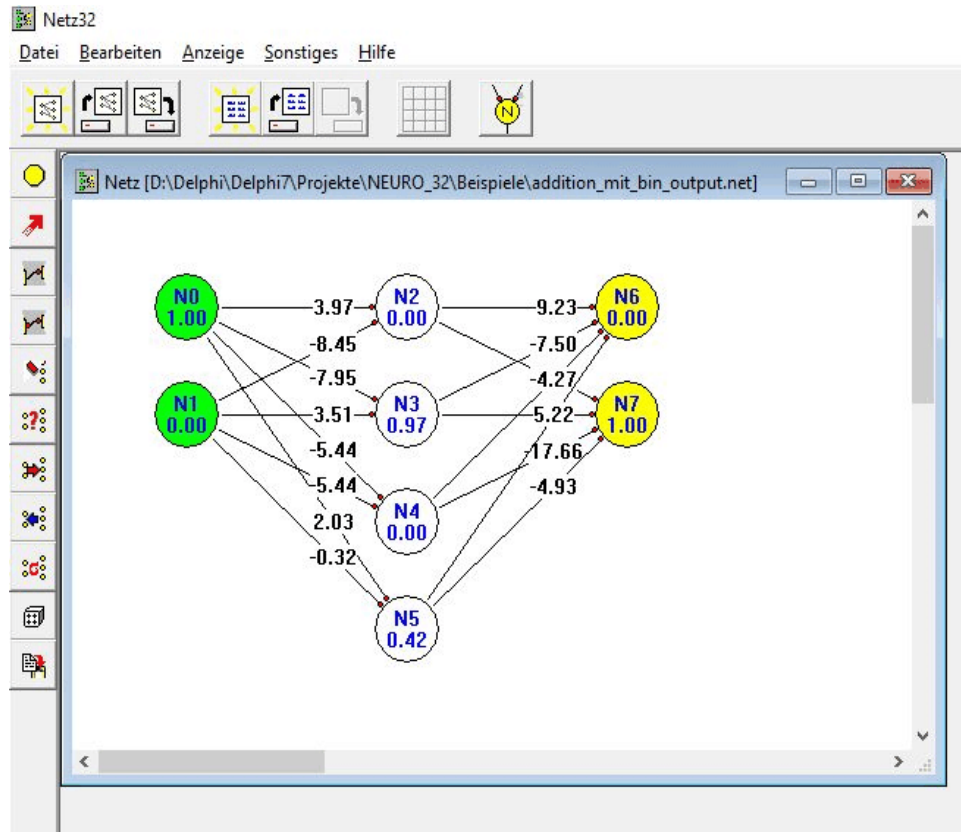


Netz32



**Eine praktische Einführung
in die Welt der
neuronalen Netze**

*von Georg Heinrichs
31.03.2026*

Netz32

Netz32 ist ein Lern-Programm, mit dem sich einzelne Neuronen und neuronale Netze eingehend studieren lassen. Es läuft unter Windows ab der Version Windows XP, insbesondere auch unter Windows 10 und 11 (64 Bit).

1. Installation

Entpacken Sie die ZIP-Datei **Netz32** in ein Verzeichnis ihrer Wahl. Öffnen Sie die Datei **ReadMe.txt** und befolgen Sie die Anweisungen in dieser Datei.

2. Aufbau

Im Wesentlichen besteht die Programm-Oberfläche aus einer **Menüleiste**, einer **Werkzeuggeste**, einem **Fenster für Netze**, in welchem mit neuronalen Netzen experimentiert werden kann, sowie aus einer **Zeile mit Hinweisen** für den Benutzer (vgl. Abb. 1)

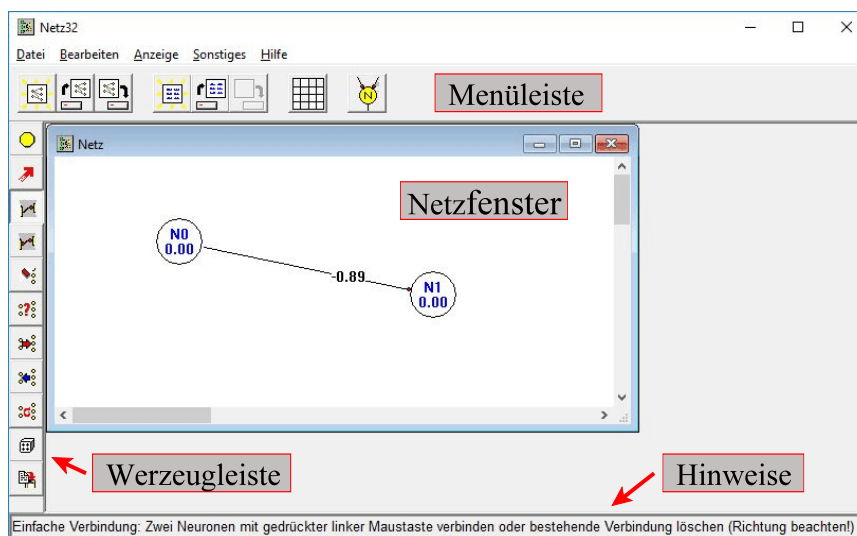


Abb. 1

Über die **Menüleiste** können Netze in einem Fenster erzeugt, aus einer Datei geöffnet oder auch gespeichert werden. Zudem können hier Lernaufgaben für diese Netze erzeugt, geladen und gespeichert werden. Zum detaillierten Studium der Funktionsweise eines einzelnen Neurons kann man über die Menüleiste auch ein Fenster mit einem Test-Neuron öffnen (vgl. Abb. 2).

Netz32

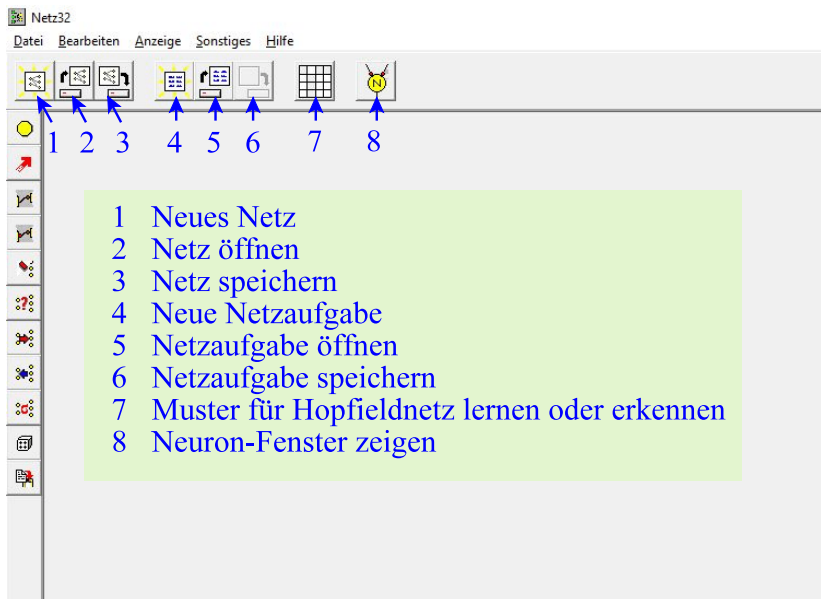


Abb. 2

Die **Werkzengleiste** steht nur zur Verfügung, wenn bereits ein Netz-Fenster mit den Schaltflächen 1 oder 2 erzeugt worden ist. Wie man mit diesen Werkzeugen umgeht, wird in den folgenden Kapiteln ausführlich dargestellt.

