Bachelorarbeit

Einführung in die Neuroevolution am Beispiel eines Computerspiels

Matrikel Nr.: 15314

vorgelegt von: Jan-Paul Schulz
Prohner Straße 31a
18435 Stralsund
[redacted]

E-Mail: jan-paul.schulz@fh-stralsund.de

1. Gutachter: Prof. Dr. rer. nat. Thomas Wengerek
2. Gutachter Prof. Dr. rer. nat. Gero Szepannek
Hochschule Stralsund
Zur Schwedenschanze 15
18435 Stralsund

Eingereicht am: Montag, 25. Februar 2019

Hochschule Stralsund, Fakultät Wirtschaft
Studiengang Wirtschaftsinformatik

Erklärung

Hiermit erkläre ich, die vorliegende Arbeit selbständig verfasst und keine anderen, als die angegebenen Hilfsmittel benutzt zu haben. Die Stellen der Hausarbeit, die anderen Quellen im Wortlaut oder dem Sinn nach entnommen wurden, sind durch Angaben der Herkunft kenntlich gemacht. Dies gilt auch für Zeichnungen, Skizzen, bildliche Darstellungen, sowie für Quellen aus dem Internet. Die Arbeit hat mit gleichem bzw. in wesentlichen Teilen gleichem Inhalt noch keiner Prüfungsbehörde vorgelegen. Weiterhin stimme ich zu, dass von der Arbeit eine elektronische Kopie gefertigt und gespeichert werden darf, um eine Überprüfung mittels einer Plagiatssoftware zu ermöglichen.

Stralsund, den 25. Februar 2019

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Glossar

Ruhepotenzial	Die Potenzialdifferenz zwischen einer Zelle und seinem Milieu.
Aktionspotenzial	Eine vorübergehende Abweichung des Ruhepotenzials.
Neuron	Eine, auf Informationsübertragung, spezialisierte Zelle.
Neurotransmitter	Botenstoffe, welche ein Aktionspotenzial chemisch übertragen.
Permeabilität	Durchlässigkeit für bestimmte Stoffe.
Kognition	Die Umgestaltung von Informationen.
Ion	Ein elektrisch geladenes Atom oder Molekül.
Genotyp	Summe der Erbinformationen eines Individuums.
Overfitting	Die Überanpassung eines Modells an Trainingsdaten.
Ära	Eine von einem bestimmten Ereignis ausgehende Zeitzählung. Hier: 500 Generationen der Neuroevolution.
Durchschnitt	Aus mehreren vergleichbaren Größen errechneter Mittelwert. In dieser Arbeit ist immer das arithmetische Mittel gemeint.
Quantil	Eine Größe der Statistik, welche eine Stichprobe in n Mengen gleicher Kardinalität teilt.
Median	Das 0,5 Quantil, teilt eine Stichprobe in zwei Mengen gleicher Kardinalität.
Quartil	Als Quartil werden die Quantile bezeichnet, welche die Stichprobe in 4 Mengen einteilt. Zwischen ersten und dritten bzw. oberen und unteren Quartil befindet sich die Hälfte der Stichprobe.

1. Einleitung

1.1. Abstract

Im Rahmen dieser Bachelorarbeit wird Neuroevolution als mächtige Alternative zu anderen Trainingsverfahren vorgestellt. Am Beispiel des Computerspiels Snake wird gezeigt, dass komplexe Problemsequenzen mit unterschiedlichen Problemklassen mithilfe von Neuroevolution optimiert werden können, solange der Anwender bessere bzw. schlechtere Lösungen differenzieren kann. Ein vollständiges Verständnis der Problemsequenzen des Anwenders wird dafür nicht benötigt. Des Weiteren werden die Lernerfolgsfaktoren von vier Parametern der Neuroevolution untersucht und geschildert. Hierfür wurde der Lernfortschritt nach je 500 Generationen gesichert. Diese 500 Generationen werden als Ära bzw. als Iteration bezeichnet. Je Attributausprägung wurden zehn Iterationen gerechnet und anschließend miteinander verglichen. Die Ergebnisse dieser Untersuchungen sind dabei kritisch zu betrachten, da je Attributausprägung lediglich zehn Iterationen gerechnet wurden. Deshalb dienen die Ergebnisse eher als Indiz für die tatsächlichen Parametereinflüsse. Es wurde ein positiver Lerneinfluss von größeren Populationen festgestellt, ein Erweitern des Suchraums durch Erhöhen der Mutationsrate sowie ein Verringern der Streuung um den Erwartungswert durch eine kleinere Anzahl von Eliten je Generation. Ein Verringern der Mutationsrate bewirkt ebenfalls eine Reduzierung der Streuung um den Erwartungswert, allerdings bedeutet dies auch eine Eingrenzung des Suchraums. Darüber hinaus wurden weitere Erkenntnisse über das Lernverhalten der Neuroevolution beobachtet und geschildert. Es wird gezeigt, dass der Lernfortschritt der Neuroevolution stufenweise entsprechend der Problemklassen der Problemsequenz stattfindet.

1.2. Motivation

Das Interesse an künstlicher Intelligenz ist in den letzten drei Jahren stark gestiegen [1]. Die Vorstellung, komplexe Probleme von einer künstlichen Intelligenz lösen zu lassen, ist sehr verlockend. Seit der industriellen Revolution entwickelt und baut die Menschheit Maschinen, um sich die Arbeit zu erleichtern oder sie komplett von Maschinen bewerkstelligen zu lassen. Ähnlich den Maschinen, dienen Software und Computer ebenfalls dem Zweck, dem Menschen die Arbeit zu erleichtern oder ggf. komplett abzunehmen. Der größte Unterschied besteht darin, dass eine herkömmliche Maschine körperliche Arbeit und Software geistige Arbeit verrichtet. Maschinelles Lernen befasst sich mit der Informationsgewinnung aus Daten, zumeist der Mustererkennung, um problembewältigende Software zu entwerfen. Der Fortschritt der Problembewältigung durch Maschinelles Lernen ist so weit, dass für den Entwickler der Software nicht mehr notwendig ist, dass zu bewältigende Problem selbst lösen zu können, mehr noch, zum Teil ist es nicht mehr notwendig das Problem an sich definieren zu können. Mithilfe der Neuroevolution ist es möglich, Lösungen zu optimieren, sofern zwischen besseren und schlechteren Lösungen differenziert werden kann. Im Rahmen dieser Bachelorarbeit zeigt eine

wissenschaftliche Einführung in die Neuroevolution ihre Fähigkeit komplexe Probleme zu erlernen, ohne diese vollständig zu definieren. Zunächst werden die Grundlagen und der biologische Ursprung erörtert. Darauf folgt die Beschreibung, des für die Bachelorarbeit angefertigten Prototyps, der Experimentaufbau und die Experimentdurchführung und abschließend die Evaluation der erzielten Ergebnisse.

1.2. Wissenschaftliche Einordnung

Bei der Neuroevolution werden neuronale Netze mithilfe eines Evolutionsalgorithmus trainiert. Sie stellt eine Alternative zum herkömmlichen überwachten Lernen dar. Im Gegensatz zum überwachten Lernen, bei dem neuronale Netze mittels Inputwerten und dazugehörigen Outputwerten trainiert werden, bietet die Neuroevolution die Möglichkeit, lediglich mithilfe einer Evaluation des Outputs einen Lösungsansatz zu optimieren. Da die Neuroevolution eine Alternative zu den herkömmlichen Lernverfahren bietet, gehört sie zur Disziplin des maschinellen Lernens. Maschinelles Lernen wird als Schlüsseltechnologie der künstlichen Intelligenz verstanden, welche wiederum ein Teilgebiet der Informatik ist. Maschinelles Lernen befasst sich mit der ‚Generierung von Wissen aus Erfahrung' indem Algorithmen basierend auf Trainingsdaten Modelle und Strukturen erlernen. [2, p. 9]

1.3. Stand der Technik

Im Vergleich zu den beiden zu Grunde liegenden Techniken der Neuroevolution, zum einen den neuronalen Netzen und zum anderen den Evolutionsalgorithmen, ist die Neuroevolution eine verhältnismäßig junge Disziplin. Bereits 1949 hat Donald Hebb die Grundlagen des neuronalen Lernens untersucht und erste Modelle unüberwachten Lernens entworfen [3]. Das erste künstliche neuronale Netz, das Perzeptron erfand Frank Rosenblatt 1958 [4]. Die Erfindung der Evolutionsalgorithmen liegt ähnlich lange zurück. Bereits 1970 beschrieb Alex Fraser die Lösungsoptimierung mithilfe der Evolutionsoperationen Selektion, Rekombination und Mutation [5]. Die ersten wissenschaftlichen Artikel über Neuroevolution bzw. dem Modellieren neuronaler Netze mithilfe von Evolutionsalgorithmen erschienen erst Anfang der neunziger Jahre. So wurde beispielsweise 1993 ein Artikel über das Konstruieren von rekurrenten, neuronalen Netzen mithilfe von Evolutionsalgorithmen veröffentlicht [6].

2. Grundlagen

2.1. Biologische Informationsverarbeitung

Die biologischen Neuronen sind auf Informationsverarbeitung bzw. Informationsübertragung spezialisierte Zellen. Ein Neuron empfängt Aktionspotenziale vorgeschalteter Neuronen über seine fein verästelten Nervenzellfortsätze, Dendriten genannt, und trägt es über sein Axon weiter zu den Synapsen. Zwischen den Synapsen und den Dendriten des nachfolgenden Neurons befindet sich der synaptische Spalt. Der synaptische Spalt überträgt die Informationen mit Hilfe von Neurotransmittern. Man unterscheidet zwei Arten von Synapsen, exzitatorische bzw. erregende Synapsen und Inhibitorische bzw. hemmende Synapsen. Dabei sind die Neurotransmitter maßgebend für die Art der

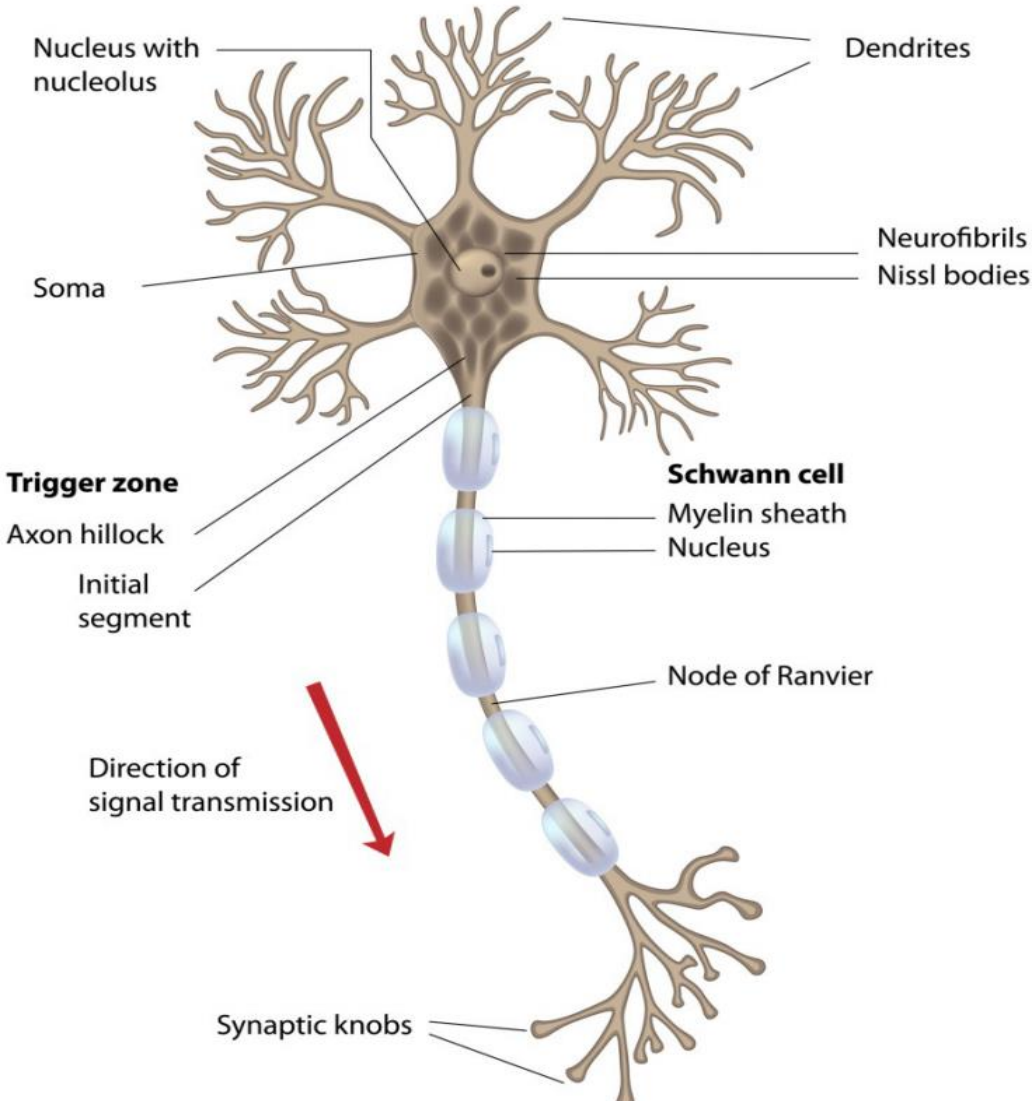

Abbildung 1: Typisches biologisches Neuron

Quelle: [11, p. 46]

Wirkung und entscheidend für die Informationsübertragung über den synaptischen Spalt.

Neuronen haben eine selektiv permeable Zellmembran, sprich eine Membran mit Ionenkanälen, welche den Austausch spezifischer Ionen ermöglichen. Durch Diffusion bestreben die Teilchen eine gleichmäßige Verteilung im umliegenden Milieu. Aufgrund der Ionenkanäle ist die Diffusion auch eingeschränkt über die Zellmembran hinaus möglich. Dies hat zu Folge, dass Kationen die Neuronen über die Ionenkanäle aufgrund der Diffusionskraft verlassen, bis die elektrostatische Kraft die Diffusionskraft aufhebt und ein Gleichgewichtszustand erreicht wird. Das Neuron ist gegenüber dem Außenmedium negativ geladen. Diese Potenzialdifferenz wird Ruhepotenzial genannt und liegt zwischen -70 und -80 Millivolt. Als Aktionspotenzial gilt eine Depolarisation der Zelle. Dies tritt ein, sobald das Schwellenpotenzial der spannungssensitiven Ionenkanäle überschritten wird. Kationen strömen über die Ionenkanäle in die Zelle, sodass die ursprünglich negativ geladene Zelle nun positiv gegenüber ihrer Umgebung geladen ist. [7, pp. 9 - 24]

2.1.1. Synaptische Plastizität

Unter synaptischer Plastizität versteht man die aktivitätsabhängige Anpassung der Stärke des Aktionspotenzials durch die Synapsen. Die synaptische Plastizität ist ein neurophysiologischer Mechanismus für Lernprozesse und Gedächtnisbildung. Man unterscheidet zwischen Kurzzeitplastizität, wo Anpassungen nur für eine kurze Zeit anhalten, und Langzeitplastizität, bei der die Anpassungen der Signalübertragung bis zu mehreren Jahren anhalten. [7, p. 29]

2.1.2. Summation

Unter Summation versteht man die räumliche- und zeitliche Summierung der eintreffenden Aktionspotenziale. Die Summation ist die Hauptfunktion des Dendriten-Baums und am Soma (Zellkörper) abgeschlossen. Sowohl hemmende als auch erregende eintreffende Potenziale werden zu einem einzigen, entsprechend starken Potenzial aufsummiert. Wenn eintreffende Potenziale eine geringe Zeitverzögerung von ein paar Millisekunden haben, werden diese ebenfalls zu einem einzigen Aktionspotenzial aufsummiert. [7, pp. 26 - 28]

2.2. Künstliche Neuronale Netze

Als Neuronales Netz bezeichnet man eine Menge miteinander verbundener Neuronen, welches einer spezifischen, kognitiven Funktion oder Aufgabe, zumeist der Mustererkennung, dient. Künstliche neuronale Netze sind den Nervensystemen von Tieren und Menschen nachempfunden, allerdings nicht mit dem Ziel, Nervensysteme abzubilden oder zu simulieren. Ein erwachsener Mensch hat etwa

