MC- Computermagazin, Heft 2, 48-61 (1989)

Wie neuronale Netzwerke Roboter steuern können

Helge Ritter, Thomas Martinetz, Klaus Schulten

Neuronale Netzwerke sind ein vielversprechender Ansatz für die Verwirklichung am Vorbild des Gehirns ausgerichteter, lernfähiger Computer. Anhand von Beispielen aus der Robotersteuerung zeigen wir, was neuronale Netzwerke heute schon können und wo noch wichtige Forschungsfragen liegen. Ein Pascal-Programm gibt dem Leser die Möglichkeit zu eigenen Experimenten.

Robotersteuerung – eine komplexe Aufgabe

Industrieroboter bevölkern heute schon manche Fabrikhalle und helfen viele Produkte rationeller und billiger herzustellen. Der Roboter als Hilfe im Alltag ist aber dennoch weithin nicht in Sicht. Warum?

Heutige Industrieroboter kosten von 100 000 DM an aufwärts, aber dieser hohe Preis ist nicht allein die Ursache für die langsame Verbreitung des Roboters im Alltag. Grund ist vielmehr die außerordentliche Komplexität der Aufgabe, einen Roboter mit der Fähigkeit auszustatten, automatisch auf eine Alltagsumwelt zu reagieren und sinnvolle Arbeiten zu verrichten. Diese Aufgabe ist bis heute im wesentlichen ungelöst, und der Einsatz von Industrierobotern nur deshalb möglich, weil viele Industriearbeiten extrem stereotyp sind und durch immer wieder gleichartige, starre Bewegungsmuster ausgeführt werden können. Karosserielackierung oder Punktschweißen sind typische Beispiele solcher Arbeiten. Sorgt man für eine jedesmal gleiche Lage des Werkstücks, so kann der Roboter seine Bewegung blind und jedesmal gleichartig durchführen – gleichsam wie ein Tonbandgerät, das immer dieselbe, einmal gespeicherte Melodie, abspielt. Im Gegensatz zum Menschen kann der Roboter dies schneller, ausdauernder und genauer.

Sobald die Umwelt jedoch variabel wird, wie dies in den meisten Alltagssituationen der Fall ist, steht der blinde Roboter auf verlorenem Posten. Sinnvolles Agieren in einer variablen Umwelt benötigt Sensoren, besonders Sehen ist dabei vorteilhaft. Computersehen ist aber auch heute noch ein Gebiet mit vielen ungelösten Problemen. Immerhin ist die Erkennung und Lagebestimmung von bekannten Werkstücken unter günstigen Beleuchtungsbedingungen heutzutage möglich. Doch allein schon Aufnehmen und Eindrehen einer Schraube — eine dem Menschen lächerlich einfach erscheinende Aufgabe — verlangt dem Roboter weit mehr ab, als lediglich Art und Lage der vor ihm liegenden Werkstücke zu erkennen. Will der Roboter die Schraube aufnehmen, so muß er zunächst eine geeignete Griffposition auswählen. Die Wahl hängt im allgemeinen ab von der Form des Werkstücks, seiner Lage auf dem Tisch und der Anwesenheit anderer Objekte, die eine unter anderen Umständen geeignete

Griffposition behindern können. So kann die Schraube etwa zusammen mit vielen "Artgenossen" in einer Schachtel liegen. Auch der dem Werkstück zugedachte Verwendungszweck beeinflußt die Griffposition: so bedeutet es z.B. einen Unterschied, ob die Schraube nur zum Zweck des Transports oder aber zum späteren Einschrauben aufgenommen wird. Sind diese Fragen gelöst, so muß als nächstes eine geeignete Bewegungstrajektorie zum Bestimmungsort festgelegt werden. Hierbei müssen Hindernisse in Gestalt anderer Gegenstände berücksichtigt werden. In keinem Punkt der Bahn dürfen die Grenzdaten des Roboterarms (maximale Gelenkwinkel und -kräfte) überschritten werden und dennoch soll die Bewegung möglichst rasch erfolgen. Ist die Schraube schließlich am Zielort, muß der Roboter die beiden Gewinde auf hunderstel Millimeter genau ausrichten, um ein Einschrauben zu ermöglichen. Diese Aufgabe fällt uns leicht, weil wir nach ungefährem Aufsetzen der Schraube auf das Gewindeloch uns beim Ausrichten von den Gegenkräften des Gewindes leiten lassen. Eine ähnliche Strategie wird auch der Roboter brauchen, da entsprechend präzise Ortsangaben aus einem Kamerabild kaum je zu gewinnen sind.

Dies gibt einen leichten Eindruck von der Komplexität, die bereits mit "einfachen" Verrichtungen verbunden ist. Wir verdanken es unserem Gehirn, daß unser Bewußtsein meistens von der Wahrnehmung der ungeheuren Koordinationsleistung verschont bleibt, die hinter jeder unserer täglichen Verrichtungen steckt.

Unter Inkaufnahme eines hohen Programmieraufwands kann man für nicht allzu komplexe Aufgaben einige der auftretenden Probleme lösen (einen guten Überblick verschafft z.B. [2]). Gemessen am Menschen sind die Fähigkeiten der resultierenden Systeme aber immer noch kläglich, zudem ist eine wirtschaftliche Realisierung noch nicht möglich. Einem Roboter die Fähigkeit zur Bewältigung gewöhnlicher Alltagsaufgaben zu verleihen, übersteigt die heutigen Grenzen weit.

Wie macht es unser Gehirn?

Nur selten begegnen wir einer Situation, für die unser normales Bewegungsrepertoire nicht ausreicht. Dies war nicht immer so. Während unseres Lebens haben wir uns nach und nach dieses Bewegungsrepertoire durch Lernen erworben. Das meiste davon in den ersten Lebensjahren, aber auch in späteren Jahren konnten und können wir uns noch neue Bewegungsfähigkeiten aneignen, so etwa beim Erlernen eines Kunsthandwerks, eines Musikinstruments oder einer Sportart. Im Gegensatz zum Roboter haben wir dazu – glücklicherweise – nur sehr wenig explizite "Programmierung" nötig.

Es hat nicht an Anstrengungen von Seiten der "Künstlichen Intelligenz" gefehlt, diese eindrucksvolle Fähigkeit des Gehirns durch Computerprogramme nachzubilden, und es wurden dabei auch einige bemerkenswerte Erfolge erzielt. Diese

Programme, die auf plausibel erscheinenden Problemlösungsheuristiken basieren, funktionieren jedoch immer nur in der engen und genau definierten Modellumwelt, für die sie konzipiert worden waren. Diese Schwierigkeit hat zu Zweifeln an der Tragfähigkeit des heuristischen Ansatzes geführt. Daher haben in den letzten Jahren verstärkt Bemühungen eingesetzt, die von der Natur im Gehirn eingesetzten Lösungsstrategien besser zu durchschauen und zur Grundlage neuartiger, "neuronaler" Lernalgorithmen zu machen.

Das Gehirn geht völlig andere Wege als ein herkömmlicher Computer ([1],[3-6]). Anstelle eines oder weniger zentraler Prozessoren, die mit extrem hoher Geschwindigkeit lange, sequentielle Programme abarbeiten, setzt es eine große Anzahl (beim Menschen ca. 10^{11}) relativ langsam reagierender, dafür aber parallel arbeitender und eng vernetzter Nervenzellen oder "Neuronen" ein. Ein Großteil davon bildet beim Menschen eine ca. $0.2m^2$ große und 2-3mm dicke Schicht, die unter jedem Quadratmillimeter ca. 100000 Neuronen beherbergt und die, kunstvoll gefaltet, die Außenseiten unserer beiden Gehirnhemisphären bildet. Das Verhalten einzelner Neuronen kann sehr komplex sein, oft modelliert man es jedoch gemäß einem binären Element mit zwei möglichen Zuständen: das Neuron ist entweder im Ruhezustand oder es ist erregt und "feuert" dann einen elektrischen Impuls. Dieser wird über eine "Ausgabeleitung", das sogenannte Axon, bis zu mehreren Tausend anderen Nervenzellen zugeleitet. Umgekehrt empfängt jedes Neuron von ebensovielen Partnern Signale. Jedes Signal führt auf der Zellmembran des Neurons zu einer elektrischen Spannungsänderung, wobei sich die Spannungsänderungen der einzelnen Signale summieren. Immer wenn diese Summe einen Schwellwert übersteigt, wird das Neuron erregt und sendet seinerseits einen Impuls aus. Bevor ein Signal jedoch ein Neuron erreicht und zu der Summe beiträgt, muß es erst eine Kontaktstelle zwischen der Nervenfaser und dem Empfängerneuron passieren. Diese als Synapsen bezeichneten Übergangsstellen (von denen das menschlichen Gehirn die astronomische Zahl von 10^{15} besitzt) spielen eine Schlüsselrolle für die Funktion des Gehirns. Sie können nämlich ganz unterschiedliche "Leitfähigkeit" besitzen und sind daher für das Verschaltungsmuster zwischen den Neuronen wesentlich verantwortlich (die meisten Synapsen sind keine elektrischen, sondern "chemische Kontakte", wirken in ihrer Funktion aber wie ein elektrische Kontakte). Aufgrund der dichten Verschaltung ist die Aktivität der Neuronen einer komplexen Dynamik unterworfen. Im Ablauf dieser Dynamik steckt die Rechenleistung des Neuronensystems. Was das Neuronensystem berechnet, hängt von der Verschaltung zwischen den Neuronen ab, d.h. diese legt das "Programm" fest. Die "Dateneingabe" erfolgt, indem externe Sinnesrezeptoren einem Teil der Neuronen ein Aktivitätsmuster aufprägen, das über die interne Verschaltung mit der bereits vorhandenen Aktivität wechselwirkt — dies ist die Phase der "Berechnung" — bis schließlich Neuronen erfaßt

werden, deren Axone zu Muskeln führen und dort die "Datenausgabe" veranlassen.

Die Verschaltung, die diese Dynamik bestimmt, ist nicht fest, sondern sie kann sich im Laufe der Zeit in Abhängigkeit von der Netzwerkaktivität verändern. Diesem Umstand verdanken wir unsere Lernfähigkeit. Die Veränderlichkeit der Verschaltung beruht auf der Fähigkeit der Synapsen, ihre Leitfähigkeit in Abhängigkeit von ihrer "Benutzung" langfristig zu verändern. Die genauen Gesetzmäßigkeiten dieser Veränderungen sind bisher noch nicht im einzelnen bekannt. Mit hoher Wahrscheinlichkeit gibt es eine ganze Reihe von unterschiedlichen Regeln für verschiedene Typen von Synapsen. Eine verbreitete Annahme für das Änderungsverhalten vieler Synapsen ist die "Hebbsche Regel". Ihrzufolge erhöht sich die Leitfähigkeit einer Synapse von einem Neuron A zu einem Neuron B in dem Maße, in dem A zur Erregung von B beiträgt. Neuronen, die über solche Synapsen miteinander verschaltet sind, können ihre Verschaltung im Laufe der Zeit so ändern, daß sich Aktivitätsmuster, die häufig gleichzeitig auftreten, mehr und mehr gegenseitig begünstigen. Schließlich kann ein Aktivitätsmuster allein das andere auslösen: Das Netzwerk hat gelernt, zusammengehörende Aktivitätsmuster zu assoziieren.

Diese "innere Maschinerie" ist unserer unmittelbaren Wahrnehmung jedoch entzogen. Wir erleben stattdessen unser Bewußtsein als eine Wanderung durch Vorstellungen, die sich gegenseitig anstoßen oder die von äußeren Eindrücken wachgerufen werden. Die ganz ähnliche Abfolge von Aktivitätsmustern in einem neuronalen Netz legt daher eine Entsprechung zwischen Aktivitätsmustern und Vorstellungen nahe. Obwohl wir heute weit davon entfernt sind, diese Interpretation mit Details ausfüllen zu können, spricht eine Reihe von Umständen für eine solche Interpretation. So besteht eine große Ähnlichkeit zwischen der weiter oben geschilderten Assoziation von häufig zusammen aufgetretenen Aktivitätsmustern und der Assoziation häufig zusammen aufgetretener Gedanken (z.B. "Paris"–"Eiffelturm"). Elektrische Stimulierung bestimmter Gehirnareale (erforderlich bei manchen Gehirnoperationen) kann lebhafte Vorstellungsbilder auslösen. Unvollständige Aktivitätsmuster können von einem Netzwerk sinnvoll ergänzt werden, ähnlich wie wir Erinnerungen anhand fragmentarischer Information abrufen können.

