Kapitel 2

Neuronale Netze

Das große Vorbild bei der Entwicklung (künstlicher) Neuronaler Netze in der Informatik war stets die Leistungsfähigkeit des menschlichen Gehirns. Daher wird im Interesse einer besseren Verständlichkeit und zur Nachvollziehbarkeit wichtiger Entscheidungen zuerst das biologische Vorbild erläutert. Anschließend werden zwei bedeutende historische Entwicklungen von künstlichen Neuronalen Netzen vorgestellt. In den letzten beiden Abschnitten werden die für diese Arbeit wichtigsten Definitionen eingeführt und einige Eigenschaften des verwendeten Modells besprochen.

2.1 Das biologische Vorbild

Untersuchungen lassen vermuten, daß unser Gehirn aus etwa 100 Milliarden Neuronen besteht, die untereinander vielfach verknüpft sind. Durch die Fülle von Neuronen und der damit verbundenen großen Zahl möglicher Verschaltungen ist unser Gehirn zu seinen atemberaubenden Leistungen imstande. Wir können unter anderem Lernen, Denken, Handeln, Musik hören und mit anderen Menschen Gespräche führen. Desweiteren ist das Gehirn mitverantwortlich für einfachste Funktionen wie Temperaturregelung, Atemregelung, Auslösen eines Hungergefühls und vieles mehr. Trotz all' dieser Leistungen sind die einzelnen Neuronen, aus denen das Gehirn besteht, überraschend einfach aufgebaut. Die hohe Leistungsfähigkeit entstand im Evolutionsprozeß zum Großteil erst durch die starke Vergrößerung des Gehirns (mehr Neuronen und dadurch mehr Verknüpfungsmöglichkeiten[1]) und die Fähigkeit zu lernen.

[1] Untersuchungen von [Valentin Braitenberg and Almut Schüz, 1989] sprechen von etwa 100 Billionen (10^{14}) Synapsen allein im Cortex.

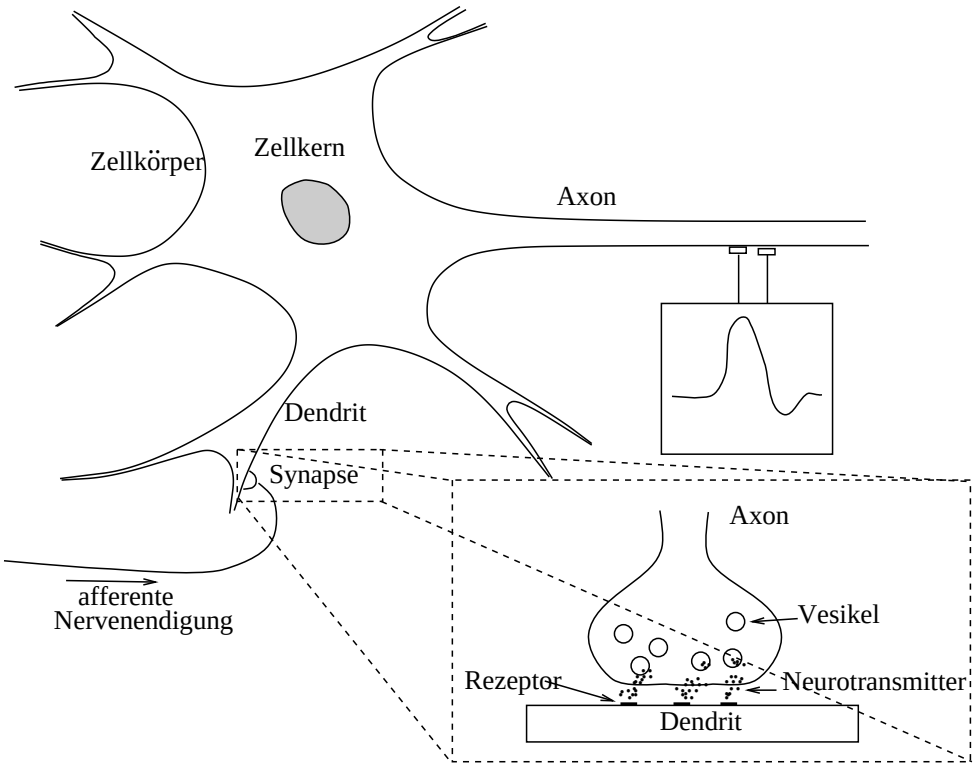

Abbildung 2.1: Schematische Darstellung eines natürlichen Neurons nach [Katz, 1987, Seite 3] und [Ornstein und Thompson, 1986, Seite 79].

Obwohl aus dem menschlichen Gehirn mehr als 100 verschiedene Arten von Neuronen bekannt sind, haben doch alle einen vergleichbaren Aufbau, wovon ein einzelnes in Abbildung 2.1 schematisch dargestellt ist. Ein Neuron besteht im Wesentlichen aus Zellkörper (*Soma*), Nervenfaser (*Axon*), *Dendriten* (bis zu etwa 10 000 je Neuron) und *Synapsen*. Dabei ist wichtig, daß jedes Neuron nur ein einziges sich verzweigendes Axon und viele Dendriten besitzt. Über das Axon kann ein Neuron ein elektrisches *Aktionspotential* zu den Dendriten anderer Neuronen weiterleiten. Die Verbindungsstelle von Axon und Dendrit heißt Synapse. Die Synapse ist ein recht kompliziertes Gebilde. Am Ende einer Axonverzweigung sind in einer kleinen Verdickung viele kleine *Vesikel* eingelagert, in denen sich chemische Botenstoffe (*Neurotransmitter*) befinden. Bei Eintreffen eines Aktionspotentials öffnen sich die Vesikel, sodaß die Neurotransmitter durch die *präsynaptische Membran* über den *synaptischen Spalt* zu den Rezeptoren gelangen. Die Anlagerung von Neurotransmittern an Rezeptoren hat eine elektrische Potentialverschiebung im Zellkörper des empfangenden Neurons zur Folge. Bei genügend großer Potentialverschiebung in die richtige Richtung ist dieses empfangende Neuron seinerseits in der Lage, ein eigenes Aktionspotential über das eigene Axon „auf die Reise zu schicken“. Die Richtung der resultierenden Potentialverschiebung hängt von der Konstruktion der *postsynaptischen Membran* ab. Es gibt hemmende (inhibitorische) und erregende (exzitorische) Synapsen. Die Stärke der Potentialverschiebung ist im wesentli-

chen von der Entfernung der Synapse zum Zellkern und von der Größe der Synapse selbst abhängig. Größere Synapsen haben in der Regel auch einen größeren Vorrat an Vesikeln und Rezeptoren, wodurch eine größere Potentialverschiebung bewirkt werden kann.

Lernvorgänge im Gehirn sind nach [Katz, 1987] und [Valentin Braitenberg and Almut Schüz, 1989] an Veränderungen der Synapsen erkennbar. Einerseits ist ein „Verdicken“ von häufig verwendeten Synapsen und andererseits ein Verkümmern wenig verwendeter Synapsen oder ganzer Axonverästelungen (ein einziger Axon-Dendrit Übergang) zu beobachten. Neuere Untersuchungen in der Molekularbiologie lassen vermuten, daß Lernvorgänge ähnlich der Hebb'schen Lernregel vollzogen werden. Die Hebb'sche Lernregel besagt nach [Hinton, 1989, Seite 215], daß genau dann eine Verbindung verstärkt wird, wenn beide beteiligten Neuronen gleichzeitig oder zumindest fast gleichzeitig ein Aktionspotential senden. Genauere Modellbeschreibung der Lernvorgänge sind jedoch zum gegenwärtigen Zeitpunkt noch nicht bekannt.

2.2 Historischer Überblick

In diesem Abschnitt wird ein kurzer Überblick über zwei grundlegende historische Entwicklungen von Neuronalen Netzen in der Informatik gegeben. Dabei werden einige Begriffe verwendet, die erst im nächsten Abschnitt genau erklärt werden. Für das Verständnis dieses historischen Überblicks sollten jedoch intuitive Vorstellungen ausreichen. Im Zweifelsfalle muß der Leser einen Vorgriff auf einzelne Definitionen aus Abschnitt 2.3 vornehmen.

2.2.1 Das Perceptron

Rosenblatt (siehe [Minsky und Papert, 1969]) beschäftigte sich mit einstufigen Systemen, die er Perceptron nannte und aus je einer Ein- und Ausgabeschicht bestehen. In jeder Schicht befindet sich eine bestimmte Anzahl von Units, wobei jede Unit der Eingabeschicht über eine Gewichtung mit jeder Unit der Ausgabeschicht verbunden ist. Jede Unit kann nur die beiden Zustände *Aktiv* und *Nichtaktiv* annehmen, die mit den Symbolen (Zahlenwerten) 1 und 0 gekennzeichnet werden. Die Entscheidung welcher der beiden Zustände angenommen werden soll, wird durch den Vergleich der Eingabe mit einem Schwellwert bestimmt. Die Eingabe ist die Summe, bestehend aus dem Zustand der Units in der Eingabeschicht, multipliziert mit der entsprechenden Gewichtung zu dieser Unit. Durch die Schwellwertfunktion wird eine „alles oder nichts“ Klassifikation vorgenommen.

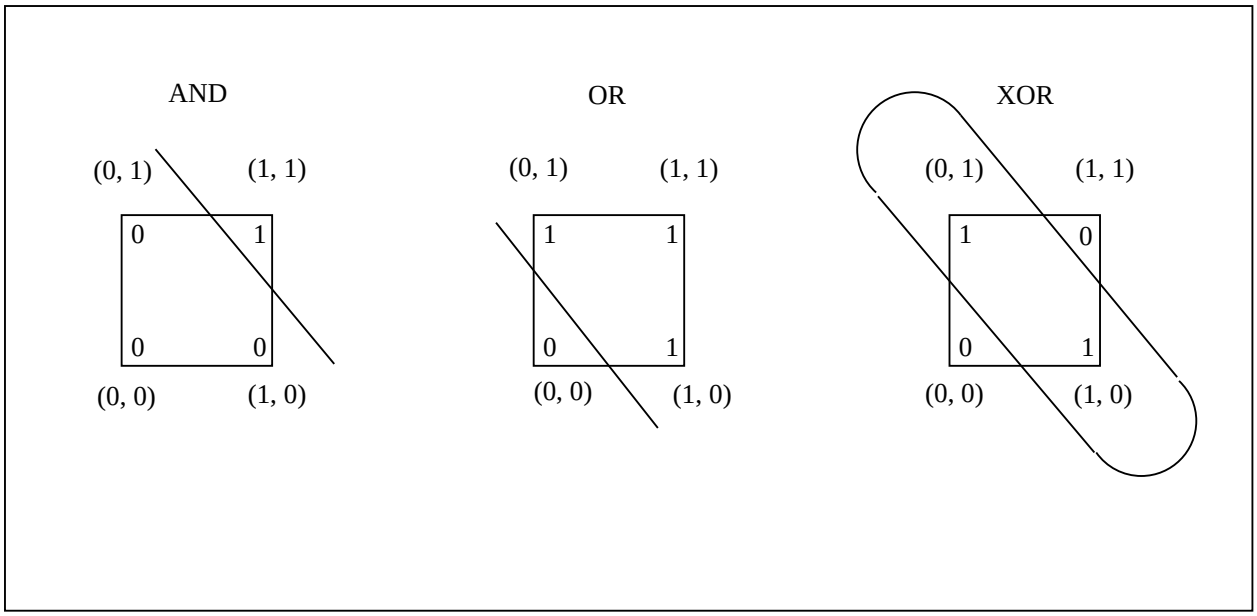

Abbildung 2.2: Beispiele zur linearen Separierbarkeit zweidimensionaler Funktionen nach [McClelland und Rumelhart, 1988a, Seite 124].

Für diese einstufigen *Perceptrons* entwickelte Rosenblatt eine Lernregel; die *perceptron convergence procedure* (siehe zum Beispiel [Ossen, 1990, Seite 22] oder [McClelland und Rumelhart, 1988a, Seite 123]). Mit Hilfe dieser perceptron convergence procedure können durch wiederholtes Anlegen der Eingabedaten und Vergleich der errechneten mit den erwarteten Ausgaben die einzelnen Gewichtungen und Schwellwerte systematisch so geändert werden, daß das Netz die geforderten Ausgaben berechnen kann, sofern dies überhaupt möglich ist. Die Perceptrons können mit Hilfe der besprochenen Lernregel einfache logische Funktionen erlernen; doch scheitern sie schon an der Exklusiv-Oder Funktion.

Das Scheitern des Perceptrons an der Exklusiv-Oder Funktion läßt sich entweder durch Widerspruch beweisen (siehe zum Beispiel [McClelland und Rumelhart, 1988a, Seite 123] oder [Linden, 1989, Seite 21]) oder durch eine geometrische Interpretation veranschaulichen. Bei einer einzigen Unit in der Aus- und zwei Units in der Eingabeschicht können nur diejenigen Funktionen „gelernt“ werden, deren verschiedene Ausgaben in der Ebene durch eine Gerade separiert werden können. Dieser Sachverhalt ist in Abbildung 2.2 dargstellt. Es ist erkennbar, daß bei den ersten beiden Aufgaben diese Trennung durch eine Gerade möglich ist, bei der dritten (rechtes Teilbild) jedoch nicht.

Diese engen Grenzen, nach denen das Perceptron nur linear separierbare Funktionen erlernen kann, lassen sich nur durch Hinzunahme von Zwischenschichten überwinden. Für derartige mehrstufige Systeme konnte jedoch bisher keine Lernregel entwickelt werden, sodaß zwischenzeitlich das Perceptron an Bedeutung verlor. Erst durch die

Entwicklung von Lernverfahren wie Backpropagation, die nach der Gradientenmethode arbeiten, können mehrschichtige Netze zielgerichtet verändert werden.

2.2.2 Boltzmann Maschine

Die nach Boltzmann benannte Maschine[2] ist ein Neuronales Netz, dem ein *thermodynamisches Modell* zugrunde liegt und das vorzugsweise bei physikalischen Problemen verwendet wird. Als thermodynamisches Modell werden Vielteilchensysteme der Natur betrachtet, deren Verhalten mittels statistischer Methoden beschreibbar ist. Ein Beispiel dafür ist der Abkühlungsprozess eines flüssigen Kristalls. Bei vorheriger Erwärmung und anschließendem Abkühlen verlangsamen sich die spontanen Eigenbewegungen der Atome, und es läßt sich beobachten, daß sich die Atome in einem regelmäßigen Kristallgitter anordnen. Dieses Kristallgitter entspricht einem minimalem Energiezustand, oder kommt diesem zumindest sehr nahe. Dieses Energieminimum wird jedoch nur erreicht, wenn der Abkühlungsvorgang *genügend* langsam vor sich geht; *genügend* langsam ist dabei abhängig vom verwendeten Material.

