HUMBOLDT-UNIVERSITÄT ZU BERLIN

MATHEMATISCH-NATURWISSENSCHAFTLICHE FAKULTÄT II
Institut für Informatik

Bachelorarbeit

ZUM ERWERB DES AKADEMISCHEN GRADES
BACHELOR OF ARTS

„Evolution rekurrenter neuronaler Netze mit dem AGE-Verfahren (Analog Genetic Encoding)“

vorgelegt von

Fabian Friedrich
geb. am 28.07.1985 in Bernau

Betreuer: Dr. Manfred Hild
1. Gutachter: Prof. Dr. Hans-Dieter Burkhard
2. Gutachter: Prof. Dr. Bernd-Holger Schlingloff

angefertigt am Lehrstuhl für Künstliche Intelligenz
des Instituts für Informatik

Berlin, September 2008

Erklärung der Selbstständigkeit

Hiermit versichere ich, die vorliegende Bachelorarbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet, sowie Zitate deutlich kenntlich gemacht zu haben.

Berlin, den 16. September 2008

Fabian Friedrich

Danksagungen

Mit Freude bedanke ich mich bei denen, die mich bei der Erstellung dieser Arbeit unterstützt haben.
Vor allem möchte ich mich bei Herrn Dr. Manfred Hild, Herrn Prof. Dr. Hans-Dieter Burkhard und Herrn Prof. Dr. Bernd-Holger Schlingloff, für die Betreuung und Begutachtung dieser Arbeit und die Offenheit für meine Fragen, bedanken.
Dank gilt auch meiner Frau Anne Friedrich und meiner Tochter Josephin, die mir die Arbeitszeit immer erträglich gemacht und mich mit Freude erfüllt haben.
Vielen Dank auch an Herrn Dr. Claudio Mattiussi und Herrn Peter Dürr für die Beantwortung meiner zahlreichen Fragestellungen.
Herrn Christian Thiele und Alexander Häusler gilt mein Dank, für die Hilfe bei der Einarbeitung in die am Lehrstuhl vorhandene Software und die Durchführung der beschriebenen Experimente.
Spezieller Dank für die viele investierte Zeit, die zahlreichen Anregungen und Hinweise und das erfolgreiche Durchstreifen meines orthographischen Dschungels gilt meiner Frau Anne Friedrich, Frau Constanze Reichstein, Herrn Dr. Fritz-Ullrich Schneider, Herrn Fabian Sauer, Herrn Robert Wolff, Herrn Michael Fuchs und Herrn Stephan Otto.

I. Inhaltsverzeichnis

II. Abbildungsverzeichnis

1 Einführung

Analoge Netzwerke und vor allem neuronale Netzwerke sind in der Lage, komplexe Funktionen zu realisieren. Jedoch ist nicht immer bekannt, wie ein Netz, welches die gesuchte Funktionalität erfüllt, strukturell aufzubauen ist. Daher bedient man sich evolutionärer Verfahren, die den unbekannten und oft hochparameterisierten Suchraum unter Zuhilfenahme des Zufalls nach Maxima absuchen. Einer dieser Ansätze ist das Verfahren des „Analog Genetic Encoding". Dieses Verfahren wurde erstmals im März 2005 von Claudio Mattiussi in seiner Doktorarbeit zum Thema „Evolutionary synthesis of analog networks"[1] an der École Polytechnique de Lausanne (EPFL) diskutiert und soll in dieser Arbeit näher untersucht werden. Dieses relativ neue Verfahren stellt eine Alternative zu anderen Ansätzen, wie dem „Genetischen Programmieren"[2] oder „Neuroevolution of Augmented Topologies (NEAT)"[3], dar und orientiert sich dabei stärker am biologischen Vorbild der RNA-Polymerase. Interessant ist dieser Ansatz deshalb, weil auch die Steuerung von Robotern durch neuronale Netze, die meistens nicht einfach zu finden sind, realisiert werden kann. Aufgrund dessen wird im Folgenden mit der Erläuterung dieses Verfahrens begonnen, wobei zahlreiche Beispiele das Verständnis erleichtern. Abschließend werden neuronale Netze, unter anderem zur Bewegungssteuerung eines humanoiden Roboters, evolviert.

2 Überblick Grundlagen

Die folgenden sehr kurzen und durchaus nicht vollständigen Betrachtungen sollen dabei helfen, das AGE-Verfahren besser einzuordnen.

2.1 Neuronale Netze

Künstliche neuronale Netze sind eine abstrahierte Form natürlicher neuronaler Netze, wie sie in den meisten höheren Lebensformen zu finden ist. Sie können zum Beispiel benutzt werden, um komplexe funktionale Zusammenhänge darzustellen oder um Steuerungsaufgaben zu übernehmen.

Dargestellt wird es in Form von mehreren Neuronen, die miteinander verbunden sind. Einige Neuronen fungieren als Ein- und Ausgänge des Netzes und liefern ein festgelegtes Eingangssignal bzw. zeichnen das Ausgangssignal des Netzes auf. Die neuronalen Netze in dieser Arbeit werden in diskreten Zeitschritten simuliert.

Der Verbindung zwischen den Neuronen mit den Indizes i und j wird eine Gewichtung $w_{i,j}$ zugeordnet. Außerdem besitzt jedes Neuron eine eigene Aktivierung x. Diese ergibt sich aus den Aktivierungen aller mit dem Neuron verbundenen Neuronen und deren Gewichtung sowie der Aktivierungsfunktion f des betrachteten Neurons. Die Aktivierung x_i des Neurons i im Zeitschritt t ergibt sich also zu:

$$ {}_t x_i = f\left(\sum_j w_{i,j} \cdot {}_{t-1} x_j\right) $$

Es gibt verschiedene Topologien[4] neuronaler Netze, welche verschiedene Restriktionen für die

1 Übersetzung: "evolutionäre Synthese analoger Netzwerke".
2 Siehe [Koza].
3 Siehe [Stanley].
4 Siehe zum Beispiel [Lund] Seite 17.

entstehenden Verbindungen auferlegen (z. B. Feed-Forward-Netze). In dieser Arbeit wird ausschließlich von rekurrenten neuronalen Netzen ausgegangen, was bedeutet, dass jedes Neuron auch eine Verbindung mit sich selbst besitzt. Weiterhin sind den Verbindungen innerhalb des Netzes keine Beschränkungen auferlegt, so dass jedes Neuron zu jedem anderen Neuron[5] eine Verbindung herstellen kann.

2.2 Simulation neuronaler Netze

Für die Simulation eines neuronalen Netzes benötigt man die Gewichtsmatrix W, in der alle Verbindungsgewichte zwischen den einzelnen Neuronen enthalten sind. Wenn im Netz e Eingangsneuronen, a Ausgangsneuronen und n weitere Neuronen, mit $e, a, n \in \mathbb{N}$, vorhanden sind, hat diese die Dimension $(n+a)\times(n+e)$. Der Vektor der Aktivierungen der einzelnen Neuronen v im Zeitschritt t lässt sich dann wie folgt berechnen:

$$v_t = f(W \cdot v_{t-1})$$

Der so ermittelte Vektor v_t enthält also auch die Werte der Ausgangsneuronen, während v_{t-1} stattdessen das Eingangssignal in Form der Aktivierung der Eingangsneuronen beinhaltet. Detailliertere und weiterführende Informationen sind beispielsweise in [Hild 2008][6] zu finden. Eine exemplarische Gewichtsmatrix und das daraus resultierende neuronale Netz, findet sich in Abbildung 26 bzw. 27.

2.3 Evolution

Bei dem AGE-Verfahren handelt es sich um einen evolutionären Algorithmus, so dass hier kurz dessen charakteristischen Merkmale aufgeführt werden.

Im Rahmen einer Evolution spricht man von einem Satz von Individuen, welche jeweils eine konkrete Ausprägung des gesuchten Parametersatzes darstellen. Alle Individuen zusammengefasst bilden eine Generation. Diese Individuen werden nun auf ihre Tauglichkeit hinsichtlich der Problemlösung bewertet und ihnen wird ein Fitness-Wert zugewiesen, welcher von einer gegebenen Fitnessfunktion abhängt. Durch Rekombination (Crossover) und Mutation einiger der bestehenden Individuen werden nun neue Individuen für die nächste Generation erzeugt. Die Auswahl der Individuen, die als Grundlage für die folgende Generationen dienen, ist dabei meist direkt an den Fitness-Wert gekoppelt, sodass besser geeignete Individuen eine höhere Wahrscheinlichkeit besitzen, selektiert zu werden. Der in der Einführung angesprochene Suchraum wird also mit einem stochastischen Verfahren nach Lösungen durchsucht.

Die wichtigsten Anwendungsgebiete evolutionärer Verfahren sind Optimierungsprobleme, die aufgrund von Nichtlinearitäten oder Diskontinuitäten nicht mithilfe eines traditionellen Verfahrens lösbar sind. Dabei ergeben sich jedoch auch Probleme. So kann beispielsweise nicht sichergestellt werden, dass das globale Optimum überhaupt oder mit einem vordefinierten Zeitaufwand ermittelt werden kann. Weiterhin bedürfen evolutionäre Verfahren meist eines hohen rechentechnischen Aufwandes.

5 Außer den Ein- und Ausgangsneuronen aufgrund ihrer Sonderstellung.

6 Auf die dort eingeräumte Möglichkeit, ein Ausgangsneuron mit allen anderen zu verbinden, wurde hier jedoch verzichtet, was aber keine Einschränkung für die Funktionalität des Netzes bedeutet.

3 Analog Genetic Encoding

Das Verfahren des "Analog Genetic Encoding" abstrahiert biologische Prozesse. Die Grundlage der Evolution besteht darin, dass nicht die Gewichte eines neuronalen Netzes oder seine Struktur (der Phänotyp) direkt verändert werden, sondern zuerst das neuronale Netz kodiert in einem Genom (der Genotyp) vorliegt und auf diesem Veränderungen ausgeführt werden. Das besondere dabei ist, dass durch die genetischen Veränderungen sowohl kleinere Anpassungen an den einzelnen Verbindungsgewichten als auch strukturelle Veränderungen an dem Netz entstehen können. Das folgende Kapitel soll das Kodierungsverfahren und alle für eine Evolution nötigen Rahmenbedingungen erläutern und ist an die Ausführungen aus [Mattiussi 2005] angelehnt.

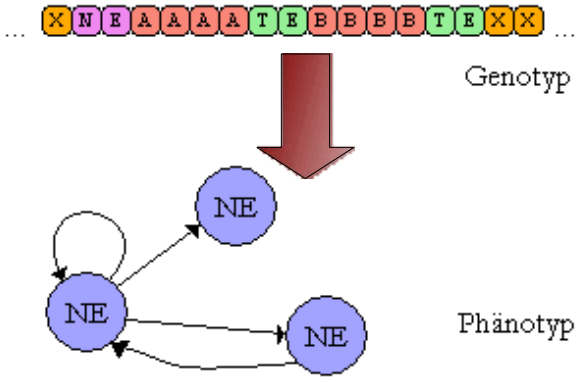

Abbildung 1: Symbolische Darstellung der Abbildung eines Genotyps auf einen Phänotyp.

3.1 Das Genom

Ein Genom besteht aus einer oder mehreren Zeichenketten, die auch Chromosome genannt werden. Die einzelnen Zeichen innerhalb der Chromosome werden als Nukleotide bezeichnet. Ein Nukleotid entspricht einem Zeichen aus einem vorher definierten Alphabet.

3.1.3 Das Alphabet

Analog zu [Mattiussi 2005] wird das Alphabet mit $\mathcal{G}$ und die Anzahl der Elemente mit $|\mathcal{G}|$ bezeichnet. Für die folgenden Beispiele werden die 26 Großbuchstaben des ASCII-Zeichensatzes als Alphabet angenommen.

3.1.4 Terminale und Token

Innerhalb dieses Genoms müssen nun bestimmte Bereiche festgelegt werden, aus denen die unterschiedlichen Einheiten des resultierenden Netzes extrahiert werden können. Dazu legt der Anwender einen Geräte-/Einheitensatz („Device-Set") fest. Der Beginn einer Einheit im Genom wird mit einem Token angezeigt. Im Fall eines Neurons könnte dies z. B. „NE" oder „NEUR"

sein.[7] Zusätzlich werden noch beliebig viele, mindestens jedoch ein Terminal festgelegt. Das Terminal zeigt das Ende einer Zeichenkette an. Für ein Neuron könnten zwei Terminale „TE“ festgelegt werden und die beiden Zeichenketten, die nun vom Token und den beiden Terminalen umschlossen werden, definieren den für das Neuron notwendigen Ein- und Ausgang. Wenn eine Einheit auf diese Weise vollständig kodiert vorliegt, wird der betreffende Teil des Genoms als Gen bezeichnet. So können für unterschiedliche Anwendungszwecke verschiedene Einheitensätze verwendet werden. Dadurch können auch andere Netzwerke wie z. B. elektronische Schaltkreise kodiert werden.

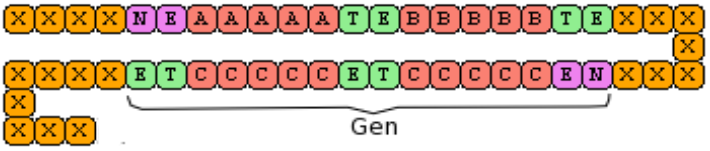

Abbildung 2: Ein Beispiel Genom in dem innerhalb eines Chromosoms zwei Neuronen kodiert sind. Das Token ist in diesem Fall „NE“ und es sind 2 Terminale „TE" enthalten. Der restliche Teil des Genoms wird nicht benutzt und kodiert nichts.

3.2 Extraktion

Um die einzelnen Einheiten zu extrahieren wird das Genom vom ersten Zeichen ausgehend durchsucht. Wird dabei ein Einheiten-Token gefunden, versucht man von seiner Position ausgehend alle benötigten Terminale zu finden, bevor ein weiteres Token im Genom erscheint oder das Ende des Chromosoms erreicht wird. Dies bedeutet, dass ein Überlappen mehrerer Gene nicht möglich ist.

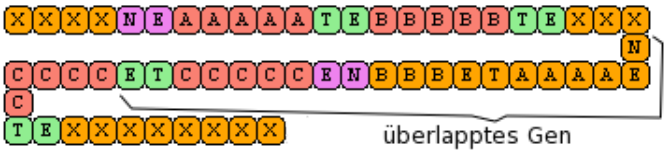

Abbildung 3: Ein überlapptes Gen, das nicht dekodiert wird.

Wenn alle benötigten Elemente gefunden wurden, wird daraus die entsprechende Einheit erstellt und ihren Ein- und Ausgängen werden die entsprechenden Zeichensequenzen zugeordnet. In dem Beispiel der Abbildung 3 entstehen zwei Neuronen.

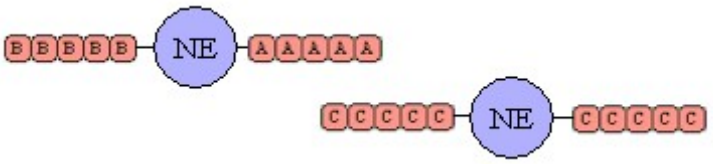

Abbildung 4: Aus dem Genom in Abbildung 3 dekodierte Neuronen.

7 Die Wahl eines Tokens ist komplett freigestellt, es bietet sich jedoch zum Zwecke der besseren Lesbarkeit an, Abkürzungen der entsprechenden Einheit zu verwenden.

3.3 Verbinden der Neuronen

Nun müssen aus den gegebenen Zeichenketten Verbindungen geschaffen werden. Im Falle eines künstlichen neuronalen Netzes ist dies ein reeller Wert. Dazu bedient sich das Verfahren einer Abbildungsfunktion ("Device Interaction Map"). Da die benötigten Verbindungsgewichte sehr stark von der Art des zu extrahierenden Netzes abhängen, erfolgt die Ermittlung der Verbindungsstärke in zwei Stufen. Dazu gibt es eine Abbildungsfunktion $L(s_1, s_2)$, welche zuerst die gegebenen Zeichenketten s_1 und s_2 in eine natürliche Zahl umwandelt. Dazu wird der in [Gusfield] beschriebene Algorithmus der lokalen Ausrichtung von Zeichenketten ("local string alignment") verwendet[8]. Auf diesen Wert kann dann eine netzwerkspezifische Abbildungsfunktion $N(L(s_1, s_2))$ angewendet werden. Der Vorteil dessen ist, dass der Anwender sich nicht um die Interna der Abbildung von zwei Zeichenketten kümmern muss, sondern nur eine geeignete Abbildung $\mathbb{N} \rightarrow \mathbb{R}$ zu finden hat. Für den Fall eines neuronalen Netzes ergibt sich also eine Gewichtsmatrix W mit der Dimension $(n \times n)$, wenn man die Anzahl der entstandenen Neuronen mit n bezeichnet.