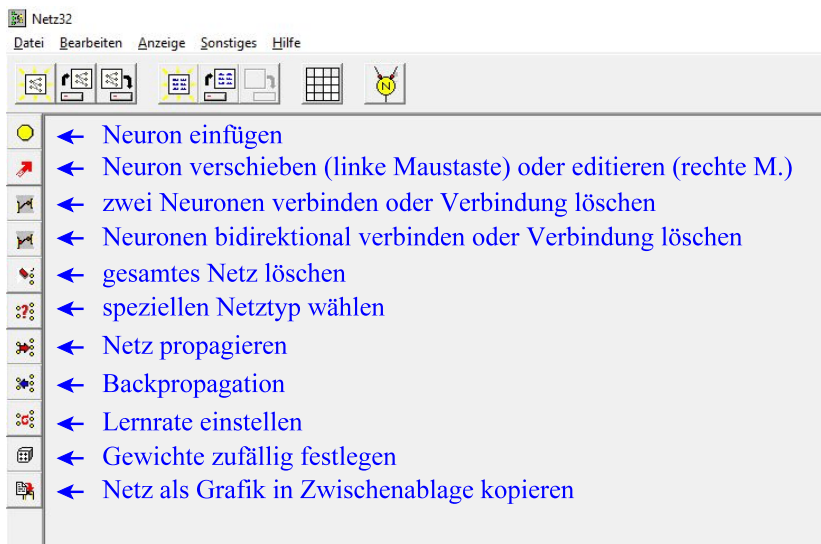


Abb. 3

Neuronen

1. Biologische Neuronen

Neuronen sind Zellen, die Signale empfangen, verarbeiten und aussenden können. In Abb. 4 geht von dem “blauen” Neuron (1) eine lange Nervenfaser (**Axon** genannt) aus und koppelt über eine so genannte **Synapse** an eine Nervenfaser (**Dendrit**) des “grünen” Neuron (2). Über diese Verbindung können nun Signale von dem Neuron 1 an das Neuron 2 übertragen werden. Dieses Neuron 2 verarbeitet die Signale, die sie von ihren sämtlichen Dendriten erhält und gibt das resultierende Signal über ein Axon aus; dieses Axon kann verzweigen, so dass das Ausgangssignal von unserem Neuron 2 auch zu mehreren anderen Neuronen gelangen kann. Bei der Verarbeitung kommt es darauf an, wie **stark** die empfangenen Signale sind und ob sie **erregend** oder **hemmend** wirken. Wie stark ein empfangenes Signal ist, hängt nicht nur davon ab, wie stark es ausgesendet wird, sondern auch davon, wie stark die Kopplung von Axon und Dendrit ist. Die verschiedenen Stärken dieser Kopplung machen die neuronale Struktur unseres Gehirns aus: **Lernen bedeutet, diese Kopplungen zu verändern.**

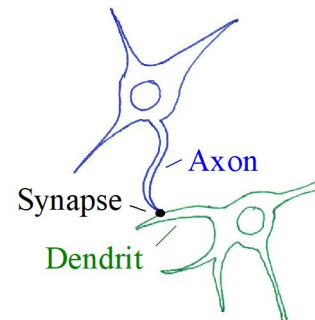


Abb. 4

2. Künstliche Neuronen

Künstliche Neuronen können wir uns als Objekte vorstellen: Diese haben eine Reihe von Eingängen, die den Dendriten entsprechen. Die Stärke der Signale wird durch Zahlenwerte beschrieben; positive Werte stehen für eine erregende Wirkung, negative für eine hemmende. Die Eingangssignale werden verarbeitet und über einen Ausgang (der dem Axon entspricht) an andere Neuronen weitergegeben.

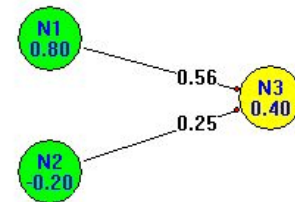


Abb. 5

2.1 Abb. 5 zeigt ein Neuron N3, welches Signale von den beiden Neuronen N1 und N2 erhält. Das Signal, welches vom Neuron N1 ausgeht, wird nun nicht in gleicher Stärke beim Neuron N3 ankommen; die unterschiedliche Kopplung von Axon und Dendrit berücksichtigen wir, indem wir die Ausgangswerte a_1 und a_2 der Neuronen N1 und N2 mit Zahlen w_1 und w_2 multiplizieren. (In Abb. 2 sind es die Werte 0,56 und 0,25.) Diese Produkte bestimmen die Eingangswerte unseres Neurons N3; sie werden zu einem einzigen Wert i zusammen gefasst, der “gewichteten Summe” von a_1 und a_2 :

$$i = w_1 \cdot a_1 + w_2 \cdot a_2$$

Die Werte w_1 und w_2 bezeichnet man als **Gewichte**, sie können positiv (erregend), Null (wirkunglos) oder negativ (hemmend) sein.

Neuronen

- 2.2 Diesen Eingangswert verarbeitet nun unser Neuron N3. Diese Verarbeitung lässt sich durch eine so genannte **Aktivierungsfunktion** beschreiben. Im einfachsten Fall benutzt man dafür eine **Schwellwertfunktion**:

$$f_{\text{Schwellwert}}(i) = \begin{cases} 1 & \text{für } i \geq \vartheta \\ 0 & \text{sonst} \end{cases}$$

Dabei wird die Größe ϑ als **Schwellwert** bezeichnet: Wenn der Wert i größer oder gleich dem Schwellwert ist, dann ist der Ausgangswert gleich 1, sonst 0.

Häufig benutzt man aber als Aktivierungsfunktionen andere Funktionen; so sind z. B. für die Anwendung des Backpropagations-Verfahrens differenzierbare Aktivierungsfunktionen erforderlich.

Das Programm **Netz32** stellt mehrere Aktivierungsfunktionen zur Verfügung. Alle setzen sich aus 3 Schritten zusammen:

1. Schritt: Einführung eines Schwellwertes :

$$x = i - \vartheta$$

2. Schritt: Eigentliche Verarbeitung durch eine der folgenden 4 Funktionen:

$$y = f(x) = \frac{1}{1 + e^{-p \cdot x}} \quad (\text{zur Bedeutung von } p \text{ s. u.})$$

$$y = f_p(x) = \tanh(p \cdot x) = \frac{e^{p \cdot x} - e^{-p \cdot x}}{e^{p \cdot x} + e^{-p \cdot x}}$$

$$y = f(x) = \begin{cases} -1 & \text{für } x < 0 \\ +1 & \text{sonst} \end{cases}$$

$$y = f_p(x) = \begin{cases} -1 & \text{für } p \cdot x < -1 \\ p \cdot x & -1 < p \cdot x < 1 \\ +1 & p \cdot x \geq 1 \end{cases}$$

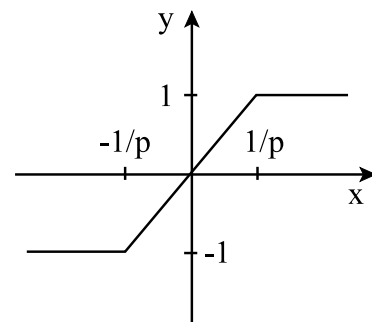


Abb. 6

Die dritte Funktion ist unstetig: An der Stelle 0 macht ihr Graph einen Sprung. Die letzte Funktion "glättet" diesen Sprung, indem hier auf dem Intervall $[-1/p; +1/p]$ eine lineare Funktion vorliegt, vgl. Abb. 6. Der Parameter p beschreibt dabei, wie steil dieser Über-

Neuronen

gang ist: Je größer p ist, desto rascher ist der Übergang.