Wesentliches Merkmal dieser Art des "Rechnens" mit Aktivitätsmustern ist die Verteilung der Gesamtaufgabe auf eine große Anzahl gleichartiger "Prozessoren", die alle einigen wenigen, gleichartigen Regeln unterliegen, und von denen jeder nur einen geringfügigen Beitrag zur Gesamtaufgabe leistet. Durch die parallele Arbeitsweise wird auch mit langsamen Bauelementen eine hohe Verarbeitungsgeschwindigkeit möglich. Gleichzeitig ergibt sich ein hohes Maß an Fehlertoleranz: der Ausfall einiger weniger Bauelemente kann die Gesamtleistung nur wenig verschlechtern. Anstelle der Programmierung tritt die Vorgabe einer geeigneten Vorverschaltung zwi-

schen den Neuronen und einiger Lernregeln für die Änderung dieser Verschaltung in Abhängigkeit von der Aktivität der Neuronen. Die Vorverschaltung braucht meist nur grob festgelegt werden. Durch Vorgabe von Beispielen kann sich das Netzwerk nach und nach selbst verbessern.

Die Wahl einer geeigneten Vorverschaltung und geeigneter Lernregeln kann man vielleicht ganz grob mit der Wahl einer Programmiersprache vergleichen. Je nach gewählter Programmiersprache kann die Programmierung einer festen Aufgabe einfach oder kompliziert ausfallen und daher unterschiedlich lange dauern. Bei einem neuronalen Netzwerk, das sich durch Lernen selber programmieren soll, ist dies ganz ähnlich: Erst die Wahl einer geeigneten Netzwerkstruktur führt zu raschem Lernen und einem "guten" Programm. Ähnlich wie bei den Programmiersprachen haben sich daher mittlerweile auch einige besonders vielseitig einsetzbare Netzwerktypen herausgebildet. Die bekanntesten davon sind vielleicht der Assoziative Speicher ([4,6]) und der Backpropagation-Algorithmus ([6]). Ein drittes, sehr interessantes Netzwerkmodell stammt von Kohonen ([4]) und wurde von den Autoren für die Anwendung auf Aufgabenstellungen aus der Robotik weiterentwickelt ([5]). Im folgenden Abschnitt werden wir dieses Modell kennenlernen.

Gehirn für den Roboter: Neuronale Netzwerke

Wir wollen nun ein neuronales Netzwerkmodell beschreiben, das sich als einfaches "Robotergehirn" eignet. Der Aufbau des Modells ist angelehnt an den Strukturen der Gehirnrinde. Die Neuronen der Gehirnrinde sind in einer Schicht angeordnet, die sich in eine Anzahl Felder unterteilen läßt. Jedes Feld ist für eine Teilaufgabe zuständig und läßt sich als ein "Team" vorstellen, dessen Mitglieder, die einzelnen Neuronen, einer gemeinsamen Aufgabe dienen. Sie werden über einlaufende Nervenfasern über die gerade vorliegende Aufgabensituation informiert. Jedes Mitglied ist aber nur jeweils für einen ganz engen Aufgabenbereich zuständig. In jeder Situation werden nur diejenigen Mitglieder aktiv, in deren Zuständigkeitsbereich die vorhandene Situation gerade fällt. Diese bestimmen dann die Antwort des Teams. Im Falle des Gehirns bestimmen die Synapsen, auf welche Signale ein Neuron reagiert, d.h. wie sein Zuständigkeitsbereich aussieht. Die Antwort setzt sich aus dem Impulsmuster der aktivierten Neuronen zusammen. (In einem Computermodell ist es einfacher, jedem Neuron als Ausgabesignal eine oder mehrere Zahlen zuzuordnen und auch seinen Zuständigkeitsbereich durch einige Zahlen festzulegen).

Bild 1 zeigt im unteren Teil ein "Team" aus neun Neuronen, die an den Plätzen eines 3×3-Gitters angeordnet sind. Zur Illustration stellen wir den Neuronen eine außerordentlich einfache (und für Anwendungen völlig uninteressante) Aufgabe, die aber das Modell gut veranschaulicht. Das Netzwerk soll für jeden beliebigen Punkt

der darüber dargestellten Fläche den Abstand $d = \sqrt{x_1^2 + x_2^2}$ vom Flächenmittelpunkt $(0,0)$ lernen. Jede Eingabe besteht aus dem Koordinatenpaar $\mathbf{x} = (x_1, x_2)$ eines beliebig herausgegriffenen Punktes. Jedes Neuron besitzt in der Fläche einen Zuständigkeitsbereich und prüft, ob der Eingabepunkt in seinen Zuständigkeitsbereich fällt. Mathematisch geschieht dies im Modell, indem jedem Neuron $\mathbf{r}$ ein Vektor $\mathbf{w}_\mathbf{r}^{in}$ zugeordnet wird. Der Zuständigkeitsbereich von Neuron $\mathbf{r}$ umfaßt dann alle Eingabewerte $\mathbf{x}$, die am nächsten bei seinem Vektor $\mathbf{w}_\mathbf{r}^{in}$ liegen (d.h. die Eingabe $\mathbf{x}$ fällt in den Zuständigkeitsbereich von Neuron $\mathbf{r}$, wenn der Abstand $||\mathbf{x} - \mathbf{w}_\mathbf{r}^{in}||$ zu $\mathbf{w}_\mathbf{r}^{in}$ kleiner ist als jeder andere Abstand $||\mathbf{x} - \mathbf{w}_{\mathbf{r}'}^{in}||$). In unserem Beispiel ergeben sich als Zuständigkeitsbereiche Parzellen in der "Eingabefläche", und jeder Vektor $\mathbf{w}_\mathbf{r}^{in}$ ist ein Zeiger vom Neuron am Gitterpunkt $\mathbf{r}$ ins Zentrum der zugehörigen Parzelle. Fällt der Eingabepunkt etwa in den schraffierten Bereich, so wird das Neuron $\mathbf{r}$ in der Gittermitte aktiviert. Dieses bestimmt dann die Antwort des Teams. Im vorliegenden Beispiel ist die Antwort ein einzelner Zahlenwert für den Abstand $d = \sqrt{x_1^2 + x_2^2}$ und ist in Bild 1 für jedes Neuron durch die Höhe eines Balkens angedeutet. Im allgemeinen Fall kann das Modell jedem Neuron einen ganzen Satz von Ausgabewerten zuordnen, die wiederum in einem Vektor $\mathbf{w}_\mathbf{r}^{out}$ zusammengefaßt sind (dies verleiht den Modellneuronen mehr Flexibilität, die Natur könnte die Funktion derartiger "formaler" Neuronen durch mehrere parallelgeschaltete Einzelneuronen realisieren).

Zu Beginn kennen die Neuronen weder die richtigen Ausgabewerte, noch eine günstige Verteilung ihrer Zuständigkeitsbereiche. Dies ist in Bild 1 erkennbar, wo die Vektoren $\mathbf{w}_\mathbf{r}^{in}$ (gestrichelte Linien) alle zufällig und die Ausgabewerte $\mathbf{w}_\mathbf{r}^{out}$ (Balkenhöhen) alle gleichgroß gewählt sind. Dies entspricht einem Team mit lauter völlig "dummen" Mitgliedern, die ihre Zuständigkeitsbereiche und Ausgangswerte anfangs willkürlich festgelegt haben. Natürlich ist auch die Anfangsleistung entsprechend miserabel. Ziel ist nun in einer Lernphase die Zuständigkeitsbereiche und die Antworten der jeweils aktivierten Mitglieder allmählich immer besser zu organisieren, bis schließlich eine sehr gute Gesamtleistung möglich wird.

Dazu muß ein Mitglied, immer wenn es aktiviert wurde, sein Verhalten durch einen "Lernschritt" für das nächste Mal geeignet korrigieren. Dazu stehen ihm zwei Wege offen: Es kann einerseits seinen Zuständigkeitsbereich verlagern, etwa hin auf häufige und daher vielleicht besonders wichtige Situationen. Es kann andererseits sein Antwortverhalten so zu verbessern versuchen, daß sein Beitrag zu einem geringeren Gesamtfehler als vorher führt. Dazu braucht es eine Rückmeldung über den erzielten Fehler, sowie eine Regel, nach der es sein Verhalten in Abhängigkeit von diesem Fehler verbessern kann. Beides kann im Prinzip durch biologisch plausible Synapsenregeln nach Art der erwähnten Hebbschen Regel geschehen. Bei der Rea-

lisierung im Computer ist die Beachtung biologische Einschränkungen jedoch nicht unbedingt zwingend, und man kann bei der Wahl der Lernregeln auch andere Gesichtspunkte, wie etwa größtmögliche mathematische Einfachheit, berücksichtigen.

In unserem Modell besteht der Lernschritt darin, daß das aktivierte Neuron $\mathbf{r}$ seine beiden "Zeiger" $\mathbf{w}_\mathbf{r}^{in}$ und $\mathbf{w}_\mathbf{r}^{out}$ ein Stück in Richtung auf den angebotenen Eingabewert $\mathbf{x}$ (in unserem Beispiel der Punkt $\mathbf{x}$ in der schraffierten Parzelle) bzw. einen verbesserten Ausgabewert $\mathbf{y}$ hin verschiebt. Die erste Maßnahme verlagert die Zuständigkeitsbereiche der Neuronen, die zweite verbessert ihre Ausgabewerte. Mathematisch wird dies durch die beiden Gleichungen

$$\mathbf{w}_\mathbf{r}^{in,\,neu} = \mathbf{w}_\mathbf{r}^{in,\,alt} + h \cdot (\mathbf{x} - \mathbf{w}_\mathbf{r}^{in,\,alt})$$

$$\mathbf{w}_\mathbf{r}^{out,\,neu} = \mathbf{w}_\mathbf{r}^{out,\,alt} + h \cdot (\mathbf{y} - \mathbf{w}_\mathbf{r}^{out,\,alt})$$

beschrieben (im Falle unserer Anwendungen ist $\mathbf{y}$ meist nicht unmittelbar gegeben und muß, wie weiter oben erwähnt, anhand einer geeigneten Regel aus dem vom Netzwerk begangenen Fehler berechnet werden). Dabei ist h eine Zahl zwischen Null und Eins, die die Größe eines Lernschritts festlegt.

