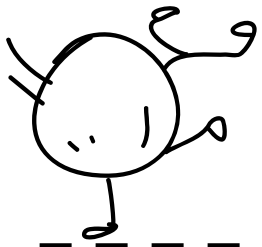


Neuronale Netze

Vorlesung 186.844

03.12.2015

TECHNISCHE
UNIVERSITÄT
WIEN
Vienna University of Technology

Überblick

I. WH: Perceptron
II. Grundlagen neuronaler Netze
III. Training neuronaler Netze
IV. Wichtige Hinweise und Tipps
V. Anwendungsbeispiele

I. Wiederholung: Perceptron

WH: Perceptron

Das **Perceptron** ist ein Trainingsalgorithmus, um für binäre Klassifikationsprobleme (zwei Klassen) eine lineare Entscheidungsgrenze zu finden.

- Annahmen: d-dimensionaler Merkmalsvektor $\boldsymbol{x} \in \mathbb{R}^d$ und zwei Klassen w_1 und w_2 → binäre Klassifikationsproblem

- Ziel: Finde eine Diskriminantenfunktion $g: \mathbb{R}^d \rightarrow \mathbb{R}$ welche die Klassenzugehörigkeit wie folgt kodiert:

$$g(\boldsymbol{x}) > 0 \; falls \; \boldsymbol{x} \in \omega_1$$
$$g(\boldsymbol{x}) < 0 \; falls \; \boldsymbol{x} \in \omega_2$$

WH: Signumfunktion

- man kann eine lineare Diskriminantenfunktion auch als Signumfunktion anschreiben:

$$o = o(\boldsymbol{x}) = sgn(\boldsymbol{w}^T\boldsymbol{x} - \theta) = \begin{cases} 1 & falls\ \boldsymbol{w}^T\boldsymbol{x} \geq \theta \\ -1 & falls\ \boldsymbol{w}^T\boldsymbol{x} < \theta \end{cases}$$

$o(\boldsymbol{x})$ ... ist die Ausgabe des Perceptrons

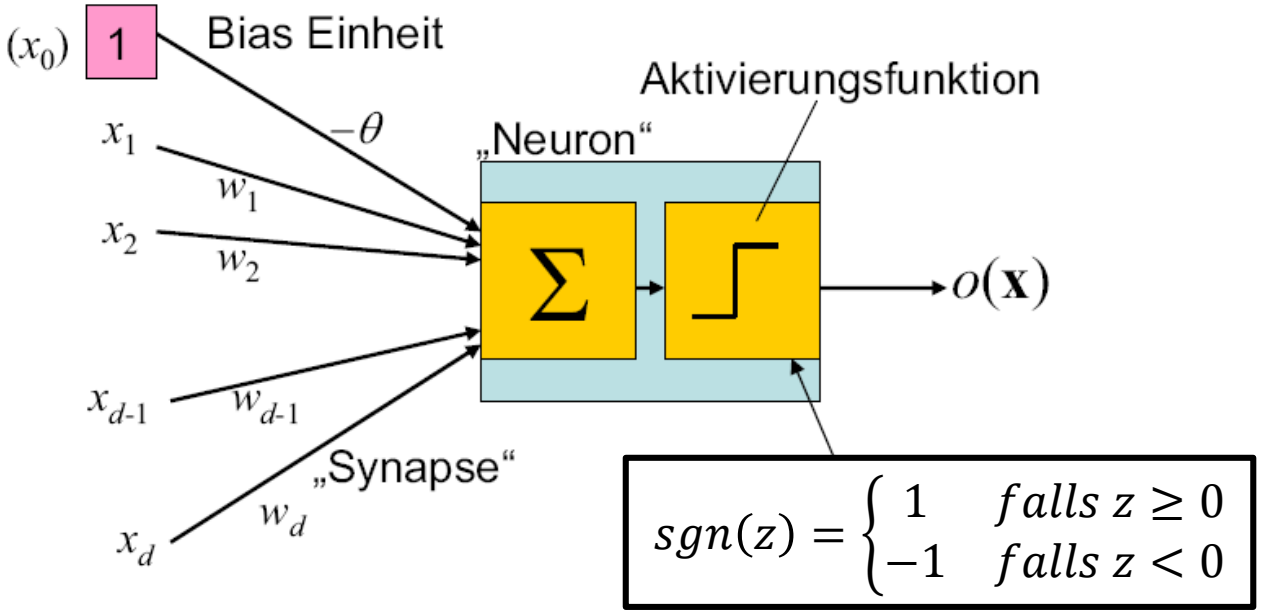
Ziel des Perceptron-Algorithmus: Bestimme einen Gewichtsvektor $\boldsymbol{w}$ und einen Bias θ, sodass

$$o(\boldsymbol{x}) = 1 \quad (\Leftrightarrow \boldsymbol{w}^T\boldsymbol{x} \geq \theta) \quad falls\ \boldsymbol{x} \in \omega_1$$
$$o(\boldsymbol{x}) = -1 \quad (\Leftrightarrow \boldsymbol{w}^T\boldsymbol{x} < \theta) \quad falls\ \boldsymbol{x} \in \omega_2$$

WH: Darstellung als Neuron

Erinnerung Signumfunktion: $o(\boldsymbol{x}) = sgn\left(\sum_{i=1}^{d} w_i x_i\right)$ — in homogenen Koordinaten

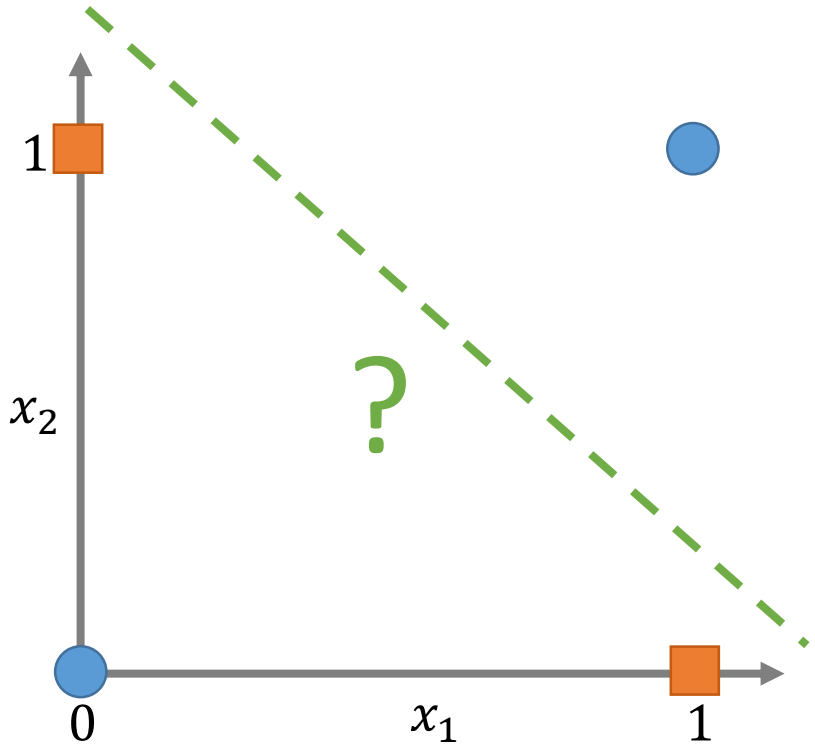
Perceptron ist eine gute Annährung an ein biologisches Neuron:



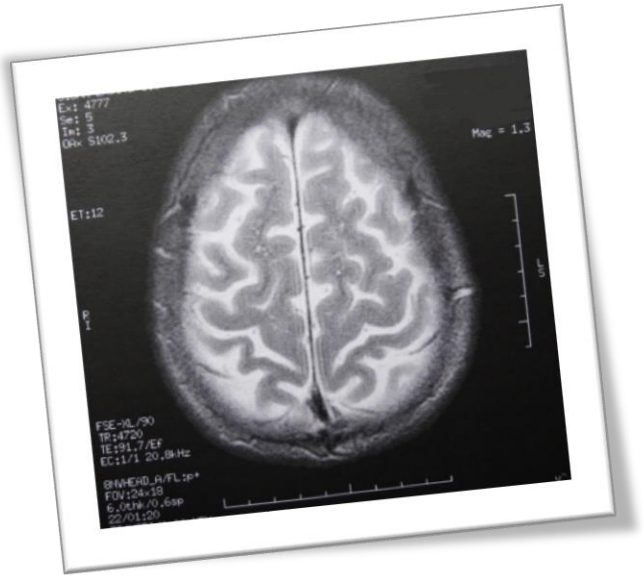
WH: Beispiel XOR

Das XOR-Problem ist nicht linear separierbar!

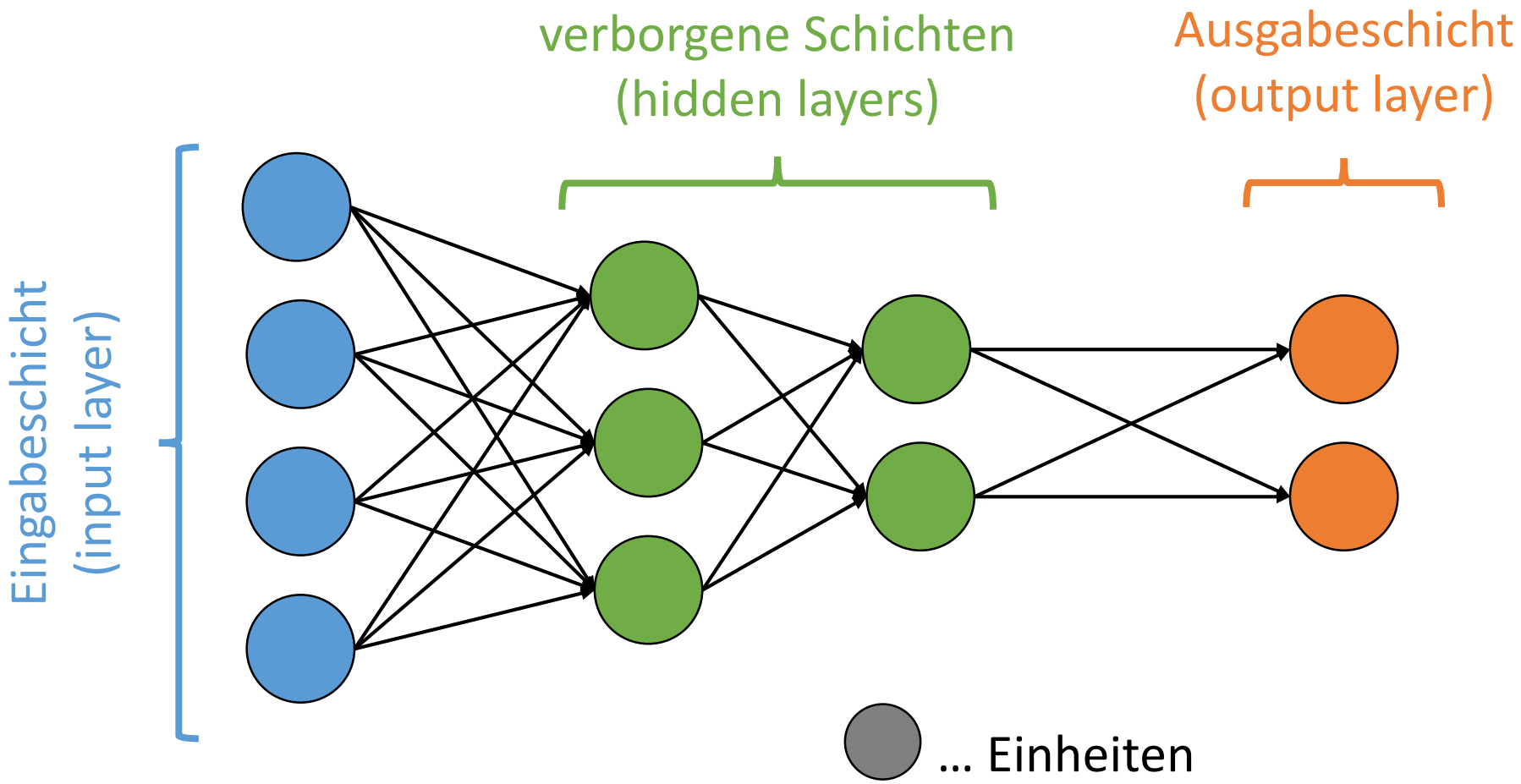
x_1	x_2	$o(x)$
0	0	-1
0	1	1
1	0	1
1	1	-1



Grundlagen neuronaler Netze

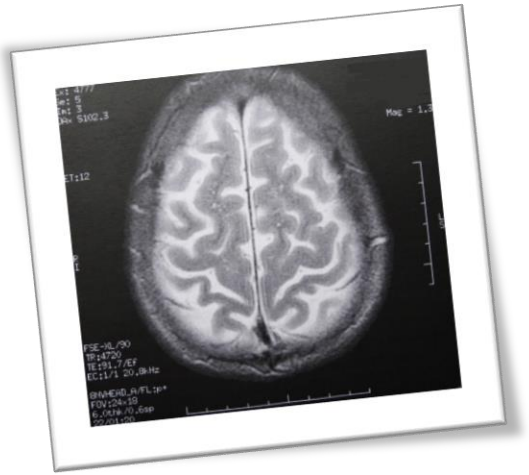


Was ist ein neuronales Netz?



