Neuronale Netze
Eine Einführung

Thomas Friedrich

10. Juni 1998

Seminar über nichtlineare Approximationsmethoden

Universität zu Köln

1 Einleitung

In diesem Vortrag soll durch eine Einführung ein Überblick über das Gebiet der Neuronalen Netze gegeben werden.

Nach einer kurzen Vorstellung der Grundideen erfolgt eine Beschreibung der Komponenten Neuronaler Netze.
Es werden dann verschiedene Lernverfahren behandelt, wobei Schwerpunkte auf der klassischen Backpropagation-Methode und auf einer speziellen Methode für exaktes Lernen in zweistufigen Feedforward-Netzen liegen.
Es schließt sich eine Betrachtung von Varianten Neuronaler Netze an mit besonderem Augenmerk auf Time Delay-Netzen, wonach einige ausgewählte Anwendungen Neuronaler Netze folgen.

Eine kurze Abhandlung über Möglichkeiten der Verkleinerung Neuronaler Netze beschließt diesen Vortrag.

1.1 Biologische Motivation

Das Konzept der Neuronalen Netze hat seinen Ursprung in der Biologie. Vorbild ist hier der Netzverbund von Nervenzellen (Neuronen), die über elektrische Impulse Informationen austauschen. Dabei „sammelt“ jede Zelle die Ausgaben ihrer Vorgänger im Netzwerk über verschieden stark ausgebildete Verbindungen, und wenn dadurch ein bestimmter Schwellenwert überschritten wird, sendet die Zelle ihrerseits einen Impuls. Dieser kann auf die nachfolgenden Zellen anregend (exzitatorisch), aber auch hemmend (inhibitorisch) wirken.

Das System „lernt“, indem die Stärken der Verbindungen zwischen den einzelnen Zellen den erforderlichen Aufgaben angepaßt werden.

1.2 Mathematische Modellierung

Die Mathematik abstrahiert den Begriff der Neuronalen Netze stark auf einen gerichteten Graphen, bei dem die Knoten die Neuronen und die gewichteten Kanten die Verbindungen zwischen ihnen darstellen sollen.

Lernmethoden verändern dabei meistens in bestimmter Weise die Gewichte auf den Kanten.

Den Bezug der (mathematischen) Neuronalen Netze zur Biologie sollte man daher nicht überbewerten. Mathematische Modelle (auch *konnektionistische Modelle* genannt) haben teilweise biologisch nicht motivierte Entwicklungen vollzogen.

Deshalb soll hier auch nicht der Eindruck erweckt werden, man könnte die Funktion des menschlichen Gehirns nachbilden. Dazu reicht einerseits die bisher erreichte Rechengeschwindigkeit auch schnellster Computer bei weitem nicht aus, andererseits ist der Stand der medizinischen Forschung nicht auf einem entsprechenden Stand, so daß viele Vorgänge in der Hauptschaltzentrale des Menschen noch unbekannt sind.

2 Beschreibung konnektionistischer Modelle

2.1 Aufbau Neuronaler Netze

Neuronale Netze bestehen aus *Neuronen* (auch *Zellen* genannt), die als Knoten eines Graphen aufgefaßt werden können. Zwischen ihnen verlaufen gewichtete Verbindungen (Kanten). Die Kantengewichte sind nicht fest; sie können verändert (*trainiert*) werden. Zwei Neuronen müssen nicht unbedingt durch eine Kante verbunden sein. Die Verbindung von Neuron i zu Neuron j wird mit w_{ij} bezeichnet. Die gesamte Gewichtsmatrix ist dann $W = (w_{ij})$.

Zellen, die einen Input von außen erhalten, heißen *Eingabeneuronen*. Sie geben diesen unverändert an das System weiter. Die Zellen, die den Output des Netzes liefern, heißen *Ausgabeneuronen*.

Jedes Neuron (außer den Eingabezellen) sammelt die über die eingehenden Kanten eintreffenden Informationen. Auf diese *Netzeingabe* wendet die Zelle ihre *Aktivierungsfunktion* f_{act} an und gibt diesen Wert an nachfolgende Zellen weiter. Evtl. wird darauf vorher noch eine *Ausgabefunktion* angewendet. Da die Aktivierungsfunktion aber i. a. schon nichtlinear ist, wird als Ausgabefunktion meist die Identität gewählt. Die Ausgabe von Neuron i wird mit o_i bezeichnet.

Die Neuronen, die weder Ein- noch Ausgabezellen sind, also innere Knoten des Netzes, heißen *verdeckte* Neuronen. Neuronen werden häufig in *Schichten* angeordnet, wobei die Neuronen einer Schicht ihre Ausgaben an die Neuronen der nächsten Schicht weitergeben.

Die Schicht der Eingabeneuronen heißt *Eingabeschicht*, die der Ausgabeneuronen *Ausgabeschicht*. Alle anderen sind *verdeckte Schichten*.

Ein Neuronales Netz heißt *n-stufig*, wenn es n Schichten trainierbarer Verbindungen besitzt. Es werden also nicht die Schichten von Neuronen gezählt, da die Leistungsfähigkeit eines Systems i. a. von den gewichteten Verbindungen und nicht von den Zellen selbst abhängt. Im Bild ist folglich ein dreistufiges Neuronales Netz dargestellt.

Lernverfahren verändern die Gewichte auf den Kanten derart, daß die Ausgabe des Netzes mit der gewünschten Ausgabe möglichst übereinstimmt.

2.2 Die Netzeingabe

Jedes Neuron (außer den Eingabezellen) erhält seine Informationen aus dem Netz über die sog. *Netzeingabe*. Dies ist meist die gewichtete Summe der Informationen, die über die eingehenden Kanten eintrifft. Dazu werden Kanten Information mit dem entsprechenden Kantengewicht bewertet (multipliziert) und danach aufsummiert. Es ist also $\mathrm{net}_j = \sum\limits_i o_i w_{ij}$.

2.3 Schwellenwert

Beim einfachsten Modell Neuronaler Netze ist die Aktivierungsfunktion eine binäre Schwellenwertfunktion: erreicht die Netzeingabe, also die gewichtete Summe der eintreffenden Informationen, mindestens einen für das Neuron festgelegten Schwellenwert, so ist die Ausgabe 1, sonst 0.

Der Schwellenwert muß für jede Zelle getrennt gespeichert werden.

On-Neuron

Es ist aber auch möglich, die Speicherung der Schwellenwerte als Kantengewichte vorzunehmen. Dazu wird ein zusätzliches Neuron, das *On-Neuron*, hinzufügt, das immer die Ausgabe 1 liefert. Dieses On-Neuron ist mit allen Zellen des Netzes außer denen der Eingabeschicht verbunden, wobei das Gewicht einer Kante genau dem negativen Schwellenwert der Zelle entspricht.

Damit wird bei der Netzeingabe eines Neurons immer der Schwellenwert abgezogen, und es erzeugt genau dann eine Ausgabe, wenn die Eingabe ≥ 0 ist.

