Proseminar Machine Learning

Neuronale Netze: mehrschichtige Perzeptrone

Christina Schmiedl
Betreuer: Christian Spieth, Andreas Dräger

27.Mai 2006

Inhaltsverzeichnis

1 Biologische Motivation

Das Nervensystem von Tieren besteht – unabhängig von der Komplexität – aus Neuronen. Neuronen (oder auch Nervenzellen) empfangen Signale und leiten diese an andere Neuronen weiter.

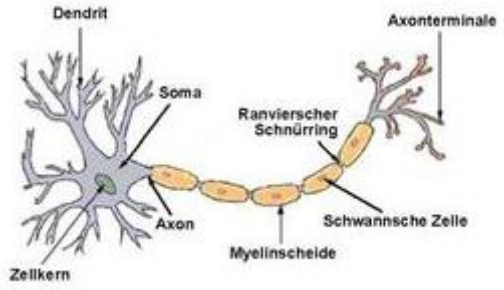

Abbildung 1: Aufbau eines Neurons [1]

In Abbildung 1 wird der Aufbau eines Neurons schematisch dargestellt: es besteht aus Dendriten, dem Soma (Zellkörper) und dem Axon und steht über Synapsen mit anderen Neuronen in Verbindung. Die ankommenden Signale anderer Nervenzellen werden von den Dendriten aufgenommen. Am Soma werden die Änderungen des Membranpotenzials aufsummiert und je nachdem, ob die Feuerschwelle überschritten wird, am Axonhügel nach dem Alles- oder Nichts-Gesetz ein Aktionspotential ausgelöst. Dieses wird über das Axon bis zu den Endknöpfchen, welche den präsynaptischen Teil der Synapse bilden, weitergeleitet.

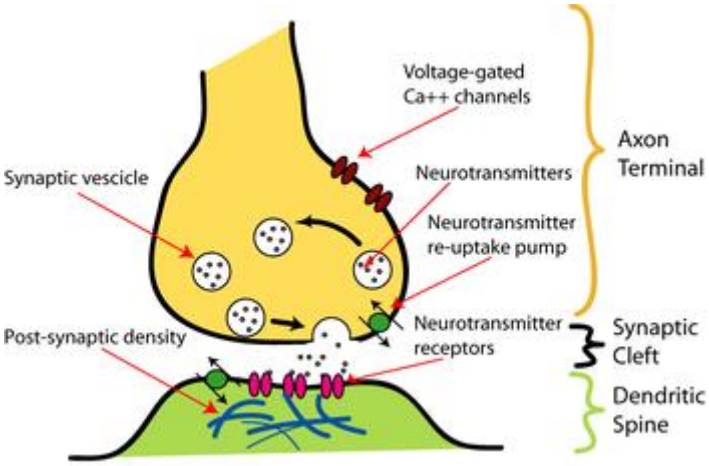

Abbildung 2: synaptische Übertragung [2]

Wie in Abbildung 2 dargestellt ist, wird die Erregung über den synaptischen Spalt auf die Dendriten einer anderen Nervenzelle übertragen. Dies geschieht durch Ausschüttung von Neurotransmittern (z. B. Acetycholin) in den synaptischen Spalt. Diese binden an Rezeptoren auf der postsynaptischen Membran, was dazu führt, dass Ionenkanäle geöffnet werden, die zur Entstehung eines elektrischen Signals beitragen. Durch diesen Mechanismus wird das Signal von einer Nervenzelle auf eine andere übertragen.
(Abschnitt 1 bezieht sich auf [1] und [8])

2 Neuronale Netze

2.1 Grundlagen

Mit Hilfe dieser biologischen Grundlagen kann man nun versuchen, ein abstraktes Modell für künstliche neuronale Netze zu formulieren.

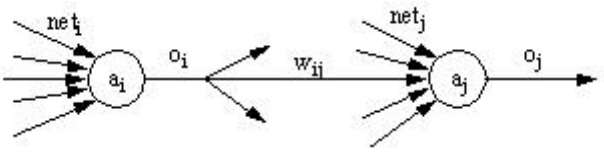

Abbildung 3: künstliches Neuronales Netz [7]

Wie im biologischen Vorbild gibt es verschiedene Neuronen, die über gerichtete, gewichtete Verbindungen w_{ij} miteinander verknüpft sind. Dabei bezeichnet w_{ij} das Gewicht zwischen Neuron i und Neuron j. Die Netzeingabe eines Neurons j wird abhängig von den Ausgaben der Vorgängerneuronen, die eine Verbindung mit diesem Neuron haben, und deren Verbindungsgewichten mit Hilfe der Propagierungsfunktion berechnet:

$$net_j = \sum_i o_i(t) \cdot w_{ij}$$

Der Schwellenwert Θ_j entspricht im biologischen Sinne der Feuerschwelle. Ist die Netzeingabe eines Neurons j größer als Θ_j, so feuert das Neuron.
Der Aktivierungszustand $a_j(t)$ gibt den Aktivierungsgrad der Zelle an. Die Aktivierungsfunktion f_{act} errechnet den neuen Aktivierungszustand $a_j(t+1)$ aus dem vorherigen Aktivierungszustand, der Netzeingabe und dem Schwellenwert:

$$a_j(t+1) = f_{act}(a_j(t), net_j(t), \Theta_j)$$

Oftmals ist der neue Aktivierungszustand unabhängig von dem vorherigen Aktivierungszustand $a_j(t)$. In Abschnitt 2.3 wird zusätzlich beschrieben, wie man den Schwellenwert in die Berechnung der Netzeingabe einrechnen kann. Somit kann es sein, dass die Aktivierungsfunktion nur noch von der Netzeingabe abhängig ist.

Die Ausgabe o_j einer Zelle j wird durch die Ausgabefunktion f_{out} bestimmt:

$$o_j = f_{out}(a_j)$$

2.2 Anzahl der Schichten

Ein Neuronales Netz besteht aus einer Eingabeschicht, einer Ausgabeschicht und dazwischen n verdeckten Schichten.

