Genetische Algorithmen und Backpropagation für Neuronale Netzwerke im Vergleich am Beispiel des Spiels 2048

Bachelorarbeit

Victor Emanuel Dietenberger
383315

1. Juni 2025

Betreuer: Prof. Dr. Benjamin Blankertz
Dr. Stefan Fricke

Technische Universität Berlin
Fakultät IV – Elektrotechnik und Informatik
Institut für Softwaretechnik und Theoretische Informatik
Fachgebiet Neurotechnologie

Kurzfassung

In dieser Arbeit wurden Trainingsmethoden für Neuronale Netzwerke verglichen und am Beispiel des Spiels 2048 getestet.

Zum einen wurden Genetische Algorithmen getestet. Diese erstellen eine Vielzahl an zufällig initialisierten Netzwerken, testen diese und erstellen aus den besten eine neue Generation, wobei jeder Nachkommen leicht verändert wird. Wie in der echten Evolution, welche dieses Verfahren abbildet, sollten sich nach einigen Wiederholungen die besten Netzwerke und nützliche Mutationen durchsetzen und sich in der Population ausbreiten.

Verglichen wurde das mit einem Backpropagation-Algorithmus, bei dem direkt geschaut wird, wie man die Parameter des Neuronalen Netzwerkes verändern muss, um ein gewünschtes Ergebnis zu erzielen. Es wurden Bewertungen von Spielfeldern durch Expectimax, der einige Züge in die Zukunft schaut und das beste erreichbare Spielfeld aussucht, gelernt. Mit diesem Netzwerk als Bewertungsfunktion kann Expectimax nun nochmals angewendet werden, um neue Trainingsdaten zu erzeugen, um auf diesen Daten das Netzwerk erneut zu trainieren. Dieser Vorgang kann beliebig oft wiederholt werden, wodurch man die Suchtiefe erweitert, ohne dabei den gesamten Suchbaum zu durchlaufen, da dies vom Netzwerk imitiert wurde.

Verglichen wurden der durchschnittliche Score, die beste erreichte Kachel und die Trainings- sowie Spielzeit der Algorithmen.

Der Backpropagation-Algorithmus hat deutlich besser abgeschnitten. Zeit war in dieser Arbeit der limitierende Faktor, da jede Iteration mehrere Tage gebraucht hat, während der Genetische Algorithmus vergleichsweise schnell war, aber weitaus schlechtere Ergebnisse erzielt hat.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

1.1 Neuronale Netzwerke

Neuronale Netzwerke sind in den letzten Jahren durch generative Modelle wie ChatGPT berühmt geworden. Jedoch reicht ihr Nutzen weit über Text- oder Bildgeneration hinaus. Sie können dabei helfen, riesige Datenmengen auszuwerten und zu interpretieren und klassische Algorithmen zu verbessern. Sie können allgemeines Wissen aus Daten extrahieren, was zum Beispiel in der Bilderkennung, in der Medizin oder Robotik angewendet werden kann.

1.2 Spieltheorie

Die Spieltheorie [6] ist die Wissenschaft strategische Entscheidungen in einer klar definierten Umgebung zu treffen. Sie findet Anwendung in Feldern wie Politik, Ökonomie oder sogar in persönlichen Lebensentscheidungen. Sie eignet sich gut, um Neuronale Netzwerke und klassische Algorithmen zu testen, da Probleme durch eindeutig definierte Regeln und Aktionen oft weniger komplex zu simulieren und zu lösen sind und sich das Resultat durch das Spielergebnis leicht beurteilen lässt. Gleichzeitig lassen sich Algorithmen der Spieltheorie oft auf reale Probleme übertragen.

1.3 Ziel dieser Arbeit

Das Ziel dieser Arbeit war es, das Spiel 2048 mit verschiedenen Algorithmen zu lösen und diese zu vergleichen. Der Fokus lag dabei darauf Neuronale Netzwerke zu trainieren einen höchstmöglichen score zu erreichen. Hierbei wurden Genetische Algorithmen verwendet, um durch wiederholtes Spielen von verschiedenen Neuronalen Netzwerken und dem Auswählen der Besten, mit der Zeit immer bessere Ergebnisse zu erreichen. Als zweiten Ansatz wurde Backpropagation benutzt, um die Bewertungsfunktion eines Expectimax-Algorithmus von einem Neuronalen Netzwerk zu lernen.

1.4 Value Netzwerk vs Policy Netzwerk

Man kann Netzwerke, die Spiele spielen, in zwei Gruppen einteilen: Value Netzwerke und Policy Netzwerke. [20] Das durch Genetische Algorithmen trainierte Netzwerk ist ein Policy Netzwerk, welches als Output die Wahrscheinlichkeit für jeden Spielzug errechnet, also das Spiel direkt spielt. Der Backpropagation-Ansatz ist ein Value Netzwerk, welches keinen Spielzug errechnet, sondern Spielfelder bewertet. Mit dieser Bewertung kann man dann mit einem klassischen Suchbaum Algorithmus

den Spielzug finden, der im besten Spielfeld resultiert.

1.5 Aufbau dieser Arbeit

Zuerst wird das Spiel und bereits vorhandene Literatur zum Thema vorgestellt, dann Neuronale Netzwerke eingeführt und anschließend die beiden Trainingsmethoden genauer erklärt. Hierbei wurde versucht, die vorgestellten Methoden von Grund auf und leicht verständlich darzustellen.

Abschließend werden erreichte Ergebnisse präsentiert und sowohl miteinander als auch mit Ergebnissen aus der Literatur verglichen.

1.6 Das Spiel 2048

2048 ist ein stochastisches, Einspieler-Spiel mit perfekter Information. Das heißt, dass Zustandsübergänge nach jedem Spielzug nicht deterministisch, sondern teilweise zufällig generiert sind und Spielerinnen und Spieler zu jedem Zeitpunkt jegliche Information des Spiels zur Verfügung haben.

Es wurde 2014 vom damals 19-jährigen Italiener Gabriele Cirulli für Webbrowser und Mobilgeräte programmiert und ließ sich stark durch ähnliche Spiele inspirieren.[1]

2048 wird auf einem 4x4-Feld gespielt. Jedes Feld kann mit einer Kachel belegt sein, die mit einer 2er-Potenz beschriftet ist.

Zu Beginn des Spiels werden zwei Kacheln an zufälligen Positionen, und nach jedem Spielzug eine weitere Kachel an einer freien, zufälligen Position generiert. Eine neu-generierte Kachel hat mit 90% Wahrscheinlichkeit eine 2, und mit 10% Wahrscheinlichkeit eine 4.

Jeder Spielzug beinhaltet eine von vier Richtungen (oben, unten, links, rechts). Alle Kacheln werden in die ausgewählte Richtung geschoben, bis sie auf den Rand des Spielfelds oder auf eine andere Kachel stoßen, wobei Kacheln mit derselben Zahl verschmelzen und sich addieren, und somit die nächsthöhere Zweierpotenz bilden.

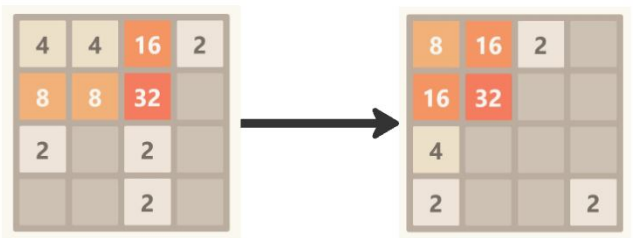

Abbildung 1.1: Beispiel-Spielzug: links [4]

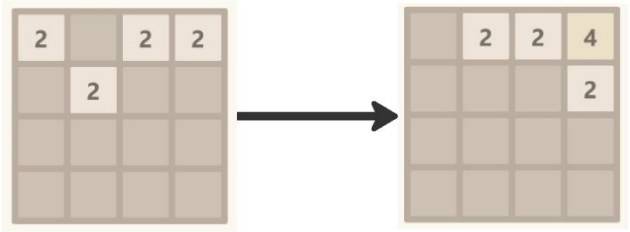

Abbildung 1.2: Beispiel-Spielzug: rechts [4]

Ist kein möglicher Spielzug vorhanden, ist das Spiel verloren. Ziel des Spiels ist es, die namensgebende Kachel 2048 zu erreichen. Zusätzlich dazu gibt es einen score. Immer wenn zwei Kacheln verschmolzen werden, wird die neue Zahl auf den score addiert. Dieser eignet sich ideal für einen Messwert, KI-Algorithmen zu bewerten. Die höchste erreichte Kachel kann auch für eine Bewertung genommen werden, allerdings bietet der score eine präzisere Bewertung.

1.7 Literatur

1.7.1 2048

- **Temporal Difference Learning of N-Tuple Networks for the Game 2048** [28]: Training von N-Tuple Netzwerken durch Anwendung von Temporal Difference Learning, einer Methode, die ohne Suchbäume oder menschliches Wissen auskommt, in der Agenten durchs spielen und belohnt werden lernen. Vergleich drei verschiedener Varianten, von der die beste über 97% der Spiele gewonnen hat.
- **Developing a 2048 Player with Backward Temporal Coherence Learning and Restart** [19]: Verbesserung von Temporal Difference Learning und Temporal Coherence Learning durch umdrehen der Lernrichtung und durch Neustart von der Spielmitte Verbesserung des Lernens der hinteren Spielzüge. Hierbei wurde mit einem durchschnittlichen Score von 438.515 das beste Ergebnis erzielt, welches in der Recherche gefunden wurde.
- **Developing Value Networks for Game 2048 with Reinforcement Learning** [20]: Training eines Convolutional Neural Networks durch Reinforcement Learning, um Bewertungen des Spielfeldes zu geben und diese Bewertungen durch Expectimax zu durchsuchen. Auch diese Arbeit kam mit einer 100% Gewinnrate und einem durchschnittlichen Score von 386.973 zu einem sehr guten Ergebnis.
- **Playing Game 2048 with Deep Convolutional Neural Networks Trained by Supervised Learning** [14]: Basierend auf den Spieldaten der vorherigen Arbeit, wurde ein Convolutional Neural Network trainiert, 2048 zu spielen. Es wurden zwei und fünf Convolutional Layer gegenübergestellt und das Netzwerk mit fünf Layern kam zu einem besseren Ergebnis mit einem durchschnittlichem Score von 93.830.
- **An Early Attempt at Applying Deep Reinforcement Learning to the Game 2048** [11]: Training eines Convolutional Neural Networks durch Reinforcement Learning 2048 direkt zu spie-

len. Dabei wurden Temporal Difference Learning und Q-Learning als Trainingmethoden benutzt und moderate Ergebnisse erzielt.

- **Evolving Neural Network to Play Game 2048** [8]: Es wurde ein Genetischer Algorithmus angewendet um ein Neuronales Netzwerk zu trainieren 2048 zu spielen, dies führte zu mäßigen Ergebnissen. Der höchste erreichte durchschnittliche Score aus mehreren Experimenten mit verschiedenen Parametern war 2.500.

- **Minimax and Expectimax Algorithm to Solve 2048** [29]: Vergleich Minimax und Expectimax, zwei Suchbaumalgorithmen, um 2048 zu lösen. Minimax erreichte durch Alpha-Beta pruning, einer Technik nicht den gesamten Suchbaum zu durchlaufen eine schnellere Laufzeit, die in Expectimax nicht einsetzbar ist. Dieser erreichte allerdings ein besseres Ergebnis mit einem durchschnittlichen Score von 39.163.

- **Pedagogical Possibilities for the 2048 Puzzle Game** [23]: Es wurde Expectimax mit einfachem Greedy Play, sprich das Spielen nach einer Bewertungsfunktion und einem Monte-Carlo Algorithmus, welcher für alle Spielzüge zufällig weiter spielt und somit eine Einschätzung der Spielzüge liefert, verglichen. Außerdem wurde der Einfluss von Chance-Sampling auf Expectimax in Suchtiefe 2 und 3 verglichen. Das beste Ergebnis lieferte Expectimax mit Suchtiefe 3 ohne Chance-Sampling mit einem durchschnittlichen Score von 25.270. Chance-Sampling hatte in dieser Suchtiefe nur einen kleinen Nachteil mit einem durchschnittlichen Score von 24.883.

1.7.2 Neuronale Netzwerke

- **Artificial Neural Networks** [7]: Allgemeiner Überblick über Neuronale Netzwerke

- **Backpropagation and the brain** [17]: Einführung des Backpropagation Algorithmus als Trainingsmethode für Neuronale Netzwerke

- **A brief introduction to supervised, unsupervised, and reinforcement learning** [22]: Überblick über verschiedene Lernmethoden für Neuronale Netzwerke

1.7.3 Genetische Algorithmen

- **Evolutionary Algorithms and Neural Networks** [21]: In diesem Buch werden im entsprechenden Kapitel Genetische Algorithmen allgemein eingeführt

- **Neural Networks Optimization through Genetic Algorithm Searches: A Review** [10]: Beschreibung wie man durch Genetische Algorithmen Neuronale Netzwerke trainieren und verbessern kann.

