Eberhard Karls Universität Tübingen
Mathematisch-Naturwissenschaftliche Fakultät
Wilhelm-Schickard-Institut für Informatik

Masterarbeit Informatik

Evolutionäre Methoden zur Generierung der Struktur bei spikenden neuronalen Netzen

Johannes Anufrienko

30. September 2016

Gutachter

Prof. Dr. Wolfgang Rosenstiel
Wilhelm-Schickard-Institut für Informatik
Universität Tübingen
Prof. Dr. Andreas Zell
Wilhelm-Schickard-Institut für Informatik
Universität Tübingen

Betreuer

Dr. Martin Spüler
Wilhelm-Schickard-Institut für Informatik
Universität Tübingen

Anufrienko, Johannes:
Evolutionäre Methoden zur Generierung der Struktur bei spikenden neuronalen Netzen
Masterarbeit Informatik
Eberhard Karls Universidtät Tübingen
Bearbeitungszeitraum: 01.04.2016-30.09.2016

Zusammenfassung

TODO

Inhaltsverzeichnis

Kapitel 1

Einleitung

Kapitel 2

Grundlagen

2.1 Motorcortex

Der Motorcortex ist eine Hirnregion im Grohirn, die im Frontallappen angesiedelt ist. Er erüllt seinen Zweck, in dem er die bewusste Muskelbewegungen im Körper des Menschen steuert, indem im Endeeffekt Aktionspotentiale in die jeweiligen Regionen des Körpers weitergeleitet werden. Abb. 2.1 zeigt, dass der somatosensorische Cortex, der sich schon im Parietallappen befindet, sich in unmittelbarer Nähe zum Motorcortex befindet. [1, S. 1446 f] Dieser ist mit dem Motorcortex verbunden und liefert diesem Informationen, die Entscheidend sind für die Aktivierungen von Bewegungsimpulsen. Die Verbindungen zwischen diesen beiden Regionen, ist eine wichtige topologische Ordnung dieser Hirnregionen. [2] Noch interessanter wird dieser strukturelle Aufbau, wenn man bedenkt, dass dieser ein Lernverhalten der unterschiedlichen Regionen fördern kann, [3] diese Interaktion wird auch bei Nagern vermutet. [4]

Die Funktionalität des Motorcortex wird auch in verschiedenen Anwendungen in der Informatik ausgenutzt. Das beste Beispiel dafür ist das Motor Imagery (MI). Dabei wird der Fakt ausgenutzt, dass selbst die Vorstellung einer Bewegung, z. B. des Armes, genau die gleichen Signale erzeugt wie eine physikalische Bewegung des Armes. Diese können mittels Elektroenzephalografie (EEG) gemessen werden und anschließend dafür als Signale benutzt werden um ein Computerprogramm zu steuern. [6] Abb. 2.2 zeigt, ein Trainingsprogramm bei dem ein Programm auf eine Testperson trainiert wird, mit dem entgültingen Ziel mittels MI den Roboterarm steuern zu können.

Es wurde Ratten sogar erfolgreich beigebracht einen Roboterarm zu Steuern. Dabei wurden mehrere Neuronen des Motorcortex parallel abgeleitet. Diese Informationen wurden mittels eines Künstlichen Neuronalen Netzes (KNN) weiterverarbeitet, so dass es den Ratten möglich war einen Roboterarm so zu

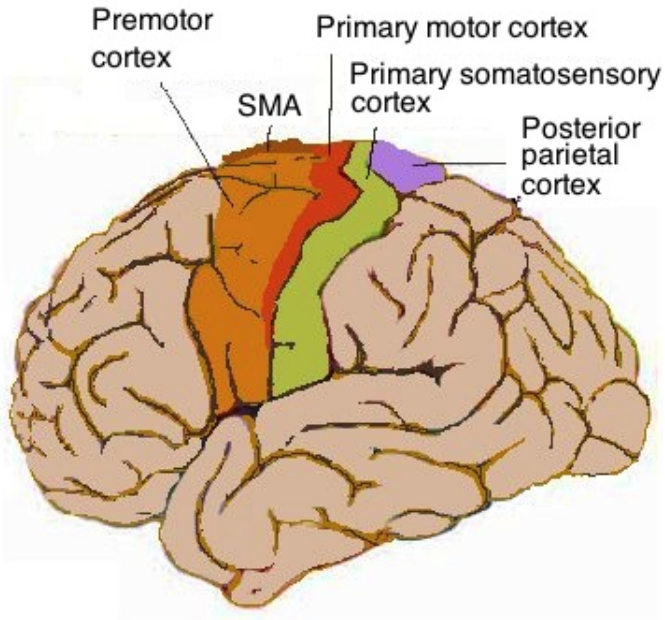

Abbildung 2.1: Eine Skizze des Menschlichen Gehirnes. Es werden die verschiedenen Unterregionen des Motorcortex hervorgehoben, so wie der daran angrenzende somatosensorischer Cortex.
Quelle: [5]

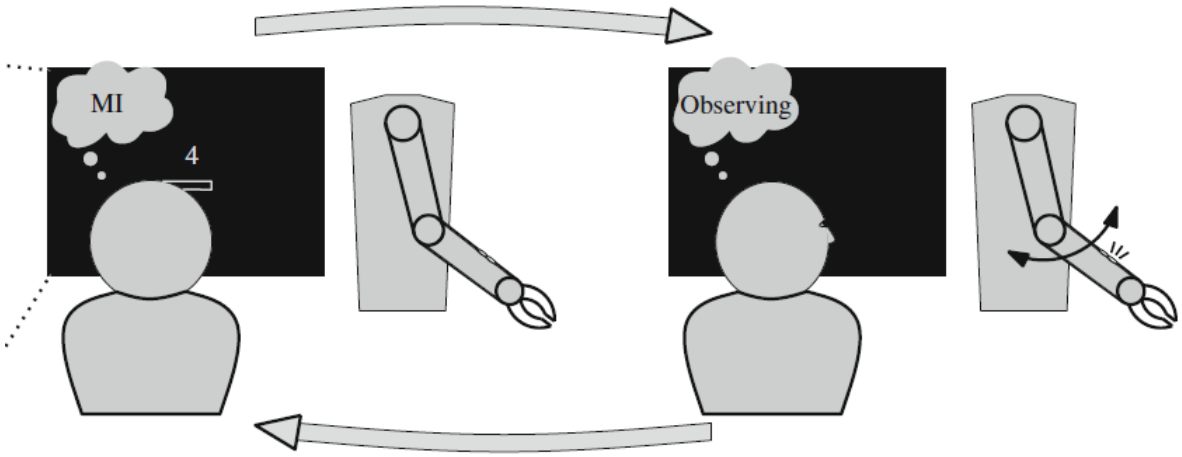

Abbildung 2.2: Die Testperson sieht sich die Bewegung eines Roboterarmes an. Anschließend stellt sie sich vor, dass sie genau die selbe Bewegung mit ihrem eigenen Arm durchführt. Dieser Prozess wird wiederholt um die genauen Messdaten für die Testperson zu bestimmen.
Quelle: [6]

steuern damit sie ihre Belohnung bekommen konnten. [7]

2.2 Künstliche Neuronale Netze

KNNs sind eine Methode des Machine Learnings um verschiedene, vorallem nichtlineare, Probleme zu lösen. [8] Wie in Abb. 2.3 zu sehen ist, besteht ein KNN aus verschiedenen Schichten (engl. Layer). Die Neurone sind miteinander verbunden, im Beispiel ist jedes Neuron aus einem Layer mit jedem Neuron aus dem darauf folgenden Layer verbunden, was natürlich nicht immer der Fall sein muss. Die relevanten unterschiede zwischen verschiedenen Neuronen sind nicht nur die Gewichte der einzelnen Verbindungen, sondern auch die Anzahl der Neuronen in jedem Layer oder ob überhaupt Verbindugen zwischen bestimmten Neuronen bestehen.
Ein KNN lernt immer pro Iterationsschritt. Dabei werden nach jeder Iteration die Gewichte der einzelnen Verbindung modifiziert. Ein KNN hat dann erfolgreich gelernt wenn es selbst verrauschte Signale richtig verarbeiten kann. [8]

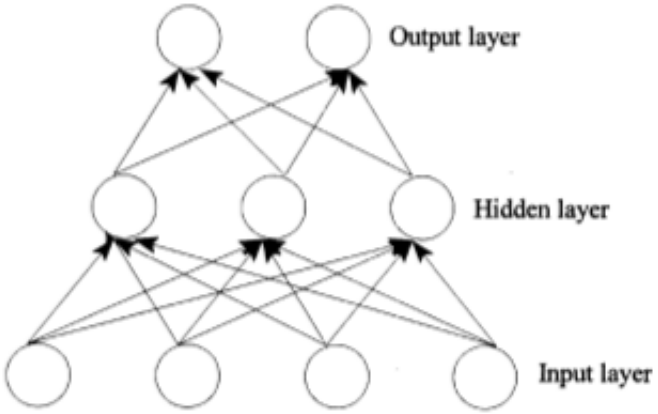

Abbildung 2.3: Einfache Struktur eines Feedforward-KNNS. Der Inputlayer leitet das Signale, gewertet mit den einzelnen Gewichten an den Hiddenlayer weiter. Dieser leitet die ankommenden Signale mit den entsprechend gewichten Weiter an den Outputlayer.
Quelle: [8]

Die Inspiration für die KNN kommt von den natürlichen Neuronen. Wie in Abb. 2.4 zu sehen ist, kriegt ein Neuron Signale von anderen Neuronen, die es weiterleitet wenn die Spannung im Zellkörper eine bestimmte Schwelle überschreitet. Das dabei entsehende Aktionspotential ist das Signal, dass an das nächste Neuron weitergeleitet wird.

