Georg-August-Universität
Göttingen

III. Physikalisches Institut - Biophysik

Bachelorarbeit

Entwicklung synaptischer Plastizität mit genetischen Algorithmen in neuronalen closed-loop Systemen

Evolution of Synaptic Plasticity using Genetic Algorithms in Neural Closed-Loop Systems

Angefertigt von:
LARS MELCHIOR
aus Göttingen

Göttingen, den 8. Juni 2013

Betreuer: DR. CHRISTIAN TETZLAFF
Erstgutachter: PROF. DR. FLORENTIN WÖRGÖTTER
Zweitgutachter: DR. PORAMATE MANOONPONG

Zusammenfassung

Wir untersuchen in dieser Arbeit die Möglichkeit, mit genetischen Algorithmen Lernregeln für neuronale Netzwerke zu entwickeln. Die Fitness einer Lernregel wird dabei anhand der Leistung eines Netzwerkes mit zufälligen Gewichten in einer von zwei closed-loop Tasks bestimmt. Zusätzlich wird die Möglichkeit untersucht, Lernregeln und Netzwerktopologien mit und ohne versteckten Neuronen parallel zu entwickeln. Im Laufe der Arbeit wird ein Genetischer Algorithmus zusammen mit Vorschriften für Mutation und Rekombination von Lernregeln und Topologien definiert und dieser in 5 unterschiedlichen Simulationen erfolgreich getestet.

Abstract

We explore the possibility of evolving rules for synaptic plasticity using genetic algorithms. The fitness-measure is determined by measuring the performance of a randomly generated network in a closed-loop task. We also explore the possibility of simultaneously developing network topology and plasticity for networks with and without hidden neurons. A genetic algorithm, and operations for mutation and cross-over of learning rules and network topology, are defined and successfully tested in five different simulations.

Inhaltsverzeichnis

1 Vorwort

Genetische Algorithmen bieten ein sehr einfaches und gleichzeitig sehr nützliches Modell natürlicher Evolution von Systemen, denen eine Fitness, also eine Leistung, zugewiesen werden kann. Im Bereich des maschinellen Lernens und der computergestützten Neurowissenschaften haben sich genetische Algorithmen daher als effektives Lernverfahren für analytisch schwer zugängliche Systeme etabliert. Meines Wissens nach wurde jedoch die Möglichkeit, mit genetischen Algorithmen Lernregeln für Modelle natürlicher Systeme zu erzeugen, kaum untersucht. Mit dieser Arbeit möchte ich daher ein einfaches und beliebig erweiterbares System vorstellen, mit dem sich Lernregeln in Form von Funktionalen für neuronale Netzwerke entwickeln lassen. Das System wird dann anhand von zwei einfachen Aufgaben getestet und erweitert, um die Funktionale automatisch zu vereinfachen und Lernregeln und Topologien parallel zu entwickeln.

2 Theoretische Grundlagen

In diesem Abschnitt werden die zum Verständnis dieser Arbeit nötigen Grundlagen vorgestellt. Es werden jedoch grundlegende Kenntnisse in Biologie sowie in Differenzial- und Integralrechnung vorausgesetzt.

2.1 Neuronen

Zunächst folgt eine kurze Einführung in Neuronen und ratenbasierte Modelle neuronaler Netzwerke sowie ein Überblick über bekannte Modelle synaptischer Plastizität. Quellen und weiterführende Literatur dieses Kapitels sind [4] sowie [1].

2.1.1 Struktur

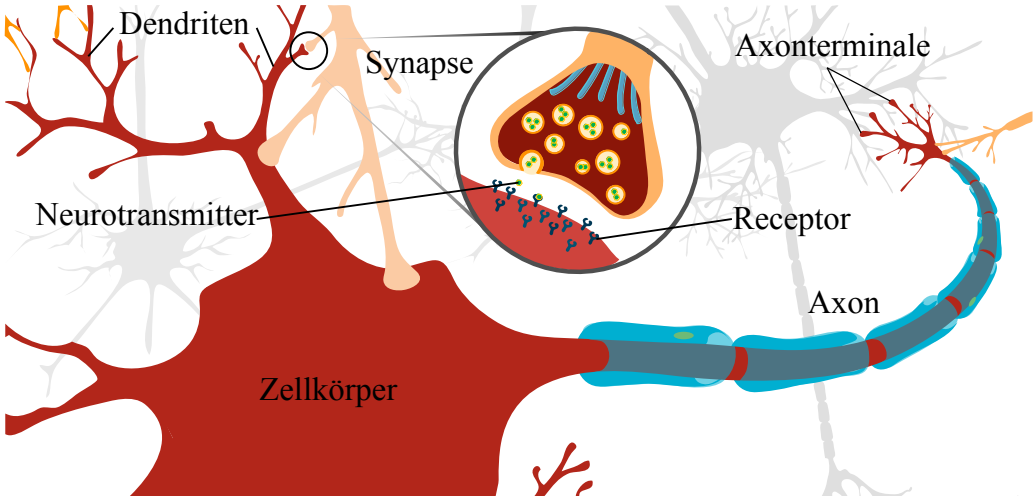


Abbildung 1: Schema eines Neurons. Bei dem hier dargestellten Neuron handelt es sich um ein typisches Motoneuron, welches Signale vom Zentralen Nervensystem zu verschiedenen Körperteilen leitet. Es gibt jedoch eine Vielzahl verschiedener spezialisierter Neuronen mit unterschiedlicher Morphologie und Physiologie. Modifizierte Grafik aus: [18].

Neuronen (oder Nervenzellen) sind Zellen, die der Informationsübermittlung und Verarbeitung von elektrischen Signalen in Tieren und damit auch dem Menschen dienen. Sie sind verantwortlich für die Aufnahme und Verarbeitung von Signalen des Körpers oder der Außenwelt, sowie für die Steuerung des Körpers als Reaktion auf diese Signale [15]. Der Zellkörper des Neurons beinhaltet die Organellen, die für die Lebenserhaltung der Zelle zuständig sind. Über Synapsen an den Dendriten, kurzen Zellfortsätzen mit einer Länge von einigen hundert μm, nimmt das Neuron Signale von anderen Zellen auf. Diese werden mit biochemischen Botenstoffen, sogenannten Neurotransmittern, übermittelt und binden an spezielle Rezeptoren des Neurons, die daraufhin Ionenkanäle öffnen oder schließen (siehe Abbildung 1). Als Ausgabe dient dem Neuron das Axon, ein langer Zellfortsatz, mit einer Länge zwischen <1 mm und 1 m, an dessen Ende sich

Axonterminale befinden, die selbst Neurotransmitter freisetzen können. Mit den Begriffen prä- und postsynaptisch beziehen wir uns auf den Teil einer Synapse, von der ein Signal ausgeht (prä-) oder das ein Signal empfängt (post-).

2.1.2 Aktionspotentiale

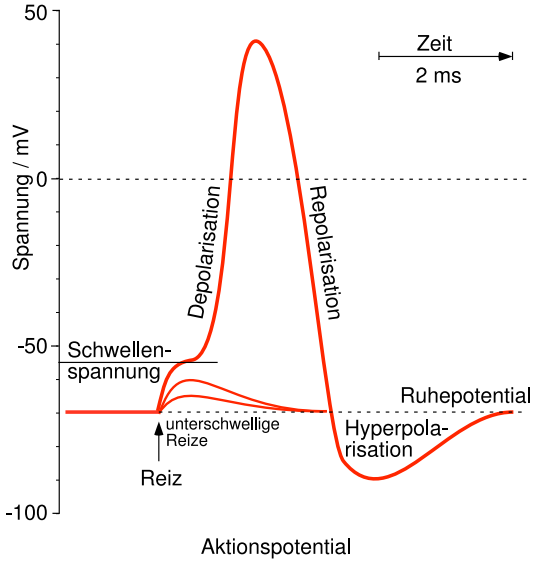


Abbildung 2: Zeitlicher Verlauf eines Aktionspotentials. Überlagert sind zwei unterschwellige Reize, bei denen die Schwellenspannung nicht überschritten wird und sich kein Aktionspotential bildet. Quelle: [16].

Ein Aktionspotential oder Spike ist ein elektrischer Puls von meist etwa 100 mV Stärke und 1 ms Dauer, der entlang des Axons wandert, bis er die Axonterminale erreicht. Dort sorgt das Aktionspotential dafür, dass Neurotransmitter freigesetzt werden, die durch den synaptischen Spalt (den etwa 20 nm großen Zwischenraum zwischen präsynaptischen Axonterminal und postsynaptischer Dendrite [20]) diffundieren und an Rezeptoren einer Dendrite eines anderen Neurons binden. Dadurch öffnen sich kurzzeitig an dem postsynaptischen Neuron Ionenkanäle, durch die, abhängig von der Beschaffenheit der Synapse, positiv geladene Ionen in die Zelle hineinfließen oder aus der Zelle austreten. Hierdurch fließt ein Strom durch den Zellkörper, der das Membranpotential entsprechend erhöht oder erniedrigt. Bei einer Erhöhung sprechen wir dabei von einer exzitatorischen, bei einer Erniedrigung von einer inhibitorischen Synapse. Die Gesamtladung, die nach einem präsynaptischen Spike durch den Zellkörper fließt, bezeichnen wir als die Stärke der Synapse. Regulierende Ionenkanäle senken das Membranpotential langsam zurück auf ein Ruhepotential, welches bei etwa -70 mV relativ zu dem umliegenden Medium liegt. Überschreitet das Membranpotential (zum Beispiel angeregt durch Signale weiterer Neuronen) vorher einen bestimmten Schwellenwert, die Schwellenspannung, kommt es zu einem positiven Rückkoppelungseffekt und ein Aktionspotential wird erzeugt. Das Neuron befindet sich anschließend für wenige Millisekunden in einem Zustand der Hyperpolarisation, in dem es keine weiteren Aktionspotentiale bilden kann. Erst nach dieser kurzen Verzögerung ist die Zelle in der Lage,

weitere Aktionspotentiale zu bilden. Die Dauer eines Aktionspotentials unterscheidet sich zwischen verschiedenen Neuronentypen, liegt jedoch meist in der Größenordnung einer Millisekunde.

2.1.3 Feuerraten

Der Informationsgehalt eines Aktionspotentiales liegt alleine in dem Zeitpunkt seines Auftretens. Eine Folge von Aktionspotentialen zu den Zeiten t_i mit $i \in [1, n]$ lässt sich daher Formal beschreiben als eine Summe diracscher Delta-Distributionen (Siehe Anhang 1):

$$\rho(t) = \sum_{i=0}^{n} \delta(t - t_i).$$

$\rho(t)$ wird die neuronale Antwortfunktion genannt. Da Aktionspotentiale bei identischer Anregung stochastisch verteilt auftreten, wird die resultierende Aktivität eines Neurons oft über dessen Feuerrate $r(t)$ beschrieben:

$$r(t) = \lim_{\Delta t \to 0} \frac{1}{\Delta t} \int_{t}^{t+\Delta t} \overline{\rho(\tau)} \mathrm{d}\tau. \tag{1}$$

Dabei ist $\overline{\rho(t)}$ der Mittelwert aller möglichen Antwortfunktion auf die Anregung. Die Feuerrate hat die Einheit $1/[t]$ und entspricht der Wahrscheinlichkeitsdichte für die Anzahl auftretender Spikes. Aus experimentellen Messdaten kann $r(t)$ nicht direkt bestimmt werden, sondern wird oft genähert, zum Beispiel über die Faltung einer gemessenen Antwortfunktion mit einer Fensterfunktion (einer Funktion die für großes $|t|$ gegen 0 geht).

2.1.4 Neuronale Netzwerke

Im menschlichen Gehirn befinden sich mehr als 10^{11} Neuronen, jedes davon ist durchschnittlich mit tausenden weiteren Verbunden. Diese bilden ein extrem organisiertes Netzwerk, welches alle anderen biologischen Systeme an Komplexität übertrifft. Die Grundstruktur des Netzwerkes wird über die Gene bestimmt, während sich das Gedächtnis und Lernen erst durch aktivitätsabhängige Anpassung der Synapsen und deren Stärke im Laufe des Lebens entwickeln. Die einzelnen Neuronen eines Netzwerkes entstehen zunächst ohne Verbindungen zu anderen Neuronen. Die Eigenschaften eines Neurons werden, wie bei anderen Zellen, über die chemische Zusammensetzung der Außenwelt und Genregulation festgelegt. Anschließend wachsen den Neuronen Dendriten und Axone entlang spezifischer Pfade, die genregulatorisch und chemotaxisch (also entlang eines chemischen Gradienten) aufgebaut werden. Diese zeichnen ein grobes vorläufiges Bild der Verbindungen in dem Netzwerk. In der Schlussphase werden kontinuierlich neue Verbindungen auf- und abgebaut sowie die Stärke der Synapsen verändert.

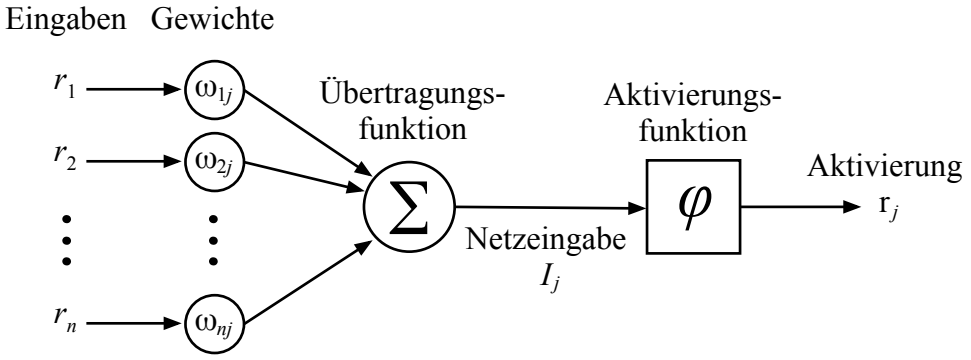


Abbildung 3: Schema des ratenbasierten Neuronmodells in einem neuronalen Netzwerk aus n Neuronen. Die Übertragungsfunktion bildet die gewichtete Summe der Eingaben r_i vom Neuron j, woraus die Aktivierungsfunktion φ die Aktivierung r_j des Neurons bestimmt. Übernommen von [17].

2.1.5 Ratenbasierte Modelle neuronaler Netzwerke

In ratenbasierten Modellen neuronaler Netzwerke aus N Neuronen wird jedem Neuron $i \in [1, N]$ eine Feuerrate $r_i(t)$ zugewiesen. Feuert ein präsynaptisches Neuron i zum Zeitpunkt t', werden an den Dendriten des postsynaptischen Neurons j Ionenkanäle geöffnet, durch die ein Strom in das Neuron fließt. Diesen Stromfluss bezeichnen wir als $I(t) = K(t' + t)\omega_{ij}(t')$. Die Größe $K(t)$ hat die Einheit $1/[t]$ und wird als synaptischer Kern bezeichnet. $\omega_{ij}(t)$ gibt die Stärke oder das Gewicht einer synaptischen Verbindung zwischen Neuron i und j zum Zeitpunkt t an. Man beachte, dass ω_{ij} die Einheit $[I] \cdot [t]$, also Ladung, hat. Für exzitatorische Synapsen, die I erhöhen, ist $\omega_{ij} > 0$ und für inhibitorische, die I verringern, ist $\omega_{ij} < 0$. Der Gesamtstrom $I_j(t)$ durch ein Neuron j, welches nur mit dem Neuron i verbunden ist, wird aus der Faltung der gewichteten präsynaptischen Antwortfunktion mit dem Kern bestimmt:

$$I_j(t) = \int_{-\infty}^{\infty} \omega_{ij}(\tau)\rho_i(\tau)K(t-\tau)\mathrm{d}\tau.$$

Unter der Annahme, dass sich die Effekte der Synapsen linear addieren, kann der Gesamtstrom eines Neurons mit mehreren Dendriten zu weiteren Neuronen $\{i\}$ aus der Summe der Einzelströme gewonnen werden:

$$I_j(t) = \sum_i \int_{-\infty}^{\infty} \omega_{ij}(\tau)\rho_i(\tau)K(t-\tau)\mathrm{d}\tau.$$

In Ratenbasierten Modellen wird die Antwortfunktion $\rho_i(t)$ mit der Feuerrate $r_i(t)$ des Neurons ersetzt:

$$I_j(t) = \sum_i \int_{-\infty}^{\infty} \omega_{ij}(\tau)r_i(\tau)K(t-\tau)\mathrm{d}\tau. \tag{2}$$

Meistens wird ein mit Zeitkonstante τ_s exponentiell abfallender Kern angenommen:

$$K(t) = \frac{e^{-t/\tau_s}}{\tau_s}\theta(t/\tau_s).$$

Dabei ist $\theta(t)$ die Heaviside-Funktion (Siehe Anhang 1). Die Ableitung des Kerns ist:

$$\frac{\mathrm{d}K(t)}{\mathrm{d}t} = \frac{1}{\tau_s}\left(-K(t) + \frac{e^{-t/\tau_s}}{\tau_s}\delta(t/\tau_s)\right).$$

