Neuronale Netzwerke mit genetischen Algorithmen

am Beispiel von Computerspielen

Wettbewerbsarbeit für
Schweizer Jugend forscht

Kantonsschule Sursee 2023/24

Autor
Flurin Graeff

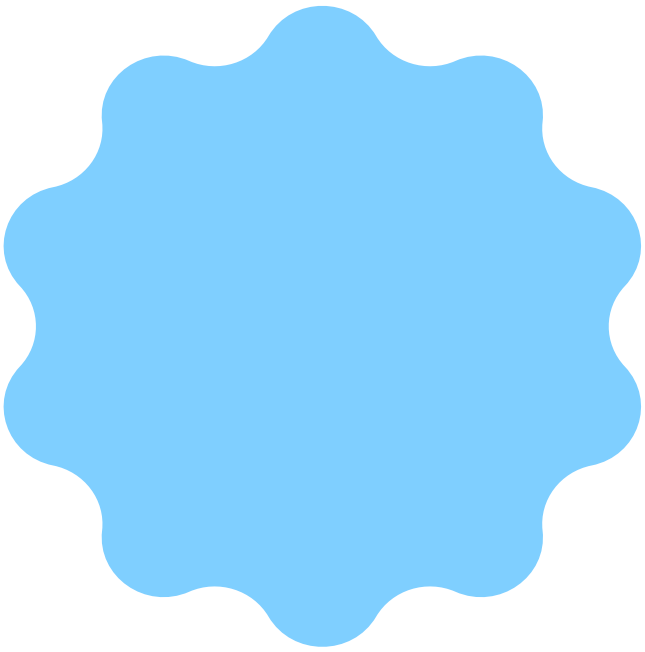
Betreuer Kantonsschule
Philipp Hurni

Betreuer SJF
Stefan Ganscha

«Eine jede Kunst
ist eine Nachahmung
der Natur.»

- Seneca[1]

[1] (Lucius Annaeus Seneca,
römischer Philosoph,
1 n. Chr. - 65 n. Chr.)

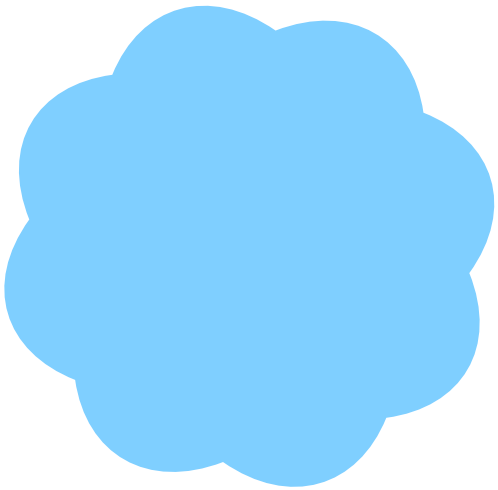


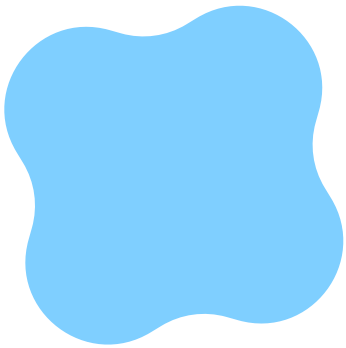
Abstract

Dank starker digitaler Weiterentwicklung und somit zunehmender Komplexität für computerbasierte Problemstellungen wird immer häufiger der Punkt erreicht, an dem es für einen Menschen fast unmöglich ist, die Lösung eines Problems mit einem einfachen Algorithmus zu formulieren. Neuronale Netzwerke schaffen hierbei Abhilfe. Die dem menschlichen Gehirn nachempfundene Datenstruktur ermöglicht eigenständiges maschinelles Lernen dank selbstständiger Klassifizierung von zur Verfügung gestellten Daten. Dadurch lassen sich computerbasiert weitaus komplexere Problemstellungen lösen.

Damit Neuronale Netze eingesetzt werden können, müssen sie allerdings trainiert werden. Meistens existieren für das Training schon Trainingsdaten – aber nicht immer. In letzteren Fällen eröffnen genetische Algorithmen als effektive Optimierungsverfahren neue Möglichkeiten. Diese bieten bedeutende Vorteile im Vergleich zu Neuronalen Netzwerken.

Diese Arbeit präsentiert die Entwicklung einer Bibliothek, die die Stärken von Neuronalen Netzwerken bezüglich maschinellen Lernens und die Optimierung durch genetische Algorithmen auf elegante Weise vereint. Diese Bibliothek wird exemplarisch an verschiedenen Computerspielen angewandt. Zusätzlich wird die Bibliothek auf einer interaktiven Website visualisiert und zur praktischen Anwendung zur Verfügung gestellt.





Inhaltsverzeichnis

1. Vorwort

Im Sommer 2022 erreichte ich bei einer freizeitbasierten Programmierarbeit zur Erstellung eines Computerspiels den Punkt, an dem ich an die Grenzen meiner damaligen Mathematik- und Programmierkenntnisse kam. Zur Umsetzung der Spielgegner in einem selbstentwickelten Computerspiel fehlten externe Bibliotheken. Einen eigenen Algorithmus zu implementieren, sah ich nicht als Option an. Einerseits wäre durch meine Vorstellung einer selten in anderen Spielen angetroffenen Spielgegnerintelligenz eine neue Stufe an Komplexität geschaffen worden, andererseits mangelte es mir in dieser Zeit noch stark an langjähriger Erfahrung, welche ein wichtiger Grundbaustein eines jeden Programmierers ist. Somit musste eine andere Lösung her. Bei meinen Recherchen stiess ich schon bald auf das Prinzip des maschinellen Lernens.

Mein erster Versuch begann damit, Googles Machine-Learning-Modell *TensorFlow* zu integrieren. Ich stellte aber schnell fest, dass ich nicht über die nötigen Trainingsdaten verfügte und damit eine solche Vorgehensweise nutzlos war. Das Prinzip ohne Trainingsdaten namens «Reinforcement Learning» konnte auch ausgeschlossen werden, da der Zeitaufwand zur Erarbeitung des Verständnisses der Algorithmen zu hoch war. Bis heute ist es für mich wichtig, jegliche externen Tools so gut wie möglich zu verstehen, bevor ich sie anwende, denn ansonsten ist es schwierig deren volles Potential auszuschöpfen. Zufälligerweise stiess ich wenige Tage später auf ein Projekt von *Rafael Matsunaga* namens *Genetic Cars 2* [1], welches mir ein ganz anderes Gebiet des computerbasierten Lernens präsentierte. Nun war es nicht weit hergeholt, das mich faszinierende Prinzip eines Neuronalen Netzwerkes und die genetischen Algorithmen wie aus *Matsunaga*'s Projekt zu kombinieren. Solch eine Bibliothek zu finden war aber zu diesem Zeitpunkt fast unmöglich. Entweder existierten unbeendete, benutzerunfreundliche oder stark leistungsintensive Produkte oder diese Produkte wichen sehr stark von meiner Vorstellung ab und hatten deshalb keinen konkreten Nutzen für mich.

Im späteren Sommer absolvierte ich ein Praktikum bei der Firma *W-Vision* in einem Webentwicklungsteam. Während vier Wochen entwarf ich eine komplette Website inklusive Backend. Die Möglichkeiten, die mir JavaScript und CSS-Animationen boten, faszinierten mich. Schon bald wurde JavaScript meine präferierte Programmiersprache. Sie ermöglichte mir meine Projekte für fast alle Geräte zu erstellen. Zudem war sie als prozedurale und objektorientierte Programmiersprache einsetzbar. Die Mehrheit meiner Ideen konnten damit umgesetzt werden. Abgesehen davon, dass durch das absolvierte Praktikum die Fächerwahl für die Maturaarbeit schon um vieles vereinfacht wurde, beeinflusste es auch Aspekte in diesem Projekt wie beispielsweise die Umsetzungs-Programmiersprache. Denn nun kannte ich das Potential von JavaScript.

Die Idee für meine Maturaarbeit war geboren: einen Machine-Learning-Algorithmus implementiert als JavaScript-Bibliothek, welcher auf den Prinzipien des Neuronalen Netzwerkes und den genetischen Algorithmen basiert. Die Bibliothek sollte leicht einzubinden sein, über eine gute Dokumentation verfügen und eine hohe Benutzerfreundlichkeit aufweisen.

2. Problemstellung

Das Ziel dieser Arbeit ist es, als Projekt eine Programmierbibliothek zu entwickeln. Dazu gehört das selbstständige Aneignen von benötigten theoretischen Grundlagen, das Planen der Umsetzung des Projektes, sowie die tatsächliche Umsetzung auf Codebasis. Anschliessend soll das Endprodukt des Projektes kritisch analysiert und ausgewertet werden.

Da es sich für mich um ein noch eher unbekanntes Gebiet handelt, besteht ein Grossteil der Arbeit aus der Erarbeitung von theoretischen Grundlagen. Die theoretischen Grundlagen basieren hauptsächlich auf zwei Lernmaterialien [2] [3].

Nach dem Erarbeiten der theoretischen Grundlagen wird das Projekt und dessen Aufbau bis ins Detail geplant, was den folgenden Umsetzungsprozess um vieles erleichtert. Die Programmierarbeit besteht einerseits aus dem Umsetzen von Methoden, die auf den durch die Planung erstellten Entwürfen basieren, andererseits auch aus dem Beheben von während der Umsetzung entstehenden Fehlern. Da sich das Projekt mit seiner Komplexität vom vor Arbeitsbeginn existierenden Vorwissen weit wegbewegt, muss während der Umsetzungsphase immer wieder neues Wissen erlernt werden. Auch das erschwerte Fehlerfinden durch die limitierte Visualisierbarkeit des Produktes während der Entwicklung, führt zum Erlernen neuer Methoden zum Aufspüren dieser Fehler. Schliesslich benötigen das Optimieren und Komprimieren des Codes auch weiteres Erarbeiten von neuem Wissen.

Die Arbeit besteht somit nicht nur aus reiner Programmierarbeit, sondern ebenso aus dem Aneignen von neuem Wissen.

2.1 Inhalt des Projektes

Der Hauptfokus des Projektes liegt auf dem Entwickeln einer Bibliothek. Es handelt sich dabei um mehrere Algorithmen, die in den Bereich des maschinellen Lernens einzuordnen sind. Die Bibliothek kombiniert das häufig in diesem Bereich eingesetzte Neuronale Netzwerk sowie den dazugehörenden Backpropagation-Algorithmus mit einem auf genetisch-basierten Methoden aufgebauten Algorithmus. Während beim Neuronalen Netzwerk und der Backpropagation der Fokus darauf liegt, vorhandene Entwürfe als Programmiercode effizient umsetzen zu können, ist beim genetischen Algorithmus auch das Planen und Erstellen eines eigenen Entwurfes Teil des Projektes.

Das Projekt beinhaltet neben der Bibliothek auch eine zur Bibliothek gehörende ausführliche Dokumentation sowie mehrere Beispielanwendungen. Beides bietet dem Benutzer der Bibliothek einen Einblick in deren Fähigkeiten sowie Hilfe zur sinnvollen Nutzung der Bibliothek. Die Dokumentation wie auch die Beispielanwendungen werden durch eine designtechnisch einladende, im Projekt inkludierte Website für den Benutzer visuell dargestellt. Sie präsentiert das Projekt auf eine ansprechende Weise.

Auch die Bibliothek selbst wird benutzerfreundlich aufgebaut. Einerseits ist der Code Open Source, wodurch dem Benutzer ein erweitertes Verständnis geboten wird. Der Codeinhalt der Bibliothek ist so aufgebaut, dass er dank klar gewählten Variablen- und Methodennamen eine gute Lesbarkeit aufweist.

2.1.1 Fähigkeiten der Bibliothek

Die Bibliothek soll sich von anderen Produkten abheben und ihre eigenen Stärken aufzeigen können. Um dies erreichen zu können, wird der Fokus der Bibliothek auf ganz bestimmte Bereiche gelegt. Die Bibliothek soll hauptsächlich Personen mit wenigen Jahren Programmiererfahrung ansprechen, die beispielsweise ihr erstes Mal mit maschinellem Lernen in Kontakt kommen. Durch die gut umgesetzte Dokumentation, die Beispielanwendungen und die gute Implementierbarkeit der Bibliothek, lässt sich

die Bibliothek mit minimalem Aufwand in eigene Projekte einbinden. Durch die gewählte Umsetzungssprache der Bibliothek, sowie der starken Codekomprimierung liegt der Anwendungsbereich in der Webentwicklung. Da die Umsetzungssprache JavaScript vor allem als Frontend-Programmiersprache eingesetzt wird, ist die Bibliothek mit ihrer kleinen Dateigrösse, welche die Ladezeiten einer Website stark beeinflussen kann, und mit ihren optimierten Matrixberechnungen für nicht sehr leistungsstarke Geräte, wie Smartphones optimiert. Das Ausführen der Bibliothek in einem Webbrowser ist somit problemlos möglich.

Der Fokus der Bibliothek unterscheidet sich stark von anderen, inhaltlich vergleichbaren Bibliotheken. Andere gut dokumentierte Bibliotheken sind meist für hochkomplexe Problemstellungen ausgelegt und laufen nur auf speziell für maschinelles Lernen optimierter Hardware.

3. Theoretische Grundlagen

Zum Verständnis des Aufbaus und der Umsetzungsidee des Projektes sind viele theoretische Grundlagen nötig. Sie stammen entweder aus schon vorhandenem Vorwissen oder basieren auf den folgenden Lernmaterialien:

Für das Fachgebiet Neuronale Netzwerke wird auf das Werk «*Make Your Own Neural Network*» von *Tariq Rashid* [2] zurückgegriffen. Das Ziel ist, das ganze benötigte Wissen anhand von einem ähnlichen Beispiel mit einer anderen Zielsetzung und mit einer anderen Umsetzungssprache zu erlernen und dies dann auf eine eigene Weise selbst umzusetzen. Für das Erlernen der Funktionsweise von genetischen Algorithmen wird auf das Werk von *S. N. Sivanandam* und *S. N. Deepa* mit dem Namen «*Introduction to Genetic Algorithms*» [3] gesetzt. Das Werk behandelt weit komplexere Themen als nötig, stellt aber auch alle Grundideen von genetischen Algorithmen vor, wodurch sich ein eigener genetischer Algorithmus planen und umsetzen lässt.

Um dem Leser die Inhalte besser vermitteln zu können, wird die Arbeit mit Inhalten aus externen Internetquellen ergänzt. Alle folgenden Informationen dieses Kapitels enthalten keine selbstentwickelten Ideen. Um das Verständnis der darauffolgenden Kapitel zu vereinfachen, werden hier die für das Projekt benötigten Grundlagen auf eine leicht verständliche Weise erläutert.

3.1 Neuronale Netzwerke

3.1.1 Inspiration Natur

Mechanismen des maschinellen Lernens orientieren sich vielfach an der Natur. Die Prinzipien Neuronaler Netzwerke sind ebenfalls auf biologische Vorbilder zurückzuführen. Das aus der Neurowissenschaft stammende (organische) neuronale Netz, so wie es sich im menschlichen Gehirn befindet, bildet die Grundlage für das (Künstliche) Neuronale Netzwerk (*KNN*). Auch das *KNN* ist deshalb eine Ansammlung an Neuronen, die miteinander verbunden sind – ähnlich wie die Synapsen im organischen neuronalen Netz die Neuronen miteinander verbinden. Die Grundidee des *KNN*s ist es, eine dem Menschen vergleichbare Lernmethode zu entwickeln, mit der sich praktisch jede Funktion berechnen lässt.

3.1.2 Einzelnes Neuron

Die Grundbausteine des *KNN*s sind Knoten, meist auch Neuronen genannt. Sie nehmen Informationen von aussen oder von anderen Neuronen auf. Diese Informationen werden dann im Neuron verarbeitet und anschliessend an andere Neuronen oder nach aussen wieder ausgegeben. Meist besitzt ein Neuron mehr als einen Eingang. Der Ausgabewert hingegen ist zu allen weiteren Empfängern gleich.

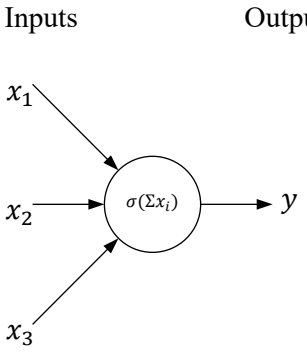


Abb. 1 Grundstruktur eines einzelnen Neurons

Um den Ausgabewert zu berechnen, werden alle Inputs (x_i) miteinander addiert und darauf eine Aktivierungsfunktion angewandt ($\sigma(x)$). So lässt sich der Ausgabewert problemlos berechnen mit der folgenden Formel (n steht für die Anzahl an Inputs):

$$y = \sigma(\sum_{i=1}^{n} x_i)$$

3.1.2.1 Aktivierungsfunktionen

Eine Aktivierungsfunktion ist für die Verarbeitung der erhaltenen Daten verantwortlich. Einerseits sorgt sie dafür, dass der Output (y) im gewünschten Bereich bleibt (meist zwischen 0 und 1 oder -1 und 1), da eine grosse Zahl grössere Auswirkungen im finalen Netzwerk haben kann als eine kleine. Andererseits kann die Funktion nebst der Normalisierung der Inputs auch einen Schwellenwert definieren, der die Inputs als bedeutungslos erkennen kann. Somit ergeben sich verschiedene Arten von Aktivierungsfunktionen:

- **Schwellenwertfunktion**: Nimmt nur die Werte 0 oder 1 an. Sobald die Inputs grösser als der gewünschte Wert sind, wird 1 ausgegeben. Ansonsten ist der Output 0.
- **Lineare Funktion**: Wendet einen bestimmten Faktor auf die Inputs an.
- **Stückweise lineare Funktion**: Eine lineare Funktion mit Begrenzung auf einen bestimmten Bereich wie zum Beispiel zwischen 0 und 1 .
- **Normalisierungsfunktion**: Gibt immer einen Wert $0 < y < 1$ zurück. Der Input wird in den Bereich 0 bis 1 normalisiert. Beispielfunktionen sind $sigmoid$ oder $tanh$. [4]

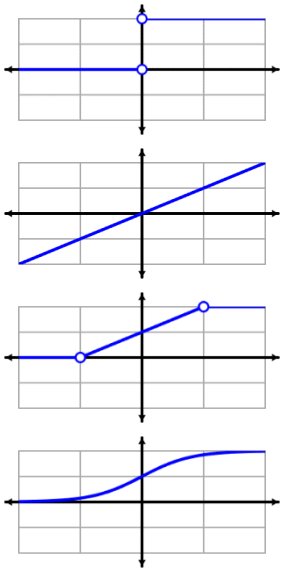


Abb. 2 Funktionen [5]

In diesem Projekt wird alleinig mit der Funktion $sigmoid$ gearbeitet:

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

Normalisierungsfunktionen bieten die Möglichkeit in der später noch erklärten Backpropagation eine Rückverfolgung anhand der erhaltenen Outputs durchzuführen, da sie injektiv (eineindeutig) sind. Denn bei Schwellenwertfunktionen und stückweise linearen Funktionen können verschiedene Inputs den gleichen Output ergeben (eindeutig). Dadurch ist mit einem bekannten Output der Input nicht garantiert bestimmbar.

3.1.3 Komplettes Netzwerk

Während man schon ein einzelnes Neuron als Netzwerk bezeichnen kann und unter dem Namen Perceptron einfache Berechnungen wie eine lineare Trennung zweier Datengruppen durchführen kann, wird im maschinellen Lernen vor allem das Prinzip des Feed Forward Neural Network benutzt. Als die am häufigsten verwendete neuronale Netzwerkarchitektur erlaubt es vor allem ein breites Anwendungsspektrum wie auch das Lösen von komplexeren Problemstellungen. [6]

Grundlegend ist das Feed Forward Neurale Netzwerk in mehrere Schichten, genannt Layer, unterteilt. Pro Layer befindet sich mindestens ein Neuron. Für jedes Neuron eines Layers existieren zu allen Neuronen des nächsten Layers Verbindungen. Innerhalb eines Layers existieren keine Verbindungen zwischen den Neuronen. Der erste Layer wird Input-Layer genannt. Jedes dieser Neuronen im ersten Layer besitzt genau einen Input. Dieser Input kommt von ausserhalb des Netzwerkes. Die Neuronen im letzten Layer, genannt Output-Layer, geben ihren Output ausserhalb des Netzwerkes weiter.

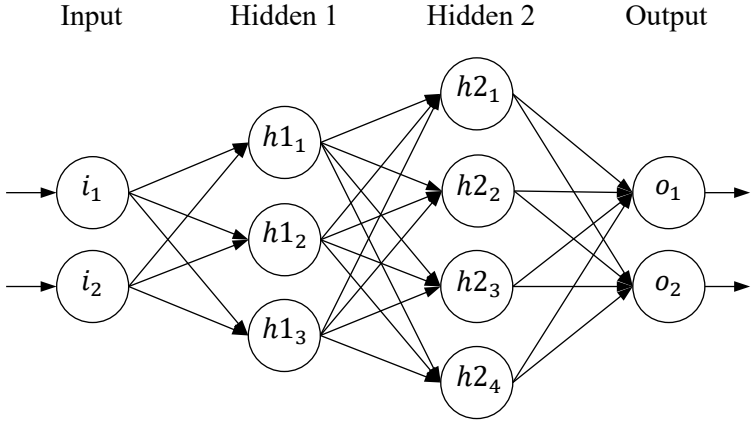


Abb. 3 Grundstruktur eines Feed Forward Neural Network

3.1.3.1 Gewichte

Um eine Lernfähigkeit zu ermöglichen, wird zu jeder Verbindung ein Gewicht (w) hinzugefügt (siehe *Abb. 4*). Diese Gewichte fügen dem Output aus dem vorherigen Neuron einen bestimmten Faktor hinzu. Somit lässt durch Anpassung der Gewichte über das ganze Netzwerk hinweg der Netzwerkoutput für bestimmte Inputs beeinflussen. Jedes Gewicht ist eine Zahl zwischen -1 und 1.

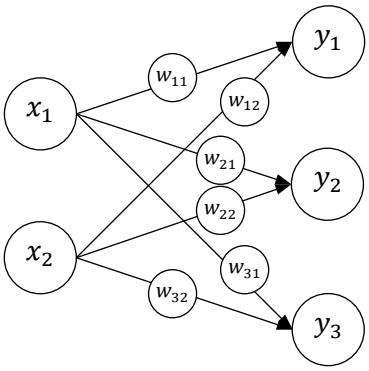


Abb. 4 Neuronales Netzwerk mit Gewichten

Alle Gewichte zwischen zwei Layern können als Matrix notiert werden, was weitere Berechnungen vereinfachen kann. Ein Gewicht wird als w_{ij} bezeichnet. i ist der Index des Neurons, welches den zu übertragenden Wert entgegennimmt und j der Index des Neurons, das den Wert abgibt. Eine Darstellung in einer Matrix sieht wie folgt aus:

$$w_{ij} = \begin{bmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \\ w_{31} & w_{32} \end{bmatrix}$$

Die Gewichte können problemlos in die Berechnungsformel $y = \sigma(\sum_{i=1}^{n} x_i)$ eines Neurons eingebaut werden. Zu jedem Input wird das zugehörige Gewicht hinzugefügt. Somit ergibt sich für ein Neuron i :

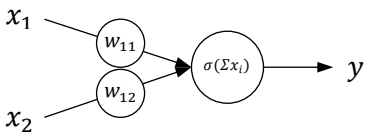


Abb. 5 Einzelnes Neuron inkl. Gewichte

$$y = \sigma(\sum_{j=1}^{n} w_{ij} x_j)$$

Dank der Matrix-Multiplikation lässt sich diese Formel in einem einzelnen Rechnungsschritt für einen ganzen Layer anwenden. Die Summe innerhalb der $sigmoid$-Funktion ist ausgeschrieben $w_{i1}x_1 + w_{i2}x_2 + w_{i3}x_3 + \cdots + w_{in}x_n$. Das Produkt einer Matrix-Multiplikation gibt genau das gleiche Resultat zurück, wenn die Inputs (x) in einen Vektor gesetzt werden. Somit sähe das Beispiel aus *Abb. 4* wie folgt aus:

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \begin{bmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \\ w_{31} & w_{32} \end{bmatrix} \times \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} w_{11}x_1 + w_{12}x_2 \\ w_{21}x_1 + w_{22}x_2 \\ w_{31}x_1 + w_{32}x_2 \end{bmatrix}$$

Um den Output eines Neuronalen Netzwerkes anhand der gegebenen Inputs bestimmen zu können, muss nach jeder Berechnung der Outputs der Neuronen in einem Layer eine Matrix-Multiplikation mit den dazugehörenden Gewichten stattfinden. Das Matrixprodukt kann dann als Input für die Neuronen des nächsten Layers verwendet werden.

3.1.4 Bias-Neuron

Das Bias(-Neuron) ist eine Konstante im Neuronalen Netzwerk, die zum Produkt der Inputs und Gewichte in einem Neuron hinzugefügt wird, bevor die Aktivierungsfunktion durchlaufen wird.

3.1.4.1 Stellenwert des Bias

Das Bias behebt eine starke Limitierung der meisten Aktivierungsfunktionen und kann so das Neuronale Netzwerk darin unterstützen, schneller und mit einer höheren Wahrscheinlichkeit zu einer Lösung zu kommen. Grundsätzlich ermöglichen die Gewichte den Output zu manipulieren, dies aber nur auf eine limitierte Weise.

Dargestellt an einer $sigmoid$-Funktion, die durch die folgende Gleichung dargestellt wird:

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

Wird die Variable x durch die Formel für einen einzelnen Input ersetzt, ergibt das:

$$\sigma(x) = \frac{1}{1 + e^{-(w*x)}}$$

x gilt als externer Input und w als Gewicht, das es zu optimieren gilt. Wenn der Wert von w frei verändert wird, ergibt sich daraus ein Graph wie beispielsweise den aus *Abb. 6*. Unabhängig vom Wert des Gewichts w führt dies immer dazu, dass sich der Übergangspunkt von $y < 0.5$ zu $y > 0.5$ oder umgekehrt immer bei $x = 0$ befindet. Somit ist bei einem Inputwert $x = 0$ der Ausgabewert immer $y = 0.5$.

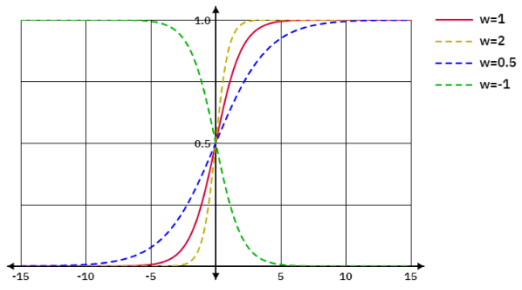


Abb. 6 [5]

Während also das Gewicht nur die Steilheit der Kurve verändert, kann ein Wert dazu addiert werden, der uns eine Verschiebung auf der x-Achse ermöglicht. Hierbei handelt es sich um das Bias (b):

$$\sigma(x) = \frac{1}{1 + e^{-(w*x+\boldsymbol{b})}}$$

Wird das Bias verändert, aber die Gewichte bleiben konstant, so ergeben sich Graphen vergleichbar mit denen aus *Abb. 7*.

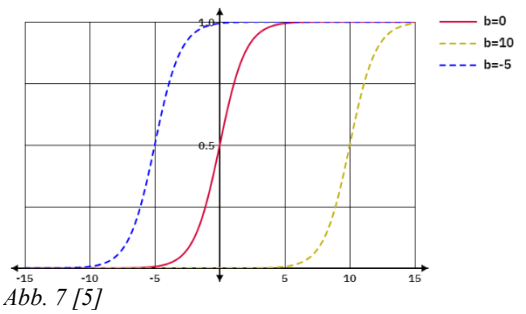


Abb. 7 [5]

Doch was bedeuten diese Auswirkungen auf das Endresultat eines auf einem Neuronalen Netzwerk basierenden Algorithmus? Auf ein Perceptron (*KNN* mit nur einem einzelnen Neuron) angewandt, bei einem Versuch zwei Gruppen auf einer zweidimensionalen Fläche linear zu trennen (zwei Inputs: x-Achse und y-Achse) ist man beim Verzicht auf ein Bias limitiert auf eine lineare Linie, welche den Punkt $[0,0]$ zwangsläufig durchläuft (siehe *Abb. 8*).

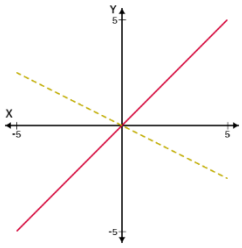


Abb. 8 [5]

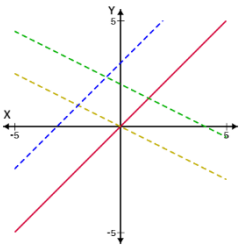


Abb. 9 [5]

Sobald ein Bias implementiert wird, kann diese Limitierung aufgehoben werden (siehe *Abb. 9*).

