▲ Hochschule Harz
Hochschule für angewandte Wissenschaften

Maschinelles Lernen in Kartenspielen mit imperfekten Informationen

Bachelorarbeit

Erstprüfer: Prof. Jürgen Singer, Ph.D.
Zweitprüfer: Prof. Daniel Ackermann

Fachbereich: Automatisierung und Informatik
Studiengang: Medieninformatik

Bachelorarbeit-Nr.: BA AI 25/2023
Abgabedatum: 24.07.2023

Autor: Lennart Keidel
Matrikelnr.: 23288

Hochschule Harz
Fachbereich Automatisierung und Informatik
Lennart Keidel
m23288

Thema und Aufgabenstellung der Bachelorarbeit
BA AI 25/2023

Maschinelles Lernen in Kartenspielen mit imperfekten Informationen

Künstliche Intelligenz im Allgemeinen kann als lernende Agenten in fest definierten Umgebungen verstanden werden. Spiele bieten durch klare Abläufe und Regeln von vornherein eine Abstraktionsebene, wodurch sie attraktiv für die Anwendung von maschinellem Lernen sind. Innerhalb der letzten Jahre gab es viele Erfolge von künstlichen Intelligenzen in Spielen. Künstliche Intelligenzen haben in Spielen mit perfekter Information wie Schach (1996) und Go (2016), dem komplexesten dieser Kategorie, seither das "super-human level" erreicht; heißt künstliche Intelligenz entscheidet in diesen Brettspielen besser als Menschen. Ein Spiel mit perfekten Informationen bedeutet, dass es für jeden Spieler zu jedem Zeitpunkt des Spielverlaufs möglich ist, alle bisherigen Züge, als auch die Möglichkeiten des Gegners zu erkennen.
In Spielen mit imperfekter Information zeichnet sich ein ähnlicher Trend ab: In den Kartenspielen Heads-Up No-Limit Poker (2017) und Bridge (2022), für die strategisches Handeln rudimentär sind, wurde durch maschinelles Lernen ein "high-human level" erreicht; heißt Künstliche Intelligenz kann in diesen Kartenspielen auf Expertenniveau spielen. Ein Spiel mit imperfekter Information bedeutet, dass nur eigene mögliche Züge und offen liegende Informationen zur Bewertung der Züge bleiben.

Ziel dieser Bachelorarbeit ist es, maschinelles Lernen für ein Kartenspiel mit imperfekten Informationen umzusetzen und dafür bestehende Ansätze und Erkenntnisse dieses Gebiets zu recherchieren.

Die Bachelorarbeit beinhaltet folgende Teilaufgaben:

- Recherche bestehender Ansätze von Künstlichen Intelligenzen in Kartenspielen mit imperfekten Informationen
- Analyse geeigneter Tech-Stacks für maschinelles Lernen in diesem Umfeld
- Erprobung eigener Ansätze bzw. Abwandlungen oder Kombination bestehender für ein gewähltes Kartenspiel mit imperfekten Informationen und Begründung dieser
- Entwicklung eines Prototypen
- Bewertung der Ergebnisse

Prof. Jürgen Singer, Ph.D.	Prof. Daniel Ackermann

Inhaltsverzeichnis

Glossar

Abbildung (maschinelles Lernen) — Im maschinellen Lernen ist eine Abbildung der Zusammenhang von Eingabedaten, die ein maschinelles Lernverfahren erlernen soll [1, S. 205-206].

Agent — „Als Agent bezeichnen wir ganz allgemein ein System, welches Information verarbeitet und aus einer Eingabe eine Ausgabe produziert. Diese Agenten lassen sich in vielfältiger Weise klassifizieren. In der klassischen Informatik werden hauptsächlich Software-Agenten verwendet.“ [1, S. 19-20]

Aktion (im MDP) — Eine Aktion in einem MDP ist eine Handlung eines Agenten, die in einem Zustand möglich ist. Eine Aktion kann jeder Art von Entscheidung entsprechen [2, S. 47-48].

Aktionsmenge, Aktionsraum — Die Aktionsmenge (Synonym Aktionsraum) ist in einem MDP die Menge aller möglichen Aktionen.

Aktivierungsfunktion — Eine Aktivierungsfunktion nimmt die Summe der gewichteten Eingabewerte eines Neurons und wendet eine zuvor festgelegte mathematische Funktion an, aus der die Ausgabe (Aktivierung) des Neurons resultiert [3, S. 31].

Algorithmus — „Ein Algorithmus ist grob gesprochen eine wohldefinierte Rechenvorschrift, die eine Größe oder eine Menge von Größen als Eingabe verwendet und eine Größe oder eine Menge von Größen als Ausgabe erzeugt. Somit ist ein Algorithmus eine Folge von Rechenschritten, die die Eingabe in die Ausgabe umwandeln.“ [4, S. 5]

approximieren — Approximieren ist ein fachsprachliches Synonym für annähern und wird häufig im mathematischen Kontext verwendet [5].

Array — Ein Array ist eine Datenstruktur, die in der Informatik verwendet wird, um mehrere ähnlich strukturierte Daten zu speichern.

Backupdiagramm Ein Backupdiagram wird verwendet, um die Zusammenhänge von Zuständen, Aktion und Belohnungen in einem MDP zu visualisieren.

Baum (Datenstruktur) Ein Baum ist eine hierarchische Datenstruktur, in der Knoten durch Kanten (auch Äste genannt) miteinander verbunden sind. Knoten die miteinander verbunden sind, haben einen Bezug zueinander. Es gibt unterschiedliche Spezifizierungen von Baum-Datenstrukturen, die im jeweiligen Kontext sinnvoll sind (bspw. Suchbaum oder Spielbaum).

Bellman-Gleichung Die Bellman-Gleichung, auf der die Wertfunktionen basieren, wird im MDP verwendet, um den rekursiven Zusammenhang zwischen dem Wert eines Zustands und dem kumulativen gewichteten Wert seiner Folgezustände unter Berücksichtigung der Übergangswahrscheinlichkeiten zu ermitteln [2, S. 58-59, 6].

Bellman-Optimalitätsgleichung Die Bellman-Optimalitätsgleichung wird in einem MDP angewandt, um optimale Wertfunktionen zu bestimmen und daraus eine optimale Strategie. [2, S. 62-63, 6]

Bellman-Prinzip Das Bellman-Prinzip besagt: „Unabhängig vom Startzustand s_t und der ersten Aktion a_t müssen ausgehend von jedem möglichen Nachfolgezustand s_{t+1} alle folgenden Entscheidungen optimal sein.“ Basierend auf diesem Prinzip lässt sich im MDP eine globale optimale Strategie π_* finden, in den lokale Aktionen optimiert werden [1, S. 358, 6].

Belohnung (im MDP) In einem MDP ergeben sich aus der Umgebung Belohnungen (numerische Werte), die der Agent durch die Wahl seiner Aktionen maximieren möchte [2, S. 47-48].

Belohnungsmenge, Belohnungsraum Die Belohnungsmenge (Synonym Belohnungsraum) ist in einem MDP die Menge aller möglichen Belohnungen.

beobachtbar (Spieltheorie) Eine Aktion oder ein Spielzug ist beobachtbar, wenn für andere Spieler nachvollziehbar sind, welche Entscheidung der Spieler getroffen hat [1, S. 109, 7, S. 159].

Bias Der Bias ist ein numerischer Wert, der in jedem Neuron enthalten ist und bei Aktivierung eines Neurons relevant ist. Im Lernprozess wird unter anderem der Bias-Wert iterativ angepasst. Je nach Darstellung wird der Bias-Wert auch als eigenes Neuron gesehen, wobei die Anwendung und Funktion identisch sind [3, S. 54].

Blattknoten In einer Baumstruktur ist ein Knoten ein Blattknoten, wenn dieser keinen Kindknoten hat.

Breitensuche Die Breitensuche iteriert durch einen Suchbaum von links nach rechts, Ebene für Ebene, bis eine Lösung gefunden wurde. Breitensuche ist eine uninformierte Suche, da sie keine Heuristik verwendet [1, S. 110].

Convolutional Neural Network Ein Convolutional Neural Network (deutsch gefaltetes neuronales Netz) ist eine spezielle Art von künstlichem neuronalem Netz, welches besonders für die Anwendung in hochdimensionalen Merkmalsräumen geeignet ist und bspw. für die Dimensionsreduktion genutzt wird [3, S. 65-70, 8].

deterministisch (Spieltheorie) Eine Aktion (bzw. ein Spielzug) ist deterministisch, wenn es bei einen gleichem Ausgangszustand immer zum gleichen Nachfolgezustand kommt. Sie ist somit zufallsfrei. Bekannte Beispiele für deterministische Spiele sind Schach und Dame [1, S. 128].

Diskontierungsfaktor Um verzögerte Belohnungen in einem MDP variabel zu gewichten, wird der Diskontierungsfaktor γ (Wert zwischen 0 und 1; $0 \geq \gamma \leq 1$) verwendet. Die Belohnung der Folgezustände wird um den Faktor γ abgeschwächt [2, S. 55].

diskrete Probleme Probleme sind diskret, wenn sie eine endliche Anzahl an Lösungsmöglichkeiten haben.

Dynamische Programmierung — Dynamische Programmierung wird zum algorithmischen Lösen von Optimierungsproblemen eingesetzt, indem das Problem in Teilprobleme zerlegt und Zwischenergebnisse verwendet werden, um die optimale Lösung zu finden [1, S. 358].

Elternknoten — Ein Elternknoten bezieht sich auf den Knoten, welcher der Vorgänger des aktuellen Knotens in einer Baumstruktur ist.

Endknoten — Ein Endknoten ist ein Blattknoten in einer Baumstruktur. Endknoten werden bspw. im Kontext des Spielbaums verwendet, wobei der Endknoten eine Auszahlung in einem dynamischen Spiel enthält [7, S. 89].

entscheidbares (oder unentscheidbares) Problem — Für ein entscheidbares Problem existiert ein Algorithmus, der in endlicher Zeit eine korrekte Antwort oder eine eindeutige Lösung erzeugen kann. Das Gegensätzliche trifft auf ein unentscheidbares Problem zu.

Entscheidungsbaum — Ein Entscheidungsbaum ist eine Baum-Datenstruktur, die alle Entscheidungsmöglichkeiten abbildet. Sie ist grundsätzlich ähnlich zu einem Spielbaum und einem Suchbaum, wird jedoch in anderem Kontext verwendet.

Exploration-Exploitation-Dilemma — Das Exploration-Exploitation-Dilemma beschreibt ein grundlegendes Problem in Entscheidungssituationen: Auf die Bewertung der Alternativen vertrauen (bisheriges Wissen) oder die Alternativen weiter untersuchen (neues Wissen), denn die Bewertung der Alternativen könnte eventuell noch nicht optimal sein. Dieses Dilemma betrifft auch die KI-Forschung und wurde bereits im Rahmen des Multi-Armed-Bandit-Problems [9] mit vielen unterschiedlichen Lösungsansätzen untersucht.

Feed-Forward-Netz — In einem neuronalen Feed-Forward-Netz (deutsch Forwärtsnetz oder Netz mit Vorwärtskopplung) geht der Informationsfluss nur vorwärts, von der Eingabeschicht über

	die versteckten Schichten bis zur Ausgabeschicht. Dabei werden vergangene Zustände (anderer Schichten) nicht berücksichtigt [3, S. 62-65].
Framework	Ein Framework ist ein vorgefertigtes Grundgerüst, das in der Software-Entwiklung verwendet wird. Es enthält Funktionen oder Werkzeuge, auf die Entwickler zurückgreifen können.
generalisieren, Generalisierungsfähigkeit	Wenn ein Vorgang gelernt wurde und anhand neuer, noch nie gesehener Beispiele umgesetzt werden kann, dann nennt man es Generalisierung [1, S. 202].
Gewicht	Der Verbindung zwischen Neuronen sind Gewichte zugeordnet (numerische Werte zwischen null und eins), die angeben, wie stark diese Verbindung ist [3, S. 18-26].
Gleichgewicht (Spieltheorie), Nash-Gleichgewicht	Ein Gleichgewicht (auch Nash-Gleichgewicht genannt) ist dann erreicht, „wenn keiner der Spieler individuell seine Strategie nachträglich noch ändern wollen würde, nachdem er erfahren hat, welche Strategien der oder die anderen Spieler gewählt haben.“ [7, 15; 21]
Heuristik	Eine Heuristik ist eine geschätzte Bewertung, die einer exakten Bewertung vorgezogen wird, da die exakte Bewertung zu zeit- oder rechenaufwändig ist [1, S. 117-118].
Heuristische Funktion	Eine heuristische Funktion erstellt mittels einer mathematischen Funktion eine numerische geschätzte Bewertung. Die Heuristische Funktion ist Basis einer Heuristik [1, S. 117].
Heuristische Suche	Eine heuristische Suche priorisiert Alternativen anhand ihrer Heuristik bspw. im Suchbaum eines Entscheidungsproblems [1, S. 117-118].
imperfekte Informationen	Imperfekte Information ist ein spieltheoretischer Begriff und eine Steigerung zu unvollständigen Informationen.

	Imperfekte Informationen liegen vor, wenn zusätzlich zu unvollständigen Informationen nicht alle Züge aller Spieler beobachtbar sind [10, S. 46].
informierte Suche	Eine informierte Suche verwendet eine Heuristik, um Knoten im Suchbaum zu bewerten. Dadurch kann die Suche effizienter sein, da Knoten mit schlechter Heuristik benachteiligt oder sogar ignoriert werden, da diese weniger wahrscheinlich für eine Lösung in Frage kommen [1, S. 117-118].
iterativ	Iterativ ist ein fachsprachliches Synonym für wiederholend und wird häufig im mathematischen Kontext verwendet [11].
Kindknoten	Ein Kindknoten bezieht sich auf den Knoten, der auf den aktuellen Knoten in einer Baumstruktur folgt.
kumulativ	Kumulativ ist ein fachsprachliches Synonym für anhäufend oder steigernd [12].
Künstliche Intelligenz	KI ist ein Teilgebiet der Informatik und E. Rich [13] fasst bündig zusammen: KI befasst sich mit der Frage, wie man Computer dazu bringen kann, Dinge zu tun, die Menschen im Moment noch besser können.
künstliche neuronale Netze	Künstliche Neuronale Netze sind ein Teil des Maschinellen Lernens, die ursprünglich vom Aufbau des menschlichen Gehirns inspiriert wurden [3, S. 90]. Kombiniert mit anderen maschinellen Lernverfahren, lassen sich durch künsltiche neuronale Netze bspw. nichtlineare Abbildungen auf Daten erlernen [3, S. 2].
Leduc-Poker	Leduc Poker ist eine vereinfachte Version des Kartenspiels Texas Hold'em Poker. Sie eignet sich besonders gut für Studien und Analysen in der Spieltheorie. In Leduc Poker besteht das Kartendeck nur aus zwei Paaren von König, Dame und Bube, also insgesamt sechs Karten. Jedes Spiel wird mit

	zwei Spielern, zwei Runden, maximal zwei Einsätzen und Erhöhungen gespielt [14, S. 2].
Lernender Agent	„Für die KI von besonderem Interesse sind lernfähige Agenten, die anhand von Trainingsbeispielen erfolgreicher Aktionen oder durch positives oder negatives Feedback auf die Aktionen in der Lage sind, sich selbst so zu verändern, dass der mittlere Nutzen ihrer Aktionen im Laufe der Zeit wächst […].“ [1, S. 21]
Lernmethode	Eine Lernmethode ist im maschinellen Lernen ein Algorithmus, mit dem ein Modell iterativ eine Abbildung erlernt.
Lernverfahren	Im maschinellen Lernen wird zwischen diesen drei Lernverfahren unterschieden: supervised learning (deutsch überwachtes Lernen), unsupervised learning (deutsch unüberwachtes Lernen) und reinforcement learning (deutsch Lernen durch Verstärkung). Die Lernverfahren unterscheiden sich grundlegend darin, wie sie eine Abbildung aus Daten lernen, also eine sinnvolle Beziehung in den Eingabedaten finden.
Log-Odd	Log-Odd spielt eine wichtige Rolle in logistischer Regression. Ein Log-Odd ist Logarithmus aus dem Verhältnis des Gegenstücks einer Wahrscheinlichkeit (engl. Odd-Ratio) [15].
Markov-Entscheidungsprozess	Ein Markov-Entscheidungsprozess (MDP) (engl. Markov-Decision-Process) ist eine Formalisierung eines sequenziellen Entscheidungsproblems, bei der Aktionen nicht nur direkte Belohnungen, sondern auch nachfolgende Zustände und damit zukünftige Belohnungen beeinflussen. Es ist ein zentraler Bestandteil des reinforcement learnings. [2, S. 47-48]

Maximumfunktion Eine Funktion, die den höchsten Wert über Eingabewerte wählt.

Minimax-Algorithmus Der Minimax-Algorithmus erstellt Heuristiken für alle Spielzustände im Voraus bis zu einer festgelegten Tiefe und verwendet eine Kombination aus Breiten- und Tiefensuche, um den Spielbaum rekursiv zu beschneiden. Alpha-Beta-Pruning ist eine Erweiterung des Minimax-Algorithmus [1, S. 129].

Mittelwertfunktion Eine Funktion, die den Mittelwert über Eingabewerte ermittelt.

Modell (maschinelles Lernen) Beim maschinellen Lernen wird ein mathematisches Modell erlernt, indem Muster bzw. Gesetzmäßigkeiten aus Daten extrahiert werden und daraus bspw. Vorhersagen für neue unbekannte Daten genieren kann [3, S. 8].

Monte-Carlo-Tree-Search Monte-Carlo-Tree-Search ist ein heuristischer Suchalgorithmus, der zur Klasse der Monte-Carlo-Verfahren gehört und in einem Suchbaum durch zufällige Simulation die Heuristik anpasst und iterativ die Knoten mit der besten Heuristik expandiert [16].

Monte-Carlo-Verfahren Die Monte-Carlo-Verfahren sind eine Klasse an Methoden, die unter anderem Zufallsverfahren verwenden, um eine optimale Lösung zu finden oder mathematische Probleme zu approximieren und bspw. bei Problemen angewandt werden, bei denen analytische Ansätze schwierig oder nicht umsetzbar sind [16, S. 4].

Neuron Ein Neuron ist Bestandteil des künstlichen neuronalen Netzes. Ein Neuron empfängt Eingabewerte, die es aufsummiert, mit der individuellen Gewichtung multipliziert, der Bias-Wert addiert und darauf eine Aktivierungsfunktion anwendet. Der finale numerische Wert ist die Ausgabe (auch

Aktivierung genannt) des Neurons, die an andere Neuronen weitergegeben wird [3, S. 18-26].

nichtdeterministische — Nichtdeterministisch ist das Gegenteil von deterministisch und bedeutet, dass ein Ausgangszustand nicht immer den gleichen Nachfolgezustand erzeugt, weil es bspw. von einem Zufallsereignis wie einem Würfelergebnis abhängig ist. Beispiele für nichtdeterministische Spiele wären Mensch ärgere Dich nicht oder Backgammon [1, S. 129].

Nullsummenspiel — „[Nullsummenspiele] sind Spiele, bei denen jeder Gewinn eines Spielers einen Verlust des Gegenspielers in gleicher Höhe bedeutet. Die Summe aus Gewinn und Verlust ist also immer gleich null. Dies trifft auf die oben erwähnten Spiele Schach, Dame, Reversi und Go zu.“ [1, S. 128]

Perceptron — Das Perceptron ist das erste und simpelste neuronale Netz (F. Rosenblatt 1958 [17]). Es ist ein Feed-Forward-Netz, bestehend aus einer Eingabeschicht mit zwei Neuronen und einer Ausgabeschicht mit einem einem Neuron und ist bspw. für lineare Klassifzierungsaufgaben im supervised learning einsetzbar [1, S. 210].

perfekte Informationen — Perfekte Information ist ein spieltheoretischer Begriff und eine Steigerung zu vollständigen Informationen. Perfekte Informationen liegen vor, wenn zusätzlich jedem Spieler immer der komplette Spielzustand bekannt ist. Schach, Dame und Go sind Spiele mit perfekter Information, da jeder Zug beobachtbar ist [1, S. 128].

Python — Python ist eine Programmiersprache, die oft für Anwendungen im maschinellen Lernen verwendet wird.

Q-Funktion — Die Q-Funktion (auch Aktionswertfunktion) ist eine Wertfunktion, die im MDP den Wert einer Aktion ausgehend vom einem Zustand angibt unter Einbezug der erwarteten kumulativen Belohnung aller Folgezustände [2, S. 58-59].

Q-Learning

Q-Learning (deutsch Q-Lernen) ist eine wertbasierte Lernmethode, welche iterativ die Bewertung jeder Aktion in jedem Zustand (mittels Q-Funktion) ermittelt und tabellarisch abspeichert, wobei ein Tabelleneintrag die Bewertung für eine Aktion in einem Zustand ist [1, S. 364, 2, S. 131-132].

Realzeitanforderung

Die Realzeitanforderung ist der Anspruch innerhalb eines realistischen Zeitrahmens eine Lösung zu finden. Eine genaue Auslegung ist dabei von den Anforderungen des Problem abhängig.

Regret

Der Regret (deutsch Bedauern) ist spieltheoretische eine Metrik, die angibt, wie sehr die getroffenen Entscheidungen von den optimalen Entscheidungen abweichen. Der Regret ergibt sich aus der Subtraktion von der maximal erreichbareren Auszahlung und der tatsächlich erhaltenen Auszahlung [18, S. 5]. Der Regret wird bspw. oft als Metrik im Multi-Armed-Bandit-Problem [9] verwendet, um Ergebnisse von Algorithmen zu vergleichen [19], ist aber auch wichtiger Bestandteil des Counterfactual-Regret-Minimization-Algorithmus.

rekurrentes Netz

In einem neuronalen rekurrenten Netz haben Neuronen Verbindungen zu Neuronen der vorherigen Schicht oder mit sich selbst (Rückwärtskopplung) und vergangene Zustände werden berücksichtigt [3, S. 62-65].

RGB

RGB (Rot-Grün-Blau) ist ein Farbmodell zur Darstellung von Farben.

Sigmoid-Funktion

Die Sigmoid-Funktion ist eine Aktivierungsfunktion, die in künstlichen Neuronalen Netzen eingesetzt wird [1, S. 291].

Skat (Kartenspiel)

Skat ist ein Stich-Kartenspiel mit imperfekten Informationen für drei Personen. Charakteristisch für Skat

	sind die abwechselnden Reiz- und Spielphasen. (Siehe Spielregeln unter [20])
Spielbaum	Der Spielbaum ist eine Struktur der Spieltheorie, mit der die sequenzielle Struktur eines Spiels (bspw. die abwechselnden Spielzüge der Spieler) erfasst wird. „Jeder Zug eines Spielers wird durch einen Knoten dargestellt, an dem der Spieler zwischen verschiedenen Ästen (seinen Handlungsalternativen) wählen kann.“ [10, S. 14] Der Spielbaum kann auch als eine Spezialform des Suchbaums gesehen werden.
Spiele mit imperfekten Informationen	Bei Spielen mit imperfekten Informationen ist keinem Spieler der komplette Spielzustand bekannt, denn nicht jeder Spielzug ist für jeden Spieler komplett beobachtbar. [10, S. 46] "Viele Kartenspiele, wie zum Beispiel Skat, sind nur teilweise beobachtbar, denn der Spieler kennt die Karten des Gegners nicht oder nur teilweise." [1, S. 128]
Spiele mit perfekten Informationen	Bei Spielen mit perfekten Informationen ist jedem Spieler immer der komplette Spielzustand bekannt, denn jeder Spielzug ist für jeden Spieler beobachtbar. Schach, Dame oder GO sind beobachtbare Spiele mit perfekten Informationen [1, S. 128, 10, S. 46].
Spieltheorie	Die Spieltheorie ist ein mathematisches Modell, welches zur Analyse von strategischen Entscheidungssituationen eingesetzt wird [10, S. 1] und oft in Verbindung mit Wirtschaftswissenschaften steht.
stetiges Problem	Ein Problem ist stetig, wenn es einen kontinuierlichen und nicht endlichen Lösungsraum hat.
Strategie (reinforcement learning)	Eine Strategie ist eine Abbildung von Zuständen auf Aktionen und gibt an, welche Aktion in welchem Zustand gewählt werden soll [1, S. 354-355].

Strategie (Spieltheorie) Eine Strategie ist ein vollständiger Plan über alle Handlungen eines Spielers in einem Spiel. Eine Strategie stellt sicher, dass der Spieler in Übereinstimmung mit seinen Zielen handelt und dabei seine Chancen auf die erzielte Auszahlung maximiert [7, S. 7].

sublinear Eine mathematische Funktion ist sublinear, wenn das Wachstum unter dem einer linearen Funktion liegt, wie bspw. bei einer Wurzelfunktion oder logarithmischen Funktion.

Suchbaum Ein Suchbaum ist eine hierarchische Datenstruktur, die alle Entscheidungsmöglichkeiten in einem Zustand und dessen Folgezustände abbildet. Sie ist grundsätzlich sehr ähnlich zu einem Spielbaum und Entscheidungsbaum, wird jedoch in anderem Kontext verwendet.

TD-Learning TD-Learning (Temporal Difference Learning) (deutsch Lernen mit zeitlicher Differenz) ist eine Gruppe von wertbasierten modellfreien Lernmethoden des RL, welche zufällige Schätzungen, eine Eigenschaft der Monte-Carlo-Verfahren, und die Wertiteration, ein Konzept der dynamischen Programmierung, miteinander vereinen [2, S. 119]. Im TD-Learning wird eine Wertfunktion geschätzt, die sich iterativ der optimalen Wertfunktion nähert [2, S. 121].

terminieren Das Wort "terminieren" wird bspw. im Rahmen eines Programmablaufs verwendet. Ein Programm terminiert, wenn es einen Endzustand erreicht hat. Wenn ein Algorithmus oder Agent eine Endbedingung erreicht, dann terminiert bzw. stoppt dieser [21].

