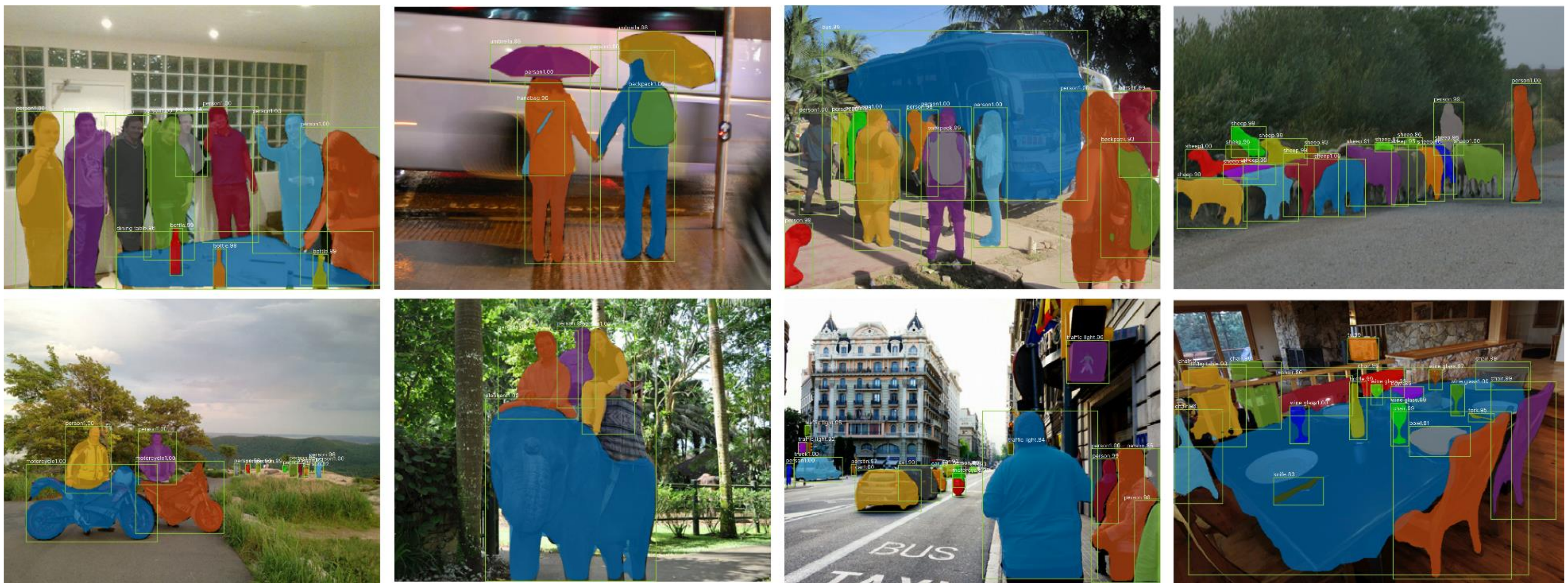
Künstliche neuronale Netze
Deep Learning

Dipl.-Inform. Ingo Boersch
Prof. Dr.-Ing. Jochen Heinsohn
FB Informatik und Medien

Instance segmentation mit Deep Learning

Jede Instanz einer Objektklasse wird erkannt:



He, K.; Gkioxari, G.; Dollár, P. & Girshick, R. B.: **Mask R-CNN**. In: *CoRR, abs/1703.06870*, **2017**

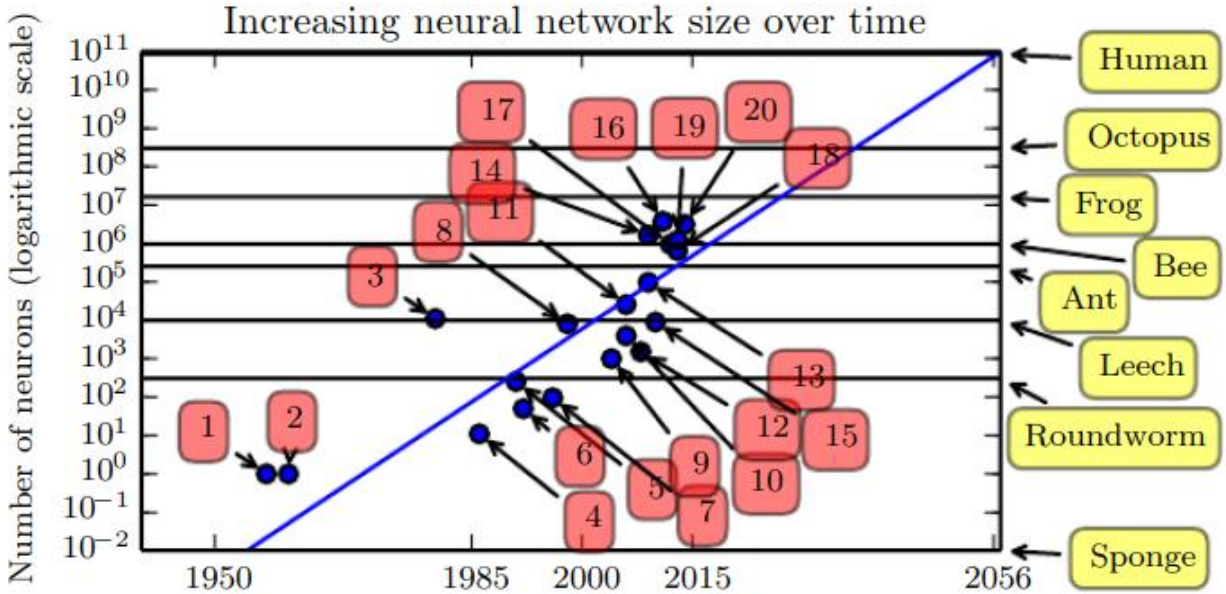


Figure 1.11: Since the introduction of hidden units, artificial neural networks have doubled in size roughly every 2.4 years. Biological neural network sizes from Wikipedia (2015).

1. Perceptron (Rosenblatt, 1958, 1962)
2. Adaptive linear element (Widrow and Hoff, 1960)
3. Neocognitron (Fukushima, 1980)
4. Early back-propagation network (Rumelhart et al., 1986b)
5. Recurrent neural network for speech recognition (Robinson and Fallside, 1991)
6. Multilayer perceptron for speech recognition (Bengio et al., 1991)
7. Mean field sigmoid belief network (Saul et al., 1996)
8. LeNet-5 (LeCun et al., 1998b)
9. Echo state network (Jaeger and Haas, 2004)
10. Deep belief network (Hinton et al., 2006)
11. GPU-accelerated convolutional network (Chellapilla et al., 2006)
12. Deep Boltzmann machine (Salakhutdinov and Hinton, 2009a)
13. GPU-accelerated deep belief network (Raina et al., 2009)
14. Unsupervised convolutional network (Jarrett et al., 2009)
15. GPU-accelerated multilayer perceptron (Ciresan et al., 2010)
16. OMP-1 network (Coates and Ng, 2011)
17. Distributed autoencoder (Le et al., 2012)
18. Multi-GPU convolutional network (Krizhevsky et al., 2012)
19. COTS HPC unsupervised convolutional network (Coates et al., 2013)
20. GoogLeNet (Szegedy et al., 2014a)

Quelle: Goodfellow, I.; Bengio, Y. & Courville, A.: **Deep Learning**. MIT Press, 2016

Beispiel 1

Januar 2014: Google kauft DeepMind Technologies für

480 Millionen Euro

DeepMind Technologies: britisch, klein (75 Mitarbeiter), gegründet 2011 vom Neurowissenschaftler Demis Hassabis, Technologien: **künstliche neuronale Netze, reinforcement learning, convolutional networks**

Beispiel 2: email vom 31.03.2014

Sehr geehrter Herr Dr. Heinsohn,

ich bin verantwortlich für das Kapitalmarktgeschäft und die Marketingaktivitäten in der xxxxxx Bank. In diesem Funktion habe ich in den letzten Quartalen den **Einsatz neuronaler Netzwerke forciert** und möchte nun einen größeren Personenkreis durch ein Inhouse-Seminar schulen lassen. Bereits in der Vergangenheit haben wir hierfür sehr gerne und erfolgreich mit Fachhochschulen zusammengearbeitet (z.B. bzgl. der mathematischen Bewertung von exotischen Optionen). Wir möchten nun diese Praxis weiter fortsetzen und würden Sie gerne hierfür gewinnen.

Geplant wäre eine **eintägige Veranstaltung bei uns im Hause für zirka 15-20 Personen**

.....

Wir **verwenden derzeit im Marketing einfache neuronale Netze für Zielgruppenselektionen (Mehrschicht-Perzeptron, 2 Hidden Layer),** können uns aber auch vorstellen, das Einsatzspektrum um Kapitalmarkaktivitäten und um den Einsatz in der Vertriebssteuerung zu erweitern. Zudem möchte ich perspektivisch Aufgaben des **Databased Marketing auf genetische Algorithmen, bzw. evolutionäre Algorithmen** ausrichten.

Vor diesem Hintergrund habe ich eine Online-Recherche vorgenommen und **bin auf Ihren Lehrstuhl aufmerksam geworden**. Haben Sie Interesse daran, unsere Mitarbeiter zu diesem Thema durch ein Inhouse-Seminar bei uns im Hause zu schulen?

Für Rückfragen stehe ich Ihnen gerne zur Verfügung und freue mich auf eine Rückantwort.

Mit freundlichen Grüßen

Motivation –
Anwendungsbeispiele künstlicher neuronaler Netze (KNN)

„Eines der überzeugendsten Merkmale für die Rechtfertigung einer (neuen) Theorie ist der Grad ihrer Verwendbarkeit in Wirtschaft, Produktion, Dienstleistung, etc...." (Kinnebrock,1994).

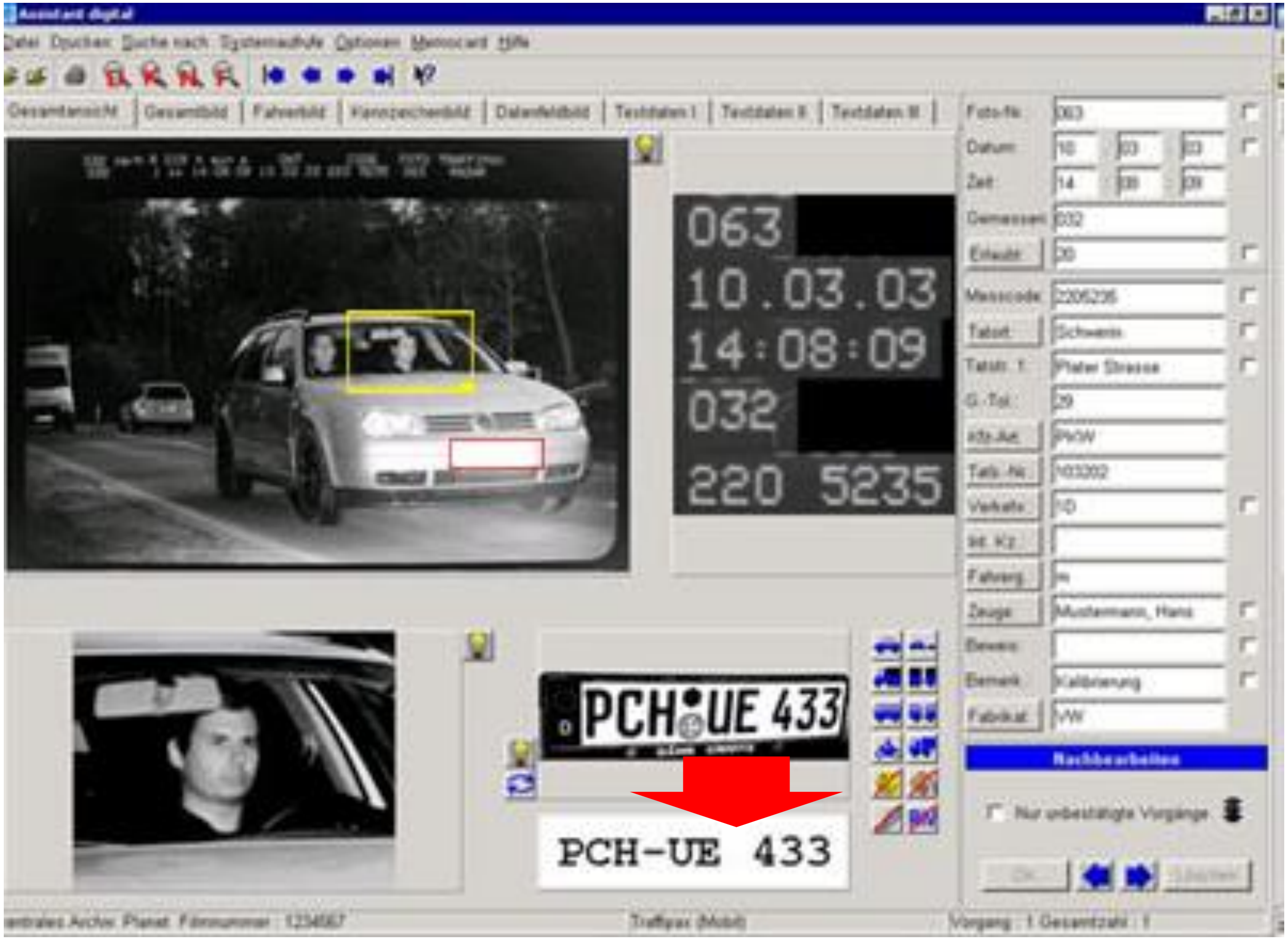
Breites Anwendungsspektrum KNN, z.B.:

- Qualitätskontrolle von Motoren durch Abhören des Motorengeräusches
- Vorlesen eines geschriebenen Textes
- Erkennen handgeschriebener Buchstaben (Daimler Benz AG Ulm)
- Management von Rentenfonts (Allg. Deutsche Direktbank)
- Vorhersage von Wirtschaftsdaten/Aktienkursen (Siemens AG)
- Bonitätsanalyse von Kreditnehmern (Neckermann Versand AG)
- Flexible Tourenplanung mit NN (Karstadt AG)

Beispiel 3: Neuronale Netze in der Bilderkennung

Quelle: PLANET intelligent systems GmbH, Mai 2009
http://www.planet.de/unternehmen/index.html

Beispiel 3: Intelligente Bild- und Schrifterkennung



Quelle: PLANET intelligent systems GmbH, Mai 2009
http://www.planet.de/produkte/verkehr/module_asdigital.html

Beispiel 4: Neuronale Netze in der Prognose

» Kontakt » Suche » Sitemap » Impressum » English

otto group

Presse

- Home
- Die Gruppe
- Segmente
- Die Unternehmen
- Karriere
- Geschäftsbericht
- Presse
  - » Pressemitteilungen
  - » Pressefotos/Logos
  - » TV-Schnittmaterial
  - » Unternehmens-informationen

Pressemitteilung

Hamburg, 19. Mai 2008

Otto Group gründet Joint Venture mit Phi-T und revolutioniert Artikel-Absatz-Prognosen

Die Otto Group, Hamburg, größte Versandhandelsgruppe der Welt und die Physics Information Technologies GmbH, Phi-T, Karlsruhe, haben das Gemeinschaftsunternehmen Phi-T products & services gegründet. Dies kündigten Hans-Otto Schrader, Vorstandsvorsitzender der Otto Group und Professor Michael Feindt, Mitbegründer von Phi-T, heute auf einer Pressekonferenz in Hamburg an. Ziel der Zusammenarbeit von Otto und Phi-T ist der branchenexklusive Einsatz der auf künstlichen neuronalen Netzen beruhenden NeuroBayes-Technologie von Prof. Feindt zur Optimierung von Artikel-Absatz-Prognosen innerhalb der Otto Group. Durch die Zusammenarbeit mit Phi-T erhält Otto Zugang zu den modernsten wissenschaftlichen Erkenntnissen und Methoden im Bereich des Data Mining und sichert sich somit einen entscheidenden Wettbewerbsvorteil.

Quelle: Otto (GmbH & Co KG), Mai 2009
http://www.ottogroup.com/701.html

Beispiel 5: Bedarfsprognose

Vorhersage des Strombedarfs der Stadt Brandenburg in den nächsten 24 Stunden mit Hilfe neuronaler Netze

FACHHOCHSCHULE BRANDENBURG

StWB
Stadtwerke Brandenburg an der Havel GmbH
Strom - Wärme - Gas

Masterarbeit

Analyse und Optimierung der Prognosegüte des Strombedarfs als Grundlage der Querverbundsoptimierung der Stadtwerke Brandenburg

vorgelegt von

David Walter

an der Fachhochschule Brandenburg
im Fachbereich Informatik und Medien

Zur Erlangung des akademischen Grades

Master of Science (M.Sc.)

Erstgutachter:	Prof. Dr.-Ing. Jochen Heinsohn (FHB)
Zweitgutachter:	Dr. Tino Schonert (StWB)
Betreuer:	Dipl.-Inform. Ingo Boersch (FHB)

Brandenburg, 19. April 2012

Was macht menschliche Intelligenz aus ?

