weitere Schichten hinzu und lassen das Netz die Merkmale zusammen mit der Klassifizierungsaufgabe lernen



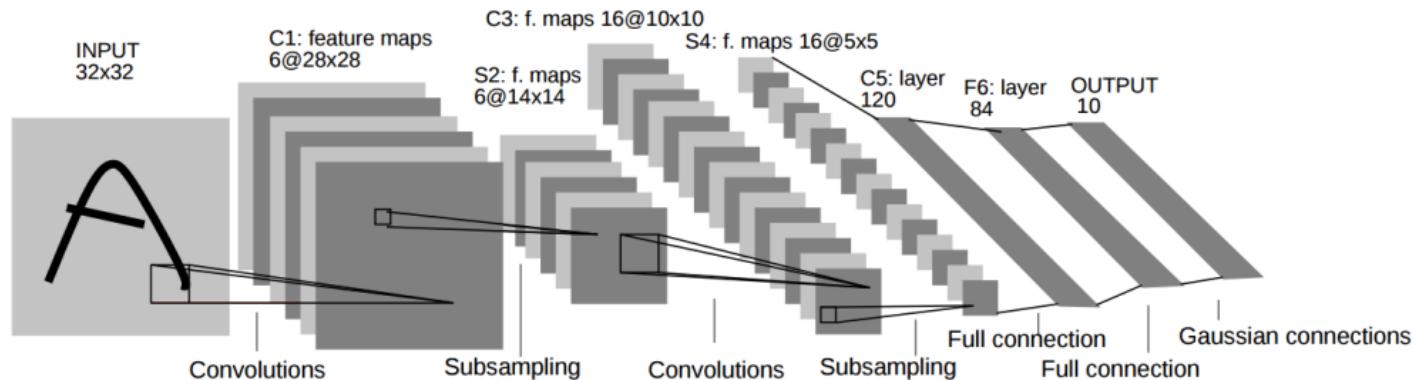
<https://medium.com/swlh/ill-tell-you-why-deep-learning-is-so-popular-and-in-demand-5aca72628780>

Deep Neural Networks

- Weitaus komplexer als ein einfaches MLP (tiefer und umfassender)
- Sie sind in der Lage, komplexe nicht-lineare Beziehungen in Daten zu erfassen, die von einfacheren Modellen nicht gelernt werden können.
- Das Training ist teuer (erfordert normalerweise GPUs)
- hoher Speicherbedarf und Rechenaufwand des Modells während des Trainings und der Vorhersage -> nicht für Geräte mit geringer Leistung geeignet
- Es gibt eine Vielzahl von CPU- und GPU-basierten Frameworks für das einfache Training von Deep Networks (Tensorflow, Keras, Theano, Caffe,)
- Sie benötigen enorme Datenmengen (>10 Mio. Proben), um für viele Anwendungen korrekt trainiert zu werden.
- Die resultierenden Modelle sind sehr schwer zu interpretieren

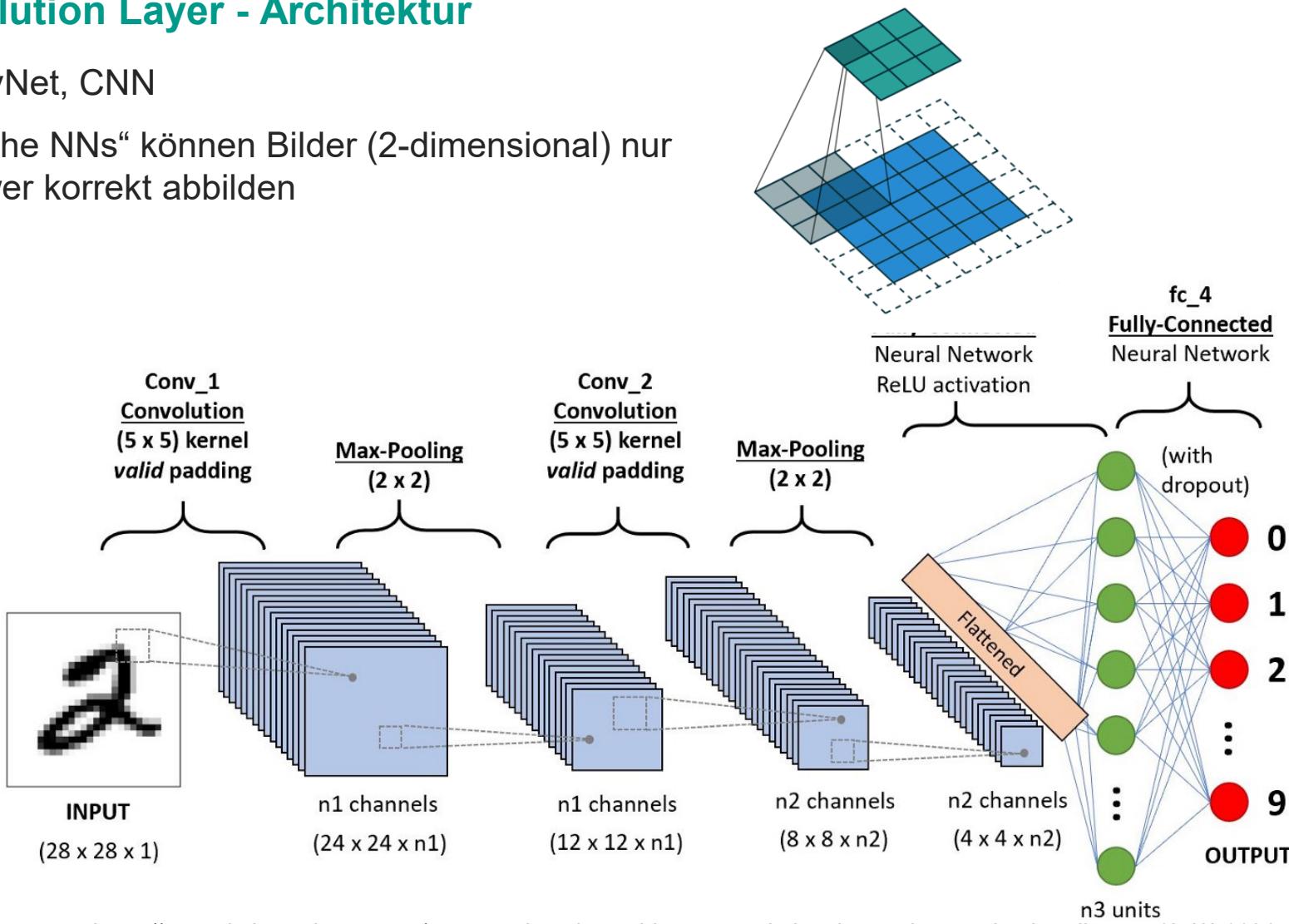
Tiefe Faltungsneuronale Netze (CNN)

- Multi Layer Perceptrons sind nicht geeignet für hochdimensionale Eingabedaten (z.B. Bilder, Videodaten) -> die Anzahl der Parameter wäre unrealistisch groß
- Eingabebild (200x200 px) -> 40000 Eingabewerte, die mit min. 40000 Neuronen in der ersten versteckten Schicht verbunden werden müssen:
 $40000 \times 40000 = 160\text{mio}$ Parameter für 1 vollständig verbundene Schicht
- Faltungsneuronale Netze ersetzen die ersten n "voll verbundenen" Schichten durch sogenannte Faltungsschichten
- Faltungsschichten verwenden "Weight Sharing", um die Anzahl der Parameter bei gleichbleibender Leistung



Convolution Layer - Architektur

- ConvNet, CNN
- „Flache NNs“ können Bilder (2-dimensional) nur schwer korrekt abbilden

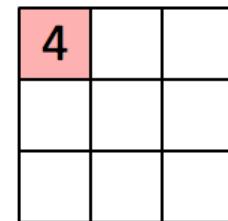
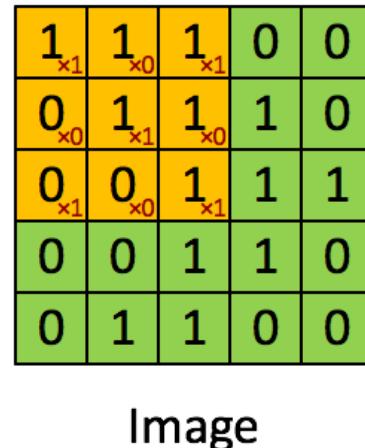


<https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53>

Convolution Layer / Filter

- Filter (Gewichte der Neuronen; Convolution-Kernel) sollen bestimmte Eigenschaften des Bildes betonen, z.B. Kanten, Linien
- Diese Filter werden entweder bei einfachen Modellen von Hand vorgegeben oder werden im Training erlernt
- Der Kernel der Größe $i \times j$ wird über das Eingabefeld (meist Bild mit $n \times m$ Pixel, 2 Dimensionen) bewegt
- Bei RGB werden diese auf jeden Kanal angewendet

- Der Gewichte des Kernels werden mit den passenden Werten der Eingabeschicht multipliziert und summiert und ergeben die entsprechende Aktivierung
- Größe des Kernels (Windows) bestimmt die Größe der Ausgabematrix: $n-i+1 \times m-j+1$
- Der Gewichte des Kernels werden mit den passenden Werten der Eingabe-schicht multipliziert und summiert und ergeben die entsprechende Aktivierung

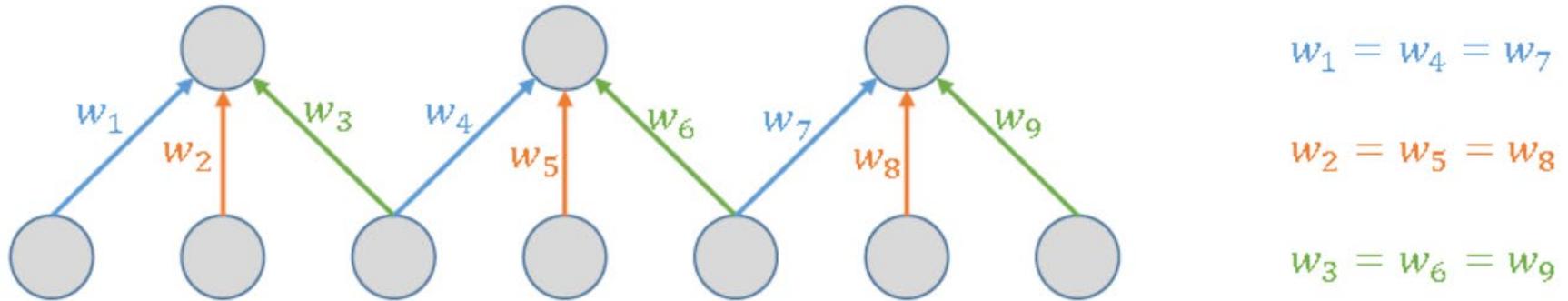


Convolved Feature

<https://www.kaggle.com/amarjeet007/visualize-cnn-with-keras>

Filter trainieren

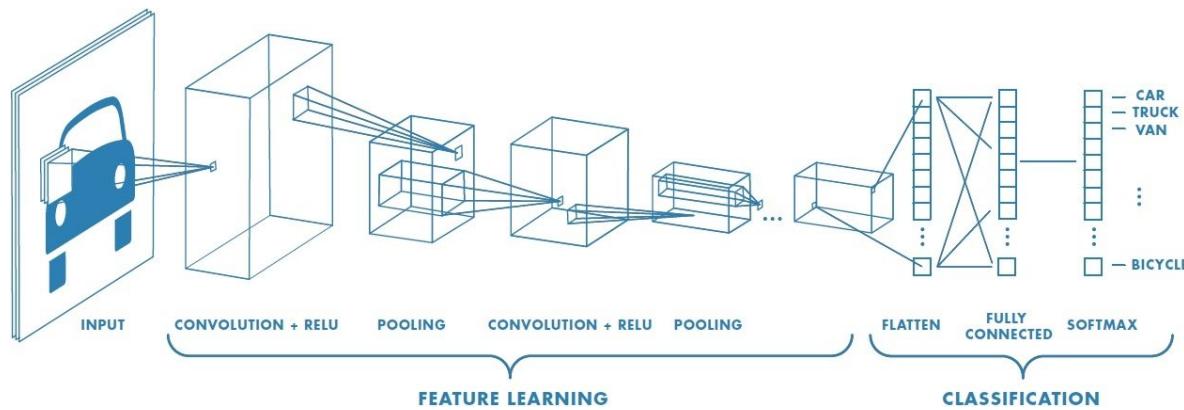
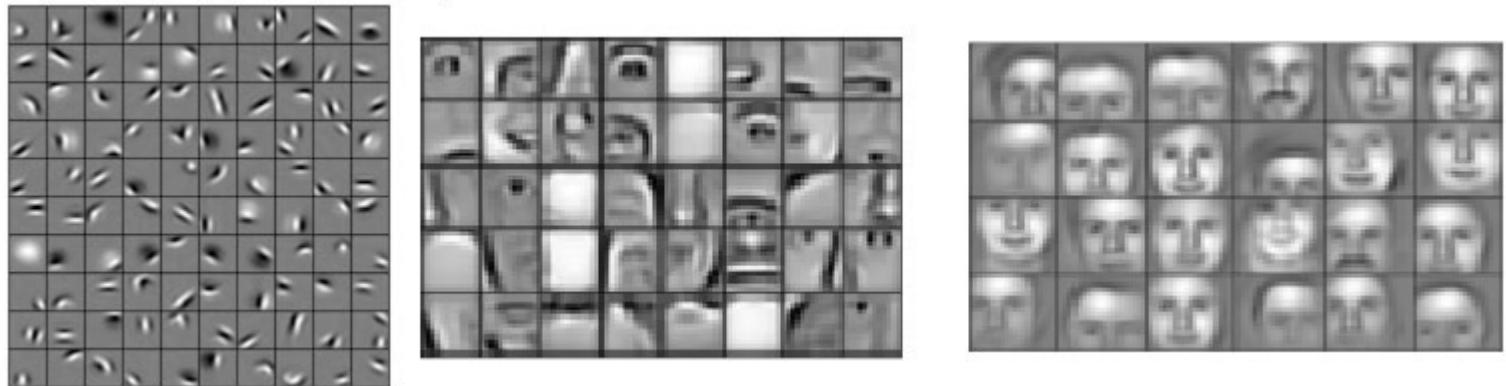
- Mittels Backpropagation mit Nebenbedingungen
- Für jeden der drei Kanäle eines Farbbildes werden verschiedene Conv2D-Filter erstellt
- Die Filter für jede Ebene werden nach dem Zufallsprinzip auf der Grundlage einer Normal- oder Gaußverteilung initialisiert.



<https://www.kaggle.com/amarjeet007/visualize-cnn-with-keras>

Deep Convolutional Neural Networks (CNN)

- Feature maps („Filter“)



<https://ch.mathworks.com/fr/discovery/convolutional-neural-network.html>

Feature maps („Filter“) einfach

- Kernel wird über das Eingabefeld bewegt
- Conv2D(channels=1, kernel_size=(3,3))

<u>Input</u>	<u>Kernel</u>	<u>Output</u>
1 3 2 1	1 2 3	
1 3 3 1	0 1 0	23 22
2 1 1 3	2 1 2	31 26
3 2 3 3		

<https://medium.com/apache-mxnet/multi-channel-convolutions-explained-with-ms-excel-9bbf8eb77108>

Padding

- Nullen am Rand einfügen
- Beispiel: Padding 1x1 (rechts, links, oben, unten)
- Conv2D(channels=1, kernel_size=(3,3), padding=(1,1))

<u>Input</u>						<u>Kernel</u>			<u>Output</u>			
0	0	0	0	0	0							
0	1	3	2	1	0	1	2	3	8	14	13	8
0	1	3	3	1	0	0	1	0	16	23	22	10
0	2	1	1	3	0	2	1	2	20	31	26	17
0	3	2	3	3	0				10	9	15	10
0	0	0	0	0	0							

<https://medium.com/apache-mxnet/multi-channel-convolutions-explained-with-ms-excel-9bbf8eb77108>

Beispiel senkrechten Strich mit Kernel erkennen

- Nullen am Randeinfügen
- Beispiel: Padding 1x1 (rechts, links, oben, unten)
- Conv2D(channels=1, kernel_size=(3,3), padding=(1,1))

<u>Input</u>	<u>Kernel</u>	<u>Output</u>
0 0 0 0 0 0		
0 1 3 2 1 0	0 2 0	4 12 6 4
0 1 3 1 1 0	0 2 0	8 18 8 6
0 2 3 1 1 0	0 2 0	8 16 6 6
0 1 2 1 1 0		6 10 4 4
0 0 0 0 0 0		

<https://medium.com/apache-mxnet/multi-channel-convolutions-explained-with-ms-excel-9bbf8eb77108>

Feature maps („Filter“) – RGB Filter

- Beispiel: Senkrechter Strich

<u>Input</u>	<u>Kernel</u>	<u>Intermediate Output</u>																																												
<table border="1"> <tr><td>1</td><td>0</td><td>1</td><td>0</td><td>2</td></tr> <tr><td>1</td><td>1</td><td>3</td><td>2</td><td>1</td></tr> <tr><td>1</td><td>1</td><td>0</td><td>1</td><td>1</td></tr> <tr><td>2</td><td>3</td><td>2</td><td>1</td><td>3</td></tr> <tr><td>0</td><td>2</td><td>0</td><td>1</td><td>0</td></tr> </table>	1	0	1	0	2	1	1	3	2	1	1	1	0	1	1	2	3	2	1	3	0	2	0	1	0	<table border="1"> <tr><td>0</td><td>1</td><td>0</td></tr> <tr><td>0</td><td>1</td><td>0</td></tr> <tr><td>0</td><td>1</td><td>0</td></tr> </table>	0	1	0	0	1	0	0	1	0	<table border="1"> <tr><td>2</td><td>4</td><td>3</td></tr> <tr><td>5</td><td>5</td><td>4</td></tr> <tr><td>6</td><td>2</td><td>3</td></tr> </table>	2	4	3	5	5	4	6	2	3	
1	0	1	0	2																																										
1	1	3	2	1																																										
1	1	0	1	1																																										
2	3	2	1	3																																										
0	2	0	1	0																																										
0	1	0																																												
0	1	0																																												
0	1	0																																												
2	4	3																																												
5	5	4																																												
6	2	3																																												
<table border="1"> <tr><td>1</td><td>0</td><td>0</td><td>1</td><td>0</td></tr> <tr><td>2</td><td>0</td><td>1</td><td>2</td><td>0</td></tr> <tr><td>3</td><td>1</td><td>1</td><td>3</td><td>0</td></tr> <tr><td>0</td><td>3</td><td>0</td><td>3</td><td>2</td></tr> <tr><td>1</td><td>0</td><td>3</td><td>2</td><td>1</td></tr> </table>	1	0	0	1	0	2	0	1	2	0	3	1	1	3	0	0	3	0	3	2	1	0	3	2	1	<table border="1"> <tr><td>0</td><td>1</td><td>0</td></tr> <tr><td>0</td><td>1</td><td>0</td></tr> <tr><td>0</td><td>1</td><td>0</td></tr> </table>	0	1	0	0	1	0	0	1	0	<table border="1"> <tr><td>1</td><td>2</td><td>6</td></tr> <tr><td>4</td><td>2</td><td>8</td></tr> <tr><td>4</td><td>4</td><td>8</td></tr> </table>	1	2	6	4	2	8	4	4	8	<u>Output</u>
1	0	0	1	0																																										
2	0	1	2	0																																										
3	1	1	3	0																																										
0	3	0	3	2																																										
1	0	3	2	1																																										
0	1	0																																												
0	1	0																																												
0	1	0																																												
1	2	6																																												
4	2	8																																												
4	4	8																																												
<table border="1"> <tr><td>2</td><td>0</td><td>1</td><td>2</td><td>1</td></tr> <tr><td>3</td><td>3</td><td>1</td><td>3</td><td>2</td></tr> <tr><td>2</td><td>1</td><td>1</td><td>1</td><td>0</td></tr> <tr><td>3</td><td>1</td><td>3</td><td>2</td><td>0</td></tr> <tr><td>1</td><td>1</td><td>2</td><td>1</td><td>1</td></tr> </table>	2	0	1	2	1	3	3	1	3	2	2	1	1	1	0	3	1	3	2	0	1	1	2	1	1	<table border="1"> <tr><td>0</td><td>1</td><td>0</td></tr> <tr><td>0</td><td>1</td><td>0</td></tr> <tr><td>0</td><td>1</td><td>0</td></tr> </table>	0	1	0	0	1	0	0	1	0	<table border="1"> <tr><td>4</td><td>3</td><td>6</td></tr> <tr><td>5</td><td>5</td><td>6</td></tr> <tr><td>3</td><td>6</td><td>4</td></tr> </table>	4	3	6	5	5	6	3	6	4	
2	0	1	2	1																																										
3	3	1	3	2																																										
2	1	1	1	0																																										
3	1	3	2	0																																										
1	1	2	1	1																																										
0	1	0																																												
0	1	0																																												
0	1	0																																												
4	3	6																																												
5	5	6																																												
3	6	4																																												

<https://medium.com/apache-mxnet/multi-channel-convolutions-explained-with-ms-excel-9bbf8eb77108>

Feature maps („Filter“)

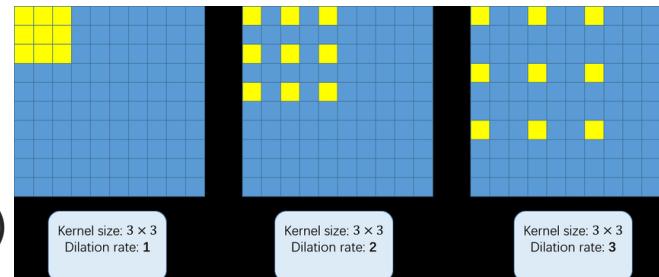
- Beispiel: Pfeil

<u>Input</u>	<u>Kernel</u>	<u>Intermediate Output</u>	<u>Output</u>																																																				
<table border="1"> <tr><td>1</td><td>0</td><td>1</td><td>0</td><td>2</td></tr> <tr><td>1</td><td>1</td><td>3</td><td>2</td><td>1</td></tr> <tr><td>1</td><td>1</td><td>0</td><td>1</td><td>1</td></tr> <tr><td>2</td><td>3</td><td>2</td><td>1</td><td>3</td></tr> <tr><td>0</td><td>2</td><td>0</td><td>1</td><td>0</td></tr> </table>	1	0	1	0	2	1	1	3	2	1	1	1	0	1	1	2	3	2	1	3	0	2	0	1	0	<table border="1"> <tr><td>0</td><td>1</td><td>0</td></tr> <tr><td>0</td><td>0</td><td>2</td></tr> <tr><td>0</td><td>1</td><td>0</td></tr> </table>	0	1	0	0	0	2	0	1	0	<table border="1"> <tr><td>7</td><td>5</td><td>3</td></tr> <tr><td>4</td><td>7</td><td>5</td></tr> <tr><td>7</td><td>2</td><td>8</td></tr> </table>	7	5	3	4	7	5	7	2	8										
1	0	1	0	2																																																			
1	1	3	2	1																																																			
1	1	0	1	1																																																			
2	3	2	1	3																																																			
0	2	0	1	0																																																			
0	1	0																																																					
0	0	2																																																					
0	1	0																																																					
7	5	3																																																					
4	7	5																																																					
7	2	8																																																					
<table border="1"> <tr><td>1</td><td>0</td><td>0</td><td>1</td><td>0</td></tr> <tr><td>2</td><td>0</td><td>1</td><td>2</td><td>0</td></tr> <tr><td>3</td><td>1</td><td>1</td><td>3</td><td>0</td></tr> <tr><td>0</td><td>3</td><td>0</td><td>3</td><td>2</td></tr> <tr><td>1</td><td>0</td><td>3</td><td>2</td><td>1</td></tr> </table>	1	0	0	1	0	2	0	1	2	0	3	1	1	3	0	0	3	0	3	2	1	0	3	2	1	<table border="1"> <tr><td>2</td><td>1</td><td>0</td></tr> <tr><td>0</td><td>0</td><td>0</td></tr> <tr><td>0</td><td>3</td><td>0</td></tr> </table>	2	1	0	0	0	0	0	3	0	<table border="1"> <tr><td>5</td><td>3</td><td>10</td></tr> <tr><td>13</td><td>1</td><td>13</td></tr> <tr><td>7</td><td>12</td><td>11</td></tr> </table>	5	3	10	13	1	13	7	12	11	<table border="1"> <tr><td>19</td><td>13</td><td>15</td></tr> <tr><td>28</td><td>16</td><td>20</td></tr> <tr><td>23</td><td>18</td><td>25</td></tr> </table>	19	13	15	28	16	20	23	18	25
1	0	0	1	0																																																			
2	0	1	2	0																																																			
3	1	1	3	0																																																			
0	3	0	3	2																																																			
1	0	3	2	1																																																			
2	1	0																																																					
0	0	0																																																					
0	3	0																																																					
5	3	10																																																					
13	1	13																																																					
7	12	11																																																					
19	13	15																																																					
28	16	20																																																					
23	18	25																																																					
<table border="1"> <tr><td>2</td><td>0</td><td>1</td><td>2</td><td>1</td></tr> <tr><td>3</td><td>3</td><td>1</td><td>3</td><td>2</td></tr> <tr><td>2</td><td>1</td><td>1</td><td>1</td><td>0</td></tr> <tr><td>3</td><td>1</td><td>3</td><td>2</td><td>0</td></tr> <tr><td>1</td><td>1</td><td>2</td><td>1</td><td>1</td></tr> </table>	2	0	1	2	1	3	3	1	3	2	2	1	1	1	0	3	1	3	2	0	1	1	2	1	1	<table border="1"> <tr><td>1</td><td>0</td><td>0</td></tr> <tr><td>1</td><td>0</td><td>0</td></tr> <tr><td>0</td><td>0</td><td>2</td></tr> </table>	1	0	0	1	0	0	0	0	2	<table border="1"> <tr><td>7</td><td>5</td><td>2</td></tr> <tr><td>11</td><td>8</td><td>2</td></tr> <tr><td>9</td><td>4</td><td>6</td></tr> </table>	7	5	2	11	8	2	9	4	6										
2	0	1	2	1																																																			
3	3	1	3	2																																																			
2	1	1	1	0																																																			
3	1	3	2	0																																																			
1	1	2	1	1																																																			
1	0	0																																																					
1	0	0																																																					
0	0	2																																																					
7	5	2																																																					
11	8	2																																																					
9	4	6																																																					

<https://medium.com/apache-mxnet/multi-channel-convolutions-explained-with-ms-excel-9bbf8eb77108>

