Hochschule Esslingen
University of Applied Sciences

Fakultät Informationstechnik
Softwaretechnik und Medieninformatik

Bachelorarbeit

Entwicklung einer Künstlichen Intelligenz unter Einsatz der Evolutionären Strategie zum Lösen eines Videospiels

Benedikt Grau
Wintersemester 2018/2019

Erstprüfer: Prof. Dr. rer. nat. Dipl.-Math. Jürgen Koch
Zweitprüfer: Dipl.-Ing. (FH) Kevin Erath M. Sc.

IT DESIGNERS GRUPPE

Firma: IT-Designers GmbH, Esslingen am Neckar
Betreuer: Dipl.-Ing. (FH) Kevin Erath M. Sc.

Ehrenwörtliche Erklärung

Hiermit versichere ich, die vorliegende Arbeit selbstständig und unter ausschließlicher Verwendung der angegebenen Literatur und Hilfsmittel erstellt zu haben.

Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungsbehörde vorgelegt und auch nicht veröffentlicht.

Esslingen, den _____

Benedikt Grau

Danksagung

An dieser Stelle möchte ich mich bei einigen Personen bedanken, die mir während meines Studiums und während der Durchführung dieser Arbeit persönlich und fachlich zur Seite standen und mich unterstützt haben.

Besonderer Dank gilt dabei meinen Betreuern Herrn Prof. Dr. rer. nat. Dipl.-Math. Jürgen Koch und Herrn Dipl.Ing.(FH) Kevin Erath M.Sc., welche mir diese Arbeit ermöglicht haben und für Fragen gerne zur Verfügung standen.

Zudem möchte ich mich bei der Firma IT-Designers GmbH für die Ermöglichung dieser Arbeit und die Erfahrungen die ich dabei machen konnte bedanken.

Für ihre konstruktive Kritik, Anmerkungen und Korrekturen danke ich außerdem Herrn Stefan Kraft, Herrn Felix Segiet und Frau Mara Lehmann.

Desweiteren möchte ich mich bei den Jungs und Mädels von CHG und TheBrightDarkSide bedanken, dafür, dass sie immer an meiner Seite standen und für einen entspannenden Ausgleich sorgten.

Zum Schluss möchte ich noch meiner Familie danken, welche mich immer wieder ermutigt hat, meine Fähigkeiten zu testen und zu erweitern.

„'Come on, [...]! It isn't rocket science, it's just quantum physics!' -The Doctor (Matt Smith)“

– Steven Moffat

Kurzfassung

Diese Arbeit befasst sich mit dem Entwickeln und dem Trainieren einer Künstlichen Intelligenz. Die Implementierung erfolgt in Python und verwendet das Gym-Retro System von OpenAI als Schnittstelle zu einem Emulator. Auf diesem Emulator läuft die Trainingsumgebung der KI, ein Spiel für das Nintendo Entertainment System.

Die KI soll dabei ein Convolutional Neural Network verwenden, um Entscheidungen zu treffen. Dieses wird mit Keras und Tensorflow implementiert und soll als Eingabe den Bildschirm und als Ausgabe einen vereinfachten Controller verwenden.

Ziel der KI ist es, einzelne oder mehrere Level des Spieles nach einem längeren Training lösen zu können.

Inhaltsverzeichnis

Abbildungsverzeichnis

Listings

Abkürzungsverzeichnis

KI Künstliche Intelligenz

ML Maschinelles Lernen

EA Evolutionäre Algorithmen

ES Evolutionäre Strategie

RL Bestärkendes Lernen

IQ Intelligenzquotient

FFNN Feedforward Neural Network

RNN Recurrent Neural Network

CNN Convolutional Neural Network

ReLU Recitfied Linera Units

NES Nintendo Entertainment System

1 Einleitung

1.1 Motivation

Künstliche Intelligenzen sind in Hollywood schon lange im "Einsatz". Oftmals spielen sie dabei die Rolle des Antagonisten. Schon im Jahr 1982, im Film Tron, versucht die Künstliche Intelligenz (KI) MCP (Master Control Program) die Kontrolle über eine künstliche Welt zu erlangen [11]. Auch im Film Matrix, aus dem Jahre 1999, versuchen KI-Roboter die Menschen zu unterwerfen. Ihr Ziel dabei ist es, Energie aus den Menschen für ihre eigenen Zwecke zu gewinnen [10].

Auch im realen Leben gibt es KIs, die sich bestimmten Probleme stellen, doch bis zur menschenvernichtenden Super-KI sollte es noch ein paar Jahre brauchen. So verwendet zum Beispiel Apple bei ihrem Gesichterkennungssystem eine KI mit Neuronalem Netz. Sie kann dem Nutzer dabei helfen bessere Bilder zu kreieren oder in einem Bild erkennen, wer zu sehen ist [25].

Doch warum sollte man eine KI darauf trainieren ein Videospiel zu lösen? Und vor allem, warum investieren sogar große Firmen wie Apple deswegen so viel?

Ein zentraler Punkt bei der Kreierung von KIs ist vor allem die Forschung verschiedener Trainingsmethoden. Und hier kommen die Videospiele zum Einsatz. Sie werden vor allem verwendet, um neue Algorithmen zu testen und diese mit bisherigen Methoden zu vergleichen. [13]

Aber warum testet man diese neuen Algorithmen anhand von Videospielen und nicht an anderen Umgebungen? Der Grund für den Einsatz von Videospielen ist der, dass der Aufwand und die Kosten gering sind. Videospiele bieten eine riesige Anzahl an sehr unterschiedlichen Umgebungen, welche trotzdem die gleiche Eingabe (Tastatur & Maus, Joystick, ...) und Ausgabe (Bildschirm) besitzen. Eine KI, die somit gleichzeitig mit unterschiedlichen Spielen trainiert, kann Zusammenhänge verstehen und lernt nicht nur auswendig, wie man ein Spiel löst. [27]

Die Umgebungen, die dadurch zur Verfügung stehen, sind mehr als ausreichend, um neue Algorithmen zu testen. Und auch die Schnittstelle zu diesen Umgebungen ist durch den Einsatz von Emulatoren nicht nur simpel, sondern ermöglicht auch einen einheitlichen Zugriff auf verschiedene Spiele.

1.2 Zielsetzung

Das Ziel dieser Arbeit ist es, eine KI für das Spiel SuperMarioBros., ein Jump'n Run für das Nintendo Entertainment System, zu entwickeln. Nach einer Trainingsphase soll es der KI möglich sein die ersten Level des Spieles ohne sterben durchspielen zu können. Dies soll zeigen, dass es ein besseres Ergebnis erzielen kann, als eine zufällig agierende Maschine.

Hierzu wird OpenAIs Umgebungsframework Gym-Retro verwendet. Dieses Framework bietet eine Schnittstelle zu Retro-Konsolen, wie zum Beispiel Game Boy oder Nintendo Entertainment System. Über diese Schnittstelle zu den Konsolen können dann auf die einzelnen Spiele der Systeme zugegriffen werden.

Das Gehirn der KI wird durch ein Convolutional Neural Network mit Keras und Tensorflow implementiert und mithilfe eines Evolutionären Algorithmus trainiert.

1.3 Aufbau dieser Arbeit

Der Aufbau dieser Arbeit lässt sich in die nachfolgenden Kapitel gliedern:

Grundlagen
Die Grundlagenkapitel bieten einen theoretischen Einblick in das Maschinelle Lernen und die Evolutionäre Strategie. Es wird gezeigt, wie diese beiden Konzepte zusammen verwendet werden, um eine KI zu kreieren und zu trainieren.

Systemaufbau
Im Systemaufbau geht es darum, die Grenzen des Systems aufzuzeigen. Es werden Anforderungen an das System gestellt und ein systematischer Aufbau des Systems aufgezeigt.

Implementierung
In der Implementierung wird dokumentiert, wie das System umgesetzt wird. Es wird gezeigt, wie die einzelnen Module zusammenspielen und wie dies implementiert wird.

Ergebnisse
In diesem Kapitel werden die Ergebnisse des Trainings der KI präsentiert. Hierbei wird aufgezeigt, wie diese Ergebnisse zustande kommen und was diese aussagen.

2 Künstliche Intelligenz

2.1 Definition

Wie entwickelt und baut man eine KI? Doch bevor man diese Frage beantworten kann, müssen erst einmal ein paar Begriffe definiert und verstanden werden. Was ist das Ziel einer KI und wie erreicht sie dieses? Um diese Fragen zu beantworten, ist es wichtig, den Unterschied zwischen einer KI und dem Maschinellen Lernen zu verstehen.

Die Begriffe KI und Maschinelles Lernen (ML) werden oftmals austauschbar verwendet, sie bezeichnen jedoch nicht die gleiche Sache. ML ist nur ein Teil von KI. [14] Es beschreibt eine Methode für die Kreierung einer KI. Und KI ist die Bezeichnung eines Systems oder einer Maschine, die sich aufgrund eines "Gehirns" intelligent verhält.

Doch was bedeutet "intelligent" zu sein für eine Maschine? Ist ein Taschenrechner intelligent, weil er zwei Zahlen zusammenrechnen kann? Um diese Frage zu klären, muss zuerst einmal der Begriff "Intelligenz" im Zusammenhang mit einer Maschine definiert werden.

Laut dem Duden ist die "Intelligenz" die "Fähigkeit [des Menschen], abstrakt und vernünftig zu denken und daraus zweckvolles Handeln abzuleiten" (vgl. [6]). Es wurde sogar eine Skala erfunden, die genau dies durch einen Wert wiedergeben soll. Der sogenannte Intelligenzquotient (IQ) soll mithilfe von Tests die Intelligenz eines Menschen messen. [31]

Aber was bedeutet dies im Falle einer Maschine? Hier kann man nicht einfach eine Skala erfinden und mit einer simplen Funktion den IQ der Maschine berechnen. Jedoch kann man den Menschen direkt mit der Maschine vergleichen und so eine ungefähre Annäherung der Intelligenz der Maschine bestimmen. Als durchschnittlich intelligent gilt ein Erwachsener, wenn er oder sie einen IQ im Bereich von 90 bis 109 besitzt. Also kann man bei einer Maschine dann von einer Intelligenz ausgehen, wenn sie genauso "schlau" ist wie der Mensch.

Nun stellt sich aber eine neue Frage: Was bedeutet so "schlau" sein wie der Mensch? Dies lässt sich so erklären: Die Ergebnisse, die ein Mensch und eine Maschine für eine spezielle Aufgabe liefern, müssen gleich oder zumindest ähnlich sein.

Zusammengefasst kann man von einer Künstlichen Intelligenz dann reden, wenn eine Maschine eine bestimmte Aufgabe löst und dabei vergleichbare Resultate erzielt, wie sie im Durchschnitt der Mensch erzielen würde.

2.2 Maschinelles Lernen

2.2.1 Entwicklung einer KI

Es gibt viele verschiedene Möglichkeiten eine KI zu entwickeln. Bei ML geht es darum, einer Maschine etwas beizubringen, sie soll selber Sachen lernen. Das Ziel ist nicht, Befehle zu programmieren, die dann ausgeführt werden. Die Maschine soll selber zu diesen Befehlen und schlussendlich zu einer Lösung kommen. [14]

Dazu wird ein Model auf vorhandene Daten trainiert, um anschließend eine Vorhersage auf noch unbekannte Daten machen zu können. Man benötigt also ein Programm, welchem man eine Eingabe senden kann und das eine Ausgabe zurückgibt. Dieses Programm muss eine Möglichkeit haben, die Ausgabe anpassen zu können.

Generell gibt es zwei Möglichkeiten, wie man dem Model diese Daten zum Lernen übergibt.

Überwachtes Lernen (Supervised Learning)
: Dem Model werden sowohl Eingabe- als auch Ausgabe-Werte übergeben. Das Model soll so trainiert werden, dass die Ausgabe an das gewünschte Ergebnis anpasst wird.

Unüberwachtes Lernen (Unsupervised Learning)
: Dem Model werden nur Eingabe-Werte übergeben. Es soll daraus selber lernen, welche Ausgabe erwartet wird.

Diese zwei Trainingsarten machen aber nicht direkt alle Arten von ML aus. Es gibt weitere Trainingsarten, welche beide Arten kombinieren können. [15] Beispielsweise nimmt zwar das Bestärkende Lernen, wie das Überwachte Lernen Eingabe- und Ausgabe-Werte entgegen, aber diese Ausgabe ist nicht unbedingt die gewünschte Ausgabe, sondern nur eine Richtung, in welche es lernen soll (siehe auch Kapitel 2.4).

2.2.2 Funktionsweise

Die Funktionsweise des ML hängt sehr stark von der eigentlichen Trainingsart und dem implementierten Algorithmus ab. Jedoch ist das Ziel all dieser Methoden dasselbe: Eine generelle Lösung finden. Diese Lösung soll nicht nur auf bekannten (trainierten) Daten eine korrekte Ausgabe liefern, sondern vor allem auf noch unbekannten Daten eine vernünftige Ausgabe erzeugen. [21]

Dabei ist es wichtig, dass die Komplexität des Lösungsansatzes gleich der Komplexität der Aufgabe ist. Ist sie größer, dann kommt es zu einer Überanpassung (engl. overfitting) auf den Trainingsdaten. Das heißt, dass das Model zwar korrekte Ausgaben für alle Trainingsdaten liefern kann, jedoch ist die Lösung nicht generell genug. Sie scheitert an neuen Daten. Ist dies der Fall, muss ein komplexerer Lösungsansatz gewählt werden.

Andersherum kann es dieses Problem nicht geben. Das heißt, ist die Komplexität des Lösungsansatzes kleiner als die der Aufgabe, so führt dies erst einmal nicht zu einem

Problem. Je komplexer der Lösungsansatzes ist, desto länger braucht es normalerweise auch, um auf eine Lösung zu kommen. [17]

2.3 Neuronale Netze

2.3.1 Aufbau

Der Mensch nimmt seine Umgebung durch seine sechs Sinne wahr. Die dadurch erworbenen Daten werden hauptsächlich im Gehirn zu Informationen verarbeitet und interpretiert. Sieht der Mensch zum Beispiel ein Bild, auf dem ein Stuhl abgebildet ist, dann nimmt er diese Daten mit den Augen auf und sie werden an das Gehirn weitergeleitet. Das Gehirn kann aus diesen Daten wiederum herleiten, dass ein Bild mit einem Stuhl von den Augen gesehen wurde. Um dies bewerkstelligen zu können, besteht das Gehirn aus ungefähr 100 Milliarden Nervenzellen, die durch Verbindungen untereinander zu einem Netz aufgespannt sind. [30] Vereinfacht dargestellt, kommen Daten an der einen Seite dieses großen Netzes an und werden einmal durch dieses Netz durchgeschickt. An den einzelnen Nervenzellen werden diese Daten weiterverarbeitet, so dass am Ende der anderen "Seite" des Netzes Informationen stehen, die der Mensch interpretieren kann.

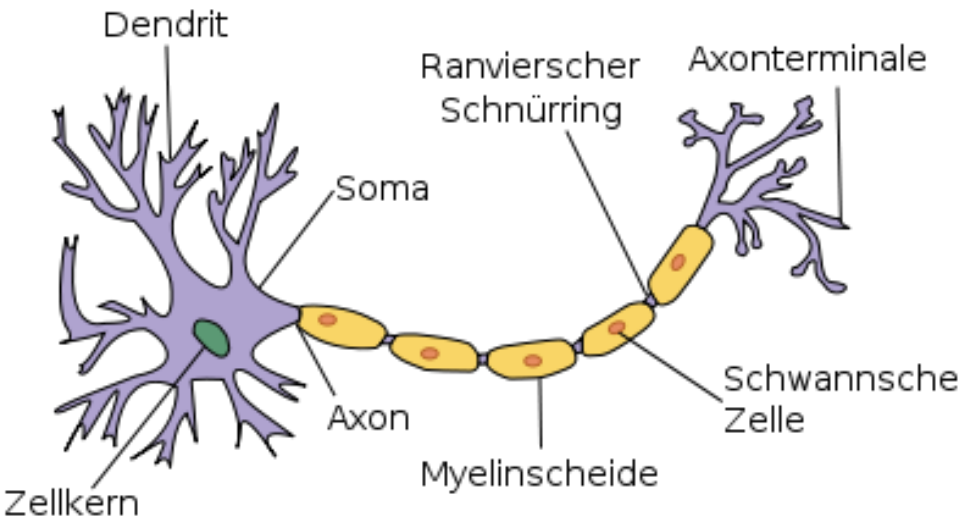


Abbildung (2.1): Schematische Darstellung einer Nervenzelle des Menschen [33]

Um eine Maschine also so intelligent (oder sogar intelligenter) zu machen wie einen Menschen, benötigt sie ein "Gehirn". Hier kommt das Neuronale Netz ins Spiel. Es funktioniert nach dem gleichen Prinzip wie das Gehirn. Es besteht aus vielen verschiedenen Neuronen, die wie beim Gehirn untereinander verbunden sind. [23] Auch hier kommen die Daten, die man untersuchen möchte, an der einen Seite des Netzes rein und die verarbeiteten Informationen an der anderen Seite heraus.

Im Neuronalen Netz übernehmen Neuronen die Aufgabe der Nervenzellen im echten Gehirn. Mehrere solcher Neuronen werden üblicherweise in Schichten zusammengefasst. Ein Neuronales Netz besteht je nach Architektur aus unterschiedlich vielen Schichten mit jeweils unterschiedlich vielen Neuronen. Es werden jedoch immer mindestens zwei Schichten benötigt. [18] Dies ist zum einen die Eingabeschicht, welche die Daten nur entgegennimmt und zum anderen die Ausgabeschicht, welche das Endresultat der Verarbeitung wieder zurückgibt. Zwischen dieser Eingabe- und Ausgabeschicht können sich beliebig viele versteckte Schichten (engl. hidden layer) befinden (siehe Abbildung 2.2).

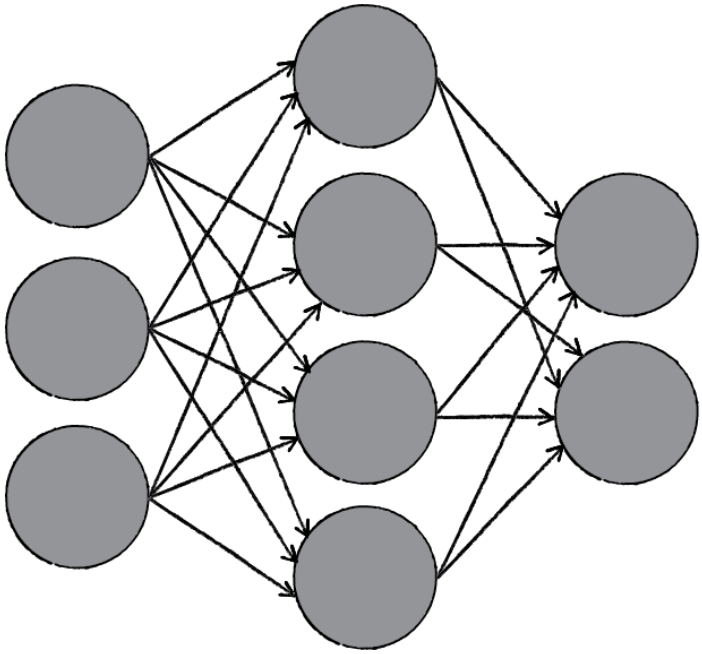


Abbildung (2.2): Aufbau eines einfachen Neuronalen Netzes mit Eingabeschicht, einer versteckten Schicht und Ausgabeschicht [23]

2.3.2 Funktionsweise

Die Funktionsweise eines Neuronalen Netzes versteht man zunächst nur, wenn man weiß, wie einzelne Neuronen funktionieren. In Abbildung 2.3 ist ein solches künstliches Neuron und dessen Bausteine zu sehen. Die Funktionsweise eines solchen Neurons ist im Folgenden in mehrere Schritte aufgeteilt.

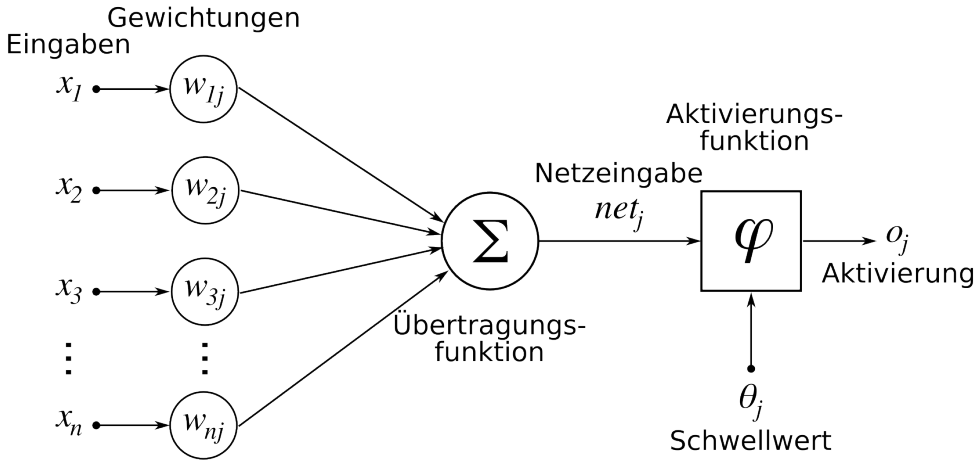


Abbildung (2.3): Künstliches Neuron mit seinen Eigenschaften [3]

Der erste Schritt ist die Eingabe an das Neuron. Jedes Neuron bekommt durch seine eingehenden Verbindungen mehrere Eingaben zugesendet. Diese Eingaben können entweder die Ausgaben von anderen Neuronen sein oder direkt von der Eingabe in das Neuronale Netz kommen. Die Eingabefunktion hat nur die Funktion die Werte entgegenzunehmen aber verarbeitet sie nicht weiter. Als Nächstes werden alle diese Eingabewerte mit jeweils einem Gewicht verrechnet. Jedes Gewicht wird dabei mit seinem korrespondierenden Eingabewert multipliziert. Alle gewichteten Werte werden anschließend in der Übertragungsfunktion zu einem Wert addiert. Dieser Wert bildet die Netzeingabe in das künstliche Neuron. [18]

Die Formel 2.1 zeigt die Berechnung der Netzeingabe des Neurons **j** unter Berücksichtigung der Eingabe **x** und deren Gewichte **w** (siehe dazu auch Abbildung 2.3).

$$net_j = \sum_{i=1}^{n} x_i w_{ij} \tag{2.1}$$

Die Berechnung der Ausgabe des künstlichen Neurons ist der letzte Schritt. Dazu wird zuerst ein Schwellenwert Θ auf die Netzeingabe **net** addiert. Dieser Schwellenwert verschiebt die Netzeingabe des Neurons und beeinflusst somit die Empfindlichkeit eines Neurons. Dies ist die Eingabe für die Aktivierungsfunktion ρ (siehe Formel 2.2).

$$o_j = \rho(net_j + \Theta_j) \tag{2.2}$$

Die Aktivierungsfunktion bestimmt den Ausgabewert **o** des Neurons **j** unter Berücksichtigung der Netzeingabe **net** und dessen Stellenwertes Θ.

