Kapitel VI

Neuronale Netze

(basierend auf Material von Andreas Hotho)

Agenda

1. Einführung & Grundbegriffe
   - Motivation & Definition
   - Vorbild Biologie
   - Historie der NN
   - Überblick über verschiedene Netzwerktypen
2. Einfaches Perzeptron
3. Multi Layer Perzeptron
4. Beispiele
5. Rekurrente Netze
6. Entwurfsüberlegungen

Was sind künstliche Neuronale Netze

Künstliche Neuronale Netze sind

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

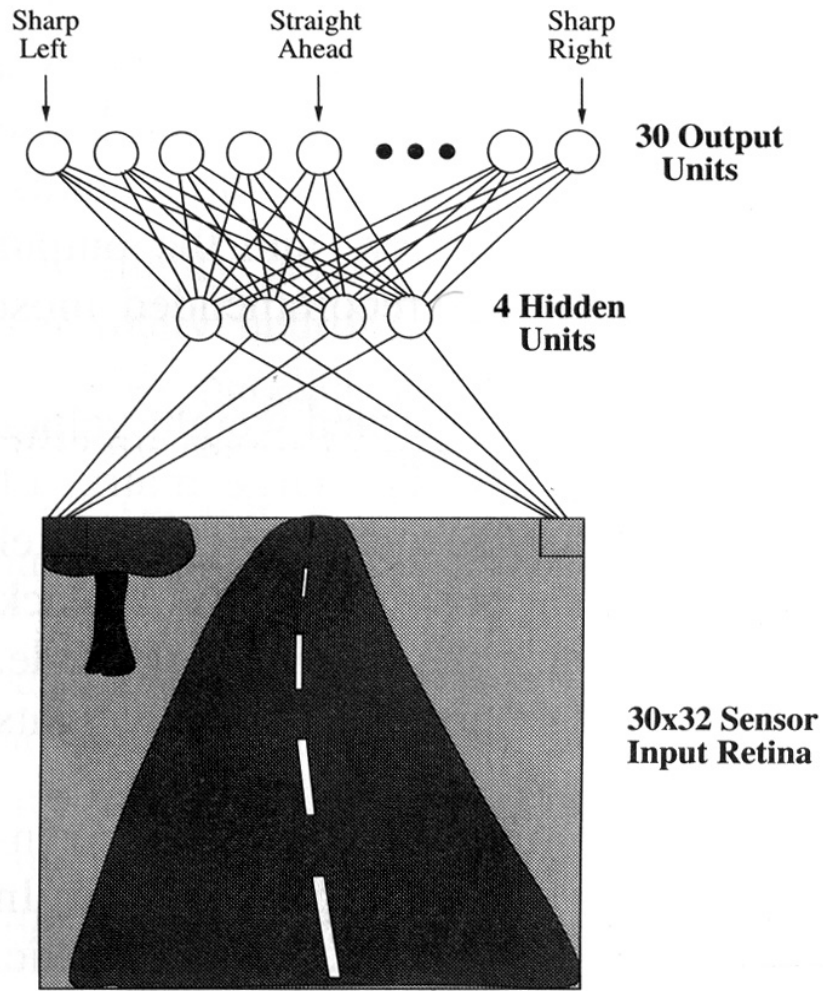
die mit der Welt in der selben Art wie biologische Nervensysteme interagieren sollen.

(Kohonen 84)

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



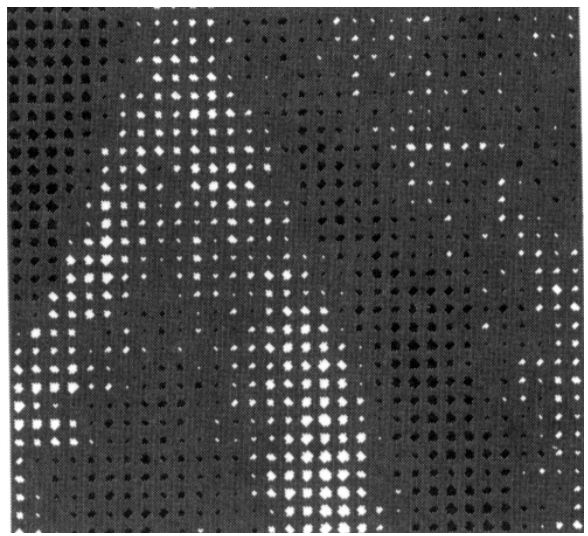


Figure 1: Neural Network learning to
steer an autonomous vehicle (Mitchell 1997)

– 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

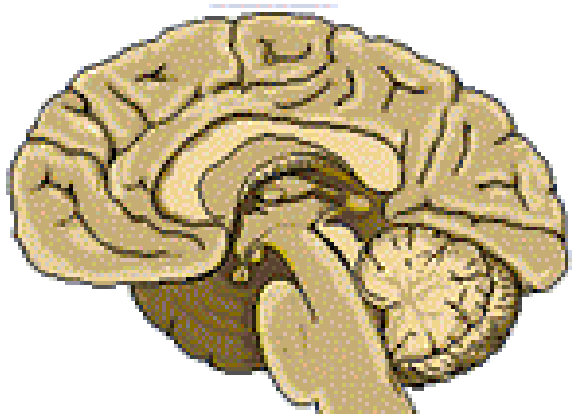
es werden die:

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

(Phasen können auch überlappen)

Vorbild Biologie

Gehirn



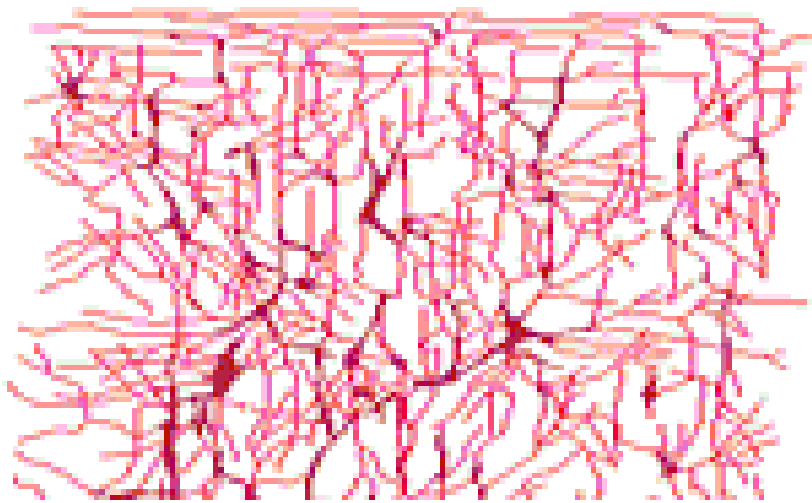
In der Vergangenheit wurden Forscher aus den verschiedensten Fachgebieten durch das biologische Vorbild motiviert.

Das Gehirn besteht aus

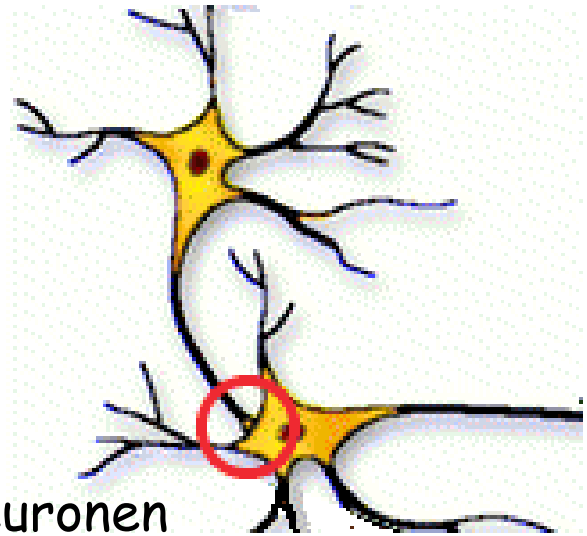
- ca. 10^{11} Neuronen, die mit
- ca. 10^{4} anderen Neuronen
- ca. 10^{13} Synapsen

verschaltet sind.