Die Boltzmann Maschine besteht aus Units, die mittels Gewichtungen verbunden sind. Jede Unit kann wie beim Perceptron nur die Zustände 0 und 1 annehmen. Da jede Unit mit jeder anderen verbunden sein kann, kann man nicht davon reden, daß Informationen innerhalb des Netzes von Schicht zu Schicht geschickt werden. Vielmehr wird für eine Boltzmann Maschine folgende *Kostenfunktion* definiert, die auch *Globale Energie* genannt wird:

$$E = -\sum_{i<j} s_i \cdot w_{ij} \cdot s_j \quad , \tag{2.1}$$

wobei s_i und s_j den Zustand (der Wert mit Betrag null oder eins) der Unit i bzw. j beschreibt, und w_{ij} die Stärke der Gewichtung zwischen den beiden Units i und j angibt.

Die Zustände aller Units müssen so gewählt werden, daß die nach Gleichung (2.1) definierte Energie ihr globales Minimum annimmt; davon sind natürlich all diejenigen Units ausgenommen, deren Aktivation von der Umgebung vorgegeben (festgehalten) werden. Die Auswahl der Zustände ist unter Umständen eine recht mühsame Aufgabe, da bei n Units 2^n Kombinationen von Zuständen existieren. Um nicht alle Kombinationen testen zu müssen, wird der Abkühlungsvorgang künstlich nachgeahmt. Bei diesem

[2] Eine genaue Darstellung der Boltzmann Maschine ist zum Beispiel in [Fahlman und Hinton, 1987, Seite 106-108], [Hinton, 1989, Seite 209-214] oder [Rumelhart *et al.*, 1986a, Kapitel 7] zu finden.

simulated annealing wird für jede Unit ihr lokaler Energiebeitrag bestimmt.

$$\Delta E_i = E_{i\ \text{off}} - E_{i\ \text{on}} = \sum_k w_{ik} s_k \quad . \tag{2.2}$$

Die notwendige Eigenbewegung wird durch eine Wahrscheinlichkeit p_i für den Zustand der Unit i realisiert, die sich aus dem lokalen Energiebeitrag nach Gleichung (2.2) und der Temperatur T wie folgt berechnet:

$$p_i = \frac{1}{1 + e^{\frac{-\Delta E_i}{T}}} \quad . \tag{2.3}$$

Diese Wahrscheinlichkeit p_i gibt an, daß sich Unit i bei Temperatur T und lokalem Energiebeitrag ΔE_i im Zustand 1 (on) befindet. Beim *simulated annealing* wird erst eine relative hohe Temperatur verwendet, die in einigen Zyklen stark reduziert wird, bis sich die Boltzmann-Maschine im thermodynamischen Gleichgewicht bei Temperatur $T = 1$ befindet. Innerhalb eines Zyklus wird für jede Unit überprüft, welchen Zustand sie nach Gleichung (2.3) annehmen muß. Die Geschwindigkeit, mit der die Tempereatur abgesenkt wird, ist eine kritische Größe. Bei zu schnellem Absenken ist der endgültige Zustand unter Umständen weit vom globalen Energieminimum entfernt und bei zu langsamen Absenken wird zu viel Zeit verbraucht.

Zu der Boltzmann-Maschine ist auch eine Lernregel definiert, die aus den beiden Phasen P^+ und P^- besteht. Nach dieser Lernregel wird zunächst in P^+ im thermodynamischen Gleichgewicht für jede Gewichtung w_{ij} über mehrere Zyklen der Prozentsatz p_{ij}^+ gemessen, bei dem die beide beteiligten Units i und j zur gleichen Zeit den Zustand 1 annehmen. In einer zweiten Phase P^- wird diese Messung für den Prozentsatz p_{ij}^- wiederholt. Bei dieser Phase werden jedoch die Units der Ausgabeschicht nicht wie vorher auf festen Werten gehalten, sondern die beobachteten Zustände mit den erwarteten verglichen, sodaß die Resultate für p_{ij}^+ und p_{ij}^- in der Regel verschieden sind. Nachdem alle Werte für p^+ und p^- ermittelt wurden, werden alle Gewichtungen um ein Vielfaches der Diferenz $p_{ij}^+ - p_{ij}^-$ wie folgt verändert:

$$\Delta w_{ij} = \varepsilon \cdot \left(p_{ij}^+ - p_{ij}^-\right) \quad . \tag{2.4}$$

Es läßt sich zeigen, daß für genügend kleine ε die obige Lernregel konvergiert.

Durch Festhalten in der Phase P^+ und den Vergleich in der Phase P^- wird der Unterschied zwischen erwartetem und beobachtetem Verhalten ermittelt und als Einflußgröße festgehalten. Erst wenn die beobachteten Zusände mit den erwarteten übereinstimmen, sind beide Messungen gleich und die Gewichtungen werden nicht mehr verändert.

Der große Nachteil der Boltzmann Maschine ist der hohe Rechenaufwand. Mit der obigen Lernregel werden zwar akzeptable Werte erreicht, doch werden für das Abkühlen sehr viele Zyklen benötigt. Die weiter unten beschriebenen Backpropagation Netze benötigen für vergleichbare Probleme etwa so viel Zeit, wie die Boltzmann Maschine für ein einmaliges Abkühlen.

2.3 Definitionen

Im letzten Abschnitt wurden zwei wichtige historische Entwicklungen Neuronaler Netze vorgestellt und deren Nachteile aufgezeigt. Eine deutliche Verbesserung stellen die aus der Literatur bekannten Backpropagation oder *feed-forward* Netze dar. Da sich diese Arbeit mit der Verbesserung von Lernverfahren beschäftigt, die nach der Gradientenmethode arbeiten, verbindet die folgende Beschreibung allgemeingültige Aspekte mit den Besonderheiten (Spezialisierungen), die durch den Schwerpunkt dieser Arbeit (notwendigerweise) entstehen.

Auch bei den Backpropagation Netzen sind einzelne Units zentrales Kernstück. Ausgehend vom biologischen Vorbild wird hier ein stark vereinfachtes Modell verwendet. Dieses stark vereinfachte Modell ist notwendig, da erstens eine detaillierte Nachbildung in Hard- und Software viel zu aufwendig und daher unpraktikabel ist, und zweitens der aktuelle Forschungsstand in Neurobiologie und Medizin noch nicht alle Mechanismen hinreichend genau kennt und somit nicht deren Effekte durch gegenseitige Wechselwirkung beschreiben kann.

Ausgehend vom biologischen Modell wird in diesem Abschnitt zuerst ein in der Informatik weit verbreitetes Modell einer einzelnen Unit entwickelt, und anschließend werden die durch die Zusammenschaltung mehrerer Units entstehenden topologischen Aspekte geklärt. Bei der Begriffsbildung wurde vesucht, weitestgehend eine deutschsprachige Terminologie durchzuhalten, die sich stark an die Normungsvorschläge der GMD[3] hält. Diese Normungsvorschläge sind im wesentlichen Übersetzungen der englischsprachigen Terminologie, wie sie in [Rumelhart *et al.*, 1986a] zusammengefaßt sind.

2.3.1 Die Unit

In der Literatur werden je nach Anwendung und Forschungsrichtung teilweise sehr unterschiedliche Modelle verwendet. Trotz aller Unterschiede haben die meisten Modelle

[3]Gesellschaft für Mathematik und Datenverarbeitung mbH, Sankt Augistin, Forschungsgruppe Mühlenbein, Linden, Kindermann und andere.

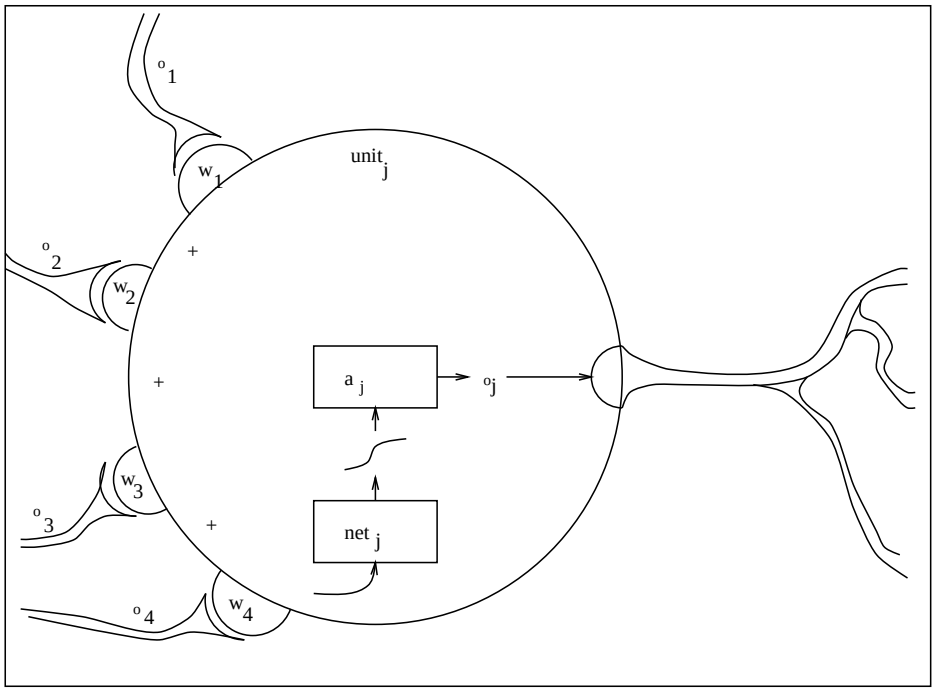

Abbildung 2.3: Das Modell einer einzelnen Unit.

sehr viele Gemeinsamkeiten, die in diesem Abschnitt beschrieben werden.

Der gundlegende Aufbau einer Unit ist in Abbildung 2.3 wiedergegeben. Im einzelnen besitzt eine Unit folgende Eigenschaften:

Aktivation: Jede Unit hat zu einem gegebenen Zeitpunkt t einen eindeutigen Aktivationswert a_i, der einem inneren Zustand entspricht. Durch die verwendete *Aktivierungsfunktion* $f(x)$ liegt der Wertebereich im Intervall $(0, 1)$. Bei anderen Modellen sind auch andere Wertebereiche denkbar.

Ausgabefunktion: Durch die Ausgabefunktion wird die Aktivation a_i auf den Ausgabewert o_i abgebildet. Dieser Ausgabewert ist nach außen hin sichtbar und kann von anderen Units als eingehende Information genutzt werden. Bei den hier betrachteten Units ist die Ausgabefunktion die identische Abbildung, sodaß Ausgabewert und Aktivation immer identisch sind.

Gewichtung: Verschiedene Units eines Neuronalen Netzes sind über Gewichtungen[4] miteinander verbunden. Über diese Gewichtungen machen sich verschiedene

[4]Hier wird kein Unterschied zwischen Vorhandensein bzw. Nichtvorhandensein einer Verbindung und der Stärke dieser Verbindung gemacht. Nichtvorhandensein einer Verbindung wird durch eine Gewichtung mit Betrag null realisiert. Eine Gewichtung mit Betrag ungleich null bedeutet immer, daß auch eine entsprechende Verbindung vorhanden ist.

Units ihre Aktivation gegenseitig bekannt, sodaß es dadurch zu einer Informationsübertragung oder einem Informationsaustausch kommt. Die Informationsübertragung geschieht derart, daß eine Unit j ihre Aktivation über die Gewichtung w_{ij} zur Unit i „sendet". Bei der Vergabe der Indizes einer Gewichtung w_{ij} ist die Reihenfolge Zielunit, Quellunit zu beachten. In einigen Modellen wie zum Beispiel der Boltzmann Maschine geschieht die Informationsübertragung über ein und dieselbe Gewichtung in beiden Richtungen. Aus Gründen der bequemeren Betrachtungsweise werden manchmal alle Gewichtungen w_{ij} in der Adjazenzmatrix $\mathbf{W}$ bzw. zu einem Vektor $\vec{w} = (w_{00}, w_{01}, \cdots, w_{0n}, \cdots, w_{mn})^T$ zusammengefaßt. Bezüglich der Aktivation haben Gewichtungen mit negativem Vorzeichen *inhibitorischen* (hemmenden) und mit positivem Vorzeichen *exzitorischen* (verstärkenden) Einfluß.

Netinput: Der Netinput net_i ist eine wichtige Zwischengröße innerhalb einer Unit i. Er ist die gewichtete Summe aller eintreffenden Aktivationen anderer Units und wird üblicherweise wie folgt berechnet: $net_i = \sum_j a_j w_{ij}$.

Aktivierungsfunktion: Der funktionelle Zusammenhang von Aktivation a_i und Netinput net_i ist durch die Aktivierungsfunktion gegeben. Im allgemeinen Fall wird durch $a_i^{t+1} = f(a_i^t, net_i)$ die Aktivation zum Zeitpunkt[5] $t+1$ aus Netinput und aktueller Aktivation berechnet. Derartige zeitvariante Modelle werden hier jedoch nicht behandelt. Im Rahmen dieser Arbeit kommen nur zeitinvariante Aktivierungsfunktionen der Form $a_i = f(net_i)$ zur Anwendung. Durch die Aktivierungsfunktion ist eine Klassifikation in *lineare* und *nichtlineare* Systeme gegeben. In einem linearen System ist die resultierende Aktivation ein Vielfaches des Netinput. Bei den hier betrachteten Units wird üblicherweise die sigmoidale Funktion $f(x) = \frac{1}{1+e^{-x}}$ als Aktivierungsfunktion benutzt. Durch diese spezielle Aktivierungsfunktion wird jeder Wert des Netinput auf eine Aktivation im Intervall $(0, 1)$ abgebildet.