2.4 Aktivierungsfunktionen

Die binäre Schwellenwertfunktion mit ihren Werten 0 und 1 hat den Nachteil, daß eine Zelle für ihre Nachfolger höchstens exzitatorisch (anregend) wirken kann, nicht jedoch inhibitorisch (hemmend). Dies kann dadurch behoben werden, daß der Wertebereich von $\{0, 1\}$ auf $\{-1, 1\}$ gesetzt wird.

Ferner ist es unangenehm, daß die Reaktion des Neurons so „plötzlich" erfolgt. Für einige Lernverfahren ist es außerdem erfordelich, daß die Aktivierungsfunktion differenzierbar ist. (Die binäre Schwellenwertfunktion ist an der Stelle des Schwellenwertes nicht einmal stetig.)

Daher wird häufig die *logistische* Aktivierungsfunktion

$$f_{\text{act}}(x) = \frac{1}{1 + \mathrm{e}^{-\lambda x}}$$

verwendet. Sie ist überall differenzierbar, und durch Variation des Parameters λ kann die Steigung und damit die Trägheit des Neurons beeinflußt werden. Es ist durch Wahl großer Werte von λ sogar möglich, die binäre Schwellenwertfunktion beliebig genau zu approximieren.

Für die meisten Anwendungen genügt $\lambda = 1$.

Solche S-förmigen Funktionen heißen *sigmoid*. Theoretisch ist jede *semilineare*, d. h. monoton steigende, differenzierbare, aber i. a. nicht-lineare, Funktion als Aktivierungsfunktion geeignet.

Weitere Möglichkeiten von Aktivierungsfunktionen zeigt das Bild.

2.5 Netztopologien

Man unterscheidet zwei grundsätzliche Typen von Netztopologien: Netze *ohne* Rückkopplungen (*Feedforward*-Netze) und solche *mit* Rückkopplungen (*rekurrente* Netze).

Feedforward-Netze sind kreisfrei, d. h. es gibt keinen Pfad, der von einem Neuron über evtl. andere Neuronen wieder zu dem Neuron zurückführt. Wenn eine Verbindung zwischen zwei Zellen eine oder mehrere Ebenen „überspringt“, so nennt man diese *Shortcut* (Abkürzung).

Am häufigsten anzutreffen sind ebenenweise vollständig verbundene Feedforward-Netze ohne Shortcuts. Auch in diesem Vortrag wird meist mit solchen Netzen gearbeitet.

Bei rekurrenten Netzen unterscheidet man zwischen *direkten* und *indirekten* Rückkopplungen. Direkte Rückverbindungen führen wieder zu der Zelle selbst hin, während es bei indirekten nur möglich ist, über zwischengeschaltete andere Neuronen wieder zurückzugelangen.
Mit rekurrenten Netzen beschäftigt sich Kap. 4.

Im Bild ist neben einer schematischen Darstellung der Netztopologie auch jeweils die Struktur der Gewichtsmatrix W eingetragen.

2.6 Beispiele

Die folgenden Beispiele sollen zur Illustration der Funktionsweise Neuronaler Netze dienen.

2.7 Beispiel: Die XOR-Funktion

Die Funktion XOR: $\{0,1\}^2 \longrightarrow \{0,1\}$ ist genau dann Eins, wenn die beiden Eingabebits verschieden sind, sonst Null.
Als Aktivierungsfunktion wird eine binäre (0-1-)Schwellenwertfunktion verwendet. Ein Neuronales Netz, das die XOR-Funktion berechnet, sieht z. B. so aus:

Die Gewichte sind neben der jeweiligen Verbindung, die Schwellenwerte im jeweiligen Neuron eingetragen. (Die Eingabeneuronen besitzen keine Schwellenwerte.) Das einzige Ausgabeneuron liefert das korrekte Ergebnis.

2.8 Beispiel: Das Perzeptron

Der Begriff des Perzeptrons ist in der Literatur nicht eindeutig. Das hier betrachtete Modell besteht aus einer Schicht Neuronen, die über feste Gewichte mit der Eingabeschicht verbunden ist. Diese ist über variabel gewichtete Verbindungen mit einem einzigen Ausgabeneuron verbunden.

Die Neuronen in der Verarbeitungsschicht und das Ausgabeneuron wenden auf die gewichtete Summe der Eingaben eine binäre Schwellenwertfunktion an.
(Ein Perzeptron kann auch mehrere Ausgabeneuronen besitzen, ohne daß sich an seinem Prinzip etwas ändert.)

Besteht die Eingabe aus n Daten und faßt man diese als n-dimensionalen Raum auf, so ist ein einstufiges Perzeptron (d. h. mit einer einzigen Verarbeitungsschicht) dazu in der Lage, diesen Raum linear zu separieren.

Für die oben betrachtete XOR-Funktion reicht dies aber nicht aus: es ist nicht möglich, im zweidimensionalen Raum die Punkte $A_0 := (0,0)$ und $A_1 := (1,1)$ von den Punkten $B_0 := (0,1)$ und $B_1 := (1,0)$ durch eine Gerade zu trennen.
Deswegen war es notwendig, die Eingabeneuronen duch Shortcuts direkt mit der Ausgabezelle zu verbinden.

Zweistufige Perzeptrons (d. h. mit zwei Verarbeitungsschichten) können durch Kombination (Schnitt) von Halbräumen, die die erste Schicht getrennt hat, bereits Polytope darstellen.

Durch Hinzufügen einer dritten Verarbeitungsschicht kann ein Perzeptron auch komplexere Mengen, die nicht zusammenhängend oder konvex sind, separieren (im Bild dunkel schraffiert dargestellt).

Ein Theorem von Frank Rosenblatt besagt: Das Perzeptron kann in endlicher Zeit alles lernen, was es repräsentieren kann. Diese sehr stark klingende Aussage wird durch die geringe Repräsentierbarkeit eines Perzeptrons wieder relativiert.

3 Lernverfahren

Der interessanteste Aspekt bei der Betrachtung Neuronaler Netze ist das Lernen.

3.1 Arten des Lernens

Lernen in Neuronalen Netzen kann auf verschiedene Arten erfolgen.
Einmal gibt es mehrere Möglichkeiten, *was* verändert wird:

- Einfügen neuer Verbindungen (Kanten)
- Löschen existierender Verbindungen (Kanten)
- Änderung der Stärke von Verbindungen (Kantengewichten)
- Änderung des Schwellenwertes eines Neurons (Knotens)
- Änderung der Netzeingabe-, Aktivierungs- oder Ausgabefunktion
- Einfügen neuer Zellen (Knoten)

- Löschen existierender Zellen (Knoten).

Die am häufigsten verwendete Methode ist das Verändern der Stärke von Verbindungen. Einfügen und Löschen von Verbindungen sind auf diese Weise ebenfalls realisierbar.
Die Änderung des Schwellenwertes ist im wesentlichen auch nichts anderes als die Änderung von Verbindungen (vgl. Kap. 2.3: On-Neuron), und daher problemlos möglich.

Desweiteren klassifiziert man die Art des Lernens in:

Überwachtes Lernen: Beim überwachten Lernen werden dem System nacheinander die Eingabe und die erwünschte Ausgabe präsentiert. Das heißt, daß das Netz nach Berechnung seiner eigenen Ausgabe immer weiß, wie groß sein Fehler war und es sich entsprechend korrigieren kann.

