

Universität zu Köln
Institut für Informatik
Direktor: Prof. Dr. Ewald Speckenmeyer
Dozent: Dr. Bert Randerath

Seminararbeit im Rahmen der Vorlesung
„Aussagenlogikbasierte Suchprobleme“
im Wintersemester 2001/02

Thema der Arbeit:
„Neuronale Netze“

Vorgelegt von:
Holz, Patrick

Jüchen, den 26. 3. 2002

Inhaltsverzeichnis

1. Einleitung	Seite 3 - 4
1.1. Motivation	Seite 3
1.2. Was sind Neuronale Netze?	Seite 3
1.3. Vorteile Neuronaler Netze	Seite 3 - 4
1.4. Vergleich zu Vorbildern aus der Biologie	Seite 4
2. Modellierung und Arbeitsweise	Seite 4 - 7
2.1. Modellierung Neuronaler Netze	Seite 4 - 5
2.2. Netzstrukturen	Seite 5 - 6
2.3. Neuronen-Verhalten	Seite 6 - 7
2.3.1. <i>Relevante Größen</i>	<i>Seite 6</i>
2.3.2. <i>Eigenschaften von Aktivierungsfunktionen</i>	<i>Seite 6</i>
2.3.3. <i>Perzeptrons</i>	<i>Seite 7</i>
3. Möglichkeiten der Anwendung	Seite 8 - 11
3.1. Anwendungsbeispiele	Seite 8 - 10
3.1.1. <i>Ein einfaches Beispiel: Das XOR-Problem</i>	<i>Seite 8 - 9</i>
3.1.2. <i>Funktionsapproximation</i>	<i>Seite 9</i>
3.1.3. <i>Klassifikation</i>	<i>Seite 9 - 10</i>
3.2. Berechenbarkeit Boole'scher Funktionen	Seite 10 - 11
4. Lernverfahren mit Neuronalen Netzen	Seite 11 - 14
4.1. Merkmale von Lernverfahren	Seite 11 - 12
4.2. Backpropagation-Lernen	Seite 12 - 14
5. Literaturverzeichnis	Seite 15

1. Einleitung

1.1. Motivation

Etwa seit 1985 erfreut sich das Thema „Neuronale Netze“ im Rahmen der Erforschung künstlicher Intelligenz wieder großer Beliebtheit. Viele Forschungen richten seit dieser Zeit ihr Interesse auf diesen Bereich der Informatik, der versucht, die Eigenschaften des (menschlichen) Gehirns in Bezug auf die Fähigkeiten zur Mustererkennung und zum Lernen durch Parallelverarbeitung nachzubilden. In dieser Arbeit werde ich versuchen, die Komponenten und den Aufbau Neuronaler Netze zu erläutern, die Fähigkeiten und Möglichkeiten Neuronaler Netze zu analysieren und einen kleinen Ausschnitt aus den Anwendungsmöglichkeiten exemplarisch darzustellen. Grundlage hierfür bildete ein Skript von Alois Heinz von der Universität Freiburg. Darüber hinaus habe ich weitere Literatur verwendet, die ich im angefügten Literaturverzeichnis angeben werde.

1.2. Was sind Neuronale Netze?

Neuronale Netze sind im Wesentlichen modular aufgebaute Rechenmodelle, die dem Nervengeflecht des menschlichen Gehirns nachempfunden sind. Ihre Modularität gewinnen sie vor allem durch die Einteilung in atomare informationsverarbeitende Elemente, die „Neuronen“. Je nach Aufgabe und Funktionsweise unterscheidet man Inputneuronen (enthalten konstante Werte oder empfangen Werte aus der Systemumwelt), Berechnungsneuronen (erhalten Werte von anderen Neuronen und verarbeiten diese) und Outputneuronen (geben die Ergebnisse aus dem Neuronalen Netz an die Systemumwelt). Als Verbindungen untereinander und zur Systemumwelt dienen ihnen dabei die „Synapsen“. Diese sind grundsätzlich gerichtet (führen eindeutig von einem Neuron A zu einem Neuron B) und gewichtet (transportierte Informationen werden mit einem Gewichtungsfaktor versehen). Neuronen und Synapsen sind die einzigen Bestandteile Neuronaler Netze.