Es gibt keine Regeln, welche Aktivierungsfunktion verwendet werden muss. Im Prinzip kann jede Funktion als Aktivierungsfunktion fungieren. Jedoch gibt es fünf Funktionen, welche sehr häufige Verwendung finden, sei es wegen ihrer Einfachheit oder um einem speziellen Zweck zu dienen. [28]

Die einfachste dieser Aktivierungsfunktionen ist die lineare Funktion. Da die Ausgabe dieser Funktion gleich der Eingabe ist, wird nichts Weiteres gemacht, als den Wert zur Ausgabe des Neurons weiterzuleiten (siehe Abbildung 2.4).

$$f_{linear}(x) = x$$

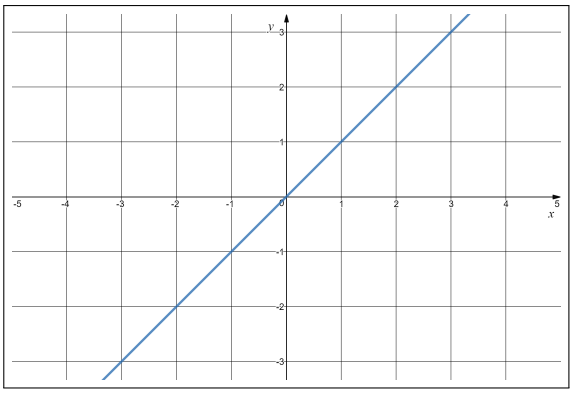


Abbildung (2.4): Lineare Funktion

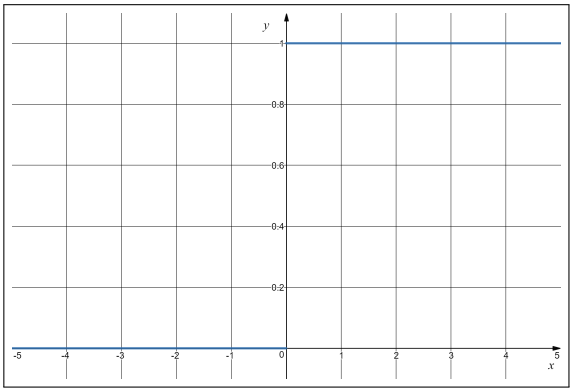


Abbildung (2.5): Schwellenwertfunktion

Die Schwellenwertfunktion ist genauso simpel wie die lineare Funktion. Sie nimmt die Eingabe und schaut, ob diese positive oder negativ ist. Ist sie größer als oder gleich 0, so ist die Ausgabe 1. Ist sie nicht größer als 0, so ist die Ausgabe gleich 0 (siehe Abbildung 2.5).

$$f_{schwellwertfunktion}(x) = \begin{cases} 1, x \geq 0 \\ 0, sonst \end{cases}$$

Die Sigmoid-Funktion ähnelt die Schwellenwertfunktion. Im Gegensatz dazu hat sie aber einen glatten Übergang und ist nicht abrupt. Kleine Wertunterschiede im Bereich um x=0 haben einen großen Unterschied in der Ausgabe zufolge. Außerdem werden alle Werte normalisiert und liegen anschließend zwischen 0 und 1 (siehe Abbildung 2.6).

$$f_{sigmoid}(x) = \frac{1}{1 + e^{-x}}$$

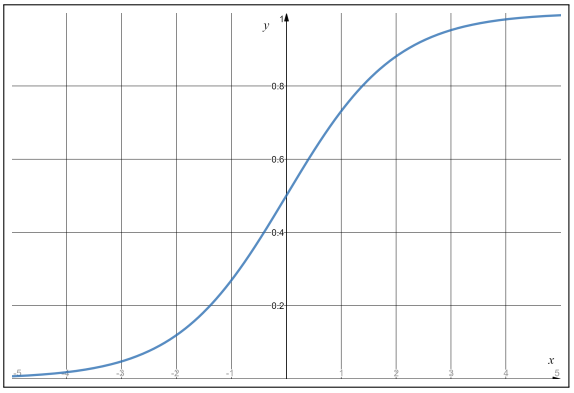


Abbildung (2.6): Sigmoid-Funktion

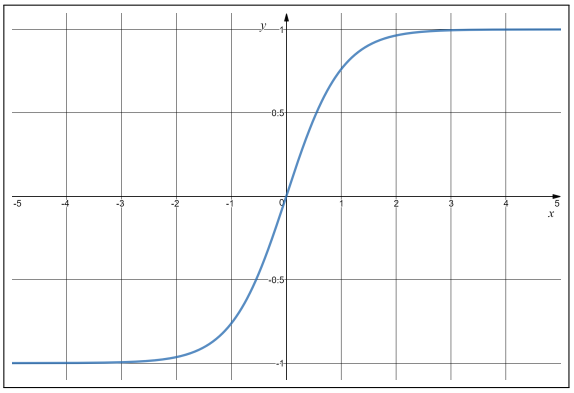


Abbildung (2.7): Tanh-Funktion

Ähnlich zur Sigmoid-Funktion ist die Tanh-Funktion. Sie ist mathematisch gesehen eine gestreckte Version der Sigmoid-Funktion. Es gibt nur einen entscheidenden Unterschied zur Sigmoid-Funktion: Die Werte liegen in einem Intervall von -1 bis 1 anstelle von 0 und 1 (siehe Abbildung 2.7).

$$f_{tanh}(x) = tanh(x)$$

Die Recitfied Linera Units (ReLU)-Funktion ist eine Aktivierungsfunktion, die häufig in Convolutional Neural Networks (CNN) zum Einsatz kommen. Sie ist der linearen Funktion sehr ähnlich. Positive Werte bleiben, wie bei der linearen Funktion, unverändert und werden direkt zur Ausgabe weitergeleitet. Negative Werte hingegen werden, egal wie sehr negativ sie sind, immer auf 0 gesetzt (siehe Abbildung 2.8).

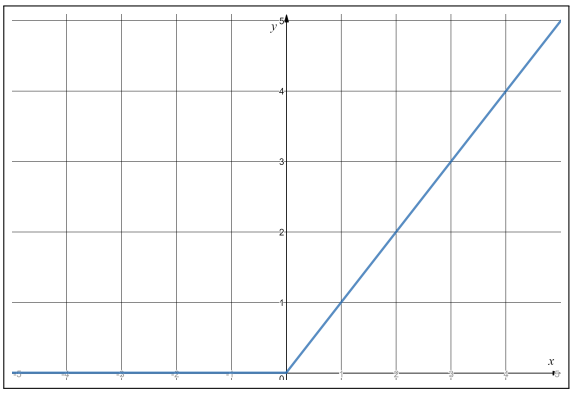


Abbildung (2.8): Relu-Funktion

$$f_{relu}(x) = \begin{cases} x, x \geq 0 \\ 0, sonst \end{cases}$$

Die Verbindung zwischen den Neuronen ist einzig und allein dazu da, den Ausgabewert eines Neurons zur Eingabe der Neuronen der nächsten Schicht zu transportieren.

2.3.3 Verschiedene Architekturen

Feedforward Neural Network

Das Feedforward Neural Network (FFNN) ist das einfachste von allen Neuralen Netzwerken. [9] Wie der Name dieses Netzwerkes auch bereits sagt, werden in diesem Fall die Daten von der Eingabe in eine Richtung bis zu Ausgabe verarbeitet (Feedforward, zu Deutsch Vorwärtsschieben).

Ein FFNN besteht aus einer Eingabeschicht, einer Ausgabeschicht und null bis mehreren versteckten Schichten (siehe Abbildung 2.9). In jeder Schicht ist die Ausgabe jedes einzelnen Neurons wiederum die Eingabe eines einzelnen Neurons der nächsten Schicht.

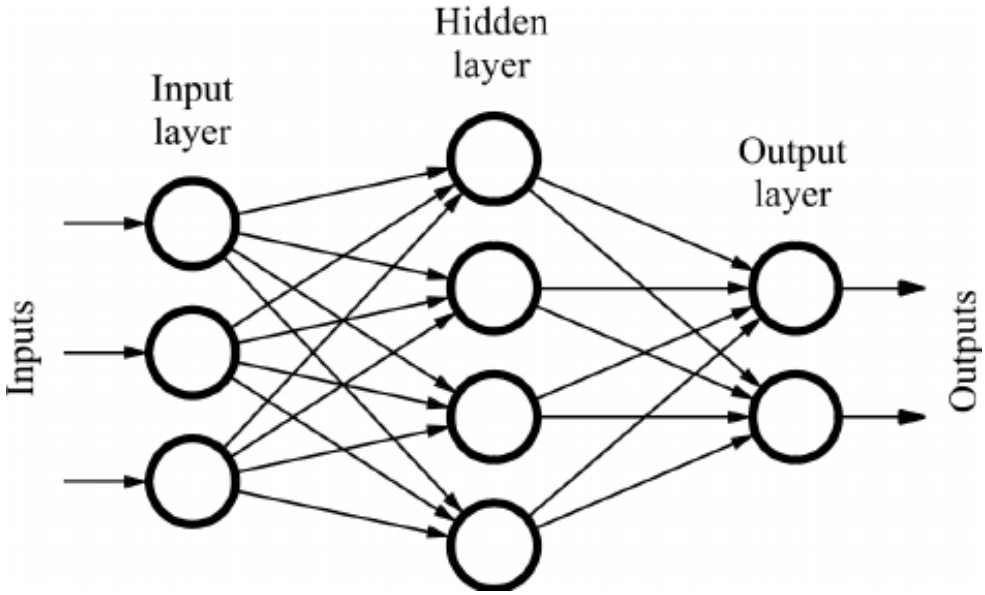


Abbildung (2.9): Ein Feedforward Neuronales Netz [26]

Recurrent Neural Network

Das Recurrent Neural Network (RNN) ist ein Wiederholendes Netzwerk. Was diese Architektur besonders macht: Es können zeitliche Abhängigkeiten erkannt und verarbeitet werden. Solche Netzwerke werden deshalb gerne in Audio- und Videoanalysen verwendet.

Das liegt daran, dass die Eingabe nicht einfach nur von vorne nach hinten verarbeitet wird und eine Ausgabe erzeugt wird, wie es beim FFNN der Fall ist, sondern es werden auch Daten gespeichert. Diese Daten werden bei späteren Eingaben und der Berechnung deren Ausgaben mitberücksichtigt. [9] Durch diesen Speicher, in dem eine oder sogar mehrere Datenreihen aus vorherigen Eingaben gespeichert ist, können zeitliche Zusammenhänge von Informationen entstehen. Abbildung 2.10 zeigt, wie einzelne Neuronen eine Verbindung nach ”hinten” besitzen. Die bedeutet, dass die Ausgabe dieser Neuronen im nächsten Durchlauf als zusätzliche Eingabe dient.

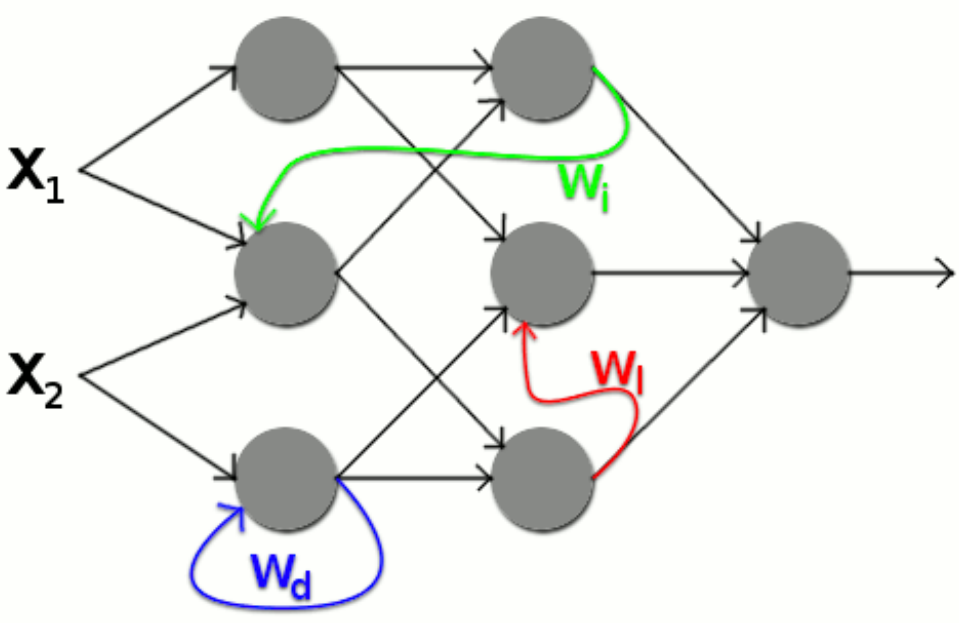


Abbildung (2.10): Beispiel eines Recurrent Neural Networks [32]

Convolutional Neural Network

Das CNN ist eine spezielle Form des FFNN. Der Vorteil dieser Architektur ist es, Zusammenhänge in einem Raum zu erkennen. Es kann Strukturen auf einem Bild erkennen und somit feststellen, welche Objekte und wo sie sich in einem Bild befinden.

Es besteht, wie ein FFNN aus einer Eingabe-, mehreren versteckten und einer Ausgabeschicht. Zusätzlich dazu werden zwischen der Eingabeschicht und der ersten versteckten Schicht noch eine oder mehrere Convolutional Schichten (zu Deutsch faltende Schicht) und eine oder mehrere Poolingschichten gelegt. [9]

Eine Poolingschicht fasst die Eingabe nur zusammen und verkleinert sie dadurch. Dazu nimmt sie jeweils alle Eingaben in einem bestimmten Bereich und berechnet daraus einen einzigen Wert, somit wird die Datenmenge verringert. Ein Problem bei Poolingschichten ist jedoch, dass man die Position von Objekten in einem Raum verlieren kann. Wenn man Eingaben in einer gemeinsamen Umgebung zusammenfasst, dann verliert man die genaue Position des berechneten Wertes. Wie in Abbildung 2.11 zu sehen ist, wird dabei oftmals der maximale Wert in diesem Bereich übernommen (siehe rechte Seite). Dieses Verfahren wird auf alle Eingaben angewendet (siehe linke Seite). Es kann aber auch je nach Architektur ein Durchschnitt berechnet werden oder auch der minimale Wert genommen werden.

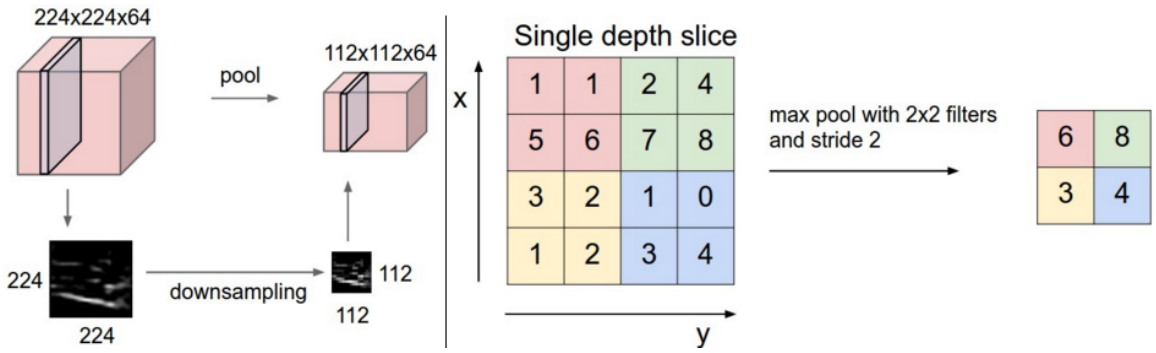


Abbildung (2.11): Beispiel einer Max-Poolingschicht [14]

Eine Faltschicht besteht nicht aus mehreren Neuronen, die für sich ihre Berechnung machen, sondern aus mehreren Filtern, die nach Zusammenhängen in verschiedenen Bereichen der Eingabe suchen. Sie funktioniert nach dem gleichen Prinzip wie die Faltung in der Mathematik. In der Mathematik nimmt man dazu zwei Funktionen, spiegelt eine von ihnen, lässt sie gegeneinander laufen und berechnet das Integral, das dadurch an jeder Stelle entsteht (siehe Formel 2.3, dabei steht der "*" (Stern) nicht für die Multiplikation, sondern für die Faltung).

$$h(x) = f(x) * g(x) = \int_{-\infty}^{\infty} f(\tau)g(x-\tau)d\tau \tag{2.3}$$

Im Falle eines Neuronalen Netzes ist die erste Funktion die Eingabe der Schicht und die zweite Funktion die Filter dieser Schicht. Diese Filter bestehen aus den Gewichten dieser Schicht. Wie in der Mathematik wird die zweite Funktion, in dem Fall Filter, über die erste geschoben. [5] Hier ist eine Spiegelung des Filters jedoch nicht notwendig. Da es sich nicht um Funktionen handelt, wird auch keine Fläche berechnet, sondern das Frobenius-Skalarprodukt (siehe Formel 2.4). Dieses Filter bewegt sich somit einmal über das ganze Bild und berechnet jeden Wert einzeln (siehe Abbildung 2.12).

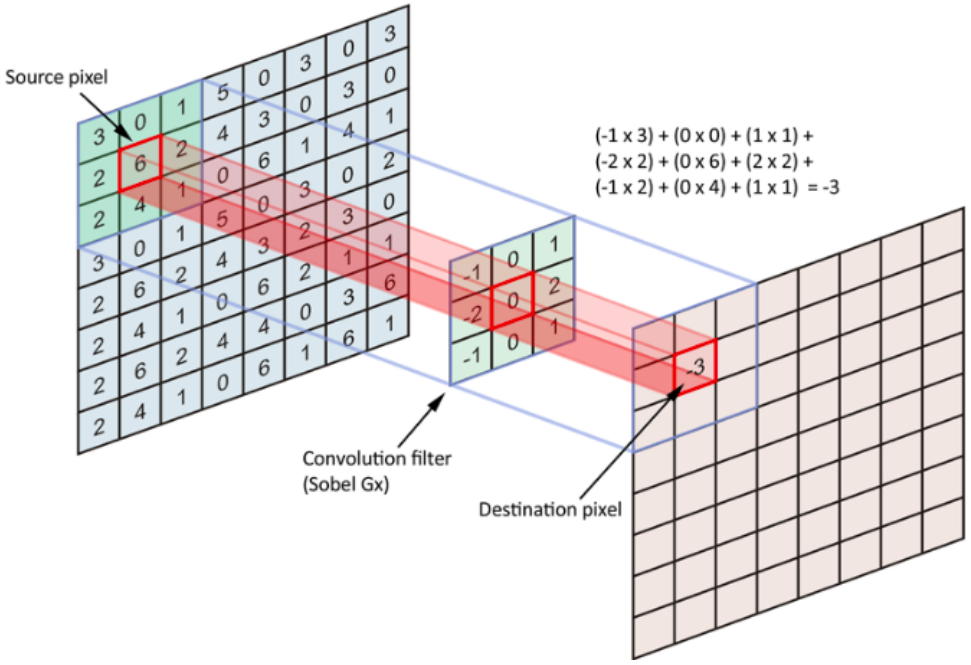


Abbildung (2.12): Eine Faltschicht mit Filter und einer Beispielrechnung [4]

$$\left\langle A, B \right\rangle_F = \sum_{i=1}^{m} \sum_{j=1}^{n} a_{ij} b_{ij} \tag{2.4}$$

Das Ziel einer 2-dimensionalen Faltschicht und deren Filter ist es, Muster und deren Position in einem Bild zu erkennen. Dafür braucht es für jedes Muster, das erkannt werden soll, ein eigenes Filter und Gewichte. Die Größe der zu erkennenden Muster beeinflusst direkt die Größe der Filter.

Eine weitere Stellschraube bei Faltschichten ist die Schrittweite. Diese gibt an, wie groß die Sprünge der Filter bei der Faltung sind und haben somit direkten Einfluss auf die Ausgangsgröße der Schicht. Sind diese Sprünge nämlich größer, so wird der Filter in größeren Abständen angelegt und somit auch seltener. Dies führt zu einer kleineren Ausgangsgröße. Eine kleinere Ausgangsgröße bedeutet eine geringere Weiterverarbeitungszeit, jedoch können dabei auch wichtige Informationen verloren gehen.

2.3.4 Lernprozess eines Neuronalen Netzes

Der Aufbau eines Neuonalen Netzes ist nun klar, die Frage ist jetzt aber: Wie lernt so ein Netz? Es kann zwar nun aufgrund einer Eingabe eine Ausgabe errechnen, aber ist das Ergebnis dann auch richtig und wenn nicht, wie lernt es dies?

Das Geheimnis: die **Gewichte** der Neuronen. Diese Gewichte sind die Stellschrauben des Neuronalen Netzes. Durch verschiedene Lernarten (siehe Kapitel 2.4) lassen sich diese

Gewichte "trainieren". [23] Sie halten die Informationen, wie die KI reagieren soll und werden beim Kreieren des Netzes zufällig initialisiert.

2.4 Lernarten

Ein Neuronales Netz lernt durch das Anpassen der Gewichte (siehe Kapitel 2.3.4). Doch wie passt man diese Gewichte so an, dass das Neuronale Netz die gewünschten Ausgaben auch auf noch unbekannte Eingaben berechnen kann?

In Kapitel 2.2.1 wurde bereits erklärt, dass es zwei Hauptarten beim Maschinellen Lernen gibt, um eine KI zu trainieren. Zum einen das überwachte Lernen mit Datenpaaren und zum anderen das unüberwachte Lernen mit einfachen Daten. [21] Es gibt jedoch in beiden Kategorien viele verschiedene Methoden. Zudem gibt es Methoden, die in keine oder beide Kategorien fallen. Bekannte Methoden sind zum Beispiel der Decision Tree oder das Reinforcement Learning.

Man kann zwar oftmals ein Problem mit sehr vielen unterschiedlichen Lernvorgehensweisen lösen, doch haben sich über die Jahre bestimmte Vorgehensweisen für bestimmte Problemstellungen hervorgehoben. Der Vorteil dieser Vorgehensweisen im Gegensatz zu anderen ist fast immer derselbe: die Geschwindigkeit. Es gibt Lernarten, die bei manchen Problemen einfach schneller zu einem Ergebnis kommen als andere.