Bestärkendes Lernen: Hierbei wird dem System lediglich mitgeteilt, ob die von ihm berechnete Ausgabe richtig oder falsch ist. Diese Art des Lernens ist biologisch stärker motiviert als überwachtes Lernen, muß i. a. aber intensiver trainiert werden.

Unüberwachtes Lernen: Jetzt ist das System auf sich selbst gestellt. Es muß seine Fehler selbständig erkennen und korrigieren können. Im mathematischen Modell ist diese Art des Lernens weniger verbreitet, da sie sehr zeitintensiv ist. Außerdem ist sie für viele Fragestellungen nicht geeignet.

Schließlich unterscheidet man noch:

Offline-Lernen: Die Gewichte auf den Kanten werden jeweils erst nach Präsentation *aller* Eingabemuster verändert.

Online-Lernen: Hier wird nach *jedem* Eingabemuster reagiert, und die Gewichte werden entsprechend angepaßt.

3.2 Gradientenabstiegsverfahren

Trägt man den Fehler eines Neuronalen Netzes bezüglich der Gewichte graphisch auf, erhält man eine Fehlerfläche, die im zweidimensionalen Fall noch recht anschaulich darstellbar ist:

Die Idee von Gradientenabstiegsverfahren besteht darin, die Gewichte so zu korrigieren, daß der Fehler möglichst stark minimiert wird, d. h. daß die Fehlerfläche an der entsprechenden Stelle ein Minimum aufweist.

Da der Gradient (grad) einer Funktion den steilsten Anstieg angibt, erfolgt somit der steilste Abstieg längs des negativen Gradienten der Fehlerfunktion

$$\mathrm{E}(W) = \mathrm{E}(w_1, \ldots, w_n).$$

(Fasse die Matrix W dabei als „langen“ Vektor auf.)

Dieser wird noch mit einer „Lernrate“ η multipliziert, die angibt, wie groß die Schrittweite sein soll. Es ergibt sich also:

$$\Delta W = -\eta \cdot \mathrm{grad}(\mathrm{E}(W))$$

bzw. für ein einzelnes Element von W:

$$\Delta w_{ij} = -\eta \cdot \frac{\partial \mathrm{E}(W)}{\partial w_{ij}}.$$

Der Gesamtfehler E setzt sich aus der Summe der Fehler der einzelnen Eingabemuster zusammen, also $\mathrm{E} = \sum_p \mathrm{E}_p$, wobei man üblicherweise

$$\mathrm{E}_p = \frac{1}{2} \cdot \sum_j (t_{p_j} - o_{p_j})^2$$

setzt. Dies ist die Hälfte des Quadrates des euklidischen Abstandes zwischen erwünschter und tatsächlicher Ausgabe. Der Faktor 1/2 kürzt sich später mit einer durch Differenzieren entstehenden 2 weg.

Die Δ-Regel

Zur Vereinfachung setzen wir hier eine lineare Aktivierungsfunktion voraus. Es ergibt sich:

$$\Delta w_{ij} = -\eta \cdot \frac{\partial \mathrm{E}(W)}{\partial w_{ij}} = \sum_p -\eta \cdot \frac{\partial \mathrm{E}_p}{\partial w_{ij}},$$

wobei nach Kettenregel

$$\frac{\partial \mathrm{E}_p}{\partial w_{ij}} = \frac{\partial \mathrm{E}_p}{\partial o_{pj}} \cdot \frac{\partial o_{pj}}{\partial w_{ij}}$$

gilt. Der erste Faktor hiervon ist

$$\frac{\partial \mathrm{E}_p}{\partial o_{pj}} = -(t_{pj} - o_{pj}) =: -\delta_{pj}$$

und der zweite (wegen Linearität)

$$\frac{\partial o_{pj}}{\partial w_{ij}} = \frac{\partial}{\partial w_{ij}} \sum_k o_{pk} w_{kj} = o_{pi}.$$

Somit ist

$$\frac{\partial \mathrm{E}_p}{\partial w_{ij}} = -o_{pi} \delta_{pj}.$$

Damit ergibt sich die Modifikationsregel

$$\Delta w_{ij} = \eta \cdot \sum_p o_{pi} \delta_{pj}.$$

Dies ist wegen dem Bezug auf *alle* Eingabemuster ein Offline-Verfahren. Die entsprechende die Online-Variante lautet:

$$\Delta_p w_{ij} = \eta \cdot o_{pi} \delta_{pj} = \eta \cdot o_{pi} \cdot (t_{pj} - o_{pj}).$$

Man nennt dieses Lernverfahren *Delta-* (Δ-) oder *Widrow-Hoff-Regel*.

3.3 Backpropagation

Die Δ-Regel eignet sich so nur für einstufige Netze, da bei mehreren Stufen die erwünschte Ausgabe der *verdeckten* Neuronen nicht bekannt ist.

Daher soll die Δ-Regel nun für mehrschichtige Netze verallgemeinert werden.

Die Netzeingabe von Neuron j bei Eingabemuster p ist die gewichtete Summe $\mathrm{net}_{pj} = \sum_i o_{pi} w_{ij}$. Wenn f die (semilineare) Aktivierungsfunktion bezeichnet, ist die Ausgabe von Zelle j

$$o_j = f(\mathrm{net}_{pj}).$$

Analog zur Δ-Regel gilt

$$\Delta w_{ij} = \sum_p -\eta \cdot \frac{\partial \mathrm{E}_p}{\partial w_{ij}}$$

und nach Kettenregel

$$\frac{\partial \mathrm{E}_p}{\partial w_{ij}} = \frac{\partial \mathrm{E}_p}{\partial \mathrm{net}_{pj}} \cdot \frac{\partial \mathrm{net}_{pj}}{\partial w_{ij}}.$$

Der zweite Faktor ist

$$\frac{\partial \mathrm{net}_{pj}}{\partial w_{ij}} = \frac{\partial}{\partial w_{ij}} \sum_i o_{pi} w_{ij} = o_{pi}.$$

Definiert man nun

$$\delta_{pj} := -\frac{\partial \mathrm{E}_p}{\partial \mathrm{net}_{pj}},$$

so ergibt sich

$$\Delta w_{ij} = \eta \cdot \sum_p o_{pi} \delta_{pj},$$

genau wie bei der Δ-Regel in der Offline-Version, nur daß die δ_{ij} komplizierter definiert sind.
Wiederum nach Kettenregel ist

$$\delta_{pj} = -\frac{\partial \mathrm{E}_p}{\partial \mathrm{net}_{pj}} = -\frac{\partial \mathrm{E}_p}{\partial o_{pj}} \cdot \frac{\partial o_{pj}}{\partial \mathrm{net}_{pj}},$$

wobei für den zweiten Faktor gilt

$$\frac{\partial o_{pj}}{\partial \mathrm{net}_{pj}} = \frac{\partial f(\mathrm{net}_{pj})}{\partial \mathrm{net}_{pj}} = f'(\mathrm{net}_{pj}),$$

d. h. die erste Ableitung der Aktivierungsfunktion.