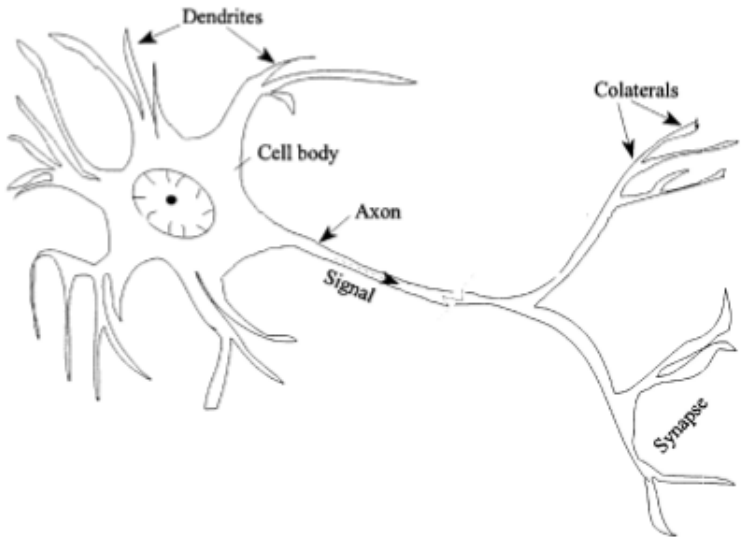

Abbildung 2.4: Ein biologisches Neuron. Der Zellkörper kann über verschiedene Signale erregt werden, die über die Dendrite zu ihm weitergeleitet werden. Erreicht er ein bestimmtes Potential, feuert das Neuron. Das dabei entstehend Aktionspotentiale wird über das Axon weitergeleitet. Die Synapse liegt an einem anderen Neuron an, wodurch das Potential in dessen Zellkörper erhöht werden kann.
Quelle: [8]

2.2.1 Spikende Künstliche Neuronale Netze

Normale KNNs stellen eine abstraktes Ebenbild eines natürlichen neuronalen Netzes dar. Spikende KNNs (SKNNs) führen dieses abstrakte Abbild näher an das biologische Vorbild. Die Idee dahinter ist, dass sich jedes Neuron ähnlich zu einem biologischen Neuron verhält. Dabei muss der Input den ein Neuron bekommt eine bestimmte Schwelle überschreiten, bevor es feuert (ein Signal weiter leitet). Dabei wird sich sehr nah an der Realität orientiert, in dem ein einfaches Modell von Neuronen implentiert wird. Die funktionolität dahinter orientiert sich dabei an der Beschreibung des Membranpotentials von Hodgkin und Huxley. [9]
Der erste große Unterschied besteht darin, dass es zwei unterschiedliche Arten von Neuronen gibt: inhibitorische und exitatorische. Während die exitatorischen Neuronen das Potential eines verbundenen Neurons erhöhen, senken die inhibitorischen Neuronen eben dieses. Desweiteren haben die Neuronen eine Refraktärzeit, weswegen sie erst nach einer bestimmten Zeiteinheit wieder feuern können, da sich die die Spannung innerhalb der Zelle erstmal wieder normalisieren muss. Abb. 2.5 beschreibt die Formel mit der die aktuelle Spannung eines Neurons pro Iteration berechnet wird. Die Spannung eines Neurons hat einen Memoryeffekt, da der aktuelle Wert nach jeder Iteration

erhalten bleibt. [10]

$$v' = 0.04v^2 + 5v + 140 - u + I$$
$$u' = a(bv - u)$$

if $v = 30$ mV,
then $v \leftarrow c$, $u \leftarrow u + d$

Abbildung 2.5: Berechnung der aktuellen Spannung eines Neurons für jede Iteration. v ist die aktuelle Spannung. a, b, c und d sind dimensionslose Parameter, die das Verhalten des Neurons modifizieren können.
Quelle: [10]

Die vier verschiedenen Parameter a, b, c und d sind dafür um das Verhalten der einzelnen Neuronen voneinander zu unterscheiden. Sie bestimmen ob ein Neuron inhibtorisch oder exitatorisch und weiter noch wie der Feuermodus aussieht. Die unterschiedlichen Modi sind in Abb. 2.6 dargestellt.

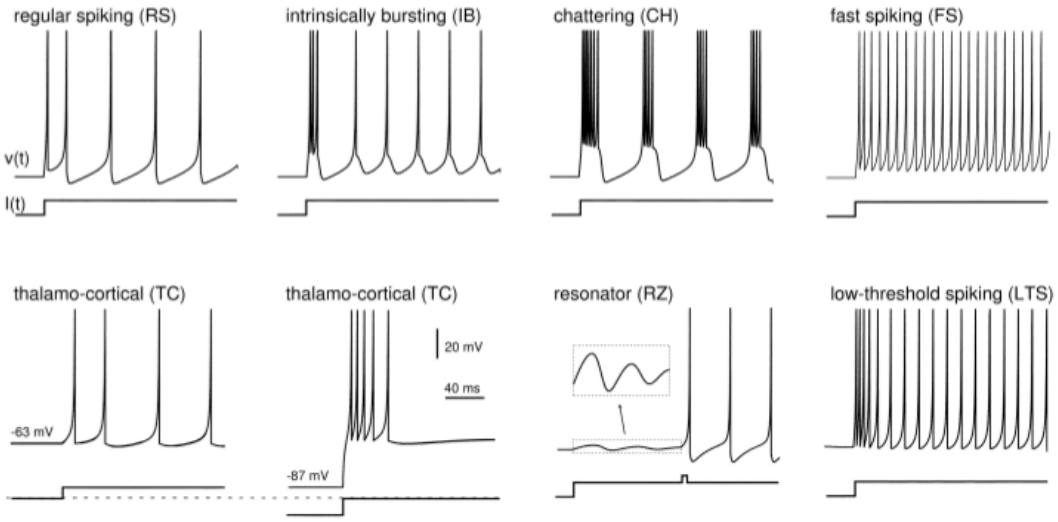

Abbildung 2.6: Verschiedene bekannte Neuronenarten, die durch unterschiedliche Parameter a, b, c und d erzeugt werden können.
Quelle: [10]

2.3 Neuronale Kodierung

Ein einzelnes Aktionspotential eines Neurons beinhaltet quasi keine Information, dennoch wird mittels Neuronen ein großer Informationsgehalt übermittelt. Das ist dadurch möglich, dass die eigentliche Information kodiert ist über das

Feuern von mehreren Neuronen. Ein Stimulus, der den Arm bewegen soll, ist demnach nicht nur ein einzelnes Neuron das feuert, sondern eher mehrere Neuronen, die Feuern oder auch nicht. Das Konzept der Neuronalen Kodierung beinhaltet, dass Informationen in einem Neuronalen System kodiert sind und deswegen die Aktivität eines einzelnen Neurons selten Informationen beinhaltet. [11]
Das Population Coding ist dabei definiert über die Aktivität einer Gruppe von Neuronen zu einem bestimmten Zeitpunkt. Diese Gruppe von Neuronen kann man sich als Bitvektor vorstellen, und immer wenn ein Neuron feuert wird dessen Wert auf 1 gesetzt, Abb. 2.7 ist ein Beispiel von Population Coding, bei dem immer nur ein einzelnes Neuron aktiv sein kann. Der Informationsgehalt der Aktivität dieser Gruppe von Neuronen wird dadurch drastisch erhöht, da mit jedem zusätzlichen Neuron der potentielle Informationsgehalt deutlich zunimmt. Der Informationsgehalt eines Population Codings mit n Neuronen beträgt dabei:

$$H = -\sum_{i=1}^{2n} \frac{1}{2n} log_2 \frac{1}{2n} = 1 + log_2 n$$

[12, S. 113 ff]
Es wurde auch nach gewiesen, dass die Bewegung des Armes bei Primaten über Population Coding im Motorcortex kodiert ist. [13]

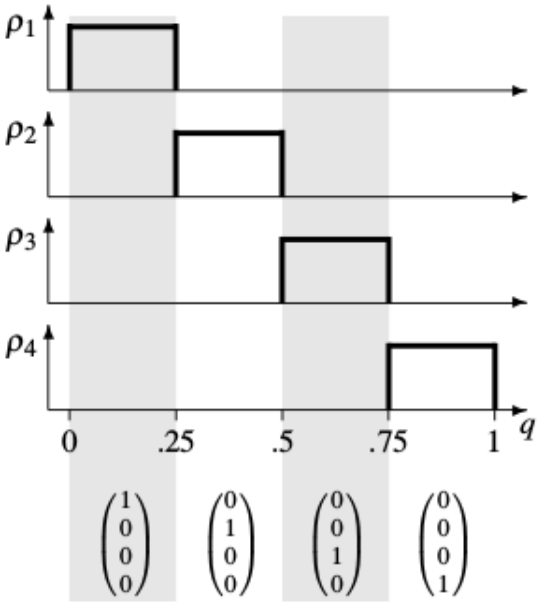

Abbildung 2.7: Beipspiel eines Vektors für das Population Coding. P1 bis P4 sind verschiedene Neuronen. Immer wenn ein einzelnes Neuron feuert, wird nur ein Informationskanal benutzt. Die so kodierte Information ist nicht überlappend. Dadurch bleibt der Informationsgehalt auf einem einfachen Level.
Quelle: [12, S. 115]