Beim Trainieren einer KI gibt es dieselben Module, die immer ihre Aufgabe ausführen. Die *Umgebung* ist dazu da, das Geschehen zu simulieren und auszuführen. Sie kann so komplex sein wie das echte Leben oder so simple wie ein Videospiel und interagiert mit dem *Agenten* der KI. Dieser Agent interagiert und führt *Aktionen* aus, die von der KI bestimmt werden. Sie bekommt mit jeder Aktion die sie ausführt eine neue *Beobachtung* von der Umgebung zurück. Diese Beobachtung gibt den aktuellen Zustand an. Eine Beobachtung kann jedoch in vielen unterschiedlichen Formen kommen. Sie kann aus einem Bild bestehen, das auf dem Bildschirm ausgegeben wird, einer Audioaufnahme, die abgespielt werden kann oder sogar eine Kombinationen aus mehreren Formen.

2.5 Lernarten für Videospielen

Für das Kreieren einer KI zum Spielen eines Videospiels ist bisher das Reinforcement Learning (deutsch Bestärkendes Lernen) die beliebteste Methode. Dabei interagiert ein sogenannter "Agent" mit der Umgebung und bekommt von dieser eine Belohnung (engl. reward) für gute Resultate. Dazu bekommt der Agent von der Umgebung eine Beobachtung. [19] Anschließend kann er dann mit der Umgebung durch eine Aktion interagieren und bekommt daraufhin eine neue Beobachtung und eine Belohnung zurück (siehe Abbildung 2.13). Die Belohnung ist hierbei der interessante Teil. Diese sagt aus, ob die Aktion, die der Agent für seine Beobachtung gemacht hat, gut oder schlecht war. Somit kann er

lernen, welche Aktionen für welche Beobachtungen das Optimum an Belohnungen herausholen kann.

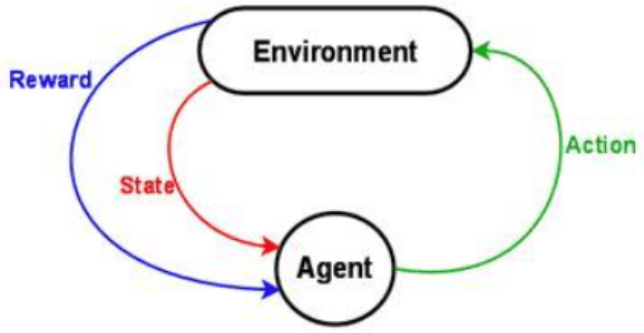


Abbildung (2.13): Interaktion zwischen Agent und Umgebung [16]

Eine Belohnung kann sich von Spiel zu Spiel und von Aufgabe zu Aufgabe sehr unterscheiden. So kann der Agent dafür belohnt werden, wenn er bei Space Invaders einen Gegner abschießt oder bei Pac-Man eine Pille frisst. Er kann aber auch bei beiden Spielen eine Belohnung bekommen, falls er am Leben bleibt und nicht stirbt. Je nachdem, wie diese Belohnungen aussehen, können verschiedene Verhalten bei einem Agenten trainiert werden.

In den letzten Jahren hat sich jedoch noch eine weitere Vorgehensweise herausgehoben. Evolutionäre Algorithmen wurden früher als langsam abgestempelt und sind somit aus dem Fokus gerückt. Doch durch die schier unbegrenzte Rechenleistung wurden sie wiederentdeckt. Zwar ist das Bestärkende Lernen und dessen Strategien auf einer begrenzteren Umgebung effizienter, es wird jedoch durch die einfache Skalierbarkeit der Evolutionären Algorithmen übertroffen. Heutzutage müssen die Berechnungen nicht alle an einer Maschine gemacht werden und dies ist der große Vorteil. [22] Viele Methoden an sich haben sich seit damals nicht wirklich verändert, einzig und allein die Aufteilung auf viele Maschinen bringt einen Geschwindigkeitsvorteil gegenüber dem Bestärkenden Lernen. Somit brauchen Evolutionäre Algorithmen (EA) zwar immer noch mehr Iterationen als Bestärkendes Lernen (RL), diese können jedoch durch die Parallelisierung schneller ausgeführt werden.

Man könnte diese Parallelisierung zwar auch bei RL anwenden, dies würde aber nicht den gleichen Erfolg haben wie bei EA. Das Problem der Parallelisierung im RL ist der hohe Kommunikationsaufwand zwischen den einzelnen Maschinen, denn es müssen beim RL mehr Daten hin- und hergeschickt werden als bei EA. [22] Wie zuvor schon beschrieben, ist das wichtigste bei so einem Algorithmus die Geschwindigkeit. Und dabei ist nicht entscheidend, wie lange der Agent in der simulierten Umgebung für seine Aufgabe braucht, sondern wie viel Realzeit das komplette Training benötigt.

3 Evolutionäre Algorithmen

3.1 Funktionsweise

Evolutionäre Algorithmen haben Konzeptionell ihren Ursprung in der Natur. Dabei gibt es drei Grundkonzepte, die von der Natur in die maschinelle Welt übernommen worden sind. Diese drei Konzepte sind die Vererbung(Heredity), die Variation(Variation) und die Selektion(Selection).[23]

Vererbung(Herdity)
: Bei der Vererbung geht es darum, dass die Eltern ihre Eigenschaften (Erbgut) an ihre Kinder weitergeben können. Wenn Individuen lange leben, können sich diese dann reproduzieren und Nachfahren zeugen.

Variation(Variation)
: Variation muss gegeben sein, damit der Genpool der Individuen groß und vor allem sehr vielseitig ist. Wenn alle Individuen exakt gleich sind, werden auch deren Nachfolger exakt gleich sein und können sich somit nicht weiterentwickeln. Wenn es kein Wesen gibt, das eine rote Haarfarbe hat, dann kann es in Zukunft auch kein Wesen mit roter Haarfarbe geben.

  Variation kann aber nicht nur durch eine ursprünglich große und vielseitige Population gegeben sein. Es gibt noch eine zweite Möglichkeit gegen eine Stagnation der Evolution zu sorgen und dies ist die Mutation. Die Eigenschaften von Kindern werden zwar vorerst aus einer Kombination der Eigenschaften der Eltern gebildet, aber hin und wieder ist es möglich, dass Kinder neue oder veränderte Eigenschaften ganz alleine durch Mutation entwickeln. Somit bleibt der Genpool vielseitig und alle Individuen besitzen unterschiedliche Eigenschaften.

Selektion(Selection)
: Die Selektion ist ein Mechanismus, der bestimmt, welche Individuen überleben und sich weiterentwickeln können. Sie gibt an welche Individuen ihre Eigenschaften und ihr Erbgut an die Kinder weitergeben können und welche nicht. Dieses Konzept wird oftmals als "Überleben des Stärkeren" bezeichnet, die englische Bezeichnung "survival of the fittest" ("Überleben des Anpassungsfähigsten") ist aber eigentlich die bessere Beschreibung, da es nicht immer darum geht, wer der Stärkste ist, sondern welches Individuum sich besser anpassen kann. In einem Faustkampf zwischen einem Menschen und einem Bären würde der Mensch normalerweise keine Chance haben, jedoch hat sich der Mensch dahin entwickelt, Werkzeuge zu verwenden, um somit

sein Überleben auf der Erde zu sichern. Es gibt also verschiedene Möglichkeiten ein Individuum als gut oder fit zu bezeichnen, dies wird oftmals vom Lebensraum des Individuums definiert.

3.2 Typischer Ablauf

Die Implementierung der drei Grundkonzepte, beschrieben im vorherigen Kapitel, kann auf viele verschiedene Möglichkeiten realisiert werden, jedoch folgen diese Implementierungen immer demselben Ablauf. [2]

1. Im ersten Schritt wird eine Population von N Individuen erstellt. Hier wird der erste Teil des Konzepts der Variation angewandt. Dafür muss diese Population groß und vielseitig in ihren Eigenschaften sein, um die Evolution nicht zu verhindern.
2. Anschließend werden alle Individuen der Population ausgewertet und ihre *Fitness* berechnet. Dieser Wert bestimmt, wer sich durch Reproduktion weiter entwickeln darf. Dies ist das Konzept der Selektion.
3. Der letzte Schritt ist die Reproduktion, sie läuft folgendermaßen ab:
   a) Zwei Eltern werden unter Berücksichtigung der *Fitness*-Werte ausgewählt.
   b) Durch Reproduktion wird ein "Kind" erzeugt, das eine Mischung aus den Eigenschaften der Eltern ist.
   c) Nun können durch Mutation noch einzelne Eigenschaften des Kindes verändert werden. Hier wird der zweite Teil des Konzeptes der Variation angewandt.
   d) Das neue Individuum (Kind) wird zu einer neuen Population hinzugefügt.
4. Nun wird die alte Population durch die neue Population ausgetauscht und es wird erneut bei Schritt 2 angefangen.

Abbildung 3.1 zeigt diese einzelnen Schritte noch einmal in einem Flussdiagram dargestellt.

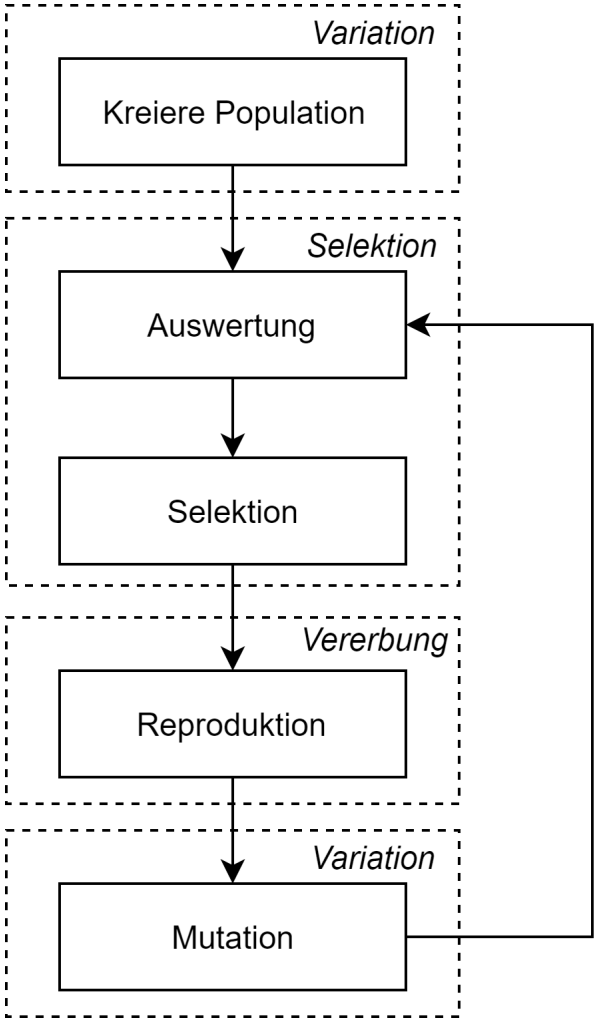


Abbildung (3.1): Ablauf eines Evolutionären Algorithmus

3.3 Prinzipien

Im Folgenden werden weitere Prinzipien der Evolution erklärt. Diese sind teilweise konkreter als davor.

3.3.1 Erbgut gegen Erscheinungsform

Was macht eine Eigenschaft eines Individuums aus? Eine Eigenschaft wird in zwei Teile aufgeteilt: das Erbgut und die Erscheinungsform.

Das Erbgut ist der Ursprung der Information über die Eigenschaft. Damit werden die eigentlichen Daten beschrieben, die eine Eigenschaft ausmachen. [23] Zum Beispiel können Daten im Erbgut als einfache Zahl gespeichert werden. Diese Zahl kann dann als Erscheinungsbild eine Farbe im Grauwert sein.

Erbgut	Erscheinungsform 1	Erscheinungsform 2
int c = 255;	weiß	255 cm
int c = 127;	grau	127 cm
int c = 0;	schwarz	0 cm

Tabelle (3.1): Unterschied zwischen Erscheinungsformen bei gleichem Erbgut [23]

Diese Zahl, die im Erbgut gespeichert ist, kann nun aber auch eine andere Erscheinungsform haben. (siehe Tabelle 3.1) So kann das gleiche Erbgut eine andere Erscheinungsform haben.

3.3.2 Fitness

Der Fitness-Wert eines Individuums gibt an, wie gut es in der Umgebung, in der es ist, zurechtkommt. [22] Dabei kommt es nicht nur auf das Individuum an sich an, sondern auch insbesondere, welcher Umgebung es ausgesetzt ist. Nimmt man zum Beispiel einen Fisch und setzt ihn in einen Ozean, dann kommt er dort sicherlich gut zurecht und kann lange überleben und sich vermehren. Nimmt man nun aber den gleichen Fisch, der so exzellent im Wasser zurechtkommt, setzt ihn ans Land, dann wird er dort nicht lange überleben und kann sich auch nicht vermehren.

Mit dem Fitness-Wert kann auch ”eingestellt” werden, wie sich die Individuen verhalten sollen. Dies kann dadurch gemacht werden, dass unterschiedliche Aspekte unterschiedlich stark in den Wert einfließen. In der gleichen Umgebung kann ein Individuum darauf trainiert werden, so lange wie möglich zu überleben und ein anderes Individuum wiederum darauf, so viele Punkte wie möglich zu erreichen.

3.3.3 Selektion

Bei der Selektion geht es darum, Individuen für den Paarungspool der nächsten Generation auszuwählen. Der Paarungspool ist eine Ansammlung von Individuen, die für die Reproduktion der nächsten Generation verwendet werden. Dabei ist das Hauptkriterium, um für den Paarungspool ausgewählt zu werden, der Fitness-Wert, den jedes Individuum erzielt hat. Je höher dieser Wert ist, desto besser hat sich das Individuum geschlagen, also desto höher sollte auch seine Wahrscheinlichkeit sein, in den Paarungspool aufgenommen zu werden. [29] Es wird hier deswegen von Wahrscheinlichkeit gesprochen, weil auch der Zufall bei der Auswahl eine Rolle spielen muss. Das Ziel dabei ist es nicht, den Zufall zu verwenden, nur um das Vorgehen der Natur widerzuspiegeln, sondern um zu verhindern, dass sich die Evolution in einem lokalen Maximum der Fitness fest fährt und nicht mehr weiter entwickelt, da sich alle neu gebildeten Individuen nicht genug unterscheiden, um bessere Ergebnisse zu erzielen (siehe Abbildung 3.2). Befindet sich die Evolution im lokalen Maximum links in der Abbildung, so kann es geschehen dass kein Nachfahre sich

genug vom Original unterscheidet (weiter links oder rechts auf der x-Achse vom lokalen Maximum aus) entwickelt, welches ein besseres Ergebnis liefert. Alle Nachfahren sind zu nah am Original und kommen nicht aus der Erhebung, das lokale Maximum, heraus.

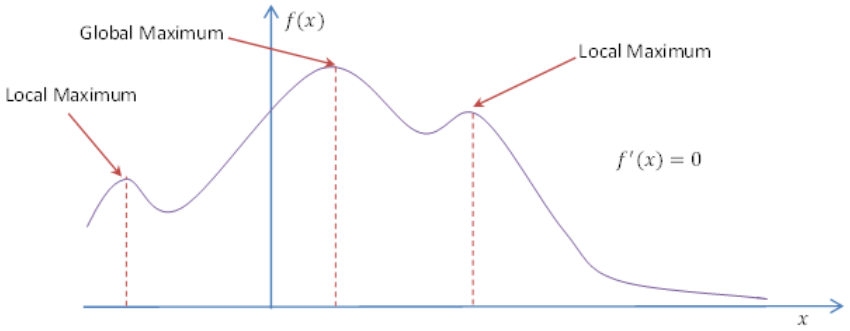


Abbildung (3.2): Funktion mit lokalem und globalem Maximum [7]

Im Sinne der Evolution möchte man durch diesen Zufall verhindern, dass wichtige Eigenschaften für die Evolution verloren gehen. Eigenschaften, die nur bei schwächeren Individuen auftreten, sind nicht unbedingt schlecht. Es kann sein, dass genau diese Eigenschaften bei stärkeren Individuen zu einem noch größeren Erfolg führen könnten. Würden diese Eigenschaften zu früh eliminiert, also aus dem Genpool entfernt, könnte es somit zu einem Stillstand in der Evolution kommen.

3.3.4 Reproduktion

Die Reproduktion ist der vorletzte Schritt für jede Generation. Dabei wird aus dem von der Selektion ausgewählten Paarungspool eine neue Generation gebildet.

In der Natur geschieht die Reproduktion in der Regel durch zwei Individuen, den Eltern. Dabei entsteht ein neues Individuum, ein Kind, welches Eigenschaften von beiden Eltern aufweist. Es kann aber auch ”neue”, gemischte Eigenschaften aufweisen, die kein Elternteil alleine aufweisen kann und die erst durch Reproduktion entstanden sind.

Bei maschinellen Individuen kann die Reproduktion genauso erfolgen wie in der Natur. Man nimmt die Eigenschaften von zwei Individuen und mischt diese zu einem neuen Individuum. [23] Jedoch kann man bei maschinellen Individuen oftmals nicht sagen, was genau eine Eigenschaft eines Individuums ist. Dies macht das Kreuzen zweier Individuen zu aufwändig.

Stattdessen wird nur ein Individuum zur Reproduktion verwendet. Das Kind, das dadurch entsteht, ist eine Kopie des einzigen Elternteils. Es hat alle Eigenschaften, die auch das Elternteil hat und würde somit den exakt gleichen Fitness-Wert erzielen. Dieser Prozess würde einfach nur schlechte Individuen herrausfiltern und somit zwar den Durchschnitt der Population anheben, aber es kommt zu keiner Evolution, da Individuen nie besser werden können als das aktuell beste Individuum.

Um die Evolution gewährleisten zu können, müssen sich die Kinder von ihren Eltern unterscheiden. Dies entsteht in diesem Fall durch eine Mutation. Die Mutation sorgt dafür, dass Kinder nicht nur eine Kopie ihrer Eltern sind und somit auch andere (bessere) Ergebnisse liefern können als diese.

3.3.5 Mutation

Mutation ist vor allem dann ein wichtiger Punkt, wenn die Reproduktion von nur einem Individuum vollführt wird. Ohne Mutation würde es dadurch keine Veränderung geben und somit auch keine Evolution. Aber auch bei "normaler" Reproduktion wird Mutation häufig verwendet. Das eigentliche Ziel von Mutation ist es, neue Eigenschaften zu finden und in den Genpool aufzunehmen. [23]

Eine Mutation kann auf verschiedene Arten geschehen. Zum einen kann eine Mutation erfolgen, indem bei einer Eigenschaft eine kleine (zufällige) Veränderung vorgenommen wird. Zum Beispiel kann die Haarfarbe eines Kindes heller sein, als die der Eltern.

Eine weitere Möglichkeit ist es, eine Eigenschaft von Grund auf zu mutieren. Dies bedeutet, der Eigenschaft einen zufälligen, neuen Zustand zu geben. So können zum Beispiel die Eltern braune Augen haben und das Kind kann durch diese Mutation blaue Augen entwickeln.

Die dritte Möglichkeit einer Mutation ist es, eine komplett neue Eigenschaft zu entwickeln, die davor bei keinem Elternteil vorhanden war. Diese Art von Mutation kommt in der Natur eher sehr selten vor. So kann es vorkommen, dass Kinder mit einem zusätzlichen Finger an einer Hand geboren werden.

Mutation ist wichtig, um Variation zu garantieren. Bei der Ursprünglichen Kreierung der Population können Individuen mit völlig verschiedenen Eigenschaften erstellt werden, dabei kann jedoch nicht garantiert werden, dass alle nötigen Eigenschaften zum Lösen des Problems in diesem ursprünglichen Genpool vorhanden sind. Mutation soll dieses Problem durch die zufällige Veränderung und Neuerschaffung von Eigenschaften ermöglichen.

3.4 Evolutionäre Strategien

3.4.1 Erklärung

Evolution Strategie (ES) ist eine spezielle Implementierung des Evolutionärer Algorithmus und gehört zu den Blackbox Optimierungsalgorithmen [29]. Das heißt, es ist nichts weiter über die Fitness oder einen anderen Aspekt bekannt, als, wie der Fitness-Wert eines Individuums berechnet wird. Das Prinzip ist es, in einer Schleife immer wieder diesen Fitness-Wert für alle Individuen auszurechnen und die Individuen aufgrund ihres Ergebnisses anzupassen.

