Perzeptron-Netze

1. Grundlagen

Abb. 1 stellt ein typisches Perzeptron dar. Es besteht aus drei Schichten, der Eingabe-Schicht (links, input-layer), der Ausgabe-Schicht (rechts, output-layer) und einer mittleren Schicht, die meist als verdeckte Schicht (hidden layer) bezeichnet wird. Signale werden hier jeweils von links nach rechts weiter geleitet. Solche Netze bezeichnet man auch als vorwärts verkettete Netze.

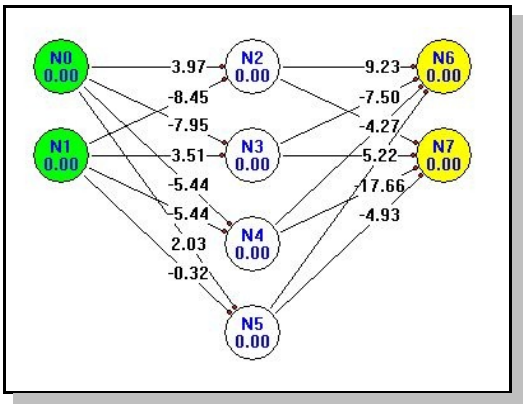

Abbildung 1

Die **Eingabe-Schicht** besteht aus Neuronen vom Input-Typ: Solche Neuronen verarbeiten selbst keine Signale; ihre Ausgabewerte werden vorgegeben (vgl. Kapitel "Neuronale Netze"). Diese Werte geben sie über Verbindungen an Neuronen in den nächsten Schichten weiter. Eingabe-Neuronen werden bei **`Netz32`** gelb markiert.

Die Neuronen in der **Ausgabe-Schicht** geben keine Signale weiter. Ihre Ausgabewerte stellen das Ergebnis der Verarbeitung durch das Netz dar. Ausgabe-Neuronen werden bei **`Netz32`** grün markiert.

Neuronen, die nicht zur Ein- bzw. Ausgabe-Schicht gehören, heißen verdeckte Neuronen; sie werden bein **`Netz32`** weiß markiert. Ein vorwärts verkettetes Netz kann auch mehrere **verdeckte Schichten** besitzen oder gar keine.

Perzeptren ohne verdeckte Schichten sind in ihrer Leistungsfähigkeit eingeschränkt; sie können noch nicht einmal eine XOR-Funktion realisieren. Dagegen sind Perzeptren mit mehr als 2 Schichten in der Lage, jede berechenbare Funktion zu realisieren. Im Folgenden werden wir nur dreischichtige Perzeptren betrachten.

2. Backpropagation

Damit ein Perzeptron eine bestimmte gewünschte Funktion hat, muss es eine adäquate Struktur haben: Z. B. muss es eine passende Anzahl von Eingabe-Neuronen und auch von Ausgabe-Neuronen besitzen. Auch die Anzahl der Neuronen bei der verdeckten Schicht ist wichtig für die Funktionsfähigkeit.

Zudem muss es angelernt werden. Das bedeutet: Es muss ein geeigneter Satz von Gewichten gefunden werden. Man unterscheidet dabei verschiedene Arten des Lernens: **überwachtes Lernen**, bestärkendes Lernen und überwachtes Lernen. Wir werden hier nur das überwachte Lernen betrachten. In dem Kapitel "Neuronale Netze" hatten wir es schon an einem einfachen

Beispiel kennen gelernt: Eingabe-Daten und zugehörige Sollwerte für die Ausgabe-Neuronen stehen zur Verfügung. Damit trainiert man das Netz, um so einen passenden Satz von Gewichten zu erhalten.

Wie läuft diese Training ab? Zunächst belegt man die Gewichte des Netzes mit Zufallszahlen zwischen -1 und 1. Die Eingabe-Neuronen werden nun mit den Eingabe-Daten gefüttert; anschließend wird das Netz propagiert. Jetzt vergleicht man die so bestimmten Ausgabewerte mit den Sollwerten. Aus der Differenz dieser beiden Werte gibt das Fehlersignal für jedes Ausgabe-Neuron an; damit kann man im Netz *rückwärts schreitend* die Fehlersignale zunächst für die verdeckten Neuronen und dann auch für die Eingabe-Neuronen ermitteln. Mit Hilfe dieser Fehlersignale kann man dann neue Werte für die Gewichte ausrechnen; im Allgemeinen sind diese noch nicht optimal, aber immerhin besser als die vorhergehenden Gewichtswerte. Dieses Vorgehen bezeichnet man als **Backpropagation**.