Schwellwert: Schon sehr einfache Versuche oder folgendes kleines Gedankenexperiment zeigt, daß durch das bisher beschriebene Modell die mögliche Funktionalität noch sehr eingeschränkt ist:

> Betrachtet man ein kleines Netz mit nur zwei Units, die mittels einer einzigen Gewichtung verbunden sind, und fordert, daß eine Aktivation mit Betrag null bzw. eins auf ihren dualen Wert in der anderen Unit abgebildet werden soll, dann findet man keine Lösung. Aus der Elektrotechnik ist für derartige Probleme der Schwellwert θ bekannt. Daher wird ähnlich wie bei der Boltzmann Maschine der Netinput üblicherweise um den Schwellwert θ_i wie folgt erweitert: $net_i = \sum_j a_j w_{ij} + \theta_i$. Jetzt kann

[5] Durch den hochgestellten Index t wird die zeitliche Abhängigkeit einer Größe ausgedrückt. Wenn t den Zustand zu einem festen Zeitpunkt angibt, dann bezeichnet $t+1$ den Folgezustand. Im weiteren wird dieser Zeitindex nur angegeben, wenn es im Zusammenhang zu Unklarheiten kommen könnte.

die gestellte Aufgabe mittels eines großen Schwellwertes und einer Gewichtung mit doppelt so großem negativen Betrag gelöst werden.

Diese Form eines Schwellwertes führt zu einer neuen Größe in der Gleichung des Netinput. Um diese Größe zu eliminieren, ohne jedoch wieder die Funktionalität zu verringern, kann man innerhalb des Netzes eine **Bias-Unit** einführen, die immer die konstante Aktivation eins besitzt. Dadurch kann der jeweilige Schwellwert θ_i durch eine Gewichtung gleichen Betrages zu dieser Bias-Unit ersetzt werden. Durch den gleichen Betrag und die konstante Aktivation haben Schwellwert und Gewichtung zur Bias-Unit die gleiche Funktionalität; jedoch ist die Gleichung für den Netinput homogen, da wie ursprünglich nur die gewichtete Summe gebildet werden muß. Da die Bias-Unit nur von „technischer Bedeutung" ist, wird sie in der Regel in einer graphischen Darstellung eines Netzes fortgelassen.

Da die bisherigen Definitionen und besprochenen mathematischen Zusammenhänge innerhalb dieser Arbeit von grundlegender Bedeutung sind, werden diese wie folgt mathematisch zusammengefaßt:

$$\begin{aligned}
\text{Netinput} \quad & \text{net}_i = \sum_j w_{ij} a_j && (2.5)\\
\text{Aktivierungsfunktion} \quad & f(x) = \frac{1}{1+e^{-x}} && (2.6)\\
\text{Aktivation} \quad & a_i = f(\text{net}_i) && (2.7)\\
\text{Ausgabe} \quad & o_i = a_i \quad . && (2.8)
\end{aligned}$$

2.3.2 Topologie

2.3.2.1 Schichten

In der Regel besteht ein Neuronales Netz aus mehreren Units, die durch Gewichtungen mehr oder minder stark miteinander verknüpft sind. Die daraus resultierenden Topologien können sehr verschiedenartig sein. Diese reichen von einfach bis total verknüpften Netzen. Es ist nützlich, ein Netz als eine Abfolge von Schichten S_0 bis $S_n (n \geq 1)$ aufzufassen. Im Falle klassischer *feed-forward networks* (vergl. auch Abbildung 2.4) gelten für Units und Gewichtungen folgende Bedingungen:

1. Eine Unit gehört genau einer Schicht S_i an.

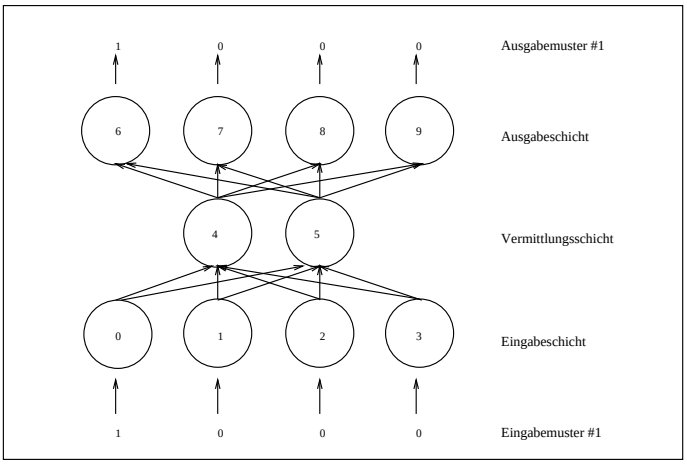

Abbildung 2.4: Ein einfaches *feed-forward network*.

2. Alle Gewichtungen, die von der Schicht S_k ausgehen, enden in der Schicht S_{k+1}.

Durch den räumlichen Aufbau und die Funktionalität wird zwischen folgenden drei verschiedenen Schichtarten unterschieden:

Eingabeschicht: Die Schicht S_0 heißt Eingabeschicht (*input-layer*). Keine Unit dieser Schicht besitzt Gewichtungen, über die sie Information erhält. Sie haben die Funktion von (biologischen) Rezeptoren, indem sie durch Setzen ihrer Aktivation auf gegebene Werte Information aus der Umgebung an das Netz weiterreichen. Units dieser Schicht werden auch *Eingabeunits* genannt.

Vermittlungsschicht: Jede Schicht von S_1 bis S_{n-1} befindet sich zwischen Ein- und Ausgabeschicht und wird Vermittlungsschicht (*hidden-layer*) genannt. Die Aktivation jeder einzelnen Unit dieser Schichten kann von der Umgebung weder unmittelbar verändert noch beobachtet werden.

Ausgabeschicht: Die Schicht S_n heißt Ausgabeschicht (*output-layer*). In dieser Schicht werden die Ergebnisse des Netzes nach außen hin sichtbar gemacht; sie hat die Funktion von (biologischen) Aktoren. Units dieser Schicht werden auch *Ausgabeunits* genannt.

In dieser Arbeit wird von *tieferen* und *höheren* Schichten gesprochen. Das beruht auf der Konvention, daß die Schichten eines Netzes in der graphischen Darstellung von unten nach oben in der Reihenfolge: Eingabeschicht, Vermittlungsschicht, Ausgabeschicht angeordnet sind.

2.3.2.2 Nichtrekurrent vs. rekurrent

Eine wichtige Klassifikation von Netzen ist die Einteilung in rekurrente und nichtrekurernte Netze. Nichtrekurrente Netze sind derart konstruiert, daß sie keine Rückkopplungen durch vorhandene Gewichtungen enthalten. Ein Vertreter dieser nichtrekurrenten Netze ist das gebräuchliche *feed-forward network*. Da die Verbesserung von Lernverfahren, die nach der Gradientenmethode arbeiten, Gegenstand dieser Arbeit ist und diese auf feed-forward networks angewendet werden, werden innerhalb dieser Arbeit ausschließlich nichtrekurrente Netze (feed-forward networks) behandelt. Ein nichtrekurrentes Netz wird wie folgt definiert:

Definition: Ein nichtrekurrentes Netz kann derart in Schichten eingeteilt werden, daß alle Gewichtungen, die von einer Unit der Schicht i ausgehen, in einer Unit der Schicht j enden, für die $j > i$ gilt; dabei muß nicht notwendigerweise $j = i + 1$ gelten.

Diese Definition besagt, daß bei einem gegebenen Netz eine Umsortierung aller Units in Schichten derart möglich ist, daß jede Gewichtung die aus einer Schicht herausführt, in einer höheren Schicht endet. Bezüglich der Adjazenzmatrix $\mathbf{W}$ bedeutet dies, daß alle Elemente der unteren Dreiecksmatrix mit Werten ungleich null belegt sein können. Ferner besitzen alle Units einer gemeinsamen Schicht i in den gleichen Spalten einen Wert ungleich null.

2.3.2.3 Short cuts

Für feed-forward networks gilt, daß eine Gewichtung, die von einer Unit der Schicht i ausgeht auch bei einer Unit der Schicht $i + 1$ endet. In der Adjazenzmatrix $\mathbf{W}$ ist dies daran zu erkennen, daß zwei Units aus unterschiedlichen Schichten in keiner gemeinsamen Spalte Elemente mit Werten ungleich null besitzen. Bei dieser Art von Netzen wird die Information von Schicht zu Schicht weiterverarbeitet. Versuche von Lang und Witbrock [1989] haben gezeigt, daß eine derartig eingeschränkte Topologie für verschiedene Aufgabenstellungen nicht leistungsfähig genug ist. Sie zeigten, wie durch Einführen von *short cuts* diese Leistungsfähigkeit in ihrem Experiment deutlich gesteigert werden konnte. Dies ist auch schon bei sehr einfachen Aufgabenstellungen wie beispielsweise der Exklusiv-Oder Funktion (siehe auch Abschnitt 4.2) der Fall. Unter short cuts sind Gewichtungen zu verstehen, die von Units der Schicht i ausgehen, mindestens eine Vermittlungsschicht überspringen und in Units der Schicht $j > i + 1$ enden. Die Verwendung von short cuts führt in der Adjazenzmatrix $\mathbf{W}$ dazu, daß **alle** Elemente der unteren Dreiecksmatrix mit Werten ungleich null belegt sein können.

Die deutliche Verbesserung der Leistungsfähigkeit durch short cuts kann wie folgt plausibel gemacht werden: Das Ergebnis eines Lernvorgangs in Netzen mit mehr als einer Vermittlungsschicht kann man so deuten, daß in der untersten Vermittlungsschicht „primitive“ *micro-features* erkannt werden, die in weiter oben liegenden Vermittlungsschichten immer abstrakter werden. Das Erkennen von abstrakteren Merkmalen kann durch Verwendung primitiverer Merkmale aus weiter unten liegenden Vermittlungsschichten oder sogar aus der Eingabeschicht stark vereinfacht werden.

2.3.3 Environment

Besondere Vorzüge Neuronaler Netze sind unter anderem die außerordentlich schnelle Signal- und Informationsverarbeitung durch größtmögliche Parallelität und die Fähigkeit, anhand von Beispielen zu lernen. Sie werden unter anderem in der Signalvorverarbeitung (zum Beispiel visueller Cortex[6] bei Lebewesen und Erkennen von Primitiven in *computer graphics*), Informationsverarbeitung (zum Beispiel innerhalb eines klassischen KI Systems) und in der Signalendverarbeitung (zum Beispiel Transformation von Steuerwerten in Signalpegel zur Steuerung eines Roboters) eingesetzt. Da also ein Neuronales Netz wie jedes andere System nur innerhalb einer bestimmten Umgebung sinnvoll arbeitet, müssen auch die Schnittstellen zu dieser Umgebung betrachtet werden. Die dafür notwendigen Begriffe und Definitionen werden in diesem Abschnitt behandelt.

2.3.3.1 Kodierung

In Abbildung 2.5 ist die Einbettung eines Neuronalen Netzes in ein symbolverarbeitendes, klassisches (KI) System skizziert. Die beiden wichtigsten Punkte in diesem Zusammenhang sind die jeweiligen Schnittstellen. Es ist deutlich ersichtlich, daß an der Eingabeseite die auftretende Information in einen Zahlenvektor *I kodiert* werden muß, um diese Information dem Netz bekannt machen zu können. Auf der anderen Seite muß die Ausgabe des Netzes entsprechend *dekodiert* werden. Für die Art der Kodierung Cod_I und Dekodierung Cod_O^{-1} hat der Systementwickler zu sorgen. In Abschnitt 2.4.2 werden einige gebräuchliche Kodierungen vorgestellt. Erst durch diese Kodierung/Dekodierung erhalten die Aktivationen der einzelnen Units eine Semantik.

[6] Die Großhirnrinde des menschlichen Gehirns heißt *Cortex* und ist in verschiedene rezeptive Felder unterteilt. In jedem rezeptiven Feld treffen alle Informationen eines Sinnesorgans zusammen und werden hier einer Vorverarbeitung unterzogen. Beispielsweise werden im visuellen Cortex aus den Informationen der Augen Linien und Flächen erkannt.

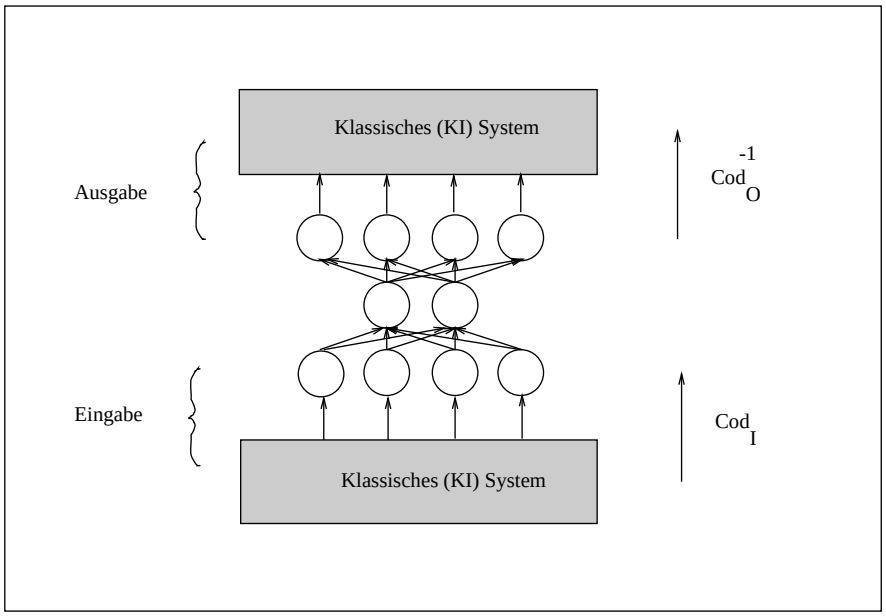

Abbildung 2.5: Eine Möglichkeit der Informationsverarbeitung eines Neuronalen Netzes innerhalb eines klassischen KI Systems.

2.3.3.2 Muster

Im vorherigen Abschnitt wurde gezeigt, wie ein Neuronales Netz in eine Umgebung mittels Kodierung und Dekodierung eingebettet werden kann. Ferner wurde bisher erklärt, daß ein Neuronales Netz Informationen erhält, verarbeitet und das Ergebnis an der Ausgabeschicht zur Verfügung stellt. Dabei weicht anfangs in der Regel das vom Netz errechnete Ergebnis (*Istwert*) vom erwarteten Wert (*Sollwert, target-pattern* t_p) ab. Durch Anwenden eines geeigneten *Lernverfahrens* kann diese Abweichung durch Training verringert werden. Für das Training muß eine Auswahl aus allen möglichen *Mustern* getroffen werden. Ein einzelnes Muster ist ein Paar, bestehend aus einem *Ein-* und einem *Ausgabemuster*[7]. Ein Eingabemuster ist identisch mit dem an der Eingabeschicht anzulegenden Zahlenvektor, wie er im letzten Abschnitt eingeführt wurde. Das heißt, ein Eingabemuster ist die geordnete Menge aller Aktivationen der Units innerhalb der Eingabeschicht. Entsprechend ist das Ausgabemuster die zugehörige Menge der Aktivationen in der Ausgabeschicht. Dabei muß sorgfältig zwischen berechnetem Muster (Istwert) und erwartetem Muster (Sollwert) unterschieden werden.