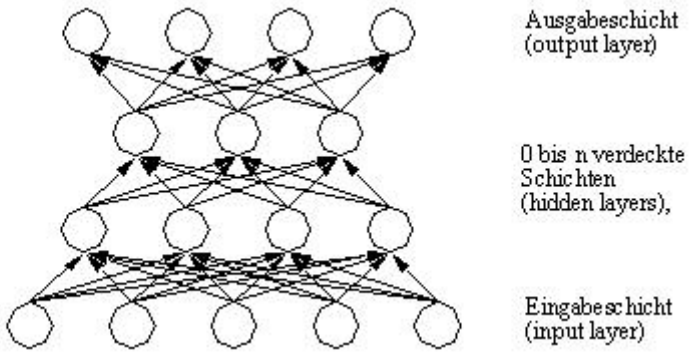

Abbildung 4: Neuronales Netz [7]

Leider wird die Anzahl der Stufen Neuronaler Netze in der Literatur nicht eindeutig verwendet. In dieser Ausarbeitung wird ein Netz als n-stufig bezeichnet, wenn es n Schichten trainierbarer Verbindugen hat. Das Neuronale Netz in Abbildung 4 wird also als 3-stufig bezeichnet.

2.3 On-Neuron

Für viele Anwendungen ist es einfacher, für jedes Neuron mit einem konstanten Schwellenwert 0 zu rechnen. Dies kann mit einem sogenannten On-Neuron realisiert werden:

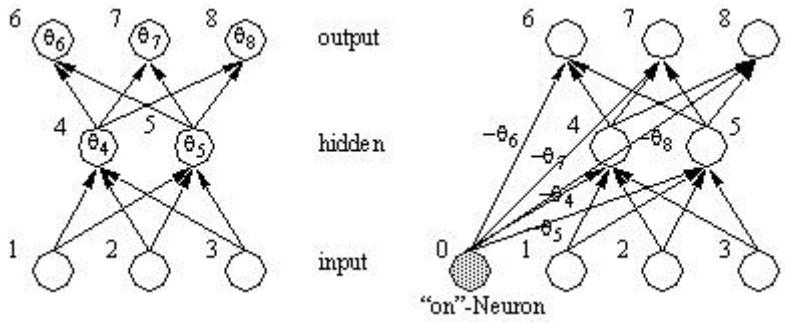

Abbildung 5: Neuronales Netz ohne und mit On-Neuron [7]

Es gibt zusätzlich ein Eingabeneuron, welches immer auf 1 gesetzt ist, und von welchem jeweils eine mit dem negativen Schwellenwert Θ_j gewichtete Verbindung zu Neuron j verläuft (vgl. Abbildung 5 rechts). Da dies jedoch schnell unübersichtlich wird, würde man zur graphischen Verdeutlichung Abbildung 5 links bevorzugen. Aber zum Beispiel für den Backpropagation-Algorithmus ist die Realisierung mit einem On-Neuron von Vorteil, da sonst neben den Gewichten auch die Schwellenwerte angepasst werden müssten. Der Schwellenwert geht also schon in die Netzeingabe mit ein und nicht mehr in die Aktivierungsfunktion.
(Abschnitt 2 bezieht sich auf [9])

3 Perzeptron

3.1 Definition

Historisch gesehen gibt es keine allgemeine Definiton des Perzeptrons. Als Perzeptron wird eine ganze Familie verwandter Modelle Neuronaler Netze bezeichnet, die von Frank Rosenblatt Anfang der 60er Jahre entwickelt wurden. Dieses Modell wurde von Minsky und Papert verfeinert. Ein allgemeines Perzeptron hat folgende Gestalt:

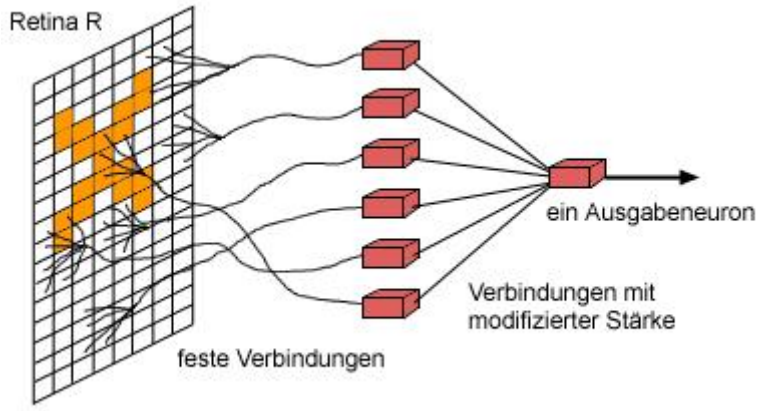

Abbildung 6: Struktur eines Perzeptrons [3]

Für die bei Perzeptronen zumeinst untersuchten Anwendungen der visuellen Mustererkennung, dienen die Zellen der Retina (Netzhaut) als Eingabe. Diese haben feste Verbindungen, können also von einem Lernalgorithmus nicht verändert werden. Bei Perzeptronen gibt es lediglich ein Ausgabeneuron, welches variable Verbindungen zu den Vorgängerneuronen besitzt. Oftmals können in einem Perzeptron die Eingaben und die Aktivierungen der Neuronen nur binäre Werte annehmen.

3.2 Lernfähigkeit und lineare Trennbarkeit

Wichtig für das Verständnis des Perzeptrons ist es, sich den Unterschied zwischen Lernfähigkeit und Repräsentierbarkeit klar zu machen:

Repräsentierbarkeit Fähigkeit eines Netzes, eine Funktion realisieren zu können

Lernfähigkeit Fähigkeit eines Lernalgorithmus, Gewichte und Schwellenwerte durch einen Algorithmus korrekt bestimmen zu lassen

Die Repräsentierbarkeit ist somit von der Topologie abhängig, die Lernfähigkeit hängt jedoch vom gewählten Lernalgorithmus ab.

Der zentrale Satz über den Zusammenhang von Repräsentierbarkeit und Lernfähigkeit eines Perzeptrons wurde von F. Rosenblatt formuliert:

Perzeptron-Lern-Theorem
Der Lernalgorithmus des Perzeptrons konvergiert in endlicher Zeit, d. h. das Perzeptron kann in endlicher Zeit alles lernen, was es repräsentieren kann.

Dieses Theorem wurde früher, als die Unterscheidung in Repräsentierbarkeit und Lernfähigkeit noch nicht bekannt war, oft falsch verstanden.

