Raúl Rojas

Künstliche Neuronale Netze als neues Paradigma der Informationsverarbeitung

1. Die biologische Motivation

Neuronale Netze bilden ein alternatives Berechnungsmodell, das in den letzten Jahren in den Ingenieurwissenschaften zunehmende Beachtung gefunden hat. Die Motivation dieses neuen Paradigmas liegt, wie der Name schon andeutet, in der Biologie: man möchte Automaten bauen, die gewisse Aspekte der Informationsverarbeitung bei Mensch und Tier nachahmen. In der Sprache der Informatik ausgedrückt, können wir sagen, daß biologische Nervensysteme *massiv parallel* arbeiten, *fehlertolerant* sind und sich *adaptiv* verhalten. Gerade diese Eigenschaften sollten auch künstliche Automaten aufweisen. Bei gewissen Anwendungen sind sie sogar unerläßlich, wie z.B. Fehlertoleranz in der Robotik und adaptives Verhalten bei der Spracherkennung. Neuronale Netze sollen sich außerdem als *lernende Systeme* verhalten, die ihre eigenen Parameter bestimmen und anpassen können (Hertz et al. 1991, Ritter et al. 1989).

Obwohl neuronale Netze erst in den letzten Jahren allgemeine Anerkennung fanden, ist ihre Geschichte fast so alt wie die der traditionellen Computersysteme selbst. Die ersten Modelle abstrakter Neuronen wurden von Warren McCulloch und Walter Pitts im Jahre 1943 vorgeschlagen, also zu einer Zeit, in der gerade die ersten Computeranlagen gebaut wurden. Nur wenige wissen, daß sich John von Neumann in seiner Arbeit, die die Architektur zukünftiger Computersysteme

auf lange Zeit festschrieb, in seiner Argumentation des biologischen Modells bediente. Dabei bildeten die Untersuchungen von McCulloch und Pitts, sowie die Kybernetik Norbert Wieners den Hintergrund seiner Ausführungen. Daß später über künstliche neuronale Netze eher marginal geforscht wurde, ist zum Teil das Ergebnis des überwältigenden Erfolges der traditionellen Computersysteme, die einen alternativen Zugang – den symbolischen – zur maschinellen Intelligenz bieten.

Erst seit den siebziger aber vor allem seit den achtziger Jahren erleben wir eine Renaissance der Forschung auf dem Gebiet des Konnektionismus. Die *subsymbolischen* Fähigkeiten neuronaler Netze sollen jetzt jene Systeme unterstützen, die bis heute menschliche Intelligenz auf der rein symbolischen Ebene, d.h. auf der Ebene der Logik, modelliert haben. In diesem Aufsatz wollen wir besprechen, was neuronale Netze sind und welche Art von Berechnungen sie durchführen können. Wir zeigen, wie solche Systeme trainiert werden, welches der Unterschied zum traditionellen algorithmischen Zugang ist und wie adaptives Verhalten implementiert werden kann. Es wird anhand des Beispieles der automatischen Sprachsynthese erläutert, wie künstliche neuronale Netze in traditionelle algorithmische Verfahren eingebettet werden können.

Auch wenn die einzelnen biologischen Modelle des Gehirns und der Nervensysteme von Lebewesen sich in vielen Aspekten unterscheiden, herrscht allgemeine Übereinstimmung darüber, daß „das Wesen der Funktion des Nervensystems Kontrolle durch Kommunikation ist". Nervensysteme bestehen aus vielen Tausenden oder Millionen von Nervenzellen, die miteinander vernetzt sind. Jede individuelle Nervenzelle ist sehr komplex in ihrem Aufbau und kann eintreffende Signale auf vielfältige Weise verarbeiten.

Trotz aller Unterschiede bei biologischen und nichtbiologischen Berechnungsmodellen, ist für sie folgende metaphorische Gleichung vorgeschlagen worden:

$$\textit{Berechnung} = \textit{Speicherung} + \textit{Übertragung} + \textit{Verarbeitung}$$

Wird z.B. mit einem konventionellen Computer operiert, so ist ein Befehlssatz zur Berechnung bestimmter primitiver

Funktionen notwendig, die dann zu unterschiedlichen Programmen kombiniert werden können. Im Fall von künstlichen neuronalen Netzen befinden sich die primitiven Funktionen an den Knoten des Netzes, und die Verarbeitungsregeln werden durch die Vernetzung sowie durch die Synchronizität oder Asynchronizität der Informationsübertragung ausgedrückt. Typische künstliche neuronale Netze haben die in Abb. 1 gezeigte Struktur.

Das Netz kann als eine Funktion F interpretiert werden, die für die Eingabe (x,y,z) evaluiert wird. An den Knoten werden elementare Funktionen f_1, f_2 usw. berechnet. Deren Komposition definiert F. Vor allem drei Elemente sind für die Definition des Netzes von Bedeutung: die Struktur der einzelnen Knoten, das Vernetzungsmuster und der Lernalgorithmus, der die Netzgewichte oder andere Netzparameter bestimmt. Die verschiedenen Modelle unterscheiden sich in diesen drei Merkmalen.

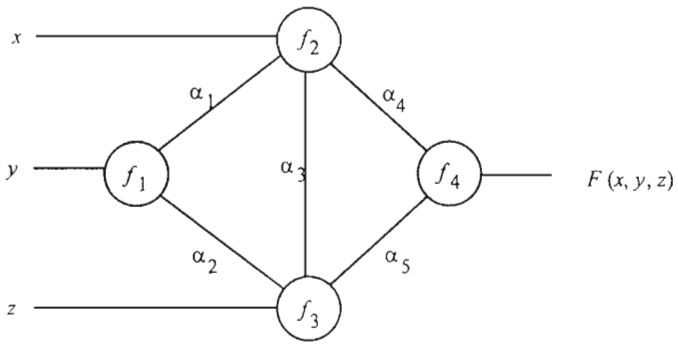

Abb. 1 Funktionales Modell eines neuronalen Netzes

Schon dieses Beispiel macht deutlich, daß bei jedem Berechenbarkeitsmodell eine globale oder eine lokale Sichtweise eingenommen werden kann. Man kann sich global dafür interessieren, welche Klasse von Funktionen überhaupt durch ein bestimmtes Schema berechenbar ist, oder lokal den operationellen Vorgang selbst minutiös studieren.

Sollte eine Definition für die angesprochenen Systeme gegeben werden, könnte folgendes gesagt werden:

Künstliche neuronale Netze sind Netze von einfachen primitiven Funktionen.

Die primitiven Funktionen sitzen an den Knoten des Netzes und gehen meistens nicht über Summation von Information und eine eindimensionale nichtlineare Funktion hinaus. Informationsübertragung findet nur über die Kanten des Netzes statt. Die Weitergabe wird durch die *synaptische Stärke*, d.h. ein Kantengewicht, moduliert.

2. Die Berechnungselemente - Perzeptrone

Künstliche neuronale Netze werden im Prinzip als eine Art *black box* verwendet, die für eine gewisse Eingabe eine bestimmte Ausgabe erzeugen soll. In das Netz wird im allgemeinen ein n-dimensionaler reeller Vektor $(x_1,x_2,...,x_n)$ eingegeben, dem ein m-dimensionaler Vektor $(y_1,y_2,...,y_m)$ als Ausgabe entspricht.