U.a.: die Fähigkeit, Informationen und Informationseinheiten zueinander in Relation setzen zu können.

Beispiele:

- Visuelles Überprüfen von Schweißnähten: Ein erfahrener Meister hat gelernt, fehlerfreie und fehlerhafte Schweißnähte zu identifizieren
- Handschrifterkennnung: Wir alle können handgeschriebene Buchstaben erkennen, obwohl keiner dem anderen gleicht

Wie erlangen Menschen diese Fähigkeiten?

Woher können wir diese schwierige Abbildung der Eingabe (Bild eines Zeichens) auf eine Ausgabe (erkannter Buchstabe)?

Die Abbildung kann anhand von Beispielen gelernt werden!

Maschinelles Lernen

Approximation von Funktionen und Verteilungen anhand von Beispielen

Wir beobachten einen Prozess, der Daten nach einer bestimmten Verteilung erzeugt – aber wir kennen diese nicht. Die beobachteten Daten sind unsere Trainingsmenge.

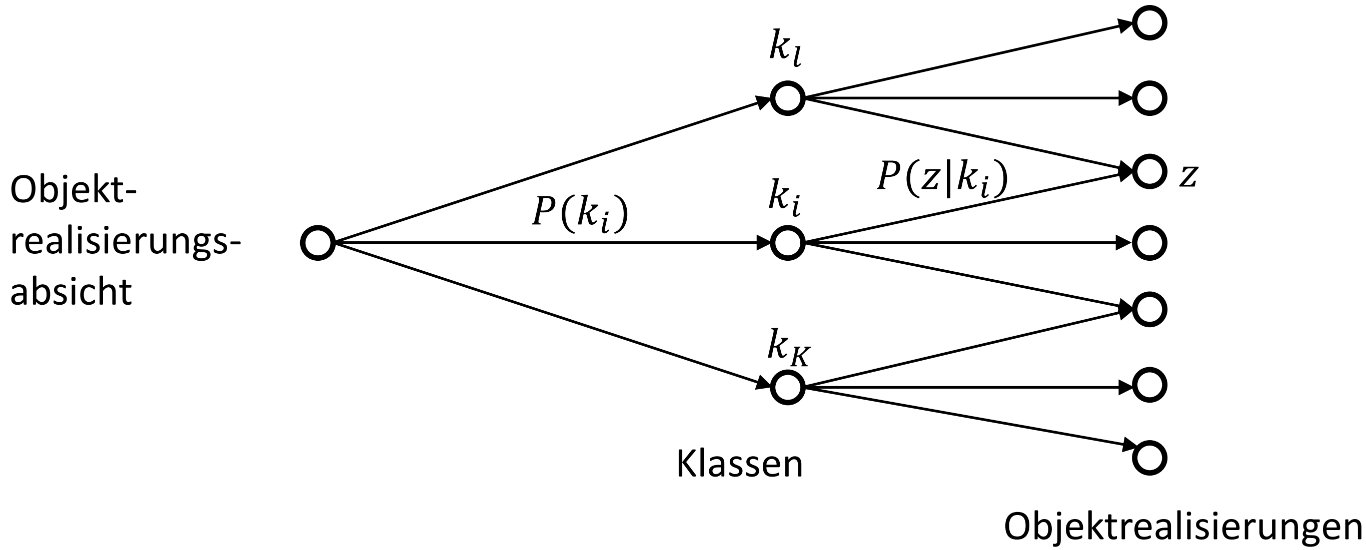
Beispiel:

- gegeben: data ∈ (Schweißnaht, Güte) $p(z, k)$
- gesucht: $Güte \ = \ f(Schweißnaht)$ $k = f(z)$

  oder $\hat{p}$(Schweißnaht, Güte) $\hat{p}(z, k)$

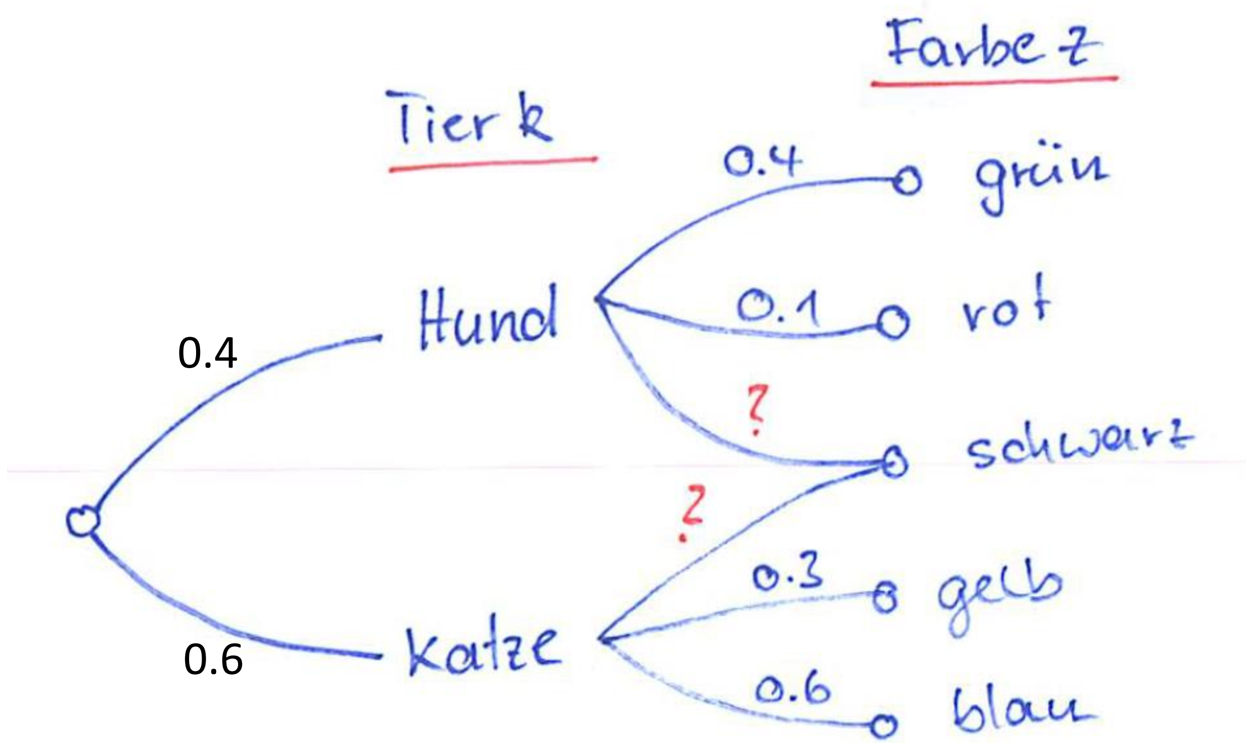
- Wie kann f repräsentiert werden?
- Wie kann f an die Trainingsdaten angepasst werden?
- Generalisierung: f soll auch auf ungesehen Eingaben gut funktionieren.

Objekterzeugungsmodell



- a-priori Wahrscheinlichkeiten (*class prior*): $P(k_i)$
- Beobachtungen / Trainingsmenge: (z, k) verteilt nach unbekannter $p_{data}(z, k)$
- Wkt. einer Beobachtung z bei der Klasse k (sog. *likelihood*): $P(z|k_i)$,

Objekterzeugungsmodell - Beispiel



Bayes-optimaler Klassifikator (BOK), maximum a-posteriori estimation (MAP)

Wähle die Klasse k, die bei einer gegebenen Beobachtung z am wahrscheinlichsten ist,
d.h. deren a-posteriori-Wahrscheinlichkeit $P(k|z)$ maximal ist.

Bayes:

$$P(k|z) := \frac{P(z,k)}{P(z)} = \frac{P(z|k)\cdot P(k)}{P(z)} \rightarrow \text{maximieren über k}$$

$$posterior = \frac{likelihood \cdot prior}{evidence}$$

Vereinfachte Entscheidung, da $P(z)$ nicht von der Klasse abhängt:

$$\mathrm{k}_{\mathrm{BOK}} = \mathrm{k}_{\mathrm{MAP}} = argmax_k(P(z|k)\cdot P(k))$$

Lernen: Schätzen von $P(z|k)$ und $P(k)$

BOK Beispiel

$$p(Katze \mid gelb) = p(k = Katze \mid z = gelb) = \frac{p(gelb \mid Katze) \cdot p(Katze)}{p(gelb)}$$

Bayes

NR:

$$p(gelb) = p(Katze) \cdot p(gelb \mid Katze) = 0.7 \cdot 0.3 = \frac{21}{100}$$

$$p(Katze \mid gelb) = \frac{0.3 \cdot 0.7}{0.7 \cdot 0.3} = 1$$

$$p(Katze \mid schwarz) = \frac{0.1 \cdot 0.7}{0.3 \cdot 0.5 + 0.7 \cdot 0.1} = \frac{7}{22}$$

a-posteriori Wkt. der Katze

totale Wkt.

$$p(Hund \mid schwarz) = \frac{0.5 \cdot 0.3}{s.o.} = \frac{15}{22}$$

a-posteriori Wkt. des Hundes

BOK

$$k = \underset{k \in \{Katze, Hund\}}{\operatorname{argmax}} \; p(z \mid k) \cdot p(k)$$

$$= \underset{k}{\operatorname{argmax}} \big(\underbrace{0.1 \cdot 0.7}_{Katze}, \underbrace{0.5 \cdot 0.3}_{Hund} \big) = \underset{k}{\operatorname{argmax}} (0.07, 0.15)$$

$$= \underline{\underline{Hund}}$$

Maximum Likelihood Esimator (MLE)

Sind die a-priori-Wahrscheinlichkeiten $P(k)$ für alle Klassen gleich, so tragen sie nicht zur Entscheidungsfindung bei und MAP vereinfacht sich zur Maximum Likelihood Estimation (MLE):

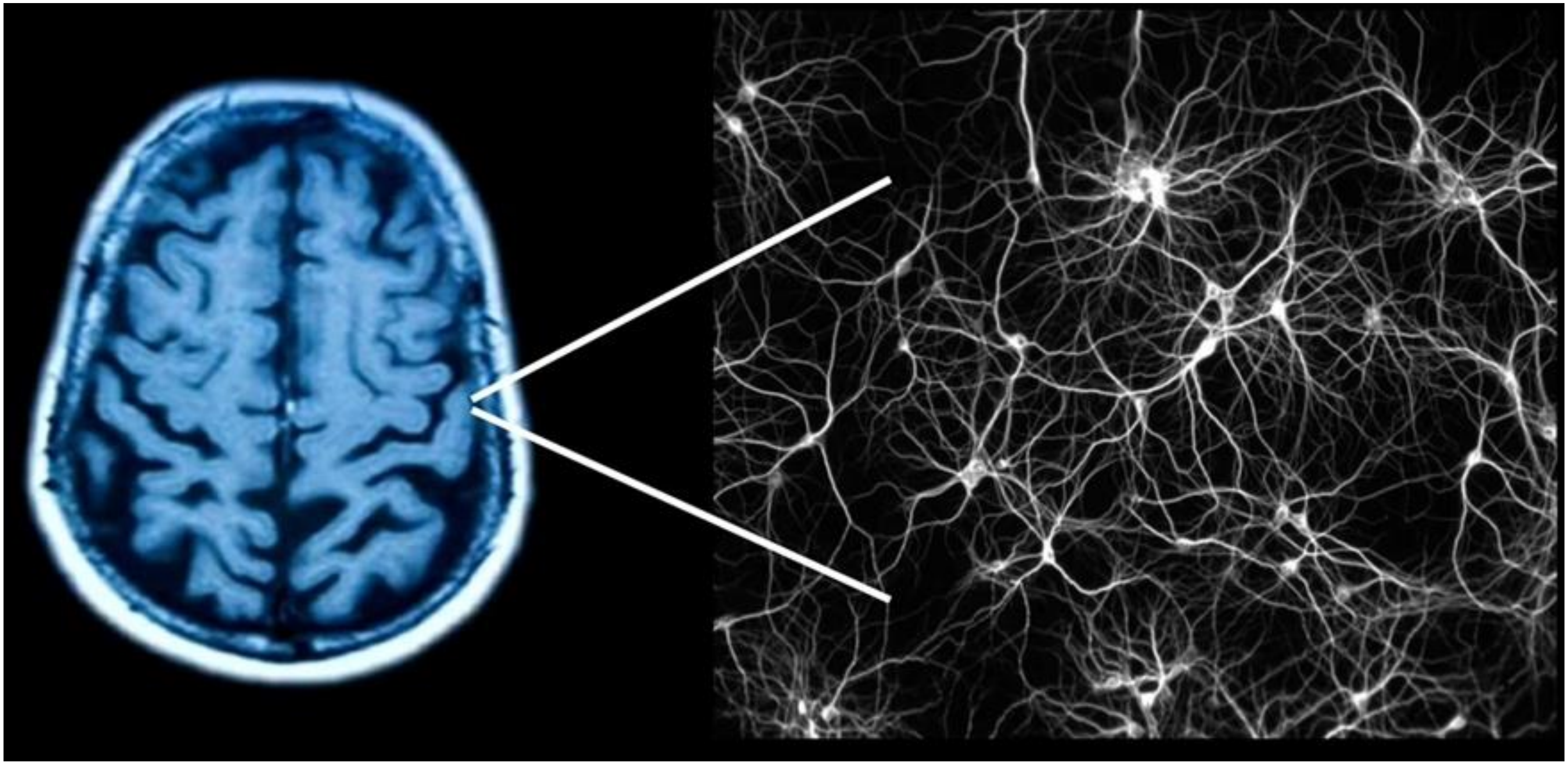
$$\mathrm{k}_{\mathrm{MLE}} = argmax_k \, P(z|k)$$

Wähle die Klasse k, bei der die vorliegende Beobachtung z am wahrscheinlichsten ist.

Also: MLE ist ein Spezialfall von MAP bei uniformer Klassenverteilung.

Log-Likelihood: Die Entscheidung anhand der sog. Log-Likelihood $\log(P(z|k))$ führt zur gleichen Klasse, da ‚log' eine streng monotone Funktion ist.

Das biologische Vorbild „Gehirn“



Lighting up the Brain | Adam Cohen | TEDxCambridge

Das biologische Vorbild „Gehirn“

- 2 Prozent des Körpergewichts, 25 Prozent der Körperenergie
- Neuron als elementare Verarbeitungseinheit,
- insgesamt ca. 10^{10} Neuronen, ca. 10.000 Verbindungen/Neuron
- → 10^{14} Verbindungen (sog. Synapsen)

- [Bei der Geburt sind bereits alle Neuronen vorhanden]
- Lernen und Vergessen bedeuten **Veränderung von Verbindungen**

Aber: Viele Funktionen des menschlichen Gehirns sind noch nicht verstanden ...

<u>Großprojekte</u>

- Blue Brain Projekt (Schweiz, USA): Nachbau eines Säugetiergehirns (zunächst Ratte) auf neuronaler Ebene mit realistischen Neuronenmodellen
- Kartierung und Nachbau: Human Brain Project (Europa), Brain Activity Map Project (USA), Human Connectome Project (USA)

Das biologische Vorbild „Gehirn“

Computer	Gehirn
von-Neumann-Architektur	Parallelverarbeitungssystem
Wenige Prozessoren	Milliarden von Prozessoren
Stärke: Bearbeitung von mathematischen Rechenaufgaben	Stärke: Wiedererkennen einer Person (auch leicht verändert oder auf einem Foto)

Das biologische Vorbild „Gehirn“

Reizverarbeitung im Gehirn (vereinfacht)

- Rezeptoren: nehmen Reize auf, erzeugen elektro-chemische Signale = ähnlich Sensoren in der Technik (Licht, Druck, Temp., Ton, ...)

