PS Einführung in die Bio-Informatik SS 1999
Growing Artificial Neural Networks

Christian Spielberger
spielberger@vpn.at

Rainer Trummer
rtrummer@cosy.sbg.ac.at

Ferdinand Schober
fschober@cosy.sbg.ac.at

LV-Leiter: Schwaiger Roland, Dipl.Ing. Mag.

Zusammenfassung

Diese Abhandlung beschreibt verschiedene Modelle für die Umsetzung der organischen Gehirnstruktur und deren Prozesse in sogenannte künstliche Neuronale Netze (ANN - Artificial Neural Networks). Motiv für die Entwicklung von ANN ist es Probleme zu lösen, die schwer mathematisch modellierbar sind. Die Idee dafür besteht darin, sich die Werkzeuge der Natur abzuschauen. Im speziellen soll darauf eingegangen werden, wie diese Netze ohne äußere Einwirkung wachsen können.

Projekt:
Ein Roboterarm soll durch ein Artificial Neural Network gesteuert werden. Ein Genetischer Algorithmus übernimmt die Anpassung und somit die Lernphase des ANN. Nach einer gewissen Zeit ist die Lernphase beendet, und die Trainingseinheit kann abgehängt werden, danach wird der Roboterarm nur mehr von einem ANN gesteuert.

Inhaltsverzeichnis

Kapitel 1

Einleitung [1]

Schon zum Ende des 19. Jh entstanden erste Ansätze für ANN. William James veröffentlichte eine Theorie, in der er annam, daß das Gehirn aus zufällig angeordneten Verbindungen besteht, und daß Gedankengänge und Körperbewegungen auf elektrischen Strömen basieren. Die Kernaussage, seiner Theorie ist, daß Verbindungen, die öfter benutzt werden gestärkt werden und somit weniger Widerstand für die Nervenströme bieten. Verbindungen, die nicht genutzt werden, verschwinden mit der Zeit. James gab somit den Anstoß für die Forschungen in Richtung der ANN (Artificial Neural Networks).

Zahlreiche Forscher entwickelten diese ersten Ideen weiter. Heute gibt es unterschiedliche Modelle für ANN, von denen manche schon in bestimmten Bereichen eingesetzt werden (Mustererkennung, Robotersteuerung, Spracherkennung, Börsenkurse, ...). Folgende Grundidee steckt in allen dieser Modelle. Das ANN soll mit möglichst wenigen, vom Entwickler vorgegebenen Daten und Algorithmen, ausgestattet werden. In der Trainingsphase wird das ANN darauf trainiert, immer bessere Ergebnisse zu liefern.

1.1 Biologische Grundlagen [2]

1.1.1 Neuronen

Es gibt viele Typen von Nervenzellen im Gehirn, die alle unterschiedliche Aufgaben erfüllen. Wir wollen davon nur zwei Arten genauer betrachten:

- die relativ großen, wie eine Pyramide aussehenden Pyramidenzellen, die sehr häufig im Gehirn vorkommen,
- und eine kleinere sternförmig aussehende Sorte (Stern- oder Gliazellen).

Nach neueren Erkenntnissen gibt es eine Arbeitsteilung zwischen beiden Zelltypen. Die Pyramidenzellen verarbeiten die elektrischen Impulse und die Sternzellen sichern dabei die Stoffwechselversorgung. Die Sternzellen sind die Mittler zwischen Blutsystem und Pyramidenzellen und bilden somit die Blut-Hirn-Schranke. Außerdem tragen sie durch die Vorgabe der Wachstumsrichtungen entscheidend zur Entwicklung des Gehirns bei.
Die Abbildung 1.1 zeigt eine typische Pyramidenzelle.
Die kleinen, astartigen, mit stachelartigen Stellen besetzen Auswüchse der Nervenzellen heißen Dendriten und leiten den elektrischen Strom, den sie erhalten, an den eigentlichen

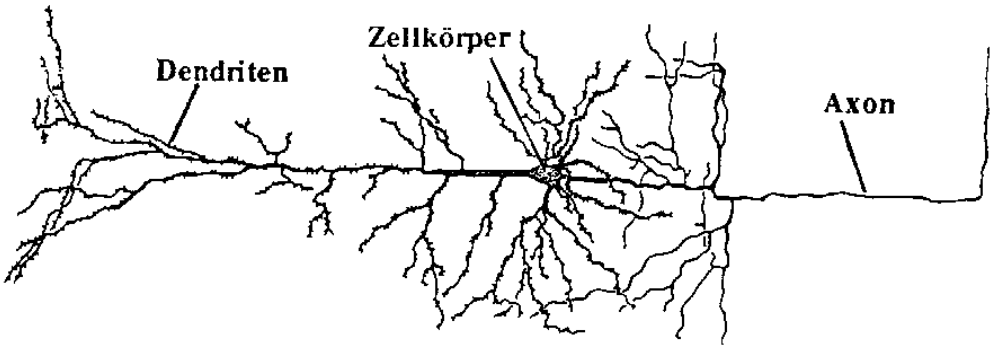

Abbildung 1.1: Pyramidenzelle

Zellkörper (Soma) weiter. Der Zellkern speichert die elektrische Ladung. Überschreitet die Ladung einen bestimmten Grenzwert, so entlädt sich der Kern mit einem kurzen, hohen Impuls. Dieser Impuls, den man Spike nennt, pflanzt sich vom Zellkörper auf dem Axon bis in die entferntesten Verzweigungen fort. Axone und benachbarte Dendriten anderer Neuronen können aufeinander zuwachsen und elektro-chemische Kontaktstellen (Synapsen) bilden. Der Informationsfluß der Nervenzellen geht normalerweise vom Zellkern über das Axon durch die Synapsen auf die Dendriten zum Zellkern des anderen Neurons. Bidirektionale, rein elektrische Synapsen sind zwar bekannt, die chemischen Synapsen sind aber die Regel.

1.1.2 Synapsen

Die Verbindung zwischen dem Axon eines Neurons und dem Dendrit eines anderen Neurons bzw. einem Muskel wird Synapse genannt. An dieser Synapse sind die beiden Zellen durch einen kleine Spalt (ca. 200 nm) getrennt. Das sendende Neuron nennt man in diesem Zusammenhang presynaptisches, das Empfangende nennt man postsynaptisches Neuron. Diese Synapsen können auch direkt am Zellkörper auftreten, wie z.B. bei Muskelzellen. Die Signale werden durch diese Synapse chemisch übertragen, wobei die elektrischen Impulse eine kleine Menge von einer speziellen Substanz freisetzen, die sogenannten Neurotransmitter. Diese überqueren den Spalt der Synapse und lösen am postsynaptischen Neuron eine Entladung aus. Stark benutzte Synapsen wachsen und werden verstärkt, selten benutzte Synapsen werden mit der Zeit immer schwächer, dies stellt einen wichtigen Beitrag zum Lernprozeß dar.

1.1.3 Kodierung der Information

Der oben beschriebene Vorgang kann mit Mikroelektroden elektrisch gemessen werden. Eine Stimulation des Zellkörpers mittels Reizstromes bewirkt am Axon des Neurons aber nur eine Potentialerhöhung. Überschreitet der Reiz geringfügig einen Schwellwert, so folgen periodische Entladungen (Aktionspotentiale oder Spikes), deren Größe unabhängig von der Reizstromstärke ist. Obwohl das resultierende Axonsignal damit von Natur aus binär ist, läßt sich auch nicht-binäre Information darin kodieren. Eine leichte Absenkung der Reizstromstärke bewirkt eine Verkleinerung der Impulsfrequenz der Spikes bzw. eine

Vergrößerung der Zeitabstände zwischen den Spikes. Nimmt man für jeden Spike eine Einheitsladung an, die vom Axon zum Dendriten transportiert wird, so ergibt sich die mittlere Stromstärke am Dendriten, die der Reizstromstärke entspricht. (siehe Abb. 1.2)

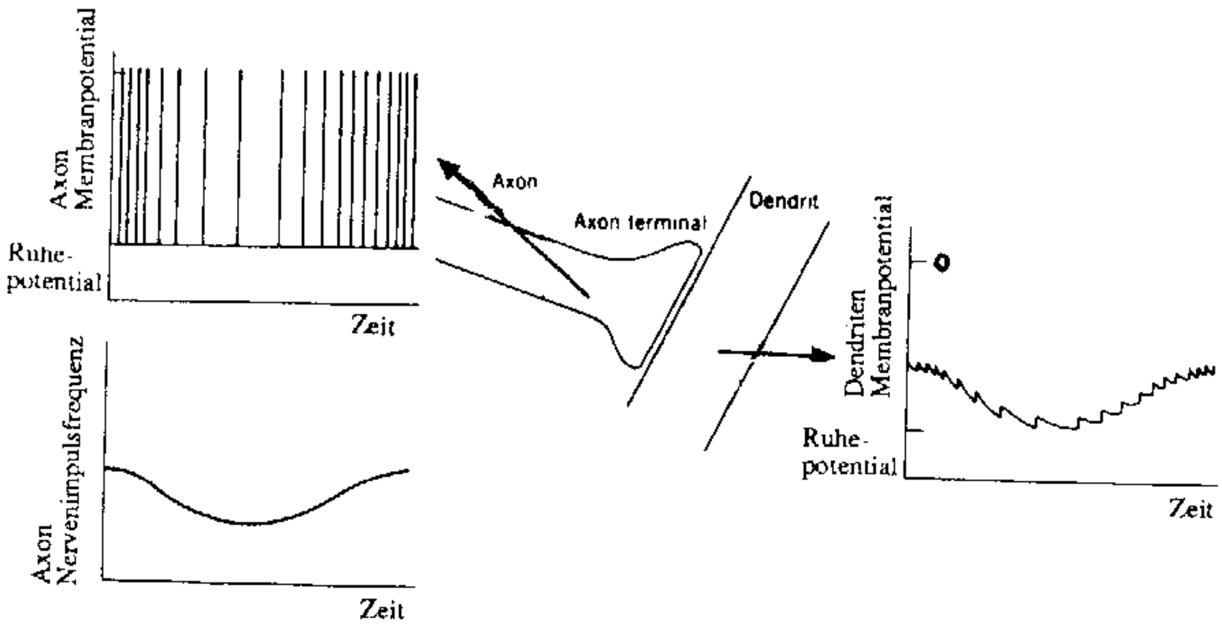

Abbildung 1.2: Frequenzmodulierung und Dekodierung

Bei hohen Reizstärken geht die Zellaktivität in eine Sättigung über, die durch eine minimale (ca. 1 ms lange), nötige Regenerationszeit nach einem Aktionspotential (Spike) bestimmt wird. Damit ist die maximale Taktfrequenz der Neuronen auf einen relativ niedrigen Wert von ca. 1 KHz begrenzt.

1.1.4 Biologisches Wachstum des menschlichen Gehirns

Die Komplexität des menschlichen Gehirns kann man durch die große Anzahl der Neuronen und deren Verbindungen untereinander erklären. Es gibt im Gehirn verschiedenste Arten von Neuronen, die alle jedoch nach dem gleichen Grundprinzip arbeiten. Es wird angenommen, daß jedes Neuron im menschlichen Gehirn im Durchschnitt Signale von 10.000 anderen Neuronen erhält und seine eigenen Signale an bis zu 100 Neuronen weitersendet. Es gibt jedoch auch gewisse Neuronen, sog. Purkinje Zellen, die bis zu 80.000 Signale empfangen können. Eine einzige sogenannte Granule Zelle kann mit 50 oder mehr solcher Purkinje Zellen verbunden sein.

Die gesamte Anzahl von Neuronen im menschlichen Gehirn ist immens und kann nur geschätzt werden. Messungen in kleineren Gehirnregionen ergaben eine Dichte von ca. 150.000 bis 200.000 Neuronen pro mm^2, woraus auf eine gesammte Neuronenanzahl von ca. $3 * 10^{10}$ geschlossen werden kann. Dies ergibt eine Höchstanzahl von 10^{15} synaptische Verbindungen, die sich in den ersten paar Monaten nach der Geburt entwickeln. Die Information über diese Verbindungen kann nicht genetisch vererbt werden, da die Größe der

DNA für diese Informationsmenge nicht ausreicht. Es wird daher stark vermutet, daß nur gewisse Grundstrukturen genetisch vererbt werden. Die weiteren Verbindung werden demnach höchstwahrscheinlich vom Gehirn selbst generiert. Wenn gewisse Gehirnstrukturen in diesem Stadium nicht entwickelt wurden, dann können diese auch zu einem späteren Zeitpunkt, obwohl sie voll funktionsfähig wären, nicht mehr gebildet werden. Z.B: Die Unterdrückung gewisser Sinneswahrnehmungen kann zu einer irreperablen Fehlentwicklung führen. (siehe Abb. 1.3)

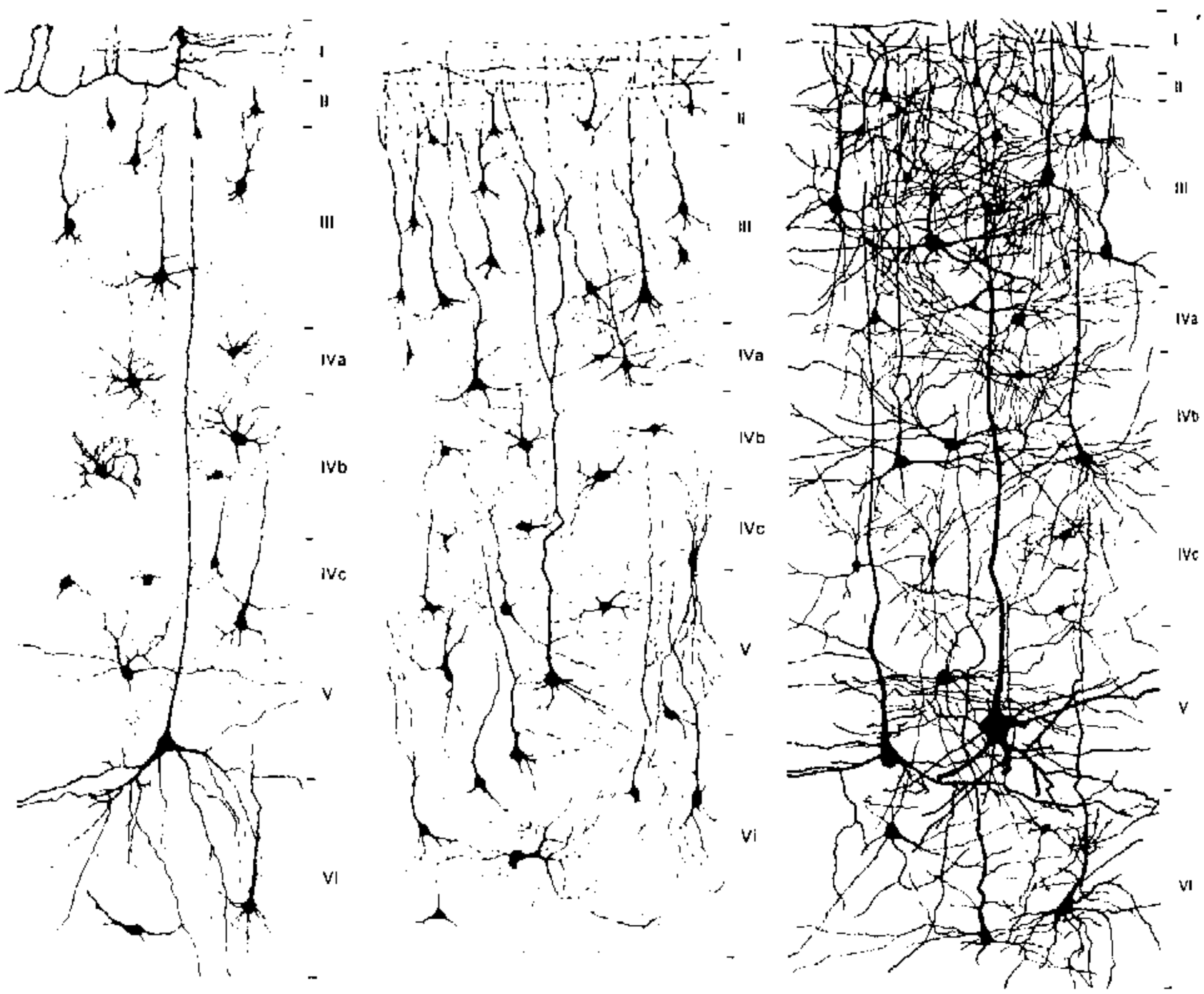

Abbildung 1.3: Fig.1.9. Entwicklung des Nervensystems in der menschlichen Hirnrinde, vom (links) Neugeborenen, (mitte) drei Monate alten und (rechts) zwei jahre alten Säugling