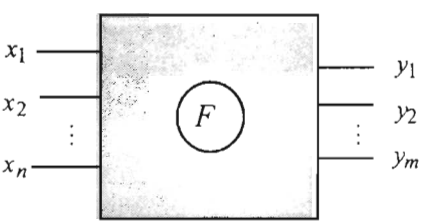

Abb. 2 Ein neuronales Netz als black box

Ein neuronales Netz verhält sich also wie eine „Abbildungsmaschine", die eine Funktion $F : R^n \rightarrow R^m$ modelliert. Einige Probleme treten jedoch auf, sobald wir den Berechnungsprozeß der Funktion F näher betrachten. Wird die Funktion mit einem Netz von Funktionen evaluiert, fließt die Information

zwischen den Knoten in eine durch die Vernetzung festgelegte Richtung. Einige Knoten berechnen Resultate, die als Funktionsargumente an andere Knoten übertragen werden. Gibt es in dem Netz von Funktionen keine Rückkopplungsschleifen, so ist der Berechnungsprozeß eindeutig. Es kann dann angenommen werden, daß die Funktionsauswertung an jedem Knoten ohne Zeitverlust vonstatten geht. Eine bestimmte Eingabe führt zu einer eindeutigen Ausgabe.

Ein Netz von Funktionen besteht aus Verbindungen (Kanten) und Knoten, die die Funktionsauswertung durchführen. Die einzelnen Knoten des Netzwerkes besitzen mehrere unabhängige Eingänge und einen Ausgang, d.h. die Information fließt in eine bestimmte Richtung. Die Kanten des Netzwerkes sind gerichtete Informationskanäle, die die Information von einem Neuron zum anderen transportieren. Wenn an einem Knoten n ankommende Verbindungen angeschlossen sind, transportiert jede davon ein Argument für die Funktionsauswertung an diesem Knoten, wobei n nicht beschränkt ist. Diese Eigenschaft wird als unbegrenztes *fan-in* der Knoten bezeichnet.

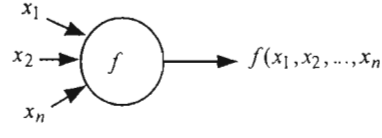

Abb. 3 Auswertung einer Funktion mit n Argumenten

Um den Lernvorgang der Netze zu vereinfachen, und in Anlehnung an das biologische Modell, wird an jedem Knoten eine primitive Funktion von einem einzigen Argument berechnet. Gewöhnlich wird in einem Netz in jedem Knoten dieselbe Funktion verwendet. Die Reduktion von n Argumenten auf ein einziges Argument für die Auswertung von f wird durch eine *Integrationsfunktion* geleistet. In den meisten neuronalen Modellen werden die Argumente, die über die ankommenden Leitungen an den Knoten übertragen werden, einfach addiert. Abb. 4 zeigt die Struktur eines generischen Neurons, das aus zwei Teilen besteht: einem Integrationsteil, an dem die Einga-

be zusammengefaßt, und einem Ausgabeteil, an dem die Ausgabe des Neurons berechnet wird.

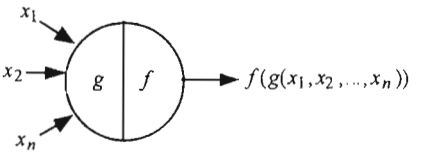

Abb. 4 Generisches Neuron

Ein wichtiger Spezialfall eines Berechnungselements ist das *Perzeptron* dessen Diagramm in Abb. 5 gezeigt wird (Minsky 1967, Minsky & Papert 1969).

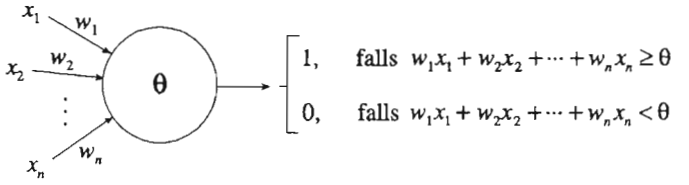

Abb. 5 Diagramm eines Perzeptrons

Ein Perzeptron besitzt n Eingabeleitungen, die jeweils mit den Gewichten w_1, w_2,..., w_n behaftet sind. Das Perzeptron kann nur Schwellenwertentscheidungen treffen, d.h. die Ausgabe der Zelle ist 1, falls $x_1w_1 + x_2w_2 + \ldots + x_n w_n \geq \theta$ gilt, wobei θ der sogenannte Schwellenwert der Zelle ist. Falls $x_1w_1 + x_2w_2 + \ldots + x_n w_n < \theta$ gilt, wird eine Null ausgegeben. Mit Perzeptronen können logische Funktionen ohne weiteres realisiert werden. Die AND-Verknüpfung zweier binärer Variablen x_1 und x_2 kann durch eine Zelle mit zwei Eingabeleitungen implementiert werden, bei der w_1 und w_2 beide gleich 1 sind und $\theta = 2$ gilt. Die Zelle wird nur dann eine 1 feuern, wenn $x_1 + x_2 \geq 2$ erfüllt ist, d.h. bei binären Eingaben genau dann, wenn $x_1 = x_2 = 1$ ist. Abb. 6 zeigt die Implementierung der lo-

gischen Funktionen AND, OR und NOT mittels eines Perzeptrons. Da diese drei Funktionen eine Basis für die Implementierung aller möglichen booleschen Funktionen bilden, ist klar, daß mit einem Netz von Perzeptronen beliebige logische Funktionen berechnet werden können.

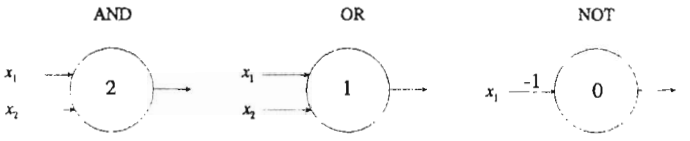

Abb. 6 Einige logische Funktionen

3. Klassifizierungsnetze

Das Diagramm zeigt die allgemeine Struktur eines Klassifizierungsnetzes. Dies ist ein Beispiel für ein vorwärtsgerichtetes Netz mit mehreren Schichten von Verarbeitungselementen.

Die Eingabe in das Netz ist ein n-dimensionaler Vektor von Daten und die Ausgabe ist die dazugehörige Klassifikation. Soll das Netz z.B. zwischen M verschiedenen Klassen unterscheiden, dann besitzt es M Ausgabeleitungen. Wenn der Eingabevektor zur Klasse i gehört, dann soll die i-te Ausgabeleitung gleich Eins sein, und alle anderen Null. Diese *sparse* Darstellung der Ausgangswerte kann bei der Codierung der Trainingsmenge vereinbart werden. Ein Beispiel wären akustische Eingabevektoren, die als eines von z.B. 64 Phonemen zu identifizieren sind. Das Netz soll lernen, solche Vektoren den entsprechenden Phonemen zuzuordnen. Was passiert jedoch, falls eine unbekannte Eingabe in das Netz eingegeben wird? In einem solchen Fall antwortet das Netz in der Regel mit Ausgabewerten zwischen Null und Eins. Interessant ist dann aber, daß diese Ausgabewerte echte Bayes-Wahrscheinlichkeiten (Klassenzugehörigkeit-Wahrscheinlichkeiten) darstellen (Rojas 1996). So ist die Ausgabe o_i die Wahrscheinlichkeit, daß der Eingabevektor x zur Klasse i gehört (Richard & Lippmann 1991).