3.1.4.2 Implementierung des Bias

Das Bias lässt sich wie ein einzelnes Neuron ins *KNN* implementieren. Es ist exklusiv mit den Neuronen des darauffolgenden Layers verbunden. Das Bias hat grundsätzlich den Wert 1. Das Gewicht zwischen Bias und Neuron ist somit für die Wirkung des Bias verantwortlich. Die Berechnung in einem Neuron findet wie folgt statt:

$$y = \sigma(w_1 * x_1 + \cdots + w_b * 1)$$

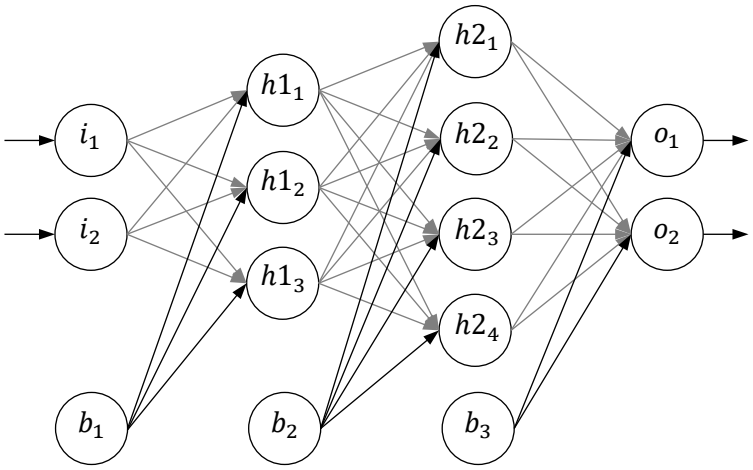


Abb. 10 Neurales Netzwerk mit Bias

3.2 Backpropagation

Ein Neuronales Netzwerk mit einem Feed Forward Algorithmus verfügt selbst nicht über die Fähigkeit etwas zu erlernen. Einzig wird der Input anhand der Gewichte verändert. Der Wert der Gewichte wird im Normalfall zufällig gesetzt. Das Ziel ist nun, die Gewichte so zu verändern, dass der gewünschte Output anhand des Inputs ausgegeben wird. Hier kommt der Backpropagation Algorithmus (*BP*) ins Spiel. Er sorgt für die Verbesserung der Ausgabe des Neuronalen Netzwerkes während des Trainingsprozesses. Er wendet das Gegenteil des Feed Forward Algorithmus an, um anhand eines gegebenen Outputs die falschliegenden Gewichte zu optimieren.

3.2.1 Gradient Descent

Das Gradientenverfahren (engl. Gradient Descent) stammt aus dem mathematischen Teilgebiet der Numerik und ist ein effizienter Weg, sich so schnell wie möglich einem Minimum einer Funktion zu nähern.

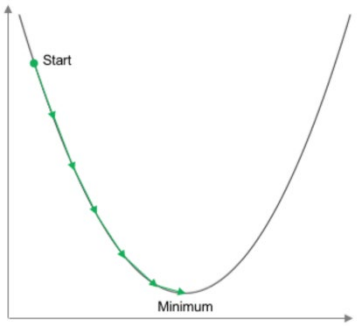


Abb. 11 Gradient Descent [8]

Dem Gradientenverfahren liegt die Idee zugrunde, idealerweise ein globales, ansonsten ein lokales Minimum einer Funktion exakt zu erreichen, indem Schritte grössenabhängig von der Steigung der Funktion getätigt werden. Ist die Steigung gross, wird ein grosser Annäherungsschritt getätigt. Ist die Steigung hingegen klein, so sind auch die Schritte klein. Dadurch kann eine Annäherung in Richtung Minimum weiter stattfinden, ohne dieses zu verpassen (siehe *Abb. 11*). Daraus lässt sich die folgende Formel formen:

$$b = a - \gamma \nabla f(a)$$

Hier steht b für die als nächstes anzutretende Position, a für die aktuelle Position. Das Gamma γ steht für die Lernrate, vorstellbar als die Schrittweite, und $\nabla f(a)$ für den Gradienten der Zielfunktion $f(a)$ an der Stelle a. Ein Gradient ist ein Vektor, der die Richtung des steilsten Anstiegs einer Funktion von einem bestimmten Punkt aus angibt. Er zeigt an, wie sich $f(a)$ ändert, wenn wir den Wert von a leicht verändern. Da der Gradient in die Richtung des steilsten Anstiegs zeigt, bewegen wir uns in die entgegengesetzte Richtung, um den steilsten Abstieg zu erreichen. Darum wird der Gradient von a abgezogen.

Der Erfolg einer Annäherung lässt sich anhand des Verlustes in den Output-Werten eines Neuronalen Netzwerkes bestimmen. Haben alle Gewichte im Gradient Descent das globale Minimum erreicht, so entspricht der Output des Neuronalen Netzwerkes exakt dem gewünschten Output. Der Verlust bezeichnet die Differenz zwischen erhaltenem Output und gewünschtem Output. Ein perfektes Neuronales Netzwerk hat einen Verlust von 0. Der Verlust eines einzelnen Neurons wird Error genannt.

Nun existiert nur noch die Herausforderung die perfekte Lernrate herauszufinden. Ist die Lernrate zu hoch, wird das Ziel nicht genau getroffen und der Verlust kann nicht richtig an 0 angenähert werden. Bei einer zu tiefen Lernrate braucht der Prozess der Suche nach dem Minimum sehr lang und ist somit ineffizient. Ist die Lernrate um vieles zu hoch, kann es sein, dass sich das Resultat über die Zeit verschlechtert, erkennbar dadurch dass der Verlust sogar ansteigt (siehe *Abb. 12*).

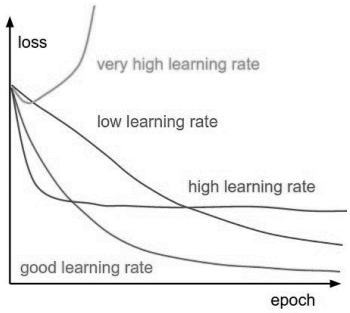


Abb. 12 Verlust nach Lernrate [7]

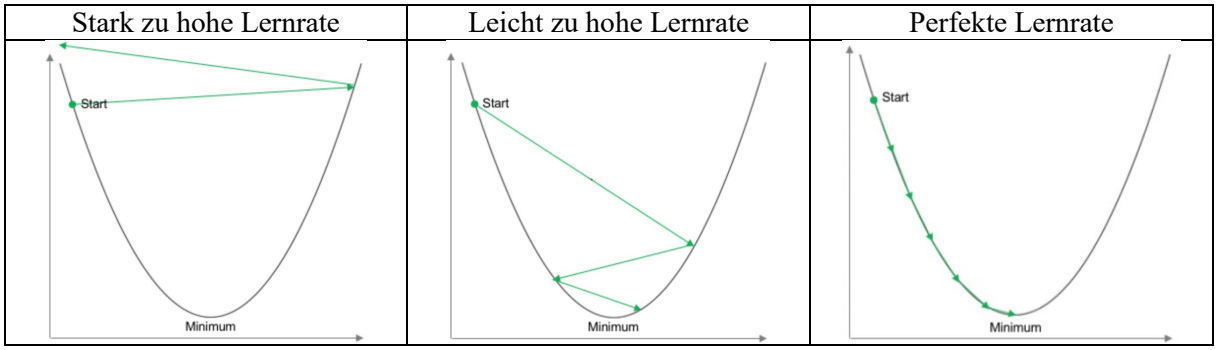


Abb. 13 Verschiedene Lernraten [8]

3.2.2 LMS-Algorithmus

Der Least-Mean-Squares-Algorithmus (LMS-Algorithmus), auch Delta-Regel genannt, bildet den ersten Schritt der Backpropagation und wendet das Prinzip des Gradient Descent erstmals an.

Mit der uns durch den Gradient Descent bekannten Formel können wir im Anwendungsbereich des Neuronalen Netzwerkes wenig anfangen. Es fehlt uns jegliche Information über die Funktion, an der wir den Gradient Descent anwenden und somit lässt sich auch kein Gradient bestimmen. Als Ersatz für den Gradienten wird ein Error verwendet zusammen mit dem Netzinput, erhalten durch die Feed Forward Propagation. Der Netzinput bezeichnet das Resultat aller Inputs multipliziert mit deren Gewichten, aber vor dem Einsatz der Aktivierungsfunktion. Um Berechnungsschritte zu vereinfachen, ist es zusätzlich sinnvoll, nur die Differenz zwischen zwei Schritten zu berechnen. Aus $b = a - \gamma * \nabla f(a)$ ersetzen wir $\nabla f(a)$ mit dem Error und dem Netzinput, zusammen genannt Delta-Wert δ. Wenn wir nun anstatt b zu berechnen nur die Differenz zwischen b und a bestimmen, erhalten wir:

$$\Delta w_{ij} = \gamma * \delta$$

Da die Differenz die Veränderung eines Gewichtes im *KNN* ist, nennen wir sie Δw_{ij}. i und j stehen für die angebundenen Neuronen.

Der Netzinputwert ist für uns in reinem Zustand nutzlos, da der Error auf Werten basiert, die die Aktivierungsfunktion durchlaufen haben. Um dies zu beheben, wird die Ableitung der verwendeten Aktivierungsfunktion $sigmoid$ auf den Netzinputwert angewandt. $\sigma'(x)$ lässt sich leicht mit der Kettenregel bestimmen. Die Ableitung sieht wie folgt aus:

$$\sigma'(x) = \sigma(x) * (1 - \sigma(x))$$

Der Error hat die gleiche Auswirkung wie der Gradient auf die Schrittweite. Je kleiner der Error, desto kleiner wird die Gewichtsmanipulation sein. Zur Bestimmung des Errors verwenden wir den gewünschten Ausgabewert (d) und subtrahieren davon den Ausgabewert (y), den wir durch den FF-Algorithmus erhalten haben. Der Delta-Wert lässt sich dann wie folgt definieren (b steht für den Netzinputwert):

$$\delta_i = \sigma'(b_i) * (d_i - y_i)$$

Nun wird aber jedes Gewicht der Inputs gleich verändert, auch wenn nicht alle Inputs den Ausgabewert gleich stark beeinflussen. Somit wird noch ein Faktor hinzugefügt, der gleich gross ist, wie der dazugehörende Inputwert (a) (ohne Multiplikation mit Gewicht). Die Delta-Regel sieht nun wie folgt aus:

$$\Delta w_{ij} = \gamma * \delta_i * a_j$$

3.2.3 Anwendung auf mehrere Layer

Die Delta-Regel ist problemlos auf das Output-Layer des *KNN*s anwendbar. Für alle weiteren Layer ist aber der Error unbekannt. Somit kommen wir zum Kernelement der Backpropagation. Der Bestimmung der Error aller Layer. Die Grundidee $\Delta w_{ij} = \gamma * \delta_i * a_j$ der Delta-Regel bleibt. Hingegen muss δ_i verändert werden. Neu lautet die Formel für δ_i:

$$\delta_i = \sigma'(b_i) * \sum_k (\delta_k * w_{ki})$$

Das bedeutet, dass der Error eines Neurons, das sich nicht im Output-Layer befindet, die Summe der Deltas mal den Verbindungsgewichten (zum Neuron, an dem wir den Error bestimmen) aller Neuronen

ist, die sich im darauffolgenden Layer (k) befinden. Mit dieser Methode lassen sich durch das Bestimmen der Deltas vom Ende des *KNN*s her alle Deltas und somit auch alle Gewichtsveränderungen im ganzen Netzwerk bestimmen. Der Input-Layer kann vernachlässigt werden, da sich vor dem Input-Layer keine Gewichte befinden.

3.3 Genetische Algorithmen

3.3.1 Natürliche Selektion

Ein häufig bei Optimierungsproblemen eingesetzter, zur Klasse der evolutionären Algorithmen gehörender Algorithmus ist der genetische Algorithmus (*GA*). Die an der von Charles Darwins Theorie der natürlichen Evolution inspirierte Aussortierungsmethode spiegelt den Prozess der natürlichen Selektion wider, bei der die fittesten Individuen gewählt werden, um die Grundlage der folgenden Generation zu legen.

Gleich wie in der natürlichen Selektion werden auch im *GA* die geeignetsten Individuen aus einer Population ausgewählt. Sie bringen dann Nachkommen hervor, die die Eigenschaften der Eltern erben. Gleichzeitig finden auch zufällige Mutationen statt, um neue bessere Eigenschaften in die Generation bringen zu können. Der gesamte Prozess wird über viele Generationen wiederholt, bis ein perfektes Individuum erschaffen wurde.

3.3.2 Ablauf des genetischen Algorithmus

Die Umsetzung des *GA* kann sich abhängig vom Einsatzgebiet stark unterscheiden. Hingegen trifft man immer wieder die gleichen Grundabläufe an. Der *GA* basiert immer auf aufeinanderfolgenden Populationen.

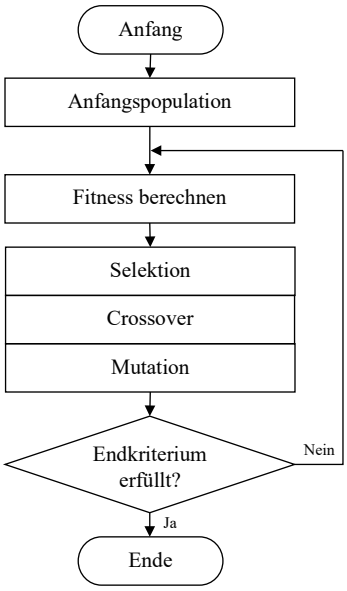


Abb. 14 Flowchart genetischer Algorithmus

Der Grundablauf des *GA* beginnt immer mit der Erzeugung einer Anfangspopulation. In dieser Population befinden sich mehrere Individuen, die alle eine Anzahl an Parameter, genannt Gene, aufweisen. In den meisten Anwendungsfällen wird jedes Gen durch einen Wert in Form eines binären Wertes oder einer Zahl repräsentiert. Anschliessend werden die Individuen ihrem Problem ausgesetzt. Eine Fitnessfunktion bestimmt den Erfolg jedes einzelne Individuum. Somit erhält jedes Individuum einen Fitnesswert. Anhand der Fitnesswerte wird nun entschieden, welche Individuen in der Selektion ausgeschieden werden. In der Selektion ausgewählte Individuen werden im Abschnitt Crossover miteinander kombiniert und somit neue Nachkommen erzeugt. Wie dieses Crossover umgesetzt wird, unterscheidet sich je nach Anwendungsbereich. Meist aber wird die Hälfte der Gene vom ersten Elternteil übernommen und die andere Hälfe vom zweiten. Schliesslich findet noch eine Mutation der Nachkommen statt. Einzelne Gene werden zufällig abgeändert. Dies ermöglicht die Erzeugung neuer, noch inexistenter Fähigkeiten. Anschliessend wird überprüft, ob eines der Individuen schon alle gewünschten Kriterien erfüllt und damit perfekt ist. Ist dies nicht erfüllt, wiederholt sich der Prozess (siehe *Abb. 14*).

4. Projektumsetzung

Beim Projekt handelt es sich vor allem um eine Programmierarbeit. Mit mehr als 7′000 Zeilen selbstgeschriebenem Code, verteilt auf acht Unterprojekte (*GitHub*-Repositories) und mit einem schlussendlichen Codeumfang von 38′000 Zeilen wurde das Projekt innerhalb eines halben Jahres anhand von über 200 *Commits* umgesetzt [9]. Zahlreiche verschiedene Versionen wurden innerhalb dieses Zeitraumes erstellt. Das Projekt selbst lässt sich in drei Bereiche aufteilen. Im Zentrum der Arbeit steht die implementierte JavaScript-Bibliothek. Sie basiert auf einem *GitHub*-Repository. In den zweiten Bereich lassen sich die zur Bibliothek dazu entwickelten Beispielanwendungen einordnen. Es handelt sich dabei um vier der acht Repositories. Zuletzt wurde noch eine Website erstellt die, umgesetzt durch drei Repositories, die ersten zwei Bereiche zusammenführt.

Das Projekt basiert nicht nur rein auf Programmierarbeiten. Nebst dem zeitlich grossen Programmieraufwand war auch die Planung der codebasierten Umsetzung Teil des Projektes sowie das Entwerfen eines eigenen genetischen Algorithmus.

4.1 Realisierung der Bibliothek

Zur Realisierung der Bibliothek wird die Programmiersprache JavaScript verwendet. Genau gesagt wird die Bibliothek basierend auf der JavaScript-Version ES6, genannt *ECMAScript 2015* umgesetzt. Somit werden grundsätzlich alle modernen Browser, sowie JavaScript-basierte Serverumgebungen wie beispielsweise *Node.js* unterstützt. Auch wurde im Verlauf der Implementierung eine externe Bibliothek namens *Mathjs* hinzugefügt. Sie erweitert die in JavaScript schon implementierte Mathematik-Bibliothek und fügt Funktionen wie Matrixberechnungen hinzu.

Zur Umsetzung und Programmierung der Bibliothek wird *WebStorm* von JetBrains verwendet. Die durch JetBrains für dieses Projekt kostenlos zur Verfügung gestellte Entwicklerumgebung enthält alle nötigen Tools, um eine hohe Effizienz während des Programmierprozesses zu erzielen. Der Programmiercode wurde schon während des Implementierungsprozesses auf *GitHub* veröffentlicht. Der Code der Bibliothek ist aufgelistet unter *SimpleJS-lib/lib/simple.js* (siehe: *9.1 Verzeichnisbaum*).

In diesem Abschnitt gezeigte Codebeispiele können von der tatsächlichen Implementierung in die Bibliothek abweichen, da originale Codeausschnitte dank der starken Komprimierung teils schlecht lesbar sind.

4.1.1 Programmierung des Neuronalen Netzwerkes

4.1.1.1 Struktur der Klasse NeuralNetwork

Auch wenn in JavaScript vor allem mit prozeduralen Abläufen und klassenunabhängigen Objekten gearbeitet wird, lassen sich in moderneren Versionen, wie beispielsweise ES6, auch Klassen definieren. Somit kann das Prinzip der objektorientierten Programmierung angewandt werden. Dies eignet sich besonders gut für Bibliotheken, da im Beispiel der Klasse des Neuralen Netzwerkes nur ein neues Objekt der Klasse `NeuralNetwork` generiert werden muss und durch Aufruf der in der Klasse enthaltenen Methoden alles, was die Klasse bietet, aufgerufen/durchgeführt werden kann. Auch lässt sich dadurch mit Instanzvariablen arbeiten.

Klasse **NeuralNetwork**

Parameter
`- input : number`
`- hidden : number`
`- output : number`
`- lr : number = 0.1`

Methoden
`+ ff(input : Array, debug : Boolean = false) : number[]`
`+ bp(desired_input : number[], desired_output : number[]) : any[]`
`+ _activation(x : number) : number`
`+ _dActivation(x : number) : number`

Die in der implementierten Bibliothek eingesetzte Klasse namens `NeuralNetwork` bildet, wie der Name schon sagt, die Struktur des Neuronalen Netzwerkes sowie auch den Backpropagation-Algorithmus. Als zu definierende Parameter gelten `input`, `hidden` und `output`. Sie geben an, wie gross das Neuronale Netzwerk sein wird. Durch `input` wird übergeben, wie viele Eingänge benötigt werden, durch `output` wie viele Ausgänge man erwartet. Der Parameter `hidden` definiert, wie gross der Hidden-Layer sein wird. Dies wird abhängig von der Komplexität der Problemstellung gewählt und ermöglicht Variabilität, um zwischen Leistung und Optimierungsgeschwindigkeit experimentieren zu können. Auch `lr`, als Abkürzung für die Lernrate (engl. Learning Rate), bietet dem Nutzer einen Spielraum, das Neuronale Netzwerk perfekt auf die spezifische Problemstellung abstimmen zu können (siehe *3.2.1 Gradient Descent*). Als Methoden stehen dem Nutzer `ff()` und `bp()` zur Verfügung. Möchte der Nutzer auf das trainierte Neuronale Netzwerk zugreifen und den passenden Output für seine Inputs erhalten, so kann er die Methode `ff()` verwenden. Sie nimmt als ersten Parameter alle Inputs als Array entgegen und gibt als Array die Outputs nach dem Durchlaufen des Neuronalen Netzwerkes wieder zurück. Der zweite Parameter ermöglicht es, eine Rückgabe von allen Zwischenschritten des Feed-Forward-Algorithmus anzufordern. Dies wird aber vor allem von bibliothekeigenen Methoden verwendet. Möchte man hingegen das Neuronale Netzwerk mit einem Datensatz trainieren, so verwendet man die `bp()`-Methode. Sie führt eine Backpropagation durch. Im ersten Parameter wird in einem Array ein Input angegeben und im zweiten Parameter der zum Input gehörende, gewünschte Output. Durch die Backpropagation führt diese Methode eine Optimierung der Gewichte durch. Die Optimierungsgeschwindigkeit ist abhängig von der definierten Lernrate.

Da JavaScript ES6 offiziell nicht über die Möglichkeit verfügt, Methoden privat zu setzen, wurden Methoden, auf die nicht zugegriffen werden soll, mit einem Unterstrich (‘_’) markiert. Die privatgestellten Methoden werden direkt von der Klasse selbst verwendet und sollen nicht von ausserhalb der Bibliothek benutzt werden.

4.1.1.2 Aufbau des Neuronalen Netzwerkes

Abgesehen von den durch den Konstruktor durchgegebenen Variablen, werden auch die Gewichte und die Bias als Instanzvariablen definiert. Gewichte werden als zwei Matrizen definiert, eine für die Gewichte zwischen den Input- und Hidden-Neuronen und eine für jene zwischen den Hidden- und

Output-Neuronen. Zur Erzeugung der Matrizen wird die *Mathjs* Bibliothek verwendet. Den Gewichten wird zuerst ein zufälliger Wert zwischen −1 und 1 zugewiesen.

```
// Gewichte definieren
this._wInputHidden = math.zeros(this.hidden, this.input+1);
this._wInputHidden = math.map(this._wInputHidden, () => Math.random() * 2 - 1);
this._wHiddenOutput = math.zeros(this.output, this.hidden+1);
this._wHiddenOutput = math.map(this._wHiddenOutput, () => Math.random() * 2 - 1);

// Bias definieren
this._bHidden = 1;
this._bOutput = 1;
```

Die Aktivierungsfunktion *sigmoid* wird in der Methode `_activation(x)` berechnet.

```
_activation(x) {
  return 1 / (1 + Math.exp(-x));
}
```

4.1.1.3 Programmierung des Feed Forward Algorithmus

Zuerst wird überprüft, ob die Anzahl der angegebenen Inputs mit der Grösse des Input-Layers übereinstimmt. Ist dies nicht der Fall, wir die Methode verlassen. Zu den Inputs wird dann das Bias für das Hidden-Layer hinzugefügt, da es in den Berechnungen genau gleich verarbeitet wird.

```
if(input.length !== this.input)
  return false;
input.push(this._bHidden);
```

Die Inputs werden nun auch in eine Matrix umgewandelt, um eine Matrix-Multiplikation durchführen zu können. Anschliessend wird diese Matrix mit denen der Matrix der Gewichte zwischen Input- und Hidden-Layer multipliziert. Da die Gewichtsmatrix noch nicht transponiert ist und somit die Anzahl der Zeilen der Inputmatrix und die Anzahl der Spalten der Gewichtsmatrix nicht übereinstimmen, werden mit `math.transpose()` die Zeilen und Spalten der Gewichtsmatrix getauscht. Anschliessend wird auf das erhaltene Matrixprodukt mit der `math.map()`-Funktion die Aktivierungsfunktion angewandt.

```
let inputMatrix = math.matrix(input);
let hiddenMatrix = math.multiply(inputMatrix,math.transpose(this._wInputHidden));
hiddenMatrix = math.map(hiddenMatrix, this._activation);
```

Das Hinzufügen des Bias – nun für das Output-Layer – und die Multiplikation der Matrizen, wie auch das Anwenden der Aktivierungsfunktion werden erneut für das Output-Layer angewandt. Somit erhält man eine Output-Matrix, die alle Outputs enthält.

```
hiddenMatrix = math.concat(hiddenMatrix, [this._bOutput]);
outputMatrix = math.multiply(hiddenMatrix, math.transpose(this._wHiddenOutput));
outputMatrix = math.map(outputMatrix, this._activation);
```

Da auch die Backpropagation-Methode auf die Feed-Forward-Methode zurückgreift, aber noch zusätzliche Daten von Zwischenschritten des Feed-Forward-Algorithmus benötigt, können alle generierten Layer-Matrizen zurückgegeben werden, wenn der zweite Parameter dem Wert `true` entspricht. Auch wird im `return`-Statement die Output-Matrix in ein Array umgewandelt.

```
return debug ? [inputMatrix,hiddenMatrix,outputMatrix] : outputMatrix.toArray();
```

4.1.1.4 Programmierung des Backpropagation Algorithmus

Als erstes wird überprüft, ob die angegebenen Werte mit der Grösse des Neuronalen Netzwerkes übereinstimmen. Darauffolgend wird der Feed-Forward-Algorithmus einmal mit dem gewünschten Input durchgeführt, um die aktuellen Outputs zu erhalten. Hier werden nun einzelne Werte der Zwischenschritte des FF-Algorithmus angefordert, um diese Berechnungen einfacher rückpropagieren zu können.

```
if(desired_input.length !== this.input || desired_output.length !== this.output)
  return false;
let [inputMatrix, hiddenMatrix, outputMatrix] = this.ff(input, true);
```

Nun werden die Errors des Output-Layers berechnet. Sie setzen sich aus der Differenz der beiden Outputs – dem gewünschten und dem erhaltenen – zusammen.

```
let e_output = math.subtract(desired,
    outputMatrix);
```

Um den Delta-Wert und die Veränderung der Gewichte berechnen zu können, muss zuerst der Rest des Gradienten bestimmt werden. Dazu gehört die Steigung des Netzinputs. Um vom Output zur Steigung zu kommen, wird die Ableitung der Aktivierungsfunktion – in diesem Falle die Ableitung von $sigmoid$ – auf den Output angewandt. Die Ableitungsfunktion wird mit `_dActivation(x)` berechnet. Nun wird das Resultat aus der Ableitung mit dem Error multipliziert. Da es sich in beiden Fällen um eine einzeilige Matrix, also um einen Vektor handelt, muss das *Hadamard-Produkt* – auch genannt Punktmultiplikation – bestimmt werden.

```
_dActivation(x) {
  return x * (1 - x);
}
```

```
let g_output = math.map(outputMatrix, this._dActivation);
g_output = math.dotMultiply(g_output, e_output);
```

Zur Bestimmung des Gradienten für die Hidden-Layer wird auf die Outputs des Layers die Ableitung der Aktivierungsfunktion angewandt. Der Error für die Hidden-Layer lässt sich durch die Matrix-Multiplikation des Output-Gradienten und der Gewichte zwischen Hidden- und Output-Layer bestimmen. Da in der Matrix der Gewichte auch die Gewichte des Bias abgelegt sind, müssen diese entfernt werden, da sie für davorliegende Layer irrelevant sind. Dies geschieht mit `math.subset()`. Nun werden diese beiden Werte noch miteinander multipliziert.

```
let g_hidden = math.map(hiddenMatrix, this._dActivation);
let e_hidden = math.subset(
    this._wHiddenOutput,math.index(
        math.range(0, this._wHiddenOutput.size()[0]),
        math.range(0, this._wHiddenOutput.size()[1]-1)
    )
);
g_hidden = math.dotMultiply(g_hidden, e_hidden);
```

Da nun alle Gradienten bestimmt sind, können die Delta-Werte bestimmt werden. Die Delta-Werte sind eine Matrix-Multiplikation eines Gradienten und der ein Layer davorliegenden Layer-Output-Matrix. Zusätzlich wird als Faktor die Lernrate angewandt.

```
// Multiplikation mit Layer-Output Matrix
let d_output = math.multiply(g_output, hiddenMatrix);
let d_hidden = math.multiply(g_hidden, inputMatrix);

// Faktor Lernrate anwenden
d_output = math.dotMultiply(d_output, this.lr);
d_hidden = math.dotMultiply(d_hidden, this.lr);
```

Der Delta-Wert kann jetzt auf die Gewichte angewandt werden und das Neuronale Netzwerk verbessern.

```
this._wHiddenOutput = math.add(this._wHiddenOutput, d_output);
this._wInputHidden = math.add(this._wInputHidden, d_hidden);
```

4.1.1.5 Umplanung auf externe Bibliothek

In den zuvor aufgezeigten Codebeispielen wird vor allem mit der *Mathjs*-Bibliothek gearbeitet (grossgeschriebenes «`Math`» ist die in JavaScript integrierte Mathematikbibliothek, kleingeschriebenes «`math`» ist die externe Bibliothek namens *Mathjs*). Die erste Version der Bibliothek ist hingegen mit einer eigenen Implementierung von Matrixfunktionen programmiert worden. Da aber die Effizienz der Algorithmen stark unter der Vielzahl von `for`-Loops litt, wurde in der späteren Entwicklung der Bibliothek auf die externe Mathematikbibliothek zugegriffen. Sie bietet eine hoch effiziente Art, Matrizen zu erstellen, zu multiplizieren und zu addieren. Die alte Implementierung ist immer noch in der Datei *SimpleJS-lib/lib/simple-old.js* (siehe: *9.1 Verzeichnisbaum*) ersichtlich. Wegen starker Einschränkung der Funktionalität aber nicht nutzbar.

4.1.2 Implementierung des genetischen Algorithmus

Zur Implementierung des genetischen Algorithmus wird ebenfalls auf die objektorientierte Programmierung gesetzt. Das Ziel ist es, dem Benutzer eine einzelne Klasse namens `GeneticAlgorithm` zur Verfügung zu stellen, die alle zu definierenden Variablen als Parameter aufnimmt und die selbstständig die Population und deren Individuen erzeugt. Somit befindet sich in einem `GeneticAlgorithm`-Objekt ein Array an Individuen der Klasse `Individual` (siehe *Abb. 15*). Dieses Array kann als Population bezeichnet werden. Ein einzelnes Individuum enthält schliesslich ein Objekt der gewünschten Klasse, welches das Individuum auszeichnet. Die Klasse dieses Objektes basiert auf dem Anwendungsbereich und wird vom Nutzer selbst erstellt. Wichtig ist dabei anzumerken, dass der Nutzer der Bibliothek gezwungen ist, zumindest teilweise objektorientiert zu arbeiten.