Der Mensch besitzt, im Gegensatz zu anderen Lebewesen, ein zu hohes Gehirn/Körpermasse Verhältnis, das nicht mit der erhöhten Signalmenge des menschlichen Körpers übereinstimmt. Es wird daher angenommen, daß gewisse Regionen des Gehirns, die offensichtlich keinerlei Funktionen erfüllen, einen wichtigen Teil zur allgemeinen Organisation im Gehirn beitragen. Dadurch wird klar, daß es trotz intensivster Forschungen auf dem Gebiet der Neurologie noch ein weiter Weg bis zum vollständigen Verständnis des menschlichen Gehirns ist.

1.2 Aufbau eines ANN [1]

- Das ANN kann als gerichteter Graph dargestellt werden.
- Die Knoten des Graphen entsprechen den Neuronen.
- Die Kanten des Graphen entsprechen den Verbindungen der Neuronen untereinander.

1.2.1 Knoten - Neuronen

Die Funktion eines Neurons kann sich dynamisch ändern. Z.B. ein einfaches Neuron nach Warren McCulloch und Walter Pitts (1943) besteht aus einer Additionsoperation und einem Schwellwert-Komparator. (siehe Abb. 1.4)

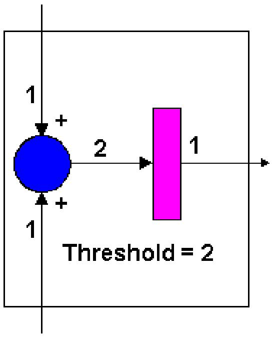

Abbildung 1.4: Schwellwert Logik, UND Operation

Zwei Eingangssignale werden addiert. Erreicht das Ergebnis einen Schwellwert, so wird an den Ausgang eine 1 weitergegeben. Mit einem Schwellwert von 2 hat man eine Und-Funktion mit einer 1 eine Oder-Funktion. Man nennt diese Art von Logik threshold logic. Später wurden noch komplexere Neuronen-Modelle vorgeschlagen mit mehr Eingängen und mehr Operationsgliedern. Zusätzlich wird im Neuron eine Ausgangsfunktion definiert, die das Signal verändert, bevor es auf den Ausgang gelegt wird.

1.2.2 Schichten

Ein ANN wird häufig in mehrere Schichten unterteilt.

- Zunächst gibt es auf der Seite der Sensorik die Eingangsschicht (Input-Layer). Für jeden der Eingangsparameter existiert ein Knoten, dh. ein Neuron mit einer Eingangsleitung, die das Eingangssignal aufnimmt.
- Dann werden für jeden Ausgangsparameter Neuronen festgelegt, die die Ausgangsschicht (Output-Layer) bilden.
- Zwischen diesen beiden Schichten, werden dann eine oder mehrere versteckte Schichten (Hidden-Layers) definiert, die die ´Intelligenz´ des Netzes ausmachen.

1.2.3 Verbindungen

Nun werden zwischen den Neuronen, unter Hinblick auf zuvor definierte Layer, Verbindungen hergestellt. Ausgänge von Neuronen werden mit Eingängen anderer Neuronen verbunden. Die Eingangssignale werden mit den Neuronen des Input-Layers verbunden. Jeder Ausgang dieser Eingangsneuronen wird mit jedem Neuron im ersten Hidden-Layer verbunden. Für die Ausgänge im Hiddenlayer gilt das Gleiche. Zusätzlich können in den Hiddenschichten Rückkopplungen eingefügt werden. Das so entstehende Netzwerk ist sehr unüberschaubar und sein Verhalten schwer vorher zu berechnen.

Die Verbindungen können nun mit einer Gewichtung (weights) versehen werden. Das ist ein Multiplikator der das Signal auf dieser Verbindung abschwächt bzw. verstärkt. Durch diese Methode werden die Verbindungen bewertet. Das Verhalten des gesamten Netzwerks wird durch die Gewichtungen beeinflußt und somit das Trainieren ermöglicht.

1.3 Lernmethoden für neuronale Netze [3]

1.3.1 Überwachtes Lernen

Unter Überwachtem Lernen (**Supervised Learning**) versteht man die Anpassung der Gewichtungen eines Neuronalen Netzes bezüglich eines Beurteilungskriteriums, meist in der Form einer sogenannten Objektfunktion (**Objective Function**). Die Objektfunktion, Zielfunktion je nach Typ auch Fehlerfunktion oder Energiefunktion genannt, bewertet die Ausgabe eines Netzes. Niedrigere Werte der Objektfunktion stehen für bessere Qualität, sodaß die Aufgabe darin besteht, einen Satz von Parametern zu finden, für den die Objektfunktion minimal wird. Wobei für Parameter hier die Gewichtungen der Verbindungen gemeint sind.

Gradientenabstieg

Der Gradientenabstieg (**Gradient Descent**) stellt ein einfaches und für neuronale Netze meist verwendetes Verfahren zum Überwachten Lernen dar. Die Gewichte im Lernschritt n + 1 werden jeweils aus den Gewichten des Lernschrittes n berechnet. Dazu werden dem Netz nacheinander alle zu lernenden Eingabemuster eingegeben und die jeweiligen Ausgaben entsprechend der Objektfunktion bewertet. Zu einem Satz von Gewichten W ergibt sich dann im Lernschritt n ein bestimmter Wert der Objekfunktion $E(W(n))$. Da der Gradient von $E(W(n))$ die Richtung des stärksten Anstiegs der Objektfunktion angibt, verändert man die Gewichte in kleinen Schritten in entgegengesetzter Richtung. Die Größe der Schritte wird durch die Lernrate ϵ angegeben. Jeder Durchlauf durch den Satz der Gewichte W, der zu lernenden Muster, bildet einen Lernzyklus. Der Lernvorgang wird beendet, wenn sich der Wert der Objektfunktion nicht mehr deutlich verringert oder wenn die Aufgabenstellung mit der geforderten Genauigkeit gelöst wird. Jedes der möglicherweise zahlreichen Minima der Objektfunktion hat einen Einzugsbereich, in dem der Gradientenabstieg auf die entsprechenden Werte für die Gewichte zustrebt. Oft existiert jedoch nur ein globales Minimum, das sich durch die beste Ausgabequalität auszeichnet. Der Gradientenabstieg kann sich in einem lokalen Minimum verfangen, das einen höheren Wert der Objektfunktion aufweist, als das gesuchte globale Minimum oder Optimum. Einfachste Abhilfe ist oftmals ein wiederholter Neustart des Lernens mit veränderten Anfangsgewichten. Während des Gradientenabstiegs können, weil man die Schrittweite nicht beliebig fein wählen kann, zum Teil große Sprünge in der Objektfunktion auftreten. (siehe Abb. 1.5)

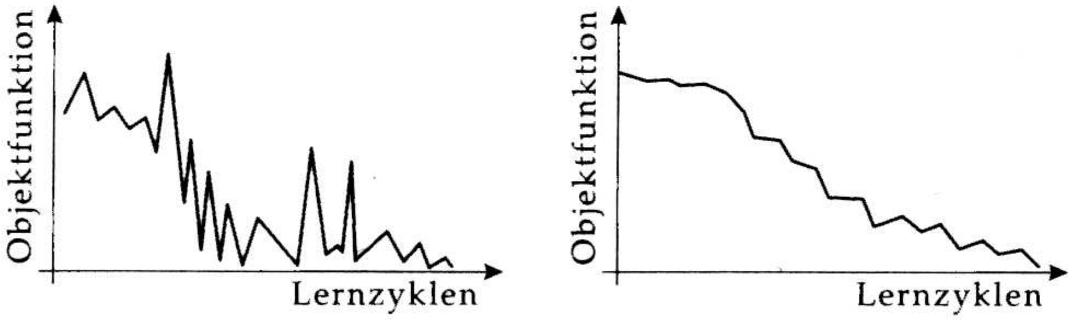

Abbildung 1.5: Gradientenabstieg

Die Lernrate ϵ (Learning Rate) bestimmt den Verlauf des Gradientenabstiegs. Eine kleinere Lernrate führt zu kleineren Sprüngen in der Objektfunktion und eventuell auch exakterer Adaption. Eine größere Lernrate kann den Lernvorgang beschleunigen und ein notwendiges Entkommen aus lokalen Minima erleichtern. Eine geeignete Lernrate ist von Fall zu Fall durch probieren zu ermitteln. (siehe Abb. 1.6)

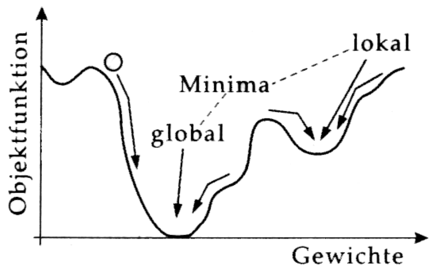

Abbildung 1.6: Objektfunktion während des Lernens, links große, rechts kleine Lernrate ϵ

Inkrementelles Lernen

Die Zuordnunge eines Eingangstuppels zu einem Ausgangstuppel nennt man Muster. Unter Inkrementellem Lernen **Incremental Learning** versteht man nun die Anpassung der Gewichte jeweils nach dem Auswerten eines einzelnen Musters. Dazu wird für jedes Lernmuster einmal der Gradient einer entsprechenden Objektfunktion berechnet. Es wird also kein echter Gradientenabstieg durchgeführt, bei dem alle zu lernenden Muster auf einmal erfaßt würden (**Batch Learning**). Bei großen Datensätzen, die viele Wiederholungen beinhalten, kann Inkrementelles Lernen eine unter Umständen erhebliche Verminderung des Aufwands für das Lernen bewirken. Es vermindert sich auch die Wahrscheinlichkeit, daß sich der Lernvorgang in lokalen Minima verfängt, vorallem wenn nach jedem Lernzyklus die Reihenfolge der Muster im Lernsatz (zufällig) durchmischt wird.

1.3.2 Unüberwachtes Lernen

Beim Unüberwachten Lernen (**Unsupervised Learning**) stehen dem lernenden System aus den Daten keine Angaben über ein gewünschtes Ausgabeverhalten zur Verfügung. Eine Bewertung (Beurteilung) von Ausgaben während des Lernens findet nicht statt! Das Gewünschte Verhalten des Netzes muß deshalb durch den Lernalgorithmus festgelegt sein. Eine wesentliche Anwendung solcher Systeme ist die Entwicklung von Datentransformationen verschiedenster Art. Die Redundanz (mehrfaches Auftreten von gleichen Daten) spielt im unüberwachten Lernprozeß eine entscheidende Rolle, da sie den einzigen in den Daten enthaltenen Anhaltspunkt zur Lösung einer Aufgabe darstellt.

In manchen Anwendungen wird der Einsatz des neuronalen Netzes bereits mit dem Lernen abgeschlossen, da der dadurch gewonnene Gewichtssatz selbst die Lösung des Problems darstellt. Selbstorganisierende Systeme haben in der Regel die Aufgabe, ein Zuordnungsschema zweier geometrisch verschiedener Räume (Ein- und Ausgaberaum) zu finden, das soweit wie möglich Nachbarschaftsbeziehungen erhält, also topologieerhaltend ist.

Das wohl bekannteste selbstorganisierende neuronale System ist das sogenannte Kohonen Modell. Jede Komponente eines Eingabevektors (jeder Eingabeknoten) wirkt über Verbindungen mit veränderlichen Gewichten auf eine Anzahl aktiver Knoten, denen ein geometrischer Zusammenhang zugeordnet ist. Typischerweise reagieren im trainierten Netz benachbarte Koten auf ähnliche Eingabewerte, wobei sie jeder Knoten auf einen bestimmten Bereich des Eingaberaums spezialisiert (Topologieerhaltung). Die Auflösung (Zahl der zuständigen Knoten) steigt mit der Wichtigkeit (Anzahl der Stimulationen) des entsprechenden Areals im Eingaberaum. Die Gewichte werden mit Zufallswerten initialisiert. Für jeden Lernschritt unter einem Eingabevektor wird diejenige Knotenposition ermittelt, für die der Wert minimal ist. Dies stellt eine Vereinfachung der Methode dar, nach Berechnung herkömmlicher Knotensignale jeweils den Knoten mit maximalen Signal zu wählen. Die Verbindungsgewichte für alle Knoten aus der Umgebung eines bestimmten Knotens (hier tritt der geometrische Zusammenhang in Erscheinung) werden in Richtung auf die aktuelle Eingabe verschoben. Der Einfluß der Verschiebung hängt von der Entfernung zwischen zwei benachbarten Knoten und deren zugeordneten Positionen ab. Die sogenannte Identitätsfunktion (z.B. Gauß-Glocken-Funktion) muß, darauf bezogen, mit steigendem Abstand abnehmen. Die Lernrate ϵ gibt wieder die Größe der Lernschritte an. Im Verlauf des Lernens werden dem Netz sehr viele verschiedene Eingabevektoren vorgesetzt, die meist zufällig, aber in der Regel nicht gleichverteilt, gewählt werden. Die Dimensionen des Ein- und Ausgaberaums können unterschiedlich sein.