Alle in der Trainingsphase zur Anwendung kommenden Muster werden häufig als *Lernmuster* bezeichnet. Diese Lernmuster werden auch in der *Ein-/Ausgaberelation* $\mathcal{R}$ zusammengefaßt. Diese besteht aus allen zu lernenden Paaren von Eingabemustern und

[7] Vielfach wird auch ein einzelnes Ein- oder Ausgabemuster kurz als „Muster“ bezeichnet, wenn durch den Zusammenhang keine Verwechslungen oder Unklarheiten entstehen können.

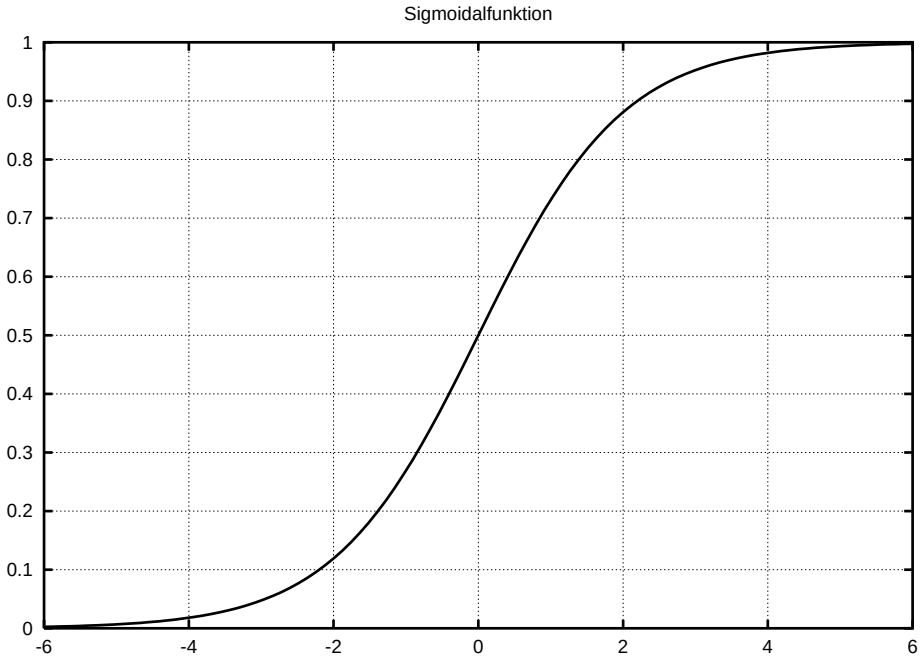

Abbildung 2.6: Die Sigmoidalfunktion $f(x) = \frac{1}{1+e^{-x}}$.

zugeordneten Ausgabemustern.

Eine weitere wichtige Eigenschaft Neuronaler Netze ist ihre Fähigkeit zu *generalisieren*. Unter Generalisieren ist zu verstehen, daß das Netz für ein in der Trainingsphase nicht gesehenes Muster das korrekte Ausgabemuster berechnet. Die zur Bestimmung der Generalisierungsfähigkeit verwendeten Muster heißen *Testmuster*. Im Verlauf der Trainigsphase ist manchmal zu beobachten, daß anfangs die Generalisierungsfähigkeit zunimmt sich dann jedoch wieder verschlechtert, obwohl die Lernmuster immer besser gelernt werden. Diese Abnahme der Generalisierungsfähigkeit wird *overlearning* genannt.

2.4 Einige Eigenschaften des Modells

2.4.1 Verschiedene Aktivierungsfunktionen

In Abschnitt 2.3.1 wurde die sigmoidale Funktion $f(x) = \frac{1}{1+e^{-x}}$, auch Sigmoidalfunktion genannt, als Aktivierungsfunktion eingeführt, die in Abbildung 2.6 graphisch dargestellt ist. Die wichtigsten Eigenschaften dieser Sigmoidalfunktion sind: streng monoton

steigend, beschränkt, semilinear im Bereich um $x = 0$, und die Ableitung der Funktion läßt sich durch $f'(x) = f(x) \cdot (1 - f(x))$ ausdrücken.

Vielfach werden auch andere sigmoidale Funktionen verwendet. Bekannteste Beispiele dafür sind der $atanh$ (Arcustangens hyperbolicus) und die symmetrische Funktion $f(x) = \frac{1}{1+e^{-x}} - 0.5$. Diese beiden Funktionen liegen im Intervall $(-1, 1)$ bzw. $(-0.5, 0.5)$. Die Verwendung dieser beiden Funktionen hat folgenden Grund:

> Bei Verwendung von *Backpropagation* (siehe Abschnitt 3.4.1) als Lernverfahren, treten einzelne Terme auf, in denen die Aktivation enthalten ist. Eine Aktivation mit Betrag null hat in diesen Fällen zur Folge, daß bei diesem Lernschritt kein Fortschritt erzielt wird. Bei Verwendung von symmetrischen Aktivierungsfunktionen hat man in derartigen Fällen immer eine Aktivation mit Betrag ungleich null. In manchen Fällen führt dies zu einer Beschleunigung des Lernens.

Da die Verwendung der beiden letztgenannten Aktivierungsfunktionen nur bei bestimmten Aufgabestellungen zu einer Beschleunigung der Lernphase führt und da diese beiden Funktionen in der Literatur nicht so weit verbreitet sind wie die zuerst eingeführte, wird innerhalb dieser Arbeit nur die eingangs angegebene Sigmoidalfunktion $f(x) = \frac{1}{1+e^{-x}}$ verwendet.

2.4.2 Verschiedene Kodierungen

In Abschnitt 2.3.3.1 heißt es lapidar, *daß für die Art der Kodierung* Cod_I *und Dekodierung* Cod_O^{-1} *der Systementwickler zu sorgen habe*. Das Finden einer geeigneten Kodierung ist besonders wichtig, aber gleichzeitig auch äußerst schwierig. Diese Schwierigkeit wird nicht nur durch praktische Versuche, sondern auch durch die Evolution bestätigt.

Aufgrund der Wichtigkeit dieses Themas werden in diesem Abschintt einige der gebräuchlichsten Kodierungen kurz vorgestellt.

2.4.2.1 Direkte 1-zu-1 Kodierung

Bei dieser Kodierungsform wird der zu kodierende (Zahlen-) Wert direkt als Aktivationswert übernommen. Dabei könnte es sich beispielsweise um einen Winkel, eine Temperatur, eine Geschwindigkeit oder ein Zeichen des ASCII-Codes handeln. Ein Winkel

von 13 Grad würde zu einer Aktivation mit Betrag 13 bei der entsprechenden Unit der Eingabeschicht führen.

Eine Normierung dieser Eingabewerte a_j auf einen Bereich a_j' der innerhalb des Intervalls der Aktivierungsfunktion liegt, ist nicht notwendig, da sich dann auch die von dieser Unit ausgehenden Gewichtungen w_{ij} gemäß $w_{ij} \cdot a_j = w_{ij}' \cdot a_j'$ ändern würden.

Diese Kodierungsform ist äußerst kompakt, da für jede zu kodierende Größe nur eine Unit benötigt wird. Jedoch hat dies unter Umständen einen erhöhten Lernaufwand zur Folge.

2.4.2.2 Verteilte Binärkodierung

Die Verwendung einer Binärkodierung ist eine weitere Möglichkeit, einen Wert darzustellen. Bei dieser *verteilten* Kodierungsform werden bei w Werten für gewöhnlich $\log_2 w$ Units benötigt. Beispielsweise wird bei Verwendung von fünf Units in der Eingabeschicht der Wert neun durch den Vektor $(0, 1, 0, 0, 1)^T$ dargestellt.

Ein Problem bei Verwendung einer Binärkodierung ist sehr häufig der nicht äquidistante Hammingabstand[8] zweier benachbarter Werte. Beispielsweise haben die beiden Werte sieben und acht einen Hammingabstand von vier; die Werte acht und neun hingegen einen Hammingabstand von eins. Aus diesem Grund wird häufig ein Graycode verwendet, der sicherstellt, daß zwei benachbarte Werte immer einen Hammingabstand von genau eins besitzen.

2.4.2.3 Verteilte 1-aus-n-Kodierung

Bei der *lokalen 1-aus-n-Kodierung* werden ebensoviele Units wie darzustellende Werte benötigt. Ein ganzzahliger Wert w wird derart kodiert, daß mit Ausnahme der Unit w, die eine Aktivation mit Betrag eins hat, alle anderen Units eine Aktivation mit Betrag null besitzen. Bei dieser Kodierung ist der Hammingabstand zweier beliebiger verschiedener Zahlen immer zwei.

[8] Unter dem Hammingabstand zweier *Bitstrings* wird die Anzahl derjenigen Stellen verstanden, an denen sich beide Bitstrings in ihrem *Bit* unterscheiden.

2.4.2.4 Thermometerkodierung

Die Thermometerkodierung ähnelt sehr stark der lokalen 1-aus-n-Kodierung. In Gegensatz zur 1-aus-n-Kodierung haben alle Units i, mit $i \leq w$ eine Aktivation mit Betrag eins. Zwei benachbarte Werte haben ebenfalls einen Hammingabstand mit Betrag zwei.

2.4.2.5 Skalarkodierung

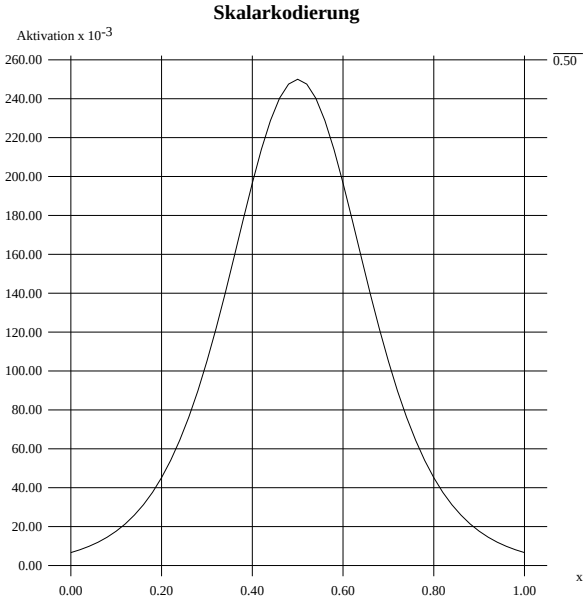

Abbildung 2.7: Ein Wert mit Betrag 0.5 durch die Funktion $f^{'}(10\,(x-0.5))$ skalar kodiert. Mit $f^{'}(x) = f(x) \cdot (1 - f(x))$, $f(x) = \frac{1}{1+e^{-x}}$ und $t = 10$ als Transformationsfaktor.

Hancock [1989], Saund [1989] und Ossen [1990] benutzten eine unimodale Funktion $g(x)$ zur Kodierung von Werten. In Abbildung 2.7 ist beispielhaft ein Wert mit Betrag 0.5 auf diese Weise skalar kodiert worden. Bei dieser Kodierungsform werden alle Units gleichmäßig über das zu kodierende Intervall verteilt (Abszisse). Auf der Ordinate sind dann die Aktivationen der jeweiligen Units abzulesen. Als Kodierungsfunktion $g(x)$ kann beispielsweise die erste Ableitung der Sigmoidalfunktion $f^{'}(x) = f(x) \cdot (1 - f(x))$ mit $f(x) = \frac{1}{1+e^{-x}}$ verwendet werden. Ein Wert mit Betrag w wird dann mittels $g(x - w)$ kodiert. Die unimodale Funktion hat an der Stelle des zu kodierenden Wertes ihr Maximum. Um bei der Skalarkodierung ein genügend scharfes Fenster zu erhalten, ist es sinnvoll, die Argumente durch einen Transformationsfaktor t in $g(t\,(x-w))$ zu transformieren. Versuche von Ossen [1990] zeigten, daß es sich als sehr günstig erwiesen hat, die unimodale Funktion an ihren Rändern zyklisch fortzusetzen. Ein Vorteil dieser Kodierungsform ist die Flexibilität, die durch die variable Zahl von Units

für jede zu kodierende Größe gegeben ist. Weniger Units führen zu einer kompakteren Kodierung und kleineren Netzen; mehr Units hingegen verringern die Lernzeit und stellen unter Umständen eine höhere Leistungsfähigkeit zur Verfügung. Dem Hammingabstand vergleichbar, läßt sich der Abstand zweier skalar kodierter Werte durch das Integral der Differenzquadrate beider Skalarfunktionen bestimmen.

2.4.2.6 Zusammenfassung

Zusammenfassend läßt sich sagen, daß verschiedene Kodierungen auch verschieden kompakt sind. Dabei werden für die direkte nur eine einzige und für die verteilte 1-aus-n-Kodierung sehr viele Units benötigt. Obwohl die kompakte Kodierung nur wenige Units benötigt, braucht ein Lernverfahren hierbei wesentlich mehr Zeit als dies bei verteilter Kodierung der Fall ist. Bei Versuchen von Linden [1989] mit englischsprachigen Wörtern, die aus vier Buchstaben bestanden, zeigte sich, daß die verteilte 1-aus-n-Kodierung der verteilten Binärkodierung deutlich überlegen ist. Die Skalarkodierung eignet sich dagegen wesentlich besser zur Kodierung reellwertiger Zahlen als die direkte Kodierung.

Kapitel 3

Lernen in Neuronalen Netzen

Aus der Biologie ist bekannt, daß viele Lebewesen lernfähig sind. Lernen ist für den Neurobiologen unter anderem an Veränderungen der Synapsen erkennbar. Nach [Valentin Braitenberg and Almut Schüz, 1989] läßt sich beobachten, daß sich durch den Lernvorgang häufig benutzte Synapsen räumlich ausdehnen und andere nicht benötigte Synapsen soweit degenerieren, bis sie nicht mehr an der Signalverarbeitung teilnehmen können. Durch die räumliche Vergrößerung der Synapsen wird der Vesikel/Rezeptor Vorrat erhöht und die Signale können mit verstärkter Wirkung weitergeleitet werden.

