FernUniversität in Hagen
Fakultät für Mathematik und Informatik

AIG
Artificial Intelligence Group

Anwendung von Reinforcement Learning Algorithmen basierend auf einem kooperativen Multi-Agenten Szenario

Masterarbeit

zur Erlangung des Grades eines Master of Science (M.Sc.)
im Studiengang Praktische Informatik (M.Sc.)

vorgelegt von
Julien Menth

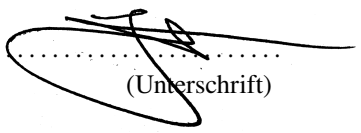
Erstgutachter:	Prof. Dr. Matthias Thimm Artificial Intelligence Group
Betreuer:	Prof. Dr. Matthias Thimm Artificial Intelligence Group

Erklärung

Ich erkläre, dass ich die Masterarbeit selbstständig und ohne unzulässige Inanspruchnahme Dritter verfasst habe. Ich habe dabei nur die angegebenen Quellen und Hilfsmittel verwendet und die aus diesen wörtlich oder sinngemäß entnommenen Stellen als solche kenntlich gemacht. Die Versicherung selbstständiger Arbeit gilt auch für enthaltene Zeichnungen, Skizzen oder graphische Darstellungen. Die Arbeit wurde bisher in gleicher oder ähnlicher Form weder derselben noch einer anderen Prüfungsbehörde vorgelegt und auch nicht veröffentlicht. Mit der Abgabe der elektronischen Fassung der endgültigen Version der Arbeit nehme ich zur Kenntnis, dass diese mit Hilfe eines Plagiatserkennungsdienstes auf enthaltene Plagiate geprüft werden kann und ausschließlich für Prüfungszwecke gespeichert wird.

	Ja	Nein
Mit der Einstellung dieser Arbeit in die Bibliothek bin ich einverstanden.	☒	☐
Der Veröffentlichung dieser Arbeit auf der Webseite des Lehrgebiets Künstliche Intelligenz stimme ich zu.	☒	☐
Der Text dieser Arbeit ist unter einer Creative Commons Lizenz (CC BY-SA 4.0) verfügbar.	☒	☐
Der Quellcode ist unter einer GNU General Public License (GPLv3) verfügbar.	☒	☐
Die erhobenen Daten sind unter einer Creative Commons Lizenz (CC BY-SA 4.0) verfügbar.	☒	☐

Köln,11.10.23



(Ort, Datum) (Unterschrift)

Zusammenfassung

Reinforcement Learning hat sich als wichtige Methode im Bereich der künstlichen Intelligenz etabliert, insbesondere wenn es darum geht, Agenten in komplexen Umgebungen zu trainieren. Diese Arbeit konzentriert sich dabei auf das Teilgebiet des Multi-Agent Reinforcement Learning. Grundlegend beschäftigt sich dieser Ansatz damit, mithilfe der KI mehrere Agenten in einem vorgegebenen Szenario zu trainieren, um eine optimale Lösung zu erarbeiten. Als Untersuchungsumgebung dient das Szenario „Pommerman“, welches an das von Nintendo veröffentlichte Spiel „Bomberman“ angelehnt ist. In dieser Umgebung wurden verschiedene Algorithmen implementiert und evaluiert. Dazu zählen die Algorithmen Value-Decomposition Networks (VDN), Mixed Q-Learning (QMIX) und Value-Decomposition Networks for Actor-Critic (VDAC). Mit diesen Algorithmen wurden Modelle trainiert, die rundenbasiert gegen regelbasierte Gegnerteams antraten. Zur Überwachung und Evaluierung des Trainings der Agenten wurde die Pommerman-Umgebung im Rahmen dieser Arbeit modifiziert. Zusätzlich wurde eine spezifische Belohnungsfunktion eingeführt, um den Agenten während des Trainings detailliert aufzuzeigen, welches Verhalten erwünscht ist und welches nicht erwünscht ist. Die Ergebnisse legen nahe, dass trotz aller Implementierungen und Anpassungen alle getesteten Algorithmen im Vergleich zu den regelbasierten Agententeams im Nachteil waren. Diese Erkenntnisse unterstützen die Annahme, dass Algorithmen, die auf Value Decomposition basieren, Schwierigkeiten bei der Anpassung in kooperativen Szenarien haben, in denen keine direkte Interaktion der Teammitglieder zur Lösungsfindung erforderlich ist.

Abstract

Reinforcement learning has established itself as an important method in the field of artificial intelligence, especially when it comes to training agents in complex environments. This thesis focuses on the subfield of multi-agent reinforcement learning. Basically, this approach deals with using AI to train multiple agents in a given scenario in order to come up with an optimal solution. The scenario „Pommerman“, which is based on the game „Bomberman“ published by Nintendo, serves as the study environment. In this environment different algorithms were implemented and evaluated. These include the algorithms Value-Decomposition Networks (VDN), Mixed Q-Learning (QMIX), and Value-Decomposition Networks for Actor-Critic (VDAC). These algorithms were used to train models that competed against rule-based opponent teams in a turn-based manner. To monitor and evaluate the training of the agents, the Pommerman environment was modified as part of this work. In addition, a specific reward function was introduced to detail to the agents during training which behaviors are desirable and which are not. The results suggest that despite all implementations and adaptations, all tested algorithms were at a disadvantage compared to the rule-based agent teams. These findings support the assumption that algorithms based on value decomposition have difficulty adapting in cooperative scenarios where direct interaction among team members is not required to find a solution.

Danksagung

Zuallererst möchte ich meinem Betreuer, Prof. Dr. Matthias Thimm, für seine unermüdliche Unterstützung, seine wertvollen Anregungen und sein konstruktives Feedback während des gesamten Forschungsprozesses danken. Seine Geduld und Expertise waren entscheidend für das Gelingen dieser Arbeit. Meiner Familie, insbesondere meiner Lebensgefährtin, danke ich von Herzen. Ihre anhaltende Ermutigung, ihr Glaube an mich und ihre kontinuierliche Bestärkung waren meine ständigen Begleiter und die stärkste Motivation.

Abbildungsverzeichnis

Tabellenverzeichnis

Inhaltsverzeichnis

1 Einführung

Im ersten Abschnitt dieser Arbeit wird der Leser mit den Beweggründen und der Motivation hinter dieser Masterarbeit vertraut gemacht. Dabei wird zum zentralen Thema und zur zugrundeliegenden Forschungsfrage hingeführt. Das Kapitel schließt mit einer Einordnung der aktuellen Forschungslandschaft zum Themenschwerpunkt ab.

1.1 Motivation und Problemstellung

Viele Experten sowie Studien deuten darauf hin, dass der aktuelle technische Stand der künstlichen Intelligenz (im Folgenden als KI abgekürzt) in den letzten Jahren signifikante Fortschritte erlebt hat. Dies ist mitunter auf die fortschreitende Entwicklung von Machine-Learning-Techniken und leistungsfähigeren Hardware-Plattformen zurückzuführen [20]. Ein aktuelles Beispiel hierfür sind die sogenannten Deep-Learning -Technologien, die es ermöglichen, komplexe Aufgabenstellungen unter Zuhilfenahme von neuronalen Netzwerken zu lösen [34]. Ein aktuell in der Öffentlichkeit viel publiziertes Beispiel ist die Fähigkeit von KI-Systemen, menschliche Sprache zu verstehen und darauf reagieren zu können. Dies ist mitunter durch die Entwicklung der Technologie Natural-Language-Processing ermöglicht worden. Ein aktueller Vertreter dieser Technologie ist das von OpenAI veröffentlichte ChatGPT3-Modell. Dieses bietet laut den Entwicklern verschiedene Dialogoptionen und Funktionalitäten, um mit der KI zu interagieren. Hierzu zählen laut OpenAI unter anderem „Fragen zu beantworten, Fehler einzugestehen, falsche Prämissen infrage zu stellen und unangemessene Anträge abzulehnen“ [8]. Um dieses Model anzulernen, wurde die Methode des Reinforcement Learning verwendet [8]. Diese wird im Verlauf der Ausarbeitung im Fokus stehen, wenngleich auch unter anderen Bedingungen und Szenarien. Diese, sowie allgemeine Fortschritte innerhalb des Forschungsgebietes eröffnen neben den vorher genannten auch zahlreiche weitere Anwendungsmöglichkeiten für KI.

Hierzu zählen diverse Möglichkeiten, beispielsweise die Automatisierung von Geschäftsprozessen, das Entwickeln von Assistenzsystemen für den menschlichen Gebrauch sowie die Navigation von Fahrzeugen im Sinne des autonomen Fahrens [6]. Dabei hat insbesondere die durch Sutton und Barto veröffentliche Methodik des Reinforcement Learnings (z.dt. Bestärkendes Lernen) in den letzten Jahren an Relevanz gewonnen. Bei der Methode geht es vereinfacht gesagt um einen kontinuierlichen Lernprozess, der dem Ablauf des menschlichen Lernens nachempfunden wurde. Dieser oft als Training bezeichnete Ablauf soll Stück für Stück einer Entität das gewünschte Verhalten mittels Belohnung und Bestrafung beibringen. Dies ist primär von der menschlichen Evolution inspiriert, denn auch Menschen lernen durch kontinuierliches Ausprobieren in der Kindheit die wichtigsten Verhaltensbausteine zum Überleben [37]. Dabei hat das Konzept des Reinforcement Learnings auch abseits von ChatGPT einige Herausforderungen meistern können, beispielsweise das Spielen von Schach auf Weltklasseniveau [24]. Ein Problem des Reinforcement Learnings, das hauptsächlich bei der Anwendung von Einzel-Agenten-Methoden auftritt ist, dass diese oft schwer auf reale Szenarien übertragbar sind [39]. Solche Methoden haben gemein, dass in der Regel nur eine einzelne Entität trainiert wird. Es ist herausfordernd aus den in definierten Umgebungen stattfindenden Testaufbauten relevante Ergebnisse für die Praxis zu extrahieren. Eine der

Begründungen ist es, dass in der Regel mehr als ein Akteur für das erfolgreiche Ausführen einer Tätigkeit notwendig ist [11][7]. Vorstellbare Situationen wären, wenn Teamleistungen zu erbringen sind, oder die Umgebung eine Koordination von verschiedenen Akteuren notwendig macht. Weshalb sich im Rahmen dieser Arbeit mit Multi-Agent Reinforcement Learning, abgekürzt MARL, auseinandergesetzt wurde.

Bei MARL werden ähnliche Verfahren eingesetzt, die auch aus dem Reinforcement Learning bekannt sind. Allerdings unter Berücksichtigung mehrerer parallel agierender und auch zu trainierender Entitäten. Eines der denkbaren Probleme bei dem Einsatz von MARL ist es, dass jeder Agent unabhängig des bereits erlangten Wissens der anderen Entitäten, Erfahrungen sammelt und lernt. Demzufolge hat jeder Agent einen unterschiedlichen Wissensstand, einige Agenten sind unter Umständen bereits weiter vorangeschritten bei der Erlernung gewünschter Verhaltensweisen, wohingegen andere Agenten gewisse Situationen noch nicht durchlaufen haben und deshalb Verhaltensmuster noch nicht erlernt haben [40]. Zudem kann es herausfordernd sein, viele eigenständig lernende und agierende Entitäten zu überwachen. Mitunter haben sich deshalb innerhalb des Forschungsgebiets von MARL verschiedenste Methodiken etabliert. Eines dieser Methoden ist das sogenannte „centralized training and dezentalized execution“ im Folgenden mit CTDE abgekürzt (dt. zentrales Training und dezentrales Ausführen). Diese Methode hat das Ziel, dass Training der Agenten und die Ausführung voneinander getrennt durchzuführen. Der Ansatz zentralisiert den Trainingsprozess, so ist es zum Beispiel möglich unterschiedliche Versionen von trainierenden Agenten gegeneinander antreten zu lassen. Hierdurch kann die KI von unterschiedlichen Versionen ihrer selbst lernen. Das kann einerseits helfen den Fortschritt durch Versionierung der jeweiligen Modelle zu behalten und vereinfacht andererseits die Überwachung. Darüber hinaus ermöglicht der Einsatz von CTDE, aufgrund der Teilung von Training und Ausführung, der KI während des Trainingsprozesses mehr Informationen zukommen zu lassen. Die Teilung verfolgt unter anderem das Ziel, das Training der KI zu beschleunigen. Dabei unterstützen die Mehrinformationen dabei, dass seltene Situationen aus realen Szenarien in einem kontrollierten Trainingsszenario häufiger berücksichtigt werden können [21].

Durch den Einsatz beziehungsweise die Kombination von Techniken der künstlichen Intelligenz wurden die bis dato besten Ergebnisse in verschiedensten Disziplinen erreicht, weshalb das Multi-Agent Reinforcement Learning Konzept um Techniken aus dem Bereich des Deep-Learnings erweitert wurde [24][27]. Das so zum Einsatz kommende Deep-Reinforcement-Learning setzt unter anderem auf die Methodik des Q-Learnings [43]. Diese und weitere zum Verständnis relevante Informationen werden in nachfolgenden Grundlagenkapiteln detailliert erläutert. Die Kombination dieser Ansätze wird Multi-Agent Deep-Reinforcement-Learning genannt und im nachfolgenden mit MADRL abgekürzt. Das Prinzip ist ähnlich dem regulären MARL, mit dem Unterschied, dass Deep-Reinforcement-Learning zum Einsatz kommt. Einige der Konzepte von MADRL, sowie bekannte Methodiken werden im Fokus dieser Ausarbeitung beziehungsweise im Rahmen des später durchgeführten experimentellen Abschnittes stehen. Im Grundsatz beschäftigt sich diese Ausarbeitung daher mit dem Einsatz von MARL. Dabei wird der Kernbestandteil dieser Arbeit darin liegen, verschiedene veröffentlichte Algorithmen in einem definierten Rahmen zu validieren. Im Speziellen werden die Methoden Value-Decomposition Networks (VDN), Mixed Q-Learning (QMIX) und Value-

Decomposition Networks for Actor-Critic (VDAC) zum Einsatz kommen, die im späteren Verlauf der Arbeit näher erläutert werden.

1.2 Stand der Wissenschaft und Technik

Der aktuelle technische Stand der künstlichen Intelligenz, im Folgenden mit KI abgekürzt, ist von außerordentlichem Fortschritt geprägt [20]. In der heutigen Zeit spielt die Informatik in fast allen Bereichen des Lebens eine zentrale Rolle. Ob in der Kommunikation, Unterhaltung, Bildung oder im Geschäftsleben. Die Verwendung von Computern und anderen digitalen Geräten hat sich zu einem festen Bestandteil der Gesellschaft entwickelt. Insbesondere die Relevanz von Systemen unter Einsatz künstlicher Intelligenzen soll hier besonders hervorgehoben werden. Weshalb sich im folgenden Abschnitt dem aktuellen wissenschaftlichen Stand des Forschungsbereichs gewidmet wird.

Insbesondere die im Rahmen dieser Arbeit behandelten Themenkomplexe der künstlichen Intelligenz waren erst kürzlich Bestandteil eines am 30. November 2022 veröffentlichten Artikels [21]. In dem von Pascal Leroy et al. [21] publizierten Artikel ging es darum, das beste Lernszenario für ein Agententeam zu ermitteln. Dies sollte erreicht werden, indem wertbasierte Methoden innerhalb einer Umgebung, die sowohl kooperative als auch kompetitive Elemente enthält, miteinander verglichen wurden. Die Autoren entschieden sich für das CTFD Verfahren, bei welchem das Training eines Lernmodells an einem zentralen Ort und die anschließende dezentralisierte Bereitstellung beziehungsweise Ausführung des trainierten Modells an einem anderen Ort stattfindet. Darüber hinaus hat sich das Team auf ein symmetrisches, teilweise beobachtbares Markov Szenario mit zwei Teams beschränkt. Als Trainingsmethoden wurden die Algorithmen QMIX, MAVEN und QVMix eingesetzt [21]. Neben speziellen aktuellen Forschungsentwicklungen, wie der Evaluation des Einsatzes der CTDE Methoden, wird auch grundlegend an der Weiterentwicklung von Deep-Reinforcement-Learning Algorithmen gearbeitet [25].

Dieser Forschungszweig ist gerade auch wegen seiner jüngeren Erfolge, wie dem im ersten Abschnitt erwähnten ChatGPT Gegenstand einiger Publikationen und befindet sich daher in stetiger Weiterentwicklung. Diesen Erfolgen stehen auch Herausforderungen gegenüber, so haben Multi Agent-Reinforcement-Learning Ansätze beispielsweise Schwierigkeiten bei nur teilweise beobachtbaren Szenarien und in Situationen, in denen Belohnungen eventuell nur nach Abschluss von Aktionen anderer Agenten ausgegeben werden können. Beispielsweise beim Bewältigen eines Labyrinthes, in dem verschiedene Agenten Aktionen voneinander abhängig sind. Hierzu wurde ebenfalls erst kürzlich ein neues Konzept der Kreditgewährenden Belohnungen (CCR) durch F. Bredell, H. A. Engelbrecht und J. C. Schoeman veröffentlicht [5]. Beiden genannten Veröffentlichungen liegen die Basistechnologien MARL beziehungsweise MADRL zugrunde. Die Zusammenarbeit von verschiedenen Agenten in verschiedensten Szenarien möglichst reibungslos und effizient zu gestalten ist Teil der aktuellen Forschung und Entwicklung in diesem Gebiet.

1.3 Schwerpunktabgrenzung und Forschungsfrage

Abgeleitet aus der im ersten Abschnitt erläuterten Relevanz und Problemstellung, sowie der im Abschnitt Stand der Wissenschaft und Technik gezeigten aktuellen Forschung wird der Schwerpunkt wie folgt dargelegt. Ziel dieser Ausarbeitung ist es, die in dem veröffentlichten Artikel „Value-based CTDE Methods in Symmetric Two-team Markov Game“ [21] gezeigten Ansätze weiterzuführen. Diese Abschlussarbeit unterscheidet sich von der genannten Publikation in den folgenden Aspekten. Im experimentellen Abschnitt der Ausarbeitung kommen alternative Methoden und Verfahren zum Einsatz. Im Speziellen fokussiert sich diese Ausarbeitung auf die Anwendung der Verfahren VDN, QMIX und VDAC, welche nicht oder nur teilweise in dem genannten Artikel behandelt worden sind. In der ursprünglichen Veröffentlichung kam das StarCraft Multi-Agent Challenge-Szenario zum Einsatz. Für diese Ausarbeitung wurde jedoch ein alternatives Testszenario ausgewählt, um die Leistungsfähigkeit der Methoden zu evaluieren. So dient im Rahmen dieser Arbeit als Umgebung ein Szenario, das dem klassischen Bomberman nachempfunden wurde. Bei Bomberman handelt es sich um ein im Jahr 1983 von Nintendo veröffentlichten Spiel, bei welchem in einer zweidimensionalen Umgebung Bomben abgeworfen werden müssen, um gegnerische Einheiten zu eliminieren. Das Szenario bietet wahlweise die Möglichkeit, zwei Teams oder einzelne Agenten gegeneinander antreten zu lassen. Wobei sich diese Ausarbeitung auf die Verwendung der Team-Funktionalitäten beschränkt. Aus den soeben genannten Abgrenzungen leiten sich die folgenden Forschungsfragen ab:

1. Welche der angewandten Methoden ist gemessen anhand der erlangten Siege und verglichen zu den durch das Szenario bereitgestellten Referenzagenten die beste?

2. Wie können die genannten Methoden für ein Partially observable Markov decision process POMDP Szenario angewendet werden?

Die erste Forschungsfrage wird damit beantwortet, dass zum Abschluss der experimentellen Arbeit ein Turnier der im vorherigen Schritt implementierten Algorithmen abgehalten wird. Im Rahmen dieses Turniers wird über eine Iteration hinweg das jeweils pro Algorithmus trainierte Modell gegen die durch das verwendete Szenario bereitgestellten Referenzagenten antreten gelassen. Während der laufenden Iteration wird jede Partie ausgewertet. Diese Auswertung definiert, welcher Kontrahent gewonnen oder verloren hat oder in welchen Fällen es zu einem Unentschieden gekommen ist. Nach Abschluss des Turniers werden die gesammelten Informationen aufbereitet und analysiert dargestellt. Die Herangehensweise zur Beantwortung der zweiten Forschungsfragen wird wie folgt hergeleitet. Im experimentellen Abschnitt wird basierend auf den Veröffentlichungen der jeweiligen Methoden ein lauffähiger Multi Agent-Reinforcement-Learning Ansatz realisiert, auf dessen Basis die jeweiligen Algorithmen implementiert werden. Ferner sollen alle Methoden unter möglichst ähnlichen Rahmenbedingungen angewendet werden, dies schließt die Anzahl der Trainingsepisoden sowie die Trainingsumgebung mit ein. Anschließend findet eine Evaluation sowie Erläuterung der Implementierung der jeweiligen Ansätze statt.

1.4 Aufbau und Ablauf der Arbeit

Im nachfolgenden Absatz wird das Konzept der Ausarbeitung sowie die Methodik des späteren experimentellen Abschnitts der Abschlussarbeit näher erläutert. Das übergeordnete Ziel ist es, dass dem Leser ein Grundverständnis im Bereich Künstlicher Intelligenz ferner im Fachbereich des Reinforcement-Learning und Multi Agent-Reinforcement-Learning vermittelt wird. Zusätzlich wird er in der Lage sein, die gezeigten Algorithmen basierend auf der genannten Technologie zu trainieren und zu implementieren.

Hierzu folgt die Ausarbeitung dem nachfolgenden Schema. Diese Arbeit beginnt mit einer Einleitung, hier wird neben der grundlegenden Problemstellung und Relevanz des übergeordneten Themas auch der aktuelle Stand von Wissenschaft und Technik, basierend auf einer eingängigen Literaturrecherche dargelegt. Im anschließenden zweiten Kapitel folgt eine Einführung in die wichtigsten Begrifflichkeiten und Grundlagentechnologien. Diese sind notwendig, damit die im späteren Verlauf gezeigten Implementations-Ansätze und Schlussfolgerungen nachvollzogen werden können. Neben diesen grundsätzlichen Basistechnologien wird in diesem Kapitel eine detaillierte Einführung in die im experimentellen Absatz genutzten Algorithmen stattfinden. Nach diesen Einführungsabschnitten wird die Ausarbeitung in einen praktischen Teil übergeleitet. In diesem wird das verwendete Szenario eingeführt und die für das Verständnis notwendigen Rahmenbedingungen definiert. Außerdem erfolgt hier eine detaillierte Definition der in den darauffolgenden Kapiteln verwendeten Trainingsvariablen und Kennzahlen. Darauffolgend wird im vierten Kapitel der experimentelle Bestandteil der Ausarbeitung gezeigt.

Hierzu findet neben einer Einführung und Erläuterung der jeweiligen Testumgebung sowie des Implementation-Aufbaus auch die eigentliche Evaluation der Ergebnisse nach dem Training statt. Dem experimentellen Abschnitt der Ausarbeitung folgt eine kritische Reflexion der in den vorherigen Kapiteln gesammelten Resultate und Informationen sowie einer generellen kritischen Auseinandersetzung der Testverfahren zum gewählten Testszenario. Die Ausarbeitung schließt mit einem Fazit ab. Innerhalb des Fazits werden die während der Arbeit erlangten Erkenntnisse kritisch betrachtet. Darüber hinaus werden basierend auf den Erfahrungen, die während der Implementations- und Ausführungsphase gesammelt worden sind, Vorschläge unterbreitet, wie das Training der Algorithmen theoretisch verbessert werden könnte. Die Arbeit schließt damit ab, dass basierend auf den Forschungsergebnissen, Forschungsschwerpunkte für zukünftige Arbeiten abgeleitet werden.

Im Detail ist vorgesehen das in Open-Source veröffentlichte Framework MARLlib einzusetzen. Dieses bietet eine Sammlung an Werkzeugen für verschiedene Multi Agent-Reinforcement-Learning Algorithmen. Als Testumgebung kommt das Szenario Pommerman zum Einsatz, das in einer Zwei-Team-Varianz verwendet wird. Während des Benchmarks, sowie im Verlauf des durchgeführten Turniers werden alle Algorithmen unter gleichen Voraussetzungen und unter identischen Bedingungen, das heißt insbesondere gleicher Trainingsmethoden, Trainingsepisoden usw. durchgeführt. Dadurch soll ein möglichst homogenes und vergleichbares Ergebnis sichergestellt werden.

2 Theoretische Grundlagen

Dieser Teil behandelt die theoretischen Grundlagen vor dem experimentellen Teil der Ausarbeitung. Die nachfolgenden Erläuterungen sind grundlegend, damit die im späteren Verlauf gezeigten Implementationstechniken und Methoden verstanden werden können. Neben den technologiebasierten Grundlagen erfolgt hier zusätzlich eine Einführung in die angewandten Algorithmen. Zum besseren Verständnis sind einige der nun folgenden Definitionen um Anwendungsbeispiele erweitert. Der Abschnitt beginnt mit einer Einführung in den Themenkomplex des Reinforcement Learnings und geht über in einen Exkurs im Bereich des Deep-Reinforcement-Learnings. Darauf aufbauend wird der Bereich MARL bzw. Multi Agent-Deep-Reinforcement-Learning erläutert. Das Kapitel schließt mit einer Einführung der verwendeten Algorithmen ab.

2.1 Reinforcement Learning

In diesem Segment wird grundlegend auf den Themenkomplex des Reinforcement Learning eingegangen. Die nun folgende Einführung ist inspiriert von der Darstellung von Sutton and Barto [39]. Reinforcement Learning wird zu Deutsch auch bestärkendes Lernen genannt. Reinforcement Learning wird in der Fachliteratur und in dieser Ausarbeitung mit RL abgekürzt. RL kann auch vereinfacht als Lernen durch Interaktion mit der Umgebung und dem Betrachten, wie sich diese Umgebung verhält, nachdem mit ihr interagiert wurde, bezeichnet werden. Das ist ein Verhalten, das schon bei Kindern in der Entwicklungsphase beobachtet werden kann. Vereinfacht gesagt ist Reinforcement Learning eine Methode, die versucht, für Situationen Maßnahmen zu finden, um ein Belohnungssignal zu maximieren [39]. Wobei hier klar abzugrenzen ist, dass während eines solchen Lernprozesses dem Lernenden, in dem Kontext oft Agent genannt, nicht vorgegeben wird, welche Aktion zu wählen ist. Stattdessen muss der Agent selbst durch kontinuierliches Ausprobieren geeignete Strategien entwickeln, um letztlich die Aktion zu wählen, die die höchste Belohnung verspricht. Zur Unterstützung beim Erlernen der geeigneten Strategie ist es wichtig, dass vorher festgelegt wird, in welchen Fällen eine Belohnung ausgeben wird. Eine vorstellbare Situation wäre es, dem Agenten eine positive Belohnung auszugeben, wenn dieser einen Siegpunkt in einer Runde erzielt hat [39].

Für RL lösbare Problemsituationen sollten dem Markov Decision Process folgen [28]. In diesem Prozess geht es darum, einem Agenten ein mögliches Problem in einer definierten Umgebung zur Erreichung eines Ziels zu modellieren. Ein solcher Agent muss die Möglichkeit haben, die Umgebung wahrnehmen zu können. Beispielsweise muss er seine aktuelle Position oder die Position anderer wichtiger Objekte, wie Hindernisse erkennen können. Außerdem ist es wichtig, die Aktionen zu definieren, mit der es einem lernenden Agenten möglich ist, diese Umgebungsvariablen zu verändern. Greift man hier die Variable der Position auf, wäre eine solche Aktion beispielsweise, dass der Agent sich innerhalb der Umgebung bewegen kann, somit würden sich die Koordinaten seiner Position sowie die relative Position zu Hindernissen für ihn ändern. Zudem ist es notwendig, dass ein Agent Ziele hat, welche mit der Umgebung zusammenhängen, beispielsweise das Abschließen eines Levels oder der Sieg einer Spielpartie [39]. Abbildung 1 zeigt bildlich dargestellt das soeben erläuterte Verhältnis des Agenten zur Umgebung.