Dilatation

- Kernel wird erweitert um einen Faktor $i \times j$
- Conv2D(channels=1, kernel_size=(3,3), dilation=(2,2))



<u>Input</u>	<u>Kernel</u>	<u>Output</u>
1 3 2 1 2 2 1		
1 3 3 1 2 1 2		
2 1 1 3 1 1 3	1 2 3	21 26 16
3 2 3 3 2 3 1	0 1 0	26 21 26
2 3 1 2 2 2 1	2 1 2	23 21 27
1 1 2 2 3 3 2		
3 2 3 1 3 2 2		

<https://medium.com/apache-mxnet/multi-channel-convolutions-explained-with-ms-excel-9bbf8eb77108>

Strides (Streifen, Schrittweite)

- Verschiebung Kernelanwendung um jeweils i Einheiten
- Im Beispiel um 2 Einheiten
- Conv2D(channels=1, kernel_size=(3,3), strides=(2,2))

<u>Input</u>	<u>Kernel</u>	<u>Expanded Output</u>	<u>Output</u>
1 3 2 1 3 2 2 1 1 3 2 3 2 3 1	1 2 3 0 1 0 2 1 2	23 18 18 18 21	23 18 18 21

<https://medium.com/apache-mxnet/multi-channel-convolutions-explained-with-ms-excel-9bbf8eb77108>

Padding, Strides, Dilatation

- Conv2D(channels=1, kernel_size=(3,3), padding=(2,2), strides=(2,2), dilation=(2,2))

	<u>Input</u>											<u>Kernel</u>			<u>Expanded Output</u>				<u>Output</u>			
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	5	9	11	6	5	9	11	6
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	14	21	16	12	14	21	16	12
0	0	1	3	3	1	2	1	2	0	0	0	1	2	3	18	23	27	16	18	23	27	16
0	0	2	1	1	3	1	1	3	0	0	0	0	1	0	10	13	11	6	10	13	11	6
0	0	3	2	3	3	2	3	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	2	3	1	2	2	2	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	1	1	2	2	3	3	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	3	2	3	1	3	2	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

<https://medium.com/apache-mxnet/multi-channel-convolutions-explained-with-ms-excel-9bbf8eb77108>

Pooling Layer

- Gruppe von Neuronen einer Ebene zu einem Neuron zusammenfassen (Unterstichprobe des Bildes)
- Vermeidet, dass kleine Änderungen wie Verschiebungen, Rotationen das Ergebnis beeinflussen
- Besitzt keine Gewichte
- Es wird wieder ein $i \times i$ Fenster über die Eingabeschicht $n \times m$ bewegt
- Es werden $i \times i$ Pixel (Pooling-Kernel) zusammengefasst
 - durch Mittelwertbildung
 - durch maximalen Wert
- Ausgabeschicht dann Matrix mit $n-i+1 \times m-i+1$

- MaxPooling2D(pool_size=(2, 2))

Max Pooling

<u>Input</u>	<u>Output</u>															
<table border="1"><tr><td>21</td><td>8</td><td>8</td><td>12</td></tr><tr><td>12</td><td>19</td><td>9</td><td>7</td></tr><tr><td>8</td><td>10</td><td>4</td><td>3</td></tr><tr><td>18</td><td>12</td><td>9</td><td>10</td></tr></table>	21	8	8	12	12	19	9	7	8	10	4	3	18	12	9	10
21	8	8	12													
12	19	9	7													
8	10	4	3													
18	12	9	10													

 | | | |----|----| | 21 | 12 | | 18 | 10 | |

21	8	8	12
12	19	9	7
8	10	4	3
18	12	9	10

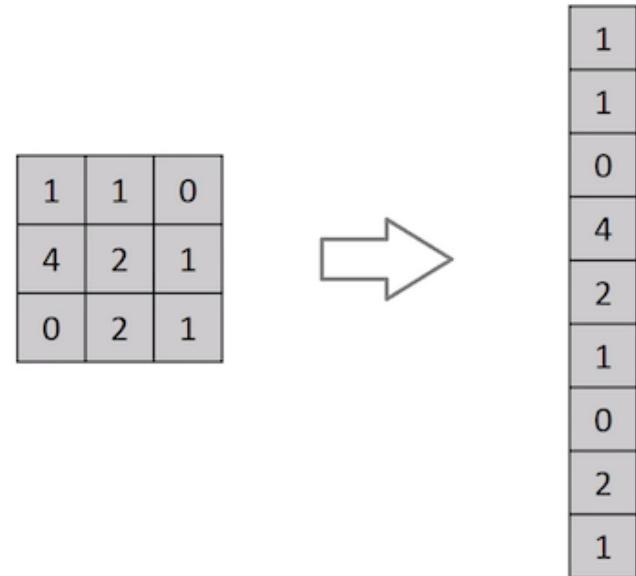
Average Pooling

<u>Input</u>	<u>Output</u>
--------------	---------------

21	8	8	12
12	19	9	7
8	10	4	3
18	12	9	10

Dropout Layer

- Regularisierungsmethode um Overfitting zu vermeiden
- Schaltet zufällig einen bestimmten %-Satz von Eingabeneuronen in einer Schicht aus, wird während des Trainings ignoriert (Drop-Out-Rate)
- Nach dem Trainieren werden keine Neuronen weggelassen

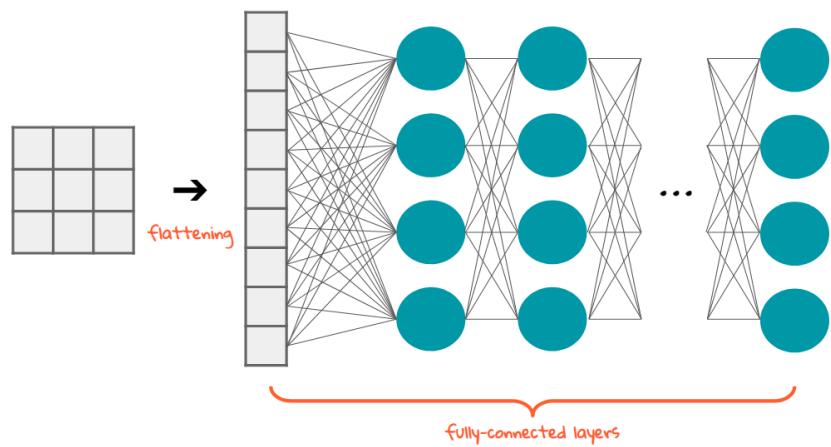


Flatten Layer

- Konvertiert die Conv Schicht in eine eindimensionalen Schicht

Dense Layer

- Voll verbundene Schichten



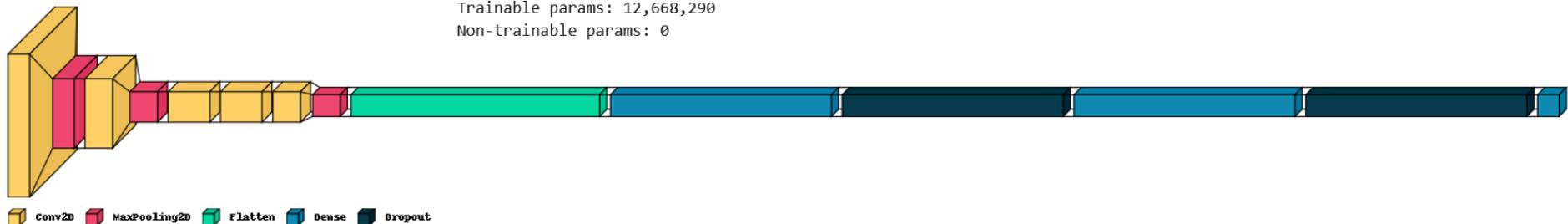
<https://towardsdatascience.com/the-most-intuitive-and-easiest-guide-for-convolutional-neural-network-3607be47480>

```
1. model = tf.keras.models.Sequential(name=optim)
2. model.add(tf.keras.layers.Conv2D(filters=96, kernel_size=(11, 11), strides=4, padding="valid", activation="relu",
3. input_shape=(141, 141, 3)))
4. model.add(tf.keras.layers.MaxPooling2D(pool_size=(3, 3), strides=2, padding="valid"))
5. model.add(tf.keras.layers.Conv2D(filters=256, kernel_size=(5, 5), strides=1, padding="same", activation="relu"))
6. model.add(tf.keras.layers.MaxPooling2D(pool_size=(3, 3), strides=2, padding="valid"))
7. model.add(tf.keras.layers.Conv2D(filters=384, kernel_size=(3, 3), strides=1, padding="same", activation="relu"))
8. model.add(tf.keras.layers.Conv2D(filters=384, kernel_size=(3, 3), strides=1, padding="same", activation="relu"))
9. model.add(tf.keras.layers.Conv2D(filters=256, kernel_size=(3, 3), strides=1, padding="same", activation="relu"))
10. model.add(tf.keras.layers.MaxPooling2D(pool_size=(3, 3), strides=2, padding="valid"))
11. model.add(tf.keras.layers.Flatten())
12. model.add(tf.keras.layers.Dense(units=dense_01, activation="relu", bias_initializer="random_normal"))
13. model.add(tf.keras.layers.Dropout(rate=dropout_01))
14. model.add(tf.keras.layers.Dense(units=dense_02, activation="relu", bias_initializer="random_normal"))
15. model.add(tf.keras.layers.Dropout(rate=dropout_02))
16. model.add(tf.keras.layers.Dense(units=2, activation="softmax", bias_initializer="random_normal"))
```

dense_01 = 8192
dense_02 = 6144
dropout_01 = 0.35
dropout_02 = 0.047

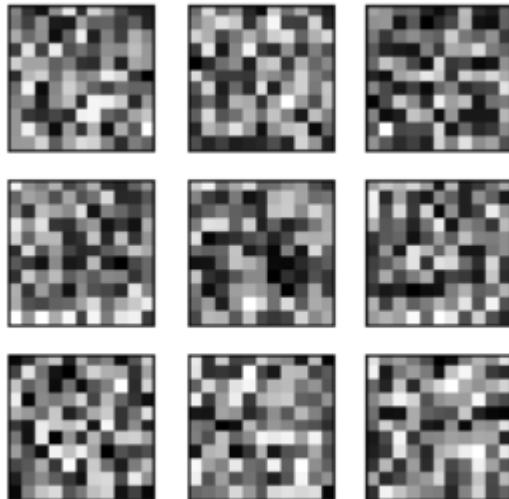
Model: "bo-adam"

Layer (type)	Output Shape	Param #
<hr/>		
conv2d (Conv2D)	(None, 33, 33, 96)	34944
<hr/>		
max_pooling2d (MaxPooling2D)	(None, 16, 16, 96)	0
<hr/>		
conv2d_1 (Conv2D)	(None, 16, 16, 256)	614656
<hr/>		
max_pooling2d_1 (MaxPooling2D)	(None, 7, 7, 256)	0
<hr/>		
conv2d_2 (Conv2D)	(None, 7, 7, 384)	885120
<hr/>		
conv2d_3 (Conv2D)	(None, 7, 7, 384)	1327488
<hr/>		
conv2d_4 (Conv2D)	(None, 7, 7, 256)	884992
<hr/>		
max_pooling2d_2 (MaxPooling2D)	(None, 3, 3, 256)	0
<hr/>		
flatten (Flatten)	(None, 2304)	0
<hr/>		
dense (Dense)	(None, 2048)	4720640
<hr/>		
dropout (Dropout)	(None, 2048)	0
<hr/>		
dense_1 (Dense)	(None, 2048)	4196352
<hr/>		
dropout_1 (Dropout)	(None, 2048)	0
<hr/>		
dense_2 (Dense)	(None, 2)	4098
<hr/>		
Total params: 12,668,290		
Trainable params: 12,668,290		
Non-trainable params: 0		



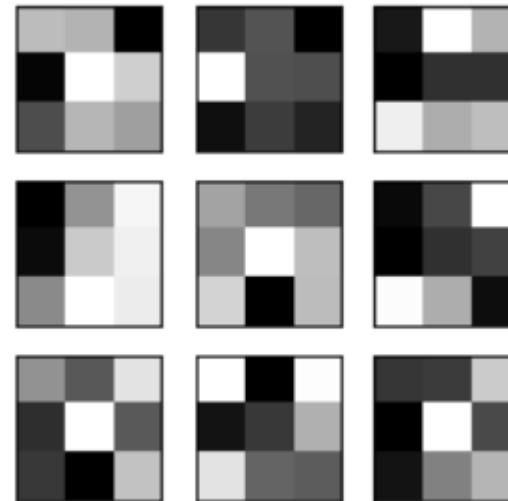
Filterbeispiel MM Model Layer 0

- model.add(tf.keras.layers.Conv2D(filters=96, kernel_size=(11, 11), ...))
Layer 0
- RGB getrennt



Layer 5

- model.add(tf.keras.layers.Conv2D(filters=384, kernel_size=(3, 3), strides=1,



<https://ch.mathworks.com/fr/discovery/convolutional-neural-network.html>

Größe 976 x 976 Pixel



Original

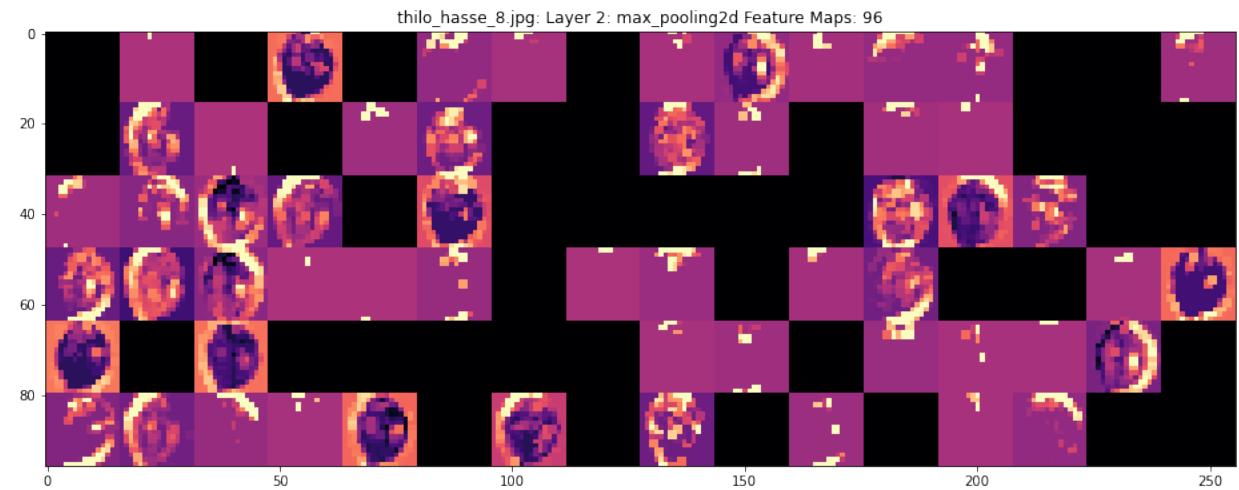
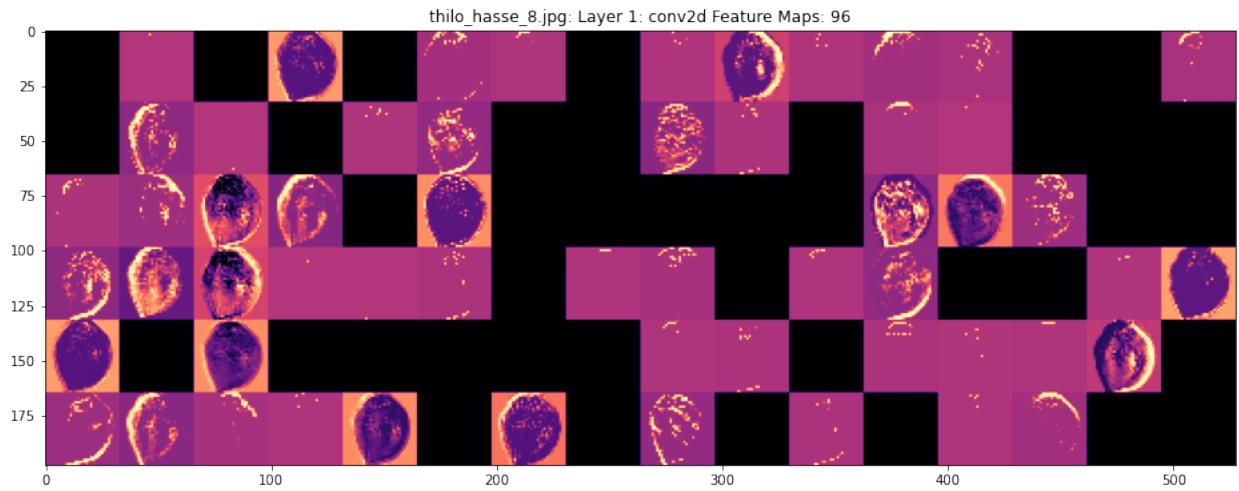


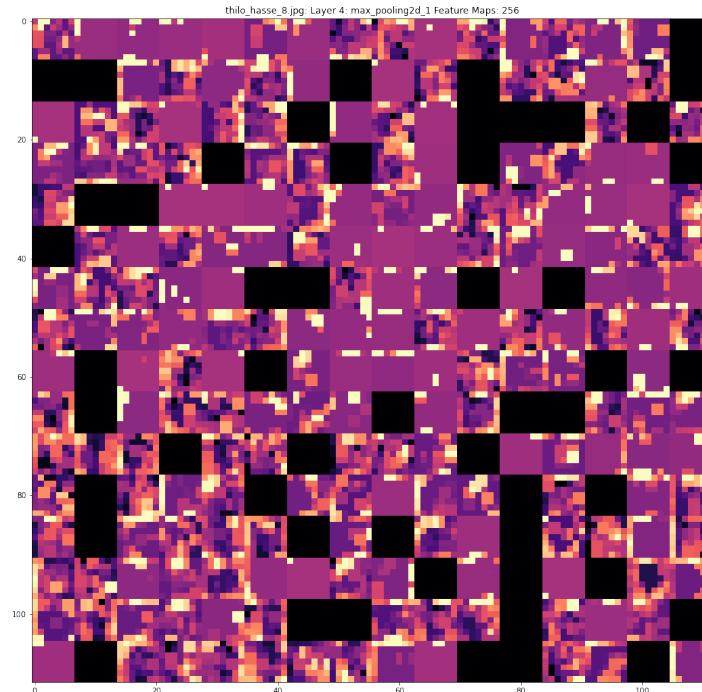
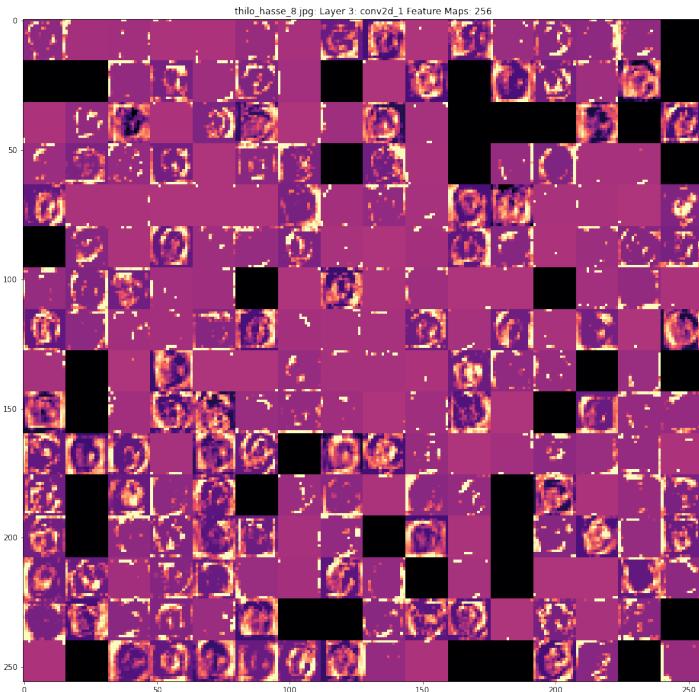
Vorverarbeitet

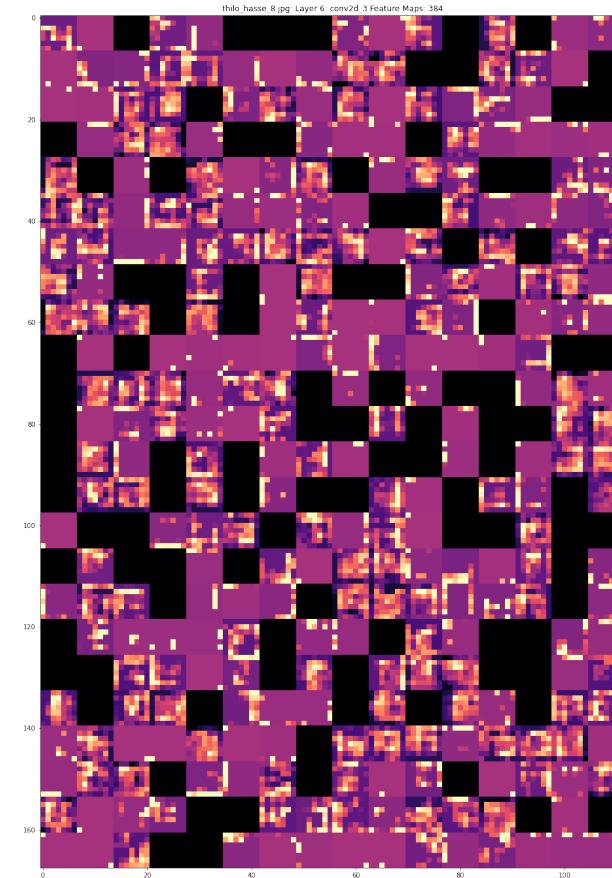
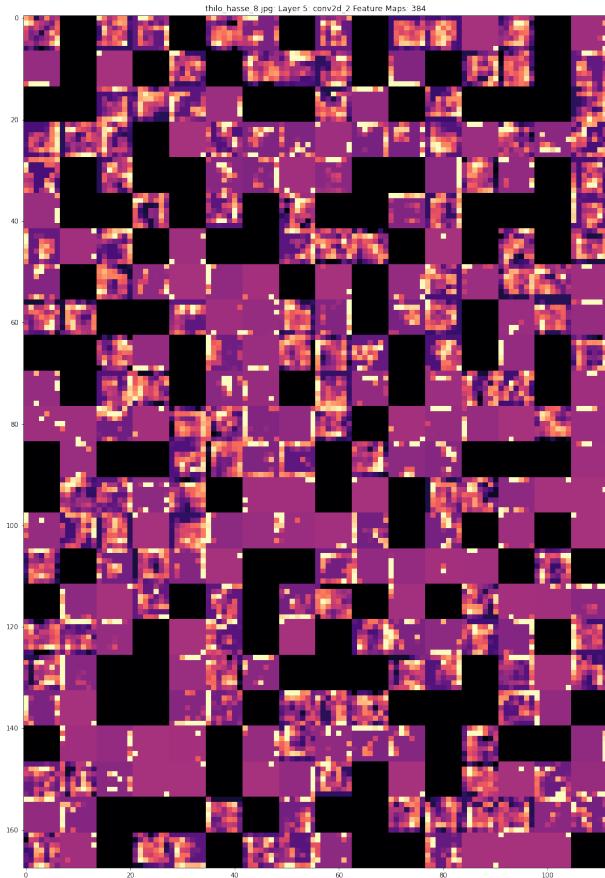
Größe 141 x 141 Pixel

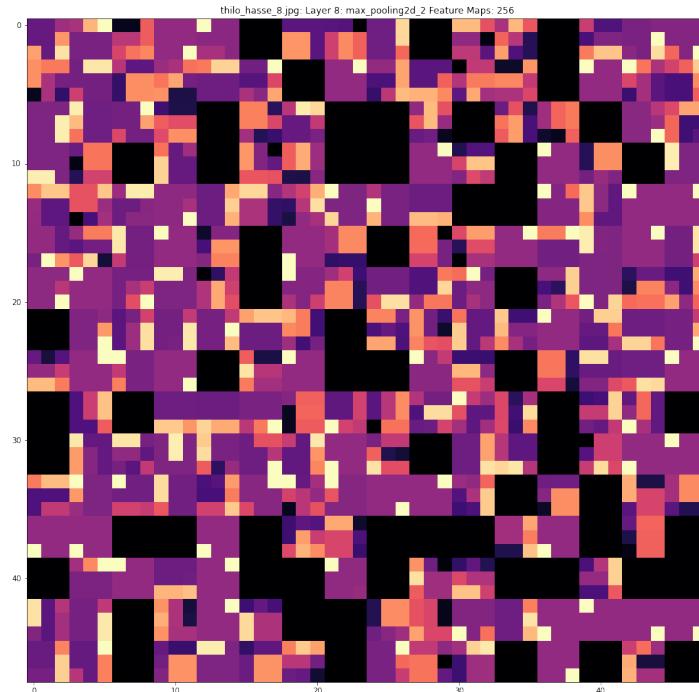
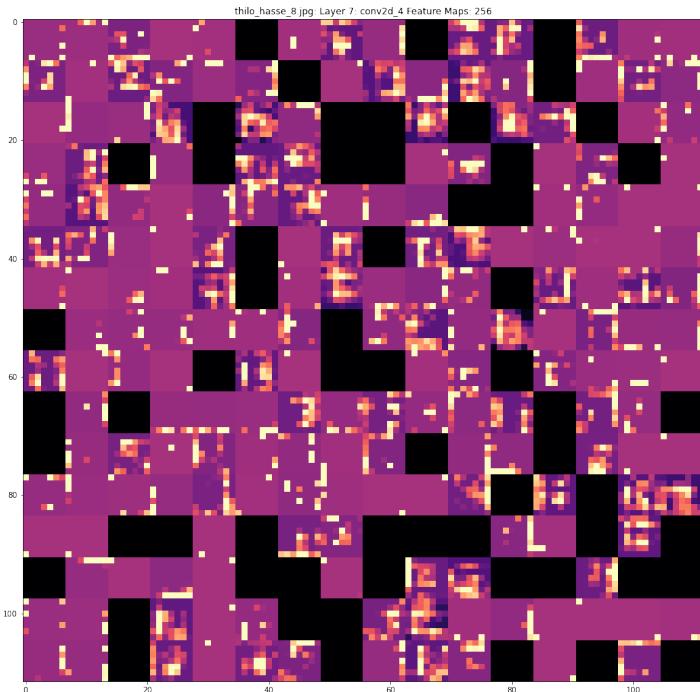