Nun führt man das gleiche Verfahren mit diesen Gewichten erneut durch; jetzt sollten die Fehlersignale bei den Ausgabe-Neuronen schon kleiner sein als zuvor. Dementsprechend kleiner sind dann auch die restlichen Fehlersignale beim Rückwärts-Schreiten; auch die fällige Änderung bei den Gewichtswerten wird kleiner ausfallen.

Auf diese Weise werden sukzessive die Gewichte abgeändert, idealerweise bis die Fehlersignale bei den Ausgabe-Neuronen (nahezu) null sind.

In der Regel wird unser Netz nicht nur eine einzige Aufgabe beherrschen sollen. In diesem Fall bietet es sich an, bei jedem Iterationsschritt die Backpropagation nacheinander für alle Aufgaben durchzuführen.

Wie schnell der Lernfortschritt vonstatten geht, wie stark sich also die Gewichte von Iterationsschritt zu Iterationsschritt ändert, kann man mit der so genannten **Lernrate** σ (sigma) steuern, die bei der Backpropagation benutzt wird: Je größer dieser Wert ist, desto größer ist die Veränderung bei den Gewichten und um so rascher das Lerntempo. Bei **`Netz32`** hat sie standardmäßig den Wert 1. Sollte die Iteration einmal nicht zu stabilen Gewichten führen, empfiehlt es sich die Lernrate auf einen niedrigeren Wert zu stellen.

Für detailliertere Informationen zum Backtracking sei auf die Fachliteratur verwiesen. Hier wollen wir in den nächsten Abschnitten lieber einige Beispiele vorstellen.

Vorher aber noch eine ***wichtige Bemerkung***: Das Programm Netz32 führt die Propagation in der Reihenfolge durch, mit der die Neuronen indiziert sind. Damit die Propagation bei einem Perzeptron-Netz korrekt bei den Neuronen der Eingabeschicht beginnt und dann von Schicht zu Schicht berechnet wird, müssen beim Erstellen eines Netzes zunächst die Neuronen der Eingangsschicht, dann erst die der verdeckten Schicht und zuletzt die der Ausgangsschicht erzeugt werden. Dies spielt natürlich auch eine Rolle für die Backpropagation, bei der die Rechnung im Netz rückwärts schreitet.

3. Neuronale Netze für die XOR-Funktion

Die XOR-Funktion (eXclusiv OR = ausschließendes Oder) ist eine logische Funktion mit der rechts stehenden Wertetabelle. Die von uns benutzten (differenzierbaren) Aktivierungsfunktionen können die Werte 0 und 1 nur näherungsweise erreicht werden; deswegen werden wir in der rechten Spalte diese Werte durch 0,05 und 0,95 ersetzen. Gegebenenfalls kann man nach dem Lernprozess beim Ausgabe-Neuron die Outputfunktion von "identisch" auf "binär" mit der Schwelle $\xi = 0{,}5$ umstellen; so beschränken sich die Ausgabe-Werte auf 0 und 1.

A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

Wir beginnen mit dem Netz aus Abb. 2. Die benutzten Aktivierungsfunktionen sind vom Typ "sigmoid" mit Schwellenwert 0 und Parameterwert 1. Man findet die entsprechende Datei `xor1a.net` im Ordner `xor`.

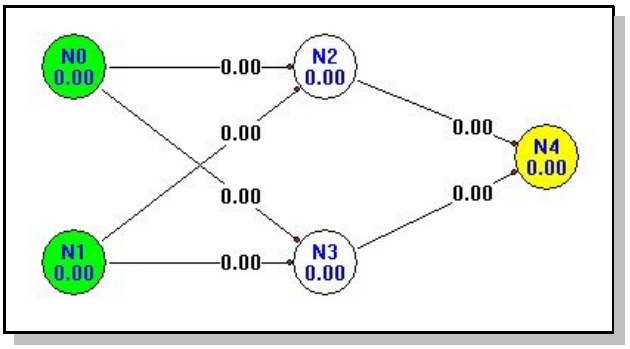

Abbildung 2

Die Gewichte werden nun mit Hilfe der Schaltfläche  mit zufälligen Werten belegt. Dann wird die (modifizierte) Wertetabelle als Aufgabe eingeben und anschließend die Schaltfläche <Alle> betätigt.