- **Comparative evaluation of genetic algorithm and backpropagation for training neural networks** [26] und **Comparing backpropagation with a genetic algorithm for neural network training** [12]: Vergleich Backpropagation und Genetische Algorithmen als Trainingsmethode für Neuronale Netzwerke

- **Tuning of the Structure and Parameters of a Neural Network Using an Improved Genetic Algorithm** [16]: Ein verbesserter Genetischer Algorithmus wurde vorgeschlagen als Trainingsmethode und Strukturfindung für Neuronale Netzwerke

1.7.4 Suchbäume

- **Artificial intelligence: a modern approach** [25] und **Rediscovering*-minimax search** [13]: Vorstellung von Suchbaumalgorithmen wie Minimax und Expectimax

2 Methoden

2.1 Neuronale Netzwerke

Künstliche Neuronale Netzwerke sind Algorithmen, die Funktionen approximieren können und auf biologischen Neuronalen Netzwerken im menschlichen Gehirn basieren.

2.1.1 Neuron

Ein einzelner Neuron hat beliebig viele Inputs, die mit einem weight multipliziert werden. Dieser weight repräsentiert die Stärke des Inputs, wieviel Einfluss jeder Input auf den Output hat. Die gewichteten Inputs werden aufsummiert und an eine activation function gegeben. Die activation function ist eine nicht-lineare Funktion, die verhindert, dass das Netzwerk ein simples, lineares Input-, Output mapping ist und das approximieren von komplexen, nicht-linearen Funktionen ermöglicht.

Der Output eines Neuron kann also durch folgende Funktion beschrieben werden:

$$O = f(\sum_{j=1}^{n} w_j x_j)$$

wobei w_j der Vektor ist, der die weights beinhaltet, x_j der Vektor, der die Inputs des Neurons beinhaltet und f die activation function.[7]

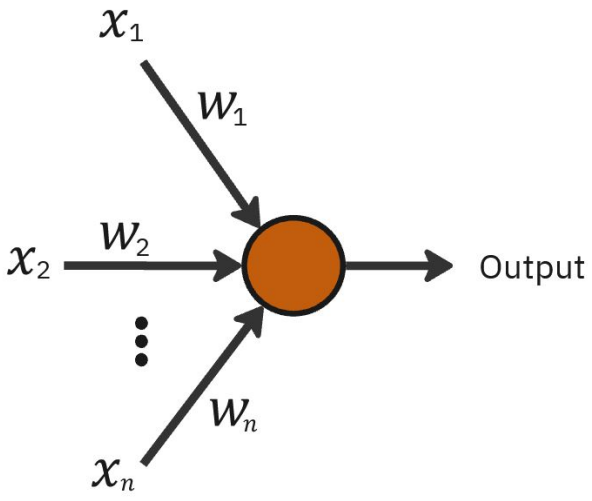

Abbildung 2.1: Beispielabbildung eines Neurons [5]

Abschließend wird noch ein sogenannter Bias hinzugefügt, ein weiterer Parameter des Neurons, der die Funktion nach links oder rechts schieben kann. Ohne den Bias wäre es schwierig Funktionen zu approximieren, die nicht durch den Nullpunkt verlaufen.

$$O = f(b + \sum_{j=1}^{n} w_j x_j)$$

2.1.2 Fully Connected Layer

Ein Neuronales Netzwerk besteht aus Layern von Neuronen. Einem Input Layer, einem Output Layer und beliebig vielen sogenannten Hidden Layern. Jedes Layer besteht aus einer Anzahl an Neuronen, wobei jeder Neuron mit jedem Neuron des nächsten Layers verbunden ist. Das heißt der Input jedes Neurons sind die aufsummierten, gewichteten Outputs jedes Neurons des vorhergehenden Layers. Ein solches Layer ist fully connected und ein Netzwerk bestehend aus solchen Layern nennt man Fully Connected Neural Network.

Das Input Layer liest und kodiert gegebene Daten. Zum Beispiel könnte von einem Bild jeder Pixel als Grauwert eingelesen werden. Diese Daten werden dann, wie oben beschrieben Layer für Layer durch das Netzwerk gespeist bis sie am Ende beim Output Layer ankommen. Die Neuronen des Output Layers können dann ausgelesen und interpretiert werden und zum Beispiel eine Klassifikation oder Aktion erzeugt werden.

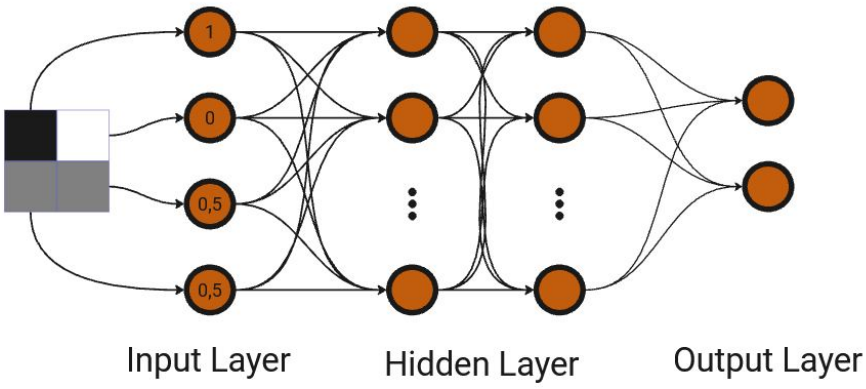

Abbildung 2.2: Beispielabbildung eines Neuronalen Netzwerkes [5]

2.1.3 Convolutional Layer

Convolutional Neural Networks sind besonders gut darin Muster in Daten zu erkennen, und werden daher oft für Bilderkennung eingesetzt [24]. Diese Mustererkennung passiert in Convolutional Layern, wo ein Filter (Kernel) über die Daten gelegt wird, um zu erkennen, wie stark die Daten dem Kernel ähneln. So können in frühen Layern beispielsweise Ecken, Kanten oder Kreise und in späte-

ren Layern mehr abstrakte Konzepte erkannt werden.

Convolutional Layer bieten sich auch für 2048 an, da so die Nachbarn von Feldern beurteilt werden können.

Der Kernel ist eine Matrix aus Weights. Um einen Output zu generieren, nimmt man sich einen gleichgroßen Abschnitt des Inputs, multipliziert beide Matrizen Elementweise und addiert das Ergebnis. Anschließend verschiebt man den Kernel um den nächsten Output zu generieren.

Folgende Parameter definieren ein Convolutional Layer:

- **Kernel Size**: Größe des Filters
- **Stride**: Wert, um den der Kernel für jeden Output verschoben wird
- **Zero-Padding**: Reihen und Spalten mit Nullen um die Inputmatrix, um die Größe der Outputmatrix zu bestimmen.

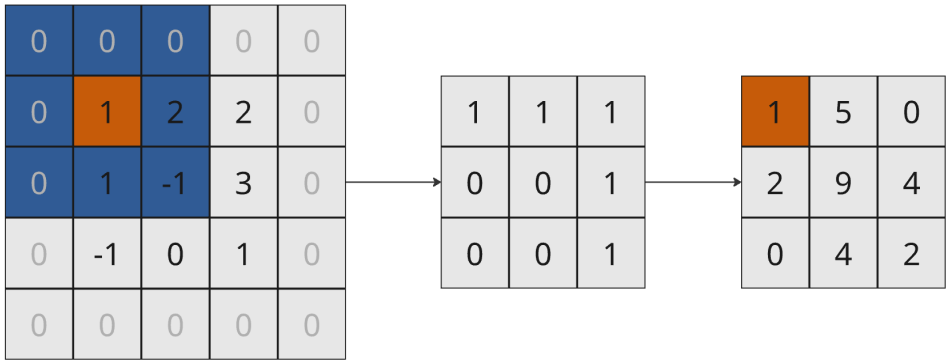

Abbildung 2.3: Convolution Beispiel mit 3x3 Input, 3x3 Kernel, Stride = 1 und Zero-Padding = 1. Kernel erkennt Ecken im Bild. [5] Gekennzeichneter Output: $(0 * 1) + (0 * 1) + (0 * 1) + (0 * 0) + (1 * 0) + (2 * 1) + (0 * 0) + (1 * 0) + (-1 * 1) = 1$

2.1.4 Backpropagation

Ein Neuronales Netzwerk versucht also die Input- Outputrelation komplexer Funktion durch das Anpassen der weights zu approximieren. Um die richtigen weights zu setzen, muss das Netzwerk trainiert werden. Eine Möglichkeit hierfür ist die Backpropagation.

Hierbei wird ein Trainingsdatenset genommen, welches Inputdaten, und die jeweiligen gewünschten Ergebnisse enthält. Für jeden Eintrag im Datenset generiert man einen Output und vergleicht diesen mit den gewünschten Outputs. Dieser Unterschied ist der Error des Netzwerkes. Eine oft verwendete Error Funktion ist der Squared Error:

$$E = \frac{1}{2} \sum_{l} (y_l - t_l)^2$$

wobei y_l der Ouput des Netzwerkes und t_l der gewünschte Output ist. [17]

Nun wird beim Output-Layer begonnen, das generierte Ergebnis mit dem gewünschten verglichen und versucht die weights des Output-Layers als auch die Outputs des vorherigen Layers so zu verändern, dass das generierte Ergebnis näher am gewünschten ist. Die weights des Output Layers, die mit Neuronen verbunden sind, die eine höhere Aktivierung (Output) haben, werden stärker verändert, da diese auch die Aktivierung des Output Neurons stärker beeinflussen. Um die Aktivierungen des vorherigen Layers zu verändern, wendet man dieses Verfahren einfach auf die Neuronen dieses Layers an und arbeitet sich somit von hinten nach vorne durch das Netzwerk.

Da man keinen direkten Einfluss auf die Aktivierung von Neuronen, sondern lediglich auf die Weights hat, berechnet man für jeden Weight den Gradienten des Errors bezüglich des Weights und ändert den Weight proportional zum negativen Gradienten.

Das Update eines nicht-ouput Layer Weights sieht also folgendermaßen aus: [17]

$$\Delta W_{ij} = -\eta \frac{\delta E}{\delta W_{ij}}$$

W_{ij} ist das Weight das Neuron i und j verbindet.

η ist die learning rate und bestimmt wie groß die Veränderungen der Weights sind. Ist die learning rate zu groß, kann das gewünschte Minimum der error Funktion nur schwer genau getroffen werden. Ist sie zu klein dauert es zu lange, um das Minimum zu erreichen. Eine häufig angewendete Technik ist es, mit einer großen learning rate zu starten, um sich schnell dem Minimum anzunähern und dann über Zeit beim Trainingsprozess zu verringern, um es genau zu treffen.

$\frac{\delta E}{\delta W_{ij}}$ ist der Gradient des errors bezüglich des Weights.

2.1.5 Backpropagation Trainingsprozess

Während des Trainings durch Backpropagation werden die Trainingsdaten in Batches aufgeteilt. Dies hat den Vorteil, dass der gesamte Batch in den Arbeitsspeicher passt und somit effizienter verarbeitet werden kann. [2] Jeder Batch wird einzeln mit Backpropagation durch das Netzwerk gespeist. Jeder Durchgang, so alle Daten einmal zu verarbeiten wird Epoch genannt.

Da komplexe Funktionen nicht nach einer Backpropagation gelernt werden, wird der Trainingsprozess oft für viele Epochs wiederholt.

2.1.6 Overfitting

Falls das Netzwerk zu groß ist, oder zu wenig Trainingsdaten zu Verfügung stehen, besteht eine Gefahr auf Overfitting, das heißt, dass das Netzwerk die Trainingsdaten auswendig lernt und nicht auf neue Daten generalisiert.

Um Overfitting zu überprüfen, wird das Datenset in Trainingsdaten und Testdaten vor dem Training aufgeteilt. Auf den Trainingsdaten wird das Netzwerk trainiert. Anschließend werden Outputs auf den Testdaten generiert und der Error berechnet. Da Trainingsdaten und Testdaten der gleichen Funktion entspringen, sollte falls kein Overfitting besteht, der Error genauso klein sein, wie auf den Testdaten beim Abschluss des Trainings.

Falls Overfitting wegen zu wenig zur Verfügung stehenden Daten besteht, können durch Modifikation (drehen oder spiegeln) der bestehenden, neue Daten erstellt werden. Das wird Data Augmentation genannt.

2.2 Genetische Algorithmen

Genetische Algorithmen basieren auf dem Konzept der Evolution aus der Biologie. In der Natur haben Lebewesen, die bessere Gene besitzen eine höhere Überlebenschance und somit mehr Chancen Nachkommen zu zeugen. Diese Nachkommen erben die guten Gene und haben somit ebenfalls gute Überlebenschancen. Beim Kopieren der Gene können Fehler auftreten, die man Mutation nennt. Diese Fehler können sich positiv oder negativ auf die Überlebenschancen auswirken. Über einen Zeitraum von vielen Generationen führt das dazu, dass eine Spezies sich immer besser an ihre Umgebung anpasst, da immer die besten Gene ausgewählt werden um eine neue Generation zu bilden.

In der Programmierung kann man diesen Prozess simulieren, um Algorithmen oder Daten zu optimieren, ohne dabei Eigenwissen über das Modell haben zu müssen. Folgende Begriffe werden für einen Genetischen Algorithmus definiert: [10] [16]

2.2.1 Chromosom

Eine Sequenz an Daten, die eine mögliche Lösung für ein Problem beschreibt. Auf Chromosomen werden genetische Operatoren angewendet und somit die nächste Generation gebildet.

2.2.2 Population

Eine Menge an Chromosomen bilden eine Population. Eine initiale Population wird zufällig erstellt, wobei die Größe vom Problem abhängt.