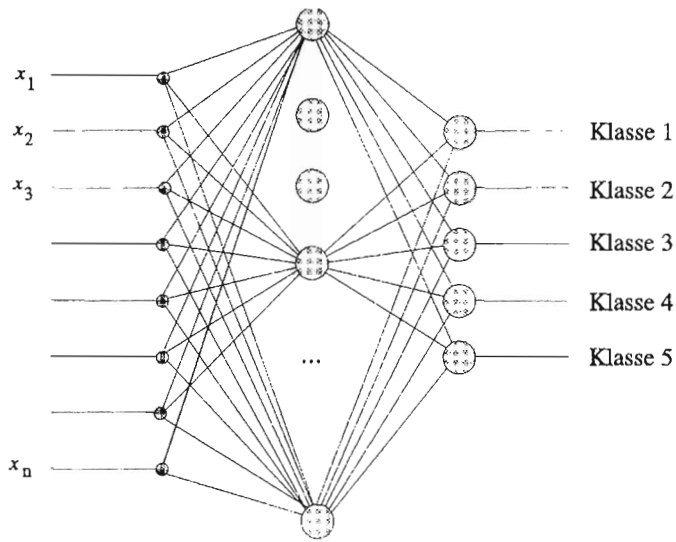

Abb. 7 Allgemeine Struktur eines Klassifizierungsnetzes

Daß mit einem neuronalen Netz Klassenzugehörigkeit-Wahrscheinlichkeiten berechnet werden, kann mit Hilfe von Abb. 8 visualisiert werden. In einem Eingaberaum (Merkmalsraum) gibt es verschiedene Cluster, die unterschiedliche Klassen definieren. Die Zugehörigkeit zu einer Klasse ist aber nur probabilistisch gegeben. Ein solcher Merkmalsraum könnte z.B. aus n-dimensionalen Vektoren bestehen, die n verschiedene Krankheitssymptome beschreiben. Die definierten Klassen sind dann bestimmte Krankheiten. Ein Vektor von Symptomen gehört dann mit einer gewissen Wahrscheinlichkeit zu einer bestimmten Krankheit. Dies wird in der Abbildung mit stilisierten Gauß-Glocken dargestellt. Cluster von Symptomen können sich sogar überlappen, so daß ähnliche Symptome z.B. zu unterschiedlichen Krankheiten gehören. Solche Überlappungen wären nur durch zusätzliche Information aufzulösen. In einem noch höher dimensionalen Raum könnten die Cluster damit entflochten werden. Diese zusätzliche Informa-

tion steht aber normalerweise nicht zur Verfügung, so daß nur probabilistische Aussagen möglich sind. Dies ist aber gerade das, was das neuronale Netz anhand der vorklassifizierten Beispiele lernt.

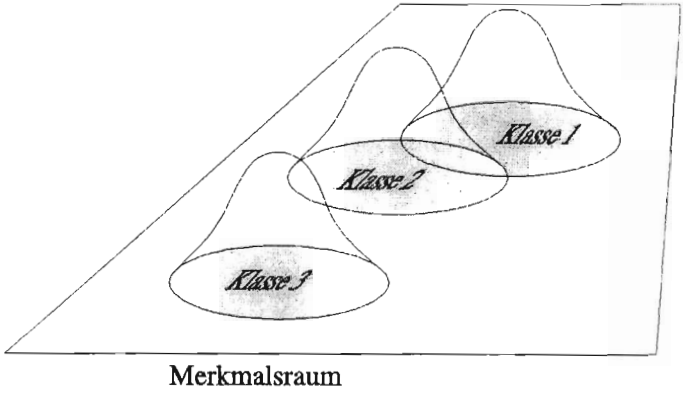

Abb. 8 Wahrscheinlichkeitsverteilung der Klassenzugehörigkeit

Damit ist schon eine mögliche Anwendung solcher Klassifizierungsnetze umrissen worden, nämlich der Einsatz in der Medizin. Existierende medizinische Datenbanken können als Trainingsmenge für ein Netz genommen werden. Das Netz erstellt dann eine erste statistische Prognose anhand der Symptome, und ein Arzt kann die Bayes-Wahrscheinlichkeiten in seine Überlegungen einbeziehen.

3.1 Ein Beispiel: NETtalk

Ein elegantes Beispiel für die Anwendung von Klassifizierungsnetzen finden wir bei der Sprachsynthese. Systeme für die Ausgabe von Sprache werden schon seit Jahren kommerziell angeboten. Sie transformieren Zeichenketten durch die Anwendung linguistischer Regeln in eine Reihe von sprachlichen Phonemen. Die dafür notwendige Regelmenge ist keineswegs trivial. Mitte der achtziger Jahre wurde ein Backpro-

pagation-Netz entwickelt, das ohne eingebaute linguistische Regeln in der Lage war, englische Texte mit einer ähnlichen Erfolgsquote wie die komplexeren Systeme auszusprechen. Das NETtalk-Netz besteht aus 7 Gruppen von 29 Eingabestellen, 80 Neuronen in der mittleren Schicht und 26 Ausgabeneuronen (Abb. 9), wobei die Verbindungsleitungen nicht gezeichnet wurden). Der vorzulesende Text wird durch ein Fenster gescannt. In jedem Schritt werden sieben Buchstaben abgetastet. Jeder davon setzt eine der 29 zugehörigen Eingabestellen auf 1 und den Rest auf 0. Die Aufgabe des Netzes besteht darin, die richtige Aussprache für die im Textfenster gezeigten Buchstaben zu produzieren. Insgesamt können 26 Phoneme selektiert werden. Das Netz besitzt rund 18000 Gewichte. Die große Anzahl von Freiheitsgraden erlaubt es, selbständig statistische Regelmäßigkeiten herauszufinden, die gewisse linguistische Regeln widerspiegeln.

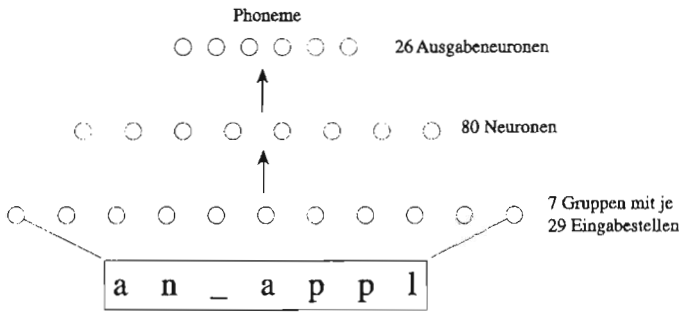

Abb. 9 Die NETtalk-Architektur

Das Netz wird mit einem Text und dessen Codierung in Phoneme trainiert. Dafür müssen mehrere hundert Worte per Hand codiert werden. Nach dem Training wird dem Netz ein unbekannter Text mit neuen Worten vorgelegt. Die Aussprache des Netzes ist qualitativ ähnlich der von regelbasierten Systemen. Ein interessantes Phänomen des NETtalk-Lernprozesses ist die Tatsache, daß das Netz am Anfang des Trainings ähnliche sprachliche Schwächen wie man sie bei Vorschulkin-

schulkindern findet, zeigt. Im Laufe des Lernens werden die Fehler allmählich korrigiert. Eine Analyse der Neuronengewichte zeigt, daß einige Neuronen der verborgenen Schicht sich auf spezielle linguistische Regeln spezialisieren.