In der Informatik werden nur sehr vereinfachte Modelle der Natur verwendet. Dabei werden nicht alle elektrochemischen Vorgänge simuliert, sondern nur einige für die Anwendung wesentlichen Funktionsprinzipien nachgeahmt. Den zur Zeit untersuchten Lernverfahren ist gemeinsam, daß sie die beobachteten Veränderungen an den Synapsen auf das Einstellen von Gewichtungen übertragen. Da jedoch keine neurophysiologischen Mechanismen (sofern diese genau erforscht und bekannt sind) simuliert werden, wird das Einstellen der Gewichtungen mit mathematischen Methoden bewerkstelligt. Im weiteren werden einige dieser (mathematischen) Lernverfahren vorgestellt.

3.1 Übersicht über Lernverfahren

Wie schon eingangs erwähnt geschieht das Lernen in Neuronalen Netzen durch Einstellen der Gewichtungen. Durch das Einstellen aller Gewichtungen wird das erworbene Wissen nicht im Netz lokal, etwa durch jeweils eine einzige Unit oder gar Gewichtung, sondern in Form einer *internen verteilten Repräsentation* (distributed representation) ge-

speichert. Das Ergebnis des Lernprozesses ist demnach eine Anzahl von Gewichtungen und damit verbunden ein zu jedem Ein-/Ausgabemuster typisches Aktivationsmuster (*pattern of activity*) der Units in der Vermittlungsschicht. Dieses Aktivationsmuster wird so gedeutet, daß in der Vermittlungsschicht einzelne *features* bzw. *micro features* erkannt werden. Durch die verteilte Repräsentation des erworbenen Wissens besitzt das Netz ein hohes Maß an Ausfallsicherheit bzw. Fehlertoleranz. Der Ausfall einer einzelnen Unit oder Gewichtung führt nicht notwendigerweise zum völligen Verlust des erworbenen Wissen. Unter Umständen können die Daten rekonstruiert werden (über diese *graceful degradation* siehe auch [McClelland *et al.*, 1986]).

Allen Lernverfahren für Neuronale Netze ist gemeinsam, daß das Netz durch Präsentation von Daten lernt. Es genügen also Fakten. Regeln müssen nicht explizit angegeben werden, sondern sind implizites Ergebnis der Lernphase. Ein weiterer Punkt, der auch als Vorteil gesehen werden kann, ist, daß dem Netz nicht alle gültigen Daten präsentiert werden müssen; es genügt die Präsentation von Beispielen. Sofern die Beispiele typische Vertreter einer ganzen Klasse von Datensätzen sind, kann das Netz ein noch nie gezeigtes, aber ähnliches Muster unter Umständen richtig erkennen und verarbeiten. Diese Eigenschaft wird Generalisierungsfähigkeit genannt.

Es sind mehrere Lernverfahren bekannt. So gibt es zum Beispiel für die Boltzmannmaschine das *simulated anealing*, für das Perceptron die *perceptron convergence procedure* und für Neuronale Netze, wie sie in Abschnitt 2.3 beschrieben wurden, eine Reihe anderer Lernverfahren, die teilweise weiter unten noch genauer erläutert werden.

Die Lernverfahren für Neuronale Netze lassen sich unterscheiden in *supervised* und *unsupervised learning*. Der wesentliche Unterschied zwischen beiden Lernverfahren ist das Wirken einer bewertenden Instanz beim *supervised learning*. Backpropagation, Evolutionsstrategie und die *perceptron convergence procedure* sind Vertreter dieser Klasse. Eine recht ausführliche Erklärung der ersten beiden Lernverfahren ist in den Abschnitten 3.4.1 und 3.4.2 zu finden. Bei allen Lernverfahren dieser Klasse wird ein Gütekriterium (Quatlitätsmaß) definiert. Durch das Lernverfahren wird immer wieder überprüft, wie gut sich das Netz dem geforderten Verhalten angepaßt hat, und dementsprechend wird das Qualtitätsmaß bestimmt. Anschließend wird das Netz mittels des Lernverfahrens derart verändert, daß sich sein Verhalten im geforderten Sinne verbessert.

Häufig werden noch diejenigen Aufgaben gesondert klassifiziert, bei denen eine *auto-assoziative* Zuordnung der Ein-/Ausgabemuster gelernt werden soll. *Auto-assoziative* Zuordnung (auch *auto-assoziation*, *identity mapping* oder Selbstabbildung genannt) bedeutet, daß Ein- und Ausgabe bei jeder Abbildung der Ein-/Ausgaberelation $\mathcal{R}$ identisch sind. In einem derartigen Fall spricht man auch von *unsupervised backpropagation*, obwohl es sich um *supervised learning* handelt.

Da Backpropagation und Evolutionsstrategie im weiteren noch näher erläutert werden,

wird hier noch kurz die *perceptron convergence procedure* vorangestellt. Die *perceptron convergence procedure* ist auf zweischichtige Perceptrons beschränkt. Für diese Lernregel gilt nach [Minsky und Papert, 1969] der Satz:

> Die *perceptron convergence procedure* findet für jede beliebige Kombination von Ein-/Ausgabemustern eine Gewichtungsmatrix derart, daß die Eingaben die zugehörigen Ausgaben erzeugen – sofern eine solche Matrix existiert.

Die Vorraussetzung für die Existenz einer solchen Matrix ist die lineare Separierbarkeit der Ein-/Ausgaberelation (siehe auch Abschnitt 2.2.1 Ossen [1990, Seite 23] oder [Linden, 1989, Seite 20-23]). Diese lineare Separabilität ist beispielsweise schon für die Exklusiv-Oder Funktion nicht gegeben. Eine Erweiterung der Leistungsfähigkeit ist nur durch die Verwendung mehrerer Schichten oder von Nichtlinearitäten möglich. Eine Lernregel für *multi-layer* Perceptrons ist allerdings nicht bekannt.

Im Gegensatz dazu ist beim *unsupervised learning* das Wirken einer bewertenden Instanz nicht notwendig. Vielmehr verändert sich das Netz bei jedem angelegten Muster selbsttätig. Die wohl bekannteste Regel dieser Art ist die Lernregel von Hebb (siehe beispielsweise [Hinton, 1989, Seite 215]). Nach der Regel von Hebb führt die gleichzeitige Aktivation zweier Units zur Stärkung der sie verbindenden Gewichtungen. Diese Regel wurde später von Neurobiologen an natürlichen Nervenfasern bestätigt. Neuere Untersuchungen von [van Hemmen *et al.*, 1990] zeigen, daß die Hinzunahme von *Verlernen* die Leistungsfähigkeit Neuronaler Netze drastisch verbessern kann. Dieses Verlernen hat die REM-Phasen des menschlichen Schlafes zum Vorbild. Nach van Hemmen wurde auch dieses Verlernen von den Neurobiologen im nachhinein bestätigt. Desweiteren kommt *unsupervised learning* sehr stark bei Assoziativ-Speichern zur Anwendung. Auf diesem Gebiet besonders bekannt geworden sind die Arbeiten von Kohonen (siehe zum Beispiel [Kohonen, 1989]).

3.2 Supervised learning

In Kapitel 2 wurde ein detailliertes, mathematisches Modell von Neuronalen Netzen entwickelt, wie sie in der Informatik verwendet werden. Nachdem im letzten Abschnitt verschiedene Klassen von Lernverfahren für Neuronale Netze vorgestellt wurden, soll in diesem Abschnitt supervised learning genauer entwickelt werden, da die Verbesserung dieser Methode Gegenstand dieser Arbeit ist. Es werden dabei wichtige Begriffe wie *Fehlerfunktion*, *Fehlerwert*, *Maximalfehler*, *on-line* und *batch* eingeführt.

Nach dem bisher Dargestellten ist ein Neuronales Netz in der Lage, für jedes präsentierte Muster eine Ausgabe zu berechnen. Nach Abschnitt 2.3.3.2 ist eine Ein-/Ausgaberelation $\mathcal{R}$ vorgegeben, die von dem Neuronalen Netz nach Möglichkeit eingehalten werden soll. Im allgemeinen besteht ein Unterschied zwischen den Ausgaben (Sollwert t_{ip} (*engl. target pattern*)), die durch die Relation $\mathcal{R}$ vorgegeben sind und denjenigen, die durch das Netz (als Istwert o_{ip}) berechenet werden. Durch die wiederholte Anwendung einer Lernregel soll dieser Unterschied minimiert werden. Um ein quantitatives Bewertungsmaß, wie es bei supervised learning notwendig ist, zu erhalten, wird der Differenzbetrag zwischen Soll- und Istwert an einer Unit i bei angelegtem Muster p als (individueller) Fehler $e_{ip} = |t_{ip} - o_{ip}|$ gebildet. Die (Partial-) Fehlersumme E_p ist dann die halbe Summe aller Fehlerquadrate e_{ip} bei angelegtem Muster p, und der (Gesamt-) Fehler E ist die Summe der einzelnen Fehler E_p. Die mathematische Formulierung der Fehlerfunktion lautet:

$$E(\vec{w}) = \sum_p E_p(\vec{w}) = \frac{1}{2} \sum_p \sum_i (t_{pi} - o_{pi})^2 \quad . \tag{3.1}$$

In Abschnitt 3.4.1 wird noch gezeigt, daß der Faktor $\frac{1}{2}$ in dieser Gleichung nur technische Bedeutung hat. Aus zwei Gründen erfolgt die Summenbildung nicht über die eigentlichen Differenzen, sondern über deren Quadrate. Erstens wird dadurch auf indirekte Weise der Betrag gebildet und zweitens werden größere Abweichungen gegenüber kleineren überproportional stark bewertet. Dadurch werden in der Regel vordringlich die größten Abweichungen beseitigt.

Mit Hilfe der Fehlerfunktion ist es nun sehr einfach, eine Lernregel für die *on-line* Variante zu formulieren:

Für alle Muster der Ein-/Ausgaberelation:
Lege das nächste Eingabemuster an der Eingabeschicht an
Warte bis das Netz die Ausgabe berechnet hat
Bestimme aus der Differenz von Soll- und Istwert alle Änderungen
Verändere alle Gewichtungen entsprechend der Änderungen

Nach dieser Lernregel werden dem Netz nacheinander alle Muster präsentiert. Aus den Abweichungen von Soll- und Istwert werden Änderungen für alle Gewichtungen ermittelt. Ein wichtiger Punkt ist, daß alle Gewichtungen und damit das Ein-/Ausgabeverhalten des Netzes nach jedem Muster verändert wird.

Batch und *on-line* Verfahren sind verwandt. Der einzige Unterschied ist, daß beim batch Verfahren alle Muster der Ein-/Ausgaberelation dem gleichen Netz präsentiert werden.

Das geht nur, wenn die jeweils ermittelten Änderungen zwischengespeichert werden und das Netz erst am Ende mit der vektoriellen Summe verändert wird.

Die Präsentation aller Muster, egal ob on-line oder batch Variante, wird Epoche oder (Lern-) Zyklus genannt. In der Regel werden Lernzyklen nicht *ad infinitum* durchgeführt. Vielmehr erfolgt die Beendigung durch Erreichen eines vorher festgelegten *Abbruchkriteriums*. Als Abbruchkriterium wird normalerweise eine der beiden folgenden Formulierungen gewählt. Entweder wird eine zu erreichende Fehlerquadratsumme (kurz Gesamtfehler oder Fehlerwert) vorgegeben, oder man legt einen *Maximalfehler* τ fest. Der Maximalfehler MBE (maximum bit error) ist der Betrag der größten Abweichung $|t_{pi} - o_{pi}|$ gemäß $\forall_{i,p}\, |t_{pi} - o_{pi}| \leq \mathrm{MBE}$.

Als Abbruchkriterium wurden ein zu unterschreitender Gesamtfehler und der Maximalfehler vorgestellt. Die Verwendung eines zu unterschreitenden Maximalfehlers ist der Kritik ausgesetzt, daß statistische Ausreißer nicht problemangemessen berücksichtigt werden. Wenn bei einer Aufgabenstellung ein Gesamtfehler von zum Beispiel 0.8 gewählt wird, der einem durchschnittlichen Wert von 0.05 je Unit je Ausgabemuster entspricht, dann kann es vorkommen, daß mit Ausnahme eines einzelnen Fehlerwertes von 0.6 alle anderen sehr klein sind. Es ist dann zwar der Gesamtfehler unterschritten, doch ist ein Muster, da der Fehler an einer Unit größer als 0.5 ist, nicht richtig gelernt worden.

3.3 Skalarfeld – Qualitätsgebirge

In diesem Abschnitt wird die Fehlerfunktion E nicht nur als Bewertungsmaß für die Güte eines Neuronalen Netzes, sondern als eine mathematische Funktion betrachtet. Es werden hier nur die wesentlichen Aspekte in aller Kürze vorgestellt. Eine ausführliche, vertiefende Darstellung findet man in [Bourne und Kendall, 1973].

Im mathematischen Sinne ist die Fehlerfunktion E

$$E = \frac{1}{2} \sum_p \sum_i (t_{pi} - o_{pi})^2 \tag{3.2}$$

eine Skalarfunktion mit n unabhängigen Variablen.

$$E : (x_1, x_2, \cdots, x_n) \rightarrow \frac{1}{2} \sum_p \sum_i (t_{pi} - o_{pi})^2 \tag{3.3}$$

Diese unabhängigen Variablen sind im Fall der Neuronalen Netze die einzelnen Gewichtungen. Es muß aber nicht immer sein, daß *alle* Gewichtungen als Argumente der

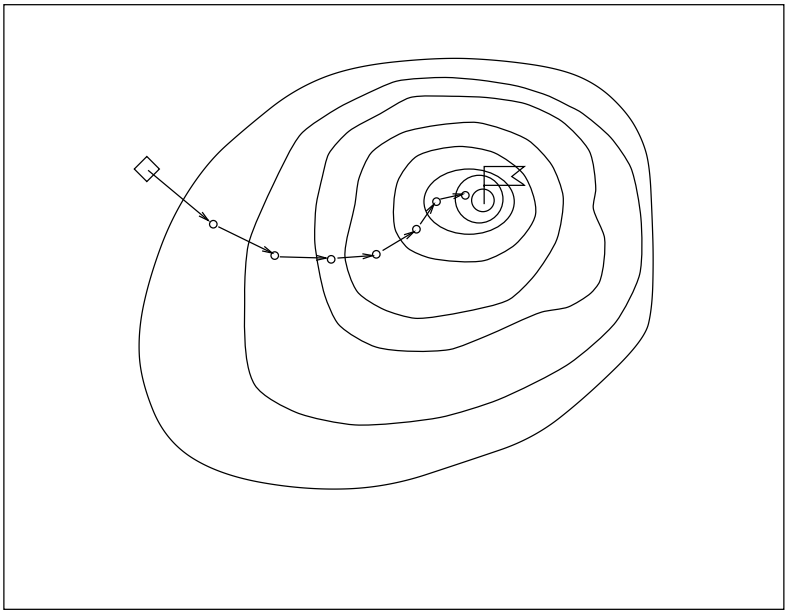

Abbildung 3.1: Das Qualitätsgebirge mit konvergierender Punktfolge.