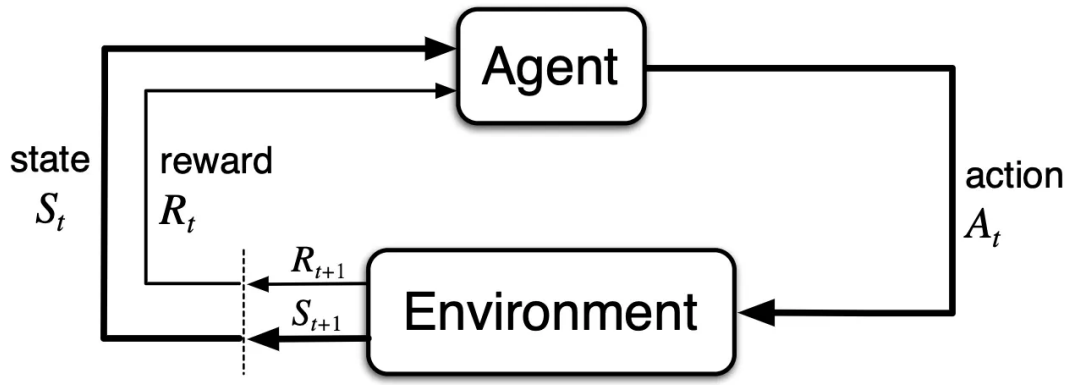


Abbildung 1: Reinforcement Learning Algorithmus [39]

Ein besonderes Merkmal von Reinforcement-Learning ist die Unterscheidung zwischen Erforschung und Ausnutzung. Dieses Merkmal stellt gleichzeitig eines der wichtigsten Stellschrauben beim späteren Training der Agenten dar. Damit ein Agent möglichst nur die Aktion auswählt, welche die höchste Erfolgschance auf eine maximale Belohnung gewährt, müsste er sich vermehrt auf das Ausnutzen der Umgebung fokussieren. Dies wäre notwendig, um Erlerntes anzuwenden und möglichst viele Belohnungen zu erhalten. Im Gegensatz dazu muss der Agent aber auch erforschen, ansonsten kann er nicht mit Sicherheit wissen, was die bestmögliche Aktion darstellt, die er in der jeweiligen Situation ausführen könnte. Bei dem genannten Phänomen handelt es sich um das sogenannte Exploration-Exploitation-Dilemma, das Ziel kann nie erreicht werden, wenn der Agent sich nur auf eines der beiden Merkmale fokussiert. Ein bekanntes Schema ist es u. a. den Exploration-Wert in der Anfangsphase eines Lernprozesses möglichst hoch anzusetzen, damit viele verschiedene Erfahrungen gesammelt werden und dann sukzessive diesen zu reduzieren. Allerdings sollte dieser Wert niemals null erreichen, stattdessen könnte hier auch mit einem Zufallswert gearbeitet werden, der im kleinen einstelligen Bereich liegt. Somit würde der Agent vermehrt auf bereits bekanntes Wissen setzen und trotzdem sporadisch neues ausprobieren, um gegebenenfalls noch unbekannte, idealere Lösungsvariationen zu erlernen [39].

Strategie / Policies Eine Policy (dt. Strategie) ist eine Funktion, die im Reinforcement Learning verwendet wird, um Entscheidungen in einer bestimmten Umgebung zu treffen. Die Policy nimmt als Eingabe den aktuellen Zustand (State) eines RL-Systems und gibt als Ausgabe eine Aktion aus, die ausgeführt werden soll. Die Policy kann deterministisch oder stochastisch sein. Eine deterministische Policy gibt immer die gleiche Aktion für einen bestimmten Zustand aus, während eine stochastische Policy eine Wahrscheinlichkeitsverteilung für verschiedene mögliche Aktionen zurückgibt. Das Ziel bei der Verwendung einer Policy besteht darin, eine gute Strategie zu finden, die das maximale Belohnungssignal in der RL-Umgebung erzeugt. Dies wird normalerweise durch den Einsatz von Algorithmen wie z.B. Q-Learning erreicht. Q-Learning wird im Rahmen eines späteren Kapitels im Detail erläutert. Die Policy eines Agenten ist eine Schlüsselrolle bei der Lösung von RL bezogenen Problem-

stellungen, da sie die Handlungen bestimmt, die der Agent ausführt, um ein definiertes Ziel zu erreichen [39].

Belohnungsfunktion / Rewards function Die Belohnungsfunktion ist dafür zuständig, das Ziel eines Problems zu definieren, dabei wird an den Agenten in einem gegebenen Zeitintervall eine Belohnung ausgeben. Die Funktion beinhält die Kriterien, nach denen festgelegt wird, ob eine Belohnung positiv oder negativ ausfällt. Insbesondere in Verbindung mit der Policy ergibt sich eine Synergie, die es wahrscheinlich macht, dass die Policy angepasst werden muss, wenn die Belohnung niedrig ausfällt. In diesem Fall sollte der Agent idealerweise zukünftig entscheiden, eine andere Aktion auszuwählen, welche eine höhere Belohnung zum Ergebnis hätte [39].

Wertefunktion / Value function Die im vorherigen Absatz erläuterte Erfolgsfunktion kann als kurzfristiger Indikator gesehen werden, ob eine durch die Policy entschiedene Aktion als positiv zu werten ist. Im Gegensatz dazu gibt die Wertefunktion an, wie positiv eine Aktion längerfristig betrachtet für den Agenten wäre. Dabei gilt, dass die Wertefunktion beziehungsweise das Vorhersagen eines effektiven Wertes eines der Hauptbestandteile vieler RL-Algorithmen darstellt. Das ergibt sich insbesondere daraus, dass es in der Regel einfach ist, die korrekten Werte einer Belohnung zu bestimmen, da diese meist direkt durch die Umgebung selbst definiert wird. Die Wertefunktion hingegen baut sich auf vielen Beobachtungen der Vergangenheit auf und ist damit schwieriger zu bestimmen [39].

Centralized Training and Decentralized Execution Centralized Training and Decentralized Execution abgekürzt auch CTDE ist eine im Bereich RL oft verwendete Methode. Diese folgt dem Ansatz, das Training eines oder mehrerer Agenten zentralisiert unter Laborbedingungen abzuhandeln und das Ausführen des trainierten Modells an einem anderen Ort stattfinden zu lassen. Oft handelt es sich bei dem Ort des Trainings um abgewandelte Umgebungen, in denen es vorkommt, dass im Verlauf des Trainingsprozesses dem Agenten mehr Informationen bereitgestellt werden als es in der tatsächlichen Ausführung der Fall ist. Beispielsweise erhält der Agent so während des Trainings mehr Informationen über seine Umgebung, oder eine Möglichkeit der direkten Kommunikation mit seinen Team-Agenten [33].

Actor critic Ein Critic im Reinforcement Learning dient dazu, die Qualität der aktuell angewendeten Strategie zu bewerten. Konkret schätzt der Critic die erwartete zukünftige Belohnung, indem er mögliche Belohnungen für jeden Zustand berechnet und aggregiert. Bei den sogenannten Actor-Critic-Methoden gibt es zwei Hauptkomponenten, den Actor und den Critic. Der Actor repräsentiert die aktuelle Strategie bzw. Policy, die beschreibt, mit welcher Wahrscheinlichkeit welche Aktion in einem gegebenen Zustand gewählt wird. Der Critic hingegen bewertet diese Aktionen, indem er eine Wertefunktion abbildet, die den Zuständen erwartete zukünftige Belohnungen zuordnet. Der Critic und der Actor arbeiten eng zusammen. Während der Actor Aktionen basierend auf der aktuellen Policy auswählt, gibt der Critic

Feedback zu diesen Entscheidungen. Mithilfe dieses Feedbacks kann der Actor seine Policy anpassen und optimieren, um bessere Ergebnisse im Lernprozess zu erzielen [4].

Gedächtnis / Replay buffer Bei einem Replay buffer handelt es sich im Grunde um einen einfachen Speicher, in den ein Agent rundenbasierte Informationen ablegen kann. Daher kann der Replay buffer vereinfacht gesagt auch als Agenten Gedächtnis bezeichnet werden [25]. Der Replay buffer wird typischerweise als Tupel formalisiert dargestellt, $e_t = (s_t, a_t, r_t, s_{t_+1})$. Wobei s_t den Zustand pro Zeitintervall, a_t die Aktion pro Zeitintervall, r_t die Belohnung pro Zeitintervall und s_{t_+1} den nächstfolgenden Zustand innerhalb des Gedächtnisses definiert. Eine Möglichkeit, welche die Verwendung eines Replay buffers mitbringt, ist es, dass während des Trainingsprozesses der Agent beispielsweise mittels Minibatch-Updates eine Erfahrung aus seinem Gedächtnis abrufen kann. Mit der Konsequenz, dass eine entsprechende Aktion, basierend auf den Informationen aus dem Replay buffer ausgeführt wird. Dieses Schema erlaubt es dem Agenten aus seinen Erfahrungen, die er in der Vergangenheit gemacht hat, zu lernen [25]. Der Einsatz dieser Technik bringt einige Vorteile mit sich. Insbesondere ist das Lernen so wesentlich effizienter, weil Berechnungen für Informationen im Gedächtnis bereits erfolgt sind. Zusätzlich ist es dem Agenten so möglich, auch von Ereignissen zu lernen, die bei realer Anwendung gegebenenfalls selten auftreten würden, aber im Gedächtnis zur Verfügung stehen [25].

Multi-Agent-Reinforcement Learning MARL Multi-Agent-Reinforcement Learning, abgekürzt mit MARL, ist ein Teilsegment des Reinforcement-Learnings. Im Gegensatz zu traditionellen Reinforcement-Learning Ansätzen, bei denen nur ein einzelner Agent betrachtet wird, bezieht sich Multi-Agent-Reinforcement Learning auf die Fähigkeit von mehreren Agenten in einer gemeinsamen Umgebung zusammenzuarbeiten. Oft sind es kooperative Szenarien, bei denen Teams gegeneinander antreten, beispielsweise bei Strategiespielen wie StarCraft oder dem von OpenAI veröffentlichen Szenario Hide-and-Seek. Multi-Agent-Reinforcement Learning basiert auf den gleichen Grundprinzipien wie Reinforcement-Learning. Die Ausnahme ist, dass MARL häufig in komplexen Umgebungen eingesetzt wird, in denen es schwierig ist, die optimalen Entscheidungen unmittelbar festzulegen. Insbesondere für den Einsatz in der Praxis, d.h. außerhalb von Laborbedingungen spielt MARL eine essenzielle Rolle. Viele Probleme, welche es notwendig machen, dass mehrere Agenten an einer gemeinsamen Aufgabe zugleich arbeiten, können als MARL Problem definiert werden, z. B. das Steuern von Fahrzeugen [35].

2.1.1 Markov Decision Process MPD

Ein Markov Decision Process (dt. Markov-Entscheidungsprozess), abgekürzt und im nachfolgenden mit MDP bezeichnet, ist ein mathematisches Modell, das verwendet wird, um Entscheidungen unter Unsicherheiten, d. h. unter nicht vollständig einsehbaren Szenarien zu optimieren. Es wird häufig in der Künstlichen Intelligenz und im maschinellen Lernen eingesetzt, um die besten Entscheidungen für eine gegebene Situation zu treffen. Insbesondere findet es Anwendung, wenn es darum geht, eine komplexe Umgebung für einen Agenten zu modellie-

ren. Ein MDP besteht aus einer Menge von Zuständen, die vom System durchlaufen werden können, und einer Menge von Aktionen, die vom System ausgeführt werden können. Dieser definiert sich als ein Tupel mit den Inhalten (S, A, R, P). Innerhalb des Tupel bezeichnet S eine Menge an beliebigen Zuständen. A steht für die Menge möglicher Aktionen, die ausführbar sind, beispielsweise die Bewegungsmöglichkeiten eines Agenten. Das P definiert die Übergangswahrscheinlichkeit der Zustände mittels $P(s'|s,a)s, s' \in S, a \in A$. Darüber hinaus repräsentiert R die Belohungsfunktion, die den zu erwarteten Gewinn beschreibt, diese kann formell als $R : (SxA) \rightarrow \mathbb{R}$ definiert werden [16].

Jeder Zustand S ist durch eine bestimmte Wahrscheinlichkeit für jede mögliche Aktion gekennzeichnet. Diese Wahrscheinlichkeiten werden als Übergangsfunktion bezeichnet und beschreiben, welche Aktionen vom aktuellen Zustand aus möglich sind und wie wahrscheinlich es ist, dass das System in einen bestimmten Zustand gelangt, wenn eine bestimmte Aktion ausgeführt wird. Ein weiteres wichtiges Konzept im MDP ist die sogenannte Belohnungsfunktion. Diese gibt an, welche Belohnung, ob negativ oder positiv das System für jeden Zustand erhält. Die Belohnungsfunktion kann entweder absolut sein, d.h. es gelten feste Werte für jeden Zustand, oder relativ, d.h. sich auf den Unterschied zwischen zwei aufeinanderfolgenden Zuständen beziehen. Allgemein löst ein MDP das Problem der Entscheidungsfindung, indem es eine Strategie festlegt, die das System befolgen soll, um die höchstmögliche Belohnung zu erhalten. Diese zu findende Strategie wird optimale Policy bezeichnet und kann entweder explizit oder implizit angegeben werden. Eine explizite Policy legt für jeden Zustand eindeutig fest, welche Aktion ausgeführt werden soll. Eine implizite Policy hingegen gibt nur an, welche Aktion für jeden Zustand am wahrscheinlichsten ist, ohne festzulegen, welche Aktion ausgeführt werden muss. Darüber hinaus unterscheidet man zwei Arten von MDP, deterministisch oder stochastisch. Ein deterministischer MDP legt fest, dass jeder Zustand immer nur durch eine bestimmte Aktion erreicht werden kann. Ein stochastischer MDP hingegen berücksichtigt die Unsicherheit und beschreibt diese [39].

Partially observable Markov decision process POMDP Ein Partially observable Markov decision process, abgekürzt mit POMDP ist eine Spezialisierung des im vorherigen Absatz erläuterten MDP. Im Gegensatz zum normalen MDP ist der Zustand der Umgebung für einen POMDP nur teilweise beobachtbar. Die Wahrscheinlichkeit, dass sich der Zustand der Umgebung nach einer Entscheidung oder aufgrund externer Einflüsse ändert, wird im POMDP-Modell zusätzlich berücksichtigt [22]. Abweichend zum MDP definiert sich der POMDP formell über einen Tupel mit den folgenden Werten (S, A, Z, T, R, O, γ). Der Inhalt des Tupel ist angelehnt an die Definitionen der MDP, wobei die folgenden abweichenden Definitionen greifen [39]:

1. S, definiert eine endliche Menge an Zuständen
2. A, definiert eine endliche Menge an ausführbaren Aktionen
3. Z, definiert den möglichen Beobachtungshorizont
4. T, beschreibt die Transaktionsfunktion $T(s'|s,a)$

5. R, beschreibt die Belohnungsfunktion $R(s, a)$
6. O, beschreibt die Beobachtungsfunktion $O(o|s')$
7. γ, definiert den Discountfaktor, typischerweise liegt dieser zwischen 0 und 1

Insbesondere liegt der Unterschied, zwischen einem POMDP und MDP in der Tatsache, dass bei einem POMDP, der Zustand S nie als sicher gilt. Stattdessen wird von einem sogenannten Belief ausgegangen. Also einem Glauben, daran, was der richtige Zustand S ist. Das ist darauf zu zurückzuführen, dass innerhalb eines Szenarios, das durch POMDP formuliert werden kann, nicht alle Informationen der Umgebung vollständig zur Verfügung stehen [18][25].

2.2 Deep Reinforcement-Learning

Im nachfolgenden Abschnitt wird der Bereich des Deep-Reinforcement-Learning, der mit D-RL abgekürzt wird, erläutert. D-RL stellt eine erweiterte Art des bereits definierten Reinforcement-Learning dar. Im Gegensatz zu anderen Bereichen des maschinellen Lernens, beispielsweise bei der Anwendung von einfach neuronalen Netzwerken, wird D-RL nicht auf vorgegebenen Daten trainiert. Stattdessen können Algorithmen des Deep Reinforcement Learning selbstständig lernen, indem sie Belohnungen und Strafen für ihre Handlungen erhalten und daraus lernen, welche Handlungen in bestimmten Situationen am vorteilhaftesten sind. Das Prinzip ist identisch dem normalen Reinforcement-Learning. Deep Reinforcement Learning kombiniert die Technologien von Deep Learning und Reinforcement Learning, um leistungsfähigere Algorithmen zu entwickeln. Deep Learning ermöglicht es, sehr komplexe Funktionen zu approximieren und auf großen Datenmengen zu trainieren, während Reinforcement Learning eine Rahmenbedingung bereitstellt, in der die Algorithmen lernen können. Deep Reinforcement Learning wird häufig in komplexen Umgebungen eingesetzt, in denen es schwierig ist, die optimalen Entscheidungen festzulegen [24]. Beispiele hierfür sind die Steuerung von Robotern in unbekannten Umgebungen oder das Lernen von Spielstrategien in Computerspielen. D-RL findet im späteren Verlauf der Arbeit seine Anwendung bei einigen der gezeigten Algorithmen, die im experimentellen Teil Anwendung finden.

2.2.1 Q-Learning QL

Im nachfolgenden Kapitel wird erläutert, worum es sich bei Q-Learning handelt und wie dieses funktioniert. Q-Learning ist ein Verfahren des Reinforcement Learning, bei dem ein Agent lernt, optimale Entscheidungen in einer gegebenen Umgebung zu treffen. Im Gegensatz zu anderen Reinforcement Learning-Techniken, bei denen der Agent Belohnungen oder Strafen direkt erhält bzw. nach vorher definierten Rahmenbindungen, verwendet Q-Learning eine sogenannte Q-Tabelle als Referenz. Dies hat zum Ziel, die Belohnungen für jede mögliche Aktion in jedem möglichen Zustand zu speichern. Der Agent wählt dann immer die Aktion aus, die im aktuellen Zustand die laut der erzeugten Q-Tabelle höchste Belohnung verspricht [43]. Eine Q-Tabelle, welche die namensgebenden Q-Values enthält, kann wie folgt definiert und aufgebaut werden:

1. Initialisierung
   - Die Q-Tabelle wird initialisiert, indem alle Einträge auf einen festen Startwert gesetzt werden.
2. Wählen einer Aktion
   - Der Agent wählt eine Aktion aus, basierend auf dem aktuellen Zustand und der Q-Tabelle. Dies kann entweder deterministisch oder stochastisch geschehen, je nachdem, ob der Agent immer die gleiche Aktion wählt oder zufällig aus mehreren möglichen Aktionen auswählt.
3. Ausführen der Aktion
   - Der Agent führt die gewählte Aktion aus und gelangt in einen neuen Zustand.
4. Berechnen der Belohnung
   - Der Agent erhält eine Belohnung für sein Verhalten, die von der Umgebung berechnet wird.
5. Aktualisieren der Q-Tabelle
   - Die Q-Tabelle wird aktualisiert, indem der Eintrag für den aktuellen Zustand und die gewählte Aktion anhand der erhaltenen Belohnung angepasst wird.

Dies geschieht mithilfe folgender Formel:

$$\underbrace{Q(s,a)}_{\text{Neuer Q-Value}} = \underbrace{Q(s,a)}_{\text{Aktueller Q-Value}} + \underbrace{\alpha}_{\text{Lernrate}} \Big[\underbrace{R(s,a)}_{\text{Belohnung}} + \underbrace{\gamma}_{\text{Discount-Faktor}} \overbrace{\max Q'(s',a')}^{\substack{\text{Maximale vorhergesagte Belohnung, für gegebenen} \\ \text{neuen Zustand und alle möglichen ausführbaren Aktionen}}} - Q(s,a) \Big]$$

Wobei $Q(s, a)$ der aktuelle Eintrag in der Q-Tabelle für den Zustand s und die Aktion a ist. Alpha α ist der Lernfaktor, r die erhaltene Belohnung, γ der Discount-Faktor und $Q(s', a')$ ist der maximal erwartete Wert für den neuen Zustand s' und alle möglichen Aktionen a'. Der Discount-Faktor γ gibt an, wie sehr die Belohnungen in der Zukunft gewichtet werden sollen. Ein hoher Discount-Faktor bedeutet, dass die Belohnungen in der Zukunft stärker berücksichtigt werden, während ein niedriger Discount-Faktor die Belohnungen im aktuellen Zustand stärker berücksichtigt [39].

Um nun mittels der Q-Funktion lernen zu können, wird der Agent in und mit der Umgebung agieren und kontinuierlich Daten erzeugen. Zu sammelnde Daten sind beispielsweise welche Aktionen in welchen Zuständen die höchstmögliche Belohnung erzielten. Anfangs wird die Q-Funktion zufällig initialisiert und im Laufe der Zeit wird sie durch die Interaktion des Agenten mit der Umgebung verbessert, bis sie schließlich nicht mehr auf einem Zufallswert basiert. Dabei verwendet Q-Learning einen sogenannten Exploration-Exploitation-Tradeoff, um zu entscheiden, ob der Agent eine Aktion ausführen soll, die er bereits kennt

oder ob er eine neue Aktion ausprobieren soll, um neue Erfahrungen zu sammeln. Das Ziel des Exploration-Exploitation-Tradeoff ist es, ein Gleichgewicht zwischen Exploration und Exploitation zu finden. Exploration bezieht sich darauf, neue Aktionen auszuprobieren und neue Erfahrungen zu sammeln, während Exploitation darauf abzielt, bekannte Aktionen auszuführen, um die höchstmögliche Belohnung zu erzielen. Ein Agent, der sich hauptsächlich auf Exploration konzentriert, wird in der Lage sein, viele neue Erfahrungen zu sammeln, kann aber möglicherweise nicht die höchstmögliche Belohnung erzielen, da er bekannte Aktionen nicht ausreichend nutzt. Verwendet er auf der anderen Seite hauptsächlich Exploitation, wird der Agent in der Lage sein, die lokal höchstmögliche Belohnung zu erzielen, kann aber möglicherweise nicht auf neue Situationen angemessen reagieren. Deshalb wird versucht, das bestmögliche Gleichgewicht der beiden Elemente zu finden [39].

Deep Q-Networks DQN DQN auch als Deep Q-Networks bezeichnet, ist ein Verfahren bekannt aus dem Bereich des Reinforcement-Learnings. DQN setzt insbesondere auf eine Kombination der bekannten Technologien aus den Bereichen des RL mit neuronalen Netzen. So kann beispielsweise die in vorherigen Abschnitten definierte Q-Funktion, bekannt aus dem Q-Learning, über neuronale Netze dargestellt werden. Dabei verwendet DQN viele für den Erfolg der Methode wichtige, neuartige Schlüsseltechnologien [24][25]. Im nachfolgenden Kapitel wird im Kontext von DQN, auf Spezifika von neuronalen Netzen eingegangen, welche zum Verständnis späterer Methoden notwendig sind. Hierbei wird sich vorwiegend auf relevante Teilaspekte von neuronalen Netzen fokussiert, welche für die Ausarbeitung von Bedeutung sind.

2.2.2 Künstliche Neuronale Netze

Im nachfolgenden Absatz wird grundlegend auf notwendige Teilaspekte zum Thema künstliche neuronale Netze, welche nachfolgend mit KNNs abgekürzt werden, eingegangen. KNNs finden in allen später eingesetzten Methoden ihre Anwendung, das liegt daran, dass alle betrachteten Methoden auf den Einsatz von DQN setzen. KNNs basieren auf Beobachtungen aus der Biologie, vorwiegend der Betrachtung des menschlichen Gehirns und dem Ablauf von Verarbeitungsprozessen innerhalb des Gehirns. Deswegen spricht man im Kontext von KNNs auch davon, dass es aus Knoten und Kanten besteht, angelehnt an Neuronen und Synapsen aus der Neurobiologie [12].

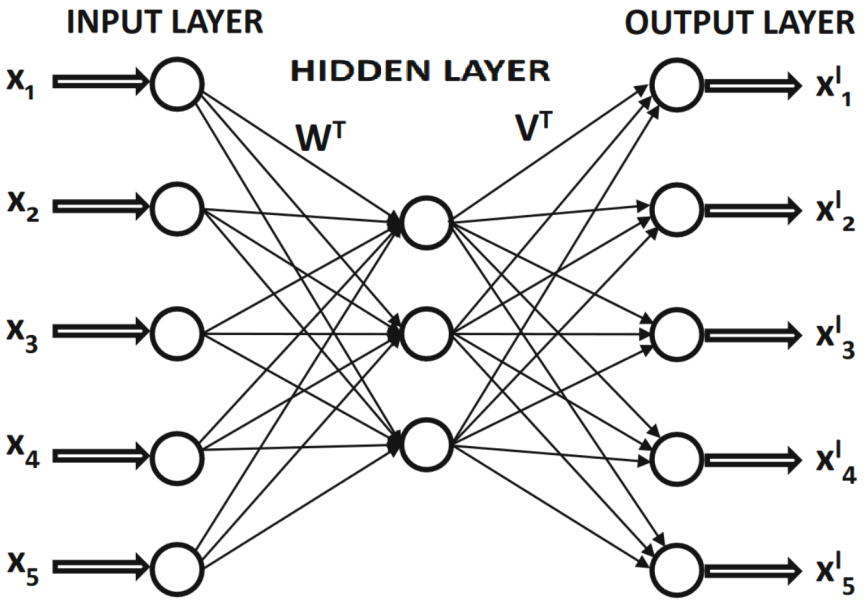


Abbildung 2: KNN Architektur in Anlehnung [2]

Wie in Abbildung 2 zu sehen ist, unterscheidet man zwischen drei verschiedenen Arten von Knotenpunkten bzw. Neuronen. Diese werden auch als Layer bzw. Ebenen bezeichnet. Der Layer, zu sehen auf der linken Seite in Abbildung 2, stellt den sogenannten Input-Layer dar, dieser repräsentiert die Eingangsinformationen zum KNN. Die Eingangswerte variieren und unterscheiden sich je nach tatsächlichem Einsatzzweck. Diese können über numerische Werte bis hin zu Pixeln von Bildern viele unterschiedliche Eingangswerte abbilden. Dem Input-Layer folgend ist der Hidden-Layer dargestellt. Ein KNN kann aus einem oder beliebig vielen Hidden-Layern bestehen, diese dienen dann als Bindeglied zwischen den verschiedenen Schichten des KNN. Das KNN schließt ab mit einem Output-Layer, der die Ausgangsdaten, respektive das Ergebnis des KNNs ausgibt. Alle Layer sind direkt miteinander verbunden, in diesem Fall spricht man von einem sogenannten „feed-forward“ KNN. Ein in der Literatur viel verwendetes Anwendungsszenario für KNNs ist die Klassifizierung von Bildern, zum Beispiel nach Hund oder Katze. Der Output-Layer des KNNs würde dann ausgeben, mit welcher Wahrscheinlichkeit es sich um eines der beiden Varianten handelt [12]. Diese Layer sind über Kanten miteinander verbunden, die jeweils eine Gewichtung und einen Bias enthalten. Beide, Gewichtung und Bias, ändern sich im Laufe des Trainingsprozesses eines künstlichen neuronalen Netzwerks, bis schließlich ein zu definierendes Optimum erreicht ist [12].

Multilayer Perceptrons MLP Als Multilayer Perceptrons (z. dt. mehrschichtiges Perzeptron), abgekürzt mit MLP, bezeichnet man eine bestimmte Variante eines „feed-forward“ KNNs. Dieses MLP besteht aus mehreren Ebenen von Knoten, bzw. Neuronen, die direkt miteinander verbunden sind. Das MLP basiert auf der Definition des normalen Perceptrons, das in Anlehnung an Abbildung 2 bereits beschrieben wurde. Mit dem Unterschied, dass hier, wie

in Abbildung 3 zu sehen, mehrere miteinander verbundene Ebenen, i.d.R. sind dies Hidden-Layer, vorhanden sind. Die interne Architektur des MLP folgt dem Schema beliebig viele, Input-Layer, Hidden-Layer und einem Output-Layer. Jede dieser Schichten besteht wiederum aus einer beliebigen Anzahl an Knoten, die miteinander verbunden sind. Jeder Knoten verwendet eine eigene Aktivierungsfunktion, die je nach Einsatzzweck abweichen kann, oft wird aber für einen Layer auf eine einzelne einheitliche Aktivierungsfunktion gesetzt. Somit stellt MLP eine Grundarchitektur eines KNNs mit mehren Hidden-Layer dar [12].

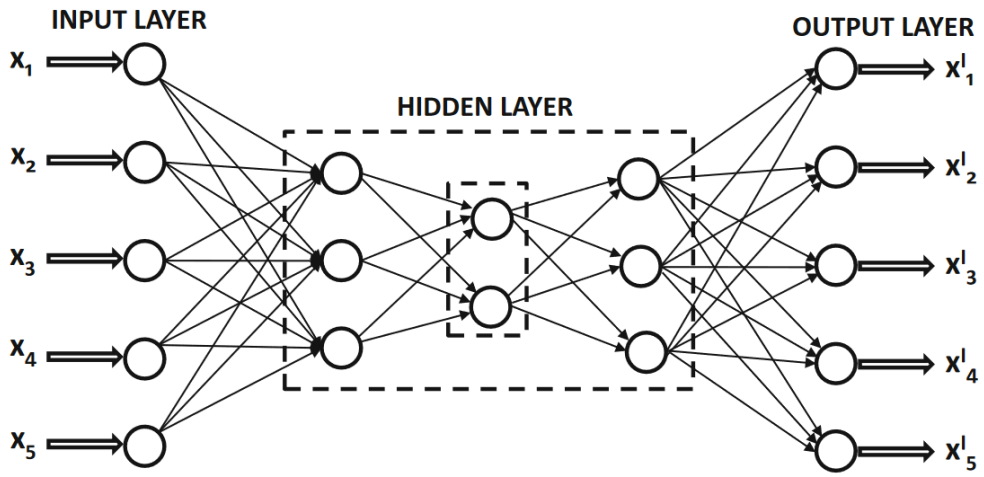


Abbildung 3: MLP Architektur in Anlehnung [2]

Aktivierungsfunktion Eine Aktivierungsfunktion stellt einen Kernaspekt von KNNs dar, diese ist notwendig, um eine nicht Linearität innerhalb des Netzwerkes zu ermöglichen. Diese nicht Linearität ist deshalb wichtig, weil sonst, mit fortlaufendem Lernprozess, die Berechnungen des KNNs exponentiell steigen würden. Durch reine Linearität ist es dem Netzwerk nicht möglich, komplexe Beziehungen zu erlernen. Jedoch ist es durch Nicht-Linearität möglich, solche komplexen Beziehungen zwischen dem Input und dem Output innerhalb des Netzwerkes zu erfassen. Dabei unterscheidet man eine Vielzahl an verschiedenen Aktivierungsfunktionen [2]. Nachfolgend sind die Aktivierungsfunktionen aufgelistet, die innerhalb der Ausarbeitung zum Einsatz kommen.