1.3.Vorteile Neuronaler Netze

Die Reduktion auf lediglich zwei Bestandteile scheint die Funktionalität Neuronaler Netze zunächst stark einzuschränken. Gerade diese starke Modularisierung ist es aber, die die Verwendung Neuronaler Netze so nützlich macht. Aufgrund des modularen Aufbaus arbeiten

Neuronale Netze inhärent parallel, sodass die Implementierung insbesondere auf Parallelrechnern sehr leicht realisiert werden kann und sehr gute Laufzeitergebnisse erzielt werden können.

In sicherheitssensitiven Anwendungsbereichen kann durch eingebaute Redundanz ein hohes Maß an Fehlertoleranz und Ausfallsicherheit erzielt werden, ohne dass die Laufzeit allzu sehr darunter leiden würde. Lediglich die benötigten Ressourcen (Speicherplatz bzw. Neuronenzahl) steigen an.

Durch gezieltes Training kann ein Neuronales Netz zur automatisierten Lösung bestimmter Problemfelder „erzogen“ werden. Der hier erzielte Lerneffekt kann mit speziellen Algorithmen erzielt werden, mit denen ich mich in dieser Arbeit vorrangig in Kapitel 4 beschäftige.

1.4. Vergleich zu Vorbildern aus der Biologie

Wie bereits erwähnt versuchen Neuronale Netze, das Nerven-Netz im menschlichen Gehirn nachzubilden. Das menschliche Gehirn besteht aus ca. 10-100 Milliarden Neuronen, die jeweils eine Größe von 5-100 Mikrometern haben können. Sie sind über etwa 100-1000 Billionen (eine 1 mit 14-15 Nullen) Synapsen untereinander verbunden.

Die Kommunikation zwischen den Neuronen im Gehirn läuft über so genannte „spike trains“, das sind exakt festgelegte zeitliche Signalrhythmen, ab. Man spricht in der Biologie auch von der „Gehirnmusik“. Dazu kommen diverse chemische Reaktionen. Bei künstlichen neuronalen Netzen versucht man dies mit Hilfe von Bitmustern und physikalischen Phänomenen (elektrischer Strom) nachzubilden. Die „Taktfrequenz“ des Gehirns beträgt im übrigen ca. 1 KHz, ist im Vergleich zu heutigen Prozessoren also eigentlich sehr gering.

Das Gehirn erreicht seine erstaunlichen Fähigkeiten also nicht über eine schnelle sequentielle Verarbeitung, sondern über eine massiv parallele. Natürlich spielt die bereits angesprochene Fehlertoleranz und Ausfallsicherheit im Gehirn eine wesentliche Rolle.

2. Modellierung und Arbeitsweise

2.1. Modellierung Neuronaler Netze

Neuronale Netze lassen sich aufgrund der Eigenschaften ihrer Synapsen gut als gerichteter und gewichteter Graph darstellen. Dabei werden die Neuronen als Knoten interpretiert, die Synapsen als Kanten. Interessant zu beobachten ist dabei die sich ergebende Struktur (Topologie) des Netzes. Hierzu gleich mehr (Abschnitt 2.2.).

Darüber hinaus müssen auch geeignete Formen zur Modellierung des Neuronenverhaltens gefunden werden (Abschnitt 2.3.), sowie zur Darstellung der Schnittstellen zur Außenwelt. Schließlich müssen möglichst effektive Algorithmen für die Verwendung Neuronaler Netze, insbesondere im Hinblick auf den Lernaspekt, entwickelt werden (Abschnitt 4).

2.2. Netzstrukturen

Beliebige Netzstrukturen, in denen auch Zyklen möglich sind, nennt man auch „rekurrente Netze“. Sie sollen hier nicht näher betrachtet werden.

Interessant sind dagegen die 1982 von Hopfield entwickelten „symmetrischen Netze“, auch „Hopfield-Netze“ genannt (Abbildung 1a). Sie enthalten jede Verbindung mit gleichem Gewicht auch in umgekehrter Richtung und eignen sich zum Beispiel zur Lösung schwieriger Optimierungsprobleme.