Die Breite des beeinflußten Bereichs wird durch die Breite der Intensitätsfunktion bestimmt und durch die Größe α angegeben. Um eine brauchbare Grobstruktur der Zuordnung zu erhalten, muß das Lernverfahren zu Beginn mit großen α betrieben werden, das dann bei feiner werdender Zuordnung (Auflösung) verringert wird. Ebenso wird die Lernrate ϵ, die am Anfang groß gewählt wird, um rasch eine Struktur zu erzeugen, im Verlauf des Lernens verkleinert. Die Wahl entsprechender Verläufe ist stets mit einer gewissen Willkür verbunden. Möglich sind zum Beispiel lineare, exponentielle oder hyperbolische Verläufe. Die beste Wahl muß im Einzelfall durch Probieren ermittelt werden. (siehe Abb. 1.7)

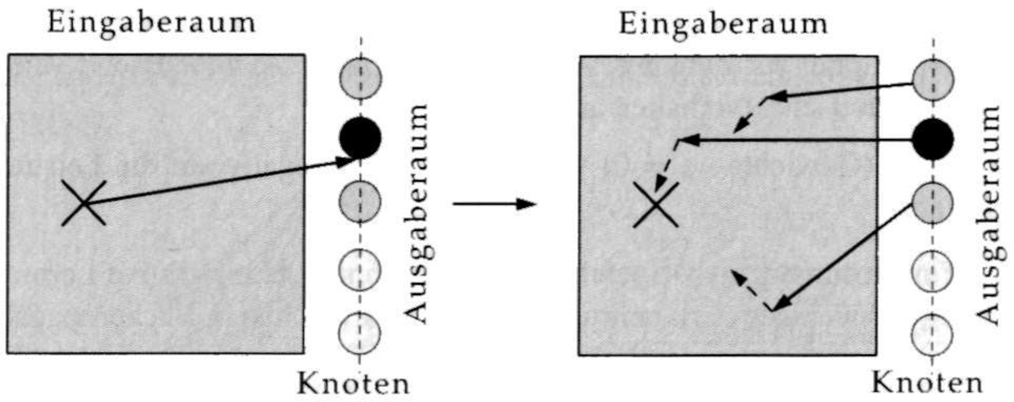

Abbildung 1.7: Lernen im Kohonenmodell; links: die Wahl der Knotenposition mit maximaler Anregung, rechts: die Adaption der Gewichte entsprechend der Knotenpositionen

Kapitel 2

Berichte

2.1 Formation of Neural Strucutes [4]

In dieser Abhandlung wird ein neuer Ansatz für das Wachstum von neuralen Strukturen vorgeschlagen.

Es wurde versucht, mit Hilfe von Diffusionsfeld-Modellen (siehe unten) zusammen mit genetischer Vererbung, neurale Systeme möglichst naturnah nachzubauen. Speziell wurde versucht einen virtuellen Roboter durch eine virtuelle Umgebung zu steuern. Diese Steuerung sollte das ANN übernehmen.

2.1.1 Diffusion field modeling

Das Diffusionsfeld Modell (DFM - diffusion field modeling) wurde bis dahin hauptsächlich für Bakterienmodellierung verwendet. Es beruht auf dem Grundsatz, daß eine simulierte Zelle in ein zwei- oder mehrdimensionales Gitter eingetragen wird. Die Wahrscheinlichkeit der Ausbreitung der Zellen in Nachbarfelder wird mit der Variable η angegeben, wobei eine hohe Wahrscheinlichkeit durch einen hohen Wert repräsentiert wird.

Mit Hilfe dieses Modells können nicht nur Bakterien, sondern, wie in diesem Fall, auch Verzweigungsstrukturen für Verbindungen in 2D, sowie 3D-Räumen realisiert werden.

2.1.2 Wachstum der Verbindungen

Mit Hilfe des DFM werden die Verbindungen gebildet. Jede Verbindungseinheit wird über eine andere Verbindung mit der sogenannten Diffusionsmaterie (diffusion matter) versorgt, die nur dazu dient, um festzustellen ob das Wachstum möglich und gewollt ist.

Eine Verbindung die viel Diffusionsmaterie erhält wächst, eine Verbindung die wenig davon erhält wird im Wachstum unterdrückt. Verbindungen, die keinerlei Diffusionsmaterie erhalten, sterben ab. Dadurch wird Platz für neue Verbindungen frei.

2.1.3 Neuronenmodell

Ein Neuron besteht hier aus einem Zellkern (Nucleus), aus Eingangsleitungen (Dendriten) und Ausgangsleitungen (Axone). Der Zellkern wendet eine von zwei Funktionen (S, S´) auf das Eingangssignal an. S läßt das Signal eins zu eins durch und S´ invertiert niedrige Signal zu hohe und umgekehrt hohe Signale zu niedrige. Welche von den beiden Funktionen zum Zug kommt, ist in den Erbinformationen festgelegt.

Bei den Dendriten gibt es zwei verschiedene Arten von Ästen. Zum Einen die excitory, zum Anderen die inhibitory Typen. Letztere kehren das Vorzeichen des Signals um.

2.1.4 Wachstum der Neuronen

In der Natur wachsen nicht nur die Verbindungen zwischen den Neuronen, sondern auch die Anzahl der Neuronen. In diesem Modell ist die Anzahl der Neuronen konstant bei fünf.

2.1.5 Signalwirkung

Es gibt mehrere Signalquellen, die wie Sender wirken. Die Verbindungen (Dendrite, Axone) wirken wie Empfangsantennen. Umso näher eine Verbindung zu einer Signalquelle ist, desto stärker ist das Signal. Die Signalquellen können sowohl die Sensoren des virtuellen Roboters, als auch deren Motoren sein. Bei den Verbindungen ist festgelegt, daß Dendrite zu den Sensoren wachsen um das Eingangssignal für den Zellkern zu liefern. Die Axone hingegen wachsen zu den Motoren, um das Antriebssignal zu übertragen.

2.1.6 Lernfähigkeit

Die Lernfähigkeit hängt teilweise von der Anzahl und der Länge der Verbindungen ab, aber es wird auch zusätzlich sogenanntes ´Verstärkendes Lernen´ (reinforcement learning) benutzt. Dieses beeinflußt sowohl Verbindungswachstum, als auch die Überlebensfähigkeit der Verbindung.

2.1.7 Sensormodell

Die Lichtquellen im virtuellen Raum beeinflussen die Sensoren des Roboters, woraufhin der Sender des Sensors ein Signal zu den Neuronen sendet. Die Stärke hängt von der Entfernung zur Lichtquelle ab, je weiter entfernt, desto schwächer ist das Signal, daß vom Sensor abgegeben wird. Die zweite Art von Sensoren sind die Berührungssensoren, die die Nähe des Roboters zu Hindernissen messen.

2.1.8 Motormodell

Die Motoren erzeugen eine Bewegung, deren Geschwindigkeit vom Signal abhängt, daß sie erhalten. Mit Hilfe unterschiedlicher Geschwindigkeiten der Motoren kann der Roboter gesteuert werden.

2.1.9 Genetischer Code

Hier wird, im Gegensatz zu anderen Modellen, wenig genetische Vererbungen benutzt, sondern das Wachstum als Funktion der Umwelt angesehen. Es gibt eine klare Trennung zwischen den ´Naturgesetzen´ und der genetischen Information, dadurch zeigt sich, ob ein ANN fähig ist, sich an seine Umwelt während seines Wachstums anzupassen, also ohne genetische Veränderung.
In den fünf verwendeten Neuronen gibt es jeweils 8 Bit Gencode, die mit 1 Bit Fehlerwahrscheinlichkeit pro Generation übertragen werden. Im Gencode ist festgelegt:

- Konstante η, die das Wachstum der Dendriten und Axone beeinflußt.
- Art der Dendriten (Excitory, Inhibitory)
- Type des Zellkerns (Transparent C, Inversion C´)
- und ein Parameter, der das Lernen beeinflußt.

2.1.10 Durchführung der Simulation

Die Simulation wurde mit einem virtuellen Roboter mit 5 Neuronen in einem 16x26x12 Diffusionsgitter durchgeführt. Die simulierte Außenwelt bestand aus vier Lichtquellen und mehreren Wänden, die das Licht blockierten. Das Ziel bestand darin, daß die ANN den Roboter von einem Licht zum anderen, ohne Berührung von Wänden, steuern kann. Es wurde das Erreichen eines Lichts mit 100 Überlebenspunkten belohnt und die Kollisionen mit 10 Punkten Abzug bestraft. Es wurde festgelegt, daß aus dem Roboter bis zu 12 neue Generationen entstehen können.

2.1.11 Ergebnisse

Nach Abschluß der Simulation wurde festgestellt, daß 5 Neuronen mehr als ausreichend für diese Aufgabenstellung sind. Die Verbindungen können ohne Probleme mit dem Diffusionsmodell realisiert werden. Die genetische Information kann mit dem verwendeten Modell reduziert werden, indem die Umgebung Teile der Kontrolle übernimmmt. Es wurde nur ein einziger Roboter in der virtuellen Umgebung getestet, aber in Zukunft sollen mehrere Roboter realisiert werden, die untereinander Informationen austauschen.

Kapitel 3

Genetische Algorithmen [5]

Als Trainingsalgorithmus für ANN werden zum Beispiel Genetische Algorithmen eingesetzt.

3.1 Einleitung

GA sind Optimierungs-Technicken basierend auf dem Konzept der natürlichen Selektion und Genetik. In diesem Zusammenhang, werden die Variablen repräsentiert als Gene eines Chromosoms. Folgende Faktoren bewirken, daß Chromosomen mit besserer Fitneß gefunden werden:

- natürliche Selektion
- genetische Operatoren
- Mutation
- Rekombination

Die natürliche Selektion garantiert, daß sich Chromosomen mit der besten Fitneß in zukünftigen Populationen durchsetzen. Der Rekombinations-Operator wird verwendet um Gene von zwei benachbarten Chromosomen zu kombinieren und daraus zwei neue Chromosomen (Kinder) zu formen, die eine hohe Wahrscheinlichkeit aufweisen, eine bessere Fitneß zu haben. Mutation bietet die Möglichkeit neue Bereiche der Reaktionsoberfläche zu erforschen. Das bieten eine Generationsverbesserung in der Fitneß der Chromosomen und nach vielen Generationen entstehen Chromosomen mit optimalen Variablenwerten.
Vorteile für die Verwendung von GA

- Man benötigt kein Wissen über die Form der Reaktionsoberfläche.
- Unterbrechungen der Reaktionsoberfläche habe kleine Auswirkungen auf die Performance der Optimierung.
- Sie vermeiden das Stehenbleiben bei lokalen Optimas.
- Sie funktionieren sehr gut für umfangreiche Optimierungsprobleme.
- Sie können in vielen Bereichen für Optimierung eingesetzt werden.

Nachteile für die Verwendung von GA

- Es gibt Schwierigkeiten beim exakten Bestimmen von globalen Maximas.
- Man hat einen hoher Aufwand beim Berechnen der Fitneßfunktion.
- Die Konfiguration ist kompliziert.

3.2 Funktionsweise

Standard GAs beinhalten vier wichtige Schritte:
1. Initialisierung (Initialize) - Zu Beginn wird die Population der Chromosomen entweder zufällig erstellt oder durch Veränderung eines Eingabe- Chromosoms. Auf jeden Fall muß die Inital-Population einen großen Bereich der Variablenwerte abdecken. Folglich kann auch explizites Wissen über das System in die Inital-Population einfließen.

2. Bewertung (Evaluate) - Im zweiten Schritt wird die Fitneß berechnet. Dazu gibt es eine Fitneßfunktion, die vom jeweiligen System abhängt. Das Ziel der Fitneßfunktion ist die Leistung der Chromosomen numerisch zu bewerten. Für Optimierungsaufgaben in der realen Welt, ist die Wahl der Fitneßfunktion der wichtigste Schritt.

3. Auswertung (Exploit) - Der dritte Schritt ist die Auswertung oder die natürliche Selektion. Dabei werden die Chromosomen mit den größten Fitneßwerten einmal oder mehrmals in einem Paarungsraum (Mating-Pool), teils zufällig gesteuert, abgelegt. Chromosomen mit niedrigen Fitneßwerten werden aus der Population entfernt. Es gibt verschiedene Methoden für die natürliche Selektion. Eine der meist verbreiteten Methoden ist die binäre Turnier-Paarungs-Teilmengen- Selektion. In dieser Paarungsselektionsmethode versucht jedes der Chromosomen in der Population einen Platz im Paarungsraum zu bekommen. Zwei Chromosomen werden zufällig aus der Population selektiert und in der Paarungsteilmenge abgelegt. Dabei haben Chromosomen mit hoher Fitneß eine höhere Wahrscheinlichkeit gewählt zu werden. Die Selektion wird solange durchgeführt, bis der Paarungsraum voll ist. Eine Charakteristik dieses Schemas ist, daß das schlechteste Chromosom nie selektiert wird.

4. Untersuchung (Explore) - Der vierte Schritt besteht aus den Rekombinations- und Mutationsoperatoren. Zwei Chromosomen (Eltern) in der Paarungsteilmenge werden zufällig für die Paarung ausgewählt. Die Wahrscheinlichkeit, daß diese Chromosomen rekombiniert (gepaart) werden wird gewöhnlich sehr hoch gesetzt (z.B.: 0,95). Wenn sie sich paaren, wird ein Rekombinationsoperator angewendet um Gene zwischen den beiden Eltern auszutauschen und zwei Kinder zu erzeugen. Wenn sie sich nicht paaren, werden sie unverändert in die nächste Generation übergeben. Die zwei meist gebräuchlichen Rekombinationsoperatoren sind die Einpunkt- und die Zweipunkt- Cross-over Methode. In der Einpunkt-Methode wird ein Cross-over-Punkt entlang des Chromosoms selektiert. Dann werden die Gene auf einer Seite dieses Punktes zwischen den Eltern ausgetauscht. In der Zweipunkt-Methode werden zwei Cross- over-Punkte selektiert, und die Gene die sich dazwischen befinden ausgetauscht. Die Kinder ersetzen dann die Eltern in der nächsten Generation. Ein dritter Rekombinationsoperator ist die Uniform-Cross-over Methode. In dieser Cross-over- Methode wird die Rekombination auf die individuellen Gene im Chromosom angewendet. Wenn Cross-over ausgeführt wird, werden die Gene zwischen den Eltern ausgetauscht, ansonsten werden sie unverändert gelassen. Diese Methode hat eine hohe Wahrscheinlichkeit Kinder zu erzeugen, die von ihren Eltern sehr unterschiedlich sind. Deshalb wird die Wahrscheinlichkeit für die Paarung sehr niedrig gesetzt (dh. 0,1). Die Wahrscheinlichkeit, daß eine Mutation stattfindet ist eine andere Option und wird gewöhnlich sehr niedrig gewählt (meist $1/l$ wobei l die Länge des Chromosoms angibt), damit gute Chromosomen nicht zerstört werden. Eine Mutation verändert einfach den Wert für ein bestimmtes Gen.

Nach der Untersuchung ist die Population voll mit neukreierten Chromosomen (Kindern). Der Prozeß beginnt bei Punkt 2 und wiederholt sich solange bis eine bestimmte Anzahl von Generationen erreicht ist.