1. Sigmoid
    - Bildet den Input auf einen Wert zwischen 0 und 1 ab.
2. ReLU Rektifizierte lineare Einheit
    - Gibt einen positiven Wert zurück, ist der Wert negativ wird eine 0 zurückgegeben [3].
3. Tanh Hyperbolic Tangent
    - Bildet Eingabewerte auf einen Wert zwischen -1 und 1 ab.

Verlustfunktion Eine Verlustfunktion ist ein zentraler Bestandteil beim Trainingsprozess eines KNNs. Der primäre Zweck dieser Funktion besteht darin, den Unterschied zwischen dem vom Modell vorhergesagten Ergebnis und dem tatsächlich auftretenden Ergebnis zu quantifizieren. Aufgrund dieses Umstandes verfolgt man grundlegend das Ziel, die Verlustfunktion im Verlauf des Trainings eines Modells zu minimieren. Eine Minimierung würde bedeuten, dass die Vorhersagen des Modells zuverlässiger mit den tatsächlichen Ergebnissen übereinstimmen [13][17].

Backpropagation Bei Backpropagation handelt es sich um einen populären Algorithmus, der zum Training von KNNs eingesetzt wird. Dieser findet auch häufig Anwendung innerhalb des Lernprozesses bei MLPs. Der Algorithmus wurde erstmals in den 1980ern von David E. Rumelhart et al. [32] publiziert. Im Kern des Verfahrens steht die Idee, Fehler rückwärts durch das Netzwerk zu propagieren, bis zum Input-Layer, um die Gewichtungen der Knoten im KNN anzupassen. Der exemplarische Ablauf des Algorithmus entspricht dem nachfolgenden Schema. Als Erstes wird die Ausgabe des KNNs, basierend auf einem Eingabewert berechnet, anschließend wird die Ausgabe mit der Ausgabe des vorherigen Durchlaufs mittels Verlustfunktion verglichen. Der berechnete Fehler wird rückwärts durch das KNN propagiert. Abschließend wird die Gewichtung des KNNs entsprechend angepasst [41].

2.2.3 Rekurrierende Neuronale Netze RNN

Ein rekurrierendes neuronales Netz (engl. recurrent neural networks) auch RNN genannt, ist eine Abwandlung des normalen „feed forward“ KNNs. Der größte Unterschied zu einem normalen KNN liegt darin, dass der Output des vorherigen Zustands auch als Input für den aktuellen Zustand dienen kann. RNN funktionieren insbesondere dann gut, wenn es darum geht, sequenzielle Daten zu verarbeiten, beispielsweise bei der Verarbeitung von natürlicher Sprache [41].

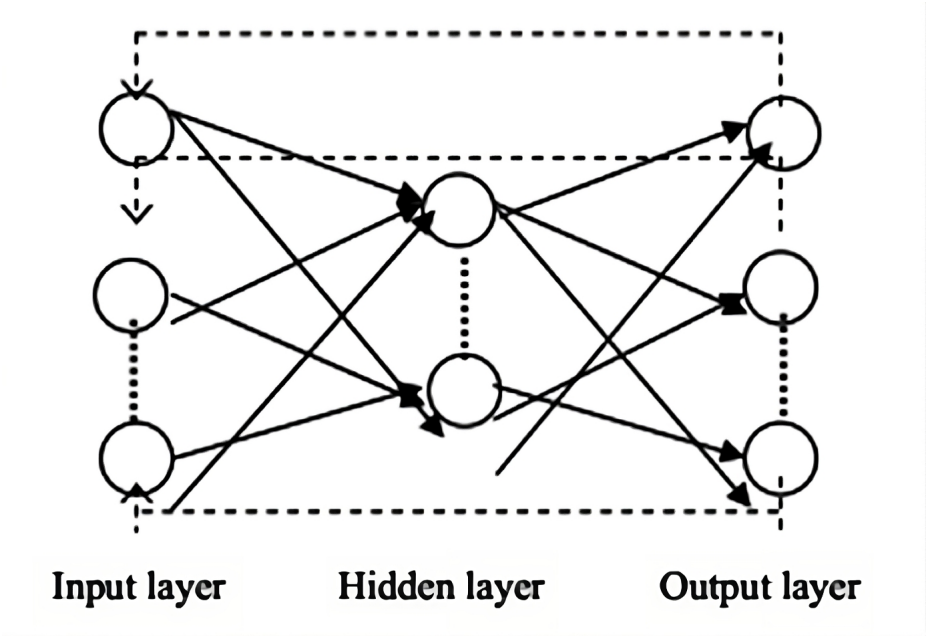


Abbildung 4: Beispiel eines RNN Aufbaus [14]

Die Abbildung 4 zeigt exemplarisch den Aufbau eines solchen RNNs. Im direkten Vergleich mit dem in Abbildung 2 gezeigten „feed forward“-Netzwerk, zeigt sich, dass RNNs mit sogenannten Rückkopplungsschleifen arbeiten. Diese erlauben dem Netzwerk Informationen aus dem vorherigen Durchlauf zu speichern und wiederzuverwenden. Einer der größten Vorteile gegenüber einem normalem KNN liegt darin, dass RNNs gut mit sequenziellen Daten umgehen können, außerdem ist es dem RNN möglich, von längerfristigen Abhängigkeiten zu lernen. Dies wird unter anderem mittels bestimmten Techniken ermöglicht, die im nachfolgenden Absatz erläutert werden [41].

Langes Kurzzeitgedächtnisses LSTM Unter einem langen Kurzzeitgedächtnis (engl. long-short term memory), abgekürzt mit LSTM, wird eine Technik der im vorherigen Abschnitt besprochenen RNNs verstanden. RNNs haben ebenso wie LSTM die Möglichkeit über eine Speicherfunktion auf vorherige Ereignisse zurückzugreifen. Im Gegensatz zu RNNs haben LSTMs aber den Vorteil auch längerfristige Abhängigkeiten erlernen zu können. LSTM selbst wurde von Hochreiter et al. [15] im Jahr 1997 vorgestellt, mit dem Zweck ein im klassischen RNN häufig auftretendes Problem zu lösen. Dieses Problem ist unter der Bezeichnung verschwindender Gradienten bekannt [15].

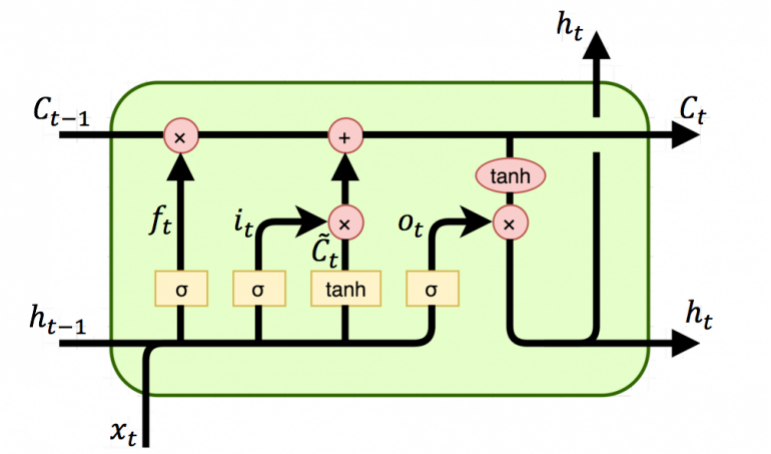


Abbildung 5: Beispiel einer LSTM-Architektur in Anlehnung [31]

LSTMs verwenden eine Reihe von Zellen bzw. Gates. Mithilfe der Gates entsteht die Möglichkeit, Informationen aus den Zellen zu entfernen bzw. Informationen hinzuzufügen, gesteuert von den jeweils zuständiges Gates. LSTMs bestehen in der Regel aus drei verschiedenen Gates, wobei jedes Gate aus einer Ebene mit einer Sigmoid Aktivierungsfunktion und einem Multiplikationsoperator besteht. Die im Netzwerk verwendete Sigmoid Aktivierungsfunktion reguliert die in die Zellen überführte Menge an Informationen und damit auch was an Informationen in das Gedächtnis geschrieben wird [26]. Dabei unterscheidet man zwischen drei verschiedenen Arten von Gates, Input-Gate, Output-Gate und Forget-Gate. Darüber hinaus ist der Zellzustand eine wichtige Komponente zu sehen in Abbildung 5, definiert als C_{t-1}.

In Abbildung 5 zu sehen ist die interne Architektur eines LSTMs. Das Input-Gate, in der Abbildung bezeichnet mit i_t wird definiert als $i_t = \sigma(W_i * x_t + U_i * h_(t-1) + b_i)$. Das Input-Gate hat hierbei die Funktion zu bestimmen, was in den Zellzustand übernommen werden sollen. Verwendung findet hierbei die Sigmoid Aktivierungsfunktion, die den Wert auf einen Index zwischen 0 und 1 abbildet, W_i, U_i und b_i stellen die Gewichtung des Netzwerks bzw. den Bias dar. Es folgt das zweite Gate, das unter der Bezeichnung Forget-Gate bekannt ist, dieses entscheidet wie viele Informationen aus dem vorherigen Zellzustand tatsächlich übernommen werden sollen. Hierzu wird wieder eine Sigmoid Aktivierungsfunktion eingesetzt. Das Forget-Gate ist in der Abbildung zu finden als f_t und wird formalisiert als $f_t = \sigma(W_f * x_t + U_f * h_(t-1) + b_f)$, wobei hier W_f, U_f und b_f wieder für die Gewichtung bzw. das Bias des Netzwerkes stehen. Die LSTM-Architektur schließt mit dem Output-Gate ab, das in der Abbildung mit o_t bezeichnet und als $o_t = \sigma(W_o * x_t + U_o * h_(t-1) + b_o)$ spezifiziert ist. Die Definition der Gewichtung, der Bias und die Aktivierungsfunktion bleiben gleich zu denen der beiden anderen Gates. Die Aufgabe des Output-Gates ist es zu kontrollieren, wie viel des Zellzustandes in den Hidden-State überführt wird. Neben den Gates ist der im Absatz erwähnte Zellzustand zu erwähnen, dieser definiert sich über C_t, der Zellzustand wird über das gesamte LSTM aktualisiert aus einer Kombination des vorherigen Zellzustandes mit den Werten des Input-Gates, gewichtet durch den Wert des Forget-Gates. Dieser beschriebene Mechanismus erlaubt es dem LSTM selektiv Informationen zu speichern und diese bei

Bedarf zu vergessen.

Gated Recurrent Units GRU Gated Recurrent Units, auch als GRU bezeichnet, sind ein Teilbereich von RNNs. GRU stellen eine logische Weiterentwicklung der im vorherigen Absatz definierten LSTMs dar. Ursprünglich wurde GRU von Kyunghyun Cho et al. im Jahr 2014 entwickelt, um eine Simplifizierung gegenüber LSTMs zu erreichen [10]. Eine solche Simplifizierung wird dadurch ermöglicht, dass weniger Parameter sowie Gates zum Einsatz kommen, was GRU einen zeitlichen Vorteil bei der Berechnung gegenüber LSTMs gibt.

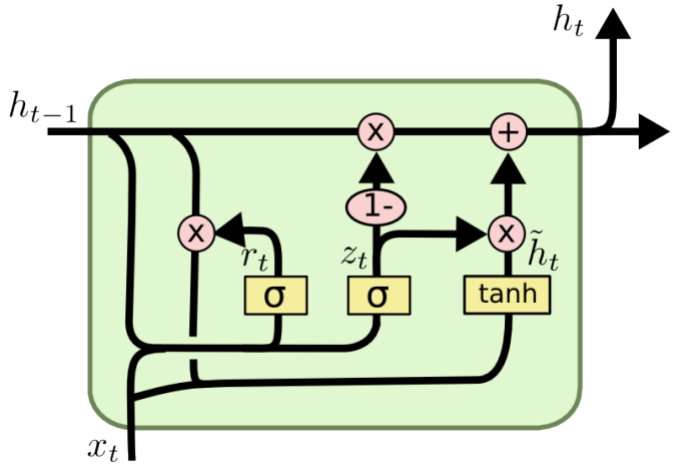


Abbildung 6: Beispiel einer GRU-Architektur [1]

In Abbildung 6 zu sehen ist der beispielhafte Aufbau einer GRU-Architektur. Die Architektur basiert auf einem ähnlichen Gate-System, wie es auch bei LSTM zum Einsatz kommt. Mit der Ausnahme, dass hier zwei statt drei Gates verwendet werden. Im Detail setzen GRU auf den Einsatz eines Update-Gates und eines Reset-Gates. Die Gates sind wie bei LSTM für die Regelung des Informationsflusses zuständig [1].

Das Update-Gate trägt innerhalb des Konstruktes die Verantwortung darüber zu bestimmen, welche Informationen aus dem vorherigen Hidden-State und wie viele Informationen aus dem aktuellen State übernommen werden. Dadurch lernt das GRU ähnlich dem LSTM über Zeit, welche der aktuellen Eingaben es wert sind gespeichert zu werden. Das Update-Gate wird definiert als $z_t = \sigma(W_z * x_t + U_z * h_(t-1) + b_z)$, dabei steht σ für die Sigmoid-Aktivierungsfunktion, die alle Werte auf einen Index zwischen 0 und 1 abbildet. W_z, U_z ist die Gewichtung des Netzwerkes, b_z stellt den Bias des Netzwerkes dar. Auf der anderen Seite steht das Reset-Gate, was ähnlich dem Update-Gate funktioniert, mit der Differenzierung, dass hier entschieden wird, wie viel an Informationen vergessen werden sollen.

Mittels Reset-Gate ist es dem GRU demnach möglich, nicht relevante Informationen zu entfernen. Das Reset-Gate ist in der Abbildung 6 zu finden unter der Bezeichnung r, dieses wird formalisiert als $r_t = \sigma(W_r * x_t + U_r * h_(t-1) + b_r)$. Hierbei kommt, identisch zum

Update-Gate eine Sigmoid Aktivierungsfunktion zum Einsatz. W_r, U_r und b_r stehen wieder für die Gewichtung, respektive den Bias im Netzwerk. Darüber hinaus verwenden GRUs einen vereinheitlichten Zustand, den sogenannten Hidden-State. Im Gegensatz zu LSTMs, die einen separaten Cell-State und Hidden-State haben, kombinieren GRUs diese Informationen in einem einzigen Zustand. Während des Aktualisierungsprozesses in einem GRU wird dieser Hidden-State unter Verwendung der tanh-Aktivierungsfunktion sowie der Gates modifiziert [9].

Die Abbildung 6 zeigt diesen Zwischenzustand als h an, dieser kann formalisiert werden durch $h_t = \tanh(W * x_t + U * (r_t * h(t-1)) + b)$. Hierbei definiert r_t den Wert des Reset-Gates, das mit dem vorherigen Hidden-State multipliziert wird. Von diesem Zwischenzustand wird auf den eigentlichen finalen Hidden-State, in der Abbildung mit h deklariert, übergeleitet. Formalisiert dargestellt als $h_t = (1 - z_t)h_(t-1) + z_t * h_t$, dieser ist eine Kombination des im vorherigen Schrittes berechneten Zwischenzustandes mit dem vorherigen früheren Hidden-State. Die Kombination der beiden Zustände wird mittels des Update-Gates z_t per Gewichtung abschließend gesteuert.

2.3 Verfahren und Methoden

Im nachfolgenden Kapitel werden die für diese Ausarbeitung wichtigsten Methoden Value-Decomposition Networks VDN, Mixed Q-Learning QMIX und Value-Decomposition Networks for Actor-Critic VDAC vorgestellt. Diese werden im Detail erläutert, das Verständnis ist relevant für den experimentellen Absatz der Ausarbeitung.

2.3.1 Value-Decomposition Networks VDN

In dem nun folgenden Kapitel wird in die Thematik des Value-Decomposition Networks, das mit VDN abgekürzt wird, eingeführt. Eines der Kernelemente der Methode ist es, dass unter Einsatz von MARL mit nur einem einzigen gemeinsamen Belohnungssignal, statt eines Belohnungssignales pro Agenten gearbeitet wird. Ein bekanntes Problem des VDN ist es, dass falsche Belohnungen ausgeschüttet werden können, insbesondere dann, wenn eine Belohnung ausgegeben wird, die eigentlich durch einen anderen Agenten verdient wurde [6][27]. Laut den Autoren liegt der Ansatz zur Lösung darin, dass jeder Agent mittels eines eigenen VDN trainiert wird. Das Netzwerk lernt die gemeinsame Aktionswertfunktion Q in individuelle Aktionswertfunktionen der einzelnen Agenten zu zerlegen. Ein von den Autoren genanntes Praxisbeispiel wäre die Steuerung von Verkehrssignalanlagen. Ein weiteres unter MARL bekanntes Problem, das unter dem Namen „faules Lernen“ bekannt ist, kann ebenfalls durch die Methode verbessert werden. Beim faulen Lernen handelt es sich um ein Phänomen, bei dem ein Agent einer Teamkonstellation vom Lernen einer für sich individuell sinnvollen Strategie abgehalten wird, weil sie zu einer schlechteren Teamleistung führen würde [36].

VDN basiert auf Grundsätzen des DQN, dieses wurde bereits in einem der vorherigen Abschnitte näher erläutert. Hierbei wird auf den e-Greedy-Ansatz zurückgegriffen, wobei bei $1 - e$, mit der Wahrscheinlichkeit $argmax_a(Q_i(s, a))$ eine Aktion ausgewählt oder bei e eine zufällige Aktion durchgeführt werden soll. Darüber hinaus verwendet VDN Zielnetzwerke

sowie Erfahrungsreplays, welche aus dem Bereich DQN stammende Methodiken sind. Zusätzlich kommen Techniken wie das rekurrente Netzwerk zum Einsatz, das unter zu Hilfenahme einer Dueling Architektur [42] den Lernprozess beschleunigen soll.

Das Kernprinzip des VDN Ansatzes besteht darin, dass das Value-Decomposition Network im Hinblick auf eine lineare Wertzerlegung aus dem Teambelohnungssignal optimiert wird. Hinzu kommt der Ansatz der additiven Wertzerlegung zum Einsatz, dieser bietet laut den Autoren den Vorteil, falsche Belohnungssignale zu vermeiden, die beim unabhängigen Lernen von mehreren Agenten auftreten können [36]. Dem Ansatz liegt die zentrale Annahme der Autoren zugrunde, dass es möglich ist die gemeinsame Aktionswertfunktion additiv in Wertfunktionen für jeden einzelnen Agenten zu zerlegen. Diese Annahme kann formalisiert wie folgt dargestellt werden:

$$Q((h^1, h^2, ..., h^d), (a^1, a^2, ..., a^d)) \approx \sum_{i=1}^{d} \tilde{Q}_1(h^i, a^i)$$

Der linke Abschnitt der Formel $Q(h^1, h^2, h^d), (a^1, a^2, a^d)$ beschreibt den zu erwartenden Gesamtwert, welchen man erhält, sollte eine bestimmte Aktion $a^1, a^2, ..., a^d$ ausgeführt werden. Dabei wird von einem Zustand, der durch eine Sequenz von Zuständen $(h^1, h^2, ..., h^d)$ beschrieben wird, ausgegangen. Hierbei gilt, dass Q für den Gesamtwert, $a1, a2...$ für die Aktionen der einzelnen Agenten und $h1, h2$ für den aktuellen Zustand der Agenten steht. Der zweite Abschnitt der Formel $\tilde{Q}_i(h_i, a_i)$ gibt an, wie geeignet eine bestimmte Aktion a^i bei Betrachtung eines bestimmten Zustandes h^i ist, wobei nur ein Agent einzeln betrachtet wird. Anschließend werden alle Einzelwerte summiert. Die Formel versucht den Gesamtwert der Q-Funktion zur Summe an Einzelwert Q-Funktionen jedes Agenten anzunähern und sagt damit im Kern aus, dass die Teamwertfunktion ungefähr der Summe der Einzelwertfunktionen entsprechen sollte.

Netzwerk Architektur Zur Realisierung der Zerlegung der Teamwertfunktion werden Techniken aus dem Bereich des DQN herangezogen. Die Architektur stützt sich dabei auf Erfahrungsreplay und Zielnetzwerke, ergänzt durch LSTMs. Des Weiteren werden Backpropagation und Lernen mit mehrstufige Aktualisierungen mit vorwärts gerichteten Eignungsspuren (forward view eligibility traces) sowie der Einsatz einer „Dueling"-Architektur berücksichtigt. Ein exemplarisches Schema dieser Netzwerkarchitektur ist in Abbildung 7 zu sehen.

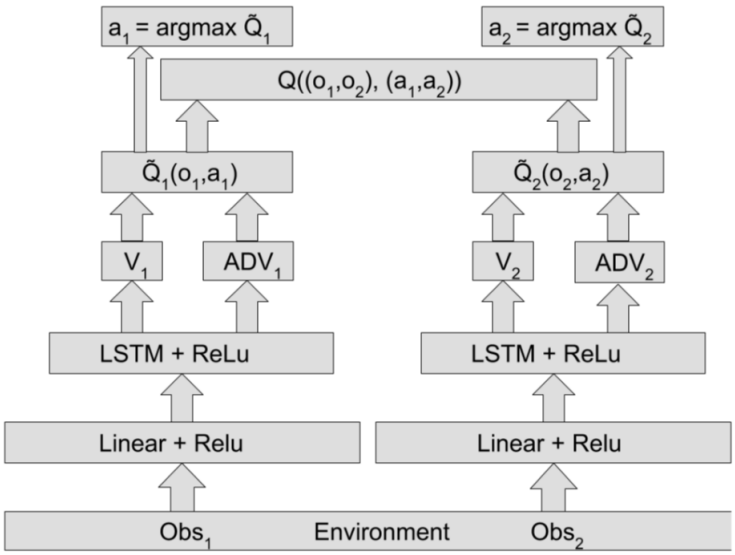


Abbildung 7: Beispiel einer möglichen VDN KNN-Architektur [36]

Die Basis des in Abbildung 7 gezeigten Aufbaus des neuronalen Netzes kann im Detail wie folgt beschrieben werden:

1. 32 Hidden-Layer, vollverbunden und linear + ReLu-Aktivierungsfunktion
2. 32 Hidden-Units LSTM + ReLu-Aktivierungsfunktion
3. Linearer „Dueling"-Layer mit 32 Units, der $V(s)$ und $A(s, a)$ erzeugt.

Dieser Abschnitt des neuronalen Netzwerks erzeugt im Ergebnis somit zwei Produkte, eine Wertefunktion $V(s)$ und eine Vorteilfunktion $A(s, a)$, welche schlussendlich verwendet werden, um die Q-Funktion zu berechnen. Die Q-Funktion berechnet sich durch die Formel $Q(s, a) = V(s) + A(s, a)$. Die so durch das Netzwerk erzeugten Werte werden abschließend nach $argmaxQ_n$ gefiltert, ausgegeben bzw. über Q summiert. Zusammenfassend, zeigt die im oberen Abschnitt zu sehende Abbildung bildlich den Aufbau des unter VDN verwendeten neuronalen Netzwerkes.

2.3.2 Mixed Q-Learning QMIX

Im nachfolgenden Kapitel wird die Methode QMIX eingeführt, die Abkürzung steht für „Q-Learning with mixed networks". Die folgende Einführung ist inspiriert von den Autoren Rashid et al. [30]. QMIX stellt eine der verwendeten Algorithmen im Rahmen dieser Ausarbeitung dar. Dieser kann als eine Erweiterung des im vorherigen Absatz erläuterten VDN-Ansatzes betrachtet werden. Hierbei wird wie bei VDN auf eine gemeinsame Wertfunktion gesetzt, mit dem Unterschied, dass diese bei QMIX als monotone Funktion repräsentiert wird und somit reichhaltigere Wertfunktionen abgebildet werden können [30].

Grundlegend handelt es sich bei QMIX um eine wertebasierte Methode, bei der es unter anderem möglich ist, dass dezentrale Policys zentralisiert trainiert werden können. Dadurch ist es zum Beispiel möglich den Agenten innerhalb des Trainingsprozesses mehr Information zuteilwerden zu lassen, als im späteren Anwendungsszenario üblich ist. Beispielsweise könnte es sich um vollständige Umgebungsvariablen wie Koordinaten oder eine uneingeschränkte Kommunikation der Agenten untereinander handeln. Das Schema unterliegt hierbei dem CTDE-Ansatz, der bereits in den vorherigen Kapiteln erläutert wurde. Dabei unterscheidet sich diese Methode im Vergleich zu VDN darin, dass keine vollständige Faktorisierung stattfinden muss, um eine dezentralisierte Policy zu erhalten und damit dem CTDE-Ansatz Genüge zu tun [30].

Die Methode QMIX hat in mehreren Anwendungen, wie z. B. in der Verkehrssteuerung, gute Ergebnisse gezeigt [29]. Sie ermöglicht es Agenten gemeinsam eine Aufgabe zu lösen, indem sie ihre individuellen Lernprozesse miteinander kombinieren. Dies hat gezeigt, dass diese Methode eine bessere Leistung als andere MARL-Algorithmen erreichen kann [29]. Darüber hinaus verwendet QMIX im Gegensatz zu VDN eine namensgebende Mixing Network Architektur zur Kombination der individuell ermittelten Aktionwertfunktionen. Ein Unterschied zu klassischen VDNs liegt darin, dass QMIX nicht darin begrenzt ist, dass während des Trainings nur partiell Zusatzinformationen verwendet werden können. Stattdessen wird lediglich darauf geachtet, dass ein $argmax$ gegen Q_{tot} die gleichen Ergebnisse liefert wie ein $argmax$ gegen jedes Q_a [30]. Formalisiert wird QMIX daher wie folgt dargestellt:

$$argmax Q_{tot}(\tau, u) = \left(argmax_{u^1} Q_1(\tau^1, u^1) ... argmax_n Q_n(\tau^n, u^n)\right)$$

Als weitere Abgrenzung zu VDNs verwendet QMIX monotone Funktionen. Um diese Monotonie zu gewährleisten, wird folgender Zustand erzwungen:

$$\frac{\partial Q_{tot}}{\partial Q_a} \geq 0, \forall a$$

QMIX baut auf einer Kombination verschiedener neuronaler Netze auf, welche im nachfolgenden Absatz im Detail erläutert werden. Dabei handelt es sich um dedizierte Agentennetzwerke, ein Mixing Netzwerk sowie mehreren Hyper-Netzwerken. Das stellt einen weiteren der Unterschiede zu VDN dar, wo vergleichsweise mit einer Summierung gearbeitet wird, wird bei QMIX mit Mixing-Netzwerken gearbeitet.

Netzwerk-Architektur von QMIX In Abbildung 8 ist der exemplarische Aufbau der gesamten Netzwerkarchitektur des QMIX-Algorithmus zu sehen. Der Aufbau folgt einer Kombination mehrerer aufeinander aufbauender neuronaler Netzwerke, es beginnt wie in der Abbildung unter Abschnitt c) zu sehen ist mit einem dedizierten Agentennetzwerk, eines pro eingesetztem Agenten im MARL Szenario. Das Agentennetzwerk selbst basiert in seinen Grundzügen auf der Struktur von DQN Netzwerken, welche im vorherigen Grundlagenkapitel in ihren Grundzügen erläutert wurden. Das Agentennetzwerk liefert im Ergebnis die Agenten individuelle Wertefunktion $Q_a(\tau^a, u^a)$. Der Input wird definiert als o_t^1, was die individuelle Beobachtung eines Agenten in einem Zeitintervall t beschreibt und u_{t-1}^1, was die letzte ausgewählte Aktion eines Agenten definiert.

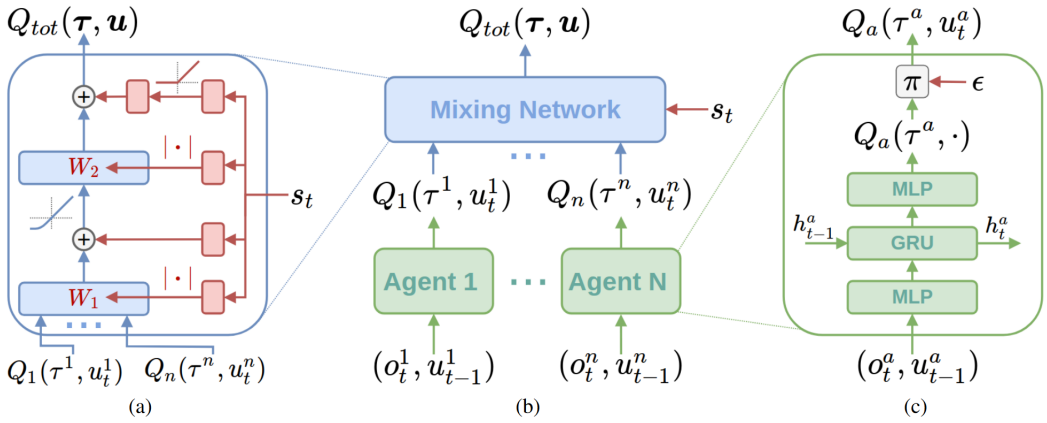


Abbildung 8: QMIX Architektur a) Zeigt das Mixing Netzwerk, in Rot eingezeichnet die Hyper-Netzwerke. b) Darstellung der Allgemeinen QMIX Architektur c) Netzwerk der Agenten [30]

Die aus dem Agentennetzwerk gewonnene individuelle Wertefunktion dient als Input für das Mixing-Netzwerk. Das Mixing-Netzwerk selbst ist ein nach DQN Standard definiertes „feed forward“ Netzwerk, das heißt ein Netzwerk mit einem einfachen nach vorne gerichteten Durchlauf, ausschließlich in eine Richtung. Dieses Netzwerk dient dem Zweck, die eingehenden individuellen Agentenwertfunktionen zu kombinieren, im Ergebnis erzeugt das Mixing-Netzwerk durch monotonisches Vermischen den Wert Q_{tot}. Im Detail zu sehen ist das Mixing-Netzwerk in Abbildung 8, Abschnitt a). Hier ist zudem zu erkennen, dass zur Wahrung der Monotonie die Gewichtung der Knoten des Netzwerkes nur positive Werte abbilden können. Die Gewichtung innerhalb des Mixing-Netzwerkes der einzelnen Knoten wird wiederum durch sogenannte Hyper-Netzwerke erzeugt. Die Hyper-Netzwerke basieren auf einer einfachen linearen Ebene, folgend zu einer absoluten Aktivierungsfunktion. Absolut deshalb, damit die Werte der Gewichtung rein positiv sind. Die Hyper-Netzwerke erzeugen als Ergebnis einen Vektor, der laut den Autoren zur Verarbeitung in eine Matrix mit angemessener Größe umgewandelt werden muss [30].