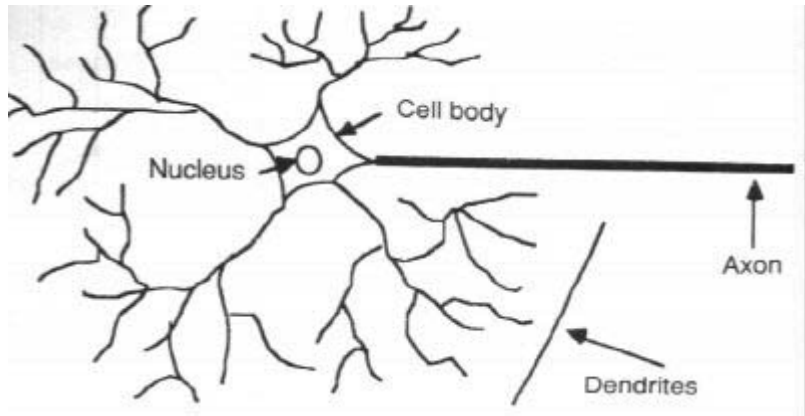
Geflecht aus Neuronen

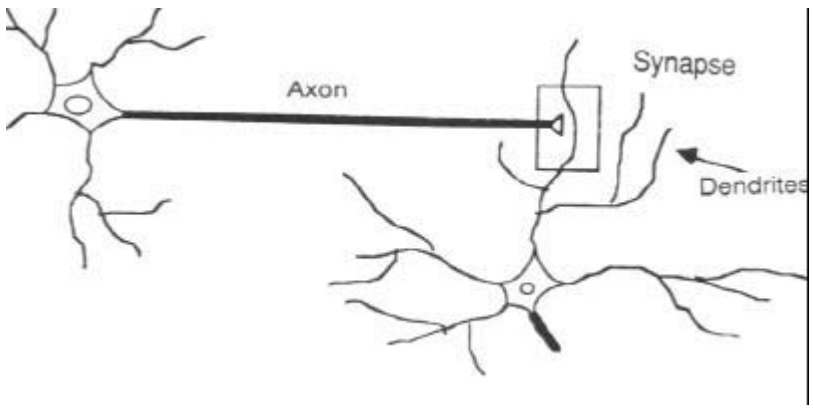


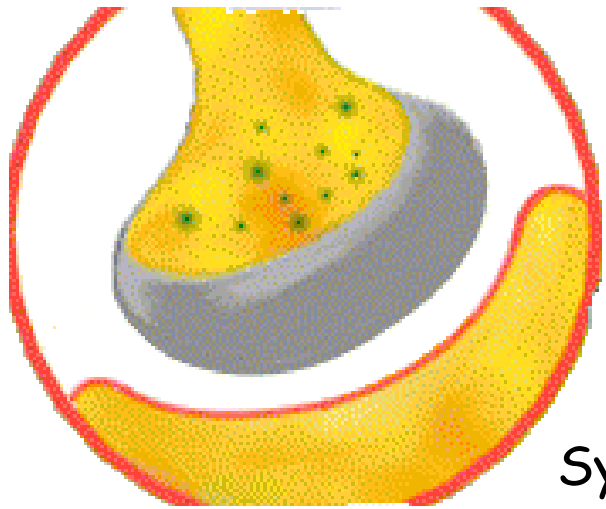
Vorbild Biologie



Neuronen



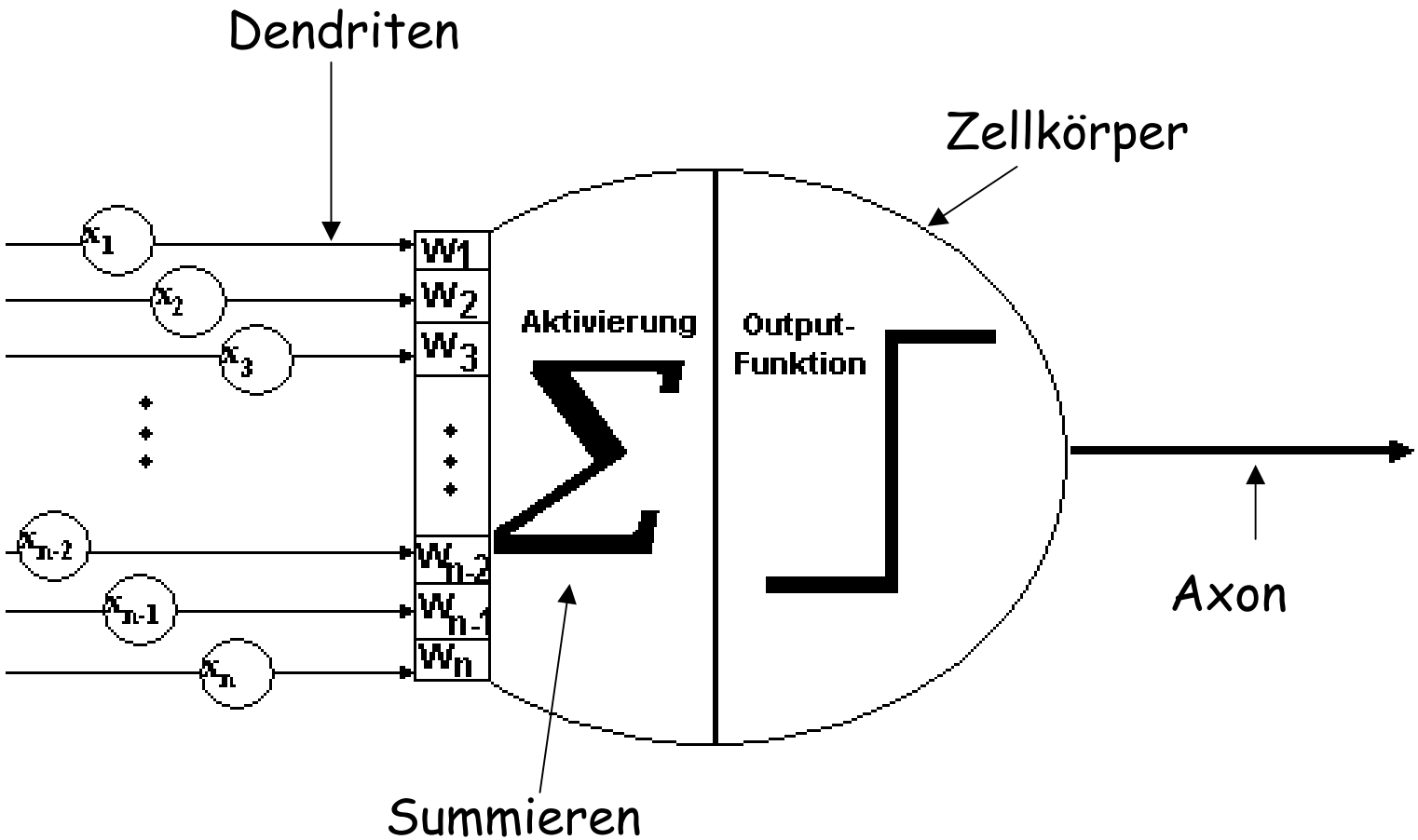




Synapse

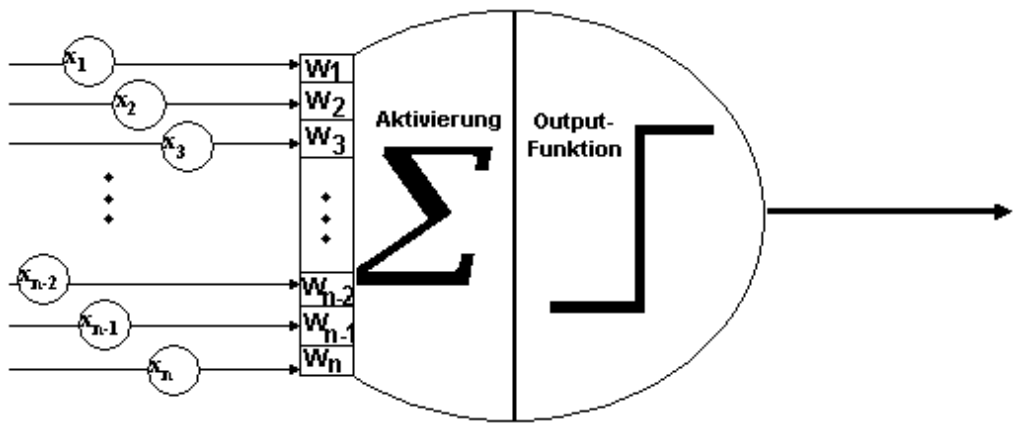
Vorbild Biologie

Vom natürlichen zum künstlichen Neuronalen Netz



Elemente eines Neuronalen Netzes

- Menge von Neuronen (Berechnungseinheiten, oder elementare Prozessoren)
- Verbindungen zwischen den Neuronen im Netz (Synapsen)
- Gewichte der Verbindungen
- Propagierungsregel für das Weiterleiten des Inputs
- Aktivierungs-Zustände der Neuronen
- Output-Funktionen und Schwellwerte der Neuronen
- Aktivierungsregel (Verknüpfung des Inputs und des Zustands zum Output)
- Lernregel
- Umgebung, in der das lernende System arbeiten soll



Geschichte der künstlichen Neuronalen Netze

Biologische Vorbilder wurden in den letzten 150 Jahren erforscht

- 1850 Nervenimpulse als elektrisches Phänomen
- 1901 Richtung der Informationsübertragung
- 1910-1930 synaptische Informationsübertragung
- 1949-1956 Informationsweiterleitung (ionische Depolarisation) geklärt
- ca. 1979 Abbildung der sensorischen Informationen als Karte im Gehirn (Verteilung über die Großhirnrinde)

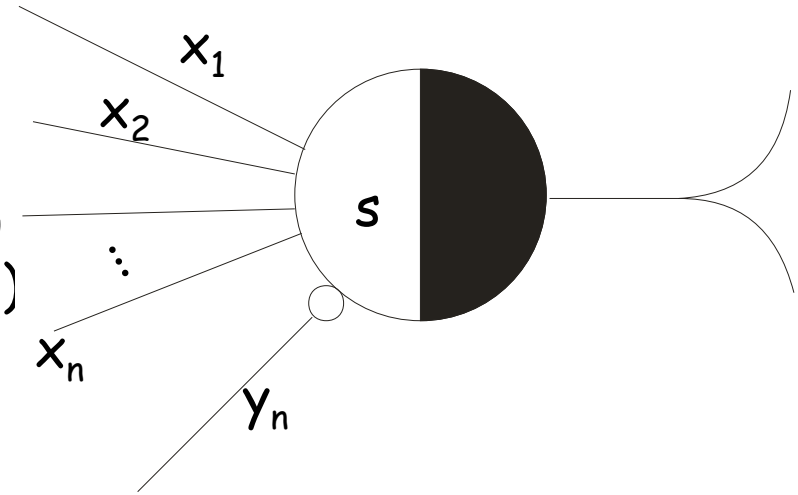
Neben den Biologen haben auch Psychologen und Ingenieure viel zur Weiterentwicklung des Verständnisses über Neuronale Netze beigetragen.

Geschichte der künstlichen Neuronalen Netze

1943	McCulloch und Pitts entwickelten erste künstliche Neuronale Netze auf ihrem Verständnis der Neurologie
1949	Hebb'sche Lernregel
1957	entwickelt Rosenblatt das Perceptron
1969	Minski und Papert führten eine rigorose Analyse des Perceptrons durch
1972	Kohonen Assoziativspeicher
1974	Werbos entwickelt Backpropagation Algorithm
1976	Grossberg entwickelt ART1 Modell
1982	Kohonen's selbstorganisierende Karten
1985	Hopfield, Tank realisierten neue Ideen wie rückgekoppelte Netzwerke mit selbstorganisierendem Lernen
1986	Rumelhart und Hinton führten die alten Ideen des Perceptrons in mehrschichtigen Modellen weiter (Backpropagation Algorithmus) MLP

Geschichte der künstlichen Neuronalen Netze

McCulloch und Pitts (1943) Netze



- binäre Ein- und Ausgabe
- n erregende Leitungen (x_1, ..., x_n)
- m hemmende Leitungen (y_1, ..., y_m)
- Schwellenwert s
- Ausgabefunktion
- darf zyklisch sein
- Ausgabefunktion ist eine Treppenfunktion
- Berechnung benötigt eine Zeiteinheit

Falls $m \geq 1$ und eines der Signale y_1, ..., y_m gleich eins ist
dann: Output = 0
sonst: Summe (x_1, ..., x_m) wird berechnet
und mit Schwellenwert s verglichen
Falls Erregung $\geq$ s ist
dann: Output = 1
sonst: Output = 0

Geschichte der künstlichen Neuronalen Netze

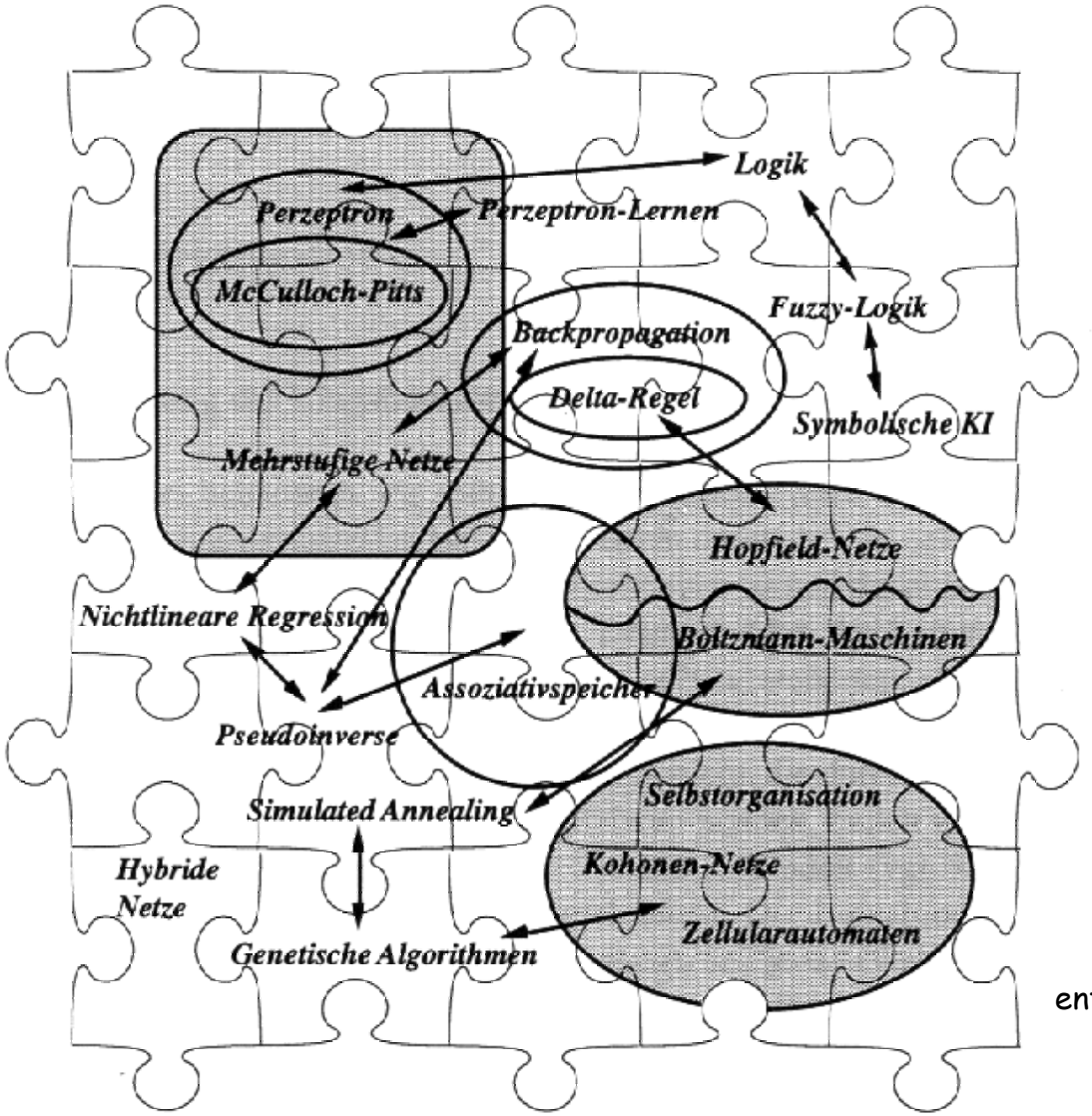
Vorteile

- erlaubt die Konstruktion von beliebigen logischen Funktionen
- jeder endliche Automat kann mit einem Netz aus McCulloch-Pitts Zellen simuliert werden

Nachteil

- Lernen sehr aufwendig (Topologieänderung nötig)

Typologie künstlicher neuronaler Netzwerk

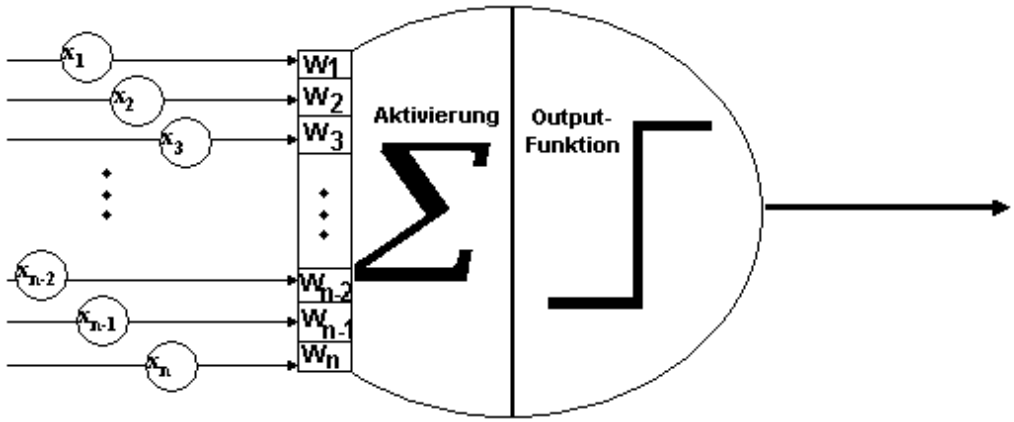


entnommen Rojas, S. 337

Typologie künstlicher neuronaler Netzwerk

gewichtete	<--->	ungewichtete Netze
synchrone	<--->	asynchrone Netze
mit	<--->	ohne gespeichertem Zustand
mit	<--->	ohne Rückkopplung
in Schichten	<--->	ohne Schichten
binäre	<--->	reelle Zustände

Aktivierungsfunktionen



Es gibt i.w. vier Arten der Aktivierung von Neuronen:

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:
   $a = a_1+a_2+\ldots+a_n$

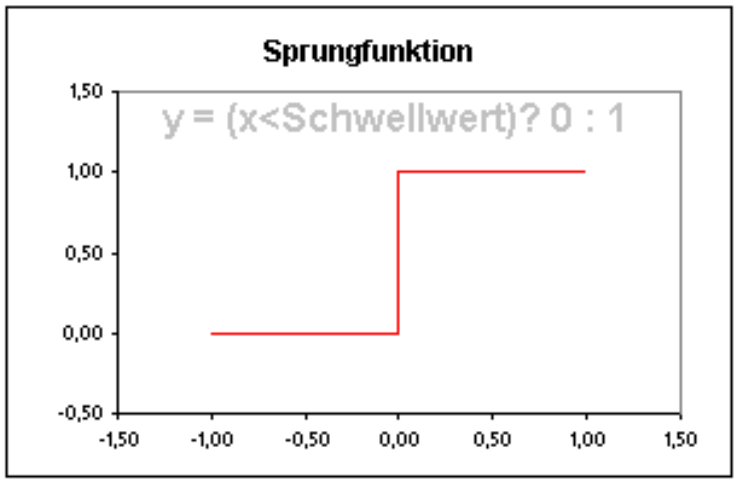
Aktivierungsfunktionen

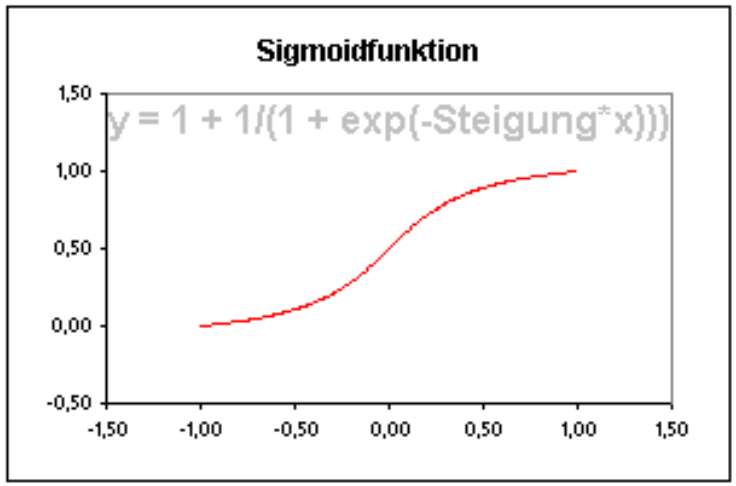
2. Sehr häufig ist ebenfalls die **Winner-take-all-Regel:** bei der die Aktivität a zunächst nach der Skalarproduktregel ermittelt wird, dann aber mit allen Aktivitäten in derselben Schicht verglichen wird und auf 0 herabgesetzt wird, wenn ein anderes Neuron höhere Aktivität hat.