3.3 Genetische Darstellung

Es gibt grundsätzlich drei Klassen von Optimierungsproblemen. Zwei davon, die sogenannten SUB- und NUM-Probleme treten oft in der analytischen Chemie auf. Die SUB-Probleme beinhalten die Auswahl der Untergruppen (subsets) oder der Merkmale (features). Die öfters auftretenden NUM-Probleme dagegen beinhalten Realzahlen-Variablen, Anwendungen wie nichtlineare Modellbildung und Positionierung von Gewichtsvektoren für nichtparametrische leardiskriminante Analysis. Die beste Wahl der Darstellung hängt vom Problem selbst ab, wofür es die zwei hauptsächlich verwendeten Arten textbfBinär- und Realzahlen-Codierung gibt (binary- and real coding). Diese unterscheiden sich lediglich dadurch, wie die Rekombinations- und Mutationsoperationen angewendet werden.

Der Großteil der Forschung auf dem Gebiet der GAs wurde mit binärer Codierung durchgeführt und daher am meisten verwendet. Bei der Binärcodierung besteht ein einzelnes Chromosom aus einem Bitstring, wobei jedes Bit ein Gen darstellt und mit 0 oder 1 kodiert wird. Beim SUB-Problem codiert man mit jedem Gen ein bestimmtes Merkmal. Für Optimierungsprobleme mit Realzahlen werden die Fließkommavariablen (floats) in einen Bitstring fixer Länge umgewandelt. Durch diese Darstellung kann eine Mutation oder ein Crossover in einer Fließkommadarstellung des Bitstrings auftreten, wodurch eventuell sinnlose Ergebnisse produziert werden. Deshalb werden die Darstellungen von Fließkommavariablen oft als einzelnes Gen behandelt und können nicht aufgetrennt werden. Die Mutation wird dann durchgeführt indem man eine Zufallszahl, die meistens mit der sogenannten Gauß-Glockenverteilung produziert wird, zur Fließkommazahl addiert.

Die dritte Art der Darstellung ist die Ganzzahl-Codierung (integer coding). Dabei wird eine Fließkommazahl durch eine Ganzzahl ersetzt, das zu einem veränderten Mutationsoperator führt.

David Goldberg gab 1989 in seinem Buch über GAs folgenden Rat: "Man sollte das kleinste Alphabet wählen, das die Darstellung eines Problems ermöglicht." Allgemein kann man sagen, daß eine Darstellungsart die für ein bestimmtes Problem hinreichend ist, nicht unbedingt für ein anderes Problem zielführend sein muß.

3.4 Wann sollte man GAs für ein Optimierungsproblem verwenden?

Allgemein gibt es kein Patentrezept wann man einen GA verwenden sollte. Ebenso hängt der Erfolg eines GAs sehr entscheident von der Wahl seiner Parameter ab.
Wenn nicht viel über die Antwortoberfläche bekannt ist und die Berechnung des Gradienten sehr aufwendig bzw. numerisch instabil ist, verwenden sehr viele Forscher GAs. Allgemein kann gesagt werden, daß GAs für SUB-Probleme eine gute Wahl sind. Ebenso sind sie eine gute Wahl, wenn man eine Vermischung verschiedenster Variablendarstellungen hat (zB: Mischung von ganzen, reellen und binären Zahlen). Hierbei muß man jedoch den GA mit verschiedenen Operationen für unterschiedliche numerische Teile des Chromosoms ausstatten.

3.5 Wann sollte man GAs nicht benutzen?

Wenn der Gradientenvektor präzise und schnell zu berechnen ist, sollte man von der Benützung von GAs absehen. Ebenso ist es eventuell ein großes Problem für GAs das exakte globale Optimum zu finden. Deshalb werden GAs oft nur zu einer Annäherung an das globale Optimum benutzt und danach andere Methoden angewendet. Es gibt auch einige Probleme, die als sogenannte **GA-Hards** bezeichnet werden. In diesem Fall liefern die GAs sogut wie keine vernünftigen Ergebnisse. Eine Ursache hierfür kann zum Beispiel sein, daß die Gene auf einem Chromosom nicht richtig geordnet sind und die Länge des Schemas zu lang ist, um vom GA korrekt verarbeitet zu werden.

3.6 Konfiguration des GA

Es ist sehr schwierig die jeweils richtigen Konfigurationswerte für den gewählten GA zu finden. Allgemein gelten folgende Grundsätze:

- **Populationsgrößen**

  Die Populationsgröße gibt die Anzahl der Chromosomen in einer Population an. Eine hohe Anzahl von Chromosomen steigert das Ausmaß der möglichen Variationen pro Population zu Kosten der Fitneß- bzw. Effizienzberechnung, die dadurch sehr aufwendig werden kann. Daher sollte eine gute Populationsgröße immer abhängig von der Chromosomengröße sein. Für lange Chromosome und schwierige Optimierungsprobleme werden hohe Populationsgrößen benötigt, oder man erweitert die Mutations- bzw. Crossover Rate. Viele Forscher schlagen eine Populationsgröße zwischen 25 und 100 vor.

- **Mutationsrate**

Die Mutationsrate gibt die Wahrscheinlichkeit an, wann Mutation effektiv auftritt. Hohe Mutationsraten können zur Zerstörung von Information führen, aber sie vergrößern andererseits die Diverität der Population. Die besten Mutationsraten liegen grundsätzlich zwischen 0.001 und 0.1.

Allgemein kann gesagt werden, daß die Verwendung von GAs viel Erfahrung erfordert und der tatsächliche Erfolg nicht selten von einer günstigen Wahl der Parameter sowie der Codierung abhängt.

Kapitel 4

Projekt-Dokumentation

4.1 Simulierte Welt

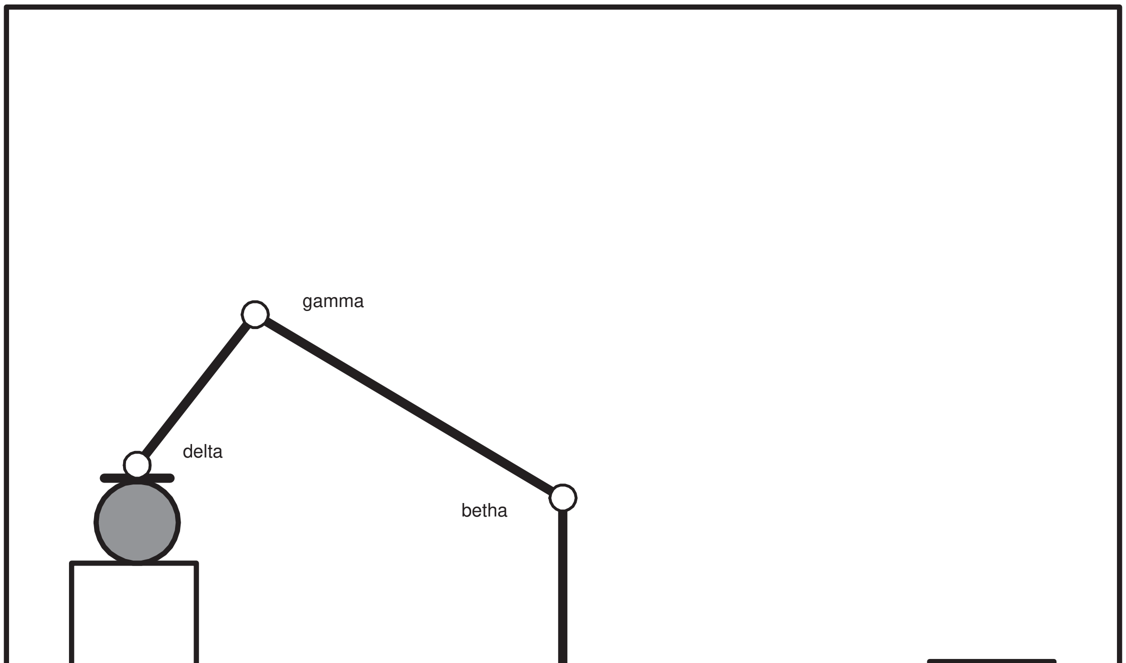

Abbildung 4.1: Simulierte Welt

Der Roboterarm hat drei Glieder und vier Gelenke. Das erste Gelenk ist am Boden befestigt, am letzten ist eine Magnetscheibe befestigt. Die Gelenke können jeden beliebigen Winkel annehmen. Auf beiden Seiten befindet sich ein Podest mit beliebiger Höhe. Auf einem der beiden Podeste liegt eine Eisenkugel.

4.1.1 Aufgabenstellung

Aufgabe für den Roboterarm ist es, die Kugel vom einen Podest auf das andere zu heben. Dazu kann er seine Magnetscheibe benutzen, um die Kugel aufzuheben. Die Aufgabe muß getrennt werden in

- Kugel holen
- Kugel von Podest A nach Podest B bewegen
- Kugel auf Podest B ablegen

Nebenbedingungen sind:

- Die Kugel sollte möglichst in der Mitte des Podests abgelegt werden.
- Der Arm bzw. die Kugel sollte nicht mit dem Podest kollidieren.
- Es soll möglichst wenig Energie verbraucht werden. Dh. der Roboterarm sollte den direktesten Weg anstreben.

4.1.2 Motorik

An jedem Gelenk befindet sich ein Motor. Jeder der Motoren hat einen Eingang, der positive und negative Werte Δw annehmen kann und eine entsprechende Änderung des Winkels w hervorruft.

4.1.3 Sensorik

Es gibt Sensoren die folgende Meßwerte liefern.

- Position der Kugel Kugel.x, Kugel.y
- Position der Magnetscheibe bzw. des untersten Punkts der Kugel, falls die Kugel bereits dranhängt (Quelle.Mx, Quelle.My)
- Zielkoordinaten Ziel.x, Ziel.y
- Einen Status, der die Richtung des Transports angibt:

-1	.....	nach links
+1	.....	nach rechts

- Winkel der Gelenke α , β , γ , δ

4.2 Oberfläche

Die Oberfläche wird als Fenster realisiert, von dem aus alle Einstellungen des Programms entweder über die Menüleiste oder durch direkte Tastatureingabe möglich sind. Der simulierte Greifarm wird im Fenster dargestellt und wird entweder von dem ANN (soweit dieses bereits trainiert ist) oder vom Benutzer direkt gesteuert. Dadurch kann der Benutzer erkennen, wie schwierig es ist die Kugel von einem Podest zum anderen Podest zu bewegen und dabei den kleinsten Weg zurückzulegen.

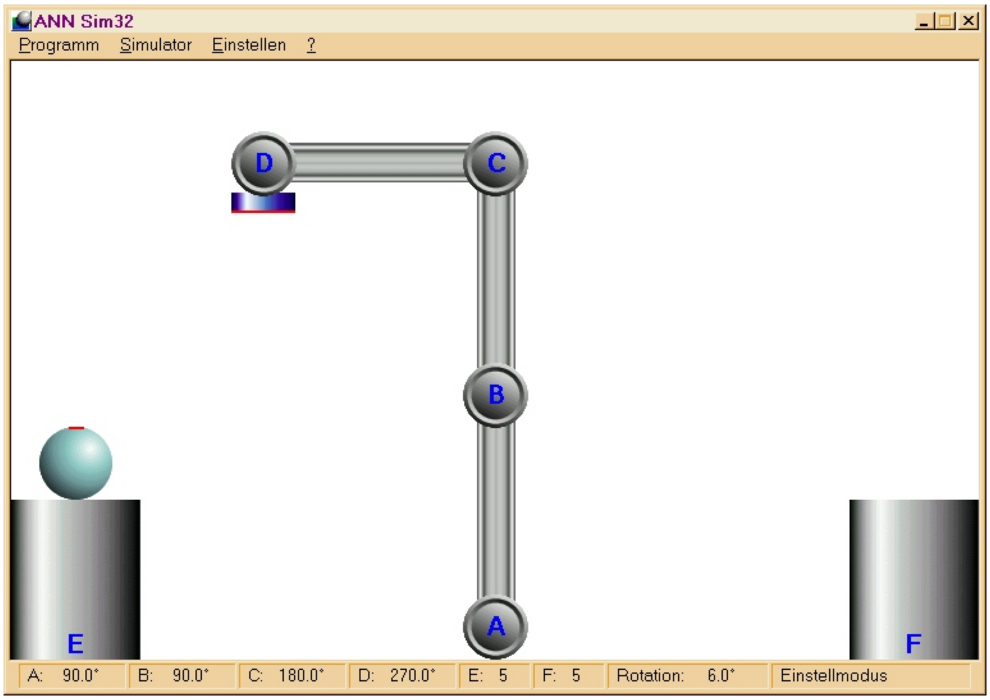

Abbildung 4.2: Benutzeroberfläche - Anfangsstatus

Bei jedem Start des Programmes befindet sich der Greifarm in der Ausgangsposition, wie in Bild 1 zu sehen. Danach kann der Benutzer über den Menüpunkt Simulator/Trainiere ANN oder mit F3 das Training 1 oder F4 Training 2 für das ANN aktivieren. Nach abgeschlossenem Training kann über den Menüpunkt Simulator/Aktiviere ANN oder mit F5 die Steuerung des Greifarmes an das ANN übergeben werden. Weiters kann der Greifarm direkt gesteuert werden, solange das ANN nicht trainiert wird oder die Steuerung des Greifarmes übernommen hat. Die Steuerung erfolgt entwerder über den Menüpunkt Einstellen oder durch die jeweiligen Tasten. Mit dem Menüpunkt Simulator/Zurücksetzen wird der Greifarm sowie die Podesthöhen auf die Startwerte zurückgesetzt. Über Simulator/Umrißmodus wird die gesammte Darstellung des Simulators nur in Umrissen gezeichnet, wodurch die Bewegungen, besonders bei langsamen Rechnern, weitaus schneller und flüssiger als in der Standarddarstellung sind.

4.2.1 Training

Sobald eine der beiden Trainingsarten gestartet wird, erscheint das Statistikfenster für das Training des ANN. Angezeigt werden die Gewichtsdaten des besten ANN, sowie weitere, von der Trainingsart abhängige Daten. Das Ende des Trainings erfolgt über den Knopf Abbrechen.

Trainingsart 1

Die Trainingsart 1 bewertet die einzelnen Mikrobewegungen des Greifarmes. Eine Bewegung in die richtige Richtung ergibt hohe Fitneßwerte. Dies hat zur Folge, daß sofort Erfolge für den Benutzer sichtbar sind und die ANN sehr schnell trainiert ist. Allerdings wird auf diese Weise dem ANN der Gesammtüberblick über das Problem entzogen, wodurch es innerhalb einer Ablegephase zu 'Vergessen' bereits bekannter Bewegungen kommen kann. Bei dieser Trainingsart kann der Benutzer die Bewegungen des Balles während des Trainings direkt im Grafikfenster beobachten. Hier werden zusätzlich die Distanz zum Zielpunkt, die beste Fitness und die Anzahl der besten verwendeten ANN im Trainingsfenster angezeigt.