Der Bias des Mixing-Netzwerkes ist ähnlich wie die Gewichtung ebenfalls mittels Hyper-Netzwerken definiert. Wobei hier keine absolute Aktivierungsfunktion zum Einsatz kommt und somit der Wert des Bias nicht ausschließlich positiv sein muss. Der finale Bias, der den abschließenden Knoten des Netzwerkes darstellt, wird abweichend zu der vorherigen Definition wie folgt erzeugt. Zum Einsatz kommt ein Hyper-Netzwerk, bestehend aus zwei Ebenen, statt einer absoluten Aktivierungsfunktion wird eine ReLu-Aktivierungsfunktion eingesetzt. So auch zu sehen, in rot gekennzeichnet in Abbildung 8, Abschnitt a). Das gesamte QMIX-Netzwerk wird darüber hinaus trainiert, indem die nachfolgende Verlustfunktion minimiert wird:

$$\mathcal{L}(\theta) = \sum_{i=1}^{b} \left[y_i^{tot} - Q_{tot}(\tau, u, s; \theta))^2 \right]$$

Wobei sich y_i^{tot} spezifiziert als $r + \gamma max_{u'} Q_{tot}(\tau', u', s'; \theta^-)$. Der Paramater des Zielnetzwerkes ist θ, darüber hinaus definiert b die für das Training verwendete Batch-Größe. Alle anderen Variablen der Funktion folgen der allgemeingültigen Definition des DQN. Die Autoren stellen ebenso fest, dass diese eingesetzte Verlustfunktion den Standard aus DQN widerspiegelt [30].

Zusammenfassend zeigt sich, dass QMIX eine Erweiterung des VDN Ansatzes darstellt, indem eine Veränderung der Netzwerkarchitektur vorgenommen wurde, um eine größere Facette an wertebasierten Funktionen abbilden zu können. Im nun folgenden Absatz wird die Methode VDAC vorgestellt, welche einen weiteren Aspekt des VDNs optimiert.

2.3.3 Value-Decomposition Networks for Actor-Critic VDAC

Value Decomposition for Actor-Critic, im folgenden abgekürzt mit VDAC, ist eine Methode, welche im Bereich MARL ihre Anwendung findet. Hierbei ist die nun folgende Einführung in den Algorithmus inspiriert von der Darstellung von Su et al. [35]. Bei VDAC handelt es sich um eine Erweiterung der in den vorherigen Abschnitten erläuterten QMIX und VDN Ansätze durch das Prinzip des Actor-Critic. Dieses definiert sich charakteristisch darüber, dass der Wert einer Aktion in einen Wert des aktuellen Zustands und einen Wert der Aktion selbst zerlegt wird. VDAC nutzt hierzu eine sehr ähnliche Architektur wie VDN bzw. QMIX. Hierbei wird der bestehende Ansatz durch die Definition eines Critic-Netzwerks und eines Actor-Netzwerks erweitert. Das Critic-Netzwerk wird verwendet, um den Wert einer Aktion in einem gegebenen Zustand zu schätzen, während das Actor-Netzwerk verwendet wird, um die beste Aktion in einem gegebenen Zustand zu bestimmen [35]. VDAC basiert laut den Autoren auf drei Kernaspekten:

1. VDAC ist kompatibel mit dem Advantage-Actor-Critic Framework (A2C)
2. VDAC erzwingt wie QMIX eine Beziehung zwischen $\frac{\partial Q_{tot}}{\partial Q_a} \geq 0, \forall a$
3. VDAC verwendet temporal-difference policy gradients zur Lösung.

Aus den genannten Kernaspekten ergeben sich einige Vorteile gegenüber dem Einsatz von QMIX. Auf der einen Seite wird durch die Unterstützung von A2C gewährleistet, dass das Training als effizient eingestuft werden kann. Der Trainingsprozess ist also verbessert. Auf der anderen Seite wird zum Training eine im Bereich des DQN sehr bekannte „policy gradient"-Methode eingesetzt, welche „temporal differences" [38] genannt wird. Dabei wird das aus der QMIX bekannte Mixing-Netzwerk innerhalb des VDAC als Critic betrachtet [35].

Im Kontrast zu QMIX und VDN, zwei bereits in vorherigen Kapiteln erläuterte Methoden, setzt VDAC darauf, globale Zustandswerte zu zerlegen. Gegensätzlich zu QMIX und VDN, welche sich auf die Zerlegung von globalen Aktionswerten spezialisiert haben. Konkret fügt VDAC dem Actor-Critic einen sogenannten lokalen Critic hinzu. Die Unterschiede zu QMIX unter Berücksichtigung des lokalen Critic, zeigen sich im Detail und werden im nachfolgenden Absatz näher beleuchtet.

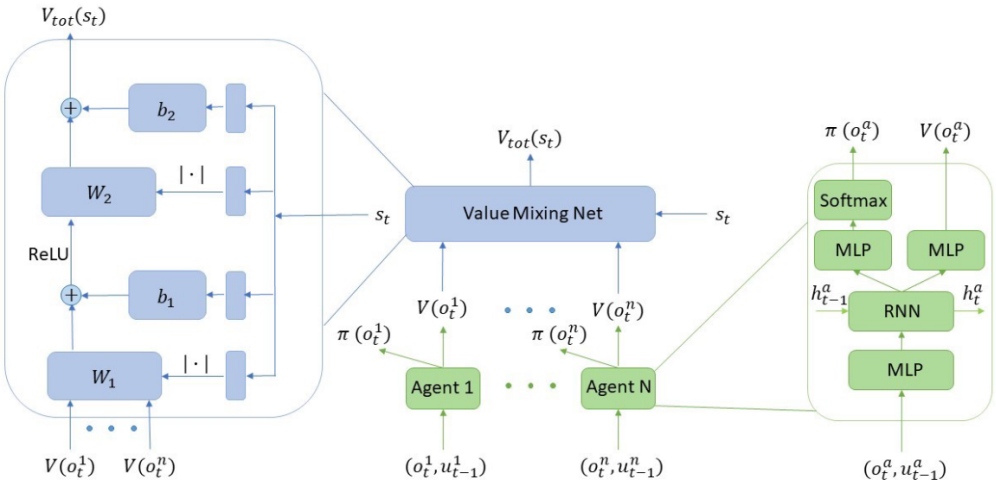


Abbildung 9: VDAC-Architektur [35]

Architektur von VDAC Die Autoren der Methode VDAC präsentieren zwei Varianten, eine Version VDAC-sum, welche den reinen VDN-Ansatz erweitert, sowie VDAC-vmix, welche QMIX um A2C ausdehnt. Im Rahmen des praktischen Abschnittes der Ausarbeitung kommt VDAC-vmix zum Einsatz. In Abbildung 9 ist der strukturelle Aufbau des VDAC-vmix-Ansatzes zu sehen. Hierbei zeigt sich, dass der Aufbau sehr ähnlich der in Abbildung 8 gezeigten QMIX-Architektur ist. Daher ist auch der grundlegende Aufbau identisch, mit der Ausnahme, dass VDAC auf sogenannte lokale Critic setzt, zu sehen in Abbildung 9 auf der rechten Seite.

Das Agentennetzwerk oder in diesem Fall auch Actornetzwerk genannt, erhält wie bei QMIX als Input Werte, die aktuelle Beobachtung des Agenten, sowie die im vorherigen Schritt durchgeführte Aktion. Im Ergebnis erzeugt das Actornetzwerk dann allerdings zwei Output Werte. Zum einen wird der Wert $V(o_t^a)$ ausgegeben, dieser repräsentiert die Zustandswertefunktion für einen Agenten a im Zeitintervall t für seine lokale Beobachtung o. Somit spiegelt dies die erwartete Gesamtbelohnung für den einzelnen Agenten im aktuellen Zustand wider. Des Weiteren gibt $\pi(o_t^a)$ die Policy des Agenten an, die Parameter o, a, t folgen der gleichen Definition wie $V(o_t^a)$. Der VDAC-vmix-Ansatz verfolgt hierbei das Ziel zur Minimierung der nachfolgenden Verlustfunktion:

$$L_t(\theta_v) = \left(y_t - V_{tot}(s_t)\right)^2$$

Wobei sich $V_{tot}(s_t)$ definiert über $f_{mix}((V_{0_v}(o_t^1), ..., V_{0_v}(o_t^n)))^2$. Zu bedenken ist, dass f_{mix} das gesamte Mixing-Netzwerk umfasst. $L_t(\theta_v)$ beschreibt die Verlustfunktion im Zeitintervall t in Abhängigkeit von θ_v. Darüber hinaus bezeichnet y_t die Wertefunktion im Zeitintervall t. Insgesamt wird das Ziel verfolgt, die Verlustfunktion wie auch bei QMIX im Verlauf des Trainingsprozesses zu minimieren. Dadurch wird dem erlernten Modell ermöglicht, eine Annäherung an die tatsächlich erwünschte Wertefunktion zu erlernen, was zu einem allgemein

besseren Lernergebnis führen sollte [35].

Advantage-Actor-Critic A2C Das im Rahmen dieser Arbeit verwendete VDAC-Verfahren basiert auf der in diesem Abschnitt erläuterten Technologie des Actor-Critic mit der Erweiterung des Advantage-Actor-Critic (A2C). Das Verfahren unterscheidet sich vom herkömmlichen Actor-Critic durch die Erweiterung um das Konzept der Vorteilsschätzung zur Handlungsbewertung. Dieses Design beabsichtigt einige der Herausforderungen, die mit der herkömmlichen Actor-Critic-Methode verbunden sind, wie hohe Schwankungen und möglicherweise langsame Annäherung, zu überwinden [19]. Dadurch wird A2C in vielen Szenarien eingesetzt um zu einer stabileren und effizienteren des Trainings beizutragen [23].

3 Klassifizierung der Umgebung

In dem nun folgenden Teil des praktischen Abschnittes dieser Arbeit wird zuerst auf die verwendete Umgebung und die daraus resultierenden Besonderheiten eingegangen. Darauffolgend werden die Trainingsgegebenheiten näher betrachtet mit Fokus auf den verwendeten Parametern während der jeweiligen Trainingsessions.

3.1 Pommerman

Das in dieser Arbeit verwendete Szenario ist „Pommerman“. Hierbei handelt es sich um eine speziell für das Testen von Reinforcement-Learning-Algorithmen konzipierte Umgebung. Das Regelwerk und der grundlegende Aufbau sind dem 1983 von Nintendo veröffentlichten „Bomberman“ nachempfunden. Das Spielfeld von Pommerman kann entweder die Größe eines 8x8 oder 11x11 Rasters haben, abhängig vom gewählten Modus während der Initialisierung. Das 8x8 Raster wird in dieser Arbeit nicht berücksichtigt, da es primär für den Einsatz von Einzelbegegnungen zwischen zwei Agenten entwickelt wurde. Das in Abbildung 10 zu sehende Spielfeld der Größe 11x11 zeigt das Standardspielfeld für Teambegegnungen und wird so in dieser Ausarbeitung für Training, Validierung und Begegnungen zum Einsatz kommen.

Abbildung 10: Pommerman Spielfeld im 11x11 Raster, Modus „PommeTeamCompetition“

Die Umgebung bietet neben den standardmäßigen unterschiedlichen Spielfeldgrößen auch verschiedene Spielmodi an. Mithilfe dieser Modi kann gesteuert werden, ob die Begegnungen kooperativ oder kompetitiv verlaufen. Tabelle 1 listet alle verfügbaren Modi auf.

Bezeichnung	**Begegnung**	**Bedeutung**
PommeFFACompetition	Kompetitiv	Jeder gegen Jeden
OneVsOne	Kompetitiv	Agent gegen Agent
PommeTeamCompetition	Kooperativ / Kompetitiv	Team gegen Team

Tabelle 1: Verschiedene Modi Pommerman

Der gewählte Modus bestimmt nicht nur die Konstellation der Agenten, sondern auch, wie viele Agenten an einer Partie teilnehmen. Der Modus „PommeFFACompetition“ impliziert, dass eine Partie zwischen vier Agenten ausgetragen wird. Dabei stellt jeder der vier Agenten ein eigenständiges Team dar. Folglich tritt jeder Agent gegen jeden anderen Agenten an, und der zuletzt überlebende Agent gewinnt die entsprechende Partie. Der Modus „OneVsOne“ hingegen definiert, dass auf dem kleineren der beiden Spielfelder zwei Agenten direkt gegeneinander antreten. Auch hier gewinnt der zuletzt überlebende Agent die Partie. Der letzte mögliche Modus ist „PommeTeamCompetition“, welcher im Rahmen dieser Arbeit ausschließlich zum Einsatz kommt. Dieser Modus nutzt das 11x11 Raster-Spielfeld, wobei zwei Teams gegeneinander antreten. Der Spielmodus ist in Abbildung 10 dargestellt. Jeweils zwei Agenten pro Team sind in Rot und Blau repräsentiert. Die Zuweisung der Agenten zu einem Team erfolgt nicht zufällig, sondern kann per Konfiguration festgelegt werden. Eine Partie endet entweder, wenn kein Agent eines Teams mehr am Leben ist, oder wenn die vorher konfigurierte maximale Rundenanzahl erreicht wurde. Die so definierte Partielänge sowie

weitere Einstellungsmöglichkeiten können in Pommerman angepasst werden. Eine Auswahl an möglichen Konfigurationsparametern, sowie die dafür vorgesehen Standardkonfiguration ist in der Tabelle 2 zu sehen.

Bezeichnung	**Bedeutung**	**verwendete Einstellung**
numrigid	Anzahl unzerstörbarer Wände	64
numwood	Anzahl zerstörbarer Wände	32
numitems	Anzahl Bonusgegenstände	0
maxsteps	Anzahl maximaler Runden	800
agentviewsize	Sichtfeld des Agenten	5
ispartiallyobservable	Aktivierung Sichtfeldeinschränkung	True

Tabelle 2: Konfigurationsmöglichkeit einer Pommerman-Begegnung

Für jede neue Partie wird ein zufälliges Spielfeld, basierend auf den per Konfiguration definierten Parametern erstellt. Die in dieser Arbeit ausgewählten Einstellungen der Begegnungen folgen grundsätzlich den Standardeinstellungen welche durch Pommerman vorgegeben sind, mit Ausnahme der Anzahl an Bonusgegenständen. In dieser Arbeit wurde darauf verzichtet, dem Spiel Bonusgegenstände hinzuzufügen, da diese das Spielgeschehen auf unvorhersehbarer Art und Weise verändert können. Das zufällige Erstellen in diesem Kontext meint die Anordnung der jeweiligen Elemente, wie zerstörbare und unzerstörbare Wände. Im Detail wird eine Runde, wie in Tabelle 2 zu sehen ist, initialisiert mit 64 unzerstörbaren Wänden, sowie 32 zerstörbaren Wänden. Die maximale Rundenanzahl pro Partie beläuft sich auf 800. Wird mit Erreichen dieser Rundenanzahl kein eindeutiger Gewinner ermittelt, gilt die Begegnung als unentschieden. Zudem ist die Sicht der Agenten auf einen Radius von 5 Einheiten limitiert, diese Limitierung tritt durch die Aktivierung der Sichtfeldeinschränkungen inkraft, somit gilt das Szenario als POMDP. In Abbildung 11 wird diese Einschränkung des Sichtfeldes der Agenten dargestellt. Die Abbildung zeigt von oben nach unten die Ansicht eines jeden Agenten einzeln, wobei der grau dargestellte Bereich anzeigt, was nicht in den Betrachtungsradius fällt.

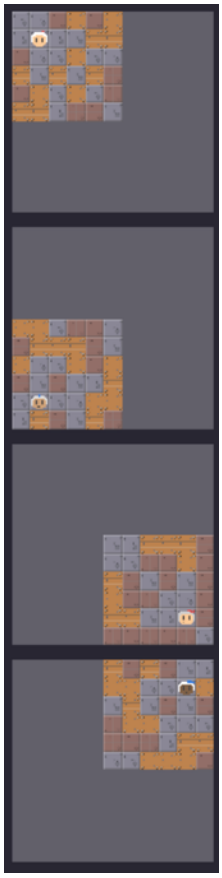


Abbildung 11: Sichtfeldeinschränkungen in Pommerman, Ansicht pro Agenten

Aus der Sichtweise des zu trainierenden Agenten ist jedem Spielelement auf der Spieloberfläche ein eindeutiger Zahlenwert zugeordnet. In Abbildung 12 sind diese Zahlenwerte zu sehen, welche die aktuelle Runde für einen Agenten darstellen. Hierbei zu beachten ist, dass zur besseren Darstellbarkeit die Abbildung keinen eingeschränkten Sichtradius enthält. Jeder dieser Zahlenwerte referenziert ein bestimmtes Element des Spielfeldes, wobei die wichtigsten wie folgt lauten:

Zahlenwert	**Bezeichnung**
0	offener Durchgang
1	unzerstörbare Wand
2	zerstörbare Wand
3	Bombe
4	Feuer

Tabelle 3: Wertedefinition der Agenten spezifischen Sichtweise

	0	1	2	3	4	5	6	7	8	9	10
0	0	1	1	2	2	2	1	2	1	1	1
1	1	4	4	0	2	2	2	0	0	0	1
2	1	4	0	0	0	2	1	0	1	0	0
3	2	0	0	0	2	2	0	0	1	0	0
4	2	2	0	2	0	0	1	1	2	2	1
5	2	2	2	2	0	0	1	0	2	2	2
6	1	2	1	0	1	1	0	1	2	2	2
7	2	0	0	0	1	0	1	0	1	0	0
8	1	0	1	1	2	2	2	1	0	0	1
9	1	0	0	0	2	2	2	0	0	3	1
10	1	1	0	0	1	2	2	0	1	1	0

Abbildung 12: Exemplarische Agenten spezifische Sichtweise des Spielfeldes

Die in Tabelle 3 dargestellten Elemente dienen den Agenten als Orientierungshilfe sowie zur Erkennung von eventuell vorhandenen Gefahren. Eine Besonderheit an „Pommerman“ im Vergleich zu anderen Reinforcement-Learning-Szenarien liegt in dem Umstand, dass es den Agenten möglich ist, sich selbst oder einem Teammitglied Schaden zuzufügen. Ein Agent kann nur einmal pro Partie Schaden erleiden und wird daraufhin sofort aus dem Spiel genommen.

Ein Agent kann auf zweierlei Arten Schaden erleiden. Die erste Möglichkeit ist, dass ein Agent sich zum Zeitpunkt der Explosion einer gelegten Bombe innerhalb des Explosionsradius befindet. Die zweite Möglichkeit für einen Agenten Schaden zu nehmen ist es, wenn er sich in ein auf der Karte existierendes Feuer bewegt. Eine Bombe explodiert nach einer fest definierten Zeitspanne von zehn Runden. Die Bombe wird, wie in Tabelle 3 zu sehen ist, mit dem Zahlenwert 3 für den Agenten sichtbar. Der Agent kann allerdings den Explosionsradius selbst nicht sehen. Der Radius beläuft sich auf die Position der gelegten Bombe plus einen Block in jede Richtung, sowohl horizontal als auch vertikal. Dadurch, dass keine Bonusgegenstände vorhanden sind, ist dieser Radius fest und kann nicht verändert werden. Zudem ist es möglich, dass es durch die Explosion einer Bombe zu einer weiteren Explosion kommen kann, sollte sich in ihrem Explosionsradius eine weitere Bombe befinden. In diesem Fall wird das Zeitintervall von zehn Runden ignoriert.

Das Feuer ist für den Agenten, wie in Tabelle 3 gezeigt, durch den Zahlenwert 4 ersichtlich. Feuer bleibt für ein fest definiertes Zeitintervall von fünf aufeinander folgenden Runden innerhalb des vorherigen Explosionsradius bestehen. Weder die Explosion der Bombe noch das Feuer unterscheiden zwischen den Agenten. Deshalb ist es in beiden Fällen möglich, sowohl sich selbst als auch sein Teammitglied aus dem Spiel zu nehmen.

3.1.1 Aktionen

Die Pommerman-Agenten haben die Möglichkeit, in jeder Spielrunde maximal eine von sechs verschiedenen Aktionen auszuwählen. Die Spielrunde endet, nachdem jeder Agent eine Aktion ausgeführt hat. Tabelle 4 listet alle möglichen Aktionen auf.

Zahlenwert	Bezeichnung
1	Bewegung hoch
2	Bewegung runter
3	Bewegung links
4	Bewegung rechts
5	Bombe legen

Tabelle 4: Mögliche Aktionen der Agenten

Die Aktionen 1 bis 4 dienen zur direkten Navigation der Agenten im Spielfeldraster. Bei Auswahl der jeweiligen Aktion wird die Position des Agenten um einen Block in die entsprechende Richtung versetzt. Wird die Aktion 0 von einem Agenten gewählt, verbleibt der Agent an seiner aktuellen Position. Die Auswahl der Aktion 5 führt dazu, dass der Agent eine Bombe genau an seiner aktuellen Position platziert. Jeder Agent kann nur eine einzelne Bombe platzieren. Solange diese nicht explodiert ist, kann der Agent diese Aktion zwar erneut auswählen, jedoch wird keine weitere Bombe platziert. Stattdessen bewirkt dies, ähnlich wie bei der Aktion 0, keine Positionsveränderung.

3.1.2 Pommerman-Referenzagenten

Pommerman wird mit einer Reihe von Agenten bereitgestellt, wobei zwei im Rahmen dieser Arbeit als Referenzagenten dienen. Diese Referenzagenten werden durch das eingesetzte Framework MARLlib dazu verwendet, die Modelle zu trainieren, indem wahlweise diese Agenten als Gegner fungieren. Diese dienen neben dem Training dazu, eine Referenzlinie festzulegen, anhand derer die Leistung der abschließend trainierten Modelle bewertet werden kann. Hierbei sind zwei Varianten von besonderer Bedeutung. Auf der einen Seite ein Agent, welcher in jeder Runde eine zufällige Aktion aus den in Tabelle 4 verfügbaren Aktionen auswählt. Auf der anderen Seite einen Agenten, der regelbasiert agiert. Dieser setzt unter anderem auf den Dijkstras Wegfindungsalgorithmus zur Navigation, verwendet ein Gedächtnis, um nicht in Ablaufschleifen hängenzubleiben und setzt auf Heuristik, um zu entscheiden, welche der verfügbaren Aktionen in der aktuellen Situation am ehesten verwendet werden sollte. Beispielsweise wird überprüft, ob das Platzieren einer Bombe an der aktuellen Position dazu führen würde, dass sich der Agent selbst einsperrt. Eingesperrt wäre der Agent in dem Fall, wenn er um seine Position herum von Wänden umgeben ist und die Bombe vor ihm platziert wird. Bei beiden Agententypen kommen keine Ansätze des maschinellen Lernens zum Tragen. Hierbei ist zu erwähnen, dass, wie in Tabelle 3 zu sehen, ein Agent sich nur bewegen kann, wenn der Zahlenwert der Position, auf die er sich bewegen will, den Wert 0 aufweist. In allen anderen Fällen scheitert die Bewegung.

3.1.3 Modifikationen von Pommerman

Die Pommerman-Umgebung wurde in einigen Bereichen erweitert. Diese Anpassungen beeinflussen dabei den Ablauf der Umgebung nicht. Insbesondere wurden diese Änderungen vorgenommen, um die im nachfolgenden Absatz beschriebene Belohnungsstruktur zu realisieren. Für jedes Element, wie Feuer, Bomben und Bombenradius, wurden spezifische Identifikationsnummern hinzugefügt, damit klar ist, welches Element welchem Agenten zugeordnet ist. Diese ID besteht primär aus der individuellen Agenten ID, die einzigartig innerhalb einer Spielpartie vergeben wird. Durch diese Erweiterung wurde es möglich, nachzuvollziehen, wessen Bombe beziehungsweise welches daraus resultierende Feuer für die Eliminierung eines bestimmten Agenten verantwortlich ist. Die bestehende Belohnungsstruktur wurde zudem grundlegend überarbeitet, und die spezifischen Änderungen werden im nachfolgenden Absatz detailliert beschrieben. Mit den Erweiterungen können zudem Daten darüber gesammelt werden, welches Team innerhalb einer Iteration wie oft gewonnen oder verloren hat. Diese Daten sind für spätere Vergleiche zwischen verschiedenen Methoden und Referenzagenten von Bedeutung. Darüber hinaus wurde in die Umgebung eine Zwischenspeichermöglichkeit von Rundeninformationen eingebaut. Diese Informationen sind den Agenten während des Trainings und der Ausführung nicht zugänglich. Die Speicherung umfasst die Informationen der letzten zehn Runden der aktuell laufenden Partie, pro Agenten. Die Speicherung der Informationen dient ebenfalls dem Zweck der Optimierung der eingesetzten Belohnungsstruktur.

3.2 Eingesetzte Bibliotheken

Das Training der Agenten im Pommerman-Szenario wird durch die Open-Source-Softwarebibliothek MARLlib realisiert. Laut den Autoren wurde MARLlib speziell für das Training von MARL-Algorithmen entwickelt. Darüber hinaus beinhaltet MARLlib viele bereits integrierte Testumgebungen, einschließlich des Pommerman-Szenarios, das in seinen Grundzügen von MARLlib implementiert wurde. Zudem verfügt MARLlib über eine Reihe von Algorithmen, die im MARL-Segment relevant sind und bereits integriert wurden. Hierzu gehören ebenfalls die in dieser Arbeit verwendeten Algorithmen VDN, QMIX und VDAC. Für die Skalierung und das Tuning von Reinforcement-Learning-Algorithmen verwendet MARLlib [1] unter anderem die ebenfalls als Open Source verfügbare Softwarebibliothek Ray [2], insbesondere dem von Ray bereitgestellte Tool Rllib kommt eine wichtige Bedeutung während des Trainings zu. Die Autoren beschreiben Rllib [3] als eine Bibliothek, die den Lernprozess von Reinforcement-Learning-Algorithmen steuert und überwacht. Grundlegend erweitert MARLlib Rllib um verschiedene Umgebungen, Algorithmen und die Möglichkeit von Multi-Agent-Anwendungen. Die nun folgenden Absätze gehen hierzu im Speziellen auf für das Training notwendige Rahmenbedingungen wie die implementierte Belohnungsstruktur ein.

[1] https://github.com/Replicable-MARL/MARLlib
[2] https://github.com/ray-project/ray
[3] https://docs.ray.io/en/latest/rllib/index.html

3.2.1 Modifikationen MARLlib

Es gab geringfügige Modifikationen, so wurde der MARLlib-Bibliothek unter anderem eine neue Funktion hinzugefügt, die zum Zeitpunkt der Erstellung dieser Arbeit in der Basisinstallation nicht vorhanden war. Hierzu gehört die Modifikation, dass eine bildliche Darstellung von Pommerman ermöglicht wird, was im ursprünglichen Zustand aufgrund von Fehlern in der aktuellen Integration von Pommerman in MARLlib und insbesondere aufgrund fehlerhaft verlinkter und fehlender Assets nicht möglich war. Schließlich wurde die Pommerman-Konfigurationsdatei erweitert, sodass bei der Initialisierung der Umgebung durch MARLlib festgelegt werden konnte, ob das laufende Training dargestellt werden soll oder nicht. Gerade die so hergestellte Beobachtbarkeit der Umgebung ist von besonderer Wichtigkeit, da diese eine manuelle Beobachtung der Agenten im Training und bei Ausführung erst ermöglichte.

3.3 Belohnungssystem

Das Szenario im Installationszustand kommt mit einem rudimentär implementierten Belohnungssystem daher. Dieses schüttet Belohnungen lediglich dann aus, wenn eine Partie als beendet gilt. Die Belohnungsfunktion ermittelt im Anschluss welches Team gewonnen hat und schüttet entsprechend eine positive Belohnung an das Siegerteam und eine negative Belohnung an das Verliererteam aus. Dieses System wurde von Grund auf neu aufgebaut, sodass nun in jeder Spielrunde der aktuelle Zustand der Partie bewertet und dem Agenten als konsolidierte Belohnung zurückgegeben werden kann. Zu eben dieser Modifikation des Belohnungssystems war es notwendig, die im vorherigen Absatz beschriebenen Änderungen an der Pommerman-Umgebung durchzuführen.