3.4.2 Spezieller Ablauf

Der Ablauf von Evolutionären Strategien ist spezieller als der Ablauf von Evolutionären Algorithmen. Es wird jedoch keine Ursprungpopulation erstellt, sondern nur ein einziges Individuum, das Hauptindividuum. Dieses Hauptindividuum ist das, das trainiert wird. Dazu werden in einer Schleife immer wieder neue Populationen aus diesem Hauptindividuum generiert und ausgewertet. [22] Folgende Schritte beschreiben diesen Ablauf:

1. Ein Individuum wird mit zufälligen Eigenschaften erstellt. Dies ist das Hauptindividuum, das sich mit der Zeit verbessern und weiterentwickeln soll.
2. Eine Population wird aus dem Hauptindividuum erstellt. Alle Individuen dieser Population sind eine Kopie davon mit jeweils unabhängigen, zufälligen Mutation.
3. Alle Individuen der Population werden ausgewertet und deren Fitness-Wert berechnet.
4. Die Mutationen der Individuen werden nun mit deren errungenem Fitness-Wert gewichtet. Alle vertetenen Mutationen werden dann zu einer "Durchschnitts-Mutation" zusammengerechnet.
5. Diese "Durchschnitts-Mutation" wird nun am Hauptindividuum durchgeführt.
6. Dieses neue Hauptindividuum bildet die Grundlage für eine neue Population und es wird von Schritt 2 an neu angefangen.

```
1   # simple example: minimize a quadratic around some solution point
2   import numpy as np
3   solution = np.array([0.5, 0.1, -0.3])
4   def f(w): return -np.sum((w - solution)**2)
5
6   npop = 50      # population size
7   sigma = 0.1    # noise standard deviation
8   alpha = 0.001  # learning rate
9   w = np.random.randn(3) # initial guess
10  for i in range(300):
11      N = np.random.randn(npop, 3)
12      R = np.zeros(npop)
13      for j in range(npop):
14          w_try = w + sigma*N[j]
15          R[j] = f(w_try)
16      A = (R - np.mean(R)) / np.std(R)
17      w = w + alpha/(npop*sigma) * np.dot(N.T, A)
```

Listing (3.1): Code Beispiel einer Implementiereung einer Evoltuionären Strategie von OpenAI [22]

Das Listing 3.1 zeigt eine Implementierung der Evolutionären Strategie von OpenAI in Py-

thon. Hier ist das Ziel, die Minima einer quadratischen Funktion zu finden. Dieses Ziel ist in der Fitness-Funktion definiert. Sie gibt den Fitness-Wert für jedes Individuum zurück. Der Algorithmus selber braucht nur vier Parameter. Dies ist zum einen die Populationsgröße *npop* (Zeile 6), die aussagt wie viele Individuen es pro Generation gibt. Weiterhin braucht es noch eine Mutationsrate *sigma* (Zeile 7). Dies ist die Standardabweichung und gibt an, wie stark die Mutationen sein können. Als drittes braucht es ein *alpha*, welches die Lernrate ist (Zeile 8). Diese gibt an, wie stark die Durchschnitts-Mutation am Hauptindividuum vorgenommen werden soll. Und als letztes braucht es noch einen Anfangszustand des Hauptindividuums, normalerweise beschrieben durch die Gewichte des Neuronalen Netzes (siehe Kapitel 3.5). Zuerst wie ein zufälliges Hauptindividuum erstellt, in diesem Fall ist dieses eine Liste mit drei Gewichten (Zeile 9). Dies ist das Model das trainiert wird. Anschließend wird in einer Schleife eine neue Population aus dem Hauptindividuum gebildet (Zeile 11), diese sind Momentan noch exakte Klone. In einer weiteren Schleife wird jedes einzelne Individuum ausgewertet (Zeile 13 bis 15). Dazu wird zuerst eine zufällige Mutation angewendet (Zeile 14) und danach mithilfe der Fitness-Funktion dessen Fitness-Wert (Zeile 15). Nachdem die Fitness jedes einzelnen Individuums der Population berechnet wurde, wird die Durchschnittsmutation berechnet und am Hauptindividuum angewendet (Zeile 16 und 17).

Der Algorithmus der Evolutionären Strategie lässt sich aber auch grafisch verdeutlichen. [22] Die Abbildung 3.3 zeigt dies für ein Problem mit nur zwei Parametern (z.B. Gewichte eines Neuronales Netzes). Die Abbildung zeigt den Zustand des Algorithmus anhand vier aufeinander folgender Schritte. Außerdem zeigt es für jede Parameter-Kombination den dazugehörigen Fitness-Wert (rot = hoch, blau = niedrig). Dies ist die Fitness-Funktion. Im ersten Schaubild ist das Hauptindividuum mit einem weißen Punkt gekennzeichnet. Die schwarzen Punkte stellen die einzelnen Individuen der Generation dar. Diese sind durch ihre zufälligen Mutationen kleine Veränderungen der zwei Parameter. Grafisch gesprochen sind sie in einem kleinen Bereich um das Hauptindividuum. Die daraus resultierende Durchschnitts-Mutation ist mit einem weißen Pfeil eingezeichnet. Dieser Pfeil gibt jetzt an, in welche ”Richtung” diese Mutation geht und wie stark sie ist. Das Schaubild zeigt das Hauptindividuum mit dieser Mutation und einer neuen Population. So kann Generation für Generation eine Richtung berechnet werden, in der der Fitness-Wert immer höher ist.

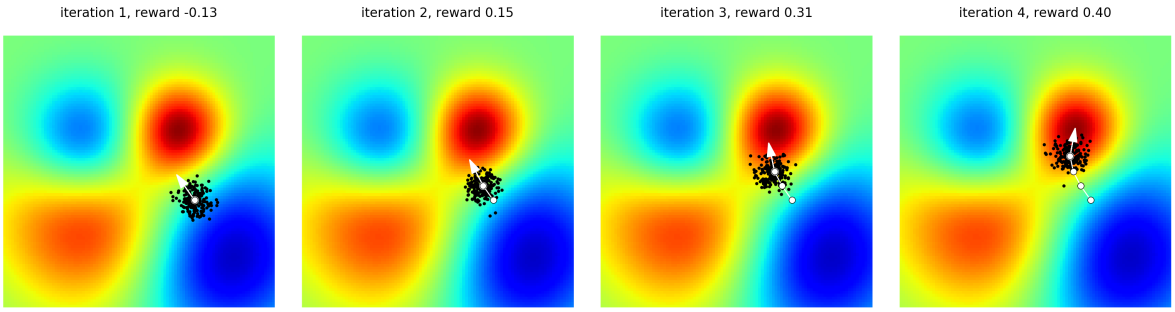


Abbildung (3.3): Evolutionäre Strategie am Beispiel eines Problems mit 2 Parametern [2]

Abbildung 3.3 zeigt dies zwar nur für ein einfaches Problem mit zwei Parametern, aber das Ganze funktioniert für beliebig viele Parameter. Nur die grafische Darstellung ist für den Menschen ab drei Parametern (drei Dimensionen) schwierig nachzuvollziehen. Jedoch kann man auch dort von einer "Richtung" sprechen, in die sich die Individuen entwickeln. Diese Richtung hat aber dann nicht mehr zwei Dimensionen, wie in Abbildung 3.3, sondern Tausend oder gar Millionen Dimensionen.

3.5 Evolution mit Neuronalen Netzen

In Kapitel 2 wurde bereits gezeigt, wie es möglich ist, einer Maschine ein "Gehirn" zu geben und es somit zum "Denken" anzuregen. Kapitel 3 hat gezeigt, wie man mithilfe von Evolution einen Algorithmus entwickelt, der es ermöglicht, Individuen dazu zu bringen, ein spezielles Problem zu lösen. Dieser Evolutionsalgorithmus kann nun zum Training einer KI verwendet werden. [23]

Dazu müssen aber noch einige Konzepte der Evolution im Zusammenhang eines Neuronalen Netzes, des Gehirns der KI, definiert und erklärt werden. Der wichtigste Punkt dabei ist, dass eine KI in diesem Fall **nur** durch ihr "Gehirn", das Neuronale Netzwerk, definiert ist. Es spielt dabei keine Rolle, in was für einer Computerarchitektur sie sich befindet, solange die Umgebungen und die Interaktion mit der Umgebung dieselbe ist.

Eigenschaft

In der Evolution wurde sehr oft von einer Eigenschaft gesprochen und wie diese das Individuum und sein Verhalten ausmachen. Dazu gehören Körpereigenschaften, wie z.B. die Größe einzelner Glieder oder die Farbe von Haaren.

Bei einem Gehirn und dessen Denkprozessen sind Eigenschaften nicht physisch, sondern abstrakt. Dazu gehört bei einem Menschen zum Beispiel das Lesen oder Hören. [24] Im Gehirn geht es vor allem darum, Daten von den Sinnen zu interpretieren. Führt dies zu einer richtigen Annahme, dann spricht man von der Fähigkeit oder Eigenschaft etwas zu tun.

Auch bei einem Neuronalen Netz geht es, ähnlich wie bei einem echten Gehirn, nicht darum, wie es aufgebaut ist (Netzwerkarchitektur), sondern welche Ergebnisse es liefern kann. Wie bei einem Gehirn sind verschiedene Teile des Netzwerkes für verschiedene Eigenschaften verantwortlich. Welche Teile letztendlich für welche Eigenschaft verantwortlich sind, ist schwer zu sagen. Jedoch sind gleiche Schichten für gleiche Eigenschaften verantwortlich (siehe Kapitel 2.3.2).

Erbgut und Erscheinungsform

Das Erbgut eines Neuronalen Netzes besteht aus zwei Bestandteilen, diese zwei beschreiben ein Neuronales Netzwerk vollständig. Der erste Bestandteil ist die Netzwerkarchitektur. Diese beschreibt, wie viele Schichten ein Netzwerk hat und wie es untereinander verbunden ist. Der zweite Bestandteil ist die Gewichtung der einzelnen Neuronen. Dies beschreibt das Erbgut vollständig, d.h. dass alle Netze mit gleicher Netzwerkarchitektur und Gewichtung exakt gleich sind.

Reproduktion

Bei der Reproduktion geht es darum, ein neues Individuum aus einem oder mehreren Individuen zu bekommen. Da alle Individuen bei der Maschine nur durch ihr Neuronales Netz definiert sind, ist dies das Einzige, was bei der Reproduktion vererbt werden muss.

Bei Evolutionären Algorithmen gibt es eine feste Netzwerkarchitektur. Alle Individuen haben somit die gleiche Anzahl und Art an Schichten und die gleiche Neuronenanzahl. Die einzige Sache, die von den Elternteilen vererbt wird, sind die Gewichtungen der einzelnen Neuronen. Wie viele und welche Gewichte von welchem Elternteil vom Kind übernommen werden, ist dabei komplett frei. Dadurch entsteht bei den Kindern eine Mischung aus Gewichten.

Die Reproduktion bei Evolutionären Strategien funktioniert mit nur einem einzigen Elternteil. Das daraus resultierende Kind ist somit eine Kopie. In diesem Fall kann also das ganze Neuronale Netz mit all seinen Gewichten genommen und kopiert werden. Da es nur einen Elternteil gibt, muss keine Mischung an Gewichten geschehen.

Mutation

Eine Mutation in einem Neuronalen Netz kann auf verschiedene Arten geschehen. Die einzige Bedingung einer Mutation in einem Neuronalen Netz ist, dass sich das Erbgut verändert. Dies kann auf verschiedene Weisen erzielt werden.

Zum einen kann dies durch eine Veränderung der Netzwerkarchitektur erzielt werden. Das heißt, neue Schichten hinzufügen, bestehende Schichten entfernen, die Anzahl an Neuronen in den einzelnen Schichten verändern oder andere Aktivierungsfunktionen verwenden. Es gibt viele Möglichkeiten eine Mutation an der Netzwerkstruktur zu verwirklichen.

Die zweite und häufiger verwendete Möglichkeit eine Mutation zu erzielen, ist das Verändern der Gewichte der Neuronen. Eine kleine Veränderung der Gewichte kann das komplette Resultat des Neuronalen Netzes verändern. Da bei Evolutionären Strategien die Architektur des Netzes normalerweise gleich bleibt, werden hier nur die Gewichte verändert.

4 Anforderungen

Anforderungen (engl. Requirements) stellen Forderungen und Richtlinien an die Funktionalität und Qualität eines Systems oder Software. Sie sind die Bedingungen und Wünsche des Kunden an den Entwickler.

Anforderungen werden hierbei bezüglich ihrer Verbindlichkeit in vier verschiedene Klassen eingeordnet. Es wird unterschieden zwischen einem Wunsch (”soll”), einer festen Absicht (”wird”), einem Vorschlag (”kann”) und einer Pflicht (”muss”) (nach [20]).

4.1 Funktionale Anforderungen

In diesem Abschnitt werden die funktonalen Anforderungen an das Systems aufgelistet. Diese beschreiben welche Funktionalitäten ein System haben muss und was es leisten soll. Dazu werden zuerst die Top-Level Anforderung und anschließend die Anwendungsfälle (engl. use-cases) und deren Anforderungen aufgezeigt.

4.1.1 Top-Level Anforderung

Requirement 1000:
Es soll eine Künstliche Intelligenz entwickelt werden, welche nach einem Training, ein Spiel lösen kann.

4.1.2 Geforderte Anwendungsfälle

Requirement 1100:
Die KI soll folgende Funktionalitäten erfüllen:

- Das KI-System soll über mehrere Generationen seinen Fitness-Wert steigern können. (Anwendungsfall ”KI trainieren”)
- Das KI-System soll soweit wie möglich im Spiel kommen und somit ihren Fortschritt maximieren. (Anwendungsfall ”KI ausführen”)

4.1.3 Anforderungen an die Anwendungsfälle

Anwendungsfall ”KI trainieren” und ”KI ausführen”:

- **Requirement 1110:**
  Das KI-System muss Eingaben an das Spiel senden können.
- **Requirement 1111:**
  Das KI-System muss die Bildschirmausgabe des Spieles auslesen können.
- **Requirement 1112:**
  Das KI-System soll den aktuellen Spielzustand ermitteln können.
- **Requirement 1113:**
  Das KI-System soll einzelne Trainingssessions laden können.

Anwendungsfall ”KI trainieren”:

- **Requirement 1120:**
  Das KI-System muss die Fitness eines Individuums berechnen können.
- **Requirement 1121:**
  Das KI-System soll Statistiken über ihren Fortschritt erstellen können.
- **Requirement 1122:**
  Das KI-System soll einzelne Trainingssessions speichern können.

4.2 Nichtfunktionale Anforderungen

In diesem Abschnitt werden alle Nichtfunktionalen Anforderungen aufgezeigt. Hierzu gehören Anforderungen an die Qualität und die Implementierung des Systems. Diese beschreiben wie das System funktioniert. Dazu gehören zum Beispiel Performance oder Architektur des Systems.

4.2.1 Anforderungen an die Implementierung

- **Requirement 2100:**
  Die KI soll mit einer Umgebung von OpenAI Retro trainieren und spielen können.
- **Requirement 2101:**
  Das KI-System soll ein Neuronales Netzwerk verwenden um Entscheidungen zu treffen.
- **Requirement 2102:**
  Die KI soll in der Programmiersprache Python geschrieben werden.
- **Requirement 2103:**
  Das Neuronale Netz der KI soll in Tensorflow und Keras implementiert werden.
- **Requirement 2104:**
  Das KI-System soll die Evolutionäre Strategie verwenden.

5 Systemanalyse

In der Systemanalyse werden die Grenzen des Systems aufgezeigt. Die Anforderungen werden zuerst in einem Anwendungsfalldiagramm in Kapitel 5.1 grafisch dargestellt und anschließend in Kapitel 5.2 und Kapitel 5.3 näher beschrieben.

5.1 Anwendungsfalldiagramm

Abbildung 5.1 zeigt das Anwendungsfalldiagramm des KI-Systems mit den beiden Anwendungsfällen ”KI trainieren” und ”KI ausführen”.

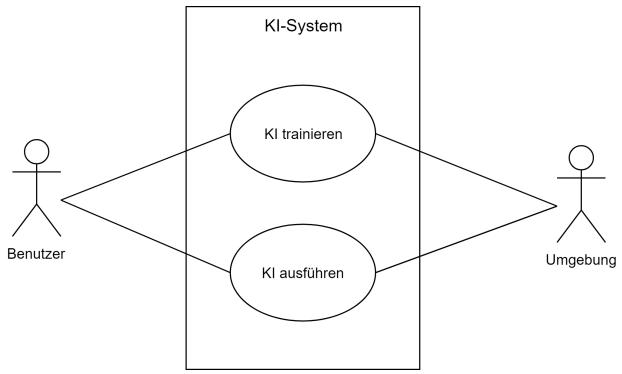


Abbildung (5.1): Anwendungsfalldiagramm der Künstlichen Intelligenz

5.2 Kurzbeschreibung der Anwendungsfälle

Im Folgenden werden die beiden Anwendungsfälle ”KI trainieren” und ”KI ausführen” der Künstlichen Intelligenz kurz beschrieben. Dabei ist es wichtig zu verstehen, dass unter einer Trainingssession ein Training mit einer einzigen speziellen KI vollführt wird (siehe 7.5).

KI trainieren
Beim Training wird eine neue Trainingssession erstellt oder eine schon vorhandene Trainingssession geladen. Anschließend wird das Training gestartet. Hierzu wird die Evolutionäre Strategie angewendet und in mehreren Generation die Künstliche Intelligenz verbessert. Während des Trainings wird die Trainingssession immer wieder gespeichert.

KI ausführen
Die Künstliche Intelligenz lädt eine Trainingssession und versucht das Spiel mit einem maximalen Fortschritt abzuschließen. Hierbei verwendet die KI ihr bisher bestes Neuronales Netzwerk als "Gehirn".

5.3 Langbeschreibung der Anwendungsfälle

In diesem Abschnitt werden die beiden Anwendungsfälle "KI trainieren" und "KI ausführen" länger und ausführlicher beschrieben.

Anwendungsfall "KI trainieren"

Initiator:	Benutzer
Beteiligte Akteure:	KI-System
Basisablauf:	Der Benutzer kreiert oder lädt eine Trainingssession in die Künstliche Intelligenz. Dabei wird das aktuelle (oder neue) Neuronale Netz und dessen Trainingsparameter geladen. Anschließend wird das Training gestartet. Die aktuellen Gewichte des Neuronalen Netzes werden der Evolutionären Strategie übergeben. Diese berechnet einen Generationsschritt und speichert anschließend den neuen aktuellen Stand und zusätzliche Informationen über die Generation ab.

Anwendungsfall "KI ausführen"

Initiator:	Benutzer
Beteiligte Akteure:	KI-System
Basisablauf:	Der Benutzer lädt eine Trainingssession in die Künstliche Intelligenz. Anschließend versucht Sie das Spiel mit dem geladenen Neuronalen Netzwerk zu spielen und dabei einen besonders hohen Fortschritt zu erzielen. Das Neuronale Netzwerk wird währenddessen nicht weiter trainiert.

6 Systementwurf

In diesem Kapitel werden Entwurf und Architektur des Systems vorgestellt. Dazu werden zuerst in Kapitel 6.1 und Kapitel 6.2 externe Abhängigkeiten näher erläutert. Anschließend wird der Ablauf des Gesamtsystems in Kapitel 6.3 aufgezeigt.

6.1 Umgebung mit OpenAI Retro

Die verwendete Umgebung wurde mit Gym-Retro implementiert. Gym-Retro ist eine Wrapper Bibliothek von OpenAI für Videospielemulatoren. Sie wurde speziell für das Maschinelle Lernen konzipiert, indem sie eine einheitliche Schnittstelle zu verschiedenen Videospielkonsolen und deren Spiele liefert. So können damit Spiele als Umgebungen für das Entwickeln und Testen von Künstlichen Intelligenzen dienen.

Gym-Retro unterstützt aktuell Windows, Mac und Linux Systeme und läuft mit Python 3.5, 3.6 und 3.7. Im Moment unterstützte Konsolen sind ältere Konsolen, wie das Nintendo Entertainment System, Atari2600 oder Sega Genesis.

```
1   import retro
2
3   env = retro.make("SuperMarioBros-NES", state="Level1-1")
4
5   obs = env.reset()
6
7   done = False
8   while not done:
9       action = env.action_space.sample()
10      obs, reward, done, info = env.step(action)
```

Listing (6.1): Umgebung mit Gym-Retro

In Listing 6.1 kann man sehen wie eine Umgebung mit Gym-Retro initialisiert und verwendet werden kann. Beim Erstellen der Umgebung in Zeile 3 muss der Namen der Umgebung (Spielenamen) und ein Zustand (State/Level) spezifiziert werden. Dafür braucht Gym-Retro sowohl die ROM (das Spiel) als auch eine State-Datei (*.state), die den aktuellen Zustand gespeichert hat. Außerdem gehört zur Initialisierung noch das Zurücksetzten (engl. reset) in Zeile 5 dazu. Es sorgt dafür, dass das Spiel an der richtigen Stelle gestartet

wird und gibt die erste Beobachtung (obs kurz für engl. observation) zurück. Die Beobachtung ist ein 2-dimensionales Array der Bildschirmausgabe, wie auch der Benutzer sie sehen würde.

Nun kann, wie in Zeilen 7 bis 9 zu sehen ist, durch die Umgebung Schritt für Schritt durchiteriert werden. Dazu nimmt man eine Aktion (engl. action) und gibt sie der Schrittfunktion (engl. step) der Umgebung als Argument. Diese liefert eine neue Beobachtung, eine Belohnung (engl. reward), eine Indikation ob die Umgebung fertig ist (engl. done) und weitere zusätzliche Informationen (info kurz für engl. information).

Die Aktion besteht aus einer Liste aus Boolescher Variablen, die angeben, welche Tasten gedrückt sind und welche nicht. Da die Eingabegeräte der unterschiedlichen Konsolen verschieden sind, haben diese Listen unterschiedliche Längen und Reihenfolge der Tasten. Jedoch ist sie bei allen Spiele der gleichen Konsole exakt gleich. In Zeile 8 wird hierfür eine zufällige Liste genommen. Diese sollte später von der KI kommen.

Die zusätzlichen Informationen die man bekommt, müssen für jede Umgebung einzeln in einer Json-Datei (data.json) definiert werden. Diese Informationen sind zum Beispiel Angaben über Position einer Spielfigur oder die Punktezahl des Spielers. Sie werden normalerweise direkt aus dem Speicher des Emulators ausgelesen.

Auch die Belohnung und Done-Indikation wird aus dem Speicher ausgelesen. Auch dies wird in einer Json-Datei festgelegt (scenario.json) Bei der Belohnung kann spezifiziert welche Werte wie stark in die Berechnung eingehen. Bei der Done-Indikation muss eine Bedingung angegeben werden, welche bestimmt wann diese Variable als Wahr zurückgegeben wird. Meist ist die, wenn das Leben, dass aus dem Speicher ausgelesen wird (data.json) auf 0 gefallen ist.

6.2 Keras und Tensorflow

Für das Neuronale Netzwerk wurden die beiden Bibliotheken ”Keras” und ”Tensorflow” verwendet. Tensorflow ist eine Open-Source-Bibliothek von Google für Maschinelles Lernen [8]. Sie ermöglicht es Neuronale Netzwerke zu bauen und zu trainieren.

Keras ist eine höher liegende Schnittstelle zu maschinellen Lernbibliotheken wie Tensorflow. Sie verwendet im Hintergrund die komplette Implementation von Tensorflow, ermöglicht dem Nutzer aber einen leichteren Einstieg, da er nicht wie bei Tensorflow vieles selber spezifizieren muss, sondern die Voreinstellungen von Keras verwenden kann [12].

```
1         import keras
2         from keras.models import Sequential
3         from keras.layers.core import Dense
4
5         model = Sequential()
6         model.add(Dense(100, input_shape=(100,), activation="relu"))
7         model.add(Dense(100, activation="relu"))
8         model.add(Dense(5, activation="sigmoid"))
9         model.compile(optimizer="sgd", loss="categorical_crossentropy")
10
```

Listing (6.2): Neuronales Netz mit Keras

Listing 6.2 zeigt wie schnell man ein Neuronales Netz mit Keras erstellen kann. Zuerst wird der Typ definiert, in diesem Fall wird in Zeile 5 ein sequentielles Netz gebaut. Dies bedeutet ein Netzwerk, dass von vorne nach hinten in einer Sequenz ausgeführt wird. Anschließend können Schichten hinzugefügt werden. In den Zeilen 6 bis 8 werden drei Fully-connected Schichten hinzugefügt, dabei muss bei jeder Schicht die Anzahl dann Neuronen angegeben werden. Bei der ersten Schicht in jedem Netz muss zusätzlich noch die Größe Eingabe angegeben werden. Außerdem kann man in jeder Schicht angeben, ob und welche Aktivierungsfunktion verwendet werden soll. Keras liefert eine Handvoll und Funktionen die man verwenden kann, es kann aber auch eine eigene definiert werden.

