

Bachelorarbeit

zur Erlangung des akademischen Grades
Bachelor

Technische Hochschule Wildau
Fachbereich Ingenieur- und Naturwissenschaften
Studiengang Telematik (B. Eng.)

Thema (deutsch): Maschinelles Lernen: Entwicklung einer künstlichen Intelligenz für das Spiel „Vier Gewinnt“ mit Hilfe von Reinforcement Learning

Thema (englisch): Machine Learning: Development of an Artificial Intelligence for the game „Connect Four“ with the help of Reinforcement Learning

Autor/in: Henning Steinert

Seminargruppe: T/16

Betreuer/in: Prof. Dr. rer. nat. Janett Mohnke

Zweitgutachter/in: Master of Engineering Janine Breßler

Eingereicht am: 11.08.2019

Bibliografische Beschreibung und Referat

Steinert, Henning

Maschinelles Lernen: Entwicklung einer künstlichen Intelligenz für das Spiel „Vier Gewinnt“ mit Hilfe von Reinforcement Learning

Bachelorarbeit, Technische Hochschule Wildau 2019,

???

Seiten,

???

Abbildungen,

???

Tabellen,

???

Anlagen.

Ziel

Zielstellung der Arbeit

Inhalt

Inhalt der Arbeit

Abstract in deutsch

???

Abstract in english

???

Selbstständigkeitserklärung

Ich erkläre hiermit, dass ich die vorliegende Abschlussarbeit selbstständig angefertigt und nur die angegebenen Hilfsmittel und Quellen verwendet habe.

Wildau, den 7. August 2019

.....

Henning Steinert

Inhaltsverzeichnis

Hinweise zum Lesen der Arbeit	3
1 Einleitung	4
1.1 Motivation	4
1.2 Zielsetzung	5
1.3 Abgrenzung	5
1.4 Aufbau der Arbeit	5
2 Theoretische Grundlagen	7
2.1 Künstliche Intelligenz	7
2.1.1 Neuronale Netze	13
2.1.2 Maschinelles Lernen	17
2.2 Reinforcement Learning	21
2.2.1 Markow Entscheidungsproblem	22
2.2.2 Dynamic Programming	29
2.2.3 Temporal Difference Learning	32
2.2.4 Monte Carlo Methoden	38
2.2.5 Taxonomie der Lösungsverfahren	39
2.3 4 Gewinnt	41
3 Anforderungsanalyse	42
3.1 Persona	42
3.2 Anwendungsfälle	47
3.3 Anforderungen	48
4 Evaluation von Entwicklungswerkzeugen	50
4.1 Auswahl der Programmier- /Skriptsprachen	50
4.2 Auswahl von Frameworks	51
4.3 Auswahl des Algorithmus	53
5 Konzept	55
5.1 Frontend	56

5.2 Backend	58
6 Implementierung	64
7 Auswertung	71
7.1 Beurteilung des Projekterfolgs	71
7.2 Chancen von Reinforcement Learning	71
7.3 Persönliches Fazit	71
Glossar	72
Abkürzungsverzeichnis	73
Tabellenverzeichnis	74
Abbildungsverzeichnis	75
Quellenverzeichnis	75

Hinweise zum Lesen der Arbeit

Nachfolgend werden die verwendeten Konventionen genannt, die innerhalb der Arbeit verwendet werden.

Kapitel 1

Einleitung

1.1 Motivation

Fast jeder hat schon mal Schlagwörter wie Künstliche Intelligenz, maschinelles Lernen oder neuronale Netze gehört und trotzdem wissen die Wenigsten was sich dahinter verbirgt. Viele Menschen verbinden dieses Thema mit ihrem vermeintlichen Wissen aus Filmen und Fernsehen, sodass der Eindruck entsteht, dass künstliche Intelligenz in naher Zukunft bereits alle Aufgaben des Menschen übernehmen kann. Doch was steckt wirklich hinter dem Wort Künstliche Intelligenz und was ist maschinelles Lernen? Auf diese und weitere spannende Fragen möchte der Studiengang Telematik im Rahmen der NaWiTex Schülerlabore¹ eine Antwort geben.

Dabei gibt es verschiedene Themenfelder unter dem Oberbegriff künstliche Intelligenz, welche im Rahmen der Lehrveranstaltungen erläutert werden. Ein großes Themengebiet ist hierbei das maschinelle Lernen. Dieses kann wiederum in weitere algorithmische Ansätze unterteilt werden, wobei einer dieser Ansätze Reinforcement Learning genannt wird. Bei der Methode des Reinforcement Learning soll der Algorithmus selbstständig die beste Strategie zur Lösung eines Problems finden. Um dieses Ziel zu erreichen, wird ein Belohnungssystem genutzt, welches an das Prinzip des menschlichen Lernens angelehnt ist. Diese Art des maschinellen Lernens gewinnt auf Grund von Anwendungen wie AlphaGoZero in den letzten Jahren immer mehr an Bedeutung und Popularität. [1] Diese Künstliche Intelligenz kann bereits problemlos die besten Go-Spieler der Welt schlagen und das lediglich mit einer Trainingszeit von wenigen Tagen. Solch ein beeindruckendes Beispiel zeigt das Potential von Reinforcement Learning für den Bereich künstliche Intelligenz. Trotzdem steckt diese Methode des maschinellen Lernens noch in den Kinderschuhen. Deshalb ist es umso wichtiger, die grundsätzliche Funktionsweise von Reinforcement Learning zu verstehen, um zukünftig komplexe Anwendungen in diesem Bereich verstehen oder selbst entwickeln zu können. Um den Lernprozess dabei möglichst praktisch zu gestalten, soll die Funktionsweise von

¹<https://icampus.th-wildau.de/icampus/home/de/nawitex-sch%C3%BClerlabore>

Reinforcement Learning am Spiel „4 Gewinnt“ nachvollzogen werden.

1.2 Zielsetzung

Ziel der Arbeit ist die Konzeption und Entwicklung des Spiels „4 Gewinnt“ zur Veranschaulichung der Funktionsweise von Reinforcement Learning für die NaWiTex Schülerlabore. Dabei soll die optimale Strategie für das Spiel vom Programm selbst erlernt werden. Um dieses Ziel zu erreichen, werden gängige Methoden aus dem Bereich Reinforcement Learning implementiert. Außerdem soll zur Interaktion mit dem Programm, eine grafische Benutzeroberfläche erstellt werden. Damit soll es möglich sein, gegen das Programm zu spielen oder dem Programm beim Spielen zuzusehen, sodass die Teilnehmer der NaWiTex Schülerlabore das Verhalten der Künstlichen Intelligenz beobachten und bewerten können. Darüber hinaus soll es möglich sein, den Schwierigkeitsgrad des Programms zu verändern und weitere Informationen über den Lernprozess abzurufen.

1.3 Abgrenzung

Das im Rahmen dieser Arbeit entwickelte Programm, soll ausdrücklich keine explizite Spielstrategie implementiert bekommen. Das bedeutet, die optimale Vorgehensweise, muss von dem Programm selbst entwickelt werden und wird nicht von außen zugeführt. Dies ist der Wesenskern von Reinforcement Learning und unterscheidet es von anderen Programmiermethodiken.

Des Weiteren besteht im Rahmen dieser Arbeit nicht der Anspruch, eine unschlagbare künstliche Intelligenz zu entwickeln. Stattdessen sollen die theoretischen Ansätze sowie die Grundidee des Reinforcement Learning am Beispiel einer spielerischen Anwendung verdeutlicht werden.

1.4 Aufbau der Arbeit

Die Arbeit ist in die Teilebereiche Grundlagen, Anforderungsanalyse, Konzeption, Implementierung und Auswertung aufgeteilt.

Dabei werden unter dem Punkt Grundlagen, zunächst die wichtigsten Begriffe aus dem Bereich künstliche Intelligenz erklärt. Dazu zählen vor allem auch die Bereiche neuronale Netze und maschinelles Lernen. Anschließend wird die Grundidee des Reinforcement Learning dargestellt und es werden die gebräuchlichsten Algorithmen aus diesem Bereich erläutert.

Danach kann mit Hilfe der Anforderungsanalyse und den zuvor gewonnenen theoretischen Grundlagen, das Konzept für die Entwicklung eines selbst lernenden Programms nachvollzogen werden. Zu diesem Konzept gehört außerdem die Planung der grafischen Benutzeroberfläche sowie der verwendeten Softwarebausteine. Damit ist beispielsweise die Auswahl pas-

sender Frameworks, Bibliotheken oder Programmiersprachen gemeint. Im Anschluss werden unter dem Punkt Implementierung, der Entwicklungsverlauf, auftretende Probleme sowie die getätigten Versuchsreihen dokumentiert. Die Arbeit wird abgeschlossen durch die Auswertung der Ergebnisse. Hierzu wird der Erfolg des Projekts beurteilt und eine Betrachtung des Potentials von Reinforcement Learning in der Zukunft durchgeführt.

Kapitel 2

Theoretische Grundlagen

In diesem Kapitel werden als erstes die wichtigsten Begriffe zum Thema Künstliche Intelligenz sowie zum Schluss die Grundlagen zum Spiel „4 Gewinnt“ erläutert. Dabei beschränkt sich die Erläuterung dieser Themen auf die Grundlagen, weil der Fokus im Rahmen der Arbeit auf dem Reinforcement Learning liegt. Deshalb ist dieses auch als extra Themenpunkt aufgeführt, obwohl es ein Teilbereich des maschinellen Lernens darstellt.

Zum Ende des theoretischen Teils, soll die notwendige Wissensbasis gelegt sein, um anschließend eine entsprechend intelligente Anwendung konstruieren und implementieren zu können.

2.1 Künstliche Intelligenz

Der Begriff **künstliche Intelligenz (KI)** ist auf Grund aktueller Entwicklungen weit verbreitet und wird viel diskutiert. Allerdings existieren die notwendigen Grundlagen dafür bereits seit vielen Jahrzehnten. Doch wieso ist dieses Thema ausgerechnet jetzt so populär? Verantwortlich hierfür sind verschiedene Faktoren, zum einen entwickelt sich die Rechenleistung stetig weiter, sodass aufwändige Lernprozesse und Berechnungen immer weniger Zeit in Anspruch nehmen. Zum anderen hat sich auch die entsprechende Software weiterentwickelt. Neben fortschreitender Parallelisierung ist hier in erster Linie auch die Weiterentwicklung neuronaler Netze zum tiefgehenden Lernen zu nennen. Außerdem werden immer größere Datenmengen erfasst und gespeichert. Mit Hilfe dieser Daten können intelligente Programme trainiert und somit weiter verbessert werden.

Als Folge dieser Entwicklungen eröffnen sich neue Möglichkeiten zum Einsatz Künstlicher Intelligenz in verschiedenen Anwendungsgebieten. Aktuell geläufige Beispiele lassen sich unter anderem in den Bereichen Spracherkennung und Bildverarbeitung finden.

Die Spracherkennung wird mit Hilfe von Künstlicher Intelligenz optimiert und in Form von Anwendungen wie dem Sprachassistenten Alexa¹ von vielen Menschen bereits genutzt.

¹<https://www.amazon.de/b?ie=UTF8&node=12775495031>

Ebenso verhält es sich mit der Bildverarbeitung, welche beispielsweise zur Qualitätskontrolle innerhalb einer Fließbandproduktionsstraße eingesetzt werden kann.

Allerdings gibt es auch Anwendungsmöglichkeiten, die zu Kontroversen führen können. Hierbei können beispielsweise kommerzielle Systeme wie die **Dunkelverarbeitung** innerhalb eines Versicherungsunternehmens genannt werden. Als Dunkelverarbeitung werden in der Versicherungswirtschaft alle Prozesse bezeichnet, welche ohne Kontrolle durch die Mitarbeiter voll automatisiert durchgeführt werden. Der Anteil dieser Geschäftsprozesse könnte mit Hilfe von Künstlicher Intelligenz drastisch erhöht werden und somit zur Entlassung von vielen Mitarbeitern führen. Laut Zahlen des Gesamtverbandes deutscher Versicherer² gibt es in Deutschland knapp 500.000 Beschäftigte in der Versicherungswirtschaft. Somit haben Lösungen aus dem Bereich **KI**, die nur einen kleinen Teil der notwendigen Arbeiten innerhalb eines Versicherungsunternehmens substituieren, das Potential, viele Mitarbeiter arbeitslos zu machen. Andererseits kann dies für die entsprechenden Unternehmen eine große Produktivitätssteigerung und vor allem gesenkte Personalkosten bedeuten. Dadurch entstehen die zwei Fronten zwischen Unternehmensführung und Arbeitnehmern. Anhand dieses Beispiels wird deutlich, warum dieses Thema viele Menschen polarisiert und welche zukünftigen Herausforderungen durch Künstliche Intelligenz entstehen können.

Im Rahmen dieser Diskussion werden jedoch oft die Fähigkeiten von Künstlicher Intelligenz überschätzt und unterschiedliche Begrifflichkeiten miteinander vermischt. Zunächst sind Neuronale Netze und das Maschinelle Lernen Teilbereiche Künstlicher Intelligenz und werden als Werkzeuge zur Umsetzung dieser genutzt (siehe Abbildung 2.1). Demnach kann man die Begriffe nicht gleich stellen. Eine ausführliche Beschreibung davon folgt in den nächsten Abschnitten der Arbeit.

²<https://www.gdv.de/de/zahlen-und-fakten/versicherungsbereiche/erwerbstaeigte-24122#ErwerbstaeigteBDL>

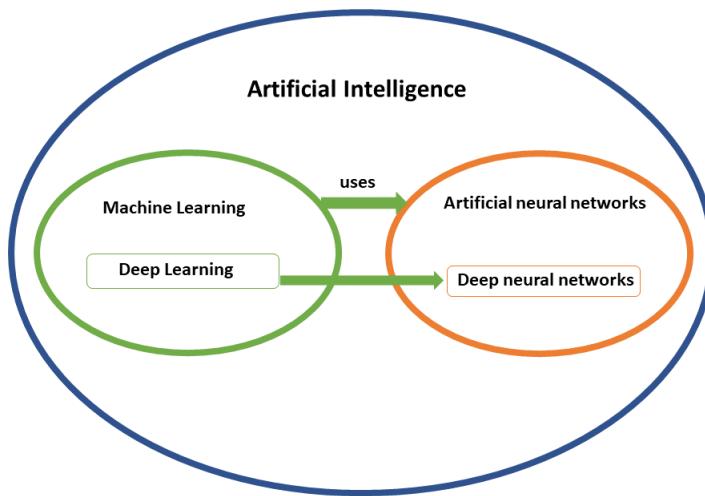


Abbildung 2.1: Darstellung Teilbereiche von **KI**

Des Weiteren sollte sich bei einer Diskussion über Künstliche Intelligenz darauf geeinigt werden, über welche Form gesprochen wird, denn es kann grob zwischen schwacher und starker **KI** unterschieden werden. Dabei bezeichnet die schwache **KI** die spezialisierte Intelligenz zur Lösung von spezifischen Problemen wie beispielsweise zur Analyse von Röntgenbildern. Die starke **KI** bezeichnet hingegen eine allgemeine Intelligenz ähnlich der menschlichen. Im Grunde sind alle aktuell existierenden Systeme der schwachen **KI** zuzuordnen. Starke Künstliche Intelligenz ist zum jetzigen Zeitpunkt höchstens Gegenstand der Philosophie, auf Grund von ethischen Fragestellungen. Dementsprechend fokussiert sich diese Arbeit auch auf die Entwicklung einer schwachen, also aufgabenorientierten Künstlichen Intelligenz. Doch selbst nach dieser Eingrenzung ist es nicht möglich, eine exakte Definition für Künstliche Intelligenz zu formulieren, weil dieses Gebiet durch viele weitere naturwissenschaftliche Felder beeinflusst wird. Beispiele dafür sind die Neurowissenschaften, die Medizin oder Linguistik. Deshalb wird sich in diese Arbeit auf eine sehr praxisorientierte Definition aus Sicht eines Softwareentwicklers beschränkt:

“Künstliche Intelligenz (Engl. Artificial Intelligence) ist ein Zweig der Informatik, der sich mit der Entwicklung von Computersystemen befasst, die selbstständig Funktionen ausführen können, für die normalerweise menschliche Intelligenz erforderlich ist, beispielsweise logisches Denken, Problemlösung, Lernen aus Erfahrung oder Spracherkennung.” [2]

Für den Entwickler einer **KI** geht es also darum, diese in die Lage zu versetzen, selbstständig Probleme zu lösen, ohne dass die Art und Weise vom Menschen festgelegt wird. In der Vergangenheit waren die Ergebnisse dieser Programmiermethodik allerdings unzureichend. Erst durch die oben genannten Entwicklungen in Bezug auf die Rechenleistung und

die Menge sowie Qualität der Trainingsdaten hat sich dieser Umstand verändert, sodass die Resultate von Künstlicher Intelligenz bisherige menschliche Ansätze übertreffen.

Um den Grund für die potentiell zukünftige Überlegenheit von **KI** genauer zu beleuchten, betrachten wir das bereits aufgeworfene Beispiel der Analyse von Röntgenbildern. Die Aufgabe besteht in diesem Anwendungsbeispiel darin, mögliche Krankheitsbilder des Patienten zu entdecken. Ein Vorteil dabei ist die große Menge kategorisierter Testdaten zum Trainieren der **KI**. Mit Hilfe dieser Daten kann eine Künstliche Intelligenz dazu trainiert werden, entsprechende Krankheitsbilder auf den Röntgenaufnahmen zu erkennen. Dadurch wäre es möglich die Radiologen zu entlasten und ihnen die Möglichkeit zu geben andere Aufgaben wahrzunehmen. In Anbetracht der oft hohen Wartezeiten für diesbezügliche Befunde wäre das ein hoher Mehrwert für den Bereich der Radiologie.

Ein Beispiel für die Umsetzung eines solchen Systems bietet die britische Forschungsgruppe um Giovanni Montana vom King's College in London³. Diese haben eine Künstliche Intelligenz entwickelt, welche Röntgenaufnahmen vom Brustraum erwachsener Patienten auswerten soll. Grund für die Ausarbeitung dieser Anwendung waren die hohen Wartezeiten auf entsprechende radiologische Befunde in Großbritannien. Das System der britischen Wissenschaftlichen sollte deshalb eine Kategorisierung der Dringlichkeit der Befunde vornehmen, sodass kritische Röntgenaufnahmen frühzeitig durch einen Arzt kontrolliert werden können. Dabei sollte eine Klassifikation in die Kategorien „normal“, „nicht dringend“, „dringend“ und „kritisch“ vorgenommen werden. Allerdings zeigte das System im Test mit 16.000 Aufnahmen eine noch nicht optimale Genauigkeit, sodass teilweise falsche Zuweisungen durchgeführt wurden. Nachstehend werden die Resultate des Tests genannt:

*„Normal chest radiographs were detected by our AI system with a sensitivity of 71%, specificity of 95%, **PPV** of 73%, and **NPV** of 94%. The average reporting delay was reduced from 11.2 to 2.7 days for critical imaging findings ($P < .001$) and from 7.6 to 4.1 days for urgent imaging findings ($P < .001$) in the simulation compared with historical data.“ [3]*

Trotz dieser Ungenauigkeiten bezeichnet die Forschungsgruppe das Ergebnis zusammenfassend als erfolgreich:

„Automated real-time triaging of adult chest radiographs with use of an artificial intelligence system is feasible, with clinically acceptable performance.“ [3]

Anhand dieses Anwendungsbeispiels kann zum einen das Potential zur Produktivitätssteigerung erkannt werden und zum anderen wird deutlich, dass viele solcher Anwendungen noch

³<https://www.n-tv.de/wissen/Kuenstliche-Intelligenz-hilft-bei-Roentgenbild-Analyse-article20821927.html>

in der Entwicklungsphase stecken. Außerdem zeigt sich die bereits erwähnte Überschneidung von verschiedenen Naturwissenschaften mit der Informatik. Mit weiterer Optimierung dieses Verfahrens könnte es zukünftig möglich sein, dass sich die Radiologen lediglich die auffälligen Aufnahmen anschauen müssen und sich somit mehr Zeit für komplizierte Fälle nehmen können. Des Weiteren ist es denkbar, dass die **KI** dazu in die Lage versetzt werden kann, Krankheitsbilder zu erkennen, welche sonst unentdeckt geblieben wären. Unabhängig von den potentiellen Fähigkeiten der Anwendung hat es bereits im oben genannten Testlauf zu einer deutlichen Verkürzung der Wartezeiten für einen Befund geführt.

Es gibt neben den genannten Beispielen eine Vielzahl von weiteren Problemen, welche bisher menschliche Intelligenz erfordern und zukünftig mit Hilfe von Künstlicher Intelligenz gelöst werden können. Im Folgenden sollen deshalb weitere Nutzungsszenarien zur Hervorhebung der Bedeutung von Künstlicher Intelligenz und den weitreichenden Anwendungsmöglichkeiten genannt werden. Anschließend wird unter den Punkten neuronale Netze und maschinelles Lernen die Funktionsweise von Künstlicher Intelligenz näher beleuchtet.

Bildverarbeitung

- Gesichts-, Mimik-, Gestenerkennung
- Qualitätssicherung bei Werkstücken in der Industrie (Produktion)
- Hautkrebserkennung
- Auswertung mikroskopischer Aufnahmen von Bakterienproben

Spracherkennung

- Fremdsprachenübersetzer
- Telefonbots bei Call Centern
- Chatbots für Webportale

Weitere Anwendungsgebiete

- Wetterprognose
- Marktprognose für Aktienmarkt
- Predictive Policing: **KI** gestützte Polizeiarbeit
- autonomes Fahren
- Analyse von großen Datenmengen für statistische Prognosen z.B. Kaufverhalten

- Robotik
- Optimierung von Reiserouten

2.1.1 Neuronale Netze

Künstliche neuronale Netze sind nicht das einzige Werkzeug zur Umsetzung vom maschinellen Lernen, aber das aktuell populärste. Hierbei geht es generell darum, eine mathematische Zielfunktion mit Hilfe eines neuronalen Netzes zu approximieren. [4] Diese Funktion wird durch eine Vielzahl einfacher mathematischer Operationen dargestellt und kann Probleme lösen, welche für den Menschen schwer zu bewältigen sind. Im Endeffekt soll also im simpelsten Fall eine Funktion $f(x) = y$ approximiert werden, wobei x die Eingabedaten sind, y das gewünschte Ergebnis und f die Funktion, welche in Form eines neuronalen Netzes abgebildet wird.

Die Inspiration für künstliche neuronale Netze entspringt aus dem biologischen Vorbild des Nervensystems. Der Grundbaustein des Nervensystems ist dabei ein Neuron oder auch Nervenzelle genannt. Allerdings ist ein Neuron noch nicht intelligent, erst die multiple Verknüpfung von Neuronen führt zur menschlichen Intelligenz. Deshalb hat die Informatik eine abstrahierte Version einer biologischen Nervenzelle entwickelt (künstliches Neuron) und verknüpft eine Vielzahl davon miteinander, um eine Künstliche Intelligenz zu erzeugen. Die nachfolgende Abbildung 2.2 zeigt eine schematische Darstellung eines solchen künstlichen Neurons, dem Grundbaustein eines künstlichen neuronalen Netzes.