Bezeichnung	Zeitpunkt	Belohnung	
		min	max
Reduktion der Entfernung zum Gegner	pro Spielrunde	0	0.01
Erhöhung der Distanz zur Bombe	pro Spielrunde	-0.01	0.01
Erhöhung der Distanz zum Feuer	pro Spielrunde	-0.01	0.01
Platzierung der Bombe beim Gegner	pro Spielrunde	0	0.5
Basis Rundenbelohnung	pro Spielrunde	-0.001	-0.001
Besiegen des Gegners	pro Spielrunde	0	1
Besiegen von Teamagenten	pro Spielrunde	-1	0
Suizid	pro Spielrunde	-2	0
Sieg	pro Partie	0	2
Niederlage	pro Partie	-2	0
Unentschieden	pro Partie	-0.1	-0.1

Tabelle 5: Pommerman Belohnungsstruktur

Tabelle 5 zeigt die neu implementierte Belohnungsstruktur. Diese wurde für alle Agenten in der dargestellten Form zum Training der Modelle verwendet. Vorher bestand die Rückmeldung für die Agenten lediglich aus den Werten eins, beziehungsweise minus eins, für Sieg

oder Niederlage. Zweck der Optimierung der Belohnungsstruktur war es den Agenten wesentlich kleinschrittiger Rückmeldungen zu tatsächlich stattfindendem Verhalten innerhalb einer jeden Runde geben zu können. Mit dem Ziel, dass diese effizienter lernen können, um mit den im nachfolgenden Absatz beschriebenen Herausforderungen und Unwegsamkeiten der Umgebung umgehen zu können. Die neue Belohnungsstruktur sieht vor, die Agenten für tendeziell positives Verhalten zu belohnen, um so eine Entwicklung hin zur optimalen Strategie zu fördern. Hierzu zählen kleinere Belohnungen für das Auswählen der Bewegungsaktion, um sich von Bomben oder Feuern zu entfernen oder sich direkt auf Kontrahenten zu zubewegen. Darüber hinaus erhält der Agent ebenfalls positive Rückmeldungen, wenn Bomben in der Nähe von Gegnern oder zerstörbaren Objekten, wie Wänden platziert worden sind.

Der Tabelle 5 sind zudem die Werte von Belohnungen zu entnehmen. Einige Belohnungen können sowohl positive als auch negative Rückmeldungen enthalten. Beispielsweise im Fall einer „Erhöhung der Distanz zur Bombe“, wenn der Agent sich einer Bombe nähert, statt sich zu entfernen, erhält er eine negative Belohnung, da dies unerwünschtes Verhalten darstellt. Alle Belohnungsfunktionen sind, wenn sinnvoll, mit Sicherheitsmechanismen ausgestattet. Im Falle der Erhöhung von Distanzen wird beispielsweise berücksichtigt wie lange die Bombe schon auf dem Spielfeld liegt. Das soll verhindern, dass ein Agent unmittelbar nach dem Platzieren einer eigenen Bombe negativ belohnt wird, obwohl er noch nicht die Möglichkeit hatte, sich von dieser zu entfernen. Darüber hinaus wurden Belohnungen hinzugefügt, die nach dem Eliminieren eines Agenten ausgeschüttet werden. Hier ist von besonderer Relevanz, dass bei Eliminierung eines Teammitglieds oder von sich selbst, ein verhältnismäßig hoher negativer Belohnungswert zurückgegeben wird. Das hat den Hintergrund, dass solches Verhalten starke negative Auswirkungen auf den restlichen Verlauf der Partie hat und daher entsprechend stark geahndet wird.

Ähnlich der ursprünglichen Implementation gibt es eine Belohnung für Sieg, Niederlage und Unentschieden. Hierbei wird bei einem Sieg die maximal mögliche positive Belohnungshöhe ausgeschüttet, wohingegen bei einer Niederlage und bei Unentschieden eine negative Belohnung ausgeschüttet wird. Die negative Belohnung bei Unentschieden hat den Zweck, dass Agenten keine Strategie entwickeln sollen, bei der sie den Gegner zwar eliminieren, sich selbst aber automatisch mit zerstören. Zusätzlich wird pro Runde eine kleine negative Belohnung ausgeschüttet. Das soll verhindern, dass Agenten möglichst lange abwarten und stattdessen eine aktive Spielweise gefördert wird. Jeder Belohnung sind zwei Werte zugeordnet, diese bestehen aus Belohnung min. und Belohnung max., welche in der Tabelle 5 dargestellt sind. Nach Abschluss einer Runde und nach der Überprüfung aller Belohnungen werden alle Werte für jeden einzelnen Agenten addiert und zurückgemeldet. Darüber hinaus wurden alle Belohnungshöhen, ob negativ oder positiv durch manuelles Tarieren optimiert, wenn gleich versucht wurde diese so zu gestalten, dass keine überhöhten Gesamtbelohnungswerte in den späteren Auswertungen zu finden sind, sondern, dass diese möglichst homogen der entsprechenden Wichtigkeit nach verteilt sind.

Herausforderungen der Pommerman-Umgebung Die Umgebung „Pommerman“ hält einige Herausforderungen bereit. Die im vorherigen Absatz beschriebene angepasste Belohnungsstruktur soll den Agenten eine Möglichkeit bieten optimale Strategien für die Um-

gebung entwickeln zu können. Eine der größten Herausforderungen neben der Navigation der Agenten innerhalb des 11x11 Spielfeldes stellt der Umstand dar, dass Agenten sich und Teammitgliedern Schaden zufügen können. Besonders durch die Tatsache, dass Bomben eine längere Bedrohung darstellen, nach der Explosion Feuer hinterlassen und dass Bombenexplosionen zu einer Kettenreaktion führen können, erschwert die Situation zusätzlich deutlich. Dem hinzu kommt, dass Agenten nicht nur eine eingeschränkte Sicht haben, zusätzlich wird mit jeder neuen Runde auch der Aufbau des gesamten Spielfeldes neu gesetzt.

3.4 Belohnungssystem Basis und Individual

Im Folgenden wird der Ablauf und Umfang des Vergleichs der beiden Belohnungssysteme Basis gegenüber Individual dargelegt. Das Ziel besteht darin, den Effekt der im Rahmen dieser Arbeit eingesetzten individuellen Belohnungsfunktion auf verschiedene Trainingskonstellationen zu evaluieren und zu begründen. Zudem wird erläutert, wie ermittelt wurde gegen welche Art von Teamkonstellation die zu trainierenden Agenten während des Trainings angetreten sind.

Die Evaluation erfolgte unter einem verkürzten Trainingsverfahren. Es wurde die von Pommerman bereitgestellte Belohnungsfunktion, die ausschließlich auf Sieg oder Niederlage basiert, mit einer eigens entwickelten Belohnungsfunktion verglichen. Diese legt den Fokus auf positives Verhalten und bietet eine feingranulare Rückmeldung. Details hierzu sind im Kapitel „Belohnungssystem“ beschrieben. Das Training umfasste 300 Iterationen für die Algorithmen QMIX und VDN, sowie 100 Iterationen für VDAC.

Trainingsgegner
Algorithmus trainierte gegen den zufallsbasierten Agenten
Algorithmus trainierte gegen den regelbasierten Agenten
Algorithmus trainierte gegen eine eigene Instanz

Tabelle 6: Übersicht der Trainingsgegner des Algorithmus

In der Tabelle 7 repräsentiert „Basis“ die standardmäßige, von Pommerman bereitgestellte Belohnungsfunktion, während „Individual“ die speziell entwickelte Belohnungsstruktur bezeichnet. Die Tabelle zeigt, dass insgesamt sechs Trainingsdurchläufe durchgeführt wurden, in denen beide Belohnungsfunktionen unter drei verschiedenen Bedingungen zum Training eingesetzt worden sind. Nach Abschluss dieser Trainingsphasen traten alle erstellten Trainingsmodelle in einem Turnier über eine Iteration nach dem in Tabelle 6 gezeigten Schema an. Insgesamt sind so zwischen den Systemen im Rahmen dieser Evaluation 18 Modelle erstellt worden. Zudem wurden die während des Trainings gesammelten Metriken zur abschließenden Einordnung herangezogen. Besonders beachtet wurden die Metriken des durchschnittlichen Belohnungswertes und der durchschnittlichen Rundenlänge.

Belohnungsfunktion	Trainingsmethode
Basis	Algorithmus trainiert vs. Zufallsbasierte Agenten Algorithmus trainiert vs. eine Version von sich selbst Algorithmus trainiert vs. Regelbasierte Agenten
Individual	Algorithmus trainiert vs. Zufallsbasierte Agenten Algorithmus trainiert vs. eine Version von sich selbst Algorithmus trainiert vs. Regelbasierte Agenten

Tabelle 7: Trainings Szenario für VDN, QMIX, VDAC

4 Training

Im folgenden Abschnitt dieser Arbeit wird zunächst der verwendete Trainingsaufbau vorgestellt, gefolgt von der Methodik zur Ermittlung optimaler Hyperparameter. Für jeden der drei in dieser Arbeit untersuchten Algorithmen ist ein separates Kapitel vorgesehen, in dem die Evaluation der durchgeführten Trainings dargestellt wird. In jedem dieser Kapitel werden Evaluationsmetriken präsentiert und verschiedene Trainingsvariationen, wie beispielsweise die Wahl des Kontrahenten und die spezifische Belohnungsfunktion, gegenübergestellt. Abschließend erfolgt eine zusammenfassende Bewertung aller untersuchten Algorithmen sowie eine explorative Analyse potenzieller Optimierungsansätze für ihren Einsatz.

4.1 Trainingsaufbau

Durchgeführt wurde jedes Training mittels MARLlib. Die Pommerman-Umgebung wurde entsprechend der in den vorherigen Abschnitten gezeigten Umgebungskonfiguration und Belohnungsstruktur für jeden Algorithmus identisch eingesetzt. Neben diesen Konfigurationen wurde das Training der drei für die Ausarbeitung eingesetzten Algorithmen unter identischen Rahmenbedingungen durchgeführt. Das umfasst die Länge des Trainings in Iterationen, die eingesetzte Hardware sowie die verwendete Art des Trainingsgegners. Jedes Training fand über ein 1000 Trainingsiterationen andauerndes Zeitintervall statt. In einer Trainingsiteration werden im Durchschnitt mehrere hundert Partien gespielt. Jede Partie besteht aus mehreren Runden, deren maximale Anzahl von der Konfiguration abhängt. Das Training aller Algorithmen fand stets gegen ein Team aus regelbasierten Agenten statt, diese Form wird durch Pommerman selbst bereitgestellt. Die tatsächliche Anzahl an trainierten Episoden pro Algorithmus kann variieren, weil die Anzahl der Trainingsiterationen festgelegt ist, jedoch die Anzahl der Partien und Runden pro Iteration variabel ist. Das hat neben den Eigenheiten der einzelnen Algorithmen auch damit zu tun, dass jeder Algorithmus eigenständig optimiert wurde und es in der Konsequenz daher zu einer unterschiedlichen Anzahl an Episoden pro Iteration kommen kann. Darüber hinaus wurde jeder Algorithmus von Grund auf neu trainiert, es fand keine Verwendung eines vor-trainierten Modells statt. Im Kontext des Reinforcement Learning bezeichnet eine Episode einen vollständig abgeschlossenen Durchlauf oder Interaktionszyklus zwischen dem Agenten und der Umgebung. In dieser Ausarbeitung wird der

Begriff „Episode“ synonym zu „Partie“ verwendet werden, da besonders wenn es sich um spielbasierte Szenarien handelt, beide Begriffe einen abgeschlossenen Prozess oder Zyklus beschreiben und daher im Kern das gleiche Aussagen.

4.1.1 Hyperparameter

Parameter, welche zur Optimierung des Trainings von Algorithmen verwendet wurden, nennt man Hyperparameter. Diese unterscheiden sich in Art und Funktion je nach eingesetztem Algorithmus. MARLlib bietet für ausgewählte Szenarien bereits voreingestellte und optimierte Hyperparameter an. Hiervon ausgenommen ist jedoch Pommerman, welches zum Zeitpunkt der Erstellung dieser Arbeit nicht mit einer solchen Optimierung ausgeliefert wurde. Das Finden der idealen Hyperparameter ist ein essenzieller Bestandteil des Trainings und zugleich notwendig, um ein ideales Trainingsergebnis zu erzielen [2].

Da es in der Regel aufgrund der Vielzahl an Variationen innerhalb der Hyperparameter nicht möglich ist alle manuell zu überprüfen, wurde im Rahmen der Arbeit, um die idealen Parameter zu finden, auf eine Kombination aus manuellem Testen und der Techniken der Zufallssuche und Grid-Suche zurückgegriffen. Hierzu wurde im ersten Schritt ein Minimum- und Maximum-Bereich durch manuelles Testen definiert, in welchem die Parameter agieren sollen. Zur abschließenden Optimierung dieser Parameter wurde jedem Algorithmus ein Zeitfenster von 24 Stunden zugewiesen, in dem mithilfe von Rlib und der Techniken der Zufallssuche bzw. Grid-Suche versucht wurde, ein Optimum zu finden. Diese Limitierung wurde vorgenommen, um die Vergleichbarkeit der einzelnen Algorithmen zu gewährleisten und sicherzustellen, dass alle Algorithmen unter den gleichen Voraussetzungen optimiert wurden.

Nach Abschluss des Zeitfensters wurde, basierend auf der durchschnittlichen Episodenbelohnung jene Konfiguration ausgewählt, die den höchsten Wert aufwies. Jede per Zufallssuche getestete Konfiguration hatte eine interne Begrenzung von 100 Trainingsiterationen. Ein Beispiel für einen über verschiedene Algorithmen hinweg vorkommenden Hyperparameter ist die Lernrate. Diese beschreibt die Rate, mit der das trainierende Modell die Gewichtungen im neuronalen Netz aktualisiert. Eine Herausforderung, insbesondere bei der Lernrate, besteht darin, diese weder zu hoch noch zu niedrig anzusetzen. Gängige Praxis bezüglich der Lernrate ist, dass sie einen positiven Wert im Bereich zwischen 0.0 und 1.0 annimmt. Ist die Lernrate zu niedrig eingestellt, kann es passieren, dass eine ideale Strategie nicht erlernt wird, da ein Update des neuronalen Netzes nicht bzw. zu langsam stattfindet. Ist der Wert hingegen zu hoch, kann es vorkommen, dass keine langfristig ideale Strategie gefunden wird und das Modell zu impulsiv lernt, wodurch es eventuell positive Ansätze schnell verwirft.

4.1.2 Methodik zur Auswertung

Während einer jeden Trainingseinheit wurden fortlaufend Kennzahlen gesammelt, welche im nachfolgenden dazu genutzt werden Tendenzen des Trainingserfolgs abzuleiten. Neben dieser reinen wertebasierten Analyse wurde nach Abschluss des Trainings jedes erzeugte Modell der drei Algorithmen über eine Iteration, gegen die in vorherigen Kapiteln vorgestellten Referenzagenten antreten gelassen. Abschließend wurden vereinzelte Partien manuell in die

Pommerman-Umgebung geladen und bildlich dargestellt, um eine eigenhändige Beobachtung durchzuführen. Diese drei Säulen konsolidiert führen zu einer Zwischenbewertung des Trainingserfolgs der Algorithmen. Außerdem sind alle durchgeführten Trainings einheitlich auf einem Intel i7 13700k CPU, sowie einer Nvidia 4080 RTX GPU durchgeführt worden.

4.2 Training Value-Decomposition Networks

Als erstes der drei betrachteten Algorithmen wurde der Ansatz VDN Value-Decomposid-Network implementiert und evaluiert. Der Trainingsprozess, welcher über 1000 Iterationen stattfand, verlief über einen Zeitraum von 1.500.000 Zeitintervallen beziehungsweise circa 30.000 Episoden. Die Hyperparamater zur idealen Anpassung des Trainingsprozesses wurden, wie in dem vorherigen Absatz erläutert, ermittelt. Das so gefundene Optimum der verwendeten Hyperparameter wird in Tabelle 8 wie folgt dargestellt:

Parameter	**Werte**
buffer_size	4500
epsilon_timesteps	45000
final_epsilon	0.01
lr	0.0001
optimizer	adam
reward_standardize	False
rollout_fragment_length	1
target_network_update_freq	200
batch_episode	8

Tabelle 8: VDN optimale Hyperparameter

4.2.1 Trainingsevaluation und Auswertung

Im nachfolgenden Absatz werden die zur Trainingseinschätzung relevantesten Kennzahlen dargestellt und analysiert. Alle gesammelten Trainingskennzahlen wurden während einer durchgängigen Trainingssession gesammelt. Die nachfolgend gezeigten Grafiken wurden mit der Python Bibliothek Plotly erstellt.

In Abbildung 13 dargestellt ist die durchschnittliche Belohnung, welche pro Trainingsiteration an den Agenten ausgeschüttet wurde. Die x-Achse zeigt den Wert der Belohnung an, die y-Achse zeigt die Iteration an. Zusammen wird anhand der Abbildung verdeutlicht wie sich die Belohnung im Verlauf der Trainings verhielt. Insgesamt zeigt das Diagramm einen Aufwärtstrend über den Gesamttrainingszyklus hinweg.

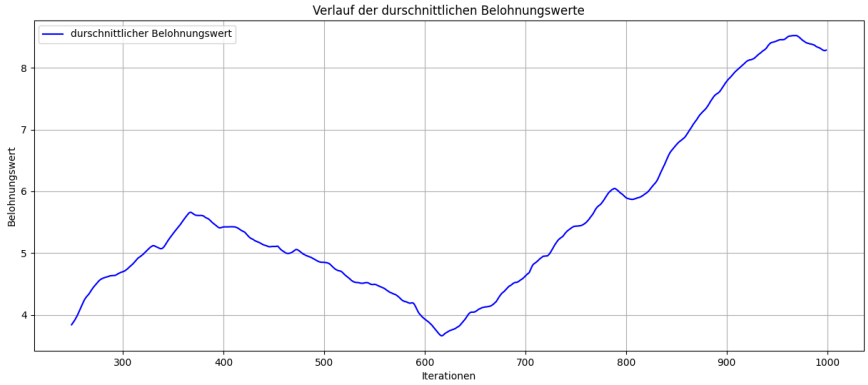


Abbildung 13: VDN: durchschnittliche Belohnung während Training

Abbildung 14 zeigt die durchschnittliche Länge einer Partie. Die x-Achse zeigt die durchschnittliche Länge einer Partie in Zeitintervallen und die y-Achse zeigt die Iteration an. Der Graf zeigt hier einen deutlichen Abwärtstrend, gerade zu Beginn des Trainings sinkt die durchschnittliche Länge einer Partie rapide. Dieser Abwärtstrend kehrt allerdings ab circa der Hälfte des Trainings um in einen Aufwärtstrend. Das Diagramm verdeutlicht erstmals eine gewisse Fluktuation, mit einer Umkehr des Trends ab der Hälfte des Trainingsvorhabens.

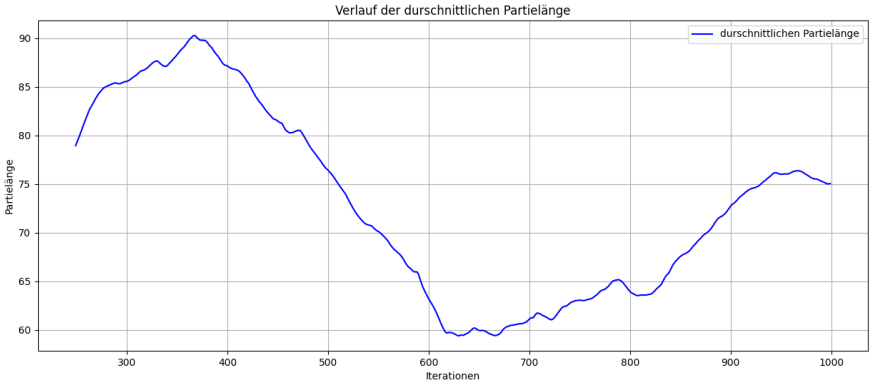


Abbildung 14: VDN: durchschnittliche Länge einer Partie

Die Abbildung 15 wiederum zeigt den Verlauf der Verlustfunktion während der Trainingseinheit. Die x-Achse zeigt einen Referenzwert und die y-Achse zeigt die Iteration an. In der Abbildung zeigt sich deutlich ein starker Abwärtstrend, der kontinuierlich in eine Seitwärtsbewegung übergeht. Eine Ausnahme bildet eine am Ende sichtbare, aber kurzlebige Anomalie. Wie der Abbildung zu entnehmen ist, normalisiert sich diese Situation schnell.

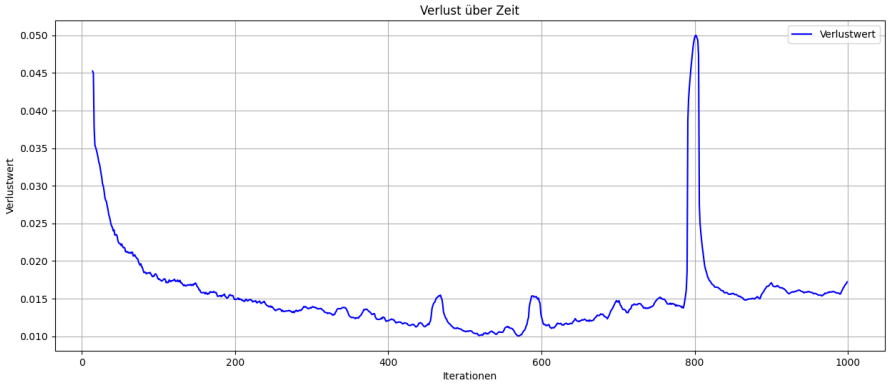


Abbildung 15: VDN: Verlauf der Verlustfunktion

Der letzte auf den gesammelten Kennzahlen basierende Graf zu sehen in Abbildung 16, zeigt den Q-Wert und Ziel-Wert Verlauf während des Trainings an. Die x-Achse zeigt in beiden Fällen einen Referenzwert und die y-Achse zeigt die Iteration an. Beide Diagramme zeigen einen sehr ähnlichen, starken Aufwärtstrend, kontinuierlich über den gesamten Trainingsverlauf hinweg.

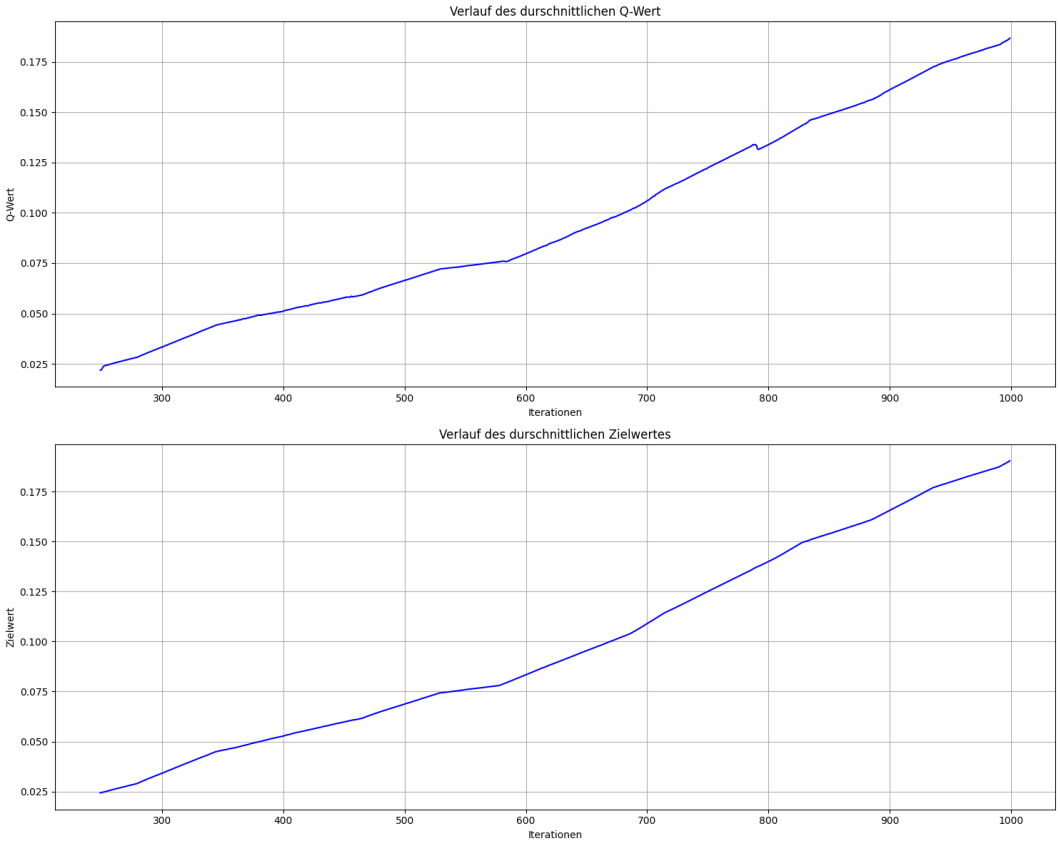


Abbildung 16: VDN: Q-Wert und Ziel-Wert-Verlauf

4.2.2 Vergleich gegen regel- und zufallsbasierte Agententeams

Im nachfolgenden Abschnitt wird das nach Abschluss des Trainings erstellte Modell gegen die Referenzagenten-Teams von Pommerman antreten. Zu diesem Zweck wurde das entsprechende Modell für die Dauer einer Iteration erneut in die Pommerman-Umgebung geladen. Basierend auf den in diesem Intervall gesammelten Daten wurde eine prozentuale Aufschlüsselung von Siegen, Niederlagen und Unentschieden erstellt.

Name	**Sieg (%)**	**Niederlage (%)**	**Unent. (%)**	**Gesamt**
AI vs. regelbasierte	54 (9,91 %)	487 (89,36 %)	4 (0,73 %)	545 (100 %)
AI vs. zufallsbasierte	423 (60,5 2%)	267 (38,20 %)	9 (1,29 %)	699 (100 %)

Tabelle 9: Vergleich der Leistung VDN gegen regelbasiert und zufallsbasierte Agenten

Die Tabelle 9 repräsentiert die Ergebnisse des Aufeinandertreffens beider Teams. Hierbei zeichnet sich ein deutlicher Trend ab. Das VDN-Agententeam verzeichnet eine durchschnittliche Niederlagenrate von 89,36 % im Vergleich zu einer 9,91%igen Siegesrate gegen das

regelbasierte Agententeam. Demgegenüber erzielte es eine 60,52%ige Siegesrate und eine 38,20%ige Verlustrate gegen das zufallsbasierte Agententeam. In beiden Fällen endeten jeweils circa 1 % der Partien mit dem Ergebnis Unentschieden. Die Daten wurden abschließend mittels des Chi-Quadrat-Tests, in der Variante Verteilungstest, analysiert, um die Ergebnisse auf statistische Signifikanz zu prüfen. Dabei wurde die Annahme getroffen, dass alle drei in Tabelle 9 genannten Varianten (Sieg, Niederlage, Unentschieden) bei zufälligem Auftreten jeweils zu einem Drittel vorkommen würden und einem angenommenen Signifikanzniveau von 0,05. Das Ergebnis zeigt, dass die genannten Varianten in Bezug auf die Gesamtanzahl der durchgeführten Partien als statistisch signifikant eingestuft werden können.

4.2.3 Vergleich Basis- und Individualbelohnungsfunktion

Im folgenden Kapitel wird gemäß dem im Kapitel „Klassifizierung der Umgebung“ beschriebenen Schema ein Vergleich zwischen den Belohnungssystemen „Individual“ und „Basis“ unter Verwendung des Algorithmus VDN durchgeführt. Dies dient zur Legitimation der im vorherigen Kapitel angewandten Trainings- und Belohnungsstrategien. Das testweise durchgeführte Training war auf 300 Iterationen je Variante beschränkt. Diese Anzahl wurde festgelegt, um eine zügige Auswertung der Ergebnisse sicherzustellen, ohne viel Rechenzeit und Rechenleistung zu beanspruchen. Sie stellt eine robuste Basis dar, um die relevanten Schlüsse und den Vergleich der beiden Systeme zu ziehen.

In Tabelle 10 sind die Ergebnisse der Partien dargestellt, die nach Abschluss der jeweiligen Trainingssitzungen gegen die entsprechenden Gegnerteams durchgeführt wurden. Es zeigt sich, dass das individuelle Belohnungssystem über alle drei Kategorien (AI vs. AI, AI vs. Zufall, AI vs. Regel) bessere Ergebnisse in Form der prozentual erreichten Siege erzielte. Diese Daten resultieren aus einem Turnier, abgehalten über die Dauer einer Iteration, in dem das trainierte Agententeam gegen das jeweilige Pendant antrat. Zu erkennen ist eine Verbesserung durch den Einsatz des Individual Belohnungssystems, welches bei Betrachtung AI vs. Zufall und AI vs. Regel wichtig ist.

Type	**Name**	**AI (%)**	**Gegner (%)**	**Unent. (%)**	**Gesamt**
Basis	AI vs. AI	169 (49.27 %)	168 (48.98 %)	6 (1.75 %)	343
Basis	AI vs. Zufall	243 (61.21 %)	146 (36.78 %)	8 (2.02 %)	397
Basis	AI vs. Regel	43 (15.52 %)	234 (84.48 %)	0 (0.00 %)	277
Individual	AI vs. AI	165 (54.10 %)	134 (43.93 %)	6 (1.97 %)	305
Individual	AI vs. Zufall	264 (64.23 %)	137 (33.33 %)	10 (2.43 %)	411
Individual	AI vs. Regel	40 (16.53 %)	202 (83.47 %)	0 (0.00 %)	242

Tabelle 10: Ergebnispunktzahl Basis gegen individuelles Belohnungssystem VDN

In der Abbildung 17 werden Metriken dargestellt, die während des Trainings der jeweiligen Varianten erfasst wurden. Dabei konzentrierte sich die Darstellung auf zwei wesentliche Kennzahlen, die durchschnittliche Rundenbelohnung und die durchschnittliche Rundenlänge. Beide sind aussagekräftige Indikatoren für einen Vergleich der Varianten. Die Diagram-

me fassen die gesammelten Metriken aller sechs getesteten Varianten zusammen. Es bestätigt sich das bereits durch Tabelle 10 ermittelte Bild, dass das individuelle Belohnungssystem in allen Diagrammen eher zu einer Verbesserung beigetragen hat als der alleinige Einsatz des Basis-Belohnungssystems. Dies ist unter anderem daran zu erkennen, dass in den dargestellten Diagrammen die als „Individual“ gekennzeichneten Metriken in allen Belangen über den Basis-Metriken liegen.