Trainingsart 2

Bei der Trainingsart 2 des ANN werden gesammte Bewegungsabläufe des Greifarmes bewertet, die zum Aufnehmen, bzw. Ablegen der Kugel führen. Dabei dauert es, im Vergleich mit Trainingsart 1, bedeutend länger zufriedenstellende Ergebnisse zu erzielen. Allerdings wird an einer Gesamtlösung gearbeitet. Das Training wird auch für verschiedene Höhen ausgeführt. Im Trainingsfenster werden zusätzlich die durchschnittliche und beste Fitness, die Anzahl der Generationen und Individuen von ANNs angezeigt.

4.2.2 ANN Steuerung

Sobald das Training des ANN abgeschloßen ist, kann der Greifarm von der ANN gesteuert werden. Dadurch ist für den Benutzer sofort ersichtlich, ob mit den vorhandenen Einstellungen des Trainings das ANN ausreichend für diese Aufgabenstellung ist. Der Greifarm führt alle Winkelveränderungen der ANN durch und sollte den Ball vom linken zum rechen Podest bewegen. Da das ANN auf verschiedene Höhen der Podeste trainiert worden ist, sollte es für das ANN nach längerem Training möglich sein, auch verschiedene Höhen der Podeste zu meistern.

4.3 Artificial Neural Network

4.3.1 Neuronen

Das Neuronale Netz besteht aus dem Eingangslayer, mit elf Neuronen, dem Ausgangslayer mit vier Ausgängen und einem oder mehreren Hiddenlayer. Die Neuronen haben beliebig viele Eingänge, deren Signale bei der Berechnung aufaddiert werden. Sie können einen positiven bzw. negativen Sättigungswert erreichen (Standardmäßig = ±1000).

$\overrightarrow{X} = (x_1, x_2, \ldots, x_n)$	Eingangsvektor des Neurons
y	Ausgang eines Neurons

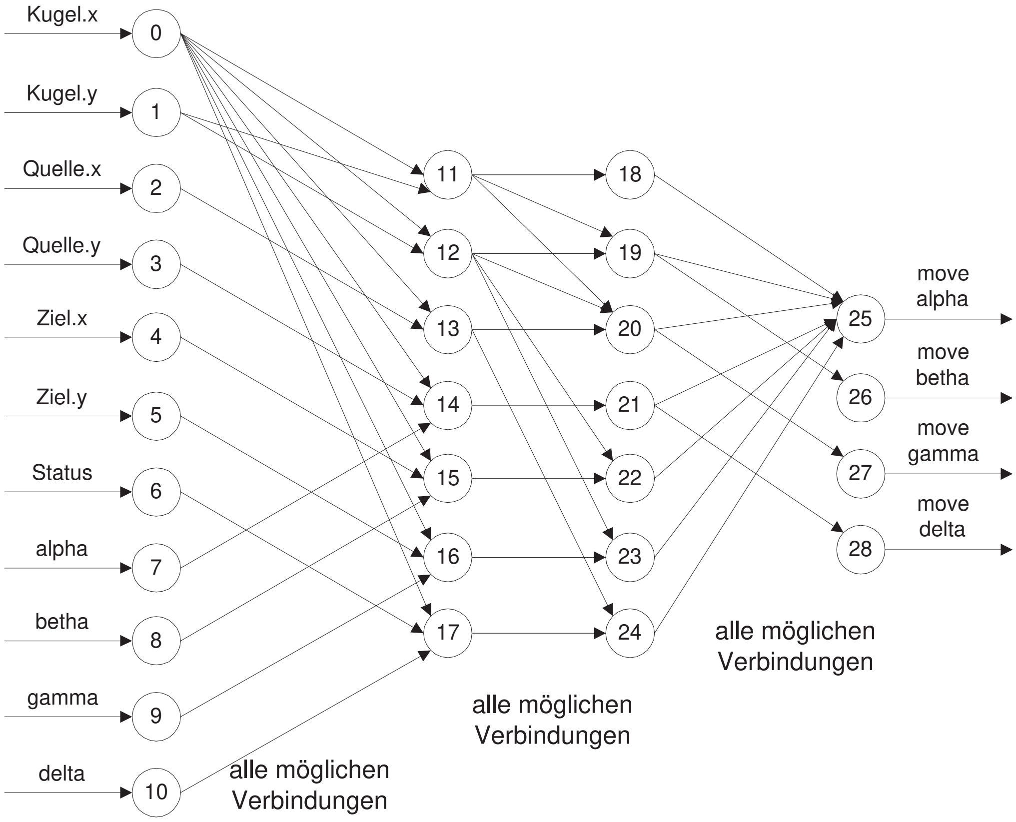

Abbildung 4.3: Artificial Neural Network

Somit ergibt sich für den Ausgang y:

$$y = \sum_{i=1}^{n} x_i$$

Eingangsneuronen

Die Eingangsneuronen müssen der Sensorik wie unter Abschnitt 4.1.3 schon beschrieben entsprechen.

Kugel.x	X-Koordinate des Kugelmittelpunkts
Kugel.y	Y-Koordinate der Kugel
Quelle.x	X-Koordinate der Quelle
Quelle.y	Y-Koordinate der Quelle
Ziel.x	X-Koordinate zu der sich der unterste Punkt der Kugel hinbewegen soll
Ziel.Y	Y-Koordinate des Ziels
Status	gibt die Richtung des Transportes an (siehe Abschnitt 4.1.3)
α	aktueller Winkel des Gelenks am Boden.
β	aktueller Winkel des zweiten Gelenks.
γ	aktueller Winkel des dritten Gelenks.
δ	aktueller Winkel zwischen Magnetscheibe und letztem Arm.

Hidden-Neuronen

Die Anzahl der Neuronen im Hiddenlayer wurde in der Testphase experimentell ermittelt. Zudem wurde eine zweite Schicht eingebaut und die Möglichkeit eingeräumt, die Anzahl der Hiddenlayer bei der Instanzierung der Klasse TANN frei zu wählen. Die Anzahl der Hiddenknoten pro Layer ist für alle Hiddenlayer gleich, und kann ebenfalls bei der Instanzierung angegeben werden.

Ausgangsneuronen

Es gibt vier Ausgänge des Neuronalen Netzes, somit gibt es auch vier Ausgangsneuronen. Die Ausgänge sind mit der Motorik (siehe Abschnitt 4.1.2) verbunden und bedeuten folgendes:

$\Delta\alpha$	Änderung des ersten Winkels
$\Delta\beta$	Änderung des zweiten Winkels
$\Delta\gamma$	Änderung des dritten Winkels
$\Delta\delta$	Änderung des Magnet-Winkels

4.3.2 Verbindungen

Wie schon erwähnt geht von jedem Eingangsneuron eine gerichtete, bewertete Verbindung zu jedem Hiddenneuron der ersten Hiddenschicht . Von jedem Hiddenneuron der ersten Hiddenschicht geht eine Verbindung weiter zu jedem Neuron der zweiten Hiddenschicht, usw. bis zu den Ausgängen.

Speichermodell der Verbindungen:
Das gesamte Neuronale Netz wird als gerichteter bewerteter Graph angesehen und die Verbindungen somit durch die Adjazenzmatrix repräsentiert. Wird ein Wert der Matrix gleich Null gesetzt, so bedeutet dies, daß die entsprechende Verbindung nicht existiert. Da in der Adjazenzmatrix ja alle möglichen Verbindungen berücksichtigt werden, bleiben viele der Einträge gleich Null. Das sind:

- es gibt keine Verbindungen zwischen Neuronen einer Schicht.
- es gibt keine Verbindungen von Schicht i nach Schicht $i - 1$.
- es gibt keine Verbindungen von Schicht i nach Schicht $i + \Delta i$, wobei $\Delta i > 1$.

Die Gewichtungen der Verbindungen geben an, wie stark das Signal des sendenden Neurons verstärkt bzw. abgeschwächt wird. Die Gewichtungen können auch negativ werden. Sie haben aber auf jedem Fall einen maximalen Wert der für den posiviten und den negativen Bereich gilt. Die Werte werden als short-integer codiert. Sie werden allerdings beim Einsatz des Netzes durch einen Genauigkeitsparameter (standardmäßig = 100) dividiert.

4.3.3 Einsatz des ANN

Am Eingang werden zu einem Zeitpunkt t bestimmte Werte anliegen. Das Netz soll nun von diesen Werten abhängig, die Ausgangswerte berechnen.

$\overrightarrow{I} = (i_1, i_2, \ldots, i_l)$	Eingangsvektor des Netzes
$\overrightarrow{O} = (o_1, o_2, \ldots, o_m)$	Ausgangsvektor des Netzes
$Y = \{y_1, y_2, \ldots, y_k, \ldots, y_n\}$	Menge aller Neuronenausgänge (Axone)
y_k	Ausgang des Neurons k
w_{jk}	Gewichtung der Verbindung von Neuron j nach k

Zu einem Zeitpunkt t liegen am Eingang bestimmte Werte $\overrightarrow{I}$ an. Diese bewirken entsprechende Werte $\overrightarrow{O}$ am Ausgang. Es gibt also die Funktion:

$$\overrightarrow{O} = f(\overrightarrow{I})$$

Der Ausgangsvektor ist unabhängig von der Zeit t, da keine Rückkopplungen zugelassen werden. Die Funktion selbst wird durch die Gewichtungen an den Verbindungen des ANN bestimmt. Betrachtet man ein Neuron genauer ergibt sich für dessen Ausgang:

$$y_k = \sum_{j=1}^{n} y_j \cdot w_{jk}$$

4.4 Genetic Algorithm

Die Gewichtungen des ANN werden durch einen Genetischen Algorithmus so eingestellt, sodaß im Endeffekt der Roboter auf Werte, die die Sensorik liefert richtig reagiert. Das erste Struktogramm beschreibt den Grundablauf des Genetischen Algorithmus (Abb: 4.4).

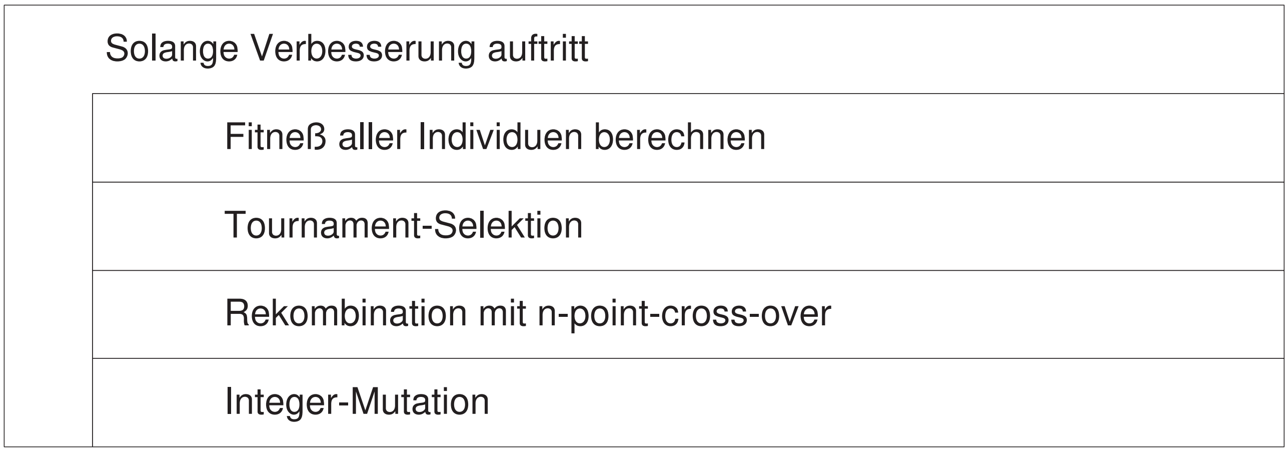

Abbildung 4.4: Genetischer Algorithmus

4.4.1 Chromosomen

Wie üblich bei GA bestehen Chromosomen aus Genen. Jedes Gen entspricht in unserem Falle einer Gewichtung. Da jede Verbindung mit einer Gewichtung bewertet ist entspricht die Anzahl der Gene pro Chromosom (=:g) der Anzahl aller möglichen Verbindungen $g = n^2$. Die Größe der Adjazenzmatrix ist n Zeilen mal n Spalten. Die Gewichtungen werden mit Integerwerten codiert, die jeweils vier Byte im Speicher belegen.

n	Anzahl der Knoten im Netz = Zeilen/Spalten-Anzahl der Adjazenzmatrix
g	Anzahl aller möglichen Verbindungen
l	Anzahl der Eingangsneuronen
m	Anzahl der Ausgangsneuronen
h	Anzahl der Hiddenneuronen pro Hiddenlayer
k	Anzahl der Hiddenlayer
v	Anzahl tatsächlich verwendeter Verbindungen
s	Speicherbelegung der Adjazenzmatrix

Die Anzahl der Verbindungen v, die wirklich vorhanden sind, das heißt wo die Gewichtung ungleich Null ist, ergibt sich zu:

$$v = l \cdot h + h^2 \cdot k + h \cdot m$$

bei 7 Neuronen pro Hiddenlayer:

$$v = 11 \cdot 7 + 7^2 \cdot 2 + 7 \cdot 4 = 203$$

mit:

$$n = l + h \cdot k + m$$

$$n = (11 + 7 \cdot 2 + 4) = 29$$

$$s = n^2 \cdot 4 \text{ Byte} = 29^2 \cdot 4 = 3,4 \text{ KB}$$

4.4.2 Fitneßberechnung Variante 1

Jedes der Individuen wird hier auf ihre Fitneß getestet. Das heißt das ANN, des Individuums errechnet aus den aktuellen Werten der realen Welt die Ausgangsparameter, die nun an die reale Welt zurückgegeben werden. Die reale Welt führt jetzt lediglich alle Berechnung durch, führt aber die Änderungen nicht aus. Diese Berechnung liefert dem Fitneßalgorithmus nun neue Parameter der realen Welt zurück. Die Fitneßfunktion hat nun zwei Parameterlisten, die sie vergleichen kann. Das Struktogramm in Abb 4.5 veranschaulicht diesen Algorithmus.

Fitneßfunktion

Die Sensorik und folgende Kriterien wurden so gewählt, daß die Fitneßfunktion nach jeder Mikro-Bewegung errechnet werden kann.

- Gute Bewertung wenn Magnetscheibe waagrecht ist.
- Gute Bewertung, wenn sich die Magnetscheibe genau in Richtung Ziel bewegt.
- Beim Holen ist das Ziel der oberste Punkt der Kugel.
- Beim Bewegen zum anderen Podest ist das Ziel der Mittelpunkt der Plattform. Der unterste Punkt der Kugel soll dem obersten Punkt der Plattform angenähert werden.

Die Fitneßfunktion übernimmt zudem die Trennung der unterschiedlichen Schritte, die notwendig sind um die Aufgabe zu lösen. (siehe Absatz 4.1.1) Während der Berechnung der Fitneß, wird die maximale Fitneß ermittelt und das entsprechende Chromosom gesichert.