Durch Einsetzten des Kerns in Gleichung (2) und Ableiten erhalten wir eine Differenzialgleichung für I_j:

$$\frac{\mathrm{d}I_j(t)}{\mathrm{d}t} = \frac{1}{\tau_s}\left(-I_j(t) + \sum_i \omega_{ij}(t) r_i(t)\right). \tag{3}$$

Hieran können wir erkennen, dass der Stromfluss einen stabilen Fixpunkt an der Stelle $I_j = \sum_i \omega_{ij} r_i$ besitzt. Die Feuerrate, die sich bei einem Neuron mit konstantem Eingangsstrom I_j einstellt, wird durch die Aktivierungsfunktion $\varphi(I_j) = r_j$ bestimmt, die oft S-förmig (oder sigmoid) genähert wird. Die Feuerrate r_j kann alternativ auch als Abweichung von einer durchschnittlichen (Hintergrund-) Feuerrate gewählt werden. In diesem Fall ist die Aktivierungsfunktion nicht auf positive Werte beschränkt. Vereinfacht können wir hier annehmen, dass die Feuerrate auch bei zeitlich veränderlichem Eingangsstrom durch die Aktivierungsfunktion gegeben ist:

$$r_j(t) = \varphi(I_j(t)). \tag{4}$$

Um nun das zeitliche Verhalten der Feuerrate numerisch zu untersuchen, linearisieren wir zunächst den Eingangsstrom um t_0 für eine kleine Schrittweite Δt:

$$I_j(t_0 + \Delta t) = I_j(t_0) + \left.\frac{\mathrm{d}I_j(t)}{\mathrm{d}t}\right|_{t=t_0} \Delta t + \mathcal{O}(\Delta t^2).$$

Setzen wir für die Ableitung Gleichung (3) ein, erhalten wir:

$$I_j(t_0 + \Delta t) = I_j(t_0) + \frac{\Delta t}{\tau_s}\left(-I_j(t_0) + \sum_i \omega_{ij}(t_0) r_i(t_0)\right) + \mathcal{O}(\Delta t^2). \tag{5}$$

Es bietet sich an, die Schrittweite mit der Zeitkonstante τ_s gleichzusetzen. Dadurch vereinfacht sich Gleichung (5) zu:

$$I_j(t_0 + \Delta t) = \left(\sum_i \omega_{ij}(t_0) r_i(t_0)\right) + \mathcal{O}(\Delta t^2). \tag{6}$$

Die Eingangsströme I_j und Feuerraten r_j mit $j \in [1, N]$ lassen sich in Vektorschreibweise als $(\vec{r})_j = r_j$ und $(\vec{I})_j = I_j$ vereinigen. Wir führen zudem die vektorielle Aktivierungsfunktion $(\vec{\varphi}(\vec{I}))_j = \varphi(I_j)$ und die Gewichtsmatrix Ω ein als:

$$(\Omega(t))_{ij} = \begin{cases} \omega_{ij}(t), & \text{falls eine Synapse von Neuron } i \text{ nach } j \text{ besteht,} \\ 0, & \text{sonst.} \end{cases}$$

Dadurch lässt sich die Summe $\sum_i \omega_{ij} r_i$ als Element des Produktes $(\Omega\vec{r})_j$ schreiben. Die Größen werden nun zeitlich diskretisiert angegeben als $f_i = f(i \cdot \Delta t)$. Dies erlaubt uns schließlich unter Ausnutzung von Gleichung (4) und (6) eine einfache Update-Vorschrift für die Feuerraten anzugeben:

$$\vec{r}_{i+1} = \vec{\varphi}(\Omega_i \vec{r}_i) + \mathcal{O}(\Delta t^2). \tag{7}$$

Das Neuronenmodell ist schematisch auf Abbildung 3 dargestellt.

2.1.6 Synaptische Plastizität

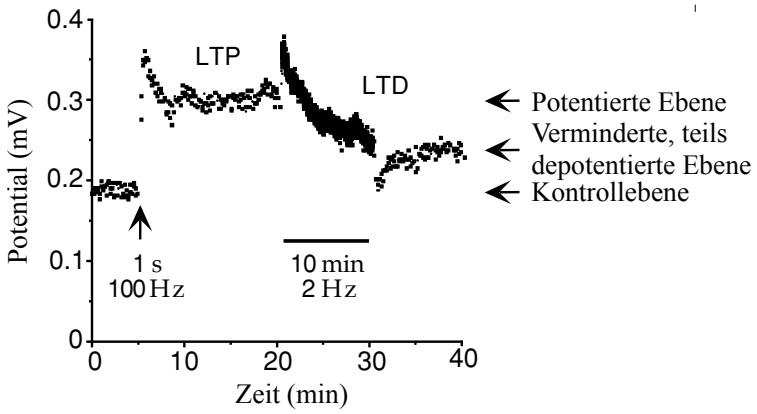


Abbildung 4: LTP und LDP Messungen am Schaffer Collateral einer Hippocampusscheibe einer Ratte. Die Punkte geben das gemessene Antwortpotential bei einer Stimulation konstanter Amplitude an. Bei t = 5 min wird durch eine 1 s dauernde Stimulation eine langfristige Erhöhung (Potentierung) des Antwortpotentials erreicht. Zwischen Minute 20 und 30 sorgt eine Stimulation von 2 Hz für eine langfristige Abnahme des Antwortpotentials. Quelle: [4].

Die aktivitätsabhängige Änderung synaptischer Stärke wird synaptische Plastizität genannt. Diese gilt als eines der hauptverantwortlichen Phänomene für Lernen und Gedächtnis. Messungen an Gehirnarealen haben gezeigt, dass durch eine hochfrequente Stimulation die Stärke synaptischer Verbindungen langfristig zunimmt (LTP, engl: long term potentiation), während bei niederfrequenter Stimulation die Stärke langfristig abnimmt (LTD, engl: long term depression). Siehe hierzu auch Abbildung 4.

Eine der bekanntesten Theorien synaptischer Plastizität ist die 1949 von DONALD OLDING HEBB aufgestellte "Hebbsche Lernregel", nach der die Stärke einer Synapse zwischen zwei Neuronen zunimmt, wenn das präsynaptische Neuron zum Feuern des postsynaptischen Neurons beigetragen hat. Die einfachste denkbare Formulierung der Hebbschen Lernregel ist es, die Änderung des Gewichtes ω_{ij} proportional zu dem Produkt aus prä- und postsynaptischer Aktivität zu setzen:

$$\frac{\mathrm{d}\omega_{ij}}{\mathrm{d}t} = \frac{r_i r_j}{\tau_w}. \tag{8}$$

τ_w ist dabei die Lernrate, die die Geschwindigkeit der Änderung der Gewichte festlegt. Diese Lernregel enthält aber selbst bei Modellen, die eine negative Aktivierung erlauben, eine positive Rückkopplung, die langfristig zu einer stetigen Zunahme des Gewichtes führt und LTP somit nicht erklären kann.
Experimentell wurde gezeigt, dass Gewichte abnehmen, wenn hohe postsynaptische Aktivität von niedriger präsynaptischer Aktivität begleitet wird. Gewichte nehmen hin-

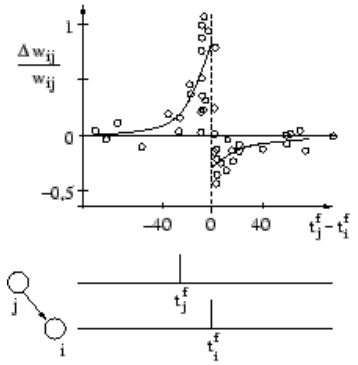


Abbildung 5: Änderung der Stärke einer Synapse ω_{ij} in Abhängigkeit der Zeitdifferenz zwischen präsynaptischem Aktionspotential j und postsynaptischem Aktionspotential i. Experimentelle Messdaten werden als Kreise dargestellt. Es findet nur eine Änderung statt, wenn die Aktionspotentiale ausreichend kurz hintereinander gebildet werden. Die Stärke der Synapse nimmt dann zu, wenn das präsynaptische dem postsynaptischen Aktionspotential vorangeht. Übernommen aus: [6] nach [2].

gegen zu, wenn die postsynaptische Aktivität hoch ist. Eine Regel, die diesen Effekt modelliert, ist die Kovarianz-Regel:

$$\frac{\mathrm{d}\omega_{ij}}{\mathrm{d}t} = \frac{(r_i - \theta_v) r_j}{\tau_w} \tag{9}$$

$$\text{oder: } \frac{\mathrm{d}\omega_{ij}}{\mathrm{d}t} = \frac{r_i (r_j - \theta_v)}{\tau_w}. \tag{10}$$

θ_v ist dabei eine Aktivitätsschwelle, über der LTD zu LTP übergeht. Jedoch kommt es bei diesen Regeln, im Gegensatz zu den experimentellen Daten, selbst ohne prä- bzw. postsynaptische Aktivität zu LTP bzw. LTP. Eine Lernregel, die auch dies berücksichtigt, ist die BCM-Regel:

$$\frac{\mathrm{d}\omega_{ij}}{\mathrm{d}t} = \frac{r_i r_j (r_j - \theta_v)}{\tau_w} \tag{11}$$

Dabei entscheidet der Term $(r_j - \theta_v)$ darüber, ob LTP oder LTD stattfinden soll.
Eine weitere Klasse von Lernregeln basiert auf der experimentell belegten Tatsache, dass die zeitliche Differenz prä- und postsynaptischer Spikes darüber entscheidet, ob LTP oder LTD stattfinden soll (siehe auch Abbildung 5). Kommt der präsynaptische Spike dem postsynaptischen zuvor, so wird die Stärke der Synapse erhöht und ansonsten verringert. Ist der Abstand zwischen zwei Spikes größer als 50 ms, so ändert sich das Gewicht der Synapse nicht. Es gibt jedoch auch Neuronen bei denen dies genau umgekehrt abläuft. Dies wird als “Anti-Hebbsches Lernen” bezeichnet.

2.2 Evolution

Evolution bezeichnet die Veränderung der vererbbaren Merkmale einer Population zwischen Generationen. Die Gesamtheit der vererbbaren Merkmale eines Organismus wird als Genom bezeichnet. Bei Lebewesen wird dieses größtenteils in der Abfolge der Nukleinsäuren in der DNA verschlüsselt. Ein Individuum i kann als eine Dualität zwischen dessen Genotyp (dem zugrundeliegenden genetischen Code) und dessen Phänotyp (der Menge der Merkmale des Individuums) betrachtet werden. Als Population einer Menge S mit Größe n bezeichnen wir einen Vektor aus dem Raum S^n. Wir bezeichnen die Mengen aller Populationen von Phänotypen als **P** und die Menge aller Populationen von Genotypen als **G**. Jedem Individuum einer Population aus **P** und **G** ist auch eine Umweltbedingung zugeordnet. Den Übergang einer Population $p_g \in \mathbf{G}$ von einer Generation g zur nächsten $g+1$ beschreiben wir ähnlich wie [5] durch sukzessive Anwendung von vier Operatoren:

$$\begin{aligned} \text{Epigenese} &: \mathbf{G} \to \mathbf{P} \\ \text{Überleben und Migration} &: \mathbf{P} \to \mathbf{P} \\ \text{Selektion} &: \mathbf{P} \to \mathbf{G} \\ \text{Mutation und Rekombination} &: \mathbf{G} \to \mathbf{G}. \end{aligned}$$

Die Epigenese bildet Genotypen auf Phänotypen ab, wobei Umweltbedingungen die Entwicklung beeinflussen. Überleben und Migration beinhalten Effekte, die die Anzahl der Phänotypen reduzieren und neue Phänotypen aus anderen Populationen hinzufügen. Selektion beinhaltet alle natürlichen und (a-)sexuellen Selektionsprozesse und erzeugt eine Population von elterlichen Genotypen. Hierbei werden durch einen statistischen Prozess Individuen als Eltern ausgewählt, deren Phänotypen anhand bestimmter Eigenschaften besser als andere an die Umweltbedingungen angepasst scheinen. Alle Einflüsse, durch die Genome verändert werden, sind im letzten Schritt, der Mutation und Rekombination, enthalten. Die Evolution sorgt über diese vier Operatoren iterativ für eine Anpassung der Population an die Umwelt. Die treibende Kraft der Evolution ist die Selektion, die ein Individuum i nach einem statistischen Prozess mit einer Wahrscheinlichkeit proportional zur Größe f_i, die wir als Fitness des Individuums bezeichnen, auswählt.

2.3 Genetische Algorithmen

Genetische Algorithmen sind stochastische Suchalgorithmen, die auf eine diskrete Menge X wirken. Das Ziel ist es, ein Element $x_i \in X$ zu finden, das ein Minimum der Funktion $V : X \to \mathbb{R}$ darstellt. Die Idee des genetischen Algorithmus ist es, die Elemente x_i als Individuen einer Population p zu betrachten und $-V(x_i)$ als die Fitness des Individuums. Die Optimierung erfolgt, indem iterativ neue Generationen einer anfänglichen Population erzeugt werden. Dabei wird die nächste Population über Anwendung von genetischen Operationen $G : X^n \to X^m$ erzeugt. Anschaulich sind die genetischen Operationen Funktionen, die aus n Eltern einer Generation m Kinder der folgenden Generation erzeugen. Meistens werden diese am natürlichen Vorbild der Mutation und Rekombination angelehnt. Die Elemente aus X werden daher auch als Genome bezeichnet.

Genetische Algorithmen wirken auf einen diskreten Suchraum und werden daher bevorzugt in Bereichen angewendet, wo gradientenbasierte Suchverfahren nicht möglich sind [14]. Genetische Algorithmen benötigen zudem nur wenig Informationen über das zugrundeliegende Problem: es reicht, das Fitnessmaß $F : X \to \mathbb{R}$ sowie mindestens eine genetische Operation zu definieren. Im Gegensatz zu anderen Optimierungsverfahren wie *Simulated Annealing* muss jedoch in jedem Iterationsschritt die gesamte Population ausgewertet werden, wodurch bei gleicher Iterationszahl deutlich mehr Rechenaufwand benötigt wird. Dennoch werden genetische Algorithmen in vielen Gebieten angewendet, unter anderem als effiziente Modelle natürlicher Evolution [12]. Einige weitere Anwendungsgebiete sind: die Suche nach Quantenalgorithmen, Erzeugung von elektrischen Schaltkreisen, Regelsuche bei zellulären Automaten sowie der Erzeugung von Sortieralgorithmen [11].

3 Methoden

Zur Evolution von Lernregeln definieren wir zunächst den verwendeten genetischen Algorithmus. Anschließend werden die verwendeten Fitnessfunktionen vorgestellt. Zur Entwicklung von neuronalen Netzwerken werden dann topologische Genome und zur Entwicklung von Lernregeln funktionale Genome eingeführt. Für beide Genome müssen dann die verwendeten genetischen Operationen definiert werden.

3.1 Genetischer Algorithmus

Der in dieser Arbeit verwendete Algorithmus ist an der Definition genetischer Algorithmen aus [11] angelehnt. Er besitzt zwei genetische Operatoren: Mutation und Rekombination. Zudem werden, um den Evolutionsdruck zu erhöhen, die von F zugeteilten Fitnesswerte mit der noch zu definierenden Funktion S skaliert. Der Algorithmus benötigt fünf Parameter: N, $g_{\max}$, p_E, p_M und p_R mit der Bedingung, dass $p_M + p_R \leq 1$ ist. Der Algorithmus wird über folgende Vorschrift definiert:

1. Erzeuge eine Anfangspopulation p_0 mit N zufälligen Elementen aus X.
2. Für g von 0 bis $g_{\max} - 1$ werden folgende Schritte ausgeführt:
   (a) Teile jedem Element $x_i \in p_g$ die vorläufige Fitness $f_i' = F(x_i)$ zu.
   (b) Bestimme aus den vorläufigen Fitnesswerten für jedes $x_i \in p_g$ die endgültige Fitness $f_i = S(f_i')$.
   (c) Fülle die Population p_{g+1} mit den $p_E \cdot N$ Elementen höchster Fitness aus p_g.
   (d) Solange die Population p_{g+1} weniger als N Elemente besitzt: führe eine der genetischen Operationen G_k mit Wahrscheinlichkeit p_k und $k \in \{M, R\}$ oder G_X mit Wahrscheinlichkeit $1 - p_M - p_R$ aus:
      G_M: Mutation: Selektiere ein Element aus p_g und füge es mit einer Mutation zu p_{g+1} hinzu.
      G_R: Rekombination: Selektiere zwei Elemente aus p_g und füge zwei Elemente zu p_{g+1} hinzu, die sich Eigenschaften der beiden selektierten Elemente teilen.
      G_X: Reproduktion: Selektiere ein Element aus p_g und füge es zu p_{g+1} hinzu.
3. Das Genom mit höchster Fitness aus der Population $p_{g_{\max}}$ ist das Ergebnis der Simulation.