Tiefensuche Die Tiefensuche iteriert durch einen Suchbaum, in dem es jeden Knoten expandiert, bis ein Blattknoten erreicht ist. Wenn am Blattknoten keine Lösung vorliegt, werden iterativ alle Kindknoten des Elternknotens auf diese Weise durchsucht, bis eine Lösung gefunden wird. Die Tiefensuche

	ist eine uninformierte Suche, da sie keine Heuristik verwendet [1, S. 112].
Übergangswahrscheinlichkeit	In einem MDP wird jeder Zustand und dessen Belohnung mit einer gewissen Übergangswahrscheinlichkeit tatsächlich erreicht. Es ist also möglich mit der gleichen Aktion verschiedene Belohnungen und Folgezustände zu erreichen [2, S. 47-48].
uninformierte Suche	Eine uninformierte Suche verwendet keine Heuristik, sondern iteriert durch den Suchbaum, bis der optimale Endknoten gefunden ist. Im Gegensatz kann die uninformierte Suche bei komplexen Suchbäumen ineffizient sein, da es keine Priorisierung der Knoten gibt.
unvollständige Informationen	Unvollständige Information ist ein spieltheoretischer Begriff. In einem Spiel mit unvollständigen Informationen sind die Informationen über andere Spieler nicht ausreichend, um mit Sicherheit zu wissen wie sie ihre unterschiedlichen Spielausgänge bewerten [7, S. 2].
Verzweigungsfaktor	Der Verzweigungsfaktor eines Baums ist die durchschnittliche Zahl der Kindknoten jedes Knotens [1, S. 111].
V-Funktion	Die V-Funktion (auch Zustandswertfunktion) ist eine Wertfunktion, die im MDP den Wert eines Zustands angibt unter Einbezug der erwarteten kumulativen Belohnung aller Folgezustände [2, S. 58-59].
vollständige Informationen	Vollständige Information ist ein Begriff der Spieltheorie. Wenn alle Spieler „die Konsequenzen jedes möglichen Spielausgangs für alle anderen Spieler“ kennen und sich so die Perspektive jedes Spielers nachvollziehen lässt, dann hat das Spiel vollständige Informationen [7, S. 2].
Wertfunktion	Wertfunktionen (V-Funktion und Q-Funktion) werden im MDP verwendet, um den rekursiven Zusammenhang

	zwischen dem Wert eines Zustands und dem kumulativen gewichteten Wert seiner Folgezustände unter Berücksichtigung der Übergangswahrscheinlichkeiten zu ermitteln [2, S. 58-59].
Zustand (im Spielbaum/Suchbaum)	Ein Knoten eines Spielbaums oder Suchbaums „enthält den Zustand und noch weitere für die Suche relevante Informationen wie zum Beispiel die Tiefe im [Baum] und die heuristische Bewertung des Zustands.“ [1, S. 117]
Zustand (im MDP)	Ein Zustand im MDP enthält beliebige Informationen, die dem Agenten zugänglich sind und seine Entscheidungen beeinflussen [2, S. 47-48].
Zustandsmenge, Zustandsraum	Die Zustandsmenge (Synonym Zustandsraum) ist in einem MDP die Menge aller möglichen Zustände.

Abkürzungsverzeichnis

AI	Artificial Intelligence (deutsch: Künstliche Intelligenz)
bspw	beispielsweise
bzw	beziehungsweise
CFR	Counterfactual Regret Minimization (deutsch kontrafaktisches Minimieren des Bedauerns)
CNN	Convolutional Neural Network (deutsch gefaltetes neuronales Netz)
engl	englisch
FFN	Feed-Forward-Netz
*IDA**	iterative deepening A* (deutsch iteratives vertiefen A*)
KI	Künstliche Intelligenz
KNN	Künstliches neuronales Netz
MCTS	Monte-Carlo-Tree-Search (deutsch Monte-Carlo-Baum-Suche)
MDP	Markov-Decision-Process (deutsch Markov-Entscheidungs-Prozess)
NN	Neuronales Netz (Synonym für Künstliches neuronales Netz)
POMDP	teilweise beobachtbarer Markov-Entscheidungsprozess (engl. partially observable Markov-Decision-Prozess)
RL	reinforcement learning (deutsch Lernen durch Verstärkung)
RNN	rekurrentes Netz
SL	supervised learning (deutsch überwachtes Lernen)
UL	unsupervised learning (deutsch unüberwachtes Lernen)

Zusammenfassung der Notation

$\doteq$	Gleichheitsbeziehung die per Definition wahr ist
$E[X]$	Erwartungswert einer Zufallsvariable X
$\in$	ist ein Element von
$\ln x$	natürlicher Logarithmus von x
ε	Wahrscheinlichkeit eine zufällige Aktion in einer ε-Greedy-Strategie durchzuführen
$\sum_{k=0}^{n} a_k$	Summe über a_k von $k = 0$ bis $k = n$

Spieltheorie

i	ein Spieler i
j	ein anderer Spieler j
S	der Strategieraum S der alle möglichen Strategiekombinationen enthält
s	eine Strategiekombination aus dem Strategieraum S**,** der einem kompletten Spieldurchlauf entspricht
S_i	Menge aller Strategien, die für Spieler i möglich sind
s_i	eine Strategie s_i des Spielers i aus seiner Strategiemenge S_i
$u_i(s)$	Auszahlung für Spieler i wenn Strategie s_i gespielt wird

Markov-Entscheidungsprozess (Teil des reinforcement learnings)[1]

s	ein Zustand
s'	ein Folgezustand

[1] Die Notation in einem Markov-Entscheidungsprozess ist in der Literatur nicht durchweg einheitlich, auch wenn gemeinsame Sachverhalte und Formeln bestehen. Aus diesem Grund wird sich auf diese Notation von R. S. Sutton und A. Barto [2] beschränkt.

a	eine Aktion
r	eine Belohnung
S	Menge alle Zustände
$A(s)$	Menge aller Aktionen im Zustand s
R	Menge aller Belohnungen
t	diskreter Zeitpunkt
$t+1$	nächster Zeitpunkt
S_t	Zustand S zum Zeitpunkt t
A_t	Aktion A zum Zeitpunkt t
R_t	Belohnung R zum Zeitpunkt t
G_t	Rückgabewert zum Zeitpunkt t
γ	Diskontierungsfaktor für eine kumulative Belohnung
π	Strategie in einem Markov-Entscheidungsprozess
π_*	optimale Strategie
$\pi(a\|s)$	Wahrscheinlichkeit, dass Aktion a gewählt wird im Zustand s unter Beachtung einer stochastischen Strategie π
$\pi(s)$	Gewählte Aktion a unter Beachtung einer deterministischen Strategie π
$E_\pi[X]$	Erwartungswert einer Zufallsvariable X unter Beachtung der Strategie π
$p(s', r \mid s, a)$	Übergangswahrscheinlichkeit in den Zustand s' mit der Belohnung r zu gelangen, ausgehend vom Zustand s und der Aktion a
$p(s' \mid s, a)$	Übergangswahrscheinlichkeit in den Zustand s' zu gelangen, ausgehend vom Zustand s und der Aktion a
$v_\pi(s)$	Wert von Zustand s unter Strategie π

$v_*(s)$	Wert von Zustand s unter der optimalen Strategie π_*
$q_\pi(s,a)$	Wert die Aktion a zu im Zustand s zu wählen unter der Strategie π_*
$q_*(s,a)$	Wert die Aktion a zu im Zustand s zu wählen unter der optimalen Strategie π_*
$\max_a$ bzw. $\max_{a'}$	wähle die Aktion a bzw. a', welche den höchsten Nutzen erreicht
$\max_{a \in A(s)}$	wähle die Aktion a aus der Aktionsmenge A im Zustand s, welche den höchsten Nutzen erreicht

Künstliche Neuronale Netze

x	Eingabewert eines Neurons
w	Gewicht eines Neurons
θ	Schwellenwert θ der Schwellenwertfunktion

1 Einleitung

Aktuell dominiert ein Thema die mediale Aufmerksamkeit immer häufiger: ***Künstliche Intelligenz (KI)*** und ihre scheinbar unzähligen Anwendungsmöglichkeiten .Die Frage nach dem Bewusstsein von KI, die Unsicherheit über zukünftige Entwicklungen und Auswirkungen bspw. auf die Arbeitswelt sowie Sprachmodelle, die den häufig zitierten Turing-Test[2] bestehen [23], generative KI, die anhand von wenigen Eingabebefehlen beliebige fotorealistische Bilder erstellt [24] und Spiele-KI, die menschliche Expertenspieler schlägt [25–31]. KI hat durch diese Erfolge besonders in den letzten Jahren eine immer größere mediale Reichweite erhalten, auch wenn die KI-Forschung und das maschinelle Lernen auf eine lange und stetige Entwicklung seit den 1940er Jahren zurückgreifen [17, 32]. Grund genug im Rahmen dieser Arbeit das spannende und anscheinend unerschöpfliche Forschungsfeld der KI zu betrachten.

Wie allgemein bekannt, ist das Gesellschaftsspiel eine beliebte Form des spielerischen Wettstreits mit anderen Teilnehmern. Jedes Spiel definiert eigene Regeln und Teilnehmeranzahl, setzt anderes Spielmaterial voraus und variiert in Komplexität beim Spielablauf. Eines haben jedoch alle Gesellschaftsspiele gemeinsam: Es gibt klar definierte Abläufe und am Ende stehen Sieger und Verlierer fest - eine Umgebung, die sich für den Einsatz von KI eignet. Das zeigt sich unter anderem dadurch, dass Spiele laut R. Hadsell [33] von Google DeepMind[3] als Testumgebung für KI-Forschung verwendet werden, da sich anhand von Gesellschaftspielen unter anderem gut die Qualität einer KI bewerten lässt. Aus dieser Tatsache ergibt sich nun grob der Themenbereich dieser Arbeit, der im Folgenden weiter erläutert wird.

1.1 Künstliche Intelligenz und Gesellschaftsspiele

KI ist ein Teilgebiet der Informatik und E. Rich [13] fasst bündig zusammen: KI befasst sich mit der Frage, wie man Computer dazu bringen kann, Dinge zu tun, die Menschen im Moment noch besser können. KI ist äußerst interdisziplinär, denn es vereint viele Ergebnisse anderer Forschungsgebiete wie bspw. Logik, Statistik, Regelungstechnik, Bildverarbeitung und Neurobiologie [1, S. 12]. KI wurde mehrfach erfolgreich in Gesellschaftsspielen eingesetzt, oft unter großer medialer Begleitung. Meilensteine waren die Brettspiele Schach (Deep Blue 1997 [37]) und Go (Alpha-Go 2016 [26]), in denen KI das “super-human level” erreichte, heißt die Spielentscheidungen des Computers übertrumpfen die der weltbesten Spieler. Schach und Go haben dabei folgende Gemeinsamkeiten: Es sind klassische Brettspiele, die trotz simpler Spielregeln eine

[2] Ein Test, wobei ein Mensch erkennen muss ob es menschliches Handeln oder das eines Computers ist, benannt nach dem britischen Mathematiker Alan Turing [22].

[3] DeepMind ist eine Tochterfirma des Technologiekonzerns Google, welche sich der KI-Forschung widmet und in der Vergangenheit bereits mehrfach Meilensteine auf diesem Gebiet setzte [25–27, 34–36].

geradezu realistisch unlösbare Komplexität aufweisen. Auf dem $8x8$ großen Schachspielfeld sind innerhalb von 40 Zügen schätzungsweise 10^{115} bis 10^{120} verschiedene Spielverläufe möglich, wobei beim $19x19$ großen Spielfeld von Go noch wesentlich mehr existieren werden [38]. Schach als auch Go sind Spiele mit perfekter Information, was bedeutet, dass alle Handlungen und deren Konsequenzen für jeden Spieler nachvollziehbar sind und der gesamte bisherige Spielverlauf für jeden Spieler ersichtlich ist [1, S. 128, 7, S. 2]. Mit diesen Eigenschaften wäre es also theoretisch möglich immer die beste Strategie im Voraus zu berechnen, doch durch diese Komplexität war es für Computer zuvor unter Realzeitanforderungen nicht möglich auf dem Niveau der besten Expertenspieler zu entscheiden. Menschliche Spieler waren einer KI lange durch gelernte Erfahrung und Intuition überlegen. Seit AlphaGo [26] in 2016 den mehrfachen Weltmeister im Brettspiel Go schlagen konnte, dem Brettspiel mit der höchsten bekannten Komplexität, bekamen Spiele mit perfekten Informationen in der KI-Forschung zunehmend weniger Aufmerksamkeit als Testumgebung.

In Anwendungen des echten Lebens wie bspw. Verhandlungen, Finanzmärkten oder medizinischen Diagnosen sind niemals perfekte Informationen über die komplette Situation vorhanden. Jede Entscheidung wird unter Unsicherheit getroffen, da mit imperfekten Informationen bspw. die Konsequenzen vorher nicht vollständig bekannt sein können. Spiele mit imperfekten Informationen stehen also in engerem Bezug zu realen Problemen. Ein Spiel hat imperfekte Informationen, wenn die Informationen über andere Spieler nicht ausreichen, um mit Sicherheit zu wissen, welche Handlungsmöglichkeiten sie haben und nicht alle Spielzüge beobachtbar sind [7, S. 2, 10, S. 46]. Da Informationen fehlen, ist ein strategisches Handeln unter Unsicherheit notwendig. Aus diesem Grund eignen sich Spiele mit imperfekten Informationen besonders für KI-Forschung.

Gesellschaftsspiele eignen sich besonders für die KI-Forschung, da sie durch klar definierte Regeln und Abläufe eine Abstraktionsebene bieten, innerhalb der die KI agieren kann. Durch den Spielverlauf lassen sich die Entscheidungen der KI gut bewerten. Um ein Gesellschaftsspiel zu meistern, muss ein Spieler flexibel Strategien entwickeln und auf Strategien der Gegenspieler reagieren können. Dennoch gab es KI-Meilensteine in Spielen mit imperfekten Informationen: Für die Kartenspiele Heads-Up No-Limit Poker (DeepStack 2017 [29]) und Bridge (Nook 2022 [30]) gibt es KIs, die eine statistisch signifikante Zahl an Spielen gegen Experten gewinnen konnten. In 2019 konnte die KI Pluribus [39] in No-Limit Texas Hold'em Poker sogar gegen fünf menschliche Spieler gewinnen und damit ebenfalls das „super-human level" erreichen. Seit diese Meilensteine in klassischen komplexen Spielen mit imperfekten Informationen erreicht wurden, widmet sich die KI-Forschung allmählich noch komplexeren Spielen, oft mit sogar noch weniger verfügbaren Informationen, wie Recon Chess [40] und Diplomacy [41].

1.2 Problemstellung: Maschinelles Lernen in Kartenspielen mit imperfekten Informationen

Das maschinelle Lernen ist ein Teilbereich der KI [3, S. 3], welches sich mit der Frage befasst, wie sich Computerprogramme erstellen lassen, die ihre Leistung bei einer bestimmten Aufgabe durch Erfahrung verbessern [42, S. 17]. Anstatt, dass Regeln (zum Beispiel zur visuellen Unterscheidung von Objekten in Bildern) durch Menschen implementiert werden, findet das maschinelle Lernen durch Algorithmen selbstständig statt. Es gibt bereits viele KIs die in verschiedenen klassischen Kartenspielen eingesetzt werden und dabei oft unterschiedliche Lösungswege anwenden. Für diese Arbeit ergeben sich folgende Fragestellungen: Wie lässt sich ein Spiel lösen? Welche Herausforderungen entstehen dabei? Welche Ansätze gibt es im maschinellen Lernen? Welche Lösungswege nutzen KIs in komplexen Kartenspielen? Wie kann eine KI in einem Kartenspiel mit imperfekten Informationen mittels maschinellen Lernens sinnvolle Entscheidungen treffen? Wie lässt sich maschinelles Lernen in einem Kartenspielen mit imperfekten Informationen einsetzen? Diese Fragen werden in der folgenden Gliederung den jeweiligen Kapiteln zugeordnet.

1.3 Gliederung

Kapitel 2 und dessen Unterkapitel widmen sich der Frage, wie sich ein beliebiges Spiel formal beschreiben und lösen lässt. Dazu werden in Abschnitt 2.1 notwendige Grundlagen der Spieltheorie erläutert – eigentlich eine Methode der Wirtschaftswissenschaft, die sich jedoch ebenso in ihrem ursprünglichen Sinn, nämlich der Analyse von Gesellschaftsspielen, einsetzen lässt. Anhand von einfachen Spielen wird die Effektivität der spieltheoretischen Methoden aufgezeigt, doch in Abschnitt 2.2 auch die Hürden, die beim Lösen komplexer Spiele auftreten. In Abschnitt 2.3 werden konkrete, in der Spieltheorie anwendbare Algorithmen vorgestellt, die bereits erfolgreich in Spielen mit hoher Komplexität und imperfekten Informationen eingesetzt wurden.

Abschnitt 3 und dessen Unterkapitel widmen sich dem maschinellen Lernen: Einem sehr umfangreichen und aktivem Forschungsgebiet mit einer breit gefächerten Auswahl an Lösungsmöglichkeiten für sehr diverse Problemstellungen. Da es gemäß „No-Free-Lunch-Theorem“[4] [43] keine Lösung geben kann, die einer „Weltformel“ nahekommt, wird sich nicht auf einen Ansatz des maschinellen Lernens beschränkt, sondern das ganze Forschungsfeld in jeweils angemessener Ausführlichkeit erläutert und mit relevanten Anwendungsfällen begründet. Abschnitt 3.2 bis 3.4 erläutert die drei Lernverfahren, in denen maschinelles Lernen hauptsächlich unterteilt wird, welche sich wiederum stark in ihrem Lösungsweg unterscheiden.

[4] Ein Problem der Informatik, welches besagt, dass es keine Lösung geben kann, die auf jedes Problem gleichermaßen optimal anwendbar ist.

Abschnitt 4 widmet sich der finalen Frage: Wie lässt sich maschinelles Lernen in einem Kartenspielen mit imperfekten Informationen einsetzen? Bevor auf die konkrete Umsetzung erläutert wird, werden in Abschnitt 4.1 Anforderungen an die Umsetzung aufgestellt und wird der Rahmen der Umsetzung erläutert. Darauf folgt in Abschnitt 4.2 eine Anleitung für die technische Vorbereitung der Umsetzung, damit die Ergebnisse reproduzierbar sind. In Abschnitt 4.3 werden mehrere Algorithmen auf das Kartenspiel Leduc Poker [14, S. 2] (eine vereinfachte Version von Texas Hold'em Poker) angewandt und in Abschnitt 4.4 die Qualität der trainierten Modelle anhand der Nash-Konvergenz [44] miteinander verglichen.

Abschnitt 5 beinhaltet eine kritische Auseinandersetzung mit der Umsetzung und den Ergebnissen aus dem vorherigen Abschnitt 4 und im letzten Abschnitt 6 ein letztes zusammenfassendes Fazit.

2 Wie lässt sich ein Gesellschaftsspiel untersuchen und lösen?

Dieser Frage widmete sich 1928 John von Neumann [45] und bewies mit der Min-Max-Theorie, wie ein 2-Personen-Nullsummenspiel (bspw. Schach oder Vier-Gewinnt) formal gelöst werden kann. Dies bildete die Grundlage für die Spieltheorie, die John von Neumann zusammen mit Oskar Morgenstern 1944 in der Veröffentlichung „Theory of games and economic behavior“ [46] weiterentwickelte und so eine sinnvolle Verbindung von Spieltheorie und Wirtschaftswissenschaften erstellte. Die Spieltheorie ist mittlerweile eine anerkannte Methode der Wirtschaftswissenschaft und wird eingesetzt, um Entscheidungen und Strategien in Spielen zu analysieren. Das Spiel bezieht sich dabei auf eine Situation, in der mehrere Spieler beteiligt sind, die miteinander interagieren und versuchen eigene Ziele zu verfolgen [10, S. 1]. Die Spieltheorie wird in Wirtschaftswissenschaften eingesetzt, um bspw. einen Markteintritt und darauffolgende Entscheidungsmöglichkeiten eines Konkurrenten zu analysieren [10, S. 17] und hat auch bereits Anwendungen in anderen Wissenschaftsfeldern wie Biologie, Politik und Informatik gefunden. Doch die Spieltheorie lässt sich ebenso in ihrem ursprünglichen Sinn einsetzen, nämlich um Gesellschaftsspiele formal zu beschreiben und die Entscheidungsmöglichkeiten des Spielverlaufs zu analysieren. Aus diesem Grund ist es gängige Praxis für Künstliche Intelligenz, die für klassische Gesellschaftsspiele entwickelt wird, ebenso Grundlagen der Spieltheorie anzuwenden, wie bspw. das Aufstellen eines Spielbaums, dem Bewerten der Spielzüge und dem Suchen nach der optimalen Strategie. Da die Spieltheorie also eine Grundlage für die Analyse und Lösung eines Gesellschaftsspiels bietet, ist es angebracht, die Grundzüge in dieser Arbeit zu behandeln. Der Fokus liegt dabei auf Spielen mit imperfekten Informationen.

2.1 Grundbegriffe der Spieltheorie

„Die Spieltheorie ist eine Methodenwissenschaft“. Sie stellt Methoden zur Analyse von interdependenter Entscheidungsprobleme bereit. „Interdependente Entscheidungsprobleme liegen immer dann vor, wenn mehrere Akteure durch ihre individuellen Entscheidungen gegenseitig ihr Wohlergehen beeinflussen.“ Das Ergebnis hängt also nicht nur von eigenen Entscheidungen ab, sondern auch von den Entscheidungen anderer. In diesem Kontext werden diese Entscheidungsprobleme als ***Spiele*** bezeichnet und ihre Akteure als ***Spieler*** [7, S. 1-2]. Alle beteiligten Spieler beeinflussen sich untereinander durch ihre Entscheidungen im Spielverlauf. Dabei geht die Spieltheorie immer von mindestens zwei Spielern aus, denn für Spiele mit einem Spieler (oder auch Spiel gegen die Natur genannt), befasst sich die Entscheidungstheorie, eine Abspaltung der Spieltheorie, die jedoch in dieser Arbeit nicht weiter betrachtet wird [10, S. 1].

In einem Spiel verfolgen die Spieler immer Ziele, höchste Punkte, wie zum Beispiel den maximalen möglichen Gewinn bzw. die Zahl für sich selbst zu erreichen. Wenn ein Spiel mit der

Spieltheorie analysiert wird, ist es immer das Ziel eine Kombination aus Strategien für alle Spieler zu finden, die optimal ist. Eine Strategie ist ein vollständiger Spielplan für einen Spieler, der für das gesamte Spiel dessen Züge festlegt [10, S. 34]. Eine ***Strategie*** stellt sicher, dass der Spieler in Übereinstimmung mit seinen Zielen handelt und dabei seine Chancen auf die erzielte Auszahlung maximiert. Ein ***Zug*** ist eine einzelne Entscheidung eines Spielers zu einem bestimmten Zeitpunkt des Spiels [7, S. 7]. Je nach Spiel kann ein Zug auch aus mehreren Aktionen bestehen, also einzelnen Entscheidungen innerhalb eines Zugs, doch im Folgenden wird von einer Aktion in einem Zug ausgegangen, also einer einzelnen Entscheidung in einem Zug.

Jeder Spieler i entscheidet sich für eine eigene Strategie s_i aus seiner ***Menge an Strategien*** $s_i \in S_i$. Wenn die individuellen Strategien jedes Spielers kombiniert werden, entsteht eine ***Strategiekombination*** s, durch die ein kompletter möglicher Spielverlauf dargestellt werden kann. Die Strategiekombination ist Teil des Strategieraums S, der alle möglichen Strategiekombinationen enthält [10, S. 33-34][7, S. 10]. Eine Strategiekombination ist dann optimal, wenn sie insgesamt zu einem ***Gleichgewicht*** an Strategien (auch Nash-Gleichgewicht genannt) führt. Ein Gleichgewicht ist dann erreicht, „wenn keiner der Spieler individuell seine Strategie nachträglich noch ändern wollen würde, nachdem er erfahren hat, welche Strategien der oder die anderen Spieler gewählt haben.“ [7, 15; 21] Das Gleichgewicht ist die optimale Strategiekombination, in der jeder Spieler die höchstmögliche Auszahlung erreicht, die mit der Kombination der Strategien anderer Spieler möglich ist. Das bedeutet jedoch nicht, dass das Gleichgewicht zwangsläufig der bestmögliche individuelle Spielausgang für jeden Spieler ist. Das Nash-Gleichgewicht ist benannt nach John Forbes Nash, der das spieltheoretische Gleichgewicht in Strategiekombinationen erstmals definierte [47]. Das Nash-Gleichgewicht und Gleichgewicht werden in der Spieltheorie als Synonyme für den gleichen Sachverhalt verwendet, doch im Folgenden wird es nur als Gleichgewicht bezeichnet.

„Jede Strategiekombination führt zu einem ganz bestimmten Verlauf des Spiels und damit auch zu einem ganz bestimmten Spielausgang. Dieser Spielausgang hat für die Spieler Konsequenzen, die sie positiv oder negativ bewerten.“ [7, S. 12] Dafür wird eine ***Auszahlung*** verwendet. Sie steht für eine Konsequenz, die aus dem Spielverlauf für einen Spieler entsteht und sich in Zahlen quantifizieren lässt, wie bspw. den erwirtschafteten Gewinn oder die erreichte Punktzahl am Ende des Spiels. Die Spieltheorie geht von einem rational handelnden Spieler aus, der versucht die eigene Auszahlung zu maximieren. Um die Auszahlung numerisch ausdrücken zu können, wird eine ***Auszahlungsfunktion*** $u_i(s)$ (Auszahlung für Spieler i wenn Strategie s_i gespielt wird) aufgestellt, die beschreibt, wie sich die finale Auszahlung eines Spielverlaufs zusammensetzt [10, S. 4]. Je nach Kontext des Spiels wird die Auszahlungsfunktion oft auch Nutzenfunktion, Gewinnfunktion oder Pay-Off-Funktion genannt. Die Auszahlungsfunktion eines nicht-kooperativen Zwei-Personen-Nullsummenspiels, dass die erreichte Punktzahl angibt, könnte bspw. so formuliert werden:

$$u_i(s) = Punkte_i - Punkte_j$$

Formel 2.1 In diesem Beispiel ergibt sich die finale Auszahlung durch die Auszahlungsfunktion $u_i(s)$ für die Strategie s_i aus der Differenz der Punktzahl von Spieler i und dem zweiten Spieler j [10, S. 4].

In der Spieltheorie wird zwischen kooperativen Spielen und nicht-kooperativen Spielen unterschieden, die jeweils unterschiedliche Methoden zur Analyse des Spiels verwenden und mit anderen Vorrausetzungen arbeiten. In ***kooperativen*** Spielen können Spieler durch Absprache bindende Verträge schließen und eine optimale Strategiekombination finden, die die Auszahlung aller Spieler maximiert [10, S. 6]. Dem kooperativen Spiel steht das ***nicht-kooperative*** Spiel gegenüber, bei dem jeder Spieler lediglich die eigene Auszahlung maximiert und unabhängig voneinander handelt [7, S. 3]. Fokus dieser Arbeit sind nicht-kooperative Kartenspiele. Auch wenn es bspw. im Kartenspielklassiker Skat Teil des Spiels ist, dass zwei der drei Spieler kooperativ spielen, ist dennoch die Vorrausetzung für ein kooperatives Spiel (im spieltheoretischen Sinne) in diesem Fall nicht gegeben, da keine Absprache zwischen den Spielern erlaubt ist [20]. Daher wird im Folgenden auf den Bereich der kooperativen Spiele in der Spieltheorie verzichtet, da er für den Fokus dieser Arbeit nicht relevant ist.

Der zeitliche Ablauf eines Spiels ist relevant für die Analyse. Wenn Spieler ihre Strategie simultan bzw. unabhängig voneinander wählen, dann hat das Spiel keine explizite Reihenfolge von Zügen und läuft ***statisch*** ab. Ein Beispiel für ein statisches Spiel ist Schere-Stein-Papier, bei dem beide Spieler gleichzeitig entscheiden müssen und anschließend die Auszahlung erhalten. Wenn die Reihenfolge der Züge jedoch bekannt ist und die Züge zeitlich nacheinander ablaufen, handelt es sich um ein ***dynamisches*** Spiel (auch sequenzielles Spiel genannt). In einem dynamischen Spiel wählen die Spieler ihre Strategie nacheinander [7, S. 2]. Ein Beispiel für ein dynamisches Spiel wäre das Kartenspiel Mau-Mau: Es wird abwechselnd eine Karte ausgespielt und der Spieler muss immer auf die Strategie des vorherigen Spielers (die ausgespielte Karte) reagieren. Das dynamische Spiel trifft auf den Großteil der klassischen Kartenspiele zu, weil die Züge nacheinander ablaufen und die Reihenfolge der Züge ebenso bekannt ist.