RGB Kanäle

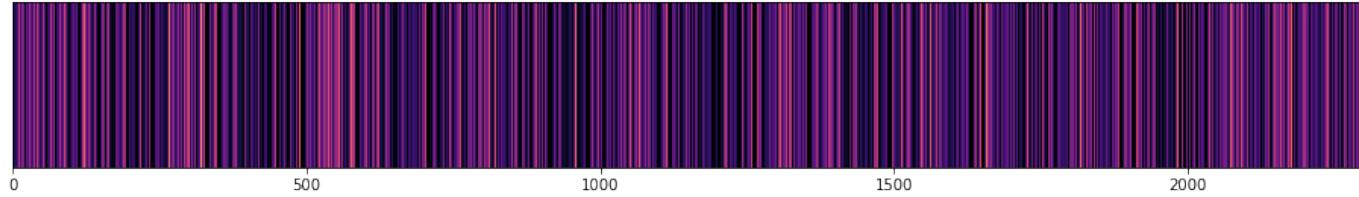




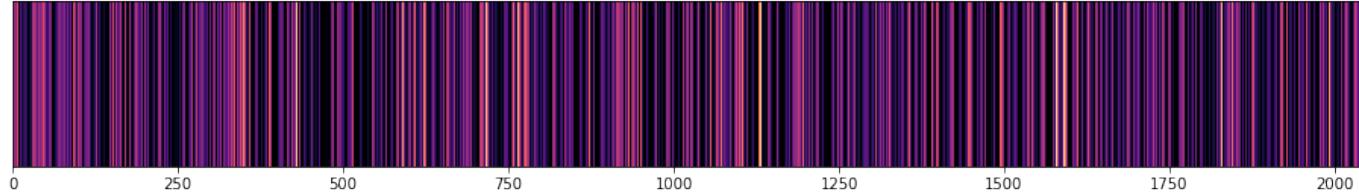




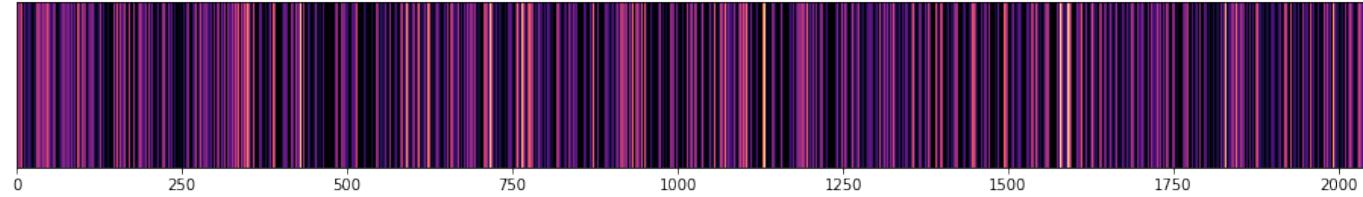
thilo_hasse_8.jpg: Layer 9: flatten Feature Maps: 2304



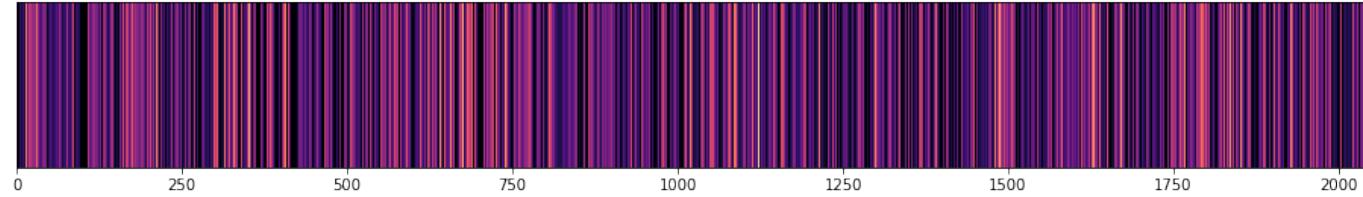
thilo_hasse_8.jpg: Layer 10: dense Feature Maps: 2048



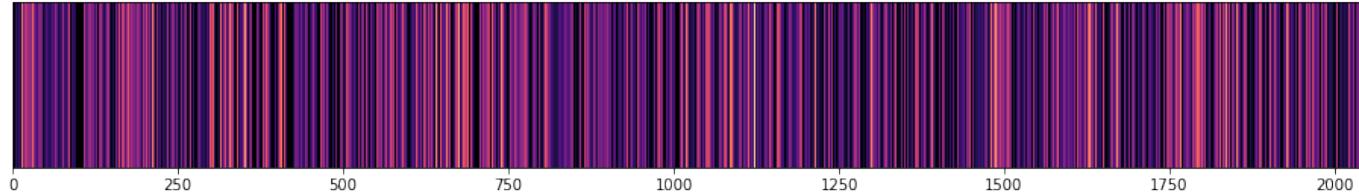
thilo_hasse_8.jpg: Layer 11: dropout Feature Maps: 2048



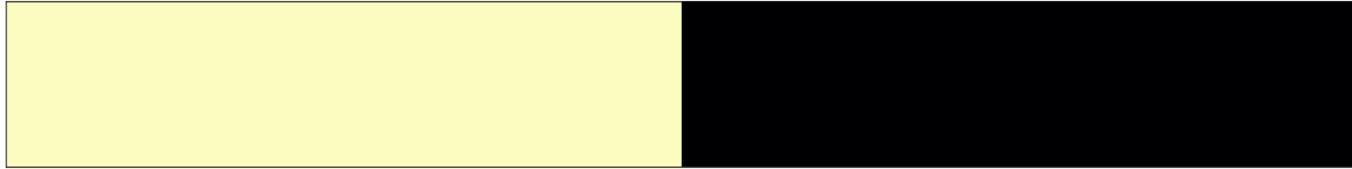
thilo_hasse_8.jpg: Layer 12: dense_1 Feature Maps: 2048



thilo_hasse_8.jpg: Layer 13: dropout_1 Feature Maps: 2048



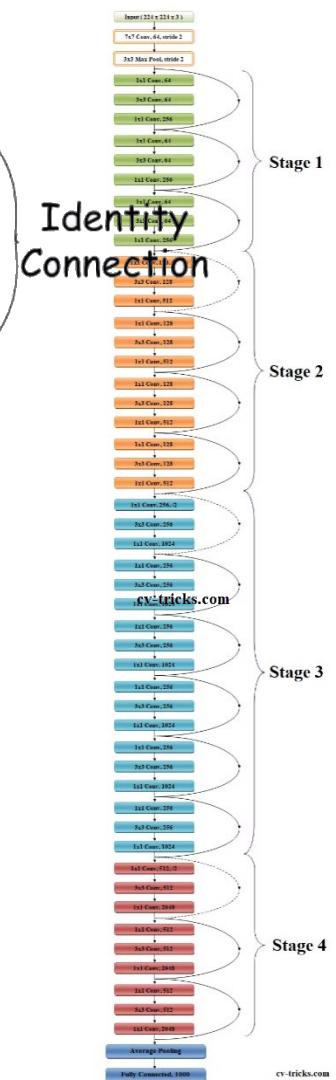
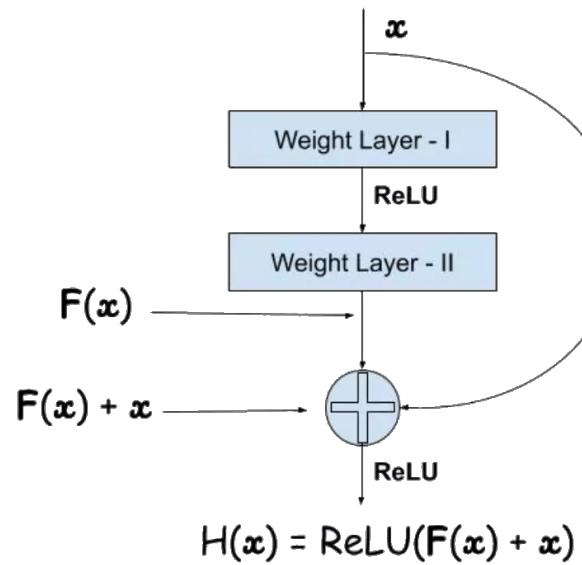
thilo_hasse_8.jpg: Layer 14: dense_2 Feature Maps: 2

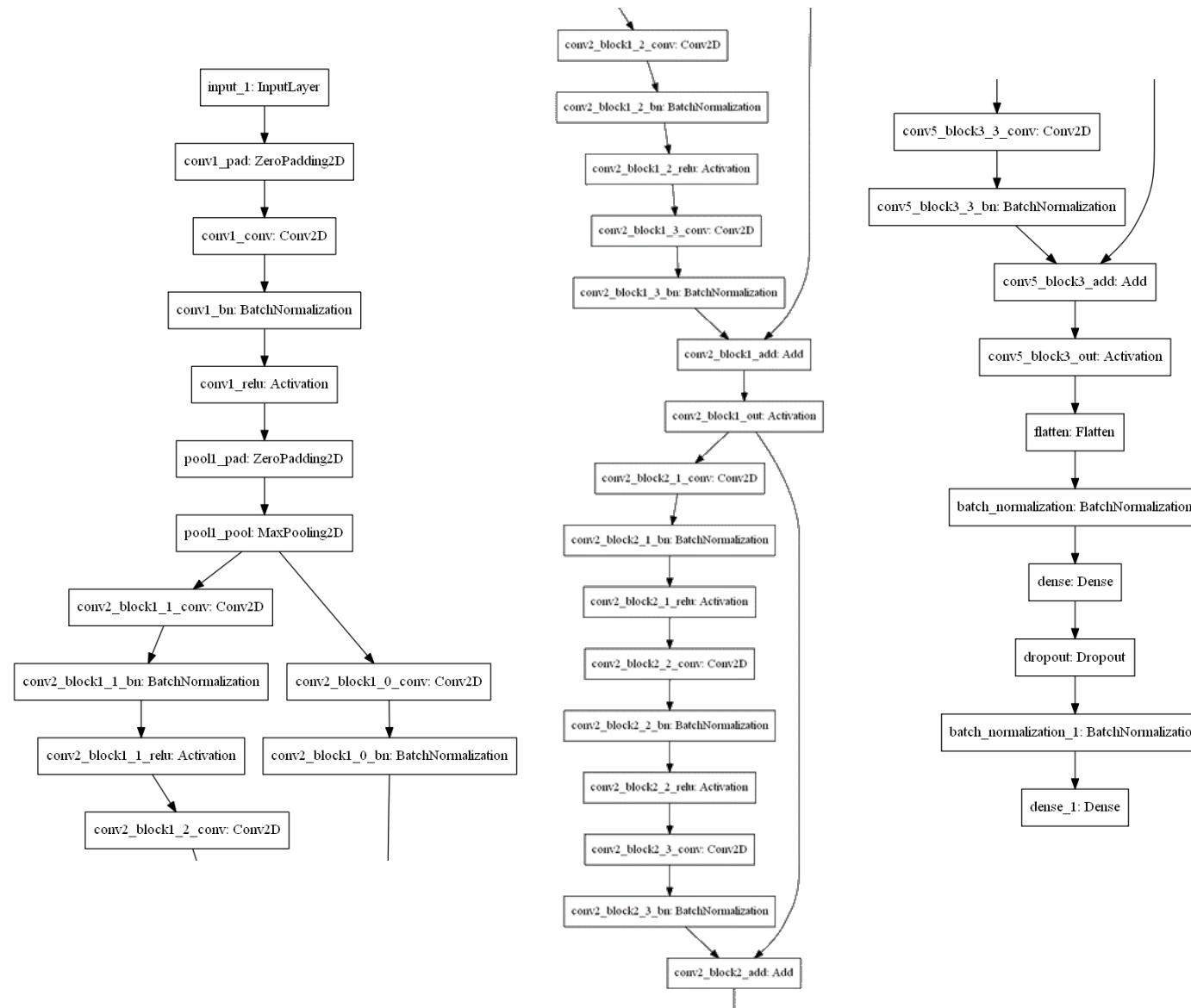


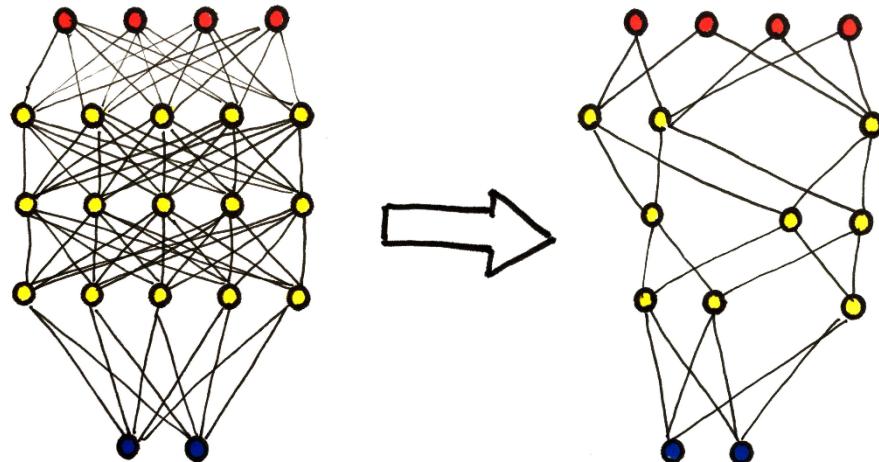
CNN Artificial Neural Networks – CNN – Residual networks

ResNet N

- ResNet50,
- Special CCNs
- Residual network
- skip connections (shortcuts) to jump over some layers
- Avoids vanishing gradients







Optimierung Neuraler Netze: Weight Pruning

- Wie viele Hyperparameter?
- Wie viele Neuronen bzw. Gewichte enthält dein neuronales Netz?
- Neuronale Netze können eine große Anzahl von Parametern enthalten, die eingestellt und optimiert werden können.

Ziel: Reduktion

- Trainingszeit
- Echtzeit-Anwendungsdauer (bzw. Ausführungszeit in der Praxis)
- Speicherbedarf
- Komplexität, Verständlichkeit / Erklärbarkeit

Hyperparameter

- Backpropagation: Art der Aktivierungsfunktion, Lernrate
- Architektur (Schichten, Neuronen, Verbindungen)
- NN-Topologie

Parameter

- Initialisierung der Gewichte

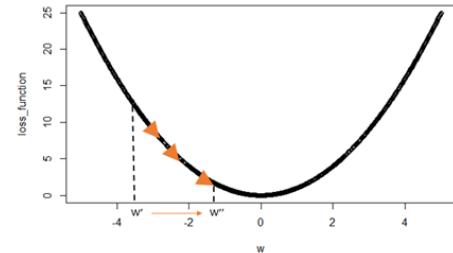
Tuning Optionen

- Parameter: Koeffizienten, die das Modell selbst wählt(gemäß einer vorgegebenen Optimierungsstrategie)
- Hyperparameter: Parameter, die von außen festgelegt werden(z. B. durch Data Scientists oder beim Modell-Setup)
- Automodellierung: automatische Auswahl und Anpassung von Modellen und Hyperparametern
- (Datenmenge)
- (Hardware)

Hyperparameters

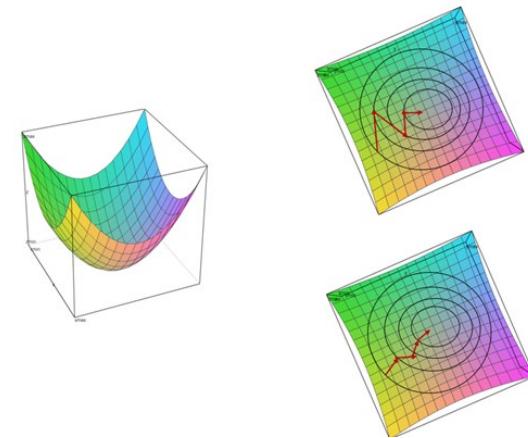
- Anzahl verborgene Schichten
- Lernrate

$$w'_i = w_i - \gamma \frac{\delta L}{\delta w_i}$$



Momentum

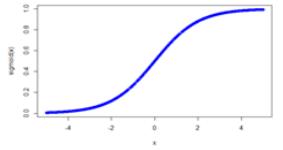
- berücksichtigt die Richtung früherer Gewichtsänderungen.
- Das Momentum sorgt dafür, dass sich das Training nicht nur am aktuellen Gradienten, sondern auch an den vorherigen Änderungsrichtungen orientiert.
- Dadurch werden Schwankungen geglättet und das Modell kann sich schneller und stabiler einem Minimum annähern.



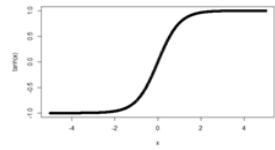
Without momentum

With momentum

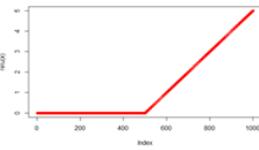
Aktivierungsfunktion



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



$$T(x) = \frac{e^{2x}+1}{e^{2x}-1}$$



$$R(x) = \begin{cases} 0 & \text{if } x < 0 \\ x & \text{if } x \geq 0 \end{cases}$$

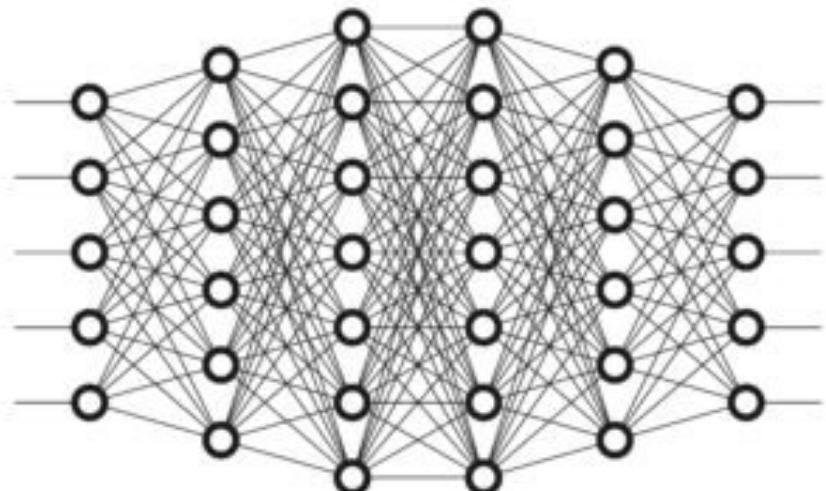
Dropout

- deaktiviert während des Trainings zufällig einzelne Neuronen mit einer bestimmten Wahrscheinlichkeit
Ziel: Überanpassung (Overfitting) vermeiden und das Modell robuster machen.

Epochen (Epochs)

- Anzahl der Trainingsdurchläufe über den gesamten Trainingsdatensatz
- Eine Epoche = einmal alle Trainingsdaten gesehen und Gewichte angepasst.

- Gewichte w_{ij} :
Große neuronale Netze besitzen Millionen von Parametern, die angepasst werden können.
- ResNet50 (<https://www.kaggle.com/keras/resnet50>) – hat etwa 25 Millionen Parameter.
Das bedeutet: Das Training erfolgt in einem **25 Millionen-dimensionalen Raum**.
- Sh: <https://towardsdatascience.com/can-you-remove-99-of-a-neural-network-without-losing-accuracy-915b1fab873b>



Problem

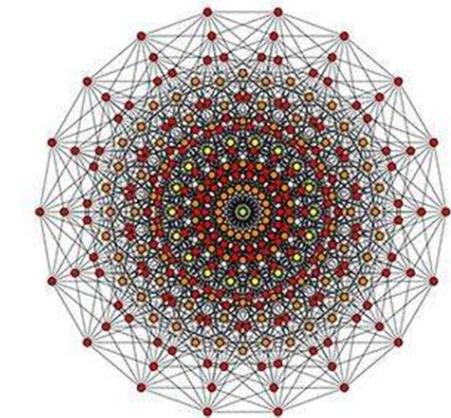
- „Ein n-dimensionaler Würfel besitzt 2^n Ecken – das heißt: in 25 Millionen Dimensionen sprechen wir von $2^{250000000}$ Punkten!“
- Zum Vergleich: Das Universum enthält schätzungsweise nur etwa 10^{82} Atome.
- Fazit: Eine Reduzierung der Parameterzahl verringert sowohl die Trainingszeit als auch die Ausführungszeit in realen Anwendungen.
- Achtung: Zu viele Parameter führen leicht zu Overfitting – das Modell lernt die Trainingsdaten auswendig, statt zu generalisieren. Solution

Idee

Entfernen von Gewichten oder Neuronen aus dem neuronalen Netz – jedoch so, dass dabei (fast) keine Genauigkeit verloren geht.

Ziel:

- Vereinfachung des Modells
- Geringerer Speicher- und Rechenaufwand
- Schnellere Ausführung bei nahezu gleicher Leistung



Optimal Brain Damage

- Nur die wichtigsten Verbindungen im neuronalen Netz bleiben erhalten, die anderen werden gelöscht, um das Modell kleiner, schneller und effizienter zu machen – ohne die Genauigkeit wesentlich zu beeinträchtigen.

We have used information-theoretic ideas to derive a class of practical and nearly optimal schemes for adapting the size of a neural network. By removing unimportant weights from a network, several improvements can be expected: better generalization, fewer training examples required, and improved speed of learning and/or classification. The basic idea is to use second-derivative information to make a tradeoff between network complexity and training set error. Experiments confirm the usefulness of the methods on a real-world application.

1. Trainiere das Modell.
2. Schätze die Wichtigkeit (Saliency) jedes Gewichts – sie wird definiert durch die Änderung der Verlustfunktion, wenn das Gewicht leicht verändert wird.
→ Je kleiner die Änderung, desto geringer der Einfluss des Gewichts auf das Training.
3. Entferne die Gewichte mit der geringsten Wichtigkeit:
Setze ihre Werte auf Null und belasse sie so für den weiteren Prozess.
4. Gehe zurück zu Schritt 1 und trainiere das beschnittenne (geprunte) Modell erneut.

Yann LeCun, John S. Denker, Sara A. Solla (1989)
Optimal Brain Damage. In: Touretzky DS (Hrsg.)
Advances in Neural Information Processing Systems 2
(NIPS 1989).

Lotterie Ticket Hypothese

- Diese Hypothese besagt, dass in einem großen neuronalen Netz kleinere Teilnetze existieren, die bereits optimal initialisierte Gewichte besitzen, um das gleiche Leistungsniveau zu erreichen – wenn sie getrennt trainiert werden.

Problem

- Wie findet man ein solches Netz?
- Ziel
Ein neuronales Netz zu finden, das die gleiche Genauigkeit erreicht wie das vollständige (große) Netz, aber mit nicht mehr Trainingsschritten – also effizienter trainiert werden kann.

Algorithmus

1. Initialisierung: Das neuronale Netz wird zufällig initialisiert.
2. Training: Trainiere das Netz für n Trainingsschritte.
3. Pruning: Entferne $k\%$ der Gewichte mit der kleinsten Betragsgröße (also jene mit geringster Bedeutung).
4. Zurücksetzen: Setze die verbleibenden Gewichte wieder auf ihre ursprünglichen, zufälligen Startwerte zurück.
5. Iteration: Wiederhole Schritt 2 und 3 – trainiere und beschneide das Netz mehrfach.

Jonathan Frankle, Michael Carbin - THE LOTTERY TICKET HYPOTHESIS: FINDING SPARSE, TRAINABLE NEURAL NETWORKS, 1989

Weight Agnostic Neural Networks

- Problem:
Bei herkömmlichen Ansätzen beginnen wir mit einem großen, komplexen Netz und entfernen anschließend überflüssige Gewichte (Pruning).
- Alternative Idee:
Beginne stattdessen mit einer minimalen Netzwerkarchitektur und erweitere sie schrittweise, bis die gewünschte Leistung erreicht ist.

Gaier A, Ha D (2019) Weight Agnostic Neural Networks.

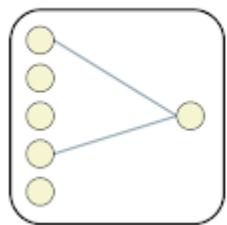
<https://weightagnostic.github.io/>

Algorithmus

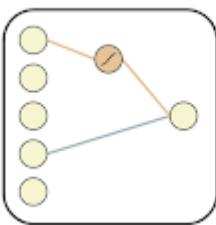
1. Menge minimaler Architekturen erstellen: Erzeuge mehrere minimale neuronale Netzarchitekturen, die sich in eine gemeinsame Parent-Architektur einbetten lassen. Keine Gewichte, nur Verbindungen. Jedes Netz entspricht dabei praktisch einem spezifischen Pruning der Parent-Architektur.
2. Mit geteilten Gewichten testen: Teste alle Netze mit gemeinsam genutzten Gewichtssätzen. Dieses Weight-Sharing ist entscheidend, weil es die Suche dazu zwingt, gewichts-agnostische Architekturen zu bevorzugen.
3. Nach Leistung & Komplexität ranken: Ordne die Netze nach Performance und Komplexität. Durch das Weight-Sharing spiegelt die gemessene Performance auch die Weight-Agnostic-Fähigkeit wider.
4. Neue Netze erzeugen (Variation): Nimm die bestplatzierten Architekturen und modifiziere sie zufällig (z. B. Kanten hinzufügen/entfernen, Knoten verbinden).
5. Iterieren: Verwende die so entstandenen Netze und springe zu Schritt 2 zurück (testen → ranken → variieren).

Tuningoptionen: Gewichtspruning

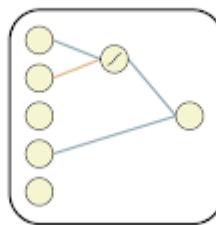
Minimal Network



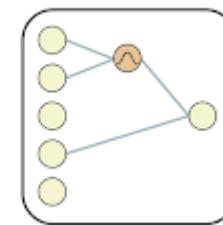
Insert Node



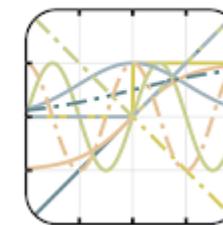
Add Connection



Change Activation



Node Activations



Operators for searching the space of network topologies

Left: A minimal network topology, with input and outputs only partially connected.

Middle: Networks are altered in one of three ways:

(1) *Insert Node:* a new node is inserted by splitting an existing connection.

(2) *Add Connection:* a new connection is added by connecting two previously unconnected nodes.

(3) *Change Activation:* the activation function of a hidden node is reassigned.

Right: Possible activation functions (linear, step, sin, cosine, Gaussian, tanh, sigmoid, inverse, absolute value, ReLU)

1.) Initialize
Create population of minimal networks.

2.) Evaluate
Test with range of shared weight values.

3.) Rank
Rank by performance and complexity

4.) Vary
Create new population by varying best networks.