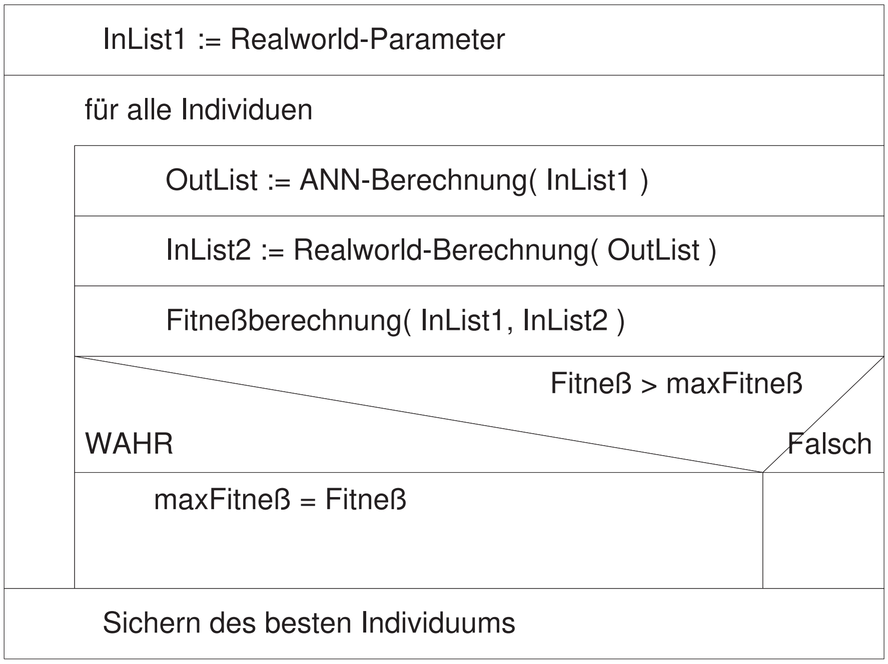

Abbildung 4.5: TWorld::calcFitness

4.4.3 Fitneßberechnung Variante 2

Beim Testen wurde erkannt das die Bestimmung der Fitneß nach jeder Mikrobewegung folgende Nachteile birgt:

- Die Entwicklung führt leicht zu lokalen Maxima, und zwar in der Nähe des rechten Podests.
- Das ANN ändert sich ständig und erreicht nie einen konstanten Zustand. Das kommt daher, da das Netz an jeder Stelle neu lernt sich in Richtung Ziel zu bewegen. Es verlernt also alte Stellungen.

Aus diesen Gründen wurde die Fitneßberechnung umgestellt. Jedes Individuum bekommt eine bestimmte Zeit lang eine Chance zu zeigen was es kann. Die meisten werden natürlich irgendwie sinnlos herumfahren. Aber eines von hundert wird vielleicht dabei sein, daß die Kugel erwischt und dann von links nach recht transportiert.

Das Individuum bekommt 200 Schritte zur Verfügung. Wird die Kugel getroffen, so wird es mit Fitneß belohnt und bekommt diesmal etwas weniger Schritte dazu, um vom linken zum rechten Podest zu gelangen. Erreicht es dieses Ziel so, bekommt es wieder einige Schritte um das nächste Ziel zu erreichen, usw... Die Größe der Belohnung ist umgekehrt proportional zu den Schritten, die dazu benötigt werden.

Gehen die Schritte aus, so ist die Fitneßberechnung, für das Individuum beendet und das selbe Spiel beginnt für das nächste. Das Individuum mit der größten Fitneß wird extra gespeichert. Es wird später bei der Steuerung verwendet. Für die Statusausgabe wird außerdem die durchschnittliche Fitneß berechnet. Sie gibt Auskunft über die Entwicklung der Population.

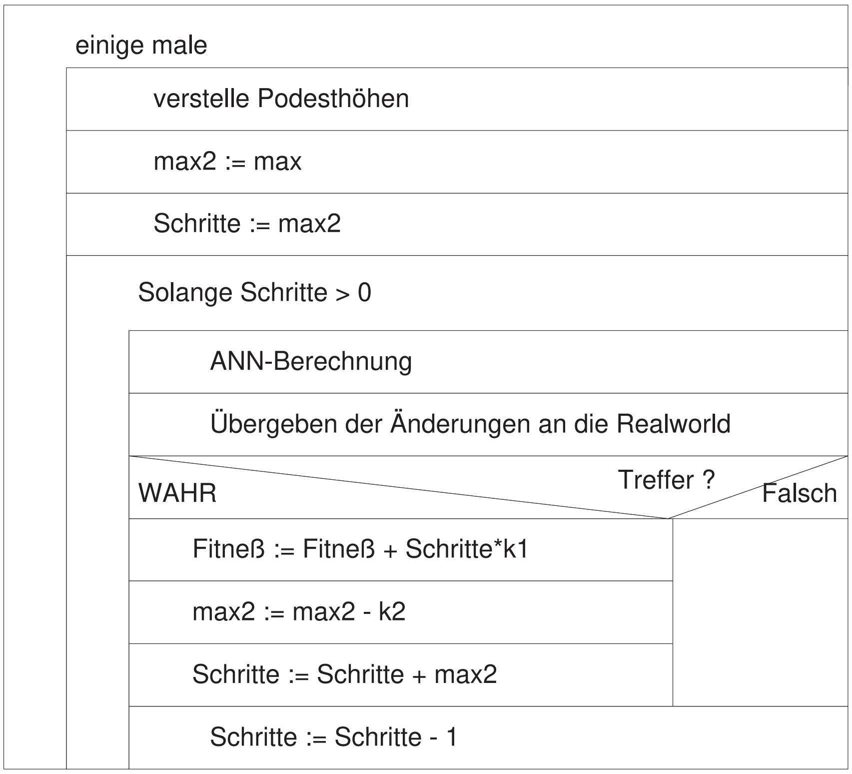

Abbildung 4.6: TChromosom::calcFitness2

4.4.4 Selektion

Als Selektionsmethode wurde hier eine spezielle Variante der Tournamentselektion gewählt. Es werden immer zwei Individuen zufällig gewählt, verglichen und das bessere der beiden überschreibt das schlechteste der ganzen Population. (siehe Abb 4.7)

4.4.5 Rekombination

Zunächst wird die Rekombination in zwei Teile unterteilt. Im ersten Teil wird die Hälfte der Population zufällig miteinander gepaart. Jedes gepaarte Individuum wird markiert, damit es nicht nocheinmal gepaart wird. Die zweite Hälfte wird dann der Reihe nach gepaart, damit der Zufallsgenerator nicht ständig auf die selben Individuen trifft und somit Rechenzeit verschwendet.
Vor dem Paaren zweier Individuen entscheidet wieder ein Zufallswert der mit einer Wahrscheinlichkeit p_R bewertet ist, ob die Rekombination zwischen diesen beiden Chromosomen überhaupt stattfindet. Es wird nun n-Point-Crossover verwendet. Die Anzahl der Crossover-Punkte wird mit dem Zufallsgenerator ermittelt, die n Crossover-Punkte ebenfalls. Zwischen jedem zweiten Crossover-Punkt werden nun die Gene ausgetauscht.

4.4.6 Mutation

Jeder Wert der Adjazenzmatrix, der eine mögliche Verbindung im ANN darstellt, wird mutiert. Dazu werden die Integerwerte in ihrer Binärdarstellung betrachtet. Die Bits werden

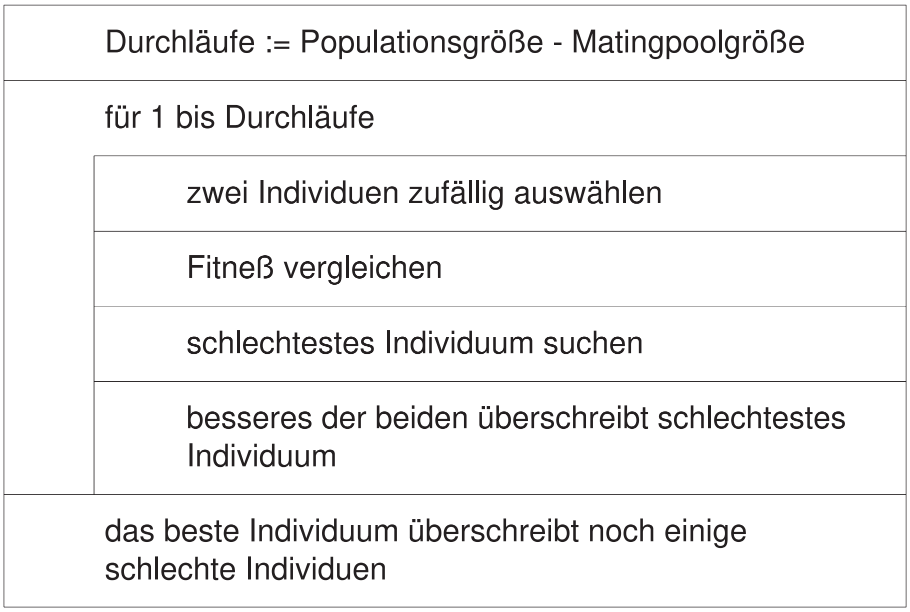

Abbildung 4.7: TWorld::Selection

der Reihe nach durchgegangen und kippen mit einer Wahrscheinlichkeit von p_{Mi}, wobei diese Wahrscheinlichkeit abhängig ist von der Wertigkeit des Bits 2^i. Bits, die weiter links in der Binärdarstellung der Integerzahl stehen sind mit einer geringeren Wahrscheinlichkeit p_{Mi} behaftet. Begonnen wird die Mutation beim höchstwertigten Bit ganz links. Die Mutationsrate soll die Kippwahrscheinlichkeit des Bits 0 ganz rechts angegeben. Die Kippwahrscheinlichkeit für das vorderste Bit errechnet sich also mit:

$$p_{Mn} = p_M \cdot k^{-n}$$

Die Wahrscheinlichkeit für jedes weitere Bit ergibt sich zu:

$$p_{Mi} = p_M \cdot k^{-i}$$

Wobei k die relative Änderung der Wahrscheinlichkeit von einem Bit zum nächsten angibt und i den Abstand zum least significant Bit.

Die Wertigkeit w der einzelnen Bits wird benötigt damit das Kippen mittels XOR-Funktion durchgeführt werden kann. Die Wertigkeit eines Bits i ist also:

$$w_i = 2^i$$

Die Entscheidungszahl für das Kippen eines Bits ergibt sich nun zu:

$$E = p_{Mi} + U(\,[0,1[\,)$$

Wobei U hier eingführt wird als gleichverteilte Zufallszahl im Intervall $[0,1[$. Der linke Teil in der Summe ist die Wahrscheinlichkeit für die Änderung. Die Änderung wird durchgeführt, wenn die Summe $E \geq 1$.

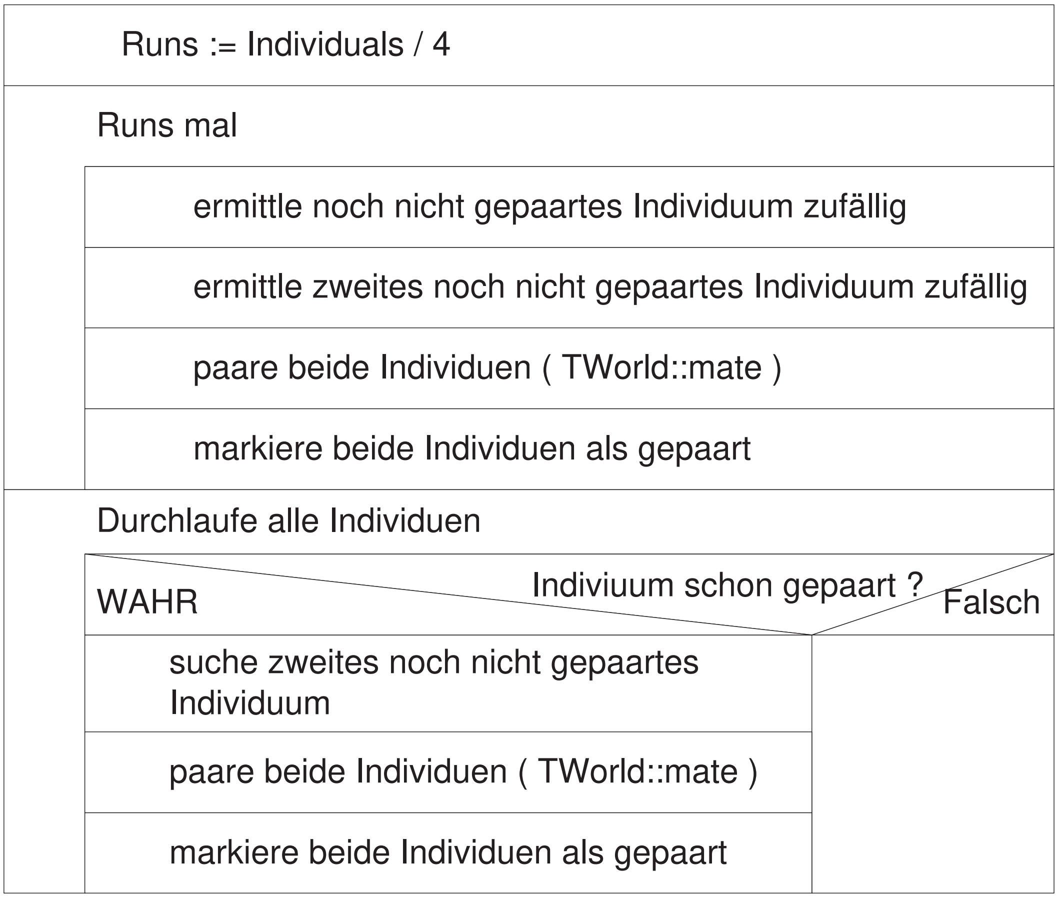

Abbildung 4.8: TWorld::Crossover

Vorzeichen:
Die Wahrscheinlichkeit, daß das Vorzeichen kippt ist abhängig vom Absolutwert des Gens. Umso größer die Zahl desto unwahrscheinlicher ist, daß sich das Vorzeichen ändert.

$$p_{MV} = \frac{p_M \cdot (m - \mid g \mid)}{m} + U(\,[0,1[\,)$$

Wobei m der maximale Wert der auftreten kann ist und g der Wert des Gens.

4.4.7 Abbruchbedingung

Der Genetische Algorithmus wird dann beendet, wenn nach einer bestimmten Anzahl von Generationen n_G keine Änderung mehr aufgetreten ist.

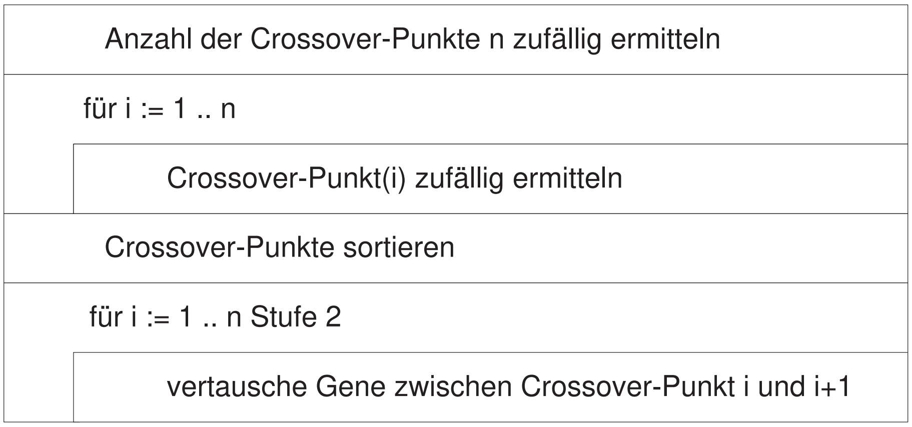

Abbildung 4.9: TWorld::mate

Anhang A

Fragenaufstellung

Fragen der Version vom 17.03.99 15:45

- Maximale Anzahl der zur Zeit realisierbaren Neuronen?
  Ein Schaltwerk aus fast $40 * 10^6$ *Neuronen ist geplant.[6]*

- Sind Neuronen immer kombinatorische Schaltkreise?
  nein

- Hauptanwendungsgebiete für ANN?
  Mustererkennung, Börsenkurse, Spracherkennung, Roboterfußball, ...