> „Die Lösung eines Spiels hängt stark davon ab, welche Informationen den einzelnen Spielern zur Verfügung stehen.“ [10, S. 45]

Die Informationen, die einem Spieler bei seinen Entscheidungen zur Verfügung stehen, sind relevant, da sie die optimale Strategie beeinflussen. Wenn alle Spieler „die Konsequenzen jedes möglichen Spielausgangs für alle anderen Spieler“ kennen und sich so die Perspektive jedes Spielers nachvollziehen lässt, dann hat das Spiel ***vollständige Informationen*** [7, S. 2]. Bei einem Spiel mit vollständigen Informationen sind also die Strategiemengen S_i und die Auszahlungsfunktion $u_i(s)$ aller anderen Spieler bekannt. ***Perfekte Informationen*** liegen vor, wenn zudem jedem Spieler immer

der komplette Spielzustand bekannt ist. Schach, Dame und Go sind Spiele mit perfekter Information, da jeder Zug beobachtbar ist [1, S. 128]. Im Gegensatz dazu sind bei Spielen mit ***unvollständigen Informationen*** die Informationen über andere Spieler nicht ausreichend, um mit Sicherheit zu wissen, wie sie ihre individuellen Spielausgänge bewerten [7, S. 2]. In einem Spiel mit unvollständigen Informationen sind also die Strategiemengen S_i und die Auszahlungsfunktion $u_i(s)$ anderer Spieler nicht bekannt. ***Imperfekte Informationen*** liegen vor, wenn zusätzlich nicht alle Züge aller Spieler beobachtbar sind [10, S. 46]. „Viele Kartenspiele, wie zum Beispiel Skat, sind nur teilweise beobachtbar, denn der Spieler kennt die Karten des Gegners nicht oder nur teilweise.“ [1, S. 128] Diese unterschiedlichen Informationslevel sind nicht zwangsläufig für alle Spieler gleich. Es ist ebenso möglich, dass ein Spieler vollständige Informationen hat, während der andere Spieler imperfekte Informationen hat. Die Zugänglichkeit von Information und Beobachtbarkeit der Spielzüge hängt dabei vom Spiel selbst ab.

Um ein Spiel zu lösen, ist es hilfreich, es in eine geeignete Darstellungsform zu übertragen. Dabei gibt es zwei Darstellungsformen: die ***Auszahlungsmatrix*** und der ***Spielbaum***. Mit der Auszahlungsmatrix (auch strategische Form oder Normalform genannt) werden alle Strategiekombinationen und die Auszahlungen, die aus jeder Strategiekombination resultieren, in einer Tabelle angeordnet [7, S. 12-15]. In Abbildung 2.1 wurde eine Auszahlungsmatrix anhand des Gefangenendilemmas aufgestellt. Das Gefangenendilemma ist ein bekanntes Problem in der Spieltheorie, in dem zwei Gefangene gleichzeitig im Verhör sitzen und beide vor der Wahl stehen, ob sie gestehen und damit den Komplizen verraten oder schweigen. Die Gefangenen können sich dabei nicht absprechen [10, S. 2]. Es handelt sich bei dem Gefangenendilemma um ein nicht-kooperatives statisches Spiel mit zwei Spielern. Die Strategien s_{11} und s_{12} von Spieler 1 werden vertikal eingetragen, die Strategien s_{21} und s_{22} von Spieler 2 horizontal. Die Zellen enthalten die Auszahlungen für die Strategiekombinationen von Spieler 1 und Spieler 2. Mit der Auszahlungsmatrix in Abbildung 2.1 lässt sich nun gut erkennen: Schweigt ein Spieler und ein anderer sagt aus, bekommt der schweigende Spieler zehn Jahre Gefängnis, während der Komplize, der ihn verraten hat, Kronzeuge wird und keine Gefängnisstrafe bekommt. Schweigen beide, bekommen beide ein Jahr Gefängnis. Gestehen beide, bekommen beide acht Jahre Gefängnis.

	Spieler 2	
Spieler 1	schweigen s_{21}	gestehen s_{22}
schweigen s_{11}	1 Jahr für Spieler 1 1 Jahr für Spieler 2	10 Jahre für Spieler 1 Freiheit für Spieler 2
gestehen s_{12}	Freiheit für Spieler 1 10 Jahre für Spieler 2	8 Jahre für Spieler 1 8 Jahre für Spieler 2

Abbildung 2.1 Auszahlungsmatrix für das Gefangenendilemma [10, S. 3]

Mit dieser Darstellung lässt sich für dieses Spiel das Gleichgewicht gut ermitteln. Jeder Spieler versucht seine Auszahlung zu maximieren, also die geringste Zeit im Gefängnis zu erhalten. Um ein Gleichgewicht für dieses Spiel zu finden, wird nun jede Strategiekombination mit deren Alternativen

anhand der Auszahlung verglichen. Angenommen Spieler 1 gesteht nicht (Strategie s_{11}) und geht davon aus, dass Spieler 2 auch schweigt (Strategie s_{21}). In diesem Fall hätte Spieler 1 eine Auszahlung von einem Jahr Gefängnis, aber die Alternative zu gestehen mit der Auszahlung von nur drei Monaten Gefängnis. Um seine Auszahlung zu maximieren, wird sich Spieler 1 für das Gestehen (Strategie s_{12}) entscheiden. Dafür wird in Abbildung 2.2 ein roter Pfeil von der Strategiekombination $\{s_{11}; s_{21}\}$ zur anderen Strategiekombination $\{s_{12}; s_{21}\}$ eingezeichnet. Das bedeutet, dass dieser Spieler statt der einen Strategiekombination die andere Strategiekombination wählen würde. Wenn dieser Prozess für jeden Spieler und jede Strategiekombination wiederholt wird, lässt sich durch die roten Pfeile gut erkennen, dass sich Spieler 1 und Spieler 2 immer für die Strategie Gestehen entscheiden würden. Es besteht also ein Gleichgewicht bei der Strategiekombination $\{s_{12}; s_{22}\}$[7, S. 41]. Da beide Gefangene zum gleichen Zeitpunkt entscheiden müssen (statisches Spiel), ist dies zusätzlich ein Spiel mit imperfekten Informationen, da beide Spieler entscheiden, ohne zu wissen, welche Strategie der andere Spieler wählt.

	Spieler 2	
Spieler 1	schweigen s_{21}	gestehen s_{22}
schweigen s_{11}	1 Jahr für Spieler 1 1 Jahr für Spieler 2	10 Jahre für Spieler 1 Freiheit für Spieler 2
gestehen s_{12}	Freiheit für Spieler 1 10 Jahre für Spieler 2	8 Jahre für Spieler 1 8 Jahre für Spieler 2

Abbildung 2.2 Auszahlungsmatrix für das Gefangenendilemma mit dem eingezeichneten Gleichgewicht bei (s_{12}, s_{22}), wenn beide Spieler die Strategie Gestehen ausspielen.

Die Strategie zu gestehen ist in diesem Fall für beide Spieler eine ***dominante Strategie***. Eine Strategie ist dominant, wenn sie immer die ***beste Antwort*** ist, unabhängig davon welche Strategie die anderen Spieler spielen [7, S. 28]. Für Außenstehende scheint das Gleichgewicht des Gefangenendilemmas sicher nicht als die beste Lösung, denn beide nehmen die Auszahlung von acht Jahren Gefängnis in Kauf. Würden beide schweigen, könnten sie beide ihre Auszahlung auf ein Jahr Gefängnis verbessern. Daher spricht man in diesem Fall von einem ***ineffizienten Gleichgewicht*** und bei diesem Spiel von einem Dilemma.

Ein Problem bei der Auszahlungsmatrix ist, dass diese Darstellungsform nur bei wenigen Spielern und kleinem Strategieraum übersichtlich und damit auch anwendbar bleibt. Ein dynamisches Spiel kann wegen der zeitlichen Reihenfolge von Entscheidungen nicht mehr in einer Auszahlungsmatrix dargestellt werden. Aus diesem Grund ist es für dynamische Spiele sinnvoll, statt der Auszahlungsmatrix die Spielbaum Darstellung zu verwenden [7, S. 88]. Der ***Spielbaum*** (auch dynamische oder extensive Form) ist eine hierarchische Datenstruktur der Spieltheorie, mit der die sequenzielle Struktur eines Spiels (bspw. die abwechselnden Züge der Spieler) erfasst wird. „Jeder Zug eines Spielers wird durch einen Knoten dargestellt, an dem der Spieler zwischen verschiedenen Ästen (seinen Handlungsalternativen) wählen kann.“ [10, S. 14] Die Abfolge ist in dieser Darstellungsform von oben nach unten, also vom Startknoten zum Endknoten. Die Endknoten stehen für die Auszahlung unter dieser Strategiekombination [7, S. 89]. Um den Spielbaum zu

veranschaulichen, nehmen wir das Gefangenendilemma, doch stellen uns vor, es wäre ein dynamisches Spiel mit vollständigen Informationen, in dem erst Spieler 1 und danach Spieler 2 seine Strategie wählt (siehe Abbildung 2.3). Wenn Spiel 2 an der Reihe ist, ist ihm also bekannt, welche Strategie Spieler 1 gewählt hat.

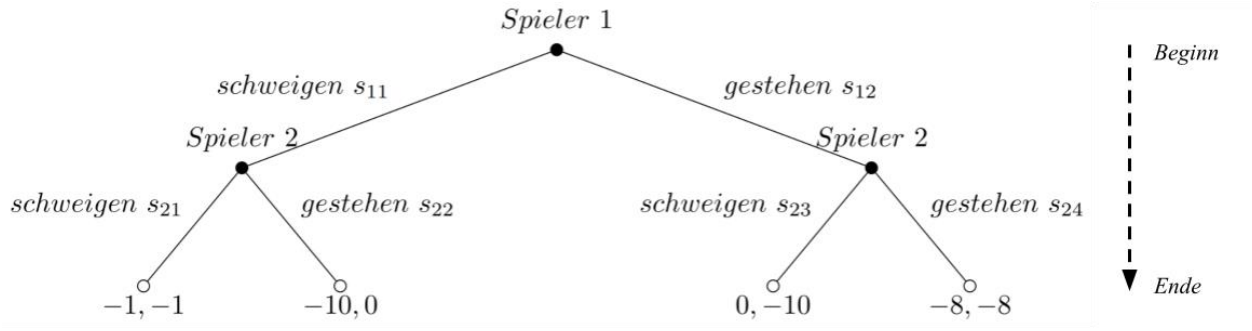


Abbildung 2.3 Das Gefangenendilemma als dynamisches Spiel, dargestellt in einem Spielbaum.

Spieler 1 beginnt das Spiel und hat die Wahl zu schweigen oder zu gestehen. Da Spieler 2 in diesem dynamischen Spiel auf die Entscheidung von Spieler 1 reagiert, befindet er sich bei seiner Entscheidung im linken oder rechten Knoten. In Abbildung 2.3 wurden die Auszahlungen nur als negative Zahlenwerte (Gefängnisdauer ist negativ) eingetragen, wobei die erste Zahl für die Auszahlung von Spieler 1 steht und die zweite für die von Spieler 2. Um das Gleichgewicht zu finden, kann in einem dynamischen Spiel mit vollständigen Informationen die Rückwärtsinduktion angewandt werden. Dafür wird das Spiel rekursiv vom Endknoten aus in ***Teilspiele*** zerlegt, in denen jeweils die ***teilspielperfekte Strategie*** bestimmt wird. Jedes Teilspiel wird dabei, wie ein selbstständiges Spiel behandelt, in dem davon ausgegangen wird, dass die Spieler rational handeln und ihre Auszahlung in jedem Teilspiel maximieren möchten. Im Teilspiel 1 hat Spieler 2 kein Interesse die Strategie schweigen s_{21} zu wählen, weil er mit gestehen s_{22} eine bessere Auszahlung erreichen kann (auf Abbildung 2.4: roter Pfeil), daher wird der Ast für diese Strategie beschnitten (auf Abbildung 2.4: doppelte rote Linie) [7, S. 98]. Genauso in Teilspiel 2. Mit dieser Beschneidung des Spielbaums ist es für Spieler 1 ersichtlich, dass Spieler 2 immer die gestehen Strategien (s_{22} oder s_{24}) spielen wird. Dadurch wird Spieler 1 ebenso seine Strategie schweigen s_{11} beschneiden, denn er kann nur mit der Strategiekombination $\{s_{12};s_{24}\}$ seine Auszahlung maximieren. Da in jedem Teilspiel die teilspielperfekte Strategie gewählt wurde, ist damit das Gleichgewicht bei der Strategiekombination $\{s_{12};s_{24}\}$ ermittelt. In diesem Fall handelt es sich um ein ***teilspielperfektes Gleichgewicht*** [7, S. 99]. Auch hier liegt wieder ein ineffizientes Gleichgewicht vor, da beide Spieler mit der Strategiekombination $\{s_{11};s_{21}\}$ eine höhere Auszahlung erreichen könnten, doch in nicht-kooperativem Spielen sind keine Absprachen möglich. Wenn Spieler 1 die Strategie gestehen s_{12} spielt, ist für Spieler 2 die Strategie gestehen s_{24} die beste Antwort. Daher handelt es sich wieder um eine dominante Strategie.

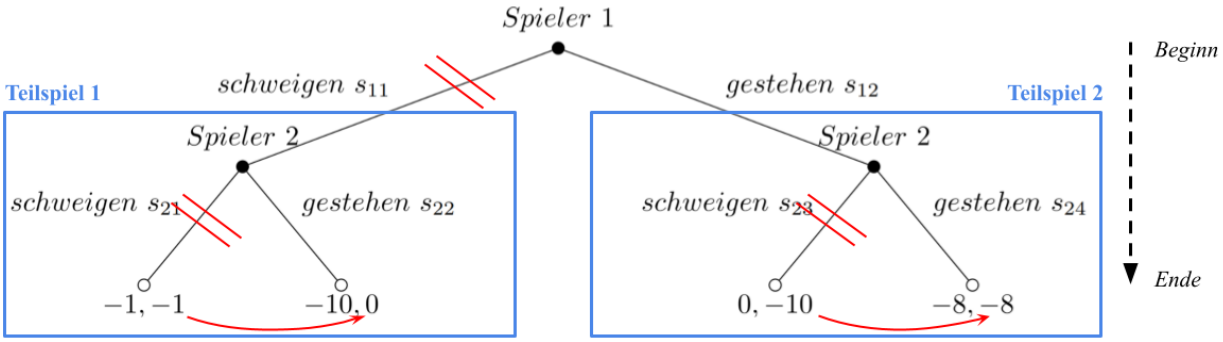


Abbildung 2.4 Das Gefangenendilemma als dynamisches Spiel mit der Beschneidung des Spielbaums. Durch Rückwärtsinduktion wird das Spiel in Teilspiele zerlegt und das teilspielperfekte Gleichgewicht gefunden.

In einem nicht-kooperativem dynamischen Spiel mit unvollständigen Informationen bzw. imperfekten Informationen ist dem Spieler nicht bekannt, in welchem Knoten er sich zum beliebigen Zeitpunkt des Spiels befindet. Der Spieler kann bei unvollständigen Informationen nur durch die beobachtbaren Züge anderer Spieler Vermutungen aufstellen, mit welcher Wahrscheinlichkeit er sich in welchem Knoten des Spielbaums befindet [7, S. 159]. Von einem separierenden Gleichgewicht spricht man, wenn aus den beobachtbaren Handlungen der Mitspieler erkennbar ist, in welchem Knoten sich ein Spieler befindet und wie er sich als Nächstes entscheiden wird. Falls das nicht möglich ist, spricht man von einem Pooling-Gleichgewicht und der Spieler muss ***Beliefs (deutsch Wahrscheinlichkeitseinschätzungen)*** über die Strategie anderer Spieler aufstellen, da sie für ihn nicht beobachtbar sind [7, S. 167-168]. Ein Spieler verwendet den Belief, um die Wahrscheinlichkeit anzugeben, mit der er glaubt, dass andere Spieler diesen Zug wählen werden, und formuliert seine Strategie auf der Grundlage des Belief. Ein dynamisches nicht-kooperatives Spiel mit unvollstänigen bzw. imperfekten Informationen lässt sich zwar ebenso mittels der vorgestellten Rückwertsinduktion lösen, doch bei der Beschneidung des Spielbaums werden zusätzlich die Beliefs der Spieler berücksichtigt [7, S. 166-167]. Für ein dynamisches Spiel mit unvollständigen bzw. imperfekten Informationen wird der Spielbaum etwas angepasst: Zwischen dem Knoten einer Ebene wird eine gestrichelte Linie eingezeichnet, um darzustellen, dass keinem der Spieler bekannt ist, in welchem Knoten sich das Spiel aktuell befindet, da diese Handlung nicht beobachtbar ist [7, S. 109].

Die Spieltheorie bietet sehr umfangreiche Methoden zur Lösung und Analyse von Spielen, wobei nochmals explizit darauf hingewiesen werden muss, dass die vorgestellten Grundlagen der Spieltheorie nur eine kleine Auswahl darstellen, die für den Fokus dieser Arbeit von Bedeutung sind.

2.2 Hürden komplexer nicht-kooperativer dynamischer Spiele

Das vorgestellte Gefangenendilemma ist selbst als dynamisches Spiel simpel, da die Handlungsalternativen auf Gestehen und Schweigen beschränkt sind und das Spiel nach jeweils einem Zug terminiert. Doch für viele Gesellschaftsspiele trifft das nicht zu, auch wenn sie auf den

ersten Blick leicht lösbar erscheinen: Tic-Tac-Toe [48] hat ein Spielbrett mit 3×3 Feldern, also maximal 9! (entspricht 362.880) möglichen Kombinationen (auch obere Schranke genannt), in denen die Züge beider Spieler angeordnet werden können. Dies ist eine sehr große Zahl, die zwar vollständig in einem Spielbaum dargestellt werden kann, aber wegen der großen Anzahl von Verzweigungen nur mit großem Aufwand gelöst werden kann. Es muss also versucht werden, die ***Spielbaumkomplexität*** zu reduzieren. Wenn in Tic-Tac-Toe alle nicht regelkonformen Züge aus dem Spielbaum entfernt werden und man die Symmetrie von Spielzügen berücksichtigt, lässt sich die Spielbaumkomplexität um etwa 92% senken (auf 26.830 mögliche Spielzüge) [49]. Im Fall von Vier-Gewinnt [50], einem Gesellschaftsspiel für zwei Personen mit 7×6 Feldern, ist die geschätzte realistische Spielbaumkomplexität bei etwa 4.06×10^{31} [51] und so nur noch schwer mit den vorgestellten Methoden allein vollends analysierbar. Wie man an diesen beiden Beispielen leicht erkennt, explodiert die Spielbaumkomplexität für Spiele regelrecht, auch wenn das Spielfeld, im Vergleich von Tic-Tac-Toe zu Vier-Gewinnt, nur etwas größer ist. Dieses Verhältnis von steigender Spielbaumkomplexität ist in vielen Fällen ebenso bei Kartenspielen mit imperfekten Informationen gegeben: Bspw. das Kartenspiel Mau-Mau [52] hat mit einem typischen Skatblatt eine obere Schranke von mindestens 32! (etwa $2{,}63 \times 10^{35}$). Aus diesem Grund muss die Herangehensweise zum Lösen eines komplexen Spiels angepasst werden.

Zuvor wurde der Spielbaum durch die Methode der Rückwärtsinduktion untersucht und so Spielzüge gestrichen, die aufgrund der Auszahlung nicht in Frage kommen. Bei komplexen Spielen geht man jedoch davon aus, dass der Spielbaum so groß ist, dass die finalen Auszahlungen aller Endknoten nur mit großem Aufwand berechenbar sind, wie zuvor grob gezeigt wurde. Daher ist die Rückwärtsinduktion, die vom Endknoten ausgeht, in dieser Form nicht anwendbar. Um die Entscheidungsmöglichkeiten trotzdem zu bewerten, wird eine ***Heuristik*** verwendet, die die Auszahlung für einen beliebigen Knoten im Spielbaum schätzt, auch wenn dieser kein Endknoten ist. Es wird also eine gut geschätzte Lösung einer perfekten Lösung vorgezogen. Eine gut gewählte Heuristik kann den Rechenaufwand beim Lösen eines entscheidbaren Problems dramatisch reduzieren. Dies gilt jedoch nicht für unentscheidbare Probleme, die unlösbar sind und so eine unendliche Tiefe im Spielbaum haben [1, S. 117-118]. Im Folgenden werden die unentscheidbaren Probleme nicht weiter betrachtet.

Um eine Heuristik mathematisch zu modellieren, wird eine ***heuristische Bewertungsfunktion*** für die Zustände der Knoten verwendet. Ziel ist es, durch diese Bewertung schneller zu einer Lösung zu finden [1, S. 117]. Es wird zwischen ***Knoten*** und ***Zustand*** unterschieden: „Der Knoten enthält den Zustand und noch weitere für die Suche relevante Informationen wie zum Beispiel die Tiefe im

Suchbaum[5] und die heuristische Bewertung des Zustands.“ [1, S. 117] Wenn die Bewertung des Zustands gut ist, wird dieser Knoten in der Suche priorisiert. Ist die Bewertung nicht gut, wird er vernachlässigt oder sogar ignoriert [1, S. 117-118]. Die heuristische Bewertung kann sich bspw. auf den Spielzustand beziehen: Beim Schach würde sie in die Heuristik einfließen, ob der nächste Zug zum Schachmatt führt oder ob eine Figur des Spielers geschlagen wird.

Um den komplexen Spielbaum zu durchsuchen, wird ein ***heuristisches Suchverfahren*** (auch informierte Suche genannt) verwendet, also ein Suchalgorithmus, der vom Startknoten ausgehend rekursiv durch Spielbaum iteriert und mittels Heuristiken den besten nächsten Knoten oder im optimalen Fall sogar die beste Antwort findet [1, S. 117-118]. Es gibt viele heuristische Suchverfahren, die auf einen komplexen Spielbaum angewandt werden können, die jeweils mit Vorteilen und Einschränkungen verbunden sind. Für jedes komplexe Spielproblem muss daher abgewogen werden, welcher für den jeweiligen Fall sinnvoll ist. Der Minimax-Algorithmus mit Alpha-Beta-Pruning erstellt Heuristiken für alle Spielzustände im Voraus bis zu einer festgelegten Tiefe und verwendet eine Kombination aus Breiten- und Tiefensuche, um den Spielbaum, ähnlich wie die Rückwärtsinduktion, rekursiv zu beschneiden. Einschränkungen dieses Suchalgorithmus sind jedoch, dass er auf vollständigen Informationen im Spiel angewiesen ist und bei hohem Verzweigungsfaktor nicht effizient bleibt [1, S. 129]. Die IDA*-Suche (engl. iterative deepening A*) ist ein Suchalgorithmus, welcher eine wiederholende heuristische Tiefensuche einsetzt, bis eine optimale Lösung gefunden wird und ist dabei im Gegensatz zum Minimax-Algorithmus speichereffizient, da unter anderem nur der aktuelle Pfad gespeichert und bewertet wird. [1, S. 125]. Dieses Suchverfahren hat jedoch die Einschränkung, dass der Erfolg sehr von der zuvor definierten heuristischen Bewertungsfunktion abhängig ist. Aufgrund der Einschränkungen dieses Suchalgorithmus (bspw. verdeckte Informationen der Gegenspieler), kann jedoch nicht immer eine optimale Lösung gefunden werden. Der IDA*-Algorithmus ist zwar in Spielbäumen mit unvollständigen Informationen einsetzbar, aber eben nicht immer optimal.

Eine weitere Hürde komplexer dynamischer Spiele besteht im ***Exploration-Exploitation-Dilemma*** (deutsch: Erkunden oder Verwerten Dilemma) [1, S. 367-368], einem grundlegenden Problem in Entscheidungssituationen: Auf die Heuristik vertrauen (bisheriges Wissen) oder die Alternativen weiter untersuchen (neues Wissen), denn die heuristische Bewertung der Alternativen könnte eventuell noch nicht optimal sein und eine Alternative sogar eine höhere Auszahlung bedeuten. Dieses Problem wird bspw. im Rahmen des Multi-Armed-Bandit-Problems [9] mit vielen unterschiedlichen Lösungsansätzen untersucht und dabei mittels der ***Regret*** -(deutsch Bedauern: Differenz zwischen optimaler Auszahlung und erreichter Auszahlung [18, S. 5]) Metrik verglichen.

[5] Der Spielbaum ist eine Spezialisierung des Suchbaums. Der Suchbaum ist wie der Spielbaum eine Baum-Datenstruktur, die in sequenziellen Entscheidungsproblemen verwendet wird. Der Spielbaum beschränkt sich jedoch auf die Spieltheorie. Die erklärten Begriffe treffen auf Spielbaum als auch Suchbaum zu.

Ein weiteres Problem besteht darin, dass sich allein von einem Spielausgang nicht ablesen lässt, wie entscheidend jeder einzelne Zug für das Ergebnis war. Wenn bspw. eine Strategie für Schach gelernt wird, kann erst am Ende der Partie bewertet werden, ob diese gesamte Strategie zum Sieg geführt hat oder nicht, aber nicht wie wichtig jeder Zug dafür war. Im Fall von Schach lässt sich der individuelle Zug bspw. noch dadurch bewerten, ob eine Figur mit diesem Zug geschlagen wird. Auf Spiele mit imperfekten Informationen ist dies nicht immer anwendbar, da die Konsequenzen des eigenen Handelns nicht zwangsläufig beobachtbar sind. Dieses Problem ist als das ***Credit Assignment-Problem*** bekannt und gilt ebenso für das reinforcement learning (siehe Abschnitt 3.4) [1, S. 354, 53, 20–28].

2.3 Ein komplexes nicht-kooperatives dynamisches Spiel mit imperfekten Informationen lösen (State of the Art)

Das Finden eines (Nash-)Gleichgewichts in einem Spiel mit gemischten Strategien wird in der Komplexitätsklasse der NP-Probleme als PPAD-komplett eingestuft [54, S. 516]. Es gilt als schwierig, ein spieltheoretisches Gleichgewicht der Strategien in einem allgemeinen Spiel zu ermitteln. Vielversprechend erscheint dabei der ***Monte-Carlo-Tree-Search (MCTS)***: Ein Suchalgorithmus, der in komplexen Spielen mit imperfekten Informationen bereits mehrfach erfolgreich angewendet wurde und bspw. in Kombination mit maschinellem Lernverfahren und auch in AlphaGo (2016) [26] als Suchalgorithmus für den Spielbaum eingesetzt wurde. Der MCTS ist dazu ein Any-time-Algorithmus, der zu einem beliebigen Zeitpunkt unterbrochen werden kann und die bisherige Bewertung der Züge ausgibt, was es besonders interessant für Anwendungen mit Realzeitanforderung macht. Durch diese Eigenschaft ist der MCTS in der Lage mit längerer Bearbeitungszeit oder höherer Rechenkapazität immer ein besseres Ergebnis zu erzielen [16, S. 5-9]. Aus diesen Gründen wird der MCT folgend genauer betrachtet.