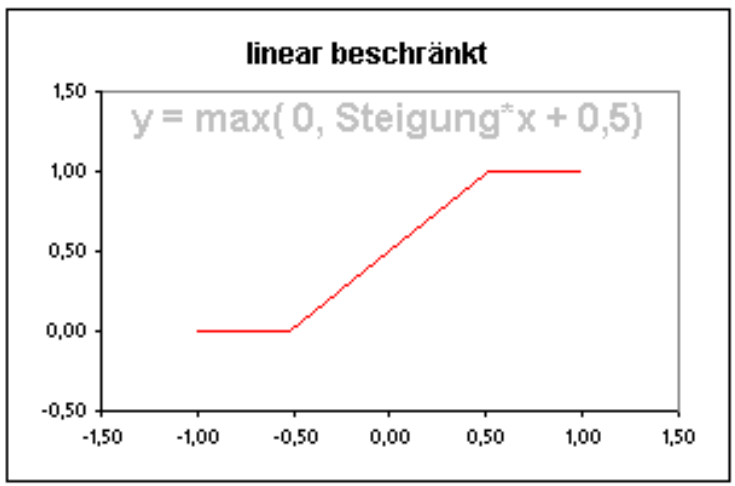
3. Bisweilen kommt auch die **sigma-pi-Regel** vor: Die Inputs werden in k Gruppen aufgeteilt und die modifizierten Inputs jeder Gruppe miteinander multipliziert, danach werden die Produkte $p_1, p_2, \ldots, p_k$ zur Aktivität des Neurons aufaddiert $a = p_1 + p_2 + \ldots + p_k$. Der Effekt ist, daß ein Input von fast 0 die gesamte Gruppe blockiert.

4. Selten ist die Regel, bei der ebenfalls die Inputs in Gruppen eingeteilt werden (z.B. 2 Gruppen: inhibitorische und excitatorische) und gruppenweise nach Skalarproduktregel zusammengefasst werden. Die Ergebnisse werden dann mit einer nichtlinearen Funktion miteinander verknüpft (z.B. $a=(e-i)/|i|$)

Outputfunktionen







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

--> Demo (funktion.html)

Lernparadigmen

Lernverfahren lassen sich grob unterteilen in:

- supervised (überwachtes)
  Der Trainingsdatensatz enthält nicht nur den Input für das Neuronale Netz sondern auch den gewünschten Output. Anhand dieses Outputs kann der Fehler des Netzes bestimmt und korrigiert werden.

- unsupervised (unüberwachtes)
  Neuronales Netz muß anhand des präsentierten Inputs die Muster in den Daten erkennen.

Overfitting

- Mit Overfitting beschreibt man das Problem der Überanpassung eines Modells an einen 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
    - Trainingset
    - Validierungsset
    - Testset

Fuzzy

- Neuronale Netze sind in der Lage, Regelaufgaben zu erlernen.
- Aber es ist keine Interpretation möglich
- kombiniert man Fuzzy und Neuronale Netze, so ist man in der Lage, die gelernten Regel aus dem Neuronalen Netz zu extrahieren
- Regel können dann auch interpretiert und von Experten geprüft werden

mehr dazu unter:
http://fuzzy.cs.uni-magdeburg.de/index.html

Genetische Algorithmen

- Genetische Algorithmen können für das Training von Neuronalen Netzen eingesetzt werden und sind in der Lage, das globale Optimum zu finden.
- eine ganze Population von Netzen wird konstruiert
- beste Netze erhalten dann auch die besten Chancen
- schlechte Netze sterben aus

mehr dazu unter:
http://www-ra.informatik.uni-tuebingen.de/forschung/welcome.html

Statistik

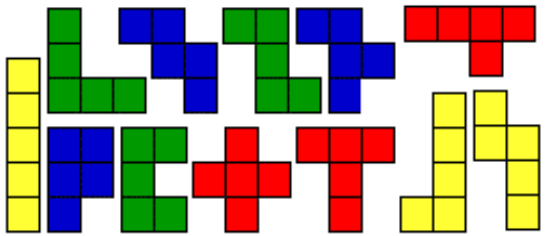
In welchem Zusammenhang stehen Neuronale Netze und statistische Analyse?

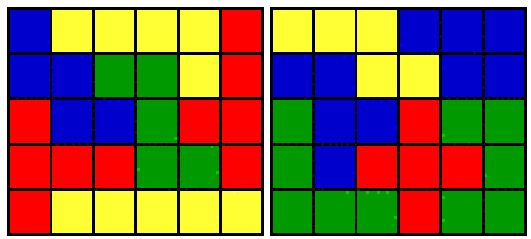
- Teilgebiete der Statistik beschäftigen sich mit Datenanalyse
- aus Sicht der Neuronalen Netz bedeutet dies:
  --> statistisches Schlußfolgern heißt für NN, Verallgemeinerungen aus verrauschten Daten zu lernen
- Beispiel:
  - vorwärtsgerichtete Netze ohne versteckte Schicht entsprechen linearen Modellen
  - Kohonen Netz ist dem K-Means Clusterverfahren ähnlich
  - Hebbian Lernen ist Hauptkomponentenanalyse ähnlich

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

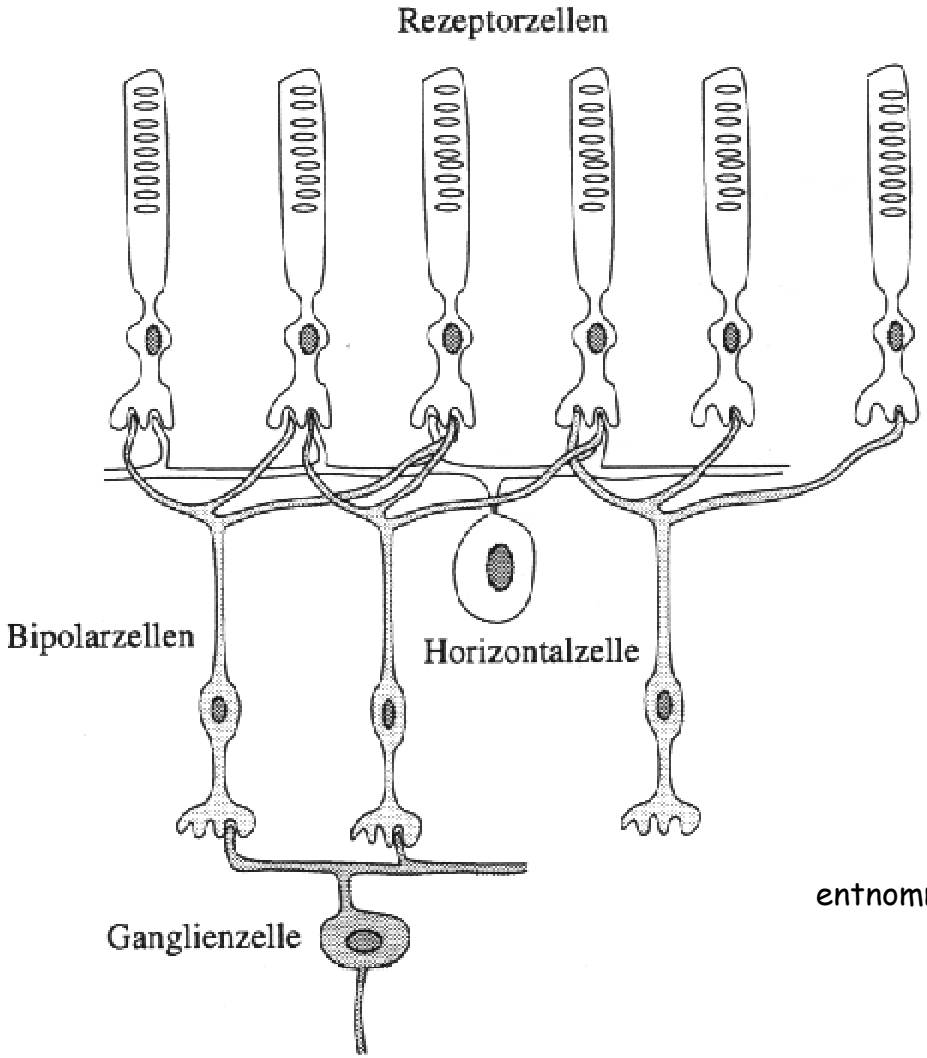
Nachteile

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

Agenda

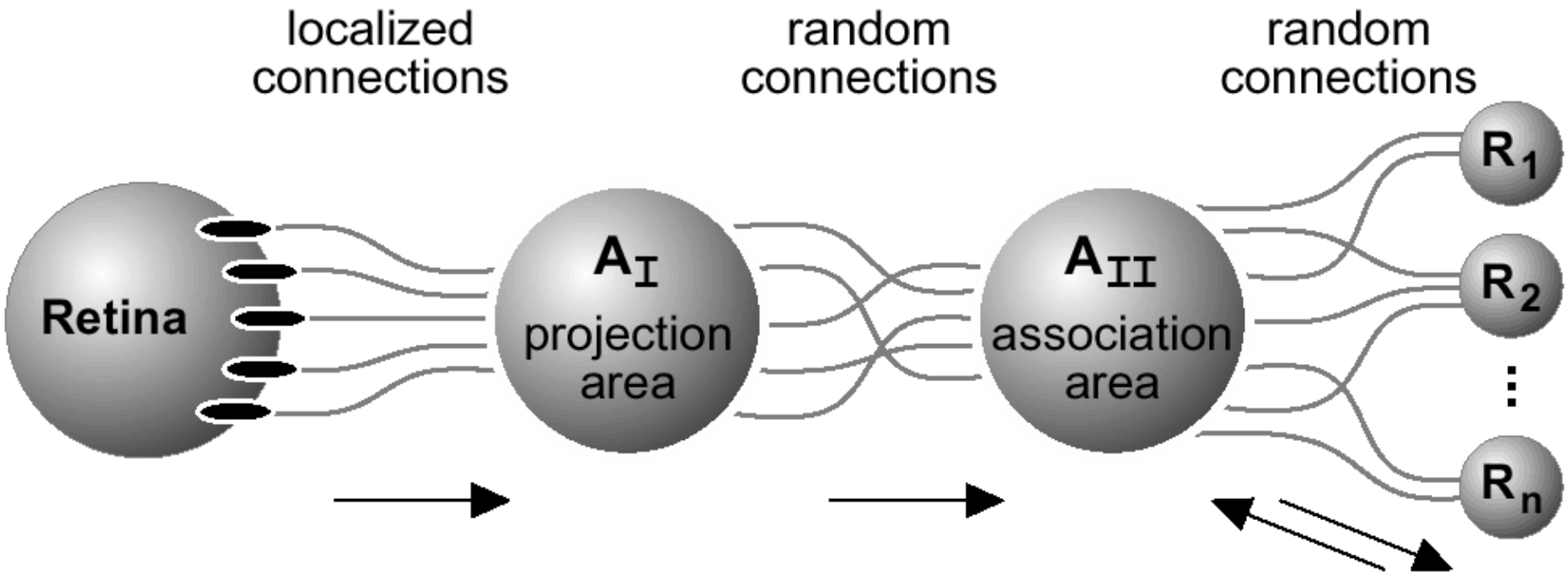
1. Einführung
2. Einfaches Perzeptron
   - Motivation
   - Definition Perzeptron
   - Geometrische Interpretation
   - Lernen - Delta Regel
   - XOR Problem
3. Multi Layer Perzeptron
4. Beispiel
5. Rekurrente Netze
6. Entwurfsüberlegungen