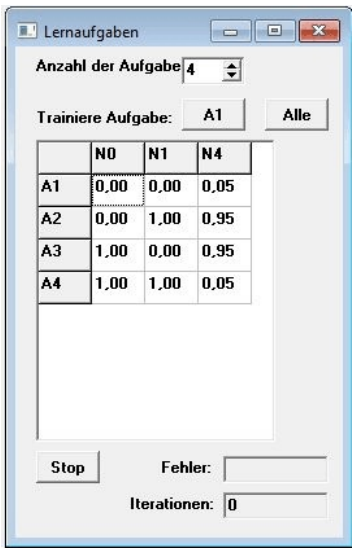

Abbildung 3

Das Verfahren konvergiert nicht: Auch nach über 500 000 Iterationen gibt es gerade bei der 4. Aufgabe einen großen Fehlerwert.

Sind die Gewichte zufälligerweise ungünstig belegt worden? Wie starten die Iteration mit einer anderen Belegung der Gewichte. Wieder ist das Konvergenzverhalten vergleichbar schlecht.

Ist der Schwellenwert 0 vielleicht ungeschickt gewählt? Wir ändern ihn in der verdeckten Schicht und beim Ausgabe-Neuron auf verschiedene Weisen ab. Das Verfahren konvergiert nun und nach etwa 100 000 Iterationen ist der maximale Fehlerwert $6 \cdot 10^{-5}$.

Nun könnte man auf diese Weise weitere Werte für ϑ austesten. Dies ist sehr mühselig. Mit einem Trick können wir indes dafür sorgen, dass das Netz selbst bessere Schwellenwerte lernt. Dazu benutzen wir das Netz aus Abb. 4. Gegenüber dem Netz aus Abb. 4 besitzt es ein weiteres Neuron N0. Als Eingänge A und B dienen jetzt die Neuronen N1 und N2.

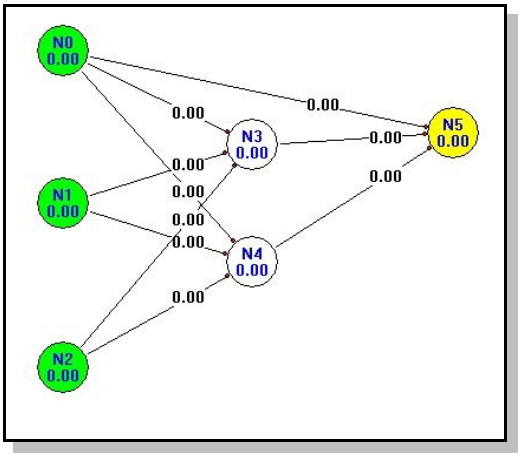

Abbildung 4

Das Neuron N0 erhält nun den konstanten Ausgangswert 1 (vgl. Abb. 5); man bezeichnet es deswegen auch als **on-Neuron**. Zusammen mit den Gewichten wirkt dieses Neuron bei den Neuronen N3, N4 und N5 wie ein Schwellenwert. Das schauen wir uns einmal für das Neuron N5 an: Ohne das Neuron N0 ist der Eingangswert von N5 gegeben durch

$$i_5 = w_{3,5} \cdot a_3 + w_{4,5} \cdot a_4 .$$

Mit dem Neuron N0 ist der Eingangswert hingegen

$$\begin{aligned} i_5 &= w_{3,5} \cdot a_3 + w_{4,5} \cdot a_4 + w_{0,5} \cdot 1 \\ &= w_{3,5} \cdot a_3 + w_{4,5} \cdot a_4 + w_{0,5} \end{aligned}$$

Das Gewicht $w_{0,5}$ wirkt sich also genauso aus wie ein Schwellenwert $\vartheta_5 = - w_{0,5}$. Das Gleiche gilt natürlich für die beiden anderen Gewichte $w_{0,3}$ und $w_{0,4}$. Bei der Backpropagation werden nunmehr auch die Schwellenwerte von N3, N4 und N5 in Form dieser Gewichte optimiert. Dadurch wird die Konvergenz deutlich beschleunigt: Bereits nach etwa 10 000 Iterationen haben wir die Genauigkeitsgrenze von $1 \cdot 10^{-5}$ bei allen Aufgaben erreicht.