Ein weiterer vielversprechender Lösungsansatz für komplexe dynamische Spiele mit imperfekten Informationen ist die ***Counterfactual Regret Minimization (CFR)*** (deutsch kontrafaktisches Minimieren des Bedauerns), ein spieltheoretischer Algorithmus, der bereits erfolgreich in kleineren Varianten des Kartenspiels Poker (bspw. Kuhn-Poker) eingesetzt wurde. Der CFR-Algorithmus konvergiert bei vielen Iterationen im Selbstspiel zu einem Gleichgewicht oder nähert sich diesem zumindest [55]. Dies wird zum Anlass genommen, folgend auch auf den CFR-Algorithmus einzugehen.

2.3.1 Monte-Carlo-Tree-Search

Der MCTS ist ein heuristischer Suchalgorithmus, der zur Klasse der Monte-Carlo-Verfahren gehört: Methoden die unter anderem Zufallsverfahren verwenden, um eine optimale Lösung zu

finden oder mathematische Probleme zu approximieren und bspw. bei Problemen angewandt werden, bei denen analytische Ansätze schwierig oder nicht umsetzbar sind [16, S. 4]. Wie zuvor beschrieben, trifft das auf Spiele mit einer hohen Spielbaumkomplexität zu. MCTS wiederholt ein Verfahren aus vier Phasen, um den Spielbaum zu untersuchen (siehe Abbildung 2.5): Knoten auswählen mittels Heuristik (Auswahl), Knoten erweitern (Erweiterung), vom Ausgangsknoten bis zum Endknoten mittels zufälliger Züge simulieren (Simulation) und die Auszahlung des Endknotens für die Heuristik des Ausgangsknoten nutzen (Backpropagation) [16]. Das besondere an MCTS ist, dass es nicht auf eine zuvor definierte heuristische Bewertungsfunktion angewiesen ist (im Gegensatz zu Minimax und IDA*), sondern die Bewertung der Knoten durch die Simulation eines Strangs bis zum Endknoten erstellt wird. Die Auszahlung des zufällig erreichten Endknotens fließt in die Heuristik des Ausgangsknoten ein. Darin liegt der große Vorteil der Monte-Carlo-Verfahren. Mit dieser zufälligen Simulation des MCTS ist kein Expertenwissen über das Spiel notwendig, um eine gute heuristische Bewertungsfunktion zu definieren. MCTS lässt sich so in komplexen Spielbäumen sogar mit komplexen Spielregeln einsetzen, da die Heuristik durch Simulation und Wiederholung verfeinert wird [2, S. 185-188].

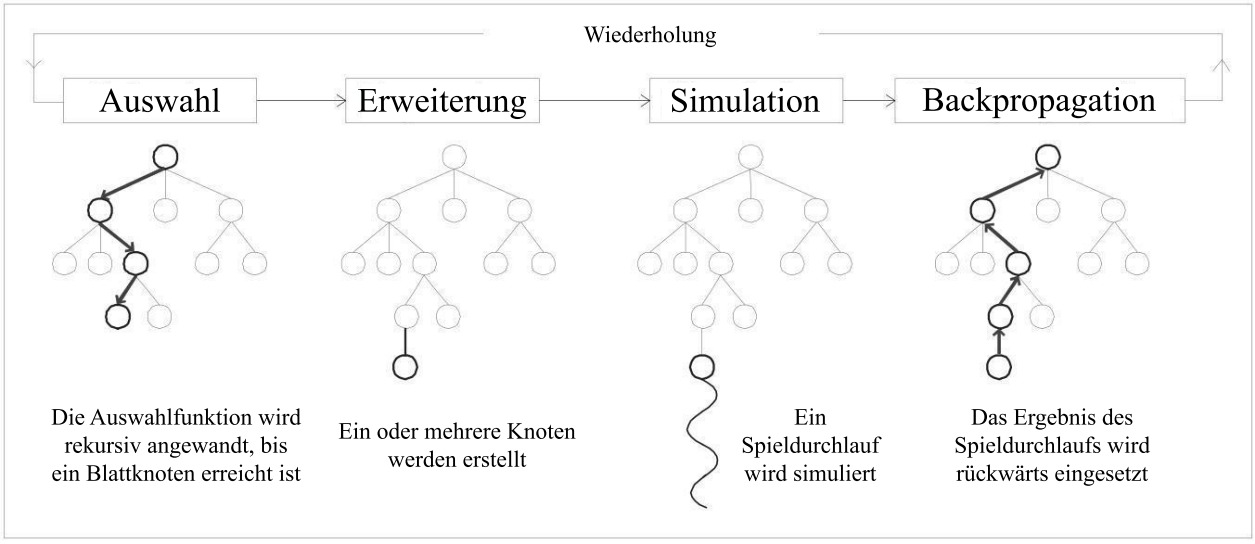


Abbildung 2.5 Die vier wiederholenden Phasen des Monte-Carlo-Tree-Search Algorithmus [55]

Nochmals ein Hinweis zum besseren Verständnis: Komplexe Spielbäume sind in vielen Fällen nur mit steigendem Aufwand untersuch- und speicherbar, daher wird die Heuristik zur effektiven Suche verwendet. Zusätzlich geht MCTS (wie auch andere heuristischen Suchverfahren) davon aus, dass nicht jeder Knoten des Spielbaums bekannt ist und generiert schrittweise die Kindknoten des Spielbaums an den Knoten mit der höchsten heuristischen Bewertung. Für die erste Phase (Auswahl) nutzt MCTS die UCB1-Formel als heuristische Bewertungsfunktion der Knotenzustände S (siehe Formel 2.2). Die UCB1-Formel errechnet den Upper Confidence Bound, also eine Schätzung der oberen Grenze für die erwartete Auszahlung eines Knotenzustands S_i und berücksichtigt dabei

ebenfalls die Unsicherheit dieser Schätzung [19, S. 237-243]. Der Blattknoten i, der in der ersten Phase (Auswahl) die höchste Bewertung durch die UCB1-Formel erhält, wird in der zweiten Phase (Erweiterung) um seine Kindknoten erweitert. In Phase drei (Simulation) wird von einem der Kindknoten j ausgehend der ganze Spielbaum bis zu einem Endknoten durchsimuliert, mittels zufällig gewählter Züge. Der durchsimulierte Spielverlauf wird anschließend verworfen und die erreichte Auszahlung in Phase vier (Backpropagation) in allen Knotenzuständen der Elternknoten gespeichert. Nach dieser Phase wird der Prozess wiederholt und die finalen Auszahlungen aus allen bisher simulierten Spielverläufen fließen in die UCB1 Formel für die heuristische Bewertung ein [16, S. 5-9].

$$UCB1(S_i) = \overline{x_j} + \sqrt{\frac{2\,ln\,n}{n_j}}$$

Formel 2.2 die UCB1 Formel [19, S. 237], heuristische Bewertungsfunktion des MCTS-Algorithmus

S_i Zustand des Knoten i

$\overline{x_j}$ Durchschnittliche Auszahlung des Zustands j

n_j Anzahl der Wiederholungen für Zustand j

n Anzahl aller Wiederholungen

Ein Problem, dass die UCB1 Formel auf diesem Weg ebenfalls effizient löst, ist das zuvor erwähnte Exploration-Exploitation-Dilemma. Im Rahmen des Multi-Armed-Bandit-Problems [9] erreicht die UCB1 Formel logarithmischen Regret gleichmäßig über n (Anzahl der Wiederholungen) und ist somit sublinear [19, S. 237]. Dadurch kann im Durchschnitt ein guter Regret-Wert erzielt werden.

Der MCTS ist nicht direkt auf Suchbäume mit imperfekten Informationen anwendbar, da es nicht die Unsicherheit des unvollständigen Wissens über den tatsächlichen Zustand der Knoten einbezieht. Dem gegenüber steht die Weiterentwicklung Information-Set Monte-Carlo-Tree-Search (IS-MCTS) [56]. Diese Baumsuche kann auch effizient in Spielen mit imperfekten Informationen angewandt werden.

2.3.2 *Counterfactual Regret Minimization*

Der ***Counterfactual Regret Minimization (CFR)*** ist ein spieltheoretischer Algorithmus, der auf komplexe dynamische Nullsummenspiele mit zwei Spielern und imperfekten Informationen anwendbar ist. Durch wiederholendes Selbstspiel mehrerer CFR-Algorithmen, wird dabei eine Strategie gefunden, die sich schrittweise einem Gleichgewicht nähert. Der Algorithmus basiert auf dem spieltheoretischen Regret-Matching [57] (Methode zum Berechnen des Regrets in wiederholenden Spielen) und erstellt ein Modell des Gegenspielers und dessen Strategie, welches mit jeder Wiederholung des Spiels angepasst wird. Ein zentraler Bestandteil ist das Einschätzen von hypothetischen Spielverläufen; dem so genannten Counterfactual Regret (deutsch kontrafaktisches

Bedauern). Sie werden verwendet, um die erwartete Auszahlung eines Zuges zu berechnen, selbst wenn dieser nicht gespielt wird (ähnlich wie die heuristische Bewertung). Jeder Spieler aktualisiert nach einem Spieldurchlauf die Strategie, um den Counterfactual Regret zu minimieren [55, 58]. Ein Nachteil des CFR ist, dass es einen kompletten Durchlauf des Spielbaums erfordert und dadurch rechenintensiv ist. Alle gelernten Daten über die Spielverläufe werden tabellarisch gespeichert, was bei simplen Spielen effizient bleibt, doch nicht mit steigender Komplexität und Teilnehmerzahl. Anstatt der limitierten tabellarischen Speicherung wird in der Weiterentwicklung Deep CFR (deutsch tiefes CFR) ein tiefes neuronales Netz eingesetzt (siehe Abschnitt 3.5), um die gelernten Werte zu approximieren [59, 60]. Die Weiterentwicklung CFR+ beschleunigt die Konvergenzgeschwindigkeit von CFR und konnte sogar für die Poker Variante Heads-up Limit Hold'em erfolgreich eingesetzt werden [61, 62]. Die Weiterentwicklung discounted CFR (DCFR) wurde erfolgreich in der Poker-KI Pluribus [39] eingesetzt. Diese KI konnte als erste die Weltmeister in No-Limit Texas Hold'em Poker (die klassische Poker Variante) schlagen. Dazu gibt es noch weitere CFR-Erweiterungen, die jedoch im Kern dem vorgestellten CFR sehr ähnlich sind und nicht tiefer beschrieben werden.

2.4 Zwischenfazit: Ein Spiel untersuchen und lösen

In den vorherigen Abschnitten konnte anhand von simplen Spielen anschaulich erläutert werden, wie es mittels der Spieltheorie möglich ist, ein allgemeines Spiel formal zu beschreiben und zu lösen. Dabei ist es immer das Ziel der Spieltheorie, ein Gleichgewicht an Strategien zu finden, bei denen kein Spieler nachträglich seine Strategie ändern würde, um eine höhere Auszahlung zu erreichen [7, 15; 21]. Die vorgestellten Methoden der Spieltheorie erreichen jedoch Hürden der Berechenbarkeit unter Realzeitanforderungen bereits bei scheinbar kleinen dennoch komplexen dynamischen Spielen, wie Tic-Tac-Toe und Vier-Gewinnt, bei denen der Verzweigungsfaktor und die Tiefe des Spielbaums so hoch sind, dass die Anzahl der möglichen Spielverläufe geradezu explodiert [49, 51]. Dazu wurden heuristische Suchverfahren vorgestellt, die Spielzüge im Spielbaum iterativ mit einer geschätzten Auszahlung bewerten und anhand der Heuristik die besten Spielzüge bevorzugen. Weitere Hürden, die bei komplexen dynamischen Spielen auftreten, sind das Exploration-Exploration-Dilemma [1, S. 367-368], also die Unsicherheit über die tatsächliche Qualität der heuristischen Bewertung, und das Credit-Assignment Problem [1, S. 134], das besonders in Spielen mit imperfekten Informationen die Bewertung der Spielzüge zusätzlich erschwert. Anschließend wurden zwei vielversprechende Algorithmen vorgestellt: der Monte-Carlo-Tree-Search und der Counterfactual Regret Minimization-Algorithmus. Die heuristische Baumsuche des Monte-Carlo-Tree-Search löst viele der vorgestellten Hürden, da durch zufällige Simulation eines Spielverlaufs die Heuristik angepasst wird und iterativ die Knoten mit der besten Heuristik expandiert werden [16, S. 5-9]. Monte-Carlo-Tree-Search ist vielversprechend, da es bereits mehrfach in starken Spiele-KIs in Verbindung mit maschinellem Lernen als heuristische Baumsuche eingesetzt wurde [26, 27]. Das

vorgestellte Counterfactual Regret Minimization ist ein sehr interessanter spieltheoretischer Algorithmus, der durch iteratives Minimieren des Regrets oft zu einem spieltheoretischen Gleichgewicht konvergiert [58]. Durch ihn ließen sich bereits einfache Poker-Varianten lösen. Die Weiterentwicklungen CFR+ und Deep CFR erreichten sogar KIs, die auf hohem Niveau in komplexeren Poker-Varianten spielen können [59–62].

3 Wie kann eine KI durch maschinelles Lernen sinnvolle Entscheidungen treffen?

Dieser Frage widmet sich dieses Kapitel und stellt dazu die notwendigen Grundlagen des maschinellen Lernens vor. Nach einer Einleitung in die wichtigsten Grundbegriffe, werden die drei großen Lernverfahren des maschinellen Lernens vorsgestellt. Jedes der Lernverfahren unterscheidet sich stark darin, wie sie aus Daten lernen und welche Ergebnisse sie erzeugen. Besonders ausführlich wird dabei das reinforcement learning (deutsch: Lernen durch Verstärkung oder bestärkendes Lernen) betrachtet. Dieses Lernverfahren ist besonders relevant für den Fokus dieser Arbeit, da viele erfolgreiche KIs für Spiele existieren, welche das reinforcement learning einsetzen.

3.1 Grundbegriffe des maschinellen Lernens

Das maschinelle Lernen ist Teil des Bereichs der künstlichen Intelligenz [3, S. 3] und befasst sich mit der Frage, wie man Computerprogramme erstellt, die ihre Leistung bei einer bestimmten Aufgabe durch Erfahrung verbessern [42, S. 17]. Anstatt, dass Regeln (zum Beispiel zur Unterscheidung von Objekten) durch Menschen implementiert werden, findet das Lernen durch Algorithmen selbstständig statt. Beim maschinellen Lernen wird ein mathematisches ***Modell*** erlernt, indem Muster bzw. Gesetzmäßigkeiten aus Daten extrahiert werden und daraus bspw. Vorhersagen für neue unbekannte Daten generiert werden [3, S. 8]. Dabei versucht man durch maschinelles Lernen immer die beste mathematische Annäherung an ein Problem zu finden. Die „Aufgabe eines [maschinellen] Lernverfahrens ist es, eine ***Abbildung*** zu lernen". Dies kann bspw. in einer Apfelsortiermaschine die Abbildung zwischen der Größe und Farbe eines Apfels auf die zugehörige Handelsklasse sein [1, S. 205-206]. Wenn der Agent der Apfelsortiermaschine die Abbildung gelernt hat, wird dieser bei gegebener Größe und Farbe eines Apfels die passende Handelsklasse finden. Im Kontext des maschinellen Lernens spricht man dabei oft von ***lernenden Agenten***: automatischen Software-Einheiten, die ihre Ausgaben durch positives oder negatives Feedback anpassen können [1, S. 21]. Ein zentraler Faktor im maschinellen Lernen sind die Daten, aus denen gelernt wird. Um Zusammenhänge in Eingabedaten zu finden, werden ***Merkmale (engl. features)*** in den Daten definiert, die für das Ergebnis relevant sind und vom Modell nach deren gelernter Relevanz unterschiedlich gewichtet werden [1, S. 205]. Im Beispiel der Apfelsortiermaschine sind die Merkmale die Größe und Farbe des Apfels, weitere könnten aber bspw. das Erntedatum oder Gewicht sein, die eventuell mehr oder weniger relevant für das Ergebnis sind. Die Wahl der ausschlaggebenden Datenmerkmale stellen je nach Aufgabenstellung und Komplexität der Daten ein Problem dar, welchem sich der Teilbereich der Merkmalsextraktion (engl. feature engineering) widmet. Die Anzahl der Merkmale eines Datensatzes werden als Dimensionen angegeben. Hat ein Datensatz n Merkmale, so spricht man von n-dimensionalen Daten [1, S. 204]. Eingabedaten mit

einer hohen Dimensionalität bringen jedoch eine Hürde mit sich: den sogenannten „Fluch der Dimensionen“ [63, S. 314-315], welcher bedeutet, „dass die Trainingszeiten sehr schnell, oft exponentiell, mit der Dimension der Eingabedaten wachsen.“ [1, S. 373] Aus diesem Grund ist es sinnvoll, hochdimensionale Eingabedaten bspw. durch Hauptkomponentenanalyse oder t-SNE (siehe Abschnitt 3.3) auf geringere Dimensionen zu reduzieren (***Dimensionsreduktion***), um den Rechenaufwand während des Lernprozesses einzusparen, doch ohne, dass relevante Zusammenhänge in den Daten verloren gehen.

Der Lernprozess verwendet ***Trainingsdaten***, die aus einer repräsentativen Stichprobe der Aufgabe bestehen. Anschließend wird die Qualität des Modells mittels ***Testdaten*** geprüft und die Genauigkeit bspw. anhand einer Fehlerrate quantifiziert [1, S. 205-206]. Hat das Modell die Abbildung der Daten nicht gut gelernt, spricht man von einer ***Unteranpassung***. Bei einer Unteranpassung muss der Lernprozess wiederholt oder zu einer anderen Lernmethode oder sogar Lernverfahren gewechselt werden. Dieser Vorgang wiederholt sich, bis die Fehlerrate des Modells gering genug ist und konstant bleibt. Im Gegensatz kann es auch zur ***Überanpassung*** an die Daten kommen, einem zentralen Problem im maschinellen Lernen: Das Modell hat die Abbildung der Trainingsdaten und Testdaten so gut gelernt, dass es neue nie zuvor gesehene Daten nicht gut generalisieren kann oder bei Vorhersagen eine hohe Varianz hat [3, S. 94]. Die Fähigkeit des trainierten Models zu generalisieren, hängt stark von der Qualität und Menge der Daten ab, die beim Trainingsprozess zur Verfügung stehen, ganz unabhängig von der Wahl der Lernmethode. Dabei ist es nicht immer einfach die Trainingsdaten mit neuen Daten zu erweitern. Hierzu kann die Datenvermehrung sinnvoll sein. Dazu wird der Datensatz kopiert und so angepasst, dass die Kopie nicht mehr identisch ist. Dies ist möglich in dem bspw. Rauschen hinzugefügt wird oder eine leichte Abweichung. So kann auf einfache Weise die Menge der Trainingsdaten vergrößert werden. Die Anpassung der Daten auf diese Weise ist außerdem sinnvoll, um eine Überanpassung zu vermeiden, da mehr Variation in den Daten vorhanden ist [1, S. 249-250].

Das Forschungsfeld des maschinellen Lernens bietet eine ständig wachsende Zahl an Lösungsmöglichkeiten mit jeweils sehr verschiedenen Herangehensweisen, doch laut des "No-Free-Lunch-Theorem" [43] kann es nie die eine Lösungsmöglichkeit für jedes Problem geben. Ein Problem in der Informatik, welches besagt, dass es kein Verfahren gibt, welches in jedem Aspekt perfekt ist (bspw. Rechenzeit, Fehlerrate, etc.). Daher hängt es komplett von der Natur der Problemstellung und den Anforderungen an das Modell ab, welche Verfahren und Methoden optimal sind. Es ist daher sinnvoll, verschiedene Verfahren und Methoden in Betracht zu ziehen und gegebenenfalls zu kombinieren (auch Ensemble-Lernen genannt).

Im maschinellen Lernen wird zwischen diesen drei ***Lernverfahren***[6] unterschieden: ***supervised learning*** (deutsch: überwachtes Lernen), ***unsupervised learning*** (deutsch: unüberwachtes Lernen) und ***reinforcement learning*** (deutsch: Lernen durch Verstärkung oder bestärkendes Lernen). Die Lernverfahren unterscheiden sich grundlegend darin, wie sie eine Abbildung aus Daten lernen, also eine sinnvolle Beziehung in den Eingabedaten finden. Je nach den Gegebenheiten der Aufgabe, der Art und Menge der vorhanden Trainingsdaten bietet sich ein anderes maschinelles Lernverfahren an. Die Lernverfahren werden folgend im jeweils sinnvollen Umfang erläutert. Hinweis: Jedes dieser Lernverfahren kann mit künstlichen neuronalen Netzen in Kombination angewandt werden, doch da künstliche neuronale Netze ein umfangreiches und für diese Arbeit relevantes Themengebiet ist, wird es separat im Abschnitt 3.5 behandelt.

3.2 Supervised Learning

Im ***supervised learing***[7] ***(SL)*** (deutsch: überwachtes Lernen) wird das Modell mit Daten trainiert, zu denen die richtigen Antworten bekannt sind. Die Trainingsdaten enthalten im SL für jeden Datenpunkt die jeweilge richtige Antwort; man spricht auch von ***beschrifteten Daten*** [64]. Doch darin liegt oft die Herausforderung, denn beschriftete Daten mit garantiert richtigen Antworten sind je nach Anwendungsfall nur unter großem Aufwand erstellbar. Ein Beispiel ist der MNIST-Fashion-Datensatz [65], ein bekanntes Traingsumfeld für Bilderkennungsmodelle. Er besteht aus 70.000 Bildern von Kleidungsstücken, die (wahrscheinlich) alle von Menschen fotografiert und mit den richtigen Antworten beschriftet wurden - ein sehr aufwendiger Prozess. Für die Klassifzierungsaufgabe, die mit diesem Datensatz gelöst werden soll, reicht diese Menge zwar aus, doch je nach Komplexität der Aufgabe kann selbst diese Datenmenge noch zu gering sein. Ein Lösungversuch für dieses Problem kann die zuvor erwähnte Datenvermehrung bieten.

Bei SL-Modellen wird unterschieden, ob das Ziel eine ***Klassifizierung***, also Einteilung in vordefinierte diskrete Klassen (bspw. Handelsklasse des Apfels) oder eine ***Regression*** ist, eine Vorhersage von stetigen numerischen Werten (bspw. Gewicht des Apfels bei gegebener Größe), denn Klassifizierungs- und Regressionsmodelle verwenden oft unterschiedliche Lernmethoden. Eine ***Lernmethode***[8] ist ein Algorithmus, der anhand der Trainingsdaten das mathematische Modell erstellt. Die Lernmethoden des SL unterteilen sich in ***eifriges Lernen***, das im Lernprozess anhand der Trainingsdaten das Modell erstellt, und ***faules Lernen***, bei dem kein mathematisches Modell

[6] In der Literatur werden die Begriffe Lernverfahren und Lernmethode nicht immer einheitlich und oft sogar als Synonyme verwendet, doch für den Rahmen dieser Arbeit gelten die formulierten Bedeutungen inklusive deren inhaltliche Abgrenzung zueinander und werden konsistent verwendet.

[7] Hier wird der englische statt dem deutschen Begriff verwendet, da es zum Eigennamen für dieses Lernverfahren geworden ist.

[8] Der Begriff Lernmethode wird im gesamten maschinellen Lernen verwendet und ist nicht auf SL beschränkt.

erstellt wird, sondern die Trainingsdaten gespeichert und für Vorhersagen oder Klassifizierungen herangezogen werden, wenn neue Datenpunkte eintreffen [1, S. 225].

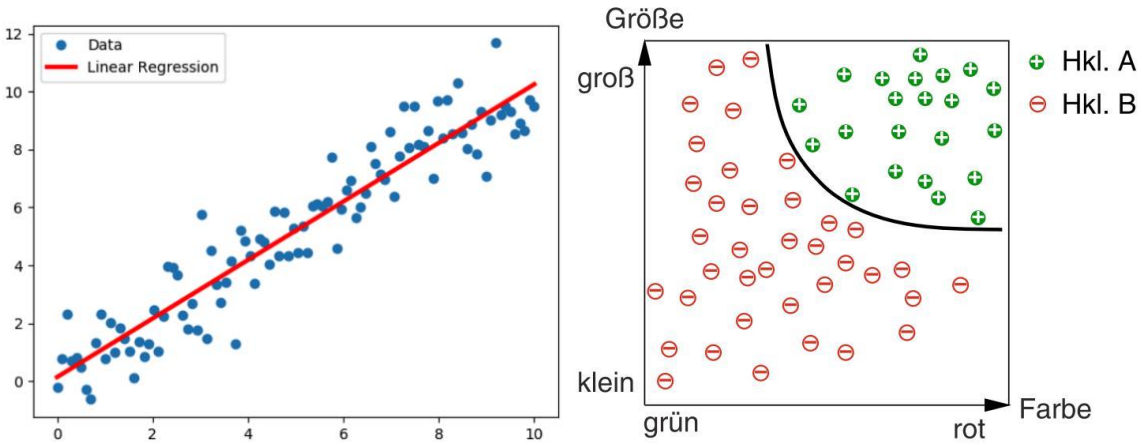


Abbildung 3.1 (links) Beispiel für ein lineares Regressionsmodell: zweidimensionale Datenpunkte werden mittels einer linearen Funktion approximiert [66, S. 4].

Abbildung 3.2 (rechts) Beispiel für ein binäres Klassifizierungsmodell: Äpfel werden anhand der Merkmale Farbe und Größe in die zwei Handelsklassen eingeteilt [1, S. 204].

Je nach Dimensionalität der Trainingsdaten muss im SL auf eine andere Lernmethode zurückgegriffen werden. Für binäre Klassifizierung bietet sich die Lernmethode der logistischen Regression[9] an, die mittels Log-Odd- und Sigmoid-Funktion die Wahrscheinlichkeit für die Zugehörigkeit eines Datenpunkts zur Klasse errechnet [1, S. 313-314]. Für eine mehrdimensionale Klassifizierung sind der C4.5 oder Random Forest-Algorithmus sinnvoll, die einen Entscheidungsbaum anhand der Merkmale erstellen, wobei Blattknoten für eine Klassifizierung stehen und Äste für Wahrscheinlichkeiten der Knoten. Durch Ablaufen des Entscheidungsbaums lässt sich eine Klassifizierung für jeden Datenpunkt ermitteln [1, S. 241-242]. Für Regressionsaufgaben mit linearem Zusammenhang der Eingabedaten lässt sich die Lernmethode der linearen Regression anwenden (siehe Abbildung 3.1), die schrittweise die Fehlerrate der Vorhersage minimiert, durch Anpassen des linearen Modells [63, S. 747-750]. Bei Regression in mehrdimensionalen Daten bietet sich bspw. die Lernmethode der Support Vector Regression an, die mittels Kernel-Funktion die Eingabedaten in einen höherdimensionalen Raum transformiert, in dem eine lineare Trennung der Daten möglich ist [67, S. 3812].

Da das SL in Abschnitt 4.1für die Anwendung als nicht umsetzbar eingestuft wurde, wird auf eine weitere Vertiefung des SL verzichtet.

[9] Logistische Regression ist ein Klassifzierungsverfahren, auch wenn es „Regression“ im Namen hat.