Analog zur Δ-Regel ergibt sich für die Online-Version

$$\Delta_p w_{ij} = \eta \cdot o_{pi} \delta_{pj}.$$

Somit fehlt nur noch die Bestimmung von $-\frac{\partial \mathrm{E}_p}{\partial o_{pj}}$. Hier lassen sich zwei Fälle unterscheiden:

1. j ist eine Ausgabezelle
2. j ist eine verdeckte Zelle.

Im ersten Fall erhält man ohne Probleme

$$-\frac{\partial \mathrm{E}_p}{\partial o_{pj}} = t_{pj} - o_{pj}$$

analog zur Δ-Regel.

Der zweite Fall ist schon komplizierter, weil die partielle Ableitung $-\frac{\partial \mathrm{E}_p}{\partial o_{pj}}$ nur indirekt berechnet werden kann; nach Kettenregel und bereits Bekanntem gilt:

$$-\frac{\partial \mathrm{E}_p}{\partial o_{pj}} = -\sum_k \frac{\partial \mathrm{E}_p}{\partial \mathrm{net}_{pk}} \cdot \frac{\partial \mathrm{net}_{pk}}{\partial o_{pj}} = \sum_k (\delta_{pk} \cdot \frac{\partial}{\partial o_{pj}} \sum_i o_{pi} w_{ik}) = \sum_k \delta_{pk} w_{jk}.$$

Das bedeutet, daß die Korrektur des Fehlers einer Zelle mit Hilfe der (gewichteten) zurückgemeldeten Fehler der nachfolgenden Schichten erfolgt, der Gesamtfehler also durch das Netz zurückpropagiert wird – daher auch der Name dieses Lernverfahrens: *Backpropagation*.

Im ersten Fall ist also

$$\delta_{pj} = f'(\mathrm{net}_{pj}) \cdot (t_{pj} - o_{pj}),$$

im zweiten Fall

$$\delta_{pj} = f'(\mathrm{net}_{pj}) \cdot \sum_k \delta_{pk} w_{jk}.$$

Zusammengefaßt ergibt sich mit diesen δ_{pj} die Online-Modifikationsregel

$$\Delta_p w_{ij} = \eta \cdot o_{pi} \delta_{pj}.$$

Speziell für die logistische Aktivierungsfunktion ergibt sich wegen

$$f'(\text{net}_{pj}) = f(\text{net}_{pj}) \cdot (1 - f(\text{net}_{pj})) = o_{pj} \cdot (1 - o_{pj})$$

das δ_{pj} als

$$\delta_{pj} = \begin{cases} o_{pj} \cdot (1 - o_{pj}) \cdot (t_{pj} - o_{pj}), \text{ falls j Ausgabezelle} \\ o_{pj} \cdot (1 - o_{pj}) \cdot \sum_k \delta_{pk} w_{jk}, \text{ falls j verdeckte Zelle.} \end{cases}$$

Probleme von Backpropagation

Backpropagation hat aber auch einige Probleme; nicht zuletzt, weil es als Gradientenverfahren nur lokale Informationen über den Fehler besitzt.

So kann Backpropagation in einem *lokalen* Minimum hängenbleiben, da hier kein Abstieg auf der Fehlerfläche mehr möglich ist (a).

Bei ungünstiger Wahl der Startgewichte (z. B. alle gleich) ergibt sich in ebenenweise vollständig verbundenen Feedforward-Netzen eine nicht gewollte Symmetrie. Dies kann aber leicht durch zufälliges „Verrauschen“ der Werte behoben werden.

Weiterhin gibt es das Problem, daß sich Backpropagation auf flachen Plateaus der Fehlerfläche wegen dem geringen Betrag des Gradienten nur sehr langsam bewegt; dadurch werden extrem viele Iterationsschritte nötig (b).

Im Gegensatz dazu kann bei zu großer Schrittweite ein tiefes Tal übersehen werden (d). Außerdem kann es vorkommen, daß das Verfahren in tiefen Tälern (und damit mit großen Korrekturtermen) oszilliert (c).

3.4 Modifikationen von Backpropagation

Gegen einige dieser Probleme ist Backpropagation zu den folgenden Lernverfahren erweitert (verbessert) worden. I. a. „erkauft“ man sich den behobenen Defekt durch höhere Rechenzeiten. Es ist also vorher für jede Anwendung individuell auszuloten, für welche Methode man sich entscheidet.

Momentum-Term

Bei Backpropagation mit Momentum-Term wird die aktuelle Gewichtsänderung von der jeweils vorhergehenden abhängig gemacht. Es ergibt sich die Modifikationsregel

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

wobei α üblicherweise zwischen 0.2 und 0.99 gewählt wird. Dieses Verfahren bewirkt durch Erhöhung der Gewichtsänderung eine Beschleunigung auf flachen Plateaus.

Flat-Spot Elimination

In weit außerhalb liegenden Bereichen haben sigmoide Aktivierungsfunktionen eine nur sehr geringe Steigung, so daß die Gewichtsänderungen klein ausfallen. Solche Bereiche werden *flat spots* genannt. Um dem entgegenzuwirken, kann man zu der Ableitung jeweils eine Konstante hinzuaddieren, so daß sie einen Mindestbetrag nicht unterschreitet.

Weight Decay

Ein weiterer Nachteil besteht darin, daß große Gewichte einen zu großen Einfluß haben. Dazu ist es möglich, zu groß gewordene Gewichte dadurch zu bestrafen, daß ein bestimmter Anteil des Quadrates des Gewichtes zum Fehler hinzuaddiert wird. (Quadratisch deshalb, damit es sich auf größere Gewichte viel stärker auswirkt.)

Manhattan-Training

Hierbei wird die Aufmerksamkeit mehr auf das Vorzeichen als auf den Betrag des Fehlers gelegt. Dadurch wird erreicht, daß die Schritte auf der Fehlerfläche mit nahezu gleicher Länge erfolgen. Dies ist schneller zu berechnen als eine „echte“ Normierung. Man erhält dadurch besseres Verhalten auf flachen Plateaus.

SuperSAB

Bei dieser Methode wird für jedes Gewicht (d. h. in jeder Richtung auf der Fehlerfläche) eine eigene (variable) Schrittweite verwendet, die dem Fehler entsprechend angepaßt wird.
SuperSAB ist sehr rechenzeit- und speicherintensiv.

Second-Order Backpropagation

Man hat auch auf verschiedene Weisen die Verwendung der zweiten Ableitung für die Gewichtsänderungen untersucht. Man erhält i. a. eine schnellere Konvergenz (weniger Iterationen). Noch höhere Ableitungen brachten keine nennenswerten Verbesserungen mehr.

Quickprop

Quickprop orientiert sich bei der Bestimmung des minimalen Fehlers am Newton-Verfahren. Dabei wird die Fehlerfunktion als lokal quadratisch angenommen und der nächste Schritt zum Scheitel dieser Parabel hin unternommen (wenn diese nach *oben* geöffnet ist).