Fehlerfunktion auftauchen. Bei Benutzung von *constrained networks* (siehe z.B. [le Cun, 1989a]) werden mehrere Gewichtungen miteinander gekoppelt und durch nur einen einzigen Parameter bestimmt. In diesem Fall sind die unabhängigen Variablen nicht mehr alle Gewichtungen, sondern die neu eingeführten Parameter. Doch das ändert nichts an dem grundlegenden Ansatz, da dann die Fehlerfunktion von m neuen Variablen abhängt. Im Weiteren wird die Fehlerfunktion als Funktion von n Variablen betrachtet; gleichgültig ob es konkrete Gewichtungen oder neue Parameter von speziell durchzuführenden Transformationen sind.

Ab hier wird davon ausgegangen, daß die Variablen als voneinander unabhängig angesetzt sind. Eine Skalarfunktion mit n Variablen ist im n-dimensionalen Raum als Abbildung von $\mathcal{R}^n \rightarrow \mathcal{R}$ erklärt. Mitunter ist es nützlich die Skalarfunktion im $n+1$-dimensionalen Raum graphisch darzustellen. Diese graphische Darstellung (siehe Abbildung 3.1) wird in Anlehnung an die Terminologie von Rechenberg [1973] *Qualitätsgebirge* genannt. Die eingezeichneten *Höhenlinien* repräsentieren Orte gleichen Skalarwertes.

Im Bereich der Neuronalen Netze gilt es, das globale Minimum der Fehlerfunktion zu finden. Innerhalb des Skalarfeldes ist der Gradient als

$$\text{grad}E = \nabla E = \begin{pmatrix} \frac{\partial E}{\partial x_1} \\ \vdots \\ \frac{\partial E}{\partial x_n} \end{pmatrix} \tag{3.4}$$

definiert. Die Komponenten sind die partiellen Ableitungen nach den korrespondierenden Variablen. Durch den Gradienten ist in jedem Punkt die Richtung der stärksten Änderung gegeben. Diese Richtungsinformation wird in den nach der Gradientenmethode arbeitenden Optimierungsverfahren genutzt, um durch kleine Iterationsschritte das Optimum[1] zu finden. Je nachdem ob Maximum oder Minimum gesucht wird, sieht ein Iterationsschritt wie folgt aus:

$$\begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix}^{t+1} = \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix}^{t} \pm \eta \cdot \nabla E \tag{3.5}$$

in Vektorschreibweise

$$\vec{x}^{t+1} = \vec{x}^{t} \pm \eta \cdot \nabla E \quad . \tag{3.6}$$

In Gleichung (3.5) und (3.6) ist η die Schrittweite und $\vec{x}$ die vektorielle Schreibweise aller Variablen x_i. Die hochgestellten Indizes t und $t+1$ bezeichnen den Zustand vor bzw. nach dem Iterationsschritt. In Gleichung (3.5) bzw. (3.6) wird ausgedrückt, daß jede Variable um ein Vielfaches η der entsprechenden Gradientenkomponente in die gewünschte Richtung verändert wird. Das Iterationsverfahren wird solange durchgeführt, bis ein gegebenes *Abbruchkriterium* erfüllt ist. Die Anzahl der durchgeführten Iterationsschritte wird auch *Zyklenzahl* genannt. Ein notwendiges Kriterium für Konvergenz ist eine genügend kleine Schrittweite η. Genügend klein bedeutet, daß das Produkt aus Schrittweite η und Länge des Gradienten klein im Verhältnis zur Entfernung zum Optimum sein muß. Andernfalls kann es vorkommen, daß die Gradientenmethode das Optimum verfehlt.

Weiter oben wurde die Suche des globalen Minimums als Ziel eines Optimierungsverfahrens genannt. Nur wenn ein Optimierungsverfahren **immer** das globale Optimum findet, verdient es die Bezeichnung globales Optimierungsverfahren. Zur Zeit ist jedoch kein universelles Optimierungsverfahren bekannt, mit dessen Hilfe **immer** das globale Optimum gefunden werden kann. Insbesondere finden die nach der Gradientenmethode arbeitenden Optimierungsverfahren in der Regel nur das nächstgelegene lokale Optimum. Es hängt vom Startpunkt ab, ob das gefundene lokale Optimum mit dem globalen identisch ist. Nur unter ganz bestimmten Voraussetzungen ist die Evolutionsstrategie [Rechenberg, 1973] in der Lage, aus lokalen Nebenoptima herauszukommen.

[1] Mit Optimum ist die Verallgemeinerung von Minimum und Maximum gemeint. Es hängt von der konkreten Anwendung eines Optimierungsverfahrens ab, welches der beiden gesucht wird und somit als Optimum anzusehen ist.

3.4 Das Gradientenverfahren

In diesem Abschnitt wird ausgehend von der Iterationsvorschrift (Gleichung (3.6)) des letzten Abschnitts das Gradientenverfahren genauer erläutert. Dabei wird insbesondere auf die Eigenheiten Neuronaler Netze eingegangen. Im Anschluß daran wird die Evolutionsstrategie von Rechenberg als Beispiel eines *reinforcement* Verfahrens vorgestellt.

3.4.1 Backpropagation als Gradientenverfahren

Nachdem die Lernregel formuliert, das Lernverfahren (Iterationsverfahren) nach der Gradientenmethode vorgestellt und die Bestandteile eines Neuronalen Netzes durch Formeln (Gleichung (2.6) bis (2.8)) beschrieben wurde, fehlt jetzt nur noch die Gradientenbildung der Fehlerfunktion.

Backpropagation ist nichts anderes, als die Anwendung der Gradientenmethode auf ein Neuronales Netz. Das Besondere an Backpropagation ist die Art und Weise der Gradientenbildung in Neuronalen Netzen. Im Folgenden wird die Gradientenbildung bei Backpropagation anhand eines einfachen feed-forward Netzes so ausführlich hergeleitet, daß eine Erweiterung auf umfangreichere Topologien (insbesondere der Verwendung von *short cuts* [Lang und Witbrock, 1989]) und eine algorithmische Umsetzung nicht schwer fallen sollte. Ein weiterer Grund für diese Ausführlichkeit ist, daß einige Gleichungen dieses Abschnitts zur Aufwandsabschätzung und Beurteilung von Verbesserungen der Gradientenmethode benötigt werden. Weitere umfangreiche Darstellungen befinden sich zum Beispiel in [Rumelhart *et al.*, 1986a] und [Kindermann und Linden, 1988].

Nach den Ergebnissen aus Abschnitt 3.2 und 3.3 läßt sich ein Iterationsschritt durch folgende zwei Gleichungen beschreiben:

$$E(\vec{w}) = \sum_p E_p(\vec{w}) = \frac{1}{2} \sum_p \sum_i (t_{pi} - o_{pi})^2 \tag{3.7}$$

$$\vec{w}_{k+1} = \vec{w}_k - \eta \cdot \nabla E(\vec{w}_k) \quad . \tag{3.8}$$

In Abschnitt 3.2 wurde bereits erläutert, daß sich on-line und batch Verfahren im Zeitpunkt unterscheiden, in dem die ermittelten Änderungen auf das Netz übertragen werden. Da in Gleichung (3.7) alle Summanden linear unabhängig sind, spielt die Summa-

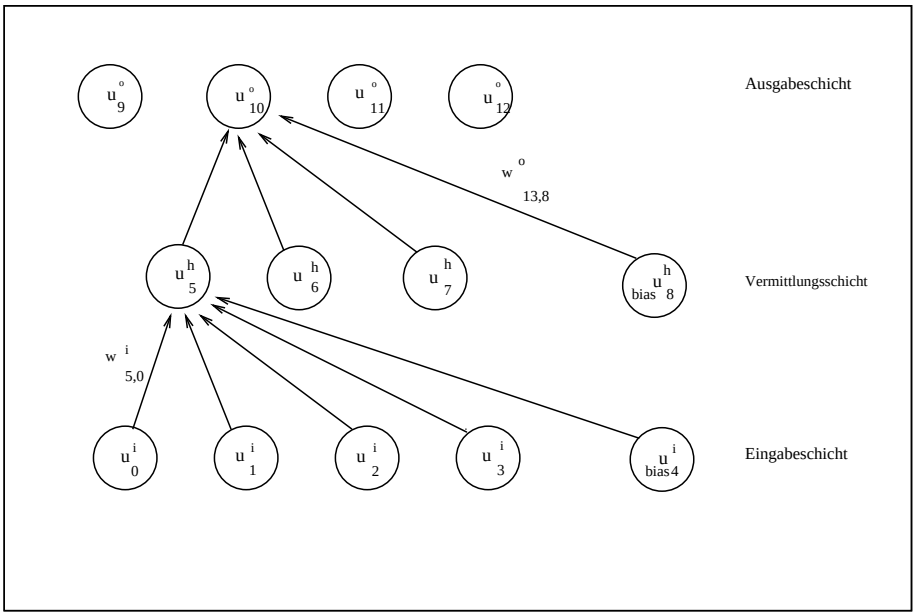

Abbildung 3.2: Ein einfaches *feed-forward network* mit modifizierter Notation.

tionsreihenfolge keine Rolle und beide Verfahren können auf dem selben mathematischen Ansatz aufgebaut werden, denn es gilt:

$$\nabla E(\vec{w}_k) = \sum_p \nabla E_p(\vec{w}_k) \quad . \tag{3.9}$$

Um im Folgenden die Gleichungen einfacher herzuleiten und um sie geschlossener darstellen zu können, wird von dem in Abbildung 3.2 abgebildeten, einfachen feed-forward Netz ausgegangen. In Abbildung 3.2 sind der Übersicht halber etliche Gewichtungen weggelassen. Ferner wird abweichend zu Abschnitt 2.3.1 durch einen hochgestellten Index gekennzeichnet, in welcher Schicht sich eine Unit oder Gewichtung befindet. Alle Gleichungen werden in Komponentenschreibweise dargestellt und die Indizes k für die Kennzeichnung eines bestimmten Iterationsschrittes sowie p für das präsentierte Muster werden unterdrückt. Im einzelnen wird folgende modifizierte Notation verwendet:

u_i ist die Aktivation einer beliebigen Unit i,

u_i^i ist die Aktivation der i-ten Unit der Eingabeschicht,

u_i^h ist die Aktivation der i-ten Unit der Vermittlungsschicht,

u_i^o ist die Aktivation der i-ten Unit der Ausgabeschicht,

w_{ij}^h ist die Gewichtung von der Unit j der Eingabeschicht zu der Unit i der Vermittlungsschicht.

w_{ij}^o ist die Gewichtung von der Unit j der Vermittlungsschicht zu der Unit i der Ausgabeschicht.

Es erscheint nützlich zuvor noch einmal die wichtigsten Gleichungen aus Abschnitt 2.3.1 zusammenzufassen und deren Ableitungen anzugeben:

$$\frac{\partial \,\mathrm{netinput}_{u_i}}{\partial w_{ij}} = \frac{\partial}{\partial w_{ij}} \sum_j w_{ij} u_j = u_j \quad , \tag{3.10}$$

$$\frac{\partial \,\mathrm{netinput}_{u_i}}{\partial u_j} = \frac{\partial}{\partial u_j} \sum_j w_{ij} u_j = w_{ij} \quad , \tag{3.11}$$

$$\frac{\partial u_i}{\partial \,\mathrm{netinput}_{u_i}} = \frac{\partial}{\partial \mathrm{netinput}_{u_i}} \frac{1}{1 + e^{-\mathrm{netinput}_{u_i}}} = u_i \,(1 - u_i) \quad . \tag{3.12}$$

Die einzelnen Komponenten des Gradienten sind nach Gleichung (3.4) die partiellen Ableitungen der Fehlerfunktion nach den korrespondierenden Gewichtungen. Zunächst werden nur die w_{ij}^o Komponenten betrachtet. Unter mehrmaligem Anwenden der Kettenregel erhält man:

$$\begin{aligned}
\frac{\partial E_p}{\partial w_{ij}^o} &= \frac{\partial}{\partial w_{ij}^o} \frac{1}{2} \sum_i (t_i - u_i^o)^2 \\
&= \frac{\partial}{\partial u_i^o} \frac{1}{2} \sum_i (t_i - u_i^o)^2 \cdot \frac{\partial u_i^o}{\partial \,\mathrm{netinput}_{u_i^o}} \cdot \frac{\partial \,\mathrm{netinput}_{u_i^o}}{\partial w_{ij}^o} \\
&= -(t_i - u_i^o) \cdot u_i^o (1 - u_i^o) \cdot u_j^h \\
&= -e_i \cdot u_i^o (1 - u_i^o) \cdot u_j^h \quad .
\end{aligned} \tag{3.13}$$

Somit hätten wir für die w_{ij}^o Komponenten eine sehr einfache und leicht interpretierbare Gleichung. Die betrachteten Komponenten des Gradienten ergeben sich als Produkt aus dem Fehler e_{ip}, der Ableitung der Aktivierungsfunktion an der Unit i und der Aktivation der Unit j.

Nun müssen noch die Gewichtungen zwischen Eingabe- und Vermittlungsschicht betrachtet werden. Bedauerlicherweise reduziert sich die Summe in Gleichung (3.13) beim Ableiten nicht auf einen einzigen Summanden. An Abbildung 3.2 kann man erkennen, daß eine ausgewählte Gewichtung w_{ij}^h sehr wohl einen Einfluß auf alle Units der Ausgabeschicht hat, und somit alle Summanden zu berücksichtigen sind.