3.3 Unsupervised Learning

Im ***unsupervised learning***[10] ***(UL)*** (deutsch: unüberwachtes Lernen) wird das Modell mit Daten trainiert, zu denen die richtigen Antworten nicht bekannt sind. Die Trainingsdaten enthalten im Gegensatz zu SL keine Beschriftung. Aufgabe eines UL-Modells ist es, sinnvolle Zusammenhänge, Muster oder Strukturen in den Traingsdaten zu finden, ohne dass dabei Hinweise vorgegeben sind. UL wird bspw. für die Clusteranalyse in ungeordneten Daten oder für Dimenisonreduktion in hochdimensionalen Daten eingesetzt.

Bei der ***Clusteranalyse*** (auch Clustering genannt) teilt das Modell die Eingabedaten in Gruppen (***Cluster***) ein, anhand ähnlicher Datenpunkte, ähnlich wie die Klassifizierung beim SL, doch ohne, dass Klassen vordefiniert sind [64, 68, S. 255]. Dies ist bspw. sinnvoll bei großen Mengen an Bilddaten, die sich durch Ähnlichkeiten in Gruppen einteilen lassen, doch bei denen es durch die Menge der Daten zu aufwändig wäre, sie durch Menschen manuell zu beschriften. In diesem Fall könnte UL mit einer Clusteranalyse als Vorverarbeitung eingesetzt werden, um die Bilddaten anhand ihrer Ähnlichkeit in Cluster vorzusortieren, diese Cluster anschließend mit Klassen zu versehen und dann ein SL Klassifizierungsmodell damit zu trainieren [68, S. 259]. Eine Lernmethode für die Clusteranalyse ist bspw. das ***k-Means-Verfahren*** (deutsch: k-Durchschnitts-Verfahren), welches die Eingabgedaten in die zuvor festgelegete Anzahl von *k* Clustern einteilt (siehe Abbildung 3.3). Das k-Means-Verfahren setzt für jedes Cluster einen zufälligen Datenmittelpunkt fest und verschiebt diesen iterativ mittels durchschnittlichen Abstand vom Mittelpunkt zu den nächstgelegenen Daten bis es zu einer stabilen Lösung konvergiert [68, S. 272-274]. Wenn die Anzahl der Cluster in den Daten im Voraus nicht bekannt ist, kann der EM-Algorithmus, eine stetige Variante des k-Means, verwendet werden, der für jeden Datenpunkt die Wahrscheinlichkeiten der Klassenzugehörigkeit angibt [1, S. 262-263].

UL lässt sich auch für die Dimensionsreduktion in hochdimensionalen Eingabedaten einsetzen. Bspw. ist das t-distributed stochastic neighbor embedding (t-SNE) eine Lernmethode, die die Wahrscheinlichkeit von zusammengehörigen Datenpunkten über deren Nähe im Datenraum kalkuliert und visualisiert, ohne dass dabei nicht-lineare Zusammenhänge der Datenpunkte verloren gehen (siehe Abbildung 3.4). Das Ergebnis des t-SNE reduziert die Daten auf eine niedrige Dimension und ordnet die Datenpunkte anhand ihrer Ähnlichkeit zueinander im Streudiagram an. Dabei muss beachtet werden, dass es zu Verzerrungen zwischen den Abständen kommen kann und t-SNE eher als eine grobe explorative Untersuchung der Daten eingesetzt werden sollte, anstatt komplett auf das Ergebnis zu vertrauen [68, S. 269-270].

[10] Hier wird der englische statt dem deutschen Begriff verwendet, da es zum Eigennamen für dieses Lernverfahren geworden ist.

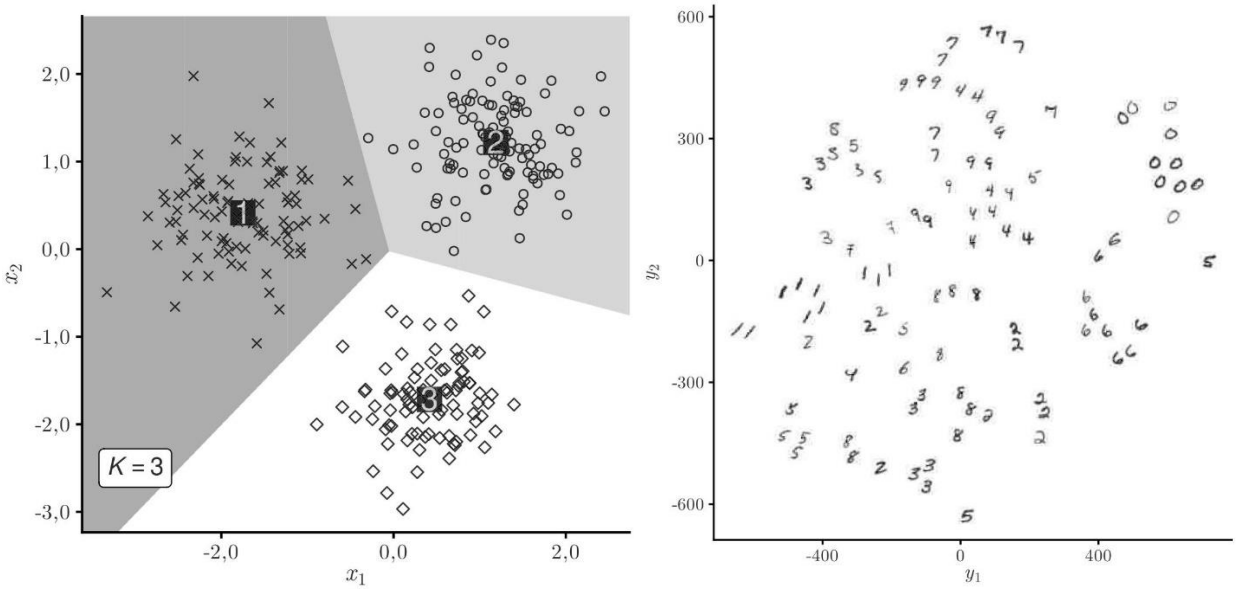


Abbildung 3.3 (links) Ergebnis des k-Means-Verfahrens mit drei (k) Clustern (graue Flächen) in einer zweidimensionalen Datenmenge. [68, S. 278]

Abbildung 3.4 (rechts) t-SNE ordnet handgeschriebene Ziffern der MNIST-Datenbank [69] in einem Streudiagramm anhand der Ähnlichkeit und Nähe der Datenpunkte (in diesem Fall Pixel) an, ohne dabei Wissen über die geschriebene Ziffer selbst zu haben. So wurde mittels t-SNE in diesem Beispiel die Dimension der Daten von ≈ 784 Merkmalen (28×28 Pixel pro Bild) auf eine zweidimensionale Koordinate reduziert [68, S. 271].

Da das UL in Abschnitt 4.1 für die Anwendung als nicht umsetzbar eingestuft wurde, wird für eine weitere Vertiefung des SL verzichtet.

3.4 Reinforcement Learning

Modelle des reinforcement learnings[11] (RL) (deutsch: Lernen durch Verstärkung oder bestärkendes Lernen) treffen Vorhersagen, indem sie Belohnungen oder Bestrafungen auf der Grundlage von Aktionen erhalten, die sie selbstständig in einer Umgebung durchgeführt haben. Ein System mit RL generiert eine Strategie, die den Erhalt der Belohnungen maximiert und diese Strategie iterativ verbessert [64]. Es ist ein zielgerichtetes Lernen, dass sich zu anderen Lernverfahren darin unterscheidet, dass es keine Eingabedaten gibt, wie im Sinne von SL und UL, sondern der lernende Agent mit einer Umgebung direkt und selbständig interagiert, ohne dass dabei eine beispielhafte Überwachung oder ein vollständiges Modell der Umgebung erforderlich ist [2, S. 13]. RL lässt sich so in Situationen einsetzen, in denen keine Trainingsdaten vorhanden sind oder es die Komplexität der Aufgabe nicht zulässt diese zu kodieren, wie bspw. in der Robotik [1, S. 360-362, 1, S. 351]. Darin liegt die universelle Stärke dieses Lernverfahrens. RL wurde erfolgreich auf

[11] Hier wird der englische statt dem deutschen Begriff verwendet, da es zum Eigennamen für dieses Lernverfahren geworden ist.

das äußerst komplexe chinesische Brettspiel Go (AlphaGo [26], 2016, unter anderem RL) und Texas Hold'em (DeepStack, 2017, tiefe neuronale Netze und RL) [29], eine Variante des Kartenspiels Poker, angewendet. Beide Modelle haben menschliche Expertenspieler in Wettbewerben geschlagen. Neben komplexen Gesellschaftsspielen wurde RL bspw. erfolgreich in Single-Player-Videospielen (sieben der Atari 2600 Spiele in 2013 [25]), als auch in komplexen Multi-Player-Echtzeitstrategiespielen (Dota 2 in 2019 [28]) eingesetzt. Das RL-Modell konnte jeweils menschliche Expertenspieler überbieten. Aufgrund dieses äußerst großen Potenzials in der Anwendung von komplexen dynamischen Kartenspielen mit imperfekten Informationen wird das RL im Folgenden etwas genauer betrachtet.

3.4.1 Umgebung formalisieren mittels Markov-Entscheidungsprozess

Im RL lernt der Agent durch ***Belohnung*** aus ***Aktionen*** in einer ***Umgebung***. Der RL-Agent und die Umgebung interagieren ständig, wobei der Agent ***Aktionen*** A auswählt, die Umgebung auf diese Aktionen reagiert und dem Agenten neue ***Zustände*** S präsentiert. Ein Zustand enthält beliebige Informationen, die dem Agenten zugänglich sind und seine Entscheidungen beeinflussen. Aus der Umgebung ergeben sich ***Belohnungen*** R (numerische Werte), die der Agent durch die Wahl seiner Aktionen maximieren möchte. Jeder Zustand und dessen Belohnung sind mit einer gewissen ***Übergangswahrscheinlichkeit*** tatsächlich erreichbar. Es ist also möglich mit der gleichen Aktion verschiedene Belohnungen und Folgezustände zu erreichen. Um eine Umgebung formal zu beschreiben, wird der ***Markov-Entscheidungsprozess (MDP)*** (engl.: Markov-Decision-Process) angewandt. MDPs sind eine klassische Formalisierung der sequenziellen Entscheidungsfindung, bei der Aktionen nicht nur direkte Belohnungen, sondern auch nachfolgende Zustände und damit zukünftige Belohnungen beeinflussen [2, S. 47-48]. MDPs sollen das Problem des zielgerichteten Lernens aus Interaktionen auf einfache Weise darstellen (siehe Abbildung 3.5). Die Anwendung eines MDPs kann sehr abstrakt sein und dadurch auf viele verschiedene Probleme angewandt werden, gleich ob digital oder analog. Eine Aktion kann jeder Art von Entscheidung entsprechen, ein Zustand einer beliebigen Information und die Belohnung eine Angabe darüber, wie sehr die Aktion den Agenten dem Ziel näherbringt [2, S. 47-50]. In einem MPD wird davon ausgegangen, dass ein Zustand Informationen aller bisherigen Interaktionen zwischen Agenten und Umwelt enthält, die für zukünftige Entscheidungen von Bedeutung sind. Wenn dies zutrifft, gilt die sogenannte Markov-Eigenschaft [2, S. 49]. Wenn in einem MDP der tatsächliche Zustand eines Agenten nicht exakt bekannt ist und die Zustände nur komplett beobachtbar sind, spricht man von einer Erweiterung des MDP, dem ***teilweise beobachtbaren Markov-Entscheidungsprozess (POMDP)*** (engl.: partially observable Markov-Decision-Prozess) [1, S. 355, 2, S. 467].

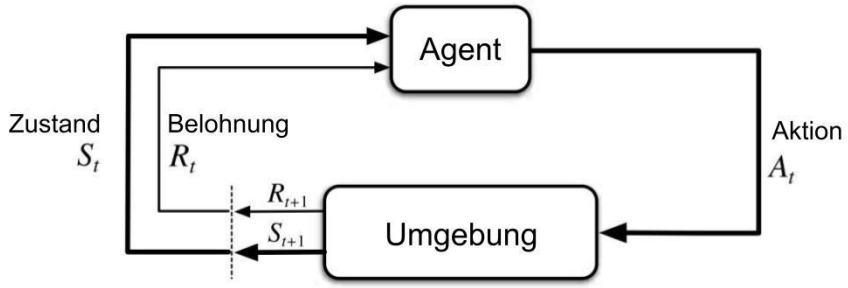


Abbildung 3.5 Die Interaktion zwischen Agenten und Umgebung im MDP [2, S. 48]
$S_t \in S$ ist der Zustand zum Zeitpunkt t aus der Zustandsmenge
$A_t \in A(s)$ ist die Aktion Zeitpunkt t aus der Aktionsmenge, die im Zustand s verfügbar ist
$R_t \in R$ ist die Belohnung zum Zeitpunkt t aus der Belohnungsmenge
$t + 1$ nächster Zeitpunkt

Um im MDP zwischen direkter und zukünftiger Belohnung zu unterscheiden, wird ein diskreter Zeitpunkt t eingesetzt, in dem sich Agent und Umgebung befinden. MDPs beinhalten also eine direkte Belohnung zum Zeitpunkt t und eine verzögerte Belohnung zum Zeitpunkt $t + 1$, wobei zwischen unmittelbarer und verzögerter Belohnung abgewogen wird [2, S. 47-50]. Um verzögerte Belohnungen variabel zu gewichten, wird der ***Diskontierungsfaktor***[12] γ (Wert zwischen 0 und 1; $0 \geq \gamma \leq 1$) verwendet und die Belohnung der Folgezustände um den Faktor γ abgeschwächt [2, S. 55]. Die kumulative diskontierte Belohnung zum beliebigen Zeitpunkt t lässt sich mit dem Rückgabewert G_t errechnen (siehe Formel 3.1).

$$G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

Formel 3.1 Der Rückgabewert G zum Zeitpunkt t ergibt sich aus der Summe aller Folgebelohnungen R_{t+k+1}, die mit dem potenzierten Diskontierungsfaktor γ^k multipliziert werden, um die Gewichtung von direkten und verzögerten Belohnungen zu gewährleisten. [2, S. 55]

Je nach Anwendungsfall lässt sich die Umgebung in Wiederholungen von Aktionen bis zu einem finalen Zeitpunkt T aufteilen (bspw. Spielpartien oder Labyrinth-Durchläufe), die sich dann in ***Episoden*** wiederholen lassen [2, S. 54]. In diesem Fall ist die Umgebung durch einen ***endlichen MDP*** (auch diskreter MDP oder episodisches RL genannt) umsetzbar, in dem die Menge der Zustände S, Aktionen A und Belohnungen R begrenzt sind [2, S. 48]. Wenn es nicht möglich ist die Umgebung in Episoden zu unterteilen, handelt es sich um einen ***stetigen MDP*** [2, S. 54]. Die Unterschiede zwischen endlichen und stetigen MDPs haben Auswirkungen auf die Art der Modellierung und Algorithmen. Bei endlichen MDPs können oft ***tabellarische Methoden*** verwendet werden, bei denen Bewertungen von Zuständen und Aktionen bspw. in Tabellen oder Arrays

[12] Der Diskontierungsfaktor ist symbolisiert durch den griechischen Buchstaben γ (Gamma). Ein Wert nahe null gewichtet direkte Belohnung stärker und ein Wert nahe 1 gewichtet verzögerte Belohnung stärker.

gespeichert und aktualisiert werden. Bei einem stetigen MDP sind Methoden zur Funktionenapproximation notwendig (bspw. neuronale Netze (siehe Abschnitt 3.5)) um den kontinuierlichen Zustands- und Aktionsraum abzubilden. In einem stetigen MDP wären tabellarische Methoden mit unendlichem Speicheraufwand verbunden [2, S. 67].

Im Ziel des MPDs ist es, eine ***Strategie***[13] π zu finden, welche die kumulative erwartete Belohnung maximieren kann und einer ***optimalen Strategie*** π_* möglichst nahe kommt oder diese erreicht. Eine Strategie ist eine Abbildung von Zuständen auf Aktionen und gibt an, welche Aktion in welchem Zustand gewählt werden soll [1, S. 354-355]. Dies kann eine deterministische Strategie sein $\pi(s)$, bei der jedem Zustand genau eine Aktion zugeordnet ist. Oder es kann eine stochastische Strategie $\pi(a|s)$ sein, wobei die Strategie jeder Aktion eine Wahrscheinlichkeit zuordnet, mit der diese Aktion gewählt werden soll. Eine Aktion erhält eine hohe Wahrscheinlichkeit, wenn ihre kumulative erwartete Belohnung hoch ist [2, 76–79]. Um eine optimale Strategie π_* zu ermitteln, wird im RL ***dynamische Programmierung*** angewandt. Dynamische Programmierung wird zum algorithmischen Lösen von Optimierungsproblemen eingesetzt, indem das Problem in Teilprobleme zerlegt und Zwischenergebnisse verwendet werden, um die optimale Lösung zu finden. Das Bellman-Prinzip[14] besagt: „Unabhängig vom Startzustand s_t und der ersten Aktion a_t müssen ausgehend von jedem möglichen Nachfolgezustand s_{t+1} alle folgenden Entscheidungen optimal sein.“ Basierend auf diesem Prinzip lässt sich im MDP eine globale optimale Strategie π_* finden, indem lokale Aktionen optimiert werden [1, S. 358, 6].

Um Zustände und Zustand-Aktions-Paare anhand ihrer erwarteten Belohnung zu bewerten, kommen ***Wertfunktionen*** zum Einsatz. Wertfunktionen basieren auf der ***Bellman-Gleichung***, die den rekursiven Zusammenhang zwischen dem Wert eines Zustands und dem kumulativen gewichteten Wert seiner Folgezustände und Übergangswahrscheinlichkeiten darstellt. Die Bewertung eines Zustands s unter Berücksichtigung der Strategie π lässt sich mittels ***V-Funktion*** $v_\pi(s)$ (auch Zustandswertfunktion genannt) errechnen (siehe Formel 3.2), wobei der errechnete Wert die kumulative Belohnung für alle Folgezustände s' ist, vom Zustand s ausgehend unter Beachtung der Strategie π. Die V-Funktion gibt also an, wie gut es ist, sich in einem bestimmten Zustand zu befinden, basierend auf den erwarteten zukünftigen Belohnungen und den Übergängen zwischen den Zuständen [2, S. 58-59].

---

[13] Eine Strategie im RL hat Ähnlichkeit zur Strategie der Spieltheorie (siehe Abschnitt 2.1), doch die Begriffe stehen in keinem direkten Zusammenhang. Die Strategie ist symbolisiert durch den griechischen Buchstaben π (Pi), doch es steht in keinem Zusammenhang zur Kreiszahl π.

[14] Benannt nach dem amerikanischen Mathematiker Richard Bellman, einem Pionier der dynamischen Programmierung [6, 70].

$$v_\pi(s) \doteq E_\pi[\, G_t \mid S_t = s \,] = E_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s\right] \textit{für alle } s \in S$$

Formel 3.2 Die V-Funktion im Zustand s ist der Erwartungswert unter der Strategie π von der diskontierten Rückgabe G_t vom Zustand s ausgehend [2, S. 58].

Der Wert einer Aktion a im Zustand s (Zustand-Aktions-Paar) lässt sich mittels ***Q-Funktion*** $\mathrm{q}_\pi(\mathrm{s},\mathrm{a})$ (auch Aktionswertfunktion genannt) errechnen (siehe Formel 3.3). Der errechnete Wert ist die kumulative Belohnung für alle Folgezustände s' und Folgeaktionen a' ausgehend vom Zustand s mit der Aktion a, wenn anschließend die Strategie π beachtet wird [1, S. 355, 2, S. 58].

$$q_\pi(s,a) \doteq E_\pi[\, G_t \mid S_t = s, A_t = a \,] = E_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s, A_t = a\right] \text{ für alle } s \in S \text{ und } a \in A(s)$$

Formel 3.3 Die Q-Funktion für Aktion a im Zustand s ist der Erwartungswert unter der Strategie π von der diskontierten Rückgabe G_t ,wenn vom Zustand s ausgehend Aktion a gewählt wird [2, S. 58].

Eine Strategie π ist besser als eine andere Strategie π', wenn ihre V-Funktion $\mathrm{v}_\pi(\mathrm{s})$ besser ist als eine andere V-Funktion $\mathrm{v}_{\pi'}(\mathrm{s})$ für alle Zustände der Zustandsmenge[15]. Wertfunktionen sind optimal ($v_*(s)$[16] und $q_*(s,a)$[17]), wenn sie für jeden Zustand bzw. jedes Aktions-Zustands-Paar die maximal erreichbare kumulative Belohnung zuweisen, die unter jeder Strategie möglich ist und so die ***Bellman-Optimalitätsgleichung*** lösen (siehe Formel 3.4) [2, S. 62-63, 6]. Eine Strategie, die optimale Wertfunktionen besitzt, ist immer auch eine ***optimale Strategie*** π_*. Das bedeutet, dass es reicht, optimale Wertfunktionen zu finden, um eine optimale Strategie π_* zu erreichen (ein zentraler Fakt zur Lösung des MDPs). In jedem MDP sind optimale Wertfunktionen und ebenso mindestens eine optimale Strategie möglich. Es lässt sich also durch Lösen der Bellman-Optimalitätsgleichung immer eine optimale Strategie π_* für einen MDP bestimmen [2, S. 68]. Die Wertfunktionen lassen sich beispielhaft durch ein Backup-Diagramm visualisieren (siehe Abbildung 3.6 und Abbildung 3.7), in dem weiße Kreise Zustände und schwarze Kreise Aktionen symbolisieren und Pfeile die Aktualisierungsrichtung darstellen.

[15] Formal ausgedrückt ist Strategie π besser $\pi \geq \pi'$ wenn $\mathrm{v}_\pi(\mathrm{s}) \geq \mathrm{v}_{\pi'}(\mathrm{s})$ für alle $\mathrm{s} \in \mathrm{S}$ ist [2, S. 62].
[16] Für eine optimale V-Funktion gilt $v_*(s) \doteq \max_\pi v_\pi(s)$ für alle $\mathrm{s} \in \mathrm{S}$ [2, S. 62-63].
[17] Für eine optimale Q-Funktion gilt $q_*(s,a) \doteq \max_\pi q_\pi(s,a)$ für alle $\mathrm{s} \in \mathrm{S}$ und $\mathrm{a} \in \mathrm{A(s)}$ [2, S. 62-63].

$$v_*(s) = \max_{a \in \mathcal{A}(s)} q_{\pi_*}(s,a)$$

$$v_*(s) = \max_a \sum_{s',r} p(s',\, r \mid s,\, a)[r + \gamma\, v_*(s')]$$

$$q_*(s,a) = \sum_{s',r} p(s', r \mid s, a)\left[r + \gamma \max_{a'} q_*(s', a')\right]$$

Formel 3.4 (obere) Die Bellman-Optimalitätsgleichung wird angewandt, um die optimale V-Funktion und die optimale Q-Funktion abzuleiten und so eine optimale Strategie π_* zu bestimmen [2, S. 63].

Formel 3.5 (mittlere) Die optimale Bewertung eines Zustands Zustand *s* mittels V-Funktion, wobei rekursiv nach der Aktion mit der höchsten erwarteten Belohnung maximiert wird [2, S. 63].

Formel 3.6 (untere) Die optimale Bewertung für eine Aktion *a* im Zustand *s* (Zustand-Aktions-Paar) mittels Q-Funktion, wobei rekursiv nach der Folgeaktion mit der höchsten erwarteten Belohnung maximiert wird [2, S. 64].

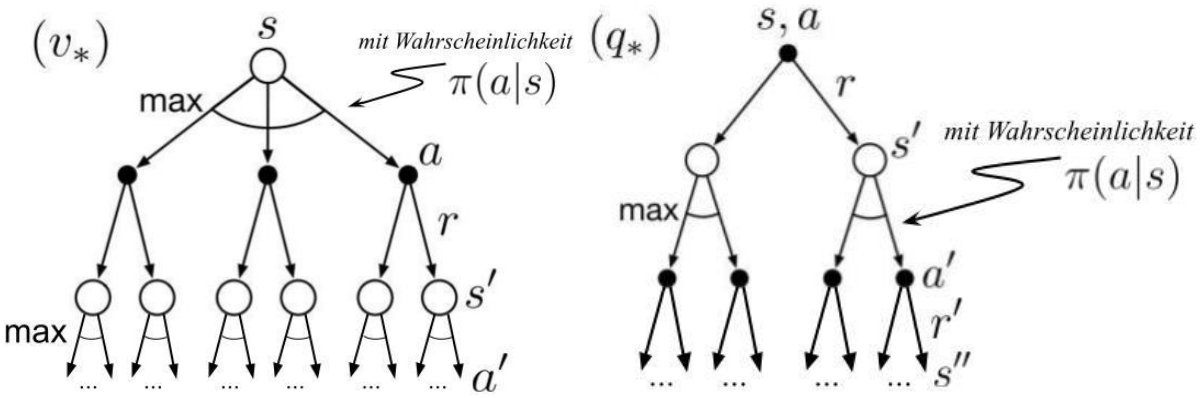


Abbildung 3.6 (links) Backupdiagramm für die optimale V-Funktion, welche ausgehend vom Zustand *s* die erwartete kumulative Belohnung rekursiv maximiert [2, S. 64]

Abbildung 3.7 (rechts) Backupdiagramm für die optimale Q-Funktion, welche ausgehend vom Zustand-Aktions-Paar s, a die erwartete kumulative Belohnung rekursiv maximiert [2, S. 64]

3.4.2 Hürden eines komplexen MDPs in realistischen Anwendungen

Eine Umgebung lässt sich also mittels des beschriebenen MPD formalisieren und durch Lösen der Bellman-Optimalitätsgleichung eine optimale Strategie des MPDs finden, doch dazu müssen Zustandsmenge *S*, Aktionsmenge *A*, Belohnungsmenge *R* und die Funktion der Übergangswahrscheinlichkeiten $p(s' \mid s, a)$ bekannt sein, also in anderen Worten muss ein komplettes Modell der Umgebung[18] vorhanden sein (***vollständiges Wissen***)[19]. Selbst wenn ein Agent vollständiges Wissen der Umgebung hätte, wäre es nur in kleinen Zustands- und Aktionsräumen komplett anwendbar, da Speicherbedarf und Rechenzeit exponentiell mit der Größe der Umgebung

[18] In diesem Kontext ist mit dem Begriff der Umgebung immer die mittels MPD formalisierte Umgebung gemeint.

[19] Vollständiges Wissen in einem MDP ist nicht dasselbe wie vollständige Information. Vollständige Information ist ein spieltheoretischer Begriff.

wachsen, um alle Zustände zu untersuchen [1, S. 356-357]. Doch in realistischen Anwendungen sind Speicher und Rechenzeit begrenzt und stehen eben nicht quasi unendlich zur Verfügung [2, S. 69]. Daher braucht es eine effektivere iterative Methode, welche eine Lösung der Bellman-Optimalitätsgleichung approximiert und dazu bestenfalls nur einen Teil der Umgebung tatsächlich untersuchen muss. Ein weiteres Problem besteht darin, dass sich nur in wenigen Anwendungsfällen ein vollständiges Wissen über die Umgebung voraussetzen lässt (***unvollständiges Wissen***), also muss der Agent erst die Belohnungen und Übergangswahrscheinlichkeiten des MDPs erlernen [1, S. 362-363, 2, S. 67].