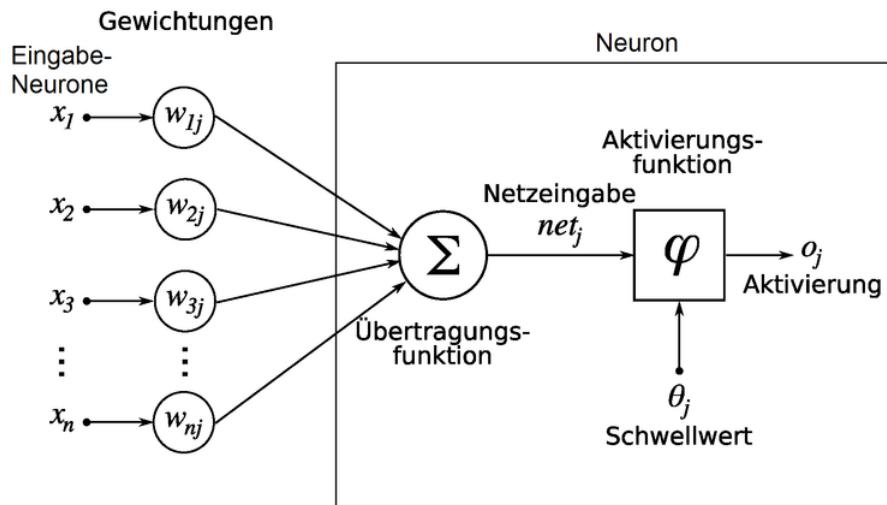


Abbildung 2.2: Struktur eines Neurons

Das Neuron bekommt wie in der Abbildung zu sehen, eine beliebig festgelegte Anzahl an Eingabesignalen. Diese können allerdings noch eine Gewichtung erhalten, sodass verschiedene Eingangssignale unterschiedliche prozentuale Bewertungen erhalten. Dieser Vorgang wird in der Abbildung als Übertragungsfunktion dargestellt. Damit ist die Multiplikation des Signals mit dem Gewicht und das anschließende Aufsummieren gemeint. Dies kann als Formel mit der Übertragungsfunktion σ , den Eingabesignalen x_i und den Gewichtungen w_i

wie folgt beschrieben werden:

$$\sigma = \sum_{i=0}^n x_i * w_i \quad (2.1)$$

Die Gewichte sind wichtig um Veränderungen am Netz durchführen zu können, sie sind die Stellschrauben und bilden schlussendlich die sich entwickelnde Intelligenz ab. Zusätzlich zu den Gewichten gibt es noch den sogenannten Bias, dieser kann als extra Input dazu geschaltet werden. Mit der Hilfe dieses zusätzlichen Parameters kann die Aktivierungsfunktion, welche nachfolgend erklärt wird, verschoben werden. Im einfachen Fall einer linearen Funktion $f(x) = mx + n$ mit dem Parameter m für die Gewichte, wäre der Bias somit das n und ermöglicht die vertikale Verschiebung der Funktion im zweidimensional Raum.⁴

Das Ergebnis der Übertragungsfunktion wird als Netzwert oder Netzeingabe bezeichnet. Mit der Aktivierungsfunktion wird geprüft ob dieser entstandene Wert einen bestimmten Schwellwert übersteigt. Falls dies passiert löst das Neuron die Aktivierung aus und entsendet ein entsprechendes Ausgangssignal. Im einfachsten Fall sieht die Aktivierungsfunktion wie folgt aus:

$$\phi(\sigma) = \begin{cases} 1 & \text{wenn } \sigma \geq 0 \\ 0 & \text{sonst} \end{cases} \quad (2.2)$$

In der Praxis finden allerdings viele unterschiedliche, vor allem nicht lineare Aktivierungsfunktionen ihre Anwendung. Ein Beispiel dafür ist die Sigmoid Funktion (siehe Abbildung 2.3), diese ist vollständig differenzierbar und ermöglicht dadurch wichtige Lernmechanismen, welche im weiteren Verlauf des Abschnitts erläutert werden.

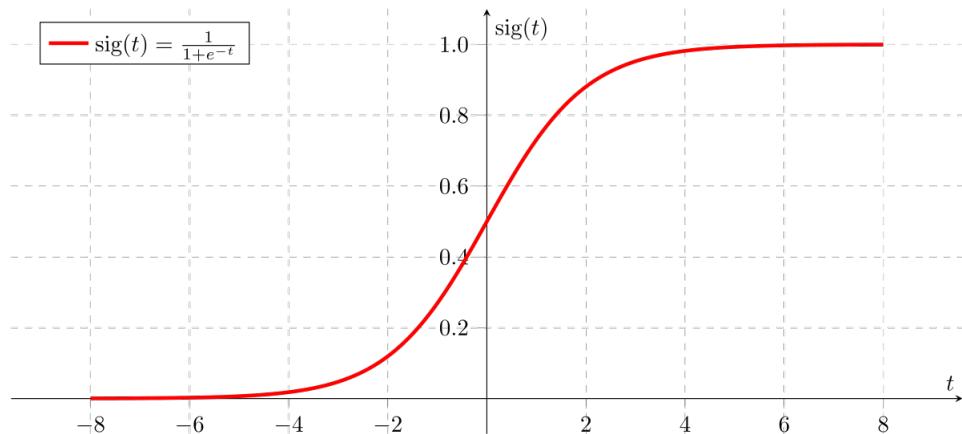


Abbildung 2.3: Sigmoid Funktion

Nun haben wir einen Überblick über die Neuronen gewonnen, allerdings kann wie bereits

⁴<https://www.geeksforgeeks.org/effect-of-bias-in-neural-network/>

erwähnt mit einem Neuron noch keine Intelligenz abgebildet werden. Deshalb werden viele Neuronen miteinander verknüpft, um ein künstliches neuronales Netz zu bilden. Dabei besteht ein solches Netz immer aus einer Eingabeschicht, einer beliebigen Anzahl versteckter Schichten sowie einer Ausgabeschicht. Jede Schicht enthält dabei eine frei wählbare Anzahl an Neuronen. Sobald mehrere versteckte Schichten genutzt werden, bezeichnet man das Netz als tiefes neuronales Netz. Jedoch existiert keine exakte Definition für die dafür notwendige Anzahl der Schichten. Zusätzlich existiert kein Maximum für die Anzahl der Schichten. Trotzdem spricht man im allgemeinen Konsens davon, dass ab zwei versteckten Schichten von einem tiefen Netz gesprochen wird und bei mehr als zehn versteckten Schichten von einem extrem tiefen neuronalen Netz gesprochen wird. [5]

Eine modellhafte Darstellung der beiden Formen findet sich in der unten stehenden Abbildung 2.4.

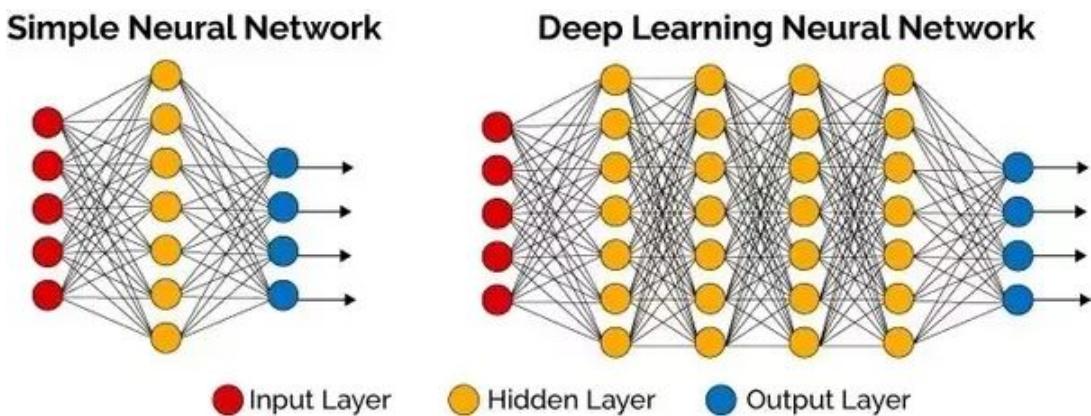


Abbildung 2.4: Tiefes Neuronales Netz [6]

Mit Hilfe eines solchen tiefen neuronalen Netzes wird dann auch das bereits erwähnte tiefgehende Lernen realisiert. Dieses ist ebenso wie die Entwicklung der Rechenleistung ein Haupttreiber für die aktuelle Verbreitung von Künstlicher Intelligenz. Neben einer Vielzahl von Schichten und Neuronen gibt es außerdem noch verschiedene Netzarten, welche passend zu einem Problem ausgewählt werden sollten. Eine sehr umfangreiche Zusammenstellung verschiedener Arten von künstlichen neuronalen Netzen bietet das Asimov Institut. [7]

Nun kennen wir die grundlegende Struktur von Neuronen und einem neuronalen Netz, wir wissen wie eingegebenen Daten verarbeitet werden und eine Ausgabe entsteht. Doch wie lernt das Netz nun dazu?

Dafür werden die oben bereits angesprochenen Lernmechanismen genutzt. Im Grunde wird im Rahmen dieser Verfahren ein Fehler im Ergebnis entdeckt und dieser anschließend zurückgerechnet auf ihre Verursacher. Es werden also die Neuronen beziehungsweise Gewichtungen so verändert, dass der vorhandene Fehler minimiert wird. Das verbreitetste Beispiele für ein

solches Verfahren ist Backpropagation⁵. Durch die Minimierung der Fehlerquellen, nähert sich das Ergebnis mit zunehmender Lerndauer der Zielfunktion und kann im Anschluss zur Verarbeitung von realen Daten genutzt werden.

Da das Thema neuronale Netze sehr umfangreich ist, muss an dieser Stelle auf eine detailliertere Beschreibung verzichtet werden. Wir werden allerdings im Rahmen des praktischen Teils noch eine exemplarische Anwendung eines neuronalen Netzes durchführen.

⁵<https://de.wikipedia.org/wiki/Backpropagation>

2.1.2 Maschinelles Lernen

Maschinelles Lernen ist ein Teilbereich von Künstlicher Intelligenz und der Oberbegriff für verschiedene algorithmische Ansätze Wissen aus Erfahrung zu generieren. Hierzu wird auf Grundlagen von Testdaten eine spezifische Zielfunktion durch das maschinelle Lernen generiert, welche dazu in der Lage ist bestimmte Problemstellungen zu lösen.

Zur Umsetzung des maschinellen Lernens werden häufig die bereits erläuterten neuronalen Netze genutzt. Allerdings gibt es auch eine Vielzahl anderer Vorgehensweisen um maschinelles Lernen umzusetzen. Die untenstehende Abbildung zeigt eine Übersicht der drei gängigsten Formen des maschinellen Lernens mit ihren jeweiligen speziellen Funktionen sowie Anwendungsmöglichkeiten, welche nachfolgend einzeln erläutert werden.

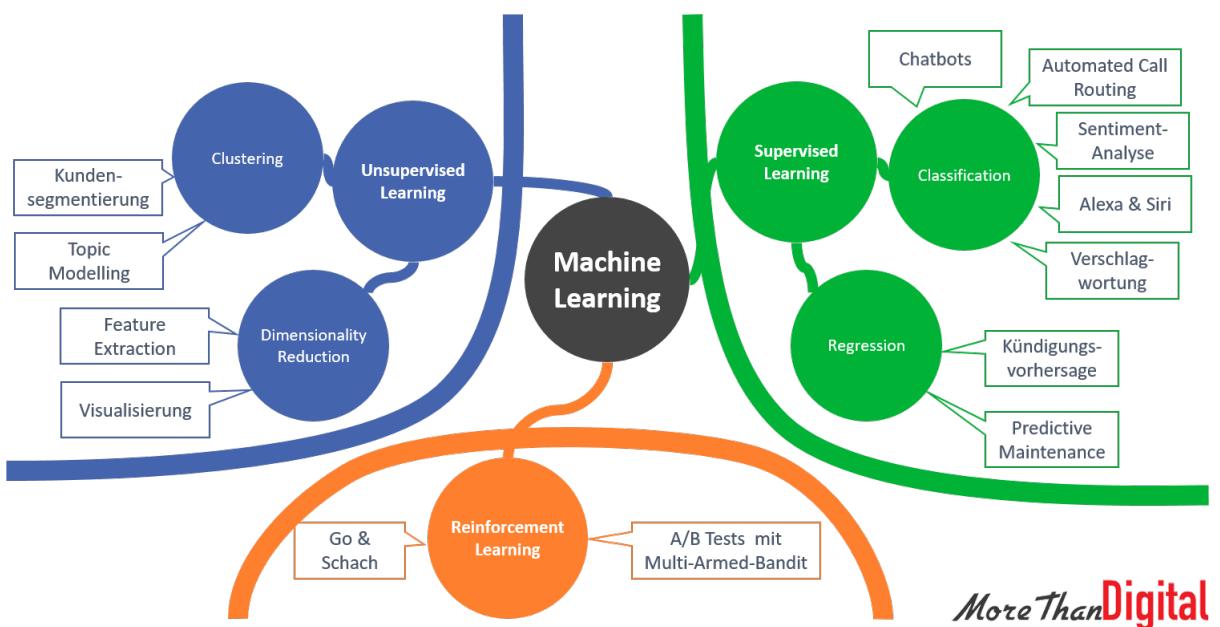


Abbildung 2.5: Übersicht der Teilbereiche vom Maschinellen Lernen [8]

Supervised Learning

Das Supervised Learning (dt. Überwachtes Lernen) ist aktuell die am stärksten verbreitete Form des maschinellen Lernens. Wie der Name es dabei schon suggeriert, wird der Lernprozess der künstlichen Intelligenz bei diesem Verfahren überwacht. Das geschieht im Regelfall durch die Bewertung des Ergebnisses von einem neuronalen Netz. Diese Bewertung muss dabei nicht von Hand durchgeführt werden, sondern wird durch gelabelte Daten realisiert. Das bedeutet als Datensätze werden beispielsweise Bilder von Katzen und anderen Tieren genutzt und es wird als Zusatzinformation zu einem Bild abgespeichert, um was es sich beim jeweiligen Beispiel handelt. Ziel soll es dabei sein, Katzen aus den Daten herauszufiltern. Wenn das neuronale Netz die Datensätze verarbeitet und Fehler über die vorhandenen Labels erkennt, kann es über die Fehlerrückführung Anpassungen durchführen

und sich so stückweise verbessern. Die Klassifizierung von Katze oder keine Katze ist ein geläufiges Beispiel für die Mustererkennung durch das überwachte Lernen. Hierbei muss die Klassifizierung nicht binär sein, sondern sie kann auch mehrere Klassen beinhalten.

Ein anderes Anwendungsgebiet des überwachten Lernens ist die Regression. Mit Hilfe der Regression soll anstelle der Klassifizierung ein reeller Wert als Ausgabe entstehen. Ein Beispiel zur Unterscheidung der beiden Vorhersagemodelle wäre, analysiere die aktuelle Temperatur (Regression) der Maschine durch bestimmte Eingabedaten oder klassifizierte ob die Maschine erhitzt ist oder nicht (Klassifikation).

Mathematisch betrachtet suchen wir bei der Klassifikation eine Funktion, welche im einfachsten Fall zwei Punktwolken (aus Datensätzen) in einem zweidimensionalen Koordinatensystem voneinander trennt (siehe Abbildung 2.6).

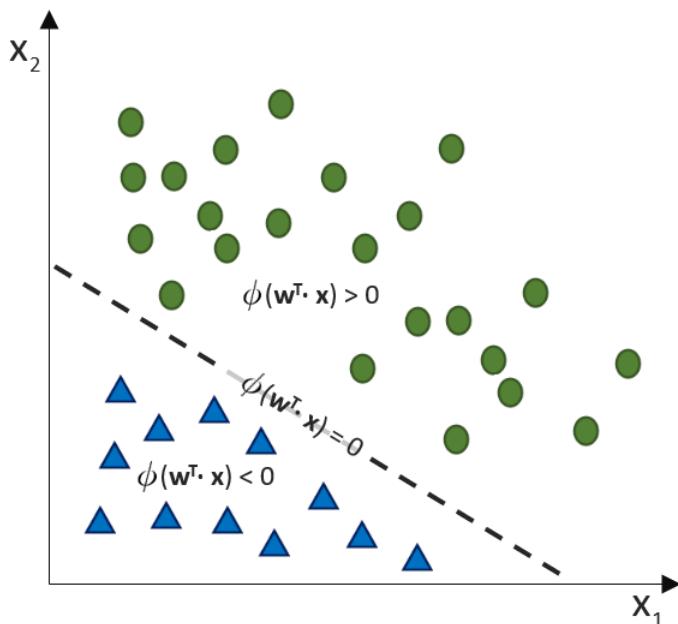


Abbildung 2.6: Klassifikation von Punktwolken [9]

Bei der Regression wird mathematisch eine Funktion generiert, welche eine vorhandene Punktwolke (aus Datensätzen) bestmöglich abbildet, um so zukünftige Werte genau vorherzusagen (siehe Abbildung 2.7).

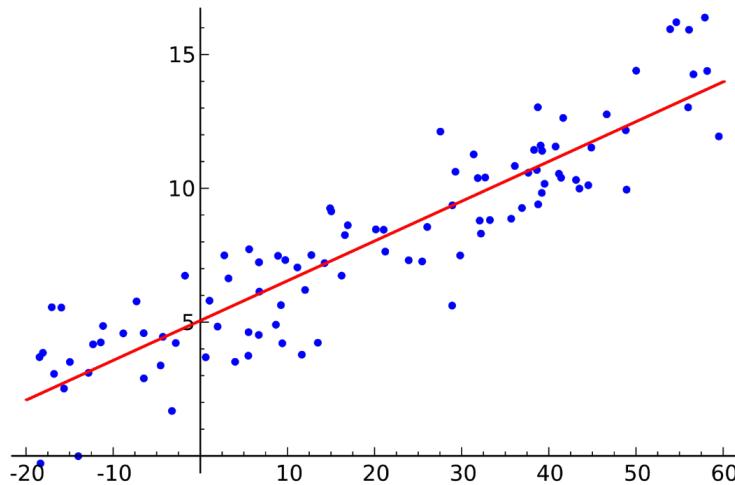


Abbildung 2.7: Lineare Regression von Punktfolke [9]

Solche Funktionen sind anschließend die Künstliche Intelligenz beziehungsweise die erwähnte Zielfunktion. Wir können in diese Funktionen Daten eingeben und bekommen daraus das gewünschte Ergebnis. Also zum Beispiel die Einordnung von Bildern oder die Vorhersage von zukünftigen Aktienkursen.

Unsupervised Learning

Im Gegensatz zum Supervised Learning, existieren beim Unsupervised Learning (dt. Unüberwachtes Lernen) keine Labels. Das kann verschiedene Gründe haben, beispielsweise ist die Datenerfassung möglicherweise unzureichend gewesen, der Aufwand des Labelns zu hoch oder es sollen explizit Algorithmen des Unsupervised Learning angewandt werden, um neue Informationen über eine vorhandene Datenmenge zu generieren.

Der Umstand dass keine „Überwachung“ existiert ist dabei Fluch und Segen zu gleich. Denn es bedeutet einerseits weniger Aufwand in der Vorbereitung der Daten, aber anderseits schwindet damit der Einfluss auf die Zielfunktion. Das Ergebnis ist beim Unsupervised Learning im Vorfeld somit nicht klar definiert, sondern es werden von der **KI** während des Lernvorgangs selbstständig Zusammenhänge in den Datensätzen erkannt.

Wie in Abbildung 2.5 zu sehen gibt es beim unüberwachten Lernen ebenfalls zwei große Anwendungsbereiche. Zum einen das Clustering, was im Ergebnis einer multiplen Klassifikation aus dem Supervised Learning entspricht und zum anderen die Dimensionsreduktion. Das Clustering eignet sich dabei hervorragend, um günstige Datenmengen zu kategorisieren. Mit günstig ist damit der Umstand gemeint, dass wir die Daten nicht labeln müssen und somit dieser Vorbereitungsschritt entfällt. Ein simples Beispiel wären Datentupel bestehend aus Gewicht und Größe eines Hundes. Anhand dieser Informationen kann durch das Unsupervised Learning eine Gruppierung der einzelnen Hunderassen erfolgen. Ein deutlich praxisorientierter Ansatz wäre die Erkennung von Mustern in Kundendaten. Daraus abge-

leitet könnte ein Unternehmen beispielsweise passendes Marketing für die entsprechenden Zielgruppen betreiben.

Ein Beispiel für die Dimensionsreduktion hingegen, ist die Komprimierung von Daten, wie beispielsweise Bilddaten. In eine Funktion der Dimensionsreduktion würde demnach eine Bilddatei inseriert werden und es würde die selbe Datei als Ergebnis erzeugen. Allerdings soll innerhalb des Verfahrens die Dateigröße reduziert werden, in dem unwichtige Informationen durch den Algorithmus herausgefiltert werden. Welche Informationen unwichtig sind wird dabei nicht durch den Entwickler festgelegt, sondern durch die **KI** selbst berechnet.

Die folgende Abbildung 2.8 visualisiert das mathematische Reduzieren der Dimensionen von einem drei dreidimensionalen in einen zweidimensionalen Raum.

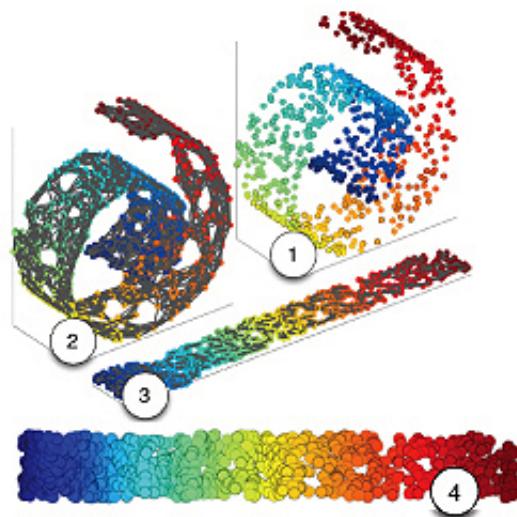


Abbildung 2.8: Dimensionsreduktion 3D zu 2D [10]

2.2 Reinforcement Learning

Dieser Abschnitt handelt von Reinforcement Learning (dt. Bestärkendes Lernen) und soll einen Überblick über das Konzept und die wichtigsten Methoden dieser Form des maschinellen Lernens geben.

Wie bereits oben beschrieben, bildet das Reinforcement Learning die dritte große Gruppe von Verfahren des maschinellen Lernens. Es ist ähnlich wie die neuronalen Netze von der Natur inspiriert und abstrahiert das natürliche Lernverhalten von uns Menschen.

Insbesondere in frühen Lernstadien mit wenig vorhandenem Wissen über die zu lösende Aufgabe agieren Menschen zunächst mittels „Trial and Error“. Das bedeutet es werden zufällige Aktionen durchgeführt und die Auswirkungen dieser beobachtet. Wenn die Aktion schlecht war erhalten wir dann eine Bestrafung (z.B. Schmerzen) und wenn die Aktion gut war erhaltene wir eine Belohnung (z.B. soziale Anerkennung). Das Ziel eines Menschen ist am Ende eines solchen Lernprozess die Maximierung der erhaltenen Belohnung.