Die ersten beiden Funktionen ähneln der letzten Funktion; allerdings sind diese beiden Funktionen nicht nur stetig, sondern auch differenzierbar.

3. Schritt (optional)

Meist stellt unser y -Wert aus dem 2. Schritt schon den Ausgangswert unseres Neurons dar. In manchen Fällen ist es sinnvoll, als Ausgangswerte nur 1 und 0 zuzulassen. In diesem Fall benutzt man als Ausgangswert

$$z = \begin{cases} 1 & \text{für } y \geq \xi \\ 0 & \text{sonst} \end{cases},$$

wobei hier ξ einen Schwellenwert für den Ausgang bedeutet.

- 2.3 Wie sich unser künstliches Neuron N3 verhält, hängt neben den Ausgangswerten a_1 und a_2 der Neuronen N1 und N2 von den Gewichten, den benutzten Aktivierungsfunktionen sowie den Schwellenwerten ab. Dies kann man an dem Test-Neuron von **Netz32** eingehend studieren: Hier werden die Ausgangswerte a_1 und a_2 in den grün unterlegten Feldern eingetragen; darunter gibt man die Gewichte ein. In dem grauen Neuronenfeld stellt man die Verarbeitungsart ein; dabei wird unsere erste Verarbeitungsart aus Schritt 2 (s. o.) als Sigmoid bezeichnet. Ansonsten sind alle Bezeichnungen so wie oben eingeführt.

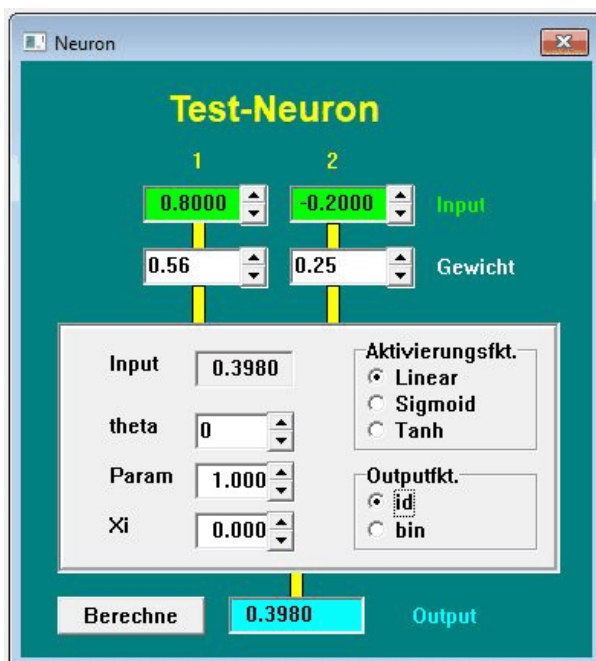


Abb. 7

Neuronale Netze

Neuronale Netze bestehen aus künstlichen Neuronen, die durch synaptische Verbindungen gekoppelt sind. Mithilfe des Lern-Programms **Netz32** lassen sich solche Netze erstellen und trainieren. Eine graphische Benutzeroberfläche erlaubt es, (nahezu) beliebige Strukturen von Netzen zu erzeugen; dabei kann man zwischen verschiedenen Aktivierungsfunktionen mit frei wählbaren Werten für die Schwellenwerte auswählen. Obendrein können spezielle Netztypen (3-schichtiges Perzeptron und Hopfieldnetz) automatisch generiert werden. Die Anzahl der Neuronen ist auf 60 beschränkt.

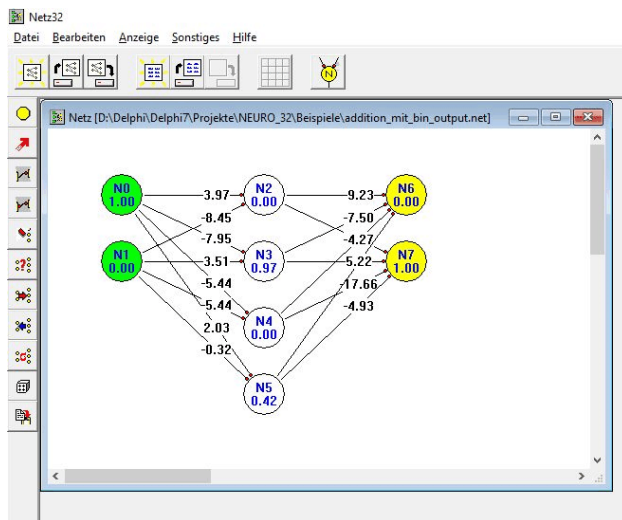


Abb. 8

1. Neuronale Netze erstellen

Um ein neuronales Netz zu erstellen, öffnen wir zunächst das Netz-Fenster, indem wir die linke Schaltfläche in der Menüleiste betätigen; alternativ können wir auch im Datei-Menü auf <Neues Netz> klicken.

In diesem Fenster wollen wir nun ein Netz wie in Abb. 9 erstellen. Zunächst erzeugen wir die 3 Neuronen. Dazu betätigen wir die oberste Schaltfläche in der Werkzeugleiste; diese Schaltfläche rastet ein und bleibt so lange aktiviert, bis sie erneut angeklickt wird (oder eine andere Schaltfläche aus der Werkzeugleiste betätigt wird). Anschließend klicken wir mit der linken Maustaste auf diejenige Position, an der das erste Neuron positioniert werden soll. Es erscheint ein Kreis mit der Bezeichnung N0; die Bezeichnung wird automatisch vom Programm erzeugt und kann nicht verändert werden. Die Zahl darunter gibt den Aktivierungs- bzw. Ausgangswert an. Beim Erzeugen eines Neurons ist dieser Wert standardmäßig 0.

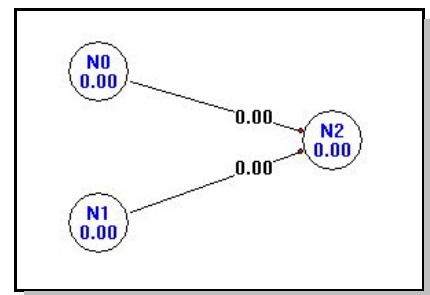




Abb. 9

Neuronale Netze


Für die beiden nächsten Neuronen klicken wir im Netzfenster auf die entsprechenden Positionen.