Was ist ein neuronales Netz?

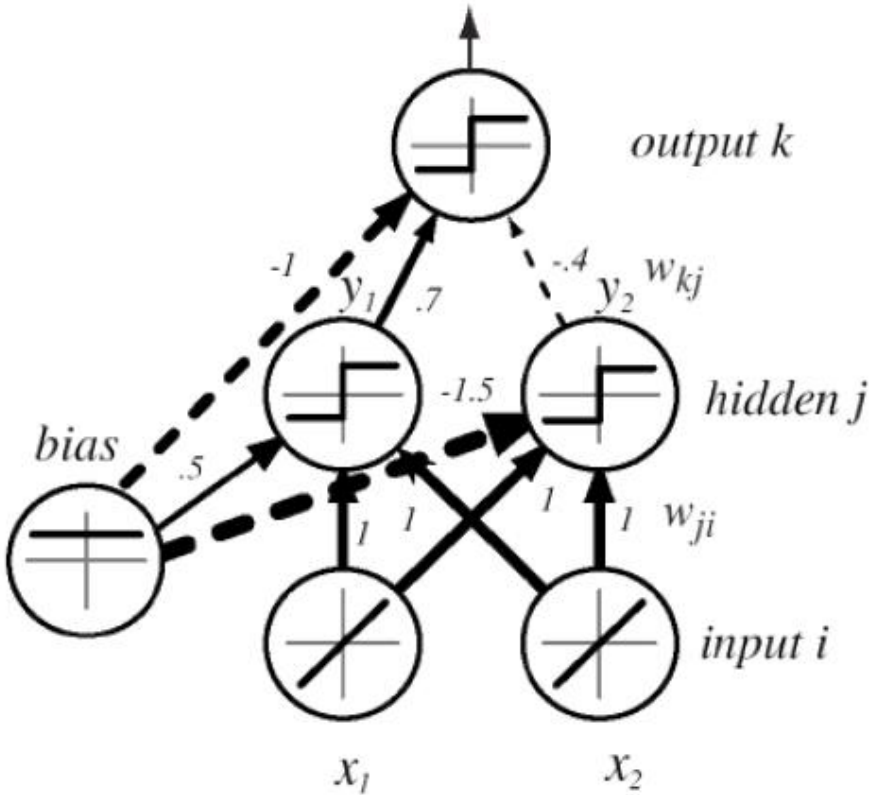
Neuronale Netze …

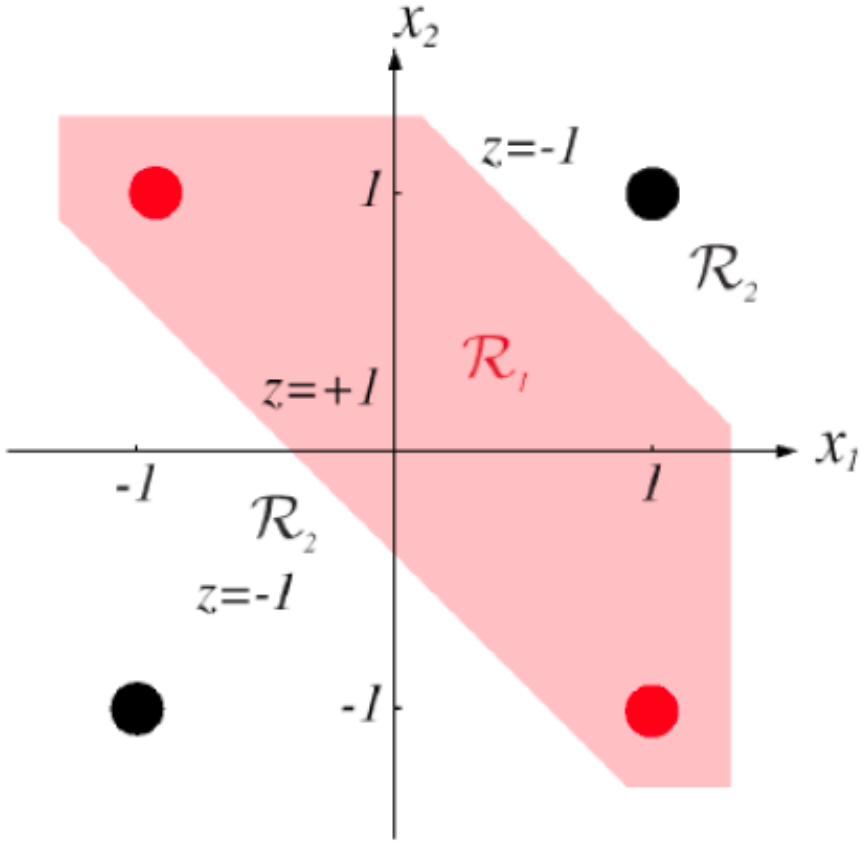


- bestehen aus 10 bis 1000 künstlichen Neuronen (Einheiten)
- sind nicht intelligent
- können nicht wirklich denken
- können begrenzte Mustererkennungsprobleme lösen
- sind nicht zwingend besser als andere Mustererkenn-ungsalgorithmen
- menschliches Gehirn besteht im Vergleich dazu aus 10.000.000.000 Neuronen

Lösung XOR

Ausgabe: Summe der gewichteten Signale der „verborgenen“ Schicht

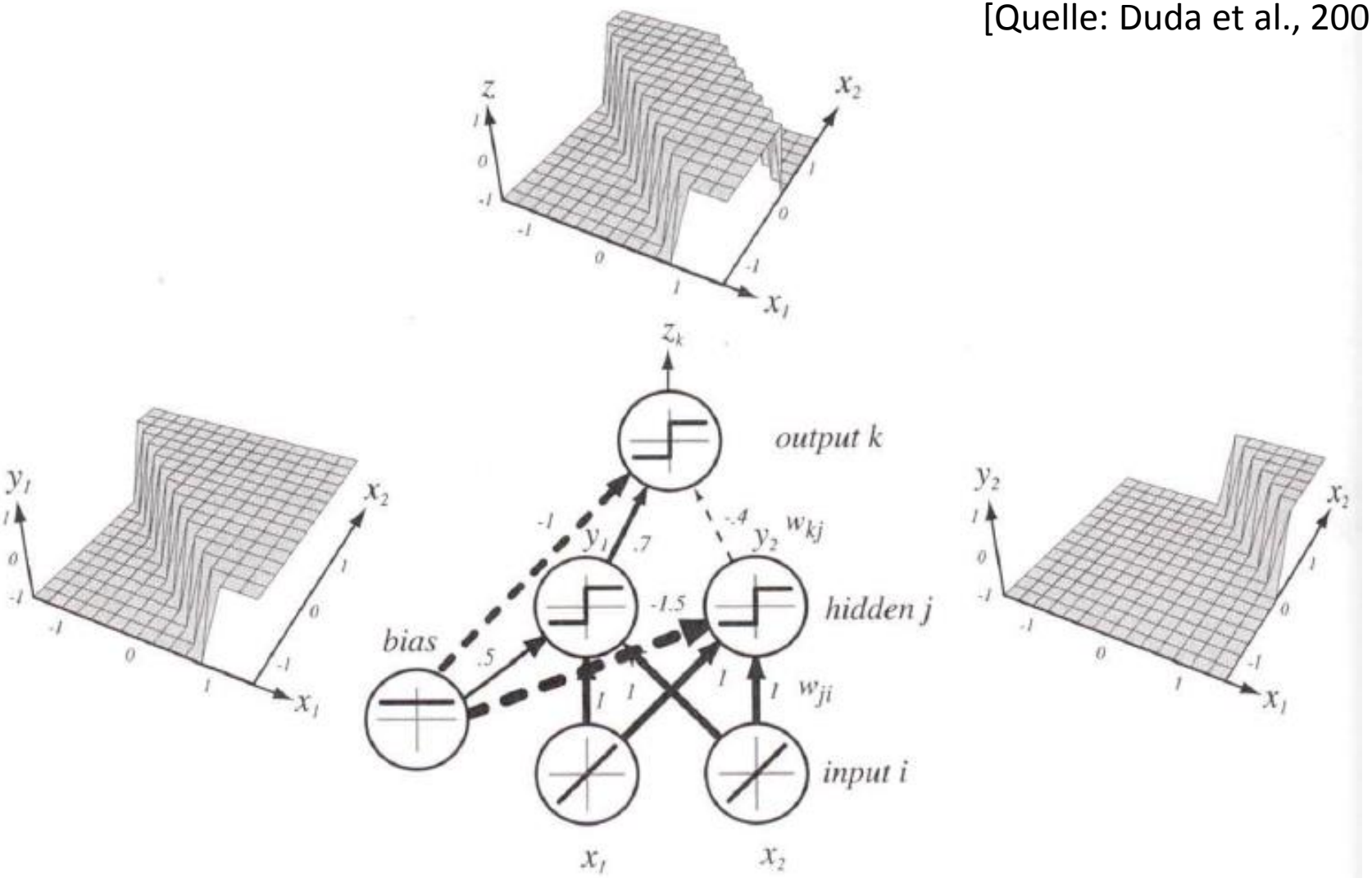




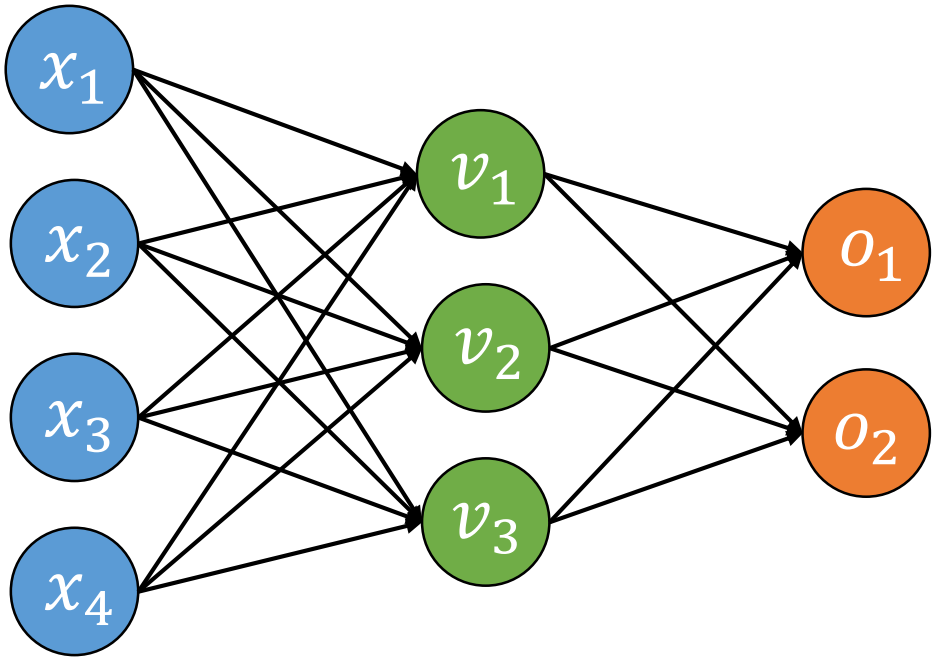
[Quelle: Duda et al., 2001]

Lösung XOR

[Quelle: Duda et al., 2001]



Einheiten



$\boldsymbol{x}$ ... Eingabevektor mit N_i Eingabeeinheiten (input units)