3.4.3 Methoden zum Lösen eines komplexen MDPs ohne vollständiges Wissen

Die Lernmethoden des RL unterscheiden sich grundlegend darin, wie sie eine algorithmische Lösung des MDPs erzielen und werden daher in ***wertbasierte***, ***strategiebasierte Methoden*** und diese wiederum in ***modellfreie und modellbasierte Methoden*** unterteilt. Je nach Problemstellung und Aufbau der Umgebung bietet sich eine andere Lernmethode an [2, S. 321, 2, S. 27, 2, S. 12].

Wertbasierte Lernmethoden sind modellfreie Methoden, die auf der zuvor erläuterten Tatsache basieren, dass eine optimale Strategie π_* sich aus optimalen Wertfunktionen (V- oder Q-Funktion) ableiten lässt, und versuchen daher diese direkt zu ermitteln, ohne dabei die Strategie π selbst zu optimieren. In anderen Worten wird bei einer wertbasierten Lernmethode allein die Bewertung der Zustände bzw. Zustand-Aktions-Paare gelernt und iterativ optimiert, bis es zu einer optimalen Strategie π_* kommt, unter der immer die Aktion mit der höchsten Bewertung gewählt wird. Eine Hürde der wertbasierten Lernmethoden ist das ***Exploration-Exploitation-Dilemma*** (siehe Abschnitt 2.3), bei dem unklar ist, ob es sich um ein globales oder lokales Optimum der approximierten Wertfunktion handelt, also ob eine bessere Wertfunktion $v'(s)$ existiert, weil bspw. ein späterer Folgezustand s' mit insgesamt besserer Bewertung nicht berücksichtigt oder zu stark durch den Diskontierungsfaktor γ abgeschwächt wurde. Um das Exploration-Exploitation-Dilemma in wertbasierten Lernmethoden zu umgehen, werden Methoden wie bspw. ε-greedy als Strategie verwendet. Dabei wird die Umgebung mit einer zuvor definierten Wahrscheinlichkeit von ε exploriert, anstatt die bisherige Bewertung zu verwenden. Mit jeder Wiederholung wird die ε-Wahrscheinlichkeit verringert [1, S. 367-368, 2, S. 100].

TD-Learning[20] (Temporal Difference Learning) (deutsch: Lernen mit zeitlicher Differenz) ist eine Gruppe von wertbasierten modellfreien Lernmethoden des RL. Sie vereint zufällige Schätzungen, eine Eigenschaft der Monte-Carlo-Methoden, und die Wertiteration, ein Konzept der dynamischen

[20] Hier wird der englische statt dem deutschen Begriff verwendet, da es zum Eigennamen für diese Lernmethode geworden ist.

Programmierung, miteinander [2, S. 119]. Im TD-Learning wird eine Wertfunktion geschätzt, die sich iterativ der optimalen Wertfunktion nähert. Dazu wird die Differenz von der aktuellen Schätzung des Zustands bzw. Zustand-Aktions-Paares mit der erwarteten zukünftigen Belohnung genutzt (TD-Fehler δ), um die letzte Bewertung zu korrigieren (daher zeitliche Differenz). Durch Wiederholung dieses Verfahrens konvergiert die geschätzte Wertfunktion schrittweise gegen die optimale Wertfunktion. Die Werte werden wie beim Q-Learning tabellarisch gespeichert. TD-Learning kann auf V- als auch Q-Funktion angewandt werden und hat den Vorteil, dass kein vollständiges Wissen über das Modell vorhanden sein muss, da iterativ die Wertfunktion durch die tatsächlich erreichten Belohnungen und den beobachteten Übergangswahrscheinlichkeiten (also der Erfahrung durch Handlungen) gelernt werden. TD(0) ist eine Art des TD-Lernens, dass sich bei der Bewertung nur auf die Belohnung einen Schritt in Zukunft (zum Zeitpunkt $t + 1$) beschränkt und weitere zukünftige Folgezustände nicht berücksichtigt [2, 119–128]. TD-Learning wurde bspw. sehr erfolgreich für das Brettspiel Backgammon (dynamisches Spiel mit perfekten Informationen) eingesetzt. Die Besonderheit war, dass für die Schätzung der Wertfunktion ein tiefes neuronalen Netz anstelle tabellarischer Methoden eingesetzt wurde, welches mittels TD-Fehler trainiert wird. Das Programm TD-Gammon kann auf dem Niveau der weltbesten Backgammonspieler entscheiden und hat nachhaltig die Szene der menschlichen Expertenspieler geprägt, da es bspw. Startzüge spielte, die unter Experten nicht üblich wahren und seitdem übernommen wurden [2, S. 421-426, 31].

Q-Learning[21] (deutsch: Q-Lernen) ist eine wertbasierte vom TD-Learning abgeleitete Lernmethode, welche iterativ die Bewertung jeder Aktion in jedem Zustand (mittels Q-Funktion) ermittelt und tabellarisch abspeichert, wobei ein Tabelleneintrag die Bewertung für eine Aktion in einem Zustand ist. Für die episodische Optimierung werden die Werte der Q-Funktion und der optimalen Q-Funktion verglichen und die Differenz wird als Verlust angegeben. Dieser Verlust wird über die Episoden hinweg minimiert, bis sich die Q-Funktion der optimalen Q-Funktion annähert und dadurch eine optimale Strategie erreicht wird. Ein offensichtlicher Nachteil des tabellarischen Q-Learning ist es, dass die Zustands- und Aktionsmenge durch den verfügbaren Speicher begrenzt wird, weil die Tabelle stark mit der Zustands- und Aktionsmenge wächst (100 Zustände mit jeweils 100 Aktionen entspricht eine Tabelle mit 10000 Zellen mit Q-Werten). Daher ist Q-Learning in dieser Form meist nur in kleinen endlichen MDPs effizient anwendbar [2, S. 131-132, 71]. Eine Alternative ist das ***Deep Q-Learning*** (deutsch: tiefes Q-Lernen) - eine Lernmethode die Q-Learning mit tiefen neuronalen Netzen (siehe Abschnitt 3.5) verbindet. Diese neuronalen Netzte lernen eine Schätzung der Q-Werte für Zustands-Aktions-Paare, anstatt die Q-Werte tabellarisch zu speichern.

[21] Hier wird der englische statt dem deutschen Begriff verwendet, da es zum Eigennamen für diese Lernmethode geworden ist.

Dadurch bleibt Deep Q-Learning in endlichen und stetigen MDP anwendbar [1, S. 368]. Deep Q-Learning verwendet ein CNN (siehe Abschnitt 3.5) zur Dimensionsreduktion (siehe Abschnitt 3.1), um eine hohe Anzahl von Zustand-Aktionspaaren zu erlernen. Erstmals wurde das Deep Q-Learning in dieser Form erfolgreich von V. Mnih et al. [25] in 2013 für sieben der Atari 2600 Videospiele angewandt. Nach einigen hundert gelernten Spieldurchläufen spielte das Modell bereits besser als menschliche Spieler. Sie formulierten als erste das ***Deep Q-Network***, eine Weiterentwicklung des Deep Q-Learning, wobei das neuronale Netz auf eine bestimmte Art trainiert wird.

3.4.4 Lösen eines komplexen Multi-Agenten MDPs ohne vollständiges Wissen und mit imperfekten Informationen (State of the Art)

Die vorgestellten Lernmethoden vereint alle eine Tatsache: Sie sind auf Umgebungen mit einem Agenten (Single-Agent Umgebungen) ausgelegt. In nicht statischen Umgebungen mit mehr als einem Akteur, sind die zuvor vorgestellten Methoden in einem POMDP nicht mehr zuverlässig einsetzbar. Das trifft jedoch nicht auf die folgenden Lernmethoden zu.

Strategiebasierte Lernmethoden im RL approximieren iterativ eine optimale Strategie, ohne ein explizites Modell der optimalen Wertfunktionen zu verwenden. Dadurch haben sie den Vorteil, dass sie auf hochdimensionale, als auch stetige MDPs anwendbar sind, da keine Zustandswerte gespeichert werden, sondern die Funktion der optimalen Strategie approximiert wird [1, S. 369]. Ein weiterer Vorteil ist, dass das ***Exploration-Exploitation-Dilemma*** (siehe Abschnitt 2.3) nicht zwingend besteht, da es in einer stochastischen Strategie durch die Wahrscheinlichkeitsverteilung über die Aktionen zu ausreichend Exploration kommen sollte.

Die ***Policy-Gradient-Methode (englisch Strategie-Gradienten)*** ist eine strategiebasierte modellfreie Lernmethode, bei der mittels Gradienten-Verfahren der Anstieg der Strategiefunktion ermittelt wird und die erwartete kumulative Belohnung so angepasst wird, dass sie iterativ gegen eine optimale Strategie konvergiert. Policy-Gradient-Methoden lassen sich auch in POMDPs gut anwenden [2, S. 321-326].

Extensive Form Fictitious Play (XFP) eine Lernmethode des RL, die auf dem spieltheoretischen Algorithmus des fitkiven Selbstpiels (englisch Fictitious Play) basiert. Es erweitert diesen Algorithmus auf dynamische Spiele (auch extensive Form genannt, siehe 2.1). XFP approximiert eine optimale Strategie, indem die Strategien der Gegenspieler modelliert werden und daraufhin die eigene Strategie angepasst wird [36]. ***Neural Fictitious Self-Play (NFSP)*** ist eine Erweiterung des XFP und verbindet dieses zusätzlich mit einem Deep Q-Network [34].

3.5 Künstliche Neuronale Netze

Künstliche neuronale Netze (KNN oder NN) sind ein Teil des Maschinellen Lernens. KNN wurden ursprünglich vom Aufbau des menschlichen Gehirns inspiriert, eignen sich jedoch nur

bedingt, um die Funktionsweise des menschlichen Gehirns selbst zu beschreiben [3, S. 90]. Kombiniert mit anderen maschinellen Lernverfahren lassen sich durch NN bspw. nichtlineare Abbildungen auf Daten erlernen. Sie sind sehr gut für die Extraktion oder Anwendung von Mustern auf Daten geeignet [3, S. 2]. NN sind nicht auf eines der vorgestellten Lernverfahren oder eine bestimmte Art von Problemstellungen beschränkt, sondern können regelrecht universell eingesetzt werden [3, 71–72, 76]. Dabei muss jedoch abgewägt werden, ob ein NN gegenüber anderen Methoden angebracht ist, da durch ein NN unter anderem meist ein exponentiell steigender Rechenaufwand mit steigender Anzahl an Neuronen und Schichten für die Lernphase anfällt [3, S. 90].

Das erste neuronale Netz wurde vom Psychologen F. Rosenblatt [17] im Jahr 1958 entwickelt und wird das ***Perceptron*** genannt, anhand dessen sich die Bestandteile und Grundfunktionen des neuronalen Netzes im Folgenden gut erklären lassen (siehe Abbildung 3.8).

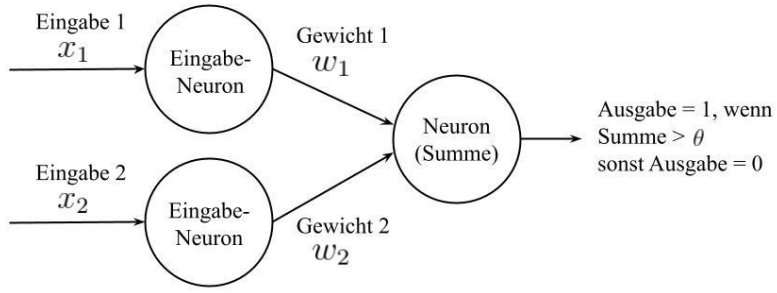


Abbildung 3.8 Der schematische Aufbau eines Perceptrons. Kreise sind Neuronen. Neuronen, die (bildlich) vertikal übereinanderstehen, gehören zu einer Schicht. Der Informationsfluss geht von links nach rechts. Die Neuronen der Eingabeschicht erhalten eine numerische Eingabe x von extern. Die Pfeile sind die Verbindungen zwischen Neuronen, die mit Gewicht w gewichtet werden. Das Neuron der Ausgabeschicht verwendet die Schwellenwertfunktion als Aktivierungsfunktion: Es gibt einen binären Ausgabewert aus, der vom Schwellenwert θ abhängig ist [3, S. 19].

Ein Perceptron ist ein KNN mit zwei ***Schichten,*** einer ***Eingabeschicht*** und einer ***Ausgabeschicht***. Eine Schicht enthält ***Neuronen*** (auch Knoten genannt), die mit anderen Neuronen anderer Schichten (feed-forward) oder auch mit sich selbst (recurrent) verbunden sind. Die Eingabeschicht des Perceptrons enthält mindestens zwei Neuronen, die numerische Eingabewerte x erhalten und diese an das Neuron der Ausgabeschicht weitergeben. Die Verbindung (auch gerichtete Kante genannt) zwischen Neuronen enhält ein ***Gewicht*** w, mit denen das Signal, welches von einem zum anderen Neuron übergeht, variabel gewichtet wird. Das Neuron der Ausgabeschicht generiert eine Ausgabe anhand der gewichteten Signale, die es von anderen Neuronen erhalten hat und wendet auf diese Signale eine Aktivierungsfunktion an. Eine ***Aktivierungsfunktion*** nimmt die Summe der gewichteten Eingabewerte eines Neurons und wendet eine zuvor festgelegte mathematische Funktion an, aus der die Ausgabe (Aktivierung) des Neurons resultiert. Im Fall des Perceptrons ist die

Aktivierungsfunktion eine Schwellenwertfunktion, die für alle Werte über dem Schwellenwert[22] θ den Wert eins und für alle anderen Werte den Wert null ausgibt. Das Perceptron könnte man bspw. als Formel 3.7 formulieren [3, S. 31, 3, S. 18-26].

$$\text{Ausgabe} = 1, \text{wenn } x_1 w_1 + x_2 w_2 > \theta, \text{sonst Ausgabe} = 0$$

Formel 3.7 Das Perceptron beispielhaft als Formel ausgedrückt.

Damit das Perceptron lernt, werden die Gewichte w und die Schwelle θ so lange iterativ angepasst, bis die Fehlerrate auf geringem Niveau stabil ist (bspw. mittels Delta-Regel [3, S. 48-51]). Die ***Fehlerrate*** ist die durchschnittliche Zahl der Ergebnisse, die nicht dem gewünschten Ergebnis entsprechen. Es ist also immer das Ziel bei der Lernphase eines NN, die individuellen Gewichte der Neuronen-Verbindungen iterativ so anzupassen, dass die Fehlerrate möglichst gering ist und stabil bleibt. Es lässt sich für binäre Klassifizierung in linear trennbaren Mengen im SL einsetzen (siehe Abbildung 3.9) (siehe Abschnitt 3.2) [1, S. 210].

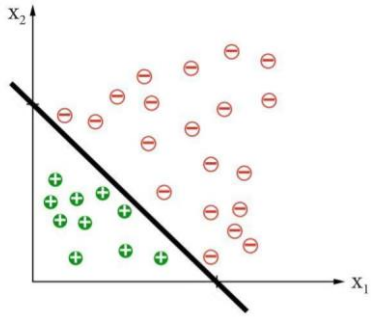


Abbildung 3.9 Eine beispielhafte lineare Klassifizierung, die mittels Perceptron möglich ist. Die Datenpunkte müssen anhand der Eingabewerte (Merkmale) x_1 und x_2 verteilt sein [1, S. 210].

Ein NN ist nicht auf das Perceptron beschränkt. Die Neuronen haben zusätzlich einen eigenen ***Bias-Wert,*** eine individuelle Verschiebung der Aktivierung des Neurons. Je nach Darstellung wird der Bias-Wert auch als eigenes Neuron umgesetzt, wobei die Funktion aber identisch ist. NN bestehen immer aus einer Eingabeschicht, einer Ausgabeschicht und meist mehreren Schichten dazwischen, die als ***versteckte Schichten*** bezeichnet werden. Ein KNN mit mehr als drei versteckten Schichten wird als ein tiefes NN bezeichnet. Die Anzahl der Neuronen jeder Schicht, als auch die Verbindungen zwischen Neuronen, kann an den Verwendungszweck angepasst werden [3, S. 26]. Daraus ergeben sich verschiedene Typen von NN, die sich in der ***Netz-Topologie*** unterscheiden, also der Anzahl der Schichten, der Anzahl der Neuronen jeder Schicht und die Vernetzung der Neuronen untereinander [3, S. 61]. In einem ***Feed-Forward-Netz (FFN)*** (deutsch: Vorwärtsnetz oder Netz mit Vorwärtskopplung) geht der Informationsfluss nur vorwärts, von der Eingabeschicht über die versteckten Schichten bis zur Ausgabeschicht. Dabei werden vergangene Zustände (anderer Schichten) nicht berücksichtigt. Dem gegenüber stehen ***rekurrente Netze (RNN),*** bei denen Neuronen Verbindungen zu Neuronen der vorherigen Schicht oder mit sich selbst haben (Rückwärtskopplung) und vergangene Zustände berücksichtigt werden [3, S. 62-65]. Diese

[22] griechischer Buchstabe Theta

verschiedenen Arten der Verbindungen in einem NN erfordern wiederum unterschiedliche Lernmethoden zum Lernen der Gewichte (bspw. Backpropagation [3, S. 51-60] für FFN), wobei besonders rekurrente Netze komplexere Lernprozesse mit sich bringen, da die Feedbackschleifen des Netzes bspw. in einzelne Netze aufgeklappt werden müssen, die separates Training erfordern [3, S. 64-65].

In der Literatur werden eine Vielzahl verschiedener Netzwerktypen diskutiert, die sich in ihrem Aufbau unterscheiden und jeweils für unterschiedliche Anwendungsfälle ausgelegt sind. Das ***Hopfield-Netz*** bspw., ein rekurrentes Netz, in dem Neuronen nur binäre Werte annehmen, ist gut für die Anwendung als assoziativer Speicher von kleinen einfachen Mustern geeignet [1, S. 292-297, 72]. Oder das ***Kohonen-Netz*** (auch engl. Self-Organizing Map genannt) ist ein Feed-Forward-Netz, welches sich im UL für Clusteringaufgaben einsetzen lässt und in der Lernphase die Gewichtsvektoren der Neuronen, welche benachbarte Datenpunkte abbilden, iterativ zueinander bewegt, weshalb es selbstorganisierend genannt wird [73]. Eine besonders vielversprechende Netzart ist das ***Convolutional Neural Network (CNN)*** (deutsch: gefaltetes neuronales Netz), welches sich besonders für Anwendungen mit hochdimensionalen Daten eignet (siehe Abschnitt 3.1). Erstmals wurde diese Netzart von LeCun et al. [8] im Jahr 1989 zum Erkennen einzelner Zeichen in menschlicher Handschrift eingesetzt. CNNs verwenden mathematische Faltung, um Merkmale aus den Eingabedaten zu extrahieren und haben so zum Durchbruch der automatischen Objekterkennung auf Bildern[23] im SL beigetragen [3, S. 65-70]. Die Netz-Topologie des CNN (siehe Abbildung 3.10) besteht aus Merkmalsschichten, die nur mit wenigen Neuronen anderer Schichten vernetzt sind und bspw. einen linearen Filter auf die Eingabedaten anwenden. Auf Faltungsschichten folgen Poolingschichten, die auf die Ausgabe der Faltungsneuronen wiederum eine Mittelwerts- oder Maximum-Funktion anwenden. Die Faltungs- und Pooling-Schichten sind für die Dimensionsreduktion zuständig. Die darauffolgenden Schichten sind wiederum voll vernetzt, da sie den eigentlichen Lernprozess umsetzen [1, S. 325-327].

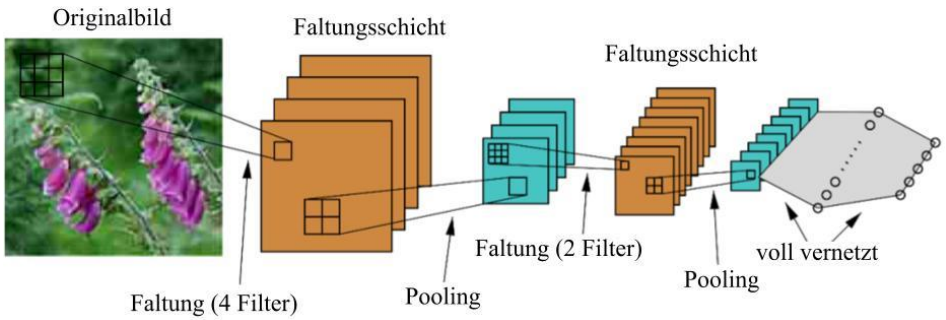


Abbildung 3.10 Der schematische Aufbau eines CNN mit insgesamt sieben Schichten, bestehend aus zwei Faltungsschichten, gefolgt von jeweils einer

[23] Bilder bedeuten hochdimensionale Daten, da pro Pixel drei Farbwerte (RGB) gespeichert werden. Selbst mit Graustufen (nur Helligkeit des Pixels), entstehen durch Bilder selbst in niedriger Skalierung eine unpraktikabel hohe Daten-Dimensionen. Folge dessen sind mehr Rechenaufwand für die Lernphase notwendig und oft eine instabile Konvergenz der Fehlerrate [1, S. 326].

Pooling-Schicht und zwei voll-vernetzten Schichten. In diesem Beispiel wird ein Bild mit hochdimensionalen Daten durch die dargestellten Schichten des CNNs auf eine niedrige Dimension reduziert [1, S. 325].

Das CNN eignet sich für die Anwendungen von hochdimensionalen Trainingsdaten. Es ist in diesem Kontext so interessant, da es bspw. unter anderem von Deep-Q-Learning (siehe Abschnitt 3.4) angewandt wird, als auch von Alpha-Go [26] zur Dimensionsreduktion des Spielzustands auf dem 19x19 Felder großen Spielbrett.

Je nach Komplexität des Netzes ist es nicht mehr nachvollziehbar, wie einzelne Ergebnisse konkret entstehen und welche Faktoren dabei ausschlaggebend waren. Daher werden NN oft als Blackbox bezeichnet, was ein Problem in Anwendungen wie gesundheitlichen Diagnosen darstellt, doch im Rahmen dieser Arbeit nicht weiter relevant ist. [3, S. 91-92]

3.6 Zwischenfazit: Maschinelles Lernen

Maschinelles Lernen ist ein Teilgebiet der KI, das sich mit der Frage beschäftigt, wie Computerprogramme erstellt werden können, die ihre Leistung durch Erfahrung verbessern können [42, S. 17]. Dazu wird ein mathematisches Modell erstellt, das eine Abbildung der Daten lernt [3, S. 8]. Um eine gute Generalisierungsfähigkeit des Modells zu erreichen, müssen sowohl Überanpassung als auch Unteranpassung vermieden werden [3, S. 94]. Daten enthalten Merkmale, anhand derer Zusammenhänge gelernt werden [1, S. 205]. Die Anzahl der Merkmale wird oft in Dimensionen angegeben. Daten mit einer sehr hohen Dimension erfordern lange Trainingszeiten, was sich durch eine Dimensionsreduktion der Daten vermeiden lässt [1, S. 204]. Das maschinelle Lernen unterteilt sich in drei große Lernverfahren, die jeweils unter anderen Vorraussetzungen und anderen Methoden eine Abbildung der Daten erlenen.

Das SL nutzt beschriftete Trainingsdaten, um Regressions- (numerische Prognose) oder Klassifikationsaufgaben (Einteilung in definierte Klassen) zu lösen [64]. Dazu wurden mehrere Lernmethoden erwähnt, die bei bestimmten Anwendungsfällen und Dimensionen anwendbar sind. Damit das SL-Modell eine Abbildung lernen kann, sind je nach Komplexität der Aufgabenstellung große Mengen an Trainingsdaten notwendig. Dies kann ein Problem sein, da das Erstellen beschrifteter Trainingsdaten mit richtigen Antworten mit großem Aufwand verbunden sein kann. Je nach Art der Daten lassen sich für dieses Problem bspw. Verfahren der Datenvermehrung anwenden [1, S. 249-250].

Das UL nutzt Trainingsdaten, bei denen die richtigen Antworten nicht bekannt sind. Dazu werden durch Methoden zur Clusteranalyse Zusammenhänge in Trainingsdaten ermittelt und Datenpunkte anhand ihrere Merkmale in Klassen (Cluster) zugeordnet, die zuvor nicht festgelegt wurden [64]. Dies ist bspw. mittels der Lernmethoden k-means[68, S. 272-274] (wenn Anzahl der Cluster bekannt) oder EM-Algorithmus [1, S. 262-263] (wenn Anzahl der Cluster unbekannt) möglich. UL lässt sich

auch zur Dimensionsreduktion in hochdimensionalen Daten einsetzen: Der t-SNE-Algorithmus ordnet hochdimensionale Daten in einem zweidimensionalen Streudiagram an, ohne dass dabei nicht-lineare Zusammenhänge der Datenpunkte verloren gehen [68, S. 269-270].

Das RL ist ein äußerst vielversprechendes Lernverfahren, da es bereits mehrfach erfolgreich für KIs in Spielen eingesetzt wurde (TD-Gammon [31], Atari 2600 [25], AlphaGo [26], Dota 2 [28]). Im RL lernt ein Modell durch Interaktion mit einer Umgebung. Um eine Umgebung als ein sequentielles Entscheidungsproblem formal zu beschreiben, wird ein MDP der Umgebung erstellt. Der MDP definiert Zustände, Aktionen und Belohnungen. In einem Zustand sind Aktionen möglich, die zu einer Belohnung und einem Folgezustand führen, die mit einer gewissen Übergangswahrscheinlichkeit tatsächlich erreicht werden. Belohnungen resultieren aus den Aktionen und geben dem Agenten so Feedback darüber, wie gut oder schlecht diese Aktion sich auf sein Ziel ausgewirkt hat. Ein Zustand enthält Informationen, welche für die Bewertung des Zustands relevant sind. Mittels MDP kann ein Agent eine Aktion wählen, eine Belohnung für die Aktionen erhalten und sich anschließend im nächsten Zustand befinden [2, S. 47-48]. Im MDP sind Methoden der dynamischen Programmierung üblich, um die Aktionen und Zustände (unter Berücksichtigung der Übergangswahrscheinlichkeiten) anhand ihrer zukünftigen erwarteten Belohnung zu bewerten (Wertfunktionen) [1, S. 358, 6]. Es ist immer das Ziel im MDP eine optimale Strategie zu finden, welche immer die Aktionen wählt, die die maximale zukünftige Belohnung erreicht. Zur Formalisierung einer optimalen Strategie, wird die Bellman-Optimalitätsgleichung angewandt, aus der sich auch optimale Wertfunktionen ableiten lassen [2, S. 62-63, 6]. In diesem theoretischen Rahmen lassen sich mit vollständigem Wissen über die Umgebung und unbegrenzter Rechen- und Speicherkapazität immer eine optimale Strategie ermitteln [1, S. 356-357]. Diese Annahmen treffen jedoch auf praktische Anwendungen mit komplexen Umgebungen nur selten zu, da meist nur unvollständiges Wissen über die Umgebung gegeben ist und Rechenzeit, als auch Speicher begrenzt sind [2, S. 69]. Weitere Hürden sind das Credit-Assignment-Problem [53, 20–28], wenn der Agent nicht in der Lage ist, Aktionen korrekt zu bewerten (bspw. verzögerte Belohnung), und das Exploration-Exploitation-Dilemma [1, S. 367-368], die Unsicherheit über die Qualität der Bewertung von Aktionen und Zuständen. Daraufhin werden wertbasierte und strategiebasierte Lernmethoden vorgestellt, die auch unter unvollständigen Informationen eine optimale Strategie approximieren können. Wertbasierte Methoden basieren darauf, dass durch eine optimale Wertfunktion immer auch eine optimale Strategie gegeben ist [2, S. 68]. Das Q-Learning und TD-Learning sind wertbasierte modellfreie Lernmethoden, die eine optimale Wertfunktion iterativ approximieren und die errechneten Werte tabellarisch abspeichern [2, S. 131-132, 2, 119–128]. Diese tabellarische Speicherung ist limitierend in MDPs mit sehr großen Zustandsräumen, was jedoch durch die Erweiterung mit tiefen neuronalen Netzen (Deep Q-Learning) gelöst werden kann [1, S. 368]. Die Strategie-Gradienten-Methode ist eine modellfreie strategiebasierte Lernmethode, die eine

optimale Strategie approximiert anstatt einer Wertfunktion und eignet sich für hochdimensionalen, als auch stetige MDPs [2, S. 321-326].