Beispiel als Erläuterung:

Ein Mensch probiert das Gitarre Spielen zu erlernen. Zu Beginn besitzt er keinerlei Wissen über die Funktion der Saiten sowie die unterschiedlichen Grifftechniken am Gitarrenhals. Würde diese Person nun eine Anleitung durch einen Lehrer erfahren, wäre dies im Rahmen unseres Beispiels vergleichbar mit dem Supervised Learning. Hätte er stattdessen einige Videos von spielenden Gitarrist, könnte er implizit das Gitarre Spielen erlernen und das könnte als Unsupervised Learning bezeichnet werden. Diese beiden Vorgehensweisen sind zum Erlernen einer solch komplexen Fähigkeit sicherlich am üblichsten. Doch wie würde der Mensch vorgehen, wenn er völlig auf sich allein gestellt ist?

Wenn kein Wissen über das Spielen der Gitarre vorhanden ist, besteht die einzige Möglichkeit zum Erwerben der notwendigen Informationen im zufälligen Ausprobieren von verschiedenen Saiten oder Griffen. Jede Interaktion mit der Gitarre gibt dabei ein neue Beobachtung in Form eines Tons zurück. Stark vereinfacht würden „gerade“ Töne dabei als positiv und „ungerade“ Töne als negativ bewertet werden. Dadurch kann der Mensch die nicht vorhandenen Daten selbstständig generieren und durch die Belohnung „Labels“, sodass er in Folge dessen mit Hilfe dieser erworbenen Daten den Lernprozess vollziehen kann. Innerhalb des Lernprozesses würden dann negativ bewertete Aktionen unterlassen und positiv bewertete Aktionen verstärkt ausgeführt werden. Durch die daraus resultierende Ausrichtung auf möglichst positives Feedback, erlernt der Mensch ohne Vorwissen das Gitarre spielen.

Als Ergebnis dieser Vorgehensweise ist eine Reinforcement Learning **KI** in vielen Fällen extrem flexibel, weil sie ohne Vorwissen eine neue Umwelt erkunden kann und durch die Definition von Belohnungen ein gewünschtes Zielverhalten approximiert. In Folge dessen kann sich die **KI** auf neue Variablen wie beispielsweise die Veränderung von Spielregeln reagieren.

Ein eindrucksvolles Beispiel dafür ist eine Reinforcement Learning **KI** von Google aus dem Jahre 2015, welche sich 49 Atari⁶ Spiele ohne Vorwissen selbstständig beigebracht hat.[11]

Neben den bereits erwähnten Beispielen gibt es noch weitere Anwendungsmöglichkeiten für Reinforcement Learning. Im Folgenden sollen einige Beispiele dafür aufgeführt werden.

Beispiele:

- Kontrollsysteentechnik: Kraftstoffsparende Steuerung einer Turbine
- Robotik: Greifen von beliebigen Gegenständen
- Aktienhandel: **KI** die selbstständig Kauf- und Verkaufsentscheidungen trifft
- Autonomes Fahren
- Bilderkennung: Interpretation von menschlicher Gestik- und Mimik

Die populären spielerischen Anwendungen wie das bereits erwähnte AlphaGoZero oder die Atari **KI** haben in der Vergangenheit lediglich zum Demonstrieren des Entwicklungsstandes und Evaluieren der Fähigkeiten einer Reinforcement Learning **KI** gedient. Außerdem lassen sich Spiele hervorragend simulieren, wodurch die notwendigen Lerndaten ohne größeren Aufwand generiert werden können.

Nachfolgend soll das grundlegende Konzept zur formalen Beschreibung eines Reinforcement Learning Problems erläutert werden. Mit Hilfe dieses Konzepts kann anschließend die Vorgehensweise zur Lösung eines solchen Problems geschildert werden.

2.2.1 Markow Entscheidungsproblem

Zur Formulierung eines Reinforcement Learning Problems wird ein mathematisches Modell genutzt, welches sich **Markow Entscheidungsproblem (MDP)** nennt. Mit Hilfe dieser Abstraktionsebene wird die Bearbeitung und Lösung eines entsprechenden Problems vereinfacht. Außerdem kann das grundlegende Modell der **MDPs** auf alle Reinforcement Learning Probleme angewandt werden. Deshalb soll in diesem Abschnitt erklärt werden, worum es sich dabei handelt und anschließend werden die daraus abgeleiteten Lösungsansätze erläutert.

Als **MDP** wird ein mathematisches Modell zur Formulierung eines sequentiellen Entscheidungsproblems bezeichnet, das Ziel im Rahmen eines **MDP** ist es, die optimale Handlungsstrategie zur Lösung der entsprechenden Problemstellung zu finden. Dabei wird die optimale

⁶<https://www.atari.com/>

Strategie durch die maximale Belohnung repräsentiert. Eine solche Belohnung muss durch den Entwickler definiert werden, dass ist der Grund wieso Reinforcement Learning in manchen Quellen auch als „Semi-Supervised Learning“ bezeichnet wird, denn die Belohnung ist ein implizites Label für die im Lernprozess gesammelten Daten.

Im Rahmen eines **MDP** ist der Hauptakteur der sogenannte Agent. Dieser interagiert mit der Umwelt (Environment) in der er sich befindet. Dafür stehen ihm eine Menge von Aktionen (Actions) zur Verfügung. Damit er sich für eine Aktion entscheiden kann erhält der Agent zunächst den aktuellen Zustand (State) der Umwelt und im Anschluss an eine Aktion eine Belohnung (Reward) von der Umwelt. Wie bereits erwähnt ist das Ziel des Agenten die Maximierung der dabei erhaltenen Belohnung. Die Problemstellung bei diesem Prozess liegt darin, dass die Lösung nicht in einem Schritt, sondern nur in einer Abfolge von vielen Schritten erreicht werden kann (sequentielles Entscheidungsproblem). In Folge dessen ist es notwendig, dass der Agent seine Handlung bezogen auf die Gesamtdauer der Problemstellung plant. Die daraus resultierende Handlungsstrategie wird Policy genannt und mappt jeden erhaltenen Zustand zu der optimalen Aktion. Dabei muss garantiert werden, dass die ausgewählte Aktion nicht nur für einen kurzen Zeithorizont, sondern für die Gesamtdauer der Aufgabenstellung die höchst mögliche Belohnung erzeugt. In dieser Vorhersage über die zukünftige Belohnung steckt die Schwierigkeit eines **MDP**.

Des Weiteren muss ein vorliegendes Problem die Eigenschaft erfüllen, dass der aktuelle Zustand alle notwendigen Informationen bereitstellt um die optimale Handlung zu ermitteln, dass bedeutet die Historie der Zustände ist nicht von Relevanz und der Agent kann jeden Zustand so betrachten, als wäre ein Problemdurchlauf gerade erste begonnen worden. Dieses Prinzip wird als Markow Eigenschaft bezeichnet. Allerdings ist es dehnbar, beispielsweise bei Computerspielen wird als Zustand an den Agenten nicht ein Bild, sondern eine Sequenz von Bildern übergeben, weil sonst die Informationen über die Bewegungsgeschwindigkeit der einzelnen Objekte fehlt. Damit wäre die Eigenschaft trotzdem erfüllt, es wird nur der Zustandsbegriff gedanklich erweitert.

Ein weiteres Prinzip des klassischen **MDP** ist das Übergangsmodell. Dadurch wird die Wahrscheinlichkeit für den Übergang von einem Ausgangszustand in den Folgezustand auf Basis einer gewählten Aktion bezeichnet. Dies ist vor allem bei beispielsweise realen Umgebungen relevant, denn ein Roboter kann nicht garantieren das eine gewählte Aktion zum gewünschten Folgezustand führt, er könnte ebenso dabei umgestoßen werden und landet dadurch in einem anderen Zustand. Eine solche Umgebung nennt man stochastisch. Im Gegensatz dazu wird in dieser Arbeit die deterministische Umgebungen „4 Gewinnt“ betrachtet, bei der alle Aktionen garantieren in die gewünschten Folgezustände führen. Deshalb kann der Wahrscheinlichkeitsfaktor dabei vernachlässigt werden.

Da wir nun mit dem grundsätzlichen Ablauf sowie Eigenschaften eines **MDP** vertraut sind, sollen im Folgenden das Modell und die Notationen zur Beschreibung eines **MDP** genauer erläutert werden.

Modell

Das dem Reinforcement Learning sowie **MDP** zugrunde liegende Modell, besteht aus den zwei Entitäten *Agent* und *Environment (Umwelt)* sowie den Kommunikationskanälen *Action (Aktion)*, *State (Zustand)* und *Reward (Belohnung)*. Eine Visualisierung dieses Modells bietet die Abbildung 2.9. Im Folgenden sollen die einzelnen Begriffe mit Hilfe von Beispielen erläutert werden.

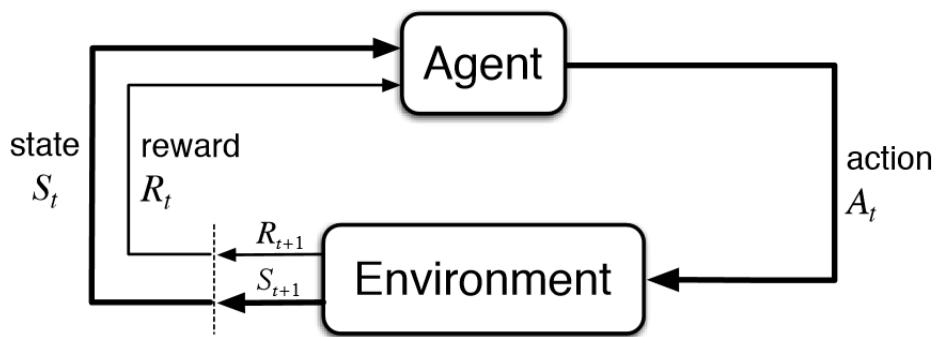


Abbildung 2.9: Reinforcement Learning Modell [12]

Agent

Der Agent ist die Kernfigur in dem Modell. Er kann auf Grundlagen von Beobachtungen der Umwelt Aktionen ausführen und bekommt dafür eine Belohnung als Rückmeldung. Auf Basis dieser Rückmeldungen kann der Agent anschließend eine Handlungsstrategie entwickeln, indem er negativ bewertetes Handeln tendenziell unterlässt und positiv bewertetes Handeln öfter ausführt. Damit bildet der Agent die zu entwickelnde Künstliche Intelligenz ab.

Beispiele für Agents:

- 4 Gewinnt: Ein menschlicher Spieler oder die **KI** als Computerprogramm
- Autonomes Fahren: Das Fahrzeug
- Hundetraining: Der Hund

Environment

Mit der Umwelt ist der Bereich gemeint, indem der Agent agieren kann beziehungsweise von dem er seine Informationen bekommt. Wie der Agent mit der Umwelt interagieren kann,

wird unter den Punkten State und Actions deutlich.

Beispiele für Environments:

- 4 Gewinnt: Das Spielfeld mit allen Spielsteinen
- Aktienhandel: Der Aktienmarkt
- Turbinensteuerung: Die Turbine

State

Der Zustand ist ein Abbild der Umwelt zu einem bestimmten Zeitpunkt. Dieses wird dem Agenten kommuniziert, sodass er auf Grundlage der aktuellen Situation eine Entscheidung treffen kann.

Beispiele für States:

- 4 Gewinnt: Die aktuelle Position aller Spielsteine
- Aktienhandel: Der aktuelle Aktienkurs
- Pong (Computerspiel): Die aktuelle Position des eigenen Cursors sowie des Balles

Action

Mit Hilfe der Aktionen kann der Agent Einfluss auf die Umwelt ausüben. Dabei ist in der Regel ein Aktionsraum (engl. action space) vorgegeben, aus dem der Agent eine Aktion wählen kann. Bei „4 Gewinnt“ besteht der Aktionsraum aus allen möglichen Spalten, bei dem Spiel „Pong“ beispielsweise nur aus dem nach oben oder nach unten bewegen des Cursors. In der Folge einer Aktion wird anschließend ein neuer Zustand und je nach Situation auch eine Belohnung an den Agenten übermittelt.

Beispiele für Actions:

- 4 Gewinnt: Auswahl einer freien Spalte
- Hundetraining: Ausführung eines Kommandos
- Pong (Computerspiel): Bewegung nach oben

Reward

Der Reward bezeichnet die Belohnung, welche der Agent für seine Aktionen bekommt. Damit kann der Agent sein eigenes Handeln bewerten. Die Belohnung wird hierbei immer durch eine reelle Zahl repräsentiert. Diese Zahl kann eine positive, aber auch negative Ausprägung

annehmen. Dadurch wird gutes Handeln bestärkt und schlechtes bestraft, denn der Agent probiert immer seine erhaltene Belohnung zu maximieren.

Beispiele für Rewards:

- 4 Gewinnt: Positive Belohnung beim Sieg oder einem Unentschieden, negative bei einer Niederlage
- Aktienhandel: Der Handelsprofit ist die Belohnung
- Schule: Die Schulnoten

Notation

- $S = \{s_1, \dots, s_n\}$ ist die Menge der Zustände.
- $A = \{a_1, \dots, a_n\}$ ist die Menge der Aktionen.
- $R = \{r_1, \dots, r_n\}$ ist die Menge der Belohnungen.
- $T(s, a, s')$ bezeichnet die Übergangsfunktion (engl. transition function) und drückt die Wahrscheinlichkeit für den gewünschten Zustandsübergang mit Hilfe der Aktion a_t aus. Wenn $T = 0.8$ ist würde der entsprechende Zustand mit einer Wahrscheinlichkeit von 80% erreicht werden. Deshalb kann die Funktion auch als $P(s, a, s')$ beschrieben werden.
- $R(s, a, s')$ bezeichnet die Belohnungsfunktion und beschreibt den Reward der im nächsten Zustand s_{t+1} zu erwarten ist.
- $a = \pi(s)$ bezeichnet die Policy für deterministische Umgebungen.
- $\pi(s, a) = P(s, a)$ bezeichnet die Policy für stochastische Umgebungen.

Diskontierter Reward

In diesem Abschnitt soll der kumulative Reward erläutert werden, welcher innerhalb eines **MDP** vom Agenten gesammelt wird. Wie bereits zuvor erwähnt, ist es das Ziel vom Agenten diesen gesammelten Reward zu maximieren.

Auf Grundlage der oben dargestellten Notation kann der kumulative Reward im einfachsten Fall wie folgt beschrieben werden:

$$G_t = R_{t+1} + R_{t+2} + R_{t+3} + \dots + R_T \quad (2.3)$$

Allerdings wird bei dieser Beschreibung von groß T für den finalen Zeitschritt ausgegangen. Diese Annahme ergibt vor allem für endliche Probleme wie „4 Gewinnt“ Sinn, weil hierbei der Abschluss des Spiels durch groß T symbolisiert werden kann. Im Reinforcement Learning wird dabei auch von episodischen Problemen gesprochen, wobei ein Spieldurchlauf in diesem Beispiel als eine Episode bezeichnet wird.

Jedoch existieren auch kontinuierliche Probleme, wodurch $T = \infty$ wäre und dementsprechend der kumulative Reward ebenfalls unendlich wird. Da die Maximierung des Rewards in die Unendlichkeit aus mathematischer Sicht wenig erfolgversprechend ist wird eine Lösung für dieses Problem gebraucht. Diese Lösung ist der sogenannte Diskontierungsfaktor γ .

$$G_t = R_{t+1} + R_{t+2} + R_{t+3} + \dots + R_T \quad (2.4)$$

$$G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \quad (2.5)$$

Der Faktor γ kann immer nur Ausprägungen zwischen 0 und 1 annehmen und erzeugt durch den zeitlichen Parameter in der Potenz eine mathematische Begrenzung der Unendlichkeit. Je mehr Zeitschritte vergangen sind, umso kleiner wird der Faktor γ . Da dieser wiederum multipliziert wird mit dem jeweiligen Reward des aktuellen Zeitschritts wird der kumulative Reward begrenzt.

Neben der Lösung für unendliche Probleme kann mit Hilfe des Diskontierungsfaktos die Gewichtung von kurzfristigen sowie langfristigen Entscheidungen gesteuert werden. Diese Funktion ist dadurch auch für episodische Problemstellungen wichtig. Dabei lassen kleine γ Werte den Agenten eher kurzfristig relevante Entscheidungen treffen und hohe γ Werte führen zu der verstärkten Berücksichtigung von langfristigen Auswirkungen der Entscheidungen.

Optimale Policy und Value Functions

Im Folgenden soll definiert werden, wie ein Zustand sowie ein Zustand-Aktions Tupel bewertet wird. Auf dieser Grundlage kann anschließend die Definition für eine optimale Policy formuliert werden. Für die jeweiligen Definitionen ist das Verständnis des diskontierten Rewards aus dem vorherigen Abschnitt notwendig. Die optimale Policy beschreibt die Lösung eines **MDP** und ist somit das angestrebte Ziel.

State-Value Function

Die State-Value Function beschreibt den erwarteten diskontierten Reward aus dem Zustand s unter der Annahme, dass die Policy π verfolgt wird. Der Parameter E_π steht für den Erwartungswert und ist notwendig in stochastischen Umgebungen, weil die Zustandsabfolge

hierbei nicht garantiert ist. Ansonsten kann das E_π vernachlässigt werden.

$$V_\pi(s) := E_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \right] \quad \forall s \in S \quad (2.6)$$

Die State-Value Function mit dem insgesamt höchsten Reward ist gleichzeitig die optimale. Demnach wird das Maximum über alle State-Value Functions bestimmt:

$$V_*(s) := \max_{\pi} V_\pi(s) \quad \forall s \in S \quad (2.7)$$

Die optimale Policy entspricht wiederum der optimalen State-Value Function und kann mit Hilfe dieser wie folgt definiert werden:

$$\pi_* := \arg \max_{\pi} V_\pi(s) \quad \forall s \in S \quad (2.8)$$

Action-Value Function

Neben der Beurteilung eines Zustandes besteht zudem die Möglichkeit ein Tupel aus Zustand und Aktion zu bewerten. Die entsprechende Funktion wird Action-Value Function genannt und beschreibt den erwarteten diskontierten Reward, wenn im aktuellen Zustand s eine Aktion a ausgewählt wird und anschließend die Policy π verfolgt wird.

$$Q_\pi(s, a) := E_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \right] \quad \forall s \in S \quad \forall a \in A \quad (2.9)$$

$Q_\pi(s, a)$ ist für den Agenten ein Indikator zur Bewertung aller Aktionen a im Zustand s . Das Q steht dabei für Quality, weil der resultierende Wert die Qualität einer Aktion im gegebenen Zustand beschreibt. Dieser Ansatz wird vor allem für das danach benannte Q-Learning in einem späteren Abschnitt wichtig.

Da $V_*(s)$ der maximale erwartete Reward ausgehend von Zustand s ist, muss das Maximum von $Q_\pi(s, a)$ über alle möglichen Aktionen dem gleichen Wert entsprechen. Der Zusammenhang kann wie folgt beschrieben werden:

$$V_*(s) := \max_a Q_*(s, a) \quad \forall s \in S \quad \forall a \in A \quad (2.10)$$

Die optimale Policy entspricht hierbei ebenso der optimalen Action-Value Function und kann mit Hilfe dieser wie folgt definiert werden:

$$\pi_* := \arg \max_a Q_*(s, a) \quad \forall s \in S \quad \forall a \in A \quad (2.11)$$

Nun sind alle relevanten Notationen, das Modell und die Zielsetzung eines **MDP** beschrieben worden. Auf Grundlage dieses Wissens werden im Anschluss die drei Verfahrensklassen Dynamic Programming, Temporal Difference Learning und Monte Carlo näher erläutert.

Zusätzlich werden für die Klassen Dynamic Programming und Temporal Difference Learning entsprechende Methoden vorgestellt. Diese können anschließend zur Lösung eines **MDP** genutzt werden.

2.2.2 Dynamic Programming

Das Dynamic Programming bezeichnet eine Vorgehensweise zur Lösung von Optimierungsproblemen. Dabei werden zuerst die Lösungen der kleinsten Teilprobleme berechnet und diese anschließend zu einer Lösung für das nächst größere Teilproblem zusammengesetzt. Diese Vorgehensweise wird solange fortgesetzt, bis das ursprüngliche Gesamtproblem gelöst ist. Die hierbei entstehenden Teilergebnisse werden zur Verbesserung der Laufzeit des Verfahrens in einer Tabelle gespeichert.⁷

Dieser Dynamic Programming Algorithmus basiert auf dem Konzept der sogenannten Bellmann Equation, welche ein wichtiger Baustein für die Umsetzung vieler Verfahren im Bereich des Reinforcement Learnings darstellt.

Ein Grund dafür ist, dass das Dynamic Programming die älteste Verfahrensklasse zur Lösung eines **MDP** ist und somit von vielen nachfolgenden Algorithmen als Basiskonzept genutzt wurde. Deshalb ist das Verständnis für den Algorithmus und vor allem von der Bellman Equation wichtig für die nachfolgenden Lösungsverfahren.

Bevor nun eine entsprechende Methode des Dynamic Programming vorgestellt wird, betrachten wir zunächst die oben genannte Bellmann Equation für V_* und anschließend für Q_* :

$$V_*(s) = \max_a \sum_{s'} P(s, a, s')[R(s, a, s') + \gamma V_*(s')] \quad \forall s \in S \quad (2.12)$$

$$Q_*(s, a) = \sum_{s'} P(s, a, s')[R(s, a, s') + \gamma \max_{a'} Q_*(s', a')] \quad \forall s \in S \quad \forall a \in A \quad (2.13)$$

Warum wird diese Gleichung zur Lösung eines **MDP** gebraucht, obwohl wir bereits die Value Functions kennengelernt haben?

Das Problem der Value Functions liegt darin, dass wir diese nicht effizient berechnen können. Denn zur Berechnung der Value Function wäre es notwendig, in jedem Zustand alle Folgezustände nach der entsprechenden Policy zu betrachten und dieses Ergebnis als neuen Value für den aktuellen Zustand zu setzen. Diese Prozedere müsste für jeden einzelnen Zustand des **MDP** durchgeführt werden.

Aus diesem Grund wird das Dynamic Programming beziehungsweise die Bellmann Equation

⁷https://de.wikipedia.org/wiki/Dynamische_Programmierung

eingesetzt. Wie an der oben aufgeführten Gleichung zu sehen, zerlegen wir unser Gesamtproblem in den Reward des nächsten Zeitschritts $R(s, a, s')$ und die Aktion mit der besten Bewertung \max_a im nächsten Zustand s' , multipliziert mit dem Diskontierungsfaktor γ . Dabei steckt durch den hinteren Teil der Gleichung $Q_\pi(s', a')$ ein rekursiver Aufruf in dieser Funktion. Im Grunde betrachten wir also in jedem Zustand s den Folgezustand s' und aktualisieren auf Basis davon unseren aktuellen Wert. Im Reinforcement Learning wird diese Vorgehensweise auch als Bootstrapping bezeichnet.

