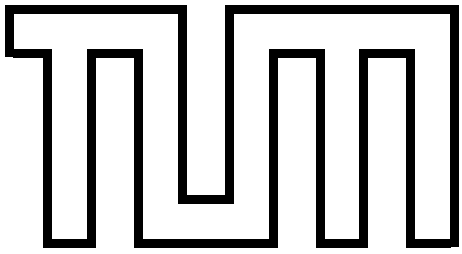


# Technische Universität München

## Fakultät für Informatik

### Forschungs- und Lehreinheit Informatik VI


# Neuronale Netze - Supervised Learning

Proseminar Kognitive Robotik (SS12)


**Hannah Wester**


**Betreuer:** Dr. Florian Röhrbein

**Leitung:** Prof. Dr.-Ing. habil. Alois Knoll

**Abgabetermin:** 21. Juli 2012

# Inhaltsverzeichnis

# 1 Einleitung

Die Arbeit soll einen Überblick über die Grundlagen von Neuronalen Netzen und deren Lernverfahren geben. Ein Neuronales Netz stellt die Neuronen des menschlichen Gehirns nach und versucht dessen Lernfähigkeit nachzuahmen, da es Problemstellungen gibt, die der Mensch ohne Probleme lösen kann, für die es uns aber schwer fällt einen Algorithmus zu finden.
Zu Beginn werden wir einfache Netze einführen, um den Aufbau der Netze verständlich zu machen und grundlegende Lernregeln anschaulich darzustellen. Anschließend gehen wir zu umfangreicheren Netzen mit dem Lernalgorithmus *Backpropagation* über.
Die in dieser Ausarbeitung behandelten Algorithmen gehören zur Kategorie des überwachten Lernens (engl. *supervised Learning*).
Überwachtes Lernen findet Anwendung bei Problemstellungen, wo wir bekannte Ausgangsdaten haben, aber keinen Algorithmus dafür finden können und dennoch Vorhersagen über weitere Ausgänge machen wollen. Beispiele dafür sind Spiellogiken oder auch die Wettervorhersage.

# 2 Das Perzeptron

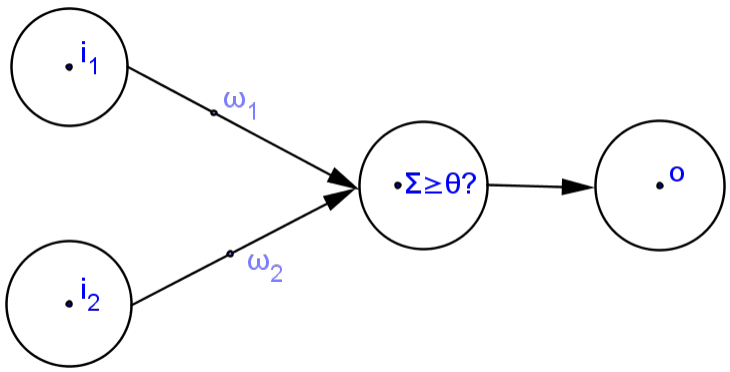


Abbildung 1: Einfaches Perzeptron

Das (einschichtige) Perzeptron (nach engl. perception, *Wahrnehmung*) beschreibt die einfachste Form eines linearen Neuronalen Netzes, dessen Konzept 1958 von Frank Rosenblatt vorgestellt wurde. Es geht aus der McCulloch-Pitts-Zelle hervor, die bereits 1943 entworfen wurde [7].
Das Perzeptron ist ein künstliches Neuronales Netz mit einstellbaren Gewichten und einem Schwellwert $\theta$. In seiner einfachsten Form besitzt es zwei Eingänge mit Eingangswerten $i_1$ und $i_2$ und einen Ausgang mit Wert $o$, die über einen gerichteten Graphen mit Gewichten

$\omega_1$ und $\omega_2$ verbunden sind und lediglich binäre Signale verwenden (siehe Abbildung 1). Der Ausgang besitzt einen Schwellwert und gibt den Wert Eins aus, falls die Summe der eingehenden Werte (also der Eingänge mit ihren Gewichten belegt) größer als dieser ist, ansonsten Null. Die binäre Ausgabe o ist also eine Funktion von ( $sum = \omega_1 i_1 + \omega_2 i_2) \geq \theta$.

