ENDOWED CHAIR OF THE HERTIE FOUNDATION
Knowledge and Data Engineering
DEPARTMENT OF MATHEMATICS & COMPUTER SCIENCE

Vorlesung Künstliche Intelligenz Wintersemester 2006/07

Teil III: Wissensrepräsentation und Inferenz

Kap.5: Neuronale Netze

- Dieses Kapitel basiert auf Material von Andreas Hotho
- Mehr Details sind in der Vorlesung „Neuronale Netze“ von Prof. Werner zu finden.

Übersicht

1. Einführung & Grundbegriffe
   - Motivation & Definition
   - Vorbild Biologie
   - Historie der NN
   - Überblick über verschiedene Netzwerktypen
2. Einfaches Perzeptron
3. Multi-Layer-Perzeptron

Was sind künstliche Neuronale Netze?

Künstliche Neuronale Netze sind

- massiv parallel verbundene Netzwerke aus
- einfachen (üblicherweise adaptiven) Elementen in
- hierarchischer Anordnung oder Organisation,

die mit der Welt in der selben Art wie biologische Nervensysteme interagieren sollen.

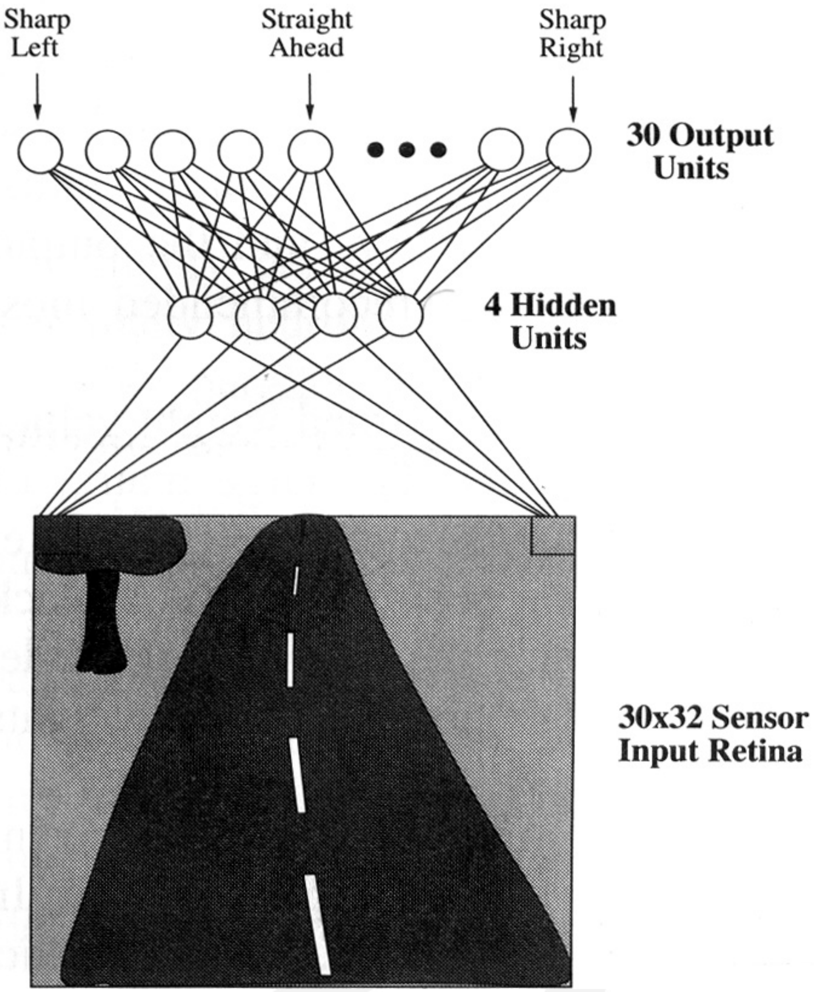
(Kohonen 84)

ENDOWED CHAIR OF THE HERTIE FOUNDATION

Wofür nutzt man künstliche Neuronale Netze?

- Forschung:
  - Modellierung & Simulation biologischer neuronaler Netze
  - Funktionsapproximation
  - Speicherung von Informationen
  - ...

- Anwendungen (z.B.):
  - Interpretation von Sensordaten
  - Prozeßsteuerung
  - Medizin
  - Elektronische Nase
  - Schrifterkennung
  - Risikomanagement
  - Zeitreihenanalyse und -prognose
  - Robotersteuerung



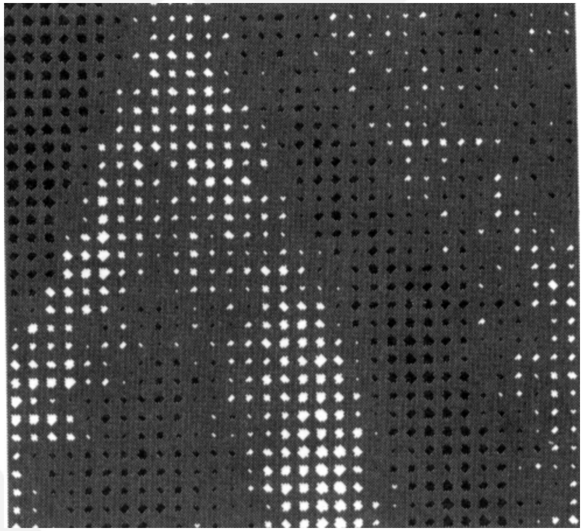


Figure 1: Neural Network learning to steer an autonomous vehicle (Mitchell 1997)

Charakteristische Eigenschaften von Problemen, die mit BACKPROPAGATION-ANNs gelöst werden können:

- Trainingsbeispiele sind repräsentiert durch Attribut-Wert-Paare
  - Werte können reellwertig sein
- \+ zu generierende Funktion (target function) kann ...
  - diskrete Funktionswerte
  - reellwertige Funktionswerte oder
  - Vektor solcher Funktionswerte

  haben
- \+ Trainingsbeispiele dürfen fehlerhaft sein
- – lange Trainingszeiten sind akzeptabel
- \+ schnelle Berechnung der Funktionswerte der gelernten Funktion kann erforderlich sein
- – für Menschen verständliche Interpretation der gelernten Funktion ist nicht wichtig ("Black-Box-Ansatz")

Arbeitsweise Neuronale Netze

gekennzeichnet durch:

- massiv parallele Informationsverarbeitung
- Propagierung der Informationen über Verbindungsstellen (Synapsen)
- verteilte Informationsspeicherung
- Black-Box-Charakter

Es werden

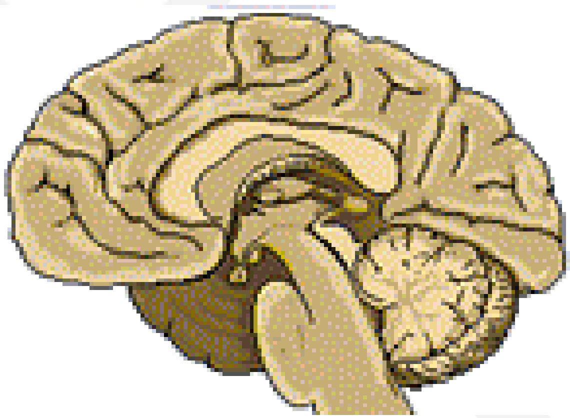
- Aufbauphase (Topologie),
- Trainingsphase (Lernen) und
- Arbeitsphase (Propagation)

unterschieden.

(Die Phasen können auch überlappen.)

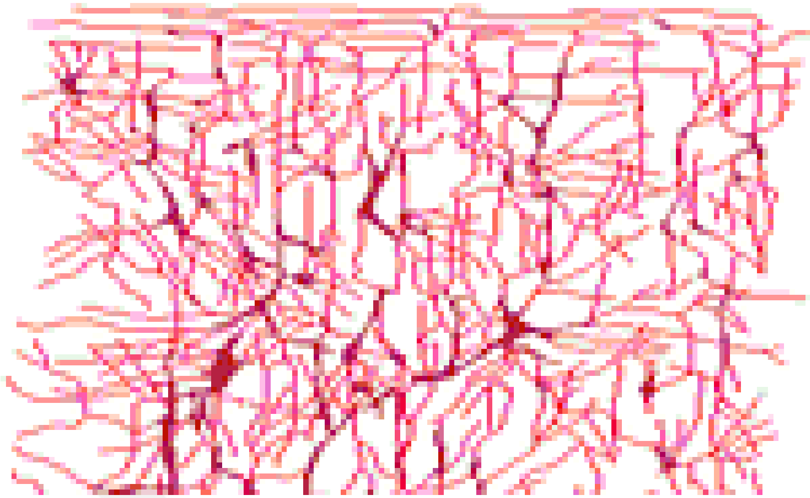
Vorbild Biologie

Gehirn



In der Vergangenheit wurden Forscher aus den verschiedensten Fachgebieten durch das biologische Vorbild motiviert.

Geflecht aus Neuronen

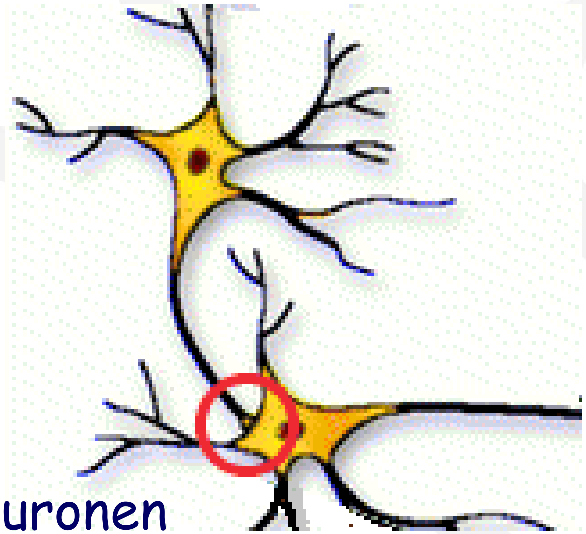


Das Gehirn besteht aus

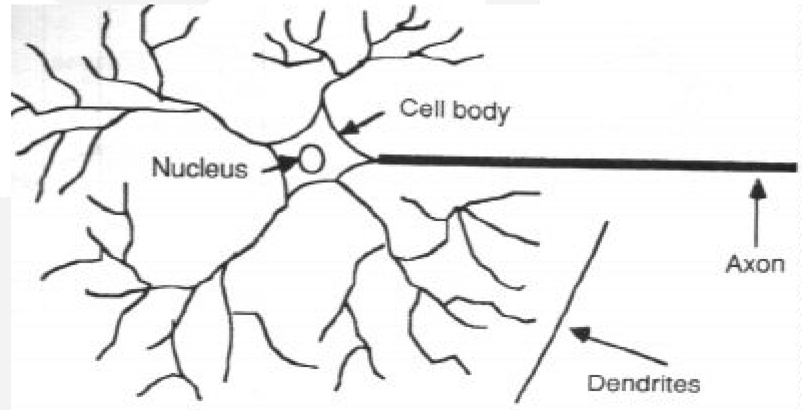
- ca. 10^{11} Neuronen, die mit
- ca. 10^{4} anderen Neuronen
- durch ca. 10^{13} Synapsen

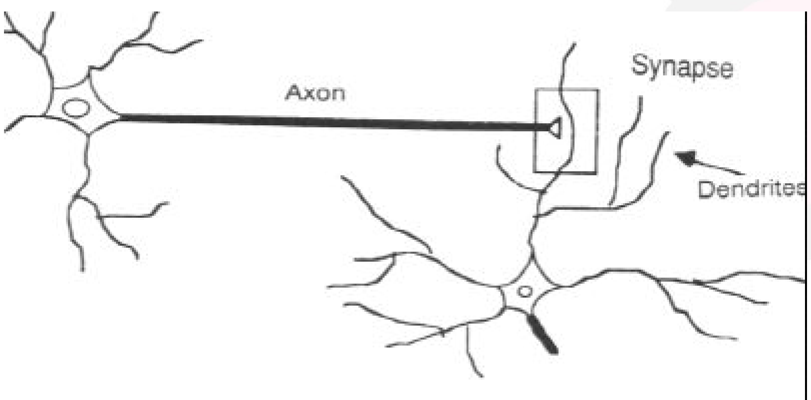
verschaltet sind.