Neben dem Population Coding findet man auch das Rate Coding im Motorcortex sowie im somatosensorischen Cortex. Beim Rate Coding ist die Information über die Frequenz der Neuronen kodiert. Beim Ratecoding steigt dabei typischer Weise die Frequenz der neuronalen Antwort mit steigendem Stimulus. Auch wenn man Rate Coding mit einem Rauschen im Signal verwechseln könnte is es dennoch ein Signal mit einem Informationsgehalt. Wie in Abb. 2.8 gezeigt wird das Ratecoding durch Rauschen nicht verändert, da die Anzahl der Aktionspotentiale in einem Zeitintervall nicht verändert werden. Gerade wegen der Resistenz gegen Rauschen ist das Ratecoding sehr interessant in der Neuronalen Kodierung. [14] Wenn man im vergleich dazu das Population Coding resistenter gegen Rauschen machen wollte, müsste die Information die damit übermittelt wird zur Sicherheit mit einem gewissen Maß an Redundanz kodiert werden.

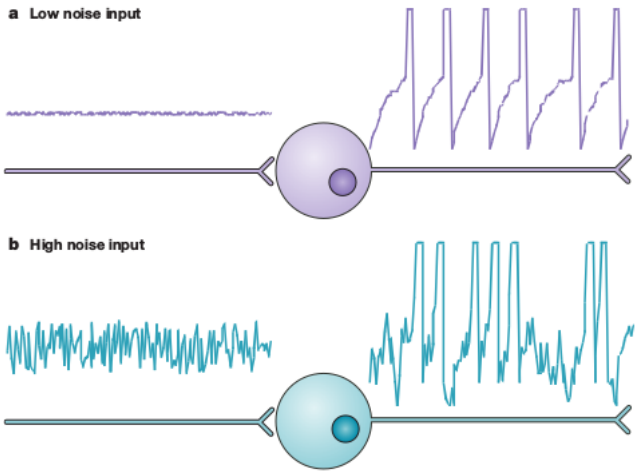

Abbildung 2.8: a. Ist ein Neuronales Signal mit niedrigem Rauschen. b. Zeigt das gleiche Signal mit einem stärkeren Rauschen. Trotz des Rauschens haben beide Signale die gleich Frequenz, da das Rauschen jediglich die Intervalle der einzelnen Spikes verändert, jedoch nicht die eigentliche Frequenz.
Quelle: [14]

2.4 Generierung von Künstlichen Spikenden Neuronalen Netzen

Die Simulation von verschiedenen Funktion des Gehirns wurde schon auf verschiedenen Ebenen getestet. Ein Verwendungszweck davon ist z. B. die Simulation von EEG Daten des Motorcortex um die Coadaptivität eines Brain-Computer Interfaces (BCIs) zu testen. Dies kann dazu benutzt um mittels MI einen Roboterarm zu steuern, der entweder physikalisch vorhanden ist oder

auch nur simuliert. [15]Während das erfolgreich getestet wurde, kann man in dieser Simulation noch eine Ebene tiefer gehen und SKNNs zu erzeugen. Diesen KNNS kann man genau die gleiche Aufgabe stellen: das einfache Bewegen eines Roboterarmes in einer Dimension.
Betrachtet man diese Problemstellung nun genauer will man ein Netzwerk erzeugen, dass eine Variable so verändert, bis Sie den gewünschten Wert erreicht hat oder sich wenigstens diesem nähert. Um dabei jedoch am biologischen Vorbild zu bleiben orientiert man sich dabei am Motorcortex und dem somatosensorischen Cortex. Das bedeutet, dass der Input für das gewünschte SKNN zwei verschiedene Informationen enthalten muss, und zwar die aktuelle Position des Roboterarmes und die gewünschte Zielposition, was biologisch gesehen der propriozeptiven, so wie der visuellen Wahrnehmung entspricht. Diese Information muss in dem Netzwerk so verarbeitet werden, dass der Output in einer Form den Arm anschließend bewegen kann.
Dieser Versuchsaufbau wurde erfolgreich umgesetzt, in dem ein SKNN mit Reinforcement Learning trainiert wurde. [16] In Abb. 2.9 sieht man ein paar der Ergebnisse die damit erzielt wurden. Ein Problem, das dabei jedoch deutlich ersichtlich wird ist, dass es wohl nicht einfach ist für das SKNN die Position still zu halten, da es immer ein Zittern gibt sobald das Ziel erreicht ist.

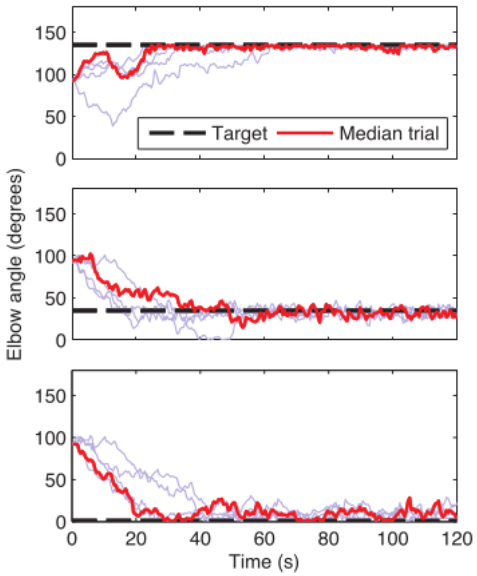

Abbildung 2.9: Die Bewegung eines simulierten Roboterarms, der von einem SKNN gesteuert wird. Die rote Trajektorie zeigt den Median der Bewegung des Roboterarms über mehrere Trials, während die schwarze Linie das Ziel ist.
Quelle: [16]

Ein weiterer wichtiger Punkt auf den geachtet werden sollte ist dann auch noch die Struktur des ganzen Netzwerk. Um sich dem Simulationsgedanken weiter zu nähern reicht es nicht die ganzen Neuronen nur in inhibitorische

und exitatorische Neuronen zu unterteilen. Bei der Bewegung des Armes spielt nicht nur der Motorcortex eine große Rolle, sondern auch der somatosensorische Cortex. Deswegen muss man in diesem Prozess die Neuronen noch weiter unterscheiden nach sensorischen und motorischen Neuronen. Die sensorischen Neuronen verarbeiten dabei die Informationen, die über den Input bereit gestellt werden und leiten diese dann weiter an die motorischen Neuronen, die diese Informationen weiterverarbeiten und anschließend eine Bewegung des Armes induzieren. Wenn man die verschiedenen Neuronengruppen in Betracht zieht, kann man randomisierte Netzwerke mit der gewünschten Struktur erzeugen. Das kann man durch Plastizität erreichen. Dabei hat jedes Neuron eine bestimmte Wahrscheinlichkeit eine Verbindung zu einem Neuron aus einer anderen Gruppe zu haben. Trainiert man so ein Netzwerk mittels Reinforcement Learning ist es auch möglich Netzwerke zu erzeugen, die genau diese strukturelle Voraussetzungen erfüllen und es schaffen den Arm zum gewünschten Ziel zu bewegen. Die SKNNs sind sogar in der Lage Zielpositionen zu erreichen, auf die sie nicht direkt trainiert wurden.[17]

2.5 NEAT

NeuroEvolution of Augmenting Topologies (NEAT) gehört zu den Methoden der Neuroevolution. Dabei werden Genetische Algorithmen (GA) als Lernmethode für KNNs benutzt. Dabei werden nicht nur die einzelnen Gewichte manipuliert sondern auch die Strukture des ganzen Netzwerkes. Bei Neat wird ein KNN in zwei Gruppen von genen aufgeteilt. Die eine Gruppe sind die Verbindungsgene, jedse Gen ist eine Verbindung zwischen zwei Neuronen und beinhaltet alle Informationen, die die Verbindung betreffen. Die zweite Gruppe sind die Neuronengene, wobei jedes Gen einem Neuron entspricht. Weiternoch sichert NEAT jede Neuerung die passiert in einem Innovation Record, damit ist jede Veränderung die in jedem Individuum passiert gespeichert und katalogisiert, wodurch z. B. der Effekt einer Mutation über mehrere Generationen verfolgt werden kann. Dadurch ist es auch moch möglich ein Crossover von zwei verschiedenen Netzwerken zu betreiben ohne einen Informationsverlust zu erleiden, wie er in Abb. 2.10 dargestellt wird. [18]