## 2.1 Trainingsphase

Genauso wie die Neuronen in unserem Gehirn wollen wir auch unsere Perzeptronen lernfähig machen. Das Training geschieht durch Veränderung der Gewichte und des Schwellwertes. Der Einfachheit halber wird der Schwellwert (in diesem Fall häuftig als *Bias* bezeichnet) oft als Eingang 0 mit festem Wert $i_0 = -1$ definiert und dem verstellbaren Gewicht $\omega_0$, so dass man ihn als Eingang betrachten und sein Gewicht mit den anderen zusammen verändern kann.
Unter *überwachtem Lernen* versteht man, dass man ein Trainingsset hat, bei dem man zu einer Anzahl an Eingängen den Ausgang festlegt, den man erreichen möchte. Man wählt zufällige Gewichte und lässt die Eingänge des Trainingsset durch das Perzeptron laufen und überprüft, ob die gewollten Ausgänge $\bar{o}$ herauskommen. Ein Durchlauf durch ein Trainingsset wird als eine *Epoche* bezeichnet. Es gibt unterschiedliche Lernregeln, nach denen die Gewichte verändert werden. Zwei davon werden im nächsten Abschnitt vorgestellt.

## 2.2 Lernregeln

### 2.2.1 Hebb-Regel

Die Hebb-Regel wurde 1945 durch den Psychologen Donald Hebb bekannt. Sie ist ein Beispiel für *verstärkendes Lernen*.
Die Hebb-Regel basiert auf den Vorgängen im menschlichen Gehirn und verfolgt die Strategie, dass alle aktiven Eingänge mit aktiven Ausgängen - also alle Eingänge ungleich null mit Ausgang ungleich null - verstärkt werden sollen (wenn sie die gewünschte Ausgabe liefen). Der Faktor $\varepsilon \in \mathbb{R}$ zur Verstärkung (auch *Lernrate* genannt) kann beliebig zwischen $0 < \varepsilon < 1$ gewählt werden, jenachdem wie stark die Gewichte verändert werden sollen [2].

$$\Delta\omega_i = \varepsilon i_i o \quad \text{bzw. bei mehreren Ausgängen} \quad \Delta\omega_{ij} = \varepsilon i_i o_j$$

Die Hebb-Regel kann auch bei *nichtüberwachtem Lernen* genutzt werden.

### 2.2.2 Delta-Regel

Während die Hebb-Regel alle aktiven Eingänge verstärkt ohne die Größe des Fehlers zu berücksichtigen, ermittelt die Delta-Regel die Größe und kann dadurch verbessert lernen.

Die Delta-Regel verfolgt somit das *korrigierende Lernen*. Die Lernrate $\varepsilon$ wird ebenso wie bei der Hebb-Regel verwendet [2].

$$\Delta\omega_i = \varepsilon i_i(\bar{o} - o) \quad \text{bzw. bei mehreren Ausgängen} \quad \Delta\omega_{ij} = \varepsilon i_i(\bar{o_j} - o_j)$$

Veröffentlicht wurde die Delta-Regel 1960 durch Widrow und Hoff, weshalb sie auch als *Widrow-Hoff Regel* bekannt ist.

Anhand dieser Regel soll das Prinzip des überwachten Lernens an einem kleinen Beispiel verdeutlicht werden. Wir definieren uns ein Trainingsset für ein einfaches Perzeptron (Tabelle 1).

| Eingang1 | Eingang2 | Ausgang |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

Tabelle 1: Beispiel – Trainingsset

Zu Beginn setzen wir die Gewichte auf 1, der Eingang $i_0$ für den Schwellwert ist fest auf -1. Die Lernrate $\varepsilon$ setzen wir auf 0.2.

Den Ausgang können wir also folgendermaßen berechnen: $sum = \omega_0(-1) + \omega_1 i_1 + \omega_2 i_2$

$$o = \begin{cases} 1 \text{ falls } sum \geq 0 \\ 0 \text{ sonst} \end{cases}$$

Tabelle 2 zeigt, wie sich die Gewichte in einer Epoche (also einem Durchlauf durch das Trainingsset) verändern. In jeder Spalte stehen die neuen, geänderten Gewichte, nach Durchlauf der verschiedenen Eingangswerte.

| | $i_1 = 0/i_2 = 0$ | $i_1 = 0/i_2 = 1$ | $i_1 = 1/i_2 = 0$ | $i_1 = 1/i_2 = 1$ |
|---|---|---|---|---|
| $\omega_0$ | 1.0 | 1.0 | 1.2 | 1.2 |
| $\omega_1$ | 1.0 | 1.0 | 0.8 | 0.8 |
| $\omega_2$ | 1.0 | 1.0 | 1.0 | 1.0 |

Tabelle 2: Beispiel – Epoche 1

Die Gewichte sind nun auf $\omega_0 = 1.2$, $\omega_1 = 0.8$ und $\omega_2 = 1.0$ justiert. Dass das Ergebnis nicht zufriedenstellend ist, ist offensichtlich, da die Eingänge $i_1 = 0/i_2 = 1$ den Ausgang $o = 0$ liefern ($sum = -1.2 + 1.0 = -0.2$) und nicht wie gewünscht den Ausgang $\bar{o} = 1$.
Das Perzeptron durchläuft also eine weitere Lern-Epoche.