Die Population P ist beschrieben durch: $P = \{\mathbf{p_1}, \mathbf{p_2}, ..., \mathbf{p_{pop\_size}}\}$, während pop_size die Populationsgröße ist.

2.2.3 Fitness

Jedes Chromosom einer Generation wird als Lösung eines Problems genommen und bewertet. Diese Bewertung ist die Fitness, auf deren Basis Eltern für die nächste Generation ausgewählt werden.

$$\text{fitness} = f(\mathbf{p_i})$$

2.2.4 Selektion

Nachdem jedem $\mathbf{p_i}$ aus der Population ein Fitness Wert zugewiesen wurde, wird eine neue Generation gebildet. Hierfür wird jedem $\mathbf{p_i}$ eine Chance q_i zugewiesen, einen Nachkommen für die nächste Generation zu zeugen. Diese Chance ist proportional zum Fitness Wert. $\mathbf{p_i}$ mit hoher Fitness, haben eine höhere Chance selektiert zu werden.

$$q_i = \frac{f(\mathbf{p_i})}{\sum_{k=1}^{pop\_size} f(\mathbf{p_k})}$$

Auf den ausgewählten Nachkommen werden die genetischen Operatoren Crossover und Mutation angewendet.

2.2.5 Crossover

Mehrere Chromosomen werden verbunden, um einen Nachkommen zu erzeugen. Hierfür werden mehrere Eltern aus der Population ausgewählt. Häufig wird ein zufälliger Punkt im Chromosom ausgewählt. Alle Daten im Chromosom vor dem gewählten Punkt werden von einem Elternteil genommen, alle Daten dahinter von einem anderen Elternteil. Der Grund hierfür ist, Chromosomen zu durchmischen aber gute Sequenzen innerhalb dieser nicht auseinanderzureißen. Dieses Verfahren nennt man Ein-Punkt-Crossover.

Es existieren ebenfalls andere Arten des Crossovers: Zwei-Punkt, Mehrfach-Punkt, gleichmäßig oder Drei-Eltern, unter anderem.

2.2.6 Mutation

Da eine global optimale Lösung wahrscheinlich nicht in der initialen Population enthalten ist, werden Chromosomen beim Bilden einer neuen Generation zufällig verändert. Hier kann man zum Beispiel jeden Datenpunkt im Chromosom mit einer kleinen Wahrscheinlichkeit durch einen zufälligen Wert austauschen.

2.3 Verbesserter Genetischer Algorithmus[16]

Der folgende Abschnitt wurde komplett aus dieser [16] Arbeit entnommen.

Genetische Operatoren kann man verbessern, indem man für jeden Nachkommen mehrere Kandidaten durch leicht unterschiedliche Regeln erstellt, für alle die Fitness evaluiert und dann den besten als Nachkommen auswählt.

2.3.1 Crossover

Für das verbesserte Crossover werden 4 Kandidaten erstellt.

$$os_1 = \frac{\mathbf{p}_1 + \mathbf{p}_2}{2} \tag{2.3.1}$$

$$os_2 = \mathbf{p}_{\max}(1 - w) + \max(\mathbf{p}_1, \mathbf{p}_2)w \tag{2.3.2}$$

$$os_3 = \mathbf{p}_{\min}(1 - w) + \min(\mathbf{p}_1, \mathbf{p}_2)w \tag{2.3.3}$$

$$os_4 = \frac{(\mathbf{p}_{\max} + \mathbf{p}_{\min})(1 - w) + (\mathbf{p}_1 + \mathbf{p}_2)w}{2} \tag{2.3.4}$$

- $\mathbf{p}_1$ und $\mathbf{p}_2$ sind Vektoren, die alle Chromosomen beider ausgewählten Eltern enthalten. os_1 nimmt also für jedes Chromosom den Durchschnitt beider Eltern.
- $\mathbf{p}_{\max}$ sind die maximalen Chromosomen der gesamten Population und $\max(\mathbf{p}_1, \mathbf{p}_2)$ sind die maximalen Chromosomen beider Eltern. w ist ein vom Benutzer bestimmter Wert, der die Gewichtung dieser beiden bestimmt. os_2 ist also generell ein Nachkommen mit höheren Chromosomen.
- os_3 ist analog dazu ein Nachkommen mit niedrigeren Chromosomen. $\mathbf{p}_{\min}$ sucht die niedrigsten Chromosomen der Population und $\min(\mathbf{p}_1, \mathbf{p}_2)$ die niedrigsten Chromosomen der Eltern.
- os_4 sucht die durchschnittlichen Chromosomen der Population und die durchschnittlichen Chromosomen der Eltern und gewichtet diese ebenfalls mit dem Parameter w.

2.3.2 Mutation

Für die verbesserte Mutation werden drei potenzielle Nachkommen erstellt. Für diese Nachkommen werden nach verschiedenen Regeln manche Chromosomen randomisiert.

- os_1: Ein zufällig ausgewählter Wert im Chromosom wird randomisiert
- os_2: Beliebig viele zufällig ausgewählte Werte im Chromosom werden randomisiert
- os_3: Alle Werte im Chromosomen werden randomisiert

Von diesen Nachkommen wird wieder die Fitness berechnet und die schlechteren zwei verworfen.

Da für diese verbesserten genetischen Operatoren jeweils mehrere Kandidaten für Nachkommen erstellt werden, die getestet werden müssen (und Testen in Falle dieser Arbeit heißt, immer mindestens ein ganzes Spiel zu spielen) wirken sich diese Verbesserung stark negativ auf die Performance aus.

2.4 Genetische Algorithmen für Neuronale Netzwerke

Backpropagation als Trainingsmethode für Neuronale Netzwerke konvergieren oft zu einem lokalen Minimum der Error Funktion. [12] Um zuverlässiger das globale Minimum zu finden, bieten sich einige Alternativen an, zum Beispiel Genetische Algorithmen. Durch sie kann ein Netzwerk trainiert werden, indem man die Weights des Netzwerks als Chromosomen nimmt. [10]

Man erstellt eine Population an Netzwerken mit zufällig gewählten weights. Anschließend testen man jedes Netzwerk mit einer Fitness Funktion, welche man maximieren will. Unter Verwendung der oben beschriebenen genetischen Operatoren, erstellt man nun eine neue Generation, während man Netzwerke mit besserer Fitness auswählt.

Die neue Generation besteht also aus Netzwerken mit besserer Fitness und Mutationen, die sich positiv auf Fitness auswirken und haben eine höhere Chance als Nachkommen für weitere Generationen ausgewählt zu werden. So steigt Fitness über Zeit.

2.4.1 Input Kodierung

Damit das Netzwerk besser zwischen gleichen und ungleichen Kacheln unterscheiden kann, wurde im Falle dieser Arbeit sich für eine binäre Input kodierung entschieden. Es wurde, ähnlich wie im Abschnitt 2.6.2 beschrieben, jede Kachel in eine 16-stellige Binärform umgewandelt, wobei die erste Stelle eine leere Kachel repräsentiert und alle weiteren Stellen die Zweierpotenzen 2^1 bis 2^{15} darstellen. Diese 16 16-stelligen Binärzahlen wurden hintereinander als Input in das Neuronale Netzwerk gegeben.

2.5 Expectimax

Expectimax ist ein Suchbaumalgorithmus für stochastische Spiele, der mehrere Züge in die Zukunft blickt, um den bestmöglichen Zug zu errechnen. Der Suchbaum besteht abwechselnd aus Zufalls-, und Maximierungsknoten. Ein Maximierungsknoten hat für jeden möglichen Spielzug eine Verzweigung und errechnet das Maximum von allen seinen Kindsknoten, wählt also den besten folgenden Spielzug aus.

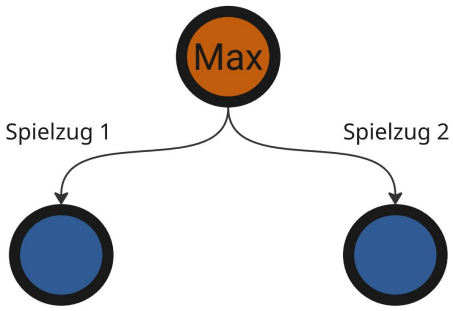

Abbildung 2.4: Expectimax Maximierungsknoten mit zwei möglichen Spielzügen [5]

Anschließend passiert ein zufallbasiertes Event, welches Einfluss auf das Spielfeld hat. Jedes mögliche Resultat des Events bekommt eine Abzweigung vom Zufallsknoten. Von diesem wird das durchschnittliche Resultat berechnet. Danach wird wieder ein Spielzug berechnet und so verzweigt sich der Baum immer weiter, bis zu einer bestimmten Suchtiefe.

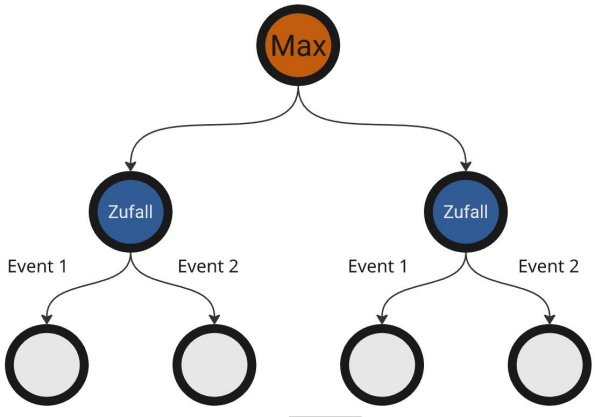

Abbildung 2.5: Expectimax Maximierungs- und Zufallsknoten mit zwei Spielzügen und zwei möglichen Zufallsevents [5]

Ist der Baum bei der gegebenen Suchtiefe angekommen, wird für jeden untersten Knoten des Baumes, der Zustand des Spielfelds durch eine Bewertungsfunktion errechnet. Hierfür werden Heuristiken, die den resultierten Spielzustand abschätzen benutzt.

Diese Bewertung des Zustands wird jetzt zurück durch den Baum gespeist, indem bei Zufallsknoten immer der Durchschnitt aller Zufallsevents berechnet wird, also die Bewertungen der Resultate des Events, multipliziert mit der Wahrscheinlichkeit des Events aufsummiert und durch die Anzahl der Events geteilt. Bei Maximierungsknoten wird der beste Spielzug ausgewählt und die Bewertung des Spielzuges weitergeleitet. Beim ersten Knoten angelangt, hat man also für jeden möglichen Spielzug eine Bewertung des Spielzustandes, der im Durchschnitt erreicht wird, wenn $\frac{n}{2}$ Züge weiter gespielt wird, während n die Suchtiefe des Algorithmus ist. Nun sucht man sich den Spielzug mit bester Bewertung aus, spielt diesen und durchsucht den Suchbaum erneut.

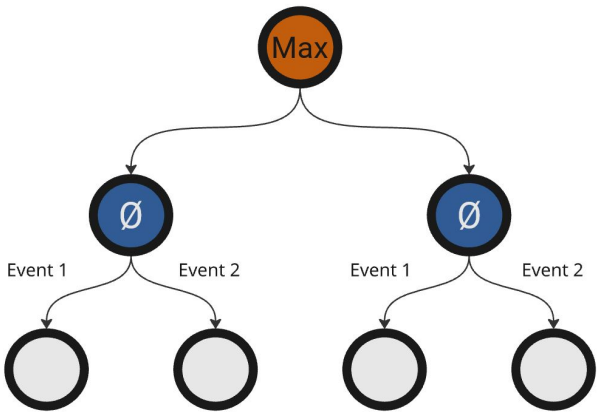

Abbildung 2.6: Expectimax Beispiel mit Suchtiefe 2 [5]

Zusammengefasst schaut Expectimax also einige Züge in die Zukunft, wobei durchschnittliche gegnerische Züge und bestmögliche eigene Züge betrachtet werden. So bekommt man bei einer hohen Suchtiefe entsprechend gute Anweisungen für den nächsten Spielzug.

Algorithm 1 Expectimax[29]

```
1:  function EXPECTIMAX(node, depth, turn)
2:      if depth = 0 or node is a terminal node then
3:          return the heuristic value of node
4:      end if
5:      if turn == player1 then
6:          bestValue ← −∞
7:          for all child of node do
8:              val ← EXPECTIMAX(child, depth - 1, player2)
9:              bestValue ← MAX(bestValue, val)
10:         end for
11:         return bestValue
12:     else
13:         expectedValue ← 0
14:         for all child of node do
15:             val ← EXPECTIMAX(child, depth - 1, player1)
16:             expectedValue += Probability[child]∗val;
17:         end for
18:         return expectedValue
19:     end if
20: end function
```

Im Spiel 2048 existieren vier mögliche Spielzüge, jeder Maximierungsknoten verzweigt also viermal. Anschließend wird eine zufällige Kachel generiert. Diese wird auf einem leeren Feld platziert und hat mit einer 90% Wahrscheinlichkeit eine Zwei und ansonsten eine Vier. Es gibt 16 verschiedene

Felder und zwei verschiedene Möglichkeiten für Kacheln. Jeder Zufallsknoten verzweigt den Baum also im schlechtesten Fall $16 * 2 = 32$ mal. Dies führt schon bei niedrigen Suchtiefen schnell zu einer schlechten Performance. Bei einer Suchtiefe von 10 gibt es schon $1.1805916e{+}21$ Verzweigungen und selbst bei einer moderaten Tiefe von 6 gibt es $4.3980465e + 12$ Verzweigungen. Durch Anpassungen ist es allerdings möglich diese Performance zu verbessern.