Das Neuronale Netzwerk ist nun funktionsfähig und kann verwendet werden. Für andere Trainingsmethoden muss das Netz noch mit einem optimizer und einer loss-Funktion kompiliert werden (z.B Reinforcement Learning). Dies ermöglicht, dass das Netzwerk auf Ein- und Ausgabewerte trainieren kann.

6.3 Ablaufdiagramme

Im folgenden Abschnitt werden die Prozesse und Funktionen der beiden Anwendungsfälle, ”KI trainieren” und ”KI ausführen”, jeweils in ein Ablaufdiagram aufgezeigt.

Abbildung 6.1 zeigt den Ablauf des Trainings. Hierbei wird zuerst ein neues Training erstellt oder ein vorhandenes Training geladen. Anschließend wird mit den Trainingsparameter und dem Algorithmus der Evolutionären Strategie das Training begonnen. Hierbei wird nach jedem Generationsschritt das Training abgespeichert und zusätzliche Statistiken erstellt. Das Training wird dabei solange fortgesetzt, wie vom Nutzer angegeben.

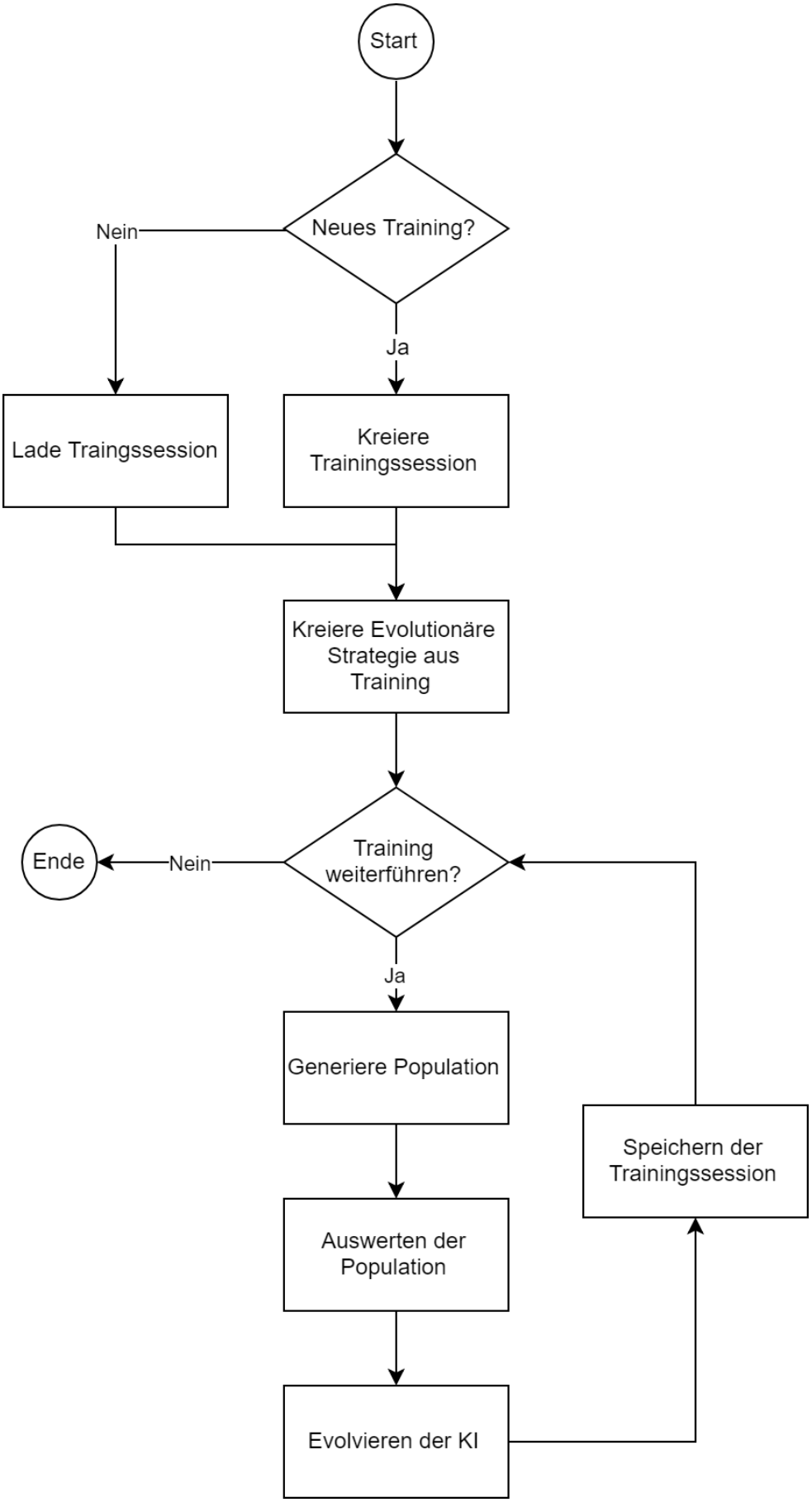


Abbildung (6.1): Ablaufdiagramm des Trainings

Der Ablauf des Anwendungsfalles ”KI ausführen” ist in Abbildung 6.2 zu sehen. Hierbei wird zuerst ein vorhandenes Training geladen. Anschließend wird das Spiel von der KI gestartet und die KI versucht ihr bestes Ergebnis zu erreichen.

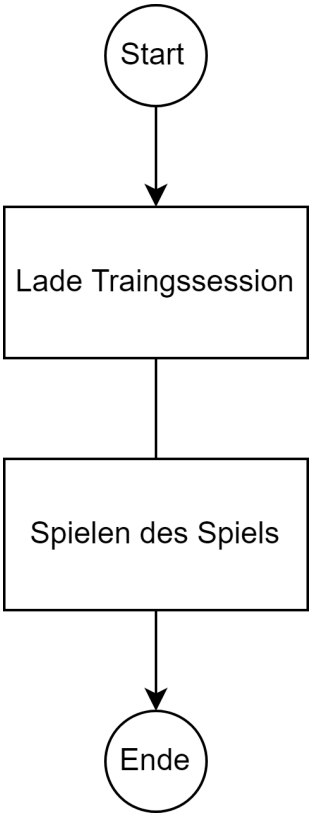


Abbildung (6.2): Ablaufdiagramm des Ausführens

7 Implementierung

In diesem Kapitel wird die Implementation des Systems beschrieben. Dazu werden die einzelnen Module anhand von Code-Beispielen näher erläutert.

Die Implementierung der Komponenten des Systems ist sehr allgemein gehalten. Dies liegt einerseits daran, dass die KI ständig getestet und verändert werden muss und zum anderen ist es das Ziel, ein System zu entwickeln, das in verschiedenen Umgebungen und mit unterschiedlichen Neuronalen Netzwerken funktioniert.

7.1 KI-System

In diesem Kapitel wird gezeigt wie die zwei Anwendungsfälle implementiert werden, welche Komponenten dafür benötigt werden und wie diese funktionieren.

```
1   from es import EvolutionStrategy
2   from model import NeuralNetwork
3   from training_manager import Training
4   from agent import SMBAgent
5
6   # Game parameters
7   new_training = True
8   render = False
9   training_name = "src/Mario-ES/trainingName"
10
11  # Training parameters
12  pop_size = 50
13  sigma = 0.4
14  learning_rate = 0.2
15
16
17  if new_training:
18      training = Training.create(training_name, pop_size, sigma,
    learning_rate)
19  else:
20      training = Training.load(training_name)
```

Listing (7.1): Ausschnitt 1 aus der Main-Funktion des Programms

```
21      model = training.model.copy()
22      def get_reward(weights):
23          model.set_weights(weights)
24
25          agent = SMBAgent("Level1-1")
26          fitness, _ = agent.play(model, render)
27
28          return fitness
29
30
31      es = EvolutionStrategy(training.model.get_weights(), get_reward,
        training.population_size, training.sigma, training.learning_rate)
32
33      while True:
34          (main_weights, main_reward), (population_weights,
        population_rewards) = es.run_generation()
35
36          training.save(main_weights, main_reward, population_weights,
        population_rewards)
```

Listing (7.2): Ausschnitt 2 aus der Main-Funktion des Programms

Das Training der KI wird gestartet, indem entweder eine vorhandene Trainingssession geladen oder eine neues erstellt wird. Aus den Trainingsparametern, dem Neuronalen Netz und einer Fitness-Funktion wird die Evolutionäre Strategie gebildet. Diese trainiert dann auf das Maximieren der Fitness hin. Die Fitness-Funktion beinhaltet einen Agenten, der mit der Umgebung arbeitet und das Spiel spielt.

```
1       from model import NeuralNetwork
2       from agent import SMBAgent
3
4       model = NeuralNetwork.load(f"src/Mario-ES/trainingName/best.h5")
5
6       agent = SMBAgent("Level1-1")
7       agent.play(model, realtime=True, render=True)
```

Listing (7.3): Ausschnitt aus der Play-Funktion des Porgramms

Der Code aus Listing 7.1 und Listing 7.2 zeigt die Struktur und die Komponenten, die für das Trainieren der Künstlichen Intelligenz verwendet werden. Zuerst wird eine vorhandene Trainingssession geladen oder eine neue kreiert (Zeilen 11 bis 20). Anschließend wird die Fitness-Funktion definiert, welche den Agenten und das Neuronale Netz zum Trainieren verwendet (Zeile 21 bis 28). Zusammen mit den Trainingsparameter wird die Evolutionäre Strategie daraus erstellt (Zeile 31 bis 36).

Das Listing 7.3 zeigt, wie eine trainierte KI ihr Ergebnis wiedergibt und welche Komponenten des Systems dafür verwendet werden. Hierbei wird eine Trainingssession und dessen Neuronales Netz geladen. Anschließend spielt der Agent das Spiel um einen möglichst besten Fitness-Wert zu erlangen.

Wie diese Komponenten im Einzelnen implementiert werden und was ihr genauer Zweck ist, wird in den folgenden Abschnitten näher beschrieben. Dabei wird im Agenten auch näher auf das Spiel SuperMarioBros. eingegangen. Trotzdem werden die Komponenten allgemein implementiert, so dass nur die Fitness-Funktion und dessen Agenten ausgetauscht werden müssen, um die KI auf ein anderes Spiel oder eine andere Umgebung trainieren zu können.

7.2 Agent

In diesem Kapitel wird der Agent des Systems genauer beschrieben. Er ist für das eigentliche Spielen zuständig, für das Starten und Kommunizieren mit der Umgebung. Er steuert die Spielfigur mit Hilfe des Neuronalen Netzwerkes und berechnet den Fitness-Wert.

7.2.1 Umgebung SuperMarioBros.

Der erste wichtige Schritt für das Entwickeln einer KI, die ein Spiel spielen und lösen soll, ist es, das Spiel zu verstehen. Dazu gehört ein Verständnis für die Steuerung, mögliche Power-Ups und Gewinnkonditionen

Ziel des Spiels ist es, die Prinzessin Peach aus den Fängen des Kontrahenten Bowser zu befreien. Dazu steuert man den Charakter Mario. Das Spielegenre ist Jump'N'Run, der Charakter wird von links nach rechts bewegt und besitzt zusätzlich die Fähigkeiten sich zu ducken und zu springen. Das Spiel ist in acht Welten unterteilt, die wiederrum jeweils in vier einzelnen Level unterteilt sind. Die ersten drei Level jeder Welt enden, sobald Mario den Fahnenmast berührt. Das letzte bzw. vierte Level in jeder Welt ist ein Boss-Kampf gegen Bowser. In diesem Kampf muss Mario an Bowser vorbei kommen, um die "Axt" berühen zu können.

Die einzelnen Level gehören zu einem von sechs verschiedenen Typen. Diese besitzen unterschiedliche Eigenschaften. So gibt es Level, die einen hellblauen Hintergrund haben und wiederum andere Level mit schwarzem Hintergrund. Dies ist ein wichtiger Punkt für die KI, wenn sie lernen möchte, zwei dieser unterschiedlichen Level zu lösen. Weiterhin gibt es Level, welche die Steuerung verändern. So führen Unterwasserlevel und Gebiete dazu, dass Mario nicht mehr springen kann, dafür aber schwimmen. Auf diese Level wird im Folgenden jedoch nicht weiter eingegangen.

Zusätzlich zu den 13 verschiedenen Gegnertypen gibt es noch Power-Ups, dem Spieler das Spielen erleichtern sollen. Diese Power-Ups können Mario größer machen und ihm zusätzliche Fähigkeiten verleihen. Da der Power-Up Status direkt das Erscheinungsbild

von Mario beeinflusst, muss der KI keine zusätzliche Information gegeben werden. In Kapitel 7.2.3 wird die Steuerung noch einmal näher beschrieben.

7.2.2 Spielen

Um das Spiel zu spielen verwendet der Agent Gym-Retro (siehe Kapitel 6.1). Dabei wird die Umgebung bei der Initialisierung des Agenten selbst initialisiert, dazu gehören sowohl das Spiel (SuperMarioBros), als auch der aktuelle Zustand (Level). Um das Spiel zu starten, muss dem Agenten jedoch noch das Neuronale Netz übergeben werden, mit welchem das Spiel gespielt werden soll. Nachdem das Spiel gestartet wurde, wird nach jedem Frame die Beobachtung (das Bild, siehe Kapitel 7.2.4) an das Neuronale Netz übergeben, welches seine Ausgabe wieder zurückgibt. Diese Ausgabe gibt an, welche Aktion als nächstes ausgeführt wird. Sie wird an die Umgebung weitergeleitet. Für jede Aktion, die der Agent macht, bekommt er eine neue Beobachtung und eine Belohnung zurück.

```
1     def play(self, model):
2         obs = self.env.reset()
3         done = False
4         fitness = 0
5
6         while not done:
7
8             action = model.predict(obs)[0]
9
10            obs, reward, done, info = self.env.step(action)
11            fitness += reward
12
13        return fitness
```

Listing (7.4): Vereinfachte Darstellung der Spiele Funktion des Agenten

Dies wird so lange wiederholt, bis das Spiel entweder beendet wurde oder der Agent gescheitert und gestorben ist. Ist dies der Fall, so werden die Belohnungen zu einer Fitness zusammengerechnet und zurückgegeben. Listing 7.4 zeigt eine vereinfachte Darstellung, wie sich das Spielen des Agenten zusammensetzt (vergl. Kapitel 6.1). Das Spiel wird zuerst auf den Anfangszustand zurückgesetzt (Zeile 2 bis 4). Anschließend wird das Spiel Bild für Bild abgespielt. Dabei hat die KI bei jedem Schritt die Möglichkeit eine Aktion zu wählen (Zeile 8). Diese wird vom Agenten ausgeführt und der neue Zustand der Umgebung wird an die KI zurückgegeben (Zeile 10). Ist das Spiel beendet wird die erzielte Fitness an das System übergeben.

7.2.3 Steuerung

Wie in Kapitel 6.1 bereits beschrieben wurde, geschieht die Steuerung der Spielfigur durch das Übergeben einer Aktion an die Schritt-Funktion der Umgebung. Die Aktion besteht aus einer Liste, die angibt, welche Tasten gedrückt werden sollen und welche nicht.

Abbildung (7.1): NES-Controller [1]

Der Controller für das Nintendo Entertainment System (NES) besteht aus insgesamt 8 Tasten, Taste "A", Taste "B", jeweils einer Start- und Select-Taste und einem Steuerkreuz, das aus vier Tasten für Oben, Unten, Rechts und Links besteht (siehe Abbildung 7.1).

Für das Spiel SuperMarioBros. sind jedoch nicht alle Tasten von Relevanz. Sowohl die Start- als auch die Select-Taste haben keinen Einfluss auf dies Steuerung von Mario, weswegen diese vernachlässigt werden können. Als weitere Einschränkung gilt, dass er weder den Pfeil nach oben, noch nach unten auf dem Steuerkreuz drücken darf. Dies schränkt die KI zwar in seinem Handeln ein, bedeutet aber auch, dass er weniger Optionen hat und somit weniger ausprobieren muss. Er kann somit auch nicht in eine Röhre hineinklettern, die sich oberhalb oder unterhalb von ihm befindet. All diese Bewegungseinschränkungen hindern ihn aber nicht am Erreichen des Ziels. Auch mit nur den vier Tasten – Rechts, Links, A und B – kann die KI das Ziel erreichen.

Diese vier Tasten darf die KI nach Belieben kombinieren. Er kann eine, zwei, drei oder alle vier gleichzeitig drücken. Er kann aber auch keine Taste drücken und abwarten was passiert. Dadurch, dass sowohl Rechts als auch Links gleichzeitig gedrückt werden kann, unterscheidet sich die KI von einem echten Spieler, da der Controller diese Eingabe normalerweise nicht erlaubt. Da diese Eingabe aber das gleiche Ergebnis erzeugt, wie keine Richtungstaste zu drücken, ermogelt sich die KI auch keinen Vorteil.

Die Steuerung ist direkt mit dem Neuronalen Netz verbunden, da dessen Ausgabe als direkte Eingabe zur Steuerung verwendet wird. Somit ist die Dimension der Ausgabeschicht des Neuronalen Netzes von der Steuerung abhängig.

7.2.4 Beobachtung

Das Training der KI basiert auf den Beobachtungen, die gemacht werden. Das bedeutet, dass lediglich diese Beobachtungen entscheiden, welche Tasten die KI drücken soll. Die Beobachtung ist vergleichbar mit dem, was der Mensch wahrnimmt. Genau wie eine KI "beobachtet" er seine Umgebung und trifft dann eine Entscheidung, wie er handeln wird. Die Umgebung, in der Menschen diese Entscheidungen treffen, besteht aus deren Umwelt. Durch seine Sinne beobachtet er seine Umgebung, so kann er diese sehen, riechen, schmecken, fühlen, hören und per Gleichgewicht spüren.

Die KI beobachtet das Spiel nur mit ihren "Augen", sie kann ihre Umgebung nur sehen. Die Beobachtung besteht nur aus einem Bild. Dies ist das Bild das auch auf dem Bildschirm des Spielers ausgegeben wird. Damit ist die KI ein bisschen benachteiligt gegenüber dem menschlichen Spieler, da dieser zusätzlich zu einem Bild auch noch die Audioausgabe des Spieles "beobachten" kann. Aber diese ist nicht nötig, um das Spiel durchspielen zu können und würde das Training der KI zu erschweren.

Abbildung (7.2): Screenshot des Spieles SuperMarioBros.

Abbildung 7.2 ist eine einzelne Beobachtung. Dargestellt sieht man einen Screenshot des Spieles, zu sehen ist das, was dem menschlichen Spieler auf dem Bildschirm ausgegeben wird. Dieses Bild wird dem Neuronalen Netz zur Entscheidungsfindung übergeben. Doch zuvor wird dieses mit einem Vorbearbeitungsprozess vereinfacht (siehe **??**).

7.2.5 Belohnung/Fitness

Die KI soll die Fitness maximieren. Der Agent berechnet diese, indem er die Belohnungen, die er bekommt, zusammenrechnet. Dadurch kann auch verschiedenes Verhalten trainiert werden. Möchte man viele Punkte bekommen, dann belohnt man den Agenten dafür, dass

er Münzen einsammelt oder Gegner tötet. Ist das Ziel, so lange wie möglich am Leben zu bleiben, dann belohnt man ihn, indem man ihm Punkte für jede Sekunde gibt, die er am Leben ist.

Im Falle von SuperMarioBros. ist das Ziel erreicht, sobald Mario die Fahne am Ende des Levels erreicht. Dabei kann er Münzen einsammeln und Gegner töten, was ihm auch im Spiel Punkte gibt, aber nicht seinem Endziel näher bringt, nämlich die Prinzessin zu retten. Das einzig wichtige hierbei ist, das Level abzuschließen.

Mario wird also belohnt, je näher er dem Ziel kommt oder einfacher gesagt, je weiter er nach rechts vordringt, desto höher ist seine Fitness. Die Position von Mario innerhalb eines Levels zu berechnen ist dabei jedoch nicht trivial. Daher wird die Position des aktuell zu sehenden Bildschirms im Level als Fortschritt genommen. Dieser Wert lässt sich aus dem Speicher des Emulators auslesen.

Um das Spielen und Berechnen der Fitness zu beschleunigen, wird das Spiel vorzeitig abgebrochen, falls für eine gewisse Zeit kein Fortschritt gemacht wird. Läuft Mario die ganze Zeit in die falsche Richtung, dann ist es unnötig das Spiel zu Ende zu spielen, da er sich in der restlichen Zeit vermutlich nicht mehr verbessert. Ist dies der Fall, wird das Spiel abgebrochen und der aktuelle Fitness-Wert als endgültige Fitness übernommen.

7.3 Neuronales Netzwerk

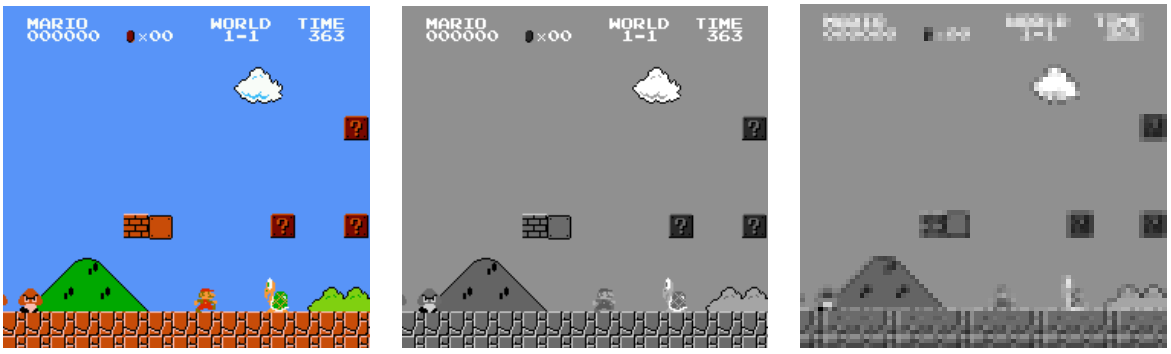
Das Neuronale Netzwerk ist das Zentrum der KI, es ist das "Gehirn". Es trifft alle Entscheidungen und passt sich an, um neue Dinge lernen zu können. Für die Implementierung wird Keras und Tensorflow verwendet.