NEAT implementiert auch ein System der Spezifizierung. Das bedeutet, dass jedes Individuum eine Spezies hat. Verschiedene Spezien konkurieren nicht miteinander, was dazu führt, dass ein Indviduum nur mit den Individuen der gleichen Spezies konkuriert. Das hat den Vorteil, dass verschiedene Topologien nicht miteinander konkurieren müssen. Dadurch wird die Wahrscheinlichkeit gesenkt in einem lokalen Maximum zu Enden, da Strukturen, die zu Beginn noch schlecht performen nach mehreren Generationen auch zu einem Maximalperfomer werden können. [18]
Der letzte große Vorteil ist, dass NEAT nach dem kleinst möglichen Netzwerk

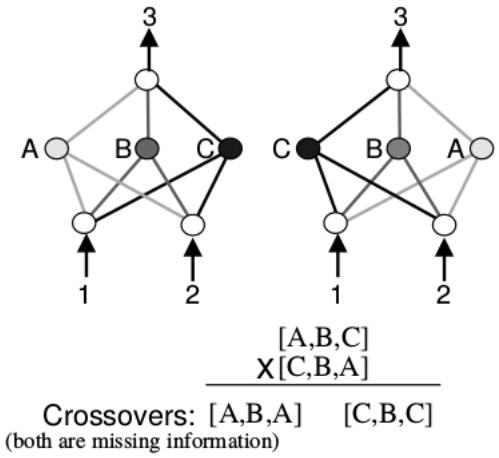

Abbildung 2.10: Zwei Netzwerke werden gezeigt, die beide nach einem möglichen Crossover beide einen Informationsverlust haben, der auch zu einer nicht funktionalität führen kann.
Quelle: [18]

sucht, das das gestellte Problem am besten löst. Dies wird dadurch erreicht, dass der Entwicklungsprozess bei NEAT mit dem kleinstmöglichen Netzwerk beginnt, also Netzwerk, dass nur aus Input und Output Neuronen besteht. Die KNNs können jede Generation größer werden durch Mutationen, die neue Kanten und Knoten hinzufügen können oder auch durch Crossovers. Das bedeutet natürlich auch, dass Probleme, deren optimale Lösung erst mit einer hohen Anzahl an Neuronen gefunden werden kann, tendentiel sehr viele Generationen brauchen, bis sie gelöst werden. [18] Um einzelne Individuen zu bewerten muss ihnen in jeder Generation ein Performance Wert zugewiesen werden. NEAT versucht natürilcher Weise die Performance zu maximieren, daher muss also eine Methode gewählt werden, die die Leistung ausschließlich positiv bewertet, da eine Performance von genau 0 bedeutet, dass ein Individuum getötet wird.

Kapitel 3

Methoden und Ergebnisse

Die Idee war es SKNNs der Art, wie sie in Kapitel 2.4 beschrieben wurden. Der fundamentale Unterschied dazu ist, dass kein Reinforcement Learning dafür benutzt wird sondern NEAT. Das Ziel ist es demnach, SKNNs zu erzeugen die mit einem Input von der Zielposition und aktuellen Position des simulierten Armes, den Arm so zu bewegen, dass er das vorbestimmt Ziel erreicht und diese Position auch hält. Jedes Individuum hat dafür nur eine begrenzte Zeit. Die Zeit das Ziel zu erreichen wurde hauptsächlich aus Performance Gründen limitiert, damit aktuelle Berechnungen nicht so lange dauern. Um die Performance zu bewerten wurde die Root-Mean-Square Distance (RMSD) zu dem Vordefinierten Ziel benutzt. Da NEAT probiert den Performance zu maximieren und der RMSD mit vortschreitender optimierung immer minimiert wird, wurde die eigentliche Performance Wertung modifiziert. Die eigentliche Formel, die die Leistung eines Individuums bestimmt ist demnach:

$$\frac{1}{RMSD+1}$$

Das führt dazu, dass die beste Performance die erreicht werden kann demanch 1 ist, da der beste RMSD 0 wäre.
Da nicht bekannt ist wie gut NEAT mit SKNNs für die gegebene Aufgabenstellung abschneidet, wurde sich langsam an das eigentliche Problem herangetastet. Begonnen wurde dabei mit einem sehr simplem Model, welches dann nach und nach immer komplexer wurde, bis man möglichst bei den SKNN angekommen ist. Zu Beginn konnte sich der Roboterarm noch von 0° bis 135° bewegen. Später wurde der maximale Winkel auf 10° reduziert, damit die KNN nicht zu groß werden und die Rechenzeit adäquat bleibt. Der Bewegungsraum war strikt vorgegeben und konnte nicht überschritten werden. Das bedeutet, dass wenn der Arm einen Winkel von 0° und ein Signal bekommt, dass er sich weiter nach unten bewegen soll, bewegt er sich nicht weiter ohne, dass es Nachteile für die Bewertung der Leistung gibt.
Für fast alle Methoden wurde ausschließlich NEAT als Lernalgorithmus benutzt und ansonsten keine weiteren Lernmethoden.

3.1 Normales Künstliches Neuronales Netzwerk

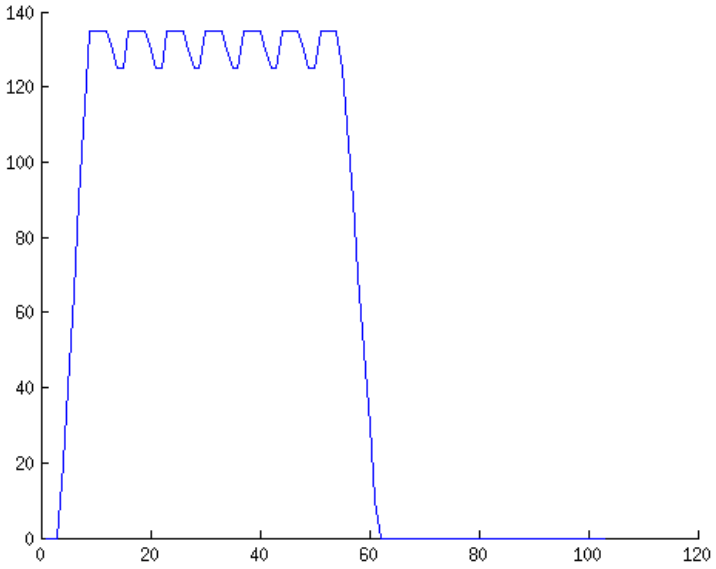

Abbildung 3.1: Simulierte Roboterarmbewegung, die vom besten Individuum erzeugt wurde, bei einem einfachen KNN.

Für den ersten Schritt wurde ein normales KNN benutzt. Es hatte zwei Input Neuronen, das eine war die aktuelle Position des Armes (eine Zahl von 0 bis 135) und das andere die gewünschte Zielposition. Das KNN hatte 50 Iterationen lang Zeit die Zielposition zu erreichen und zu halten. Abb. 3.1 zeigt eines der KNN mit der besten Performance, die das Problem gelöst haben. Das Ziel war es hierbei, die ersten 50 Iterationen den Arm auf 135° und die nächsten 50 auf 0° zu halten.

3.2 Kodierung mit einfachem Population Coding

Ab dieser Problemstellung besitzen alle Hidden Neuronen, also alle Neurone die erst durch NEAT erzeugt werden, noch die Eigenschaft inhibitorisch oder exitatorisch. Weiterhin sind die Gewichte zwischen den Neuronen jetzt nur noch zwischen 0 und 1. Wenn ein inhibitorisches Neuron ein Signal weiterleitet hat es ein negatives Vorzeichen, ein exitatorisches Neuron leitet immer positive Signale weiter. Der Input besteht aus einer Gruppe von 90

Neuronen, die über Population Coding kodiert ist. Dabei entspricht er der Differenz zwischen Zielposition und aktueller Armposition. Dadurch wird das Problem etwas vereinfacht und erspart den KNNs selber eine Struktur zu generieren, die die Differenz von zwei unterschiedlichen Werten berechnen muss. Der Output besteht aus zwei Gruppen von je 25 Neuronen. Wenn ein Output einen positivien Wert hat, bedeutet es, dass das Neuron feuert, ansonsten nicht. Die eine Gruppe der Output Neuronen ist für die aufwärts Bewegung zuständig und die andere für die abwärts Bewegung. Immer wenn ein Output Neuron feuert bewegt sich der Arm um 5° in die entsprechende Richtung. In Abb. 3.2 zeigt ein KNN, das eine bessere Leistung gezeigt hat. Das Ziel war es hierbei auch, zuerst 50 Iterationen lang der Arm auf 135° und anschließend 50 Iterationen lang auf 0°.