Wie die Idee dieser Gleichung nun dazu beiträgt eine optimale Policy zu berechnen wird nachfolgend an einem Beispiel erläutert.

Innerhalb des Dynamic Programmings gibt es zwei Prominente Verfahren zur Lösung eines **MDPs**, die Value Iteration und die Policy Iteration. Wir werden nun exemplarisch den Algorithmus der Value Iteration betrachten und damit das erste Verfahren zur Lösung eines **MDP** kennenlernen.

Value Iteration

Algorithm 1 : Value Iteration

Initialize array V arbitrarily (e.g. $V(s) = 0$ for all $s \in S$)

```

repeat
     $\Delta \leftarrow 0$ 
    for each  $s \in S$  do
         $v \leftarrow V(s)$ 
         $V(s) \leftarrow \max_a \sum_{s'} P(s, a, s')[R(s, a, s') + \gamma V(s')]$ 
         $\Delta \leftarrow \max(\Delta |v - V(s)|)$ 
    end for
until  $\Delta < 0$  (small positive number)

```

Zur Veranschaulichung des Algorithmus soll dieser anhand eines einfachen Beispiels erläutert werden.

Anwendungsbeispiel:

Es sei eine 3x3 Matrix als Umwelt gegeben (siehe Abbildung 2.10). Es gibt eine Belohnung von +1 und eine Bestrafung von -1. Das Ziel des Agenten ist es die maximale Belohnung zu erhalten, wenn er auf einem zufälligen Feld in der Matrix ausgesetzt wird. Dafür kann er sich zu jedem Zeitschritt k um ein Feld in alle Himmelsrichtungen bewegen. Die Übergangsfunktion ist hierbei der einfacheheitshalber für jede Richtung mit 0,25 und γ mit 0,9 definiert. Des Weiteren bleibt der Agent bei Bewegungen die ihn außerhalb der Matrix oder auf das mittlere Feld führen stehen.

Diese Anwendung ist realitätsfern, aber es geht darum an einem möglichst simplen Beispiel der Algorithmus zu verdeutlichen. Wenn das Anwendungsszenario etwas ausgebaut würde,

könnte damit beispielsweise auch ein Labyrinth gelöst werden oder ein Staubsaugerroboter die Navigation zu seiner Ladestation erlernen.

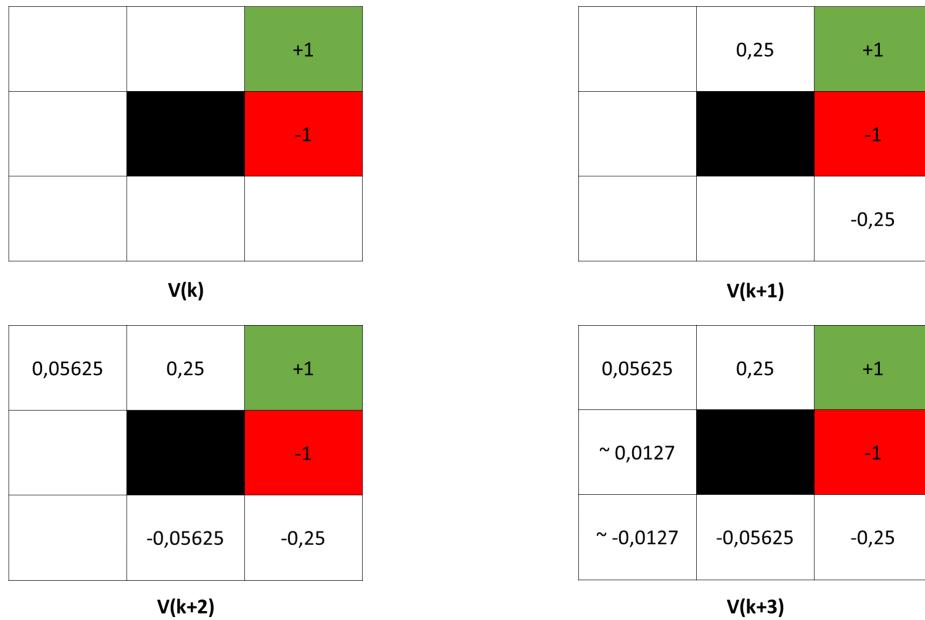


Abbildung 2.10: Ablauf des Value Iteration Algorithmus

Zu Beginn des Lernprozesses werden alle Zustände mit 0 initialisiert. Anschließend wird im ersten Schleifendurchlauf für jeden Zustand die Aktualisierung des State-Values mit der Formel $V(s) \leftarrow \max_a \sum_{s'} P(s, a, s') [R(s, a, s') + \gamma V(s')]$ vorgenommen. Dabei ist das Ergebnis dieser Operationen für fast alle Zustände 0, sodass nur in den Zuständen neben der Belohnung sowie der Bestrafung eine Veränderung stattfindet.

Für das Feld, welches an die Belohnung angrenzt lautet die Berechnung wie folgt $V(s) \leftarrow 0,25 * [1 + 0,9 * 0]$. Das Ergebnis ist wie in der Abbildung 2.10 zu sehen 0,25. Die gleiche Berechnung wird ebenfalls für den Zustand der an die Bestrafung angrenzt durchgeführt, hierbei ändert sich lediglich das Vorzeichen.

Im zweiten Zeitschritt $k + 1$ kann nun einen Zustand früher eine Aktualisierung der State-Values vorgenommen werden. Die Berechnung des Zustands in der oberen linken Ecke lautet nun wie folgt: $V(s) \leftarrow 0,25 * [0 + 0,9 * 0,25]$ und das Ergebnis davon lautet 0,05625.

Durch dieser Vorgehensweise propagiert sich die Belohnung durch die gesamte Matrix. Wie in Zeitschritt $k + 3$ zu sehen erhalten wir dadurch anhand der State-Values eine Möglichkeit das vorliegende Problem zu lösen, indem in jedem Zustand der Folgezustand mit dem mathematisch höchsten Wert gewählt wird. Die daraus resultierende Handlungsstrategie ist unsere Policy, welche die Belohnung maximiert. Zuvor müssen allerdings weitere Schleifendurchläufe ausgeführt werden, damit der vorhandene Fehler Δ nach und nach minimiert wird, bis das Ergebnis zur gewünschten Lösung des Problems konvergiert.

Im Anschluss an den Lernvorgang kann der Agent an einer beliebigen Stelle in der Matrix ausgesetzt werden und findet von dort immer den optimalen Pfad zur Belohnung. Damit wurde das vorliegende **MDP** gelöst. Gleichzeitig konnte an diesem Beispiel die Grundidee der Bellmann Equation demonstriert werden.

Wie am obigen Beispiel gezeigt, funktioniert der Value Iteration Algorithmus und wir sehen, dass er zur gewünschten Lösung konvergiert. Allerdings existiert auch hier ein Problem, denn komplexere Probleme mit größeren Zustandsräumen können nicht bewältigt werden. Zum einen wegen des wiederholten Iterierens durch alle möglichen Zustände sowie zum anderen wegen der Speicherung der Values. Dafür wird in der Regel eine einfache Tabelle genutzt, deshalb kann die Value Iterationen auch den tabellarischen Reinforcement Learning Verfahren zugeordnet werden.

Ein Beispiel für ein komplexeres Problem wäre das Spiel „4 Gewinnt“. Würden wir eine Value Iteration dafür nutzen, müsste in diesem Anwendungsfall durch $4.5 * 10^{12}$ [13] Zustände iteriert werden. Außerdem müssten alle dabei entstehenden Werte in einer Tabelle gespeichert werden. Dadurch hätten wir durch das Verfahren eine nicht hinnehmbare Lösungszeit und die Tabelle würde sehr viel Speicherplatz in Anspruch nehmen. Gleichzeitig wären Zugriffe auf die Tabelle ebenfalls langsam. Dieser Zustandsraum wäre bereits sehr groß, wenn allerdings Beispiele wie Schach oder Computerspiele mit tausenden Pixeln betrachtet werden, erkennt man die Unmöglichkeit mit Hilfe der Value Iteration eine optimale Policy zu ermitteln.

Aus diesem Grund sollen mit der nächsten Verfahrensklasse weitere Methoden vorgestellt werden, die auch komplexere Probleme lösen können.

2.2.3 Temporal Difference Learning

Das Temporal Difference Learning ist ebenso wie das Dynamic Programming eine Verfahrensklasse des Reinforcement Learning und enthält verschiedene Methoden zur Lösung eines **MDP**. Hierbei wird ebenfalls Bootstrapping verwendet. Ein großer Unterschied zu der Vorgehensweise des Dynamic Programming liegt im sogenannten Sampling. Damit wird das Generieren von Beispieldaten durch „Trial and Error“ bezeichnet. Dies hat zur Folge, dass es nicht notwendig ist alle Zustände eines Problems zu betrachten. Es werden lediglich die scheinbar relevanten Zustände betrachtet, weil diese im Rahmen der Erkundung des Problems häufiger vorkommen.

Außerdem benötigen viele Methoden außerhalb des Dynamic Programming sowie auch im Temporal Difference Learning kein Modell von ihrer Umwelt. Damit ist das nicht Vorhandensein der Übergangs- oder Rewardfunktion gemeint. Solche Verfahren werden als Model-Free bezeichnet. Im Gegensatz dazu werden Methoden, welche ein solches Modell der Umwelt nutzen als Model-Based bezeichnet.

In Folge dessen sind Model-Free Verfahren flexibler einsetzbar, weil keine Modellierung der

Umwelt vorgenommen werden muss. Außerdem ist es für manche Probleme nicht möglich ein entsprechendes Abbild der Umwelt für die **KI** zu erzeugen. Dies ist der Grund wieso Model-Free Verfahren bisher die stärkste Verbreitung im Reinforcement Learning gefunden haben. Dennoch haben auch Model-Based Methoden ihre Daseinsberechtigung, weil diese vor allem effizienter lernen. Der bekannteste Vertreter eines solchen Verfahrens ist das bereits erwähnte AlphaGoZero.

Beim Temporal Difference Learning wird mit jedem Zeitschritt eine Aktualisierung der State- oder Action-Value Function vorgenommen, sodass diese über einen ausreichend großen Zeithorizont die optimale Value Function approximiert. Daraus kann im Anschluss die optimale Policy abgeleitet werden. Mathematisch wird die Aktualisierungsregel für die State-Value Function wie folgt formuliert.

$$V(s_t) \leftarrow V(s_t) + \alpha[r_{t+1} + \gamma V(s_{t+1}) - V(s_t)] \quad \forall s \in S \quad (2.14)$$

Das α ist hierbei die sogenannte Lernrate. Diese kann Werte zwischen 0 und 1 annehmen und beschreibt die prozentuale Gewichtung der neuen Information für die Bewertung des aktuellen Zustandes.

Nachfolgend sollen zwei Verfahren aus dem Bereich des Temporal Difference Learning erläutert werden, welche dazu in der Lage sind ein **MDP** zu lösen.

Q-Learning

Die wahrscheinlich prominente Methode zum Trainieren von Reinforcement Learning Systemen ist das Q-Learning. Der Name leitet sich dabei aus der bereits erwähnten Q-Funktion $Q(s, a)$ ab, welche den erwarteten Nutzen Q einer Aktion a im Status s beschreibt. Beim Q-Learning muss der Agent zunächst mittels Sampling so viele Zustände durchlaufen wie möglich. Während er dies tut aktualisiert er nach jedem Schritt seinen Q-Value für das entsprechende State/Action Tupel und trägt diesen Wert in seine Q-Matrix ein, denn das Q-Learning nutzt ebenso wie die Value Iteration eine Tabelle zur Speicherung der gesammelten Werte. Diese Q-Matrix besitzt die Dimensionen Aktionen und Zustände.

Wenn der Zeithorizont zum Approximieren der Q-Values groß genug ist, dass heißt genug Sampling durchgeführt werden konnte und die Anzahl der möglichen Zustände klein genug, konvergiert die aus der erlernte Q-Matrix abgeleitete Policy zu einer optimalen Lösung des Problems.

Im Anschluss ist es dem Agenten demnach möglich, die entstandene Q-Matrix als Policy zu nutzen, indem die Aktion mit dem höchsten Q-Value im dazugehörigen Zustand ausgewählt wird.

Nachfolgend soll der Algorithmus des Q-Learning dargestellt werden:

Algorithm 2 : Q-Learning

Initialize $Q(s,a)$ arbitrarily (e.g. $Q(s,a) = 0$) $\forall s \in S \quad \forall a \in A$

for each episode **do**

 Initialize S

for each step of episode **do**

 Choose A from S using policy derived from Q (e.g. ϵ greedy)

 Take action A , observe R, S'

$Q(S, A) \leftarrow Q(S, A) + \alpha[R + \gamma \max_a Q(S', a) - Q(S, A)]$

$S \leftarrow S'$

end for

end for

Wie im oben gezeigten Algorithmus zu sehen, wird im Gegensatz zur Value Iteration, bei dem Update des Q-Values keine Übergangsfunktion genutzt, weil das Q-Learning ein Model-Free Verfahren ist. Außerdem handelt es sich hierbei um eine Action-value Function die verbessert werden soll. Des Weiteren wird nicht durch alle Zustände iteriert, sondern eine vorgegebene Anzahl an Episoden durchlaufen (Sampling). Falls die Anzahl der Episoden zu klein gewählt ist, wird der Lernerfolg ausbleiben, weil die Approximation an die optimale Action-Value Function noch nicht statt gefunden haben kann.

Allerdings liegt eine Gemeinsamkeit der beiden Verfahren im Bootstrapping, wie bereits am Beispiel der Value Iteration gezeigt, propagiert sich dabei der Reward durch alle besuchten Zustände durch. Das gleiche Verhalten ist auch beim Q-Learning zu beobachten.

Im oben dargestellten Algorithmus ist mit der Erwähnung der ϵ – *greedy* Policy eine bisher unbekannte Bezeichnung aufgetaucht. Diese Bezeichnung war bisher nicht notwendig, weil bei der Value Iteration alle Zustände betrachteten werden. Jedoch nutzen wir in der Klasse des Temporal Difference Learning das bereits erwähnte Sampling und dadurch sind dem Agenten immer nur die Zustände bekannt, die er bereits besucht hat.

Nachfolgend soll das Prinzip dahinter kurz erläutert werden, weil dieses Konzept in vielen Reinforcement Learning Methoden seine Anwendung findet.

Das Problem welches durch das ϵ – *greedy* Konzept gelöst werden soll nennt sich Exploration versus Exploitation Dilemma. Es beschreibt den tradeoff zwischen Ausbeutung der erlernten Strategie und Erkundung von neuen Strategien. Selbst wenn eine **KI** eine gute Policy gefunden hat, kann es eine bessere geben. Deshalb muss gelegentlich ein anderer Pfad ausprobiert werden.

Deshalb kann das ϵ Werte zwischen 0 und 1 annehmen und beschreibt dadurch die Wahrscheinlichkeit eine der beiden Möglichkeiten durchzuführen. Um mit Hilfe dieses Parameters entweder Exploitation oder Explorationen durchzuführen wird anschließend einfach eine Zahl zwischen 0 und 1 zufällig ausgewählt und geprüft ob diese über oder unter dem ϵ Schwellwert liegt. Danach kann die daraus resultierende Aktion durchgeführt werden. Vor allem zu Beginn eines Lernprozesses wird deshalb das ϵ auf einen hohen Wert gesetzt, weil zu

diesem Zeitpunkt viele Zustände ausprobiert werden sollen. Mit zunehmender Dauer kann das ϵ anschließend dekrementiert werden, bis es einen gewünschten Endwert erreicht. Das greedy im Namen des Konzepts steht für die Auswahl der Aktion mit der höchsten Bewertung (z.B. $\max Q(s,a)$) für den Fall, dass nicht zufällig gehandelt werden soll.

Zum Abschluss dieser Passage stellt sich die Frage, ob es mit Q-Learning möglich ist das Spiel „4 Gewinnt“ zu erlernen. Die Antwort lautet nein, weil die Schwachstelle bei diesem Verfahren die Q-Matrix ist. Wir haben durch das Sampling einen großen Vorteil gegenüber der Value Iteration gewonnen, aber die Tabellenform zur Speicherung der Werte ist immer noch ineffizient und würde große Zustandsräume nicht geeignet. Die Lösung dieser Problematik liegt im sogenannten Deep Q-Learning, welches im nachfolgenden Abschnitt beschrieben werden soll.

Deep Q-Learning

Dieses Verfahren ist eine Erweiterung des Q-Learnings und nutzt tiefe neuronale Netze um die Q-Matrix zu approximieren. Die Idee dafür stammt von der Firma DeepMind⁸, welche wie bereits erwähnt 2015 eine KI zur Lösung von 49 Atari Spielen entwickelte. Um diese bis dahin unmögliche Aufgabenstellung zu meistern entwickelte die Firma das Deep Q-Learning. Allein wegen dieses Verfahrens wurde die Firma DeepMind von Google für 500 Millionen Dollar gekauft. Seit 2015 bis zu der Veröffentlichung der ebenfalls angesprochenen AlphaGoZero Anwendung dominierte das Deep Q-Learning den Bereich des Reinforcement Learning als verbreitetes und performanteste Methode.

Der Kern von Deep Q-Learning liegt in der Substituierung der Q-Matrix durch ein tiefes neuronales Netz (siehe Abbildung 2.11). Wie dies genau abgebildet wird kann jedoch nicht beschrieben werden, weil die Art und Weise der Speicherung durch das neuronale Netz nicht zu entschlüsseln ist. Worüber jedoch eine Aussage getroffen werden kann ist die Performance. Wie das Anwendungsbeispiel für Deep Q-Learning schon zeigt, können durch diese Erweiterung auch Probleme mit großen Zustandsräumen gelöst werden. Dadurch bietet sich diese Erweiterung des Q-Learning zum Erlernen des Spiels „4 Gewinnt“ an.

⁸<https://deepmind.com/research/dqn/>

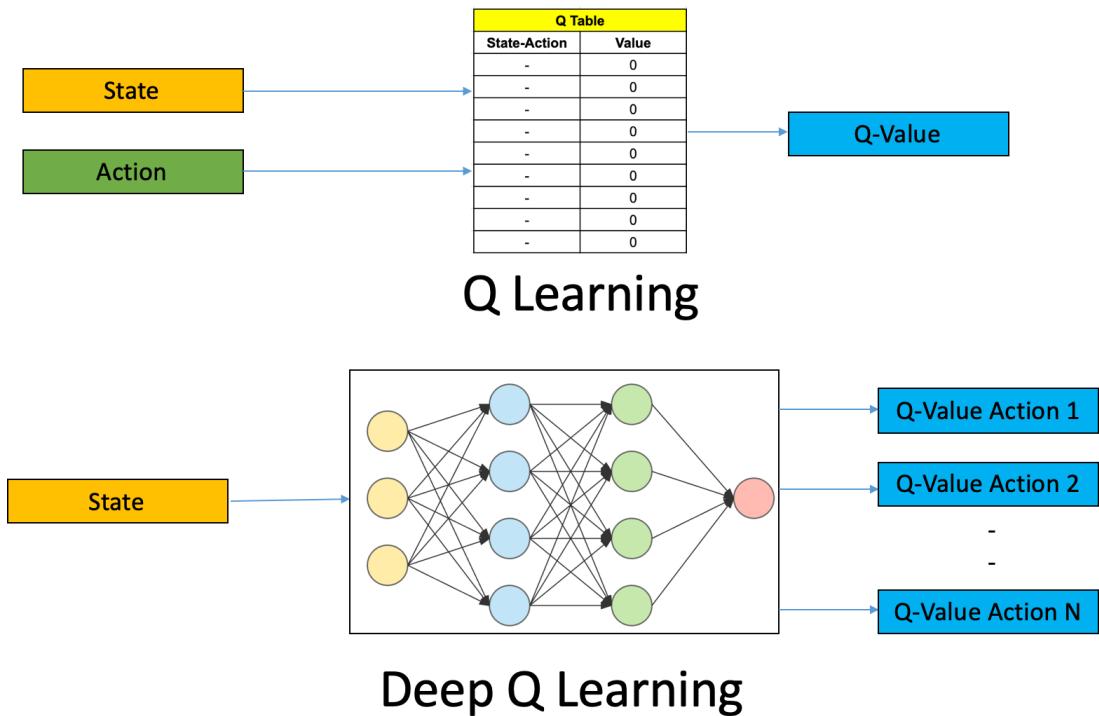


Abbildung 2.11: Vergleich Q-Learning und Deep Q-Learning [14]

Self-Play

In diesem Abschnitt soll ein weiteres Verfahren aus dem Bereich des Temporal Difference Learning vorgestellt werden. Dabei handelt es sich in diesem Fall um keinen so prominenten Algorithmus wie das Q-Learning. Dennoch soll diese Methode als zweite Auswahlmöglichkeit für die Lösung des „4 Gewinnt“ Problems erläutert werden. Die Inspiration für das anschließend vorgestellte Verfahren entspringt einer anderen wissenschaftlichen Ausarbeitung. [15] Der entsprechende Algorithmus aus dieser Veröffentlichung ist in Abbildung 2.12 dargestellt. Jedoch werden davon nicht alle Bestandteile gebraucht, sodass von diesem Algorithmus eine Abwandlung genutzt werden kann.

Nachfolgend soll die Funktionsweise dieses Verfahrens erläutert und anschließend die Frage beantwortet werden, ob sich dieses Verfahren ebenfalls für „4 Gewinnt“ eignet.

Algorithm 2 „Self-Play“: Incremental TD(λ) algorithm for board games. Input: Starting player p [= +1 (White) or -1 (Black)], initial position s_0 . Furthermore (partially trained) weights w_0 for function $f(w; s_t)$ approximating the value function $V(s_t)$. Reward function $r(s_t)$ from the game-playing environment. Output: Improved weights w_N .

```

function SELFPLAY( $p, s_0, w_0$ )
     $e_0 \leftarrow \nabla_w f(w_0; s_0)$                                  $\triangleright$  Initial eligibility traces
    for ( $t \leftarrow 0$  ;  $s_t \notin S_{Final}$  ;  $t \leftarrow t + 1$ ) do
         $V_{old} \leftarrow f(w_t, s_t)$                                  $\triangleright S_{Final}$ : set of all final states
        Choose randomly  $q \in [0, 1]$                                  $\triangleright$  Value function current state
        if ( $q < \epsilon$ ) then
            Choose a random  $s_{t+1}$                                  $\triangleright$  Explorative move (random move)
        else
            Choose a legal after-state  $s_{t+1}$  such that           $\triangleright$  Greedy move
             $p \cdot (r(s_{t+1}) + \gamma V(s_{t+1}))$ 
            is maximized.
        end if
         $V(s_{t+1}) \leftarrow \begin{cases} 0, & \text{if } s_{t+1} \in S_{Final} \\ f(w_t; s_{t+1}), & \text{otherwise} \end{cases}$            $\triangleright$  Value function next state
         $\delta_t \leftarrow \begin{cases} 0, & \text{if } q < \epsilon \wedge s_{t+1} \notin S_{Final} \\ r(s_{t+1}) + \gamma V(s_{t+1}) - V_{old}, & \text{else} \end{cases}$            $\triangleright$  Error signal
         $w_{t+1} \leftarrow w_t + \alpha \delta_t e_t$                                  $\triangleright$  Learning step
         $e_{t+1} \leftarrow \gamma \lambda e_t + \nabla_w f(w_{t+1}; s_{t+1})$            $\triangleright$  Eligibility traces
         $p \leftarrow (-p)$ 
    end for
    return  $w_N$ 
end function

```

Abbildung 2.12: Vergleich Q-Learning und Deep Q-Learning [15]

Dieser Algorithmus simuliert abwechselnd zwei Spieler die gegeneinander antreten $p(r(s_{t+1}) + \gamma V(s_{t+1}))$. Deshalb heißt das Verfahren auch „Self-Play“. Realisiert wird dies wie oben zu sehen über den Faktor p , welcher die Werte +1 und -1 annehmen kann. Dadurch wird für die beiden imaginären Spieler mathematisch versucht das Maximum, also die beste mögliche Aktion aus der Sicht des aktuellen Spielers zu wählen. Um hierbei den Spielerwechsel zu gewährleisten wird nach jedem Schritt der Faktor p invertiert $0 \leftarrow (-p)$.