Backpercolation

Beim Backpercolation ist es durch spezielle Betrachtung des Fehlers jedes einzelnen Neurons möglich, daß lokale Minima auf der Fehlerfläche auch wieder verlassen werden können. Es werden sozusagen „Berge durchgetunnelt“ (woher auch der Name stammt).

Kombinationen von Verfahren

Obige Verbesserungen von Backpropagation lassen sich auch miteinander kombinieren. Dabei sollte man aber verstärkt auf die Rechenzeit achten: es kann sinnvoller sein, mehr Iterationen durchzuführen, die dafür nicht so rechenintensiv sind.
Außerdem lohnt sich z. B. eine Verbesserung für flache Plateaus nicht, wenn die Fehlerfläche keine solchen aufweist.

3.5 Exaktes Lernen in Feedforward-Netzen

Zum Abschluß dieses Kapitels soll noch ein Lernverfahren vorgestellt werden, das exaktes Lernen in zweistufigen Feedforward-Netzen ermöglicht.

Normalerweise steigt die Fähigkeit des genauen Lernens der Eingabemuster eines solchen Netzes mit der Anzahl der Neuronen in der verdeckten Schicht. Wenn n diese Anzahl und N die Zahl der Eingabemuster bezeichnet, ist dies bei $n \geq N$ kein Problem.

Backpropagation (und andere Gradientenverfahren) zeigen aber ein entgegengesetztes Verhalten: die Konvergenzgeschwindigkeit läßt mit zunehmender Zahl innerer Zellen nach; außerdem erhöht sich die Wahrscheinlichkeit, in lokalen Minima hängenzubleiben.

Zunächst betrachten wir den Fall mit nur einer einzigen Ausgabezelle. Das Netzwerk habe m Eingabezellen, n Zellen in der verdeckten Schicht und eine Ausgabezelle. Die Gewichte der Verbindungen zwischen Eingabe- und verdeckter Schicht seien mit (v_{kj}), die Gewichte zwischen verdeckter und Ausgabeschicht mit (w_j) bezeichnet. Es sollen N Muster $\{(i_{pk}), t_p\}$ gelernt werden.

Die Aktivierungsfunktion σ sei streng monoton, so daß σ^{-1} existiert.
Definiere $s_p := \sigma^{-1}(t_p)$.

Ferner sei die Matrix $\Sigma := (\sigma_{lj})$, wobei $\sigma_{lj} := \sigma\left(\sum_{k=1}^{m} i_{lk}v_{kj}\right)$ die Ausgabe von verdecktem Neuron j bei Anlegen von Beispiel i bezeichne.

Ein exaktes Lernen der Eingabemuster ist möglich, wenn es eine Lösung $W = (w_j)$ für

$$\begin{pmatrix} \sigma_{11} & \dots & \sigma_{1n} \\ \vdots & & \vdots \\ \sigma_{N1} & \dots & \sigma_{Nn} \end{pmatrix} \cdot \begin{pmatrix} w_1 \\ \vdots \\ w_n \end{pmatrix} = \begin{pmatrix} s_1 \\ \vdots \\ s_N \end{pmatrix}$$

oder kurz

$$\Sigma \cdot W = S$$

gibt. Dafür muß der Vektor S in dem linearen Unterraum liegen, der von den Spalten von Σ aufgespannt wird. Die j-te Spalte von Σ sei mit σ_j bezeichnet. Wenn σ_1 bereits zu S linear abhängig gemacht werden könnte, wäre diese Bedingung mit Sicherheit erfüllt. I. a. wird man aber nur „nahe herankommen". Man versucht also, das normierte Skalarprodukt

$$\frac{\langle \sigma_1, S \rangle^2}{||\sigma_1||^2}$$

zu maximieren. (Dies geschieht z. B. mit einem schrittweitengesteuerten Gradientenverfahren.)

Danach wird w_1 auf die Projektion von S auf σ_1 gesetzt:

$$w_1 := \frac{\langle \sigma_1, S \rangle}{||\sigma_1||^2},$$

womit der Defekt minimiert wird.

Gäbe es nur ein einziges verdecktes Neuron, wäre dies bereits die bestmögliche Lösung. Ansonsten fahren wir mit den weiteren wie folgt fort:
Setze $c_j := S - \sum\limits_{k=1}^{j-1} w_k \sigma_k$ und maximiere den Ausdruck

$$\frac{\langle \sigma_j, c_j \rangle^2}{||\sigma_j||^2}.$$

Setze dann

$$w_j := \frac{\langle \sigma_j, c_j \rangle}{||\sigma_j||^2}.$$

Definiert man am Anfang alle $w_j := 0$, so kann man bereits in der ersten Iteration $c_j := S - \sum\limits_{k=1}^{n} w_k \sigma_k$ für $k \neq j$ benutzen.

Nachdem man auf diese Art die Matrix Σ durchlaufen (also eine Iteration durchgeführt) hat, beginnt eine Phase der Nachiteration. Da es nicht kompliziert ist, die w_j zu bestimmen, werden sie an dieser Stelle (evtl. wiederholt) erneut berechnet.

Es werden solange Iterationen durchgeführt, bis keine nennenswerte Verbesserung des Fehlers mehr geschieht. Da der Fehler monoton fällt, hat das System dann die Eingabemuster exakt (in numerischen Versuchen bis auf Rechnergenauigkeit genau) gelernt.

Der allgemeinere Fall mit mehreren Ausgabezellen, wenn also eine mehrdimensionale Funktion interpoliert werden soll, läßt sich mit eben beschriebenen Netzen simulieren, indem ein solches Netz für jeden Ausgabewert der Funktion benutzt wird. Da dabei viele redundante Rechenschritte entstehen, sei hier noch eine Verallgemeinerung des Verfahrens angegeben.

Nun bestehe das Netzwerk zusätzlich zu den m Eingabeneuronen und n Neuronen in der verdeckten Schicht noch aus r Ausgabezellen. Die Eingabeschicht sei über Gewichte (v_{kj}) mit der verdeckten, diese über Gewichte (w_{jq}) mit der Ausgabeschicht verbunden.
Die N zu lernenden Eingabemuster seien $\{(i_{pk}), (t_{pq})\}$.
Analog sei $s_{pq} := \sigma^{-1}(t_{pq})$ definiert.

Die Bedingung zur exakten Lernbarkeit hängt nun an der Lösbarkeit des Systems

$$\begin{pmatrix} \sigma_{11} & \dots & \sigma_{1n} \\ \vdots & & \vdots \\ \sigma_{N1} & \dots & \sigma_{Nn} \end{pmatrix} \cdot \begin{pmatrix} w_{11} & \dots & w_{1r} \\ \vdots & & \vdots \\ w_{n1} & \dots & w_{nr} \end{pmatrix} = \begin{pmatrix} s_{11} & \dots & s_{1r} \\ \vdots & & \vdots \\ s_{N1} & \dots & s_{Nr} \end{pmatrix}$$

oder kurz wieder

$$\Sigma \cdot W = S.$$

Um die Spalten der Matrix S wieder als Linearkombination der Spalten der Matrix Σ darzustellen, sind die folgenden zwei Methoden geeignet:

1. Wie eben wollen wir die Spalten der Matrix Σ (und damit die Neuronen der verdeckten Schicht) der Reihe nach durchlaufen. Für den j-ten Schritt

definieren wir

$$\begin{aligned}
D &:= S - \Sigma \cdot W \\
||D||^2 &:= \sum_{i=1}^{N} \sum_{k=1}^{r} D_{ik}^2 \\
D^j &:= S - \tau^j \cdot W \\
\rho^j &:= \Sigma - \tau^j,
\end{aligned}$$

wobei τ^j die Matrix ist, die aus Σ entsteht, indem alle Einträge der j-ten Spalte Null gesetzt werden. (ρ^j ist demnach *nur* die j-te Spalte von Σ.)