N_j verborgene Einheiten (hidden units) mit Aktivierung v_j

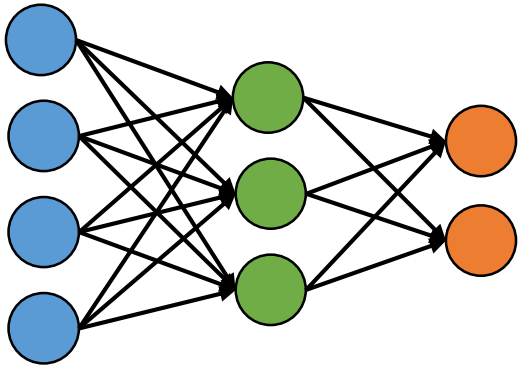
N_k Ausgabeeinheiten (output units) mit Aktivierung o_k

Anzahl der Einheiten

Anzahl der Eingabeeinheiten = Anzahl der Elemente des Merkmalsvektors

Anzahl der verborgenen Einheiten → nicht einfach festzulegen, Erfahrungswerte oder Experimente

Anzahl der Ausgabeeinheiten = Anzahl der Werte im Ausgabevektor = Anzahl der Klassen (in den meisten Fällen)



Ausgabevektor

… der Ausgabevektor $\boldsymbol{o}$ zeigt die Klassenzugehörigkeit auf

2 Klassen

- nur eine Ausgabeeinheit notwendig
- $\mathrm{o} = \{0\}$ oder $\mathrm{o} = \{1\}$ für Klasse 1 oder 2

$N > 2$ Klassen

- Ausgabevektor $\boldsymbol{o}$ besteht aus N Elementen
- 1-of-N Kodierung, z.B.: $\boldsymbol{o} = \{0,0,1,0,0\}$ → Klasse 3 von 5

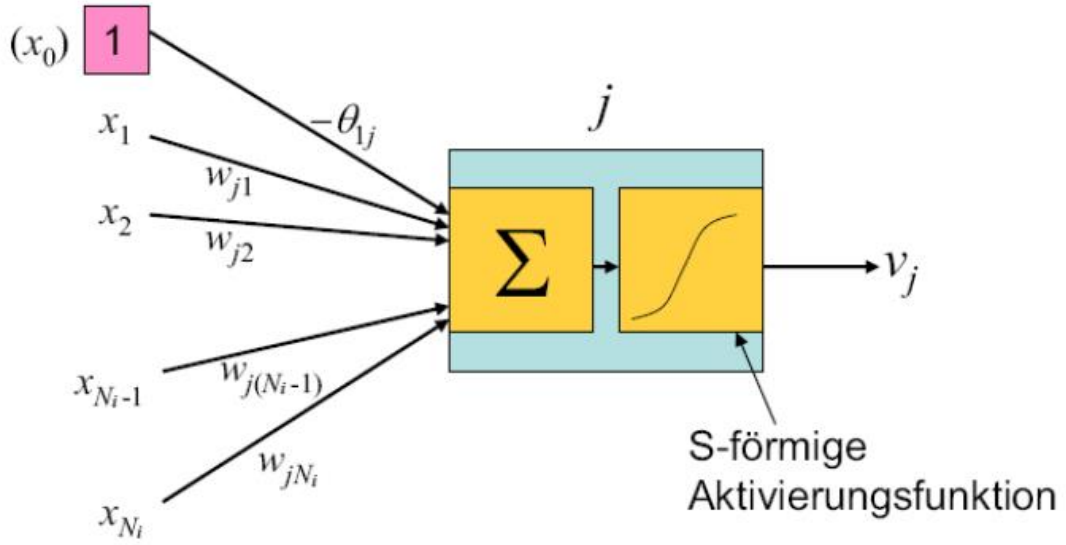
Details zu den Einheiten

Eingabeeinheiten

- verändern Daten (Merkmale, Beobachtungen) nicht
- schicken Daten an alle Einheiten der zweiten Schicht

Verborgene Einheiten und Ausgabeeinheiten

- ähnlich zu einem Perceptron
- Unterschied: Aktivierungsfunktion ist **keine Signumfunktion**



Aktivierungsfunktion

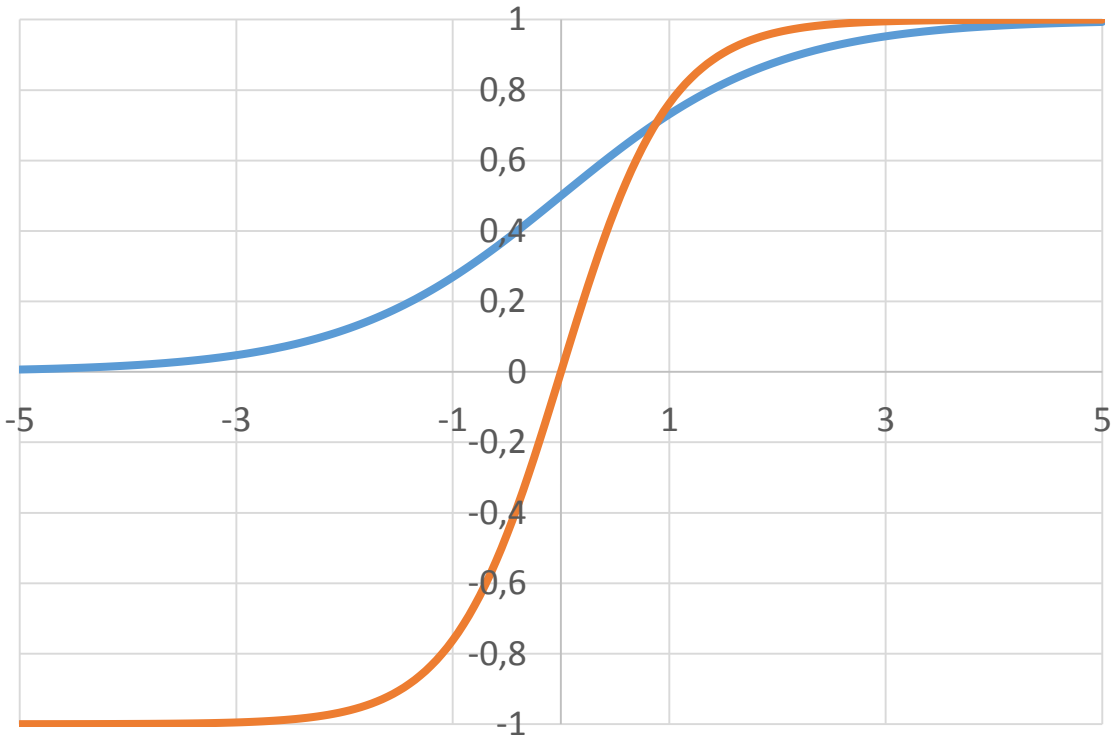
- statt Signumfunktion → differenzierbare, s-förmige Aktivierungsfunktion

- **sigmoid-Funktion**

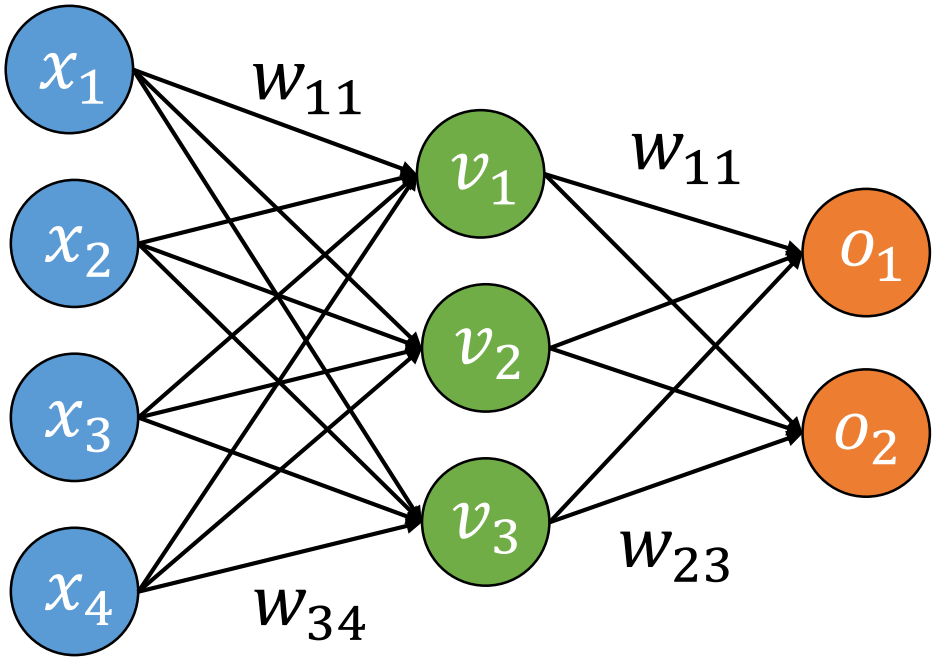
$$g(x) = \frac{1}{1 + e^{-x}}$$

- **Tangens Hyperbel-funktion**

$$g(x) = \tanh(x)$$



„Synapsen“ und Gewichte

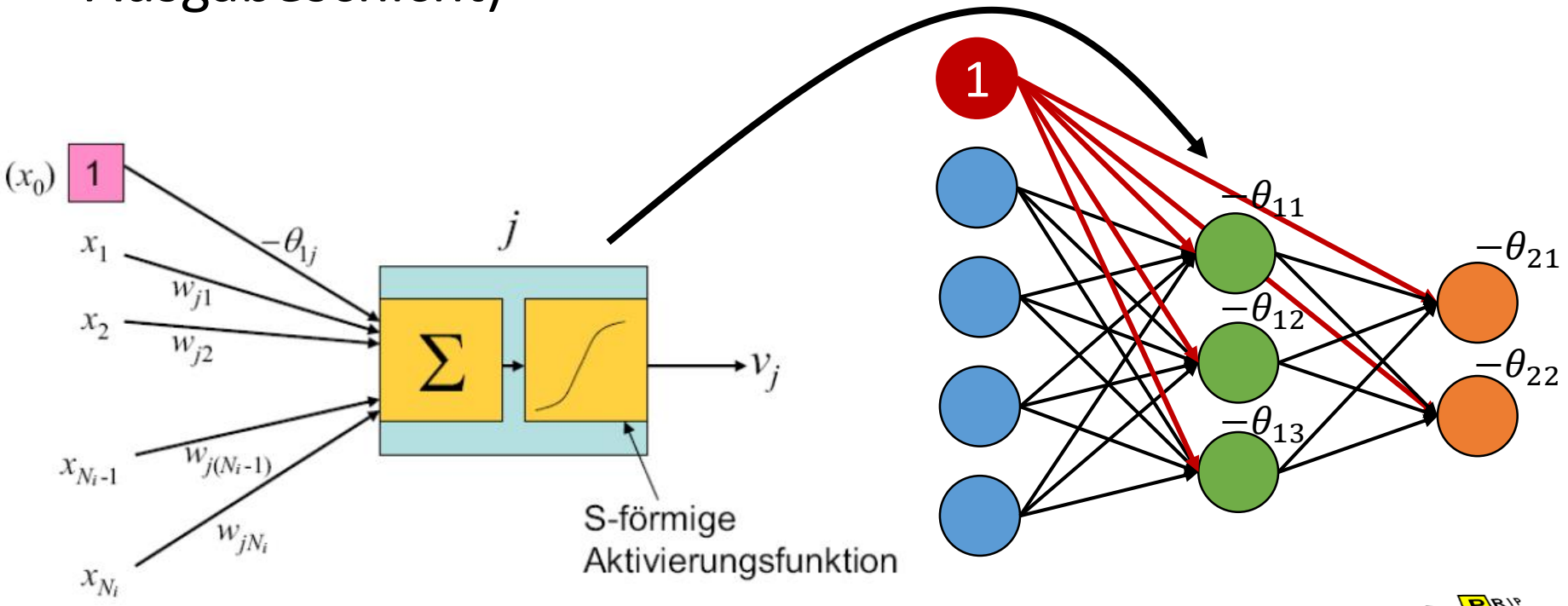


Jede Einheit einer Schicht ist mit allen Einheiten in der folgenden Schicht verbunden.