Die Ableitung aus Gleichung (3.13) kann anhand von Abbildung 3.2 wie folgt veranschaulicht werden: Betrachtet man die Gewichtung w_{ij}^o, so muß man von der Ausgabeschicht über die Unit i und die Gewichtung w_{ij}^o zu der Unit j gehen. Entsprechend muß

auch bei der Gewichtung w_{ij}^h verfahren werden. Man muß von der Ausgabeschicht, in diesem Fall alle Units k, über die Gewichtungen w_{ki}^o, die Unit i und die Gewichtung w_{ij}^h zur Unit j gehen.

$$\begin{aligned}
\frac{\partial E_p}{\partial w_{ij}^h} &= \frac{\partial}{\partial w_{ij}^h}\frac{1}{2}\sum_k (t_k - u_k^o)^2 \\
&= \frac{\partial}{\partial u_k^o}\frac{1}{2}\sum_k (t_k - u_k^o)^2 \cdot \frac{\partial u_k^o}{\partial \,\mathrm{netinput}_{u_k^o}} \frac{\partial \,\mathrm{netinput}_{u_k^o}}{\partial u_i^h} \frac{\partial u_i^h}{\partial \,\mathrm{netinput}_{u_i^h}} \frac{\partial \,\mathrm{netinput}_{u_i^h}}{\partial w_{ij}^h} \\
&= -\sum_k (t_k - u_k^o) \cdot u_k^o(1 - u_k^o) \cdot w_{ki}^o \cdot u_i^h(1 - u_i^h) \cdot u_j^i \quad . \qquad (3.14)
\end{aligned}$$

Die gestellte Aufgabe, den Gradienten zu ermitteln, ist gelöst, doch leider ist Gleichung (3.14) äußerst unbefriedigend, da sie nicht so einfach und klar wie Gleichung (3.13) aufgebaut ist. Es fällt auf, daß in Gleichung (3.14) neben einer Reihe weiterer Faktoren auch eine Summenbildung über alle Units der Ausgabeschicht durchzuführen ist. Desweiteren wird Gleichung (3.14) für jede weitere Vermittlungsschicht oder durch Einführen von *short cuts* wesentlich aufwendiger.

Gleichung (3.13) hat die Struktur „$\mathrm{Fehlerwert} \cdot f'(\mathrm{netinput}) \cdot \mathrm{Aktivation}$“. Im Folgenden wird versucht, auch Gleichung (3.14) diese Struktur zu geben. Durch Klammern erhält man:

$$\frac{\partial E_p}{\partial w_{ij}^h} = -\left(\sum_k (t_k - u_k^o) \cdot u_k^o(1 - u_k^o) \cdot w_{ki}^o\right) u_i^h(1 - u_i^h) \cdot u_j^i \quad . \qquad (3.15)$$

Durch Einführen einer neuen Fehlergröße $e_i^{'} = \sum_k e_k \cdot u_k^o(1 - u_k^o) \cdot w_{ki}^o$ ergibt sich die gleiche Struktur.

$$\frac{\partial E_p}{\partial w_{ij}^h} = -e_i^{'} \cdot u_i^h(1 - u_i^h) \cdot u_j^i \quad . \qquad (3.16)$$

Der neu eingeführte Fehlerterm $e_i^{'}$ ist die Summe der an der Ausgabeschicht entstandenen Fehler, wobei jeder einzelne Fehlerwert mit der Ableitung der Aktivation und der Gewichtung, die zwischen den korrespondierenden Units liegt, bewertet wird.

Das geschickte Bilden der Fehlerterme $e_i^{'}$ ist das eigentliche Kernstück des Backpropagation Algorithmus. Die oben geforderte Verallgemeinerung ist durch folgende (rekursive) Interpretation leicht möglich:

Wenn alle Fehlerwerte an der äußersten Schicht (Ausgabeschicht) festgestellt sind, kann für jede (nach *unten* führende) Gewichtung direkt die Komponente des Gradienten berechnet werden. Gleichzeitig muß das Produkt aus Fehler, Ableitung an der betrachteten Unit und Betrag der betrachteten Gewichtung an die sendende Unit der unteren Schicht als Partialfehlerterm *rückpropagiert* werden. Durch den letzten Schritt wird für jede Unit der unteren Schicht ein fiktiver Fehlerwert berechnet.

Durch das bewertete Rückpropagieren der Fehlerwerte erhält man einen rekursiven Algorithmus mit dem in jeder Schicht in gleicher Art und Weise gearbeitet werden kann. Auch die Berücksichtigung von *short cuts* erfordert keine Spezialbehandlung.

3.4.2 Die Evolutionsstrategie

Die Evolutionsstrategie ist ein universelles Optimierungsverfahren, das durch eine ganz andere Vorgehensweise im statistischen Mittel in Richtung des Gradienten voranschreitet. Dabei wird nicht die Differenzierbarkeit wie bei der Gradientenmethode sondern nur die Existenz der Funktionswerte vorausgesetzt. Sie wurde von Ingo Rechenberg Anfang der siebziger Jahre entwickelt, dessen Grundidee die technische Nutzung der wesentlichen Prinzipien der natürlichen Evolution war. In diesem Abschnitt wird die Evolutionsstrategie kurz vorgestellt. Für tiefer gehende Studien sei [Rechenberg, 1973] empfohlen.

Die wesentlichen Prinzipien der natürlichen Evolution sind *Mutation* und *Selektion*. Im Mutations-Prozeß werden mehrere Nachkommen erzeugt, die sich nur geringfügig von ihren Eltern unterscheiden. Im anschließenden Selektions-Prozeß werden die besten Nachkommen herausgesucht und zu den neuen Eltern der nächsten *Generation* g gemacht.

Die einzige Voraussetzung der Evolutionsstrategie ist nach Rechenberg die Erfüllung des *Strengen Kausalitätsprinzips*. Dies besagt, daß kleine Änderungen nur kleine Wirkungen zur Folge haben dürfen. Diese Forderung ist in der Regel bei Neuronalen Netzen erfüllt. Eine kleine Änderung einer Gewichtung hat normalerweise eine kleine Änderung der Ausgabe zur Folge. Das strenge Kausalitätsprinzip ist beispielsweise [Deker und Thomas, 1983, Seite 63] nicht auf dem Billiard-Tisch erfüllt. Nach neunmaliger Bandenberührung hat die Anwesenheit eines Menschen in der Nähe des Tisches zur Folge, daß keine Aussage mehr über den Lauf des Balles gemacht werden kann.

In der Evolutionsstrategie von Rechenberg werden einzelne *Objekte* (zum Beispiel ein Neuronales Netz) behandelt, die durch ihre einzelnen *Objektvariablen* o_i (zum Beispiel

die einzelnen Gewichtungen) beschrieben werden. Im Mutations-Prozeß werden einzelne Nachkommen erzeugt, deren Objektvariablen leicht von denen der Eltern abweichen. Diese Abweichung erhält man, indem **jede** Objektvariable mit einer gaußverteilten *Zufallszahl* z_i modifiziert wird. Dabei wird für jeden Nachkommen und für jede Objektvariable in jeder Generation eine individuelle Zufallszahl gezogen. Die Streuung der durch die Zufallszahlen bewirkte Modifikationen wird durch die *Schrittweite* η gesteuert. Mit Hilfe dieser Schrittweite geschieht eine dynamische Anpassung in jedem Punkt des Qualitätsgebirges. Die mathematische Formulierung für die Bildung eines Nachkommen mit den Objektvariablen $\vec{o} = (o_1, \cdots, o_n)^T$ lautet wie folgt:

$$\begin{aligned} \vec{o}_1^{\,g+1} &= \vec{o}_0^{\,g} + \eta\vec{z}_1 \\ \vec{o}_2^{\,g+1} &= \vec{o}_0^{\,g} + \eta\vec{z}_2 \\ \vdots &= \vdots \\ \vec{o}_n^{\,g+1} &= \vec{o}_0^{\,g} + \eta\vec{z}_n \quad . \end{aligned} \tag{3.17}$$

Im Selektions-Prozeß wird für jeden Nachkommen die Qualität (zum Beispiel der Fehlerwert) bestimmt. Bei der anschließenden Auswahl gibt es zwei Varianten. Wenn bei μ Eltern und λ Nachkommen die Eltern in die Selektion mit einbezogen werden, spricht man von einer $(\mu + \lambda)$ und im anderen Fall von einer (μ, λ) Strategie.

Für die Anpassung der Schrittweite werden von Rechenberg zwei Verfahren vorgeschlagen. Das erste Verfahren ist die $\frac{1}{5}$ Erfolgsregel, die besagt, daß die Schrittweite den optimalen Wert hat, wenn ein Fünftel der Nachkommen besser als die Eltern sind. Im Falle des Unterschreitens muß die Schrittweite erhöht und im Falle des Überschreitens verringert werden. Beim zweiten und wohl wichtigeren Verfahren ist die Schrittweite Bestandteil des Objektes. Bei jedem Nachkommen wird die Schrittweite durch Modifikation der Schrittweite des Elter ermittelt. Nach der Voraussetzung, daß sich im statistischen Durchschnitt diejenigen Nachkommen durchsetzen, die die am besten angepaßte Schrittweite haben, erfolgt die geforderte dynamische Anpassung. Diese Anpassung wird *Mutative Schrittweiten Regelung MSR* genannt.

3.4.3 Vergleich Gradientenverfahren und Evolutionsstrategie

In diesem Abschnitt wird ein Vergleich der beiden behandelten Lernverfahren gegeben. Dabei werden insbesondere allgemeine Abschätzungen über den Aufwand gemacht und geprüft, inwieweit sich die Evolutionsstrategie als Lernverfahren für Neuronale Netze eignet.

Bei einem Vergleich der beiden Verfahren muß der benötigte Aufwand je Zyklus untersucht werden. Hier sollten die beiden folgenden Fälle unterschieden werden:

- In Aufgabenstellungen, die hinreichend beschrieben sind, bereitet die Bestimmung des Gradienten, vom eventuellen mathematischen Aufwand einmal abgesehen, keine Schwierigkeiten. In Abschnitt 3.4.1 wurde gezeigt, daß der Gradient sehr leicht aus den Resultaten der Propagierung eines Musters errechnet werden kann. In diesen Fällen ist die (spezialisierte) Gradientenmethode wesentlich schneller als die (universelle) Evolutionsstrategie.

- Andererseits gibt es Aufgaben, für die keine hinreichende mathematische Beschreibung bekannt ist. Häufig sind bestenfalls ungefähre Abschätzungen über den Einfluß einzelner Objektvariablen auf das Ergebnis bekannt. In einem derartigen Fall muß bei der Gradientenmethode jede einzelne Komponente des Gradienten experimentell bestimmt werden, um anschließend einen Schritt in diese Gradientenrichtung durchzuführen. Daraus resultiert, daß der Fortschritt φ in Richtung des Gradienten gegeben ist durch:

  $$\varphi \sim \frac{1}{n+1} \quad . \tag{3.18}$$

  Bei der Evolutionsstrategie werden bei jedem Nachkommen alle Objektvariablen einzeln und jede für sich zufällig verändert. Bei $n \gg 1$ liegen alle Nachkommen auf einer $n+1$-dimensionalen Kugel. Dies liegt daran, daß bei sehr vielen Gewichtungen $n \gg 1$ und gaußverteilten Zufallszahlen die Summe aller Zufallszahlen zum Quadrat gleich dem n-fachen der Standardabweichung zum Quadrat ist. Der resultierende Fortschritt ist:

  $$\varphi \sim \frac{1}{\sqrt{n+1}} \quad . \tag{3.19}$$

  Ein Vergleich der beiden Gleichungen (3.18) und (3.19) zeigt, daß mit immer größer werdendem n die Evolutionsstrategie immer besser wird. Ein weiterer Vorteil der Evolutionsstrategie liegt in der dynamischen Anpassung der Schrittweite, die zu einer nochmaligen Beschleunigung der Konvergenz führt.

Die Erläuterung der beiden Fälle zeigt, daß die Evolutionsstrategie als Lernverfahren für Neuronale Netze, bei dem nur die Gewichtungen modifiziert werden, der Gradientenmethode unterlegen ist. Das liegt daran, daß bei Neuronalen Netzen der Gradient direkt bestimmt werden kann und nicht experimentell ermittelt werden muß. Ein weiteres Problem für die Evolutionsstrategie als Lernverfahren für Neuronale Netze ist, daß die Schrittweitenregelung bei bestimmten Qualitätsfunktionen versagt (siehe dazu [Schwefel, 1977]). Bei diesen Qualitätsfunktionen wird die Schrittweite durch die $\frac{1}{5}$ Erfolgsregel

bzw. die Mutative Schrittweitenregelung so stark verkleinert, daß die Änderungen der Gewichtungen durch Rundungsfehler verloren gehen, wodurch die Evolutionsstrategie zum Stillstand kommt. Dieses Verhalten ist zum Beispiel beim 8-2-8 Enkoder zu beobachten.

Ein anderer Sachverhalt ergibt sich bei der Verwendung der Evolutionsstrategie zur Strukturoptimierung. Dabei wird mittels Evolutionsstrategie eine Netztopologie gesucht, die bestimmte Anforderungen erfüllen soll. Interessante Resultate wurden unter anderem von Schiffmann [Schiffmann und Mecklenburg, 1990] veröffentlicht, der Netze solange veränderte, bis diese eine gestellte Aufgabe in einer vorgegebenen Anzahl von Zyklen lernten. Hier ist die Evolutionsstrategie der Gradientenmethode weit überlegen, da es keine bekannten Verfahren gibt, mit deren Hilfe der Einfluß einzelner Gewichtungen oder Units auf die Zahl der benötigten Zyklen bestimmt werden kann. Desweiteren könnte mittels der Gradientenmethode nicht das Verhalten bei Hinzunahme weiterer Gewichtungen oder Units vorhergesagt werden.

3.4.4 Präsentationsreihenfolge

Bisher wurde nicht darauf eingegangen, inwieweit eine bestimmte Reihenfolge der Präsentation der einzelnen Muster der Ein-/Ausgaberelation $\mathcal{R}$ einen Einfluß auf die Konvergenz des Lernverfahrens hat.

Bei der Präsentationsreihenfolge ist eine Reihe veschiedenster Möglichkeiten denkbar. Bei der *batch* Variante werden alle Muster dem gleichen Netz präsentiert. Die ermittelten Änderungen werden erst am Ende des Lernzyklus im Netz übernommen. Demnach hat die Reihenfolge der präsentierten Muster keinen Einfluß auf die Konvergenz.