Nun sorgen wir für die Verbindungen: Wir klicken zuerst auf die Schaltfläche ; diese bleibt wieder eingerastet und wir können nun solange Verbindungen herstellen, bis diese Schaltfläche wieder deaktiviert wird. Dann positionieren wir den Mauszeiger auf dem Neuron N0 und ziehen ihn mit gedrückter linker Maustaste auf das Neuron N2. Hier lassen wir die Maustaste los und die Verbindung wird mitsamt ihrem Gewicht angezeigt. Der Gewichtswert ist standardmäßig 0.

Der rote Punkt am Neuron N2 zeigt an, dass hier ein Axon/Ausgang von N0 an ein Dendrit/Eingang von N2 ankoppelt. Möchte man eine umgekehrte Kopplung vornehmen, also ein Axon/Ausgang von N2 an ein Dendrit/Eingang von N0 koppeln, muss der Mauszeiger entsprechend von N2 nach N0 gezogen werden.

Die Verbindung von N1 nach N2 wird auf die gleiche Weise hergestellt. Anschließend deaktivieren wir die Schaltfläche .

2. Neuronale Netze speichern und öffnen

Unser erstes Netz ist fertig und soll jetzt mit der Schaltfläche  (oder mit <Datei>-<Netz speichern unter...>) gesichert werden; wir benutzen hier den Namen **Bsp1a** und legen die Datei in einem eigenen Ordner für dieses Kapitel ab. **Beachten Sie, dass beim Speichern nur die Struktur des Netzes und seiner Neuronen, nicht aber Signalwerte gesichert werden.**

Nach dem Sichern können wir unser Netz löschen; das geschieht mit der Schaltfläche . Mit der Schaltfläche  können wir es wieder öffnen.

3. Neuronale Netze editieren


Unser gerade geöffnetes Netz können wir ergänzen oder ändern: Mit <Bearbeiten> - <Letztes Neuron löschen> können wir das letzte Neuron, also das mit dem höchsten Index, löschen. Verbindungen zwischen zwei Neuronen kann man löschen, indem man genau so verfährt wie beim Herstellen der Verbindung: Erneutes Zeichnen löscht die Verbindung.

Auch die Schwellenwerte und Aktivierungsfunktionen können wir editieren. Dazu aktivieren wir in der Werkzeugleiste die Schaltfläche mit dem roten Zeiger und klicken mit der *rechten* Maustaste auf das zu editierende Neuron. Es erscheint ein Fenster mit umfangreichen Editiermöglichkeiten, vgl. Abb. 11.

Wie die Gewichte verändert werden können, wird in dem Kapitel über Lernprozesse ausführlich erläutert.

Neuronale Netze

4. Propagieren eines Netzes

Bei biologischen neuronalen Netzen werden die Signale unmittelbar von einem zum anderen Neuron weitergeleitet. Bei unserem künstlichen neuronalen Netz geschieht dies der Reihe nach (entsprechend dem Index der Neuronen); diese Signalweitergabe bezeichnet man als **Propagieren**. Eine solches Propagieren wird bei unserem Programm ausgeführt, indem man die Schaltfläche  betätigt.

Bei unserem Beispiel aus Abb. 10 wollen wir dies praktisch durchführen. Zunächst öffnen wir die Datei **Bsp1c** aus dem Ordner **Beispiel1**. Dass die beiden Neuronen N0 und N1 hier grün eingefärbt sind, kennzeichnet sie als **Eingabe-Neuronen**. Bei biologischen Neuronen entspricht das Sinneszellen, deren Ausgabewert durch einen äußeren Einfluss bestimmt wird. Hier bedeutet es, dass der Ausgabewert durch den Benutzer vorgegeben werden kann. Die Farbe Gelb kennzeichnet das Neuron N2 als **Ausgabe-Neuron**. Dies wird erst im Zusammenhang mit Lernprozessen eine Rolle spielen.

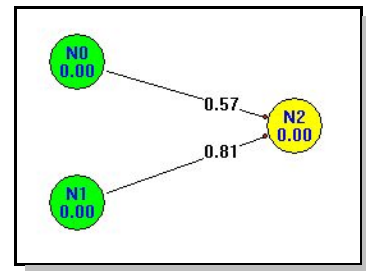



Abb. 10

Die Gewichte 0,57 und 0,81 sind hier zufällig festgelegt worden; dies geschieht mit der Schaltfläche .

Welchen Wert erhält das Neuron N2 als Ausgangswert, wenn die Neuronen N0 und N1 die Werte 0,8 bzw. 0,2 ausgeben? Zunächst editieren wir die beiden Neuronen N0 und N1, um deren Ausgabewerte festzulegen. Dazu aktivieren wir die Schaltfläche mit dem roten Pfeil und klicken das Neuron N0 mit der rechten Maustaste an. Es erscheint das Fenster aus Abb. 4. Hier geben wir als Sollwert die Zahl 0,8 ein; die restlichen Angaben spielen dann keine Rolle. Dann schließen wir das Fenster mit der OK-Schaltfläche. Ähnlich verfahren wir mit dem Neuron N1, um ihm den Ausgangswert 0,2 zu geben. Die beiden Neuronen N0 und N1 im Netzfenster weisen jetzt die neuen Werte 0,8 und 0,2 auf.

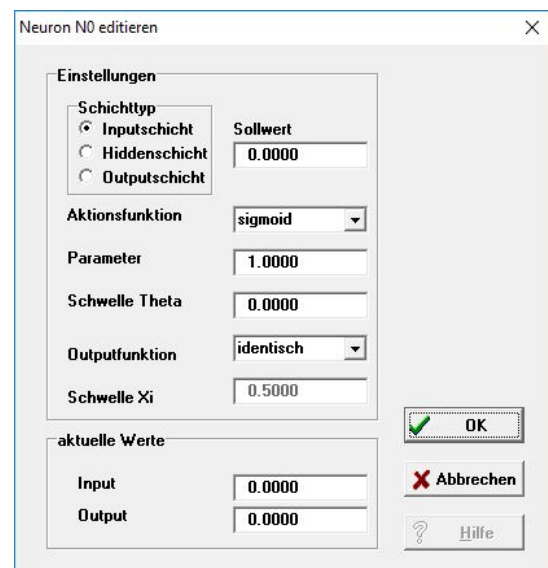



Abb. 11

Nun können wir eine Propagation mit Hilfe der Schaltfläche  durchführen. Das Neuron N2 erhält den Wert 0,62. Dies können wir nachrechnen; dazu müssen wir die speziellen Eigenschaften des Neurons N2 kennen. Im Editierfenster von N2 können wir sehen, dass die Schwelle

Neuronale Netze

den Wert 0 hat; die Aktivierungsfunktion ist “linear” mit der Steigung 1, vgl. Abb. 3 des Kapitels *Neuronen*. Das Feld für die Schwelle ξ ist hier deaktiviert; es spielt nur eine Rolle, wenn die Outputfunktion “binär” ist.

Die Rechnung lautet daher: $0,9 \cdot 0,57 + 0,2 \cdot 0,81 = 0,618 \approx 0,62$.

Ist der Wert von N0 nicht 0,8, sondern 2,0, dann erhält das Neuron N2 nach dem Propagieren den Wert 1,0. Hier macht sich bemerkbar, dass die Werte der Aktivierungsfunktion (betragsmäßig) maximal den Wert 1 haben 1, vgl. Abb. 6 des Kapitels *Neuronen*.