Im Folgenden beschäftige ich mich vorrangig mit sogenannten „vorwärtsberechnenden Netzen“ (Abbildung 1b). Sie zeichnen sich dadurch aus, dass sie kreisfrei sind und ihre Neuronen in der Reihenfolge der Berechnung durchnummeriert werden können (topologische Sortierung). An dieser Stelle exemplarisch ein Wort zur Laufzeit: Bei vorwärtsberechnenden Netzen ist sehr leicht einzusehen, dass ihre Worst-Case-Laufzeit linear in der Anzahl der beteiligten Neuronen beschränkt ist, da jedes Neuron nur einmal eine Berechnung durchführt. Eine besondere Ausprägungsform vorwärtsberechnender Netze sind die „geschichteten Netze“ (Abbildung 1c). Bei dieser Netztopologie können die Neuronen in Schichten eingeteilt werden, so dass keinerlei Synapsen innerhalb der Schichten verlaufen, sondern nur zwischen den Schichten. Auch hier ist die Laufzeit (im Hinblick auf Parallelberechnung) einleuchtend, sie ist nämlich linear in der Anzahl der Schichten.

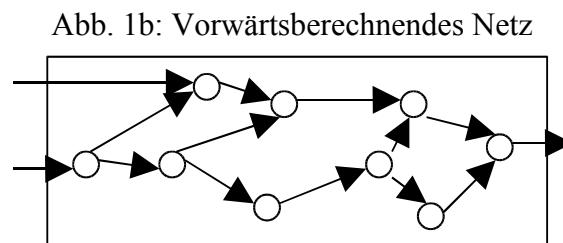
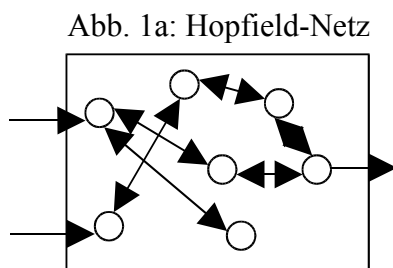
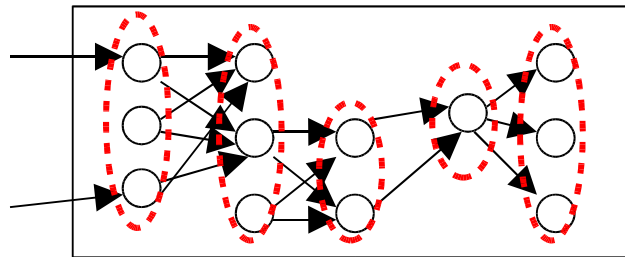


Abb. 1c: geschichtetes Netz



2.3. Neuronen-Verhalten

2.3.1. Relevante Größen

An den Netzstrukturen erkennt man, dass im Normalfall ein Neuron j von n anderen Neuronen Eingabewerte $\{x_{j1}, \dots, x_{jn}\}$ erhält, die jeweils mit Gewichten $\{w_{j1}, \dots, w_{jn}\}$ versehen sind. Diese Parameter lassen sich zu Vektoren x_j bzw. w_j zusammenfassen.

Des Weiteren befindet sich jedes Neuron in einem aktuellen Zustand s_j . Dies kann ein Ausgangszustand sein oder ein bereits durch andere Neuroneninputs beeinflusster Zustand.

Bildet man nun eine Funktion $h_j(x_j, w_j, s_j)$, die sogenannte „Aktivierungsfunktion“, so lässt sich durch die Analyse dieser Funktion das Verhalten des jeweiligen Neurons analysieren.

2.3.2. Eigenschaften von Aktivierungsfunktionen

Aktivierungsfunktionen lassen sich anhand diverser Eigenschaften charakterisieren:

- Linearität: Aktivierungsfunktionen können linear, semilinear (monoton steigend und differenzierbar) oder nicht-linear sein.
- Determinierung: Deterministische Aktivierungsfunktionen liefern bei gleichem Input stets das gleiche Ergebnis, wogegen stochastische Funktionen verschiedene Ergebnisse mit einer jeweils festgelegten Wahrscheinlichkeit zurückgeben.
- Abstufung: Die Funktion kann stetig (kontinuierlich) sein oder sie kann nur diskrete Werte annehmen.
- Des Weiteren kann die Abhängigkeit von der Variablen s_j entfallen, wie es der Einfachheit halber in den folgenden Beispielen auch angenommen wird.