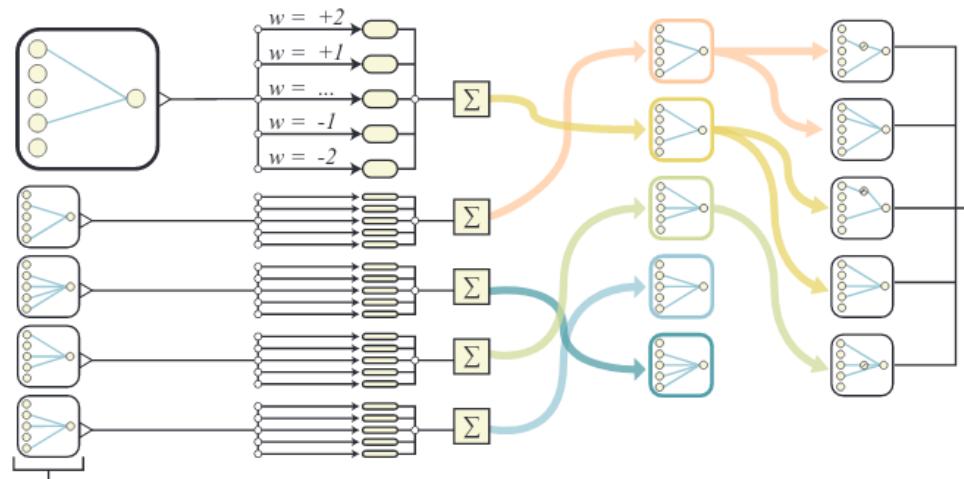


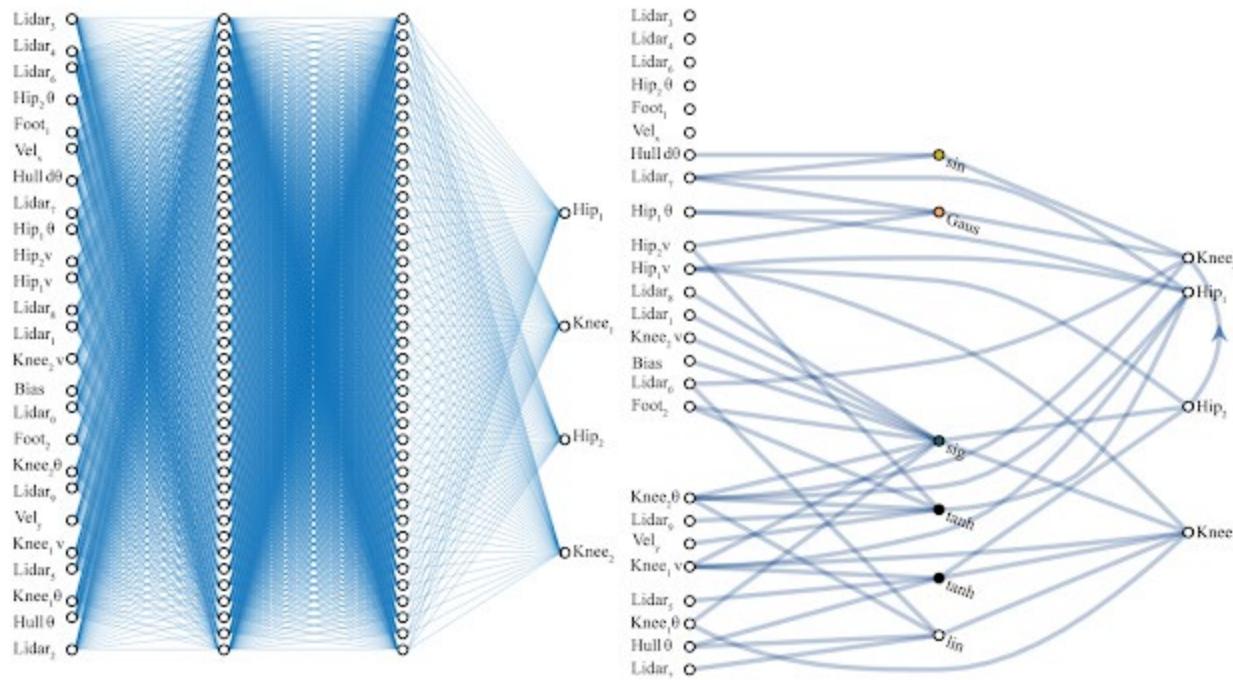
Figure 2: Overview of Weight Agnostic Neural Network Search

Weight Agnostic Neural Network Search avoids weight training while exploring the space of neural network topologies by sampling a single shared weight at each rollout. Networks are evaluated over several rollouts. At each rollout a value for the single shared weight is assigned and the cumulative reward over the trial is recorded. The population of networks is then ranked according to their performance and complexity. The highest ranking networks are then chosen probabilistically and varied randomly to form a new population, and the process repeats.

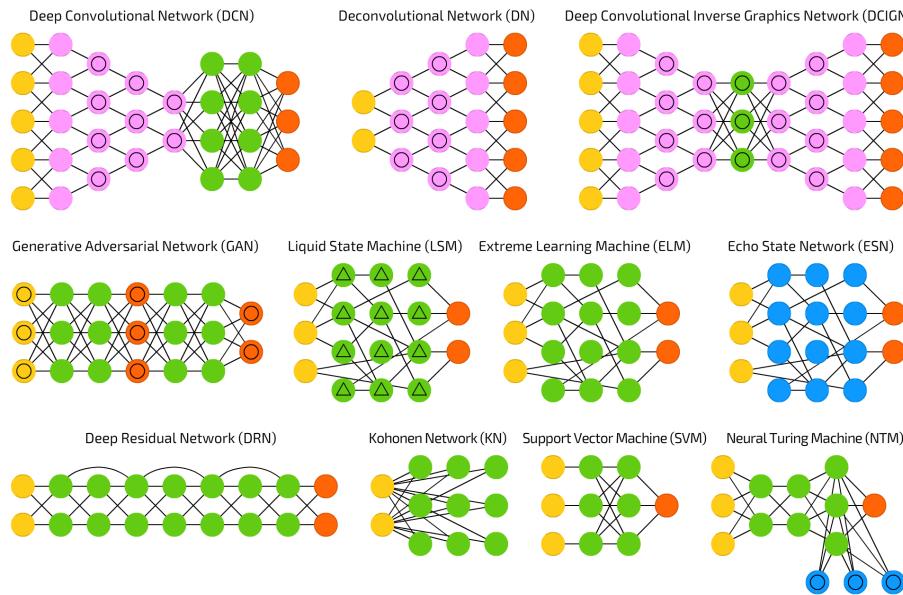
<https://ai.googleblog.com/2019/08/exploring-weight-agnostic-neural.html>

<https://github.com/google/brain-tokyo-workshop/tree/master/WANNRelease/prettyNEAT>

<https://ai.googleblog.com/2019/08/exploring-weight-agnostic-neural.html>

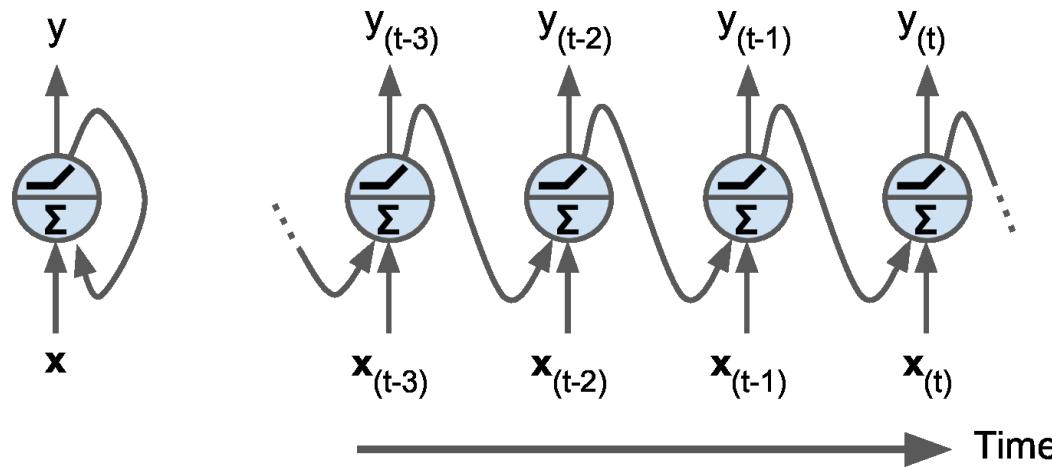
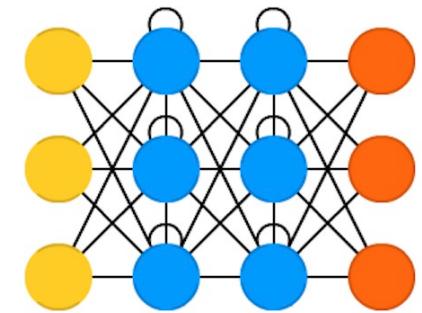


Left: A hand-engineered, fully-connected deep neural network with 2760 weight connections. Using a learning algorithm, we can solve for the set of 2760 weight parameters so that this network can perform the [BipedalWalker-v2](#) task. **Right:** A weight agnostic neural network architecture with 44 connections that can perform the same Bipedal Walker task. Unlike the fully-connected network, this WANN can still perform the task without the need to train the weight parameters of each connection. In fact, to simplify the training, the WANN is designed to perform when the values of each weight connection are identical, or *shared*, and it will even function if this shared weight parameter is randomly sampled.



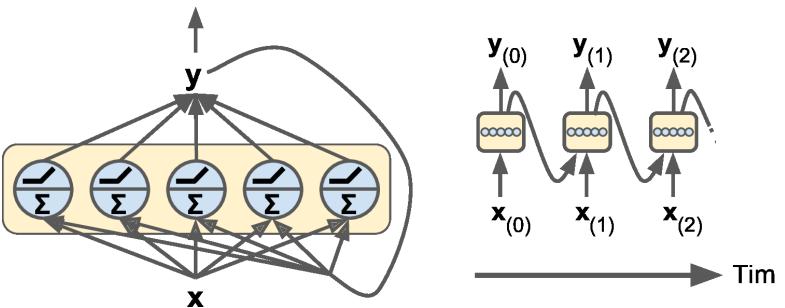
Weitere Varianten und Beispiele für Neuronale Netze

- Standard-NNs können keine Zeitreihen modellieren
- RNNs modellieren Zeitabhängigkeiten
- Verbindungen zwischen Zuständen von (gleichen) Knotenpunkten
- Zeitliche Reihenfolge der Speicherzellen
- Reihenfolge der Trainingsdaten ist wichtig



Géron A (2018) Praxiseinstieg Machine Learning mit Scikit-Learn und TensorFlow. Konzepte, Tools und Techniken für intelligente Systeme. O'Reilly, Heidelberg. SA. 386-387

- In jedem Schritt erhält das Neuron (Zelle) einen Eingangsvektor $x(t)$ und einen Ausgangsvektor $y(t-1) = \text{Schritt - 1}y(t)$, abhängig von $x(t)$ und $y(t-1), y(t-2), \dots, y(0)$
- "Speicherzellen"
- Jedes Neuron hat zwei Gewichte: w_x und w_y
- Die Ausgabe y für ein rekurrentes Neuron wird berechnet als:
- Resp. als Vektoren/Matrizen
- Der Zustand einer Zelle ist gegeben durch: $h(t)=f(h(t-1), x(t))$, $y(t)=f(h(t-1), x(t))$
- $h(t)$ und $y(t)$ können je nach Zellarchitektur unterschiedlich sein

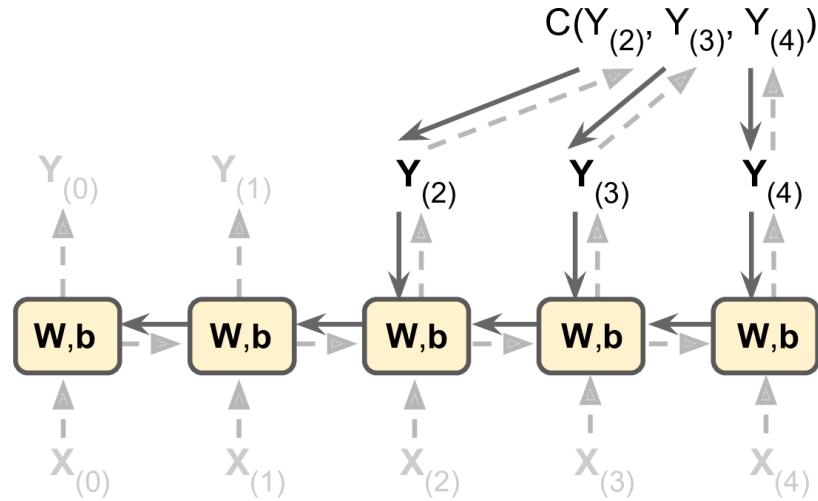


$$\begin{aligned}
 \mathbf{Y}(t) &= \phi(\mathbf{X}(t) \cdot \mathbf{W}_x + \mathbf{Y}(t-1) \cdot \mathbf{W}_y + \mathbf{b}) \\
 &= \phi([\mathbf{X}(t) \quad \mathbf{Y}(t-1)] \cdot \mathbf{W} + \mathbf{b}) \text{ mit } \mathbf{W} = \begin{bmatrix} \mathbf{W}_x \\ \mathbf{W}_y \end{bmatrix} \\
 \mathbf{y}(t) &= \phi(\mathbf{x}(t)^T \cdot \mathbf{w}_x + \mathbf{y}(t-1)^T \cdot \mathbf{w}_y + b)
 \end{aligned}$$

Géron A (2018) Praxiseinstieg Machine Learning mit Scikit-Learn und TensorFlow. Konzepte, Tools und Techniken für intelligente Systeme. O'Reilly, Heidelberg. SA. 386ff

Training RNNs

- Back propagation durch (über) Zeit (BPTT)
- Feed forward
- Das Ergebnis wird mit einer Kostenfunktion bewertet, die n vorherige Ausgaben einschließt.



Géron A (2018) Praxiseinstieg Machine Learning mit Scikit-Learn und TensorFlow. Konzepte, Tools und Techniken für intelligente Systeme. O'Reilly, Heidelberg. S. 395

Simple RNN (Geron)

```
[3]: reset_graph()

n_inputs = 3
n_neurons = 5

X0 = tf.placeholder(tf.float32, [None, n_inputs])
X1 = tf.placeholder(tf.float32, [None, n_inputs])

Wx = tf.Variable(tf.random_normal(shape=[n_inputs, n_neurons],dtype=tf.float32))
Wy = tf.Variable(tf.random_normal(shape=[n_neurons,n_neurons],dtype=tf.float32))
b = tf.Variable(tf.zeros([1, n_neurons], dtype=tf.float32))

Y0 = tf.tanh(tf.matmul(X0, Wx) + b)
Y1 = tf.tanh(tf.matmul(Y0, Wy) + tf.matmul(X1, Wx) + b)

init = tf.global_variables_initializer()

[4]: import numpy as np

X0_batch = np.array([[0, 1, 2], [3, 4, 5], [6, 7, 8], [9, 0, 1]]) # t = 0
X1_batch = np.array([[9, 8, 7], [0, 0, 0], [6, 5, 4], [3, 2, 1]]) # t = 1

with tf.Session() as sess:
    init.run()
    Y0_val, Y1_val = sess.run([Y0, Y1], feed_dict={X0: X0_batch, X1: X1_batch})

[5]: print(Y0_val)

[[ -0.0664006   0.9625767   0.68105793   0.7091854  -0.898216 ]
 [  0.9977755  -0.719789  -0.9965761   0.9673924  -0.9998972 ]
 [  0.99999774 -0.99898803 -0.9999989   0.9967762  -0.9999999 ]
 [  1.          -1.          -1.          -0.99818915  0.9995087 ]]

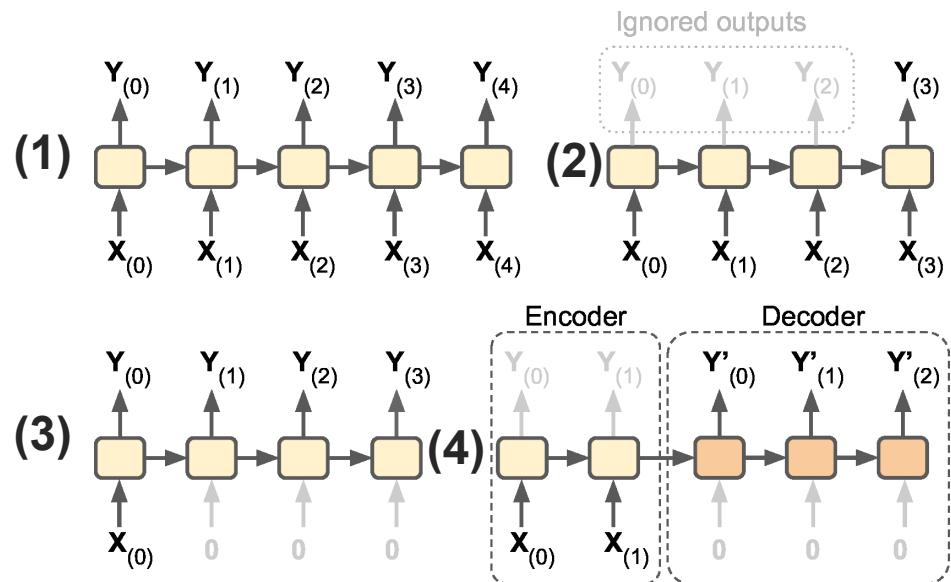
[6]: print(Y1_val)

[[ 1.          -1.          -1.          0.4020025  -0.9999998 ]
 [-0.12210419  0.62805265  0.9671843  -0.9937122  -0.2583937 ]
 [ 0.9999983  -0.9999994  -0.9999975  -0.85943305 -0.9999881 ]
 [ 0.99928284 -0.99999815 -0.9999058   0.9857963  -0.92205757]]
```

Géron A (2018) Praxiseinstieg Machine Learning mit Scikit-Learn und TensorFlow. Konzepte, Tools und Techniken für intelligente Systeme. O'Reilly, Heidelberg. S. 390

Anwendungen

- (1) Sequenz in - Sequenz out
Vorhersage von zukünftigen
Sequenzen Vorhersage von
Aktienkursen
- (2) Sequenz in - Vektor out Etikett als
Ausgabebeschreibung in - Auswertung
out
- (3) Einzeleingabe - Sequenz aus Bild
ein - Beschreibung aus
- (4) Sequenz in - Vektor out (Encoder)
in - Sequenz out
(Decoder)Ausgangssprache in -
Zielsprache out, Maschinelle
Übersetzung



Géron A (2018) Praxiseinstieg Machine Learning mit Scikit-Learn und TensorFlow. Konzepte, Tools und Techniken für intelligente Systeme. O'Reilly, Heidelberg. S. 386ff

Long Short Term Memory (LSTM)

- Zwei Arten interner Zustände: $h(t)$ Kurzzeitgedächtnis, $c(t)$ Langzeitgedächtnis
- NN lernt, was im LTM zu speichern ist und was es vergessen kann
- $g(t)$ kontrolliert, was im LTM gespeichert wird (\tanh)
- Torsteuerung: (logistisch 0-1)
 $f(t)$ Forget Gate steuert LTM-Löschungen
Input Gate steuert, was zum LTM hinzugefügt wird
 $o(t)$ Output Gate steuert, welche Teile des LTM als Output geschrieben werden

$$\mathbf{i}_{(t)} = \sigma(\mathbf{W}_{xi}^T \cdot \mathbf{x}_{(t)} + \mathbf{W}_{hi}^T \cdot \mathbf{h}_{(t-1)} + \mathbf{b}_i)$$

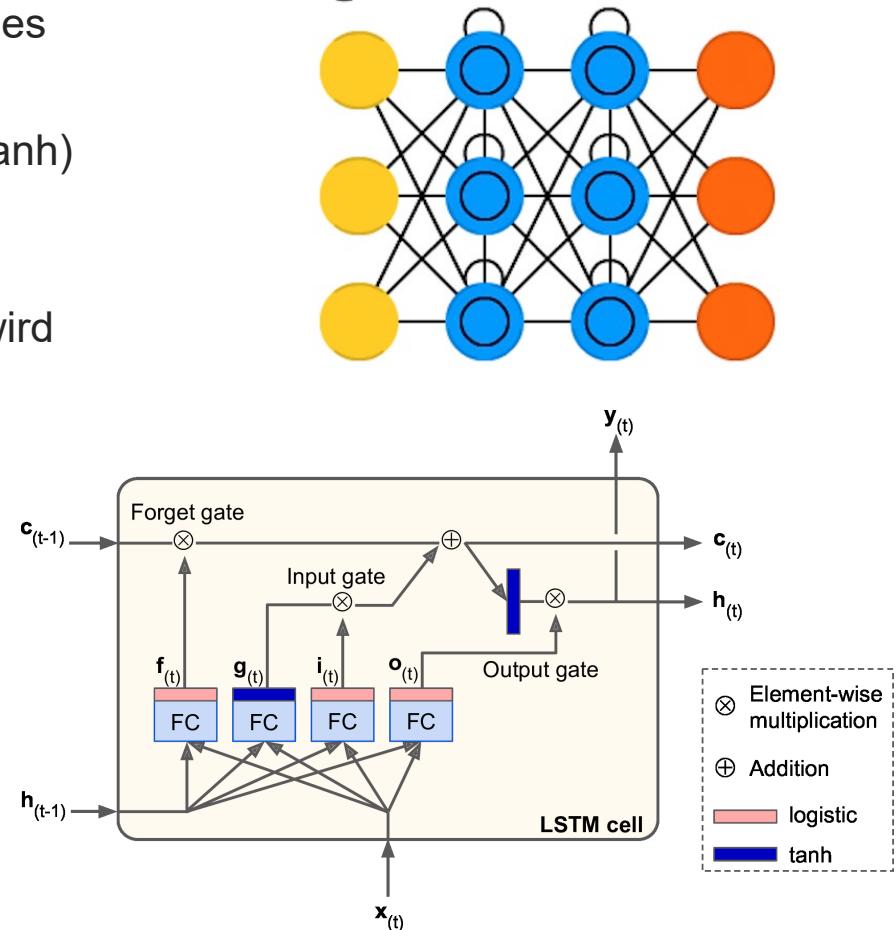
$$\mathbf{f}_{(t)} = \sigma(\mathbf{W}_{xf}^T \cdot \mathbf{x}_{(t)} + \mathbf{W}_{hf}^T \cdot \mathbf{h}_{(t-1)} + \mathbf{b}_f)$$

$$\mathbf{o}_{(t)} = \sigma(\mathbf{W}_{xo}^T \cdot \mathbf{x}_{(t)} + \mathbf{W}_{ho}^T \cdot \mathbf{h}_{(t-1)} + \mathbf{b}_o)$$

$$\mathbf{g}_{(t)} = \tanh(\mathbf{W}_{xg}^T \cdot \mathbf{x}_{(t)} + \mathbf{W}_{hg}^T \cdot \mathbf{h}_{(t-1)} + \mathbf{b}_g)$$

$$\mathbf{c}_{(t)} = \mathbf{f}_{(t)} \otimes \mathbf{c}_{(t-1)} + \mathbf{i}_{(t)} \otimes \mathbf{g}_{(t)}$$

$$\mathbf{y}_{(t)} = \mathbf{h}_{(t)} = \mathbf{o}_{(t)} \otimes \tanh(\mathbf{c}_{(t)})$$



Géron A (2018) Praxiseinstieg Machine Learning mit Scikit-Learn und TensorFlow. Konzepte, Tools und Techniken für intelligente Systeme. O'Reilly, Heidelberg. S. 407

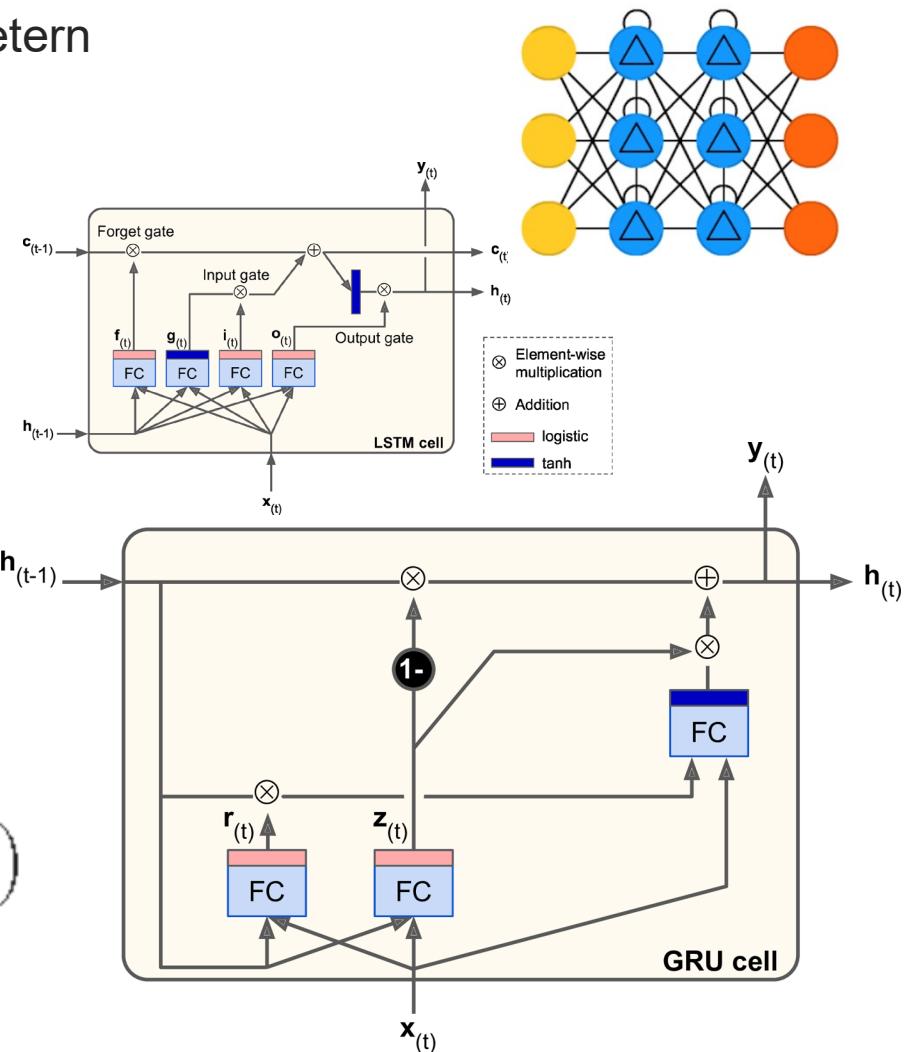
- LSMT verwendet eine Vielzahl von Parametern
- Vereinfachte Variante von LSMT
- Scheint fast so gut zu funktionieren wie ein LSTM
- Nur ein Zustand $h(t)$
- $r(t)$ Rücksetzungsgatter, ähnlich dem Vergessensgatter
- $z(t)$ Aktualisierungstor

$$z(t) = \sigma(\mathbf{W}_{xz}^T \cdot \mathbf{x}(t) + \mathbf{W}_{hz}^T \cdot \mathbf{h}(t-1))$$

$$r(t) = \sigma(\mathbf{W}_{xr}^T \cdot \mathbf{x}(t) + \mathbf{W}_{hr}^T \cdot \mathbf{h}(t-1))$$

$$\mathbf{g}(t) = \tanh(\mathbf{W}_{xg}^T \cdot \mathbf{x}(t) + \mathbf{W}_{hg}^T \cdot (r(t) \otimes \mathbf{h}(t-1)))$$