↓

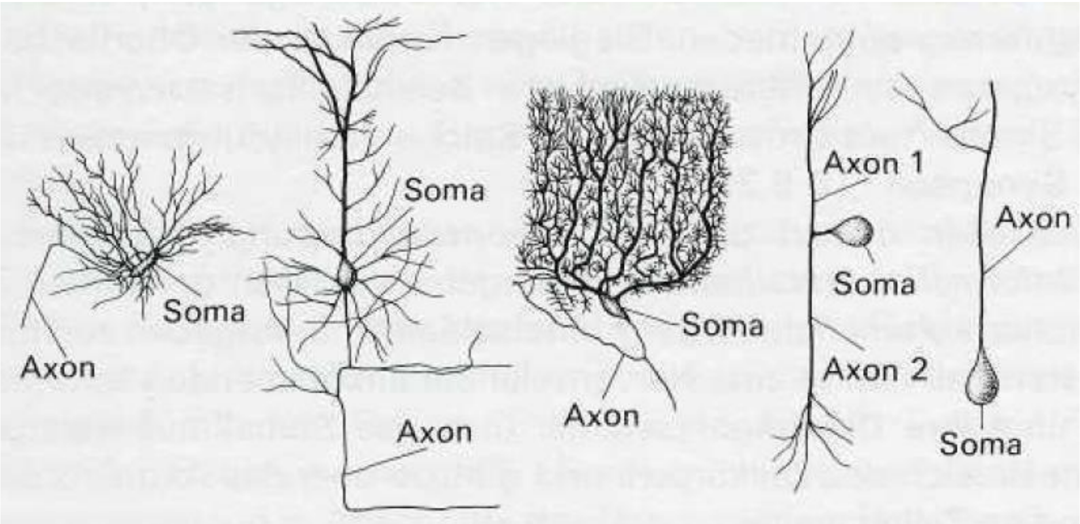
- Verarbeitung der Reize in Nervenknoten (bspw. Gehirn)

↓

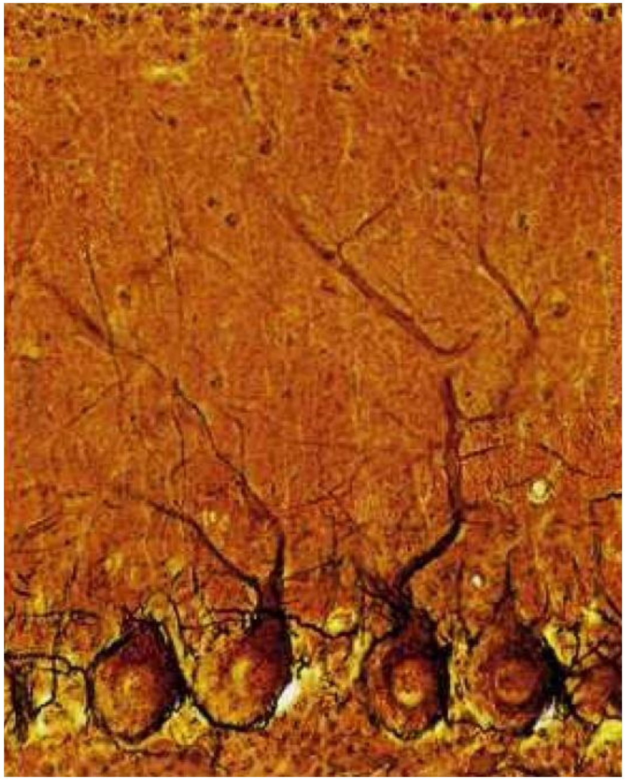
- Effektoren (angesteuerte Gewebeteile, z.B. Muskeln, Drüsen) = Aktoren in der Technik (Motor, Licht, Pumpe, ...)

Sie kennen EVA

Reale Nervenzellen



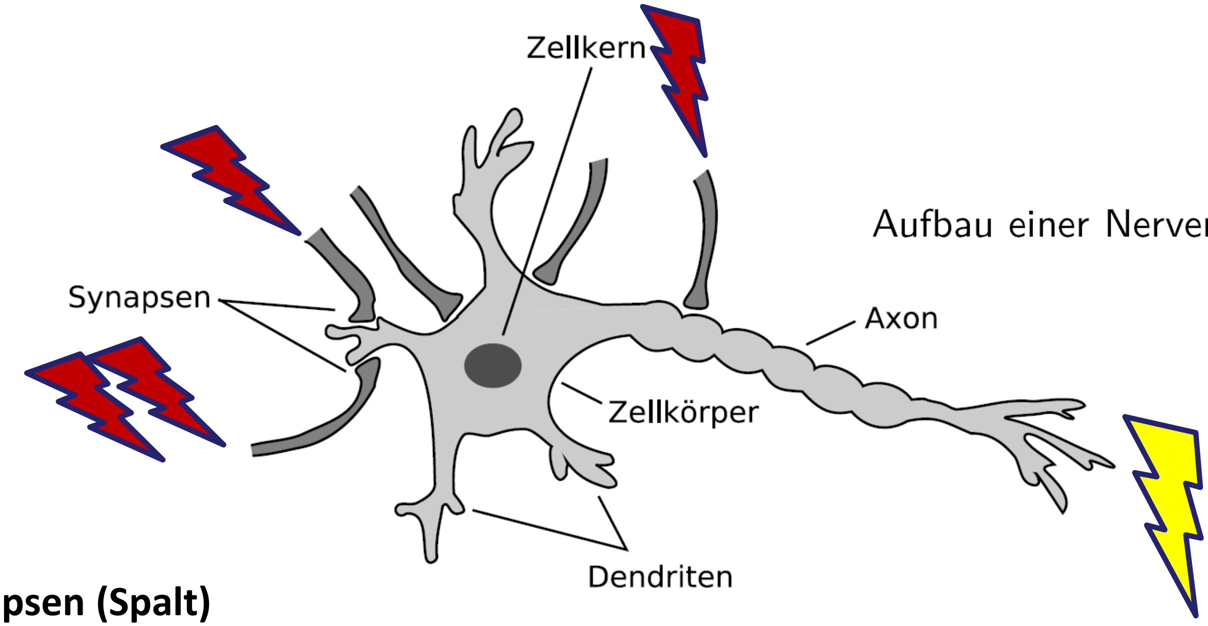
Verschiedene Typen von Nervenzellen.
Von links nach rechts: Korbzelle aus dem Kleinhirn, große Pyramidenzelle aus der Großhirnrinde, Purkinje-Zelle aus dem Kleinhirn, bipolare Schaltzelle und monopolare Nervenzelle
(Soma = Zellkörper)



Nervenzellen aus dem Kleinhirn einer Katze

Quelle: Anatomische und physiologische Grundlagen des Verhaltens. Lehr- und Arbeitsbuch Sek2, B. Löwe, W. D. Thiel, J. Prantl, 1994

Aufbau eines Neurons



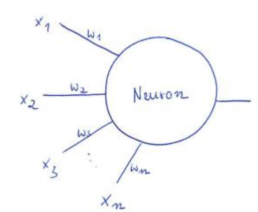
Synapsen (Spalt)

- können ankommende Potentialwerte (elektro-chem. Reize) verstärken oder hemmen
- können diese Wirkung im Laufe der Zeit verändern

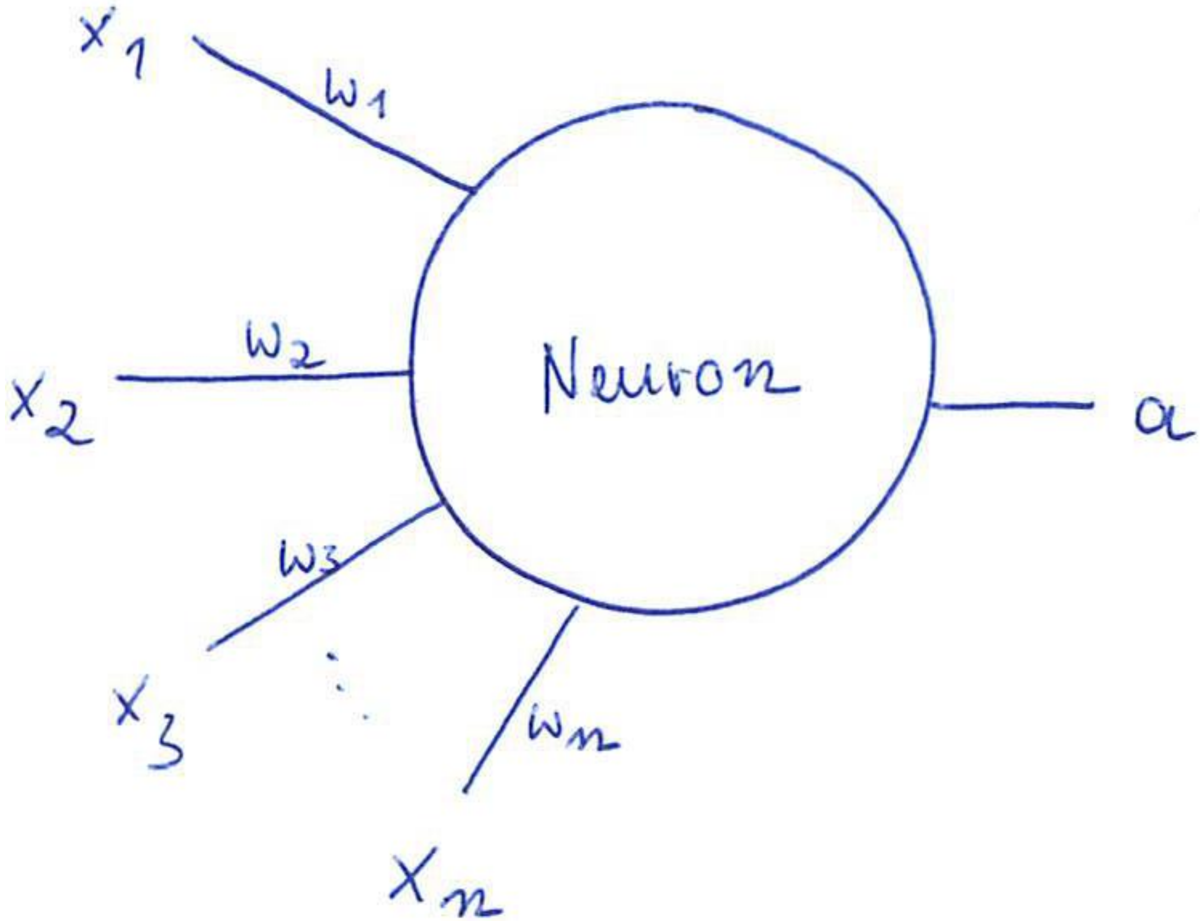
Arbeitsweise Neuon:

- Wenn die Summe der Eingabewerte als elektrisches Potential einen Schwellwert überschreitet, wird das Neuron aktiv - es „feuert“

- Siehe bspw. EEG im Biosignal-Labor
- Zeit für Video von Adam Cohen?



Vereinfachte Vorstellung



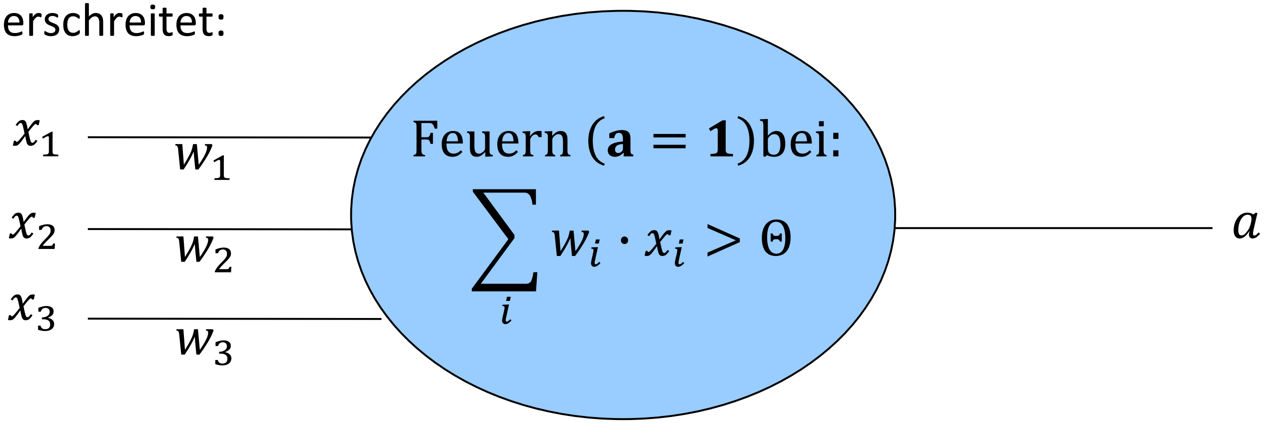
Das mathematische Modell eines Neurons

- Neuronen-Zustand $\boldsymbol{a}$: 0 oder 1 (das Neuron „feuert“ oder „feuert nicht“
- Eingabewerte $\boldsymbol{x}$ sind 0 oder 1
- Synapsen verstärken oder hemmen = Multiplikation mit positiven oder negativen Zahlen $\boldsymbol{w}$ (sog. Wichtungen)

1 Netzaktivität $\boldsymbol{net}$:

$$net = \sum_i w_i \cdot x_i = w_1 x_1 + w_2 x_2 + \cdots + w_n x_n$$

2 Das Neuron „feuert“, wenn die Netzaktivität $\boldsymbol{net}$ einen Schwellwert (Theta $\boldsymbol{\Theta}$) überschreitet:



Schwellwert als Wichtung eines ON-Neurons

Vereinfachung, um den Schwellwert nicht immer extra betrachten zu müssen:

Statt zu testen, ob die Summe den Schwellwert Θ überschreitet, zählt man $-\Theta$ zu den Gewichten und vergleicht die dann entstehende Summe mit 0:

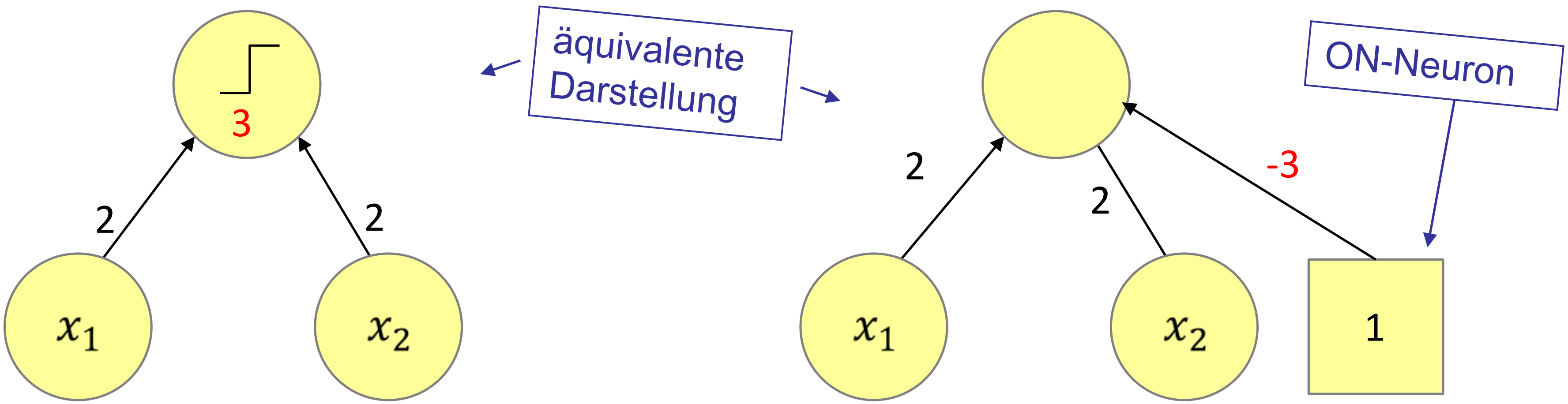
Die Bedingung zum Feuern (dass also a = 1 wird): $\sum_i w_i \cdot x_i > \Theta$

ist äquivalent zu: $\sum_i w_i \cdot x_i - \Theta > 0$

Dann können wir Theta als Wichtung eines „Extra-Neurons“ interpretieren:

Schwellwert als Wichtung eines ON-Neurons

- BIAS: Zusatzeingabe mit Wert 1 und dem Gewicht $-\Theta$ (Minus! Theta)
- Beispiel: Neuron mit zwei Eingängen x1 und x2



äquivalente Darstellung

$$2 \cdot x_1 + 2 \cdot x_2 > 3$$

$$2 \cdot x_1 + 2 \cdot x_2 + -3 \cdot 1 > 0$$

$\mathrm{Bias} = \mathrm{w}_3 = -3$ und $\mathrm{ON} - \mathrm{Neuron}\ \mathrm{x}_3 = 1$

Definition 1: Neuron (Perzeptron)

Es seien $x_1, x_2, x_3, \ldots\ x_n$ Eingangswerte von der Größe 0 oder 1. Zudem seien die Synapsenwerte $w_1, w_2, w_3, \ldots\ w_n$ beliebige reelle Zahlen (Gewichte) und

$$net = \sum_i w_i \cdot x_i$$ die Netzaktivität.