2.3.3. Perzeptrons

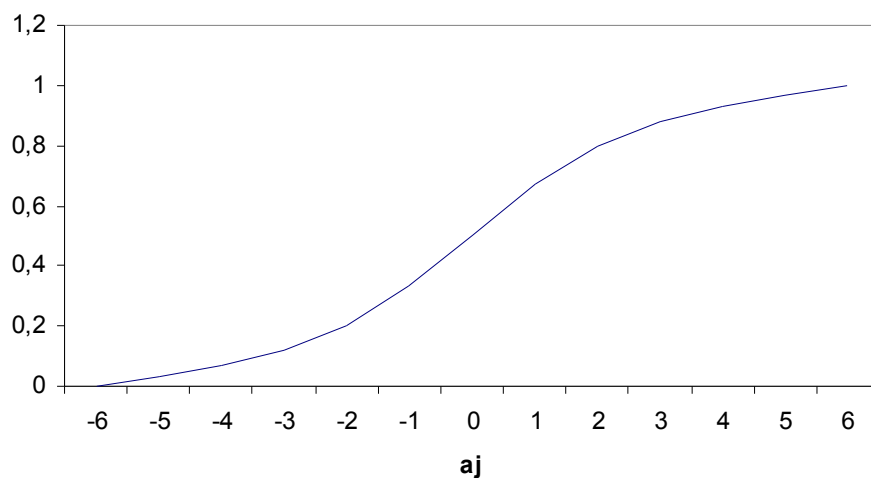
Neuronen, die eine bestimmte Aktivierungsfunktion besitzen, die sogenannte „logistische Sigmoide“, nennt man „Perzeptrons“. Eine Sigmoidfunktion zeichnet sich durch einen S-förmigen Verlauf aus, die logistische Sigmoidfunktion folgt der Gleichung

$$\sigma(a) = \frac{1}{1 + \exp(-a)}$$

Nun ist der Parameter a geeignet zu wählen:

$$a = \gamma * a_j = \gamma * w_j^T x_j = \gamma * \sum_{i \in V(j)} w_{ji} x_{ji}$$

Hierbei bezeichnet γ einen Multiplikator zur Einstellung der gewünschten Intensität von a_j und $V(j)$ bezeichnet die Menge der Vorgänger von j . Graphisch sieht die Funktion $\sigma(a)$ für $\gamma=1$ wie folgt aus:



Wird die Funktion nun durch Erhöhung von γ weiter gestaucht, so erhält man für den Grenzwert $\gamma \rightarrow \infty$ die sogenannte „Heaviside“- oder „Sprung“-Funktion $\Theta(a_j)$:

$$\Theta(a_j) = \begin{cases} 0, & \text{falls } a < 0 \\ 1, & \text{falls } a > 0 \end{cases}$$

Perzeptrons, die die Heaviside-Funktion als Aktivierungsfunktion besitzen, nennt man auch „klassische Perzeptrons“. Sie funktionieren wie ein 0-1-Schalter, d.h. sie kennen nur zwei Werte, nämlich 0 und 1 bzw. „an“ und „aus“. Liegt der Input unter dem neuroneneigenen Schwellenwert, so wird 0 zurückgegeben, ansonsten 1.

3. Möglichkeiten der Anwendung

3.1. Anwendungsbeispiele

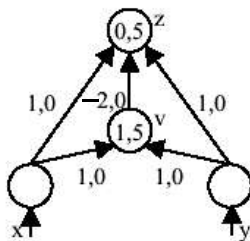
3.1.1. Ein einfaches Beispiel: Das XOR-Problem

Das XOR-Problem stellt ein relativ leicht verständliches Beispiel für die Lösung eines Boole'schen Funktionsproblems mittels Neuronaler Netze dar. Ausgangspunkt bildet die folgende Boole'sche Formel („XOR“-Formel) mit nebenstehender Input-Output-Tabelle (Inputs x und y; Output z).

$$z(x, y) = (\neg x \wedge y) \vee (x \wedge \neg y)$$

<u>x</u>	<u>y</u>	<u>z</u>
1	1	0
0	0	0
1	0	1
0	1	1

Dieser Sachverhalt soll nun mit einem Neuronalem Netz modelliert werden. Benötigt werden zwei Inputneuronen, die die Eingangswerte für x und y aus der Systemumgebung erhalten, sowie ein Outputneuron, das den Wert für z ausgibt. Des weiteren wird ein Berechnungsneuron benötigt, um die vorgegebene Formel umsetzen zu können. Es wird angenommen, dass es sich bei den Neuronen um klassische Perzeptrons handelt. Das Netz sieht dann so aus:



Bemerkung: Die Zahlen im Kreis stellen die Schwellenwerte dar, die Zahlen an den Synapsen die Gewichtung.