Bereits nach zwei Epochen haben sich die Gewichte ausreichend verändert:

$$\omega_0 : 1.0, \omega_1 = 0.8, \omega_2 = 1.2$$

| | $i_1 = 0/i_2 = 0$ | $i_1 = 0/i_2 = 1$ | $i_1 = 1/i_2 = 0$ | $i_1 = 1/i_2 = 1$ |
|---|---|---|---|---|
| $\omega_0$ | 1.2 | 1.0 | 1.0 | 1.0 |
| $\omega_1$ | 0.8 | 0.8 | 0.8 | 0.8 |
| $\omega_2$ | 1.0 | 1.2 | 1.2 | 1.2 |

Tabelle 3: Beispiel – Epoche 2

### 2.3 Ausführungsphase

Nach jeder Epoche durchläuft das Neuronale Netz die Ausführungsphase und überprüft, ob die gewünschten Ausgaben herauskommen. Falls dies nicht der Fall ist, wird eine weitere Epoche eingeleitet, bis das Netz trainiert genug ist.
Alternativ kann dem trainierten Netz in der Ausführungsphase ein neues Problem übergeben werden, also andere Eingaben, die das Erlernte erweitern und verifizieren, ob es auch diese erweiterten Probleme lösen kann. Ansonsten muss das Netz ebenfalls weiter trainiert werden.

### 2.4 Das XOR-Problem

Versuchen wir ein weiteres Beispiel mit der Delta-Regel zu lösen.

| Eingang1 | Eingang2 | Ausgang |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

Tabelle 4: Beispiel – Trainingsset

Tabelle 4 entspricht einem logischen XOR. Lassen wir nun die Delta-Regel mit diesem Trainingsset über das Perzeptron laufen,erhalten wir selbst nach tausenden Epochen keine sinnvollen Gewichte. Das folgende Kapitel soll erklären, weshalb dies der Fall ist und wie das Problem gelöst werden kann.

## 3 Mehrschichtige Netze

Ein mehrschichtiges Perzeptron (sog. *Multilayer Perzeptron*) hat neben den Ein- und Ausgangsschichten weitere Schichten, die sich versteckte Schichten (engl. *Hidden Layers*) nennen, da man auf diese von außen nicht zugreifen kann.
Die Ausgänge der ersten Schicht werden zu den Eingängen der versteckten Schichten, die Anzahl der Ein- und Ausgänge kann beliebig sein. Auch müssen die Eingangswerte nicht,

wie beim einfachen Perzeptron gefordert, binär sein, sondern können beliebige Werte annehmen. Abbildung 2 skizziert solch ein mehrschichtiges Netz mit zwei Ein- und zwei Ausgängen. Die Gewichte werden nun mit $\omega_{ij}$ bezeichnet, wobei i und j die Nummern der Neuronen bezeichnet, die durch den gewichteten Graphen verbunden sind.