100 Billionen Synapsen. Aufgrund der Größe von Nervensystemen ist eine Simulation zurzeit ausgeschlossen.

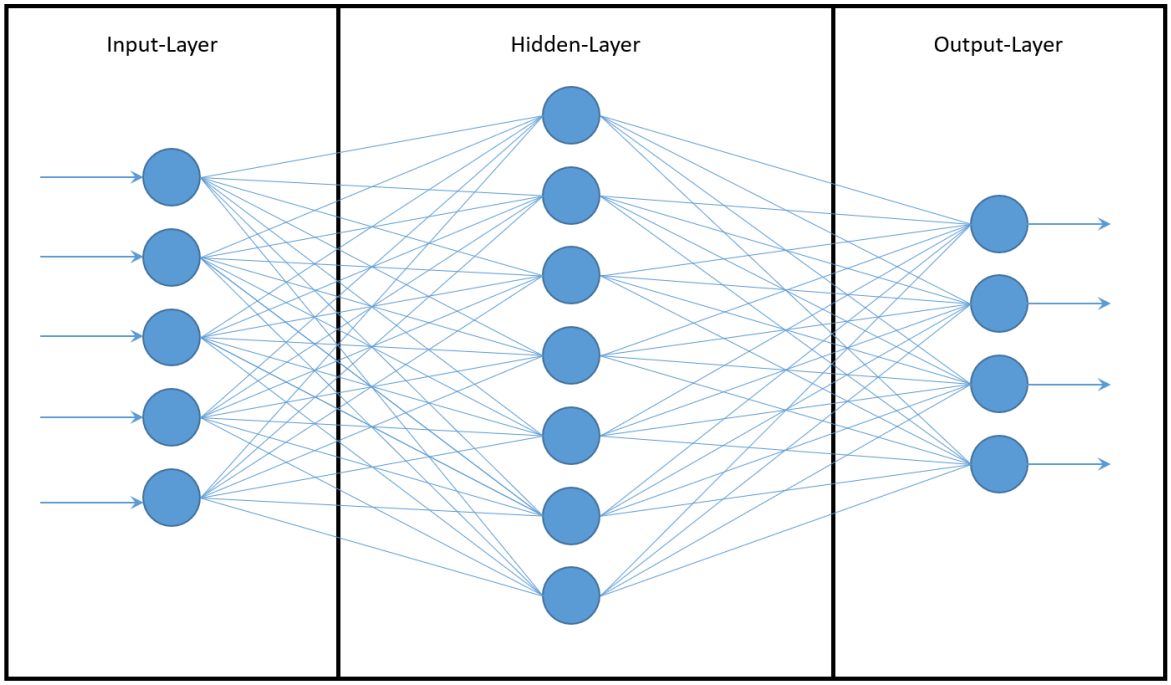

Abbildung 2: Künstliches neuronales Netz

Quelle: eigene Darstellung

Abbildung 2 zeigt eine Visualisierung eines künstlichen neuronalen Netzes. Jedes Neuron einer Schicht ist jeweils mit allen Neuronen der vorhergehenden Schicht verbunden. Diese Verbindungen liefern den Input des Neurons. Jedes Neuron gibt seinen Output an jedes Neuron der folgenden Schicht weiter. Ähnlich dem Vorbild aus der Natur werden Inputs mit der Wichtung der Verbindung bzw. Synapse multipliziert und anschließend aufsummiert, um einen Output zu erzeugen, sofern der Schwellenwert des Neurons überschritten wird. Der Output des Neurons dient wiederum als Input für ein folgendes Neuron oder als Output des neuronalen Netzes. Dabei sind die Inputs mit den kognitiven Reizen unserer Sinnesorgane zu vergleichen. Nach dem Aufsummieren der Inputwerte wird mittels einer Aktivierungsfunktion der Output auf einen Wertebereich zwischen 0 und 1 abgebildet. Der Output mit dem größten Wert hat die höchste Wahrscheinlichkeit, korrekt zu sein. Als Aktivierungsfunktion wurde eine Sigmoidfunktion verwendet, da sie zu den meist verbreitetsten Aktivierungsfunktionen gehört. [8, p. 115] Die Sigmoidfunktion, die in dieser Arbeit verwendet wurde ist wie folgt definiert:

$$sig(x) = \frac{e^x}{1 + e^x}$$

Sei o_j der Output des Neurons j, w_{ij} die Wichtung der Verbindung zwischen Neuron i und j, n die Anzahl der Neuronen der vorherigen Schicht und $sig(x)$ die Aktivierungsfunktion, dann berechnet sich der Output eines jeden Neurons mit folgender Formel:

$$o_j = sig\left(\sum_{i=1}^{n} w_{ij} * o_i\right)$$

Damit das neuronale Netz entsprechend seiner Aufgabe genutzt werden kann, muss es zunächst trainiert werden.

2.2.1. Trainieren neuronaler Netze

In dem man Wichtungen einzelner Synapsen oder Schwellenwerte anpasst, kann man neuronale Netze trainieren, um Muster zu erkennen. Das Training findet dabei iterativ statt. Deshalb werden neuronale Netze unter anderem im Bereich der Data Science verwendet, um Klassifikations- und Prognoseprobleme zu lösen. Man unterscheidet drei Arten von Training für neuronale Netze:

- überwachtes Lernen
- unüberwachtes Lernen
- bestärkendes Lernen

2.2.1.1. Überwachtes Lernen

Überwachtes Lernen bezeichnet das Trainieren neuronaler Netze mittels Input und erwarteten Output Paaren. In Folge des Trainings wird eine Verlustfunktion minimiert. Diese berechnet eine Güte bzw. einen Fehler über die Abweichung von erwartetem und tatsächlichem Ergebnis. Einer der meistverwendeten Algorithmen zum Minimieren der Verlustfunktion ist der Backpropagationsalgorithmus. Backpropagation berechnet für jede Wichtung den Gradienten der Verlustfunktion und skaliert diesen mit einer Lernrate. [9, pp. 151-172] [10]

2.2.1.2. Unüberwachtes Lernen

Im Gegensatz zum überwachten Lernen verwendet das unüberwachte Lernen keine Zielwerte oder Verlustfunktion, um den Lernprozess zu steuern. Die Anpassungen der Wichtungen und Bias-Werte erfolgt aufgrund der In- und Outputs des neuronalen Netzes bzw. des einzelnen Neurons. Unüberwachte Lernalgorithmen orientieren sich an den Lernprozessen natürlicher neuronaler Netze, da der natürliche, biologische Lernprozess der neuronalen Netze des Nervensystems ebenfalls ohne Belohnungsfunktion oder Lehrer funktioniert und lernt. Man unterscheidet zwei Arten von unüberwachtem Lernen.
Zum einen Algorithmen, welche das neuronale Netz als Ganzes betrachten und die einzelnen Neuronen im Wettkampf zueinander stehen. Nach jeder Iteration werden die Wichtungen des besten Neurons adjustiert. Ein Beispiel für einen kompetitiven Lernalgorithmus sind die Kohonen-Maps.
Zum anderen Algorithmen, welche jedes einzelne Neuron des Netzes entsprechend der synaptischen Plastizität trainieren. Neuronen, welche wiederholt gleichzeitig aktiviert werden, werden miteinander assoziiert. „Neurons that fire together, wire

together.“ [3] Ein Beispiel für solch einen Lernalgorithmus ist der Hebbian-Algorithmus. [11, pp. 66 - 69]

2.2.1.3. Bestärkendes Lernen

Bestärkendes Lernen ist Lernen mit dem Ziel, eine Belohnungsfunktion zu maximieren. Dabei wird dem neuronalen Netz nicht vorgegeben, wie Signale zu interpretieren sind oder ob explizite Interpretationen korrekt oder fehlerbehaftet sind. Das neuronale Netz lernt nach dem trial-and-error Prinzip. Lösungsstrategien müssen ausprobiert werden und in dem Fall, dass das neuronale Netz auf eine Problemsequenz trainiert wird, wird erst nach Abschluss der Sequenz die Evaluation und somit die Belohnung angewandt. Bei bestärkendem Lernen wird die Interaktion mit der Umgebung durch die Interpretationen und Aktionen des neuronalen Netzes als Basis der Evaluation verwendet. Für das Anpassen der Wichtungen und Schwellenwerte werden zumeist Monte-Carlo-Algorithmen verwendet. Durch das zufällige Verändern der Attribute der neuronalen Netze werden verschiedene Ergebnisse produziert, welche wiederum anhand der Belohnungsfunktion evaluiert werden können. [12]

2.2.2. Implementierung

Künstliche neuronale Netze bestehen aus drei funktionalen Komponenten, den Wichtungen der Synapsen, den Schwellenwerten der Neuronen und der Aktivierungsfunktion. Der Schwellenwert kann durch einen Bias-Wert implementiert werden. Der Bias-Wert, der vor dem Anwenden der Aktivierungsfunktion auf die Summe der Produkte addiert wird, fungiert als Schwellenwertersatz. Durch den Bias-Wert werden Neuronen zum ‚feuern‘ begünstigt oder gehemmt. Eine Begünstigung durch einen verhältnismäßig hohen Bias-Werts ist mit einem geringen Schwellenwert zu vergleichen. Der Bias-Wert verschiebt die, durch das neuronale Netz abstrahierte Funktion entlang der X-Achse. Die Wichtungen der Verbindungen werden in Matrizen realisiert, die Schwellenwerte bzw. die Bias-Werte werden als Vector implementiert. Der Biasvektor enthält die Bias-Werte aller Neuronen einer Schicht des neuronalen Netzes. Eine Wichtungsmatrix wird mit M Zeilen und N Spalten initialisiert. Dabei entspricht M der Anzahl der Neuronen der folgenden Schicht und N der Anzahl der Neuronen der vorangegangenen Schicht. Die Aktivierungsfunktion kann als Funktion implementiert werden und benötigt keine weitere Abstrahierung. So ist es möglich, die Vorhersage bzw. die Funktionalität des neuronalen Netzes auf Schichtebene zu berechnen. Wenn die Wichtungen und Bias Werte initialisiert bzw. trainiert sind, benötigt das neuronale Netz einen Inputvektor, um eine Vorhersage zu treffen.

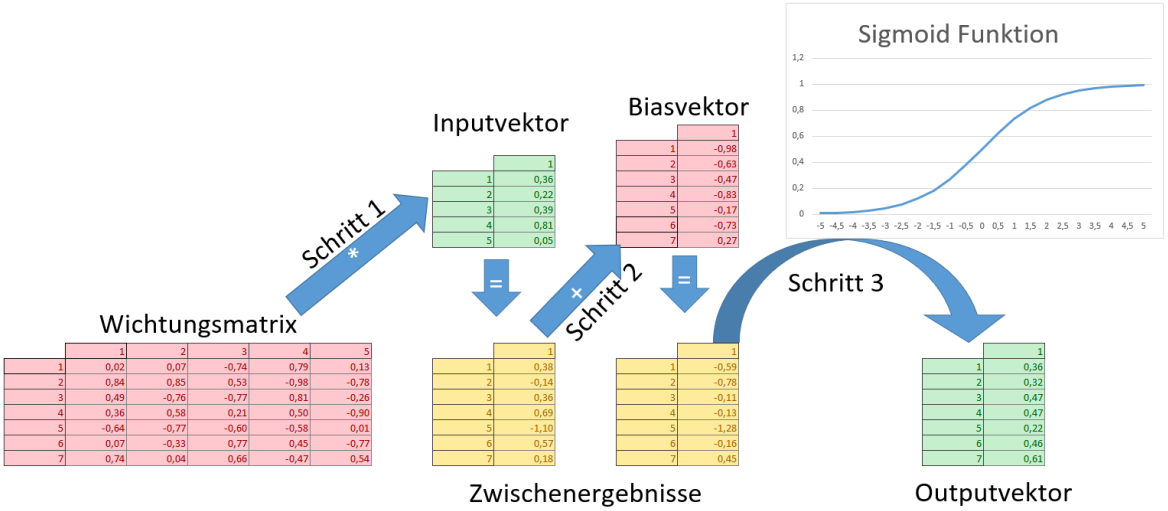

Abbildung 3: Berechnung der Vorhersage des neuronalen Netzes

Abbildung 3 zeigt den Berechnungsvorgang einer Schicht beim Treffen einer Vorhersage. Dabei sind die roten Matrizen die funktionalen Attribute des neuronalen Netzes (Wichtungsmatrix und Biasvektor). Der Inputvektor und der berechnete Outputvektor sind grün dargestellt und die Zwischenergebnisse sind gelb eingefärbt. Die Outputs werden berechnet, indem zunächst der Inputvektor per Matrizenmultiplikation mit dem Inputvektor multipliziert wird (Schritt 1), anschließend wird der Biasvektor addiert (Schritt 2) und als letzter Schritt wird die Aktivierungsfunktion angewandt (Schritt 3). Diese Berechnung findet iterativ für alle Schichten des neuronalen Netzes statt.

2.3. Evolutionsalgorithmen

Evolutionsalgorithmen sind Algorithmen zur näherungsweisen Lösung von Optimierungsproblemen. Sie sind der Evolutionstheorie von Charles Darwin nachempfunden. In Generationen genannten Iterationen wird die Population, also die Menge der Individuen einer Generation, evaluiert, selektiert und mittels Rekombination und Mutation diversifiziert. Jedes Individuum repräsentiert eine Lösung für das Optimierungsproblem. Mithilfe einer Fitnessfunktion wird die Güte der Lösung evaluiert, welche wiederum als Selektionskriterium dient. Die Fitness eines jeden Individuums wird in Abhängigkeit der gesamten Fitness der aktuellen Population gemessen. Bei der Selektion ist jedoch zu beachten, dass Individuen mit höherer Fitness lediglich eine höhere Wahrscheinlichkeit haben, als Elternteil eines Individuums der kommenden Generation selektiert zu werden. Ebenso können Individuen mit verhältnismäßig geringer Fitness, allerdings auch mit geringerer Wahrscheinlichkeit als Elternteil selektiert werden. Um ein Individuum der kommenden Generation zu erzeugen, werden zwei Individuen der aktuellen Generation selektiert und ihre Genotypen miteinander rekombiniert. Unter der Rekombination versteht man in der Genetik die Neukombination des vererbten Genotyps des Nachkommen bei der Fortpflanzung. Der Genotyp des Erbenden setzt sich aus den Genotypen der Eltern zusammen. Dabei bestimmt der Zufall über das Verhältnis der Erbanteile. Zur Darstellung des Genotyps wird eine genetische

Repräsentation, eine Aufreihung der Schlüsselattribute der Individuen bzw. Lösungen verwendet.

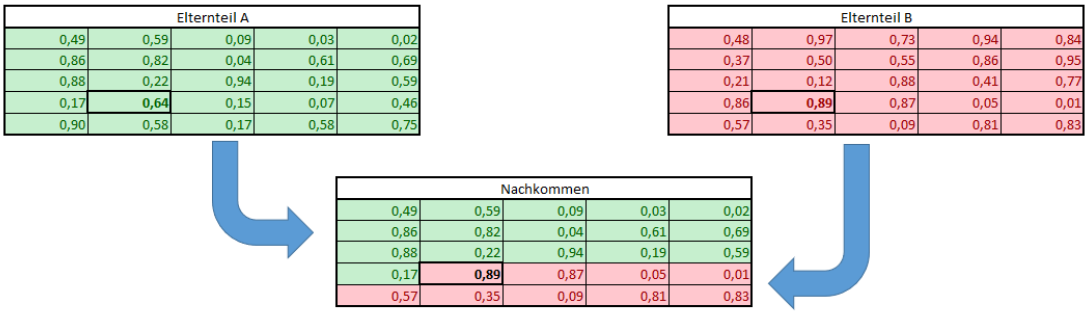

Abbildung 4: Vereinfachte genetische Repräsentation eines Genoms eines künstlichen neuronalen Netzes

Quelle: eigene Darstellung

Abbildung 4 zeigt den Rekombinationsprozess der Vererbung. In der Abbildung werden zwei beispielhafte Wichtungsmatrizen als genetische Repräsentation eines Genoms gezeigt. Eine Zelle der Matrizen wird zufällig als Trennpunkt bestimmt. Bis zu diesem Punkt wird der Genotyp des Elternteils A vererbt und ab diesem Punkt der Genotyp des Elternteils B. Anschließend werden die Nachkommen zufällig mutiert. Für jedes Attribut des Genotyps wird mit einer Wahrscheinlichkeit, der sogenannten Mutationsrate, das Attribut manipuliert. Diese Attributveränderung wird mithilfe eines zufälligen, normalverteilten Wertes, der auf den ursprünglichen Attributwert addiert wird, realisiert. Der Versatz ist normalverteilt, um geringe Veränderungen des Attributs häufiger zu gestalten, sodass starke Mutationen zwar möglich, aber seltener sind. Die daraus resultierende Variabilität innerhalb einer Population deckt so einen breit gefächerten Lösungsraum ab. Neben den, durch die Natur inspirierten Evolutionsoperationen bieten Evolutionsalgorithmen weitere Methoden zum Erhalt von erfolgreichen Individuen. So bezeichnet man das unveränderte Weitertragen von besten Individuen einer Generation in die nächste als Elitismus. Dies hat zum Vorteil, dass trotz der Manipulation durch die Evolutionsoperationen kein Rückschritt im Evolutionsprozess zu verzeichnen ist. [13]

2.4. Neuroevolution

Als Neuroevolution bezeichnet man die Kombination von neuronalen Netzen und Evolutionsalgorithmen. Dabei repräsentiert jedes Individuum einer Generation ein neuronales Netz. Der Genotyp des Individuums entspricht den Schlüsselattributen, also den Wichtungen und Bias-Werten eines neuronalen Netzes. Neuroevolution findet, gleich den Evolutionsalgorithmen, in vier Phasen statt, welche iterativ wiederholt werden. Dabei gilt zu beachten, dass die neuronalen Netze nicht basierend auf ihren funktionalen Eigenschaften, sondern aufgrund der erzielten Ergebnisse bewertet werden.