Nun stellt sich natürlich die Frage, was ein Perzeptron mit einer bestimmten Topologie überhaupt repräsentieren kann. Dazu wird ein einschichtiges Perzeptron mit zwei Eingaben betrachtet, welches die XOR-Funktion repräsentieren soll.

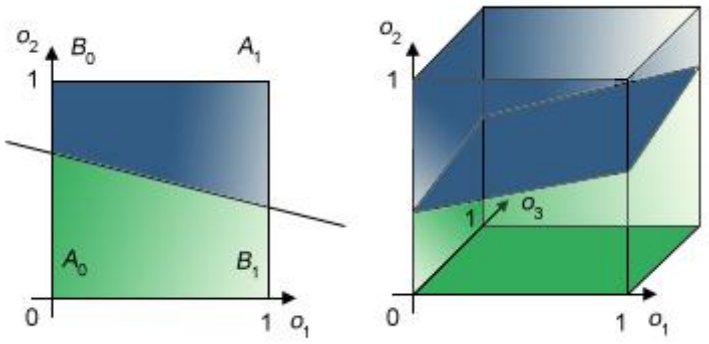

Abbildung 7: Lineare Trennbarkeit am Beispiel des XOR-Problems [4]

Um das XOR-Problem zu lösen, müsste das Perzeptron A_0 und A_1 (Ausgabe = 0) von B_0 und B_1 (Ausgabe = 1) linear trennen können. Mit einem einschichtigen Perzeptron lässt sich der Eingaberaum durch eine Gerade, die man durch Anpassung der Gewichte beliebig verschieben und drehen kann, linear trennen. Wie aus Abbidung 7 links hervorgeht, ist dies bei der XOR-Funktion nicht möglich. Es gibt somit kein einschichtiges Perzeptron, welches das XOR-Problem lösen kann.
Im dreidimensionalen Fall – es gibt also 3 Eingaben – berechnet das Perzeptron eine Ebene (siehe Abbildung 7 rechts), es kann also alle Punkte über dieser Ebene von den Punkten unter der Ebene trennen. Im mehrdimensionalen Fall ergibt sich eine Hyperebene.
Der Prozentsatz der linear trennbaren Funktionen mit n Variablen geht gegen 0. Mit 2 Variablen kann man 16 verschiedene Funktionen realisieren, von denen 14 linear trennbar sind. Mit 4 Variablen lassen sich $65\,536$ Funktionen realisieren, davon sind allerdings nur noch 1772 linear separierbar.
Man sieht also, dass einstufige Perzeptrone nur für einfache Probleme mit einer geringen Anzahl von Eingaben geeignet sind.

3.3 XOR-Netzwerk als Beispiel

Es wird nun ein kleines Beispiel zu einem Neuronalen Netz gegeben. Das folgende zweischichtige Netz soll das XOR-Problem lösen:

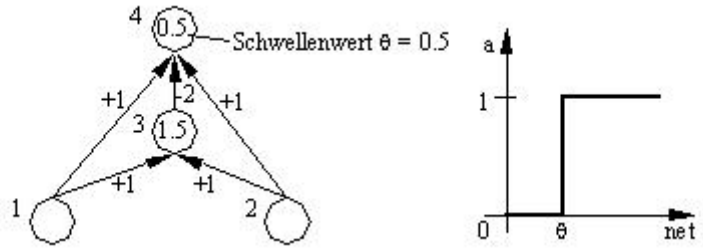

Abbildung 8: XOR-Netzwerk [7]

Das Beispiel wird nur mit binären Eingaben und mit der binären Schwellwertfunktion

$$o_j = a_j = \begin{cases} 1, & \text{falls} \quad net_j \geq \Theta_j \\ 0, & \text{sonst} \end{cases}$$

berechnet. Die Ausgabe ist in der folgenden Tabelle verdeutlicht:

o_1	o_2	net_3	o_3	net_4	o_4
0	0	$(0 \cdot 1 + 0 \cdot 1) = 0$	0	$(0 \cdot 1 + 0 \cdot 1 + 0 \cdot (-2) = 0)$	0
0	1	$(0 \cdot 1 + 1 \cdot 1) = 1$	0	$(0 \cdot 1 + 1 \cdot 1 + 0 \cdot (-2) = 1)$	1
1	0	$(1 \cdot 1 + 0 \cdot 1) = 1$	0	$(1 \cdot 1 + 0 \cdot 1 + 0 \cdot (-2) = 0)$	1
1	1	$(1 \cdot 1 + 1 \cdot 1) = 2$	1	$(1 \cdot 1 + 1 \cdot 1 + 1 \cdot (-2) = 0)$	0

Es kann leicht mit Hilfe der Tabelle verifiziert werden, dass dieses zweischichtige Perzeptron das XOR-Problem löst.

3.4 Zweischichtiges Perzeptron

Dieses Beispiel verdeutlicht, dass zweischichtige Perzeptrone mächtiger sind als einstufige; sie können also mehr Funktionen repräsentieren (und somit auch lernen).
In Abbildung 9 ist ein zweischichtiges Perzeptron dargestellt. Die Neuronen der 1. Ebene berechnen Hyperebenen, die jeweils eine Trennung des Eingaberaums repräsentieren. Diese werden durch das Neuron in der 2. Ebene mit einem logischen AND zum Schnitt gebracht, wodurch ein konvexes Polygon (Vieleck) entsteht.
Das logische AND kann durch $w_{36} = w_{46} = w_{56}$ und dem Schwellenwert $\Theta_6 = 3 \cdot w_{36}$ realisiert werden.