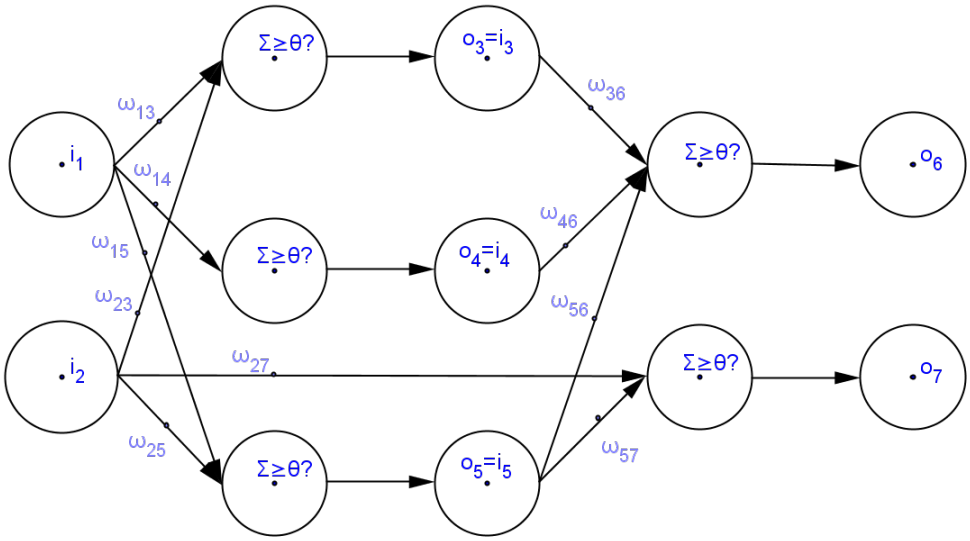


Abbildung 2: Mehrschichtiges Neuronale Netz

Meist zeigen die gerichteten Graphen nur von niedrigeren Schichten auf höhere, weshalb solche Netze auch Feedforward-Netze genannt werden. Es existieren allerdings auch (zyklische) Feedback-Netze mit Rückkopplung.

## 3.1 Lineare Trennbarkeit

Was genau bringen uns nun solche erweiterten Netze?
Unser Perzeptron summiert in Beispiel 1 die zwei Eingangswerte, vergleicht diese Summe mit dem Schwellwert und gibt abhängig davon null oder eins aus.
Da wir Gewichte ungleich null haben, kann, falls $\omega_2 > 0$ ist, die Ungleichung $\omega_1 i_1 + \omega_2 i_2 \geq \theta$ umgestellt werden zu

$$i_2 \geq \frac{\theta}{\omega_2} - \frac{\omega_1}{\omega_2} \cdot i_1$$

Falls das Gewicht $\omega_2 < 0$ ist, muss das Verhältniszeichen umgedreht werden. Graphisch gesehen ist dies in der durch $i_1$ und $i_2$ aufgespannten Ebene eine Halbebene, die durch die Gerade $i_2 = \theta/\omega_2 - \omega_1/\omega_2 \cdot i_1$ begrenzt wird. Diese Gerade trennt die Eingaben, bei denen der Ausgang null ausgibt, von denen, bei denen der Ausgang den Wert eins annimmt.
In Abbildung 3 ist dies für Beispiel 1 aufgezeichnet, die grünen Punkte bedeuten der Ausgang $o = 0$ und die blauen Punkte stehen für $o = 1$ [4].

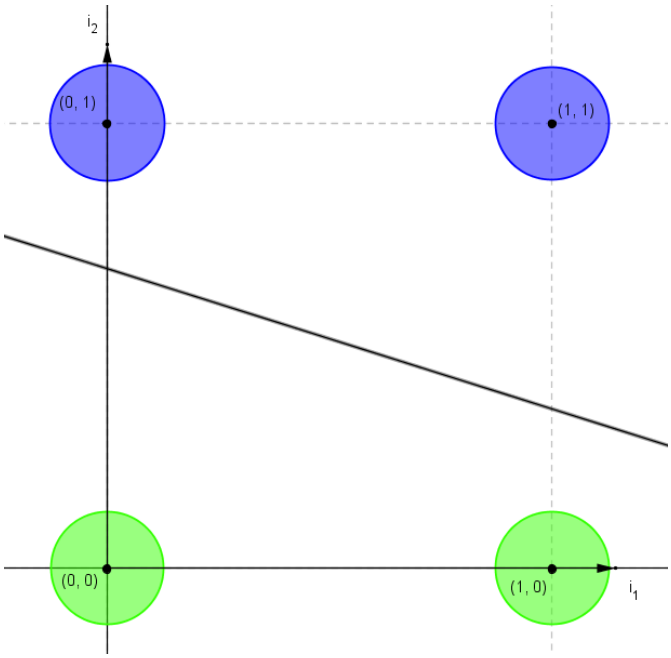


Abbildung 3: Lineare Trennbarkeit für Beispiel 1 – Trennbarkeit erreichbar

Betrachten wir nun das XOR-Beispiel, können wir die Ausgaben, die den Wert null ausgeben sollen, nicht mehr mit einer Gerade von den Ausgaben, die eins ergeben sollen, trennen, wie Abbildung 4 zeigt.