7.3.1 Bildvorbearbeitung

Das Neuronale Netz bekommt die Bilder vom Agenten überliefert und trifft basierend auf diesen Beobachtungen Entscheidungen. Doch bevor das Bild in das Netz übergeben wird, geschieht noch eine Bildvorverarbeitung, durch welche die Größe des Bildes verkleinert wird und somit die Größe des ganzen Netzes klein gehalten werden kann. Ein kleineres Netz bedeutet zwar, dass es weniger lernen kann, jedoch lernt so ein Netz schneller, da es weniger Werte anzupassen gibt. Natürlich darf das Netz nicht zu klein sein, da es dadurch die gewünschte Aufgabe nicht mehr lösen kann. Dies ist ein wichtiges Ziel des maschinellen Lernens. Die Komplexität des Lösungsansatzes, also das Neuronale Netzwerk, sollte so groß sein, wie die Komplexität des Problems. Ist sie kleiner, dann kommt es zu overfitting und das Problem kann nicht gelöst werden, ist sie größer, dann kann das Problem zwar gelöst werden, benötigt jedoch mehr Zeit. Ziel ist es also das Netzwerk klein genug zu halten, dass das Problem noch gelöst werden kann, aber das Training in möglichst kurzer Zeit den Erfolg liefert.

In diesem Fall wird das Bild in drei Schritten vorverarbeitet. Zuerst wird das Farbbild

in ein Graubild umgewandelt. Somit werden aus den drei Werten jedes Pixel nur noch ein einziger Wert. Anschließend wird die Höhe und Breite des Bildes auf ein Drittel der Originalgröße skaliert. Durch diese Skalierung verringert sich die Größe des Gesamtbildes auf nur noch ein Neuntel des Graubildes. Dies macht es zwar schwieriger etwas auf dem Bild zu erkennen, es ist jedoch noch groß genug und hat noch genügend Informationen über das Originalbild, um alle Objekte eindeutig erkennen zu können. Im letzten Schritt werden alle Graupixelwerte auf einen Bereich zwischen 0 und 1 normalisiert. Dies hat den Vorteil, dass das NN schneller Entscheidungen treffen kann.



(a): Original Bild **(b):** Graubild **(c):** Skaliertes Bild

Abbildung (7.3): Bilder des Vorbearbeitungsprozesses für das Neuronale Netz

Abbildung 7.3 zeigt den Bildvorbearbeitungsprozess eines einzelnen Bildes. Links im Originalbild hat das Bild noch volle Auflösung und Farbe. Obwohl das mittlere Bild die Originalfarben verloren hat, können noch alle Hindernisse und Gegner von einem Menschen deutlich erkannt werden. Und auch nachdem die Auflösung des Bildes verringert wurde (Bild c) sind die einzelnen Objekte zu erkennen. Die Schrift am oberen Ende des Bildes ist im Verlauf dieses Prozesse zwar undeutlich geworden, da diese Informationen aber irrelevant für die KI sind, ist dies kein Hindernis für sie.

7.3.2 Architektur

Die Struktur des Netzes ist bis auf die Ein- und Ausgabe fest implementiert. Sie besteht aus drei Convolutional Schichten und anschließend aus zwei Fully-connected Schichten. Listing 7.5 zeigt, wie das Netzwerk in Keras realisiert wird. Die Form der Eingabe ist variabel und je nach Umgebung unterschiedlich gestaltet. Zum einen lässt dies unterschiedlich große Bilder zu und zum anderen kann die Anzahl der Eingabebilder festgelegt werden. Das heißt, es können für einen Durchgang mehrere Bilder zusammen in das Netz übergeben werden. Dazu speichert sich das System die einzelnen Bilder und gibt diese in einem ”Paket” an das Neuronale Netz. Dies ermöglicht es der KI zeitliche Zusammenhänge zwischen Ereignissen, wie zum Beispiel Beschleunigung und Geschwindigkeit, herauszufinden.

Die Steuerung von Mario wird im Verlauf der Entwicklung aus Testzwecken umgestellt,

dies ist der Grund, warum auch die Anzahl an Ausgabewerten variabel bleibt. Außerdem ermöglicht dies später, die gleiche Netzwerkarchitektur bei verschiedenen Spielen verwenden zu können, da diese möglicherweise eine andere Steuerung und Bildschirmausgabe besitzen. Dies bedeutet jedoch nicht, dass die Struktur des Netzes während des Trainings verändert werden kann.

```
1     def _build_model(self, input_shape, output_dim):
2         model = Sequential()
3         model.add(Conv2D(filters=32, kernel_size=(8, 8), strides=4,
      activation="relu", input_shape=input_shape))
4         model.add(Conv2D(filters=64, kernel_size=(4, 4), strides=2,
      activation="relu"))
5         model.add(Conv2D(filters=64, kernel_size=(3, 3), strides=1,
      activation="relu"))
6         model.add(Flatten())
7         model.add(Dense(512, activation="relu"))
8         model.add(Dense(output_dim))
9         return model
```

Listing (7.5): Implementierung des Convolutional Neuronales Netz mit Keras

Abbildung 7.4 zeigt eine schematische Darstellung des Neuronalen Netzwerkes. Sie zeigt die Eingabeschicht mit den vier vorverarbeiteten Graubilder, die drei Faltschichten für die Objekterkennung, eine Fully-connected-Schicht und die Ausgabeschicht mit den vier Werten für die Tasten des Controllers.

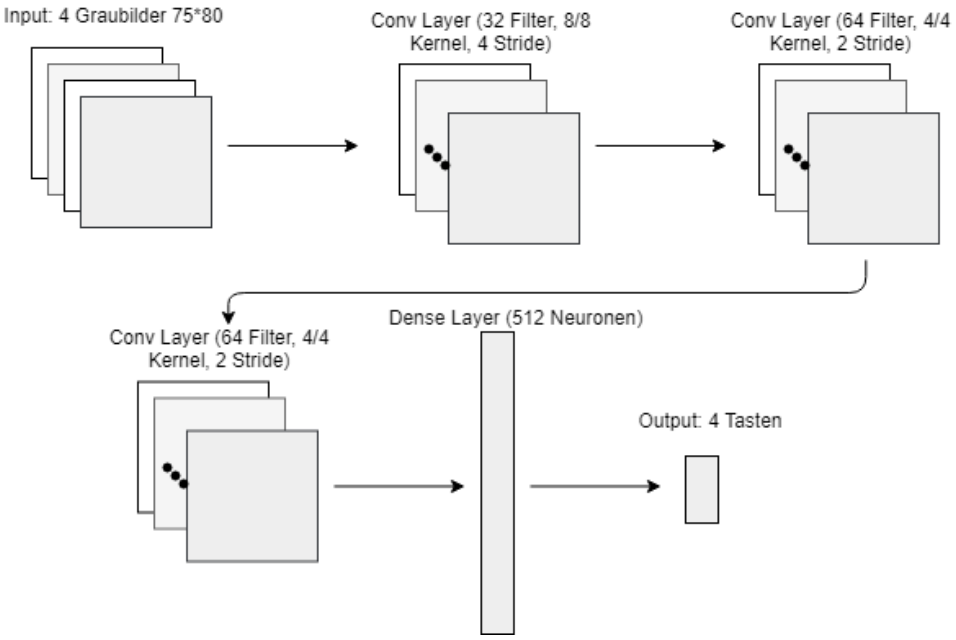


Abbildung (7.4): Schematische Darstellung des Neuronalen Netzes

7.4 ES Implementation

Die Evolutionäre Strategie ist das Kernstück der Implementierung. Sie trainiert die KI und verbessert das Neuronale Netz immer weiter. Es ist so implementiert, dass es mit verschiedenen Neuronalen Netzen funktioniert.

```
1     def __init__(self, weights, get_fitness_func, population_size, sigma,
      learning_rate):
2         self.weights = weights
3         self.get_fitness = get_fitness_func
4         self.population_size = population_size
5         self.sigma = sigma
6         self.learning_rate = learning_rate
```

Listing (7.6): ES Initialisierung

Listing 7.6 zeigt wie die ES initialisiert wird und welche Werte übergeben werden. Zum einen müssen die Gewichte des Neuronalen Netzes übergeben werden, das trainiert werden soll. Hier kann auch ein Netz übergeben werden, welches schon einmal trainiert wurde, um das Training fortzusetzen. Zudem braucht es eine Funktion, die einen Zahlenwert der Fitness zurückgibt. Die ES trainiert auf das Maximieren dieses Wertes hin. Als letztes müssen noch drei Trainingsparameter übergeben werden. Die Populationsgröße gibt an, wie viele Individuen pro Generation generiert werden sollen. Wie groß die Mutationsrate pro Gewicht maximal sein kann, wird durch Sigma bestimmt. Die Lernrate gibt schließlich an, wie stark die Durchschnittsmutation auf das Neuronale Netz angewendet wird.

```
1     def run_generation(self):
2         population = self._get_population()
3
4         population_weights = []
5         population_rewards = []
6         for p in population:
7             weights_try = self._get_weights_try(self.weights, p)
8             population_weights.append(weights_try)
9             population_rewards.append(self.get_fitness(weights_try))
10
11        self._update_weights(population_rewards, population)
12        main_reward = self.get_fitness(self.weights)
13
14        return (self.weights, main_reward), (population_weights,
      population_rewards)
```

Listing (7.7): Generationsschritt der ES

Nachdem die ES initialisiert wird, kann das Training gestartet werden. Dazu kann ein einzelner Generationsschritt ausgeführt werden. Wie so ein Schritt aussieht, ist in Listing 7.7 zu sehen. Die Funktionsweise ist in Kapitel 3.4 genau beschrieben. Zuerst wird die Population erstellt (Zeile 2), wobei jedes Individuum eine exakte Kopie des Hauptindividuums ist. Anschließend wird bei jedem Individuum eine Mutation vollführt und mit Hilfe der Funktion, die beim Initialisieren übergeben wurde, wird dessen Fitness berechnet (Zeilen 4 bis 9). Nachdem die Fitness aller Individuen berechnet wurde, wird anhand dieser Werte und der einzelnen Mutationen, die Durchschnittsmutation berechnet und am Hauptindividuum durchgeführt (Zeile 11). Zuletzt wird die Fitness des Hauptindividuums berechnet.

Es werden sowohl die Gewichte und die Fitness des Hauptindividuums, als auch der kompletten Population dieser Generation zurückgegeben. Dies ermöglicht es, den Fortschritt abzuspeichern und zusätzliche Statistiken für die Generation zu erstellen, mit welchen der Erfolg des Trainings überwacht werden kann.

7.5 Trainingssession

Eine Trainingssession besteht aus dem Netzwerk des Hauptindividuums und den Trainingsparametern. Da jede Session ihr komplett eigenes Netzwerk und eigene Parameter besitzt, sind sie verschiedene KIs. Mit jeder Trainingssession kann eine neue KI erstellt werden.

Wie in Kapitel 7 beschrieben wurde, ist es wichtig, die KI zu testen. Durch die verschiedenen Trainingssessions kann die KI während der Entwicklung immer wieder komplett neu getestet werden. Außerdem ist es ermöglicht verschiedene KIs zu entwickeln, die verschiedene Aspekte der Umgebung trainieren (siehe Kapitel 8).

7.5.1 Parameter

Die Parameter, die neben dem Neuronalen Netzwerk gespeichert werden, sind die Populationsgröße, die Mutationsrate Sigma und die Lernrate. Diese drei Werte benötigt die Evolutionäre Strategie, um das Training fortsetzen zu können.

Zusätzlich werden noch Informationen über die aktuelle Iteration, die Fitness des besten Hauptindividuums, die Fitness des insgesamt besten Individuums, sowie ein Neuronales Netzwerk des besten Individuums festgehalten. Dies ermöglicht es das beste Ergebnis, das von der KI gefunden wird, wiederzugeben.

7.5.2 Kreieren

Beim Kreieren einer neuen Trainingssession (KI) müssen sechs Werte spezifiziert werden. Zum einen müssen die drei Trainingsparameter – Populationsgröße, Mutationsrate und

Lernrate – angegeben werden. Des Weiteren müssen die zwei Werte für Ein- und Ausgabedimension des Netzwerkes angegeben werden (siehe Kapitel 7.3.2). Als letztes muss noch ein Pfad angegeben werden. Dieser Pfad ist das Verzeichnis, in dem alle Daten und Informationen gespeichert werden.

Da es beim Kreieren einer Session keine Rolle spielt mit welcher Umgebung trainiert wird, mit Ausnahme der Größe der Netzwerkausgabe (siehe Kapitel 7.2.3), müssen hier weder Fitness-Funktion noch Agent spezifiziert werden. Diese KI kann auch auf verschiedenen Umgebungen trainieren, solange Ein- und Ausgabe des Netzwerkes übereinstimmen.

7.5.3 Speichern

Das Speichern einer Trainingssession geschieht nach jedem Generationsschritt. Dabei werden die Werte der aktuellen Iteration, des besten Haupt- und Normal-Individuums, sowie das neue Neuronale Netzwerk abgespeichert. Wird ein neuer Rekord erzielt, so wird das Netzwerk zusätzlich abgespeichert, welches diesen Rekord aufgestellt hat.

7.5.4 Laden

Soll das Training einer KI fortgesetzt werden, muss dieses zuerst geladen werden. Dabei werden das Neuronale Netzwerk, sowie die drei Trainingsparameter geladen. Anschließend kann das Training mit der Evolutionären Strategie fortgesetzt werden.

Soll die KI nur ausgeführt werden, um das Ergebnis des bisherigen Trainings zu begutachten, reicht es, das Neuronale Netz, mit dem die KI das bisher beste Ergebnis erzielt hat, zu laden. Es kann jedoch auch das aktuelle Trainingsnetzwerk geladen werden, um dessen Fortschritt zu begutachten.

7.5.5 Zusätzliche Statistiken

Zusätzlich zum Speichern und Laden der Trainingsparameter werden Informationen zur Fitness jedes einzelnen Individuums aller Generationen gespeichert. Diese Informationen werden für das Training der KI nicht benötigt.

Statistiken helfen jedoch dem Nutzer die KI zu überwachen, da damit der Fortschritt über die Generationen eingesehen werden kann. Somit kann erkannt werden, ob und wie stark sich die KI pro Generation verbessert.

Diese Daten werden in Datenreihen in einer CSV-Datei gespeichert. Dabei werden die Fitness-Werte des Hauptindividuums, des besten und schlechtesten, sowie die Werte von jedem einzelnen Individuum in dieser Reihenfolge gespeichert.

8 Ergebnisse

Wie in Kapitel 7 beschrieben wurde ein allgemeiner Ansatz für die Implementierung gewählt. Deshalb konnte durch das verwenden einzelner Trainingssession das System verwendet werden um verschiedene KIs zu entwickeln, die das Spiel oder Teile des Spieles mit unterschiedlichen Parametern trainierten.

Die Ergebnisse der verschiedenen Trainingssession werden in diesem Kapitel aufgezeigt. Dabei wird auf Probleme eingegangen, die während der Entwicklung aufgetreten sind und wie sie gelöst wurden. Außerdem wird erklärt weshalb Änderungen am Training vorgenommen wurden und was für eine Auswirkung diese hatten.

8.1 Erste Trainingsläufe

Die ersten Trainingsläufe werden schon in der Entwicklung gemacht, um die Implementierung des Trainingsalgorithmuses zu testen. Zudem werden weitere kurze Trainingsläufe mit verschiedenen Trainingsparametern vollzogen um eine Kombination an Werten für längere Trainingssession zu finden.

Diese Trainings werden mit einer vereinfachten Variante des Agenten durchgeführt. Der Aktionsraum des Agenten wird hierbei verringert, da dieser nicht die ”Sprinttaste” verwenden kann. Durch den verringerten Aktionsraum muss und kann die KI nicht so viele unterschiedliche Aktionen ausführen. Somit kann schon früh im Training erkannt werden, ob dieses erfolgreich ist oder nicht.

Im späteren Verlauf wird dem Agenten dann der volle Aktionsraum zur Verfügung gestellt. Dies gibt ihm mehr Möglichkeiten ein Level zu lösen und soll vor allem bei schwierigeren Level eine Hilfe sein.

8.1.1 Testen und Parametersuche

Die ersten Trainingsläufe werden mit sehr unterschiedlichen Parametern gestartet. Dabei werden Populationsgrößen im Bereich von 10 bis 100, eine Mutationsrate von 0.1 bis 1 und einer Lernrate von 0.001 bis 0.5 verwendet. Diese Testläufe können frühzeitig Fehler entdecken die bei der Implementierung gemacht worden sind. Außerdem können dadurch Trainingsparameterkombinationen herausgefunden werden, welche ein erfolgreiches Training versprechen.

Auch können Veränderungen an der Fitness-Funktion vorgenommen werden, falls die KI auf ein anderes Ziel hinarbeitet als geplant ist.

Nachdem die ersten Testläufe abgeschlossen sind und sich vielversprechende Parameterkombinationen hervorgehoben haben, können länger Trainingsläufe gestartet werden. Diese Trainingsläufe sollen laufen bis sie entweder das gewünschte Ziel erreicht haben oder bis sie ein lokales Minimum erreicht haben und die Evolution stagniert ist. Wann die Evolution stagniert ist, kann man nicht mit hundertprozentiger Wahrscheinlichkeit sagen, jedoch können Statistiken über die Fitness jeder Generation einen guten Ansatz bieten.

8.1.2 Erster Levelabschluss

Für den ersten großen Trainingslauf wurde das Ziel gesetzt das erste Level der ersten Welt abzuschließen. Das heißt die KI soll ein Neuronales Netzwerk entwickeln, mit welchem dessen Agenten Mario so durch Level 1-1 (die erste 1 steht für die Welt und die zweite 1 steht für das Level innerhalb der Welt) steuert, dass er ohne zu sterben die Fahne am Ende des Levels erreicht.

Für dieses Training wird der vereinfachte Agent, der nicht sprinten kann, verwendet. Sprinten ist in diesem Level nicht notwendig um es abzuschließen. Alle Hindernisse können auch ohne die Fähigkeit zu Sprinten überwunden werden.

Trainingsparameter	Wert
Populationsgröße	50
Mutationsrate	0,4
Lernrate	0,2
Fitness-Funktion	Distanz Level 1-1

Tabelle (8.1): Parameter für das Trainieren der KI auf Level 1-1

Die Trainingsparameter dieses Trainings sind in Tabelle 8.1 aufgelistet. Populationsgröße und Mutationsrate sind so aufeinander abgestimmt, dass jegliche Mutationen von einem oder mehreren Individuen abgedeckt sind. Zudem sollte die Mutationsrate groß genug sein, dass die KI nicht in einem lokalen Fitness-Wert Maximum hängen bleibt. Je größer die Population dadurch aber wird, desto länger braucht die KI jedoch für einen Generationsschritt.

Die Lernrate ist relativ hoch gewählt worden, dadurch legt sich die KI schon früh auf einen Lösungsweg fest. Dadurch kann die KI zwar schneller ihr Ziel erreichen, jedoch steigt auch die Gefahr in ein lokales Maximum hängen zu bleiben.

Die Fitness berechnet sich aus der Distanz die Mario in Level 1-1 zurücklegt.

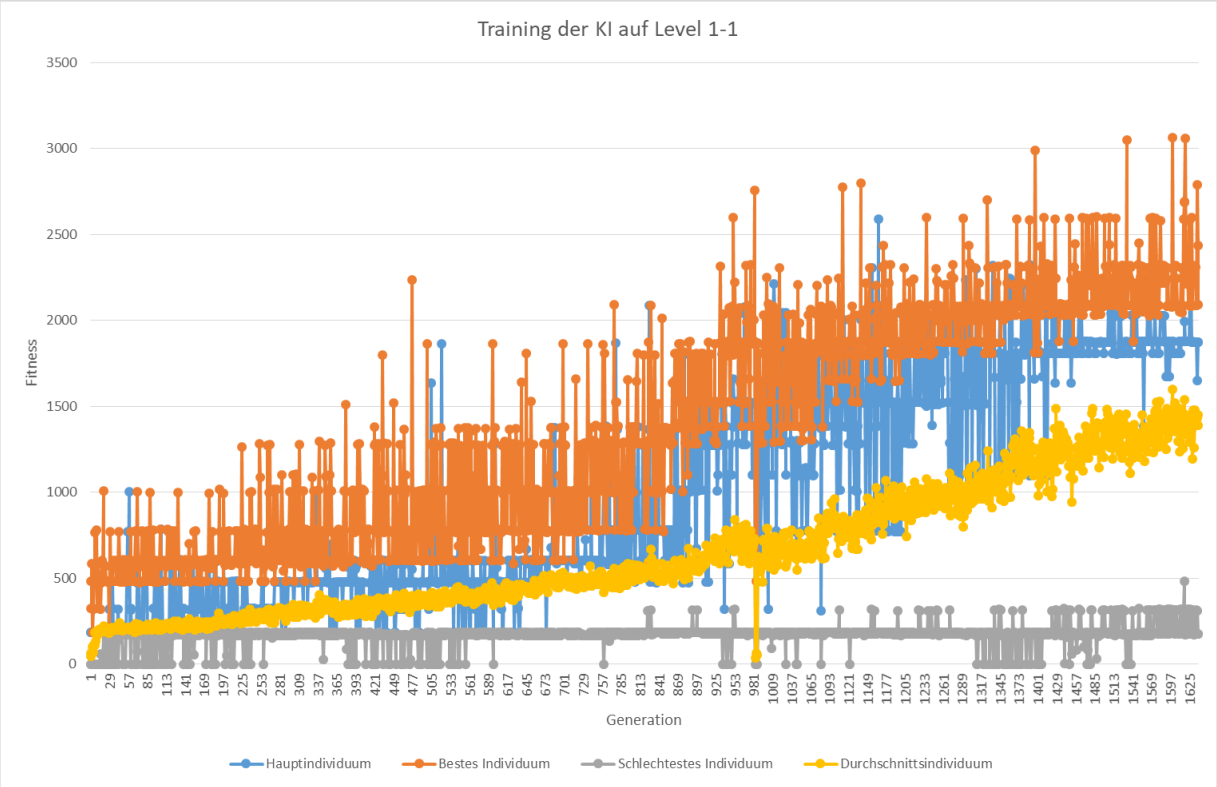


Abbildung (8.1): Trainingsstatistiken des Agenten ohne sprinten auf Level 1-1

In Abbildung 8.1 ist der Fortschritt der KI bei ihrem Training auf Level 1-1 zu beobachten. Sie zeigt den Fitness-Wert für das jeweils beste und schlechteste Individuum jeder Generation, sowie den Fitness-Wert des Hauptindividuums und einen errechneten Durchschnitts-Fitness-Wert jeder Generation. Das Training lief insgesamt für 1638 Generationen und ca. 2 $\frac{1}{2}$ Tage in Echtzeit. Das erste Neuronale Netz das das Level lösen konnte, wurde jedoch schon nach 1538 Generationen gefunden. Der Agent kann mit diesem Netz das Level abschließen, ohne vorher zu sterben.