4. Lernen in vorwärtsgerichteten Netzen

Nun stellt sich die Frage nach dem Training von solchen Systemen wie NETtalk. Wie werden die Parameter für das Netz automatisch gefunden? In diesem Abschnitt besprechen wir den *Backpropagation Algorithmus*, der als „Marktführer" auf diesem Gebiet gelten kann. Dieses numerische Verfahren wurde in den siebziger Jahren von verschiedenen Forschern unabhängig voneinander und für unterschiedliche Anwendungen entwickelt, geriet jedoch bis 1985 in Vergessenheit, als Rumelhart und Mitarbeiter den Algorithmus einer breiteren Öffentlichkeit vorlegten. Seitdem ist es zu einer der am weitesten verbreiteten Lernmethoden für neuronale Netze geworden (Rumelhart & McClelland 1986).
Wenn ein Netz für eine bestimmte Aufgabe vorgegeben wird, werden seine Parameter zuerst per Zufall ausgewählt. Dies führt zu einem großen Fehler bei der Ausgabe: das Netz ist zunächst einmal nicht in der Lage die Trainingsmenge korrekt zu bearbeiten. Die Beziehung zwischen Parameter und Fehler nennen wir die „Fehlerfunktion".

Backpropagation sucht das Minimum der Fehlerfunktion eines bestimmten Lernproblems durch Abstieg in der Gradientenrichtung. Die Kombination derjenigen Gewichte eines Netzes, die den Berechnungsfehler minimiert, wird als Lösung des Lernproblems betrachtet. Der Gradient der Fehlerfunktion muß also für alle Punkte des Gewichteraums existieren, d.h. die partiellen Ableitungen der Fehlerfunktion nach den einzelnen Gewichten müssen überall definiert sein.

4.1 Differenzierbare Aktivierungsfunktionen

In Klassifizierungsnetzen kann eine beliebige Aktivierungsfunktion eingesetzt werden, aber aus technischen Gründen wird meistens eine Sigmoide verwendet, eine Funktion, die in gewisser Weise die Funktionsweise eines Perzeptrons verallgemeinert. Die Sigmoide $s_c : R \to (0,1)$ wird durch folgenden Ausdruck definiert:

$$s_c(x) = \frac{1}{1+e^{-cx}}.$$

Der Graph der Sigmoide verändert sich in Abhängigkeit von c. Abb. 10 zeigt die Varianten für $c=1$, $c=2$ und $c=3$. Je größer c ist, desto ähnlicher wird der Graph der Sigmoide dem Graphen der Treppenfunktion, die den Grenzwert der Sigmoide für $c \to \infty$ darstellt. Im weiteren Verlauf setzen wir $c=1$ voraus und nennen die Sigmoide s_1 einfach s.

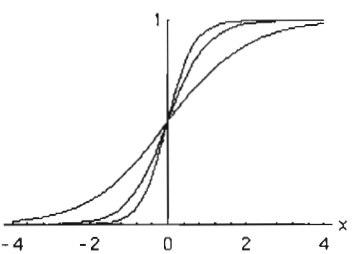

Abb. 10 Drei Sigmoiden (für $c=1$, $c=2$, $c=3$)

Die Sigmoide ist in ihrem ganzen Definitionsbereich differenzierbar. Ihre Ableitung nach x, die wir weiter unten für die Darstellung des Backpropagation-Algorithmus brauchen, lautet:

$$\frac{ds(x)}{dx} = \frac{e^{-x}}{(1+e^{-x})^2} = s(x)(1-s(x)).$$

Durch die Verwendung der Sigmoide als Neuronen-Aktivierungsfunktion wird auch die Fehlerfunktion des Netzes stetig und überall differenzierbar. Denn ein mehrschichtiges Netz mit der Sigmoide als Aktivierungsfunktion berechnet ausschließlich eine Reihe von Funktionskompositionen. Da alle beteiligten Funktionen stetig und differenzierbar sind, hat auch die Fehlerfunktion selbst diese Eigenschaften.

4.2 Backpropagation in Funktionennetzen

Nun behandeln wir Netze, deren Neuronen ausschließlich die Sigmoide als Ausgabefunktion verwenden. Das Lernproblem für solche Netze besteht darin, m vorgegebene n-dimensionale Eingabevektoren $\mathbf{v}_1, \mathbf{v}_2, ..., \mathbf{v}_m$ in m vorgegebene k-dimensionale Ausgabevektoren $\mathbf{t}_1, \mathbf{t}_2, ..., \mathbf{t}_m$ so genau wie möglich abzubilden. Die Menge von Ein- und Ausgabevektorpaaren bildet die Trainingsmenge. Die Genauigkeit der Abbildung wird durch den *quadratischen Fehler* des Netzes definiert. Gibt das Netz die m k-dimensionalen Ausgabevektoren $\mathbf{y}_1, \mathbf{y}_2, ..., \mathbf{y}_m$ aus, wenn die m Vektoren $\mathbf{v}_1, \mathbf{v}_2, ..., \mathbf{v}_m$ eingegeben werden, so ist der quadratische Fehler des Netzes

$$E = \frac{1}{2} \sum_{i=1}^{m} \|\mathbf{t}_i - \mathbf{y}_i\|^2.$$

Ziel des Lernverfahrens ist es, Netzgewichte zu finden, die E minimieren. Nach dem Training werden unbekannte Vektoren in das Netz eingegeben, in der Erwartung, daß es eine gute *Interpolation* durchführt. Das Netz soll automatisch erkennen, ob eine neue Eingabe einem Eingabevektor der Trainingsmenge ähnlich ist, und dann eine ähnliche Ausgabe erzeugen.

Der Backpropagation-Algorithmus dient dazu, ein lokales Minimum der Fehlerfunktion E zu finden. Das Netz wird mit zufälligen Gewichten initialisiert. Danach wird der Gradient der Fehlerfunktion in Abhängigkeit der gegenwärtigen Ge-

wichte berechnet. Neue Gewichte werden durch eine Korrektur in die entgegengesetzte Richtung des Gradienten bestimmt.

Wir zeigen, daß der Backpropagation-Algorithmus eigentlich nur aus der wiederholten Anwendung der *Kettenregel* der Differentialrechnung besteht und beschreiben ihn für den speziellen Fall eines Netzes mit zwei Neuronenschichten.

4.2.1 Berechnungen in einem Backpropagation-Netz

Betrachten wir ein Netz mit n Eingabestellen, ℓ verborgenen und k Ausgabeneuronen. Das Gewicht der Kante zwischen Eingabestelle i und verborgenem Neuron j nennen wir w_{ij}^1. Das Gewicht der Kante zwischen verborgenem Neuron i und Ausgabeneuron j nennen wir w_{ij}^2. Wir behandeln, um die Darstellung zu vereinfachen, ausschließlich Neuronen mit Schwellenwert null. Die Eingabevektoren in der Trainingsmenge werden deswegen um eine Dimension erweitert, so wie dies bei Perzeptronen der Fall war. Die Ausgabe der verborgenen Schicht wird ebenfalls erweitert. Dafür wird, wie Abb. 11 zeigt, die konstante Eingabe 1 mit den Neuronen in der verborgenen und der Ausgabeschicht verschaltet. Das Gewicht zwischen der Eingabe 1 und dem Neuron j in der verborgenen Schicht nennen wir $w_{n+1,j}^1$. Das Gewicht von der Verbindung zwischen 1 und Ausgabeneuron j nennen wir $w_{\ell+1,j}^2$.