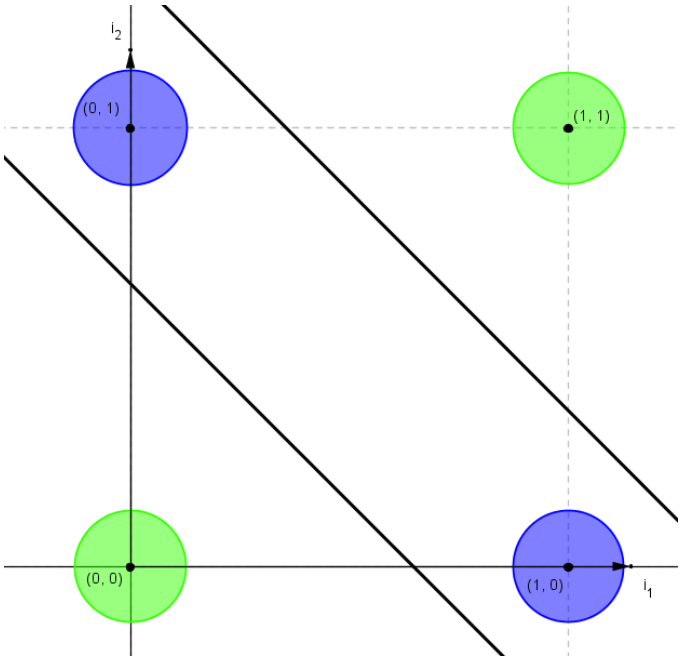


Abbildung 4: Lineare Trennbarkeit für Beispiel 2 (XOR) – Trennbarkeit nicht erreichbar

Wenn wir statt einschichtiger Netze zweischichtige verwenden, wird aus der Geraden eine Ebene, weshalb wir auch komplexere Ausgaben verlangen können.
Für n-schichtige Netze können wir also einen (n+1)-dimensionalen Würfel erstellen, dessen Eingaberaum durch eine n-dimensionale Hyperebene getrennt werden kann. Da mit dreischichtigen Netzen schon alle Mengen durch Schnitte und Überlagerungen getrennt werden können, werden 4- oder mehrdimesionale Netze praktisch nicht gebraucht [3].

Somit müssten wir unser XOR-Problem lösen können, doch wie können wir die Delta-Regel auf mehrschichtige Netze übertragen?

# 4 Nicht-lineare Netze

Um eine allgemeinere Regel zum Trainieren zu finden, sollten wir zunächst das neuronale Netz erneut verallgemeinern [3].

Bis jetzt haben wir unsere Eingabewerte immer addiert. Das wird auch im Normalfall gemacht, dennoch können die Werte natürlich auch multipiliziert werden oder lediglich die kleinste, oder die größte Eingabe übernommen werden. Wir haben also folgende *Netto-Inputfunktionen* zur Auswahl (so genannt, da sie sich nur aus der reinen Eingabe bildet, die meistens noch verifiziert wird):

$$\text{Summe:} \quad net_j = \sum_i \omega_{ij} \cdot i_i$$

$$\text{Produkt:} \quad net_j = \prod_i \omega_{ij} \cdot i_i$$

$$\text{Maximum:} \quad net_j = \max\{\ \omega_{ij} \cdot i_j\ \} \quad \text{bzw. Minimum:} \quad net_i = \min\{\ \omega_{ij} \cdot i_j\ \}$$

Abbildung 5: Netto-Inputfunktion $net_i$

In den vorherigen Abschnitten haben wir die Inputfunktion einfach übernommen und verglichen, ob sie größer dem Schwellwert, oder kleiner ist. Dies kommt einer linearen Funktion gleich, welche uns ziemlich einschränkt. Deshalb wollen wir auch die Menge unserer *Aktivierungsfunktionen* erweitern (In Abbildung 6 ist der Schwellwert $\theta$ schon in den Netto-Input $net_i$ mit einbefasst).

Desweiteren brauchen wir eine *Outputfunktion*, die aber im Normalfall einfach den Wert der Aktivierungsfunktion übernimmt, alternativ kann aber auch die Schwellwertfunktion verwendet werden, falls nur binäre Ausgaben gewünscht sind.

| | | | |
|---|---|---|---|
| lineare Schwellwertfunktion: | $a_i$ | = | $\begin{cases} 1 & \text{falls } net_i \geq 0 \\ 0 & \text{sonst} \end{cases}$ |
| linear bis Sättigung: | $a_i$ | = | $\begin{cases} 0 & \text{falls } net_i < 0 \\ net_i & \text{falls } 0 \leq net_i - < 1 \\ 1 & \text{sonst} \end{cases}$ |
| Sigmoide Funktion (Fermi Funktion): | $a_i$ | = | $\frac{1}{1+e^{-net_i}}$ |

Abbildung 6: Aktivierungsfunktion $f(net_i) = a_i$

| | | | |
|---|---|---|---|
| Identität: | $o_i$ | = | $a_i$ |
| lineare Schwellwertfunktion: | $o_i$ | = | $\begin{cases} 1 & \text{falls } a_i \geq 0 \\ 0 & \text{sonst} \end{cases}$ |

Abbildung 7: Outputfunktion $o_i$

Abbildung 8 versucht solch ein allgemeines neuronales Netz graphisch aufzuzeigen.