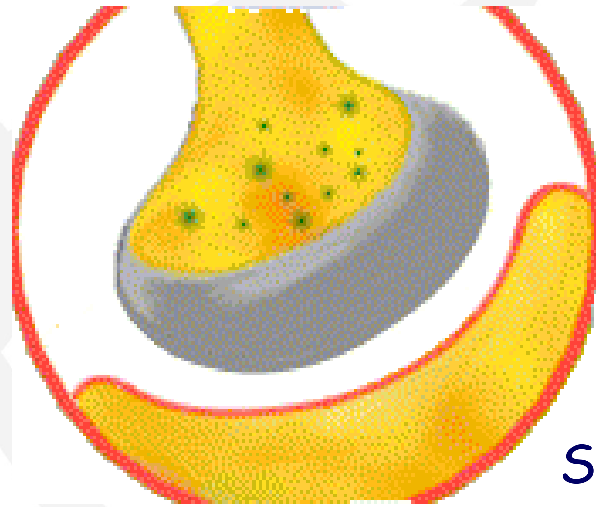
Vorbild Biologie



Neuronen



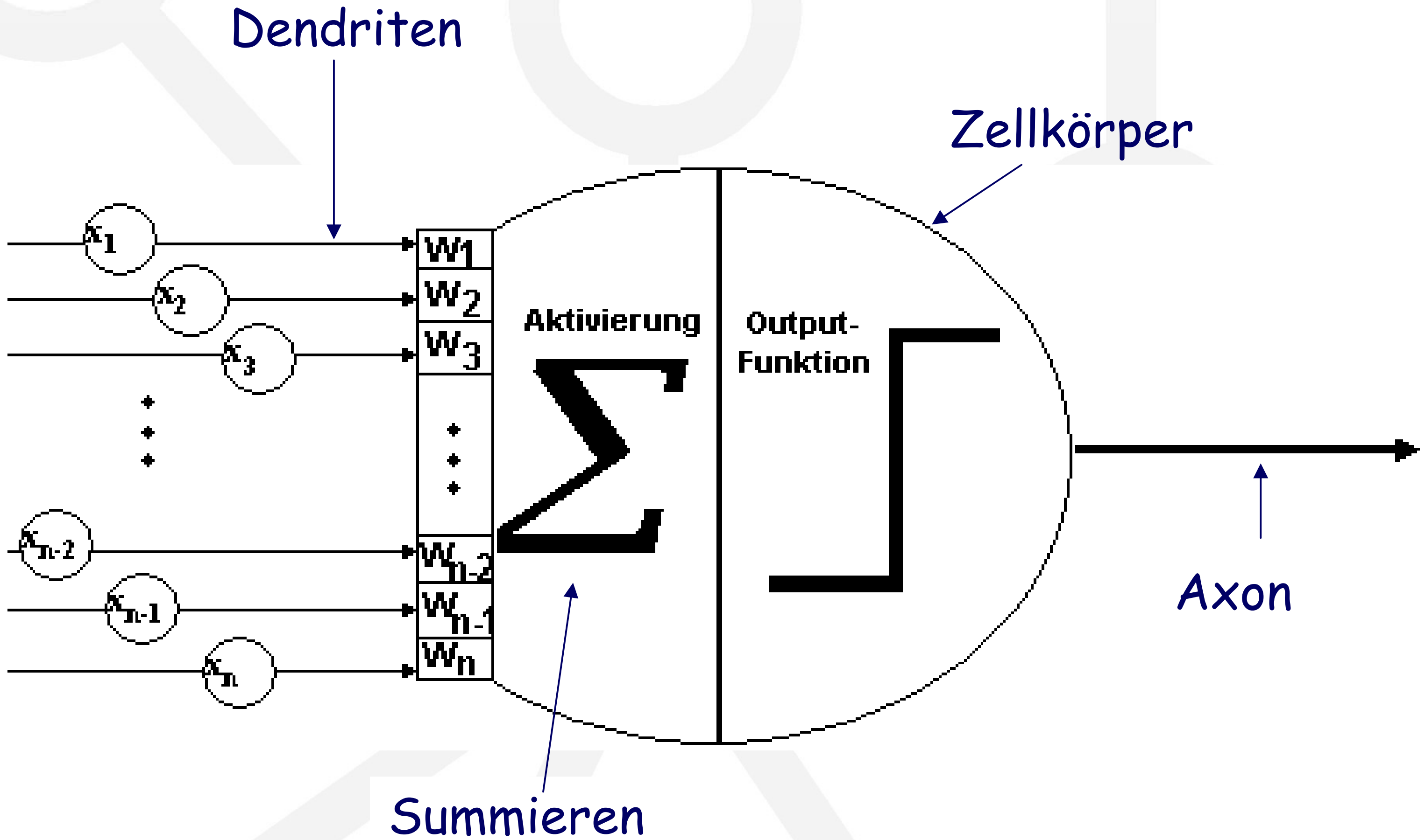




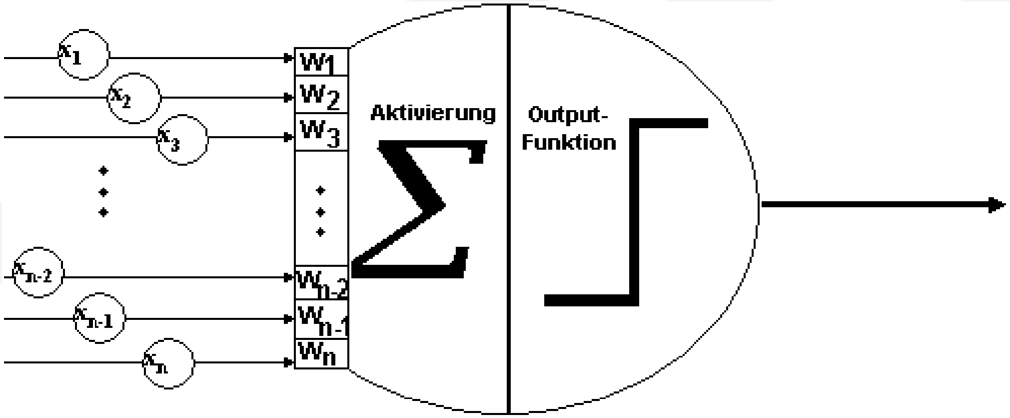
Synapse

Vorbild Biologie

Vom natürlichen zum künstlichen Neuronalen Netz



Aktivierungsfunktionen



Es gibt mehrere Arten der Aktivierung von Neuronen:

Allen gemeinsam ist die **Gewichtung** der Inputs. Die Inputs $x_1,x_2,...,x_n$ werden stets mit ihren (Synapsen-)Gewichten $w_1,w_2,...,w_n$ multipliziert: $a_1=x_1*w_1$, $a_2=x_2*w_2$,..., $a_n=x_n*w_n$.

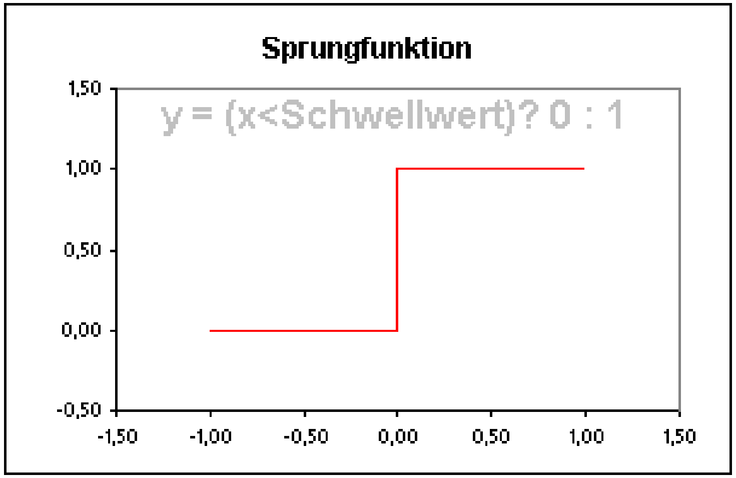
Aktivierungsfunktionen

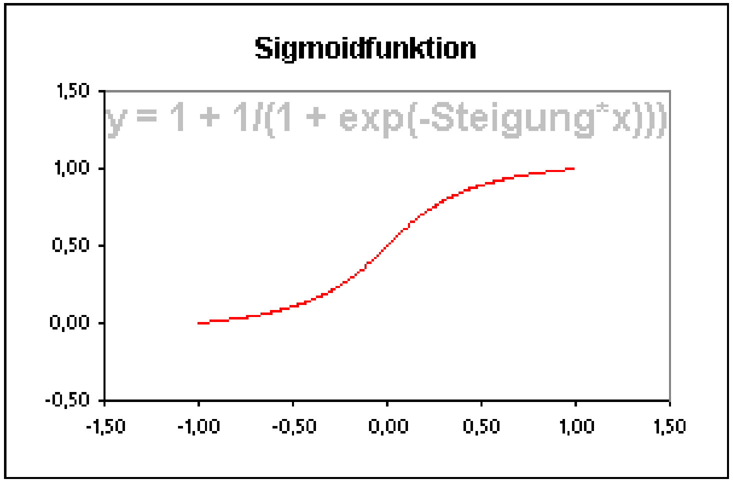
Allen gemeinsam ist die **Gewichtung** der Inputs. Die Inputs $x_1,x_2,\ldots,x_n$ werden stets mit ihren (Synapsen-)Gewichten $w_1,w_2,\ldots,w_n$ multipliziert: $a_1=x_1*w_1$, $a_2=x_2*w_2,\ldots$, $a_n=x_n*w_n$.