Nun nimmt in einem biologischen Neuronennetz aufgrund der gegenseitigen Verschaltung benachbarter Neuronen niemals nur ein einzelnes Neuron an einer Aktion teil. Vielmehr wird immer eine ganze Anzahl benachbart liegender Neuronen gleichzeitig aktiviert. Daher müssen wir in unserem Modell das Neuron $\mathbf{r}$, in dessen Parzelle der Eingangswert fällt, als Zentrum einer lokalisierten "Insel" aktivierter Neuronen betrachten und jedes Neuron $\mathbf{r}'$ im Bereich dieser Insel am Lernschritt teilnehmen lassen. Diese Teilnahme sollte jedoch umso geringer sein, je weiter ein Neuron vom Zentrum $\mathbf{r}$ der Insel entfernt ist. Mathematisch berücksichtigen wir dies dadurch, daß wir die beiden vorstehenden Gleichungen zu

$$\mathbf{w}_{\mathbf{r}'}^{in,\,neu} = \mathbf{w}_{\mathbf{r}'}^{in,\,alt} + h(\mathbf{r}' - \mathbf{r}) \cdot (\mathbf{x} - \mathbf{w}_{\mathbf{r}'}^{in,\,alt})$$

$$\mathbf{w}_{\mathbf{r}'}^{out,\,neu} = \mathbf{w}_{\mathbf{r}'}^{out,\,alt} + h(\mathbf{r}' - \mathbf{r}) \cdot (\mathbf{y} - \mathbf{w}_{\mathbf{r}'}^{out,\,alt})$$

erweitern. Hier durchläuft $\mathbf{r}'$ nun alle Neuronen (einschließlich $\mathbf{r} = \mathbf{r}'$) und die frühere Lernschrittweite h ist durch eine Funktion $h(\mathbf{r}' - \mathbf{r})$ ersetzt, die vom Abstand $\mathbf{r}' - \mathbf{r}$ eines Neurons vom Zentrum $\mathbf{r}$ der Insel aktivierter Neuronen abhängt und wie in Bild 3 aussieht. $h(\mathbf{r}' - \mathbf{r})$ ist also groß in der Nähe des Inselzentrums, und nimmt nach außen rasch ab. Dadurch nehmen jedesmal nur Neuronen innerhalb einer Nachbarschaftsregion um das ausgewählte Neuron $\mathbf{r}$ an einem Lernschritt teil. Im Falle unseres einfachen Beispiels erregt ein beispielsweise in den schraffierten Zuständigkeitsbereich fallender Eingangswert am meisten das zentrale Neuron und wählt es

als Inselzentrum $\mathbf{r}$ aus. Weniger stark erregt werden seine vier unmittelbaren Gitternachbarn $\mathbf{r}'$ und noch weniger stark die vier Neuronen $\mathbf{r}'$ auf den Gitterecken. Beim anschließenden Lernschritt verschieben alle neun Neuronen ihre Zeiger $\mathbf{w}_{\mathbf{r}'}^{in}$ in Richtung auf den markierten Punkt und ihre Ausgabewerte $\mathbf{w}_{\mathbf{r}'}^{out}$ (Balkenhöhen) in Richtung auf den Wert des Abstandes des markierten Punkts vom Flächenzentrum, aber in entsprechend unterschiedlichem Ausmaß. Durch den Lernschritt verbessert jedes Neuron seine Reaktion bei einer Wiederkehr desselben Eingangswerts. Dabei stehen die Neuronen miteinander in Wettbewerb, und der Wettbewerbsvorteil eines Neurons ist umso größer, je besser es bereits auf den Eingangsreiz anspricht. Häufige Eingangssignale $\mathbf{x}$ ziehen auf Kosten seltenerer Signale viele Zuständigkeiten an. Dadurch werden ihnen allmählich mehr Neuronen zugeordnet und das Netzwerk kann auf solche Signale in feineren Abstufungen reagieren, eine nützliche Eigenschaft, wie sie auch von biologischen Nervensystemen her bekannt ist. Wählen wir in unserem Beispiel die Eingabepunkte mit überall auf der Fläche gleicher Wahrscheinlichkeit aus, so ergibt sich nach einiger Zeit die Situation in Bild 2. Die Neuronen haben ihre Zuständigkeiten so verteilt, daß benachbarte Neuronen für benachbarte Parzellen zuständig sind, und alle Parzellen sind regelmäßig angeordnet und etwa gleich groß. Zugleich hat jedes Neuron einen Ausgabewert (Balkenhöhe) gelernt, den es den Punkten seiner Parzelle als Abstand von der Flächenmitte zuordnet. Das Netzwerk ordnet also allen Punkten einer Parzelle denselben Abstand vom Zentrum zu. Eine hohe Genauigkeit erfordert daher kleine Parzellen und somit viele Neuronen. Eine wirksamere Verbesserungsmöglichkeit besteht darin, daß die Neuronen innerhalb einer Parzelle linear interpolieren. Die nötige Verallgemeinerung des Modells, die hier nicht dargestellt werden soll, wurde bei der im folgenen Kapitel beschriebenen Simulation erprobt und ermöglicht eine erhebliche Genauigkeitssteigerung.

Nach soviel Theorie wollen wir im folgenden Abschnitt einem "Neuronenteam" beim Lernen zusehen. Wir werden sehen, wie das Netzwerk nach und nach lernen kann, den Greifer eines Roboterarms an vorgegebene Zielorte im Gesichtsfeld einer Stereokamera zu positionieren, ohne vorher irgendetwas über die erforderlichen geometrischen Daten zu wissen. Damit erwirbt unser Roboter eine wichtige Grundfähigkeit, die "Hand-Auge-Koordination", zur Ausführung visuell kontrollierter Bewegungen.

Hand-Auge-Koordination als Beispiel

Bild 4 zeigt uns den im Computer simulierten Roboter, der im Beispiel die Hand-Auge-Koordination lernen soll. Der Roboter besteht aus einem dreigelenkigen Arm, der hinter einem Tisch montiert ist. Der Sockel ist um seine eigene Achse drehbar, und die beiden anderen Gelenke können den Arm in einer vertikalen Ebene bewegen. Um in seiner Umwelt erfolgreich agieren zu können, braucht unser Roboter

Informationen über die Lage anvisierter Objekte im Raum. Uns werden diese Informationen durch unsere Augen geliefert. Entsprechend statten wir unseren Roboter mit zwei Kameras aus, die den Arbeitsbereich beobachten sollen. Dabei spielt es eine wichtige Rolle, daß wir zwei und nicht nur eine Kamera verwenden, weil erst dann — wie mit den Augen — die Dreidimensionalität des Raumes erfaßt wird.

Der Roboter soll lernen, seinen Greifer zu beliebigen Zielpunkten auf der Arbeitsfläche zu führen. Die Lage der Zielpunkte auf der Arbeitsfläche wird zufällig ausgewählt. Das Ziel wird jedesmal von den Kameras erfaßt, und deren Signale dem Neuronennetz zugeführt. Jedes Neuron ist für ein ganz bestimmtes, von den Kameras beobachtetes Raumgebiet zuständig. Wenn ein Zielpunkt dort ausgewählt wird, dann wird dieses Neuron aktiviert und gibt seine Ausgabegröße an die Motorsteuerung weiter. Die Lage des Ziels steht dabei jedoch nicht einfach in dreidimensionalen Koordinaten zur Verfügung, sondern die Kameras liefern nur die beiden Orte, an denen der Zielpunkt in ihren beiden Gesichtsfeldern erscheint. Das Netzwerk muß daraus geeignete Ausgabegrößen $\mathbf{w}^{out}$ für die Gelenkmotoren gewinnen, die den Greifer richtig positionieren. Weitere Informationen, etwa über die Abmessungen der Roboterarmsegmente und über die Stellungen der beobachtenden Kameras, erhalten die Neuronen nicht. Vielmehr müssen sie diese geometrischen Daten für die richtige Umsetzung der Kamerainformation in Motorsignale selbst lernen. Daher wird der Arm anfangs eine fehlerhafte Endposition ansteuern. Die Abweichung wird von den Kameras beobachtet und jedesmal zu einer Verbesserung des benutzten Ausgabewertes $\mathbf{w}^{out}$ verwendet. Anschließend wird dem Roboter ein anderer Zielpunkt vorgegeben, wodurch er Gelegenheit für einen neuen Lernschritt erhält. Der Roboter stellt dadurch ein in sich geschlossenes, autonomes System dar.

Nun aber zu den auf einer VAX 750 und einer MicroVAX II durchgeführten Simulationen. In unserem Beispiel verwenden wir ein neuronales Netzwerk aus 252 Neuronen, die als 12×21 Rechteckgitter angeordnet sind. Bild 5 zeigt uns das Lernverhalten des Roboters, wie es Kamera 1 aus Abb.1 beobachtet. Die Arbeitsfläche sowie der Roboterarm sind in jedem Bild angedeutet. Die Bilder der linken Spalte zeigen uns die Aufteilung der Zuständigkeitsbereiche unter den Neuronen. Jedem Neuron sind zwei "Bildparzellen" zugeordnet, eine im Gesichtsfeld von Kamera 1 und eine im Gesichtsfeld von Kamera 2. Ein Neuron wird aktiv, wenn die Bildpunkte des Ziels in beide Parzellen gleichzeitig fallen. Wie man sich mittels der Gesetze der Strahlenoptik überlegen kann, gibt es Paare von Parzellen, für die dies für keine Lage des Ziels eintritt. Welche Paare relevant sind, hängt von der Position der Kameras relativ zum Arbeitstisch ab. Unser neuronales Modellnetz ist in der Lage, während der Versuchsbewegungen seine Neuronen automatisch auf die relevanten Paare zu verteilen. Außerdem, und das ist eine sehr wichtige Eigenschaft, ordnen sich im Netz benachbarte Neuronen benachbarten Parzellen in den Bildebe-

nen der Kameras zu. In unserer Darstellung in der linken Spalte entspricht jeder Netzknoten einem Neuron und ist ins Zentrum seiner Parzelle im Gesichtsfeld von Kamera 1 gesetzt. Knoten zu im Gitter benachbarten Neuronen sind durch Linien verbunden. Dadurch wird die Zuordnung benachbarter Parzellen zu benachbarten Neuronen erkennbar.

Anfangs ordnen wir jedem Neuron ein beliebiges Paar von Parzellen zu, denn wir wissen ja nicht, welche Paare wirklich vorkommen. Die Verbindungen der nächsten Nachbarn überschneiden sich daher vielfach und es entsteht das Liniengewirr im obersten Bild. Doch bereits nach 4000 Lernschritten, zu sehen im mittleren Bild der ersten Spalte, haben die Neuronen ihre Zuständigkeitsbereiche so umorganisiert, daß das Netz "entfaltet" auf der Arbeitsfläche liegt. Dies bedeutet, daß im Gitter benachbarte Neuronen jetzt nebeneinanderliegende Parzellen im Gesichtsfeld "verwalten". Allerdings sehen wir, daß die Verteilung der Neuronen noch nicht ganz gerecht ist, denn einige Neuronen verwalten größere, andere kleinere Parzellen. Die Verteilung ist jedoch nicht starr, wie uns die Momentaufnahme vielleicht suggerieren könnte. Vielmehr setzen sich in dieser Phase des Lernens mal die einen, mal die anderen Neuronen in ihrem Wettbewerb um die Größe der Zuständigkeitsbereiche stärker durch, und das Netz ist dadurch in ständiger Bewegung. Die Fluktuationen werden mit der Anzahl der Versuchsbewegungen immer geringer, und nach 20000 Versuchen hat das Netz schließlich seine Endlage eingenommen. Das unterste Bild der ersten Spalte zeigt diesen Zustand. Jedes Neuron verwaltet jetzt ungefähr gleich große Raumbereiche. Wir sehen, daß das Netz scheinbar im Raum auf der Arbeitsfläche liegt. Dies bedeutet, daß sich die Neuronen gerade denjenigen Parzellen im Gesichtsfeld zugeordnet haben, die durch Zielpunkte aus dem Arbeitsbereich angesprochen wurden.

Gleichzeitig zu diesem Vorgang hat unser Roboter auch seine Armpositionierung gelernt. Die mittlere Spalte zeigt das Lernen des Grobpositionierens und die rechte Spalte die Fähigkeit zur Feinansteuerung. Beim Grobpositionieren soll der Greifer ins Zentrum des Raumbereichs des angesprochenen Neurons geführt werden. Die für jedes Neuron tatsächlich erreichte Position ist mit einem Kreuz gekennzeichnet. Um den noch vorhandenen Fehler sichtbar zu machen, ist diese Position mit dem gewünschten Zielort in den Bildern durch eine Linie verbunden. Am Anfang weiß unser Roboter noch nichts über die gegebenen geometrischen Verhältnisse. Die langen Fehlerlinien des obersten Bildes zeigen uns, daß unser Robotermodell, ähnlich wie ein Baby, sich zunächst planlos bewegt und gewünschte Zielpunkte teilweise bei weitem verfehlt. Nach und nach werden die Fehler jedoch immer geringer, hier zu sehen im mittleren Bild nach 4000 Versuchen, wo die Länge der Verbindungslinien schon wesentlich kürzer geworden ist. Nach 20000 Lernschritten dann ist die erforderliche Genauigkeit und Sicherheit beim Positionieren des Greifers erreicht,

ohne daß dem System jemals Kenntnisse über Geometrie oder Informationen über Armsegmentlängen und Kamerastellungen von außen mitgeteilt wurden.