Dann ist

$$a = f(net)$$

der **Ausgabewert des Neurons**, wobei die Funktion $f(net)$ definiert ist durch

$$f(net) = \begin{cases} 1 & \text{, falls } net > 0 \\ 0 & \text{, falls } net \leq 0 \end{cases}$$

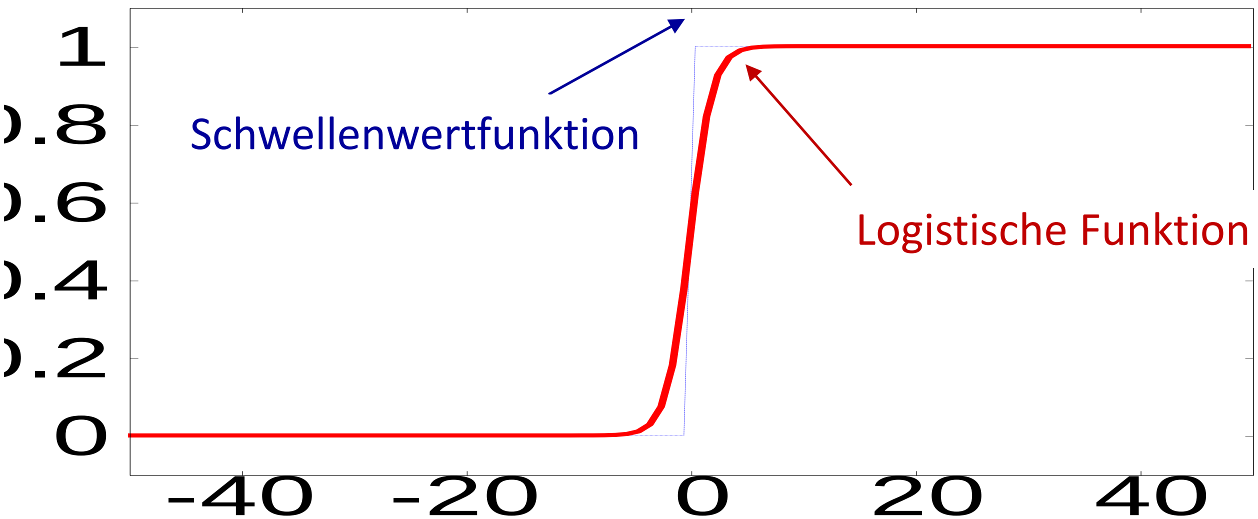
Transfer- bzw. Aktivierungsfunktion

f kann auch anders aussehen

Mögliche Transferfunktionen f(net)

1. Schwellenwertfunktion (Binäre Funktion): Abrupter Wechsel von 0 („feuert nicht") zu 1 („feuert“)
2. Logistische Funktion, eine sigmoide (=s-förmige) Funktion: Fließender Übergang, wird verwendet, wenn Differenzierbarkeit verlangt wird

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



3. Deep Learning: Rampenfunktion und Tangens hyperbolicus (tanh)

Rampenfunktion bei ReLU

Rampenfunktion bei der ReLU (**rectified linear unit**)

$$f(net) = \max(0, net) = \begin{cases} net & falls\ net > 0 \\ 0 & falls\ net \leq 0 \end{cases}$$

<u>Vorteil</u> gegenüber tanh, sigmoid:

- keine Sättigung, Gradient bleibt bei Aktivierung stark und erreicht vordere Schichten →

"Deep convolutional neural net-works with ReLUs train several times faster than their equivalents with tanh units." [NIPS2012_4824]

<u>Nachteil</u>: bei negativer Netzaktivität ist der Gradient = 0, es wird nichts gelernt. Kann das Neuron keine positive Aktivierung erreichen, ist es „tot".

<u>praktische Variante</u>: **Leaky ReLU**

$$f(net) = \begin{cases} net & falls\ net > 0 \\ 0.01 \cdot net & falls\ net \leq 0 \end{cases}$$

Rampenfunktion bei ReLU

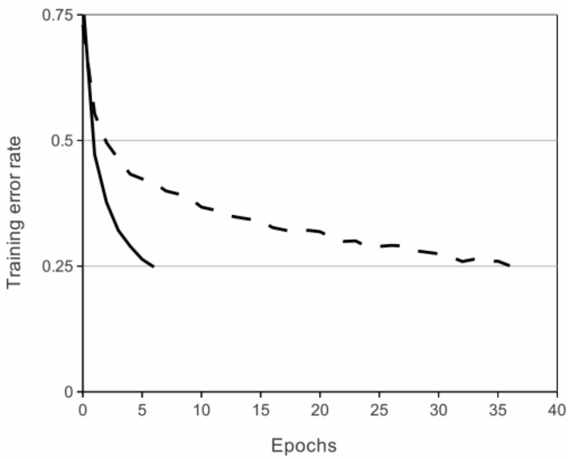
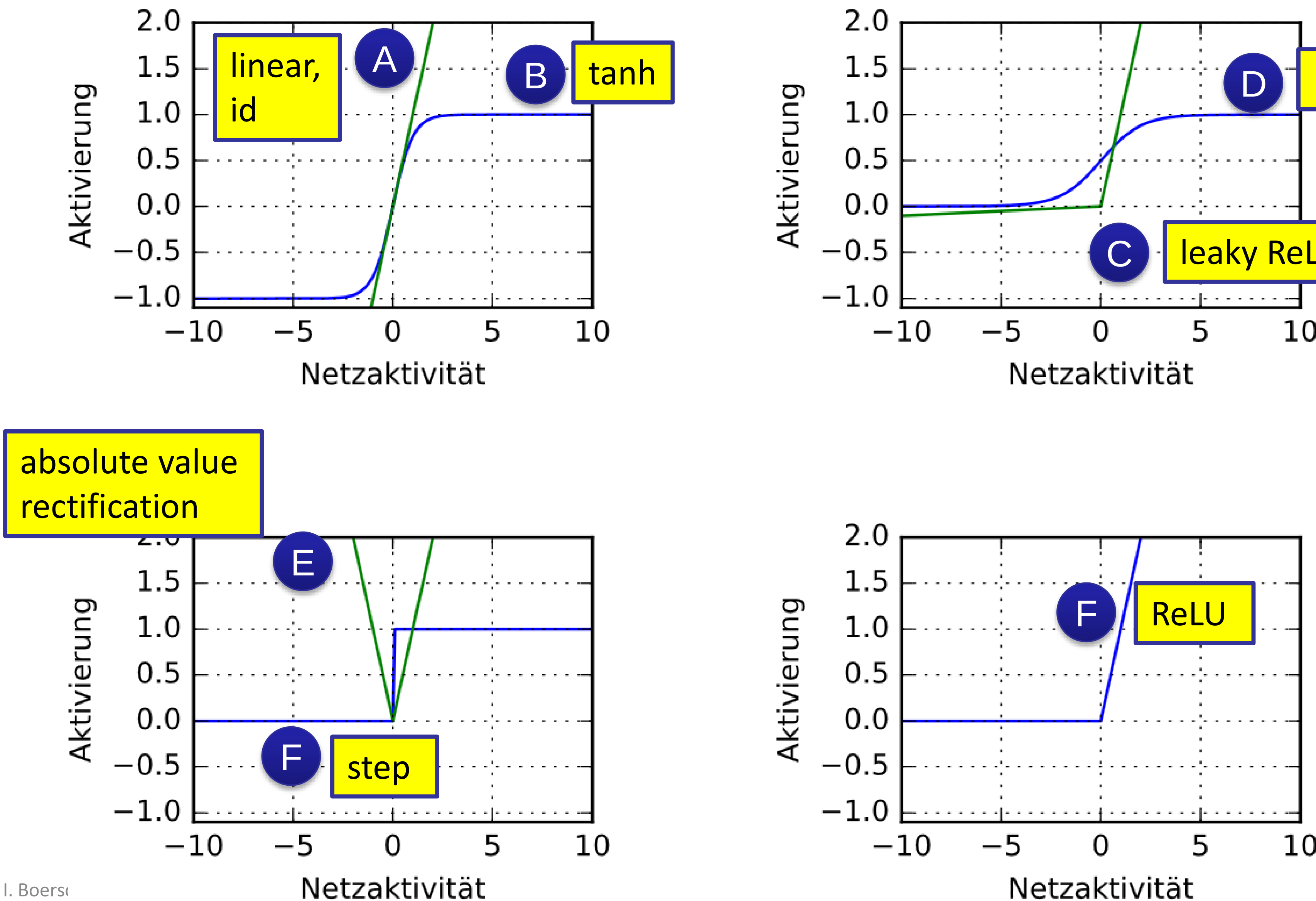


Figure 1: A four-layer convolutional neural network with **ReLUs (solid line)** reaches a 25% training error rate on CIFAR-10 six times faster than an equivalent network with tanh neurons **(dashed line)**. The learning rates for each network were chosen independently to make training as fast as possible. No regularization of any kind was employed. The magnitude of the effect demonstrated here varies with network architecture, but networks with ReLUs consistently learn several times faster than equivalents with saturating neurons.

[NIPS2012_4824] Krizhevsky, A.; Sutskever, I. & Hinton, G. E. Pereira, F.; Burges, C. J. C.; Bottou, L. & Weinberger, K. Q. *(Eds.)* **ImageNet Classification with Deep Convolutional Neural Networks**. *Advances in Neural Information Processing Systems 25, Curran Associates, Inc.,* 2012, 1097-1105

Quiz – Erkennen Sie 5 von 7?



Source Quiz

Notebook

```
%matplotlib inline

import matplotlib.pyplot as plt
import numpy as np
# Make inline plots vector graphics instead of raster graphics
from IPython.display import set_matplotlib_formats
set_matplotlib_formats('pdf', 'svg')

def pp(ax,X,y,sp):
    ax = fig.add_subplot(sp)
    ax.set_ylim([-1.1, 2])
    ax.grid(True)
    ax.plot(X,y)
    ax.set_ylabel('Aktivierung')
    ax.set_xlabel(u'Netzaktivität')

X=np.arange(-10,10,0.1)
fig = plt.figure()

y=np.tanh(X) # Tanh
pp(ax,X,y,221)

y=X # Adaline
pp(ax,X,y,221)

y=[1/(1+np.exp(-x)) for x in X] # Sigmoid
pp(ax,X,y,222)

y=[(np.sign(x)+1)/2 for x in X] # Schwellwertfunktion
pp(ax,X,y,223)

y=[np.abs(x) for x in X] # absolute value rectification
pp(ax,X,y,223)

y=[max(x,0.01*x) for x in X] # leaky ReLU
pp(ax,X,y,222)

y=[max(x,0) for x in X] # ReLU
pp(ax,X,y,224)

plt.tight_layout(pad=0.4, w_pad=10, h_pad=5.0)
plt.show()
```

Softmax

- Ausgabe einer Wahrscheinlichkeitsverteilung über Klassen
- „Wettbewerb“ zwischen den Aktivierungen
- Tipp: Lossfunktion sollte log enthalten
- Aktivierung als Exponent, dann normalisieren auf Schichtsumme Eins

$$\mathrm{softmax}(x_i) = \frac{\exp(x_i)}{\sum_j \exp(x_j)} = softmax(\boldsymbol{x} - \max_i x_i)$$

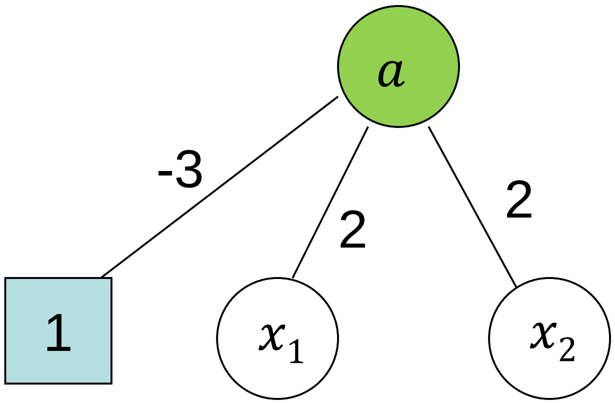
Beispiel: [0.55, 0.88, 0.06] -> [0.333, 0.463, 0.204]

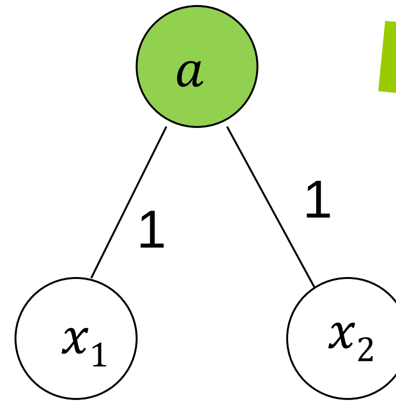
```
import numpy as np
from keras.models import Sequential
from keras.layers import Activation
model = Sequential()
model.add(Activation('softmax',input_shape=(3,)))
X = np.array([[200000000, 200000000, 200000000]])
print (model.predict(X))
X = np.array([[0.55, 0.88, 0.06]])
print (model.predict(X))
```

```
[[0.33333334 0.33333334 0.33333334]]
[[0.33293444 0.46310118 0.20396441]]
```

Beispiel: Boolesche Funktionen

UND und ?



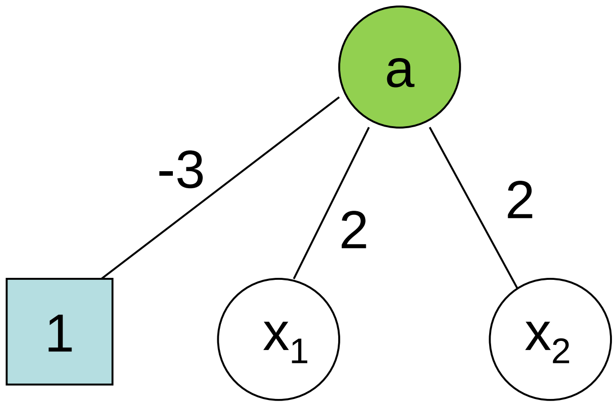


Übung

x_1	x_2	net	$a = f(net)$
0	0	-3	0
0	1	-1	0
1	0	-1	0
1	1	1	1

net	$a = f(net)$

Matrixdarstellung



$$w_{11} = 2, w_{21} = 2, w_{31} = -3,$$
$$x_1 = 1, x_2 = 0, x_3 = 1$$

$$\boldsymbol{w} = \begin{pmatrix} 2 \\ 2 \\ -3 \end{pmatrix}, \boldsymbol{x} = \begin{pmatrix} 1 & 0 & 1 \end{pmatrix}$$

$$net = \sum_i w_i x_i = w_1 x_1 + w_2 x_2 + w_3 x_3 = \boldsymbol{x} \cdot \boldsymbol{w}$$

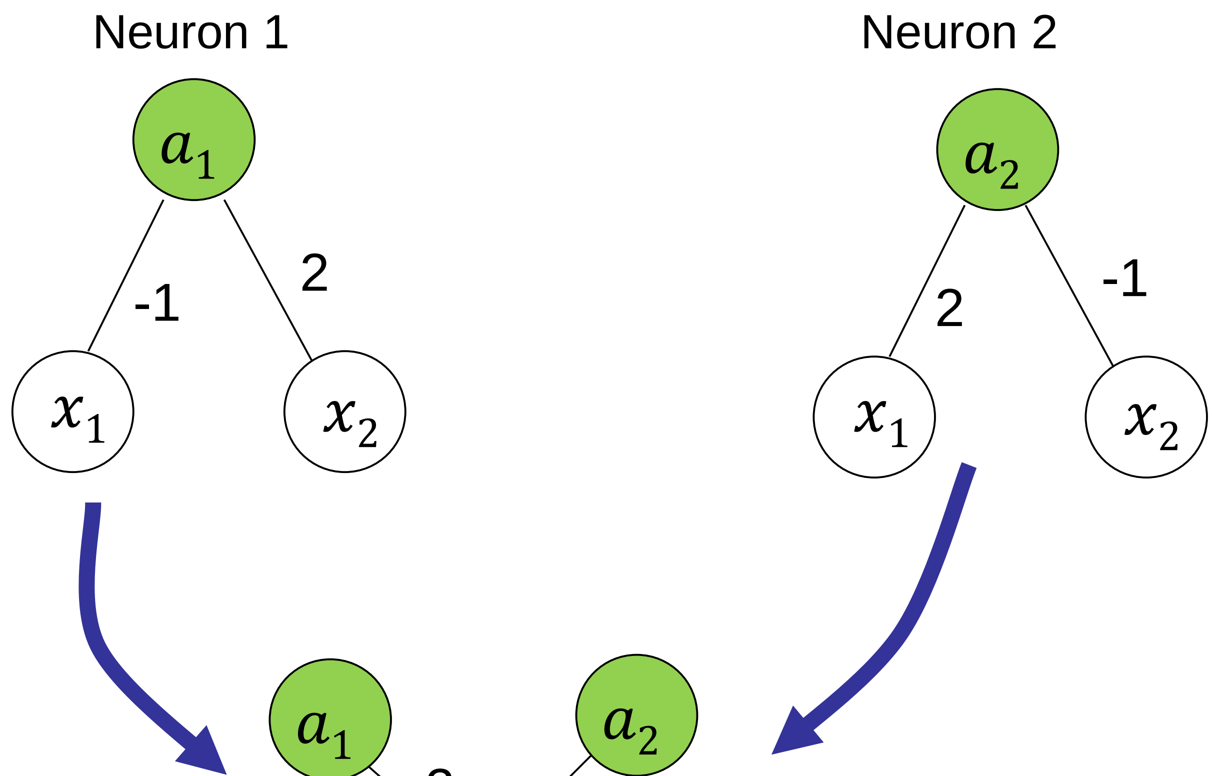
$$= \begin{pmatrix} 1 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 2 \\ 2 \\ -3 \end{pmatrix} = 1 \cdot 2 + 0 \cdot 2 + 1 \cdot -3 = -1$$

Skalarprodukt als Spezialfall des Matrixproduktes

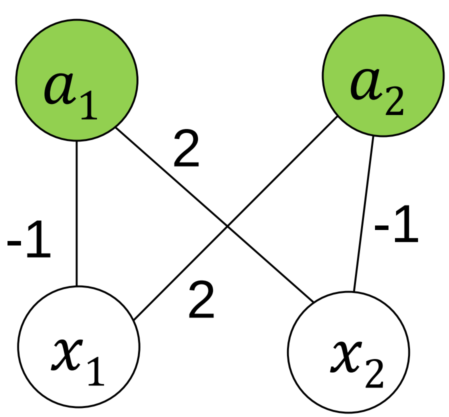
Wir haben jetzt ein formales Modell eines einzelnen Neurons (sog. Perzeptron) und seines Verhaltens.