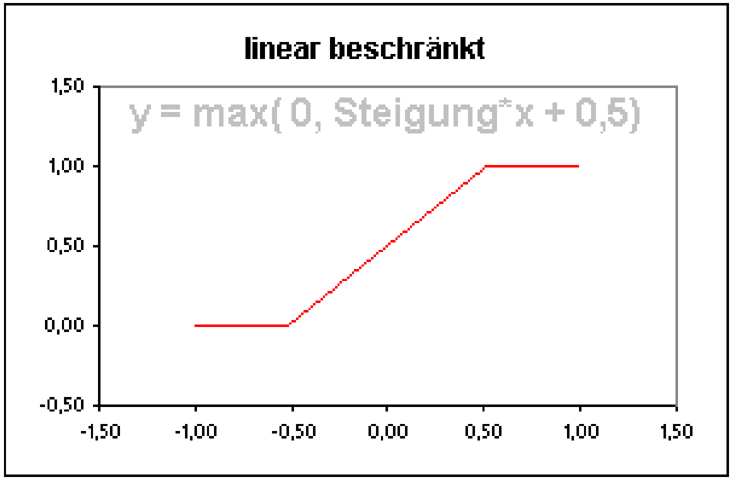
1. Die am häufigsten angewandte Regel ist die **Skalarprodukt-Regel:** Die gewichteten Inputs $a_1,a_2, \ldots,a_n$ werden zur Aktivität des Neurons aufaddiert:
   $a = a_1+a_2+\ldots+a_n$

2. Sehr häufig ist ebenfalls die **Winner-take-all-Regel:** bei der die Aktivität a zunächst nach der Skalarproduktregel ermittelt wird, dann aber mit allen Aktivitäten in derselben Schicht verglichen wird und auf 0 herabgesetzt wird, wenn ein anderes Neuron höhere Aktivität hat.

Outputfunktionen







Alle Funktionen könnten wie die Sprungfunktion mit einem Schwellwert verschoben werden. Wie wir aber sehen werden kann auf diese Verschiebung verzichtet werden.

Overfitting

- Mit Overfitting beschreibt man das Problem der Überanpassung eines Modells an einen Trainings-Datensatz. Das Modell paßt sich sehr gut an den kleinen Weltausschnitt an, kann aber wegen der fehlenden Generalisierung nur schlecht auf neue Situationen reagieren.

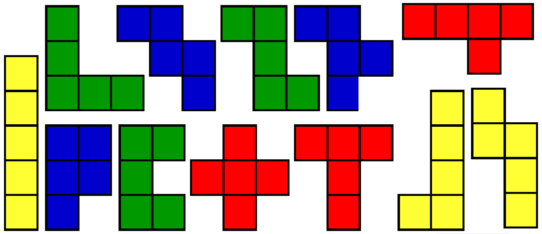
- Kontrolle der Generalisierung während des Trainings durch Teilen des Datensatzes:
  - Trainings-Datensatz (Output bekannt)
  - Validierungs-Datensatz (Output bekannt)
  - Anwendungsdaten (Output unbekannt)

- (Zur Evaluierung des Verfahrens insgesamt verwendet man auch in der Anwendung Daten mit bekanntem Output (den „Testdatensatz“), um gewünschten und errechneten Output vergleichen zu können.)

Vor- und Nachteile Neuronaler Netze

Vorteile

- sehr gute Mustererkenner



- verarbeiten verrauschte, unvollständige und widersprüchliche Inputs
- verarbeiten multisensorischen Input (Zahlen, Farben, Töne, ...)
- erzeugen implizites Modell für Eingaben (ohne Hypothesen des Anwenders)
- fehlertolerant auch gegenüber Hardwarefehlern
- leicht zu handhaben

Vor- und Nachteile Neuronaler Netze

Nachteile

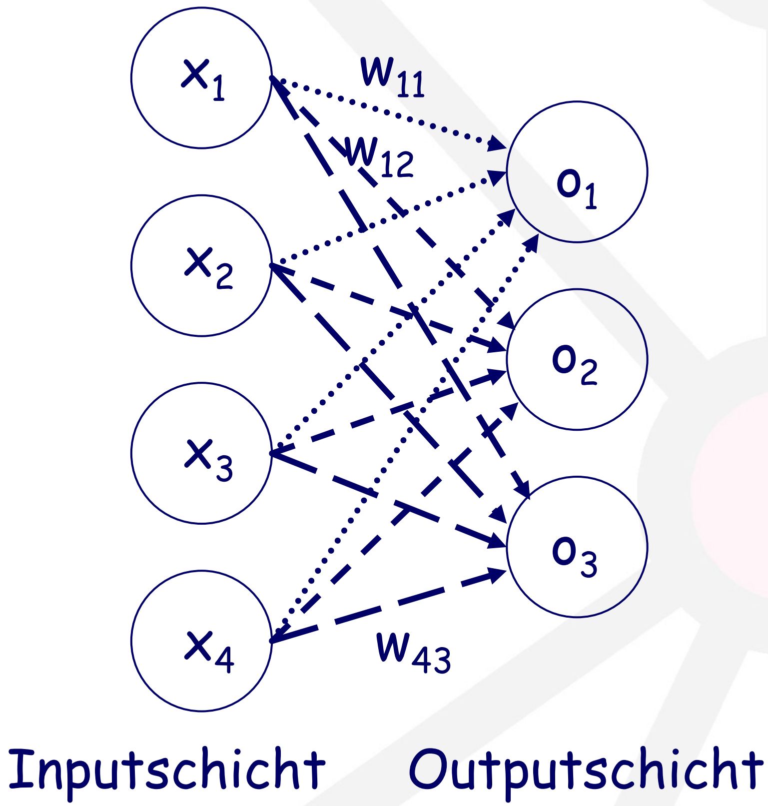
- lange Trainingszeiten
- Lernerfolg kann nicht garantiert werden
- Generalisierungsfähigkeit kann nicht garantiert werden (Overfitting)
- keine Möglichkeit, die Begründung einer Antwort zu erhalten (Blackbox)

Agenda

1. Einführung
2. Einfaches Perzeptron
   - Motivation
   - Definition Perzeptron
   - Geometrische Interpretation
   - Lernen: Delta-Regel
   - XOR-Problem
3. Multi-Layer-Perzeptron

Perzeptron

- jedes Outputneuron hat einen eigenen unabhängigen Netzbereich



- d.h., für weitere Betrachtungen genügt ein Netz mit einem Neuron in der Outputschicht

- Input $\boldsymbol{x} = (x_1, \ldots, x_n)$
- Gewichte $\boldsymbol{w} = (w_1, \ldots, w_n)$
- Output (o)

Perzeptron

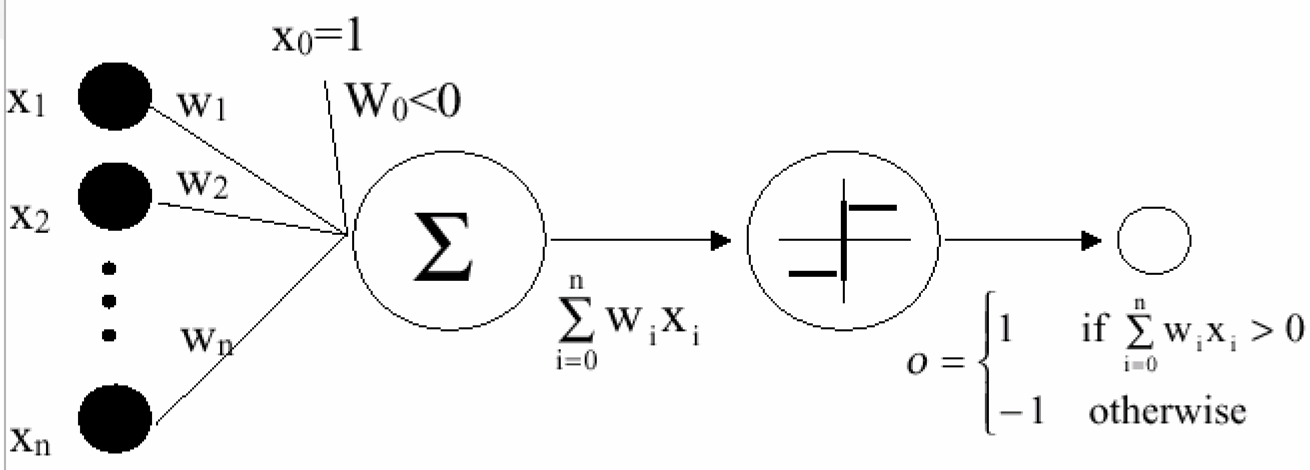


Figure 2: A Perceptron (Mitchell 1997)

- Input $\boldsymbol{x} = (x_1, \ldots, x_n)$
- Gewichte $\boldsymbol{w} = (w_1, \ldots, w_n)$
- Output (o)

Perzeptron

- Outputneuron hat:
  - Schwellenwert s
  - Aktivität $a := \boldsymbol{xw} := x_1 w_1 + \ldots + x_n w_n$ (Skalarproduktregel)
  - verwendet Sprungfunktion
- Output berechnet sich wie folgt:

  $o = 0$, falls $a < s$

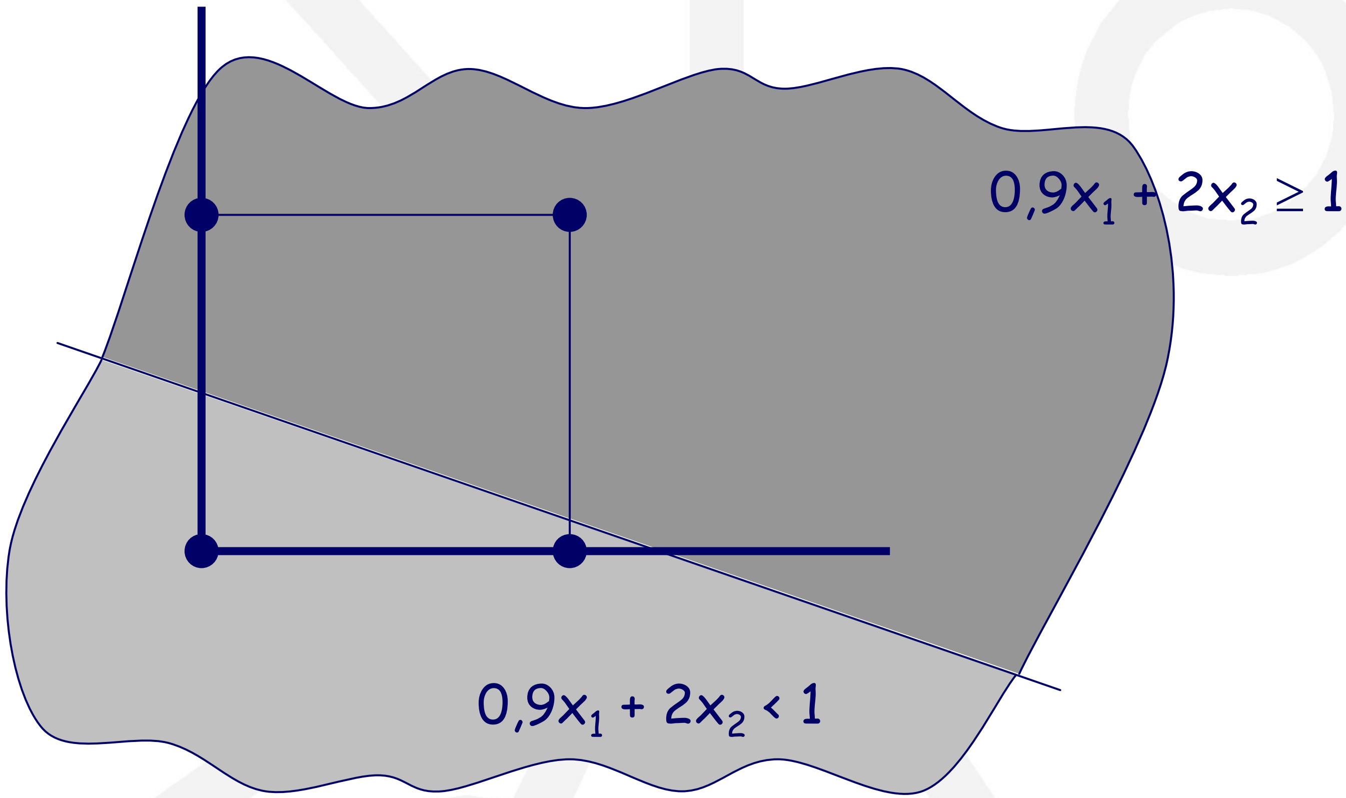
  $o = 1$, sonst.
- äquivalente Beschreibung: erweitert man die Vektoren $\boldsymbol{x}$ und $\boldsymbol{w}$ zu $\boldsymbol{y} = (x_1, \ldots, x_{n,1})$ und $\boldsymbol{v} = (w_1, \ldots, w_n, s)$, so ist