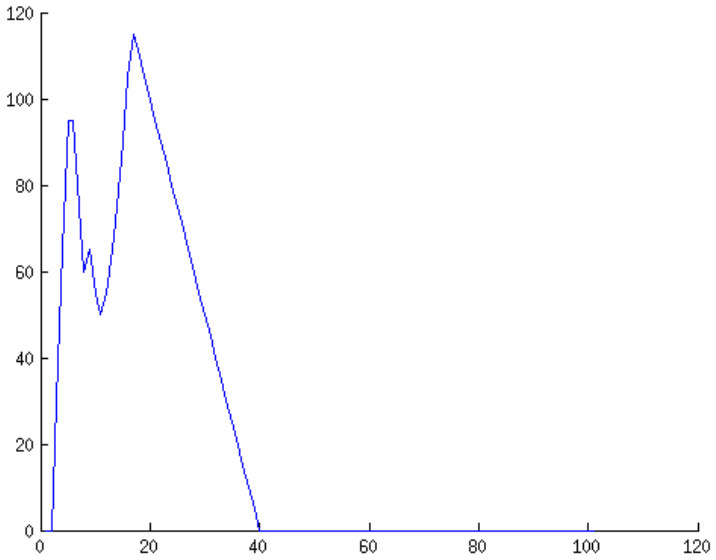

Abbildung 3.2: Simulierte Roboterarmbewegung, die vom besten Individuum erzeugt wurde, bei einem KNN mit einem Input der mit Population Coding verschlüsselt ist.

Um die allgemeine Leistung zu verbessern wurden anschließend einige Parameter verändert. Zuerst wurde das Bewegungsfeld des Armes auf ein Maximum von 10° reduziert, was erlaubt die Anzahl der Input so wie der Output Neurone auf 21 und 20 jeweils zu verringern. Das auch dazu führt, dass weniger Hidden Neuronen gebraucht werden um den Input richtig zu verarbeiten. Die Interpretation der Outputneurone wurde in dem Sinne modifiziert, dass nur noch darauf geachtet wurde wievele Neuronen aus der jeweiligen Gruppe feuern. Falls eine Gruppe dominiert hat, hat sich der arm um 1° in die entspre-

chende Richtung bewegt, ansonst hat sich der Arm nicht bewegt. Das machte das Problem etwas einfacher aber vor allem senkt es auch die Rechenzeit drastisch. Wie bei der durschnittlichen Performance in Abb. 3.4 zu sehen ist, gab es fast 2000 Generationen bis das Problem gelöst wurde.
Die zweite Änderung ist, dass das KNN sogar nur mit drei Input und zwei Output Neuronen begonnen hatte um das Problem noch weiter zu vereinfachen. Wenn anschließend ein bestimmter Performance Wert überschritten wurde, wurde die Anzahl der Input und Output Neuronen um jeweils 2. Diese Neuronen waren dann auch automatisch mit allen anderen Neuronen, die nicht auf der gleichen Ebene waren verbunden. Dadurch stieg die komplexität immer dann an, wenn es KNNs gab, die gut genug funktioniert haben um das Problem zu lösen. Die Komplexität wurde so weit gesteigert bis 21 Input und 20 Output Neurone vorhanden waren. Abb. 3.3 zeigt eines der KNNs die das Problem am besten gelöst haben.

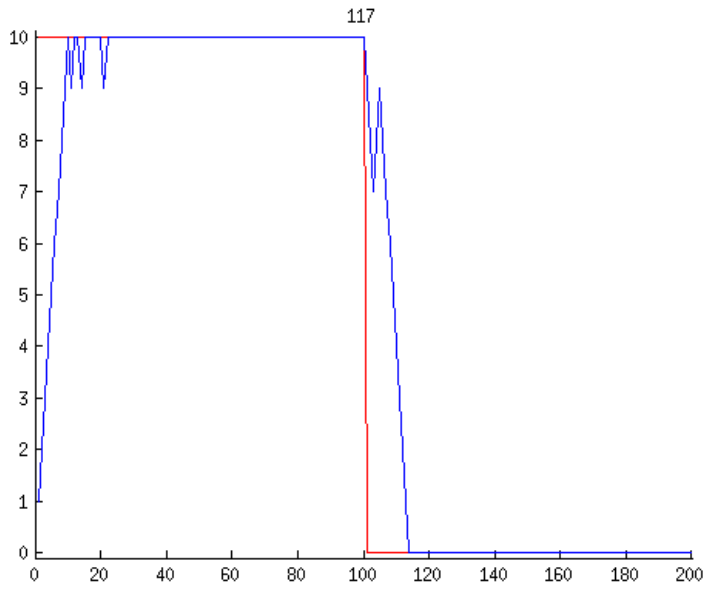

Abbildung 3.3: Simulierte Roboterarmbewegung, die vom besten Individuum erzeugt wurde. Die Rote Linie ist die Zieltrajektorie, die Blaue, die Bewegung des Armes. Das KNN hat einfaches Population Coding als Input. Beide Ziele werden relativ schnell erreicht.

3.2.1 Population Coding mit Weight Decay

Nach aufstellen eines KNNs, das das Problem gut löst musste nun noch an der Struktur des Netzwerk gefeilt werden. Damit man sich dem biologischen Vorbild weiter nähert benötigt man ein KNN dessen Input und Output Neuronen

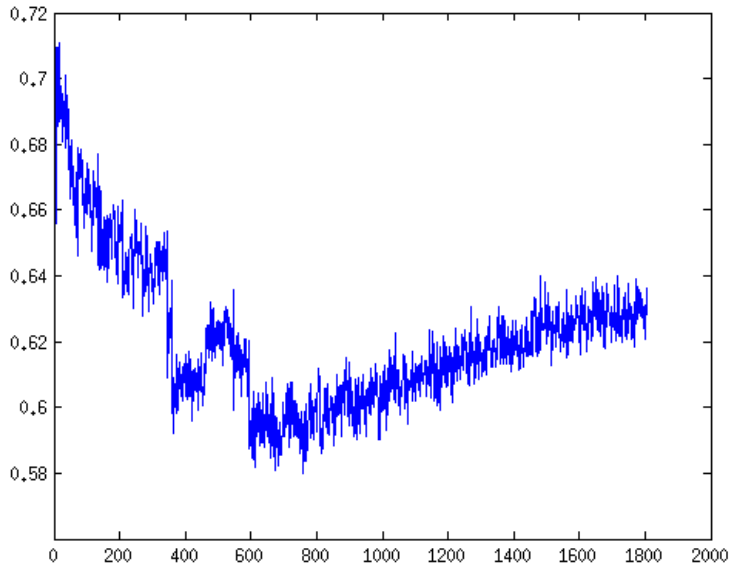

Abbildung 3.4: Die Durchschnittliche Performance aller Individuen pro Generation. Die KNNs hatten alle einfaches Population Coding als Input.

nicht miteinander Verbunden sind. Da die Output Neuronen den Neuronen des Motorkortex entsprechen, sollten sie nur Informationen aus dem somatosensorischen Cortex bekommen, was den Hidden Neuronen entspricht.
Da NEAT mit einem minimalen Netzwerk das Problem lösen will, und deswegen nur mit den Input und Output Neuronen beginnt, stellt dies ein gewissses Problem dar. Deswegen wurde bei dieser Variante der Aufbau mit einem Weight Decay erweitert. Bei einem Weight Decay wird das Gewicht von bestimmten Kanten über die Dauer des Lernprozesses reduziert. Von dem Weight Decay sind jedoch nur die Verbindungen zwischen den Input und Output Neuronen betroffen. Der Weight Decay wurde so implementiert, dass er mit jeder Komplexitätsstufe zunimmt, bis die Verbindungen auf der höchsten Stufe nur noch ein Gewicht von 0 haben. In Abb. 3.5 wird ein Testlauf eines KNNs gezeigt, das eine bessere Performance hatte. Während man die Entwicklung der durchschnittlichen Performance über allen Individuon in Abb. 3.6 sehen kann.

3.3 Kodierung mit doppeltem Population Coding

Ein anderer Versuchsaufbau der die Problemstellung komplexer gestaltet, ist derjenige, der den Input weiter aufspaltet. Bisher bestand der Input nur aus

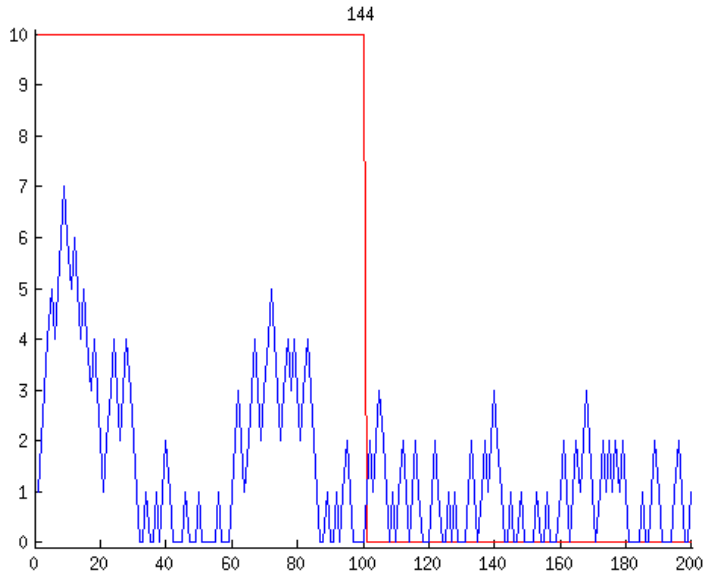

Abbildung 3.5: Simulierte Roboterarmbewegung, die von einem gut performenden Individuum erzeugt wurde. Die Rote Linie ist die Zieltrajektorie, die Blaue, die Bewegung des Armes. Der Input war über Population Coding kodiert. Die Kanten von den Input zu den Outputneuronen hatten einen weight decay.

einer Gruppe von Neuronen, die die Differenzen zwischen Armposition und Zielposition kodiert haben. Bei dieser Variante besteht der Input aus zwei Gruppen mit je 21 Neuronen. Beide Gruppen kodieren ihre Informationen mit Population Coding. Die eine Gruppe representiert dabei die aktuelle Position des Armes, während die andere die Zielposition darstellt. Das macht das ganze Problem deutlich komplexer, da nun auch noch die Differenz von zwei verschiedenen Werten mitberehnet werden muss. Weswegen hierbei die Verbindungen von Input zu Output Neuron nicht manipuliert worden sind. In Abb. 3.7 kann man dabei die Trajektorie von einem der besseren KNNs betrachten und passend dazu sieht man die durchschnittliche Performance der Individuen in Abb. 3.8. Die Staffelung des Problems mittels Komplexitätsklassen bleibt dabei natürlich erhalten.