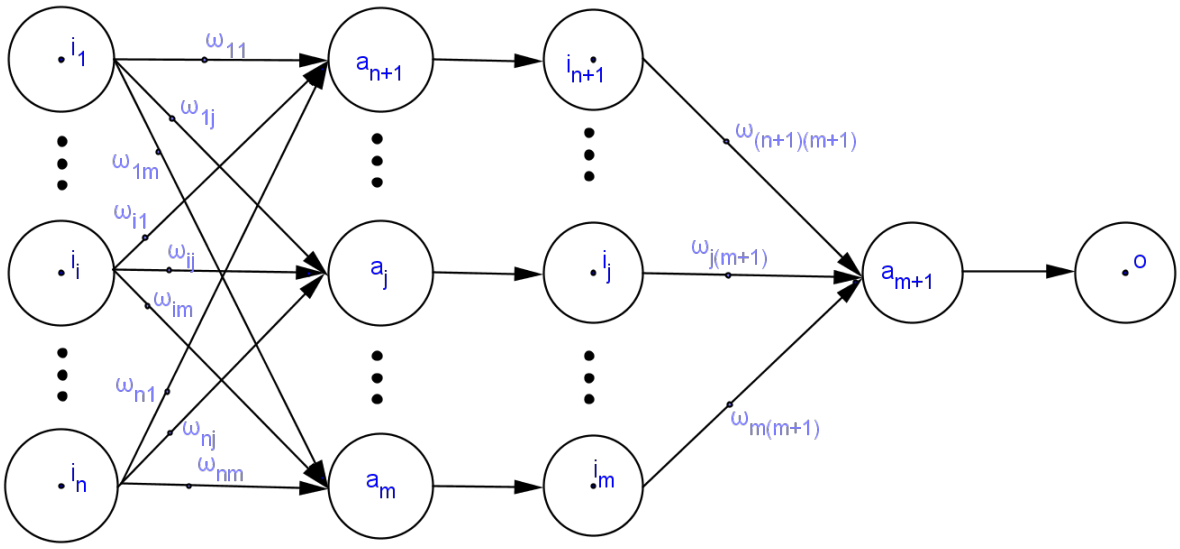


Abbildung 8: Mehrstufiges Netz mit nichtlinearen Aktivierungsfunktionen

## 4.1 Backpropagation

Da wir auf die versteckten Neuronen nicht ohne weiteres zugreifen können, müssen wir uns einen neuen Algorithmus überlegen, um auch mehrstufige, nicht lineare Netze zu trainieren.
Backpropagation (*Fehlerrückführung*) kam 1974 erstmals durch Paul Werbos auf und wurde 1986 durch eine Veröffentlichung von Rumelhart, Hinton und Williams bekannt. Es ist eine Erweiterung der Delta-Regel.
Der Fehler wird beim Trainieren am Ausgang festgestellt und anschließend rückwärts versucht auszubessern. Es wird eine Fehlerfunktion E definiert, die abgeleitet werden soll, um das Minimum, also den kleinsten Fehler zu finden.

$$E = \frac{1}{2}\sum_j (\bar{o}_j - o_j)^2$$

Man baut sich eine Fehlerfläche, indem man den Fehler des Netzes als Funktion der Gewichte graphisch darstellt und die Gewichte solange verändert, bis man das globale Minumum gefunden hat. Dieses Verfahren wird *Gradientenabstiegsverfahren* genannt [4].

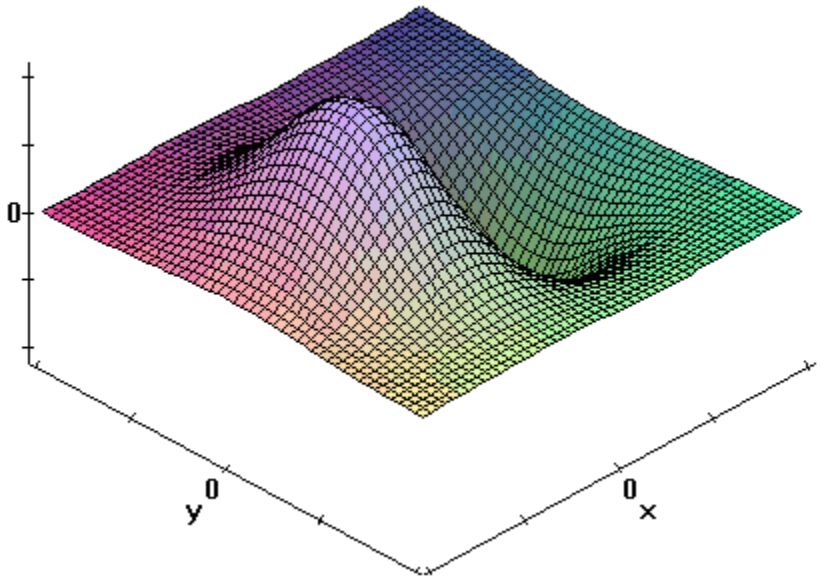


Abbildung 9: Gradientenabstiegsverfahren im dreidimensionalen Raum (Graphik aus [4])

Im zweidimensionalen Raum wird das Verfahren in Abbildung 10 verdeutlicht.