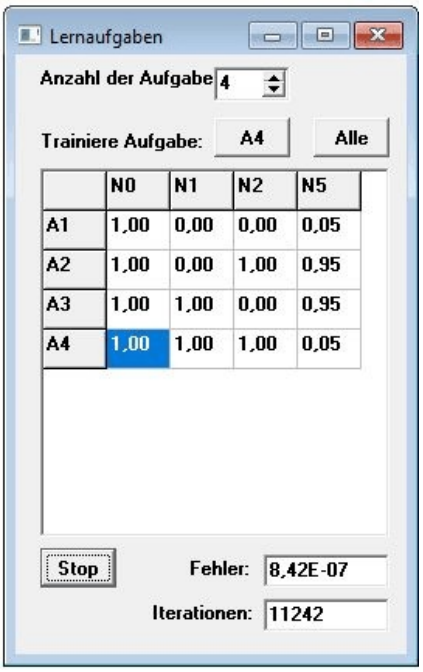

Abbildung 5

In der Regel werden beim Perzeptron nur die Neu ronen benachbarter Schichten verbunden. In Abb. 4 hatten wir uns mit dem on-Neuron schon darüber hinweg gesetzt. Bei dem Netz von Abb. 7 ist nun nicht nur das on-Neuron N0 mit den Neuronen aus der verdeckten Schicht und der Ausgabe-Schicht verbunden, sondern auch die beiden Eingabe-Neuronen. Dafür wurde die verdeckte Schicht auf ein einzige Neuron reduziert.

Dieses Netz lässt sich fast so rasch wie das von Abb. 5 trainieren.

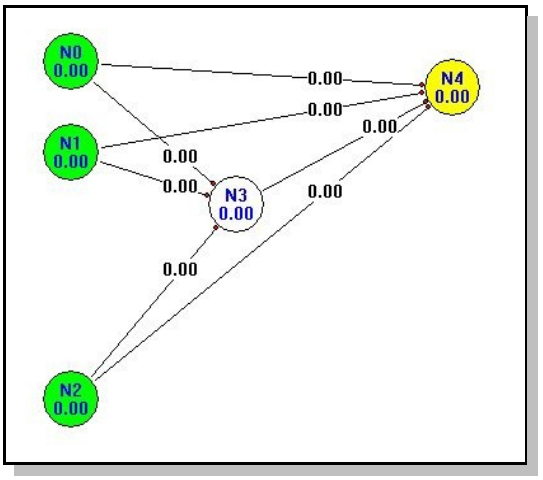

Abbildung 6

4. Addierer

Unser Binär-Addierer soll zwei Eingabe-Neuronen für die beiden (einstelligen) Summanden A und B sowie zwei Ausgabe-Neuronen für die Summe S und den Übertrag Ü besitzen. Für die verdeckte Schicht spendieren wir 4 Neuronen.

Bei umfangreicheren 3-schichtigen Perzeptren lohnt es sich den Netzgenerator von **`Netz32`** zu benutzen. Dazu betätigen wir in der Werkzeugleiste die Schaltfläche  . Es erscheint das Fenster aus Abb. 7; hier bestätigen wir die vorgegebenen Einstellung mit <OK>.

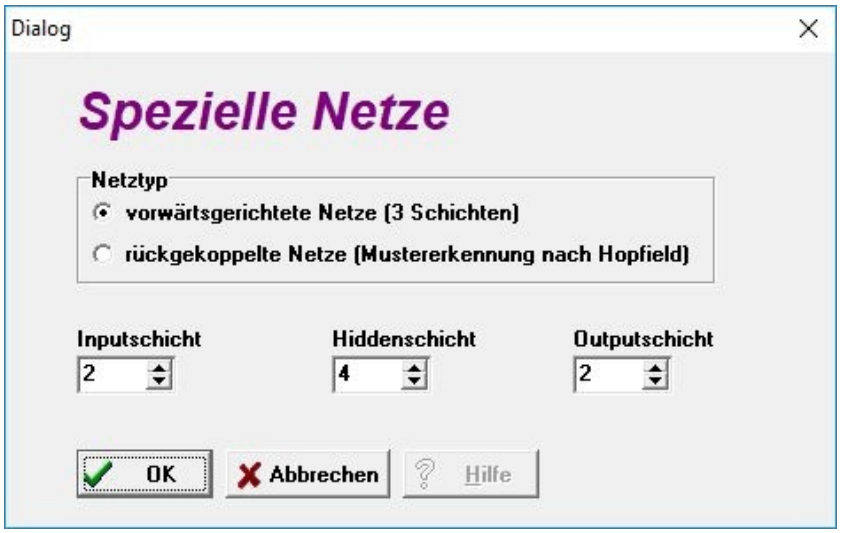

Abbildung 7

Damit erhalten wir das Netz aus der Abbildung 8.