3.4 Kodierung mit Population Coding und Rate Coding

Dieser Aufbau ist nur eine kleine Modifkation von dem Aufbau in Kapitel 3.3. Hierbei wird die Gruppe der Input Neuronen, die das Ziel darstellen mittels Rate Coding kodiert. Alles andere bleibt unverändert. Eine der besseren Tra-

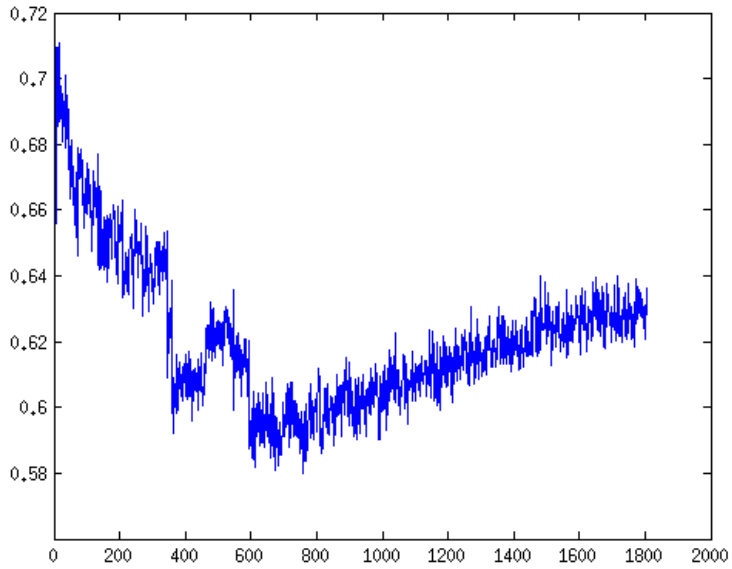

Abbildung 3.6: Die Durchschnittliche Performance aller Individuen pro Generation. Der Input war über Population Coding kodiert. Die Kanten von den Input zu den Outputneuronen hatten einen weight decay.

jektorien wird in Abb. 3.9 dargestellt, während Abb. 3.10 die durchschnittliche Performance darstellt.

3.5 NEAT Hybird mit Reinforcement Learning

Dieser Versuchsaufbau benutzt als Basis den Aufbau der zu Beginn in 3.2 beschrieben wurde. Es gibt also keine Komplexitätsstufen und kein Weight Decay. Stattdessen wird jedes Individuum in jeder Generation für maximal 30 Sekunden mittels Reinforcement Learning trainiert und anschließend wird die Performance bewertet. Dabei wird das Reinforcement Learning benutzt, dass in [17] benutzt wurde. Die Idee hierbei ist, dass NEAT in jeder Generation hauptsächlich die Topologie der SKNNs optimiert, während im Reinforcement Learning die Gewichte optimiert werden. Eine der besseren Trajektorie wird in Abb. 3.11 dargestellt. Die durchschnittliche Performance über alle Individuen wird in Abb. 3.12 gezeigt.

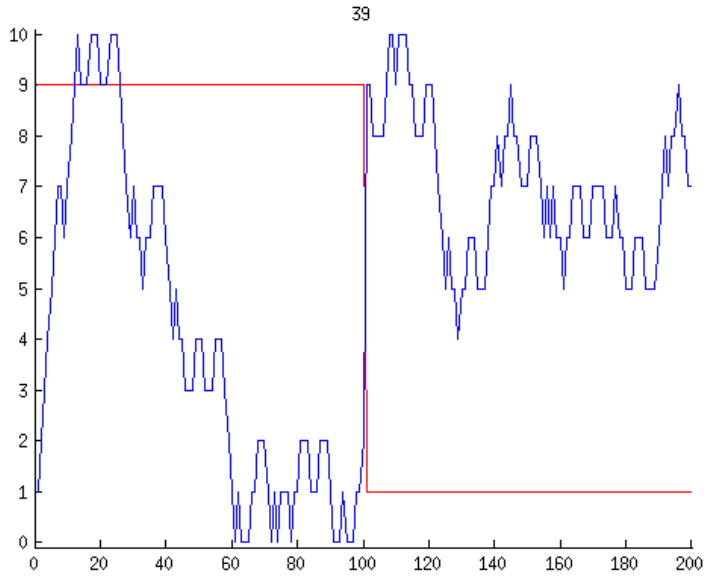

Abbildung 3.7: Simulierte Roboterarmbewegung, die vom besten Individuum erzeugt wurde. Die Rote Linie ist die Zieltrajektorie, die Blaue, die Bewegung des Armes. Es gab zwei Gruppen von Input Neuronen und beide waren mit Population Coding kodiert.

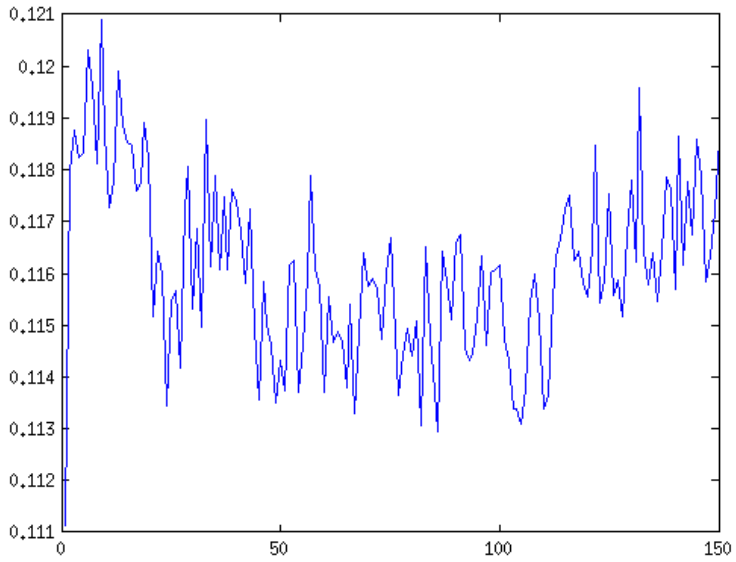

Abbildung 3.8: Die Durchschnittliche Performance aller Individuen pro Generation. Es gab zwei Gruppen von Input Neuronen und beide waren mit Population Coding kodiert.

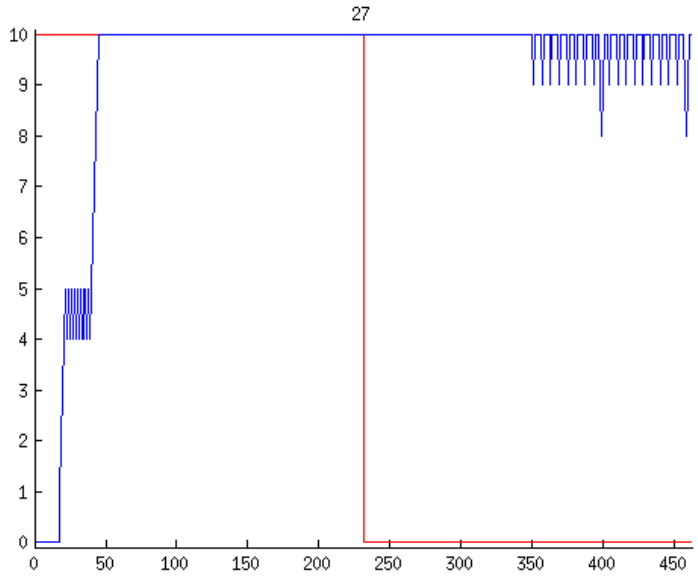

Abbildung 3.9: Simulierte Roboterarmbewegung, die vom besten Individuum erzeugt wurde. Die Rote Linie ist die Zieltrajektorie, die Blaue, die Bewegung des Armes. Es gab zwei Gruppen von Inputneuronen, die eine war über Population Coding und die andere über Rate Coding kodiert.