Verschaltung der Netzhaut

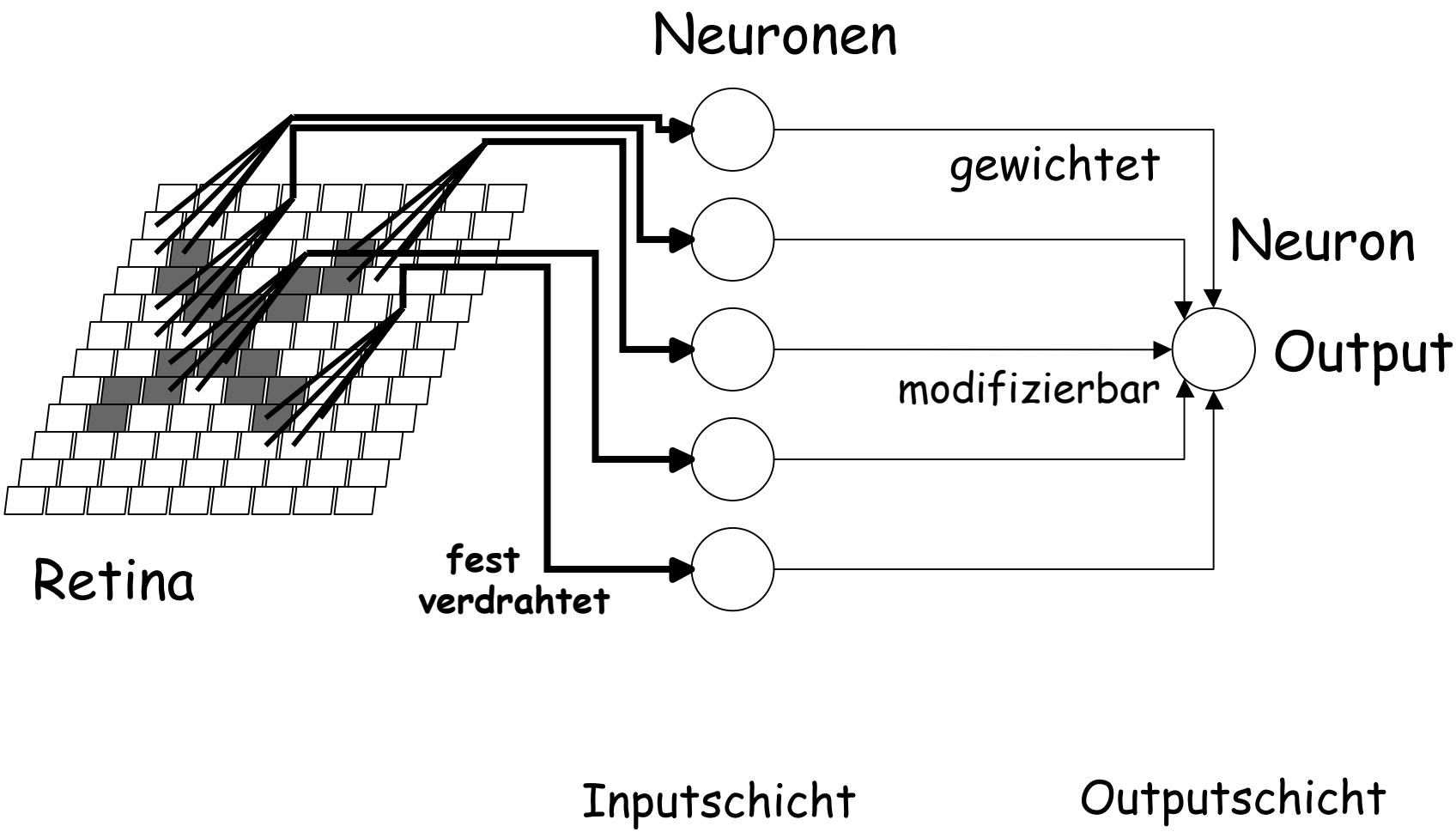


entnommen Rojas, S. 69

Perzeptron von Rosenblatt 1958



Perzeptron von Rosenblatt 1958

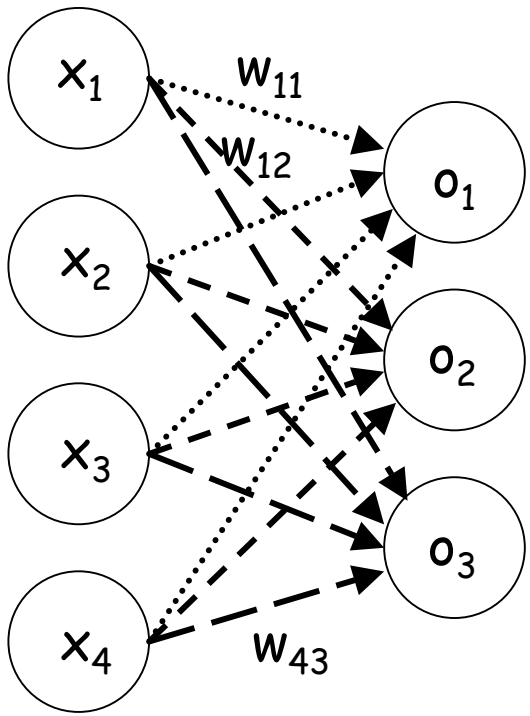


Perzeptron von Rosenblatt 1958

- feste Verbindung zwischen Retina und Inputschicht
- Vorverarbeitung erfolgt zwischen Retina und Inputschicht
- Ausgabe der Inputschicht ist 0 oder 1
- Art der Vorverarbeitung (Verbindung + Neuronen) wurde im Originalmodell nicht spezifiziert
- Lernvorgang konzentriert sich auf die gewichteten Verbindungen zwischen Input- und Outputschicht

Perzeptron

- jedes Outputneuron hat einen eigenen unabhängigen Netzbereich



Inputschicht Outputschicht

- 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

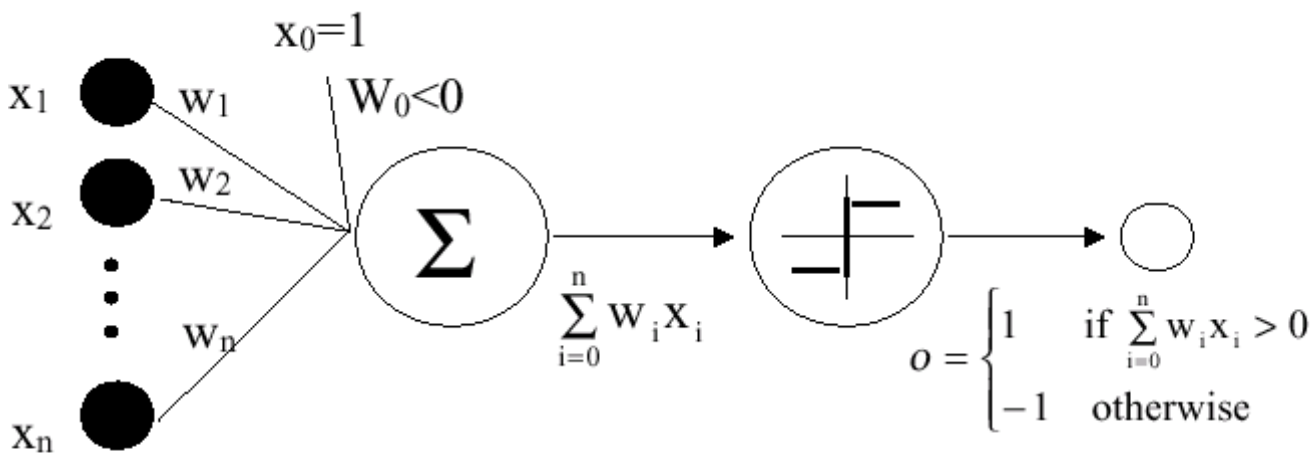


Figure 2: A Perceptron (Mitchell 1997)

- 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 $a < s$

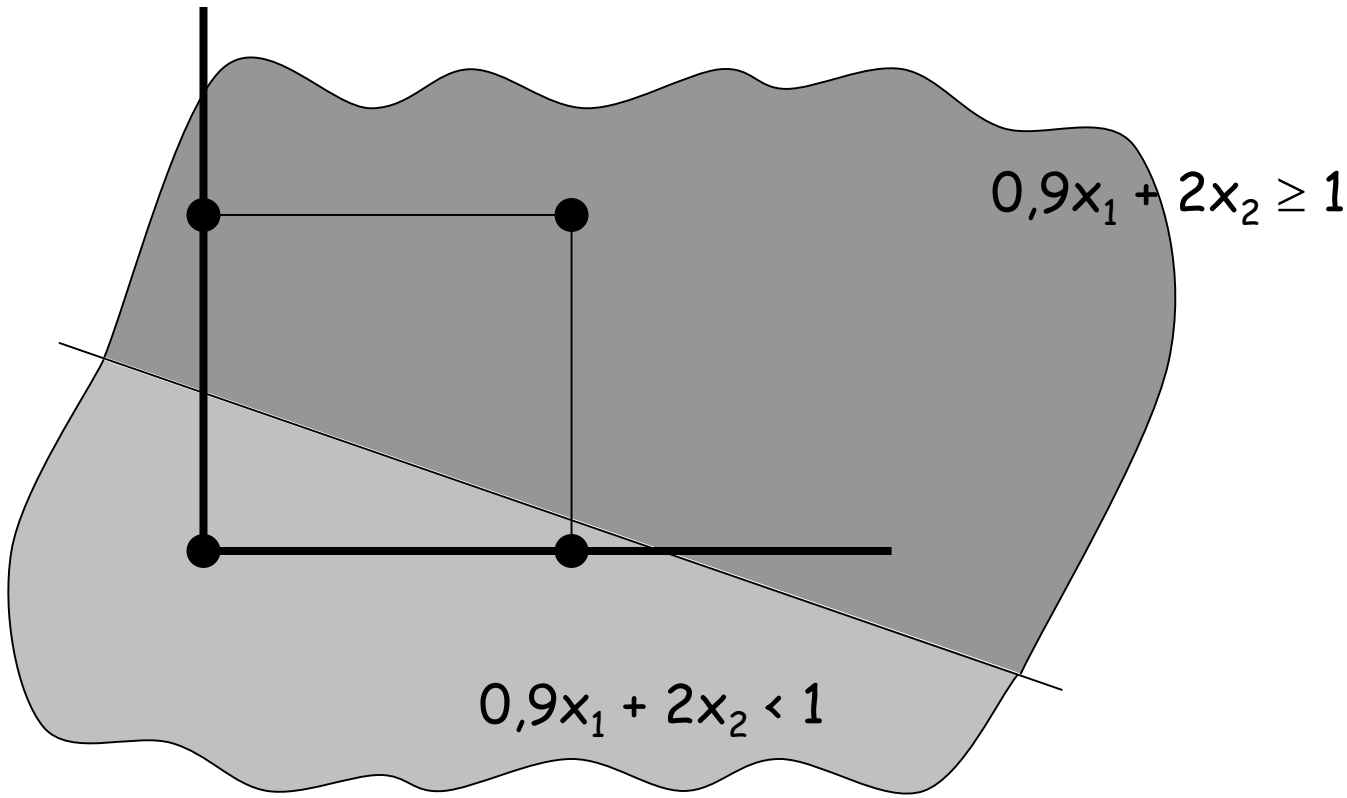
  $o = 1$, sonst.
- äquivalente Beschreibung: erweitert man die Vektoren $\boldsymbol{x}$ und $\boldsymbol{w}$ zu $\boldsymbol{y} = (x_1, \ldots, x_{n,1})$ und $\boldsymbol{v} = (w_1, \ldots, w_n, s)$, so ist

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

  $o = 1$, sonst

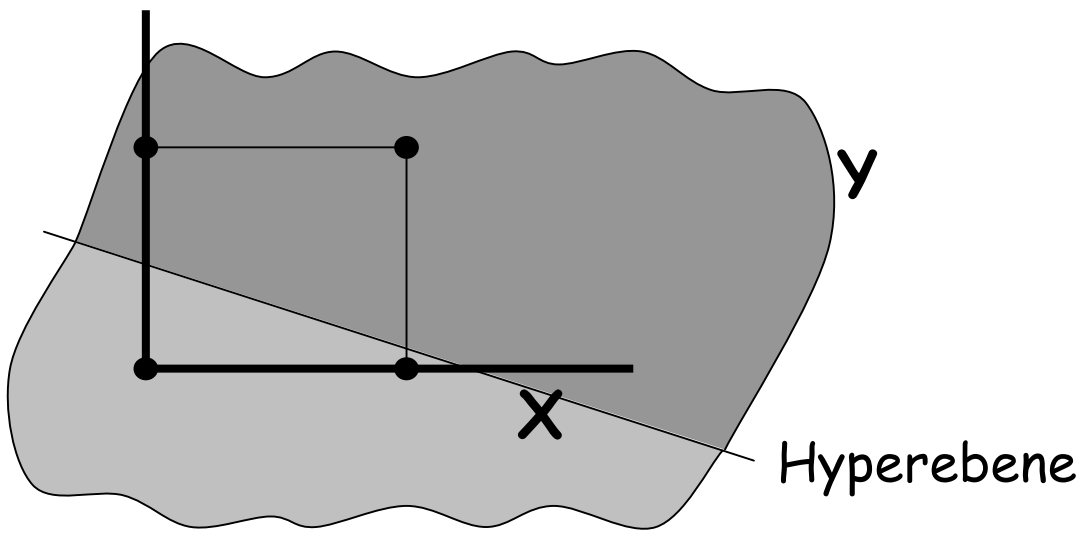
Geometrische Interpretation

- 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 $\boldsymbol{w}$ $(w_1, \ldots, w_n)$, so daß eine Hyperebene spezifiziert wird, die beide Mengen X und Y voneinander trennt
- Mengen müssen linear trennbar sein, damit die Aufgabe lösbar ist.



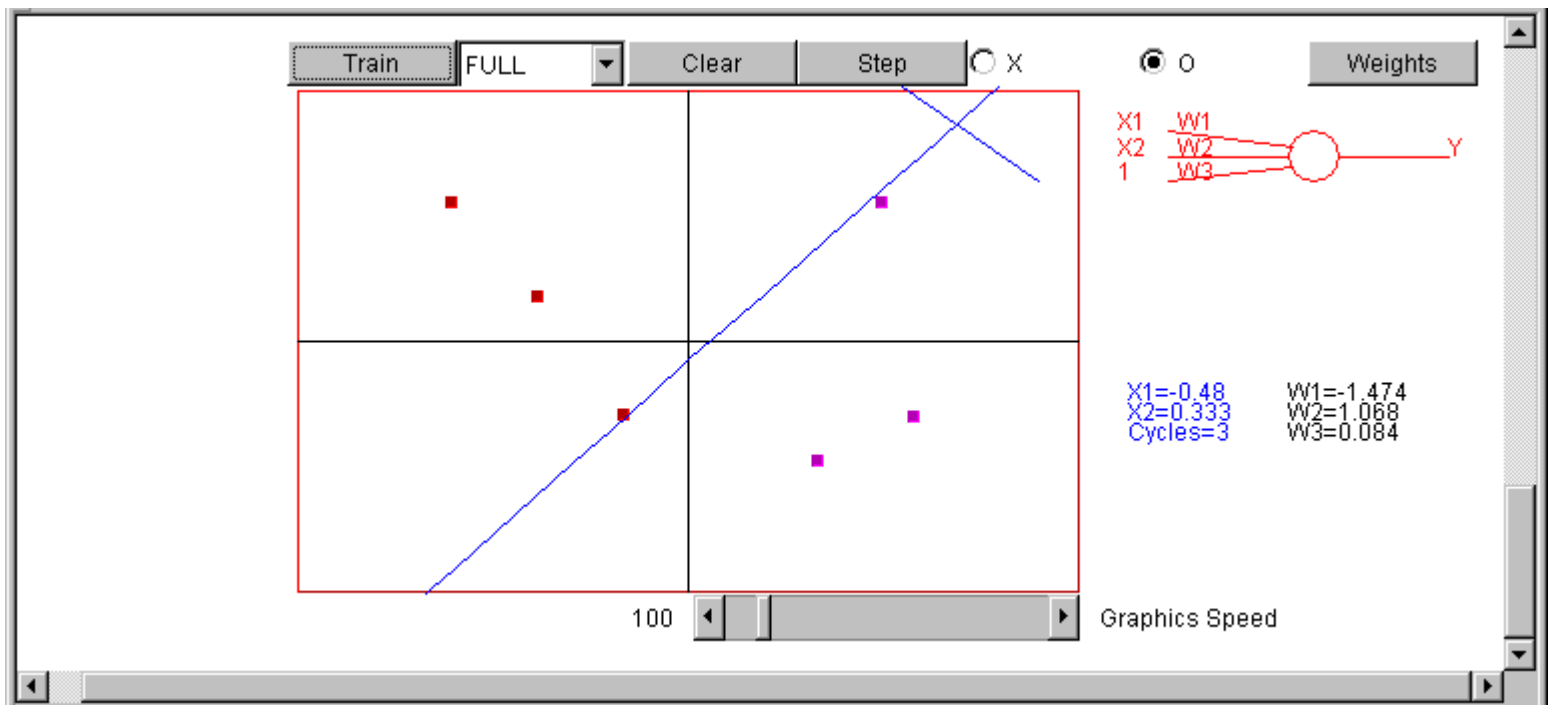
Delta-Regel

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 und bei Diskrepanz werden Schwellenwert und Gewichte nach folgender Delta-Regel angepasst:

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

η Lernrate

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

Delta-Regel

Algorithmus mit Lernrate $\eta = 1$, als Output nur 0 und 1 möglich!