  $o = 0$, falls $\boldsymbol{yv} = x_1 w_1 + \ldots + x_n w_n - 1s < 0$

  $o = 1$, sonst

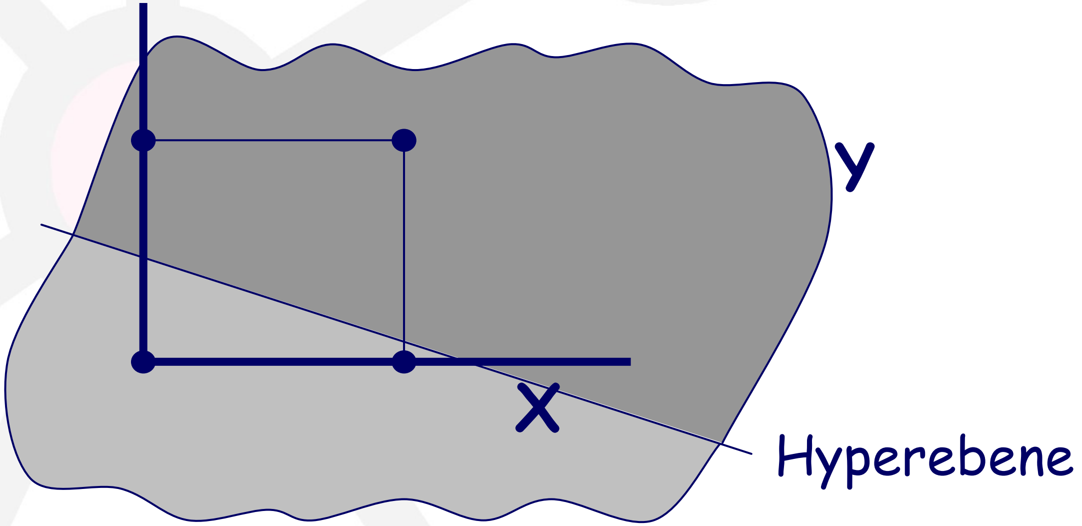
Geometrische Interpretation

- Gleichung ***xw*** = s beschreibt eine Hyperebene im n -dimensionalen Raum
- Beispiel: $0{,}9x_1 + 2x_2 = 1$



Lernen der Gewichte

- Gegeben ist eine Menge von Beispielen.
- Überwachte Lernaufgabe: Daten sind disjunkt in zwei Mengen X,Y geteilt
- Gesucht ist der Gewichtsvektor ***w*** $(w_1, \ldots, w_n)$, so dass eine Hyperebene spezifiziert wird, die die Mengen X und Y voneinander trennt.
- Mengen müssen linear trennbar sein, damit die Aufgabe lösbar ist.



ENDOWED CHAIR OF THE HERTIE FOUNDATION

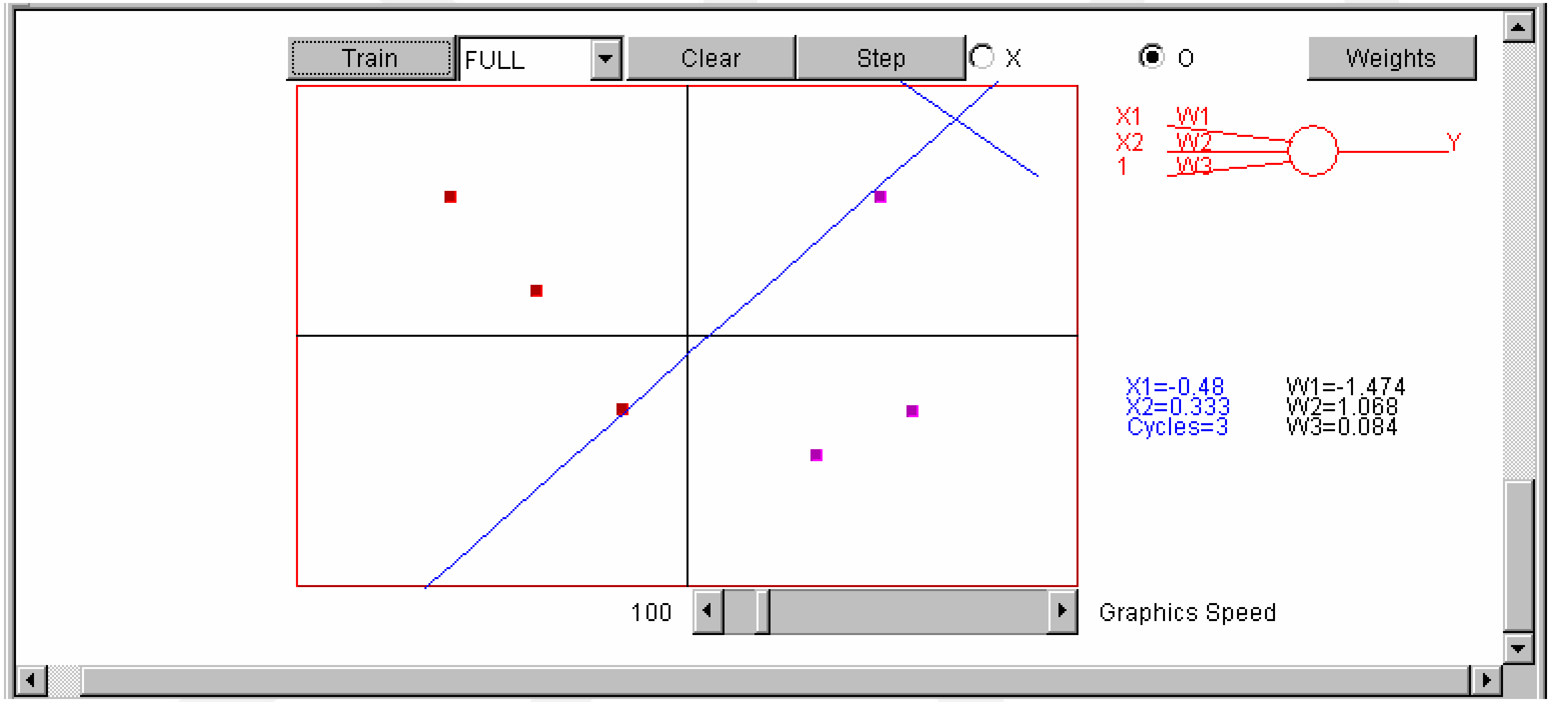
Delta-Regel

Menge X

x_1	x_2
0,552	-0,605
0,176	-0,384
-0,296	-0,164

Menge Y

x_1	x_2
0,552	0,497
-0,304	0,579
-0,48	0,333



→ Demo (New38.html)

Delta-Regel

- Beim Training werden die Beispiele dem Netz als Input präsentiert.
- Output ist für die Beispiele bekannt
  --> überwachte Lernaufgabe (supervised)

  (hier: liegt Beispiel in X oder Y?)
- Soll und Ist-Output werden verglichen.
  Bei Diskrepanz werden Schwellenwert und Gewichte nach folgender Delta-Regel angepasst:

$$w_{i,neu} = w_{i,alt} + \eta x_i * (Output_{soll} - Output_{ist})$$

(mit $w_0 = -s$, $x_0 = 1$)

Lernrate

Delta-Regel

Annahme hier: - Algorithmus mit Lernrate $\eta = 1$
- als Output nur 0 und 1 möglich (d.h. Trennung von zwei Klassen wird gelernt)

Start: Der Gewichtsvektor $\boldsymbol{w}_0$ wird zufällig generiert. Setze t:=0.

Testen: Ein Punkt $\boldsymbol{x}$ in $X \cup Y$ wird zufällig gewählt
Falls $\boldsymbol{x} \in X$ und $\boldsymbol{w}_t \cdot \boldsymbol{x} > 0$ gehe zu Testen
Falls $\boldsymbol{x} \in X$ und $\boldsymbol{w}_t \cdot \boldsymbol{x} \leq 0$ gehe zu Addieren
Falls $\boldsymbol{x} \in Y$ und $\boldsymbol{w}_t \cdot \boldsymbol{x} < 0$ gehe zu Testen
Falls $\boldsymbol{x} \in Y$ und $\boldsymbol{w}_t \cdot \boldsymbol{x} \geq 0$ gehe zu Subtrahieren

Addieren: Setze $\boldsymbol{w}_{t+1} = \boldsymbol{w}_t + \boldsymbol{x}$.
Setze t := t + 1. Gehe zu Testen

Subtrahieren: Setze $\boldsymbol{w}_{t+1} = \boldsymbol{w}_t - \boldsymbol{x}$.
Setze t := t + 1. Gehe zu Testen

Delta-Regel - Beispiel

t *x* *X/Y Vhs. Error* *zu_addieren/subtr.* *neue_Gewichte*

	i	t	a_v	e	$\Delta W(u_1,v)$	$\Delta W(u_2,v)$	$\Delta\theta$	$W(u_1,v)$	$W(u_2,v)$	θ
1. Epoche	0 0	0	0	0	0	0	0	0	0	0
	0 1	0	0	0	0	0	0	0	0	0
	1 0	0	0	0	0	0	0	0	0	0
	1 1	1	0	1	1	1	−1	1	1	−1
2. Epoche	0 0	0	1	−1	0	0	1	1	1	0
	0 1	0	1	−1	0	−1	1	1	0	1
	1 0	0	0	0	0	0	0	1	0	1
	1 1	1	0	1	1	1	−1	2	1	0
3. Epoche	0 0	0	0	0	0	0	0	2	1	0
	0 1	0	1	−1	0	−1	1	2	0	1
	1 0	0	1	−1	−1	0	1	1	0	2
	1 1	1	0	1	1	1	−1	2	1	1
4. Epoche	0 0	0	0	0	0	0	0	2	1	1
	0 1	0	0	0	0	0	0	2	1	1
	1 0	0	1	−1	−1	0	1	1	1	2
	1 1	1	0	1	1	1	−1	2	2	1
5. Epoche	0 0	0	0	0	0	0	0	2	2	1
	0 1	0	1	−1	0	−1	1	2	1	2
	1 0	0	0	0	0	0	0	2	1	2
	1 1	1	1	0	0	0	0	2	1	2
6. Epoche	0 0	0	0	0	0	0	0	2	1	2
	0 1	0	0	0	0	0	0	2	1	2
	1 0	0	0	0	0	0	0	2	1	2
	1 1	1	1	0	0	0	0	2	1	2

entnommen Nauk, Kruse, S. 50

Konvergenz und Korrektheit der Delta-Regel

Satz: Wenn das Perzeptron eine Klasseneinteilung überhaupt lernen kann, dann lernt es diese mit der Delta-Regel in endlich vielen Schritten.

