



FernUniversität in Hagen

– Fakultät für Mathematik und Informatik –

**Adaption von MCTS und AlphaZero für
Spiele mit imperfekter Information am
Beispiel Doppelkopf**

Abschlussarbeit zur Erlangung des akademischen Grades
Master of Science (M.Sc.)

vorgelegt von
Theodor Diesner-Mayer

Matrikelnummer: 6861920

Referentin : Prof. Dr. Uta Störl
Betreuer : Dr. Fabio Valdés

ABSTRACT

The automatic playing of board and computer games has traditionally served as an important testbed and application domain for advances in artificial intelligence (AI). While remarkable successes have already been achieved in games with perfect information, such as Go or Chess, games with imperfect information, where players can only observe part of the game state, still pose a significant challenge due to the resulting uncertainty.

This thesis investigates how the established search method *Monte Carlo Tree Search* (MCTS) and the learning algorithm *AlphaZero*, which in their basic form are only applicable to games with perfect information, can be extended to also work in games with imperfect information. To this end, a determinization approach is pursued: from a player's incomplete observation, one or more plausible complete game states are reconstructed. MCTS and AlphaZero are then applied to these reconstructed states. For the reconstruction, both a static, rule-based algorithm and a trainable autoregressive model based on the Transformer architecture are employed.

The proposed approach is evaluated using the German card game Doppelkopf as an example. As a trick-taking game with four players, who act in partially hidden partnerships, Doppelkopf offers strategic depth and serves as a suitable testbed for the developed methods. The evaluation of the resulting playing strength using several game-specific metrics shows that the autoregressive model reliably generates plausible states and that the determinization approach leads to a competent, albeit not optimal, playing strategy. The thesis concludes by discussing the conceptual and practical limitations of this approach and its applicability to other games.

ZUSAMMENFASSUNG

Das automatische Spielen von Brett- und Computerspielen dient traditionell als wichtiges Testfeld und Anwendungsgebiet für Fortschritte in der künstlichen Intelligenz (KI). Während für Spiele mit perfekter Information, wie Go oder Schach, bereits herausragende Erfolge erzielt wurden, stellen Spiele mit imperfekter Information, bei denen Spieler nur einen Teil des Spielzustands beobachten können, aufgrund der daraus resultierenden Unsicherheiten nach wie vor eine große Herausforderung dar.

Diese Arbeit untersucht, wie das etablierte Suchverfahren *Monte Carlo Tree Search* (MCTS) sowie das Lernverfahren *AlphaZero*, die in ihrer Grundform nur auf Spiele mit perfekter Information anwendbar sind, so erweitert werden können, dass sie auch auf Spiele mit imperfekter Information angewendet werden können. Hierfür wird der Ansatz der Determinisierung verfolgt: Aus der unvollständigen Beobachtung eines Spielers werden zunächst ein oder mehrere plausible vollständige Spielzustände rekonstruiert. Auf diesen rekonstruierten Zuständen werden anschließend der MCTS und AlphaZero ausgeführt. Für die Zustandsrekonstruktion kommen sowohl ein statischer, regelbasierter Algorithmus als auch ein trainierbares autoregressives Modell, basierend auf der Transformer-Architektur, zum Einsatz.

Der vorgestellte Ansatz wird exemplarisch am deutschen Kartenspiel Doppelkopf evaluiert. Als Stichspiel mit vier Spielern, die in teils unbekannten Partnerschaften agieren, bietet Doppelkopf strategische Tiefe und dient als geeignetes Testfeld für die entwickelten Methoden. Die Evaluierung der resultierenden Spielstärke anhand mehrerer spielspezifischer Metriken zeigt, dass das autoregressive Modell zuverlässig plausible Zustände generiert und der Determinisierungsansatz zu einer kompetenten, wenn auch nicht optimalen, Spielstrategie führt. Die Arbeit beleuchtet abschließend die konzeptionellen und praktischen Limitationen dieses Ansatzes und die Anwendbarkeit auf andere Spiele.

INHALTSVERZEICHNIS

1	Einleitung	1
1.1	Forschungsfrage	2
1.2	Aufbau	2
2	Grundlagen	4
2.1	Spieltheorie	4
2.1.1	Spiele in Extensivform	4
2.1.2	Nutzen	9
2.1.2.1	Ohne Zufallsereignisse	10
2.1.2.2	Mit Zufallsereignissen	10
2.1.3	Spiele in Strategieform	12
2.1.3.1	Reine Strategien	12
2.1.3.2	Strategieform	13
2.1.3.3	Nullsummenspiele	13
2.1.3.4	Dominanz	14
2.1.3.5	Nash-Gleichgewicht	15
2.1.3.6	Gemischte Strategien	16
2.1.3.7	Dominanz in gemischten Strategien	17
2.1.3.8	Nash-Gleichgewicht in gemischten Strategien	17
2.1.3.9	Verhaltensstrategien	18
2.2	Reinforcement Learning	19
2.2.1	Grundbegriffe	20
2.2.2	Vollständig beobachtbare Umgebungen	21
2.2.3	Partiell beobachtbare Umgebungen	23
2.2.4	Multi-Agent Reinforcement Learning	24
2.2.5	Modellbasierte und modellfreie Methoden	25
2.2.6	Funktionsapproximation	25
2.2.7	Zusammenhang zur Spieltheorie	27
3	Doppelkopf	28
3.1	Spielregeln	28
3.2	Abweichungen von den Spielregeln	29
3.2.1	Spiel statt Serie von Spielen	29
3.2.2	Vereinfachung der Vorbehaltspause	29
3.2.3	An- und Absagen als eigener Spielzug	29
3.2.4	Keine Spielabkürzungen	30
3.3	Doppelkopf in Extensivform	30
3.4	Grundlegende Spielstruktur	30
3.4.1	Informationsmengen	31
3.4.2	Nutzenfunktion	31
3.5	Doppelkopf als RL-Problem	32
3.6	Komplexität	32
3.6.1	Spielbaumkomplexität	32
3.6.2	Anzahl möglicher Spielzustände	33

4	Methoden und Algorithmen	34
4.1	Monte Carlo Tree Search	34
4.1.1	Grundform	34
4.1.1.1	Phasen in einer Iteration	35
4.1.1.2	Aktionsauswahl	36
4.1.2	Upper Confidence Bounds applied for Trees	36
4.1.3	Funktionsweise	38
4.1.4	Zusammenhang zur Spieltheorie	39
4.1.5	Zusammenhang zum Reinforcement Learning	39
4.2	Funktionsapproximation mittels neuronaler Netze	39
4.2.1	Multilayer Perceptron	40
4.2.1.1	Aufbau eines MLP	40
4.2.1.2	Universelle Approximation	42
4.2.2	Optimierung eines neuronalen Netzes	42
4.2.2.1	Verlustfunktion	43
4.2.2.2	Gradientenabstiegsverfahren	44
4.2.2.3	Backpropagation	45
4.2.3	Transformer-Architektur	46
4.2.3.1	Embeddings	47
4.2.3.2	Self-Attention und Multi-Head-Attention	48
4.2.3.3	Residuale Verbindungen	51
4.2.3.4	Layer Normalization	51
4.2.3.5	Zusammenspiel der Komponenten	52
4.3	AlphaZero	54
4.3.1	Funktionsapproximationen	55
4.3.1.1	Wertfunktion	55
4.3.1.2	Strategiefunktion	55
4.3.1.3	Architektur des KNN	55
4.3.2	MCTS	56
4.3.2.1	Wertfunktion statt Simulation	56
4.3.2.2	Priore Wahrscheinlichkeiten und PUCT	56
4.3.2.3	Aktionsauswahl	57
4.3.3	Ausführung des AlphaZero	57
4.3.4	Training der Funktionsapproximatoren	58
4.3.4.1	Spiel gegen sich selbst	58
4.3.4.2	Trainingsphase	59
4.3.5	Funktionsweise	59
4.3.6	Anpassung für mehrere Spieler	60
4.4	MCTS und AlphaZero in Spielen mit imperfekter Information	60
4.4.1	Determinisierung	61
4.4.1.1	Auswahl der Zustände	61
4.4.1.2	Strategie-Fusion	63
4.4.2	Behandlung von Zufallsereignissen	63
4.4.2.1	Zufallsereignisse im MCTS	63
4.4.2.2	Zufallsereignisse im AZ	63
4.4.2.3	Zufallsereignis im Wurzelknoten	63

4.5	Generative Modelle	64
4.5.1	Klassische Statistik	64
4.5.2	Unbedingte generative Modellierung	64
4.5.3	Bedingte generative Modellierung	65
4.5.4	Autoregressive Modelle	65
5	Determinisierung in Doppelkopf	67
5.1	Auswahl der Zustände	67
5.1.1	Autoregressives Modell	67
5.1.1.1	Strategieannahme	67
5.1.1.2	Schrittweise Generierung	68
5.1.1.3	Zustandsgenerierung	68
5.1.1.4	Training durch Rückwärtskonstruktion	69
5.1.1.5	Modelleingaben	69
5.1.1.6	Konsistenz durch Maskierung	69
5.1.2	CAP-Algorithmus	70
5.2	Einzelzustandsanalyse	71
5.2.1	Monte Carlo Tree Search	71
5.2.1.1	Simulationsheuristik	71
5.2.1.2	Einschränkung des Selektionsteilbaums	72
5.2.2	AlphaZero	73
5.2.2.1	Eingabemerkmale	73
5.2.2.2	Trainingsheuristik	73
5.2.2.3	Einschränkung des Selektionsteilbaums	74
5.2.3	Zufallsereignisse	74
5.3	Strategie-Fusion	74
5.3.1	Mittels kumulierten Rangs	74
5.3.2	Mittels durchschnittlicher Strategie	75
5.4	Zusammensetzung der Komponenten	76
5.4.1	AlphaZero mit autoregressivem Modell	76
5.4.2	MCTS mit autoregressivem Modell	78
5.4.3	MCTS mit CAP-Algorithmus	78
6	Implementation	79
6.1	Spielsimulator für Doppelkopf	79
6.2	Implementierung von MCTS	79
6.3	Implementierung von AlphaZero	80
6.3.1	Architektur der Funktionsapproximatoren	80
6.3.1.1	Kodierung der Eingaben	80
6.3.1.2	Architektur des neuronalen Netzes	81
6.3.2	Heuristiken	82
6.3.3	Performance-Optimierungen	82
6.3.4	Parameter des verwendeten Modells	84
6.4	Implementierung des autoregressiven Modells	85
6.4.1	Funktionsapproximationen	85
6.4.1.1	Kodierung der Eingaben	85
6.4.1.2	Architektur des neuronalen Netzes	85
6.4.2	Training mit Strategie	85

6.4.3	Parameter der verwendeten Modelle	86
6.5	Determinisierung	86
6.6	Web-Anwendung zur selektiven Evaluation	87
7	Evaluation	89
7.1	Evaluationsmetriken	89
7.2	Evaluationsvorgehen	90
7.3	MCTS mit perfekter Information	90
7.3.1	Ermittlung geeigneter Parameter	91
7.3.1.1	Experimentelles Vorgehen	91
7.3.1.2	Versuch 1: Breite Streuung	91
7.3.1.3	Versuch 2: Engere Auswahl	91
7.3.2	MCTS gegen Zufallsspieler	92
7.3.3	MCTS gegen MCTS	93
7.3.3.1	Leistungssteigerung mit Iterationen	94
7.3.3.2	Einfluss des UCT-Parameters	95
7.3.3.3	Spielmodi-Verteilung	96
7.3.3.4	Verteilung der An- und Absagen	96
7.3.4	Analyse einzelner Spiele	97
7.4	AlphaZero mit perfekter Information	99
7.4.1	Trainingsprozess und Evaluation	99
7.4.2	AlphaZero gegen MCTS	102
7.5	Training des autoregressiven Modells	103
7.5.1	Trainingsprozess und Evaluation	103
7.5.2	Evaluation einzelner Vorhersagen	104
7.5.2.1	Spielverlauf 1: Farbenspiel ohne Ass	104
7.5.2.2	Spielverlauf 2: Ansage eines Solos	105
7.6	Imperfekte Information	106
7.6.1	Vergleich verschiedener Determinisierungsansätze	106
7.6.2	Einfluss der Stichprobengröße	107
7.6.3	Entwicklung im Selbstspiel	107
7.6.4	Analyse einzelner Spiele	109
7.6.4.1	Spiel 1: Starkes Re-Blatt	109
7.6.4.2	Spiel 2: Ausgeglichenes Spiel	110
8	Diskussion	111
8.1	Spielstärke	111
8.2	Einzelzustandsanalyse	112
8.2.1	Mangelnde Exploration	112
8.2.2	Wiederholte Entscheidungssituationen	113
8.2.3	AlphaZero und MCTS	114
8.3	Autoregressives Modell	115
8.4	Ansatz der Determinisierung	116
8.4.1	Stichprobengröße	116
8.4.2	Strategie-Fusion und Nicht-Lokalität	117
8.4.3	Gemischte Strategien	118
8.4.4	Opponent Modeling	119
8.4.5	Kommunikation	119

8.5 Zielerreichung	119
8.6 Fazit	120
I Appendix	
A Ergänzende Abbildungen	122
Literatur	125

ABBILDUNGSVERZEICHNIS

Abbildung 2.1	Ausschnitt aus der Extensivform des Spiels TTT	6
Abbildung 2.2	Ein Ausschnitt des Spielbaums von Kuhn-Poker (KP) [Wik16].	9
Abbildung 2.3	Vergleich RL vs. MARL	21
Abbildung 2.4	Mittels Funktionsapproximation kann der RL-Algorithmus auf andere Zustände generalisieren.	26
Abbildung 4.1	Die vier Phasen des MCTS [Świ+23].	35
Abbildung 4.2	Ein Ausschnitt eines Selektionsteilbaum in der Vorberhaltsphase eines Doppelkopf-Spiels.	38
Abbildung 4.3	Vom einzelnen Neuron zum vollständig verbundenen Mehrschicht-Perceptron (MLP).	42
Abbildung 4.4	Embedding-Layer für Karten aus Doppelkopf.	48
Abbildung 4.5	Self-Attention-Matrix bei der Ermittlung der Stichgewinner.	50
Abbildung 4.6	Ein einzelner Transformer-Block für Eingaben mit Embedding-Dimension d_e	53
Abbildung 4.7	Schematische Darstellung der Determinisierung.	62
Abbildung 5.1	Die Zusammensetzungen der unterschiedlichen Determinisierungsansätze.	77
Abbildung 6.1	Die Architektur des verwendeten NN für den AlphaZero-Algorithmus im Spiel Doppelkopf.	83
Abbildung 6.2	Die verwendeten Hyperparameter für das AZ-Modell sowie das autoregressive Modell.	84
Abbildung 6.3	Web-Anwendung zur selektiven Evaluation.	88
Abbildung 7.1	Durchschnittliche Punktzahl der MCTS-Strategie gegen die jeweilige MCTS-Strategie mit weniger Iterationen ($C = 2,5$).	94
Abbildung 7.2	Die Durchschnittspunktzahl in Vergleichsspielen zwischen $C \in \{2,5, 3,75, 5,0\}$ für verschiedene Iterationszahlen. (volle Ergebnisse: Tabelle A.1)	94
Abbildung 7.3	Entwicklung der Solo-Quoten im Selbstspiel des Monte-Carlo Tree Search (MCTS) Verlauf zunehmender Iterationsanzahl.	95
Abbildung 7.4	Entwicklung der Ansage-Quoten im Selbstspiel des MCTS im Verlauf zunehmender Iterationszahlen.	95
Abbildung 7.5	Verteilung der Spielmodi und An- und Absagen der Parteien im Selbstspiel mit verschiedenen Iterationen ($C = 2,5$)	96
Abbildung 7.6	Zwei Spielverläufe im Selbstspiel MCTS mit $I = 1.000.000$ und $C = 2,5$	97

Abbildung 7.7	Die Verteilung der Besuche in zwei Zügen des ersten Spiels.	98
Abbildung 7.8	Der geglättete Verlust des gemeinsamen Strategie- und Wertennetzwerks des AlphaZero (AZ)-Agenten.	100
Abbildung 7.9	Durchschnittliche Punktzahl des AZ-Agenten im Trainingsverlauf gegen einen MCTS-Spieler mit 1.000 Iterationen.	100
Abbildung 7.10	Verteilung von Spielmodi sowie An- und Absagen des AZ-Spielers gegen einen MCTS-Spieler.	101
Abbildung 7.11	Verhältnis konsistenter zu inkonsistenten Zuständen im Verlauf des Trainings des autoregressiven Modells. . .	104
Abbildung 7.12	Die Verteilung der Spielmodi und An- und Absagen bei Selbstspiel Strategie $\pi_{\text{MCTS}^{\text{II}}}^{\text{AR}}$ (avg, $C = 2,5$, $n_{\text{sample}} = 25$).	108
Abbildung 7.13	Zwei Spielverläufe der Strategie $\pi_{\text{MCTS}^{\text{II}}}^{\text{AR}}$ (avg, $C = 2,5$, $n_{\text{sample}} = 25$, $I = 1.000.000$).	109
Abbildung A.1	Die Dirichlet-Verteilung von einigen Beispielen [Sus].	122
Abbildung A.2	Die Architektur des künstliche neuronale Netze (NN) für das autoregressive Modell.	123
Abbildung A.3	Verlustverlauf des autoregressiven Modells.	123

TABELLENVERZEICHNIS

Tabelle 2.1	Das Spiel Schere-Stein-Papier (SSP) in Strategieform.	13
Tabelle 2.2	Ein Spiel in Strategieform, welches das Konzept der strikten und schwachen Dominanz zeigt.	15
Tabelle 2.3	Das Gefangenendilemma mit einem Nash-Gleichgewicht.	16
Tabelle 5.1	Ein Beispiel der Fusionsstrategie mittels kumulierten Rangs.	75
Tabelle 7.1	Metriken des MCTS im Vergleichsspiel mit unterschiedlichen Upper Confidence Bounds applied for Trees (UCT)-Werten bei $I = 100.000$ (I).	91
Tabelle 7.2	Metriken des MCTS im Vergleichsspiel mit unterschiedlichen UCT-Werten bei $I = 100.000$ (II).	92
Tabelle 7.3	Metriken des MCTS mit unterschiedlichen Iterationswerten bei $C = 2,5$	92
Tabelle 7.4	Die erste Entscheidung von Spieler <i>Unten</i> zugunsten eines Solos.	99
Tabelle 7.5	Durchschnittliche Punktzahl des trainierten AZ-Agenten im Vergleichsspiel mit MCTS-Strategien mit unterschiedlicher Iterationszahl und $C = 2,5$	103
Tabelle 7.6	Vergleich der vollständigen Determinisierungsstrategien mit unterschiedlicher Parametrisierung.	106
Tabelle 7.7	Untersuchung der Strategie $\pi_{\text{MCTS}^{\text{II}}}^{\text{AR}}$ ($\text{avg}, C = 2,5$) hinsichtlich der Stichprobengröße.	107
Tabelle 7.8	Ergebnisse der Strategie $\pi_{\text{MCTS}^{\text{II}}}^{\text{AR}}$ ($\text{avg}, C = 2,5, n_{\text{sample}} = 25$) im Selbstspiel.	108
Tabelle A.1	Vergleichsspiele zwischen $C \in \{2,5, 3,75, 5,0\}$ für verschiedene Iterationszahlen.	124

ABKÜRZUNGSVERZEICHNIS

- KP Kuhn-Poker
SSP Schere-Stein-Papier
KI Künstliche Intelligenz
RL Reinforcement Learning
DP Decision Process
MDP Markov Decision Process
POMDP Partially observable Markov Decision Procss
MA-MDP Multi-Agent Markov Decision Process
MA-POMDP Multi-Agent Partially observable Markov Decision Process
MC Monte-Carlo
MARL Multi-Agent Reinforcement Learning
NN künstliche neuronale Netze
TTT Tic-Tac-Toe
AZ AlphaZero
PI-MCTS Perfect Information Monte Carlo Tree Search
MCTS Monte-Carlo Tree Search
CAP Card-Assignment-Problem
UCT Upper Confidence Bounds applied for Trees
PUCT Predictor + UCT
AZ-MCTS AZ-MCTS
MLP Mehrschicht-Perceptron
UAT Universelles Approximationstheorem
CNN Convolutional Neural Network
CE Cross Entropy
MSE mittlere quadratische Abweichung
ML Machine Learning
CFR Counterfactual Regret Minimization
MC-CFR Monte-Carlo Counterfactual Regret Minimization
Softmax Softmax

EINLEITUNG

Seit den Anfängen der Künstlichen Intelligenz (KI) bis zum heutigen Zeitpunkt ist das automatische Spielen von Computer- und Brettspielen ein zentrales und beliebtes Anwendungsgebiet. In dieser Historie dienten und dienen Spiele nicht nur der Unterhaltung, sondern auch als Benchmark für neuartige und verbesserte KI-Systeme und Algorithmen. Diese Popularität als Benchmark lässt sich gut erklären: Spiele bieten eine in sich geschlossene und kontrollierbare Umgebung mit klar definierten Regeln und Zielen. Dadurch lassen sich Spiele einfach, akkurat und schnell simulieren. Darüber hinaus haben Spiele klare Bewertungsregeln. Die Folge dieser Eigenschaften sind schnelle Experimentierzyklen und die Möglichkeit für klare Vergleiche zwischen verschiedenen Ansätzen. Gleichzeitig weisen (viele) Spiele eine hohe strategische Komplexität auf und erfordern komplexe Entscheidungsprozesse, was die Anwendung fortgeschrittener Techniken der KI für starke Strategien notwendig macht.

Erwähnenswerte Meilensteine dieser Geschichte von KI in (Brett-)spielen sind beispielsweise: Deep Blue [CHH02] von IBM, ein Schach-Programm, welches 1997 erstmals den amtierenden Weltmeister Garri Kasparow besiegte. Ein weiterer Meilenstein war 2006 die Einführung des Monte-Carlo Tree Search (MCTS), einer Suchmethode, die in Spielen wie Go große Erfolge erzielte [Cou06].

Basierend auf dieser Technik entwickelte Google DeepMind das System AlphaGo [Sil+16], welches 2016 den weltbesten Go-Spieler Lee Sedol schlug. Das ist insofern bemerkenswert, als Go im Vergleich zu Schach eine deutlich höhere Komplexität aufweist. AlphaGo Zero [Sil+17b] und kurze Zeit später AlphaZero (AZ) [Sil+17a] waren Weiterentwicklungen von AlphaGo, welche ausschließlich durch Selbstspiel (ohne menschliche Partien als Grundlage) lernten und eine noch höhere Spielstärke erreichten.

Während frühe Systeme noch im Wesentlichen auf spielspezifischen und regelbasierten Techniken beruhten, verfolgen moderne Ansätze das Ziel, weitestgehend ohne manuelle Anpassungen für ein bestimmtes Spiel nur durch Interaktion mit der Spielumgebung effektive Strategien zu erlernen. Genau dieses Problemfeld des Reinforcement Learning (RL) bildet neben dem klassischen Suchverfahren des MCTS den Schwerpunkt dieser Arbeit.

Verfahren wie der MCTS und AZ haben sich als leistungsfähige Algorithmen für die Entwicklung von Strategien in Spielen mit perfekter Information gezeigt. Das sind solche Spiele, wie zum Beispiel Schach und Go, in denen alle Spieler mit dem gleichen Wissensstand über das Spielgeschehen agieren. Viele Spiele geben den Spielern jedoch nur imperfekte Information preis. In diesem Fall kennen die Spieler nicht den vollständigen Zustand des Spiels, sondern nur einen Ausschnitt davon. In vielen Kartenspielen kennen die ein-

zernen Spieler beispielsweise nur ihre eigenen Karten, nicht aber die der anderen Spieler. Spiele mit imperfekter Information stellen eine besondere Schwierigkeit dar, da die tatsächlich vorliegenden Spielzustände nur probabilistisch eingeschätzt oder bestimmt werden können.

1.1 FORSCHUNGSFRAGE

Diese Arbeit soll sich mit der Frage beschäftigen, wie die eng miteinander verwandten Algorithmen Monte-Carlo Tree Search (MCTS) und AlphaZero (AZ) so erweitert werden können, dass sie auch auf Spiele mit imperfekter Information angewendet werden können. Das zentrale Ziel der Algorithmen ist es, eine möglichst effektive Strategie für ein Spiel mit perfekter Information zu ermitteln. In einfachen Worten sollen die Algorithmen eine Strategie entwickeln, mit der ein Spiel möglichst *gut* gespielt wird. Um die Algorithmen auf Spiele mit imperfekter Information anzuwenden, wird ein Ansatz verfolgt, der auf Determinisierung beruht: Aus der eigenen Beobachtung des Spielgeschehens soll dabei der tatsächlich herrschende Spielzustand rekonstruiert werden, um das Spiel im Anschluss das Spiel als Spiel mit perfekter Information betrachten zu können.

Zur Veranschaulichung wird das eingesetzte Verfahren schrittweise anhand des Kartenspiels Doppelkopf aufgebaut, erläutert und experimentell evaluiert. Abschließend wird diskutiert, inwieweit sich der Ansatz auf andere Spiele übertragen lässt. Dabei soll sich der Ansatz von rein spielspezifischen und regelbasierten Methoden abgrenzen. Der Ansatz bewegt sich im Kontext des Reinforcement Learning (RL), dessen Ziel es ist, allgemein anwendbare Techniken zu entwickeln, die sich prinzipiell auch auf andere Spiele übertragen lassen und die Spielstrategien im Wesentlichen aus der Interaktion mit der Spielumgebung selbst ableiten.

1.2 AUFBAU

Zur Beantwortung der Forschungsfrage gliedert sich die Arbeit wie folgt:

In Kapitel 2 werden die Grundlagen, um das Konzept des „guten Spielens“ theoretisch zu fundieren, dargelegt. Der Abschnitt zur Spieltheorie gibt an, wie ein Spiel definiert wird, was Strategien sind, wie Strategien bewertet werden („gut spielen“) und erläutert den Unterschied zwischen Spielen mit perfekter und imperfekter Information. Da optimale Strategien für komplexe Spiele wie Doppelkopf nicht direkt berechenbar sind, führt der zweite Abschnitt des Grundlagenkapitels in das sogenannte Reinforcement Learning (RL) ein. Anstelle einer direkten Berechnung optimaler Züge beschreibt das RL Methoden für die sukzessive Verbesserung des Spielverhaltens durch Interaktion mit einer Spielumgebung. Das Kapitel definiert das zentrale RL-Problem und die wesentlichen Begriffe dieses lernbasierten Ansatzes zur Ermittlung „guter“ Strategien.

Aufbauend auf diesen Grundlagen stellt Kapitel 3 das Kartenspiel Doppelkopf vor, das als zentrales Anwendungsbeispiel dient. Es formalisiert das

Spiel sowohl in spieltheoretischer Hinsicht als auch als Umgebung des RL, damit die nachfolgenden Algorithmen und Methoden konkret angewendet werden können.

Kapitel 4 widmet sich den Kernmethoden dieser Arbeit. Es führt die Algorithmen Monte-Carlo Tree Search (MCTS) und AlphaZero (AZ) ein. Anschließend werden grundlegende Ansätze vorgestellt, um diese Algorithmen für Spiele mit imperfekter Information anzupassen, wobei der Fokus auf dem hier verfolgten Determinisierungsansatz mit seinen drei Komponenten Zustandsauswahl, Einzelzustandsanalyse und Strategie-Fusion liegt.

Die konkrete Anwendung der Determinisierung auf das formalisierte Doppelkopf-Spiel ist Gegenstand von Kapitel 5. Hier wird detailliert dargelegt, wie die zuvor erläuterten Einzelkomponenten zusammenspielen, um verschiedene Strategien für Doppelkopf zu bilden.

Kapitel 6 beschreibt die technische Implementierung der Determinisierung und aller für die Evaluation benötigten Komponenten, einschließlich des Spielsimulators und der neuronalen Netzarchitekturen. Die Evaluation des entwickelten Ansatzes und seiner Teilkomponenten erfolgt in Kapitel 7. Es präsentiert die Ergebnisse der durchgeführten Untersuchungen. Abschließend interpretiert die Diskussion in Kapitel 8 die Evaluationsergebnisse im Kontext der Forschungsfrage. Sie analysiert kritisch die Stärken und Schwächen des gewählten Ansatzes und prüft dessen Übertragbarkeit auf andere Spiele mit imperfekter Information.

2

GRUNDLAGEN

In diesem Kapitel werden die Grundlagen aus der Spieltheorie und dem Reinforcement Learning (RL) gelegt, die für das Verständnis des Problems notwendig sind.

2.1 SPIELTHEORIE

Die Spieltheorie ist ein mathematischer Ansatz zur Modellierung und Analyse von Entscheidungssituationen, in denen mehrere Akteure, sogenannte Spieler, miteinander interagieren. Die Spieler verfolgen in diesen Entscheidungssituationen unterschiedliche Ziele und die Entscheidungen der Spieler haben Auswirkungen auf das Ergebnis der anderen Spieler. In solchen Situationen untersucht die Spieltheorie das optimale Verhalten der Spieler.

Obwohl der Begriff „Spiel“ verwendet wird, beschränkt sich die Spieltheorie nicht auf Spiele im Sinne der Freizeitbeschäftigung. Vielmehr lassen sich zahlreiche reale Entscheidungssituationen als spieltheoretische Modelle formulieren und analysieren.

Die folgenden Ausführungen und Definitionen beruhen, soweit nicht anders angegeben, auf dem Lehrbuch *Game Theory* [MSZ20].

2.1.1 Spiele in Extensivform

Ein Spiel kann in der Spieltheorie in der sogenannten Extensivform (extensive-form game) beschrieben werden. Diese Darstellung ermöglicht eine vollständige Modellierung der Spielregeln, einschließlich der erlaubten Züge, der verfügbaren Informationen eines Spielers in einem bestimmten Spielzustand, der Reihenfolge der Spielzüge und der möglichen Ergebnisse. Auch vom Zufall abhängige Ereignisse können modelliert werden.

SPIELE MIT PERFEKTER INFORMATION Ein Spiel, in dem allen Spielern der komplette Spielzustand bekannt ist, nennt man ein Spiel mit perfekter Information.

Definition 1 (Spiel mit perfekter Information) Ein Spiel mit perfekter Information in Extensivform ist ein geordnetes Tupel

$$\Gamma = (N, V, E, x^0, (V_i)_{i \in N}, O, u)$$

für das gilt:

- N ist eine endliche Menge von Spielern.

- (V, E, x^0) ist ein gerichteter Baum, der als Spielbaum bezeichnet wird.
- $(V_i)_{i \in N}$ ist eine Partition der Menge von Knoten des Spielbaums, die keine Blätter sind.
- O ist eine Menge der möglichen Spielausgänge (outcomes).
- u ist eine Funktion, die jedes Blatt des Spielbaums auf einen Spielausgang abbildet.

Der Spielbaum (V, E, x^0) ist ein Baum. Dementsprechend ist V die Knotenmenge und E die Menge der gerichteten Kanten zwischen den Knoten. Als Baum gibt es zwischen dem Wurzelknoten x^0 und einem Knoten $x \in V$ nur einen eindeutigen Pfad. Die Knotenmenge V enthält alle erreichbaren Spielzustände. Die Partition $(V_i)_{i \in N}$ gibt für jeden Spieler i an, in welchen Spielzuständen er an der Reihe ist und die nächste Aktion auswählt. Der Spieler wird in diesem Zustand Entscheidungsträger am Knoten x genannt und mit $J(x)$ bezeichnet.

Die Kinder eines Knotens x aus V , die keine Blätter sind, stellen die von x aus erreichbaren Spielzustände dar und werden auch als $C(x)$ bezeichnet. Eine Kante $(x, x') \in E$ zwischen zwei Spielzuständen x, x' entspricht einer möglichen Aktion, die einen Übergang von x nach x' bewirkt. Die Menge der möglichen Aktionen an einem Knoten, die ein Spieler in diesem Spielzustand auswählen kann, wird mit $A(x)$ bezeichnet. In einfachen Worten gibt eine Kante eine mögliche Aktion eines Spielers in einem Spielzustand und den daraus resultierenden Spielzustand an. x^0 ist der Start-Spielzustand. Die Blätter des Spielbaums sind die erreichbaren Endzustände des Spiels, die zu dem durch u definierten Spielausgang aus O führen.

Eine Spielsequenz kann dann als Sequenz x^0, x^1, \dots, x^k von Knoten aus V betrachtet werden. Dabei ist x^0 der Wurzelknoten des Spielbaums, x^k ist ein Blatt und es gilt $x^{l+1} \in C(x^l)$ für $l = 0, 1, \dots, k - 1$. In einfachen Worten kann eine Spielsequenz durch eine erlaubte Abfolge von Spielzuständen vom Start-Spielzustand zum End-Spielzustand beschrieben werden.

Bei einem Spiel mit perfekter Information weiß ein Spieler i in welchem Zustand $x \in V$ sich das Spiel befindet. In Konsequenz kennt er auch alle bis zum Zustand x gewählten Aktionen von sich selbst und den anderen Spielern.

Beispiel 1 (Tic-Tac-Toe in Extensivform) Tic-Tac-Toe (TTT) ist ein Spiel mit perfekter Information. Die Spieler sind $N = \{X, O\}$ und V ist die Menge aller möglichen Spielfeldkonfigurationen, die durch erlaubte Züge möglich sind. x^0 ist das leere Spielfeld. Die Kanten E geben die von einem Spielzustand $x \in V$ möglichen Aktionen an; also die Spielfelder, in denen der aktuelle Spieler seinen Buchstaben noch eintragen kann. Die möglichen Spielausgänge $O = \{X \text{ gewinnt}, \text{Unentschieden}, O \text{ gewinnt}\}$ werden mittels u den Blattknoten zugeordnet, in denen ein Gewinn oder Unentschieden eingetreten ist. V_X und V_O definieren die Zustände, in denen X beziehungsweise O an der Reihe ist und sind in Abbildung 2.1 farblich markiert.

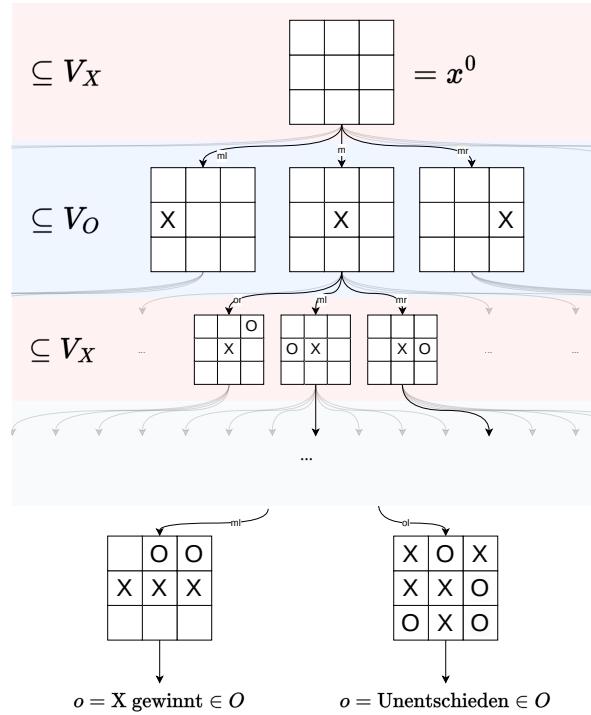


Abbildung 2.1: Ausschnitt aus der Extensivform des Spiels TTT.

SPIELE MIT ZUFALLSEREIGNISSEN Viele Spiele beinhalten Zufallsereignisse, beispielsweise das Mischen des Kartendecks in Kartenspielen oder das Würfeln in Brettspielen. Solche Spiele lassen sich durch eine Erweiterung der Extensivform modellieren.

Definition 2 (Spiel mit perfekter Information und Zufallsereignissen)
Ein Spiel mit perfekter Information und mit Zufallsereignissen in Extensivform ist ein geordnetes Tupel

$$\Gamma = (N, V, E, x^0, (V_i)_{i \in N \cup \{0\}}, (p_x)_{x \in V_0}, O, u)$$

für das gilt:

- N ist eine endliche Menge von Spielern.
- (V, E, x^0) ist der Spielbaum.
- $(V_i)_{i \in N \cup \{0\}}$ ist eine Partition der Knotenmenge des Spielbaums, die keine Blätter sind.
- Für jeden Knoten $x \in V_0$ ist p_x eine Wahrscheinlichkeitsverteilung über alle ausgehenden Kanten von x .
- O ist eine Menge der möglichen Spieldausträge.
- u ist eine Funktion, die jedes Blatt des Spielbaums auf einen Spieldaustrag abbildet.

In dieser Definition wird das Spiel um einen Spieler 0 erweitert, welcher die Zufallsereignisse repräsentiert. Dieser Spieler ist Entscheidungsträger an den Knoten aus V_0 . In diesen Spielzuständen wählt er den nachfolgenden Spielzustand anhand der in p_x definierten Wahrscheinlichkeitsverteilung. Anders ausgedrückt: Das Spiel wird um einen weiteren Spieler, den Zufall (in [MSZ20] als *Nature* bezeichnet), erweitert, der seine Züge entsprechend einer Wahrscheinlichkeitsverteilung zufällig trifft.

SPIELE MIT IMPERFEKTER INFORMATION Spiele mit imperfekter Information zeichnen sich dadurch aus, dass ein Spieler nicht (immer) weiß, in welchem Spielzustand sich das Spiel gerade befindet. Das bedeutet, dass er möglicherweise auch die zuvor von anderen Spielern ausgeführten Aktionen nicht kennt. Um solche Spiele zu modellieren, müssen die vorherigen Definitionen um das Konzept der Informationsmengen erweitert werden.

Definition 3 (Informationsmenge) Für ein Spiel

$$\Gamma = (N, V, E, x^0, (V_i)_{i \in N \cup \{0\}}, (p_x)_{x \in V_0}, O, u)$$

in Extensivform gilt: Eine Informationsmenge eines Spielers i ist ein Paar $(U_i^j, A(U_i^j))$, wenn gilt:

- $U_i = \{x_i^1, x_i^2, \dots, x_i^j\}$ ist eine Teilmenge von V_i , wobei von jedem Spielzustand $x \in U_i$ die gleiche Anzahl von Aktionen $l_i = l_i(U_i)$ ausgehen. Anders ausgedrückt:

$$|A(x_i^j)| = l_i, \forall j = 1, 2, \dots, m.$$

- $A(U_i)$ ist die Menge der möglichen Aktionen des Spielers i in der Informationsmenge U_i . Sie entsteht durch eine Partitionierung der Menge der $m l_i$ Kanten $\bigcup_{j=1}^m A(x_i^j)$ in l_i disjunkte Äquivalenzklassen, von der jede genau eine Aktion aus jeder Menge $A(x_i^j)$ enthält. Die Elemente der Partition werden mit $a_i^1, a_i^2, \dots, a_i^{l_i}$ bezeichnet.

Eine Informationsmenge $(U_i^j, A(U_i^j))$ eines Spielers beschreibt die Menge an Spielzuständen, die aus seiner Perspektive aktuell möglich sind. Der Spieler kennt den tatsächlichen Zustand nicht. Er weiß nur, dass sich das Spiel in irgendeinem Spielzustand aus der Informationsmenge befindet.

Da der Spieler nicht zwischen den einzelnen Zuständen in U_i^j unterscheiden kann, muss er in allen diesen Zuständen dieselbe Entscheidung treffen. Dies bedeutet, dass alle Aktionen aus einer Informationsmenge in Äquivalenzklassen gruppiert werden. Jede Äquivalenzklasse enthält genau eine Aktion aus jedem Zustand $x \in U_i^j$. Die Aktionsmenge $A(U_i^j)$ besteht dann aus diesen Äquivalenzklassen, nicht aus individuellen Aktionen für jeden möglichen Zustand.

Sieht der Spieler in einem Kartenspiel beispielsweise nur seine eigene Hand und hat er die Möglichkeit jede dieser Karten auszuspielen, dann entspricht jede Karte einer Äquivalenzklasse von Aktionen. Jede dieser Äquivalenzklassen enthält genau eine ausgehende Kante aus jedem Zustand innerhalb der Informationsmenge, die das Spielen dieser Karte in diesem Zustand repräsentiert.

Definition 4 (Spiel mit imperfekter Information) Ein Spiel mit imperfekter Information und mit Zufallsereignissen in Extensivform ist ein geordnetes Tupel

$$\Gamma = (N, V, E, x^0, (V_i)_{i \in N \cup \{0\}}, (p_x)_{x \in V_0}, (U_i^j)_{i \in N}^{j=1, \dots, k_i}, O, u)$$

für das gilt:

- N ist eine endliche Menge von Spielern.
- (V, E, x^0) ist der Spielbaum.
- $(V_i)_{i \in N \cup \{0\}}$ ist eine Partition der Menge von Knoten des Spielbaums, die keine Blätter sind.
- Für jeden Knoten $x \in V_0$ ist p_x eine Wahrscheinlichkeitsverteilung über alle ausgehenden Kanten von x .
- Für jeden Spieler $i \in N$ ist $(U_i^j)_{j=1, \dots, k_i}$ eine Partition von V_i , wobei k_i die Anzahl der Informationsmengen für Spieler i ist.
- Für jeden Spieler $i \in N$ und jeden $j \in \{1, 2, \dots, k_i\}$ ist das Paar $(U_i^j, A(U_i^j))$ eine Informationsmenge des Spielers i .
- O ist eine Menge der möglichen Spieldausträge.
- u ist eine Funktion, die jedes Blatt des Spielbaums auf einen Spieldaustrag abbildet.

Innerhalb einer Spielsequenz wählt ein Spieler i , der sich in der Informationsmenge U_i^j befindet, statt einer einzelnen Aktion aus $A(x)$ nun eine Äquivalenzklasse aus $A(U_i^j)$.

Beispiel 2 (Kuhn-Poker) Ein anschauliches Beispiel für ein Spiel mit imperfekter Information ist Kuhn-Poker (KP) [Kuh51]. KP ist ein Spiel für zwei Spieler mit imperfekter Information und Zufallsereignissen. Zu Beginn des Spiels werden zwei Karten aus einem Kartendeck mit drei Karten: König (K), Dame (Q) und Bube (J) an die beiden Spieler zufällig verteilt. Der Spielablauf ist wie folgt:

1. Spieler 1 beginnt und kann entweder schieben (check) oder setzen (bet).
2. Falls Spieler 1 gesetzt hat, kann Spieler 2 entweder passen (fold) oder mitgehen (call). Beim call kommt es zum Showdown.

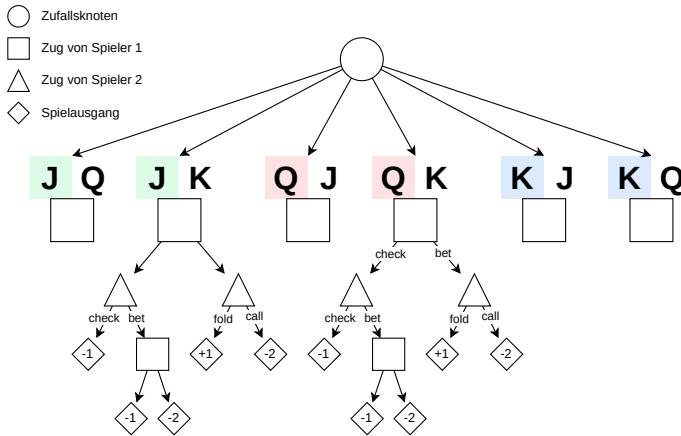


Abbildung 2.2: Ein Ausschnitt des Spielbaums von Kuhn-Poker (KP) [Wik16].

3. Falls Spieler 1 geschoben hat, erhält Spieler 2 dieselben Optionen (schieben oder setzen). Spieler 1 kann daraufhin entsprechend reagieren.
4. Legt ein Spieler ab (fold), verliert er einen Punkt, während der andere einen Punkt gewinnt.
5. Kommt es zum Showdown, gewinnt der Spieler mit der höheren Karte ($K > Q > J$) zwei Punkte, während der andere zwei Punkte verliert.

Die zufällige Kartenverteilung wird durch einen Zufallsknoten dargestellt. Sobald die Karten verteilt sind, kennt Spieler 1 nur seine eigene Karte, nicht aber die des Gegners. Dadurch entstehen nach dem Zufallzug drei mögliche Informationsmengen für Spieler 1:

$$U_1^{J?} = \{x_{1JQ}, x_{1JK}\}, U_1^{Q?} = \{x_{1QJ}, x_{1QK}\}, U_1^{K?} = \{x_{1KJ}, x_{1KQ}\}$$

Spieler 1 weiß, wenn er den Buben (J) besitzt, dass er sich in der Informationsmenge $U_1^{J?}$ befindet, aber er weiß nicht, welche Karte der andere Spieler in der Hand hält. In Frage kommen x_{1JQ} und x_{1JK} . Von beiden Zuständen aus sind die semantisch äquivalenten Aktionen $[a_{1JQ}^{\text{check}}, a_{1JK}^{\text{check}}]$ und $[a_{1JQ}^{\text{bet}}, a_{1JK}^{\text{bet}}$ möglich.

2.1.2 Nutzen

Jedes Spiel definiert eine Menge der möglichen Spielausgänge O . Diese sind abstrakt zu verstehen, das heißt, sie besitzen untereinander keine natürliche Ordnung oder Struktur. Allerdings kann ein Spieler bestimmte Spielausgänge gegenüber anderen bevorzugen. Beispielsweise wird ein Spieler in der Regel einen Gewinn einem Verlust vorziehen.

Die Spieltheorie beschreibt solche Präferenzen im Rahmen der Nutzentheorie (utility theory). Sie definiert eine Präferenzrelation für einen Spieler i über die Menge der möglichen Spielausgänge $o \in O$ sowie über Lotterien von Spielausgängen. Dadurch wird den Spielausgängen eine Ordnung gemäß den Präferenzen des Spielers zugewiesen. Diese Präferenzrelationen ermöglichen es,

Spiele dahingehend zu analysieren, dass Spieler ihre Aktionen basierend auf bevorzugten Spielausgängen wählen.

2.1.2.1 Ohne Zufallsereignisse

Im deterministischen Fall wird die Präferenzrelation als $\succsim_i \subseteq O \times O$ definiert. Zur Vereinfachung schreibt man $x \succsim_i y$, falls $(x, y) \in \succsim_i$ gilt. Diese Relation ist vollständig, reflexiv und transitiv.

Um die Analyse zu erleichtern, werden die Spielausgänge oft mit reellen Zahlen versehen, die den Nutzen eines Spielers ausdrücken. Diese Nutzenfunktion wird als $u(o)$ für einen Spielausgang $o \in O$ bezeichnet und gibt an, wie wünschenswert ein Spielausgang aus Sicht des Spielers ist. Je höher der Nutzenwert, desto begehrter ist der Spielausgang.

Beispiel 3 In TTT kann die Menge der Spielausgänge als

$$O = \{X \text{ gewinnt}, O \text{ gewinnt}, Unentschieden\}$$

definiert werden. Ob ein Spieler einen bestimmten Ausgang einem anderen vorzieht, wird durch die Präferenzrelation festgelegt. Eine übliche Relation für das Spiel TTT ist:

$$\begin{aligned} X \text{ gewinnt } &\succsim_X \text{Unentschieden} \succsim_X O \text{ gewinnt}, \\ O \text{ gewinnt } &\succsim_O \text{Unentschieden} \succsim_O X \text{ gewinnt}. \end{aligned}$$

Die Spielausgänge können numerisch als Nutzenwerte dargestellt werden: Ein Gewinn wird mit 1, ein Unentschieden mit 0 und eine Niederlage mit -1 bewertet. Die Nutzenwerte aller Spieler werden oft in einem Nutzenvektor (utility vector) zusammengefasst. In diesem Beispiel entspricht der erste Eintrag dem Nutzen für X und der zweite dem Nutzen für O:

$$\begin{aligned} [1, -1] &\succsim_X [0, 0] \succsim_X [-1, 1], \\ [-1, 1] &\succsim_O [0, 0] \succsim_O [1, -1]. \end{aligned}$$

2.1.2.2 Mit Zufallsereignissen

In Spielen mit Zufallsereignissen oder in Fällen, in denen wenigstens ein Spieler seine Züge nicht deterministisch wählt, kann der Ausgang des Spiels vom Zufall abhängen. In solchen Situationen muss sich der Spieler entscheiden, welche Risiken er eingehen möchte. Um dies formal zu modellieren, beschreibt die Spieltheorie Präferenzrelationen nicht nur über einzelne Spielausgänge, sondern über Lotterien, also Wahrscheinlichkeitsverteilungen von Spielausgängen. Der Spieler kann durch seine Strategie Einfluss auf die Lotterie nehmen, die letztlich über den Ausgang des Spiels entscheidet.

Definition 5 (Lotterie über Spielausgänge) Eine Lotterie L über eine Menge von Spielausgängen $\{o_1, o_2, \dots, o_n\}$ ist eine Wahrscheinlichkeitsverteilung:

$$L = (p_1, p_2, \dots, p_n),$$

wobei p_i die Wahrscheinlichkeit ist, dass der Spielausgang o_i eintritt und die Summe aller Wahrscheinlichkeiten p_i ergibt 1:

$$\sum_{i=1}^n p_i = 1, \quad p_i \in [0, 1] \quad \forall i = 1, 2, \dots, n.$$

Anstatt einer Präferenzrelation über einzelne Spielausgänge wird in diesem Fall eine Präferenzrelation über Lotterien definiert. Diese gibt an, welche Wahrscheinlichkeitsverteilungen ein Spieler gegenüber anderen bevorzugt. Ein Spieler wählt also eine Strategie nicht nur basierend auf sicheren Spielausgängen, sondern darauf, welche Wahrscheinlichkeitsverteilung für ihn vorteilhafter erscheint.

Um eine numerische Bewertung von Lotterien zu ermöglichen, ordnet man ihnen Nutzenwerte zu. Im Folgenden wird angenommen, dass Spieler lineare Nutzenfunktionen besitzen. Diese zeichnet sich dadurch aus, dass der Nutzenwert einer Lotterie dem Erwartungswert der Nutzenwerte der zugrunde liegenden Spielausgänge entspricht.

Definition 6 (Nutzenfunktion über Lotterien) Eine Nutzenfunktion u_i für einen Spieler i und für die Spielausgänge $A_1, A_2, \dots, A_K \in \Omega$ wird als linear bezeichnet, wenn für jede Lotterie

$$L = [p_1(A_1), p_2(A_2), \dots, p_K(A_K)],$$

gilt:

$$u_i(L) = p_1 u_i(A_1) + p_2 u_i(A_2) + \dots + p_K u_i(A_K).$$

Es sei darauf hingewiesen, dass Spieler unterschiedliche Risikoprofile haben können. Diese können durch eine Nutzenfunktion modelliert werden, welche Wahrscheinlichkeitsverteilungen nicht nur durch den Erwartungswert, sondern auch durch weitere Kriterien wie die Varianz der möglichen Ergebnisse bewertet. Im Folgenden soll jedoch immer von linearen Nutzenfunktionen ausgegangen werden.

Beispiel 4 Angenommen, ein Spieler kann den Spielverlauf so beeinflussen, dass er zwischen den folgenden Lotterien wählen kann:

$$L_1 = \left[\frac{1}{2}(A), \frac{1}{2}(B) \right], \quad L_2 = \left[\frac{1}{1}(C) \right].$$

Die Nutzenwerte der Spielausgänge seien gegeben durch:

$$u(A) = 20, \quad u(B) = 10, \quad u(C) = 13.$$

Dann berechnen sich die Nutzen der beiden Lotterien als:

$$u(L_1) = \frac{1}{2} \cdot u(A) + \frac{1}{2} \cdot u(B) = \frac{1}{2} \cdot 20 + \frac{1}{2} \cdot 10 = 10 + 5 = 15,$$

$$u(L_2) = \frac{1}{1} \cdot u(C) = 1 \cdot 13 = 13.$$

Da $u(L_1) > u(L_2)$ wird ein risikoneutraler Spieler (mit linearer Nutzenfunktion) die Lotterie L_1 bevorzugen. Ein risikoaverser Spieler könnte hingegen L_2 wählen, da er die sichere Auszahlung von 13 dem unsicheren Erwartungswert von 15 vorzieht.

2.1.3 Spiele in Strategieform

Die Extensivform eines Spiels dient der Definition und Untersuchung seiner Dynamik. Dazu gehören beispielsweise Spielregeln, Zufallsereignisse und die Reihenfolge der Aktionen. Um mögliche Strategien und deren Auswirkungen auf das Spielergebnis zu analysieren, verwendet die Spieltheorie das Modell des Spiels in Strategieform (strategic-form game). In dieser Darstellungsweise lässt sich untersuchen, wie die Strategien der einzelnen Spieler den Ausgang des Spiels beeinflussen und welche Strategien bevorzugt werden sollten. Dies hilft insbesondere bei der Beantwortung der Frage, was es bedeutet, ein Spiel *gut* zu spielen.

2.1.3.1 Reine Strategien

Um ihr präferiertes Ergebnis zu erreichen, müssen die Spieler die richtigen Aktionen ausführen. Eine Strategie beschreibt, welche Aktionen ein Spieler in einer gegebenen Spielsituation wählt. Eine reine Strategie ist eine feste Handlungsanweisung, die jeder möglichen Situation des Spielers genau eine definierte Aktion zuweist.

Definition 7 (Reine Strategie in einem Spiel mit perfekter Information)
Eine reine Strategie eines Spielers i ist eine Funktion s_i , die jeden Knoten $x \in V_i$ auf eine Aktion in $A(x)$ abbildet.

$$s_i : V_i \rightarrow A(x).$$

Definition 8 (Reine Strategie in einem Spiel mit imperfekter Information)
Eine reine Strategie eines Spielers i ist eine Funktion s_i , welche jede Informationsmenge eines Spielers auf eine Aktion, die in dieser Informationsmenge möglich ist, abbildet:

$$s_i : U_i \rightarrow \bigcup_{j=1}^{k_i} A(U_i^j),$$

		Spieler 2		
		Schere	Stein	Papier
Spieler 1	Schere	0, 0	-1, 1	1, -1
	Stein	1, -1	0, 0	-1, 1
	Papier	-1, 1	1, -1	0, 0

Tabelle 2.1: Das Spiel Schere-Stein-Papier (SSP) in Strategieform.

wobei $\mathcal{U}_i = \{U_i^1, \dots, U_i^{k^i}\}$ die Menge der Informationsmengen von i ist und für jede Informationsmenge $U_i^j \in \mathcal{U}_i$ gilt:

$$s_i(U_i^j) \in A(U_i^j).$$

Anschaulich bedeutet dies, dass eine Strategie für jeden möglichen Spielzustand oder jede Informationsmenge vorschreibt, welche Aktion der Spieler ausführen soll.

2.1.3.2 Strategieform

Definition 9 Ein Spiel in Strategieform ist ein geordnetes Tripel $G = (N, (S_i)_{i \in N}, (u_i)_{i \in N})$, für das gilt:

- $N = \{1, 2, \dots, n\}$ ist eine endliche Menge der Spieler.
- S_i ist die Menge aller Strategien des Spielers i für jeden Spieler $i \in N$.

Die Menge aller Strategiekombinationen (Strategievektoren) wird durch das kartesische Produkt

$$S_1 \times S_2 \times \dots \times S_n$$

beschrieben. Die Funktion $u_i : S \rightarrow \mathbb{R}$ ist die Nutzenfunktion von Spieler i . Sie ordnet jeder möglichen Strategiekombination $s = (s_1, s_2, \dots, s_n)$ einen reellen Nutzenwert $u_i(s)$ zu. Der Nutzenwert gibt an, wie vorteilhaft die entsprechende Strategiekombination für Spieler i ist.

Die Strategieform ermöglicht die Untersuchung der möglichen Spielergebnisse unter der Annahme, dass jeder Spieler einer bestimmten Strategie folgt. Befolgen alle Spieler die im Strategievektor s angegebenen Strategien, ergibt sich der durch $u(s)$ bestimmte Auszahlungsvektor.

In einem Spiel ohne Zufallsereignisse führt jede Strategiekombination zu einem eindeutigen Auszahlungsvektor. In einem Spiel mit Zufallsereignissen oder imperfekter Information ergibt sich dagegen eine Wahrscheinlichkeitsverteilung über die möglichen Auszahlungsvektoren.

2.1.3.3 Nullsummenspiele

Spiele, bei denen die Summe des Auszahlungsvektors stets 0 beträgt, werden als Nullsummenspiele (zero-sum game) bezeichnet. In einem solchen Spiel

entspricht der Gewinn eines Spielers genau dem Verlust der anderen Spieler. Während sich die meisten spieltheoretischen Untersuchungen auf Nullsummenspiele mit zwei Spielern konzentrieren, da diese einige interessante Eigenschaften besitzen, kann das Konzept grundsätzlich auf Spiele mit mehr als zwei Spielern erweitert werden.

Definition 10 Ein Spiel mit zwei Spielern ist ein Nullsummenspiel, wenn für jedes Paar von Strategien (s_I, s_{II}) gilt:

$$u_I(s_I, s_{II}) + u_{II}(s_I, s_{II}) = 0.$$

2.1.3.4 Dominanz

Mit den Begriffen von Strategie und Nutzen kann nun präziser beschrieben werden, was es bedeutet, ein Spiel gut zu spielen. Ziel eines Spielers ist es, eine Strategie zu wählen, die seinen Nutzen maximiert. Dabei muss er jedoch berücksichtigen, dass auch die anderen Spieler ihre eigenen Strategien wählen, wodurch Wechselwirkungen entstehen.

Ein zentraler spieltheoretischer Ansatz zur Bestimmung guter Strategien basiert auf dem Konzept der Dominanz. Hierbei wird angenommen, dass Spieler rational handeln und ihre Strategiewahl entsprechend optimieren. Die folgenden Annahmen gelten:

- Jeder Spieler spielt rational.
- Jeder Spieler weiß, dass alle anderen Spieler rational spielen.
- Ein rationaler Spieler wählt keine dominierte Strategie.

Eine Strategie gilt als dominiert, wenn es eine andere Strategie gibt, die in jeder Spielsituation einen gleich hohen oder höheren Nutzen erzielt.

Definition 11 (Strikte Dominanz) Eine Strategie s_i eines Spielers i ist strikt dominiert, wenn es eine andere Strategie t_i desselben Spielers gibt, sodass für alle Strategiekombinationen der anderen Spieler $s_{-i} \in S_{-i}$ gilt:

$$u_i(s_i, s_{-i}) < u_i(t_i, s_{-i}).$$

In diesem Fall sagt man, dass s_i strikt von t_i dominiert wird beziehungsweise dass t_i eine strikt dominierende Strategie ist.

Definition 12 (Schwache Dominanz) Eine Strategie s_i eines Spielers i wird als schwach dominiert bezeichnet, wenn es eine andere Strategie t_i desselben Spielers i gibt, die folgende Bedingungen erfüllt: (a) Für alle Strategien der anderen Spieler $s_{-i} \in S_{-i}$ gilt:

$$u_i(s_i, s_{-i}) \leq u_i(t_i, s_{-i}).$$

		Spieler 2	
		D	E
		A	(2, 2) (1, 1)
Spieler 1		B	(3, 1) (2, 2)
		C	(2, 1) (2, 1)

Tabelle 2.2: Ein Spiel in Strategieform, welches das Konzept der strikten und schwachen Dominanz zeigt.

(b) Es gibt mindestens eine Strategiekombination $t_{-i} \in S_{-i}$ für die gilt:

$$u_i(s_i, t_{-i}) < u_i(t_i, t_{-i}).$$

In diesem Fall sagt man, dass s_i von t_i schwach dominiert wird und dass t_i s_i dominiert.

Die Dominanzanalyse vergleicht immer Strategien desselben Spielers miteinander, nicht Strategien verschiedener Spieler.

- Eine strikt dominierte Strategie ist in jeder Situation schlechter als eine andere Strategie.
- Eine schwach dominierte Strategie ist mindestens einmal schlechter, aber in anderen Fällen gleich gut.

Beispiel 5 (Strikte und schwache Dominanz) Tabelle 2.2 zeigt ein Spiel in Strategieform, wobei Spieler 1 zwischen den Strategien A, B und C wählen kann, während Spieler 2 zwischen den Strategien D und E wählt. Eine Betrachtung der Strategien von Spieler 1 ergibt:

- Strategie A wird strikt von Strategie B dominiert, da B unabhängig von der Wahl von Spieler 2 immer einen höheren Nutzen bringt.
- Strategie C wird schwach von B dominiert, da B gegen E genauso gut wie C ist, aber gegen D besser abschneidet.

Für Spieler 2 gibt es keine Dominanzen, da keine Strategie durchgehend besser ist:

- D ist gegen A besser.
- E ist gegen B besser.

2.1.3.5 Nash-Gleichgewicht

Ein zentrales Konzept der Spieltheorie ist das der strategischen Stabilität, bekannt als Nash-Gleichgewicht (nash equilibrium). Es beschreibt eine Situation, in der kein Spieler einen Anreiz hat, seine Strategie einseitig zu ändern, solange die anderen Spieler ihre Strategien beibehalten [Nas50].

		Spieler 2	
		C	D
		C	$3, 3$
Spieler 1	C	$3, 3$	$0, 5$
	D	$5, 0$	$1, 1$

Tabelle 2.3: Das Gefangenendilemma mit einem Nash-Gleichgewicht.

Definition 13 (Nash-Gleichgewicht) Ein Strategievektor $s^* = (s_1^*, \dots, s_n^*)$ ist ein Nash-Gleichgewicht (nash equilibrium), wenn für jeden Spieler $i \in N$ und jede alternative Strategie $s_i \in S_i$ gilt:

$$u_i(s^*) \geq u_i(s_i, s_{-i}^*).$$

Das bedeutet, dass kein Spieler durch eine einseitige Abweichung seinen Nutzen erhöhen kann. Ein Spiel kann mehrere Nash-Gleichgewichte haben. Falls nur reine Strategien erlaubt sind, ist es möglich, dass ein Spiel kein Nash-Gleichgewicht besitzt.

Beispiel 6 (Gefangenendilemma) Ein klassisches Beispiel für ein Nash-Gleichgewicht ist das Gefangenendilemma (Tabelle 2.3). Jeder Spieler kann zwischen Kooperation (C) und Verrat (D) wählen. Wenn beide kooperieren (C, C), erhalten beide eine moderate Belohnung (3,3). Entscheidet sich ein Spieler für Verrat (D), während der andere kooperiert, erhält der Verräter die maximale Belohnung (5), während der kooperative Spieler die schlechteste Auszahlung (0) bekommt. Wenn beide verraten (D, D), erhalten beide eine niedrige, aber nicht die schlechteste Auszahlung (1,1). Das einzige Nash-Gleichgewicht ist der Strategievektor (D, D), da keiner der Spieler seinen Nutzen durch einseitige Abweichung verbessern kann.

2.1.3.6 Gemischte Strategien

Bisher wurden nur reine Strategien behandelt, bei denen ein Spieler in jeder Situation eine bestimmte Aktion ausführt. In Spielen mit imperfekter Information und Zufallsereignissen kann es jedoch vorteilhaft oder sogar notwendig sein, zufällig und damit unvorhersehbar zu handeln. Diese Unvorhersehbarkeit wird durch gemischte Strategien modelliert.

Definition 14 (Gemischte Strategie) Sei $G = (N, (S_i)_{i \in N}, (u_i)_{i \in N})$ ein Spiel in Strategieform mit endlicher Strategiemenge S_i für jeden Spieler i . Eine gemischte Strategie ist eine Wahrscheinlichkeitsverteilung über seine möglichen reinen Strategien. Die Menge der gemischten Strategien des Spielers i ist dann:

$$\Sigma_i := \left\{ \sigma_i : S_i \rightarrow [0, 1] \mid \sum_{s_i \in S_i} \sigma_i(s_i) = 1 \right\}.$$

Eine gemischte Strategie beschreibt also, mit welcher Wahrscheinlichkeit ein Spieler eine bestimmte reine Strategie wählt. Um Spiele mit gemischten

Strategien zu analysieren, kann das Modell der Strategieform um gemischte Strategien erweitert werden:

$$\Gamma = (N, (\Sigma_i)_{i \in N}, (U_i)_{i \in N})$$

Auf eine genaue Definition soll hier verzichtet werden und stattdessen sei auf [MSZ20] verwiesen. Entscheidend ist, dass der Strategievektor σ nun aus gemischten Strategien besteht und die erwartete Auszahlung $U(\sigma)$ den erwarteten Nutzen widerspiegelt, wenn jeder Spieler gemäß seiner gemischten Strategie agiert.

2.1.3.7 Dominanz in gemischten Strategien

Das Konzept der dominanten Strategien lässt sich auch auf gemischte Strategien übertragen.¹ Während bei reinen Strategien zwei Strategien anhand der direkten Auszahlung $u_i(s)$ verglichen werden, betrachtet man nun den erwarteten Nutzen $U_i(\sigma)$.

- Eine strikt dominierende gemischte Strategie führt unabhängig von den Strategien der Gegner immer zu einem höheren erwarteten Nutzen.
- Eine schwach dominierende gemischte Strategie führt gegen mindestens eine Strategie des Gegners zu einem höheren erwarteten Nutzen, während sie in allen anderen Fällen mindestens gleich gut ist.

Dieses Konzept erlaubt es, suboptimale Strategien auch in Spielen mit gemischten Strategien auszuschließen.

2.1.3.8 Nash-Gleichgewicht in gemischten Strategien

Das Nash-Gleichgewicht kann auch für gemischte Strategien definiert werden und ist dort von besonderer Bedeutung. Im gemischten Fall stellt das Nash-Gleichgewicht auf den erwarteten Nutzen ab. Während es bei einem Spiel mit nur reinen Strategien nicht unbedingt ein Nash-Gleichgewicht gibt, gibt es in Spielen mit gemischten Strategien immer eins.

Theorem 1 (Existenz eines Nash-Gleichgewichts in gemischten Strategien) *Jedes Spiel in Strategieform mit einer endlichen Anzahl von Spielern und einer endlichen Menge reiner Strategien besitzt mindestens ein Nash-Gleichgewicht in gemischten Strategien [Nas50] [Nas51].*

Beispiel 7 (Schere-Stein-Papier) Das Spiel SSP (Tabelle 2.1) ist ein klassisches Beispiel für ein Spiel mit imperfekter Information, da kein Spieler im Voraus weiß, welche Wahl der Gegner treffen wird. Jeder Spieler hat drei mögliche reine Strategien:

$$S_1 = S_2 = \{\text{Schere, Stein, Papier}\}.$$

¹ Auf eine erneute Definition soll hier verzichtet werden.

Falls ein Spieler immer die gleiche Strategie wählt, kann der Gegner dies ausnutzen, indem er stets die überlegene Strategie spielt. In diesem Fall besteht kein Nash-Gleichgewicht in reinen Strategien, da es immer eine bessere Antwort gibt. Erst durch Unberechenbarkeit kann sich ein Spieler absichern. Die optimale Strategie ist, zwischen den drei Möglichkeiten gleichverteilt zu wählen:

$$\sigma^* = \left(\frac{1}{3}, \frac{1}{3}, \frac{1}{3}\right).$$

Das bedeutet, dass jeder Spieler mit einer Wahrscheinlichkeit von $\frac{1}{3}$ jeweils Schere, Stein oder Papier wählt. Dieser Strategievektor ist das einzige Nash-Gleichgewicht im Spiel.

2.1.3.9 Verhaltensstrategien

Die Extensivform und die Strategieform von Spielen unterscheiden sich in ihrem Fokus. Während die Extensivform die dynamischen Aspekte eines Spiels beschreibt, konzentriert sich die Strategieform auf die Strategien der Spieler und den daraus resultierenden Nutzen. Dennoch erfassen beide Darstellungen dasselbe zugrunde liegende Spiel. In der Strategieform werden die Spielausgänge untersucht, die sich gemäß den in der Extensivform definierten Regeln ergeben.

Gemischte Strategien sind in der Strategieform so definiert, dass ein Spieler mittels einer Wahrscheinlichkeitsverteilung eine reine Strategie wählt. Würde eine solche gemischte Strategie in einem Spiel in Extensivform angewendet werden, könnte der Spieler zu Beginn des Spiels zufällig (entsprechend der Wahrscheinlichkeitsverteilung) eine reine Strategie wählen und diese konsequent ausführen. Eine alternative Möglichkeit, die für die Extensivform natürlicher wirkt, besteht darin, für jede Informationsmenge (bzw. für jeden Zustand, in dem sich der Spieler befindet) eine neue Zufallsentscheidung dagehend zu treffen, welche Aktion der Spieler ausführt. Solche Strategien werden Verhaltensstrategien bezeichnet.

Definition 15 (Verhaltensstrategie) *Eine Verhaltensstrategie eines Spielers in einem Spiel in Extensivform ist eine Funktion, die jeder Informationsmenge eine Wahrscheinlichkeitsverteilung über die an dieser Informationsmenge verfügbaren Aktionen zuordnet.*

In den meisten Fällen und auch in dieser Arbeit bei der Beschäftigung im Rahmen des RL und der verwendeten Algorithmen ist es einfacher, mit Verhaltensstrategien zu arbeiten. Um sicherzustellen, dass Erkenntnisse über Dominanz und das Nash-Gleichgewicht auch für Verhaltensstrategien gelten, muss geprüft werden, ob eine Äquivalenz zwischen gemischten Strategien und Verhaltensstrategien besteht. Der Satz von Kuhn [Kuh53] stellt sicher, dass diese Äquivalenz unter bestimmten Bedingungen gegeben ist.

Definition 16 (Satz von Kuhn) Für jedes Spiel in Extensivform gilt: Wenn der Spieler i perfektes Erinnerungsvermögen (recall) hat, dann existiert für jede seiner gemischten Strategien eine äquivalente Verhaltensstrategie.

Ein Spieler hat perfektes Erinnerungsvermögen, wenn er sich in jeder Informationsmenge an alle zuvor erlangten Informationen erinnert. Dazu gehören:

- seine eigenen vorherigen Aktionen,
- die Ergebnisse früherer Zufallsereignisse
- sowie die zuvor beobachteten Aktionen der anderen Spieler.

Die Umkehrung des Satzes von Kuhn gilt mit Einschränkungen auch.

Definition 17 Sei

$$\Gamma = \left(N, V, E, v_0, (V_i)_{i \in N \cup \{0\}}, (p_x)_{x \in V_0}, (\mathcal{U}_i)_{i \in N}, O, u \right)$$

ein Spiel in Extensivform, in dem jeder Zustand mindestens zwei mögliche Aktionen besitzt. Jede Verhaltensstrategie eines Spielers i besitzt eine äquivalente gemischte Strategie, wenn jede Informationsmenge von i in jedem Pfad, der von der Wurzel v_0 ausgeht, höchstens einmal vorkommt.

Die Bedingung, dass eine Informationsmenge in jedem Pfad nur einmal vorkommen darf, ist notwendig. Falls eine Informationsmenge mehrmals auf demselben Pfad vorkäme, könnte der Spieler nicht unterscheiden, in welcher Instanz dieser Informationsmenge er sich befindet. Aus diesem Grund ist perfektes Erinnerungsvermögen eine zentrale Voraussetzung für die Äquivalenz zwischen Verhaltens- und gemischten Strategien.

Da diese Äquivalenz unter den genannten Bedingungen gegeben ist, können die Konzepte der Dominanz und des Nash-Gleichgewichts auch auf Verhaltensstrategien angewendet werden. Für detaillierte Beweise und eine formale Definition des perfekten Erinnerungsvermögens sei auf [MSZ20] verwiesen.

2.2 REINFORCEMENT LEARNING

Reinforcement Learning (RL) ist ein Teilbereich des maschinellen Lernens, welcher sich mit der Definition des RL-Problems und Methoden zur Lösung beschäftigt. RL-Probleme sind Probleme, in denen ein Agent mit einer Umgebung interagiert und bei dieser Interaktion Belohnungen erhält. Der Agent soll (nur) durch die Interaktion mit der Umgebung eine Strategie entwickeln, mit welcher er die erhaltenen Belohnungen bei nachfolgenden Interaktion maximiert.

Nach [SB20b] sind die drei wichtigsten Merkmale, welche RL-Probleme definieren:

- **Geschlossenheit:** Ein Agent interagiert mit einer Umgebung, was den Zustand der Umgebung verändert. Diese Veränderung führt zu veränderten Rückmeldungen durch die Umgebung als Eingabe für den Agenten. Insofern entsteht ein geschlossener Kreislauf, bei dem der Agent durch seine Aktionen die eigene Eingabe mitbestimmt.
- **Keine direkten Anweisungen:** Der Agent erhält keine direkten Anweisungen, sondern muss selber herausfinden, welche Aktionen zu den besten Ergebnissen führen.
- **Langfristige Konsequenzen von Aktionen:** Eine Aktion des Agenten wirkt sich nicht nur auf die unmittelbare Belohnung, sondern auch auf den nächsten Zustand der Umgebung und damit auf alle nachfolgenden Belohnungen aus. Das bedeutet, der Agent muss Strategien entwickeln, die nicht nur kurzfristige Belohnungen, sondern langfristige Belohnungen maximieren.

Die in diesem Abschnitt verwendeten Definitionen orientieren sich, soweit nicht anders aufgeführt, an [WO12].

2.2.1 Grundbegriffe

Im RL interagiert der sogenannte Agent mit einer Umgebung (Environment). Diese Interaktion erfolgt in diskreten Zeitschritten (z.B. $t = 0, 1, 2, 3, \dots$). In jedem Zeitschritt erhält der Agent eine Beschreibung des aktuellen Zustands s_t der Umgebung sowie eine Belohnung r_t (Reward) als Reaktion auf seine letzte Aktion a_{t-1} . Die Belohnung ist ein numerischer Wert. Der Agent wählt auf Basis des ihm bekannten Zustands s_t eine Aktion a_t , mit der er mit der Umgebung interagiert. Diese Aktion wirkt sich auf die Umgebung aus und diese liefert als Ergebnis eine neue Belohnung r_{t+1} für den nächsten Zeitschritt sowie einen neuen Zustand s_{t+1} , in dem sich die Umgebung befindet (vgl. Abbildung 2.3a).

Das Ziel eines Agenten ist es, eine Strategie (Policy) durch Interaktion mit der Umgebung abzuleiten, welche die erhaltene Belohnung über eine lange Zeit maximiert. Eine im Rahmen der Interaktionsschleife erlebte Erfahrung kann als Tupel $(s_t, a_t, s_{t+1}, r_{t+1})$ beschrieben werden.

Das Verhalten einer Umgebung wird im Falle *eines* Agenten und einer vollständig beobachtbaren Umgebung durch einen Markov Decision Process (MDP) vollständig definiert. Das Verhalten gibt an, welche Belohnung in welchem Zustand und bei welcher Aktion zurückgegeben wird. Außerdem gibt der MDP die Zustandsübergänge und deren Voraussetzungen an.

Neben vollständig beobachtbaren Umgebungen gibt es Umgebungen, die nur teilweise beobachtbar sind. Diese werden durch sogenannte Partially observable Markov Decision Procss (POMDP) beschrieben. Ein eigener Teilbereich des RL, das sogenannte Multi-Agent Reinforcement Learning (MARL),

² Die Abbildung ist angelehnt an die Übersichtsgrafiken von [MI20] und [SB20b]. Die verwendeten Icons entstammen der Symbolbibliothek „Neu“[RP].

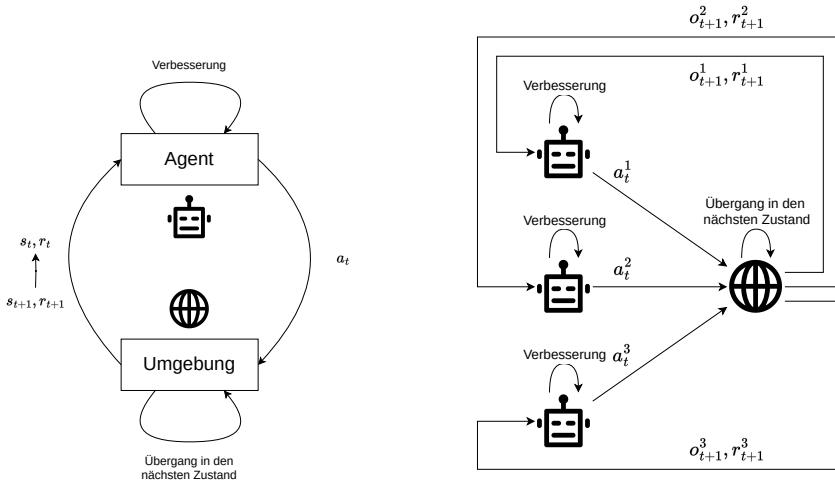


Abbildung 2.3: Interaktionsschleifen im Multiagenten- und Einzelagenten-RL.²

beschäftigt sich mit Umgebungen mit mehreren, meist konkurrierenden Agenten. In einem ersten Schritt werden die MDP definiert und erläutert. Darauf aufbauend werden die anderen Umgebungen beschrieben.

2.2.2 Vollständig beobachtbare Umgebungen

Definition 18 (MDP) Ein episodisches MDP ist definiert als ein Tupel

$$MDP(S, A, T, R, I, G),$$

für das gilt:

- S ist eine Menge von Zuständen.
- A ist eine Menge von Aktionen.
- $T : S \times A \times S \rightarrow [0, 1]$ ist die Übergangsfunktion (transition function), welche ein Tupel (s, a, s') auf eine Wahrscheinlichkeit abbildet.
- $R : S \times A \times S \rightarrow \mathbb{R}$ ist die Belohnungsfunktion (reward function), welche ein Tupel (s, a, s') auf einen Skalar r abbildet.
- $I : S \rightarrow [0, 1]$ ist eine Wahrscheinlichkeitsverteilung über alle Zustände $s \in S$ (initial state distribution).
- $G \subseteq S$ ist eine Menge von Zuständen, bei denen die Episode endet (goal state area).

Ein Zustand $s \in S$ gibt einen Zustand an, in dem sich eine Umgebung befinden kann. Die Transitionsfunktion gibt für einen Ausgangszustand $s \in S$ an, in welchen Zielzustand $s' \in S$ die Umgebung gelangt,

wenn die Aktion a im Ausgangszustand s ausgeführt wird. Die Angabe einer Wahrscheinlichkeit ermöglicht es, Probleme zu modellieren, in denen der Ausgang stochastischen Einflüssen unterliegt, also mehrere Zielzustände für ein Zustands-Aktion-Tupel möglich sind. Die Belohnungsfunktion bildet ein Zustand-Aktion-Zustand-Tupel (s, a, s') auf einen skalaren Wert ab, den die Umgebung im Falle, dass die Umgebung zu s' fortschreitet, an den Agenten emittiert. Nicht jede der insgesamt verfügbaren Aktionen $a \in A$ muss in jedem Ausgangszustand zur Verfügung stehen. Für diese Eingaben ist die Transitionsfunktion nicht definiert.

Definition 18 bezieht sich auf episodische MDP. Episodische MDP zeichnen sich dadurch aus, dass die Interaktion ausgehend von einem Startzustand in endlich vielen Schritten in einem Endzustand terminiert. Eine Interaktionschleife von einem Startzustand bis zu einem Endzustand nennt man eine Episode. Da Spiele stets einen Anfang und ein Ende haben, wird in dieser Arbeit nur auf episodische Entscheidungsprozesse eingegangen. Bei einem episodischen MDP gibt I für jeden Zustand an, mit welcher Wahrscheinlichkeit dieser Zustand der Startzustand ist. Zustände, die kein Startzustand sind, haben damit eine Wahrscheinlichkeit von 0. Die Teilmenge G gibt die Zustände an, in denen die Episode endet.

Ein MDP bekommt seinen Namen durch die Erfüllung der Markov-Eigenschaft. Diese besagt, dass der zukünftige Zustand der Umgebung allein vom aktuellen Zustand und der ausgeführten Aktion abhängt und nicht von vorhergehenden Zuständen, in denen sich die Umgebung vorher einmal befunden hat.

Das Ziel eines RL-Algorithmus ist es, eine optimale Strategie (Policy) zu finden. Dabei unterscheidet man zwischen deterministischen und stochastischen Strategien. Eine deterministische Strategie für ein MDP (S, A, T, R, I, G) ist eine Funktion $\pi : S \rightarrow A$, welche einen Zustand $s \in S$ auf eine Aktion $a \in A$ abbildet. Eine stochastische Strategie ist eine Funktion, welche einen Zustand $s \in S$ auf eine Wahrscheinlichkeit abbildet: $\pi : S \times A \rightarrow [0, 1]$. Es gilt dabei für jeden $s \in S$ $\sum_{a \in A} \pi(s, a) = 1$. Die stochastische Strategie wählt daher aus den möglichen Aktionen in einem Zustand eine Aktion zufällig nach der angegebenen Wahrscheinlichkeitsverteilung aus.

Prinzipiell gibt es mehrere Möglichkeiten, die Optimalität einer Strategie zu definieren. Für endliche episodische Probleme, wie zum Beispiel die hier betrachteten Brett- und Kartenspiele, wird in der Regel die Maximierung des Erwartungswerts $E \left[\sum_{t=0}^h r_t \right]$ der erhaltenen Belohnungen über alle Zeitschritte h einer Episode angestrebt. Dabei ist h die Anzahl der diskreten Zeitschritte t , die in der Ausführung einer Sequenz mit der Strategie durchlaufen wurden. r_t ist dabei die vom MDP an den Agenten ausgegebene Belohnung im Zeitschritt t . Für andere Probleme, wie zum Beispiel solche mit sehr großer oder unbegrenzter Länge, wird meistens ein zusätzlicher Diskontfaktor λ^t berücksichtigt, der kurzfristig erreichbare Belohnungen gegenüber späteren Belohnungen bevorzugt. In Brettspielen macht ein solcher Faktor meist keinen Sinn, da die Episodenlänge meist klein ist und eine Belohnung meist ohnehin

erst am Ende des Spiels (in Form des Spielergebnisses) vergeben wird. Daher sollen andere Formen der Optimalität hier nicht betrachtet werden.

2.2.3 Partiell beobachtbare Umgebungen

In partiell beobachtbaren Umgebungen erhält der Agent nicht den vollständigen Zustand der Umgebung, sondern nur einen Teil davon. Dieser Teil des Zustands wird als Beobachtung bezeichnet. Hinter einer Beobachtung können unterschiedliche Zustände stehen. Der Agent weiß also nicht, in welchem Zustand sich die Umgebung genau befindet. Mit diesen partiell beobachtbaren Umgebungen können Szenarien mit imperfekter Information modelliert werden.

Definition 19 (Partially observable Markov Decision Proecss (POMDP))

Ein episodisches POMDP ist ein Tupel

$$(S, A, \Omega, T, O, R, I, G)$$

wobei gilt:

- S ist eine Menge von Zuständen.
- A ist eine Menge von Aktionen.
- Ω ist eine Menge von Beobachtungen.
- $T : S \times A \times S \rightarrow [0, 1]$ ist die Übergangsfunktion (transition function), welche ein Tupel (s, a, s') auf eine Wahrscheinlichkeit abbildet.
- $O : S \times A \times \Omega \rightarrow [0, 1]$ ist eine Beobachtungsfunktion, welche ein Tupel (s', a, o) auf eine Wahrscheinlichkeit abbildet.
- $R : S \times A \times S \rightarrow \mathbb{R}$ ist die Belohnungsfunktion (reward function), welche ein Tupel (s, a, s') auf einen Skalar r abbildet.
- $I : S \rightarrow [0, 1]$ ist eine Wahrscheinlichkeitsverteilung über alle Zustände $s \in S$ (initial state distribution).
- $G \subseteq S$ ist eine Menge von Zuständen, bei denen die Episode endet (goal state area).

Die Definition des POMDP generalisiert das MDP. Ergänzt wird es um die Menge der möglichen Beobachtungen Ω und eine Beobachtungsfunktion O , welche für ein Tupel (s', a, o) die Wahrscheinlichkeit angibt, dass der Agent nach Ausführung von Aktion a bei Erreichen des Folgezustands s' die Beobachtung o erhält. Falls die Wahrscheinlichkeit für eine Beobachtung o nicht von der letzten Aktion a abhängt, kann man die Beobachtungsfunktion auch als $O : S \times \Omega \rightarrow [0, 1]$ definieren.

2.2.4 Multi-Agent Reinforcement Learning

Die bisher aufgezeigten RL-Probleme haben sich stets mit der Interaktion zwischen einem Agenten und einer sich nicht verändernden Umgebung beschäftigt. Im MARL geht es um Probleme, bei denen mehrere Agenten in der gleichen Umgebung in jedem Zeitschritt gleichzeitig agieren und jeder Agent eine eigene Belohnung und eine eigene Beobachtung (im Falle von partiell beobachtbaren Umgebungen) erhält. Die Modellierung der Belohnungssignale entscheidet darüber, ob die Agenten kooperativ oder kompetitiv agieren. Jeder Agent versucht auch hier durch Interaktion mit der Umgebung eine Strategie zu finden, die den Erwartungswert der eigenen erhaltenen Belohnung über eine Episode maximiert. Ein Überblick über die Interaktionsschleife zwischen Agenten und einer partiell beobachtbaren Umgebung wird in Abbildung 2.3b gezeigt.

Die Definitionen des episodischen MDP und POMDP lassen sich nun zu einem episodischen Multi-Agent Markov Decision Process (MA-MDP) und Multi-Agent Partially observable Markov Decision Process (MA-POMDP) ergänzen. Statt eine vollständige (repetitive) Definition zu geben, sollen an dieser Stelle nur die notwendigen Änderungen an den bestehenden Definitionen angegeben werden, um eine Intuition für das MARL-Problem zu geben. In der Literatur wird eine vollständige Definition nur selten angegeben.³

- Es wird eine Menge von α Agenten eingeführt.
- Für jeden Agenten wird eine eigene Aktionsmenge definiert: $\{A_i\}_{i \in \alpha}$.
- Die Transitionsfunktion $T : S \times (A_1 \times A_2 \times \dots \times A_\alpha) \times S \rightarrow [0, 1]$ modelliert die Übergangswahrscheinlichkeiten basierend auf den (gleichzeitigen) Aktionen aller Agenten.
- Für jeden Agenten i wird eine eigene Belohnungsfunktion $R_i : S \times A_i \times S \rightarrow \mathbb{R}$ definiert, die den individuellen Nutzen des Agenten beschreibt.
- Im Fall von MA-POMDP erhält jeder Agent i eine eigene Beobachtungsmenge Ω_i .
- Im Fall von MA-POMDP erhält jeder Agent eine Beobachtungsfunktion $O_i : S \times (A_1 \times A_2 \times \dots \times A_\alpha) \times \Omega_i \rightarrow [0, 1]$, die beschreibt, mit welcher Wahrscheinlichkeit Agent i nach Ausführung der gleichzeitigen Aktionen $(a_1, a_2, \dots, a_\alpha)$ bei Erreichen des Folgezustands s' die Beobachtung o_i erhält.

Im Falle des RL mit nur einem Agenten ist die Umgebung, mit welcher er interagiert, stationär. Das zugrundeliegende Modell (Zustände, Transitionsfunktion, Belohnungsfunktion) verändert sich im Rahmen der Interaktion nicht. Hier liegt der entscheidende Unterschied zum MARL: Im MARL ver-

³ Eine Definition für ein MA-MDP befindet sich zum Beispiel in [Bou96] und für ein MA-POMDP in [Gal+24].

ändert sich aus Sicht eines Agenten das (ihm bekannte) Verhalten der Umgebung, wenn die anderen Agenten ihre Strategie anpassen. Anders ausgedrückt: Betrachtet man das Ganze als RL-Problem mit nur einem Agenten, wäre die zugrundeliegende Umgebung nicht stationär und das zugrundeliegende Modell unterläge Änderungen als Folge von Interaktion.

2.2.5 *Modellbasierte und modellfreie Methoden*

Im RL unterscheidet man zwischen modellbasierten Algorithmen (model-based) und modellfreien Algorithmen (model-free). Ein Algorithmus wird als modellfrei bezeichnet, wenn der Agent keinen Zugriff auf das Modell des Decision Process (DP), also dessen Transitions- und Belohnungsfunktion, hat und diese auch nicht approximiert. Modellbasierte Ansätze können hingegen die Kenntnis darüber, wie sich die Umgebung auf Reaktion von Aktionen verändert, zur Planung der eigenen Aktion nutzen. Mit dem Modell können sie beispielsweise verschiedene Szenarien im Vorhinein ausprobieren.

2.2.6 *Funktionsapproximation*

Die Aktionen wurden bisher immer als eine abstrakte Menge von Elementen ohne semantische Bedeutung und Struktur betrachtet und definiert. Die so genannten tabellarischen RL-Algorithmen erlernen Strategien auf Basis dieser abstrakten Menge von Elementen. Dies hat jedoch einen gravierenden Nachteil: Um eine sinnvolle Strategie abzuleiten, muss der Agent im Rahmen des Trainings jedes Zustands- und Aktionspaar mindestens einmal besucht haben, um überhaupt für jeden Zustand eine Aktion ermitteln zu können. Um eine optimale Aktion zu ermitteln, muss ein Zustand sogar sehr oft besucht werden. In einfachen Umgebungen mag dieses Vorgehen noch tragbar sein. In großen Zustandsräumen ist das jedoch sehr rechenintensiv und im Fall von unendlichen Zustandsräumen sogar unmöglich.

In solchen Fällen mit großen oder unendlichen Zustandsräumen werden RL-Algorithmen verwendet, die mit Funktionsapproximation arbeiten. Dabei wird versucht, eine unbekannte mathematische Funktion, die beispielsweise den Zusammenhang zwischen Zuständen und deren erwarteten Belohnungen beschreibt, zu approximieren. Als Funktionsapproximatoren werden im RL-Kontext meist künstliche neuronale Netze (NN) verwendet. Den Beobachtungen, Zuständen und Aktionen der Umgebung wohnen bestimmte Eigenschaften und Ähnlichkeitsverhältnisse inne. Mithilfe von Funktionsapproximation lassen sich diese Strukturen ausnutzen, um die optimale Strategie schneller zu erreichen. Das Ausnutzen gestaltet sich so, dass ein Lernschritt des Agents für eine erlebte Erfahrung (s, a, s') durch Funktionsapproximation sich nicht nur auf den Zustand s , sondern auch auf semantisch und strukturell ähnliche Zustände auswirkt. Um diese Eigenschaften auszunutzen, bedarf es einer semantischen Beschreibung der Zustände, Beobachtungen und Aktionen, die diese Eigenschaften und Ähnlichkeitsverhältnisse des Modells abbildet.

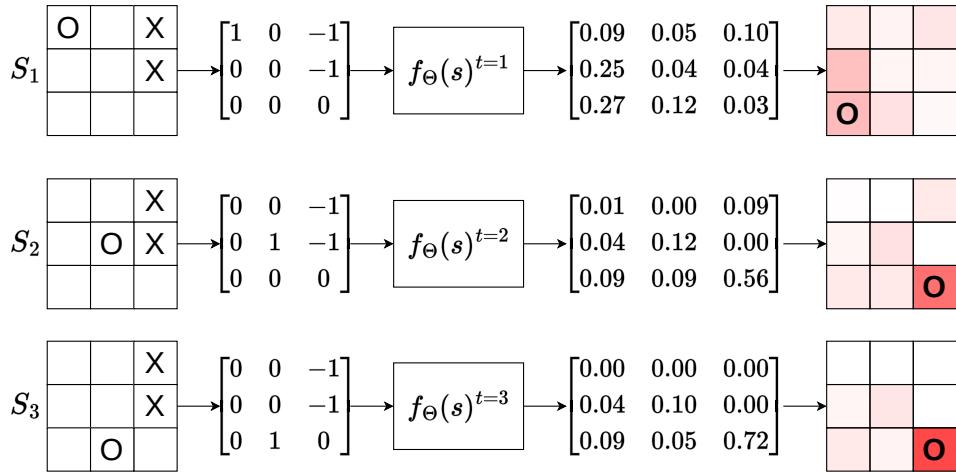


Abbildung 2.4: Mittels Funktionsapproximation kann der RL-Algorithmus auf andere Zustände generalisieren.

Beispiel 8 Die Abbildung 2.4 zeigt ein Beispiel, anhand dessen der Nutzen von Funktionsapproximation in RL-Problemen verdeutlicht werden soll. In der ersten und zweiten Spalte befindet sich eine semantische Beschreibung der darunterliegenden Spielzustände des Spiels Tic-Tac-Toe (TTT). Jedes der neun Spielfelder ist ein Merkmal des Zustands und wird durch -1 (X), 1 (O) oder 0 (leer) kodiert. Die Beschreibung eines Spielzustands erfolgt dann als Vektor mit neun Elementen. Alle vier Zustände der Abbildung haben gemeinsam, dass die optimale Reaktion von O allein die Aktion ist, bei der O die rechte untere Ecke belegt. Die Beschreibung dieser Aktion erfolgt hier in Spalte 4 als Vektor mit neun Elementen für jedes Spielfeld. Der Vektor wird vom Funktionsapproximator in Spalte 3 ausgegeben und gibt an, mit welcher Wahrscheinlichkeit die Aktion ausgewählt wird.

Die Abbildung zeigt, wie der Funktionsapproximator über mehrere Lernschritte t lernt und seine Ausgaben verändert. Aufgrund der numerischen (semantischen) Beschreibung kann er insoweit generalisieren, dass die optimale Aktion in diesem Fall nur von den Feldern oben rechts und Mitte rechts abhängt. Sind diese vom Gegner belegt, ist die optimale Aktion immer unten rechts. Die Ausgabe ist somit nur vom dritten und sechsten Vektorelement des Zustands abhängig. Der erste Lernschritt in einem Zustand S_1 wirkt sich auch auf die gewählte Aktion in ähnlichen Zuständen S_2 und S_3 aus. Der Agent kann die Semantik ausnutzen und auf andere Zustände generalisieren. In einem tabellarischen RL-Algorithmus hätte sich ein Erfahrungstupel im Zustand S_1 nur auf die Evaluation von S_1 ausgewirkt.

Mit Funktionsapproximation ist es somit nicht mehr notwendig, dass ein Agent beim Training den gesamten Zustands- und Aktionsraum durchsucht. Stattdessen kann er durch Erfahrungen lernen und anhand dieser auch für andere Zustände und Aktionen generalisieren.

2.2.7 Zusammenhang zur Spieltheorie

Die Spieltheorie beschäftigt sich mit Situationen, in denen mehrere Entscheidungsträger interagieren, wobei die Entscheidungen der Entscheidungsträger Auswirkungen auf das Ergebnis der anderen Entscheidungsträger haben. Beim RL-Problem mit nur einem Agenten und stationärer Umgebung sind diese Merkmale gerade nicht gegeben. Solche Szenarien treten vielmehr beim MARL auf. Durch mehrere Agenten, deren Aktionen sich gegenseitig beeinflussen, besteht die Verbindung zur Spieltheorie. MARL und Spieltheorie unterscheiden sich jedoch in ihrem Fokus. Die Spieltheorie zielt darauf ab, Phänomene in solchen strategischen Entscheidungssituationen zu beschreiben und zu analysieren. Zum Beispiel beschreibt die Spieltheorie das Nash-Gleichgewicht von Strategien sowie dessen Eigenschaften. Das MARL hingegen zielt darauf ab, eine solche optimale Strategie der Agenten in Umgebungen durch Interaktion mit der Umgebung und anderen Agenten durch geschicktes Ausprobieren zu erlernen. Das MARL konzentriert sich also auf Lösungskonzepte für konkrete Entscheidungssituationen. Insofern ergänzt die Spieltheorie im Rahmen dieser Arbeit das MARL, als dass die sich durch die erlernten Strategien ergebende Dynamik analysiert werden kann.

Es ist zu beachten, dass sich die Modellierung von Entscheidungssituationen in der Spieltheorie (z.B. mittels Extensivform) von der in MARL-Umgebungen unterscheiden kann. Beispielsweise können die Agenten im Falle des MARL ihre Aktionen gleichzeitig ausführen, während in den Extensivformen nur ein Spieler am Zug ist. Die Modellierung ist dem unterschiedlichen Fokus der Disziplinen geschuldet. Meistens kann die Modellierung entsprechend angepasst werden, um eine vergleichbare Modellierung zu schaffen. Im Falle von gleichzeitigen Aktionen können diese beispielsweise durch Anpassung der Entscheidungssituation auch als sequentielle Aktionen modelliert werden.

DOPPELKOPF

Das Kartenspiel Doppelkopf dient in dieser Arbeit als Beispiel für ein Spiel mit imperfekter Information, an dem algorithmische Ansätze getestet werden.

Doppelkopf ist ein in Deutschland populäres Kartenspiel für vier Spieler. Es ist ein Stichspiel, bei dem vier Spieler in zwei Parteien zusammenspielen. Ein besonders interessanter Aspekt liegt in der Parteienfindung: Zu Beginn des Spiels sind die Zusammensetzungen der Parteien meist unbekannt. Erst durch den Spielverlauf erkennen die Spieler, mit wem sie in einem Team sind.

Da es zahlreiche regionale und individuelle Variationen des Spiels gibt, werden im Folgenden die Begriffe und Regeln des Deutschen Doppelkopf-Verbandes e.V. (DDV) als Grundlage verwendet, sofern nicht explizit davon abgewichen wird [DDV].

3.1 SPIELREGELN

Zu Beginn eines Spiels werden die 48 vorhandenen Karten des Decks gleichmäßig auf die vier Spieler verteilt. Das Deck enthält die Karten 9, 10, J (Bube), Q (Dame), K (König), A (Ass) in jeweils jeder Farbe ♦ (Karo), ♥ (Herz), ♠ (Pik), ♣ (Kreuz) in doppelter Ausführung. Der Startspieler wechselt in einer Serie von Spielen reihum.

Das Spiel beginnt mit der Vorbehaltphase. In dieser Phase geben die Spieler reihum an, ob sie einen Vorbehalt anmelden möchten oder nicht. Ein Vorbehalt zeigt an, ob der Spieler ein Sonderspiel (Solo oder Hochzeit) spielen möchte und kann damit einen entscheidenden Einfluss auf das gesamte Spielgeschehen haben und den Spielausgang maßgeblich beeinflussen. Er legt fest, welche Karten als Trumpf gelten und welche Spieler zu welcher Partei gehören. Danach beginnt die Kartenphase, in der die Karten, die sich auf den Händen der Spieler befinden, nacheinander abgelegt werden. Doppelkopf ist ein Stichspiel, was bedeutet, dass in jeder Spielrunde (einem Stich) alle Spieler reihum eine Karte ausspielen. Der Spieler, der die höchste Karte gelegt hat, gewinnt den Stich. Ziel des Spiels ist es, möglichst viele Stiche zu gewinnen und dadurch Augen zu sammeln.

Ein besonderer Aspekt des Spiels ist, dass ein oder mehrere Spieler innerhalb einer Runde in Parteien zusammenspielen. Die Zusammensetzung der Parteien wird dabei im Normalspiel erst im Verlauf der Partie ersichtlich. Im Normalspiel bilden die Spieler, die jeweils eine ♣Q besitzen, die Re-Partei. Die Mitspieler dürfen zwar nicht direkt, können jedoch indirekt durch geeignete Auswahl ihrer Züge, kommunizieren.

Während der Kartenphase sind An- und Absagen möglich: Mit einer Anfrage („Re“, „Kontra“) legt ein Spieler seine Parteizugehörigkeit offen und erhöht zugleich den Punktegewinn im Fall eines Sieges seiner Partei. Hat er

eine Ansage gemacht und verliert seine Partei ist der Verlust höher. Mit Absagen können die Spieler das Risiko zusätzlich erhöhen: Sie können „unter 90“, „unter 60“, „unter 30“ und „schwarz“ absagen und wetten darauf, dass die Gegner unter einer bestimmten Augenzahl bleiben. Verliert eine Partei die Absage, bekommt die gegnerische Partei, die sonst erhaltenen Punkte und die absagende Partei entsprechende Minuspunkte.

3.2 ABWEICHUNGEN VON DEN SPIELREGELN

Um das Spiel Doppelkopf im Rahmen dieser Arbeit als geeignetes Testfeld verwenden zu können, werden einige Vereinfachungen des Grundspiels vorgenommen. Diese Anpassungen ermöglichen eine einfachere und zugänglichere Modellierung und sind insbesondere erforderlich, da die verwendeten Algorithmen MCTS und AZ auf einem Spielbaum operieren und daher keine simultanen Züge erlauben. Zudem führen sie zu einer erheblichen Steigerung der Recheneffizienz, da der Suchbaum kleiner wird. Die grundlegenden strategischen Elemente des Spiels werden dabei nicht verfälscht.

3.2.1 *Spiel statt Serie von Spielen*

In der Praxis spielen Doppelkopf-Spieler meist eine Serie von Spielen hintereinander. In dieser Arbeit wird jedoch stets nur ein einzelnes Spiel betrachtet. Daher entfallen Pflichtsoli, die in einer Serie vorgeschrieben wären. Zudem wechselt der Startspieler nicht reihum, sondern wird für jedes einzelne Spiel zufällig bestimmt. Diese Anpassungen reduzieren die Komplexität des Spielbaums erheblich. Allerdings können spielübergreifende Strategien dadurch nicht mehr untersucht werden.

3.2.2 *Vereinfachung der Vorbehaltsphase*

Die Vorbehaltsphase wird hier dahingehend angepasst, dass die Spieler zusammen mit der Anmeldung des Vorbehalts diesen auch direkt verdeckt taufen. Verdeckt in diesem Sinne bedeutet, dass der getaufte Spieltyp den anderen Spielern nicht preisgegeben wird, bis alle anderen Spieler ihren Vorbehalt angemeldet haben und ein Taufen im regulären Spielverlauf erforderlich gewesen wäre. Diese Anpassung macht den Spielbaum etwas kleiner und den Ablauf verständlicher. Die optimalen Strategien sollten sich zu den regulären Spielregeln kaum unterscheiden, da die Spieler sich bei der Anmeldung des Vorbehalts meistens bereits bewusst sind, welches Sonderspiel sie spielen möchten und es selten Anlass für Wechsel gibt.

3.2.3 *An- und Absagen als eigener Spielzug*

Im regulären Spiel können An- und Absagen jederzeit erfolgen, auch wenn gerade ein anderer Spieler am Zug ist. Diese simultanen Züge können im Spiel-

baum jedoch nicht ohne Weiteres modelliert werden. Stattdessen lassen sich die simultanen Züge in sequentielle Züge aufteilen. Dafür wird der Phase, in der die nächste Karte gespielt werden muss, eine separate An- und Absagerunde (im Folgenden: Ansagerunde) vorangestellt. In dieser Ansagerunde werden die Spieler reihum gefragt, beginnend bei dem Spieler, der die nächste Karte im Stich legen muss, ob sie eine An- oder Absage machen möchten. Dies ist nur möglich, wenn die Spieler nach den Spielregeln noch eine An- oder Absage tätigen können. Diese Änderung macht die Modellierung als Spielbaum erst möglich.

3.2.4 Keine Spielabkürzungen

Spielabkürzungen im Sinne des Art. 5.4 der DDV-TSR sind nicht möglich. Spielabkürzungen sind Aktionen eines Spielers, mit denen er die verbleibenden Züge des Spiels in den Fällen, in denen kein anderes Spielergebnis mehr möglich ist, überspringt.

3.3 DOPPELKOPF IN EXTENSIVFORM

Das angepasste Doppelkopf-Spiel kann als Spiel mit imperfekter Information und Zufallsereignissen in Extensivform modelliert werden.

3.4 GRUNDLEGENDE SPIELSTRUKTUR

Doppelkopf wird im Uhrzeigersinn gespielt. Die Spieler werden in dieser Arbeit soweit nicht anders angegeben relativ betrachtet, wobei Unten immer der gerade betrachtete Spieler ist. Die Spieler sind dann:

$$N = \{\text{Unten, Links, Oben, Rechts}\}$$

Ein Spielzustand $v \in V$ wird eindeutig durch die folgenden Eigenschaften identifiziert.¹

- Die aktuellen Handkarten aller vier Spieler.
- Die in den Stichrunden gespielten Karten.
- Die von den Spielern in der Vorbehalsrunde getätigten Vorbehalte oder Gesundmeldungen.
- Die in den Stichrunden getätigten An- und Absagen beziehungsweise Nicht-Ansagen.

Im Startzustand x^0 sind noch keine Handkarten verteilt, keine Vorbehalte angesagt, keine Karten gespielt und keine An- und Absagen gemacht. Auf den

¹ Zustände im Sinne der Spieltheorie sind abstrakt zu verstehen. Anschaulich lässt sich für jede Eigenschaftskombination ein Zustand definieren.

Startzustand folgt das einzige in Doppelkopf zu modellierende Zufallereignis: Die initiale Verteilung der im gesamten Kartendeck verfügbaren Karten auf die Hände der Spieler. Die verbleibenden Knoten und Kanten des Spielbaums ergeben sich dann aus den Spielregeln von Doppelkopf mit den oben definierten Einschränkungen. Die Blattknoten sind jene Knoten, in denen alle Spieler ihre Handkarten ausgespielt haben.

3.4.1 Informationsmengen

Eine Informationsmenge $(U_i^j, A(U_i^j))$ für einen Spieler i ist eine Teilmenge aller Spielzustände, die aus seiner Sicht noch möglich sind. Eine Informationsmenge eines Spielers wird dann aus folgenden Eigenschaften eindeutig beschrieben:

- Seine eigenen Handkarten.
- Die in den Sticherunden gespielten Karten.
- Die Vorbehalte und Gesundansagen sowie gegebenenfalls getaufte Vorbehalte der Vorbehaltsrunde.

Die Karten der anderen Mitspieler sowie gegebenenfalls nicht getaufte Vorbehalte kennt der Spieler nicht. Die Menge der möglichen Spielzustände ergibt sich dann aus dem dem Spieler bekannten Teil des Spielzustands sowie den noch möglichen Kartenverteilungen und gegebenenfalls möglichen Taufen. Es handelt sich hierbei um eine Modellierung von Informationsmengen, bei der jeder Spieler perfektes Erinnerungsvermögen hat. Dadurch, dass alle gelegten Karten und die gemachten Vorbehalte Teil der Definition sind, kann der Spieler die durchgeföhrten Aktionen rekonstruieren, die zu der aktuellen Informationsmenge geföhrt haben. Dadurch gelten die unter 2.1.3.9 gemachten Ausführungen und eine Verhaltensstrategie hat eine äquivalente gemischte Strategie und umgekehrt.

3.4.2 Nutzenfunktion

In Doppelkopf wird am Ende eines Spiels ein numerischer Punktwert ermittelt, welcher der Gewinnerpartei als positive Punktzahl und der Verlustpartei als negative Punktzahl zugewiesen wird. Die Punktzahlen summieren sich stets zu 0. Somit sind beispielsweise Ergebnisse wie $[+2, -2, +2, -2]$ und $[-1, +3, -1, -1]$ möglich. Jeder dieser erreichbaren Kombinationen ist ein möglicher Spielausgang $o \in O$.

Praktischerweise kann durch diese Ermittlung auch direkt eine sinnvolle Nutzenfunktion abgeleitet werden. Ein Spieler bevorzugt ein Ergebnis, das ihm eine höhere Punktzahl zuordnet. Die Ergebnisse sind somit bereits als Nutzenvektoren interpretierbar. Beispielsweise gilt:

$$[+3, -1, -1, -1] \succsim_{playerbottom} [+2, -2, +2, -2]$$

In dieser Modellierung ist ein Solo-Gewinn mit sechs Punkten ($[+6, -2, -2, -2]$) genauso nützlich wie ein Gewinn mit einem anderen Spieler zusammen ($[+6, -6, +6, -6]$).

3.5 DOPPELKOPF ALS RL-PROBLEM

Doppelkopf kann und soll im Folgenden auch als MARL-Problem mit einer partiell beobachtbaren Umgebung betrachtet werden. Auf eine genaue Definition der Umgebung soll hier verzichtet werden, um Redundanzen zum vorigen Kapitel zu vermeiden. Es gibt im Wesentlichen drei Besonderheiten:

- Obwohl das MA-POMDP simultane Züge erlaubt, wird in der Modellierung ausschließlich von sequentiellen Zügen ausgegangen.
- Die Aktionsmenge ist diskret und endlich.
- Die Übergangsfunktion ist bis auf den Initialzustand, auf den folgend die Karten verteilt werden, deterministisch.

3.6 KOMPLEXITÄT

Doppelkopf ist mit seiner strategischen und algorithmischen Komplexität im unteren Mittelfeld von bekannten Brett- und Kartenspielen anzusiedeln. Prinzipiell zeichnet es sich durch eine hohe Anzahl möglicher Spielzustände und einer hohen Anzahl von möglichen Beobachtungen aus. Der sogenannte Verzweigungsfaktor, der beispielsweise für die Effizienz des MCTS von Bedeutung ist, ist jedoch eher klein.

3.6.1 Spielbaumkomplexität

Ein Maß zur Abschätzung der Komplexität eines Spiels ist die sogenannte Spielbaumkomplexität. Diese setzt sich aus dem Verzweigungsfaktor b (*branching factor*) und der durchschnittlichen Anzahl der Spielzüge d zusammen [All94]. Der Verzweigungsfaktor beschreibt die durchschnittliche Anzahl möglicher Aktionen, die einem Spieler in einem beliebigen Zustand innerhalb des Spielbaums zur Verfügung stehen. In Kombination mit der durchschnittlichen Anzahl der Spielzüge d ermöglicht der Verzweigungsfaktor eine Abschätzung der Größe des Spielbaums und somit des Suchraums, innerhalb dessen ein RL-Algorithmus oder ein Suchalgorithmus operieren muss.

Experimentell wurde im Rahmen der Evaluation für das Spiel Doppelkopf mit den vorgestellten Abweichungen ein Verzweigungsfaktor von $3,08 \pm 0,19$ festgestellt.² Gleichzeitig dauerte eine Partie im Durchschnitt etwa $97 \pm 10,34$ Züge. Diese Werte wurden unter der Annahme ermittelt, dass alle 4 Spieler bereits eine relativ gute Strategie verfolgen. Sie sind daher als grobe Einschätzung zu verstehen.

² Die Daten beruhen auf den Experimenten aus Kapitel 7, bei denen 4 MCTS-Spieler mit jeweils 100.000 Iterationen gegeneinander spielten.

Mit diesen Werten ist Doppelkopf hinsichtlich der Spielbaumkomplexität als moderat komplex einzuordnen. Es ist deutlich komplexer als einfache Spiele wie TTT ($b \approx 4, d \approx 9$) oder wie 4 gewinnt ($b \approx 4, d \approx 36$ [All94] [Tro10]), jedoch deutlich weniger komplex als Schach ($b \approx 35, d \approx 70$ [Sha88]) oder Go ($b \approx 250, d \approx 211$ [All94] [Bro]).

3.6.2 Anzahl möglicher Spielzustände

Die Anzahl der möglichen Spielzustände, die von der Startposition eines Spiels aus erreicht werden können, bezeichnet man als Zustandskomplexität [All94]. Allein für die initiale Verteilung der Karten auf vier Spieler ergeben sich etwa $2,25 \cdot 10^{21}$ mögliche Kombinationen [SH15]. Jeder von diesen Startzuständen aus erreichbare Zustand trägt zur Zustandskomplexität bei. [SH15] geben als grobe obere Schranke eine Zustandskomplexität von $2,4 \cdot 10^{39}$ an. Diese Schätzung berücksichtigt jedoch ausschließlich Kartenverteilungen und ignoriert weitergehende Regeln wie Vorbehalte oder An- und Absagen, weshalb sie nur als grobe obere Schranke betrachtet werden kann.

METHODEN UND ALGORITHMEN

Aufbauend auf den in Kapitel 2 gelegten Grundlagen, werden in diesem Kapitel die beiden Algorithmen der Forschungsfrage sowie die notwendigen Komponenten für die Zusammensetzung zu einer Strategie für Spiele mit imperfekter Information mittels Determinisierung beschrieben.

4.1 MONTE CARLO TREE SEARCH

Monte-Carlo Tree Search (MCTS) ist ein iterativer Suchalgorithmus in Bäumen, der von [KSW06] und [Coug06] entwickelt wurde, um einen Computerspieler für das Brettspiel Go zu realisieren. Der hier beschriebene Ablauf des MCTS sowie die verwendete Notation folgen, sofern nicht anders angegeben, der Darstellung in [Świ+23].

4.1.1 *Grundform*

MCTS ist ein iterativer Algorithmus, welcher den Spielbaum eines Spiels durchsucht und im Rahmen dieser Suche statistische Informationen über die besuchten Spielzustände und ausgeführten Aktionen ableitet. Diese statistischen Informationen liefern Rückschlüsse darauf, welche Aktion die beste Aktion für den Spieler ist. In seiner Grundform ist MCTS nur auf Spiele mit perfekter Information und ohne Zufallsereignisse anwendbar.

Der Spielbaum eines Spiels ist, in komplexeren Spielen so groß, dass er in vertretbarer Rechenzeit nicht vollständig durchsucht werden kann. Betrachtet man dabei den Spielbaum unter dem Gesichtspunkt, welche Aktion zu einem günstigen Spielausgang führt, so erkennt man schnell Zustände, die einen in eine deutlich schlechtere Position bringen als Alternativen. MCTS ist dabei ein Ansatz, der nur vielversprechende Teilbäume des Spielbaums durchsucht und damit die Rechenzeit deutlich reduziert. Welche Teilbäume vielversprechend sind, wird durch sogenannte Monte-Carlo (MC)-Simulationen bestimmt. Dadurch erhält der MCTS auch seinen Namen.

MCTS wird meist mit einer festen Anzahl von Iterationen parametrisiert. Es handelt sich um einen sogenannten Anytime-Algorithmus. Das bedeutet, dass er nach jeder Iteration gestoppt werden kann und das bis dahin beste gefundene Ergebnis zurückgibt. Mit jeder Iteration wird das Ergebnis verbessert.

Als Eingabe nimmt der MCTS einen Spielzustand $s_{root} \in V$ an und gibt eine von diesem Spielzustand aus mögliche Aktion zurück. Damit erfüllt der MCTS die Definition einer Verhaltensstrategie. Es handelt sich dabei jedoch nicht um eine deterministische Strategie, da die MC-Simulationen dem Zufall unterliegen und damit jeder Aufruf des Algorithmus zu einem anderen Ergeb-

nis führen kann. Diese Instabilität des Algorithmus wird jedoch mit einer zunehmenden Anzahl von Iterationen kleiner. In ihrer Arbeit zeigen [KSW06], dass die Wahrscheinlichkeit, die optimale Aktion zu wählen, mit zunehmender Anzahl an Iterationen gegen 1 konvergiert. Somit wird die Strategie mit steigender Iterationsanzahl deterministischer.

Während seiner Ausführung erzeugt der MCTS einen Teilbaum des Spielbaums. Initial besteht der Baum nur aus dem Wurzelknoten, der den Zustand s_{root} repräsentiert, in dem sich der Spieler, für den eine Aktion ermittelt werden soll, gerade befindet. Der während der Ausführung entstehende Teilbaum wird auch Selektionsteilbaum genannt. Jede Iteration des MCTS besteht dann aus vier Phasen:

4.1.1.1 Phasen in einer Iteration

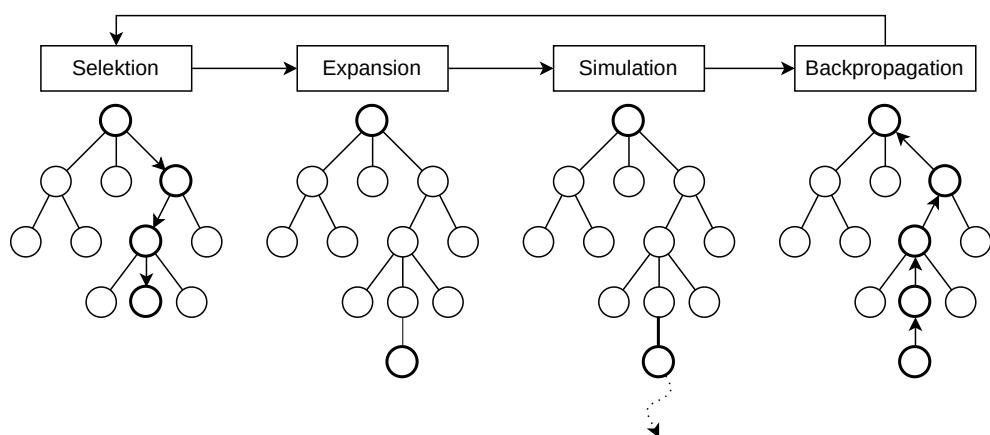


Abbildung 4.1: Die vier Phasen des MCTS [Świ+23].

SELEKTION In der Selektionsphase wird ausgehend vom Wurzelknoten ein Blattknoten des Selektionsteilbaums ausgewählt. Dabei erfolgt die Auswahl jedes Kindknotens anhand einer vorgegebenen Selektionsstrategie. Dies erfolgt, bis ein Blattknoten erreicht wird. Dieser Blattknoten wird für diese Iteration als *vielversprechender* Knoten betrachtet.

EXPANSION In der Expansionsphase wird der Selektionsteilbaum um die vom vielversprechenden Spielzustand aus erreichbaren Spielzustände erweitert. Falls der vielversprechende Knoten bereits ein Endzustand des Spiels ist, wird die Simulationsphase übersprungen und mit der Backpropagation fortgefahrene.

SIMULATION In der Simulationsphase wird ein zufälliger Kindknoten des expandierten vielversprechenden Knotens ausgewählt (Simulationsknoten). Von diesem Simulationsknoten aus wird eine vollständige Simulation des Spiels bis zu einem Endzustand durchgeführt. In dieser Simulation wählt jeder Spieler unter den legalen Aktionen zufällig.

BACKPROPAGATION Die Backpropagation beginnt, sobald in der Expansionsphase ein Endzustand als vielversprechender Knoten erreicht oder eine vollständige Simulation durchgeführt wurde.

Ausgehend vom Simulationsknoten werden entlang des Selektionsteilbaums alle übergeordneten Knoten bis zum Wurzelknoten durchlaufen und deren gespeicherte Statistiken aktualisiert. Die Statistiken werden auf Basis des im Rahmen der Simulation erreichten Spielausgangs (Nutzen) aktualisiert. Falls der vielversprechende Knoten bereits ein Spielendzustand war, wird der Spielausgang des vielversprechenden Knotens verwendet. Dieser Prozess kann auch in mehreren Iterationen erfolgen. Dadurch kann der gleiche Spielverlauf die Statistiken des Selektionsteilbaums mehrfach beeinflussen.

In der Regel werden die folgenden Statistiken geführt und aktualisiert:

- $N(s)$: Anzahl der Besuche des Spielzustands s innerhalb des Selektionsteilbaums.
- $N(s, a)$: Anzahl der Besuche der Aktion a ausgehend von Spielzustand s innerhalb des Selektionsteilbaums.
- $Q(s, a)$: Durchschnittlicher Nutzen, der in den vorherigen Iterationen nach Ausführung der Aktion a im Zustand s erzielt wurde. Dieser Wert bewertet den Spielzustand aus Sicht des Spielers, der in s am Zug ist.

4.1.1.2 Aktionsauswahl

Nach Abschluss aller vorgesehenen Iterationen wird die Aktion ausgewählt, welche den höchsten durchschnittlichen Nutzen für den aktuellen Spieler im Zustand s_{root} aufweist. Formell wird also die Aktion a_{root}^* bestimmt:

$$a_{root}^* = \arg \max_{a \in A(s_{root})} \{Q(s_{root}, a)\}$$

Dabei gilt:

- $A(s)$ ist die Menge der möglichen Aktionen im Zustand s .
- $Q(s, a)$ ist die oben definierte Bewertungsfunktion, die den durchschnittlichen Nutzen einer Aktion a im Zustand s angibt.

In vielen Veröffentlichungen wird jedoch anstelle dieser Aktion diejenige Aktion gewählt, die am häufigsten besucht wurde, also die Aktion mit maximalem $N(s, a)$. Laut [Cha+08] zeigen eine Vielzahl von Experimenten, dass die Wahl der Aktion bei einer hohen Anzahl von Iterationen keinen signifikanten Unterschied aufweist. Bei einer geringeren Iterationszahl lieferte $N(s, a)$ bessere Ergebnisse.

4.1.2 Upper Confidence Bounds applied for Trees

Die vermutlich bekannteste und am häufigsten eingesetzte Selektionsstrategie ist der von [KSW06] vorgestellte Upper Confidence Bounds applied for

Trees (UCT). Beim MCTS mit UCT erfolgt die Selektion des vielversprechenden Knotens auf Grundlage des UCT-Werts. In der Selektionsphase wird vom Wurzelknoten ausgehend die Aktion mit dem höchsten UCT-Wert gewählt, bis der vielversprechende Knoten zur Expansion erreicht wird.

Definition 20 (Berechnung des UCT) Der UCT-Wert berechnet sich wie folgt:

$$uct(s, a) = \begin{cases} \infty & \text{wenn } N(s, a) = 0, \\ Q(s, a) + C \cdot \sqrt{\frac{\ln N(s)}{N(s, a)}} & \text{sonst.} \end{cases}$$

wobei zusätzlich:

- C eine Konstante ist, welche den Explorationsgrad des Algorithmus bestimmt.

Definition 21 (Selektionsstrategie UCT) Die UCT-Selektionsstrategie folgt in der Selektionsphase jeweils der Aktion a^* :

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

Die Idee hinter der Selektionsstrategie liegt darin, Exploration und Exploitation auszugleichen:

- Der Term $Q(s, a)$ bewertet die Aktionen hinsichtlich ihres bereits ermittelten durchschnittlichen Nutzens. Aktionen, die in vorherigen Iterationen bereits zu guten Ergebnissen geführt haben, werden häufiger ausgewählt. Anders formuliert: Der Term nutzt bereits erlangtes Wissen über vielversprechende Aktionen aus (Exploitation).
- Der Term $\sqrt{\frac{\ln N(s)}{N(s, a)}}$ priorisiert Aktionen, die bislang selten gewählt wurden. Der Algorithmus erkundet somit neue, bislang selten betrachtete Aktionen, hinter denen sich bessere Ergebnisse (und ein höherer durchschnittlicher Nutzen) befinden könnten, als zuvor gedacht.

Der Explorationsparameter C dient als Hyperparameter, der die Balance zwischen Exploration und Exploitation bestimmt. Ein hoher Wert sorgt für mehr Exploration, ein niedriger für mehr Exploitation. In Spielen, deren Nutzen im Intervall $[-1, 1]$ liegt (zum Beispiel Verlust oder Gewinn), wird häufig ein Wert von $C = \sqrt{2}$ verwendet [Świ+23]. Grundsätzlich ist der Parameter aber abhängig vom untersuchten Spiel und muss gegebenenfalls experimentell bestimmt werden.

Angemerkt sei, dass in einem Zustand s stets die Aktion mit dem höchsten UCT-Wert aus Sicht des Spielers p , der in s an der Reihe ist, selektiert wird. Dadurch wird erreicht, dass die beste Aktion für den Spieler p_{root} des Wurzelzustands s_{root} unter der Annahme gewählt wird, dass die anderen Spieler ihren eigenen Nutzen ebenfalls maximieren wollen.

4.1.3 Funktionsweise

Die zentrale Idee des MCTS besteht darin, Aktionen und Zustände iterativ immer genauer zu bewerten. Auf Grundlage dieser Bewertungen entsteht ein Ausgleich zwischen Exploration und Exploitation, der wiederum bestimmt, welche Aktionen und Zustände im weiteren Verlauf genauer untersucht und bewertet werden sollen.

Der MCTS bewertet hierfür Zustände und Aktionen innerhalb des Selektionsteilbaums. Die Bewertung erfolgt auf Basis der Ergebnisse der MC-Simulationen des Spielzustands sowie der von diesem Zustand aus erreichbaren Spielzustände. Somit kann eine Aktion und der gesamte darunterliegende Teilbaum des Spielbaums approximativ bewertet werden. Zwar basiert jede Simulation zunächst auf zufälligem Verhalten und stellt damit keine gute Schätzung dar. Durch eine hohe Anzahl solcher Simulationen und der Tatsache, dass auch zunehmend tiefere Spielzustände untersucht werden, steigt die Genauigkeit mit zunehmender Erkundung des Teilbaums.

Diese Bewertungen dienen anschließend als Entscheidungsgrundlage dafür, welche Teilbäume genauer erkundet werden sollen. Dabei werden vor allem solche Aktionen und Teilbäume häufiger untersucht, die sich im bisherigen Verlauf als vielversprechend erwiesen haben. Die Bewertung dieser Teilbäume verbessert sich damit, wodurch sie genauer mit anderen Teilbäumen beziehungsweise anderen vielversprechenden Aktionen verglichen werden können. Andererseits werden Teilbäume, die bereits nach wenigen Simulationen schlechte Ergebnisse liefern, weniger oft untersucht. Durch die Exploration stellt der Algorithmus sicher, dass auch weniger häufig besuchte Aktionen berücksichtigt werden, um möglicherweise gute Aktionen nicht zu übersehen.

Die UCT-Selektionsstrategie ermöglicht genau diesen selbst ausgleichenden Mechanismus zwischen Exploration und Exploitation. Als Konsequenz aus diesem Gesamtvergehen muss nur ein Bruchteil des Spielbaums durchsucht werden, was den MCTS deutlich effizienter als eine vollständige Durchsuchung des Spielbaums macht.

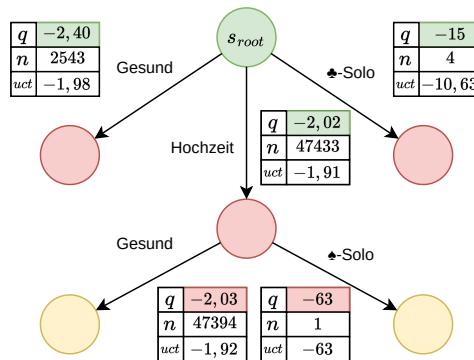


Abbildung 4.2: Ein Ausschnitt eines Selektionsteilbaum in der Vorbehaltsphase eines Doppelkopf-Spiels.

Beispiel 9 (Auswahl eines Solos im MCTS) Abbildung 4.2 zeigt einen Ausschnitt aus einem Selektionsteilbaum nach 50.000 Iterationen. Spieler Grün ist aktuell an der Reihe und muss zwischen einer Hochzeit und einem stillen Solo (Gesund) wählen. Beide Alternativen werden vom Algorithmus häufig besucht, da sie gegenüber den Alternativen vielversprechend erscheinen. Das ♣-Solo hingegen wird schon nach wenigen Iterationen kaum weiter betrachtet, da es deutlich schlechtere Ergebnisse liefert. Gut zu erkennen ist, dass die UCT-Werte der beiden Aktionen nahe beieinander liegen, weshalb beide Optionen intensiv untersucht wurden. Dies spiegelt den Ausgleich wider, den UCT vornimmt. Entscheidet sich Grün für die Hochzeit, so nimmt er an, dass Rot Gesund ansagen wird, da dies aus Sicht von Rot mit durchschnittlich $-2,03$ Punkten den besten Nutzen verspricht.

4.1.4 Zusammenhang zur Spieltheorie

Der MCTS (mit UCT) agiert auf dem Spielbaum, wie er in der Extensivform für Spiele mit perfekter Information definiert ist, und liefert bei unendlich vielen Iterationen annähernd eine deterministische Strategie. Für Nullsummenspiele mit zwei Spielern und perfekter Information weisen Kocsis und Szepesvári in [KS06] und [KSW06] nach, dass der Algorithmus im Grenzwert gegen die optimale Strategie und somit gegen das deterministische Nash-Gleichgewicht konvergiert (vgl. auch [Bro+12]). Für Spiele mit mehr als zwei Spielern oder Nicht-Nullsummenspiele folgt dies jedoch nicht.

4.1.5 Zusammenhang zum Reinforcement Learning

MCTS wird nicht als RL-Algorithmus klassifiziert, da es ihm am zentralen Element, dem Lernen durch Interaktion mit einer Umgebung, fehlt. Vielmehr handelt es sich um einen statischen Suchalgorithmus. Dennoch ist es in bestimmten Fällen möglich, mittels MCTS Strategien für MDP und auch MA-MDP zu ermitteln. Voraussetzung hierfür ist jedoch, dass keine Zufallsereignisse auftreten. Das bedeutet, die Transitionsfunktion muss deterministisch sein (Übergangswahrscheinlichkeit gleich 1). Die Umgebung muss zudem episodisch sein und lediglich am Ende der Episode eine Belohnung zurückgeben. Schließlich ist der Einsatz auf diskrete Aktionsräume beschränkt.

4.2 FUNKTIONSSAPPROXIMATION MITTELS NEURONALER NETZE

Moderne Verfahren des RL basieren häufig auf Funktionsapproximation, um auch in Umgebungen mit hoher Komplexität effektiv agieren zu können. Eine dabei zentrale Methode zur Funktionsapproximation sind künstliche neuronale Netze (NN).

Ziel eines NN ist es, eine unbekannte Funktion $y = f(x)$ durch eine approximierende Funktion $y = f^*(x, \theta)$ nachzubilden. Die Parameter θ beschreiben dabei die Gewichtungen des Netzes, welche die Approximation steuern. Sie werden auf Grundlage von Trainingsdaten mithilfe von Optimierungsverfah-

ren iterativ angepasst. Für diese iterative Anpassung der Parameter zur Approximation reicht es aus, Beispieleingaben x und die zugehörigen Ausgaben y der zu approximierenden Funktion zu kennen. Direktes Wissen über die grundlegende Funktion f ist nicht erforderlich. Die Namensgebung der NN ist lose an die Funktionsweise biologischer Neuronen angelehnt, deren Prinzip sie in stark vereinfachter Form nachbilden.

Die Erklärungen zu MLP und Optimierungsverfahren basieren im Wesentlichen auf dem Lehrbuch *Deep Learning* von [GBC16].

Der folgende Abschnitt ist wie folgt aufgebaut: Zunächst wird die grundlegende Architektur des sogenannten MLP eingeführt. Im Anschluss wird erläutert, wie neuronale Netze dieses Typs mithilfe des Gradientenabstiegsverfahrens trainiert werden können. Dieses Optimierungsverfahren dient als Basis und lässt sich auch auf komplexere Architekturen, wie die darauffolgend vorgestellten Transformer-Architekturen, anwenden.

4.2.1 Multilayer Perceptron

Ein Mehrschicht-Perceptron (MLP) ist eine spezielle Form eines NN. Es approximiert die Zielfunktion durch mehrere hintereinandergeschaltete Schichten von Neuronen, die jeweils einfache nichtlineare Funktionen berechnen. Ein MLP nimmt einen Eingabevektor entgegen und erzeugt daraus einen Ausgabevektor.

4.2.1.1 Aufbau eines MLP

Die Funktionalität eines MLP lässt sich am besten als Netzwerkstruktur beschreiben. Ein Knoten stellt in solch einer Darstellung ein Neuron dar. Ein Neuron erhält ein oder mehrere Eingabewerte: Entweder Werte des Eingabevektors oder Ausgaben vorhergehender Neuronen.

Die Eingaben x_i eines Neurons werden durch eingehende Kanten gekennzeichnet. Für jede Eingabe eines Neurons wird ein (optimierbares) Gewicht w_i vorgehalten. Zusätzlich wird für jedes Neuron eine für alle Eingaben geltende Aktivierungsfunktion ϕ definiert und ein (optimierbarer) Bias b gespeichert. Die Ausgabe eines einzelnen Neurons berechnet sich dann wie folgt:

Definition 22 (Ausgabe eines Neurons) Der Ausgabewert y eines Neurons in einem NN berechnet sich wie folgt:

$$y = \phi \left(\sum_{i=1}^n w_i x_i + b \right)$$

wobei

- x_i die i -te Eingabe des Neurons ist,
- w_i das i -te (optimierbare) Gewicht für die i -te Eingabe des Neurons ist,
- b der (optimierbare) Bias des Neurons ist,

- ϕ die für das Neuron geltende Aktivierungsfunktion ist.

In einem MLP werden mehrere Neuronen, welche die gleichen Eingabewerte erhalten, in einer Schicht (Layer) zusammengefasst. Diese bilden gemeinsam eine Funktion, die einen Eingabevektor in einen Ausgabevektor transformiert. Zur Vereinfachung der Notation werden die Gewichte der Neuronen einer Schicht in einer gemeinsamen Matrix W sowie die Eingaben als Vektor \vec{x} , die Ausgaben als Vektor \vec{y} und Biase als Vektor \vec{b} dargestellt.

Definition 23 (Schicht eines neuronalen Netzes) Der Ausgabevektor \vec{y} einer Schicht in einem MLP berechnet sich wie folgt:

$$\vec{y} = \phi(\vec{x}W^T + \vec{b})$$

wobei

- \vec{y} den Ausgabevektor der Schicht bezeichnet,
- \vec{x} der Eingabevektor ist. Im Falle einer vorgelagerten Schicht handelt es sich um deren Ausgabe; bei der ersten Schicht um die Eingabe des MLP,
- W eine Matrix der (optimierbaren) Gewichte der Neuronen ist,
- \vec{b} der Bias-Vektor der Schicht ist.

Ein MLP besteht aus mehreren solcher (verborgenen) Schichten, wobei die Ausgaben einer Schicht als Eingaben für die jeweils nachfolgende Schicht dienen. Die gesamte durch das MLP definierte Funktion $y = f^*(x, \theta)$ ergibt sich als Komposition der Einzelfunktionen der Schichten:

Definition 24 (MLP-Funktion) Ein MLP definiert eine Funktion $f^*(x, \theta)$, die eine Eingabe x auf eine Ausgabe y abbildet. Diese Funktion ist definiert durch eine Verkettung der durch die Schichten definierten Funktionen:

$$f^*(x, \theta) = \phi^{(L)} \left(\phi^{(L-1)} \left(\dots \phi^{(1)} \left(xW^{(1)T} + b^{(1)} \right) \dots \right) W^{(L)T} + b^{(L)} \right)$$

wobei

- L die Anzahl der versteckten Schichten (hidden layer) im MLP ist,
- $\phi^{(k)}$ die Aktivierungsfunktion der k -ten Schicht ist,
- $W^{(k)}$ die Gewichtsmatrix der k -ten Schicht ist. Sie ist Teil von θ ,
- $b^{(k)}$ der Bias-Vektor der k -ten Schicht ist. Er ist Teil von θ ,
- x der Eingabevektor ist.

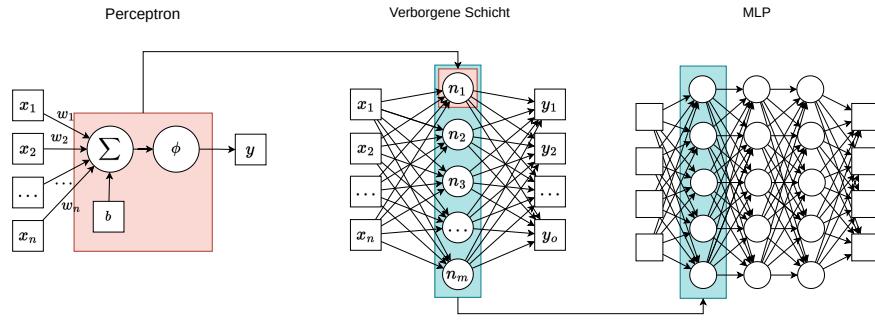


Abbildung 4.3: Vom einzelnen Neuron zum vollständig verbundenen MLP.

4.2.1.2 Universelle Approximation

Das sogenannte Universelles Approximationstheorem (UAT) liefert eine theoretische Grundlage für die Leistungsfähigkeit eines MLP zur Funktionsapproximation. Es besagt im Kern, dass bereits ein MLP mit nur einer einzigen verdeckten Schicht jede stetige Funktion mit beliebiger Genauigkeit approximieren kann. Voraussetzung ist, dass die Schicht genügend Neuronen enthält und eine nichtlineare Aktivierungsfunktion verwendet wird [HSW89]. Diese Nichtlinearität ist wichtig, da ohne sie neuronale Netze nur lineare Transformationen abbilden könnten und somit nicht jede stetige Funktion approximieren könnten. Das Theorem untermauert die prinzipielle Fähigkeit von NN, nämlich eine Vielzahl von Funktionen zu approximieren.

Die praktische Aussagekraft des UAT ist jedoch begrenzt. Es handelt sich primär um eine Existenzaussage. Das UAT garantiert zwar, dass eine Parameterkonfiguration θ existiert, die eine Funktion approximiert. Es macht jedoch keine Aussage darüber, ob ein konkreter Trainingsprozess diese Parameterkonfiguration tatsächlich findet. Zum anderen macht das Theorem keine Aussage über die Effizienz einer Architektur. Obwohl theoretisch eine einzelne verdeckte Schicht ausreicht, werden in der Praxis deutlich komplexere und tiefere NN zur Funktionsapproximation verwendet. Diese können häufig mit weniger Parametern eine bessere Leistung erzielen und lassen sich auch stabiler trainieren.

In Konsequenz bedeutet dies: Das UAT bestätigt die grundlegende Mächtigkeit von NN. Die erfolgreiche Anwendung in der Praxis erfordert jedoch meistens deutlich komplexere und vor allem tiefere Architekturen.

4.2.2 Optimierung eines neuronalen Netzes

Bisher wurde nur festgestellt, dass ein MLP durch die geeignete Wahl der Parameter θ eine Funktion repräsentieren kann, welche die Zielfunktion approximiert. Prinzipiell gibt es mehrere Möglichkeiten, Parameter zu ermitteln, welche die Funktion approximieren. Der große Erfolg von NN beruht jedoch maßgeblich auf Gradientenabstiegsverfahren.

4.2.2.1 Verlustfunktion

Für die Evaluierung der Approximationsqualität eines NN wird die sogenannte Verlustfunktion aufgestellt. Sie misst für einen einzelnen Datenpunkt, wie stark eine Vorhersage $f^*(x, \theta)$ des Approximators vom tatsächlichen Zielwert $y = f(x)$ abweicht. Die Verlustfunktion J definiert also eine Metrik für den punktuellen Fehler.

Über eine reine Funktionsapproximation hinaus wird bei der Aufstellung des erwarteten Verlusts auch die Verteilung der vorliegenden Daten, mit der der Funktionsapproximator trainiert wird, berücksichtigt. Der erwartete Verlust $J^*(\theta)$ aggregiert diesen punktuellen Fehler über die zugrunde liegende (aber oft unbekannte) Datenverteilung p_{data} aller möglichen Ein- und Ausgabepaare (x, y) .

$$J^*(\theta) = \mathbb{E}_{(x,y) \sim p_{\text{data}}} L(f^*(x, \theta), y)$$

Das Ziel eines Optimierungsverfahrens, insbesondere des Gradientenabstiegsverfahrens, ist es, die Parameter θ so zu wählen, dass dieser erwartete Verlust $J^*(\theta)$ minimiert wird.

Bei komplexen Problemen können nicht alle möglichen Ein- und Ausgaben der tatsächlichen Datenverteilung p_{data} hinsichtlich des erwarteten Verlusts überprüft werden. Vielmehr steht im Rahmen des maschinellen Lernens meist nur ein Ausschnitt der Verteilung von Ein- und Ausgaben der Zielfunktion zur Verfügung. Diese Verteilung wird als Trainingsdatensatz \hat{p}_{data} bezeichnet. Statt nun den erwarteten Verlust der echten Verteilung zu minimieren, wird versucht, den erwarteten Verlust der Trainingsdatenverteilung zu minimieren. Idealerweise führt das dazu, dass die dadurch trainierte Funktion $f^*(x, \theta)$ auch für andere Elemente aus p_{data} , die nicht Teil der Trainingsdaten sind, eine gute Approximation liefert. Man spricht dann davon, dass die Funktion generalisiert:

$$J(\theta) = \mathbb{E}_{(x,y) \sim \hat{p}_{\text{data}}} L(f^*(x, \theta), y),$$

Die tatsächliche Berechnung des Verlusts ist stark problembezogen. Zwei sehr häufig verwendete Verlustfunktionen sind die sogenannte mittlere quadratische Abweichung (MSE)-Verlustfunktion und die Cross Entropy (CE)-Verlustfunktion.

MSE-VERLUSTFUNKTION Die sogenannte mittlere quadratische Abweichung (MSE)-Verlustfunktion ist eine Verlustfunktion, die im maschinellen Lernen für Regressionsaufgaben verwendet wird. Dies sind Aufgaben, bei denen für eine Eingabe ein kontinuierlicher Wert vorhergesagt werden soll ($f : \mathbb{R}^n \rightarrow \mathbb{R}$). Die Verlustfunktion misst für eine Menge von Beispielen den durchschnittlichen quadratischen Fehler zwischen Vorhersage und tatsächlichem Wert:

$$\text{MSE} = \frac{1}{m} \sum_{i=1}^m (\hat{y}^{(i)} - y^{(i)})^2$$

wobei

- $\hat{y}^{(i)}$ die Vorhersage des Modells für das Beispiel i ist,

- $y^{(i)}$ der tatsächliche Wert des Beispiels i ist.

CROSS-ENTROPY-VERLUSTFUNKTION In Klassifikationsaufgaben soll ein Modell vorhersagen, zu welcher Klasse eine Eingabe gehört. Im erweiterten Sinne gehört dazu auch die Ermittlung der Wahrscheinlichkeitsverteilung über mögliche Klassen. Die Cross Entropy (CE)-Verlustfunktion wird verwendet, um zu messen, wie stark die vom Modell vorhergesagte Wahrscheinlichkeitsverteilung von der tatsächlichen Zielverteilung abweicht:

$$\text{CrossEntropy} = -\frac{1}{m} \sum_{i=1}^m \sum_{j=1}^K p_j^{(i)} \log q_j^{(i)}$$

wobei

- m die Anzahl der Beispiele im Datensatz ist,
- K die Anzahl der Klassen ist,
- $p_j^{(i)}$ die tatsächliche (Ziel-)Wahrscheinlichkeit für Klasse j im Beispiel i ist,
- $q_j^{(i)}$ die vom Modell vorhergesagte Wahrscheinlichkeit für Klasse j im Beispiel i ist.

4.2.2.2 Gradientenabstiegsverfahren

Das Gradientenabstiegsverfahren basiert auf der Erkenntnis, dass der Gradient einer Funktion in Richtung des steilsten Anstiegs zeigt. Dieser Gradient lässt sich bei stetigen differenzierbaren Funktionen durch partielle Ableitung ermitteln. Folgt man dem Gradienten in kleinen Schritten in negativer Richtung (also in Richtung des steilsten Abstiegs), nähert man sich einem lokalen Minimum an.

Wendet man dieses Prinzip auf eine differenzierbare Verlustfunktion $J(\theta)$ an, kann man die Gradienten bezüglich der einzelnen Parameter $\theta_i \in \theta$ durch Differenzierung ermitteln. Diese Ableitung $\frac{\partial J(\theta)}{\partial \theta_i}$ gibt an, wie stark und in welche Richtung sich die Verlustfunktion $J(\theta)$ bei einer Änderung des Parameters θ_i verändert. Nun können alle Parameter simultan in kleinen Schritten entgegen der Richtung ihres jeweiligen Gradienten angepasst werden. Idealerweise werden die Parameter dadurch so angepasst, dass die Verlustfunktion schrittweise minimiert wird. Bei dieser schrittweisen Anpassung bestimmt die sogenannte Lernrate η die Größe des Schrittes, mit dem die Gewichte aktualisiert werden. Eine zu große Lernrate birgt die Gefahr, dass das (gesuchte) Minimum übersprungen wird oder der Trainingsprozess instabil wird. Eine zu kleine Lernrate kann dazu führen, dass der Trainingsprozess nur sehr langsam konvergiert.

Definition 25 (Optimierung mittels Gradientenabstieg) Bei der Optimierung eines NN erfolgt die Aktualisierung für jeden Parameter θ_i des NN wie folgt:

$$\theta_i \leftarrow \theta_i - \eta \cdot \frac{1}{|\hat{p}_{batch}|} \sum_{(x,y) \in \hat{p}_{batch}} \frac{\partial L(f^*(x, \theta), y)}{\partial \theta_i}$$

wobei

- η die Lernrate ist, welche die Schrittgröße der Aktualisierung bestimmt,
- $\hat{p}_{batch} \subseteq \hat{p}_{data}$ eine Teilmenge der Trainingsdaten (Batch) ist,
- L die verwendete Verlustfunktion bezeichnet.

Führt man einen solchen Trainingsschritt wie in Definition 25 nur mit einem einzigen Trainingsbeispiel $((x, y) \in \hat{p}_{data}$, also $|\hat{p}_{batch}| = 1$) aus, spricht man vom stochastischen Gradientenabstiegsverfahren (SGD). Entspricht der Batch der gesamten Trainingsmenge ($\hat{p}_{batch} = \hat{p}_{data}$), spricht man vom Batch-Gradientenabstiegsverfahren. Eine Mischung aus beiden Verfahren besteht darin, eine zufällige Teilmenge (Mini-Batch) fester Größe aus der Trainingsmenge zu wählen und die Trainingsschritte mit diesem Mini-Batch durchzuführen. Dies wird als Mini-Batch-Gradientenabstiegsverfahren bezeichnet. In der Regel wird das Mini-Batch-Gradientenabstiegsverfahren für das Training von NNs verwendet, da es meist einen guten Kompromiss zwischen Berechnungsaufwand und Stabilität der Konvergenz bietet und auf moderner Hardware effizient parallelisiert werden kann.

4.2.2.3 Backpropagation

Die Ermittlung der Gradienten der Parameter bezüglich der Verlustfunktion $\nabla_{\theta} J(\theta)$ erfolgt mittels des sogenannten Backpropagation-Algorithmus. Grundsätzlich ermöglicht dieser die effiziente Berechnung des Gradienten einer zusammengesetzten differenzierbaren Funktion, wie sie durch ein NN und die Verlustfunktion gegeben ist [RHW86]. Beim Gradientenabstiegsverfahren wird mithilfe der Backpropagation $\nabla_{\theta} J(\theta)$ bestimmt, also die Gradienten aller Parameter $\theta_i \in \theta$ bezüglich der Verlustfunktion $J(\theta)$. Dabei wird der Verlust für einen gesamten Batch \hat{p}_{batch} berücksichtigt. Im konkreten Fall unterteilt sich das Vorgehen in die folgenden Schritte:

- **Forward Propagation:** Für einen Batch von Eingaben $x \subseteq \hat{p}_{batch}$ wird das Modell $f^*(x, \theta)$ durchlaufen, um Vorhersagen \hat{y} zu erhalten. Während dieses Durchlaufs werden Zwischenergebnisse, wie die Ausgaben einzelner Schichten und Aktivierungen, gespeichert.
- **Verlustberechnung:** Nach der Forward Propagation wird mit den Vorhersagen \hat{y} und den Zieldaten y die Verlustfunktion $J(\theta)$ für diesen Batch berechnet, die einen skalaren Verlustwert liefert.

- **Backward Propagation:** Anschließend wird der Berechnungsgraph, der bei der Forward Propagation entstanden ist, in umgekehrter Reihenfolge durchlaufen. Ausgehend von der partiellen Ableitung des Verlustes bezüglich der Ausgabe $\frac{\partial J}{\partial \hat{y}}$ werden mithilfe der Kettenregel schrittweise die Ableitungen (Gradienten) für die Parameter θ aller vorhergehenden Schichten berechnet. Es ergibt sich schließlich $\nabla_{\theta} J(\theta)$, der gesuchte Gradient der Verlustfunktion bezüglich aller Modellparameter.

Die Berechnungen des Backpropagation-Algorithmus basieren im Kern auf der wiederholten Anwendung der Kettenregel der Differentialrechnung. Auf eine detailliertere Erläuterung soll an dieser Stelle verzichtet werden.

4.2.3 *Transformer-Architektur*

Im vorherigen Abschnitt wurde bereits die Grundstruktur des MLP erläutert und dessen Trainingsprozess allgemein beschrieben. Dieser Prozess ist nicht nur auf MLP beschränkt, sondern kann auf beliebige differenzierbare Netzwerkarchitekturen angewendet werden. Für eine Vielzahl von Anwendungen haben sich angepasste und meist deutlich komplexere Netzwerkarchitekturen durchgesetzt, welche die spezielle Struktur der zugrunde liegenden Eingabedaten besser verarbeiten können.

Ein zentrales Problem von MLP besteht darin, dass sie keine Annahmen über die Struktur der Eingabedaten machen. Man spricht auch davon, dass sie keinen induktiven Bias haben. In Aufgaben der Objekterkennung in Bildern kann man beispielsweise die Kenntnis, dass die Eingabedaten Bilder sind, nutzen: Da Objekte oft lokal zusammenhängende Pixelmuster aufweisen und an verschiedenen Positionen im Bild auftreten können, nutzen Convolutional Neural Networks (CNN) beispielsweise lernbare Faltungsfilter. Diese wenden dieselben Gewichte auf verschiedene lokale Bereiche eines Bildes an und können so effektiver lernen.

Die im Folgenden vorgestellte Transformer-Architektur wurde ursprünglich für die Verarbeitung von Textsequenzen, wie Textgenerierung und Übersetzung, entwickelt. Ihr induktiver Bias liegt in der Modellierung globaler Abhängigkeiten zwischen gleichartigen Eingabeelementen. Diese Eigenschaft macht sie geeignet für Aufgaben, bei denen strukturell gleichartige Eingaben, zum Beispiel Wortfolgen, verarbeitet werden. Im Kontext dieser Arbeit sind diese Eingaben Kartenkonstellationen im Spiel Doppelkopf, also zum Beispiel die auf dem Tisch liegenden Karten des Spiels, die Handkarten sowie die angesagten Vorbehalte [Vas+17].

Im Rahmen dieser Arbeit wird der sogenannte Self-Attention-Mechanismus ausschließlich in seiner unmaskierten Form eingesetzt und vorgestellt. Die maskierte Variante wird typischerweise in autoregressiven Textgenerierungsmodellen verwendet und stellt sicher, dass bei der Vorhersage eines Elements nur Informationen aus vorhergehenden Elementen berücksichtigt werden. Der folgende Abschnitt ist wie folgt aufgebaut: Zunächst werden die zentralen Komponenten der Transformer-

Architektur vorgestellt. Anschließend wird erläutert, wie diese Komponenten zusammenwirken.

4.2.3.1 *Embeddings*

Ein praktisches Problem beim Umgang mit NN ist die Kodierung der Eingabedaten, insbesondere von kategorischen Daten. Kategorische Daten repräsentieren diskrete Werte ohne inhärente numerische Ordnung (z. B. Wörter, Kartentypen). Für solche Daten werden meist sogenannte *Embeddings* verwendet. Im Fall eines Textgenerierungsmodells sind kategorische Daten beispielsweise Wörter. Im Kartenspiel können es die einzelnen Karten sein: $\diamondsuit_9, \heartsuit_{10}, \clubsuit A$ [JMo8].

Eine einfache Kodierung als Ordinalzahl (z. B. $\diamondsuit_9 = 1, \heartsuit_{10} = 2, \dots$) ist für kategorische Daten in der Regel ungeeignet, da sie eine künstliche numerische Ordnung annimmt, die nicht der Semantik der Daten entspricht. Das NN müsste in einem solchen Fall lernen, mit dieser unzutreffenden Ordnung umzugehen, was zu Fehlinterpretationen führen kann.

Bei der Kodierung mittels Embeddings wird jeder Ausprägung einer Kategorie (jedem diskreten Wert) ein d -dimensionaler Vektor (der Embedding-Vektor) zugeordnet, wobei d die Embedding-Dimension ist. Diese Vektoren werden im Rahmen des Trainingsprozesses gelernt und kodieren die Bedeutung der Kategorien im Kontext des spezifischen Problems. Ähnliche Kategorien (im Sinne ihrer Rolle für die Aufgabe) erhalten dabei tendenziell ähnliche Embedding-Vektoren.

Beispiel 10 (Embedding-Beispiel in Doppelkopf) In einem Experiment wurde ein NN darauf trainiert, für zwei Karten aus Doppelkopf zu ermitteln, welche Karte einen höheren Zählwert hat und welche Karte eine andere im Normalspiel sticht. Dabei wurden gemeinsame dreidimensionale Embeddings trainiert, die im Endergebnis wie folgt aussehen:

$$(\diamondsuit_9, \begin{bmatrix} -0,44 \\ -0,26 \\ -0,97 \end{bmatrix}), (\heartsuit_{10}, \begin{bmatrix} -1,08 \\ 0,37 \\ 2,42 \end{bmatrix}), \dots, (\clubsuit A, \begin{bmatrix} 2,33 \\ 0,06 \\ 2,45 \end{bmatrix})$$

Die dritte Komponente scheint beispielsweise den Kartenzählwert ($A = 11, 10 = 10, 9 = 0$) zu kodieren. Die anderen beiden Komponenten kodieren in irgendeiner Weise die Rangfolge innerhalb eines Stiches.

In einem NN übernimmt die sogenannte Embedding-Schicht die Aufgabe, kategorische Eingaben in kontinuierliche Repräsentationen zu überführen. Jeder Ausprägung einer Kategorie wird dabei ein eigener lernbarer Embedding-Vektor zugewiesen. Die Schicht wird durch zwei Parameter bestimmt: Die Anzahl der möglichen Kategorien (Wörterbuchgröße) und die Dimension der Embedding-Vektoren [PyE].

Die resultierenden Embedding-Vektoren dienen innerhalb des NN als Eingabe für die nachfolgenden Schichten. Ein zentraler Vorteil der Embedding-Schicht besteht darin, dass dieselben gelernten Gewichte (Embedding-Vektoren) für alle Vorkommen einer Kategorie verwendet werden – insbesondere auch dann, wenn eine Kategorie mehrfach innerhalb derselben Eingabe auftritt. In solchen Fällen wird der entsprechende Vektor mehrfach verwendet, aber es existiert nur ein einziger Parametersatz, der beim Backpropagation-Schritt konsistent aktualisiert wird. Der Optimierungsprozess muss somit eine gemeinsame Repräsentation finden, die über alle Vorkommen einer Kategorie hinweg sinnvoll ist und den erwarteten Verlust minimiert. Dadurch wird das Training effizienter: Eine Änderung der Embedding-Parameter für eine Kategorie wirkt sich automatisch auf sämtliche zukünftigen (und parallelen) Verwendungen dieser Kategorie aus [Mik+13]. Im Rahmen der Zustandskodierung im Kapitel 6.3.1 im Spiel Doppelkopf werden beispielsweise 62 Objekte (Handkarten, Tischkarten, Vorbehalte, An- und Absagen) mittels des gleichen Embedding-Layers kodiert.

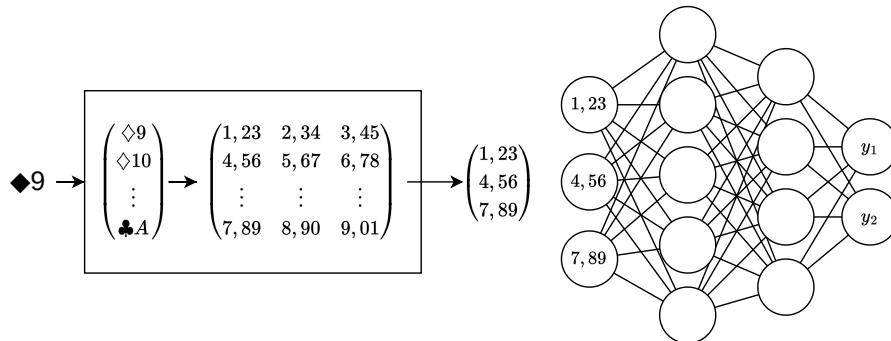


Abbildung 4.4: Embedding-Layer für Karten aus Doppelkopf.

4.2.3.2 Self-Attention und Multi-Head-Attention

Das zentrale Element der Transformer-Architektur ist der sogenannte Self-Attention-Mechanismus [Vas+17]. Dieser erlaubt dem Netzwerk, jedes Eingabeelement (repräsentiert durch seinen Embedding-Vektor) mit allen anderen Eingabeelementen in Kontext zu setzen und dabei deren Beziehungen und Abhängigkeiten explizit zu modellieren.

Am Anfang eines Self-Attention-Blocks werden die Embedding-Vektoren jedes Eingabeelements in drei unterschiedliche Repräsentationen mittels linearer Projektionen umgewandelt:

- **Query (Q):** Beschreibt, auf welche anderen Eingabeelemente das betrachtete Element seine Aufmerksamkeit richten soll.
- **Key (K):** Beschreibt, wie sich ein Element gegenüber den Anfragen der anderen Elemente präsentiert.

- **Value (V)**: Beschreibt den eigentlichen Informationsgehalt der Elemente.

Die Umwandlung erfolgt mittels lernbarer Gewichtsmatrizen W_Q, W_K, W_V und Bias-Terme. Lineare Projektionen sind vergleichbar mit einer einzelnen Schicht eines MLP ohne nichtlineare Aktivierungsfunktion.

Der Self-Attention-Mechanismus erzeugt für jedes Eingabeelement x_i eine neue Repräsentation z_i , die eine gewichtete Summe der Value-Vektoren aller Eingabeelemente ist. Die Gewichte α_{ij} werden anhand der Ähnlichkeit zwischen dem Query-Vektor q_i des aktuellen Elements und den Key-Vektoren k_j aller Elemente (einschließlich sich selbst) bestimmt. Je ähnlicher q_i und k_j sind, desto höher ist das Gewicht α_{ij} und desto stärker beeinflusst der zugehörige Value-Vektor v_j die Ausgabe z_i .

Die Ähnlichkeit zwischen Query- und Key-Vektoren wird über das Skalarprodukt berechnet. Für n Eingabeelemente werden somit n^2 Ähnlichkeitswerte berechnet. Um daraus eine Wahrscheinlichkeitsverteilung zu bilden, werden die Skalarprodukte skaliert und anschließend mittels der differenzierbaren Softmax (Softmax)-Funktion normalisiert. Die Softmax-Funktion wandelt dabei eine Menge beliebiger Zahlen in eine Verteilung mit Werten zwischen 0 und 1 um, deren Summe 1 bildet. Formal lässt sich der Self-Attention-Mechanismus wie folgt beschreiben:

$$\text{Attention}(Q, K, V) = \text{softmax} \left(\frac{QK^T}{\sqrt{d_k}} \right) V$$

wobei gilt:

- $Q \in \mathbb{R}^{n \times d_k}$ ist die Matrix aller Query-Vektoren der Eingabeelemente.
- $K \in \mathbb{R}^{n \times d_k}$ ist die Matrix aller Key-Vektoren der Eingabeelemente.
- $V \in \mathbb{R}^{n \times d_v}$ ist die Matrix aller Value-Vektoren der Eingabeelemente.
- d_k ist die Dimension der Key- und Queryvektoren.
- d_v ist die Dimension der Value-Vektoren.
- $\sqrt{d_k}$ ist der Skalierungsfaktor.

Von Multi-Head-Attention spricht man, wenn mehrere solcher Self-Attention-Operationen parallel in mehreren *Heads* durchgeführt werden. Dadurch wird die Kapazität des Netzwerks, unterschiedliche Arten von Beziehungen zu erlernen, erhöht. Dabei gibt es für jeden Head h unabhängige lernbare Parameter $W_Q^{(h)}, W_K^{(h)}, W_V^{(h)}$. Am Ende der parallelen Durchführungen werden alle Ergebnisvektoren konkateniert und mittels einer weiteren linearen Projektion mit den Gewichten W_O in eine gemeinsame Repräsentation überführt.

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \dots, \text{head}_h) W^O$$

wobei $head_1, head_2, \dots, head_h$ die Ausgaben der parallel ausgeführten Self-Attention-Blöcke sind und h die Anzahl der Heads.

Beispiel 11 (Self-Attention für Stichgewinner) In Abbildung 4.5 ist die Attention-Matrix eines trainierten Self-Attention-Layers dargestellt. Das Modell erhält als Eingabe zwei Stiche aus einem Doppelkopf-Spiel, also insgesamt acht Karten. Jede Karte ist beschrieben durch ihren Typ und die Position im Spiel (Stichnummer und Position innerhalb des Stiches). Ziel des Modells ist es, vorherzusagen, ob eine Karte ihren jeweiligen Stich gewinnt.

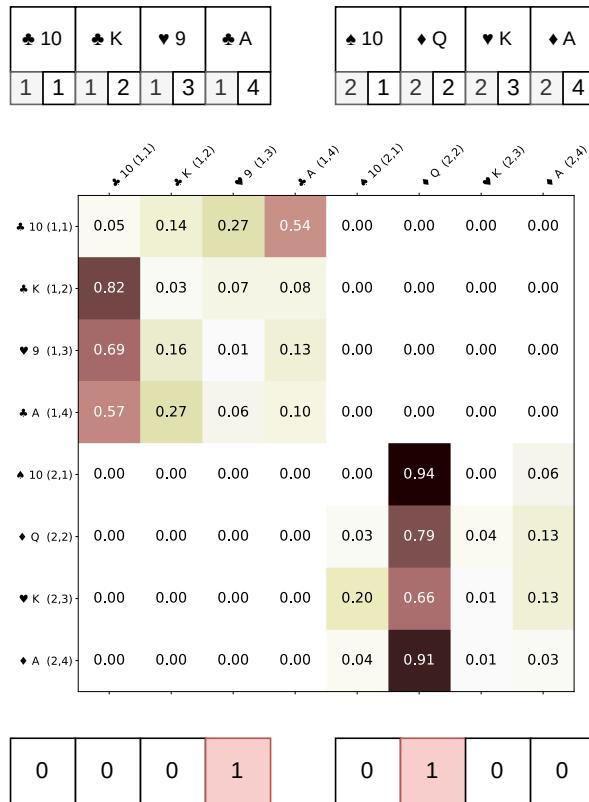


Abbildung 4.5: Self-Attention-Matrix bei der Ermittlung der Stichgewinner.

Im dargestellten Beispiel wird im ersten Stich Kreuz angespielt. Gewonnen hat ♣ A, da es die höchste Karte in der angespielten Farbe ist. Im zweiten Stich gewinnt ♦ Q, da sie ein hoher Trumpf ist und Trumpf jede Fehlfarbe sticht.

Die gezeigte Attention-Matrix gibt an, auf welche anderen Karten (Spalten) eine Karte (Zeile) während der Berechnung besonders achtet. Diese Gewichtung ergibt sich aus dem Zusammenspiel der Query- und Key-Vektoren. Folgende Beobachtungen lassen sich direkt aus der Matrix ablesen:

- **Stichtrennung:** Karten richten ihre Aufmerksamkeit ausschließlich auf andere Karten im selben Stich.
- **Fokus auf relevante Gegenspieler:** Die Aufmerksamkeit konzentriert sich innerhalb des Stiches auf Karten, die entweder höherwertig in der angespielten Farbe oder Trumpf sind.

- **Erkennung der angespielten Farbe:** Viele Karten richten ihre Aufmerksamkeit gezielt auf die erste Karte im Stich. Das Modell hat gelernt, dass die Stichfarbe von Bedeutung ist.

Stellt man sich die benötigten Query- und Key-Vektoren zu diesem Problem vor, könnte eine Abfrage umgangssprachlich wie folgt formulieren:

„Gibt es Trumpf in meinem Stich? Wenn ja: Welcher Trumpf ist am höchsten? Wenn nein: Welche Farbe wurde in meinem Stich angespielt und welche Karte ist in dieser Farbe die stärkste?“

Anhand der Attention-Matrix wird deutlich, dass das Modell durch die Optimierung der Gewichte der Query-, Key- und Value-Projektionen gelernt hat, zentrale Spielprinzipien wie Trumpfreihenfolge und Farbbedienung im hochdimensionalen Raum abzubilden.

4.2.3.3 Residuale Verbindungen

Beim Training tiefer NN tritt häufig das Problem der verschwindenden Gradienten auf. Dabei werden die Gradienten während der Backpropagation durch die vielen Schichten immer kleiner, sodass die Parameter in den vorderen Schichten (nahe der Eingabe) kaum noch aktualisiert werden. Ursache hierfür ist die Verkettung vieler Ableitungen der (nichtlinearen) Aktivierungsfunktionen im Rahmen der Backpropagation, deren Beträge meist kleiner als 1 sind. Dies führt dazu, dass die Gradienten auf ihrem Weg zu den vorderen Schichten immer stärker abgeschwächt werden [GBC16].

Eine Strategie zur Milderung dieses Problems stellen sogenannte Residualverbindungen (Skip Connections) dar. Diese ermöglichen, dass die Gradienten (die Veränderungen eines Parameters durch die partielle Ableitung) während des Rückwärtsdurchlaufs direkt und unverändert über mehrere Schichten hinweg zu den vorderen Schichten durchgeleitet werden können [He+16].

Formal lässt sich eine residuale Verbindung in der Funktionsdarstellung eines NN folgendermaßen ausdrücken:

$$f(x) + x$$

Dabei repräsentiert $f(x)$ die Ausgabe einer oder mehrerer aufeinanderfolgender Schichten des neuronalen Netzes und x deren Eingabe. Damit ändert sich das Lernziel der jeweiligen Schicht dahingehend, dass sie nicht mehr direkt die gesamte Ausgabe, sondern lediglich die Differenz zwischen Ein- und Ausgabe lernen muss. Praktisch bedeutet dies, dass eine Schicht, die zunächst wenig trainiert oder ineffektiv ist durch die Identitätsverbindung übersprungen werden kann. Es ergibt sich eine Verlustfunktion, die einfacher zu optimieren ist. Damit wird die Stabilität des Trainings verbessert [Li+18].

4.2.3.4 Layer Normalization

Beim Gradientenabstieg wird in einem Trainingsschritt jeweils ein Parameter unter der Annahme angepasst, dass alle übrigen Parameter in vorhergehen-

den und nachfolgenden Schichten unverändert bleiben. Praktisch werden jedoch alle Parameter gleichzeitig aktualisiert, was zu unerwarteten Wechselwirkungen zwischen den Schichten führen kann. Dadurch verändert sich die Verteilung der Ausgaben jeder Schicht kontinuierlich. Dieses Phänomen wird auch als *Internal Covariate Shift* bezeichnet [IS15]. Da die Ausgabe einer Schicht als Eingabe der nächsten dient, muss sich jede nachfolgende Schicht bei jedem Schritt an eine neue Verteilung der Eingabe anpassen. Dies verlangsamt das Training, macht es instabiler und erhöht das Risiko für Probleme wie verschwindende oder explodierende Gradienten [GBC16].

Das Problem kann durch Normalisierung der Ausgaben der einzelnen Schichten gemildert werden. Im von [Vas+17] vorgestellten Transformer-Modell wird dabei die sogenannte *Layer Normalization* verwendet [BKH16]. Dabei wird jeder Eingabevektor entlang seiner letzten Dimension so transformiert, dass die Einträge einen Mittelwert von 0 und eine Standardabweichung von 1 aufweisen.

Im Doppelkopf-Beispiel könnten die Eingaben in eine solche *LayerNorm*-Schicht beispielsweise die 62 im Spiel befindlichen Spielobjekte sein, die durch ein Embedding in einen kontinuierlichen Vektorraum überführt wurden. Jeder dieser Vektoren würde dann individuell normalisiert.

Zusätzlich werden bei der Layer Normalization zwei lernbare Parameter, γ (Skalierung) und β (Verschiebung), eingeführt, die elementweise nach der Normalisierung angewendet werden. Diese ermöglichen dem Netzwerk, die Normalisierung bei Bedarf rückgängig zu machen oder anzupassen, falls die ursprüngliche Skalierung und Verschiebung der Merkmale für das Training vorteilhaft sind [GBC16]. Formell ergibt sich für den j -ten Eintrag x_j eines Eingabevektors x die Transformation [PyL]:

$$y_j = \frac{x_j - \text{E}[x_j]}{\sqrt{\text{Var}[x] + \epsilon}} \cdot \gamma_j + \beta_j$$

Dabei sind γ_j und β_j erlernbare Parameter und werden für alle Eingabevektoren geteilt. $\text{Var}[x]$ ist die Varianz aller Einträge von x und $\text{E}[x]$ der Durchschnitt. ϵ ist ein Wert zur Vermeidung der Division durch 0.

4.2.3.5 Zusammenspiel der Komponenten

Alle beschriebenen Komponenten werden zu sogenannten Transformer-Blöcken zusammengefügt, wie sie in Abbildung 4.6 dargestellt sind. Ein solcher Block besteht aus zwei Hauptoperationen: Einer Multi-Head-Attention-Schicht sowie einem MLP. Ergänzt werden diese Hauptoperationen für die Trainingsstabilität um Residual-Verbindungen und LayerNorm-Schichten. Die in [Vas+17] vorgestellte Struktur ist dabei wie folgt aufgebaut:

1. **Multi-Head-Attention:** Verarbeitung der Eingabe durch mehrere parallele Self-Attention-Köpfe.
2. **Add & LayerNorm:** Residual-Verbindung zur Eingabe (Add), gefolgt von einer Normalisierung.

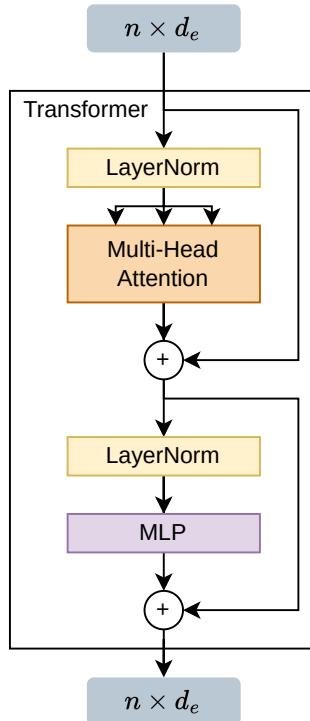


Abbildung 4.6: Ein einzelner Transformer-Block für Eingaben mit Embedding-Dimension d_e .

3. **MLP:** Transformation jedes einzelnen Vektors durch ein (typischerweise zweischichtiges) MLP.
4. **Add & LayerNorm:** Erneute Residual-Verbindung und Normalisierung.

Die Multi-Head-Attention-Schicht ermöglicht dem Modell, kontextabhängige Beziehungen zwischen allen Eingabeelementen zu erfassen. Das anschließende MLP wird für jeden Eingabevektor getrennt (jedoch mit den gleichen Gewichten) angewendet. Es dient dazu, die aus dem Attention-Mechanismus resultierenden Repräsentationen weiterzuverarbeiten und in eine für die folgende Schicht nützliche Repräsentation zu überführen. Die Residual-Verbindungen und die Layer-Normalisierung dienen der Stabilisierung des Trainings und ermöglichen den Bau sehr tiefer Transformer-Netzwerke.

Diese Transformer-Blöcke werden in der Regel mehrfach hintereinander geschaltet und bilden so oft tiefe Netzwerkstrukturen. Durch das Zusammenspiel mehrerer Blöcke kann das Modell zunehmend komplexere und abstraktere Zusammenhänge in den Eingabedaten erkennen. Frühere Schichten erfassen dabei tendenziell eher einfache Muster und lokale Beziehungen zwischen den Eingaben, während spätere Schichten darauf aufbauend abstraktere Muster und Konzepte lernen können [All+21] [BCV13]. Im Kontext des Doppelkopf-Spiels könnten die ersten Blöcke zum Beispiel erkennen, wer mit welcher Karte einen Stich gewonnen hat. Spätere Blöcke könnten diese Infor-

mation nutzen, um die Spielsituation strategisch zu bewerten und daraufhin eine (risikoreiche) Absage in Betracht zu ziehen.

4.3 ALPHAZERO

AlphaZero (AZ) ist ein allgemeiner RL-Algorithmus für Nullsummenspiele mit zwei Spielern, perfekter Information und ohne Zufallsereignisse. Er wurde von Silver et al. im Jahr 2017 in ihrer Veröffentlichung [Sil+17a] vorgestellt.¹ Der Algorithmus wurde in der Öffentlichkeit als bedeutender Fortschritt im Bereich der künstlichen Intelligenz aufgenommen, da er Meisterleistungen in den Spielen Schach, Go und Shogi vollbrachte.

AZ ist ein modellbasierter RL-Algorithmus, der zwei Konzepte kombiniert: Die Suche innerhalb des Spielbaums und Funktionsapproximation durch NN. Dadurch vereint AZ die Stärken des MCTS mit den Vorteilen des klassischen RL. Das Training erfolgt vollständig durch Selbstspiel gegen sich selbst.

AZ ist insofern allgemein, als er Entscheidungen trifft, ohne auf spieldspezifische Regeln oder Heuristiken zurückzugreifen. Nichtsdestotrotz hat AZ als modellbasierter RL-Algorithmus Zugriff auf den Spielbaum. AZ kann den Spielbaum (mittels eines Spielsimulators) während des Spiels durchsuchen, besitzt jedoch keine vorher explizit festgelegten spieldspezifischen Strategien oder Heuristiken.

Der Algorithmus verwendet Funktionsapproximation, welche in der Veröffentlichung von [Sil+17a] durch NN umgesetzt wird. Die Funktionsapproximation ermöglicht es, mittels bereits gelernten Zusammenhängen auf andere Spielzustände zu generalisieren (vgl. Abbildung 2.2.6). Das Training der verwendeten Funktionsapproximatoren erfolgt dabei durch sogenanntes Selbstspiel, also durch Spiele des Algorithmus gegen sich selbst. Alle zum Training verwendeten Spiele werden somit von AZ generiert.

AZ verwendet sowohl während des Trainings als auch bei der eigentlichen Ausführung eine Suchmethode im Spielbaum, die an den zuvor vorgestellten MCTS angelehnt ist. Obwohl die Autoren in [Sil+17a] den Begriff „MCTS“ verwenden, handelt es sich nicht um den MCTS im Wortsinne, da bei AZ gerade keine zufälligen Monte-Carlo-Simulationen durchgeführt werden. Stattdessen werden die Simulationen durch Funktionsapproximationen ersetzt. Da sich der Begriff des MCTS in diesem Kontext etabliert hat, soll er auch in den folgenden Ausführungen verwendet werden.

Die Funktionsweise von AZ wird in den folgenden Schritten beschrieben:

- Zunächst werden die verwendeten Funktionsapproximatoren sowie deren konkrete Aufgaben beschrieben.
- Anschließend wird die abgewandelte Variante des MCTS vorgestellt und herausgestellt, in welchen Aspekten sie sich vom ursprünglichen Verfahren unterscheidet.

¹ Die Arbeit beruhte auf den vorherigen Veröffentlichungen [Sil+16] und [Sil+17b].

- Im dritten Schritt wird der Trainings- und Evaluationsprozess skizziert. Dabei wird durch einen Blick auf die gesamte Funktionsweise des Algorithmus verdeutlicht, wie die zuvor beschriebenen Komponenten ineinander greifen.

4.3.1 *Funktionsapproximationen*

AZ verwendet zwei Funktionsapproximatoren.

4.3.1.1 *Wertfunktion*

Zum einen approximiert der Algorithmus eine Funktion, die einem gegebenen Spielzustand einen Wert zuordnet, welcher dessen durchschnittlichen Nutzen widerspiegelt.

Die Wertfunktion $v(s)$ bewertet für einen gegebenen Spielzustand s dessen erwarteten zukünftigen Nutzen aus Sicht eines Spielers. Dabei wird angenommen, dass beide Spieler ab diesem Zustand nach der aktuellen Strategie π spielen und versuchen, ihren eigenen Nutzen zu maximieren. Die Wertfunktion gibt somit die Güte eines Spielzustands an und ersetzt damit die zufälligen MC-Simulationen, die im klassischen MCTS zur Evaluierung der Spielzustände durchgeführt werden.

4.3.1.2 *Strategiefunktion*

Die Strategiefunktion $p(s)$ ermittelt für einen gegebenen Spielzustand s eine Wahrscheinlichkeitsverteilung über alle in diesem Zustand ausführbaren Aktionen.

Innerhalb des MCTS wird diese Wahrscheinlichkeitsverteilung verwendet, um die Suche innerhalb eines Teilbaums zu lenken. AZ modifiziert hierfür UCT zur sogenannten Predictor + UCT (PUCT), bei der die Vorhersagen der Strategiefunktion berücksichtigt werden.

Im Verlauf des Trainings approximiert die Strategiefunktion zunehmend die Ergebnisse eines vollständigen MCTS-Durchlaufs selbst. Dadurch wird es dem Algorithmus möglich, vielversprechende Aktionen zu identifizieren, ohne für jeden einzelnen Spielzustand umfangreiche Suchbäume aufzubauen.

4.3.1.3 *Architektur des KNN*

Die Funktionsapproximationen werden in [Sil+17a] durch NN umgesetzt. Prinzipiell können auch andere Methoden genutzt werden. Die Wahl der konkreten Architektur des NN ist stark vom untersuchten Spiel und dessen Komplexität abhängig. Auf die genaue Architektur von [Sil+17a] für die Spiele Shogi, Schach und Go soll daher nicht eingegangen werden.

Auf einen Aspekt der Architektur von [Sil+17a] soll jedoch eingegangen werden: Beide Funktionsapproximationen werden dort durch ein gemeinsames NN umgesetzt, das erst in seinen letzten Schichten in zwei eigenständi-

ge Köpfe verzweigt (engl.: *heads*). Die beiden Köpfe approximieren dann die Wertfunktion $v(s)$ beziehungsweise die Strategiefunktion $p(s)$.

Diese gemeinsame Nutzung von Netzwerkparametern hat verschiedene Vorteile: Zum einen reduziert sie die Zahl der zu trainierenden Parameter und ermöglicht ein schnelleres und effizienteres Training. Zum anderen verbessert sich in der Regel die Generalisierungsfähigkeit, da die in den gemeinsamen Schichten gelernten Repräsentationen für beide Aufgaben verwendet werden.

Im Folgenden wird diese gemeinsame Funktionsapproximation durch ein neuronales Netz mit Parametervektor θ dargestellt als

$$f_{\theta}^*(s) = (\vec{p}, v).$$

Der Trainingsprozess erfolgt mittels des zuvor vorgestellten Gradientenabstiegsverfahrens. Dabei wird für den Kopf der Wertfunktion $v(s)$ eine MSE-Verlustfunktion verwendet, während der Kopf der Strategiefunktion $p(s)$ mit Hilfe der CE-Verlustfunktion optimiert wird. Da beide Köpfe im selben Trainingsschritt durch Backpropagation angepasst werden, ergibt sich die gesamte Verlustfunktion als Summe der beiden einzelnen Verlustterme.

4.3.2 MCTS

Sowohl im Rahmen des Trainings als auch im Rahmen der Evaluation wird bei AZ eine abgewandelte Form des MCTS ausgeführt. Auch hier dient er dazu, für einen Spielzustand s die beste Aktion zu wählen. Dabei ergeben sich drei zentrale Unterschiede:

4.3.2.1 Wertfunktion statt Simulation

Der Nutzen eines Zustands wird beim AZ-MCTS (AZ-MCTS) nicht durch zufällige MC-Simulationen, sondern durch Aufrufe der Wertefunktion ermittelt. Hierbei ist anzumerken, dass die Wertefunktion lediglich einzelne Simulationsergebnisse ersetzt, nicht jedoch die gesamte Berechnung von $Q(s, a)$. Dieser ergibt sich weiterhin aus der Aggregation von Zustandsbewertungen innerhalb des Teilbaums von a .

4.3.2.2 Priore Wahrscheinlichkeiten und PUCT

Als Selektionsstrategie wird in AZ Predictor + UCT (PUCT) verwendet:

$$puct(s, a) = Q(s, a) + c_{puct} P(s, a) \frac{\sqrt{N(s)}}{1 + N(s, a)}$$

wobei gilt:

- $N(s)$ und $N(s, a)$ sind die Anzahl der Besuche der Knoten beziehungsweise der Aktionen in den vorigen Iterationen.

- $Q(s, a)$ ist der durchschnittliche Nutzen der Aktion a im Zustand s .
- $P(s, a)$ beschreibt die prioren Wahrscheinlichkeiten der Aktion a im Zustand s , welche durch die Strategiefunktion ermittelt werden. Beim Wurzelknoten s_{root} wird in der Trainingsphase zusätzlich ein Dirichlet-Rauschen hinzugefügt, um die Exploration zu fördern.
- c_{puct} ist eine Konstante, die den Explorationsgrad steuert.

PUCT kann wieder in einzelne Terme zerlegt werden, welche die Exploration beziehungsweise die Exploitation steuern.

- $Q(s, a)$ steuert wie im klassischen MCTS die Exploitation.
- $\frac{\sqrt{N(s)}}{1+N(s,a)}$ reduziert die Wahrscheinlichkeit häufig besuchter Aktionen und begünstigt die Exploration weniger erforschter Aktionen.
- Der Term $c_{puct}P(s, a)$ integriert nun die prioren Wahrscheinlichkeiten. Der Term ist spezifisch für AZ und führt dazu, dass Aktionen bevorzugt werden, die von der Strategiefunktion als besonders vielversprechend eingestuft werden.

Die prioren Wahrscheinlichkeiten werden aus der Strategiefunktion mittels der Softmax-Funktion (vgl. Abschnitt 4.2.3.2) abgeleitet. Dies sorgt dafür, dass alle Aktionen, die von einem Knoten ausgehen, eine Wahrscheinlichkeitsverteilung bilden, deren Summe 1 beträgt.

Da die Strategiefunktion selbst eine Approximation der Besuchshäufigkeiten der Aktionen aus vorgehenden MCTS-Durchläufen darstellt, entsteht eine Art approximativer MCTS. Aktionen, die ein vorheriger MCTS als besonders vielversprechend identifiziert hat, werden so bei der Exploration *jedes* Zustands berücksichtigt.

4.3.2.3 Aktionsauswahl

Die Aktionsauswahl ist analog zum regulären MCTS. Es wird die Aktion ausgewählt, welche vom Wurzelzustand aus den höchsten durchschnittlichen Nutzen $Q(s, a)$ oder die meisten Besuche $N(s, a)$ hatte. Es ergibt sich die Strategie:

$$\pi^{\text{AZ}}(\theta)$$

4.3.3 Ausführung des AlphaZero

Die Ausführung (Evaluation) des trainierten AZ-Modells mit trainierten Parametern θ erfolgt durch die Ausführung des AZ-MCTS für einen Spielzustand s . In der Evaluationsphase wird auf das Hinzufügen des Dirichlet-Rauschens in der Regel verzichtet.

4.3.4 Training der Funktionsapproximatoren

Das Training des AZ erfordert, die Parameter θ der zugrundeliegenden Funktionsapproximatoren mithilfe von Beispielen so zu trainieren, dass sie sinnvolle Werte zurückgeben, die zu einer starken Strategie führen.

Das Training von AZ besteht aus einer konfigurierbaren Anzahl von Trainingsiterationen. Eine solche Trainingsiteration (nicht zu verwechseln mit den Iterationen des MCTS) besteht aus der Selbstspielphase, bei der das Modell gegen sich selbst spielt, sowie einer Trainingsphase, bei der die Funktionsapproximatoren mit den gesammelten Erfahrungen trainiert werden.

4.3.4.1 Spiel gegen sich selbst

Im Rahmen der Selbstspielphase werden zahlreiche Spiele simuliert. Dazu wird ein neues Spiel initialisiert und in jedem Spielzustand bestimmt das AZ-MCTS mit den Parametern der letzten Trainingsiteration des NN θ_{i-1} die Aktion des aktuellen Spielers.

Dieser Prozess wird für alle Spielzüge wiederholt, bis das Spiel beendet ist. Für jeden Zug wird ein Erfahrungstupel gespeichert, das später für das Training verwendet wird:

$$(s, \vec{p}, v)$$

wobei gilt:

- s ist der Spielzustand, in dem der AZ-MCTS ausgeführt wurde.
- \vec{p} gibt die Wahrscheinlichkeitsverteilung der Besuchshäufigkeiten $N(s, a)$ über die im Zustand s zulässigen Aktionen $A(s)$ an, bestimmt mittels Softmax-Funktion.
- v gibt das tatsächliche Spielergebnis des vollständigen Spiels an.

Da die alte Parametrisierung θ_{i-1} verwendet wird, dominiert die zuvor erlernte Strategie $\pi^{AZ}(\theta)$ die gespielten Partien. Dies führt zu einem selbstverstärkenden Mechanismus: Das Modell nutzt zwar die beste bekannte Strategie aus, entdeckt jedoch keine neuen Spielweisen. Das Problem des Ausgleichs zwischen Exploration und Exploitation tritt somit nicht nur innerhalb des MCTS auf, sondern auch in der Trainingsschleife selbst.

Um diesem Effekt entgegenzuwirken, werden Mechanismen zur Förderung der Exploration eingesetzt. Zum einen kann die Exploration innerhalb des MCTS selbst durch die Anpassung des c_{puct} gefördert werden. Zum anderen werden die folgenden Mechanismen angewandt:

DIRICHLET-RAUSCHEN Innerhalb des AZ-MCTS wird den prioren Wahrscheinlichkeiten des Wurzelknotens ein Dirichlet-Rauschen hinzugefügt. Dieses stellt sicher, dass auch Aktionen, die vom aktuellen Modell θ^{i-1} nicht als vielversprechend angesehen werden, näher untersucht werden.

Die Dirichlet-Verteilung $\text{Dir}(\alpha)$ bewirkt, dass zufällig einige Aktionen stark gegenüber anderen bevorzugt werden. Ein kleiner Wert für α führt zu einer starken Fokussierung auf wenige zufällige Aktionen, während ein α von 1 das Rauschen gleichmäßig verteilt (vgl.: A.1). Ein Hyperparameter ϵ steuert den Einfluss der Dirichlet-Verteilung auf die prioren Wahrscheinlichkeiten:

$$(1 - \epsilon) \cdot \vec{p}_{root} + \epsilon \cdot \text{Dir}(\underbrace{\alpha, \alpha, \dots, \alpha}_{n \text{ mal}})$$

wobei n die Anzahl der im Zustand s_{root} möglichen Aktionen ist.

STOCHASTISCHE STRATEGIE Während der Selbstspielphase verwendet AZ eine stochastische Strategie, welche die Besuchshäufigkeiten eines MCTS-Durchlaufs mittels Softmax-Funktion mit dem Temperatur-Parameter τ als Wahrscheinlichkeitsverteilung interpretiert. Somit werden innerhalb der Selbstspielphase zufällig (jedoch proportional zur Qualität gemäß den Besuchshäufigkeiten) auch alternative Aktionen gewählt, die der MCTS selbst nicht priorisiert hätte. Dies ermöglicht die Erkundung neuer Spielpfade.

4.3.4.2 Trainingsphase

Sind in der Selbstspielphase ausreichend Spiele durchgeführt worden, werden die gewonnenen Erfahrungs-Tupel für das Training verwendet. Dabei wird die Strategiefunktion für den Spielzustand s anhand der Verteilung der Besuchshäufigkeiten \vec{p} trainiert. Der Wertekopf wird entsprechend dem Spielaustrag v angepasst.

Aufeinanderfolgende Erfahrungstupel sind stark korreliert, da sie aus demselben Spiel stammen. Für ein erfolgreiches Training eines NN sollten die Daten jedoch unabhängig und identisch verteilt sein. Um die negativen Effekte der Korrelation zu reduzieren, werden die Tupel aus verschiedenen Spielen vor dem Training gemischt. Im Idealfall würde man nur ein Erfahrungstupel pro Spiel zum Training verwenden. Dies wäre aufgrund des hohen Rechenaufwands jedoch unpraktikabel.

Dies führt zu einer neuen Parametrisierung θ_t . Mit dieser wird die Selbstspielphase erneut ausgeführt, sofern noch Trainingsiterationen ausstehen. Andernfalls ist die Parametrisierung final und kann für die Evaluation des Modells verwendet werden.

4.3.5 Funktionsweise

Der AZ ersetzt einzelne MC-Simulationen im klassischen MCTS durch die Wertefunktion. Diese wird auf Grundlage tatsächlicher Spielergebnisse trainiert, die mit der bislang besten Strategie erzielt wurden. Idealerweise ist sie dadurch deutlich präziser als MC-Simulationen, insbesondere in frühen Spielphasen, wenn noch viele Züge bis zum Ende verbleiben. Zusätzlich ermöglicht die Strategiefunktion eine gezieltere Exploration innerhalb der Teilbäu-

me, indem sie für jeden Zustand im Selektionsteilbaum gewissermaßen einen approximativen MCTS-Durchlauf simuliert.

Das Training dieser Funktionsapproximatoren, die AZ seine Stärke verleihen, erfolgt durch das wiederholte Ausführen des AZ-MCTS mit der jeweils aktuellsten Parametrisierung im Selbstspiel. Um diverse Spielverläufe für das Training zu sammeln, wird im Selbstspiel selbst ein eigener Ausgleich zwischen Exploration und Exploitation vorgenommen.

Durch die präzisere Evaluation der Spielzustände und Aktionen benötigt AZ, insbesondere bei komplexen Spielen, deutlich weniger Iterationen als der klassische MCTS. Diese Eigenschaft und die Generalisierung auf Zustände, die AZ vorher nie besucht hat, machen ihn auf komplexe Spiele anwendbar, bei denen MCTS versagt.

4.3.6 Anpassung für mehrere Spieler

AZ in der vorgestellten Form ist ein Algorithmus für Nullsummenspiele mit zwei Spielern. Das liegt im Wesentlichen an der Wertefunktion: Diese bewertet einen Zustand für beide Spieler gleichzeitig. Für einen Spieler ist der positive Wert maßgebend, während aufgrund der Nullsummen-Eigenschaft der Nutzen des Gegners der negierte Wert ist. In Spielen mit mehr als zwei Spielern und Spielen, die keine Nullsummenspiele sind, gilt diese Äquivalenz nicht mehr.

[PB19] machen AZ auch für Spiele mit mehreren Spielern und Nicht-Nullsummenspielen anwendbar: Anstelle eines einzelnen Wertes v gibt die Wertefunktion dann einen Vektor \vec{v} mit n Elementen aus, wobei n der Anzahl der Spieler entspricht. Der MCTS berücksichtigt nun immer die vorhergesagte Punktzahl des Spielers, der an der Reihe ist.

4.4 MCTS UND ALPHAZERO IN SPIELEN MIT IMPERFEKTER INFORMATION

In ihrer vorgestellten Form sind sowohl MCTS als auch AZ auf Spiele mit perfekter Information und ohne Zufallsereignisse beschränkt. Um diese Algorithmen auf Spiele mit imperfekter Information wie Doppelkopf anwenden zu können, sind Modifikationen notwendig.

Das grundlegende Problem liegt darin, dass die Baumsuche des MCTS davon ausgeht, dass ausgehend von einem eindeutig definierten Zustand eine eindeutige Simulation des weiteren Spielverlaufs möglich ist. Bei Spielen mit imperfekter Information liegt jedoch kein eindeutiger Zustand, sondern nur eine Informationsmenge $(U, A(U))$ beziehungsweise eine Beobachtung o vor, für die eine optimale Aktion $a \in A(U)$ bestimmt werden soll. Von einer solchen Informationsmenge (alleine) aus kann der zukünftige Spielverlauf nicht eindeutig (deterministisch) simuliert werden. Genau das ist jedoch die Voraussetzung für die Anwendung des MCTS.

Um die Grundprinzipien des MCTS und des AZ auch für Spiele mit imperfekter Information anwendbar zu machen, gibt es im Wesentlichen zwei Vorgehensweisen [Świ+23]:

- **Determinisierung:** Bei der Determinisierung werden aus der Informationsmenge ein oder mehrere Zustände ausgewählt. Auf diesen Zuständen wird dann der nahezu unveränderte MCTS oder AZ ausgeführt. Für jeden ausgewählten Zustand entsteht ein eigener Suchbaum. Der Ansatz wird häufig auch als Perfect Information Monte Carlo Tree Search (PI-MCTS) bezeichnet.
- **Informationsmengenbetrachtung:** Bei anderen Ansätzen werden MCTS oder AZ direkt so modifiziert, dass die Knoten Informationsmengen statt einzelner Zustände repräsentieren. Infolgedessen entsteht nur ein einziger Suchbaum.

Auch wenn die Informationsmengenbetrachtung einige Probleme des Determinisierungsansatzes vermeiden kann und mehr Effizienz verspricht, wird in dieser Arbeit der Ansatz der Determinisierung verfolgt. Der Determinisierungsansatz ist einfacher umzusetzen und erlaubt es, bestehende Erkenntnisse aus Literatur und Praxis zu MCTS und AZ direkter zu nutzen. Ein Ansatz auf Basis der Informationsmengenbetrachtung hingegen erfordert umfangreiche Anpassungen der bestehenden Algorithmen.

4.4.1 *Determinisierung*

Bei der Determinisierung werden vor der Ausführung des (unveränderten) MCTS oder AZ aus der Informationsmenge ein oder mehrere vollständige Spielzustände ausgewählt. Anschließend wird für jeden dieser Spielzustände das MCTS beziehungsweise AZ separat ausgeführt. Aus den jeweiligen Einzelergebnissen wird anschließend eine gemeinsame Entscheidung abgeleitet (siehe Abbildung 4.7). Daraus ergeben sich drei zentrale Problemfelder der Determinisierung [CPW12].

- **Auswahl der Zustände:** Aus der vorliegenden Informationsmenge müssen ein oder mehrere konkrete Zustände ausgewählt werden.
- **Einzelzustandsanalyse:** Durchführung des MCTS bzw. AZ auf jedem ausgewählten Zustand.
- **Strategie-Fusion:** Aus den Einzelergebnissen muss eine einzelne Entscheidung abgeleitet werden.

4.4.1.1 *Auswahl der Zustände*

Die Auswahl eines oder mehrerer Zustände aus einer Informationsmenge ist nicht ohne Weiteres möglich. Die Zustände, die eine Informationsmenge ausmachen, sind direkt nicht bekannt und aufgrund der großen Menge auch nicht aufzählbar. Vielmehr müssen mithilfe eines geeigneten Algorithmus aus einer Informationsmenge beziehungsweise einer Beobachtung die Spielzustände rekonstruiert werden. Dabei gibt es zwei wichtige Eigenschaften:

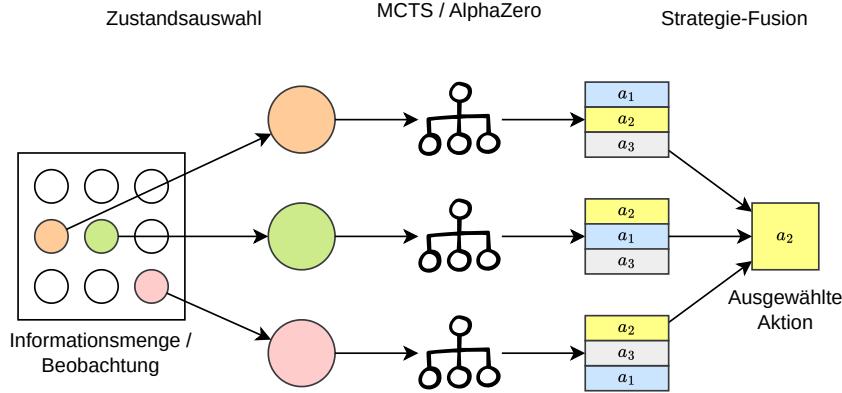


Abbildung 4.7: Schematische Darstellung der Determinisierung.

KONSISTENZ Der Prozess der Rekonstruktion eines Zustands aus einer Informationsmenge ist spielspezifisch. Der Zustand muss so rekonstruiert werden, dass er nach den geltenden Spielregeln und dem Spielverlauf auch tatsächlich vorkommen kann. Diese Eigenschaft eines Algorithmus soll im Folgenden als *Konsistenz* der erzeugten Zustände bezeichnet werden.

GLAUBENSVERTEILUNG Jeder Zustand innerhalb einer Informationsmenge tritt mit einer jeweils unterschiedlichen Wahrscheinlichkeit auf. Diese Wahrscheinlichkeiten ergeben sich sowohl aus den geltenden Spielregeln als auch aus dem Verhalten der anderen Spieler (siehe Beispiel 12). Ein Algorithmus, der diese Wahrscheinlichkeiten berücksichtigt, zieht Stichproben entsprechend der *Glaubensverteilung*.

Beispiel 12 Sagt ein rationaler Spieler beim Doppelkopf ein J-Solo als Vorbehalt an, so ist es wahrscheinlich, dass er mehrere Buben auf der Hand hält. Ein rational handelnder Spieler würde kein J-Solo mit nur wenigen Buben ansagen. Die Glaubensverteilung hängt somit sowohl vom bisherigen Spielverlauf (ein Spieler kündigt ein J-Solo an) als auch von der angenommenen Strategie (der Spieler agiert rational) ab. Gegenüber einem zufällig agierenden Spieler ließe sich eine solche Schlussfolgerung hingegen nicht ziehen.

In einfachen Worten beschreibt die Glaubensverteilung, dass bestimmte Zustände aufgrund des bisherigen Spielverlaufs und der von den Spielern verfolgten Strategien σ wahrscheinlicher sind als andere. Formell kann die Glaubensverteilung wie folgt definiert werden [CPW12]:

Definition 26 (Glaubensverteilung) Sei $(U, A(U))$ eine Informationsmenge eines Spielers i in einem Spiel Γ und σ die Strategien der Spieler. Die Glaubensverteilung $P(x | U_i^j, \sigma)$ ist eine Funktion, die jedem Zustand $x \in U$ eine Wahrscheinlichkeit zuordnet und die folgenden Bedingungen erfüllt:

1. $P(x | U, \sigma) \geq 0$ für alle $x \in U_i^j$,
2. $\sum_{x \in U} P(x | U, \sigma) = 1$.

4.4.1.2 Strategie-Fusion

Wurden für mehrere mögliche Zustände die optimalen Aktionen mittels MCTS beziehungsweise AZ ermittelt, ist daraus die tatsächlich auszuführende Aktion zu bestimmen. Dazu existiert prinzipiell eine Vielzahl möglicher Methoden. Diese können nicht nur die resultierenden Aktionen der MCTS-Durchläufe einbeziehen, sondern auch die im entstehenden MCTS-Baum gespeicherten Statistiken berücksichtigen. Häufig dienen dabei insbesondere die Anzahl der Besuche $N(s, a)$ oder der durchschnittliche Nutzen $Q(s, a)$ der Aktionen im Wurzelzustand als Entscheidungsgrundlage.

4.4.2 Behandlung von Zufallsereignissen

Spiele mit imperfekter Information sind meist auch Spiele mit Zufallsereignissen. Sowohl der vorgestellte MCTS als auch AZ können in ihrer Grundform nicht mit beliebigen Zufallsereignissen umgehen. Es existieren jedoch Anpassungen, um die Algorithmen auch auf solche Spiele anwenden zu können.

4.4.2.1 Zufallsereignisse im MCTS

In den einzelnen Simulationen ist die Berücksichtigung von Zufallsereignissen unkompliziert: Ein Nachfolgezustand wird entsprechend der Wahrscheinlichkeitsverteilung gewählt. Durch ausreichend viele Simulationen ergibt sich schließlich ein realistischer Mittelwert für den Nutzen $Q(s, a)$ der betroffenen Spielzustände.

Im Selektionsteilbaum wird ein Zustand, in dem ein Zufallsereignis eintritt, als eigener Knoten behandelt. Wird dieser im Rahmen der Selektionsphase erreicht, so wird der Nachfolger nicht anhand des höchsten UCT-Werts, sondern ausschließlich gemäß der zugrunde liegenden Wahrscheinlichkeitsverteilung ausgewählt. Diese Anpassung des MCTS zur Behandlung von Zufallsereignissen wird beispielsweise im quelloffenen OpenSpiel-Framework [Lan+19] erfolgreich eingesetzt.

4.4.2.2 Zufallsereignisse im AZ

Prinzipiell ist dieser einfache Ansatz des vorhergehenden Abschnitts auch auf den AZ-Algorithmus anwendbar. Ob der Ansatz wirklich funktioniert, ist in Literatur und Praxis jedoch kaum untersucht.

4.4.2.3 Zufallsereignis im Wurzelknoten

Tritt in einem Spiel lediglich ein einziges Zufallsereignis auf und findet dieses ausschließlich zu Beginn des Spiels statt, ist keine Anpassung der Algo-

rithmen erforderlich. Diese Eigenschaft trifft auf eine Reihe von Spielen zu, beispielsweise auf Kartenspiele, bei denen der Kartenstapel zu Spielbeginn gemischt wird und im weiteren Verlauf keine Zufallsereignisse mehr auftreten. In solchen Fällen entstehen im Rahmen des MCTS keine Zufallsknoten innerhalb der Suchbäume.

4.5 GENERATIVE MODELLE

Die Zustandsauswahl soll in dieser Arbeit auf einer Methode beruhen, welche die Glaubensverteilung der Zustände einer Informationsmenge explizit berücksichtigt. Hierfür soll ein sogenanntes generatives Modell eingesetzt werden, um aus einer gegebenen Informationsmenge sowohl konsistente als auch wahrscheinliche Zustände für das Spiel Doppelkopf zu erzeugen. Generative Modelle sind Machine Learning (ML)-Modelle, deren Ziel es ist, eine Wahrscheinlichkeitsverteilung $p(x)$ aus gegebenen Daten zu approximieren und daraus neue, bisher ungesehene Stichproben aus dieser Datenmenge zu generieren.

4.5.1 Klassische Statistik

In der klassischen Statistik werden generative Modelle häufig durch die Abgrenzung zu diskriminativen Modellen definiert. Ein diskriminatives Modell approximiert die bedingte Wahrscheinlichkeit $p(y|x)$, also die Wahrscheinlichkeit, dass eine Beobachtung x zu einer bestimmten Klasse y gehört. So mit lernt es primär, Klassen voneinander zu unterscheiden. Ein generatives statistisches Modell hingegen geht einen Schritt weiter. Es approximiert die gemeinsame Wahrscheinlichkeitsverteilung $p(x,y)$ von Daten x und Klassen y . Daher kann es nicht nur zur Klassifikation dienen, sondern ist auch fähig, neue Datenpunkte zu generieren, die aus der gelernten Verteilung stammen könnten [Mur22].

4.5.2 Unbedingte generative Modellierung

In der modernen Praxis des maschinellen Lernens werden generative Modelle oft etwas allgemeiner gefasst. Der Kernaspekt bleibt jedoch die Erlernung einer Wahrscheinlichkeitsverteilung aus Trainingsdaten.

Definition 27 (Unbedingtes generatives Modell) Ein unbedingtes generatives Modell approximiert eine unbekannte Wahrscheinlichkeitsverteilung $p_{real}(x)$:

$$p_\theta(x) \approx p_{real}(x), \text{ für } x \in X,$$

wobei

- X die Menge aller möglichen Datenpunkte ist,
- p_{real} die tatsächliche, jedoch unbekannte Datenverteilung ist,

- $p_\theta(x)$ die durch das Modell mit Parametern θ approximierte Verteilung ist.

Das Modell erlernt diese Approximation anhand einer Menge von Trainingsbeispielen $D_{\text{train}} \subset X$, welche als Stichproben aus der unbekannten Verteilung p_{real} betrachtet werden. Ein gutes generatives Modell zeichnet sich dadurch aus, dass es Stichproben generiert, die der realen Verteilung entsprechen, aber über eine reine Reproduktion der Trainingsdaten hinausgehen [Fos23].

4.5.3 Bedingte generative Modellierung

Darüber hinaus existieren bedingte generative Modelle, die dem oben genannten klassischen statistischen Begriff näherkommen. Diese Modelle approximieren eine bedingte Wahrscheinlichkeitsverteilung [Fos23].

Definition 28 (Bedingtes generatives Modell) Ein bedingtes generatives Modell approximiert die bedingte Wahrscheinlichkeitsverteilung $p_{\text{real}}(x|c)$:

$$p_\theta(x|c) \approx p_{\text{real}}(x|c), \quad \text{für } x \in X, c \in C,$$

wobei

- X die Menge aller möglichen Datenpunkte ist,
- C die Menge aller möglichen Bedingungen ist,
- $p_{\text{real}}(x|c)$ die tatsächliche, jedoch unbekannte bedingte Verteilung der Daten gegeben die Bedingung c ist.

4.5.4 Autoregressive Modelle

Für die Realisierung generativer Modelle existieren vielfältige Ansätze. Ihre Eignung hängt stark vom jeweiligen Anwendungszweck ab. Für die Textgenerierung kommen meistens autoregressive Modelle zum Einsatz. Die Zustandswahl aus einer Informationsmenge in Doppelkopf kann man als das Verteilen der noch verfügbaren Karten an die Mitspieler auffassen. Dieses Problem hat gewisse strukturelle Ähnlichkeiten zur Textgenerierung. Aus diesem Grund wird in dieser Arbeit ein autoregressiver Ansatz verfolgt.

Autoregressive Modelle basieren auf der Kettenregel der Wahrscheinlichkeit [Tom24]. Diese erlaubt es, die gemeinsame Wahrscheinlichkeit einer Sequenz von Variablen als ein Produkt von bedingten Wahrscheinlichkeiten darzustellen. Auf dieser Grundlage lässt sich ein autoregressives Modell wie folgt definieren:

Definition 29 (Autoregressives Modell) Ein autoregressives Modell approximiert eine Wahrscheinlichkeitsverteilung $p(x)$. Es zerlegt diese gemäß der Kettenregel der Wahrscheinlichkeit in ein Produkt bedingter Wahrscheinlichkeiten:

$$p(x) = \prod_{i=1}^n p(x_i | x_1, x_2, \dots, x_{i-1})$$

wobei

- $x = (x_1, x_2, \dots, x_n)$ ein Datenpunkt ist, der in n aufeinanderfolgende Komponenten zerlegt wird,
- $p(x_i | x_1, x_2, \dots, x_{i-1})$ die bedingte Wahrscheinlichkeitsverteilung der i -ten Komponente x_i ist, gegeben alle vorhergehenden Komponenten x_1, \dots, x_{i-1} .

Ein autoregressives Modell nutzt diese Zerlegung. Der zu generierende Datenpunkt $x \in X$ wird nicht als Ganzes, sondern schrittweise aus seinen Komponenten x_1, x_2, \dots, x_n aufgebaut. Der zugrunde liegende Funktionsapproximator erlernt dabei für jeden Schritt die bedingte Wahrscheinlichkeit $p(x_i | x_1, \dots, x_{i-1})$. Diese dient zur Vorhersage der nächsten Komponente x_i basierend auf den bereits generierten (oder bekannten) vorherigen Komponenten. Anstatt den vollständigen Datenpunkt in einem einzigen Schritt zu erzeugen, wird der Funktionsapproximator iterativ aufgerufen. Dieser Prozess wird fortgesetzt, bis der gesamte Datenpunkt x generiert ist.

DETERMINISIERUNG IN DOPPELKOPF

In dieser Arbeit wird der Einsatz von MCTS und AZ in Spielen mit imperfekter Information am Beispiel des Spiels Doppelkopf mithilfe von Determinisierung untersucht. Dazu müssen für alle drei Schritte des Determinisierungsansatzes geeignete Verfahren ausgewählt und zu einem Gesamtsystem kombiniert werden, das eine Strategie für Spiele mit imperfekter Information erzeugt.

5.1 AUSWAHL DER ZUSTÄNDE

Die Auswahl der Zustände erfolgt mithilfe eines autoregressiven Modells auf Basis von NN. Ein solcher Ansatz ermöglicht neben der Generierung konsistenter Zustände auch die Modellierung einer Glaubensverteilung, wodurch die wahrscheinlich vorliegenden Zustände präziser bestimmt werden können. Zum Vergleich wird der in [SH15] vorgestellte CAP-Algorithmus herangezogen, der durch spielspezifische Regeln konsistente Zustände erzeugt, jedoch keine Glaubensverteilung modelliert.

5.1.1 Autoregressives Modell

Ziel des autoregressiven Modells ist es, ausgehend von der bekannten Spielhistorie beziehungsweise der Informationsmenge U , konsistente Zustände gemäß der Glaubensverteilung zu generieren.

5.1.1.1 Strategieannahme

Die exakte Bestimmung der Glaubensverteilung erfordert idealerweise die Kenntnis der Strategien aller anderen Mitspieler. Dieses Wissen steht jedoch nicht zur Verfügung. Es ist gerade das Ziel, eine Strategie zu entwickeln, die sich gegen beliebige (unbekannte) Spieler behaupten kann. Zwar existieren fortgeschrittene RL-Algorithmen, die versuchen, Strategien der Mitspieler auf Basis des Spielverlaufs zu modellieren (sogenanntes *opponent modeling*); eine solche Modellierung wird hier jedoch nicht verfolgt.

Vielmehr liegt dem autoregressiven Modell eine Annahme zugrunde, die zwar offensichtlich nicht zutrifft, aber eine brauchbare Heuristik darstellen kann: Es wird angenommen, dass alle Spieler der gleichen Strategie folgen wie der betrachtete Agent selbst. Diese Annahme ermöglicht ein Training nur durch Selbstspiel. Es ist plausibel, dass sie auch zu guten Ergebnissen führen kann. Sofern der Agent selbst (annähernd) rational spielt, ist es naheliegend, dass sich auch die Mitspieler rational verhalten.

Die Problematiken dieser Annahme sowie mögliche Alternativen durch *opponent modeling* werden im Abschnitt 8.3 und im Abschnitt 8.4.4 im Rahmen der Diskussion nochmals aufgegriffen.

5.1.1.2 Schrittweise Generierung

Das autoregressive Modell zerlegt den Generierungsprozess eines vollständigen Zustands in eine Abfolge von Einzelschritten. Ausgangspunkt ist eine Informationsmenge, in der die Karten der Mitspieler unbekannt sind. Reihum werden Karten (und gegebenenfalls Vorbehalte) den Mitspielern zugewiesen. In jedem Schritt wird genau eine Karte einem Spieler zugeordnet. Diese Zuordnung erfolgt unter der Annahme, dass alle bisher zugewiesenen Karten sich bereits auf den Händen der jeweiligen Spieler befinden.

Neben der Vorhersage der Handkarten muss das Modell der Vollständigkeit halber auch nicht aufgedeckte Vorbehalte vorhersagen. Dabei handelt es sich um solche Vorbehalte, die nicht offenbart wurden, weil ein anderer Spieler zuvor einen höherwertigen Vorbehalt geäußert hat. Da sich die Vorhersage dieser verdeckten Vorbehalte konzeptionell nicht von der Handkartenvorhersage unterscheidet, wird im Folgenden der Einfachheit halber nur von der Vorhersage der Handkarten gesprochen.

Die vom Funktionsapproximator in jedem Zuweisungsschritt zu approximierende bedingte Wahrscheinlichkeitsverteilung lässt sich wie folgt formulieren:

$$P(c | U, H_l, H_o, H_r, p, \pi)$$

wobei gilt:

- U ist der bekannte Spielverlauf (Informationsmenge), einschließlich der eigenen Handkarten,
- H_l, H_o, H_r sind die bisherigen Annahmen über die Kartenhände der Mitspieler (*Links, Oben, Rechts*),
- p ist der Spieler, dem als Nächstes eine Karte zugewiesen werden soll,
- π ist die zugrunde gelegte (identische) Strategie aller Spieler,
- c ist eine Karte, die sich noch auf der Hand eines Mitspielers befinden könnte.

Anders ausgedrückt: Gegeben ist die bekannte Spielhistorie bzw. Informationsmenge U , die unter der Strategie π entstanden ist, sowie eine bisherige Annahme über die Kartenverteilung der Mitspieler (H_l, H_o, H_r). Unter diesen Bedingungen ergibt sich die Frage: Mit welcher Wahrscheinlichkeit besitzt Spieler p noch eine bestimmte Karte c ?

5.1.1.3 Zustandsgenerierung

Um einen oder mehrere vollständige Zustände aus einer gegebenen Informationsmenge zu erzeugen, wird der Funktionsapproximator schrittweise wiederholt aufgerufen, bis alle Karten verteilt sind. In jedem Schritt kann dabei

stochastisch oder deterministisch vorgegangen werden. Bei der *stochastischen Vorhersage* wird die nächste Karte gemäß der approximierten Verteilung zufällig ausgewählt. Bei der *deterministischen Vorhersage* wird aus jeder Verteilung jeweils die wahrscheinlichste Karte gewählt, um den wahrscheinlichsten Zustand zu erzeugen.

5.1.1.4 Training durch Rückwärtskonstruktion

Für das Training des Modells wird in umgekehrter Richtung zum Generierungsprozess vorgegangen. Ausgehend von vollständigen Zuständen, die durch Selbstspiel mit der Strategie π erzeugt wurden, wird den Spielern schrittweise eine Handkarte entnommen. Das Modell wird darauf konditioniert, die dem Spieler p weggenommenen Karten korrekt vorherzusagen, basierend auf den nun veränderten Kartenannahmen H_l, H_o, H_r . Da das Modell eine Wahrscheinlichkeitsverteilung über mögliche Karten lernen soll, wird als Verlustfunktion die CE-Verlustfunktion verwendet.

5.1.1.5 Modelleingaben

Mit den folgenden Merkmalen einer Informationsmenge stehen dem Modell prinzipiell alle Informationen zur Verfügung, die auch ein menschlicher Spieler zur Entscheidungsfindung nutzen kann:

- **Startspieler:** Der Startspieler der Stiche und Vorbehalte.
- **Vorbehalte der Spieler:** Die aufgedeckten und vermuteten Vorbehalte der Spieler.
- **Tischkarten:** Die Karten der bereits gespielten Stiche sowie des aktuellen Stiches.
- **Handkarten:** Die aktuellen eigenen Handkarten.
- **Ansagen und Absagen:** Die von den Spielern getätigten Ansagen und Absagen.
- **Nächster Spieler:** Der Spieler, dessen nächste Karte ermittelt werden soll.

Die Ausgabe des Modells ist eine Wahrscheinlichkeitsverteilung über die Karten, die sich noch auf der Hand des nächsten Spielers befinden könnten. Die konkrete Architektur und die Kodierung dieser Merkmale im Rahmen eines NN werden im Implementierungskapitel beschrieben.

5.1.1.6 Konsistenz durch Maskierung

Das Modell arbeitet probabilistisch und gibt in jedem Schritt eine Wahrscheinlichkeitsverteilung über alle möglichen Karten eines Spielers aus. Da NN kontinuierliche Funktionen mit Fließkommazahlen zur Approximation verwenden

det, können dabei auch ungültige oder inkonsistente Zuweisungen mit kleiner, aber positiver Wahrscheinlichkeit auftreten. Dies geschieht insbesondere in frühen Trainingsphasen oder bei begrenzter Modellkapazität.

Um die Generierung inkonsistenter Zustände zu reduzieren, wird vor jeder Vorhersage eine Maske angewendet. Diese weist allen Karten, die ein Spieler nach den Spielregeln nicht besitzen kann, eine Wahrscheinlichkeit von 0 zu. Die Wahrscheinlichkeiten der verbleibenden Karten werden anschließend entsprechend normalisiert.

Die für Doppelkopf geltenden Invarianten bei der Kartenzuweisung lauten:

- Jede Karte existiert genau zweimal im Spiel (auf den Händen der Spieler oder auf dem Tisch).
- Hat ein Spieler eine Farbe nicht bekannt, besitzt er keine weitere Karte dieser Farbe.
- Hat ein Spieler eine Hochzeit angesagt, so besitzt er beide ♣Q-Damen oder hat sie bereits ausgespielt.
- Im Normalspiel gilt: Wer „Re“ oder eine niedrigere Absage angesagt hat, besitzt mindestens eine ♣Q-Dame oder hat diese bereits ausgespielt.
- Im Normalspiel gilt: Wer „Kontra“ oder eine niedrigere Absage ange sagt hat, besitzt keine ♣Q-Dame.

Trotz dieser Maskierung kann es bei der hier verfolgten autoregressiven Generierung zu inkonsistenten Zuständen kommen: Angenommen, *Unten* kann sowohl die ♠10 als auch das ♠A, *Rechts* aber nur das ♠A in der Hand halten. Weist man *Unten* nun die ♠A zu, kann man die ♠10 niemandem mehr zuweisen. Dieses Problem soll im Folgenden als Zuweisungsproblem bezeichnet werden.

5.1.2 CAP-Algorithmus

[SH15] stellen in ihrer Arbeit fest, dass eine theoretische Möglichkeit zur Ermittlung konsistenter Zustände in Doppelkopf darin besteht, zunächst alle möglichen Kartenverteilungen zu einer Informationsmenge zu bestimmen und daraus zufällige Stichproben zu ziehen. Dieses Aufzählungsproblem lässt sich als graphentheoretisches Problem modellieren: Die Suche nach allen konsistenten Kartenverteilungen entspricht dem Bestimmen perfekter Matchings in einem bipartiten Graphen, wobei die noch verfügbaren Karten den Karten-slots der Mitspieler zugeordnet werden.

Da bereits das zugehörige Zählproblem #P-vollständig ist, kommen die Autoren zu dem Schluss, dass eine derart exakte und gleichverteilte Lösung praktisch nicht realisierbar ist. Sie verwerfen daher die Anforderung an eine gleichverteilte Stichprobe und entwickeln den Card-Assignment-Problem-Algorithmus, der darauf verzichtet, dafür aber praktisch berechenbar ist.

Ausgehend von einer Informationsmenge geht der Algorithmus wie folgt vor:

1. Zunächst werden für jeden Spieler alle Karten bestimmt, die er ohne Verletzung der Invarianten (vgl. Abschnitt 5.1.1.6) auf der Hand haben könnte.
2. Anschließend werden iterativ alle verbleibenden Karten zugewiesen, bis alle Karten verteilt sind. Die folgenden Schritte werden ausgeführt:
 - a) Existiert eine Karte, die nur einem Spieler zugeordnet werden kann, wird sie diesem Spieler zugewiesen.
 - b) Ist bei einem Spieler die Anzahl der verbleibenden Kartenslots gleich der Anzahl der ihm noch zuordenbaren Karten, werden ihm diese Karten direkt zugewiesen.
 - c) Muss ein Spieler aufgrund seiner „Re“-Ansage oder niedrigeren Absage zwingend eine ♣Q halten, wird sie ihm zugewiesen.
 - d) In allen übrigen Fällen wird eine beliebige noch nicht verteilte Karte zufällig ausgewählt und einem Spieler zugewiesen, der sie laut Invarianten halten darf.

Für diesen Algorithmus zeigen [SH15], dass er ausschließlich konsistente Kartenverteilungen erzeugt, immer terminiert und somit das Zuweisungsproblem vermeidet.

5.2 EINZELZUSTANDSANALYSE

Die Einzelzustandsanalyse erfolgt durch MCTS beziehungsweise AZ. Beide Verfahren wurden bereits erläutert und funktionieren grundsätzlich wie beschrieben. In diesem Abschnitt werden nur die für das Spiel Doppelkopf notwendigen Anpassungen und Eigenheiten dargestellt.

Erste Tests des MCTS beziehungsweise des AZ zeigten bereits Unzulänglichkeiten der Algorithmen, die einen Erfolg (bei begrenzter Anzahl von Iterationen) in der vorgestellten Modellierung des Spiels Doppelkopf unwahrscheinlich und praktisch unmöglich machten. Für diese Probleme sind Anpassungen der Algorithmen notwendig, die im Folgenden beschrieben werden. Diese sind heuristischer Natur und schränken die Zielerreichung, eine Strategie ohne spielspezifische Anpassungen zu finden, ein.

5.2.1 Monte Carlo Tree Search

Für die Anwendung des MCTS auf das Spiel Doppelkopf wurden die im Folgenden beschriebenen Modifikationen angewendet.

5.2.1.1 Simulationsheuristik

Während der ersten Tests von MCTS für das Spiel Doppelkopf traten ungewöhnliche Ergebnisse auf: Unabhängig von der Anzahl der Iterationen und der Wahl des UCT-Parameters führten die Partien stets dazu, dass sich beide

Parteien gegenseitig „schwarz“ absagten. Mit einer solchen Absage behauptet eine Partei, dass die Gegenpartei keinen einzigen Stich erhalten wird. In der Praxis ist das nur in wenigen Spielsituationen sinnvoll. Insofern wurde in nahezu keinem Fall eine sinnvolle Strategie ermittelt.

Die Ursache dafür liegt in der reinen Zufallssimulation innerhalb der Simulationsphase. Bei Doppelkopf (in der vorgestellten Modellierung) wird jeder Spieler reihum nach jedem Legen einer Karte gefragt, ob er eine An- oder Absage tätigen möchte. In einer Simulation, bei der die Aktionen gleichverteilt zufällig gewählt werden, kommt es aufgrund der wiederholten An- und Absagephasen nahezu immer zur „schwarz“-Absage. Die resultierenden Simulationsergebnisse sind entsprechend verzerrt: Es macht für die Spieler keinen Unterschied mehr, ob sie eine An- oder Absage treffen oder nicht, denn die Ergebnisse sind die gleichen.

Um diesem Effekt entgegenzuwirken, wurde die Simulationsphase modifiziert: Während der Zufallszüge werden keine An- oder Absagen mehr erlaubt. Die Simulation wird also unter der Annahme ausgeführt, dass danach keine An- oder Absagen von den Spielern erfolgen. Diese Vorgehensweise ähnelt dem Konzept der *Heavy Playouts* [DU07], bei dem innerhalb der Simulationsphase mit spielspezifischen Strategien gearbeitet wird. Im Unterschied dazu bleibt es hier aber im Kern bei einer zufälligen Simulation.

5.2.1.2 Einschränkung des Selektionsteilbaums

Dieses Problem der wiederholten Entscheidungssituationen führt innerhalb des Selektionsteilbaums zu erheblichem unnötigem Explorationsaufwand: Nach jeder ausgespielten Karte wird erneut überprüft, ob die Spieler An- oder Absagen treffen möchten. Da die Entscheidung für oder gegen eine An- oder Absage in jeder dieser ähnlichen Situationen die gleichen Konsequenzen nach sich zieht, entsteht unnötiger Explorationsaufwand. Bei geringer Iterationszahl führt das zu einer hohen Varianz in den Bewertungen $Q(s,a)$ und birgt das Risiko, dass Fehleinschätzungen einer Aktion aufgrund unzureichender Anzahl von Simulationen durch die häufige Wiederholung der Entscheidungsmöglichkeit den Suchprozess überproportional negativ beeinflussen.

Um dem entgegenzuwirken, wurde der MCTS so angepasst, dass im Selektionsteilbaum stets davon ausgegangen wird, dass die Spieler keine weiteren An- oder Absagen treffen. Lediglich im Wurzelknoten ist es weiterhin möglich, eine An- oder Absage zu tätigen, damit diese Option grundsätzlich erhalten bleibt. Unterhalb des Wurzelknotens wird jedoch angenommen, dass in diesem MCTS-Durchlauf keine zusätzlichen An- oder Absagen der Mitspieler erfolgen. Diese Änderung hat keine nachteiligen Auswirkungen auf die Spielstärke, da An- und Absagen im Wesentlichen unabhängig davon sind, ob die Gegner in Zukunft ebenfalls eine solche Entscheidung treffen.

Solche heuristischen Anpassungen in der Selektionsphase gelten im MCTS eher als unüblich, sind aber in diesem Fall notwendig, um die hohe Varianz

der Ergebnisse zu verringern und damit die Anzahl an benötigten Iterationen deutlich zu reduzieren [DU07].

5.2.2 *AlphaZero*

Um AZ auf das Spiel Doppelkopf anzuwenden, müssen die Funktionsapproximationen eine semantische Kodierung des Spielzustands erhalten. Die bereitgestellten Merkmale werden im Folgenden neben heuristischer Anpassungen beschrieben.

5.2.2.1 *Eingabemerkmale*

Mit der Bereitstellung der folgenden Merkmale eines Zustands stehen dem NN für die Funktionsapproximation prinzipiell alle Informationen zur Verfügung, die auch einem menschlichen Spieler zur Entscheidungsfindung bereitstehen:

- **Spielphase:** Angabe, in welcher Phase sich das Spiel befindet (Vorbehaltsphase, Ansagenphase, Kartenphase).
- **Startspieler:** Startspieler der Stiche und Vorbehalte.
- **Vorbehalte der Spieler:** Vorbehalte der Spieler.
- **Tischkarten:** Karten der bereits gemachten Stiche und des aktuellen Stiches.
- **Handkarten:** Aktuelle Handkarten aller Spieler.
- **Ansagen und Absagen:** Von den Spielern getätigte Ansagen und Absagen.

Am Ende des Wertekopfes stehen 4 Ausgaben, die jeweils eine Nutzenschätzung für einen der vier Spieler darstellen. Am Ende des Strategiekopfes stehen 39 mögliche Aktionen: Die Vorbehalte (einschließlich Gesundansage), mögliche An- und Absagen sowie jede zulässige Karte. Die genaue Architektur und Kodierung dieser Merkmale wird im Rahmen der Implementierung näher erläutert.

5.2.2.2 *Trainingsheuristik*

Das beschriebene Problem der wiederholten Entscheidungspunkte tritt in ähnlicher Form auch im Training des AZ-Modells auf. Zwar gibt es im AZ keine (zufälligen) Simulationen, jedoch führt ein untrainiertes Netzwerk in den frühen Trainingsiterationen zu einer nahezu zufälligen Auswahl der Aktionen. Dadurch tritt das gleiche Problem auf wie zuvor: In nahezu jeder Partie sagen alle Parteien „schwarz“ ab, was zu unsinnigen Spielverläufen führt. Das Training findet dann fast ausschließlich auf der Basis dieser abwegigen

Situationen statt. Realistische Spielverläufe, die für erfolgreiches Lernen erforderlich wären, treten hingegen nur mit verschwindend geringer Wahrscheinlichkeit auf. Um diesem Problem zu begegnen, wird das Ausführen von An- und Absagen durch den Agenten in den ersten t_e Trainingsiterationen explizit unterbunden. Dadurch werden die Agenten zunächst auf das Spiel ohne An- und Absagen konditioniert. In späteren Trainingsphasen wird diese Einschränkung aufgehoben, sodass An- und Absagen durch temperaturgesteuerte Exploration langsam wiederentdeckt werden können.

5.2.2.3 Einschränkung des Selektionsteilbaums

Die in Abschnitt 5.2.1.2 beschriebenen Anpassungen des Selektionsteilbaums wurden aus denselben Gründen auch für den AZ-Algorithmus übernommen. Ziel ist es, die Größe des Selektionsteilbaums zu reduzieren.

5.2.3 Zufallsereignisse

Wie in Abschnitt 4.4.2.3 ausgeführt, tritt das einzige Zufallsereignis in Doppelkopf, die initiale Verteilung der Karten, ausschließlich im Wurzelknoten des Spielbaums auf. Da der gesamte nachfolgende Spielverlauf deterministisch ist, erfordert die Anwendung von MCTS und AZ keine strukturellen Modifikationen dieser Algorithmen.

5.3 STRATEGIE-FUSION

In der Literatur werden verschiedene Strategien zur Fusion von Strategien in Spielen mit imperfekter Information vorgestellt. In dieser Arbeit werden zwei Ansätze untersucht, die für das Spiel Doppelkopf vielversprechend erscheinen. Die Anwendung einer Fusionsstrategie, die nur auf Grundlage des durchschnittlichen Nutzens erfolgt (verwendet zum Beispiel von [Gino1] und [Lev89]), wurde nicht verwendet. Aufgrund der starken Varianz der zu erreichenden Spieldurchgänge bei unterschiedlichen Kartenverteilungen im Spiel Doppelkopf erscheint der Ansatz nicht überzeugend.

5.3.1 Mittels kumulierten Rangs

Für die Fusion mittels kumulierten Rangs müssen die Besuchsstatistiken $N(s, a)$ der vom Wurzelknoten aus erreichbaren Aktionen jeder MCTS-Ausführung eines Zustands zur Verfügung stehen.

Ordnet man in einem Zustand $s \in U_i^j$ die möglichen Aktionen anhand der durch den MCTS-Durchlauf ermittelten Besuchszahlen, ergibt sich eine Rangordnung der Aktionen. Der Rang der Aktion a im Zustand s wird im Folgenden als $\text{rank}(s, a)$ bezeichnet, wobei ein niedriger Rangwert eine bessere Bewertung nach Besuchshäufigkeiten im jeweiligen Zustand anzeigt. Der kumulierte Rang ergibt sich dann wie folgt:

Definition 30 (Kumulierter Rang einer Aktion) Sei $S \subseteq U$ die Teilmenge der Zustände, die aus der Informationsmenge ausgewählt wurde. Sei $a \in A(U)$ eine in der Informationsmenge mögliche Aktion. Der kumulierte Rang von a ist definiert als:

$$\text{rank}_c(a) = \sum_{s \in S} \text{rank}(s, a).$$

Die Fusionsstrategie wählt dann die Aktion mit dem niedrigsten kumulierten Rang:

$$a^* = \arg \min_{a \in A(U)} \text{rank}_c(a).$$

Die zugrundeliegende Intuition besteht darin, solche Aktionen zu bevorzugen, die über möglichst viele Zustände hinweg eine gute oder zumindest akzeptable Bewertung erhalten. Es kann dabei durchaus vorkommen, dass eine Aktion in keinem Zustand die erste Wahl darstellt (und somit bei perfekter Information nicht gewählt würde), aber durch ihre konstante Platzierung im oberen Mittelfeld dennoch die insgesamt robusteste Wahl darstellt. Ein Beispiel für einen solchen Fall ist in Tabelle 5.1 dargestellt.

	a_1	a_2	a_3	a_4
s_1	1	2	4	3
s_2	3	3	2	2
s_3	3	2	3	1
s_4	3	2	1	4
Summe	10	9	10	10

Tabelle 5.1: Ein Beispiel der Fusionsstrategie mittels kumulierten Rangs.

5.3.2 Mittels durchschnittlicher Strategie

Als Alternative zur Fusion auf Basis des durchschnittlichen Nutzens einer Aktion stellen [BCK23] einen Ansatz vor, bei dem die in den einzelnen Zuständen ermittelten Wahrscheinlichkeitsverteilungen (Strategien) über die möglichen Aktionen gemittelt werden.

In jedem Zustand $s \in U$ wird eine individuelle Strategie $\pi(s)$ bestimmt. Diese Strategie entspricht einer Wahrscheinlichkeitsverteilung über die möglichen Aktionen $a \in A(U)$. Typischerweise wird $\pi(s, a)$ dabei aus der Anzahl der Besuche $N(s, a)$ abgeleitet.

Definition 31 (Durchschnittliche Strategie einer Aktion) Sei $S \subseteq U$ die Teilmenge der Zustände, die aus der Informationsmenge $(U, A(U))$ ausgewählt wurde. Sei $\pi(s)$ die im Zustand s ermittelte Strategie.

$$\pi_{avg}(a) = \frac{1}{|S|} \sum_{s \in S} \pi(s, a).$$

Die Aktion mit der höchsten durchschnittlichen Wahrscheinlichkeit wird ausgewählt:

$$a^* = \arg \max_{a \in A(U)} \pi_{avg}(a).$$

Im Unterschied zur Fusion mittels kumulierten Rangs fließt bei der Fusion mittels durchschnittlicher Strategie die vollständige Wahrscheinlichkeitsverteilung $\pi(s)$ in die Bewertung ein. Es zählt also nicht nur der Rang einer Aktion, sondern auch, wie deutlich sie in einem Zustand bevorzugt wird. Eine Aktion, die in wenigen Zuständen mit sehr hoher Wahrscheinlichkeit gewählt wird, kann das Gesamtergebnis stärker beeinflussen als eine Aktion, die in vielen Zuständen nur knapp vorn liegt. Dadurch bildet dieser Ansatz die Stärke individueller Präferenzen differenzierter ab.

5.4 ZUSAMMENSETZUNG DER KOMPONENTEN

Die zuvor eingeführten Komponenten zur *Zustandsauswahl*, *Einzelzustandsanalyse* und *Strategie-Fusion* müssen zu einem Gesamtsystem kombiniert werden, um eine Strategie für imperfekte Information zu bilden. Dabei lassen sich drei grundsätzliche Konstellationen unterscheiden:

- **Dynamische Strategie:** Sowohl das Verfahren der Einzelzustandsanalyse als auch das Modell zur Auswahl plausibler Zustände erfordern ein vorheriges Training. Dies ist der Fall, wenn der AZ-Algorithmus mit dem autoregressiven Modell kombiniert werden soll.
- **Teildynamische Strategie:** Das Verfahren der Einzelzustandsanalyse bedarf keines vorhergehenden Trainings; das Modell für die Auswahl der Zustände jedoch schon. Dies ist der Fall, wenn der MCTS-Algorithmus mit dem autoregressiven Modell kombiniert werden soll.
- **Statische Strategie:** Weder die Zustandsauswahl noch die Einzelzustandsanalyse basieren auf trainierten Komponenten. Dies ist der Fall, wenn der statische MCTS zusammen mit dem statischen Card-Assignment-Problem (CAP)-Algorithmus kombiniert wird.

5.4.1 *AlphaZero mit autoregressivem Modell*

In dieser Arbeit wird die Kombination des AZ-Algorithmus mit einem autoregressiven Modell zur Zustandsauswahl untersucht. Der Trainingsprozess erfolgt in den folgenden Schritten:

1. **Selbstspiel mit perfekter Information:** Zunächst wird das AZ-Trainingsverfahren im Selbstspiel durchgeführt. Dabei entsteht eine Strategie für perfekte Information, die in Abbildung 5.1 als $\pi_{\text{AlphaZero}^{\text{PI}}}$ bezeichnet wird.

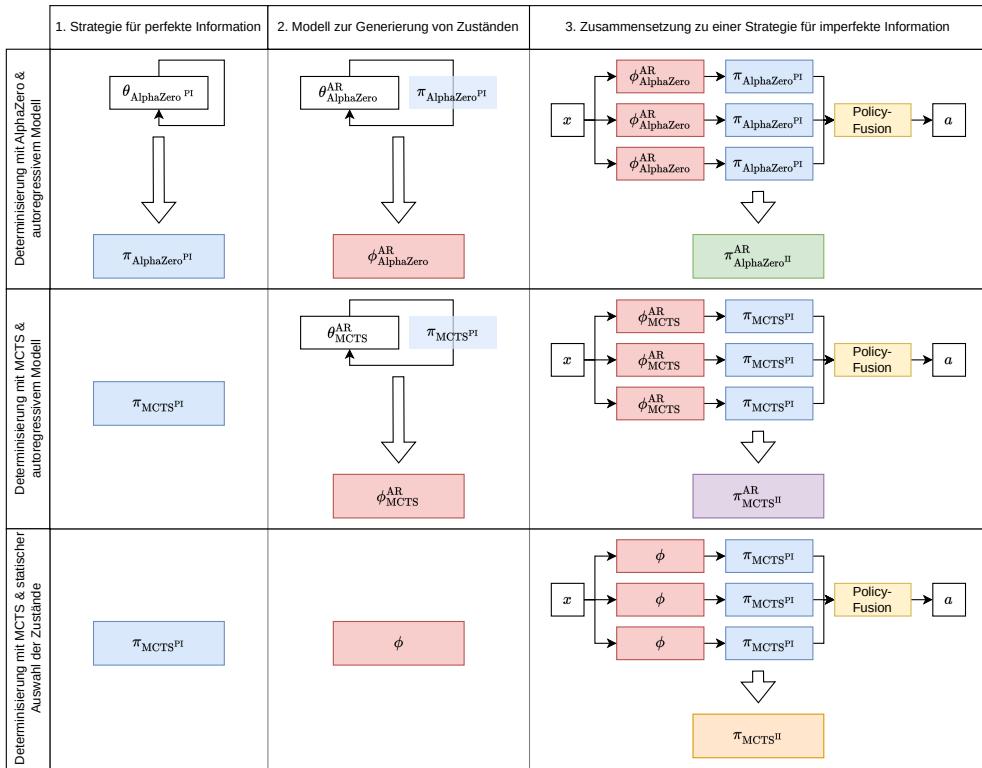


Abbildung 5.1: Die Zusammensetzungen der unterschiedlichen Determinisierungsansätze.

2. Training des autoregressiven Modells: Nachdem die Strategie $\pi_{\text{AlphaZero}^{\text{PI}}}$ vollständig trainiert wurde, werden neue Selbstspielpartien generiert, in denen alle vier Spieler dieser Strategie folgen. Auf Basis dieser Spielverläufe wird das autoregressive Modell $\theta_{\text{AlphaZero}}^{\text{AR}}$ trainiert. Es wird darauf konditioniert, aus Informationsmengen jene Zustände zu rekonstruieren, die unter dieser Strategie typischerweise auftreten. Daraus ergibt sich das Zustandsgenerierungsverfahren $\phi_{\text{AlphaZero}}^{\text{AR}}$.

Mit diesen beiden trainierten Komponenten und einer der beiden Strategie-Fusionsstrategien lässt sich eine vollständige Strategie für imperfekte Information konstruieren, die im Folgenden als $\pi_{\text{AlphaZero}^{\text{II}}}^{\text{AR}}$ bezeichnet wird. Die Zustandserzeugung erfolgt mittels stochastischer Vorhersage auf Basis der vom Modell gelieferten Wahrscheinlichkeitsverteilungen. Für jeden dieser Zustände wird der AZ-Algorithmus separat ausgeführt.

Ein zentraler Vorteil dieser Zusammensetzung und des Trainingsablaufs besteht darin, dass alle Bestandteile ausschließlich durch Selbstspiel trainiert werden. Der Agent generiert sein Wissen vollständig aus eigener Interaktion mit der Umgebung. Das ist ganz im Sinne des ursprünglichen AZ-Ansatzes. Ein wesentlicher Nachteil liegt in der fehlenden Modellierung unterschiedlicher Mitspielerstrategien.

5.4.2 MCTS mit autoregressivem Modell

Bei der Kombination des MCTS mit dem autoregressiven Modell zur Zustandsauswahl bedarf es nur eines Trainings des autoregressiven Modells. Dieses wird auf Basis von Zuständen trainiert, die durch Selbstspiel entstehen, bei dem alle Spieler der statischen MCTS-Strategie $\pi_{\text{MCTS}^{\text{PI}}}$ folgen. Es entsteht die Strategie $\pi_{\text{MCTS}^{\text{II}}}^{\text{AR}}$.

5.4.3 MCTS mit CAP-Algorithmus

Sowohl der MCTS als auch der CAP-Algorithmus zur Zustandsauswahl sind statisch und benötigen kein vorhergehendes Training. Mit der Zusammensetzung ergibt sich die Strategie $\pi_{\text{MCTS}^{\text{II}}}$.

6

IMPLEMENTATION

Nachdem in den vorangegangenen Kapiteln die theoretischen Grundlagen und Algorithmen beschrieben wurden, widmet sich dieses Kapitel der konkreten programmatischen Umsetzung des in dieser Arbeit untersuchten Determinisierungsansatzes zur Anwendung von MCTS und AZ auf das Spiel Doppelkopf mit imperfekter Information. Das entwickelte System dient als Grundlage für die experimentelle Evaluation in Kapitel 7, indem es die Simulation von Doppelkopf-Partien sowie das Training und die Ausführung der verschiedenen untersuchten Strategien ermöglicht. Die Umsetzung umfasst die folgenden Komponenten:

- Spielsimulator für Doppelkopf.
- Implementierungen der Einzelzustandsanalyse-Algorithmen: MCTS und AZ.
- Implementierungen der Zustandsauswahl-Methoden: Den statischen CAP-Algorithmus und ein trainierbares autoregressives Modell.
- Eine Integrationskomponente, welche die drei Komponenten (Zustandsauswahl, Einzelzustandsanalyse und Strategie-Fusion) zu einer Strategie für imperfekte Information verbindet.
- Eine Web-Anwendung zur selektiven Analyse und Demonstration.

6.1 SPIELSIMULATOR FÜR DOPPELKOPF

Für die Evaluierung anhand des Spiels Doppelkopf ist eine programmatische Simulation des Spiels mit allen Spielregeln als RL-Umgebung notwendig. Die Implementierung des Simulators erfolgte entsprechend der in Kapitel 3 definierten Modellierung und wurde in der Programmiersprache Rust [KN23] vorgenommen. Als systemnahe und kompilierte Sprache bietet sie eine hohe Performance. Bei der Ausführung von MCTS und AZ ist eine schnelle Ausführung von großer Bedeutung, da die Algorithmen in kurzer Zeit zahlreiche Spielsimulationen ausführen.

6.2 IMPLEMENTIERUNG VON MCTS

Die durchgeführten Experimente basieren auf einer eigenen Implementierung des MCTS-Algorithmus mit der UCT-Selektionsstrategie ebenfalls in Rust. Die Wahl von Rust erfolgte aus denselben Performance-Gründen wie beim Spielimulator, da der Aufbau und die Durchsuchung des Selektionsteilbaums

rechen- und speicherintensiv sind. Folgende Implementierungsdetails sind zu benennen:

- Die in Abschnitt 5.2.1.1 und in Abschnitt 5.2.1.2 beschriebenen Heuristiken für das Spiel Doppelkopf wurden umgesetzt.
- Als finale Aktion eines MCTS-Durchlaufs wird diejenige gewählt, die nach Abschluss aller Iterationen die höchste Besuchszahl $N(s, a)$ im Wurzelknoten aufweist (vgl. Abschnitt 4.1.1.2).

Insgesamt hat die Implementierung des MCTS damit lediglich zwei Parameter: Die Anzahl der Iterationen I sowie den UCT-Parameter C . Im Rahmen der Evaluation werden für unterschiedliche Untersuchungen auch unterschiedliche Parameter verwendet. Schrittweise wird ein geeigneter UCT-Parameter C und eine ausreichende Iterationsanzahl I ermittelt.

6.3 IMPLEMENTIERUNG VON ALPHAZERO

Die Implementierung des AZ-Algorithmus stellt eine hybride Lösung dar. Die MCTS-Komponente und die Interaktion mit dem Spielsimulator wurden in Rust implementiert. Für das Training und die Ausführung der Funktionsapproximatoren wurde hingegen das Deep-Learning-Framework PyTorch [Pas+17] verwendet. PyTorch ermöglicht mit seinem *nn*-Modul die GPU-beschleunigte Definition, Ausführung und das Training komplexer NN mittels Backpropagation.

6.3.1 Architektur der Funktionsapproximatoren

Eine Komponente des AZ sind die Funktionsapproximatoren. Um den Algorithmus auf Doppelkopf anwenden zu können, müssen die Architektur und die Kodierung der Eingaben eines Zustands festgelegt werden. Sowohl die Kodierung als auch die Architektur sowie deren Modellgröße müssen so gewählt werden, dass die Funktionsapproximatoren gut approximieren und generalisieren können.

6.3.1.1 Kodierung der Eingaben

Die hier gewählte Kodierung der Eingaben basiert auf der Idee, alle relevanten Entitäten des Spielzustands, also Vorbehalte, Tischkarten, Handkarten und An- und Absagen, als vereinheitlichte *Spielobjekte* zu behandeln. Jedes dieser Objekte wird durch seinen Typ (z. B. \diamond_9 , *Re*, *Gesund*) kategorisiert und um zusätzliche Kontextinformationen angereichert.

- **Position:** Die Position für ein Spielobjekt gibt für eine Tischkarte, einen Vorbehalt oder eine An- und Absage an, zu welchem Zeitpunkt (indexiert von 1 bis 52) das Spielobjekt gespielt beziehungsweise angesagt wurde. Im Falle einer Handkarte wird der Spieler kodiert, auf dessen Hand sich die Handkarte befindet.

- **Subposition:** Die Subposition für ein Spielobjekt gibt im Falle mehrerer An- und Absagen, die zum gleichen Zeitpunkt gemacht wurden, die Reihenfolge an. Im Falle von Handkarten gibt die Subposition an, ob es sich bei einer Karte um das erste Vorkommen innerhalb der Hand handelt oder um das zweite Vorkommen.
- **Ausspieler:** Der Ausspieler für ein Spielobjekt gibt im Falle von An- und Absagen, Tischkarten sowie Vorbehalten den Spieler an, der angesagt beziehungsweise die Karte gespielt hat.
- **Partei:** Im Falle einer An- und Absage wird zusätzlich die Partei kodiert. Damit ist es also die Angabe, ob es sich um eine „Re“- oder „Kontra“-Ansage handelt.
- **Phase:** Die aktuelle Phase wird in jedem Spielobjekt kodiert.

Insgesamt gibt es damit 62 (angereicherte) Spielobjekte, bestehend aus 4 möglichen Vorbehalten, 48 Karten sowie 10 möglichen An- und Absagen. Falls ein Objekt (z. B. eine An- oder Absage oder ein Vorbehalt) nicht existiert oder eine Zusatzinformation keinen Sinn ergibt, wird ein Leerwert an dieser Stelle kodiert.

6.3.1.2 Architektur des neuronalen Netzes

Die Architektur des NN ist in Abbildung 6.1 dargestellt und folgt diesen Schritten:

- **Embedding:** Jede der fünf Komponenten eines angereicherten Spielobjekts (Typ, Position, Subposition, Ausspieler, Partei, Phase) wird durch eine eigene Embedding-Schicht in einen Vektor der Dimension n_{embd} überführt. Die resultierenden Vektoren werden addiert, um eine Repräsentation zu erhalten, die alle diese Informationen für jedes Spielobjekt kodiert. Diese Repräsentation wird hier als das angereicherte Spielobjekt bezeichnet.
- **Gemeinsamer Transformer-Rumpf:** Die 62 Objektvektoren werden durch n_{layer} hintereinandergeschaltete Transformer-Blöcke verarbeitet. Dabei wird der Self-Attention-Mechanismus mit n_{head} Köpfen pro Block verwendet.
- **Aufteilung in Köpfe:** Nach dem gemeinsamen Rumpf teilt sich das Netzwerk in zwei separate Köpfe auf: einen für die Strategiefunktion und einen für die Wertfunktion.
- **Kopfspezifische Verarbeitung:** Beide Köpfe verfügen jeweils über eigene Transformer-Blöcke mit eigenen Parametern. Die Anzahl dieser Blöcke ist konfigurierbar und wird durch die Hyperparameter n_{layer}^{π} für den Strategiekopf und n_{layer}^v für den Wertekopf bestimmt.

- **Attention-Pooling:** Im Anschluss erfolgt ein sogenanntes *Attention-Pooling*. Dieses ähnelt dem in Abschnitt 4.2.3.2 beschriebenen Selbstaufmerksamkeitsmechanismus, unterscheidet sich jedoch in einem wesentlichen Punkt: Während dort für jede Eingabe ein eigener Query-Vektor berechnet wird, existiert hier genau ein Query-Vektor pro Ausgabe. Dieser wird direkt als Parameter gelernt und dient dazu, aus den 62 Objektvektoren gezielt nur eine kompakte Repräsentation zu extrahieren, die für die jeweilige Ausgabe relevant ist.¹
- **Ausgabe:** Nachdem für jede Ausgabe nur noch ein Vektor mit einer für die Ausgabe relevanten Repräsentation übrig ist, folgt ein kleines MLP mit einer versteckten Schicht der Größe n_{embd} mit eigenen Parametern für jede Ausgabe. Dieses ermöglicht eine finale Verarbeitung dieses Vektors spezifisch für jede mögliche Ausgabe. Im Fall des Wertekopfes also für die Nutzenschätzung jedes Spielers (4) und im Fall des Strategiekopfes für jede mögliche Aktion (39).

Die Transformer-Architektur soll in diesem Einsatzzweck das NN in die Lage versetzen, die globalen Abhängigkeiten zwischen den Spielobjekten zu erkennen und entsprechend zu generalisieren. Das Attention-Pooling sowie das MLP mit eigenen Parametern für jede Ausgabe sind ungewöhnlich, haben jedoch in der vergleichbaren Architektur für das autoregressive Modell zu deutlich genaueren Schätzungen und einem schnelleren Training geführt. Praktisch hat sich die vorliegende Architektur als hinreichend effizient und gut generalisierend erwiesen.

Die konkrete Umsetzung basiert auf der GPT2-Implementierung im Projekt *minGPT* von Andrej Karpathy [Kar20] [Rad+19]. Da sich die Anforderungen eines Sprachmodells jedoch grundlegend von denen des hier verwendeten Modells unterscheiden, wurden umfangreiche Anpassungen vorgenommen.

6.3.2 Heuristiken

Die in den Abschnitten 5.2.2.3 und 5.2.2.2 beschriebenen heuristischen Anpassungen für das Spiel Doppelkopf wurden übernommen.

6.3.3 Performance-Optimierungen

Die wesentliche Rechenlast des AZ-Algorithmus entsteht während der Selbstspielphase durch den Aufruf des NN in jeder Iteration des MCTS. Die Rechenlast des MCTS selbst ist dabei von untergeordneter Bedeutung, da AZ nur wenige Iterationen vornimmt. Zur Verbesserung der Performance wurden zwei Optimierungen eingesetzt:

- **Batching:** Mehrere MCTS-Durchläufe wurden parallel ausgeführt. Die dabei gesammelten Aufrufe an das NN wurden zu einem großen Batch

¹ Ein vergleichbarer Ansatz findet sich beispielsweise in [ITW18].

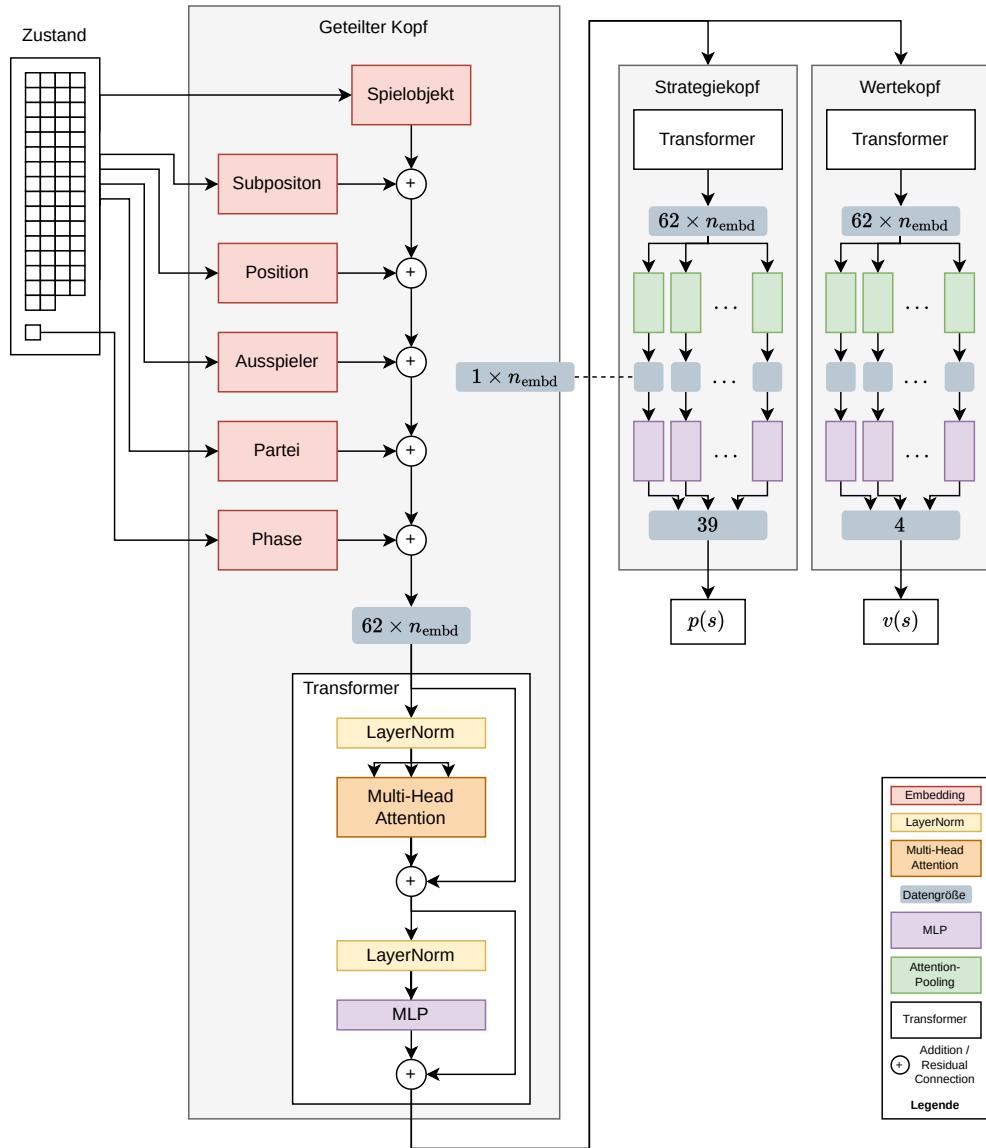


Abbildung 6.1: Die Architektur des verwendeten NN für den AlphaZero-Algorithmus im Spiel Doppelkopf.

zusammengefasst und gemeinsam verarbeitet. Da die Berechnungen auf GPUs durchgeführt wurden und diese besonders effizient im Umgang mit Batches sind, wird die Performance deutlich gesteigert.

- **Caching:** Ergebnisse bereits erfolgter NN-Aufrufe werden zwischengespeichert und bei erneutem Bedarf wiederverwendet. Dadurch lässt sich die Anzahl redundanter Berechnungen reduzieren und die Effizienz weiter erhöhen.

Beide Maßnahmen wurden in der ursprünglichen AZ-Veröffentlichung nicht beschrieben, haben jedoch keinen Einfluss auf die Ergebnisse. In modernen Open-Source-Implementierungen von AlphaZero sind diese Optimierungen gängige Praxis, beispielsweise im freien AZ-Nachbau *Leela Chess Zero* [Lco].

6.3.4 Parameter des verwendeten Modells

Aufgrund der rechenintensiven Natur des AZ-Algorithmus, einer nicht optimalen Implementierung sowie begrenzter Rechenkapazitäten wurde lediglich ein Modell mit einer Menge an Hyperparametern trainiert. Die Auswahl dieser Hyperparameter erfolgte manuell auf Grundlage kleiner Vorversuche sowie eigener Überlegungen.

Das zugrunde liegende NN wurde bewusst mit einer größeren Anzahl an lernbaren Parametern versehen, als für die vorliegende Problemstellung (vermutlich) notwendig gewesen wäre. Diese Entscheidung ist insofern gerechtfertigt, als der Fokus dieser Arbeit auf dem übergeordneten Trainingsprozess des AZ-Algorithmus liegt und nicht auf einer architektonischen Feinabstimmung des neuronalen Netzes. Da es sich um eine RL-Aufgabe handelt, bei der Trainingsdaten kontinuierlich durch Selbstspiel generiert werden, ist das Risiko einer Überanpassung (Overfitting) eher gering.

Der Trainingslauf erfolgte wegen der hohen Hardwareanforderungen mit einer relativ geringen Anzahl von 200 Iterationen. Aufgrund der bereits in Abschnitt 5.2.1.1 und später im Rahmen der Diskussion in Abschnitt 8.2.2 beschriebenen Problematik der wiederholten Entscheidungssituationen wurden für die Hyperparameter, welche die Exploration steuern, geringe Werte gewählt. Die Hyperparameter, welche die Exploration beeinflussen, sind die des Dirichlet-Rauschens (α und ϵ), der PUCT (C) sowie die Spieltemperatur (τ). Die allgemeinen Trainingsparameter des NN, Lernrate, Optimierer und Batch-Größe entstammen dem als Basis verwendeten minGPT-Projekt. Einen Überblick über die verwendeten Hyperparameter zeigt Tabelle 6.2a.

MCTS und Exploration	
MCTS-Iterationen I	200
PUCT-Parameter c_{puct}	4
Dirichlet- α	0,8
Dirichlet- ϵ	0,3
Temperatur τ	0,5
Spiele pro Trainingsiteration	8.192
An- und Absagen möglich ab t_e	10
Netzarchitektur (NN)	
Embedding-Dimension n_{embd}	160
Anzahl Heads n_{head}	4
Transformer-Blöcke n_{layer}	4
Transformer-Blöcke im Strategiekopf n_{layer}^{π}	1
Transformer-Blöcke im Wertekopf n_{layer}^v	1
Training des NN	
Optimierer	Adam ²
Lernrate	0,0003
Batch-Größe	128
(a) Verwendete Hyperparameter für AlphaZero.	
Erzeugung	
Verfolgte Strategie	$\pi_{\text{AlphaZero}}^{\text{PI}} / \pi_{\text{MCTS}}^{\text{PI}}$
Anzahl gespielter Spiele k	100.000
Netzarchitektur (NN)	
Embedding-Dimension n_{embd}	196
Anzahl Heads n_{head}	4
Transformer-Blöcke n_{layer}	6
Training des NN	
Optimierer	Adam
Lernrate	0,0003
Batch-Größe	128
(b) Verwendete Hyperparameter für die generativen Modelle.	

Abbildung 6.2: Die verwendeten Hyperparameter für das AZ-Modell sowie das autoregressive Modell.

6.4 IMPLEMENTIERUNG DES AUTOREGRESSIVEN MODELLS

Die Implementierung des autoregressiven Modells zur Zustandsschätzung erfolgte ebenfalls als hybride Lösung. Während die Simulation von Selbstspielen zur Datengenerierung, basierend auf einer gegebenen Strategie für perfekte Information, aus Effizienzgründen in Rust implementiert wurde, erfolgte das Training des zugrundeliegenden NN mittels PyTorch in Python.

6.4.1 Funktionsapproximationen

Die Architektur des NN für das autoregressive Modell sowie die Kodierung seiner Eingaben orientieren sich eng an der Implementierung für den AZ-Algorithmus.

6.4.1.1 Kodierung der Eingaben

Das autoregressive Modell verarbeitet im Gegensatz zum AZ-Modell eine Informationsmenge. Analog zur AZ-Implementierung werden alle relevanten Entitäten dieser Informationsmenge als vereinheitlichte *Spielobjekte* repräsentiert. Dazu gehören die bekannten Vorbehalte, die auf dem Tisch liegenden Karten, erfolgte An- und Absagen, die Handkarten des Spielers sowie die bisher (im Generierungsprozess) angenommenen Handkarten der Mitspieler. Jedes dieser Spielobjekte wird kategorisiert und um Kontextinformationen wie Position, Subposition, Ausspieler und Partei angereichert. Die aktuelle Spielphase ist bei der Zustandsvorhersage nicht notwendig. Stattdessen erhält das Modell als zusätzliche Information den Spieler, dessen nächste Handkarte vorhergesagt werden soll.

6.4.1.2 Architektur des neuronalen Netzes

Die Architektur entspricht im Wesentlichen der des NN für den AZ-Algorithmus. Statt eigenständiger Köpfe führt der Rumpf selbst zur Ausgabe einer Wahrscheinlichkeitsverteilung über die Karten (und Vorbehalte), welche dem Spieler zugeordnet werden sollen, dessen nächste Handkarte vorhergesagt wird. Das sind insgesamt 33 Ausgaben (24 Karten und 9 Vorbehalte). Abbildung A.2 zeigt die Architektur in einer kompakten Form.

6.4.2 Training mit Strategie

Das Training des autoregressiven Modells basiert auf Zuständen, die durch Selbstspiele einer Referenzstrategie π^{PI} für perfekte Information generiert werden. Der Trainingsprozess gliedert sich in zwei Phasen:

² Der Adam-Optimierer [KB15] ist eine Weiterentwicklung des stochastischen Gradientenabstiegs und ergänzt ihn um adaptive Lernraten und Momentum, was meist zu einem stabileren Trainingsverlauf führt.

- **Datengenerierung:** Zunächst werden mithilfe der Strategie π^{PI} eine große Anzahl k vollständiger Doppelkopf-Partien im Selbstspiel simuliert. Alle dabei auftretenden vollständigen Spielzustände s werden gespeichert.
- **Training:** Aus der Menge der in der Datengenerierungsphase erzeugten vollständigen Zustände s werden wiederholt zufällig Stichproben ausgewählt. Für jeden ausgewählten Zustand werden mittels der in Abschnitt 5.1.1.4 beschriebenen Rückwärtskonstruktion Trainingstupel generiert. In jedem Trainingsschritt wird eine neue Stichprobe von Zuständen gezogen, die zugehörigen Trainingstupel werden erzeugt und das Modell anhand dieser trainiert. Dieser Zyklus aus Datensampling und Training wird wiederholt bis das Modell gut genug (für den verfolgten Anwendungszweck) ist.

6.4.3 Parameter der verwendeten Modelle

Im Rahmen dieser Arbeit wurden zwei separate autoregressive Modelle trainiert, die jeweils auf unterschiedlichen Referenzstrategien π^{PI} für die Datengenerierung basieren:

- **MCTS:** Ein Modell ($\phi_{\text{MCTS}}^{\text{AR}}$) trainiert auf Daten aus Selbstspielen der MCTS-Strategie ($\pi_{\text{MCTS}}^{\text{PI}}$) mit 500.000 Iterationen pro Zug und einem UCT-Parameter $C = 2,5$.
- **AlphaZero:** Ein Modell ($\phi_{\text{AlphaZero}}^{\text{AR}}$) trainiert auf Daten aus Selbstspielen der trainierten AZ-Strategie ($\pi_{\text{AlphaZero}}^{\text{PI}}$), deren Parameter in Abschnitt 6.3.4 beschrieben sind. Bei der Ausführung wurden die Explorationsparameter τ , ϵ und α auf 0 gesetzt.

Ähnlich wie beim AZ-Modell wurde auch hier bewusst eine relativ hohe Modellkapazität gewählt (siehe Tabelle 6.2b). Die Embedding-Dimension und die Anzahl der Transformer-Blöcke wurden gegenüber dem AZ-Modell sogar erhöht. Vorversuche deuteten darauf hin, dass diese höhere Parametrisierung zu einer schnelleren Konvergenz während des Trainings und zu präziseren Zustandsschätzungen führt. Dies erscheint plausibel, da die Generierung konsistenter und plausibler Zustände eine detailliertere Modellierung der komplexen Abhängigkeiten im Spiel erfordert als die reine Schätzung von Werten und Strategien, die auch Ungenauigkeit aushält. Eine größere Modellkapazität kann hierbei vorteilhaft sein, um diese Zusammenhänge zu erfassen. Die Gefahr der Überanpassung wird durch die große Menge an generierten Trainingsdaten aus den Selbstspielen reduziert.

6.5 DETERMINISIERUNG

Die Determinisierungskomponente bildet das Bindeglied zwischen Zustandsauswahl, Einzelzustandsanalyse und Strategie-Fusion, um eine Gesamtstrate-

gie für imperfekte Information zu erzeugen. Sie steuert den gesamten Prozess, von der Generierung der Zustandsstichproben bis zur finalen Aktionswahl. Parametrisiert wird sie durch die Anzahl der zu ziehenden Stichproben n_{sample} . Die Implementierung erfolgte in Rust, wobei Aufrufe an die NN-Modelle (für das autoregressive Modell oder die AZ-Komponente) bei Bedarf über eine Schnittstelle zu PyTorch abgewickelt werden. Die drei untersuchten Konfigurationen sind:

- **Dynamische Strategie:** Diese kombiniert den AZ-Algorithmus ($\pi_{\text{AlphaZero}}^{\text{PI}}$) mit dem zugehörigen, trainierten autoregressiven Modell ($\phi_{\text{AlphaZero}}^{\text{AR}}$). Für dessen Training wurde der AZ-Agent trainiert und anschließend 100.000 Selbstspiele dieses Agenten zur Generierung der Trainingsdaten für das autoregressive Modell $\phi_{\text{AlphaZero}}^{\text{AR}}$ durchgeführt.
- **Teildynamische Strategie:** Hier wird der MCTS-Algorithmus ($\pi_{\text{MCTS}}^{\text{PI}}$) mit dem trainierten autoregressiven Modell ($\phi_{\text{MCTS}}^{\text{AR}}$) verbunden. Zur Erzeugung der Trainingsdaten für $\phi_{\text{MCTS}}^{\text{AR}}$ wurden 100.000 Selbstspiele mittels einer MCTS-Strategie ($\pi_{\text{MCTS}}^{\text{PI}}$) mit 500.000 Iterationen und $C = 2,5$ durchgeführt.
- **Statische Strategie:** Diese Konfiguration verwendet den MCTS-Algorithmus ($\pi_{\text{MCTS}}^{\text{PI}}$) und den statischen CAP-Algorithmus. Beide Komponenten erfordern kein vorheriges Training.

Für die Evaluation der teildynamischen Strategie mit unterschiedlichen Iterationszahlen I des MCTS wurde der Einfachheit halber ein einziges autoregressives Modell ($\phi_{\text{MCTS}}^{\text{AR}}$) verwendet. Dieses wurde, wie oben beschrieben, auf Daten trainiert, die von einer MCTS-Strategie mit $I = 500.000$ und $C = 2,5$ generiert wurden. Obwohl theoretisch für jede Iterationszahl I ein separates autoregressives Modell optimal wäre, wird angenommen, dass dieses generalisierte Modell auch für höhere Iterationszahlen hinreichend genaue Zustandsvorhersagen liefert. Die Qualität der Determinisierung mit einem autoregressiven Modell hängt nicht nur von der exakten Genauigkeit der Glaubensverteilung ab, sondern auch von der Fähigkeit, eine diverse Menge plausibler Zustände für eine gegebene Informationsmenge zu generieren. Bei allen Anwendungen des autoregressiven Modells wurde die in Abschnitt 5.1.1.6 beschriebene Maskierung verwendet, um die Konsistenz der generierten Zustände zu erhöhen. Die drei resultierenden Gesamtstrategien wurden jeweils mit unterschiedlichen Methoden zur Strategie-Fusion evaluiert.

6.6 WEB-ANWENDUNG ZUR SELEKTIVEN EVALUATION

Zur stichprobenartigen Überprüfung der erzeugten Strategien wurde im Rahmen dieser Arbeit eine Web-Anwendung entwickelt. Diese ermöglicht es, einem menschlichen Spieler, gegen die entwickelten Strategien Doppelkopf zu spielen sowie die Strategien im Selbstspiel gegeneinander antreten zu lassen.



Abbildung 6.3: Web-Anwendung zur selektiven Evaluation.

Über diese Grundfunktionalität hinaus bietet die Anwendung verschiedene Möglichkeiten zur Visualisierung von Zwischenergebnissen der Strategien:

- Visualisierung der durch die Zustandsgenerierung erzeugten Zustände,
 - Visualisierung der Ergebnisse von MCTS und AZ, beispielsweise der Besuchszahlen und der geschätzten Nutzenwerte,
 - Visualisierung der vom autoregressiven Modell vorhergesagten Wahrscheinlichkeiten.

Die Web-Anwendung wurde unter Verwendung des etablierten React-Frameworks in TypeScript entwickelt und kommuniziert über HTTP mit der hybriden Python/Rust-Anwendung. Die Benutzeroberfläche ist in Abbildung 6.3 dargestellt.

EVALUATION

Die Evaluation des Determinisierungsansatzes erfolgt schrittweise. Da die Leistung des Determinisierungsansatzes von seinen Einzelkomponenten abhängt, werden diese zunächst separat betrachtet bevor das Gesamtsystem untersucht wird.

1. **MCTS und AZ unter perfekter Information:** Zunächst wird die grundsätzliche Leistungsfähigkeit der Einzelzustandsanalyse-Algorithmen (MCTS, AZ) im vereinfachten Szenario mit perfekter Information (alle Karten und Vorbehalte offen) analysiert.
2. **Autoregressives Modell:** Anschließend wird das Training und die Genauigkeit des autoregressiven Modells zur Zustandsschätzung bewertet.
3. **Determinisierungsansatz:** Abschließend wird die Effektivität der verschiedenen Kombinationen aus Zustandsauswahl, Einzelzustandsanalyse und Strategie-Fusion des vollständigen Determinisierungsansatzes für Doppelkopf mit imperfekter Information evaluiert.

7.1 EVALUATIONSMETRIKEN

Die aussagekräftigste Metrik für den Erfolg eines Algorithmus ist seine Leistung im Spiel gegen menschliche Spieler. Eine solche Evaluierung erfordert jedoch eine große Anzahl freiwilliger Spieler, ein interaktives System zur Datenerfassung und insbesondere einen erheblichen Zeit- und Organisationsaufwand der Teilnehmenden. In Anbetracht dessen stützt sich die Bewertung der Spielstärke in dieser Arbeit auf Referenzstrategien sowie spielspezifische Metriken. Ergänzend wird eine Einschätzung der Spielstärke anhand einzelner Spielverläufe durch den Autor vorgenommen. Zur Bewertung der Strategien dienen daher folgende Metriken, die teilweise auch Einblicke in den Spielstil einer Strategie geben:

- **Durchschnittliche Punktzahl:** Die wichtigste Metrik zur Beurteilung der Spielstärke im direkten Vergleich ist die durchschnittliche Punktzahl. Eine gute Strategie sollte über viele Spiele hinweg eine positive durchschnittliche Punktzahl gegenüber den Konkurrenten erzielen.
- **Gewonnene Ansagen:** Misst den Prozentsatz der gewonnenen Spiele, in denen der Spieler eine „Re“/„Kontra“-Ansage (oder eine niedrigere Absage) gemacht hat. Ein rationaler Spieler macht eine An- oder Absage nur, wenn er sich sicher ist, dass er das Spiel gewinnen wird. Eine verlorene An- oder Absage führt zu einem erheblichen Verlust. Unter per-

fekter Information sollte ein idealer Spieler also eine Quote von 100% erzielen. Abweichungen deuten auf Fehleinschätzungen hin.

- **Gewonnene Soli:** Misst den Prozentsatz der gewonnenen Solo-Spiele. Entscheidet sich ein Spieler für ein Solo, will er es meist auch gewinnen. Die ideale Quote liegt hier jedoch nicht zwingend bei 100%, da ein taktischer kleiner Verlust bei einem Solo manchmal besser sein kann als ein hoher Verlust im Normalspiel. Eine hohe Quote ist dennoch ein Indikator für eine gute Taktik.
- **Ansage-/Solo-Quote:** Gibt an, wie häufig in Bezug auf die Gesamtheit aller gespielten Spiele eine Strategie überhaupt An- oder Absagen macht oder Solo spielt.

7.2 EVALUATIONSVORGEHEN

Für den Vergleich der erzeugten Strategien und die Messung der Metriken wurden verschiedene Evaluationsvorgehen verwendet, die wie folgt durchgeführt wurden:

- **Vergleichsspiel:** Im Vergleichsspiel werden unterschiedliche Strategien im Spiel gegeneinander untersucht. Dabei wird jede Konstellation, in der die vier Spieler mit den Strategien der untersuchten Menge zusammenspielen können, ohne Berücksichtigung der Reihenfolge gespielt. Damit wird dem Einfluss, den die Spielerpositionen auf den Spielverlauf haben können, Rechnung getragen.
- **Selbstspiel:** Im Selbstspiel verwenden alle vier Mitspieler eine einzige Strategie (mit der gleichen Parametrisierung). Die durchschnittliche Punktzahl konvergiert dabei gegen 0. Das Selbstspiel gibt dennoch einen Einblick in den Spielstil mittels der definierten Metriken sowie durch die selektive Analyse einzelner Spiele.

Doppelkopf ist ein Spiel mit Zufallsereignissen und daraus resultierender hoher Punktevarianz. Aus diesem Grund wurde jede durch das Evaluationsvorgehen erzeugte Kombination viele Male ausgeführt, um signifikante Ergebnisse zu erhalten. Die hohe Anzahl ist notwendig, um die erhebliche Varianz der Spielergebnisse in Doppelkopf zu mitteln und robuste Durchschnittswerte zu erhalten.

7.3 MCTS MIT PERFEKTER INFORMATION

Als erster Schritt der Evaluation wird die Leistungsfähigkeit des MCTS-Algorithmus unter der Annahme perfekter Information untersucht.

7.3.1 Ermittlung geeigneter Parameter

Der eingesetzte MCTS-Algorithmus wird durch zwei zentrale Hyperparameter gesteuert: die Anzahl der Iterationen I pro Zug und der Explorationsparameter C für die UCT-Formel. Die Anzahl der Iterationen I wird im Wesentlichen in Abhängigkeit der Rechenkapazität gewählt. Mehr Iterationen führen in der Regel zu besseren Entscheidungen. Der UCT-Parameter C muss spielspezifisch ermittelt werden. Für die nachfolgenden Experimente zur Bestimmung eines geeigneten C wurde I fest auf 100.000 gesetzt, da dieser Wert bereits hoch ist und eine verhältnismäßig schnelle Ausführung ermöglicht.

7.3.1.1 Experimentelles Vorgehen

Um einen geeigneten UCT-Parameter C zu finden, trat eine Parametermenge im Vergleichsspiel jeweils mit $I = 100.000$ an. Für jede sich ergebende Konstellation wurden 750 Spiele durchgeführt. Die Wahl der C -Werte fiel auf leicht handhabbare Zahlenwerte, anstatt komplexe Wurzelausdrücke zu verwenden, da der optimale Wert ohnehin experimentell ermittelt werden muss.

7.3.1.2 Versuch 1: Breite Streuung

Im ersten Durchgang wurden die UCT-Parameter $C \in \{2.5, 7.5, 12.5, 17.5\}$ untersucht. Die initiale Überlegung hinter dieser breiten Streuung war die große Punktespannweite in Doppelkopf, die von -21 bis $+63$ reicht. Es wurde vermutet, dass diese im Vergleich zu Spielen mit einer $[0,1]$ -Skala, für die oft $C = \sqrt{2}$ verwendet wird (vgl. Abschnitt 4.1.2), einen höheren Explorationswert erfordert oder zumindest sinnvoll macht. Die Ergebnisse sind in Tabelle 7.1 dargestellt.

Tabelle 7.1: Metriken des MCTS im Vergleichsspiel mit unterschiedlichen UCT-Werten bei $I = 100.000$ (I).

Strategie (C)	Ansage-Quote	Ansage-Gewinn	Solo-Quote	Solo-Gewinn	$\bar{\Omega}$ -Punkte
$C = 2,5$	35,08%	72,45%	5,74%	31,24%	+0,31 ($\pm 5,90$)
$C = 7,5$	33,24%	80,94%	9,35%	23,47%	+0,16 ($\pm 6,29$)
$C = 12,5$	30,67%	78,33%	9,78%	20,18%	-0,14 ($\pm 6,44$)
$C = 17,5$	31,03%	76,55%	10,80%	18,92%	-0,33 ($\pm 6,64$)

7.3.1.3 Versuch 2: Engere Auswahl

Da die Werte $C = 2,5$ und $C = 7,5$ im ersten Versuch tendenziell besser abschnitten, wurde dieser Bereich in einem zweiten Durchgang mit den Werten $C \in \{1,5, 2,5, 5,0, 7,5\}$ genauer untersucht. Tabelle 7.2 zeigt die Resultate.

Basierend auf diesen beiden Versuchen scheint ein Wert für C zwischen 2,5 und 7,5 bei $I = 100.000$ Iterationen den besten Kompromiss hinsichtlich der Durchschnittspunktzahl zu bieten, auch wenn andere Werte bei spezifischen

Tabelle 7.2: Metriken des MCTS im Vergleichsspiel mit unterschiedlichen UCT-Werten bei $I = 100.000$ (II).

Strategie (C)	Ansage-Quote	Ansage-Gewinn	Solo-Quote	Solo-Gewinn	$\bar{\Omega}$ -Punkte
C = 1,5	37,09%	65,45%	3,55%	38,86%	-0,10 ($\pm 6,16$)
C = 2,5	37,29%	71,23%	5,95%	31,09%	+0,11 ($\pm 6,10$)
C = 5,0	34,74%	80,22%	8,26%	27,76%	+0,10 ($\pm 6,34$)
C = 7,5	33,10%	78,62%	9,33%	25,11%	-0,11 ($\pm 6,54$)

Metriken teilweise besser abschneiden. Diese Werte werden daher für die weiteren Untersuchungen als Referenzwert verwendet. Es ist jedoch zu beachten, dass der optimale Explorationsparameter C von der Anzahl der Iterationen I abhängen kann: Eine höhere oder niedrigere Anzahl von Iterationen könnte potenziell von einem anderen C-Wert profitieren.

7.3.2 MCTS gegen Zufallsspieler

Um eine grundlegende Bewertung der Spielstärke des MCTS zu erhalten, wurde dieser gegen eine einfache Referenzstrategie getestet: einen Zufallsspieler. Die Zufallsstrategie wählt jede zulässige Aktion mit gleicher Wahrscheinlichkeit. In Anbetracht der zuvor betrachteten Probleme der wiederholten Entscheidungspunkte bei An- und Absagen (siehe Abschnitte 5.2.1.1 und 5.2.1.2) wurde die Zufallsstrategie dahingehend modifiziert, dass sie keine An- oder Absagen tätigt. Andernfalls würde sie nahezu immer „schwarz“ absagen und nahezu immer verlieren. Um überhaupt als erfolgreich gelten zu können, muss der MCTS diesem einfachen Gegner deutlich überlegen sein.

Für diese Untersuchung wurde für jede Iterationszahl einer Menge von aufsteigenden Iterationszahlen und mit C = 2,5 ein Vergleichsspiel mit der modifizierten Zufallsstrategie ausgeführt. Für jede Kombination an Strategien wurden 2.000 Spiele gespielt. Tabelle 7.3 zeigt neben den anderen Metriken die durchschnittliche Punktzahl, die der MCTS-Spieler gegen den Zufallsspieler in Abhängigkeit der Iterationszahl erzielte.

Tabelle 7.3: Metriken des MCTS mit unterschiedlichen Iterationswerten bei C = 2,5

Strategie (I)	Ansage-Quote	Ansage-Gewinn	Solo-Quote	Solo-Gewinn	$\bar{\Omega}$ -Punkte	Partien
I = 10	89,93 %	52,74 %	13,38 %	46,82 %	+0,85 ($\pm 8,68$)	6.000
I = 20	84,73 %	71,16 %	2,70 %	72,22 %	+4,02 ($\pm 7,02$)	6.000
I = 100	79,98 %	89,96 %	2,63 %	76,58 %	+6,30 ($\pm 5,22$)	6.000
I = 200	77,88 %	91,61 %	3,65 %	72,15 %	+6,35 ($\pm 5,30$)	6.000
I = 1.000	77,80 %	92,33 %	4,17 %	69,60 %	+6,43 ($\pm 5,17$)	6.000
I = 10.000	76,78 %	93,99 %	4,40 %	73,86 %	+6,52 ($\pm 4,82$)	6.000
I = 100.000	76,73 %	94,18 %	3,53 %	75,47 %	+6,62 ($\pm 4,57$)	6.000

Aus den Ergebnissen lassen sich folgende Beobachtungen ableiten:

- Die modifizierte Zufallsstrategie erweist sich erwartungsgemäß als sehr schwach. Ihre Tendenz, fast immer Sonderspiele zu wählen, führt zu unrealistischen Spielverläufen und Ergebnissen.
- Der MCTS-Spieler ist dem Zufallsspieler bereits bei sehr geringen Iterationszahlen deutlich überlegen.
- Die Ergebnisse weisen eine hohe Streuung auf. Dies liegt zum einen am inhärenten Zufallsfaktor von Doppelkopf (Kartenverteilung) und wird durch die irrationale Spielweise des Zufallsspielers noch verstärkt. Die hohe Anzahl simulierter Spiele ist daher notwendig, um stabile Durchschnittswerte zu ermitteln.
- Die Leistung des MCTS gegen den Zufallsspieler verbessert sich mit steigender Iterationszahl, erreicht aber bereits bei etwa 100 Iterationen ein hohes Plateau. Dies ist höchstwahrscheinlich auf die begrenzte Herausforderung durch den schwachen Zufallsgegner zurückzuführen.

Zusammenfassend bestätigt dieses Experiment die prinzipielle Funktionsfähigkeit des MCTS für Doppelkopf, da er den Zufallsspieler klar schlägt. Die Aussagekraft für die Spielstärke in realistischeren Szenarien ist jedoch begrenzt. Der modifizierte Zufallsspieler ist eine sehr schwache Referenzstrategie, die zu atypischen Spielverläufen führt.

7.3.3 MCTS gegen MCTS

Die vorangegangenen Analysen des MCTS betrachteten Iterationszahlen bis zu 100.000. Wie die Auswertung der Metriken andeutete, ist die Strategie bei diesen Iterationszahlen von optimaler Spielweise noch weit entfernt. Dieser Abschnitt untersucht daher, wie sich die relative Spielstärke des MCTS sowie die anderen Spielmetriken bei steigenden Iterationszahlen entwickeln.

Zur Ermittlung der relativen Leistungssteigerung wurden Vergleichsspiele durchgeführt: Eine Strategie mit einer höheren Iterationszahl trat jeweils gegen eine Strategie mit der jeweils niedrigeren Iterationszahl aus einer vordefinierten Menge an. Dies ermöglicht eine direkte Messung des Punktevorteils, der durch die Erhöhung der Iterationszahl erzielt wird. Für jede der entstehenden Strategiekonstellationen wurden 2.000 Partien simuliert. Die Strategie mit 10 Iterationen spielte gegen den (modifizierten) Zufallsspieler. Jede Strategie wurde mit $C = 2,5$ parametrisiert. Die Entwicklung dieser relativen durchschnittlichen Punktzahl zeigt Abbildung 7.1.

Im Vergleichsspiel wurde die durchschnittliche Punktzahl der verschiedenen $C \in \{2,5, 3,75, 5,0\}$ für unterschiedliche Iterationszahlen ermittelt. Die sich ergebenden Werte sind in Tabelle 7.2 dargestellt.

Um die Entwicklung spezifischer Metriken zu analysieren, wurden für jede Iterationsstufe Selbstspiele durchgeführt. Dabei spielten alle vier Agenten mit identischer MCTS-Strategie (gleiche Iterationszahl, gleicher C-Parameter) gegeneinander (jeweils 3.000 Partien). Der Verlauf der Solo-Gewinnquote und der Ansage-Gewinnquote für die Parameter $C \in \{2,5, 3,75, 5,0\}$ ist in den

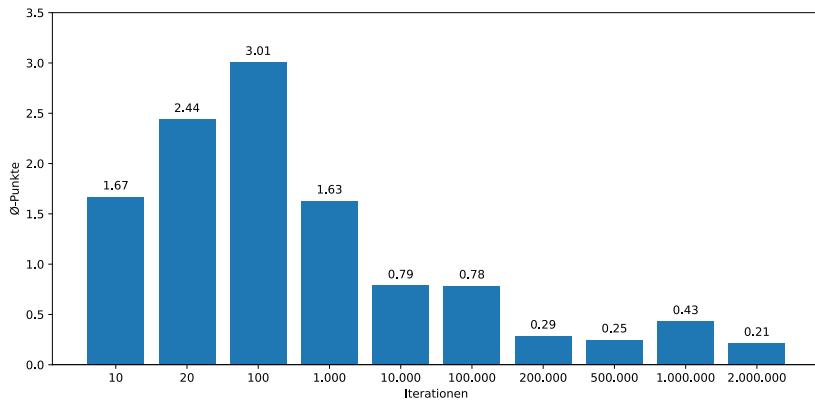


Abbildung 7.1: Durchschnittliche Punktzahl der MCTS-Strategie gegen die jeweilige MCTS-Strategie mit weniger Iterationen ($C = 2.5$).

Abbildungen 7.3 und 7.4 visualisiert. Einen tieferen Einblick in das Spielverhalten liefert Abbildung 7.5, welche die Verteilung der gewählten Spielmodi und der getätigten An-und Absagen der Parteien im Selbstspiel darstellt.

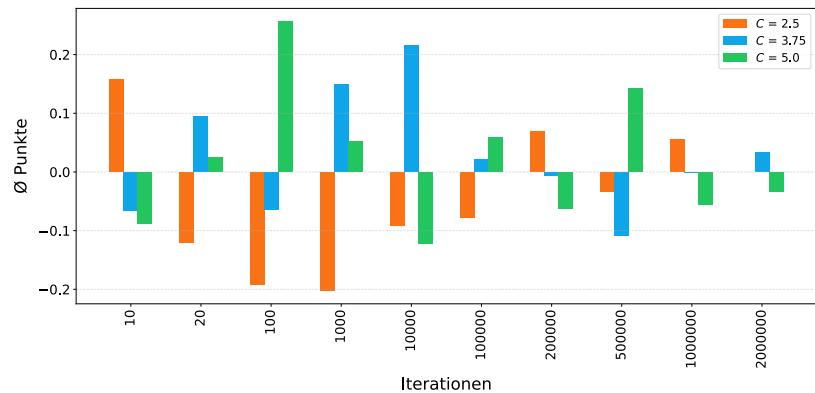


Abbildung 7.2: Die Durchschnittspunktzahl in Vergleichsspielen zwischen $C \in \{2.5, 3.75, 5.0\}$ für verschiedene Iterationszahlen. (volle Ergebnisse: Tabelle A.1)

Aus diesen Experimenten lassen sich folgende Beobachtungen ableiten:

7.3.3.1 Leistungssteigerung mit Iterationen

Jede Erhöhung der Iterationszahl führte zu einer Verbesserung der Strategie sowohl hinsichtlich der Soli- und Ansagenquote als auch der durchschnittlichen Punktzahl. Es ist anzunehmen, dass weitere Steigerungen zusätzliche Verbesserungen bewirken. Allerdings scheinen die Zuwächse an Spielstärke im höheren Bereich nur gering auszufallen.

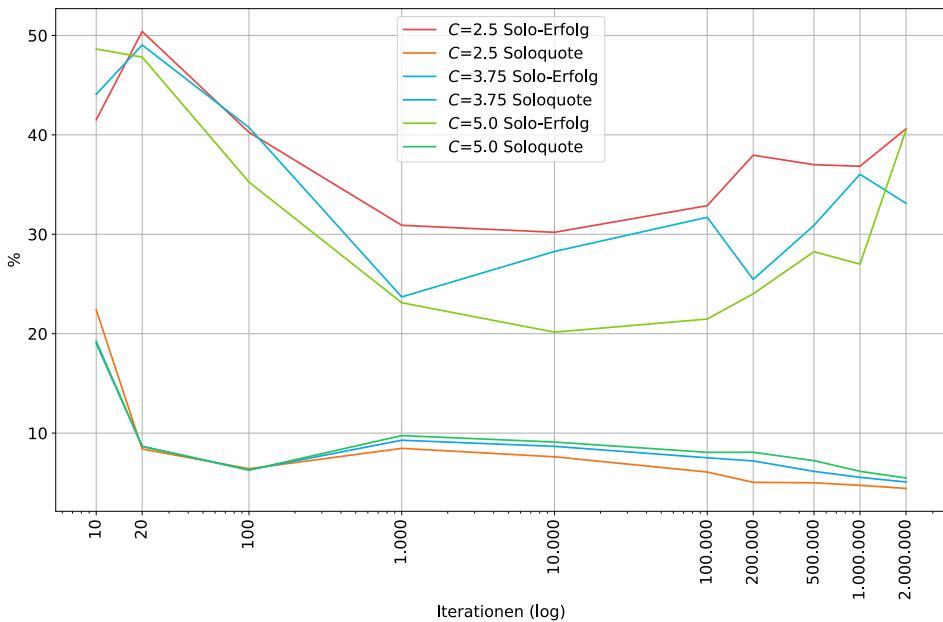


Abbildung 7.3: Entwicklung der Solo-Quoten im Selbstspiel des MCTS im Verlauf zunehmender Iterationsanzahl.

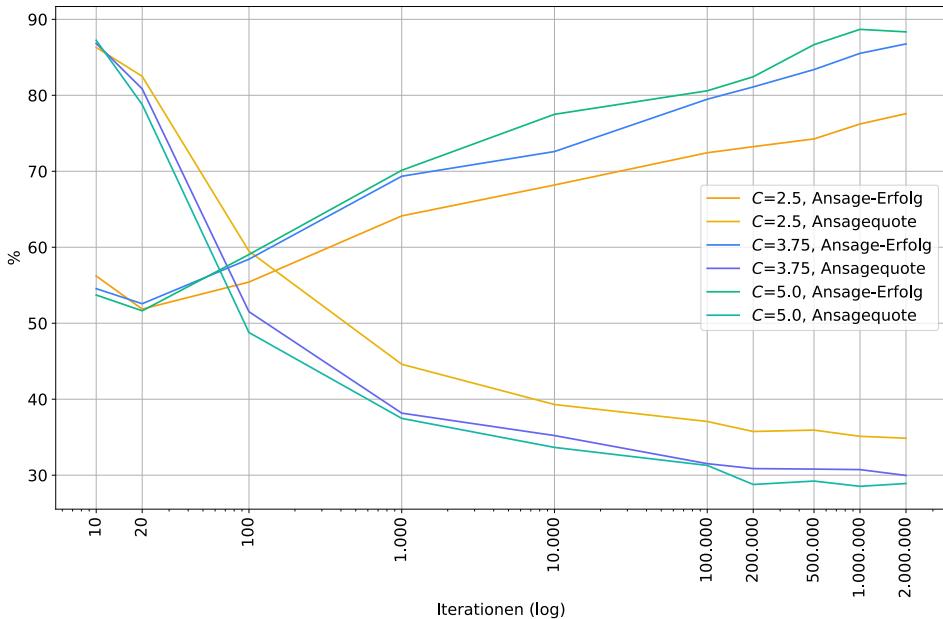


Abbildung 7.4: Entwicklung der Ansage-Quoten im Selbstspiel des MCTS im Verlauf zunehmender Iterationszahlen.

7.3.3.2 Einfluss des UCT-Parameters

Die Analyse der verschiedenen UCT-Parameter C zeigt Unterschiede bei den spezifischen Metriken für Ansagen und Soli. Der Einfluss auf die erreichte

Durchschnittspunktzahl ist jedoch gering. In Anbetracht dessen wurde für weitere Untersuchungen $C = 2,5$ verwendet.

7.3.3.3 Spielmodi-Verteilung

Die Verteilung der gespielten Spielmodi im Selbstspiel zeigt, dass der MCTS bei niedrigen Iterationszahlen noch sehr zufällig agiert. Mit zunehmender Iterationszahl nähert sich die Verteilung einem plausiblen Muster an, das auf eine relativ gute Spielstärke hinweist.

- Soli werden selten gespielt. Sie benötigen meist starke und damit seltene Blätter.
- Das Fleischlos-Solo tritt im Vergleich zu anderen Soli relativ häufig auf. Dies ist plausibel, da es oft eine Alternative zu schwachen Normalspielen darstellt.
- Das \diamond -Solo kommt weniger häufig vor als andere Farbsoli. Oft ist ein gutes Blatt für ein \diamond -Solo auch im Normalspiel sehr stark und das Normalspiel profitabler.
- Das Q-Solo wird seltener als ein J-Solo gewählt. Ein Blatt mit vielen Damen gewinnt ein Normalspiel meist besser als ein Q-Solo.
- Ein stilles Solo wird nahezu nie gespielt. Durch eine Hochzeit werden meist bessere Punktzahlen erreicht.

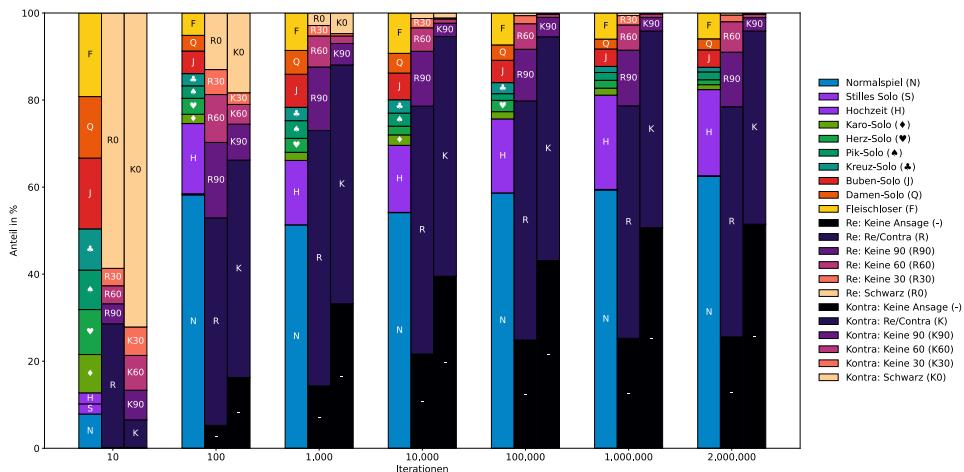


Abbildung 7.5: Verteilung der Spielmodi und An- und Absagen der Parteien im Selbstspiel mit verschiedenen Iterationen ($C = 2,5$).

7.3.3.4 Verteilung der An- und Absagen

Die Verteilung der getätigten An- und Absagen (Abbildung 7.5) ist ebenfalls eingeschränkt plausibel:

- Die Re-Partei tatigt deutlich haufiger risikoreiche Ansagen. Dies ist erwartbar, da sie im Normalspiel durch den Besitz der ♣Q-Damen bereits im Vorteil ist und Solo-Spiele ohnehin nur mit starker Hand gespielt werden.
- Absagen auf eine niedrige Punktzahl erfordern ein sehr starkes Blatt und kommen daher selten vor.

Problematisch ist jedoch das gleichzeitige Auftreten von „Re“- und „Kontra“-Ansagen in Selbstspiel-Partien. Unter perfekter Information konnte eine optimale Strategie den Spieldausgang exakt vorhersagen. Daher ware zu erwarten, dass sich die Haufigkeit von „Re“- und „Kontra“-Ansagen bei optimalem Spiel gegen sich selbst jeweils bei 50 % eingpendelt. Bedauerlicherweise zeigt sich auch mit steigender Iterationszahl kaum eine Verbesserung dieser Metrik.

7.3.4 Analyse einzelner Spiele

Spiel 1				Spiel 2			
Links	Oben	Rechts	Unten	Links	Oben	Rechts	Unten
Re	Kontra	Re	Kontra	Re	Kontra	Kontra	Kontra
Gesund	Gesund	Gesund	Gesund	Fleischlos	♦-Solo	♥-Solo	Q-Solo
R: Re	L: R90						
1	♣A	♣K	♣9	♣10			
		R: R60			♥A	♥J	♥A
2	♥9	♥9	♥A	♥K	♥9	♥10	♥10
3	♣A	♣K	♣10	♣10	♣K	♣K	♣A
4	♦J	♦9	♦A	♦K	♦A	♦9	♦9
5	♦10	♦K	♦Q	♦J	♦10	♦J	♦J
6	♥10	♦9	♥K	♥A	♥9	♥Q	♥Q
7	♥10	♦A	♦J	♥Q	♦9	♦Q	♦K
8	♣Q	♣J	♦10	♥J	♦10	♦J	♦A
9	♣J	♦9	♦Q	♦A	♦Q	♦K	♦A
10	♣Q	♦J	♦Q	♦A	♦A	♦Q	♦J
11	♣K	♦9	♦Q	♣10	♦A	♦K	♦9
12	♣9	♥J	♥Q	♦K	♦10	♦Q	♦J
	129	0	111	0	117	30	23
	10	-10	10	-10	-3	1	1

Abbildung 7.6: Zwei Spielverlaufe im Selbstspiel MCTS mit $I = 1.000.000$ und $C = 2,5$.

Zur weiteren Untersuchung der Beobachtungen und zur Einschatzung der Spielstarke werden im Folgenden zwei exemplarische Spielverlaufe analysiert. In beiden Fallen folgen alle vier Spieler der MCTS-Strategie mit 1.000.000 Iterationen und $C = 2,5$.

SPIEL 1: STARKES RE-BLATT Abbildung 7.6 (links) zeigt eine Partie, in der die Re-Spieler (*Links* und *Rechts*) gemeinsam ein starkes Blatt halten (unter

anderem beide ♠10 und sieben der acht Damen). Beide Spieler nutzen dieses starke Blatt und gewinnen letztendlich alle Stiche. Im Spielverlauf sind prinzipiell starke taktische Züge und im Doppelkopf etablierte Strategien erkennbar. Beispielsweise spielen sich die Partner in den Stichen 2, 3 und 6 gegenseitig Karten zu. Innerhalb der Stiche folgen sie der Taktik „Kleiner Trumpf, Großer Trumpf“, bei der ein kleiner Trumpf angespielt wird, um die Gegenspieler zum Einsatz höherer Trümpfe (und dessen Verlust) zu zwingen.

Aktion (Karte)	N(s, a)	Q(s, a)
K♣	115	6.713
J♣	138	6.717
10♦	56	6.411
9♥	998716	8.062
10♥	303	6.977
Q♠	115	6.687
J♠	29	5.862
Q♣	194	6.866
9♣	49	6.347
A♣	285	6.944

Aktion	N(s, a)	Q(s, a)
Unter 30	16	6.938
Keine An/Absage	999.984	9.352

(a) Wurzelaktionen für die Entscheidung der *unter 30*-Ansage.

(b) Wurzelaktionen im zweiten Zug, erste Karte.

Abbildung 7.7: Die Verteilung der Besuche in zwei Zügen des ersten Spiels.

In Kenntnis der perfekten Information könnte das Ergebnis jedoch noch deutlich besser ausfallen. Ein optimal agierender Re-Spieler würde hier sowohl die Absage „unter 30“ als auch „schwarz“ tätigen, da das Spiel bei optimaler Strategie für die Re-Partei unverlierbar ist und alle Stiche gewonnen werden können. Die Tabelle 7.7a verdeutlicht ein Problem: Für die Entscheidung, ob eine „unter 30“-Ansage getätigter werden soll, wurden fast alle der 1.000.000 Iterationen auf die Untersuchung der Alternative (keine An- oder Absage) verwendet. Die optimale, aber risikoreichere Ansage erhielt nur 16 Besuche. Im Gegensatz dazu zeigt Tabelle 7.7b für die nachfolgende Kartenphase eine deutlich diversere Verteilung der Besuche und damit eine bessere Exploration der Alternativen.

Diese Analyse stützt die bisherige Beobachtung, dass der MCTS zwar taktisch solide Karten spielt, aber bei risikoreichen Entscheidungen wie An- und Absagen oder Soli aufgrund unzureichender Exploration oft suboptimale Entscheidungen trifft.

SPIEL 2: KNAPPES SOLO Abbildung 7.6 (rechts) zeigt einen zweiten Spielverlauf, in dem Spieler *Unten* ein Fleischlos-Solo spielt. Tabelle 7.4 offenbart die MCTS-Statistiken für diese Vorbehaltentscheidung. Die sehr ähnlichen $Q(s, a)$ -Werte für „Gesund“ (-1,091) und „Fleischloser“ (-1,088) deuten darauf hin, dass der Algorithmus das Solo nicht wegen des Gewinns wählt, sondern weil er es als geringfügig bessere Alternative zu einem ohnehin stattfindenden Verlust im Normalspiel einschätzt.

Aktion	N(s, a)	Q(s, a)
♡-Solo	2	-10,500
◊-Solo	2	-9,000
♠-Solo	2	-10,500
♣-Solo	1	-12,000
Gesund	396.778	-1,091
Q-Solo	6	-5,500
J-Solo	5	-5,400
Fleischloser	603.204	-1,088

Tabelle 7.4: Die erste Entscheidung von Spieler *Unten* zugunsten eines Solos.

Auffällig ist die geringe Exploration der Solo-Optionen: Bei 1.000.000 Gesamtiterationen erhielten die meisten Solo-Alternativen nur einstellige Besuchszahlen. Dies erscheint für eine realistische Nutzenschätzung zu wenig. Die geringen Solo-Gewinnquoten scheinen daher aus mangelnder Explorations- und der Möglichkeit des geringeren Verlusts zu resultieren.

7.4 ALPHAZERO MIT PERFEKTER INFORMATION

Nach der eingehenden Analyse des MCTS-Algorithmus unter perfekter Information wird nun der AZ-Algorithmus unter denselben Bedingungen betrachtet. Ziel ist es, den Trainingsprozess und die resultierende Spielstärke dieses lernbasierten Ansatzes zu evaluieren.

7.4.1 *Trainingsprozess und Evaluation*

Der AZ-Agent wurde mit den in Abschnitt 6.3.4 spezifizierten Hyperparametern trainiert. Um den Lernfortschritt zu verfolgen, wurde die Leistung des Agenten nach jeder Trainingsiteration kontinuierlich evaluiert. Als Referenzstrategie für diese laufende Bewertung diente die zuvor analysierte MCTS-Strategie mit $I = 1.000$ und $C = 2,5$. Diese relativ geringe Iterationszahl für den MCTS-Gegner wurde gewählt, um den Rechenaufwand während des AZ-Trainings zu minimieren. Der AZ-Agent selbst verwendete während dieser Evaluationen für seine Entscheidungen 200 Iterationen.

Abbildung 7.9 zeigt die Entwicklung der Durchschnittspunktzahl im Spiel gegen den MCTS. Trainingsiteration 0 repräsentiert dabei den Zustand mit untrainierten Funktionsapproximatoren. Zusätzlich zur vollen AZ-Strategie (mit 200 MCTS-Iterationen pro Zug) wurde auch die Leistung bewertet, wenn nur die direkte Ausgabe des Strategiekopfes (ohne MCTS-Suche) zur Aktionswahl verwendet wird.

LEISTUNGSENTWICKLUNG Die Entwicklung der durchschnittlichen Punktzahl des AZ-Agenten gegen die Referenzstrategie (Abbildung 7.9) demonstriert die prinzipielle Funktionsfähigkeit des Trainingsprozesses. Beginnend mit einem untrainierten Netzwerk (Trainingsiteration 0), das erwartungsge-

¹ Aufgrund von Fehlannahmen wurden die Ausgaben des Wertennetzwerks mit dem Faktor $\frac{1}{12}$ auf einen kleinen Wert skaliert. Der Verlust ist entsprechend niedriger.

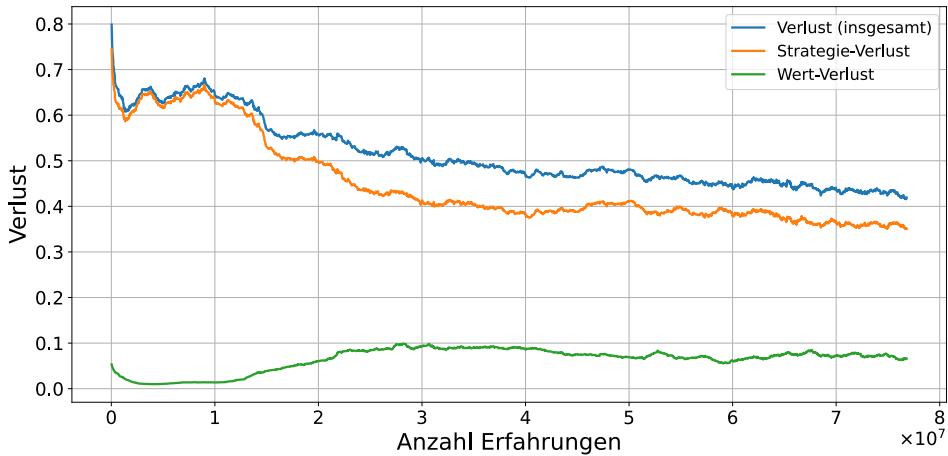


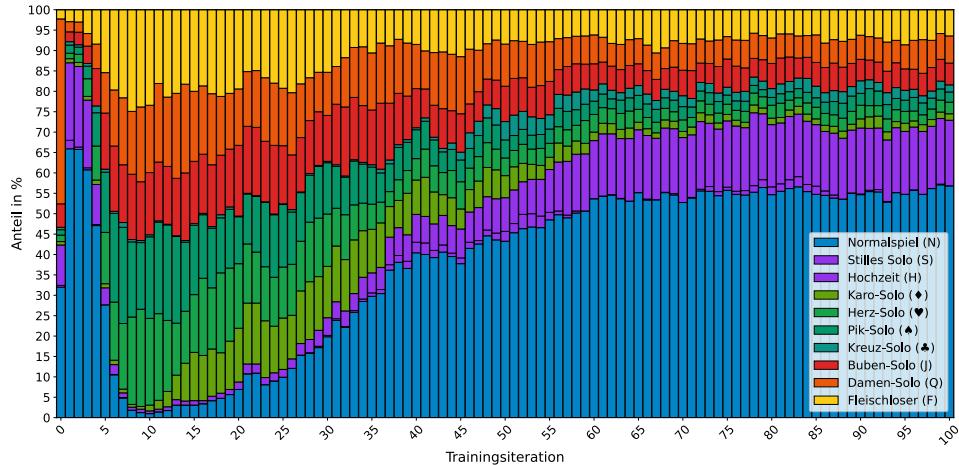
Abbildung 7.8: Der geglättete Verlust des gemeinsamen Strategie- und Wertenetwerks des AZ-Agenten.¹



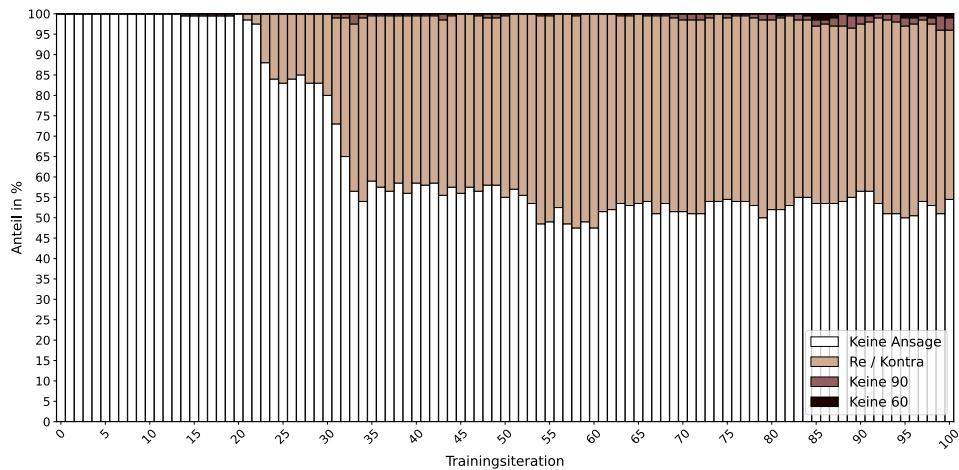
Abbildung 7.9: Durchschnittliche Punktzahl des AZ-Agenten im Trainingsverlauf gegen einen MCTS-Spieler mit 1.000 Iterationen.

mäß unterlegen ist, steigert der AZ-Agent seine Leistung kontinuierlich. Nach etwa 75 Trainingsiterationen ist der AZ-Agent mit nur 200 Iterationen in der Lage, eine höhere durchschnittliche Punktzahl als der MCTS-Agent mit 1.000 Iterationen zu erzielen. Die Leistung der reinen Strategieausgabe des NN (ohne MCTS-Suche) verbessert sich ebenfalls, bleibt aber erwartungsgemäß hinter der des vollständigen AZ-Agenten zurück.

VERLUSTENTWICKLUNG Die Verlustkurven des NN (Abbildung 7.8) zeigen den Lernprozess. Der Verlust des Strategiekopfes sinkt tendenziell, was auf präzisere Aktionsvorhersagen hindeutet. Der Wertverlust sinkt zunächst, gefolgt von einem leichten Anstieg und Fluktuationen. Dies ist typisch für RL-Prozesse, bei denen sich die Trainingsdaten durch die sich verbessernde Strategie des Agenten ständig ändern. Insbesondere das zunehmende Spielen von



(a) Spielmodi-Verteilung des AZ-Spielers im Trainingsverlauf gegen einen MCTS-Spieler mit 1.000 Iterationen und $C = 2,5$.



(b) An- und Absagen des AZ-Spielers im Trainingsverlauf im Spiel gegen einen MCTS-Spieler mit 1.000 Iterationen und $C = 2,5$.

Abbildung 7.10: Verteilung von Spielmodi sowie An- und Absagen des AZ-Spielers gegen einen MCTS-Spieler.

risikoreicheren Aktionen wie An- und Absagen führt hier zu einer höheren Varianz der Spielergebnisse und auch zu höheren Verlustwerten.

ANSAGEN-HEURISTIK Die schrittweise Entwicklung der Spielstrategie des AZ-Agenten lässt sich anhand der Verteilung der gewählten Spielmodi (Abbildung 7.10a) und der getätigten An- und Absagen (Abbildung 7.10b) nachvollziehen. Die Heuristik, An- und Absagen erst ab der 10. Trainingsiteration zuzulassen (vgl. Abschnitt 5.2.2.2), zeigt ihre Wirkung: Der Agent integriert diese risikoreicheren Aktionen erst ab etwa der 25. Trainingsiteration graduell in seine Strategie.

STRATEGIEENTWICKLUNG Die Entwicklung der Spielstrategie des AZ-Agenten lässt sich gut an der Verteilung der gewählten Spielmodi (Abbildung 7.10a) und der gemachten An- und Absagen (Abbildung 7.10b) erkennen. Der gesamte Lernprozess der Strategie durchläuft interessante Phasen:

- Zuerst spielt der Agent sehr konservativ (Normalspiel statt Solo). Durch vermeidet er hohe Verluste durch schlecht gespielte Soli und schneidet zu Beginn gegen die MCTS-Referenzstrategie passabel ab.
- Im weiteren Verlauf erkennt der Agent die Profitabilität von Soli, was zu vielen Soli führt. Gegen die MCTS-Strategie ist dieser Ansatz deutlich unterlegen.
- Durch verbessertes Spiel lernt der Agent, seine eigene vorherige Solo-Strategie zu kontern. Das Normalspiel wird wieder häufiger. Parallel beginnt er, risikoreichere Ansagen (Re und Kontra) zu integrieren.
- Gegen Ende dieses Trainingslaufs entsteht eine Verteilung der Spielmodi, die ein vergleichbar gutes Spiel erkennen lässt und der Verteilung der Spielmodi des MCTS nahekommt (vgl. Abbildung 7.5).

Diese Progression demonstriert den Lernmechanismus von AZ: Es erfolgt eine kontinuierliche Verbesserung durch Selbstspiel, wobei der Agent stets lernt, seine vorherige Strategie zu schlagen. Idealerweise ermöglichen die gewählten Trainingsparameter, diesen selbstverstärkenden Zyklus bis zum Erreichen einer nahezu optimalen Strategie beizubehalten.

FEHLENTWICKLUNG Gegen Ende des Trainingslaufs zeigte sich eine problematische Tendenz: Im Selbstspiel neigten beide Parteien dazu, in jedem Spiel „Re“ und „Kontra“ anzusagen. Dies scheint eine Folge des Problems der wiederholten Entscheidungspunkte und einer daraus resultierenden zu hohen Exploration zu sein. Diese Fehlentwicklung birgt das Risiko des katastrophalen Vergessens: Wenn Spiele ohne An- und Absagen im Training nicht mehr vorkommen, verlernt der Agent irgendwann, solche Zustände richtig zu bewerten.

7.4.2 AlphaZero gegen MCTS

Die Tabelle 7.5 zeigt die durchschnittlich erreichte Punktzahl des AZ-Agenten im Vergleich zur MCTS-Strategie, die hierbei mit $C = 2,5$ und unterschiedlichen Iterationszahlen zum Einsatz kam. Die Darstellung schließt zudem einen Vergleich mit einem Zufallsspieler (0 Iterationen) ein. Der AZ-Agent selbst agierte hierbei mit 200 Iterationen und einem c_{puct} von 4.

Die Ergebnisse bestätigen zunächst die grundsätzliche Funktionsfähigkeit des Algorithmus und der zugrundeliegenden Trainingsprozedur. Auch wenn der AZ-Agent gegenüber der MCTS-Strategie mit hohen Iterationszahlen zurückbleibt, ist es bemerkenswert, dass der AZ-Agent mit seinen vergleichsweise geringen 200 Iterationen eine durchschnittliche Punktzahl erzielt, die durch den MCTS erst mit 10.000 Iterationen erreicht wird.

Tabelle 7.5: Durchschnittliche Punktzahl des trainierten AZ-Agenten im Vergleichsspiel mit MCTS-Strategien mit unterschiedlicher Iterationszahl und $C = 2,5$.

Strategie (I)	\bar{O} -Punkte	Partien
Zufallsspieler	+4,91	6.000
$I = 10$	+3,58	6.000
$I = 100$	+1,21	6.000
$I = 1.000$	+0,50	6.000
$I = 10.000$	+0,03	6.000
$I = 100.000$	-0,42	6.000

7.5 TRAINING DES AUTOREGRESSIVEN MODELLS

Dieses Kapitel beschreibt den Trainingsprozess des autoregressiven Modells und evaluiert dessen Fähigkeit, konsistente und plausible Zustände zu generieren.

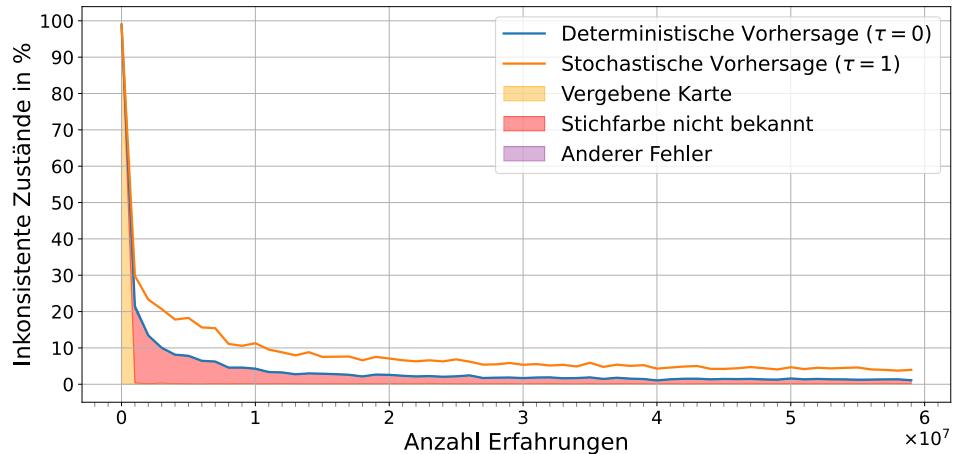
7.5.1 *Trainingsprozess und Evaluation*

Wie in Abschnitt 6.4 beschrieben, wurden zwei separate autoregressive Modelle trainiert. Eines basiert auf Spieldaten, die mit der MCTS-Strategie ($\pi_{\text{MCTS}}^{\text{PI}}$) erzeugt wurden, das andere auf Daten der AZ-Strategie ($\pi_{\text{AlphaZero}}^{\text{PI}}$).

Die Qualität der trainierten Modelle wurde innerhalb des Trainings ausschließlich anhand der Konsistenz der erzeugten Zustände untersucht. Die Zustände wurden auf Basis von Informationsmengen generiert, die durch Selbstspiel mit der jeweiligen Referenzstrategie erzeugt wurden. Gemessen wurde die Anzahl der konsistenten Zustände in Relation zu den insgesamt erzeugten Zuständen. Die Abbildungen A.3a und 7.11a visualisieren den Trainingsfortschritt beider Modelle. Dargestellt sind die Entwicklung des geglätteten Verlusts (CE-Verlust) und exemplarisch für das auf MCTS-Daten trainierte Modell die Häufigkeit verschiedener Inkonsistenztypen.

KONSISTENZ BEI DETERMINISTISCHER VORHERSAGE Bei deterministischer Vorhersage, das heißt bei der Auswahl der jeweils wahrscheinlichsten Karte, generieren die trainierten Modelle eine sehr hohe Rate konsistenter Zustände (über 99 %). Dies deutet darauf hin, dass die Modelle die grundlegenden Spielregeln und Invarianten erfolgreich gelernt haben. Beim stochastischen Sampling entsprechend der ausgegebenen Wahrscheinlichkeitsverteilung (mit $\tau = 1$) liegt die Quote noch bei über 95 %.

TRAININGSPROGRESSION Die Trainingsprogression zeigt, dass das Modell grundlegende Invarianten, wie dass keine Karte mehr als zweimal vorkommen kann, schnell erlernt. Die Invariante, dass ein Spieler eine bediente Farbe nicht mehr auf der Hand haben darf, wird erst später erlernt und macht am Ende noch den größten Vorhersagefehler aus. Andere Konsistenzverlet-



(a) Verhältnis konsistenter Zustände zu inkonsistenten Zuständen.

Abbildung 7.11: Verhältnis konsistenter zu inkonsistenten Zuständen im Verlauf des Trainings des autoregressiven Modells.

zungen treten insgesamt selten auf, wurden aber auch im Trainingsverlauf unwahrscheinlicher.

7.5.2 Evaluation einzelner Vorhersagen

Über die reine Konsistenz hinaus wurde die Fähigkeit des Modells zur Vorhersage der Glaubensverteilung entsprechender Kartenverteilungen anhand einzelner Spielverläufe evaluiert.

7.5.2.1 Spielverlauf 1: Farbenspiel ohne Ass

In einem ersten Spielverlauf wurde von allen Spielern das Normalspiel gewählt und keinerlei An- und Absagen gemacht. Der Startspieler *Unten* hat im ersten Stich bisher nur eine ♠9 ausgespielt. In der Regel bedeutet ein Anspiel mit einer Fehlfarbe, die kein Ass ist, dass der betreffende Spieler kein anderes Ass in dieser Fehlfarbe auf der Hand hat. Hätte er noch ein Ass, hätte er dieses stattdessen ausgespielt, um damit zu versuchen, den Stich zu gewinnen². Zusätzlich erfolgt ein Anspiel einer Karte in Fehlfarbe in der Regel nur durch einen Kontra-Spieler ohne ♣Q. Für einen Re-Spieler mit ♣Q ist es meist lohnenswerter, Trumpf anzuspielen.

Nun soll die Informationsmenge aus Sicht des nachfolgenden Spielers *Links* untersucht werden. Dieser hat das folgende Blatt:

$$h_{\text{Links}} = \{\clubsuit Q, \spadesuit Q, \clubsuit J, \spadesuit J, \spadesuit J, \heartsuit J, \diamondsuit J, \diamondsuit 9, \clubsuit A, \spadesuit K, \heartsuit A, \diamondsuit 9\}$$

Zur Untersuchung wurde das Modell nun 1.000 Mal (mit Maskierung und $\tau = 1,0$) ausgeführt, um den wahrscheinlichen Zustand des Spielers *Unten* zu

² Das $\diamond A$ ist kein Farbass, sondern ein Trumpf-Ass.

ermitteln. Im Rahmen dieser Ausführungen enthielt das prognostizierte Blatt von Spieler *Unten* die folgenden Karten mit der entsprechenden Wahrscheinlichkeit mindestens einmal:

$$h'_{\text{Unten}} = \{(\heartsuit 10, 53.0\%), (\clubsuit Q, 7.7\%), (\spadesuit Q, 40.4\%), (\diamondsuit Q, 67.4\%), (\diamondsuit Q, 61.2\%), \\ (\clubsuit J, 41.8\%), (\spadesuit J, 0.0\%), (\heartsuit J, 34.8\%), (\diamondsuit J, 38.5\%), \\ (\diamondsuit A, 69.0\%), (\diamondsuit 10, 66.4\%), (\diamondsuit K, 62.2\%), (\diamondsuit 9, 38.6\%), \\ (\clubsuit A, 4.0\%), (\clubsuit 10, 61.8\%), (\clubsuit K, 68.3\%), (\clubsuit 9, 63.1\%), \\ (\spadesuit A, 3.4\%), (\spadesuit 10, 13.8\%), (\spadesuit K, 4.2\%), (\spadesuit 9, 11.6\%), \\ (\heartsuit A, 16.8\%), (\heartsuit K, 61.6\%), (\heartsuit 9, 47.2\%)\}$$

Es wird deutlich erkennbar, dass das Modell einige Karten dem Spieler *Unten* deutlich seltener zuordnet. Dabei ist die Erwartung prinzipiell erfüllt:

- Die $\clubsuit Q$ wird dem Spieler vergleichsweise selten zugeordnet.
- Die Farb-Asse ($\clubsuit A$, $\spadesuit A$, $\heartsuit A$) sowie weitere Pik-Karten werden dem Spieler vergleichsweise selten zugeordnet.

Insofern stützt diese Auswertung, dass das Modell eine Glaubensverteilung entsprechend der gespielten Strategien gelernt hat.

7.5.2.2 Spielverlauf 2: Ansage eines Solos

In einem zweiten Spiel hat Spieler *Links* ein J-Solo angesagt. Ansonsten sind keinerlei Karten gelegt. Die Erwartung hier besteht darin, dass dem Spieler *Links* aus der Sicht der anderen Spieler mit einer hohen Wahrscheinlichkeit Buben und Asse zugeordnet werden. Diese beiden Kartentypen sind notwendig, um ein J-Solo üblicherweise zu gewinnen.

Spieler *Unten* hat das folgende Blatt:

$$h_{\text{Unten}} = \{\clubsuit K, \clubsuit K, \clubsuit Q, \spadesuit K, \spadesuit Q, \heartsuit 10, \heartsuit Q, \heartsuit Q, \heartsuit 9, \diamondsuit 10, \diamondsuit K, \diamondsuit 9\}$$

Die folgenden Werte für das prognostizierte Blatt von Spieler *Links* ergeben sich aus dem gleichen Verfahren wie im vorherigen Beispiel:

$$h'_{\text{Links}} = \{(\clubsuit J, 92.5\%), (\spadesuit J, 86.9\%), (\heartsuit J, 79.8\%), (\diamondsuit J, 70.6\%), \\ (\clubsuit A, 66.7\%), (\clubsuit Q, 4.8\%), (\clubsuit 10, 49.3\%), (\clubsuit K, 0.0\%), (\clubsuit 9, 37.1\%), \\ (\spadesuit A, 67.3\%), (\spadesuit Q, 17.1\%), (\spadesuit 10, 49.9\%), (\spadesuit K, 22.1\%), (\spadesuit 9, 42.8\%), \\ (\heartsuit A, 63.6\%), (\heartsuit Q, 0.0\%), (\heartsuit 10, 17.4\%), (\heartsuit K, 46.8\%), (\heartsuit 9, 21.0\%), \\ (\diamondsuit A, 76.2\%), (\diamondsuit Q, 37.2\%), (\diamondsuit 10, 22.1\%), (\diamondsuit K, 22.1\%), (\diamondsuit 9, 16.8\%)\}$$

Das Ergebnis ist erwartungsgemäß und stützt eine sinnvolle Modellierung der Glaubensverteilung.

7.6 IMPERFEKTE INFORMATION

In diesem Abschnitt erfolgt die Zusammenführung und Evaluation der zuvor untersuchten Komponenten zu vollständigen Strategien für Spiele mit imperfekter Information.

7.6.1 Vergleich verschiedener Determinisierungsansätze

Für die Untersuchung des durch die Determinisierung erzeugten Gesamtsystems wurden in einer ersten Analyse sechs verschiedene Kombinationen aus Zustandsauswahlverfahren, Strategie-Fusionsmethoden und Einzelzustandsanalyse-Algorithmen im Vergleichsspiel evaluiert. Alle MCTS-Strategien wurden dabei mit 200.000 Iterationen und $C = 2,5$ parametrisiert; AZ mit den bekannten Parametern. Für alle Strategien wurden $n_{\text{sample}} = 5$ Zustände aus der Informationsmenge gewählt. Falls das autoregressive Modell keine 5 konsistenten Zustände generieren konnte, wurde eine zufällige Aktion gewählt. Tabelle 7.6 zeigt die Ergebnisse dieser Untersuchung.

Tabelle 7.6: Vergleich der vollständigen Determinisierungsstrategien mit unterschiedlicher Parametrisierung.

Strategie	Solo		Ansage		$\bar{\emptyset}$ -Punkte	Konsistenz		% Zufall
	%	% Gewinn	%	% Gewinn		% Konsistent	% Inkonsistent	
$\pi_{\text{AlphaZero}^{\text{II}}}^{\text{AR}}$ (avg)	5,81%	75,57%	53,81%	48,09%	+0,09	91,54%	8,46%	0,64%
$\pi_{\text{AlphaZero}^{\text{II}}}^{\text{AR}}$ (rank)	5,38%	73,85%	55,55%	46,73%	-0,15	91,81%	8,19%	0,63%
$\pi_{\text{MCTS}^{\text{II}}}^{\text{AR}}$ (avg)	3,34%	72,93%	33,43%	67,62%	+0,71	93,45%	6,55%	0,45%
$\pi_{\text{MCTS}^{\text{II}}}^{\text{AR}}$ (rank)	3,49%	69,11%	31,26%	62,99%	+0,09	93,09%	6,91%	0,47%
$\pi_{\text{MCTS}^{\text{II}}}^{\text{II}}$ (avg)	7,50%	48,76%	53,55%	53,69%	-0,23	100,00%	0,00%	0,00%
$\pi_{\text{MCTS}^{\text{II}}}^{\text{II}}$ (rank)	7,97%	42,02%	53,35%	53,54%	-0,51	100,00%	0,00%	0,00%

FUSIONSSTRATEGIEN Die Fusion mittels der durchschnittlichen Strategie führte durchweg zu besseren Ergebnissen.

ZUSTANDSAUSWAHL Der regelbasierte CAP-Algorithmus führte zu einer geringeren Durchschnittspunktzahl als das autoregressive Modell. Dies ist plausibel, da das autoregressive Modell die Glaubensverteilung explizit berücksichtigt.

SOLI UND AN- UND ABSAGEN Die Häufigkeit gespielter Soli und getätigter Ansagen verringerte sich im Vergleich zum Spiel unter perfekter Information.

INKONSISTENTE ZUSTÄNDE Trotz Maskierung der Ausgaben des autoregressiven Modells erzeugte das Modell (aufgrund des Zuweisungsproblems) bis zu etwa 9 % inkonsistenter Zustände.

ALPHAZERO Interessanterweise erzielte der AZ-Agent in Kombination mit dem autoregressiven Modell eine höhere Durchschnittspunktzahl als die

Kombination aus MCTS und dem CAP-Algorithmus. Dies legt die Vermutung nahe, dass die Zustandsauswahl von größerer Bedeutung ist als die Stärke der Einzelzustandsanalyse.

7.6.2 Einfluss der Stichprobengröße

In zwei weiterführenden Untersuchungen wurde der Einfluss der Stichprobengröße sowie deren Wechselwirkung mit der verwendeten Iterationszahl I auf die Spielstärke evaluiert. Tabelle 7.7 zeigt den kombinierten Effekt bei der Parameter. Dabei wurden mehrere Kombinationen von Stichprobengrößen und Iterationszahlen in Vergleichsspielen auf die erreichte durchschnittliche Punktzahl überprüft. Tabelle 7.7b zeigt aufbauend auf diesen Ergebnissen eine noch größere Bandbreite an Stichprobengrößen n_{sample} mit einer konstanten Iterationszahl von 200.000 im Vergleichsspiel mit der jeweils niedrigeren Stichprobengröße.

Iterationen I	5	10	15	Stichproben n_{sample}	Vorteil ($\bar{\Omega}$ -Punkte)
100.000	-0,28	-0,11	0,19	5 vs 10	0,25
200.000	-0,10	-0,10	0,19	10 vs 15	0,11
500.000	0,03	-0,09	0,22	15 vs 25	0,17
				25 vs 50	0,08

(a) Durchschnittliche Punktzahl in Abhängigkeit von Iterationszahl I und Stichprobengröße n_{sample} .

(b) Höhere Durchschnittspunktzahl der höheren Stichprobengröße im Vergleichsspiel mit der kleineren.

Tabelle 7.7: Untersuchung der Strategie $\pi_{\text{MCTS}^{\text{II}}}^{\text{AR}}$ ($\text{avg}, C = 2,5$) hinsichtlich der Stichprobengröße.

Die Daten legen nahe, dass der Einfluss der Iterationszahl auf das Ergebnis mit steigender Stichprobengröße abnimmt. Zwar ist auch bei größeren Stichprobengrößen tendenziell noch eine leichte Verbesserung durch mehr Iterationen zu beobachten, jedoch ist der Zugewinn geringer als bei kleinen Stichproben. Eine ausreichend große Stichprobengröße scheint somit einen höheren Einfluss auf eine gute Durchschnittspunktzahl zu haben als eine hohe Iterationszahl. Der Effekt zusätzlicher Stichproben nimmt jedoch bei bereits großen Stichprobengrößen ebenfalls ab.

7.6.3 Entwicklung im Selbstspiel

Im Rahmen einer finalen Untersuchung wurde die entwickelte Determinierungsstrategie, basierend auf MCTS mit einer Stichprobengröße von $n_{\text{sample}} = 25$, im Selbstspiel evaluiert. Tabelle 7.8 zeigt die sich aus diesen Spielen ergebenden Metriken in Abhängigkeit der Iterationsanzahl. Abbildung 7.12 zeigt die Verteilung der Spielmodi sowie An- und Absagen.

SPIELMODI-VERTEILUNG Die Verteilung der Spielmodi unter imperfekter Information erscheint weitestgehend plausibel. Im Vergleich zum Spiel mit

Tabelle 7.8: Ergebnisse der Strategie $\pi_{\text{MCTS}^{\text{II}}}^{\text{AR}}$ ($\text{avg}, C = 2,5, n_{\text{sample}} = 25$) im Selbstspiel.

Iterationen	Solo		Ansage		\emptyset -Punkte	Partien	Konsistenz		% Zufall
	%	% Gewinn	%	% Gewinn			% Konsistent	% Inkonsistent	
10	9.01%	58.13%	36.08%	57.23%	0.00	6 000	82.33%	17.67%	2.08%
100	3.47%	53.50%	32.11%	64.92%	0.00	6 000	95.77%	4.23%	0.18%
1 000	5.62%	50.81%	31.59%	63.37%	0.00	6 000	96.34%	3.66%	0.13%
10 000	5.22%	47.41%	31.91%	64.46%	0.00	6 000	96.14%	3.86%	0.14%
100 000	4.35%	48.70%	32.22%	64.57%	0.00	6 000	96.27%	3.73%	0.15%
200 000	3.51%	54.09%	32.71%	66.55%	0.00	6 000	96.62%	3.38%	0.11%
500 000	3.46%	54.49%	33.43%	65.56%	0.00	6 000	96.49%	3.51%	0.13%
1 000 000	3.26%	54.79%	33.17%	65.75%	0.00	6 000	96.02%	3.98%	0.18%

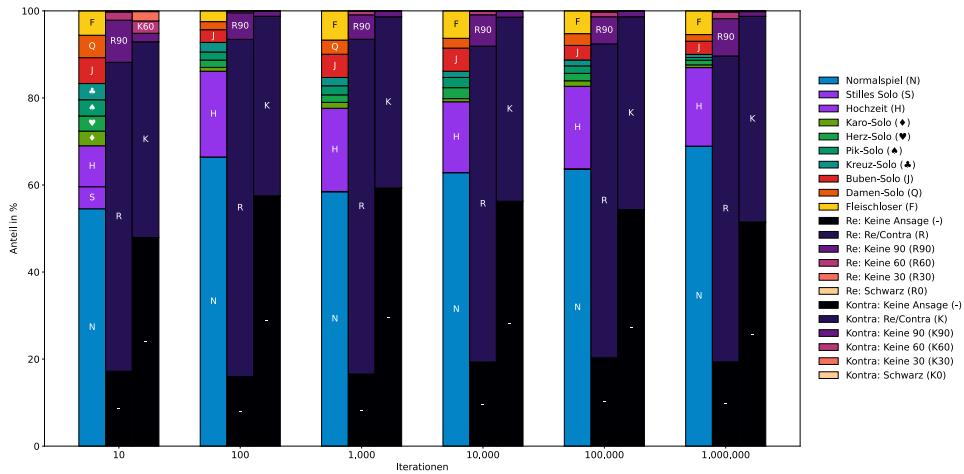


Abbildung 7.12: Die Verteilung der Spielmodi und An- und Absagen bei Selbstspiel Strategie $\pi_{\text{MCTS}^{\text{II}}}^{\text{AR}}$ ($\text{avg}, C = 2,5, n_{\text{sample}} = 25$).

perfekter Information nimmt die Anzahl der gespielten Soli erwartungsgemäß ab. Das generelle Verhältnis der verschiedenen Spielmodi zueinander erscheint auch unter imperfekter Information plausibel: Das Fleischlos-Solo und das J-Solo werden weiterhin am häufigsten gespielt. Diese Soli hängen primär vom eigenen Blatt ab und sind daher auch unter Unsicherheit bezüglich der gegnerischen Karten vergleichsweise gut einzuschätzen. Die Reduktion der Häufigkeit anderer Soloarten, insbesondere des Q-Solos und der Farbsoli, ist ebenfalls nachvollziehbar: Unter imperfekter Information können die Blätter der Mitspieler kaum eingeschätzt werden, wodurch das Risiko eines solchen Solospiele, das stärker von der Kartenverteilung bei den Gegnern abhängt, steigt.

AN-UND ABSAGENVERTEILUNG Im Vergleich zum Spiel mit perfekter Information gibt es weniger Absagen. Auch das ist prinzipiell erwartbar, da solche Absagen bei Unsicherheit über den Spielzustand risikoreicher werden.

Das zuvor identifizierte Problem der übermäßigen und oft falschen „Re“- und „Kontra“-Ansagen bleibt jedoch bestehen. Prinzipiell wäre zu erwarten, dass die Unsicherheit durch imperfekte Information zu einem deutlicheren

Rückgang der „Re“-und „Kontra“-Ansagen führt als beobachtet wurde. Die geringe Erfolgsrate dieser An- und Absagen über alle Iterationszahlen hinweg (vgl. Tabelle 7.8) unterstreicht das Problem.

REGULARISIERENDE WIRKUNG Die Determinisierung scheint eine regularisierende Wirkung auf die Strategieentwicklung zu haben. Im Vergleich zu den Ergebnissen unter perfekter Information, insbesondere bei niedrigeren Iterationszahlen des MCTS, ergibt sich durch die Determinisierung eine insgesamt plausiblere Verteilung der Spielmodi und auch der Absagen.

7.6.4 Analyse einzelner Spiele

Abbildung 7.13 zeigt einen Spielverlauf aus den gleichen Ausgangspositionen wie im Kapitel 7.3.4 zur perfekten Information sowie einen weiteren. In diesem Fall wurden sie mit der Determinisierungsstrategie $I = 1.000.000$, $C = 2,5$ und einer Stichprobengröße von 25 ausgeführt.

Spiel 1				Spiel 2			
Links	Oben	Rechts	Unten	Links	Oben	Rechts	Unten
Re	Kontra	Re	Kontra	Re	Re	Kontra	Kontra
Gesund	Gesund	Gesund	Gesund	Gesund	Gesund	Gesund	Gesund
R: Re				B: K			
1	♦A	♦K	♦9	♦10	♦A	♦9	♦9
					♦A	♦A	♦A
2	♥10	♦9	♦A	♦J	♦10	♦9	♦A
3	♦A	♦J	♦Q	♦10	♦10	♦9	♦A
4	♦9	♦9	♦A	♦K	♦9	♦J	♦10
5	♦Q	♦9	♦K	♦A	♦Q	♦K	♦J
6	♦10	♦A	♦Q	♦Q	♦10	♦A	♦A
7	♦10	♦9	♦10	♦10	♦9	♦K	♦10
8	♦K	♦J	♦Q	♦A	♦J	♦Q	♦J
9	♦J	♦K	♦Q	♦J	♦Q	♦K	♦J
10	♦Q	♦9	♦J	♦K	♦10	♦A	♦Q
11	♦9	♦J	♦Q	♦A	♦10	♦J	♦K
12	♦J	♦K	♦10	♦K	♦9	♦K	♦10
	125	0	88	27	83	27	101
	7	-7	7	-7	-6	-6	6

Abbildung 7.13: Zwei Spielverläufe der Strategie $\pi_{\text{MCTS}^{\text{II}}}^{\text{AR}}$ (avg , $C = 2,5$, $n_{\text{sample}} = 25$, $I = 1.000.000$).

7.6.4.1 Spiel 1: Starkes Re-Blatt

Innerhalb des Spiels lassen sich weiterhin einige gute Taktiken beziehungsweise Züge erkennen: In Stich 2 zeigt der Spieler *Links* seinem Partner beispielsweise, dass er als „Re“ beide ♠10 hat. Sein Mitspieler *Rechts* zeigt ihm mit dem ♦A, dass er sein Mitspieler ist. Dann folgt das gegenseitige Anspielen nach der Taktik „Kleiner Trumpf, Großer Trumpf“ beziehungsweise mit der jeweiligen Farbe, die der Partner nicht hat.

Problematisch ist jedoch der sechste Zug: Ein guter Spieler hätte an der Stelle von *Rechts* einen deutlich höheren Trumpf nehmen müssen, um nicht von *Unten* überstochen werden zu können. *Rechts* und *Links* hätten das Spiel bei optimaler Spielweise mit allen Stichen „schwarz“ gewinnen können. Letztendlich hat das autoregressive Modell in einer Mehrzahl von Zuständen die $\heartsuit Q$ nicht bei Spieler *Unten* verortet, weswegen die Einzelzustandsanalyse für die generierten Zustände die $\diamondsuit Q$ als ausreichend bewertete und diese dann in der Strategie-Fusion dominierte.

Zusätzlich hätten *Links* und *Rechts* alle Absagen bis zu „schwarz“ tätigen können. Von einem optimalen Spieler wäre zu erwarten gewesen, dass trotz der imperfekten Information wenigstens eine Absage bis „unter 60“ erfolgt. Sie haben jedoch nur „Re“ angesagt und damit viele Punkte verschenkt. Insofern zeigt die Strategie deutliche Schwächen.

7.6.4.2 *Spiel 2: Ausgeglichenes Spiel*

Spiel 2 zeigt ein Spiel, bei dem die „Kontra“-Partei ein leicht stärkeres Blatt als die „Re“-Partei hat. Insgesamt sind die Züge aller vier Spieler solide, aber nicht überragend. Das Problem der risikoreichen Ansage ist auch in diesem Spiel zu erkennen. Sowohl die „Kontra“-Ansage als auch die „Re“-Ansage sind vertretbar, aber risikoreich.

8

DISKUSSION

In diesem Kapitel werden die Ergebnisse der Evaluation interpretiert, um die Forschungsfrage beantworten zu können. Dazu werden zunächst die erreichte Spielstärke der entwickelten Strategien bewertet, anschließend die aufgetretenen Probleme und Limitationen sowohl der spezifischen Implementierung für Doppelkopf als auch des Determinisierungsansatzes an sich analysiert. Abschließend erfolgt eine Einordnung der Ergebnisse im Kontext der Forschungsfrage.

8.1 SPIELSTÄRKE

Die erzeugten Strategien zeichnen ein gemischtes Bild: Sowohl die durch den MCTS erzeugten Strategien als auch die durch den AZ erzeugten Strategien zeigen eine grundlegende Kompetenz im Spiel Doppelkopf. Sie übertreffen den Zufallsspieler deutlich. Für die Leistung in perfekter Information und in imperfekter Information mittels Determinisierung muss man jedoch unterscheiden:

In perfekter Information zeigten die Algorithmen insbesondere beim Ausspielen der richtigen Karten in der Kartenphase taktisch sinnvolles Spiel. Im Spiel Doppelkopf üblicherweise praktizierte Taktiken wie „Kleiner Trumpf, Großer Trumpf“ sind in den Partien deutlich erkennbar. Zusätzlich zeigt die Verteilung der gewählten Spielmodi und zumindest auch eingeschränkt die der An- und Absagen im Rahmen des Selbstspiels Verhältnisse, die man auch im Spiel mit Menschen erwarten würde. Allerdings erreichen die Strategien kein optimales Niveau. Die größte Schwäche scheint bei dem Treffen risikoreicher Entscheidungen wie der Wahl von Soli und von An- und Absagen zu liegen. Das äußert sich daran, dass die Solo-Gewinnquoten selbst bei hohen Iterationszahlen niedrig blieben und die Ansage-Gewinnquoten gleichermaßen niemals die theoretisch unter perfekter Information möglichen 100 % erreichen. Zusätzlich gibt es eine Vielzahl von Fällen, in denen die Strategie im Selbstspiel sowohl zu „Re“- als auch „Kontra“-Ansagen führt, was zumindest für eine Partei immer eine Fehlentscheidung ist. Die Analyse einzelner Spielverläufe bestätigte diese Problematik und wies zusätzlich darauf hin, dass nicht nur risikoreiche Züge überproportional gewählt werden, sondern darüber hinaus auch zahlreiche risikoreiche, aber vorteilhafte Züge nicht erkannt wurden.

Die Evaluation unter imperfekter Information bei Anwendung der Determinisierung offenbarte in Folge genauso Schwächen. Zwar hat die Evaluierung einzelner Spielverläufe durchaus ein akzeptables Spiel, insbesondere wieder in der Kartenphase, gezeigt. Von einem optimalen Spiel waren diese Spielverläufe aber noch weit entfernt. Positiv ist hervorzuheben, dass die

Solo-Quote sank, während die Erfolgsquote von Soli stieg. Dies legt nahe, dass die Berücksichtigung der Unsicherheit durch die Fusion mehrerer determinierter Zustände zu konservativeren und damit auch erfolgreicheren Solo-Entscheidungen führte. Gleiches ergibt sich für die Absagen („unter 90“, „unter 60“ etc.), nicht jedoch für die Ansagen („Re“, „Kontra“). Es ergab sich weiterhin eine geringe suboptimale An- und Absagegewinnquote, und die Anzahl der Spiele, in denen sowohl „Re“ als auch „Kontra“ angesagt werden, war groß. Bei optimaler Spielweise wäre das nicht so.

Zusammenfassend zeigt die erreichte Spielstärke gute Ansätze, aber definitiv kein optimales Spiel. Mit erfahrenen Doppelkopf-Spielern würden die Strategien sicher nicht mithalten. Die Strategien beherrschen die Kartenphase nach der Vorbehaltspause relativ gut, treffen jedoch insbesondere bei den risikoreichen Entscheidungen (Soli, An-/Absagen) suboptimale Wahlen. Diese Schwächen bei den risikoreichen Entscheidungen sind im Wesentlichen auf zwei Probleme, die mangelnde Exploration im Rahmen der MCTS-Baumsuche sowie das Problem der wiederholten Entscheidungspunkte, zurückzuführen.

8.2 EINZELZUSTANDSANALYSE

Bereits im Rahmen der Einzelzustandsanalyse auf perfekter Information haben sich einige Problematiken herausgestellt, die im Folgenden diskutiert werden.

8.2.1 *Mangelnde Exploration*

Ein zentrales Problem, das die Leistung des MCTS und indirekt auch die des AZ beeinträchtigte, war der schwierige Ausgleich zwischen Exploration und Exploitation bei risikoreichen Entscheidungen. Im Falle des Doppelkopfs wurden bei der Ausführung des MCTS die An- und Absagen sowie die unterschiedlichen Vorbehalte kaum untersucht. Diese mangelhafte Erkundung führt zu zwei Problemen:

- **Rauschen:** Wenige zufällige (un-)glückliche Simulationsergebnisse können (bei geringer Besuchszahl) die Nutzenschätzung $Q(s, a)$ für eine An- und Absage oder ein Solo drastisch verzerrn. Eine riskante Aktion kann durch anfängliches Glück fälschlicherweise als sehr gut bewertet werden, während eine eigentlich starke Option durch Pech als schlecht eingestuft wird.
- **Unzureichende Untersuchung:** Risikoreiche Entscheidungen erfordern für eine akkurate Nutzenschätzung oftmals optimales Spiel. Dieses kann nur bei sehr guter Untersuchung erreicht werden. Bei zu wenigen Besuchen kann eine gute Option vorschnell verworfen werden.

Die Ursache für diese Problematik liegt maßgeblich in der hohen Ergebnisvarianz und der großen Punkteskala von Doppelkopf. Der Gewinn einer An- und Absage führt nur zu einer kleinen Erhöhung der eigenen Punktzahl, der

Verlust aber zum Gewinn des gegnerischen Teams mit der Punktzahl, die man selbst ohne Ansage erreicht hätte (z. B. von +4 auf -4). Dies stellt ein hohes Risiko dar, das für optimales Spiel aber unerlässlich ist. Solo-Spiele erfordern zudem oft präzises Spiel von beiden Seiten, was in rein zufälligen Simulationen bei geringer Iterationszahl nur unzureichend abgebildet wird.

Der naheliegende Ansatz, die Exploration durch Erhöhung des UCT-Parameters C zu steigern, erwies sich in den Untersuchungen nicht als zufriedenstellend. Zwar führt ein höheres C dazu, dass der Algorithmus auch die risikoreichen Optionen häufiger besucht und robuster bewertet (was sich beispielsweise in besseren Ansage-Gewinnquoten niederschlägt). Gleichzeitig wirkt sich die global erhöhte Exploration jedoch negativ auf die häufigeren, aber oft weniger varianzbehafteten Entscheidungen aus, wie die Wahl der zu spielenden Karte. Hier sind die tatsächlichen Nutzenunterschiede oft klein. Ein hoher C -Wert führt dazu, dass unverhältnismäßig viele Besuche auf die Untersuchung offensichtlich schwächerer Kartenalternativen verwendet werden. Der Suchbaum wird für diese Entscheidungen zu breit und flach, und die Exploitation wird nicht genug ausgenutzt. Die Erkenntnis, dass der Ansagen-Erfolg bei höherem C (vgl. Abbildung 7.4) und die Ansagenquote selbst sinken, aber gleichzeitig die durchschnittliche Punktzahl (vgl. Abbildung 7.2) sehr ähnlich bleibt, stützt diese Interpretation.

Das Kernproblem liegt somit in der Starrheit des Selektionsmechanismus UCT mit seinem globalen Explorationsparameter, der der Heterogenität der Entscheidungssituationen in Doppelkopf nicht gerecht wird. Denkbare Lösungsansätze zur Verbesserung der Exploration umfassen erstens spielspezifische Heuristiken, beispielsweise Mindestbesuchszahlen für risikoreiche Aktionen oder eine selektive Anpassung des C -Parameters für solche. Zweitens könnte eine dynamische Skalierung der Nutzenwerte $Q(s, a)$ helfen, etwa mittels Min-Max-Scaling [SB20a], bei dem die in der Situation beobachteten Minimal- und Maximalwerte auf einen festen Bereich normalisiert werden. Drittens wäre die Verwendung von UCT-Varianten, wie zum Beispiel UCB1 [ACBF02], denkbar, die explizit die Varianz der Simulationsergebnisse berücksichtigen. Allgemein anwendbar oder gar besser sind derartige Adaptationen jedoch nicht. Vielmehr kann die Nutzung spielbezogen zu besseren Ergebnissen führen [Świ+23].

Es ist stark anzunehmen, dass diese grundlegende Problematik wesentlich dazu beiträgt, dass der MCTS in dieser Arbeit sehr hohe Iterationszahlen benötigt, um eine passable Leistung zu erzielen und selbst dann noch deutliches Verbesserungspotenzial aufweist. Insbesondere die suboptimalen und teilweise überlappenden An- und Absagen (besonders in Kombination mit den wiederholten Entscheidungspunkten) sowie die niedrige Solo-Gewinnquote lassen sich auf diese unzureichende Exploration zurückführen.

8.2.2 Wiederholte Entscheidungssituationen

Die hier verwendete realitätsgetreue Modellierung von Doppelkopf, bei der An- und Absagen nach jeder gelegten Karte erneut gemacht werden können,

erwies sich als problematisch. Diese wiederholten Entscheidungssituationen führten über alle erzeugten Strategien sowohl des MCTS als auch des AZ und auch bei imperfekter Information zu einem starken Bias zugunsten der Wahl von An- und Absagen.

Im MCTS konnten die gravierendsten Probleme, die aus dieser Modellierung folgen, durch die Einführung von zwei Heuristiken (keine Ansagen in Simulationen, eingeschränkte Ansagen im Selektionsbaum) vermieden werden. Der Bias zugunsten von An- und Absagen blieb bestehen, da an jedem erlaubten Entscheidungspunkt die (geringe) Wahrscheinlichkeit einer Fehleinschätzung durch unglückliche Simulationen erneut zum Tragen kommt. Über die vielen Gelegenheiten im Spielverlauf hinweg akkumuliert sich dieses Risiko. Dies wirkt als verschärfender Faktor neben dem generellen Problem der mangelhaften Exploration und trägt zur beobachteten unbefriedigenden Gewinnquote bei Ansagen bei.

Im AZ-Trainingsprozess führte diese Modellierung zu der beobachteten Fehlentwicklung, bei der die Agenten im Selbstspiel in jeder Partie „Re“ und „Kontra“ ansagten. Die explizite Konditionierung, Ansagen in frühen Trainingsphasen zu unterbinden, verzögerte dieses Problem, konnte es aber nicht verhindern. Es ist anzunehmen, dass dies auch bei anderer Parametrisierung aufgetreten wäre, da ein RL-Algorithmus zur Datengenerierung explorieren muss. Aufgrund der vielfachen Entscheidungsmöglichkeiten wird die Wahrscheinlichkeit, dass eine An- oder Absage durch Exploration irgendwann im Spielverlauf gewählt und somit in den Trainingsdaten überrepräsentiert wird, selbst bei geringen Explorationsraten (Temperatur τ) sehr hoch. Zwar könnten auch hier weitere, spezifische Heuristiken angewendet werden, wie etwa ein deutlich verringelter Temperatur-Parameter nur für An- und Absage-Aktionen im Selbstspiel. Vielversprechender erscheint jedoch eine Anpassung der Doppelkopf-Umgebung selbst (sogenanntes *Environment Shaping* [Kam+20]) zu einer solchen, bei der An- und Absagen für den Agenten nur zum letztmöglichen Zeitpunkt erlaubt sind. Das aufgetretene Problem ist ein Ausdruck des allgemeinen im RL vorherrschenden Explorations-und Exploitationsdilemma.

8.2.3 AlphaZero und MCTS

Ein Blick auf die Evaluationsergebnisse zeigt, dass die hier trainierte AZ-Strategie in direkten Vergleichen hinter der Spielstärke der MCTS-Strategie mit hohen Iterationszahlen zurückblieb. Dies wirft die Frage auf, ob der erhebliche Mehraufwand für das Training und die Implementierung von AZ gerechtfertigt ist.

Zunächst ist festzuhalten, dass der präsentierte AZ-Agent das Resultat eines einzigen, ressourcenbegrenzten Trainingslaufs über nur 100 Trainingsiterationen ist. Wie in Abschnitt 8.2.2 diskutiert, beeinträchtigte eine Fehlentwicklung bei den An- und Absagen das Training, sodass das volle Potenzial des Algorithmus nicht ausgeschöpft werden konnte. Ein weiterer Trainingslauf mit An-

passungen und mehr Rechenkapazitäten hätte voraussichtlich eine deutlich stärkere AZ-Strategie hervorgebracht.

Trotz dieser Einschränkungen deutet sich bereits das Potenzial von AZ an: Selbst dieser suboptimal trainierte Agent erreichte mit lediglich 200 MCTS-Iterationen eine Leistung, die mit der der MCTS-Strategie mit 10.000 Iterationen mithalten konnte. Dieser Effizienzvorteil gewinnt an Bedeutung, wenn man den Blick auf komplexere Spiele richtet. Doppelkopf ist nur moderat komplex. Dennoch erforderte selbst hier ein starker MCTS bereits hohe Iterationszahlen. In Spielen mit exponentiell größeren Zustandsräumen und höheren Verzweigungsfaktoren wird ein reiner Suchansatz wie MCTS schnell unmöglich.

8.3 AUTOREGRESSIVES MODELL

Der Erfolg der Determinisierung hängt entscheidend von der Qualität der ausgewählten Zustände ab. Die Evaluation zeigte hier klare Vorteile für das autoregressive Modell gegenüber dem statischen CAP-Algorithmus. Die Fähigkeit des autoregressiven Modells, eine Glaubensverteilung über mögliche Zustände zu modellieren und somit wahrscheinlichere Zustände bei der Auswahl zu bevorzugen, führte im Vergleich zur rein regelbasierten, tendenziell gleichverteilten Auswahl des CAP-Algorithmus zu besseren Ergebnissen. Die Leistungsfähigkeit des Modells im Rahmen des Determinisierungsansatzes wird jedoch durch folgende Faktoren eingeschränkt:

VERTEILUNGSVERSCHIEBUNG Die Zuverlässigkeit des autoregressiven Modells hängt von seiner Fähigkeit ab, auch auf Situationen zu generalisieren, die von den Trainingsdaten abweichen, und dabei konsistente Zustände zu erzeugen.

Die Evaluation erfolgte ausschließlich mithilfe von Selbstspielen des gleichen Verfahrens mit (ggf. unterschiedlicher) Parametrisierung. Treten im Spiel gegen andere Spieler Spielverläufe auf, die in den Trainingsdaten unterrepräsentiert oder gar nicht vorhanden waren, kann die Vorhersageleistung des NN-basierten Modells nachlassen. Tatsächlich ist dieser Vorgang dem Prinzip der Determinisierung inhärent: Die gesamte Strategie des Determinisierungsprinzips spielt (aufgrund der berücksichtigten Unsicherheiten) anders als die Einzelzustandsanalyse bei perfekter Information. Dieses Problem der Verteilungsverschiebung hat sich auch tatsächlich gezeigt: Trotz Maskierung hat sich der Anteil an inkonsistenten Zuständen gegenüber dem Trainingslauf erhöht.

INKONSISTENZEN TROTZ MASKIERUNG Das autoregressive Modell generiert trotz Maskierung global inkonsistente Zustände. Das Modell trifft lokale Vorhersagen, die nicht zwangsläufig zu einem global gültigen Zustand führen (wegen des Zuweisungsproblems). Eine mögliche Abhilfe könnte darin bestehen, explizite Algorithmen zur Erzeugung gültiger Zuweisungen mit den Wahrscheinlichkeitsvorhersagen des Modells zu kombinieren.

STRATEGIEANNAHME Das autoregressive Modell trifft die Annahme, dass die Strategien der Mitspieler denen der Referenzstrategie ähneln. Prinzipiell könnten Gegner mit stark abweichenden oder gezielt ausbeutenden Strategien diese Annahme unterlaufen. Für Doppelkopf ist anzunehmen, dass starke Abweichungen von einer soliden Grundstrategie selten zum Erfolg führen und die optimale Strategie nur begrenzte Variationen zulässt. Für andere Spiele könnte sich jedoch etwas anderes ergeben.

8.4 ANSATZ DER DETERMINISIERUNG

Nachdem die Einzelleistung der Komponenten analysiert wurde, wird im Folgenden der Determinisierungsansatz als Ganzes diskutiert. Basierend auf den Evaluationsergebnissen und theoretischen Überlegungen sollen Einflussfaktoren und die Anwendung auf andere Spiele mit imperfekter Information erörtert werden.

8.4.1 Stichprobengröße

Die durchgeführten Untersuchungen legen nahe, dass die Anzahl der pro Informationsmenge gezogenen Zustandsstichproben ein entscheidender Parameter für die Leistung der Determinisierung ist. Eine größere Stichprobe führt tendenziell zu robusteren und besseren Entscheidungen.

Zwar ist auch zu berücksichtigen, dass die Probleme der zu geringen Exploration sowie der wiederholten Entscheidungssituationen durch die Mehrfachausführung der Einzelzustandsanalyse bei höherem n_{sample} ebenfalls abgemildert wurden. Die positive Auswirkung einer größeren Stichprobengröße geht jedoch über diese spezifischen Probleme hinaus: Eine größere Anzahl von Stichproben ermöglicht eine bessere Approximation des Nutzens einer Aktion. Mit einer geringen Anzahl von Stichproben hingegen können sich zufällige Ausreißer stärker auf das Ergebnis auswirken. Die resultierende Aktion ist stärker vom Zufall der Stichprobenwahl abhängig, was zu suboptimalen Entscheidungen führen kann. Die Evaluation bestätigte dies: Eine Erhöhung der Stichprobengröße führte bei konstanter Iterationszahl der Einzelzustandsanalyse durchweg zu besseren Durchschnittsergebnissen, teilweise auch dann, wenn sich die Iterationszahl erhöht hatte.

Gleichzeitig steigt jedoch der Rechenaufwand direkt proportional zur Stichprobengröße, da für jede Stichprobe eine separate, aufwändige Baumsuche durchgeführt werden muss. Dieser hohe Rechenaufwand stellt eine wesentliche Limitation der Determinisierung dar. Hinsichtlich der erforderlichen Rechenkapazitäten, um eine sehr hohe Entscheidungsqualität durch viele Stichproben zu erreichen, erscheinen alternative Ansätze, die möglicherweise effizienter mit der Unsicherheit umgehen, vielversprechender.

8.4.2 Strategie-Fusion und Nicht-Lokalität

Der Ansatz der Determinisierung unterliegt nach einer Untersuchung von [FB98] mehreren theoretischen Unzulänglichkeiten, welche die Autoren als *Strategie-Fusion* (Strategy Fusion) und *Nicht-Lokalität* (Non-Locality) bezeichnen [Lon+10].

Die Probleme der Strategie-Fusion beruhen darauf, dass jede Baumsuche so ausgeführt wird, als kenne der Agent den wahren Zustand und könne diesen optimal ausspielen. Treffen dabei stark unterschiedliche, antikorrelierte Ergebnisse für eine Aktion über die verschiedenen determinierten Zustände hinweg zusammen, ist also eine Aktion in einem Zustand gewinnbringend, in einem anderen verlustreich und existiert zugleich eine sichere Alternative mit nicht schlechterem Ergebnis, kann die Fusion der Einzelergebnisse dazu führen, dass die riskanten Aktionen überbewertet werden. Die einzelnen Suchen verschmelzen somit inkompatible Teilstrategien zu einer scheinbar überlegenen Gesamtstrategie. Dadurch wird der tatsächliche Erwartungswert der gewählten Aktion überschätzt, während tatsächlich vorteilhafte Strategien, die die Unsicherheit korrekt berücksichtigen, womöglich unentdeckt bleiben.

Das Problem der Nicht-Lokalität (Non-Locality) resultiert aus einem Unterschied in der Bewertung von Spielzuständen zwischen Spielen mit perfekter und imperfekter Information. In Spielen mit perfekter Information hängt der Wert eines Knotens ausschließlich von den möglichen Ergebnissen ab, die von diesem Knoten aus erreichbar sind. Suchalgorithmen wie der MCTS nutzen diese lokale Eigenschaft aus. In Spielen mit imperfekter Information ist dies jedoch anders: Der Gegner kann seine verborgenen Informationen nutzen, um den Spielverlauf gezielt in Regionen des Spielbaums zu lenken, die für ihn vorteilhaft sind. Diese nicht-lokalen Abhängigkeiten kann die Determinisierung prinzipiell nicht erfassen. Aufbauend auf der Arbeit von [FB98] untersuchten [Lon+10], unter welchen Bedingungen diese Probleme in Spielen auftreten. Um Spiele hinsichtlich dieser Probleme besser einschätzen zu können, identifizierten sie drei Metriken von Spielbäumen:

1. *Leaf Correlation*: Sie ist ein Maß für die Wahrscheinlichkeit, dass direkt benachbarte Zustände im Spielbaum denselben Spieldurchgang haben. Eine hohe Korrelation deutet darauf hin, dass der genaue Ausgang von Spielen, oft unempfindlich gegenüber der exakten Wahl von Zügen ist.
2. *Bias*: Der Bias gibt die Tendenz des Spiels an, einen Spieler (in bestimmten Informationsmengen) zu bevorzugen. Ein hoher oder niedriger Bias lässt auf ganze Teilbäume des Spielbaums schließen, in denen eine Partei dominiert; der Determinisierungsansatz muss also nur die grobe Richtung finden.
3. *Disambiguation Factor*: Er ist ein Maß dafür, wie schnell die Größe der Informationsmenge eines Spielers im Laufe des Spiels durch die Aktionen der Spieler reduziert wird. Je kleiner die Informationsmenge wird, desto weniger fehleranfällig ist der Determinisierungsansatz.

Für die Stichspiele Hearts und Skat ermittelten die Autoren eine hohe Leaf Correlation, einen hohen Bias sowie einen großen Disambiguation Factor und schlossen daraus, dass bei derlei Stichspielen die Determinisierung relativ gute Ergebnisse liefern kann. Als ein Spiel mit ähnlichen Mechaniken besteht die Annahme, dass dies auch für Doppelkopf gilt.

Obwohl davon auszugehen ist, dass beide Probleme einen Einfluss auf die Leistungsfähigkeit der Determinisierung haben, muss auch der Kontext der Arbeiten von [FB98] und [Lon+10] beachtet werden. Dabei wurde ein Determinisierungsansatz betrachtet, bei dem die Glaubensverteilung der Zustände nicht berücksichtigt wurde, und die Zustandsauswahl vielmehr gleichverteilt erfolgte. Es ist anzunehmen, dass die in dieser Arbeit verwendete Modellierung der Glaubensverteilung durch das autoregressive Modell die praktischen Auswirkungen der genannten Unzulänglichkeiten reduziert, indem sie die Analyse auf wahrscheinlichere Szenarien konzentriert.

8.4.3 Gemischte Strategien

Der hier vorgestellte Ansatz der Determinisierung ist für Spiele mit imperfekter Information, die strategische Unvorhersehbarkeit und damit gemischte Strategien erfordern, wie zum Beispiel das Bluffen in Poker, grundsätzlich ungeeignet. Zwar erzeugt das Verfahren durch das zufällige Ziehen von Zustandsstichproben und die zufälligen MC-Simulationen tatsächlich eine gemischte Strategie. Diese ist aber nur aufgrund der Zufälligkeit stochastisch, nicht explizit um strategische Unvorhersehbarkeit zu modellieren. Daher werden MCTS- und AZ-basierte Ansätze in der Literatur für Spiele, die strategische Unvorhersehbarkeit (wie Bluffen) erfordern, als unzureichend angesehen [CWP15]. Sie können nicht garantieren, ein Nash-Gleichgewicht zu finden, welches oft gerade in der optimalen Mischung von Strategien liegt.

Um gute gemischte Strategien zu ermitteln, sind andere algorithmische Ansätze notwendig. Andere Ansätze für solche Spiele sind beispielsweise Abwandlungen des MCTS, welche eine Baumsuche auf Informationsmengen statt Zuständen durchführen. Diese Ansätze sind prinzipiell erfolgreich. Die meisten Algorithmen für derartige Spiele basieren jedoch auf dem sogenannten Counterfactual Regret Minimization (CFR). Bei der Grundform des CFR [Zin+07] handelt es sich um einen Algorithmus, welcher auf dem gesamten Spielbaum operiert und Aktionen unter dem Begriff des „Bedauerns“ betrachtet. Er analysiert die Spielverläufe und Aktionen danach, wie groß der Gewinn gewesen wäre, wenn anstelle einer Aktion eine andere gewählt worden wäre. Dieses Bedauern wird im Zeitverlauf minimiert. Der Algorithmus konvergiert dann zu einem Nash-Gleichgewicht. Rein praktisch ist der gesamte Spielbaum in jedem nichttrivialen Spiel so groß, dass der Basis-Algorithmus nicht angewendet werden kann. Moderne Algorithmen basieren zwar auf dem Prinzip des Basis-Algorithmus, verwenden aber zusätzlich Simulationen (z. B. Monte-Carlo Counterfactual Regret Minimization (MC-CFR) [Lan+09]) oder NN (z. B. DeepNash [Per+22]), um die hohe Komplexität der Spiele zu beherrschen.

8.4.4 *Opponent Modeling*

Der hier vorgestellte Determinisierungsansatz nimmt die gleiche Strategie und Verhaltensweisen für alle Spieler an. Ihm fehlt die Fähigkeit, adäquat auf andere Strategien zu reagieren, und darüber hinaus auch die Fähigkeit, die eigene Spielweise dynamisch auf Strategien der Mitspieler anzupassen. Der Teilbereich des RL, der sich damit beschäftigt, wird als *Opponent Modeling* bezeichnet [GWC24].

8.4.5 *Kommunikation*

In kooperativen Spielen wie Doppelkopf ist Kommunikation, oft indirekt durch Spielzüge, für hohe Spielstärke unerlässlich. Beispielsweise kann der Zeitpunkt einer Ansage oder die Wahl einer bestimmten Karte dem Partner entscheidende Informationen übermitteln und so den gemeinsamen Nutzen maximieren. Solche etablierten Konventionen werden vom vorliegenden Determinisierungsansatz nicht erlernt, da weder MCTS/AZ noch das autoregressive Modell diese subtilen Signale generieren und interpretieren. Einfache Anpassungen der Ansatz, um solche Kommunikation zu erlernen, sind nicht ersichtlich. Das selbstständige Erlernen von Kommunikation in Multi-Agenten-Umgebungen ist Gegenstand aktueller Forschung. Spiele wie Hanabi und Bridge [Bar+20] verdeutlichen diese Herausforderung; für Bridge existieren bislang keine Algorithmen auf menschlichem Expertenniveau [Kit+24].

8.5 ZIELERREICHUNG

Das Ziel dieser Arbeit war es, die Anwendbarkeit der ursprünglich für perfekte Information konzipierten Algorithmen MCTS und AZ auf Spiele mit imperfekter Information zu untersuchen, wobei das Prinzip der Determinisierung am Beispiel des Kartenspiels Doppelkopf exemplarisch eingesetzt wurde.

Trotz der suboptimalen Spielweise der resultierenden Strategien, zeigt die Arbeit, dass der Determinisierungsansatz grundsätzlich anwendbar ist und zur Entwicklung plausibler Spielstrategien für Doppelkopf führt. Insbesondere die Nutzung eines trainierten autoregressiven Modells, das eine angenäherte Glaubensverteilung über mögliche Zustände liefert, erwies sich in den Untersuchungen als vorteilhafter gegenüber statischen Ansätzen wie dem CAP-Algorithmus, der keine Zustandswahrscheinlichkeiten berücksichtigt.

Gleichzeitig offenbart die Untersuchung signifikante konzeptionelle und praktische Einschränkungen des Ansatzes:

- Die grundlegenden Probleme der Determinisierung (Strategie-Fusion, Nicht-Lokalität) sowie ihre inhärent pseudodeterministische Natur verhindern das Konvergieren zu echten gemischten Strategien und somit zu Nash-Gleichgewichten. Dies schränkt die Anwendbarkeit auf Spiele, in denen strategische Unvorhersehbarkeit (z.B. Bluffen) entscheidend ist, ein. Darüber hinaus werden fortgeschrittene Aspekte der

Mehrspieler-Interaktion, wie Opponent Modeling oder die Nutzung impliziter Kommunikation, nicht erfasst. Gerade für das Spiel Doppelkopf sind diese Fähigkeiten für das Spielen auf Expertenniveau unerlässlich.

- Der Determinisierungsansatz erfordert zwingend eine spielspezifische Komponente zur Zustandskonstruktion. Dies stellt eine Abweichung vom Idealzustand, einem Algorithmus der ohne spielspezifische Programmierung und Heuristiken auskommt, dar. Diese Notwendigkeit einer spezifischen Komponente limitiert die direkte Übertragbarkeit auf andere Spiele und erhöht die Komplexität der Implementierung im Vergleich zu anderen Ansätzen.

Über diese Aspekte hinaus zeigte die praktische Umsetzung für Doppelkopf, dass zusätzliche heuristische Anpassungen an den Algorithmen notwendig waren, um mit den spezifischen Herausforderungen des Spiels umzugehen. Die Notwendigkeit solcher Anpassungen ist aus theoretischer Sicht unschön, im RL jedoch weit verbreitet: Ein Ausgleich zwischen Exploration und Exploitation erfordert oft eine feine Abstimmung von Hyperparametern, die Modifikation von Belohnungssignalen (*reward shaping*) oder sogar die Anpassung der Umgebungsmodellierung (*environment shaping*). Der hier untersuchte Determinisierungsansatz bildet in dieser Hinsicht keine Ausnahme.

Zusammenfassend lässt sich für die Zielerreichung festhalten: Die Determinisierung stellt einen konzeptionell einfachen Weg dar, MCTS und AZ auf bestimmte Klassen von Spielen mit imperfekter Information zu erweitern. Sie liefert eine solide Basisstrategie und ermöglicht das Lernen aus Selbstspiel. Die Arbeit zeigt dabei auch klare Grenzen dieses Ansatzes.

8.6 FAZIT

Spiele mit imperfekter Information stellen durch Unsicherheit, vielfältige Gegnerstrategien sowie Aspekte wie Kommunikation und Kooperation eine größere Herausforderung für die Entwicklung effektiver Strategien und Algorithmen dar als Spiele mit perfekter Information. Diese Arbeit leistet einen Beitrag zur Auseinandersetzung mit dieser Thematik, indem sie einen einfachen und für eine bestimmte Klasse von Spielen effektiven Determinisierungsansatz vorstellt. Dieser kann als Impuls für die weitere Erforschung von Strategien in diesem komplexen Problemfeld dienen.

Teil I

APPENDIX

A

ERGÄNZENDE ABBILDUNGEN

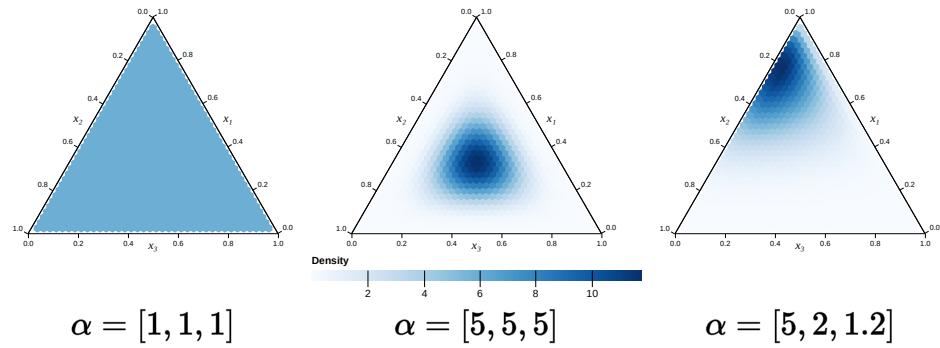


Abbildung A.1: Die Dirichlet-Verteilung von einigen Beispielen [Sus].

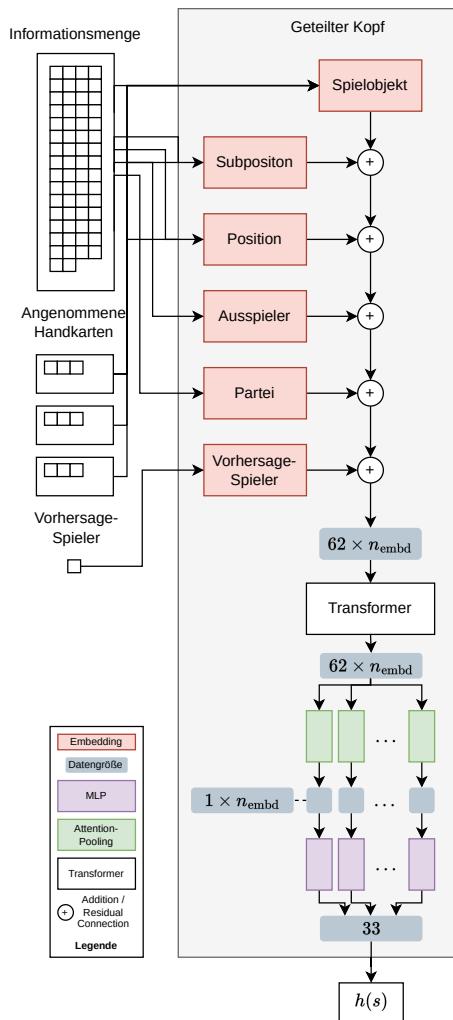
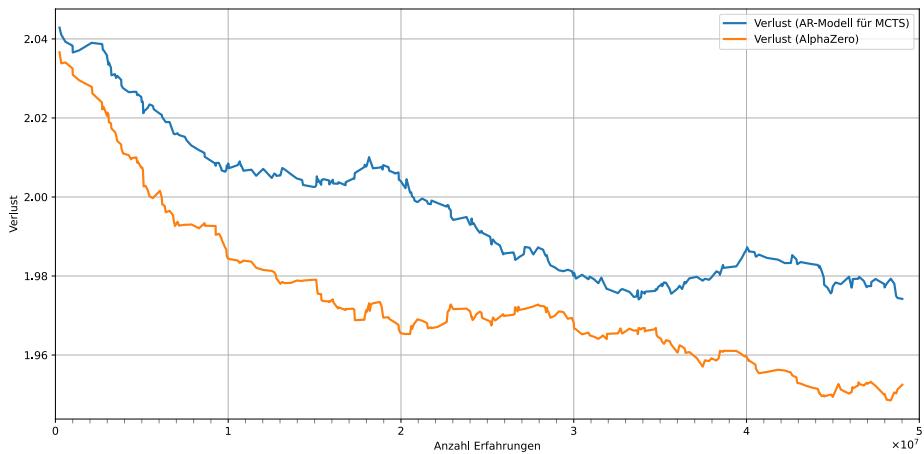


Abbildung A.2: Die Architektur des NN für das autoregressive Modell.



(a) Der geglättete Verlust des autoregressiven Modells für die MCTS-Strategie sowie die AZ-Strategie.

Abbildung A.3: Der geglättete Verlustverlauf des autoregressiven Modells für verschiedene Strategien.

Tabelle A.1: Vergleichsspiele zwischen $C \in \{2.5, 3.75, 5.0\}$ für verschiedene Iterationszahlen.

Strategie (I, C)	Ansage-Quote	Ansage-Gewinn	Solo-Quote	Solo-Gewinn	$\bar{\emptyset}$ -Punkte	Spiele
$I = 10, C = 2.5$	86.48%	54.73%	18.11%	50.00%	+0.16	6 000
$I = 10, C = 3.75$	87.68%	53.02%	18.36%	45.93%	-0.07	6 000
$I = 10, C = 5$	88.53%	53.30%	18.84%	45.62%	-0.09	6 000
$I = 20, C = 2.5$	80.75%	51.89%	8.07%	47.47%	-0.12	6 000
$I = 20, C = 3.75$	81.22%	52.60%	8.08%	53.10%	+0.09	6 000
$I = 20, C = 5$	79.66%	51.58%	8.74%	51.80%	+0.02	6 000
$I = 100, C = 2.5$	55.48%	53.52%	5.57%	37.78%	-0.19	6 000
$I = 100, C = 3.75$	52.60%	57.49%	6.35%	40.38%	-0.06	6 000
$I = 100, C = 5$	52.25%	57.63%	6.83%	38.29%	+0.26	6 000
$I = 1000, C = 2.5$	38.89%	62.37%	7.77%	25.00%	-0.20	6 000
$I = 1000, C = 3.75$	39.62%	69.39%	8.73%	21.79%	+0.15	6 000
$I = 1000, C = 5$	40.42%	69.16%	10.00%	20.87%	+0.05	6 000
$I = 10000, C = 2.5$	36.98%	66.94%	6.75%	22.07%	-0.09	6 000
$I = 10000, C = 3.75$	36.33%	74.22%	9.08%	23.88%	+0.22	6 000
$I = 10000, C = 5$	35.45%	73.85%	9.65%	28.01%	-0.12	6 000
$I = 100000, C = 2.5$	35.57%	70.84%	6.29%	30.73%	-0.08	6 000
$I = 100000, C = 3.75$	32.66%	76.90%	7.11%	29.26%	+0.02	6 000
$I = 100000, C = 5$	31.76%	82.80%	8.26%	30.00%	+0.06	6 000
$I = 200000, C = 2.5$	34.84%	73.75%	4.93%	38.99%	+0.07	6 000
$I = 200000, C = 3.75$	32.29%	80.30%	6.88%	30.67%	-0.01	6 000
$I = 200000, C = 5$	31.13%	82.29%	7.28%	31.06%	-0.06	6 000
$I = 500000, C = 2.5$	34.57%	73.74%	5.01%	34.19%	-0.03	6 000
$I = 500000, C = 3.75$	32.26%	80.43%	6.03%	26.18%	-0.11	6 000
$I = 500000, C = 5$	30.95%	85.17%	6.34%	26.63%	+0.14	6 000
$I = 1000000, C = 2.5$	35.12%	76.15%	4.56%	42.86%	+0.06	6 000
$I = 1000000, C = 3.75$	30.96%	83.43%	5.85%	33.33%	0.00	6 000
$I = 1000000, C = 5$	30.72%	87.46%	6.69%	32.08%	-0.06	6 000
$I = 2000000, C = 2.5$	34.05%	73.40%	3.44%	43.40%	0.00	6 000
$I = 2000000, C = 3.75$	32.22%	86.84%	5.32%	34.97%	+0.03	6 000
$I = 2000000, C = 5$	29.92%	90.07%	5.75%	28.98%	-0.03	6 000

LITERATUR

- [All+21] Matteo Alleman, Jonathan Mamou, Miguel A. Del Rio, Hanlin Tang, Yoon Kim und SueYeon Chung. "Syntactic Perturbations Reveal Representational Correlates of Hierarchical Phrase Structure in Pretrained Language Models". In: *Proceedings of the 6th Workshop on Representation Learning for NLP*. Hrsg. von Anna Rogers, Iacer Calixto, Ivan Vulić, Naomi Saphra, Nora Kassner, Oana-Maria Camburu, Trapit Bansal und Vered Shwartz. Online: Association for Computational Linguistics, 2021, S. 263–276. doi: [10.18653/v1/2021.repl4nlp-1.27](https://doi.org/10.18653/v1/2021.repl4nlp-1.27).
- [All94] Louis V. Allis. "Searching for Solutions in Games and Artificial Intelligence". Ph.D. thesis. Maastricht: Rijksuniversiteit Limburg, Maastricht University, 1994. doi: [10.26481/dis.19940923la](https://doi.org/10.26481/dis.19940923la).
- [ACBF02] Peter Auer, Nicolò Cesa-Bianchi und Paul Fischer. "Finite-Time Analysis of the Multi-Armed Bandit Problem". In: *Machine Learning* 47.2 (2002), S. 235–256. issn: 1573-0565. doi: [10.1023/A:1013689704352](https://doi.org/10.1023/A:1013689704352).
- [BKH16] Jimmy Lei Ba, Jamie Ryan Kiros und Geoffrey E. Hinton. *Layer Normalization*. 2016. arXiv: [1607.06450](https://arxiv.org/abs/1607.06450) [stat.ML]. URL: <https://arxiv.org/abs/1607.06450> (besucht am 09.05.2025).
- [Bar+20] Nolan Bard u. a. "The Hanabi Challenge: A New Frontier for AI Research". In: *Artificial Intelligence* 280 (2020), S. 103216. issn: 0004-3702. doi: [10.1016/j.artint.2019.103216](https://doi.org/10.1016/j.artint.2019.103216).
- [BCV13] Yoshua Bengio, Aaron Courville und Pascal Vincent. "Representation Learning: A Review and New Perspectives". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 35.8 (2013), S. 1798–1828. issn: 0162-8828. doi: [10.1109/TPAMI.2013.50](https://doi.org/10.1109/TPAMI.2013.50).
- [BCK23] Jannis Blüml, Johannes Czech und Kristian Kersting. "AlphaZe: AlphaZero-like Baselines for Imperfect Information Games Are Surprisingly Strong". In: *Frontiers in Artificial Intelligence* 6 (2023). doi: [10.3389/frai.2023.1014561](https://doi.org/10.3389/frai.2023.1014561).
- [Bou96] Craig Boutilier. "Planning, Learning and Coordination in Multiagent Decision Processes". In: *Proceedings of the 6th Conference on Theoretical Aspects of Rationality and Knowledge*. Morgan Kaufmann Publishers Inc., 1996, S. 195–210. isbn: 1558604179.
- [Bro] Andries E. Brouwer. *Go: Some Statistics*. URL: <https://homepages.cwi.nl/~aeb/go/misc/gostat.html> (besucht am 09.05.2025).

- [Bro+12] Cameron B. Browne, Edward Powley, Daniel Whitehouse, Simon M. Lucas, Peter I. Cowling, Philipp Rohlfshagen, Stephen Taverer, Diego Perez, Spyridon Samothrakis und Simon Colton. "A Survey of Monte Carlo Tree Search Methods". In: *IEEE Transactions on Computational Intelligence and AI in Games* 4.1 (2012), S. 1–43. doi: 10.1109/TCIAIG.2012.2186810.
- [CHHo2] Murray Campbell, Joseph Hoane und Feng hsiung Hsu. "Deep Blue". In: *Artificial Intelligence* 134.1 (2002), S. 57–83. doi: 10.1016/S0004-3702(01)00129-1.
- [Cha+08] Guillaume Chaslot, Mark H. M. Winands, Hendrik Jaap Van Den Herik, Jos W. H. M. Uiterwijk und Bruno Bouzy. "Progressive Strategies for Monte-Carlo Tree Search". In: *New Mathematics and Natural Computation* 4.3 (2008), S. 343–357. doi: 10.1142/S1793005708001094.
- [Couo6] Rémi Coulom. "Efficient Selectivity and Backup Operators in Monte-Carlo Tree Search". In: *Computers and Games*. Hrsg. von Jaap Van Den Herik, Paolo Ciancarini und Jeroen H. H. L. M. Donkers. Bd. 4630. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2006, S. 72–83. ISBN: 978-3-540-75537-1. doi: 10.1007/978-3-540-75538-8_7.
- [CPW12] Peter I. Cowling, Edward J. Powley und Daniel Whitehouse. "Information Set Monte Carlo Tree Search". In: *IEEE Transactions on Computational Intelligence and AI in Games* 4.2 (2012), S. 120–143. ISSN: 1943-068X, 1943-0698. doi: 10.1109/tciaig.2012.2200894.
- [CWP15] Peter I. Cowling, Daniel Whitehouse und Edward J. Powley. "Emergent Bluffing and Inference with Monte Carlo Tree Search". In: *2015 IEEE Conference on Computational Intelligence and Games (CIG)*. 2015, S. 114–121. doi: 10.1109/cig.2015.7317927.
- [DDV] Deutscher Doppelkopf-Verband e. V. *Turnier-Spielregeln (TSR) des Deutschen Doppelkopf-Verbandes e. V. (DDV)*. 2024. URL: https://www.doko-verband.de/Regeln_Ordnungen.html?file=tl_files/DDV/Docs/Downloads/Regeln%20und%20Ordnungen/Turnierspielregeln%20Stand%202024-02-24.pdf (besucht am 09.05.2025).
- [DU07] Peter Drake und Steve Urtamo. "Move Ordering vs Heavy Playouts: Where Should Heuristics Be Applied in Monte Carlo Go?" In: *Proceedings of the 3rd North American Game-On Conference*. 2007, S. 171–175.
- [Fos23] David Foster. *Generative Deep Learning*. 2nd Edition, April 2023. O'Reilly Media, Inc., 2023.

- [FB98] Ian Frank und David Basin. "Search in Games with Incomplete Information: A Case Study Using Bridge Card Play". In: *Artificial Intelligence* 100.1–2 (1998), S. 87–123. ISSN: 0004-3702. doi: 10.1016/S0004-3702(97)00082-9. URL: [https://doi.org/10.1016/S0004-3702\(97\)00082-9](https://doi.org/10.1016/S0004-3702(97)00082-9).
- [Gal+24] Maris F. L. Galesloot, Thiago D. Simão, Sebastian Junges und Nils Jansen. "Factored Online Planning in Many-Agent POMDPs". In: *Proceedings of the Thirty-Eighth AAAI Conference on Artificial Intelligence, Thirty-Sixth Conference on Innovative Applications of Artificial Intelligence, and Fourteenth Symposium on Educational Advances in Artificial Intelligence*. AAAI Press, 2024. ISBN: 978-1-57735-887-9. doi: 10.1609/aaai.v38i16.29689. URL: <https://doi.org/10.1609/aaai.v38i16.29689>.
- [GWC24] Sam Ganzfried, Kevin A. Wang und Max Chiswick. *Opponent Modeling in Multiplayer Imperfect-Information Games*. 2024. arXiv: 2212.06027 [cs.GT]. URL: <https://arxiv.org/abs/2212.06027> (besucht am 09.05.2025).
- [Gin01] Matthew L. Ginsberg. "GIB: Imperfect Information in a Computationally Challenging Game". In: *Journal of Artificial Intelligence Research* 14 (2001), S. 303–358. doi: 10.1613/jair.820.
- [GBC16] Ian Goodfellow, Yoshua Bengio und Aaron Courville. *Deep Learning*. MIT Press, 2016.
- [He+16] Kaiming He, Xiangyu Zhang, Shaoqing Ren und Jian Sun. "Deep Residual Learning for Image Recognition". In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, S. 770–778. doi: 10.1109/cvpr.2016.90.
- [HSW89] Kurt Hornik, Maxwell Stinchcombe und Halbert White. "Multi-layer Feedforward Networks Are Universal Approximators". In: *Neural Networks* 2.5 (1989), S. 359–366. ISSN: 0893-6080.
- [ITW18] Maximilian Ilse, Jakub M. Tomczak und Max Welling. "Attention-Based Deep Multiple Instance Learning". In: *Proceedings of the 35th International Conference on Machine Learning*. Hrsg. von Jennifer Dy und Andreas Krause. Bd. 80. Proceedings of Machine Learning Research. PMLR, 2018, S. 2127–2136.
- [IS15] Sergey Ioffe und Christian Szegedy. "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift". In: *Proceedings of the 32nd International Conference on Machine Learning*. ICML'15. Lille, France, 2015, S. 448–456.
- [JMo8] Daniel Jurafsky und James Martin. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*. Bd. 2. Pearson, Feb. 2008.

- [Kam+20] Parameswaran Kamalaruban, Rati Devidze, Volkan Cevher und Adish Singla. *Environment Shaping in Reinforcement Learning Using State Abstraction*. 2020. arXiv: 2006 . 13160 [cs.LG]. URL: <https://arxiv.org/abs/2006.13160> (besucht am 09.05.2025).
- [Kar20] Andrej Karpathy. *minGPT: A Minimal PyTorch Re-Implementation of the OpenAI GPT (Generative Pretrained Transformer) Training*. GitHub repository. 2020. URL: <https://github.com/karpathy/minGPT> (besucht am 09.05.2025).
- [KB15] Diederik P. Kingma und Jimmy Ba. "Adam: A Method for Stochastic Optimization". In: *International Conference on Learning Representations (ICLR)*. 2015.
- [Kit+24] Haruka Kita, Sotetsu Koyamada, Yotaro Yamaguchi und Shin Isihii. "A Simple, Solid, and Reproducible Baseline for Bridge Bidding AI". In: *2024 IEEE Conference on Games (CoG)*. 2024, S. 1–4. doi: [10.1109/CoG60054.2024.10645547](https://doi.org/10.1109/CoG60054.2024.10645547).
- [KN23] Steve Klabnik und Carol Nichols. *The Rust Programming Language*. 2nd edition. San Francisco: No Starch Press, 2023. 527 S.
- [KS06] Levente Kocsis und Csaba Szepesvári. "Bandit-Based Monte Carlo Planning". In: *Machine Learning: ECML 2006*. Hrsg. von Johannes Fürnkranz, Tobias Scheffer und Myra Spiliopoulou. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, S. 282–293. ISBN: 978-3-540-45375-8. doi: [10.1007/11871842_29](https://doi.org/10.1007/11871842_29).
- [KSW06] Levente Kocsis, Csaba Szepesvári und Jan Willemson. *Improved Monte-Carlo Search*. Techn. Ber. University of Tartu, Estonia, 2006. URL: <https://old.sztaki.hu/~szcsaba/papers/cg06-ext.pdf> (besucht am 09.05.2025).
- [Kuh51] Harold W. Kuhn. "A Simplified Two-Person Poker". In: *Contributions to the Theory of Games (AM-24), Volume I*. Hrsg. von Harold W. Kuhn und Albert W. Tucker. Princeton University Press, 1951, S. 97–104. ISBN: 978-1-4008-8172-7. doi: [10.1515/9781400881727-010](https://doi.org/10.1515/9781400881727-010).
- [Kuh53] Harold W. Kuhn. "Extensive Games and the Problem of Information". In: *Contributions to the Theory of Games (AM-28), Volume II*. Hrsg. von Harold W. Kuhn und Albert W. Tucker. Princeton University Press, 1953, S. 193–216. ISBN: 978-1-4008-8197-0. doi: [10.1515/9781400881970-012](https://doi.org/10.1515/9781400881970-012).
- [Lan+09] Marc Lanctot, Kevin Waugh, Martin Zinkevich und Michael Bowling. "Monte Carlo Sampling for Regret Minimization in Extensive Games". In: *Advances in Neural Information Processing Systems*. Hrsg. von Y. Bengio, D. Schuurmans, J. Lafferty, C. Williams und A. Culotta. Bd. 22. Curran Associates, Inc., 2009.

- [Lan+19] Marc Lanctot u. a. *OpenSpiel: A Framework for Reinforcement Learning in Games*. 2019. arXiv: 1908 . 09453 [cs.LG]. URL: <https://arxiv.org/abs/1908.09453>.
- [Lev89] David Levy. "The Million Pound Bridge Program". In: *Heuristic Programming in Artificial Intelligence*. Ellis Horwood, 1989.
- [Li+18] Hao Li, Zheng Xu, Gavin Taylor, Christoph Studer und Tom Goldstein. "Visualizing the Loss Landscape of Neural Networks". In: *Proceedings of the 32nd International Conference on Neural Information Processing Systems*. NIPS'18. Curran Associates Inc., 2018, S. 6391–6401.
- [Lon+10] Jeffrey Long, Nathan R. Sturtevant, Michael Buro und Timothy Furtak. "Understanding the Success of Perfect Information Monte Carlo Sampling in Game Tree Search". In: *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence (AAAI-10)*. AAAI Press, 2010, S. 134–140.
- [MSZ20] Michael B. Maschler, Eilon Solan und Shmuel Zamir. *Game Theory*. Second Edition. Cambridge: Cambridge University Press, 2020. ISBN: 9781108493451.
- [Mik+13] Tomas Mikolov, Kai Chen, Gregory S. Corrado und Jeffrey Dean. "Efficient Estimation of Word Representations in Vector Space". In: *International Conference on Learning Representations*. 2013.
- [MI20] Miguel Morales und Charles Isbell. *Grokking Deep Reinforcement Learning*. Shelter Island: Manning, 2020. ISBN: 978-1-61729-545-4.
- [Mur22] Kevin P. Murphy. *Probabilistic Machine Learning: An Introduction*. MIT Press, 2022.
- [Nas50] John F. Nash. "Equilibrium Points in n-Person Games". In: *Proceedings of the National Academy of Sciences* 36.1 (1950), S. 48–49. ISSN: 0027-8424, 1091-6490. doi: 10 . 1073/pnas . 36 . 1 . 48.
- [Nas51] John F. Nash. "Non-Cooperative Games". In: *The Annals of Mathematics* 54.2 (1951), S. 286. ISSN: 0003-486X. doi: 10 . 2307/1969529.
- [Pas+17] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga und Adam Lerer. "Automatic Differentiation in PyTorch". In: *Proceedings of the NIPS 2017 Workshop on Autodiff*. 2017.
- [Per+22] Julien Perolat u. a. "Mastering the Game of Stratego with Model-Free Multiagent Reinforcement Learning". In: *Science* 378.6623 (2022), S. 990–996. ISSN: 1095-9203. doi: 10 . 1126 / science . add4679.
- [PB19] Nick Petosa und Tucker Balch. *Multiplayer AlphaZero*. 2019. arXiv: 1910 . 13012 [cs.AI]. URL: <https://arxiv.org/abs/1910.13012> (besucht am 09.05.2025).

- [PyE] PyTorch Authors. *Embedding - PyTorch Documentation*. URL: <https://pytorch.org/docs/stable/generated/torch.nn.Embedding.html> (besucht am 09.05.2025).
- [PyL] PyTorch Authors. *LayerNorm - PyTorch Documentation*. URL: <https://pytorch.org/docs/stable/generated/torch.nn.LayerNorm.html> (besucht am 09.05.2025).
- [Rad+19] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei und Ilya Sutskever. *Language Models Are Unsupervised Multitask Learners*. OpenAI Blog. 2019. URL: https://cdn.openai.com/better-language-models/language_models_are_unsupervised_multitask_learners.pdf (besucht am 09.05.2025).
- [RP] Wijaya Royyan und Aldo Picaso. *Neu Icons*. URL: <https://github.com/neuicons/neu> (besucht am 09.05.2025).
- [RHW86] David E. Rumelhart, Geoffrey E. Hinton und Ronald J. Williams. "Learning Representations by Back-Propagating Errors". In: *Nature* 323 (1986), S. 533–536.
- [SB20a] Arta Seify und Michael Buro. *Single-Agent Optimization Through Policy Iteration Using Monte Carlo Tree Search*. 2020. arXiv: 2005.11335 [cs.LG]. URL: <https://arxiv.org/abs/2005.11335> (besucht am 09.05.2025).
- [Sha88] Claude E. Shannon. "Programming a Computer for Playing Chess". In: *Computer Chess Compendium*. Hrsg. von David Levy. New York, NY: Springer New York, 1988. doi: 10.1007/978-1-4757-1968-0_1.
- [SH15] Silvan Sievers und Malte Helmert. "A Doppelkopf Player Based on UCT". In: *KI 2015: Advances in Artificial Intelligence*. Hrsg. von Steffen Hölldobler, Rafael Peñaloza und Sebastian Rudolph. Bd. 9324. Lecture Notes in Computer Science. Springer, Cham, 2015. doi: 10.1007/978-3-319-24489-1_12.
- [Sil+16] David Silver u. a. "Mastering the Game of Go with Deep Neural Networks and Tree Search". In: *Nature* 529.7587 (2016), S. 484–489. ISSN: 0028-0836, 1476-4687. doi: 10.1038/nature16961.
- [Sil+17a] David Silver u. a. *Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm*. 2017. arXiv: 1712.01815 [cs]. URL: <https://arxiv.org/abs/1712.01815> (besucht am 09.05.2025).
- [Sil+17b] David Silver u. a. "Mastering the Game of Go without Human Knowledge". In: *Nature* 550.7676 (2017), S. 354–359. ISSN: 0028-0836, 1476-4687. doi: 10.1038/nature24270.
- [Sus] Herb Susmann. *Dirichlet Distribution*. URL: <https://observablehq.com/@herbps10/dirichlet-distribution> (besucht am 09.05.2025).

- [SB2ob] Richard S. Sutton und Andrew Barto. *Reinforcement Learning: An Introduction*. Second edition. Adaptive Computation and Machine Learning. Cambridge, Massachusetts London, England: The MIT Press, 2020. ISBN: 978-0-262-03924-6.
- [Świ+23] Maciej Świechowski, Konrad Godlewski, Bartosz Sawicki und Jacek Mandziuk. "Monte Carlo Tree Search: A Review of Recent Modifications and Applications". In: *Artificial Intelligence Review* 56 (2023), S. 2497–2562. doi: 10.1007/s10462-022-10228-y.
- [Lco] The LCZero Authors. *Leela Chess Zero*. URL: <https://github.com/LeelaChessZero/lc0> (besucht am 09.05.2025).
- [Tom24] Jakub M. Tomczak. *Deep Generative Modeling*. Springer Cham, 2024. ISBN: 978-3-031-64087-2. doi: 10.1007/978-3-031-64087-2.
- [Tro10] John Tromp. *John's Connect Four Playground*. 2010. URL: <https://tromp.github.io/c4/c4.html> (besucht am 09.05.2025).
- [Vas+17] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser und Illia Polosukhin. "Attention Is All You Need". In: *Proceedings of the 31st International Conference on Neural Information Processing Systems*. NIPS'17. Curran Associates Inc., 2017, S. 6000–6010. ISBN: 9781510860964.
- [WO12] Marco Wiering und Martijn Van Otterlo, Hrsg. *Reinforcement Learning: State-of-the-Art*. Bd. 12. Adaptation, Learning, and Optimization. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012. ISBN: 978-3-642-27644-6, 978-3-642-27645-3. doi: 10.1007/978-3-642-27645-3.
- [Wik16] Wikimedia Commons. *Kuhn Poker Tree*. 2016. URL: https://de.wikipedia.org/wiki/Datei:Kuhn_poker_tree.svg (besucht am 09.05.2025).
- [Zin+07] Martin Zinkevich, Michael Johanson, Michael Bowling und Carmelo Piccione. "Regret Minimization in Games with Incomplete Information". In: *Proceedings of the 21st International Conference on Neural Information Processing Systems*. NIPS'07. Red Hook, NY, USA: Curran Associates Inc., 2007, S. 1729–1736. ISBN: 9781605603520.