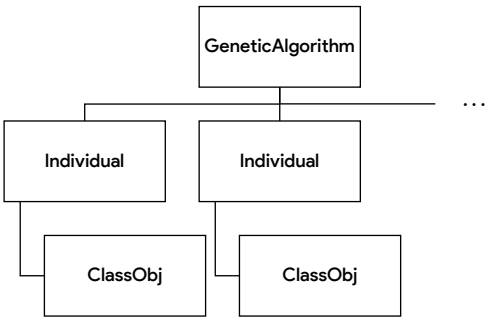


Abb. 15 Struktur GA-Bibliothek

4.1.2.1 Struktur der Klasse GeneticAlgorithm

Der genetische Algorithmus ist in der Klasse `GeneticAlgorithm` definiert. Als Parameter nimmt die Klasse die Grösse der Population mit `populationSize`, die Grösse der Neuronalen Netzwerke mit `nnInput`, `nnHidden`, `nnOutput` und die Lernrate für das Neuronale Netzwerk mit `nnLr` entgegen. Um definieren zu können, um was für Objekte es sich bei den Individuen der Population handelt, muss auch der Objektklassenname als Funktion der gewünschten Klasse für die Individuen angegeben werden. Da diese Klasse auch über eigene Argumente verfügen kann, können diese Argumente anschliessend hinzugefügt werden. Diese werden dann bei der Generierung der Individuen weitergegeben. So lässt sich alles in einem einzigen Schritt generieren. Die Argumente werden nicht als Array, sondern simpel durch Kommas getrennt nach den anderen Parametern hinzugefügt. JavaScript ES6 ermöglicht es, diese dann in der Klasse in einer einzigen Variable zu speichern. Dies geschieht mit den vor der Variable eingefügten drei Punkten (‘`...`’).

Klasse
GeneticAlgorithm

Parameter
```
- populationSize : number
- nnInput : number
- nnHidden : number
- nnOutput : number
- nnLr : number = 0.1
- objClass : function
- ...args : ...any
```

Methoden
```
+ setThreshold() : void
+ setLr() : void
+ getFittest() : Individual
+ forEach(func : function) : void
+ evolve(newSize : number =
      this.populationSize) : void
+ _selection(population : Individual[])
      : Individual[]
+ _crossover(population : Individual[],
      newSize : number) : Individual[]
+ _mutate(population : Individual[])
      : Individual[]
```

Die Methode `getFittest()` ermöglicht es, das erfolgreichste Individuum zu erhalten. Mit der Methode `forEach()` kann eine Funktion auf alle Individuen angewandt werden. Mit `evolve()` erzeugt man die nächste Generation. Auch lässt sich auf Wunsch die Grösse der Population der zukünftigen Generationen verändern. Wie auch in der Klasse `NeuralNetwork` sind mit einem Unterstrich (‘`_`’) markierte Methoden als private Methoden gedacht. `_selektion()`, `_crossover()` und `_mutate()` sind Teilschritte der `evolve()`-Methode. `setLr()` und `setThreshold()` ändern die Standardwerte Lernrate und Threshold (Schwellenwert) des genetischen Algorithmus ab.

4.1.2.2 Struktur der Klasse Individual

Ein Grossteil der erhaltenen Parameter in der Klasse `GeneticAlgorithm` wird an die einzelnen Individuen der Population als Argumente weitergegeben. Dies betrifft `nnInput`, `nnHidden`, `nnOutput`, `nnLr`, `objClass`, sowie die zur Objektklasse dazugehörenden Argumente (‘`...args`’). Zur Klasse `Individual` kommt die Methode `setFitness()` hinzu. Sie ermöglicht es, den Fitnesswert eines Individuums zu setzen, welcher danach entscheidet, welche Individuen zur Generierung der nächsten Generation ausgewählt werden.

Klasse
Individual

Parameter
```
- nnInput : number
- nnHidden : number
- nnOutput : number
- nnLr : number = 0.1
- objClass : function
- ...args : ...any
```

Methoden
```
+ setFitness(fitness : number) : void
```

4.1.2.3 Entwurf eines genetischen Algorithmus

Der genetische Algorithmus erzeugt eine Population, die durch den Verlauf des Trainings mehrere Generationen durchlaufen wird. Zuerst wird eine zufällige Population erzeugt. Jede Population besteht aus einer bestimmten Anzahl an Individuen. Diese Individuen wiederum enthalten Gene (siehe *Abb. 16*). Das Ziel ist es, die perfekten Gene für das perfekte Individuum zu finden. Der genetische Algorithmus wird dazu eingesetzt, ein Neuronales Netzwerk zu trainieren und zu optimieren. Zum Training des Neuronalen Netzwerkes werden die dazugehörenden Gewichte verändert. Sie werden im genetischen Algorithmus als Gene eingesetzt.

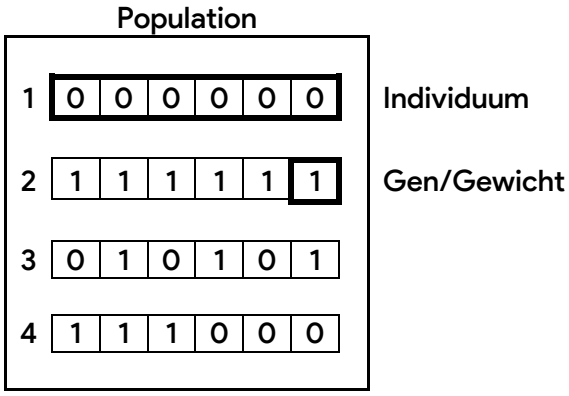


Abb. 16 Aufbau genetische Population

Die Generierung einer neuen Generation ist in drei Phasen unterteilt: Selektion, Crossover und Mutation. Alle Berechnungen in diesen Phasen sind abhängig vom Fitnesswert jedes einzelnen Individuums. Der Fitnesswert gibt an, wie erfolgreich ein Individuum war. Je höher der Wert, desto besser hat das Individuum abgeschnitten. Das Spektrum des Fitnesswertes ist irrelevant. Jeder Fitnesswert hat seine Bedeutung im Vergleich zu den Fitnesswerten der anderen Individuen.

Selektion

In der ersten Phase zur Erzeugung der nächsten Generation wird entschieden, welche Individuen geeignete Gene besitzen, die der nächsten Generation einen Mehrwert bieten. Hierbei wird das erfolgreichste Individuum gesucht und alle Fitnesswerte der anderen Individuen mit dessen Fitnesswert verglichen. Ist die Abweichung zu gross, wird das Individuum ausgeschieden (siehe *Abb. 17*). Wie gross diese Abweichung sein darf, wird durch den vordefinierten Schwellenwert angegeben.

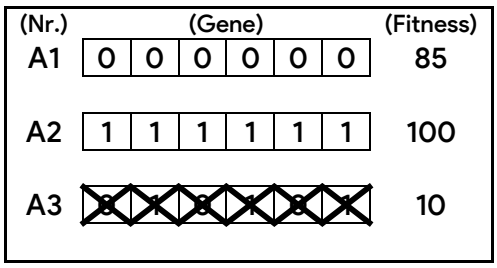


Abb. 17 Selektion im GA

Crossover

Das Crossover ist die Phase mit der grössten Auswirkung auf die Entwicklung. Sie wählt wiederholt zwei der durch die Selektion ausgewählten Individuen aus und erzeugt durch die Kombination der beiden ein neues Individuum. Dies geschieht dadurch, dass für jedes einzelne Gen zufällig entweder das Gen des ersten oder das des zweiten ausgewählten Individuums (Eltern) auf das neue Individuum (Kind) übernommen wird (siehe *Abb. 18*). So können sich auf die Fitness gut auswirkende Eigenschaften kombiniert werden, um ein noch stärkeres Individuum zu schaffen.

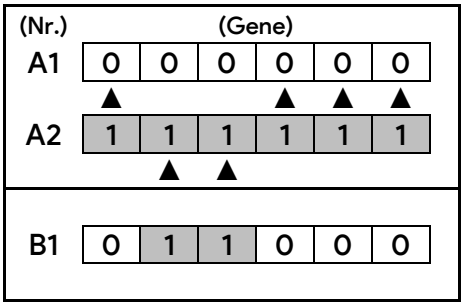


Abb. 18 Crossover im GA

Mutation

Bis jetzt existieren nur Eigenschaften, die zufällig bei der Erzeugung der ersten Generation generiert wurden. Um aber neue, in der ganzen Population inexistente Eigenschaften hinzuzufügen, findet die Mutationsphase statt. In der Mutationsphase wird ein kleiner Teil der Gene der neu erzeugten Individuen zufällig verändert (siehe *Abb. 19*). Wie gross dieser kleine Teil ist, wird durch die vordefinierte Lernrate des genetischen Algorithmus vorgegeben.

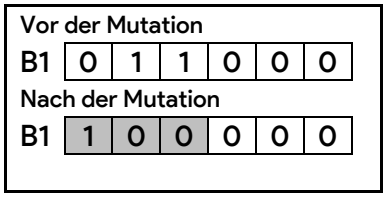


Abb. 19 Mutation im GA

4.1.2.4 Programmierung des genetischen Algorithmus

Der Code des genetischen Algorithmus befindet sich in der gleichen Datei wie der des Neuronalen Netzwerkes und der Backpropagation. Durch den genetischen Algorithmus werden der Bibliothek zwei neue Klassen hinzugefügt. Einerseits `GeneticAlgorithm`, die die Berechnungen auf Ebene einer ganzen Population beinhaltet und `Individual`, worin sich die Berechnungen befinden, die für jedes Individuum einzeln angewandt werden.

Programmierung der Klasse GeneticAlgorithm

Im Konstruktor der `GeneticAlgorithm` Klasse werden die Argumente entgegengenommen sowie auch gerade schon alle Individuen der ersten Population erzeugt.

```
constructor(populationSize, nnInput, nnHidden, nnOutput, nnLr = 0.1, objClass,
            ...args) {
  this.populationSize = populationSize;
  this.population = [];

  // Individuen erzeugen und der Population hinzufügen
  for (let i = 0; i < this.populationSize; i++) {
    this.population.push(new Individual(nnInput, nnHidden, nnOutput, nnLr,
                         objClass, ...args));
  }
}
```

Die Methode `getFittest()` iteriert durch die Population durch und ersetzt das erfolgreichste Individuum, wenn ein Individuum gefunden wird, das einen höheren Fitnesswert besitzt.

```
getFittest() {
  let fittest = this.population[0];
  for (let i = 1; i < this.populationSize; i++) {
    if (this.population[i].fitness > fittest.fitness)
      fittest = this.population[i];
  }
  return fittest;
}
```

Die `forEach()`-Methode ersetzt die in JavaScript schon enthaltene Methode dadurch, dass die gegebene Funktion nicht auf das `Individual`-Objekt angewandt wird, sondern auf das eigentliche Objekt innerhalb des Individuums.

```
forEach(func) {
  this.population.forEach(individual => func(individual.obj));
}
```

Danach folgt nur noch die für den genetischen Algorithmus wichtigste Methode. `evolve()` erzeugt eine neue Generation und baut sich aus einem Ablauf von verschiedenen Unterprozessen auf. Die neue Population wird anhand der Selektion, dem Crossover und der Mutation bestimmt. Die Variable `this.population` wird erst am Ende der Berechnungen ersetzt, da Informationen aus der alten Population nötig sind, um die neue zu generieren.

```
evolve(newSize = this.populationSize) {
  let newPopulation = this._selection(this.population);
  newPopulation = this._crossover(newPopulation, newSize);
  newPopulation = this._mutate(newPopulation);
  this.population = newPopulation;
  this.populationSize = this.population.length;
}
```

Nun geht es um die Unterprozesse selbst. Der erste ist die Selektion. Bei der Selektion werden die erfolgreichsten Individuen ausgewählt. Anhand von `this.getFittest()` lässt sich das erfolgreichste Individuum bestimmen. Es wird direkt in die neue Population aufgenommen. Da es das erste Individuum im Populationsarray ist, besitzt es den Index 0. Anhand des Fitnesswertes des erfolgreichsten Individuums wird entschieden, welche weiteren Individuen aufgenommen werden. Ist der Fitnesswert eines Individuums grösser als der durch den Fitnesswert des erfolgreichsten Individuums definierten Schwellenwert, so wird das Individuum der neuen Generation hinzugefügt.

```
_selection(population) {
  let newPopulation = [];
  let fittest = this.getFittest();
  newPopulation.push(fittest);
  let threshold = fittest.fitness * this.threshold;
    for (let i = 0; i < population.length; i++) {
      if (population[i].fitness >= threshold) {
        newPopulation.push(population[i]);
      }
    }
  return newPopulation;
}
```

Auf die Selektion folgt das Crossover. Beim Crossover werden die durch die Selektion der neuen Population hinzugefügten Individuen als Eltern eingesetzt. Die zuvor genannte neue Population wird hier als neue alte Population hinzugefügt. Die neue Population in dieser Methode ist bei Beginn noch leer. Das Erzeugen eines Kindes durch zwei Elternteile wird so lange wiederholt, bis die gewünschte Populationsgrösse erreicht ist.

```
_crossover(population, newSize) {
  let newPopulation = [];
  while(newPopulation.length < newSize){
    ...
```

Pro Durchlauf werden die Eltern definiert. Dies geschieht zufallsbasiert. Ebenso wird ein neues Individuum erzeugt, das das Kind repräsentiert.

```
    let p1 = population[math.floor(Math.random() * population.length)];
    let p2 = population[math.floor(Math.random() * population.length)];
    let child = new Individual(p1.nn.input, p1.nn.hidden, p1.nn.output,
                               p1.nn.lr, p1.objClass, ...p1.args);
```

Dieses Kind besitzt noch nicht die Gene der Eltern. Mit zwei verschachtelten Iterationen werden die Gewichte des Neuronalen Netzwerkes des Kindes an die der Eltern angepasst. Bei jedem Gewicht wird mit einer $50/50$ Wahrscheinlichkeit entschieden, von welchem Elternteil der Wert übernommen wird. Die erste Iteration ist für die Gewichte zwischen Input- und Hidden-Layer verantwortlich, die zweite für die Gewichte zwischen Hidden- und Output-Layer.

```
    for(let i = 0; i < child.nn._wInputHidden.size()[0]; i++){
      for(let j = 0; j < child.nn._wInputHidden.size()[1]; j++){
        child.nn._wInputHidden._data[i][j] = Math.random() > .5 ?
      p2.nn._wInputHidden._data[i][j] : p1.nn._wInputHidden._data[i][j];
      }
    }
    for(let i = 0; i < child.nn._wHiddenOutput.size()[0]; i++){
      for(let j = 0; j < child.nn._wHiddenOutput.size()[1]; j++){
        child.nn._wHiddenOutput._data[i][j] = Math.random() > .5 ?
        p2.nn._wHiddenOutput._data[i][j] : p1.nn._wHiddenOutput._data[i][j];
      }
    }
```

Anschliessend werden die Objekte der Eltern gelöscht. Durch diese Massnahme kann unnötige Arbeitsspeicherauslastung vermieden werden. Diese Objekte könnten extern (von ausserhalb der Bibliothek) noch mit einer Variable verknüpft sein, was folglich die automatische Löschung verhindern würde. Danach wird das Kind der Population hinzugefügt. Ist die Population genügend gross, so endet der vorhin erwähnte Loop und die neue Population wird zurückgegeben.

```
    delete p1.obj
    delete p2.obj;
    newPopulation.push(child);
  }
  return newPopulation;
}
```

Zuletzt folgt die Mutation. In dieser Methode werden für alle Individuen, die sich in der durch das Crossover erzeugten Population befinden, die Gewichte leicht abgeändert. Die Wahrscheinlichkeit, ob

ein Gewicht verändert wird, wird durch die Lernrate des genetischen Algorithmus `this.gaLr` definiert. Bei einer Veränderung eines Gewichtes wird ein zufälliger neuer Wert dem Gewicht zugewiesen.

```
_mutate(population) {
  for (let i = 1; i < population.length-1; i++) {
    for (let j = 0; j < population[i].nn._wInputHidden.size()[0]; j++) {
      for (let k = 0; k < population[i].nn._wInputHidden.size()[1]; k++) {
        if (Math.random() < this.gaLr)
          population[i].nn._wInputHidden._data[j][k] = Math.random() * 2 - 1;
      }
    }
    for (let j = 0; j < population[i].nn._wHiddenOutput.size()[0]; j++) {
      for (let k = 0; k < population[i].nn._wHiddenOutput.size()[1]; k++) {
        if (Math.random() < this.gaLr)
          population[i].nn._wHiddenOutput._data[j][k] = Math.random() * 2 - 1;
      }
    }
  }
  return population;
}
```

Programmierung der Klasse Individual

Die Klasse `Individual` besteht aus sehr wenigen Zeilen an Code, verglichen mit der Klasse `GeneticAlgorithm`. Sie ist primär zur Speicherung der drei zusammengehörenden Variablen, dem Objekt selbst, dem Gehirn des Objektes, umgesetzt als Neuronales Netzwerk, und dem Fitnesswert zuständig. Im Konstruktor erzeugt sie ein Objekt basierend auf der gewünschten Klasse und ein Neuronales Netzwerk. Mit `setFitness()` wird der Fitnesswert beeinflusst.

```
class Individual {
  constructor(nnInput, nnHidden, nnOutput, nnLR, objClass, ...args) {
    this.objClass = objClass;
    this.args = args;
    this.nn = new NeuralNetwork(nnInput, nnHidden, nnOutput, nnLR);
    this.fitness = 0;
    this.obj = new objClass(this,...this.args);
  }
  setFitness(fitnessValue){
    this.fitness = fitnessValue;
  }
}
```

4.1.2.5 Implementierung im Endprodukt

Um den genetischen Algorithmus anwenden zu können, muss eine Klasse erzeugt werden, die die Individuen repräsentieren soll. Sie kann eigene Methoden und Instanzvariablen enthalten. Wichtig ist, dass es sich beim ersten Parameter des Konstruktors um ein Objekt des Typs `Individual` handelt. Das erste Argument wird immer vom genetischen Algorithmus selbst gesetzt. Der Nutzer definiert nur die darauffolgenden Argumente. Der erste Parameter enthält immer das Individuum, in dem sich das durch das Individuum selbst definierte Objekt befindet. Dies ist wichtig, damit von Methoden des Objektes selbst Methoden des Individuums ausgeführt werden können, wie beispielsweise die `Individual.setFitness()`. Dies lässt sich mit der durch den Konstruktor erhaltenen Variable problemlos durchführen. So lässt sich auch die Fitness selbst im Objekt berechnen und muss nicht in die Bibliothek integriert werden, da je nach Anwendungsfall dieser Wert unterschiedlich berechnet wird.

```
class Example {
  constructor(individual, a, b, c) {
    this.individual = individual;
    this.a,this.b,this.c = a,b,c
    this.failed = false;
    this.fitness = 0;
  }
```

```
  setFitness(){
    this.individual.setFitness(fitness);
  }
```

```
  draw(){
    if(!this.failed) this.setFitness();
    else this.fitness ++;
  }
```

Um nun die im Codebeispiel gezeigte `draw()`-Methode ausführen zu können, kann auf die `forEach()`-Methode des genetischen Algorithmus zurückgegriffen werden.

```
let ga = new GeneticAlgorithm(popSize, nnI, nnH, nnO, nnLR, Example, a, b, c);

function draw(){
  ga.forEach(draw);
  requestAnimationFrame(draw);//führt draw im nächsten geladenen Frame erneut aus
}
```

4.1.3 Komprimierung des Codes

Da es sich bei der Bibliothek um ein dateigrössenmässig kleines Projekt handeln soll, wurden mehrere Optimierungen zur Komprimierung des Codes vorgenommen. Einerseits wurden viele Variablen gekürzt. Aus den gekürzten Variablen ist immer noch der Sinn zu entnehmen. Es handelt sich also nicht um zufällig gewählte kurze Strings. Auch wurden dank der modernisierten JavaScript-Version ganze Abschnitte auf eine einzelne Zeile gekürzt. Ein Beispiel zeigt *Abb. 20* auf, worin ein einfaches If-Else-Statement auf eine einzelne Zeile zusammengefasst wurde. Auch wurden wenn möglich alle Operationen auf einer Variable

```
//original
let a;
if(x>y) {
  a = 1;
}
else {
  a = 0;
}

//komprimiert
let a = x > y ? 1 : 0;
```

Abb. 20 Beispiel Komprimierung des Codes

in einem einzelnen Schritt durchgeführt. Schliesslich wurde der Code in eine zusätzliche Datei namens *SimpleJS-lib/lib/simple.min.js* (siehe: *9.1 Verzeichnisbaum*) übertragen, welche eine JavaScript-Minifizierung unterlief. Dies bedeutet, dem Code wurden nicht notwendige, leere Zeilen sowie Kommentare entfernt. Alles wird ohne Zeilenumschläge auf einer einzelnen Zeile aufgelistet.

4.2 Beispielanwendungen

Nur anhand einer komplexen Dokumentation eine Bibliothek in ein eigenes Projekt einzubinden, kann vor allem bei noch unerfahrenen Programmierern zu einer grossen Herausforderung werden. Deshalb wurden zur Bibliothek und der dazugehörenden Dokumentation auch noch Beispielprojekte erstellt. Jedes dieser Beispiele benutzt die Bibliothek auf eine andere Art. Sie dienen dazu, dass der Benutzer der Bibliothek Orientierungsmöglichkeiten erhält und direkt an schon umgesetztem Code sehen kann, wie die Bibliothek in Projekte integriert werden kann. Die Beispiele selbst dienen keiner Anwendung. Hingegen lassen sie sich als Präsentationsbeispiele perfekt in die noch dazu gelieferte Website einbinden und können interaktiv die Möglichkeiten der Bibliothek aufzeigen.

Wie die Bibliothek basieren auch die Beispielanwendungen auf *ECMAScript 2015*. Zusätzlich zur Programmiersprache JavaScript wird auch noch HTML und CSS zur Visualisierung der Anwendungen eingesetzt.

4.2.1 Beispiel Backpropagation

Das Beispiel Backpropagation soll den Anwendungsfall des Trainings des Neuronalen Netzwerkes mit schon existierenden Daten visualisieren. Es wird nur auf den Feed-Forward-Algorithmus sowie auf die Backpropagation zugegriffen. Der genetische Algorithmus ist hier irrelevant. Das Beispiel ist unter *SimpleJS-BPExample* (siehe: *9.1 Verzeichnis*) einsehbar.

Diese Anwendung hat ihren Ursprung in der ersten erstellten Testanwendung. Diese Testanwendung wurde benutzt, um die Funktionalität der Bibliothek zu testen. Sie löst das *XOR*-Problem und ist unter *SimpleJS-XOR* (siehe: *9.1 Verzeichnis*) aufzufinden. Um zu verstehen, zu was die Bibliothek fähig ist, mangelt es bei ihr an herausfordernden Problemen. Deswegen wurde eine erweiterte Version dieser Anwendung entwickelt. Sie ermöglicht dem Benutzer eigene Problemstellungen zu definieren. So kann das ganze Potential der Backpropagation ausgeschöpft werden. Ausserdem ist es möglich, die Lernrate anzupassen und mit deren Werten zu experimentieren. Weiter werden dem Benutzer sogenannte Templates zur Verfügung gestellt. Diese enthalten schon vorgegebene Problemstellungen. Der Standardwert der Lernrate in diesem Beispiel wurde basierend auf den Templates gewählt und dadurch auf 1 gesetzt. Es handelt sich dabei um die ideale Lernrate, im Falle, man arbeite mit den Templates (siehe *6.1.1.1 Optimale Lernrate*). Das Neuronale Netzwerk besitzt zwei Input-Neuronen, ein Output-Neuron, sowie ein Hidden-Layer mit acht Neuronen.

4.2.1.1 Klassifikationsprobleme

Beim *XOR*-Problem, sowie bei den vom Benutzer erstellten Problemen handelt es sich um Klassifikationsprobleme. Das *XOR*-Problem wird durch vier Punkte definiert. Beim benutzerdefinierten Problem kann der Benutzer eine unbegrenzte Anzahl an Punkten selbst setzen. Das Ziel der Backpropagation ist es, das Neuronale Netzwerk so zu trainieren, dass sich zwei Datensätze voneinander abtrennen lassen. In *Abb. 21* dargestellt als

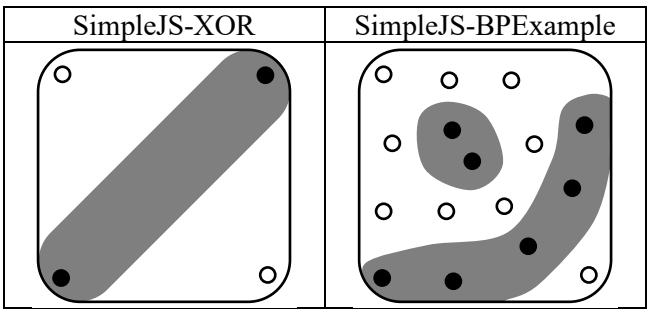


Abb. 21 Gelöste Klassifikationsprobleme

schwarze und weisse Punkte. Bei diesem Beispiel repräsentiert Weiss den Wert 0 und Schwarz den Wert 1. Dies ist der gewünschte Output an der spezifischen Position. Es sind als Inputs in das Neuronale Netzwerk die beiden Achsenwerte x und y anzugeben. Als Output erhält man idealerweise den für diese Position gewünschte Wert. Das Ziel ist es, die Gewichte so zu verändern, dass für alle Trainingsdaten, bestehend aus der Position des Punktes als Input und dem Wert des Punktes als Output, der erhaltene Output dem gewünschten Output entspricht.

Nun ist gut zu erkennen, was für zusätzliche Herausforderungen ein selbst definiertes Problem im Gegensatz zum *XOR*-Problem bieten kann (siehe *Abb. 21*). Während bei *XOR* nur in drei Bereiche geteilt werden muss, kann bei selbst erstellten Problemen die Anzahl der Bereiche um ein Vielfaches höher sein. Auch die Bereiche selbst können in ihrer Form komplexer sein.

4.2.2 Beispiel Flappy Bird

Bei *Flappy Bird* handelt es sich um ein Mobile Game des Entwicklers *Dong Nguyen* aus dem Jahr 2013, das schon wenige Monate nach dem Release den Titel als die meist heruntergeladene App besass. *Flappy Bird* ist, auch wenn es bereits 2014 aus den AppStores von Google und Apple verschwand, für viele immer noch ein Begriff [10]. *Flappy Bird* verfolgt ein ganz einfaches Spielprinzip. Man steuert einen Vogel in einer 2-dimensionalen Welt neben verschiedenen Hindernissen durch. Der Vogel bewegt sich von selbst nach vorne. Der Spieler hat nur die Kontrolle darüber, ob sich der Vogel nach oben oder unten bewegt. Wird eine Taste betätigt, führt der Vogel einen Flügelschlag durch, der ihn nach oben bewegt. Wird nichts betätigt, verliert er von selbst an Höhe. Als Hindernisse werden grosse Röhren bezeichnet, die eine Lücke beinhalten, durch welche sich der Vogel bewegen muss.

Flappy Bird eignet sich perfekt zur Visualisierung maschineller Lernalgorithmen, die Gebrauch von genetischen Algorithmen machen. Obwohl das Spiel für den Nutzer teils eine grosse Herausforderung darstellt, ist die Spielfigur, die im genetischen Algorithmus als Individuum eingesetzt wird, sehr simpel aufgebaut. Wenige Inputs können schon den einzig benötigten Output generieren und die Funktion zur Generierung dieses Outputs liesse sich sogar von Hand definieren.

4.2.2.1 Implementierung der Bibliothek

Als Inputs für das Individuum des genetischen Algorithmus, verkörpert durch die als Vogel visualisierte Spielfigur, werden der horizontale Abstand zwischen der Spielfigur und dem nächsten Hindernis, sowie die vertikalen Abstände zwischen Spielfigur und den Hindernislückenenden definiert (siehe *Abb. 22*). Als Output folgt ein Wert, der die Spielfigur nach oben bewegt, falls er einen Schwellenwert überschreitet.

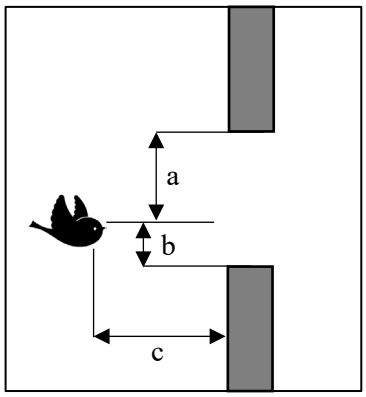


Abb. 22 Inputs für Flappy Bird

Als Population des genetischen Algorithmus wird eine grosse Menge an Vögeln bezeichnet. Da jeder Vogel ein eigenes Neuronales Netzwerk mit eigenen Gewichten enthält, ist folglich auch das Verhalten der einzelnen Vögel anders. Zuerst werden die Vögel als Population dem Spiel ausgesetzt. Sobald alle Vögel an den Hindernissen gescheitert sind, wird deren Fitness ausgewertet, anhand dieser Werte dann eine neue Population erzeugt werden kann.