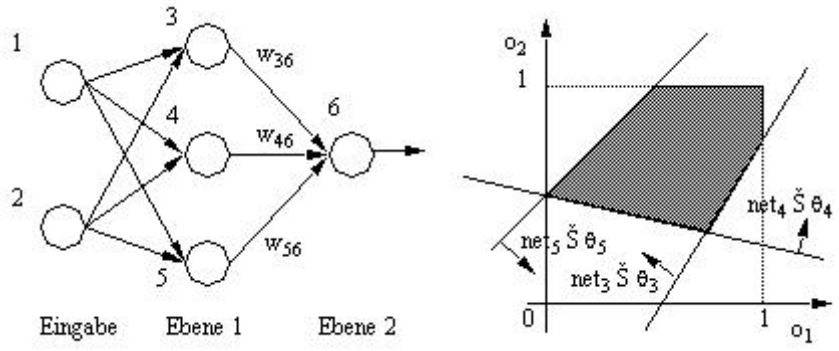

Abbildung 9: Beispiel für ein zweischichtiges Perzeptron [7]

3.5 Dreischichtiges Perzeptron

Dreischichtige binäre Perzeptrone können durch Überlagerung und Schnitt konvexer Polygone Mengen beliebiger Form repräsentieren.

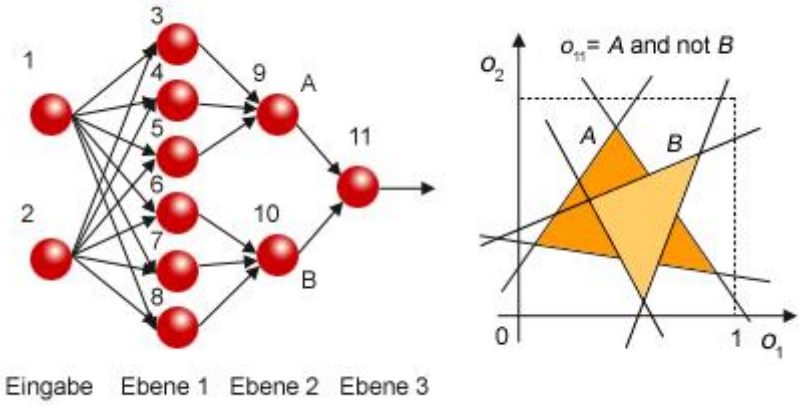

Abbildung 10: Beispiel für ein dreischichtiges Perzeptron [5]

In diesem Beispiel (siehe Abbildung 10) berechnen zwei zweistufige Perzeptrone die Polygone A und B. Diese werden von Neuron 11 in der 3. Ebene durch die Funktion 'A And Not B' zum Schnitt gebracht. Dieses dreischichtige Perzeptron repräsentiert also den dunkelorangen, unzusammenhängenden Teil.

3.6 Die sigmoide Aktivierungsfunktion

Bei dem XOR-Beispiel wurde die binäre Aktivierungsfunktion verwendet. Diese hat den Vorteil, dass sie leicht zu berechnen ist. Ein großer Nachteil ist allerdings, dass sie an der Sprungstelle nicht differenzierbar und die Ableitung überall 0 ist. Dies stellt z. B. beim Backpropagation-Algorithmus ein Problem dar, da die Ableitung in die Berechnung der Gewichtsänderungen miteingeht.

Es wird also eine Funktion benötigt, die sich der binären Schwellwertfunktion beliebig nahe annähert, aber trotzdem differenzierbar ist und deren Ableitung $\neq 0$ ist. Diese Forderung erfüllen die sogenannten sigmoiden Aktivierungsfunktionen, wie z. B.:

$$f(x) = \frac{1}{1 + e^{-\frac{x}{T}}} \tag{1}$$

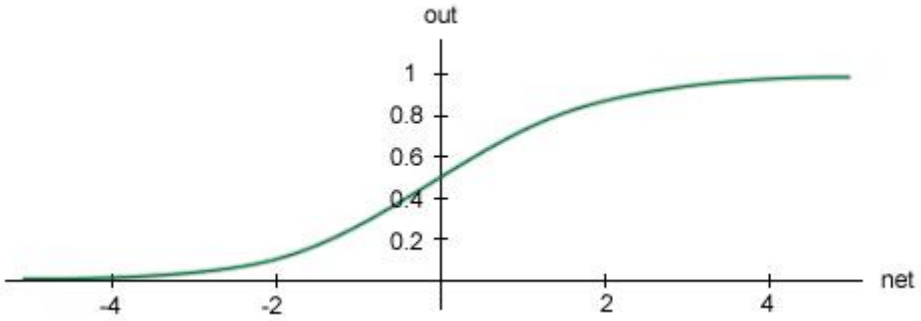

Abbildung 11: sigmoide Aktivierungsfunktion für $T = 1$ [6]

Durch die Wahl der Konstante T kann die sigmoide Aktivierungsfunktion beliebig nahe an die binäre Aktivierungsfunktion angenähert werden.
Später wird für den Backpropagation-Algorithmus die Ableitung der sigmoiden Aktivierungsfunktion benötigt:

$$\begin{aligned} f'(x) &= -\left(\frac{1}{1 + e^{-\frac{x}{T}}}\right)^2 (-\frac{1}{T} e^{-\frac{x}{T}}) = \left(\frac{1}{1 + e^{-\frac{x}{T}}}\right) \left(\frac{\frac{1}{T} e^{-\frac{x}{T}}}{1 + e^{-\frac{x}{T}}}\right) \\ &= \left(\frac{1}{1 + e^{-\frac{x}{T}}}\right) \cdot \frac{1}{T} \left(\frac{1 + e^{-\frac{x}{T}} - 1}{1 + e^{-\frac{x}{T}}}\right) = \frac{1}{T} \cdot f(x) \cdot (1 - f(x)) \end{aligned} \tag{2}$$

(Abschnitt 3 bezieht sich auf [8] und [9])

4 Lernen in Neuronalen Netzen

4.1 Verschiedene Formen des Lernens

Da nun die Fragen der Repräsentierbarkeit Neuronaler Netze geklärt ist, kann nun auf die Frage eingegangen werden, welche Formen des Lernens vorstellbar sind. Möglich sind folgende Formen:

1. Entwicklung neuer Verbindungen
2. Löschen existierender Verbindungen
3. Modifikation der Gewichtung w_{ij} von Verbindungen
4. Modifikation des Schwellenwertes von Neuronen
5. Modifikation der Aktivierungs-, Propagierungs- oder Ausgabefunktion

Ganz allgemein lässt sich feststellen, dass die Methode 3 die am häufigsten verwendete Art des Lernens darstellt. Die meisten Algorithmen verändern die Gewichtung von Verbindungen zwischen Neuronen. Die Alternativen 1 und 2 können ebenfalls dadurch realisiert werden. Die Entwicklung einer neuen Verbindung zwischen zwei Neuronen i und j wird durch das Setzen des Verbindungsgewichts $w_{ij} \neq 0$ gewährleistet. Analog werden existierende Verbindungen zwischen zwei Neuronen i und j gelöscht, indem das Verbindungsgewicht w_{ij} auf 0 gesetzt wird. Auch Methode 4 kann wie die Modifikation von Gewichten behandelt werden. Der letzte Punkt ist biologisch nicht gut motiviert und spielt deswegen bei Lernalgorithmen kaum eine Rolle.