w_{ji} … Gewicht von Einheit i nach Einheit j

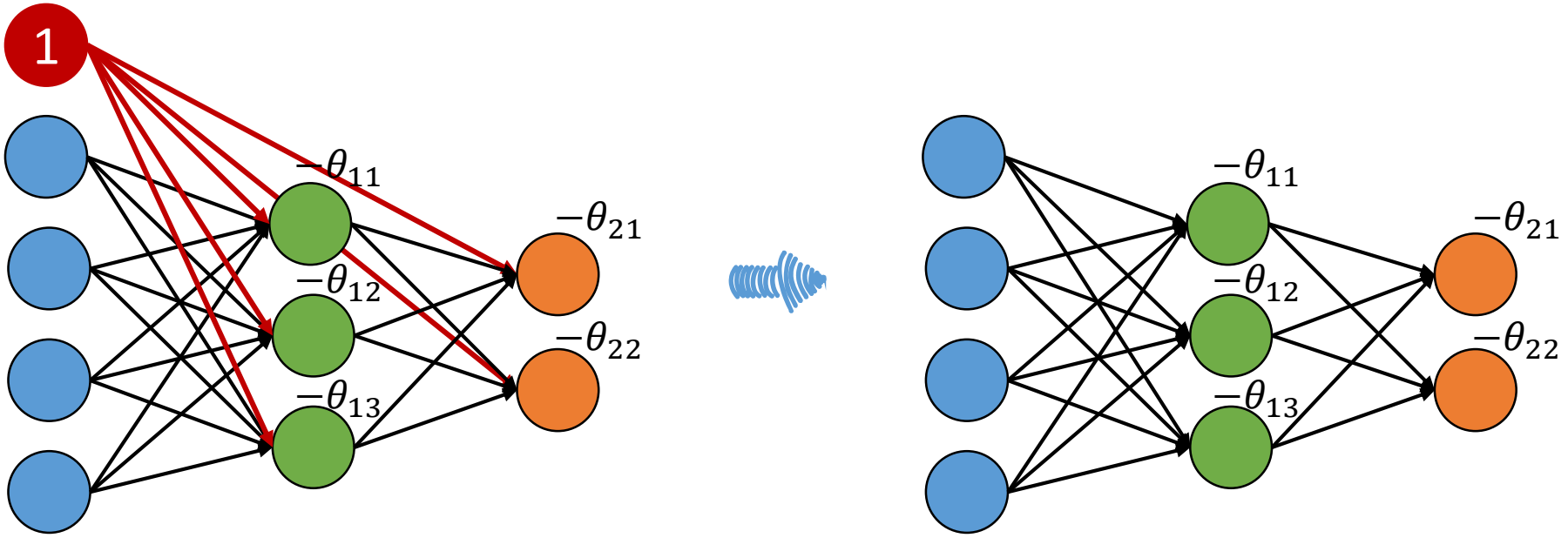
Bias und zusätzliche Einheit

- zusätzliche Einheit: Aktivierung 1, mit allen Einheiten verbunden
- Bias: je Neuron (Einheiten der verborgenen Schicht und Ausgabeschicht)



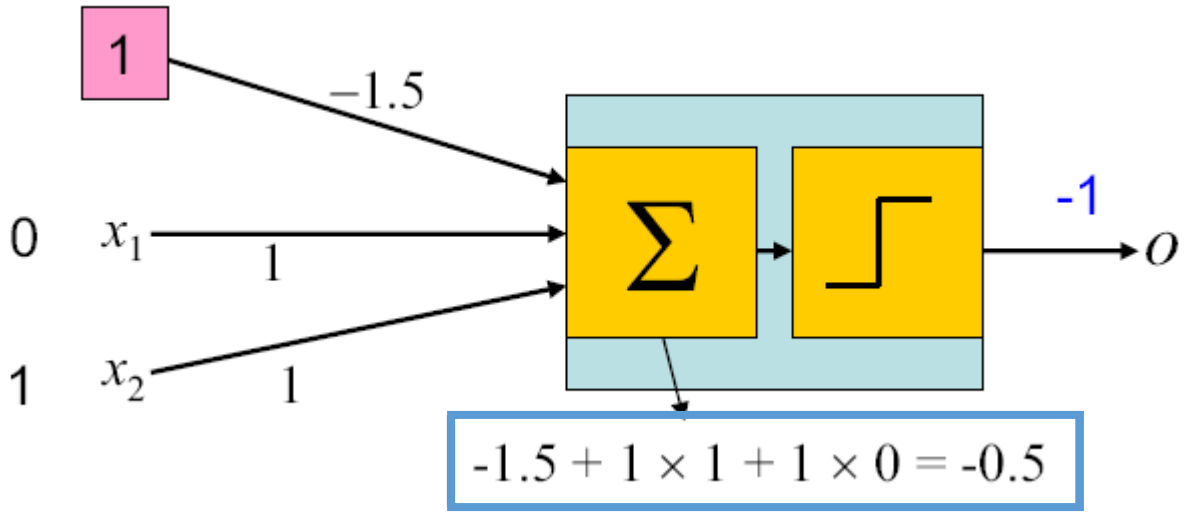
Bias und zusätzliche Einheit

Die zusätzliche Einheit und ihre Verbindungen werden oft nicht dargestellt, um die Darstellung des neuronalen Netzes zu vereinfachen.



Ausgabe des Perceptrons

- WH: Beispiel binäres UND



Berechnung der Ausgabe für Eingabevektor $\boldsymbol{x}$

Ausgabe des Netzes

… wird als „Forward Pass“ bezeichnet

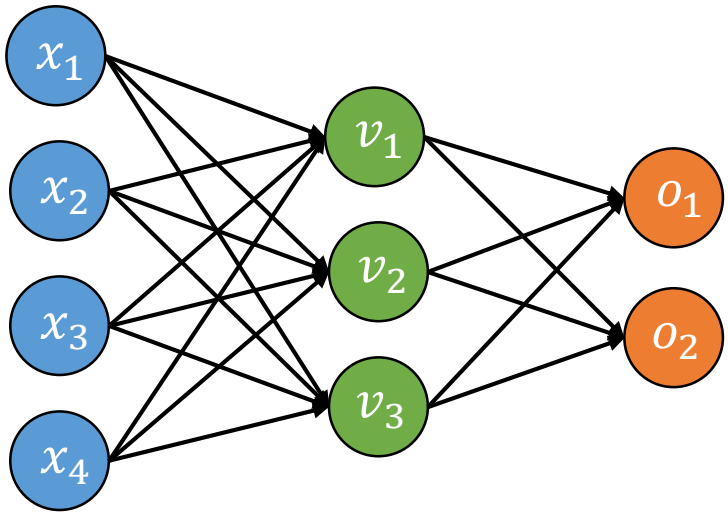
In unserem Beispiel:

1. Aktivierungswerte der verborgenen Schicht berechnen

$$v_j = g\left(\sum_{i=0}^{N_i} w_{ji}x_i\right) = g\left(\sum_{i=1}^{N_i} w_{ji}x_i - \theta_{1j}\right)$$

z.B.:

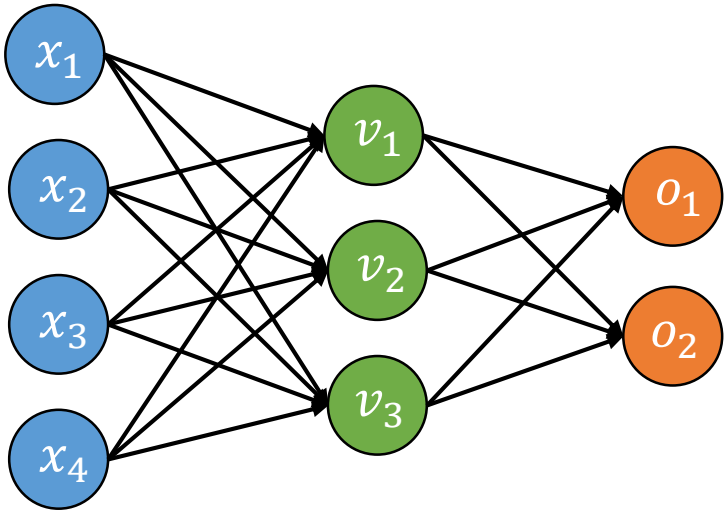
$$v_1 = g\begin{pmatrix} w_{11}x_1 + w_{12}x_2 + \\ w_{13}x_3 + w_{14}x_4 - \theta_{11} \end{pmatrix}$$



Ausgabe des Netzes

2. Aktivierungswerte der Ausgabeschicht berechnen

$$o_k = g\left(\sum_{j=0}^{N_j} w_{kj} v_j\right) = g\left(\sum_{j=1}^{N_j} w_{kj} v_j - \theta_{2k}\right)$$



Ausgabe des Netzes

… kann auch direkt berechnet werden

$$o_k = g\left(\sum_{j=0}^{N_j} w_{kj} v_j\right)$$

$$v_j = g\left(\sum_{i=0}^{N_i} w_{ji} x_i\right)$$

$$o_k = g\left[\sum_{j=0}^{N_j} w_{kj}\, g\left(\sum_{i=0}^{N_i} w_{ji} x_i\right)\right]$$

Ausgabe des Netzes

In der Praxis ist die Ausgabe eines neuronalen Netzes nicht exakt 0 oder 1 (wie auf Folie 15).

2 Klassen

- Ausgabewert o liegt im Intervall $[0,1]$
- Eingabevektor/Merkmalsvektor $\boldsymbol{x}$ gehört zu
  - Klasse 1 wenn $o < 0{,}5$
  - Klasse 2 wenn $o \geq 0{,}5$

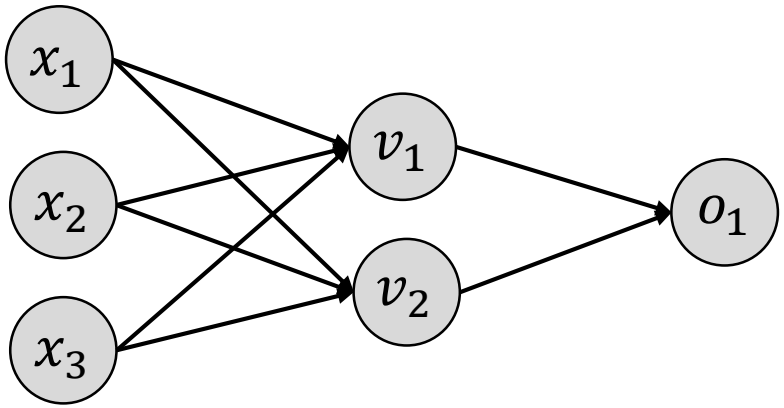
$N > 2$ Klassen

- Ausgabevektor $\boldsymbol{o}$ besteht aus Werten die im Intervall $[0,1]$ liegen
- Merkmalsvektor $\boldsymbol{x}$ gehört zur Klasse mit dem größten Aktivierungswert in $\boldsymbol{o}$
- z.B.: $\boldsymbol{o} = \{0{,}1\ 0{,}31\ \boxed{0{,}92}\ 0{,}61\}$ → $\boldsymbol{x}$ gehört zur Klasse 3

Beispiel

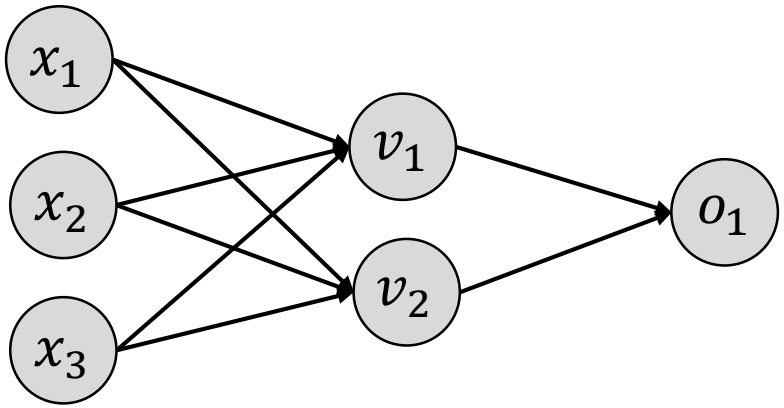
… Berechnung Forward Pass

Neuronales Netz mit $N_i = 3$ Eingabeeinheiten x_i, $N_j = 2$ verborgenen Einheiten v_j und $N_k = 1$ Ausgabeeinheiten o_k.