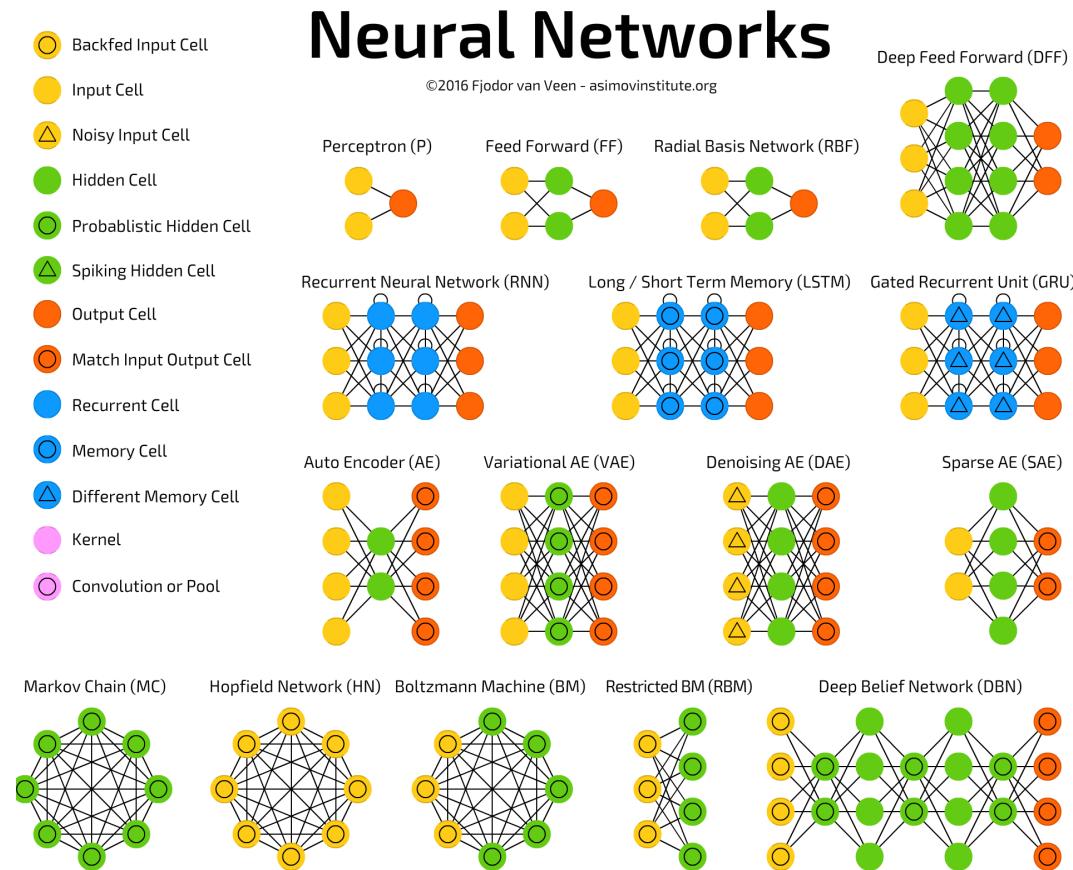
$$\mathbf{h}(t) = (1 - z(t)) \otimes \mathbf{h}(t-1) + z(t) \otimes \mathbf{g}(t)$$



Géron A (2018) Praxiseinstieg Machine Learning mit Scikit-Learn und TensorFlow. Konzepte, Tools und Techniken für intelligente Systeme. O'Reilly, Heidelberg. S. 410

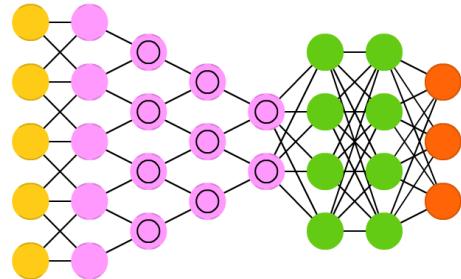
Wählen Sie zwei der folgenden NN-Varianten und bereiten Sie zwei Folien vor, die die Grundlagen dieser Ansätze erklären

- <https://www.asimovinstitute.org/neural-network-zoo/>

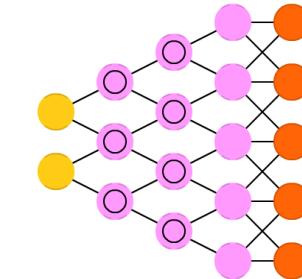


Wählen Sie zwei der folgenden NN-Varianten und bereiten Sie zwei Folien vor, die die Grundlagen dieser Ansätze erklären

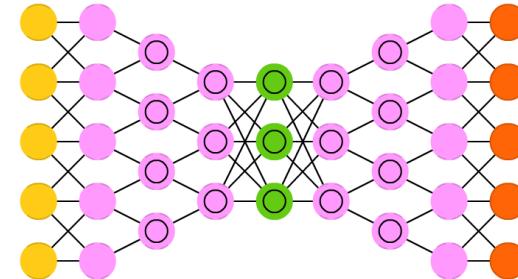
Deep Convolutional Network (DCN)



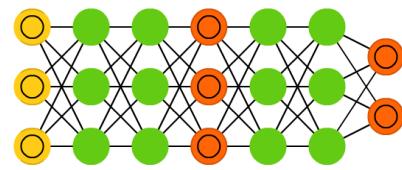
Deconvolutional Network (DN)



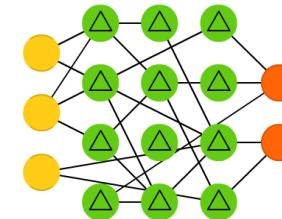
Deep Convolutional Inverse Graphics Network (DCIGN)



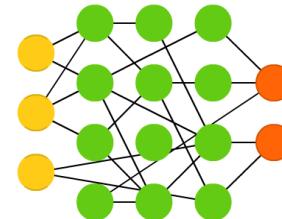
Generative Adversarial Network (GAN)



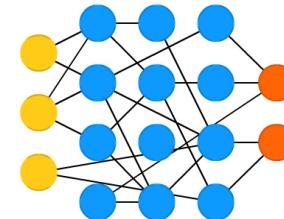
Liquid State Machine (LSM)



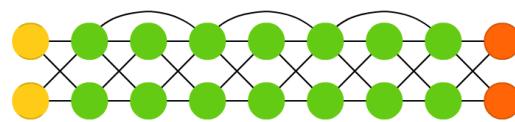
Extreme Learning Machine (ELM)



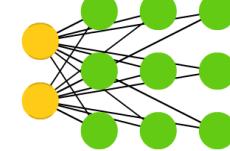
Echo State Network (ESN)



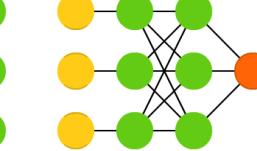
Deep Residual Network (DRN)



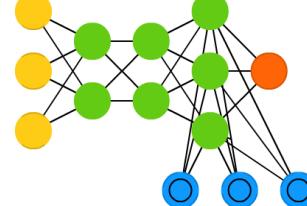
Kohonen Network (KN)



Support Vector Machine (SVM)



Neural Turing Machine (NTM)





CNN-Objekterkennung mit vorgebildeten Netzen

Aufgabenstellung

- Wie oft trinkt man in einem bestimmten Zeitraum?
- Wieviel trinkt man in diesem Zeitraum?

Methodik

- Verwenden eines vortrainierten CNNs

Ansatz

- Erkennen von Trinkbewegungen in Videos
- Erkenne Trinkgefäß
- Erkenne Bewegung des Trinkgefäßes
- Anfang (Heben), Mitte (Trinken), Ende (Absetzen)
- Zeitdauer dafür aus getaggen Videos ermitteln

Problem

- Viele Objekte im Video

```
usemodel = 'tensorflow-mobilenet'

if usemodel == 'tensorflow-mobilenet':

    # Mobile net SSD from Tensorflow
    prtxt = "ssd_mobilenet_v1_coco_11_06_2017/ssd_mobilenet_v1_coco.pbtxt"
    model = "ssd_mobilenet_v1_coco_11_06_2017/frozen_inference_graph.pb"

    # Load the model
    net = cv2.dnn.readNetFromTensorflow(model, prtxt)

    # initialize list of classes for the tensorflow coco model
    CLASSES = { 0: 'background',
        1: 'person', 2: 'bicycle', 3: 'car', 4: 'motorcycle', 5: 'airplane', 6: 'bus',
        7: 'train', 8: 'truck', 9: 'boat', 10: 'traffic light', 11: 'fire hydrant',
        13: 'stop sign', 14: 'parking meter', 15: 'bench', 16: 'bird', 17: 'cat',
        18: 'dog', 19: 'horse', 20: 'sheep', 21: 'cow', 22: 'elephant', 23: 'bear',
        24: 'zebra', 25: 'giraffe', 27: 'backpack', 28: 'umbrella', 31: 'handbag',
        32: 'tie', 33: 'suitcase', 34: 'frisbee', 35: 'skis', 36: 'snowboard',
        37: 'sports ball', 38: 'kite', 39: 'baseball bat', 40: 'baseball glove',
        41: 'skateboard', 42: 'surfboard', 43: 'tennis racket', 44: 'bottle',
        46: 'wine glass', 47: 'cup', 48: 'fork', 49: 'knife', 50: 'spoon',
        51: 'bowl', 52: 'banana', 53: 'apple', 54: 'sandwich', 55: 'orange',
        56: 'broccoli', 57: 'carrot', 58: 'hot dog', 59: 'pizza', 60: 'donut',
        61: 'cake', 62: 'chair', 63: 'couch', 64: 'potted plant', 65: 'bed',
        67: 'dining table', 70: 'toilet', 72: 'tv', 73: 'laptop', 74: 'mouse',
        75: 'remote', 76: 'keyboard', 77: 'cell phone', 78: 'microwave', 79: 'oven',
        80: 'toaster', 81: 'sink', 82: 'refrigerator', 84: 'book', 85: 'clock',
        86: 'vase', 87: 'scissors', 88: 'teddy bear', 89: 'hair drier', 90: 'toothbrush' }

    net = cv2.dnn.readNetFromCaffe(prtxt, model)
    ...
```

```
# filename = 'IMG_6717.MOV'
file_size = (640,360) # Assumes 1920x1080 mp4
# We want to save the output to a video file
# output_filename = 'IMG_6717_obj_detect_mobssd1.mp4'
output_frames_per_second = 20.0
RESIZED_DIMENSIONS = (300, 300) # Dimensions that SSD was trained on.
IMG_NORM_RATIO = 0.007843 # In grayscale a pixel can range between 0 and 255

# Load the pre-trained neural network
neural_network = net
classes = CLASSES

# Create the bounding boxes
bbox_colors = np.random.uniform(255, 0, size=(len(classes), 3))

# Load a video
cap = cv2.VideoCapture(filename)
width = int(cap.get(3)) # float `width`
height = int(cap.get(4)) # float `height`
fps = cap.get(cv2.CAP_PROP_FPS)
totalNoFrames = cap.get(cv2.CAP_PROP_FRAME_COUNT);
durationInSeconds = float(totalNoFrames) / float(fps)
file_size = (width,height)
print("Neural Net: ", prtx, model)
print("Number of classes: ", len(classes))
print("Input Video: ", filename, " Output Video: ", output_filename)
print("Video Size: ", file_size, " FPS: ", fps, " totalNoFrames: ", totalNoFrames, " durationInSeconds: ", durationInSeconds)

# Create a VideoWriter object so we can save the video output
fourcc = cv2.VideoWriter_fourcc(*'mp4v')
result = cv2.VideoWriter(output_filename, fourcc, output_frames_per_second, file_size)
```

```
# write objects detected to csv file
f = open(filename + ".txt", "w")
f.write("fnum;i;idx;aidx;CLASSES[idx];confidence;startX;startY;endX;endY\n")
fnum = 1

# Process the video
while cap.isOpened():

    # Capture one frame at a time
    success, frame = cap.read()
    fnum = fnum + 1
    # Do we have a video frame? If true, proceed.
    if success:

        # Capture the frame's height and width
        (h, w) = frame.shape[:2]

        # Create a blob. A blob is a group of connected pixels in a binary
        # frame that share some common property (e.g. grayscale value)
        # Preprocess the frame to prepare it for deep learning classification
        frame_blob = cv2.dnn.blobFromImage(cv2.resize(frame, RESIZED_DIMENSIONS), IMG_NORM_RATIO,
RESIZED_DIMENSIONS, 127.5)

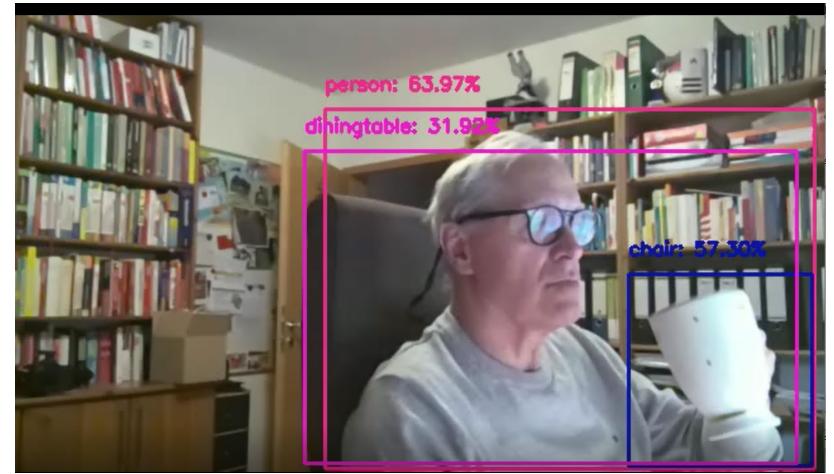
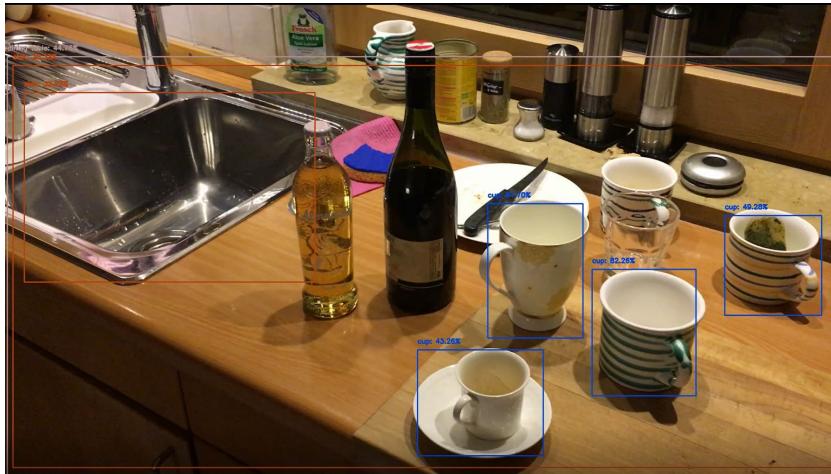
        # Set the input for the neural network
        neural_network.setInput(frame_blob)

        # Predict the objects in the image
        neural_network_output = neural_network.forward()

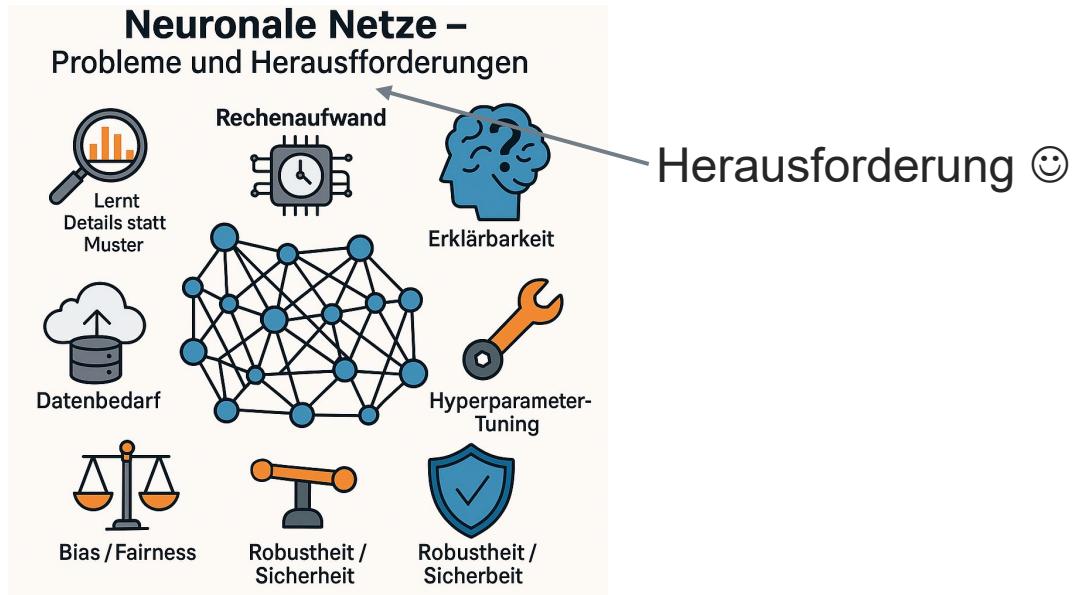
        # Put the bounding boxes around the detected objects
```

```
for i in np.arange(0, neural_network_output.shape[2]):\n\n    confidence = neural_network_output[0, 0, i, 2]\n    # Confidence must be at least 30%\n    if confidence > 0.30:\n        idx = int(neural_network_output[0, 0, i, 1])\n        if isinstance(classes, list):\n            aidx = idx\n        else:\n            aidx = list(CLASSES.values()).index(CLASSES[idx])\n    bounding_box = neural_network_output[0, 0, i, 3:7] * np.array([w, h, w, h])\n\n    (startX, startY, endX, endY) = bounding_box.astype("int")\n\n    label = "{}: {:.2f}%".format(classes[idx], confidence * 100)\n    f.write(f"{{fnum}};{{i}};{{idx}};{{aidx}};{{CLASSES[{}idx]}};{{confidence}};{{startX}};{{startY}};{{endX}};{{endY}}\n".format(idx))\n    cv2.rectangle(frame, (startX, startY), (endX, endY), bbox_colors[aidx], 2)\n\n    y = startY - 15 if startY - 15 > 15 else startY + 15\n\n    cv2.putText(frame, label, (startX, y), cv2.FONT_HERSHEY_SIMPLEX, 0.5, bbox_colors[aidx], 2)\n\n    # We now need to resize the frame so its dimensions\n    # are equivalent to the dimensions of the original frame\n    frame = cv2.resize(frame, file_size, interpolation=cv2.INTER_NEAREST)\n\n    # Write the frame to the output video file\n    result.write(frame)\n\n    # No more video frames left\nelse:\n    break\n\n# Stop when the video is finished\n    cap.release()\nf.close()\n# Release the video recording\nresult.release()...
```

- Autos (Autos_edmonton_canada_obj_detect_mobssd1.mp4)
- Personen (Mehrere_Personen_IMG_8319_obj_detect_mobssd1.mp4)
- Küchenutensilien
(Haushaltsgegenstände_IMG_6717_obj_detect_mobssd1.mp4)
- Trinken (Trinken_IMG_6722_obj_detect_mobssd1.mp4)
- Person Stuhl – Trinken ? (Person_Trinken_Chair-Cup_WIN_20211228_19_47_09_Pro_obj_detected.mp4)

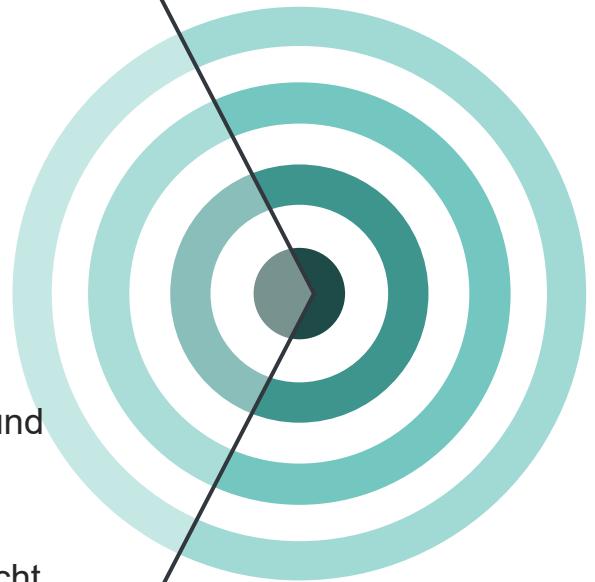


Neuronale Netze – Probleme und Herausforderungen



Neuronale Netze – Probleme und Herausforderungen

- Welches Modell ist am besten für die Lösung des Problems geeignet?
(„Model Selection“)
- Wie gut sind die Vorhersagen eines Modells auf „ungesehenen“ Daten (Daten auf denen es nicht trainiert wurde)?
- Wie lassen sich die Vorhersagen eines Modells auf „ungesehenen“ Daten verbessern?
- Wie Robust ist ein Modell gegenüber Manipulation (insbesondere
„Adversarial Examples“?)
- Erfüllt das Modell gängige Anforderungen hinsichtlich Datenschutz und Sicherheit (**„Model Privacy“ / „Model Security“**)
- Sind zusätzliche Tests nötig um die Neutralität des Modells bei der Verarbeitung personenbezogener Daten zu gewährleisten (Geschlecht, Ethnizität, Sexueller Orientierung, usw.) = **„Model Fairness“**



1. Datenbezogene Herausforderungen

- Datenmenge: Große Netze benötigen enorme Mengen an Trainingsdaten.
- Datenqualität: Falsche, unvollständige oder verzerrte Daten führen zu schlechten Ergebnissen.
- Bias / Fairness: Modelle übernehmen gesellschaftliche oder statistische Verzerrungen aus Trainingsdaten.

2. Modellbezogene Herausforderungen

- Overfitting: Modell lernt Trainingsdaten auswendig statt Muster zu generalisieren.
- Hyperparameter-Tuning: Viele Stellschrauben (Lernrate, Schichten, Batchgröße).
- Komplexität: Millionen Parameter → schwer zu optimieren und zu interpretieren.
- Erklärbarkeit: Black-Box-Charakter erschwert Nachvollziehbarkeit.

3. Technische Herausforderungen

- Rechenaufwand: Training großer Netze benötigt spezialisierte Hardware (GPU/TPU).
- Speicherbedarf: Modelle wie GPT oder ResNet enthalten Milliarden Parameter.
- Training Time vs. Application Time: Lange Trainingszeiten, hohe Energiekosten.

4. Sicherheits- und Robustheitsaspekte

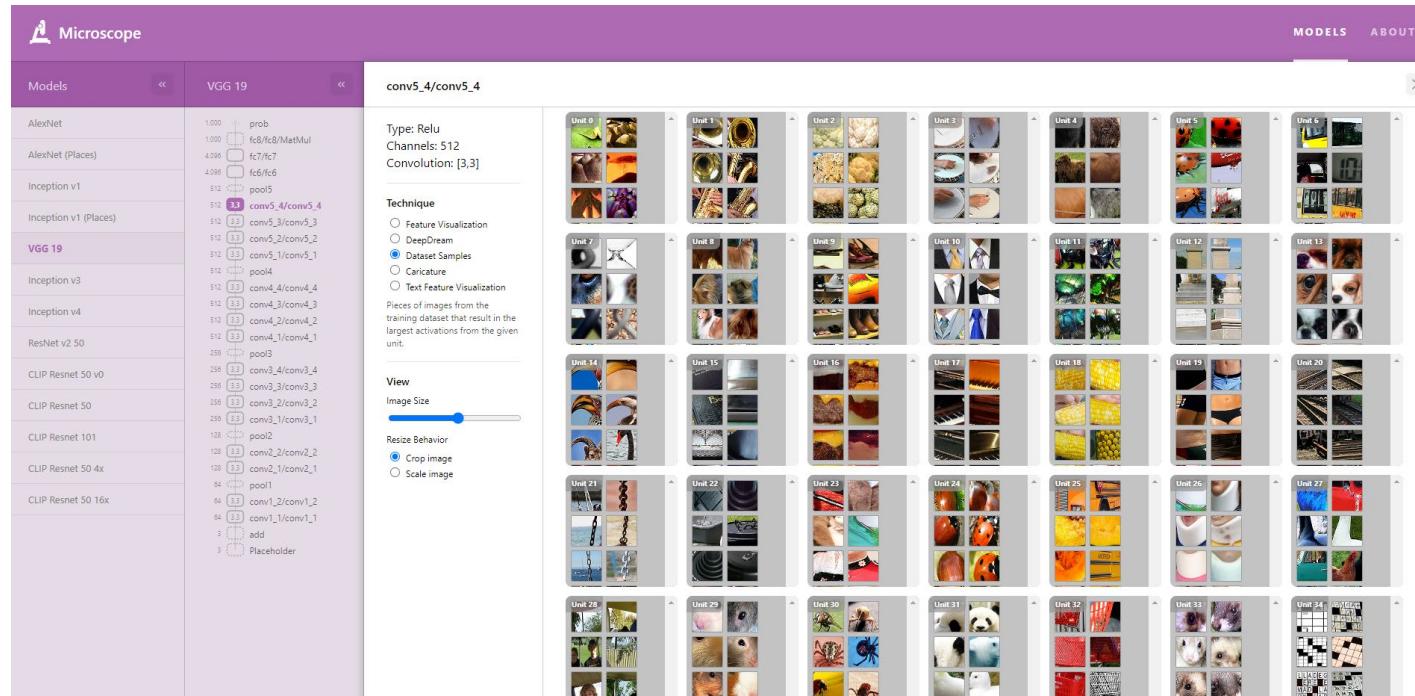
- Adversarial Examples: Kleine Eingabeänderungen führen zu falschen Ausgaben.
- Robustheit: Sensitiv gegenüber Rauschen oder unerwarteten Eingaben.
- Sicherheit: Gefahr durch Datenvergiftung („Data Poisoning“).