Künstliche Neuronen werden zu leistungsfähigen Netzen, wenn sie in großer Anzahl zusammengeschaltet werden.

Beispiel: Zusammenschalten von Neuronen



Die Schwellwerte sind hier 0.

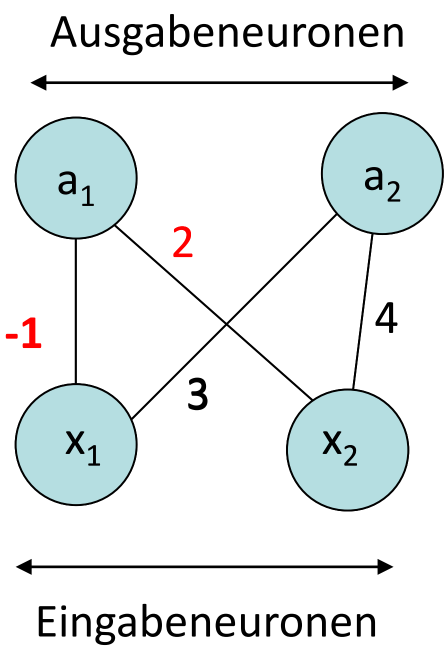


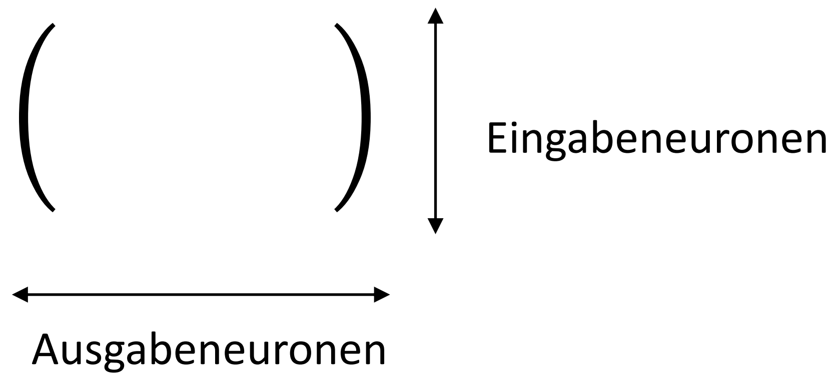
Ein neuronales Netz mit **einer** Wichtungsschicht = **einstufiges Netz**

Reaktion des Netzes:

$\boldsymbol{x_1}$	$\boldsymbol{x_2}$	$\boldsymbol{a_1}$	$\boldsymbol{a_2}$
0	0	0	0
0	1	1	0
1	**0**	**0**	**1**
1	1	1	1

Wichtungsmatrix





Wichtung w_{ij} führt vom Neuron i zum Neuron j:

$$w_{11} = -1, w_{12} = 3, w_{21} = 2, w_{22} = 4$$

oder als Wichtungsmatrix

$$\boldsymbol{W} = \begin{pmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \end{pmatrix} = \begin{pmatrix} -1 & 3 \\ 2 & 4 \end{pmatrix}$$

Und mit Matrizen dargestellt:

Sei $\boldsymbol{x} = (x_1 \quad x_2), \boldsymbol{a} = (a_1 \quad a_2)$ sowie Wichtungsmatrix $\boldsymbol{W} = \begin{pmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \end{pmatrix}$,

dann gilt:

$$\begin{aligned} \boldsymbol{net} &= \boldsymbol{x} \cdot \boldsymbol{W} \\ \boldsymbol{a} &= f(\boldsymbol{net}) \end{aligned}$$

Hier ist $\boldsymbol{net}$ der Vektor der Netzaktivitäten und $f(\boldsymbol{net})$ der Vektor:

$$f(\boldsymbol{net}) = (\, f(net_1) \quad f(net_2)\,)$$

f ist eine „vektorielle Funktion", d.h. *elementweise* anwenden

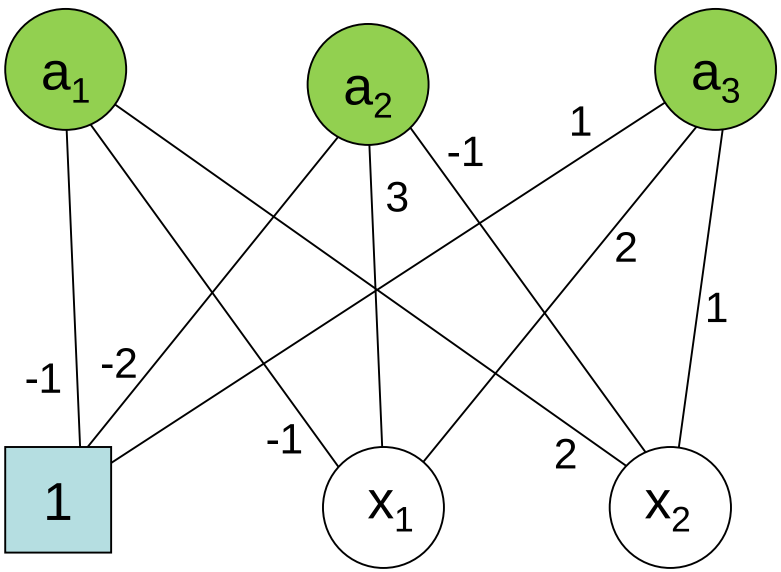
Für das Beispiel: $\boldsymbol{x} = (1 \quad 0)$ und $\boldsymbol{W} = \begin{pmatrix} -1 & 3 \\ 2 & 4 \end{pmatrix}$

$$\boldsymbol{net} = \boldsymbol{x} \cdot \boldsymbol{W} = (1 \quad 0) \cdot \begin{pmatrix} -1 & 3 \\ 2 & 4 \end{pmatrix} = (-1 \quad 3)$$

$$\boldsymbol{a} = f(\boldsymbol{net}) = (\, f(-1) \quad f(3)\,) = (0 \quad 1)$$

Übung

Ein einstufiges neuronales Netz mit zwei Eingabe- und drei Ausgabeneuronen



Welche logischen Funktionen sind dargestellt?

x_1	x_2	a_1	a_2	a_3
0	0			
0	1			
1	0			
1	1			

Übung

Definition 2: Neuronales Netz (einstufig)

Synapsen

Ein (einstufiges) neuronales Netz ist gegeben durch eine $n \times m$-Matrix $\boldsymbol{W}$, deren Elemente reelle Zahlen sind, sowie durch eine **vektorielleTransferfunktion** $\boldsymbol{f}$, so dass jedem Inputvektor $\boldsymbol{x}$ ein Outputvektor $\boldsymbol{a}$ zugeordnet wird entsprechend der Vorschrift:

$$\boxed{\begin{aligned} \boldsymbol{net} &= \boldsymbol{x} \cdot \boldsymbol{W} \\ \boldsymbol{a} &= \boldsymbol{f}(\boldsymbol{net}) \end{aligned}}$$

Zwei äquivalente Schreibweisen

M Eingabeneuronen, H versteckte Neuronen, $W = \begin{pmatrix} w_{11} & \cdots & w_{1H} \\ \vdots & \ddots & \vdots \\ w_{M1} & \cdots & w_{MH} \end{pmatrix}$

$\boldsymbol{x}$ ist Zeilenvektor ($1 \times m$-Matrix)

- $\boldsymbol{h} = \boldsymbol{x} \cdot \boldsymbol{W}$ ergibt neuen Zeilenvektor h
- MLP: $f(f(f(\boldsymbol{x} \cdot \boldsymbol{W_1}) \cdot \boldsymbol{W_2}) \cdot \boldsymbol{W_3})$
- Vorteil: keine Transponierten nötig, häufig

häufiger in dieser LV

$\boldsymbol{x}$ ist Spaltenvektor ($m \times 1$-Matrix)

- $\boldsymbol{h} = \boldsymbol{W}^T \cdot \boldsymbol{x}$ ergibt neuen Spaltenvektor h
- MLP: $f(\boldsymbol{W_3^T} \cdot f(\boldsymbol{W_2^T} \cdot f(\boldsymbol{W_1^T} \cdot \boldsymbol{x})))$
- Lässt man bei dieser Schreibweise das Transponieren weg, muss klargestellt werden, dass ein Gewicht w_{ij} vom Neuron j zum Neuron i führt.

- Umwandeln der Schreibweisen mit: $(\boldsymbol{A} \cdot \boldsymbol{B})^T = \boldsymbol{B}^T \cdot \boldsymbol{A}^T$

Zwei äquivalente Schreibweisen - Beispiel

M Eingabeneuronen, H versteckte Neuronen, $W = \begin{pmatrix} w_{11} & \cdots & w_{1H} \\ \vdots & \ddots & \vdots \\ w_{M1} & \cdots & w_{MH} \end{pmatrix}$

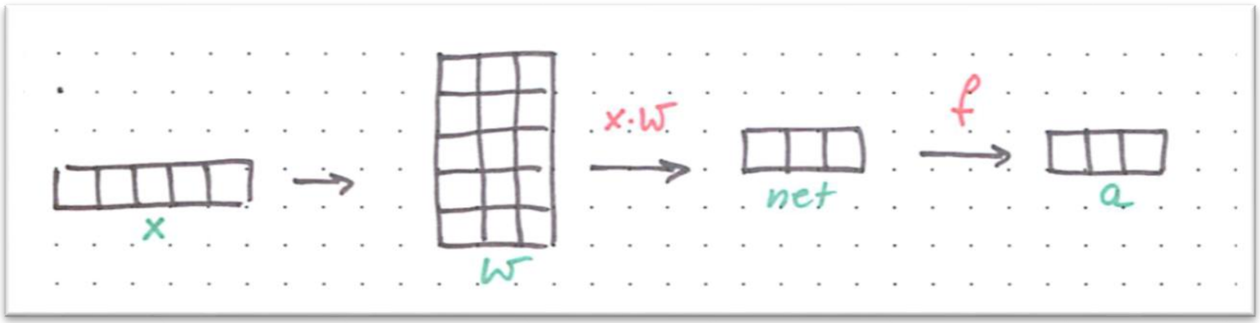
$\boldsymbol{x}$ ist Zeilenvektor ($1 \times m$-Matrix)

$$\boldsymbol{h} = \boldsymbol{x} \cdot \boldsymbol{W} = \begin{pmatrix} 1 & 2 & 3 \end{pmatrix} \begin{pmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{pmatrix} = \begin{pmatrix} 22 & 28 \end{pmatrix}$$

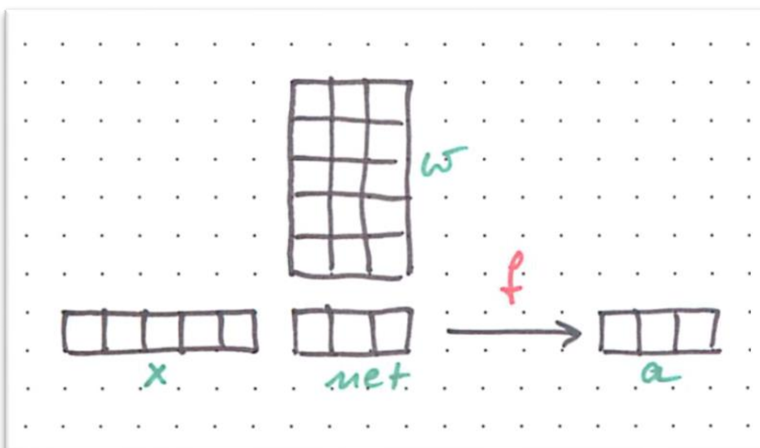
$\boldsymbol{x}$ ist Spaltenvektor ($m \times 1$-Matrix)

$$\boldsymbol{h} = \boldsymbol{W}^T \cdot \boldsymbol{x} = \begin{pmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{pmatrix}^T \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix} = \begin{pmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{pmatrix} \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix} = \begin{pmatrix} 22 \\ 28 \end{pmatrix}$$

Intuition MLP - $\boldsymbol{x}$ ist Zeilenvektor



oder kurz:



Etwas zur Geschichte

- 1969: M. Minsky, S. Papert veröffentlichen das Buch ***Perceptrons***, in dem sie nachweisen, dass es wichtige logische Funktionen gibt, die mit einstufigen Netzen nicht beschreibbar sind, z.B.: XOR.

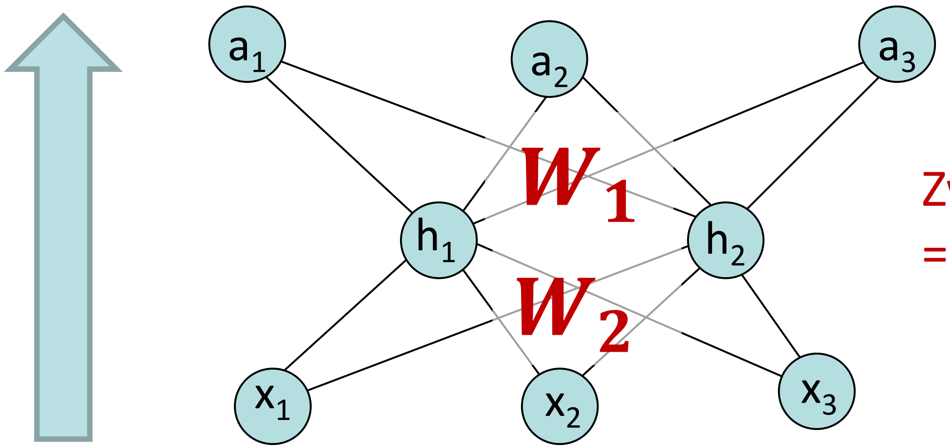
- Niedergang der NN-Forschung, da für Anwendungen nicht mehr interessant, denn ganze Funktionsklassen sind nicht modellierbar

- >10 Jahre später: Entdeckung, dass diese Aussage für mehrstufige Netze nicht gilt

- Seit 1985 gibt es einen geeigneten Lernalgorithmus (Backpropagation-Algorithmus)

> mehrstufige Netze?

Beispiel: Ein zweistufiges neuronales Netz

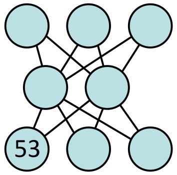


Zweistufig (2 Wichtungsschichten)
= Dreischichtig (3 Neuronenschichten)

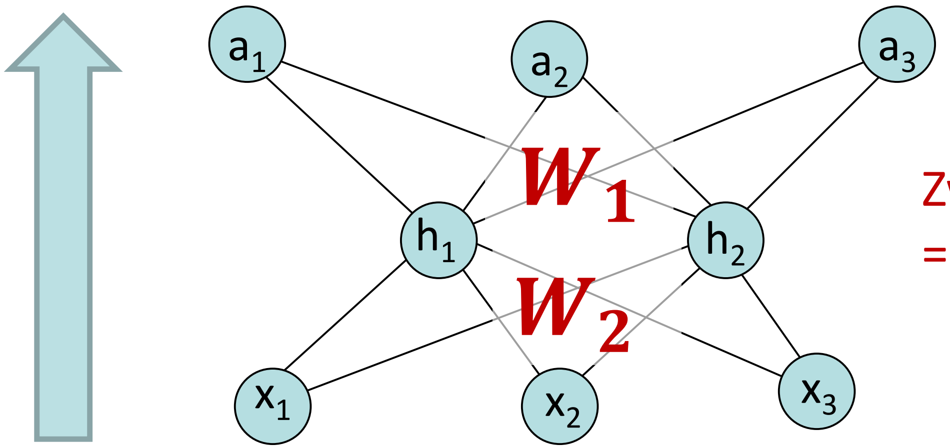
Multilayerperzeptron (kurz MLP):

Die Ausgabe der einen Schicht ist die Eingabe in die nächste Schicht.