Man vergrößert sein Gewicht $\omega$ solange, bis die Fehlerfunktion ein lokales Minimum erreicht hat. Deshalb ist es wichtig, die Lernrate $\varepsilon$ möglichst gering zu wählen, um nicht das Minimum zu "überspringen".

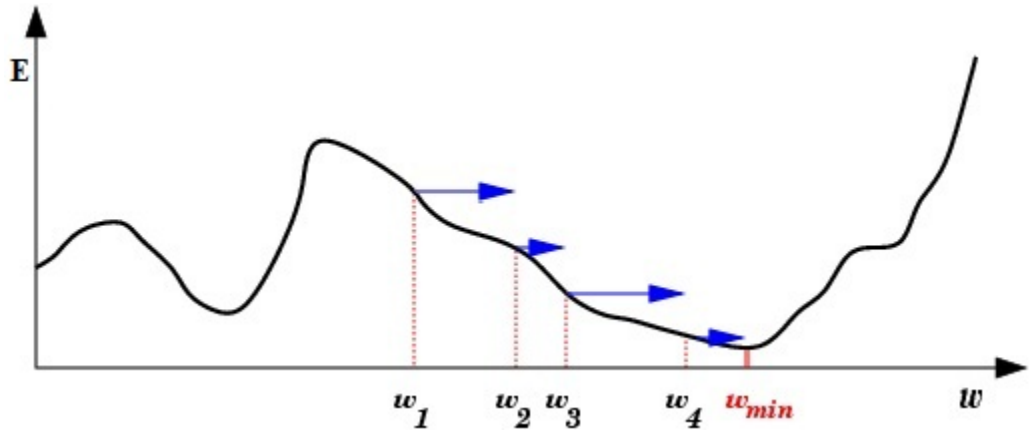


Abbildung 10: Gradientenabstiegsverfahren im zweidimensionalen Raum (modifizierte Graphik aus [4])

Die Verallgemeinerung der Delta-Regel ergibt nun eine Änderung der Gewichte durch partielle Ableitung der Fehlerfunktion nach den Gewichten [5]:

$$\Delta\omega_{ij} = -\varepsilon\frac{\partial E}{\partial \omega_{ij}}$$

Die partielle Ableitung kann durch Umformung geschrieben werden als

$$\frac{\partial E}{\partial \omega_{ij}} = \frac{\partial E}{\partial o_j}\frac{\partial o_j}{\partial net_j}\frac{\partial net_j}{\partial \omega_{ij}}$$

Da $net_j = \sum_i \omega_{ij} i_i - \theta$ und $o_i = f(net_i)$, ergeben sich die Ableitungen

$$\frac{\partial net_j}{\partial \omega_{ij}} = i_i \text{ und } \frac{\partial o_j}{\partial net_j} = f'(net_j)$$

$-\frac{\partial E}{\partial o_j} \cdot f'(net_j)$ ersetzen wir durch $\delta_j$, sodass unsere neue Formel lautet: $\Delta\omega_{ij} = \varepsilon\delta_j i_i$

Es muss unterschieden werden, ob $o_j$ in der Ausgabeschicht, oder in den versteckten Schichten liegt, um die Fehlerfunktion abzuleiten.
$o_j$ in Ausgabeschicht:

$$-\frac{\partial E}{\partial o_j} = -2 \cdot \frac{1}{2}(\bar{o_j} - o_j) \cdot (-1) = (\bar{o_j} - o_j)$$

$o_j$ in versteckter Schicht:

$$-\frac{\partial E}{\partial o_j} = -\sum_k \frac{\partial E}{\partial net_k}\frac{\partial net_k}{\partial o_j} = -\sum_k \frac{\partial E}{\partial net_k}\frac{\partial}{\partial o_j}\sum_i \omega_{ik} i_i = -\sum_k \frac{\partial E}{\partial net_k}\omega_{jk} = \sum_k \delta_k \omega_{jk}$$