Problem: Falls das Perzeptron nicht lernt, kann nicht unterschieden werden, ob nur noch nicht genügend Schritte vollzogen wurden oder ob das Problem nicht lernbar ist. (Es gibt keine obere Schranke für die Lerndauer.)

Konvergenz und Korrektheit der Delta-Regel

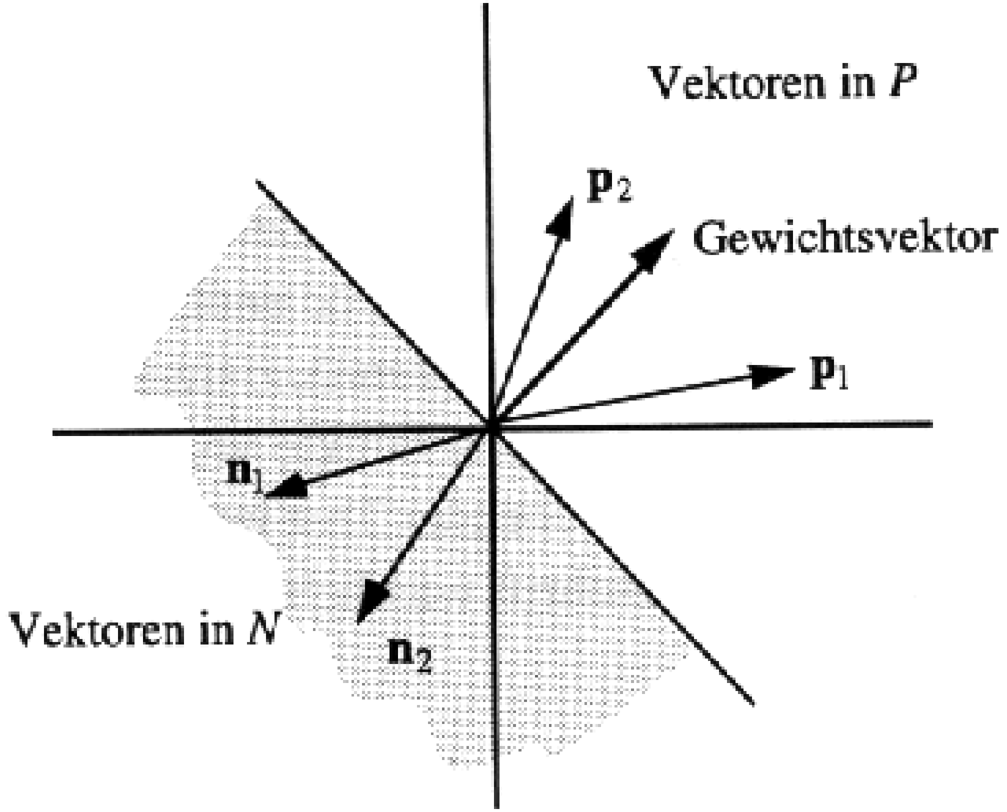


Abb. 4.8 Positiver und negativer Halbraum im Eingaberaum

entnommen Rojas, S. 84

ENDOWED CHAIR OF THE HERTIE FOUNDATION

Konvergenz und Korrektheit der Delta-Regel

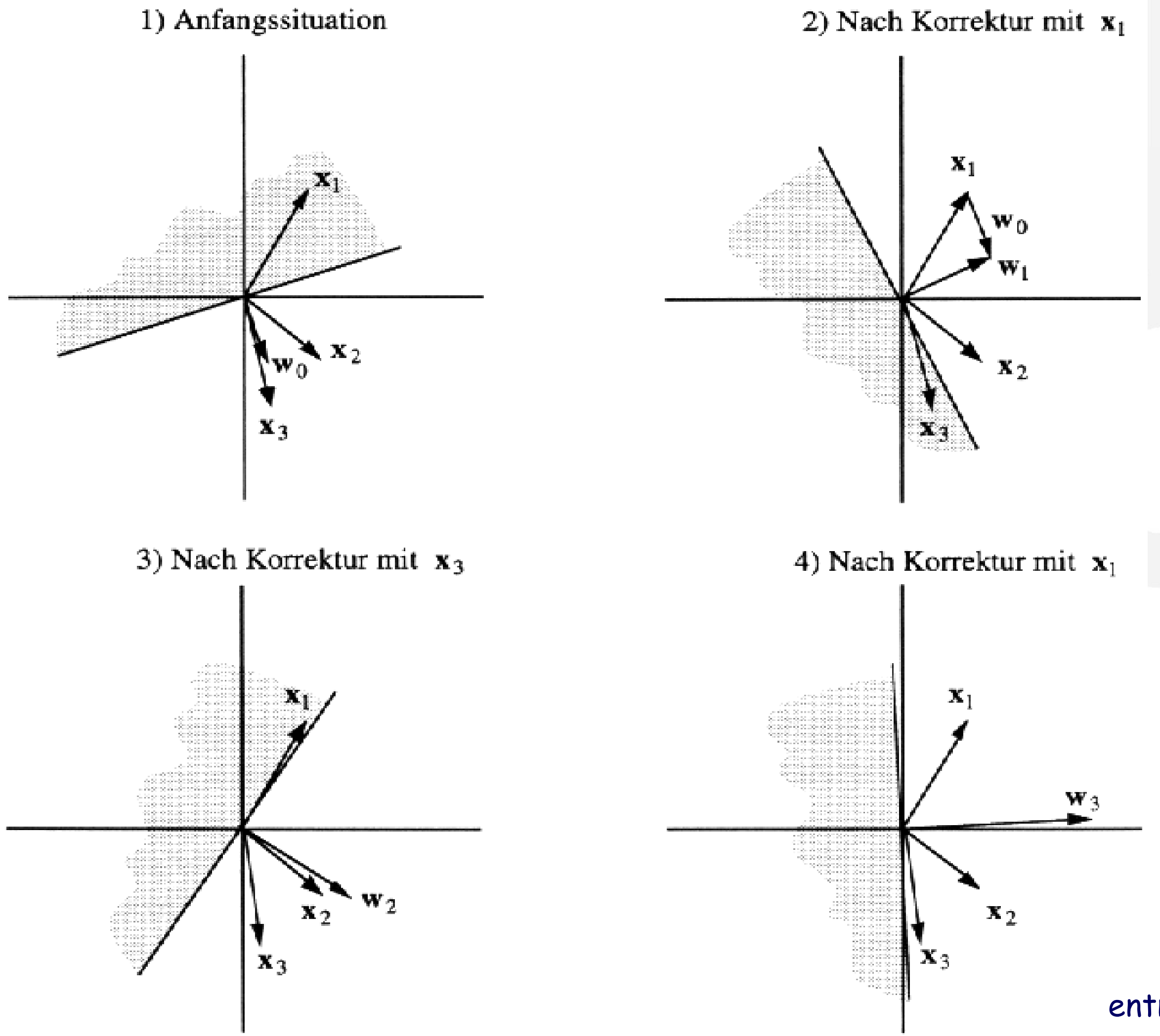
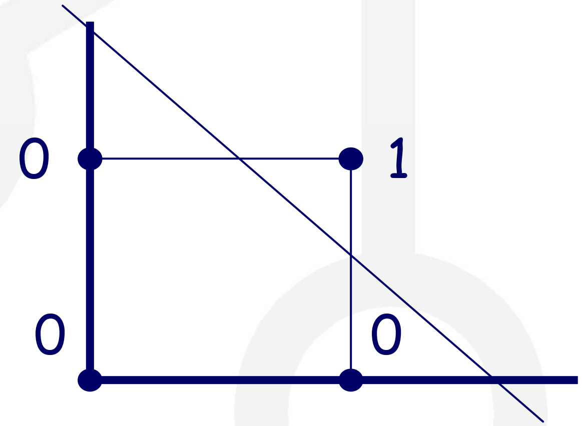


Abb. 4.9 Konvergenzverhalten des Lernalgorithmus

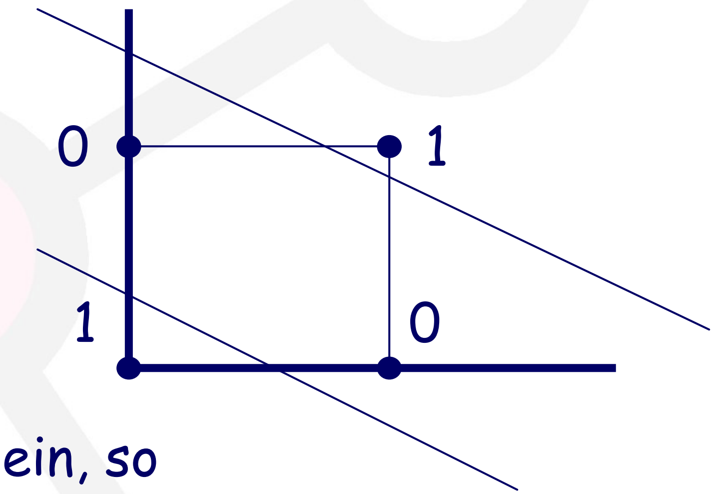
entnommen Rojas, S. 85

XOR-Problem

Logisches AND ist linear separierbar



Logisches XOR ist nicht linear separierbar



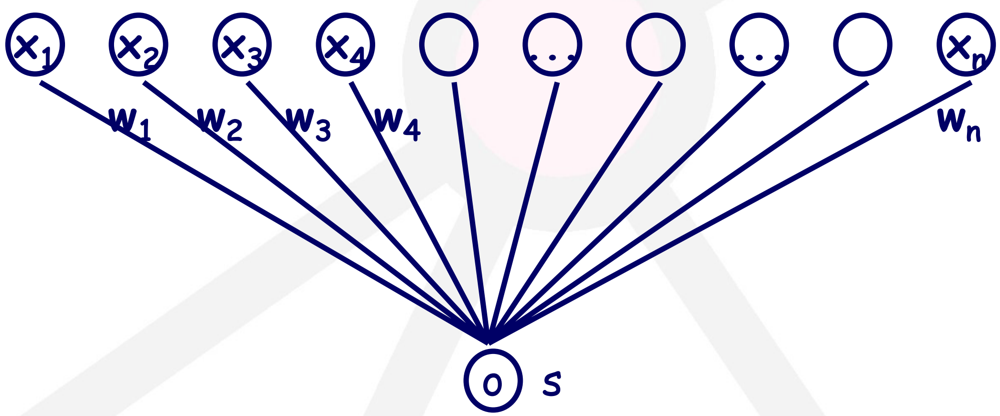
Führt man weitere Hyperebenen ein, so kann man auch hier die Klassen unterscheiden. → Realisierung durch Zwischenschichten

Agenda

Erinnerung Perzeptron

$\Delta w_i := \eta \cdot x_i \cdot e$ (Synapsenveränderung)

- Schwellwert: s
- Netz-Output: $o = \theta(x_1w_1+x_2w_2+x_3w_3+\ldots+x_nw_n-s)$
- erwarteter Output: t (target)
- gemachter Fehler: $e = t - o$ (error)
- Lernkonstante : η