2.5.1 Transposition Table [9]

Transposition Tables basieren auf dem Konzept, Teilbäume zwischenzuspeichern, um doppelte Berechnungen zu vermeiden. Führen mehrere verschiedene Abfolgen von Spielzügen zum gleichen Spielfeld, muss man den Nachfolgenden Suchbaum nur einmal durchsuchen. Man speichert also, sobald Expectimax einen Zug für einen Maximierungsknoten errechnet hat, das Ergebnis und die Suchtiefe, die nachfolgend durchlaufen wurde ab. Erreicht man diesen Spielfeldzustand nochmals in einem anderen Teilbaum, mit einer noch ausstehenden Suchtiefe, die höchstens so hoch ist wie die gespeicherte, kann man das Ergebnis einfach übernehmen, und muss den Baum nicht noch einmal durchlaufen, da man nicht tiefer sucht und somit auch kein besseres Ergebnis finden kann.

Dies führt zu einer erheblichen Erhöhung der Geschwindigkeit bei hohen Suchtiefen, und auch bei der niedrigen Suchtiefe 4 ist eine spürbare Erhöhung zu messen.

2.5.2 Chance Sampling[23]

Um den Suchbaum noch weiter zu verkleinern, kann man Chance Sampling einsetzen. Man schaut sich für den Durchschnitt der Zufallsknoten nicht alle leeren Kacheln an, sondern lediglich eine festgelegte Anzahl zufälliger Kacheln pro Zufallsknoten. Hierbei muss man beachten, nicht dieselben Kacheln doppelt zu nehmen. Die Wahrscheinlichkeit eine Kachel mit einem bestimmten Wert (Zwei oder Vier) auszuwählen ist proportional zur Wahrscheinlichkeit, dass diese im Spiel erscheint (90% für Zwei und 10% für Vier) und zur Anzahl der Kacheln die schon als sample ausgewählt wurden.

Die relative Wahrscheinlichkeit r_2 eine Kachel mit Zwei auszuwählen ist also:

$$r_2 = \frac{(u_2 * 0.9)}{(u_2 * 0.9 + u_4 * 0.1)} \tag{2.5.1}$$

wobei u_2 die Anzahl der leeren Zweierkacheln, und u_4 die Anzahl der leeren Viererkacheln sind, die noch nicht als sample ausgewählt wurden sind.

2.5.3 Heuristik

Um Spielfelder zu bewerten können verschiedene Heuristiken genutzt werden, die eine Beurteilung des aktuellen Spielstandes vornehmen und Strategien mit einfließen lassen.

Folgende Heuristik beinhaltet drei unterschiedlich gewichtete Strategien:[29]

A: Leere Kacheln

Viele leere Kacheln zu haben, belohnt das zusammenfügen von Kacheln und bildet den Hauptteil der Heuristik.

B: Glattheit

Es ist vorteilhaft, möglichst ähnliche Kacheln nebeneinander zu haben. Je weiter gleiche Kacheln voneinander entfernt sind, desto schwieriger ist es, sie zusammenzufügen. Die Heuristik summiert alle Differenzen zwischen allen möglichen nebeneinanderliegenden Paaren von Kacheln auf.

C: Große Kacheln am Rand

Große Kacheln in der Spielfeldmitte blockieren das zusammenschieben von kleineren Kacheln. Deswegen werden Kacheln in der Mitte des Spielfelds bestraft, in Relation zum Wert der Kachel.

Die Formale Definition der Heuristik H ist folgende:

$$H = A * E - B * D - C * P \tag{2.5.2}$$

E ist die Anzahl der leeren Kacheln, D ist die Summe der nebeneinander liegenden Paare und P ist die Summe der in der Mitte des Spielfelds liegenden Kacheln. A, B und C sind konstante und definieren die Skalierungen der einzelnen Komponenten.

Für A wurde ein sehr großer Wert genommen (4096), um die anderen beiden Komponenten (jeweils 10) zu dominieren.

2.6 Bewertungsfunktion Backpropagation

Dieser Abschnitt beschäftigt sich damit, ein Value Netzwerk zu trainieren. Das heißt, das trainierte Neuronale Netzwerk spielt nicht selbst, sondern bewertet Spielfelder. Durch Backpropagation wurden hier die Resultate von Expectimax mit moderater Suchtiefe gelernt. Mit diesem Netzwerk als Bewertungsfunktion kann man erneut Expectimax mit moderater Suchtiefe spielen lassen. Da das Netzwerk Expectimax simmuliert, kann dies als Verlängerung der Suchtiefe gesehen werden, ohne dabei den kompletten Suchbaum zu durchlaufen. Diese Spiele kann man dann erneut als Trainingsdaten nehmen um das Netzwerk erneut zu trainieren und das Prinzip beliebig oft wiederholen. Dieses Verfahren wird im weiteren Verlauf der Arbeit kurz Bewertungsfunktion Backpropagation genannt.

Das Konzept ähnelt Stockfish NNUE[18], welches Schachpositionen durch ein Neuronales Netzwerk bewertet und anschließend durch traditionelle Suchalgorithmen Züge findet.

2.6.1 Verfahren

Zuerst werden mit einer handgemachten Bewertungsfunktion mit Expectimax Trainingsdaten erstellt.

Die Outputs der in dieser Arbeit verwendeten Bewertungsfunktion bilden auf einen sehr großen Bereich ab. Da dies das Lernen erschwert, wird im weiteren Training der Ouput dieser durch 1.000 geteilt.

Vor dem normalen Training wird die Bewertungsfunktion an sich vom Neuronalen Netzwerk gelernt, welche für das Netzwerk leicht zu lernen ist. Diese fungiert als Grundlage für das weitere Training und erleichtert das Erlernen der folgenden, komplexeren Funktionen. Hierfür wird für das Spielfeld jedes Spielzuges aus den Trainingsdaten der Output der Bewertungsfunktion generiert.

Da das Generieren von Daten sehr lange braucht, werden durch Data Augmentation die Testdaten vervierfacht, indem das Spielfeld jeweils um 90°, 180° und 270° gedreht wird. Die Bewertungsfunktion besteht aus drei Teilen, der Anzahl leerer Kacheln, der Summe der benachbarten Kacheln und der Kacheln am Rand des Spielfeldes. Durch das Drehen des Spielfeldes verändern sich weder die Anzahl leerer Kacheln noch Nachbarschaftsverhältnisse der Kacheln, oder die Summe der Kacheln am Rand. Daher verändert sich auch die Bewertungsfunktion beim Drehen nicht und der Output kann für alle Ausrichtungen vom Originaldatenpunkt übernommen werden. Auf diesen Daten wird das Netzwerk trainiert.

Anschließend wird das Netzwerk auf den gleichen Inputdaten aber diesmal auf den Bewertungen von Expectimax trainiert. Das Modell kann also nach dem Training die Bewertung eines Spielfeldes von Expectimax mit Suchtiefe 4 approximieren, ohne den Suchbaum zu durchlaufen und kann so als Erweiterung der Suchtiefe fungieren.

Daraufhin werden neue Trainingsdaten erstellt, indem Expectimax nun mit dem trainierten Netzwerk als Bewertungsfunktion weitere Spiele spielt. Diesen Vorgang kann man nun beliebig oft wiederholen. Expectimax schaut immer ein paar Züge in die Zukunft und das Netzwerk versucht dieses vorrausschauen zu lernen und an die nächste Iteration Expectimax zu übergeben. Die Voraussicht sollte so über Zeit immer größer und somit das Ergebnis immer besser werden.

2.6.2 Input Kodierung

Eine Input Kodierung wurde aus dieser [11] Arbeit übernommen. Das Spielfeld wird in 16 4x4 Channels unterteilt, das heißt der Input des Netzwerkes ist eine 16x4x4 Matrix, wobei der erste Channel leere Kacheln und die weiteren Channel Kacheln mit Zweierpotenzen von 2^1 bis 2^{15} repräsentieren. Für jede Kachel im Spielfeld wird im entsprechenden Channel bei der Position der Kachel eine 1 als Input geschrieben, für alle Felder im Channel, die keine Kachel der entsprechenden Zweierpotenz an dieser Position haben, eine 0.

Diese binäre Kodierung hilft dem Neuronalen Netzwerk gleiche und ungleiche Kacheln zu unterscheiden. Nimmt man die Zahlen oder Potenzen der Kacheln direkt als Input, kann das Netzwerk

Probleme haben Kacheln mit ähnlicher Potenz zu unterscheiden. Kacheln mit 2^1 und 2^2 liegen sehr nah beieinander und würden vom Netzwerk als ähnlich interpretiert werden.

3 Ergebnisse

In den Ergebnissen wurden alle Zahlen auf Ganzzahlen gerundet.

Jeglicher Code wurde mit der Programmiersprache Python geschrieben, da diese mit PyTorch[3] eine gute Bibliothek für Neuronale Netzwerke hat, welche zur Erstellung und Training der Netzwerke verwendet wurde.

Alle Tests wurden auf folgendem System ausgeführt:

- **CPU:** AMD Ryzen 5 5600 6-Core Processor 3.50 GHz
- **RAM:** 16 GB
- **OS:** Windows 10

3.1 Genetische Algorithmus

Für den Genetischen Algorithmus wurden verschiedene Parameter gegenübergestellt. Hierbei wurde der verglichene Parameter verändert, während alle anderen Parameter auf einen moderaten Wert geschätzt und gleich geblieben sind.

Da keine der getesteten Parameter starken Einfluss auf das Ergebniss hatten, wurde darauf verzichtet, alle Kombinationen der Parameter durchzutesten.

Für jeden Test wurden 100 Generationen durchlaufen. Verglichen wurde der Verlauf des durchschnittlichen Scores aller Generationen und der beste erreichte Score und die im Median erreichte Kachel der letzten Generation. Außerdem wurde die durchschnittlich gebrauchte Zeit pro Generation abgebildet.

3.1.1 Experiment Crossover

Vergleich hohe (100%) mit moderater (50%) Crossover Chance über 100 Generationen.

- Populationsgröße: 100
- Mutationsrate: 5%
- Hidden Layer: 128, 256, 128

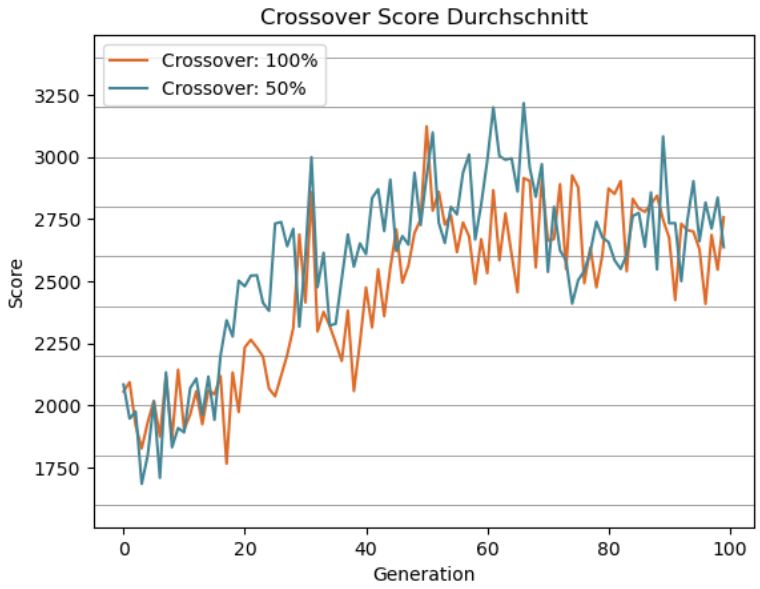

Abbildung 3.1: Durchschnittlicher Score im Trainingsverlauf durch Genetischen Algorithmus, Vergleich hoher mit moderater Crossover Rate

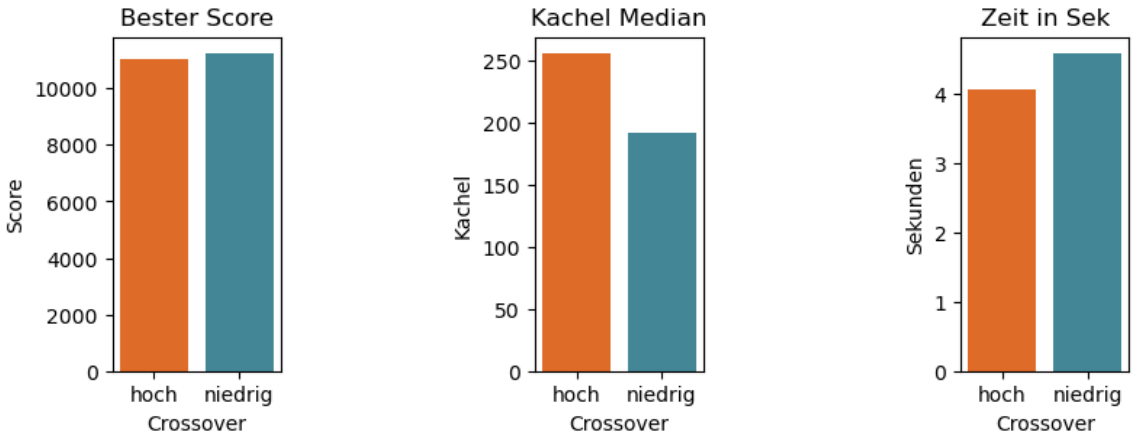

Abbildung 3.2: Bester erreichter Score (links), im Median erreichte Kachel (mitte) und gebrauchte Zeit pro Generation (rechts) im Trainingsverlauf, Vergleich hohe mit moderater Crossover Rate

3.1.2 Experiment Mutationsrate

Vergleich hohe (10%), moderate (5%) und niedrige (1%) Mutationsrate über 100 Generationen.