Es ergibt sich:

$$\begin{aligned}
D &= D^j - \rho^j \cdot W \\
&= D^j - (\sigma_{1j}, \ldots, \sigma_{Nj})^{\mathrm{T}}(w_{j1}, \ldots, w_{jr}).
\end{aligned}$$

Minimiert man jetzt $Z := ||D^j - \rho^j \cdot W||^2$ (z. B. mittels Methode des steilsten Abstiegs mit Schrittweitensteuerung), erhält man ein Verfahren, das die Norm des Fehlers $||D||$ monoton fallen läßt. Dieses Verfahren heißt *Cyclic Backpropagation*.

2. Für jedes j benennen wir als Zielvektor eine Spalte D_λ^j von D^j (etwa diejenige mit der größten Norm oder von links nach rechts durch) und versuchen, σ_j so weit es geht in Richtung D_λ^j zu legen. Dafür müssen wir den folgenden Ausdruck maximieren:

$$\frac{\langle \sigma_j, D_\lambda^j \rangle^2}{||\sigma_j||^2}$$

(z. B. wieder mit einem Gradientenverfahren).

Die w_{jq} berechnen sich dann wie folgt:

$$w_{jq} = \frac{\langle \sigma_j, D_q^j \rangle}{||\sigma_j||^2}.$$

Numerische Versuche haben ergeben, daß erheblich weniger Trainingszyklen als bei Backpropagation nötig waren, so daß sich der höhere Rechenaufwand gelohnt hat und zu kürzeren Lernzeiten führte.

Zudem ist es möglich, dynamisch Neuronen zu der verdeckten Schicht hinzuzufügen; dazu werden einfach die Matrizen Σ bzw. W um eine Spalte bzw.

Zeile erweitert und diese in nachfolgenden Iterationen mit einbezogen.

Allerdings beschränkten sich die Versuche nur auf einfache Funktionen an zufälligen Stellen. Außerdem ist es nicht immer wünschenswert, die Eingabedaten genau zu interpolieren; man denke da z. B. an physikalische Meßwerte, die *immer* eine gewisse Ungenauigkeit aufweisen.

4 Spezielle Neuronale Netze

4.1 Rekurrente Netze

Rekurrente Netze sind solche mit „Rückverbindungen“, sind also nicht mehr unbedingt kreisfrei.
Damit ist es möglich, die aktuelle Ausgabe von vorherigen Ein- und Ausgaben abhängig zu machen und somit eine Art Kontext zu schaffen. So erzeugt ein rekurrentes Netz bei derselben Eingabe nicht mehr unbedingt dieselbe Ausgabe.

4.2 Jordan- und Elman-Netze

Jordan-Netze sind Feedforward-Netze, die zusätzliche „Erinnerungszellen“ (Kontextzellen) besitzen, die jeweils mit einer Ausgabezelle verbunden und somit in der Lage sind, alte Ausgaben zu speichern.

Die Gewichte auf den Verbindungen von den Ausgabezellen zu den Kontextzellen geben an, in welchem Maße die letzte Ausgabe in den Speicherwert einfließt. Die Ausgaben der Kontextzellen gehen im nächsten Schritt als Eingaben wieder in das Netz hinein.

Bei Elman-Netzen hingegen besitzt jedes verdeckte Neuron eine eigene Kontextzelle, um ältere Aktivierungswerte zu speichern. Das Bild zeigt ein Elman-Netz mit einer einzigen verdeckten Schicht. Es kann aber durchaus auch mehrere geben; dann kann es auch sinnvoll sein, die Kontextzellen verschiedener Schichten miteinander zu verbinden, so daß das gesamte System auf vorherige Zustände zurückgreifen kann.

4.3 Lernverfahren für rekurrente Netze

Während es für Jordan- und Elman-Netze nicht schwierig ist, Backpropagation zu modifizieren (Die Ausgaben der Kontextzellen können als zusätzliche Eingaben in das Netz betrachtet werden), ist dies für allgemeine rekurrente Netze nicht ohne weiteres möglich. Kreise führen beim Zurückpropagieren des Fehlers zu Problemen. Man beachte dabei insbesondere, daß auch Rückverbindungen von Neuronen zu sich selbst zugelassen sind.

Theoretisch ist es möglich, jeden Zeitschritt eines rekurrenten Netzes in einem Feedforward-Netz zu simulieren und es so oft wie nötig hintereinanderzuschalten; die Ausgaben eines Netzes dienen dann als Eingaben für das nachfolgende Netz.
Dies hat allerdings *sehr* viele Knoten und Verbindungen zur Folge.

Stattdessen läßt man das Netz eine bestimmte Anzahl von Zeitschritten durchlaufen, vergleicht die tatsächlichen mit den erwünschten Ausgaben und korrigiert die Gewichte über genausoviele Zeiteinheiten rückwärts durch das Netz. Dieses Verfahren nennt man *Backpropagation Through Time*.

Es gibt noch eine Reihe anderer Ansätze von Lernverfahren in rekurrenten Netzen, die hier aber nicht behandelt werden.

4.4 Hopfield-Netze

Ein Hopfield-Netz kann man sich am besten als vollständigen Graphen vorstellen, d. h. daß jedes Neuron mit jedem anderen verbunden ist. Allerdings gibt es (wie im vollständigen Graphen auch) keine direkte Rückkopplung einer Zelle mit sich selbst. Die Gewichtsmatrix ist zudem symmetrisch, also gilt $w_{ij} = w_{ji}$ für alle $i \neq j$.

Zusätzlich zu den Verbindungen zwischen den Zellen bekommen einige Neuronen Werte von außerhalb und andere liefern Werte nach außen. Diese Eigenschaften, nämlich Eingabe- bzw. Ausgabezelle zu sein, kann prinzipiell jedes Neuron besitzen. Eine Zelle kann theoretisch sogar gleichzeitig Eingabe- *und* Ausgabeneuron sein.

Es gibt verschiedene Möglichkeiten, in welcher Reihenfolge die einzelnen Neuronen ihre Aktivierung berechnen. Man kann sie in zufälliger, oder aber (nach einmaliger Numerierung) in einer festen Reihenfolge abarbeiten. Bei heutigen massiv parallelen Rechnersystemen ist auch die gleichzeitige Behandlung aller Neuronen möglich. Die Kommunikationsstruktur eines solchen Parallelrechners muß allerdings den Kontakt jedes Prozessors mit jedem anderen zulassen.