Es ist ein deutlicher Erfolg im Training der KI zu erkennen. Sowohl der Fitness-Wert das besten als auch das Haupt- und Durchschnittsindividuum verbessert sich mit den Generationen. Nur das schlechteste Individuum scheint keinen richtigen Fortschritt zu machen. Wie aber der Durchschnitt in jeder Generation zeigt, kommen die meisten Individuen mit der Zeit immer weiter im Level. Da die Mutation gleichverteilt ist, zeigt dies, dass die Evolution noch immer erfolgreich voranschreitet. Auch wenn das Hauptindividuum keinen sichtlichen Fortschritt macht, zeigt der Durchschnitt-Fitness-Wert, dass die ”Richtung”, in der sich die KI entwickelt, immer bessere Ergebnisse liefert.

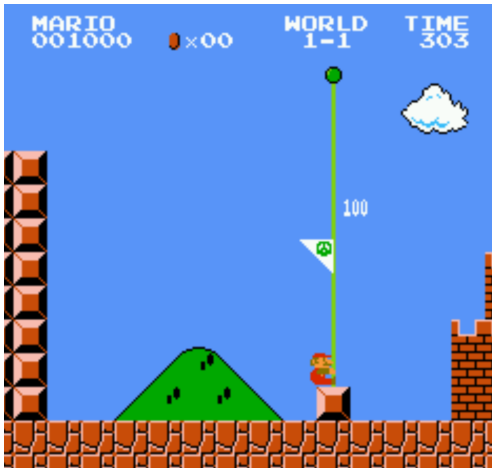


Abbildung (8.2): Erster Mario der das Ziel erreicht hat

Abbildung 8.2 zeigt den ersten Mario, den die KI durch das komplette erste Level des Spieles steuern konnte. Er hat ohne zu sterben alle Hindernissen und Gegner überwunden und nach 97 Sekunden Spielzeit den Fahnenmasten erreicht.

8.2 Weitere Trainingsläufe

Da sowohl der Algorithmus als auch die Trainingsparameter erfolgreich getestet waren, können weitere Trainingsläufe auf weiteren Level des Spieles gestartet werden. Hierzu wurde der vollständige Agent verwendet. Dies bedeutet, dass Mario nun die Fähigkeit hat zu Sprinten. Dies ist vor allem in den späteren Level wichtig, da diese schwer oder teilweise nicht ohne Sprinten zu bewältigen sind.

8.2.1 Training auf einzelnen Level

Es werden nun weitere Trainingsläufe mit diesem Agenten auf allen Level der ersten Welt gestartet. Alle diesen Trainings sind komplett unabhängig voneinander. Die einzelnen KIs versuchen jeweils ihr zugewiesenes Level zu lösen.

Level 1-1

Das erste Level wurde schon mit dem vereinfachten Agenten gelöst, es ist aber interessant zu vergleichen, wie die beiden unterschiedlichen Agenten das Level lösen. Es werden dazu die gleichen Trainingsparameter verwendet.

Trainingsparameter	Wert
Populationsgröße	50
Mutationsrate	0,4
Lernrate	0,2
Fitness-Funktion	Distanz Level 1-1

Tabelle (8.2): Parameter für das Trainieren der KI auf Level 1-1

Für das Training auf Level 1-1 mit Sprinten werden die Parameter der Tabelle 8.2 verwendet. Diese Parameter sind dieselben wie im Training ohne Sprinten. Auch die Fitness-Werte berechnen sich gleich.

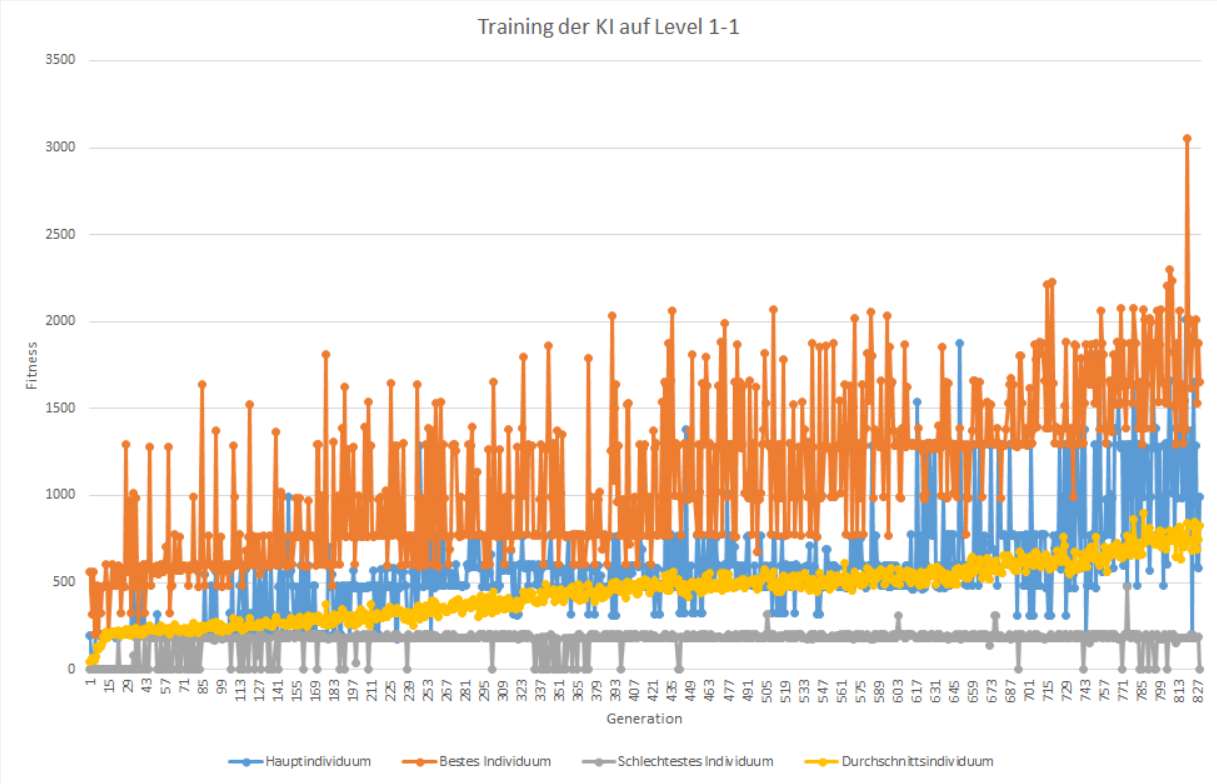


Abbildung (8.3): Trainingsstatistiken des Agenten mit sprinten auf Level 1-1

Abbildung 8.3 zeigt den Fortschritt des Trainings mit Sprinten auf Level 1-1. Dieses Training lief nur 828 Generationen, da die KI in dieser Zeit schon das Ziel erreichen konnte. Dies kann zum einen daran liegen, dass durch die Fähigkeit des Sprintens der Agent schneller auf eine Lösung kommt, da ihm mehr Optionen und somit auch Lösungsmöglichkeiten zur Verfügung stehen. Er könnte das Level immer noch komplett ohne sprinten bewältigen, aber zusätzlich gibt es noch weitere Möglichkeiten mit sprinten das Level zu beenden.

Ein weiterer Grund für den schnelleren Erfolg des zweiten Agenten, kann der Zufall sein. Beim Initialisieren der KI wird zufällig eine Neuronales Netz generiert. Dabei kann ein Netz generiert werden, dass ewig braucht, bis es einen Erfolg aufweist, es kann aber auch ein Netz generiert werden, dass ohne Training das Level lösen kann. Der zweite Agent kann also ein "besseres" Netz am Anfang schon generiert bekommen haben und somit einen Vorteil haben. Und auch bei der Mutation spielt der Zufall eine entscheidende Rolle. Auch hier kann der zweite Agent einen Vorteil bekommen.

Die Grafik zeigt auch, dass das Erreichen des Zieles "glücklich" verlaufen ist, da das zweitbeste Individuum deutlich schlechter ist. Trotzdem werden weitere Trainingsläufe mit der Fähigkeit zu Sprinten ausgeführt, da es der KI mehr Möglichkeiten bietet das Level zu lösen.

Level 1-2

Eine frische KI wird nun auf das zweite Level der ersten Welt losgelassen. Sie soll wie in Level 1 das Ziel erreichen ohne einmal zu sterben. Das zweite Level ist ein Stück weit schwerer als das Erste. Es gibt bewegende Plattformen die überwindet werden müssen. Außerdem muss Mario kurz vor Ende des Levels in eine Röhre kriechen. Dies könnte ihm ein Problem bereiten, da es im ersten Moment mehr Punkte gibt auf die Röhre hinaufzuspringen. Schlussendlich sollte sich Mario aber dennoch für die Röhre entscheiden, da das Level dort weitergeht und der Fitness-Wert verbessert werden kann.

Trainingsparameter	Wert
Populationsgröße	50
Mutationsrate	0,4
Lernrate	0,2
Fitness-Funktion	Distanz Level 1-2

Tabelle (8.3): Parameter für das Trainieren der KI auf Level 1-2

Wie Tabelle 8.3 zeigt, wird auch für das Training auf Level 1-2 die gleichen Parameter verwendet. Nur die Fitness-Funktion ist unterschiedlich, diese berechnet die Distanz die Mario in Level 1-2 zurücklegt.

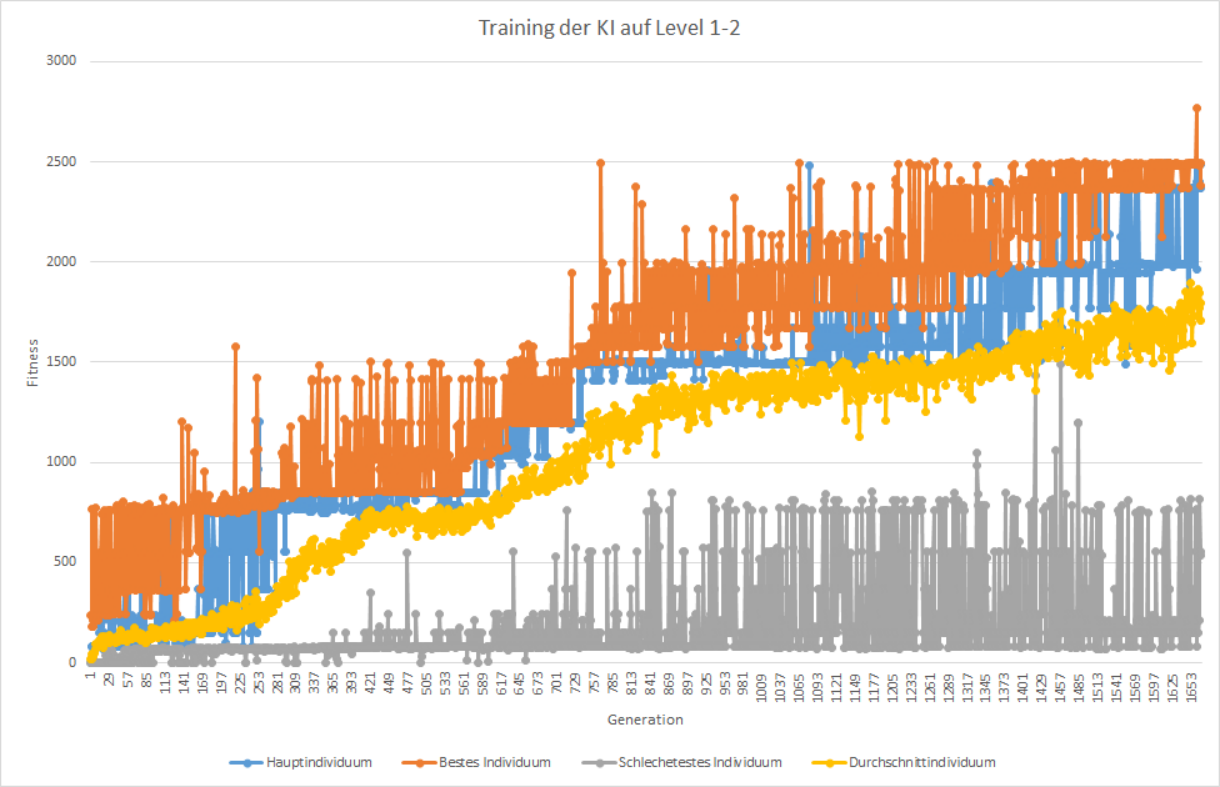


Abbildung (8.4): Trainingsstatistiken des Agenten mit sprinten auf Level 1-2

Das Resultat des Trainings in ist Abbildung 8.4 abgebildet. Wie in den vorherigen Trainings ist auch hier ein deutlicher Anstieg der Fitness über die Generationen zu erkennen und nach 1661 Generationen hat die KI es geschafft das Level zu lösen. Dabei ist sehr gut zu erkennen, dass obwohl das Hauptindividuum keinen ersichtlichen Fortschritt zu verzeichnen hat, der Durchschnitt mit den Generationen trotzdem ansteigt. Das Hauptindividuum macht deswegen keinen Fortschritt, weil es sich nicht genug entwickelt. Es bleibt immer an demselben Hindernis hängen. Es schaffen es aber immer mehr und mehr Individuen über dieses Hindernis hinaus und nachdem genügend herausgefunden haben wie man es überwindet, schafft es auch das Hauptindividuum einen Weg zu finden.

Level 1-3

Nachdem das System KIs entwickeln kann, die jeweils das erste beziehungsweise das zweite Level der ersten Welt durchspielen können, ist das nächste Ziel eine KI zu entwickeln das dritte Level dieser Welt zu lösen. Dazu wird wieder ein neuer Trainingslauf erstellt welcher nur den Fortschritt in der dritten Welt misst und in einen Fitness-Wert umwandelt.

Das dritte Level stellt auch eine größere Herausforderung als die ersten zwei Level dar. Dieses Level spielt auf Plattformen in der Luft und besitzt viele einzelne Lücken. Zudem gibt es mehr bewegende Elemente in diesem Level als in den vorherigen.

Trainingsparameter	Wert
Populationsgröße	50
Mutationsrate	0,4
Lernrate	0,2
Fitness-Funktion	Distanz Level 1-3

Tabelle (8.4): Parameter für das Trainieren der KI auf Level 1-3

Für das Training in Level 1-3 werden die gleichen Parameter verwendet die sich schon auf den ersten beiden Level bewiesen haben (siehe Tabelle 8.4). Auch die Fitness-Funktion wird wieder so angepasst, dass die Distanz die Mario in diesem Level zurücklegt berechnet wird.

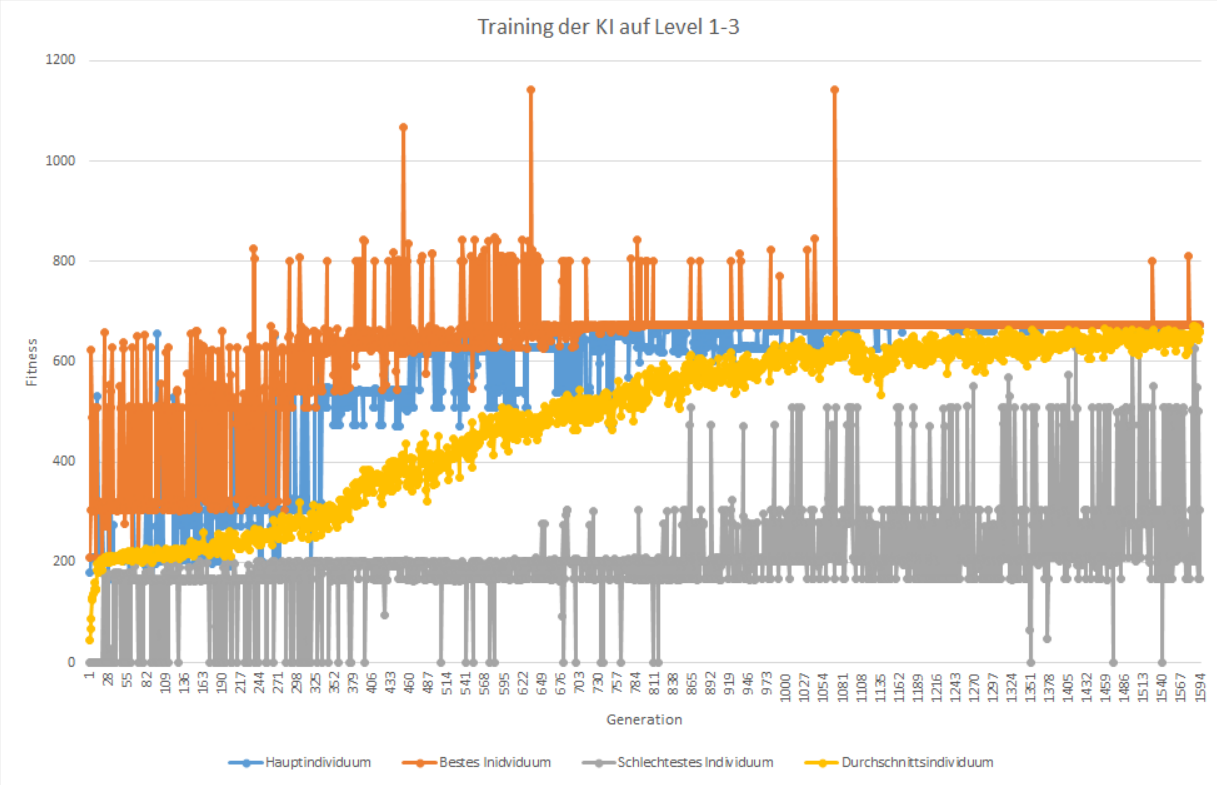


Abbildung (8.5): Trainingsstatistiken des Agenten mit sprinten auf Level 1-3

Der Fortschritt des Trainings ist in Abbildung 8.5 zu sehen. Das Training ist für 1594 Generationen gelaufen und die KI hat es nicht geschafft das Ziel zu erreichen. Der Anfang ist ähnlich dem Training auf Level 1-2. Obwohl das Hauptindividuum keinen sichtlichen Fortschritt macht, steigt der Durchschnitt mit jeder Generation. Dies passiert solange bis auch das Hauptindividuum das blockierende Hindernis überwindet. Die Durchschnitts-Fitness steigt somit weiter an bis sie bei einen Fitness-Wert von circa 600 zum Stillstand

kommt. Zu diesem Zeitpunkt scheint sie keinen Fortschritt mehr zu machen. Die Individuen einer Generation bekommen die gleichen Fitness-Werte wie die Generation davor. Das Hauptindividuum und somit die komplette KI verbessert sich nicht mehr. Sie hat sich in einem lokalen Maximum festgefahren.

Dies muss aber nicht daran liegen, dass die KI es nie schaffen kann, dieses Level zu lösen. Dies liegt am Zufall der Generierung des Hauptindividuums und am Zufall der Mutation. Es wird also ein weiteres Training mit den gleichen Parametern gestartet, um zu sehen, ob diese falsch gewählt wurde, um dieses Level lösen zu können.

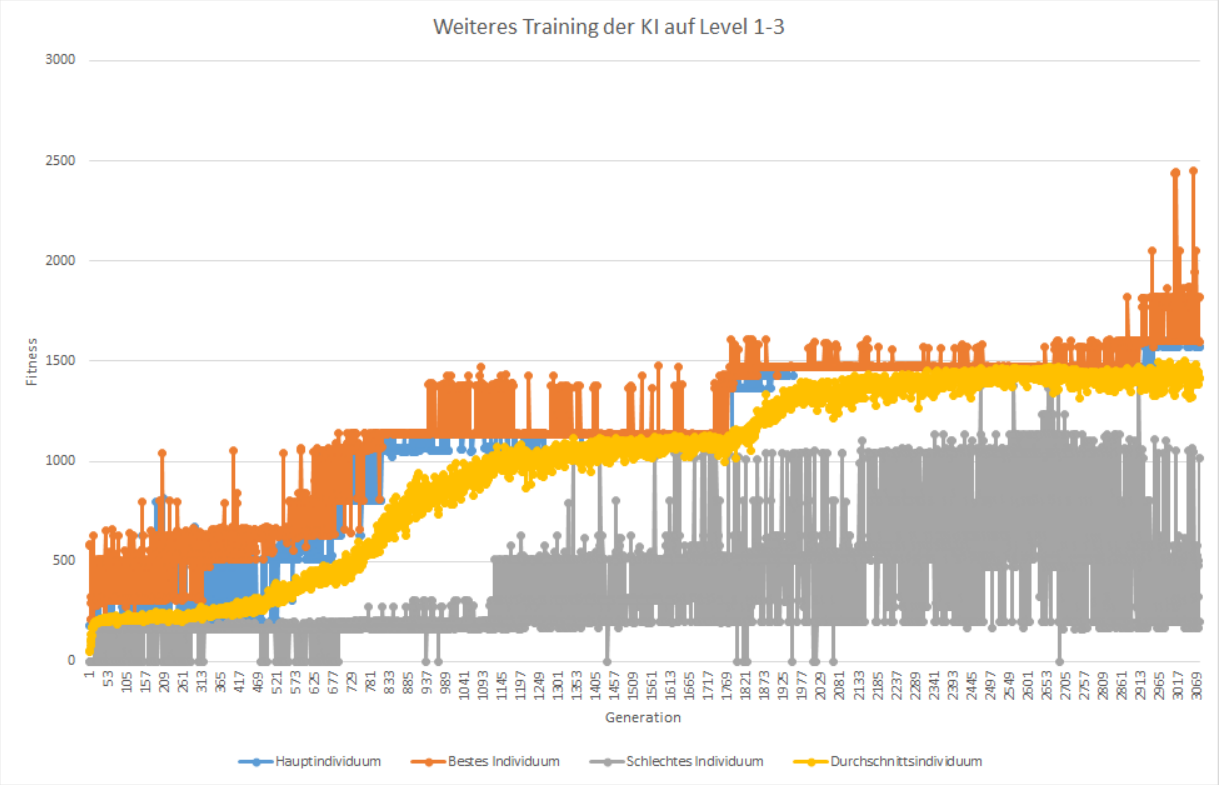


Abbildung (8.6): Trainingsstatistiken des Agenten mit sprinten auf Level 1-3 der zweite Versuch

Dieses weitere Training auf demselben Level ergibt den Graphen in Abbildung 8.6. In diesem Training hat es die KI geschafft nach 3000 Generationen das Level zu lösen. Es ist also möglich mit diesen Trainingsparameter das Level zu lösen. Auch bei diesem Training gab es Hindernisse, bei denen die KI Probleme hatte und länger gebraucht hat, um diese zu überwinden. Doch die Durchschnittsfitness zeigt einen kontinuierlichen Fortschritt in der Evolution.