Ein weiteres Kernelement ist das neuronale Netz, welches zur Speicherung der Value Function genutzt wird. Im oben dargestellten Algorithmus wird das neuronale Netz durch das ω für die Gewichte symbolisiert. Diese haben einen initialen Wert und werden mit jedem Schleifenauflauf optimiert. Deshalb ist das Rückgabeegebnis dieses Algorithmus ω_N , weil das neuronale Netz nach dem erfolgreichen Lernprozess als Policy genutzt werden kann. Dafür muss dem Netz lediglich der aktuelle Zustand übergeben werden und dieses gibt die Bewertung für alle möglichen Folgezustände zurück $f(\omega_t, s_t)$. Wenn davon immer der maximale Wert gewählt wird erhalten wir die optimale Strategie zur Lösung des Problems.

Des Weiteren bietet die Methode mit den oben dargestellten if/else Bedingungen ein Imple-

mentationsbeispiel für das bereits erwähnte ϵ – *greedy* Konzept. Mit Hilfe dieses Prinzips wird das Verhalten während des Smaplings gesteuert.

Der letzte unbekannte Parameter in diesem Algorithmus ist das δ , welches den ermittelten Fehler während eines Lernschrittes symbolisiert. Dafür wird der von der alten Value Function(in Form des neuronalen Netzes) ermittelte Wert mit dem neuen verglichen und mit der Lernrate multipliziert. Anschließend kann mit diesem Wert das neuronale Netz optimiert werden.

Was bei der oben gezeigten Darstellung fehlt ist die äußere Schleife mit der die gewählte Anzahl an Episoden durchlaufen wird. Innerhalb dieser Schleife würde wiederum die oben gezeigte Methode Self-Play aufgerufen werden.

Wie bereits erwähnt sind nicht alle Bestandteile des Algorithmus notwendig um ein **MDP** zu lösen. Deshalb kann der Parameter e für die Eligibility Traces in der Betrachtung des Algorithmus vernachlässigt werden.

Wie in der Kopfzeile des Algorithmus beschrieben, eignet sich dieser hervorragend für Brettspiele wie auch „4 Gewinnt“. Grund dafür ist zum einen, dass besagte Spiele episodische Probleme sind und zum anderen das der Spielerwechsel durch den Self-Play Algorithmus hervorragend simuliert werden kann. Dadurch ist kein gesondert implementierter Gegner notwendig und die Optimierung findet gleichzeitig zu jedem Schritt statt und nicht nur zu jedem zweiten. In Folge dessen ist die Lerngeschwindigkeit besser als beispielsweise beim Deep Q-Learning(bezogen auf das Spiel „4 Gewinnt“). Ein weiterer Umstand der dies begünstigt ist der Fakt, dass wir beim Self-Play Verfahren die State-Value Function optimieren und nicht die Action-Value Function wie beim Deep Q-Learning. Dadurch ist es notwendig das zu jedem Zustand die Werte für die dazugehörigen Aktionen gelernt werden müssen. Je mehr mögliche Aktionen vorhanden sind, umso mehr muss dadurch auch erlernt werden. Im Gegensatz dazu braucht der Self-Play Ansatz lediglich einen Wert für einen Zustand zu optimieren.

Aus diesen Gründen eignet sich dieses Verfahren besser für das Spiel „4 Gewinnt“ als das Deep Q-Learning.

2.2.4 Monte Carlo Methoden

Die Monte Carlo Methoden sollen in diesem Abschnitt erwähnt werden, weil diese eine prominente Verfahrensklasse im Reinforcement Learning darstellen. Der große Unterschied zwischen dem Temporal Difference Learning und den Monte Carlo Methoden liegt vor allem in den Umstand, dass hierbei ganze Episoden beim Sampling genutzt werden. Beim Temporal Difference Learning wird zu jedem Zeitschritt innerhalb einer Episode ein Update der vorhandenen Value Function vorgenommen. Diese Aktualisierung findet hier nur für eine komplette Episode statt.

Eine Folge daraus ist außerdem, dass kein Bootstrapping für Monte Carlo Methoden angewandt wird.

Da allerdings im Rahmen der Umsetzung des vorliegenden Projekts eine Methode aus dem Bereich des Temporal Difference Learning genutzt wird, beschränkt sich dieser Abschnitt auf die Erwähnung dieser Verfahrensklasse. Dies obwohl Verfahren wie AlphaGoZero aus dem Bereich der Monte Carlo Methoden bereits große Erfolge für ähnliche Probleme wie „4 Gewinnt“ erzielen. Jedoch ist das Ziel dieser Arbeit nicht die perfekt spielende **KI** zu entwickeln, sondern mit Hilfe von einem möglichst simplen Verfahren die grundlegende Funktionsweise von Reinforcement Learning darzustellen. Weitere Informationen zu dieser Entscheidung finden sich unter dem Punkt Evaluation der Entwicklungswerzeuge.

2.2.5 Taxonomie der Lösungsverfahren

Reinforcement Learning ist nicht durch einzelne Methoden oder Algorithmen definiert, sondern durch verschiedene Klassen von Problemen und deren Lösung. Beispielsweise bieten sich unterschiedliche Methoden für kontinuierliche (z.B. Kontrollprobleme) sowie diskrete Probleme (z.B. „4 Gewinnt“) an. Da jede Klasse von Problemen wiederum mehrere Verfahren zur Lösung dieser beinhaltet, ist die Anzahl an Methoden beim Reinforcement Learning groß. Deshalb soll in diesem Abschnitt die Taxonomie für Reinforcement Learning Methoden erläutert werden.

Grundlegend können die meisten Reinforcement Learning Verfahren mit Hilfe der folgenden Eigenschaften klassifiziert werden:

- Model-Free / Model-Based
- Value-Based / Policy-Based
- On-Policy / Off-Policy

Die Unterscheidung von Model-Free und Model-Based Methoden wurde bereits in den vorherigen Abschnitten erklärt.

Eine Value-Based Methode ist beispielsweise die Value Iteration, weil wir hierbei probieren die Value Function zu optimieren und erst anschließend die optimale Policy daraus ableiten. Im Gegensatz dazu optimiert eine Policy-Based Methode direkt die Policy anstelle der Value Function.

On-Policy Methoden nutzen eine bereits vorhandene Policy zum Auswählen von Aktionen während des Lernprozesses.

Off-Policy Methoden hingegen erforschen auch Zustände außerhalb ihrer bisherigen Policy indem sie gelegentlich zufällige Aktionen auswählen.

Allerdings gibt es neben dieser Klassifizierung noch weitere Möglichkeiten die einzelnen Verfahren zu gruppieren. Beispielsweise sind die Value Iteration und das Q-Learning, welche

bereits erläutert wurden, den tabellarischen Methoden zuzuordnen. Des Weiteren wurden ebenfalls die drei Klassen Dynamic Programming, Temporal Difference Learning und Monte Carlo Methoden angesprochen. Daran lässt sich erkennen, dass es viele unterschiedliche Gruppierungsvarianten gibt. Als Folge daraus existiert keine klare Struktur wie beispielsweise eine Baumhierarchie für Reinforcement Learning, sondern diese Struktur kann eher als ein stark verzweigtes Netz beschrieben werden, weil eine Methode vielen unterschiedlichen Klassen zugehörig sein kann.

2.3 4 Gewinnt

Das Spiel 4 Gewinnt ist ein Strategiespiel für zwei Personen. Es besteht aus einem Spielfeld mit typischerweise 6 Reihen und 7 Spalten sowie 42 Spielsteinen. Es gibt zwei Spieler, welche jeweils die Hälfte der Spielsteine in zwei unterschiedlichen Farben besitzen. Nachdem sich die Spieler darauf geeinigt haben wer beginnt, werden abwechselnd die Spielsteine von oben in eine beliebige freie Spalte geworfen. Ziel des Spiels ist es vier Steine seiner eigenen Farben horizontal, vertikal oder diagonal aneinander zu reihen. Daher auch der Name „4 Gewinnt“, wenn ein Spiel abgeschlossen wird, wechselt die Startreihenfolge.



Abbildung 2.13: 4 Gewinnt Spielbrett [16]

Das Spiel ist ein mathematisch gelöstes Problem⁹, sodass der zuerst setzende Spieler theoretisch immer gewinnen kann. Eine perfekte Künstliche Spielintelligenz würde somit gegen die optimale Verteidigung immer gewinnen, sofern sie den Eröffnungszug hat und den ersten Spielstein in die mittlere Spalte setzt. Bei beidseitig perfekten Spiel und einem Eröffnungszug links oder rechts neben der Mitte, würde es zu einem Remis kommen. Sobald der erste Stein in eine der übrigen Spalten gesetzt wird, verliert der startende Spieler gegen eine perfekte Verteidigung.

Allerdings ist die für ein perfektes Spielverhalten erforderliche Intelligenz recht komplex, sodass die meisten Menschen auch mit dem Vorteil des Spielstarts gegen einen perfekten Gegner verlieren würden. Damit gliedert sich 4 Gewinnt vom Komplexitätsgrad zwischen dem simplen Spiel Tic-Tac-Toe sowie dem schweren Spiel Schach ein. Dies gilt für die Regeln sowie die Spielintelligenz. Außerdem hat das Spiel einen hohen Bekanntheitsgrad in der Bevölkerung. Dadurch eignet es sich besonders als Lehrbeispiel für Reinforcement Learning.

⁹https://de.wikipedia.org/wiki/Vier_gewinnt

Kapitel 3

Anforderungsanalyse

In diesem Kapitel soll zunächst anhand von exemplarischen Personae, die vorgesehene Nutzungsweise der Anwendung sowie die entsprechende Zielgruppe dargestellt werden. Anschließend werden von diesen Personae die Anwendungsfälle sowie die Anforderungen abgeleitet, sodass die notwendigen und optionalen Funktionen sowie Bestandteile der zu entwickelnden Anwendung deutlich werden.

3.1 Persona



1

Max Knabe (Nutzer)

Max Knabe ist Schüler in der Sekundarstufe 2 eines Gymnasiums und wohnt zusammen mit seinen Eltern und seinem großen Bruder Tim am Rande von Berlin. Er ist 19 Jahre alt, geht gerne ins Kino und treibt oft Sport mit seinen Freunden. Sein größtes Hobby ist aber das

¹<https://pixabay.com/de/photos/unternehmer-start-start-up-mann-593358/>

Zusammen- sowie Umbauen von Computern. Immer wenn einer seiner Freunde Probleme mit seinem Computer hat, wird Max gefragt ob er sich nicht darum kümmern könne. Deshalb überlegt Max auch nach dem Abitur einen Studiengang im Bereich der Informatik zu wählen.

Er möchte aber nicht nur mit Theorie und Software zu tun haben, sondern eben auch mit der entsprechenden Hardware. Dafür bietet sich der Studiengang Telematik von der Technischen Hochschule Wildau aus dem Nachbarort perfekt an. Glücklicherweise hat sein Informatik Lehrer Herr Schmidt die Teilnahme an den NaWiTex Schülerlaboren organisiert. Im Rahmen dieser Veranstaltung werden praxisnah Inhalte aus dem Studium der Naturwissenschaften, der Informatik und insbesondere der Telematik vorgestellt. Somit kann Max den Studiengang kennenlernen und anschließend eine Entscheidung über seine Zukunft nach dem Abitur fällen.

Nach ersten Recherchen über die Hochschule und den Studiengang Telematik ist Max begeistert von den vielfältigen Themenfeldern, die im Rahmen des Studiums vermittelt werden. Außerdem sieht er die Chance, im späteren Verlauf des Studiums eigene Projekte zu verwirklichen und so praktische Erfahrungen vor dem Einstieg in das Berufsleben sammeln zu können. Ein Thema was ihn dabei besonders fasziniert hat, ist das maschinelle Lernen. Doch er hat noch nicht ganz verstanden, worin sich die Arten des maschinellen Lernens unterscheiden und wie ein solches Programm eine künstliche Intelligenz abbildet. Darum möchte er mehr über dieses Thema in Erfahrung bringen. Als Max erfährt, dass die NaWiTex Labore eine Veranstaltung zu diesem Thema durchführen, ist er begeistert und hofft seine Wissenslücken diesbezüglich schließen zu können.

Eine Woche später ist es dann soweit und Max sitzt zusammen mit seiner Informatikklassie in der Hochschule. Er lauscht gespannt den Ausführungen der Lehrkraft über künstliche Intelligenz, neuronale Netze sowie das maschinelle Lernen. Allerdings bleibt es nicht bei grauer Theorie, denn sie bekommen sogar einen praktischen Einblick in das Reinforcement Learning, ein Teilgebiet des maschinellen Lernens, welches Max schon bei seinen Recherchen faszinierend fand.

Dabei besteht die praktische Anwendung zu diesem Thema aus einer Weboberfläche, mit der man die Möglichkeit hat „4 Gewinnt“ gegen eine **KI** zu spielen. Hierbei konnte zuvor ein Schwierigkeitsgrad eingestellt werden, welcher sich unter anderem nach der Lerndauer der **KI** bemisst. Nach einigen Partien gegen die **KI** entschied Max sich dazu, zwei künstliche Intelligenzen unterschiedlicher Schwierigkeit gegeneinander spielen zu lassen und zu beobachten wie sich diese verhalten. Dabei konnte er unter dem „4 Gewinnt“ Spielfeld die Entscheidungen der **KI** anhand von Spaltenbewertungen in Form von reellen Zahlen nachvollziehen. Nachdem Max die Spielweise der **KI** analysiert hatte, wollte er erfahren, wie diese Anwendung im Hintergrund funktioniert beziehungsweise aufgebaut ist. Die nötigen Informationen dazu konnte er in der dazugehörigen Bachelorarbeit sowie der Weboberflä-

che finden. Dafür gab es in der Benutzeroberfläche einen Informationsbutton. Dieser hat ein Fenster geöffnet in dem die Anwendung erklärt wird und weitere Verlinkungen zu noch mehr Informationen bereitgestellt worden sind. Max fand es toll zu sehen, was für Arbeiten im Rahmen des Telematikstudiums entstehen können. Deshalb entschied er im Anschluss an die Veranstaltung sich für das Studium der Telematik in Wildau zu bewerben.



2

Julia Schüler (Betreuerin)

Julia Schüler ist Laboringenieurin für den Studiengang Telematik an der Technischen Hochschule Wildau. Sie ist 29 Jahre alt und lebt in Königs Wusterhausen in der Nähe der Hochschule. In ihrer Freizeit geht sie gerne klettern oder unternimmt etwas mit ihrer Familie. Außerdem absolvierte sie im Vorfeld den Bachelor- sowie Masterabschluss im Studiengang Telematik. Deshalb hat sie sich außerordentlich gefreut, als sie die Arbeitsstelle für den Studiengang angeboten bekommen hat.

Dabei sind ihre Aufgaben als Laboringenieurin hauptsächlich die Wartung und Instandhaltung aller technischen Einrichtungen sowie das betreuen verschiedener Veranstaltungen der Hochschule und des Studiengangs.

Zu einer solchen Veranstaltung zählen auch die NaWiTex Schülerlabore. Dabei stellt sie einer Schulkasse der zweiten Sekundarstufe ein bestimmtes Thema der Telematik vor. Eines dieser Themen ist das Maschinelle Lernen. Damit es dabei nicht nur bei einer theoretischen Lehrveranstaltung bleibt hat sie verschiedene Anwendungsbeispiele für die unterschiedlichen Arten des maschinellen Lernens vorbereitet. Eine solche Anwendung ist das Spiel „4 Gewinnt“ mit Hilfe von Reinforcement Learning, welches von einem Studenten als Bachelorarbeit entwickelt worden ist. Mit Hilfe dieser Applikation können die Schüler gegen eine Reinforcement Learning **KI** antreten.

²<https://pixabay.com/de/photos/frau-buch-lesen-bibliothek-young-2701154/>

Um die Nutzung der Anwendung vorzubereiten muss diese lediglich als Webapplikation auf einer bestimmten IP Adresse und einem bestimmten Port gestartet werden. Anschließend können sich alle Nutzer im gleichen Netzwerk mit der Applikation verbinden und diese nutzen. Dadurch hat Julia wenig Arbeitsaufwand mit der Vorbereitung der Anwendung. Außerdem muss sie nicht auf allen Clientrechnern eine Installation der Anwendung durchführen.



3

Eva Turing (Entwicklerin)

Eva Turing ist 23 Jahre alt und Studentin der Telematik an der Technischen Hochschule Wildau. Sie lebt in Berlin und muss deshalb immer mit der S-Bahn zur Hochschule pendeln. Damit hat sie allerdings kein Problem, denn den Studiengang Telematik gibt es nicht oft in Deutschland. In ihrer Freizeit geht Eva gerne Schwimmen und entwickelt Webseiten für Freunde oder Bekannte. Weil sie dafür nicht immer bezahlt wird, verdient sie als Werksstudent neben dem Studium etwas Geld an der Hochschule.

Im Rahmen ihrer Arbeitsstelle erledigt sie hierbei immer wieder kleinere Projekte für den Studiengang. Das können reine Softwareprojekte, aber auch Hardware Angelegenheiten, wie die Instandhaltung von Robotern sein. Außerdem hilft sie des öfteren bei Veranstaltungen der Hochschule. Beispiele hierfür wären der Hochschulinformationstag oder die NaWiTex Schülerlabore. Diesbezüglich hat sie die Laboringenieurin Julia Schüler angesprochen. Sie möchte, dass Eva einige Änderungen an einer Reinforcement Learning Applikation für die NaWiTex Schülerlabore durchführt.

Nachdem Eva sich eingehend mit der Anwendung auseinandergesetzt hat, war sie in der

³<https://pixabay.com/de/photos/m%C3%A4dchen-junge-student-sitzt-tisch-3718526/>

Lage, die an sie herangetragenen Aufgaben zu erledigen. Dazu zählten eine Anpassung der grafischen Benutzeroberfläche, das Hinzufügen eines neuen Spielalgorithmus sowie das Trainieren einer künstlichen Intelligenz mit Hilfe des Programms. Dabei haben ihr Schnittstellen zum Trainieren des Systems sowie der modulare Aufbau zum Hinzufügen und Ändern der Software die Arbeit erheblich erleichtert.

3.2 Anwendungsfälle

Die Anwendungsfälle beschreiben Interaktionen von verschiedenen Personengruppen mit der Anwendung. Diese sind abgeleitet aus den Personae. Mit Hilfe der Anwendungsfälle sollen anschließend Anforderungen an das zu entwickelnde System abstrahiert werden.

Nr.	Als (Akteur)	möchte ich (Beschreibung)	um (Ergebnis)
1	Entwickler	einzelne Anwendungsbestandteile getrennt voneinander austauschen können, ohne die komplette Applikation zu überarbeiten	die Anwendung flexibel erweitern oder aktualisieren zu können.
2	Entwickler	den Trainingsprozess der künstliche Intelligenz starten und mit Hilfe von Parametern steuern können	den Einfluss der Lernparameter zu testen und neue Spiellogiken zu hinterlegen.
3	Entwickler	weitere Spielalgorithmen einfügen können	die Auswahl der Anwendung zu vergrößern sowie Tests im Vergleich zu anderen Vorgehensweisen durchzuführen.
4	Betreiber	die Anwendung als zentralen Webserver starten	den Zugriff auf die Applikation von verschiedenen Endgeräten ohne vorherige Installation zu ermöglichen.
5	Betreiber	Informationen über aktuell auftretende Fehler der Applikation erhalten	beurteilen zu können, ob die Anwendung ordnungsgemäß funktioniert.
6	Nutzer	„4 Gewinnt“ gegen eine durch Reinforcement Learning entstandene Künstliche Intelligenz spielen	um das Niveau der Spielintelligenz einschätzen zu können.
7	Nutzer	zwei Künstliche Intelligenzen oder Algorithmen beim Spielen zusehen	um die Unterschiede festzustellen und das Spielverhalten verstehen zu können.

8	Nutzer	weitere Informationen über die Anwendung sowie das Themengebiet erhalten können	die komplexe Hintergründe der Thematik besser zu verstehen.
---	--------	---	---

Tabelle 3.1: Anwendungsfälle zur „4 Gewinnt“ Anwendung

3.3 Anforderungen

Mit den Anforderungen sollen die zukünftigen Bestandteile und Eigenschaften des Systems definiert werden. Dadurch kann im Anschluss das Konzept entworfen und der Erfolg des Projekts beurteilt werden.

Die jeweiligen Anforderungen sind in funktionale und nicht funktionale Anforderungen eingeteilt. Außerdem wird innerhalb der Anforderungen zwischen sogenannten „muss“ sowie „soll“ Kriterien unterschieden.

Funktionale Anforderungen beschreiben dabei die gewünschte Funktionalität eines Systems. Nicht funktionale Anforderungen hingegen beschreiben die gewünschte Qualität der einzelnen Systembestandteile. Des weiteren sind Muss Kriterien Anforderungen, die zwingend umgesetzt werden müssen und Soll Kriterien Anforderungen, welche für eine optimale Funktionsweise des Systems wünschenswert, aber nicht zwingend sind.