Beispiel

w_{ji}	θ_{1j}	w_{kj}	θ_{2k}
$w_{11} = 0{,}2$	$\theta_{11} = 1$	$w_{11} = 0{,}4$	$\theta_{21} = 0{,}5$
$w_{12} = 0{,}6$	$\theta_{12} = 0{,}5$	$w_{12} = 0{,}6$	
$w_{13} = 0{,}2$			
$w_{21} = 0{,}1$			
$w_{22} = 0{,}8$			
$w_{23} = 0{,}1$			



Beispiel

Schritt 1: Ermittle die Aktivierungswerte der verborgenen Schicht für Eingabevektor $x_1 = 3, x_2 = 1$ und $x_3 = 2$.

$$v_j = g\left(\sum_{i=1}^{N_i} w_{ji}x_i - \theta_{1j}\right)$$

w_{ji}	θ_{1j}
$w_{11} = 0{,}2$	$\theta_{11} = 1$
$w_{12} = 0{,}6$	$\theta_{12} = 0{,}5$
$w_{13} = 0{,}2$	
$w_{21} = 0{,}1$	
$w_{22} = 0{,}8$	
$w_{23} = 0{,}1$	

$$v_1 = 0{,}2 \cdot 3 + 0{,}6 \cdot 1 + 0{,}2 \cdot 2 - 1 = 0{,}6$$
$$v_2 = 0{,}1 \cdot 3 + 0{,}8 \cdot 1 + 0{,}1 \cdot 2 - 0{,}5 = 0{,}8$$

Beispiel

Schritt 2: Ermittle Aktivierungswert der Ausgabeeinheit.

$$o_k = g\left(\sum_{j=1}^{N_j} w_{kj} v_j - \theta_{2k}\right)$$

w_{kj}	θ_{2k}
$w_{11} = 0{,}4$	$\theta_{21} = 0{,}5$
$w_{12} = 0{,}6$	

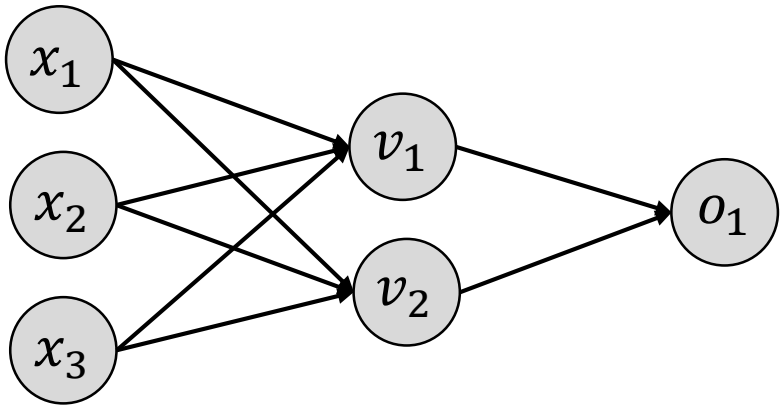
$$v_1 = 0{,}6$$
$$v_2 = 0{,}8$$

$$o_1 = 0{,}6 \cdot 0{,}4 + 0{,}8 \cdot 0{,}6 - 0{,}5 = 0{,}22$$

Beispiel

Interpretation des Ergebnisses: o_1 =0,22

- **Klasse 1 wenn $o < 0,5$**
- Klasse 2 wenn $o \geq 0,5$



Terminologie

Multilayer-Perceptron (MLP): Neuronale Netze werden oft als MLP bezeichnet. Dieser Name ist aber irreführend, da die Neuronen in einem neuronalen Netz andere Aktivierungs-funktionen verwenden als das Perceptron.

Backpropagation-Netz nennt man neuronale Netze die mit dem Backpropagation-Algorithmus trainiert wurden. Dieser Algorithmus ist ein überwachtes Lernverfahren das aus drei Schritten besteht (mehr dazu im nächsten Abschnitt).

Training neuronaler Netze

Backpropagation

… ist ein **Algorithmus zum Training** neuronaler Netze mit mehreren Schichten

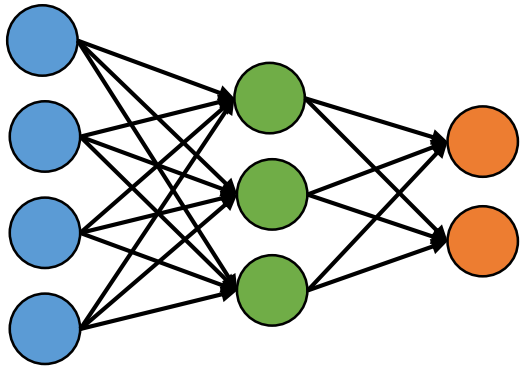
Interessante Fakten:

- in den 70er Jahren von mehreren Autoren unabhängig voneinander vorgeschlagen
- nach 10 Jahren Vergessenheit von mehreren Autoren wiederentdeckt
- bekannteste Publikation: “Parallel Distributed Processing: Explorations in the Microstructure of Cognition“ von Rumelhart, Hinton und Williams, 1986

Was wird trainiert?

Beispiel:

- $N_i = 4$ Eingabeeinheiten
- $N_j = 3$ verborgene Einheiten
- $N_k = 2$ Ausgabeeinheiten



Wenn jede Einheit mit jeder anderen in der darauffolgenden Schicht verbunden ist, gibt es folgende Parameter zu trainieren:

$$\underbrace{N_i \cdot N_j}_{\text{Gewichte zwischen Schicht 0 und 1}} + \underbrace{N_j \cdot N_k}_{\text{Gewichte zwischen Schicht 1 und 2}} + \underbrace{N_j + N_k}_{\text{Bias}}$$

- für das Beispiel: $4 \cdot 3 + 3 \cdot 2 + 3 + 2 = 23$ Parameter

Grundidee des Trainings

- für Parameter (Gewichte, Bias) werden **Zufallswerte** angenommen

- Backpropagation-Algorithmus ändert die Parameter mit Hilfe eines **Trainingsdatensatzes**
  - überwachtes Lernen (Supervised Learning)

- Trainingsdatensatz besteht aus:
  - N Eingabevektoren $\boldsymbol{x}_n$ und den zugehörigen
  - N Ausgabevektoren $\boldsymbol{t}_n$ (target vectors)
  - idealerweise ein möglichst großer Datensatz

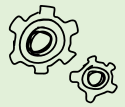
Gewichtsvektor

Der aktuelle Status des neuronalen Netzes kann während des Trainings durch einen Gewichtsvektor $\boldsymbol{w}$ beschrieben werden. $\boldsymbol{w}$ enthält für ein neuronales Netz mit einer verborgenen Schicht:

- alle Gewichte w_{ji}
- alle Gewichte w_{kj}
- und alle Biaswerte θ_j und θ_k

Grundidee des Trainings

Folgende Schritte werden wiederholt durchlaufen (iterativ) bis das Netz ausreichend trainiert ist → der Fehler klein genug ist:



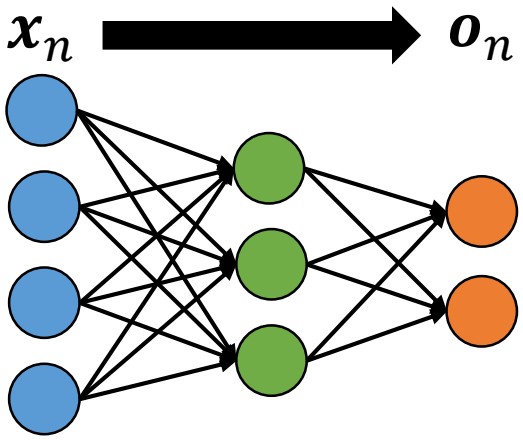
1. Forward Pass
2. Bestimmung des Fehlers
3. Backward Pass

- Training läuft üblicherweise in mehreren Zyklen (Epochen) ab, wobei ein Zyklus bedeutet, dass alle Eingabevektoren $\boldsymbol{x}_n$ abgearbeitet wurden.

Forward Pass

Für einen beliebigen Eingabevektor $\boldsymbol{x}_n$ aus dem Trainingsdatensatz wird der Ausgabevektor $\boldsymbol{o}_n$ ermittelt. Seine Elemente o_k ergeben sich wie folgt:

$$o_k = g\left[\sum_{j=0}^{N_j} w_{kj}\, g\left(\sum_{i=0}^{N_i} w_{ji} x_i\right)\right]$$

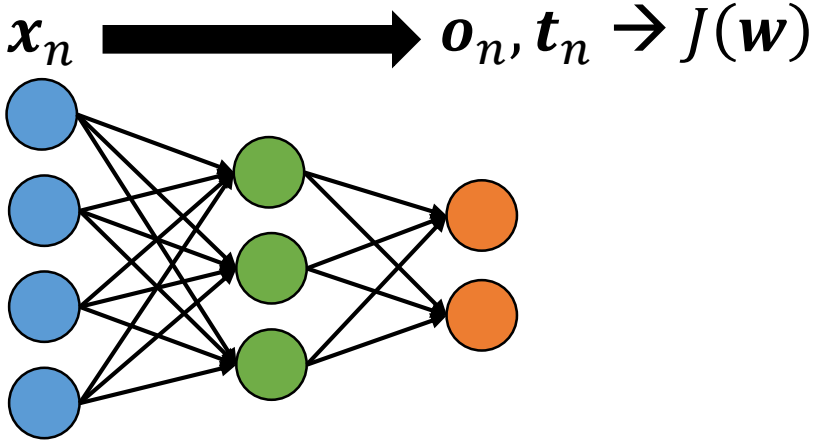


Bestimmung des Fehlers

Der vom neuronalen Netz gelieferte Ausgabevektor $\boldsymbol{o}_n$ wird mit der Zielausgabe $\boldsymbol{t}_n$ verglichen:

$$J(\boldsymbol{w}) = \frac{1}{2}\|\boldsymbol{t}_n - \boldsymbol{o}_n\|^2 = \frac{1}{2}\sum_{k=1}^{N_k}(t_k - o_k)^2$$

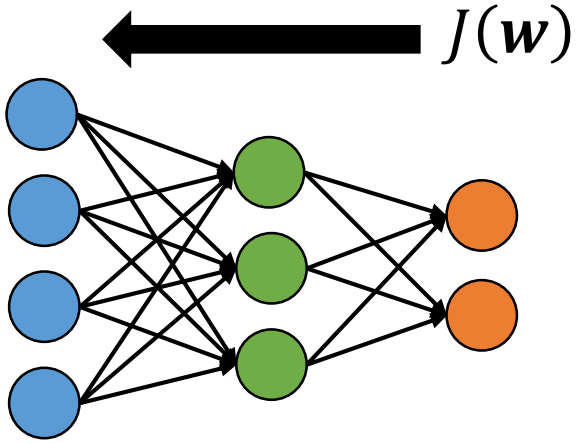
Das Ziel des Trainings ist es $J(\boldsymbol{w})$ (Sum of Squares Error) zu minimieren. $J(\boldsymbol{w})$ wird auch als Kostenfunktion bezeichnet.



Backward Pass

… nur der Grundgedanke

Der Backward Pass erfolgt in umgekehrter Reihenfolge wie der Forward Pass. Beginnend mit den Gewichten der Ausgabeschicht werden die Gewichte sukzessive anhand des Fehlers verändert.



Gesamtfehler

Man kann den Fehler auch für den gesamten Trainingsdaten-satz berechnen:

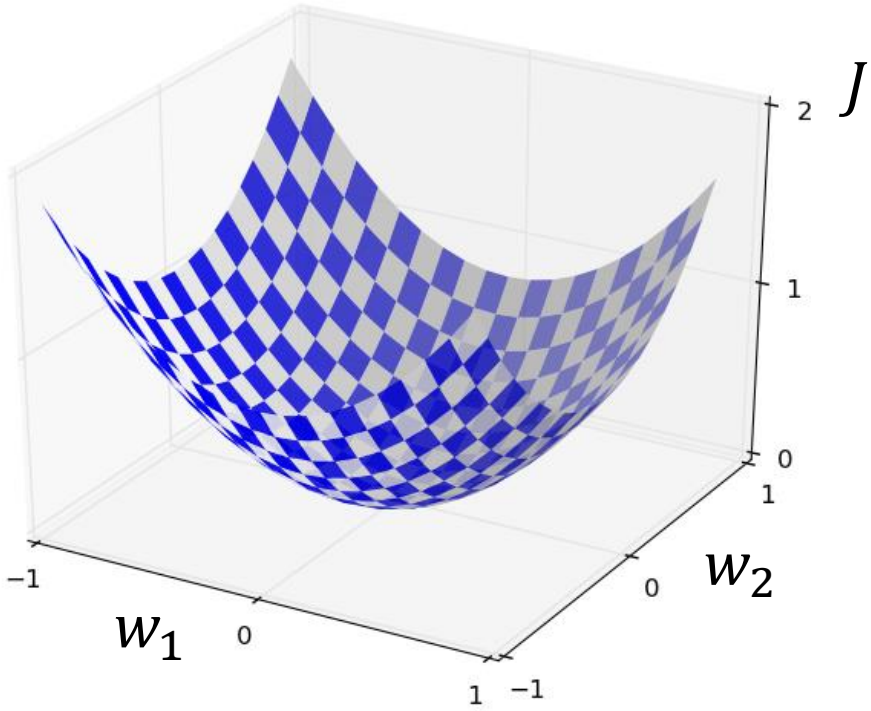
$$J(\boldsymbol{w}) = \sum_{n=1}^{N} J_n(\boldsymbol{w})$$

wobei N die Anzahl der Eingabevektoren ist und J_n der Fehler für den Eingabevektor $\boldsymbol{x}_n$.