Zur Überprüfung des "Feingefühls" veranlassen wir den Roboter zu kleinen, L-förmigen Bewegungen an verschiedenen Orten des Tisches. Die Feineinstellung wird benötigt, falls der anzusteuernde Zielpunkt nicht genau im Zentrum der Parzelle, sondern etwas daneben liegt. Auch hier sehen wir wieder Momentaufahmen des Verhaltens am Anfang, nach 4000, sowie nach 20000 Lernschritten. Wir sehen deutlich, daß der Roboter am Anfang noch sehr unkoordinierte Bewegungen macht. Auch dies ist wieder Ausdruck seiner noch vorhandenen Unkenntnis der geometrischen Gegebenheiten. Jedoch lernt das System schnell, und am Ende werden die verlangten L's gut ausgeführt. Damit ist der Arm jetzt an jeder Stelle der Arbeitsfläche in der Lage, auch die Feinpositionierung genauestens auszuführen. Die Genauigkeit liegt dabei bei einigen Zehntelmillimetern und hängt direkt mit der Größe des verwendeten neuronalen Modellnetzes zusammen. Je mehr Neuronen wir verwenden, um so besser wird die Positionierungsgenauigkeit des Roboters. Wenn also demnächst in zunehmendem Maße Parallelrechner verfügbar werden, lassen sich die Netzgrößen wesentlich steigern, und die Restfehler können entsprechend weiter gesenkt werden.

Unser Roboter bleibt ständig lernfähig und kann sich ändernden äußeren Bedingungen, wie etwa infolge von Verschleiß, permanent anpassen. Damit ist unser Roboter auch in dieser Beziehung biologischen Systemen sehr ähnlich, denn gerade das Problem sich ändernder mechanischer Eigenschaften scheinen wir mühelos zu erledigen. Ob nun als Kind, als Erwachsener oder im Alter, die mechanischen Eigenschaften unserer Gliedmaßen sind infolge von Wachstum und Alterung zu keiner Zeit völlig konstant, aber trotzdem beherrschen wir die Koordination unserer Bewegungen jederzeit in sehr hohem Grade. Wir bekommen schon bald nach der Geburt ein Gefühl für unsere Bewegungen im Raum und verlieren dies auch nicht mehr bei sich ändernden Gegebenheiten.

Was sind nun die wesentlichen Prinzipien, nach denen unser System lernt? Am wichtigsten ist die Fähigkeit der Neuronen, die komplexe Gesamtaufgabe untereinander aufzuteilen. Dadurch ergibt sich für jedes einzelne Neuron eine wesentlich einfachere Teilaufgabe, die es auch ohne kompliziertes Verhalten bewältigen kann. Bei unserem Beispiel der Hand-Augen-Koordination ist jedes Neuron nur für einen kleinen Raumbereich verantwortlich. Dadurch braucht die Ausgabe jedes Neurons auch nur sehr ähnliche Greiferpositionen kodieren. Mathematisch ausgedrückt bedeutet das, daß die exakte Eingabe-Ausgaberelation durch Linearisierungen an vielen einzelnen Stützstellen angepaßt wird. Die Anpassung der Ausgabegröße $\mathbf{w}^{out}$ versucht jedes Neuron dabei mittels eines Gradientenabstiegsverfahren zu erreichen. Dazu braucht das aktive Neuron bei jedem Bewegungsversuch die über die Kameras ge-

lieferte Information über die Lage des Zielpunkts sowie die tatsächliche Endstellung des Greifers.

Das hier als Gradientenabstieg bezeichnete Verfahren ist aus der Optimierungstheorie bekannt. Für die Bewältigung der Hand-Augen-Koordination mittels eines neuronalen Netzwerkmodells ist diese Lernprozedur allein allerdings nicht ausreichend. Es hat sich gezeigt, das der gewünschte Lernerfolg nur dann erzielt werden kann, wenn sich die Neuronen auch zum Optimieren der Ausgabegrößen nicht isolieren, sondern "Teamgeist" entwickeln. Ohne Hilfe seiner "Kameraden" muß jedes Neuron für sich allein den für seinen Zuständigkeitsbereich passenden Ausgabewert w^{out} herausfinden. Verhält sich der ganze Verband der Neuronen aber als Team, so werden sich benachbarte Neuronen gegenseitig zu Hilfe kommen. Wir erwähnten bereits, daß im Netz benachbarte Neuronen auch benachbarte Zuständigkeitsbereiche besetzen. Das bedeutet aber, daß im Netz benachbarte Neuronen auch jeweils ähnliche Werte für ihre Ausgabegrößen lernen müssen. Wenn also jetzt ein Neuron seinen Ausgabewert verbessert, dann ist es für das Gesamtssyssten vorteilhaft, diese Verbesserung einfach seinen Nachbarn mitzuteilen und diese an der Verbesserung teilhaben zu lassen. Je weiter die Nachbarn im Rechteckgitter vom aktiven Neuron entfernt sind, um so weniger wird ihnen vom Lernerfolg mitgeteilt, denn um so unterschiedlicher werden ja auch die gewünschten Ausgabewerte. Durch diesen Teamgeist wird der Lernerfolg überhaupt erst ermöglicht. Außerdem wird das Verhalten des Systems dadurch sehr robust, es wird auch bei einer sehr ungünstigen zufälligen Anfangsbelegung der Ein- und Ausgabegrößen in der Lage sein, sich den gegebenen äußeren Bedingungen anzupassen. Zusätzlich steigert sich die Lerngeschwindigkeit des Roboters beträchlich, was ja auch plausibel ist, wenn sich die Mitglieder des Teams gegenseitig helfen.

Solche "Kooperation" ist eine aus der Neurophysiologie wohlbekannte Eigenschaft von Nervennetzen. Es sind in der Regel nicht einzelne, sondern ganze Verbände von Neuronen, die ihr Verhalten äußeren Gegebenheiten anpassen. Es ist daher sehr interessant festzustellen, daß unser künstliches Robotermodell dieses Verhalten ebenfalls unbedingt benötigt, um in seiner Umwelt erfolgreich agieren zu können.

Wir haben jetzt gesehen, wie ein Roboter die sehr wichtige Aufgabe des Positionierens auf der Basis eines neuronalen Netzwerkmodells lernen kann. Was aber passiert, wenn unser Roboter dabei sehr schnelle Bewegungen ausführen soll? Dabei treten komplizierte Steuerungsprobleme auf, die in den Bereich der sogenannten "Dynamik eines Roboterarms" fallen. Wie diese Probleme aussehen und wie auch sie mit unserem neuronalen Netzwerkmodell angegangen werden können, soll im nächsten Abschnitt beschrieben werden.

Auch ein Roboterarm ist träge

Ein Roboter wird im Gegensatz zum Menschen nie ermüden, — aber mit der Trägheit hat auch er zu kämpfen. Und wie es für die Trägheit typisch ist, macht sie sich besonders bei raschen Bewegungen störend bemerkbar. Schuld daran sind die Massen der einzelnen Roboterarmsegmente. Sie können, wie jede Masse, niemals "plötzlich" bewegt werden, sondern müssen zunächst beschleunigt und dann wieder abgebremst werden. Dies macht sich bei langsamen Bewegungen oder sehr leichten Armen kaum bemerkbar und der Roboter kann sich damit begnügen, lediglich die Geometrie zu berücksichtigen. Dies haben wir im vorigen Kapitel kennengelernt. Sobald die Bewegungen schneller oder die Armsegmente schwerer werden, wird eine Berücksichtigung der Massenträgheitseffekte jedoch unvermeidbar. Es genügt dann nicht mehr, jedes Gelenk für sich allein in der gewünschten Weise zu verstellen, weil die an jedem Gelenk aufzubringende Drehkraft (man spricht auch von Drehmoment) jetzt auch davon abhängt, wie sich die anderen Gelenke gerade bewegen. Alle Teile des Roboterarms sind also in komplizierter, "nichtlinearer" Weise miteinander gekoppelt. Die genaue Form dieser Kopplung ist einerseits wieder von der Geometrie des Arms, also seinen Abmessungen und der gegenseitigen Anordnung seiner Gelenke, zusätzlich aber auch noch von seiner Masse und ihrer Verteilung in den einzelnen Armsegmenten abhängig (mathematisch beschrieben durch sogenannte "Trägheitstensoren"). Bei vollständiger Kenntnis dieser Daten — ein in der Praxis seltener Fall — ist eine Vorausberechnung der erforderlichen Drehkräfte im Prinzip möglich. Jedoch sind die erforderlichen Berechnungen mittels der beschreibenden Bewegungsgleichungen selbst für einen dreigliedrigen Arm schon außerordentlich kompliziert. Durch einen Algorithmus zum Lernen der Berechnungen wird diese Schwierigkeit vermieden und zugleich die Kenntnis der genauen Geometrie und der Trägheitstensoren des Arms entbehrlich. Wir wollen im folgenden Abschnitt zeigen, wie unser Netzwerkmodell diese Aufgabe lösen kann.

Unser Roboter soll lernen, aus jeder Position heraus den Greifer seines Arms durch einen kurzen "Drehmomentpuls" in den Gelenken auf eine beliebig vorgegebene Geschwindigkeit zu beschleunigen — eine Aufgabe, zu deren Lösung er die ihm anfangs unbekannten Armträgheitsmomente sowie die Armgeometrie bereits voll berücksichtigen und daher nach und nach aufgrund von Versuchsbewegungen lernen muß.

Bild 6 zeigt die Ergebnisse unserer Computersimulation, diesmal mit einem Netz von 360 Neuronen. Dargestellt ist eine Draufsicht auf den Arbeitstisch. Während der Simulation wird der Greifer nacheinander an zufällig gewählte Orte auf der Arbeitsfläche positioniert. Für jeden Vorgabeort wird vom Roboter verlangt, den Greifer durch einen kurzen Drehmomentpuls auf eine bei jedem Versuch neu fest-

gelegte Anfangsgeschwindigkeit zu beschleunigen. Die dabei tatsächlich bewirkte Geschwindigkeit wird registriert und die Abweichung zur Sollgeschwindigkeit wieder zum Verbessern der Ausgabegröße w^{out} des angesprochenen Neurons benutzt. w^{out} legt fest, wie sich die nötigen Drehmomente aus der Sollgeschwindigkeit ergeben.

An jedem Ort des Arbeitstisches ist der Zusammenhang zwischen Geschwindigkeit und Drehmomenten natürlich anders, denn durch die jeweils andere Ausgangsstellung des Roboterarms sind die Hebelverhältnisse jedesmal verschieden. Dieses Problem wird ähnlich wie in unserem vorherigen Beispiel gelöst. Jedes Neuron spezialisiert sich wieder automatisch auf einige wenige Ausgangskonfigurationen des Arms und wird dadurch verantwortlich für eine kleine Parzelle auf dem Arbeitstisch, in der sich der Greifer in seiner Ausgangsstellung befinden kann. Dadurch wird die recht komplizierte Aufgabe des Gesamtsystems wieder auf viele kleine, für jedes einzelne Neuron dann leichter zu lösende Teilaufgaben reduziert. Die obere Bildreihe von Abb.3 zeigt, wie sich die Neuronen während der Versuchsbewegungen den Arbeitstisch aufteilen. Anfangs ist die Aufteilung von uns willkürlich vorgegeben worden. Als Folge der Versuchsbewegungen verschieben sich die Zuständigkeitsbereiche dann wieder derart, daß am Ende des Lernvorgangs im Rechteckgitter benachbart angeordnete Neuronen benachbarte Parzellen auf dem Tisch verwalten und jedes Neuron für einen etwa gleich großen Bereich verantwortlich wird.