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 (keine obere Schranke)

Konvergenz und Korrektheit der Delta-Regel

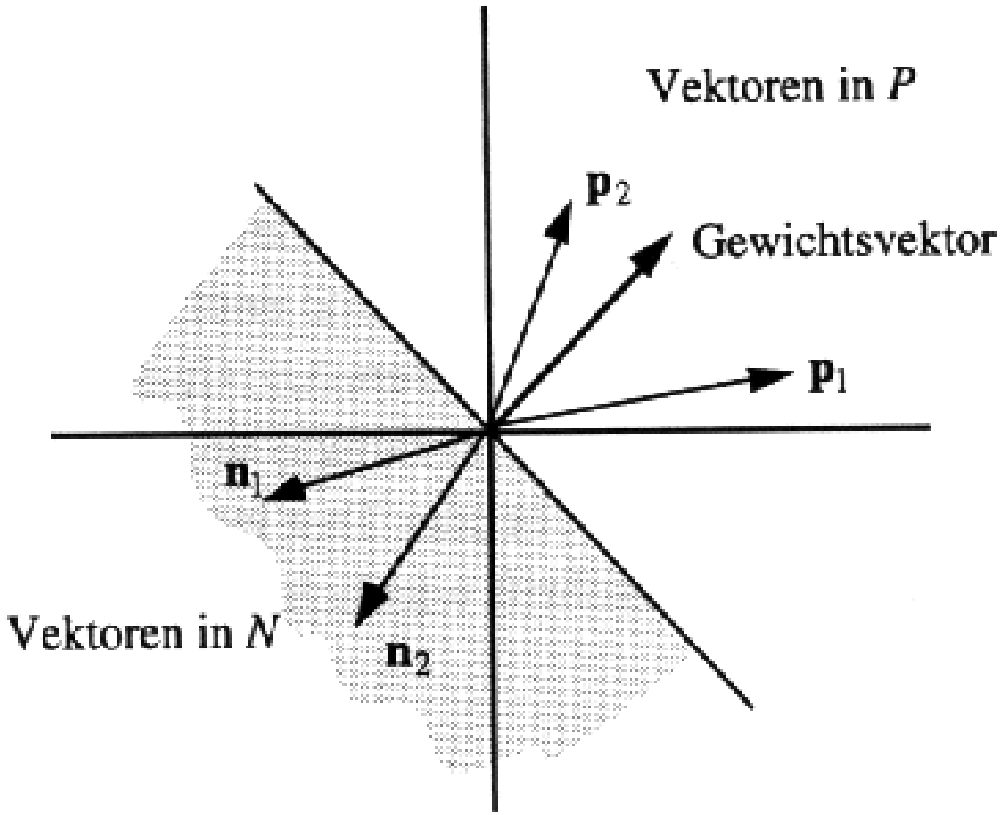
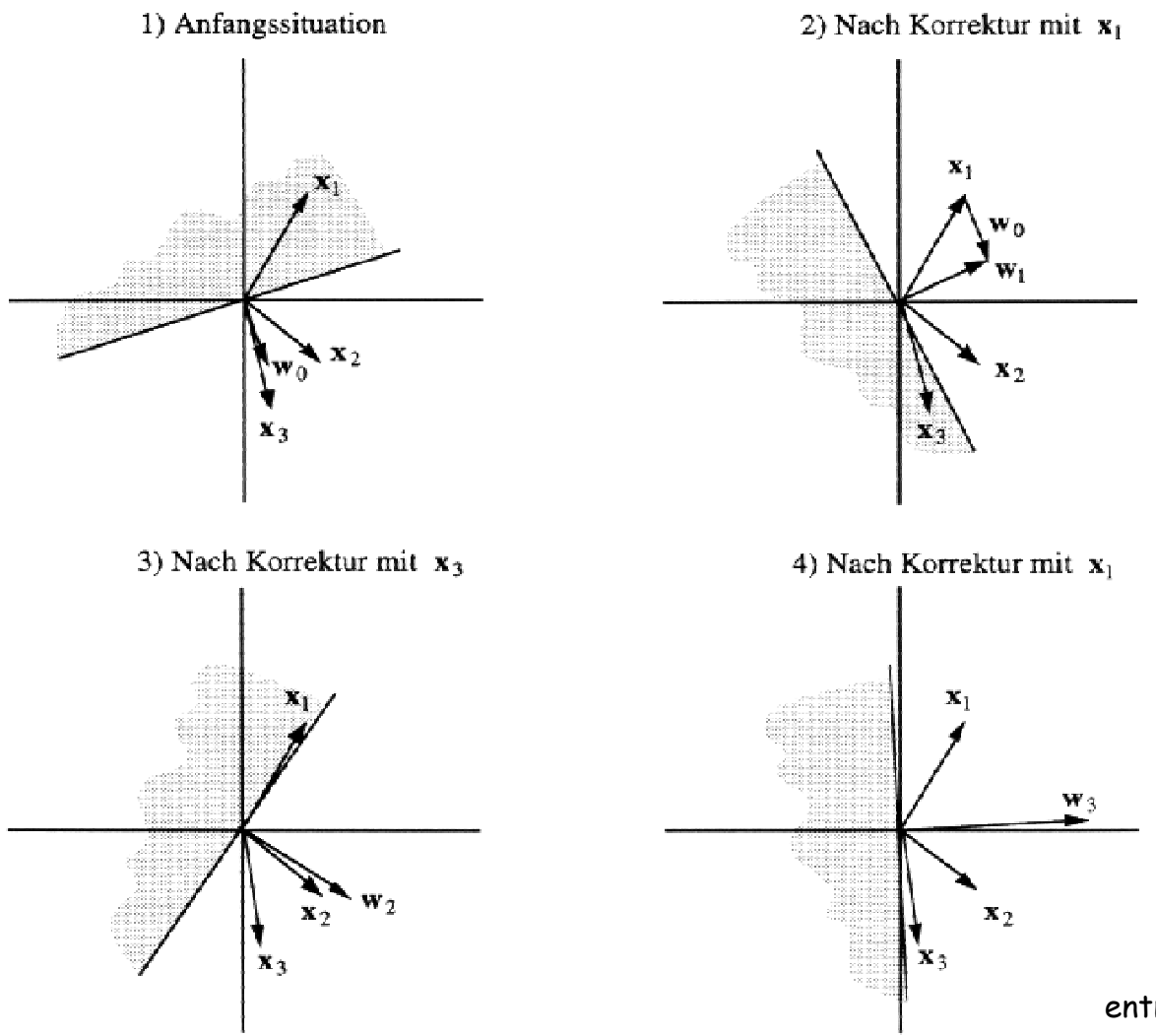


Abb. 4.8 Positiver und negativer Halbraum im Eingaberaum

entnommen Rojas, S. 84

Konvergenz und Korrektheit der Delta-Regel

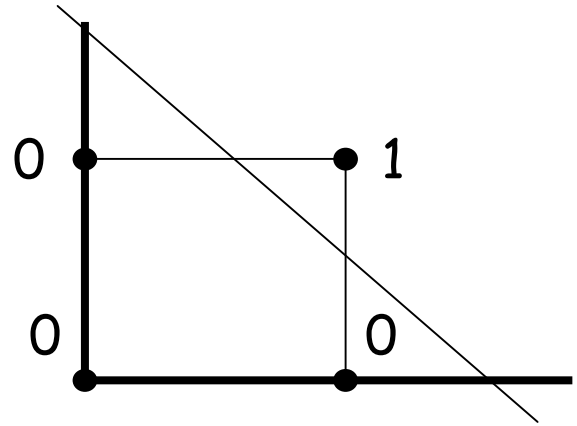


entnommen Rojas, S. 85

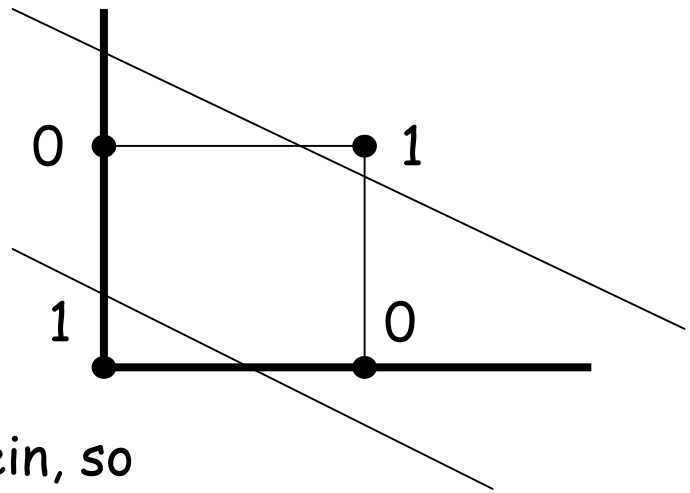
Abb. 4.9 Konvergenzverhalten des Lernalgorithmus

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

– Perzeptron kann z.B. verschiedene Boolsche Funktionen repräsentieren: AND, OR, NAND, NOR (aber: XOR ist nicht darstellbar)

- Realisierung von AND:
  - wähle Perzeptron mit zwei Eingabegrößen
  - wähle $w_0 = -0.8$, $w_1 = w_2 = 0.5$

- AND, OR sind Spezialfall sogenannter ‘‘m-aus-n Funktionen‘‘ (m-of-n functions): wenigstens m der n Eingabegrößen müssen 1 sein, hier:
  - AND: m=1 (wenigstens 1 Eingabegröße ist 1)
  - OR: m=n (alle Eingabegrößen sind 1)

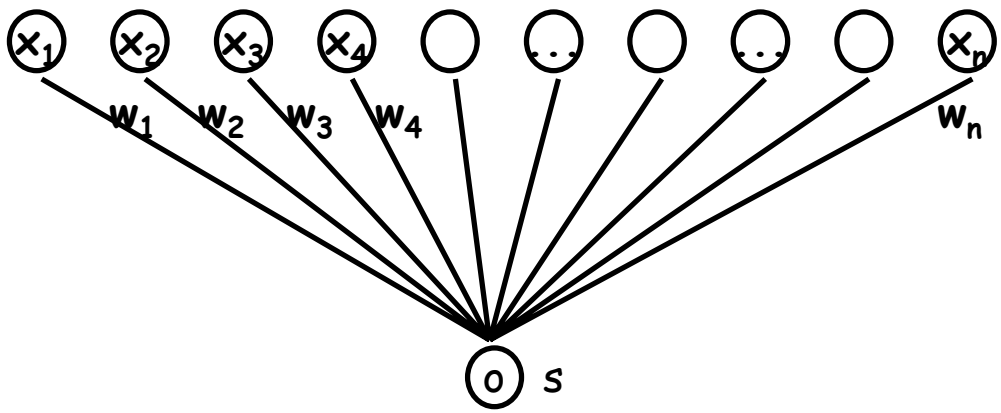
Agenda

1. Einführung
2. Einfaches Perzeptron
3. Multi Layer Perzeptron
   - Vektorschreibweise der Deltaregel
   - Schichten des MLP
   - Backpropagation
   - Probleme der Backpropagation
   - Varianten der Backpropagation
4. Beispiel
5. Rekurrente Netze
6. Entwurfsüberlegungen

Erinnerung Perzeptron

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

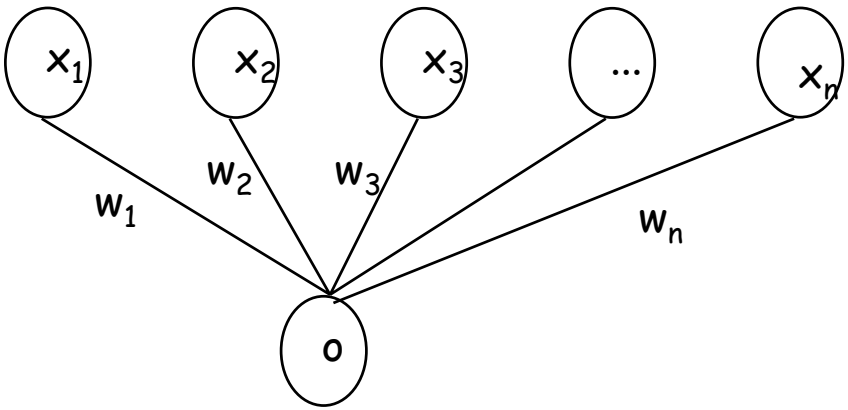
- Schwellwert: s
- Netz-Output: $o = \theta(x_1w_1+x_2w_2+x_3w_3+...+x_nw_n-s)$
- erwarteter Output: t (target)
- gemachter Fehler: $e = t - o$ (error)
- Lernkonstante : η



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 Ableitung für Perzeptron



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

Fehlergradient:

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

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

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

$= \underbrace{2\, \theta'(\underline{x} \cdot \underline{w})}_{\eta} (o-t)\, x_i$

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

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

(unter der Annahme: θ ist diff.-bar)