Um Elemente hoher Fitness nicht zu verlieren, werden die $p_E \cdot N$ besten Lösungen einer Generation p_n in die nachfolgende Generation p_{n+1} übernommen. Dieser Schritt wird als Elitarismus bezeichnet.
Da in einer Population mit hoher durchschnittlicher Fitness der evolutionäre Druck abnimmt (schlechte Lösungen werden etwa genauso häufig selektiert wie gute) skalieren wir die zunächst von F ermittelten Fitness-Werte f_i' mit der Funktion S. Die Skalierung S hat die Form:

$$S(f_i') = \begin{cases} 1 + \frac{f_i' - \overline{f'}}{2\sigma}, & \text{falls } \sigma \neq 0 \\ 1, & \text{sonst.} \end{cases}$$

$\overline{f'}$ ist dabei der Mittelwert und σ die Standardabweichung der vorläufigen Fitness der Population. Diese Operation wird auch als σ-Skalierung bezeichnet und wurde in [12] eingeführt.

3.2 Fitnessfunktionen

Zum Entwickeln von Lernregeln müssen die Aufgaben (Tasks) definiert werden, die von den neuronalen Netzwerken gelöst werden sollen. Hierbei benutzten wir für die Netzwerke auch den Begriff Agent. Wir haben uns für sehr einfache und allgemeine Aufgaben entschieden. Beim ersten Task sollte ein Netzwerk bestenfalls lernen, die Eingabeneuronen (Neuronen, dessen Aktivität von der Außenwelt vorgegeben wird) direkt auf bestimmte Ausgabeneuronen (Neuronen, dessen Aktivität ausgelesen wird und die damit direkt Einfluss auf die Außenwelt nehmen können) zu leiten. Neuronen, die weder zu Ein- noch Ausgabeneuronen gehören, bezeichnen wir als versteckte Neuronen. Die Zahl der versteckten Neuronen nennen wir V. Im zweiten Task sollen Agenten in einer Energielandschaft möglichst effizient navigieren. Da in den Simulationen ein Agent seine Außenwelt beeinflussen kann und damit auch Einfluss auf seine Eingaben nimmt, sprechen wir von einem closed-loop System. Dies steht im Gegensatz zu open-loop Systemen, bei denen ein Agent keinen Einfluss auf seine Außenwelt ausüben kann.

3.2.1 Simulationen

Die neuronalen Netzwerke werden mit der Updatevorschrift aus Gleichung 7 für insgesamt T Zeitschritte simuliert. Nach jedem Zeitschritt werden alle Gewichte des Netzwerks entsprechend der zugehörigen Lernregel aktualisiert. In Simulationen, bei denen eine Topologie entwickelt wird, nimmt das Netzwerk direkt die Topologie als Gewichtsmatrix an. Ansonsten werden 50 Simulationen mit zufällig generierten Gewichtsmatrizen durchgeführt und der Mittelwert als Fitness ausgegeben. Alle nicht zu Inputneuronen gehörende Einträge der generierten Gewichtsmatrizen werden dabei auf Zufallszahlen aus dem Intervall $[0,1]$ gesetzt. Die Aktivitäten werden zu Beginn einer Simulation auf 0 gesetzt.

Alle Simulationen und Algorithmen wurden in C++ geschrieben. Teile des Quellcodes befinden sich im Anhang 5.

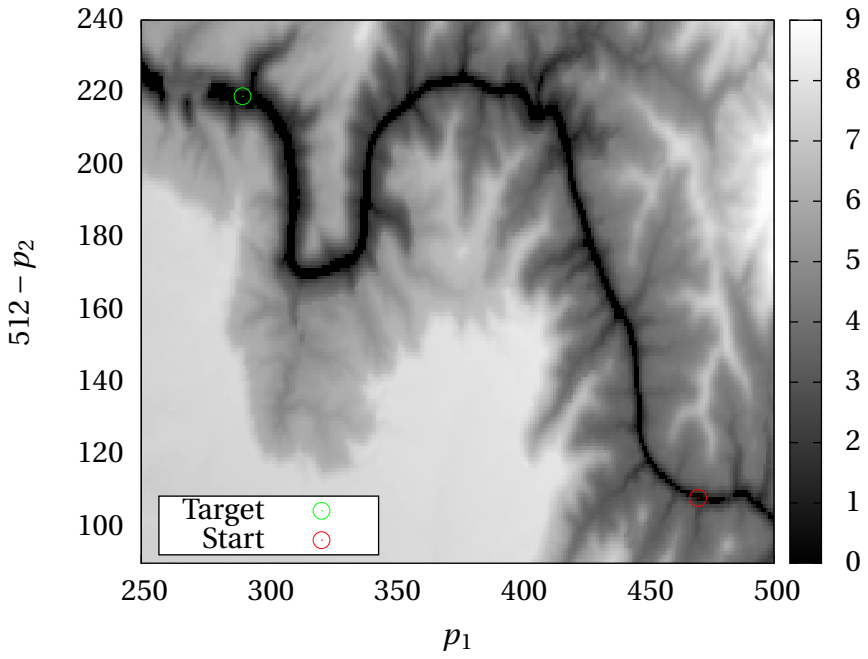


Abbildung 6: Die in Task 2 verwendete Energielandschaft mit Start- und Zielpunkt. Außerhalb des dargestellten Ausschnittes beträgt der Wert der Energielandschaft 10.

3.2.2 Task 1: Einzelnes Target

Dem Netzwerk wird eine zufällige Position $\vec{p}_0$ auf einem dimensionslosem Feld der Größe 2×2 mit dem Zentrum $\vec{z}$ zugeteilt. Zwei Neuronen des Netzwerkes werden als Eingabeneuronen und zwei als Ausgabeneuronen ausgewählt. Die Aktivität der Eingaben wird in jedem Zeitschritt t auf die x- und y-Koordinate des Abstandvektors $\vec{d}_t = \vec{z} - \vec{p}_t$ von der Position des Agenten $\vec{p}_t$ zu dem Zentrum des Feldes $\vec{z}$ gesetzt. Die Werte der beiden Ausgabeneuronen werden in den Vektor $\Delta\vec{p}_t$ eingetragen, der die Positionsänderung im Folgenden Zeitschritt angibt: $\vec{p}_{t+1} = \vec{p}_t + \Delta\vec{p}_t$. Das Netzwerk erhält schließlich die Fitness:

$$F(x_i) = \frac{1}{T}\sum_{t=1}^{T} e^{-|\vec{d}_t|^2}.$$

Die Funktion ist so gewählt, dass die maximal erreichbare Fitness 1 beträgt und Netzwerke, die schneller zum Zentrum laufen, eine höhere Fitness erhalten. Bestenfalls würde das Netzwerk lernen, die Eingabeneuronen direkt mit den entsprechenden Ausgaben zu vernetzen.

3.2.3 Task 2: Energielandschaft

Das Netzwerk soll lernen, in einer Energielandschaft möglichst effizient zu einem Target $\vec{z}$ zu navigieren. Die Engergielandschaft besteht hierbei aus quadratischen Feldern mit der Kantenlänge 1, denen ein Energiewert $E(\vec{p}) \in [0, 10]$ zugewiesen ist. Dem Netzwerk wird eine Startposition $\vec{p}_0$ zugewiesen. Als Eingabe erhält das Netzwerk im Zeitschritt t den Abstandsvektor $\vec{d}/D$ zum Target, wobei D die Diagonallänge der Ener-

gielandschaft entspricht, sowie die Energiedifferenz des Feldes von $\vec{p}_t$ zu den 4 angrenzenden Feldern in vertikaler und horizontaler Richtung. Insgesamt werden also 6 Eingabeneuronen verwendet. Es werden zwei weitere Neuronen als Ausgabeneuronen festgelegt, die die Positionsänderung $\Delta\vec{p}_t$ angeben. Die Fitnessfunktion soll bewerten, wie energieeffizient das Netzwerk zum Ziel navigiert ist. Diese wird festgelegt auf:

$$F(x_i) = \exp\left(-T^{-1}\sum_{t=1}^{T} E(\vec{p}_t)\right)\frac{2-|\vec{d}_T|/D}{2+0.1\,\langle\bar{\omega}\rangle^2}.$$

Dabei soll der Term $0.1\,\langle\bar{\omega}\rangle^2$, mit dem Zeitlich und über alle synapsen ω_{ij} des Netzwerks gemitteltem Gewicht $\langle\bar{\omega}\rangle$, den Suchraum auf nicht-divergente Lernregeln einschränken. Die Fitness ist so gewählt, dass die maximal erreichbare Fitness 1 beträgt.

Das Profil der Energielandschaft stammt von einem Höhenprofil des Grand-Canyons, welches unter [8] gefunden wurde. Die Energielandschaft ist in Abbildung 6 dargestellt.

3.3 Topologieentwicklung

Das topologische Genom T eines neuronalen Netzwerkes stellen wir als $n \times m$ Matrix dar, wobei n die Gesamtzahl der Neuronen und m die Anzahl der Neuronen ist, die keine Eingabeneuronen sind. Wir wählen T als:

$$(T)_{ij} = \begin{cases} 1, & \text{falls eine exzitatorische Synapse zwischen dem Neuron } i \text{ und } j \text{ besteht,} \\ -1, & \text{falls eine inhibitorische Synapse zwischen dem Neuron } i \text{ und } j \text{ besteht,} \\ 0, & \text{sonst.} \end{cases}$$

Mit T wird also neben der Topologie auch das Vorzeichen der Einträge der Gewichtsmatrix des Netzwerkes vererbt. Um T auf eine Gewichtsmatrix abzubilden, werden die fehlenden oberen $n-m$ Reihen als die der $n \times n$ Einheitsmatrix interpretiert.

3.3.1 Mutation

Bei der Mutation wird ein zufälliger Eintrag $(T)_{ij}$ des topologischen Genoms ausgewählt und durch einen zufälligen Wert aus der Menge $\{-1,0,1\}$ ersetzt. Dieser Vorgang wird mit der Wahrscheinlichkeit 0,5 wiederholt, so dass sich mehrere Einträge gleichzeitig verändern können. Im Mittel werden so 2 Einträge verändert.

3.3.2 Rekombination

Bei der Rekombination zweier topologischer Genome T^A und T^B werden vier zufällige Zahlen $i,k \in [1,n]$ sowie $j,l \in [j,m]$ mit $i+j\cdot m < k+l\cdot m$ ausgewählt und alle Elemente $(T^A)_{ab}$ und $(T^B)_{ab}$ vertauscht, bei denen $i+j\cdot m \le a+b\cdot m \le k+l\cdot m$ gilt. Anschaulich bedeutet dies, dass ein zusammenhängender Teil der Matrix T^A durch den gleichen Teil der Matrix T^B vertauscht wird und umgekehrt. Dies ist beispielhaft auf Abbildung 7 dargestellt.

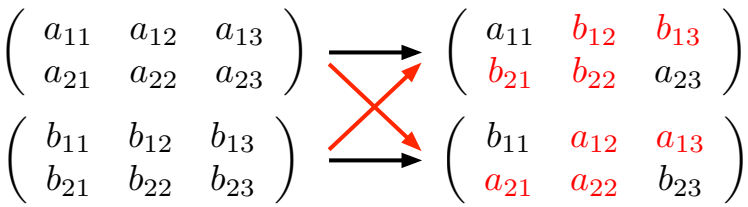
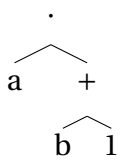


Abbildung 7: Beispiel für die Rekombination zweier topologischer Genome. Hierbei wurden die Kreuzungspunkte $\{i, j\} = \{1,2\}$ und $\{k, l\} = \{2,2\}$ gewählt.

3.4 Funktionalentwicklung

Funktionale werden als Genom kodiert, indem diese in einer Baumstruktur dargestellt werden, deren Blätter Zahlen oder Variablen repräsentieren. Die inneren Knoten des Baumes sind Funktionen, deren Argumente den Knoten der abzweigenden Äste entsprechen. Das Funktional $a \cdot (b+1)$ wird beispielsweise als



dargestellt. Die Menge der kodierbaren Lernregeln $\mathbb{L}'$ entspricht nun der Menge aller gerichteten Bäume, deren innere Knoten aus einer Menge von binären Operatoren $\mathbb{F}$ und deren Blätter aus einer Menge von Variablen $\mathbb{V}$ oder Zahlen $\mathbb{Z}$ stammen. Wir definieren nun einen Wahrscheinlichkeitsraum $\mathbb{L}$ mit der Ergebnismenge $\mathbb{L}'$, dessen Maß über eine stochastische kontextfreie Grammatik $(\{E\}, (\mathbb{Z} \cup \mathbb{V} \cup \mathbb{F}), P, E)$ definiert wird (siehe Anhang 2 für eine Definition kontextfreier Grammatiken). Wir wählen die Produktionsregeln P als

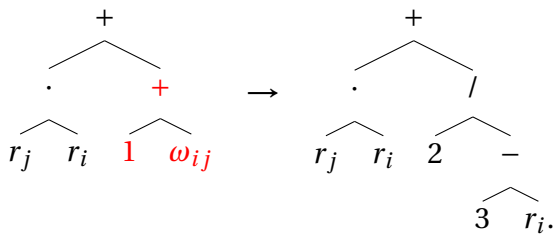
$$\begin{array}{lll} E & \leftarrow & x \in \mathbb{Z} \quad : p_{\mathbb{Z}} \\ E & \leftarrow & x \in \mathbb{V} \quad : p_{\mathbb{V}} \\ E & \leftarrow & E\ (x \in \mathbb{F})\ E \quad : p_{\mathbb{F}} \end{array}$$

mit den Wahrscheinlichkeiten $p_{\mathbb{Z}}$, $p_{\mathbb{V}}$ sowie $p_{\mathbb{F}} \in [0, 1/2)$. Über diese lässt sich unter anderem der Erwartungswert (siehe Anhang 1) der Knotenzahl bzw. Symbolzahl n einer zufälligen Lernregel aus $\mathbb{L}$ festlegen: $E(n) = 1/(1-2p_{\mathbb{F}})$ (Beweis im Anhang 3). Die Grammatik liefert zudem eine Konstruktionsvorschrift, mit der sich zufällige Elemente aus $\mathbb{L}$ generieren lassen.

3.4.1 Mutation

Die Mutation wird nun als Operation $\mathbb{L}' \to \mathbb{L}'$ definiert. Dabei wird ein zufällig ausgewählter Knoten und dessen Teilbaum durch ein zufälliges Element aus $\mathbb{L}$ ersetzt.

Als Beispiel wird nun die Lernregel $f = r_j \cdot r_i + (1 + \omega_{ij})$ mutiert. Dabei wird das rot markierte $1 + \omega_{ij}$ durch die Lernregel $2/(3 - r_i)$ ersetzt. In der Baumstruktur wird die Operation folgendermaßen dargestellt:

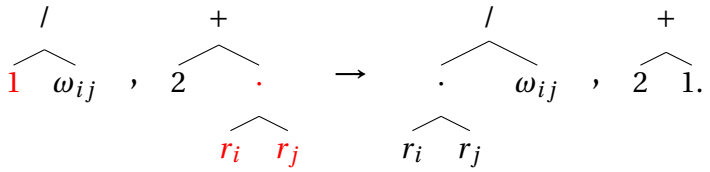


Die mutierte Lernregel f' entspricht also dem Ausdruck $f' = r_j \cdot r_i + 2/(3 - r_i)$.

3.4.2 Rekombination

Die genetische Operation Rekombination wirkt auf zwei Lernregeln f_a und f_b und vertauscht zwei zufällige Knoten der Lernregeln.

Als Beispiel betrachten wir den Fall $f_a = 1/\omega_{ij}$ und $f_b = 2 + r_i \cdot r_j$. In der Baumstruktur würde die Operation folgendermaßen aussehen:



Die resultierenden Lernregeln sind also nun $f'_a = (r_i \cdot r_j)/\omega_{ij}$ und $f'_b = 2 + 1$.

3.4.3 Energiekosten

Wie sich in Kapitel 4.1.1 herausstellt, wachsen die Längen der Lernregeln ohne Einschränkungen stark an. Da wir jedoch die möglichst einfachste Darstellung einer Lernregel suchen, werden die Fitnessfunktionen durch die hundertste Wurzel von m, der benötigten Anzahl an Symbolen zur Darstellung der Regel, geteilt:

$$F'(x_i) = \frac{F(x_i)}{m^{1/100}}.$$

Hierdurch wird bei zwei Lernregeln gleicher Fitness $F(x_i)$ die kürzere Lernregeln bevorzugt.