Was geschieht nun bei den unterschiedlichen Möglichkeiten der Eingabe?

Angenommen, x und y seien beide 0. An das Neuron v wird dementsprechend der Wert $0 \cdot 1,0 + 0 \cdot 1,0 = 0$ übermittelt. Dieser Wert liegt unter dem Schwellenwert von Neuron v (1,5) und somit gibt Neuron v, ebenso wie x und y, eine 0 an Neuron z weiter. Dieses erhält nun den Wert $0 \cdot 1,0 + 0 \cdot 1,0 + 0 \cdot (-2,0) = 0$ und gibt damit eine 0 aus.

Angenommen, x sei 1 und y sei 0. Dann erhält v den Wert $1 \cdot 1,0 + 0 \cdot 1,0 = 1$ und gibt eine 0 weiter. z erhält $1 \cdot 1,0 + 0 \cdot 1,0 + 0 \cdot (-2,0) = 1$. Dadurch wird z aktiv und gibt eine 1 aus. Der Fall $x=0$ und $y=1$ verläuft analog.

Bleibt noch die Möglichkeit, dass sowohl x als auch y gleich 1 ist. An v wird $1 * 1,0 + 1 * 1,0 = 2$ übermittelt, v wird also aktiv und sendet eine 1 an z . z bekommt den Input $1 * 1,0 + 1 * 1,0 + 1 * (-2,0) = 0$. Somit wird z nicht aktiv und der Output ist 0.

3.1.2. Funktionsapproximation

Neuronale Netze können verwendet werden, um Ergebnisse von Funktionen bei bestimmten Variablen-Konstellationen zu antizipieren. Ein typisches Praxisbeispiel für diese Approximation ist die Zeitreihenanalyse, die beispielsweise in der Spieltheorie, der Industrieproduktion, auf den Finanzmärkten oder auch in der Meteorologie ihre Anwendung findet.

Als gegeben werden dabei verschiedene Systemzustände mit entsprechenden Ergebnissen aus der Vergangenheit (Erfahrungswerte) vorausgesetzt. Gesucht sind nun die optimalen Stellgrößen (die optimale Konstellation) zur Erreichung eines bestimmten (erwünschten) Systemzustandes.

Die Menge an Konstellationen kann dabei sehr schnell sehr zahlreich werden, sodass eine geeignete Modellierung unter Umständen kompliziert und unübersichtlich aussehen kann. Der Vorteil bei der Verwendung Neuronaler Netze liegt nun insbesondere darin, dass die große Menge von Varianten relativ leicht modelliert werden kann, da durch die extreme Modularisierung nur an den relevanten Stellen eine Modifikation durchgeführt werden muss. Auch das Testen und Überprüfen der Varianten wird durch die transparente Rechenweise zum Teil stark vereinfacht.

3.1.3. Klassifikation

Die Klassifikation ist ein weiteres Anwendungsfeld für Neuronale Netze. Die Aufgabe besteht darin, erhaltene Eingabewerte vorgegebenen Klassen zuzuordnen.

Das Vorgehen entspricht dem Finden einer maximalen a-posteriori-Wahrscheinlichkeit $P(C_k | x)$ dafür, dass es sich beim Auftreten des Merkmals x um ein Objekt der Klasse C_k handelt. Die konkrete Berechnung kann über den Satz von Bayes erfolgen, soll hier aber nicht weiter thematisiert werden. Stattdessen möchte ich ein konkretes Beispiel anbringen, und zwar die Zuordnung von Geräuschen durch Hörgerät-Software.

Angenommen, das zu untersuchende Geräusch weist die Merkmale „hohe Lautstärke“, „geringe Tonhöhe“ und „Doppler-Effekt“ auf.

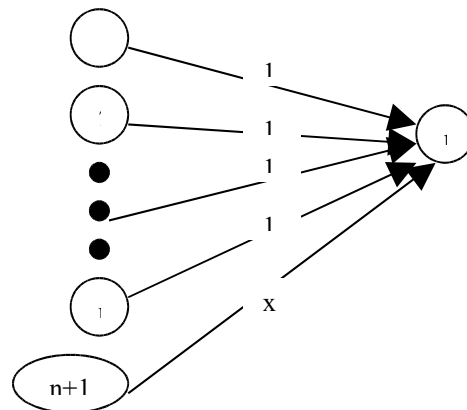
Der Doppler-Effekt weist mit nahezu sicherer Wahrscheinlichkeit auf ein vorbeiziehendes Objekt hin. Die hohe Lautstärke legt nahe, dass sich das Objekt in unmittelbarer Nähe be-

findet. Problematisch kann das Merkmal „geringe Tonhöhe“ sein, denn hier gibt es sehr viele (und vermutlich recht gleichwahrscheinliche) Möglichkeiten, z.B. ein Motorengeräusch, ein dumpfes Klopfen oder ein Mensch mit tiefer Stimme.