5. Eine erste Lernaufgabe

Wie die Ausgangs-Neuronen auf die Einstellungen bei den Eingabe-Neuronen reagieren, hängt entscheidend von den Gewichten des neuronalen Netzes ab. Im letzten Abschnitt hatten wir diese willkürlich vorgegeben und damit den Ausgabewert berechnen lassen. In der Praxis ist es aber häufig so, dass zunächst nur eine Reihe von Zielverhaltensweisen bekannt sind und daraus passende Gewichte ermittelt werden müssen. Diese Zielverhaltensweisen können wir mit Hilfe von **Lernaufgaben** beschreiben. Diese legen fest, welche Ausgabewerte gewissen Eingabewerten zugeordnet werden sollen.

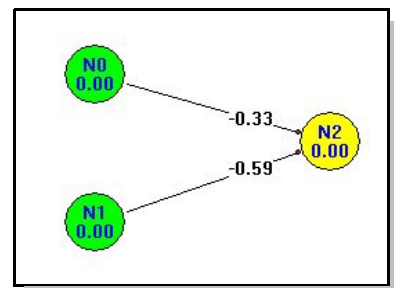


Abb. 12

Unser neuronales Netz aus Abb. 12 findet man in der Datei **Bsp1b**. Es soll folgende Aufgaben erfüllen:

Aufgabe	Eingabe		Ausgabe
	N0	N1	N2
A1	0,7	0	0,2
A2	0,2	0,6	0,9

Man überzeugt sich leicht, dass es weder die erste noch die zweite Aufgabe erfüllt. Welche Werte müssen nun die Gewichte haben? Für dieses einfache Netz lässt sich dies leicht ausrechnen. Da die Aktivierungsfunktion von Neuron N2 wieder “linear” mit der Steigung 1 ist, muss für die beiden Gewichte $g_{0,2}$ und $g_{1,2}$ das folgende lineare Gleichungssystem gelten:

$$\begin{cases} 0,7 \cdot g_{0,2} + 0 = 0,2 \\ 0,2 \cdot g_{0,2} + 0,6 \cdot g_{1,2} = 0,9 \end{cases}$$

Neuronale Netze

Aus der ersten Zeile ergibt sich unmittelbar: $g_{0,2} = \frac{0,2}{0,7} \approx 0,2857$. Setzen wir diesen Wert in die zweite Gleichung ein, erhalten wir für $g_{1,2}$ den Wert 1,4048.

Die Situation wird deutlich komplizierter, wenn das Netz komplexer ist und die Anzahl der Aufgaben größer ist. Eine zusätzliche Schwierigkeit taucht auf, wenn die Aktivierungsfunktion nicht linear ist. Für solche Fälle stellt das Programm **Netz32** ein Verfahren zur Verfügung, welches die gesuchten Gewichte durch **Iteration** näherungsweise bestimmt.


Wir wollen dieses Iterationsverfahren einmal auf den eben betrachteten Fall anwenden: Zunächst geben wir die Lernaufgaben ein. Dazu betätigen wir die Schaltfläche . Es erscheint das Fenster aus Abb. 13. Hier tragen wir in die A1-Zeile als erstes die Zahl 0,7, in das nächste Feld den Wert 0 und in das letzte Feld den Wert 0,2 ein. In die A2-Zeile tragen wir genauso die Werte für die zweite Aufgabe ein.



Abb. 13


Da unser Netz alle Aufgaben beherrschen soll, müssen alle “trainiert” werden; wir betätigen deswegen die Schaltfläche “Alle”. Nun beginnt das Iterationsverfahren.

Bei jedem Iterationsschritt werden aus den vorliegenden Gewichten verbesserte Werte für die Gewichte gebildet. Mit diesen neuen Werten wird für beide Aufgaben das Netz propagiert und kontrolliert, wie stark die jeweiligen Ausgangswerte von N2 nun von dem jeweiligen Sollwert abweichen. Diese Abweichung wird in dem Fehler-Feld angezeigt.

Sie können mit der Stop-Taste die Iteration abbrechen, wenn der Fehlerwert klein genug (z. B. kleiner als $1E-6$) ist.

Neuronale Netze

Während der Iteration werden die Gewichte im Netzfenster laufend aktualisiert; so erhalten Sie am Ende der Iteration die gesuchten Werte. Sie stimmen mit unseren berechneten Werten überein.

Was geschieht, wenn die Iteration mit anderen Startwerten für die Gewichte beginnt? Mit der Schaltfläche  sorgen wir für zufällige Gewichtswerte und starten die Iteration neu. Nach kurzer Zeit werden wieder die bekannten Gewichtswerte angezeigt.

Es dürfte klar sein, dass nicht zu jedem Satz von Lernaufgaben passende Gewichte gefunden werden können; insbesondere können keine passenden Gewichte gefunden werden, wenn sich einzelne Aufgaben widersprechen.

Übrigens: Ersetzt man beim Neuron N2 die Aktivierungsfunktion “linear” durch “sigmoid” oder “tanh”, erhält man ganz andere Werte für die beiden Gewichte.

Perzeptron-Netze

1. Grundlagen

Abb. 14 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.

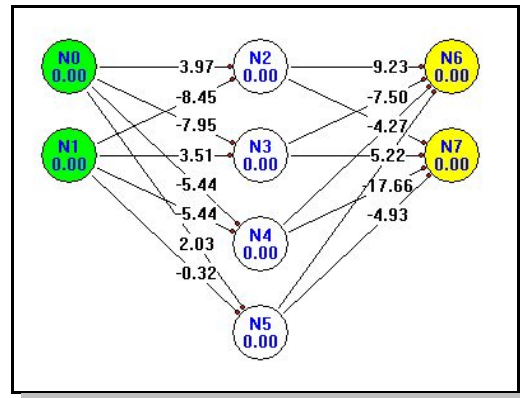


Abb. 14

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** grün 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** gelb markiert.

Neuronen, die nicht zur Ein- bzw. Ausgabe-Schicht gehören, heißen verdeckte Neuronen; sie werden bei **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 nicht überwachtes Lernen. Wir werden hier nur das über-

Perzeptron-Netze

wachte Lernen betrachten. In dem Kapitel “Neuronale Netze” hatten wir es schon an einem einfachen Beispiel kennengelernt: 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 dieses 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 ergibt sich das Fehlersignal für jedes Ausgabe-Neuron; 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 ändern, 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**: Unser Programm 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.

Perzeptron-Netze

3. Neuronale Netze für die XOR-Funktion

Die XOR-Funktion (eXklusiv OR = ausschließendes Oder) ist eine logische Funktion mit der rechts stehenden Wertetabelle. Mit den 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. 15. Die benutzten Aktivierungsfunktionen sind vom Typ "sigmoid" mit Schwellenwert 0 und Parameterwert 1. Man findet die entsprechende Datei **xor1a.net** im Ordner **xor**.