Vektorschreibweise der Delta-Regel

Aktivität der Input-Schicht:

$$\underline{\mathbf{x}} = (x_1, x_2, x_3, \ldots, x_n, 1)$$

Gewichtsvektor (einschl. Schwellwert):

$$\underline{\mathbf{w}} = (w_1, w_2, w_3, \ldots, w_n, -s)$$

Aktivität des Ausgabeneurons:

$$o = \theta(\underline{\mathbf{x}} \cdot \underline{\mathbf{w}})$$

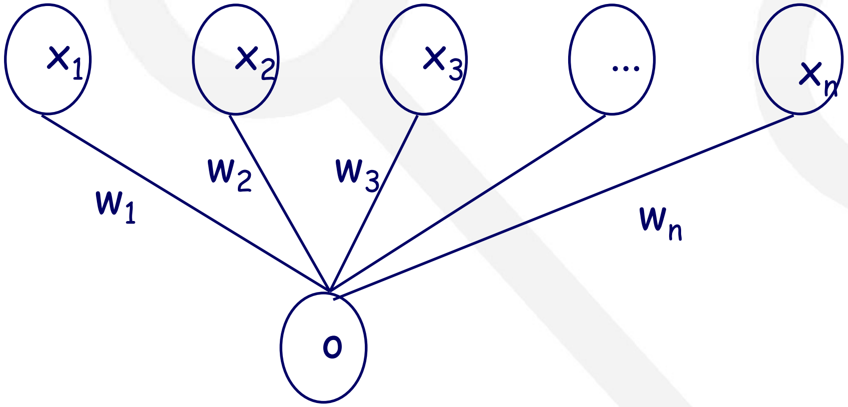
Fehler des Ausgabeneurons (t = erwarteter Wert):

$$e = t - o$$

Gewichtsänderung:

$$\Delta\underline{\mathbf{w}} := \eta \cdot e \cdot \underline{\mathbf{x}}$$

Delta-Regel als Ableitungsregel für Perzeptron



$o = \theta(\underline{\mathbf{x}} \cdot \underline{\mathbf{w}})$

Fehlergradient:

$$F = (o - t)^2 = (\theta(\underline{\mathbf{x}} \cdot \underline{\mathbf{w}}) - t)^2$$

$$\begin{aligned} \partial F / \partial w_i &= \partial(o - t)/ \partial w_i \\ &= \partial(\theta(\underline{\mathbf{x}} \cdot \underline{\mathbf{w}}) - t)^2 / \partial w_i \\ &= \underbrace{2\, \theta'(\underline{\mathbf{x}} \cdot \underline{\mathbf{w}})}_{\eta}\, (o-t)\, x_i \end{aligned}$$

Die Delta-Regel kann als Gradientenabstieg mit (variablem) Lernfaktor interpretiert werden:

$$\Delta w_i = \eta\, (o-t)\, x_i \quad \text{mit} \quad \eta = 2\, \theta'(\underline{\mathbf{x}} \cdot \underline{\mathbf{w}})$$

(unter derAnnahme: θ ist diff.-bar)

2-Layer-Perzeptron

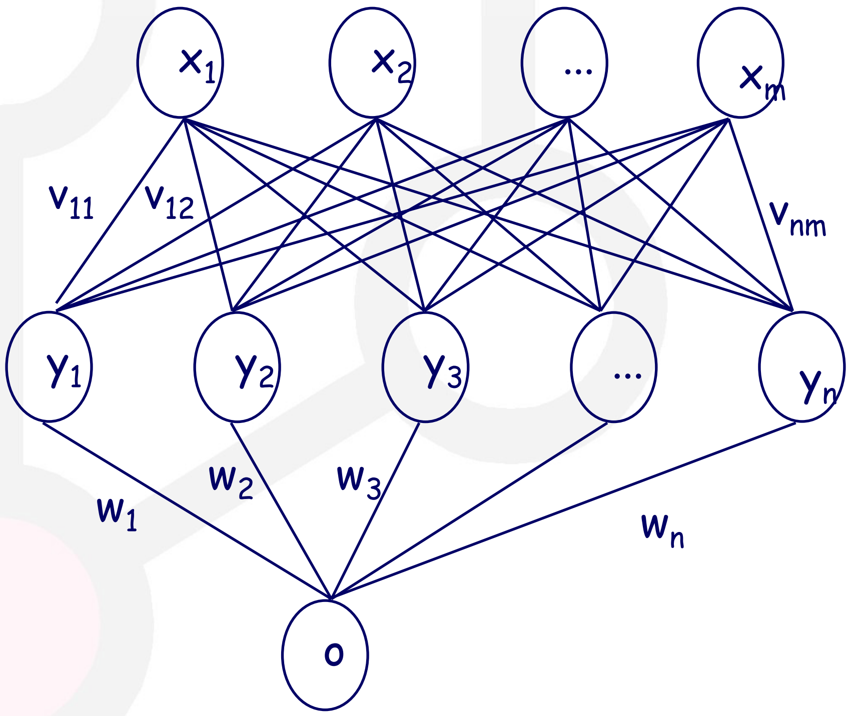
Input-Vektor $\underline{x}$

Gewichtsmatrix $\underline{v}$

Aktivitätsvektor $\underline{y}$

Gewichtsvektor $\underline{w}$

Output o

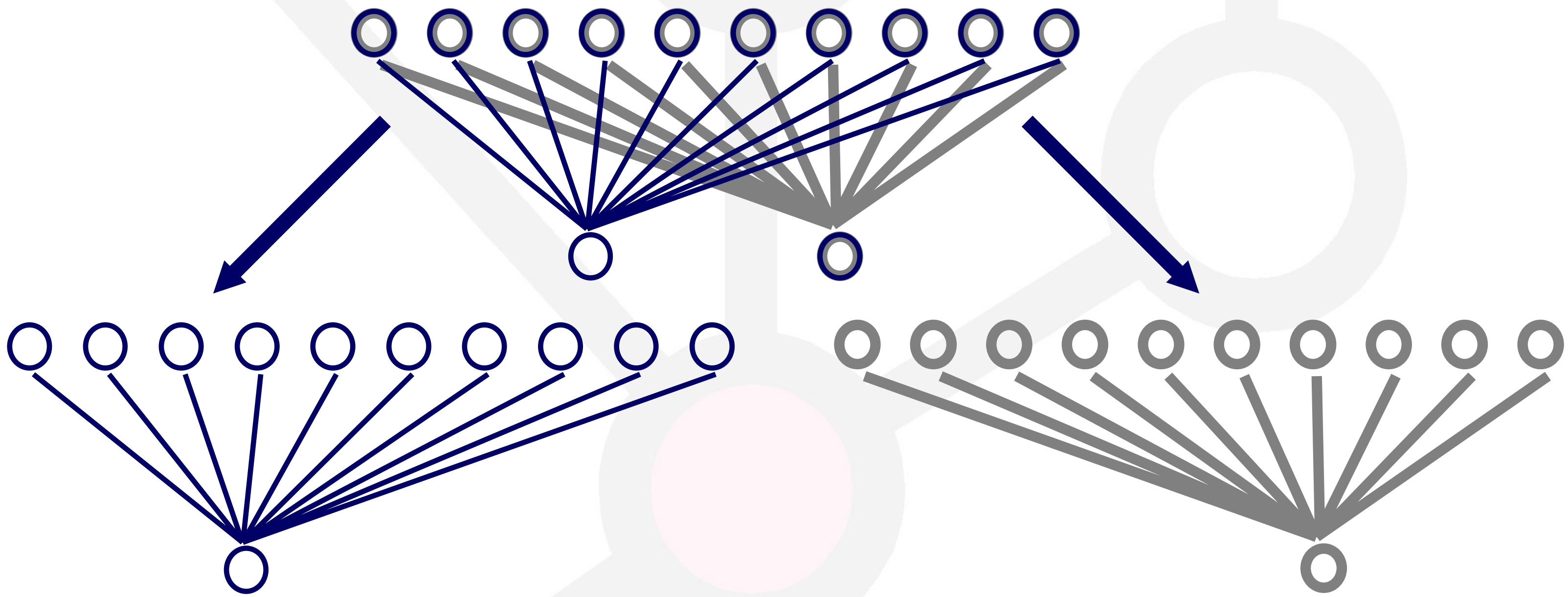


$$\underline{y} = \theta(\underline{v} \cdot \underline{x})$$

$$o = \theta(\underline{w} \cdot \underline{y})$$

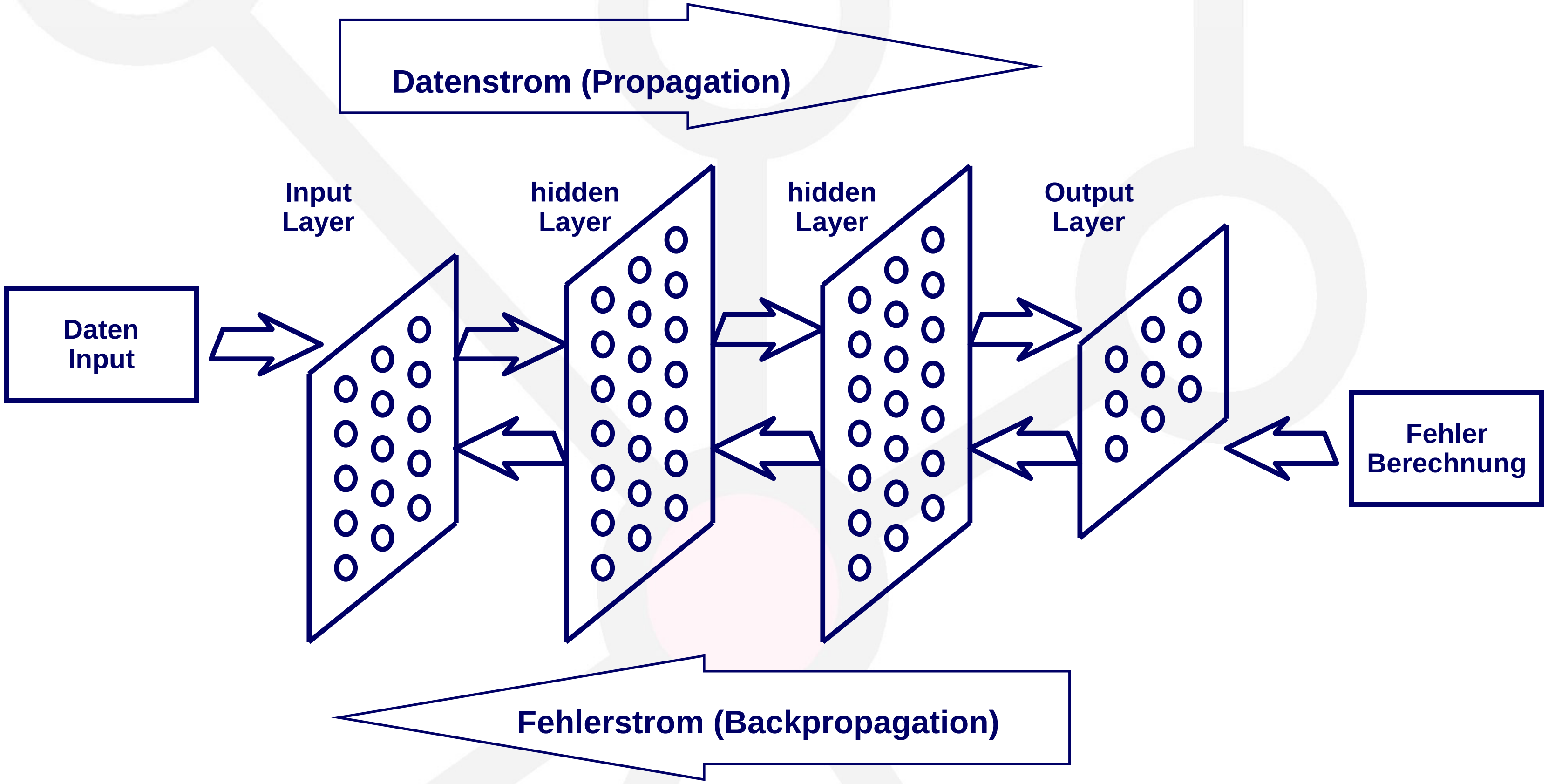
Multi-Layer-Perzeptron

In einem 2-Schichten-Netz beeinflussen die Neuronen der 2. Schicht einander nicht, deshalb können sie voneinander getrennt betrachtet werden.