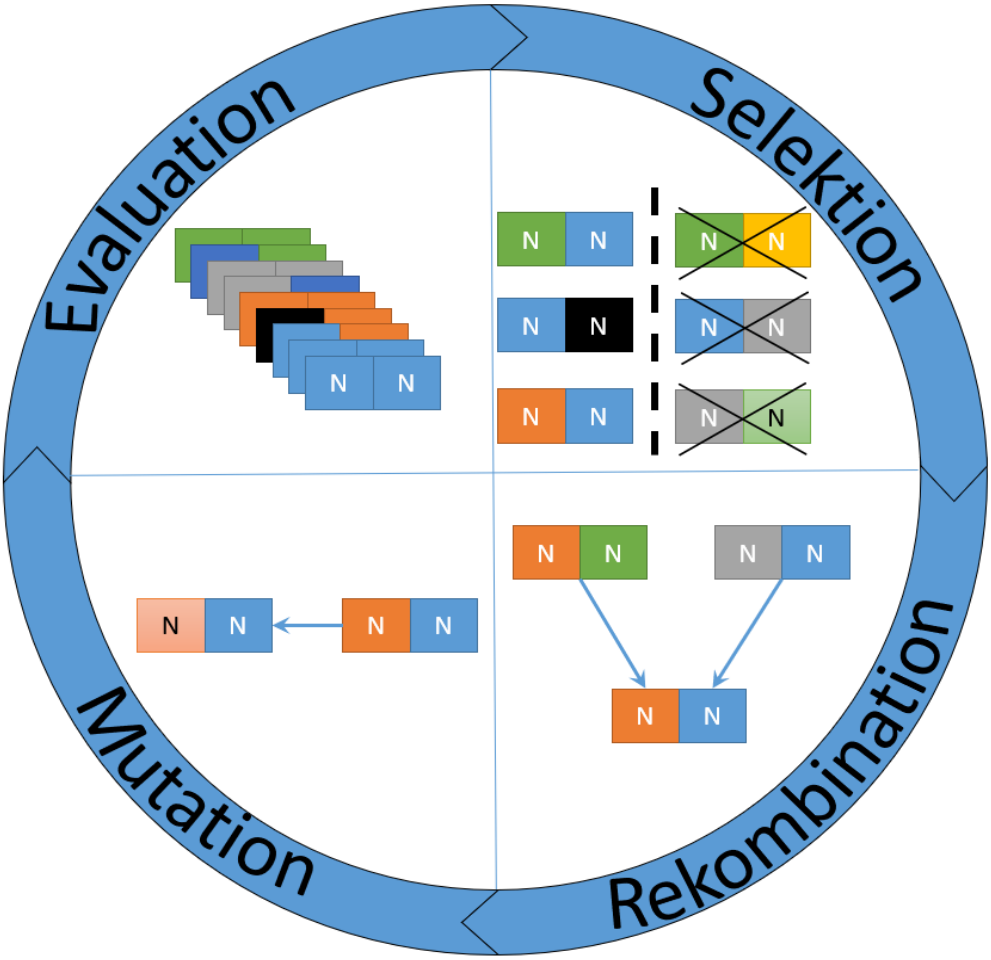

Abbildung 5: Visualisierung der Neuroevolution

Quelle: eigene Darstellung

Abbildung 5 ist eine Visualisierung der vier Phasen der Neuroevolution. In der Evaluationsphase erhält jedes Individuum der Population die gleiche Aufgabe bzw. die gleichen Aufgabenfolge. Nach dem Bearbeiten der Aufgabe werden die neuronalen Netze auf Basis der erbrachten Ergebnisse evaluiert und anschließend selektiert. Dabei werden die neuronalen Netze nicht trainiert, die Anpassung der Wichtungen und Schwellenwerte findet ausschließlich über die Rekombination und Mutation des Evolutionsalgorithmus statt. Dies ermöglicht ein erfolgsorientiertes Trainieren von neuronalen Netzen. Lösungsstrategien müssen nicht vorgegeben oder implementiert werden, da sie dynamisch während der Neuroevolution entstehen.

Je nach Anwendung und Aufgabe kann die Evaluationsphase aus mehreren, zu bewältigenden Problemen bestehen. So können komplexe Probleme mit sich ändernden Bedingungen und verschiedenen Problemklassen als eine Reihe von Entscheidungsfindungsproblemen betrachtet werden. Dies ermöglicht es neuronalen Netzen, ein Computerspiel spielen zu lassen, indem es in jeder Situation, die dem Spieler eine Aktion erlaubt, einen Inputvektor auf die möglichen Eingaben des Spiels abbildet. Dabei sollte der Inputvektor aus, für die Bewältigung der gegebenen Aufgabe relevanten, Informationen bestehen. Beim Trainieren von

neuronalen Netzen, zum Erlernen eines Computerspiels, existieren viele Möglichkeiten das Lernpotenzial zu beeinflussen, da bei zugänglichem Quellcode des Spiels jede Komponente der Neuroevolution angepasst werden kann. Unter anderem ist es möglich, das Problem bzw. das Spiel selbst anzupassen, um Einfluss auf den Lernerfolg zu nehmen. Allerdings sind solche Anpassungen sorgfältig zu wählen, da das Verändern des zu lösenden Problems nicht die Lösung des ursprünglichen Problems ist. Aber solche Anpassungen können auch vorteilhaft sein, da sie die Evaluationsphase und somit den Lernprozess verkürzen können.

2.5. Zufallszahlengeneratoren

Zufallszahlengeneratoren (RNG) sind Algorithmen, die von einem Ursprungswert ausgehend eine Reihe zufälliger Zahlen generieren. Dabei unterscheidet man zwischen deterministischen- und nichtdeterministischen RNGs. Ideale RNGs würden gleichverteilte, unabhängige Zahlenfolgen in einem endlichen Wertebereich liefern. „*An ideal random number generator, however, is fiction.*“ [14, p. 1]
Grundsätzlich besteht jeder RNG aus einem Startwert, dem sogenannten Seed, sowie einem Algorithmus, der eine neue, vermeintlich zufällige Zahl, ausgehend von der vorherigen Zufallszahl generiert. Einer der bekanntesten Algorithmen für deterministische RNGs ist der lineare Kongruenzgenerator. [15] Dieser Algorithmus besteht aus einem Modul m>0, Faktor a<m, Inkrement b<m und einem Startwert bzw. Seed X_0. Um eine neue Zahl zu berechnen, verwendet man folgende Formel:

$$X_{n+1} = (x * X_n + b)\ mod\ m$$

Für moderne RNGs, wie zum Beispiel der Implementierung des RNGs von Java (java.util.Random), werden ähnliche Algorithmen mit zusätzlichen Bitshifting-Operationen verwendet. Beim logischen Bitshifting (in Java und Javascript: x << n und x>>>n) werden die Bits einer Variablen um n Positionen verschoben. Diese Operationen sind äquivalent zu den Rechnungen $x * 2^n$ für einen Bitshift nach links und $\frac{x}{2^n}$ für einen Bitshift nach rechts. Als Seed wird hier, sofern kein Seed manuell festgelegt wurde, die Systemzeit in Millisekunden seit Mitternacht 01.01.1970 UTC verwendet. [16] [17]
Für nicht-deterministische RNGs werden ebenso deterministische Algorithmen verwendet, allerdings wird der Seed mithilfe von Sensordaten bestimmt. Dafür werden physikalische Prozesse mit rauschendem- oder spontanen Verhalten beobachtet oder Benutzerinputs als Ausgangswert verwendet. Solche Prozesse sind zum Beispiel Widerstandsrauschen, Atmosphärenrauschen oder radioaktiver Zerfall. Für die Schlüsselgenerierung in der Kryptographie werden mitunter Benutzer gebeten, ihre Maus in einem Feld zu bewegen, um ausgehend von der menschlichen Willkür, welche sich in der Mausbewegung wiederfindet, nicht deterministische Zufallszahlen zu erzeugen.

3. Prototypische Umsetzung

Um die Lernerfolgsfaktoren der Neuroevolution zu untersuchen, wurde für die Implementierung das neuronale Netz, sowie der Evolutionsalgorithmus bzw. der Genetic Algorithim von Daniel Shiffman [18] verwendet und ergänzt, um das Spiel ‚Snake' zu erlernen. Dafür wurde die Snake-Implementierung, von Prashant Gupta [19], für die Neuroevolution angepasst. Damit das neuronale Netz das Spiel, oder allgemein formuliert, ein Problem lernen kann, ist es notwendig, drei Schnittstellen zwischen dem Problem und der Neuroevolution zu implementieren:

1. eine Inputschnittstelle für die Problemwahrnehmung
2. Interaktionsmöglichkeiten für die Problembewältigung
3. eine Evaluationsfunktion für die Bewertung der Problembewältigung

Ein Vorteil der Neuroevolution bzw. bestärkendem Lernen besteht darin, dass für die Problembewältigung kein vollständiges Verständnis des Anwenders für das Problem notwendig ist. Deshalb wird die Implementierung der Schnittstellen möglichst simpel gehalten, um nur die nötigen Aspekte des Problems abzudecken und um der Neuroevolution bei der Problembewältigung möglichst geringe Unterstützung zu bieten. Ziel des Prototyps ist es, den Lernerfolg und Lernprozess zu untersuchen und zu veranschaulichen. Das Spiel erfolgreich zu beenden, indem die Schlange das gesamte Spielfeld abdeckt, ist nicht das Ziel.

3.1. Das Spiel ‚Snake'

Snake ist ein simples zweidimensionales Spiel, in dem man die namensgebende Schlange steuert. Das Spielfeld ist ein Raster, indem sich die Schlange ausschließlich geradeaus oder mittels rechtwinkliger Abbiegevorgänge bewegen kann. Wenn der Schlangenkopf den Spielfeldrand oder eines ihrer eigenen Schlangensegmente berührt, ist das Spiel beendet. Ziel des Spiels ist es, möglichst viele Früchte, welche zufällig in das Spielfeld platziert werden, einzusammeln. Die Schlange bleibt dabei permanent in Bewegung, mit jedem neuen Bild bewegt sie sich um ein Feld in dem Raster. Mit jeder eingesammelten Frucht wächst die Schlange. Das bedeutet, ihre Länge wird um eine festgelegte Anzahl von Segmenten erhöht, was wiederum den Schwierigkeitsgrad erhöht, da die Anzahl an Hindernissen im Spielfeld wächst. Über die Pfeiltasten können Richtungswechsel befohlen werden, allerdings bewirken nur zwei der vier Eingabemöglichkeiten eine Veränderung. Ausschließlich die Pfeiltasten, die orthogonal zur Ausgangsbewegungsrichtung sind, bezwecken eine Veränderung der Bewegungsrichtung. Je nach Implementierung und Schwierigkeitsgrad können Bildrate, Feldgröße und Wachstum je Frucht variieren. Snake besteht aus verschiedenen Problemklassen mit einer Vielzahl unterschiedlicher Probleminstanzen je Problemklasse und ist somit ein ideales Beispiel für die komplexe Entscheidungsketten.

3.2. Aufbau des Prototyps

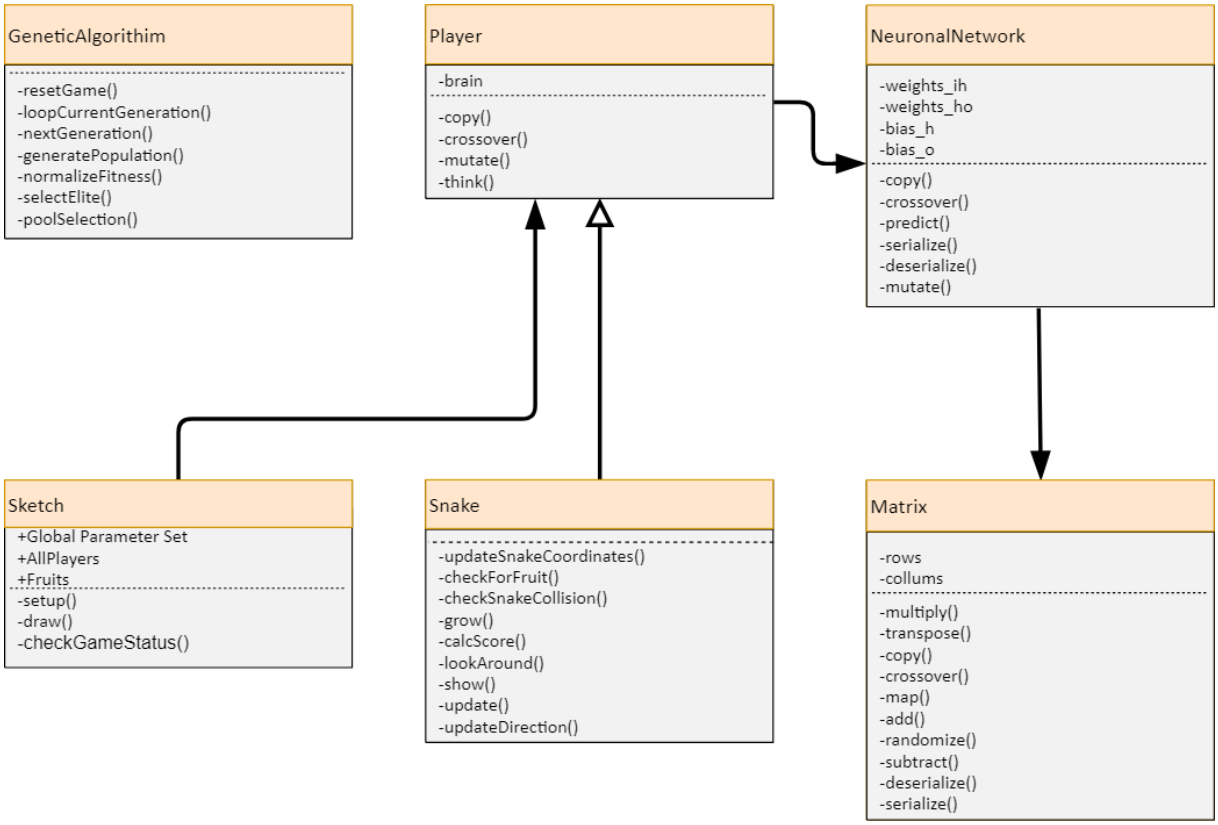

Abbildung 6: Funktioneller Aufbau des Prototyps

Quelle: eigene Darstellung

Abbildung 6 zeigt eine vereinfachte, schematische Übersicht des Projekts, in Anlehnung an ein UML-Klassendiagramm. Zwar ist es möglich, in JavaScript objektorientiert zu programmieren, allerdings ist JavaScript in erster Linie eine funktionale Skriptsprache, weswegen die Abbildung von der herkömmlichen UML-Notation abweicht. Für die bessere Übersicht wurde größtenteils auf Klassenattribute bzw. Variablen verzichtet.
Das Diagramm besteht aus den, für die Neuroevolution relevanten Klassen bzw. aus JavaScript-Dateien. Darüber hinaus sind die Dateien ChartTools.js, Era.js und IOFunctions.js ebenfalls Teil des Projekts, allerdings enthalten diese ausschließlich Funktionen für Dokumentationszwecke.
Das Array *AllPlayers* entspricht der Population der aktuellen Generation. Jedes Objekt des Arrays ist eine Instanz der Klasse Player, welche wiederum von Snake erbt. Die Klasse Snake enthält alles, was zur eigentlichen Spielmechanik gehört. Um die Neuroevolution möglichst unabhängig vom Spiel zu implementieren, erbt die Klasse Player von der Spielfigur, in diesem Fall von der Klasse Snake. Grundsätzlich besteht die Trennung zwischen Spiellogik in der Spielfigurenklasse und der Individuumlogik in der Playerklasse. Die Playerklasse besitzt ein Attribut *brain,* welches eine Instanz der *NeuronalNetwork* Klasse ist. Wie bereits geschrieben, besteht die Implementierung des neuronalen Netzes aus einer Menge von Matrizen, daher sind die Attribute *weights_ih*, *weights_ho*, *bias_h* und *bias_o* jeweils eine Instanz der *Matrix* Klasse. Die Population wird zum Ende der

Evaluationsphase mithilfe der Funktionen des *GeneticAlgorithms* entsprechend des Evolutionsalgorithmus manipuliert. Um während der Evaluationsphase das Verhalten der aktuellen Generation beobachten zu können und um alle Individuen einer Generation die gleiche Aufgabe, genauer gesagt die gleiche Probleminstanz bewältigen zu lassen, wurde ein globales Array für die Früchte implementiert. Aufgrund dessen existiert eine Abweichung von der herkömmlichen Spielmechanik. Normaler Weise ist es nicht möglich, dass eine neue Frucht an einer Position erscheint, an der sich gleichzeitig ein Schlangensegment befindet. Dadurch wird das Spiel grundsätzlich schwerer, allerdings wird diese Erschwernis erst bei zunehmender Schlangenlänge wahrnehmbar.