Man unterscheidet dann Eingabeneuronen, Ausgabeneuronen und **versteckte** Neuronen (hidden neurons)



Beispiel: Ein zweistufiges neuronales Netz

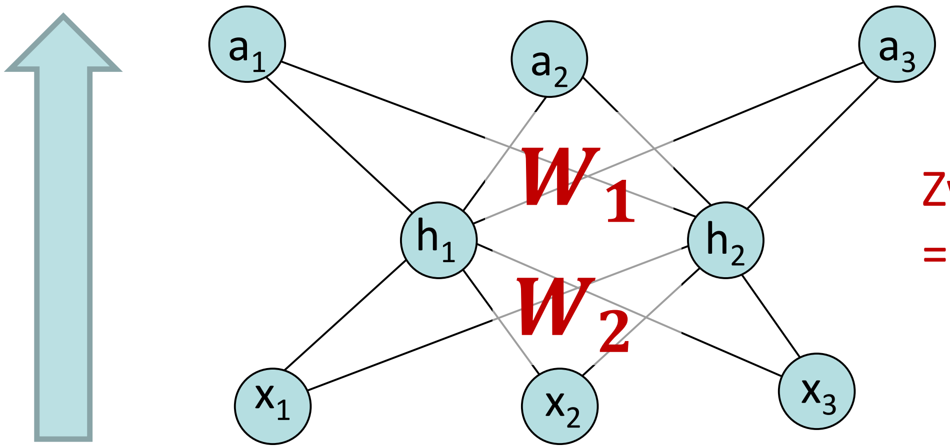


Zweistufig (2 Wichtungsschichten)
= Dreischichtig (3 Neuronenschichten)

Multilayerperzeptron (kurz MLP)

- Eingabewerte $x_1, x_2, x_3 \rightarrow$ Vektor $\boldsymbol{x}$
- Ausgabewerte $a_1, a_2, a_3 \rightarrow$ Vektor $\mathbf{a}$
- Versteckte Neuronen h_1, h_2 (engl. hidden neurons, hidden layer) $\rightarrow$ Vektor $\mathbf{h}$
- Wichtungsmatrizen $\boldsymbol{W_1}$ und $\boldsymbol{W_2}$

Beispiel: Ein zweistufiges neuronales Netz



Zweistufig (2 Wichtungsschichten)
= Dreischichtig (3 Neuronenschichten)

Multilayerperzeptron (kurz MLP)

- Die Aktivierung $\boldsymbol{h}$ der versteckten Neuronen erhält man durch:

$$\boldsymbol{net_1} = \boldsymbol{x} \cdot \boldsymbol{W_1} \text{ und } \boldsymbol{h} = f(\boldsymbol{net}_1)$$

- Die zweite Stufe wird beschrieben durch:

$$\boldsymbol{net_2} = \boldsymbol{h} \cdot \boldsymbol{W_2} \text{ und } \boldsymbol{a} = f(\boldsymbol{net_2})$$

Komplexe logische Funktionen

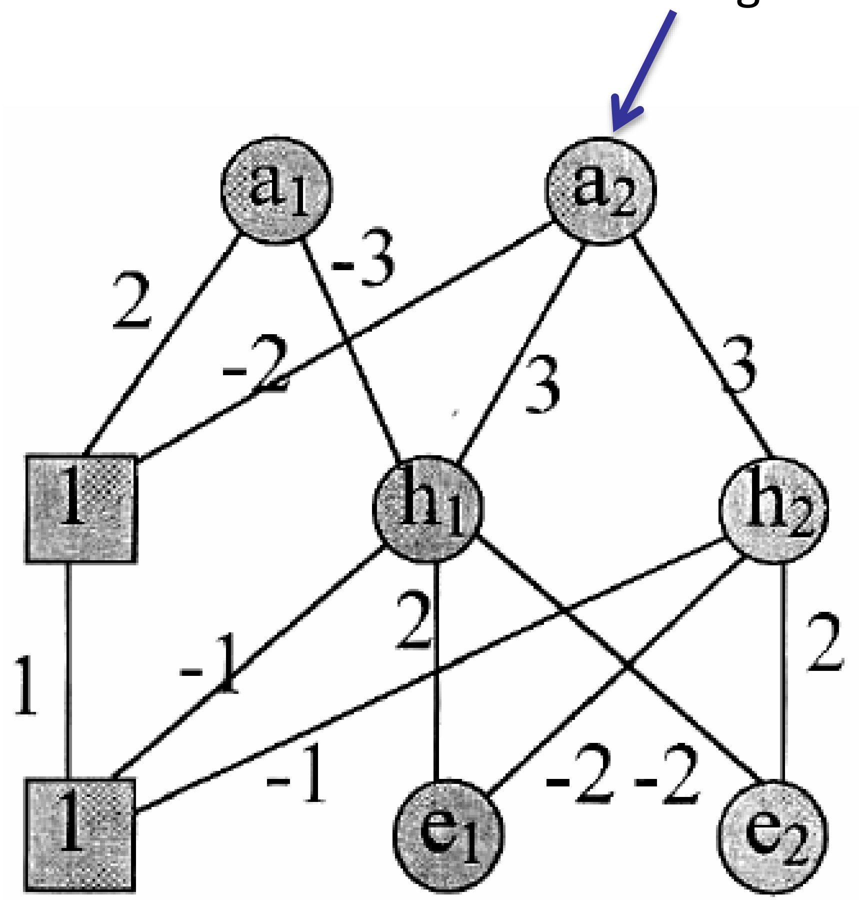
Somit zwei Arten des Zusammenschaltens:

1. Mehrere Neuronen lassen sich zusammenschalten, so dass man mehrere Ausgabekanäle erhält.

2. Mehrschichtige Netze. Die Ausgabe der einen Schicht ist die Eingabe in die darüber liegende Schicht. Einige Funktionen lassen sich nicht mit 2 Neuronen-Schichten realisieren (XOR).

Komplexe logische Funktionen

Ein zweistufiges neuronales Netz kann die XOR-Funktion darstellen, hier Ausgabe a2.



Definition 3: Mehrstufiges neuronales Netz

Es sei $\boldsymbol{x}$ ein Eingabevektor und $\boldsymbol{a}$ ein Ausgabevektor sowie $\boldsymbol{h_1}, \boldsymbol{h_2}, \ldots$ Hilfsvektoren.

Es sei f eine Transferfunktion und $\boldsymbol{W_1}, \boldsymbol{W_2}, \boldsymbol{W_3} \ldots$ Matrizen.

Dann berechnet ein **n-stufiges neuronales Netz** den Ausgabevektor $\boldsymbol{a}$ aus dem Eingabevektor $\boldsymbol{x}$ wie folgt:

$$\boldsymbol{h_1} = f(\boldsymbol{x} \cdot \boldsymbol{W_1})$$

$$\boldsymbol{h_2} = f(\boldsymbol{h_1} \cdot \boldsymbol{W_2})$$

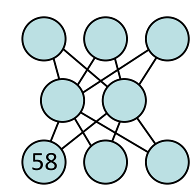
$$\boldsymbol{h_3} = f(\boldsymbol{h_2} \cdot \boldsymbol{W_3})$$

$$\ldots$$

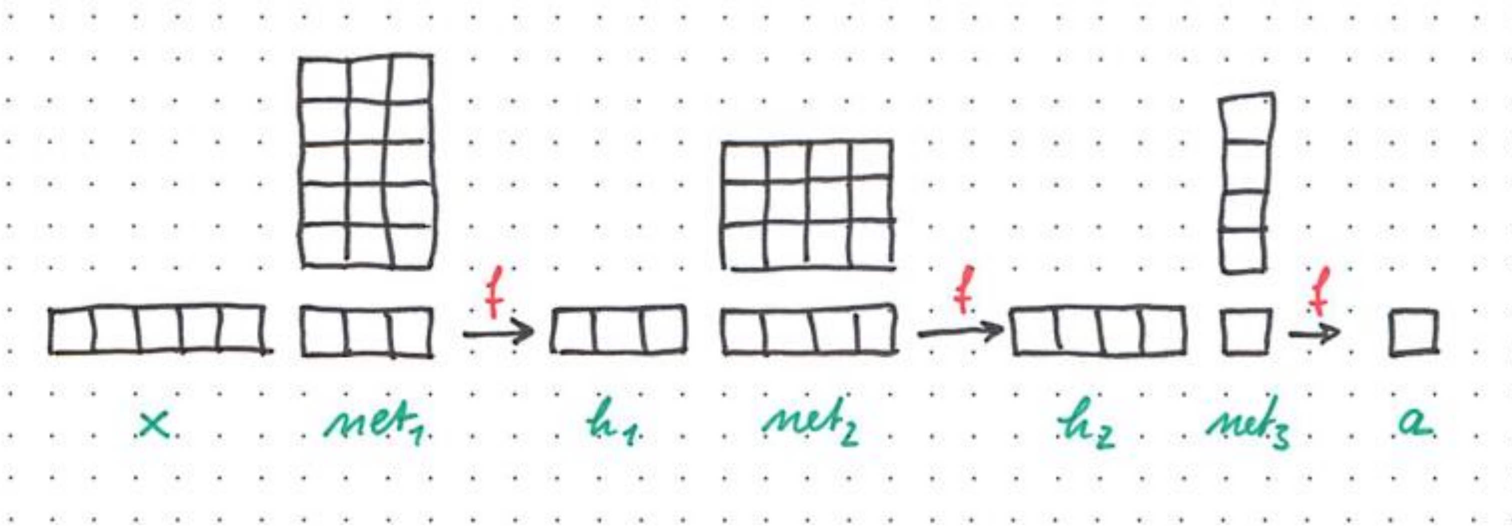
$$\boldsymbol{a} = f(\boldsymbol{h_{n-1}} \cdot \boldsymbol{W_n})$$

(Die Schwellwerte sind in den Matrizen enthalten, wenn man z.B. x_1 auf 1 einfriert – Kennen Sie schon).

Die Vektoren $\boldsymbol{h_1}, \boldsymbol{h_2}, \boldsymbol{h_3} \ldots$ bilden die verborgenen Schichten (hidden layer).



5-3-4-1

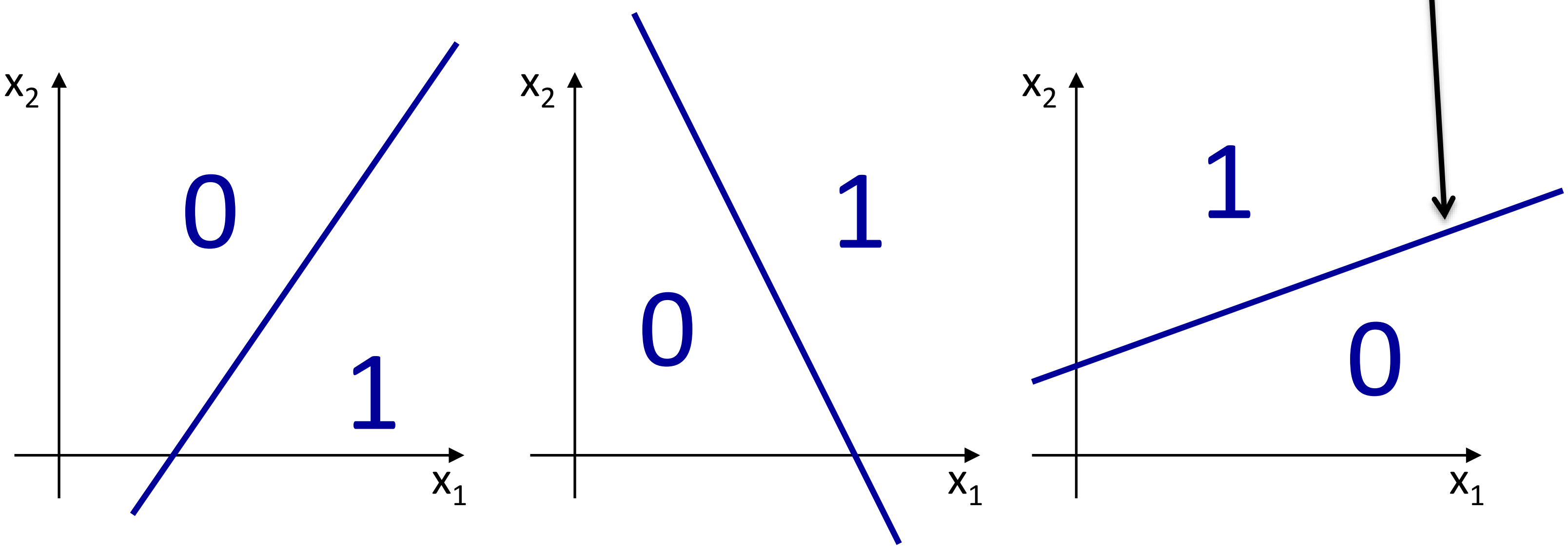


- Wie viele Eingabeneuronen? 5
- wie viele Ausgabeneuronen? 1
- wie viele versteckte Schichten? 2
- Wie viele Neuronen in den versteckten Schichten? [3, 4]
- Wie viele Wichtungen? 5x3 + 3x4 + 4x1=31

Von der logischen Funktion zum Perzeptron

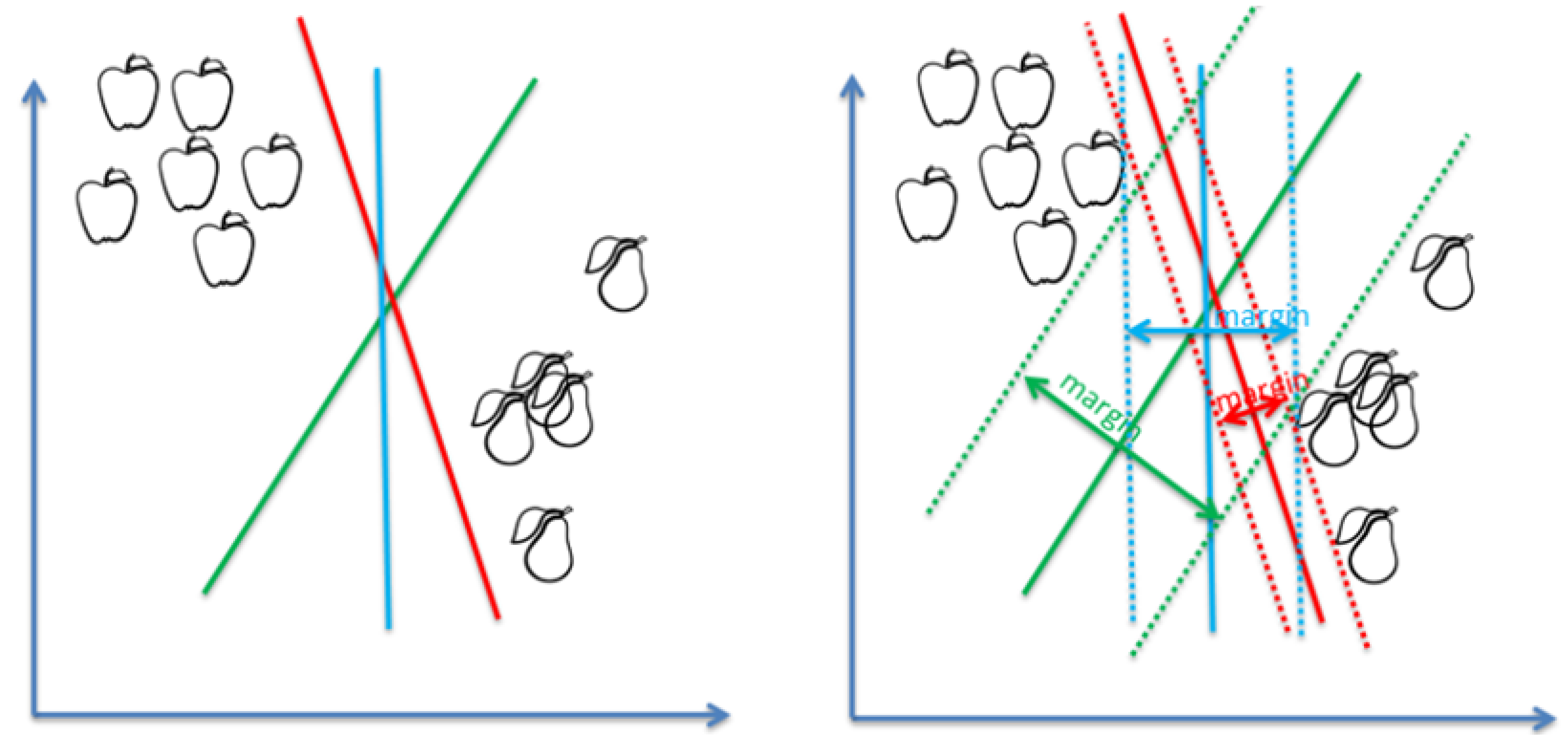
Die Wichtungen eines Perzeptrons beschreiben die Lage der Trenngerade.

Für einige Muster aus dem Eingaberaum **soll** das Perzeptron mit 0, für andere mit 1 reagieren:



Wie wählen wir die Wichtungen, so dass **net>0** (= das Neuron feuert), genau in dem 1-Teilraum gilt?

Perzeptron vs. SVM



While the Perceptron classifier will find any of the seperating hyperplanes, a SVM classifier will find one with the maximum margin (indicated here with green).