ID	Komponente	Kriterium	Anforderung	Art
F-001	Die Anwendung	muss	die Spielintelligenz mit Hilfe von Methoden des Reinforcement Learnings implementieren.	F
F-002	Die Anwendung	muss	modular in die Bestandteile Frontend und Backend aufgeteilt sein.	F
F-003	Die Anwendung	muss	um weitere Reinforcement Learning Algorithmen erweiterbar sein.	F
F-004	Die Anwendung	muss	eine in dem Browser Google Chrome (min. Version 75.0.3770.100) darstellbare grafische Benutzeroberfläche zur Interaktion anbieten (Weboberfläche)	F
F-005	Die Anwendung	muss	als Webserver lauffähig sein, um den Zugriff über ein Netzwerk zu ermöglichen.	F
F-006	Die Anwendung	muss	die Möglichkeit anbieten, gegen einen computergesteuerten Kontrahenten (KI) Spielen zu können.	F
F-007	Die Anwendung	muss	die Möglichkeit anbieten, zwei computergesteuerten Kontrahenten (KI) beim Spielen zusehen zu können.	F
F-008	Die Anwendung	muss	die Konfiguration des Schwierigkeitsgrades anbieten. Dafür müssen mindestens die Anzahl der Lernepisoden sowie der Erkundungsfaktor eingestellt werden können.	F
F-009	Die Anwendung	muss	die allgemeingültigen Regeln (siehe Abschnitt: Theoretische Grundlagen) des Spiels „4 Gewinnt“ implementieren.	F
F-010	Die Anwendung	soll	weitere Algorithmen zur Realisierung der Spiellogik anbieten, um diese mit der Reinforcement Learning KI vergleichen zu können.	F
F-011	Die Anwendung	soll	die Entscheidungen des computergesteuerten Spielers mit Hilfe von Zahlenwerten zu Spalten nachvollziehbar machen.	F
F-012	Die Anwendung	soll	zusätzliche Informationen zum Thema KI und Reinforcement Learning über einen Informationsbutton anbieten. Als Informationen können hierfür unter anderem die Daten aus der Bachelorarbeit verwendet werden.	F
F-013	Die Anwendung	soll	für andere „4 Gewinnt“ Anwendungen über eine Schnittstelle nutzbar sein. (Bsp: NAO spielt „4 Gewinnt“ aus dem Fachbereich Telematik der TH Wildau)	F
F-014	Die Anwendung	soll	die Möglichkeit anbieten, einen Trainingsprozess für eine KI durchzuführen und das Ergebnis zu speichern.	F
NF-001	Die Anwendung	soll	eine skalierbare Benutzeroberfläche anbieten.	NF

F = Funktionale Anforderung, NF = nicht Funktionale Anforderung

Tabelle 3.2: Anforderungen an die „4 Gewinnt“ Anwendung

Kapitel 4

Evaluation von Entwicklungswerkzeugen

In diesem Kapitel soll begründet werden, wieso sich der Entwickler für bestimmte Programmiersprachen, Frameworks und Algorithmen zur Umsetzung des Projekts entschieden hat. Anschließend wird das Konzept für das Projekt erläutert.

4.1 Auswahl der Programmier- /Skriptsprachen

Laut der Anforderungsanalyse soll das zu entwickelnde Programm als Webanwendung lauffähig sein (siehe F-005). In Folge dessen wird eine Weboberfläche zur Darstellung des Spiels benötigt. Für die Erstellung einer solchen Weboberfläche ist eine Kombination der Websprachen HTML, CSS und Javascript notwendig.

Ansatz 1:

Auf Grundlage dieser Notwendigkeit, wäre es möglich die Logik der Anwendung, also das Trainieren des neuronalen Netzes sowie die Spiellogik für „4-Gewinnt“ ebenfalls in Javascript zu implementieren, sodass die komplette Anwendung mit Hilfe von Javascript implementiert wird und über CSS und HTML dargestellt.

Ansatz 2:

Gleichzeitig kann im Rahmen der Webanwendung eine einfache Trennung von Darstellung und Logik gewährleistet werden, indem ein Server die Logik implementiert und Anfragen von etwaigen Clients entgegennimmt und beantwortet, sodass die Clients lediglich die Visualisierung der Informationen übernehmen. Allerdings muss für diesen Ansatz eine Kommunikationsschnittstelle für Server und Clients eingerichtet werden.

Abwägung:

Für den ersten Ansatz spricht, dass keine Kommunikationsschnittstelle notwendig ist und

keine Serverkapazitäten für die Anwendung bereit gestellt werden müssen. Allerdings existieren für Javascript wenig Frameworks zur Erstellung und zum Trainieren von neuronalen Netzen. Deshalb ist auch die Community für **KI** Anwendungen in Javascript kleiner und in Folge dessen existieren diesbezüglich weniger Beispielprojekte und Informationen im Internet.

Durch den zweiten Ansatz ist die Wahl der Programmiersprachen und den dazugehörigen Frameworks völlig offen. Der am meisten verbreitete Ansatz zur Programmierung von künstlicher Intelligenz ist die Sprache Python mit Frameworks wie Tensorflow, Keras oder PyTorch zur Modellierung der dazugehörigen neuronalen Netze. Ein weiterer Vorteil dieses Ansatzes ist die daraus resultierende Austauschbarkeit der einzelnen Anwendungsbestandteile (siehe Anforderung F-002). Zusätzlich ist Python Code oft besser zu lesen und kürzer als vergleichbarer Code in anderen Hochsprachen wie Java. Dadurch bietet es sich für die Umsetzung des Projekts an, denn ein wichtiger Faktor ist das Verständnis der Softwarestruktur für Dritte. Damit sind vor allem auch Schüler aus den NaWiTex Schülerlaboren gemeint, falls diese über die Veranstaltung hinausgehendes Interesse an der Anwendung zeigen.

Urteil:

Da die Vorteile der Umsetzung mit Python überwiegen und diese Sprache für den Bereich **KI** der de facto Standard ist, wird das Projekt im Backend mit Python umgesetzt und auf eine ausschließliche Implementierung der Anwendung mit Hilfe von Javascript verzichtet.

4.2 Auswahl von Frameworks

Frontend

Für die Erstellung der Benutzeroberfläche werden wie bereits erwähnt HTML, CSS und Javascript genutzt. Dabei dient HTML der Modellierung des Inhalts, CSS der optischen Aufbereitung und Javascript verwaltet die Nutzerinteraktionen.

Um den Aufwand der Erstellung eines Designs zu reduzieren, wurde sich für das CSS Framework Bootstrap entschieden. Dieses löst mit Hilfe eines Gridsystems außerdem das Problem der Darstellung auf Endgeräten von verschiedener Größe (Anforderung NF-001).

Die gleichen Funktionen bieten auch andere CSS Frameworks wie Materialize. Allerdings hat der Entwickler bereits diverse Projekte mit Bootstrap umgesetzt und beurteilt den Funktionsumfang für die zu entwickelnde Anwendung als ausreichend. Zusätzlich erhöht sich die Arbeitsgeschwindigkeit durch den bereits vorhanden Erfahrungsschatz.

Des Weiteren wird für die Erstellung der Benutzeroberfläche das Javascript Framework

D3.js eingesetzt, welches die Darstellung des Spielfeldes sowie die Verwaltung der im Hintergrund liegenden Daten regeln soll. Mit Hilfe dieses Frameworks können Veränderungen auf den Daten ausgeführt werden, welche dann anschließend automatisch dargestellt werden. Außerdem ist es möglich verschiedenste Animationen durchzuführen, weil das Framework auf SVG Basis funktioniert.

Es wäre ebenfalls möglich das Spielfeld mit HTML und CSS abzubilden. Allerdings wären dann keine Animationen möglich, der Zugriff von Javascript aus wäre aufwendiger und die Datenhaltung würde nicht innerhalb von Javascript stattfinden.

Backend

Im Backend wird vor allem für die Modellierung und Verwaltung eines neuronalen Netzes ein Framework benötigt. Zur Auswahl stehen dabei Tensorflow, Keras sowie PyTorch.

Bei TensorFlow handelt es sich um eine plattformunabhängige Programmzbibliothek unter Open-Source-Lizenz, welche von dem Unternehmen Google entwickelt wird. Jedoch ist Tensorflow eine Low Level API und deshalb ist entsprechender Code oft unübersichtlich, das Framework ist dadurch schwerer zu erlernen und der Code wird schnell umfangreich. Gleichzeitig bietet das Framework hierdurch eine potentiell höhere Performance und einen höheren Funktionsumfang.

Keras ist eine High Level API und baut auf Tensorflow auf. Dadurch können die genannten Probleme einer Low Level API umgangen werden und es können somit schnelle Ergebnisse mit Hilfe von wenig Code entstehen. Allerdings sind die eben genannten Vorteile von Tensorflow die Nachteile von Keras.

Ein weiteres Framework mit einem deutlich geringeren Marktanteil ist PyTorch. Dieses Framework ist ebenfalls eine High Level API und steht Keras in nichts nach. Trotzdem soll es für das vorliegende Projekt nicht eingesetzt werden, weil die Nutzung von Keras stärker verbreitet ist.

Dieser Faktor ist dem Entwickler besonders wichtig, weil dadurch potentiell mehr Informationen bezüglich dieses Frameworks abrufbar sind.

Aus den oben genannten Gründen wird deshalb das Framework Keras ausgewählt, weil dadurch schnelle Prototypen entwickelt werden können, die Nutzung intuitiver ist auf Grund einer höheren Abstraktionsebene und der Code im Ergebnis eine höhere Lesbarkeit aufweisen wird als bei der Nutzung von Tensorflow.

Kommunikation

Für die Kommunikation zwischen Frontend und Backend bieten sich HTTP-Requests an. Dabei können die Nachrichten mit Hilfe des JSON Formats verpackt und entpackt werden. Aus Javascript heraus ist das Versenden von solchen Requests an den Server ohne weiteres möglich.

Auf der Backendseite muss für das Verarbeiten dieser Nachrichten allerdings ein Webserver aufgesetzt werden. Für die Einrichtung dieses Servers kamen die zwei Python Microframeworks WebPy und Flask in Frage. Nach eingehender Recherche und dem Entwickeln von Minimalbeispielen wurde im Ergebnis Flask als Framework für die Umsetzung des Projekts ausgewählt.

Die Gründe dafür waren, dass die WebPy Dokumentation nicht sehr umfangreich ausgestaltet ist und dieses Framework aktuell nur Python in der Version 2 unterstützt. Dadurch würde eine Abhängigkeit auf eine bestimmte Version erzeugt werden, welche negative Folgen nach sich ziehen könnte. Im Gegensatz dazu ist Flask sehr ausführlich dokumentiert und unterstützt auch Python in der Version 3.

4.3 Auswahl des Algorithmus

Wie bereits in den theoretischen Grundlagen erwähnt, kommen für die Realisierung einer Reinforcement Learning **KI** für das Spiel „4 Gewinnt“ die zwei Algorithmen Deep Q-Learning und Self-Play in Frage. Die Verfahren stammen beide aus der Klasse des Temporal Difference Learning. Der Grund dafür ist, dass diese Klassen die Kernelemente der meisten Reinforcement Learning Anwendungen in sich vereint. Dazu zählen beispielsweise das Bootstrapping, das Sampling und die Unabhängigkeit von einem Modell der Umwelt. Unter anderem deshalb beschreibt Richard Sutton das Temporal Difference Learning in seinem Buch „Reinforcement Learning: An Introduction“, welches die umfangreichste Lektüre für das Themengebiet des Reinforcement Learning ist, hierbei wie folgt:

„If one had to identify one idea as central and novel to reinforcement learning, it would undoubtedly be temporal-difference (TD) learning.“ [12]

Auf Grundlage der Stellung von Temporal Difference Learning innerhalb des Reinforcement Learning soll ein Verfahren aus dieser Klasse zur Umsetzung des Projekts gewählt werden.

In den theoretischen Grundlagen wurden ebenfalls die Gründe genannt, welche dazu füh-

ren, dass der Self-Play Algorithmus mit weniger Spieldaten einen schnelleren Lernerfolg erzielen kann als das Deep Q-Learning. Dennoch wurde auf Grund der Prominenz des Deep Q-Learnings ein Prototyp mit diesem Algorithmus für das Spiel „4 Gewinnt“ entwickelt. Durch diese praktische Anwendung konnte erst die Erkenntnis erlangt werden, dass der Lernprozess mit Deep Q-Learning ineffizient ist. Hierzu mehr im Kapitel Umsetzung. Als Folge dieses Testlaufes mit dem Deep Q-Learning Algorithmus soll deshalb die Methode Self-Play zur Realisierung des Projekts genutzt werden.

Unter Berücksichtigung der Anforderungsanalyse und den gewählten Softwarepaketen soll nachfolgend das Konzept für die Anwendung erläutert werden. Anschließend wird der Entwicklungsverlauf, etwaige Probleme und das Resultat im Kapitel Implementierung besprochen.

Kapitel 5

Konzept

In Rahmen dieses Kapitels soll das Konzept für die softwaretechnische Umsetzung der Anwendung beschrieben werden. Dazu werden die modularen Bestandteile Frontend und Backend jeweils gesondert betrachtet (Anforderung F-002). Eine Darstellung der geplanten Anwendung auf der höchsten Abstraktionsebene bietet Abbildung 5.1.

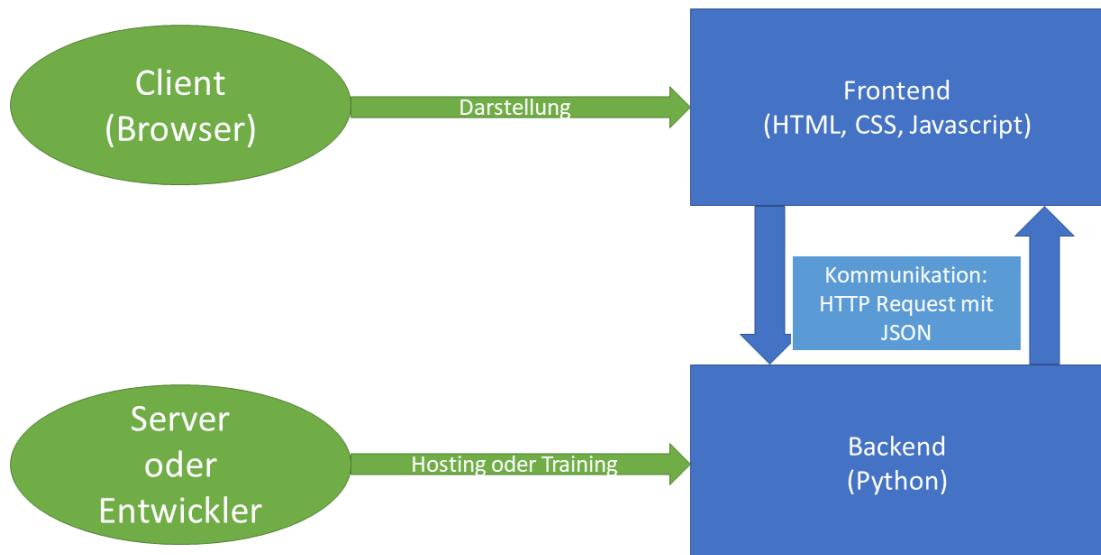


Abbildung 5.1: Modell der zu entwickelnden Anwendung

Die in der Abbildung 5.1 blau dargestellten Elemente bilden zusammen die zu entwickelnde Anwendung. Die grünen Elemente hingegen stellen den Zugriff auf die Anwendung durch externe Hard- und Software sowie Nutzer dar. Dabei dient der Browser (Anforderung F-004) als externe Software zur Darstellung des Frontend. Gleichzeitig zeigt dies auch, dass der Client keinen direkten Zugriff auf das Backend hat. Die Funktionen des Frontends bezogen auf

die Menüführung und Darstellung des Spielfeldes werden mittels Javascript realisiert. Die darüber hinaus gehenden Funktionalitäten, wie das Trainieren eines neuronalen Netzes oder das Berechnen von neuen Spielzügen durch eine **KI** werden im Backend durchgeführt. Um auf diese Anwendungslogik zuzugreifen, soll das Frontend mit Hilfe von HTTP Requests mit dem Backend kommunizieren. Das Senden und empfangen solcher Requests ist mit Javascript möglich. Das Backend wiederum muss für die Request einen Endpoint bereitstellen indem es einen Webserver auf einer dafür vorgesehenen IP startet. Dies soll mit Hilfe des Webframeworks Flask erfolgen. Der in Abbildung 5.1 dargestellte Server soll diesen Webserver entsprechend hosten.

Des Weiteren ist es für Entwickler möglich direkt auf das Backend zurückzutreten. Dafür bietet dieses ein Menü in Form einer Konsolenanwendung an. Über diesen Zugriff soll es möglich sein neuronale Netze mit Hilfe von einem Reinforcement Learning Verfahren zu trainieren, sich Statistiken dazu auszugeben und zusätzlich durch eine Konfigurationsdatei verschiedene Lerneinstellungen zu testen (Anforderung F-014).

5.1 Frontend

Die Benutzeroberfläche soll im Rahmen der Anforderungsanalyse verschiedene Eigenschaften erfüllen. Diese Eigenschaften sind in die Erstellung des User Interface mit eingeflossen. Die Abbildungen 5.2 und 5.3 zeigen den entsprechenden Entwurf dafür.

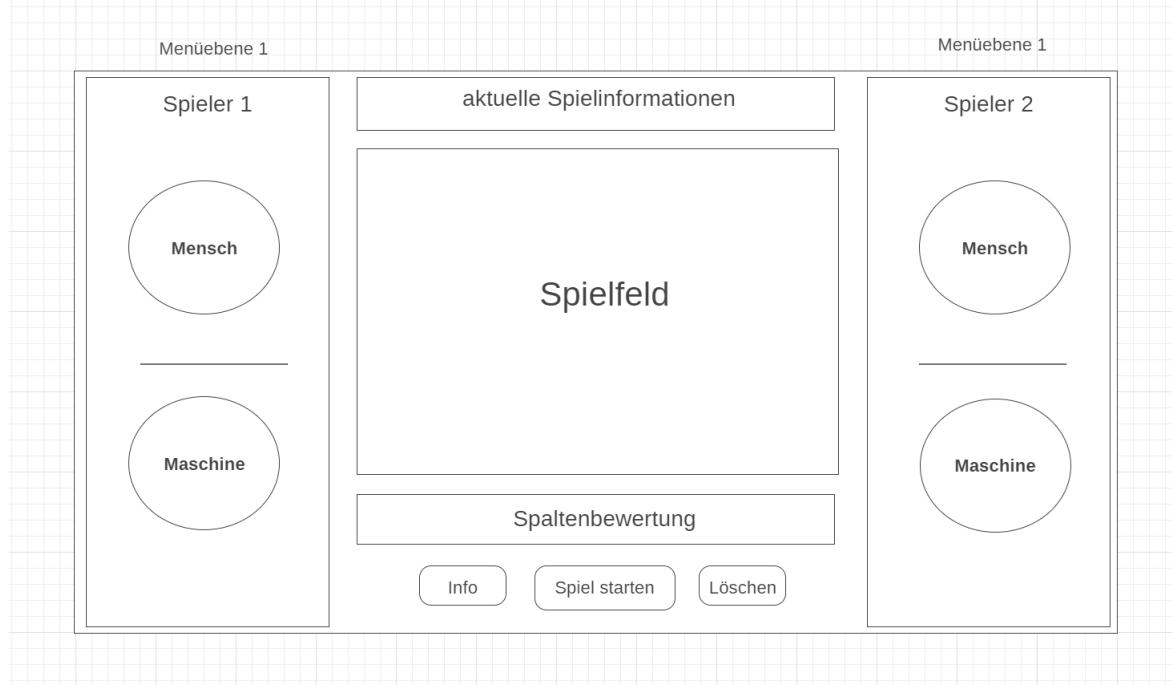


Abbildung 5.2: Mockup für das Frontend mit Menüebene 1

Für die Erstellung des Frontends soll wie bereits erwähnt das CSS Framework Bootstrap

verwendet werden. Dieses beinhaltet ein Gridsystem, welches für die Oberfläche angewandt werden soll. Ein Vorteil davon ist die implizite Umsetzung der Anforderung NF-001, weil die Bootstrap Klassen ein responsives Verhalten der einzelnen Zeilen und Spalten des Grids vorsehen. Des Weiteren soll von Bootstrap die Container Klasse verwendet werden, sodass die Anwendung nur circa zwei Drittel der insgesamt angezeigten Fläche einnehmen. Die dabei übrig bleibende Fläche soll als Hintergrund die Technischen Hochschule darstellen, weil die NaWiTex Schülerlabore eine Veranstaltung dieser sind. Innerhalb des Containers sollen sich die Elemente in ein Grid bestehend aus drei Spalten sowie drei Zeilen einordnen.

Die linke sowie rechte Spalte beinhaltet die Einstellungsmöglichkeiten für die beiden Spieler. Die mittlere Spalte beherbergt das Spielfeld, mit allen relevanten Spielinformationen sowie die Buttons zum Starten sowie Löschen eines Spiels. Zusätzlich existiert wie in Abbildung 5.2 zu sehen ein Info Button, welcher zur Erklärung des Spiels dienen soll und außerdem weitere Informationen bezüglich Reinforcement Learning sowie der Bachelorarbeit anbieten kann (Anforderung F-012).

Die erste Zeile beinhaltet die Anzeige von Spieler 1, Spieler 2 und den aktuellen Spielinformationen wie zum Beispiel dem Sieg oder der Niederlage eines Spielers. Die zweite Zeile beinhaltet die Einstellungsmöglichkeiten sowie das Spielfeld mit den Spaltenbewertungen (Anforderung F-011). Die dritte Zeile enthält den Bestätigungsbutton für Spieler 1 und Spieler 2 in der Menüebene 2 (siehe Abbildung 5.3) sowie die Buttons Info, Spiel starten und Löschen.

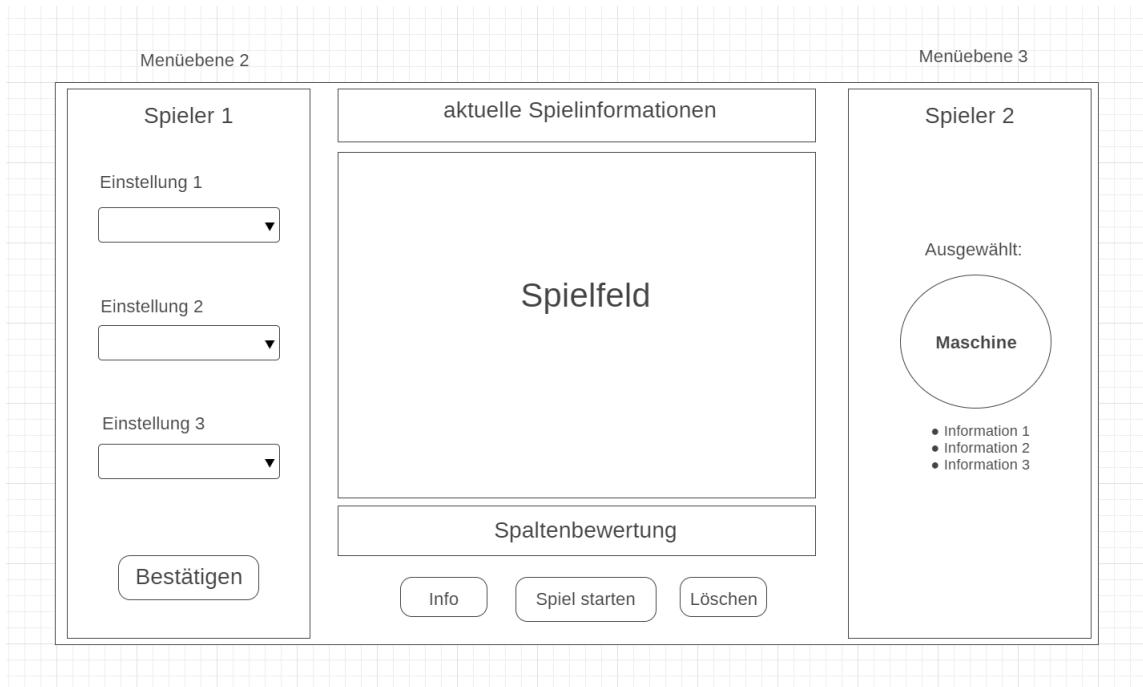


Abbildung 5.3: Mockup für das Frontend mit Menüebene 2 und 3

Wie in den Abbildungen 5.2 und 5.3 zu sehen existieren drei unterschiedliche Menüebenen im Entwurf. Die Menüebene 1 dient der Spielerauswahl, also ob ein Mensch gegen eine **KI** spielen möchte (Anforderung F-006) oder er zwei **KIs** beim Spielen zuschauen will (Anforderung F-007). Falls der Button Maschine betätigt wird öffnet sich die Menüebene 2, in der Einstellungen wie die Auswahl des Schwierigkeitsgrad vorgenommen werden können (Anforderung F-008). Durch die Bestätigung mit dem darunter liegenden Button, werden diese Einstellungen festgelegt und es öffnet sich die Menüebene 3, welche die ausgewählten Einstellungen anzeigt. Für den Fall das der Button mit der Aufschrift Mensch betätigt wird, öffnet sich sofort die Menüebene 3. Erst wenn Spieler 1 und Spieler 2 ihre Einstellungen vorgenommen haben kann das Spiel gestartet werden. In Folge dessen wird bei einem versuchten Spielstart ohne vorherige Einstellungen eine entsprechende Fehlermeldung ausgegeben. Der Löschen Button lädt die Seite neu und entfernt dadurch alle getätigten Einstellungen sowie den aktuellen Spielstand. Gleichzeitig bietet der Informationsbutton über das Öffnen eines Bootstrap Modals eine Anleitung zum Spiel sowie weiterführende Informationen zum Thema Künstliche Intelligenz an.