4.2 Hebb'sche Lernregel

Die Hebb'sche Lernregel wurde 1949 von Donald Hebb als möglicher biologischer Lernmechanismus vorgeschlagen. Dabei sollen Neuronen, die gleichzeitig aktiv sind, bevorzugt aufeinander reagieren. D. h. das Gewicht zwischen diesen beiden Neuronen wird erhöht. Neuronen, deren Aktivität nicht korreliert, haben eine schwache bzw. gar keine Verbindung.
Die Hebb'sche Lernregel lautet:

$$\Delta w_{ij} = \eta o_i a_j$$

Dabei bezeichnet Δw_{ij} die Änderung des Gewichts w_{ij}, η die Lernrate, o_i die Ausgabe der Vorgängerzelle i und a_j die Aktivierung der Nachfolgerzelle j.
Werden nur binären Eingaben 0 und 1 zugelassen, sind die Gewichtsänderungen stets positiv (oder 0). Dieses Problem kann umgangen werden, indem die Eingaben -1 und 1 zugelassen werden.

4.3 Delta-Regel

Die Delta-Regel kann nur bei linearer Aktivierungsfunktion für Netze mit einer Schicht trainierbarer Gewichte verwendet werden. Sie ist ein Beispiel für überwachtes Lernen: das Neuronale Netz bekommt nach jeder Berechnung einen sogenannten *teaching input* t_j. Der Fehler, den das Neuronale Netz bei einer Eingabe macht, berechnet sich aus der Differenz des teaching inputs und der erwarteten Ausgabe o_j. Die Gewichtsänderung Δw_{ij} ist proportional zu diesem Fehler. Die Delta-Regel lautet also

$$\Delta w_{ij} = \eta o_i (t_j - o_j) = \eta o_i \delta_j \tag{3}$$

wobei δ_j als Fehlersignal definiert wird.

4.4 Der Backpropagation-Algorithmus

4.4.1 Grundlagen

Da, wie vorher gezeigt, einstufige neuronale Netze nicht so mächtig sind wie mehrstufige, ist es sinnvoll, einen Lernalgorithmus zu bestimmen, der auch für mehrstufige Netze geeignet ist. Ein solcher Algorithmus ist der Backpropagation-Algorithmus, der für Netze mit mehreren Schichten trainierbarer Gewichte geeignet ist.
Der Backpropagation-Algorithmus ist ein Gradientenabstiegsverfahren, welches häufig zur Suche von Minima n-dimensionaler Funktionen verwendet wird. Dabei ist der Gradient ein Vektor der partiellen Ableitungen, welcher in die Richtung des steilsten Anstiegs zeigt. Dieser Vektor wird mit einer negativen Schrittweite (oder auch Lernfaktor) multipliziert und bewegt sich somit Richtung Minimum.

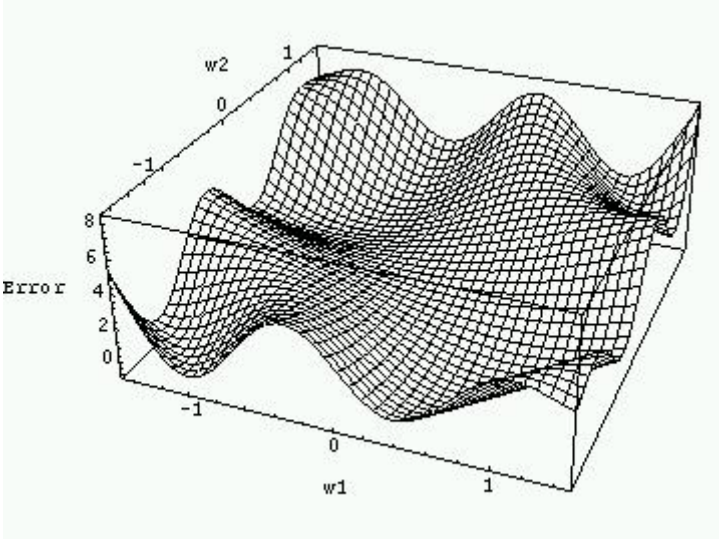

Abbildung 12: Fehlerfläche [7]

In Abbildung 12 ist eine mögliche Fehlerfunktion abgebildet, welche ein Netzwerk, mit den Gewichten $w_1 ... w_n$ über alle Trainingsmuster aufsummiert, besitzt.

Der Backpropagation-Algorithmus geht in vier Schritten vor. Zuerst wird von einem Muster p der Fehler durch *Feedforward*-Berechnung bestimmt. Ausgehend von der Ausgabeschicht wird der Fehler nun zurückpropagiert, die Berechnung der Gewichtsänderungen erfolgt also rückwärts bis zur Eingabeschicht. Danach erfolgt die Korrektur der Gewichte. Dieser Zyklus wiederholt sich so lange, bis der Fehler klein genug ist, und der Algorithmus terminiert.

4.4.2 Herleitung

Im folgenden Abschnitt wird der Backpropagation-Algorithmus hergeleitet.