Dieses Beispiel nutzt genetische Algorithmen und Neuronale Netzwerke ohne Einsatz der Backpropagation. Die Parameter für die genetischen Algorithmen wurden bei den Standardwerten der Bibliothek belassen. Die Lernrate liegt bei 0.1 und der Schwellenwert bei 0.8. Das Neuronale Netzwerk

weist eine Grösse von drei Input-Neuronen, zehn Neuronen im Hidden-Layer und einem Output-Neuron auf. Die Populationsgrösse wurde testmässig auf 100 gesetzt.

4.2.2.2 Perfektes Individuum

Das Beispiel *Flappy Bird* verfügt über eine suboptimale Situation zur Feststellung der Existenz eines perfekten Individuums. Da die Umwelt zufällig generiert ist, lässt sich in solch einer Umwelt nicht bestimmen, ob das Individuum auch in einer anderen Umwelt gleich gut performen würde. Da es sich aber hierbei um eine nicht sehr komplexe Umwelt handelt, lassen sich durch das Testen von Extremsituationen an einem Individuum Rückschlüsse über dessen Immortalität ziehen. Übersteht das Individuum alle Extremsituationen, so wird es theoretisch auch alle anderen Situationen meistern. Wichtig ist hierbei, dass sich Ereignisse in der Umwelt voneinander abtrennen lassen. Das bedeutet, dass ein vorheriges Ereignis keinen Einfluss auf das Individuum hat während des nächsten Ereignisses. Wird das Individuum durch das erste Ereignis geschädigt, so könnte es das nächste beispielsweise nicht mehr meistern, obwohl dieses leicht und ohne eine Schädigung überwindbar wäre. Glücklicherweise ist beim *Flappy Bird*-Beispiel nur die vertikale Position des Vogels ein zu verändernder Wert. Der einzige Extremfall für das Individuum ist folglich die maximal mögliche Höhendifferenz der Lücken zweier aufeinanderfolgender Hindernisse. Da sich der Vogel anders nach oben bewegt als nach unten, müssen beide Fälle berücksichtigt werden: Die erste Hindernislücke zuoberst und die folgende zuunterst, sowie genau das Umgekehrte. Übersteht das Individuum beide Herausforderungen fehlerfrei, so lässt sich annehmen, dass es in jeder, diesen Parametern entsprechenden Umwelt klarkommen wird.

4.2.3 Beispiel Car Game

Im dritten Beispiel werden Backpropagation und der genetische Algorithmus kombiniert. Anhand von durch den Benutzer erhaltenen Daten wird ein Neuronales Netzwerk mit der Backpropagation trainiert. Danach wird mit dem genetischen Algorithmus das Neuronale Netzwerk perfektioniert.

Das *Car Game* Beispiel ist eine Anwendung, die anhand der Bibliothek einem virtuellen Fahrzeug das Fahren beibringt. Durch ein Vorzeigen der richtigen Fahrweise durch den Benutzer anhand von Tasteneingaben lernt das virtuelle Fahrzeug dies selbst zu meistern. Das Fahren findet auf einer virtuellen Rundstrecke statt. Das Ziel ist es, eine Umrundung fahren zu können, ohne eine Wand zu berühren. Um dem Benutzer die Fähigkeiten der Bibliothek aufzeigen zu können, gibt es die Möglichkeit die Strecke durch das Setzen von neuen Wänden abzuändern und somit den Fahrzeugen eine neue Herausforderung zu schaffen.

Das Beispiel wird mit einer externen Bibliothek namens *ThreeJS* umgesetzt. Sie ermöglicht es, mit einem dreidimensional visualisierbaren *HTML-Canvas* zu arbeiten.

4.2.3.1 Implementierung der Bibliothek

Als Inputs für das Neuronale Netzwerk wird die Sicht des Fahrzeuges genutzt, verkörpert als Distanz vom Fahrzeug zu einem anderen Objekt in eine bestimmte Richtung (siehe *Abb. 23*). Als Outputs werden zwei Werte zurückgegeben. Einer entscheidet, ob das Fahrzeug beschleunigen soll oder nicht. Der andere steht dafür, wie fest nach rechts oder links gesteuert werden soll. Das Neuronale Netzwerk hat somit 9 Input-Neuronen, 50 Hidden-Neuronen und 2 Output-Neuronen. Die Lernrate liegt bei 0.1.

Abb. 23 Sichtfeld des Fahrzeuges

Um das Training anhand der kombinierten Methodik umsetzen zu können, müssen Daten gewonnen werden, die zum Training eingesetzt werden können. Der Benutzer steuert das Fahrzeug zuerst anhand von Tasteneingaben manuell durch die Strecke. Bei jedem Frame wird die Sicht des Fahrzeuges sowie der Lenkwinkel und eine allfällige Beschleunigung gespeichert. All diese

erhaltenen Daten trainieren anschliessend ein Neuronales Netzwerk mit der Backpropagation. Da der Benutzer voraussichtlich nicht den schnellsten und effizientesten Weg gefahren ist, wird nun der genetische Algorithmus eingesetzt. Eine Population wird erzeugt. Anstatt den Individuen zufällige Gewichte zuzuweisen, werden allen Individuen die Gewichte des durch die Backpropagation trainierten Neuronalen Netzwerkes übergeben. Da nun alle Individuen gleich sind, kann direkt ein Evolutionsschritt stattfinden, ohne Fitnesswerte ermitteln zu müssen. Durch die Mutation können nun noch bessere Eigenschaften dem Individuum hinzugefügt werden. Nun wird der genetische Algorithmus wie gewohnt ausgeführt. Die Effizienz der Fahrzeuge sollte sich nun verbessern. Beim genetischen Algorithmus werden die Parameter beim Bibliotheksstandard belassen. Die Populationsgrösse ist 50.

Der Fitnesswert der Individuen im genetischen Algorithmus wird anhand der gefahrenen Strecke ermittelt. Erreicht ein einzelnes Individuum das Ziel geht die Fitnesswertberechnung in eine zweite Phase über. Nun zählt die aufgewandte Zeit, um das Ziel zu erreichen. Der Fitnesswert ist höher, wenn die Fahrzeit kürzer ist. Auch sorgen Checkpoints während beider Phasen für weitere zusätzliche Punkte. Einerseits müssen alle Checkpoints durchfahren werden, damit das Erreichen des Ziels gewertet wird, andererseits sorgen sie dafür, dass sich im Kreis drehende Fahrzeuge nicht gut gewertet werden, verglichen mit Fahrzeugen die weiterkamen, aber eine kleinere Distanz zurücklegten.

Die genetische Lernrate ist beim *Car Game* Beispiel sehr niedrig angesetzt, da nur noch eine kleine Optimierung stattfinden soll. Die Fahrzeuge sollten idealerweise schon vor dem Durchlauf des genetischen Algorithmus wissen, wie das Fahren funktioniert.

4.2.3.2 Herausforderung Garbage-Collector

Das *Car Game* Beispiel bot mit Abstand die grössten Herausforderungen des ganzen Projektes. Das Beispiel litt während des Implementierungsprozesses lange an einer extremen Ineffizienz. Das dreidimensionale Darstellen von hunderten von Fahrzeugen ist nur auf rechenleistungsstarken Computern möglich und somit schlecht für eine Browseransicht geeignet. Auch verfügt *ThreeJS* über einen ungeeigneten *Garbage-Collector*, wodurch nach mehreren genetischen Generationen eine starke Belastung des Arbeitsspeichers stattfand. Grossteils konnten diese Probleme behoben werden. Das Benutzererlebnis ist aber immer noch stark abhängig von der Rechenleistung des zur Visualisierung eingesetzten Rechners.

4.3 Implementierung der Website

Zur öffentlichen Visualisierung der Beispielanwendungen und Präsentation der Bibliothek wurde eine Website programmiert. Sie bietet eine Übersicht aller Möglichkeiten der Bibliothek und eine Darstellung aller Beispielanwendungen mit für den Benutzer gut verständlichen Erklärungen. Der Benutzer sollte nach einem Websitebesuch problemlos entscheiden können, ob sich die Komponenten der Bibliothek in seine eigenen Anwendungen gut einbinden lassen. Zusätzlich ist die Website benutzerfreundlich und ansprechend gestaltet anhand eines schlichten, leicht verspielten Designs, das die einfache Einsetzbarkeit der Bibliothek repräsentiert.

4.3.1 Entwurf der Website

Da es sich bei einer Website um tausende von Zeilen an Code in verschiedenen Programmiersprachen handelt, ist vor der Implementierung der Website eine gute Planung notwendig. Aufbau und Inhalt müssen zuvor klar definiert werden. Darauf folgt ein Entwurf eines Designsystems, das die Website auf schöne Weise gestalten soll.

4.3.1.1 Grundstruktur

Die Einfachheit der Website beginnt schon beim Ladebildschirm. Da die Website eine grosse Menge an externen Bibliotheken, sowie eigene Beispielanwendungen laden muss, kann es sich bei der Ladezeit, abhängig von der Internetgeschwindigkeit des Benutzers, um mehrere Sekunden handeln. Deshalb wird dem Benutzer eine auf dem Logo basierende Animation präsentiert. Die Animation zusammen mit ihrem Hintergrund füllt den ganzen Bildschirm aus. Somit sind keine unschönen, noch nicht fertig geladenen Elemente ersichtlich.

Handelt es sich beim Websitebesuch um das erste Mal, wird eine zusätzliche, flüssig animierte den Bibliothekstitel präsentierende Animation aufgezeigt. Diese wiederholt sich bei einem wiederholten Besuch der Website nicht. Die Anzahl Besuche wird als lokales Cookie in den Browserdaten hinterlegt.

Nach der Ladeanimation wird man vom Bibliothekstitel «*SimpleJS*» und dem dazugehörigen Slogan «*Lightweight Brilliance, Infinite Possibilites*» begrüsst. Gleich wie der Name präsentiert auch der Slogan die Grundideen der Bibliothek: Einfache und breit gestreute Einsetzbarkeit. Im unteren Teil der Ansicht ist schon der weitere Inhalt zu erkennen. Der Benutzer erkennt, dass seine weitere Vorgehensweise daraus bestehen muss, nach unten zu scrollen.

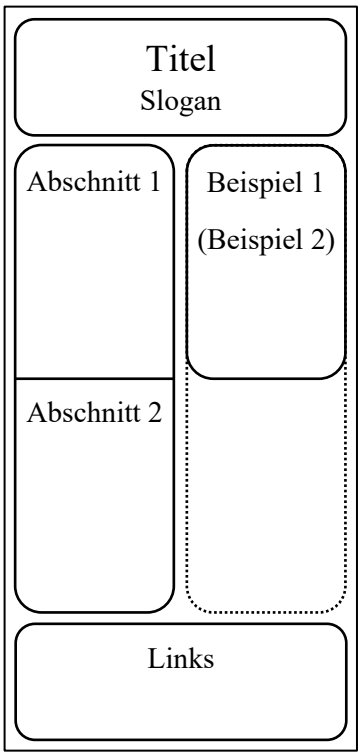


Abb. 24 Struktur der Website

Der Hauptteil der Website ist aufgeteilt in eine linke und in eine rechte Spalte. Die linke Spalte verhält sich dem Scrollen entsprechen. Die rechte Spalte hingegen ist fixiert, sobald sie die ganze Höhe des Fensters ausfüllt, und bewegt sich erst weiter nach oben, wenn der Hauptteil verlassen wird. Auch die rechte Spalte wechselt den Inhalt. Da sie dafür verantwortlich ist, ein passendes Beispiel zum Text in der linken Spalte aufzuzeigen, wird durch eine einfache Fade-Animation der alte Inhalt mit dem neuen ersetzt, sobald sich der linke Abschnitt verändert. Um die Abschnitte voneinander unterscheiden zu können, verfügt jeder Abschnitt über eine eigene Akzentfarbe. Somit entsprechen alle Elemente einem einheitlichen Farbdesign, unterscheiden sich aber von Elementen der anderen Abschnitte.

Am Ende der Website befindet sich der sogenannte Footer. Er präsentiert nochmals eine kurze Zusammenfassung des Projektes, gibt mit dem Projekt im Bezug stehende Websites als Link an und führt die Rechte an externer Benutzung und Lizenzen der Bibliothek und Website auf. Als Links sind zuerst die Kontaktmöglichkeiten mit dem Ersteller der Bibliothek und Website angegeben, gefolgt von projektspezifischen Links, wie die Verlinkung zum Open Source-Code, und den *GitHub*-Projektlinks zu den Beispielanwendungen.

Optimierung für Mobilgeräte

Um auch eine Ansicht für Geräte mit einer tiefen Bildschirmbreite zu ermöglichen, wird für Mobilgeräte die Struktur der Website leicht abgeändert. Im Hauptteil der Website wird nur noch die linke Spalte angezeigt. So kann diese den ganzen Platz einnehmen. Auch sind die Beispielanwendungen für Mobilgeräte teils nutzlos, da Smartphones meist über weniger Prozessorleistung verfügen und somit nicht die gleiche Bibliothekseffizienz stattfinden kann. Auch der Footer wird abgeändert. Die Linksgruppierungen werden nicht mehr nebeneinander angezeigt, sondern listen sich untereinander auf.

4.3.1.2 Designsystem

Die Website sowie die interaktiven Beispiele verfügen über ein einheitliches Designsystem. Das Designsystem repräsentiert die Grundideen der Bibliothek. Es wird auf ein benutzerfreundliches, schlicht gehaltenes, verspieltes Designschema gesetzt. Das Designsystem orientiert sich an dem von Google für das Betriebssystem Android angewandten *Material Design*. Seit der dritten Version des *Material Designs* hat sich Google weit von anderen, in professionell technischen Gebieten eingesetzten Designs entfernt. Durch grosse Eckabrundungen und dicke Schriftarten wirkt das Design fast schon verspielt. Dies passt perfekt zur Bibliothek dieses Projektes. Das verspielte Design steht für die einfache, benutzerfreundliche Anwendbarkeit. Dies wird auch durch die intuitiven Bedienelemente wie übergrosse Buttons und leicht zu verstehende Icons repräsentiert, da sie die einfache Bedienbarkeit der Website fördern. Auch werden im Design viele Gestaltungsformen als naturhergeleitete Formen, wie beispielsweise Blumen, dargestellt. Diese organischen Formen repräsentieren die Herkunft der Ideen des Neuronalen Netzwerkes sowie des genetischen Algorithmus, dessen Ursprünge sich auf biologische Prozesse zurückführen lassen.

Farben

Farblich teilt sich die Website in fünf Abschnitte auf. Dadurch existieren fünf Akzentfarben auf der Website. Zu jeder Akzentfarbe existiert eine Textfarbe, die auf dem entsprechenden Hintergrund eingesetzt wird. Dazu kommt noch eine Containerfarbe, die vor allem für Hover-Effekts eingesetzt wird. Die fünf Farbgruppen werden durch eine neutrale Farbgruppe ergänzt, die über zwei Hintergrundfarben sowie eine Textfarbe verfügt. Die neutralen Farben werden über die ganze Website verteilt eingesetzt.

Alle Farben sind für ein dunkles Design ausgelegt. Der Standardhintergrund der Website ist somit dunkel und darauf liegt ein heller Text (verwendetes Farbdesign siehe *9.2 Farbdesign Website*).

Formen

Das am *Material Design* inspirierte Designschema ist stark von den eigenen Formen abhängig. Vor allem die grossen Eckabrundungen erzeugen die verspielte Weise. Im Designschema werden zwei verschiedene Eckabrundungen verwendet. Einerseits die für grosse Elemente angewandten $24px$ und die für kleinere Buttons angewandten $12px$ (siehe *Abb. 25*). Das Ziel ist es, die Eckabrundungen so einheitlich wie möglich zu halten. Da aber die verhältnismässig grosse Eckabrundung von $24px$ nicht für kleine Buttons geeignet ist, wird bei zu kleinen Buttons auf die halbgrosse Eckabrundung gesetzt. Die kleinen Buttons treten nur in den Beispielanwendungen als Steuerungsbuttons auf.

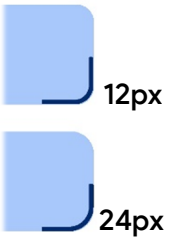


Abb. 25 Eckabrundungen

Auch wenn die meisten Formen der Website Rechtecke mit abgerundeten Ecken sind, gibt es auch Ausnahmen. Um die Website interessanter und abwechslungsreicher zu gestalten, sind im Designschema blumenartige Formen enthalten (siehe *Abb. 26*). Sie repräsentieren die Nähe zur Natur der Grundideen der Bibliothek. Die Formen werden primär zur Umrahmung von Icons eingesetzt. Neben all diesen Formen existiert noch eine an die Formstruktur der blumenartigen Formen angepasste, wellenartige Linie (siehe *Abb. 27*). Sie trennt verschiedene Bereiche der Webseite voneinander ab.

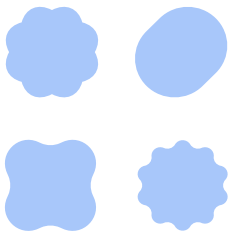


Abb. 26 Blumenformen

Abb. 27 Trennlinie

Abstände

Die Abstände zwischen zwei Elementen sind auf der ganzen Website einheitlich gehalten. Als Standardgrösse wird der Margin von 8px eingesetzt. Zwischen zusammengehörenden Elementen, sowie zwischen einem Element und dem Bildschirmrand wird diese Grösse eingesetzt. Handelt es sich um den Abstand zwischen zwei Abschnitten oder zwischen zwei verschiedenen Elementen wird die doppelte Menge an Pixel eingesetzt.

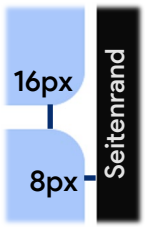


Abb. 28 Abstände

Icons

Zur Implementierung der Icons auf der Website wurde auf die einfache Lösung von *Google Fonts* gesetzt. Einerseits herrscht zwischen diesen Icons ein einheitliches Design und sie entsprechen dem am *Material Design* angelehnten Websitedesign. Andererseits lassen sich die Icons sehr einfach implementieren.

Abb. 29 Google Font Icons

Typographie

Die Website verfügt über vier verschiedene Textgrössen. Drei davon werden für Header eingesetzt. Als `h1` mit einer Grösse von $86px$ wird der Haupttitel der Website definiert. Er verfügt über eine extra dicke Schriftart mit einer Schriftdicke von 900 (basierend auf der CSS-Deklaration `font-weight`). Bei der Schriftart handelt es sich um *Product Sans*. Diese Schriftart wird bei allen Textstellen auf der Webseite eingesetzt. Abgesehen vom Haupttitel wird die Schriftdicke 700 für alle weiteren Übertitel eingesetzt. Die Übertitel der Websiteabschnitte werden als `h2` bezeichnet und weisen eine Schriftgrösse von $60px$ auf. Als Titel von Buttonelementen wird `h3` eingesetzt mit einer Schriftgrösse von $24px$. Der restliche Text nutzt die Schriftgrösse $16px$ mit einer Dicke von 400.

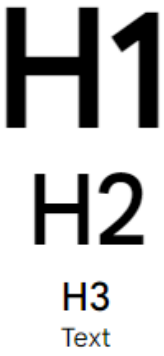


Abb. 30 Schriften

Elemente

Ein gutes Designsystem zeichnet nicht nur definierte Schriftarten, Formvorgaben oder Elementabstände aus. Ein Design soll aus sich wiederholenden Elementen bestehen. So gibt es beispielsweise in diesem Designsystem eine einheitliche Vorgabe, wie ein Button aussehen soll. Bei mehrfachem Anwenden des Buttons bleiben Schriftart, Schriftfarbe, Hintergrundfarbe so wie auch der Eckradius gleich. Auch die Veränderung des Buttons bei einem Anklicken oder einem Darüberfahren mit dem Cursor sind immer einheitlich. *Abb. 31* zeigt den auf der Website angewandten Standardbutton, der sich beispielsweise unter dem Websiteübertitel befindet. Seine Veränderung bei einer Hover-Aktion mit dem Cursor ist ein anderer Eckradius. Das Aufzeigen aller auf der Website angewandten Elemente macht in dieser Arbeit wenig Sinn. Die Website kann online unter https://simplejs.ai/ betrachtet werden.

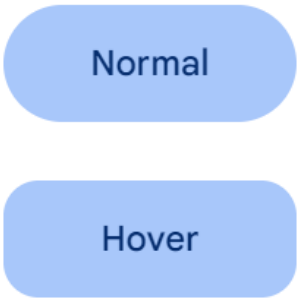


Abb. 31 Standardbutton

4.3.1.3 Name & Logo

Der Name der Bibliothek repräsentiert den Grundgedanken. Sie soll einfach einzusetzen sein. So bildet sich aus dem englischen Wort «Simple» zusammengesetzt mit der Abkürzung von der im Projekt eingesetzten Programmiersprache JavaScript der Name «*SimpleJS*».

Abb. 32 Logo

Um ein quadratförmiges Logo visualisieren zu können, wird als Logo der erste Buchstabe «S» der Bibliothek mit einem zusätzlichen Punkt verwendet. Der Punkt verkörpert das Pünktchen des darauffolgenden Buchstabens «i». Auch steht er, eingesetzt als Satzende-Punkt für eine kurze (visualisiert durch die Anzahl Buchstaben davor), simple Verwendung der Bibliothek. Der Aufwand zur Implementierung der Bibliothek in ein eigenes Projekt erfordert nur einen Schritt. Als Farbe für das Logo wird ein Hellblau eingesetzt. Der gleiche Farbton wie derjenige, der auch den Nutzer beim Websitebesuch als Akzentfarbe begrüsst. Das Hellblau stellt dar, dass es sich bei der Bibliothek um eine Implementierung von Algorithmen aus den Bereichen des maschinellen Lernens handelt. Hellblau ist eine Farbe, die in der Gesellschaft häufig mit Intelligenz und Technologie in Verbindung gebracht wird. Auch werden in vielen Science-Fiction-Filmen KI-Systeme mit blauer Farbe dargestellt [11]. Unabhängig von der Verknüpfung zum maschinellen Lernen, eignet sich Hellblau grossartig für Webdesigns, da es mit Vertrauen und Zuverlässigkeit assoziiert wird und Aufmerksamkeit erregt, ohne überwältigend zu sein.

4.3.1.4 Domain

Zwar wurde durch das von *GitHub* zur Verfügung gestellte Hosting eine Subdomain mitgeliefert, jedoch wurde für das Projekt zusätzlich eine eigene Domain erworben. Unter dem Namen www.simplejs.ai/ kann die implementierte Website eingesehen und mit ihr interagiert werden. Als Domainname (*Second-Level-Domain*) wurde der Name der Bibliothek ausgewählt. Optimal wäre www.simple.js/ gewesen. Hingegen existiert zum Zeitpunkt der Erstellung des Projektes die *Top-Level-Domain* «.js» noch nicht. Somit wurde auf die *TLD* namens «.ai» zurückgegriffen. «AI» steht im englischen als Abkürzung für «Artificial Intelligence», was auf Deutsch «Künstliche Intelligenz» bedeutet. Die *TLD* repräsentiert somit die Art des Produktes, das auf der Website präsentiert wird. Wichtig ist anzumerken, dass diese *TLD* nicht spezifisch für künstliche Intelligenzen angelegt wurde, sondern als *TLD* für das sich in der Karibik befindende, britische Überseegebiet namens Anguilla. Jedoch wird diese *TLD* sehr häufig im Bereich des maschinellen Lernens eingesetzt.

4.3.2 Programmierung der Website

Bei der Website handelt es sich um eine statische Website, die über kein Backend verfügt. Somit wird alles in von einem Browser lesbaren Programmier- und Markupsprachen umgesetzt. Zum Aufbau der Grundstruktur der Website wird HTML eingesetzt. Für alle Berechnungen wird auf JavaScript gesetzt, und die designtechnische Umsetzung findet durch CSS statt. CSS ist aber nicht die Sprache, die als Code geschrieben wurde. Der Code ist in SCSS geschrieben, das danach in CSS kompiliert wird. SCSS, entstanden aus dem bekannteren SASS («Syntactically Awesome Stylesheets»), erweitert CSS mit vielen Funktionen. Einer der grössten Vorteile ist, dass zur Selektion eines Elementes nicht mehr der ganze Pfad angegeben werden muss, sondern dass eine Deklaration direkt in den für das Elternelement angelegten Bereich hinzugefügt werden kann. So kann viel an Programmierarbeit gespart werden. Die Möglichkeit, Loops sowie Zufallsberechnungen nutzen zu können, kann dann den zur Website gehörenden JavaScript-Code kürzen.

Bei der Website handelt es sich um tausende Zeilen an Code. Hinter jedem Codeabschnitt liegen auch komplexe Überlegungen. Der Fokus dieser Arbeit liegt jedoch auf der Bibliothek und nicht auf der Website, wodurch es wenig Sinn macht, den kompletten Code zu erklären. Der Code ist aber trotzdem Teil des Projektes und kann er unter *SimpleJS* (siehe: *9.1 Verzeichnis*) angesehen werden.

4.3.3 Browsersupport

Da wegen der Komplexität der Website Optimierungen für alle gebräuchlichen Browser zu einer grossen zusätzlichen zeitlichen Herausforderung geworden wären, wurde der Fokus auf den weltweit am weitesten verbreiteten Browser *Chrome* gesetzt. Dazu gehören auch alle Browser, die auf Googles Open Source-Browserversion *Chromium* basieren. Somit sind der native Android-Browser und *Microsoft Edge* von Windows inkludiert. Anhand des *UserAgents* wird ermittelt, um welchen Browser es sich handelt und bei einem Nichtentsprechen ein Overlay angezeigt, das den Nutzer warnt, dass das Besuchserlebnis Einschränkungen mit sich bringen könnte.

4.4 Dokumentation

Die für die Bibliothek entwickelte Dokumentation besteht aus vier Abschnitten. Im Home-Abschnitt wird kurz aufgelistet, zu was die Bibliothek fähig ist. Ein Benutzer weiss nach dem Durchlesen dieses Abschnittes schon, ob die Bibliothek für ihn geeignet ist. Darauf folgt der Abschnitt Installation. Er zeigt eine einfache Anleitung auf, wie die Bibliothek einem Projekt hinzugefügt wird, so dass sie eingesetzt werden kann. Im dritten Abschnitt wird dem Benutzer erklärt, wie die Bibliothek benutzt werden soll. Es findet eine Auflistung aller Parameter und Methoden der Klassen statt. Alles verfügt über genaue Beschreibungen. Es wird auch klar angegeben, ob ein Parameter optional ist und ob ein Standardwert existiert. Der letzte Abschnitt zeigt anhand von Beispielen auf, wie die Bibliothek in einem Projekt angewandt werden kann.

5. Produktbezogene Auswertung

Nach dem Fertigstellen eines Projektes ist es wichtig, dessen Umsetzungserfolg wie auch dessen Funktionalität zu beurteilen. Hierfür gilt dieser Abschnitt. In den folgenden Analysen soll sichtbar werden, wie gut das Endprodukt wirklich der ursprünglichen Idee entspricht.

Die Bibliothek selbst ist vollkommen funktionsfähig und kann somit jegliche Problemstellungen bearbeiten und lösen. Auch weist sie eine unerwartet hohe Effizienz auf, welche später noch klarer aufgezeigt wird. Somit wird der erwartete Anwendungsbereich um ein weites übertroffen. Neben der Bibliothek entsprechen auch die fertiggestellten Beispielanwendungen der gewünschten Vorstellung. Sie verwenden die Bibliothek auf eine Weise, die dem Benutzer zeigt, was die Bibliothek leistet, und dies auf eine Weise, die durch das Betrachten des Codes leicht reproduzierbar ist. Auch weisen die Beispielanwendungen keine Fehler auf und sind komplett funktionstüchtig. Auch die Website entspricht der ursprünglichen Vorstellung. In ihr konnte das entworfene Designsystem perfekt umgesetzt werden. Sie macht einen für den Benutzer ansprechenden Eindruck, integriert die Beispielanwendungen auf elegante Weise und bringt dem Benutzer die Fähigkeiten der Bibliothek nahe.

5.1 Effizienz der Bibliothek

5.1.1 Effizienz der Backpropagation

Anhand der Beispielanwendung *BPExample* lässt sich leicht der Erfolg der Backpropagation bestimmen. Alle folgenden Versuche wurden anhand von zur Beispielanwendung gehörenden Templates durchgeführt. Nach jedem Durchlauf des Backpropagation Algorithmus, verglichen mit einem einzelnen Trainingsschritt, wurde der Gesamtverlust der Optimierung des Neuronalen Netzwerkes ermittelt. Der Verlust lässt sich mit dem in der Backpropagation angewandten Error vergleichen. Anhand des Outputs des teiltrainierten Neuronalen Netzwerk wird die Abweichung zu allen verfügbaren Daten ermittelt. Der Verlust ist schliesslich der Durchschnitt aller dieser Abweichungen. Ist das Neuronale Netzwerk perfekt trainiert, so entspricht der Verlust dem Wert 0.

Zur Auswertung jedes Templates wurden mehrere Durchläufe durchgeführt, um einzelne Abweichungen feststellen zu können. Pro Grafik sind je zwei dieser Durchläufe sichtbar. Die Effizienz der Backpropagation ist aber bei jedem Durchlauf fast gleich hoch. Anhand von je zehn Durchläufen auf die vier Templates konnten keine grossen Abweichungen festgestellt werden. Veranschaulicht am Template 1 lässt sich gut erkennen, dass beide visualisierten Durchläufe die gleiche Form aufweisen.