Die untere Bildreihe zeigt uns, wie jedes Neuron die Regelung der Drehmomente für die in seine Zuständigkeit fallenden Armstellungen gelernt hat. Dazu wird der Greifer ins Zentrum jeder einzelnen Parzelle gesetzt. Zum Testen soll der Roboter aus dieser Position seinen Greifer nacheinander auf eine fest vorgegebene Geschwindigkeit jeweils parallel zu den Kanten des Tisches beschleunigen. Die tatsächlich erreichte Anfangsgeschwindigkeit wird in unserer Darstellung durch einen Pfeil veranschaulicht, dessen Länge den Betrag der Geschwindigkeit angibt und dessen Richtung in die Richtung der resultierenden Bewegung zeigt. Da unser Robotersystem am Anfang nichts über seine Massen und Abmessungen weiß, sind die resultieren Bewegungen im obersten Bild noch recht chaotisch. Nach 500 Versuchsbewegungen sind die Fehler schon sehr viel kleiner, und nach 10000 Lernschritten werden tatsächlich die geforderten Geschwindigkeiten erreicht. Das Robotersystem hat seine für die Regelung der Drehmomente erforderlichen "dynamischen Eigenschaften" gelernt.

Die wesentlichen Prinzipien des Lernalgorithmus entsprechen denen beim Lernen der Hand-Augen-Koordination. Wichtig ist wieder die selbständige Aufteilung der Gesamtaufgabe unter den einzelnen Neuronen. Außerdem ist auch wieder die gegenseitige Unterstützung unter den benachbarten Neuronen beim Verbessern der Ausgabegrößen unerläßlich für das Funktionieren. Der Hauptunterschied zwischen

den beiden Beispielen liegt lediglich in der Art der verarbeiteten Eingabe- und Ausgabesignale. In einem Fall ging es darum, die Ort-Winkel-Relation des Roboters zu adaptieren, wozu allein geometrische Eigenschaften, die sogenannte "Kinematik" gelernt werden mußte. Im zweiten Fall lernte das System eine Beschleunigung-Drehmoment-Relation, wozu noch zusätzlich physikalische Daten wie Massenverteilungen und Trägheitsmomente, die "Dynamik", benötigt wurden.

Der Weg ist das Ziel

Mit der Fähigkeit, die Kinematik und Dynamik eines Arms zu lernen, löst unser neuronales Netzwerk zwei wichtige Robotersteuerungsaufgaben. Wie wir eingangs gesehen haben, erfordern praktisch wichtige Aufgabestellungen jedoch noch weitere Fähigkeiten. Eine der wichtigsten ist die Planung einer möglichst günstigen Bewegungstrajektorie. Dies ist ein Optimierungsproblem und wir wollen nun anhand eines Beispiels zeigen, daß unser Netzwerkmodell auch zur Lösung derartiger Aufgaben geeignet ist. Ein Pascal-Programm (Listing 1) soll dem Leser eigene Experimente hierzu ermöglichen.

Eine häufige Industrieaufgabe ist das Bohren von Löchern, etwa in Leiterplatten. Soll ein Roboter diese Aufgabe erledigen, so muß er der Reihe nach alle Lochpositionen mit einem Bohrwerkzeug ansteuern und dann beim nächsten Werkstück neu beginnen.

Zur Minimierung von Zeitbedarf und Abnutzung ist dabei eine Reihenfolge erstrebenswert, die für jede solche "Rundtour" den Gesamtweg zwischen den Lochpositionen minimiert. Dieselbe Situation bietet sich auch einem Handlungsreisenden, der eine Reihe Städte besuchen muß und dazu den kürzesten Weg wählen möchte. Daher ist diese Aufgabe auch als "Handlungsreisendenproblem" bekannt.

Mathematisch handelt es sich dabei um ein endliches Suchproblem: im Prinzip könnte man alle denkbaren Wege auflisten, um davon einfach den kürzestmöglichen auszuwählen. In der Praxis scheitert man mit diesem Ansatz jedoch sehr rasch. Für L Lochpositionen gibt es $\frac{1}{2}(L-1)! = \frac{1}{2} \cdot 1 \cdot 2 \cdot 3 \ldots (L-2) \cdot (L-1)$ unterschiedliche Reihenfolgen (bei offenen Enden wären es sogar $L!$). Bereits für nur 20 Lochpositionen bräuchte ein Computer, der pro Sekunde 1000 Wege prüft, mit diesem Verfahren nahezu 2 Millionen Jahre Rechenzeit und bei 30 Löchern wäre selbst für einen Cray-Supercomputer das Alter des Universums zu kurz. Dieses extrem rapide Anwachsen abzusuchender Alternativen ist als "kombinatorische Explosion" bekannt und ist charakteristisches Merkmal vieler interessanter, sogenannter NP-harter, Probleme. Nach einer berühmten, aber bislang unbewiesenen Vermutung der "algorithmischen Komplexitätstheorie" läßt sich diese Schwierigkeit auch durch geschicktere Algorithmen nur teilwiese beheben, jedoch nicht entscheidend über-

winden. Für viele Anwendungen ist aber glücklicherweise eine sehr gute, wenn auch nur nahezu optimale Lösung vollständig ausreichend. Aber selbst dann ist die direkte Suche noch viel zu zeitaufwendig. Wir wollen nun zeigen, wie auch hier unser neuronales Netzwerkmodell rasch eine gute Lösung finden kann.

Wir benutzen dazu ein "Team" aus Neuronen, die diesmal als Ring angeordnet sind. Der Ring enthält mindestens soviele Neuronen, wie Lochpositionen zu besuchen sind, eine größere Anzahl Neuronen ist jedoch günstiger. Jedes Neuron soll sich für eine Lochposition entscheiden. Zusammen mit der Ringanordung der Neuronen ergibt sich daraus eine Reihenfolge, in der die Lochpositionen zu besuchen sind. Um einen möglichst kurzen Weg zu erhalten, müssen die Neuronen bei ihrer Entscheidung auch diesmal wieder kooperieren. So ist es besonders günstig, wenn Neuronen, die im Ring aufeinanderfolgen, sich auch für möglichst eng benachbarte Lochpositionen entscheiden. Andererseits müssen am Ende die Neuronen ihre "Stimmen" über alle Lochpositionen verteilt haben, da der Weg andernfalls Lochpositionen auslassen würde (dies wird wesentlich erleichtert, wenn man pro Bohrposition mehrere, etwa zwei, besser drei, Neuronen vorsieht). Die erste Forderung ist lokaler, die zweite globaler Natur. Für das einzelne Neuron wird sich daraus in der Regel ein Konflikt ergeben. Je besser diese Konflikte gelöst werden, desto kürzer ist der am Ende erhaltene Weg. Um die Konflikte möglichst gut zu lösen, findet die Entscheidungsbildung allmählich und in ständiger gegenseitiger "Tuchfühlung" der Neuronen statt. Diese Tuchfühlung erfolgt anfänglich über große Ringdistanzen hinweg und ermöglicht den Neuronen eine Art "Grobabstimmung". In dieser Phase entwickelt jedes Neuron eine Bevorzugung von Lochpositionen eines begrenzten Gebiets, trifft aber noch keine klare Entscheidung. Nach und nach wird die Kooperation zwischen den Neuronen auf immer kürzere Ringdistanzen eingeschränkt. Im selben Maße verengt sich für jedes Neuron der Kreis seiner Kandidaten für die endgültige Entscheidung. Am Ende hat jedes Neuron nur noch einen Kandidaten behalten, und die Reihenfolge der Lochpostionen liegt eindeutig fest.

Listing 1 ist ein Pascal-Programm zur Simulation dieses Prozesses. Das Programm wurde mit der "ST Pascal Plus" Entwicklungsumgebung des Atari erstellt und benutzt auch deren GEM-Bibliothek für die graphische Darstellung. Das Programm sucht eine Trajektorie für ein quadratisches Werkstück. Die x- und y-Koordinaten der Bohrpositionen sind im Feld *bohrpos* gespeichert. Zur Demonstration wählt das Programm zu Beginn mittels eines Zufallszahlengenerators *lochzahl* zufällig im Quadrat positionierte Lochorte. Das Programm verwendet *neuronzahl* Neuronen. Jedes Neuron trifft seine Entscheidung für eine Lochposition durch einen "Zeiger", den es im Quadrat frei umherbewegen kann. $w(r)$ ist die Position des Zeigers von Neuron r. Je näher der Zeiger eines Neurons bei einer Lochposition liegt, desto eindeutiger hat sich das Neuron für diese Position entschieden. Zu Beginn wer-

den die Zeiger auf einer Kreislinie verteilt. Von dort aus sollen die Neuronen ihre Zeiger schrittweise verschieben, bis jedes Neuron genau auf eine Lochposition deutet. Ein Programmlauf dauert *maxiters* Schritte. Ein Verschiebungsschritt kommt zustande, indem eine Lochposition u zufällig ausgewählt wird und das Neuron mit dem nächstliegenden Zeiger, nennen wir es s, diesen ein Stück weiter auf die Lochposition u zubewegt. Alle übrigen Neuronen r kooperieren mit diesem Schritt, indem auch sie ihre Zeiger ein Stück in Richtung auf den Ort u zubewegen. Diese "Anteilnahme" wird umso geringer, je weiter r und s im Ring auseinanderliegen. Mathematisch läßt sich dies so beschreiben: Für alle Neuronen r ($r = s$ eingeschlossen) setzen wir

$$w(r)^{neu} = w(r)^{alt} + h(r,s,a,b)(u - w(r)^{alt})$$

Die Funktion $h(r,s,a,b)$ bestimmt die Schrittweite und ist gegeben durch

$$h(r,s,a,b) = a \cdot \exp\left(-d(r,s)^2/b^2\right).$$

$d(r,s) = \min(|r-s|, N-|r-s|)$ ist der Ringabstand zwischen r und s. $h(r,s,a,b)$ fällt mit wachsendem $d(r,s)$ von seinem größtmöglichen Wert a allmählich auf sehr kleine Werte ab. Parameter b legt fest, ab welcher Entfernung sich die Abnahme bemerkbar macht und wird nach jedem Schritt verringert. Ein großer Startwert b_{Start} (im Programm $b_{Start} = neuronzahl/8$) ermöglicht anfangs die Kooperation vieler Neuronen. Ein kleiner Endwert b_{Ende} (im Programm $b_{Ende} = 0.7$) erzwingt gegen Schluß für jedes Neuron eine eindeutige Wahl. Die Abnahme von b wird nach jedem Schritt durch Multiplikation mit einem festen Abnahmefaktor $q = (b_{Ende}/b_{Start})^{(1/maxiters)}$ Schritt erreicht

$$b^{neu} = b^{alt} \cdot q.$$

Zu Beginn fragt das Programm nach Eingaben für die Anzahlen *lochzahl* und *neuronzahl* der Bohrpositionen bzw. der Neuronen sowie für die Anzahl *maxiters* der Iterationsschritte. Eine typischer Programmlauf ergibt sich für *lochzahl* = 30, *neuronzahl* = 90 und *maxiters* = 3000. Alle 50 Iterationsschritte werden die Bohrpositionen zusammen mit den erreichten Zeigerpositionen der Neuronen dargestellt. Der Verlauf der Trajektorie wird sichtbar gemacht, indem die Zeigerpositionen in der durch die Ringanordnung der Neuronen festgelegten Reihenfolge verbunden werden. Dadurch läßt sich gut verfolgen, wie in der Anfangsphase die Trajektorie zunächst ihre ungefähre Gestalt annimmt, und wie in dem Maße, in dem die Kooperation zwischen den Neuronen abnimmt, die Zuordnung zwischen Neuronen und Bohrpositionen genauer wird und die feineren Details der Trajektorie allmählich hervortreten (Abb.4). Durch Variation der maximalen Schrittzahl und der Anzahl der Neuronen kann man leicht den Einfluß dieser Größen studieren. Bei zuwenigen Neuronen

kommt es vor, daß die Trajektorie einige Orte ausläßt. Bei zu knapp bemessener Schrittzahl geraten die die Neuronen in "Streß" und können sich nicht mehr ausreichend untereinander abstimmen, wodurch sich längere Trajektorien ergeben.