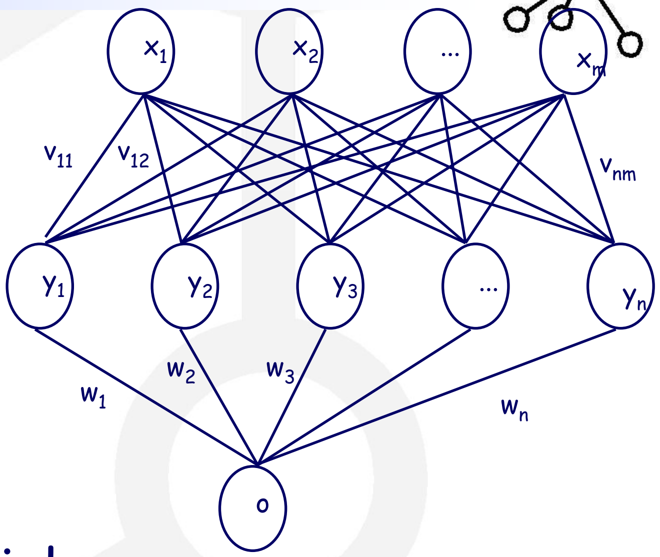


Bei mehrschichtigen Netzen geht die Unabhängigkeit verloren, d.h. sie können nicht so getrennt werden.

Backpropagation



Multi-Layer-Perzeptron



Fehlerfunktion F (mittlerer quadratischer Fehler) für das Lernen:

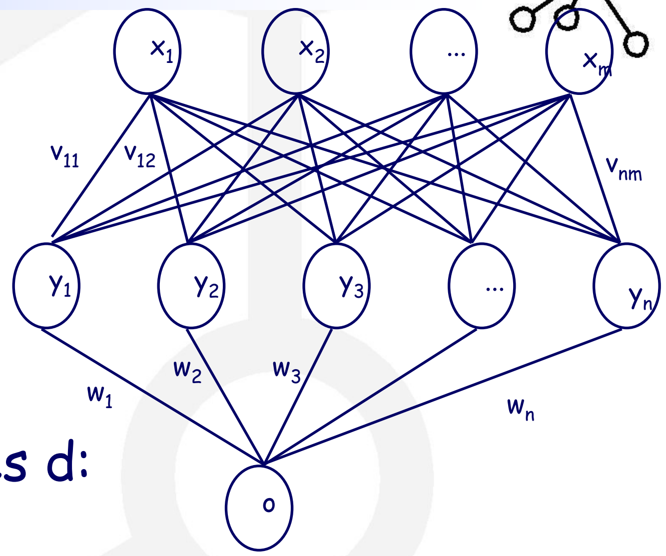
$$F_D = \frac{1}{2}\sum_{d\in D}(t_d - o_d)^2$$

wobei gilt:

D Menge der Trainingsbeispiele
t_d korrekter Output für $d \in D$
o_d berechneter Output für $d \in D$

Die Gewichte müssen so angepasst werden, daß der Fehler minimiert wird. Dazu bietet sich das Gradientenabstiegsverfahren an. (D.h.: Bergsteigerverfahren mit Vektorraum der Gewichtsvektoren als Suchraum!)

Multi-Layer-Perzeptron



Sei nun ein $d \in D$ gegeben.
Anders geschrieben ist

$$F_d = (o - t)^2 \quad = (\theta(\underline{\mathbf{w}} \cdot \underline{\mathbf{y}}) - t)^2$$

Der Fehlergradient für w_i lautet für dieses d:

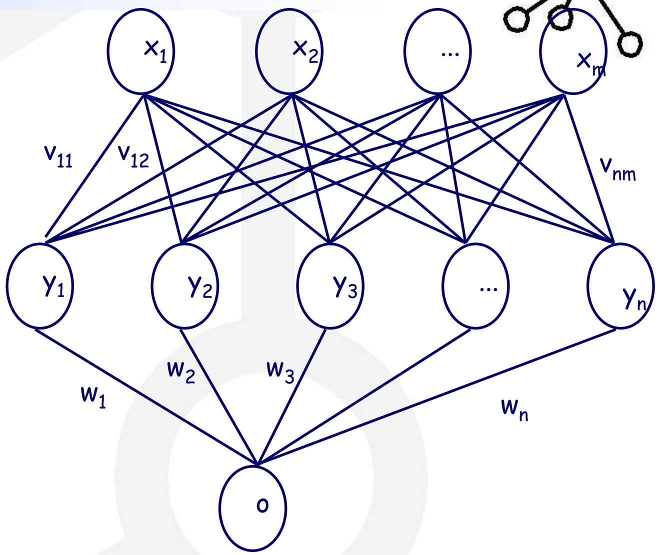
$$\partial F / \partial w_i = \partial(o-t)^2 / \partial w_i$$

$$= \partial(\theta(\underline{\mathbf{w}} \cdot \underline{\mathbf{y}}) - t)^2 / \partial w_i$$

$$= 2 \cdot (o-t) \cdot \theta'(\underline{\mathbf{w}} \cdot \underline{\mathbf{y}}) \cdot \partial(\underline{\mathbf{w}} \cdot \underline{\mathbf{y}}) / \partial w_i$$

$$= 2 \cdot (o-t) \cdot \theta'(\underline{\mathbf{w}} \cdot \underline{\mathbf{y}}) y_i$$

Multi-Layer-Perzeptron



Fehlergradient für v_{ij} lautet:

$$\partial F/\partial v_{ij} = \partial F/\partial y_i \cdot \partial y_i/\partial v_{ij}$$

$$= \partial F/\partial y_i \cdot \partial\ \theta(\underline{v}_i \cdot \underline{x})/\partial v_{ij}$$

(Fehler von Neuron i)

$$= \partial F/\partial y_i \cdot \theta'(\underline{v}_i \cdot \underline{x}) \cdot x_j$$

$$= \partial F/\partial o \cdot \partial o/\partial y_i \cdot \theta'(\underline{v}_i \cdot \underline{x}) \cdot x_j$$

$$= \partial F/\partial o \cdot \partial\theta(\underline{w} \cdot \underline{y})/\partial y_i \cdot \theta'(\underline{v}_i \cdot \underline{x}) \cdot x_j$$

$$= \partial F/\partial o \cdot \theta'(\underline{w} \cdot \underline{y}) \cdot w_i \cdot \theta'(\underline{v}_i \cdot \underline{x}) \cdot x_j$$

$$= 2 \cdot (o-t) \cdot \theta'(\underline{w} \cdot \underline{y}) \cdot w_i \cdot \theta'(\underline{v}_i \cdot \underline{x}) \cdot x_j$$

Fehler bei der Ausgabe

Info von Zwischenschicht

Gewicht

Info von Inputschicht

Input

Multi-Layer-Perzeptron

- Die schichtweise Berechnung der Gradienten ist auch für mehr als zwei Schichten möglich.
- Der Fehler wird dann schichtenweise nach oben propagiert (Back-Propagation).

- Allgemein gilt für den Output-Fehler e eines Neurons j
  $e_j = \partial F / \partial x_j$

- Gewichtsänderung
  $\Delta w_{ik} = a \cdot e_k \cdot \theta'(a_k) \cdot x_i$

Backpropagation-Algorithmus

1. Wähle ein Muster x aus der Menge der Trainingsbeispiele D aus.

2. Präsentiere das Muster dem Netz und berechne Output (Propagation).

3. Der Fehler F wird als Differenz von errechnetem und gewünschten Output ermittelt.

4. Die Fehlerinformationen werden durch das Netz zurückpropagiert (Backpropagation).

5. Die Gewichte zwischen den einzelnen Schichten werden so angepasst (Δw_{ik}), dass der mittlere Ausgabefehler für das Muster sinkt.

6. Abbruch, wenn Fehler auf Validierungsmenge unter gegebenem Schwellwert liegt. (Siehe spätere Folien.) Sonst gehe zu 1.

Backpropagation Algorithmus

- Betrachtet man den Fehler des Netzes als Funktion aller Gewichte **w**, so kann man zeigen, daß bei jedem Schritt der Fehler kleiner wird. Dies erfolgt unabhängig vom gewählten Netz, also unabhängig von **w**.

- $e(\mathbf{w}) = (o - t)^2 = (t - \theta(\mathbf{w} \cdot x))^2$

- Es gilt bei jedem Schritt:

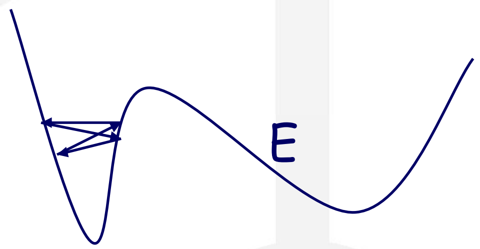
$$e(\mathbf{w} + \Delta\mathbf{w})^2 < e^2$$

Backpropagation-Algorithmus

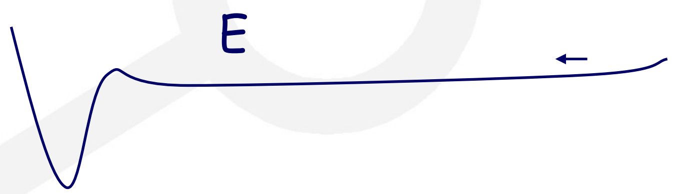
- Die Gewichtsänderungen können auf zwei Arten erfolgen:
    - Online Training:jedes Gewicht wird sofort angepasst (folgt nur im Mittel dem Gradienten)
    - Batch-Verfahren: es werden alle Datensätze präsentiert, die Gewichtsänderung des Gewichtes berechnet, summiert und dann erst angepasst (entspricht dem Gradienten über dem Datensatz)

Probleme des Backpropagation-Algorithmus

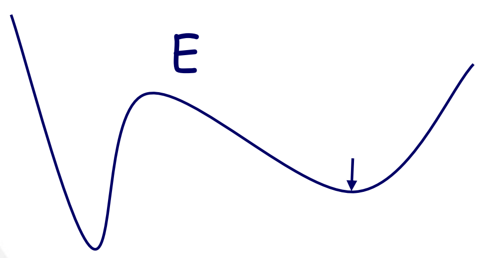
- Oszillation in engen Schluchten



- Stagnation auf flachen Plateaus $(\theta'(aj) \approx 0)$
  → kaum Veränderung im Training



- lokale Minima



Varianten des Backpropagation-Algorithmus

- Änderung der Fehlerfunktion
- Weight Decay
- Lernkonstanten und Output-Funktion können für jede Schicht oder für jedes Neuron einzeln festgelegt werden
- Lernkonstanten können dynamisch angepasst werden
- Gewichtsanpassung modifizieren
- Flat Spots eleminieren
- Lernrate η wird an die Gewichte und den Gradienten gekoppelt
- Konvergenz durch Nutzung der zweiten Ableitung beschleunigen

Bemerkungen

- Jede Boolesche Funktion kann durch derartiges Netz repräsentiert werden.
- Beliebige Funktionen können durch ein ANN mit drei Schichten beliebig genau approximiert werden.
- Hypothesenraum ist kontinuierlich im Gegensatz z.B. zum diskreten Hypothesenraum von Entscheidungsbaumverfahren.
- ANN kann für interne Schicht sinnvolle Repräsentationen lernen, die im vorhinein nicht bekannt sind;
  dies ist Gegensatz zu z.B. ILP (ein Verfahren mit vorgegebenem Hintergrundwissen).