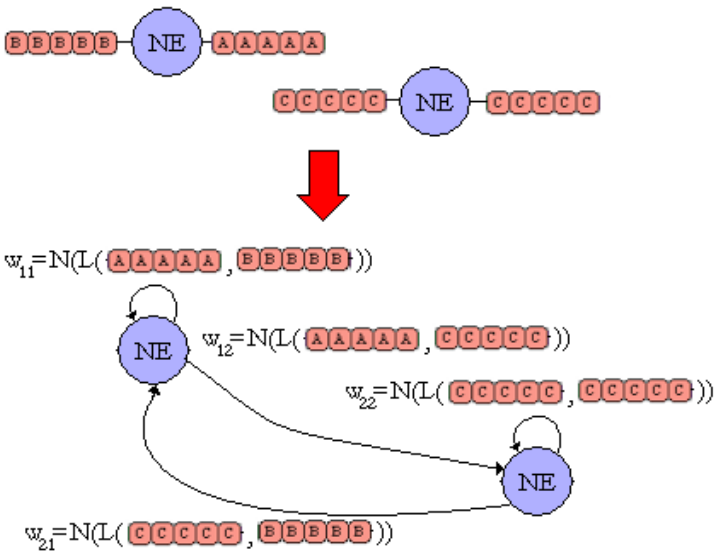

Abbildung 5: Verbinden zweier dekodierter Neuronen.

Die Einträge W_{ij} der Matrix werden durch die Funktion $N(L(s_{i1}, s_{j0}))$ festgelegt, wobei s_{ab} die Zeichenkette bezeichnet, die dem a-ten Neuron als b-tes Terminal zugewiesen wurde. Dieses Beispiel geht davon aus, dass das erste Terminal für den Ausgang fungiert, während das zweite dem Eingang zugeordnet wurde. Die daraus resultierende Gewichtsmatrix kann direkt für die Simulation des Netzes genutzt werden. In der Diagonalen findet sich die Stärke der Selbstbindung bzw. der Rekurrenz.

8 Dieser wird tiefergehend im Abschnitt „Lokale Ausrichtung (local alignment) von Zeichenketten" erläutert.

3.4 Netzwerkspezifische Abbildungsfunktion

Die Aufgabe der netzwerkspezifischen Abbildungsfunktion ist, den abstrakten Ausrichtungswert in eine Verbindungsstärke umzuwandeln. Der Raum der Ausrichtungswerte ist abzählbar und - sollte die Länge der Zeichenketten begrenzt sein - sogar endlich. Dies schränkt den Erfolg der Evolution jedoch nicht ein, wenn die Abbildungsfunktion sinnvoll aufgestellt wird. In [Mattiussi 2005] wird das Problem einer sinnvollen Aufteilung eines überabzählbaren Raums in diskrete Werte dezidierter behandelt. Für ein neuronales Netz bietet sich eine logarithmische Verteilung der Verbindungswerte an, da vor allem im niedrigen Wertebereich kleine Abstufungen sinnvoll sind. Bei der Verwendung einer logarithmischen Aufteilung der Werte wird zum Dekodieren eines diskretisierten Wertes eine exponentielle Funktion benötigt. Um diese Funktion festzulegen, müssen zunächst einige Parameter festgelegt werden:

1. Eine minimale und maximale Verbindungsstärke x_{min}/x_{max}
2. Ein minimaler und maximaler Ausrichtungswert i_{min}/i_{max}

Daraus ergibt sich die Anzahl der unterschiedlichen Verbindungsstärken, die durch die Funktion dargestellt werden können $N = i_{max} - i_{min}$.
Die Funktion zur Umwandelung eines Ausrichtungs-Wertes in eine Verbindungsstärke lautet dann:

$$x(i) = x_{min} \cdot \beta^{i - i_{min}} = x_{min} \cdot \left(\frac{x_{max}}{x_{min}} \right)^{\frac{i - i_{min}}{N-1}}$$

Für das erste Experiment (XOR-Funktion) weiter unten ergibt sich:

- $x_{max} = 1000$
- $x_{min} = 0.001$
- $N = 37$

$$\beta = \left(\frac{1000}{0.0001} \right)^{37-1} = \sim 1{,}48$$

Dabei werden sechs Dekaden mit jeweils sechs Werten abgedeckt. Wenn nun noch $i_{min} = 1$ und $i_{max} = 37$ angenommen wird, ergibt sich folgende Abbildungsfunktion:

1	0,0010000	14	0,1467799	27	21,5443469
2	0,0014678	15	0,2154435	28	31,6227766
3	0,0021544	16	0,3162278	29	46,4158883
4	0,0031623	17	0,4641589	30	68,1292069
5	0,0046416	18	0,6812921	31	100,0000000
6	0,0068129	19	1,0000000	32	146,7799268
7	0,0100000	20	1,4677993	33	215,4434690
8	0,0146780	21	2,1544347	34	316,2277660
9	0,0215443	22	3,1622777	35	464,1588834
10	0,0316228	23	4,6415888	36	681,2920691
11	0,0464159	24	6,8129207	37	1.000,0000000
12	0,0681292	25	10,0000000		
13	0,1000000	26	14,6779927		

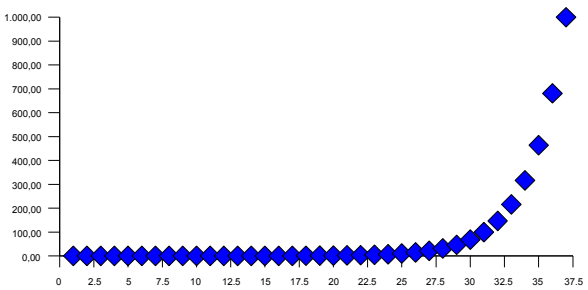

Abbildung 6: Grafische Darstellung der Dekodier-Funktion.

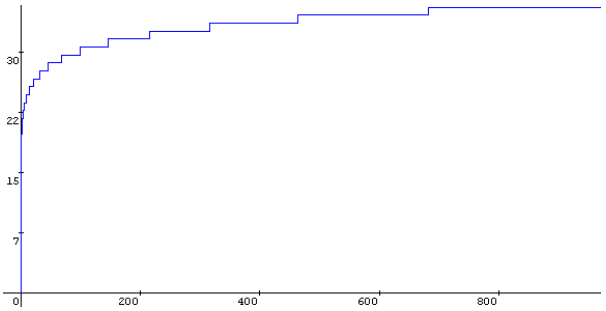

Abbildung 7: Verteilung der Werte (Anzahl der Werte bis zur gegebenen Grenze).

3.4.1 Verbindungsunterdrückung

In einem neuronalen Netz ist es jedoch nicht immer erwünscht, eine Verbindung zwischen allen Neuronen zu generieren. Eine Unterdrückung der Verbindung kann dadurch erreicht werden, dass der Wert für i_{min} höher angesetzt wird. Dies erscheint sinnvoll, da die lokale Ausrichtung von Zeichenketten immer einen positiven Wert erzeugen wird, auch wenn beide Zeichenketten vollkommen zufällig gewählt wurden. Die Bestimmung eines geeigneten Werts für i_{min} ist jedoch nicht trivial, da der Ausrichtungs-Wert von der Länge der betrachteten Zeichenketten abhängt. Eine Möglichkeit diesen Wert zu bestimmen, ist die Wahrscheinlichkeitsverteilung der Ausrichtungs-Werte für zufällig erzeugte Zeichenketten zu betrachten[9]. Man beginnt einfach damit, dass man $i_{min}=0$ und $i_{max}=i_{min}+N$ setzt. Die Länge einer Zeichenkette, welche einen Ausrichtungs-Wert von i_{max} erreichen kann, ist dann durch den größtmöglichen Substitutionswert in der Substitutionsmatrix c_{max} definiert. Daraufhin wird die Länge durch $l=i_{max}/c_{max}$ bestimmt. Nun kann die Wahrscheinlichkeitsverteilung der Ausrichtungs-Werte für zufällige Zeichenketten erzeugt werden und dadurch eine bessere Schätzung für i_{min} gewonnen werden. Dieser Prozess wird solange wiederholt, bis sich ein stabiler Wert für i_{min} ergibt. Wenn die sich ergebenden Werte nicht zu groß sind, vor allem in Betrachtung der benötigten Rechenzeit für sehr lange Zeichenketten, können die Größe des Alphabets, die Substitutionsmatrix und der Wert für i_{min} verwendet werden.

9 Verfahren entnommen aus [Mattiussi 2005] Kapitel 3.4.5.

Als Beispiel soll die Substitutionsmatrix aus Abbildung 14 dienen. Nach mehreren Iterationen ergibt sich ein Wert von $i_{min}=20$, bei einer Länge der Zeichenkette $l=10$ wird als Grenze eine Wahrscheinlichkeitsdichte von 99% angegeben. Dies bedeutet, dass bei zufällig erzeugten Zeichenketten $i_{min}/c_{max}=4$ Zeichen übereinstimmen und daher nicht in Betracht gezogen werden sollten. Diese Werte sind vertretbar für den auf Seite 13 beschriebenen Algorithmus. Die Ursache für die geringen Werte ist der geringe Wert für $N=36$.

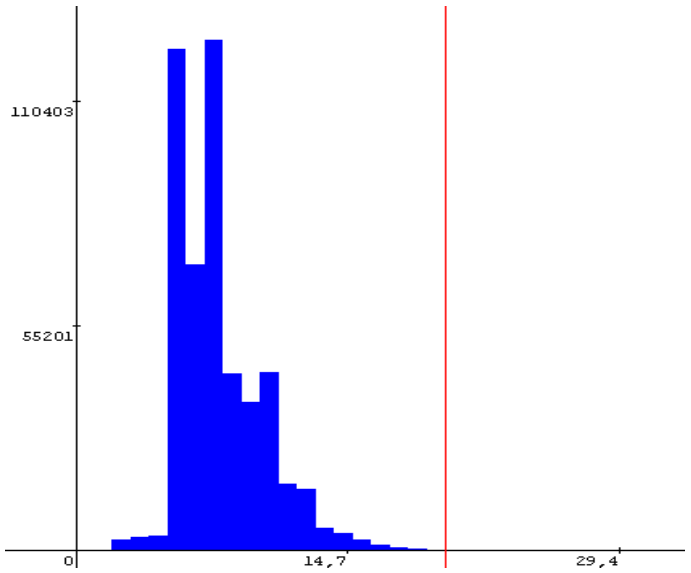

Abbildung 8: Häufigkeitsverteilung für 500.000 zufällig erzeugte Zeichenketten. Die Bimodalität der Verteilung resultiert aus der Gestalt der Substitutionsmatrix in Verbindung mit der geringen Länge der Zeichenketten (10), welche bedingt, dass bestimmte Werte seltener auftreten. Die rote Linie markiert den Wert, ab dem eine Wahrscheinlichkeitsdichte von 99% erreicht wird.

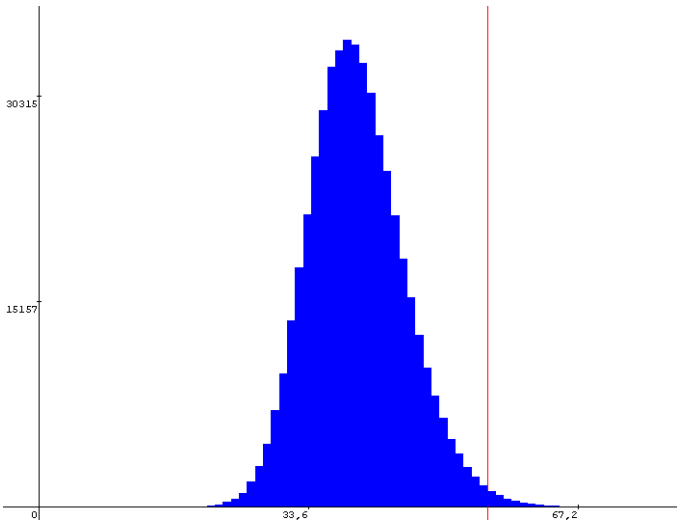

Abbildung 9: Verteilung der Ausrichtungs-Werte für 500.000 zufällig erzeugte Zeichenketten, wobei das Alphabet auf die Größe 12 reduziert wurde.

Wird stattdessen $N=100$ gesetzt, ergeben sich nach einigen Iterationen die Werte $i_{min}=23$ und $l=24$, welche auch akzeptabel sind. Wenn nun aber die Größe des Alphabets auf $|G|=12$ gesetzt wird, erhöht sich die Anzahl der zufälligen Übereinstimmungen beträchtlich. (siehe Abbildung 9) Daraus ergibt sich ein Wert für $i_{min}=56$ und $l=31$, wodurch die Wahl dieses kleinen Alphabets weniger ansprechend erscheint.

3.5 Ein- und Ausgänge

In der Arbeit [Mattiussi 2005] wurden verschiedene Möglichkeiten diskutiert, um das entstandene Netz mit den benötigten Ein- und Ausgängen zu verbinden. Der für den Anwender am einfachsten zu handhabende Ansatz ist wahrscheinlich der eines IO-Ports. Um eine Verbindung mit den vorher festgelegten Ein- und Ausgängen herzustellen, werden dem Einheitensatz neue Einheiten mit einem einmaligen Token und einem Terminal zugewiesen, welche diesen IO-Port repräsentieren. Der Ein- respektive Ausgang kann wie ein Neuron mit allen anderen Elementen verbunden werden.

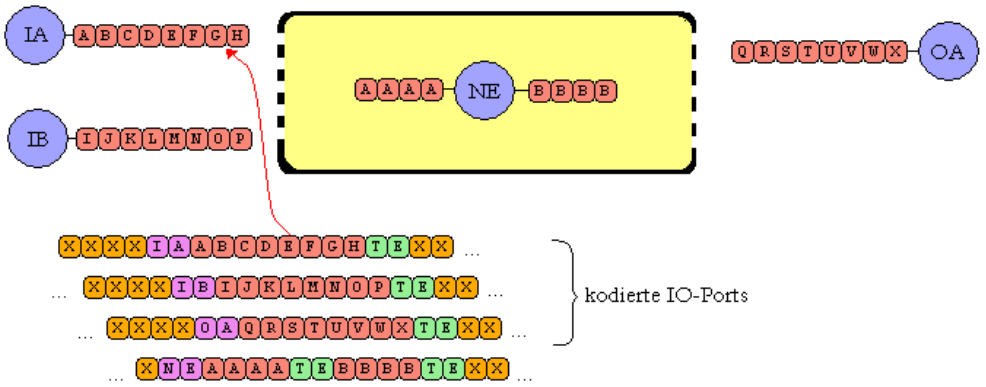

Abbildung 10: Beispielgenom aus dem zwei Eingänge und ein Ausgang dekodiert wurden.

Dabei ist es auch möglich, dass ein Ein- bzw. Ausgang mehrmals innerhalb des Genoms vorkommt. Ist dies der Fall, werden die berechneten Verbindungsgewichte addiert. Um das zu realisieren, kann man alle IO-Ports und Neuronen dekodieren und die berechneten Verbindungsgewichte in einer Matrix, die auch für die spätere Simulation genutzt werden kann, speichern. Danach können alle Zeilen und Spalten, welche doppelt auftreten, einfach addiert werden[10].

Eine Alternative dazu wäre, die Simulation mit der vollständigen Gewichtsmatrix durchzuführen und die eingehenden Eingangswerte an einem Ausgang aufzusummieren und erst nach vollständiger Berechnung eines Schrittes die neue Aktivierung zu kalkulieren. Da jedoch im Allgemeinen viele Simulationsschritte berechnet werden müssen, ist dieses Verfahren ineffizient, weil in jedem Schritt mehrere Additionen und Multiplikationen erforderlich sind.

10 Interessanterweise scheint es hinderlich für das Ergebnis der Evolution zu sein, die doppelt vorkommenden IO-Ports zu ignorieren.

	IA	IA	IB	IB	NE
OA	0,50	0,10	0,30	0,25	0,25
OA	0,10	0,05	0,10	0,25	0,25
OA	0,20	0,05	0,05	0,05	0,50
NE	0,20	0,80	0,70	0,30	1,00

	IA	IB	NE
OA	1	1	1
NE	1	1	1

Abbildung 11: Beispiel einer Gewichtsmatrix mit mehreren Ein- und Ausgangsgewichten

3.6 Genetische Operatoren

Das bisher beschriebene Modell für das Genom erlaubt es, verschiedene genetische Operatoren durchzuführen, ohne dass das Genom dadurch undekodierbar wird. Die einzige Grenze, welche definiert werden sollte, ist die der Länge der Zeichenketten, die einer Einheit zugeordnet werden können, um sicherzustellen, dass der Algorithmus für die lokale Ausrichtung mit sinnvollem Zeit- und Platzverbrauch ausgeführt werden kann. Die genetischen Operatoren können auf unterschiedliche Komponenten des Genoms angewendet werden:

- Operationen auf einem einzelnen Nukleotid
- Operationen auf einem Chromosom-Fragment
- Operationen auf einem gesamten Chromosom
- Operationen die das gesamte Genom betreffen

Ein genetischer Operator betrachtet das Genom lediglich als Zeichenkette, in der die Token, Terminale und IO-Ports keine besondere Bedeutung haben, was bedeutet, dass die Token und Terminale nicht in irgendeiner Weise geschützt sind und durch eine genetische Operation auch invalidiert werden können.

3.6.1 Nukleotid-basierte Operatoren

Einfügen eines Nukleotids
Mit der Wahrscheinlichkeit p_{ni} wird zwischen je zwei bereits vorhandenen Nukleotiden ein neues zufälliges Nukleotid aus dem Alphabet eingefügt.

Löschen eines Nukleotids
Mit der Wahrscheinlichkeit p_{nd} besteht die Möglichkeit, dass ein Nukleotid aus dem Genom gelöscht wird.

Substitution eines Nukleotids
Mit der Wahrscheinlichkeit p_{ns} wird ein Nukleotid innerhalb des Genoms durch ein anderes, zufällig gewähltes Nukleotid des Alphabets ersetzt.