3.3. Funktionsweise des Prototyps

Die *Sketch*-Datei ist der Kern des Programms. Sie enthält die *Setup*- und *Draw*-Funktionen von Processing und somit auch die ‚Game Loop' genannte Ereignisschleife des eigentlichen Spiels. Die *Setup* Funktion wird, neben der Initialisierung des eigentlichen Spiels und des Processing-Sketches auch für die Initialisierung von einigen Variablen, der Population, der Diagramme und der Dokumentationsklasse verwendet. Die *Draw*-Methode bzw. der Game Loop besteht aus drei grundlegenden Schritten:

1. die *Think*-Methode eines jeden ‚lebenden' Individuums wird aufgerufen
2. die *Update*-Methode der ‚lebenden' Individuen wird aufgerufen
3. die Methode *checkGameStatus* wird aufgerufen
    - optional: jedes lebende Individuum wird in dem Sketch dargestellt

Die *Think* Methode repräsentiert die kognitiven Fähigkeiten der Schlange. Hier wird der Inputvektor, bestehend aus der Wahrnehmung der Schlange bzw. ihrem Sichtfeld von dem neuronalen Netz interpretiert und eine neue Richtung wird festgelegt.
Die *Update*-Methode ist zu vergleichen mit den motorischen Fähigkeiten der Schlange. Zum einen wird das Kopf-Segment der Schlange in die, durch die *Think*-Methode festgelegte, neue Richtung bewegt und anschließend jedes nachfolgende Segment an die Position des vorherigen Segments bewegt. Zum anderen werden hier die verschiedenen Kollisionen geprüft und gegebenenfalls das Wachsen der Schlange bzw. das Hinzufügen neuer Segmente durch vorheriges Erreichen einer Frucht ausgelöst. Wenn die Schlange die Spielfeldränder bzw. die Ränder des Processing-Sketches überschreitet, ist dies die Todesursache ‚*wall*'. Wenn der Schlangenkopf mit einem ihrer eigenen Schlangensegmente kollidiert, verendet die Schlange durch die Todesursache ‚*self*'. Die dritte und letzte Todesursache ist das Verhungern der Schlange, welche eintritt, wenn die Schlange zu viele Bewegungen hintereinander getätigt hat, ohne eine Frucht aufzusammeln und heißt entsprechend ‚*starved*'. Diese Mechanik existiert in der Form nicht in den meisten Versionen von Snake und wurde als Abbruchszenario für endlos rotierende Schlangen implementiert. Dieser Eingriff in die Spielmechanik ist in dieser oder

ähnlicher Form notwendig, da aufgrund des zufälligen Aspekts des Evolutionsalgorithmus solches Verhalten nie auszuschließen ist. Des Weiteren wurde eine Einschränkung auf die Fruchtpositionen entfernt, da diese für das Spiel eher untypisch ist. Das Spiel an sich ist dabei die Evaluationsphase des Evolutionsalgorithmus. Die Individuen erreichen einen Score bzw. Punktestand während des Spiels, welcher später für die Berechnung der Fitness verwendet wird. Im einfachsten Fall genügt es, den Punktestand der Zahl der eingesammelten Früchte gleichzusetzen. Ein Präzisieren des Punktestands zur Differenzierung von besserem und schlechterem Verhalten bei gleicher Anzahl eingesammelter Früchte ist von Vorteil, wenn die Leistung der Generation nahezu homogen ist, da so ein relativ gutes Verhalten bei gleicher Anzahl von Früchten dennoch eine größere Chance zur Selektion hat.
Zuletzt wird in der *Draw* Methode die Methode *checkGameStatus* aufgerufen. Diese Methode überprüft den Status der aktuellen Generation. Es wird geprüft, wie viele Individuen der Population aktiv bzw. lebendig sind und leitet, sofern alle Individuen an einer der 3 Todesursachen verendet sind, die nächste Generation ein, indem die Methode *nextGeneration* der Evolutionsalgorithmus-Funktionen aufgerufen wird. Beim Erstellen einer neuen Generation wird zunächst die Fitness, basierend auf dem Punktestand des jeweiligen Individuums in Abhängigkeit der Gesamtfitness der Generation berechnet:

$$f(P_i) = \frac{s(P_i)^2}{\sum_i^n s(P_i)^2}$$

Dabei ist P_i der Player (das Individuum), $s(P_i)$ ist der Score bzw. der Punktestand des Individuums, n ist die Kardinalität der Population und $f(P_i)$ ist die zu berechnende Fitness. Anschließend wird die Elite der Population unverändert in die kommende Generation vorgetragen. Die Differenz der Elite und der Populationsgröße wird dann mittels Selektion, Rekombination und ggf. Mutation erzeugt. Für die Selektion wird das Roulette-Rad-Verfahren verwendet.

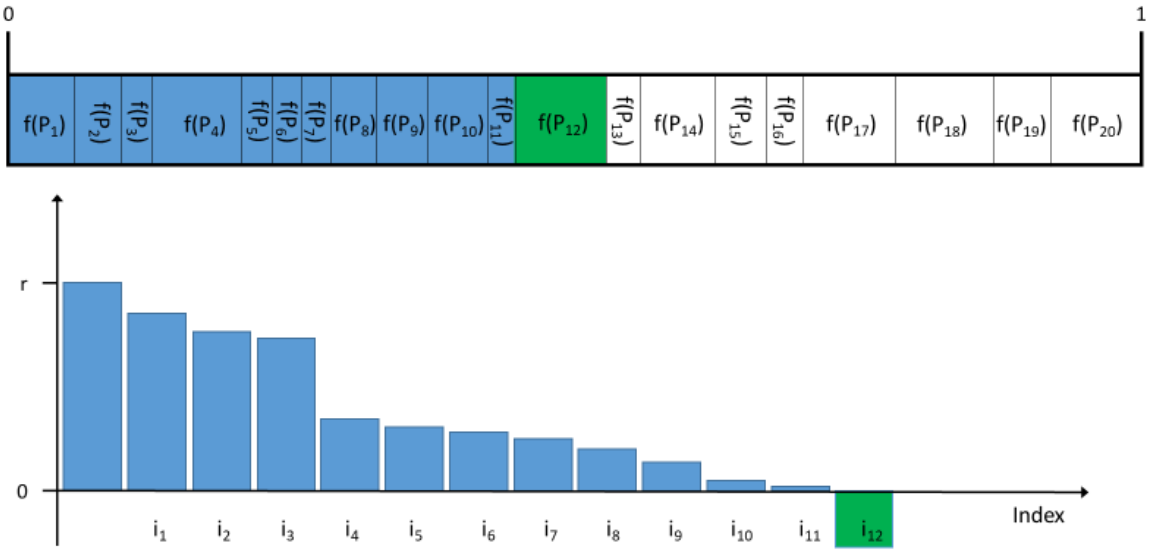

Abbildung 7: Roulette-Selektion

Quelle: eigene Darstellung

Abbildung 7 ist die Visualisierung des verwendeten Roulette-Rad-Verfahrens einer beispielhaften Population mit 20 Individuen. Die Summe der Fitness aller Individuen der Population entspricht eins. Für die Selektion wird ein zufälliger, gleichverteilter Wert r zwischen null und eins festgelegt. Dieser Wert r wird iterativ um die Fitness des Individuums i_x verringert. Sobald der Wert von r kleiner 0 ist, wird das Individuum der aktuellen Iteration selektiert. So wird in der Abbildung P_{12} nach 12 Iterationen selektiert. [20] Individuen mit verhältnismäßig größerer Fitness haben eine entsprechend größere Chance, als Elternteil eines Individuums der kommenden Generation selektiert zu werden. Für jedes zu erzeugende Individuum werden zwei Eltern selektiert, welche anschließend einen Nachkommen mithilfe der *Crossover* Funktion des neuronalen Netzes und der Matrizen erzeugen. Dabei wird für jede Matrize der neuronalen Netze ein zufälliger Punkt gewählt. Anschließend wird mittels einer verschachtelten Iteration durch die Matrizen iteriert. Bis zu diesem Punkt erbt der Nachkomme von Elternteil A und nach diesem Punkt erbt der Nachkomme von Elternteil B (*Siehe Abbildung 4)*.

Dieser Prozess repräsentiert die Rekombination des Evolutionsalgorithmus. Anschließend wird die *mutate* Funktion jedes Individuums aufgerufen. Dabei wird für jedes funktionale Attribut des neuronalen Netzes, sprich für jede Wichtung und für jeden Bias-Wert, eine zufällige, gleichverteilte Zahl zwischen 0 und 1 bestimmt. Sofern dieser Wert kleiner ist als die Mutationsrate wird dieses Attribut mutiert. Dafür wird ein weiterer zufälliger, allerdings um 0 normalverteilter Wert bestimmt, welcher als Versatz auf das Attribut addiert wird. Sobald mittels Selektion, Rekombination und Mutation die folgende Population mit Nachfahren gefüllt ist, beginnt die nächste Generation des Evolutionsalgorithmus mit der Evaluationsphase.

3.4. Dokumentation

Um die Evolutionserfolge vergleichen zu können, werden nach der Evaluationsphase deskriptive stochastische Werte der Population in die *Era* (zu Deutsch: Ära) Klasse geschrieben. Je Generation werden für die vier Attribute Punktestand, Anzahl der Früchte, Anzahl der Schritte und Anzahl der Wenden jeweils der höchste und geringste Wert, der Durchschnitt, der Median, die Standardabweichung, die Varianz (redundant) sowie das erste und dritte Quantil festgehalten (insgesamt 8 Werte je Attribut). Darüber hinaus wird die Anzahl der Tode durch die drei verschiedenen Todesursachen je Generation festgehalten. Somit werden je Generation 35 Attribute (4*8+3) gespeichert. Für Monitoringzwecke kann eine Teilmenge dieser Attribute während der Evolution als Liniendiagramm dargestellt werden. Diese Attribute können als CSV-Datei exportiert werden, sodass Evolutionen nach einer Ära miteinander verglichen werden können. Die Größe einer Ära wurde auf 500 Generationen festgelegt, da ein Anstieg des Erfolgs oftmals erst nach mehreren hundert Generationen zu beobachten ist (siehe 5.2.3. Stufenweiser Fortschritt insbesondere Abbildung 15).

4. Experimentaufbau

4.1. Vergleichbarkeit der Testläufe

Wie bereits beschrieben, basiert die Diversifizierung durch den Evolutionsalgorithmus auf Evolutionsoperationen (Selektion, Rekombination, Mutation), welche wiederum auf zufälligen Zahlen basieren. Des Weiteren ist die Positionsbestimmung der Früchte zufällig. Jede Generation hat eine andere Problemsequenz, sodass der Schwierigkeitsgrad von Generation zu Generation variiert. Aufgrund dieser Zufälligkeit ist eine Vergleichbarkeit von zwei Testläufen mit unterschiedlichen Parametern nicht gegeben, da ein besseres Abschneiden nicht zwingend durch die veränderten Parameter bedingt ist. Um dennoch eine Vergleichbarkeit gewährleisten zu können, wurden vier Ansätze ausgearbeitet.

1. Für den RNG einen Seed zu setzen und für alle zufällig generierten Zahlen den RNG mit Seed verwenden.
2. Ausschließlich für die Position der Früchte einen RNG mit Seed festlegen.
3. Für jede Parameterausprägung mehrere Testläufe berechnen und von diesen den Durchschnitt als Vergleichsbasis wählen.
4. Ein Hybrid des zweiten- und dritten Ansatzes. Für die Testläufe werden verschiedene Seeds ausschließlich für die Positionen der Früchte festgelegt.

Der erste Ansatz bietet keine Vergleichbarkeit, da durch das Verändern der Parameter die Anzahl von zufällig generierten Zahlen je Generation variiert. Dadurch ist keine Vergleichbarkeit gewährleistet, da die Positionen der Früchte und deren Abfolge variiert.
Der zweite Ansatz würde eine Vergleichbarkeit hinsichtlich der neuronalen Netze und des Evolutionsprozesses bieten, allerdings wird dann nicht die Problemklasse, ‚Schlange sucht Frucht', sondern eine explizite Instanz dieses Problems trainiert.

Es besteht die Gefahr des Overfittings. Individuen, die in solchen Szenarien trainiert wurden, sind nicht zwingend in vergleichbaren, anderen Situationen ähnlich performant. Lediglich die Erfolgsvarianz aufgrund von besseren oder schlechteren Fruchtpositionen kann dadurch ausgeschlossen werden.
Der dritte Ansatz bietet grundsätzlich die beste Methodik, weil die Ergebnisse, gemäß dem Gesetz der großen Zahlen, reiner werden, je mehr Testläufe pro Parameterausprägung durchgeführt werden. Je nach Lernerfolg der Ära kann eine einzige Evaluationsphase mehrere Minuten dauern, da die Individuen zum Teil mehr als 7000 Frames oder Schritte in herkömmlichen Testläufen überleben. Bei 30 Bildern pro Sekunde beträgt die Dauer der Evaluationsphase allein mehr als 3 ½ Minuten. Um die Evaluationsphase zu beschleunigen, wurden zwei Maßnahmen implementiert. Zum einen ein Speed-Slider, welcher die Anzahl an berechneten Schritten je Iteration der Draw-Methode erhöht. Zum anderen die Möglichkeit, auf die Darstellung der Evaluationsphase zu verzichten. Dadurch lässt sich die Evaluation zwar beschleunigen, allerdings entspricht die Beschleunigung nicht dem Faktor des Speed-Sliders, da die Berechnungen durch das Ressourcenmanagement des Browsers eingeschränkt werden. In der Praxis bedeutet das, dass mit 50 Berechnungen je Iteration der Draw-Methode die Evaluation von 500 Generationen ca. fünf bis sechs Stunden dauert. Um die Evaluationsphase weiter zu verkürzen, wurden mehrere Browserinstanzen parallel verwendet.
Der vierte Ansatz bietet die Möglichkeit, die Evolutionseinflüsse an wenigen expliziten Ausprägungen der Problemklasse vergleichen zu können. Hier besteht allerdings wieder das Problem, dass Overfitting auf die expliziten Problemfälle nicht ausgeschlossen werden kann. Anhand dieser Problemfälle kann man jedoch den Parametereinfluss auf den Lernerfolg der Problemausprägungen untersuchen und ausschließen, dass ein Testlauf bzw. eine Generation aufgrund von günstigeren Fruchtpositionen besser abschneidet.

4.2. Versuchsdurchführung

Um eine Vergleichbarkeit der Ergebnisse trotz des Einflusses durch zufällig generierte Zahlen gewährleisten zu können, wurde der oben beschriebene dritte Ansatz verwendet. Allerdings ist es aufgrund des zeitlichen Rahmens der Bachelorarbeit nicht möglich, hinreichend viele Testläufe je Attributausprägung durchzuführen. Als wissenschaftlicher Kompromiss wurden daher je Attributausprägung zehn Durchführungen vorgenommen, um den Parametereinfluss anhand des arithmetischen Mittels besagter zehn Durchführungen zu untersuchen.
Neuroevolution bietet viele Einstellungsmöglichkeiten, da jede Komponente der Neuroevolution angepasst werden kann. Denkbar sind Anpassungen der folgenden Bestandteile:

- Evolutionsalgorithmus
- Neuronale Netze

- die oben beschriebenen Schnittstellen zum Problem
- das Problem selbst

Im Folgenden werden die Attributeinflüsse von vier Parametern der Neuroevolution untersucht. Dabei gilt zu beachten, dass die Einflüsse auf den Lernerfolg von der Problemsequenz abhängen. Ein positiver Einfluss auf das Erlernen der Problemsequenz ‚Snake' hat nicht zwingend einen allgemein positiven Einfluss auf Neuroevolution. Untersucht wurden die folgenden Attributausprägungen:

- Größe des Hiddenlayers (6, 8, 10, 12, 14, 16, 18, 20, 22)
- Größe der Population (100, 200, 400, 600, 800, 1000, 1200, 1400, 1600, 1800, 2000)
- Größe des Elitismus (0, 5, 10, 20, 30, 40, 80, 160)
- Mutationsrate (0.02, 0.04, 0.06, 0.08, 0.1, 0.12, 0.14, 0,16, 0.18)

Da für die Auswertungen 10 Iterationen (oder Testläufe bzw. Ären) je Attributausprägungen anzufertigen sind, wurde eine leichte Abwandlung des Prototyps implementiert, sodass nach einer Ära automatisch die in 3.4. beschriebene CSV-Datei gespeichert wird und eine neue Iteration gestartet wird.

4.3. Ergebnisse und Aggregation

Für die Aggregation und Auswertung der Ergebnisse wurden die zwei R-Skripte *AggregateCSVs.R* und *CompareAndPlot.R* angefertigt. Zuerst werden die zehn, in 3.4. beschriebenen, CSV-Dateien je Attributausprägung mithilfe des ersten Skripts aggregiert. Hierfür wurden Ordnerstrukturen zur Begünstigung des Prozesses angelegt. Je zu vergleichendem Attribut wurde ein Ordner angelegt, welcher wiederum je Attributausprägung einen Unterordner besitzt, in dem sich die zehn CSV-Dateien befinden. Das Skript iteriert über die Unterordner des Attributordners und erstellt je Attributausprägung eine aggregierte CSV-Datei (z.B. AVG-ES10.CSV). Dabei wird für jede Spalte der CSV-Datei je Generation ein Durchschnittswert berechnet, welcher in der aggregierten CSV als repräsentativer Wert für die zehn Instanzen der Attributausprägung dient. Darüber hinaus wird für die Anzahl der Früchte die Standardabweichung über die zehn Instanzen berechnet. Für die Auswertung und den Vergleich der Ergebnisse wird die Anzahl der eingesammelten Früchte verwendet, da dieser Wert, unabhängig von der Berechnung des Punktestands, den Spielfortschritt am besten beschreibt.