Level 1-4

x Das nächste Training wird auf dem letzten Level in der ersten Welt vollzogen. Eine neue KI soll lernen dieses Level zu lösen. Das vierte Level in der ersten Welt findet in einer Festung statt und hat somit einen dunklen Hintergrund. Es ist leichter zu lösen als das dritte Level, da es nur wenige Hindernisse und Gegner gibt. Am Ende des Levels wartet jedoch ein ”Kampf” gegen Bowser, in welchem er an ihm vorbeikommen und eine Axt berühren muss. Es muss diesmal kein Fahnenmast berührt werden.

Trainingsparameter	Wert
Populationsgröße	50
Mutationsrate	0,4
Lernrate	0,2
Fitness-Funktion	Distanz Level 1-4

Tabelle (8.5): Parameter für das Trainieren der KI auf Level 1-4

Auch für das Training auf Level 1-4 werden wieder die gleichen Parameter verwendet wie auf den vorherigen Level (siehe Tabelle 8.5). Nur die Fitness-Funktion berechnet diesmal die Distanz die Mario in Level 1-4 zurücklegt.

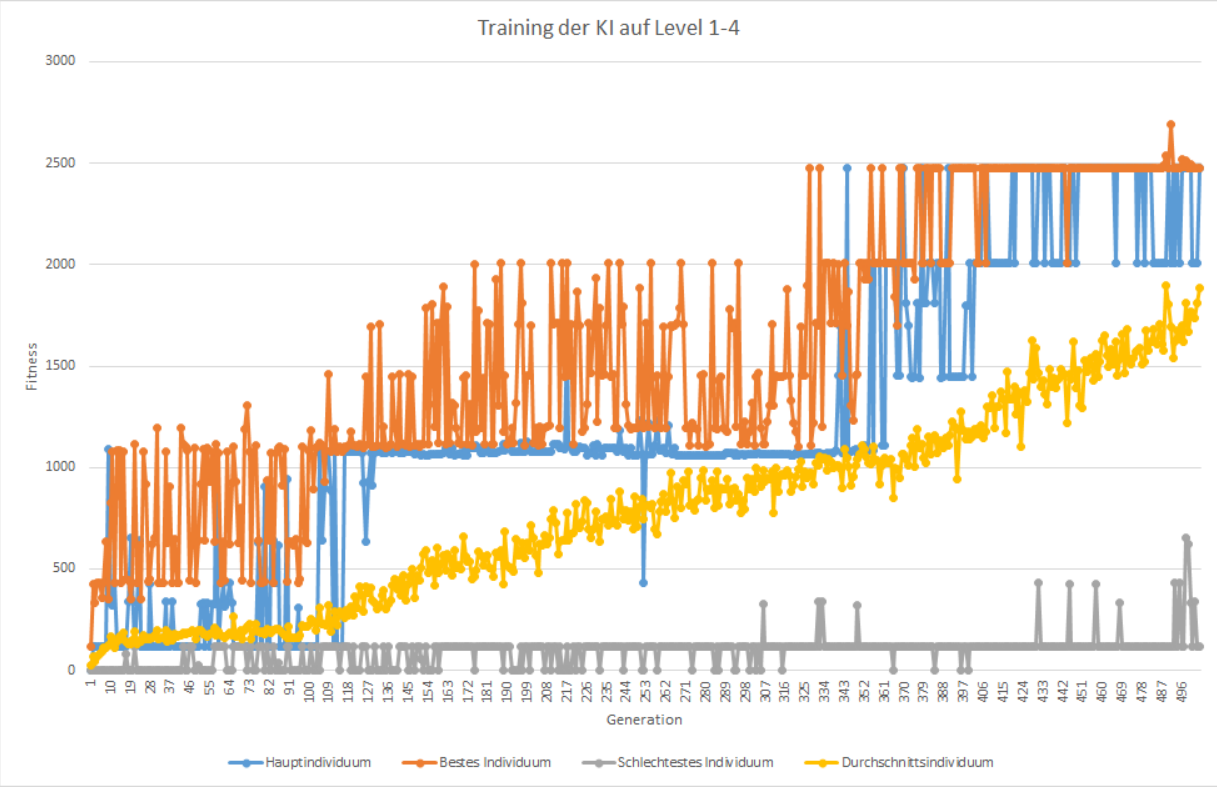


Abbildung (8.7): Trainingsstatistiken des Agenten mit sprinten auf Level 1-4

Das diese Level leichter ist als andere, kann in Abbildung 8.7 erkannt werden. Die KI braucht zum Lösen dieses Levels weniger als 500 Generationen. Auch hier ist der stetige Anstieg des Durchschnitts-Fitness-Wertes deutlich zu erkennen. Die Fitness-Wert-Entwicklung des Hauptindividuums zeigt auch hier wieder wo es Hindernisse gibt und wie schwer es war diese zu überwinden.

8.2.2 Training auf mehreren Level

Bis jetzt wurden einzelne KIs erstellt die jeweils ein einziges Level des Spieles trainieren und lösen sollten. Das nächste Ziel ist es nun mit einer KI mehrere Level gleichzeitig zu trainieren. Dazu wird die Fitness eines Individuums aus dem Fortschritt mehrere Level zusammen gerechnet.

Level 1-1 und 1-2

Die Herrausforderung im Trainieren der ersten beiden Level liegt darin, dass sie sich enorm unterscheiden. Während sich das erste Level komplett über der Erde abspielt, so befindet sich Mario im zweiten Level zum größten Teil im Untergrund. Dies hat zur Folge, dass der Hintergrund von einem hellen blau im ersten Level zu einem schwarz im zweiten Level ändert. Dies bereitet zwar dem menschlichen Spieler kein Problem, da er dies mit nur einem Blick erkennt. Die KI jedoch muss dieses Konzept erst einmal verstehen.

Trainingsparameter	Wert
Populationsgröße	50
Mutationsrate	0,4
Lernrate	0,2
Fitness-Funktion	Distanz Level 1-1 + Distanz Level 1-2

Tabelle (8.6): Parameter für das Trainieren der KI auf Level 1-1 und Level 1-2

Für das Training auf den Level 1-1 und Level 1-2 werden die Parameter verwendet, mit welchen die KI auch schon die Level einzeln lösen konnte (siehe Tabelle 8.6). Die Fitness der KI berechnet sich in diesem Training aus der Kombination der Distanz in beiden Level. Dazu muss jedes Individuum jedoch beide Level spielen. Dies verdoppelt die Zeit die jeder einzelne Generationsschritt braucht. Der Grund dafür diese beiden Level zu nehmen, ist der, dass sie von unterschiedlichen Typ sind. Dies stellt die KI vor der Herausforderung einen generellen Zusammenhang zu finden.

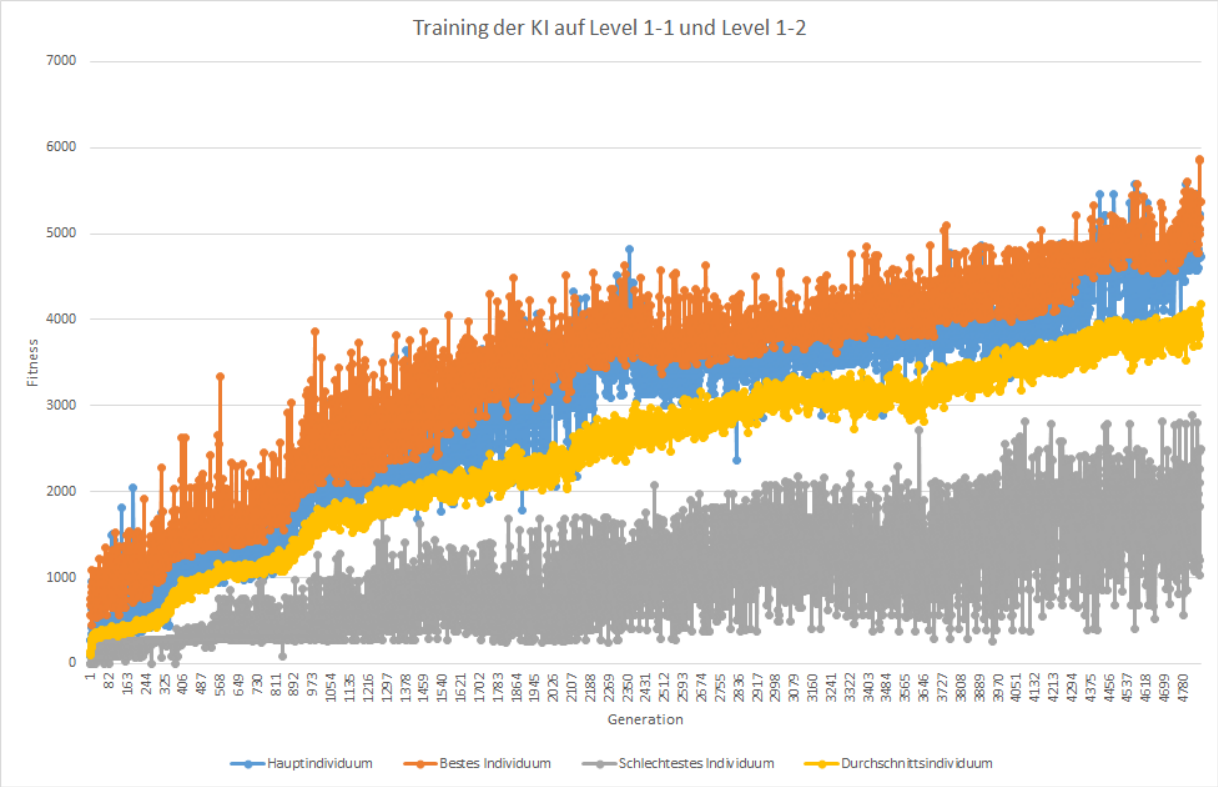


Abbildung (8.8): Trainingsstatistiken des Agenten mit sprinten auf Level 1-1 und Level 1-2

Der Graph in Abbildung 8.8 zeigt dieses Training auf zwei Level gleichzeitig. Die KI braucht 4852 Generationen, bis sie beide Level gleichzeitig lösen kann. Dies sind fast sechsmal so viele Generationen wie im Training nur auf Level 1-1 und fast dreimal so viele Generaionen wie im Training nur auf Level 1-2. Wie schon beschrieben liegt diese große Anzahl an nötigen Generationen an den verschiedenen Typen der beiden Level. Dennoch scheint die KI auch mit diesem Training kein Problem aufzuweisen. Auch hier ist ein stetiger Anstieg der Durchschnittsfitness zu beobachten. Zwar gibt es auch wieder steile und flache Teile, jedoch kann man nicht wie im Training auf Level 1-4 (siehe Kapitel 8.2.1) schwierige Hindernisse erkennen, an denen die KI länger gebraucht hat.

8.3 Interessante Statistiken

Während dem Training der KIs sind einige Interessante Fakten und Statistiken aufgefallen, die im Folgenden aufgezeigt werden.

Hindernisse Level 1-2
Sieht man sich die Fitness aller Individuen jeder Generation an, können die einzelnen Hindernisse eines Levels erkannt werden. Abbildung 8.9 zeigt diese Statistik beim Training auf Level 1-2 (siehe Kapitel 8.2.1). Hier sind deutliche Linien zu erkennen, an welchen über

das ganze Training hinweg immer wieder Individuen hängen bleiben. So ist zum Beispiel bei einer Fitness (Distanz zum linken Rand des Levels) von 550 ein Hindernis, an dem die KI, vor allem Anfang, oftmals hängen bleibt. Es gibt wiederum Hindernisse, welche breiter sind als andere. Diese breiten Stellen weisen oftmals auf ein Loch hin. Da dieses eine längere Fläche bietet, indem die KI scheitern kann.

Andererseits zeigen weiße Teile, die Abwesenheit von Individuen mit solcher Fitness. In diesen Teilen des Levels, gibt es keine Hindernisse, an denen Mario hängen bleiben kann. An diesen Stellen kann die KI nur dann scheitern, falls sie stehen bleibt oder ihre Richtung ändert.

Abbildung (8.9): Trainingsstatistiken aller Individuen im Training auf Level 1-2

Geopferte Marios

Bis zum ersten Mal ein Mario das Ziel erreicht hat, sind in allen Trainingssession bis dato über 470.000 Marios gestorben. Die längste Trainingssession, in der Mario Level 1-1 und Level 1-2 gelernt hat, hat über 240.000 Marios geopfert, um ihr Ziel zu erreichen. Und während aller Trainingssession die die KI gemacht hat, sind insgesamt 1.473.765 Marios gestorben.

9 Zusammenfassung und Fazit

Diese Arbeit sollte zeigen wie man mit Hilfe von Maschinellen Lernen eine Künstliche Intelligenz kreieren und trainieren kann. Für dieses Training wurde die Evolutionäre Strategie verwendet und als Trainingsumgebung und Problemstellung wurden Level aus dem Spiel SuperMarioBros. verwendet. Hierzu verwendet das das System OpenAIs Gym-Retro, eine Bibliothek mit der über ein Emulator (Retro-)Spiele gespielt werden können. Gym-Retro bietet dabei Schnittstellen zur Bildschirmausgabe und zur Controllereingabe des Spieles. Die Entscheidungsfindung der KI funktioniert dabei durch ein Convolutional Neural Network, welches dem menschlichen Gehirn nachempfunden ist. Dieses analysiert die Bilder und entscheidet welche Tasten zu jeden Zeitpunkt gedrückt werden.

Die ersten Trainingssessions haben gezeigt, dass es wichtig ist das Spiel zu verstehen um die richtigen Parameter zu ermitteln. Diese ersten Trainingsläufe beanspruchen vor allem Geduld, diese nicht zu früh abzubrechen.

Weitere Trainingsläufe haben gezeigt, dass mit genügend Zeit die KI es schaffen kann Wege zu finden die einzelnen Level dieses Spieles zu lösen. Auch mehrere Level gleichzeitig zu lernen und lösen ist für die KI möglich.

Als Fazit lässt sich sagen, dass das System alle Anforderungen erfüllt und auch die Trainingsläufe die gewünschten Ergebnisse erzielt haben.

Für die Zukunft könnten jedoch weitere Trainingssession auf spätere Level des Spieles ausgeführt werden. Auch eine Optimierung der Trainingsparameter oder Netzwerkarchitektur könnte eine Verbesserung herbeiführen.

Ein letzter Punkt der interessant wäre, ist das Trainieren auf zufällig generierten Level. Das Risiko dadurch die KI auf speziellen Daten zu overfitten geht dadurch drastisch zurück. Das Ziel dabei soll sein, das komplette Spielprinzip zu verstehen, wie man alle Gegner und Hindernisse überwindet. Durch dieses Training soll die KI nicht nur verstehen wie sie ein bestimmtes Level überwindet, sondern wie sie alle möglichen Level dieses Spiel überwinden kann.

Literatur

[1] Evan Amos. *Nes-Controller*. 30. Juli 2016. URL: https://de.wikipedia.org/wiki/Datei:Nintendo-Entertainment-System-NES-Controller-FL.jpg.

[2] Thomas Back. *Evolutionary Algorithms in Theory and Practice*. Oxford University Press, 11. Jan. 1996. URL: https://www.ebook.de/de/product/21368698/thomas_back_evolutionary_algorithms_in_theory_and_practice.html.

[3] Chrislb. *Künstliches Neuron*. 14. Juli 2005. URL: https://de.wikipedia.org/wiki/K%C3%BCnstliches_Neuron#/media/File:ArtificialNeuronModel_deutsch.png.

[4] Daphne Cornelisse. *An intuitive guide to Convolutional Neural Networks*. 24. Apr. 2018. URL: https://medium.freecodecamp.org/an-intuitive-guide-to-convolutional-neural-networks-260c2de0a050.

[5] Adit Deshpande. *A Beginner's Guide To Understanding Convolutional Neural Networks*. 20. Juli 2016. URL: https://adeshpande3.github.io/A-Beginner%27s-Guide-To-Understanding-Convolutional-Neural-Networks/.

[6] Duden. *Duden Wörterbuch*. 27. Sep. 2018. URL: https://www.duden.de/suchen/dudenonline/Intelligenz.

[7] esDifferent. *Unterschied zwischen lokalem und globalem Maximum*. 7. Jan. 2019. URL: https://www.esdifferent.com/difference-between-local-and-global-maximum.

[8] Google. 12. Dez. 2018. URL: https://www.tensorflow.org/.

[9] Simon O. Haykin. *Neural Networks and Learning Machines (3rd Edition)*. Pearson, 2008. ISBN: 978-0-13-147139-9. URL: https://www.amazon.com/Neural-Networks-Learning-Machines-3rd/dp/0131471392?SubscriptionId=AKIAIOBINVZYXZQZ2U3A&tag=chimbori05-20&linkCode=xm2&camp=2025&creative=165953&creativeASIN=0131471392.

[10] IMDb. *Matrix*. 7. Jan. 2019. URL: https://www.imdb.com/title/tt0133093/.

[11] IMDb. *Tron*. 7. Jan. 2019. URL: https://www.imdb.com/title/tt0084827/.

[12] Keras-Team. 12. Dez. 2018. URL: https://github.com/keras-team/keras.

[13] Joel Lehman und Risto Miikkulainen. „General Video Game Playing as a Benchmark for Human-Competitive AI“. In: *AAAI-15 Workshop on Beyond the Turing Test*. 2015. URL: http://nn.cs.utexas.edu/?lehman:aaai15.

[14] leonardoaraujosantos. *Arificial Inteligence*. Hrsg. von gitbook. 4. Okt. 2018. URL: https://leonardoaraujosantos.gitbooks.io/artificial-inteligence/content/pooling_layer.html.

[15] Gustavo Machado. *ML Basics: supervised, unsupervised and reinforcement learning*. 6. Okt. 2016. URL: https://medium.com/@machadogj/ml-basics-supervised-unsupervised-and-reinforcement-learning-b18108487c5a.

[16] Tambet Matiisen. *Demystifying Deep Reinforcement Learning*. 19. Dez. 2015. URL: https://neuro.cs.ut.ee/demystifying-deep-reinforcement-learning/.

[17] Ryszard S. Michalski, Jaime G. Carbonell und Tom M. Mitchell, Hrsg. *Machine Learning*. Springer Berlin Heidelberg, 1983. DOI: 10.1007/978-3-662-12405-5.

[18] Tariq Rashid. *Make Your Own Neural Network*. CreateSpace Independent Publishing Platform, 2016. ISBN: 978-1530826605. URL: https://www.amazon.com/Make-Your-Own-Neural-Network/dp/1530826608?SubscriptionId=AKIAIOBINVZYXZQZ2U3A&tag=chimbori05-20&linkCode=xm2&camp=2025&creative=165953&creativeASIN=1530826608.

[19] Andrew G. Barto Richard S. Sutton. *Reinforcement Learning*. The MIT Press, 11. Mai 1998. 338 S. ISBN: 0262193981. URL: https://www.ebook.de/de/product/3643662/richard_s_sutton_andrew_g_barto_reinforcement_learning.html.

[20] Chris Rupp. *Requirements-Engineering und -Management*. Hanser Fachbuchverlag, 2009. ISBN: 9783446418417. URL: https://www.amazon.com/Requirements-Engineering-Management-Chris-Rupp/dp/3446418415?SubscriptionId=AKIAIOBINVZYXZQZ2U3A&tag=chimbori05-20&linkCode=xm2&camp=2025&creative=165953&creativeASIN=3446418415.

[21] Stuart Russell. *Artificial Intelligence: A Modern Approach*. Pearson Education India, 2015. ISBN: 9789332543515. URL: https://www.amazon.com/Artificial-Intelligence-Approach-Stuart-Russell/dp/9332543518?SubscriptionId=AKIAIOBINVZYXZQZ2U3A&tag=chimbori05-20&linkCode=xm2&camp=2025&creative=165953&creativeASIN=9332543518.

[22] Tim Salimans u. a. „Evolution Strategies as a Scalable Alternative to Reinforcement Learning“. In: (10. März 2017). arXiv: 1703.03864v2 [stat.ML].

[23] Daniel Shiffman. *The Nature of Code: Simulating Natural Systems with Processing*. The Nature of Code, 2012. ISBN: 0985930802. URL: https://www.amazon.com/Nature-Code-Simulating-Natural-Processing/dp/0985930802?SubscriptionId=AKIAIOBINVZYXZQZ2U3A&tag=chimbori05-20&linkCode=xm2&camp=2025&creative=165953&creativeASIN=0985930802.

[24] Mayfield Brain & Spine. *Anatomy of the Brain*. 26. Okt. 2018. URL: https://mayfieldclinic.com/pe-anatbrain.htm.

[25] Computer Vision Machine Learning Team. *An On-device Deep Neural Network for Face Detection*. Nov. 2017.

[26] MSA Technosoft. *Artificial Neural Network | Types | Feed Forward | Feedback | Structure | Perceptron | Machine Learning | Applications*. 21. Mai 2018. URL: https://msatechnosoft.in/blog/tech-blogs/artificial-neural-network-types-feed-forward-feedback-structure-perceptron-machine-learning-applications.

[27] Julian Teglius. 11. Jan. 2016. URL: http://togelius.blogspot.com/2016/01/why-video-games-are-essential-for.html.

[28] Avinash Sharma V. *Understanding Activation Funtions in Neural Networks*. 30. März 2017. URL: https://medium.com/the-theory-of-everything/understanding-activation-functions-in-neural-networks-9491262884e0.

[29] Daan Wierstra u. a. „Natural Evolution Strategies“. In: (22. Juni 2011). arXiv: 1106.4487v1 [stat.ML].

[30] Wikipedia. *Brain*. 27. Sep. 2018. URL: https://en.wikipedia.org/wiki/Brain.

[31] Wikipedia. *IQ classification*. 27. Sep. 2018. URL: https://en.wikipedia.org/wiki/IQ_classification.

[32] Wikipedia. *RNN*. 24. Okt. 2018. URL: https://de.wikipedia.org/wiki/Rekurrentes_neuronales_Netz.

[33] Wiktionary. *Nervenzelle*. 26. Okt. 2018. URL: https://de.wiktionary.org/wiki/Nervenzelle.