So schwierig die Einordnung auch erscheinen mag, dank Neuronaler Netze wird in der modernen Forschung bereits eine Erkennungsrate von bis zu 96% erreicht. Eine entscheidende Bedeutung erhält hier natürlich das Prinzip des Lernens, denn nur so kann die Software neue Geräusche richtig einschätzen und weitergeben.

3.2. Berechenbarkeit Boole'scher Funktionen

Das XOR-Beispiel hat bereits gezeigt, dass Boole'sche Operatoren mit Neuronalen Netzen prinzipiell berechenbar sind. Explizit möchte ich nun noch die Berechenbarkeit der Operatoren OR und AND zeigen. Die Implementierung erfolgt in allen drei Fällen über den Gewichtsvektor w . Vorausgesetzt sei ein klassisches Perzeptron p mit n Vorgängerperzeptrons. Hinzugezogen wird ein fiktives Neuron $n+1$, dessen Synapse zu p mit einem operatorenspezifischen Gewicht versehen wird, während die Gewichte der anderen Synapsen zu p jeweils gleich 1 gesetzt werden. Zur Veranschaulichung folgt nun eine grafische Darstellung des Sachverhaltes:



Bemerkung: p besitzt den Schwellenwert 0.

Die Frage ist nun, wie x für die einzelnen Operatoren zu bestimmen ist.

Für OR muss x gerade so groß sein, dass ein aktiviertes Vorgängerperzeptron genügt, um auch p zu aktivieren. Somit gilt $|x| < 1$. Da x aber auch dafür sorgen muss, dass p inaktiv bleibt, sofern die Neuronen 1 bis n inaktiv sind, muss x negativ sein. Demnach gilt: $-1 < x < 0$

Für AND muss x das Perzeptron p deaktivieren, sofern mindestens ein Neuron von 1 bis n inaktiv ist. Somit gilt hier $-n < x < 1-n$

Beispielhaft kann man sagen, x muss für „OR“ $-0,5$ betragen, für „AND“ $0,5-n$.

Mit Hilfe dieser Vorüberlegungen lässt sich folgender Satz nachweisen:

„Boolesche Funktionen sind durch zweischichtige Neuronale Netze berechenbar.“

Der Beweis des Satzes folgt aus der Konstruktionsweise der Darstellungsformen DNF und CNF, die eine Konjunktion von Disjunktionen (oder umgekehrt) darstellen. Daher sind lediglich die beiden o.g. Operatoren je einmal vonnöten, die jeweils eine Perzeptronschicht benötigen. Negationen werden durch Umkehrung der Gewichte erzeugt, so dass insgesamt nur zwei Schichten benötigt werden.

4. Lernverfahren mit neuronalen Netzen

4.1. Merkmale von Lernverfahren

Wie das Beispiel zur Klassifikation (3.1.3.) zeigt, sind Lernverfahren für die sinnvolle Verwendung Neuronaler Netze essentiell. Nur mit Hilfe von Lernalgorithmen können Neuronale Netze neue Informationen richtig bewerten und sich dementsprechend anpassen. Und dies ist eine der größten Leistungen, die das Vorbild der Neuronalen Netze, das Gehirn, vollbringen kann. Lernverfahren lassen sich nach folgenden Gesichtspunkten charakterisieren:

- Informationsgewinnung: Die Information, was und wie gelernt werden soll, kann das Netz entweder durch Benutzeranweisung erhalten, aus der Rückmeldung des Benutzers schließen („Orakel“-Form) oder völlig autark von der Umgebung selbst aus bekannten Informationen folgern.
- Wissensanpassung: Die Informationen, auf die sich das Netz beim Lernen beziehen kann, können entweder gegeben und unveränderbar (statisch) sein oder sie passen sich den durch die bisherigen Lernvorgänge gewonnenen Zusatzinformationen an (dynamisch).
- Informationsauswahl: Bei der passiven Form der Informationsauswahl nehmen die Netze die zur Verfügung gestellten Informationen so auf, wie sie ihnen angeboten werden, während sie bei der aktiven Informationsauswahl selbst versuchen, die relevanten Information aus dem Angebot herauszufiltern.
- Verhaltensanpassung: Das Netz kann sich aufgrund neu gewonnener Informationen oder veränderter Situationen einerseits durch Änderung seiner Topologie anpassen (Strukturänderung), indem etwa Komponenten (Neuronen, Synapsen) hinzugefügt oder entfernt werden, andererseits können aber auch die eingesetzten Parameter, wie z.B. die Synapsengewichtung, angepasst werden (Parameteränderung).

Im folgenden Abschnitt (4.2.) wird ein Beispiel für ein (benutzer-)angewiesenes, statisches und passives Netz mit Parameteränderung vorgestellt.

4.2. Backpropagation-Lernen

Backpropagation ist das in der Praxis wohl am häufigsten vorkommende Lernverfahren und soll daher hier als Beispiel für eine Anwendung Neuronaler Netze im Bereich Lernen dienen. Gegeben sei ein vorwärtsgerichtetes Schichtenetz, eine Eingabemenge und eine zu lernende Funktion $f(x)$, die aus später ersichtlichen Gründen differenzierbar sein muss. Zunächst sei hier eine grob formulierte Pseudo-Code-Version des Backpropagation-Algorithmus gegeben:

PROCEDURE Backpropagation

 REPEAT

 FOR $i:=1$ TO AnzahlMuster DO

 Erstelle Eingabemuster

 FOR $j:=1$ TO Neuronenzahl DO

 Bestimme Fehlergröße des jeweiligen Neurons

 Addiere Fehler zu der Gesamtfehlergröße

 FOR $s:=$ Outputschicht DOWNTO Inputschicht DO

 Korrigiere sukzessive die Synapsengewichte

 UNTIL Fehler tolerierbar

END

Der Algorithmus lässt sich in drei Schritte unterteilen:

Im ersten Schritt werden bestimmte Werte aus der Eingabemenge zur Berechnung ausgewählt und durchlaufen das Netz. Die Schichten erhalten als Eingabewerte jeweils die Ausgabewerte ihrer Vorgängerschicht, bis schließlich die Ausgabeschicht ein Ergebnis der Berechnung liefert.

Im zweiten Schritt wird nun überprüft, ob das Ergebnis innerhalb einer vorher festzulegenden Fehlertoleranz-Grenze liegt oder nicht. Ist dies der Fall, so bricht der Algorithmus ab und gibt das Ergebnis sowie die gelernte Funktion aus. Ist das Ergebnis „zu falsch“, so wird mit Schritt 3 fortgefahren.

Im dritten Schritt werden die Gewichte des Neuronales Netzes ausgehend von der Ausgangschicht sukzessive verändert, wobei man einem Gradientenabstiegsverfahren (einer Form des Newton-Verfahrens) folgt. Das heißt nun Folgendes:

Der Fehler der Ausgabe an Neuron j wird als Funktion E der Synapsengewichte von j und seinen direkten Vorgängern gesehen:

$$E(W_j) = E(w_{1j}, w_{2j}, \dots, w_{nj})$$

Diese Fehlerfunktion versucht man nun zu minimieren. Da es sich zumeist um nichtlineare Aktivierungsfunktionen handelt, verwendet man das oben bereits erwähnte Gradientenabstiegsverfahren. Die Formel für die Änderung des Synapsengewichtes von Neuron i zu Neuron j lautet:

$$\Delta w_{ij} = -\eta * \frac{\partial E}{\partial o_j} * \frac{\partial o_j}{\partial e_j} * \frac{\partial e_j}{\partial w_{ij}}$$

Die Formel besteht aus dem so genannten „Lernfaktor“ η sowie drei Ableitungen. Im ersten Faktor wird die Fehlerfunktion E nach dem optimalen Wert für j abgeleitet. Eine Darstellungsform von E wäre zum Beispiel diese:

$$E = \frac{1}{2} * \sum_k (t_k - o_k)^2$$

Dabei bezeichnet t_j den tatsächlichen Ausgabewert für Neuron j und o_j den optimalen Wert.