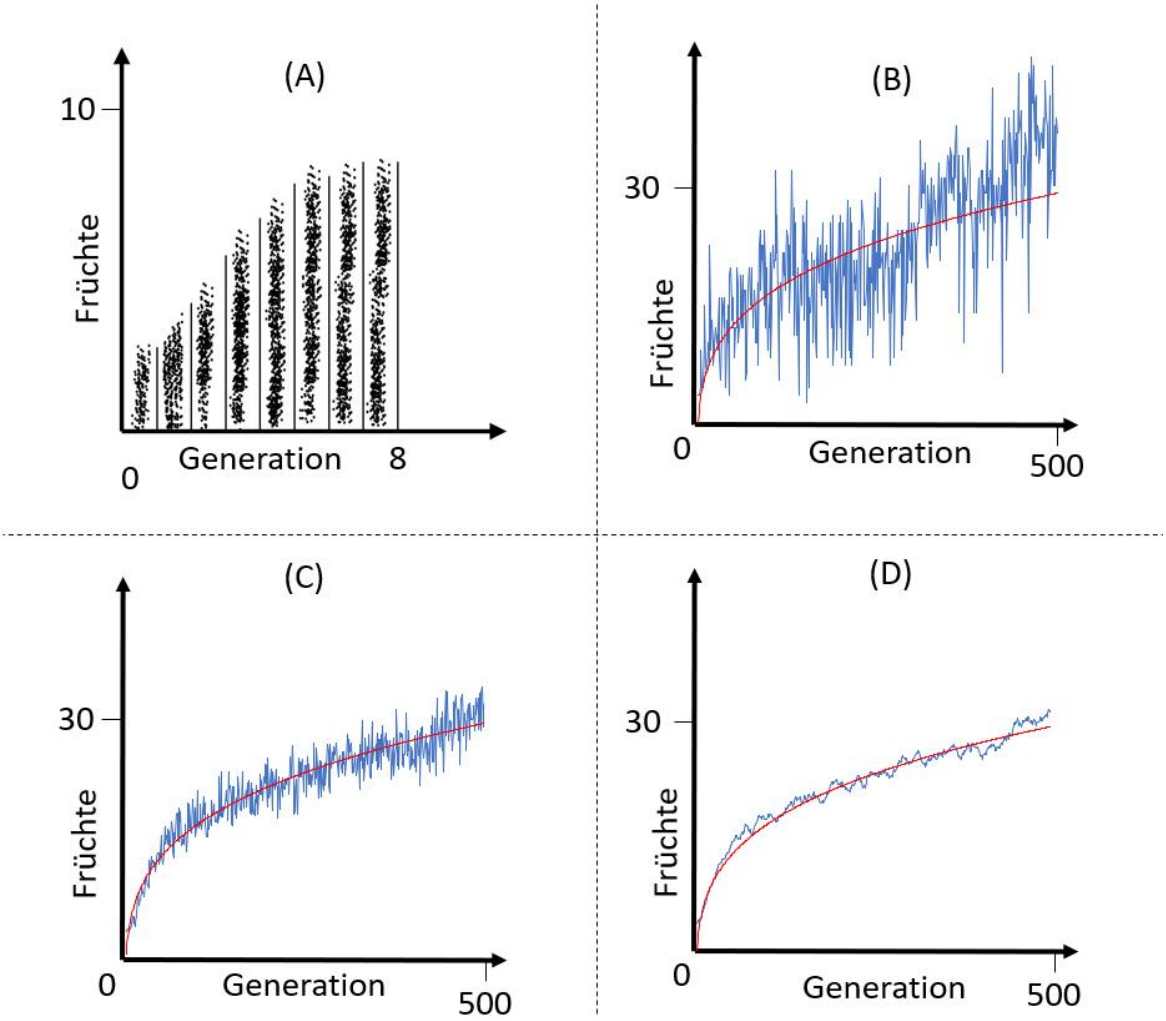

Abbildung 8: Datenaggregation

A – Visualisierung der Leistung aller Individuen je Generation über 8 Generationen als Punktwolke

B – Visualisierung einer Datenreihe der in 3.4. erstellten CSV-Dateien

C – Visualisierung der aggregierten CSV-Dateien

D – gleitender Durchschnitt über 10 Werte der aggregierten CSV-Dateien

Quelle: eigene Darstellung

Abbildung 8 zeigt die Aggregation der Daten als schematische Darstellung. Während in Abbildung (A) die erzielten Früchte alle Individuen dargestellt sind, werden in den weiteren 3 Abbildungen die meisten eingesammelten Früchte je Generation dargestellt. Die Abbildung (B), (C) und (D) beruhen auf tatsächlichen Daten und zeigen die Entwicklung über 500 Generationen. Zusätzlich wurden in diesen 3 Abbildungen die Wurzelfunktion der nichtlinearen Regression auf den Daten der aggregierten CSV-Datei dargestellt. Die erste Aggregation erfolgt bereits in JavaScript beim Erstellen der CSV-Dateien. Statt je Generation die erzielten Früchte jedes einzelnen Individuums (Abbildung 8 (A) Darstellung als Punktwolke) zu speichern, werden die in 3.4. beschriebenen stochastischen Größen (höchster und geringster Wert, erstes, zweites (Median) und drittes Quartil, Durchschnitt und

Standardabweichung) gesichert (Abbildung 8 (B)). Die Ausreißer in dieser Graphik kommen durch verhältnismäßig schwere Fruchtpositionen und deren Kombinationen zustande. Beispielsweise eine direkt am Rand befindliche Frucht kann ein verhältnismäßig schlechtes Abschneiden einer, mitunter auch späteren Generation verursachen. Beim Ansteuern einer solchen Frucht konkurrieren zwei Interessen: zum einen das Erreichen der Frucht und zum anderen das Vermeiden einer Kollision mit der Wand. Durch das Aggregieren über die zehn Instanzen werden solche Ausreißer und positive Ausreißer gleichermaßen reduziert. Es ist ein, für die Neuroevolution, charakteristischer Zuwachs der Früchte in Abhängigkeit von der Generation zu erkennen. Mithilfe der Funktion nlsLM (nonlinear Least Squares Levenberg-Marquardt) [21] wurde eine nichtlineare Regression durchgeführt. Die Funktion verwendet die Methode der kleinsten Quadrate, um ein Parameterset einer Funktion zu optimieren. Aufgrund der charakteristischen Form des Zuwachses wurden zwei nichtlineare Regressionen mit verschiedenen Funktionen durchgeführt. Eine mit einer Logarithmusfunktion der Form $f(x) = a * log(x) + c$ mit den Ausgangsparametern $a = 1$, $c = 0$ und eine mit einer Exponential- bzw. Wurzelfunktion der Form $f(x) = a * x^b + c$ mit den Ausgangsparametern $a = 1$, $b = \frac{1}{2}$, $c = 0$. In allen durchgeführten, nicht linearen Regressionen beschreibt die Wurzelfunktion das tatsächliche Wachstum präziser als die Logarithmusfunktion. Die Korrelation der Wurzelfunktion mit den tatsächlichen Werten war stets größer als die der Logarithmusfunktion. Auch die Residuenquadratsumme der Wurzelfunktion war stets geringer. Aufgrund dessen wird die nichtlineare Regression einer Logarithmusfunktion vernachlässigt. Um das Rauschen des Graphen weiter zu reduzieren und um eine bessere Vergleichbarkeit zu gewährleisten, wurde jeweils der gleitende Durchschnitt berechnet. Dabei gilt zu beachten, dass ein Verringern des Rauschens der Graphen durch ein Erhöhen der Iterationen ebenfalls möglich[1] ist und das beim Erhöhen der Schrittgröße s der gleitende Durchschnitt sich zunehmend der Wurzelfunktion angleicht. Dabei berechnet sich die Stelle i des gleitenden Durchschnitts wie folgt:

$$mvgAvg(x_i) = \frac{1}{s} * \sum_{k=i}^{s} x_k$$

Die neue Stelle i ist der Durchschnitt der s auf i folgenden ursprünglichen Werte. Beim Berechnen des gleitenden Durchschnitts wird daher die Kardinalität der Datenmenge um $s - 1$ verringert.

5. Auswertung

Je zu vergleichendem Attribut werden insgesamt fünf Graphen gezeigt, der gleitende Durchschnitt, die Wurzelfunktionen der nichtlinearen Funktion sowie der

[1] Siehe Anhang A

gleitende Durchschnitt der besten und schlechtesten Attributausprägung mit einfacher Standardabweichung über die zehn Iterationen. Dabei wurden für die gleitenden Durchschnitte und die Wurzelfunktionen jeweils zwei Graphen erstellt, um eine bessere Übersicht zu gewährleisten. Darüber hinaus wird zu jedem untersuchten Attribut eine Tabelle gezeigt, in der die Wurzelfunktionen sowie beste Werte und Standardabweichung aufgelistet werden.

Attributausprägung	Wurzelfunktion der Form: $f(x) = a * x^b + c$						bester Wert			SD
	a	b	c	x=500	Kor	SQR	all	Avg	MvgAvg	
100	0,06	0,79	0,14	7,92	0,909	477,2	35	10,1	8,4	7,4

Abbildung 9: Tabellenkopfzeile

Quelle: eigene Darstellung

Abbildung 9 zeigt die Tabellenkopfzeile der Auswertungstabellen. Zur besseren Übersicht wurden Abkürzungen für die Spaltenbezeichnungen gewählt. Die Spalten *a*, *b* und *c* enthalten die entsprechenden Parameter der Wurzelfunktion, die Spalte *x=500* enthält den Funktionswert der Wurzelfunktion nach 500 Generationen. *Kor* ist kurz für Korrelation und gibt die Korrelation zwischen den Funktionswerten der Wurzelfunktion und den durchschnittlichen Werten der meisten eingesammelten Früchte über die zehn Iterationen an. Der Wert wurde mithilfe der R-Funktion *cor()* berechnet. *SQR* ist die Abkürzung für Summe der Quadrate der Residuen bzw. die Residuenquadratsumme, ausgelesen aus dem nichtlinearen Modell. Die Spalte *bester Wert all* enthält die meisten erzielten Früchte der zehn Iterationen unverändert. Die Spalte *bester Wert Avg* enthält den besten Wert des errechneten Durchschnitts über die zehn Iterationen. In der Spalte *bester Wert MvgAvg* ist der höchste Wert des gleitenden Durchschnitts über zehn Generationen der Aggregation aufgeführt. Die letzte Spalte *SD* enthält den Durchschnitt der Standardabweichungen der letzten 50 Generationen, wobei die Standardabweichung über der zehn Iterationen berechnet wurde.

5.1. Auswertung der untersuchten Parameter

Für jeden untersuchten Parameter wird zunächst eine Prognose bzw. ein erwarteter Einfluss geschildert, anschließend die Graphen und Tabelle präsentiert und abschließend eine Evaluation des tatsächlichen Einflusses durchgeführt. Die Parameter wurden nacheinander untersucht und nach dem Abschluss eines Parameters wurde die vermeintlich beste Ausprägung für die kommenden Versuche verwendet. Die Ausgangsparameter für die Versuche sind wie folgt:

- Populationsgröße: 1000
- Neuronen im Hiddenlayer: 14
- Größe des Elitismus: 10
- Mutationsrate: 0,1

5.1.1. Einfluss der Größe des Hiddenlayers

Grundsätzlich ist ein trainiertes neuronales Netz mit einer Funktion zu vergleichen, da es gleich der Funktion Inputwerte auf Outputwerte abbildet. Aufgrund des Aufbaus und der Funktionsweise neuronaler Netze und aufgrund der Verwendung von tiefen neuronalen Netzen für komplexe Aufgaben liegt die Vermutung nahe, dass größere neuronale Netze komplexere Funktionen abbilden können. Außer gegebenen Falls redundante Netzstrukturen und zusätzlich benötigter Rechenleistung sollten größere neuronale Netze keine weiten Nachteile mit sich bringen.

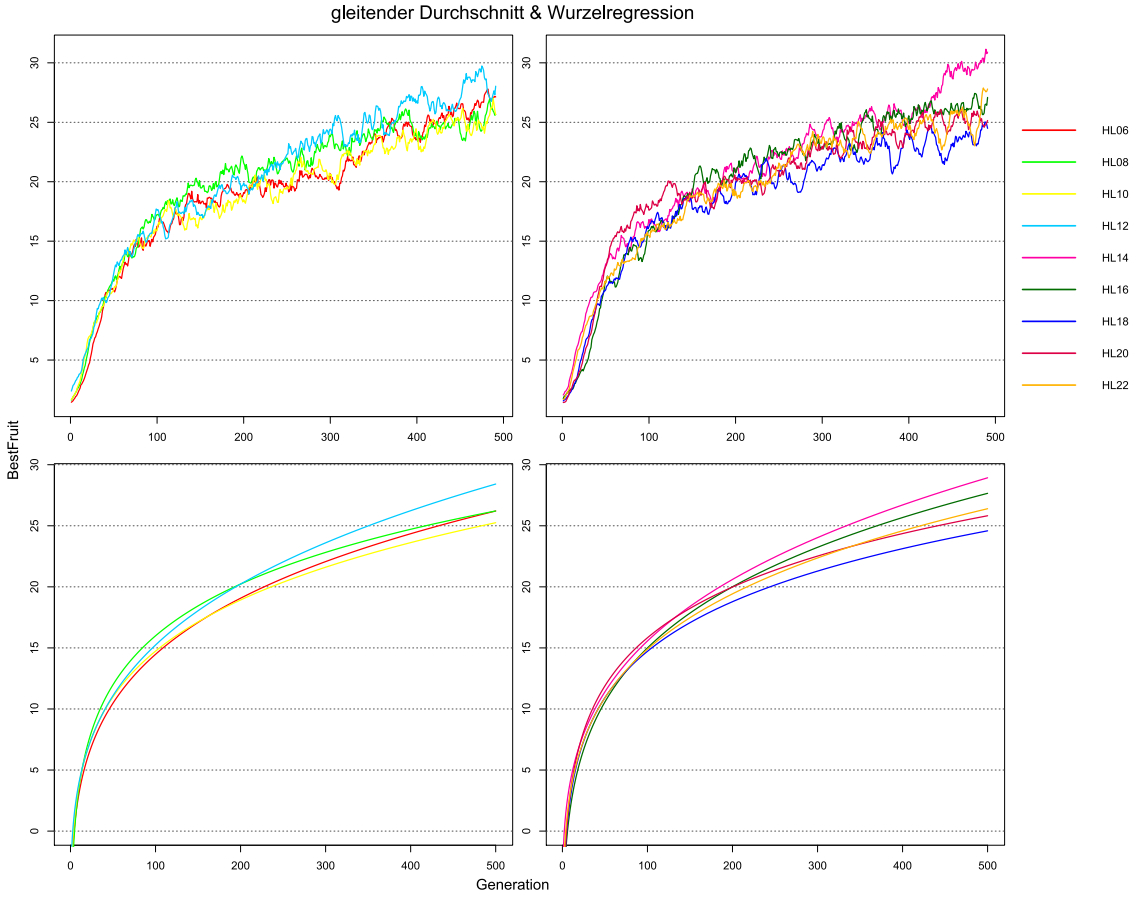

Abbildung 10: Variation der Größe des Hiddenlayers

Quelle: eigene Darstellung

Auf den ersten Blick lassen sich nur marginale Unterschiede feststellen, die einzigen beiden Attributausprägungen die sich, wenn auch nur geringfügig, von den restlichen Ausprägungen unterscheiden sind HL12 und HL14. Diese Beobachtung stützt die obige Vermutung in keiner Weise. In der Grafik ist darüber hinaus kein bedeutender Unterschied zwischen der Leistung der drei größten Netze (HL18 – HL22) und den drei kleinsten Netzen (HL06 – HL10) festzustellen. Lokale Spitzen sind dabei zu vernachlässigen, da sie insbesondere bei der Anwendung von Elitismus auf die Variation der Problemsequenzen zurückzuführen sind. Anhand der Graphen lässt sich nicht eindeutig sagen, welche Attributausprägung den besten Lernerfolg verspricht. Aufgrund der Aufwärtstendenz von HL14 würde sich diese Attributgröße eignen.