Fehlerfunktion

… oder die Fehleroberfläche für ein einfaches Netz mit nur zwei Gewichten w_1 und w_2

Die Koordinaten des globalen Minimums dieser Fehler-funktion, sind gleichzeitig die Gewichte die den Fehler $J(w_1, w_2)$ minimieren.

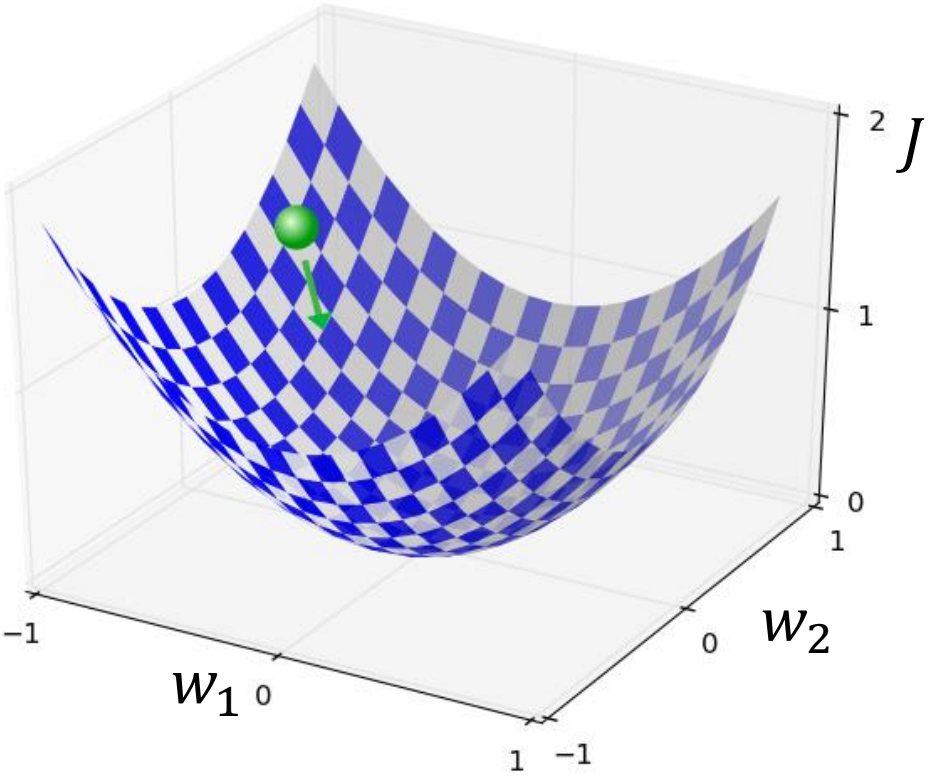


Quelle: http://neuralnetworksanddeeplearning.com

Gradient Descent

- Verfahren zum Auffinden lokaler Minima

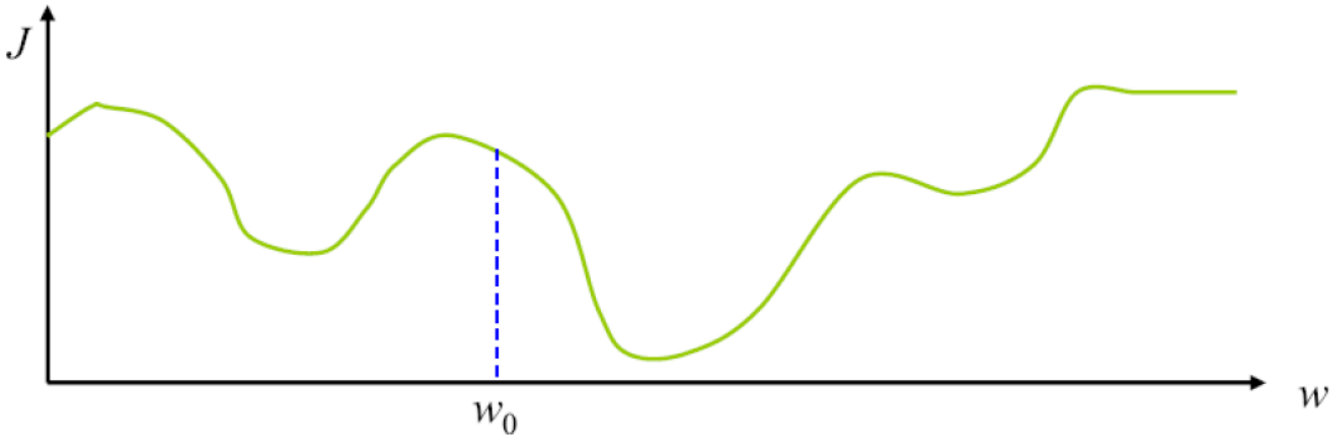
- intuitiv kann man es sich wie das Hinunterrollen eines Balls ins “Tal” der Funktion vorstellen (unter Vernachlässigung der Schwungkraft, etc.)



Quelle: http://neuralnetworksanddeeplearning.com

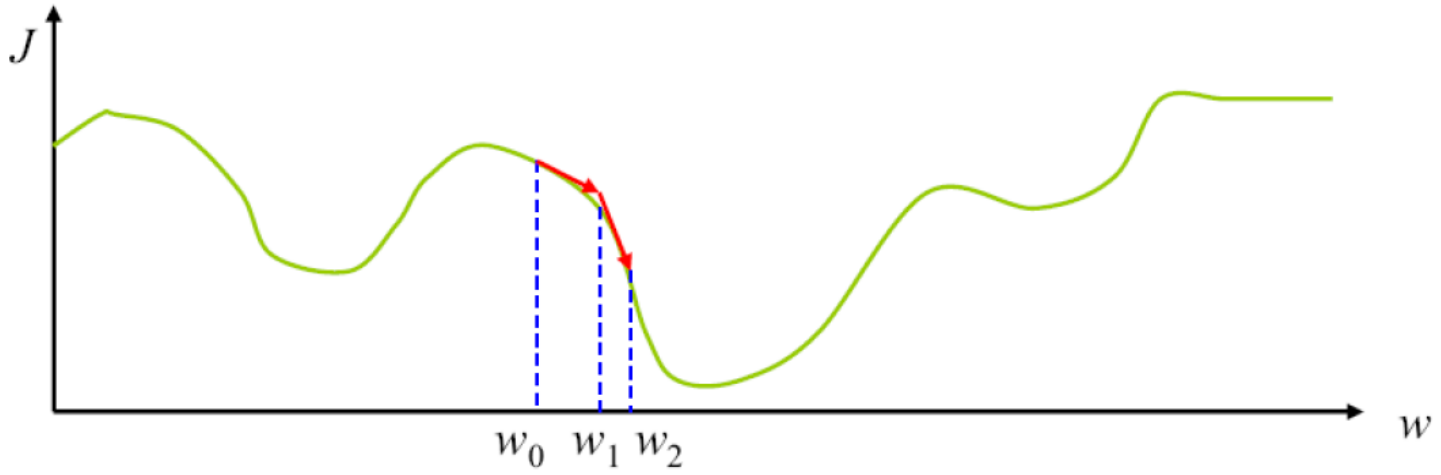
Backpropagation-Lernverfahren

- Gewichtsvektor $\boldsymbol{w}_o$ entspricht einem bestimmen Punkt auf der Fehleroberfläche (üblicherweise eine multidimensionale Oberfläche)
- Lernverfahren verändert die Gewichte von $\boldsymbol{w}_o$, sodass $\boldsymbol{w}_o$ in ein (möglichst globales) Minimum bewegt wird



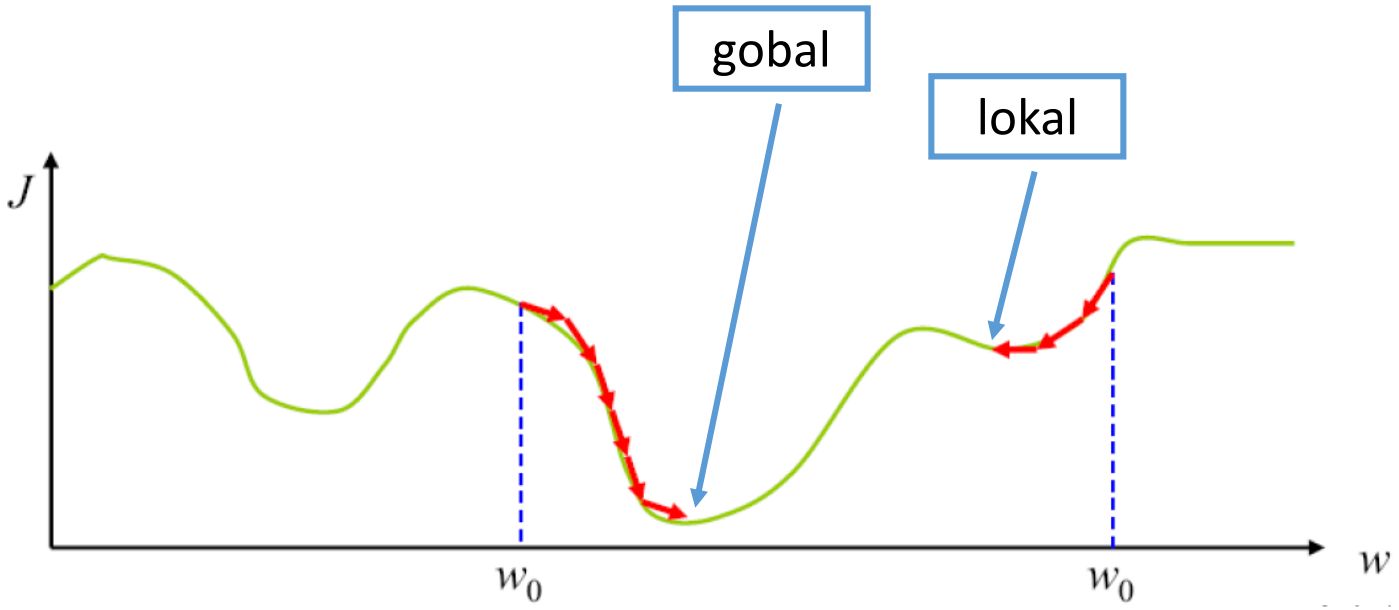
Backpropagation-Lernverfahren

- Verwendung eines Gradient-Descent-Verfahrens
- Punkt $\boldsymbol{w}_o$ wird in Richtung des steilsten Abstiegs um eine vorgegebene Länge (**Lernrate**) bewegt → dadurch erhält man $\boldsymbol{w}_1$



Problem des Lernverfahrens

Der Erfolg des Gradient-Descent-Verfahrens hängt stark von den Anfangsgewichten in $\boldsymbol{w}_o$ ab. Es ist nicht sicher, dass das Lernverfahren das globale Minimum findet.



Nachteile

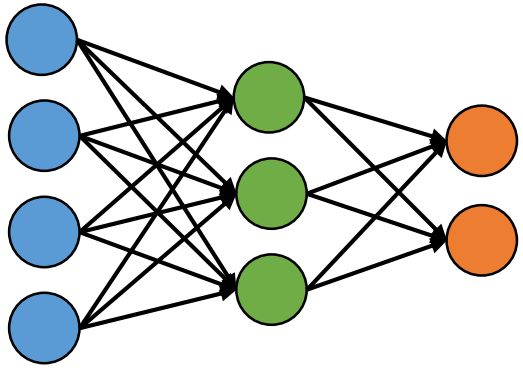
… des Backpropagation-Algorithmus

- **Konvergenzzeit**: langsam (Anzahl der benötigten Iterationen)
- **Paramter Auswahl**: schwierig
  - Anzahl der verborgenen Einheiten
  - Lernrate γ
- **Stoppen des Trainingsalgorithmus**: schwierig
  - Wie weiß man, dass das globale Minimum gefunden wurde?