Wohin führt die Forschung?

Lernen von Hand-Auge-Koordination, Armdynamik und Planung günstiger Trajektorien sind wichtige Einzelprobleme für einen Roboter. Wir beginnen heute zu verstehen, wie neuronale Netzwerke solche Aufgaben lösen können. Die besondere Stärke neuronaler Netzwerke beruht dabei auf ihrer inhärenten Parallelität, ihrer Lernfähigkeit und ihrer dadurch bedingten Flexibilität. Außerdem zeigen uns biologische Gehirne, daß eine auf diesem Ansatz basierende, höchst erfolgreiche Lösung existiert. Auf dem Weg auch nur in die Nähe dorthin sind aber noch zahlreiche Fragen zu lösen, sowohl beim Studium und Verständnis des biologischen Vorbilds, als auch bei der Konzeption leistungsfähigerer neuronaler Algorithmen.

Die in fast allen Modellen wichtigsten Parameter, die Synapsenstärken, sind experimentell nur in ganz wenigen Fällen zugänglich. Daher klafft heute noch eine weite Lücke zwischen experimentell zugänglichen Resultaten und theoretischen Modellen. Neuartige Experimentiertechniken öffnen jedoch ständig neue "Fenster ins Gehirn" und geben Hoffnung, diese Lücke allmählich zu schließen. So kennt man heutzutage Farbstoffe, mit denen man lebendes Hirngewebe anfärben kann, und die in Abhängigkeit von der elektrischen Aktivierung des Gewebes ihren Farbton ändern. Dadurch kann man neuronale Aktivitätsmuster optisch direkt aufnehmen. Mittels moderner computertomographischer Methoden (PET, NMR) läßt sich das momentane Gehirnstoffwechselniveau bis zu Ortsauflösungen im Millimeterbereich aufzeichnen und damit sogar beim Menschen ohne jeglichen Eingriff die Verteilung zerebraler Aktivierung untersuchen.

Auf der theoretischen Seite eröffnen leistungsfähigere Computer das Tor zur Simulation immer anspruchsvollerer Netzwerkmodelle. Bereits einfache Netzwerke sind mathematisch nur außerordentlich schwierig zu analysieren. In den meisten Fällen sind wir daher auf Computersimulationen angewiesen. Aber auch auf leistungsfähigen Rechnern sind gegenüber dem biologischen Gehirn drastische Vereinfachungen notwendig. Dabei ist keineswegs klar, wieviel von der für die Gehirnfunktion tatsächlich wichtigen Struktur in diesen Vereinfachungen überlebt. Es ist nahezu sicher, daß viele wichtige "Tricks" des Gehirns in den heutigen Modellen noch nicht enthalten sind. Umgekehrt legt mathematische Plausibilität oft Architekturen und Lernregeln nahe, deren biologische Realisierung zumindest fraglich ist. Aus der Sicht des Anwenders braucht dies jedoch nicht beunruhigen, solange solche "unbiologischen" Netzwerke zu interessanten Leistungen in der Lage sind. In

beiden Fällen wird aber das Herausschälen der für die jeweilige Funktion wichtigen Strukturmerkmale eine wesentliche Aufgabe künftiger Forschung sein.

Die meisten Ansätze beschränken sich bisher meist auf die Untersuchung von Einzelnetzwerken einer einheitlichen Architektur. Eine gewaltige Erhöhung des Potentials wird sich aus der Integration mehrerer neuronaler Einzelmodule zu einem größeren Gesamtsystem ergeben, ähnlich, wie sich die Anwendungen konventioneller Programmiersprachen als Folge der Entwicklung der Unterprogrammtechnik und der Möglichkeit des Zusammenlinkens komplexer Programme aus kleineren Modulen vervielfacht haben. Dies erfordert die Entwicklung eines Repertoires neuronaler "Standardmodule", deren Verhalten und Eignungsbereich gut verstanden ist. Die in diesem Beitrag vorgestellten Beispiele zeigen, wie solche Module aussehen könnten. Andere Teilaufgaben, wie z.B. assoziatives Speichern oder frühe Bildverarbeitung, verlangen andere Typen von Netzwerkmodulen (siehe z.B. [1]) mit anderen Eigenschaften.

Die Lernfähigkeit neuronaler Netze hat zu der häufig anzutreffenden Meinung geführt, daß es sich dabei um eine Art "algorithmisches Allheilmittel" handelt, das immer dann, wenn wir selbst nicht weiterwissen, für uns die Lösung lernt. Diese Wunschvorstellung wird wohl immer Wunsch bleiben (theoretisch kann schon eine Turingmaschine "alles" lernen. Die entscheidende Frage ist stets die benötigte Rechenzeit). Auch ein neuronales Netzwerk benötigt zum effizienten Lernen einer bestimmten Aufgabe ein gewisses Maß an Vorstrukturierung und eine geeignete Kodierung seiner Eingangsdaten.

Je komplexer die Aufgabe, desto mehr Vorstrukturierung ist in der Regel erforderlich. Dies ist auch in der Natur nicht anders. So lernt ein junger Vogel binnen Tagen Flugmanöver, die jeden Piloten und Flugzeugkonstrukteur vor Neid erblassen lassen. Wäre das Vogelgehirn für diese Aufgabe nicht geeignet vorstrukturiert, so hätte der Vogel mindestens ähnliche Probleme wie ein menschlicher Flugschüler. Menschen besitzen ein spezielles Sprachzentrum, das uns zum Erlernen von Sprache befähigt. Schon bei Primaten fehlt dieses Zentrum, und entsprechend fruchtlos verliefen auch alle Versuche, ihnen mehr als nur einige rudimentäre Symbolbedeutungen beizubringen. Die Liste ließe sich leicht fortsetzen. Wir ersehen daraus die Wichtigkeit einer geeigneten "neuronalen Infrastruktur" als Basis für das Erlernen einer Aufgabe. Die Wahl einer geeigneten Infrastruktur ist ein wichtiges Designproblem, dessen Lösung bisher noch weitgehend auf empirischer Erfahrung beruht. Die systematische Klassifikation von Aufgaben nach der dafür benötigten Netzwerkstruktur ist eine wichtige Voraussetzung, um für gut definierte Anwendungsgebiete, wie etwa in der Robotik, Spezifikation und Kombination geeigneter Netzwerke automatisieren zu können und dadurch dem eingangs erwähntem Wunschziel zumindest

näherzukommen.

Eng verknüpft mit der Frage nach einer geeigneten Netzwerkstruktur ist das Problem der Verallgemeinerung. Wir erwarten von einem lernfähigen System nicht nur die Wiedergabe früher kennengelernter Beispiele, sondern vielmehr die Fähigkeit zu sinnvoller Übertragung gelernten Wissens auf neue Situationen, d.h. Verallgemeinerung. Welche Art von Verallgemeinerung dabei sinnvoll ist, ist jedoch keineswegs immer klar. Ein einfaches Beispiel macht dies deutlich. Wie lautet das nächste Element der Folge 2, 4, 6, 8...? Den meisten Lesern dürfte wohl "10" als "richtige" Antwort erscheinen. Die Folge könnte aber genausogut von den Werten des Polynoms $P(x) = 2x + (x-2)(x-4)(x-6)(x-8)$ für $x = 1, 2, 3, 4, \ldots$ herrühren. In diesem Falle wäre die "korrekte" Fortsetzung $P(5) = 19$, und ebensogut könnte es sich um die ersten Ziffern der Dezimalentwicklung des Bruchs 39/158 handeln. Welche Verallgemeinerung die "richtige" ist, liegt ohne zusätzliche Information nicht fest. Die Entscheidung für "10" trafen wir, weil diese Möglichkeit uns "näherliegt" als andere, aber Leser, deren Telefonnummer wie "2468..." beginnt, spüren hier vielleicht schon deutlich, daß sie aufgrund einer anderen "Vorstrukturierung" zu einer ganz anderen "Verallgemeinerung" neigen. In einem neuronalen Netzwerk bestimmt die Netzwerkarchitektur, welche Art Verallgemeinerung für das Netzwerk naheliegt. Ein Netzwerk, das im gewünschten Sinne gut verallgemeinert, kann seine Aufgabe mit wesentlich weniger Trainingsschritten lernen, als ein Netzwerk mit ungünstigem Verallgemeinerungsverhalten. Daher ist ein genaues Verständnis des Zusammenhangs zwischen Netzwerkarchitektur und Verallgemeinerungsverhalten eine wichtige Voraussetzung für den Entwurf effizient lernender Netzwerke.

Von ebensogroßer Bedeutung ist die Wahl einer geeigneten Datenkodierung. Eine ungünstige Datenkodierung kann die Lösung einer Aufgabe unnötig erschweren. Dies merkten schon die alten Römer beim kaufmännischen Rechnen, denn schon eine einfache Addition in ihrer Zahlennotation bereitet einige Schwierigkeiten. Umgekehrt kann eine geschickte Datenkodierung eine Problemlösung erheblich erleichtern, indem sie günstige Verallgemeinerungen nahelegt oder Transformationen vereinfacht. Müssen wir viel mit Zweierpotenzen umgehen, werden wir z.B. das Hexadezimalsystem dem Zehnersystem vorziehen. Allerdings sind Zahlensysteme gerade Beispiele für "digitale" Datenkodierungen, die besonders auf die Art von Problemen zugeschnitten sind, für die menschliche Gehirne — und daher wohl auch neuronale Netzwerke — nicht besonders gut geeignet sind. Von neuronalen Netzwerken erwarten wir vielmehr die Lösung der zahlreichen Aufgaben, die uns mühelos leicht fallen, heutige Computer aber erheblich überfordern. Vieles deutet darauf hin, daß wir dabei "analoge" Datenkodierungen bevorzugen. So kommt uns etwa das analoge Zifferblatt einer Uhr zur Zeitplanung weit mehr entgegen als sein digitales Gegenstück. Die Zifferblattanzeige ist meist weniger genau, macht dafür

aber wesentlich mehr Beziehungen zwischen Zeitpunkten und Zeitabschnitten sichtbar und ist gleichzeitig robuster, etwa gegenüber teilweiser Verdeckung. Dies sind genau die Stärken, die biologische Informationsverarbeitung auszeichnen, und ihre Nachbildung erfordert neben der Entwicklung neuer Algorithmen auch die Erforschung von Datenkodierungen, die diese Stärken mittragen.

Das Gebiet der "neuronalen Netzwerke" beinhaltet also noch eine ganze Reihe offener und faszinierender Forschungsprobleme. Lösungen zu diesen Fragen hat die Natur in jahrmillionenlanger Evolution gefunden. Sie liegen uns vor, und mit vereinten Kräften beginnen Neurobiologen, Physiker, Mathematiker, Computerwissenschaftler und Psychologen diese Lösungen zu entschlüsseln. Angesichts der für die Zukunft absehbaren hohen Bedeutung des Gebiets hat das Bundesministerium für Forschung und Technologie im vorigen Jahr ein Verbundprojekt ins Leben gerufen, das die Erforschung neuronaler Informationsverarbeitung zum Ziel hat und dem auch die Autoren angehören. Wenn man bedenkt, daß breite Anstrengungen auf diesem Gebiet erst seit wenigen Jahren erfolgen, sind die bereits erzielten Fortschritte sehr ermutigend. Wir dürfen gespannt auf die Zukunft sein.

Literatur:

[1] Buhmann J., Divko R., Ritter H., Schulten K., Physik und Gehirn — wie dynamische Modelle von Nervennetzen natürliche Intelligenz erklären. MC 9/1987

[2] Fu K.S., Gonzalez R.C., Lee C.S.G. (1987), Robotics — Control, Sensing, Vision and Intelligence. Mc-Graw-Hill Int. Editions.