2 Layer Perzeptron

Input-Vektor $\underline{\mathbf{x}}$

Gewichtsmatrix $\underline{\mathbf{v}}$

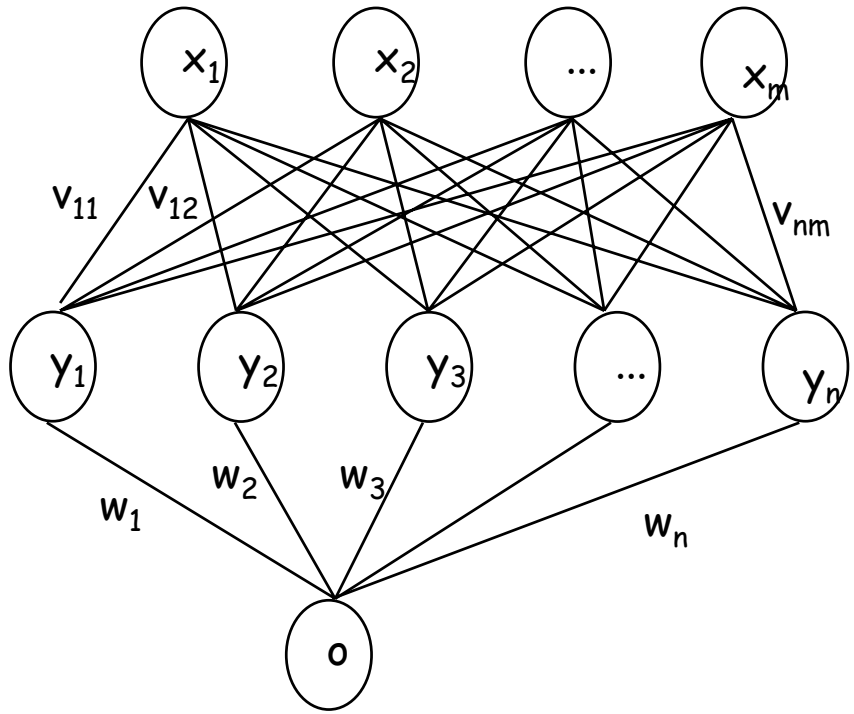
Aktivitätsvektor $\underline{\mathbf{y}}$

Gewichtsvektor $\underline{\mathbf{w}}$

Output o

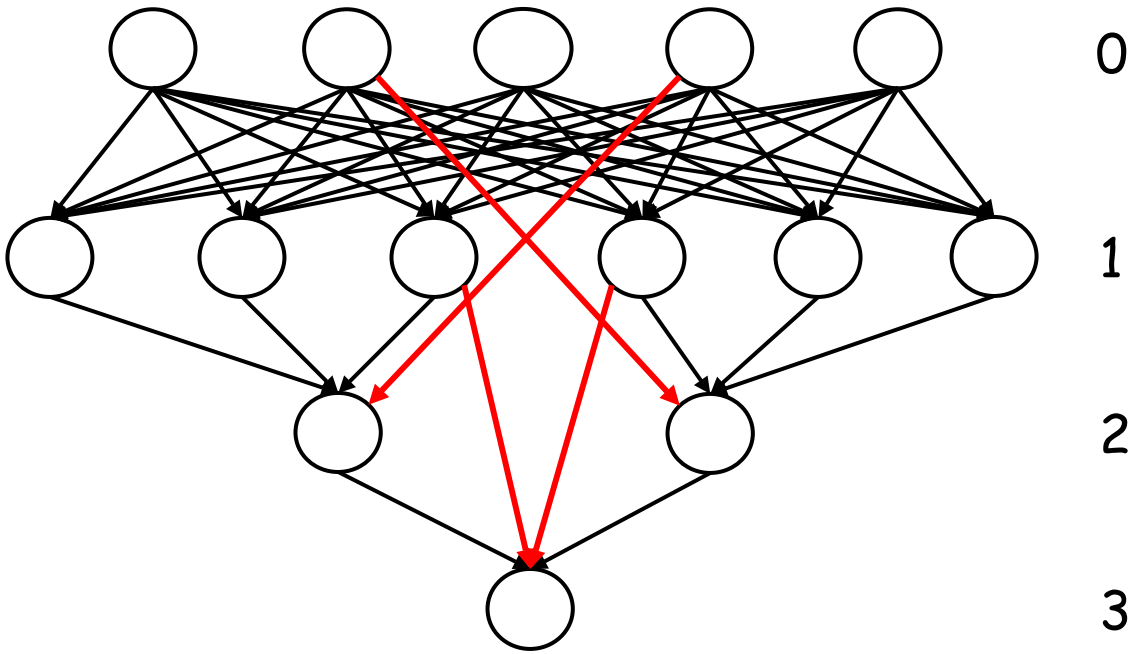
$$\underline{\mathbf{y}} = \theta(\underline{\mathbf{v}} \cdot \underline{\mathbf{x}})$$

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



Multi Layer Perzeptron

- Netze ohne Rückkopplungen können stets in die Form eines Schichtennetzwerkes gebracht werden, wobei aber durchaus Verbindungen eine oder mehrere Schichten überspringen dürfen



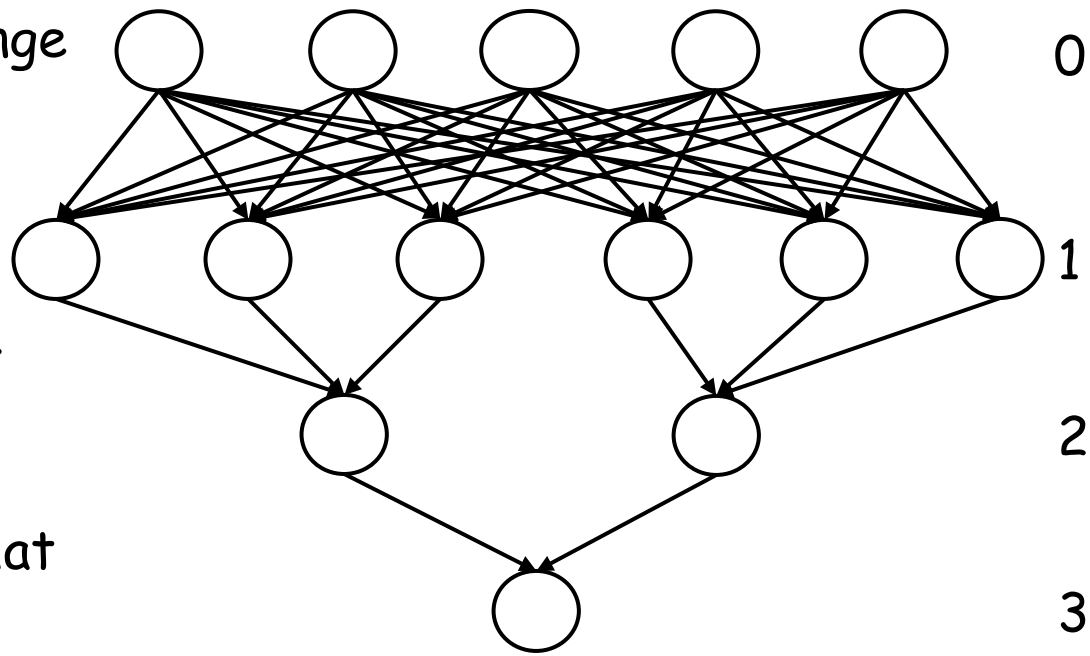
Multi Layer Perzeptron

- Wenn in einem Schichtennetz keine Verbindungen existieren, die eine oder mehrere Schichten überspringen, spricht man von einem **strengen** Schichtennetz.

- In diesem Fall ist jede Schicht durch die Länge des Weges zu ihren Neuronen gekenn-zeichnet.

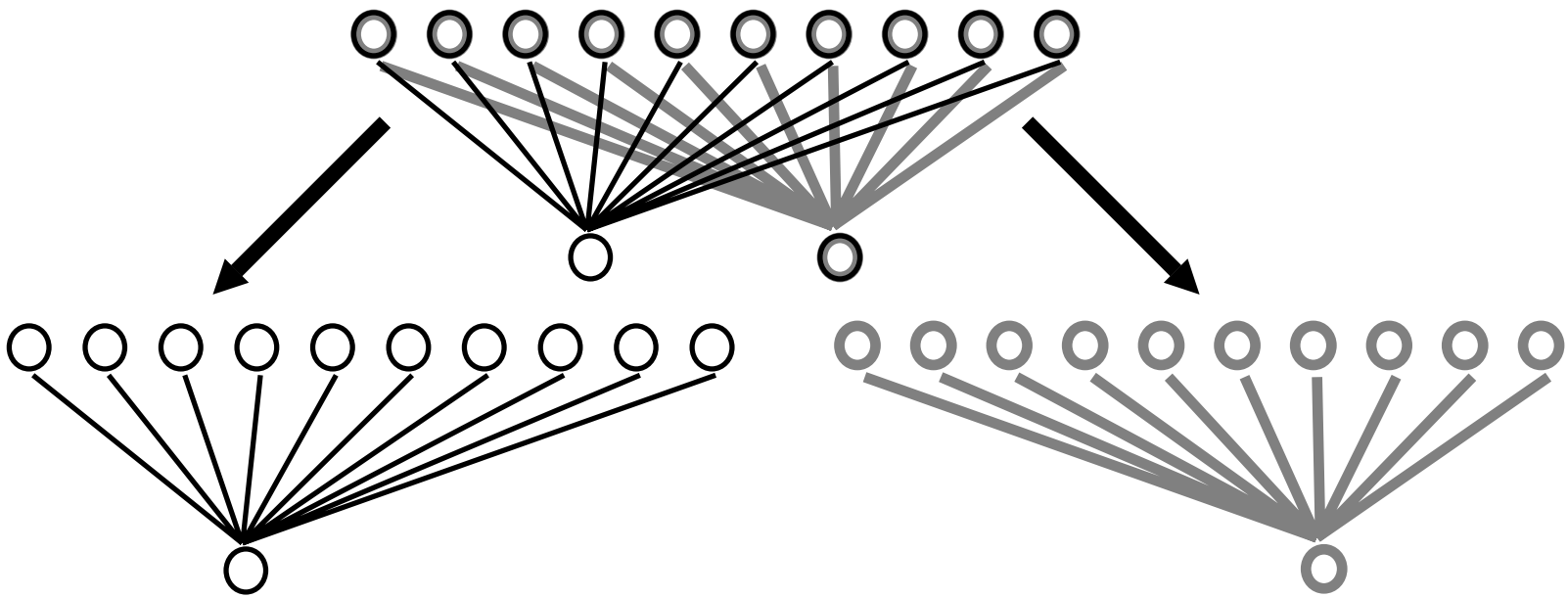
- Die input-Schicht hat Weglänge 0.

- Die Output-Schicht hat maximale Weglänge.



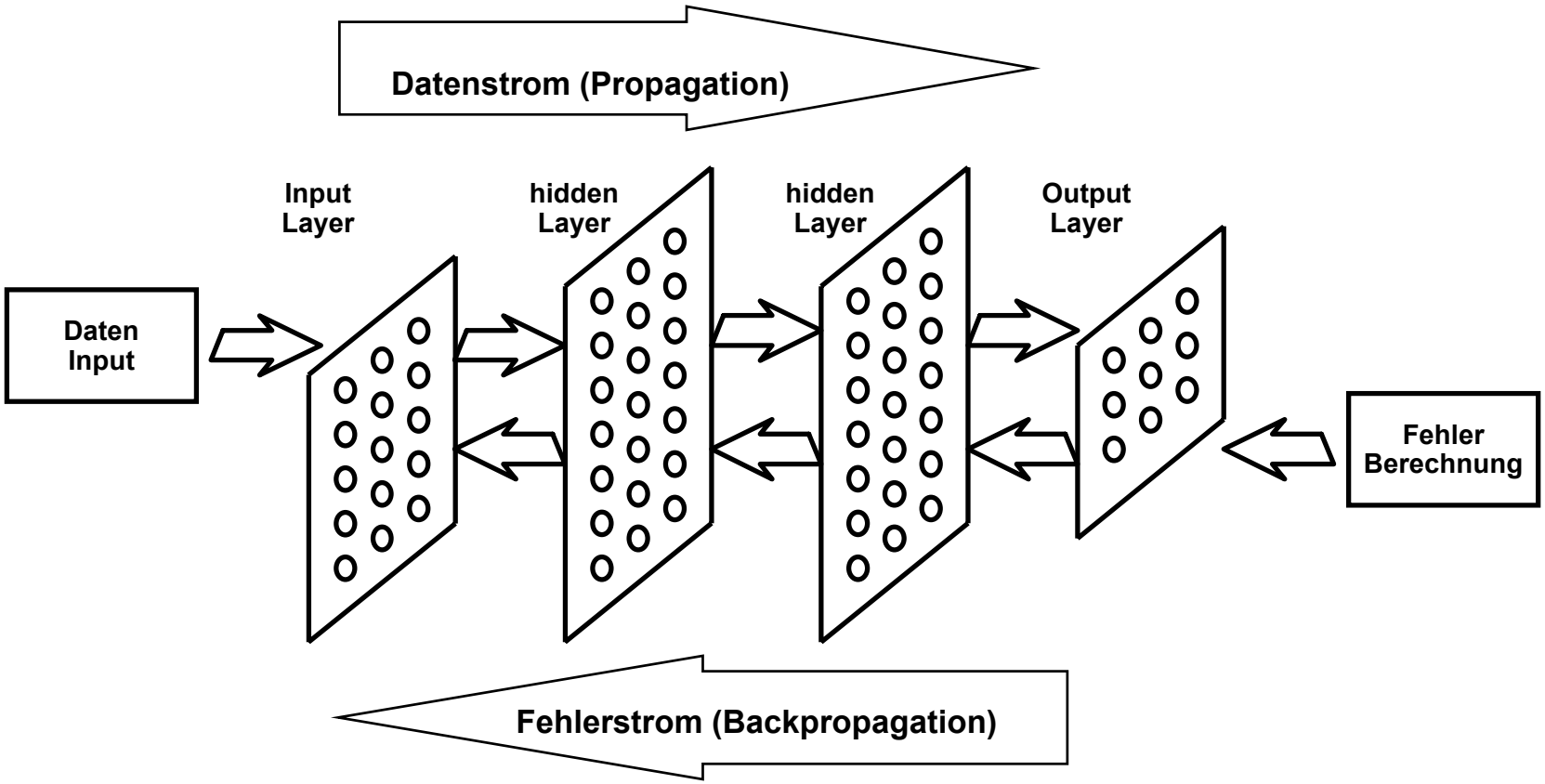
Multi Layer Perzeptron

Die beiden Neuronen der 2. Schicht beeinflussen einander nicht, deshalb können sie voneinander getrennt betrachtet werden.