Die Propagierungsfunktion bestimmt die Netzeingabe einer Zelle j bei einem Muster p:

$$net_{pj} = \sum_i o_{pi}\, w_{ij} \tag{4}$$

Die Netzausgabe ergibt sich durch die Aktivierungsfunktion:

$$o_j = f_{act}(net_{pj}) \tag{5}$$

Der Gesamtfehler E berechnet sich aus den Einzelfehlern E_p, die bei einem bestimmten Muster p entstanden sind:

$$E = \sum_p E_p$$

Die Fehler E_p bei einem Muster p berechnen sich aus dem mittleren quadratischen Fehler (das $\frac{1}{2}$ wird verwendet, da es die spätere Herleitung erleichtert):

$$E_p = \frac{1}{2} \sum_j (t_{pj} - o_{pj})^2 \tag{6}$$

Der Fehler soll minimiert werden, indem jedes Gewicht um einen durch die Lernrate η festgelegten Bruchteil des negativen Gradienten der Fehlerfunktion geändert wird. Bei Ableitung der Fehlerfunktion nach dem zu ändernden Gewicht ergibt sich für die Gewichtsänderung:

$$\Delta w_{ij} = -\eta \, \frac{\partial E_p}{\partial w_{ij}} \tag{7}$$

Dies lässt sich mit Hilfe der Kettenregel umschreiben zu:

$$\begin{aligned} \frac{\partial E_p}{\partial w_{ij}} &= \frac{\partial E_p}{\partial net_{pj}} \cdot \frac{\partial net_{pj}}{\partial w_{ij}} \overset{(4)}{=} \frac{\partial E_p}{\partial net_{pj}} \cdot \frac{\partial}{\partial w_{ij}} \sum_i o_{pi}\, w_{ij} \\ &= \frac{\partial E_p}{\partial net_{pj}} \cdot o_{pi} = -\delta_{pj} \cdot o_{pi} \\ &\quad \text{mit } \delta_{pj} = -\frac{\partial E_p}{\partial net_{pj}} \end{aligned} \tag{8}$$

wobei δ_{pj} als Fehlersignal definiert wird.
Setzt man dies nun in Gleichung (7) ein, ergibt sich:

$$\Delta w_{ij} = -\eta \cdot \frac{\partial E_p}{\partial w_{ij}} = \eta \cdot \sum_p o_{pi}\, \delta_{pj} \tag{9}$$

In dieser Schreibweise sieht man die Gemeinsamkeit von Backpropagation-Regel (9) und Delta-Regel (3). Der Unterschied besteht nur darin, dass die δ_{pj} komplizierter zu berechnen sind.

Diese δ_{pj} werden folgendermaßen berechnet:

$$\begin{aligned}\delta_{pj} &= -\frac{\partial E_p}{\partial net_{pj}} = -\frac{\partial E_p}{\partial o_{pj}} \cdot \frac{\partial o_{pj}}{\partial net_{pj}} \\ &\overset{(5)}{=} -\frac{\partial E_p}{\partial o_{pj}} \cdot \frac{\partial}{\partial net_{pj}} f_{act}(net_{pj}) = -\frac{\partial E_p}{\partial o_{pj}} \cdot f'_{act}(net_{pj})\end{aligned}$$

Nun müssen zwei Fälle unterschieden werden:

1. j ist Index einer Ausgabezelle:

   $\frac{\partial E_p}{\partial o_{pj}}$ kann direkt aus Gleichung (6) berechnet werden, und es ergibt sich:

$$\begin{aligned}\delta_{pj} &= -\frac{\partial(\frac{1}{2}\sum_j (t_{pj} - o_{pj})^2)}{\partial o_{pj}} \cdot f'_{act}(net_{pj}) = -(t_{pj} - o_{pj}) \cdot (-1) \cdot f'_{act}(net_{pj}) \\ &= (t_{pj} - o_{pj}) \cdot f'_{act}(net_{pj})\end{aligned}$$

2. j ist Index einer verdeckten Ebene:

   $\frac{\partial E_p}{\partial o_{pj}}$ kann nicht direkt bestimmt werden. Deswegen wird erneut die Kettenregel angewendet:

$$\begin{aligned}\delta_{pj} &= -\sum_k \left(\frac{\partial E_p}{\partial net_{pk}} \cdot \frac{\partial net_{pk}}{\partial o_{pj}} \right) \cdot f'_{act}(net_{pj}) \\ &\overset{(4)}{=} \sum_k (\delta_{pk} \cdot \frac{\partial}{\partial o_{pj}} \sum_i o_{pi}\, w_{ik}) \cdot f'_{act}(net_{pj}) \\ &\overset{(8)}{=} f'_{act}(net_{pj}) \cdot \sum_k \delta_{pk}\, w_{jk}\end{aligned}$$

Nun muss noch $f'_{act}(net_{pj})$ bestimmt werden:

$$\begin{aligned}f'_{act}(net_{pj}) &\overset{(2)}{=} f_{act}(net_{pj}) \cdot (1 - f_{act}(net_{pj})) \\ &\overset{(5)}{=} o_{pj} \cdot (1 - o_{pj}) \qquad (10)\end{aligned}$$

Die δ_{pj} lassen sich also wie folgt berechnen:

$$\begin{aligned}\delta_{pj} &= \begin{cases} f'_{act}(net_{pj}) \cdot (t_{pj} - o_{pj}), & \text{falls } j \text{ Index einer Ausgabezelle ist} \\ f'_{act}(net_{pj}) \cdot \sum_k \delta_{pk}\, w_{jk}, & \text{falls } j \text{ Index einer verdeckte Ebene ist} \end{cases} \\ &\overset{(10)}{=} \begin{cases} o_{pj}(1 - o_{pj}) \cdot (t_{pj} - o_{pj}), & \text{falls } j \text{ Index einer Ausgabezelle ist} \\ o_{pj}(1 - o_{pj}) \cdot \sum_k \delta_{pk}\, w_{jk}, & \text{falls } j \text{ Index einer verdeckte Ebene ist} \end{cases}\end{aligned}$$

4.4.3 Probleme des Backpropagation-Algorithmus

Natürlich hat der Backpropagation-Algorithmus auch einige Schwachstellen. Es ist möglich, dass der Algorithmus nur ein lokales Minimum der Fehlerfunktion findet und dort terminiert. Ein weiteres Problem ist, dass bei einem Perzeptron mit vielen inneren Schichten die erste Schicht (also die Schicht nahe der Eingabe) eventuell lernunwillig ist. Zwei weitere Probleme sind hier graphisch dargestellt:

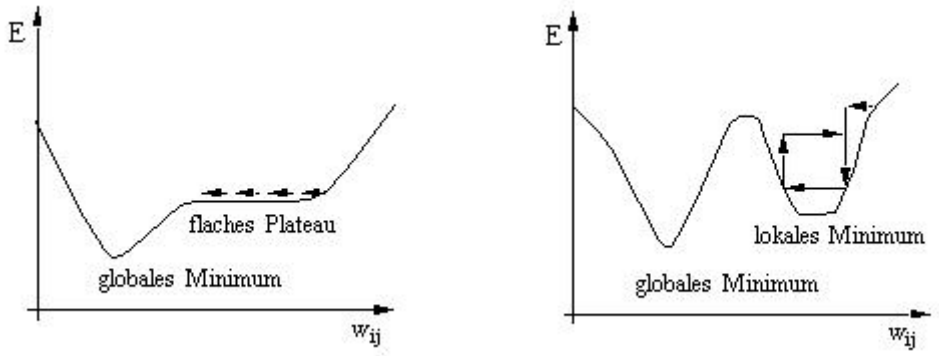

Abbildung 13: Backpropagation an Plateaus und an steilen Schluchten [7]

An weiten Plateaus braucht der Backpropagation-Algorithmus viele Iterationsschritte, um zum Minimum zu gelangen, da die Ableitung ≈ 0 ist. Ein weiteres Problem gibt es bei einer stark zerklüfteten Fehlerfunktion, da der Backpropagation-Algorithmus an steilen Schluchten oszillieren kann.

4.4.4 Unterschied zwischen Online- und Batch-Training

Es gibt grundsätzlich zwei verschiedene Trainingsmethoden für den Backpropagation- Algorithmus: das Online-Training und das Batch- (bzw. Offline-) Training. Beim Online-Training findet die Gewichtsänderung nach jedem Trainingsmuster statt. Die Schreibweise hierfür ist $\Delta_p w_{ij}$. Das Problem bei dieser Trainingsmethode besteht darin, dass eventuell bereits gemachte Gewichtsänderungen im nächsten Schritt (also bei Betrachtung eines anderen Musters) wieder rückgängig gemacht werden.
Beim Batch- oder Offline-Training werden Gewichtsänderungen erst nach der Präsentation aller Trainingsmuster vorgenommen. Der Gesamtfehler E wird dabei aufsummiert $E = \sum_p E_p$. Diese Methode garantiert, dass der Gesamtfehler stets kleiner wird und somit ein (lokales) Minimum gefunden wird. Der Nachteil beim Batch-Training besteht darin, dass sich kleine Gewichtsänderungen oft gegenseitig aufheben.
Für das XOR-Problem ist jedoch das Batch-Training schneller als das Online-Training. Eine generelle Aussage lässt sich allerdings nicht treffen. Die Laufzeit ist von dem jeweiligen Problem abhängig.

4.4.5 Modifikationen

In diesem Abschnitt wird kurz auf einige Modifikationen eingegangen, mit deren Hilfe einige Probleme des Backpropagation-Algorithmus übergangen werden können.

Momentum-Term:
Diese Modifikation ist schneller an weiten Plateaus und in steilen Schluchten der Fehlerfunktion. Dies wird erreicht durch die Einführung eines Terms, der die im letzten Schritt gemachte Änderung der Gewichte berücksichtigt:

$$\Delta_p w_{ij}(t+1) = \eta o_{pi} \delta_{pj} + \alpha \Delta_p w_{ij}(t)$$

wobei $0,2 \leq \alpha \leq 0,99$. Diese Modifikation führt zu einer Beschleunigung in weiten Plateaus und einem Abbremsen in stark zerklüfteten Fehlerflächen.

Flat-Spot Elimination:
Ein Problem ist, dass die Ableitung der Aktivierungsfunktion

$$f'_{act}(net_{pj}) = o_{pj}(1 - o_{pj})$$

in δ_{pj} mit eingeht. Für Neuronen, die Werte um 0 oder 1 annehmen, ist $f'_{act} \approx 0$, was dazu führt, dass die Gewichte nur geringfügig geändert werden. Deswegen wird bei der Flat-Spot Elimination $0,1$ zu $f'_{act}(net_{pj})$ hinzuaddiert.

SuperSAB:
Die Idee bei der Modifikation SuperSAB ist die Einführung einer eigenen Schrittweite $\eta_{ij}(t)$ für jedes Gewicht. Dabei wird die Schrittweite $\eta_{ij}(t)$ eines Gewichtes erhöht, wenn das Vorzeichen der partiellen Ableitung $\frac{\partial E_p}{\partial w_{ij}}$ gleich bleibt, ansonsten verkleinert. Dies führt dazu, dass die Schrittweite an weiten Plateaus vergrößert und an steilen Schluchten verkleinert wird.

4.5 Quickprop-Algorithmus

Der Quickprop-Algorithmus ist eine weitere Modifikation des Backpropagation-Algorithmus und zeigt für viele Probleme ein deutlich schnelleres Training.

4.5.1 Grundlagen

Der Quickprop-Algorithmus geht von zwei Annahmen aus:

- die Fehlerfunktion kann durch nach oben geöffnete Parabel dargestellt werden
- Δw_{ij} ist unabhängig von den Änderungen der anderen Gewichte

Das Verfahren ähnelt dem Newton-Verfahren. Es wird versucht, anhand der Steigung der Parabel an zwei Punkten und deren Abstand in einem Schritt zum Minimum zu springen:

$$\Delta w_{ij}(t) = \frac{\frac{\partial E_p}{\partial w_{ij}}(t)}{\frac{\partial E_p}{\partial w_{ij}}(t-1) - \frac{\partial E_p}{\partial w_{ij}}(t)} \cdot \Delta w_{ij}(t-1) \tag{11}$$

wobei $\frac{\partial E_p}{\partial w_{ij}}(t)$ die Ableitung der Fehlerfunktion nach dem Gewicht w_{ij} zum aktuellen Zeitpunkt, $\frac{\partial E_p}{\partial w_{ij}}(t-1)$ die Ableitung der Fehlerfunktion nach dem Gewicht w_{ij} zum vorhergehenden Zeitpunkt und $\Delta w_{ij}(t-1)$ die Änderung des Gewichts zum vorhergehenden Zeitpunkt ist.
Zur Vereinfachung wird auch $S(t)$ statt $\frac{\partial E_p}{\partial w_{ij}}(t)$ verwendet.

4.5.2 Herleitung

Die Herleitung des Quickprop-Algorithmus ist relativ einfach.