[3] Kandel E.R., Schwartz J.H. (1985) Principles of Neural Science (2. Aufl.).Elsevier, New York.

[4] Kohonen T. (1988) Self-Organization and Associative Memory (2. Aufl.). Springer Series in Information Sciences 8, Heidelberg

[5] Ritter H., Martinetz T., Schulten K., Neuroinformatik selbstorganisierender Abbildungen. Addison-Wesley-Verlag, Bonn (in Vorbereitung).

[6] Rumelhart D.E., McClelland J.L. (1984), Parallel Distributed Processing I&II, MIT-Press, Cambridge, Massachusets.

Bild 1 Anfangszustand eines "Teams" aus neun Neuronen, die Abstände von Punkten einer darüber dargestellten Eingabefläche lernen sollen.

Bild 2 Zustand des Neuronenteams, wenn es seine Aufgabe gelernt hat.

Bild 3 Abhängigkeit der Lernschrittweite $h(\mathbf{r}' - \mathbf{r})$ vom Abstand $||\mathbf{r}' - \mathbf{r}||$ eines Neurons vom Inselzentrum $\mathbf{r}$.

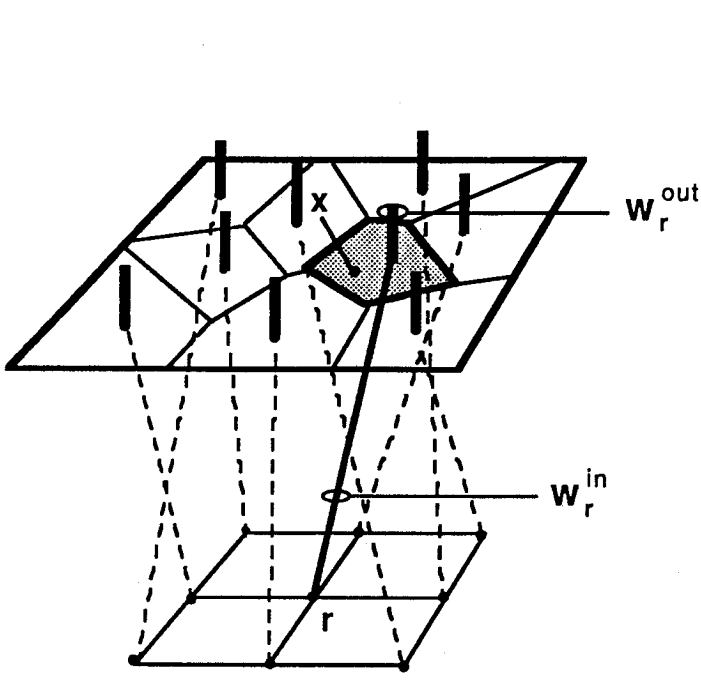
Bild 4 Das Robotersystem, welches die Hand-Auge-Koordination lernt. Der Roboterarm soll den markierten, von außen zufällig vorgegebenen Zielpunkt auf der Arbeitsfläche anfahren. Kamera 1 und Kamera 2 beobachten die Szene. Im linken Bildteil ist dargestellt, wie der Zielpunkt in den beiden Kameragesichtsfelder erscheint. Die beiden Orte des Zielpunkts in den Kameragesichtsfeldern werden in der Größe x zusammengefaßt und an das neuronale Netz übergeben, wo das zuständige Neuron aktiviert wird. Dessen Ausgabesignal $\mathbf{w}_{out}$ bestimmt die Bewegung der drei Gelenkmotoren zum Positionieren des Greifers.

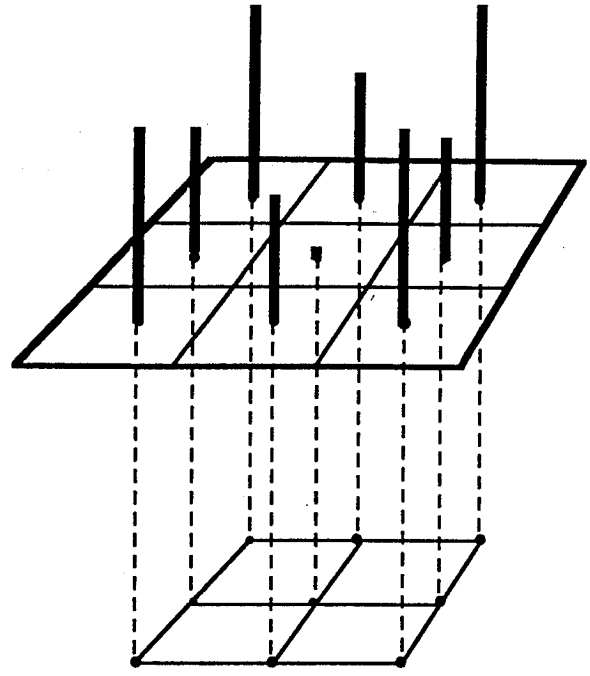
Bild 5 Eine Reihe von Bildern zeigt uns, wie der Roboter nach und nach die korrekte Plazierung seiner "Hand" lernt. Jedes Bild zeigt Roboterarm und Arbeitsfläche (durch Linien nur schematisch wiedergegeben) aus der Sicht von Kamera 2. Die linke Spalte veranschaulicht, wie sich die Zuständigkeiten der Neuronen für die Punkte des Kameragesichtsfelds allmählich regelmäßig anordnen (oben: Anfangszustand, Mitte: nach 4000 Lernschritten, Unten: am Ende der Simulation). Dazu wurde jedes Neuron des Netzes ins Zentrum seiner "Zuständigkeitsparzelle" gezeichnet und mit seinen Gitternachbarn durch Linien verbunden. Die mittlere Spalte zeigt die erreichte Plaziergenauigkeit, wenn als Zielorte die Parzellenmitten gewählt werden. Der tatsächlich erreichte Ort ist durch ein Kreuzsymbol und die Abweichung vom Vorgabeort durch eine Linie dargestellt. Zu Beginn sind die Fehler sehr groß (lange Linien, oberstes Bild), am Ende dagegen nicht mehr erkennbar (unterstes Bild). Die rechte Spalte zeigt den Lernfortschritt des Roboters anhand der Fähigkeit, "L"-förmige Testfiguren auf die Tischebene zu zeichnen.

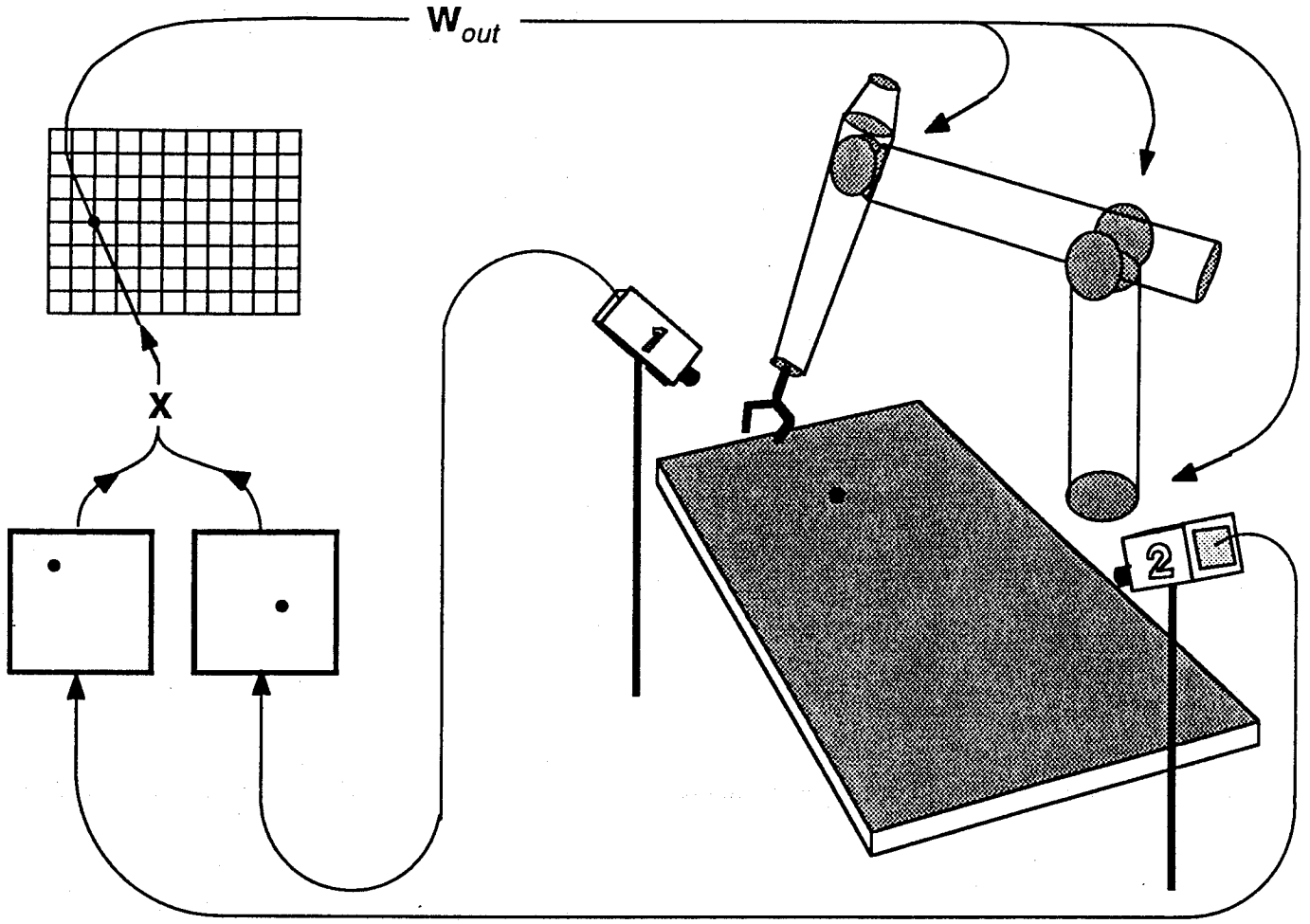
Bild 6 Hier sehen wir, wie der Roboterarm seine Trägheit kontrollieren lernt. Die obere Bildreihe zeigt, wie die Neuronen ihre Zuständigkeiten auf dem Arbeitstisch (Rechteck) allmählich ordnen. Die untere Reihe zeigt die Ausführung von Testbewegungen. Von jeder Parzellenmitte aus sollte die Hand mit jeweils gleicher Geschwindigkeit nach (in Bildrichtung) rechts bzw. oben bewegt werden. Die Pfeile

geben die tatsächlich erreichten Geschwindigkeiten an und zeigen, wie der Roboter nach und nach seine Armdynamik lernt.