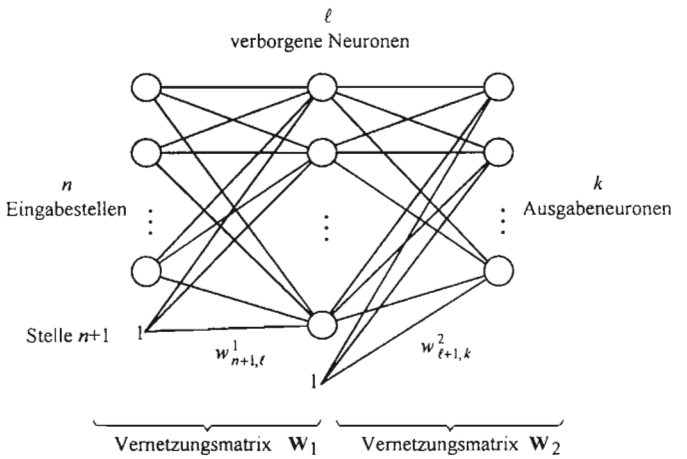

Abb. 11 Struktur eines Backpropagation-Netzes

Zwischen Eingabestellen und verborgener Schicht sind $(n+1)\times\ell$ und zwischen dieser und der Ausgabeschicht $(\ell+1)\times k$ Gewichte vorhanden. Sei $\mathbf{W}_1$ die $(n+1)\times\ell$-Matrix mit den Einträgen w^1_{ij} für $i=1,...,n+1$ und $j=1,...,\ell$. Sei ferner $\mathbf{W}_2$ die Matrix mit den Einträgen w^2_{ij}, für $i=1,...,\ell+1$ und $j=1,...,k$. Der n-dimensionale Eingabevektor $\mathbf{o}=(o_1,o_2,...,o_n)$ wird erweitert und in $\tilde{\mathbf{o}}=(o_1,o_2,...,o_n,1)$ transformiert. Die Erregung des j-ten Neurons in der verborgenen Schicht ist

$$err_j=\sum_{i=1}^{n+1} w^1_{ij}\tilde{o}_i.$$

Die Aktivierungsfunktion ist eine Sigmoide und die Ausgabe o^1_j dieses Neurons deshalb:

$$o^1_j=s\left(\sum_{i=1}^{n+1} w^1_{ij}\tilde{o}_i\right).$$

Die Erregung aller Neuronen in der verborgenen Schicht kann durch das Produkt $\tilde{\mathbf{o}}\mathbf{W}_1$ berechnet werden. Der Vektor $\mathbf{o}^1$ der Ausgabe der Neuronen in der verborgenen Schicht ist dann

$$\mathbf{o}^1 = s(\hat{\mathbf{o}}\,\mathbf{W}_1),$$

wobei wir der Konvention folgen, daß die Sigmoide komponentenweise berechnet wird. Die Erregung der Neuronen in der Ausgabeschicht wird mit dem erweiterten Vektor $\hat{\mathbf{o}}^1 = (o_1^1, o_2^1, ..., o_\ell^1, 1)$ berechnet. Die Ausgabe des Netzes ist der k-dimensionale Vektor $\mathbf{o}^2$, wobei

$$\mathbf{o}^2 = s(\hat{\mathbf{o}}^1\mathbf{W}_2).$$

Diese Notation kann für jede Anzahl von Netzschichten verallgemeinert werden und entspricht einer direkten Berechnung der Netzdynamik mit Mitteln der linearen Algebra.

4.2.2 Die Kettenregel

Ein Backpropagation-Netz entspricht einem Funktionennetz, bei dem eine Kette von Funktionskompositionen berechnet wird. Die Parameter der Berechnung sind die Netzgewichte. Der quadratische Fehler E für eine vorgegebene Trainingsmenge ist in einem Netz mit zwei Neuronenschichten eine Funktion der Netzgewichte w_{ij}^1 und w_{ij}^2. Für die Korrektur der Gewichte muß

$$\bar{\nabla}E = \left(\underbrace{\frac{\partial E}{\partial w_{11}^1}, ..., \frac{\partial E}{\partial w_{n+1,\ell}^1}}_{(n+1)\times\ell \text{ Ableitungen}}, \underbrace{\frac{\partial E}{\partial w_{11}^2}, ..., \frac{\partial E}{\partial w_{\ell+1,k}^2}}_{(\ell+1)\times k \text{ Ableitungen}} \right)$$

berechnet werden. Dies erfordert die mehrmalige Anwendung der Kettenregel der Differentialrechnung. Aus diesem Grund zeigen wir in den folgenden Abschnitten, wie die Kettenregel in Funktionennetzen berechnet werden kann.

Nehmen wir an, daß f und g eindimensionale, reelle, differenzierbare Funktionen sind. Sei die Funktionskomposition $f \circ g$. Die partielle Ableitung von $f \circ g$ nach x ist nach der Kettenregel:

$$\frac{\partial f(g(x))}{\partial x} = \left(\frac{\partial f}{\partial g} g(x)\right) \frac{\partial g}{\partial x}$$

oder auch kurz $(f(g(x)))' = f'(g(x))g'(x)$. Abb. 12 zeigt ein Netz mit zwei Neuronen, wobei deren Ausgabefunktion g bzw. f ist. Die Neuronen besitzen jetzt auch einen linken Teil, in dem die Ableitungen ihrer Aktivierungsfunktionen, also g' und f' berechnet werden. Die Leitungen sind nicht gewichtet. Ein Eingabewert x wird in die Ausgabe $f(g(x))$ verwandelt. Der *Feedforward*-Teil des Verfahrens durchläuft das Netz von links nach rechts unter alleiniger Verwendung von Funktionskompositionen. Im linken Teil der Berechnungselemente wird die Ableitung der Aktivierungsfunktion für die eingetroffene Information berechnet und *gespeichert*. Abb. 12 zeigt den Zustand des Netzes nach der Bearbeitung der Eingabe x.

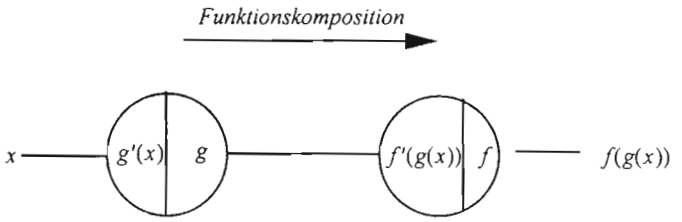

Abb. 12 Feedforward-Schritt

Die *Backpropagation*-Berechnung verläuft von rechts nach links. Angefangen mit dem Traversierungswert 1 wird das Netz rückwärts durchlaufen. An jedem Knoten wird der Traversierungswert mit dem gespeicherten Wert der Ableitung der jeweiligen Aktivierungsfunktion *multipliziert*.