- Populationsgröße: 100
- Crossover: 50%
- Hidden Layer: 128, 256, 128

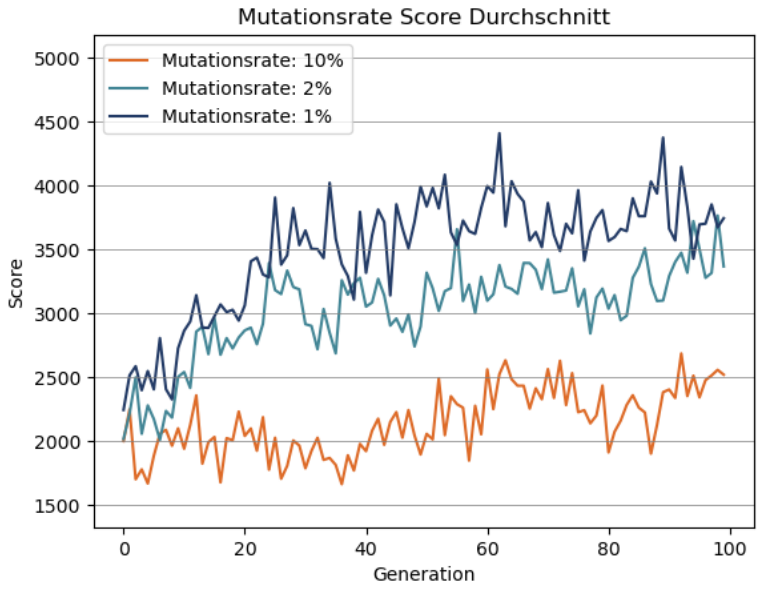

Abbildung 3.3: Durchschnittlicher Score im Trainingsverlauf durch Genetischen Algorithmus, Vergleich hohe, moderate und niedrige Mutationsrate

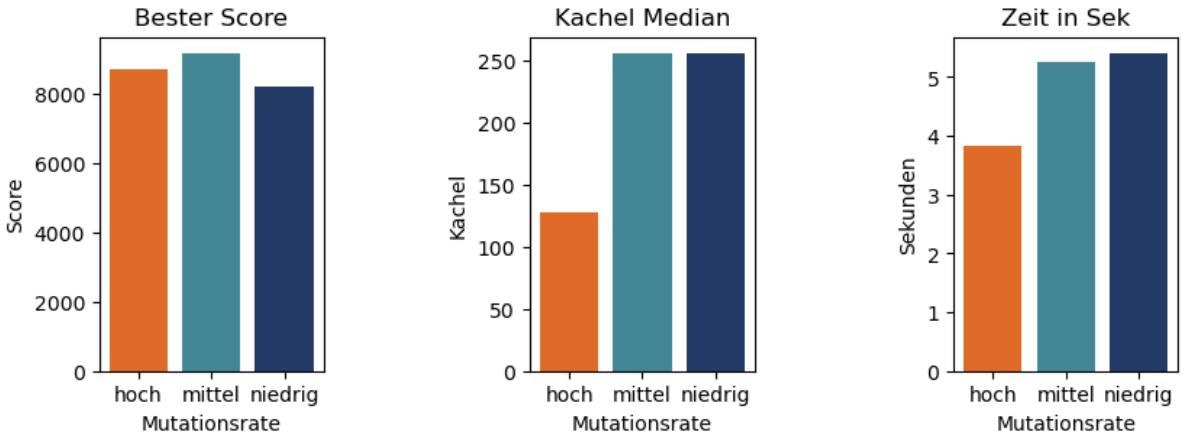

Abbildung 3.4: Bester erreichter Score (links), im Median erreichte Kachel (mitte) und gebrauchte Zeit pro Generation (rechts) im Trainingsverlauf, Vergleich hohe, moderate und niedrige Mutationsrate

3.1.3 Experiment Netzwerkgröße

Vergleich Anzahl Neuronen von Hidden Layer groß (128, 256, 512, 256, 128) mit klein (128, 256, 128) über 100 Generationen.

- Populationsgröße: 500
- Crossover: 50%

- Mutationsrate: 1%

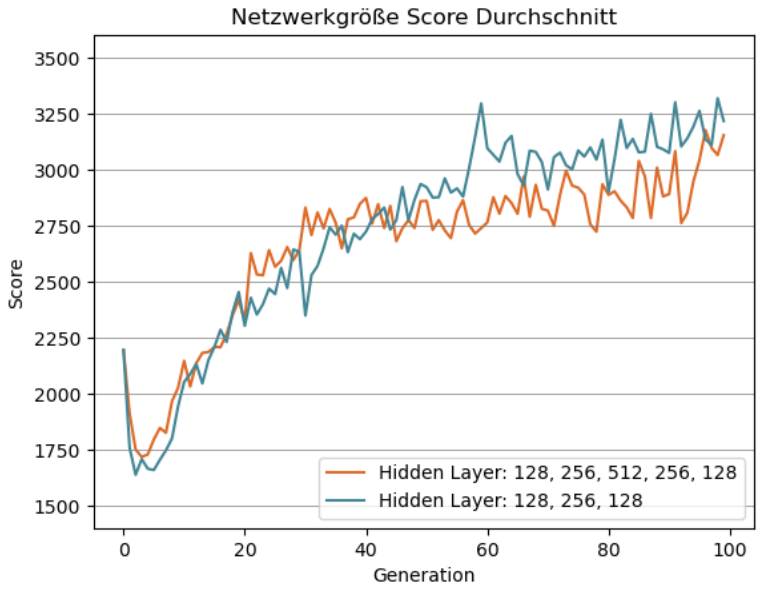

Abbildung 3.5: Durchschnittlicher Score im Trainingsverlauf durch Genetischen Algorithmus, Vergleich große und kleine Netzwerkgröße

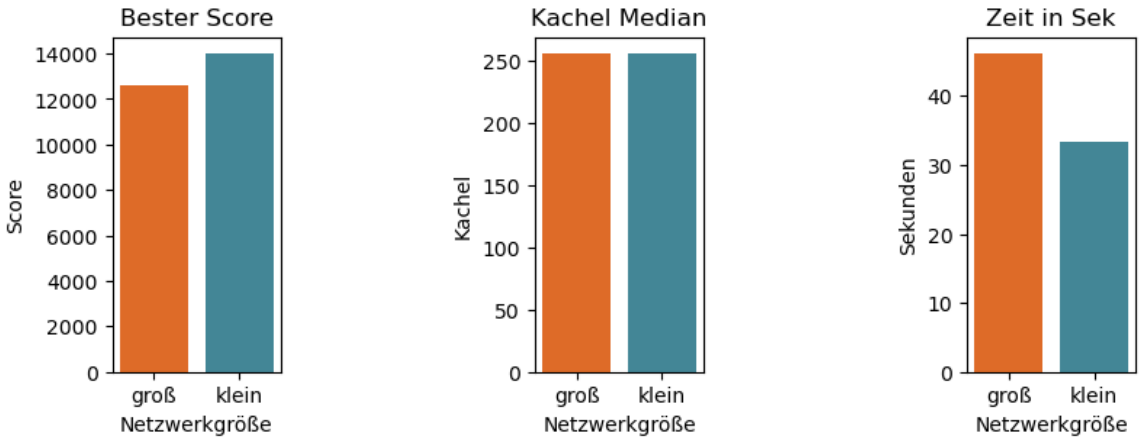

Abbildung 3.6: Bester erreichter Score (links), im Median erreichte Kachel (mitte) und gebrauchte Zeit pro Generation (rechts) im Trainingsverlauf, Vergleich große und kleine Netzwerkgröße

3.1.4 Experiment Populationsgröße

Vergleich große (500) mit kleiner (100) Populationsgröße über 100 Generationen.

- Crossover: 50%

- Mutationsrate: 1%
- Hidden Layer: 128, 256, 128

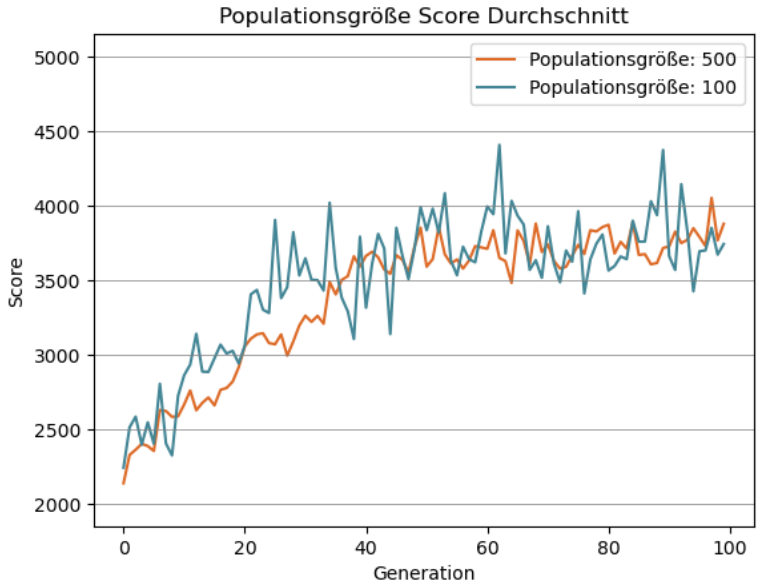

Abbildung 3.7: Durchschnittlicher Score im Trainingsverlauf durch Genetischen Algorithmus, Vergleich große und kleine Populationsgröße

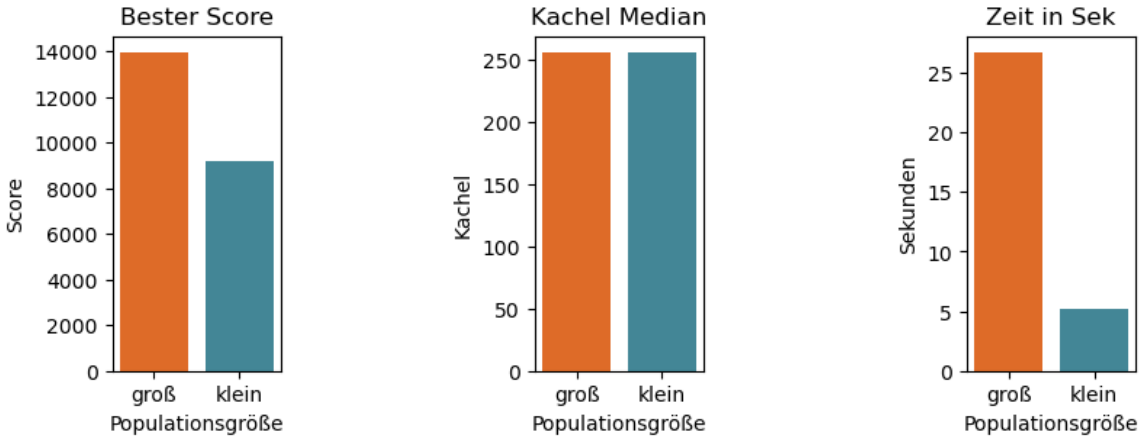

Abbildung 3.8: Bester erreichter Score (links), im Median erreichte Kachel (mitte) und gebrauchte Zeit pro Generation (rechts) im Trainingsverlauf, Vergleich große und kleine Populationsgröße

3.1.5 Experiment Convolutional Layer

Vergleich Fully Connected Netzwerk mit Convolutional Netzwerk. Für das Fully Connected Netzwerk wurden Hidden Layer mit den Größen 128, 256 und 128 verwendet. Für das Convolutional

Netzwerk wurde die gleiche Architektur benutzt wie für das Backpropagation Training, welches in Abschnitt 3.3 vorgestellt wird.