Wichtige Hinweise und Tipps

Reihenfolge der Trainingsdaten

Da die Gefahr besteht, dass das Backpropagation-Netz die „Reihenfolge“ der Trainingsdaten lernt, sollte die Reihenfolge in jedem Zyklus verändert werden.



Skalierung der Einheiten

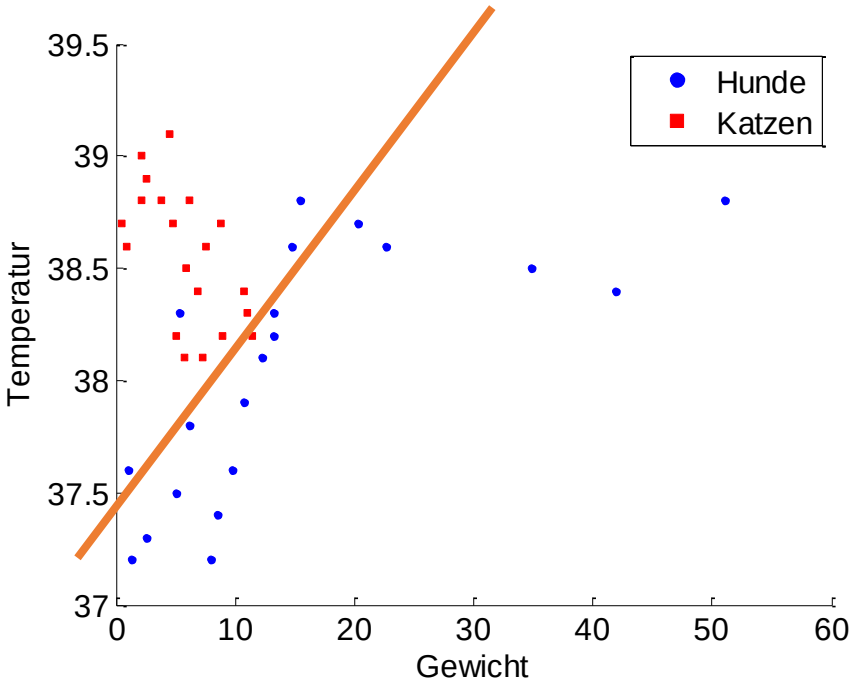
- Merkmale liegen oft in unterschiedlichen Einheiten vor
  - z.B.: Meter, Millimeter, Liter, etc.

- damit alle Merkmale gleichen Einfluss haben → **Standardisierung** (Mittelwert = 0, Varianz = 1)

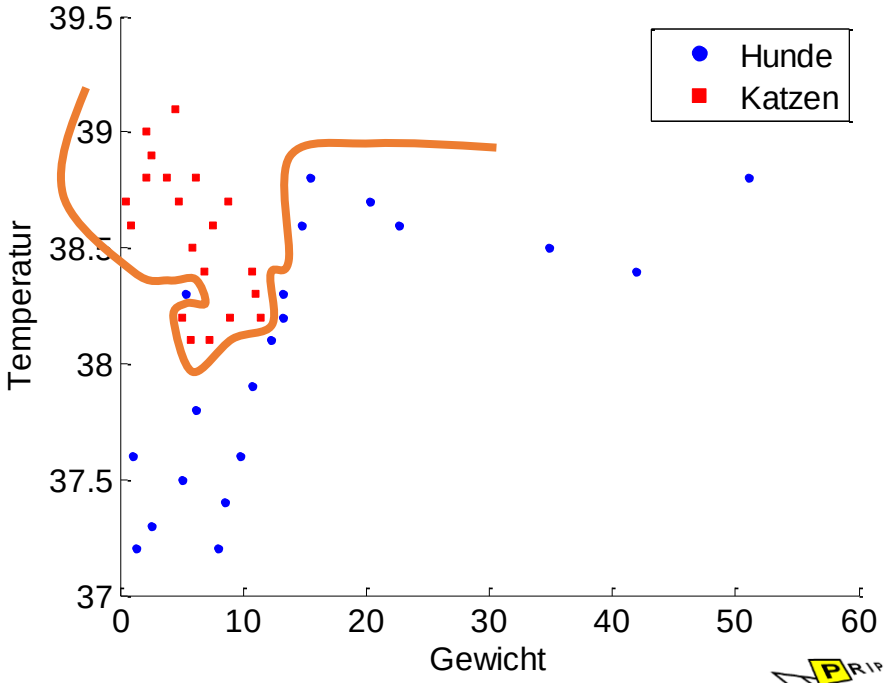
- **zusätzlicher Vorteil**: Gradient-Descent-Verfahren konvergieren schneller auf skalierten Merkmalen

Verborgene Einheiten

Netz mit **zu wenig verborgenen Einheiten** kann Trainingsdaten-satz nicht ausreichend lernen (underfitting).

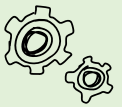


Netz mit **zu vielen verborgenen Einheiten** trainiert zu lange und hat schlecht Generalisierungs-fähigkeit (overfitting)



Verborgene Einheiten

Daumenregel: Anzahl der verborgenen Einheiten $\approx \frac{N}{10}$



Ermittlung der geeigneten Anzahl an verborgenen Einheiten aus Trainings- und Testdaten:

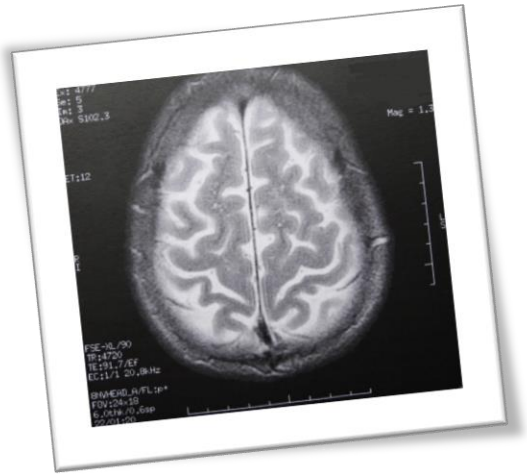
1. Beginne mit einer großen Anzahl an verborgenen Einheiten
2. Verborgene Einheiten nacheinander entfernen (Pruning)
3. Fehler für Trainings- und Testdaten messen

Regularisierung

- beim **„normalen“ Lernen** versucht man den Ausgabefehler eines neuralen Netzes zu minimieren

- beim **regularisierten Lernen** versucht man gleichzeitig die Komplexität des neuralen Netzes so niedrig wie möglich zu halten.

- z.B. „Optimal Brain Damage“ und „Optimal Brain Surgeon“ Algorithmen
  - entfernen die Gewichte die den kleinsten Einfluss auf den Ausgabefehler haben



Anzahl der Schichten

- neuronale Netze bestehen oft **aus drei Schichten**
  - eine Eingabeschicht
  - eine verborgene Schicht
  - und eine Ausgabeschicht

- **vier oder mehr Schichten** nur in bestimmten Fällen nötig
  - z.B.: in der Bildverarbeitung, wenn man invariant gegenüber Skalierung, Rotation und Translation sein will/muss

- Praxis: Neurale Netze mit mehr als einer verborgenen Schicht neigen eher dazu **in lokalen Minimas stecken zu bleiben**.

Lernrate

… wird oft klein angesetzt

- für viele Probleme ist $\gamma = 0{,}1$ geeignet

Man sollte die Lernrate nicht zu klein ansetzten, sonst dauert es sehr lange bis das Gradient-Descent-Verfahren zu einem lokalen Minimum konvergiert.

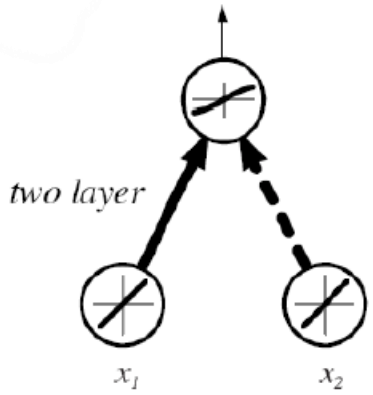
- Praxis: es wird beim Training von neuronalen Netzen nicht gewartet bis das Lernverfahren konvergiert

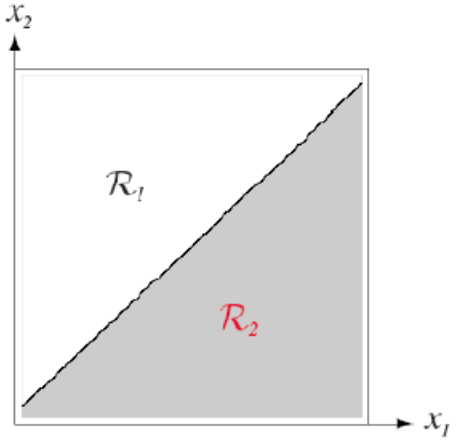
Mächtigkeit …

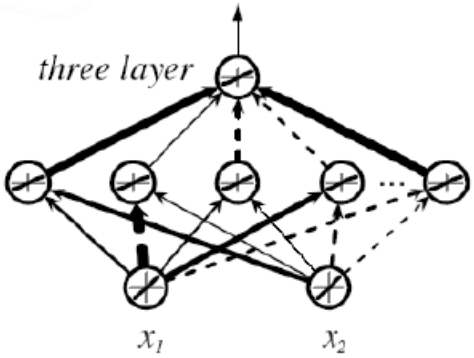
… des Backpropagation-Netzes (BN)

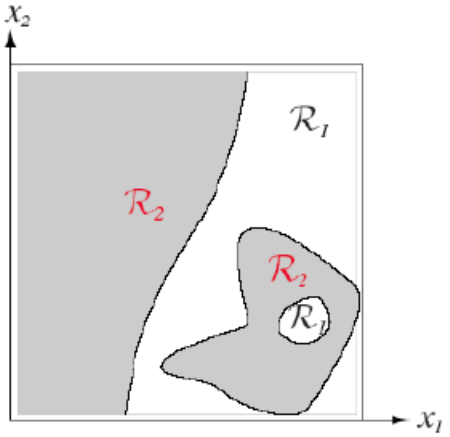
- BN können beliebige Entscheidungsgrenzen modellieren
- Leider gibt es keine konkreten Regeln für die Anzahl der verborgenen Einheiten

[Quelle: Duda et al., 2001]