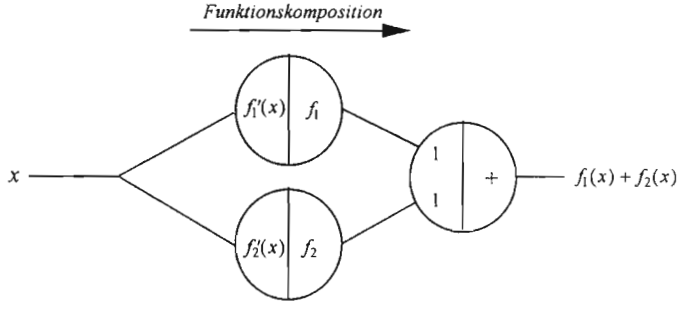

Abb. 14 Addition von Funktionen

Für die Addition der Funktionen f_1 und f_2 gilt

$$\frac{\partial(f_1(x)+f_2(x))}{\partial x} = \frac{\partial f_1(x)}{\partial x} + \frac{\partial f_2(x)}{\partial x}.$$

Dies kann ebenfalls durch Backpropagation berechnet werden. Im *Feedforward*-Schritt wird die Funktionskomposition $f_1(x) + f_2(x)$ berechnet, und in den Knoten werden die ausgewerteten Ableitungen der Aktivierungsfunktionen gespeichert. Beim *Backpropagation*-Schritt wird wieder mit dem Traversierungswert 1 gestartet. Nach dem Plus-Knoten teilt sich die Traversierung in zwei Pfade, die sich später wieder treffen. Für jeden Pfad wird ein Traversierungswert berechnet. Beim Durchlaufen des oberen Pfades wird der Traversierungswert $f_1'(x)$, beim Durchlaufen des unteren wird $f_2'(x)$ berechnet. An der Stelle, an der sich beide Pfade wieder treffen, werden beide Werte addiert. Das Resultat ist die partielle Ableitung von $f_1 + f_2$ nach x, wie in Abb. 15 gezeigt wird.

Die „parallele" Verknüpfung von Funktionen kann auf mehrere Pfade verallgemeinert werden. Durch Induktion kann gezeigt werden, daß die Berechnung der partiellen Ableitung der Addition von beliebig vielen Funktionen durch einen Backpropagation-Schritt geleistet werden kann.

In Backpropagation-Netzen sind nur Funktionskomposition und Addition von Funktionen vorhanden, da die Integrati-

onsfunktion eines jeden Neurons als von der Sigmoide getrennt gedacht werden kann.

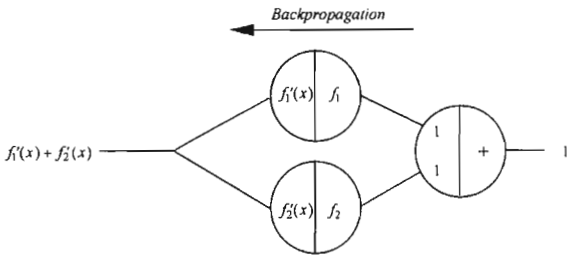

Abb. 15 Backpropagation-Schritt

Eine sehr nützliche Vereinfachung für die Sigmoide ist es, die Ableitung $s'(x)$ als $s(x)(1-s(x))$ darzustellen. Hat ein Neuron im *Feedforward*-Schritt die Ausgabe o berechnet, dann ist der im linken Teil gespeicherte Wert $o(1-o)$.

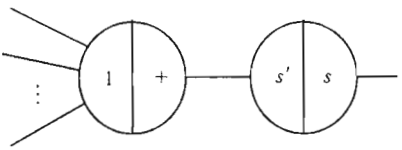

Abb. 16 Integrations- und Ausgabefunktion eines Neurons

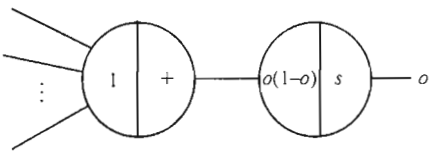

Abb. 17 Gespeicherter Wert bei der Ausgabe o

Auf diese Weise verfügen wir über eine Methode, mit der beliebige partielle Ableitungen einer Komposition von Funktionen berechnet werden können.

4.2.4 Gewichtete Kanten

Gewichtete Kanten können wie die Funktionskomposition behandelt werden (Multiplikation mit einer Konstanten). Sie lassen sich aber auch einfacher darstellen.

Im *Feedforward*-Teil des Verfahrens wird eine Eingabe x über eine gewichtete Kante einfach mit dem Gewicht w der Kante multipliziert. Im Backpropagation-Teil wird genau dasselbe mit dem Traversierungswert gemacht. Das Resultat ist die partielle Ableitung von wx nach x, nämlich w. Interessant ist auch, daß durch Vertauschen von x und w im Netz die *Feedforward*-Berechnung unverändert bleibt; im *Backpropagation*-Schritt wird jedoch die Ableitung von wx nach w, nämlich x, berechnet. In Backpropagation-Netzen interessiert uns die Ableitung der Fehlerfunktion nach den Gewichten w, wobei die Eingabe x als das Gewicht und w als die Eingabe behandelt werden muß (Abb. 18).

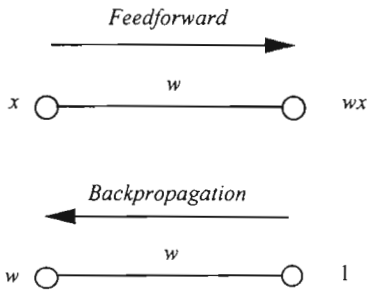

Abb. 18 Behandlung der gewichteten Kanten

Die Regeln für graphbasierte Berechnung der drei oben behandelten Fälle sind hinreichend zum Verständnis des Backpropagation-Algorithmus. Die wichtigste Schlußfolge-

rung, die wir aus unserer Darstellung ziehen können, ist die folgende:

> *Um die partielle Ableitung der von einem Funktionennetz berechneten Funktion nach seiner Eingabe zu berechnen, muß das Netz lediglich rückwärts mit der Anfangseingabe 1 traversiert werden, wobei in jedem Knoten eine Multiplikation mit dem im Feedforward-Schritt in der linken Hälfte gespeicherten Wert stattfindet. Die Information, die über mehrere Pfade an einem Knoten ankommt, wird addiert. Kantengewichte werden wie in Feedforward-Betrieb verwendet.*

Dies gilt auch für Netze, die n reelle Eingaben $x_1, x_2, ..., x_n$ verarbeiten. Die partiellen Ableitungen nach jeder Variablen werden durch die Traversierung der n Subnetze gefunden, die von der Ausgabe bis zu den n Eingabestellen in die umgekehrte Richtung des Informationsflusses vorhanden sind. Diese Methode kann für jede beliebige vorwärtsgerichtete Netzarchitektur verwendet werden.