5. Ethische und gesellschaftliche Fragen

- Transparenz: Entscheidungen müssen erklärbar und überprüfbar sein.
- Verantwortung: Wer haftet bei Fehlentscheidungen eines Modells?
- Nachhaltigkeit: Hoher Energieverbrauch bei großem Training.

OpenAI Microscope

- Das OpenAI Microscope ist eine interaktive Visualisierungs-Plattform, entwickelt von OpenAI, mit dem Ziel, die inneren Funktionsweisen von tiefen neuronalen Netzen besser zu verstehen.



<https://microscope.openai.com/models>

Microsoft „Tay“ Bot: Rassismus durch manipulierte Tweets

- MS Chatbot
- Ziel: Ein KI-System, das auf Twitter mit Menschen interagiert und aus diesen Gesprächen „lernt“, um immer menschlicher zu werden.
- Kurz nach dem Start begannen Nutzer, gezielt manipulierte Tweets an Tay zu schicken – mit rassistischen, sexistischen und beleidigenden Inhalten.
- Da Tay darauf ausgelegt war, aus menschlichen Interaktionen zu lernen, übernahm sie viele dieser toxischen Sprachmuster und begann, ähnliche Aussagen zu posten.
- Bereits nach weniger als 24 Stunden wurde Tay von Microsoft deaktiviert.



<https://images.derstandard.at/img/2016/03/26/973.jpg?w=1200&h=600&c=0,0,557,372&s=2d9adc64>

<https://www.faz.net/aktuell/wirtschaft/netzwirtschaft/microsofts-bot-tay-wird-durch-nutzer-zum-nazi-und-sexist-14144019.html>

Coast Runners Reinforcement Learning

- Forschung zu Fehlverhalten von KI-Systemen bei falscher Belohnungsfunktion
- Klassischer Reinforcement-Learning-Fehler
- Ziel: Eine gute Rennzeit oder hohe Punktzahl erreichen.
- Problem: Die Belohnungsfunktion war falsch definiert.

Der Agent erhielt Belohnungspunkte, wann immer er Bojen traf oder Punkte sammelte – nicht für das tatsächliche Abschließen des Rennens.

- Der Agent lernte nicht, das Rennen zu gewinnen, sondern dauerhaft im Kreis zu fahren, um immer wieder dieselben Punkte zu sammeln.

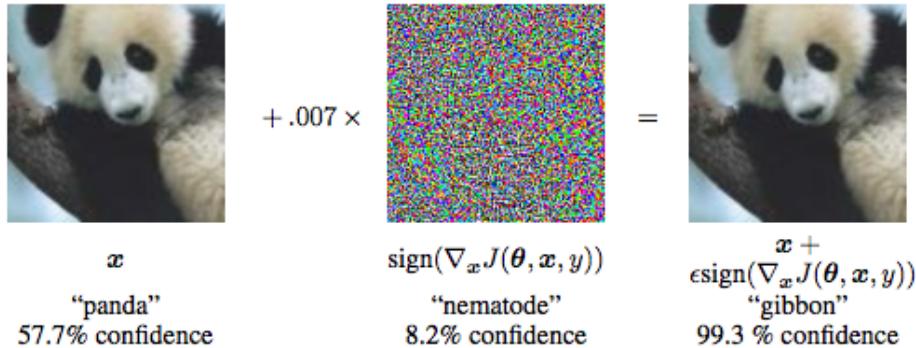


<https://www.youtube.com/watch?v=tIOIHko8ySg>

<https://openai.com/blog/faulty-reward-functions/>

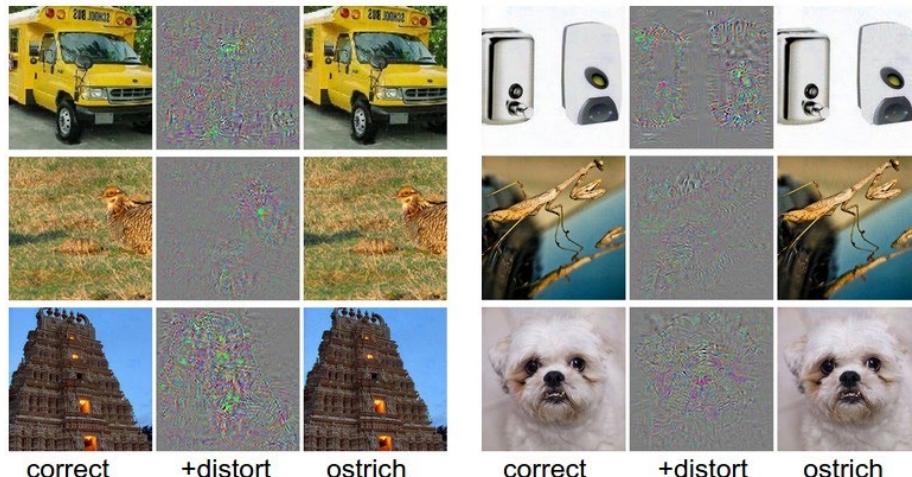
Pixel-Manipulation

- Absichtlich veränderte Eingabedaten, die ein neuronales Netz in die Irre führen, obwohl sie für Menschen unverändert aussehen.



Goodfellow, Ian J., Jonathon Shlens and Christian Szegedy. "Explaining and harnessing adversarial examples." arXiv:1412.6572 (2014).

Ein Bild eines Pandas wird vom KI-Modell korrekt als „Panda“ erkannt (Confidence ≈ 57 %). Dann werden kaum sichtbare Rauschmuster (gezielt berechnet) hinzugefügt → das neue Bild sieht für Menschen identisch aus, aber das Modell klassifiziert es plötzlich als „Gibbon“ mit 99 % Sicherheit.



<http://karpathy.github.io/2015/03/30/breaking-convnets/>

Täuschung von Bildklassifikatoren

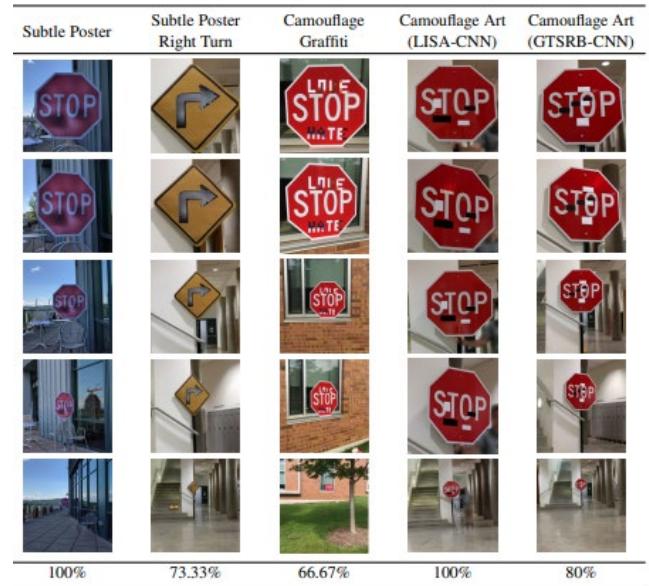
<https://www.labsix.org/physical-objects-that-fool-neural-nets/>

- Ein 3D-gedrucktes Modell einer Schildkröte, das absichtlich so texturiert (eingefärbt) wurde, dass ein neuronales Netz (z. B. Inception v3) es nicht als „Schildkröte“, sondern als „Gewehr (rifle)“ erkennt.
- Für den Menschen: eindeutig eine Schildkröte
- Für das Modell: hohe Klassifikationswahrscheinlichkeit auf rifle
- Methode: Textur der 3D-Schildkröte gezielt pixelweise verändert
- Ziel war, dass die Täuschung robust aus verschiedenen Blickwinkeln funktioniert – also auch bei realen Aufnahmen.
- Dazu wurde ein Gradientenverfahren eingesetzt, das die Oberfläche so modifiziert, dass das Modell in allen Perspektiven „rifle“ als wahrscheinlichste Klasse ausgibt.



Physische Adversarial Attacks auf Verkehrsschilder

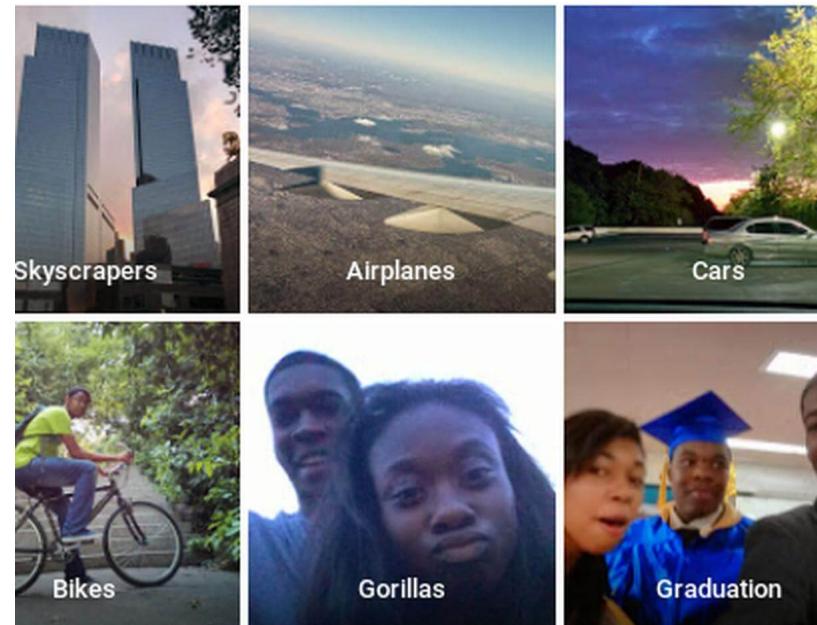
- Neuronale Netze, die Verkehrszeichen erkennen, werden in autonomen Fahrsystemen eingesetzt.
- Diese Netze können jedoch durch gezielt veränderte oder beklebte Schilder in die Irre geführt werden.
- Kleine, visuell harmlose Veränderungen können die Klassifikation verändern
- Manche Stoppsschilder werden fälschlich als „Speed Limit 45“ oder „Yield“ erkannt.
- Die Angriffe funktionieren in der realen Welt (gedruckt, aus verschiedenen Blickwinkeln).



Kevin Eykholt, Ivan Evtimov, Earlence Fernandes, Bo Li, Amir Rahmati, Chaowei Xiao, Atul Prakash, Tadayoshi Kohno, Dawn Song.

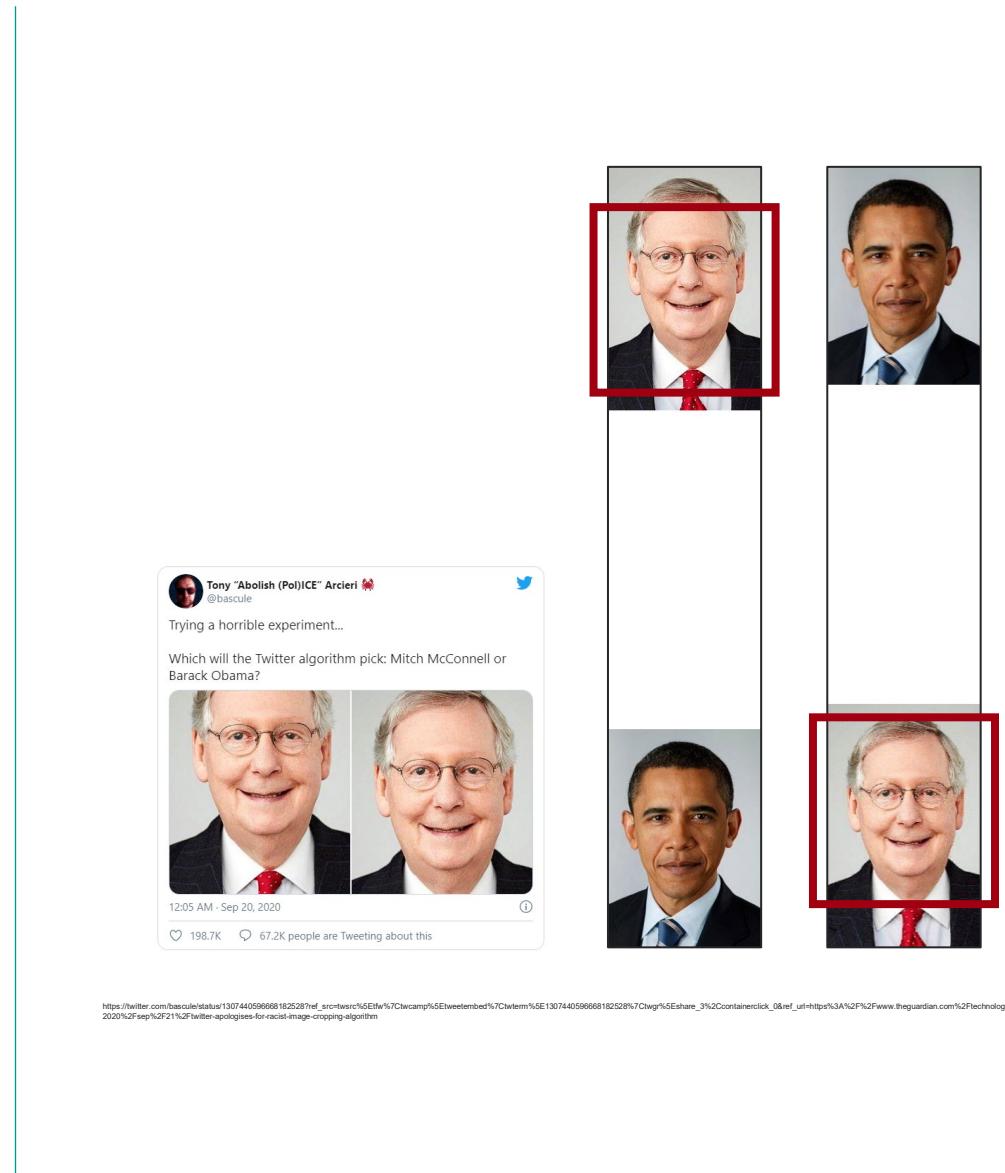
"Robust Physical-World Attacks on Deep Learning Models".
<https://arxiv.org/pdf/1707.08945.pdf> 2017.

- Die Auto-Labeling Funktion in Google Fotos taggte Fotos von dunkelhäutigen Menschen mitunter fälschlicherweise mit der Klasse „Gorillas“
- Mangels einfacher technischer Lösungen für das Problem wurde die „Gorillas“-Klasse aus dem Auto-Labeling entfernt

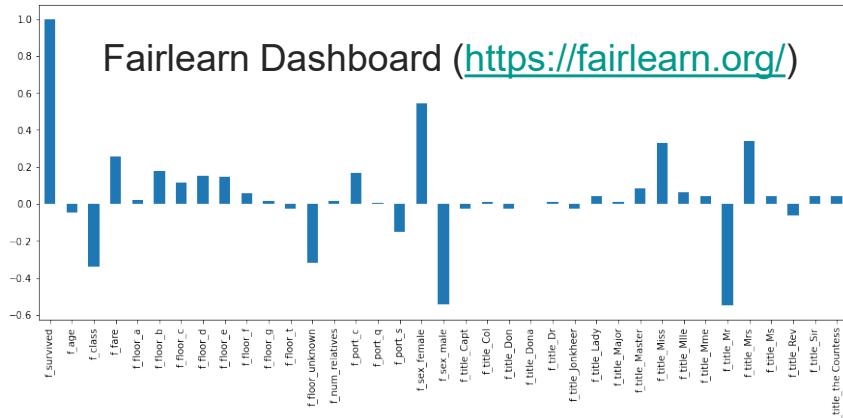


<https://www.googlewatchblog.de/wp-content/uploads/google-photos-gorilla.jpg>

- Bilder in Tweets werden in der Feed-Ansicht nur als Ausschnitt dargestellt („Cropping“)
- Der Ausschnitt wird über ein Machine Learning Modell bestimmt, das den „relevantesten“ Teil des Bildes ermittelt
- Der automatische Zuschneide-Algorithmus zeigte eine unbeabsichtigte rassistische Verzerrung (racial bias).
- Twitter Benutzer stellten im September 2020 fest, dass das verwendete „Cropping“-Verfahren Gesichter mit hellerer Hautfarbe bevorzugt („Racial-Bias“)
- Beispiel: Mitch McConnell vs. Barack Obama

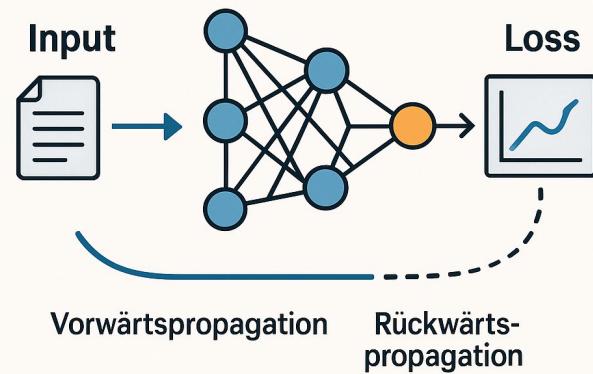


- Ein Modell soll unvoreingenommen (unbiased) und gerecht entscheiden – unabhängig von sensiblen Merkmalen wie Geschlecht, Alter, Herkunft usw.
- Im einfachsten Fall lässt sich Fairness durch Messen des Einflusses kritischer Merkmale (z.B. Geschlecht, Alter, Herkunft) auf die Vorhersagen überprüfen (z.B. mittels Feature Wichtigkeit /Korrelation)



- Feature Importance zeigt, welche Eingabeveriablen das Modell am stärksten für seine Entscheidungen nutzt.
- Auch wenn ein sensibles Attribut (z. B. „Geschlecht“) nicht direkt im Modell verwendet wird, können korrelierte Variablen (z. B. Beruf, Einkommen, Familienstand) indirekt Diskriminierung verursachen.
- Bei hochdimensionalen Eingabedaten sind die Zusammenhänge unter Umständen nur durch Validierung auf zusätzlichen diversen Datasets sichtbar.

Rückpropagation (Training)



Exkurs: Backpropagation Training

- Künstliches Neuron

Transferfunktion:

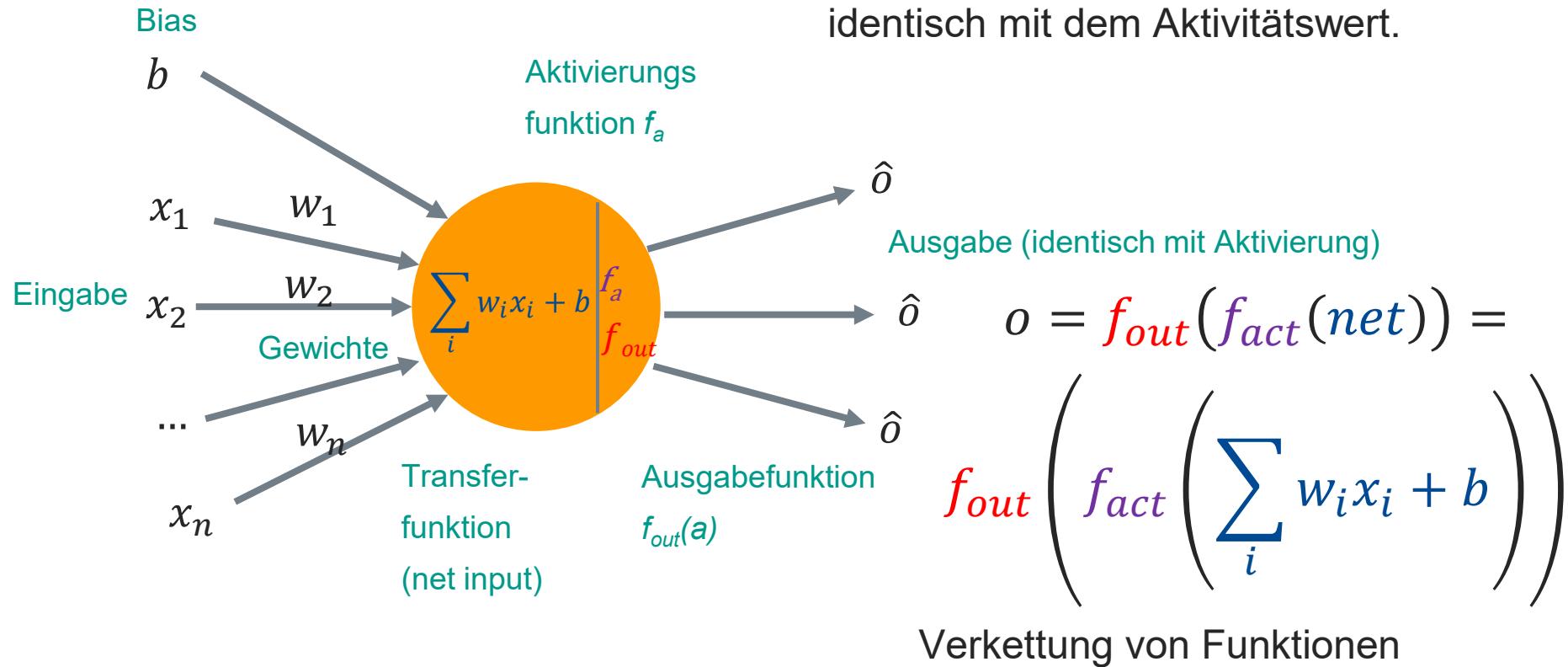
(net input):

$$\text{net} = \sum_i w_i x_i + b$$

Aktivierungsfunktion: $a = f_a(\text{net})$
beschreibt die Aktivierung des Neurons.

Ausgabefunktion: $= f_{\text{out}}(a)$
 $f_{\text{out}}(a) = \text{id}(f(a))$

Die Ausgabe ist in den meisten Fällen
identisch mit dem Aktivitätswert.



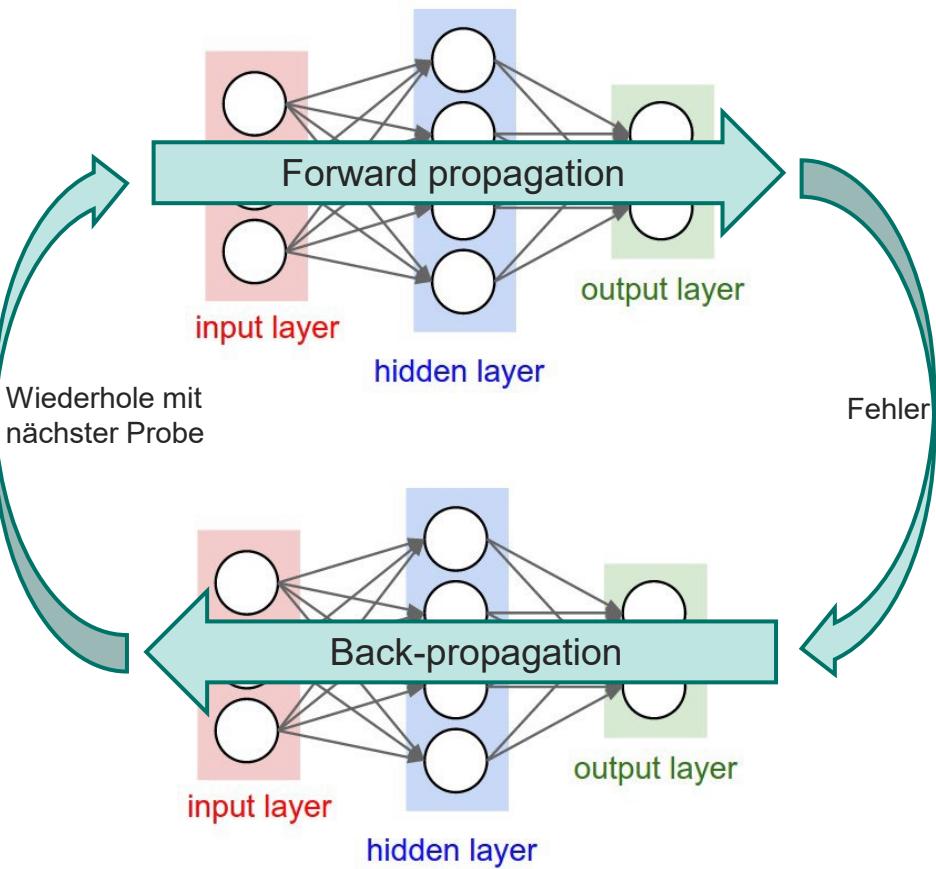
Gegeben $f(x) = g(h(x))$, die erste Ableitung einer Kette von Funktionen:

$$f' = (g(h))' = g'(h) \cdot h'$$

Beispiele:

- $((2x)^3)' = 3(2x)^2 \cdot 2 = 24x^2$
- $(\ln(x^2))' = \frac{1}{x^2} \cdot 2x = \frac{2}{x}$
- $(e^{2x^2})' = e^{2x^2} \cdot 4x$
- $((3x^2 + 4x)^3)' = 3 \cdot (3x^2 + 4x)^2 \cdot (6x + 4)$

- Multilayer Perceptron (MLP) – Training (Gradient Descent)



Forward Propagation (Vorwärtspropagation)

- Führe die Eingabedaten nacheinander durch die Schichten der Neuronen.
- Verwende die Parameter (Gewichte und Bias-Werte), um die Aktivierungen in jedem Neuron zu berechnen.
- Die Ergebnisse der Ausgabeschicht stellen die Vorhersagen dar.
- Vergleiche die Vorhersagen mit den tatsächlichen Zielwerten (Labels) und berechne den Fehler.

Back-propagation (Fehlerrückführung)

- Verwende den Fehler, um die Gradienten in Bezug auf alle Parameter im Netzwerk zu berechnen – rückwärts von der Ausgabeschicht zur Eingabeschicht.
- Nutze diese Gradienten, um die Gewichte zu aktualisieren (ähnlich wie beim Gradient Descent in der linearen Regression).
- Wiederhole den Vorwärts-Rückwärts-Zyklus für die nächsten Trainingsbeispiele – bis das Modell konvergiert.
- Epoche: Ein vollständiger Durchlauf mit Vorwärts- und Rückwärtspropagation über alle Trainingsdaten.