Tabelle 1: Einfluss der Größe des Hiddenlayers

Attributausprägung	Wurzelfunktion der Form: $f(x) = a * x^b + c$						bester Wert			SD
	a	b	c	x=500	Kor	SQR	all	Avg	MvgAvg	
6	12,18	0,20	-16,25	26,21	0,934	2588	72	32,0	27,8	9,8
8	49,68	0,08	-56,48	26,21	0,923	2817	63	32,4	27,0	9,3
10	16,19	0,16	-19,86	25,24	0,929	2391	60	29,9	27,1	8,7
12	7,14	0,27	-9,26	28,43	0,943	2553	75	33,1	29,7	12,0
14	6,73	0,28	-8,38	28,93	0,952	2117	58	33,8	31,2	9,9
16	16,48	0,18	-22,72	27,65	0,948	2393	69	30,8	27,4	9,1
18	37,62	0,10	-43,96	24,59	0,939	1972	60	28,8	25,0	8,3
20	54,44	0,08	-61,31	25,82	0,924	2693	52	29,4	26,1	8,6
22	13,77	0,19	-17,76	26,39	0,946	2021	64	34,5	27,9	9,4

Beim Betrachten der Tabelle bestätigt sich die Vermutung, dass ein neuronales Netz mit 14 Neuronen im Hiddenlayer die potenziell besten Ergebnisse hervorbringen könnte. Dafür spricht in erster Linie, dass die Attributausprägung 14 den größten Exponenten besitzt. Zusätzlich hat diese Ausprägung eine verhältnismäßig geringe Residuenquadratsumme und entsprechend hohe Korrelation. Dies bestärkt den Exponenten als ausschlaggebenden Indikator. Lediglich der beste unveränderte Wert aller Iterationen ist unerwartet schlecht für die vermeintlich beste Attributausprägung. Diese kann wiederum durch die Variation der Problemsequenz bedingt sein. Aufgrund dessen wurde diese Ausprägung für die folgenden Experimente gewählt. Grundsätzlich lässt sich kein markantes Muster des Einflusses der Größe des Hiddenlayers erkennen. Eine Ursache könnte sein, dass selbst das kleinste neuronale Netz mit sechs Neuronen im Hiddenlayer genügend Wichtungen und Bias-Werte (in Summe 112) besitzt, um das Problem abzudecken.

5.1.2. Einfluss der Populationsgröße

Bei der Neuroevolution wird die Suchraumabdeckung über die Diversifizierung durch Mutation und Rekombination realisiert. Wenn Individuen dem Optimum näher sind als der Rest ihrer Generation, müssen sie zusätzlich selektiert werden, um den Fortschritt in die nächste Generation weitertragen zu können. Die drei Evolutionsoperationen (Mutation, Rekombination, Selektion) sind mittels zufälliger Zahlen realisiert. Deshalb liegt die Vermutung nahe, dass eine größere Population bessere Chancen, hat dem Optimum näher zu kommen bzw. ggf. das Optimum zu erreichen als kleinere Populationen.

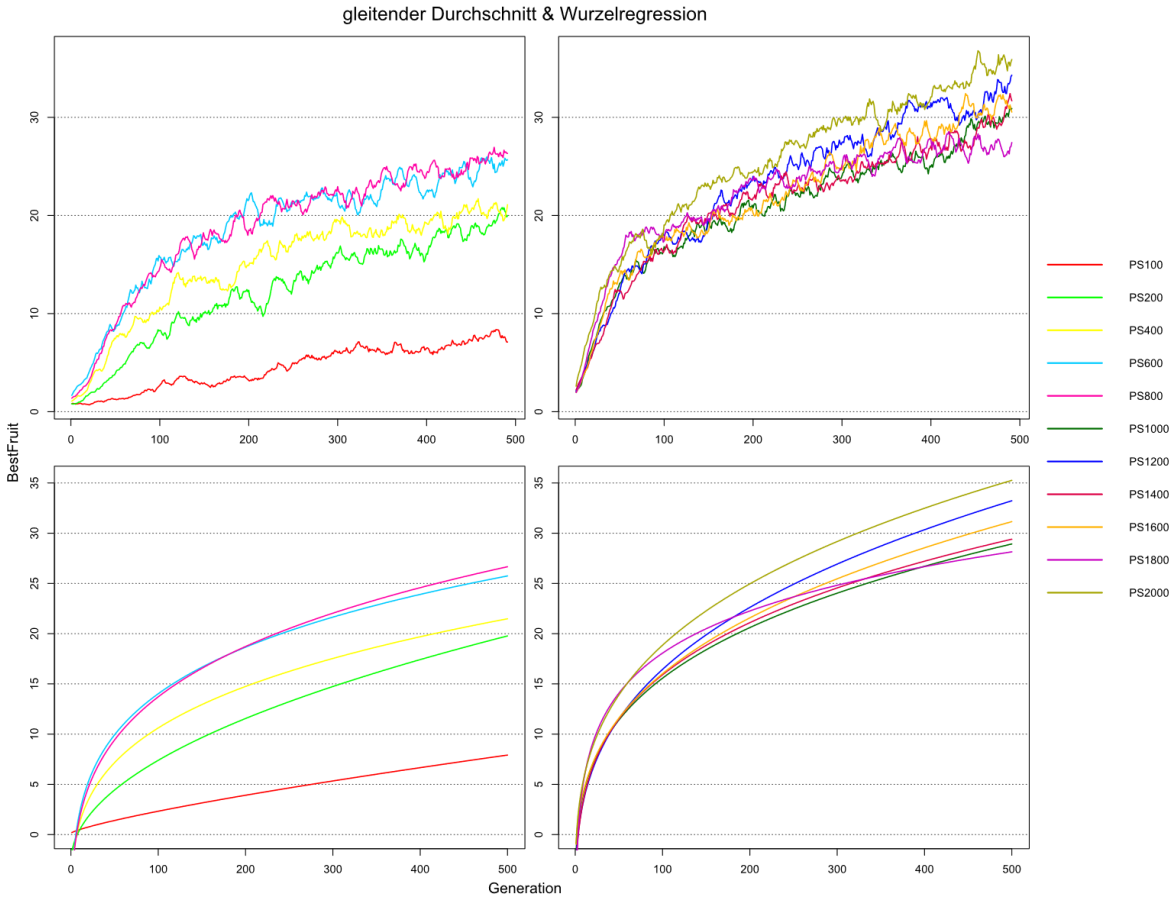

Abbildung 11: Variation der Populationsgröße

Quelle: eigene Darstellung

Die Graphen der linken Seite zeigen die ersten 5 Attributausprägungen (100 – 800) und die Graphen der rechten Seite zeigen die restlichen 6 Attributausprägungen (1000 – 2000). Die Graphen scheinen die Vermutung zu bestätigen, dass eine größere Population eine bessere Entwicklung bedingt. Besonders die linke Seite stützt diese Vermutung, da die Graphen entsprechend der Populationsgröße aufsteigend geordnet sind. Auch die Graphen der rechten Seite stützen diese Vermutung, zumindest bedingt. Die Graphen vom geringsten zu höchstem Erfolg sind in der Reihenfolge (1800 - 1000 - 1400 - 1600 - 1200 - 2000) geordnet. Wobei zu beobachten ist, dass der Graph PS1800 zunächst ein vergleichbar steiles Wachstum wie der Graph PS2000 bestrebt und erst ab ca. Generation 100 abflacht. Ebenso hebt sich der Graph PS1200 bis zur Generation 150 nicht augenscheinlich von den Graphen 1000-1600 ab. Dies lässt vermuten, dass aufgrund der zufallsbedingten Streuung, besonders aufgrund des zufälligen Schwierigkeitsgrads der Problemsequenz (Fruchtpositionsabfolge) die tatsächliche Reihenfolge von der erwarteten Reihenfolge der Graphen abweicht.

Tabelle 2: Einfluss der Populationsgröße

Attributausprägung	Wurzelfunktion der Form: $f(x) = a * x^b + c$						bester Wert			SD
	a	b	c	x=500	Kor	SQR	all	Avg	MvgAvg	
100	0,06	0,79	0,14	7,92	0,909	477,2	35	10,1	8,4	7,4
200	1,04	0,49	-2,79	19,78	0,948	1556	52	23,2	20,8	10,9
400	6,37	0,26	-10,40	21,49	0,934	2046	44	25,0	21,7	10,2
600	15,05	0,18	-20,64	25,74	0,935	2635	65	31,3	26,0	9,9
800	10,63	0,22	-16,12	26,67	0,950	2249	61	32,3	27,0	10,0
1000	6,73	0,28	-8,38	28,93	0,952	2117	58	33,8	31,2	9,9
1200	5,48	0,32	-8,00	33,22	0,958	2758	71	39,7	34,3	12,4
1400	9,80	0,24	-13,23	29,42	0,953	2265	77	34,2	32,4	11,5
1600	4,82	0,33	-5,78	31,16	0,946	2917	70	36,3	32,4	12,9
1800	57,72	0,07	-62,75	28,15	0,932	2452	75	33,1	28,6	10,8
2000	6,57	0,30	-7,45	35,27	0,961	2471	79	41,7	36,8	14,5

Eine Prognose für die Generation 5000 ist ausgehend von den ersten 500 Generationen nicht möglich bzw. sehr unpräzise. Die geringste Populationsgröße hat den höchsten Wert des Exponenten der Wurzelfunktion, allerdings auch die geringste Korrelation. Das bedeutet, dass das Wachstum dieser Attributausprägung am ehesten mit dem einer linearen Funktion zu vergleichen ist. Würden diese Modelle den tatsächlichen Lernerfolg der Attributausprägungen beschreiben, würde das bedeuten, dass die kleinste Population früher oder später (in diesem Fall nach ca. 16000 Generationen) die größeren Populationen überholen würde. Zu beachten ist die geringe Abweichung des höchsten Werts des gleitenden Durchschnitts und des Wurzelfunktionswerts nach 500 Generationen. Dies ist ein Indiz dafür, dass sich der gleitende Durchschnitt bzw. das weitere Glätten des Graphen eine Angleichung an die Wurzelfunktion bewirkt. Wie auch in der Grafik ist hier bei der Wurzelfunktion und dem gleitenden Durchschnitt die Ordnung nach Populationsgröße mit Ausnahme der Populationsgrößen von 1200 und 1800 zu erkennen. Betrachtet man allerdings den höchsten unveränderten Wert über alle Iterationen je Attributausprägung, lässt sich keine klare Ordnung, sondern nur eine Tendenz erkennen. Dies ist auf die Variation der Problemsequenz zurückzuführen. Ausgehend von diesen Ergebnissen lässt sich die anfängliche Vermutung des positiven Einflusses einer größeren Population bestätigten. Zwar wurden die besten Ergebnisse mit einer Populationsgröße von 2000 erzielt, allerdings waren die Iterationen sehr zeitintensiv. Daher wurden für die weiteren Experimente eine Populationsgröße von 1200 Individuen verwendet.

5.1.3. Einfluss der Größe des Elitismus

Um mögliche Rückschritte durch potenziell negative Veränderung vorhergehender Topscorer aufgrund von Rekombination und Mutation zu verhindern, wurde Elitismus angewandt. Dabei wird eine festgelegte Menge der Besten der vorhergehenden Generation unverändert in die kommende Generation vorgetragen. Mit dem Erhöhen der Elitismusgröße ohne die Populationsgröße zu

verändern, vermindert man den Vorteil einer größeren Population, da die Diversifikation durch Elitismus reduziert wird. Da die Individuen, welche als Elite identifiziert wurden auch zusätzlich als Elternteil selektiert werden können, steigt die Chance über mehrere Generationen Nachkommen von (ehemaligen) Topscorern zu erzeugen. Elitismus grenzt somit den Suchraum um aktuelle beste Lösungen ein. Je größer der Elitismus desto stärker die Eingrenzung. Daher die Vermutung, dass Elitismus ab einer gewissen Größe den Lernerfolg einschränkt, aber gleichzeitig die Streuung der Leistung reduziert wird.

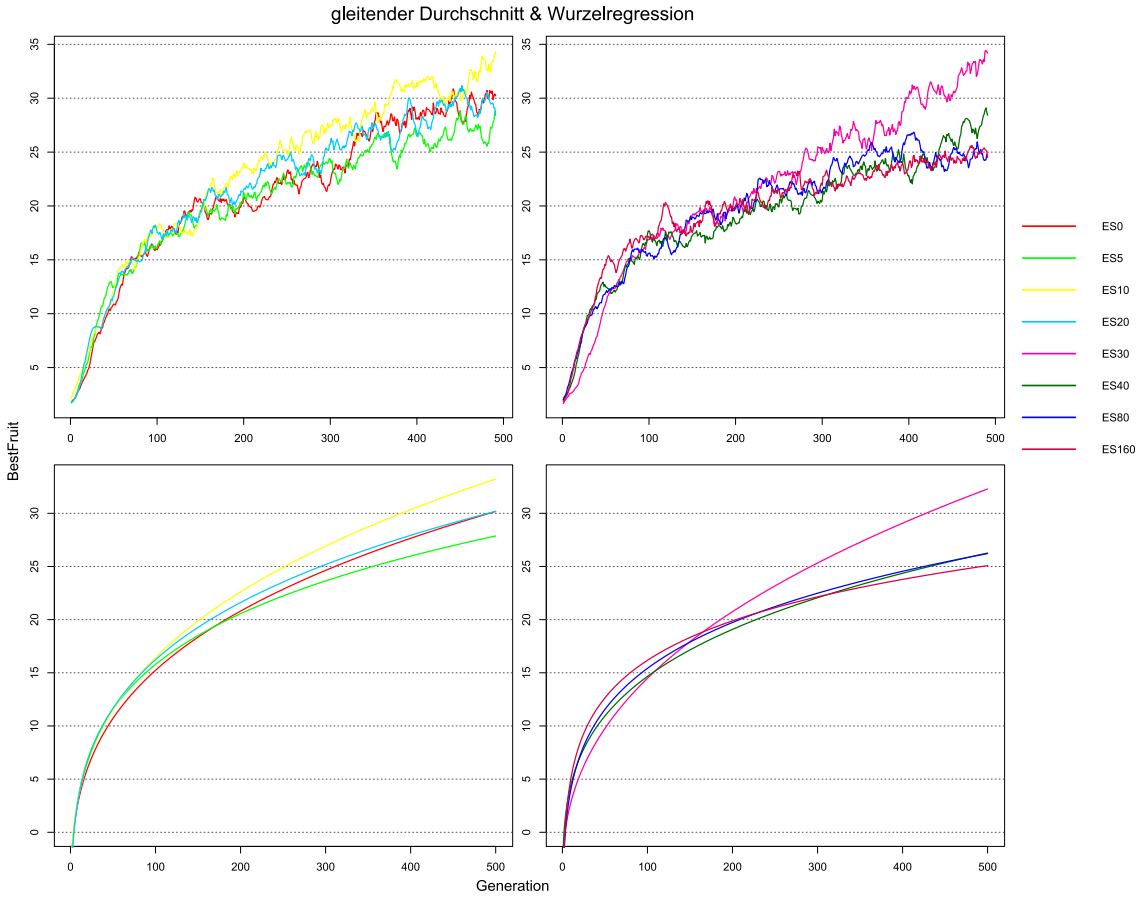

Abbildung 12: Variation der Elitismusgröße

Quelle: eigene Darstellung

Anhand der Graphen ist zu erkennen, dass keine großen Unterschiede zwischen den geringen Elitismusgrößen bestehen. Ein deutliches Abflachen der Graphen ist erst ab einer Elitismusgröße von 40 Eliten und aufwärts zuerkennen. Auffällig ist das bessere Abschneiden von 30 Eliten gegenüber den restlichen Elitismusgrößen, insbesondere den direkten Nachbarn (20 und 40 Eliten). Es wurde erwartet, dass die beste Leistung mittig einzuordnen ist. Zwei Spitzen mit einem verhältnismäßig schlechten Wert zwischen den beiden besten Ausprägungen sind unvorhergesehen. Dies könnte durch ungünstige Problemsequenzen für die Testreihe mit 20 Eliten verursacht sein. Ebenfalls

denkbar ist, dass für eine der beiden besten Ausprägungen entsprechend begünstigende Problemsequenzen auftraten.

Tabelle 3: Einfluss des Elitismus

Attributausprägung	Wurzelfunktion der Form: $f(x) = a * x^b + c$						bester Wert			SD
	a	b	c	x=500	Kor	SQR	all	Avg	MvgAvg	
0	6,09	0,30	-8,91	30,17	0,949	2769	69	36,1	30,9	12,3
5	15,09	0,18	-19,35	27,88	0,948	2180	60	33,8	28,9	10,9
10	5,48	0,32	-8,00	33,22	0,958	2758	71	39,7	34,3	12,4
20	10,47	0,23	-14,40	30,18	0,955	2275	68	33,8	31,2	12,0
30	2,90	0,41	-4,61	32,29	0,959	2750	70	40,6	34,4	11,2
40	7,19	0,25	-8,41	26,27	0,933	2405	66	33,1	29,1	8,8
80	17,93	0,16	-21,76	26,23	0,947	1840	54	29,5	26,9	8,3
160	52,70	0,07	-57,02	25,08	0,913	2522	60	30,9	25,6	8,9

Die Daten der Tabelle der beiden Größen von zehn und 30 Eliten sind ebenfalls relativ ähnlich. Auf den ersten Blick auffallend, ist die sinkende Standardabweichung mit wachsender Größe des Elitismus[2]. Darüber hinaus ist zu beobachten, dass die Testreihe ohne Elitismus nicht bedeutend schlechter abgeschnitten hat. Dies wird insbesondere deutlich, wenn man die besten Werte betrachtet.