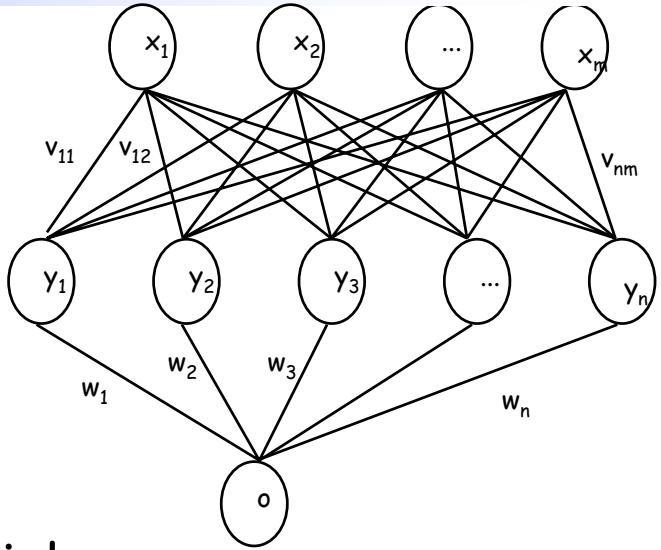
Bei mehrschichtigen Netzen geht die Unabhängigkeit verloren, d.h. sie können nicht so getrennt werden.

Backpropagation



Multi Layer Perzeptron

Fehlerfunktion F (mittlerer quatratischer Fehler) für das Lernen:



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

wobei gilt:

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

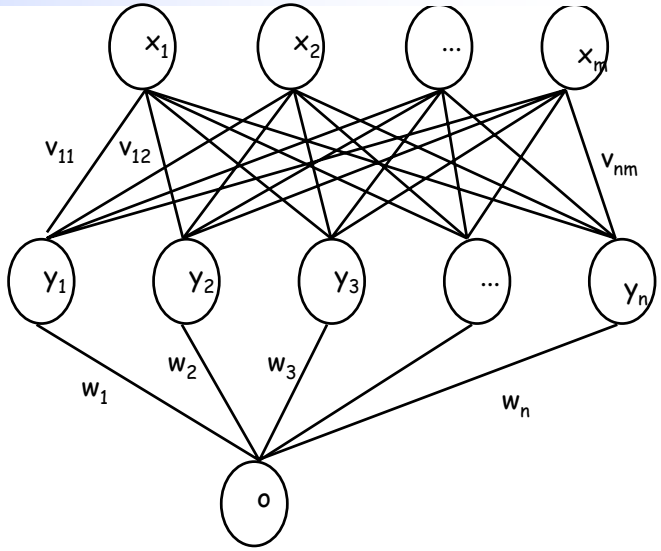
Anders geschrieben ist

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

Die Gewichte müssen so angepasst werden, daß der Fehler minimiert wird. Dazu bietet sich das Gradientenabstiegsverfahren an.

Multi Layer Perzeptron

Fehlergradient für w_i lautet:



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

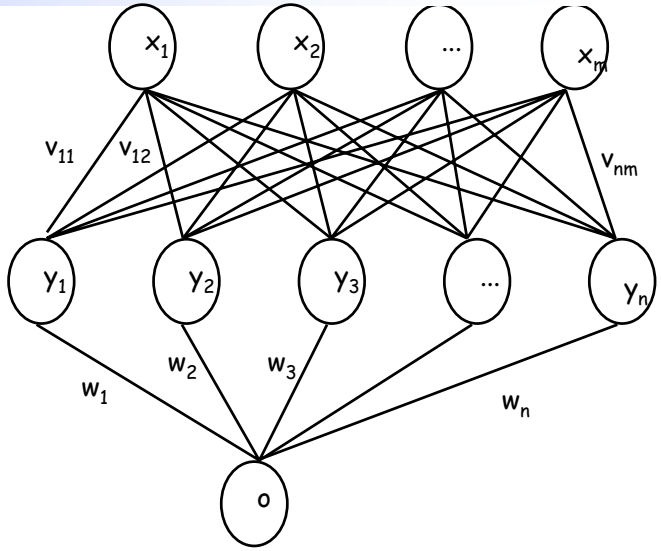
$$= \partial(\theta(\underline{\mathbf{w}}\cdot\underline{\mathbf{y}}) - t)^2/\partial w_i$$

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

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

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

Input

Info von Inputschicht

Multi Layer Perzeptron

- Die schichtweise Berechnung der Gradienten ist auch für mehr als zwei Schichten möglich
- Fehler wird 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

- Wähle ein Muster x aus der Menge der Trainingsbeispiele D aus
- präsentiere das Muster dem Netz und propagiere es
- anhand der resultierenden Neuronenaktivität wird der Fehler F ermittelt
- die Fehlerinformationen werden durch das Netz zurückpropagiert (Backpropagation)
- die Gewichte zwischen den einzelnen Schichten werden so angepasst (Δw_{ik}), dass der mittlere Ausgabefehler für das Muster sinkt

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:

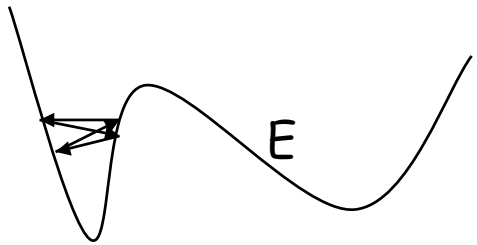
$$e(\mathbf{w} + \Delta\mathbf{w})^2 < e^2$$

Backpropagation Algorithmus

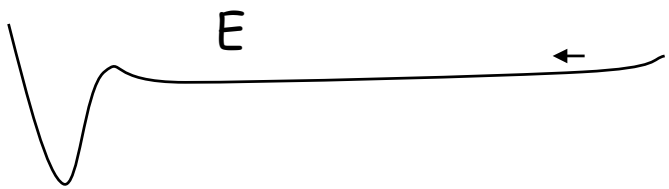
- Die Gewichtsänderungen können auf zwei Arten erfolgen:
  - Online Training:jedes Gewicht wird sofort angepaßt (folgt nur im Mittel dem Gradienten)
  - Batch Verfahren: es werden alle Datensätze präsentiert, die Gewichtsänderung des Gewichtes berechnet, summiert und dann erst angepaßt (entspricht dem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

Probleme des Backpropagation Algorithmus

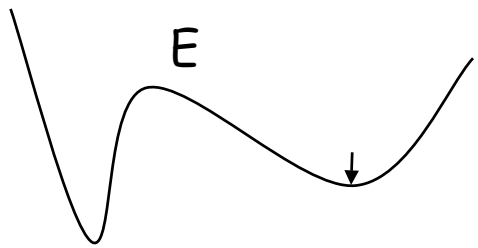
- Oszillation in engen Schluchten



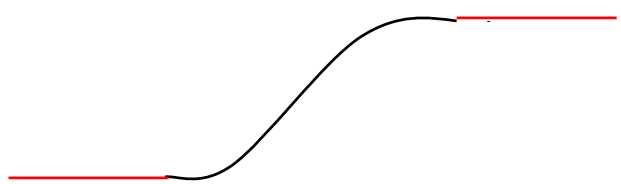
- Stagnation auf flachen Plateaus



- lokale Minima



- Flat Spots ($\theta'(a_j) \approx 0$) kaum Veränderung im Training



Varianten des Backpropagation Algorithmus

- Änderung der Fehlerfunktion
- Momentum-Term einführen
- Weight Decay
- Lernkonstanten und Output Funktion können für jede Schicht oder für jedes Neuron einzeln festgelegt werden
- Lernkonstanten können dynamisch angepaßt werden
- Gewichtsanpassung modifizieren (Manhatten Training)
- 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
- bel. Funktionen können durch ein ANN mit drei Schichten bel. 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 - Verfahren mit vorgegebenem Hintergrundwissen

VI.3.3.3 Abbruchbedingungen und Überspezialisierung

– Beschreibung des BACKPROPAGATION Algorithmus läßt Abbruchbedingung offen

- Algorithmus solange zu iterieren, bis Fehler für die Trainingsbeispiele unter einem vorgegebenen Schwellwert liegt, ist schlechte Strategie, da Algorithmus zur Überspezialisierung (overfitting) neigt
  $\Rightarrow$ (vgl. Entscheidungsbäume)

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

aber: Fehler auf Validierungsmenge muß nicht monoton fallen! (im Gegensatz zu Fehler auf Trainingsmenge. Vgl. auch Abbildung 8 a/b)

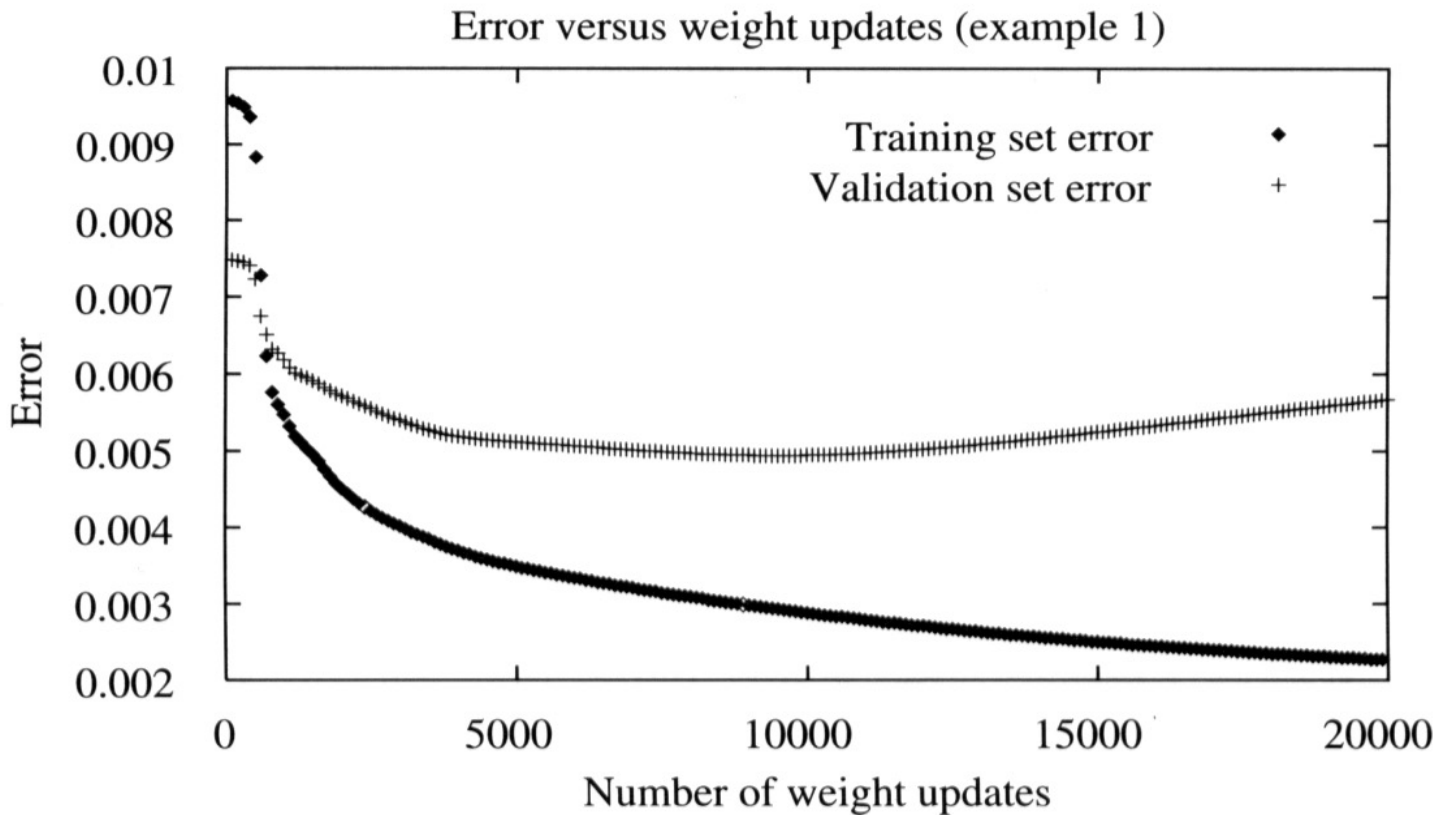


Figure 8a: Plots of error E as a function of the number of weight updates, for two different robot perception tasks (Mitchell 1997)

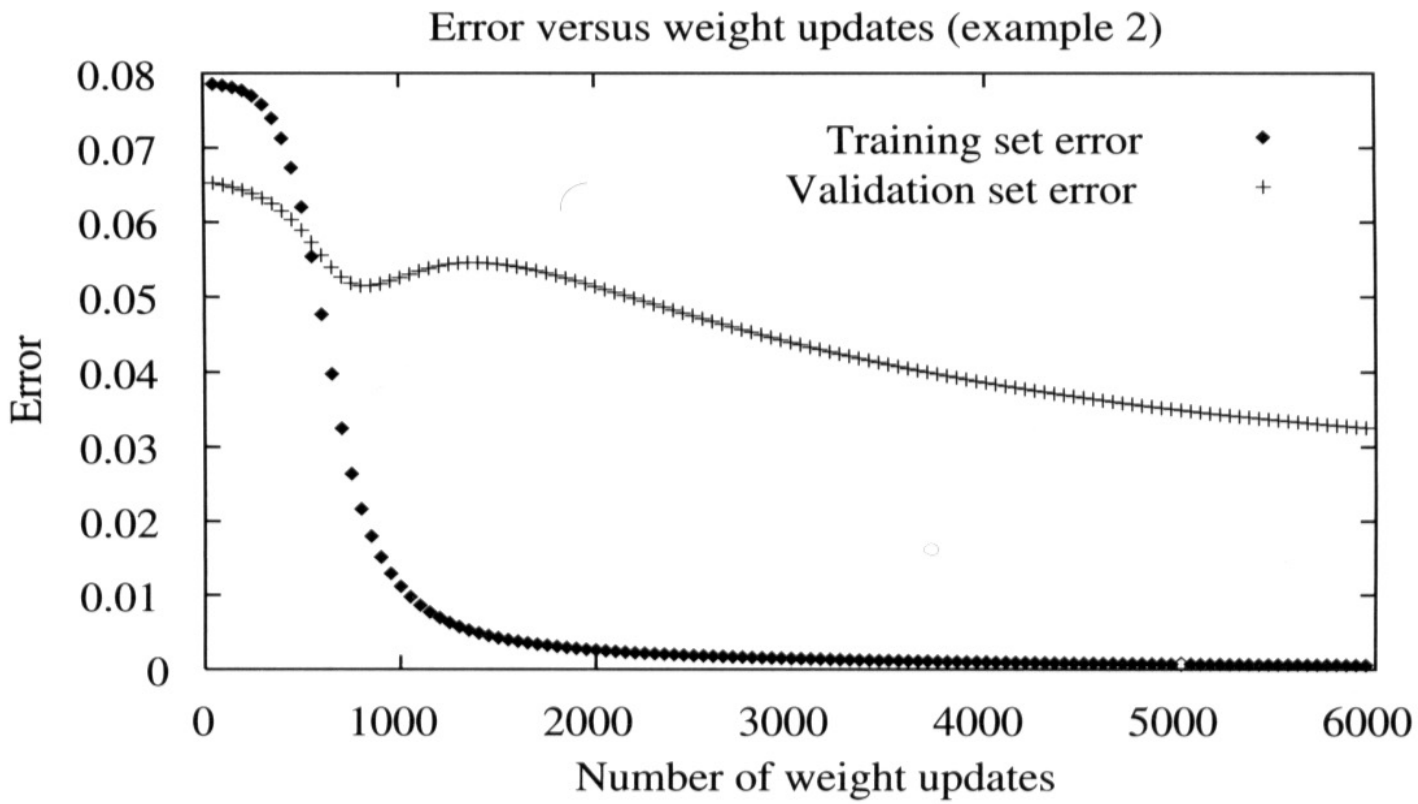


Figure 8b: Plots of error E as a function of the number of weight updates, for two different robot perception tasks (Mitchell 1997)

– Alternativ 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 Fehler für Beispiele aus Validierungsmenge)
  - Mittelwert von i über alle k Trainingsphasen wird letztendlich verwendet, um geg. 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.