Bild 7 Bildung einer Trajektorie für ein quadratisches Werkstück mit 30 zufällig positionierten Bohrlöchern. Ausgehend von einer Kreisformation verschieben die Neuronen nach und nach ihre "Zeiger", bis sich am Ende die gesuchte Trajektorie ergibt.


x
w_r^{out}
w_r^{in}
r




W_{out}
X
1
2

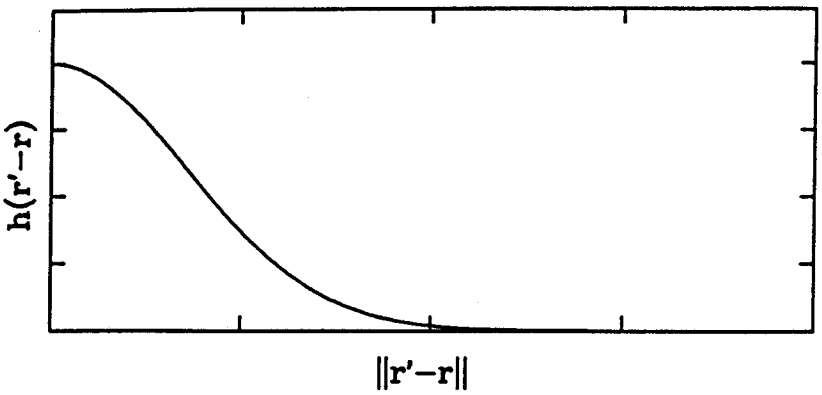


Bild 3

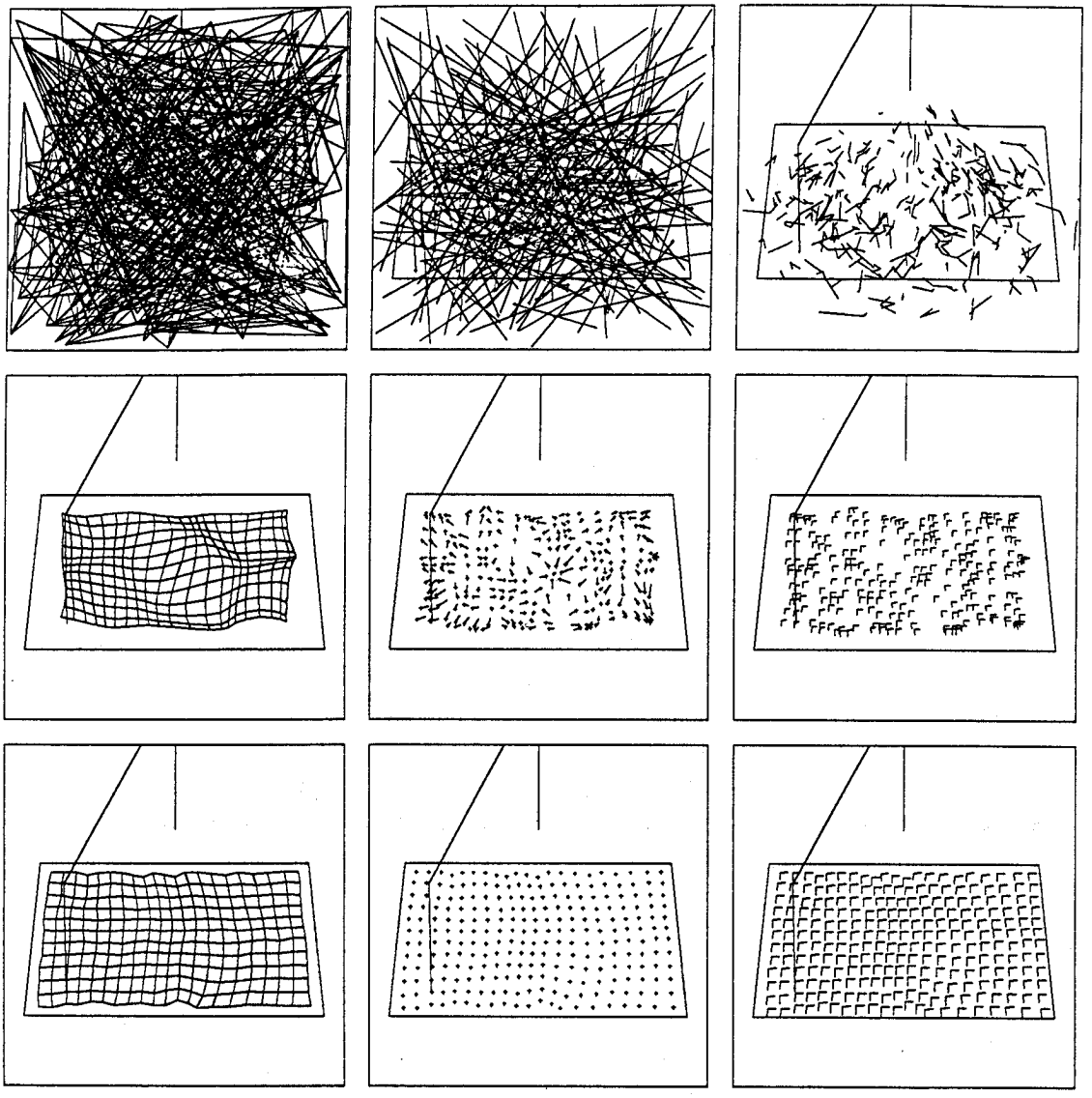


Bild 5

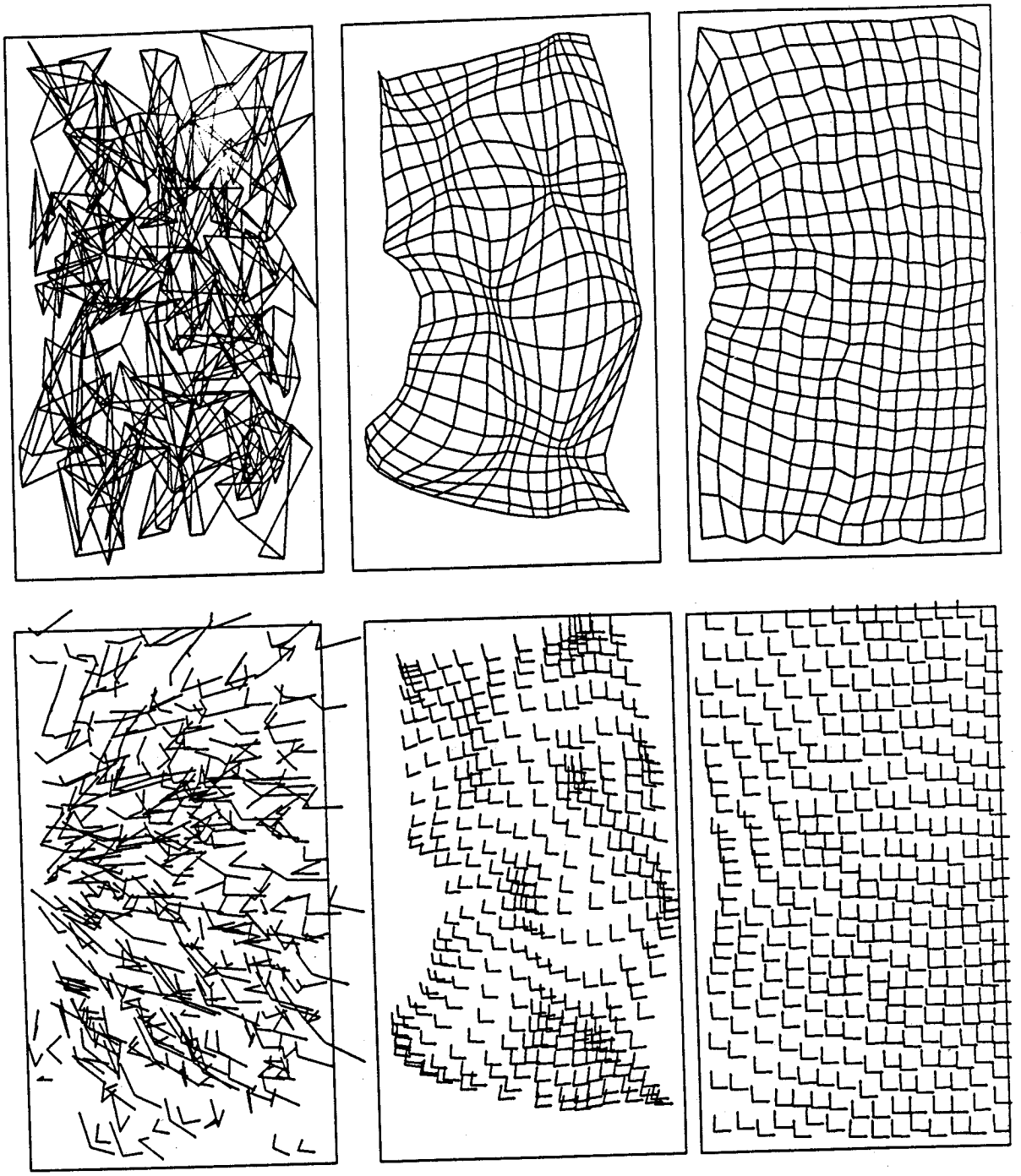


Bild 6

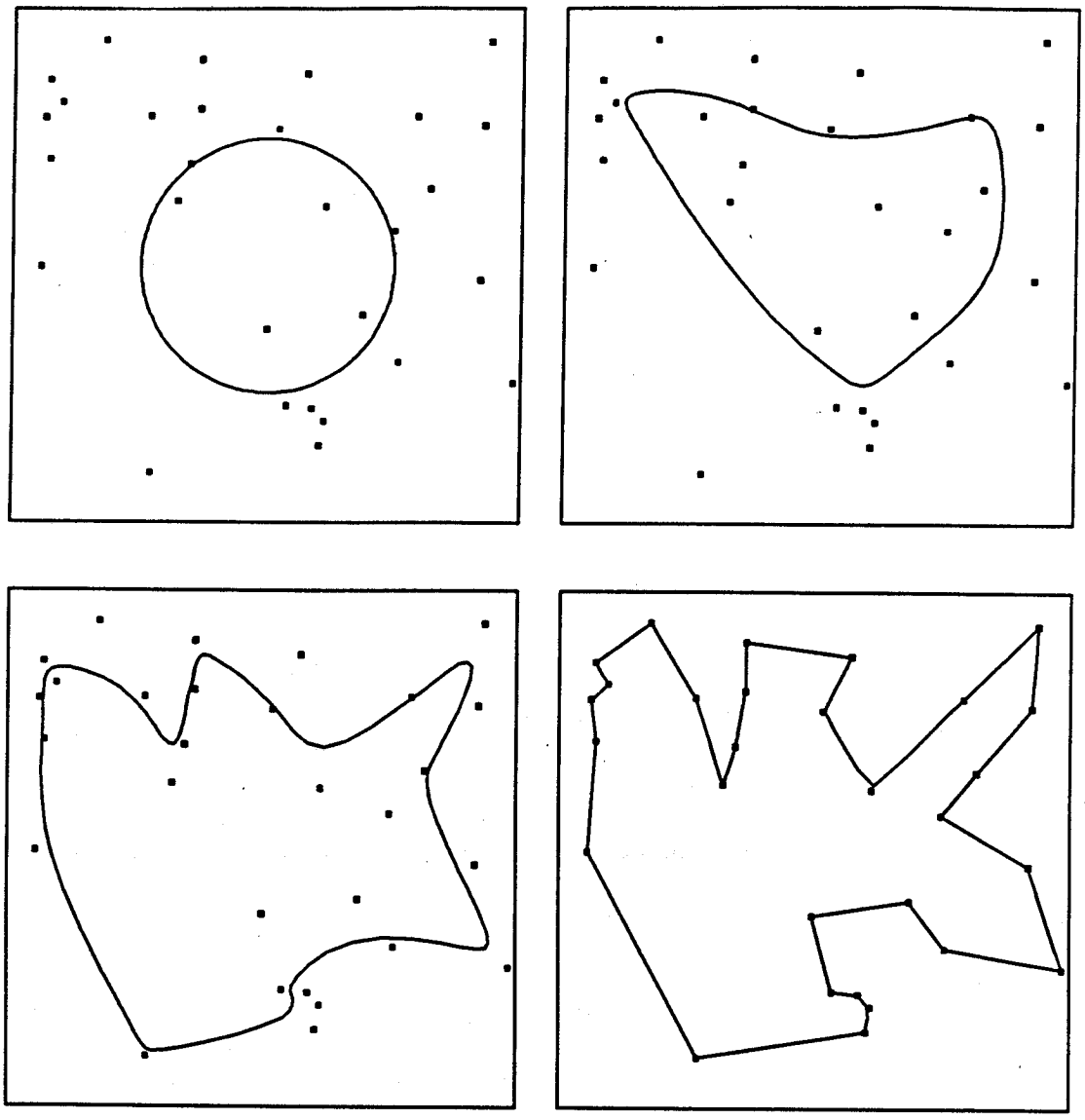


Bild 7