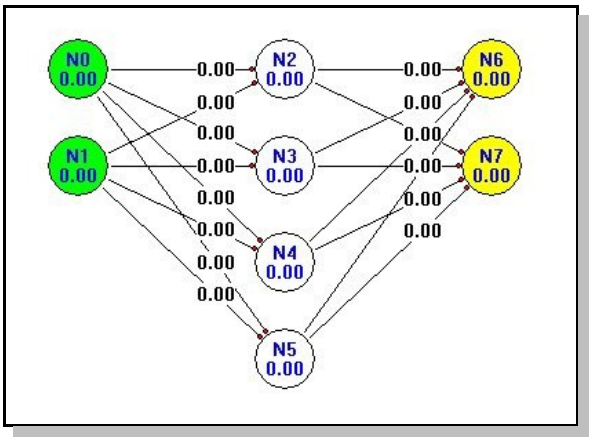

Abbildung 8

Dieses Netz muss nun die Wertetabelle

A	B	Ü	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	1

lernen. Wie oben bereits dargestellt, geben wir diese Daten im Lernaufgaben-Fenster ein. Anschließend geben wir den Gewichten zufällige Werte und starten die Iteration. Wenn die Genauigkeit 10^{-3} unterschritten wird, können wir die Iteration abbrechen. Nachträglich stellen wir die Ausgabe-Neuronen auf "binär" mit der Schwelle $\xi = 0{,}5$ ein.

5. 4-2-4-Encoder

In unserem letzten Beispiel soll ein Perzeptron die Bitmuster 1-0-0-0, 0-1-0-0. 0-0-1-0 und 0-0-0-1von den 4 Neuronen der Eingabe-Schicht in die 4 Neuronen der Ausgabe-Schicht übertragen. Das Entscheidende dabei ist, dass die verdeckte Zwischenschicht aus nur zwei Neuronen besteht. Die Informationen aus der Eingabe-Schicht müssen bei der Propagation durch die verdeckten Schicht wie durch einen engen Flaschenhals gehen. Es ist interessant zu erforschen, in welcher Form die verschiedenen Bitmuster in dem trainierten Netz aus der Eingabe-Schicht in der verdeckten Schicht repräsentiert werden.

Wir benutzen das Netz `encoder_4_2_4.net` aus dem Ordner `encoder`. Das Neuron N4 hat hier wieder die Funktion eines on-Neurons. Die zugehörigen Aufgaben-Datei findet man in

der Datei `encoder_4_2_4.lrn`. Wir führen die Iteration mit einer Lernrate von σ = 0,1 durch. Nach etwa 200 000 Iterationen ist der Fehler nur noch ca. 10^{-4} groß. Wir brechen nun die Iteration ab und schauen uns an, welche Werte die Neuronen der verdeckten Schicht jetzt bei den 4 Bitmustern aufweisen. Dazu stellen wir zunächst unser erstes Bitmuster 1-0-0-0 in der Eingabe-Schicht ein; wir vergessen nicht, unserem On-Neuron den Wert 1 zu geben. Anschließend wird das Netz propagiert.

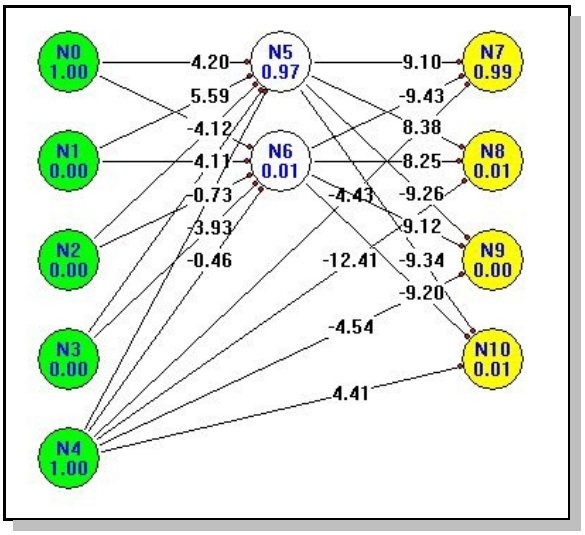

Abbildung 9

Abb.9 zeigt zunächst, dass das Bitmuster in der Ausgabe-Schicht von kleinen Abweichungen abgesehen dem Bitmuster aus der Eingabe-Schicht entspricht. Es ist also korrekt durch den "Flaschenhals" übertragen worden. Interessant sind nun die Werte der Neuronen N5 und 6. Hier finden wir das Wertepaar (0,97; 0,01). In der folgenden Tabelle haben wir die Wertepaare für alle 4 Bitmuster aufgelistet; dabei haben wir die Werte aus der verdeckten Schicht jeweils gerundet.

Eingabe-Schicht	verdeckte Schicht
1-0-0-0	(1,0 | 0,0)
0-1-0-0	(1,0 | 1,0)
0-0-1-0	(0,0 | 1,0)
0-0-0-1	(0,0 | 0,0)

Unser Netz hat hier für die Repräsentation der 4 Bitmuster (näherungsweise) eine binäre Codierung benutzt! Die linke Seite des Netzes kann man somit als Kodierer, die rechte als Dekodierer ansehen.