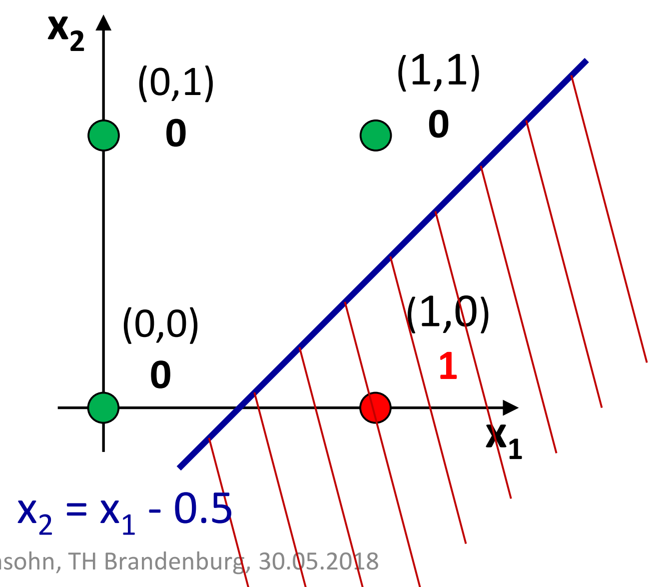
Quelle: http://ataspinar.com/2016/12/22/the-perceptron/

Von der logischen Funktion zum Perzeptron

Gegeben sei:

x_1	x_2	a
0	0	0
0	1	0
1	0	1
1	1	0

Darstellung im Eingaberaum:



1. Gerade so wählen, dass sie die Ausgabe-werte 0 und 1 trennt, z.B. $x_2 = x_1 - 0.5$

2. Ungleichung des „feuernden“ Teilraumes mit Hilfe der Geraden:

$$a = 1 \text{ falls } x_2 < x_1 - 0.5$$

3. Ungleichung umformen in **net > 0**

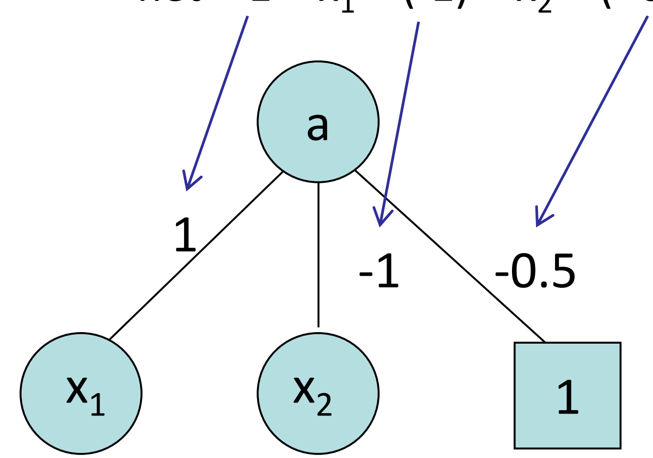
hier beide Seiten $-x_2$

-> $a = 1$ falls $x_1 - x_2 - 0.5 > 0$

-> $\text{net} = x_1 - x_2 - 0.5$!!

4. Wichtungen ablesen

$$\text{net} = 1 * x_1 + (-1) * x_2 + (-0.5) * 1$$



Übung

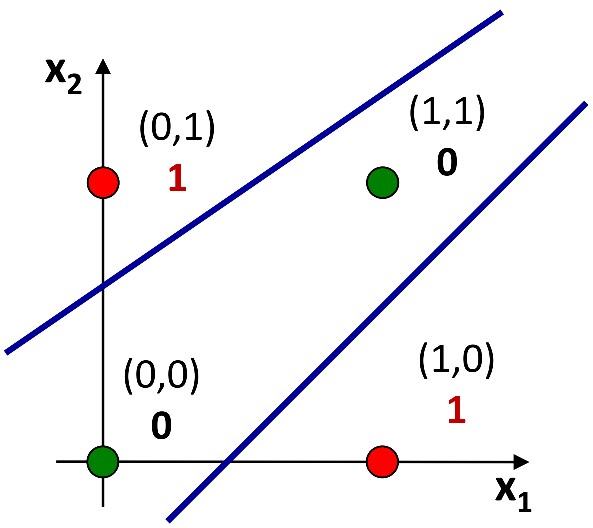
Wann reicht ein einstufiges Netz nicht mehr?

Gegeben sei:

XOR-Funktion

x_1	x_2	a
0	0	0
0	1	1
1	0	1
1	1	0

Darstellung im Eingaberaum:



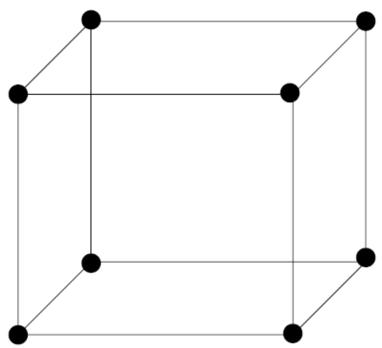
Bei XOR wären zwei Geraden notwendig!

Also :

Für XOR gibt es kein einstufiges Netz, da ein einstufiges Netz nur eine lineare Ungleichung auswerten kann.

Darstellbarkeit bei 3 und n Eingabeneuronen

Eingaberaum bei 3 Eingabebits:



Eingabevektoren = Eckpunkte eines Würfels

lineare Teilbarkeit = Beschreibung der Grenzfläche durch eine Ebene ist möglich

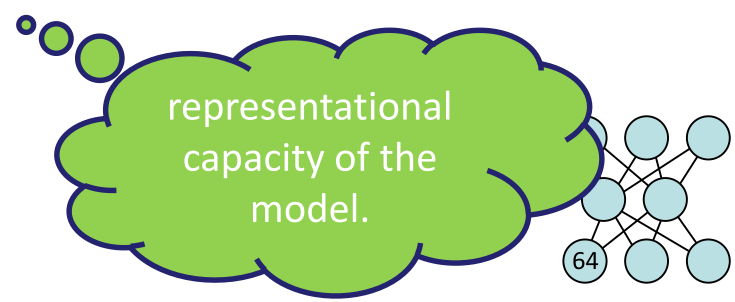
Allgemein bei n Eingabeneuronen:

- n Eingabebits
- n-dimensionaler Raum
- (n-1)-dimensionale Hyperebene teilt die Eingabevektoren in 2 Gruppen, die auf 1 bzw. 0 abgebildet werden.

Ausdruckskraft, Repräsentationsfähigkeit

Ein **einstufiges** neuronales Netz kann nur linear teilbare Funktionen darstellen.

Ein **zweistufiges** neuronales Netz darstellen kann jede beliebige Funktion darstellen.



Wozu eine nichtlineare Aktivierungsfunktion?

M Eingabeneuronen, H versteckte Neuronen, $W = \begin{pmatrix} w_{11} & \cdots & w_{1H} \\ \vdots & \ddots & \vdots \\ w_{M1} & \cdots & w_{MH} \end{pmatrix}$

3-stufiges MLP: $f(\,f(\,f(\,\boldsymbol{x} \cdot \boldsymbol{W_1}) \cdot \boldsymbol{W_2}) \cdot \boldsymbol{W_3})$

Ist f linear, so ist ein mehrstufiges Netz eine lineare Funktion seiner Eingabe und kann nur linear separierbare Funktionen repräsentieren.

Alle Wichtungsmatrizen können zu einer Matrix $\boldsymbol{W}$‘ zusammengefasst werden.

Eine Schicht:

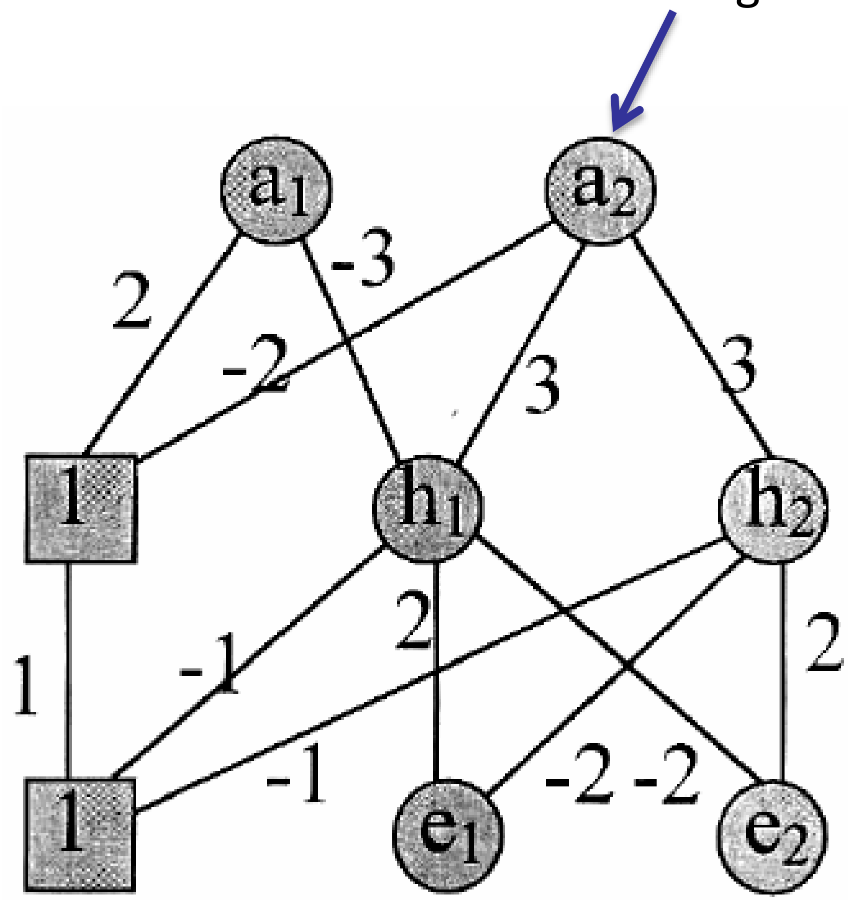
$$\boldsymbol{a} = f(\boldsymbol{net}) = \boldsymbol{net} \cdot \boldsymbol{L} = (\boldsymbol{x} \cdot \boldsymbol{W}) \cdot \boldsymbol{L} = \boldsymbol{x} \cdot (\boldsymbol{W} \cdot \boldsymbol{L}) = \boldsymbol{x} \cdot \boldsymbol{W'}$$ (sog. Madaline)

Drei Schichten:

$$\boldsymbol{a} = f(\,f(\,f(\,\boldsymbol{x} \cdot \boldsymbol{W_1}) \cdot \boldsymbol{W_2}) \cdot \boldsymbol{W_3}) = \boldsymbol{x} \cdot (\boldsymbol{W_1} \cdot \boldsymbol{L} \cdot \boldsymbol{W_2} \cdot \boldsymbol{L} \cdot \boldsymbol{W_3} \cdot \boldsymbol{L}) = \boldsymbol{x} \cdot \boldsymbol{W'}$$

Komplexe logische Funktionen

Ein **zweistufiges** neuronales Netz kann die XOR-Funktion darstellen, hier Ausgabe a2.




```
# XOR mit ON-Neuron als e0, 4 Samples
X = [ [1,0,0],[1,0,1],[1,1,0],[1,1,1] ]
# Neuron a2 in der Abbildung, 4 Samples
y = [[0],[1],[1],[0]]
# Wichtungen w_ij
W1 = [[1,-1,-1], [0,2,-2],[0,-2,2]]
W2 = [[-2],[3],[3]]

h_net = np.dot(X,W1)              # Netzaktivität h
h = np.sign(h_net)*0.5+0.5        # Sprungfunktion
a_net = np.dot(h,W2)              # Netzaktivität a
a = np.sign(a_net)*0.5+0.5        # Sprungfunktion
delta = (y - a)  # Fehler
```

```
X [[1, 0, 0], [1, 0, 1], [1, 1, 0], [1, 1, 1]]
y [[0], [1], [1], [0]]
W1 [[1, -1, -1], [0, 2, -2], [0, -2, 2]]
W2 [[-2], [3], [3]]
h_net [[ 1 -1 -1]
 [ 1 -3  1]
 [ 1  1 -3]
 [ 1 -1 -1]]
h [[ 1.  0.  0.]
 [ 1.  0.  1.]
 [ 1.  1.  0.]
 [ 1.  0.  0.]]
a_net [[-2.]
 [ 1.]
 [ 1.]
 [-2.]]
a [[ 0.]
 [ 1.]
 [ 1.]
 [ 0.]]
delta [[ 0.]
 [ 0.]
 [ 0.]
 [ 0.]]
```

AND

1. Spalte: ON-Neuron in h

kein Fehler

Komplexe logische Funktionen

Ein **zweistufiges** neuronales Netz kann die XOR-Funktion darstellen.

```
# XOR mit ON-Neuron als e0, 4 Samples
X = [ [1,0,0],[1,0,1],[1,1,0],[1,1,1] ]
# Neuron a2 in der Abbildung, 4 Samp
y = [[0],[1],[1],[0]]
# Wichtungen w_ij
W1 = [[1,-1,-1], [0,2,-2],[0,-2,2]]
W2 = [[-2],[3],[3]]

h_net = np.dot(X,W1)            # Netzaktivität h
h = np.sign(h_net)*0.5+0.5     # Sprungfunktion
a_net = np.dot(h,W2)           # Netzaktivität a
a = np.sign(a_net)*0.5+0.5     # Sprungfunktion
delta = (y - a)   # Fehler
```

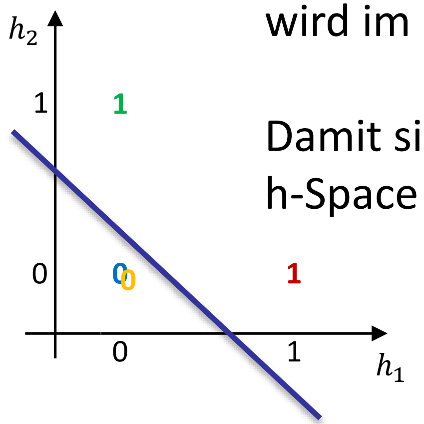
Eingabemuster

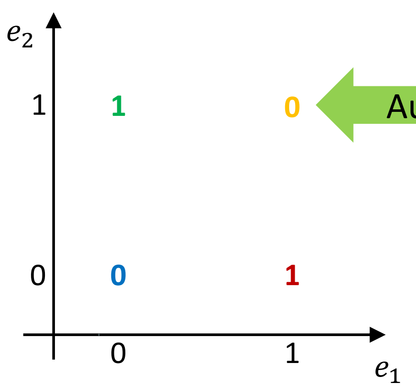
$$\boldsymbol{e} = (1{,}1)$$

wird im $\boldsymbol{h}$-Feature-Space zu

$$\boldsymbol{h} = (0{,}0)$$

Damit sind die Muster im h-Space linear trennbar





Ausgabe bei $\boldsymbol{e} = (1{,}1)$

```
X [[1, 0, 0], [1, 0, 1], [1, 1, 0], [1, 1, 1]]
y [[0], [1], [1], [0]]
W1 [[1, -1, -1], [0, 2, -2], [0, -2, 2]]
W2 [[-2], [3], [3]]
h_net [[ 1 -1 -1]
 [ 1 -3  1]
 [ 1  1 -3]
 [ 1 -1 -1]]
h [[ 1.  0.  0.]
 [ 1.  0.  1.]
 [ 1.  1.  0.]
 [ 1.  0.  0.]]
a_net [[-2.]
 [ 1.]
 [ 1.]
 [-2.]]
a [[ 0.]
 [ 1.]
 [ 1.]
 [ 0.]]
delta [[ 0.]
 [ 0.]
 [ 0.]
 [ 0.]]
```

Features h_1, h_2

logisches AND

1. Spalte: ON-Neuron in h

h(1,1) = (0,0)

a(1,1)

kein Fehler

Wie viele versteckte Neuronen sind nötig?

Exkurs

Die vom Netz zu adaptierende Funktion sei gegeben durch:

$$(x_1,a_1),(x_2,a_2),....,(x_n,a_n)$$

In der versteckten Schicht müssen n verschiedene Belegungen möglich sein:

Bei k hidden neurons:

$$2^k \geq n \text{ bzw. } k \geq ld(n)$$

<u>Beispiel:</u> n=17 verschiedene Muster zu lernen -> k > …

Was haben wir erreicht?

- Dreischichtige Netze (= 2 Wichtungsschichten) sind universelle Approximatoren, d.h. mit einem solchen Netz lassen sich alle binären Funktionen darstellen (mathematische Funktionen, Steueranweisungen für Roboter, Prognosen, etc.)

Wo kommen die Gewichte her? Vorschrift zur Berechnung der Gewichte w existiert für die meisten Anwendungen nicht. Lassen sich - wie beim biologischen Vorbild - die „richtigen" Gewichte erlernen?