- Crossover: 50%
- Mutationsrate: 1%
- Hidden Layer: 128, 256, 128

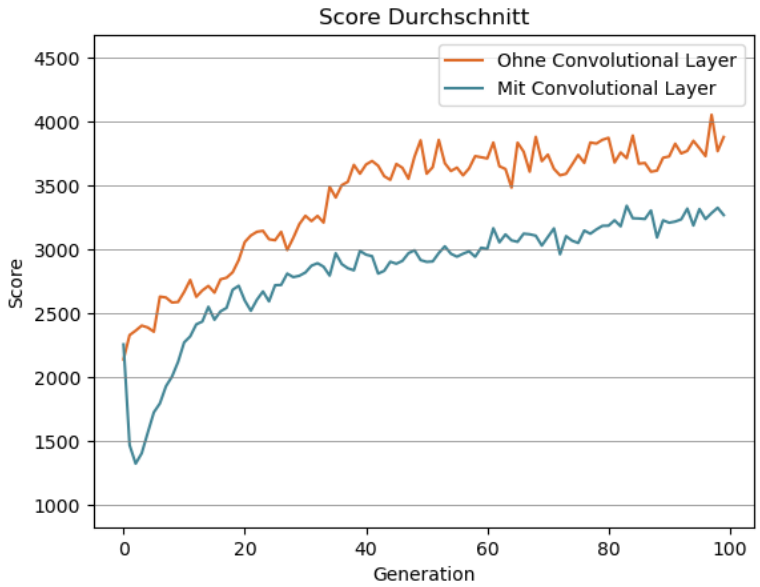

Abbildung 3.9: Durchschnittlicher Score im Trainingsverlauf durch Genetischen Algorithmus, Vergleich Convolutional Netzwerk und Fully Connected Netzwerk

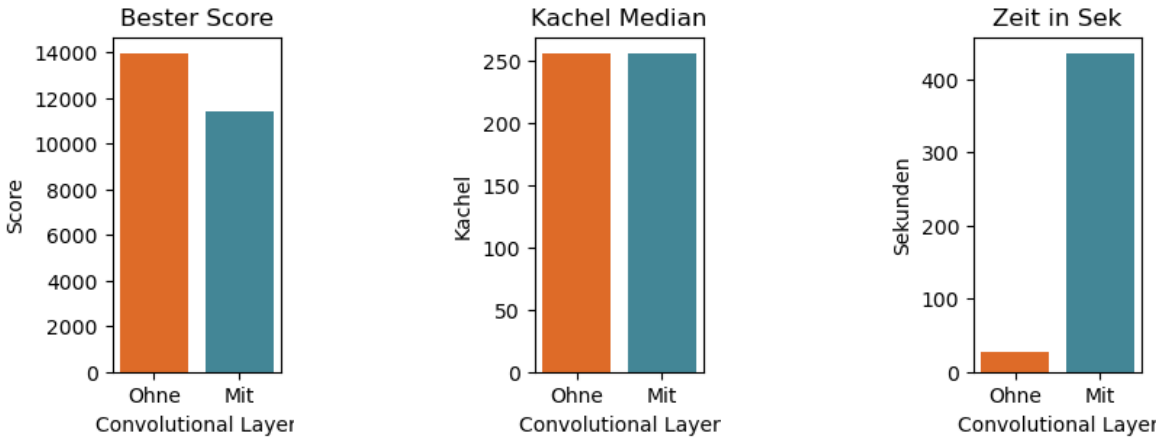

Abbildung 3.10: Bester erreichter Score (links), im Median erreichte Kachel (mitte) und gebrauchte Zeit pro Generation (rechts) im Trainingsverlauf, Vergleich Convolutional Netzwerk und Fully Connected Netzwerk

3.2 Expectimax

Für Expectimax wurden 300 Spiele mit Suchtiefe 4 und 36 Spiele mit Suchtiefe 6 gespielt. In beiden Fällen wurde Chance-Sampling 10 angewendet und eine Transposition Table genutzt. Da Suchtiefe 6 zu lange brauchte um schnell ausreichend Trainingsdaten zu erstellen und das Ergebnis von Suchtiefe 4 ausreichend gut schien und sehr schnell war, wurde der Test mit Suchtiefe 6 vorzeitig abgebrochen und darauf verzichtet weitere Tests mit etwa anderer Chance-Sampling durchzuführen.

	Suchtiefe 4	Suchtiefe 6
Durchschnittlicher Score	15784	23675
Median Score	15386	24740
Median erreichte Kachel	1024	2048
Kachel 2048 erreicht	21.91%	57.69%
Median Zeit pro Spiel	51s	18m 21s

Tabelle 3.1: Expectimax Ergebnisse

Erreichte Kacheln	1024	2048	4096
Suchtiefe 4	84%	22%	0%
Suchtiefe 6	96%	58%	4%

Tabelle 3.2: Expectimax Erreichte Kacheln

3.3 Bewertungsfunktion Backpropagation

Die im Abschnitt 3.2 vorgestellten 300 Spiele von Expectimax mit Suchtiefe 4 wurden als ursprüngliche Trainingsdaten genommen. Auf diesen Trainingsdaten wurde dann ein Fully Connected Netzwerk, und ein Convolutional Netzwerk trainiert und verglichen.

Das Fully Connected Netzwerk hatte vier Hidden Layer mit jeweils 128, 64, 32 und 16 Neuronen und es wurden 50 Spiele gespielt. Mit anderen Layergrößen wurden ähnliche Ergebnisse erziehlt.

Das Convolutional Network hatte vier Convolutional Layer, einem 16x4x4 Input Layer, welches die 16 Input Channel des Spielfeldes liest, gefolgt von einem 256x3x3 großen Layer, einem 256x4x4 und wieder einem 256x3x3 großen Convolutional Layer. Anschließend folgten vier Fully Connected Layer mit jeweils $256*3*3$, $128*3$, $64*2$ und 32 Neuronen und das Output Layer mit einem Neuron.

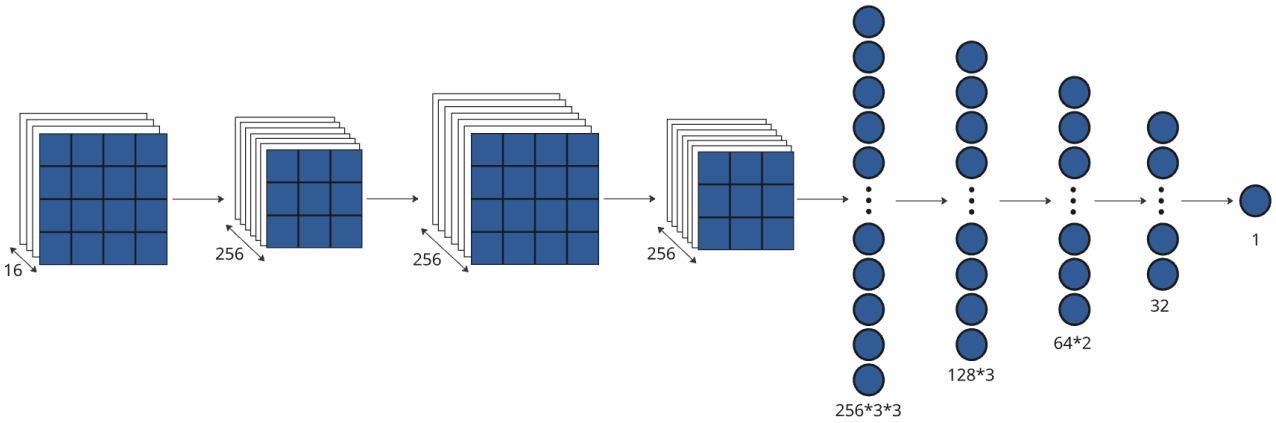

Abbildung 3.11: Genutztes Convolutional Netzwerk [5]

Die Activation Functions der Convolutional Layer war ReLU und die der Fully Connected Layer waren SELU.

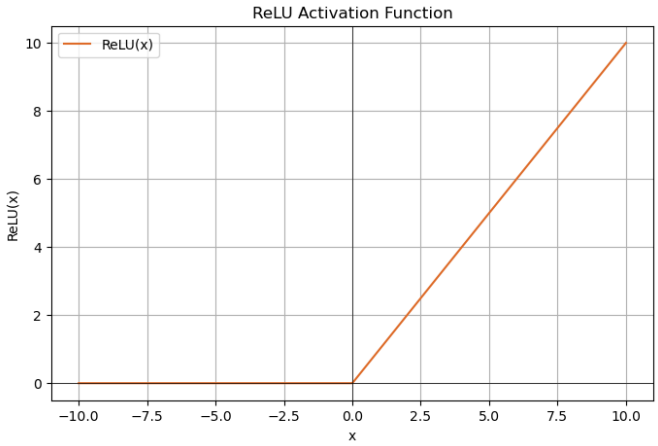

Abbildung 3.12: ReLU Activation Function

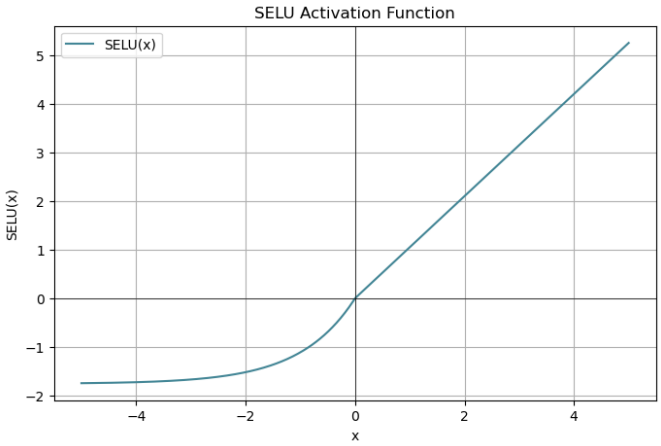

Abbildung 3.13: SELU Activation Function

Es wurde eine Kernel Size von 2x2 und ein Stride von 1 in allen Convolutional Layern verwendet. Für das erste und dritte Layer wurden wurden Zero-Padding von 0 und für das zweite Zero-Padding von 1 benutzt.

Dieses Netzwerk wurde für alle Tests genutzt, in denen ein Convolutional Netzwerk verwendet wurde.

Als Error Funktion wurde der Root Mean Square Error verwendet.

$$RMSQ = \sqrt{\frac{\sum_{i=1}^{N} (output_i - label_i)^2}{N}}$$

wobei N die Anzahl der Trainingsdaten sind. [27]

Beim Training wurde eine Learning Rate von 0.001 und eine Batch Size von 1024 benutzt. Mit diesen Parametern wurden 100 Epochs trainiert.

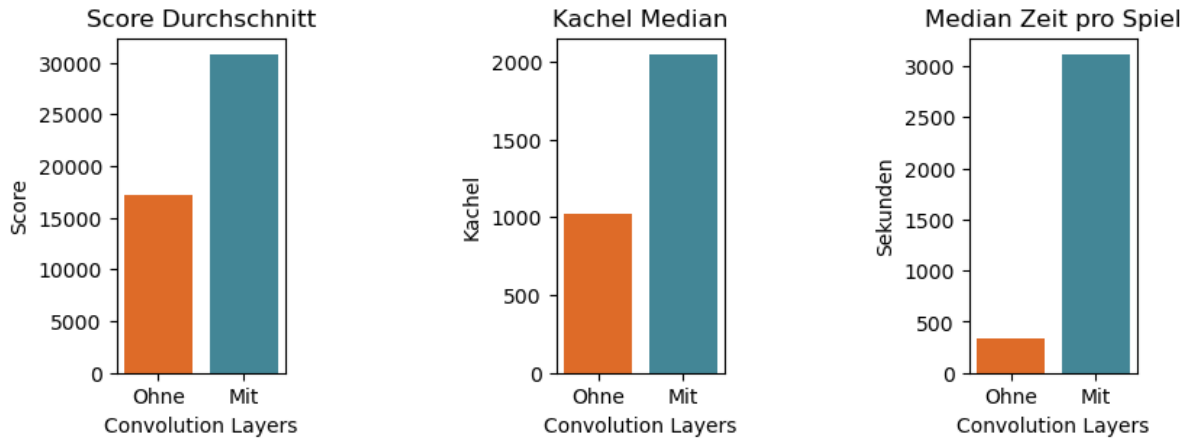

Abbildung 3.14: Bewertungsfunktion Backpropagation Durchschnittlicher Score (links), Median erreichte Kachel (mitte) und Median gebrauchte Zeit pro Spiel (rechts), Vergleich Fully Connected Netzwerk und Convolutional Netzwerk

Anschließend wurden 100 Epochs auf der Bewertungsfunktion als pretrain mit dem Convolutional Netzwerk trainiert, hierbei wurde eine Learning Rate von 0.001 verwendet und nach 50 Epochs auf 0.0001 verringert. Die Batch Size war 1024.