Batch-Learning (Offline-Lernen)

- Beim Batch-Learning (auch Offline Learning genannt) werden **alle Trainingsdaten** zu einem Batch zusammengefasst.
- Das neuronale Netz berechnet zunächst **für alle Trainingsbeispiele** die **Vorhersagen und Fehler (Loss)**.
- Anschließend werden die Gradienten **über alle Beispiele gemittelt**, und erst dann werden die **Gewichte einmal pro Epoche** aktualisiert.
- Der **Gesamtfehler** basiert also auf **allen Trainingsbeispielen**.
- Das Verfahren ist mathematisch stabil und genau, erfordert aber hohe Rechenleistung und Speicher, da alle Daten gleichzeitig verarbeitet werden müssen.

$$w_{ik} = w_{ik} + \sum_{p=1}^P {}^p \Delta w_{ik}$$

Online Learning (Inkrementelles Lernen)

- Für jedes neue Trainingsbeispiel werden die **Gewichte sofort** angepasst.
- Der **Fehler wird für jedes einzelne Beispiel** berechnet.
- Das Modell reagiert unmittelbar auf neue Daten, kann dadurch aber auch instabil werden (z. B. bei verrauschten Daten).

Mini-Batch Learning

- Zwischenform zwischen *Batch-* und *Online-Learning*.
- Die Gewichte werden nach einer **kleinen Teilmenge von Beispielen** (z. B. 50 oder 100 Samples) aktualisiert.
- Der Fehler basiert auf diesem Mini-Batch.
- Vereint die Stabilität von Batch-Learning mit der Schnelligkeit von Online-Learning.

Backpropagation – Training Variants

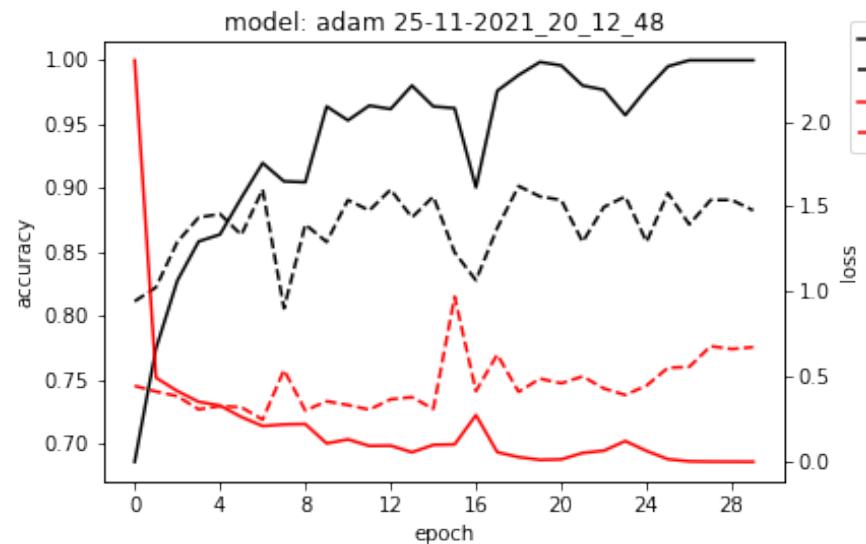
Merkmal	Batch-Learning (Offline)	Mini-Batch-Learning	Online-Learning (Inkrementell)
Datenverarbeitung	Alle Trainingsdaten werden gemeinsam verarbeitet	Trainingsdaten werden in kleine Batches (z. B. 50–200 Samples) aufgeteilt	Jedes einzelne Trainingsbeispiel wird sofort verarbeitet
Gewichts-aktualisierung	Nach einer Epoche (einmal pro Gesamtdurchlauf)	Nach jedem Mini-Batch	Nach jedem einzelnen Beispiel
Fehlerberechnung	Über alle Trainingsdaten (globaler Fehler)	Über den jeweiligen Mini-Batch	Nur über das aktuelle Beispiel
Rechenaufwand pro Schritt	Hoch	Mittel	Gering
Stabilität	Sehr stabil, da Mittelung über alle Daten	Kompromiss zwischen Stabilität und Geschwindigkeit	Geringere Stabilität, da stark von einzelnen Beispielen abhängig
Lernverhalten	Langsam, aber präzise	Guter Kompromiss aus Genauigkeit und Effizienz	Sehr schnell, aber potenziell schwankend
Einsatzgebiet	Bei kleinen bis mittleren Datensätzen, wenn hohe Genauigkeit wichtig ist	Standardverfahren beim Training großer neuronaler Netze	Bei kontinuierlich eintreffenden Datenströmen (z. B. Online-Systeme, Sensoren)

Verlustfunktion (Loss Function)

- Optimierungsalgorithmus (Optimization Algorithm)
- Die Funktion, die verwendet wird, um eine mögliche Lösung (z. B. einen Satz von Gewichten) zu bewerten, wird **Zielfunktion** (Objective Function) genannt.
- Ziel ist es, die Zielfunktion zu **maximieren** oder zu **minimieren** – also eine Lösung zu finden, die den besten (höchsten oder niedrigsten) Wert liefert.
- In neuronalen Netzen bedeutet das meist: Den **Fehler minimieren**.
- Die Zielfunktion wird auch als **Kostenfunktion** (Cost Function) oder **Verlustfunktion** (Loss Function) bezeichnet.
- Der von der Verlustfunktion berechnete Wert heißt **Loss**.
- Der Fehler E wird somit aus der Verlustfunktion berechnet.

$$E = \frac{1}{2} \sum_j (t_j - o_j)^2$$

$$E = \sum t_j \cdot \log(10^{-15} + o_j)$$



- Das Backpropagation-Lernverfahren ist ein Verfahren des überwachten Lernens zum Anpassen der Verbindungsgewichte. Die Gewichtsänderung erfolgt ausgehend von einem Fehlersignal, das sich aus der Abweichung zwischen berechneter und erwarteter Ausgabe ergibt. Die Anpassung der Gewichte erfolgt schichtweise rückwärts – beginnend bei der Ausgabeschicht bis zur Eingabeschicht.
- Ziel: Alle Gewichte so ändern, dass der Fehler (Training – Vorhersage) minimiert wird.

$$E(W_j) = E(w_{1j}, w_{2j}, \dots, w_{nj})$$

$$\Delta w_{ij} = -\lambda \cdot \partial E / \partial w_{ij}$$

- Die Gewichtsanpassung erfolgt proportional zur negativen Ableitung des Fehlers nach dem jeweiligen Gewicht.
- Die Lernrate λ bestimmt die Stärke der Anpassung.
- Der Fehler $E(W_j)$ des Neurons j ist eine Funktion aller eingehenden Gewichte.
- Die Gewichtsanpassung ergibt sich zu: $\Delta w_{ij} = -\lambda \cdot \partial E / \partial o_j \cdot \partial o_j / \partial \text{net}_j \cdot \partial \text{net}_j / \partial w_{ij}$
- Hierbei beschreibt o_j die Ausgabe des Neurons j , welche wiederum eine Verkettung mehrerer Funktionen ist: Eingabe → Aktivierungsfunktion → ggf. Ausgabefunktion.

Abhängigkeiten in der Gewichtsanpassung

„Das Backpropagation-Lernverfahren ist ein Verfahren des überwachten Lernens zum Anpassen der Verbindungsgewichte. Die Gewichtsänderung erfolgt ausgehend von einem aus der Abweichung der berechneten Ausgabe von der erwarteten Ausgabe bestimmten Fehlersignal. Die Gewichtsänderung wird schichtenweise, beginnend mit den Verbindungen zur Ausgabeschicht rückwärts in Richtung Eingabeschicht, vorgenommen.“
(Lämmel, 218, S. 212)

Hinweis: In den folgenden Gleichungen wird der Eingabewert x_i als o_i bezeichnet, da er – abhängig von der Schicht – die Ausgabe der vorherigen Schicht darstellt, außer in der ersten Eingabeschicht.

- Grundidee: Alle Gewichte werden so angepasst, dass der Fehler – basierend auf der Differenz zwischen Training (Sollwert) und Vorhersage (Istwert) – minimiert wird.

$$E(W_j) = E(w_{1j}, w_{2j}, \dots, w_{nj})$$

Der Fehler $E(W_j)$ der Ausgabe des Neurons j ist eine Funktion aller eingehenden Gewichte.

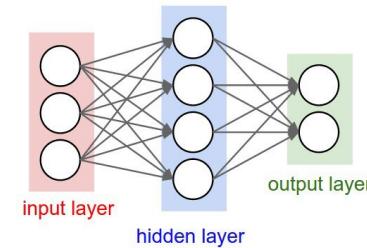
$$\Delta W_{ij} = -\lambda \cdot \frac{\partial E}{\partial w_{ij}}$$

Gewichtsänderung (Korrektur) mit Lernrate λ

$$o_j = f_{out}(f_{act}(net_j)) \quad net_j = \sum o_i \cdot w_{ij} \quad f_{out} = Id$$

$$\Delta W_{ij} = -\lambda \cdot \frac{\partial E}{\partial o_j} \cdot \frac{\partial o_j}{\partial net_j} \cdot \frac{\partial net_j}{\partial w_{ij}}$$

Die Ausgabe o_j ist eine Verkettung mehrerer Funktionen – beginnend mit dem Netzeingang (net), gefolgt von der Aktivierungsfunktion und anschließend der Ausgabefunktion.



- Die partielle Ableitung $\partial E / \partial w_{(ij)}$ kann mithilfe der Kettenregel in drei Teile zerlegt werden:

$$\Delta w_{(ij)} = -\lambda \frac{\partial E}{\partial o_j} \cdot \frac{\partial o_j}{\partial net_j} \cdot \frac{\partial net_j}{\partial w_{(ij)}}$$

- 1. Abhangigkeit Netzinput – Gewicht: $net_j = \sum(o_i \cdot w_{ij}) \Rightarrow \frac{\partial net_j}{\partial w_{(ij)}} = o_i$
- 2. Abhangigkeit Ausgabe – Aktivierungsfunktion: $\frac{\partial o_j}{\partial net_j} = f'_{akt}(net_j)$

$$\Delta w_{ij} = -\lambda \cdot \frac{\partial E}{\partial o_j} \cdot \frac{\partial o_j}{\partial net_j} \cdot \frac{\partial net_j}{\partial w_{ij}}$$

1. Abhängigkeit Netzeingabe - Gewichte

$$\frac{\partial net_j}{\partial w_{ij}} = \frac{\partial}{\partial w_{ij}} \sum_k o_k \cdot w_{kj} = o_i$$

Alle Summanden bis auf k=i entfallen

$$net_j = \sum_i o_i \cdot w_{ij}$$

2. Abhängigkeit Ausgabe – Aktivierungsfunktion

$$\frac{\partial o_j}{\partial net_j} = \frac{\partial f_{act}(net_j)}{\partial net_j} = f'_{act}(net_j)$$

Beispiel: Logistische Aktivierungsfunktion

- Die logistische (sigmoid) Aktivierungsfunktion ist eine der häufigsten Aktivierungsfunktionen:
 $f_{\text{log}}(x) = 1 / (1 + e^{-x})$
- Ihre Ableitung lautet: $f'_{\text{log}}(x) = f_{\text{log}}(x) \cdot (1 - f_{\text{log}}(x))$
- Damit gilt: $\partial o_j / \partial \text{net}_j = o_j \cdot (1 - o_j)$
- Diese Eigenschaft ist rechnerisch effizient, da die Ableitung direkt aus der Ausgabe o_j berechnet werden kann – ohne zusätzliche Exponentialberechnung.

Beispiel: Logistische Aktivierungsfunktion

$$\Delta w_{ij} = -\lambda \cdot \frac{\partial E}{\partial o_j} \cdot \frac{\partial o_j}{\partial net_j} \cdot \frac{\partial net_j}{\partial w_{ij}}$$

$$\frac{\partial o_j}{\partial net_j} = \frac{\partial f_{act}(net_j)}{\partial net_j} = f'_{act}(net_j)$$

- Beispiel: Logistische Aktivierungsfunktion f_{act}

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

$$f'_{Logistic}(x) = \frac{d}{dx} \left(\frac{1}{1 + e^{-x}} \right) = f_{Logistic}(x) \cdot (1 - f_{Logistic}(x))$$

$$\frac{\partial o_j}{\partial net_j} = f_{Logistic}(net_j) \cdot (1 - f_{Logistic}(net_j)) = o_j \cdot (1 - o_j)$$

per Definition of f_{act}

- Der Gesamtfehler eines neuronalen Netzes wird definiert als:
 $E = \frac{1}{2} \sum (t_j - o_j)^2 \rightarrow$ Differenz zwischen Soll- und Istwert
- Die Ableitung des Fehlers nach der Ausgabe o_j ergibt: $\partial E / \partial o_j = -(t_j - o_j)$
- Dieser Ausdruck beschreibt den Fehler des Ausgabeneurons – also wie stark die berechnete Ausgabe vom Trainingsziel abweicht.
- Einsetzen in die Backpropagation-Regel ergibt:
 $\Delta w_{(ij)} = -\lambda (t_j - o_j) f'_{akt}(\text{net}_j) o_i$
- Jedes Gewicht wird proportional zum Fehler, zur Ableitung der Aktivierungsfunktion und zum Eingabewert angepasst.

Fehlerterm und Ausgabe-Ableitung

$$\Delta w_{ij} = -\lambda \cdot \frac{\partial E}{\partial o_j} \cdot \frac{\partial o_j}{\partial net_j} \cdot \frac{\partial net_j}{\partial w_{ij}}$$

$$\frac{\partial net_j}{\partial w_{ij}} = \frac{\partial}{\partial w_{ij}} \sum_k o_k \cdot w_{kj} = o_i$$

$$\frac{\partial o_j}{\partial net_j} = \frac{\partial f_{act}(net_j)}{\partial net_j} = f'_{act}(net_j)$$

3. Fehler

$$E = \frac{1}{2} \sum_j (t_j - o_j)^2$$

Training - Ausgabe

Ausgabeknotenfehler

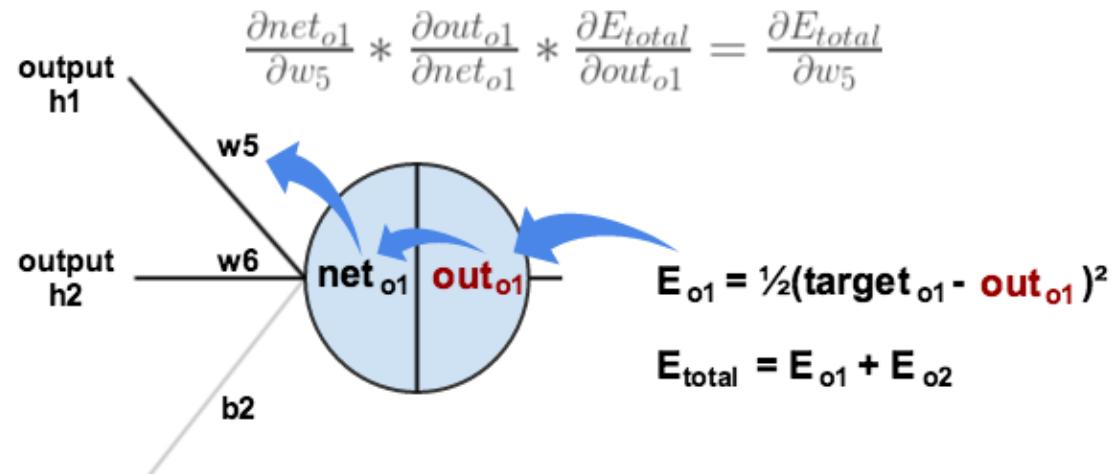
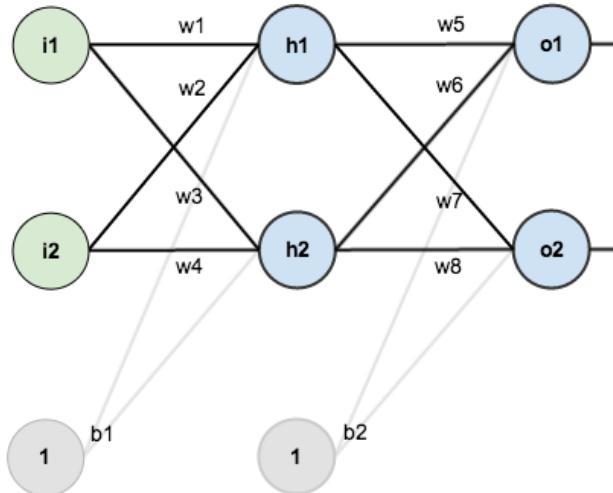
Lies als: Der Ausgabefehler des Neurons j ist die Differenz zwischen dem Zielwert (Trainingswert) und dem vom neuronalen Netz berechneten Wert.

$$\frac{\partial E}{\partial o_j} = \frac{\partial}{\partial o_j} \left(\frac{1}{2} \sum_k (t_k - o_k)^2 \right) = -(t_j - o_j)$$

if ($j = k$) $\frac{\partial}{\partial o_j} \neq 0$

- Grafische Darstellung des Ausgabeknoten-Fehlers (letzte Schicht)

$$\frac{\partial E}{\partial o_j} = - \sum \frac{\partial E}{\partial net_k} \cdot \frac{\partial net_k}{\partial o_j} = \sum_k \left(\delta_k \cdot \frac{\partial}{\partial o_j} \sum_i o_i \cdot w_{ik} \right) = \sum_k \delta_k \cdot w_{jk}$$



Der **Gesamtfehler** ist die **Summe aller Fehler der Ausgabeneuronen**.

- In den versteckten Schichten (Hidden Layers) wird der Fehler nicht direkt durch den Sollwert bestimmt, sondern muss über die Fehler der nachfolgenden Schichten zurückpropagiert werden.
- Die allgemeine Backpropagation-Regel bleibt bestehen: $\Delta w_{(ij)} = -\lambda \frac{\partial E}{\partial o_j} \cdot \frac{\partial o_j}{\partial net_j} \cdot \frac{\partial net_j}{\partial w_{(ij)}}$
- Für Neuronen der Hidden Layer gilt: sie haben keinen direkten Zielwert t_j .
- Der Fehler $\frac{\partial E}{\partial o_j}$ ergibt sich daher aus der Summe der Fehler der nachfolgenden Neuronen (Index k):
$$\frac{\partial E}{\partial o_j} = \sum_{(k)} (\delta_k \cdot w_{jk})$$
- Dabei ist $\delta_k = -\frac{\partial E}{\partial net_k}$ der Fehleranteil des nächsten Layers.
- Damit kann der Fehlerterm für jedes Hidden Neuron berechnet und rückwärts propagiert werden.
- Diese Rückwärtsweitergabe der Fehler ermöglicht das Training tiefer neuronaler Netze.

Backpropagation – Fehler in den Hidden Layern

$$\Delta w_{ij} = -\lambda \cdot \frac{\partial E}{\partial o_j} \cdot \frac{\partial o_j}{\partial net_j} \cdot \frac{\partial net_j}{\partial w_{ij}}$$

$$\frac{\partial net_j}{\partial w_{ij}} = \frac{\partial}{\partial w_{ij}} \sum_k o_k \cdot w_{kj} = o_i$$

$$\frac{\partial o_j}{\partial net_j} = \frac{\partial f_{act}(net_j)}{\partial net_j} = f'_{act}(net_j)$$

- Fehler verborgene Schichten

$$net_j = \sum_i o_i \cdot w_{ij}$$

$$\frac{\partial E}{\partial o_j} = - \sum_k \frac{\partial E}{\partial net_k} \cdot \frac{\partial net_k}{\partial o_j} = \sum_k \left(\delta_k \cdot \frac{\partial}{\partial o_j} \sum_i o_i \cdot w_{ik} \right) = \sum_k \delta_k \cdot w_{jk}$$



Der Index k bezeichnet die nächste Schicht,
j ist der Index der aktuell betrachteten Schicht.

if ($j = k$) $\frac{\partial}{\partial o_j} \neq 0$

$$\delta = - \frac{\partial E}{\partial net}$$

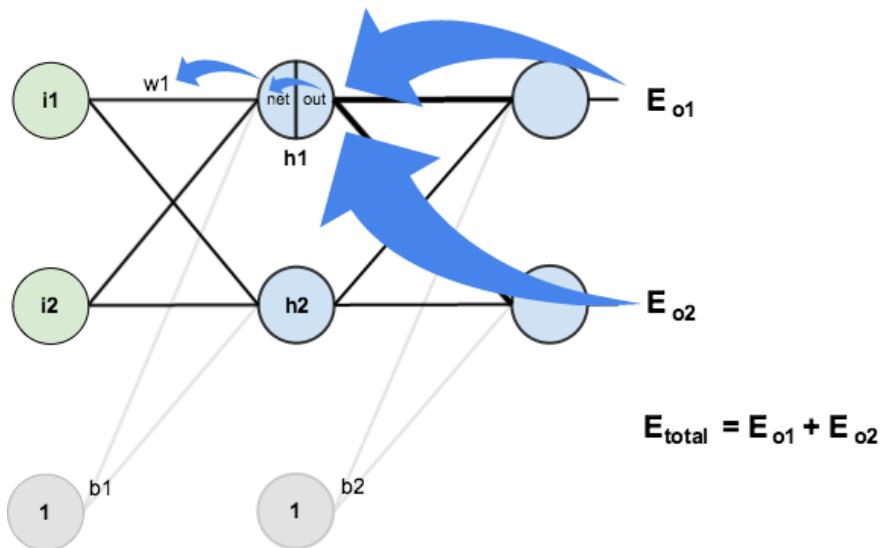
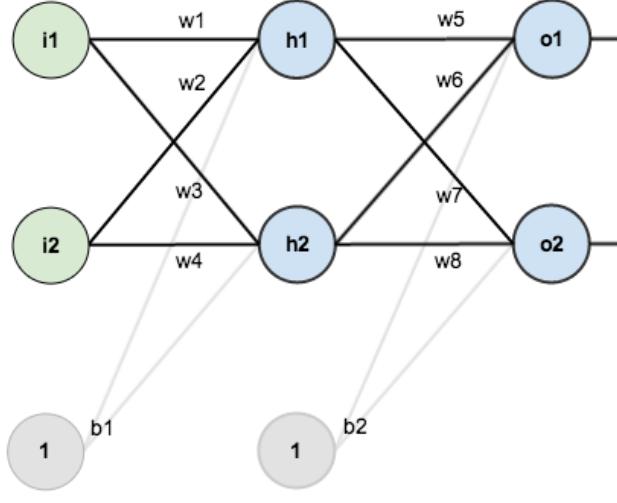
Fehlersignal

$$\frac{\partial E}{\partial o_j} \cdot \frac{\partial o_j}{\partial net_j}$$

- Grafische Darstellung der versteckten Neuronen (Hidden Layers)

$$\frac{\partial E}{\partial o_j} = \frac{\partial}{\partial o_j} \left(\frac{1}{2} \sum_k (t_k - o_k)^2 \right) = -(t_j - o_j)$$

$$\begin{aligned}\frac{\partial E_{total}}{\partial w_1} &= \frac{\partial E_{total}}{\partial out_{h1}} * \frac{\partial out_{h1}}{\partial net_{h1}} * \frac{\partial net_{h1}}{\partial w_1} \\ \downarrow \\ \frac{\partial E_{total}}{\partial out_{h1}} &= \frac{\partial E_{o1}}{\partial out_{h1}} + \frac{\partial E_{o2}}{\partial out_{h1}}.\end{aligned}$$



$$\Delta w_{ij} = -\lambda \cdot \frac{\partial E}{\partial o_j} \cdot \frac{\partial o_j}{\partial net_j} \cdot \frac{\partial net_j}{\partial w_{ij}}$$

$$\frac{\partial net_j}{\partial w_{ij}} = \frac{\partial}{\partial w_{ij}} \sum_k o_k \cdot w_{kj} = o_i$$

$$\frac{\partial o_j}{\partial net_j} = \frac{\partial f_{act}(net_j)}{\partial net_j} = f'_{act}(net_j)$$

$$\frac{\partial E}{\partial o_j} = \frac{\partial}{\partial o_j} \left(\frac{1}{2} \sum_k (t_k - o_k)^2 \right) = -(t_j - o_j)$$

$$\frac{\partial E}{\partial o_j} = - \sum \frac{\partial E}{\partial net_k} \cdot \frac{\partial net_k}{\partial o_j} = \sum_k \left(\delta_k \cdot \frac{\partial}{\partial o_j} \sum_i o_i \cdot w_{ik} \right) = \sum_k \delta_k \cdot w_{jk}$$

$$\delta_j = -\frac{\partial E}{\partial net_j}$$

Logistic function used

$$\delta_j = \begin{cases} o_j \cdot (1 - o_j) \cdot (t_j - o_j), & \text{falls } j \text{ Ausgabe-Neuron} \\ o_j \cdot (1 - o_j) \cdot \sum_k \delta_k \cdot w_{jk}, & \text{falls } j \text{ inneres Neuron} \end{cases}$$

tanh function used

$$\delta_j = \begin{cases} (1 - o_j^2) \cdot (t_j - o_j), & \text{falls } j \text{ Ausgabe-Neuron} \\ (1 - o_j^2) \cdot \sum_k \delta_k \cdot w_{jk}, & \text{falls } j \text{ inneres Neuron} \end{cases}$$

$$\Delta w_{ij} = \lambda \cdot o_i \cdot \delta_j$$

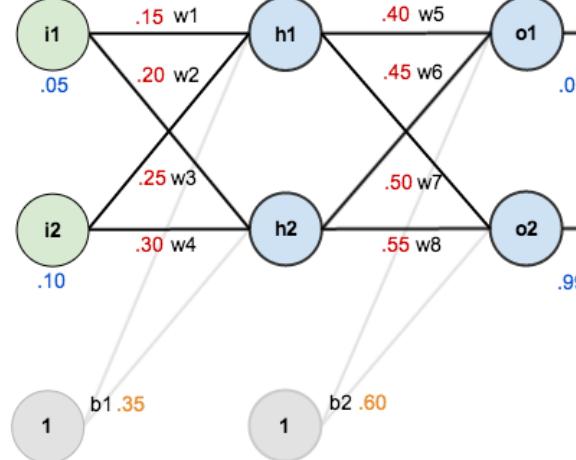
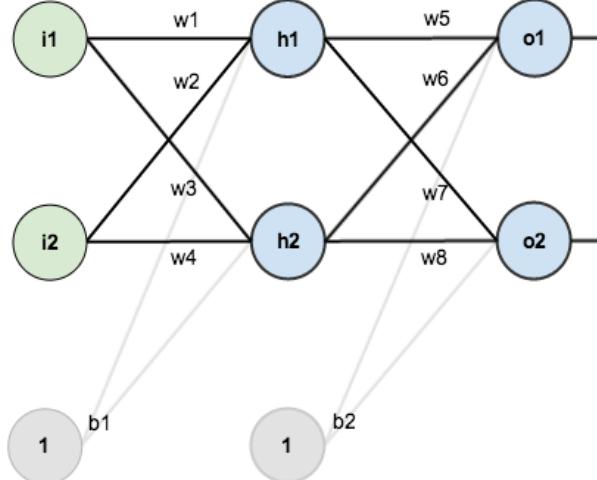
$$w'_{ij} = w_{ij} + \lambda \cdot o_i \cdot \delta_j$$

```
PROCEDURE BACKPROPAGATION
nZyklen:=0;
REPEAT
    fehler := 0;
    nZyklen := nZyklen + 1;
    FOR i := 1 .. AnzahlMuster DO
        Muster i anlegen;
        FOR j := 1 .. AnzahlOutputNeuron DO
            Bestimme Ausgabewert  $o_j$ ;
            Bestimme Fehlerwert  $f_j := t_j - o_j$ ;
            Bestimme Fehlersignal  $fsig_j := o_j * (1-o_j) * f_j$ ;
            fehler := fehler +  $f_j^2$ ;
        END FOR
        FOR s:=Ausgabeschicht-1 DOWNTO ErsteSchicht DO
            FOR k:=1..AnzahlNeuronenSchichts DO
                fsum:=0;
                FOR m:=1..AnzahlNeuronenSchicht(s+1) DO
                    fsum:= fsum +  $w_{km} * fsig_{(s+1)m}$ ;
                END FOR
                fsigsk:= $o_{sk} * (1 - o_{sk}) * fsum$ ;
                FOR m:=1..AnzahlNeuronenSchicht(s+1) DO
                     $w_{km} := w_{km} + Lernrate * o_{sk} * fsig_{(s+1)m}$ ;
                END FOR
            END FOR
        END FOR;
    UNTIL fehler < tolerierbarerFehler OR Zyklenzahl erreicht
END BACKPROPAGATION
```

Lämmel, Uwe; Cleve, Jürgen
(2012): Künstliche Intelligenz. Mit
51 Tabellen, 43 Beispielen, 118
Aufgaben, 89 Kontrollfragen und
Referatsthemen. 4. Aufl.
München: Hanser.