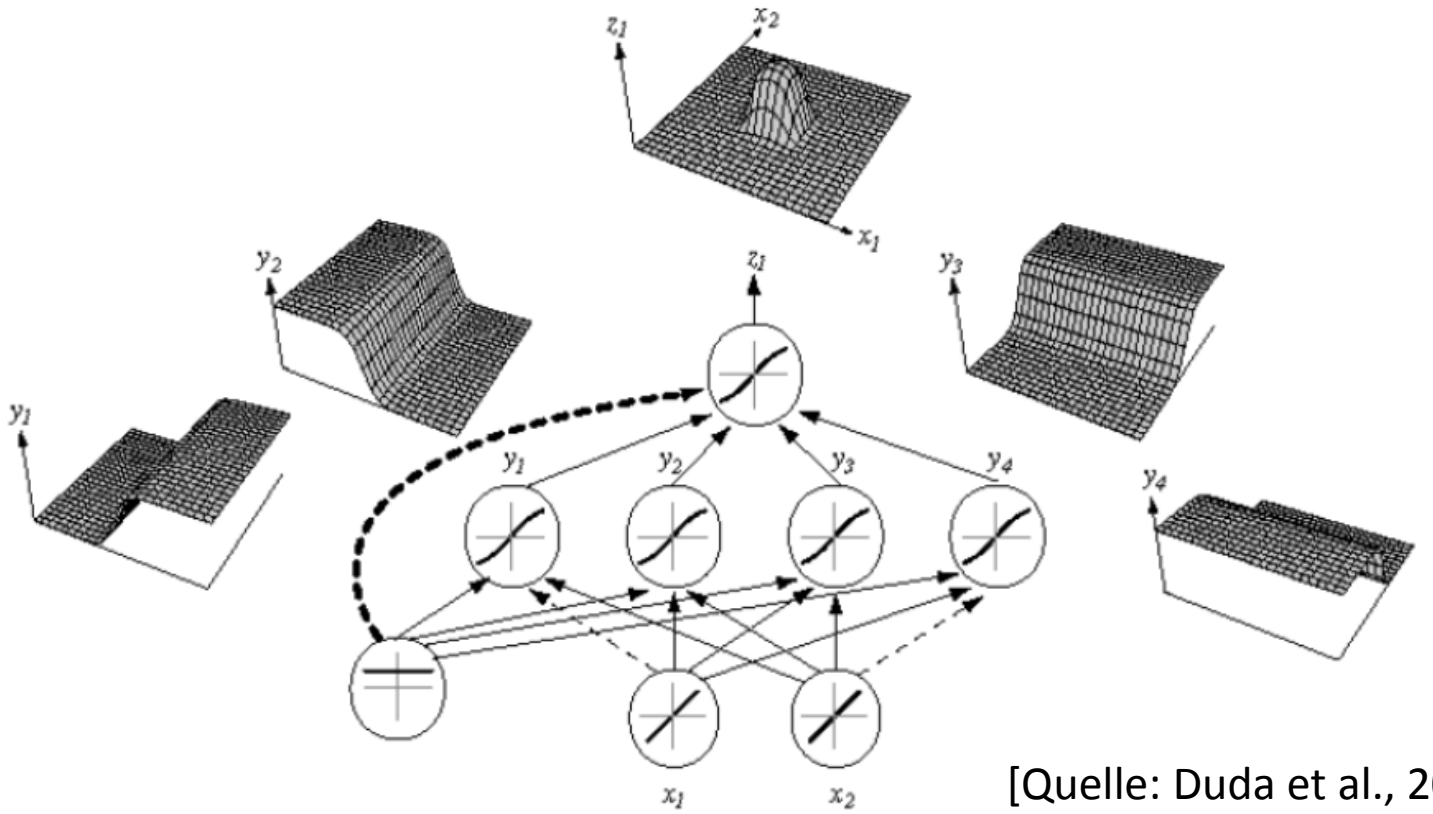






Mächtigkeit …

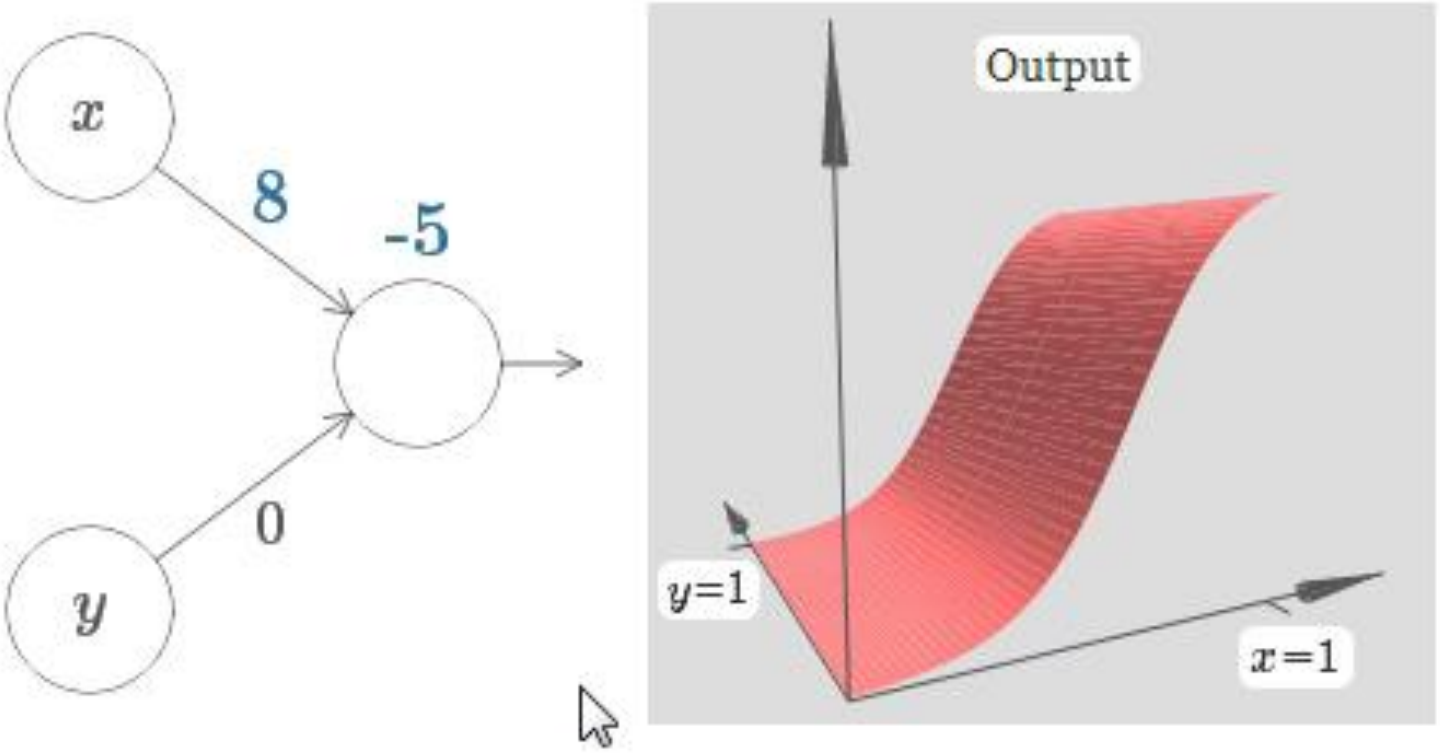
… des Backpropagation-Netzes (BN)



[Quelle: Duda et al., 2001]

Mächtigkeit …

… des Backpropagation-Netzes (BN)



Quelle: http://neuralnetworksanddeeplearning.com

Vorteile Neuronaler Netze

- Parallelisierbar → Hardware-Implementierung
- einfaches Trainingsverfahren (Backpropagation)
- beliebige Entscheidungsgrenzen

Nachteile Neuronaler Netze

- Training kann **langwierig** sein (falsch gewählte Lernrate, etc.)
- Training benötigt **viele Parameter** die teilweise schwer wählbar sind
  - Anzahl der Schichten
  - Anzahl der verborgenen Einheiten
  - Lernrate
- Neurale Netze mit vielen verborgenen Einheiten neigen zum **„Overfitting“**
  - um dem entgegen zu wirken, sollte der Trainingsdatensatz groß sein
- **„Black-Box“**
  - es ist schwierig nachzuvollziehen wie ein Netz ein Problem löst

Anwendungsbeispiele

Handschrifterkennung

- einfaches „Problem“ für Menschen
- schwieriges Problem für Computer
- Beispiel: vereinfachtes Problem → Erkennung der Zahlen von 0 – 9

504192

Quelle: http://neuralnetworksanddeeplearning.com

Handschrifterkennung

1. Problem: Segmentierung

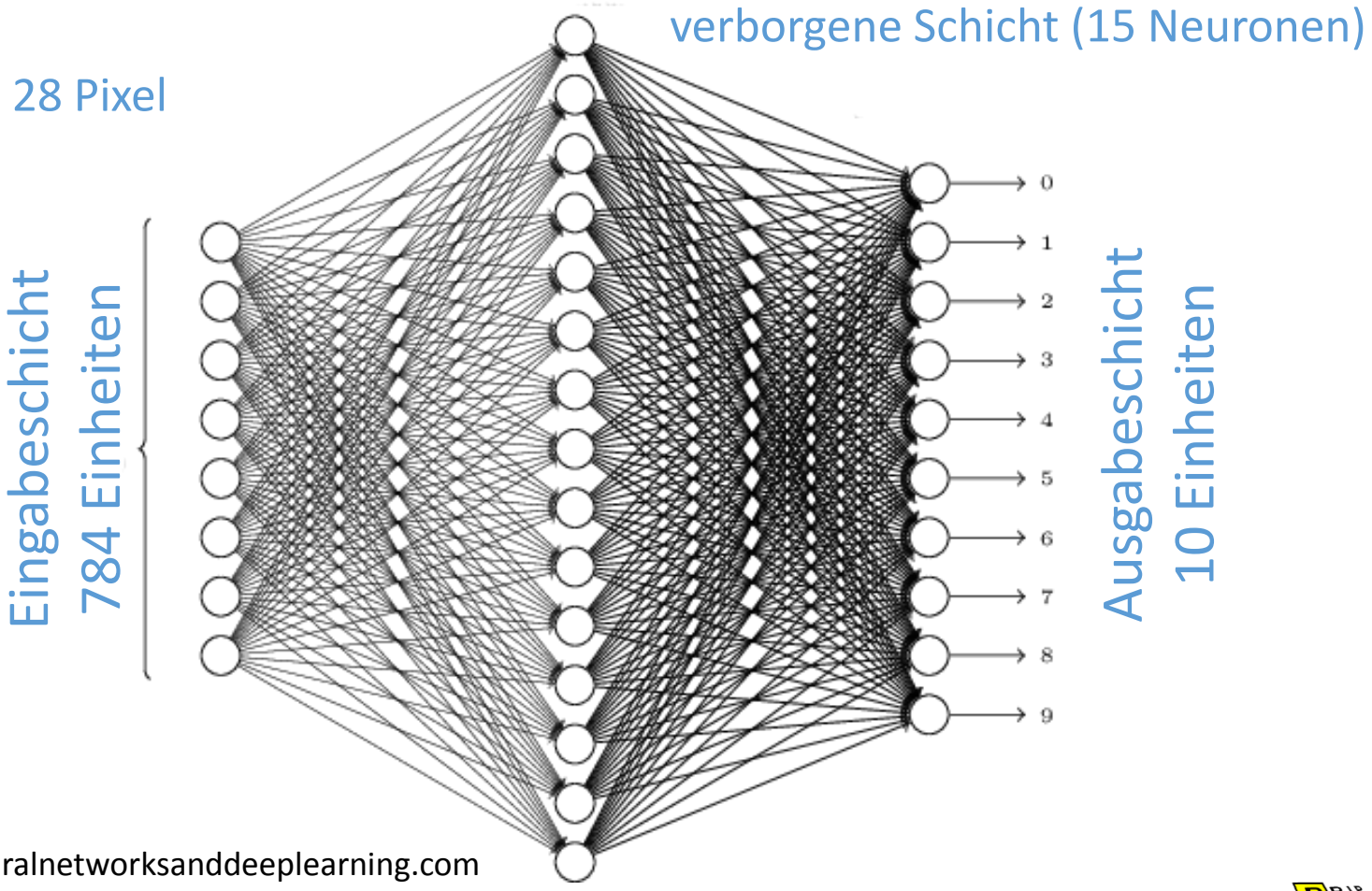
Für Menschen ist diese Aufgabe sehr einfach, aber in Computer Vision (maschinelles Sehen) ist Segmentierung ein noch immer ungelöstes Problem.

Quelle: http://neuralnetworksanddeeplearning.com

Handschrifterkennung

2. Problem: Design neurales Netz

28 x 28 Pixel

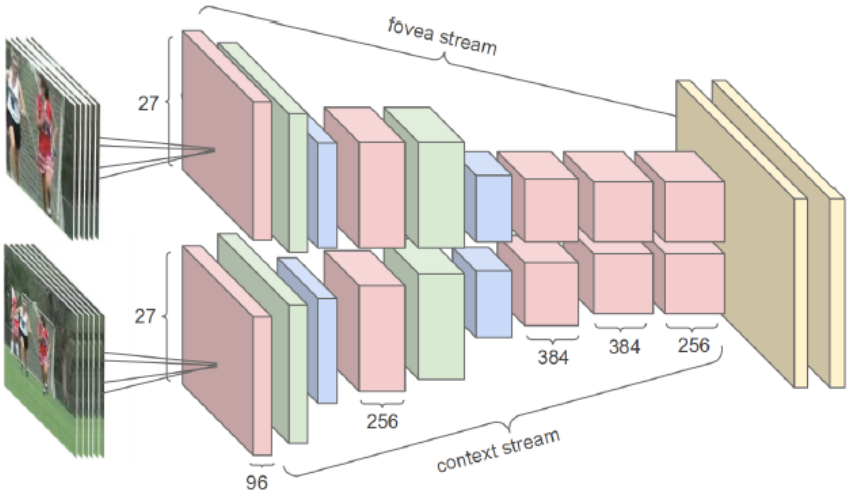


Quelle: http://neuralnetworksanddeeplearning.com

Convolutional Neural Networks

„Large-scale Video Classification with Convolutional Neural Networks“ präsentiert bei CVPR2014 von Karpathy et al.

- Trainingsdatensatz: 1 Million YouTube Videos
- 487 Klassen
- Millionen von Parameter!



Quelle: http://cs.stanford.edu/people/karpathy/deepvideo
Präsentation: http://techtalks.tv/talks/large-scale-video-classification-with-convolutional-neural-networks-2/60272/

Convolutional Neural Networks

Quelle: http://cs.stanford.edu/people/karpathy/deepvideo