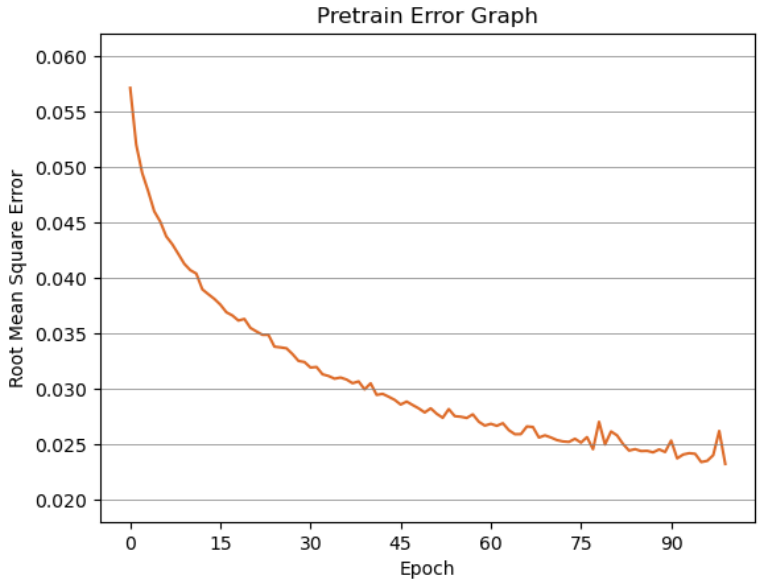

Abbildung 3.15: Pretrain Error

Dann wurden zwei Iterationen des Bewertungsfunktion Backpropagation Algorithmus mit dem Convolutional Netzwerk durchgeführt. Hierbei wurde eine Learning Rate von 0.001 verwendet und die Batch Size war ebenfalls 1024. Es wurden 100 Epochs bei jeder Iteration Trainiert. Bei der ersten Iteration wurden 150 Spiele und bei der Zweiten 100 Spiele gespielt

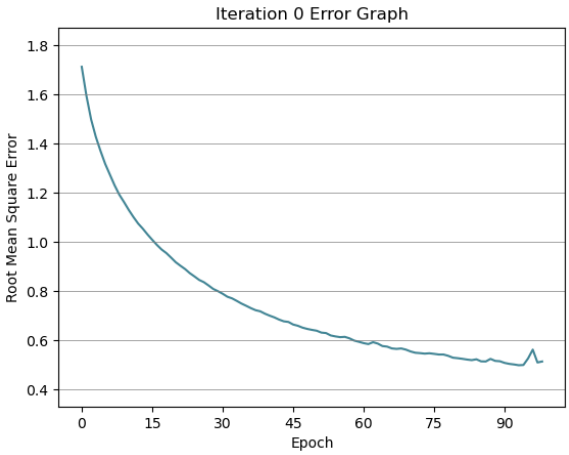

Abbildung 3.16: Iteration 1 Error

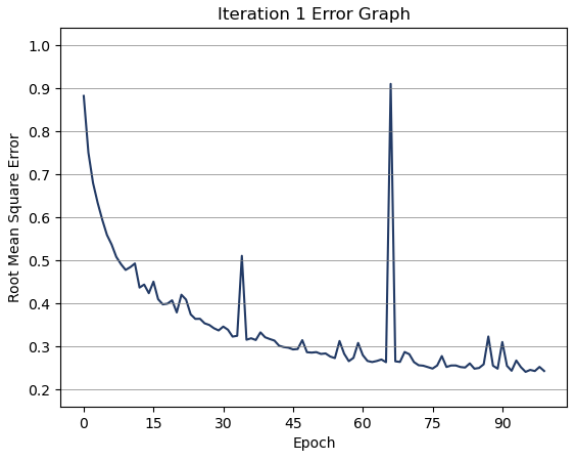

Abbildung 3.17: Iteration 2 Error

Im folgenden wurden die ursprünglichen Trainingsdaten von Expectimax als Iteration 0 bezeichnet.

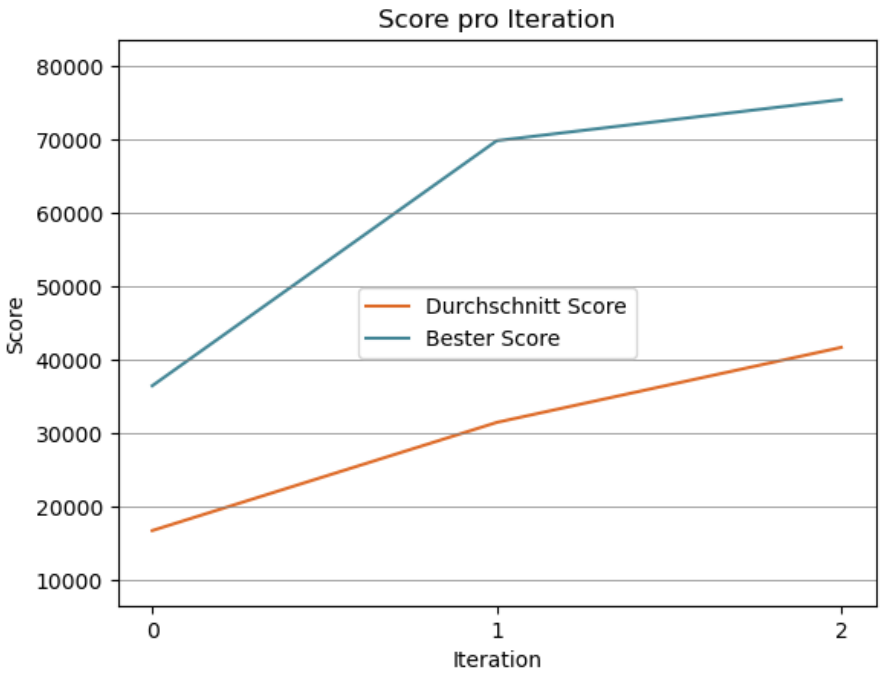

Abbildung 3.18: Bewertungsfunktion Backpropagation durchschnittlicher und bester Score pro Iteration

	Iteration 0	Iteration 1	Iteration 2
Durchschnittlicher Score	15.784	31.432	41.363
Bester Score	36.424	69.800	75.388
Median erreichte Kachel	1024	2048	2048
Durchschnittlich erreichte Kachel	1161	2016	2702
Beste Kachel	2048	4096	4096
Median Zeit pro Spiel		51m 56s	62m 33s

Tabelle 3.3: Expectimax Bewertungsfunktion Training Ergebnisse

Erreichte Kacheln	1024	2048	4096
Iteration 0	84%	22%	0%
Iteration 1	98%	79%	10%
Iteration 2	94%	85%	40%

Tabelle 3.4: Expectimax Bewertungsfunktion Training Erreichte Kacheln

3.4 Überblick

Hier wurden nochmal für die Algorithmen der durchschnittliche Score und die Häufigkeit der Kacheln, die erreicht wurden, aufgezählt.

Expectimax Ergebnisse wurden mit Suchtiefe 6 erstellt und Genetische Algorithmus Ergebnisse vom Experitment mit Großem Netzwerk. Die Backpropagation Ergebnisse sind von Iteration 2. Alle übrigen Ergebnisse stammen aus der Literatur (siehe Abschnitt 1.2.1).

	Score	1024	2048	4096
Suchbaumalgorithmen				
Expectimax (diese Arbeit)	23.675	96%	58%	4%
Minimax [29]	-	95%	70%	15%
Expectimax [29]	39.163	100%	80%	40%
Policy Netzwerke				
Genetischer Algorithmus (diese Arbeit)	3.800	94%	1%	0%
Supervised Learning [14]	93.830	94%	87%	71%
Genetischer Algorithmus [8]	2.500	-	-	-
Value Netzwerke				
Backpropagation (diese Arbeit)	41.363	95%	86%	40%
Reinforcement Learning [20]	386.973	100%	100%	99%
TD-Learning [28]	99.916	-	97%	-
TC-Learning [19]	438.515	-	-	-
Andere				
Simple Greedy [23]	3.620	-	-	-

Tabelle 3.5: Übersicht Ergebnisse

4 Diskussion

4.1 Bewertungsfunktion Backpropagation

4.1.1 Vergleich Convolutional- und Fully Connected Netzwerk

Zuerst kann man feststellen, dass das Convolutional Netzwerk deutlich besser abgeschnitten hat als das Fully Connected Netzwerk. Der durchschnittliche Score wurde verdoppelt und es wurde in über 50% der Spiele die gewinnende Kachel 2048 erreicht. Das Fully Connected Netzwerk hatte im Vergleich zu den Trainingsdaten sogar eine leichte Verschlechterung des durchschnittlichen Scores und im Median erreichte Kachel, was auf leichtes Overfitting zurückzuführen ist.

Die Zeit pro Spiel wurde im Median um 922% erhöht, also fast verzehnfacht. Die große Anzahl der Convolutions, einer komplexen Berechnung führen zu einem stark vergößerten Zeitaufwand, der im weiteren Training allerdings, wegen des guten Ergebnisses, in Kauf genommen wurde.

Für das Spielen von 2048 ist es wichtig, welche Kacheln nebeneinander liegen. Der wichtigste Teil der Bewertungsfunktion ist die Anzahl der leeren Kacheln. Die leeren Kacheln werden durch das zusammenschieben gleicher Kacheln erzeugt. Dazu ist es hilfreich zu erkennen, wo gleiche Kacheln nebeneinander liegen. Auch die Summe aller benachbarten Kacheln spielen für die Bewertungsfunktion eine Rolle. Beides lässt sich leicht durch einen Kernel eines Convolutional Layers erkennen. Ohne diese bräuchte man wahrscheinlich ein weitaus größeres Netzwerk, wodurch Overfitting mehr und mehr zum Problem wird und man weitaus mehr Trainingsdaten bräuchte.

4.1.2 Ergebnisse

Schon nach zwei Iterationen des Algorithmus war das Ergebnis sehr gut. Die Steigerung des durchschnittlichen Scores nahm kaum ab, und würde wahrscheinlich noch für einige Iterationen weiter hoch sein. Da das Spiel gegen Ende hin immer schwieriger wird, und höhere Kacheln nicht in den ursprünglichen Trainingsdaten enthalten waren, wird die Steigerung jedoch irgendwann abflachen.

Nach zwei Iterationen war ein durchschnittlicher Score von 41.532 zu sehen. Die Steigerung des besten erreichten Scores ist etwas mehr abgeflacht, aus den oben genannten Gründen.

Beide Iterationen konnten weder die im Median erreichte Kachel von 2048 auf 4096 noch die beste Kachel von 4096 auf 8192 verbessern.

In der durchschnittlichen erreichten Kachel war eine Steigerung zu sehen. Diese lag nach der ersten Iteration noch knapp unter der 2048 Kachel, nach der zweiten Iteration dann deutlich drüber, was

darauf hindeuted, dass nach mehr Iterationen die im Median erreichte Kachel auch irgendwann auf 4096 hochgehen würde.

Die Zeit pro Spiel hat sich im Vergleich zu Expectimax mit Suchtiefe 6 zur Iteration 2 mehr als verdreifacht. Während jedoch die Zeit mit Expectimax pro Suchtiefe exponentiell ansteigt, bleibt sowohl Trainingszeit als auch Spielzeit mit dem Backpropagation Algorithmus pro Iteration konstant (die Spielzeit verlängert sich lediglich etwas, auf Grund besserer, dh. längerer Spiele). Wenn man den Algorithmus noch ein bisschen verbessert und mehr Iterationen laufen lässt, ist vermutlich ziemlich schnell ein Punkt gekommen, wo er bessere Ergebnisse in schnellerer Zeit liefert als selbst optimierte Expectimax Algorithmen.

Die Error Graphen der Iterationen sehen zwar insgesamt gut aus, jedoch ist nicht ganz klar, ob mit mehr Epochs der Error nicht noch weiter runter gehen würde, vor allem wenn man die Learning Rate nochmal verringern würde. Iteration 2 hat eine generell sehr grobe Kurve mit vielen Spitzen, was für eine generell zu hohe Learning Rate spricht. Generell hat sich der Error allerdings gut verringert.

4.2 Genetischer Algorithmus

Das Training durch Genetische Algorithmen hat in den Tests nicht gut abgeschnitten. Crossover und Netzwerkgröße haben sich kaum auf das Ergebnis ausgewirkt. Durch eine niedrige Mutationsrate und große Populationsgröße ließ sich das Ergebnis ein wenig, jedoch nicht bedeutend verbessern. Die Populationsgröße hat sich dabei wesentlich stärker auf die Zeit pro Generation ausgewirkt und nicht bedeutend bessere Ergebnisse erzielt als mit kleiner Populationsgröße und niedriger Mutationsrate.

Interessant war, dass das Wechseln zu einen Convolutional Netzwerk in einer Verschlechterung der Performance resultierte. Gleichzeitig dauerte auch hier das Spielen der Testspiele spürbar länger. Nach den 100 Testgenerationen ließ sich weiterhin eine kleine Steigerung der durchschnittlichen Fitness feststellen, jedoch so gering, dass der Unterschied kaum merkbar war. Dies deutet darauf hin, dass die höhere Komplexität der Convolutional Layer zwar besser für das Spiel geeignet sind, aber auch schwieriger durch den Genetischen Algorithmus zu lernen und deutlich mehr Zeit oder größere Populationsgröße braucht.

Durch den im Abschnitt 2.3 beschriebenen verbesserten Genetischen Algorithmus wurde versucht das Ergebnis zu erhöhen. Die beschriebenen negativen Auswirkungen auf die Laufzeit waren jedoch so hoch, dass sich kein sinnvolles Ergebnis produzieren ließ.

4.3 Vergleich Bewertungsfunktion Backpropagation und Genetischer Algorithmus

Im direkten Vergleich schneidet das Training durch Genetische Algorithmen deutlich schlechter ab als der Bewertungsfunktion Backpropagation Algorithmus. Der einzige Vorteil ist die Trainingszeit.

Während Genetische Algorithmen auch mit etwas größerer Populationsgröße mehrere hundert Generationen am Tag schaffen, brauchte das Backpropagation Prinzip auf dem für die Tests verwendeten System mehrere Tage pro Iteration. Der größte Faktor hier war das Erstellen der Trainingsdaten. Es mussten trotz Data Augmentation mindestens 100 Spiele pro Iteration gespielt werden, um Overfitting zu vermeiden. Auch mussten diese Spiele mit Suchtiefe 4 gespielt werden, da Suchtiefe 2 deutlich schlechtere Ergebnisse erzielt hat.

Trotz dieses Nachteils ist es fraglich, ob der Genetische Algorithmus auch mit mehr Zeit und größerer Population ansatzweise an die Ergebnisse des anderen herankommt, da er immer schon nach 100 Generationen ein Plateau erreicht hat, und dabei in den besten Fällen lediglich ähnliche Ergebnisse wie von Expectimax mit Suchtiefe 4 erzielte, während das Wachstum der Ergebnisse des Backpropagation nach 2 Iterationen noch konstant hoch war.