Das zum Spielstand dazugehörige Spielfeld soll mit Hilfe von D3.js entwickelt werden. Es wird dynamisch zum Aufruf der Seite geladen und erfordert eine gesonderter Behandlung bezüglich der Skalierbarkeit, weil es keine Bootstrap Klasse nutzt. Dadurch kann die Datenhaltung vollständig in Javascript ausgelagert werden, weil D3.js die Spieldaten mit der Anzeige verknüpft, dass bedeutet Änderung der Daten werden automatisch durch das Framework angezeigt. Ein weiterer Vorteil von D3.js ist die Möglichkeit Animationen einzubinden, sodass beispielsweise der Steineinwurf animiert werden könnte.

Für den Fall das eine **KI** als Gegner ausgewählt ist, wird bei jedem Zug dieser, eine Post Request per Javascript an den entsprechenden Endpoints des Backends gesendet. Im Folge dessen wird die entsprechende Berechnung zur Ermittlung des nächsten Zuges durchgeführt und als Antwort zurückgesandt. Das Frontend verwertet diese Antwort und führt die ausgewählte Aktion aus und stellt die entsprechenden Spaltenbewertungen dar.

5.2 Backend

Das Backend soll die in Abbildung 5.4 dargestellte Struktur erhalten. Die Main Klasse repräsentiert hierbei eine Python Konsolenanwendung die dem Nutzer drei verschiedene Funktionen zur Auswahl stellt.

Die beiden Hauptfunktionen sind hierbei das Starten eines Webservers und das Trainieren eines neuronalen Netzes mit Hilfe von Reinforcement Learning (Anforderung F-001). Da es möglich sein soll die Trainingsvorgänge mit Hilfe von Statistiken nachzuvollziehen, existiert außerdem die Möglichkeit sich mit der dritten Funktion eine solche anzeigen zu

lassen (in der Abbildung hellblau dargestellt).

Für das Training soll zusätzlich eine Konfigurationsdatei existieren, mit der die Lernparameter zentral eingestellt werden können. Auf der anderen Seite hostet die Serverfunktion einen Webserver mit den dazugehörigen Frontend Dateien und beantwortet anschließend ankommende Anfragen des Frontends (in der Abbildung grau dargestellt).

Des Weiteren nutzen die Bestandteile Training und Server die in grün dargestellten Funktionen. Die Klasse Agent realisiert eine Reinforcement Learning **KI** mit einem entsprechenden Lernalgorithmus. Die Klasse Policy soll weitere herkömmliche Algorithmen zur Lösung des „4 Gewinnt“ Spiels implementieren (Anforderung F-010). Beide Klassen nutzen zur Realisierung der jeweiligen Funktionalität die Hilfsklasse Board, welche zum einen das Environment darstellt und zum anderen diverse Funktionen zur Realisierung der Spiellogik bereitstellt.

Nachfolgend soll das Konzept für die einzelnen Bestandteile ausführlich erläutert werden.

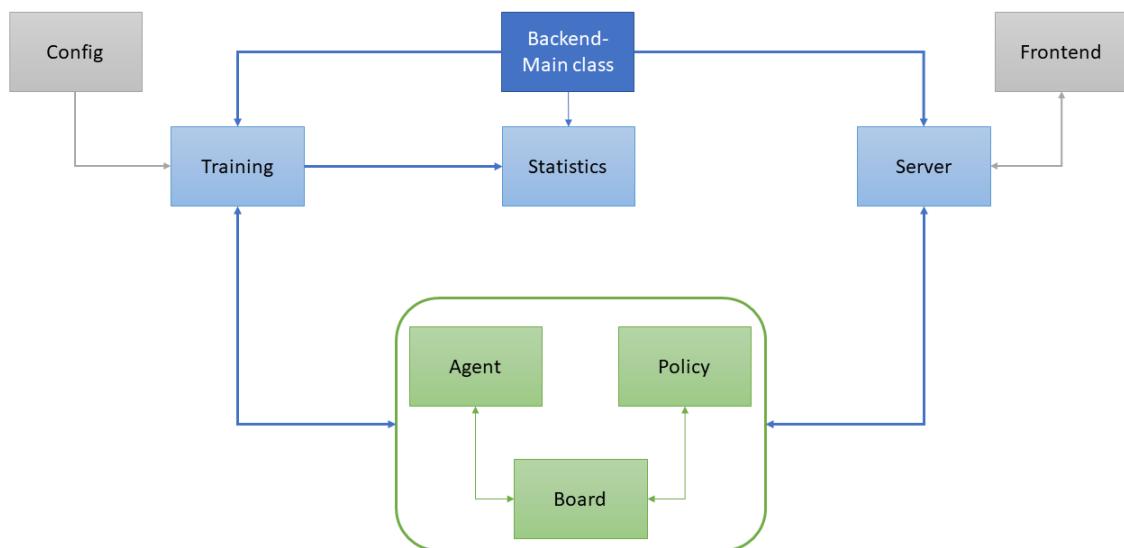


Abbildung 5.4: Bestandteile des Backends

Server

Die Serverfunktion soll innerhalb der Main Klasse implementiert werden, weil es ein Kernelement der späteren Anwendung darstellt. Das Training und die Statistik werden davon getrennt in extra Klassen implementiert. Dadurch ist eine Trennung der drei Funktionen gewährleistet.

Zum Hosten des Webservers soll das Framework Flask genutzt werden. Mit Hilfe des Frameworks kann der gehostete Server die Frontend Dateien unter der angegebenen Kombination aus IP und Port zur Verfügung stellen, sodass anschließend die Anwendung von Nutzern unter dieser Adresse aufgerufen werden kann (Anforderung F-005).

Damit eine Kommunikation des Frontends mit dem Backend ermöglicht wird ist es notwendig einen Endpoint zu initialisieren. Danach horcht der Server unter diesem Endpoint auf HTTP Post Anfragen und beantwortet diese entsprechend mit dem gewünschten Ergebnis. Solche Endpoints können ebenfalls mit dem Flask Framework realisiert werden.

Konzeptionell ist für die Post Anfrage vom Frontend vorgesehen, dass die aktuelle Spiel situation und die gewünschte Spielintelligenz zur Berechnung des nächsten Zuges an das Backend übermittelt wird. Die Antwort auf diese Anfrage besteht aus der berechneten Spalte für den nächsten Zug und die jeweiligen Spaltenbewertung wenn eine Reinforcement Learning **KI** als Spielintelligenz gewählt wird.

Durch diesen Aufbau ist keine stehende Verbindung notwendig, somit können auch andere Anwendungen die Spielintelligenz im Backend nutzen (Anforderung F-013).

Falls als Spielalgorithmus keine Reinforcement Learning **KI** gewählt wird berechnet das Backend den nächsten Zug mit Hilfe der Policy Klasse. Darin sind alle anderen Spielalgorithmen gesammelt, welche wiederum die Board Klasse nutzen um ihre jeweiligen Berechnungen durchzuführen. Beispiele für Funktionen der Board Klasse wären das Überprüfen ob ein Spieler gewonnenen hat oder das Berechnen der noch freien Spalten.

Für den Fall das die **KI** gewählt wird, nutzt das Backend die Klasse Agent, welche ein neuronales Netz implementiert mit der die Vorhersage für den nächsten Zug durchgeführt werden kann. Dabei nutzt diese Klasse ebenfalls Funktionen aus der Klasse Board.

Training

Die Trainingsklasse bildet den Rahmen für den Lernvorgang des Agenten. Hauptelement soll hierbei eine Schleife durch die angegebene Anzahl an Lernepisoden sein. Innerhalb eines Durchlaufs spielt der Agent gegen sich selbst eine Episode „4 Gewinnt“ und optimiert nach jedem Zug seine Fähigkeiten. Gleichzeitig können die dabei anfallenden Daten mit Hilfe der Statistik Klasse gespeichert werden. Das eigentliche maschinelle Lernen findet jedoch in der Agentklasse statt, welche nachfolgend detailliert erläutert wird.

Konfigurationsdatei

Außerdem sind für das Training verschiedene Lernparameter wichtig, einige Beispiele dafür wurden bereits in den theoretischen Grundlagen behandelt. Die besagten Parameter werden aus einer Konfigurationsdatei im JSON Format geladen und bestimmen die Rahmenbedingungen des Trainings. Zum Beispiel wird darüber gesteuert, wie viele Lernepisoden trainiert werden sollen oder ob ein bestehendes Training fortgesetzt werden soll oder nicht.

Der Vorteil einer solchen Konfigurationsdatei liegt in der zentralen Verwaltung unabhängig vom restlichen Anwendungscode. Dadurch muss sich der Entwickler der ein Training durchführen möchte nicht mit dem Code auseinandersetzen und kann mit Hilfe dieser Datei alle relevanten Lernparameter konfigurieren.

Statistik

Für die Statistik soll es zwei unterschiedliche Modi geben. Die Liveanzeige des aktuellen Trainings und die Speicherung des Lernvorgangs in einer Datei.

Die Liveüberwachung ist dafür gedacht, die Auswirkung von Änderungen an den Lernparametern über einen kurzen Trainingszeitraum zu beobachten. Dadurch kann bestimmt werden, ob getätigte Einstellungen wie gewünscht funktionieren.

Die Speicherung der Trainingsdaten in einer Datei soll vor allem für längerfristige Trainingsperioden genutzt werden. Außerdem soll anhand dieser Statistiken eine Aussage über den Lernerfolg getroffen werden können. Zusätzlich wäre es damit möglich zu beobachten, an welchen Punkten eine Verbesserung oder Verschlechterung des Lernprozesses stattfindet. Die gespeicherten Dateien sollen durch die Konsolenanwendung aufgerufen werden können. Durch den Aufruf einer entsprechenden Datei soll die Visualisierung mit Hilfe der matplotlib erfolgen.

Ob eine der beiden Statistikfunktionen eingeschaltet werden soll, wird durch die Konfigurationsdatei gesteuert. Falls die Funktionen aktiviert sind, wird die Aktualisierung der Liveanzeige sowie die zyklische Speicherung in einer Datei während des Trainingsprozesses durchgeführt.

Die entsprechenden Daten fallen während des Lernvorgangs an, dementsprechend wird die Speicherung der statistischen Daten durch die Trainingsklasse ausgelöst.

Agent

Die Klasse Agent soll im Rahmen dieses Konzepts den bereits erläuterten Self-Play Algorithmus als Verfahren des Reinforcement Learning einsetzen. Für die Implementierung des Algorithmus ist eine Self-Play Methode notwendig, welche von der Trainingsklasse aus aufgerufen wird. Diese Methode lässt die **KI** abwechselnd aus den beiden unterschiedlichen Spielerperspektiven gegen sich selbst spielen. Hierbei lernt die **KI** mit jedem Spielzug, indem das dazugehörige neuronale Netz jedes mal optimiert wird (siehe Self-Play Algorithmus in Abbildung 2.12).

Durch den Aufruf der Self-Play Methode soll insgesamt eine Episode des Spiels simuliert. Sobald diese beendet ist, können Statistiken zum abgeschlossenen Spiel gespeichert und im Anschluss von der Trainingsklasse eine neue Episode gestartet werden.

Außerdem soll für den Aufruf eines bereits trainierten neuronalen Netzes eine Methode zur Verfügung gestellt werden, die vom Server aufgerufen werden kann, sodass die Agentenklasse ein neuronale Netz lädt, welches zur Prädiktion des besten nächsten Spielzuges dient.

Da wie erwähnt für den Algorithmus ein neuronales Netz notwendig ist wird ein solches mit Hilfe des Keras Frameworks im Agenten implementiert. Des Weiteren werden Funktionen zum Laden sowie Speichern eines solches Netzes benötigt. Zusätzlich wird eine Methode für die Optimierung des Gewichte des neuronalen Netzes gebraucht. Näheres zur Wahl des entsprechenden Netzes in der nachfolgenden Sektion.

Neuronales Netz

Da das Spiel „4 Gewinnt“ mit $4.5 * 10^{12}$ [13] möglichen Zuständen bereits ein recht komplexes Spiel ist, sollte ein tiefes neuronales Netz zur Approximation der Value Function dienen. Wie im Abschnitt neuronale Netze erwähnt liegt ein tiefes neuronales Netz zwischen zwei und zehn versteckten Schichten. Demnach sollte die Anzahl in der Mitte davon liegen, also circa zwischen fünf und sieben Schichten.

Des Weiteren sollen für das Netz sogenannte Convolutional Layer genutzt werden. Diese Schichtenart wird vor allem für neuronale Netze genutzt die Bilder verarbeiten sollen. Der Grund hierfür ist, dass diese speziellen Schichten eine Matrix als Eingabedaten entgegennehmen können. Andere Schichten verlangen beispielsweise einen Vektor als Eingabe. Durch Eingabedaten aufgeteilt in Vektoren könnten jedoch Informationen verloren gehen, weshalb auf die Convolutinal Layer zurückgegriffen wird. Da das Spiel „4 Gewinnt“ ebenfalls als Matrix repräsentiert wird (6 x 7 Matrix) bietet sich diese Art eines neuronalen Netzes an. Ein Netz bestehend aus solchen Schichten wird als Convolutional Neural Network (CNN) bezeichnet. Zusätzlich wird bei einem CNN mit Filtern gearbeitet, diese werden über die Eingabematrix gelegt und stückweise verschoben. Dadurch sollen einzelne Details in den Eingabedaten für das neuronale Netz sichtbar gemacht werden. Die Größe der Filter soll für unseren Fall 4 x 4 betragen, weil es auch notwendig ist vier Steine für einen Sieg aneinander zu reihen.

Generell ist es jedoch schwer zu sagen, wie viele Schichten optimal sind und welche Arten von Schichten gewählt werden sollen, denn es gibt hierfür keine allgemeingültigen Regeln oder Vorgehensweisen. Die Grundlage für solche Entscheidungen könnte lediglich Intuition aus Erfahrung sein und diese besitzt der Entwickler zum Zeitpunkt der Entwicklung des Projekts noch nicht. Dementsprechend bleibt hierbei nur die Möglichkeit des „Trial and Error“ übrig.

Eine detaillierte Beschreibung des Lernalgorithmus und des fertigen Modells des neuronalen Netzes ist im Kapitel Implementierung wieder zu finden.

Policy

Die Policy Klasse soll zusätzliche Algorithmen als Abbildung einer Spielintelligenz für „4 Gewinnt“ anbieten (Anforderung F-003). Geplant sind hierfür der bekannte Minimax Algorithmus als klassische Lösung des Spiels, ein Algorithmus der einen Zug voraus schaut und den Gegenspieler entsprechend blockt oder seinen Siegeszug durchführt sowie ein völlig zufällig setzender Spieler.

Der Minimax Algorithmus soll zur Evaluation der Spielintelligenz dienen, ebenso wie der zufällig setzende Spieler. Der Zufall soll in frühen Lernstadien der **KI** anhand der prozentualen Siegquote feststellen, ob ein Lernerfolg vorliegt. Der Minimax Algorithmus hingegen, soll am Ende des Projekts als Maßstab für die Spielintelligenz der **KI** dienen. Der dritte Algorithmus kann die potenzielle Lücke zwischen den Lernstadien füllen beziehungsweise als Vergleichswert neben den anderen Verfahren dienen.

Die besagten Algorithmen benötigen zur Realisierung entsprechende Hilfsfunktionen. Diese sind in der Board Klasse zu finden und sollen nachfolgend erläutert werden.

Board

Die Board Klasse dient als Hilfestellung für die Policy sowie Agent Klasse. Dafür werden Konstanten bereitgestellt, welche definieren wie viel Reward es für einen Sieg, eine Niederlage oder ein Unentschieden gibt, welche Dimensionen das Spielfeld hat und welche Zahlen die jeweiligen Spielsteine repräsentieren.

Dadurch dient die Board Klasse nicht nur mit Hilfsfunktionen, sondern bildet in gewisser Weise auch das Environment für den Reinforcement Learning Ansatz ab.

Des Weiteren ist es notwendig das eine Funktion zum Simulieren von Spielzüge existiert. Ebenso muss geprüft werden können ob ein Spieler gewonnen hat und welche Spalten noch frei sind. Außerdem sind weitere Funktionen für den Minimax Algorithmus denkbar, welche bestimmte Ausschnitte des Spielfeldes aufgrund ihres Zustandes bewerten.

Kapitel 6

Implementierung

In diesem Kapitel soll das oben vorgestellte Konzept für die Anwendung umgesetzt werden. Dazu werden die einzelnen Bestandteile der Anwendung in der entsprechenden Entwicklungsreihenfolge erläutert.

- Erstellen von Frontend - Erstellung Webserver Backend - Kommunikationseinrichtung
- Minimax zum Testen - Erweiterung des Backend um Training + KI - Zunächst Q-Learning
- > schlechte Ergebnisse - dann Self-Play - dann Statistiken

Frontend

Im Rahmen der Implementierung wurde als erstes die Benutzeroberfläche entwickelt. Wie im Konzept beschrieben sollte hierfür als Grundelement die Container Klasse von Bootstrap verwendet werden. Diese nimmt ein Großteil der angezeigten Fläche ein und beinhaltet die komplette Darstellung der Anwendung. Zusätzlich soll die restliche Fläche als Hintergrundbild die Technische Hochschule Wildau erhalten. Wie in Abbildung 6.1 zu sehen wurde hierfür noch ein CSS Farbeffekt über das Hintergrundbild gelegt, damit die Anwendung im Zentrum hervorgehoben wird.

Als nächstes konnte das Gridsystem innerhalb des Containers angewendet werden, sodass die im Konzept beschriebene Aufteilung von drei Spalten und drei Zeilen implementiert wurde. Dadurch war das Grundgerüst für die Benutzeroberfläche fertiggestellt (siehe Abbildung 6.1).

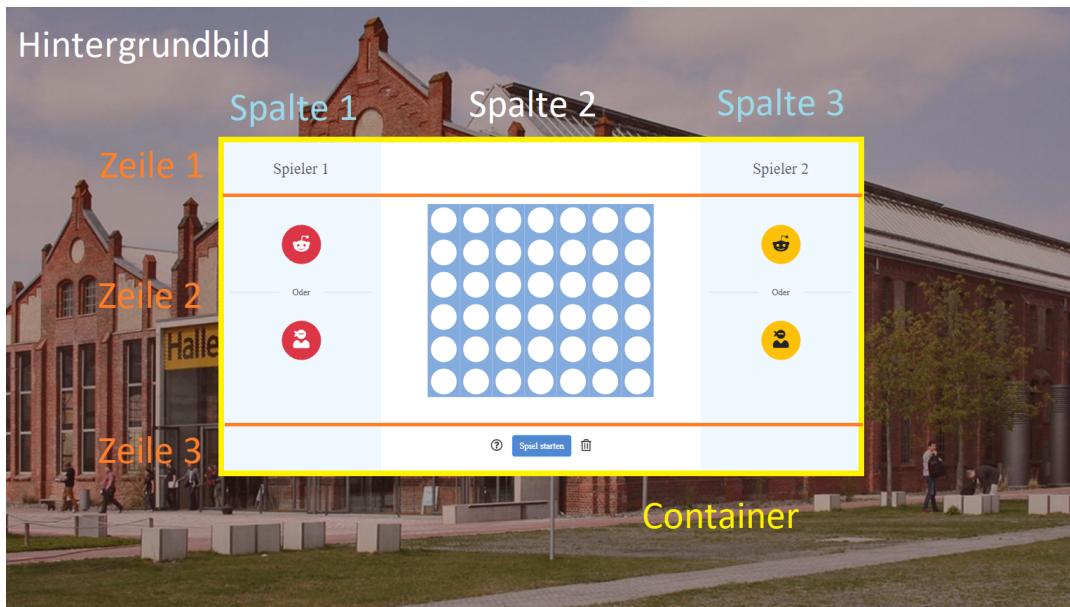


Abbildung 6.1: Grundstruktur Frontend

Im Anschluss wurden Hintergrundfarben für die einzelnen Spalten festgelegt, um die Übersicht für den Nutzer zu verbessern. Gleichzeitig solle diese Farbgestaltung ein Designelement sein, welches die Farben der Hochschule Wildau (Blau und Weiß) widerspiegelt.

Diese Farbstruktur findet sich ebenfalls im Spielfeld und bei der Wahl der verschiedenen Bedienelemente wieder.

Menüführung

Die Spalte 1 und Spalte 2 stellen die Menüführung für den Nutzer dar. Wie im Konzept beschrieben besteht die erste Menüebene aus der Auswahl der Spieler. Dazu muss je Spalte eine der beiden Optionen gewählt werden, welche durch die runden Buttons dargestellt wird. Die Optionen lauten hierbei Computer oder menschlicher Spieler. Aus Designgründen wurden hierfür Icons von Font Awesome verwendet. Zusätzlich wurde ein Tooltip implementiert, sodass dem Nutzer angezeigt wird, welcher Wert hinter einem Button steht, wenn er mit der Maus darüber fährt (für den Fall das die Symbole nicht eindeutig sind).

Die in Buttons zur Spielerauswahl haben hierbei die Farben rot und gelb, weil dies die jeweilige Spielsteinfarbe repräsentieren soll.