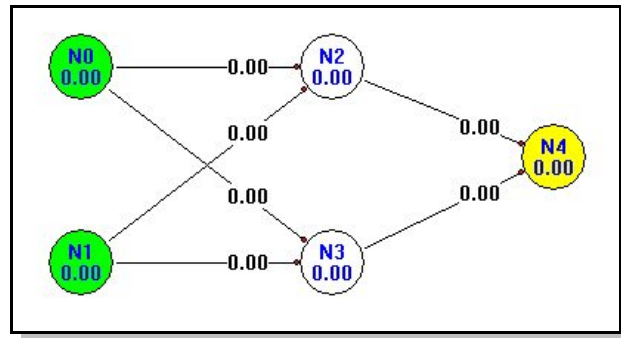



Abb. 15

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

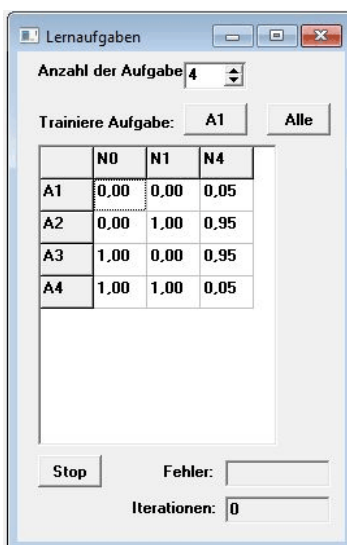


Abb. 16

Perzeptron-Netze

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. 17. Gegenüber dem Netz aus Abb. 4 besitzt es ein weiteres Neuron N0. Als Eingänge A und B dienen jetzt die Neuronen N1 und N2.

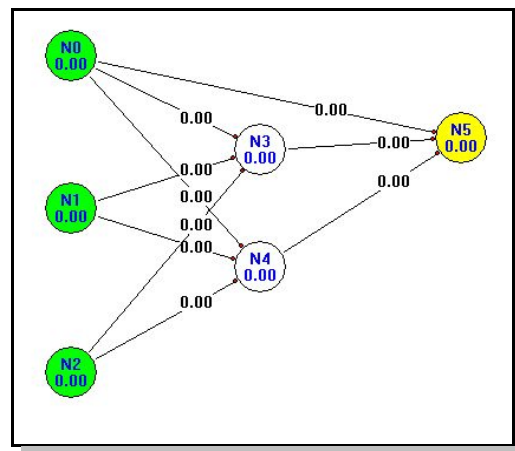


Abb. 17

Das Neuron N0 erhält nun den konstanten Ausgangswert 1 (vgl. Abb. 18); 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.

Lernaufgaben

Anzahl der Aufgabe: 4

Trainiere Aufgabe: A4 Alle

	N0	N1	N2	N5
A1	1,00	0,00	0,00	0,05
A2	1,00	0,00	1,00	0,95
A3	1,00	1,00	0,00	0,95
A4	1,00	1,00	1,00	0,05

Stop Fehler: 8,42E-07 Iterationen: 11242

Abb. 18

Perzeptron-Netze

In der Regel werden beim Perzeptron nur die Neuronen benachbarter Schichten verbunden. In Abb. 17 hatten wir uns mit dem on-Neuron schon darüber hinweg gesetzt. Bei dem Netz von Abb. 19 ist nun nicht nur das on-Neuron N0 mit den Neuronen aus der verdeckten Schicht und der Ausgangsschicht verbunden, sondern auch die beiden Eingabe-Neuronen. Dafür wurde die verdeckte Schicht auf ein einziges Neuron reduziert.

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

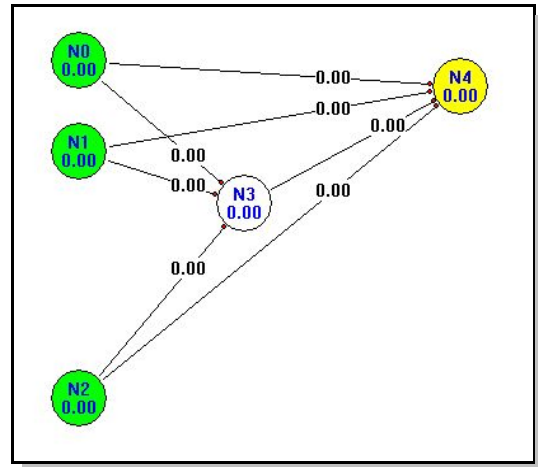



Abb. 19

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. 20; hier bestätigen wir die vorgegebenen Einstellung mit <OK>.

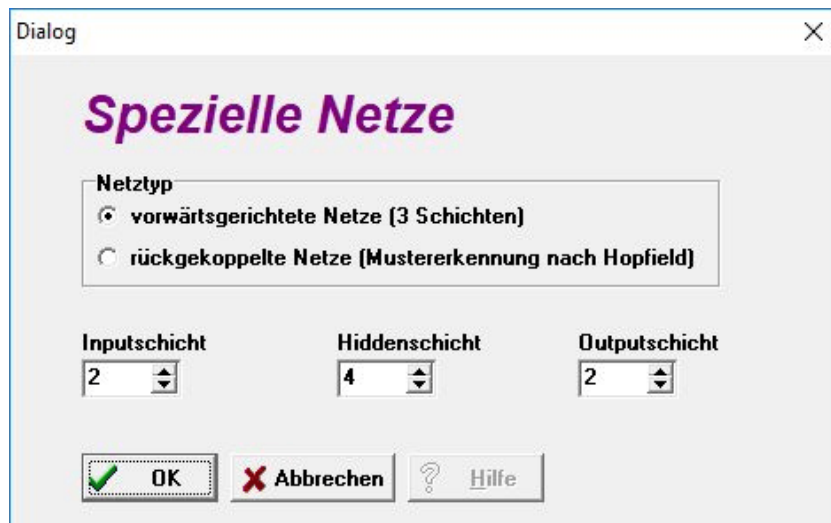


Abb. 20

Damit erhalten wir das Netz aus der Abbildung 21.

Perzeptron-Netze

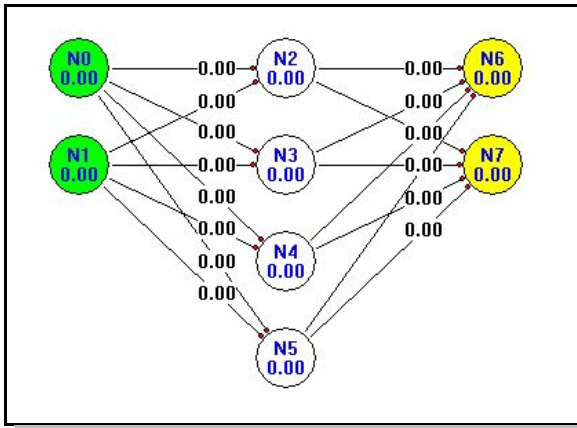


Abb. 21

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 0-0-0-1 von 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 verdeckte 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

Perzeptron-Netze

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 $\sigma = 0,1$ 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.