3.6.2 Chromosom-Fragment-basierte Operatoren

Ein Chromosom-Fragment wird bestimmt, indem in einem Genom ein zufälliges Chromosom gewählt und zwei Indizes gezogen werden. Die Zeichenkette zwischen diesen beiden Indizes bildet das Chromosom-Fragment. Mögliche Operationen darauf sind:

Chromosom-Fragment-Verdopplung
Mit der Wahrscheinlichkeit p_{f2} wird das gewählte Chromosom-Fragment kopiert und an einer zufällig gewählten Stelle, in einem zufällig gewählten Chromosom innerhalb des Genoms wieder eingefügt.

Komplementierte Chromosom-Fragment-Verdopplung
Mit der Wahrscheinlichkeit p_{fc} wird zu jedem Nukleotid innerhalb des gezogenen Chromosom-Fragmentes das Komplement gebildet, indem z. B. das Nukleotid mit der größten Distanz aus dem Alphabet entnommen wird. Dieses komplementierte Chromosom-Fragment wird dann an eine zufällige Stelle in einem zufällig gewählten Chromosom eingefügt.

Chromosom-Fragment-Transposition
Mit der Wahrscheinlichkeit p_{ft} wird das gezogene Chromosom-Fragment von seiner ursprünglichen Stelle entfernt und dann an einer zufällig gewählten Stelle in einem zufällig gewählten Chromosom wieder eingefügt.

Löschen eines Chromosom-Fragmentes
Mit der Wahrscheinlichkeit p_{fd} wird das gewählte Chromosom-Fragment aus dem Chromosom entfernt.

Einfügen einer Einheit
Mit der Wahrscheinlichkeit p_{di} wird der genetische Code einer Einheit aus dem Einheitensatz erzeugt und an einer zufällig gewählten Stelle in einem zufällig gewählten Chromosom eingefügt.

3.6.3 Chromosom-basierte Operatoren

Chromosom-Verdopplung
Jedes Chromosom kann mit der Wahrscheinlichkeit p_{c2} verdoppelt werden und wird als neues Chromosom dem Genom hinzugefügt.

Löschen eines Chromosomes
Jedes Chromosom kann mit der Wahrscheinlichkeit p_{cd} aus dem Genom entfernt werden.

Crossover Operator
Mit der Wahrscheinlichkeit p_{cx} findet eine Crossover-Operation statt. Dafür wird zu dem Individuum, auf das der Operator angewendet werden soll, ein Partner ermittelt. Wenn beide Partner dieselbe Anzahl an Chromosomen aufweisen, kann die Operation durchgeführt werden. Dazu wird in je zwei Chromosomen ein Index gewählt und die so ermittelten Chromosom-Fragmente werden zwischen den Chromosomen ausgetauscht.
Dabei kann der Index rein zufällig gewählt werden, was jedoch dazu führt, dass die Änderung im Genom sehr stark ausfiele[11]. Ein anderer Ansatz ist der, im ersten Chromosom einen zufälligen Index zu bestimmen und die Zeichenkette, die sich in direkter Nachbarschaft zu diesem Index befindet, in dem anderen Chromosom entweder exakt oder mit einem Ähnlichkeitsmaß[12] zu suchen und wenn sie dort gefunden wird, den entsprechenden Index zu verwenden.

11 Siehe [Mattiussi 2005] respektive [Harvey].
12 Dafür würden sich zum Beispiel die "Edit Distance" oder "Global String Alignment" wie beschrieben in [Gusfield] anbieten.

3.6.4 Genom-basierte Operatoren

Genom-Verdopplung
Das ganze Genom wird mit der Wahrscheinlichkeit p_{g2} verdoppelt. Dabei werden die vorhandenen Chromosomen dupliziert und dem Genom ein zweites Mal hinzugefügt.

Genom-Verkürzung
Durch die bisher beschriebenen Operatoren können Token und Terminale innerhalb des Genoms invalidiert werden und die entsprechenden Nukleotiden werden nicht mehr verwendet. Dieser nicht verwendete Teil des Genoms kann eine wichtige Rolle in der Evolution spielen und behindert das Dekodieren des Netzes nicht. Dennoch erscheint es sinnvoll, das Genom bei Bedarf in regelmäßigen Abständen zu verkürzen. Deshalb kann ein Genom mit der Wahrscheinlichkeit p_{gt} von allen, bis auf eine vorher festgelegte Mindestgrenze, nicht benutzten Nukleotiden befreit werden. Dies trifft auch auf die gesamte Population mit einer Wahrscheinlichkeit von p_{pt} bei jedem Generationswechsel zu.

3.7 Generationswechsel

Nachdem alle Individuen in einer Population dekodiert und bewertet wurden, werden diese Individuen dazu benutzt, eine neue Generation zu erstellen. Dabei werden die besten Individuen der Generation unverändert in die neue Generation übernommen. Die Anzahl dieser Individuen wird Elite genannt.
Um die restlichen Individuen zu bestimmen, bietet sich das Verfahren der „Tournament Selection“ wie beschrieben in [Bäck] und analysiert in [Blickle/Thiele] an. Dabei werden aus der bestehenden Population eine bestimmte Anzahl an Individuen, die „Tournament Size“ genannt wird, zufällig gezogen. Das Individuum mit dem höchsten Fitness-Wert wird dann in eine Zwischengeneration übernommen. Selbstverständlich kann das selbe Individuum mehrmals selektiert werden. Auf alle Individuen dieser Zwischengeneration werden dann die genetischen Operatoren angewendet, um ihre Eigenschaften zu verändern und sie werden anschließend in die neue Generation übertragen.

4 Lokale Ausrichtung (local alignment) von Zeichenketten

4 Lokale Ausrichtung (local alignment) von Zeichenketten

Wie bereits erwähnt, bietet sich das Verfahren des "local alignment", beschrieben in [Gusfield], an, um aus zwei Zeichenketten einen Wert für die Weiterverarbeitung zu einer Verbindung zwischen zwei Neuronen herzustellen. Bei der lokalen Ausrichtung von Zeichenketten ist es das Ziel, die Teilzeichenketten zu finden, deren Ähnlichkeit (bzw. globaler Ausrichtungs-Wert) unter allen möglichen Teilzeichenketten optimal ist. Von den beiden gegebenen Zeichenketten S_1 und S_2 können also frei wählbare Prä- und Suffixe ausgespart werden. Es kann auf alle Zeichen verzichtet werden, der sich ergebende Ausrichtungs-Wert („Alignment Score") ist dann 0.

```
AB_ENTEU_ERIN
ADVENT_URE
```

Abbildung 12: Beispiel einer lokalen Ausrichtung der Zeichenketten "ABENTEUERIN" und "ADVENTURE".

In dem Beispiel aus Abbildung 12 wird der Suffix „RIN" der ersten Zeichenkette nicht verwendet, da er keinen positiven Beitrag zum Ausrichtungs-Wert leisten konnte.
Um beide Zeichenketten aneinander auszurichten, gibt es drei mögliche Operationen:

1. Die beiden sich gegenüberstehenden Zeichen sind gleich (match). A <> A
2. Sie sind nicht gleich (mismatch). A <> B
3. Ein Zeichen wird gelöscht/eingefügt (Insertion/Deletion). A <> _

Wenn die dritte Operation angewendet wird, so wird dem Zeichen das Element _ gegenübergestellt.

4.1 Substitutionsmatrix

Zur Quantifizierung der einzelnen Operationen wird eine Substitutionsmatrix angegeben. Diese Matrix hat die Dimension $(|G|\times|G+1|)$, wobei die letzte Spalte[13] der Matrix die Werte für eine Einfüge- bzw. Löschoperation angibt. Eine beispielhafte Matrix für das Alphabet A = {A,B,C,D} ist in Abbildung 13 dargestellt. Diese Matrix ist nicht nur symmetrisch sondern auch zirkulierend, was bedeutet, dass alle Elemente in den Diagonalen gleich sind. Bei dieser Form reicht es aus, die erste Zeile einer Matrix anzugeben. Bei der Konstruktion einer Substitutionsmatrix sollten folgende Eigenschaften beachtet werden:

- Der Wert zweier Löschoperationen sollte nie niedriger sein als der einer Substitution. Die Substitution wird sonst nie angewendet, da das gegenseitige Löschen der Zeichen einen höheren Wert für die Ausrichtung ergibt.
- Negative Werte sollten überwiegen, da die Ausrichtung sonst sehr viele - bzw. wenn alle Werte positiv sind alle – Zeichen mit einbezieht und die Ausrichtung dann eher global als lokal ausfällt.

$$\begin{pmatrix} & A & B & C & D & \_ \\ A & 2 & 0 & -2 & 0 & -2 \\ B & 0 & 2 & 0 & -2 & -2 \\ C & -2 & 0 & 2 & 0 & -2 \\ D & 0 & -2 & 0 & 2 & -2 \end{pmatrix}$$

Abbildung 13: Beispielhafte Substitutionsmatrix.

13 Es ist natürlich freigestellt, auch die letzte Zeile zu benutzen oder den Vektor für Löschoperationen separat zu behandeln.

4.2 Algorithmus

Analog zu [Gusfield] wird zuerst von dem Problem der lokalen Suffixausrichtung ausgegangen[14]. Seien nun die beiden zu untersuchenden Zeichenketten S_1 und S_2 mit der Länge n bzw. m und außerdem zwei Indizes $i \le n$ und $j \le m$ gegeben. Das Problem der lokalen Suffixausrichtung besteht nun darin, zwei Suffixe, die auch leer sein können, $\alpha = S_1[1..i]$ und $\beta = S_2[1..j]$, zu finden, für die der Wert $V(\alpha, \beta)$ für alle möglichen α und β maximal wird. $v(i, j)$ bezeichnet dabei den Wert der optimalen Suffixausrichtung für die Indizes i und j. Da bei der lokalen Suffixausrichtung der maximale Wert über alle Suffixe gesucht wird, lässt sich zeigen, dass der Wert v^*, welcher den Wert der optimalen lokalen Zeichenkettenausrichtung angibt, gleich $max[v(i, j) : i \le n, j \le m]$ und die beiden Probleme gleich sind[15].

$max[v(i, j) : i \le n, j \le m]$ lässt sich wiederum einfach rekursiv berechnen mit der folgenden Rekursion und den dazugehörigen Abbruchbedingungen:

- $v(i, 0) = 0$
- $v(0, j) = 0$
- $$v(i, j) = max[\ 0,\ v(i-1, j-1) + s(S_1(i), S_2(j)),\ v(i-1, j) + s(S_1(i), \_),\ v(i, j-1) + s(\_, S_2(j))]$$

$S_1(i)$ gibt dabei den i-ten Buchstaben aus der Zeichenkette S_1 an und $s(C, D)$ bestimmt den Substitutionswert für die Buchstaben 'C' und 'D' aus der Substitutionsmatrix.

Würde man diese Rekursion direkt implementieren, wäre die Laufzeit im exponentiellen Bereich. Das liegt daran, dass viele Werte mehrfach berechnet werden würden. Um dies zu verhindern bedient man sich des dynamischen Programmierens. Dabei werden die errechneten Zwischenwerte in einer Tabelle gespeichert und müssen so nicht mehrfach berechnet werde. Die Tabelle wird dann vom kleinsten Element ausgehend entweder reihen- oder spaltenweise gefüllt. Da die Tabelle $n \times m$ Felder hat und für die Berechnung des Wertes in einem Feld nur 4 Vergleiche und 3 Additionen benötigt werden, kann die gesamte Tabelle also in $O(nm)$ Zeit gefüllt werden.

Dies soll nun anhand des Beispiels aus Abbildung 12 verdeutlicht werden. Als Substitutionsmatrix dient dabei die folgende zirkulierende Matrix:

14 Siehe [Gusfield] Seite 230.

15 Das $v^* \ge max[v(i, j) : i \le n, j \le m]$ gilt ist klar, da die optimale Lösung des Suffixausrichtungsproblems auch eine mögliche Lösung für das lokale Ausrichtungsproblem ist. Seien nun jedoch α^* und β^* die optimale Lösung des lokalen Ausrichtungsproblems und nehmen wir an, dass α^* an der Position i^* und β^* an der Position j^* endet, dann sind α^* und β^* auch eine Lösung für die Suffixausrichtung für die Indizes i^* und j^*, was wiederum bedeutet das $v^* \le v(i^*, j^*) \le max[v(i, j) : i \le n, j \le m]$.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	_
A	5	2	1	0	-1	-3	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-3	-1	0	1	2	-3

Abbildung 14: Beispiel für eine Substitutionsmatrix[16].

	A	B	E	N	T	E	U	E	R	I	N
A	+5	+2	0	0	0	0	0	0	0	0	0
D											
V											
E											
N											
T											
U											
R											
E											

Abbildung 15: dynamische Programmiertabelle Füllen der ersten Zeile.

	A	B	E	N	T	E	U	E	R	I	N
A	5	2	0	0	0	0	0	0	0	0	0
D	2	6	4	1	0	2	0	2	0	0	0
V	0	3	1	0	2	0	4	1	1	0	0
E	0	0	8	5	2	7	4	9	6	3	0
N	0	0	5	13	10	7	4	6	8	5	8
T	0	0	2	10	18	15	12	9	7	4	5
U	0	0	0	7	15	13	20	17	14	11	8
R	0	0	0	4	12	10	17	15	22	19	16
E	0	0	5	2	9	17	14	22	19	21	18

Abbildung 16: dynamische Programmiertabelle (ausgefüllt).

Abbildung 15 verdeutlicht, wie die erste Zeile der dynamischen Programmiertabelle gefüllt wird. In der rot markierten Zelle ergibt sich ein Wert von fünf, da 'A' und 'A' übereinstimmen. Die Zelle daneben erhält den Wert 2, da entweder 'A' gegen 'B' substituiert oder der Wert in der Zelle links (5) zuzüglich einer Löschoperation verwendet werden kann.
Während die Tabelle gefüllt wird, kann gleichzeitig der maximale Wert in der Tabelle gespeichert werden. So ist nach einem Durchlauf bereits die Zelle mit dem maximalen Wert bekannt, sodass kein weiterer rechentechnischer Aufwand erforderlich wird, diese zu ermitteln.
Um die Ausrichtung der beiden Zeichenketten wieder zu rekonstruieren, müssen außerdem Informationen über die Herkunft des Zellwerts gespeichert werden. In Abbildung 18 ist dies grafisch in Form von kleinen Pfeilen dargestellt[17]. Folgt man nun von einer Zelle mit dem maximalen Wert aus diesen Pfeilen, bis man eine Zelle mit dem Wert Null oder das Ende der Tabelle erreicht, hat man eine mögliche Ausrichtung gefunden. Dabei steht ein diagonaler Pfeil für eine Substitution der Buchstaben, ein horizontaler Pfeil für ein Löschen des Buchstabens aus der ersten Zeichenkette S_1 und ein vertikaler Pfeil für ein Löschen eines Buchstabens aus der zweiten Zeichenkette S_2.

16 Quelle [Mattiussi 2005].
17 Softwaretechnisch bietet sich hier z. B. eine zweite Tabelle an, in der diese „Pfeile“ kodiert gespeichert werden. So könnte man eine 1, 2 und 4 für die unterschiedlichen Pfeilrichtungen verwenden und dementsprechend mehrere Pfeile durch Addition dieser Werte darstellen.

	A	B	E	N	T	E	U	E	R	I	N
A	5	2	0	0	0	0	0	0	0	0	0
D	2	6	4	1	0	2	0	2	0	0	0
V	0	3	1	0	2	0	4	1	1	0	0
E	0	0	8	5	2	7	4	9	6	3	0
N	0	0	5	13	10	7	4	6	8	5	8
T	0	0	2	10	18	15	12	9	7	4	5
U	0	0	0	7	15	13	20	17	14	11	8
R	0	0	0	4	12	10	17	15	22	19	16
E	0	0	5	2	9	17	14	22	19	21	18

Abbildung 17: Herkunft der Zellwertberechnungen.

In Abbildung 17 ist der Weg eingezeichnet, welcher zu der Ausrichtung in Abbildung 12 führte. Alternativ könnte auch von der anderen markierten Zelle ausgehend eine Ausrichtung konstruiert werden.

```
AB_ENTEUERIN
ADVENT_U_R  E
```

Abbildung 18: Alternative Ausrichtung der Zeichenketten aus Abbildung 12.

Das Verfahren der lokalen Ausrichtung von Zeichenketten in Kombination mit der netzwerkspezifischen Abbildungsfunktion, wie in Kapitel 3.4 beschrieben, bietet sich deshalb an, weil sie die von Claudio Mattiussi[18] aufgestellten Bedingungen erfüllt, die eine Evolution ermöglichen bzw. erleichtern:

- Sie bildet viele verschiedene Zeichenkettenpaare auf die gleichen natürlichen Zahlen ab.
- Sie ist unabhängig von der Position der Zeichenketten innerhalb des Genoms.
- Sie ist unabhängig von den angewandten genetischen Operatoren.
- Die Verbindungsstärke zwischen zwei Neuronen kann in kleinen Schritten oder auch in großen Intervallen geändert werden.
- Sie ermöglicht es, mehrere unabhängige Verbindungen von einem Neuron zu mehreren anderen Neuronen zu etablieren, da diese auf unterschiedlichen Segmenten einer Zeichenkette operieren. - Abbildung 19 -
- Sie sollte einen geringen rechentechnischen Aufwand erfordern.
- Es muss die Möglichkeit geben, mehrere Neuronen ohne jegliche Verbindung zu erzeugen.

18 Siehe [Mattiussi 2005] Kapitel 2.8.

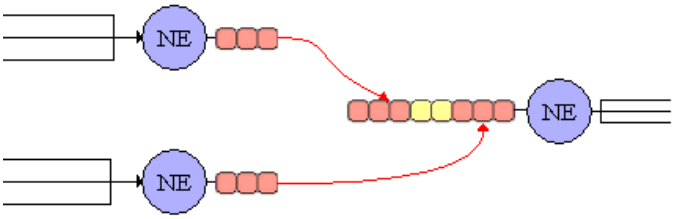

Abbildung 19: Verwendung unterschiedlicher Segmente zur Ermittlung der Verbindungsstärke.

5 Experimente

Im folgenden Abschnitt wird das soeben dargestellte Verfahren dazu benutzt, verschiedene Aufgabenstellungen zu bewältigen. Dadurch sollen die Leistungsfähigkeit, sowie auch die Grenzen des Verfahrens, umrissen werden.

5.1 XOR-Funktion

Dieses Experiment soll einen ersten Eindruck der Funktionsfähigkeit des AGE-Verfahrens vermitteln. In [Mattiussi 2005] ist es das letzte durchgeführte Experiment[19]. Ziel ist es, ein neuronales Netz zu finden, dass eine XOR-Funktion realisiert. Die entsprechenden Parameter sind zum Zwecke der Vergleichbarkeit aus [Mattiussi 2005] entnommen. Da durch die Gestalt der netzwerkspezifischen Abbildung keine negativen Verbindungsgewichte entstehen können, gibt es jeden Eingang zweimal. Zum einen mit einer positiven Ausgabe und zum anderen mit einer negativen. Insgesamt gibt es sechs Eingänge. Je zwei der binären Eingaben sowie einen positiven und einen negativen Bias.

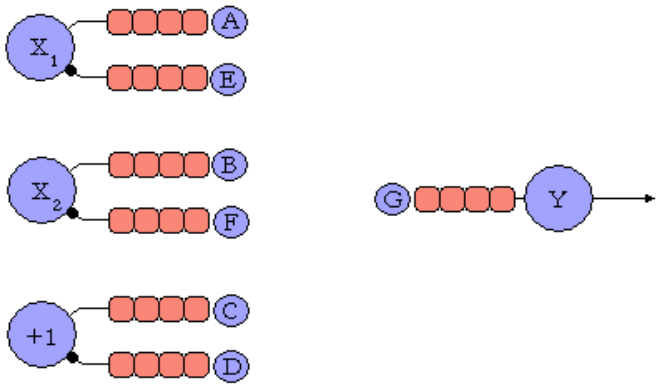

Abbildung 20: Schematische Darstellung der Ein- und Ausgänge dieses Experiments. Analog zu einer digitalen Schaltung wird die Negation des Ausgangssignals durch einen Punkt symbolisiert.

5.1.1 Einheitensatz

Die verwendeten Neuronen besitzen die Aktivierungsfunktion:

$$y = \frac{1}{1 + e^{-5\sum_j w_j x_j}}$$

Außerdem gibt es zwei unterschiedliche Neuronentypen. Ein Neuron, welches eine positive (anregendes Neuron) und eines, das eine negative (hemmendes Neuron) Aktivierung liefert. Realisiert wird dies, indem die Aktivierungsfunktion negiert wird.

19 Experiment 14 in Kapitel 4.3.4.

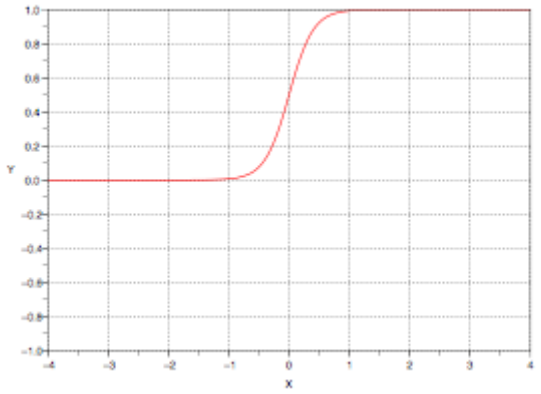

Abbildung 21:Aktivierungsfunktion eines anregenden Neurons.

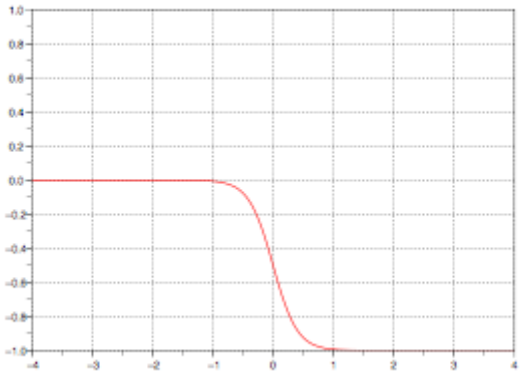

Abbildung 22: Aktivierungsfunktion eines hemmenden Neurons.

Die netzwerkspezifische Abbildungsfunktion wird definiert wie in Kapitel 3.4 beschrieben. Da das geringste Verbindungsgewicht x_{min} sehr klein ist, stellt es kein Problem dar, den Wert für $i_{min}=1$ zu setzen, denn derart schwache Verbindungen wirken sich nur marginal auf die Ausgabe des Netzes aus. Alle Parameter für die Ausrichtung von Zeichenketten sind unverändert zu Kapitel 3.

5.1.2 Fitnessfunktion

Die beiden Eingangsneuronen liefern die binären Werte, denen das Ausgangssignal zugrunde liegen soll. Dabei wird jedes Wertepaar für 20 Zeitschritte von den Eingangsneuronen erzeugt. Der Ausgang berechnet daraus den erwarteten Wert, den das Netz erzeugen soll und nutzt diesen für die

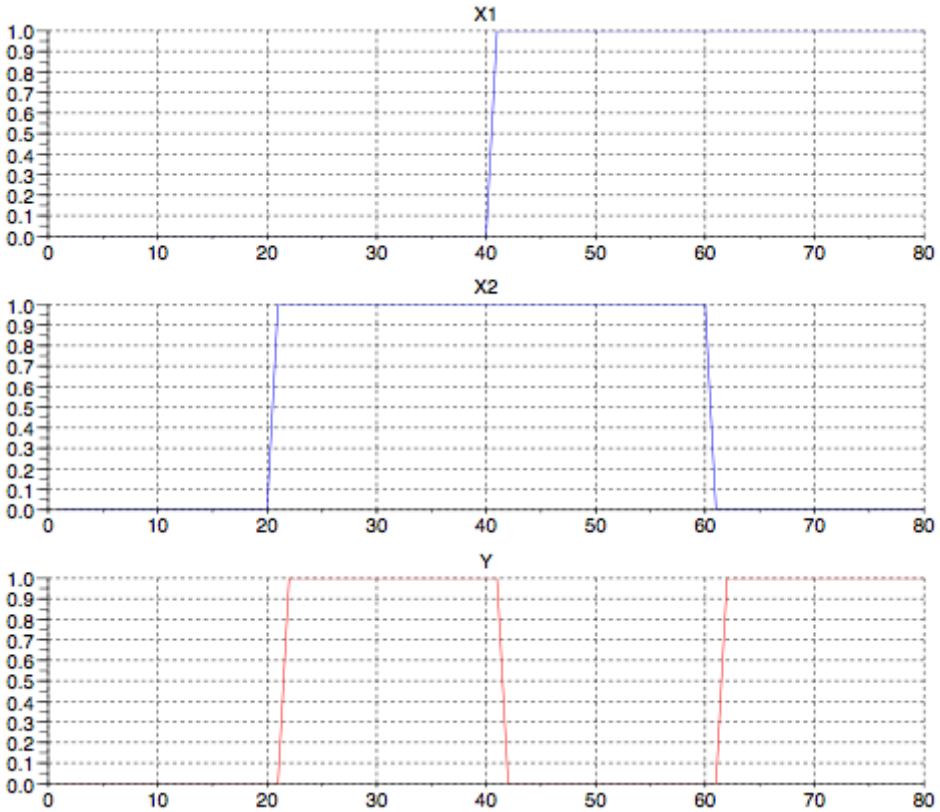

Abbildung 23: Eingangswerte für dieses Experiment (X1, X2) und erwartete Ausgabe (Y) mit einem Zeitschritt Verzögerung.

Berechnung der Fitness. Dabei wird der Wert jedoch um einen Zeitschritt verzögert geliefert, da das neuronale Netz mindestens diesen einen Zeitschritt benötigt, weil mindestens eine Neuronen-Schicht durchlaufen werden muss. Andernfalls ist es dem neuronalen Netz nicht möglich, eine Fitness von Null zu erreichen. Außerdem wird ein Toleranzintervall der Größe $\delta = 0{,}001$ um den erwarteten Wert Y definiert, so dass es sich nicht nachteilig auf die Fitness auswirkt, wenn die Ausgabe des Neuronalen Netzes (Y^*) innerhalb dieses Intervalls liegt. Der Grund dafür ist, dass die Aktivierungsfunktion der Neuronen zwar gegen eins konvergiert, diesen Wert jedoch nie ganz erreicht. Ohne ein Toleranzintervall würde die Evolution immer größere Netze hervorbringen, um näher an die binären Werte zu gelangen. Zusammenfassend lautet die angewendete Fitnessfunktion:

$$f = -\sum_{i=1}^{80} \begin{cases} (Y_i - Y_i^*)^2 & , Wenn\,|Y_i - Y_i^*| \geq \delta \\ 0 & , Sonst \end{cases}$$

5.1.3 Parameter der Evolution

Die Wahrscheinlichkeiten für die genetischen Operationen lauten[19]:

Wahrscheinlichkeit für das Einfügen eines Nukleotids	$p_{ni} = 0.001$
Wahrscheinlichkeit für das Löschen eines Nukleotids	$p_{nd} = 0.001$
Wahrscheinlichkeit für die Substitution eines Nukleotids	$p_{ns} = 0.001$
Wahrscheinlichkeit für Chromosom-Fragment-Verdopplung	$p_{f2} = 0.01$
Wahrscheinlichkeit für Komplementierte Chromosom-Fragment-Verdopplung	$p_{fc} = 0.0$
Wahrscheinlichkeit für Chromosom-Fragment-Transposition	$p_{ft} = 0.01$
Wahrscheinlichkeit für das Löschen eines Chromosom-Fragments	$p_{fd} = 0.01$
Wahrscheinlichkeit für das Einfügen einer Einheit	$p_{di} = 0.010$
Wahrscheinlichkeit für Chromosom-Verdopplung	$p_{c2} = 0.001$
Wahrscheinlichkeit für das Löschen eines Chromosoms	$p_{cd} = 0.001$
Wahrscheinlichkeit für den Crossover Operator	$p_{cx} = 0.1$
Wahrscheinlichkeit für Genom-Verdopplung	$p_{g2} = 0.001$
Wahrscheinlichkeit für Genom-Verkürzung	$p_{gt} = 0.01$
Wahrscheinlichkeit für Genom-Verkürzung der gesamten Population	$p_{pt} = 0.01$

Die Parameter der initialen Population lauten:

Anzahl der Chromosome in einem initialen Genom	$n_{ic} = 1$
Anzahl Neuronen im initialen Genom	$n_{id} = 10$
Vorkommen jedes IO-Ports im initialen Genom	$n_{ip} = 2$
Länge der Zeichenketten die einem Terminal zugeordnet werden	$l_{it} = 20$
Anzahl der Zeichen, die sich zwischen einzelnen Elementen befinden	$l_{is} = 20$
Größe der Population	$n_p = 100$
Elite	$n_r = 1$
Tournament Size	$n_e = 5$

5.1.4 Resultate

Die Evolution konnte in mehreren Versuchen innerhalb von 300 Generationen einen Fitnesswert von Null erreichen. Wenn die Evolution dennoch weitergeführt wurde, konnte außerdem beobachtet werden, dass die Struktur des Netzes weiter optimiert wurde, bis nur noch ein bis zwei Neuronen im Netz verblieben. Dies ist damit zu begründen, dass die genetischen Operatoren dazu in der Lage sind, ein kodiertes Neuron unbrauchbar zu machen und somit die Anzahl der Neuronen zu verringern. Diese Beobachtungen decken sich mit denen in [Mattiussi 2007].
Es stellte sich jedoch auch heraus, dass die Einführung eines neuronenabhängigen Fitnesswertes diesen Prozess zusätzlich beschleunigen konnte. Was vor allem im Hinblick auf die zur Verfügung stehende Rechenzeit sinnvoll war. So ermöglichte eine Modifikation der Fitnessfunktion mit $n = Anzahl\ Neuronen$ und $\nu = Strafe\ pro\ Neuron = 0.003$ zu

$$f = -\sum_{i=1}^{80} \begin{Bmatrix} (Y_i - Y_i^*)^2 & , Wenn\ |Y_i - Y_i^*| \geq \delta \\ 0 & , Sonst \end{Bmatrix} - n * \nu$$

ein deutlich schnelleres Erreichen eines Netzes mit nur einem Neuron und einen geringeren Zeitaufwand, da das System kleinere Netze schneller berechnen kann.

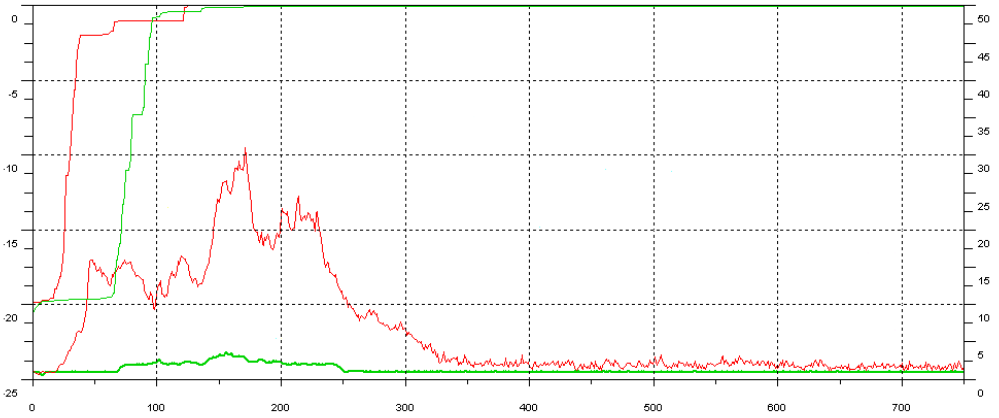

Abbildung 24: Vergleich der Fitness und der durchschnittlichen Anzahl der Neuronen zweier Experimente.
rot – neuronenunabhängige Fitness
grün – modifizierte Fitnessfunktion
x-Achse: Generation; y-Achse (links): beste Fitness; y-Achse(rechts): durchschnittliche Anzahl Neuronen

Ein vergleichbares Bild zeigt sich bei der Untersuchung der Länge des Genoms für die selben Experimente, die auch in Abbildung 24 dargestellt wurden. Dabei muss man jedoch darauf achten, dass der Wert ν möglichst klein gewählt wird. Die Evolution neigt sonst dazu, alle Neuronen aus dem Netz zu entfernen, da dies kurzfristig eine Steigerung des Fitnesswertes bewirkt. Die Evolution kommt dadurch jedoch schnell zum Erliegen, da es sich um ein lokales Maximum handelt.

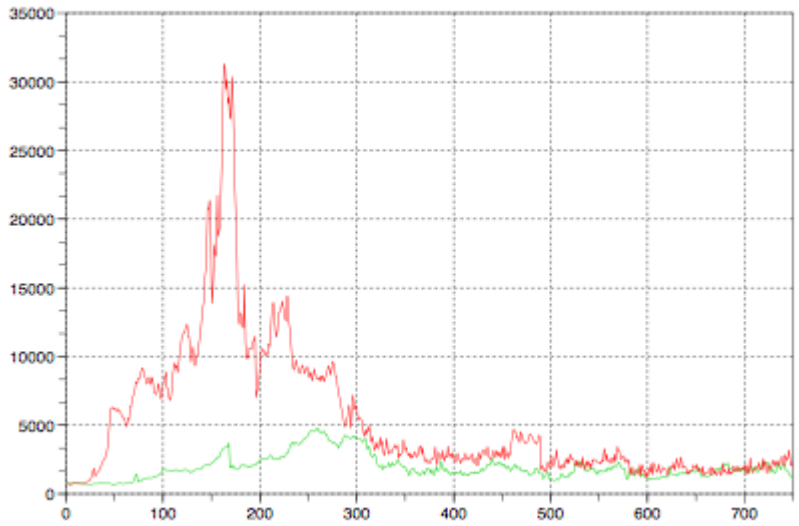

Abbildung 25: Vergleich der durchschnittlichen Länge der Genome, der selben zwei Experimente.
rot – neuronenunabhängige Fitnessfunktion
grün – modifizierte Fitnessfunktion

Die Gewichtsmatrix eines Netzes, welches die bestmögliche Fitness erreichte, ist in Abbildung 26 dargestellt. Da die vorher definierten Eingänge A/E bzw. B/F und C/D das selbe Signal liefern, kann die Matrix zusammengefasst werden. Vergleichbar zu dem vorgestellten Netz aus [Mattiussi 2005] lässt sich feststellen, dass hauptsächlich das negierte Eingangssignal genutzt wird.

	A	B	C	D	E	F	NEUE
G	~ 11,39	~ 0,71	~ 5,10	~ 12,27	~ 29,41	~ 9,22	~ 26,67
NEUR	~ 41,80	~ 1,47	~ 0,07	~ 0,47	~ 0,36	~ 0,42	~ 0,03

	A	B	C	NEUR
G	~ -18,02	~ -8,51	~ -7,17	~ 26,67
NEUR	~ 41,44	~ 1,05	~ -0,40	~ 0,03

Abbildung 26: Gewichtsmatrix eines rekurrenten neuronalen Netzes das die XOR-Funktion realisiert.
(vollständig und zusammengefasst)

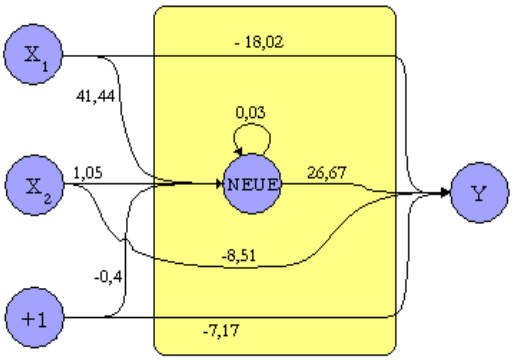

Abbildung 27: Grafische Darstellung des neuronalen Netzes aus Abbildung 26.

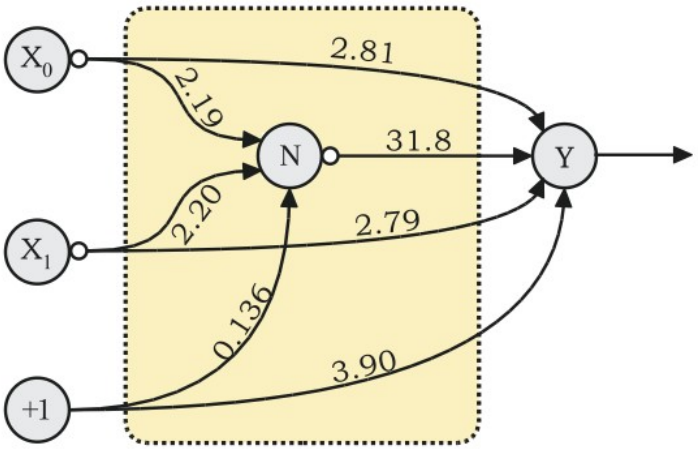

Abbildung 28: Neuronales Netz, welches die XOR-Funktion realisiert. Entnommen aus [Mattiussi 2005].

5.2 Gaussche-Glockenkurve

Auch mit diesem Experiment wurde die Leistungsfähigkeit des Verfahrens in [Mattiussi 2005] untersucht. Claudio Mattiussi evolvierte allerdings einen elektronischen Schaltkreis, der eine Gaussche Glockenkurve nachbilden sollte. In diesem Experiment wird nun versucht, ein neuronales Netz zu finden, welches die selbe Funktionalität realisiert.

5.2.1 Zielfunktion

Bei dem Experiment, welches in [Mattiussi 2005] beschrieben ist, wurde eine variable Stromquelle eingesetzt, die über 101 Simulationsschritte eine Spannung von 2.00 V bis 3.00 V in 0.01V Schritten

ausgab. Analog dazu wird in unser Beispiel ein Eingangsneuron (X) eingebettet, welches die Ausgaben 2.00 bis 3.00 liefert. Auch dieses Eingangssignal wird wieder mit positivem und negativem Signal angeboten. Zusätzlich liefert ein positives und negatives Bias-Neuron immer +1 bzw. -1 als Ausgangssignal. Das Ausgangsneuron (Y) nimmt die Ausgabe des Netzes entgegen und ermittelt die erwartete Ausgabe mit der Funktion:

$$y = 0.8\, e^{\frac{(x-2,5)^2}{0,02}}$$

Es handelt sich dabei also um eine nicht normalisierte Normalverteilung mit dem Erwartungswert 2,5 und einer Standard-Abweichung von 0,1. Die Strafe pro Neuron wird weiterhin auf $\nu = 0.0001$ reduziert. Alle anderen Parameter sind identisch zum vorangegangenen Versuch.

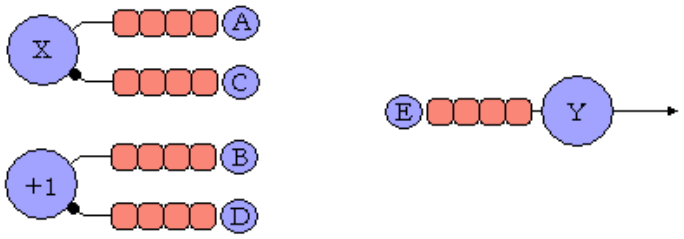

Abbildung 29: Symbolische Darstellung der Ein- und Ausgänge dieses Experiments.

5.2.2 Einheitensatz

Anders als zum vorhergehenden Experiment wird diesmal der Hyperbolische Tangens als Aktivierungsfunktion genutzt, da diese Funktion für neuronale Netze günstige Eigenschaften aufweist[20] und am NRL eingesetzt wird. Somit lauten die Aktivierungsfunktionen:

- Für anregende Neuronen: $y = \tanh(\sum_i w_i x_i)$
- Für hemmende Neuronen: $y = -\tanh(\sum_i w_i x_i)$

5.2.3 Netzwerkspezifische Abbildungsfunktion

Da es bei diesem Experiment nicht um die Erzielung einer binäre Ausgabe geht, sondern vielmehr darum, viele differenzierte Werte zu erzeugen, ist davon auszugehen, dass auch weniger große Verbindungsgewichte erforderlich sind, um die Aufgabe zu lösen. Daher wird die Abbildungsfunktion neu definiert mit:

$x_{min} = 0.001$ $x_{max} = 100$ $N = 50$

Außerdem kommt nun das zuvor in Kapitel 3.4 beschriebene Verfahren der Verbindungsunterdrückung teilweise zum Einsatz, sodass $i_{min} = 10$ gesetzt wird.

20 Für nähere Informationen siehe [Hild 2008].

5.2.4 Resultate

Weiterhin sind die Ergebnisse dieses Experiments mit denen, die in [Mattiussi 2005] präsentiert wurden, vergleichbar. Ein Netz, welches die beschriebene Normalverteilung annähert, konnte innerhalb von 30.000 Generationen gefunden werden, obwohl auch in diesem Fall Abweichungen vom Idealverlauf zu erkennen sind. Aus dem Verlauf der Fitnesswerte lässt sich erkennen, dass die Anzahl von 30.000 Generationen notwendig war, um das Experiment erfolgreich abzuschließen. Erst in den letzten 5.000 Generationen wurden noch einmal Verbesserungen des Fitnesswertes erzielt, welche die Angepasstheit des Ausgangssignals des neuronalen Netzes entscheidend verbessert haben (siehe Abbildung). Dieser erhöhte Bedarf an Evolutionszyklen deutet darauf hin, dass das gestellte Problem komplexer ist als das aus Abschnitt 5.1.1.

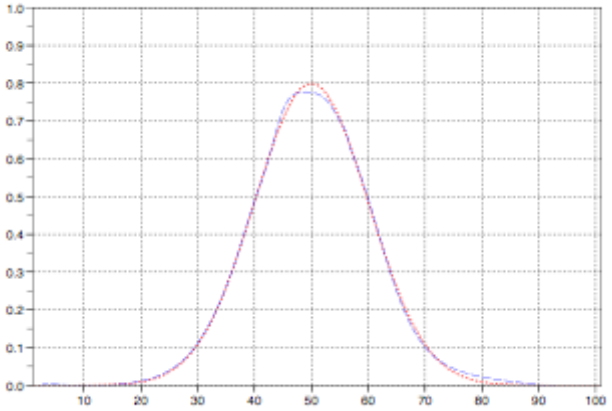

Abbildung 30: Die Ausgabe die das beste Netz erzeugte.
rot – Idealverlauf der Funktion; blau – Ausgabe des Netzes.
x-Ache: Zeitschritt (Eingangssignal bei 0 = 2,0; bei 101 = 3,0)

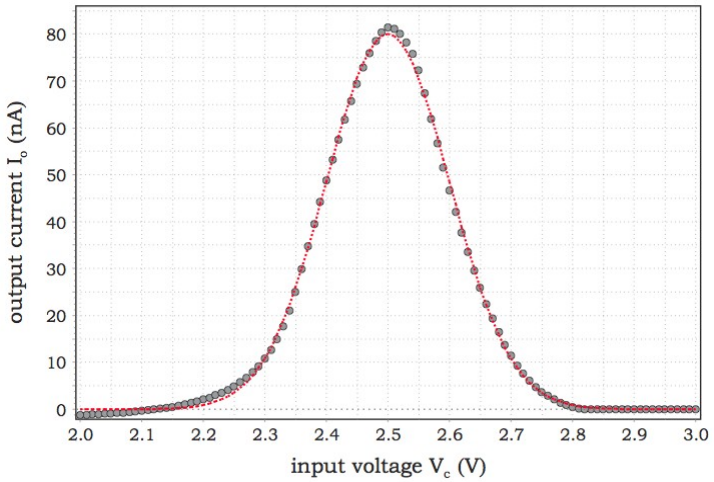

Abbildung 31: Ausgabe des elektronischen Schaltkreises, den Claudio Mattiussi evolvierte.
Entnommen aus [Mattiussi 2005].

Auf eine grafische Darstellung des neuronalen Netzes wird hier aufgrund seiner Größe von 26 Neuronen verzichtet.

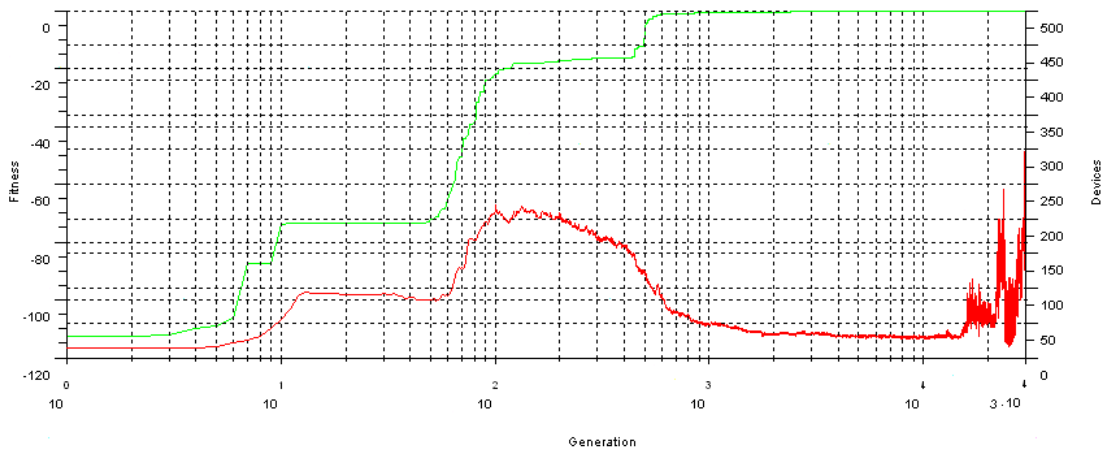

Abbildung 32: Darstellung der Fitness und der Anzahl der dekodierten Neuronen für einen ausgewählten Versuch dieses Experiments. Es zeigt sich ähnlich zu dem ersten Experiment, dass eine erhöhte Anzahl Neuronen zu Beginn erzeugt wird. Die Struktur des Netzes wird im späteren Verlauf jedoch optimiert.

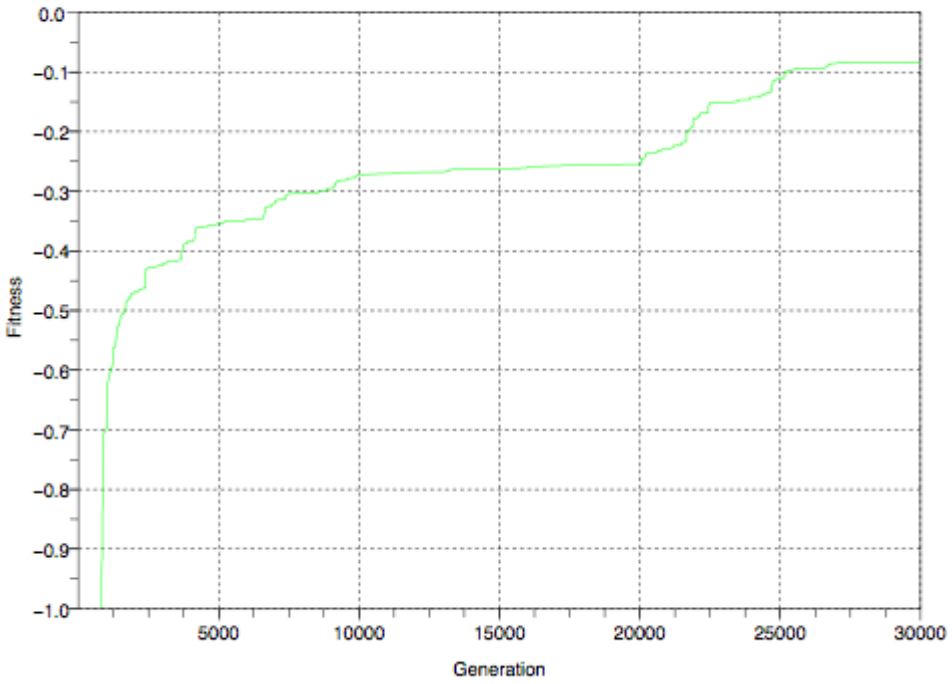

Abbildung 33: Detaillierte Ansicht des Fitnesswertes.

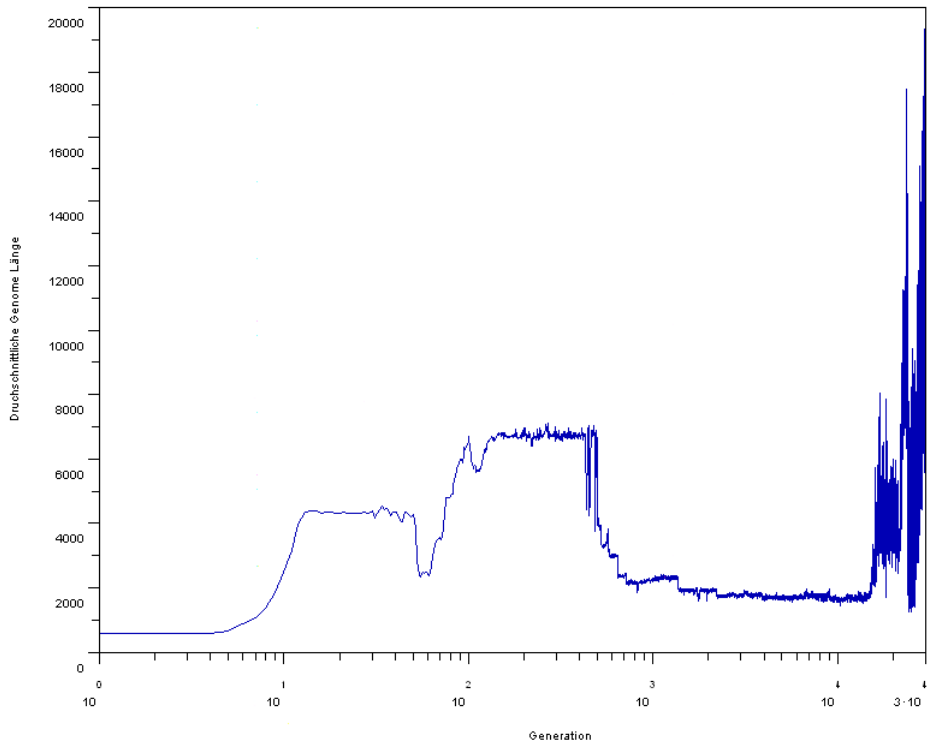

Abbildung 34: Durchschnittliche Länge des Genoms. Man erkennt, dass Länge des Genoms und Anzahl der dekodierten Neuronen in direktem Zusammenhang stehen.

5.3 Bewegungssteuerung eines humanoiden Roboters

Abschließend soll ein Experiment durchgeführt werden, welches einen höheren Schwierigkeitsgrad[21] und praktische Relevanz hat. Die Roboter, welche am Lehrstuhl für künstliche Intelligenz der Humboldt-Universität zu Berlin verwendet werden, können zur Bewegungssteuerung neuronale Netze verwenden. Dies ist vor allem in Anbetracht der Möglichkeit, sensorische Signale als Eingänge zu definieren und damit Störungen auszugleichen, interessant. Die humanoiden Roboter besitzen kleine Platinen auf denen das neuronale Netz gespeichert und berechnet werden kann. Dabei fungiert wieder der hyperbolische Tangens als Aktivierungsfunktion. Ziel dieses Experimentes ist es, ein neuronales Netz zu finden, welches die Steuerungssignale, die zuvor mit Hilfe einer fest vorgegebenen Bewegungssequenz erzeugt und aufgezeichnet wurden, nachzubilden. Bei dieser Sequenz handelt es sich um das Hinsetzen und wieder Aufstehen eines Roboters, wie er in Abbildung 35 zu sehen ist. Die drei Ausgangssignale, welche es nachzubilden gilt, sind die Winkel, die in den Servomotoren der Hüfte, Taille und des Knies anzusteuern sind. Der Verlauf dieser Signale ist in Abbildung 36 dargestellt.

21 Ein anderes Beispiel für eine komplexe Aufgabenstellung ist die Evolution eines neuronalen Netzes für das „Double-Pole Balancing“ Problem, das mit Hilfe von AGE in [Mattiussi 2006] behandelt wurde.

Abbildung 35: Humanoider Roboter des „HTH“ der Humboldt-Universität zu Berlin. Entnommen aus [HTH 2007].

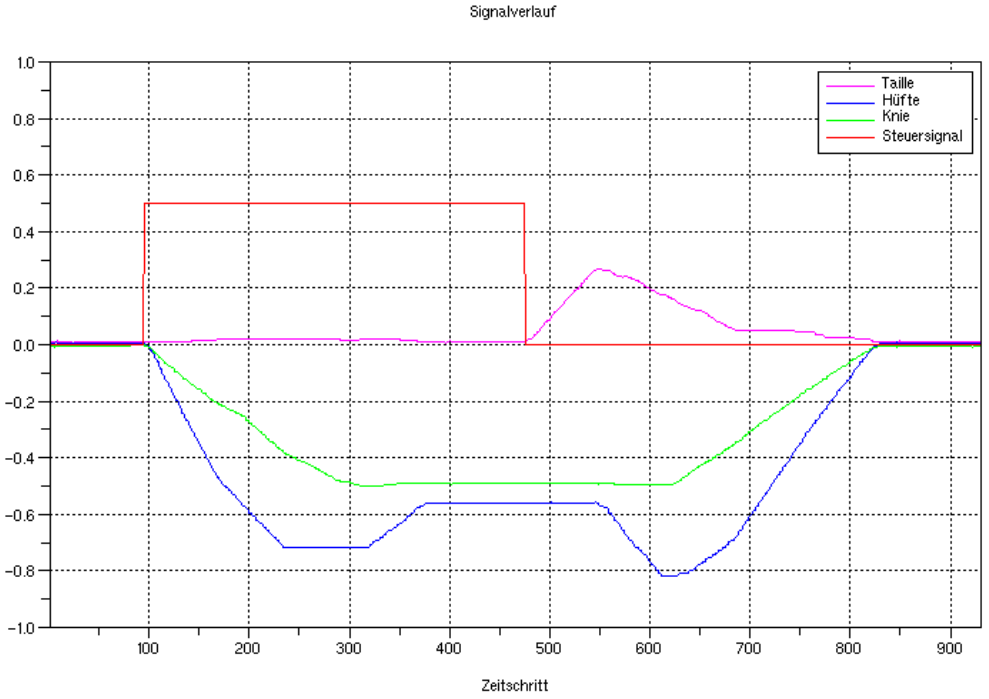

Abbildung 36: Grafische Darstellung des Eingangs- und der erwarteten Ausgangssignale dieses Experiments.

5.3.1 Netzwerkspezifische Abbildungsfunktion

Des weiteren wird die Abbildungsfunktion erneut modifiziert, da ein Verbindungsgewicht von 8,00 für dieses Experiment ausreichend sein sollte[22].

$x_{min} = 0.001$ $x_{max} = 100$ $N = 100$

Zu Beginn der Versuchsreihe wurde mit einem Wert von $i_{min} = 22$ gearbeitet, was dazu führte, dass es in der initialen Population nahezu keine Verbindungen gab. Dies hatte jedoch zur Folge, dass durch die Strafe pro Neuron die gesamte Population sehr schnell von allen Neuronen bereinigt wurde und dadurch keine sinnvolle Evolution zustande kam. Um dergleichen zu vermeiden, wird deshalb $i_{min} = 1$ gesetzt.

5.3.2 Resultate

Da sich nach mehreren Versuchen herausstellte, dass eine Evolution eines neuronalen Netzes, welches alle drei gesuchten Ausgangssignale gleichzeitig erzeugt, auch nach 50.000 Generationen nicht erfolgreich war, wurden stattdessen einzelne neuronale Netze evolviert, die jeweils nur ein Ausgangssignal vorgegeben hatten. Auch dieses Problem lies sich nicht in einem vernünftigen zeitlichen Rahmen lösen, so dass Modifikationen am Eingangssignal vorgenommen wurden, um eine schnellere Evolution zu ermöglichen. Die angepassten Signale sind in Abbildung 37 grafisch dargestellt.

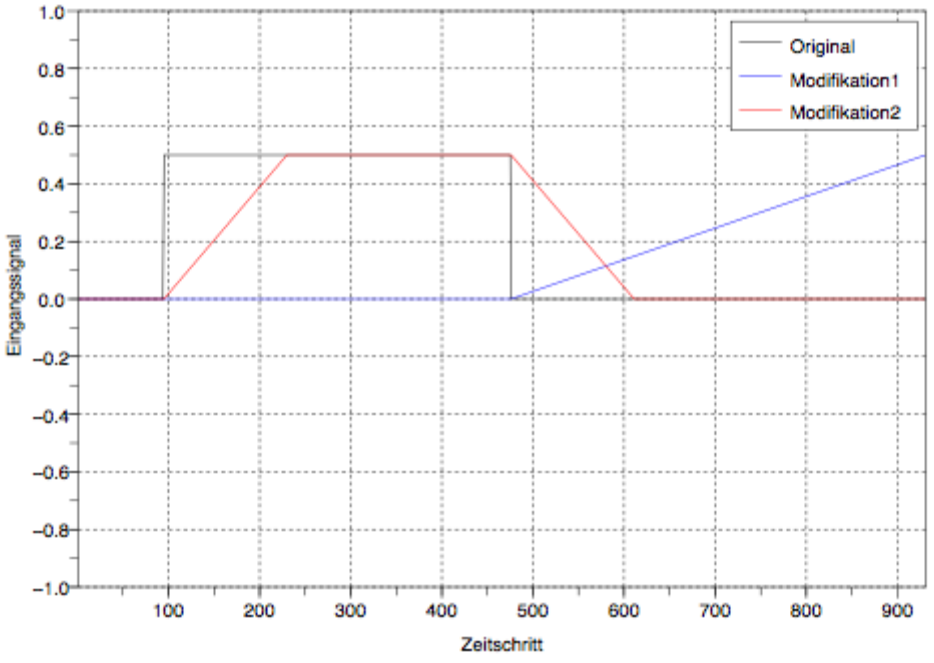

Abbildung 37: Modifizierte Eingangssignale für dieses Experiment.

22 Siehe [Hild 2008] Seite 13.

Durch diese Modifikationen konnten neuronale Netze gefunden werden, welche die gewünschte Funktionalität nahezu erfüllen, wobei jedoch das Signal der Hüfte noch größere Abweichung aufweist. Das Signal, das durch diese neuronalen Netze erzeugt wird (rot), ist in Abbildung 38, im Vergleich zum gewünschten Ausgangssignal (schwarz), dargestellt. Weiterhin ist auffällig, dass die in das Negative verlaufenden Signale in den ersten Zeitschritten eine negative Ausgabe erzeugen, bevor sie den Wert Null erreichen. Bedingt ist dies wahrscheinlich durch den Verlauf der Ausgangssignale und die Aktivierungsfunktion der Neuronen. Bei der Ausführung auf dem Roboter kann dies zu Instabilitäten führen oder die Motoren belasten und sollte deshalb manuell korrigiert bzw. ignoriert werden.

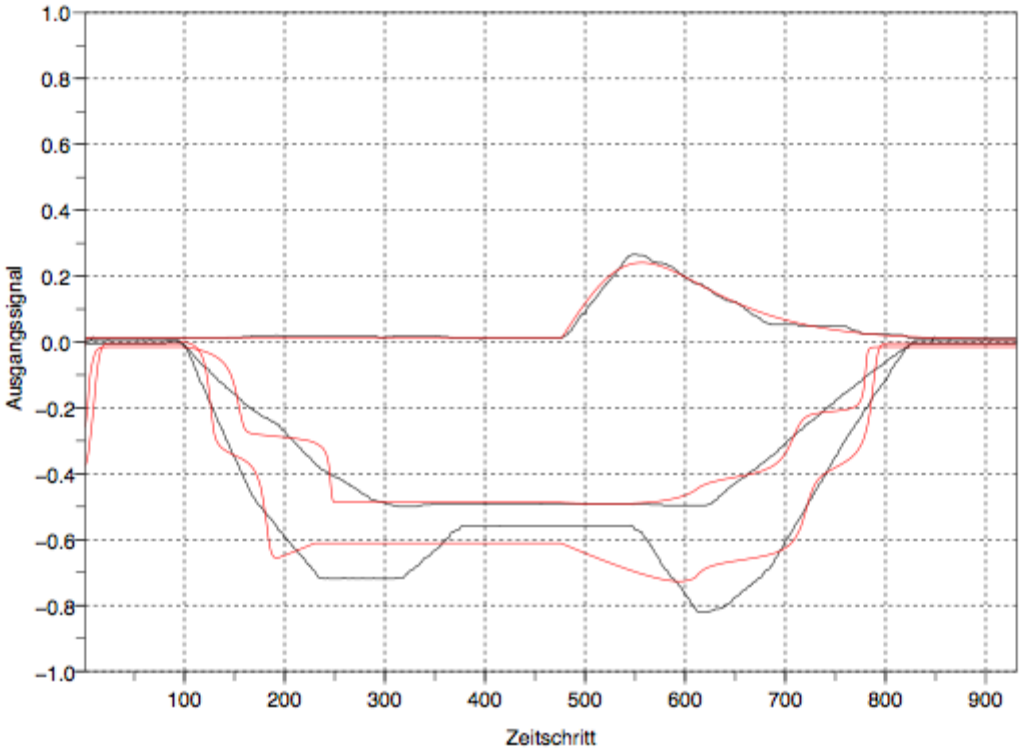

Abbildung 38: Vergleich des gewünschten Ausgangssignals (schwarz) und des Signals, welches durch das, von der Evolution gefundene, neuronale Netz erzeugt wird.

5.3.3 Hardware-Test

Zum Abschluss wurden die so evolvierten neuronalen Netze auf einem Roboter, wie er in Abb. 35 zu sehen ist, getestet. Die erwartete Bewegungssequenz war zwar deutlich zu erkennen, jedoch benötigte der Roboter bei der Aufstehbewegung manuelle Unterstützung, um nicht nach hinten umzufallen (Abb. 40). Verwantwortlich dafür ist die Diskrepanz des Signals für die Hüfte ab Zeitschritt 600 (Abb. 38), welche dazu führt, dass sich der Roboter nicht in ausreichendem Maße nach vorn beugt. Die Probleme bei der Evolution sind wahrscheinlich auf die verwendete Aktivierungsfunktion der Neuronen zurückzuführen, da die lineare Gestalt der Ausgabesignale

dadurch schlecht nachgebildet werden kann. Außerdem wurde von der Möglichkeit sensorische Signale einzubinden, kein Gebrauch gemacht, was hier aber zu besseren Ergebnissen führen könnte.

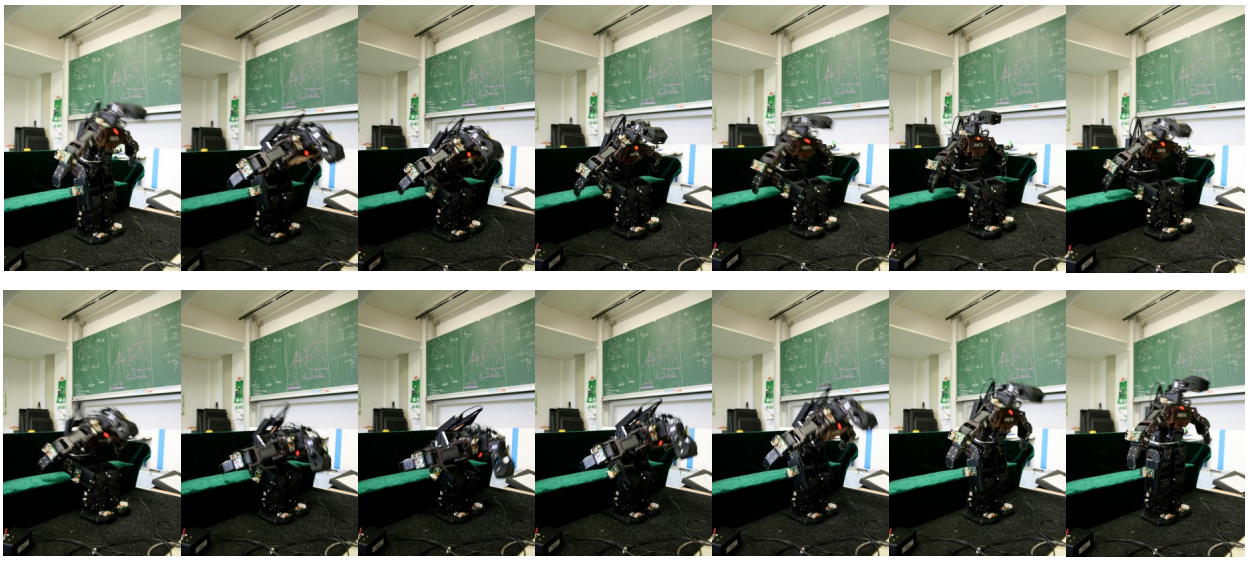

Abbildung 39: Bildersequenz des Bewegungsablaufs mit den erwarteten Signalen.

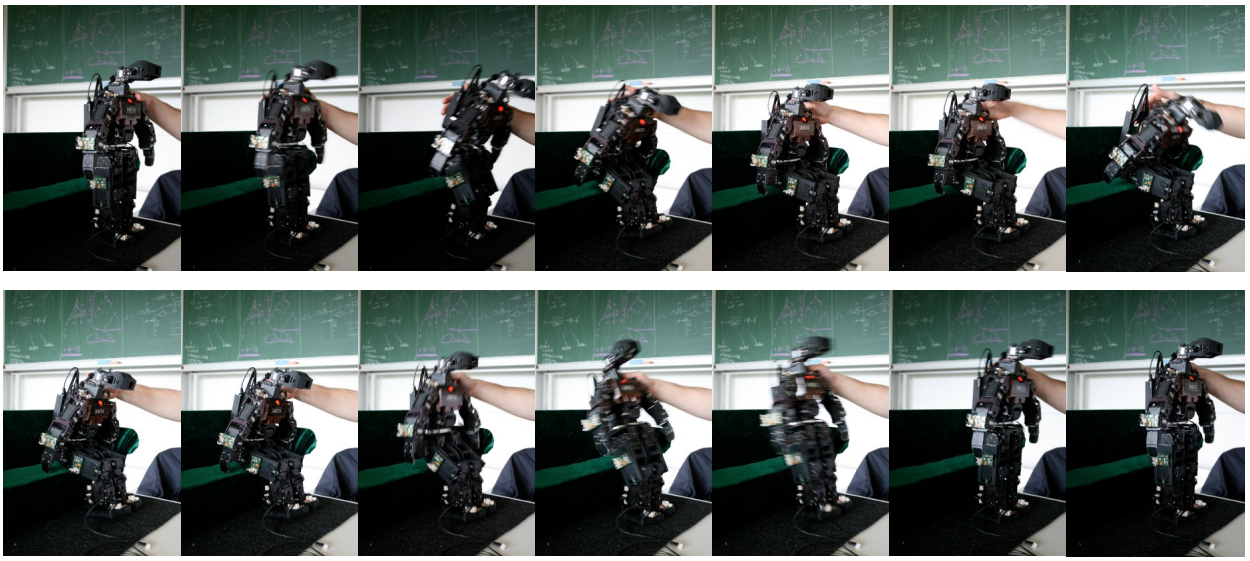

Abbildung 40: Bildersequenz des Bewegungsablaufs mit den Ausgabesignalen des neuronalen Netzes.

6 Probleme und Erweiterungen

Dieser Abschnitt behandelt das aufgetretene Problem der Erzeugung negativer Verbindungsgewichte in einem neuronalen Netz und stellt eine mögliche Lösung innerhalb des Rahmens des Verfahrens vor.

6.1 Negative Verbindungsgewichte

Durch die logarithmische Verteilung der Werte in der netzwerkspezifischen Abbildungsfunktion, war es nicht möglich negative Verbindungsgewichte zu erzeugen. Dieses Problem wurde umgangen, indem Eingangssignale auch in negativer Form angeboten wurden und ein spezielles hemmendes Neuron, welches selbst negative Ausgaben erzeugt, sobald es angeregt wird, definiert wurde. Das Problem bei dieser Implementation ist jedoch, dass unterschiedliche Neuronen nie gleichzeitig eine oder mehrere positive und negative Verbindungen zu ein und dem selben Neuron herstellen können. Dadurch wäre zum Beispiel ein Experiment zur Evolution eines SO(2)-Oszillator[23] zum Scheitern verurteilt.

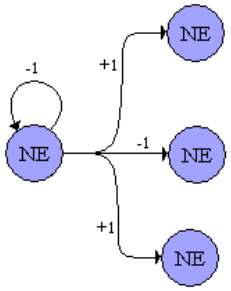

Abbildung 41: Beispiel für einen Ausschnitt aus einem neuronalen Netz, welcher nach der bisherigen Definition nicht erzeugt werden kann.

6.2 Drei-Zeichenketten basierte Verbindung

Um das im letzten Abschnitt dargelegte Problem zu umgehen, soll hier eine andere Möglichkeit vorgestellt werden, die Verbindungsstärke zu ermitteln. So soll es, ähnlich der Handhabung bei Ein- und Ausgangsneuronen im vorangegangenen Experiment, zwei Zeichenketten geben, welche mit dem Ausgang eines Neurons assoziiert werden – einen positiven und einen negativen. Dadurch stehen nun drei Zeichenketten für die Ermittlung einer Verbindung zur Verfügung.