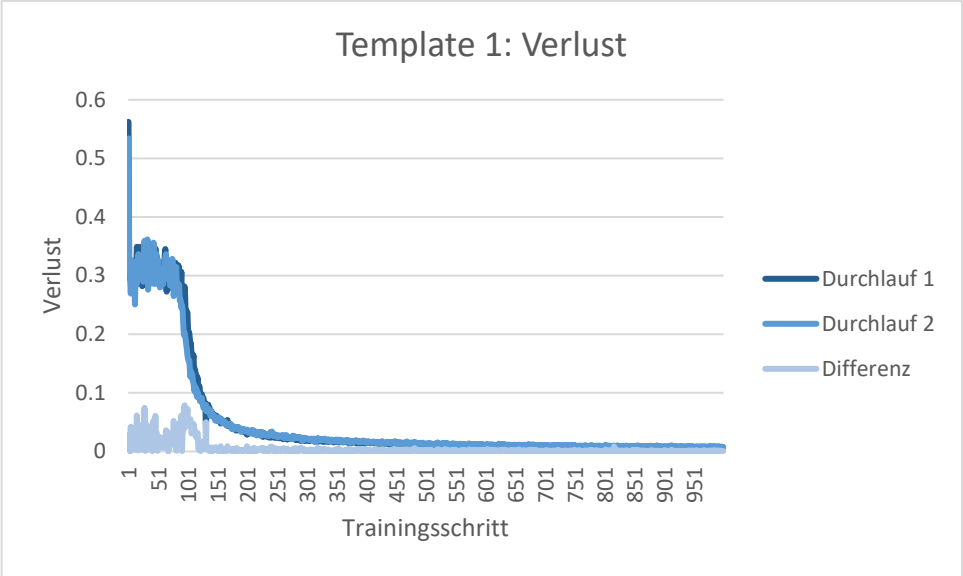


Abb. 33 Verlust Template 1 (siehe Tabelle 2)

Da die Differenzen der Durchläufe so gering sind, kann man davon ausgehen, dass alle Werte, die sich am Anfang der Durchläufe unterscheiden, eine Irrelevanz aufweisen. Denn alle darauffolgenden Prozesse der Backpropagation basieren nicht mehr auf Zufall. Eine grosse Rolle spielen die zufällig gewählten Gewichte bei der Generierung eines neuen Neuronalen Netzwerkes also nicht. Wie aber die Auswertung des Templates 2 aufzeigt, können diese Werte immer noch den ganzen Trainingsprozess minimal beeinflussen.

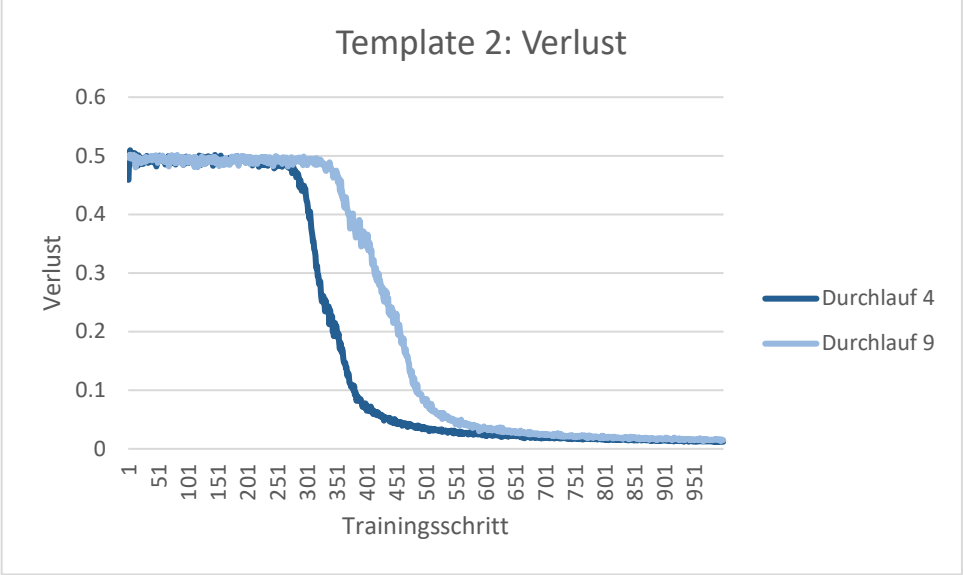


Abb. 34 Verlust Template 2 (siehe Tabelle 3)

Beim Durchlauf 4 von Template 2 konnte ein entscheidender, Grosses bewirkender Schritt nach weniger Trainingsschritten gefunden werden als bei Durchlauf 9. Hingegen ist auch hier wieder die Ähnlichkeit der beiden Graphen zu erkennen. Somit wird der gleiche Weg eingeschlagen, um die optimalen Gewichte zu finden. Bei allen weiteren Templates sind solche Ähnlichkeiten zu erkennen (siehe *9.3 Auswertungsdaten*, *Abb. 46* und *Abb. 47*). Die Durchläufe verschiedener Templates unterscheiden sich, da es sich um unterschiedliche Problemstellungen handelt.

5.1.1.1 Optimale Lernrate

Nicht nur ist die Backpropagation ein auf zufällig generierten Werten basierender Algorithmus, sondern seine Effizienz wird auch durch zuvor definierte Variablen bestimmt, darunter die Lernrate.

Eine Auswertung der Effizienz anhand verschiedener Lernraten zeigt auf, dass ein optimaler Wert existiert und dessen Limite sich nicht einem Extremwert (wie Null oder Unendlich) annähert.

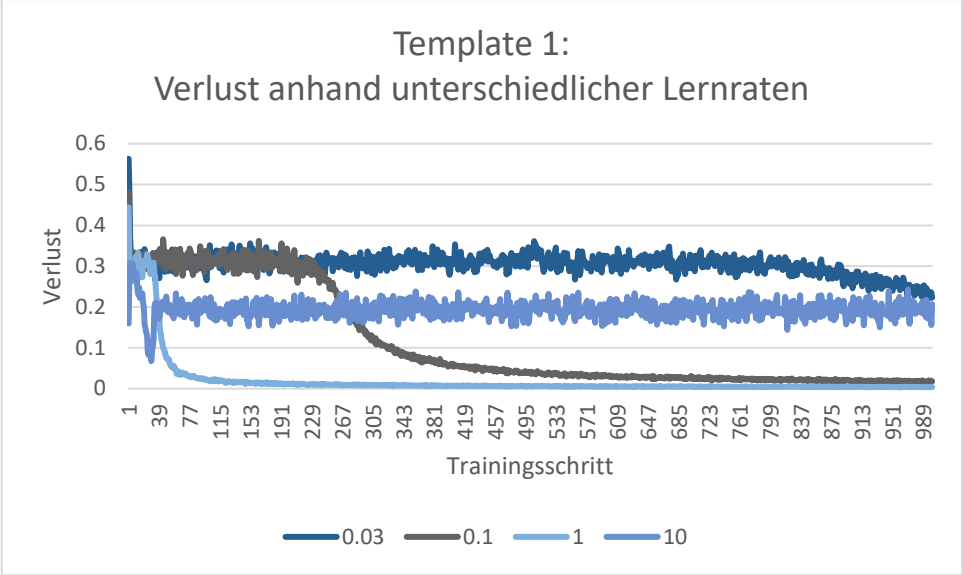


Abb. 35 Verlust anhand unterschiedlicher Lernraten (siehe Tabelle 6)

Die Werte 0.03, 0.1, 1 und 10 zeigen klar auf, welchen Einfluss die Lernrate haben kann. Wird auf Template 1 die Lernrate 0.03 angewandt, so verändert sich der Verlust fast gar nicht und erst nach einer grossen Menge an Trainingsschritten ist eine Optimierung zu erkennen. Das Trainingsverfahren benötigt also unnötig viele Rechnerressourcen, um eine Problemstellung zu lösen. Wird der Wert 0.1 als Lernrate verwendet, so kann innerhalb der tausend aufgezeigten Trainingsschritte ein Verlust von weniger als 0.02 erreicht werden. Auch die Lernrate mit dem Wert von 1 kann nach dieser Anzahl an Trainingsschritten einen solch tiefen Verlust erreichen. Hingegen kann dank den grösseren Schritten in Richtung Optimum der Verlustwert von 0.02 schon nach fast einem Fünftel der Schritte erreicht werden, verglichen mit der vorherigen Lernrate. Nun wirkt es so, als sei eine grössere Lernrate immer schneller und somit auch effizienter. Schaut man aber auf die Lernrate mit dem Wert 10, so erkennt man, dass es gar nicht mehr möglich ist, sich bei zu hohen Lernratenwerten gut an einen tiefen Verlustwert annähern zu können. Innerhalb der ersten Trainingsschritte wird das Optimum schon fast bestimmt, der Algorithmus schiesst aber über das Ziel hinaus und entfernt sich wieder von einem tiefen Verlust. Am Beispiel des Templates 1 liegt die optimale Lernrate etwa beim Wert 0.1. Dies lässt sich aber nicht auf alle Problemstellungen übertragen. Jeder Anwendungsfall verfügt über eine eigene optimale Lernrate. Allgemein lässt sich aber sagen, dass je schwieriger und komplexer der Anwendungsfall ist, desto kleiner die Lernrate sein muss.

5.1.2 Effizienz des genetischen Algorithmus

Ebenso wie die Backpropagation ist auch der genetische Algorithmus ein rechenleistungsintensiver Prozess. Somit ist es wichtig, sich auch hier mit der Effizienz und den Möglichkeiten diese zu optimieren zu beschäftigen.

Das *Flappy Bird* Beispiel eignet sich perfekt zur Feststellung der Effizienz des genetischen Algorithmus. Es lässt sich leicht bestimmen, ab wann ein perfektes Individuum geschaffen wurde. Auch kann dank

der nicht zu komplexen Problemstellung ohne hohen zeitlichen Aufwand eine grosse Menge an Auswertungsdaten erworben werden. Bei allen folgenden Versuchen wird der Fitnesswert 100′000 als perfekt gewertet. Bei jedem Durchlauf wurden aber sicherheitshalber die Individuen bis zum Wert 500′000 aktiv weiter berechnet. Bei keinem der Durchläufe fand ein Fehlschlag zwischen diesen beiden Werten statt. Somit kann man davon ausgehen, dass die Wahrscheinlichkeit, nach dem in dieser Auswertung als perfekt gewerteten Fitnesswert fehlzuschlagen, sehr gering ist. Auch in dieser Auswertung wurden zu jedem zu vergleichenden Wert zehn Durchläufe durchgeführt, um Ausreisser feststellen zu können. Dies erweist sich auch als nötig, denn bei den ersten zehn Durchläufen mit einer Populationsgrösse von 100 wurden zwischen 2 und 10 Generationen benötigt, um ein perfektes Individuum zu schaffen.

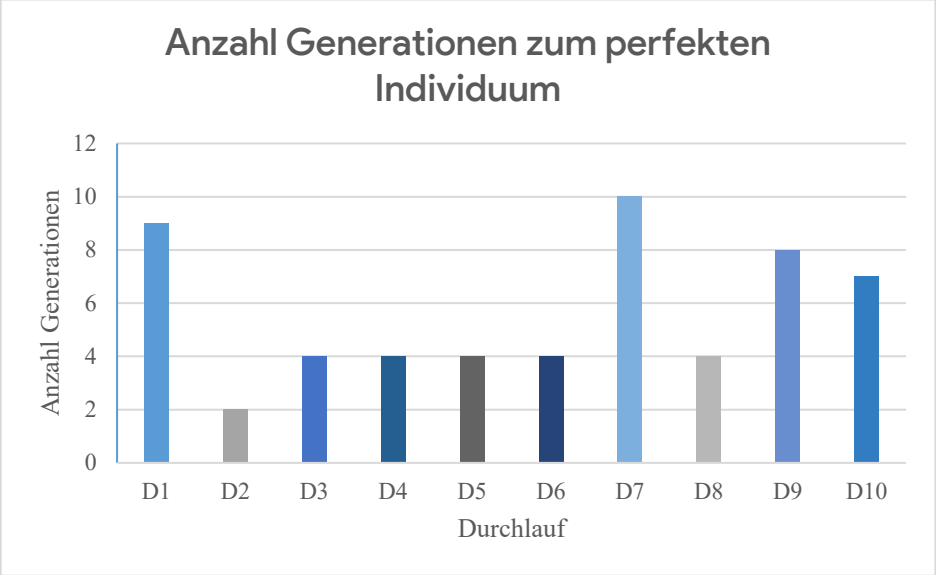


Abb. 36 Benötigte Generationen zur Erzeugung eines perfekten Individuums (siehe Tabelle 1)

Die Differenz zwischen diesen beiden Randwerten ist extrem. Somit kann das Lösen einer Problemstellung plötzlich ein Vielfaches der Zeit benötigen, verglichen mit vergangenen Durchläufen.

Ein Blick auf die Fitnesswerte der erfolgreichsten Individuen einer Generation unterstützt die Annahme, dass es sich beim Erreichen eines Fitnesswertes von 100′000 um ein fast schon perfektes Individuum handelt.

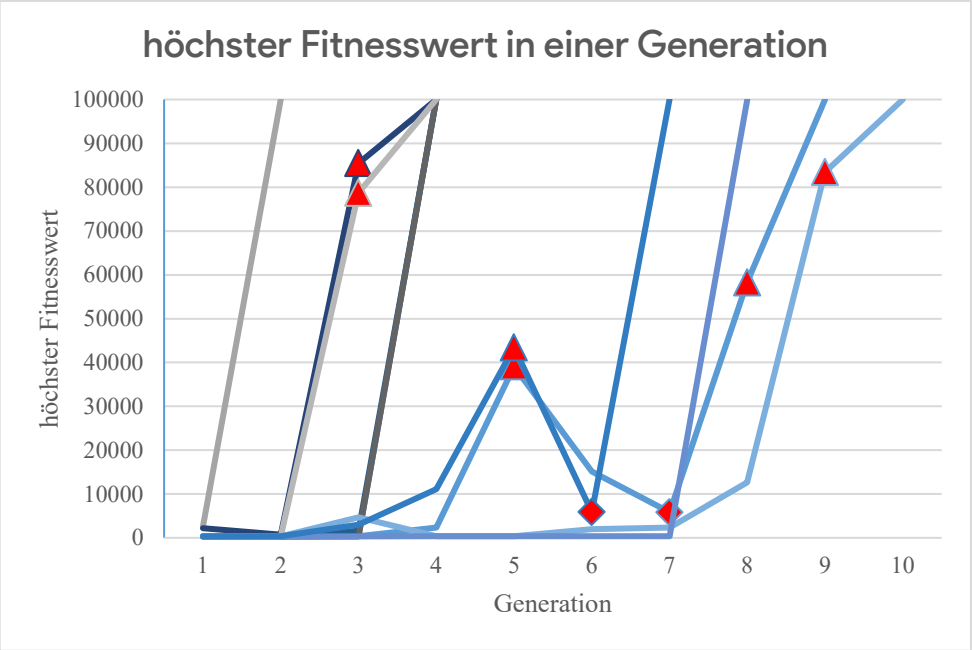


Abb. 37 Fitnesswert beim Fehlschlag einer Generation (siehe Tabelle 1)

Die meisten Fehlschläge in einem Durchlauf fanden bei einem Fitnesswert zwischen 250 und 350 statt. Ein kleiner Teil fand darüber, aber noch unter 10′000 statt. Erstaunlich ist jedoch, dass bei mehr als der Hälfte der Durchläufe einer der Fehlschläge zwischen 50′000 und 90′000 stattgefunden hat (siehe Abb. 37 ▲). Nie aber fand solch ein Fehlschlag in einem Durchlauf doppelt statt. Auch existierte kein Fehlschlag, dessen Fitnesswert bei mehr als 90′000 stand. Ausschliessen lässt sich ein Fehlschlag nach 100′000 aber immer noch nicht, hingegen kann man von einer sehr kleinen Wahrscheinlichkeit eines solchen Fehlschlages ausgehen.

Was bei der Auswertung der Daten anfangs auch für grosse Aufmerksamkeit gesorgt hat, ist, dass sich die Fitnesswerte der perfekten Individuen, welche eigentlich auch den Fortschritt des Trainingsprozesses repräsentieren, während der Trainingsprozesse teils wieder verschlechtert haben (siehe Abb. 37 ♦). In genetisch basierten Prozessen kann dies dadurch entstehen, dass sich aus Individuen mit guten und schlechten Eigenschaften Individuen bilden, die nur die schlechten Eigenschaften übernehmen. Der genetische Algorithmus kann selbst nicht einschätzen, welche Eigenschaften optimal sind und welche besser nicht weitergegeben werden sollen. Um dies zu vermeiden, sorgt die in diesem Projekt implementierte Bibliothek dafür, dass das erfolgreichste Individuum einer Generation, inklusive all dessen Eigenschaften, automatisch in die nächste Generation übergeben wird, ohne verändert zu werden. Die nächste Generation wird sich nicht zwingend verbessern, dürfte sich aber auch nicht verschlechtern.

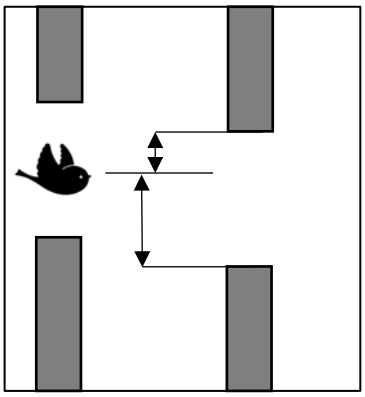


Abb. 38 Hinderniskonstellation 1

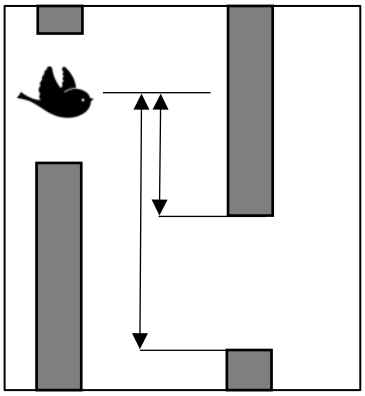


Abb. 39 Hinderniskonstellation 2

Das Verschlechtern des höchsten Fitnesswertes der folgenden Generation kann aber anhand des mit dem genetischen Algorithmus zu lösenden Problems erklärt werden. Auch wenn die Problemstellung (siehe *4.2.2 Beispiel Flappy Bird*) von aussen immer gleich definiert wird, wird in der Umsetzung auf zufällige Werte gesetzt. Die Lücke in den Hindernissen wird zufällig generiert und ist nicht bei jedem Durchlauf an gleicher Stelle. Zusätzliche Herausforderungen können entstehen. Vergleicht man *Abb. 38* mit *Abb. 39*, so lässt sich leicht erkennen, dass das Individuum bei der ersten Abbildung dank der optimalen Hinderniskonstellation nur eine kleine Höhendifferenz überwinden muss. Dank dem genügend grossen Abstand zwischen den beiden Hindernissen, bleibt dem Individuum viel Spielraum, wann es sich der neuen Höhe angleichen soll. Ist aber wie in der zweiten Abbildung die Höhendifferenz sehr gross, so muss das Individuum gerade nach dem Durchfliegen des ersten Hindernisses beginnen sich der neuen Höhe anzugleichen, denn ein Aufstieg oder Abstieg benötigt Zeit. Das Spiel ist so konzipiert, dass es immer möglich ist, sich im gegebenen Zeitfenster der neuen Höhe anzupassen. Der Spielraum, wann diese Anpassung geschieht, kann sich abhängig von der Hinderniskonstellation aber drastisch verändern. Das Verschlechtern des höchsten Fitnesswertes einer nachfolgenen Generation ist in diesem Falle nicht auf das Versagen des Trainingsprozesses zurückzuführen, sondern auf die zusätzlichen Herausforderungen, die während Durchläufen zufällig entstehen können.

Bis jetzt basieren alle Auswertungen des genetischen Algorithmus auf der Anzahl Generationen, die benötigt werden, um ein perfektes Individuum zu erschaffen. In der Praxis spielt dies aber keine Rolle, denn es gibt Generationen, die sehr schnell durch eine nächste ersetzt werden, da ihr Fehlschlag schon innerhalb der ersten paar Berechnungen stattfindet. Andere Generationen erleiden ihren Fehlschlag erst nach einer grossen Menge an Berechnungen und benötigen somit viel mehr Rechnerressourcen und Zeitaufwand als die schneller fehlschlagenden Individuen. Deshalb wird bei der Auswertung auf den Zeitaufwand geschaut. Der Zeitaufwand basiert nicht auf der tatsächlichen zeitlichen Trainingsdauer, sondern gibt die Anzahl an benötigten Berechnungen zurück, die nötig sind, um ein komplettes Training durchzuführen. Da beim *Flappy Bird* Beispiel pro geladenes Frame im Browser eine komplette Trainingsberechnung durchgeführt wird, wird der Zeitaufwand als Anzahl Frames definiert.

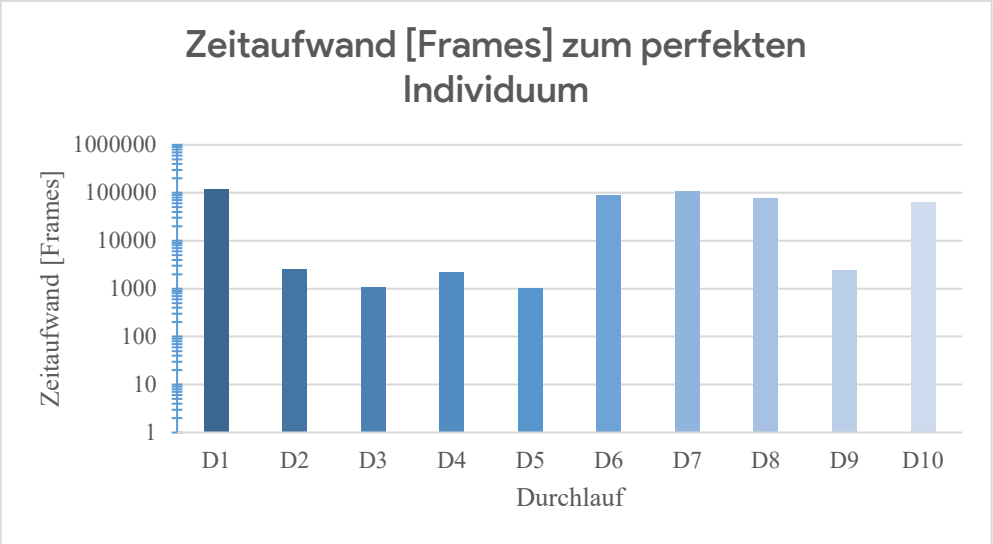


Abb. 40 Zeitaufwand zur Erzeugung eines perfekten Individuums (siehe Tabelle 1)

Betrachtet man den Zeitaufwand, so stellt man fest, dass sich die Durchläufe noch weiter voneinander unterscheiden. War zuvor bei den Generationen der höchste Wert das fünffache des tiefsten Wertes, so ist es nun ein Faktor von ca. 100 ($\frac{\sim 100'000}{\sim 1000}$).

Als Fazit lässt sich sagen, dass der genetische Algorithmus im Gegensatz zur Backpropagation ganz unterschiedlich effizient sein kann, auch wenn die Startbedingungen gleich sind, und dass die benötigten Generationen am Ende keine grosse Rolle gegenüber dem Zeitaufwand spielen.

5.1.2.1 Optimale Populationsgrösse

Neben der Backpropagation verfügt auch der genetische Algorithmus über Variablen, die vom Nutzer definiert werden. Beispielsweise muss eine passende Populationsgrösse gewählt werden, um die höchste Effizienz aus dem genetisch basierten Training zu erzielen.

Die Populationsgrösse beeinflusst stark den Fortschritt, der während einer Generation geschehen kann. Ist die Population gross, so werden mehr Kombinationen an Eigenschaften ausprobiert und mehr neue Eigenschaften erzeugt. Hingegen führt eine grosse Population zu zusätzlicher Rechnerauslastung. So können weniger Trainingsschritte pro Sekunde ausgeführt werden. Da das *Flappy Bird* Beispiel direkt in einem Browser ausgeführt wird und dadurch auch alle Berechnungen des genetischen Algorithmus auf dem Rechner des Nutzers ausgeführt werden, kann sich die Anzahl an Trainingsschritten pro Sekunde stark unterscheiden. In diesem Beispiel wird pro geladenes Frame im Browser ein Trainingsschritt ausgeführt. Die Anzahl an Trainingsschritten pro Sekunde kann also anhand von Frames pro Sekunde (*FPS*) dargestellt werden. Da moderne Browser die *FPS* in einem *HTML-Canvas* standardmässig limitieren, musste zur Datengewinnung das *FPS*-Limit deaktiviert werden. Dies geschieht mit dem Argument «`--disable-frame-rate-limit`» beim Ausführen eines *chromium*basierten Browsers.

Für jede Populationsgrösse wurden zehn Durchgänge gemessen. Pro Durchgang wurden die durchschnittlichen *FPS* ermittelt, sowie die Anzahl benötigter Generationen zur Erzeugung eines perfekten Individuums. Zum Vergleichen der Effizienz verschiedener Populationsgrössen wurden für alle Durchgänge einer Populationsgrösse jeweils die Durchschnittswerte, sowie der tiefste und höchste Wert der Anzahl Generationen bestimmt. Dadurch lässt sich ein Bereich definieren, welcher angibt, wie viele Generationen voraussichtlich bei einer bestimmten Populationsgrösse benötigt werden. Dies wurde

nur bei der Anzahl Generationen umgesetzt, da sich dieser Wert bei gleichen Startvoraussetzungen stark unterscheiden kann, wobei bei den *FPS* der Wert fast immer gleich ist. Ein Durchschnittswert allein ist also nicht genug aussagekräftig zum Aufzeigen der benötigten Generationen zur Bestimmung eines perfekten Individuums.

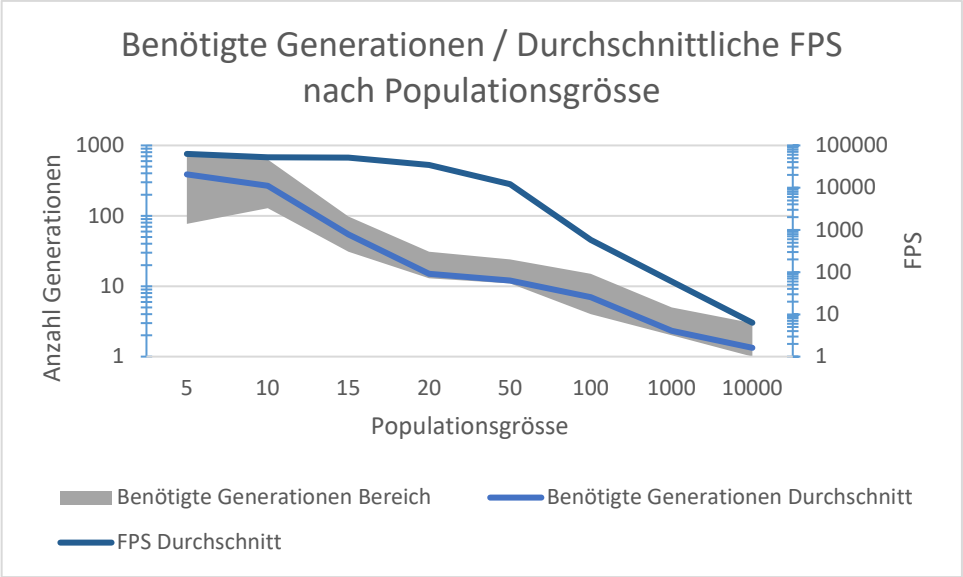


Abb. 41 Auswirkungen der Populationsgrösse (siehe Tabelle 7)

Wie in *Abb. 41* zu erkennen ist, nehmen benötigte Generationen sowie durchschnittliche *FPS* bei steigender Populationsgrösse ab. Auch ist zu erkennen, dass sich bei einer kleineren Population das Spektrum der Anzahl benötigter Generationen um vieles vergrössert. Somit kann sich bei der Populationsgrösse 5 die benötigte Generationenanzahl zwischen 70 und 700 befinden. Wobei es sich hier um einen Differenzfaktor von 10 handelt. Bei tiefen Populationsgrössenwerten wie bei 1000 liegt der Faktor zwischen den Extremwerten 2 und 3 nur bei 1.5. Dies lässt sich darauf zurückführen, dass sich bei einer neuen Generationserzeugung mit einer kleinen Population nur wenige Crossover-Möglichkeiten ergeben, wodurch es dem Zufall entspricht, wann ein optimales Crossover stattfindet. Bei einer grossen Population kann beim Crossover der erfolgreichsten Individuen eine grosse Menge an weiteren Individuen erzeugt werden, wodurch mehr Möglichkeiten des Crossovers ausprobiert werden können.

Die *FPS*-Werte sollten nicht als absolute Referenzwerte angesehen werden, da sie sich stark unterscheiden können, abhängig davon welche Leistung ein Rechner bei den Berechnungen bieten kann, sowie welche Software- und Betriebssystemlösungen eingesetzt werden. Alle Berechnungen wurden auf einem auf *Windows 11* basierenden Rechner mit einem *Intel i7-9300F* mit einer Basisgeschwindigkeit von 3.00GHz ausgeführt. Als Browser wurde *Google Chrome Beta* auf der Version 118 eingesetzt, um zusätzliche browserbasierende Hintergrundprozesse deaktivieren zu können. Auch wurden durch Flags und Startargumente die Frameratelimit sowie GPU-basierende Berechnungen deaktiviert. Ebenso kann die Auswertung durch zur Messung benötigte Methoden beeinflusst werden. Denn dank der im Browser integrierten schlecht auslesbaren und für diese Auswertung ungeeigneten *FPS*-Messmethoden, mussten eigene Funktionen in die Beispielanwendung eingebaut werden. Wegen dem zusätzlichen Auslesen der Zeitdifferenzen zwischen Frames findet ein zusätzlicher Zeitaufwand von bis zu einer Millisekunde statt. Somit liegen die Messwerte leicht neben der Realität.