3.5 Gemeinsame Entwicklung von Topologie und Funktional

Wollen wir nun Topologie und Lernregel zusammen entwickeln, stellen wir das Genom als Tupel (T, f) mit Topologischen Genom T und Lernregel f dar. Bei der Mutation werden mit Wahrscheinlichkeit 0,5 statt einem Element beide Elemente des Tupels mutiert. Bei Rekombination zweier Tupel (T_1, f_1) und (T_2, f_2) werden zunächst beide Einzeleinträge rekombiniert und anschließend mit Wahrscheinlichkeit 0,5 f_1 und f_2 vertauscht.

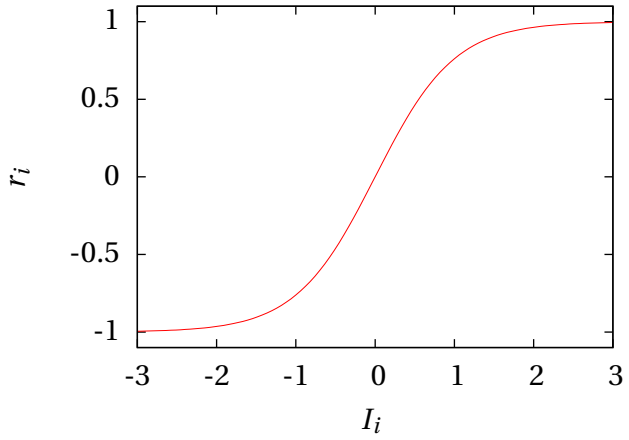


Abbildung 8: Die Aktivierung eines Neurons i in Abhängigkeit des Eingangsstromes I_i unter Annahme der Aktivierungsfunktion $\varphi(I_i) = \tanh(I_i)$.

3.6 Parameterwerte

In der folgenden Tabelle sind nun alle Parameter zusammengefasst. Falls nicht anders erwähnt, wurden in den Simulationen die hier eingetragenen Werte verwendet:

Name	Wert
N	250
$g_{max_{\text{Task 1}}}$	1000
$g_{max_{\text{Task 2}}}$	10000
$T_{\text{Task 1}}$	500
$T_{\text{Task 2}}$	3000
$\mathbb{Z}$	$[1, 10]$
$\mathbb{V}$	$\{\omega_{ij}, r_i, r_j\}$
$\mathbb{F}$	$\{+, -, \cdot, /\}$
$p_{\mathbb{Z}}$	0,4
$p_{\mathbb{V}}$	0,3
$p_{\mathbb{F}}$	0,3
p_E	0,1
p_M	0,5
p_R	0,5
V	0
Energiekosten	ja
$\varphi(I_i)$	$\tanh(I_i)$ (siehe Abbildung 8).

4 Ergebnisse

In den folgenden Kapiteln werden die Ergebnisse der durchgeführten Simulationen in chronologischer Reihenfolge präsentiert. In Kapitel 4.1.1 wird zunächst Task 1 ohne Energiekosten durchgeführt, wobei sich sehr große Lernregeln ergeben. In Kapitel 4.1.2 werden daher Energiekosten eingeführt, die erfolgreich die Größe der Lernregeln reduzieren. In Kapitel 4.2.1 wird versucht, eine Lernregel zu finden, mit der das Netzwerk effizient durch die Energielandschaft navigiert. Erst in Kapitel 4.2.2.1 kann dies tatsächlich erreicht werden, jedoch nur mit gemeinsamer Entwicklung einer Topologie. In Kapitel 4.2.2.2 wird schließlich untersucht, wie versteckte Neuronen die Entwicklung von Lernregeln beeinflussen.

4.1 Task 1: Einzelnes Target

4.1.1 Ohne Energiekosten

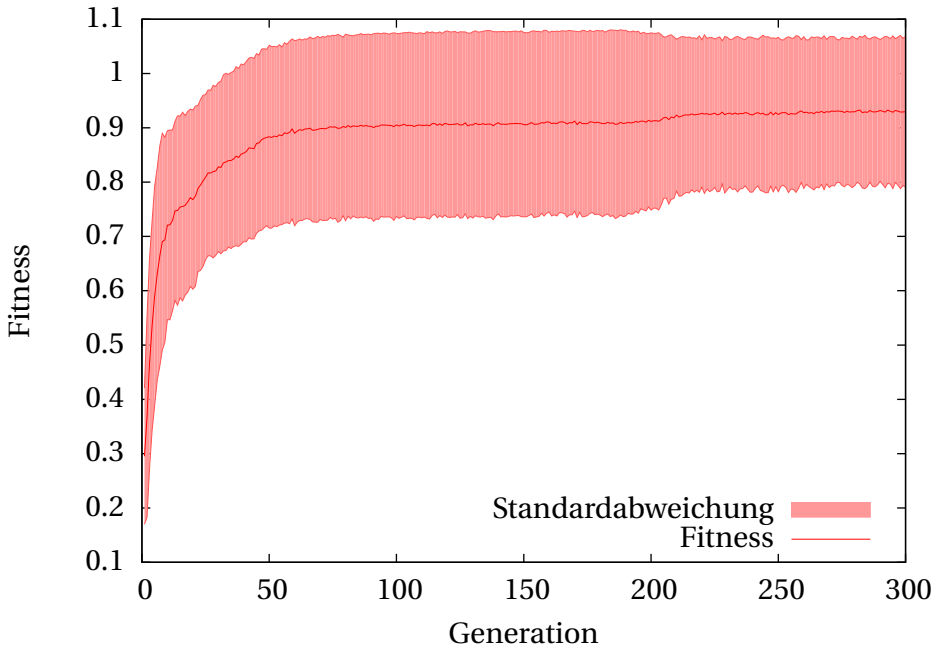
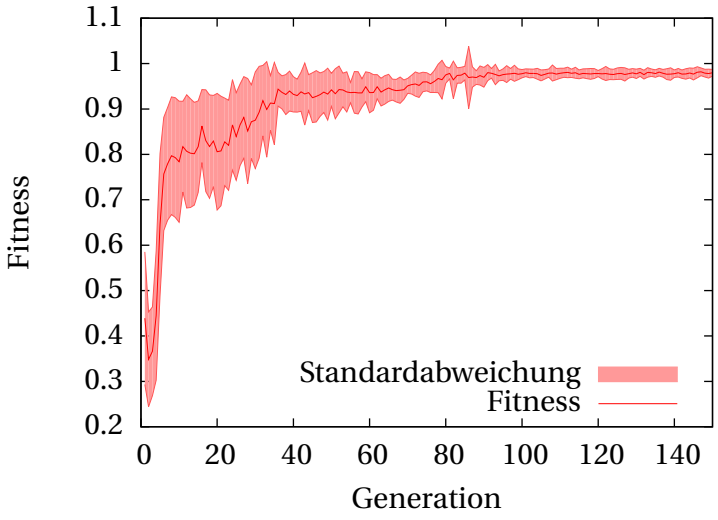


Abbildung 9: Ausschnitt des mittleren Fitnessverlaufes des besten Genoms von 50 Durchführungen des genetischen Algorithmus mit Task 1, $V = 0$ und ohne Energiekosten. Die Fitness verändert sich nach den dargestellten 300 Generationen bis zur 1000. Generation kaum und beträgt dabei 0,94 (12).

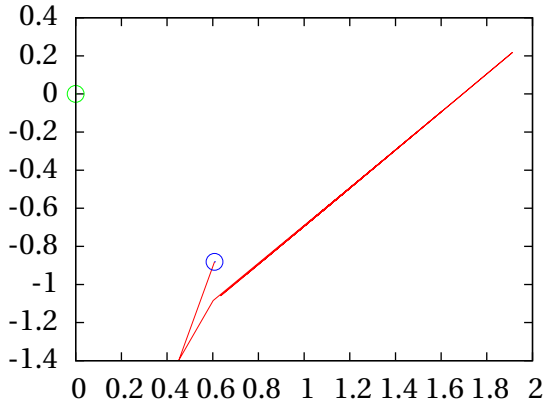
In diesem Abschnitt wird die Evolution von Lernregeln mit der Fitnessfunktion aus Task 1 mit $V = 0$ ohne Energiekosten untersucht. Die mittlere Fitness des besten Genoms aus jeweils 50 Durchführungen ist in Abbildung 9 in Abhängigkeit der Generation dargestellt. Es stellt sich nach etwa etwa 220 Generationen ein bis zur 1000. Generation nahezu konstanter Fitnesswert von 0,94 (12) ein.

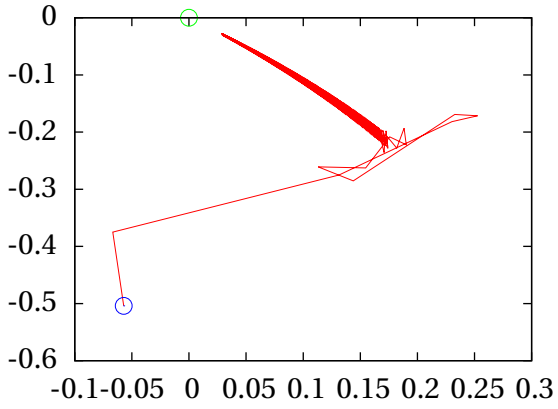
Wir betrachten im Folgenden nur die Durchführung mit höchster End-Fitness. Der mittlere Fitnessverlauf dieser Simulation ist in Abbildung 10 A dargestellt. Hierbei wird bereits nach etwa 100 Generationen ein, bis zur 1000. Generation nahezu konstanter, Fitnesswert von 0,9848 (93) erreicht. In Abbildung 10 B sind auch Trajektorien der besten Lernregel zu verschiedenen Generationen abgebildet. Da hier noch keine Energiebeschränkung für die Länge der Lernregeln eingeführt ist, sind die entwickelten Lernregeln sehr komplex und lang. Bei Generation 40 beträgt die Länge der besten Lernregel 109 Zeichen. Bei Generation 80 sind es 361 und bei Generation 1000 bereits 783 Zeichen. Da sich diese aufgrund vieler verschachtelter Brüche kaum vereinfachen lassen, sind die Lernregeln nur beispielhaft im Anhang 4 für die Generationen 90 und 1000 angegeben.

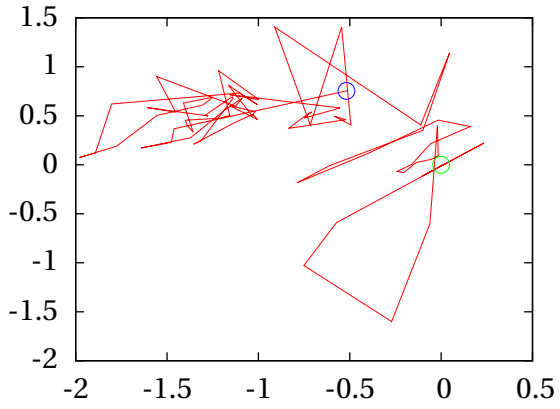
A

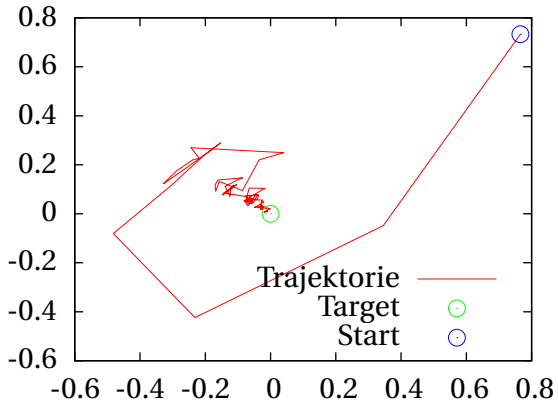


B


Trajektorie nach 20 Generationen


Trajektorie nach 40 Generationen


Trajektorie nach 60 Generationen



Trajektorie nach 80 Generationen

Abbildung 10: A und B: Übersicht der Besten aus 50 Durchführungen des genetischen Algorithmus mit Task 1 ohne Energiekosten. Unter A ist die Fitness des besten Genoms in Abhängigkeit der Generation dargestellt. Die Standardabweichung wird aus den 50 Simulationen von Task 1 mit unterschiedlichen Startnetzwerken gewonnen. Die Fitness ist ab der 100. Generation nahezu konstant bei 0,9848 (93). Unter B sind beispielhafte Trajektorien der besten Lernregel zu verschiedenen Generationen dargestellt. Auf der Abszisse ist dabei die dimensionslose Koordinate p_x und auf der Ordinate die Koodinate p_y der Position des Agenten $\vec{p}$ aufgetragen.

4.1.2 Mit Energiekosten

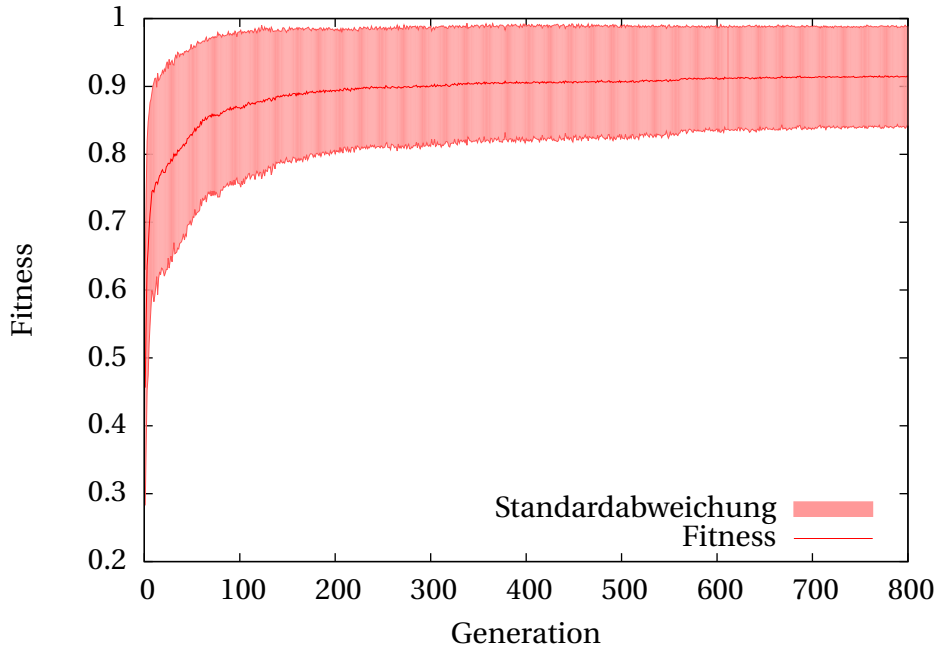


Abbildung 11: Ausschnitt des mittleren Fitnessverlaufes des besten Genoms von 50 Durchführungen des genetischen Algorithmus mit Task 1 mit Energiekosten. Die Fitness erreicht dabei nach etwa 600 Generationen nahezu den Endwert von 0,915 (73).

Nun wird die Simulation aus dem vorherigen Abschnitt mit Energiekosten wiederholt. Der mittlere Verlauf der besten Fitness ist in Abbildung 11 dargestellt. Die endgültige mittlere Fitness beträgt 0,915 (73).

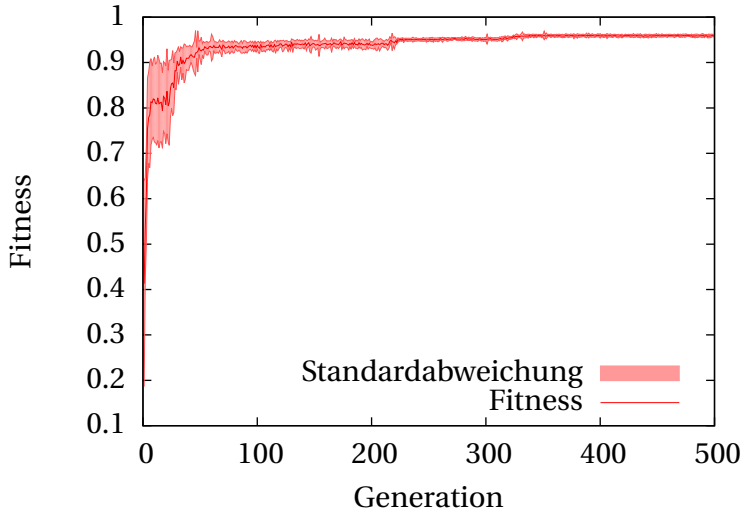
Die Durchführung mit höchster End-Fitness erreicht eine Fitness von 0,9591 (20). Der Verlauf der Fitness, sowie Trajektorien zu verschiedenen Zeiten dieser Durchführung, sind in Abbildung 12 dargestellt. Durch die Energiekosten werden Längen der Lernregeln größtenteils auf deutlich unter 100 Zeichen beschränkt. Die Lernregel mit höchster Fitness lautet:

$$\Delta\omega_{ij} = (((((((r_i-\omega_{ij})/3)-\omega_{ij})+3)+(r_i/((((((2/7)/10)/7)/(r_j\cdot(3-10)))/9)-r_j)))/3)-\omega_{ij})$$

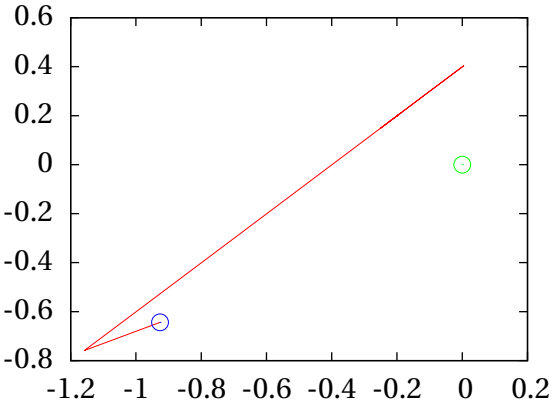
$$= 1 + r_i - \frac{5145 r_i r_j}{1+15435 {r_j}^2} - \frac{13\omega_{ij}}{9}. \tag{12}$$

Diese Lernregel hat wegen der Energiekosten eine niedrigere Fitness als die im ersten Abschnitt (0,984 (18)), würde jedoch ohne Energiekosten eine höhere Fitness von 0,9939(35) erhalten.