4.4 Vergleich Literatur

Im Vergleich mit der Literatur sieht man, dass der Genetische Algorithmus nicht wesentlich besser abschneidet als ein einfacher Greedy Algorithmus ([23]). Der Genetische Algorithmus dieser Arbeit schneidet etwas besser ab als der, der in der Literaturrecherche gefunden wurde ([8]), was wahrscheinlich hauptsächlich am Input reading lag, welches in der gefundenen Arbeit nicht in eine Binärkodierung umgewandelt wurde.

Interessant ist, dass die Implementierung des Expectimax dieser Arbeit schlechter abschneidet als die der Literatur ([29]). Dies liegt wahrscheinlich hauptsächlich am Chance Sampling, was zur Verbesserung der Spielzeit implementiert wurde und wichtig ist, um viele Trainingsdaten zu erstellen.

Was Score betrifft, ist der Bewertungsfunktion Backpropagation Algorithmus zwar besser als der Genetische Algorithmus, trotzdem wesentlich schlechter als die besten Algorithmen aus der Literatur ([14], [19], [20], [28]). Mit mehr Iterationen könnte er etwas aufholen, würde jedoch ohne Verbesserungen wahrscheinlich nicht an die besten Algorithmen ran kommen. Er schaffte jedoch jetzt schon eine beeindruckende 90% Siegesrate, was auch im Vergleich mit anderen Ansätzen schon befriedigend war.

4.5 Vergleich Value Netzwerk und Policy Netzwerk

Insgesamt sieht man, dass Value Netzwerke deutlich besser funktionieren als Policy Netzwerke. Das beste gefundene Policy Netzwerk schneidet zwar immer noch besser ab als das Value Netzwerk dieser Arbeit, vermutlich könnte dies durch mehr Iterationen zumindest ausgeglichen werden.

Alle Value Netzwerke und auch das Policy Netzwerk mit Supervised Learning zeigten bessere Ergebnisse als die gefundenen Suchbaumalgorithmen, was die Nützlichkeit vom Einsatz von Neuronalen Netzwerken in Spielalgorithmen unterstreicht.

5 Fazit

In dieser Arbeit habe ich Trainingsmethoden für Neuronale Netzwerke am Beispiel des Spiels 2048 verglichen und die Ergebnisse im Kontext zur Literatur vorgestellt.

Die Arbeit ist zu dem sehr klaren Ergebnis gekommen, dass das Prinzip eine Expectimax Bewertungsfunktion durch Backpropagation zu lernen sehr viel bessere Ergebnisse erzeugt als ein durch Genetische Algorithmen trainiertes Netzwerk, das direkt 2048 spielt.

Value Netzwerke scheinen generell besser zu funktionieren als Policy Netzwerke. Es ist jedoch fraglich, ob die Bewertungsfunktion Backpropagation Methode, die in dieser Arbeit vorgestellt wurde, an die besten Ergebnisse der Literatur mit mehr Iterationen herankommen würde.

5.1 Bewertungsfunktion Backpropagation Verbesserungen

Die Ergebnisse des durch Backpropagation trainierten Netzwerkes, würden sich noch verbessern lassen, indem man mit einer besseren Bewertungsfunktion startet.

Aus Zeitgründen wurden lediglich zwei Iterationen der Backpropagation Methode durchgeführt. Man könnte schauen, wie gut es mit mehr Iterationen werden kann. In den ursprünglichen Trainingsdaten waren nur Kacheln bis 2048 enthalten, was vermutlich bei späteren Iterationen, wo die höheren Kacheln immer öfters auftreten, in schlechteren Verhalten im späteren Verlauf der Spiele resultieren würde. Dies könnte man lösen, indem man zufällige oder teilweise zufällige Spielfelder in die Trainingsdaten der ersten Iteration hinzufügt.

Ebenfalls könnte man schauen, ob sich das Ergebnis stark verbessert, wenn man Expectimax ohne Chance Sampling laufen lässt.

Die Geschwindigkeit beim Erstellen der Trainingsdaten ließe sich verbessern, indem man Teilbäume von Expectimax parallel berechnet, wie in dieser Arbeit [15] beschrieben. Die Programmiersprache Python trägt ebenfalls zur langsamen Geschwindigkeit bei, hier könnte man das Training in Python belassen um auf die umfangreichen Bibliotheken zuzugreifen, jedoch das Erstellen der Trainingsdaten, sprich die Ausführung von Expectimax in einer anderen, schnelleren Sprache programmieren.

Ein Durchlauf mit Profiler hat gezeigt, dass das Kopieren der Spielfelder des Expectimax und das Hashen für die Transposition Table am meisten Zeit kosteten. In der Implementierung wurde ein Spielfeld als Objekt dargestellt. Ein anderer Ansatz zur Verbesserung der Geschwindigkeit könnte also sein, wenn anstatt einer objektorientierten Lösung, der Spielstatus als Liste von Zahlen gespeichert

wird, die die Kacheln repräsentieren und alle spielbezogenen Funktionen funktional programmiert werden. Dann würde nicht das ganze Objekt, sondern lediglich die Liste kopiert werden, was schneller sein sollte.

5.2 Überprüfung Korrektheit

Um zu überprüfen, wie gut der Algorithmus wirklich funktioniert, könnte man Expectimax mit Suchtiefe 8 spielen und die Ergebnisse mit der ersten Iteration vergleichen. Wenn das Netzwerk diese Bewertungen von Suchtiefe 4 perfekt lernt und dann in der ersten Iteration wieder Expectimax mit Suchtiefe 4 mit dieser Bewertungsfunktion gespielt wird, wird die Suchtiefe hinten drangehängt und das Ergebnis sollte ähnlich sein, wie wenn man gleich mit Suchtiefe 8 spielt. In diesem Fall wahrscheinlich sogar besser, da durch Chance Sampling immer Qualität verloren geht.

Falls die Ergebnisse zu stark abweichen würden, könnte man die Lernfähigkeit noch verbessern, indem man zum Beispiel Chance Sampling beim Expectimax weglässt oder mehr Netzwerkstrukturen testet.

5.3 Genetischer Algorithmus Verbesserungen

Für das durch den Genetischen Algorithmus trainierten Netzwerkes ließen sich durch experimentieren mit verschiedenen Layergrößen des Convolutional Netzwerkes eventuell etwas bessere Ergebnisse in ähnlicher Zeit finden.

Außerdem könnte man schauen ob mit einer sehr viel größeren Population ebenfalls ab Generation 100 ein Plateau entsteht oder mit sehr viel mehr Generationen diese Plateau überwunden werden kann.

Genetische Algorithmen könnte man auch benutzen um die Struktur des Neuronalen Netzwerkes zu verbessern, wie in dieser Arbeit [16] beschrieben.

5.4 Value Netzwerk mit Genetischen Algorithmen

Interessant wäre zu sehen, ob eine bessere initiale Bewertungsfunktion des Expectimax durch einen Genetischen Algorithmus trainiert werden kann, indem man als Output nicht vier Zahlen für die Spielzüge hat, sondern lediglich einen und als Fitnessfunktion der Generation ein komplettes Spiel durch Expectimax mit diesem Output als Bewertungsfunktion spielen lässt. Dies könnte in einer besseren Bewertungsfunktion als der Handgemachten resultieren, mit der man dann den Bewertungsfunktion Backpropagation Algorithmus durchführen könnte. Geschwindigkeit wäre hier allerdings ebenfalls ein limitierender Faktor.

Falls das funktioniert, könnte man beide Methoden verbinden indem man anschließend auf dieser Bewertungsfunktion das Backpropagation Prinzip anwendet.

5.5 Schlusswort

Diese Arbeit hat gezeigt, dass es nicht einfach ist in kurzer Zeit komplexes Verhalten durch Genetische Algorithmen zu erzeugen. Auf der anderen Seite lassen sich durch Neuronale Netzwerke Suchbaumalgorithmen lernen, und so die Suchtiefe erweitern. Dies ist vor allem hilfreich, wenn die Komplexität der Suche mit der Suchtiefe stark ansteigt und sich auch auf Suchbaumalgorithmen außerhalb der Spieltheorie übertragen lässt.

Literaturverzeichnis

[1] 2048 wikipedia. https://de.wikipedia.org/wiki/2048_(Computerspiel). Accessed: 28.4.2025.

[2] Professor ha-chin yi home page. https://hachinyi.wp.txstate.edu/deep-learning/?utm_source=chatgpt.com#batches. Accessed: 28.4.2025.

[3] Pytorch. https://pytorch.org/. Accessed: 28.4.2025.

[4] Screenshot aus offiziellem spiel. https://2048game.com/de/. Accessed: 28.4.2025.

[5] Selbsterstellte grafik durch miro.com.

[6] Spieltheorie wikipedia. https://de.wikipedia.org/wiki/Spieltheorie. Accessed: 28.4.2025.

[7] Ajith Abraham. Artificial neural networks. *Handbook of measuring system design*, 2005.

[8] Tuponja Boris and Šuković Goran. Evolving neural network to play game 2048. In *2016 24th Telecommunications Forum (TELFOR)*, pages 1–3. IEEE, 2016.

[9] Dennis M Breuker and Jos WHM Uiterwijk. Transposition tables in computer chess. *Department of Computer Science, University of Limburg*, 2002.

[10] Haruna Chiroma, Ahmad Shukri Mohd Noor, Sameem Abdulkareem, Adamu I Abubakar, Arief Hermawan, Hongwu Qin, Mukhtar Fatihu Hamza, and Tutut Herawan. Neural networks optimization through genetic algorithm searches: a review. *Appl. Math. Inf. Sci*, 11(6):1543–1564, 2017.

[11] H Guei, T Wei, JB Huang, and IC Wu. An early attempt at applying deep reinforcement learning to the game 2048. In *Workshop on Neural Networks in Games*, 2016.

[12] Jatinder ND Gupta and Randall S Sexton. Comparing backpropagation with a genetic algorithm for neural network training. *Omega*, 27(6):679–684, 1999.

[13] Thomas Hauk, Michael Buro, and Jonathan Schaeffer. Rediscovering*-minimax search. In *International Conference on Computers and Games*, pages 35–50. Springer, 2004.

[14] Naoki Kondo and Kiminori Matsuzaki. Playing game 2048 with deep convolutional neural networks trained by supervised learning. *Journal of Information Processing*, 27:340–347, 2019.

[15] Rigie G Lamanosa, Kheiffer C Lim, Ivan Dominic T Manarang, Ria A Sagum, and Maria-Eriela G Vitug. Expectimax enhancement through parallel search for non-deterministic games. *International Journal of Future Computer and Communication*, 2(5):466, 2013.

[16] Frank Hung-Fat Leung, Hak-Keung Lam, Sai-Ho Ling, and Peter Kwong-Shun Tam. Tuning of the structure and parameters of a neural network using an improved genetic algorithm. *IEEE Transactions on Neural networks*, 14(1):79–88, 2003.

[17] Timothy P Lillicrap, Adam Santoro, Luke Marris, Colin J Akerman, and Geoffrey Hinton. Backpropagation and the brain. *Nature Reviews Neuroscience*, 21(6):335–346, 2020.

[18] Shiva Maharaj, Nick Polson, and Alex Turk. Chess ai: competing paradigms for machine intelligence. *Entropy*, 24(4):550, 2022.

[19] Kiminori Matsuzaki. Developing a 2048 player with backward temporal coherence learning and restart. In *Advances in Computer Games: 15th International Conferences, ACG 2017, Leiden, The Netherlands, July 3–5, 2017, Revised Selected Papers 15*, pages 176–187. Springer, 2017.

[20] Kiminori Matsuzaki. Developing value networks for game 2048 with reinforcement learning. *Journal of Information Processing*, 29:336–346, 2021.

[21] Seyedali Mirjalili and Seyedali Mirjalili. Genetic algorithm. *Evolutionary algorithms and neural networks: Theory and applications*, pages 43–55, 2019.

[22] Eduardo F Morales and Hugo Jair Escalante. A brief introduction to supervised, unsupervised, and reinforcement learning. In *Biosignal processing and classification using computational learning and intelligence*, pages 111–129. Elsevier, 2022.

[23] Todd W. Neller. Pedagogical possibilities for the 2048 puzzle game. *Journal of computing sciences in colleges*, 30(3):38–46, 1 2015.

[24] Keiron O'shea and Ryan Nash. An introduction to convolutional neural networks. *arXiv preprint arXiv:1511.08458*, 2015.

[25] Stuart Russell and Peter Norvig. Artificial intelligence: a modern approach. 3rd. *Upper Saddle River, EUA: Prentice-Hall*, 2010.

[26] Randall S Sexton and Jatinder ND Gupta. Comparative evaluation of genetic algorithm and backpropagation for training neural networks. *Information sciences*, 129(1-4):45–59, 2000.

[27] Deepak Kumar Sharma, Mayukh Chatterjee, Gurmehak Kaur, and Suchitra Vavilala. Deep learning applications for disease diagnosis. In *Deep learning for medical applications with unique data*, pages 31–51. Elsevier, 2022.

[28] Marcin Szubert and Wojciech Jaśkowski. Temporal difference learning of n-tuple networks for the game 2048. In *2014 IEEE Conference on Computational Intelligence and Games*, pages 1–8. IEEE, 2014.

[29] Ahmad Zaky. Minimax and expectimax algorithm to solve 2048. 2014.