Abbildung 42: Alternative Methode der Ausgangskodierung.

Aus diesen drei Zeichenketten (s_{1-3}) lassen sich nun zwei Ausrichtungs-Werte errechnen, die durch einen beliebigen funktionalen Zusammenhang auf eine natürliche Zahl abgebildet werden können. Beispielsweise könnte sich der Ausrichtungs-Wert a , mit dem die Verbindungsstärke definiert werden soll, aus: $a = \omega_1 L(s1, s3) - \omega_2 L(s2, s3)$

23 Siehe zum Beispiel [Hein/Hild/Berger].

zusammensetzen. Durch die unterschiedlichen Gewichtungen ω_1 und ω_2 lässt sich nicht nur die Skalierung der Ausrichtungs-Werte bestimmen, sondern die Wahrscheinlichkeit des Auftretens positiver bzw. negativer Werte beeinflussen. Für den Fall $\omega_1=1$ und $\omega_2=1$ sieht eine Verteilung der Ausrichtungs-Werte für zufällige Zeichenketten der Länge $l=22$ und der unveränderten Substitutionsmatrix aus Abschnitt 3.4.1 folgendermaßen aus:

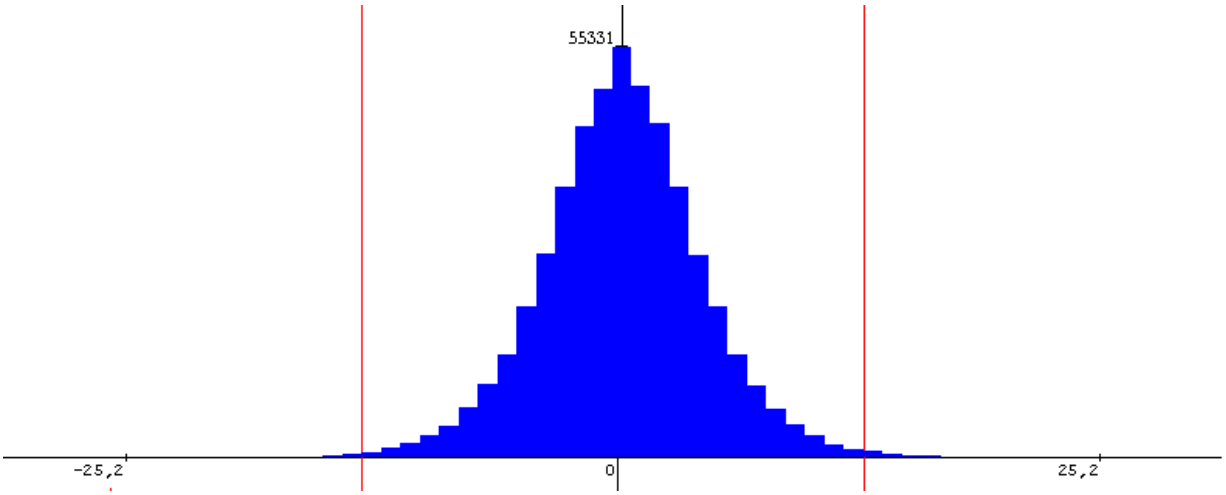

Abbildung 43: Verteilung von 500.000 Ausrichtungs-Werten aus je drei zufällig erzeugten Zeichenketten.

Die roten Linien in Abbildung 43 begrenzen wieder das Intervall in dem 99% der Ausrichtungs-Werte liegen. Durch Modifikation der Parameter zu $\omega_1=0{,}5$ und $\omega_2=1{,}5$ werden die positiven Werte schwächer gewichtet, wodurch sich die erzeugten Werte deutlich in den negativen Bereich verschieben.

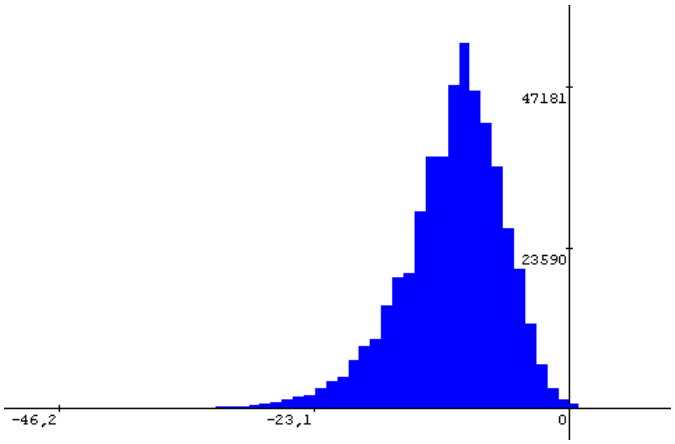

Abbildung 44: Verteilung von 500.000 Ausrichtungs-Werten aus je drei zufällig erzeugten Zeichenketten mit anderen Gewichtungsparametern.

Da nun auch negative Ausrichtungs-Werte möglich sind, ist es nötig, die netzwerkspezifische Abbildungsfunktion zu modifizieren. Dabei genügt es, eine Fallunterscheidung zu treffen:

$$x(i)=\begin{cases} x_{min}\cdot\beta^{i-i_{min}} & : Wenn\ i\geq 0 \\ -x_{min}\cdot\beta^{|i|-i_{min}} & : Wenn\ i<0 \end{cases}$$

Durch diese minimale Änderung im Verfahren zur Bestimmung der Verbindungswerte lassen sich nun auch Netze, die eine Form wie in Abbildung 42 benötigen, evolvieren.

6.3 Test

Das bereits in Abschnitt 5.1 vorgestellte Experiment zur Evolution eines neuronalen Netzes, welches die XOR-Funktion realisiert, wurde nun mit dem modifizierten Verfahren durchgeführt und konnte genauso innerhalb von 750 Generationen ein optimales neuronales Netz finden. Dabei fand die neue Funktionalität sogar mehrmals Verwendung.

	A	B	C	NEUR
G	~ -3,87	~ -2,77	~ -1,38	~ 6,64
NEUR	~ 10,14	~ 1000,0	~ -0,003	~ -0,015

Abbildung 45: Ergebnis eines XOR-Experiments mit den vorgestellten Modifikationen.

7 Zusammenfassung und Ausblick

Das hier vorgestellte Verfahren zur Evolution rekurrenter neuronaler Netze wurde angewendet, um nicht-triviale Problemstellungen, wie das Finden eines neuronalen Netzes für die Erzeugung einer normalverteilungsfolgenden Ausgabe oder die Ansteuerung einer Bewegung eines Roboters, zu lösen. Das Verfahren erwies sich dabei zwar als einstellungsabhängig, konnte die gestellten Aufgaben aber nach einigen manuellen Anpassungen zuverlässig lösen. Zusätzlich hierzu wurde eine Möglichkeit diskutiert, das Verfahren speziell für neuronale Netze abzuändern.

Jedoch konnte, wie bei evolutionären Verfahren üblich, nicht sicher gestellt werden, dass eine optimale Lösung, sofern diese überhaupt existiert, gefunden wird. Das Verfahren bietet allerdings Platz für Modifikationen und Variationen, von denen auch viele in der zugrunde liegenden Dissertation beschrieben wurden, obwohl diese nicht für die Lösung der gestellten Aufgaben benötigt wurden. Es ist daher flexibel und bietet Ansätze für weitere Forschungsarbeiten.

Die Evolutionsprozesse in den vorgestellten Experimenten erwiesen sich als langwierig, was den Schluss nahelegt, dass weiteres Optimierungspotential vorliegt. Auf der technischen Seite kann dies durch größere Rechenkapazitäten oder zunehmende Parallelisierung, z. B. beim Dekodieren unterschiedlicher Individuen oder Chromosome, erreicht werden. Softwaretechnisch ist darauf zu achten, die elementaren Operationen, wie das Ausrichten der Zeichenketten oder das Suchen von Token und Terminalen, effizient zu gestaltet.

Eine denkbare zukünftige Entwicklung ist auch eine Kopplung mit einem Lernverfahren, so dass die erstellten neuronalen Strukturen nicht mehr stochastisch sondern mit bewährten Heuristiken optimiert werden können.

A) Anhang – Softwarebeschreibung

Im Rahmen dieser Arbeit wurde das beschriebene Verfahren in einem bereits am Lehrstuhl existierendem Evolutions-Programm implementiert. Das Ergebnis dieser Arbeit soll hier kurz präsentiert werden.

Benutzeroberfläche

Das Programm „EvoManager“ wurde 2008 von Christian Thiele entwickelt und dazu benutzt, bestimmte Gewichte in einem bereits fest vordefiniertem neuronalen Netz zu evolvieren. Im Zuge der Integration des AGE-Verfahrens wurden Bestandteile dieser Software gekapselt, erweitert oder auch ersetzt.
Sie wurde in C# geschrieben und nutzt die Komponenten des .NET Frameworks. Alle Bezeichnungen in der Software sind in Englisch gehalten, um eine weitere Verbreitung und die Bedienung durch ausländische Studierende zu vereinfachen. Nach dem Starten der Applikation sieht man das Übersichtsfenster wie in Abbildung 46 zu sehen.

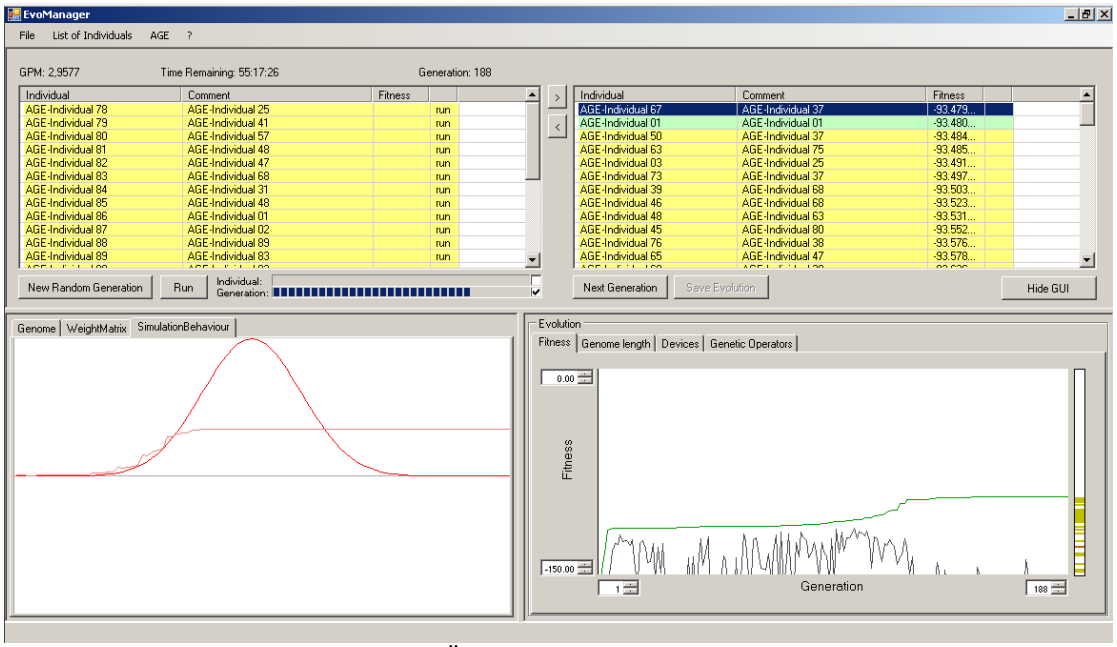

Abbildung 46: Übersichtsfenster des „EvoManager“.

Dieses Übersichtsfenster gliedert sich in drei Teile. Im oberen Bereich finden sich Informationen zu der derzeitig laufenden Evolution. Zum Beispiel in welcher Generation man sich befindet und wie lange die Evolution voraussichtlich noch benötigt um die erforderliche Anzahl an Generationen durchlaufen zu haben. Außerdem finden sich hier zwei Listen, in denen die Individuen der derzeitigen Generation vermerkt sind. Auf der linken Liste finden sich noch nicht bewertete und auf der rechten Liste die bereits dekodierten und bewerteten Individuen. Außerdem sind die Individuen farblich gekennzeichnet, um ihre Entstehung zu klären. So sind die als Elite übernommenen Individuen beispielsweise grün. Wurden alle Individuen bewertet, kann durch die Schaltfläche „New Generation“ eine neue Generation erzeugt werden, wobei alle definierten genetischen Operatoren angewendet werden.

Im unteren Bereich links sind die Werte, des in einer der beiden oberen Listen

selektierten Individuums, zu erkennen. Dort kann ebenfalls eine grafische Darstellung des Genoms, sowie die berechnete Gewichtsmatrix, des neuronalen Netzes und eine grafische Darstellung des Verlaufs der erwarteten und der tatsächlichen Ausgabe des neuronalen Netzes eingesehen werden.

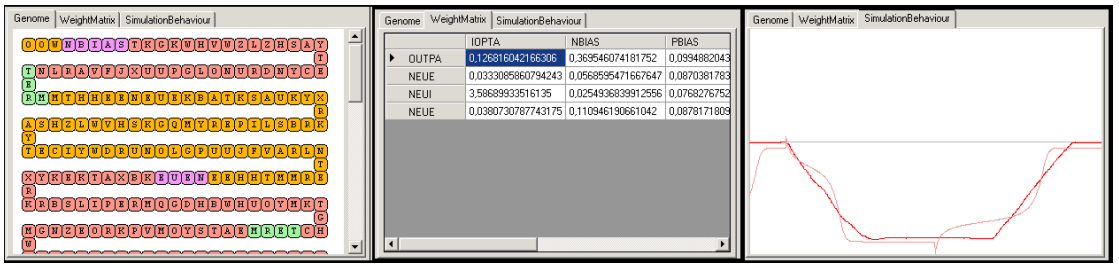

Abbildung 47: Mögliche Ansichten der Ausgabe eines Individuums.

Im rechten unteren Bereich hingegen finden sich die globalen Ausgaben, welche über den gesamten Evolutionsprozess gesammelt wurden: der Verlauf der Werte für Fitness, Länge des Genoms, Anzahl der dekodierten Neuronen sowie die absolute Häufigkeit der angewendeten genetischen Operatoren.

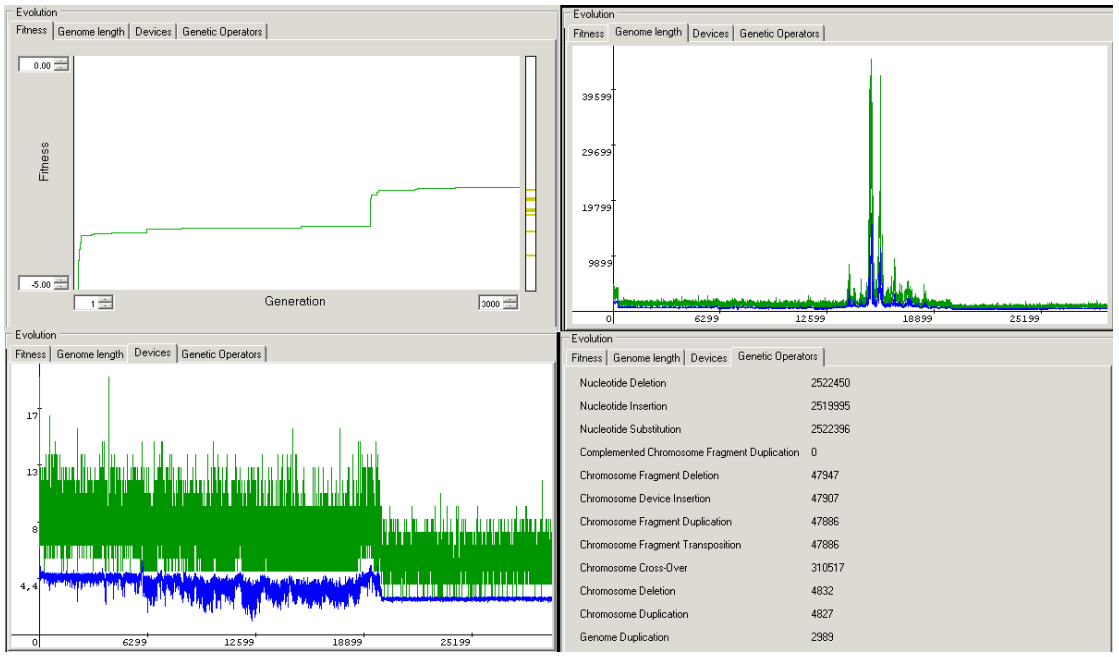

Abbildung 48: Beispiel für die vier Ausgaben des Bereichs unten rechts.
Die grüne Kurve gibt den maximalen Wert, die blaue den durchschnittlichen an.

Die Verteilung der drei Bereiche ist variabel gehalten und kann durch einen horizontalen und vertikalen Schieberegler verändert werden. Dies ermöglicht es, den Bereich von größerem Interesse zu erweitern um mehr Details wahrnehmen zu können. Da eine Evolution über mehrere tausend Generationen mehrere Stunden in Anspruch nimmt, wurde eine Schaltfläche angeboten, welche die eben beschriebenen Elemente ausblendet, was eine Beschleunigung der Berechnungen um den Faktor zwei erzielt.

Einstellungen

Über ein Menü am oberen Rand werden die verschiedenen Einstellungsmöglichkeiten erreicht.

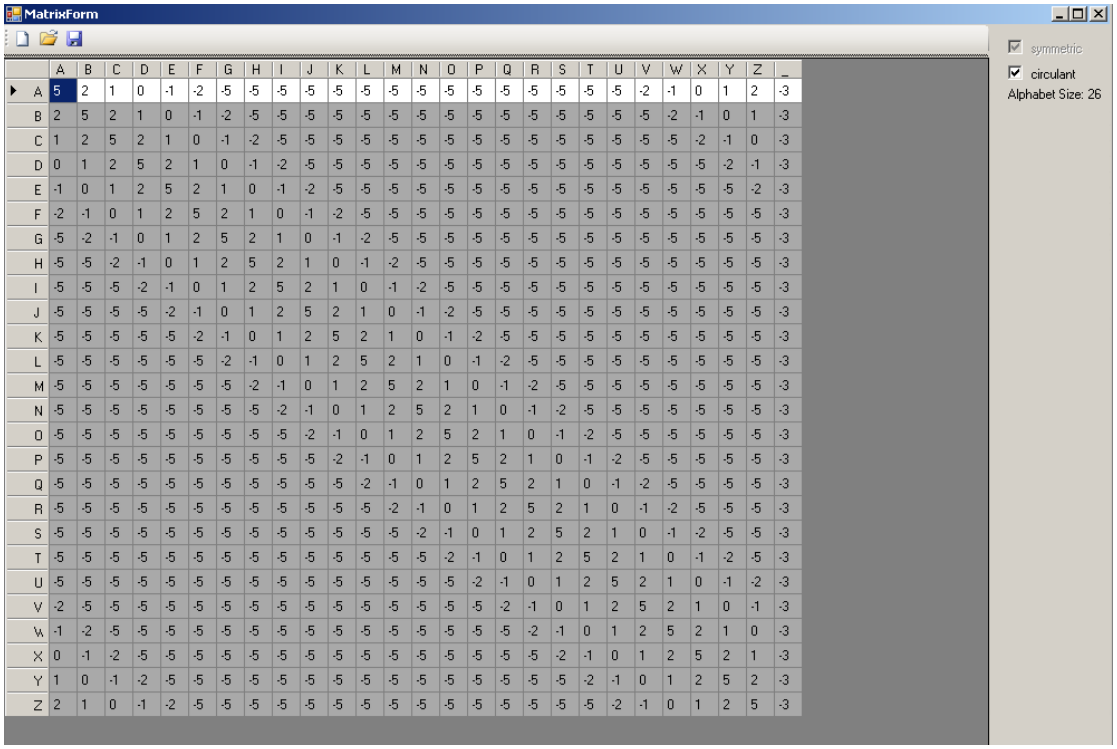

Abbildung 49: Fenster für die Definition der Substitutionsmatrix

Die erste ist das Bearbeiten der im Abschnitt „Lokale Ausrichtung (local alignment) von Zeichenketten„ beschriebenen Substitutionsmatrix. Auf der Grundlage dieser Substitutionsmatrix wird die Ausrichtung der Zeichenketten durchgeführt. Weiterhin dient sie auch dazu, die Verteilung der Ausrichtungs-Werte für zufällig erzeugte Zeichenketten zu erstellen und zu visualisieren. Diese Verteilung soll bei der Definition der netzwerkspezifischen Abbildungsfunktion behilflich sein und gegebenenfalls einen Wert für i_{min} ermitteln.

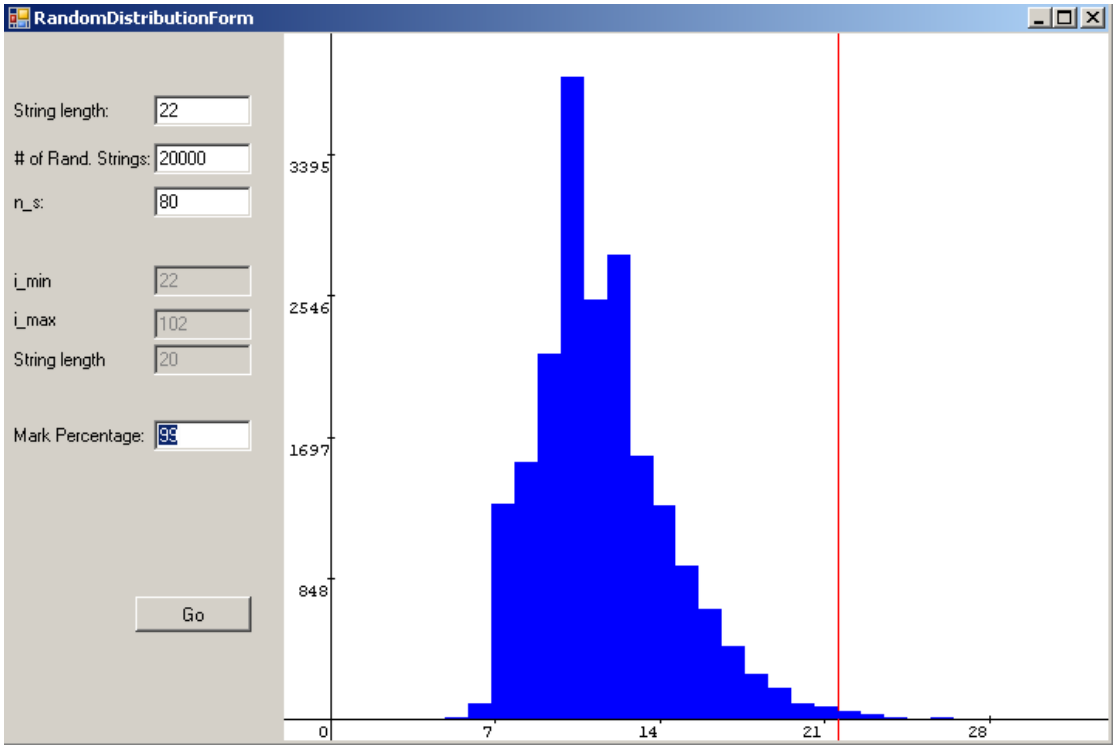

Abbildung 50: Fenster zur Ermittlung einer Verteilung.

Die Grafiken in den Abschnitten die sich mit diesem Thema befassten, wurden mit diesem Dialog generiert. Im dritten verfügbaren Dialog lässt sich nun der Einheitensatz, der für die Simulation verwendet werden soll, definieren.

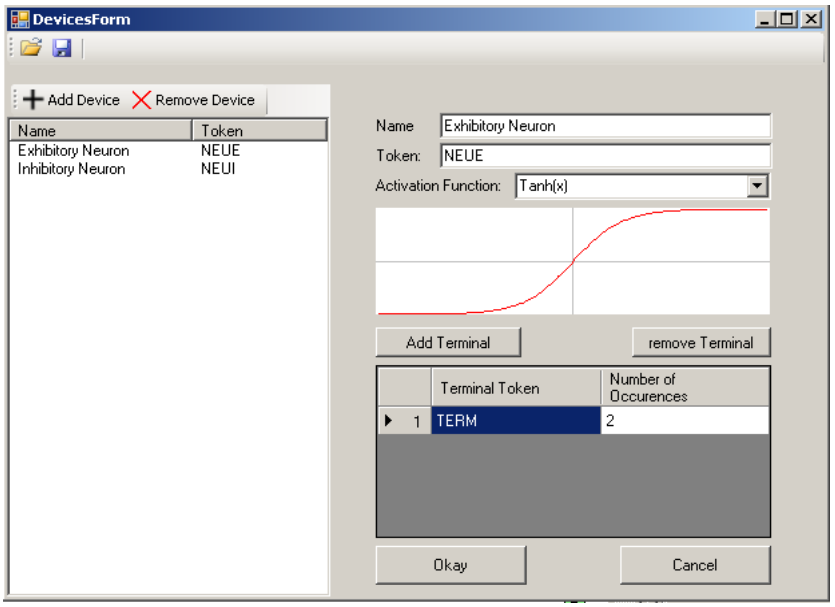

Abbildung 51: Dialog zur Definition des Einheitensatzes.

Der letzte Dialog dient dazu, die Parameter der Evolution festzulegen. So können beispielsweise die Anzahl der Simulationsschritte, der Startwerte für die initiale Population, Wahrscheinlichkeiten für die genetischen Operatoren oder, wie in Abbildung 52 zu sehen, die Eingänge und Ausgänge mit ihren Token und Terminalen definiert werden.

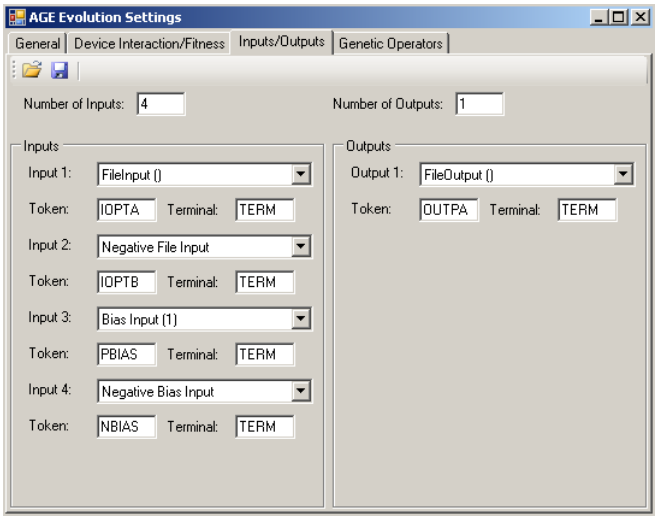

Abbildung 52: Dialog für die Evolutionsparameter.

Interne Struktur

Beim Entwurf der internen Struktur wurde darauf geachtet, eine Trennung der Komponenten nach dem MVC[24]-Modell einzuhalten. Außerdem wurde an allen Stellen, an denen Funktionalität speziell für das AGE-Verfahren benutzt wurde, entweder ein Interface oder eine abstrakte Basisklasse geschaffen, so dass die Software auch für andere Typen von Experimenten anpassbar bleibt.

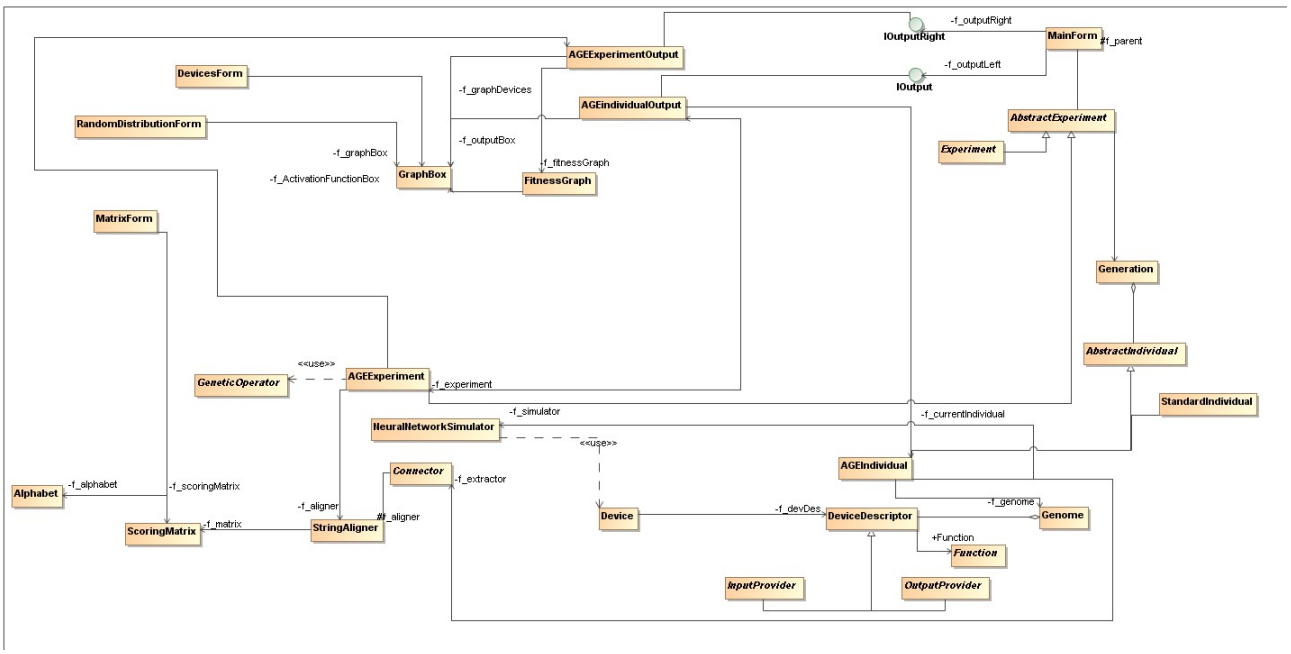

Abbildung 53: vereinfachtes Klassendiagramm der Software.
oben – Visuelle Komponenten
unten rechts – Modell Komponenten
unten links - Kontrollstrukturen

24 Abkürzung für Model-View-Controller.

Da sich Komponenten des Verfahrens ändern können, wurden bei der Implementierung abstrakte Basisklassen geschaffen, die bestimmte Funktionalität anbieten. Über die Möglichkeiten der Reflektion können dann alle Klassen gefunden werden, welche von dieser Basisklasse erben. Dies hat den Vorteil, dass andere Studenten eigene Funktionalitäten, wie eine neue netzwerkspezifische Abbildungsfunktion, einfügen können, ohne das Programm zu ändern. Es muss lediglich an einer vorgeschriebenen Stelle erweitert werden.

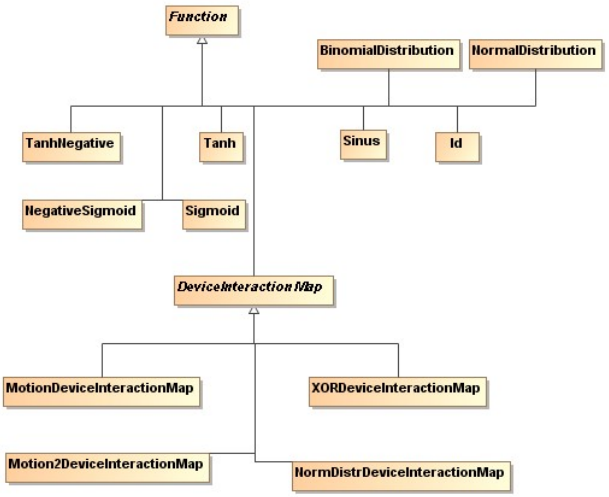

Abbildung 54: Vererbungshierarchie für die Basisklasse „Function". Um neue Funktionen einzubinden, muss nur die Klasse beerbt werden. Durch Reflektion stehen dann alle abgeleiteten Funktionen zur Verfügung.

Mögliche Erweiterungen

Erweiterungspotential an der Software gibt es vor allem im Bereich der Parallelisierung, was zum Beispiel dadurch deutlich wird, dass das System auf einem Rechner mit Dual-Core Technologie nur eine Prozessorauslastung von 50% erreicht.
Weiterhin könnten zum Beispiel Module für das Erzeugen von benötigten Funktionen zur Verfügung gestellt werden, so dass auch ohne Programmerkenntnisse die benötigte Funktionalität geschaffen werden kann.

Literaturverzeichnis

[Bäck] Bäck, Thomas: *Evolutionary Algorithms in Theory and Practice*. Oxford University Press: Oxford, 1996.

[Blickle/Thiele] Blicke, Tobias und Thiele, Lothar: *A Mathematical Analysis of Tournament Selection*, 1995.

[Gusfield] Gusfield, Dan: *Algorithms on strings, trees, and sequences*. Cambridge University Press: Cambridge, 1997.

[Harvey] Harvey, Inman: *The Artificial Evolution of Adaptive Behaviour*. PhD Thesis, University of Sussex, 1995.

[Hein/Hild/Berger] Hein, Daniel; Hild, Manfred und Berger, Ralf: *Evolution of Biped Walking Using Neural Oscillators and Physical Simulation*, Springer, RoboCup 2007: Proceedings of the International Symposium LNAI, 2007.

[Hild 2008] Hild, Manfred: *Neurodynamische Module zur Bewegungssteuerung autonomer mobiler Roboter*. Dissertation, Humboldt Universität zu Berlin, 2008.

[HTH 2007] Hild, Manfred et al., Humanoid Team Humboldt - Team Description 2007, 2007.

[Koza] Koza, John R.: *On the Programming of Computers by Means of Natural Selection*. MIT Press: Cambridge, 1990.

[Lund] Lund, Sven: *Optimierung von Neuronalen Netzen mit Hilfe Genetischer Algorithmen*. Bachelorarbeit, Universität Hamburg, 2006.

[Mattiussi 2005] Mattiussi, Claudio: *Evolutionary synthesis of analog networks*. PhD Thesis, EPFL, 2005.

[Mattiussi 2006] Dürr, Peter; Mattiussi, Claudio and Floreano, Dario : *Neuroevolution with Analog Genetic Encoding*, 2006.

[Mattiussi 2007] Mattiussi, Claudio and Floreano, Dario: *Analog Genetic Encoding for the Evolution of Circuits and Networks*, IEEE Transactions on Evolutionary Computation, Vol. 11, No. 5, 2007.

[Stanley] Stanley, Kenneth O.: *Evolving Neural Networks Through Augmenting Topologies*. Evolutionary Computation 10, pages 99-127, MIT Press, 2002.