4.2.5 Die Fehlerfunktion

Um die Fehlerfunktion eines Netzes zu berechnen, kann das Netz selbst erweitert werden. Nehmen wir an, die Trainingsmenge bestehe nur aus dem Ein-/Ausgabepaar der Vektoren $(\mathbf{o},\mathbf{t})$. Das Netz bestehe aus zwei Schichten von Neuronen und einer Schicht von Eingabestellen. Abb. 19 zeigt die letzte Schicht des Netzes und die notwendige Erweiterung, um die Fehlerfunktion für das *target* $\mathbf{t} = (t_1, t_2, ..., t_k)$ zu berechnen. Jedes Neuron i in der Ausgabeschicht berechnet die Sigmoide und erzeugt die Ausgabewerte o_i^2. Die zusätzlichen Neuronen (im Bild als Ellipsen dargestellt) berechnen den quadratischen Fehler für jedes Ausgabeneuron, indem o_i^2 mit t_i verglichen wird. Die Addition der einzelnen Komponentenfehler ergibt den Wert der Fehlerfunktion E.

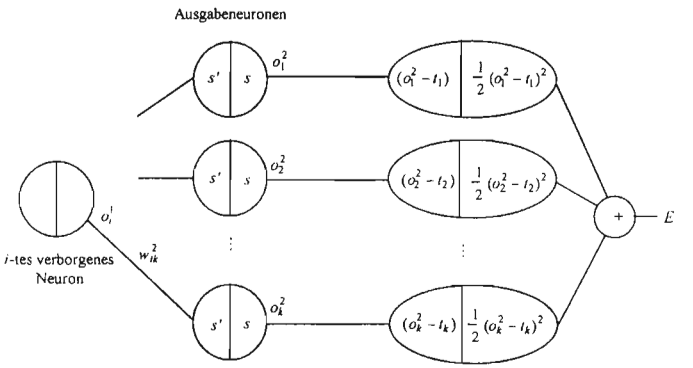

Abb. 19 Erweitertes Netz zur Berechnung der Fehlerfunktion

Die Fehlerfunktion für m Ein- und Ausgabepaare kann berechnet werden, indem das Netz m-mal auf die gezeigte Weise erweitert wird. Die einzelnen quadratischen Fehler werden am Ende zu einem Gesamtfehler addiert.

4.2.6 Schritte des Algorithmus

In diesem Abschnitt wird gezeigt, daß der Backpropagation-Algorithmus ein Gradientenabstiegsverfahren ist, wobei weiterhin ein Netz mit zwei Neuronen-Schichten verwendet wird. Der Schwellenwert der Neuronen ist null. Das Netz wird mit nur *einem* Eingabevektor **o** trainiert und die gewünschte Ausgabe ist der Vektor **t**, wobei sich die Verallgemeinerung des Verfahrens für mehrere Eingabevektoren leicht durchführen läßt. Das Netz besteht wiederum aus n Eingabestellen, ℓ verborgenen und k Ausgabeneuronen. Wir behalten zur Beschreibung der Eingabevektoren, Gewichte und sonstigen Variablen im Netz die Notation der vorherigen Abschnitte bei.

Nach einer zufälligen Initialisierung der Netzgewichte wird der Backpropagation-Algorithmus gestartet. Er besteht aus der Wiederholung folgender vier Schritte:

a) Feedforward-Berechnung
b) Backpropagation bis zur Ausgabeschicht
c) Backpropagation bis zur verborgenen Schicht
d) Korrektur der Gewichte

Der Algorithmus wird gestoppt, wenn der Wert der Fehlerfunktion klein genug geworden ist. Das Netz wird für die Berechnung der Fehlerfunktion E gemäß der Abb. 19 ergänzt. Ziel des Lernverfahrens ist die Minimierung von E, wofür die partiellen Ableitungen von E nach den Gewichten des Netzes zu berechnen sind. Dies geschieht durch die ersten drei Schritte des Algorithmus.

· *Erster Schritt*: Feedforward-Berechnung.

Als erstes wird der Eingabevektor **o** in das Netz eingespeist. Die Information fließt durch das erweiterte Netz, bis am Ausgang der Fehler E berechnet ist. An jedem Knoten wird die ausgewertete Ableitung der Sigmoide gespeichert.

· *Zweiter Schritt:* Backpropagation bis zur Ausgabeschicht

Zu finden sind die partiellen Ableitungen $\partial E/\partial w_{ij}^2$. Dafür wird das Netz von rechts nach links traversiert. Für jedes Gewicht w_{ij}^2 existiert ein Pfad vom Ausgang des Netzes bis zum Ausgabeneuron j zurück (Abb. 20).

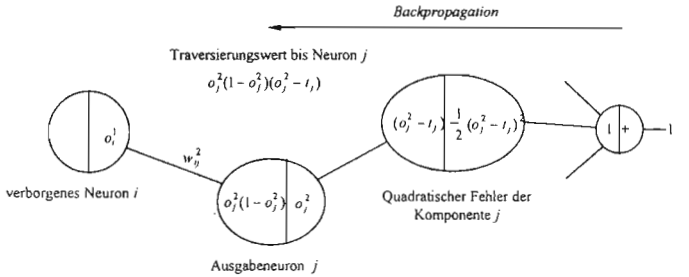

Abb. 20 Ein Pfad im Netz

Auf diesem Pfad ergibt sich der Traversierungswert T_j bis zum Ausgabeneuron j:

$$T_j = o_j^2(1-o_j^2)(o_j^2 - t_j) .$$

Wir definieren den ***rückwärtsverteilten Fehler*** für das Neuron j als $\delta_j^2 = -T_j$. Die gesuchte partielle Ableitung ist dann:

$$\frac{\partial E}{\partial w_{ij}^2} = \left(o_j^2(1-o_j^2)(o_j^2 - t_j)\right)o_i^1 = -\delta_j^2 o_i^1$$

Für diesen letzten Schritt haben wir die Tatsache benutzt, daß das Gewicht einer Kante als Eingabe und die im Feedforward-Schritt berechnete Eingabe, d.h. o_i^1, als das Gewicht der Kante betrachtet werden können.

Abb. 21 zeigt die Kante, der im letzten Schritt gefolgt wurde. An ihren beiden Enden stehen einerseits der Ausgabewert o_i^1 und andererseits der rückwärtsverteilte Fehler δ_j^2. Bei den Gewichten zwischen Eingabestellen und der verborgenen Schicht werden wir eine ähnliche Situation finden.

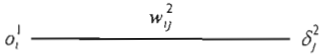

Abb. 21 Rückwärtsverteilter Fehler

· *Dritter Schritt*: Backpropagation bis zur verborgenen Schicht.

Zu finden sind die partiellen Ableitungen $\partial E/\partial w_{ij}^1$. Jedes Neuron j in der verborgenen Schicht ist mit mehreren Neuronen q in der Ausgabeschicht durch Leitungen mit Gewichten w_{jq}^2 für $q = 1,...,k$ verbunden. Der Traversierungswert im Backpropagation-Schritt ist für Neuron j in der verborgenen Schicht (Abb. 22):

$$T_j^1 = -o_j^1(1-o_j^1)\sum_{q=1}^{k} w_{jq}^2 \delta_q^2 \cdot$$

Wir definieren den bis zum verborgenen Neuron j rückwärtsverteilten Fehler als

$$\delta_j^1 = -T_j^1 = o_j^1(1-o_j^1)\sum_{q=1}^{k} w_{jq}^2 \delta_q^2.$$

So ist die gesuchte partielle Ableitung:

$$\frac{\partial E}{\partial w_{ij}^1} = -\delta_j^1 o_i \cdot$$

Der rückwärtsverteilte Fehler kann auf dieselbe Weise für beliebig viele Schichten berechnet werden. Die partielle Ableitung der Fehlerfunktion nach einem Gewicht behält dieselbe analytische Form.