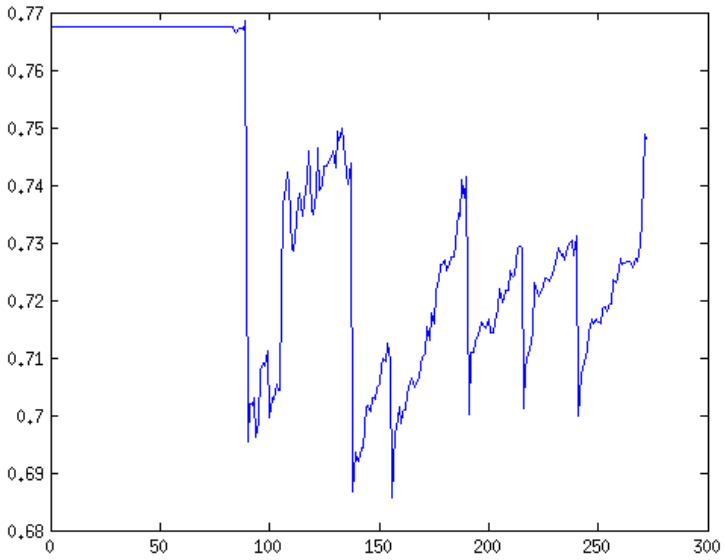

Abbildung 3.10: Die Durchschnittliche Performance aller Individuen pro Generation. Es gab zwei Gruppen von Inputneuronen, die eine war über Population Coding und die andere über Rate Coding kodiert.

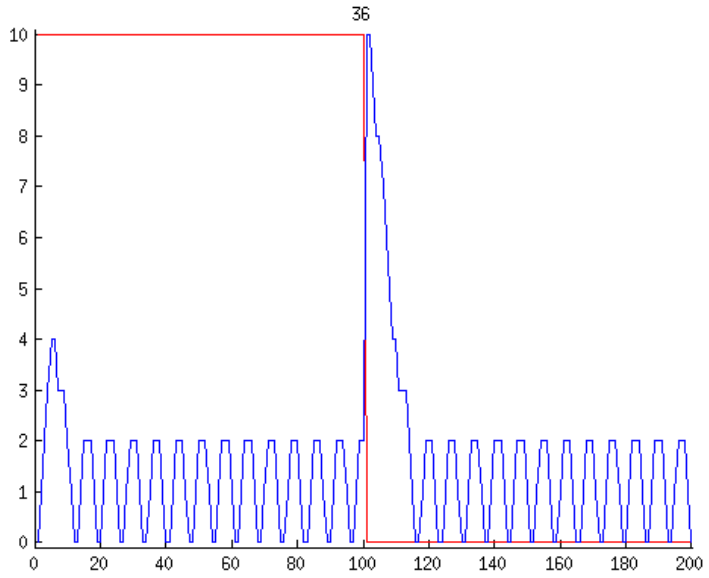

Abbildung 3.11: Simulierte Roboterarmbewegung, die vom besten Individuum erzeugt wurde. Die Rote Linie ist die Zieltrajektorie, die Blaue, die Bewegung des Armes. Jedes SKNN unterlief in jeder Generation ein Reinforcement Learning Training.

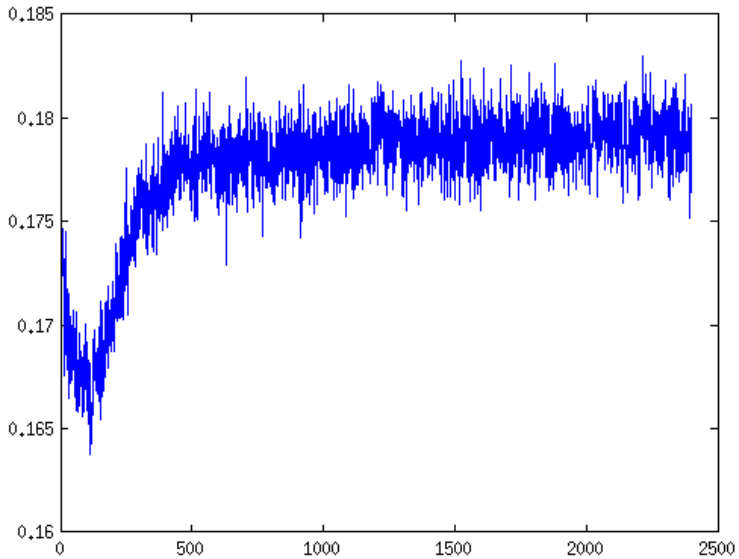

Abbildung 3.12: Die Durchschnittliche Performance aller Individuen pro Generation. Jedes SKNN unterlief in jeder Generation ein Reinforcement Learning Training.

Kapitel 4

Diskussion

Das generieren von SKNNs mit NEAT die eine doch sehr bestimmte Topologie aufweisen müssen stellte sich schwieriger fest als man es zunächst glauben mag. Das könnte verwunderlich sein, da gerade NEAT ausgelegt ist, die beste minimale Topoligie zu finden.
Es gibt mehrere Gründe warum das nicht natürlich funktioniert. Der erste Grund wäre demnach der simpelste. Und zwar würde es dann wohl wahrscheinlich daran liegen, dass es NEAT nicht immer schaffen kann die optimierte biologische neuronale Struktur des Cortex nachbilden zu können. Wenn es dann an dem Prinzip von NEAT liegt, dass der Optimierungsprozess immer ohne Hidden Neurone gestartet wird. Dem kann man einfach entgegenwirken, aber da muss man sich auch die Frage stellen um wieviele Hidden Neurone man jedes KNN zu Beginn erst erweitert. Zu wenige dürfen es nicht sein, da sie ansonsten wahrscheinlich kaum Auswirkung haben auf die gesamte Struktur. Werden zu viele Neurone hinzugefügt wird der Zufallsfaktor reduziert, die Varianz der verschiedenen Topologien wird damit schon gleich begrenzt. Das kann eventuel sogar dazu führen, dass die Klasse der optimalen Topolgien dadurch nicht mehr gefunden werden kann. Aber das war vielleicht auch gar nicht so wichtig. Das wäre vielleicht eine andere herangehensweise gewesen, die mehr Erfolg versprochen hätte. Schließlich war es ja von von Beginn an das Ziel SKNNs nach einem biologischen Vorbild zu erzeugen das zumindest im groben Sinne eine sehr deutliche Struktur hat. Ob dieser limitierende Ansatz erfolgsversprechender ist bleibt herauszufinden.

Ein weiterer interessanter Punkt ist in Abb. 4.1 zu finden. Die dort gezeigte Trajektorie macht genau das Gegenteil ihrer Aufgabenstellung - es bewegt sich immer von dem Ziel weg. Das ist ein Verhalten, das sehr viele KNNs gezeigt haben und zwar bei allen verschieden Versuchsaufstellungen. Interessanter weise hilft es auch nicht, wenn man die inhibitorischen und die exitatorischen Neuronen bei diesen KNNs vertauscht, um die Trajektorie umzudrehen. In den klassischen Machine Learning methoden wäre so ein Ergebnis fast schon per-

fekt, da man dort ja nur die Vorzeichen ändern müsste, was bei diesem Modell auch nicht funktioniert hat. Das liegt aber wahrscheinlich auch daran, dass alle Input Neurone standardmäßig als exitatorisch behandelt wurden und diese Neuronen nicht einfach so auf inhibitorisch umgestellt werden können. Genau genommen würde es ja auch kein Sinn machen, da es so keine Signalweiterleitung nach der Input Ebene geben könnte.

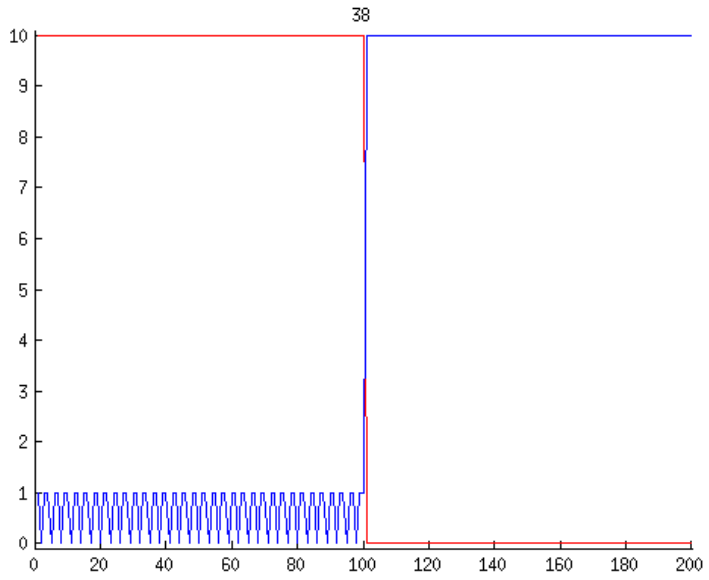

Abbildung 4.1: Die Trajektorie eines simulierten Armes, der von einem KNN gesteuert wird. Dabei ist die Bewegung Genau invertiert, die Trajektorie bewegt sich immer von dem Ziel weg.