- Gibt es bevorzugte Programmiersprachen für ANN?
  nein

- Sind bereits ANN Beispielprogramme erhältlich?
  ja, Stuttgarter Neural Network Simulator.
  http://www.informatik.uni-stuttgart.de/ipvr/bv/projekte/snns/snns.html

Fragen der Version vom 24.03.99 15:45

- Sollen die Fragen in dieser Liste (wie hier) mitgeführt oder jedesmal überschrieben werden?
  mitführen

- Gibt es die Variante beim Festlegen der Verbindungen, wo jeder Knoten mit jedem anderen Verbunden wird wirklich? Wie schauen dann die Ein-/Ausgänge der Knoten aus?
  Ja, die n Knoten haben dann n-1 Eingänge und die Ausgänge teilen sich entsprechend oft auf.

- Inwieweit stimmt diese Skizze ?

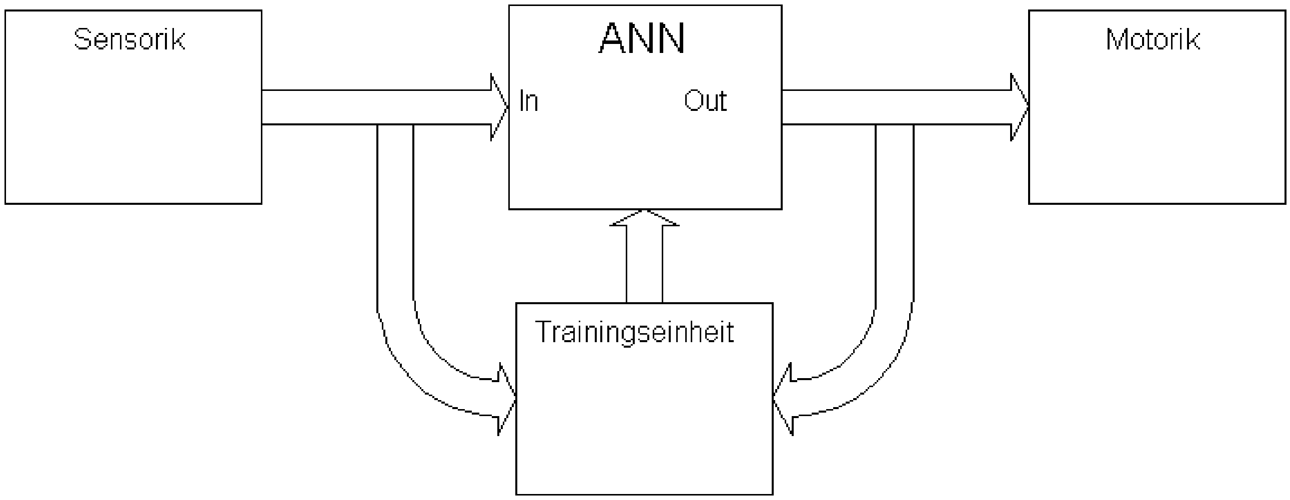

Abbildung A.1: ANN als Teil eines Regelkreises?

Stimmt. Der Trainingsalgorithmus besteht im Prinzip aus einer Tabelle, in der zu Eingangstuppel passende Ausgangstuppel gegenübergestellt werden.

- Kann der Threshold - Wert dynamisch geändert werden?
  Der Threshold steckt in der genetischen Information, ob er sich ändert hängt davon ab, ob sich in dem jeweiligen Modell der Gencode ändert.

Fragen der Version vom 14.04.99 15:45

- Gibt es in diesem Proseminar auch mal was praktisches zu machen (z.B.: Programmieren)?
  Auf freiwilliger Basis. Ansonsten ist das Ergebnis dieses Paper.

Fragen der Version vom 20.04.99 15:45

- Wie werden Rückkopplungen in die Hidden-Layers eingesetzt ?

Fragen der Version vom 05.05.99 15:45
keine Fragen

Anhang B

Glossar

ANN
Artificial Neural Network : künstliche neuronale Netze - Modell eines Nervensystems und dessen Umsetzung in ein technisches Netzwerk.

neural
neu—ral [gr.-nlat.]: einen Nerv, die Nerven betreffend, vom Nervensystem ausgehend (Med.). manchmal auch: neuronal [7]

Neuron
Neu—ron das; -s, ...onen (auch: ...one): Nerveneinheit, Nervenzelle mit Fortsätzen (Med., Biol.). [7]

Dendrit
Den—drit [gr.-nlat.] der; -en: 2. verästelter Protoplasmafortsatz (vgl. Protoplasma) einer Nervenzelle (Med.). [7]

Protoplasma
Pro—to—plàs—ma [gr.-nlat.] das; -s: Lebenssubstanz aller pflanzlichen, tierischen u. menschlichen Zellen. engl.: protoplasm [7]

Axon
Axon [gr.] das; -s, -e u. -en: Neurit (Biol., Med.). [7]

Neurit
Neu—rit der; -en -en: oft lang ausgezogener, der Reizleitung dienender Fortsatz der Nervenzellen (Med., Biol.). [7]

Synapse
Syn—àp—se [gr.] die; -, -n: (Biol., Med.) 1. Kontakt-, Umschaltstelle zwischen Nervenfortsätzen, an der nervöse Reize von einem Neuron auf ein anderes weitergeleitet werden. 2. Berührungsstelle der Grenzflächen zwischen Muskel u. Nerv. [7]

Stimulus
Sti—mu—lus der; -, ...li: a) Reiz, Antrieb; b) ein dem Sprechakt vorausgehender [äußerer]

Reiz (Sprachw.). [7]

Habituation
Ha—bi—tua—ti—on [...zion; lat.] die; -, -en: a) Gewöhnung (Med., Psychol.); b) physische u. psychische Gewöhnung an Drogen. [7]

Metabolismus
Me—ta—bo—lís—mus [gr.-nlat.] der; -: 1. Umwandlung, Veränderung. 2. Stoffwechsel (Med., Biol.). engl.: metabolism [7]

Diffusion
Dif—fu—si—on [lat.; ´das Auseinanderfließen´] die; -, -en: 1. a) ohne äußere Einwirkung eintretender Ausgleich von Konzentrationsunterschieden (Chem.) [7]

reinforcement learning
verstärkendes Lernen
excitory dendrite
dendrite, die das Signal ohne Änderung aufnehmen

inhibitory dendrite
dendrite, die das Signal aufnehmen,jedoch dessen Vorzeichen invertieren

presynaptisches Neuron
sendendes Neuron; (pre = vor)

postsynaptisches Neuron
empfangendes Neuron; (post = nach)

Anhang C

Protokolle

C.1 Protokoll vom 17.3.1999, 15:45

Anwesend:

Roland Schwaiger
Rainer Trummer
Ferdinand Schober
Christian Spielberger

- Bewertung der Version vom 10.3.1999:
  - zu wenig Informationen
  - zu ungenau
  - folglich: Wir sollen es so schreiben, daß es ein Außenstehender verstehen würde.
- Beantwortung der Fragen vom 10.3.1999
- Aufgaben bis zum nächsten Termin:
  - Dokument in latex-format umschreiben.
  - Titelseite und Literaturübersicht laut Beispiel auf PS-Homepage schreiben.
  - Abstract und Protokoll einfügen
  - im Text immer Verweise auf Literatur angeben
  - Ausbesserung des Textes laut Korrektur
  - Glossar ins Paper aufnehmen
  - Artikel ´Formation of Neural Structures´ genau durcharbeiten und zusammenfassen

Ende: 16:15

C.2 Protokoll vom 24.3.1999, 15:45

Anwesend: Roland Schwaiger Rainer Trummer Ferdinand Schober Christian Spielberger

- Bewertung der Version vom 10.3.1999: sehr viel besser
- Leitfaden zum Thema Wachstum von ANN: Growing ANN ist kein allgemein gültiger Begriff. Vielmehr gibt es:
  - Network Growing
  - Network Prunning
  - Evolutionary Design
- Besprechung einiger Details wie:
  - Addierer im Knoten mit beliebig vielen Eingängen
  - Ausgangsfunktion
  - Gewichtungen der Verbindungen
  - Wahl der Verbindungen (zufällig oder vollständig)
  - Realisierung des Thresholds mittels speziellen Knotens, der ein kontinuierliches negaties Signal liefert.
- Aufgaben bis zum nächsten Termin:
  - Dokument in latex-format umschreiben.
  - Das ´Lexirom-Mini´ ins Bib.-Verzeichnis aufnehmen.
  - Zusätzlicher Abschnitt in der Einleitung über die biologischen Grundlagen.
  - Wie funktioniert das Wachstum des Neuronalen Netzes in der Natur (Gehirn).
  - Grundlagen der GA.
  - Wie wächst ein ANN?
  - überwachtes und unüberwachtes Lernen (Robo-Cat)
  - reinforcement learning.

Ende: 16:15

Anhang D

Entwicklungsdokumentation

D.1 Datum: 14.05.1999

D.1.1 Festlegungen

- Das Programm soll in Borland C++ geschrieben werden.
- Verwendung von objektorientierter Programmierung.

D.1.2 class TANN

Deklaration einer Klasse TANN, die das ANN repräsentiert. Die Anzahl der Knoten ist variabel, und wird durch folgende **Eigenschaften** bestimmt:

Nodes	gesamte Knotenanzahl des ANN
InputNodes	Anzahl der Eingangsknoten
OutputNodes	Anzahl der Ausgangsknoten
HiddenNodes	Anzahl der versteckten Knoten

Weitere Eigenschaften der Klasse sind:
short maxWeight
Bereich der Gewichtung geht von -maxWeight bis + maxWeight.

short precWeight
Gibt die Genauigkeit der Gewichte an. Dabei bedeutet 100 z.B. das auf Hundertstel genau gerechnet wird.

short Treshold
Die vier Ausgangsknoten benötigen diesen Schwellwert, um von Null auf +1 bzw. -1 schalten zu können.

```
struct TNode
      {     int internValue
            short Output
      }
```

Der Knotentyp des ANN besteht aus zwei Unterfeldern, dem internen Wert, welcher der Summe der Eingänge entspricht und aus dem Ausgangswert. Der Ausgangswert wird durch eine Ausgangsfunktion aus dem internen Wert errechnet.

int maxIntern
Der interne Wert des Knotens kann eine Sättigung erreichen, bei der er einfach stehenbleibt, wenn die Summe der Eingänge größer wäre.

typedef TNode *PNode;
Zeiger auf Knotentyp

PNode NodeList;
Knotenliste als dynamischer Array

typedef short TWeight;
Typ der Gewichte auf den Verbindungen im ANN

typedef TWeight *PWeight;
Zeiger auf Gewichttyp

PWeight *AdjazenzMatrix;
Die Adjazenzmatrix des ANN legt die Verbindungen und deren Gewichte fest. Sie hat so viele Zeilen wie es Knoten gibt. Die Anzahl der Spalten ist gleich groß.

Methoden von TANN

initANN
Initialisiert obige Eigenschaften, erzeugt die Knotenliste und die Adjazenzmatrix, außerdem wird die Adjazenzmatrix mit Zufallswerten belegt. Dabei bleiben Verbindungen die nicht vorhanden sind (z.B.: zwischen Input-Layer und Output-Layer) gleich null.

runANN
Übernimmt die Eingangssignale in den Inputlayer. Berechnet dann der Reihe nach die Werte für die Hiddenlayer, dann für die Ausgangslayer. Beim Ausgangslayer wird der Schwellwert berücksichtigt.

D.2 Datum: 15.05.1999

D.2.1 class TChromosom

Jedes Chromosom im genetischen Algorithmus soll ein mögliches ANN repräsentieren, zusätzlich einige Eigenschaften enthalten, wie z.B. Fitneß. Die Idee ist nun die Klasse TANN der Klasse TChromosom zu vererben.

Eigenschaften von TChromosom

int Fitneß
Die Fitneß des Chromosoms sagt aus wie gut das ANN, das es enthält funktioniert.

short ApproximationFactor
Der Näherungsfaktor ist ein entscheidender Parameter für die Fitneßfunktion und wird mit der Annäherung des Greifers zum entsprechenden Ziel multipliziert. Das Produkt ergibt den Fitneßwert.

short BallRadius
Kugelradius

float MutationRate
Mutationswahrscheinlichkeit p_M gibt die Wahrscheinlichkeit an, daß ein Bit im Bit-String kippt.

Methoden von TChromosom

init
Initialisiert die das ANN und die Eigenschaften von TChromosom.

calcFitness
Berechnet die Fitneß des Chromosoms. Die Parameter aus der realen Welt werden betrachtet. Es werden genauer zwei Parametersätze, der aktuelle Satz und ein älterer, verglichen. Die Änderung der Parametersätze wird bewertet.
Hier wird einstweilen nur die Entfernung errechnet. Die Differenz zur vorigen Entfernung mit ApproximationFactor multipliziert ergibt die Fitneß.

mutate
Mutiert das Chromosom. Genauer: Verändert die Gewichtungen des ANN wie folgt:

- Gewichtungen mit Wert 0 werden unverändert gelassen, denn das würde heißen, daß neue Verbindungen hinzukommen. ($\longrightarrow$ vielleicht spätere Erweiterung - wachsende ANN)

- Jeder der Werte wird nun gesondert für die Mutation betrachtet.

- Die Werte sind Short Integer und die vordersten Bits sind somit höherwertig als die hinteren. Die Asymmetrie der Integerzahlen wurde durch Anpassung der Bit-Mutationswahrscheinlichkeit teilweise ausgeglichen. Dabei wird die Anzahl der benötigten Bits für die Gewichtungen ermittelt (Standardmäßig: 9 Bits). Dann wird das mittlere Bit auf die MutationRate gesetzt. Die Bits links davon bekommen eine mit einer Konstante dividierte Wahrscheinlichkeit, und die Bits rechts davon bekommen eine mit der Konstante multiplizierte Wahrscheinlichkeit für die Mutation.