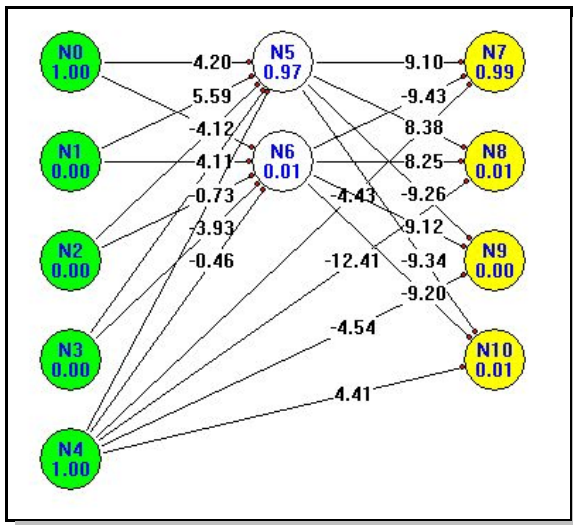


Abb. 22

Abb. 22 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 auf eine einzige Nachkommastelle 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.

Hopfield-Netze

1. Rückgekoppelte Netze

Bislang waren die Neuronen in Schichten angeordnet und (meist) auch nur von Schicht zu Schicht vernetzt. Dabei wanderten die Informationen jeweils in einer Richtung von der Eingabeschicht zur Ausgabeschicht. Bei natürlichen neuronalen Netzen können sich Neuronen aber auch wechselseitig beeinflussen. In Abb. 23 ist ein einfaches künstliches neuronales Netz dargestellt, in dem solche Rückkopplungen auftauchen. So erkennt man z. B. bei der Verbindung zwischen dem Neuron N0 und dem Neuron N1 zwei Synapsen, eine bei N1 und eine weitere bei N0. In Wirklichkeit handelt es sich hier also nicht um eine einzige, sondern um eine doppelte Verbindung mit den Gewichten $g_{0,1}$ und $g_{1,0}$. Wir werden im Folgenden nur solche rückgekoppelten Netze betrachten, bei denen die beiden Gewichte einer solchen doppelten Verbindung jeweils gleich sind. Deswegen findet man in Abb. 23 an den Verbindungslinien jeweils nur einen Gewichtswert.

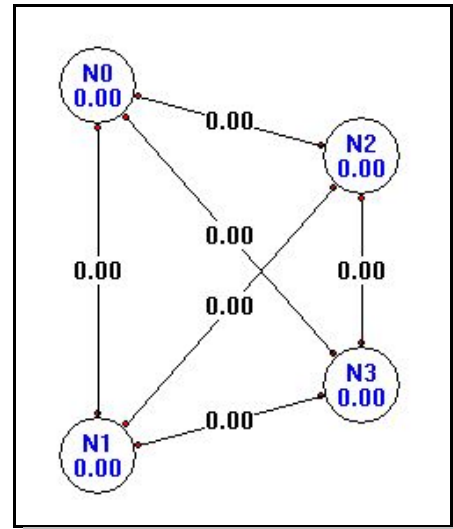


Abb. 23

2. Muster und Hopfield-Netze

Solchen rückgekoppelten neuronalen Netze können Bildmuster eingepreßt werden; sie können diese dann aus Fragmenten rekonstruieren. Dazu ordnet man jedem Bildpunkt (Pixel) des Bildes ein Neuron zu. Jedes Neuron wird mit jedem *anderen* Neuron durch eine doppelte Verbindung verknüpft. Sinnvollerweise positioniert man diese Neuronen im Netz genauso wie die einzelnen Pixel. Bei größeren Mustern blendet man dabei meist in der Grafik die Verbindungen mit ihren Gewichten aus (vgl. Abb. 24).

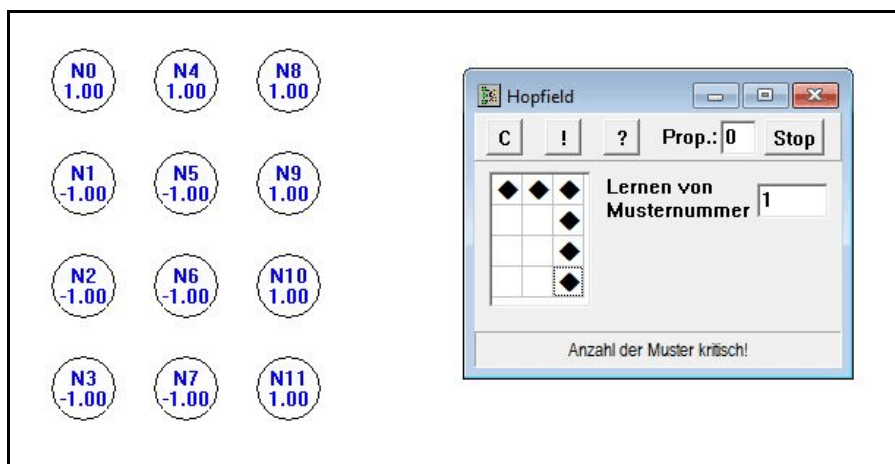


Abb. 24

Hopfield-Netze

Alle Neuronen dienen hier sowohl zur Eingabe als auch zur Ausgabe. Ein schwarzes Pixel stellen wir durch ein aktives Neuron (d. h. mit dem Aktivierungsgrad +1) dar, ein weißes (nicht gesetztes) Pixel durch ein nicht aktiviertes Neuron (d. h. mit dem Aktivierungsgrad -1). Als Aktivierungsfunktion benutzen wir dementsprechend

$$f_{\text{Hopfield}}(i) = \begin{cases} +1 & \text{für } i \geq 0 \\ -1 & \text{sonst} \end{cases} .$$

Nach Hopfield besteht der Vorgang der Mustereinprägung darin, die Gewichte nun so einzustellen, dass das Netz sich beim gelernten Muster stabilisiert, wenn es - ausgehend von einem Musterfragment - schrittweise propagiert wird. Dies soll nun an einem einfachen Beispiel genauer untersucht werden.

3. Muster einprägen und rekonstruieren

Als Beispiel benutzen wir das Hopfield-Netz aus Abb. 25. Ihm wurde das rechts daneben dargestellte Muster eingepägt. Dazu wurden die Gewichte gemäß der **Hebbschen Regel** bestimmt:

- $g_{ij} = +1$, wenn Neuron i und Neuron j denselben Pixelwert haben,
- $g_{ij} = -1$, wenn Neuron i und Neuron j einen unterschiedlichen Pixelwert haben,
- $g_{i,i} = 0$ (keine Verbindung).

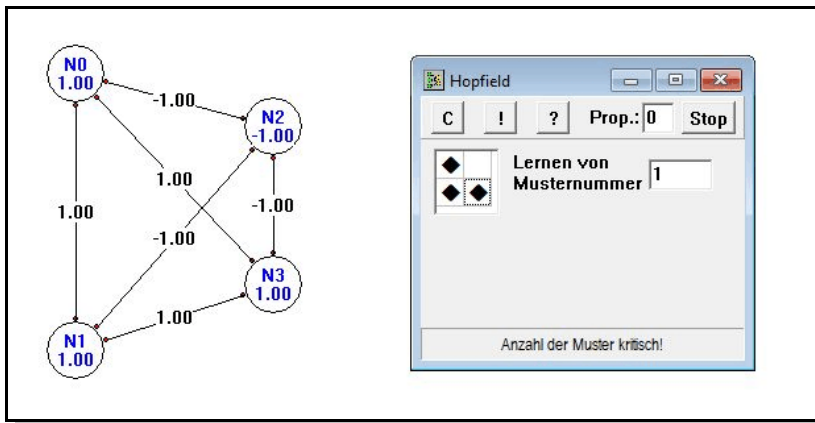


Abb. 25

In unserem Fall sieht die Gewichtsmatrix G dann so aus: $G = \begin{pmatrix} 0 & +1 & -1 & +1 \\ +1 & 0 & -1 & +1 \\ -1 & -1 & 0 & -1 \\ +1 & +1 & -1 & 0 \end{pmatrix}$

Hopfield-Netze

Die Matrix ist wegen der Hebbschen Regel symmetrisch. Die Hauptdiagonalelemente sind alle 0.