Agenda

1. Einführung
2. Einfaches Perzeptron
3. Multi Layer Perzeptron
4. Beispiel
5. Rekurrente Netze
6. Entwurfsüberlegungen

Enkoderproblem

Voraussetzungen

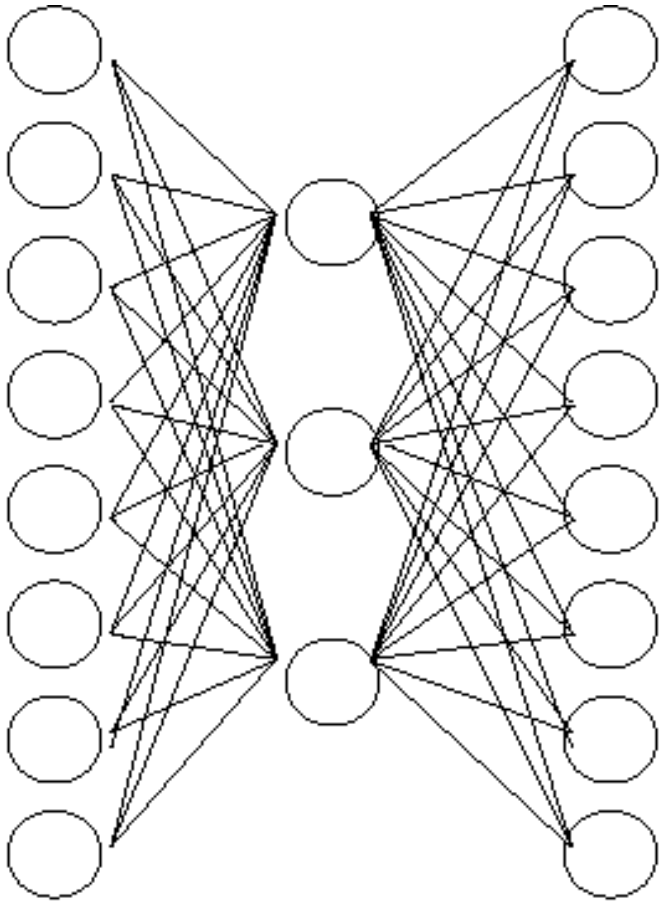
- Netz aus drei Schichten
- Ein- und Ausgabe enthalten jeweils N Neuronen
- die mittlere Schicht enthält nur M < N Neuronen

Lernaufgabe:

- Wenn Neuron n in der Eingabeschicht aktiviert wird, dann soll auch Neuron n auf der Ausgabeseite aktiviert werden.
- Die Aufgabe wäre trivial, wenn die mittlere Schicht ebenfalls N Neuronen hätte.

  ==> mittlere Schicht stellt Flaschenhals dar

Enkoderproblem



Input		Hidden Values				Output
10000000	-->	.89	.04	.08	-->	10000000
01000000	-->	.15	.99	.99	-->	01000000
00100000	-->	.01	.97	.27	-->	00100000
00010000	-->	.99	.97	.71	-->	00010000
00001000	-->	.03	.05	.02	-->	00001000
00000100	-->	.01	.11	.88	-->	00000100
00000010	-->	.80	.01	.98	-->	00000010
00000001	-->	.60	.94	.01	-->	00000001

Figure 7: Learned Hidden Layer Representation (Mitchell 1997)

Enkoderproblem

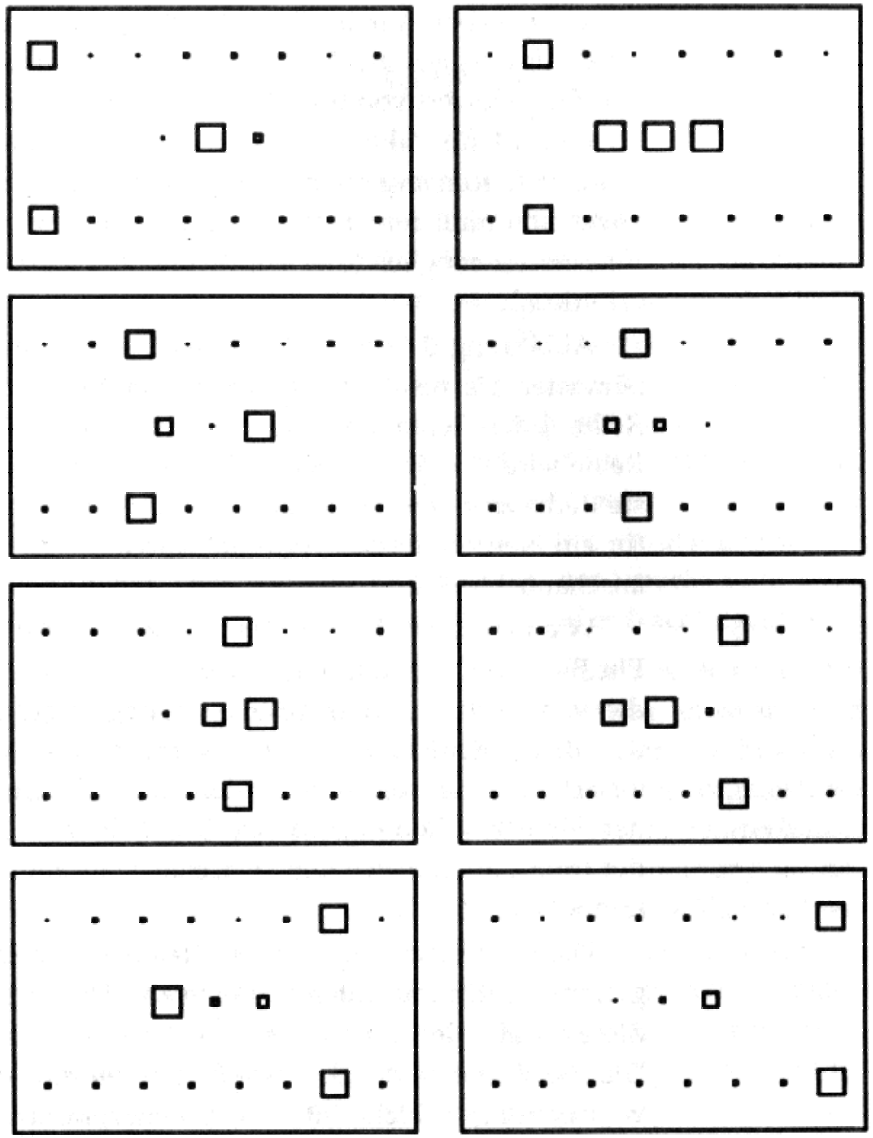
Lösung:

- Das Netz muss eine geeignete Kodierung für die Representation der aktivierten Neuronen in der mittleren Schicht finden.

Vorgehen:
- Gewichte werden zufällig gewählt
- 0.9 wird als 1 und 0.1 als 0 interpretiert (Konvergenzproblem)

Ergebnis:
- 3 bit Binärcode für die innere Repräsentation

Enkoderproblem



entnommen Ritter, S. 59

Enkoderproblem

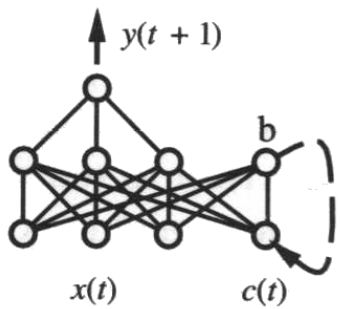
Frage: Welche interne Datenrepräsentationen sind erforderlich, um eine gegebene Aufgabe durch ein massiv paralleles Netzwerk lösen zu können?

- Backpropagation erlaubt das Trainieren von Netzen auf unterschiedliche Probleme.
- Die Analyse der „Neuroanatomie" gibt Einblicke in die Netzstruktur und ermöglicht ein besseres Verständnis auch von biologischen Netzwerken.

VI.3.3.5 Rekurrente Netzwerke

- Verarbeitung von Zeitreihen erfordert die Berechnung von Ausgabewerte zum Zeitpunkt (t+1) in Abhängigkeit der Ausgabewerte t, t-1, ...
  - z.B. bei der Prognose von Aktienkursen

- Zeitfenster in die Vergangenheit wird benötigt:



(*b*) Recurrent network

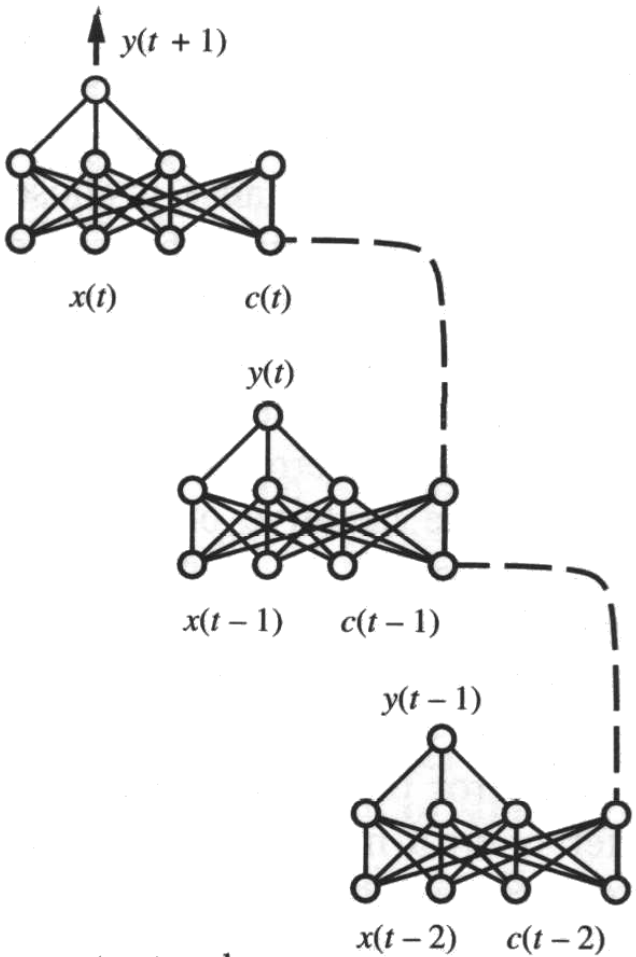
- Eingabewert c(t) ist gleich dem Wert der internen Einheit b zum Zeitpunkt (t-1)

- interne Einheit b sammelt Historie über Eingabewerte für das Netzwerk auf:

  - Wert von b(t) ist abhängig von aktuellen Eingabewerten x(t) und c(t), d.h. b(t-1)

  - damit ist Wert von b(t) indirekt abhängig von Eingabewerten x(t), x(t-1), ...

- im Prinzip kann ein rekurrentes Netzwerk mit einer leichten Variation des Backpropagation-Algorithmus trainiert werden (siehe z.B. Mozer 1995)



(c) Recurrent network unfolded in time

Figure 9: Recurrent Networks (Mitchell 1997)

VI.3.3.4 Entwurfsüberlegungen für ANNs

– Eingabecodierung:
- welche Repräsentation der vorliegenden Trainingsbeispiele ist günstig - Repräsentation muß fixe Anzahl von Eingabeelementen ermöglichen

– Ausgabecodierung:
- welche Repräsentation ist günstig
- sofern n diskrete Ausgabewerte berechnet werden sollen, bietet sich 1-aus-n-Codierung an (1-of-n output encoding):
  jeder Ausgabewert wird durch 1 Ausgabeeinheit repräsentiert
  - viele Kanten ermöglichen eine flexible Repräsentation der Ausgabefunktion über ihre zugehörigen Gewichte
  - Differenz zwischen dem höchsten und dem zweithöchsten Ausgabewert kann als Maß für die Güte der berechneten Ausgaben verwendet werden

– Struktur der internen Schicht

- wieviele Elemente soll man für die interne Schicht wählen?
- „gewisse minimale“ Anzahl von Elementen wird benötigt, um Zielfunktion genügend genau zu lernen
- Kreuzvalidierung kann Überspezialisierung durch zu große Anzahl interner Elemente vermeiden

aber: bisher fehlt klare Methodik für systematischen Entwurf von ANNs
(siehe z.B. (Anders 1997))

Weiteres Beispiel: Mustererkennung mit Hopfield-Netzen

→ Demo Hopfield.html

Bei Hopfield-Netzen sind die Ausgabe-Neuronen wieder mit den Eingabe-Neuronen verbunden.

Die Eingabe ist nun durch eine initiale Eingabe kodiert.

Das Netz versucht, durch Backpropagation in einen stabilen Zustand zu kommen.

In der Demo ist jedes Feld der Matrix einem von 64 Ausgabeneuronen zugeordnet.