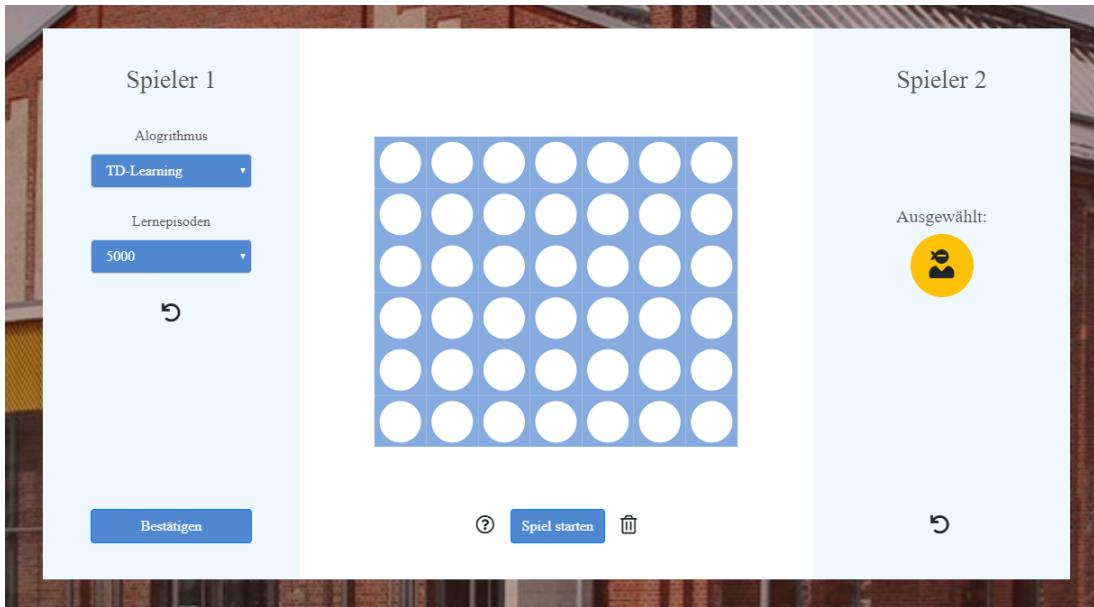


Abbildung 6.2: Menüebenen Frontend

Auf den Buttons zur Spielerauswahl liegen Event listener, welche weitere Menüebenen per Javascript öffnen. Durch die Auswahl eines menschlichen Spielers wird die Menüebene 3 geöffnet, welche in Abbildung 6.2 in der rechten Spalte zu sehen ist. Falls in der Spielerauswahl die dritte Menüebene erreicht wird, gelten die vollzogenen Einstellungen als festgelegt. Die zweite Menüebene ist in der linken Spalte der Abbildung zu sehen. Diese wird aufgerufen wenn ein computergesteuerter Spieler ausgewählt wird. Innerhalb dieser Ebene existiert die Möglichkeit verschiedenen Spielalgorithmen festzulegen. Zur Auswahl steht der Temporal Difference Learning Algorithmus, der Minimax Algorithmus sowie die Methode, welche lediglich einen Zug in voraus berechnet. Bis auf die letzt genannte Methode können zusätzlich Einstellungen für die Algorithmen festgelegt werden. Beim TD-Learning die Anzahl der Lernepisoden und beim Minimax Algorithmus die Suchtiefe (Baumverfahren). Die getätigten Einstellungen können mit dem Button in Zeile 3 der Menüebene 2 bestätigt werden. Zudem existiert in jeder Menüebene die Möglichkeit die vorherige Ebene aufzurufen. Dafür wurde das Pfeil Icon (Font Awesome) als Zurück Button implementiert. Dies wurde im Konzept nicht vorgesehen, allerdings ist diese Funktion benutzerfreundlich, weil die Anwendung bei einer falschen Eingabe nicht neu geladen werden muss.

In der Menüebene 3, welche in Abbildung 6.3 in der linken und rechten Spalte dargestellt ist, werden die ausgewählten Einstellungen angezeigt. Dieser Status wird wie bereits erwähnt in Javascript Variablen abgespeichert und diese können anschließend beim Spielstart ausgewertet werden. Auf Grundlage dessen wird der nächste Spielzug berechnet. Dafür werden die entsprechenden Einstellung an das Backend per Javascript HTTP Post Request versandt. Der dafür notwendige Webserver als Gegenstück wird im nächsten Abschnitt behandelt. Die Antwort des Servers ist die gewählte Spalte in der wiederum an der nächstmöglichen freien Stelle ein Stein per Klickevent gesetzt wird. Hierfür kann mit Javascript ein Klickevent auf

eine beliebige Position auf dem Bildschirm simuliert werden.

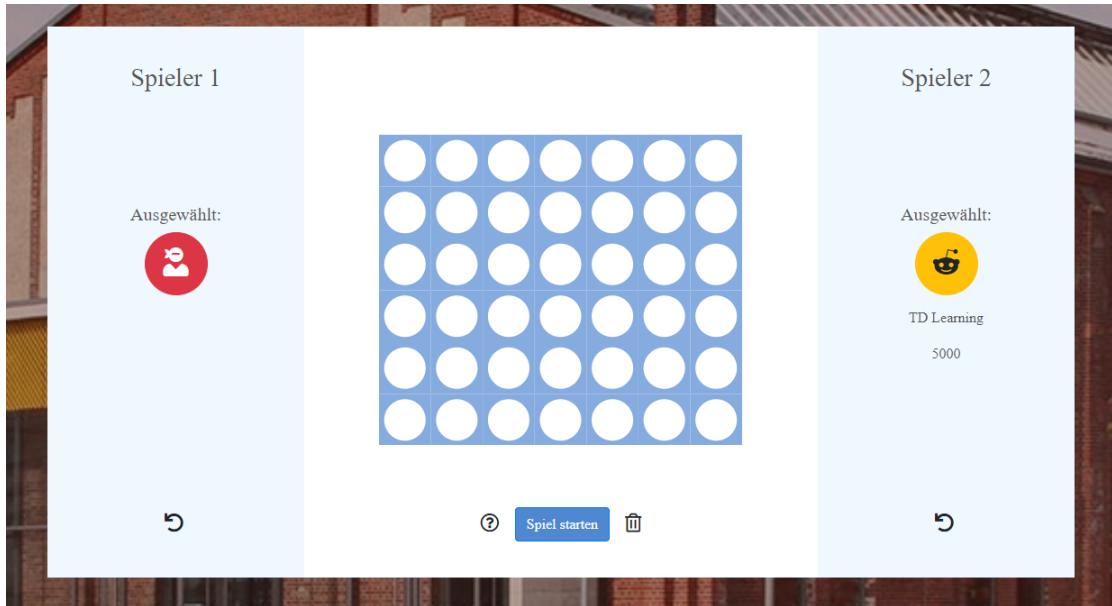


Abbildung 6.3: Menüebenen Frontend

Wenn wie in Abbildung 6.3 beide Spieler ihre Einstellungen vorgenommen haben, kann ein neues Spiel über den Button unter dem Spielfeld gestartet werden. Falls diese Voraussetzung allerdings nicht erfüllt ist, wird die in Abbildung 6.4 dargestellte Benachrichtigung angezeigt. Die Funktion wird über eine weitere globale Javascript Variable gelöst. Eine Idee die aus dieser Funktion entstand, war das Unterbrechen des aktuellen Spiels durch einen der beiden Zurück Buttons. Sobald diese aktiviert wurden ist das aktuelle Spiel unterbrochen und die Einstellungen können geändert werden. Dies ist möglich, weil die Kommunikation mit dem Server wie im Konzept stateless implementiert wird. Der Vorteil dadurch ist, dass der Nutzer während eines Spiels die Algorithmen wechseln kann und somit einen direkten Vergleich der Spielstärke bekommt.

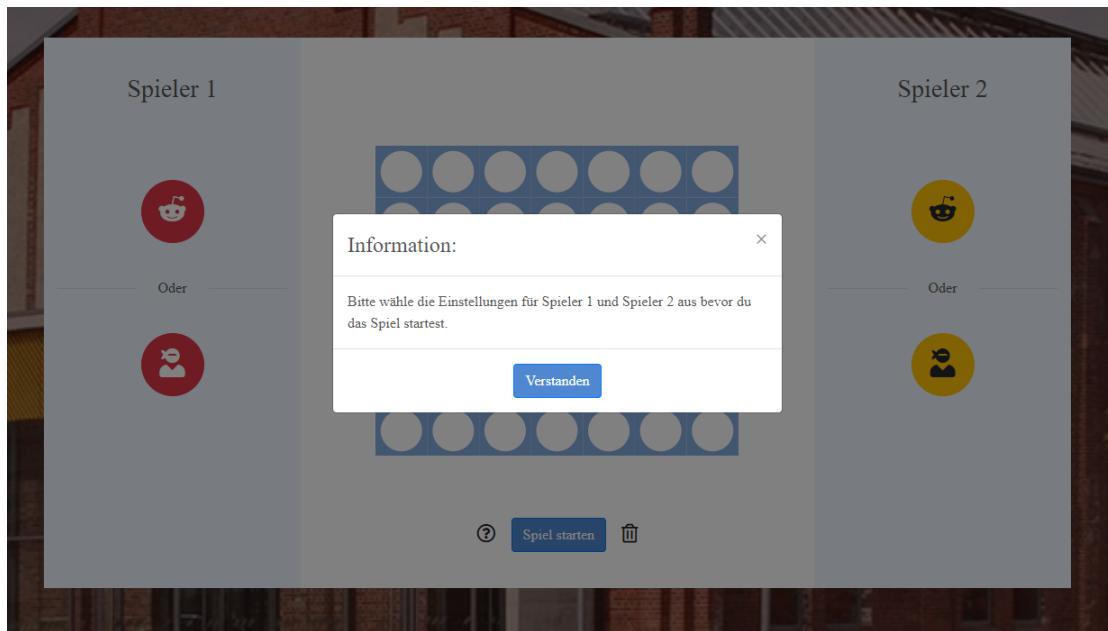


Abbildung 6.4: Modal Information Frontend

Wie im Konzept beschrieben dienen die neben dem Startknopf liegenden Symbole als Buttons um die Anwendung neu zu laden oder ein Informationsfenster aufzurufen. Sofern das Mülltonnen Symbol betätigt wird, aktualisiert sich die Seite durch ein Javascript Event. Falls das Fragezeichen Symbol betätigt wird, öffnet sich das Bootstrap Modal, welches in Abbildung 6.5 zu sehen ist.

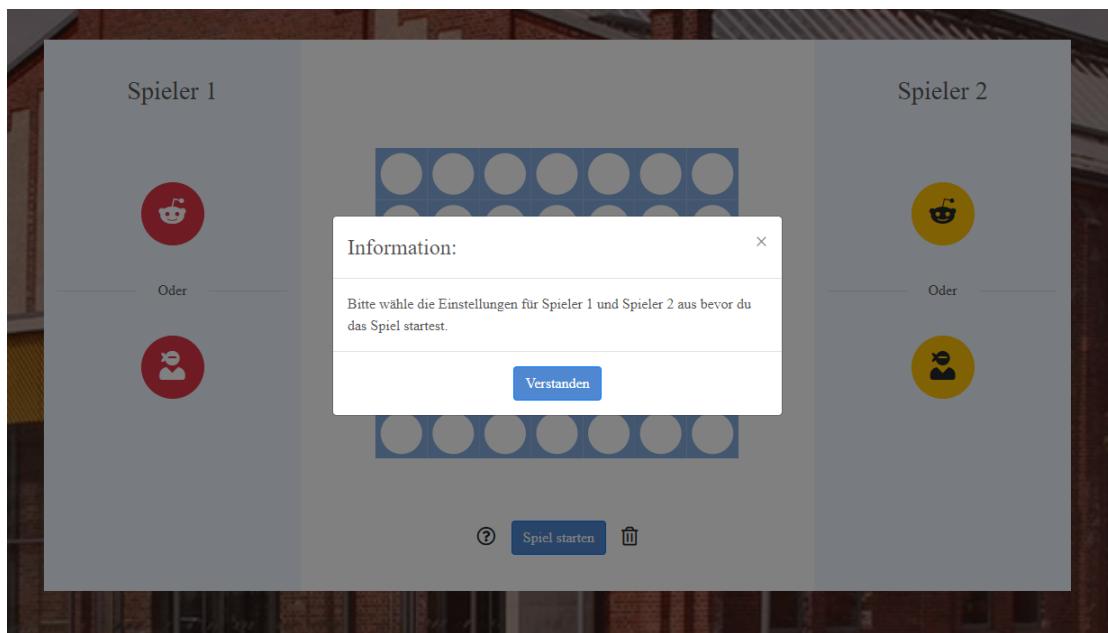


Abbildung 6.5: Menüebenen Frontend

Spielfeld

Des Weiteren befindet sich in der mittleren Spalte das Spielfeld, welches mit Hilfe des D3.js Frameworks verwirklicht wird. Dieses Framework erzeugt in HTML die entsprechenden SVG Elemente und erstellt eine Verknüpfung der Javascript Datenstrukturen mit den SVG Elementen. Dadurch führen anschließende Veränderungen der Daten zur Anpassung der verknüpften SVG Elemente. Dafür ist kein manueller Zugriff von Javascript auf die HTML Oberflächenelemente durch den Entwickler notwendig.

Demnach war es innerhalb von Javascript notwendig eine 6×7 Matrix als zweidimensionales Array zur Datenhaltung zu initialisieren. Anschließend konnte mit Hilfe dieser Matrix das Feld gezeichnet werden, dafür wird zeilenweise für jede Spalte ein Rechteck und ein Kreis Objekt angelegt. Der Kreis befindet sich zentriert innerhalb des Rechtecks und besitzt eine andere Farbe als das Rechteck. Dadurch entsteht bereits das in der Abbildung 6.1 zu sehende Spielfeld. Um die Interaktion mit dem Spielfeld zu ermöglichen besitzt jede Zelle der Spielfeldmatrix Eventlistener. Diese sind so initialisiert das zu Beginn des Spiels lediglich Aktionen in der untersten Zeile möglich sind. Falls eine Spielstein gesetzt wird, kann automatisch die entsprechende Zelle über diesem Spielstein freigeschaltet werden, sodass im nächsten Zug ein neuer Spielstein dort hingestellt werden kann. Außerdem wurde neben dem Click Event listener ein Hover Event eingefügt. Dadurch wurden die bespielbaren Zellen (bzw. Kreise) beim Herüberfahren in die Farbe des aktuellen Spielers gefärbt, sodass es für den Nutzer eindeutig ist wo er seinen Stein hinsetzen kann.

Da die Interaktionen mit dem Spielfeld wie eben beschrieben über Event listener funktionieren, ist es notwendig, dass auch die computergesteuerten Aktionen per Javascript ein Click Event an der entsprechenden Stelle ausführen.

Ein Problem mit dem Spielfeld liegt darin, dass dieses kein Bootstrap Element nutzt. Dadurch ist es nicht responsive und die automatische Skalierung muss selbstständig implementiert werden. Um dies zu Realisieren wird ein Eventlistener genutzt der ausgelöst wird, wenn das Anzeigefenster sich in der Größe verändert. Daraufhin kann anhand der Größe der mittleren Spalte die neue Größe des Spielfeldes prozentual berechnet werden. Da die Spalte durch eine Bootstrap Klasse skaliert wird, kann durch diese Vorgehensweise das Verhalten dieser Klasse implizit mit genutzt werden.

Ein weiteres Problem lag darin, dass die eigentlich geplanten Animationen nicht zu realisieren waren, weil bei SVG Elementen der sogenannte Z-Index nicht frei gesteuert werden kann. Dieser ist dafür verantwortlich auf welcher Ebene die Elemente in der dritten Dimension liegen. Dadurch hat das Spielfeld die Spielsteine bei einer Animation überlappt, sodass die Spielsteine nicht sichtbar waren.

Backend

Ein Beispiel für solch eine Datei bietet die nach stehende Abbildung einer JSON Datei.

```
args = {  
    'agent_args': {  
        'learning_rate': 0.01,  
        'gamma': 0.95,  
        'epsilon': 0.2,  
    },  
    'file_name': 'models/model.h5',  
    'resume_training': False,  
    'num_episodes': 5000,  
    'life_plot': False,  
    'create_stats': True  
}
```

Kapitel 7

Auswertung

7.1 Beurteilung des Projekterfolgs

7.2 Chancen von Reinforcement Learning

7.3 Persönliches Fazit

Glossar

Dunkelverarbeitung TODO.

NPV Der negative prädiktive Wert oder negative Vorhersagewert ist ein Parameter zur Einschätzung der Aussagekraft von medizinischen Testverfahren. Er gibt an, wie viele Personen, bei denen eine bestimmte Krankheit mittels eines Testverfahrens nicht festgestellt wurde, auch tatsächlich gesund sind.

PPV Der positive prädiktive Wert oder positive Vorhersagewert ist ein Parameter zur Einschätzung der Aussagekraft von medizinischen Testverfahren. Er gibt an, wie viele Personen, bei denen eine bestimmte Krankheit mittels eines Testverfahrens festgestellt wurde, auch tatsächlich krank sind.

Abkürzungsverzeichnis

KI künstliche Intelligenz.

MDP Markow Entscheidungsproblem.

Tabellenverzeichnis

3.1 Anwendungsfälle zur „4 Gewinnt“ Anwendung	48
3.2 Anforderungen an die „4 Gewinnt“ Anwendung	49

Abbildungsverzeichnis

2.1	Darstellung Teilbereiche von KI	9
2.2	Struktur eines Neurons	13
2.3	Sigmoid Funktion	14
2.4	Tiefes Neuronales Netz [6]	15
2.5	Übersicht der Teilbereiche vom Maschinellen Lernen [8]	17
2.6	Klassifikation von Punktwolken [9]	18
2.7	Lineare Regression von Punktwolke [9]	19
2.8	Dimensionsreduktion 3D zu 2D [10]	20
2.9	Reinforcement Learning Modell [12]	24
2.10	Ablauf des Value Iteration Algorithmus	31
2.11	Vergleich Q-Learning und Deep Q-Learning [14]	36
2.12	Vergleich Q-Learning und Deep Q-Learning [15]	37
2.13	4 Gewinnt Spielbrett [16]	41
5.1	Modell der zu entwickelnden Anwendung	55
5.2	Mockup für das Frontend mit Menüebene 1	56
5.3	Mockup für das Frontend mit Menüebene 2 und 3	57
5.4	Bestandteile des Backends	59
6.1	Grundstruktur Frontend	65
6.2	Menüebenen Frontend	66
6.3	Menüebenen Frontend	67
6.4	Modal Information Frontend	68
6.5	Menüebenen Frontend	68

Quellenverzeichnis

- [1] Christian Weber. *Computer spielt Go gegen sich selbst - und wird unschlagbar*. Hrsg. von Süddeutsche Zeitung. 19. Okt. 2017. URL: <https://www.sueddeutsche.de/digital/kuenstliche-intelligenz-champion-aus-dem-nichts-1.3713570> (besucht am 12.05.2019).
- [2] Campus Verlag, Hrsg. *Künstliche Intelligenz*. 2019. URL: <https://www.onpulson.de/lexikon/kuenstliche-intelligenz/> (besucht am 12.05.2019).
- [3] Radiological Society of North America (RSNA), Hrsg. *Automated Triaging of Adult Chest Radiographs with Deep Artificial Neural Networks*. 22. Jan. 2019. URL: <https://pubs.rsna.org/doi/10.1148/radiol.2018180921> (besucht am 04.06.2019).
- [4] Volker Tresp. *Basisfunktionen und Neuronale Netze*. Hrsg. von Ludwig-Maximilians-Universität München. 2009. URL: <http://www-old.dbs.ifi.lmu.de/Lehre/MaschLernen/SS2009/folien/BasisFunktionenNeuronaleNetze2009.pdf> (besucht am 21.07.2019).
- [5] Jürgen Schmidhuber. “Deep learning in neural networks: An overview”. In: *Neural Networks* 61 (2015), S. 85–117. ISSN: 0893-6080. DOI: <https://doi.org/10.1016/j.neunet.2014.09.003>. URL: <http://www.sciencedirect.com/science/article/pii/S0893608014002135>.
- [6] Claude. *Wie funktioniert ein neuronales Netz?* 8. Sep. 2018. URL: <https://scratchbook.ch/2018/09/wie-funktioniert-ein-neuronales-netz/> (besucht am 13.05.2019).
- [7] Fjodor Van Veen. *The Neural Network*. Hrsg. von The Asimov Institute. 14. Sep. 2016. URL: <http://www.asimovinstitute.org/neural-network-zoo/> (besucht am 13.05.2019).
- [8] Stephanie Fischer und Dr. Christian Winkler. *Machine Learning – Grundlagen Und Definition Für Anfänger Und Manager Erklärt*. Hrsg. von More Than Digital. 5. Jan. 2019. URL: <https://morethandigital.info/machine-learning-grundlagen-und-definition-fuer-anfaenger-und-manager-erklaert/> (besucht am 13.05.2019).
- [9] Benjamin Aunkofer. *Maschinelles Lernen: Klassifikation vs Regression*. Hrsg. von Data Science Blog. 20. Dez. 2017. URL: <https://data-science-blog.com/blog/>

- 2017/12/20/maschinelles-lernen-klassifikation-vs-regression/ (besucht am 13.05.2019).
- [10] Girish Mahajan. *Nonlinear dimensionality reduction*. Hrsg. von Alchetron. 5. Aug. 2018. URL: <https://alchetron.com/Nonlinear-dimensionality-reduction#-> (besucht am 15.05.2019).
 - [11] FIONA MACDONALD. *Google's New AI Has Already Learnt How to Crush Us at 49 Games*. Hrsg. von Science Alert. 26. Feb. 2015. URL: <https://www.sciencealert.com/world-first-a-computer-has-mastered-49-games-without-any-prior-knowledge> (besucht am 12.07.2019).
 - [12] Richard S. Sutton und Andrew G. Barto. *Reinforcement Learning: An Introduction*. Second. The MIT Press, 2018. URL: <http://incompleteideas.net/book/the-book-2nd.html>.
 - [13] Stefan Edelkamp und Peter Kissmann. "Symbolic Classification of General Two-Player Games". In: Sep. 2008, S. 185–192. doi: 10.1007/978-3-540-85845-4_23.
 - [14] ANKIT CHOUDHARY. *A Hands-On Introduction to Deep Q-Learning using OpenAI Gym in Python*. Hrsg. von Analytics Vidhya. 18. Apr. 2019. URL: <https://www.analyticsvidhya.com/blog/2019/04/introduction-deep-q-learning-python/> (besucht am 01.08.2019).
 - [15] Wolfgang Konen. *Reinforcement Learning for Board Games: The Temporal Difference Algorithm*. Hrsg. von Cologne University of Applied Sciences. 1. Juli 2015. URL: [file:///C:/Users/henni/Downloads/TR-TDgame_EN%20\(1\).pdf](file:///C:/Users/henni/Downloads/TR-TDgame_EN%20(1).pdf) (besucht am 31.07.2019).
 - [16] brettspiel-report, Hrsg. *Vier gewinnt*. 31. Juli 2007. URL: <http://www.brettspiele-report.de/vier-gewinnt/> (besucht am 13.05.2019).