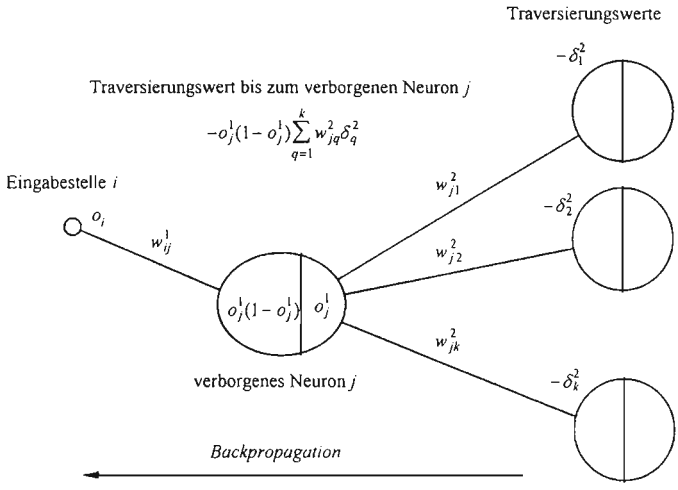

Abb. 22 Pfad bis zur Eingabestelle i

· *Vierter Schritt*: Korrektur der Gewichte

Nach der Berechnung aller partiellen Ableitungen werden die Gewichte so korrigiert, daß immer der negativen Gradientenrichtung gefolgt wird. Eine Lernkonstante γ verändert dabei die Konvergenzgeschwindigkeit des Algorithmus. Die Korrekturen für die Gewichte sind dann:

$$\Delta w_{ij}^2 = \gamma o_i^1 \delta_j^2, \qquad i = 1,...,\ell+1; j = 1,...,k,$$
$$\Delta w_{ij}^1 = \gamma o_i \delta_j^1, \qquad i = 1,...,n+1; j = 1,...,\ell.$$

Dabei sind $o_{n+1} = o_{\ell+1}^1 = 1$ die für die Erweiterung der Vektoren benutzten Konstanten. Es ist wichtig, die Korrekturen erst *nach* der Verteilung des Fehlers im ganzen Netz durchzuführen. Anderenfalls könnte die Berechnung des Gradienten ein falsches Resultat liefern, da sie mit den Korrekturen der Gewichte vermischt wurde. Diesen Fehler machen einige Autoren in ihrer Darstellung des Algorithmus (Aleksander, Morton 1990).

Falls m Eingabevektoren in der Trainingsmenge vorhanden sind, muß der Beitrag $E_1,E_2,...,E_m$ jedes Eingabevektors für den Gesamtfehler E getrennt behandelt werden. Es werden dann die Gewichtskorrekturen ebenfalls getrennt berechnet. Nehmen wir z.B. an, daß für das Gewicht w_{ij}^1 die m Korrekturen $\Delta_1 w_{ij}^1, \Delta_2 w_{ij}^1, ..., \Delta_m w_{ij}^1$ durch Backpropagation berechnet wurden. Danach braucht nur noch ein Korrekturschritt

$$\Delta w_{ij}^1 = \Delta_1 w_{ij}^1 + \Delta_2 w_{ij}^1 + \cdots + \Delta_m w_{ij}^1$$

durchgeführt zu werden. Ist dies der Fall, spricht man von einem *Batch-Verfahren*, das die echte Gradientenrichtung der Gesamtfehlerfunktion für die Gewichtskorrekturen verwendet. Werden allerdings die Korrekturen für die einzelnen Ein- und Ausgabepaare getrennt durchgeführt, spricht man von *On-line-Training*. Die für die Korrekturen verwendete Richtung im Gewichteraum stimmt dann nicht mit der Gradientenrichtung überein. Werden die Eingabebeispiele aber zufällig selektiert, so oszilliert die Korrekturrichtung um den Gradienten, und im *Durchschnitt* wird der maximal absteigenden Richtung auf der Fehlerfläche gefolgt.

5. Fazit

Wir haben in diesem Aufsatz gezeigt, was unter künstlichen neuronalen Netzen verstanden wird, und eine exemplarische Anwendung besprochen. Damit haben wir aber eigentlich nur an der Oberfläche eines sehr breiten Gebietes gekratzt. Außer den vorwärtsgerichteten Netzen, die hier besprochen wurden, gibt es eine große Anzahl alternativer Modelle. Dazu zählen u.a. Hopfield-Netze, zeitverzögerte Netze, das Kohonen-Modell und Fuzzy-Kontroller. Allen diesen Modellen ist aber eines gemeinsam: ihr Substrat ist ein gerichteter Graph von Berechnungselementen. Durch die im Text behandelten Anwendungsbeispiele sollte klar sein, daß neuronale Netze kein

exotisches Berechnungsinstrument sind. Sie sind aus einer biologischen Motivation hervorgegangen, haben sich aber bereits unabhängig gemacht und sind zu einem zusätzlichen Werkzeug im Werkzeugkasten der Informatiker und Mathematiker geworden. Vor allem ihre Fähigkeit, unbekannte Funktionen und Wahrscheinlichkeitsverteilungen approximieren zu können, kann für die statistische Modellierung verwendet werden. Dies befreit niemanden davon, weiterhin Statistik zu lernen, um umfangreiche Datensätze zu verarbeiten, liefert aber eine Systematik, um dies in bestimmten Anwendungen ohne großen Aufwand zu tun. Wir hoffen, daß dieser kurze Aufsatz dazu beiträgt, den Leser für dieses Gebiet zu interessieren, indem das Geheimnis, das hinter der Bezeichnung „neuronale Netze" steckt, gelüftet wird. Die Bezeichnung *Funktionennetze* ist sicherlich mathematisch präziser, aber die ursprüngliche Bezeichnung *neuronale Netze* ist anspruchsvoller, weil sie uns ständig daran erinnert, daß das Verstehen menschlicher Intelligenz das Hauptanliegen des konnektionistischen Unternehmens war und ist.

6. Literatur

Bourlard, H., Morgan, N. (1993), *Connectionist Speech Recognition*, Kluwer.

Hertz, J., A. Krogh und R. Palmer (1991), *Introduction to the Theory of Neural Computation*, Addison-Wesley, Redwood City.

Minsky, M. (1967), *Computation: Finite and Infinite Machines*, Prentice-Hall, Englewood Cliffs.

Minsky, M. und S. Papert (1969), *Perceptrons: An Introduction to Computational Geometry*, MIT Press, Cambridge, Massachusetts.

Rabiner, L., Bing-Hwang, J. (1993), *Fundamentals of Speech Recognition*, Prentice-Hall International, London.

Richard, M. D., Lippmann, R. P. (1991), „Neural Network Classifiers Estimate *a posteriori* Probabilities", Neural Computation, Vol. 3, N. 4, S. 461-483.

Ritter, H., T. Martinetz und K. Schulten (1990), *Neuronale Netze: Eine Einführung in die Neuroinformatik selbstorganisierender Netzwerke*, Addison-Wesley, Bonn.

Rojas, R. (1996), *Neural Networks*, Springer-Verlag, Berlin.

Rumelhart, D. und J. McClelland (1986), *Parallel Distributed Processing*, MIT Press, Cambridge, Massachusetts.