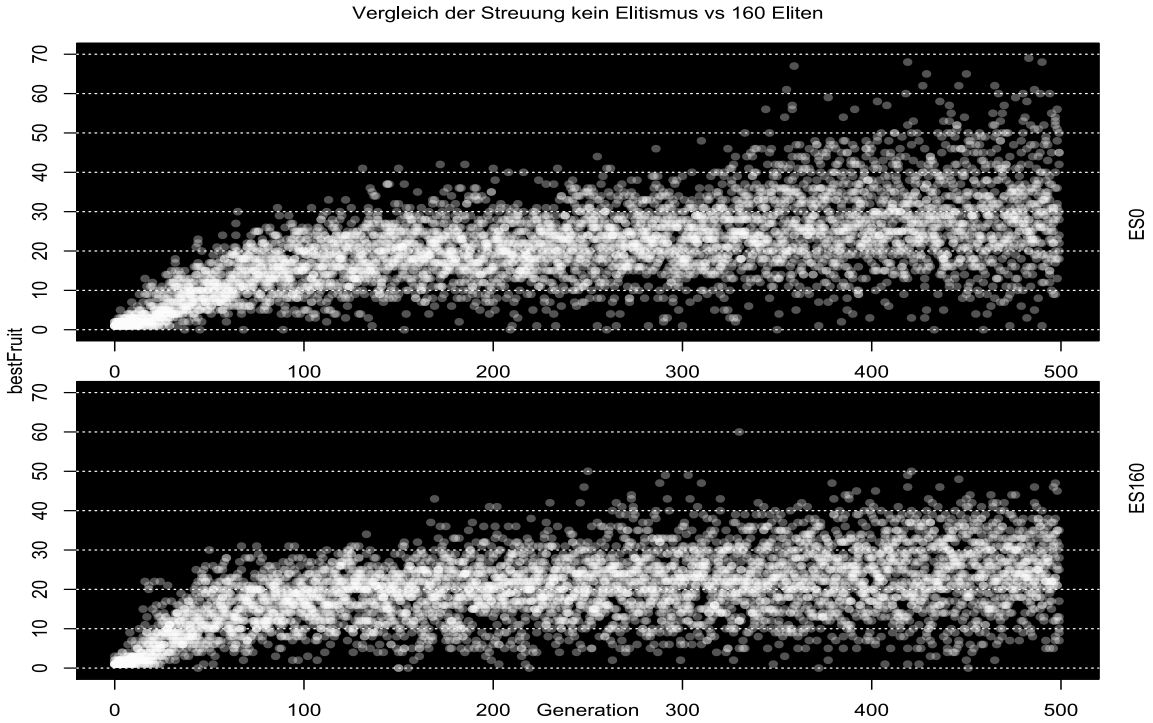

Abbildung 13: Elitismus Suchraumeingrenzung

Quelle: eigene Darstellung

[2] die relative Standardabweichung (in %) abhängig vom Erwartungswert wurde ebenfalls betrachtet.

Da die Suchraumeingrenzung anhand der Standardabweichung oder vergleichbaren Größen auf stark aggregierten Daten nur schwer möglich ist, wurde Abbildung 13 angefertigt. In dieser Abbildung sind pro Generation zehn weiße Punkte mit 33% Transparenz eingezeichnet. Jeder Punkt repräsentiert das beste Ergebnis einer Generation aus einer der zehn Iterationen. Diese Darstellung wurde gewählt, um die Streuung um den Erwartungswert über alle Beobachtungen zu visualisieren. Je deckender das Weiß, desto dichter streuen die Ergebnisse um den Erwartungswert. Bei einer stärkeren Eingrenzung des Suchraums ist eine geringere Streuung zu erwarten. Generell ist die untere Punktwolke mit dem großen Elitismus dichter, allerdings auch flacher. Da kein schlechteres Ergebnis als null Früchte erzielt werden kann, ist ein negatives Streuen eingegrenzt. Die positive Streuung ist bei dem Streudiagramm ohne Elitismus eindeutig ausgeprägter. Die negative Streuung ist beim oberen Streudiagramm ebenfalls ausgeprägter. Dies ist unter anderem an der Anzahl der Punkte, die die null Früchte Gitterlinie berühren erkennbar. Darüber hinaus ist erwähnenswert, dass der beste Wert von 60 Früchten ein deutlicher Ausreißer ist.

5.1.4. Einfluss der Mutationsrate

Die Mutationsrate nimmt Einfluss auf den Grad der Diversifizierung beim Generationswechsel. Mit Ausnahme der Eliten unterscheidet sich jedes Individuum von seinen Erzeugern aufgrund der Rekombination. Allerdings kombiniert der Prozess der Rekombination nur die Genotypen der Erzeuger. Zwar kann dies neue Verhaltensweisen hervorbringen, aber neue Attributausprägungen können dadurch nicht entstehen. Dies geschieht erst durch die Mutation. Bei 13 Inputneuronen, 14 Hiddenneuronen und vier Outputneuronen besitzt jedes neuronale Netz 238 Wichtungen ($13 * 14 + 14 * 4$) und 18 Bias-Werte ($14 + 4$) also insgesamt 256 Attribute. Die Mutationsrate bestimmt darüber, wie viele dieser Attribute verändert werden. Eine größere Mutationsrate bewirkt mehr Veränderung bzw. einen höheren Grad der Diversifizierung neuer Generationen. Eine geringe Mutationsrate hingegen grenzt die Diversifizierung ein, sodass neue Individuen ihren Vorgängern stärker ähneln. Es wird vermutet, dass ein Erhöhen der Mutationsrate bis zu einem gewissen Grad einen positiven Einfluss auf den Lernerfolg hat. Zusätzlich wird vermutet, dass ein Erhöhen der Mutationsrate durch die höhere Diversifizierung der Suchraum vergrößert wird und die Streuung

um den Erwartungswert zunimmt.

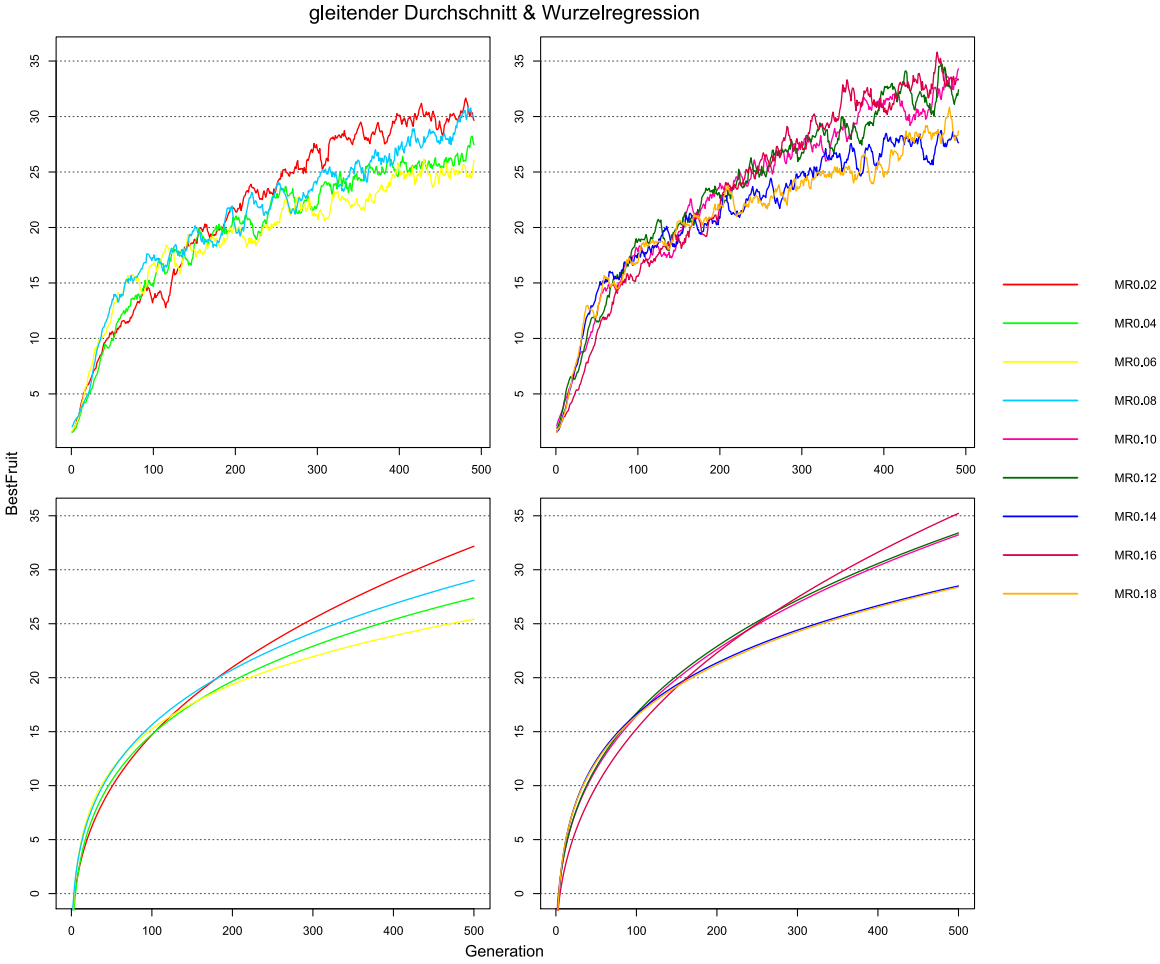

Abbildung 14: Einfluss der Mutationsrate

Quelle: eigene Darstellung

Die Graphen scheinen die Vermutung zu bestätigen. Die besten Werte wurden von der Attributausprägungen MR0.10, MR0.16 und MR0.12 erreicht. Die Graphen der linken Seite mit den geringeren Mutationsraten fallen flacher aus. Die beiden Attributausprägungen MR0.14 und MR0.18 sind wiederum flacher als die vermeintlich Besten. Aufgrund der Aggregation der Daten ist eine Deutung des Einflusses auf die Streuung anhand der Darstellung nicht möglich.

Tabelle 4: Einfluss der Mutationsrate

Quelle: eigene Darstellung

Attributausprägung	Wurzelfunktion der Form: $f(x) = a * x^b + c$						bester Wert			SD
	a	b	c	x=500	Kor	SQR	all	Avg	MvgAvg	
0,02	3,81	0,37	-6,54	32,17	0,968	2090	52	34,2	31,7	8,9
0,04	12,51	0,21	-17,50	27,40	0,947	2348	60	31,4	28,2	8,3
0,06	22,76	0,13	-26,97	25,42	0,932	2249	57	30,0	26,2	10,1
0,08	8,43	0,25	-11,21	29,04	0,943	2681	73	35,1	30,8	10,0
0,1	5,48	0,32	-8,00	33,22	0,958	2758	71	39,7	34,3	12,4
0,12	6,25	0,31	-9,22	33,40	0,955	3010	75	40,1	34,8	9,9
0,14	20,96	0,15	-25,84	28,49	0,943	2448	67	32,6	28,8	9,9
0,16	3,41	0,40	-6,62	35,24	0,964	3010	70	38,6	35,8	13,6
0,18	18,35	0,17	-22,79	28,38	0,938	2648	81	34,4	30,8	13,1

Die Tabelle lässt aufgrund der Standardabweichung vermuten, dass die Streuung mit größerer Mutationsrate zunimmt (mit Ausnahme der Attributausprägung MR0.10). Die besten Werte der Ausprägungen MR0.10 und MR0.12 sind höher als die der vermeintlich besten Attributausprägung 0,16. Deshalb wird vermutet, dass das wahre Optimum der Mutationsrate zwischen MR0.08 und MR0.14 liegt.

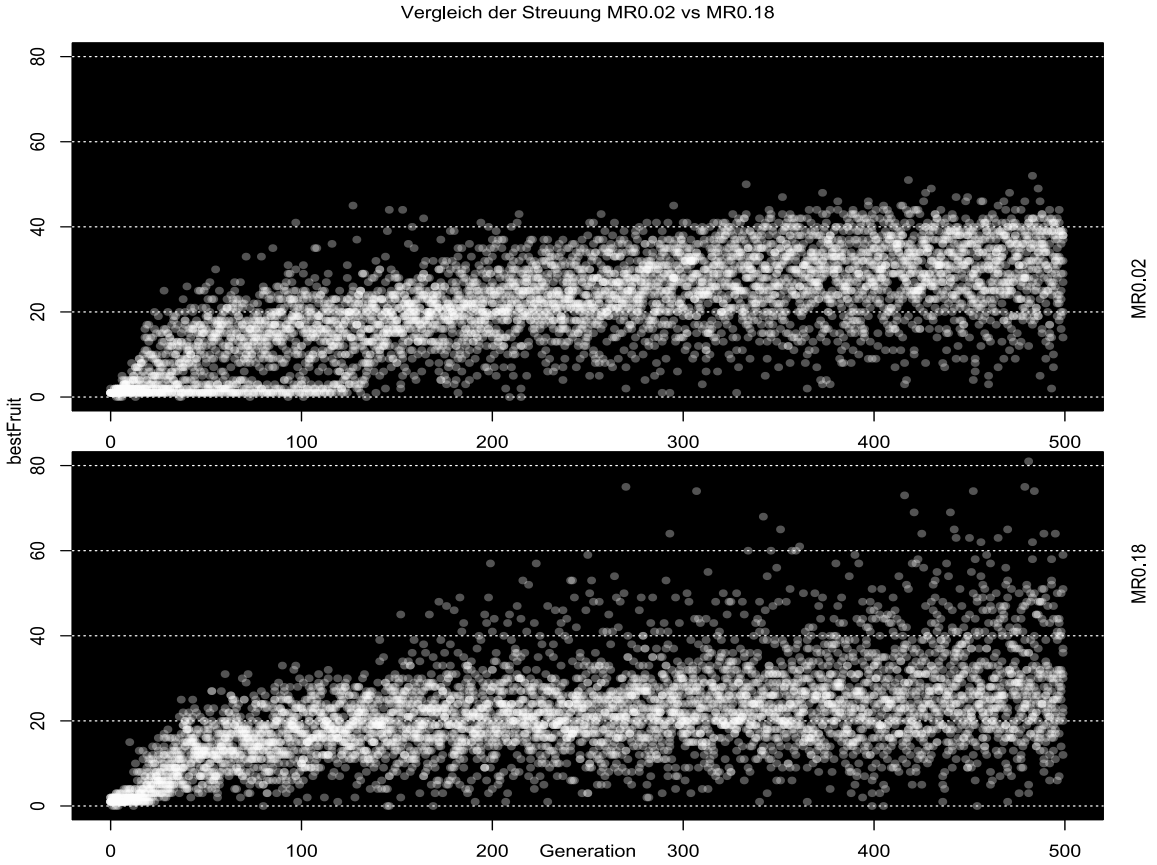

Abbildung 15: Suchraumeingrenzung der Mutationsrate

Quelle: eigene Darstellung

Um den Einfluss auf die Streuung um den Erwartungswert zu untersuchen, wurde wieder eine Punktwolkendarstellung gleich der Abbildung 13 erstellt, also jeder

Punkt repräsentiert das beste Ergebnis einer Generation aus einer, der zehn Iterationen. Die Abbildung 15 bestätigt die Vermutung der Suchraumeingrenzung. Die Testläufe mit der geringen Mutationsrate (MR0.02) sind deutlich stärker um den Erwartungswert gebündelt. Das erste Problem in Snake besteht darin, die Wände zu meiden, um den schnellen Tod nach Spielbeginn zu entgehen. Der Testlauf mit der geringen Mutationsrate benötigt deutlich länger (gemessen in Generationen), um diese erste Hürde zu überwinden. Dies ist an den Punkten mit null Früchten zu erkennen. Die Testreihe mit der hohen Mutationsrate (MR0.18) hingegen hebt sich wesentlich schneller von der null Früchte Linie ab. Aufgrund der Streuung liefert die Testreihe mit der hohen Mutationsrate mehrere, deutlich bessere Individuen als die Testreihe mit der geringen Mutationsrate. Dies ist in der Abbildung 14 nicht erkennbar und in Tabelle 4 ist lediglich der beste Wert über alle Iterationen ein Indiz dafür.

5.2. Weitere Erkenntnisse:

Neben den eigentlichen Versuchsreihen wurden weitere experimentelle Testläufe durchgeführt. Dabei wurden verschiedene Implementierungen der Kognition der Schlange, der Belohnungsfunktion, der Interaktionsmöglichkeiten der Schlange und der Problemsequenz des Spiels Snake verwendet.

5.2.1. Laufrichtungswechsel

In dem Spiel Snake haben Steuerungseingaben nur dann eine Wirkung, wenn diese orthogonal der aktuellen Laufrichtung sind. Diese Implementierung wurde für die in der Bachelor-Arbeit verglichenen Testläufe verwendet. Darüber hinaus wurden auch Abwandlungen dessen implementiert und ausprobiert. Zusätzlich wurde eine vereinfachende und eine erschwerende Methode der Verarbeitung der Steuerungsinputs implementiert. Die einfache Variante hat bei einem Input entgegen der Laufrichtung eine Abbiegung nach rechts durchgeführt und die erschwerende Variante hat es den neuronalen Netzen ermöglicht, sich jeder Zeit selbst zu töten, wenn der Steuerungsinput entgegen der aktuellen Laufrichtung gerichtet ist.