ENDOWED CHAIR OF THE HERTIE FOUNDATION
Knowledge and Data Engineering
DEPARTMENT OF MATHEMATICS & COMPUTER SCIENCE

Abbruchbedingungen und Überspezialisierung

– Beschreibung des Backpropagation-Algorithmus lässt Abbruchbedingung offen.

- Eine schlechte Strategie ist es, den Algorithmus solange zu iterieren, bis der Fehler für die Trainingsbeispiele unter einem vorgegebenen Schwellwert liegt, da der Algorithmus zur Überspezialisierung (overfitting) neigt.
  (Vgl. Entscheidungsbäume in KDD-Vorlesung)

- Besser: verwende separate Menge von Beispielen zur Validierung (validation set); iteriere solange, bis Fehler für Validierungsmenge minimal ist.

- Aber: Der Fehler auf der Validierungsmenge muss nicht monoton fallen (im Gegensatz zu Fehler auf Trainingsmenge, siehe nächste Folie)!

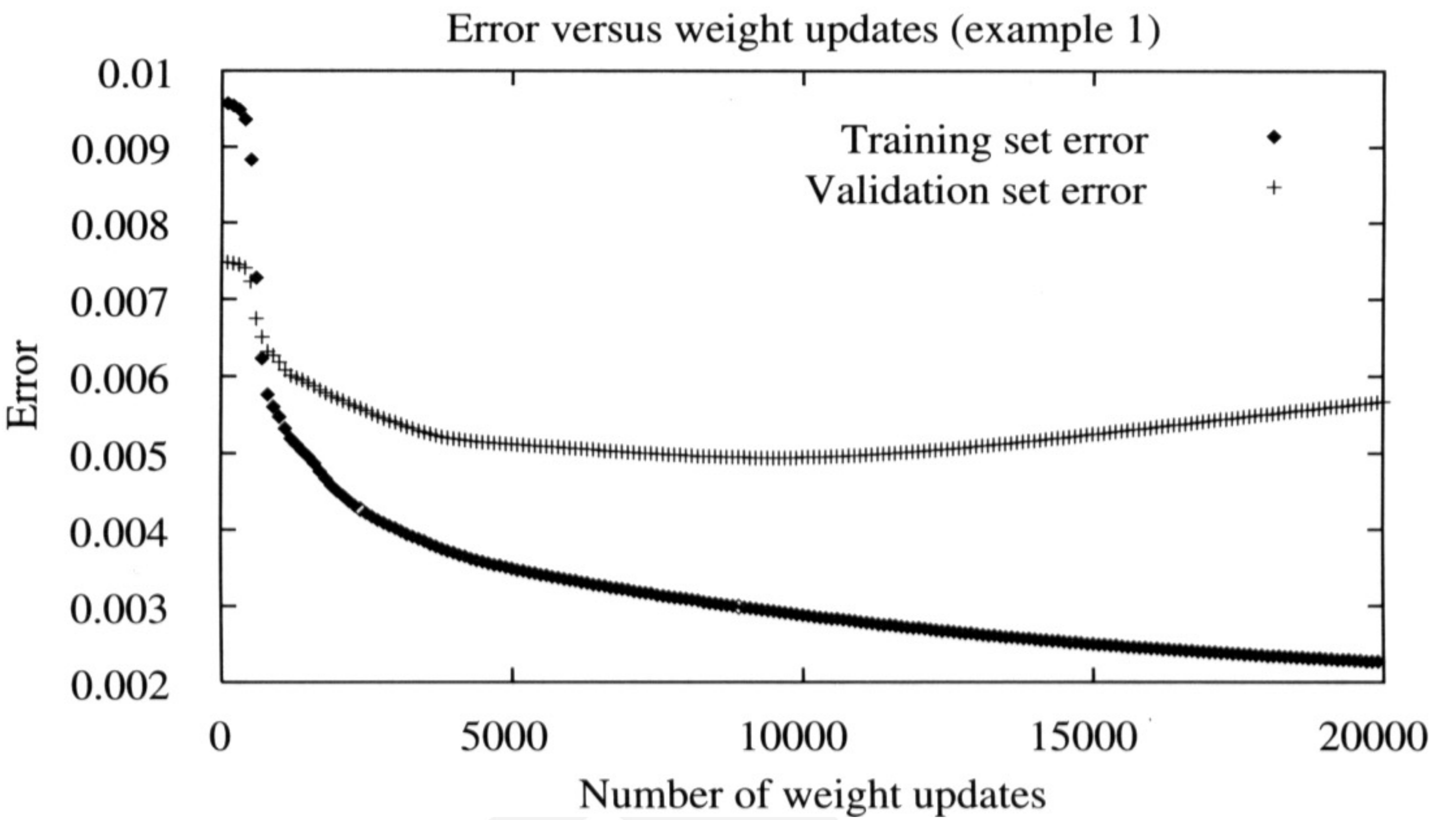


Figure 8a: Plots of error E as a function of the number of weight updates, for two different robot perception tasks (Mitchell 1997)

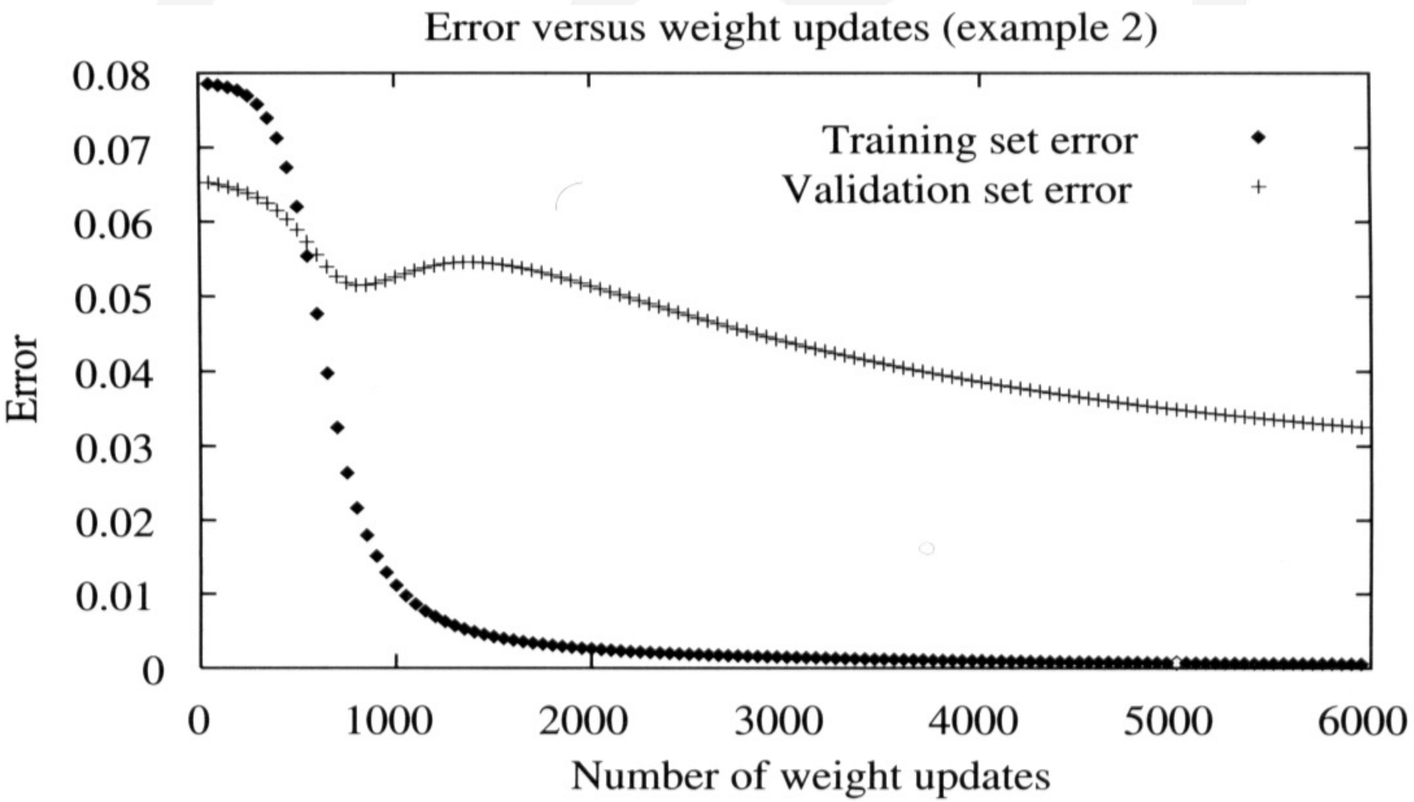


Figure 8b: Plots of error E as a function of the number of weight updates, for two different robot perception tasks (Mitchell 1997)

– Alternativ (insbes. bei wenigen Trainingsdaten) kann k-fache Kreuzvalidierung (k-fold cross-validation) verwendet werden:

  - Unterteile Menge der Trainingsbeispiele in k gleich große disjunkte Teilmengen.
  - Verwende der Reihe nach jeweils eine andere Teilmenge als Validierungsmenge und die restlichen (k - 1) Teilmengen als Trainingsmenge.
  - Für jede Validierungsmenge wird „optimale" Anzahl i von Iterationen bestimmt (d.h. mit kleinstem mittleren Fehler für die Daten aus der Validierungsmenge).
  - Der Mittelwert von i über alle k Trainingsphasen wird letztendlich verwendet, um das gegebene ANN mit allen Trainingsbeispielen zu trainieren.

Zusammenfassung Backpropagation-Algorithmus

- Backpropagation stellt trotz der beschriebenen Probleme einen bedeutenden Fortschritt dar.
- Aufgaben, wie das „exklusive oder", die mit einem Perzeptron nicht gelöst werden können, sind nun lösbar.
- Die Lösung wird durch die innere Schicht ermöglicht, die eine Umkodierung der Eingabevektoren vornimmt.