Für die Ableitung ergibt sich:

$$\frac{\partial E}{\partial o_j} = -(t_j - o_j)$$

Im zweiten Faktor wird die Funktion für den optimalen Ausgabewert von Neuron j (was nichts anderes ist als die Aktivierungsfunktion in Abhängigkeit vom Eingabewert e_j) nach dem Eingabewert e_j abgeleitet. Dieser Faktor, den man auch als $f'(e_j)$ schreiben kann, ist also der einzige, der von der Aktivierungsfunktion f abhängig ist. Als Beispiel soll wieder die logistische Sigmoidfunktion dienen:

$$f(x) = \frac{1}{1 + e^{-x}} \quad f'(x) = f(x) * (1 - f(x))$$

Somit ist $f'(e_j) = f(e_j) * (1 - f(e_j)) = o_j * (1 - o_j)$

Das Produkt aus den ersten beiden Ableitungen wird auch als „Fehlersignal“ $fsig_j$ bezeichnet. Der dritte Faktor ist sehr leicht zu verstehen. Die Funktion für den Eingabewert e_j wird nach dem Gewicht der jeweiligen Synapse abgeleitet:

$$\frac{\partial e_j}{\partial w_{ij}} = o_i$$

Man erhält als Änderung des Synapsengewichtes schließlich (natürlich wieder beispielhaft für die logistische Sigmoidfunktion):

$$\Delta w_{ij} = \eta * (t_j - o_j) * o_j * (1 - o_j) * o_i$$

Nun lässt sich der Backpropagation-Algorithmus konkretisieren:

PROCEDURE Backpropagation

 REPEAT

 E := 0

 FOR i:=1 TO AnzahlMuster DO

 Erstelle Eingabemuster

 FOR j:=1 TO Neuronenzahl DO

 fsig_j := (t_j - o_j) * o_j * (1 - o_j)

 E := E + (t_j - o_j)²

 FOR s:= Outputschicht DOWNTO Inputschicht DO

 FOR k:=1 TO AnzahlNeuronen_in_s DO

 fsum := 0

 FOR m:=1 TO AnzahlNeuronen_in_s+1 DO

 fsum := fsum + w_{km} * o_m

 fsig_k := o_k * (1 - o_k) * fsum

 FOR m:=1 TO AnzahlNeuronen_in_s+1 DO

 w_{km} := w_{km} + η * o_k * fsig_m

 UNTIL E innerhalb eines Grenzwertes

 END

Bemerkung: Durch die Verwendung von fsum werden die (bereits verbesserten) Fehler der Nachfolgeschicht(en) bei der Veränderung des Synapsengewichtes berücksichtigt.

Um einen Eindruck von der Funktionsweise des Algorithmus zu erhalten, sei an dieser Stelle auf ein Excel-Sheet verwiesen, das den Backpropagation-Algorithmus auf das XOR-Problem anwendet. Es ist erhältlich unter: <http://www.wi.hs-wismar.de/ki-buch/Kap06/Backprop.xls>

5. Literaturverzeichnis

Heinz, Alois: „Neuronale Netze“ in: T. Ottmann (Hrsg.): „Prinzipien des Algorithmenentwurfs“, Spektrum-Verlag

Lämmel, Uwe und Cleve, Jürgen: „Künstliche Intelligenz“, Fachbuchverlag Leipzig, 2001

Feldbusch, F.: „Klassifikation von Geräuschen“

URL: http://www.ira.uka.de/I3V_HTML/FORSCHUNGSVORHABEN/00264235.htm

Gläbel, Holger: „Einführung in die künstliche Intelligenz – Neuronale Netze“, Institut für Statistik und Dynamik der Luft- und Raumfahrtkonstruktionen, Universität Stuttgart

URL: http://www.isd.uni-stuttgart.de/arbeitsgruppen/pigroup/vorlesung_ki/Folien/07_Neuronale%20Netze.pdf

Faulkner, Johannah: „Einführung in Neuronale Netze“, Institut für Informatik, Universität Tübingen

URL: http://www-ti.informatik.uni-tuebingen.de/~schroedm/proseminar/PDFs/KNN_uebersicht.pdf

Petry, Nikolaus: „Fuzzy-Logik und Neuronale Netze“, JurPC Web-Dok. 187/1999, Abs. 28 – 52

URL: <http://www.jurpc.de/aufsatz/19990187.htm>