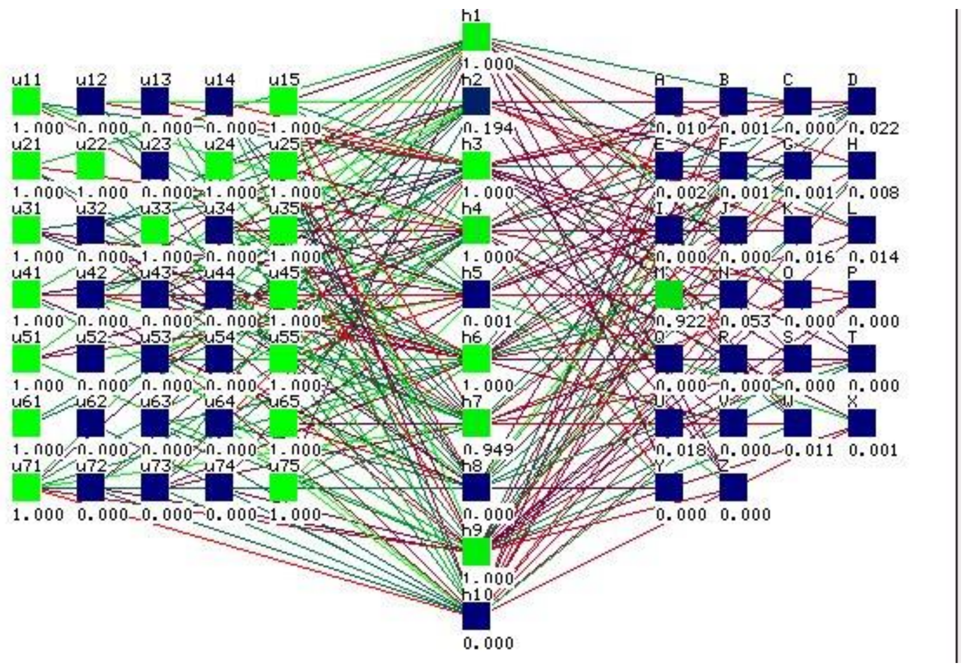
Ja, man benötigt

1. **Trainingsdaten**: Eine Menge von Vektorpaaren (Eingabevektor, gewünschter Ausgabevektor)
2. **Lernalgorithmus**, der aus Trainingsdaten und aktueller Ausgabe des Netzes die Wichtungsänderungen berechnet

Beispiel 1: Erlernen von Buchstaben (JNNS)

Trainingsdaten: Wie soll das Netz bei bestimmten Eingaben reagieren?
-> **Paare von Eingabe- und Ausgabevektoren**

- Eingabevektoren der Länge 35: eine 5x7-Grauwertmatrix
- Ausgabevektoren der Länge 26: für die 26 Buchstaben des Alphabets



Rückblick

Biologische Vorbilder Gehirn und Neuron

Wie wird ein Neuron im Computer abgebildet?

- Neuronenmodell Perzeptron
- Schwellwert als ON-Neuron
- Netzaktivität, Transferfunktion (Aktivierungsfunktion)
- Beispiel UND, ?

Welche Funktionen können ausgedrückt werden?

Wie werden Neuronen zu Netzen verbunden?

- Zusammenschalten (mehrere Ausgänge, mehrere Schichten)
- Vektordarstellung

Anwendungsbeispiele

Rückblick

Biologische Vorbilder Gehirn und Neuron

Objekterzeugungsmodell

- Bayes-optimaler Klassifikator (BOK) = MAP
- Maximum likelihood Estimation

Wie wird ein Neuron im Computer abgebildet?

- Neuronenmodell Perzeptron
- Schwellwert als ON-Neuron
- Netzaktivität, Transferfunktion (Aktivierungsfunktion)
- Softmax
- Beispiel UND, ?

Welche Funktionen können ausgedrückt werden?

Wie werden Neuronen zu Netzen verbunden?

- Zusammenschalten (mehrere Ausgänge, mehrere Schichten)
- Vektordarstellung

Anwendungsbeispiele

weiter mit: Lernen

- Lernen im Gehirn

  Die Hebbsche These

- Lernen am Perzeptron

  Zwei Phasen

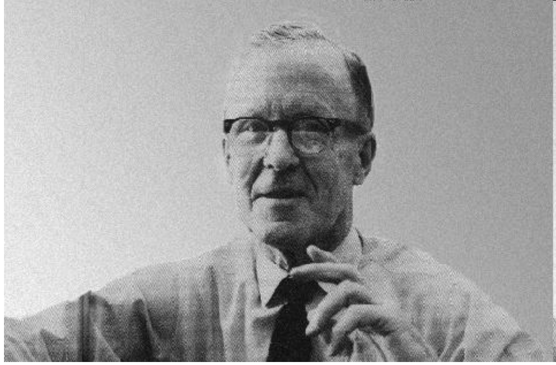
  Die Delta-Regel

- Lernen am Multilayer-Perzeptron

  Das Backpropagation-Verfahren

- Maschinelles Lernen

  Einen Schritt zurück



Wie lernt das Gehirn?

Hierzu formulierte 1949 Donald O. Hebb die Hebbsche These:

> *„When an axion of cell A is near enough to excite a cell B and repeatedly or persistently takes part in firing it, some growth process or metabolic change takes place in one or both cells such that A's efficiency, as one of the cells firing B, is increased.„*

Was bedeutet das?

- *Bei gleichzeitiger Aktivität der präsynaptischen und postsynaptischen Zelle wird die Synapse verstärkt.*
- Fire together – wire together
- neuronale Mechanismen bis heute nicht geklärt

Synaptische Plasitizität

Zwei Phasen

- Gegeben sind Beispiele einer darzustellenden (unbekannten) Funktion. Konkret liegen Eingabevektoren x vor, denen Ausgabevektoren y zugeordnet sind. Diese Funktion soll durch ein Netz dargestellt werden.
- Für das Netz ist eine Topologie zu wählen. (Heuristiken)

1. Lernphase:

- Die **Gewichte** sind so zu bestimmen, dass das Netz in der gewählten Topologie die vorgegebene Funktion darstellt
- Rechenintensiv, aber einmalig
- Minimierung einer Fehlerrate

2. Recall-Phase (Einsatzphase):

- Nachdem die Gewichte gelernt wurden, ist das Netz beliebig oft einsetzbar
- Geringer Rechenaufwand

Lernphase

Trainingsdaten sind Paare von Eingabe- und Ausgabevektoren (x,y):

- Initialisierung der Gewichte: Setze für alle Gewichte (kleine) Zufallszahlen

In der Lernphase werden dem Netz viele bekannte Zuordnungen präsentiert:

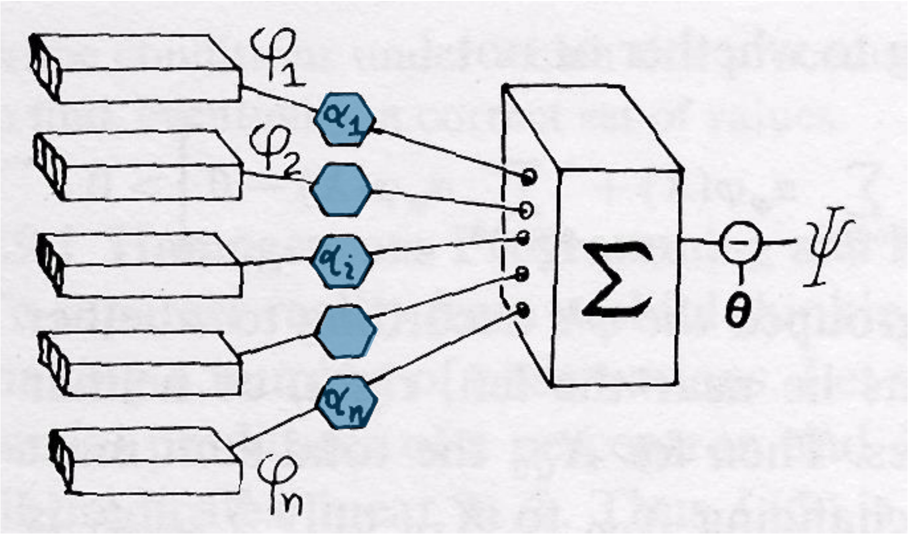
1. Wähle einen beliebigen Eingabevektor x
2. Berechne mit den momentanen Gewichten den Ausgabevektor a
3. Vergleiche den Ausgabevektor a mit dem Zielvektor y. Verwende die Abweichung von a und y zur Verbesserung der Wichtungen nach einer geeigneten Korrekturformel. Weiter bei 1.

Praktisch in der nächsten Woche …

Lernen beim Perzeptron

- Frank Rosenblatt 1958
- erstes lernfähiges Neuronenmodell - das Perzeptron

Wie sind die Wichtungen zu ändern?: Lernregel: **Delta-Regel**



Perzeptron
Quelle: Perceptrons, Marvin Minsky & Seymour Papert, 1969

Lernen beim Perzeptron – Delta-Regel

neue i-te Wichtung: $w_i(t+1) = w_i(t) + \Delta w_i$

alte Wichtung   Wichtungsänderung

Wichtungsänderung: $\Delta w_i = \eta \cdot x_i \cdot \delta$

Lernrate eta   Eingabe i   Fehler des Neurons

Fehler: $\delta = y - a$

Soll   Ist (Ausgang des Neurons)

Schwellwert hier als Wichtung: $w_{n+1} = -\Theta$ und $x_{n+1} = 1$

Der Schwellwert lernt immer.

Beispiel

$$w_i(t+1) = w_i(t) + \eta \cdot x_i \cdot (y - a)$$

gegeben ein Netz: $w_1(t) = -4,\ w_2(t) = 5,\ \Theta(t) = 2,$
d.h. als ON-Neuron $x_3 = konstant\,1,\ w_3 = -2$
zu lernen: $x_1 = 1,\ x_2 = 0,\ Sollwert\ y = 1$
Lernrate: $\eta = 0.3$

Berechnen Sie die Wichtungsänderungen!

Netzaktivität: $net = x_1 w_1 + x_2 w_2 + x_2 w_3 = 1 \cdot -4 + 0 \cdot 5 + 1 \cdot -2 = -6$
Aktivität: $a = f(net) = f(-6) = 0$
Fehler: $\delta = Soll - Ist = y - a = 1 - 0 = 1$

Wichtungsänderung 1: $\triangle\, w_1 = \eta x_1 \delta = 0.3 \cdot 1 \cdot 1 = 0.3$
neue Wichtung: $w_1(t+1) = w_1(t) + \triangle\, w_1 = -4 + 0.3 = -3.7$

analog Wichtungsänderung 2: $\triangle\, w_2 = \eta x_2 \delta = 0.3 \cdot 0 \cdot 1 = 0$
neue Wichtung: $w_2(t+1) = w_2(t) + \triangle\, w_2 = 5 + 0 = 5$
w_2 lernt nicht wegen $x_2 = 0$

analog Wichtungsänderung 3: $\triangle\, w_3 = \eta x_3 \delta = 0.3 \cdot 1 \cdot 1 = 0.3$
neue Wichtung: $w_3(t+1) = w_3(t) + \triangle\, w_3 = -2 + 0.3 = -1.7$
w_3 lernt immer wegen ON-Neuron
$\Theta(t+1) = 1.7$ Neuron feuert nun leichter

Test mit gleicher x-Eingabe führt zu $net = -3.7 - 1.7 = -5.4$, damit näher am Feuern,
$a = f(-5.4) = 0$ reicht aber noch nicht

Beispiel

$$w_i(t+1) = w_i(t) + \eta \cdot x_i \cdot (y - a)$$

Berechnen Sie die Wichtungsänderungen!

gegeben ein Netz: $w_1(t) = -4,\ w_2(t) = 5,\ \Theta(t) = 2,$
d.h. als ON-Neuron $x_3 = konstant\,1,\ w_3 = -2$
zu lernen: $x_1 = 1,\ x_2 = 0,\ Sollwert\ y = 1$
Lernrate: $\eta = 0.3$

Netzaktivität: $net = x_1w_1 + x_2w_2 + x_2w_3 = 1 \cdot -4 + 0 \cdot 5 + 1 \cdot -2 = -6$
Aktivität: $a = f(net) = f(-6) = 0$
Fehler: $\delta = Soll - Ist = y - a = 1 - 0 = 1$

Wichtungsänderung 1: $\triangle\, w_1 = \eta x_1 \delta = 0.3 \cdot 1 \cdot 1 = 0.3$
neue Wichtung: $w_1(t+1) = w_1(t) + \triangle\, w_1 = -4 + 0.3 = -3.7$

analog Wichtungsänderung 2: $\triangle\, w_2 = \eta x_2 \delta = 0.3 \cdot 0 \cdot 1 = 0$
neue Wichtung: $w_2(t+1) = w_2(t) + \triangle\, w_2 = 5 + 0 = 5$
w_2 lernt nicht wegen $x_2 = 0$

analog Wichtungsänderung 3: $\triangle\, w_3 = \eta x_3 \delta = 0.3 \cdot 1 \cdot 1 = 0.3$
neue Wichtung: $w_3(t+1) = w_3(t) + \triangle\, w_3 = -2 + 0.3 = -1.7$
w_3 lernt immer wegen ON-Neuron
$\Theta(t+1) = 1.7$ Neuron feuert nun leichter

Test mit gleicher x-Eingabe führt zu $net = -3.7 - 1.7 = -5.4$, damit näher am Feuern,
$a = f(-5.4) = 0$ reicht aber noch nicht

Lernen beim Perzeptron - Algorithmus

Das Training des Perzeptrons erfolgt nach folgendem Algorithmus:

Initialisiere alle Wichtungen und den Schwellwert mit beliebigen Werten, z. B. 0.
Wiederhole, bis alle Muster korrekt klassifiziert werden:

> *Wähle den nächsten (oder einen zufälligen) Eingabevektor x.*
> *Berechne net und die Klassifikation a.*
> *Berechne den Fehler δ zur gewünschten Ausgabe.*
> *Modifiziere Wichtungen und Schwellwert nach Gleichung (10.3) und (10.4).*

Funktioniert das sicher für die repräsentierbaren Funktionen? Ja!

Satz 10.2 (Konvergenz des Perzeptron-Lernverfahrens)
Der Perzeptron-Lernalgorithmus terminiert für linear separierbare Boolesche Funktionen.

Beispiel Lernen der ODER-Funktion

gut zum Nachrechnen

Tab. 10.2: Lernverlauf für das ODER-Prädikat

Zyklus	x_1	x_2	y	net	a	δ	Δw_1	Δw_2	$\Delta\Theta$	w_1	w_2	Θ
Init										0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0
	0	1	1	0	0	1	0	1	-1	0	1	-1
	1	0	1	0	1	0	0	0	0	0	1	-1
	1	1	1	1	1	0	0	0	0	0	1	-1
2	0	0	0	0	1	-1	0	0	1	0	1	0
	0	1	1	1	1	0	0	0	0	0	1	0
	1	0	1	0	0	1	1	0	-1	1	1	-1
	1	1	1	2	1	0	0	0	0	1	1	-1
3	0	0	0	0	1	-1	0	0	1	1	1	0
	0	1	1	1	1	0	0	0	0	1	1	0
	1	0	1	1	1	0	0	0	0	1	1	0
	1	1	1	2	1	0	0	0	0	1	1	0
4	0	0	0	0	0	**0**	0	0	0	1	1	0
	0	1	1	1	1	**0**	0	0	0	1	1	0
	1	0	1	1	1	**0**	0	0	0	1	1	0
	1	1	1	2	1	**0**	0	0	0	1	1	0

Historie

1. Delta-Regel nicht anwendbar für Multilayer-Perzeptron und
2. beschränkte Ausdruckskraft des Perzeptrons (Minsky & Papert 1969)

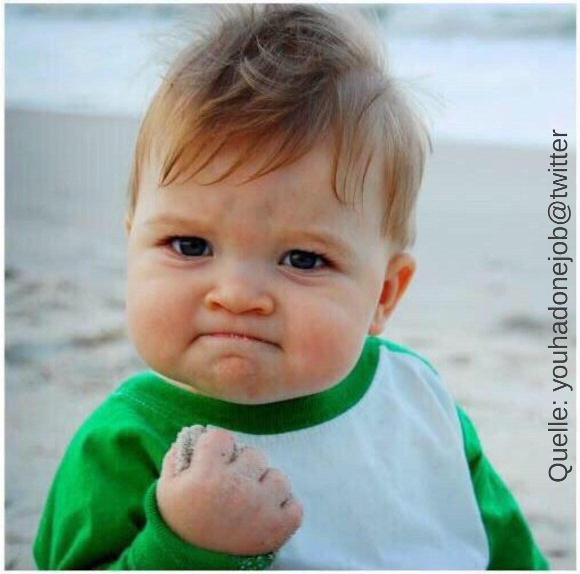
→ Stagnation der künstlichen neuronalen Netze

Wie können Multilayer-Perzeptrons lernen? **. . . ? . . . .? . .? . . . . !**

1986: Rumelhart, Hinton, Williams (und andere vor ihnen) zeigen, dass die Delta-Regel eine Form des **Gradientenabstiegs** ist und verallgemeinern zur „Generalisierten Delta-Regel“, dem sog. **Backpropagation-Verfahren**

- dafür notwendig: differenzierbare Aktivierungsfunktion

Renaissance der künstlichen neuronalen Netze.
Bis heute.



Quelle: youhadonejob@twitter