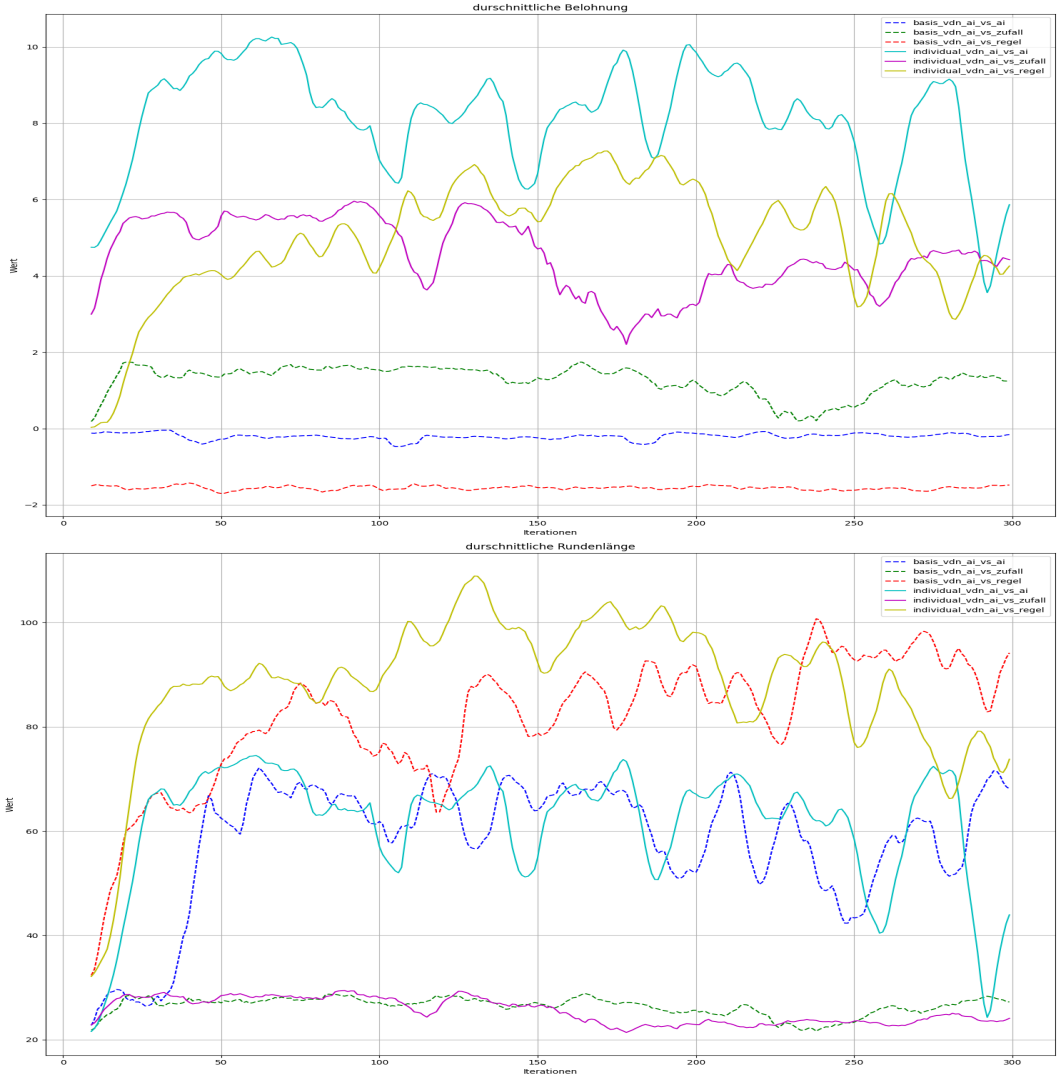


Abbildung 17: Ergebnisgrafen Trainings Basis gegen individueller Belohnungssystem VDN

4.2.4 Ergebnisinterpretation

Der nachfolgende Absatz befasst sich, basierend auf den drei beschriebenen Säulen, mit der Zusammenfassung und Interpretation der Trainingsdaten. Entsprechend der im vorherigen Trainingsabschnitt dargestellten Ergebnisse und unter Einsatz des VDN-Algorithmus gesammelten Kennzahlen, zeigt sich eine positive Trainingstendenz. Gerade bei der Betrachtung der durchschnittlichen Belohnung pro Partie ist ein deutlich positiver Trend erkennbar. Die Analyse der eingesetzten Belohnungsfunktion legt nahe, dass aufgrund des durchgehenden positiven Belohnungswertes der Agent in der Lage war, eine Strategie zu entwickeln, die in der Regel negative Aspekte umgeht.

Bezogen auf die Kennzahl der durchschnittlichen Belohnung zeigt sich, dass eine Steigerung positiv zu bewerten ist. Dies lässt sich damit begründen, dass sich die Agenten vermehrt entsprechend der eingesetzten Belohnungsfunktion positiven Verhaltensweisen zugewendet haben. Die positive Trainingstendenz wird durch den Verlauf der durchschnittlichen Länge einer Partie bestätigt. Obwohl die Belohnungsstruktur so gestaltet ist, dass in jeder Runde negative Belohnungen ausgeschüttet werden, um überlange Partien zu verhindern, ist dennoch ein gewisses Mindestmaß an Rundenlänge erforderlich, um zum Beispiel genügend Wände zu entfernen und einen Gegner erfolgreich ausschalten zu können. Daher ist der gezeigte Trend, welcher zum Ende hin wieder steigend ist, hier durchaus als positiv zu bewerten. Die Verlustfunktion wiederum zeigt zu Beginn ein als positiv zu bewertendes Verhalten, da sie gegen null tendiert. Mit Ausnahme einer gegen Ende kurzweilig stattfindenden Anomalie, es wird angenommen, dass in diesem Zeitraum unerwartete Ereignisse eingetreten sind. Trotz Analyse der Logdateien konnte jedoch keine konkrete Ursache ermittelt werden. Aufgrund der Kurzlebigkeit der Anomalie und der darauffolgenden Selbstregulierung wurde in diesem Fall nicht weiter interveniert.

Diese Beobachtung wird durch das Verhalten von Q-Wert und Ziel-Wert bestätigt, die bis zum Ende des Trainings eine kontinuierlich steigende Tendenz zeigen. Das wiederum ist ein Indikator dafür, dass die Agenten Schwierigkeiten hatten, zukünftige Aktionen optimal vorherzusagen. Die reine Kennzahlenbetrachtung liefert somit kein eindeutiges Gesamtbild, obwohl die kontinuierliche Steigerung der durchschnittlichen Belohnung als deutliches positives Signal interpretiert wird. Wenn darüber hinaus die in Tabelle 9 gezeigten Ergebnisse in die Bewertung des Trainingsvorhabens einbezogen werden, ergibt sich ein differenziertes Lagebild. Es zeigt sich, dass die durch VDN trainierten Agenten im Vergleich zu den regelbasierten Agenten nicht konkurrenzfähig sind. Im direkten Vergleich mit zufallsbasierten Agenten ergibt sich zwar eine positive Tendenz, insofern als dass häufiger gewonnen als verloren wurde, doch repräsentieren zufallsbasierte Agenten nicht denselben Schwierigkeitsgrad wie regelbasierte Agenten. Eine manuell durchgeführte Analyse einzelner Partien gegen das regelbasierte Agententeam bestätigt dieses Ergebnis.

Trotz Anpassungen der Belohnungsstrukturen liegt das primäre Defizit der VDN-Agenten darin, sich mit fortschreitendem Rundenablauf oftmals selbst zu eliminieren. Dennoch gibt es positive Tendenzen, die Agenten haben gelernt, eine Strategie zu entwickeln welche zumindest in gewissen Maße Bestand hat gegen die zufallsbasierten Agenten. Dies ist insbesondere darauf zurückzuführen, dass die VDN-Agenten länger am Leben bleiben als die zufallsbasierten Agenten, welche, aufgrund ihrer internen Logik, dazu neigen, sich durch eigene Bomben

oder Feuer zu eliminieren. Als Zwischenfazit lässt sich festhalten, dass sich im Vergleich zu zufallsbasierten Agententeams positive Entwicklungen abzeichnen, auch wenn die erlernte Strategie nicht ausreicht, um in der Mehrheit der Partien gegen ein regelbasiertes Agententeam zu gewinnen. Abschließend hat der Vergleich von Basis- und individuellem Belohnungssystem gezeigt, dass das individuelle System positive Resonanzen hervorgerufen und sowohl das Training als auch die allgemeine Leistung positiv beeinflusst hat. Zudem scheint die Wahl des Trainings gegen regelbasierte Agententeams in der getesteten Kombination die geeignetste zu sein.

4.3 Training Mixed Q-Learning

Als zweiter im Rahmen dieser Arbeit betrachteter Algorithmus wurden die Agenten mittels QMIX trainiert. Das Trainingsintervall belief sich wie bei VDN auf 1000 Trainingsiterationen. Die optimalen Hyperparameter wurden wie eingangs erwähnt ermittelt und können der Tabelle 11 entnommen werden.

Parameter	**Werte**
buffer_size	5000
epsilon_timesteps	50000
final_epsilon	0.05
lr	0.0005
optimizer	rmsprop
reward_standardize	True
rollout_fragment_length	1
target_network_update_freq	200
batch_episode	16

Tabelle 11: QMIX optimale Hyperparameter

4.3.1 Trainingsevaluation und Auswertung

Im folgenden Abschnitt werden die für die Trainingseinschätzung entscheidenden Kennzahlen vorgestellt und untersucht. Die betrachteten Trainingsdaten wurden während einer kontinuierlichen Trainingssession gesammelt. Die nachstehend gezeigten Grafiken wurden mittels der Python Bibliothek Plotly erzeugt.

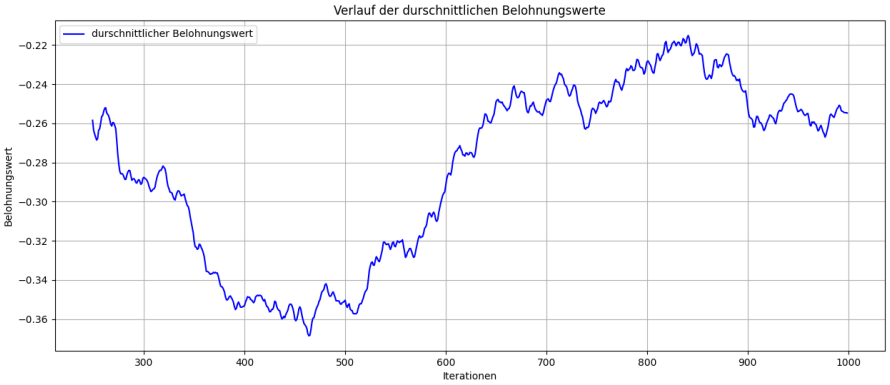


Abbildung 18: QMIX: durchschnittliche Belohnung während Training

Die Abbildung 18 zeigt, ähnlich der Darstellung aus dem VDN Absatz, die absolut erhaltene Belohnung über das Zeitintervall des gesamten Trainingszyklus. Die y-Achse zeigt hierbei die durchschnittliche Höhe der Belohnung und die x-Achse zeigt die Iteration an. Gerade zu Beginn des Trainings zeigt sich ein klarer Abwärtstrend, erst ab circa 500 Trainingsiterationen steigt die durchschnittlich erhaltene Belohnung wieder an. Zum Ende hin tendiert die Belohnung zu einer fluktuierenden Seitwärtsbewegung. So zeigt sich über die Gesamtdarstellung hinweg zwar ein positiver Trend, jedoch ist zu erkennen, dass der Höchstwert im Trainingsverlauft nie positiv ist. Darüber hinaus bewegt sich der Belohnungswert innerhalb eines vergleichsweise eingeschränkten Wertebereichs.

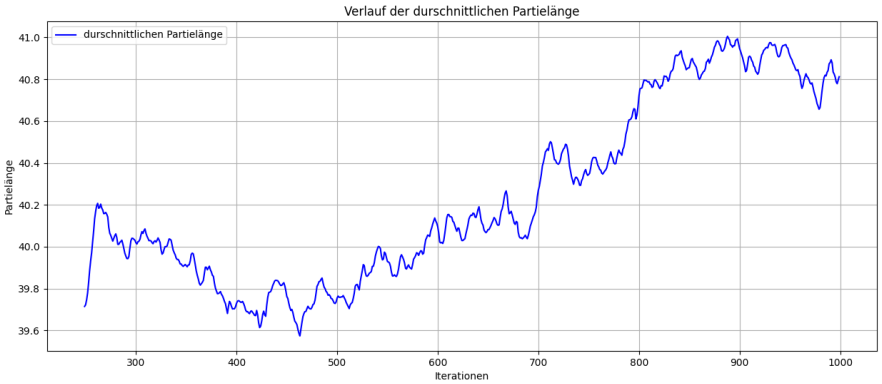


Abbildung 19: QMIX: durchschnittliche Länge einer Partie

Die Abbildung 19 zeigt insgesamt den Verlauf der Länge einer Partie über den Trainingsverlauf hinweg. Wobei die x-Achse die durchschnittliche Länge der Partie und die y-Achse die Iteration angibt. Hier zeigt sich ein ähnliches Bild wie in Abbildung 18. Bis circa zur Hälfte des Trainings zeigt sich ein leichter Abwärtstrend, die Partien enden also grundsätz-

lich schneller als zu Beginn. Ab circa 500 Iterationen steigt dann die durchschnittliche Länge einer Partie deutlich an. Hier zeigen sich ebenfalls Ähnlichkeiten bei der Betrachtung des absoluten Geltungsbereichs. Die durchschnittliche Rundenlänge beläuft sich auf einem Bereich von 39,7 bis 40,8 Sekunden.

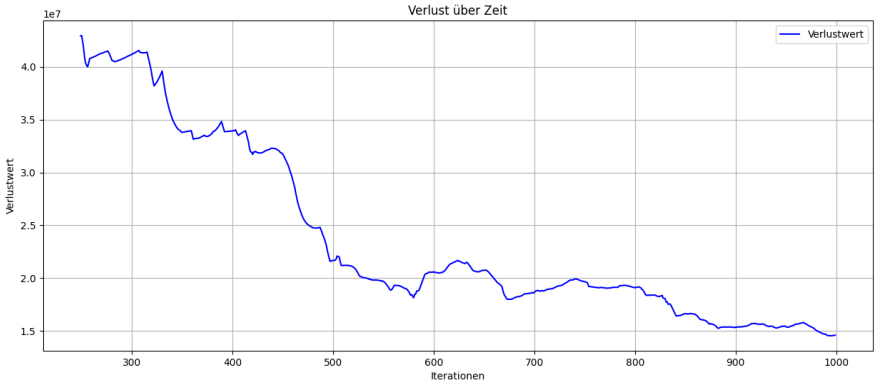


Abbildung 20: QMIX: Repräsentation der Verlustfunktion über Zeit

In Abbildung 20 wird der Verlustwert aus dem gesamten Trainingsprozess dargestellt. Die x-Achse zeigt einen Referenzwert für Verlust und die y-Achse zeigt die Iteration an. Zu erkennen ist ein klarer Abwärtstrend, welcher mit fortlaufendem Training zwar kurzzeitig in eine Seitwärtsbewegung übergeht, schlussendlich aber zum Abschluss des Trainings wieder in einer leichten Abwärtstendenz endet.

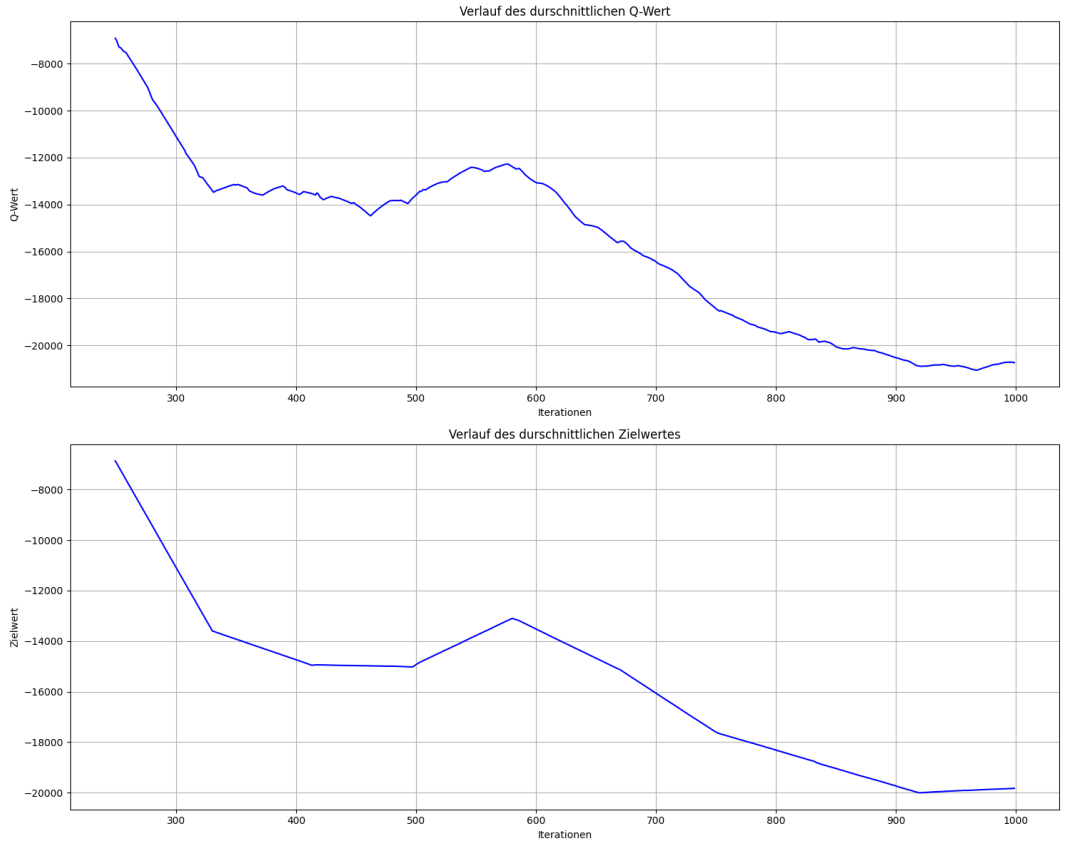


Abbildung 21: QMIX: Q-Wert und Ziel-Wert-Verlauf

Die letzten aus den Trainingsdaten abgeleiteten Diagramme, zu sehen in Abbildung 21 zeigen den durchschnittlichen Q-Wert und den durchschnittlichen Ziel-Wert des Trainingsverlaufs. Die x-Achse zeigt in beiden Fällen einen Referenzwert und die y-Achse zeigt die Iteration an. Bei beiden zeigt sich ein ähnlicher Verlauf, welcher absteigende Tendenzen aufweist. Zum Ende der Trainingsiterationen flacht der Verlauf deutlich ab und geht in eine Seitwärtsbewegung über, welche bis zum Abschluss des Trainings beibehalten wird.

4.3.2 Vergleich gegen regel- und zufallsbasierte Agententeams

Im nachfolgenden Abschnitt wird das nach Abschluss des QMIX-Trainings erstellte Modell gegen die Referenzagenten-Teams von Pommerman antreten gelassen. Zu diesem Zweck wurde das entsprechende Modell für die Dauer einer Iteration erneut in die Pommerman-Umgebung geladen. Basierend auf den in diesem Intervall gesammelten Daten wurde eine prozentuale Aufschlüsselung von Siegen, Niederlagen und Unentschieden erstellt.

Name	AI (%)	Gegner (%)	Unent. (%)	Gesamt
AI vs. regelbasierte	19 (27,94 %)	49 (72,06 %)	0 (0,00 %)	68 (100%)
AI vs. zufallsbasierte	129 (86,58 %)	17 (11,41 %)	3 (2,01 %)	149 (100%)

Tabelle 12: Vergleich der Leistung QMIX gegen regelbasiert und zufallsbasierte Agenten

In Tabelle 12 zu sehen sind die so gewonnenen Ergebnisse der Partie zwischen den QMIX-Agenten und den Pommerman-Referenzagenten. Das QMIX-Agententeam hat eine durchschnittliche 72,06%ige Niederlagenrate gegenüber einer 27,94%igen Siegesrate gegen das eingesetzte regelbasierte Agententeam erzielen können. Demgegenüber steht eine 86,58%ige Siegesrate beziehungsweise 11,41%ige Verlustrate gegen das zufallsbasierte Agententeam. Lediglich während des Aufeinandertreffens mit dem zufallsbasierten Agententeam ist es in 2,01 % der Fälle zu einem Unentschieden gekommen. Die Daten wurden schließlich unter Anwendung des Chi-Quadrat-Tests in einer Verteilungstest-Variante überprüft, um ihre statistische Signifikanz zu bestätigen. Ausgehend von der Annahme, dass alle drei in Tabelle 12 aufgeführten Ergebnisse (Sieg, Niederlage, Unentschieden) bei zufälliger Verteilung jeweils zu einem Drittel auftreten. Zudem wurde ein Signifikanzniveau von 0,05 festgelegt. Die Analyse ergab, dass die erwähnten Ergebnisse im Kontext der gesamten gespielten Partien als statistisch signifikant betrachtet werden können.

4.3.3 Vergleich Basis- und Individualbelohnungsfunktion

Im folgenden Kapitel wird gemäß dem im Kapitel „Klassifizierung der Umgebung“ beschriebenen Schema ein Vergleich zwischen den Belohnungssystemen „Individual“ und „Basis“ unter Verwendung des Algorithmus QMIX durchgeführt. Dies dient zur Legitimation der im vorherigen Kapitel angewandten Trainings- und Belohnungsstrategien. Das testweise durchgeführte Training war auf 300 Iterationen je Variante beschränkt. Diese Anzahl wurde gewählt, um eine effiziente Analyse der Ergebnisse zu ermöglichen, ohne überproportional viel Rechenzeit und Rechenleistung in Anspruch zu nehmen. Sie bietet eine solide Grundlage, um die nötigen Rückschlüsse und Vergleiche der beiden Systeme darzustellen.

In Tabelle 13 sind die Ergebnisse der Partien dargestellt, die nach Abschluss der jeweiligen Trainingssitzungen gegen die entsprechenden Gegnerteams ermittelt wurden. Es zeigt sich, dass das individuelle Belohnungssystem über alle drei Kategorien (AI vs. AI, AI vs. Zufall, AI vs. Regel) bessere Ergebnisse in Form der prozentual erreichten Siege erzielte. Diese Daten resultieren aus einem Turnier, abgehalten über die Dauer einer Iteration, in dem das trainierte Agententeam gegen das jeweilige Pendant antrat.

Type	Name	AI (%)	Gegner (%)	Unent. (%)	Gesamt
Basis	AI vs. AI	26 (50.00 %)	25 (48.08 %)	1 (1.92 %)	52
Basis	AI vs. Zufall	58 (40.28 %)	80 (55.56 %)	6 (4.17 %)	144
Basis	AI vs. Regel	13 (16.88 %)	64 (83.12 %)	0 (0.00 %)	77
Individuell	AI vs. AI	28 (49.12 %)	29 (50.88 %)	0 (0.00 %)	57
Individuell	AI vs. Zufall	132 (86.84 %)	19 (12.50 %)	1 (0.66 %)	152
Individuell	AI vs. Regel	15 (17.05 %)	73 (82.95 %)	0 (0.00 %)	88

Tabelle 13: Ergebnispunktzahl Basis gegen individuelles Belohnungssystem QMIX

In Tabelle 13 werden die Ergebnisse eines Turniers nach dem Training repräsentiert, die ein deutliches Bild des Vorteils des individuellen Belohnungssystems zeigen. Gerade im Vergleich der AI mit zufallsbasierten Agententeams erweist sich das individuelle Belohnungssystem dem Basissystem gegenüber als überlegen. Zudem zeigt sich eine tendenziell etwas bessere Performance gegenüber regelbasierten Gegnerteams, auch wenn der Unterschied geringfügig ausfällt. Basierend auf dieser Analyse scheint die „Individual“-Belohnungsfunktion in direkten Partien gegenüber der „Basis“-Belohnungsfunktion leicht überlegen zu sein. Ein signifikanter Vorteil der „Individual“-Belohnungsfunktion wird besonders in Spielen gegen zufallsbasierte Agententeams deutlich, wo sie der „Basis“-Belohnungsfunktion klar überlegen erscheint.

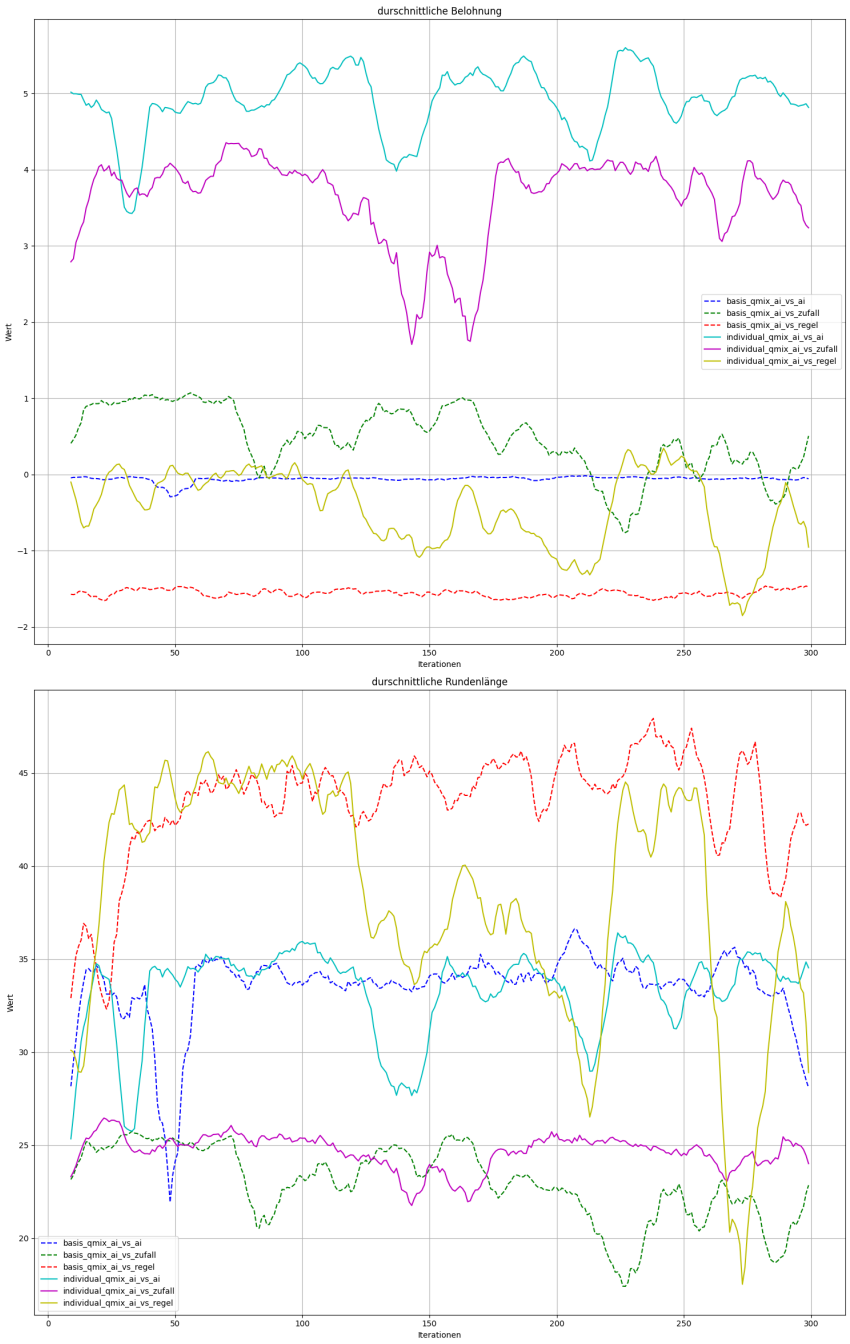


Abbildung 22: Ergebnisgrafen Trainings Basis gegen individuelles Belohnungssystem QMIX

Darüber hinaus wurde, wie in Abbildung 22 dargestellt ermittelt, mithilfe welchem Gegnerteam das Modell während des Trainings am effektivsten lernen kann. Die Abbildung prä-

sentiert für jede geprüfte Variante den Verlauf der durchschnittlichen Belohnung sowie der durchschnittlichen Rundenlänge. Nach Analyse der Darstellung zeigt sich, dass es bei der Betrachtung der durchschnittlichen Belohnung keine eindeutig bevorzugte Variante gibt. Keine Variante zeigt einen deutlich positiven Trend und alle verlaufen seitwärts. Ähnlich verhält es sich mit den durchschnittlichen Rundenlängen, der Verlauf tendiert bei allen Varianten seitwärts.