D.3 Datum: 15.05.1999 - 16.05.1999

D.3.1 class TWorld

Die Welt des Genetischen Algorithmus beinhaltet die Population, die aus einer konstant großen Liste von Induviduen besteht.
Problem: Wie deklariert man in C++ einen dynamischen Array von Objekten? Wäre das möglich, dann könnte man die Anzahl der Individuen während des Programmablaufes auch verändern.
Es wird also ein Array deklariert mit 100 Objekten von der Klasse TChromosom. Die Objekte von TChromosom werden nun Individuen genannt und mit Population[idx] angesprochen.

Eigenschaften von TWorld

short bestIndividualIdx
Index des Individuums mit der größten Fitneß.

class TChromosom bestInduvidual;
bestes Individuum

int maxFitnessLast
höchste Fitneß in der vorigen Generation. Dieser Wert wird mit der aktuellen Höchst-Fitneß verglichen. Dieser Vergleich bildet die Abbruchbedingung des GA.

short NoImprovement
Anzahl der Generationen, die keine Verbesserung mehr hervorgebracht haben.

short maxNoImprovement
Erreicht NoImprovement den Wert in maxNoImprovement so wird der GA verlassen.

short MatingPoolSize
Größe des Matingpools. Gibt die Anzahl der unterschiedlichen Individuen an, die für die Rekombination selektiert werden.

int *InputList
Diese Liste von short-Integer-Werten dient dazu mit der realen Welt Daten auszutauschen. Zusätzlich wird diese Liste dem ANN übergeben.

int *InputListOld
Zweiter Speicher für InputList.

int *OutputList
Liste der Ausgangsparameter vom ANN. Wird der realen Welt übergeben, und gibt dort die Ansteuerung der Motoren an.

Methoden von TWorld

TWorld
Konsktruktor dieser Klasse. Sie initialisiert die Chromosome und somit auch die ANNs. Initialisiert die eigenen Eigenschaften.

calcFitness
Berechnet die Fitneß aller Individuen. Dazu werden die aktuellen Parameter der realen Welt zunächst ermittelt und in InputListOld gespeichert. Dann wird für jedes Individuum ein Testlauf durchgeführt. Ein Testlauf besteht aus:

- dem Aufruf des ANN, daß mir aus den Parametern in InputListOld die Ausgangsparameter in OutList liefert,
- dem Übergeben von OutList an die Realworld-Funktion. Ein spezieller Parameter dieser Funktion ist ein Anzeiger, der hier false gesetzt wird. Dieser steuert ob nur die Berechnung durchgeführt wird, oder ob die Änderung tatsächlich ausgeführt wird. Diese Funktion liefert die neuen Realworld-Parameter in InputList zurück.
- Die Parameter in InputList und InputListOld werden nun von der Methode calcFitness verglichen. Diese Methode liefert den Fitneß-Wert des Individuums zurück.
- In der Schleife wird die maximale Fitneß ermittelt.

getWorst
Liefert den Index des schlechtesten Individuums. Dies wird für die Selektion benötigt.

Selection
Selektion der besten Individuen. Hier wird Tournament-Selektion verwendet. Es werden immer zwei Individuen zufällig ausgewählt und verglichen. Das bessere überschreibt das schlechteste in der ganzen Population. Bei jedem Vergleich fällt also ein schlechtes Individuum weg. Die Anzahl der Vergleiche ist somit:
Populationsgröße - Matingpoolgröße

Nach der Selektion sind Matingpoolgröße viele verschiedene Individuen in der Population. Es gibt also mehrfach gleiche Individuen.

crossOver
Führt die Rekombination durch (Genetischer Operator). Verwendet wird Onepoint - Crossover. Der Crossoverpoint kann hier nur zwischen zwei Werten der Adjazenzmatrix liegen, nicht mitten in einem Integerwert. Das würde auch wenig Sinn machen. Die Auswahl der beiden Individuen erfolgt anfangs zufällig. Die gepaarten Individuen werden markiert, damit sie nicht nocheinmal gepaart werden. Nachdem die Hälfte der Population zufällig miteinander gepaart wurde, werden die restlichen der Reihe nach gepaart, damit der Zufallsgenerator nicht ständig bereits gepaarte Individuen liefert und somit Zeit verschwendet.

mutate
Mutiert alle Individuen.

useBestIndividual
Verwendet das beste Individuum um die Bewegung in der realen Welt auszuführen.

D.4 Schnittstelle zwischen GA/ANN und simulierter Welt

D.4.1 Parameter der realen Welt

Die Parameter sind im folgenden Array vom Typ Int gespeichert.

Typ	**IN/OUT**	**Beschreibung**
Array[11] of Int	OUT	Darin werden die Eingangsparameter für das ANN übernommen.

Array-Einträge	**Index**	**Parameter**	
	0	Kugel.x	Kugelmittelpunkt
	1	Kugel.y	
	2	Quelle.x	Magnetunterseite, wenn die Kugel auf der Plattform liegt
	3	Quelle.y	Kugelunterseite, wenn die Kugel schon dranhängt
	4	Ziel.x	Kugeloberseite, wenn sie noch liegt
	5	Ziel.y	Plattformoberseite, wenn sie dranhängt
	6	Status	-1 nach links, +1 nach rechts
	7	alpha	
	8	beta	
	9	gamma	
	10	delta	

D.4.2 GetCalcData

Deklaration:
int far* GetCalcData(int delta[4], bool keep, bool draw);

Diese Procedure sollte die Mikro-Bewegung mit den Ansteuersignalen für die Motoren ausführen.

Eingangsparameter:

delta[4]	Übernimmt die vier Winkeländerungen, die in die Winkel der Gelenke eingerechnet werden.
keep	Entscheidet, ob die berechneten Werte in die reale Welt übernommen werden sollen, oder ob sie lediglich an die aufrufende Funktion zurückgegeben werden sollen.
draw	Entscheidet, ob die aktuelle reale Welt dargestellt werden soll, oder nicht.

Ausgangsparameter:
Es wird ein Pointer auf die Parameter der Realworld zurückgegeben. Das ist also ein Pointer auf einen Array mit elf Integer-Einträgen.

D.5 Datum: 21.05.1999

D.5.1 Testphase

Getestet wurden:

- Class TANN
- TWorld::Selection
- TChromosom::mutate somit ist auch TWorld::mutate ok.
- TWorld::mate, später muß noch TWorld::CrossOver getestet werden.
- TChromosom::calcFitness später muß noch TWorld::calcFitness getestet werden.

D.6 Datum: 22.05.1999 - 23.05.1999

D.6.1 Dokumentation

Dokumentation in Latex vervollständigen.

D.7 Datum: 31.05.1999

D.7.1 Projekt zusammenfügen

Verbinden der beiden Teile, ann-ga.h und annsim.h.

D.8 Datum: 09.06.1999

- Korrektur der Übergabeparameter. Wie unter Abschnitt D.4.1 ersichtlich.
- Es wurde ein Fenster eingebaut, daß aktuelle Daten des Programms anzeigen kann. Die jeweiligen Variablen, die darin angezeigt werden sollen, können durch Einfügen von zwei Zeilen an einer beliebigen Stelle im GA-Algorithmus hinzugefügt werden. In diesem Fenster wird auch die Adjazenzmatrix des besten Individuums dargestellt.
- Korrektur der Winkelwerte, sodaß sie zwischen -180 und $+180$ liegen.
- Da vermutet wird, daß im idealen Netz, falls es jemals gefunden wird, bestimmte Bereiche der Adjazenzmatrix für bestimmte Eingangsparameter zuständig sein werden, wurde das One-Point-**Crossover** durch ein n-Point-Crossover ersetzt. Die Anzahl n und die n Crossover-Points werden mittels Zufallsgen. erzeugt. Dann müssen die Crossover-Points sortiert werden, bevor die Bereiche zwischen zwei Punkte abwechselnd einmal vertauscht und dann wieder nicht vertauscht werden.

D.9 Datum: 10.06.1999

Mutation: Beim Testen wurde die Notwendigkeit erkannt sie anzupassen, weil manchmal eine Gewichting von null herauskam. Wenn eine Gewichtung einmal bei null angelangt ist, wird sie nämlich immer dabei bleiben.
Idee: Es wird die Möglichkeit eingebaut Nuller auch zu mutieren, allerdings nur zwischen Input- und Hidden-Layer und zwischen Hidden- und Output-Layer. Dafür werden kleine Werte null gleichgesetzt und die Anzahl der Null-Werte als positives Kriterium in die Fitneßberechnung aufgenommen. Somit wird zusätzlich zur Problembehebung ein Wachstum (Schrumpfen) der Verbindungen erreicht.

D.10 Datum: 14.06.1999

Fitneßfunktion: Beim Testen wurde erkannt das die Bestimmung der Fitneß nach jeder Mikrobewegung, dazu führt, daß:

- Die Entwicklung führt leicht zu lokalen Maxima, und zwar in der Nähe des rechten Podests.
- Das ANN ändert sich ständig und erreicht nie einen konstanten Zustand. Das kommt daher, da das Netz an jeder Stelle neu lernt sich in Richtung Ziel zu bewegen. Es verlernt also alte Stellungen.

Aus diesen Gründen wurde die Fitnessberechnung umgestellt. Jedes Individuum bekommt eine bestimmte Zeit lang eine Chance zu zeigen was es kann. Die meisten werden natürlich irgendwie sinnlos herumfahren. Aber eines von hundert wird vielleicht dabei sein, daß die Kugel erwischt und dann von links nach recht transportiert.

Die neue Fitneßberechnung ist in den Funktionen TWorld::calcFitness2 und TChromosom::calcFitness2 implementiert. Die alte kann noch immer aufgerufen werden, und ist jetzt in den Funktionen TWorld::calcFitness1 und TChromosom::calcFitness1.

In TChromosom::calcFitness2 bekommt das Individuum 200 Schritte zur Verfügung. Wird die Kugel getroffen, so bekommt es wieder (fast) 200 Schritte um vom linken zum rechten Podest zu gelangen. Erreicht es dieses Ziel so, bekommt es wieder einige Schritte um das nächste Ziel zu erreichen, usw...
In TWorld::calcFitness2 werden alle Individuen durchlaufen, die maximale und die durchschnittliche Fitness errechnet.

Hiddenlayers: Da sich noch immer keine Besserung zeigte, wurde das ANN auf beliebige Anzahl der Hiddenlayer erweitert. Zur Zeit werden zwei Hiddenlayer mit jeweils sieben Knoten verwendet.

Fitneßfunktion: Das Problem liegt darin, daß wenn der Winkel der Plattform bewertet wird, wird als Ergebnis nicht nur die Plattform waagrecht gehalten sondern auch der mittlere Arm senkrecht. Deaktiviert man diese Bewertung, so lernt das Netz. Selbst die Tatsache, daß der Weg unter dem Boden bestraft wird, kann das Netz ausgleichen. Es kann schon nach ca. 5 Generationen die Kugel holen und zum linken Podest heben.

D.11 Datum: 15.06.1999

Darstellung: Die Darstellung mit den Bitmaps sieht zwar sehr professionell aus, ist aber sehr langsam. Deswegen wurde zusätzlich die Möglichkeit eingeräumt, auf Umrißmodus umzuschalten. Der Roboter und die Podeste werden dann nur mit Strichen gezeichnet.

D.12 Datum: 18.06.1999 - 19.06.1999

Fitneßberechnung: Bisher wurde das Netz nur auf eine bestimmte Podesthöhe berechnet. Jetzt werden während die Fitneß eines Individuums berechnet wird zusätzlich, die Podesthöhen verändert. Dies geschieht vor der Berechnung und nachdem ein Ziel erreicht wurde. Die Höhen werden auf Zufallswerte gesetzt. Jedes Individuum hat dann seinen eigenen Spezialfall, für den es gut funktioniert. Wir hoffen nun, daß sich die Individuen bei der Rekombination ihre Spezialfälle austauschen.

Wachstum der Hiddenneuronen: Die Hiddenneuronen sollten durch Mutation wachsen. Die Anzahl der Hiddenneuronen wird dann bei der Fitneßberechnung als negatives Kriterium gewertet. Diese Funktion wurde jedoch wieder deaktiviert, weil es Probleme mit der Speicher-Reallokation gibt.

D.13 Datum: 21.06.1999

Dokumentation auf neuesten Stand bringen und Referat vorbereiten.

D.14 Datum: 23.06.1999 - 24.06.1999

Fitneßfunktion: Das Berechnen der Fitneß mit verschiedenen Höhen, führt dazu, daß die Fitneß stark schwankt. Mit der Rekombination läßt sich dieser Fehler nicht mehr (oder schon gar nicht) ausgleichen. Die Folgerung: Die Fitneß eines Individuums wird jetzt für mehrere Podesthöhen berechnet. Es können 1..10 verschiedene Fälle, das läßt sich durch eine Konstante festlegen, bewertet werden. Zum Schluß wird der Mittelwert verwendet. Außer einer der Werte ist kleiner als ein konsanter Mindestwert. In diesem Fall wird die kleinste Fitneß verwendet (das schwache Glied in der Kette).
Zusätzlich wird jetzt die Fitneß des besten Individuums, vor dem Durchrechnen einer Generation, neuberechnet. Es ergeben sich nämlich durch unterschiedliche Podesthöhen unterschiedliche Fitneßwerte!

Hiddenknoten: Die Anzahl der Hiddenknoten wurde testweise verringert. Es konnten schon mit nur zwei Hiddenneuronen pro Hiddenlayer gute Ergebnisse erziehlt werden.

Rekombination: Die Anzahl der Crossover-Punkte wurde nun fix festgelegt, und nicht mehr mit Zufallsgenerator ermittelt. Es wird vermutet, daß sie ein Vielfaches oder ein Teiler von der Knotenanzahl sein soll, damit ganze Zeilen der Matrix vertauscht werden können. Es werden jetzt eher weniger Cross-Overpunkte verwendet, damit gute Bereiche der Adjazenzmatrix nicht zerrissen werden.

D.15 Datum: 25.06.1999

Rekombination: Letzte Erkenntnis dazu ist, daß es am besten ist, man nimmt gleich viele Cross-over-Punkte, wie man Zeilen (=Spalten = Knotenanzahl) in der Matrix hat.

Mutation: Warum soll die MutationRate für das mittlere Bit gelten ? Es wurde umgebaut, daß sie für Bit 0 gilt. Sie Beträgt als für Bit i:

$$p_{Mi} = p_M \cdot 2^{-i}$$

Literaturverzeichnis

[1] D. D. Olmsted, *History and Principles of Neural Networks.* http://www.neurocomputing.org/history.htm, 1998.

[2] R. Brause, *Neuronale Netze.* isbn 3-519-02247-8 ed., 1991.

[3] P. D. H. Stöcker, *Taschenbuch mathematischer Formeln.* Verlag Harri Deutsch, 4. auflage ed., 1998.

[4] K. S. Jari Vaario, Akira Onitsuka, *Formation of Neural Structures.* ATR Human Information Processing Laboratories, 1998.

[5] N. R. L. Chemometrics Research Group, *Practical Guide to Genetic Algorithm.* http://chem1.nrl.navy.mil/ shaffer/practga.html.

[6] M. Dworschak, *Gefahr für Mann und Maus.* Der Spielgel, 3/1999 ed., 1999.

[7] Dudenverlag, *Lexikon.* ms office 97 ed., 1997.