Es ergibt sich somit $\Delta\omega_{ij} = \varepsilon\delta_j i_i$ mit

$$\delta_j = \begin{cases} f'(net_j)(\bar{o_j} - o_j) \text{ falls } o_j \text{ sich in der Ausgabeschicht befindet} \\ f'(net_j)\sum_k \delta_k\omega_{jk} \text{ falls } o_j \text{ sich in einer versteckten Schicht befindet} \end{cases}$$

Somit haben wir einen Algorithmus gefunden, der sowohl unser XOR-Problem lösen, als auch deutlich kompliziertere Netze trainieren kann (für Interessierte kann ich das Buch *Praktikum Neuronale Netze* empfehlen, man bekommt unteranderem gezeigt, wie man einem Neuronalen Netz das bekannte Spiel Mühle beibringen kann [1]).
Da die Schwellwertfunktion nicht differenzierbar ist, nehmen wir beispielsweise eine sigmoide Funktion als Aktivierungsfunktion und die Schwellwertfunktion als Outputfunktion, damit wir dennoch binäre Ausgaben haben. Graphisch zeigt Abbildung 11 ein für das XOR-Problem trainiertes Perzeptron.

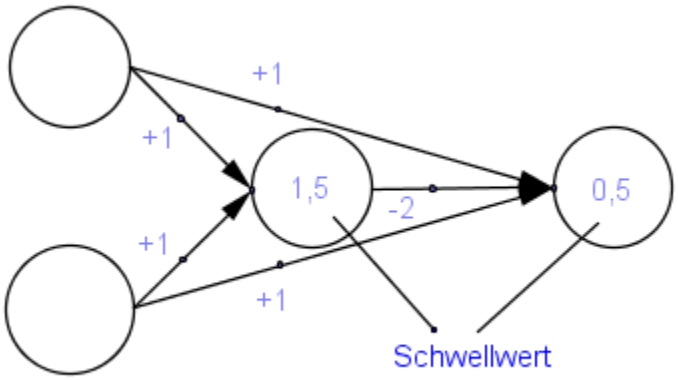


Abbildung 11: Lösung des XOR-Problems mit mehrschichtigem Netz (Graphik aus [4])

# Legende

- $a_i$ Aktivierungsfunktion für Ausgang i
- $E$ Fehlerfunktion für Backpropagation
- $\varepsilon$ Lernrate (Faktor zur Verstärkung)
- $i_i$ Eingangswert am Eingang i
- $net_i$ Netto-Inputfunktion die der Aktivierungsfunktion übergeben wird
- $o_i$ Ausgangswert am Ausgang i
- $\bar{o_i}$ vom Trainingssatz gewünschter Ausgangswert des Ausgangs i
- $\omega_i$ Gewicht in einem einschichtiges Neuronalen Netz mit einem Ausgang zwischen einem Eingang i und dem Ausgang
- $\omega_{ij}$ Gewicht zwischen Eingang i und Ausgang j in einem Neuronalen Netz
- $\theta_i$ Schwellwert am Ausgang i (auch Bias genannt)

# Hilfsmittel

Verwendete Medien:

- TeXworks editor
- die Grafikprogramme *GeoGebra* und *Paint* zur Erstellung aller nichtgekennzeichneten Abbildungen

# References


[1] Heinrich Braun Feulner Johannes Malaka Rainer. *Praktikum neuronale Netze*. Springer Verlag, 1997.

[2] Fabian Beck Günter Daniel Rey. *Neuronale Netze - Eine Einführung*. URL: `http://www.neuronalesnetz.de/`.

[3] David Kriesel. *Eine Einführung in Neuronale Netze*. erhältlich auf http://www.dkriesel.com.

[4] Prof. Dr. W.-M. Lippe. *Interaktive "Einführung in Neuronale Netze"*. 2007. URL: `http://cs.uni-muenster.de/Studieren/Scripten/Lippe/wwwnnscript/index.html`.

[5] Daniel Rios. *Backpropagation Neural Network*. 2007-2010. URL: `http://www.learnartificialneuralnetworks.com/backpropagation.html`.

[6] Raul Rojas. "Neural Networks - A Systematic Introduction". In: Springer-Verlag, 2009. Chap. 1-5,7.

[7] Benno Stein. *Neuronale Netze - Unit. Perzeptron-Lernen*. 2005-2010. URL: `http://www.uni-weimar.de/medien/webis/teaching/lecturenotes/machine-learning/unit-de-perceptron-learning.pdf`.

[8] Helge Ritter Thomas Martinetz Klaus Schulten. "Neuronale Netze - Eine Einführung in die Neuroinformatik selbstorganisierender Netzwerke". In: Addison-Wesley GmbH, 1991. Chap. 3.9. (Backpropagation-Algorithmus).