Würde man bei dem Netz aus Abb. 25 eine Propagation durchführen, würde es sich nicht ändern: Es stellt einen stabilen Zustand dar. Davon kann man sich leicht durch eine einfache Rechnung überzeugen.

Was geschieht nun, wenn dem trainierten Netz ein anderes Muster präsentiert wird, z. B. das Fragment-Muster aus Abb. 26?



Abb. 26

Wir berechnen einmal den neuen Zustand von Neuron 0:

$$f(g_{1,0} \cdot a_1 + g_{2,0} \cdot a_1 + g_{3,0} \cdot a_3) = f((+1) \cdot (+1)) + (-1) \cdot (-1) + (+1) \cdot (-1) = f(+1) = +1$$

Das Neuron N0 hat nach der Propagation also denselben Zustand wie vorher. Gleiches gilt für das Neuron N1.


Anders verhält es sich beim Neuron N3: Die zugehörige Rechnung für den neuen Zustand ist:

$$f(g_{0,3} \cdot a_0 + g_{1,3} \cdot a_1 + g_{2,3} \cdot a_2) = f((+1) \cdot (+1)) + (+1) \cdot (+1) + (-1) \cdot (-1) = f(+3) = +1$$

Das Neuron ändert seinen Zustand von -1 auf +1; aus dem weißen Pixel wird ein schwarzes. Damit ist der stabile trainierte Zustand wieder hergestellt.

Unser Hopfield-Netz kann weitere Muster lernen. Die Vorgehensweise ist einfach: Zu dem neuen Muster wird wie oben gezeigt eine weitere Gewichtsmatrix bestimmt. Die erste und die zweite Gewichtsmatrix werden addiert: Die Summenmatrix $G = G_1 + G_2$ kann sowohl Fragmente des ersten als auch des zweiten Musters rekonstruieren. Das schauen wir uns der Einfachheit halber mit unserem Programm **Netz32** an.

4. Hopfield-Netze mit **Netz32**

Das Netz aus Abb. 25 erzeugen wir, indem wir in der Werkzeugleiste die Schaltfläche <speziellen Netztyp wählen> anklicken. In dem Fenster, das sich öffnet, wählen wir die Option <rückgekoppelte Netze> und geben sowohl für die Zahl der horizontalen als auch der vertikalen Pixel jeweils die Zahl 2 ein und bestätigen dies mit <OK>. Nun lassen wir das Hopfield-Netz im Netzfenster anzeigen und öffnen dann mit der Schaltfläche  das **Hopfield-Fenster** zum Einprägen und Rekonstruieren von Mustern. Hier geben wir als erstes das Muster aus Abb. 25 ein, indem wir die entsprechenden Felder mit der Maus anklicken; ein erneutes Anklicken löscht das gesetzte Pixel. Mit der C-Schaltfläche lässt sich das gesamte Muster löschen.

Hopfield-Netze

Damit unser Netz sich dieses Muster einprägt, betätigen wir nun die !-Schaltfläche. Wer mag, kann sich die gelernten Gewichte anschauen, indem er im Anzeige-Menü die Darstellung der Gewichte und Verbindungspfeile aktiviert.

Gleich anschließend geben wir das Muster aus Abb. 27 ein und betätigen noch einmal die !-Schaltfläche. Auf die Bemerkung “Anzahl der Muster kritisch!” am unteren Rand des Hopfield-Fensters werden wir weiter unten noch zu sprechen kommen.

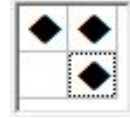


Abb. 27

Nun präsentieren wir Muster-Fragmente. Zunächst aktivieren wir nur die beiden linken Pixel und betätigen die ?-Schaltfläche. Schon nach einem einzigen Schritt ist das erste Muster rekonstruiert; die weiteren Schritte führen zu keiner Änderung - das Netz stellt jetzt einen stabilen Zustand dar. Gleiches stellt man fest, wenn als Fragment die unteren beiden Pixel aktiviert werden.

Nun präsentieren wir dem Netz auch auf ähnliche Weise Fragmente, und zwar werden einmal die beiden oberen Pixel aktiviert und einmal die beiden rechten. Diesmal wird unser zweites Muster aus Abb. 27 rekonstruiert. Bei anderen Fragmenten können Probleme auftreten:

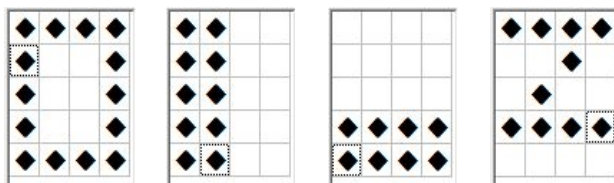
- Ein gespeichertes Muster wird nicht erkannt.
- Ein zu einem eingprägten Muster inverses Muster wird erkannt.

Generell gilt:

- Je mehr Neuronen das Netz besitzt, desto mehr Muster können gelernt werden. Für die Anzahl N_{Muster} der Muster, die in einem Hopfield-Netz recht zuverlässig gespeichert werden können, gilt die Faustregel $N_{Muster} = N_{Neuronen} \cdot 0,14$.
- Die Muster sollten in etwa gleich viele weiße wie schwarze Pixel beinhalten.
- Die Muster sollten sich nicht zu ähnlich sein.

5. Experimente mit 4×5-Mustern

Zunächst erstellen wir ein 4×5-Hopfield-Netz und lassen es die 4 Muster aus Abb. 28 lernen.



Muster 1

Muster 2

Muster 3

Muster 4

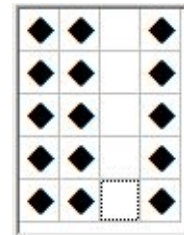
Abb. 28

Hopfield-Netze

Da die Anzahl der Muster hier etwas größer als $4 \cdot 5 \cdot 0,14 = 2,8$ ist, wird unser Netz nicht unbedingt sehr zuverlässig arbeiten. Um so interessanter ist es, die Grenzen auszuloten.

Man stellt fest:

- Viele Fragmente werden zuverlässig in einem einzigen Schritt erkannt; dies gilt insbesondere, wenn sie dem Original ähneln und aus genügend vielen (aktivierten) Pixeln bestehen. Man beachte, dass ein "Fragment" durchaus auch einige aktivierte Pixel mehr haben kann als das Original.
- Bei manchen Fragmenten sind für die Rekonstruktion zwei oder mehr Schritte nötig. Beispiele für solche Fragmente sind in Abb. 29 angegeben.



Muster 1

Muster 2

Abb. 29