Bei der *on-line* Variante ist die Konvergenz sehr wohl von der Reihenfolge der präsentierten Muster abhängig. Es ist beispielsweise denkbar, daß man durch eine modifizierte Präsentationsreihenfolge aus lokalen Minima herauskommt.

Literaturverzeichnis

[Baum und Haussler, 1989] E. B. Baum and D. Haussler. What size net gives valid generalization. *Neural Computation*, 1:151–160, 1989.

[Bialas, 1990] Till Bialas. Lernen bei Feed-Forward-Netzen als nichtlineare Optimierung. Technical report, Technische Universität Berlin, 1990. Interner Seminarbericht des Fachgebiets Informatik in Natur- und Ingenieurwissenschaften.

[Bourne und Kendall, 1973] D. E. Bourne and P. C. Kendall. *Vektoranalysis*. Teubner Studienbücherverlag, Stuttgart, 1973.

[le Cun *et al.*, 1990] Yann le Cun, John S. Denker, and Sara A. Solla. Optimal brain damage. In David S. Touretzky, editor, *Advances in Neural Information Processing Systems II*, pages 598–605. Morgan Kaufmann Publishers, San Mateo, California, 1990.

[le Cun, 1989a] Yann le Cun. Generalization and network design strategies. In Rolf Pfeiffer, Zoltan Schreter, Françoise Fogelman-Solié, and Luc Steels, editors, *Proceedings — Connectionism in Perspective*. Swiss Group for Artificial Intelligence and Cognitive Science (SGAICO), Elsevier Science Publishers B.V., 1989.

[le Cun, 1989b] Yann le Cun. A theoretical framework for back-propagation. In David Touretzky, Geoffrey Hinton, and Terrence Sejnowski, editors, *Proceedings of the 1988 Connectionist Models Summer School*, pages 21–28, San Mateo, CA, 1989. Morgan Kaufmann Publishers.

[Deker und Thomas, 1983] Uli Deker and Harry Thomas. Die chaos-theorie. *bild der wissenschaft*, pages 62–75, 1 1983.

[Eckmiller *et al.*, 1990] Rolf Eckmiller, Georg Hartmann, and Gert Hauske, editors. *Parallel Processing in Neural Systems and Computers*. Elsevier Scienc Publishers B.V., New York, 1990. ISBN 0-44488390-8.

[Fahlman und Hinton, 1987] Scott E. Fahlman and Geoffrey E. Hinton. Connectionist architectures for artificial intelligence. *IEEE Computer*, pages 100–109, January 1987.

[Hancock, 1989] Peter J. B. Hancock. Data representation in neural nets: An empirical study. In David Touretzky, Geoffrey Hinton, and Terrence Sejnowski, editors, *Proceedings of the 1988 Connectionist Models Summer School*, pages 11–20, San Mateo, CA, 1989. Morgan Kaufmann Publishers.

[van Hemmen *et al.*, 1990] J. L. van Hemmen, W. Gerstner, A. Herz, R. Kühn, B. Sulzer, and M. Vaas. Encoding and decoding of patterns which are correlated in space and time. In Georg Dorffner, editor, *Konnektionismus in Artifical Intelligence und Kognitionsforschung.6. Österreichische Artificial-Intelligence-Tagung (KONNAI)*, pages 153–162. Springer-Verlag, September 1990. Informatik-Fachberichte 252.

[Hertz *et al.*, 1991] John Hertz, Anders Krogh, and Richard G. Palmer. *Introduction to the Theory of Neural Computation*. Addison-Wesley Publishing Company, 350 Bridge Parkway, Redwood City, CA 94065, 1991. ISBN 0-201-50395-6.

[Hillis, 1985] Daniel W. Hillis. *The Connection Machine*. MIT Press, Cambridge, 1985.

[Hinton, 1986] G. E. Hinton. Learning distributed representations of concepts. In *Proceedings of the Cognitive Science Society*, pages 1–12, San Mateo, CA, 1986. Erlbaum.

[Hinton, 1989] Geoffrey E. Hinton. Connectionist learning pocedures. In JH. G. Carbonell, editor, *Machine Learning*, pages 185–234. A Bradford Book, Cambridge, Massachusetts, 1989.

[Hoffmann und Hofmann, 1971] Ulrich Hoffmann and Hannes Hofmann. *Einführung in die Optimierung mit Anwendungsbeispielen aus dem Chemie-Ingenieur-Wesen*. Verlag Chemie GmbH, Verlag Chemie GmbH, Weinheim/Bergstraße, 1971. ISBN 3-527-25340-8.

[Hush und Salas, 1988] D. R. Hush and J. M. Salas. Improving the learning rate of back-propagation with the gradient reuse algorithm. In *IEEE International Conference on Neural Networks*, pages 441–447, San Diego, CA, 1988. The Institute of Electrical and Electronic Engineers, Inc., IEEE San Diego Section and IEEE TAB Neural Network Committee.

[Jacobs, 1988] Robert A. Jacobs. Increased rates of convergence through learning rate adaption. *Neural Networks*, I:295–307, 1988.

[Katz, 1987] Bernard Katz. *Nerv, Muskel und Synapse: Einführung in die Elektrophysiologie*, volume 5 of *MED*. Georg Thieme Verlag Stuttgart, New York, 1987. Übersetzt von Friedrich-Wilhelm Bentrup und Roland Hengstenberg, ISBN 3-13-465305-2.

[Kindermann und Linden, 1988] J. Kindermann and A. Linden. Pattern completion and classification with back-propagation. In Kindermann, Jörg and Lischka, Christoph, editor, *Workshop Konnektionismus*, pages 95–121, St. Augustin, August 1988. Gesellschaft für Mathematik und Datenverarbeitung mbH. Arbeitspapiere der GMD, 329.

[Kohonen, 1989] Teuvo Kohonen. *Self-Organization and Associative Memory*. Springer Series in Information Sciences. Springer-Verlag, Springer-Verlag Berlin Heidelberg New York, third edition edition, May 1989.

[Kramer und Sangiovanni-Vincentelli, 1989] Alan H. Kramer and A. Sangiovanni-Vincentelli. Efficient parallel learning algorithms for neural networks. In David S. Touretzky, editor, *Advances in Neural Information Processing Systems I*, pages 40–48. Morgan Kaufmann Publishers, San Mateo, California, 1989.

[Lang und Witbrock, 1989] Kevin J. Lang and Michael J. Witbrock. Learning to tell two spirals apart. In David Touretzky, Geoffrey Hinton, and Terrence Sejnowski, editors, *Proceedings of the 1988 Connectionist Models Summer School*, pages 52–59, San Mateo, CA, 1989. Morgan Kaufmann Publishers.

[Linden und Kindermann, 1989] A. Linden and J. Kindermann. Inversion of multilayer nets. In *IEEE International Conference on Neural Networks*, pages II–425, San Diego, CA, 1989. The Institute of Electrical and Electronic Engineers, Inc., IEEE San Diego Section and IEEE TAB Neural Network Committee.

[Linden, 1989] Alexander Linden. Untersuchung von Backpropagation in konnektionistischen Systemen. Diplomarbeit, Rheinische Friedrich-Wilhelms-Universität Bonn und Gesellschaft für Mathematik und Datenverarbeitung mbH, Sankt Augustin, September 1989.

[Luenberger, 1984] David G. Luenberger. *Linear and Nonlinear Programming*. Addison-Wesley Company, Menlo Park, California, 1984.

[McClelland *et al.*, 1986] James L. McClelland, David E. Rumelhart, and Geoffrey E. Hinton. The appeal of parallel distributed processing. In *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, chapter 1. MIT Press/Bradford Books, 1986.

[McClelland und Rumelhart, 1988a] James L. McClelland and David E. Rumelhart, editors. *Explorations in Parallel Distributed Processing: A Handbook of Models, Programs, and Exercises*, volume Computational models of cognition and perception. MIT Press/Bradford Book, 1988.

[McClelland und Rumelhart, 1988b] James L. McClelland and David E. Rumelhart, editors. *Explorations in Parallel Distributed Processing: A Handbook of Models, Programs, and Exercises*, volume Computational models of cognition and perception. MIT Press/Bradford Book, 1988.

[Minsky und Papert, 1969] Marvin Minsky and Seymour Papert. *Perceptrons*. MIT Press, Cambridge, Massachusetts, 1969.

[Nieber, 1990] Christian Nieber. Methoden zur Verbeserung des Back-Propagation-Verfahrens– Anpassung der Lernrate und Gradient Reuse Algorithm. Technical report, Technische Universität Berlin, 1990. Interner Seminarbericht des Fachgebiets Informatik in Natur- und Ingenieurwissenschaften.

[Ornstein und Thompson, 1986] Robert Ornstein and Richard F. Thompson. *Unser Gehirn: das lebendige Labyrinth*. Rowohlt Verlag GmbH, Reinbek bei Hamburg, 1 edition, 1986. ISBN 3-498-05009-5. Übersetzt von Hainer Kober aus ¨The Amazing Brain¨.

[Ossen, 1990] Arnfried Ossen. *Zur Modularisierung and Interpretierbarkeit Neuronaler Netze*. PhD thesis, Technische Universität Berlin, 1990.

[Pfeiffer *et al.*, 1989] Rolf Pfeiffer, Zoltan Schreter, Françoise Fogelman-Solié, and Luc Steels, editors. *Proceedings — Connectionism in Perspective*. Elsevier Science Publishers B.V., 1989. ISBN 0-44488061-5.

[Prange, 1990] Stefan J. Prange. Emulation of biology-oriented neural networks. In Rolf Eckmiller, Georg Hartmann, and Gert Hauske, editors, *Parallel Processing in Neural Systems and Computers*, pages 79–82. Elsevier Scienc Publishers B.V., New York, 1990. ISBN 0-44488390-8.

[Radons *et al.*, 1990] G. Radons, H. G. Schuster, and D. Werner. Drift and diffusion in backpropagation networks. In *International Conference on Parallel Processing in Neural Systems and Computers (ICNC), Düsseldorf, FR Germany*, pages 261–264. Elsevier Science Publishers B.V., 1990.

[Rechenberg, 1973] Ingo Rechenberg. *Evolutionsstrategie*. Problemata. Frommann-Holzboog, 1973.

[Rissanen, 1989] J. Rissanen. *Stochastic Complexity in Statistical Inquiry*. World Scientific, Singapore, 1989.

[Rumelhart *et al.*, 1986a] David E. Rumelhart, James L. McClelland, and the PDP Research Group, editors. *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, volume 2, Psychological and Biological Models. MIT Press/Bradford Books, 1986.

[Rumelhart *et al.*, 1986b] David E. Rumelhart, James L. McClelland, and the PDP Research Group, editors. *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, volume 1, Foundations. MIT Press/Bradford Books, 1986.

[Salomon, 1989] Ralf Salomon. Adaptiv geregelte Lernrate bei Back-propagation. Technical Report 89-24, Technische Universität Berlin, 1989. Forschungsberichte des Fachbereichs Informatik.

[Salomon, 1990a] Ralf Salomon. Beschleunigtes lernen durch adaptive regelung der lernrate bei back-propagation in feed-forward netzen. In Georg Dorffner, editor, *Konnektionismus in Artifical Intelligence und Kognitionsforschung.6. Österreichische Artificial-Intelligence-Tagung (KONNAI)*, pages 173–178. Springer-Verlag, September 1990. Informatik-Fachberichte 252.

[Salomon, 1990b] Ralf Salomon. Ein Debugger zum Test von Evolutionsstrategien. Technical report, Technische Universität Berlin, 1990. Interner Bericht des Fachgebiets Informatik in Natur- und Ingenieurwissenschaften.

[Salomon, 1990c] Ralf Salomon. Improved convergence rate of back-propagation with dynamic adaption. In H.P. Schwefel and R. Männer, editor, *Parallel Problem Solving*, pages 269–273. Springer-Verlag, Oktober 1990. ISBN 3-540-54148-9.

[Saund, 1989] Eric Saund. Dimensionality-reduction using connectionist networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 11(3):304–314, March 1989.

[Schiffmann und Mecklenburg, 1990] Wolfram Schiffmann and Klaus Mecklenburg. Genetic generation of backpropagation trained neural networks. In *International Conference on Parallel Processing in Neural Systems and Computers (ICNC), Düsseldorf, FR Germany*, pages 205–208. Elsevier Science Publishers B.V., 1990.

[Schwefel, 1977] Hans-Paul Schwefel. *Numerische Optimierung von Computer-Modellen mittels Evolutionsstrategie*. Interdisciplinary systems research. Birkhäuser Verlag, Basel und Stuttgart, 1977. ISBN 3-7643-0876-1.

[Silva und Almeida, 1990] Fernando M. Silva and Luis B. Almeida. Speeding up backpropagation. In *Proceeding of NSMS - International Symposium on Neural Networks for Sensory and Motor Systems*. Elsevier Science Publishers B.V., March 1990.

[Stoer, 1983] Josef Stoer. *Einführung in die Numerische Mathematik I*. Springer-Verlag Berlin, 1983. ISBN 3-540-12536-1.

[Touretzky *et al.*, 1989] David Touretzky, Geoffrey Hinton, and Terrence Sejnowski, editors. *Proceedings of the 1988 Connectionist Models Summer School*. Morgan Kaufmann Publishers, San Mateo, California, 1989. ISBN 0-55860-015-9.

[Touretzky, 1989] David S. Touretzky, editor. *Advances in Neural Information Processing Systems I*. Morgan Kaufmann Publishers, San Mateo, California, 1989. ISBN 1-558-60015-9.

[Touretzky, 1990] David S. Touretzky, editor. *Advances in Neural Information Processing Systems II*. Morgan Kaufmann Publishers, San Mateo, California, 1990. ISBN 1-55860-100-7.

[U. Tietze, 1980] Ch. Schenk U. Tietze. *Halbleiter-Schaltungstechnik*. Springer, Springer-Verlag Berlin Heidelberg New York, 5 edition, 1980. ISBN 3-540-09848.

[Valentin Braitenberg and Almut Schüz, 1989] Valentin Braitenberg and Almut Schüz. Cortex: hohe Ordnung oder größtmögliches Durcheinander. *Spektrum der Wissenschaft*, pages 74–86, Mai 1989. Deutsche Ausgabe von: Scientific American.

[Waibel, 1989] Alex Waibel. Consonant recognition by modular construction of large phonemic time-delay neural networks. In David S. Touretzky, editor, *Advances in Neural Information Processing Systems I*, pages 215–223. Morgan Kaufmann Publishers, San Mateo, California, 1989.