Die gemessenen *FPS*-Werte veranschaulichen, wo sich die optimale Populationsgrösse befindet, wenn man auf den für die Messungen eingesetzten Rechner sowie auf die verwendeten Beispielanwendungen

setzt. Berechnet man anhand der *FPS* und der Trainingszeit in Frames die tatsächliche Trainingszeit in Sekunden, so lässt sich leicht feststellen, in welchem Bereich sich die optimale Populationsgrösse befindet. Die Trainingszeit ist der Quotient aus der Anzahl benötigter Frames geteilt durch die durchschnittlichen Frames pro Sekunde. Wichtig ist hier anzumerken, dass es sich bei der Trainingszeit nur um die Dauer handelt, bis ein perfektes Individuum erschaffen wurde und nicht, bis bewiesen wurde, dass das Individuum perfekt ist. Würde man die Zeit für den Beweis einberechnen, so würde die prozentuale Differenz der Trainingszeiten um vieles kleiner werden. Gerade mit dem *Flappy Bird* Beispiel ist es eine grosse zeitliche Herausforderung, solch einen Beweis aufzustellen.

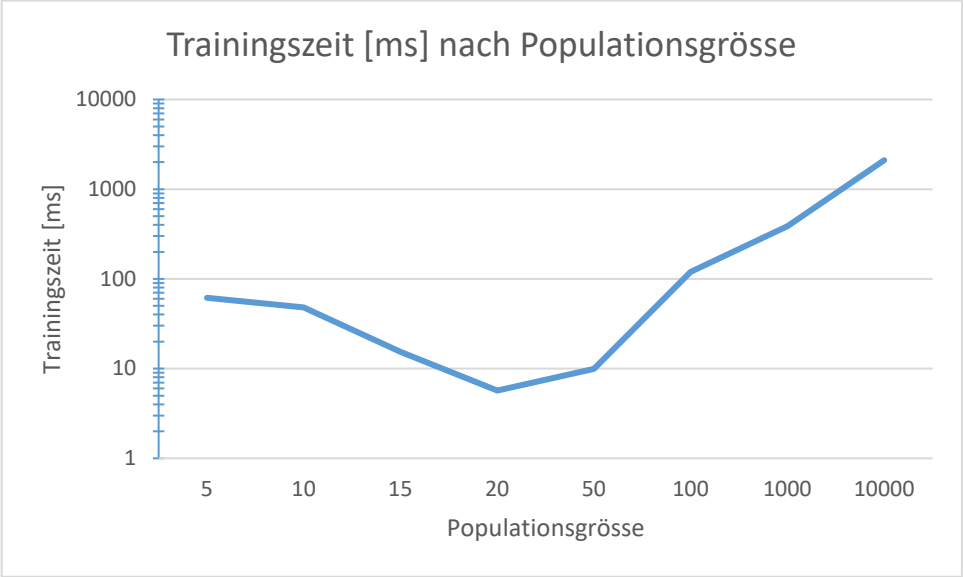


Abb. 42 Trainingszeit nach Populationsgrösse (siehe Tabelle 7)

Anhand *Abb. 42* ist nun zu erkennen, was für eine grosse Auswirkung die Populationsgrösse haben kann. Eine doppelt so grosse Population kann schon den zehnfachen Zeitaufwand erfordern. Es ist festzustellen, dass sich die optimale Populationsgrösse in der Nähe von 20 befindet.

Nun sind Werte wie tausende von *FPS* beim Benutzer nicht realistisch. Ein Alltagsnutzer würde nicht extra die Frameratelimit des Browsers deaktivieren. Somit kann man davon ausgehen, dass sich die *FPS* nie über 120 befinden werden. Anhand *Abb. 43* wird visualisiert, wie sich die Trainingszeiten mit einer Framerateeinschränkung verändern.

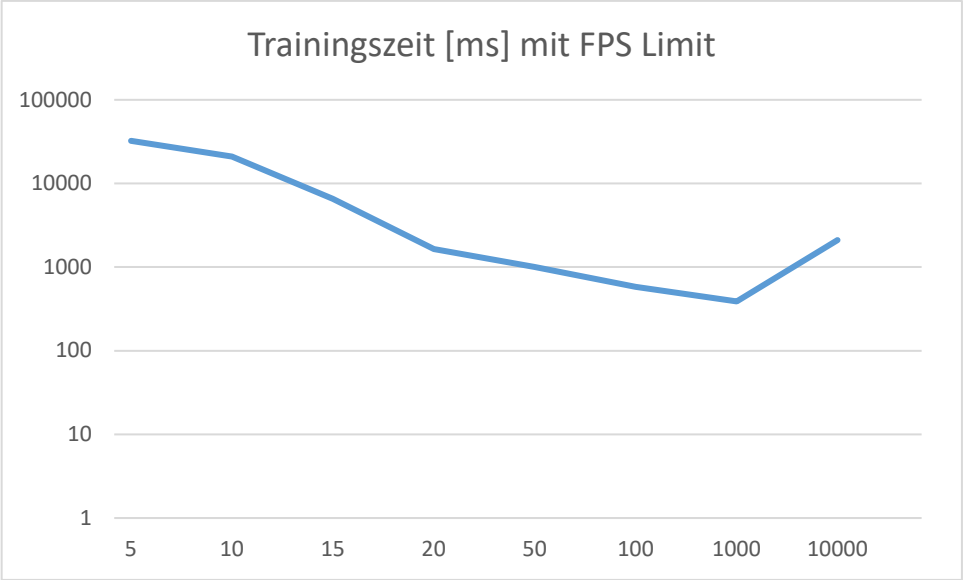


Abb. 43 Trainingszeit nach Populationsgrösse mit Frameratelimit (siehe Tabelle 7)

Verglichen mit *Abb. 42* wandert das Minimum in *Abb. 43* zu einer Populationsgrösse zwischen 100 und 1000 Individuen pro Generation. Populationsgrösse 100 ist auch der Wert, der standardmässig bei der Beispielanwendung aktiviert ist.

Werte wie Populationsgrösse 20 oder 100 wirken von aussen betrachtet klein und ineffizient. Diese tiefen Werte sind aber gerechtfertigt, da es sich beim *Flappy Bird* Beispiel um eine sehr einfach zu lösende Problemstellung handelt. Werden die Problemstellungen komplexer, so wird auch die optimale Populationsgrösse höher sein.

5.1.3 Stärken und Schwächen

Wie anhand der zuvor erläuterten Ergebnisse aus den Auswertungsdaten zu entnehmen ist, kann die Bibliothek alle Problemstellungen der Beispielanwendungen lösen. Auch geschieht dies in einer angemessenen Zeit. Die Problemstellungen weisen aber nicht eine sehr hohe Komplexität auf. Der Fokus und dadurch die Stärke der Bibliothek liegt also in einfachen Problemstellungen, die dafür dann sehr schnell gelöst werden können. Als Schwäche kann angesehen werden, dass die Bibliothek auf die Kapazitäten des ausführenden Browsers limitiert ist. Die Bibliothek entspricht somit der Fragestellung der Arbeit.

5.2 Effizienz der Beispielanwendungen

Die meisten Beispielanwendungen führen selbst wenige Berechnungen durch, wodurch sich an ihnen die Effizienz der Bibliothek gut messen lässt, ohne dass das Beispiel selbst die Resultate beeinflusst. Das *Car Game* Beispiel hingegen wurde schon im Verlauf der Arbeit als ungeeignet eingestuft. Hier folgt der Beweis für die zusätzliche Herausforderung, die dieses Beispiel bietet.

Das *Car Game* Beispiel litt speziell unter der grossen Anzahl an Individuen im genetischen Algorithmus. Da das *Flappy Bird* Beispiel den genetischen Algorithmus auf ähnliche Weise anwendet, ist es als Vergleich perfekt geeignet. Die folgende Grafik veranschaulicht die extremen *FPS*-Differenzen und deren Veränderung über mehrere Generationen hinweg.

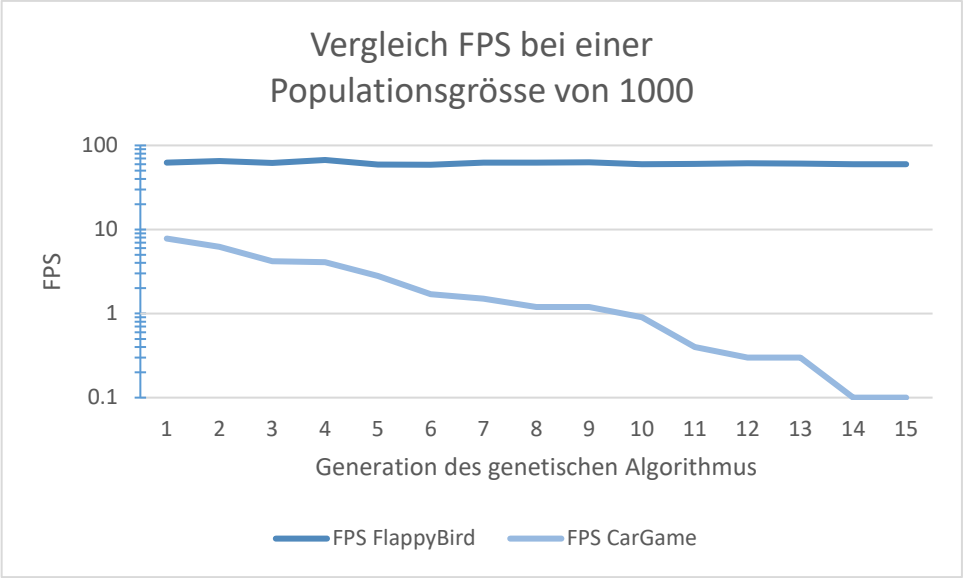


Abb. 44 Vergleich der FPS von Flappy Bird und Car Game (siehe Tabelle 8)

Schon bei der ersten Generation sind die Frames pro Sekunde des *Car Game* Beispiels nur ein Zehntel der des *Flappy Bird* Beispiels. Dies lässt sich darauf zurückführen, dass der Computer im schlechter abschneidenden Beispiel einer grösseren Last ausgesetzt ist, da es sich um eine dreidimensional generierte virtuelle Welt handelt und auf komplexere Formen und Objekte gesetzt wird. Diese Werte verschlechtern sich aber über jede weiter erzeugte Generation noch. Dies liegt daran, dass die 3D-Objekte der Individuen der vorherigen Generation nicht komplett gelöscht werden können. Somit füllt sich der Arbeitsspeicher über die Zeit immer mehr, was zu einer Verlangsamung des Systems führt. Da die Berechnungen neben der Bibliothek schon einen Grossteil der Rechenleistung anfordern, lässt sich anhand dieses Beispiels schlecht die Effizienz des genetischen Algorithmus bestimmen.

6. Diskussion Algorithmen

Nebst der Umsetzung des Algorithmus ist es auch von essenzieller Bedeutung, dessen theoretisches Potential und dessen Auswirkung zu hinterfragen. Hierbei geht es nicht um die Art der codebasierten Implementierung, sondern um die Auswertung der Algorithmenkombination als theoretisches Objekt.

6.1 Zusammenarbeit der Algorithmen

Zusätzlich zur Kombination von Feed Forward Neural Network und genetischem Algorithmus ist auch das Kombinieren der Lernmethoden Backpropagation und genetischer Algorithmus interessant. Beide bieten unterschiedliche Vorteile, die sich zu einem starken Gesamtpaket vereinen lassen. Das Beispiel *CarGame* (siehe *4.2.3 Beispiel Car Game*) setzt auf diese Art von Algorithmenkombination. Zuerst wird mit der Backpropagation trainiert. Dabei wird auf schon existierende Trainingsdaten gesetzt. Diese Trainingsdaten können im Bereich von Computerspielen beispielsweise durch humane Interaktionen gewonnen werden. Im Beispiel handelt es sich dabei um eine Aufforderung an den Nutzer, er solle zuerst selbst durch Tastaturinteraktionen versuchen, das Fahrzeug durch die Strecke zu führen. Dabei werden während jedem Frame die durch das Fahrzeug wahrgenommene Umgebung und die durch den Nutzer momentan durchgeführten Interaktionen gespeichert. Dadurch entsteht ein Datensatz, dessen Qualität auf den Fähigkeiten des Nutzers basiert. Mit diesem Datensatz kann nun ein Neuronales Netzwerk trainiert werden. Die Umgebungswahrnehmungen des Fahrzeuges sind die Inputs für das Netzwerk und die Benutzerinteraktionen die gewünschten Outputs. Es wird hierbei die Annahme getroffen, das Handeln des Nutzers entspreche dem idealen Handeln.

Das Prinzip der Backpropagation arbeitet ohne Verwendung von zufälligen Werten, abgesehen von der ersten Initialisierung eines noch untrainierten Netzwerkes. Dem Trainingsalgorithmus liegen allein die zur Verfügung gestellten Datensätze zugrunde. Dies führt dazu, dass nach einem erfolgreichen Training das Netzwerk nur über jene Fähigkeiten verfügt, die zur Generierung der Trainingsdaten vorhanden waren. Hingegen besitzt das Netzwerk keine weiteren Eigenschaften, die zu viel mehr fähig wären als nur die Trainingsdaten zu reproduzieren. Es wird bereits vor dem Trainingsprozess davon ausgegangen, die Trainingsdaten entsprächen der Qualität des gewünschten Resultates. Die Qualität wird dadurch definiert, wie erfolgreich das Netzwerk in jeder möglichen Situation handeln wird.

Bei der Anwendung in Computerspielen, wie beispielsweise beim *CarGame*, bei der auf Trainingsdaten, erzeugt durch Benutzerinteraktionen, gesetzt wird, lässt sich annehmen, dass es sich bei der Qualität dieser Trainingsdaten nicht um die höchstmögliche handelt. Das Ziel liegt aber meist darin ein möglichst qualitativ hochwertiges Resultat zu erzielen und nicht nur den Nutzer zu imitieren. Deshalb wird nach dem Training mit der Backpropagation auf ein weiteres Training mit genetischen Algorithmen gesetzt. Die genetischen Algorithmen können hingegen ohne Trainingsdaten arbeiten und setzen auf zufällige Werte. Diese zufälligen Werte indessen garantieren noch keine Erhöhung der Qualität, können aber dazu führen. Mit den implementierten Selektionsprinzipien hingegen lassen sich Qualitätsverschlechterungen vermeiden, was dazu

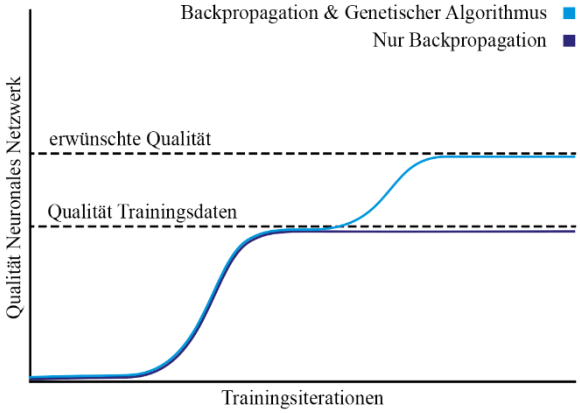


Abb. 45 Qualitätserfolge nach Auswahl der Algorithmen

führt, dass nur das Abwarten mehrerer Generationen nötig ist, um zur Entstehung eines qualitativ hochwertigeren Resultates zu kommen.

Das Kombinieren dieser Algorithmen ermöglicht folglich das Bewahren des Vorteils eines Trainingsprozesses mit zuverlässigem Fortschritt durch die Backpropagation und des Vorteils des genetischen Algorithmus, unabhängig von der Qualität der Trainingsdaten ein qualitativ hochwertiges Resultat zu erzielen.

Es wird nicht nur mit dem genetischen Algorithmus gearbeitet, da vor allem bei eher komplexen Anwendungen, bei denen ein Erfolg lange ausbleiben kann, der Fitnesswert speziell bei qualitativ schlechten Individuen schwer zu bestimmen sein kann. Dadurch wird der Selektionsprozess erschwert, und das Erreichen eines Individuums, das sich zufälligerweise in die richtige Richtung weiterentwickelt, kann etliche Generationen ausbleiben. Die Backpropagation bietet hierfür solidere Anhaltspunkte beim Beginn eines Trainingsprozesses, da nicht abgewartet werden muss, bis sich etwas zufälligerweise richtig weiterentwickelt. Ist der Trainingsprozess jedoch weit genug fortgeschritten, kann auf den genetischen Algorithmus gesetzt werden, weil die Individuen nun ein zumindest teilweise richtiges Handeln aufweisen, was das Bestimmen eines Fitnesswertes erleichtert.

Der Wechsel von der Backpropagation zum genetischen Algorithmus kann auf zwei Arten determiniert werden. Einerseits kann abgewartet werden, bis kein weiterer Trainingsfortschritt mehr mit der Backpropagation stattfindet. Determinieren lässt sich dies dann, wenn sich der Fehlerwert nur noch minimalst verändert. Dadurch kann der Trainingsprozess mit Trainingsdaten beendet werden, wenn die Backpropagation nicht mehr die Fähigkeiten aufweist, weitere Optimierungen vorzunehmen. Dies bedeutet folglich, dass sich der Fehlerwert nicht schon am theoretisch minimal Möglichen angenähert haben muss. Andererseits lässt sich der Wechsel auch dann durchführen, wenn die für den genetischen Algorithmus zu erzeugenden Individuen einen genügenden Fähigkeitenpool aufweisen, aus dem sich nutzbare Fitnesswerte schliessen lassen. Dies kann aber je nach Anwendungsfall schwer zu bestimmen sein, möchte man nicht in Intervallen einen Versuch des Trainings mit genetischen Algorithmen durchführen, dessen Erfolg nach wenigen Schritten determinieren und dann entscheiden, ob dieser schon den Erwartungen entspricht.

6.1.1 Lokale Extrema

Auch wenn die Backpropagation versucht den Fehlerwert zu minimieren, bedeutet dies noch lange nicht, dass dieser auf das minimal Mögliche zugehen wird. Einerseits kann am Minimum vorbeigewandert werden, was in *3.2.1 Gradient Descent* bei der Determinierung eines idealen Wertes für die Lernrate schon diskutiert wurde, andererseits kann der Trainingsprozess in ein sogenanntes lokales Minimum abdriften.

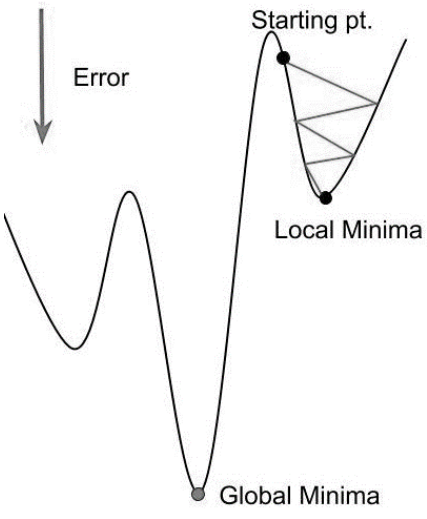


Abb. 46 lokales Minimum [13]

Im Gradientenverfahren existieren nur die Informationen eines Gradienten, durch die sich bestimmen lässt, in welche Richtung es zu einem tieferen Punkt geht. Dies bedeutet jedoch nicht, dass es sich bei diesem tieferen Punkt um den tiefst möglichen Punkt handelt. Findet nun eine Annäherung während des Trainingsprozesses an einen lokalen Minimalpunkt statt, so ist es nicht möglich, sich wieder aus diesem Minimum hinauszubewegen, denn das hätte eine temporäre Verschlechterung, beziehungsweise Erhöhung des Fehlerwertes zur Folge. Dies wird nicht geschehen, denn sobald

sich der Fehlerwert in einem Trainingsschritt erhöht, wird im nächsten Trainingsschritt eine Bewegung in die genau entgegengesetzte Richtung stattfinden, da der negative Gradient zurück in das lokale Minimum zeigt.

Der Einsatz von genetischen Algorithmen kann dies ebenfalls vermeiden. Da beim genetischen Algorithmus auch Individuen in die falsche, sich aus dem Minimum entfernender Richtung erzeugt werden, kann es vorkommen, dass ein Individuum den Bereich, der zum lokalen Minimum führt, komplett verlässt und sich in Richtung des globalen, beziehungsweise anderen lokalen Minimums weiterentwickeln kann. Ist das neu erreichte Minimum suboptimaler als das der anderen Individuen, so wird das abgedriftete Individuum oder dessen Nachkommen, zusammengefasst als den Genpool dieses Individuums, bei zukünftigen Selektionsprozessen tiefere Chancen haben, wodurch der Genpool aussterben wird. Natürlich kann ein lokales Minimum auch den genetischen Algorithmus vom Weiterentwickeln fernhalten, da die zufällig veränderten Werte der Individuen nur leicht von denen der vorherigen Generation abweichen. Mit Parametern wie der Lernrate oder dem Threshold-Wert lassen sich diese Veränderungen aber erhöhen, was trotzdem ein Ausbrechen aus einer konvexen, sich um das lokale Minimum umgebenden Kurve ermöglichen kann. Die genetischen Algorithmen verfügen selbst über keine Informationen in welcher Richtung bessere Lösungen liegen. Eine höhere Lernrate führt dadurch zu mehr Variation in den Individuen – auch in sich verschlechternde Richtungen – was das «Ausbrechen» aus einem Minimum ermöglichen kann.

6.1.2 Durchlaufzeit CarGame

Welchen Nutzen das Weitertrainieren mit dem genetischen Algorithmus hat, ist am Beispiel *CarGame* zu erkennen. Wird nach dem Backpropagation-Training mit den durch den Nutzer gewonnenen Daten der genetische Algorithmus angewendet, so lässt sich trotzdem noch eine Verbesserung erkennen, auch wenn bei der Backpropagation der Fehlerwert schon minimiert wurde. Das vortrainierte Objekt besitzt demzufolge noch nicht den maximal möglichen Fitnesswert. Wie in *Abb. 47* zu erkennen, ist es sehr davon abhängig, welche Qualität die Trainingsdaten bereits aufweisen, ob danach noch eine starke Verbesserung möglich ist. Bei den hier gezeigten Beispielen handelt es sich um Rennstrecken verschiedener Form, die die Individuen durchfahren mussten. Gemessen wurde der Zeitaufwand für das Durchfahren der Strecke nach dem Training mit der Backpropagation und nach 2, 20 und 200 Generationsdurchläufen. Bei Beispiel 3 waren die Möglichkeiten einen Pfad zu suchen viel grösser als bei Beispiel 1, wo es nur um die Optimierung weniger Millisekunden ging. Nach mehreren Generationen konnte bei Beispiel 3 ein optimierter Weg gefunden werden.

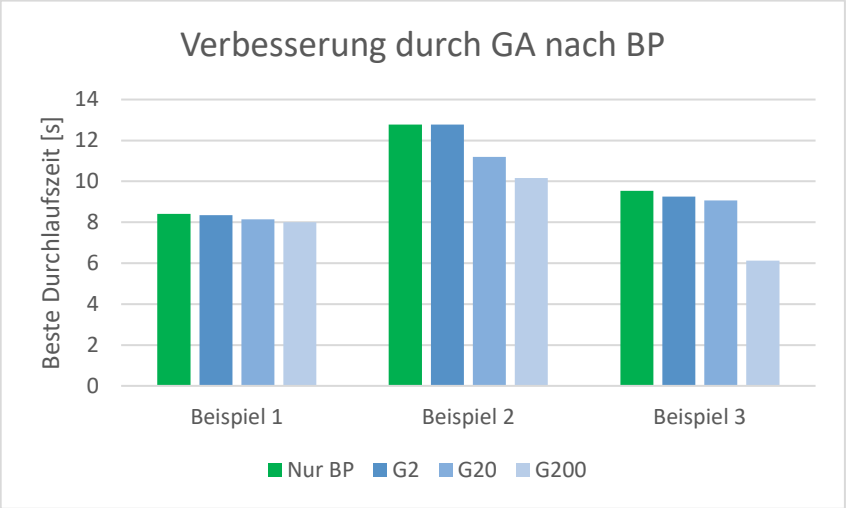


Abb. 47 Optimierung durch GA (siehe Tabelle 9)

6.1.3 Alternative Anordnung

Spricht man von der Zusammenarbeit von Backpropagation und genetischen Algorithmen, so wird die Anwendungsreihenfolge meist invertiert durchgeführt, verglichen mit der hier diskutierten Anordnung. Es wird mit genetischen Algorithmen gestartet und die Backpropagation darauffolgend eingesetzt, um letzte kleine Optimierungen vorzunehmen.

Diese alternative Anordnung eignet sich besonders gut, um mit dem Problem der lokalen Extremalpunkte klarzukommen. Es wird bei den genetischen Algorithmen auf starke Veränderungen pro Generation gesetzt und somit eine grosse Lernrate sowie einen grossen Schwellenwert genutzt. Sobald sich ein Individuum dem Optimum angenähert hat, wird mit der Backpropagation weitergearbeitet. Die Backpropagation führt das zu trainierende Objekt mit einer kleinen Lernrate möglichst nah zum nächsten Minimum hin, bei dem es sich idealerweise um das globale Minimum im Definitionsbereich handelt.

Das Tauschen der Algorithmen in der zeitlichen Abfolge, vernachlässigt hingegen die Kombination mit dem Effekt der ursprünglich erwünschten Problembehebung. Wird mit der Backpropagation abschliessend gearbeitet, so ist es erneut nicht möglich, über die Einschränkungen der Qualität der Trainingsdaten hinaus zu arbeiten.

6.1.4 Vergleich Reinforcement Learning

Beim einfachen Betrachten kann der genetische Algorithmus mit dem Prinzip des Reinforcement Learnings (RL) verglichen werden, einer Art von maschinellem Lernen, das ebenfalls mit einem Individuum arbeitet, welches sich in einer Umwelt befindet und sich über die Zeit verbessert. Dabei wird das als Agent bezeichnete Individuum konstant überwacht und dessen Handlung beurteilt. Positiv herausstechende Handlungsschritte werden belohnt. So lernt der Agent, wie er in der Umwelt handeln soll.

Das Reinforcement Learning unterscheidet sich primär darin, dass es einen Intra-Life-Lernprozess darstellt. Der Agent lernt während seines Aufenthalts in der Umwelt. Beim genetischen Algorithmus hingegen, einem Inter-Life-Lernalgorithmus, lernt nicht das Individuum selbst, sondern es findet eine Weiterentwicklung statt durch das Erzeugen neuer Individuen basierend auf den vorherigen. Zudem ist der genetische Algorithmus ein eher unkontrollierter Prozess, bei dem mit vielen Zufallswerten gearbeitet wird, wohingegen im RL häufig mit Gradienten gearbeitet wird.

Das RL ist somit ideal geeignet, um Strategien zu finden und zu versuchen das Resultat zu maximieren. Im Unterschied dazu sind genetische Algorithmen so einfach gehalten, dass sie praktisch bei fast jedem Optimierungsproblem eingesetzt werden können.

6.2 Generalisierungsversuch

Ein essenzieller Nachteil zu genetischen Algorithmen, welcher beispielsweise schon beim Reinforcement Learning besser zu handhaben ist, ist das Bestimmen von nicht direkt erkennbaren Schwächen eines Individuums. Dadurch kann es schwer zu bestimmen sein, wann ein Individuum die gewünschten Eigenschaften aufweist. Dies ist gut erkennbar beim Beispiel *Flappy Bird*: Es ist sehr schwer zu beurteilen, wann das Individuum genug gut mit der Umwelt zurechtkommt, so dass es bis ins Unendliche überleben könnte. Auch nach einer unermesslich grossen Zeit, könnte ein neuer Extremfall in der Umwelt auftreten, der das Individuum zum Fehlschlag zwingt. Folglich war das Individuum nur scheinperfekt. Dieses Problem kann mit einem Halteproblem verglichen werden, da man erst nach dem

Testen aller möglichen Inputs determinieren kann, ob das Individuum wirklich perfekt ist. Die Menge dieser Inputs ist aber unbeschränkt und so lässt sich nur bei einem Fehlschlag beurteilen, dass das Individuum nicht perfekt ist, aber nie, ob das Individuum wirklich dauerhaft perfekt ist.

6.2.1 Bestimmung eines perfekten Individuums

Da es sich bei *Flappy Bird* um ein übersichtlich einfaches Anwendungsbeispiel handelt, lässt sich zumindest eine Annäherung des Perfektseins bestimmen. Die Idee ist es, ein Individuum den extremsten Umweltverhältnissen auszusetzen und dann darauf zu schliessen, dass es beim Klarkommen dieser Extremfälle auch einfachere Fälle bewältigen wird. Als Extremverhältnis wurde bei *Flappy Bird* eine Abfolge von Hindernissen erzeugt, bei der dreimal nacheinander zwischen der höchstmöglichen Position für die Hindernislücke sowie der niedrigstmöglichen Lücke abgewechselt wird. Das Wiederholen des Extremfalles direkt mehrmals hintereinander, wurde durchgeführt, da zu erwarten ist, dass das Individuum durch den Extremfall leicht geschädigt werden könnte. Im Beispiel wäre dies eine suboptimale Ausgangsposition nach der Extremsituation, die nachfolgendes Handeln erschwert. Von allen Individuen (insgesamt ein Testpool von 100 Individuen), die die erzeugte Extremsituation überstanden hatten, konnten alle die darauffolgende wieder zufällig generierte Umwelt problemlos meistern und kein weiterer Fehlschlag konnte festgestellt werden.