KNNs sind ursprünglich vom menschlichen Gehirn inspiriert [3, S. 90] und werden mit anderen Lernverfahren kombiniert, um nicht lineare Abbildungen zu lernen [3, S. 2]. Anhand des Perceptrons [17] wurde der allgemeingültige Aufbau eines NNs erläutert. Ein NN ist organisiert in Schichten, die aus einer beliebigen Anzahl von Neuronen bestehen, welche mit Neuronen der nächsten Schicht (Feed-Forward-Netz) oder auch mit Neuronen vorherigen Schichten und sich selbst (rekurrentes Netz) verbunden sind, wobei jede Verbindung individuell gewichtet wird [3, S. 62-65]. Ein Neuron empfängt Eingabewerte, die es aufsummiert, mit der individuellen Gewichtung multipliziert, den Bias-Wert addiert und darauf eine Aktivierungsfunktion anwendet. Der finale numerische Wert ist die Ausgabe (auch Aktivierung genannt) des Neurons, die an andere Neuronen weitergegeben wird. Die Aktivierungsfunktion ist für alle Neuronen gleich [3, S. 31, 3, S. 18-26]. Die Gewichte der Verbindungen zwischen Neuronen sowie die Bias-Werte der Neuronen sind individuell und anpassbar. Damit ein NN lernt, werden die Gewichte der Verbindungen sowie die Bias-Werte durch Lernmethoden, wie dem vorgestellten Backpropagation-Algorithmus, iterativ angepasst, bis es zu einer stabilen Fehlerrate konvergiert [1, S. 210]. Abschließend wurden spezielle neuronale Netze erwähnt, wie das Hopfield-Netz [1, S. 292-297, 72] und das Kohonen-Netz [73]. Das CNN wurde etwas genauer betrachtet, wegen seiner vielseitigen Anwendbarkeit für die Dimensionsreduktion und der erfolgreichen Anwendung in anderen Spiele-KIs [25, 26], als auch Deep Q-Learning [1, S. 368].

4 Maschinelles Lernen in einem Kartenspiel mit imperfekten Informationen einsetzen

In diesem Kapitel wird anhand eines Kartenspiels mit imperfekten Informationen erläutert, wie einige der vorgestellten Algorithmen angewandt werden. Zunächst werden in Abschnitt 4.1 alle Anforderungen aufgelistet und daran die Rahmenbedingungen für eine Umsetzung begründet. In Abschnitt 4.2 wird gezeigt, welche Schritte notwendig sind, um die Ergebnisse reproduzieren zu können. Abschnitt 4.3 zeigt, wie mehrere Algorithmen eingesetzt wurden, um Modelle für das Kartenspiel Leduc Poker zu trainieren. Der letzte Abschnitt 4.4 widmet sich dem qualitativen Vergleich der trainierten Modelle.

4.1 Anforderung an die Umsetzung

4.1.1.1 Die Umsetzung muss auf einem handelsüblichen Computer problemlos durchführbar sein.

Viele der erwähnten Modelle, die Expertenspieler in komplexen Spielen wie Go und Texas No Limit Hold'em Poker[24] schlagen konnten, wurden auf Supercomputern trainiert. Diese Voraussetzung kann im Rahmen dieser Arbeit nicht erfüllt werden. Daher ist es eine Anforderung, dass die gesamte Umsetzung auf einem handelsüblichen Computer durchführbar sein muss.

4.1.1.2 Die Umsetzung muss ein maschinelles Lernverfahren anwenden, das nicht auf Trainingsdaten angewiesen ist.

Die maschinellen Lernverfahren SL (siehe Abschnitt 3.2) und UL (siehe Abschnitt 3.3) sind auf Trainingsdaten angewiesen. Für manche Spiele mit perfekten Informationen wie bspw. Schach sind Spielverläufe von Expertenspielern in einem einheitlichen Format digitalisiert und online frei zugänglich [75]. Diese Spielverläufe lassen sich sehr gut als Trainingsdaten einsetzen, da sie in ausreichender Menge und Qualität vorhanden sind. Dies ist bei Kartenspielen mit imperfekten Informationen leider nicht der Fall. Es gibt zwar kleine Trainingsdatensätze von Spielverläufen, doch nicht in ausreichender Menge. Auch wenn es Bestrebungen gibt, das Erstellen von Trainingsdaten für Kartenspiele wie Bridge zu erleichtern [76], ist es im Rahmen dieser Arbeit jedoch nicht sinnvoll. Aus diesem Grund ist es für die Umsetzung von maschinellem Lernen in einem Kartenspiel nicht angebracht, ausschließlich Methoden der Lernverfahren SL oder UL anzuwenden. Das RL (siehe Abschnitt 3.4) hingegen lernt aus Interaktionen in einer Umgebung und ist nicht auf vorgegebene

[24] Die Kartenspiele-KI Libratus [62, 74] verwendete während der Trainingsphase ~ 3 Millionen CPU-Kern Stunden und ~ 25 Terrabyte Arbeitsspeicher.

Trainingsdaten angewiesen. Dabei dient das Kartenspiel als Umgebung für den RL-Agenten. Aus diesem Grund ist es angebracht Methoden des RL anzuwenden.

4.1.1.3 Für die Umsetzung soll ein existierendes Framework verwendet werden.

Es ist nicht sinnvoll für diese Umsetzung das Kartenspiel, die spieltheoretischen Algorithmen und die maschinellen Lernmethoden selbst zu implementieren. Daher ist es eine Anforderung, dass ein existierendes Framework verwendet wird, mit dem auf bereits implementierte Kartenspiele, Algorithmen und Lernmethoden zurückgegriffen werden kann. Dazu kommen die folgenden Frameworks in Frage: RLCard [77] und OpenSpiel [35].

RLCard ist ein Python-Framework mit zehn Kartenspielen und vier Algorithmen. Es bietet eine einfache einsteigerfreundliche Bedienung und eine optionale graphische Benutzeroberfläche, mit der die Spielverläufe der lernenden Agenten nachvollziehbar sind. Es nutzt das Pyhton-Framework PyTorch zum Trainieren der Modelle.

OpenSpiel ist ebenfalls ein Python-Framework mit 17 Kartenspielen und 51 Algorithmen. Die Bedienung ist wesentlich anspruchsvoller. Es nutzt das Python-Framework TensorFlow zum Trainieren der Modelle. In OpenSpiel sind die Spiele in C++ implementiert und durch eine API an Python angebunden. Das hat den Vorteil, dass die Laufzeit der Spiele im Training wesentlich schneller ist als bei einer reinen Python-Implementierung.

Für diese Umsetzung wurde OpenSpiel verwendet, da die Zahl der implementierten Algorithmen im Gegensatz zu RLCard wesentlich höher ist.

4.1.1.4 Das Kartenspiel muss ein dynamisches nicht-kooperatives Spiel mit symmetrischen imperfekten Informationen sein und einen angemessenen Komplexitätsgrad haben.

Das Kartenspiel muss ein dynamisches nicht-kooperatives Spiel sein. Außerdem müssen imperfekte Informationen in diesem Spiel vorliegen und symmetrisch für alle Spieler sein (siehe Abschnitt 2.1). Diese Eigenschaften treffen auf die meisten klassischen Kartenspiele zu. Eine weitere Anforderung ist, dass das Kartenspiel eine angemessene Komplexität hat, damit die Modelle auf einem handelsüblichen Computer effizient trainierbar sind. Daher wurde sich für diese Umsetzung für das Kartenspiel Leduc Poker entschieden.

Leduc Poker ist eine vereinfachte Version des Kartenspiels Texas Hold'em Poker. Es eignet sich besonders gut für Studien und Analysen in der Spieltheorie und ist quasi die Standart-Testumgebung für Spiele mit imperfekten Informationen. In Leduc Poker besteht das Kartendeck nur aus zwei Paaren von König, Dame und Bube, also insgesamt sechs Karten. Jedes Spiel wird mit zwei Spielern, zwei Runden, maximal zwei Einsätzen und Erhöhungen gespielt [14, S. 2].

4.1.1.5 Die Umsetzung muss Algorithmen der Spieltheorie und des maschinellen Lernens anwenden.

Wie zuvor erläutert, wenden viele der erfolgreichen Kartenspiel-KIs Methoden der Spieltheorie, als auch Methoden des maschinellen Lernens an. Aus diesem Grund ist es eine Anforderung, dass Algorithmen beider Forschungsfelder berücksichtigt werden. Von den 51 Algorithmen, die in OpenSpiel implementiert sind, eignen sich 16 für Spiele mit imperfekten Informationen. Davon wiederum wurden neun in der Umsetzung angewandt. In der folgenden Tabelle werden diese Algorithmen nochmals kurz erläutert (siehe auch Abschnitt 2.3 und 3.4.4).

Algorithmus	**Beschreibung**
Counterfactual Regret Minimization (CFR)	CFR erstellt ein tabellarisches Modell der Gegenspieler und konvergiert zu einem Strategie-Gleichgewicht durch iteratives Minimieren des Regrets.
Counterfactual Regret Minimization against Best Response (CFR-BR)	CFR-BR erweitert CFR, indem es die beste Antwort auf die Strategie des Gegenspielers berücksichtigt.
Counterfactual Regret Minimization Plus (CFR-Plus)	CFR-Plus erweitert CFR durch ein Public Chance Sampling.
Deep Counterfactual Regret Minimization (Deep CFR)	Deep CFR erweitert CFR durch tiefe neuronale Netze.
Discounted Counterfactual Regret Minimization (DCFR)	DCFR erweitert CFR, indem zukünftiger Regret schwächer gewichtet wird.
Extensive Form Fictitious Play (XFP)	XFP erstellt ein Modell des Gegenspielers und passt die Strategie iterativ im Selbstspiel an.
Information Set Monte Carlo Tree Search (IS-MCTS)	IS-MCTS erweitert MCTS für Spielbäume mit imperfekten Informationen.
Neural Fictitious Self-Play (NFSP)	NFSP erweitert Fictitious Play durch ein Deep Q-Network.
Policy Gradient (PG)	Policy-Gradient optimiert die Strategiefunktion, indem es durch ein Gradient-Verfahren den Anstieg der Strategiefunktion ermittelt. So kann die Strategie iterativ verbessert werden.

Die folgende Tabelle erläutert, welche Algorithmen aus OpenSpiel in der Umsetzung nicht berücksichtigt wurden, auch wenn sie in einem Spiel mit imperfekten Informationen einsetzbar wären.

Algorithmus (nicht angewandt)	**Begründung warum nicht anwendbar**
Deep Q-Networks	Das Deep Q-Network konvergiert nicht zuverlässig zu einem Strategie-Gleichgewicht. Es ist für Umgebungen mit nur einem Agenten ausgelegt, ist jedoch kombiniert mit anderen Algorithmen wie NFSP anwendbar.
Ephemeral Value Adjustment (EVA)	Diese Implementation des Algorithmus ist nicht stabil. Während des Trainings stürzte das Programm mehrfach ab.
External/Outcome sampling Monte-Carlo CFR	Dieser Algorithmus konnte innerhalb dieser Umsetzung keine guten Ergebnisse in Leduc Poker erzielen.
Monte-Carlo-Tree-Search (MCTS)	MCTS ist auf Suchbäume mit vollständigen Informationen ausgelegt und erzielt keine guten Ergebnisse in einem Spiel mit imperfekten Informationen (siehe IS-MCTS)
Neural Replicator Dynamics (NeuRD)	Die Implementation dieses Algorithmus wird von OpenSpiel als nicht stabil eingestuft.
Policy Iteration, Value Iteration	Policy Iteration und Value Iteration sind nicht zuverlässig in einem POMDP (siehe Abschnitt 3.4.3) anwendbar und sind auf Umgebungen mit einem Agenten ausgelegt.
Q-Learning, Deep Q-Learning	Q-Learning und Deep Q-Learning sind nicht zuverlässig in einem POMDP (siehe Abschnitt 3.4.3) anwendbar und sind auf Umgebungen mit einem Agenten ausgelegt.
Regression CFR	Regression CFR konnte innerhalb dieser Umsetzung keine guten Ergebnisse in Leduc Poker erzielen.
Regret Matching (RM)	Regret Matching ist durch CFR und dessen Erweiterungen überflüssig geworden.

4.1.1.6 Es sollen mehrere Algorithmen und Lernverfahren in einem Spiel eingesetzt und anhand einer einheitlichen Metrik verglichen werden.

Wie bereits erwähnt gibt es mehrere Lösungsansätze für eine Kartenspiel-KI. Aus diesem Grund ist es eine Anforderung, dass mehrere Algorithmen und Lernmethoden in Leduc-Poker eingesetzt und miteinander verglichen werden. Als Metrik eignet sich die ***Nash-Konvergenz*** [44], die angibt, wie nah der Agent dem spieltheoretischen Strategie-Gleichgewicht (siehe Abschnitt 2.1) kommt.

4.2 Technische Vorbereitung

Dieser Abschnitt beschreibt, welche Schritte notwendig sind, um die Ergebnisse reproduzieren zu können. Die folgende Anleitung setzt minimale Kenntnisse im Umgang mit der Kommandozeile in Windows und Linux voraus.

4.2.1.1 Eine Virtuelle Maschine unter Windows 10 einrichten (WLS2), Ubuntu installieren und GPU-Unterstützung einrichten

Die Umsetzung fand auf einem Windows 10 Betriebssystem statt. Damit die Ergebnisse systemunabhängig sind, ist es sinnvoll eine virtuelle Maschine zu nutzen. Dazu bietet sich das in Windows 10 integrierte ***Windows-Subsystem-for-Linux (WSL)*** an, mit dem es sehr einfach ist, eine beliebige Linux-Distribution zu installieren, die parallel zu Windows läuft. Es ist wichtig, dass es sich um die aktuelle Version WSL2 handelt. Mit WSL2 wurde die Linux-Distribution Ubuntu in der Version 22.04 installiert (siehe Anleitung [78]). Das Trainieren der Modelle ist sehr rechenintensiv, kann aber durch die GPU beschleunigt werden. Damit WSL2 Zugriff auf die GPU hat, muss die GPU-Unterstützung separat eingerichtet werden (siehe Anleitung [79]).

4.2.1.2 Das OpenSpiel Framework installieren

Die Umsetzung basiert auf dem OpenSpiel Framework, in dem Kartenspiele und Algorithmen implementiert sind. OpenSpiel muss in der zuvor eingerichteten virtuellen Maschine installiert werden (siehe Anleitung [80]). Es ist äußerst wichtig, dass die OpenSpiel-Umgebung ohne Fehler kompiliert wird und keine Tests fehlschlagen (siehe Befehl: `ctest`).

4.2.1.3 Das Programm aus dem Anhang installieren

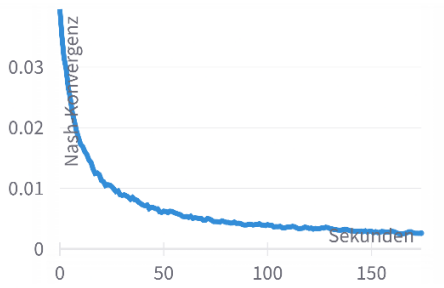
Im Anhang befinden sich die Python-Skripte und die trainierten Modelle, die in der Umsetzung verwendet wurden. Diese müssen ebenfalls in die virtuellen Maschine kopiert und installiert werden. Folgend eine kurze Erklärung des Programms aus dem Anhang: Mit dem Befehl `pip3 -r requirements.txt` werden alle notwendigen Python-Module installiert. Der Ordner `src/training` enthält alle Python-Skripte zum Trainieren der Modelle. In diesem Ordner sind die Dateien nach dem jeweiligen Algorithmus benannt. Das Bash-Skript `train_all_models.sh` führt alle Trainings-Python-Skripte nacheinander aus. Die Anzahl der Episoden im Training und andere Meta-Daten können über die Argumente (Flags) des Bash-Skripts variabel angepasst werden. Eine vollständige Referenz der unterstützten Flags ist in der Datei `src/shared_flags.py` enthalten. Die Ausgabe der Trainings-Python-Skripte wird automatisch im Ordner `logs` gespeichert. Der Ordner `trained_models` enthält die trainierten Modelle, die in der Umsetzung verwendet wurden. Um das Training der Modelle zu überwachen, wurde der Online-Service Weights & Biases [81] verwendet.

Dieser Service ist standardmäßig deaktiviert, kann aber mit dem Argument `--wandb_enable=True` für das Training aktiviert werden.

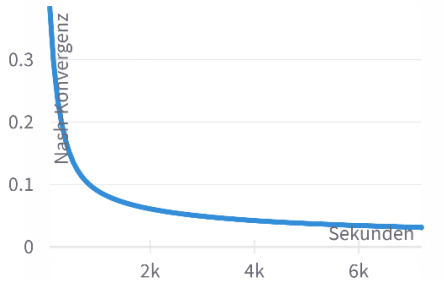
4.3 Anwenden der Algorithmen in Leduc Poker

Die neun vorgestellten Algorithmen wurden im Kartenspiel Leduc Poker angewandt. Wie zuvor erwähnt, lässt sich die Spielqualität der Algorithmen durch die Nash-Konvergenz messen. Je näher der Wert der Nash-Konvergenz dem Wert Null kommt, umso sicherer spielt der Algorithmus mit einem Strategiegleichgewicht. Die Algorithmen spielen während der Trainingsphase immer gegen sich selbst. Im Folgenden ist eine Auflistung der Ergebnisse. Ein Hinweis zu den Graphen: Die Graphen sind nicht einheitlich skaliert, da die qualitativen Ergebnisse sehr unterschiedlich sind. Jeder Graph wird in einer eigenen Skalierung dargestellt, damit die Kurve gut erkennbar bleibt.

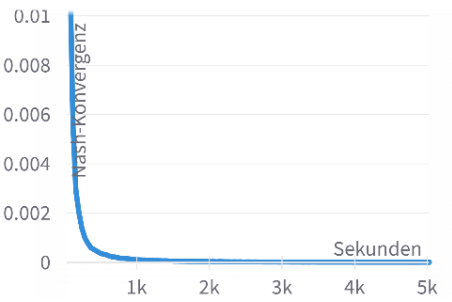
CFR CFR erreichte nach zwei Stunden Training eine stabile Nash-Konvergenz von ~0.02. In CFR sind keine weiteren Parameter einstellbar.



CFR-BR CFR-BR erreichte nach zwei Stunden Training eine stabile Nash-Konvergenz von ~0.03. In CFR-BR sind keine weiteren Parameter einstellbar.

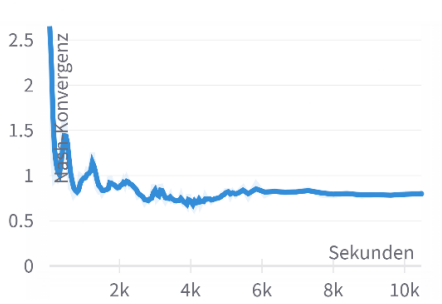


CFR-Plus CFR-Plus erreichte bereits nach 25 Minuten Training eine stabile Nash-Konvergenz bei etwa ~0.0006. Die finale Nash-Convergenz betrug ~0.0001 nach einer Stunde und fünfzehn Minuten. In CFR-Plus sind keine weiteren Parameter einstellbar.



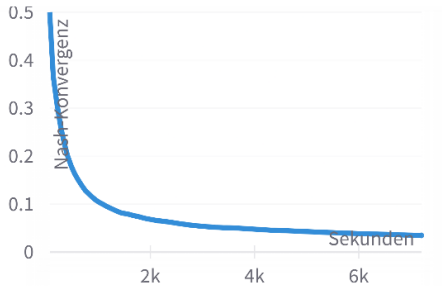
Deep-CFR

Deep-CFR erreichte nach drei Stunden Training eine stabile Nash-Konvergenz von ~0.8. Dazu wurde ein tiefes NN mit vier versteckten Schichten mit jeweils 16 Neuronen eingesetzt. Die Lernrate beträgt 0.001. Es sind noch weitere Parameter einstellbar.



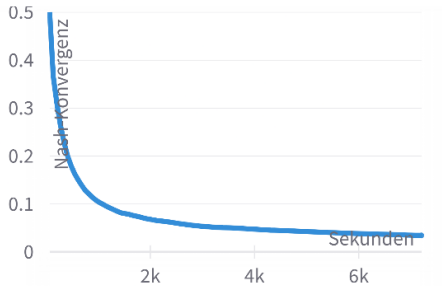
DCFR

DCFR erreichte bereits nach zehn Minuten Training eine stabile Nash-Konvergenz bei etwa ~0.00009. In CFR-Plus sind keine weiteren Parameter einstellbar.



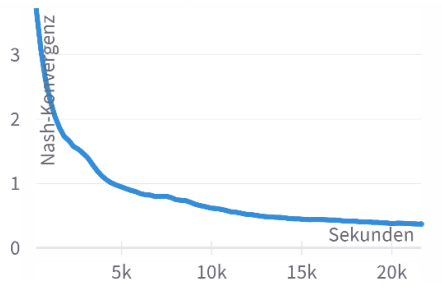
XFP

XFP erreichte nach zwei Stunden Training eine stabile Nash-Konvergenz bei etwa ~0.03. In XFP sind keine weiteren Parameter einstellbar.



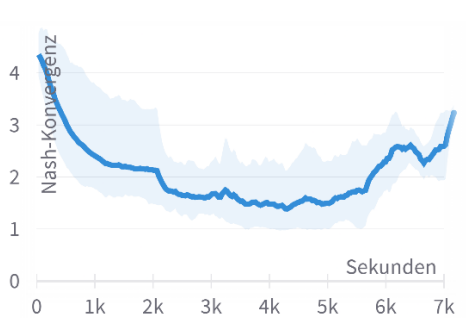
NFSP

NFSP erreichte nach sechs Stunden Training eine stabile Nash-Konvergenz bei etwa ~0.37. Dazu wurde ein NN eingsetzt, bestehend aus einer versteckten Schicht mit 128 Neuronen.



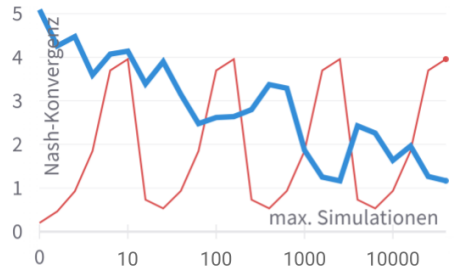
Policy-Gradient

Die Policy-Gradient Verfahren erreichten nach vier Stunden Training keine stabile Nash-Konvergenz. Es wurde ein NN eingesetzt, bestehend aus einer versteckten Schicht mit 128 Neuronen. Im Policy-Gradint-Verfahren wurden vier verschiedene Algorithmen (Advantage Actor-Critic [2, S. 331], Q-based Policy Gradient [82], Regret policy gradient und Regret Matching Policy Gradient [83]) angewandt, die jedoch ähnliche Ergebnisse erzielten. Der Graph stellt die durchschnittliche Nash-Konvergenz der vier Algorithmen dar.



IS-MCTS

IS-MCTS trainiert kein Modell. Daher muss der Graph für den IS-MCTS etwas angepasst werden. Im Graphen ist das Verhältnis der erreichten Nash-Konvergenz (blau) und die Spielverlauf-Simulationen zu sehen. Der rote Graph ist der Explorationsparameter (UCT Formel), der zwischen 0.2 und 4.0 variiert. Die finale Nash-Konvergenz liegt bei ~1.1.



4.4 Auswertung der Ergebnisse

Das Training der Algorithmen im Kartenspiel Leduc Poker hat zu sehr unterschiedlichen Ergebnissen geführt. In Abbildung 4.1 sind alle angewandten Algorithmen (außer IS-MCTS) in einem Graphen gegenübergestellt. Dargestellt wird das Verhältnis zwischen Nash-Konvergenz und Trainingszeit in Sekunden. Es ist auffällig, dass die Policy-Gradient-Verfahren, das NFSP und Deep-CFR keine sehr guten Ergebnisse erzielen konnten. Diese drei Algorithmen haben Gemeinsamkeiten: Sie verwenden NN und brauchten eine verhältnismäßig lange Trainingsphase, um zu einem stabilen Strategie-Gleichgewicht zu konvergieren. Bei den Policy-Gradient-Verfahren wurde die Nash-Konvergenz über den Trainingsverlauf sogar schlechter, was auf eine Überanpassung der NNs hinweisen könnte.

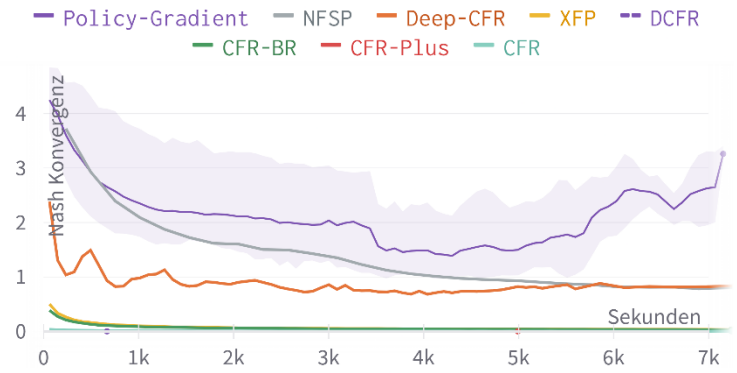


Abbildung 4.1 Alle angewandten Algorithmen (außer IS-MCTS) in einem Graphen gegenübergestellt. Dargestellt wird das Verhältnis zwischen Nash-Konvergenz und Trainingszeit in Sekunden. Diese Graphik zeigt nur einen Ausschnitt und nicht die komplette Trainingszeit von jedem Algorithmus.

In Abbildung 4.1 ist ebenfalls auffällig, dass die Verläufe der übrigen Algorithmen durch die Skalierung der Graphik ineinander verschwinden. In Abbildung 4.2 sind diese Algorithmen gegenübergestellt und auf eine kleinere Skala übertragen. Dadurch ist gut erkennbar, dass XFP und CFR-BR einen sehr ähnlichen Verlauf aufweisen. Beide konvergieren nach zwei Stunden Trainingszeit zu einem Nash-Konvergenz-Wert von ~0.03. XFP und CFR haben in dieser Umsetzung eine ähnlich gute Qualität erreicht.