4.3.4 Ergebnisinterpretation

Im nun folgenden Abschnitt wird die Zusammenfassung der datengetriebenen Ergebnisse sowie die Erstellung eines Zwischenfazits basierend auf den drei Säulen behandelt. Das Training, das auf den gesammelten Informationen basiert, zeigt bei der reinen Betrachtung der Trainingskennzahlen einen tendenziell eher negativen Verlauf. Hingegen positiv zu bewerten ist der Verlauf des Verlustwertes. Dieser zeigt einen klaren Abwärtstrend, welcher im späteren Verlauf in eine kontinuierliche Seitwärtsbewegung übergeht. Dies ist durchaus als positiv zu bewerten, da daraus abgeleitet werden kann, dass es den Agenten möglich war, den Verlust im Vergleich zum Trainingsbeginn zu stabilisieren. Dies deutet darauf hin, dass zumindest eine zum Teil optimale Strategie im Szenario entwickelt werden konnte. Allerdings wird diese positive Beobachtung durch die Betrachtung des Q-Wertes und des Ziel-Wertes nicht bestätigt.

Trotz der zeitweiligen Plateaubildungen, welche als Stabilisierungen angesehen und als durchaus positives Zeichen interpretiert werden können, zeigt der generelle Verlauf beider Kennzahlen einen Abwärtstrend. Dies lässt darauf schließen, dass das Training nicht dazu geführt hat, dass die Agenten eine dauerhaft effektive Strategie entwickeln konnten. Bei der Analyse der durchschnittlichen Länge einer Partie sowie der durchschnittlich erhaltenen Belohnung zeigt sich zunächst ein positiver Aufwärtstrend. Bei genauerer Betrachtung wird jedoch ersichtlich, dass dieser Trend in einem für beide Verhältnisse sehr geringen Spektrum stattfand. Aus einer Makroperspektive betrachtet, verläuft der Trend seitwärts, sodass die Agenten weder eine signifikante Verbesserung in der Belohnung noch in den Rundenzeiten erzielen konnten. In Anbetracht dieser Kennzahlen kann das Training insgesamt nicht als positiv bewertet werden. Besonders die über das gesamte Trainingsintervall hinweg negativ verlaufende durchschnittliche Belohnungshöhe ist der ausschlaggebende Indikator für diese Bewertung.

Im Gegensatz dazu steht der Vergleich mit den Pommerman-Referenzagenten. Der Vergleich zeigt, dass das QMIX-Agententeam in der Lage war, eine Strategie zu entwickeln, mit der das zufallsbasierte Agententeam in der Mehrzahl der Partien besiegt werden konnte. Jedoch zeigt sich ein ähnliches Bild wie bei den trainierten VDN-Agententeams, wenn die Ergebnisse gegen das regelbasierte Team in Betracht gezogen werden. In diesem Fall konnte das QMIX-Team nicht in der Mehrzahl der Partien gegen das regelbasierte Team gewinnen. Eine genauere manuelle Untersuchung des Verhaltens der Agenten während einer Partie zeigt das Problem. Die Agenten konnten keine effektive Strategie entwickeln, die verhinderte, dass sie sich regelmäßig durch eigene Bomben selbst zerstörten. Lediglich im direkten Wettkampf gegen zufallsbasierte Agenten konnten die QMIX-Agenten ihre Stärke ausspielen, da sie im Vergleich zu diesen deutlich länger überleben konnten. Im Speziellen bestand eine Strategie

der QMIX-Agenten darin, sich möglichst wenig zu bewegen und abzuwarten bis gegnerische Agenten einen Weg freigeräumt haben. Auch das QMIX-Agententeam konnte die regelbasierten Agenten nicht zuverlässig besiegen. Die Probleme resultierten hier ebenfalls aus dem häufig selbstverschuldeten Ableben der Agenten durch fehlerhaft platzierte Bomben oder falsche Entscheidungen hinsichtlich der Bewegung. Dennoch ist das Ergebnis im Vergleich zu den zufallsbasierten Agenten als positiv zu bewerten. Insbesondere prozentual betrachtet konnte QMIX in vielen Partien hier als Gewinner hervorgehen.

Darüber hinaus wurde erläutert, weshalb speziell die Kombination aus Individual Belohnungssystem und Training gegen regelbasierte Gegnerteams gewählt wurde. Durch den Vergleich von Basis- und individuellem Belohnungssystem wurde verdeutlicht, dass das individuelle System positive Tendenzen aufzeigt. Dies wird bei der Betrachtung der Ergebnisse gegen die zufallsbasierten Agententeams sichtbar, bei denen das individuelle Belohnungssystem deutlich überlegen ist. Dennoch bleibt es über alle geprüften Varianten und Systeme hinweg eine Herausforderung, das regelbasierte Agententeam zuverlässig zu übertreffen.

4.4 Training Value-Decomposition Networks for Actor-Critic

Der letzte der betrachteten Algorithmen lautet Value-Decomposition Network for Actor-Critic abgekürzt mit VDAC. Hierbei gilt zu beachten, dass dieses Verfahren innerhalb von MARLlib leicht abweichend bezeichnet wird. MARLlib führt diesen Algorithmus unter dem Namen VDA2C, trotz der unterschiedlichen Namen ist das gleiche Verfahren damit gemeint. Das Trainingsintervall ist identisch zu QMIX und VDN auf 1000 Trainingsiterationen begrenzt. Die optimalen Hyperparameter wurden wie eingangs erwähnt ermittelt und können der Tabelle 14 entnommen werden.

Parameter	**Werte**
use_gae	True
lambda	1.0
vf_loss_coeff	1.5
batch_episode	16
batch_mode	truncate_episodes
lr	0.0005
entropy_coeff	0.02
mixer	qmix

Tabelle 14: VDAC Hyperparameter

4.4.1 Trainingsevaluation und Auswertung

Im nachfolgenden Absatz werden die zur Trainingseinschätzung relevantesten Kennzahlen dargestellt und analysiert. Alle gesammelten Trainingskennzahlen wurden während einer durchgängig stattfindenden Trainingssitzung ermittelt. Die nun folgenden gezeigten Grafiken wurden mit der Python Bibliothek Plotly erstellt.

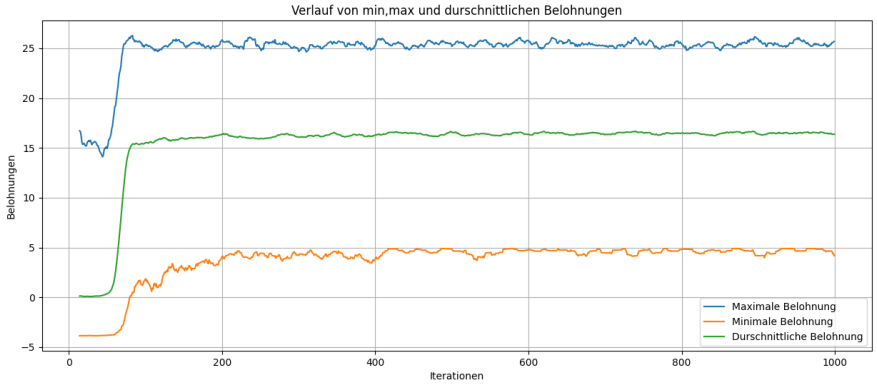


Abbildung 23: VDAC: maximale, minimal, durchschnittliche Belohnungshöhe

Abbildung 23 repräsentiert die während der Trainingsdurchführung ermittelten Belohnungswerte. Auf der y-Achse ist die Höhe der Belohnung dargestellt, während die x-Achse die Anzahl der Iterationen darstellt. Im Unterschied zu den vorherigen Algorithmen wird hier zwischen minimaler, maximaler und durchschnittlicher Belohnungshöhe differenziert. Das Diagramm unterteilt sich in drei Kernbestandteile. Der obere Verlauf stellt die maximale Belohnung dar, während der mittlere Verlauf die durchschnittliche Belohnungshöhe wiedergibt. Der untere Verlauf zeigt die minimal erhaltene Belohnung an. Alle drei Verläufe weisen eine im ersten Zehntel des Diagramms sichtbare steigende Tendenz auf. Diese geht über in eine kontinuierliche Seitwärtsbewegung, welche abgesehen von kleineren Fluktuationen bis zum Ende der gesamten Trainingseinheit konstant bleibt.

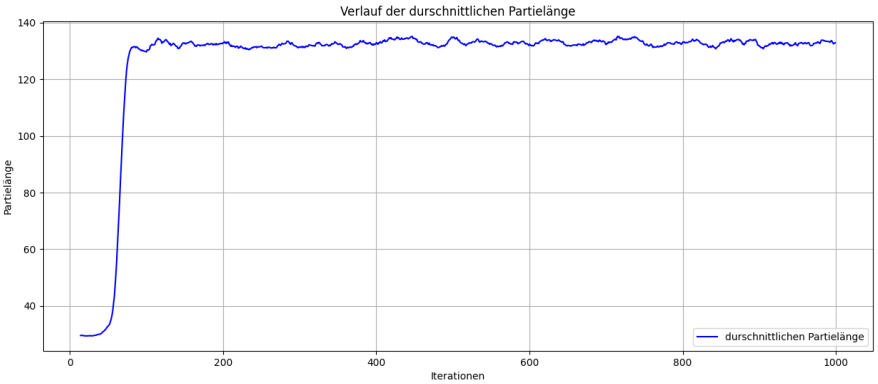


Abbildung 24: VDAC: durchschnittliche Länge einer Partie

In Abbildung 24 wird darüber hinaus die durchschnittliche Länge einer Episode über den Verlauf der Trainingseinheit dargestellt. Die Länge der Partie wird entlang der y-Achse dargestellt, wohingegen die Iterationen auf der x-Achse aufgeführt sind. Diese ist dem in Abbildung

23 gezeigten Trend sehr ähnlich, mit einer deutlichen Steigerung der Partielängen im ersten Zehntel des Trainings und einem darauffolgenden Seitwärtstrend, welcher bis zum Abschluss des Trainings beibehalten wird.

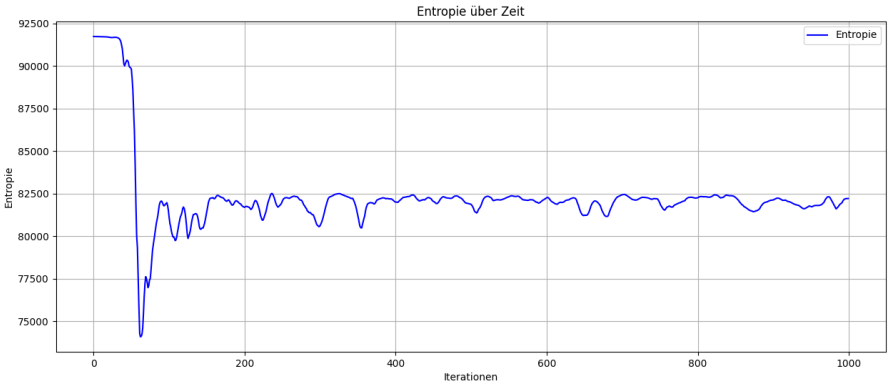


Abbildung 25: VDAC: Strategie Entropie

Die Abbildung 25 zeigt die Entropie der Strategie für die VDAC-Agenten über den Verlauf der Trainingseinheit hinweg. Entlang der y-Achse wird die Höhe der Entropie dargestellt, und die x-Achse zeigt die entsprechenden Iterationen. Deutlich zu erkennen ist ein starker Abwärtstrend zu Beginn des Trainings, welcher anschließend zwar in eine Seitwärtsbewegung übergeht, aber doch recht starke Fluktuation während dieser Bewegung aufzeigt.

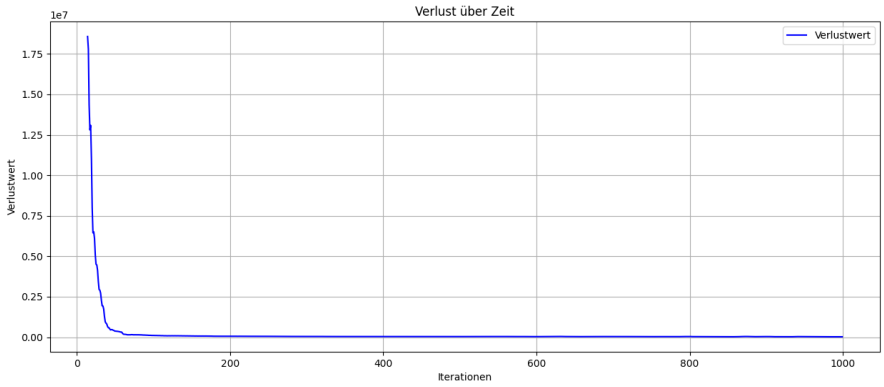


Abbildung 26: VDAC: Verlust über Zeit

Als Letztes zeigt sich in Abbildung 26 der Verlauf der Verlustfunktion über das gesamte Trainingsvorhaben hinweg. Die y-Achse repräsentiert die Höhe des Verlustwertes, und die Iterationen werden auf der x-Achse visualisiert. Hier ist eine stark fallende Tendenz im ersten Zehntel des Trainings zu erkennen, welche dann in eine Seitwärtsbewegung, übergeht, die bis zum Ende des Trainings anhält.

4.4.2 Vergleich gegen regel- und zufallsbasierte Agententeams

Im nachfolgenden Abschnitt wird das nach Abschluss des VDAC-Trainings erstellte Modell gegen die Referenzagenten-Teams von Pommerman antreten gelassen. Zu diesem Zweck wurde das entsprechende Modell für die Dauer einer Iteration erneut in die Pommerman-Umgebung geladen. Basierend auf den in diesem Intervall gesammelten Daten wurde eine prozentuale Aufschlüsselung von Siegen, Niederlagen und Unentschieden erstellt.

Name	**AI (%)**	**Gegner (%)**	**Unent. (%)**	**Gesamt**
AI vs. regelbasierte	119 (12,29 %)	849 (87,71 %)	0 (0,00 %)	968 (100 %)
AI vs. zufallsbasierte	706 (54,18 %)	572 (43,90 %)	25 (1,92 %)	1303 (100 %)

Tabelle 15: Vergleich der Leistung VDAC gegen regelbasiert und zufallsbasierte Agenten

In Tabelle 15 sind die Ergebnisse des Aufeinandertreffens der regelbasierten sowie zufallsbasierten Agententeams und dem VDAC-Agententeam dargestellt. Im direkten Vergleich mit den regelbasierten Agententeams gewann das VDAC-Team 12,29 % der Partien und verlor 87,71 % der Partien. Gegen das zufallsbasierte Agententeam konnte das VDAC-Team in 54,18 % der Fälle siegen und verlor in 43,90 % der Partien gegen dieses. Nur im Vergleich mit dem zufallsbasierten Team kam es in 1,92 % der Partien zu dem Ergebnis Unentschieden. Zur Überprüfung der statistischen Signifikanz der Daten wurde der Chi-Quadrat-Test in seiner Ausprägung als Verteilungstest verwendet. Unter der Prämisse, dass alle drei in Tabelle 15 gelisteten Ergebniskategorien (Sieg, Niederlage, Unentschieden) bei zufälliger Verteilung zu gleichen Teilen, also jeweils zu einem Drittel, erscheinen, wurde ein Signifikanzniveau von 0,05 zugrunde gelegt. Nach dieser Analyse lassen sich die besagten Ergebniskategorien anhand der Gesamtzahl der gespielten Spiele als statistisch relevant klassifizieren.

4.4.3 Vergleich Basis- und Individualbelohnungsfunktion

Im folgenden Kapitel wird gemäß dem im Kapitel „Klassifizierung der Umgebung“ beschriebenen Schema ein Vergleich zwischen den Belohnungssystemen „Individual“ und „Basis“ unter Verwendung des Algorithmus VDAC durchgeführt. Dies dient zur Legitimation der im vorherigen Kapitel angewandten Trainings- und Belohnungsstrategien. Das testweise durchgeführte Training war auf 300 Iterationen je Variante beschränkt. Die Wahl dieser Anzahl zielt darauf ab, eine effektive Untersuchung der Ergebnisse zu gewährleisten, ohne einen übermäßigen Anspruch an Rechenzeit und Rechenleistung. Sie schafft eine verlässliche Grundlage für die notwendigen Analysen und den Vergleich zwischen den beiden Systemen.

In Tabelle 16 werden die Resultate der Partien nach den jeweiligen Trainingssitzungen mit den zugehörigen Gegnerteams präsentiert. Dabei hat das individuelle Belohnungssystem in allen drei Kategorien (AI vs. AI, AI vs. Zufall, AI vs. Regel) einen Vorsprung in Bezug auf die prozentual gewonnenen Aufeinandertreffen gezeigt. Diese Daten stammen aus einem über eine Iteration laufenden Turnier, in welchem die trainierten Agententeams gegen ihre entsprechenden Gegner antraten. Ein gewisser Mehrwert durch das individuelle Belohnungssystem

wird offensichtlich, besonders in den Kategorien AI vs. Zufall und AI vs. Regel, welche als am wichtigsten für diesen Vergleich gelten.

Type	Name	AI (%)	Gegner (%)	Unent. (%)	Gesamt
Basis	AI vs. AI	98 (55.68 %)	75 (42.61 %)	3 (1.70 %)	176
Basis	AI vs. Zufall	130 (51.06 %)	112 (47.66 %)	3 (1.28 %)	235
Basis	AI vs. Regel	19 (9.55 %)	180 (90.45 %)	0 (0.00 %)	199
Individuell	AI vs. AI	107 (50.47 %)	104 (49.06 %)	1 (0.47 %)	212
Individuell	AI vs. Zufall	143 (55.86 %)	109 (42.58 %)	4 (1.56 %)	256
Individuell	AI vs. Regel	24 (12.24 %)	172 (87.76 %)	0 (0.00 %)	196

Tabelle 16: Ergebnispunktzahl Basis gegen individuelles Belohnungssystem VDAC

Darüber hinaus werden in Abbildung 27 die Metriken dargestellt, die während des Testtrainings gesammelt wurden. Jedes der beiden Diagramme weist für jede der getesteten Varianten einen eigenen Eintrag auf. Bei der Betrachtung der durchschnittlichen Belohnung im oberen Teil der Abbildung sticht insbesondere das positive Ergebnis der Trainingskombination „individuelles Belohnungssystem gegen regelbasierte Agententeams“ hervor. Dies wird durch die in gelb dargestellte Steigerung verdeutlicht. Eine ähnlich positive Tendenz zeigt sich auch im unteren Teil der Abbildung, welche die durchschnittliche Rundenlänge abbildet.

4.4.4 Ergebnisinterpretation

In diesem Teil wird anhand der Daten aus dem Kapitel „Trainingsevaluation und Auswertung“ sowie dem Vergleich der regelbasierten und zufallsbasierten Agententeams und einer manuellen Sichtprüfung der Gesamteindruck des Trainings des Algorithmus VDAC dargelegt. Eine isolierte Betrachtung der dargestellten Trainingskennzahlen vermittelt ein durchweg positives Bild des Trainingsverlaufs. Insbesondere die Diagramme bezüglich der erhaltenen Belohnungen und der durchschnittlichen Länge einer Partie weisen einen sehr guten Verlauf auf. Selbst bei einer absoluten Betrachtung der Zahlenwerte beider Diagramme bestätigt sich diese positive Tendenz. Das Diagramm zur Darstellung der Verlustfunktion zeigt einen idealen Verlauf. Besonders hervorzuheben ist hier die Annäherung gegen null. Dies lässt darauf schließen, dass eine Strategie entwickelt werden konnte, um den Verlust so minimal wie möglich zu halten. Die Kennzahl „Entropie“ gibt Indikatoren für das Verhältnis zwischen Erkundung und Erforschung der Agentenstrategie an. Ein stark fallender Trend zu Beginn weist darauf hin, dass das Agententeam immer sicherer in der Wahl seiner Aktionen wurde und die Agenten generell weniger erkundeten, sondern sich mehr auf bereits erlernte Verhaltensweisen verließen. Der seitwärtige Trend bestätigt diese Annahme und legt nahe, dass das Agententeam ein für das Szenario passendes Verhältnis zwischen Erkundung und Erforschung finden konnte. Insgesamt zeigen die Metriken, dass die Agenten in der Lage waren, eine Strategie zu entwickeln, um mit den Unwägbarkeiten der Pommerman-Umgebung zurechtzukommen. Diese Interpretation wird durch die Betrachtung der absoluten Werte der Belohnungen bestätigt, die durchweg über Null liegen. Allerdings weisen alle Diagramme

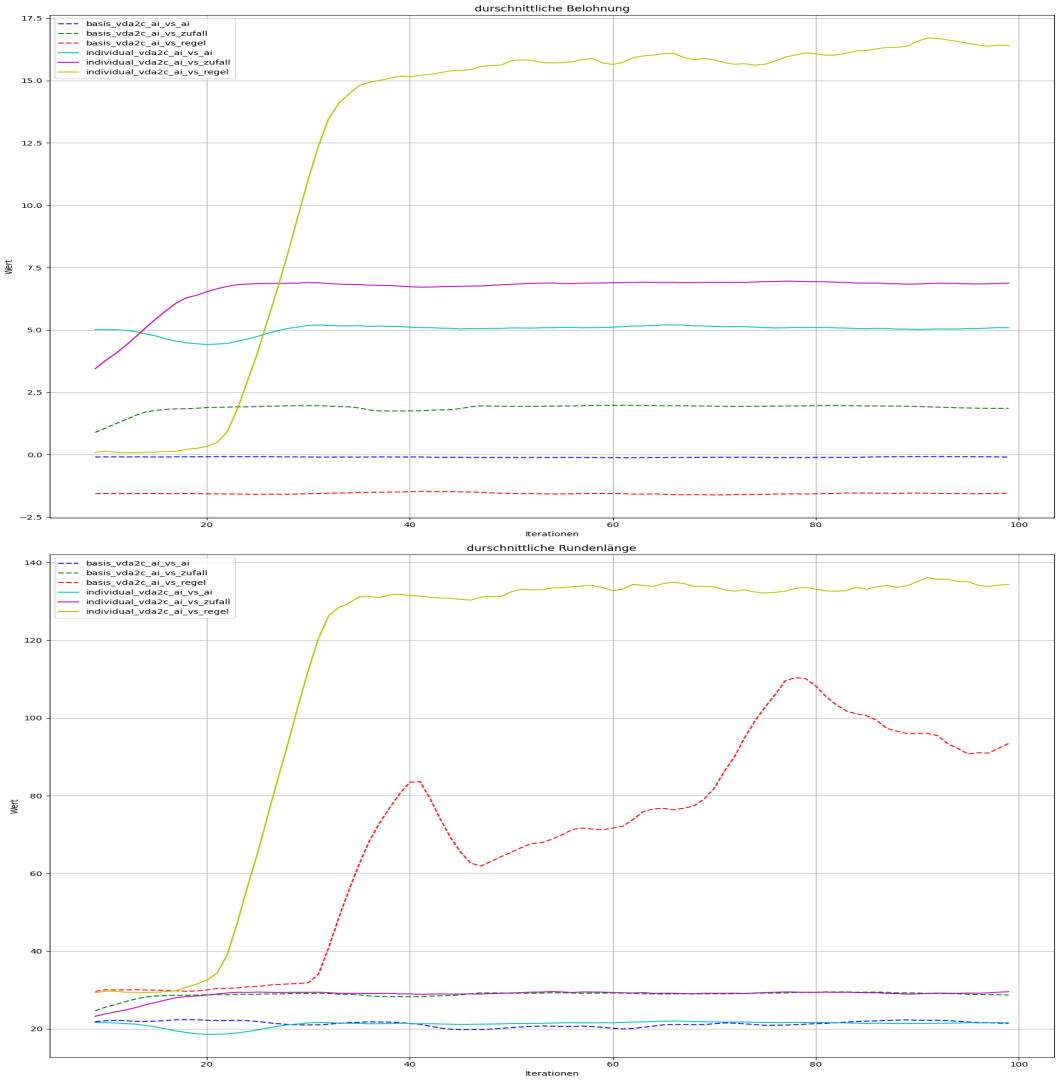


Abbildung 27: Ergebnisgrafen Trainings Basis gegen individuelles Belohnungssystem VDAC

einen ähnlichen Verlauf mit starken Veränderungen im ersten Zehntel des Trainings und einer anschließenden Seitwärtsbewegung auf. Dies könnte darauf hinweisen, dass ab diesem Zeitpunkt bereits eine vermeintlich optimale Strategie entwickelt wurde, da basierend auf den Kennzahlen ab diesem Zeitpunkt keine wesentlichen Veränderungen mehr festgestellt wurden. Gleichermaßen kann nicht ausgeschlossen werden, dass sich der Agent in einem lokalen Optimum befindet und nicht eine optimale Strategie entwickelt hat, wobei dann anzunehmen ist, dass er im weiteren Trainingsverlauf dieses lokale Optimum nicht verlassen konnte.

Wenn die Resultate gegen zufalls- und regelbasierte Agententeams in die Evaluation einbezogen werden, ergibt sich ein abweichendes Bild. Dies entspricht ähnlichen Herausforderungen, die bereits in den Abschnitten zu VDN und QMIX dargestellt wurden. Zusammengefasst zeigt auch das VDAC-Agententeam Schwierigkeiten, gegen das regelbasierte Team zu gewinnen. Das Hauptproblem besteht weiterhin darin, dass die Agenten sich mit einer zunehmenden Anzahl an Runden schließlich selbst eliminieren. Ob die Eliminierung durch eigene Bomben oder eigenes Feuer verursacht wird, verteilt sich gleichmäßig, ein klarer Trend ist hier nicht erkennbar. Wie bei VDN und QMIX sind die Ergebnisse gegen zufallsbasierte Agententeams deutlich besser. Hierbei liegt die Stärke des Modells vor allem darin, länger zu überleben als es die zufallsbasierten Agenten tun. Die gesammelten Informationen aus dem durchgeführten Vergleich mit den Pommerman-Referenzagenten werden durch manuelle Beobachtungen einzelner Partien des Modells bestätigt. Abschließend wurde gezeigt, dass durch die im vorherigen Abschnitt durchgeführte Validierung des besten Belohnungssystems, in Kombination mit dem Vergleich der Systeme unter Berücksichtigung verschiedener Trainingsgegner Konstellationen, der Einsatz der gewählten Variante am effektivsten erscheint.

4.5 Auswertung und Zusammenfassung des Trainingsergebnisses

Im nachfolgenden Kapitel werden die Ergebnisse aller Algorithmen in einen Gesamtkontext gestellt. Es erfolgt eine Eingruppierung der eingesetzten Algorithmen zur Beantwortung einer der zentralen Forschungsfragen dieser Arbeit. Außerdem wird bewertet, inwieweit das Gesamttrainingsvorhaben als erfolgreich zu bezeichnen ist. Alle drei trainierten Algorithmen folgen einem ähnlichen Schema und basieren auf einem ähnlichen technologischen Fundament, dem VDN. Daher weisen alle betrachteten Algorithmen ein ähnliches Verhalten und ähnliche Tendenzen während des Trainings auf. Jeder Algorithmus zeigt, isoliert betrachtet, positive Tendenzen hinsichtlich der Fähigkeit, Strategien zur Bewältigung der Umgebung zu entwickeln.

Jedoch konnte keiner der verwendeten Algorithmen zuverlässig und in der Mehrzahl der Partien gegen die regelbasierten Agententeams gewinnen. Strategien, die hingegen in der Mehrzahl der Partien zu Siegen führten, konnten alle Algorithmen entwickeln, wenn sie gegen zufallsbasierte Agententeams angetreten sind. Besonders hervorzuheben ist der QMIX-Algorithmus, der im Vergleich die höchste Quote an Siegen gegen diese erzielen konnte. Im Gegensatz dazu waren die VDAC-Agenten gegen zufallsbasierte Agententeams nur in circa jedem zweiten Aufeinandertreffen siegreich. Dass alle trainierten Modelle gegen diese gewinnen konnten, lag in allen Fällen daran, dass die im Trainingsverlauf entwickelten Strategien dazu führten, dass sich die Agenten nicht so schnell selbst zerstörten wie die zufallsbasierten Agenten. Dabei unterscheiden sich die Modelle in ihrem Vorgehen. Das QMIX-Modell

hat eine Strategie entwickelt, bei der es möglichst wenige Aktionen ausführt, um das Risiko einer Selbsteliminierung so gering wie möglich zu halten. VDAC- und VDN-Modelle haben hingegen effizienter gelernt, sich auf dem Spielfeld zu bewegen. Dennoch scheint basierend auf den Siegesquoten die Strategie des QMIX-Modells effektiver zu sein als die der beiden anderen betrachteten Algorithmen.

Diese gefundene Strategie aller Algorithmen konnte jedoch in keinem Fall zuverlässig und erfolgreich gegen die regelbasierten Teams eingesetzt werden. Dieses Defizit spiegelt sich unmittelbar in den Ergebnissen wider. Keines der Modelle konnte eine zufriedenstellende Quote an Siegen erreichen. Die regelbasierten Agenten wiesen eine deutlich geringere Rate an Selbstzerstörungen auf. Bei Beobachtungen wurde nur selten festgestellt, dass die regelbasierten Agenten überhaupt von dem Problem der Selbsteliminierung betroffen waren. Auf der anderen Seite zeigten Beobachtungen, dass die Agenten aller Modelle sich entweder irgendwann selbst zerstörten, beispielsweise durch unachtsames Legen einer Bombe, oder dass sie einer direkten Bombenexplosion nicht ausgewichen sind, beziehungsweise aufgrund von Navigationsproblemen in die nachfolgenden Flammen navigierten. Trotz der eingesetzten individuellen Belohnungsfunktion, welche dieses Verhalten eindeutig negativ bewertet, wurde dieses Verhalten nie vollends abgelegt. Positiv hingegen zu bemerken ist, dass das QMIX-Agententeam in etwa 27 % der Partien gegen das regelbasierte Team gewinnen konnte. Beobachtungen zeigten zudem, dass das QMIX-Team eine Strategie entwickelt hat, bei der die Agenten hauptsächlich abwarteten, bis das gegnerische Team alle nötigen Wege freigeräumt hat. Anschließend versuchen die QMIX-Agenten, Bomben in der Nähe der Gegner zu legen.