Flappy Bird bietet dank der geringen Komplexität die Möglichkeit des einfachen Bestimmens dieser Extremfälle. Andere Anwendungsfälle können hingegen nicht immer so einfach finalisiert werden. Je nach Anwendungsfall kann es sogar unmöglich sein, das Extrema der Herausforderungen zu finden.

6.2.2 Änderung Umweltparameter

Das Verhalten eines Individuums in einer Umwelt, vergleichbar mit der des Trainings, kann viel über dessen Qualität aussagen, ebenso jedoch auch wie dieses Individuum auf Änderungen in den Gesetzen der Umwelt reagiert, auch wenn es sich selbst während des eigenen Lebens nicht verändern kann. Kommt das Individuum mit leicht veränderten Umweltparametern klar, so lässt es sich als qualitativ hochwertiger einschätzen, als ein Individuum, das in der darauf trainierten Umwelt problemlos zurechtkommt, aber in einer anderen Umwelt erfolglos ist.

Natürlich darf nicht erwartet werden, Individuen würden plötzlich auf Aufforderung eine ganz andere Handlung ausführen können, die sich im Extremen von der trainierten unterscheidet. Das leichte Abändern der Umwelt stellt aber ein ideales Experiment dar.

6.2.2.1 Abänderung Hindernisabstände

In diesem Versuch werden die Abstände der Hindernisse im Beispiel *FlappyBird* leicht verändert. Je 100 Individuen, trainiert mit den standardmässigen Umweltparametern inklusive der ursprünglichen Abstände, werden der Umwelt ausgesetzt.

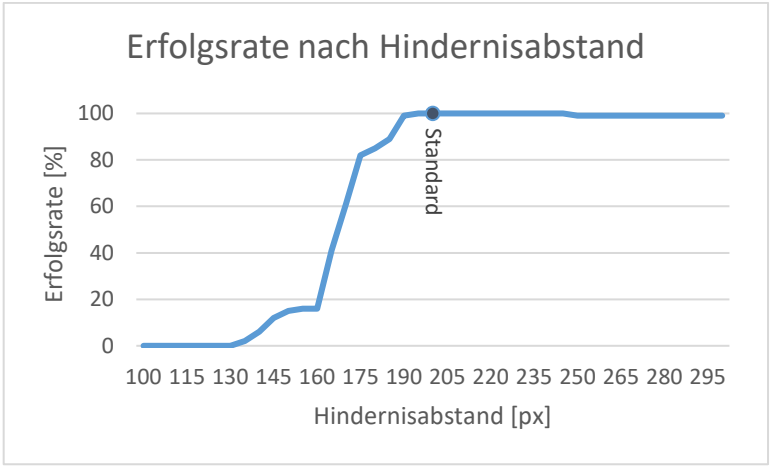


Abb. 48 Erfolgsrate basierend auf Hindernisabstand (siehe Tabelle 10)

Zu erkennen ist, dass von den Individuen, die mit Hindernisabständen von $200px$ trainiert wurden, ein Grossteil mit einer leichten Verkleinerung des Abstandes klarkommen. Wird der Abstand vergrössert, so überlebt fast jedes Individuum. Dies bedeutet, dass die Individuen basierend auf dem Abstand zum folgenden Hindernis handeln und nur Mühe haben, bei eng aufeinanderfolgenden Hindernissen schnell genug zu handeln.

6.2.2.2 Abänderung Lückengrösse

Weiter wurde die Qualität der Individuen mit der Veränderung der Lückengrösse in den Hindernissen getestet.

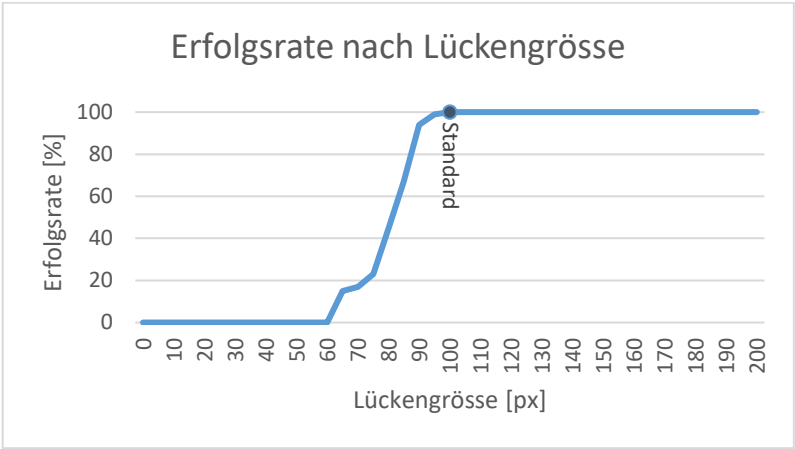


Abb. 49 Erfolgsrate basierend auf Lückengrösse (siehe Tabelle 10)

Hier ist ein noch schnellerer Abfall bei der Verkleinerung des vertikalen Abstandes zu erkennen als beim Abstand der Hindernisse horizontal. Dass bei $60px$ kein Individuum mehr Erfolg hat, kann dadurch begründet werden, dass es unmöglich ist, durch eine solch enge Lücke durchzukommen, da die Höhe eines Sprunges vordefiniert ist und Sprünge während dem Durchqueren des Hindernisses nötig sind. Versucht man das Training mit $60px$ anstatt $100px$, so kann nach vielen Generation immer noch kein Individuum die erste Lücke passieren. Die Einschränkung liegt demzufolge nicht bei der Qualität der trainierten Individuen, sondern bei den in der Simulation eingeschränkten Handlungsmöglichkeiten eines Individuums.

6.2.2.3 Abänderung Hindernisbreite

Der dritte veränderte Parameter ist die Breite des Hindernisses selbst.

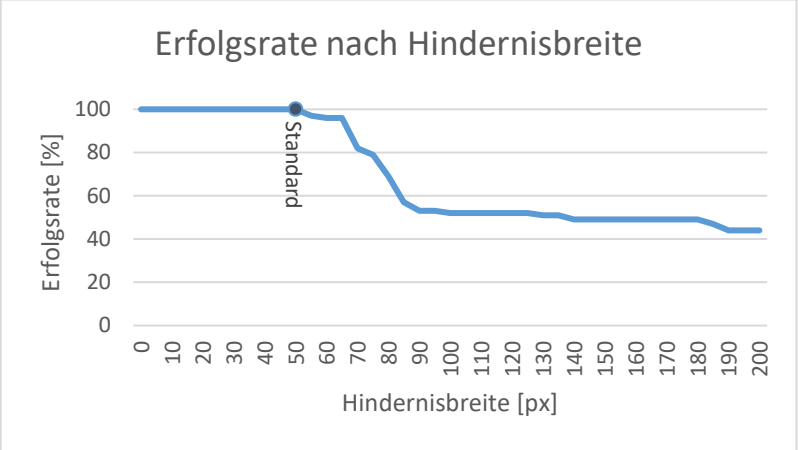


Abb. 50 Erfolgsrate basierend auf Hindernisbreite (siehe Tabelle 10)

Eine grössere Hindernisbreite stellt eine grössere Herausforderung dar. Allerdings sind alle Individuen geeignet für Werte, die niedriger als der Standard von $50px$ beim Training sind. Interessant ist, dass keine Limite existiert, bei der die Erfolgsrate auf null zugeht. Etwa die Hälfte der Individuen kann mit starken Veränderungen immer noch umgehen, während die andere Hälfte schon bei leichten Abweichungen erste Probleme hatte. Eine mögliche Begründung könnten zwei verschiedene Strategien sein, sich an die nächste Hindernislücke anzupassen. Individuen verfügen über keinerlei Informationen, wann das Hindernis endet, in dem sie sich befinden. Alle Individuen, die versuchen, sich so schnell wie möglich an das folgende Hindernis anzupassen, werden unweigerlich mit dem aktuellen Hindernis kollidieren. Bei der anderen Strategie erfolgt das Anpassen erst dann, wenn es nötig ist, denn die Individuen erhalten als Information den Abstand zum nächsten Hindernis. Bei dieser Veränderung der Umwelt ist es folglich sinnvoll, so spät wie möglich zu handeln.

6.3 Komplexere Evolution

Bei der in unserem natürlichen Umfeld herrschenden Evolution handelt es sich um ein weit komplexeres System als beim Konzept der genetischen Algorithmen. Nebst einer Art Konkurrenzkampf, entschieden durch den Fitnesswert, im genetischen Algorithmus umgesetzt mit der Selektion, können Umweltfaktoren dazu führen, dass auch weniger fitte Individuen eine Chance auf Weiterentwicklung erhalten und dafür fittere Individuen durch das Aussterben ihres Genpools bedroht sind. Ein Beispiel dafür ist die Bildung von verschiedenen Arten. Hinterfragen liesse sich nun, ob solche Situationen in einer genug komplexen Simulation, basierend auf den entworfenen genetischen Algorithmen, überhaupt vorkommen können und wenn sie dies tun, ob diese Prinzipien einen positiven Einfluss auf die Erzeugung eines perfekten Individuums haben können.

6.3.1 Artbildung

Im Prozess einer natürlichen Evolution ist es nicht selten vorzufinden, dass eine genetische Abspaltung an Individuen vom Hauptentwicklungsstrang stattfindet. Dies kann beispielsweise durch geographische Isolation erzeugt werden. Dabei handelt es sich um allopatrische Artbildung. Dabei entsteht ein grosser Vorteil für die Gesamtweiterentwicklung, da sich die abgetrennte Art weiterentwickeln kann, auch wenn die Ursprungsart ein lokales Fitnessmaximum erreicht hat – im Gegensatz zur Fehleroptimierung, spricht man hier von einem Maximum als Idealwert. Sobald die Fitness der abgetrennten Art höher ist, kann sie überhandnehmen.

Auch wenn die genetischen Algorithmen primär von den in der Natur vorzufindenden Evolutionsprinzipien inspiriert sind, ist das Entstehen von Arten zumindest bei den in der Bibliothek implementierten Algorithmen nur möglich, wenn die Fitnesswerte der beiden Arten äquivalent sind. Denn der Selektionsprozess lässt immer alle Individuen gegeneinander als Konkurrenz antreten und berücksichtigt nur den Fitnesswert unabhängig von der Umwelt, egal wie komplex die Individuen an sich sind. Hätte sich ein Individuum in der virtuellen Umwelt auf eine Nische spezialisiert und überlebte dadurch gegen generell fittere Konkurrenz, würde es beim Selektionsprozess nach einem Generationsdurchlauf trotzdem deselektiert werden, da nur ein Vergleich der Fitnesswerte über das Weiterkommen der Individuen entscheidet. Es gibt somit in dieser Art von Evolution keine Schutzmöglichkeiten vor Konkurrenz bei den Selektionsprozessen.

Umsetzbar wäre dies erst, würde man die Selektion nicht nach einem Generationsdurchlauf durchführen, sondern fortlaufend während der Simulation umsetzen. Dabei würde das Ziel der Problemlösung durch die Bibliothek verfehlt, da primär ein klares Resultat erreicht werden soll und nicht eine Ansammlung von Lösungsansätzen in Form verschiedener Arten.

6.3.2 Verwandtenselektion

Eine andere Art von Fitness ist die indirekte Fitness. Dabei handelt es sich um das Sichern eines Genpools durch das Unterstützen anderer Individuen, die einen Anteil des eigenen Genpools besitzen und dadurch ebenfalls das Weiterexistieren dieses Pool sicherstellen. In der natürlichen Umwelt ist dies bei vielen Tierarten vorzufinden. Beispielsweise wenn ein Nachkomme eines Vogelpärchens seine Eltern unterstützt, weitere Nachkommen grossziehen zu können, anstatt dafür zu sorgen, eigene Nachkommen zu erhalten. Dabei ist wichtig, dass der Nutzen, der daraus entsteht, grösser ist, als die Kosten die aufgewendet werden müssen, da ansonsten das Erzeugen eigener Nachkommen gewinnbringender gewesen wäre. Auch ist wichtig, dass das zu unterstützende Individuum nahe genug verwandt ist mit dem unterstützenden Individuum.

Betrachtet man die in dieser Arbeit implementierten Beispielanwendungen, so lässt sich dieses Prinzip ausschliessen. Es wird eine Umsetzung in ähnlichen Prinzipien realisiert. Alle Beispielanwendungen setzen die Individuen in der gleichen Umwelt aus, verhindern aber, dass sich die Individuen innerhalb der gleichen Generation beeinflussen können. Theoretisch befindet sich somit jedes Individuum in einer eigenen Umwelt, die eine Kopie der anderen ist. Würde man aber alle Individuen in der gleichen Umwelt mit Interaktionsmöglichkeiten untereinander aussetzen, so wäre der Aspekt des Unterstützen eines anderen Individuums zumindest nicht ausgeschlossen. Zu diesem Zweck müsste die Umwelt durch genügend grosse Komplexität die Möglichkeit bieten, ein supportives Verhalten durch Umweltparameter überhaupt zu ermöglichen.

Eine weitere Herausforderung stellt die Erkennung verwandter Individuen dar. Individuen müssten zuerst eine Möglichkeit entwickeln, um feststellen zu können, welche anderen Individuen einen gleichartigen Genpool besitzen, beziehungsweise eine nahe Verwandtschaft aufweisen.

Auch wenn sich dieses Phänomen mit den implementierten Algorithmen nicht ausschliessen lässt, so lässt sich dies auch nicht erwarten, da es sich um ein extrem komplexes Phänomen handelt, welches viele Entwicklungsschritte voraussetzt. Die Simulationsumgebung müsste höchst komplex sein und müsst ebenso den sich darin befindenden Individuen eine grosse Komplexität ermöglichen. Auch wäre bis zum Antreffen solch komplexer Individuen eine schlecht definierbare, hohe Anzahl an Generationen nötig. Betrachtet man die heutigen Möglichkeiten an Rechenleistung, so ist die Umsetzung sehr schlecht realisierbar.

7. Methodische Reflexion

Beim Entwickeln eines Produktes können immer Schwierigkeiten auftreten. Häufig läuft nicht alles nach Plan und im Nachhinein würde man vielleicht eine andere Vorgehensweise nutzen. In diesem Abschnitt wird auf den Arbeitsprozess zurückgeblickt, und eine kritische Beurteilung findet statt.

Anhand des erhaltenen Endproduktes lässt sich beurteilen, dass sich die methodische Vorgehensweise der Arbeit bewährt hat. Die am Anfang der Arbeit gesetzten Ziele konnten umgesetzt werden. So entstand innerhalb des zur Verfügung gestellten Zeitrahmens ein vollwertiges Produkt. Hingegen war die zeitliche Planung eine der grössten Herausforderungen. Der Aufwand verschiedener zu programmierender Prozesse wurde zeitlich eher knapp bemessen. So führten vor allem Zeitinvestments für codebasierte Problembehebungen zu einer leichten Verzögerung im selbst gesetzten Zeitplan. Vor allem die Website nahm sehr viel Zeit in Anspruch und erforderte dadurch das Vielfache der ursprünglich zu investierenden Arbeitsstunden. Der leichte Verzug konnte aber gegen Ende des Zeitplanes problemlos wieder aufgeholt werden. Auch das Aneignen an theoretischen Grundlagen führte zu zusätzlichem, nicht erwartetem Aufwand. Beispielsweise wurde schnell festgestellt, dass viel Wissen rund um Matrizen angeeignet werden musste, welches nicht im Vorwissen vertreten war. Vor allem das Verständnis für die Backpropagation forderte viele intensive Stunden des Erlernens.

Auch nicht alle Planungs- und Programmierarbeiten waren ohne Schwierigkeiten realisierbar. Das Planen des genetischen Algorithmus bestand aus vielen, schnell wieder verworfenen Versionen. Daher war es erst nach einem Überdenken der Reihenfolge der Unterelemente dieses Algorithmus möglich, eine brauchbare, realisierbare Vorlage zu erstellen. Das codebasierte Umsetzen der Bibliothek erreichte viele zu überwindende Hindernisse. Die selbst implementierte Matrixberechnungsunterbibliothek sorgte für Ineffizienz und bot nicht die nötige Zuverlässigkeit. So wurde schliesslich auf eine externe Mathematikbibliothek gesetzt. Auch bei den Beispielanwendungen gab es Schwierigkeiten. Das Beispiel *Car Game* brachte wegen dem in der Visualisierungsbibliothek angewandten *Garbage-Collector* zusätzliche Schwierigkeiten. Viele Stunden wurden investiert, um diese Problematik zu umgehen. Erst nach vielen verworfenen Lösungsansätzen konnte die Beispielanwendung mit leichten Einschränkungen finalisiert werden.

Das Programmieren selbst funktionierte problemlos. Auch wurden während dem Arbeitsprozess immer effizientere Methoden zur Fehlerfindung eingesetzt. Das eigene Programmierwissen war ausreichend für die Realisierung der Projekte. Dass es sich nicht um die effizienteste Art des Codes handelt, könnte bemängelt werden. Vor allem der Code der Website besteht aus vielen durch experimentierendes Ausprobieren entstandenen Codeabschnitten, die nicht immer die kompakteste und schnellste Methode zur Lösungsfindung anwenden. Beispielsweise zum Beheben, dass die Beispielanwendungen auf der Website nicht im Hintergrund weiterlaufen, wurden viele verschiedene Lösungsansätze ausprobiert, welche teilweise eine ganze Überarbeitung der Website erforderten. Alle Probleme aufzulisten wäre unnötig aufwendig, da es sich um hunderte kleine Herausforderungen handelt. Abgesehen von wenigen leicht stockenden Animationen, konnten sie behoben werden. Dass auf experimentierendes Programmieren gesetzt wurde, liegt vor allem daran, dass in eher kurzer Zeit sehr viel Verschiedenes realisiert werden musste und das Programmierwissen noch nicht auf langjähriger Erfahrung basiert. Spezifisch beim CSS-basierenden Designen der Website herrschte limitiertes Wissen über Tricks und fortgeschrittene Methoden, um selten angetroffene Probleme zu lösen. Hingegen führte speziell dies dazu, dass während dem Arbeitsprozess ein Fortschritt des Programmierwissens stattfand, da anhand von externen Quellen viel Neues gelernt wurde.

Grundsätzlich lässt sich sagen, die methodische Vorgehensweise für diese Arbeit hat sich bewährt, da das gewünschte Endprodukt in der erwarteten Zeit realisiert werden konnte. Anhand des neu erworbenen

Wissens und der durch den Arbeitsprozess neu erlernten Fähigkeiten, würde aber einiges anders umgesetzt werden bei einem erneuten Erstellen eines vergleichbaren Projektes. Dies soll aber nicht den vergangenen Arbeitsprozess kritisieren, da im Normalfall bei jeder Arbeit eine Weiterentwicklung stattfindet. Spezifisch würde sich in Zukunft beispielsweise die Zeitplanung verändern und verbessern. Durch den Arbeitsprozess wurde viel Wissen bezüglich zeitlichen Aufwandes eines Arbeitsschrittes gelernt. Dadurch kann in Zukunft gezielter ein solches Projekt geplant und realisiert werden.

8. Verzeichnisse

8.1 Abkürzungsverzeichnis

Abkürzung / Begriff	Beschreibung
BP	**B**ack**p**ropagation (siehe *3.2 Backpropagation*)
CSS	**C**ascading **S**tyle **S**heets
ES6	Steht für ECMAScript Version 6 (2015) (siehe *7.2 Glossar*)
FF	**F**eed **F**orward (siehe *3.1.3 Komplettes Netzwerk*)
FPS	**F**rames **p**er **S**econd (dt. Frames pro Sekunde)
GA	**G**enetischer **A**lgorithmus (siehe *3.3 Genetische Algorithmen*)
HTML	**H**yper**t**ext **M**arkup **L**anguage
JS	**J**ava**S**cript
KNN	**K**ünstliches **N**euronales **N**etzwerk (siehe *3.1 Neuronale Netzwerke*)
LMS	**L**east-**M**ean-**S**quares(-Algorithmus) (siehe *3.2.2 LMS-Algorithmus*)
SASS	**S**yntactically **A**wesome **S**tyle **S**heets
SCSS	**S**assy **C**ascading **S**tyle **S**heets
TLD	**T**op-**L**evel-**D**omain (letzter Domainabschnitt)

8.2 Glossar

Begriff	Beschreibung
Bibliothek	Eine Programmierbibliothek ist eine Sammlung von vorgeschriebenem Code, der in verschiedene Projekte integriert werden kann.
Chromium	Open Source-Browser entwickelt von Google (siehe https://www.chromium.org/)
Commit	Veränderung eines Projektes. Zum Beispiel durch einen Upload einer neuen Version auf GitHub.
ECMAScript 2015	Standardisierte Version von JavaScript (siehe https://262.ecma-international.org/6.0/)
Garbage-Collector	Automatische Speicherverwaltung in modernen Programmiersprachen. Eine Variable wird aus dem Arbeitsspeicher entfernt, wenn sie nicht mehr verwendet wird.
GitHub	Eine von Microsoft entwickelte Codehostingplattform (siehe https://github.com/)
Google Fonts	Schriftart- und Icons-Bibliothek entwickelt von Google (siehe https://fonts.google.com/)
Hadamard-Produkt	Spezielles Produkt zweier Matrizen gleicher Grösse. Resultierende Matrix besteht aus allen Multiplikationen zweier sich entsprechenden Einträge der Ausgangsmatrizen.

HTML-Canvas	HTML-Element, das verwendet wird, um Grafiken auf den Bildschirm des Benutzers zeichnen zu können.
Material Design	Designsystem entwickelt von Google (siehe https://m3.material.io/)
Mathjs	JavaScript Bibliothek zur Erweiterung der in JavaScript integrierten Mathematikfunktionen. (siehe https://mathjs.org/)
Node.js	Laufzeitumgebung für JavaScript (siehe https://nodejs.org/)
Product Sans	Eine von Google entwickelte Schriftart (siehe https://wikipedia.org/wiki/Product_Sans)
$sigmoid$ / $\sigma(x)$	S-förmige Funktion, die primär in neuronalen Netzwerken als Aktivierungsfunktion eingesetzt wird ($\sigma(x) = \frac{1}{1+e^{-x}}$) (siehe *3.1.2.1 Aktivierungsfunktion*)
TensorFlow	Machine-Learning-Plattform entwickelt von Google (siehe: https://www.tensorflow.org/)
ThreeJS	JavaScript Bibliothek zur Visualisierung von 2D oder 3D Grafiken (siehe https://threejs.org/)
UserAgent	Text, der von einem Webbrowser an eine Website gesendet wird. Enthält Informationen über den verwendeten Browser sowie das verwendete Betriebssystem.
WebStorm	Eine für die Webentwicklung optimierte Entwicklungsumgebung (IDE) (siehe https://www.jetbrains.com/webstorm/)
W-Vision	Webentwicklungsagentur in Sursee (siehe https://www.w-vision.ch/)
XOR	Logische Operation «exklusives ODER». Vergleichbar mit der ODER-Operation, gibt aber falsch zurück, wenn beide Inputwerte richtig sind.

8.3 Literaturverzeichnis

[1] R. Matsunaga, «https://rednuht.org/genetic_cars_2/,» 03 03 2020. [Online]. Available: https://rednuht.org/genetic_cars_2/. [Zugriff am 21 07 2023].

[2] T. Rashid, «Make your own Neural Network», Ingram International, 2017.

[3] S. N. Sivanandam und S. N. Deepa, «Introduction to Genetic Algorithms», Springer-Verlag, 2008.

[4] S. Raschka, «SebastianRaschka,» 2016. [Online]. Available: https://sebastianraschka.com/faq/docs/activation-functions.html. [Zugriff am 25 08 2023].

[5] GraphFree, «GraphFree,» 2020. [Online]. Available: https://www.graphfree.com/. [Zugriff am 12 09 2023].

[6] IBM (International Business Machines), «What are Neural Networks? | IBM,» o. D.. [Online]. Available: https://www.ibm.com/topics/neural-networks. [Zugriff am 25 08 2023].

[7] A. Rakhecha, «Medium,» 28 06 2019. [Online]. Available: https://towardsdatascience.com/https-medium-com-dashingaditya-rakhecha-understanding-learning-rate-dd5da26bb6de. [Zugriff am 21 08 2023].

[8] S. Heinz, «Statworx,» [Online]. Available: https://www.statworx.com/wp-content/uploads/learning-rate.png. [Zugriff am 11 09 2023].

[9] count-org-loc (by Ben Balter), «Workflow runs | Github,» Github, 02 10 2023. [Online]. Available: https://github.com/flug8/count-org-loc/actions/runs/6377535528/job/17306383925. [Zugriff am 06 10 2023].

[10] Wikipedia, «Flappy Bird - Wikipedia,» 21 09 2023. [Online]. Available: https://en.wikipedia.org/wiki/Flappy_Bird. [Zugriff am 23 09 2023].

[11] Content Manager DE, «Farbpsychologie Marketing - Die Wirkung von Farben im Marketing,» ContentManager, 07 09 2023. [Online]. Available: https://www.contentmanager.de/wissen/wirkung-von-farben-im-marketing-farbpsychologie/. [Zugriff am 20 09 2023].

[12] I. Duspara, «Codeimg.io,» [Online]. Available: https://codeimg.io/. [Zugriff am 15 09 2023].

[13] D. Kristanto, «ResearchGate,» April 2017. [Online]. Available: https://www.researchgate.net/figure/Back-Propagation-may-trapped-into-local-optimum_fig1_316988870. [Zugriff am 17 März 2024].

(Alle Abbildungen, die nicht ins Literaturverzeichnis verlinkt sind, wurden selbstständig erstellt. Alle Codebeispiele wurden mit dem Tool «Codeimg» [12] erstellt.)

9. Danksagung

Damit meine Arbeit auf diese Weise fertig gestellt werden konnte, waren zahlreiche Helfer notwendig, die mich im richtigen Moment mit passenden Lösungsvorschlägen unterstützten. Ich möchte mich darum bei allen beteiligten Personen recht herzlich bedanken.

Besonders danken möchte ich meinem Betreuer Philipp Hurni für die hervorragende Betreuung und Unterstützung bei der Umsetzung der gesamten Arbeit. Er unterstützte mich bei jedem meiner Probleme und gab zu jeder Frage seine ehrliche Meinung.

Auch bei meinem Informatiklehrer Andreas Gut möchte ich mich bedanken, da er mich beim Themenentscheid der Arbeit in Voraus unterstützte und mir für die Arbeit nützliche Lernmaterialien zur Verfügung stellte.

Dem GitHub-Supportteam sowie den Kontaktpersonen von JetBrains gilt mein Dank. Sie haben mir auf Anfrage für die Arbeit nützliche, eigentlich kostenpflichtige Produkte kostenlos zur Verfügung gestellt.

Allen Mitarbeitern von W-Vision, besonders Adrian Hess und Aaron Gerig, die mir während meines Praktikums einen ausgezeichneten Einblick in die Webentwicklung gegeben haben und somit die Eingrenzung des Themenbereiches der Arbeit stark beeinflussten.

Auch möchte ich Rafael Matsunaga, Donovan Harshbarger und Igor Duspara danken, da sie für sich selbst profitlose Onlinetools entwickelt haben, die dieser Arbeit einen grossen Mehrwert boten.

Ein grosses Dankeschön geht an meine Eltern, die mir während der ganzen Arbeit zur Seite standen und mich bei allfälligen Problemen unterstützten. Sie lasen die Arbeit mehrmals gegen und gaben mir konstruktive Korrekturvorschläge. Dank ihnen konnte die Arbeit noch weiter optimiert werden.

10. Anhang

Das Projekt ist einsehbar unter der *GitHub*-Organisation SimpleJS-AI (https://github.com/SimpleJS-AI). Der ganze erstellte Code ist dort öffentlich verfügbar. Gleichzeitig ist eine lokale Version des Codes auf einem USB-Stick am Ende der Arbeit hinterlegt. Der folgende Verzeichnisbaum zeigt die Struktur des Projektes auf. Beim USB-Stick sind die Unterprojekte als Ordner mit den dazugehörenden Namen sortiert. Auf der *GitHub*-Organisationsseite sind sie als einzelne Repositories hinterlegt. Der Titel eines Repositories entspricht dem Namen des Unterprojektes. Die Dokumentation für die Bibliothek kann in der Organisation nur über das *SimpleJS-lib* Unterprojekt oder über den Direktlink https://github.com/SimpleJS-AI/SimpleJS-lib/wiki aufgerufen werden. Beim USB-Stick befindet sie sich im Unterordner *SimpleJS-lib.wiki*.