A

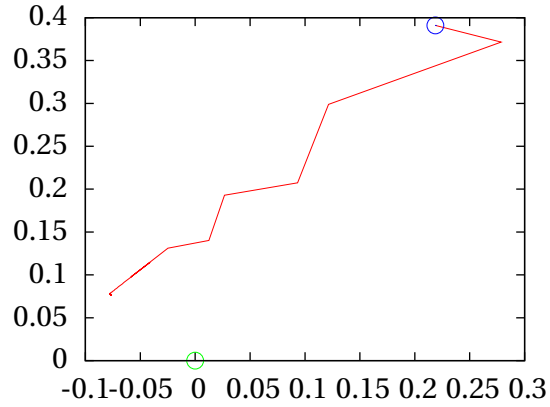


B



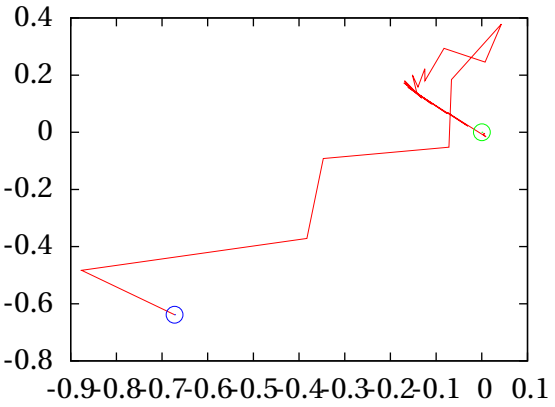
Trajektorie nach 20 Generationen

$$\Delta\omega_{ij} = 3/5 - \omega_{ij}$$



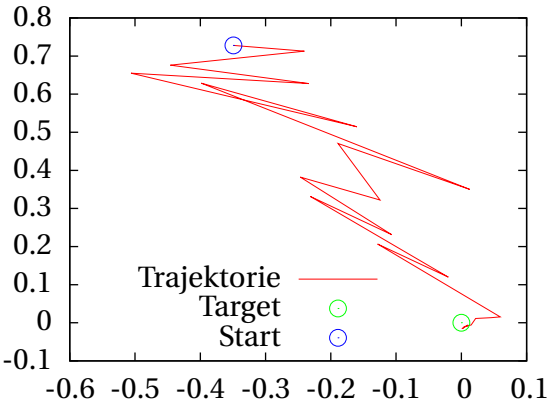
Trajektorie nach 40 Generationen

$$\Delta\omega_{ij} = \frac{1}{3}\left(2 + \frac{r_i}{-r_j + \frac{1}{360}\left(\frac{1}{120-8r_j} + r_j\right)} - 3\omega_{ij}\right)$$



Trajektorie nach 500 Generationen

$$\Delta\omega_{ij} = \frac{1}{3}\left(3 + r_i + \frac{r_i}{\frac{1}{11664(-4+r_j)r_j} - r_j} - 4\omega_{ij}\right)$$



Trajektorie nach 1000 Generationen

$$\Delta\omega_{ij} = 1 + r_i - \frac{5145 r_i r_j}{1 + 15435 {r_j}^2} - \frac{13\omega_{ij}}{9}$$

Abbildung 12: A und B: Zusammenfassung der besten aus 50 Durchführungen des genetischen Algorithmus mit Task 1 mit Energiekosten. Unter A ist die Fitness des besten Genoms in Abhängigkeit der Generation dargestellt. Die Standardabweichung wird aus den 50 Simulationen der Task gewonnen. Die Fitness erreicht ab Generation 400 bereits fast den Endwert von 0,9591 (20). Unter B sind beispielhafte Trajektorien zusammen mit der besten Lernregel zu verschiedenen Generationen dargestellt. Auf der Abszisse ist dabei die dimensionslose Koordinate p_x und auf der Ordinate die Koodinate p_y der Position des Agenten $\vec{p}$ aufgetragen.

4.2 Task 2: Energielandschaft

4.2.1 Ohne Vererbung der Topologie

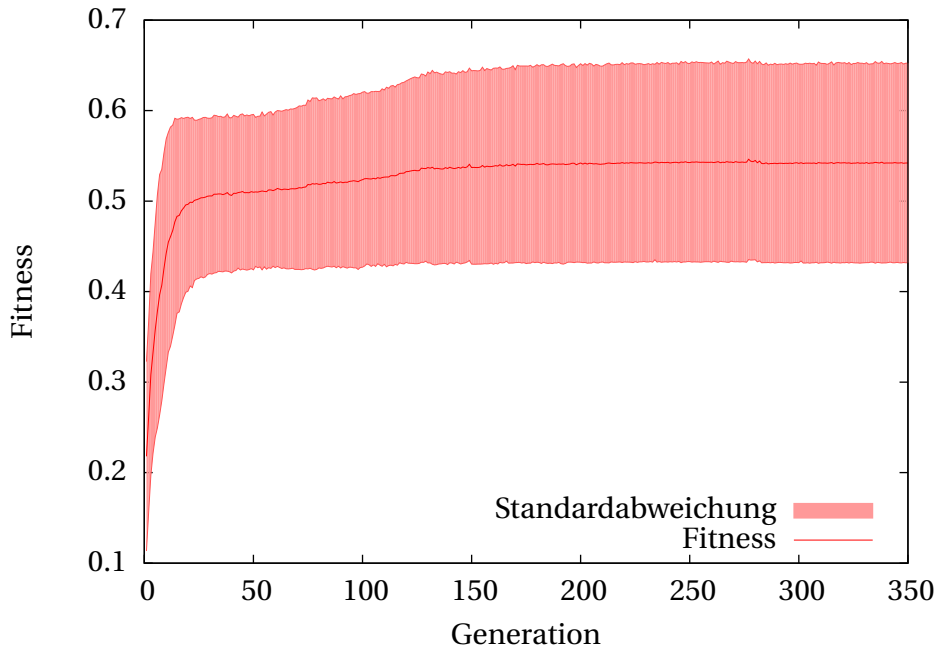


Abbildung 13: Ausschnitt der mittleren Fitness des besten Genoms von 50 Durchführungen des genetischen Algorithmus mit Task 2, g_{max}=1000 und mit Energiekosten. Die Fitness erreicht nach etwa 250 Generationen nahezu den Endwert von 0,54 (22).

In diesem Abschnitt wird die Evolution von Lernregeln mit der Fitnessfunktion aus dem zweiten Task mit g_{max}=1000 und mit Energiekosten untersucht. Die mittlere Fitness des besten Genoms aus jeweils 50 Durchführungen ist in Abbildung 13 in Abhängigkeit der Generation geplottet. Nach den 1000 Generationen wird eine mittlere Fitness von 0,54 (22) erreicht.

Die Durchführung mit höchster End-Fitness erreicht eine Fitness von 0,642 (12). Der Verlauf dieser Fitness, sowie die dazugehörige Trajektorie ist in Abbildung 14 dargestellt. Die entwickelten Lernregeln sind trotz Energiekosten stark angewachsen und deswegen in den Anhang 4 verschoben. Zudem lässt sich an der Trajektorie erkennen, dass Informationen über die Energielandschaft praktisch nicht genutzt werden.

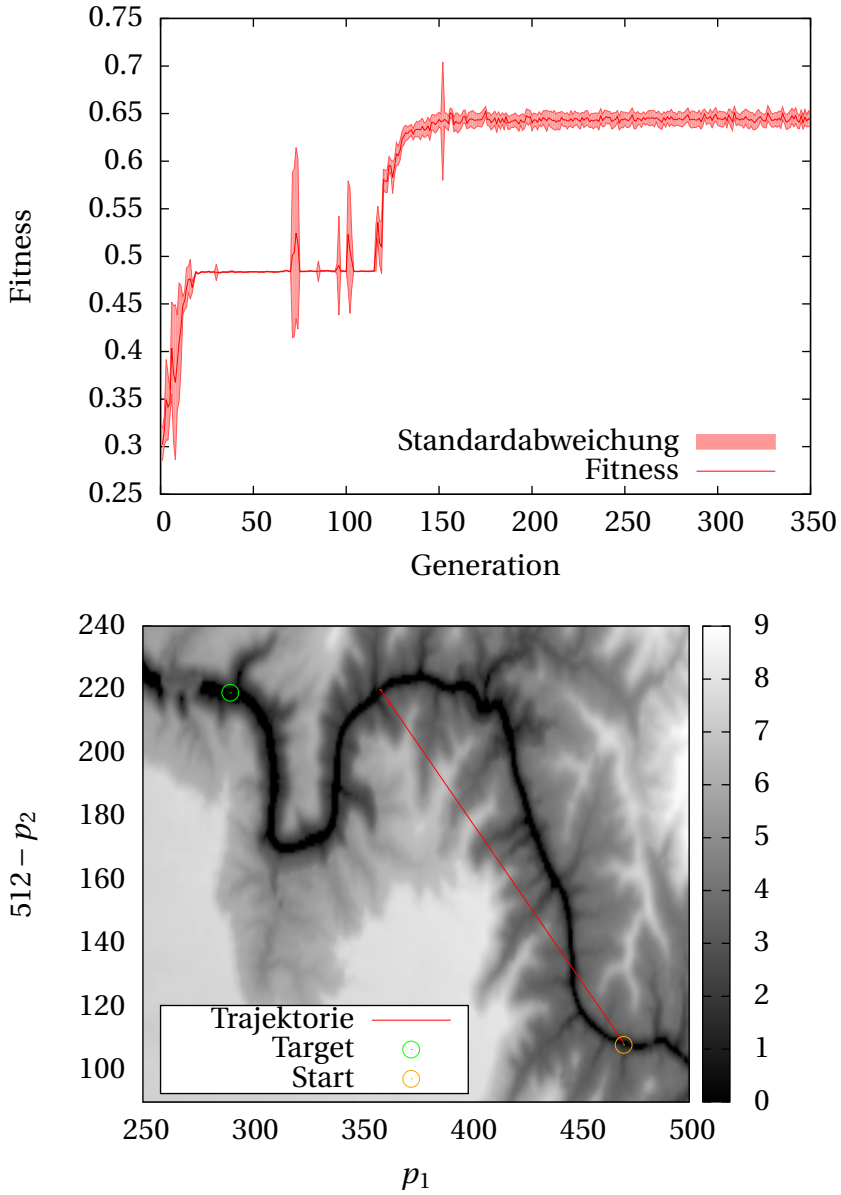


Abbildung 14: Fitness und Trajektorie der besten aus 50 Durchführungen des genetischen Algorithmus mit Task 2, g_{max}=1000 und mit Energiekosten. Oben ist die Fitness des besten Genoms in Abhängigkeit der Generation dargestellt. Die Standardabweichung wird dabei aus den 50 Simulationen der Task gewonnen. Die Fitness liegt nach 200 Generationen bereits nah am Endwert von 0,642 (12). Unten ist die Trajektorie beispielhaft nach 500 Generationen dargestellt. Diese ist jedoch nicht stark von den Startbedingungen abhängig und verschiebt sich nur um 1 bis zwei Einheiten. Die Form der Trajektorie ist bei Generation 50 jedoch fast identisch mit der bei Generation 500. Der Sprung in der Fitness bei Generation 110 kommt dadurch, dass der Agent danach auf einem Feld geringerer Energie zum stehen kommt.

4.2.2 Mit Vererbung der Topologie

4.2.2.1 Ohne versteckte Neuronen

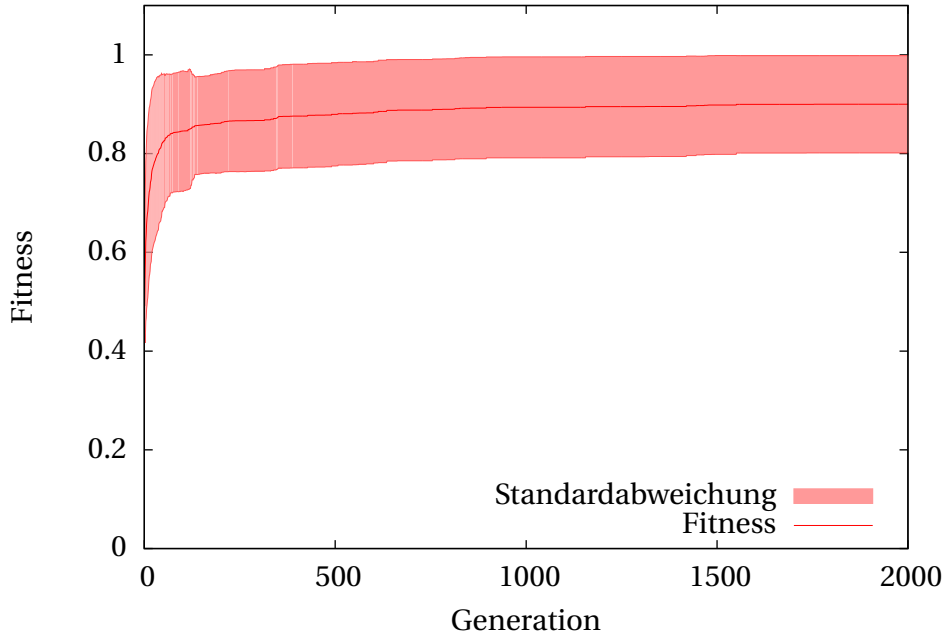


Abbildung 15: Ausschnitt der mittleren Fitness des besten Genoms von 50 Durchführungen des genetischen Algorithmus mit Task 2 und Energiekosten bei gleichzeitiger Entwicklung von Topologie und Lernregel. Die Werte verändern sich nach den dargestellten 2000 Generationen kaum und erreichen nach 2000 Generationen fast die End-Fitness von $0,913(94)$.

Nun wird die Simulation aus dem vorherigen Abschnitt wiederholt, diesmal jedoch mit gemeinsamer Entwicklung von Topologie und Lernregel. Diese erreichen nach 10000 Generationen eine mittlere End-Fitness von $0,913(94)$.

Die Durchführung mit dem besten Genom erreicht eine Fitness von 0,947. Der Fitnessverlauf der Simulation ist in Abbildung 17 zusammen mit der zugehörigen Trajektorie dargestellt.

Als Topologie des besten Genoms entwickelt sich:

d_{x_t}/D	d_{y_t}/D	$\Delta E(\vec{e}_y)$	$\Delta E(\vec{e}_x)$	$\Delta E(-\vec{e}_y)$	$\Delta E(-\vec{e}_x)$	Δx_t	Δy_t	
1	1	-1	-1	-1	1	0	-1	Δx_{t+1}
1	0	-1	0	1	0	0	1	Δy_{t+1}

mit der zugehörigen Lernregel:

$$\Delta\omega_{ij} = ((r_i/((((1-\omega_{ij})/8) - ((\omega_{ij}-2)\cdot(8\cdot 10)))/4))/(10/r_j))$$

$$= \frac{16 r_i r_j}{6405 - 3205\omega_{ij}}.$$

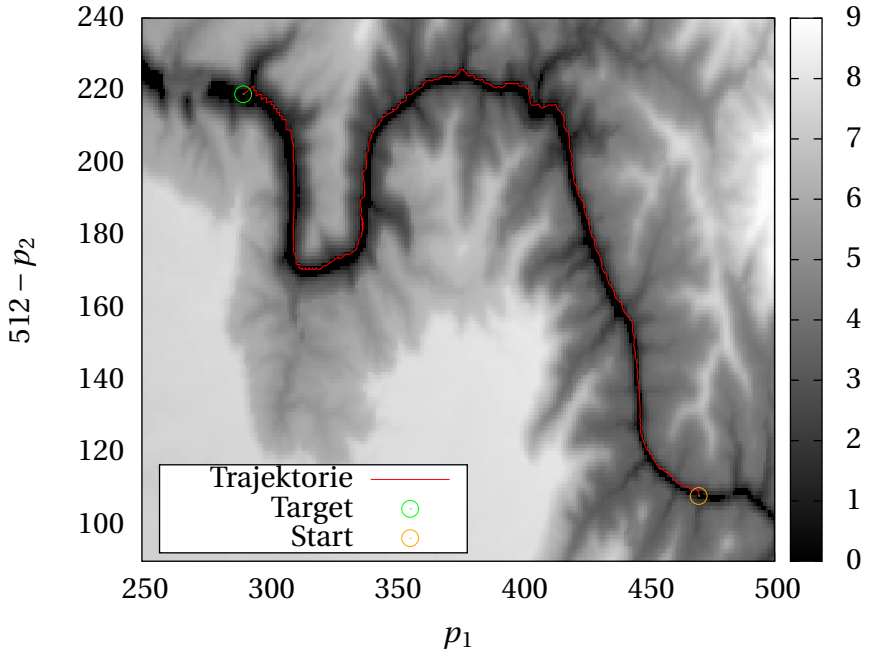


Abbildung 16: Trajektorie eines der Agenden, die bei Task 2 mit Entwicklung der Topologie eine Lernregel Äquivalent zu $\Delta\omega_{ij} = 0$ entwickelten und eine Fitness von 0,937 erreichten.

Diese Lernregel hat eine Polstelle bei $\omega_{ij} = 6405/3205 \approx 2$. Diese spielt aber in der Simulation keine Rolle, da die Gewichte während der Simulation nur Werte im Intervall $[(-1,063),(1,66)]$ annehmen. Die Lernregel ist für Netzwerke der entwickelten Topologie notwendig, um eine hohe Fitness zu erhalten: ein Netzwerk der gleichen inertialen Topologie würde statisch (also mit der Lernregel $\Delta\omega_{ij} = 0$) eine Fitness von 0,796 erhalten.

Es zeigt sich jedoch auch, dass der Task auch ohne synaptische Plastizität gut lösbar ist: 11 der 50 Läufe haben mit einer äquivalenten Lernregel zu $\Delta\omega_{ij} = 0$ eine Fitness von 0,937 erreicht. Dabei entwickelten diese die Topologie:

d_{x_t}/D	d_{y_t}/D	$\Delta E(\vec{e}_y)$	$\Delta E(\vec{e}_x)$	$\Delta E(-\vec{e}_y)$	$\Delta E(-\vec{e}_x)$	Δx_t	Δy_t	
1	0	-1	-1	-1	0	1	-1	Δx_{t+1}
1	0	-1	-1	1	0	0	1	Δy_{t+1} .

Die zugehörige Trajektorie ist in Abbildung 16 dargestellt.

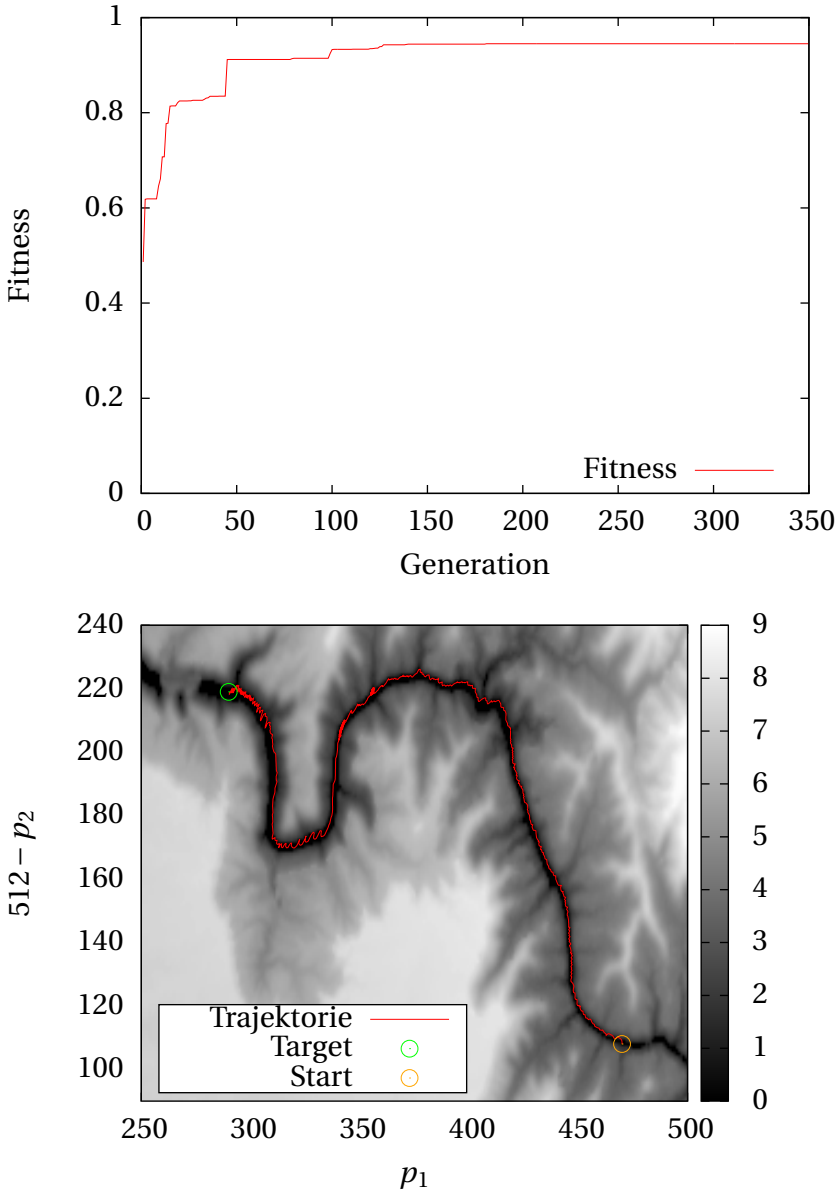


Abbildung 17: Fitness und Trajektorie der besten aus 50 Durchführungen des genetischen Algorithmus mit Task 2. Oben ist die Fitness in Abhängigkeit der Generation zu sehen. Diese erreicht nach etwa 200 Generationen fast die End-Fitness von 0,947. Unten ist die Trajektorie des Agenten dargestellt.

4.2.2.2 Mit versteckten Neuronen

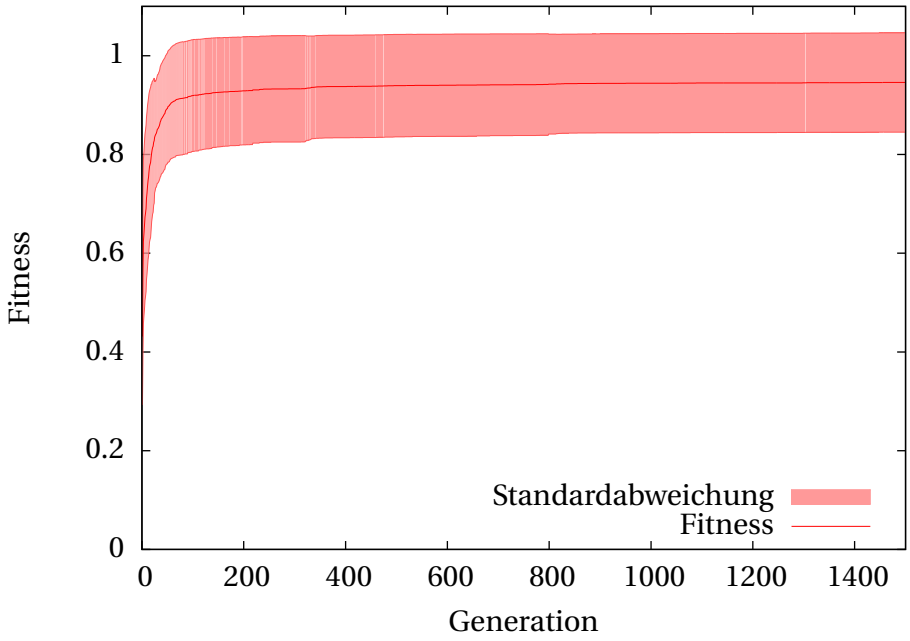


Abbildung 18: Ausschnitt der mittleren Fitness des besten Genoms von 50 Durchführungen des genetischen Algorithmus mit Task 2, V=3, g_{max}=1000 und mit Energiekosten. Die Fitness erreicht nach etwa 1000 Generationen fast End-Wert von 0,95 (10).

Die Simulation aus dem vorherigen Abschnitt wird nun mit drei versteckten Neuronen wiederholt ($V = 3$). Der mittlere Fitnessverlauf der 50 Durchführungen ist in Abbildung 18 dargestellt. Nach etwa 1000 Generationen wird bereits ein Wert sehr nah an der End-Fitness von 0,95 (10) erreicht.

Die Durchführung mit dem besten Genom erreicht eine Fitness von 0,969. Der Fitness-verlauf der Simulation ist in Abbildung 19 zusammen mit der zugehörigen Trajektorie dargestellt.

Das beste Genom entwickelt die Topologie:

d_{x_t}/D	d_{y_t}/D	$\Delta E(\vec{e}_y)$	$\Delta E(\vec{e}_x)$	$\Delta E(-\vec{e}_y)$	$\Delta E(-\vec{e}_x)$	h_1	h_2	h_3	Δx_t	Δy_t	
-1	1	-1	0	1	0	0	-1	-1	-1	1	h_1
0	0	0	0	0	-1	0	0	-1	-1	0	h_2
0	0	0	0	0	0	-1	-1	1	-1	0	h_3
1	1	-1	0	-1	1	0	-1	0	0	1	Δx_{t+1}
1	-1	-1	0	1	0	1	1	-1	0	0	Δy_{t+1}

mit der zugehörigen Lernregel:

$$\Delta\omega_{ij} = (r_i - r_i) = 0$$

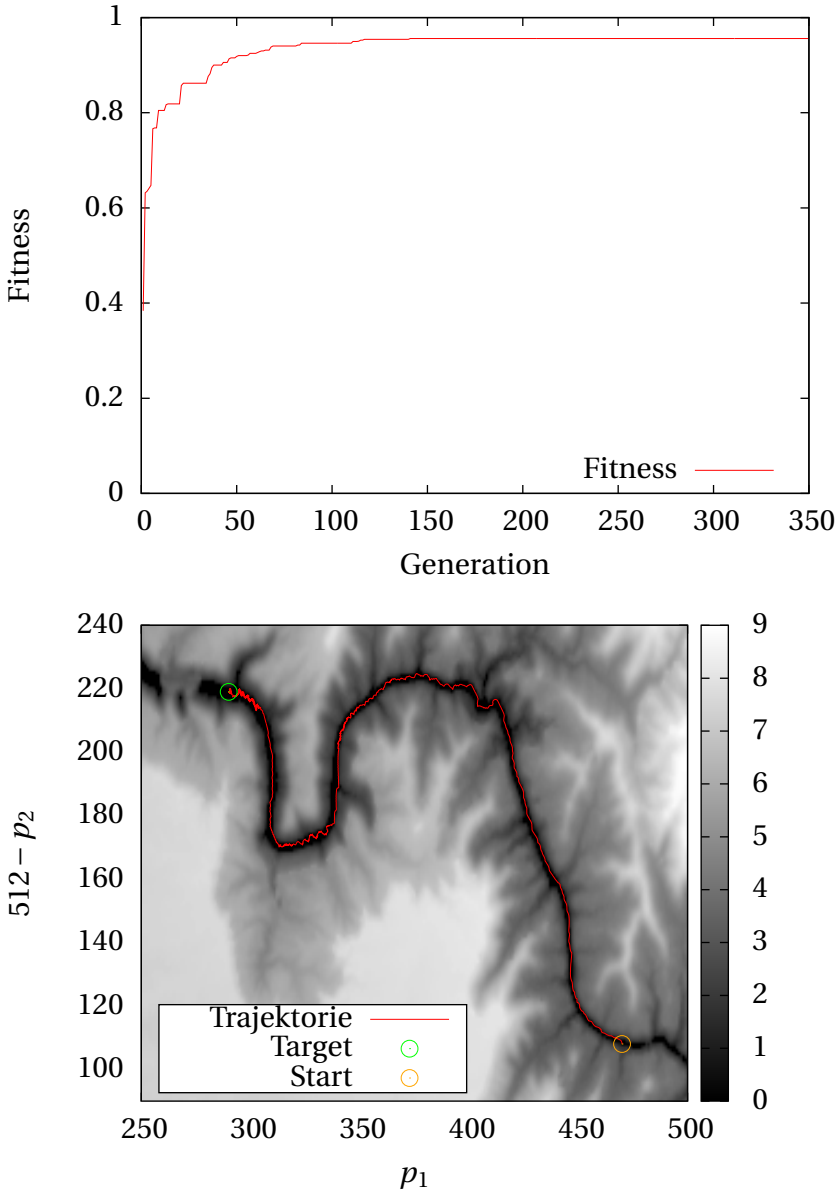


Abbildung 19: Fitness und Trajektorie des besten Genoms aus 50 Durchführungen des genetischen Algorithmus mit Task 2 und V=3. Oben ist die Fitness in Abhängigkeit der Generation zu sehen. Diese erreicht nach bereits 200 Generationen fast die endgültige Fitness von 0,969. Unten ist die Trajektorie des Agenten dargestellt.

5 Diskussion

In Kapitel 4.1.1 entwickelten sich bei Task 1 ohne Energiekosten recht erfolgreiche Lernregeln. Die entwickelten Lernregeln sind jedoch aufgrund ihrer gewaltigen Größe leider schwer analysierbar. Weitere Untersuchungen zur Klassifizierung könnten über Stabilitätsuntersuchungen, Reihenentwickelung oder PCA Analysen gemacht werden, sprengen jedoch den Rahmen dieser Bachelorarbeit. Vermutlich sind die Lernregeln stark auf die Aufgabe spezialisiert, da diese die erwünschte Topologie genau vorgibt. Diese erfordert beispielsweise keine negativen Gewichte.

Die in den folgenden Kapiteln benutzten Energiekosten erwiesen sich als sehr effektiv, die Lernregeln in Kapitel 4.1.2 wurden nicht nur verhältnismäßig kurz, sondern lassen sich auch übersichtlich darstellen. Interessant ist auch, dass die gefundene Lernregel ohne Energiekosten besser abschneidet als die beste Lernregel des vorherigen Kapitels. Die entwickelte Lernregel lässt sich keiner mir bekannten Kategorie zuordnen. Sie ist aber vermutlich, wie im vorherigen Abschnitt erklärt, stark auf die Aufgabe spezialisiert.

Das gleiche Problem trat wohl auch beim zweiten Task im Kapitel 4.2.1 auf, bei der die Lernregeln trotz Energiekosten sehr lang wurden. Diesmal zeigten Lernregeln jedoch nicht einmal erwünschte Verhalten, nämlich effizient zu navigieren, sondern sie bewegten sich auf einer diagonalen Geraden, um in einem Bereich geringer Energie zum Stehen zu kommen. Eine besser angepasste Fitnessfunktion könnte möglicherweise bessere Ergebnisse erzielen, beispielsweise indem durch Checkpoints (Zwischenziele) die Trajektorie stärker vorgegeben wird. Jedoch ist es kaum vorstellbar, dass die Aufgabe allein durch eine einfache Lernregel lösbar ist.

In Kapitel 4.2.2.1 wurden Topologie und Lernregel parallel entwickelt. Hierbei zeigte sich, dass eine sehr gute Lösung des Problems auch ohne synaptische Plastizität gefunden werden kann. Diese Lösung fanden gleich 22 % der durchgeführten Simulationen. Eine noch bessere Fitness wurde allerdings mit einer anderen Topologie und einer Hebb-ähnlichen Lernregel erreicht. Beide Lösungen verfolgen jedoch offenbar die gleiche wall-follower Strategie, orientieren sich also an einer Seite der Energielandschaft, bis sie das Ziel erreichen.

Durch das Einbringen von versteckten Neuronen in Kapitel 4.2.2.2 nutzte die beste gefundene Lösung keine synaptische Plastizität, erreichte dennoch eine höhere Fitness als die beste Lösung des vorherigen Kapitels. Hierbei wurde wahrscheinlich die größere Topologie genutzt, um Informationen über eine bessere Trajektorie zu kodieren.

Fast alle der durchgeführten Simulationen konvergierten nach einem Bruchteil der vorgegebenen Generationen zu einem stabilen lokalen Minimum. Dies sollte in Zukünftigen Implementationen des Algorithmus beachtet werden. Eine Abbruchbedingung könnte beispielsweise anhand der Stagnation der Fitness viel früher ausgelöst werden. Eine parallele Entwicklung mehrerer Populationen mit gelegentlicher Migration von Genomen ist ein möglicher Ansatz, mit dem eine höhere Fitness erreicht werden könnte.

Beide in dieser Arbeit vorgestellten Tasks sind ohne synaptische Plastizität perfekt lösbar. Ob mit der vorgestellten Methode Aufgaben lösbar sind, die Lernen und/oder Gedächtnis erfordern, ist ungeklärt, könnte jedoch Gegenstand späterer Arbeiten werden.

Die in dieser Arbeit simulierte Evolution der Lernregeln halte ich nicht für biologisch relevant. Zum einen da die Tasks 1 und 2 recht willkürlich gewählt wurden und kein mir bekanntes biologisches Analog besitzen, zum anderen da mir das Modell einer rein zufälligen anfänglichen Topologie unrealistisch vorkommt. Das gemeinsame Entwickeln von Lernregel und Topologie scheint jedoch vor allem bei hoher Anzahl an Neuronen dazu zu führen, dass sich der genetische Algorithmus allein auf die Entwicklung der Topologie konzentriert. Hierbei hätte eine deutlich schnellere Evolution erreicht werden können, wenn keine Lernregel, sondern alleine die Topologie entwickelt worden wäre. Das alleinige Entwickeln einer Lernregel zur Lösung einer Aufgabe erscheint mir aus praktischen und biologischen Gründen nicht sinnvoll. Es unpraktisch, da, um die Abhängigkeit von den Startbedingungen zu eliminieren, die Fitness über möglichst viele zufällige Startnetzwerke gemittelt werden muss, was daher auch deutlich mehr Rechenzeit erfordert. Andererseits entstehen Biologische Netzwerke nicht rein zufällig, sondern befolgen bei der Morphogenese spezielle Regeln, die sich beispielsweise über L-Systeme [13] modellieren lassen. In [9] wurde gezeigt, dass genetische Algorithmen, die Topologien großer neuronaler Netzwerke entwickeln, schneller zu guten Ergebnissen konvergieren, wenn diese statt direkter Kodierung des Netzwerks eine Vorschrift für ein erzeugendes L-System entwickeln. Die L-Systeme bieten zudem Informationen über mögliche Prozesse der Morphogenese von Netzwerken. Es wäre sicherlich sehr interessant, eine kombinierte Entwicklung von Lernregeln und L-System Regeln für Netzwerke zu betrachten. Ich könnte mir vorstellen, dass dabei, bei geeigneter Aufgabe, biologisch relevante Aussagen zur Evolution solcher Systeme gewonnen werden könnten.

Die vorgestellte Methode zur Entwicklung von Lernregeln hat größtenteils sehr gute Ergebnisse geliefert. Die kontextfreie Grammatik kann dabei beliebig erweitert werden, um Funktionen komplexerer Sprachen zu entwickeln. Werden dabei weitere nicht-Terminalsymbole eingeführt, sollten die Operationen Mutation und Reproduktion geringfügig angepasst werden, damit nur Wörter, die dem Gleichen Symbol Zugeordnet sind, vertauscht werden. Die Methode ist nicht nur auf die Entwicklung von Lernregeln neuronaler Netzwerke beschränkt. Eine vorstellbare Praktische Anwendung wäre beispielsweise, durch eine Grammatik, die eine Turing-vollständige Sprache beschreibt, ganze Programme zur autonomen Steuerung von Robotern zu entwickeln.

Glossar

Energiekosten Energiekosten für entwickelte Lernregeln 16

Funktionales Genom Das zu einer Lernregel gehörende Genom 15

$g_{\mathbf{max}}$ Anzahl an Iterationen bzw. Generationen des genetischen Algorithmus 11

$I_j(t)$ Durch das Neuron j fließender Strom 5

$\vec{I}$ Vektorieller Eingangsstrom $\left(\vec{I}\right)_j = I_j$ 6

$K(t)$ Synaptische Kern 5

$\mathbb{L}$ Wahrscheinlichkeitsraum der Funktionale 15

N Anzahl der Individuen einer Population 11

Ω Gewichtsmatrix $(\Omega)_{ij} = \omega_{ij}$, falls eine Synapse von Neuron i nach j existiert 6

$\omega_{ij}(t)$ Gewicht einer Synapse von Neuron i zu j 5

p_E Anteil Elitärer Genome in einer Population 11

$p_{\mathbb{F}}$ Wahrscheinlichkeit bei einem zufälligen Funktional eine Funktion zu finden 15

$\varphi(I_j)$ Neuronale Aktivierungsfunktion 6

$\vec{\varphi}(\vec{I})$ Vektorielle Aktivierungsfunktion $\left(\vec{\varphi}(\vec{I})\right)_j = \varphi(I_j)$ 6

p_M Mutationsrate 11

p_R Rekombinationsrate 11

$p_{\mathbb{V}}$ Wahrscheinlichkeit bei einem zufälligen Funktional eine Variable zu finden 15

$p_{\mathbb{Z}}$ Wahrscheinlichkeit bei einem zufälligen Funktional eine Zahl zu finden 15

$\rho(t)$ Neuronale Antwortfunktion 4

$r_i(t)$ Feuerrate oder Aktivität des Neurons i 5

$\vec{r}$ Vektorielle Feuerrate $(\vec{r})_j = r_j$ 6

Topologisches Genom Das zu einer Topologie gehörende Genom 14

τ_s Zeitkonstante des synatischen Kerns 5

$T_{\mathbf{Task\ 1}}$ Zeitschritte einer Simulation von Task 1 13

$T_{\mathbf{Task\ 2}}$ Zeitschritte einer Simulation von Task 2 13

V Anzahl der versteckten Neuronen 12

Literatur

[1] ALBERTS, Bruce ; JOHNSON, Alexander ; LEWIS, Julian ; RAFF, Martin ; ROBERTS, Keith ; WALTER, Peter: *Molecular Biology of the Cell*. 5. Garland Science, 2008

[2] BI, G. ; POO, M.: Synaptic modifications in cultured hippocampal neurons: dependence on spike timing, synaptic strength, and postsynaptic cell type. In: *Journal of Neuroscience* (1998)

[3] BOLZE, Kirsten ; WERNER, Frank: *Grundlagen der Stochastik*. `http://num.math.uni-goettingen.de/f.werner/files/Stochastik_Skript.pdf`. Version: 2007/2008

[4] DAYAN, P. ; ABBOTT, L.F.: *Theoretical Neuroscience: Computational And Mathematical Modeling of Neural Systems*. Mit Press, 2005 (Computational Neuroscience Series). – ISBN 9780262541855

[5] FOGEL, David B.: *Evolutionary computation: toward a new philosophy of machine intelligence*. Piscataway, NJ, USA : IEEE Press, 1995. – ISBN 0–7803–1038–1

[6] GERSTNER, W. ; KISTLER, W.M.: *Spiking Neuron Models: Single Neurons, Populations, Plasticity*. Cambridge University Press, 2002. – ISBN 9780521890793

[7] GUENNEBAUD, Gaël ; JACOB, Benoît u. a.: *Eigen v3*. http://eigen.tuxfamily.org, 2010

[8] HOWARD ZHOU, Jie S.: *Quelle des Höhenprofils vom Grand Canyon*. `http://www.howardzzh.com/research/poissonImageEditing/index.htm`, Abruf: 5.04.2013

[9] KITANO, Hiroaki: Designing Neural Networks Using Genetic Algorithms with Graph Generation System. In: *Complex Systems Journal* 4 (1990), S. 461–476

[10] KOBALICEK.PETR: *Asmjit 1.0b4*. http://code.google.com/p/asmjit/, 2012

[11] KOZA, John R. ; KEANE, Martin A. ; STREETER, Matthew J. ; MYDLOWEC, William ; YU, Jessen ; LANZA, Guido: *Genetic Programming IV : Routine Human-Competitive Machine Intelligence (Genetic Programming)*. Springer. – ISBN 1402074468

[12] MITCHELL, Melanie: *An Introduction to Genetic Algorithms*. Cambridge, MA, USA : MIT Press, 1998. – ISBN 0262631857

[13] ROZENBERG, Grzegorz ; SALOMAA, Arto: *Lindenmayer Systems: Impacts on Theoretical Computer Science, Computer Graphics, and Developmental Biology*. Springer, 1992. – ISBN ISBN 978–3–540–55320–5

[14] SHAPIRO, Jonathan: Genetic Algorithms in Machine Learning. In: *Machine Learning and its Applications, Springer, ISBN 3-540-42490-3* (2001), S. 146–168

[15] THOMPSON, R.F.: *Foundations of physiological psychology*. Harper & Row, 1967 (Harper's physiological psychology series)

[16] WIKIPEDIA: *Artikel zu Aktionspotentialen*. http://de.wikipedia.org/wiki/Aktionspotential, Abruf: 25.03.2013

[17] WIKIPEDIA: *Artikel zu künstlichen Neuronen*. http://de.wikipedia.org/wiki/Künstliches_Neuron, Abruf: 25.03.2013

[18] WIKIPEDIA: *Artikel zu Neuronen*. http://en.wikipedia.org/wiki/Neuron, Abruf: 25.03.2013

[19] WIKIPEDIA: *Artikel zum Satz von der monotonen Konvergenz*. http://de.wikipedia.org/wiki/Satz_von_der_monotonen_Konvergenz,

[20] ZUBER, Benoît ; NIKONENKO, Irina ; KLAUSER, Paul ; MULLER, Dominique ; DUBOCHET, Jacques: The mammalian central nervous synaptic cleft contains a high density of periodically organized complexes. In: *PNAS* (2005)

Anhang

Anhang 1: Mathematische Grundlagen

In dieser Arbeit werden grundlegende Kenntnisse in Differenzial- und Integralrechnung vorausgesetzt. Werden weiterführende Kenntnisse benötigt, werden sie in diesem Kapitel eingeführt oder auf die Literatur verwiesen.

Definition 1: Erwartungswert

Der Erwartungswert einer Zufallsvariable X mit Wahrscheinlichkeitsdichtefunktion $f : \mathbb{R} \to [0,\infty)$ wird definiert als:

$$E(X) = \int_{\mathbb{R}} x f(x) \mathrm{d}x.$$

Definition 2: Frobeniusnorm

Die Frobeniusnorm $\|A\|_F$ einer beliebigen Matrix $A \in \mathbb{R}^{m \times n}$ wird definiert als:

$$\|A\|_F = \sqrt{\sum_{i=1}^{m} \sum_{j=1}^{n} |(A)_{ij}|^2}.$$

Definition 3: Testfunktionen

Wir bezeichnen hier als die Menge der reellen Testfunktionen $\mathcal{T}$ die Menge aller unendlich oft differenzierbaren Funktionen $f : \mathbb{R} \to \mathbb{R}$, die einen kompakten Träger haben, die also außerhalb einer kompakten Teilmenge von $\mathbb{R}$ den Wert 0 annehmen.

Definition 4: Dirac'sche Delta Distribution

Die dirac'sche Delta Delta Distribution (oder Delta Distribution) wird für jede Zahl $x \in \mathbb{R}$ definiert als:

$$\delta(x) = \begin{cases} \infty, & \text{falls } x = 0 \\ 0, & \text{sonst.} \end{cases}$$

Zudem erfüllt sie für jede Testfunktion $f \in \mathcal{T}$ die Identität:

$$\int_{-\infty}^{\infty} f(x)\delta(x)\mathrm{d}x = f(0).$$

Definition 5: Heaviside Funktion

Die Heaviside Funktion wird für jede Zahl $x \in \mathbb{R}$ definiert als:

$$\theta(x) = \begin{cases} 1, & \text{falls } x \geq 0 \\ 0, & \text{sonst.} \end{cases}$$

Die Ableitung der Heaviside Funktion ist die Delta Distribution:

$$\frac{\mathrm{d}\theta(x)}{\mathrm{d}x} = \delta(x).$$

Dies wird schnell klar anhand der Wirkung von θ auf die Ableitung einer Testfunktion $f \in \mathcal{T}$:

$$\int_{-\infty}^{\infty} \theta(x) f'(x) \mathrm{d}x = \int_{0}^{\infty} f'(x) \mathrm{d}x = [f(x)]_0^\infty = -f(0).$$

Über partielle Integration erhalten wir jedoch:

$$\int_{-\infty}^{\infty} \theta(x) f'(x) \mathrm{d}x = [\theta(x) f(x)]_{-\infty}^{\infty} - \int_{-\infty}^{\infty} \theta'(x) f(x) \mathrm{d}x = -\int_{-\infty}^{\infty} \theta'(x) f(x) \mathrm{d}x.$$

Damit beide Gleichungen erfüllt sind, muss $f(0) = \int_{-\infty}^{\infty} \theta'(x) f(x) \mathrm{d}x$ gelten. Mit Definition 4 erhalten wir schließlich:

$$\theta'(x) = \delta(x).$$

Anhang 2: Stochastische kontexfreie Grammatiken

Eine kontexfreie Grammatik ist eine formale Grammatik, die durch ein 4-Tupel $G = (V, \Sigma, P, S)$ definiert wird. Dabei ist

- V: Eine endliche Menge nicht-terminaler Symbole.
- Σ: Eine zu V disjunkte endliche Menge terminaler Symbole. Eine Folge von Symbolen aus $V \cup \Sigma$ bezeichnen wir als Wort.
- P: Eine Relation von V zu $(V \cup \Sigma)^*$. $(V \cup \Sigma)^*$ wird als die Kleenesche Hülle von $(V \cup \Sigma)$ bezeichnet und ist die Menge aller Wörter, die durch Verknüpfung von Symbolen oder Wörtern aus $(V \cup \Sigma)$ und dem leeren Wort ϵ (ein Wort, welches aus keinem Zeichen besteht) gebildet werden können. Die Elemente von P bezeichnen wir als Produktionsregeln.
- S: Das Startsymbol: $S \in V$.

Eine Produktionsregel $p \in P$ ist ein Tupel (α, β) mit $\alpha \in V$ und $\beta \in (V \cup \Sigma)^*$. p lässt sich alternativ schreiben als $\alpha \to \beta$.

Eine Stochastische kontexfreie Grammatik G' ist erweitert die Produktionsregeln der kontexfreien Grammatik G, indem jeder Produktionsregel $(\alpha, \beta) \in P$ eine Wahrscheinlichkeit $\rho(\alpha, \beta)$ zugeordnet wird. Dabei ergibt die Summe über alle Wahrscheinlichkeiten einer Produktionsregel 1:

$$\sum_{\beta} \rho(\alpha, \beta) = 1$$

G' kann benutzt werden, um zufällige Wörter der Sprache, die von G erzeugt wird, zu erzeugen. Dabei werden, beginnend mit dem Startsymbol S, alle Nicht-Terminalsymbole durch eine Produktionsregel $(\alpha, \beta) \in P$ mit Wahrscheinlichkeit $\rho(\alpha, \beta)$ durch β ersetzt, bis das resultierende Wort nur noch aus Nicht-Terminalsymbolen oder ϵ besteht.

Anhang 3: Erwartungswert der Länge einer Lernregel

An dieser Stelle möchte ich dem Göttinger Mathematikstudent BSc. JAN-MIRKO OTTER danken, der mir in kürzester Zeit den folgenden Beweis für die Längen der Lernregeln geliefert hat.

Voraussetzung:
Es sei X ein nicht-leerer binärer Baum mit Verzweigungswahrscheinlichkeit p.

Behauptung:
Der Erwartungswert der Knotenzahl N des Baumes ist:

$$E(N) = \begin{cases} \frac{1}{1-2p}, & \text{für } p < 1/2 \\ \infty & \text{sonst.} \end{cases}$$

Beweis:
Sei n_h die Anzahl der Knoten mit Höhe h. Dann gilt: $N = \sum_{h=0}^{\infty} n_h$. Da die Partialsummen n_h nichtnegativ sind, lässt sich mit dem Satz von der monotonen Konvergenz [19] der Erwartungswert schreiben als:

$$E(N) = \sum_{h=0}^{\infty} E(n_h).$$

Mit der Wahrscheinlichkeit p verzweigt nun jeder Knoten in einer Höhe h zu zwei Knoten in der Höhe $h+1$. Ist also die Anzahl der Knoten der Ebene h bekannt, ist die Anzahl der Verzweigungen in der Ebene eine Bernoulli-verteilte Zufallsvariable mit Parametern p und n_h:

$$E(n_{h+1}|n_h) = 2pn_h.$$

Wenden wir den Satz der iterierten Erwartung ($E(E(X|Y)) = E(X)$, [3]: S. 82 ff.) an, erhalten wir die Rekursionsformel:

$$E(n_{h+1}) = E(E(n_{h+1}|n_h)) = E(2pn_h) = 2p \cdot E(n_h).$$

Mit der Bedingung, dass der Baum mindestens einen Knoten besitzt ($n_0 = 1$), lässt sich die Rekursionsformel lösen:

$$E(n_h) = (2p)^h.$$

Damit erhalten wir für $E(N)$ die Geometrische Reihe:

$$E(N) = \sum_{h=0}^{\infty} (2p)^h = \begin{cases} \frac{1}{1-2p}, & \text{für } p < 1/2 \\ \infty & \text{sonst.} \end{cases}$$

□

Anhang 4: Lange Lernregeln

Beste Lernregel für Task 1 ohne Energiekosten mit nach 90 Generationen:
$\Delta\omega_{ij} = (((((((4-(\omega_{ij}/3))-1)-\omega_{ij})+(((\omega_{ij}+((4-1)-2))/(6-((2-(((3+((1/(((10/((((((3-1)-\omega_{ij})+(((\omega_{ij}+((4-r_j)-2))/(r_j-((2-((((6+((1/(((10/((((((3-1)-\omega_{ij})+(((\omega_{ij}+((2-\omega_{ij})+1))/(r_j-((2-((\omega_{ij}\cdot 6)-10))+\omega_{ij})))-\omega_{ij}))-r_j)/8)-\omega_{ij}))-(((3+((r_i/(((10/(2/(r_i-(1-4))))-r_i)\cdot(\omega_{ij}\cdot 5)))\cdot\omega_{ij}))/((((3-1)\cdot(\omega_{ij}\cdot 5))-\omega_{ij})+(((6/2)/((10/((((((3-\omega_{ij})-\omega_{ij})+(((\omega_{ij}+((4-1)-2))/(r_i+(3-1)))-\omega_{ij}))-r_j)/8)-\omega_{ij}))-4))-3)))-10))\cdot(\omega_{ij}\cdot 5)))\cdot\omega_{ij}))/\omega_{ij})-10)-(((\omega_{ij}+((4-1)-2))/(4-((2-1)+\omega_{ij})))-\omega_{ij})))+((\omega_{ij}/3)\cdot((2-(((3+((1/(((\omega_{ij}/((((((10/((((((3-1)-\omega_{ij})+(((\omega_{ij}+((10/(10-r_j))-2))/(r_j-(((1/((8-4)\cdot 1))-((\omega_{ij}\cdot 6)-10))+\omega_{ij})))-\omega_{ij}))-r_j)/8)-\omega_{ij}))+(((\omega_{ij}+((8-1)-2))/(4-((2-r_j)+\omega_{ij})))-\omega_{ij}))-r_j)/8)-\omega_{ij}))-4)\cdot 1))\cdot\omega_{ij}))/\omega_{ij})-10))+1)))))-\omega_{ij}))-r_j)/8)-\omega_{ij}))-4)\cdot(\omega_{ij}\cdot 5)))\cdot((\omega_{ij}\cdot 6)-r_i)))/\omega_{ij})-10))+\omega_{ij})))-\omega_{ij}))-r_j)/8)-\omega_{ij})$.

Beste Lernregel für Task 1 ohne Energiekosten nach 1000 Generationen:
$\Delta\omega_{ij} = (((((((4-\omega_{ij})-1)-\omega_{ij})+(((\omega_{ij}+(3-2))/(6-((2-(((3+((1/(((4/((((((3-1)-\omega_{ij})+(((\omega_{ij}+2)/(r_j-((2-((((6+((1/((6/2)/(((((\omega_{ij}-1)-\omega_{ij})+(r_i/4))-r_j)/8)))\cdot 2))/\omega_{ij})-10)-(1-\omega_{ij})))+((\omega_{ij}/3)\cdot((2-(((3+((3/((6+((((4-9)/((((((9-r_j)+(\omega_{ij}-\omega_{ij}))\cdot(3-1))+\omega_{ij})-((((3/(3-1))+3)-((6/((((\omega_{ij}+((1/((((\omega_{ij}-1)-\omega_{ij})+(\omega_{ij}+(1\cdot 2)))/(((2/1)-4)/8)))\cdot 2))/\omega_{ij})-9)-9))/(((r_j-r_j)-(\omega_{ij}+(1\cdot 2)))/(\omega_{ij}/(3+8)))))-9))+(((((((((4-5)+r_i)-((((((6+((6/2)/(((\omega_{ij}+(r_i/4))-r_j)/(r_j-\omega_{ij}))))\cdot(r_i-8))+\omega_{ij})-((((3/\omega_{ij})+3)-((6/(1\cdot 2))/(\omega_{ij}/((6\cdot r_i)-1))))-3))/(((1/(4-r_j))-2)/((1-6)-4)))-(((6+(r_j\cdot(3-1)))/((((6/(10\cdot 5))-(((((((7-\omega_{ij})\cdot((((6+((10/(r_i+7))\cdot r_i))\cdot(7-1))/((((6-(4-\omega_{ij}))\cdot(r_i-6))+r_i)-10))-2))/((9+3)-(r_i/(\omega_{ij}-7))))-(r_i/r_i))/(10-\omega_{ij}))+8)\cdot r_j))-(1/1))-((2+r_j)-\omega_{ij})))-8)))/4)-(\omega_{ij}+(3-2)))\cdot((((((\omega_{ij}-1)-\omega_{ij})+\omega_{ij})-((10-r_j)-9))/((r_j\cdot 6)-2))-2))/(((6-r_j)/(r_j\cdot r_i))/10))-(3-10))\cdot((1/((((r_j-4)/r_j)-(2-10))\cdot r_j))-(((7/(r_j-\omega_{ij}))-1)-r_j)))))/((7-((6+((((6/2)-\omega_{ij})/(((((((r_j+7)-((((5/2)+3)-((6/((((6+((1/((7/((4-\omega_{ij})-5))/(r_j/8)))\cdot 2))/\omega_{ij})-9)-9))/(\omega_{ij}/((6\cdot 4)-r_j))))-9))/((((r_j/\omega_{ij})\cdot r_i)+\omega_{ij})/((1-r_j)-r_i)))-(6-(((6+((6/2)/(((\omega_{ij}+(r_i/4))-r_j)/(r_j-\omega_{ij})))\cdot(r_i-8))+\omega_{ij})))\cdot((((((((((6+(\omega_{ij}\cdot(3-1)))/((\omega_{ij}-(((2\cdot 3)-8)\cdot(8+\omega_{ij})))-7))-(2/(2+((7\cdot\omega_{ij})+r_j))))+r_i)-(2/(((2\cdot(1+(((r_i-7)+7)\cdot r_j)))-\omega_{ij})/r_i)))/(9-(((2\cdot r_j)-\omega_{ij})\cdot 4)))-((((4-1)-\omega_{ij})+((3/2)-\omega_{ij}))-\omega_{ij}))\cdot(((((6-((2/(r_i/(\omega_{ij}/4)))\cdot r_j))+\omega_{ij})-((7-r_j)-9))/((6\cdot 7)-2))-(((r_i-(\omega_{ij}+(1\cdot 2)))/(((((((((1-\omega_{ij})+r_j)+((\omega_{ij}/(((4-6)/2)/4))\cdot 2))/\omega_{ij})/(2/r_j))-((2/(r_i/1))\cdot r_j))-2)-7))-((((9\cdot\omega_{ij})/1)+3)-(r_i-3)))))/((7/(5\cdot r_i))/9))/3))/(4/10))-(2-10))\cdot(((((6-8)-(2-r_j))\cdot 7)/(((7-((\omega_{ij}\cdot\omega_{ij})\cdot r_i))+3)-(r_i-(((\omega_{ij}\cdot(((r_i/4)-(1-\omega_{ij}))/r_j))-r_i))))-4)))\cdot 8))-r_j))\cdot(((r_i/4)-(r_j-\omega_{ij}))/(((((3-10)-((r_i/6)\cdot r_i))+1)-r_j)/8))))\cdot r_j))\cdot 1))\cdot r_i))/\omega_{ij})-10))+3)))))-(4-r_j)))-r_j)/8)-\omega_{ij}))-4)\cdot(\omega_{ij}+((4-1)-2))))\cdot((((4-1)-\omega_{ij})+(((\omega_{ij}+(((7/(8-\omega_{ij}))-1)-r_j))/2)-\omega_{ij}))-r_i)))/\omega_{ij})-10))+\omega_{ij})))-\omega_{ij}))-r_j)/8)-\omega_{ij})$.

Beste Lernregel für Task 2 mit Energiekosten nach 100 Generationen:
$\Delta\omega_{ij} = ((((((r_i/(((6-(5+r_j))/9)-(r_i/r_i)))+(6-(5+r_j)))-r_i)/(1+(4+(((1+((1-((3-(r_i+3))\cdot 7))+(((r_j/5)\cdot 6)+r_j)))/5)+r_j))))+(((((6-(5+((((((((((1+((\omega_{ij}-((3-(r_i+3))\cdot(5+9)))+(((r_j/5)\cdot 4)+r_j)))/5)-4)/3)+(2/(6\cdot((10/((5/((((r_i/(((6-(5+r_j))/9)-(r_i/r_i)))+(6-(5+r_j)))-r_i)/(1+(4+(((1+((1-((3-(r_i+3))\cdot 7))+(\omega_{ij}+r_j)))/5)+r_j)))))-8))/(r_j-r_j)))))-(6-(5+r_j)))/8)/5)+r_j)))/(5+r_j))+\omega_{ij})-\omega_{ij})/6))-\omega_{ij})$.

Beste Lernregel für Task 2 mit Energiekosten nach 1000 Generationen:

$$\Delta\omega_{ij} = ((((((((((((3-4)/3)+r_j)-(6-(5+r_j)))/8)/6)+r_j)/(((6-(5+r_j))/9)-(r_i/r_i)))+(6-(5+r_j)))-r_i)/(1+(4+(((((\omega_{ij}\cdot 4)-4)+((1-((3-(r_i+3))\cdot 7))+(((r_j/5)\cdot 6)+r_j)))/5)+r_j))))+(((((6-(5+((((((((\omega_{ij}/(r_j+1))-(5+r_j))/((r_i+3)+r_j))+((\omega_{ij}+(((((((3-4)/3)+r_j)-(6-(5+r_j)))/8)/6)+4))/(((((10-9)\cdot(5/8))/1)\cdot 1)\cdot((((((((9+(((((6+((3-((3-r_i)\cdot\omega_{ij}))+(((\omega_{ij}+(4+(5/8)))\cdot(((((4+(r_j+(4+((1\cdot(((((\omega_{ij}\cdot 4)-4)+((1-((3-(r_i+3))\cdot 7))+((6\cdot 6)+8)))/5)\cdot 2))-(5+\omega_{ij})))))/((5+r_j)-1))+(\omega_{ij}+5))-8)/(1-(r_j/(1-(5\cdot r_j))))))+(1+(4+(((\omega_{ij}/10)/5)+r_j))))))-4)+(5+((((3+\omega_{ij})/5)\cdot 6)+8)))/5)/(1\cdot(((5/(r_j+((\omega_{ij}+((((10/7)+r_j)-(6-8))+5))-7)))+5)/(((((4-6)-((r_i\cdot r_j)/((r_i-r_j)-((1-7)\cdot((r_j+6)+r_j)))))/((((((8-(1\cdot\omega_{ij}))/(1-((5/(r_j+((\omega_{ij}+((((10/7)+r_j)-(6-8)))+(5/((r_j+r_i)/(((((((6+(\omega_{ij}+r_i))-8)/5)-r_j)/((4+r_i)+(r_j-((2-(6-(((\omega_{ij}\cdot 4)-4)\cdot(6+r_j))))/(1+(\omega_{ij}+\omega_{ij})))))))+5)+r_j)))))-7)))+4)))-8)/(8+r_j))+7)-((((r_i\cdot((r_i-((10\cdot((10/r_j)\cdot(((5+10)-r_i)/(1+(r_j+2)))))\cdot 7))+r_j))/7)+r_i)-5)))/(r_j-(9+(5/8))))-(((10/(r_j+(r_i+r_i)))+3)/((r_i/6)-((r_j+r_i)/(((((((((3-4)+(\omega_{ij}+r_i))-8)/5)-(r_i-((10\cdot(10\cdot r_j))\cdot 7)))/r_j)+5)+r_j)))))))))-9)/(4+r_j))+r_j)/((\omega_{ij}/r_i)-1))+(\omega_{ij}+r_i))-8)/(1-((\omega_{ij}/(((3+r_j)+r_j)+((\omega_{ij}\cdot 4)/1)))/(6-(8+r_j)))))/(r_j-r_j)))))-4)/4)/6)+r_j)))/(5+r_j))+\omega_{ij})-\omega_{ij})/6))-\omega_{ij}).$$

Anhang 5: Quellcode

Für Gewichtsmatrizen wurde die dünn-besiedelte Matrix Klasse der Eigen-Bibliothek [7] benutzt. Die Lernregeln wurden zur Evaluierung mit AsmJit [10] zu Maschinencode kompiliert. Alles andere wurde in C++ unter Benutzung der stl-Bibliotheken geschrieben. Weitere Fitnessfunktionen können, wie anhand der Beispiele klar werden sollte, sehr leicht hinzugefügt werden.

Quellcode 1: Update Vorschrift für neuronale Netzwerke

```
void neuralNetwork::update(){
  activationVector=weightMatrix*activationVector;

  for(int i=0;i<activationVector.size();++i){
    activationVector(i)=activationFunction->evaluate(
       activationVector(i));
  }

  for (int k=0; k<weightMatrix.outerSize(); ++k)
    for (mType::InnerIterator it(weightMatrix,k); it; ++it){
      if(!staticNeurons[it.row()]){
        synapticParameter params={it.value(),activationVector(it.
           row()),activationVector(it.col())};
        it.valueRef()+=learningRule->evaluate(params);
      }
    }
}
```

Quellcode 2: Fitnessfunktion für die "Einzelnes Target" Simulation

```
  double target::evaluate(neuralNetwork &net){
    double fitness=0,dx,dy,dist;
    double x=rand()/double(RAND_MAX)*2.-1;
    double y=rand()/double(RAND_MAX)*2.-1;

    for (int i=0; i<steps; ++i) {
      net.setNetInput(0, targetX-x);
      net.setNetInput(1, targetY-y);

      net.update();

      dx=net[net.size()-2];
      dy=net[net.size()-1];

      x+=dx;
      y+=dy;

      dist=(targetX-x)*(targetX-x)+(targetY-y)*(targetY-y);
      fitness+=exp(-dist);
    }
    fitness/=steps;
    return fitness;
  }
```

Quellcode 3: Fitnessfunktion für die "Energielandschaft" Simulation

```
double landscape::evaluate(lars::neuralNetwork &net){
  double x=initialX;
  double y=initialY;

  double energyCost=0,networkEnergy=0;
  double currentEnergy,dx,dy;

  for (int i=0; i<steps; ++i) {
    energyCost+=(currentEnergy=getPotential(x, y));
    net.setNetInput(0, (targetX-x)/diag);
    net.setNetInput(1, (targetY-y)/diag);
    net.setNetInput(2, getPotential(x, y+1)-currentEnergy);
    net.setNetInput(3, getPotential(x+1, y)-currentEnergy);
    net.setNetInput(4, getPotential(x, y-1)-currentEnergy);
    net.setNetInput(5, getPotential(x-1, y)-currentEnergy);

    net.update();

    networkEnergy+=net.weights().squaredNorm();

    dx=net[net.size()-2];
    dy=net[net.size()-1];

    x+=dx;
    y+=dy;
  }

  networkEnergy/=net.weights().size()*steps;

  double dist=(targetX-x)*(targetX-x)+(targetY-y)*(targetY-y);
  dist/=diag;
  dist=sqrt(dist);

  energyCost/=steps;

  return exp(-energyCost)*(2-dist)/(2+0.1*networkEnergy);
}
```

Quellcode 4: Genetischer Algorithmus

```
population geneticAlgorithm::evolve(population &pop){

  evaluatePopulation(pop);
  population evolved(pop.settings);

  for(int i=0;i<pop.size()*pop.settings.eliteRate;++i){
    evolved.insert(pop[i]->clone());
  }

  genome *crossTmp[2];

  while (evolved.size()<pop.size()) {
    double r=rand()/double(RAND_MAX);

    if(r<pop.settings.mutationRate){
      evolved.insert(select(pop)->mutatedClone());
    }
    else if(r<pop.settings.mutationRate+pop.settings.crossoverRate
       ){
      select(pop)->crossedClone(select(pop), crossTmp);
      evolved.insert(crossTmp[0]);
      evolved.insert(crossTmp[1]);
    }
    else evolved.insert(select(pop)->clone());
  }

  evolved.resize(pop.size());

  return evolved;
}
```

Erklärung

Nach §13(8) der Prüfungsordnung für den Bachelor-Studiengang Physik und den Master-Studiengang Physik an der Universität Göttingen:

Hiermit erkläre ich, dass ich diese Abschlussarbeit selbständig verfasst habe, keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe und alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten Schriften entnommen wurden, als solche kenntlich gemacht habe.
Darüberhinaus erkläre ich, dass diese Abschlussarbeit nicht, auch nicht auszugsweise, im Rahmen einer nichtbestandenen Prüfung an dieser oder einer anderen Hochschule eingereicht wurde.

Göttingen, den 8. Juni 2013

Lars Melchior