Die Tabelle 17 zeigt konsolidiert die Ergebnisse aller Modelle im Vergleich zu zufalls- und regelbasierten Agententeams. Zusammenfassend zeigte sich, dass die Trainings in teilweise optimale Strategien mündeten, die jedoch nicht ausreichten, um gegen regelbasierte Agenten zu besiegen. Ihre Stärken konnten die Modelle hauptsächlich im direkten Aufeinandertreffen mit zufallsbasierten Agententeams ausspielen. Alle Modelle hatten mit ähnlichen Herausforderungen zu kämpfen, wobei vornehmlich die Navigation und die Verhinderung von ungewollter Selbsteliminierung für alle Modelle die größten Herausforderungen darstellten. Abschließend zeigt sich, unter Berücksichtigung aller evaluierten Kennzahlen, dass das mittels QMIX-Algorithmus trainierte Modell die beiden anderen Modelle übertroffen hat.

Agent		Gewonnen (%)	Verloren (%)	Unentschieden (%)	Gesamt
QMIX	regelbasiert	19 (27,94 %)	49 (72,06 %)	0 (0,00 %)	68
	zufallsbasiert	129 (86,58 %)	17 (11,41 %)	3 (2,01 %)	149
VDN	regelbasiert	43 (15,52 %)	234 (84,48 %)	0 (0,00 %)	277
	zufallsbasiert	243 (61,21 %)	146 (36,78 %)	8 (2,02 %)	397
VDAC	regelbasiert	119 (12,29 %)	849 (87,71 %)	0 (0,00 %)	968
	zufallsbasiert	706 (54,18 %)	572 (43,90 %)	25 (1,92 %)	1303

Tabelle 17: Zusammenfassung Leistung verschiedener Agenten gegen regelbasiert und zufallsbasierte Agenten

4.5.1 Explorative Analyse und Handlungsempfehlung

Das Kapitel endet mit einer explorativen Analyse der Algorithmen, um deren Stärken und Schwächen zu ergründen. Darüber hinaus werden Handlungsempfehlungen formuliert, die auf den während des Trainings gemachten Erfahrungswerten basieren. Die im Grundlagenkapitel beschriebenen Algorithmen unterscheiden sich in der Herangehensweise jeweils nur in einem gewissen Maße. Jedoch bleibt die Grundannahme, dass eine gemeinsame Wertfunktion in eine agentenspezifische Einzelwertfunktion zerlegt wird, bei allen Algorithmen gleich. Diese Stärke kann, je nach betrachtetem Szenario, zugleich auch die größte Schwäche der Algorithmen darstellen. Per Definition dieser Struktur ist die Aktion eines Einzelnen innerhalb eines Teamkonstruktes nicht von besonders hoher Bedeutung, die Teamleistung ist ausschlaggebend. Das kann in Szenarien, in denen es essenziell ist als Team gemeinsam zu agieren, von Vorteil sein. Zum Beispiel im Szenario „Hide and Seek“ von OpenAI, in welchem beide Agenten zur Zielerreichung kooperieren müssen. Pommerman hingegen bietet zwar auch ein kooperatives Szenario, aber die Leistung des Einzelnen wiegt stärker als mögliche Teamleistungen. Ein Agent kann die Partie alleine gewinnen, ohne seinen Teamkameraden je im Sichtfeld gehabt zu haben. Gerade die Tatsache, dass eine eingeschränkte Sicht dazu führen kann, dass Teammitglieder sich während einer Partie nie sehen, aber dennoch die Belohnung von dem Teamkameraden abhängig ist, führt nach Beobachtung oft zu fehlerhaften Entscheidungen der Agenten.

So kann es beispielsweise vorkommen, dass durch die Art der Belohnungsausschüttung ein Agent negative Belohnungen erhält, weil er sich selbst eliminiert. Der andere Agent des Teams gewinnt jedoch die gesamte Runde für das Team, nach der Eliminierung des ersten Agenten. Daher erhält der Agent, der eigentlich unerwünschtes Verhalten gezeigt hat, ebenfalls eine positive Belohnung und lernt somit, dass Selbstzerstörung vorteilhaft sein kann. Bei genauerer Betrachtung stellt das natürlich eine deutlich falsche Strategie dar. Eine Interpretation des Geschehenen besteht darin, dass aufgrund der Konstruktion wie VDN mit Belohnung umgeht, in dem es die Belohnungen aller Teammitglieder zusammenfasst und anschließend gleichmäßig auf die Teammitglieder verteilt, zum benannten Verhalten beigetragen hat. Neben der Herausforderung, sich nicht durch eigene Bomben oder Feuer zu eliminieren, hatten alle Agenten Probleme mit der Navigation im Spielfeld. Es zeigte sich, dass die ständig wechselnde Anordnung der Elemente auf dem Spielfeld große Schwierigkeiten bereitete. Trotz des Einsatzes einer angepassten Belohnungsfunktion, die den Agenten zeigen sollte, welches Verhalten erwünscht ist, konnten diese genannten Herausforderungen nicht abschließend durch entsprechend entwickelte Strategien gemeistert werden.

Basierend auf diesen Erkenntnissen schließt dieses Kapitel mit der Empfehlung ab, zukünftig, wenn möglich, mit einem vortrainierten Modell zu arbeiten. Dieses sollte idealerweise bereits grundlegende Fähigkeiten, insbesondere in Bezug auf die Navigation im Spielfeld, erlernt haben. Darüber hinaus könnte es erfolgversprechend sein, eine Form des imitierenden Lernens einzusetzen, bei der der Agent zunächst auf Basis von vorgefilterten Daten trainiert wird. Diese Daten könnten entweder durch das Spielen in der Pommerman-Umgebung oder durch das Sammeln von Informationen während des Spiels des regelbasierten Agenten erlangt werden. Abschließend sei angemerkt, dass, sollte das Problem der Selbstzerstörung durch geeignete Strategien der Agenten behoben werden, dies zu einer wesentlichen Ver-

besserung von Sieg- und Niederlagequoten beitragen würde. Sollten diese Trainingsansätze keine erfolgversprechenden Ergebnisse liefern, wäre es ebenso möglich, von einem VDN-Ansatz abzuweichen und einen Algorithmus zu wählen, der die Eigenleistung der Agenten stärker belohnt, wie es beispielsweise durch den Einsatz von Independent Q-Learning der Fall wäre.

5 Fazit

Im Rahmen dieser Arbeit wurde das Thema des Multi-Agent Reinforcement Learning betrachtet. Das Hauptziel bestand darin, die Funktionsweise der drei verwendeten Algorithmen (VDN, QMIX, VDAC) innerhalb der Pommerman-Umgebung zu evaluieren. Ein besonderer Schwerpunkt lag auf dem direkten Vergleich dieser Algorithmen mit den von Pommerman bereitgestellten Referenzagenten, die entweder zufällig oder regelbasiert agieren. Alle eingesetzten Algorithmen wurden von Grund auf neu trainiert, ohne Verwendung von vorab trainierten Modellen. Um optimale Trainingsbedingungen zu schaffen, wurden mittels verschiedener Techniken die optimalen Hyperparameter für alle Algorithmen bestimmt. Weiterhin wurden durch verwendete Testszenarien im Hinblick auf Gegnerteam und Belohnungssystem optimale Trainingspartner ermittelt. Ein weiteres Ziel dieser Arbeit bestand darin, das Basis-Belohnungssystem von Pommerman mit einem individuellen Belohnungssystem, das im Rahmen dieser Arbeit entwickelt wurde, zu vergleichen. Dies sollte helfen, die ideale Trainingsumgebung zu identifizieren. Das Projekt nutzt MARLlib als Framework, welches auf RLib aufbaut und zusätzliche Multi-Agent-Funktionalitäten sowie eine Grundintegration der Pommerman-Umgebung bietet. Für diese Arbeit wurde die Pommerman-Umgebung modifiziert, um eine Anpassung des Belohnungssystems zu ermöglichen. Während im Standard-Pommerman die Belohnungen nur bei Sieg oder Niederlage ausgegeben werden, wurde in der modifizierten Version nach jeder Runde eine Belohnung entsprechend der aktuellen Situation vergeben. Trotz der Anpassung des Belohnungssystems zeigten die Modelle Schwierigkeiten, selbstzerstörerisches Verhalten zu verhindern. Einige Modelle entwickelten Strategien, die auf möglichst wenige Aktionen abzielten, um Selbstzerstörungsrisiken zu minimieren. Diese Strategien waren gegenüber zufallsbasierten Agenten teils erfolgreich, aber gegenüber regelbasierten Agenten weniger effektiv. Ein zentrales Ergebnis dieser Arbeit ist die Erkenntnis, dass Algorithmen, die auf Value-Decomposition basieren, möglicherweise nicht optimal für die Pommerman-Umgebung geeignet sind. Gerade das Zusammenfassen von Belohnungen und das anschließende gleichmäßige Aufteilen dieser, welches ein Kernbestandteil aller eingesetzten Algorithmen darstellt, kann den Lernprozess der Agenten negativ beeinträchtigen. Als Beispiel sei hier ein Szenario zu nennen, bei dem ein Agent des Teams positive Belohnungsignale bekommt, sein Teamkamerad hingegen ausschließlich negative Belohnungssignale erhält. Durch die Konsolidierung beider Signale findet eine Verwässerung der eigentlichen Signale statt, was zur Förderung von Fehlverhalten beitragen kann.

Als Antwort auf die erste Forschungsfrage dieser Arbeit, welcher Algorithmus am besten geeignet ist für den Einsatz in der Pommerman-Umgebung, hat sich QMIX trotz einiger Defizite als der beste herausgestellt. Dieser hat sich hauptsächlich durch seine positiven Ergebnisse im Vergleich zu einem zufallsbasierten Agententeam hervorgetan. In der geprüf-

ten Pommerman-Variante „PommeTeamCompetition“ konnte QMIX somit das beste Ergebnis aller getesteten Algorithmen erzielen. Zur Beantwortung der zweiten Forschungsfrage, „Wie können die genannten Methoden für ein Partially observable Markov decision process POMDP Szenario angewendet werden?“ wurde das MARLlib-Framework verwendet. Die Pommerman-Umgebung wurde hierzu in Kombination mit MARLlib entsprechend angepasst, indem die Einstellungen so gesetzt wurden, das ein POMDP-Szenario simuliert wurde. Durch diese vorgenommenen Einstellungen wurde die Umgebung so konfiguriert, dass die Agenten nicht das gesamte Spielfeld beobachten können, was die Komplexität und Herausforderung für die Algorithmen erhöhte. Unter anderem wurde sich durch den Einsatz von Grid-Suche, Zufallssuche und manueller Suche an ein Optimum der Hyperparameter angenähert. Zudem wurde die angepasste Belohnungsfunktion verwendet, um den Agenten während des Trainingsprozesses und den spezifischen Herausforderungen des gewählten Szenarios zu unterstützen.

Abschließend zeigt das Ergebnis dieser Arbeit, dass für die Verwendung der Pommerman-Umgebung möglicherweise ein anderer Ansatz als Value-Decomposition in Erwägung gezogen werden sollte. Das Fazit lautet daher, dass trotz umfassender Optimierung der Hyperparameter und der Bestimmung der idealen Trainingsumgebung und -konstellation es nicht möglich war für die Agenten eine optimale Strategie zu entwickeln. Anzumerken ist, dass alle im Rahmen der Arbeit erwähnten Bestandteile über mein zur Verfügung gestelltes Git-Repository [4] einsehbar sind. Zusammenfassend ergänzt diese Arbeit den wissenschaftlichen Diskurs durch einen Beitrag, der aufzeigt, wie die hier behandelten Algorithmen in einer POMDP-Umgebung angewendet und evaluiert werden können. Zudem wurde aufgezeigt, wie die Optimierung der Hyperparameter und die Anpassung der Belohnungsfunktion zu einer Verbesserung des Trainingsergebnisses führen, wenngleich es dennoch nicht ausreicht, die Leistungsfähigkeit eines regelbasierten Referenzagenten zu übertreffen.

5.1 Ausblick und mögliche Optimierungen

Die gewonnenen Erkenntnisse legen den Grundstein für zukünftige Forschungen und Untersuchungen. Es wäre besonders interessant, in zukünftigen Studien zu ermitteln, wie sich nicht auf Value-Decomposition basierende Algorithmen im Vergleich zu den in dieser Arbeit vorgestellten Ergebnissen in der Pommerman-Umgebung verhalten. Es wäre zudem spannend zu beobachten, wie die Agenten im Vergleich zu den Referenzagenten abschneiden, wenn das Training nicht von Grund auf begonnen wurde. Ein Ansatz für zukünftige Arbeiten könnte die Anwendung von imitierendem Lernen mit MARL sein. Hierbei könnten Agenten zuerst anhand von Material trainiert werden, das durch manuelles Spielen von Pommerman oder durch Spiele des regelbasierten Agententeams gesammelt wurde. Zudem wäre es interessant zu untersuchen, inwieweit sich die in dieser Arbeit präsentierten Ergebnisse unterscheiden würden, wenn die Pommerman-Umgebung so modifiziert würde, dass Selbstzerstörung und oder Teamzerstörung nicht möglich ist. Dies wäre deshalb von Relevanz, da, wie dargestellt, das selbstzerstörerische Verhalten einzelner Akteure den größten negativen Einfluss auf die Gesamtergebnisse hatten. Eine weitere Überlegung wäre, das Training über einen län-

[4] https://github.com/julz-hub/master_thesis

geren Zeitraum in einem Grafikkarten-Cluster durchzuführen, da durch die Nutzung von handelsüblicher Konsumenten Hardware lediglich ein begrenztes Training der Modelle möglich war. Obwohl die Algorithmen in dieser Arbeit nicht ihre volle Stärke zeigen konnten, bleibt das Gebiet des Multi-Agent Reinforcement Learning von hohem Interesse. Insbesondere das Pommerman-Szenario, das zum Zeitpunkt dieser Arbeit noch wenig als Forschungsobjekt genutzt wurde, bietet viele Möglichkeiten zur Optimierung und Weiterentwicklung.

5.2 Kritische Reflexion

Die Arbeit an dieser Abschlussarbeit war eine lehrreiche Erfahrung mit vielen Herausforderungen. Während der Implementierungsphase zeigte sich trotz umfangreicher Recherche, dass einige Aspekte nicht wie ursprünglich angenommen funktionierten. Dies erforderte eine intensivere Beschäftigung mit den Algorithmen und der Pommerman-Umgebung. Insbesondere die Nutzung neuerer Grafikkartenmodelle für das Training war problematisch, da bestimmte Abhängigkeiten im Pommerman-System dies verhinderten. Die Einbindung von Pommerman in MARLib stellte an einigen Punkten besondere Herausforderungen dar, die vor dem Training behoben wurden. Trotz dieser Herausforderungen konnte eine vertiefte Kenntnis dieser Systeme und Implementierungen erlangt werden. Eine besondere Herausforderung war es trotz alledem, dass die trainierten Modelle nicht gegen die Referenzagenten bestehen konnten. Dies erforderte eine eingehende Überprüfung der Implementierung und nahm viel Zeit in Anspruch, um sicherzustellen, dass die aktuell gewählte Implementierung nicht das Kernproblem darstellt. Dennoch lieferte das Ergebnis wertvolle Erkenntnisse in Bezug auf den Einsatz von Value-Decomposition-Algorithmen in der Pommerman-Umgebung.

6 Literaturverzeichnis

[1] Saddam Abdulwahab, Mohammed Jabreel und Dr Moreno. “Deep Learning Models for Paraphrases Identification”. 7. Sep. 2017. DOI: 10.13140/RG.2.2.15743.46240.

[2] Charu C. Aggarwal. *Neural Networks and Deep Learning: A Textbook*. Cham: Springer International Publishing, 2018. ISBN: 978-3-319-94462-3 978-3-319-94463-0. DOI: 10.1007/978-3-319-94463-0. URL: http://link.springer.com/10.1007/978-3-319-94463-0 (besucht am 15. 11. 2022).

[3] Ethem Alpaydin. *Maschinelles Lernen*. De Gruyter Oldenbourg, 19. Jan. 2022. ISBN: 978-3-11-074019-6. DOI: 10.1515/9783110740196. URL: https://www.degruyter.com/document/doi/10.1515/9783110740196/html (besucht am 20. 03. 2023).

[4] Shalabh Bhatnagar u. a. “Natural Actor–Critic Algorithms”. In: *Automatica* 45.11 (Nov. 2009), S. 2471–2482. ISSN: 00051098. DOI: 10.1016/j.automatica.2009.07.008. URL: https://linkinghub.elsevier.com/retrieve/pii/S0005109809003549 (besucht am 04. 02. 2023).

[5] F. Bredell, H. A. Engelbrecht und J. C. Schoeman. *Credit-Cognisant Reinforcement Learning for Multi-Agent Cooperation*. 18. Nov. 2022. arXiv: 2211.10100 [cs]. URL: http://arxiv.org/abs/2211.10100 (besucht am 14. 12. 2022). preprint.

[6] Lucian Busoniu, Robert Babuska und Bart De Schutter. “A Comprehensive Survey of Multiagent Reinforcement Learning”. In: *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 38.2 (März 2008), S. 156–172. ISSN: 1094-6977, 1558-2442. DOI: 10.1109/TSMCC.2007.913919. URL: https://ieeexplore.ieee.org/document/4445757/ (besucht am 01. 02. 2023).

[7] Lucian Buşoniu, Robert Babuška und Bart De Schutter. “Multi-Agent Reinforcement Learning: An Overview”. In: *Innovations in Multi-Agent Systems and Applications - 1*. Hrsg. von Dipti Srinivasan und Lakhmi C. Jain. Bearb. von Janusz Kacprzyk. Bd. 310. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, S. 183–221. ISBN: 978-3-642-14434-9 978-3-642-14435-6. DOI: 10.1007/978-3-642-14435-6_7. URL: http://link.springer.com/10.1007/978-3-642-14435-6_7 (besucht am 03. 10. 2023).

[8] *ChatGPT: Optimizing Language Models for Dialogue*. OpenAI. 30. Nov. 2022. URL: https://openai.com/blog/chatgpt/ (besucht am 28. 12. 2022).

[9] Kyunghyun Cho u. a. *On the Properties of Neural Machine Translation: Encoder-Decoder Approaches*. 7. Okt. 2014. DOI: 10.48550/arXiv.1409.1259. arXiv: 1409.1259 [cs, stat]. URL: http://arxiv.org/abs/1409.1259 (besucht am 03. 10. 2023). preprint.

[10] Junyoung Chung u. a. *Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling*. 11. Dez. 2014. DOI: 10.48550/arXiv.1412.3555. arXiv: 1412.3555 [cs]. URL: http://arxiv.org/abs/1412.3555 (besucht am 23. 03. 2023). preprint.

[11] Gabriel Dulac-Arnold, Daniel Mankowitz und Todd Hester. *Challenges of Real-World Reinforcement Learning*. 29. Apr. 2019. arXiv: 1904.12901 [cs, stat]. URL: http://arxiv.org/abs/1904.12901 (besucht am 03. 10. 2023). preprint.

[12] Wolfgang Ertel. *Grundkurs Künstliche Intelligenz: Eine praxisorientierte Einführung*. Computational Intelligence. Wiesbaden: Springer Fachmedien, 2021. ISBN: 978-3-658-32074-4 978-3-658-32075-1. DOI: 10.1007/978-3-658-32075-1. URL: https://link.springer.com/10.1007/978-3-658-32075-1 (besucht am 20. 03. 2023).

[13] Ian Goodfellow, Yoshua Bengio und Aaron Courville. *Deep Learning*. Adaptive Computation and Machine Learning. Cambridge, Mass: The MIT press, 2016. ISBN: 978-0-262-03561-3.

[14] Palash Goyal, Sumit Pandey und Karan Jain. *Deep Learning for Natural Language Processing*. Berkeley, CA: Apress, 2018. ISBN: 978-1-4842-3684-0 978-1-4842-3685-7. DOI: 10.1007/978-1-4842-3685-7. URL: http://link.springer.com/10.1007/978-1-4842-3685-7 (besucht am 21. 03. 2023).

[15] Sepp Hochreiter und Jürgen Schmidhuber. "Long Short-Term Memory". In: *Neural Computation* 9.8 (1. Nov. 1997), S. 1735–1780. ISSN: 0899-7667, 1530-888X. DOI: 10.1162/neco.1997.9.8.1735. URL: https://direct.mit.edu/neco/article/9/8/1735-1780/6109 (besucht am 21. 03. 2023).

[16] Ronald A. Howard. *Dynamic Programming and Markov Processes*. Cambridge, Massachusetts: The MIT Press, 1960.

[17] Alexander Jung. *Machine Learning: The Basics*. Machine Learning: Foundations, Methodologies, and Applications. Singapore: Springer Nature, 2022. ISBN: 9789811681929 9789811681936. DOI: 10.1007/978-981-16-8193-6. URL: https://link.springer.com/10.1007/978-981-16-8193-6 (besucht am 21. 03. 2023).

[18] Mykel J. Kochenderfer. *Decision Making under Uncertainty: Theory and Application*. Lincoln Laboratory Series. Cambridge, Massachusetts: The MIT Press, 2015. 323 S. ISBN: 978-0-262-02925-4.

[19] Vijay Konda und John Tsitsiklis. "Actor-Critic Algorithms". In: *Advances in Neural Information Processing Systems*. Bd. 12. MIT Press, 1999. URL: https://papers.nips.cc/paper_files/paper/1999/hash/6449f44a102fde848669bdd9eb6b76fa-Abstract.html (besucht am 11. 08. 2023).

[20] Yann LeCun, Yoshua Bengio und Geoffrey Hinton. “Deep Learning”. In: *Nature* 521.7553 (7553 Mai 2015), S. 436–444. ISSN: 1476-4687. DOI: 10.1038/nature14539. URL: https://www.nature.com/articles/nature14539 (besucht am 28. 09. 2023).

[21] Pascal Leroy, Jonathan Pisane und Damien Ernst. *Value-Based CTDE Methods in Symmetric Two-team Markov Game: From Cooperation to Team Competition*. 30. Nov. 2022. arXiv: 2211.11886 [cs]. URL: http://arxiv.org/abs/2211.11886 (besucht am 14. 12. 2022). preprint.

[22] Michael L. Littman. “Inducing Partially Observable Markov Decision Processes”. In: *Proceedings of the Eleventh International Conference on Grammatical Inference*. International Conference on Grammatical Inference. PMLR, 16. Aug. 2012, S. 145–148. URL: https://proceedings.mlr.press/v21/littman12a.html (besucht am 24. 03. 2023).

[23] Volodymyr Mnih u. a. *Asynchronous Methods for Deep Reinforcement Learning*. arXiv.org. 4. Feb. 2016. URL: https://arxiv.org/abs/1602.01783v2 (besucht am 09. 10. 2023).

[24] Volodymyr Mnih u. a. “Human-Level Control through Deep Reinforcement Learning”. In: *Nature* 518.7540 (7540 Feb. 2015), S. 529–533. ISSN: 1476-4687. DOI: 10.1038/nature14236. URL: https://www.nature.com/articles/nature14236 (besucht am 03. 02. 2023).

[25] Volodymyr Mnih u. a. *Playing Atari with Deep Reinforcement Learning*. 19. Dez. 2013. arXiv: 1312.5602 [cs]. URL: http://arxiv.org/abs/1312.5602 (besucht am 04. 02. 2023). preprint.

[26] Christopher Olah. *Understanding LSTM Networks*. 2015. URL: http://colah.github.io/posts/2015-08-Understanding-LSTMs/ (besucht am 21. 03. 2023).

[27] Liviu Panait und Sean Luke. “Cooperative Multi-Agent Learning: The State of the Art”. In: *Autonomous Agents and Multi-Agent Systems* 11.3 (1. Nov. 2005), S. 387–434. ISSN: 1573-7454. DOI: 10.1007/s10458-005-2631-2. URL: https://doi.org/10.1007/s10458-005-2631-2 (besucht am 01. 02. 2023).

[28] Christos H. Papadimitriou und John N. Tsitsiklis. “The Complexity of Markov Decision Processes”. In: *Mathematics of Operations Research* 12.3 (1987), S. 441–450. ISSN: 0364-765X. JSTOR: 3689975. URL: https://www.jstor.org/stable/3689975 (besucht am 27. 12. 2022).

[29] Bei Peng u. a. *FACMAC: Factored Multi-Agent Centralised Policy Gradients*. 7. Mai 2021. arXiv: 2003.06709 [cs, stat]. URL: http://arxiv.org/abs/2003.06709 (besucht am 14. 01. 2023). preprint.

[30] Tabish Rashid u. a. *QMIX: Monotonic Value Function Factorisation for Deep Multi-Agent Reinforcement Learning*. 6. Juni 2018. arXiv: 1803.11485 [cs, stat]. URL: http://arxiv.org/abs/1803.11485 (besucht am 17. 12. 2022). preprint.

[31] *RNN, LSTM & GRU*. URL: http://dprogrammer.org/rnn-lstm-gru (besucht am 23. 03. 2023).

[32] David E. Rumelhart, Geoffrey E. Hinton und Ronald J. Williams. "Learning Representations by Back-Propagating Errors". In: *Nature* 323.6088 (6088 Okt. 1986), S. 533–536. ISSN: 1476-4687. DOI: 10.1038/323533a0. URL: https://www.nature.com/articles/323533a0 (besucht am 10. 10. 2023).

[33] Dinh Viet Sang und Phan Ngoc Lan. "BK.Synapse: A Scalable Distributed Training Framework for Deep Learning". In: *Proceedings of the 10th International Symposium on Information and Communication Technology*. SoICT '19. New York, NY, USA: Association for Computing Machinery, 4. Dez. 2019, S. 43–48. ISBN: 978-1-4503-7245-9. DOI: 10.1145/3368926.3369690. URL: https://doi.org/10.1145/3368926.3369690 (besucht am 03. 02. 2023).

[34] Pramila P. Shinde und Seema Shah. "A Review of Machine Learning and Deep Learning Applications". In: *2018 Fourth International Conference on Computing Communication Control and Automation (ICCUBEA)*. 2018 Fourth International Conference on Computing Communication Control and Automation (ICCUBEA). Aug. 2018, S. 1–6. DOI: 10.1109/ICCUBEA.2018.8697857.

[35] Jianyu Su, Stephen Adams und Peter A. Beling. *Value-Decomposition Multi-Agent Actor-Critics | VDA2C*. 18. Dez. 2020. DOI: 10.48550/arXiv.2007.12306. arXiv: 2007.12306 [cs]. URL: http://arxiv.org/abs/2007.12306 (besucht am 22. 01. 2023). preprint.

[36] Peter Sunehag u. a. *Value-Decomposition Networks For Cooperative Multi-Agent Learning*. 16. Juni 2017. arXiv: 1706.05296 [cs]. URL: http://arxiv.org/abs/1706.05296 (besucht am 14. 12. 2022). preprint.

[37] Richard S Sutton und Andrew Barto. *Reinforcement Learning: An Introduction*. [Nachdruck]. A Bradford Book. The MIT Press, 1998. ISBN: 0-262-19398-1.

[38] Richard S. Sutton. "Learning to Predict by the Methods of Temporal Differences". In: *Machine Learning* 3.1 (1. Aug. 1988), S. 9–44. ISSN: 1573-0565. DOI: 10.1007/BF00115009. URL: https://doi.org/10.1007/BF00115009 (besucht am 03. 10. 2023).

[39] Richard S. Sutton und Andrew G. Barto. *Reinforcement Learning: An Introduction*. Second edition. Adaptive Computation and Machine Learning Series. Cambridge, Massachusetts: The MIT Press, 2018. 526 S. ISBN: 978-0-262-03924-6.

[40] Karl Tuyls und Gerhard Weiss. "Multiagent Learning: Basics, Challenges, and Prospects". In: *AI Magazine* 33.3 (3 20. Sep. 2012), S. 41–41. ISSN: 2371-9621. DOI: 10.1609/aimag.v33i3.2426. URL: https://ojs.aaai.org/index.php/aimagazine/article/view/2426 (besucht am 01. 02. 2023).

[41] B V Vishwas und Ashish Patel. *Hands-on Time Series Analysis with Python: From Basics to Bleeding Edge Techniques*. Berkeley, CA: Apress, 2020. ISBN: 978-1-4842-5991-7 978-1-4842-5992-4. DOI: 10.1007/978-1-4842-5992-4. URL: http://link.springer.com/10.1007/978-1-4842-5992-4 (besucht am 21. 03. 2023).

[42] Jianhao Wang u. a. "QPLEX: Duplex Dueling Multi-Agent Q-Learning". In: International Conference on Learning Representations. 2. Okt. 2020. URL: https://openreview.net/forum?id=Rcmk0xxIQV (besucht am 02. 10. 2023).

[43] Christopher J. C. H. Watkins und Peter Dayan. "Q-Learning". In: *Machine Learning* 8.3 (1. Mai 1992), S. 279–292. ISSN: 1573-0565. DOI: 10.1007/BF00992698. URL: https://doi.org/10.1007/BF00992698 (besucht am 24. 03. 2023).