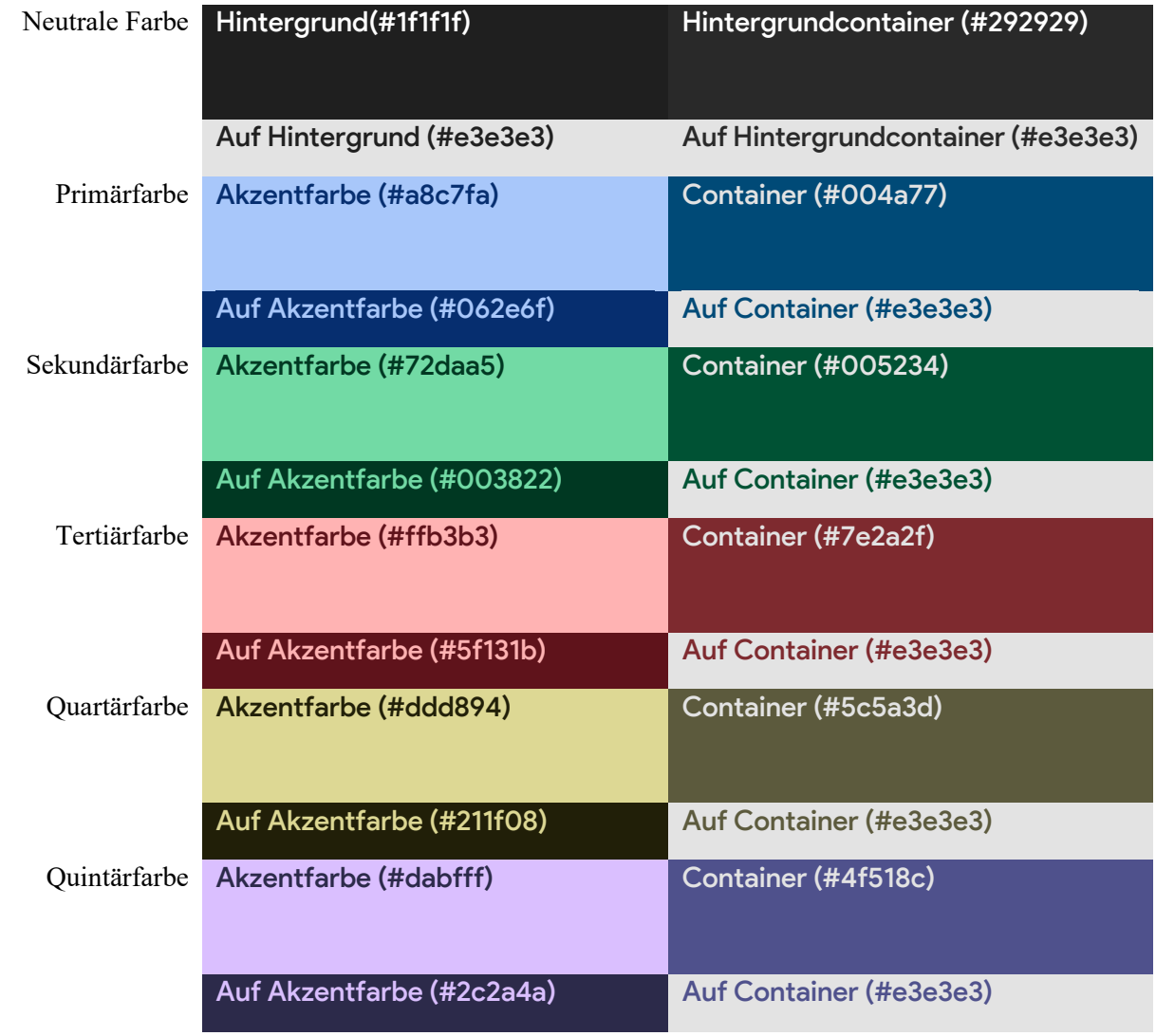
10.1 Verzeichnisbaum

```
SimpleJS-AI
├── SimpleJS/                          Website
│   ├── index.html                     (https://github.com/SimpleJS-AI/SimpleJS)
│   ├── script.js
│   ├── style.scss
│   └── style.css
├── SimpleJS-lib/                      Bibliothek
│   └── lib/                           (https://github.com/SimpleJS-AI/SimpleJS-lib)
│       ├── simple.js
│       ├── simple.min.js
│       └── simple-old.js
├── SimpleJS-lib.wiki/                 Dokumentation
│   ├── Home.md                        (https://github.com/SimpleJS-AI/SimpleJS-lib/wiki)
│   ├── 1.-Installation.md
│   ├── 2.-Usage.md
│   └── 3.-Examples.md
├── SimpleJS-XOR/                      XOR Beispiel
│   ├── index.html                     (https://github.com/SimpleJS-AI/SimpleJS-XOR)
│   └── script.js
├── SimpleJS-BPExample/                Backpropagation Beispiel
│   ├── index.html                     (https://github.com/SimpleJS-AI/SimpleJS-BPExample)
│   ├── script.js
│   ├── style.scss
│   └── style.css
├── SimpleJS-FlappyBird/               Flappy Bird Beispiel
│   ├── index.html                     (https://github.com/SimpleJS-AI/SimpleJS-FlappyBird)
│   ├── script.js
│   ├── style.scss
│   └── style.css
├── SimpleJS-CarGame/                  Car Game Beispiel
│   ├── index.html                     (https://github.com/SimpleJS-AI/SimpleJS-CarGame)
│   ├── script.js
│   ├── three.js
│   ├── style.scss
│   └── style.css
├── SimpleJS-LogoNodes/                Logo-Visualisierung durch Nodes (Element der Website)
│   ├── index.html                     (https://github.com/SimpleJS-AI/SimpleJS-LogoNodes)
│   └── script.js
└── SimpleJS-NNVisualization/          Neuronales Netzwerk-Visualisierung (Element der Website)
    ├── index.html                     (https://github.com/SimpleJS-AI/SimpleJS-NNVisualization)
    ├── script.js
    ├── style.scss
    └── style.css
```

(Es werden nur relevante Dateien erwähnt, nicht alle.)

10.2 Farbdesign Website

Abb. 51 Farbdesign Website



(Die Farben sind optimiert für eine digitale Visualisierung)

10.3 Auswertungsdaten

Tabelle 1 Auswertungen Flappy Bird Beispiel

	Anzahl Generationen	Zeitaufwand [Frames]	G1	G2	G3	G4	G5	G6	G7	G8	G9	G10
D1	9	121684	330	339	370	2330	39109	15109	5828	58269	100000	
D2	2	2617	2617	100000								
D3	4	1059	330	330	399	100000						
D4	4	2267	330	338	1599	100000						
D5	4	1049	298	369	382	100000						
D6	4	88415	2216	730	85469	100000						
D7	10	106517	346	338	4674	379	360	1997	2330	12667	83426	100000
D8	4	79394	334	330	78730	100000						
D9	8	2440	353	337	330	350	357	362	351	100000		
D10	7	64097	330	349	3007	11100	43443	5868	100000			

Abb. 52 Verlust Template 3 (siehe Tabelle 4)

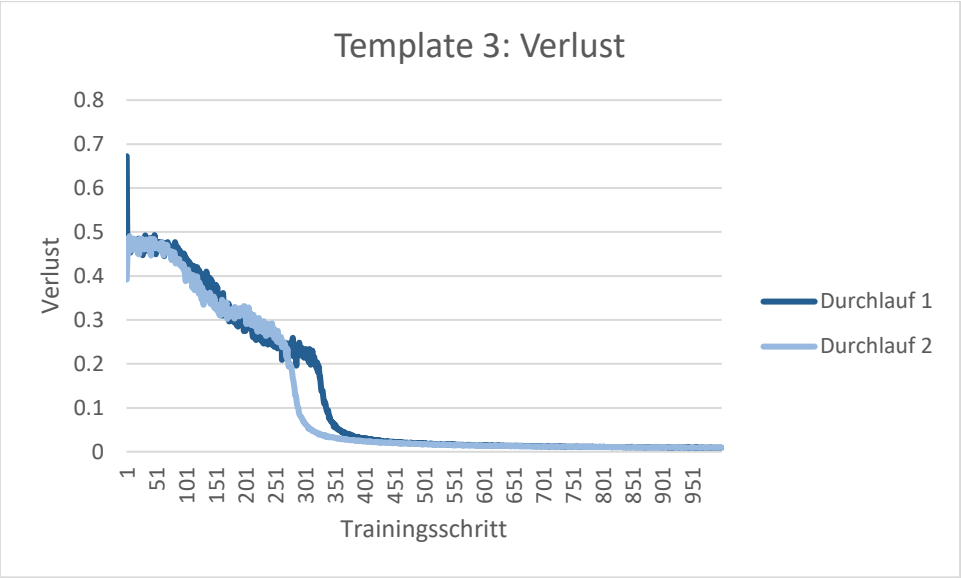


Abb. 53 Verlust Template 4 (siehe Tabelle 5)

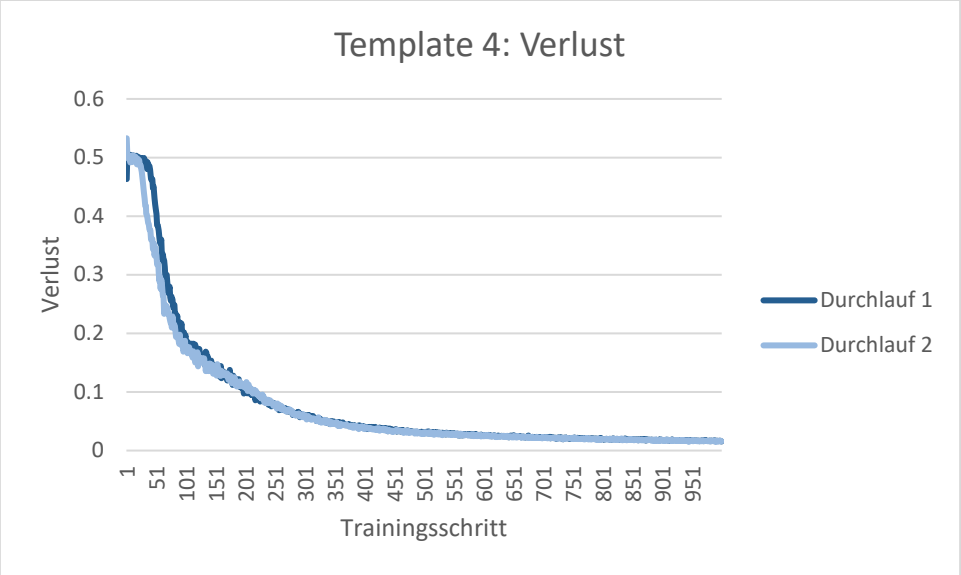


Tabelle 2 Auswertung Verlust Template 1 BPExample

Durchlauf 1	Durchlauf 2	Differenz
0.56239188	0.53410495	0.02828694
0.30175643	0.32602939	0.02427296
0.28803912	0.28655047	0.00148865
0.33440827	0.31755849	0.01684978
0.29352498	0.29242732	0.00109766
0.20502884	0.15435559	0.05067325
0.09167056	0.09107498	0.00059558
0.06119049	0.06062941	0.00056108
0.04901357	0.05132555	0.00231198
0.03989565	0.0394212	0.00047445
0.03316921	0.0323144	0.00085482
0.02767845	0.0312879	0.00360946
0.02642703	0.03397065	0.00754363
0.0220026	0.02761076	0.00560817
0.02018909	0.0245523	0.00436321
0.0200312	0.02110228	0.00107108
0.01757812	0.01928016	0.00170204
0.01881386	0.02012229	0.00130843
0.01564395	0.01725824	0.00161429
0.01737418	0.01754993	0.00017576
0.01514881	0.01556733	0.00041852
0.01500069	0.01400331	0.00099738
0.01324097	0.01492955	0.00168858
0.01176606	0.01351459	0.00174853
0.01133814	0.01669203	0.00535389
0.01243189	0.01404184	0.00160996
0.01199732	0.01161246	0.00038485
0.01117718	0.01308461	0.00190743
0.01287073	0.01153369	0.00133703
0.01083158	0.01142144	0.00058985
0.01182239	0.01139901	0.00042338
0.00929677	0.01093935	0.00164258
0.00990801	0.01152491	0.0016169
0.01005552	0.01026426	0.00020874
0.00869397	0.00989081	0.00119684
0.00882254	0.01107372	0.00225117
0.00963081	0.01002491	0.0003941
0.00876501	0.00849794	0.00026707
0.00863048	0.0094721	0.00084161
0.00839318	0.00850535	0.00011217
0.00823195	0.01138518	0.00315323
0.00782925	0.01036044	0.00253119
0.00810125	0.00741777	0.00068349
0.0074474	0.00863346	0.00118606
0.00821147	0.00933747	0.001126
0.006738	0.00886194	0.00212394
0.00732202	0.00796959	0.00064757
0.00725449	0.0092315	0.00197701
0.00757599	0.0078754	0.0002994
0.00701312	0.00699557	1.7554E-05

Tabelle 3 Auswertung Verlust Template 2 BPExample

Durchlauf 4	Durchlauf 9	Differenz
0.45911772	0.49642006	0.03730234
0.48545208	0.49567499	0.01022292
0.49021018	0.49239038	0.0021802
0.49420495	0.49337174	0.00083321
0.49957862	0.4907603	0.00881832
0.49655893	0.48860262	0.00795631
0.48661818	0.48472882	0.00188936
0.49088973	0.49116061	0.00027089
0.49294416	0.48423696	0.0087072
0.49393085	0.49997736	0.00604651
0.49055537	0.49456573	0.00401036
0.49435782	0.49625382	0.001896
0.4877994	0.49385625	0.00605686
0.48733371	0.49772806	0.01039435
0.47483001	0.49077762	0.01594762
0.42185126	0.48405206	0.06220081
0.28134802	0.48566287	0.20431484
0.21892534	0.47979135	0.260866
0.16723934	0.42669302	0.25945367
0.09176812	0.38560612	0.293838
0.06497556	0.35646634	0.29149078
0.06116643	0.29955381	0.23838738
0.05214619	0.22738	0.17523381
0.03935396	0.18263955	0.14328559
0.03841697	0.11954297	0.081126
0.03516179	0.07371438	0.03855259
0.03034619	0.06082967	0.03048348
0.0301381	0.0528049	0.0226668
0.02857315	0.04409763	0.01552448
0.02419308	0.04110361	0.01691053
0.02200658	0.0347926	0.01278601
0.02269904	0.02994541	0.00724637
0.02275187	0.03166297	0.0089111
0.0206963	0.02749498	0.00679868
0.0212024	0.02515807	0.00395567
0.01885476	0.02496795	0.00611319
0.01937881	0.02324112	0.00386231
0.01882312	0.02101142	0.00218831
0.01680352	0.02119537	0.00439185
0.01785455	0.01875662	0.00090207
0.01665283	0.0188687	0.00221588
0.01747444	0.01914576	0.00167131
0.01444612	0.01614814	0.00170202
0.0151086	0.01717515	0.00206655
0.01363148	0.01709796	0.00346648
0.01468793	0.01676165	0.00207372
0.01363421	0.01710515	0.00347094
0.0136687	0.01713581	0.00346711
0.01290202	0.01815155	0.00524954
0.01339454	0.01544825	0.00205372

Tabelle 4 Auswertung Verlust Template 3 BPExample

Durchlauf 1	Durchlauf 2	Differenz
0.6724007	0.391406	0.2809947
0.48467063	0.46456469	0.02010594
0.48147083	0.45077887	0.03069197
0.46090989	0.4581594	0.00275048
0.46531845	0.45379603	0.01152241
0.44036237	0.39486767	0.0454947
0.41481508	0.3804715	0.03434358
0.36498697	0.361518	0.00346896
0.33067556	0.34562831	0.01495275
0.29465568	0.32864367	0.03398798
0.27569634	0.3045835	0.02888716
0.26705127	0.27277982	0.00572854
0.27163419	0.2833977	0.01176351
0.25352508	0.25049434	0.00303074
0.22145839	0.16063915	0.06081924
0.22898181	0.06398597	0.16499584
0.20328085	0.04279495	0.1604859
0.07646492	0.03410142	0.04236351
0.04574752	0.02914403	0.01660349
0.0369328	0.02693005	0.01000275
0.02854007	0.02356583	0.00497424
0.02522921	0.02270784	0.00252137
0.02273071	0.02085743	0.00187327
0.0214382	0.01889421	0.00254399
0.01937293	0.01839638	0.00097655
0.01889679	0.01717955	0.00171724
0.01671258	0.01663101	8.1572E-05
0.01626128	0.01592225	0.00033903
0.0153794	0.014915	0.00046441
0.01508656	0.01483307	0.00025349
0.01484959	0.01382196	0.00102764
0.01350675	0.01341176	9.4995E-05
0.0135506	0.01327743	0.00027317
0.01295692	0.01236736	0.00058956
0.01281081	0.01260592	0.00020489
0.01260466	0.01202398	0.00058067
0.01220056	0.01186755	0.00033301
0.01154887	0.0114166	0.00013227
0.01127113	0.01118536	8.5776E-05
0.01126424	0.01124148	2.2758E-05
0.01117598	0.01067439	0.00050159
0.01091008	0.01041044	0.00049964
0.01040239	0.01033525	6.7144E-05
0.01013003	0.01013418	4.1502E-06
0.01006516	0.0099211	0.00014406
0.00955086	0.0099118	0.00036094
0.00987868	0.00966117	0.00021752
0.00977583	0.00938996	0.00038587
0.00922944	0.00939141	0.00016197
0.00985616	0.00934569	0.00051047

Tabelle 5 Auswertung Verlust Template 4 BPExample

Durchlauf 1	Durchlauf 2	Differenz
0.46311657	0.5328817	0.06976513
0.50111209	0.49261003	0.00850206
0.47437612	0.37595693	0.09841919
0.32421418	0.27019657	0.05401761
0.24831522	0.21459812	0.0337171
0.17417998	0.17731515	0.00313518
0.15079018	0.14359349	0.00719668
0.15012521	0.14578554	0.00433967
0.13679354	0.12845942	0.00833412
0.11959488	0.12107304	0.00147816
0.10982927	0.10948258	0.00034669
0.09483977	0.09767281	0.00283303
0.08244294	0.08486611	0.00242316
0.06944865	0.07082581	0.00137716
0.06523202	0.06225455	0.00297747
0.0598388	0.0587606	0.0010782
0.05415372	0.05346065	0.00069307
0.04842633	0.04815789	0.00026844
0.04560166	0.04506536	0.00053629
0.04075433	0.03893198	0.00182235
0.03953515	0.03738961	0.00214554
0.03882881	0.03773451	0.00109429
0.0338235	0.03511099	0.0012875
0.03220164	0.03254112	0.00033947
0.03180335	0.0324818	0.00067846
0.03007893	0.02884524	0.00123369
0.02845713	0.02919036	0.00073323
0.02792693	0.02736801	0.00055892
0.02567525	0.02822733	0.00255209
0.02597264	0.02541824	0.0005544
0.02457426	0.02470408	0.00012983
0.02402837	0.02294782	0.00108055
0.02419262	0.02420852	1.5893E-05
0.02440586	0.02377637	0.00062948
0.02301953	0.02269485	0.00032468
0.0218906	0.02272607	0.00083548
0.02096226	0.02054821	0.00041406
0.02137651	0.02134169	3.4823E-05
0.02037631	0.02030424	7.2066E-05
0.02052013	0.01903358	0.00148654
0.01929943	0.01890202	0.00039741
0.01962156	0.01832181	0.00129975
0.01937873	0.01732068	0.00205804
0.01792738	0.01745857	0.00046881
0.0184976	0.01765646	0.00084114
0.01790314	0.01794396	4.0824E-05
0.01775961	0.01668979	0.00106981
0.01787758	0.01703799	0.00083958
0.0166687	0.01695344	0.00028474
0.01683129	0.01589085	0.00094044

Tabelle 6 Auswertung Lernraten BPExample (Ausgedünnt, nur jeder zwanzigste Wert ist enthalten)

0.0001	0.001	0.01	0.03	0.1	0.3	1	3	10
0.27254076	0.39644517	0.91317319	0.56281368	0.48161809	0.84779686	0.44423741	0.32396376	0.16006852
0.31924875	0.35952533	0.32090228	0.32761173	0.32081998	0.31741255	0.26788486	0.32249283	0.2433088
0.32973567	0.31822538	0.31860326	0.30299936	0.30377941	0.29817699	0.30849315	0.05536323	0.11841355
0.29944086	0.3364474	0.33367926	0.30592677	0.3018655	0.30965249	0.06434463	0.014147	0.20591
0.31375388	0.31574676	0.33263018	0.321205	0.31775867	0.30603908	0.03682196	0.00920249	0.18301363
0.31808308	0.31015168	0.30998322	0.31595935	0.31921141	0.31883787	0.02544297	0.00858895	0.19690094
0.30588024	0.30917293	0.31855908	0.33764852	0.32576027	0.25026897	0.01822639	0.00752512	0.19124871
0.32510662	0.3106533	0.31184734	0.32411312	0.28131705	0.17817569	0.0177376	0.00684697	0.17787041
0.31240881	0.31694233	0.30216845	0.35543777	0.30619448	0.09582258	0.01449516	0.0063138	0.18522182
0.30421608	0.32521309	0.33635908	0.34659549	0.29214163	0.06201118	0.01589475	0.00526688	0.19169649
0.30544881	0.30383139	0.30379897	0.33815684	0.2878376	0.04837868	0.01145012	0.00530457	0.19244516
0.31845548	0.30532178	0.33906959	0.29760086	0.30167323	0.04164317	0.00953238	0.00535605	0.18060058
0.31145816	0.30604997	0.31777165	0.28539562	0.30716267	0.03374686	0.01058601	0.00475666	0.20007557
0.31071739	0.31362406	0.29979958	0.31997304	0.2587002	0.03133887	0.0099339	0.00456195	0.18927193
0.31170304	0.30615221	0.31021649	0.30466178	0.19009217	0.02855983	0.00922396	0.00428843	0.17884372
0.31062871	0.31402134	0.31591113	0.29434944	0.1510018	0.02540438	0.00809422	0.00404149	0.19927164
0.322871	0.30981812	0.30716671	0.32963472	0.11508344	0.0267285	0.00881456	0.00406527	0.16468305
0.31860366	0.29755168	0.32045195	0.31721006	0.08675304	0.02104316	0.00768421	0.00342191	0.19594029
0.31054698	0.31764959	0.30948665	0.3060724	0.07403079	0.01926741	0.00779072	0.00395846	0.18414089
0.30917851	0.30250466	0.31329124	0.32071089	0.06969243	0.01844227	0.00721334	0.00365688	0.17905331
0.30545739	0.31713426	0.31853255	0.32946864	0.05906481	0.01902755	0.00665816	0.00338266	0.19757701
0.30981348	0.3213704	0.31558079	0.31415412	0.05520758	0.01795851	0.00723712	0.00393806	0.21115742
0.31737549	0.32477412	0.31837453	0.30374564	0.04436161	0.0178298	0.0064224	0.00322328	0.20928862
0.3099981	0.29666463	0.30698998	0.31365922	0.04895533	0.01622093	0.00675082	0.00377407	0.19809431
0.31219115	0.31870137	0.33063711	0.29812584	0.04507633	0.01770919	0.00681972	0.00306261	0.23524474
0.32733042	0.30517232	0.30747167	0.30162867	0.03799158	0.01656535	0.0057277	0.00316192	0.16130034
0.31433656	0.32672077	0.31524977	0.33150779	0.03858422	0.01416851	0.00603735	0.00339622	0.1820418
0.30917957	0.30501217	0.28526299	0.34557504	0.03526558	0.01276746	0.00588579	0.00302959	0.19596415
0.32193069	0.31508964	0.30394122	0.29363296	0.03145486	0.0145819	0.00602048	0.00288365	0.20038515
0.33283849	0.32588467	0.29623234	0.30664675	0.03261934	0.01337076	0.00595503	0.0027061	0.17748051
0.32269232	0.31392797	0.32609699	0.32608148	0.03366143	0.01395358	0.00516112	0.0030526	0.20722497
0.34450615	0.31208247	0.32680746	0.31803817	0.02816897	0.01151234	0.00567945	0.00264118	0.19440624
0.29674231	0.32091703	0.32832466	0.30778761	0.02908303	0.01091921	0.00476028	0.00277908	0.20784054
0.31492316	0.33025952	0.31905575	0.31783304	0.0237409	0.01131449	0.00477299	0.00214011	0.16952106
0.31727547	0.3141273	0.28488983	0.34194946	0.02937989	0.01243308	0.0052872	0.00248968	0.17828334
0.31161515	0.32279816	0.3172346	0.2892637	0.02737005	0.01171755	0.00429444	0.00282142	0.17934854
0.30273238	0.30893	0.32949678	0.33961698	0.0266889	0.00962835	0.00519282	0.00251899	0.20307977
0.32146611	0.32842564	0.31710445	0.32513066	0.02629801	0.01097022	0.00480031	0.00257703	0.19255393
0.32388418	0.28824198	0.3031489	0.29963634	0.02213178	0.01072989	0.00475446	0.00276083	0.18827087
0.30746998	0.30193653	0.33118772	0.30207411	0.02379352	0.01127129	0.00477849	0.00228265	0.23651886
0.30337669	0.3164887	0.30533764	0.28110133	0.02448721	0.01182185	0.00509714	0.00223292	0.21345173
0.31184335	0.31999705	0.30950771	0.31464059	0.01867331	0.01016727	0.00429142	0.00248564	0.18622053
0.31880017	0.33149086	0.31851946	0.27796639	0.02047828	0.01079629	0.00468035	0.002072	0.19370802
0.31325541	0.31515664	0.3043589	0.27629917	0.02057725	0.00941201	0.00515267	0.00248272	0.20866167
0.30935221	0.31285399	0.31124372	0.29540901	0.02073362	0.00963607	0.00414499	0.00228566	0.18250535
0.31139595	0.3183264	0.33606376	0.28705534	0.01840799	0.00873817	0.00410669	0.00202163	0.17742543
0.3143743	0.32440831	0.31804956	0.26863321	0.01743357	0.00937337	0.00477192	0.00210416	0.2111865
0.29549369	0.33187954	0.32703209	0.24968624	0.02023959	0.00800837	0.00456496	0.00214017	0.1945999
0.31156392	0.32295164	0.3265004	0.25851202	0.019916	0.00948116	0.00362883	0.00211691	0.18202263
0.31148749	0.31586939	0.3184134	0.25378173	0.01742325	0.0083166	0.00401561	0.00229792	0.15884171

Tabelle 7 Auswertung Populationsgrössen Flappy Bird (Ausgedünnt, nur drei von zehn Messwerte sind enthalten)

RAW DATA			CALCULATED DATA							
PopSize	Avg. FPS	Gen's		Gmin	Gmax	Avg. G	Avg. FPS	T.Zeit	FPS + Limit	T.Z. + Limit
5	66336	721	5	77	721	388	62977	61.6097941	120	32333.3333
5	50495	77								
5	72100	366								
10	60015	174	10	81	501	252	52334.6667	48.1516394	120	21000
10	52445	81								
10	44544	501								
15	42491	31	15	31	141	79	51379	15.3759318	120	6583.33333
15	56435	65								
15	55211	141								
20	34582	14	20	13	32	19.6666667	34544.6667	5.6931123	120	1638.88889
20	34610	13								
20	34442	32								
50	10794	12	50	11	13	12	12104	9.91407799	120	1000
50	12785	13								
50	12733	11								
100	652	11	100	4	11	7	584.333333	119.794638	120	583.333333
100	512	4								
100	589	6								
1000	61	3	1000	2	3	2.33333333	60	388.888889	60	388.888889
1000	62	2								
1000	57	2								
10000	6	1	10000	1	2	1.33333333	6.33333333	2105.26316	6.33333333	2105.26316
10000	6	1								
10000	7	2								

Tabelle 8 FPS der Beispielanwendungen beim Gebrauch des genetischen Algorithmus

Generation	FPS Flappy Bird	FPS Car Game
1	62.2	7.8
2	65	6.2
3	61.7	4.2
4	67.1	4.1
5	59.3	2.8
6	58.9	1.7
7	62.3	1.5
8	62.2	1.2
9	62.9	1.2
10	59.9	0.9
11	60.4	0.4
12	61.4	0.3
13	60.9	0.3
14	59.9	0.1
15	60	0.1

Tabelle 9 Optimierung eines vortrainierten Objektes durch genetischen Algorithmus

	Nur BP	G2	G20	G200
Beispiel 1	8.415	8.349	8.149	7.998
Beispiel 2	12.772	12.773	11.198	10.155
Beispiel 3	9.535	9.248	9.068	6.119

Tabelle 10 Erfolgsrate bei manipulierten Umweltparametern

PipesGap	Erfolgsrate	PipeGap	Erfolgsrate	PipeWidth	Erfolgsrate
100	0	0	0	0	100
105	0	5	0	5	100
110	0	10	0	10	100
115	0	15	0	15	100
120	0	20	0	20	100
125	0	25	0	25	100
130	0	30	0	30	100
135	2	35	0	35	100
140	6	40	0	40	100
145	12	45	0	45	100
150	15	50	0	50	100
155	16	55	0	55	97
160	16	60	0	60	96
165	41	65	15	65	96
170	61	70	17	70	82
175	82	75	23	75	79
180	85	80	45	80	69
185	89	85	67	85	57
190	99	90	94	90	53
195	100	95	99	95	53
200	100	100	100	100	52
205	100	105	100	105	52
210	100	110	100	110	52
215	100	115	100	115	52
220	100	120	100	120	52
225	100	125	100	125	52
230	100	130	100	130	51
235	100	135	100	135	51
240	100	140	100	140	49
245	100	145	100	145	49
250	99	150	100	150	49
255	99	155	100	155	49
260	99	160	100	160	49
265	99	165	100	165	49
270	99	170	100	170	49
275	99	175	100	175	49
280	99	180	100	180	49
285	99	185	100	185	47
290	99	190	100	190	44
295	99	195	100	195	44
300	99	200	100	200	44

11. Deklaration

„Ich erkläre hiermit,

- dass ich die vorliegende Arbeit selbständig und nur unter Benutzung der angegebenen Quellen verfasst habe,
- dass ich auf eine eventuelle Mithilfe Dritter in der Arbeit ausdrücklich hinweise,
- dass ich eine allfällige Nutzung von Künstlicher Intelligenz (z.B. ChatGPT) in der Reflexion der Arbeit ausgewiesen und diskutiert habe,
- dass ich vorgängig die Schulleitung und die betreuende Lehrperson informiere, wenn ich diese Maturaarbeit, bzw. Teile oder Zusammenfassungen davon veröffentlichen werde, oder Kopien dieser Arbeit zur weiteren Verbreitung an Dritte aushändigen werde.“

Ort:

Datum:

Unterschrift: