

# OthelloZero

Vom leeren Brett  
zur Meisterstrategie



18. August 2025,  
Kreuzlingen

Maturitätsarbeit  
Von Leo Pracht  
Betreut von Franziska Flegel

## **Zusammenfassung**

Diese Arbeit untersucht die Entwicklung einer selbstlernenden Künstlichen Intelligenz für das Brettspiel Othello nach dem AlphaZero-Prinzip (Silver, Hubert et al., 2017a; Silver, Schrittwieser et al., 2017b). Ziel war es, zu zeigen, dass leistungsfähige Spiel-KI auch mit begrenzten Rechenressourcen realisierbar ist. Grundlage bildet ein neuronales Netz, das in einem iterativen Self-Play-Prozess trainiert und durch Monte Carlo Tree Search (MCTS) gesteuert wird. Das gesamte Training erfolgte lokal auf einem handelsüblichen Heimrechner, ohne externe Daten oder Hochleistungsrechenzentren.

Das entwickelte System wurde über 55 Trainingsgenerationen hinweg optimiert. Die Evaluation erfolgte in umfangreichen Vergleichspartien gegen die etablierte Othello-Engine Edax (Delorme, 2025) auf verschiedenen Schwierigkeitsstufen sowie gegen menschliche Spieler. Die Ergebnisse zeigen eine stetige Steigerung der Spielstärke bis zu einer geschätzten Elo-Wertung von etwa 1780, knapp unterhalb von Edax Level 6. Policy-Heatmaps und Partienanalysen verdeutlichen eine zunehmende strategische Reife, darunter die gezielte Nutzung von Ecken, das Vermeiden riskanter Felder und eine adaptive Positionskontrolle.

Die Resultate belegen, dass auch kompakte Architekturen in Kombination mit effizienter Suche eine hohe Spielstärke erreichen können. Die Arbeit leistet damit einen Beitrag zur Idee einer ressourcenschonenden “Green AI” (Schwartz et al., 2020) und zeigt, wie moderne KI-Konzepte praxisnah und zugänglich umgesetzt werden können.

# Inhaltsverzeichnis

<b>1 Einleitung</b>	<b>1</b>
1.1 Vorwort . . . . .	1
1.2 Fragestellung . . . . .	2
1.3 Zielsetzung . . . . .	2
<b>2 Theoretische Grundlagen</b>	<b>3</b>
2.1 Othello . . . . .	3
2.1.1 Spielregeln . . . . .	3
2.1.2 Othello in Zahlen . . . . .	5
2.1.3 Strategien . . . . .	5
2.2 Einführung in neuronale Netze . . . . .	8
2.2.1 Warum neuronale Netze? . . . . .	8
2.2.2 Grundidee eines neuronalen Netzes . . . . .	8
2.2.3 Lernprozess eines neuronalen Netzes . . . . .	10
2.2.4 Convolutional Neural Networks (CNNs) . . . . .	11
2.2.5 Residual Blocks . . . . .	12
2.3 Monte Carlo Tree Search . . . . .	13
2.3.1 Knoten . . . . .	13
2.3.2 Selektion . . . . .	14
2.3.3 Expandierung . . . . .	15
2.3.4 Rückpropagierung . . . . .	17
2.3.5 Ausführung . . . . .	18
2.3.6 Zugextraktion . . . . .	18
2.4 AlphaZero und Self-Play . . . . .	19
2.4.1 Zusammenspiel von Monte Carlo Tree Search und neuronalen Netzen .	19
2.4.2 Self-Play: Lernen durch Erfahrung . . . . .	21
2.4.3 Der Lernzyklus von AlphaZero . . . . .	21
2.5 Elo-Bewertungssystem . . . . .	23

2.5.1	Grundprinzip . . . . .	23
2.5.2	Verwendung in KI-Systemen . . . . .	24
2.6	Edax als Referenz-Gegner . . . . .	24
2.6.1	Funktionsweise von Edax . . . . .	24
2.6.2	Einsatz zur Spielstärkebewertung . . . . .	25
2.7	Warum AlphaZero für Othello? . . . . .	25
2.8	Theoretischer Rahmen im Überblick . . . . .	26
<b>3</b>	<b>Methodik</b> . . . . .	<b>27</b>
3.1	Technische Grundlagen und Tools . . . . .	27
3.2	Struktur der Implementierung . . . . .	27
3.2.1	Modularer Aufbau . . . . .	28
3.2.2	Parallelisierung durch Manager-Worker-Struktur . . . . .	29
3.3	Neuronales Netz . . . . .	30
3.3.1	Eingabeformat . . . . .	32
3.3.2	Feature-Extraktion und Residual-Blöcke . . . . .	32
3.3.3	Policy- und Value-Head . . . . .	33
3.3.4	Verlustfunktion . . . . .	34
3.4	Trainingsprozess . . . . .	34
3.4.1	Trainingsstrategie . . . . .	34
3.4.2	Optimierung . . . . .	36
3.4.3	Trainingsdauer . . . . .	36
3.5	Evaluation . . . . .	36
3.5.1	Versuchsaufbau . . . . .	36
3.5.2	Suchstrategie und Variabilität . . . . .	37
3.5.3	Datenerhebung und Zielgrösse . . . . .	37
3.6	Benutzeroberfläche und Visualisierung . . . . .	37
3.6.1	Darstellung des Spielbretts . . . . .	38
3.6.2	Policy-Darstellung . . . . .	38
3.6.3	Neuronale Bewertung . . . . .	39
3.6.4	MCTS-Informationen . . . . .	39
3.6.5	Zugliste . . . . .	41
3.6.6	Zweck und Nutzen . . . . .	41
3.7	Reflexion und Limitationen . . . . .	41
<b>4</b>	<b>Ergebnisse und Diskussion</b> . . . . .	<b>43</b>
4.1	Ergebnisse . . . . .	43
4.1.1	Trainingsverlauf . . . . .	44

4.1.2	Spielstärkeentwicklung . . . . .	45
4.1.3	Lernverhalten . . . . .	46
4.2	Analyse einer Partie gegen das Modell . . . . .	47
4.2.1	Frühe Stellung: nur ein Stein . . . . .	48
4.2.2	Endspiel mit Eckfokus . . . . .	49
4.2.3	Endsituation und Sieg . . . . .	50
4.3	Diskussion . . . . .	51
4.3.1	Bewertung des Trainingsverlaufs . . . . .	51
4.3.2	Analyse der Spielstärkeentwicklung . . . . .	51
4.3.3	Strategisches Verhalten und Policy . . . . .	52
4.3.4	Einordnung der analysierten Partie . . . . .	53
4.3.5	Limitationen und mögliche Verbesserungen . . . . .	54
<b>5</b>	<b>Fazit</b>	<b>56</b>
5.1	Zusammenfassung . . . . .	56
5.2	Beantwortung der Forschungsfrage . . . . .	57
5.3	Ausblick . . . . .	58

# Abbildungsverzeichnis

Titelbild der Arbeit (generiert mit ChatGPT, 2025) . . . . .	i
2.1 Startaufstellung einer Othellopartie. („Rules & Strategy - HKOA“, 2022). . . . .	3
2.2 Mögliche Züge von Schwarz aus der Startaufstellung. („Rules & Strategy - HKOA“, 2022). . . . .	4
2.3 Spielbrett nach gültigem Zug von Schwarz und umgedrehten Steinen. („Rules & Strategy - HKOA“, 2022). . . . .	4
2.4 Mögliche Spielzüge von Weiss. („Rules & Strategy - HKOA“, 2022). . . . .	5
2.5 Stabiler schwarzer Stein in der Ecke (h8). Aufgrund der Spielregeln kann dieser nicht mehr umgedreht werden. Auch angrenzende Steine entlang der Kante können potenziell stabil sein. („Rules & Strategy - HKOA“, 2022). . . . .	6
2.6 Bevorzugte Felder im zentralen 4×4-Bereich (weiss) und riskante Felder angrenzend an die Ecken (dunkel hervorgehoben). („Rules & Strategy - HKOA“, 2022). . . . .	7
2.7 Taktisch erfolgreiche Spielsituation: Die weisse Partei gewinnt mit nur einem verbliebenen Stein auf dem Feld. („Rules & Strategy - HKOA“, 2022). . . . .	7
2.8 Klassische Programmierung (oben) im Vergleich zu maschinellem Lernen (unten) (Eigene Abbildung). . . . .	8
2.9 Schematische Darstellung eines Feedforward-Netzes für Othello mit separaten Policy- (Softmax) und Value-Ausgaben (tanh) (Eigene Abbildung). . . . .	9
2.10 Lernzyklus eines neuronalen Netzes – bestehend aus Vorwärtsdurchlauf, Fehlerberechnung, Rückpropagation und Gewichtsupdate (Eigene Abbildung). . . . .	11
2.11 Funktionsweise einer Faltungsschicht: Ein Filter (Kernel) berechnet für jede Position eine gewichtete Summe der Nachbarschaft, woraus eine Feature Map entsteht (Eigene Abbildung). . . . .	12
2.12 Aufbau eines Residual Blocks: Der Hauptpfad $F(x)$ enthält zwei Faltungsschichten, die Skip-Verbindung gibt die Eingabe $x$ direkt weiter (Eigene Abbildung). . . . .	12

2.13 Visualisierung der Selektionsphase: Ausgehend vom Wurzelknoten $s_0$ wird schrittweise der Nachfolger mit dem höchsten PUCT-Wert (blau markierter Hauptpfad) gewählt, bis ein Blattknoten erreicht wird. Alternative Pfade sind gestrichelt dargestellt, unbesuchte Knoten bleiben leer (Eigene Abbildung). . . . .	14
2.14 Visualisierung der Expandierungsphase: Nach Erreichen eines Blattknotens $s_3$ werden neue Kindknoten für alle legalen Aktionen angelegt (grün markiert). Der Selektionspfad ist blau dargestellt, unbesuchte Knoten bleiben leer (Eigene Abbildung). . . . .	16
2.15 Visualisierung der Rückpropagierungsphase: Der vom Blattknoten stammende Wert $v$ wird entlang des Selektionspfades bis zur Wurzel zurückgegeben. Rote Pfeile zeigen den Wertfluss, wobei bei jedem Schritt die Perspektive durch Negation gewechselt wird (Eigene Abbildung). . . . .	17
2.16 Schematische Darstellung des AlphaZero-Lernzyklus. Das aktuelle Netzwerk generiert durch Self-Play neue Trainingsdaten. Nach Training und Evaluation wird es bei ausreichender Überlegenheit aktualisiert (Eigene Abbildung). . . . .	22
 3.1 Modularer Aufbau des Projekts (Eigene Abbildung). . . . .	28
3.2 Manager-Worker-Architektur zur parallelen Ausführung von Self-Play und Batch-Abfrage des neuronalen Netzes (Eigene Abbildung). . . . .	30
3.3 Architektur des neuronalen Netzes. Das Netzwerk verarbeitet ein $8 \times 8$ -Othellobrett mit drei Eingabekanälen und verzweigt sich nach einem gemeinsamen Feature-Extractor in einen Policy- und einen Value-Head (Eigene Abbildung). . . . .	31
3.4 Visualisierung des Netzwerkinputs (Startaufstellung): Die Brettstellung wird in drei separaten Kanälen dargestellt. Gezeigt werden (von links nach rechts) die Positionen der eigenen Steine, der gegnerischen Steine sowie die aktuell legalen Züge für den Spieler. Jeder Kanal hat die Dimension $8 \times 8$ , sodass sich insgesamt ein Eingabetensor der Form $3 \times 8 \times 8$ ergibt (Eigene Abbildung). . . . .	32
3.5 Residual Block: Zwei Faltungsschritte mit Batch-Normalisierung und ReLU, ergänzt durch eine Skip-Verbindung (eigene Darstellung). . . . .	33
3.6 Anzahl MCTS-Simulationen pro Zug in Abhängigkeit der Trainingsiteration (eigene Darstellung) . . . . .	35
3.7 Grafische Benutzeroberfläche in einer frühen Spielsituation. Alle GUI-Komponenten sind sichtbar: das Spielfeld, farbige Policy-Markierungen, die Value- und Entropie-Verläufe sowie die wichtigsten MCTS-Kennzahlen. Die Policy legt hier den Zug C4 nahe, was auch durch die blaue Markierung als zuletzt gespielter Zug angezeigt wird (Eigene Abbildung). . . . .	38

4.1	Verlauf der durchschnittlichen Trainingsverluste (Policy Loss und Value Loss) über alle Generationen des AlphaZero-Trainings, jeweils geglättet zur besseren Lesbarkeit (Eigene Abbildung). . . . .	44
4.2	Gewinnraten der trainierten AlphaZero-Othello-Modelle (jede fünfte Generation) gegen Edax auf den Schwierigkeitsstufen 1–6. Farbskala: Anteil gewonnener Partien in 100 Spielen pro Matchup, gleichmässig variierte Startspielerrolle (Eigene Abbildung). . . . .	45
4.3	Entwicklung der Elo-Bewertung der trainierten AlphaZero-Othello-Modelle über den Trainingsverlauf, ermittelt aus den Partien gegen Edax Level 1–6. Markiert ist der Endwert von Generation 55 ( 1780 Elo) (Eigene Abbildung). . . . .	46
4.4	Normalisierte Zughäufigkeit pro Feld für die Modellgenerationen 5, 15 und 55. Helle Farben markieren häufig gewählte Zugpositionen (Eigene Abbildung). . . . .	46
4.5	Frühe Spielsituation in einer Partie zwischen dem Modell und einem menschlichen Spieler. Das Modell (Weiss) besitzt zu diesem Zeitpunkt lediglich einen einzigen Stein. Die GUI zeigt eine fokussierte Policy, eine positive Value-Schätzung sowie die wichtigsten MCTS-Kennzahlen (Eigene Abbildung). . . . .	48
4.6	Spielsituation im Endspiel: Das Modell (Weiss) dominiert das Zentrum und hat bereits zwei Ecken besetzt. Die GUI zeigt eine eindeutige Value-Schätzung von 1.0 sowie eine neutrale Entropie. Die Policy legt den Fokus klar auf die verbleibenden Ecken (Eigene Abbildung). . . . .	49
4.7	Abschlusssituation der Partie. Das Modell (Weiss) hat alle vier Ecken besetzt und dominiert das Spielfeld klar. Die Value-Schätzung liegt bei 0,99, die Entropie beträgt 0,0 (Eigene Abbildung). . . . .	50

# Kapitel 1

## Einleitung

### 1.1 Vorwort

Im Jahr 2022 stiess ich auf eine Maturaarbeit, in der ein Schüler ein neuronales Netz trainierte, das „Vier gewinnt“ spielte. Diese Arbeit faszinierte mich so sehr, dass mein Interesse an Künstlicher Intelligenz seitdem stetig wuchs. Besonders die Vorstellung, dass ein Computer durch selbstständiges Lernen komplexe Strategien entwickeln kann, hat mich nicht mehr losgelassen. Aus dieser Begeisterung heraus entstand der Wunsch, mich im Rahmen meiner eigenen Maturaarbeit intensiv mit diesem Thema zu beschäftigen und ein solches System selbst zu entwickeln – diesmal für das strategische Brettspiel Othello.

Ziel dieser Arbeit ist es, eine KI nach dem AlphaZero-Prinzip zu entwerfen und zu trainieren, die Othello auf einem hohen spielerischen Niveau beherrscht. Der gesamte Entwicklungs- und Trainingsprozess erfolgte eigenständig und ausschliesslich auf einem Heimrechner – ohne externe Rechenressourcen oder vorgefertigte Trainingsdaten. Damit soll nicht nur die Spielstärke des entwickelten Algorithmus untersucht, sondern auch aufgezeigt werden, wie leistungsfähige KI mit begrenzten technischen Mitteln realisiert werden kann.

Die Arbeit gliedert sich in drei Hauptteile: Zunächst werden die theoretischen Grundlagen erläutert, die für das Verständnis von Othello, neuronalen Netzen und dem AlphaZero-Ansatz notwendig sind. Anschliessend folgt die Beschreibung der Methodik, einschliesslich der technischen Umsetzung und des Trainingsprozesses. Im dritten Teil werden die erzielten Ergebnisse vorgestellt, diskutiert und ein Ausblick auf mögliche Weiterentwicklungen gegeben.

Mein besonderer Dank gilt meiner Betreuungslehrperson, Frau Franziska Flegel, die mich während des gesamten Projekts mit fachlicher Kompetenz, wertvollen Anregungen und grossem Engagement unterstützt hat. Ebenso danke ich meiner Mutter, die mir durch ihre Unterstützung

den Besuch der Kantonsschule Kreuzlingen überhaupt erst ermöglicht hat. Auch den Freunden und Mitschülern, die sich die Zeit genommen haben, mein Modell in Testpartien herauszufordern, danke ich herzlich für ihre Zeit und ihr Feedback.

## 1.2 Fragestellung

Künstliche Intelligenz hat in den letzten Jahren enorme Fortschritte gemacht – insbesondere im Bereich strategischer Spiele, wo Algorithmen wie AlphaZero gezeigt haben, dass maschinelles Lernen und Monte-Carlo-Suchverfahren zu beeindruckender Spielstärke führen können. Diese Entwicklungen werfen die Frage auf, inwiefern solche Methoden auch in vereinfachter Form umgesetzt werden können – etwa auf einem Heimrechner und ohne Zugang zu Grossrechenzentren oder riesigen Datenmengen.

Vor diesem Hintergrund formuliert sich die zentrale Fragestellung dieser Arbeit:

*Wie gut performt ein selbst entwickelter AlphaZero-Algorithmus für Othello im direkten Vergleich zu menschlichen Spielern und Programmen unterschiedlicher Spielstärke?*

## 1.3 Zielsetzung

Ziel dieser Arbeit ist die Entwicklung und Evaluation eines selbstlernenden Spielalgorithmus für Othello, der sich am AlphaZero-Prinzip orientiert. Dabei soll ein neuronales Netz mit Hilfe von Monte-Carlo Tree Search aus selbst gespielten Partien lernen und schrittweise besser werden – ohne externe Trainingsdaten oder menschliches Vorwissen.

Die so entstehende KI wird systematisch auf ihre Spielstärke hin überprüft: durch Matches gegen verschiedene Versionen der Othello-Engine Edax sowie gegen menschliche Spieler. Damit soll nicht nur die Leistungsfähigkeit des Algorithmus bestimmt, sondern auch gezeigt werden, wie weit sich moderne Lernverfahren mit begrenzten technischen Mitteln umsetzen lassen.

Ziel dieser Arbeit ist es auch zu untersuchen, ob leistungsfähige KI-Systeme wie AlphaZero nicht auch ohne Hochleistungsrechner realisierbar sind. Durch intelligente Algorithmen, datenbewusstes Design und optimierte Implementierungen soll ein Modell entstehen, das auf einem handelsüblichen Heim-PC mit professionellen Engines konkurrieren kann.

Dabei steht bewusst ein dezentraler Ansatz im Vordergrund: Die gesamte KI entsteht lokal – ohne Cloud, ohne externes Training, ohne zentrale Datenabhängigkeit. So wird ein Beitrag zur Idee einer ressourcenschonenden, zugänglichen und unabhängigen *KI für jedermann* geleistet – delokalisiert, effizient und transparent.

# Kapitel 2

## Theoretische Grundlagen

### 2.1 Othello

#### 2.1.1 Spielregeln

Othello ist ein strategisches Brettspiel für zwei Personen, das üblicherweise auf einem  $8 \times 8$  Felder grossen Spielbrett gespielt wird. Die Partie beginnt mit einer festen Anfangsaufstellung, wie in Abbildung 2.1 dargestellt: Je zwei schwarze und zwei weisse Steine befinden sich diagonal zueinander im Zentrum des Brettes. Die nachfolgenden Ausführungen zu den Spielregeln orientieren sich an „Rules & Strategy - HKOA“ (2022).

Die Person mit den schwarzen Steinen beginnt. Sie muss einen Stein so setzen, dass mindestens ein gegnerischer (weisser) Stein zwischen zwei eigenen Steinen eingeklemmt wird – dies kann horizontal, vertikal oder diagonal geschehen. Ausgehend von der Startposition hat die Person mit den schwarzen Steinen vier mögliche gültige Züge wie in Abbildung 2.2 mit transparenten Spielsteinen dargestellt.

Nachdem die Person mit den schwarzen Steinen einen gültigen Zug ausgeführt hat, werden alle dabei eingeklemmten weissen Steine umgedreht und sind fortan ebenfalls schwarz wie in Abbildung 2.3 zu sehen ist.

Anschliessend ist die Person mit den weissen Steinen am Zug. Auch sie muss einen Stein so setzen, dass mindestens ein gegnerischer Stein eingeschlossen und dadurch

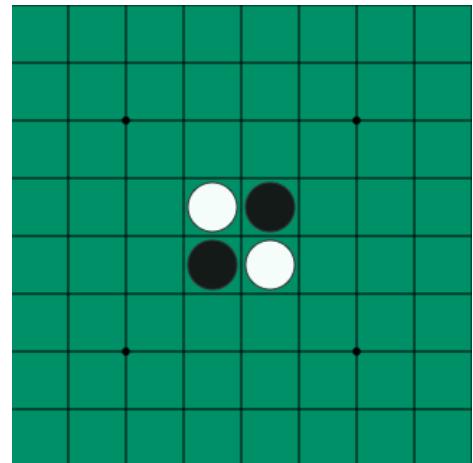


Abbildung 2.1: Startaufstellung einer Othellopartie. („Rules & Strategy - HKOA“, 2022).

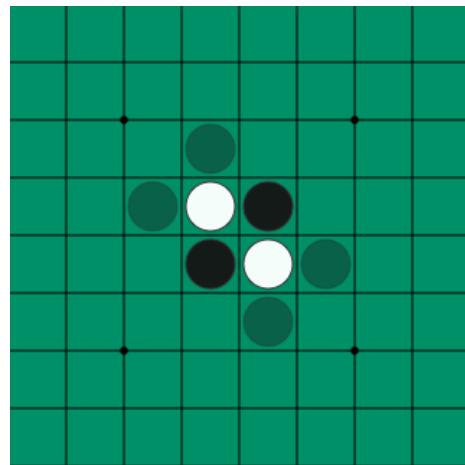


Abbildung 2.2: Mögliche Züge von Schwarz aus der Startaufstellung. („Rules & Strategy - HKOA“, 2022).

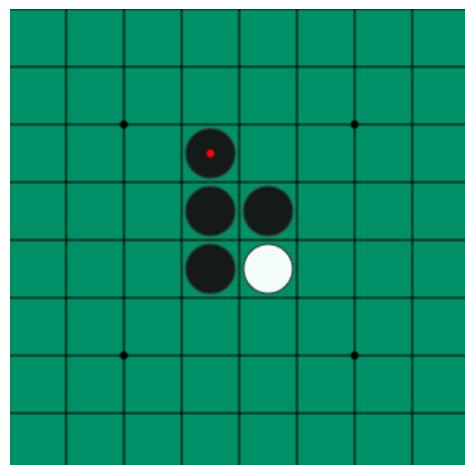
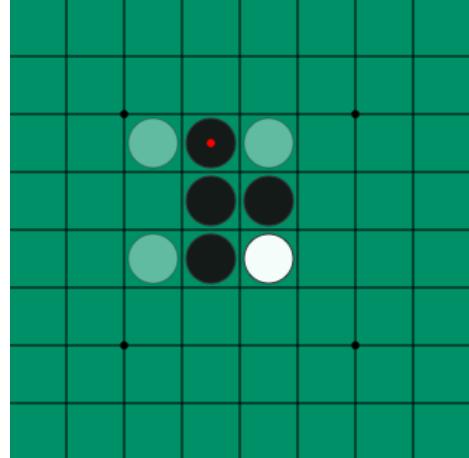


Abbildung 2.3: Spielbrett nach gültigem Zug von Schwarz und umgedrehten Steinen. („Rules & Strategy - HKOA“, 2022).

umgedreht wird. Die Spielregeln bleiben für beide spielenden Personen identisch.

Die spielenden führen abwechselnd ihre Züge aus. Kann eine spielende Person keinen gültigen Zug ausführen, so muss sie aussetzen und der Gegner oder Gegnerin ist erneut an der Reihe. Das Spiel endet, wenn keiner der beiden spielenden Personen mehr einen gültigen Zug machen kann oder das Spielbrett vollständig besetzt ist. Gewonnen hat die Person, die am Ende die meisten Steine seiner Farbe auf dem Brett hat.



### 2.1.2 Othello in Zahlen

Othello ist ein äusserst komplexes Spiel, das selbst moderne Computer vor erhebliche Herausforderungen stellt. Schätzungen zufolge gibt es rund  $10^{28}$  mögliche Spielfeldkonfigurationen. Die Spielbaum-Komplexität, also die Anzahl potenzieller kompletter Spielverläufe, wird sogar auf etwa  $10^{58}$  Spielfeldkonfigurationen geschätzt (Takizawa, 2024; Wikipedia contributors, 2025a).

Diese enorme Komplexität ist einer der Gründe, warum Othello bis heute noch nicht vollständig gelöst ist. Ein Spiel gilt als vollständig gelöst, wenn für jede legale Stellung bekannt ist, welches Ergebnis bei perfektem Spiel beider Seiten eintritt, und man für jede dieser Stellungen eine perfekte Strategie angeben kann, die zu diesem Ergebnis führt – unabhängig davon, was der Gegner macht.

Für Othello wird derzeit angenommen, dass eine perfekte Partie bei optimalem Spiel beider Seiten in einem Unentschieden endet – ein abschliessender Beweis dafür steht jedoch noch aus („Rules & Strategy - HKOA“, 2022).

Abbildung 2.4: Mögliche Spielzüge von Weiss. („Rules & Strategy - HKOA“, 2022).

### 2.1.3 Strategien

Trotz der Einfachheit der Regeln, ist die Meisterung des Spiels schwierig. Dennoch gibt es bewährte Spieltechniken und Strategien nach „Reversi Online – Tipps & Tricks für den Sieg gegen den Computer“ (2024), die verfolgt werden können, um die Qualität des Spiels zu verbessern.

#### Ecken und stabile Steine

Aufgrund der Spielregeln können einmal in den Ecken platzierte Steine nicht mehr umgedreht werden. Sie gelten daher als *stabil*. Solche Steine sind aus taktischer Sicht besonders wertvoll,

da sie als Ausgangspunkt für den Aufbau weiterer stabiler Strukturen dienen können. Besonders entlang der Kanten lassen sich von diesen Eckpunkten aus oft stabile Reihen aufbauen.

Abbildung 2.5 zeigt ein Beispiel für einen stabilen Eckstein in der unteren rechten Ecke (h8). Dieser schwarze Stein kann durch keine zukünftige Spielkonstellation mehr umgedreht werden. Auch die angrenzenden Steine entlang der Kante sind teilweise schwer angreifbar und können sich in weiterer Folge ebenfalls als stabil erweisen.

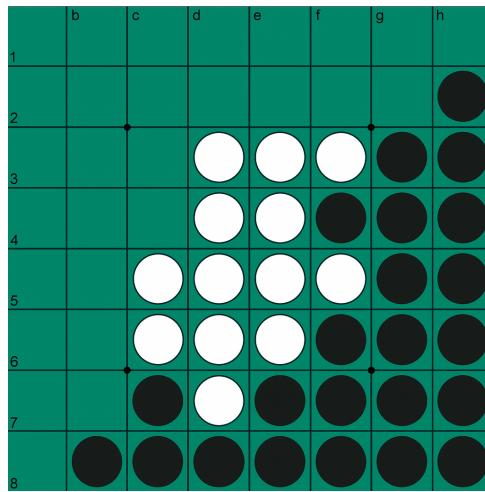


Abbildung 2.5: Stabiler schwarzer Stein in der Ecke (h8). Aufgrund der Spielregeln kann dieser nicht mehr umgedreht werden. Auch angrenzende Steine entlang der Kante können potenziell stabil sein. („Rules & Strategy - HKOA“, 2022).

### Unsichere Felder

Nicht alle Positionen auf dem Othellobrett sind gleichwertig – manche gelten als besonders unsicher. Eine bewährte Strategie ist es, Steine bevorzugt im zentralen 4×4-Bereich zu platzieren. Diese Positionen bieten gute Möglichkeiten, gegnerische Steine später einzuklemmen und langfristig Kontrolle über das Zentrum aufzubauen.

Dagegen sind Züge auf den Feldern unmittelbar angrenzend an die Ecken besonders riskant. Wer diese Positionen frühzeitig besetzt, läuft Gefahr, dem Gegner oder der Gegnerin den Zugang zu den strategisch enorm wertvollen Ecken zu erleichtern.

Abbildung 2.6 illustriert diesen Unterschied: Die weissen Steine im Zentrum repräsentieren bevorzugte (sichere) Felder, während die dunklen Steine in den sogenannten X-Feldern (z.B. b2, g2, b7, g7) als strategisch schwach gelten. („Rules & Strategy - HKOA“, 2022)

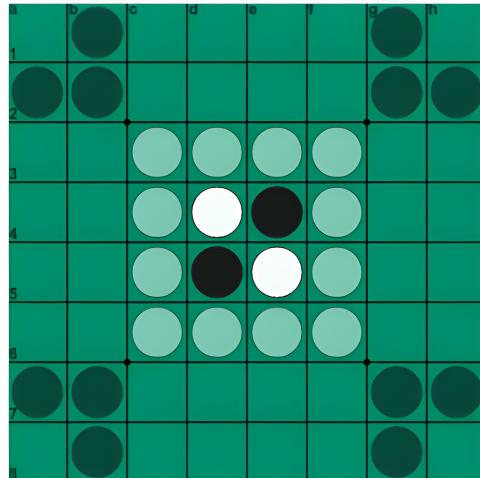


Abbildung 2.6: Bevorzugte Felder im zentralen  $4 \times 4$ -Bereich (weiss) und riskante Felder angrenzend an die Ecken (dunkel hervorgehoben). („Rules & Strategy - HKOA“, 2022).

## Dynamik

Ein häufiger Anfängerfehler besteht darin, bereits früh in der Partie möglichst viele Steine umzudrehen. Effektiver ist jedoch eine Strategie, die darauf abzielt, die Anzahl der möglichen Züge der Gegenspielerin oder des Gegenspielers gezielt zu begrenzen. Paradoxe Weise lässt sich dies oft dadurch erreichen, dass man selbst möglichst wenige eigene Steine auf dem Brett hält – zumindest in der frühen und mittleren Phase des Spiels.

Abbildung 2.7 zeigt eine scheinbar ungewöhnliche, aber taktisch effektive Spielsituation: Obwohl am Ende nur ein einziger weisser Stein auf dem Brett liegt, hat die weisse Partei das Spiel gewonnen. Dieses Beispiel verdeutlicht, dass kurzfristiger Scheingewinn (z.B. durch das frühe Umdrehen vieler Steine) langfristig nachteilig sein kann – und dass der kontrollierte Verzicht auf Dominanz oft der Schlüssel zum Sieg ist.

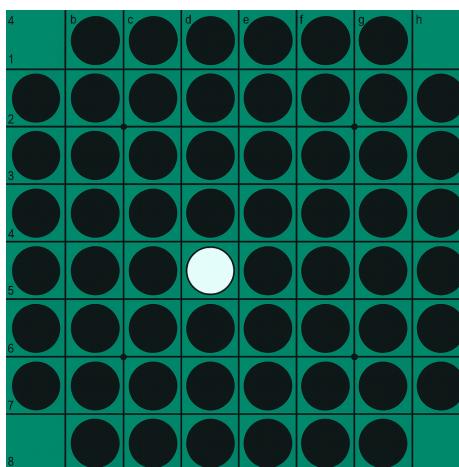


Abbildung 2.7: Taktisch erfolgreiche Spielsituation: Die weisse Partei gewinnt mit nur einem verbliebenen Stein auf dem Feld. („Rules & Strategy - HKOA“, 2022).

## 2.2 Einführung in neuronale Netze

Neuronale Netze sind ein zentrales Element moderner Künstlicher Intelligenz. Sie ermöglichen es, komplexe Zusammenhänge in Daten zu erkennen und fundierte Entscheidungen zu treffen, ohne dass diese explizit in Form fester Regeln programmiert werden müssen. In diesem Abschnitt wird die Funktionsweise neuronaler Netze anschaulich erläutert und ihre Rolle im AlphaZero-Ansatz für Othello dargestellt. Die beigefügten Abbildungen unterstützen den Text visuell.

### 2.2.1 Warum neuronale Netze?

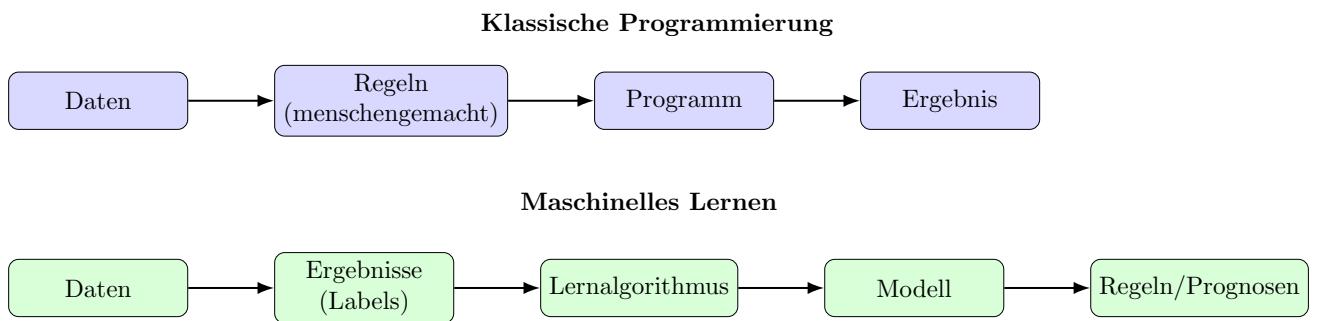


Abbildung 2.8: Klassische Programmierung (oben) im Vergleich zu maschinellem Lernen (unten) (Eigene Abbildung).

Klassische Programme arbeiten nach dem Prinzip: *Daten + Regeln* → *Ergebnis*. Die Regeln werden dabei vom Menschen formuliert und müssen alle denkbaren Situationen abdecken. In komplexen Umgebungen wie Othello stösst dieser Ansatz jedoch schnell an Grenzen, da die Anzahl möglicher Spielstellungen astronomisch hoch ist und sich nicht für jede Situation eine feste Regel definieren lässt.

Neuronale Netze bieten eine Alternative: Statt Regeln manuell vorzugeben, lernen sie diese aus Beispielen. Sie erkennen eigenständig Muster, die für eine gute Entscheidung sprechen, und übertragen dieses Wissen auf neue, unbekannte Situationen (Perrotta, 2020, p. 10). Abbildung 2.8 veranschaulicht diesen grundlegenden Unterschied zwischen klassischer Programmierung und maschinellem Lernen.

### 2.2.2 Grundidee eines neuronalen Netzes

Ein neuronales Netz besteht aus einer *Eingabeschicht*, einer oder mehreren *versteckten Schichten* und einer *Ausgabeschicht*. Jede Schicht besteht aus künstlichen Neuronen, die einfache Berechnungen durchführen: Eingaben werden gewichtet, aufsummiert und anschliessend durch eine nichtlineare *Aktivierungsfunktion* transformiert.

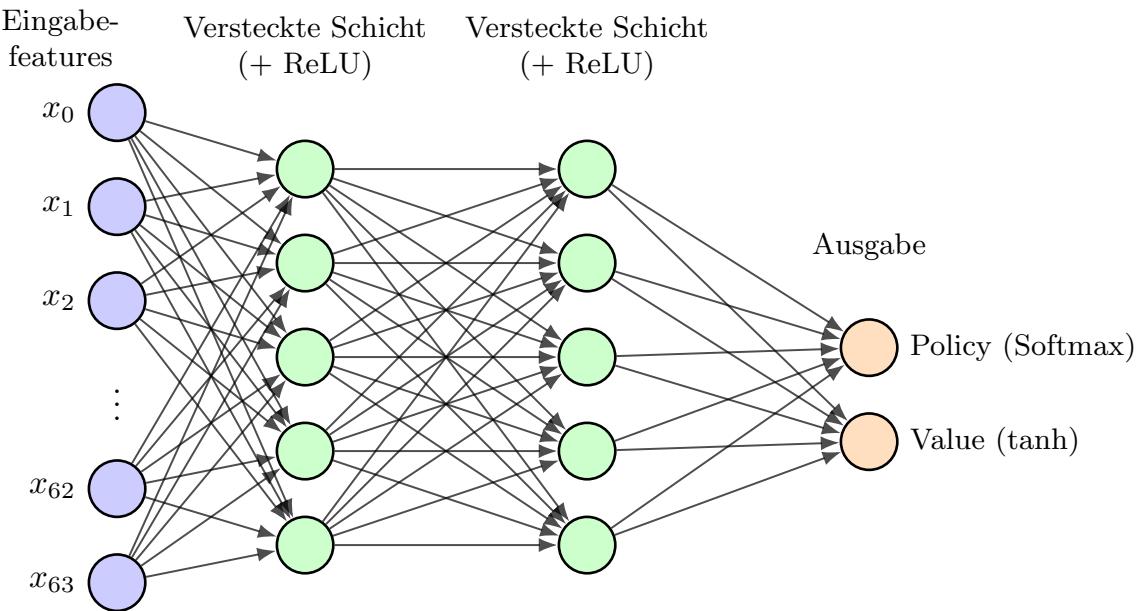


Abbildung 2.9: Schematische Darstellung eines Feedforward-Netzes für Othello mit separaten Policy- (Softmax) und Value-Ausgaben (tanh) (Eigene Abbildung).

Abbildung 2.9 zeigt schematisch den Aufbau eines solchen Netzes am Beispiel des in dieser Arbeit verwendeten Othello-Modells. Die Eingabeschicht ( $x_0$  bis  $x_{63}$ ) entspricht dabei den 64 Feldern des Othello-Bretts. Jedes Feld ist eine einzelne EingabevARIABLE und kodiert den Zustand des Feldes:

$$x_i = \begin{cases} 1 & \text{wenn ein Stein des aktuellen Spielers liegt} \\ -1 & \text{wenn ein Stein des Gegners liegt} \\ 0 & \text{wenn das Feld leer ist} \end{cases}$$

Die schwarzen Pfeile in der Abbildung stellen die *Gewichte* ( $w_{ij}$ ) dar. Diese Gewichte sind einfache Zahlen, vergleichbar mit dem Steigungsparameter  $m$  in einer linearen Funktion  $y = m \cdot x + b$ . Sie bestimmen, wie stark und in welcher Richtung eine Eingabe den Wert des nächsten Neurons beeinflusst. Beispielsweise kann ein Gewicht eine Eingabe verdoppeln ( $w = 2$ ), halbieren ( $w = 0.5$ ) oder sogar das Vorzeichen umkehren ( $w = -1$ ).

Die Berechnung innerhalb eines Neurons lässt sich mathematisch so ausdrücken:

$$z_j = \sum_i w_{ij} \cdot x_i + b_j$$

Der zusätzliche Parameter  $b_j$ , der *Bias*, ist vergleichbar mit dem Achsenabschnitt  $b$  in einer linearen Funktion. Er erlaubt es, den Ausgangswert eines Neurons unabhängig von seinen Eingaben nach oben oder unten zu verschieben. Dies verleiht dem Netz zusätzliche Flexibilität und

ermöglicht es, Muster darzustellen, die nicht allein durch die gewichteten Eingaben abgebildet werden können.

Nach dieser linearen Transformation wird der Wert  $z_j$  durch eine nichtlineare *Aktivierungsfunktion* umgeformt. In den versteckten Schichten kommt hier die *Rectified Linear Unit* (ReLU) (Wikipedia contributors, 2025b) zum Einsatz:

$$\text{ReLU}(z) = \max(0, z)$$

Diese nichtlineare Abbildung ermöglicht es dem Netzwerk, komplexe Muster zu erfassen, die über rein lineare Zusammenhänge hinausgehen.

In der Ausgabeschicht werden zwei verschiedene Aktivierungsfunktionen verwendet:

- **Policy-Head:** Softmax-Funktion zur Erzeugung einer Wahrscheinlichkeitsverteilung über alle möglichen Züge:

$$\text{Softmax}(z_i) = \frac{e^{z_i}}{\sum_j e^{z_j}}$$

- **Value-Head:** Tangens-Hyperbolicus ( $\tanh$ ), um den erwarteten Spielausgang als Zahl zwischen  $-1$  (sichere Niederlage) und  $+1$  (sicherer Sieg) darzustellen:

$$\tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$$

Zusammengefasst lässt sich das Netz so verstehen: Die Eingabeschicht liest das Brett aus, die Gewichte und Bias-Werte verändern und kombinieren diese Informationen, und die Aktivierungsfunktionen erlauben es dem Modell, auch komplexe, nichtlineare Zusammenhänge zu lernen. Dieser Prozess wiederholt sich über die versteckten Schichten, bis am Ende die Policy- und Value-Ausgaben berechnet werden (3Blue1Brown, 2017a; Redaktion, 2024).

### 2.2.3 Lernprozess eines neuronalen Netzes

Zu Beginn sind die Gewichte und Bias-Werte des Netzes zufällig initialisiert – das Modell besitzt also noch keinerlei Wissen. Um dieses Wissen schrittweise zu erwerben, durchläuft das Netz einen wiederkehrenden Lernzyklus, der in Abbildung 2.10 dargestellt ist.

1. **Vorwärtsdurchlauf:** Die Eingabedaten (*Features*) werden – wie in Abschnitt 2.2.2 beschrieben – durch die einzelnen Schichten des Netzes verarbeitet. Dabei werden die gewichteten Summen und Aktivierungsfunktionen berechnet, bis am Ende die Ausgaben (Policy und Value) entstehen.
2. **Fehlerberechnung:** Die erzeugte Vorhersage wird mit dem bekannten Zielwert verglichen.

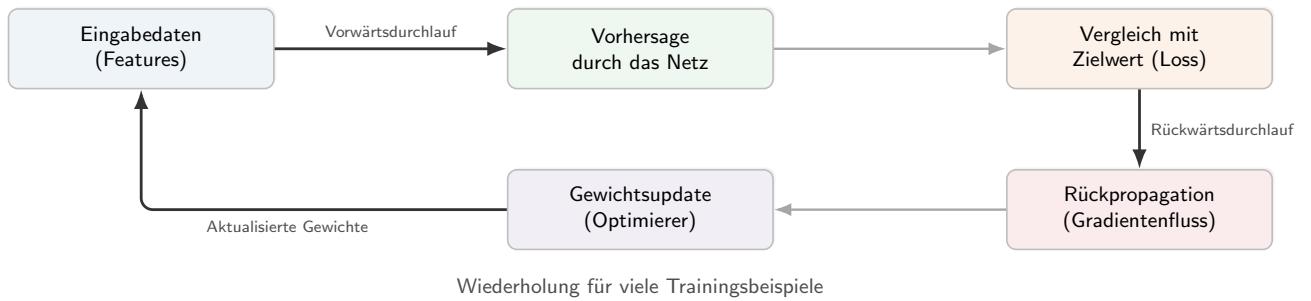


Abbildung 2.10: Lernzyklus eines neuronalen Netzes – bestehend aus Vorwärtsdurchlauf, Fehlerberechnung, Rückpropagation und Gewichtsupdate (Eigene Abbildung).

chen. Aus der Differenz wird ein *Loss* (Fehlerwert) berechnet, der quantifiziert, wie stark die Vorhersage vom Zielwert abweicht (siehe Abschnitt 2.1, in dem die verwendete Loss-Gleichung erläutert wird).

3. **Rückwärtssdurchlauf:** Auf Basis des Loss wird mit dem Verfahren der *Rückpropagation* bestimmt, welchen Einfluss jedes Gewicht und jeder Bias auf den Fehler hatte. Mathematisch geschieht dies über die Kettenregel der Differentialrechnung, sodass für jede Verbindung im Netz ein *Gradient* berechnet wird, der die Richtung und Grösse der notwendigen Anpassung angibt.
4. **Optimierung:** Ein Optimierungsverfahren aktualisiert die Gewichte und Bias-Werte anhand der berechneten Gradienten. Ziel ist es, den Loss bei der nächsten Iteration zu verringern.

Dieser Prozess wiederholt sich für viele tausend Beispiele, wodurch das Netz schrittweise lernt, seine Ausgaben zu verbessern und immer präzisere Vorhersagen zu treffen (3Blue1Brown, 2017b).

## 2.2.4 Convolutional Neural Networks (CNNs)

Da ein Othello-Brett eine feste räumliche Struktur aufweist, eignet sich ein *Convolutional Neural Network* (CNN) besonders gut, um lokale Muster zu erkennen. Im Gegensatz zu vollständig verbundenen Netzen, bei denen jedes Neuron mit allen Eingaben verknüpft ist, arbeiten CNNs mit *Faltungsschichten* (Convolutions). Dabei gleitet ein kleiner Filter (Kernel) systematisch über die Eingabe und berechnet für jede Position eine gewichtete Summe der umliegenden Werte.

Das Ergebnis ist eine *Feature Map*, die gezielt bestimmte Merkmale hervorhebt – in Othello können dies z. B. besetzte Ecken, strategisch wichtige Kanten oder spezifische Steinmuster sein. Mehrere Filter arbeiten parallel, sodass verschiedene Merkmale gleichzeitig extrahiert werden. Abbildung 2.11 zeigt schematisch eine Faltungsoperation: Links befindet sich ein Eingabebild

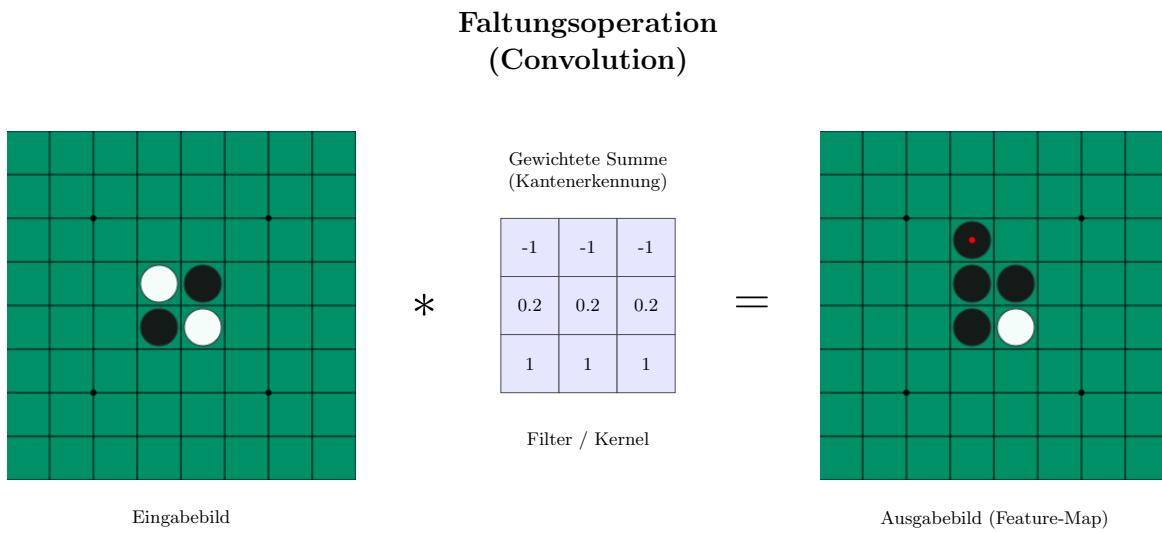


Abbildung 2.11: Funktionsweise einer Faltungsschicht: Ein Filter (Kernel) berechnet für jede Position eine gewichtete Summe der Nachbarschaft, woraus eine Feature Map entsteht (Eigene Abbildung).

(im Fall von Othello z. B. eine Brettstellung), daneben ein  $3 \times 3$ -Filter mit definierten Gewichtungen. Durch die *gewichtete Summe* der überlappenden Bereiche entsteht rechts die Feature Map, in diesem Beispiel als Zentrumswandlung dargestellt (codebasics, 2020; Mishra, 2020).

## 2.2.5 Residual Blocks

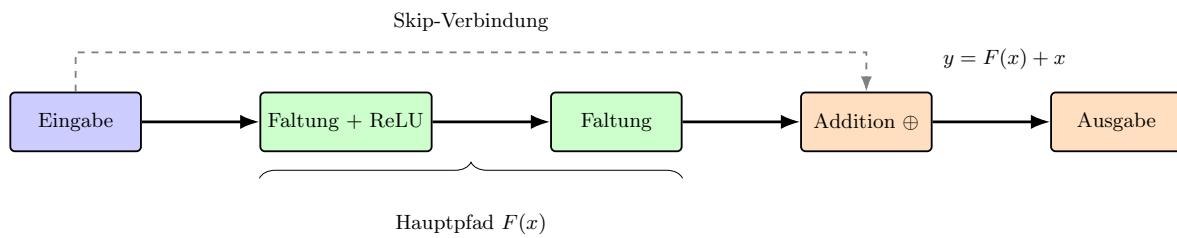


Abbildung 2.12: Aufbau eines Residual Blocks: Der Hauptpfad  $F(x)$  enthält zwei Faltungsschichten, die Skip-Verbindung gibt die Eingabe  $x$  direkt weiter (Eigene Abbildung).

Um CNNs tiefer zu machen und gleichzeitig stabil zu trainieren, werden *Residual Blocks* eingesetzt. Ein Residual Block besteht aus einem *Hauptpfad*  $F(x)$  mit mehreren Faltungsschichten und einer *Skip-Verbindung*, die die Eingabe  $x$  unverändert zur Ausgabe weiterleitet. Am Ende werden die beiden Signale addiert:

$$y = F(x) + x$$

Diese Struktur erleichtert das Training, weil Informationen und Gradienten aus früheren Schichten

ten direkt weitergegeben werden. Dadurch wird das *Vanishing-Gradient*-Problem abgemildert und das Netz kann tiefer werden, ohne dass die Leistung in frühen Schichten „verloren geht“ (Amanatullah, 2023).

In Abbildung 2.12 sind die grünen Kästen („Faltung + ReLU“) im Hauptpfad mit den versteckten Schichten eines klassischen Netzes vergleichbar. Der Unterschied: Hier wird die räumliche Struktur durch Faltungen erhalten und am Ende per Skip-Verbindung mit der ursprünglichen Eingabe kombiniert. So kann der Block lernen, nur die notwendigen Änderungen (Residuals) zur Eingabe zu berechnen, anstatt alles neu zu erzeugen (Deep Learning with Yacine, 2024; He et al., 2016; Sharma, 2021).

## 2.3 Monte Carlo Tree Search

Monte Carlo Tree Search (MCTS) ist ein Suchalgorithmus, der insbesondere in Spielen mit hoher Kombinationsvielfalt eingesetzt wird (Browne et al., 2012a). Im Gegensatz zu klassischen, deterministischen Suchmethoden basiert MCTS auf zufälliger Simulation, um Entscheidungen zu treffen. Der Algorithmus kombiniert die Vorteile einer gezielten Baumsuche mit statistischer Abschätzung und ist dadurch besonders geeignet für Brettspiele wie Schach, Go oder Othello, bei denen die Anzahl möglicher Züge exponentiell ansteigt.

Die grundlegende Datenstruktur in MCTS ist der *Knoten*, der einen konkreten Spielzustand repräsentiert. Bevor die Arbeitsweise des Algorithmus beschrieben wird, wird daher zunächst erläutert, wie ein solcher Knoten aufgebaut ist und welche Informationen er enthält. Anschließend werden die vier Phasen von MCTS – *Selektion*, *Expandierung*, *Simulation* und *Rückpropagierung* – im Detail dargestellt (Coulom, 2006; Kocsis & Szepesvári, 2006; MarbleScience, 2020; Silver, Hubert et al., 2017b).

### 2.3.1 Knoten

Ein Knoten in einem Monte Carlo Tree repräsentiert einen konkreten Spielzustand – im Fall dieser Arbeit also eine spezifische Konfiguration des Othello-Spielbretts. Jeder Knoten enthält nicht nur Informationen über den Zustand und die Zugfolge, sondern auch Abschätzungen, darunter Spielstandbewertung und Zugwahrscheinlichkeit, die mithilfe eines neuronalen Netzwerks generiert wurden (siehe Abschnitt 2.2). Diese erweitern die klassische MCTS-Logik um eine initiale Priorisierung (*Policy*) und eine Bewertung (*Value*) des Zustands. Ein AlphaZero-MCTS-Knoten besteht typischerweise aus folgenden Attributen (Josh Varty, 2020):

- **Spielzustand**  $s \in \mathcal{S}$ : Repräsentiert das aktuelle Spielbrett (z. B. als Array)
- **Aktion**  $a \in \mathcal{A}(s)$ : Die letzte Aktion, die zu diesem Spielzustand geführt hat

- **Elternknoten:** Referenz auf den Vorgängerknoten (bei der Wurzel None)
- **Kinder**  $C = \{(a_i, s_i)\}$ : Menge der Kindknoten und der von ihnen repräsentierten Spielzustände  $s_i$
- **Spieler**  $p \in \{-1, 1\}$ : Aktiver Spieler im aktuellen Spielzustand
- **Besuche**  $N(s, a) \in \mathbb{N}$ : Anzahl der Simulationen entlang einer Kante  $(s, a)$
- **Wertsumme**  $W(s, a) \in \mathbb{R}$ : Summe aller vom neuronalen Netzwerk gelieferten Value-Schätzungen für die in den Simulationen entlang  $(s, a)$  besuchten Spielzustände, aus Sicht des aktuellen Spielers
- **Priorität**  $P(s, a) \in [0, 1]$ : Von der Policy des neuronalen Netzwerks gelieferte Wahrscheinlichkeit für Aktion  $a$
- **Value-Schätzung**  $V(s) \in [-1, 1]$ : Vom Netzwerk gelieferte Bewertung des Spielzustands  $s$ , aus Sicht des aktuellen Spielers

### 2.3.2 Selektion

In der Selektionsphase bestimmt der Algorithmus, welcher Folgespielzustand  $s_{\text{next}}$  aus der Menge der möglichen Aktionen  $a \in \mathcal{A}(s)$  vom aktuellen Spielzustand  $s$  weiterverfolgt werden soll. AlphaZero nutzt hierzu eine durch das neuronale Netzwerk gelieferte Priorverteilung  $P(s, a)$ , um diese Auswahl systematisch zu steuern.

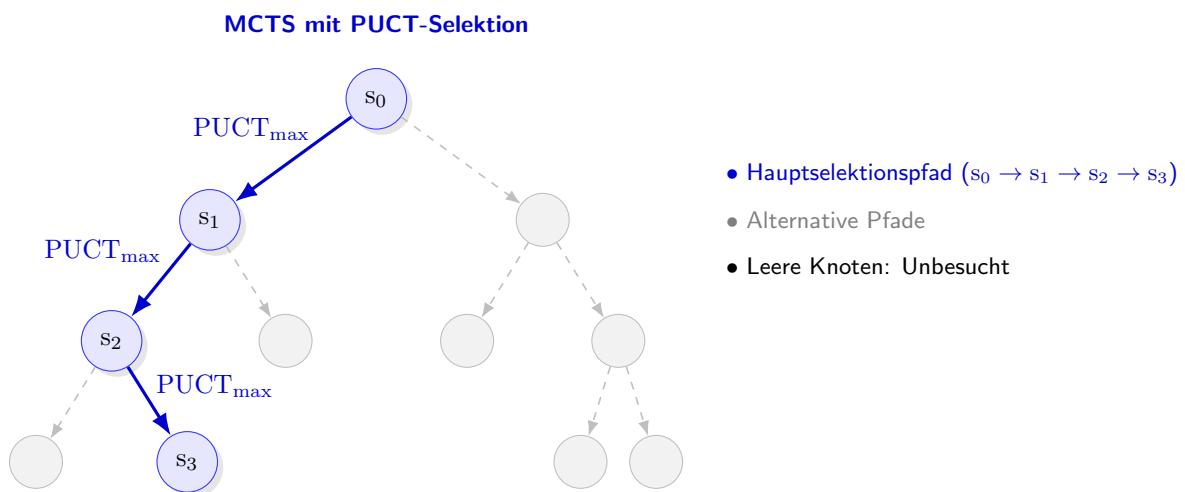


Abbildung 2.13: Visualisierung der Selektionsphase: Ausgehend vom Wurzelknoten  $s_0$  wird schrittweise der Nachfolger mit dem höchsten PUCT-Wert (blau markierter Hauptpfad) gewählt, bis ein Blattknoten erreicht wird. Alternative Pfade sind gestrichelt dargestellt, unbesuchte Knoten bleiben leer (Eigene Abbildung).

Wie in Abbildung 2.13 zu sehen, beginnt die Selektion am Wurzelknoten  $s_0$  und folgt iterativ der Kante mit dem höchsten Bewertungswert  $PUCT(s, a)$ , bis ein Blattknoten ohne Kindknoten erreicht wird. Dabei sind im Diagramm der Hauptselektionspfad  $s_0 \rightarrow s_1 \rightarrow s_2 \rightarrow s_3$  blau hervorgehoben, während alternative, in diesem Simulationsdurchlauf nicht gewählte Kanten grau dargestellt sind.

Für jede Aktion wird der PUCT-Wert wie folgt berechnet:

$$a^* = \arg \max_{a \in \mathcal{A}(s)} \left( Q(s, a) + c_{puct} \cdot P(s, a) \cdot \frac{\sqrt{N(s)}}{1 + N(s, a)} \right)$$

### Erklärung der Terme:

$Q(s, a) = \frac{W(s, a)}{N(s, a)}$	Mittlerer Wert der bisherigen Simulationen entlang $(s, a)$
$P(s, a) \in [0, 1]$	Priorwahrscheinlichkeit für Aktion $a$ , vom Netzwerk geliefert
$N(s) \in \mathbb{N}$	Gesamtanzahl der Besuche des Knotens, der $s$ repräsentiert
$N(s, a) \in \mathbb{N}$	Anzahl der Besuche der Kante $(s, a)$
$c_{puct} > 0$	Explorationskonstante zur Gewichtung der Priorität

Der Ausdruck  $Q(s, a)$  steht für die *Ausnutzung* der bisher erfolgreich getesteten Aktionen, während der zweite Term die *explorative* Gewichtung von laut Netzwerk vielversprechenden, aber noch wenig untersuchten Aktionen übernimmt. Dadurch entsteht ein balancierter Kompromiss zwischen bewährten und potentiell unterschätzten Zügen. Sobald ein *Blattknoten* erreicht wird, dessen Spielzustand wir mit  $s_{\text{next}}$  bezeichnen, wechselt der Algorithmus zur nächsten Phase, der *Expandierung* (Coulom, 2006; Silver, Schrittwieser et al., 2017b).

### 2.3.3 Expandierung

Sobald ein bisher nicht expandierter Blattknoten erreicht wird, dessen Spielzustand wir hier mit  $s_3$  bezeichnen, beginnt die **Expandierungsphase**. Wie in Abbildung 2.14 dargestellt, werden an diesem Punkt neue Kindknoten erzeugt, die jeweils mögliche Folgespielzustände  $s_{\text{next}}$  repräsentieren. In der Abbildung ist der zuvor durch die Selektion bestimmte Pfad weiterhin blau hervorgehoben, während die neu expandierten Kindknoten  $s_{1,\text{next}}$  und  $s_{2,\text{next}}$  grün dargestellt sind. Leere Knoten symbolisieren Zustände, die im aktuellen Simulationsdurchlauf noch nicht besucht wurden.

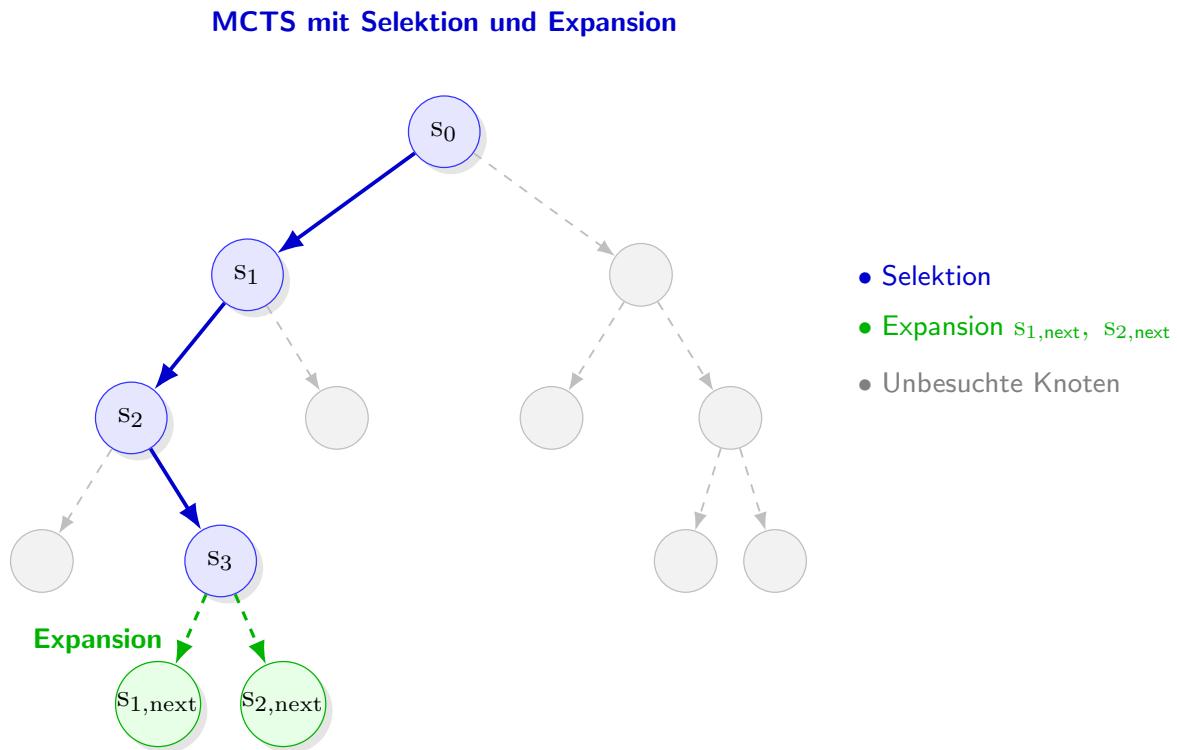


Abbildung 2.14: Visualisierung der Expandierungsphase: Nach Erreichen eines Blattknotens  $s_3$  werden neue Kindknoten für alle legalen Aktionen angelegt (grün markiert). Der Selektionspfad ist blau dargestellt, unbesuchte Knoten bleiben leer (Eigene Abbildung).

Der Algorithmus verwendet nun das neuronale Netzwerk mit  $s_3$  als Eingabe und erhält zwei Ausgaben:

- eine **Policy-Verteilung**  $P(s, a)$  über alle legalen Aktionen  $a \in \mathcal{A}(s)$ , und
- eine **Value-Schätzung**  $V(s) \in [-1, 1]$ , die den erwarteten Spielausgang aus Sicht des aktuellen Spielers angibt.

Anschliessend wird für jede legale Aktion  $a \in \mathcal{A}(s)$  ein neuer Kindknoten angelegt. Dieser Eintrag in der Kindliste enthält:

- den Folgespielzustand  $s_{\text{next}}$ , der durch Anwendung von  $a$  auf  $s$  entsteht,
- die Prioritätsinformation  $P(s, a)$ , also die vom Netzwerk gelieferte Wahrscheinlichkeit für diese Aktion,
- initiale Zählwerte  $N(s, a) = 0$  und  $W(s, a) = 0$  für folgende  $PUCT(s, a)$ -Berechnungen.

Die Value-Schätzung  $V(s)$  wird in der anschliessenden **Rückpropagierungsphase** zur Aktualisierung der Pfadstatistiken verwendet (Coulom, 2006; Silver, Schrittwieser et al., 2017b).

### 2.3.4 Rückpropagierung

Bei der Rückpropagierung nutzt der Algorithmus die vom neuronalen Netzwerk geschätzte Value-Schätzung  $V(s)$ , wobei  $s$  der Spielzustand ist, der im zuletzt ausgewählten Blattknoten gespeichert ist. Wie in Abbildung 2.15 dargestellt, wird dieser Wert  $v := V(s)$  entlang des während der Selektion durchlaufenen Pfades bis zur Wurzel propagiert. In der Abbildung sind die betroffenen Knoten  $s_1, s_2, s_3$  entlang des blauen Selektionspfades markiert, während rote Pfeile den Fluss des Wertes  $v$  (und bei jedem Schritt den negierten Wert  $-v$ ) verdeutlichen. Unbesuchte Knoten bleiben leer (Browne et al., 2012b; Silver, Schrittwieser et al., 2017b).

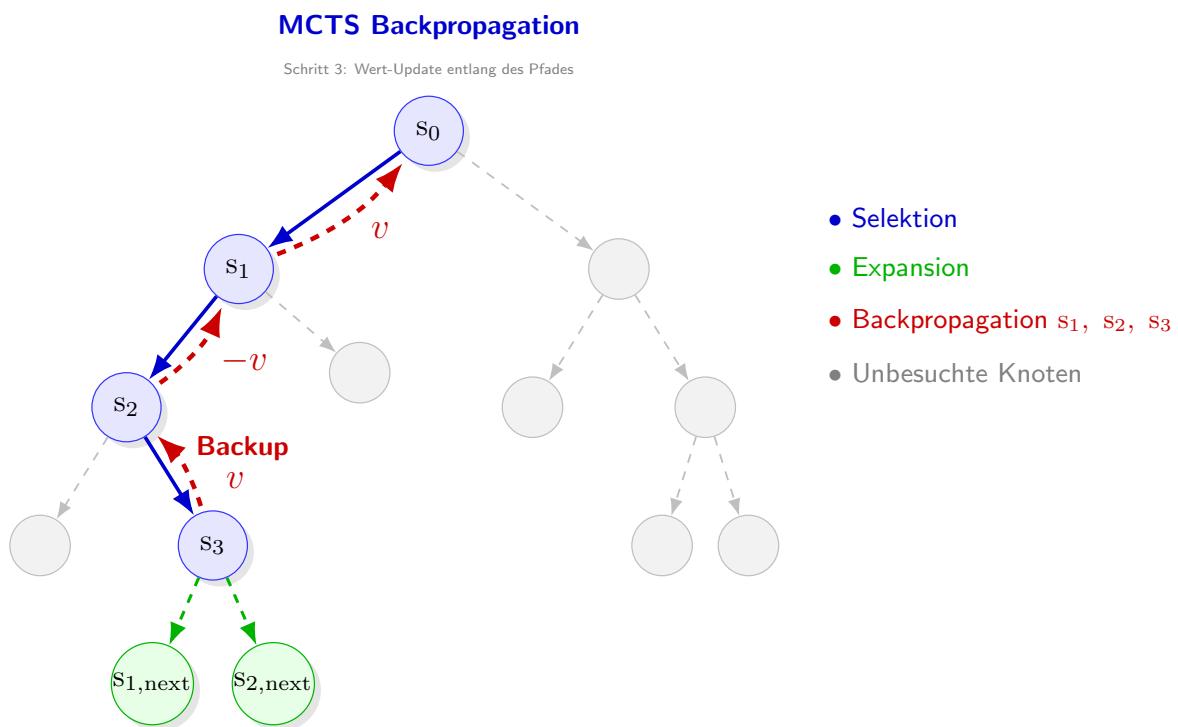


Abbildung 2.15: Visualisierung der Rückpropagierungsphase: Der vom Blattknoten stammende Wert  $v$  wird entlang des Selektionspfades bis zur Wurzel zurückgegeben. Rote Pfeile zeigen den Wertfluss, wobei bei jedem Schritt die Perspektive durch Negation gewechselt wird (Eigene Abbildung).

Für jeden besuchten Spielzustand  $s_t$  (im jeweiligen Knoten) und die zugehörige Aktion  $a_t$  werden die folgenden Statistiken aktualisiert:

- $N(s_t, a_t) += 1$ : Die Anzahl der Besuche für die Aktion  $a_t$  in Zustand  $s_t$  wird um 1 erhöht.
- $W(s_t, a_t) += v$ : Die kumulierte Bewertung wird um den Wert  $v$  ergänzt.

Da der Wert  $v$  die Perspektive des Spielers im Endzustand  $s$  widerspiegelt, wird er bei jedem

Schritt entlang des Pfades invertiert, um zwischen den Spielern zu wechseln:

$$v \leftarrow -v$$

### 2.3.5 Ausführung

Im Folgenden wird ein vollständiger Durchlauf der drei zentralen Phasen des MCTS – **Selektion**, **Expandierung** und **Rückpropagierung** – als eine einzelne *Simulation* bezeichnet. MCTS ist ein sogenannter *Anytime*-Algorithmus: Seine Laufzeit ist nicht fest vorgegeben, sondern kann je nach verfügbarer Rechenzeit flexibel angepasst werden. In der Praxis wird MCTS typischerweise entweder mit einer festen Anzahl an *Simulationen* oder innerhalb eines festgelegten Zeitbudgets (z. B. eine Sekunde pro Zug) ausgeführt.

Die zentrale Idee dabei ist: Je mehr *Simulationen* durchgeführt werden, desto zuverlässiger wird die Bewertung der möglichen Spielzüge. Die Qualität der getroffenen Entscheidung hängt daher direkt von der Anzahl der durchgeföhrten Simulationen ab (Browne et al., 2012a).

### 2.3.6 Zugextraktion

Nach Abschluss der vorgegebenen Anzahl an Simulationen (z. B. 800 pro Zug) ist der MCTS-Baum für den aktuellen Zustand  $s$  aufgebaut und enthält statistische Informationen zu allen betrachteten Aktionen. Nun wird aus der Menge der möglichen Züge derjenige extrahiert, der ausgeführt werden soll.

In AlphaZero hängt die Auswahlstrategie vom Spielmodus ab:

- **Trainingsmodus:** Um *Exploration* zu fördern – also die gezielte Berücksichtigung von Zügen, die bisher wenig beachtet wurden, aber dennoch vielversprechend sein könnten – wird eine probabilistische Auswahl verwendet. Eine Aktion  $a$  wird dabei proportional zur Anzahl der Besuche  $N(s, a)$  gewählt. Die resultierende Wahrscheinlichkeitsverteilung lautet:

$$\pi(a | s) = \frac{N(s, a)^{1/\tau}}{\sum_{b \in \mathcal{A}(s)} N(s, b)^{1/\tau}}$$

wobei  $\tau > 0$  eine *Temperaturkonstante* ist. Für hohe Werte von  $\tau$  ist die Verteilung gleichmässiger – auch selten besuchte Aktionen haben dann eine signifikante Wahrscheinlichkeit. Bei kleinen Werten von  $\tau$  konzentriert sich die Auswahl hingegen zunehmend auf die am häufigsten besuchten Züge. Der Parameter  $\tau$  reguliert damit das Gleichgewicht zwischen *Exploration* (Vielfalt) und *Exploitation* (Nutzung bewährter Züge).

- **Spielmodus:** Bei Auswertung, Wettkämpfen oder tatsächlichen Spielzügen wird meist deterministisch die Aktion mit den meisten Besuchen ausgewählt:

$$a^* = \arg \max_{a \in \mathcal{A}(s)} N(s, a)$$

Diese Vorgehensweise nutzt die in den Simulationen gesammelten Informationen am effizientesten aus.

Die resultierende Wahrscheinlichkeitsverteilung  $\pi(a | s)$  dient zudem als Zielverteilung für das Training des neuronalen Netzwerks. Das Netzwerk soll lernen, ähnliche Priorisierungen  $P(s, a)$  wie die MCTS-Suche zu liefern. Auf diese Weise verbessert sich die Qualität der Vorhersagen und damit die Effizienz der Suche sukzessive – ein zentrales Element im Lernprozess von AlphaZero.

## 2.4 AlphaZero und Self-Play

Die folgenden Ausführungen basieren im Wesentlichen auf den Originalarbeiten von Google DeepMind (2017), Lex Fridman (2020), Silver, Hubert et al. (2017b) und Silver, Schrittwieser et al. (2017a). Ergänzt wurden sie durch didaktisch aufbereitete Sekundärquellen wie Aaron Davis (2022) und ThePrincipalComponent (2022), welche insbesondere beim strukturierten Verständnis des Algorithmus geholfen haben.

### 2.4.1 Zusammenspiel von Monte Carlo Tree Search und neuronalen Netzen

Der Erfolg des AlphaZero-Algorithmus beruht wesentlich auf der engen Verzahnung zweier Komponenten: *Monte Carlo Tree Search (MCTS)* und *eines neuronalen Netzes*, das das Spielgeschehen einschätzen kann. Beide Elemente erfüllen unterschiedliche, aber sich ergänzende Aufgaben. Das neuronale Netz liefert eine schnelle, erfahrungsisierte Bewertung von Spielsituationen, während MCTS diese Einschätzungen nutzt, um systematisch nach starken Spielzügen zu suchen.

**Funktion des neuronalen Netzes** Das neuronale Netz übernimmt in AlphaZero zwei zentrale Rollen:

- **Policy-Vorhersage:** Für eine gegebene Brettstellung  $s$  liefert das Netz eine Wahrscheinlichkeitsverteilung  $P(s, \cdot)$  über alle möglichen Züge. Diese Verteilung spiegelt wider, welche Züge aus Sicht des Netzwerks vielversprechend sind.

- **Value-Schätzung:** Das Netz gibt zusätzlich einen Wert  $V(s) \in [-1, 1]$  zurück, der angibt, wie wahrscheinlich es ist, dass der aktuelle Spieler aus der Stellung  $s$  heraus das Spiel gewinnt (1 = sicherer Sieg, -1 = sichere Niederlage, 0 = ausgeglichen).

Beide Ausgaben basieren auf Erfahrung: Das Netz wurde anhand früherer Spiele trainiert, in denen es gelernt hat, welche Züge zu positiven Ergebnissen führen.

**Integration in die Monte Carlo Tree Search** MCTS wird verwendet, um für eine gegebene Stellung nicht nur einen einzigen Zug auszuwählen, sondern eine grosse Anzahl möglicher Fortsetzungen zu simulieren. Die dabei entstehenden Knoten im Suchbaum repräsentieren verschiedene zukünftige Spielverläufe. Die Policy-Vorhersage des neuronalen Netzes wird in der *Selektionsphase* verwendet, um gezielt Knoten zu bevorzugen, die laut Netz besonders relevant erscheinen. Die Value-Schätzung dient als Ersatz für eine vollständige Spielsimulation: Statt bis zum Spielende zu spielen, wird der erwartete Ausgang direkt durch das Netz geschätzt und in der *Rückpropagierung* verwendet.

Diese Form der MCTS ist also keine rein statistische Suche mehr, sondern eine **netzwerkgesteuerte Exploration des Spielbaums**. Die Auswahl der Züge ist dadurch strategischer, und weniger Rechenzeit wird auf unprononante oder schlechte Züge verschwendet. das

**Gegenseitige Verstärkung** Das Zusammenspiel dieser beiden Komponenten ist zentral für das Lernprinzip von AlphaZero:

- Das neuronale Netz liefert eine *informierte Einschätzung*, die der MCTS hilft, schneller gute Entscheidungen zu treffen.
- Umgekehrt liefert MCTS durch seine simulierten Spielverläufe Trainingsdaten, aus denen das neuronale Netz wiederum lernen kann.

So entsteht ein wechselseitiger Lernprozess: Das neuronale Netz *steuert die Suche* – und die Suche wiederum *verbessert das Netz*.

Diese Struktur erlaubt es AlphaZero, strategische Zusammenhänge nicht über explizite Regeln, sondern durch **rein datengetriebenes Lernen** zu erfassen. Statt vordefinierter Heuristiken entsteht eine dynamische Spielstärke, die sich schrittweise und selbstständig entwickelt. Die Qualität des Algorithmus hängt dabei nicht mehr allein von Rechenleistung oder Expertenwissen ab, sondern vor allem von der Qualität der eigenen Erfahrungsdaten und der Effizienz der Verzahnung zwischen Suche und Bewertung.

## 2.4.2 Self-Play: Lernen durch Erfahrung

Ein zentrales Konzept von AlphaZero ist das sogenannte *Self-Play* – also das Spielen gegen sich selbst. Dabei lernt der Algorithmus vollständig eigenständig, ohne dass ihm Strategien oder Spielzüge explizit beigebracht werden.

Der Ablauf ist einfach und effektiv:

1. Das aktuelle neuronale Netz spielt mit Hilfe von MCTS viele Partien gegen sich selbst.
2. Für jeden Spielzug werden drei Informationen gespeichert:
  - der Brettzustand  $s$ ,
  - die MCTS-Zugverteilung  $\pi(\cdot|s)$ ,
  - und das Endresultat  $z \in \{-1, 0, 1\}$  aus Sicht des aktuellen Spielers.
3. Diese Tripel  $(s, \pi, z)$  werden im sogenannten *Replay Buffer* gespeichert.
4. In der Trainingsphase wird das neuronale Netz mit diesen Daten verbessert.

Der Lernprozess ist also rein erfahrungsbasiert: Das Netz lernt nicht durch menschliche Partien, sondern durch die statistische Auswertung seiner eigenen Entscheidungen. Indem es sich kontinuierlich selbst herausfordert, passt es sein Spielniveau automatisch an.

## 2.4.3 Der Lernzyklus von AlphaZero

Der Lernprozess von AlphaZero basiert auf einem iterativen Zyklus, in dem das aktuelle neuronale Netzwerk selbstständig Partien generiert, daraus lernt und anschliessend auf seine Spielstärke hin überprüft wird. Abbildung 2.16 veranschaulicht diesen Ablauf in sechs zentralen Schritten, die in jeder Iteration durchlaufen werden.

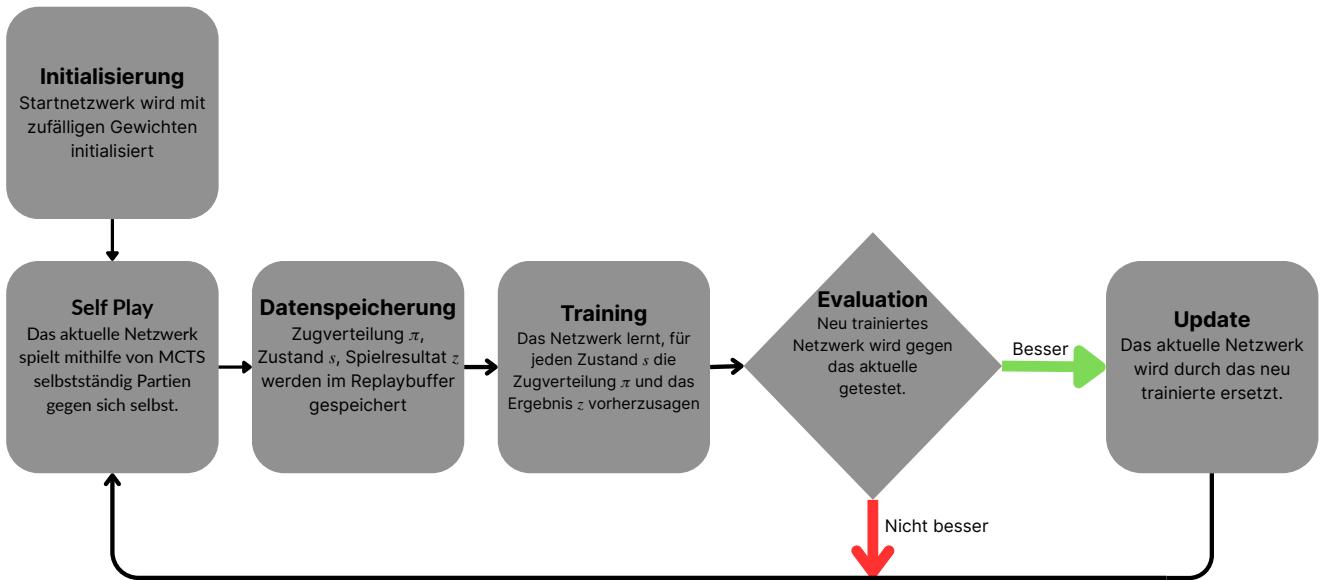


Abbildung 2.16: Schematische Darstellung des AlphaZero-Lernzyklus. Das aktuelle Netzwerk generiert durch Self-Play neue Trainingsdaten. Nach Training und Evaluation wird es bei ausreichender Überlegenheit aktualisiert (Eigene Abbildung).

**Initialisierung:** Zu Beginn wird ein neuronales Netzwerk mit zufälligen Parametern initialisiert. Es verfügt über keinerlei strategisches Vorwissen und dient als Ausgangspunkt für das selbstgesteuerte Lernen.

**Self-Play:** In einem zweiten Schritt bestreitet das Netzwerk eigenständig Partien gegen sich selbst. Für jede Stellung wird mithilfe von MCTS ein Zug ausgewählt, wobei sowohl die Policy- als auch die Value-Ausgabe des Netzes in die Suche einfließen. Um die Exploration zu fördern, erfolgt die Zugwahl in den frühen Spielphasen stochastisch, gesteuert durch einen Temperaturparameter  $\tau > 0$ .

**Datenspeicherung:** Während der Self-Play-Partien werden für jeden Zug drei Informationen gespeichert:

- der Spielzustand  $s$ ,
- die von MCTS berechnete Zugverteilung  $\pi(\cdot|s)$ ,
- sowie das Endresultat  $z \in \{-1, 0, 1\}$  aus Sicht des jeweils aktiven Spielers.

Diese Tripel  $(s, \pi, z)$  werden in einem temporären Zwischenspeicher abgelegt, dem sogenannten *Replay Buffer*. Dieser enthält eine Vielzahl vergangener Spielsituationen und bildet die Grundlage für das anschliessende Training. Durch zufälliges Sampling aus diesem Speicher wird eine decorrelierte Trainingsbasis geschaffen, die Überanpassung reduziert und den Lernprozess

stabilisiert.

**Training:** Auf Grundlage dieser Daten wird das neuronale Netz trainiert. Ziel ist es, aus einem Zustand  $s$  sowohl die optimale Zugverteilung  $\pi$  als auch den Spieldurchgang  $z$  möglichst genau vorherzusagen. Die Verlustfunktion kombiniert dabei drei Bestandteile: einen quadratischen Fehler (MSE) für den Value-Wert, einen Kreuzentropie-Term für die Policy und eine  $L_2$ -Regularisierung der Gewichte:

$$\mathcal{L} = (z - V(s))^2 - \pi^\top \log P(s, \cdot) + \lambda \cdot \|\theta\|^2 \quad (2.1)$$

Dabei steht  $V(s)$  für die vom Netzwerk geschätzte Bewertung der Stellung,  $P(s, \cdot)$  für die vorhergesagte Zugverteilung,  $\theta$  für die Netzparameter und  $\lambda$  für den Regularisierungsfaktor.

**Evaluation:** Nach dem Training wird das neue Netzwerk gegen die vorherige Version getestet. Die direkte Gewinnrate über mehrere Partien hinweg dient als Kriterium für den Vergleich. Eine Verbesserung liegt typischerweise dann vor, wenn das neue Netz eine Gewinnrate von über 55 % erreicht.

**Update:** Nur bei signifikanter Überlegenheit wird das neue Netzwerk übernommen. Andernfalls bleibt die vorherige Version aktiv, und der Zyklus beginnt von vorn.

Dieser Prozess wird über viele Iterationen hinweg wiederholt. Mit jeder Schleife verbessert sich die Spielstärke des Netzwerks, ohne dass menschliches Wissen oder externe Daten erforderlich wären. Der gesamte Lernfortschritt basiert ausschliesslich auf selbst generierten Partien und deren statistischer Auswertung. Die Qualität dieses Prozesses hängt massgeblich vom Zusammenspiel zwischen MCTS, Netzwerkarchitektur und der Vielfalt der Trainingsdaten ab.

## 2.5 Elo-Bewertungssystem

Zur Bewertung der Spielstärke von Othello-Spielalgorithmen eignet sich das Elo-Bewertungssystem, das ursprünglich für das Schachspiel entwickelt wurde. Es wird inzwischen auch häufig zur Evaluation von Künstlicher Intelligenz eingesetzt (Norelli & Panconesi, 2022). Im Folgenden werden die mathematischen Grundlagen des Systems erläutert sowie seine Anwendung zur Bewertung der Spielstärke verschiedener Programmversionen beschrieben.

### 2.5.1 Grundprinzip

Das Elo-Bewertungssystem basiert auf der Annahme, dass sich die Gewinnwahrscheinlichkeit zweier Spieler ausschliesslich aus der Differenz ihrer Wertungen ergibt. Für Spieler mit den

Ratings  $R_A$  und  $R_B$  berechnet sich die erwartete Gewinnwahrscheinlichkeit von Spieler A wie folgt:

$$E_A = \frac{1}{1 + 10^{(R_B - R_A)/400}}$$

Nach einer Partie wird die Bewertung entsprechend dem tatsächlichen Spielergebnis angepasst:

$$R'_A = R_A + K \cdot (S_A - E_A)$$

Dabei ist  $S_A \in \{0, 0.5, 1\}$  das tatsächliche Ergebnis der Partie (Verlust, Remis, Sieg) und  $K$  ein Anpassungsfaktor, der die Stärke der Anpassung steuert (Elo, 1978; Wikipedia Contributors, 2025).

## 2.5.2 Verwendung in KI-Systemen

In der Künstlichen Intelligenz wird das Elo-System genutzt, um die relative Spielstärke verschiedener Spielalgorithmen objektiv zu quantifizieren. Insbesondere in lernbasierten Verfahren, bei denen viele Versionen eines Modells über Generationen hinweg entstehen, ermöglicht es eine vergleichbare Bewertung. Das System ist besonders dann nützlich, wenn keine absolute Referenzstärke existiert, etwa durch eine offizielle Rangliste oder einen menschlichen Benchmark.

## 2.6 Edax als Referenz-Gegner

Für die Bewertung von Othello-Algorithmen auf höherem Niveau ist Edax eine etablierte Wahl. Die Engine zählt zu den stärksten öffentlich verfügbaren Othello-Programmen und eignet sich daher gut als Referenzgegner (Norelli & Panconesi, 2022). Im Folgenden wird die Funktionsweise von Edax beschrieben und sein Einsatz im Rahmen automatisierter Spielstärketests erläutert.

### 2.6.1 Funktionsweise von Edax

Edax ist eine leistungsstarke, quelloffene Othello-Engine, die auf klassischer Suchbaum-Logik basiert. Sie nutzt unter anderem Alpha-Beta-Suche, statische Bewertungsfunktionen, Zugreihenfolgenoptimierung, Hashing und Techniken wie Late Move Reductions. Die Engine hat mehrfach an internationalen Turnieren teilgenommen und wird aktiv weiterentwickelt.

Die Spielstärke von Edax lässt sich über Schwierigkeitsstufen (Level 0 bis Level 6) regulieren. Diese Levels begrenzen primär die Rechenzeit pro Zug – nicht direkt die Suchtiefe –, was

zu unterschiedlich tiefen Analysen führt. Auf Level 6 sucht Edax typischerweise 14–16 Züge tief und spielt damit auf dem Niveau eines sehr starken Amateurspielers.

Durch die fein abgestufte Spielstärke eignet sich Edax ideal zur Evaluation lernender Modelle (Delorme, 2025).

## 2.6.2 Einsatz zur Spielstärkebewertung

Zur Einschätzung der Leistungsfähigkeit von Othello-Bots wird Edax als fest definierter Gegner eingesetzt. Dabei werden Partien gegen verschiedene Edax-Level gespielt, wobei abwechselnd mit Schwarz und Weiss begonnen wird, um mögliche Anzugsvorteile auszugleichen.

Die daraus resultierenden Ergebnisse dienen als Datengrundlage für die Berechnung von Elo-Werten. Auf diese Weise lässt sich nachvollziehen, wie sich die Spielstärke eines Algorithmus über mehrere Trainingsgenerationen hinweg entwickelt.

## 2.7 Warum AlphaZero für Othello?

Obwohl Othello im Vergleich zu Schach oder Go als relativ einfaches Spiel gelten mag, stellt es klassische, auf exakter Suche basierende Methoden wie Minimax oder Brute-Force dennoch vor erhebliche Herausforderungen.

Ein wesentlicher Grund dafür ist der hohe *Branching Factor* des Spiels. In mittleren Spielphasen existieren oft 10–20 legale Züge pro Stellung, was zu einem exponentiellen Wachstum des Suchbaums führt. Ein vollständiger Brute-Force-Suchbaum über alle 60 möglichen Züge wäre astronomisch gross und selbst mit modernen Rechnern praktisch nicht durchsuchbar.

Zudem ist Othello ein sogenanntes *Late-Game-Game*: Viele Stellungen sind in der frühen Phase strategisch schwer zu bewerten, ihr Wert zeigt sich oft erst spät im Spiel. Klassische Heuristiken oder statische Bewertungsfunktionen greifen hier oft zu kurz (Świechowski et al., 2015).

Lernbasierte Verfahren wie AlphaZero bieten hierfür eine elegante Lösung: Sie erlernen Strategien nicht durch das Durchsuchen aller Möglichkeiten, sondern durch Erfahrung – in Form von selbst gespielten Partien. Das zugrundeliegende neuronale Netz entwickelt mit der Zeit ein zunehmend tiefes Verständnis für starke Spielzüge, ohne auf vorgefertigte Bewertungsfunktionen oder Endspieldatenbanken angewiesen zu sein.

Gerade für Spiele mit grossem Suchraum und schwer formulierbaren Heuristiken wie Othello ist AlphaZero daher ein besonders geeignetes Verfahren. Dieses Projekt zeigt zudem, dass ein solcher Ansatz auch ohne Hochleistungsrechner realisierbar ist – durch clevere Architektur und effiziente Implementierung.

## 2.8 Theoretischer Rahmen im Überblick

Die vorangegangenen Abschnitte haben die zentralen Konzepte des AlphaZero-Ansatzes vorgestellt: Monte Carlo Tree Search, neuronale Netzwerke zur Bewertung und Zugauswahl sowie das Training durch selbstgespielte Partien (Self-Play). Gemeinsam bilden sie ein lernfähiges System, das strategische Spielstärke allein durch Erfahrung entwickeln kann.

Gerade in Spielen wie Othello, deren Spielbaum zu gross für vollständige Suche ist und bei denen klassische Heuristiken nur begrenzt greifen, bietet dieser Ansatz erhebliche Vorteile. Gleichzeitig ermöglicht die Kombination von Planung (MCTS) und Lernen (Neuronalen Netzen) ein besonders flexibles und adaptives Spielverhalten.

Auf Basis dieser theoretischen Grundlagen wurde im Rahmen dieser Arbeit ein vollständiges AlphaZero-inspiriertes System für Othello implementiert. Die folgenden Kapitel beschreiben dessen Aufbau, Trainingsprozess und Evaluation.

# Kapitel 3

## Methodik

### 3.1 Technische Grundlagen und Tools

Für die Umsetzung des Projekts wurde die Programmiersprache Python verwendet, da sie sich aufgrund ihrer einfachen Syntax und der breiten Verfügbarkeit von spezialisierten Bibliotheken im Bereich des maschinellen Lernens besonders gut eignet. Zum Aufbau und Training des neuronalen Netzes kam das Deep-Learning-Framework PyTorch zum Einsatz. Weitere zentrale Libraries waren NumPy für numerische Operationen und Datenverarbeitung, Matplotlib für Visualisierungen sowie tqdm für Fortschrittsanzeigen beim Training.

Zum Laden und Speichern von Daten kamen unterschiedliche Formate zum Einsatz: Spielstatistiken und Trainingsverläufe (z. B. Epochenfortschritte) wurden mithilfe des json-Moduls im JSON-Format abgelegt, während die umfangreichen Self-Play-Daten effizient mit dem pickle-Modul serialisiert wurden. Diese Aufteilung ermöglicht eine klare Trennung zwischen lesbaren Konfigurationsdaten und binären Trainingsdaten und trägt so zu einer flexiblen und performanten Datenverwaltung bei.

Die Entwicklung und das Training des Algorithmus erfolgten vollständig auf einem Heimrechner. Dieser ist ausgestattet mit einer NVIDIA RTX 4060 GPU, einem Intel Core i7-Prozessor mit 24 Kernen sowie 32GB Arbeitsspeicher. Diese Hardware erlaubt es, grundlegende neuronale Netzwerke zu trainieren und MCTS Simulationen effizient durchzuführen, stellt aber im Vergleich zu professionellen Forschungsumgebungen eine klare Begrenzung dar.

### 3.2 Struktur der Implementierung

Die technische Umsetzung des Projekts basiert auf einer klar modularisierten Python-Architektur. Der Quellcode wurde in logisch getrennte Komponenten unterteilt, die jeweils spezifische Funk-

tionen übernehmen. Zusätzlich wurde ein paralleles Verarbeitungssystem entwickelt, um das Self-Play effizient zu skalieren. Die folgenden Abschnitte geben einen Überblick über den strukturellen Aufbau der Implementierung.

### 3.2.1 Modularer Aufbau

Die Implementierung wurde modular aufgebaut, um Übersichtlichkeit, Wiederverwendbarkeit und eine klare Trennung der Verantwortlichkeiten zu gewährleisten. Abbildung 3.1 zeigt die übergeordnete Struktur des Projekts, aufgeteilt in einzelne Module mit spezifischen Aufgabenbereichen.

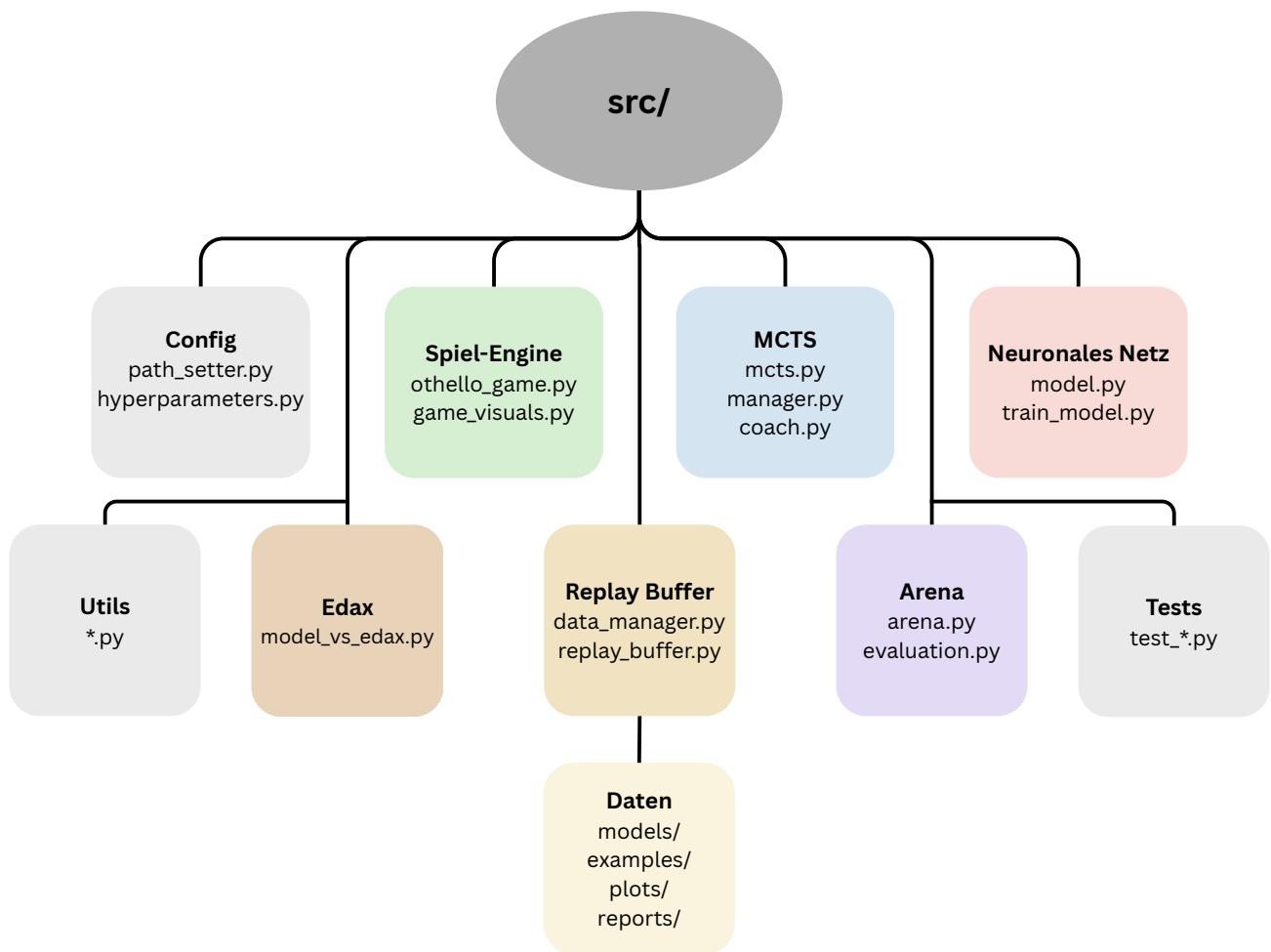


Abbildung 3.1: Modularer Aufbau des Projekts (Eigene Abbildung).

- **Spiel-Engine:** Die Klasse `OthelloGame` enthält die Spiellogik, inklusive Regelprüfung,

Spieldurchschritt und Endbedingung. Visuelle Darstellungen werden über `game_visuals.py` bereitgestellt.

- **Monte Carlo Tree Search (MCTS):** In `mcts.py` ist die zentrale Suchstrategie implementiert, die durch das neuronale Netz geführt wird. Die MCTS-Implementierung unterstützt Multiprocessing, um paralleles *Self-Play* zu ermöglichen.
- **Neuronales Netz:** Das in `model.py` definierte Netzwerk liefert für jede Brettstellung eine Policy (Zugwahrscheinlichkeiten) und eine Value-Schätzung. Das Training erfolgt über `train_model.py`.
- **Replay Buffer:** Die Dateien `data_manager.py` und `replay_buffer.py` verwalten die Trainingsdaten, die im Self-Play generiert und gespeichert werden.
- **Arena und Evaluation:** Über `arena.py` werden Netzwerke in direkten Duellen gegeneinander getestet. Mit `evaluation.py` werden daraus Elo-Ratings berechnet.
- **Edax-Anbindung:** Die Datei `model_vs_edax.py` ermöglicht Spiele gegen die Othello-Engine Edax zur externen Bewertung der Spielstärke.
- **Konfiguration:** Globale Parameter (z. B. Lernrate, MCTS-Simulationen) werden zentral in `hyperparameters.py` und `path_setter.py` definiert.
- **Hilfsfunktionen:** In `utils/*.py` befinden sich diverse Funktionen, etwa zur Umwandlung von Koordinaten zwischen MCTS und Netzwerk oder zur Konfiguration des Loggers.
- **Tests:** Zur Sicherstellung der Funktionalität wurden eigene Testskripte in `test_*.py` entwickelt.

### 3.2.2 Parallelisierung durch Manager-Worker-Struktur

Um die Durchführung von Self-Play-Spielen effizient zu gestalten, wurde eine sogenannte *Manager-Worker-Architektur* implementiert. Dabei laufen 22 parallele Worker-Prozesse, die eigenständig Spiele durchführen und Anfragen an das neuronale Netz generieren. Diese Anfragen werden nicht direkt an die GPU übergeben, sondern an einen zentralen Manager-Prozess.

Der Manager bündelt die eingehenden Anfragen zu Mini-Batches und leitet sie gesammelt an die GPU weiter, wo sie vom neuronalen Netz ausgewertet werden. Die Ergebnisse werden anschliessend wieder an die jeweiligen Worker verteilt. Diese Struktur reduziert den Overhead durch Einzelanfragen und verbessert die Auslastung der GPU erheblich. Außerdem pausiert jeder Worker nach einer abgeschlossenen Anfrage für kurze Zeit, um die CPU-Last zu reduzieren und eine Überhitzung des Systems während langer Trainingsphasen zu vermeiden. Eigene Implementierung inspiriert von Stanford Online (2024).

Abbildung 3.2 veranschaulicht diese Architektur:

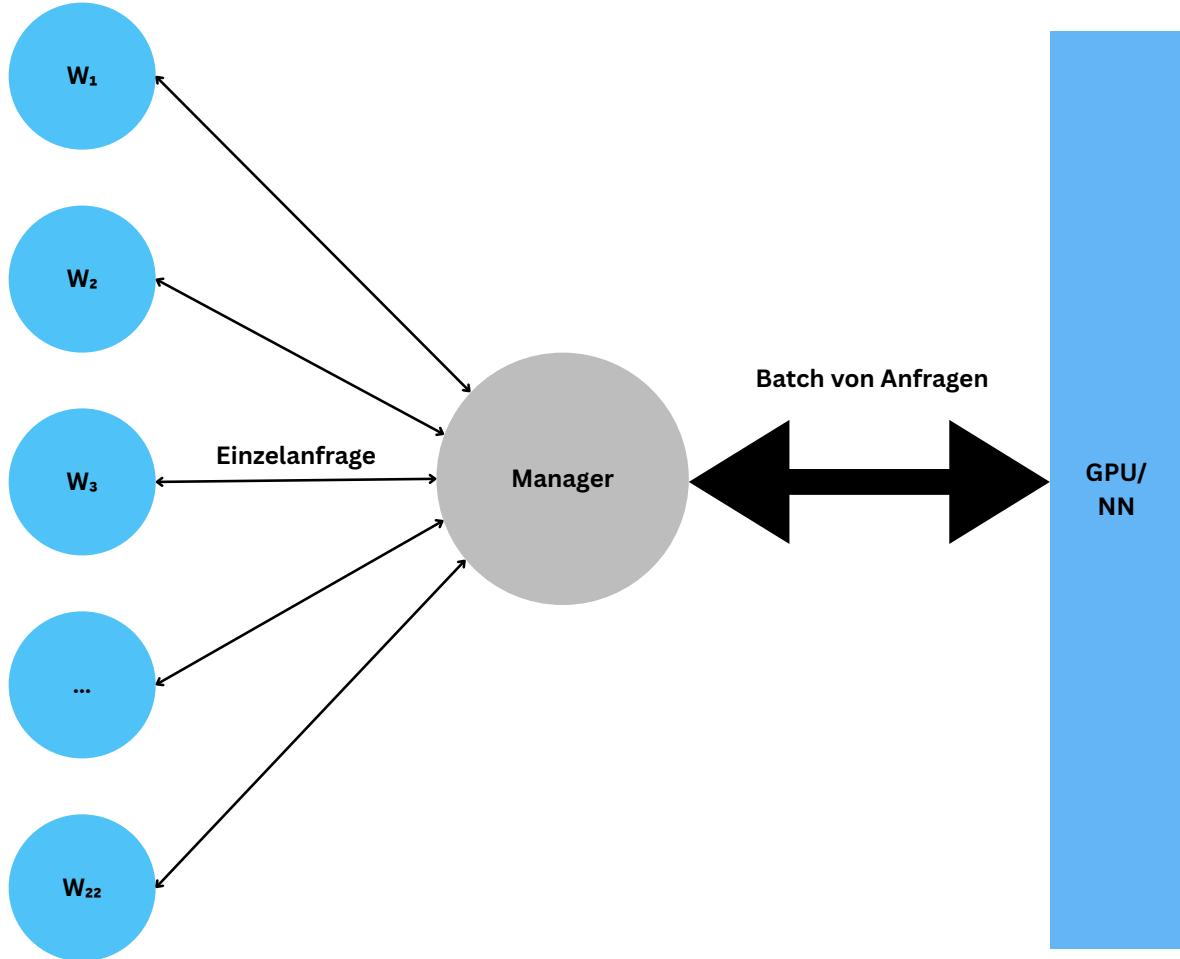


Abbildung 3.2: Manager-Worker-Architektur zur parallelen Ausführung von Self-Play und Batch-Abfrage des neuronalen Netzes (Eigene Abbildung).

### 3.3 Neuronales Netz

Das neuronale Netz dient als zentrale Bewertungsinstanz im AlphaZero-Algorithmus. Es verarbeitet die aktuelle Brettstellung und gibt zwei Vorhersagen aus: eine Policy  $\pi$ , die Wahrscheinlichkeiten für alle möglichen Züge angibt, und einen Value  $v \in [-1, 1]$ , der die erwartete Bewertung der Stellung aus Sicht des aktuellen Spielers schätzt. Die Architektur ist in Abbildung 3.3 dargestellt.

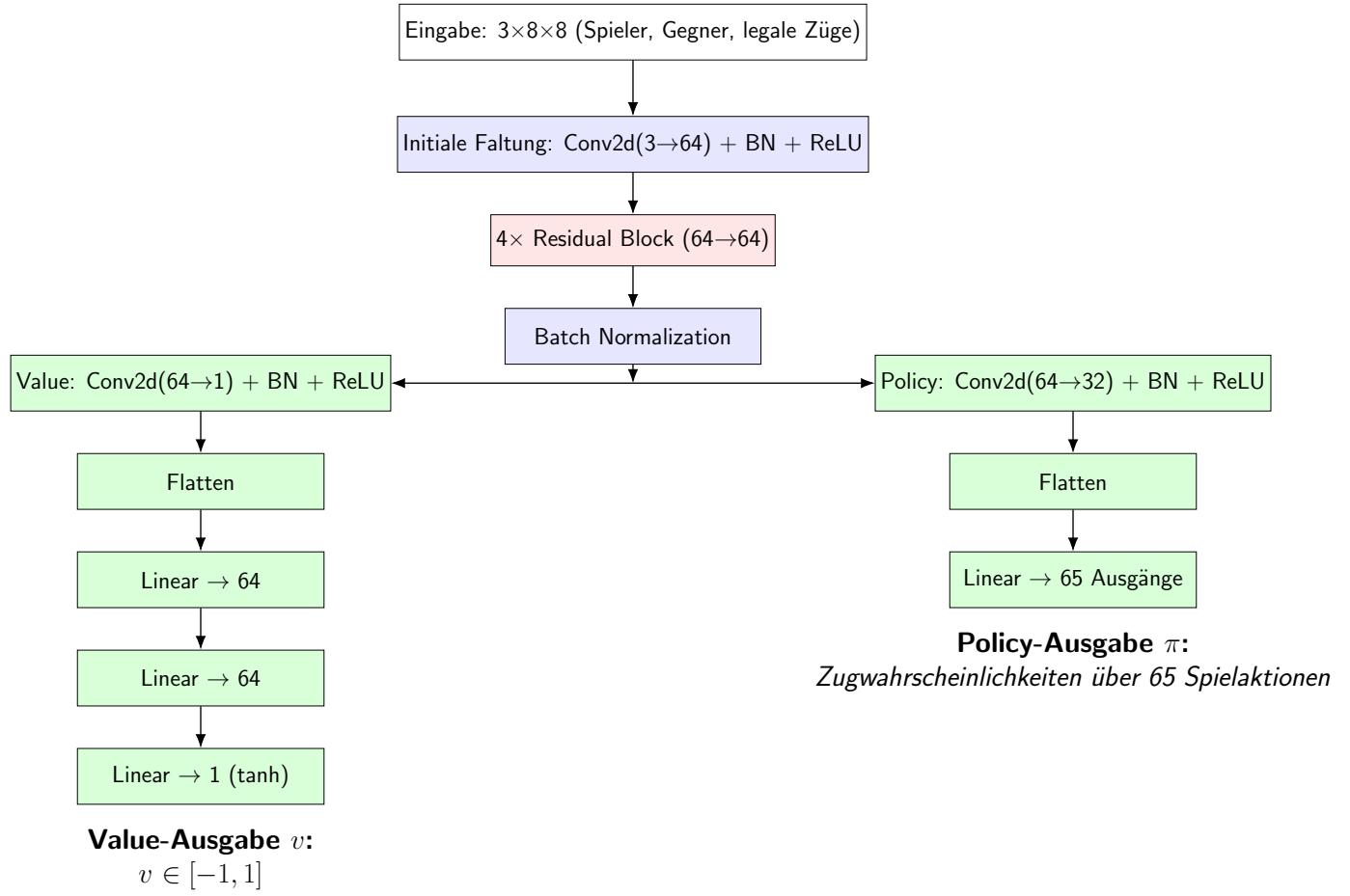


Abbildung 3.3: Architektur des neuronalen Netzes. Das Netzwerk verarbeitet ein  $8 \times 8$ -Othellobrett mit drei Eingabekanälen und verzweigt sich nach einem gemeinsamen Feature-Extractor in einen Policy- und einen Value-Head (Eigene Abbildung).

### 3.3.1 Eingabeformat

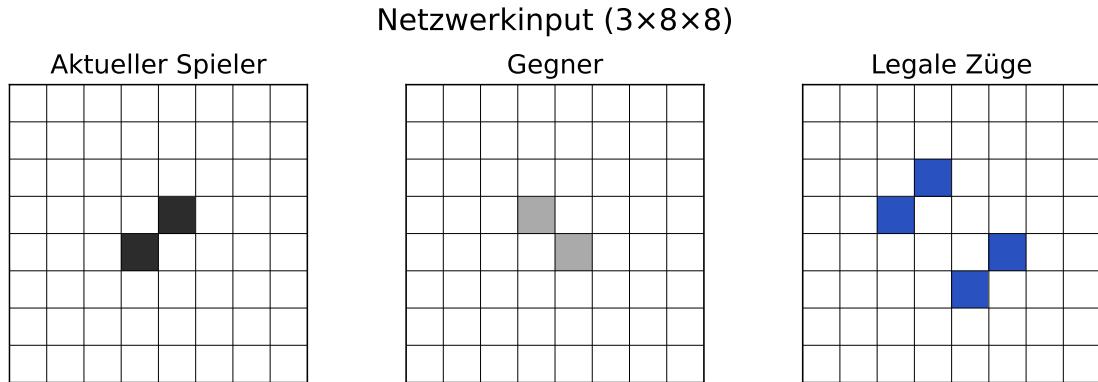


Abbildung 3.4: Visualisierung des Netzwerkinputs (Startaufstellung): Die Brettstellung wird in drei separaten Kanälen dargestellt. Gezeigt werden (von links nach rechts) die Positionen der eigenen Steine, der gegnerischen Steine sowie die aktuell legalen Züge für den Spieler. Jeder Kanal hat die Dimension  $8 \times 8$ , sodass sich insgesamt ein Eingabetensor der Form  $3 \times 8 \times 8$  ergibt (Eigene Abbildung).

Das EingabefORMAT besteht aus einem dreidimensionalen Tensor der Form  $3 \times 8 \times 8$ . Die drei Kanäle kodieren:

- die Positionen der eigenen Steine,
- die Positionen der gegnerischen Steine,
- die legalen Züge für den aktuellen Spieler.

Diese kanonische Darstellung sorgt dafür, dass das Netz stets aus Sicht des aktuellen Spielers lernt. Die Unterscheidung zwischen „eigenen“ und „gegnerischen“ Steinen ist dabei nicht absolut, sondern relativ zur Spielperspektive: Weiss und Schwarz werden je nach Spielerrolle umkodiert. Dadurch wird die Lernaufgabe symmetrischer und das neuronale Netz muss nicht separat lernen, wie Weiss oder Schwarz zu spielen ist. Das Modell behandelt beide Farben konsistent und kann generalisierte Wissen über das Spiel entwickeln.

### 3.3.2 Feature-Extraktion und Residual-Blöcke

Zu Beginn erfolgt eine Faltungsschicht mit einem  $3 \times 3$ -Kernel, die aus den drei Eingangs-kanälen 64 Feature-Maps erzeugt. Anschliessend folgen vier identische Residual-Blöcke, wie in Abbildung 3.5 dargestellt.

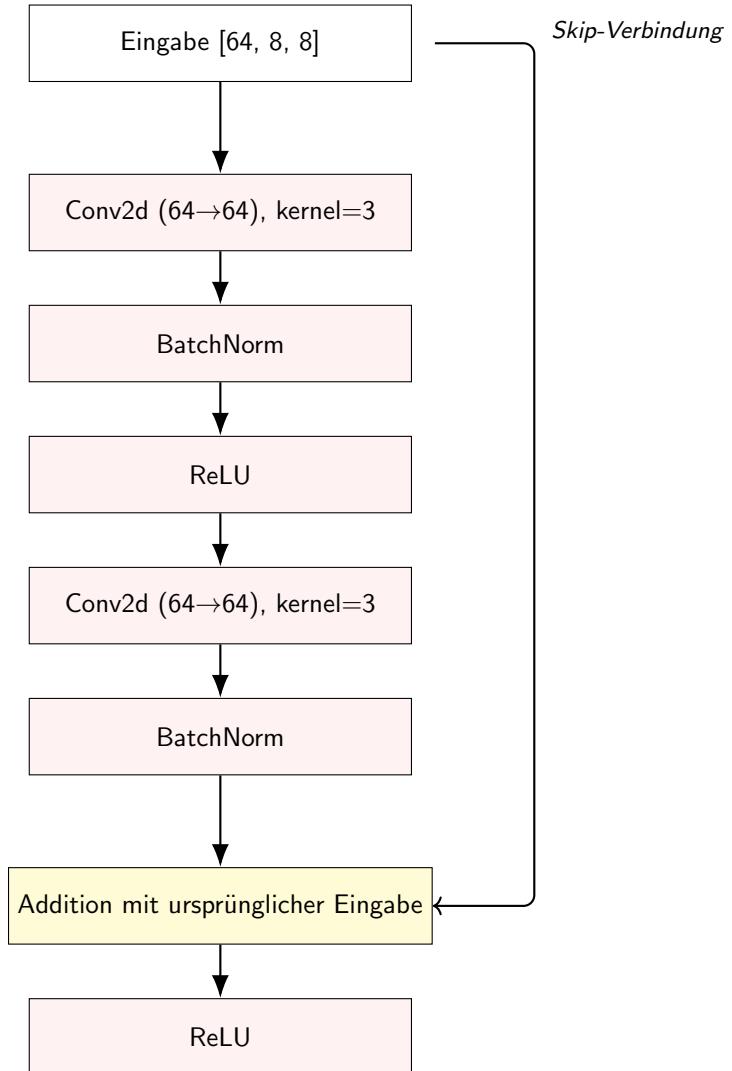


Abbildung 3.5: Residual Block: Zwei Faltungsschritte mit Batch-Normalisierung und ReLU, ergänzt durch eine Skip-Verbindung (eigene Darstellung).

Jeder Block besteht aus zwei Faltungsschichten mit je 64 Kanälen und einem  $3 \times 3$ -Kernel. Nach jeder Faltung folgen Batch-Normalisierung und ReLU-Aktivierung. Durch eine Skip-Verbindung wird die ursprüngliche Eingabe des Blocks zur Ausgabe addiert. Diese Technik stammt aus ResNet-Netzen und stabilisiert den Trainingsprozess, insbesondere bei tieferen Architekturen.

### 3.3.3 Policy- und Value-Head

Nach der gemeinsamen Feature-Extraktion teilt sich das Netz in zwei Pfade:

- **Policy-Head:** Besteht aus einer Faltungsschicht mit 32 Kanälen, Batch-Normalisierung und ReLU. Anschliessend wird die Ausgabe flachgelegt und durch eine vollständig verbundene Schicht mit 65 Ausgängen geführt – einer für jedes Feld sowie ein zusätzlicher

für den Pass-Zug.

- **Value-Head:** Beginnt mit einer Faltungsschicht mit einem Kanal, ebenfalls gefolgt von Batch-Normalisierung und ReLU. Danach folgen drei Fully-Connected-Schichten mit 64, 64 und schlussendlich einem Ausgangsneuron. Die finale Aktivierung erfolgt über eine tanh-Funktion, sodass der Ausgabewert  $v \in [-1, 1]$  liegt.

### 3.3.4 Verlustfunktion

Die Optimierung des neuronalen Netzwerks basiert auf einer kombinierten Verlustfunktion, die zwei Zielgrößen berücksichtigt:

- den **Cross-Entropy-Loss** zwischen der vom MCTS erzeugten Ziel-Policy  $\pi$  und der vom Netzwerk vorhergesagten Zugverteilung  $p$ ,
- sowie den **Mean Squared Error (MSE)** zwischen dem tatsächlichen Spielausgang  $z$  und der vom Netzwerk geschätzten Stellungseinschätzung  $v$ .

Diese beiden Komponenten werden zu einem Gesamtverlust addiert, der während des Trainings minimiert wird. Die genaue Form der Verlustfunktion wurde bereits in Abschnitt 2.4.3 (Gleichung (2.1)) eingeführt.

## 3.4 Trainingsprozess

Der Trainingsprozess orientiert sich am im Theorienteil beschriebenen AlphaZero-Prinzip und wurde in diesem Projekt wie folgt umgesetzt: Das Netzwerk wird anhand von selbstgenerierten Partien trainiert, die durch Self-Play-Duelle entstehen. Dabei wird in allen Partien dasselbe aktuelle Netzwerk verwendet, das über MCTS die Zugauswahl steuert. Die daraus entstehenden Daten — bestehend aus Brettstellung  $s$ , Ziel-Policy  $\pi$  und Spielausgang  $z$  — werden im Replay Buffer gespeichert und dienen als Trainingsbeispiele für die jeweils nächste Iteration.

### 3.4.1 Trainingsstrategie

Nach jeder Iteration im AlphaZero-Zyklus – bestehend aus Self-Play, Training und Evaluation – wurde das neuronale Netz mit Hilfe der gesammelten Daten aktualisiert. Die maximale Größe des Replay Buffers wurde auf 40'000 Beispiele begrenzt. Um sicherzustellen, dass jeder Datenpunkt im Schnitt mehrfach vom Netzwerk gesehen wird, wurde die sogenannte *Coverage* auf den Wert 5 gesetzt (Eigene Designidee, basiert auf Silver, Schrittwieser et al. (2017b)).

Das Training des Netzwerks erfolgte in sogenannten *Epochen*. Eine Epoche beschreibt dabei einen vollständigen Durchlauf durch alle gespeicherten Trainingsdaten. Damit das Netzwerk pro

Iteration etwa fünfmal so viele Trainingsbeispiele sieht, wie aktuell im Buffer gespeichert sind, wurde die Anzahl Trainingsschritte nach folgender Formel berechnet:

$$\text{Anzahl Schritte} = \frac{N \cdot \text{Coverage}}{B} = \frac{40\,000 \cdot 5}{512} \approx 390$$

Dabei ist  $N$  die Anzahl gespeicherter Beispiele im Buffer,  $B$  die gewählte Batch-Grösse (512), und „Coverage“ die gewünschte Datenabdeckung.

Die Daten für das Training wurden parallel generiert: In jeder Iteration führten 22 Worker-Prozesse jeweils 12 vollständige Partien aus. Da pro Partie im Schnitt etwa 60 Trainingsbeispiele entstehen (eines pro Spielzug), wurden pro Iteration ca. 15'840 neue Datenpunkte gesammelt.

Zur Verbesserung der Generalisierung wurde *on-the-fly Datenaugmentation* eingesetzt: Vor jedem Trainingsschritt wurde jede Brettstellung zufällig rotiert oder gespiegelt. Dadurch entstand effektiv ein achtmal grösserer Trainingsdatensatz, ohne dass zusätzliche Daten gespeichert werden mussten.

Auch die Anzahl der MCTS-Simulationen pro Zug wurde im Verlauf des Trainings schrittweise erhöht, um mit der steigenden Spielstärke des Netzwerks mitzuhalten. Zu Beginn wurden lediglich 100 Simulationen verwendet. Ab Iteration 6 wurde der Wert stufenweise erhöht, bis er ab Iteration 46 konstant bei 400 lag. Die folgende Abbildung zeigt diese Entwicklung über die Trainingsiterationen hinweg:

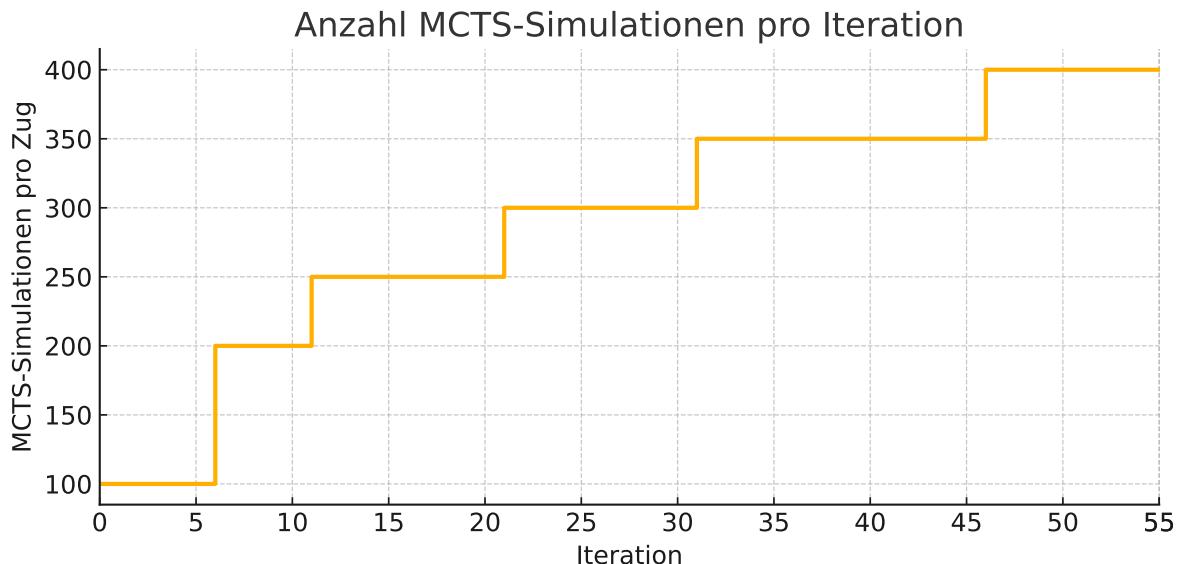


Abbildung 3.6: Anzahl MCTS-Simulationen pro Zug in Abhängigkeit der Trainingsiteration (eigene Darstellung)

Ausserdem wurde die gewählte Netzwerkarchitektur bewusst kompakt gehalten, um den Trainings- und Ausführungsaufwand zu reduzieren. Mit nur 4 Residualblöcken und einer Parameteranzahl von rund 250'000 liegt das Modell um ein Vielfaches unter der Komplexität vergleichbarer AlphaZero-Implementierungen für komplexere Spiele wie Schach oder Go.

Gemäss Schwartz et al. (2020) steht die Anzahl der benötigten FLOPs in direktem Zusammenhang mit dem Energieverbrauch eines Modells. Eine kleinere Architektur senkt somit nicht nur den Speicherbedarf, sondern auch den Stromverbrauch pro Vorwärtspass und pro Trainingsiteration. Auch ohne direkte Messung kann daher abgeleitet werden, dass dieses Design im Sinne einer ressourcenschonenden „Green AI“ zu bewerten ist.

### 3.4.2 Optimierung

Die Optimierung der Netzwerkgewichte erfolgte mit dem AdamW-Optimierer bei einer Start-Lernrate von 0.01 („AdamW — PyTorch 2.8 documentation“, 2024). Als Lernratenstrategie kam OneCycleLR mit *Cosine Annealing* zum Einsatz: Die Lernrate wurde während der ersten Trainingsschritte sanft erhöht (Warmup-Phase) und anschliessend progressiv abgesenkt – bis auf ein Endniveau im Bereich von ca.  $10^{-6}$ . Dieses Verfahren unterstützt eine stabile Konvergenz und verhindert, dass das Training zu früh in schlechte lokale Minima fällt (Pytorch, 2025).

### 3.4.3 Trainingsdauer

Das Netzwerk wurde über insgesamt 55 Iterationen hinweg trainiert. Der gesamte Trainingsprozess erstreckte sich über einen Zeitraum von etwa zwei Tagen. Dank GPU-Beschleunigung (NVIDIA RTX 4060) und der parallelen Ausführung der Self-Play-Spiele durch Multiprocessing konnte jede Iteration effizient abgeschlossen werden.

## 3.5 Evaluation

Zur Bestimmung der Spielstärke der trainierten Modelle wurden systematische Vergleichspartien gegen die Othello-Engine *Edax* durchgeführt. Diese Open-Source-Engine gilt als eines der stärksten frei verfügbaren Othello-Programme und erlaubt über konfigurierbare Schwierigkeitsstufen (Level 0–6) eine präzise Abstufung der Gegnerstärke.

### 3.5.1 Versuchsaufbau

Getestet wurden Netzwerkgenerationen im Abstand von fünf Iterationen, d. h. die Modelle der Generationen 0, 5, 10, . . . , 55. Jedes dieser Modelle trat gegen Edax auf den Leveln 1 bis 6 an. Um eine möglichst faire Bewertung zu ermöglichen, wurden pro Modell–Level-Kombination

jeweils 100 Partien gespielt. Dabei wechselte der Startspieler zwischen den Partien, sodass jedes Modell 50 Mal mit Schwarz und 50 Mal mit Weiss spielte.

Ein eigens entwickeltes Python-Skript koordinierte die Matches und rief Edax als Subprozess auf. Zur Effizienzsteigerung und Klarheit der Resultate wurden sinnlose Paarungen – z. B. sehr schwache Modelle gegen hohe Edax-Level – durch eine gezielte Vorauswahl mittels einer `skip_battle`-Funktion vermieden.

### 3.5.2 Suchstrategie und Variabilität

Jeder Zug des Modells wurde mittels *Monte Carlo Tree Search (MCTS)* ausgewählt, wobei pro Entscheidung 1400 Simulationen durchgeführt wurden. Um zu verhindern, dass stets dieselben Spielverläufe entstehen – was bei deterministischen Gegnern wie Edax leicht geschehen kann – wurde in den ersten acht Zügen der Temperaturparameter  $\tau$  auf 0,15 gesetzt. Dadurch wurde die Auswahl der Züge stochastischer gestaltet, was eine grösere Vielfalt in der Eröffnungsphase erzeugt. Nach dem achten Zug wurde  $\tau$  auf 0 gesetzt, um in der Mittel- und Endspielphase deterministisch die beste bekannte Zugoption zu wählen.

### 3.5.3 Datenerhebung und Zielgrösse

Nach jeder Partie wurde das Ergebnis (Sieg, Niederlage oder Unentschieden aus Sicht des Modells) gespeichert. Diese Daten bildeten die Grundlage für die Auswertung der Spielstärkeentwicklung über die Iterationen hinweg. Neben der direkten Analyse der Matchstatistiken dienten die Resultate auch zur Berechnung von relativen Elo-Werten (siehe Abschnitt 2.5.1), um eine objektive Einordnung der Modelle im Vergleich zu Edax zu ermöglichen.

## 3.6 Benutzeroberfläche und Visualisierung

Um die Entscheidungen des Modells nachvollziehbar zu machen, wurde eine eigene grafische Benutzeroberfläche (GUI) in `pygame` entwickelt. Sie dient sowohl der Visualisierung des Spielverlaufs als auch der Einsicht in zentrale Metriken der Monte Carlo Tree Search (MCTS) und der Ausgaben des neuronalen Netzes. Die GUI ist ein zentrales Hilfsmittel zur qualitativen Analyse und Evaluation des Spielverhaltens des Modells.

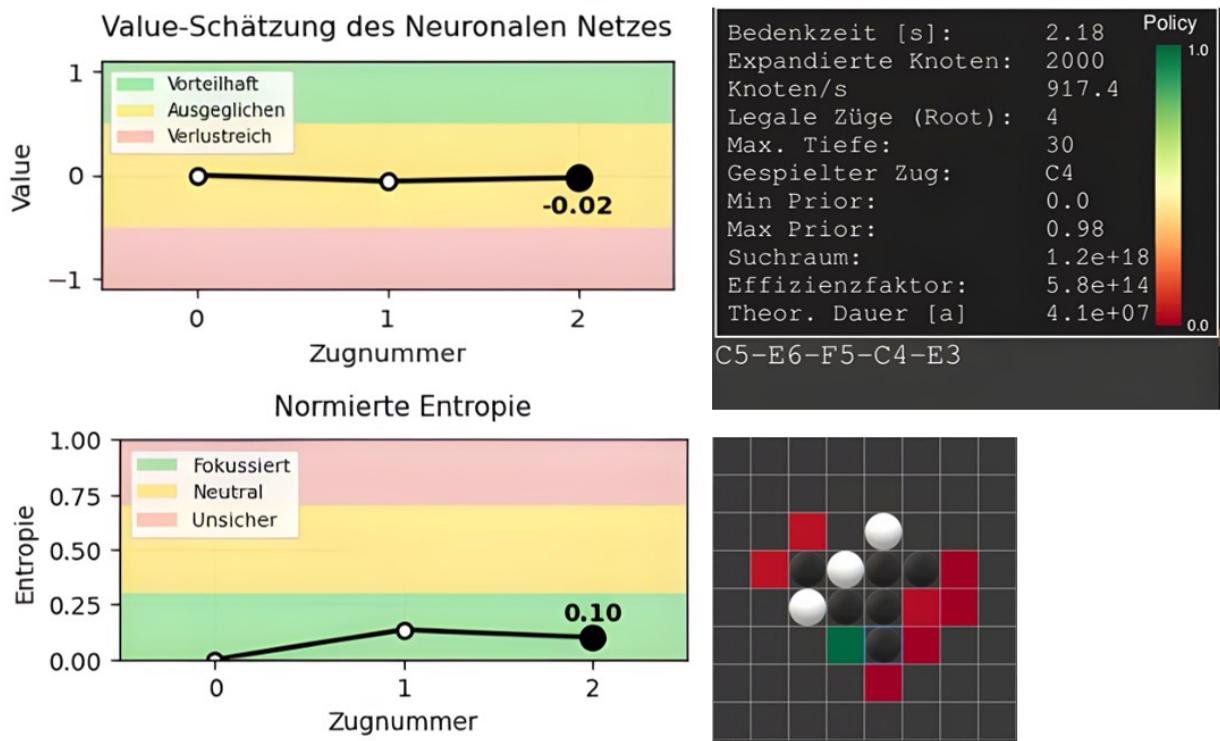


Abbildung 3.7: Grafischen Benutzeroberfläche in einer frühen Spielsituation. Alle GUI-Komponenten sind sichtbar: das Spielfeld, farbige Policy-Markierungen, die Value- und Entropie-Verläufe sowie die wichtigsten MCTS-Kennzahlen. Die Policy legt hier den Zug C4 nahe, was auch durch die blaue Markierung als zuletzt gespielter Zug angezeigt wird (Eigene Abbildung).

### 3.6.1 Darstellung des Spielbretts

Im Zentrum der Oberfläche befindet sich das Othello-Brett mit einer Grösse von  $8 \times 8$  Feldern. Die aktuell belegten Felder sind durch weisse bzw. schwarze Spielsteine dargestellt. Der zuletzt gespielte Stein ist durch eine blaue Umrandung hervorgehoben, um den Spielverlauf besser nachvollziehen zu können.

### 3.6.2 Policy-Darstellung

Die vom neuronalen Netz vorhergesagte Zugwahrscheinlichkeit (Policy) wird visuell durch farbige Einfärbung der Spielfelder angezeigt. Dabei steht:

- **Grün** für den Züge mit sehr hoher Priorität,
- **Gelb** für Züge mit mittelhoher Priorität,
- **Rot** für Züge mit niedriger Priorität.

Ein Farbverlauf rechts im Bild stellt die zugrundeliegende Skala dar, basierend auf der normierten Policy-Ausgabe.

### 3.6.3 Neuronale Bewertung

Oben rechts befinden sich zwei Liniendiagramme:

- Die **Value-Schätzung** gibt für jeden bisher gespielten Zug des Modells die Bewertung der Stellung durch das neuronale Netz an. Positive Werte deuten auf eine vorteilhafte Stellung für das Modell hin.
- Die **normierte Entropie** der Policy misst die Unsicherheit der Netzentscheidung. Geringe Entropie signalisiert eine fokussierte Strategie.

### 3.6.4 MCTS-Informationen

Im rechten unteren Bereich der grafischen Oberfläche werden zentrale Kenngrößen der Monte Carlo Tree Search (MCTS) angezeigt. Diese Metriken liefern wichtige Einblicke in die Effizienz und das Verhalten des Suchalgorithmus:

- **Bedenkzeit:** Dauer der MCTS-Suche für den aktuellen Zug (in Sekunden).
- **Expandierte Knoten:** Anzahl der im Suchbaum tatsächlich evaluierten Positionen.
- **Knoten/s:** Rechenleistung in Form der durchschnittlich expandierten Knoten pro Sekunde – ein Mass für die Effizienz der Implementierung.
- **Legale Züge (Root):** Anzahl der gültigen Züge in der aktuellen Ausgangsstellung.
- **Maximale Tiefe:** Längster Suchpfad vom Wurzelknoten bis zu einem Blattknoten im Baum. Diese Tiefe entspricht der weitesten simulierten Zugfolge.
- **Gespielter Zug:** Der letztlich vom Modell gewählte Zug, dargestellt in algebraischer Notation.
- **Min/Max Prior:** Niedrigste bzw. höchste Policy-Wahrscheinlichkeit unter den legalen Zügen in der Root-Position, direkt aus der Policy-Ausgabe des neuronalen Netzes.

#### Suchraum-Analyse

Die folgenden Größen quantifizieren den theoretischen Umfang der Suche und verdeutlichen die Rolle der MCTS als selektives Verfahren:

- **Theoretischer Suchraum:**

Der Suchraum  $S$  ergibt sich näherungsweise zu:

$$S = b^d$$

wobei  $b$  die durchschnittliche Verzweigungsbreite (d.h. Anzahl legaler Züge pro Stellung) und  $d$  die maximale Tiefe des Suchbaums ist.

*Beispiel:* Für  $b = 4$  und  $d = 30$  ergibt sich:

$$S = 4^{30} \approx 1.2 \times 10^{18} \text{ Knoten}$$

- **Effizienzfaktor:**

$$\eta = \frac{S}{N}$$

wobei  $N$  die Anzahl expandierter Knoten ist. Ein hoher Wert ( $\eta \gg 1$ ) zeigt, dass das Modell nur einen Bruchteil des theoretischen Raumes untersucht hat.

- **Theoretische Dauer einer vollständigen Suche:**

$$t_{\text{theor}} = \frac{S}{r}$$

mit  $r$  als Anzahl Knoten pro Sekunde. Diese Grösse schätzt die hypothetische Rechenzeit, die nötig wäre, um den gesamten Suchraum vollständig zu durchsuchen.

*Beispiel:* Für  $S = 1,2 \times 10^{18}$  und  $r = 917$ :

$$t_{\text{theor}} \approx 1,2 \times 10^{15} \text{ Sekunden} \approx 41 \text{ Millionen Jahre}$$

### Hinweise zur Interpretation der Metriken

- Eine grosse maximale Tiefe lässt auf langfristige Planung schliessen.
- Ein hoher Effizienzfaktor zeigt eine stark selektive Suche – typisch für MCTS.
- Eine geringe Differenz zwischen *Min* und *Max Prior* kann auf Unsicherheit oder Gleichwertigkeit der Optionen aus Sicht des Netzes hinweisen.
- Die enorme Lücke zwischen theoretischer und tatsächlicher Rechenzeit verdeutlicht, dass MCTS in Kombination mit neuronaler Heuristik extrem effizient arbeitet.

### 3.6.5 Zugliste

Am unteren Rand der Oberfläche wird eine chronologische Liste aller bisher gespielten Züge angezeigt, in algebraischer Notation. Dies erlaubt eine einfache Rückverfolgung des Spielverlaufs.

### 3.6.6 Zweck und Nutzen

Diese Oberfläche wurde primär für die Evaluation des Modells und zur Unterstützung der Analyse im Rahmen dieser Arbeit entwickelt. Sie bietet eine **intuitive** Möglichkeit, strategisches Verhalten, Unsicherheiten und Bewertungsänderungen des Modells visuell darzustellen.

## 3.7 Reflexion und Limitationen

Trotz der soliden Umsetzung des Trainings- und Evaluationsverfahrens bestehen gewisse Einschränkungen, welche die Aussagekraft und Übertragbarkeit der Resultate begrenzen.

Ein wesentlicher Aspekt betrifft die Rechenressourcen. Das gesamte Training und die Evaluation erfolgten lokal auf einem einzelnen Rechner mit einer RTX 4060 GPU und einem Intel Core i7-Prozessor. Dadurch waren sowohl die Anzahl der durchführbaren Trainingsdurchläufe als auch die Anzahl an MCTS-Simulationen pro Zug auf ein praktikables Mass begrenzt. In professionellen Umgebungen würden typischerweise deutlich höhere Simulationszahlen und grössere Netzwerke verwendet, was eine feinere Spielstärkendifferenzierung ermöglichen würde.

Auch die Architektur des verwendeten neuronalen Netzwerks fällt im Vergleich zu etablierten AlphaZero-Varianten eher kompakt aus. Sie wurde bewusst vereinfacht gehalten, um die Trainingszeit zu reduzieren und eine bessere Nachvollziehbarkeit im Rahmen der Arbeit zu gewährleisten. Damit einher geht jedoch eine begrenzte Ausdrucksstärke des Modells, insbesondere in komplexeren Spielsituationen.

Die im Rahmen der Evaluation berechneten Elo-Werte erlauben eine vergleichende Einschätzung der verschiedenen Modellgenerationen, sind jedoch lediglich relativ zu den getesteten Edax-Leveln zu verstehen. Da Edax selbst keine offiziell verifizierte Elo-Skala besitzt, können die Ergebnisse nicht mit externen Massstäben verglichen werden.

Trotz dieser Einschränkungen lassen sich mehrere Stärken des Projekts hervorheben. Es wurde ein selbstlernendes System umgesetzt, das auf dem Prinzip des reinen Selbstspiels basiert und in der Lage ist, seine Spielweise schrittweise zu verbessern. Die Evaluation gegen eine etablierte Engine wie Edax schafft die Grundlage für eine objektive und reproduzierbare Bewertung der Lernfortschritte. Darüber hinaus ist das System modular aufgebaut und dokumentiert, wodurch

spätere Erweiterungen oder Anpassungen erleichtert werden.

Insgesamt zeigt das Projekt, dass sich zentrale Prinzipien moderner Spiel-KI auch mit begrenzten Ressourcen und reduzierter Modellkomplexität umsetzen lassen. Dabei wird deutlich, dass nicht allein die Rechenleistung über den Erfolg entscheidet, sondern vor allem die Effizienz der eingesetzten Algorithmen und Strukturen - vergleiche Abschnitt 3.2.2. Dieses Ergebnis ist auch im weiteren Kontext bedeutsam: In einer Zeit wachsender Diskussionen über den Energieverbrauch grosser KI-Modelle liefert diese Arbeit ein konkretes Beispiel für einen ressourcenschonenden Ansatz. Der hier gewählte Weg steht stellvertretend für die Idee einer *Green AI*, die auf Nachhaltigkeit, algorithmische Effizienz und Zugänglichkeit setzt – ein Thema, das in den kommenden Jahren weiter an Bedeutung gewinnen dürfte (Schwartz et al., 2020).

# Kapitel 4

## Ergebnisse und Diskussion

### 4.1 Ergebnisse

Im Folgenden werden die zentralen Resultate des Trainingsprozesses vorgestellt. Ziel ist es, die Lernentwicklung des Modells sowohl quantitativ anhand objektiver Metriken wie Verlustfunktionen, Gewinnraten und Elo-Werten als auch qualitativ durch die Analyse des Zugwahlverhaltens nachvollziehbar zu machen.

Die Darstellung ist chronologisch entlang des Trainingsfortschritts aufgebaut: Beginnend mit dem Verlauf der Trainingsverluste wird anschliessend die Entwicklung der Spielstärke analysiert, bevor abschliessend das strategische Verhalten des Modells anhand von Zughäufigkeiten visualisiert wird. Zur Ergänzung der quantitativen Auswertung wird zusätzlich eine exemplarische Partie zwischen dem Modell und einem **menschlichen Spieler** analysiert, um typische Entscheidungsprozesse und Suchmuster im konkreten Spielverlauf sichtbar zu machen.

### 4.1.1 Trainingsverlauf

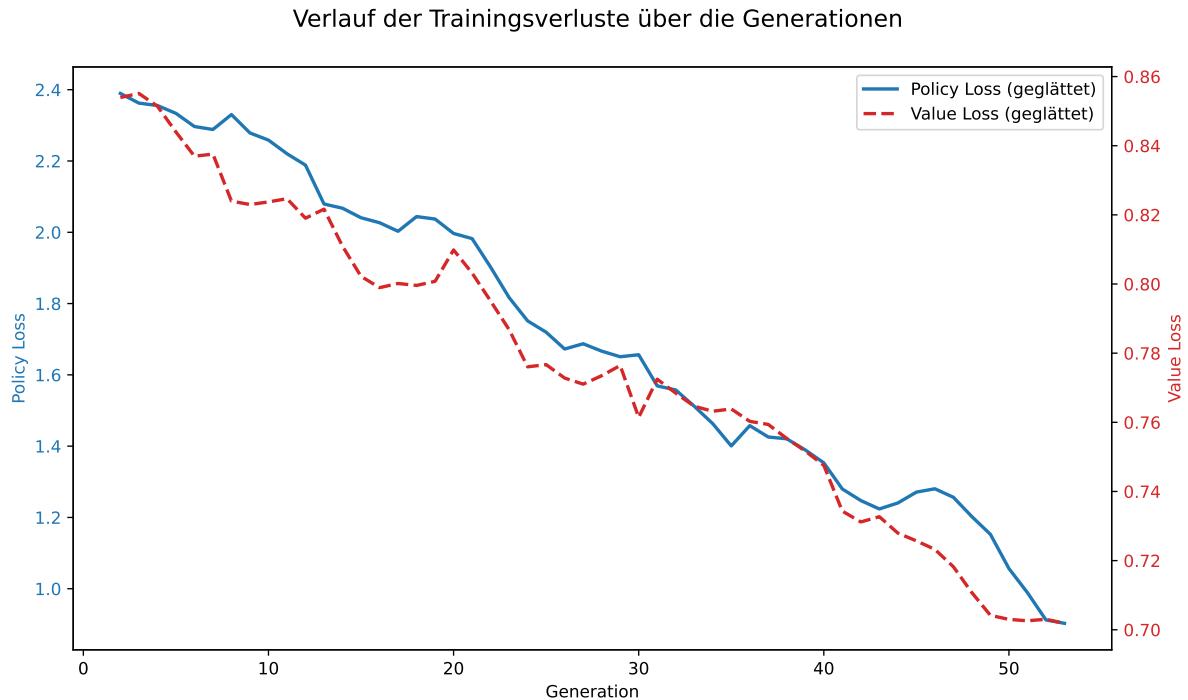


Abbildung 4.1: Verlauf der durchschnittlichen Trainingsverluste (Policy Loss und Value Loss) über alle Generationen des AlphaZero-Trainings, jeweils geglättet zur besseren Lesbarkeit (Eigene Abbildung).

Abbildung 4.1 zeigt den Verlauf der durchschnittlichen Trainingsverluste (Loss) über alle Generationen hinweg. Dargestellt sind sowohl der *Policy Loss* als auch der *Value Loss*, jeweils geglättet zur besseren Lesbarkeit.

Deutlich erkennbar ist ein durchgehend abnehmender Verlauf beider Verlustfunktionen über den Trainingsprozess hinweg. Der Policy Loss sinkt von über 2.4 auf unter 1.0, während der Value Loss von etwa 0.86 auf rund 0.70 zurückgeht. Beide Verläufe zeigen keine abrupten Sprünge oder starken Schwankungen, sondern entwickeln sich relativ stetig nach unten.

Der kontinuierliche Rückgang der Verlustfunktionen legt nahe, dass das neuronale Netz über die Trainingsgenerationen hinweg zunehmend in der Lage war, sowohl die Bewertung von Positionen als auch die Zugauswahl besser abzubilden.

### 4.1.2 Spielstärkeentwicklung

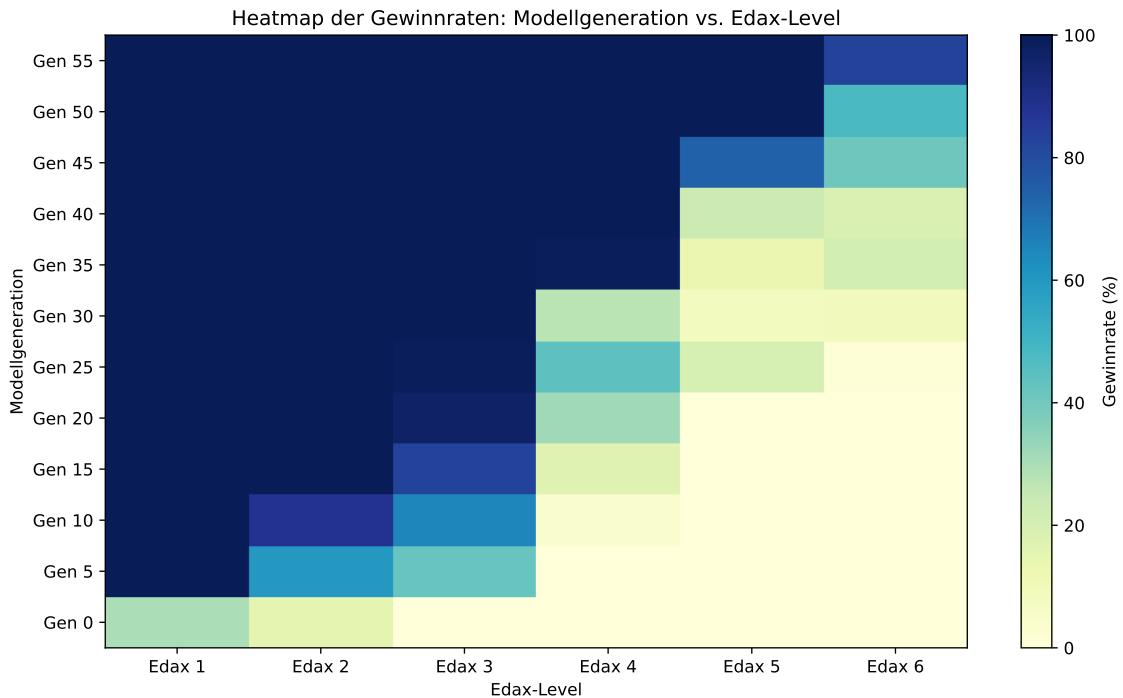


Abbildung 4.2: Gewinnraten der trainierten AlphaZero-Othello-Modelle (jede fünfte Generation) gegen Edax auf den Schwierigkeitsstufen 1–6. Farbskala: Anteil gewonnener Partien in 100 Spielen pro Matchup, gleichmäßig variierte Startspielerrolle (Eigene Abbildung).

Zur objektiven Bewertung der Spielstärke wurde das trainierte Modell in jeder fünften Generation gegen Edax auf den Schwierigkeitsstufen 1 bis 6 getestet. Dabei wurde jede Modellgeneration in jeweils 100 Partien gegen jeden Edax-Level getestet, wobei die Startspielerrolle gleichmäßig variiert wurde. Der MCTS verwendete dabei stets 1400 Simulationen pro Zug.

Abbildung 4.2 zeigt die resultierenden Gewinnraten als Heatmap. Deutlich sichtbar ist eine sukzessive Verbesserung der Modellgenerationen: Während frühe Modelle kaum gegen Edax 2 bestehen können, erreichen spätere Modelle stabile Gewinnraten bis hin zu Edax 5. Erst bei Edax 6 bricht die Erfolgsquote merklich ein.

Diese Tendenz spiegelt sich auch in der Entwicklung der Elo-Bewertung wider (vgl. Abbildung 4.3). Die Elo-Werte steigen mit zunehmender Trainingsgeneration kontinuierlich an und erreichen in Generation 55 einen Wert von etwa 1780. Zum Vergleich: Edax Level 5 wird anhand der Spielausgänge auf rund 1650 Elo geschätzt, während Edax 6 oberhalb von 1800 angesiedelt ist.

Insgesamt lässt sich eine klare Lernkurve beobachten, die mit zunehmendem Training zu einer verbesserten Spielstärke führt.

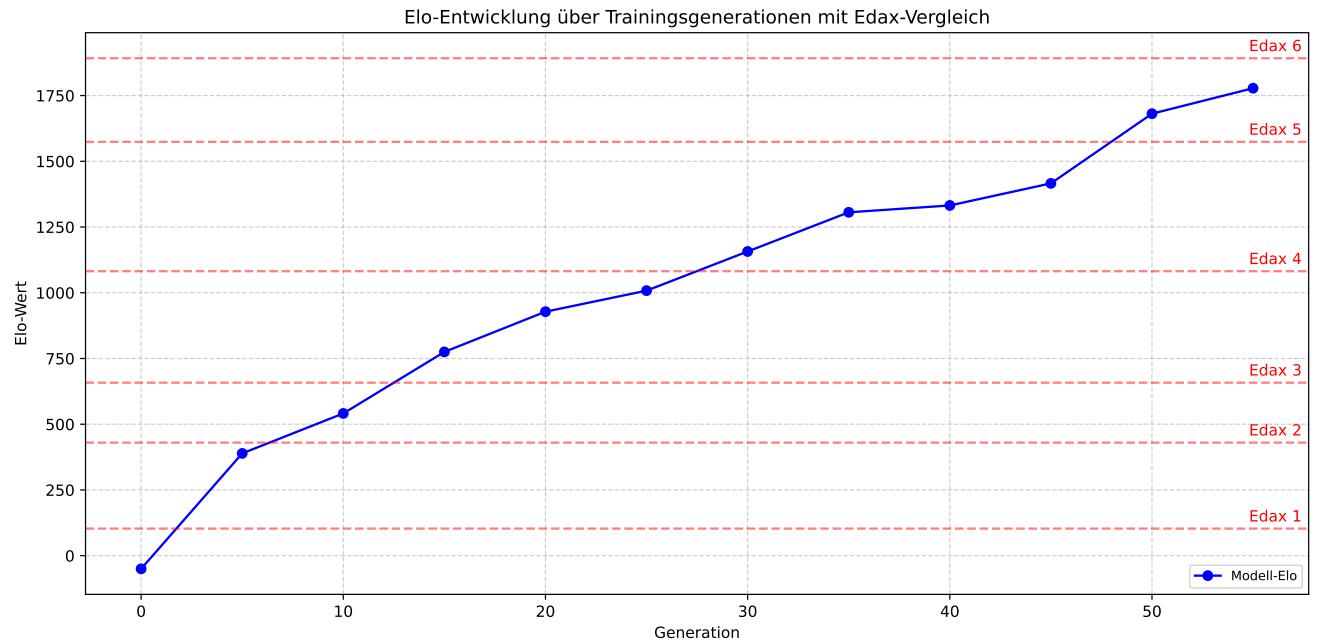


Abbildung 4.3: Entwicklung der Elo-Bewertung der trainierten AlphaZero-Othello-Modelle über den Trainingsverlauf, ermittelt aus den Partien gegen Edax Level 1–6. Markiert ist der Endwert von Generation 55 ( 1780 Elo) (Eigene Abbildung).

#### 4.1.3 Lernverhalten

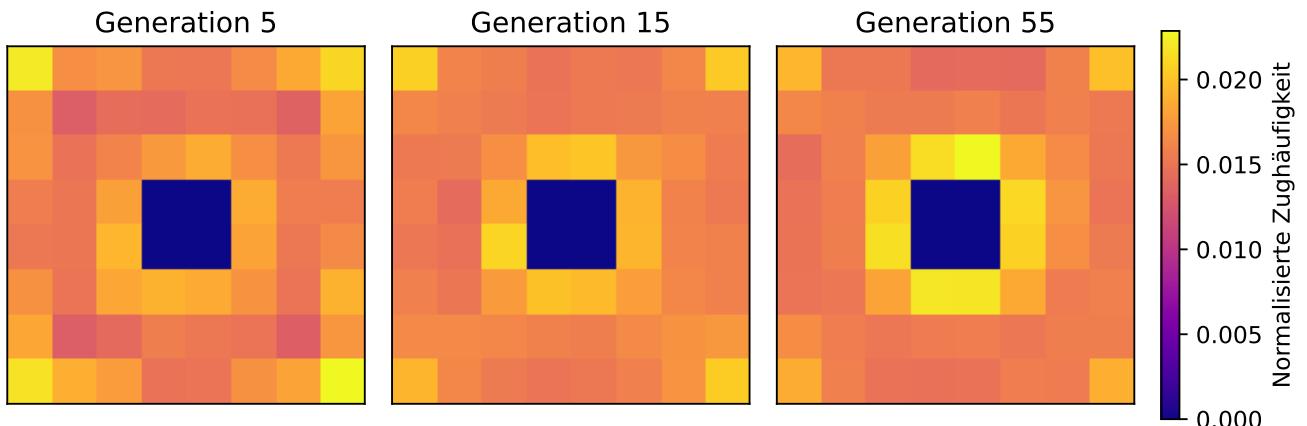


Abbildung 4.4: Normalisierte Zughäufigkeit pro Feld für die Modellgenerationen 5, 15 und 55. Helle Farben markieren häufig gewählte Zugpositionen (Eigene Abbildung).

Abbildung 4.4 zeigt die normalisierte Zughäufigkeit über alle Spiele hinweg für drei ausgewählte Modellgenerationen (5, 15 und 55). Für jede Generation wurde erfasst, welche Spielfelder bei der Zugauswahl bevorzugt wurden – oder anders formuliert: auf welche Felder sich die „Aufmerksamkeit“ des Modells im Laufe des Trainings besonders konzentriert hat.

In Generation 5 zeigt sich ein deutliches Muster: Das Modell bevorzugt Rand- und insbesondere Eckfelder, während zentrale Positionen seltener gewählt werden. Dieses Verhalten verändert sich im weiteren Trainingsverlauf signifikant. Bereits in Generation 15 ist eine ausgeglichene Verteilung erkennbar. In Generation 55 schliesslich richtet sich der Fokus klar auf zentrale Felder, während Randbereiche zunehmend gemieden werden.

Diese Entwicklung verdeutlicht eine Veränderung der internen Spielstrategie. Der Lernprozess manifestiert sich hier nicht nur in einer steigenden Spielstärke, sondern auch in einem sichtbaren Wandel des Zugwahlverhaltens über die Zeit hinweg.

## 4.2 Analyse einer Partie gegen das Modell

Im Folgenden werden drei exemplarische Spielsituationen aus einer Partie zwischen dem entwickelten Modell und einem menschlichen Spieler analysiert. Die Analyse konzentriert sich auf die sichtbaren Ausgaben des neuronalen Netzes (Policy, Value, Entropie), das Spielgeschehen auf dem Brett sowie die zugehörigen MCTS-Kennzahlen. Eine Bewertung der gezeigten Spielstärke erfolgt im anschliessenden Diskussionsteil.

### 4.2.1 Frühe Stellung: nur ein Stein

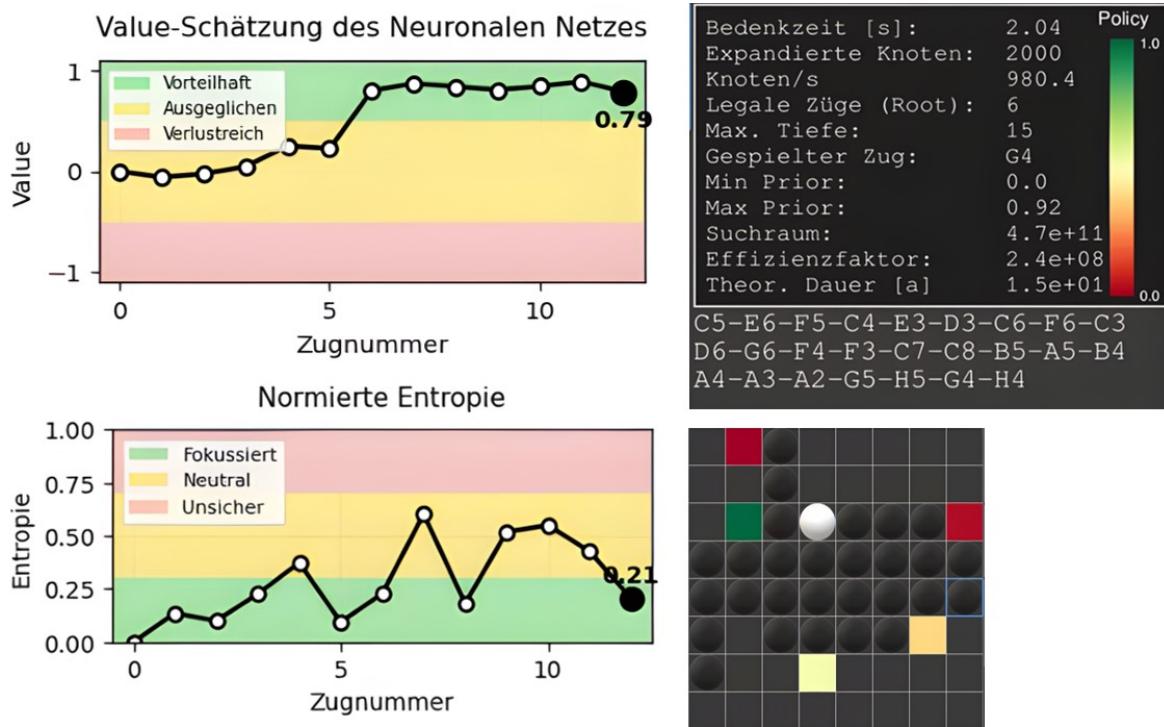


Abbildung 4.5: Frühe Spielsituation in einer Partie zwischen dem Modell und einem menschlichen Spieler. Das Modell (Weiss) besitzt zu diesem Zeitpunkt lediglich einen einzigen Stein. Die GUI zeigt eine fokussierte Policy, eine positive Value-Schätzung sowie die wichtigsten MCTS-Kennzahlen (Eigene Abbildung).

In dieser frühen Stellung verfügt das Modell (Weiss) nur über einen einzelnen Stein auf dem Spielfeld. Dennoch zeigt die Value-Schätzung des neuronalen Netzes einen positiven Wert von rund 0.79 (vgl. oberes Diagramm rechts), was auf eine als günstig eingeschätzte Stellung hindeutet.

Die Policy-Ausgabe ist klar fokussiert, was sich sowohl in der farblichen Einfärbung der Zugwahrscheinlichkeiten auf dem Brett (Policy-Heatmap) als auch in der relativ niedrigen Entropie zeigt. Der zuletzt gespielte Zug ist durch einen blauen Rahmen markiert. Die farbliche Kodierung der legalen Züge entspricht dem üblichen Schema: Grün für hohe Priorität, Gelb für mittlere, Rot für niedrige.

Im rechten unteren Bereich der GUI werden zudem zentrale MCTS-Kennzahlen angezeigt. In dieser Stellung wurden 2000 Knoten expandiert und eine maximale Suchtiefe von 15 erreicht. Zusätzlich ist der theoretische Suchraum dargestellt, ebenso wie der berechnete Effizienzfaktor der Suche (vgl. Abschnitt 3.6).

## 4.2.2 Endspiel mit Eckfokus

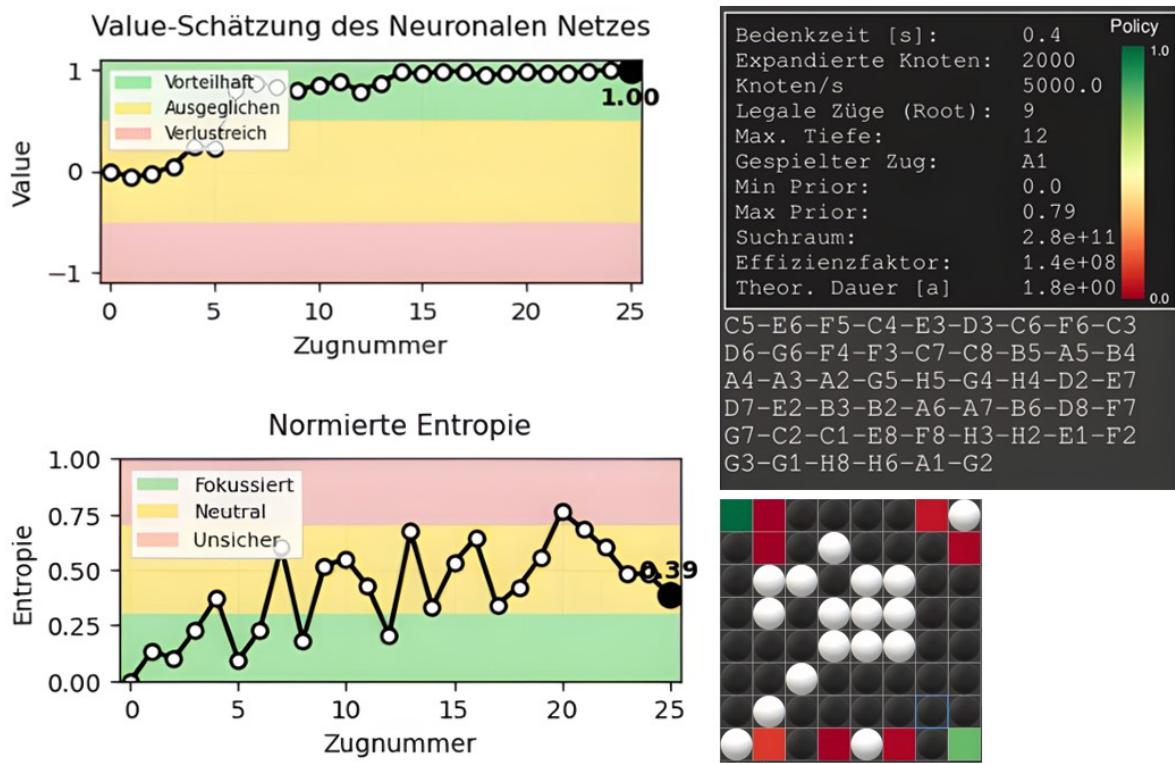


Abbildung 4.6: Spielsituation im Endspiel: Das Modell (Weiss) dominiert das Zentrum und hat bereits zwei Ecken besetzt. Die GUI zeigt eine eindeutige Value-Schätzung von 1.0 sowie eine neutrale Entropie. Die Policy legt den Fokus klar auf die verbleibenden Ecken (Eigene Abbildung).

In dieser Stellung befindet sich das Spiel bereits im Endspiel. Das Modell (Weiss) kontrolliert weite Teile des Zentrums und hat zwei der vier Ecken dauerhaft besetzt (oben rechts und unten links).

Die Value-Schätzung des neuronalen Netzes liegt bei maximalen 1.0, was auf einen aus Sicht des Modells sicheren Partiegewinn. Gleichzeitig ist die Entropie mit einem Wert von 0.39 im neutralen Bereich (vgl. Diagramme oben rechts).

Die Policy-Ausgabe fokussiert sich klar auf die beiden noch unbesetzten Ecken: Sowohl G1 als auch A8 sind in grün dargestellt und weisen die höchste Priorität auf.

Im rechten unteren Bereich zeigt die GUI erneut zentrale MCTS-Metriken: In dieser Stellung wurden 2000 Knoten expandiert, eine maximale Tiefe von 12 erreicht sowie ein theoretischer Suchraum von  $2.8 \times 10^{11}$  berechnet. Der Effizienzfaktor der Suche liegt bei rund  $2.4 \times 10^8$ , woraus sich eine starke Fokussierung auf einzelne Spielverläufe ergibt (vgl. Abschnitt 3.6).

### 4.2.3 Endsituation und Sieg

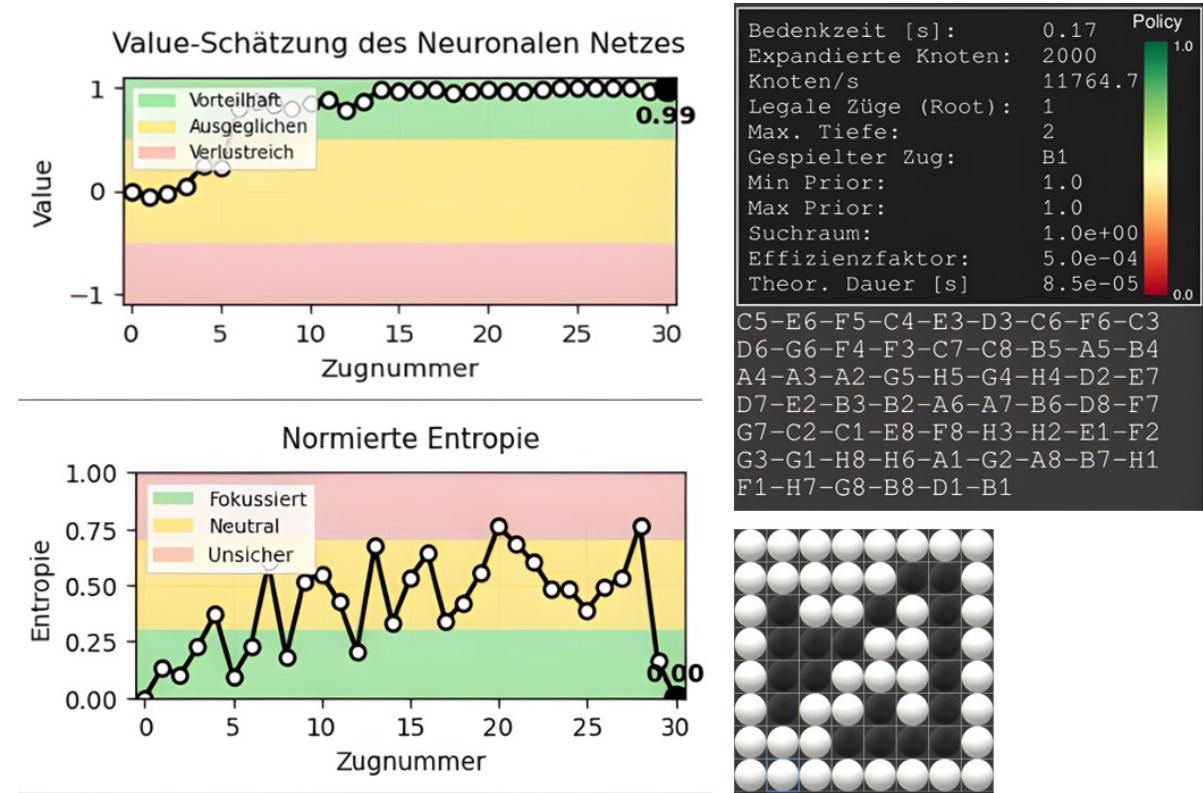


Abbildung 4.7: Abschluss situation der Partie. Das Modell (Weiss) hat alle vier Ecken besetzt und dominiert das Spielfeld klar. Die Value-Schätzung liegt bei 0,99, die Entropie beträgt 0,0 (Eigene Abbildung).

In dieser letzten Stellung ist die Partie faktisch entschieden. Das Modell (Weiss) hat das gesamte Zentrum kontrolliert und alle vier Eckfelder mit eigenen Steinen besetzt. Die verbleibenden schwarzen Steine sind passiver Lage und vollständig umschlossen.

Die Value-Schätzung des neuronalen Netzes liegt bei 0,99, was auf einen aus Sicht des Modells sicheren Sieg hindeutet. Gleichzeitig beträgt die Entropie 0,0, was auf eine vollständige Sicherheit in der Zugwahl hinweist (vgl. obere rechte Diagramme).

Der zuletzt gespielte Zug ist unten links im Feld B1 durch eine blaue Umrandung markiert.

Die Policy zeigt eine extreme Fokussierung: Nur ein Zug ist überhaupt noch legal und wurde mit einer Priorität von 1,0 bewertet (vgl. Farbskala).

Im MCTS-Bereich der GUI ist eine sehr hohe Knotengeschwindigkeit sichtbar: 11764 Knoten pro Sekunde bei insgesamt 2000 expandierten Knoten. Die maximale Suchtiefe beträgt lediglich 2. Weitere Auffälligkeiten betreffen den auffällig niedrigen Effizienzfaktor ( $5,0 \times 10^{-4}$ ) und die

theoretische Suchdauer, die mit  $8,5 \times 10^{-5}$  Sekunden unter der realen Bedenkzeit liegt. Eine Einordnung dieser Werte erfolgt im Diskussionsteil (vgl. Abschnitt 4.3.4).

## 4.3 Diskussion

### 4.3.1 Bewertung des Trainingsverlaufs

Der kontinuierlich sinkende Verlauf der Trainingsverluste deutet auf einen stabilen Lernprozess hin. Wie bereits in Abschnitt 4.1.1 beschrieben, nimmt der Policy Loss im Verlauf des Trainings von über 2,4 auf unter 1,0 ab, während der Value Loss von etwa 0,86 auf rund 0,70 sinkt. Dieser gleichmässige Rückgang ohne auffällige Sprünge spricht für eine gut abgestimmte Trainingsarchitektur, die in der Lage ist, Muster in den Daten zuverlässig zu erfassen.

Allerdings zeigt sich, dass insbesondere der Value Loss relativ früh auf einem Plateau verharrt und im weiteren Verlauf nur noch geringfügig sinkt. Auch der Policy Loss bleibt deutlich über null, was darauf hinweist, dass die Modellvorhersagen nicht vollständig mit den Zielwerten übereinstimmen. Dies ist insofern nicht überraschend, als das eingesetzte Netzwerk mit lediglich vier ResNet-Blöcken bewusst kompakt gehalten wurde, um Trainingszeit und Rechenaufwand zu begrenzen – siehe Modellarchitektur in Abbildung 3.3. Die begrenzte Modellkapazität könnte somit einen Teil der beobachtbaren Verlustwerte erklären.

Auffällig ist zudem, dass der Policy Loss während des gesamten Trainings über dem Value Loss liegt. Dies lässt sich dadurch erklären, dass die Policy-Ausgabe eine vollständige Wahrscheinlichkeitsverteilung über 65 mögliche Spielzüge abbilden muss, während das Value-Netz lediglich einen einzelnen Skalar vorhersagt. Die Policy-Lernaufgabe ist damit wesentlich komplexer, was sich auch in höheren Verlustwerten niederschlägt.

Insgesamt zeigen die Verlaufsdaten, dass das neuronale Netz im Laufe des Trainings zunehmend in der Lage war, sowohl aussichtsreiche Züge als auch Positionsbewertungen zuverlässig zu modellieren – auch wenn die Loss-Werte auf das Vorhandensein gewisser Vorhersageunsicherheiten hinweisen.

### 4.3.2 Analyse der Spielstärkeentwicklung

Die Entwicklung der Spielstärke über die Trainingsgenerationen hinweg bestätigt den Eindruck eines erfolgreichen Lernprozesses. Wie in Abschnitt 4.1.2 dargestellt, verbessert sich die Gewinnrate des Modells kontinuierlich: Während frühe Generationen kaum gegen Edax Level 2 bestehen, erreichen spätere Modellversionen zunehmend positive Resultate auch gegen stärkere Gegner. Ab etwa Generation 40 gelingt es dem Modell regelmäßig, Edax Level 5 zu schlagen.

Gegen Level 6 bleibt die Gewinnrate hingegen deutlich unter 50 %, was auf eine noch bestehende Leistungsgrenze hinweist.

Diese Entwicklung spiegelt sich auch in der berechneten Elo-Kurve wider. Der Elo-Wert des Modells steigt mit jeder Trainingsiteration an und erreicht in Generation 55 einen Wert von rund 1780. Damit liegt das Modell knapp unter der geschätzten Stärke von Edax Level 6, der sich gemäss der Spielausgänge oberhalb von 1800 Elo einordnen lässt. Insgesamt ergibt sich ein plausibles und konsistentes Bild: Der Anstieg der Elo-Bewertung verläuft relativ stetig, wenn auch mit leichten Abflachungen zwischen einzelnen Generationen – insbesondere zwischen Generation 35 und 40 ist eine gewisse Verlangsamung des Fortschritts erkennbar. Dies könnte darauf hindeuten, dass die festgelegte Anzahl von MCTS-Simulationen pro Zug – vergleiche Abbildung 3.6 – in späteren Trainingsphasen nicht mehr ausreicht, um das volle Potenzial der stärkeren Netzwerke auszuschöpfen. Eine Erhöhung der Simulationsanzahl in fortgeschrittenen Generationen könnte daher ein sinnvoller nächster Schritt sein, um die Spielstärke weiter zu verbessern.

Die Evaluation gegen ein etabliertes Referenzsystem wie Edax bietet eine robuste Grundlage zur Einschätzung der relativen Spielstärke. Gleichzeitig muss jedoch beachtet werden, dass der Vergleich mit Edax nicht gleichbedeutend mit einer absoluten Spielstärkemessung ist. Die tatsächliche Einordnung in ein standardisiertes Elo-System wäre nur über Turniere gegen andere Othello-Engines oder menschliche Spieler möglich. Dennoch erlaubt der gewählte Ansatz eine nachvollziehbare und reproduzierbare Bewertung der Lernfortschritte über das Training hinweg.

### 4.3.3 Strategisches Verhalten und Policy

Die Analyse der Zughäufigkeit über verschiedene Trainingsgenerationen zeigt, dass sich das Modell nicht nur in seiner Spielstärke, sondern auch in seinem strategischen Verhalten deutlich weiterentwickelt hat. In den frühen Generationen – exemplarisch dargestellt durch Generation 5 – konzentriert sich die Auswahl der Spielzüge stark auf Rand- und Eckfelder. Dieses Muster ähnelt dem Verhalten menschlicher Anfänger, die oft intuitiv versuchen, sich stabile Positionen am Spielfeldrand zu sichern.

Mit zunehmender Trainingsdauer verändert sich dieses Bild jedoch grundlegend. In den späteren Generationen, insbesondere ab Generation 30, verlagert sich die Aufmerksamkeit des Modells zunehmend in das Zentrum des Spielfelds. Gleichzeitig nimmt die Häufigkeit von Zügen an den Rändern ab. Besonders auffällig ist, dass die inneren 4×4-Felder im Zentrum – wie in Abschnitt 2.1.3 theoretisch eingeordnet – zunehmend an Bedeutung gewinnen. Diese Verschiebung deutet auf eine fortschreitende strategische Reifung hin: Statt einfachen Heuristiken folgt das Modell offenbar zunehmend einer komplexeren Bewertung von Positionsvoorteilen.

Bemerkenswert ist zudem, dass die Ecken über alle Trainingsgenerationen hinweg konstant hohe Zugwahrscheinlichkeiten aufweisen. Dies lässt darauf schliessen, dass das Modell früh erkennt, dass Ecken schwer angreifbare und langfristig vorteilhafte Positionen darstellen – wie in Abbildung 2.5 gezeigt.

Insgesamt legt die Entwicklung der Policy-Heatmaps nahe, dass das Modell während des Trainings nicht nur einzelne Züge besser bewerten lernt, sondern schrittweise ein konsistentes und positionssensitives Spielverständnis entwickelt.

#### 4.3.4 Einordnung der analysierten Partie

Die in Abschnitt 3.6 beschriebene grafische Oberfläche erlaubt eine detaillierte Betrachtung einzelner Spielsituationen und liefert damit eine wertvolle Grundlage für die qualitative Evaluation des Modells. Die exemplarisch analysierte Partie zwischen dem Modell und einem menschlichen Spieler verdeutlicht typische Verhaltensmuster, strategische Prinzipien sowie das Zusammenspiel von Policy-Netz, Value-Schätzung und MCTS-Suche.

Bereits in der frühen Spielphase (Abbildung 4.5) zeigt sich ein zentrales Merkmal des Modells: Trotz eines vermeintlich schlechten Spielstands – nur ein eigener Stein auf dem Brett – trifft das Modell selbstbewusste Entscheidungen. Die Value-Schätzung ist deutlich positiv, die Policy fokussiert, und die Entropie gering. Dies deutet darauf hin, dass das Modell gelernt hat, kurzfristigen Steinbesitz zugunsten langfristiger Positionsvoorteile zu vernachlässigen – eine Strategie, die bereits theoretisch in Abschnitt 2.1.3 eingeführt wurde und vom Modell nun praktisch zum Einsatz kommt.

Im späten Mittelspiel (Abbildung 4.6) verlagert sich der Fokus des Modells klar auf strategisch entscheidende Felder – insbesondere die verbleibenden Ecken. Die Policy-Heatmap zeigt eine klare Priorisierung dieser Felder, und die Wahl eines Eckzugs wird vom Netz mit einer Value von 1.0 bestätigt. Das Modell agiert mit hoher Sicherheit (niedrige Entropie) und stützt sich dabei offenbar auf generalisierte Muster, die im Selbstspiel gelernt wurden. Diese Fähigkeit zur abstrakten Positionsbeurteilung ist besonders bemerkenswert, da während des Trainings keine explizite Codierung von Strategien wie „Ecken sichern“ vorgegeben wurde.

Die abschliessende Spielsituation (Abbildung 4.7) unterstreicht die Selektivität der MCTS-Suche in Verbindung mit der Netzwerkheuristik. Die Entropie beträgt 0.0, nur ein legaler Zug ist übrig, und die Policy weist diesem eine Wahrscheinlichkeit von 1.0 zu. Auffällig sind hier die MCTS-Metriken: Trotz einer Knotenzahl von 2000 zeigt die GUI eine sehr geringe maximale Tiefe (nur 2), eine ungewöhnlich hohe Knotengeschwindigkeit (11764 Knoten/s), sowie einem Effizienzfaktor < 0 und einer theoretischen Suchdauer unterhalb der realen Bedenkzeit.

Diese scheinbaren Widersprüche lassen sich dadurch erklären, dass im Endspiel bei nur noch

einem legalen Zug sämtliche Simulationspfade identisch verlaufen. Das Modell expandiert wiederholt denselben Weg bis zum Spielende – der gesamte verbleibende Suchbaum besteht effektiv nur aus einer Linie. Dadurch wird der gesamte Suchraum mehrfach durchquert, ohne zusätzliche neue Informationen zu generieren. In diesem Sonderfall verliert der Effizienzfaktor als Kennzahl seine Aussagekraft, und auch die theoretische Suchdauer wirkt verzerrt, da der expandierte Baum nicht mehr mit einem typischen Suchraum vergleichbar ist.

Insgesamt legen die beobachteten Spielzüge und Metriken nahe, dass das Modell gelernt hat, sowohl taktische als auch strategische Konzepte erfolgreich zu kombinieren. Die Policy konzentriert sich in kritischen Phasen auf bewährte Zugmuster (z.B. Ecken), während die Value-Funktion eine konsistente Bewertung des langfristigen Spielwerts liefert. Unterstützt durch eine fokussierte MCTS-Suche resultiert daraus ein Spielverhalten, das in mehreren Phasen überlegen gegenüber einem menschlichen Spieler erscheint.

Trotz der klaren Dominanz des Modells in der exemplarisch analysierten Partie sollte betont werden, dass eine einzelne Partie keine hinreichende Grundlage für eine abschliessende Bewertung der Spielstärke darstellt. Im Verlauf der Projektarbeit wurde das Modell jedoch wiederholt von verschiedenen menschlichen Spielern herausgefordert; gegen spätere Generationen gelang dabei keinem einzigen Spieler ein Sieg. Diese Beobachtung deutet darauf hin, dass das Modell eine überdurchschnittliche Spielstärke erreicht hat, die in vielen Spielsituationen auch erfahrene menschliche Gegner vor erhebliche Schwierigkeiten stellt.

#### 4.3.5 Limitationen und mögliche Verbesserungen

Trotz der insgesamt überzeugenden Resultate sind mehrere Einschränkungen zu berücksichtigen, die die Aussagekraft und Weiterentwicklung des Modells begrenzen.

Ein zentraler Aspekt betrifft die eingesetzte Modellarchitektur. Um Trainingszeit und Ressourcenbedarf gering zu halten, wurde ein vergleichsweise kompaktes Netzwerk mit lediglich vier ResNet-Blöcken verwendet. Zum Vergleich: In OLIVAW (Norelli & Panconesi, 2022) kamen zehn Residual-Blöcke zum Einsatz, wodurch eine deutlich grösse Ausdruckskraft des Netzwerks möglich wurde. Auch die anderen Netzwerkparameter – etwa die Kanaldimensionen, Batch-Grösse oder Replay-Buffer-Grösse – wurden zugunsten der Rechenzeit relativ konservativ gewählt.

Das Training selbst erfolgte auf einem einzelnen Rechner mit handelsüblicher Hardware (RTX 4060, Intel Core i7, Luftkühlung), was die maximal realisierbare Rechenlast erheblich begrenzte. Pro Iteration konnten lediglich etwa  $22 \times 12 = 264$  Self-Play-Partien durchgeführt werden – ein Bruchteil der Datenmenge, wie sie in vergleichbaren Arbeiten zur Anwendung kommt.

Auch der Monte Carlo Tree Search war durch eine variable, aber insgesamt begrenzte Anzahl

an Simulationen pro Zug limitiert. Die Anzahl der MCTS-Simulationen wurde in Abhängigkeit der Trainingsgeneration schrittweise erhöht – von anfänglich 100 bis maximal 400 Simulationen in späteren Generationen (vgl. Abbildung 3.6). Zwar stellt dies einen sinnvollen Kompromiss zwischen Rechenaufwand und Suchqualität dar, doch könnte die gewählte Obergrenze in den späteren Trainingsphasen zu einem Engpass geworden sein. Es ist denkbar, dass stärkere Modelle mehr Suchtiefe effektiv hätten nutzen können. Eine weitergehende Erhöhung oder adaptive Anpassung der Simulationsanzahl in Abhängigkeit der Modellstärke wäre ein potenzieller nächster Schritt zur weiteren Verbesserung.

Schliesslich ist auch die Evaluation nur bedingt vergleichbar mit etablierten Benchmarks. Die berechnete Elo basiert ausschliesslich auf Matches gegen Edax (Level 0–6) und ist daher als relative Spielstärke zu interpretieren. Eine absolute Einordnung wäre nur durch Partien gegen Engines mit offiziell kalibrierter Elo-Zahl oder gegen menschliche Spieler mit dokumentierter Wettkampferfahrung möglich gewesen.

Insbesondere offizielle Turnierpartien gegen lizenzierte Spieler mit Elo-Wert hätten die Bewertung zusätzlich gestützt und den Vergleich zur menschlichen Spielstärke präziser gemacht. Aufgrund technischer und organisatorischer Einschränkungen war dies im Rahmen dieser Arbeit jedoch nicht realisierbar.

Trotz all dieser Einschränkungen ist das erreichte Resultat bemerkenswert: Dass ein selbstlernendes Modell mit einer um Größenordnungen kleineren Datenmenge und deutlich geringerer Rechenleistung eine Spielstärke nahe Edax Level 6 erreicht und auch menschliche Spieler besiegt, zeigt die Wirksamkeit des gewählten Trainingsansatzes und die Fähigkeit des Modells, auch unter begrenzten Bedingungen sinnvolle Strategien zu entwickeln.

Insgesamt zeigt die Diskussion, dass das entwickelte Modell trotz technischer Einschränkungen und bewusst kompakter Architektur beachtliche Resultate erzielt hat. Die Bewertung des Trainingsverlaufs, der Spielstärke sowie des strategischen Verhaltens unterstreicht das Potenzial des selbstlernenden Ansatzes. Im folgenden Kapitel werden die wichtigsten Erkenntnisse zusammengefasst, die Forschungsfrage abschliessend beantwortet und ein Ausblick auf mögliche Weiterentwicklungen gegeben.

# Kapitel 5

## Fazit

### 5.1 Zusammenfassung

Im Rahmen dieser Arbeit wurde ein selbstlernender Spielalgorithmus für das Brettspiel Othello entwickelt, der sich am AlphaZero-Ansatz orientiert. Grundlage des Trainings war ein selbstgesteuerter Lernprozess: In zahlreichen Partien spielte das Modell gegen sich selbst, wobei die Zugauswahl durch eine Monte-Carlo Tree Search gesteuert wurde. Die Bewertung von Positionen sowie die Empfehlung vielversprechender Züge erfolgte dabei durch ein neuronales Netz, das im Laufe des Trainings kontinuierlich verbessert wurde.

Trainiert wurde das Modell über 55 Iterationen. Das verwendete neuronale Netz bestand aus vier Residual-Blöcken und wurde auf einem Einzelrechner mit begrenzter Rechenleistung trainiert. Die Anzahl der MCTS-Simulationen wurde im Verlauf des Trainings schrittweise erhöht.

Die Evaluation erfolgte anhand von Partien gegen die etablierte Engine Edax auf den Schwierigkeitsgraden 0 bis 6. Die berechnete Elo-Kurve zeigt eine stetige Verbesserung der Spielstärke bis auf einen Wert von etwa 1780. Ergänzend dazu zeigen Policy-Heatmaps, dass sich das strategische Verhalten des Modells über die Zeit systematisch verändert hat: Anfangs bevorzugte das Modell Rand- und Eckfelder, während später zunehmend zentrale Spielfeldbereiche im Fokus standen.

Trotz technischer Limitierungen gelang es dem Modell, eine Spielstärke nahe Edax Level 6 zu erreichen – ein Resultat, das die Wirksamkeit des gewählten Ansatzes unterstreicht. Ergänzend zur quantitativen Evaluation wurde eine exemplarische Partie gegen einen menschlichen Spieler analysiert. Dabei zeigte das Modell nicht nur strategische Zielstrebigkeit, sondern auch in verschiedenen Spielsituationen eine bemerkenswerte Klarheit in der Bewertung und Zugauswahl. Beobachtungen aus zahlreichen weiteren Spielen gegen menschliche Gegner stützen den

Eindruck, dass das Modell in vielen Fällen auch erfahrene Spieler übertrifft.

## 5.2 Beantwortung der Forschungsfrage

Die zentrale Forschungsfrage dieser Arbeit lautete:

*Wie gut performt ein selbst entwickelter AlphaZero-Algorithmus für Othello im direkten Vergleich zu menschlichen Spielern und Programmen unterschiedlicher Spielstärke?*

Basierend auf der durchgeführten Evaluation lässt sich diese Frage wie folgt beantworten:

Gegenüber klassischen Othello-Engines zeigt das Modell eine starke Leistung. In umfangreichen Matchreihen gegen Edax – eine anerkannte Referenz-Engine – konnte der entwickelte Algorithmus je nach Trainingsgeneration eine kontinuierliche Leistungssteigerung bis hin zu einer geschätzten Elo-Wertung von etwa 1780 erreichen. Damit bewegt sich das Modell etwas unterhalb der Spielstärke von Edax Level 6 und liegt deutlich über Programmen mit einfacher Heuristik.

Auch im Vergleich zu menschlichen Spielern zeigte sich das Modell überlegen. In einer exemplarisch analysierten Partie demonstrierte es langfristige strategische Planung, stabile Bewertung von Spielpositionen und eine präzise Zugwahl. Ergänzende Spiele gegen verschiedene menschliche Spieler im Rahmen der Projektarbeit verliefen ebenfalls zugunsten des Modells.

Diese Ergebnisse deuten darauf hin, dass der entwickelte Algorithmus – trotz bewusst kompakter Architektur und begrenzter Rechenressourcen – ein Spielverhalten entwickelt hat, das sowohl klassischen Strategien als auch menschlichen Spielern überlegen sein kann. Die verwendete Kombination aus Monte-Carlo Tree Search und neuronaler Bewertung erwies sich dabei als wirksamer Trainingsansatz für Othello.

Besonders bemerkenswert ist, dass diese Resultate mit vergleichsweise einfachen technischen Mitteln erzielt wurden. Der gesamte Trainings- und Evaluationsprozess fand auf einem handelsüblichen Heimrechner statt – ohne Zugang zu Grossrechenzentren oder externen Datenquellen. Dieses Ergebnis unterstreicht, dass nicht die schiere Rechenleistung, sondern die Qualität und Effizienz der gewählten Algorithmen den entscheidenden Unterschied machen können. Damit liefert die Arbeit nicht nur eine Antwort auf die Forschungsfrage, sondern auch ein Beispiel für ressourcenschonende und zugängliche KI-Entwicklung im Sinne einer *Green AI* (Schwartz et al., 2020).

Gleichzeitig ist festzuhalten, dass die Evaluation primär relativ zur Engine Edax und anhand weniger Partien gegen menschliche Gegner erfolgte. Eine absolute Einordnung der Spielstärke

wäre nur durch standardisierte Vergleichsturniere oder Matches gegen Spieler mit dokumentierter Elo-Zahl möglich gewesen. Innerhalb des Rahmens dieser Arbeit liefert die Evaluation dennoch eine klare Antwort: Das entwickelte Modell performt sowohl im Vergleich zu klassischen Programmen als auch gegenüber menschlichen Gegnern überzeugend.

### 5.3 Ausblick

Die in dieser Arbeit entwickelte Othello-KI zeigt, dass auch mit begrenzten Rechenressourcen ein leistungsfähiger selbstlernender Algorithmus realisiert werden kann. Dennoch ergeben sich aus der bisherigen Umsetzung zahlreiche Ansatzpunkte für weiterführende Arbeiten.

Ein naheliegender nächster Schritt besteht darin, die Modellarchitektur zu vergrössern – etwa durch den Einsatz zusätzlicher ResNet-Blöcke oder breiterer Layer –, um der Policy- und Value-Funktion eine höhere Repräsentationskraft zu verleihen. Damit einhergehend könnten auch längere Trainingsläufe mit einer grösseren Anzahl an Self-Play-Partien pro Iteration durchgeführt werden. Hierfür wären jedoch effizientere Implementierungen (z. B. mithilfe von Bitboards oder Low-Level-Optimierungen) sowie leistungsfähigere Hardware hilfreich.

Auch die Evaluationsmethoden lassen sich erweitern. Insbesondere direkte Partien gegen menschliche Spieler – etwa in schulischen Kontexten oder Online-Plattformen – könnten interessante Erkenntnisse liefern. Ebenso wäre ein Vergleich mit weiteren etablierten Othello-Engines denkbar, um die Spielstärke noch differenzierter einzurichten.

Darüber hinaus könnten gezielte Analysen des Spielstils Aufschluss darüber geben, welche taktischen und strategischen Prinzipien das Modell verinnerlicht hat – etwa durch Clusteranalysen typischer Spielzüge oder Visualisierungen von Entscheidungsbäumen innerhalb des MCTS.

Langfristig könnte das entwickelte Framework auch als Ausgangspunkt für verwandte Projekte dienen – etwa für andere Brettspiele oder als Lernplattform für selbstlernende Systeme mit begrenzten Ressourcen. Das Projekt macht deutlich, dass sich auch komplexe KI-Konzepte wie AlphaZero auf verständliche und eigenständig umsetzbare Weise erschliessen lassen.

Darüber hinaus eröffnet die Arbeit eine Perspektive, die über das konkrete Anwendungsfeld hinausweist: die Frage nach nachhaltiger, energieeffizienter KI. In einer Zeit, in der grosse KI-Modelle zunehmend mit massivem Ressourcenverbrauch einhergehen, wird der Ruf nach *Green AI* lauter – also nach Systemen, die auch mit begrenzter Hardware leistungsfähig, nachvollziehbar und zugänglich bleiben. Das hier entwickelte System steht exemplarisch für einen solchen Ansatz: Es zeigt, dass algorithmische Intelligenz, effiziente Architektur und systematische Planung in vielen Fällen entscheidender sein können als rohe Rechenpower. Solche Modelle könnten in Zukunft nicht nur in der Forschung, sondern auch in Bildung und praxisnaher KI-Entwicklung

eine wichtige Rolle spielen.

Wer sich vom Thema dieser Arbeit inspirieren lässt, ist herzlich eingeladen, den zugrunde liegenden Code weiterzuentwickeln, zu optimieren oder als Basis für eigene Projekte zu nutzen.<sup>1</sup>

*„Man sollte alles so einfach wie möglich machen. Aber nicht einfacher.“*

— Albert Einstein

---

<sup>1</sup>Der vollständige Quellcode ist öffentlich verfügbar unter: <https://github.com/LeoJu06/OthelloZero>

# Literatur

- 3Blue1Brown. (2017a, 5. August). *But what is a neural network?* [YouTube]. Verfügbar 14. August 2025 unter <https://www.youtube.com/watch?v=aircAruvnKk>
- 3Blue1Brown. (2017b, 6. August). *Backpropagation calculus* [YouTube]. Verfügbar 14. August 2025 unter <https://www.youtube.com/watch?v=tIeHLnjs5U8>
- Aaron Davis (**typedirector**). (2022, 24. November). *AlphaZero: An Introduction* [Video]. Verfügbar 8. Januar 2025 unter <https://www.youtube.com/watch?v=gbskPpoxGQk>
- AdamW — PyTorch 2.8 documentation.* (2024, 12. Juli). Verfügbar 10. August 2025 unter <https://docs.pytorch.org/docs/stable/generated/torch.optim.AdamW.html>, %20/generated/torch.optim.AdamW.html
- Amanatullah. (2023, 12. Juni). *Vanishing gradient problem in deep learning: Understanding, intuition, and solutions* [Medium]. Verfügbar 10. August 2025 unter <https://medium.com/@amanatulla1606/vanishing-gradient-problem-in-deep-learning-understanding-intuition-and-solutions-da90ef4ecb54>
- Browne, C. B., Powley, E., Whitehouse, D., Lucas, S. M., Cowling, P. I., Rohlfschagen, P., Tavener, S., Perez, D., Samothrakis, S., & Colton, S. (2012a). A Survey of Monte Carlo Tree Search Methods. *IEEE Trans. Comput. Intell. AI Games*, 4(1), 1–43. <https://doi.org/10.1109/TCIAIG.2012.2186810>
- Browne, C. B., Powley, E., Whitehouse, D., Lucas, S. M., Cowling, P. I., Rohlfschagen, P., Tavener, S., Perez, D., Samothrakis, S., & Colton, S. (2012b). A Survey of Monte Carlo Tree Search Methods. *IEEE Trans. Comput. Intell. AI Games*, 4(1), 1–43. <https://doi.org/10.1109/TCIAIG.2012.2186810>
- codebasics. (2020, 14. Oktober). *Simple explanation of convolutional neural network | Deep Learning Tutorial 23 (Tensorflow & Python)*. Verfügbar 12. August 2025 unter <https://www.youtube.com/watch?v=zfiSAzpy9NM>
- Coulom, R. (2006). Efficient Selectivity and Backup Operators in Monte-Carlo Tree Search. *Proceedings of the 5th International Conference on Computers and Games*, 72–83. [https://doi.org/10.1007/978-3-540-75538-8\\_7](https://doi.org/10.1007/978-3-540-75538-8_7)

- Deep Learning with Yacine. (2024, 9. April). *ResNet Deep Neural Network Architecture Explained*. Verfügbar 12. August 2025 unter <https://www.youtube.com/watch?v=woEs7UCaITo>
- Delorme, R. (2025, 26. Juli). *abulmo/edax-reversi* [original-date: 2015-04-25T19:46:17Z]. Verfügbar 27. Juli 2025 unter <https://github.com/abulmo/edax-reversi>
- Elo, A. E. (1978). *The Rating of Chessplayers, Past and Present*. Arco Publishing.
- Google DeepMind (**typedirector**). (2017, 18. Oktober). *AlphaGo Zero: Starting from Scratch* [Video]. Verfügbar 8. Januar 2025 unter <https://www.youtube.com/watch?v=tXIM99xPQC8>
- He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep Residual Learning for Image Recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 770–778. <https://doi.org/10.1109/CVPR.2016.90>
- Josh Varty (**typedirector**). (2020, 24. Mai). *Alpha Zero and Monte Carlo Tree Search* [Video]. Verfügbar 8. Januar 2025 unter <https://www.youtube.com/watch?v=62nq4Zsn8vc>
- Kocsis, L., & Szepesvári, C. (2006). Bandit Based Monte-Carlo Planning. *Machine Learning: ECML 2006*, 282–293. [https://doi.org/10.1007/11871842\\_29](https://doi.org/10.1007/11871842_29)
- Lex Fridman (**typedirector**). (2020, 3. April). *David Silver: AlphaGo, AlphaZero, and Deep Reinforcement Learning | Lex Fridman Podcast #86* [Video]. Verfügbar 8. Januar 2025 unter <https://www.youtube.com/watch?v=uPUEq8d73JI>
- MarbleScience (**typedirector**). (2020, 8. September). *Monte Carlo Simulation* [Video]. Verfügbar 8. Januar 2025 unter <https://www.youtube.com/watch?v=7ESK5SaP-bc>
- Mishra, M. (2020, 26. August). *Convolutional neural networks, explained* [Towards data science]. Verfügbar 11. August 2025 unter <https://towardsdatascience.com/convolutional-neural-networks-explained-9cc5188c4939/>
- Norelli, A., & Panconesi, A. (2022, 4. März). *OLIVAW: Mastering Othello without Human Knowledge, nor a Fortune* [Comment: Accepted for publication in IEEE Transactions on Games. Presented at AAAI-21 Reinforcement Learning in Games Workshop, 8 pages]. arXiv: 2103.17228 [cs]. <https://doi.org/10.48550/arXiv.2103.17228>
- Perrotta, P. (2020). *Programming Machine Learning*.
- Pytorch. (2025, 12. August). *OneCycleLR — PyTorch 2.8 documentation*. Verfügbar 12. August 2025 unter [https://docs.pytorch.org/docs/stable/generated/torch.optim.lr\\_scheduler.OneCycleLR.html](https://docs.pytorch.org/docs/stable/generated/torch.optim.lr_scheduler.OneCycleLR.html)
- Redaktion, I. (2024, 12. September). *Feedforward Neural Network* [IONOS Digital Guide]. Verfügbar 14. August 2025 unter <https://www.ionos.de/digitalguide/websites/webentwicklung/feedforward-neural-network/>

- Reversi Online – Tipps & Tricks für den Sieg gegen den Computer.* (2024, 15. Dezember). Verfügbar 29. Mai 2025 unter <https://www.boomer.at/article/reversi-online-tipps-tricks-fuer-den-sieg-gegen-den-computer.html>
- Rules & Strategy - HKOA.* (2022, 25. Juli). Verfügbar 29. Mai 2025 unter <https://www.othello.hk/rules-strategy/>
- Schwartz, R., Dodge, J., Smith, N. A., & Etzioni, O. (2020). Green AI. *Communications of the ACM*, 63(12), 54–63. <https://doi.org/10.1145/3381831>
- Sharma, V. (2021, 29. Januar). *ResNets: Why do they perform better than classic ConvNets? (conceptual analysis)* [Towards data science]. Verfügbar 11. August 2025 unter <https://towardsdatascience.com/resnets-why-do-they-perform-better-than-classic-convnets-conceptual-analysis-6a9c82e06e53/>
- Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., Lillicrap, T., Simonyan, K., & Hassabis, D. (2017a, 5. Dezember). Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm [Issue: arXiv:1712.01815]. <https://doi.org/10.48550/arXiv.1712.01815>
- Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., Lillicrap, T., Simonyan, K., & Hassabis, D. (2017b, 5. Dezember). *Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm*. arXiv: 1712.01815 [cs]. <https://doi.org/10.48550/arXiv.1712.01815>
- Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., Chen, Y., Lillicrap, T., Hui, F., Sifre, L., Van Den Driessche, G., Graepel, T., & Hassabis, D. (2017a). Mastering the game of Go without human knowledge. *Nature*, 550(7676), 354–359. <https://doi.org/10.1038/nature24270>
- Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., Chen, Y., Lillicrap, T., Hui, F., Sifre, L., Van Den Driessche, G., Graepel, T., & Hassabis, D. (2017b). Mastering the game of Go without human knowledge. *Nature*, 550(7676), 354–359. <https://doi.org/10.1038/nature24270>
- Stanford Online. (2024, 20. September). *Stanford CS149 I 2023 I Lecture 9 - Distributed Data-Parallel Computing Using Spark*. Verfügbar 12. August 2025 unter <https://www.youtube.com/watch?v=jaMWmLq422U>
- Świechowski, M., Park, H., Mańdziuk, J., & Kim, K.-J. (2015). Recent Advances in General Game Playing. *The Scientific World Journal*, 2015, 986262. <https://doi.org/10.1155/2015/986262>
- Takizawa, M. (2024, 1. Juni). *Othello programming*. Verfügbar 14. August 2025 unter [http://www.geocities.jp/othello\\_python/](http://www.geocities.jp/othello_python/)

- ThePrincipalComponent (**typedirector**). (2022, 31. Juli). *Recreating DeepMind's AlphaZero - AI Plays Connect 4 - Part 3: Monte Carlo Tree Search* [Video]. Verfügbar 8. Januar 2025 unter <https://www.youtube.com/watch?v=XnU-E7NQKX0>
- Wikipedia Contributors. (2025, 30. Juli). Elo rating system [Page Version ID: 1303332639]. In *Wikipedia*. Verfügbar 11. August 2025 unter [https://en.wikipedia.org/w/index.php?title=Elo\\_rating\\_system&oldid=1303332639](https://en.wikipedia.org/w/index.php?title=Elo_rating_system&oldid=1303332639)
- Wikipedia contributors. (2025a, 24. Juli). *Game*. Verfügbar 14. August 2025 unter <https://en.wikipedia.org/w/index.php?title=Game&oldid=257344800>
- Wikipedia contributors. (2025b, 24. Juli). *Rectifier (Aktivierungsfunktion)*. Verfügbar 14. August 2025 unter [https://de.wikipedia.org/w/index.php?title=Rectifier\\_\(Aktivierungsfunktion\)&oldid=257344777](https://de.wikipedia.org/w/index.php?title=Rectifier_(Aktivierungsfunktion)&oldid=257344777)