4.0.1 Zusammenfassung

Überblickend kann man zu den verschiedenen Problemstellungen sagen, dass der Anfang sehr gut funktioniert hat. Dass das simple Modell funktionieren wird, dass nur die Differenz aus zwei Werten berechnen muss war im Vorfeld sicher schon klar. Bei genauerm Betrachten hätte man auch einfach die Variante aus **??**, das einfache Population Coding mit den Komplexitätsstufen, benutzen können um SKNNs zu generieren. Aber im Endeffekt können KNNs dieses Problem mit nur 2 Hidden Neuronen lösen, in dem einfach die richtigen Neurone die aufwärts Bewegung fördern und gleichzeitig die abwärts Bewegung hindern, wenn sich das Ziel über der aktuellen Position befindent. Wodurch dann die ganze Struktur vom somatosensorischen Cortex sowie dem Motorcortex nicht wirklich dargestellt werden kann. Wenn man die Resultate aus 3.2.1 betrachtet, muss man eigentlich den Schluss ziehen, dass die Verbin-

dung der Output Neurone und der Input Neurone zu wichtig ist, als das man diese wirklich verwerfen kann.
Der Ansatz aus **??** und 3.4 hat im Prinzip den Gedanken verfolgt, dass Neat auf die Initialen Verbindungen der Neurone angewiesen ist. Also wäre es interessant gewesen zu sehen welche Strukturen NEAT erzeugen würde um diese Probleme zu lösen und ob sie ähnlichkeiten zu den Strukturen im Motorcortex und somatosensorischen Cortex aufweisen. Aber um das Problem fordernder zu Gestalten wurde die komplexitäts des Inputs erhöht. Da mittlerweile ja klar ist, dass Neat das Problem mit einem vorverarbeitetem Input einfach lösen kann. Die beiden Ansätze haben leider keine positiven Ergebnisse gezeigt. Die Variante mit dem Rate Coding wäre sowieso erst bei dem Schritt zu den SKNNs relevant geworden, da es dort ein Rauschen gegeben hätte. Das wäre dann nämlich der Fall gewesen in dem das Rate Coding hätte hervorstechen sollen. Der Hybridaufbau der in 3.5 beschrieben wurde hat aller wahrscheinlich nicht geklappt, weil die zwei unterschiedlichen Methoden sich gegenseitig behindert haben. Wenn bestimmte Gewichte über das Reinforcement Learning optimiert wurden, bestand immernoch die Chance, dass der ganze Lernprozess durch eine Mutation, die durch NEAT stattfindet wieder zunichte gemacht wird. Oder ein neues Neuron, das hinzugefügt wird und dadurch die Topologie zwar erweitert, diese aber auch deutlich verändern kann. Die Tatsache, dass diese beiden Methoden in Kombination nicht so wirklich gut funktioniert haben, sieht man sehr deutlich bei der durchschnittlichen Performance in Abb. 3.12. Der Wert auf dem sich die Performance stabilisiert ist in etwa 0.18 was ingesamt eine sehr schlechte Performance darstellt.
Generell wurde davon abgesehen ein Rauschen beim Input Signal zu implementieren. Obwohl es durchaus wünschenswert gewesen wäre wenn die so generierten KNNs eine gewisse Rauschresistenz gehabt hätten, ist es dennoch ein effizienteres Training, wenn die KNNs mit einem reinen Signal trainiert werden. Dies erhöht nnämlich auch weiterhin die Wahrscheinlichkeit, dass Topologien gefunden die primär das Problem lösen können. Man hätte natürlich das Rauschen auch als weitere Komplexitätsstufe betrachten können und mit steigender Komplexität das Rauschen weiter verstärken können um die Resistenz der KNNs zu testen.
Den Effekt der Komplexitätsstufen kann man sehr gut in den Abb. **??** sehen und besonders gut in Abb. 3.10. Jedes mal wenn die durchschnittliche Performance einen großen Sprung nach unten macht, wurde die Komplexität um eine Stufe erhöht. Das bedeutet, dass zwei Output und Input Neuronen hinzugefügt wurden. Den wohl stärkeren Beitrag zu diesen Performancesprüngen haben die zusätzlichen Inputneuronen. Diese bedeuten nämlich, dass die einzelnen Neuronen nicht wieder im gleichen Fall Feuern. Dank dem Population Coding verteilen sich die Momente in denen bestimmte Neurone feuern. Hat z. B. ein Neuron für die Representation von fünf bis zehn gefeuert, feuert es eine Komplexitätsstufe nur noch von fünf bis sieben und ein neues Neuron von

acht bis zehn. Und gerade die neuen Neurone sind dann noch nicht in der Topologie neu integriert. Aber im Versuchsaufbau in **??** haben es KNNs immer wieder geschafft das Problem zu lösen. Betrachtet man aber genauer Abb. 3.4 merkt man, dass die durchschnittliche Performance nach den Sprüngen nach unten zwar wieder zunimmt, aber nicht wieder annähernd hohe Werte wie zu Beginn erreichen kann. Demnach schaffen es nicht viele Individuen sich passend weiterzuentwickeln. Selbst auf der höchstens Komplexitätsstufe bleibt der durchschnittliche Performancewert auf ca. 0.62. Dies kann auch daran liegen, dass NEAT nach so vielen Generationen zu viele unterschiedliche Spezien erzeugt hat, die überlebt haben. Da verschiedene Spezien nicht miteinander konkurieren bleiben auch viele übrig die nur eine durchschnittliche Leistung zeigen. Man könnte sich an das Problem auch anpassen indem man die Anzahl der Individuen erhöht, wenn man weiß, dass NEAT mehr als 1000 Generationen brauchen wird um das Problem zu lösen.

Literaturverzeichnis

[1] Neil A. Campbell and Jane B. Reece. *Biologie*. Kluwer Academic Press, München, 2009.

[2] Jinhu Xiong, Lawrence M Parsons, Jia-Hong Gao, and Peter T Fox. Interregional connectivity to primary motor cortex revealed using mri resting state images. *Human brain mapping*, 8(2-3):151–156, 1999.

[3] Katrin Amunts, Gottfried Schlaug, Lutz Jäncke, Helmuth Steinmetz, Axel Schleicher, Andreas Dabringhaus, and Karl Zilles. Motor cortex and hand motor skills: structural compliance in the human brain. *Human brain mapping*, 5(3):206–215, 1997.

[4] Tianyi Mao, Deniz Kusefoglu, Bryan M Hooks, Daniel Huber, Leopoldo Petreanu, and Karel Svoboda. Long-range neuronal circuits underlying the interaction between sensory and motor cortex. *Neuron*, 72(1):111–123, 2011.

[5] Wikipedia: Motor cortex. https://en.wikipedia.org/wiki/Motor_cortex#/media/File:Human_ Besucht: 25.09.2016.

[6] Alex Kreilinger, Christa Neuper, and Gernot R Müller-Putz. Error potential detection during continuous movement of an artificial arm controlled by brain–computer interface. *Medical and Biological Engineering and Computing*, pages 1–8, 2012.

[7] John K Chapin, Karen A Moxon, Ronald S Markowitz, and Miguel AL Nicolelis. Real-time control of a robot arm using simultaneously recorded neurons in the motor cortex. *Nature neuroscience*, 2(7):664–670, 1999.

[8] IA Basheer and M Hajmeer. Artificial neural networks: fundamentals, computing, design, and application. *Journal of microbiological methods*, 43(1):3–31, 2000.

[9] Alan L Hodgkin and Andrew F Huxley. A quantitative description of membrane current and its application to conduction and excitation in nerve. *The Journal of physiology*, 117(4):500, 1952.

[10] Eugene M Izhikevich et al. Simple model of spiking neurons. *IEEE Transactions on neural networks*, 14(6):1569–1572, 2003.

[11] Alexander Borst and Frédéric E Theunissen. Information theory and neural coding. *Nature neuroscience*, 2(11):947–957, 1999.

[12] Hanspeter A Mallot. *Computational neuroscience*. Springer, 2013.

[13] Apostolos P Georgopoulos, Andrew B Schwartz, Ronald E Kettner, et al. Neuronal population coding of movement direction. *Science*, 233(4771):1416–1419, 1986.

[14] Richard B Stein, E Roderich Gossen, and Kelvin E Jones. Neuronal variability: noise or part of the signal? *Nature Reviews Neuroscience*, 6(5):389–397, 2005.

[15] Martin Spüler, Wolfgang Rosenstiel, and Martin Bogdan. Co-adaptivity in unsupervised adaptive brain-computer interfacing: a simulation approach. In *Proceedings COGNITIVE 2012. The Fourth International Conference on Advanced, Cognitive Technologies and Applications*. Think Mind, 2012.

[16] George L Chadderdon, Samuel A Neymotin, Cliff C Kerr, and William W Lytton. Reinforcement learning of targeted movement in a spiking neuronal model of motor cortex. *PloS one*, 7(10):e47251, 2012.

[17] Martin Spüler, Sebastian Nagel, and Wolfgang Rosenstiel. A spiking neuronal model learning a motor control task by reinforcement learning and structural synaptic plasticity. In *2015 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE, 2015.

[18] Kenneth O Stanley and Risto Miikkulainen. Evolving neural networks through augmenting topologies. *Evolutionary computation*, 10(2):99–127, 2002.

Selbständigkeitserklärung

Hiermit versichere ich, dass ich die vorliegende Masterarbeit selbständig und nur mit den angegebenen Hilfsmitteln angefertigt habe und dass alle Stellen, die dem Wortlaut oder dem Sinne nach anderen Werken entnommen sind, durch Angaben von Quellen als Entlehnung kenntlich gemacht worden sind. Diese Bachelorarbeit wurde in gleicher oder ähnlicher Form in keinem anderen Studiengang als Prüfungsleistung vorgelegt.

Ort, Datum Unterschrift