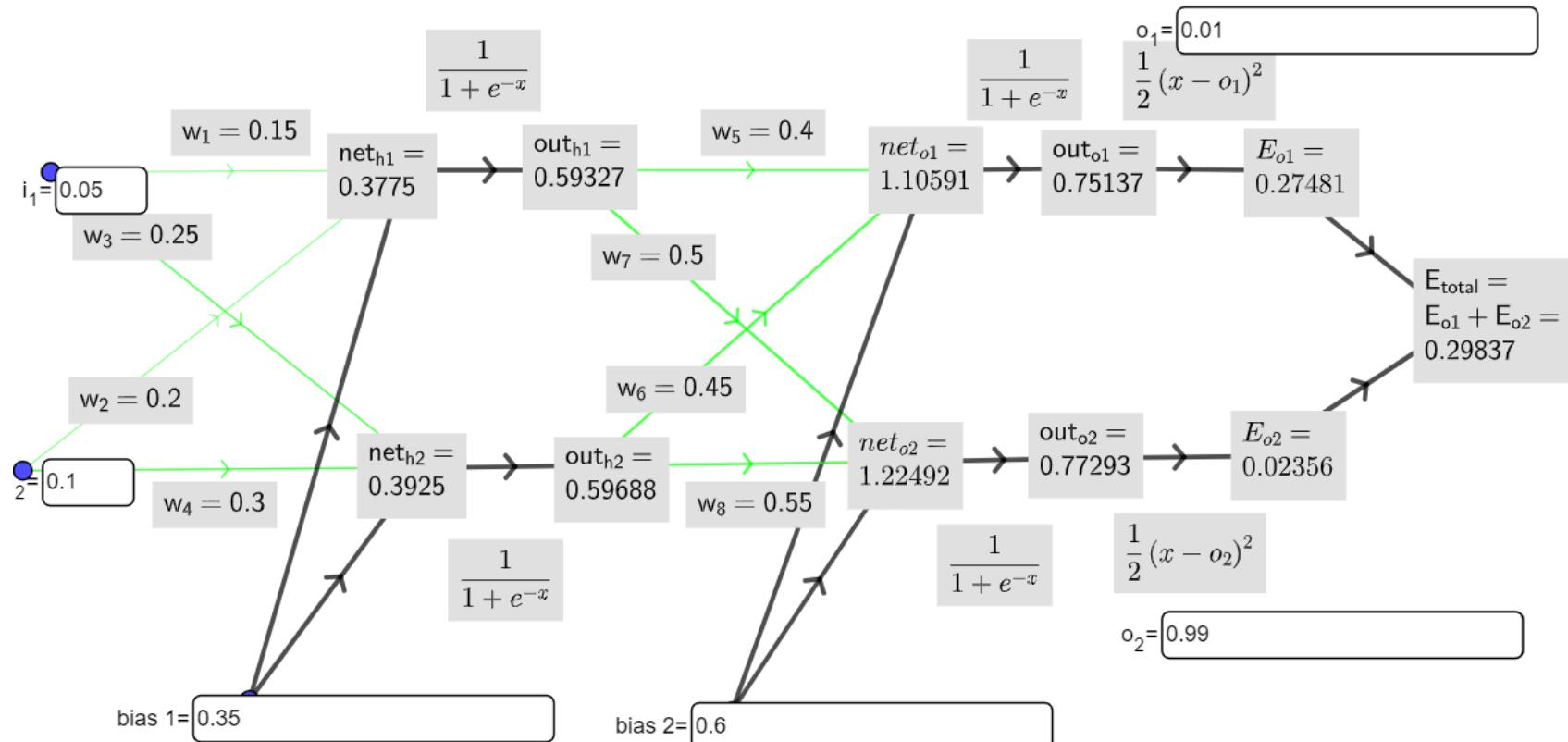
Backpropagation – Beispiel

- Das Beispiel stammt von: <https://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/comment-page-16/#comments>
- Das verwendete Tool ist GeoGebra.
- Das zugehörige Skript heißt geogebra-backprop-example.ggb.
- Quellskript: <https://www.geogebra.org/classic/dyq2rcup>
- Das Beispiel kann entweder von der Website heruntergeladen oder interaktiv online verwendet werden.



i – Eingabeschicht (Input Layer)
 h – Verborgene Schicht (Hidden Layer)
 o – Ausgabeschicht (Output Layer)
 t – Ziel- bzw. Trainingswert (Target / Training Value)
 act – Aktivierungswert (Activation Value)
 net – Netzeingangswert (Net Value)
 w – Gewichte (Weights)
 b – Bias (Schwellenwert / Bias)

Backpropagation – Beispiel Epoch 0 - initial

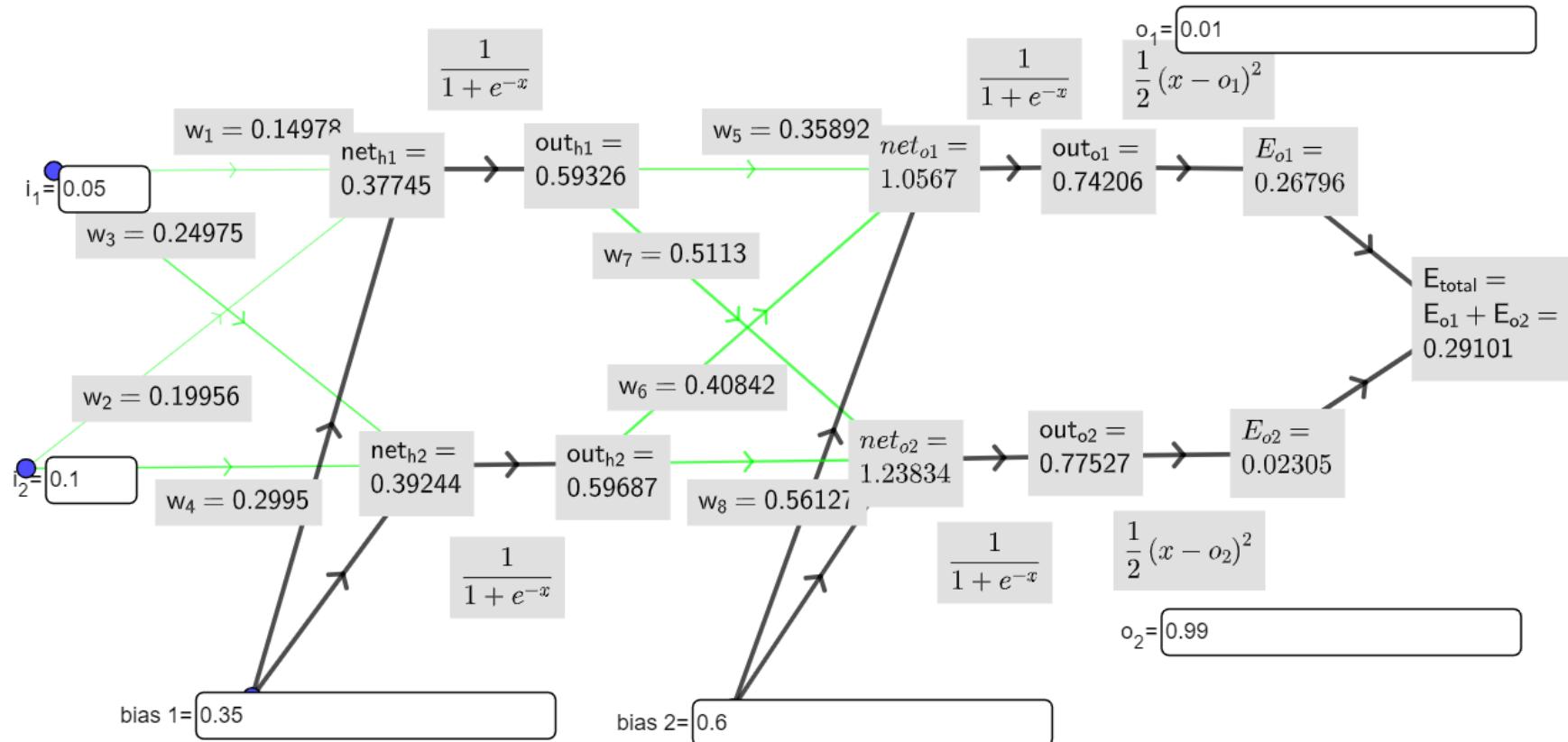


Backpropagate

learning rate ("eta")=0.5

Epoch = 0

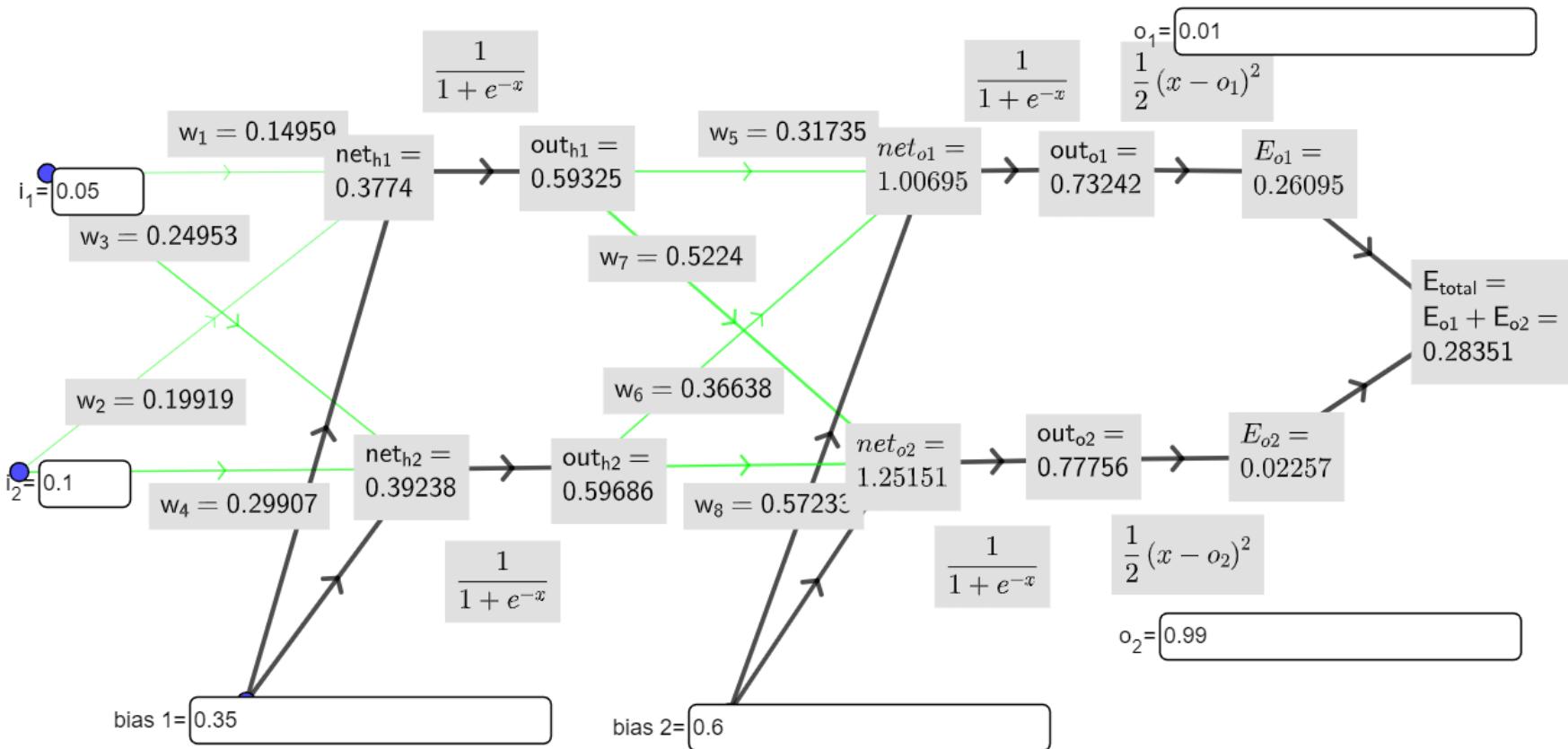
Backpropagation – Beispiel Epoch 1


Backpropagate

learning rate ("eta")=0.5

Epoch = 1

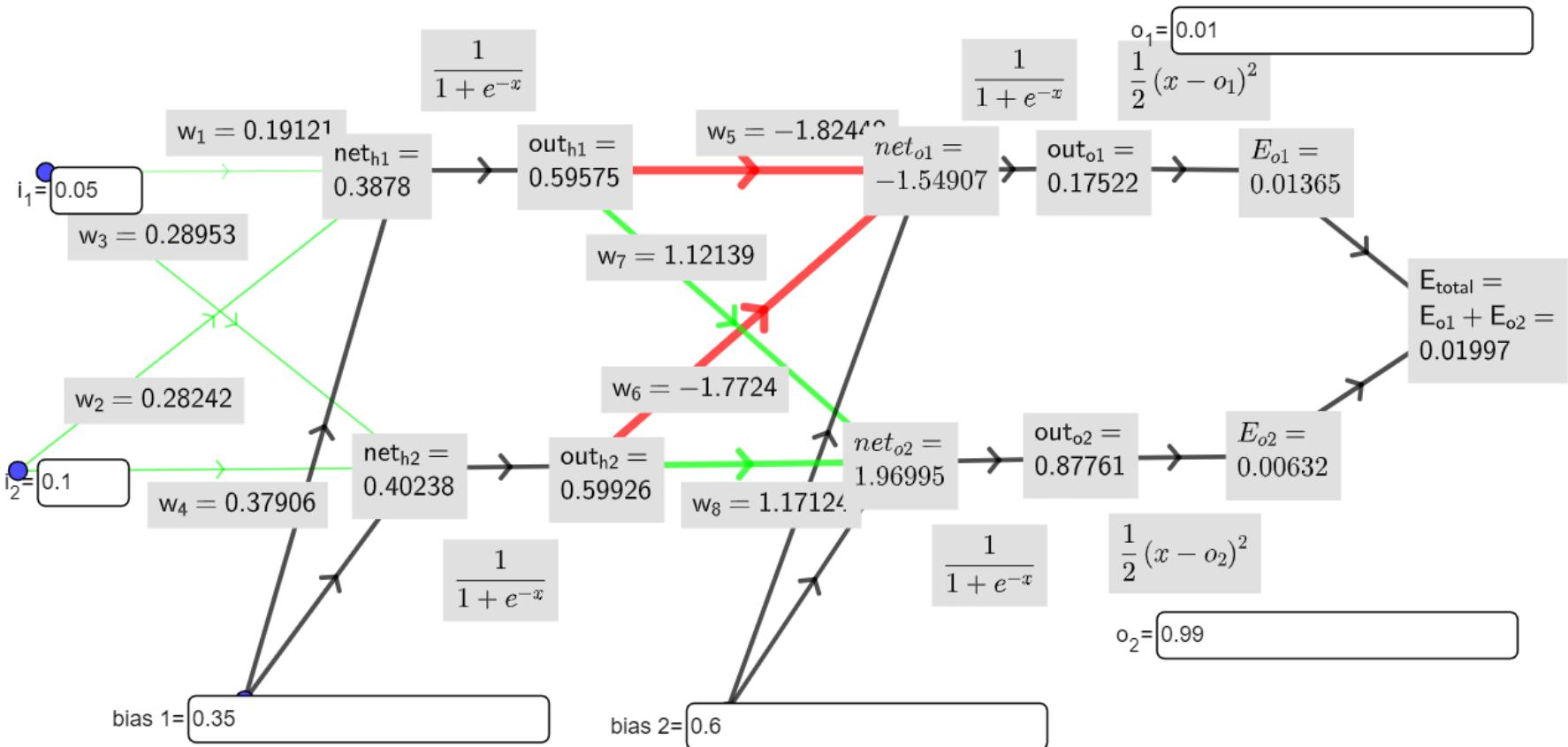
Backpropagation – Beispiel Epoch 2


Backpropagate

learning rate ("eta") = 0.5

Epoch = 2

Backpropagation – Beispiel Epoch 100


Backpropagate

learning rate ("eta")=0.5

Epoch = 100

$$\Delta w_{ij} = - \lambda \cdot \frac{\partial E}{\partial o_j} \cdot \frac{\partial o_j}{\partial net_j} \cdot \frac{\partial net_j}{\partial w_{ij}}$$

Quickprop

- Die Fehlerfunktion wird lokal besser durch eine quadratische Funktion angenähert.
- Lernverfahren zweiter Ordnung (auf Basis einer quadratischen Verlustfunktion).
- Führt zu einer schnelleren Konvergenz.

$$\Delta^{(k)} w_{ij} = \Delta^{(k-1)} w_{ij} \left(\frac{\nabla_{ij} E^{(k)}}{\nabla_{ij} E^{(k-1)} - \nabla_{ij} E^{(k)}} \right)$$

- Dabei ist w_{ij} das Gewicht des Neurons j für den Eingang i, und E die Summe der Fehler.

$$\Delta w_{ij} = - \lambda \cdot \frac{\partial E}{\partial o_j} \cdot \frac{\partial o_j}{\partial net_j} \cdot \frac{\partial net_j}{\partial w_{ij}}$$

Resilient Propagation (RPROP)

- Änderungen der Gewichte basieren nur auf dem Vorzeichen des Gradienten.
- Die Anpassung richtet sich nach der Änderung zwischen den Zeitpunkten t und t-1.

$$\Delta w_{ij}(t) = \begin{cases} -\Delta_{ij}(t) & \text{falls } S(t-1) > 0 \text{ und } S(t) > 0 \\ \Delta_{ij}(t) & \text{falls } S(t-1) < 0 \text{ und } S(t) < 0 \\ -\Delta_{ij}(t-1) & \text{falls } S(t-1) \cdot S(t) < 0 \\ -\text{sgn}(S(t))\Delta_{ij}(t) & \text{sonst} \end{cases}$$

Back percolation

- Beim klassischen Backpropagation werden die Gewichtsänderungen von Schicht zu Schicht nach hinten immer kleiner – die Gewichte der ersten Schichten ändern sich daher meist nur minimal.
- Das Back-Percolation-Verfahren modifiziert den Algorithmus so, dass auch frühere (innere) Schichten stärkere Gewichtsanpassungen erhalten.

$$\Delta w_{ij} = - \lambda \cdot \frac{\partial E}{\partial o_j} \cdot \frac{\partial o_j}{\partial net_j} \cdot \frac{\partial net_j}{\partial w_{ij}}$$

Learning Rate λ (η) – Hyperparameter

- Diese muss manuell gewählt werden.
- Der Wert der Lernrate beeinflusst, ob und wie schnell ein akzeptabler Fehler (Loss) erreicht wird.
- Verbesserungen sind möglich durch den Einsatz spezieller Optimierungsverfahren anstelle des standardmäßigen Gradienten-abstiegs.

Momentum Term

- Füge einen Anteil der vorherigen Gewichtsänderung $w(t-1)$ hinzu, gewichtet mit einem Faktor μ (im Bereich $0 \leq \mu \leq 1$), um die aktuelle Gewichtsänderung $w(t)$ zu beeinflussen.
- Dadurch wird der Lernprozess geglättet und Richtungtrends aus früheren Schritten beibehalten – dieses Verfahren ist als „Momentum-Term“ bekannt.
- $\Delta w_{ij}(t) = \lambda \cdot o_i \cdot \delta_j + \mu \cdot \Delta w_{ij}(t - 1)$

$$\Delta w_{ij} = - \lambda \cdot \frac{\partial E}{\partial o_j} \cdot \frac{\partial o_j}{\partial net_j} \cdot \frac{\partial net_j}{\partial w_{ij}}$$

AdaGrad

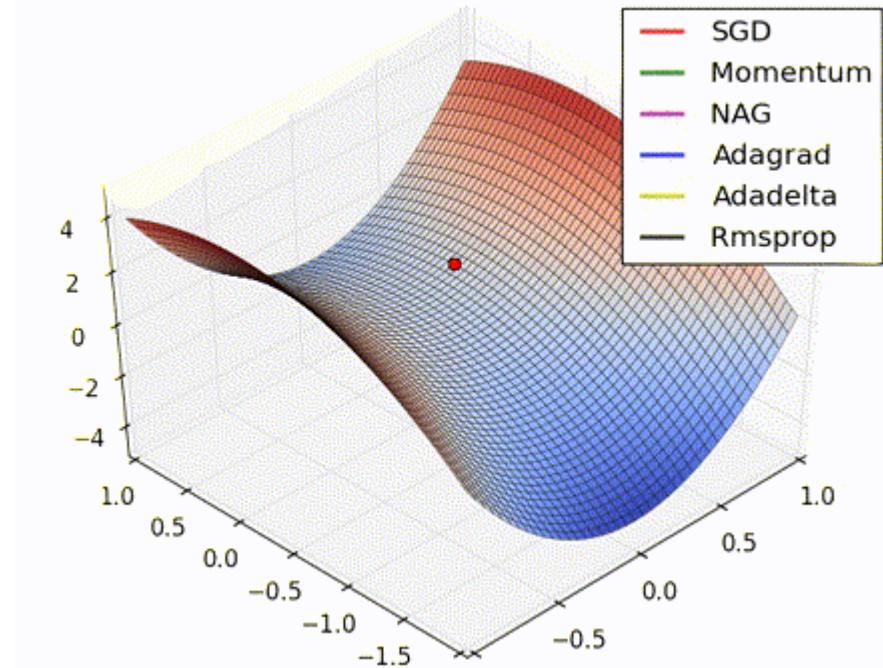
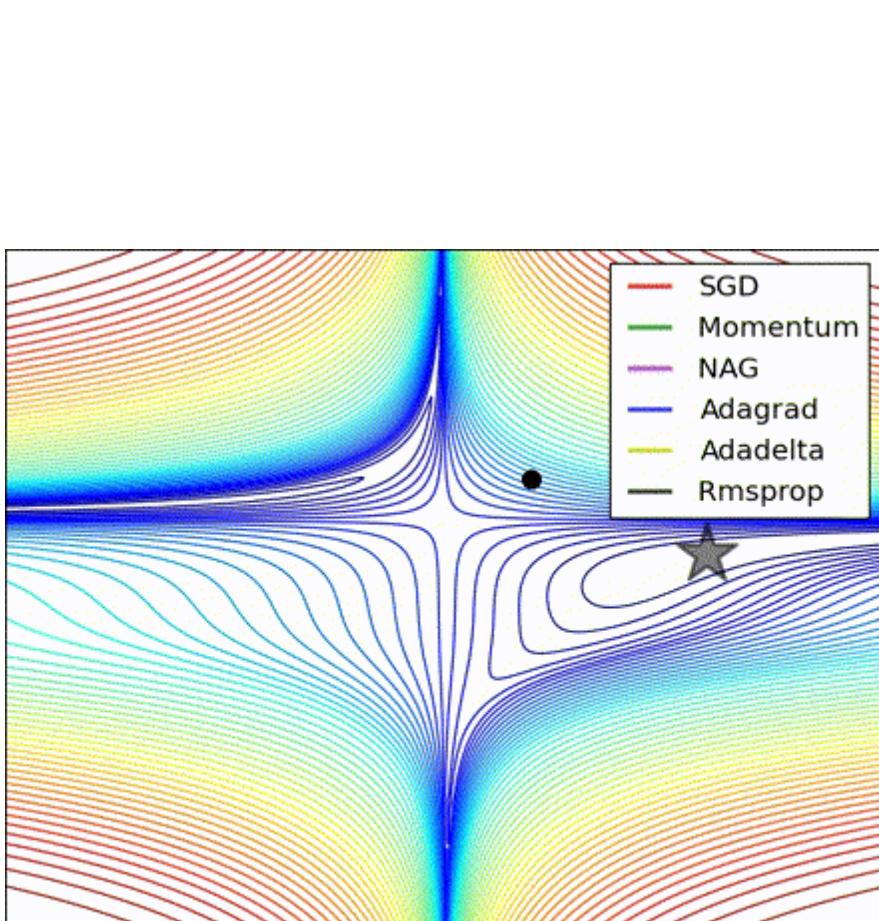
- Passt die Lernrate adaptiv für jede Dimension an – steilere Dimensionen werden schneller skaliert.
- Für einfache neuronale Netze gut geeignet, bei komplexeren Netzen führt es jedoch oft zu einem zu frühen Trainingsstopp.
- Für tiefe neuronale Netze meist nicht empfehlenswert.

RMSProp

- Berücksichtigt nur die Gradienten der letzten Iterationen, anstatt alle bisherigen einzubeziehen.
- Dadurch reagiert das Verfahren stärker auf aktuelle Veränderungen und bleibt stabiler bei langen Trainingsläufen.

ADAM (Adaptive Moment Estimation)

- Kombiniert die Vorteile von Momentum (Trägheit früherer Updates) und RMSProp (gewichteter Durchschnitt der Gradienten).
- Einer der heute am häufigsten verwendeten Optimierer für tiefe neuronale Netze.



https://ml4a.github.io/ml4a/how_neural_networks_are_trained/