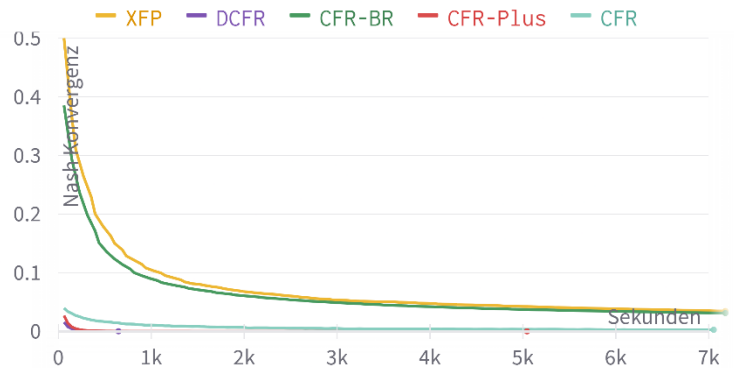


Abbildung 4.2 Die Algorithmen XFP, DCFR, CFR-BR, CFR-Plus und CFR in einem Graphen gegenübergestellt. Dargestellt wird das Verhältnis zwischen Nash-Konvergenz und Trainingszeit in Sekunden. Diese Graphik zeigt nur einen Ausschnitt und nicht die komplette Trainingszeit von jedem Algorithmus.

Auch in Abbildung 4.2 ist wieder gut zu erkennen, dass die übrigen Algorithmen durch die Skalierung ineinander laufen. In Abbildung 4.3 sind DCFR, CFR-Plus und CFR auf einer kleineren Skala abgebildet. Dadurch lässt sich die Qualität der drei Algorithmen sehr gut erkennen. CFR-Plus und DCFR konnten sogar in einer verhältnismäßig sehr kurzen Trainingsphase eine Nash-Convergenz erreichen, die sehr nahe Null ist. Das bedeuted, dass CFR-Plus als auch DCFR sehr gut geeignet sind für den Einsatz in einem komplexen Kartenspiel mit imperfekten Informationen, da sie mit verhältnismäßig wenig Rechenzeit Strategien finden können, die einem perfekten Strategiegleichgewicht sehr nahe sind. Diese Beobachtung wird unter anderem dadurch unterstrichen, dass CFR-Plus in der Poker-KI DeepStack [29, 61] eingesetzt wurde. Diese Poker-KI konnte in der kleineren Poker Variante Heads-up Limit Hold'em, eine statistisch signifikante Anzahl an Spielen gegen Expertenspieler gewinnnen [61]. DCFR ist der Algorithmus, der in dieser Umsetzung die beste Qualtität erreichen konnte, da er bereits nach zehn Minuten eine stabile Nash-Konvergenz erreicht hatte. DCFR ist auch der Algorithmus, der in der Poker-KI Pluribus [39]

angewandt wurde. Diese KI konnte als erste die Weltmeister in No-Limit Texas Hold'em Poker (die klassische Poker Variante) schlagen.

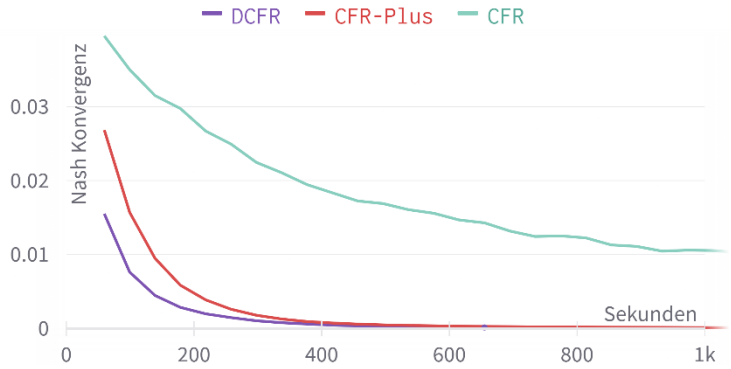


Abbildung 4.3 Die Algorithmen DCFR, CFR-Plus und CFR in einem Graphen gegenübergestellt. Dargestellt wird das Verhältnis zwischen Nash-Konvergenz und Trainingszeit in Sekunden. Diese Graphik zeigt nur einen Ausschnitt und nicht die komplette Trainingszeit von jedem Algorithmus.

5 Diskussion

Wie in der Auswertung der Ergebnisse (Abschnitt 4.4) gezeigt wurde, sind die Algorithmen CFR, DCFR und CFR-Plus für die Anwendung in einem Kartenspiel mit imperfekten Informationen geeignet. Diese Erkenntnis hat jedoch auch Einschränkungen, denn es wurde nur für Leduc-Poker umgsetzt, einer kleineren Variante des Kartenspiels Texas Hold'em Poker. Es ist bekannt, dass die CFR-Algorithmen sehr gut in Poker-Spielen anwendbar sind, doch das ist nicht zwangsläufig auf alle Kartenspiele mit imperfekten Informationen übertragbar. Kartenspiele lassen sich anhand der Spielregeln grob in Kategorien einteilen. Poker (und bspw. Bridge) gehört zur Kategorie der Biet-Kartenspiele. Es gibt jedoch auch Ablegespiele (bspw. Uno, Mau-Mau), Stichspiele (bspw. Skat, Hearts) und Solitär-Kartenspiele (Solitär, Birds of a Feather). Für die Entwicklung von KIs in diesen anderen Spielkategorien sind wiederum auch andere Lösungswege erfolgreich gewesen (siehe [30, 84, 85]). Außerdem ist die Wahl des Algorithmus davon abhängig, wie viele Spieler agieren. In der Umsetzung wurden bspw. Q-Learning und Deep Q-Learning als nicht anwendbar eingstuft, da sie nicht auf Umgebungen mit mehreren Agenten ausgelegt sind. In dem Kartenspiel Black-Jack ist Q-Learning jedoch gut anwendbar, da das Kartenspiel nur einen Spieler vorsieht. Zusammenfassend lässt sich sagen, dass in der Umsetzung unter anderem eine Schwäche darin liegt, dass es nur auf Leduc Poker und wahrscheinlich nur auf die Kategorie der Bietkartenspiele anwendbar ist, da der Fokus der Arbeit begrenzt werden musste. An dieser Stelle könnte eine fortführende Arbeit anknüpfen und einen Überblick über maschinelles Lernen in Kartenspielen anderer Spielkategorien geben.

Eine weitere Einschränkung der Umsetzung ist, dass die Ergebnisse nicht zwangsläufig repräsentabel sind. Die Policy-Gradienten-Verfahren, NFSP und Deep-CFR sind Algorithmen, die durchaus für die Anwendung in Spielen mit imperfekten Informationen geeignet sind, auch wenn sie in der Umsetzung keine herausragenden Ergebnisse erzielen konnten. Das kann zum einen an der begrenzten Rechenleistung liegen, die für die Umsetzung zur Verfügung stand. Es war eine Anforderung an die Umsetzung, dass Algorithmen effizient auf einem handelsüblichen Computer durchführbar, trainierbar sind, doch genau das könnte das Ergebnis verzerrt haben. Da das Trainieren der NN sehr rechen-, speicher- und zeitintensiv ist, kann es durchaus sein, dass diese Algorithmen eine bessere Nash-Konvergenz erreicht hätten, wenn sie länger und mit mehr Rechenleistung trainiert worden wären. Ebenso könnten die vielen Variablen der NN, wie Lernrate, die Zahl der Neuronen und versteckten Schichten schrittweise angepasst werden, bis ein besseres Ergebnis erzielt wird. Dazu könnte bspw. der Service Weights & Biases [81] eingesetzt werden: Mit Sweeps lässt sich das Anpassen der Hyperparameter eines NNs automatisieren und überwachen. So lässt sich eventuell effizient eine optimale Kombination ermitteln und die qualitativen Ergebnisse der Policy-Gradient-Verfahren, NFSP und Deep-CFR verbessern.

6 Fazit

Die Problemstellung der Arbeit lautete: Wie lässt sich maschinelles Lernen in einem Kartenspiel mit imperfekten Informationen einsetzen? Dazu wurde diese Frage in Teilgebiete unterteilt, damit sich so schrittweise einer Lösung angenähert werden kann. Zunächst wurde in Kapitel 2 die Spieltheorie erläutert und wie sie angewandt werden kann, um Spiele formal zu lösen. Dazu wurden zunächst Grundbegriffe der Spieltheorie erläutert und gezeigt mit welchen Methoden die Hürden von komplexen dynamischen Spielen mit imperfekten Informationen lösbar sind. Anschließend wurde ein kleiner Überblick über vielversprechende spieltheoretische Lösungsansätze gegeben, die auch in komplexen Kartenspiel-KIs angewandt wurden. Kapitel 3 widmete sich den Grundlagen des maschinellen Lernens. Nach einer Einführung in die Grundbegriffe und die Lernverfahren wurde das RL vertieft erläutert. RL wurde bereits mehrfach erfolgreich für Spiele mit imperfekten Informationen angewandt. Anschließend wurden Methoden des RL vorgestellt, die besonders für die Anwendung in Spiele-KIs geeignet sind. Kapitel 4 widmet sich der Lösung der Forschungsfrage. Mit dem Python-Framework OpenSpiel werden für das Kartenspiel Leduc-Poker mehrere Algorithmen aus dem Bereich der Spieltheorie des RLs angewandt, unter anderem Lösungsansätze, welche spieltheoretische Algorithmen mit RL-Lernmethoden kombinieren. In der Umsetzung wurden diese Algorithmen anhand der Nash-Konvergenz miteinander verglichen. Mittels der Nash-Konvergenz lässt sich messen, wie nah die trainierten Modelle einem spieltheoretischen Strategiegleichgewicht kommen. Das Ergebnis zeigte, dass von den neun angewandten Algorithmen DCFR (Discounted Counterfactual Regret Minimization) und Counterfactual Regret Minimization Plus (CFR-Plus) die beste Qualität erreichten. Im Kapitel 5 wurden die Ergebnisse der Umsetzung kritisch bewertet.

Literaturverzeichnis

[1] W. Ertel, *Grundkurs Künstliche Intelligenz: Eine praxisorientierte Einführung,* 5. Aufl. (Lehrbuch). Wiesbaden, Heidelberg: Springer Vieweg, 2021.

[2] R. S. Sutton und A. Barto, *Reinforcement learning: An introduction* (Adaptive computation and machine learning). Cambridge, Massachusetts, London, England: The MIT Press, 2020.

[3] D. Sonnet, *Neuronale Netze kompakt: Vom Perceptron zum Deep Learning* (IT kompakt). Wiesbaden, Heidelberg: Springer Vieweg, 2022.

[4] T. H. Cormen, C. E. Leiserson, R. Rivest, C. Stein und P. Molitor, *Algorithmen - Eine Einführung*. De Gruyter, 2017. Zugriff am: 6. Juli 2023. [Online]. Verfügbar unter: https://books.google.de/books?id=O1pSDgAAQBAJ

[5] Duden. „approximieren." https://web.archive.org/web/20230710112603/https://www.duden.de/rechtschreibung/approximieren (Zugriff am: 10. Juli 2023).

[6] R. Bellman, *Dynamic Programming*. Princeton University Press, 1957.

[7] S. Winter, *Grundzüge der Spieltheorie: Ein Lehr- und Arbeitsbuch für das (Selbst-)Studium,* 2. Aufl. Berlin, Heidelberg: Springer Berlin Heidelberg, 2019 // 2018.

[8] Y. Lecun, L. Bottou, Y. Bengio und P. Haffner, „Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, Jg. 86, Nr. 11, S. 2278–2324, 1998, doi: 10.1109/5.726791.

[9] H. Robbins, „Some aspects of the sequential design of experiments," 1952.

[10] M. J. Holler und G. Illing, *Einführung in die Spieltheorie,* 1. Aufl. (Springer eBook Collection Business and Economics). Berlin, Heidelberg, s.l.: Springer Berlin Heidelberg, 1991.

[11] Duden. „iterativ." https://web.archive.org/web/20221127125602/https://www.duden.de/rechtschreibung/iterativ (Zugriff am: 10. Juli 2023).

[12] Duden. „kumulativ." https://web.archive.org/web/20230311124258/https://www.duden.de/rechtschreibung/kumulativ (Zugriff am: 10. Juli 2023).

[13] E. Rich und K. Knight, *Artificial intelligence,* 2. Aufl. New York: McGraw-Hill, 1991.

[14] F. Southey *et al.,* „Bayes' Bluff: Opponent Modelling in Poker," *CoRR*, Jg. abs/1207.1411, 2012.

[15] Statistics How To. „Log Odds: Simple Definition & Examples, Conversions." https://web.archive.org/web/20230605110802/https://www.statisticshowto.com/log-odds/ (Zugriff am: 10. Juli 2023).

[16] C. B. Browne *et al.,* „A Survey of Monte Carlo Tree Search Methods," *IEEE Transactions on Computational Intelligence and AI in Games*, Jg. 4, Nr. 1, S. 1–43, 2012, doi: 10.1109/TCIAIG.2012.2186810.

[17] F. Rosenblatt, „The perceptron: A probabilistic model for information storage and organization in the brain.," *Psychological Review*, Jg. 6, Nr. 65, S. 386–408, 1958, doi: 10.1037/h0042519.

[18] A. Blum und Y. Monsour, „Learning, regret minimization, and equilibria," 2007.

[19] P. Auer, N. Cesa-Bianchi und P. Fischer, „Finite-time Analysis of the Multiarmed Bandit Problem," *Machine Learning*, Jg. 47, Nr. 2, S. 235–256, 2002, doi: 10.1023/A:1013689704352.

[20] International-Skat-Players-Association e.V. (ISPA-WORLD). „Internationale Skatordnung: Skatwettspielordnung." https://www.ispaworld.info/images/phocadownload/ISPA-World/ISkO-2018-3.3.pdf (Zugriff am: 20. Juni 2023).

[21] Duden. „terminieren." https://web.archive.org/web/20220724114743/https://www.duden.de/rechtschreibung/terminieren (Zugriff am: 10. Juli 2023).

[22] A. M. Turing, „Computing machinery and intelligence," *Mind*, Jg. LIX, Nr. 236, S. 433–460, 1950, doi: 10.1093/mind/LIX.236.433.

[23] S. Mihm. „ChatGPT Sounds Exactly Like Us. How Is That a Good Thing?" https://web.archive.org/web/20230118191350/https://www.washingtonpost.com/business/chatgpt-sounds-exactly-like-us-how-is-that-a-good-thing/2023/01/18/6aedf446-9736-11ed-a173-61e055ec24ef_story.html (Zugriff am: 9. Juli 2023).

[24] T. Hsu und S. L. Myers. „Can We No Longer Believe Anything We See?" https://web.archive.org/web/20230705215643/https://www.nytimes.com/2023/04/08/business/media/ai-generated-images.html (Zugriff am: 9. Juli 2023).

[25] V. Mnih *et al.*, „Playing Atari with Deep Reinforcement Learning," *CoRR*, Jg. abs/1312.5602, 2013.

[26] D. Silver *et al.*, „Mastering the game of Go with deep neural networks and tree search," *nature*, Jg. 529, Nr. 7587, S. 484–489, 2016, doi: 10.1038/nature16961.

[27] D. Silver *et al.*, „Mastering the game of go without human knowledge," *nature*, Jg. 550, Nr. 7676, S. 354–359, 2017.

[28] C. Berner *et al.*, „Dota 2 with Large Scale Deep Reinforcement Learning," *CoRR*, Jg. abs/1912.06680, 2019.

[29] M. Moravčík *et al.*, „DeepStack: Expert-level artificial intelligence in heads-up no-limit poker," *Science*, Jg. 356, Nr. 6337, S. 508–513, 2017, doi: 10.1126/science.aam6960.

[30] L. Spinney. „Artificial intelligence beats eight world champions at bridge: Victory marks milestone for AI as bridge requires more human skills than other strategy games." https://web.archive.org/web/20230215011151/https://www.theguardian.com/technology/2022/mar/29/artificial-intelligence-beats-eight-world-champions-at-bridge (Zugriff am: 16. März 2023).

[31] G. Tesauro, „Temporal difference learning and TD-Gammon,“ *Communications of the ACM*, Jg. 38, Nr. 3, S. 58–68, 1995.

[32] W. S. McCulloch und W. Pitts, „A logical calculus of the ideas immanent in nervous activity,“ *Bull Math Biophys*, Jg. 5, 1943, doi: 10.1007/BF02478259.

[33] R. Hadsell. „Artificial intelligence, video games and the mysteries of the mind.” https://www.youtube.com/watch?v=mqma6GpM7vM

[34] J. Heinrich und D. Silver, *Deep Reinforcement Learning from Self-Play in Imperfect-Information Games*.

[35] M. Lanctot *et al.*, „OpenSpiel: A Framework for Reinforcement Learning in Games,“ *CoRR*, Jg. abs/1908.09453, 2019.

[36] J. Heinrich, M. Lanctot und D. Silver, „Fictitious Self-Play in Extensive-Form Games,“ in *Proceedings of the 32nd International Conference on Machine Learning*, F. Bach und D. Blei, Hg., Bd. 37, 2015, S. 805–813.

[37] M. Campbell, A. J. Hoane und F. Hsu, „Deep Blue,“ *Artificial Intelligence*, Jg. 134, Nr. 1, S. 57–83, 2002, doi: 10.1016/S0004-3702(01)00129-1.

[38] E. Bonsdorf, K. Fabel und O. Riihimaa, *Schach und Zahl*. Düsseldorf: Rau, 1971.

[39] Carnegie Mellon University. „Carnegie Mellon and Facebook AI Beats Professionals in Six-Player Poker: "Superhuman" card shark achieves new AI milestone.” https://web.archive.org/web/20220903061919/https://www.cmu.edu/news/stories/archives/2019/july/cmu-facebook-ai-beats-poker-pros.html (Zugriff am: 14. Juli 2023).

[40] The Johns Hopkins University Applied Physics Laboratory LLC. „Welcome to the RBC Research Community.” https://web.archive.org/web/20230702021511/https://rbc.jhuapl.edu/ (Zugriff am: 14. Juli 2023).

[41] A. Bakhtin *et al.*, „Human-level play in the game of Diplomacy by combining language models with strategic reasoning,“ *Science*, Jg. 378, Nr. 6624, S. 1067–1074, 2022, doi: 10.1126/science.ade9097.

[42] T. M. Mitchell, *Machine learning* (McGraw-Hill series in computer science). New York, NY: McGraw-Hill, 1997.

[43] E. Alpaydın, *Maschinelles Lernen*, 2. Aufl. (De Gruyter Studium). Berlin, Boston: De Gruyter Oldenbourg, 2019.

[44] M. Lanctot *et al.*, „A Unified Game-Theoretic Approach to Multiagent Reinforcement Learning,“ in *Advances in Neural Information Processing Systems*, I. Guyon et al., Hg., Bd. 30, 2017. [Online]. Verfügbar unter: https://proceedings.neurips.cc/paper_files/paper/2017/file/3323fe11e9595c09af38fe67567a9394-Paper.pdf

[45] J. von Neumann, „Zur Theorie der Gesellschaftsspiele,“ *Mathematische Annalen*, Nr. 100, S. 295–320, 1928.

[46] J. von Neumann und O. Morgenstern, *Theory of games and economic behavior*, 1. Aufl. (Princeton paperbacks). Princeton: Princeton Univ. Press, 1980.

[47] J. F. Nash und J. von Neumann, *Non-cooperative games*, 1950.

[48] spielregeln.de. „Tick Tack Toe Regeln & Anleitung." https://www.spielregeln.de/tick-tack-toe.html (Zugriff am: 22. Juni 2023).

[49] H. Bottomley. „How many Tic-Tac-Toe (noughts and crosses) games are possible?" http://www.se16.info/hgb/tictactoe.htm (Zugriff am: 22. Juni 2023).

[50] brettspielnetz.de. „Vier gewinnt Spielanleitung." https://www.brettspielnetz.de/spielregeln/vier+gewinnt.php (Zugriff am: 4. Mai 2023).

[51] A. Yong und D. Yong, „An estimation method for game complexity," *arXiv preprint arXiv:1901.11161*, S. 1–4, 2019.

[52] spielregeln.de. „Mau Mau Regeln & Spielanleitung." https://www.spielregeln.de/mau-mau.html (Zugriff am: 25. Juni 2023).

[53] M. Minsky, „Steps toward Artificial Intelligence," *Proceedings of the IRE*, Jg. 49, Nr. 1, S. 8–30, 1961, doi: 10.1109/JRPROC.1961.287775.

[54] C. H. Papadimitriou, „On the complexity of the parity argument and other inefficient proofs of existence," *Journal of Computer and System Sciences*, Jg. 48, Nr. 3, S. 498–532, 1994, doi: 10.1016/S0022-0000(05)80063-7.

[55] M. Zinkevich, M. Johanson, M. Bowling und C. Piccione, „Regret Minimization in Games with Incomplete Information," in *Advances in Neural Information Processing Systems*, J. Platt, D. Koller, Y. Singer und S. Roweis, Hg., Bd. 20, 2007.

[56] P. I. Cowling, E. J. Powley und D. Whitehouse, „Information Set Monte Carlo Tree Search," *IEEE Transactions on Computational Intelligence and AI in Games*, Jg. 4, Nr. 2, S. 120–143, 2012, doi: 10.1109/TCIAIG.2012.2200894.

[57] S. Hart und A. Mas-Colell, „A simple adaptive procedure leading to correlated equilibrium," *Econometrica*, Jg. 68, Nr. 5, S. 1127–1150, 2000.

[58] T. W. Neller und M. Lanctot, „An introduction to counterfactual regret minimization," in *Proceedings of Model AI Assignments, The Fourth Symposium on Educational Advances in Artificial Intelligence (EAAI-2013)*, Bd. 11, 2013.

[59] N. Brown, A. Lerer, S. Gross und T. Sandholm, „Deep Counterfactual Regret Minimization," in *Proceedings of the 36th International Conference on Machine Learning*, K. Chaudhuri und R. Salakhutdinov, Hg., Bd. 97, 2019, S. 793–802.

[60] E. Steinberger, *Single Deep Counterfactual Regret Minimization*.

[61] M. Bowling, N. Burch, M. Johanson und O. Tammelin, „Heads-up Limit Hold'em Poker is Solved," *Commun. ACM*, Jg. 60, Nr. 11, S. 81–88, 2017, doi: 10.1145/3131284.

[62] N. Brown und T. Sandholm, „Superhuman AI for heads-up no-limit poker: Libratus beats top professionals," *Science*, Jg. 359, Nr. 6374, S. 418–424, 2018, doi: 10.1126/science.aao1733.

[63] C. Sammut und G. I. Webb, Hg. *Encyclopedia of machine learning and data mining* (Springer Reference). New York, NY: Springer, 2017.

[64] Google. „What is Machine Learning?" https://developers.google.com/machine-learning/intro-to-ml/what-is-ml?hl=en (Zugriff am: 31. Mai 2023).

[65] H. Xiao, K. Rasul und R. Vollgraf. „Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms." https://github.com/zalandoresearch/fashion-mnist (Zugriff am: 27. Juni 2023).

[66] H. Tran, *Survey of Machine Learning and Data Mining Techniques used in Multimedia System*, doi: 10.13140/RG.2.2.20395.49446/1.

[67] B. Bethke, J. P. How und A. Ozdaglar, „Approximate dynamic programming using support vector regression," in *2008 47th IEEE Conference on Decision and Control*, 2008, S. 3811–3816, doi: 10.1109/CDC.2008.4739322.

[68] M. Plaue, *Data Science: Grundlagen, Statistik und maschinelles Lernen* (Lehrbuch). Berlin, Heidelberg: Springer Spektrum, 2021.

[69] Y. LeCun, C. Cortes und C. J. Burges. „The MNIST Database: of handwritten digits." http://yann.lecun.com/exdb/mnist/ (Zugriff am: 29. Juni 2023).

[70] R. Bellman, „Dynamic Programming," *Science*, Jg. 153, Nr. 3731, S. 34–37, 1966, doi: 10.1126/science.153.3731.34.

[71] Watkins, Christopher John Cornish Hellaby, „Learning from delayed rewards," 1989.

[72] J. J. Hopfield, „Neural networks and physical systems with emergent collective computational abilities," *Proceedings of the National Academy of Sciences*, Jg. 79, Nr. 8, S. 2554–2558, 1982, doi: 10.1073/pnas.79.8.2554.

[73] T. Kohonen, „The self-organizing map," *Proceedings of the IEEE*, Jg. 78, Nr. 9, S. 1464–1480, 1990, doi: 10.1109/5.58325.

[74] N. Brown. „Depth-Limited Solving in Imperfect-Information Games." https://www.youtube.com/watch?v=S4-g3dPT2gY (Zugriff am: 19. Juli 2023).

[75] Kaggle.com. „Your Mashine Learning and Data Science Community." https://web.archive.org/web/20230708155128/https://www.kaggle.com/ (Zugriff am: 8. Juli 2023).

[76] P. Wzorek und T. Kryjak, „Training dataset generation for bridge game registration," *CoRR*, Jg. abs/2109.11861, 2021.

[77] D. Zha *et al.*, „RLCard: A Toolkit for Reinforcement Learning in Card Games," *CoRR*, Jg. abs/1910.04376, 2019.

[78] Microsoft. „Installieren von Linux unter Windows mit WSL.” https://web.archive.org/web/20230613210835/https://learn.microsoft.com/de-de/windows/wsl/install (Zugriff am: 20. Juli 2023).

[79] NVIDIA Corporation. „CUDA on WSL User Guide.” https://web.archive.org/web/20230706065553/https://docs.nvidia.com/cuda/wsl-user-guide/index.html (Zugriff am: 20. Juli 2023).

[80] DeepMind Technologies Ltd. „OpenSpiel Windows Installation using Windows Subsystem for Linux (WSL).” https://web.archive.org/web/20220117074737/https://openspiel.readthedocs.io/en/latest/windows.html#option-2-windows-installation-using-windows-subsystem-for-linux-wsl (Zugriff am: 20. Juli 2023).

[81] Weights & Biases. „Weights & Biases Homepage.” https://web.archive.org/web/20230706182159/https://wandb.ai/site

[82] C. Allen, K. Asadi, M. Roderick, A. Mohamed, G. Konidaris und M. Littman, *Mean Actor Critic*.

[83] S. Srinivasan *et al., Actor-Critic Policy Optimization in Partially Observable Multiagent Environments*.

[84] M. Wurm. „Teaching a Neural Network to Play Cards: How I trained a neural network to play a trick-taking card game without requiring human input.” https://towardsdatascience.com/teaching-a-neural-network-to-play-cards-bb6a42c09e20 (Zugriff am: 10. April 2023).

[85] B. K. Demirdöver, Ö. BAYKAL und F. Alpaslan, „Learning to play an imperfect information card game using reinforcement learning,“ *Turkish Journal of Electrical Engineering and Computer Sciences,* Jg. 30, Nr. 6, S. 2303–2318, 2022.

Abbildungsverzeichnis

Eidesstattliche Erklärung

Hiermit versichere ich, dass ich die vorliegende Arbeit bisher bei keiner anderen Prüfungsbehörde eingereicht, sie selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie Zitate kenntlich gemacht habe.

Ort, Datum	*Unterschrift*