Als Lernregel für Hopfield-Netze kann man einerseits für jedes Neuron nach einer bestimmten Anzahl Zeitschritte den Fehler korrigieren. Andereseits ist bei paralleler Bearbeitung (und schneller Hardware) das Lernen nach jedem Schritt möglich und sinnvoll.

Bei Hopfield-Netzen (wie bei anderen rekurrenten Netzen auch) besteht die Gefahr, daß das System in unstabile Zustände gelangen kann. Um dies zu vermeiden, kann man die Änderungen der Gewichte mit der Zeit gegen Null streben lassen. Damit ist die Konvergenz des Netzes gewährleistet. (Ob das Netz dann allerdings auch schon das Gewünschte berechnet, ist nicht sicher !)

In Kap. 5.1 wird als Beispiel einer Anwendung von Hopfield-Netzen das Traveling Salesman Problem (TSP) behandelt.

4.5 Time Delay-Netze

Time Delay-Netze (Time Delay Neuronal Networks, TDNN) sind durch eine besondere Behandlung des Inputs fähig, zeitlich sich verändernde Muster zu verarbeiten. Dazu wird jedes Neuron um eine feste Anzahl von zusätzlichen Neuronen (Delays) erweitert. Diese zusätzlichen Zellen geben ihre Eingabe zu verschiedenen Zeitpunkten an ihre Nachfolger weiter.

Dabei erfolgt die Weitergabe jeweils zeitlich verzögert um eine bestimmte Anzahl diskreter Zeitschritte.

Die Ausgabeneuronen liefern (entgegen bisher betrachteter Systeme) die Summe der Quadrate der Ausgaben der letzten verdeckten Schicht; es wird also quasi nach der Zeit integriert. Der Grund für die Quadrierung liegt darin, daß die am stärksten aktivierten Neuronen wesentlich die Ausgabe beeinflussen sollen.

Da obige Darstellung schnell recht unübersichtlich wird, trägt man die zeitliche Entwicklung nach rechts und die verschiedenen Eingabevektoren nach unten ab:

Damit die Vernetzung nicht zu umfangreich wird, beschränkt man die Eingabe jedes Neurons meist auf einen bestimmten Zeitausschnitt. Ausgaben, die verschiedene Zeitabschnitte betreffen, können dann in weiteren Schichten zusammengefaßt werden.

Als Aktivierungsfunktion wird meist die logistische Funktion

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

verwendet, als Lernregel eine geringfügige Modifikation von Backpropagation.

Es ist auch möglich, aus den Eingaben erst mit mehreren Time Delay-Netzen Teilergebnisse zu berechnen, die dann in späteren hierarchisch angeordneten Stufen zu einem Gesamtergebnis kombiniert werden.

5 Anwendungen

5.1 TSP mit Hopfield-Netzen

Das sogenannte *Handlunsreisenden-Problem* (Traveling Salesman Problem) besteht darin, in einem vollständigen gewichteten Graphen eine Rundreise durch alle n Knoten so zu finden, daß einerseits jeder Knoten genau einmal besucht wird, andererseits die Gesamtkosten dieser Tour minimal sein sollen.

Hintergrund ist der, daß ein Handlungsreisender alle Städte besuchen will, in denen er Kunden hat, und dabei möglichst wenig Spesen verursachen darf.

Dieses Problem ist $\mathcal{NP}$-schwer, d. h. im wesentlichen, daß es keinen deterministischen Algorithmus gibt, der signifikant schneller arbeitet, als alle $(n-1)!/2$ Touren nacheinander durchzutesten. Dies sind bei nur 30 Knoten (Städten) bereits über 10^{30} viele !

Deshalb ist man auch daran interessiert, vielleicht nicht die *optimale*, aber doch eine sehr gute Lösung zu bekommen. Hier soll nun ein Ansatz mit Neuronalen Netzen vorgestellt werden.

Dazu wird ein Netz verwendet, das aus $n \times n$ Knoten besteht. Die binäre Ausgabe von Neuron (x, i) sei o_{xi}; $o_{xi} = 1$ soll bedeuten, daß Stadt x in unserer Tour als i. besucht wird.

Zulässige Lösungen sind also genau die 0-1-Matrizen $O = (o_{xi})$, die in jeder Zeile und jeder Spalte genau eine 1 stehen haben.

Ferner bezeichne dist_{xy} den Abstand von (das Kantengewicht zwischen) Stadt x und Stadt y. Damit ergibt sich der zu minimierende Ausdruck:

$$E = \frac{A}{2} \cdot \sum_x \sum_i \sum_{j \neq i} o_{xi} o_{xj} + \frac{B}{2} \cdot \sum_x \sum_i \sum_{y \neq x} o_{yi} o_{xi} + \frac{C}{2} \cdot \left(\left(\sum_x \sum_i o_{xi} \right) - n \right)^2 + \frac{D}{2} \cdot \sum_x \sum_i \sum_y \mathrm{dist}_{xy} o_{xi} \cdot (o_{y,i+1} + o_{y,i-1}).$$

Dies ist wie folgt motiviert:

- Der erste Term wird genau dann Null, wenn jede Zeile nur eine einzige 1 enthält.
- Der zweite Term wird genau dann Null, wenn jede Spalte nur eine einzige 1 enthält.
- Der dritte Term wird genau dann Null, wenn genau n Einsen in der Matrix vorkommen.
- Der vierte Term gibt die Länge der Tour an und soll auch möglichst minimal sein.

Der Faktor 1/2 ergibt sich dadurch, daß jede Verbindung doppelt gezählt wird.

Der interessante Punkt hierbei ist die Wahl der Parameter A, B, C und D, also in welchem Maße unzulässige Lösungen „bestraft" werden sollen. Zu große Werte verhindern die Konvergenz zu sinnvollen Lösungen, zu kleine Werte liefern evtl. nicht zulässige Lösungen.

Für die Wahl der Gewichte ergibt sich nach kurzer Rechnung

$$w_{xi,yj} = w^{(A)}_{xi,yj} + w^{(B)}_{xi,yj} + w^{(C)}_{xi,yj} + w^{(D)}_{xi,yj},$$

wobei

$$\begin{aligned} w^{(A)}_{xi,yj} &:= -A \cdot \psi_{xy} \cdot (1 - \psi_{ij}), \\ w^{(B)}_{xi,yj} &:= -B \cdot \psi_{ij} \cdot (1 - \psi_{xy}), \\ w^{(C)}_{xi,yj} &:= -C, \\ w^{(D)}_{xi,yj} &:= -D \cdot \mathrm{dist}_{xy} \cdot (\psi_{j,i+1} + \psi_{j,i-1}). \end{aligned}$$

Hierbei bezeichne ψ_{xy} das Kronecker-Symbol. (Das sonst übliche Symbol δ wurde bei Backpropagation bereits als Fehlersignal verwendet.)

Die verwendete Aktivierungsfunktion lautet:

$$o_j = \frac{1}{2} \cdot \left(1 + \tanh\left(\frac{\mathrm{net}_j}{k}\right)\right)$$

und die Schwellenwerte betragen $\Theta_i := C \cdot n$.