5.2.2. Wachstumsrate

Inspiriert durch die Ergebnisse von dynamisch mit dem Evolutionsfortschritt wachsenden Aufgaben von Kenneth Stanley [22], wurde das Wachstum je eingesammelter Frucht der Schlange variiert. Zusätzlich wurde die Anzahl an zusätzlichen *moves* bzw. Abbiegungen erhöht, da die Schlange zwingend mehr Züge benötigt, um sich selbst aus dem Weg zu gehen. In dem Experiment wuchs die Schlange um 4 Segmente je erreichter Frucht. Dieser Wert ist bei vielen Implementierungen des Spiels verwendet worden. Da es sich hier um das Erzielen von möglichst guten Werten handelt und nicht um die Beobachtung des Lernprozesses von Generation null an, wurde ein bereits trainiertes neuronales Netz (ebenfalls über 500 Generationen) als Ausgang für die Population der Generation

null verwendet. Die Ergebnisse waren überragend gut. Nach bereits 300 Generationen mit einem Wachstum von zwanzig Segmenten pro Frucht wurden mehr als 30 Früchte erzielt. Das entspricht einer Länge von mehr als 610 Segmenten (zehn Segmente ist die Ausgangslänge der Schlange) und einem Äquivalent von 150 erzielten Früchten bei einem Wachstum von vier Segmenten pro Frucht. Das neuronale Netz, welches während der Neuroevolution die meisten Früchte eingesammelt hat, wurde wiederum als Ausgang gewählt, um wieder mit der herkömmlichen Wachstumsrate zu trainieren. Hierbei wurden innerhalb weniger Generationen zum Teil mehr als 200 Früchte eingesammelt. Ohne eine Anpassung der Spielgeschwindigkeit (über den Speedslider) ‚spielt' das neuronale Netz mehr als 20 Minuten, ohne zu sterben. Bei einer Länge von mehr als 800 Segmenten auf einem Spielfeld von $50 * 50$ Feldern nimmt die Schlange mehr als 30% der Fläche ein. Mit einem noch größeren Wachstum von 50 Segmenten pro Frucht wurden noch größere Werte erreicht. Das beobachtete Maximum lag über 1250 Segmenten und somit bei mehr als 50% der, durch die Schlange eingenommenen Fläche. Das beste dokumentierte Ergebnis liegt bei 1110 Segmenten. Die Frage ob Snake mithilfe von Neuroevolution ‚durchgespielt' bzw. abgeschlossen werden kann, lässt sich damit zwar nicht beantworten, allerdings ist dies ein Indiz dafür, dass es mit genügend Lernaufwand möglich wäre.

5.2.3. Stufenweiser Fortschritt

Beim Beobachten der Neuroevolution mithilfe der Graphen des Prototyps ist eine markante Form des Wachstums zu erkennen. Die Lernerfolge der Neuroevolution erfolgen sprunghaft. Nach solch einem Sprung steigt die Leistung nur langsam bis zum nächsten sprunghaften Leistungszuwachs. Die Rohdaten lassen einen stufenweisen Anstieg des Erwartungswerts des besten Werts vermuten (ein Beispiel in Abbildung 14 visualisiert). Diese Stufen sind auch bei den besten Werten in Abbildung 8.B und Abbildung 13 (ES0) erkennbar. Weitere Beispiele finden sich in den Rohdaten auf dem digitalen Anhang.

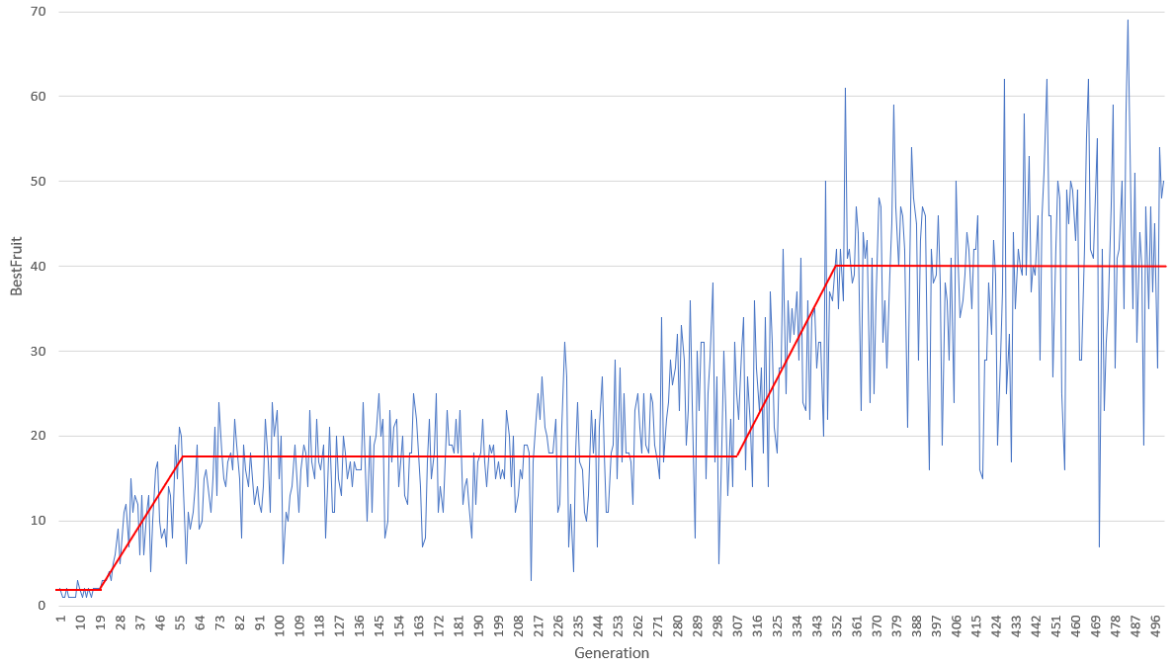

Abbildung 16: Stufenweiser Lernerfolg

Quelle: eigene Darstellung

Datenursprung: ERA_AC-ES-0_15022019_GNR500 (4).CSV

Auf den aggregierten Daten sind diese Sprünge nicht zu verzeichnen, da durch die Aggregation und zusätzlich durch den gleitenden Durchschnitt der Graph geglättet wird. Die Ebenen repräsentieren die verschiedenen, gemeisterten Problemklassen von Snake. Zuerst müssen die Individuen lernen Wände zu meiden, um einen frühzeitigen Tod zu vermeiden. Der erste Sprung repräsentiert das Erlernen der Vermeidung der Wände. Die Individuen verbringen mehr Zeit auf dem Spielfeld und beginnen das eigentliche Spielziel zu verfolgen. Des Weiteren wurde beobachtet, dass die Situation bzw. Problemklasse, in der sich zwischen Schlangenkopf und Frucht ein Schlangensegment befindet, nur indirekt gelöst wurde. Die Individuen haben keine Wegfindung um das Hindernis entwickelt. Stattdessen wurden Strategien entwickelt, um solche Probleme zu umgehen. Es wurde beobachtet, dass die Individuen am Rande des Spielfeldes Schlaufen ablaufen, bevor sie in einer Rahmenbahn das Spielfeld nach der Frucht absuchen.

5.3. Fazit

Neuroevolution bietet eine mächtige Alternative zu herkömmlichen, überwachten Lernmethoden. Mithilfe von Neuroevolution ist es möglich, neuronale Netze auf Problemklassen zu trainieren, ohne die Notwendigkeit expliziter Lösungen einzelner Probleminstanzen. Das Trainieren neuronaler Netze anhand der Güte der Lösungen, ohne die bestmögliche Lösung zu verwenden, genügt, um zufriedenstellende Ergebnisse zu erzielen. Im Anwendungsbeispiel der Bachelorarbeit wurden mittels Neuroevolution neuronale Netze trainiert, die weitaus besser in der Problemstellung Snake abschließen, als es den meisten Menschen möglich ist. Die Ergebnisse der Untersuchung des Parametereinflusses auf den Lernerfolg der Neuroevolution sind kritisch zu betrachten. Wie in Kapitel 4.1. beschrieben, ist eine Vergleichbarkeit der Testläufe nur bedingt möglich. Um signifikante Ergebnisse zu erzielen, müssten mehr als zehn Iterationen je Attributausprägung durchgeführt werden. Grundsätzlich würde die Qualität der Auswertung, gemäß dem Gesetz der großen Zahlen von mehreren Iterationen profitieren. Die stärkste Einflussnahme wurde bei der Populationsgröße beobachtet. Ein positiver Einfluss einer größeren Population auf den Lernerfolg ist nicht von der Hand zu weisen. Die Eingrenzung des Suchraums und insbesondere die Eingrenzung der Streuung um den Erwartungswert durch eine größere Anzahl von Eliten, sowie durch eine höhere Mutationsrate wurde bestätigt. Für die übrigen Parametereinflüsse liegen die Ergebnisse zu nah bei einander, sodass die Veränderung nicht eindeutig auf die veränderten Parameter zurückzuführen ist. Es wurde gezeigt, dass mit geringem Implementierungsaufwand Neuroevolution im Stande ist, zufriedenstellende Ergebnisse zu liefern. Solange Lösungen als besser bzw. schlechter differenziert werden können, ist es möglich, mit Neuroevolution komplexe Probleme zu bewältigen. Der größte Vorteil gegenüber anderen Trainingsverfahren besteht darin, dass Lösungen optimiert werden können ohne die Notwendigkeit von Lösungen konkreter Probleminstanzen. Das zeigt, dass mithilfe von Neuroevolution Probleme gelöst bzw. Lösungen optimiert werden können, ohne die Notwendigkeit, dass der Anwender die Probleme selbst lösen kann. Es existieren bereits Lernstrategien, welche die Neuroevolution um weitere Techniken ergänzen, die dem tatsächlichen, evolutionären Lernprozess von Organismen präziser abbilden und auch bessere Ergebnisse und Lernerfolge produzieren. Mithilfe dieser erweiterten Neuroevolution wird es für den Menschen zukünftig nicht nötig sein, Probleme zu verstehen, um sie zu meistern solange zwischen Lösungen differenziert werden kann.

Literaturverzeichnis

[1] Wikipedia, „Wikipedia Pageviews Analysis - Künstliche_Intelligenz,“ [Online]. Available: https://tools.wmflabs.org/pageviews/?project=de.wikipedia.org&platform=all-access&agent=user&start=2015-07&end=2019-01&pages=K%C3%BCnstliche_Intelligenz. [Zugriff am 15 02 2019].

[2] I. Döbel, M. Leis, M. M. Vogelsang, D. Neustroev, H. Petzka, S. Rüping, A. Voss, M. Wegele und J. Welz, „Maschinelles Lernen - Kompetenzen, Anwendungen und Forschungsbedarf,“ Frauehnhofer-Gesellschaft, 2018.

[3] D. O. Hebb, The Organization of Behavior, 1949.

[4] F. Rosenblatt, „The Perceptron: A probabilistic model for information storage and organization in the brain,“ 1958.

[5] A. Fraser und D. Burnell, Computer models in genetics, McGraw-Hill, 1970.

[6] P. J. Angeline, G. M. Saunders und J. B. Pollack, „An Evolutionary Algorithm that Constructs Recurrent Neural Networks,“ *IEEE Trancsactions on Neural Networks*, 1993.

[7] W.-M. Lippe, Soft-Compitung, Springer, 2006.

[8] B. Karlik, „Performance Analysis of Various Activation Functions in Generalized MLP Architectures of Neural Networks,“ in *International Journal of Artificial Intelligence And Expert Systems*, 2011, pp. 111 - 122.

[9] R. Rojas, Theorie der neuronalen Netze Eine systematische Einführung, Berlin: Springer, 1993.

[10] R. Hecht-Nielsen, „Theory of the Backpropagation Neural Network,“ in *Neural Networks for Perception*, Elsevier BV, 1992, pp. 65-93.

[11] G. Sher, Handbook of Neuroevolution Through Erlang, Springer, 2013.

[12] R. S. Sutton und A. G. Barto, Reinforcement Learning: An Introduction, 2017.

[13] R. L. Haupt und S. E. Haupt, Practical Genetic Algorithms, 1998.

[14] W. Schindler und K. Wolfgang, „Evaluation Criteria for True (Physical) Random Number Generators Used in Cryptographic Applications,“ in *Cryptographic Hardware and Embedded Systemes*, Berlin, Springer, 2002, pp. 431- 449.

[15] D. E. Knuth, The Art of Computer Programming, 1997.

[16] Oracle, „Oracle Docs,“ Oracle, [Online]. Available: https://docs.oracle.com/javase/8/docs/api/java/util/Random.html. [Zugriff am 30 01 2019].

[17] Oracle, „java.util.Random Source Code,“ [Online]. Available: http://developer.classpath.org/doc/java/util/Random-source.html. [Zugriff am 30 01 2019].

[18] D. Shiffman, „Github Repo Neuro-Evolution,“ 19 04 2018. [Online]. Available: https://github.com/nature-of-code/NOC-S18/tree/master/week10. [Zugriff am 20 10 2018].

[19] P. Gupta, „P5js.org Examples Snake,“ [Online]. Available: https://p5js.org/examples/interaction-snake-game.html. [Zugriff am 20 10 2018].

[20] A. Lipowski und D. Lipowska, „Roulette-wheel selection via stochastic acceptance,“ *Physica A: Statistical Mechanics and its Applications,* pp. 2193-2196, 15 03 2012.

[21] T. V. Elzhov, K. M. Mullen, A.-N. Spiess und B. Bolker, „minpack.lm,“ 20 11 2016. [Online]. Available: https://cran.r-project.org/web/packages/minpack.lm/index.html. [Zugriff am 25 01 2019].

[22] K. O. Stanley und J. Clune, „Uber Engineering - Endlessly Generatioing Increasingly Complex and Diverse Learning Environments and their Solutions through the Paired Open-Ended Trailblazer,“ Uber, 08 01 2019. [Online]. Available: https://eng.uber.com/poet-open-ended-deep-learning/. [Zugriff am 20 01 2019].

Anhang

Hinweis:
Der Quellcode des Prototyps sowie die verwendeten R-Scripts und sämtliche generieten Daten finden sich im digitalen Anhang.

A

Für die Attributausprägung 100 des Vergleichs der Populationsgröße liegen 60 Iterationen vor, da die Datenerfassung zum Teil über Nacht erfolgte und dementsprechend nicht nach ausreichend Iterationen (10) abgebrochen wurde. Aufgrund der verhältnismäßig schlechten Leistung des Parameters sind so viele Iterationen in gleicher Zeit gerechnet worden.

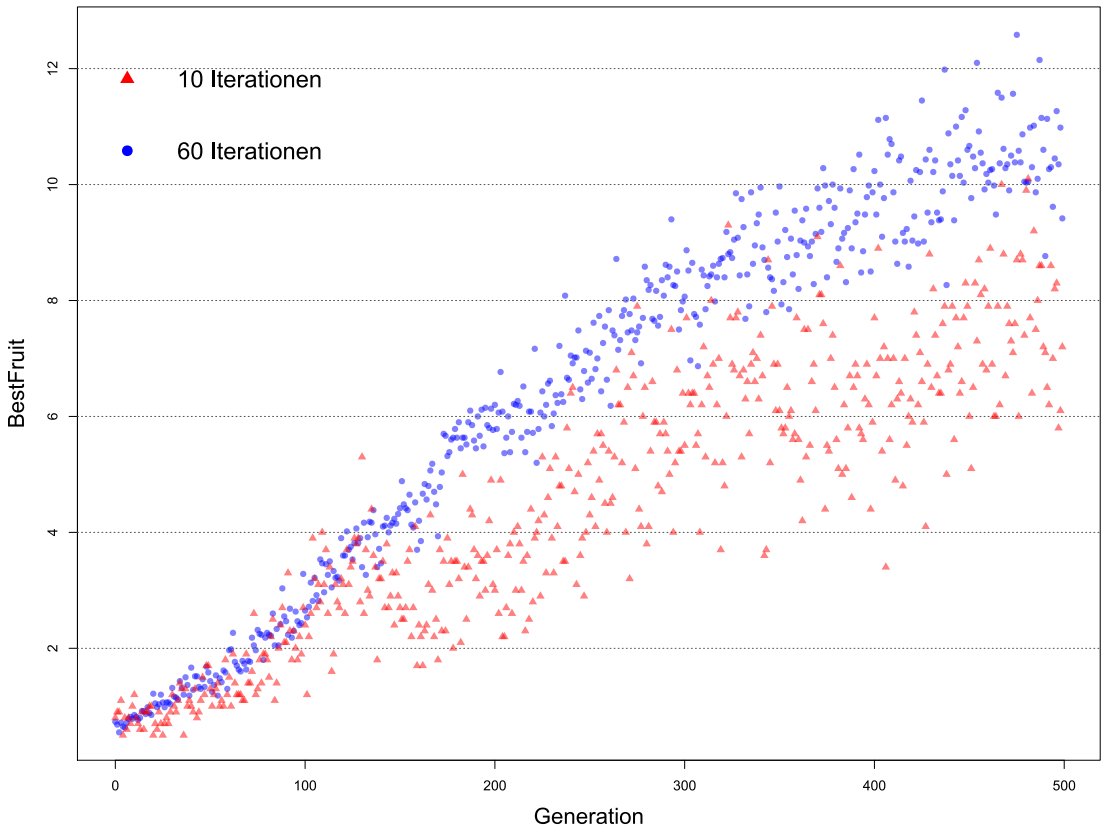

Abbildung 17: Vergleich der Streuung in Abhängigkeit der Iterationen

Quelle: eigene Darstellung