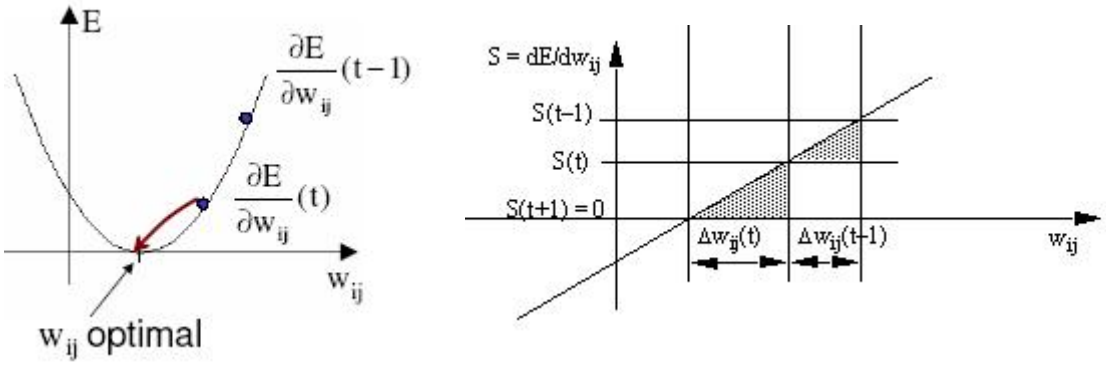

Abbildung 14: Verdeutlichung des Quickprop-Algorithmus [7]

In Abbildung 14 wird das Verfahren verdeutlicht. Außerdem dient die Abbildung für die Herleitung der Gleichung (11). Wegen der Ähnlichkeit der schraffierten Dreiecke gilt:

$$\begin{aligned}
\frac{\Delta w_{ij}(t)}{\Delta w_{ij}(t-1)} &= \frac{S(t)}{S(t-1)-S(t)} \\
\Leftrightarrow \Delta w_{ij}(t) &= \frac{S(t)}{S(t-1)-S(t)} \cdot \Delta w_{ij}(t-1) \\
&= \frac{\frac{\partial E_p}{\partial w_{ij}}(t)}{\frac{\partial E_p}{\partial w_{ij}}(t-1)-\frac{\partial E_p}{\partial w_{ij}}(t)} \cdot \Delta w_{ij}(t-1)
\end{aligned}$$

4.5.3 Verschiedene Fälle

Bei Betrachtung dieser Gleichung fällt auf, dass verschiedene Fälle unterschieden werden müssen:

1. $S(t) < S(t-1)$ und $\text{sgn}(S(t)) = \text{sgn}(S(t-1))$
   In diesem Fall erfolgt eine Gewichtsänderung in die selbe Richtung wie zuvor:

2. $S(t) > S(t-1)$ und $\text{sgn}(S(t)) = \text{sgn}(S(t-1))$
   In diesem Fall sucht der Algorithmus den Scheitel einer nach unten geöffneten Parabel und bewegt sich in Richtung eines (lokalen) Maximums, also weg vom Minimum.

3. $S(t) = S(t-1)$
   Dieser Fall führt zu einer Division durch 0, was entweder zu einem unendlich großen Schritt oder dem Abbruch des Programms führt. Deswegen wird ein maximaler Wachstumsfaktor μ mit

   $$\Delta w_{ij}(t) \leq \mu \cdot \Delta w_{ij}(t-1)$$

   eingeführt. Dies garantiert, dass die Gewichtsänderung einen bestimmten Wert nie überschreiten kann.

4. $\text{sgn}(S(t)) \neq \text{sgn}(S(t-1))$
   Sind die Vorzeichen der Ableitungen nicht gleich, wurde im vorherigen Schritt das Minimum übersprungen. In diesem Fall wird eine Position zwischen den beiden Vorherigen gewählt.

Die genaue Formel wird hier nicht angegeben, sie kann aber in [9] nachgelesen werden.
Sind die zwei oben genannten Annahmen für ein bestimmtes Netz erfüllt, ist der Quickprop-Algorithmus ein sehr schneller Lernalgorithmus.
(Abschnitt 4 bezieht sich auf [9])

5 Zusammenfassung

Der erste Teil der Ausarbeitung diente dazu, die wichtigsten Grundlagen Neuronaler Netze zu erklären. Dabei wurde geklärt, wie man die Anzahl der Schichten eines Neuronalen Netzes bestimmt und wofür ein On-Neuron nützlich ist. Außerdem wurde auf den Unterschied zwischen Repräsentierbarkeit und Lernfähigkeit eines Neuronalen Netzes eingegangen. Ausgehend vom XOR-Problem wurde beschrieben, warum mehrschichtige Perzeptrone notwendig sind. Im zweiten Teil wurden die wichtigsten Lernalgorithmen, wie die Hebb'sche Lernregel, die Delta-Regel und der Backpropagation-Algorithmus, erläutert. Dem Backpropagation-Algorithmus kam dabei die wichtigste Bedeutung zu, da dieser einen Lernalgorithmus für mehrschichtige Perzeptrone darstellt. Zusätzlich wurden noch einige Modifikationen dieses Algorithmus besprochen, welche Probleme, die beim Backpropagation-Algorithmus auftreten können, verhindern.

Literatur

[1] http://de.wikipedia.org/wiki/Nervenzelle

[2] http://en.wikipedia.org/wiki/Image:SynapseIllustration.png

[3] http://www.chemgapedia.de/vsengine/media/vsc/de/ch/13/anc/daten/neuronalenetze/images/snn_107.gif

[4] http://www.chemgapedia.de/vsengine/media/vsc/de/ch/13/anc/daten/neuronalenetze/images/snn_113.gif

[5] http://www.chemgapedia.de/vsengine/media/vsc/de/ch/13/anc/daten/neuronalenetze/images/snn_116.gif

[6] http://www.iicm.edu/greif/images/img410.gif

[7] http://www-ra.informatik.uni-tuebingen.de/virtugrade/book/

[8] Raul Rojas, Theorie der Neuronalen Netze, Springer-Verlag, 1993

[9] Andreas Zell, Simulation Neuronaler Netze, Oldenbourg, 1994