Numerische Versuche für Touren durch 10 Städte haben ergeben, daß 16 von 20 Instanzen konvergierten; die Hälfte der Lösungen lag sehr nahe beim Optimum.

Allerdings gibt es auch Approximationsverfahren ohne die Verwendung Neuronaler Netze, die stets zulässige Lösungen mit garantiert kleinem Fehler liefern.

5.2 OCR mit Neocognition

Durch Neocognition ist es möglich, Zeichen translations- und skalierungsinvariant erkennen zu können. Dies wird bei der Schrifterkennung (Optical Character Recognition, OCR) ausgenutzt. Hierbei sollen die einzelnen Buchstaben eines vorgelegten Schriftstückes, z. B. einer handgeschriebenen Nachricht, erkannt werden.

Das entsprechende Neuronale Netz hat die folgende Struktur:

Die Eingabeschicht „liest“ die Vorlage ein. Alle anderen Schichten bestehen aus zwei verschiedenen Stufen unterschiedlicher Neuronen:

- Die S-Zellen sind zur Erkennung von optischen Mustern oder Musterteilen fähig, die von der Vorgängerschicht geliefert werden. Sie geben ihre Ausgabe an die C-Schicht weiter. Die Verbindungsgewichte zu einer S-Zelle sind variabel und können trainiert werden.

- Die C-Neuronen abstrahieren die Ausgaben der S-Zellen der Vorgängerschicht lediglich dahingehend, daß die Erkennung von Mustern skalierungs- und translationsinvariant wird. Die Gewichte zu C-Zellen hin sind deshalb fest.

Alle Neuronen einer Stufe erkennen dasselbe Muster, nur an unterschiedlicher Position. Dabei werden in früheren Schichten nur einfache Muster erkannt, die in späteren Schichten zu komplexeren Strukturen zusammengefügt werden können. Welches Muster ein Neuron erkennen soll, muß durch Lernen bestimmt werden.

Zur Stimulierung einer C-Zelle reicht es aus, wenn auch nur ein einziges vorgeschaltetes S-Neuron das entsprechende Muster erkannt hat. Durch Aktivierung mehrerer S-Zellen der nächsten Schicht durch das C-Neuron wird eine künstliche „Unschärfe“ erzeugt, so daß auch leicht unterschiedliche Muster als identisch erkannt werden.

Es ist möglich, jede Stufe einzeln auf bestimmte Teilmuster zu trainieren. Dabei ist allerdings die Kenntnis der erwünschten Ausgabe auch für jede verdeckte Schicht von Neuronen notwendig. Deshalb heißt diese Form des Lernens auch *vollständig überwachtes Lernen*.

Anwendung findet dieses Verfahren z. B. bei der Adressenerkennung auf Briefen bei der Post und bei optischer Schrifterkennung in Grafikprogrammen. Bei letzteren hat diese Methode schon vor einigen Jahren das Verfahren des Bitmap-Vergleichs abgelöst.

5.3 Spracherkennung mit TDNN

Time Delay-Netze (wie in Kap. 4.5 beschrieben) können zur Spracherkennung verwendet werden. Als Eingaben dienen dann die an diskreten Zeitschritten bestimmten Meßwerte von verschiedenen Frequenzlagen. Es empfiehlt sich auch hier, daß jedes verdeckte Neuron nur einen bestimmten Zeitabschnitt kontrolliert (als Eingabe erhält). Die Ausgabe summiert wieder die Quadrate der Outputs der Neuronen der letzten Schicht und liefert den erkannten Buchstaben (bzw. Laut) zurück.

Heutige Spracherkennungs-Systeme sind schon sehr leistungsfähig. Sie können auch schon für verschiedene Sprecher trainiert werden; teilweise sind sie sogar in der Lage, männliche und weibliche Stimmen unterscheiden zu können.

5.4 Weitere Anwendungsmöglichkeiten

Das Spektrum möglicher Einsatzgebiete der Neuronalen Netze ist sehr groß. Neben den hier behandelten Beispielen seien an dieser Stelle nur genannt:

- Wettervorhersage: aus aktuellen Wetterdaten wie Temperatur, Luftdruck, Windstärke usw. wird versucht, das Wetter für einige Tage vorherzubestimmen.

- Aktienkurs-Prognose: aus der Entwicklung der Aktien in den letzten Tagen und Wochen soll eine zuverlässige Aussage über Kaufen, Halten oder Verkaufen getroffen werden.

6 Minimierung von Netzen

Wie z. B. in Kap. 5 bei OCR und Spracherkennung gesehen, bestehen Neuronale Netze schnell aus einer sehr großen Anzahl von Neuronen. Um dem entgegenzuwirken, gibt es Ansätze zur (nachträglichen) Reduktion der Anzahl Neuronen bzw. der Gewichte zwischen ihnen.

So ist es zum Beispiel möglich, Gewichte, die betragsmäßig sehr klein sind, auf Null zu setzen und damit aus der Gewichtsmatrix zu löschen.
Die Idee dabei ist, daß diese Gewichte zur Ergebnisfindung des Netzes einen nur vernachlässigbar kleinen Anteil beisteuern.

Nach dieser Maßnahme ist es ratsam, die verbleibenden Gewichte „nachzutrainieren“ und das verbleibende System an die neue Situation anzupassen. Die übrigen Gewichte müssen dann den Ausfall von Verbindungen auszugleichen versuchen. Die Qualität der Approximation wird dabei i. a. negativ beeinträchtigt. Der Rechenaufwand kann aber teilweise enorm reduziert werden.

Beim *Optimal Brain Damage* wird für jedes Gewicht geschätzt, welchen Einfluß es auf den Fehler des Netzes nimmt. Die Gewichte mit den geringsten Auswirkungen (Verschlechterungen) werden dann gelöscht. Auch hier sollte das verkleinerte Netz nachtrainiert werden.

Im Gegensatz dazu werden beim *Optimal Brain Surgeon* Gewichte tangential zu Fehlerfläche verändert. Dabei werden einige Null gesetzt; es kann aber auch vorkommen, daß dadurch andere Gewichte erhöht werden. Nach dem Ausdünnen des Neuronalen Netzes kann wieder ein Nachtraining erfolgen.

Bei der *Skelettierung* wird für jedes verdeckte Neuron einzeln bestimmt, wir wichtig es für den Wert der Ausgabe ist. Z. B. kann dazu die Differenz des globalen Fehlers und des Fehlers des einzelnen Neurons dienen.

Man kann auch von vorneherein ausschließen, daß einzelne Zellen einen zu großen Einfluß auf das Gesamtergebnis haben, indem man zu groß gewordene Gewichte „bestraft“ (vgl. Weight Decay, Kap. 3.4).

Literatur

Andreas Zell: Simulation Neuronaler Netze, Addison-Wesley 1996
(Abbildungen sind hieraus entnommen)

spezielle Literatur für Kap. 3.5:

Frank Bärmann/Friedrich Biegler-König:
On a Class of Efficient Learning Algorithms for Neuronal Networks

Inhaltsverzeichnis