technische universität
dortmund

Masterarbeit

Künstliche Intelligenz in der Logistik 4.0: Adaptives Bin Packing mittels Reinforcement Learning

Hermann Foot
September 2020

Gutachter:
Prof. Dr. Katharina Morik*
Dr. Helena Kotthaus*

Externer Betreuer:
Dipl. Inf. Benedikt Mättig†

Fakultät Informatik
Lehrstuhl für
Künstliche Intelligenz

Fraunhofer
IML

*Technische Universität Dortmund
Fakultät für Informatik
Lehrstuhl für Künstliche Intelligenz
https://www-ai.cs.uni-dortmund.de/

†Fraunhofer-Institut für Materialfluss und Logistik IML
Abteilung für Verpackungs- und Handelslogistik
https://www.iml.fraunhofer.de/

Inhaltsverzeichnis

Abstract

Digitization is finding its way more and more into everyday life by integrating intelligent electronics. The industry has not been spared this trend either: in what is now the fourth industrial revolution, the use of intelligent cyberphysical systems plays a central role. With the resulting *Industry 4.0* a working environment shall be developed in which man and machine work together symbiotically. *Digital assistants* can support the worker in his work by reducing physically demanding or monotonous work, faster communication or by providing helpful information. As a result of this cooperation, an increase in productivity is expected.

As one of the fundamental disciplines in industry, this change is also finding favor in *logistics*. Here, intelligent systems help, for example, in the transport of goods, order processing or the packaging of deliveries.

Despite the continuous development of innovative and potentially helpful solutions, these often do not find their way into industrial use. One reason for this is the lack of *empirical knowledge*. Experienced employees can make use on this knowledge to adapt processes to the conditions in the company as well as to external factors. The lack of adaptivity of technical systems is often an obstacle for companies, which can discourage them from using such technologies.

In order to address this problem, solutions are needed with the help of which technical systems can make use of the empirical knowledge of employees and thus can adapt autonomously to the working methods and needs of the company.

The present master thesis investigates possible approaches provided by the field of *machine learning* for the realization of such a system. The considered use case is the packaging process in logistics. For this purpose ideas for the realization of such a system are discussed and an appropriate implementation is worked out. With the help of machine learning models are developed, which can implement a strategy for loading a load carrier influenced by observed expert behaviour. The performance and adaptability of the system is reviewed and rated on the basis of evaluations.

Kapitel 1

Einleitung

1.1 Motivation

Kaum eine Entwicklung hat das Leben der letzten Jahrzehnte so stark beeinflusst, wie die Digitalisierung. Auch an der Industrie ist dieser Trend nicht vorbei gegangen: in der nun vierten industriellen Revolution spielen cyber-physische Systeme und deren intelligente Vernetzung eine zentrale Rolle. Als Resultat soll dabei mit der **Industrie 4.0** eine Industrie entstehen, die agiler, ressourcenschonender und effektiver als ihr Vorgänger ist [22][7].
Mit der Logistik als unerlässlichen Sektor, führt dies einhergehend zu einem Wandel in ebendieser [91]. Eine maßgebliche Rolle in der daraus resultierenden **Logistik 4.0** nehmen **digitale Assistenzsysteme** ein [68]. Hierbei handelt es sich um Systeme, die den Arbeiter in seiner Tätigkeit durch das Bereitstellen von arbeitsbezogenen Informationen oder empfohlenen Arbeitsanweisungen unterstützen [21]. Durch die Integration derartiger Assistenzsysteme in den Arbeitsablauf lassen sich unter anderem Fehler der Arbeiter reduzieren, als auch neues Personal schneller einarbeiten. Insbesondere Letzteres bietet für die Logistik großes Potential. Saisonale Schwankungen in der betrieblichen Auslastung haben den Einsatz von Leiharbeitern und damit eine hohe Mitarbeiterfluktuation zur Folge. Laut Statistik der Bundesagentur für Arbeit waren im Jahr 2019 etwa 10% aller Leiharbeiter in Deutschland in der Logistik beschäftigt, was in etwa 100.000 Menschen entspricht [20]. Umso hilfreicher wäre es dementsprechend, das Anlernen neuer Mitarbeiter mit Hilfe digitaler Assistenten realisieren zu können, um erfahrenes Stammpersonal nicht zusätzlich belasten zu müssen.
Trotz bereitstehender Lösungen finden diese jedoch häufig nicht den Weg in die Industrie [23]. Einer der Hauptgründe liegt hierbei darin, dass erfahrenes Personal, im Gegensatz zu technischen Systemen, Zugriff auf erlerntes **Erfahrungswissen** hat. Hierbei handelt es sich um eine Form des Wissens, die zum einen aus explizitem Wissen besteht, welches auf Kenntnissen über technische Abläufe und Regeln beruht und sich durch diese erklären

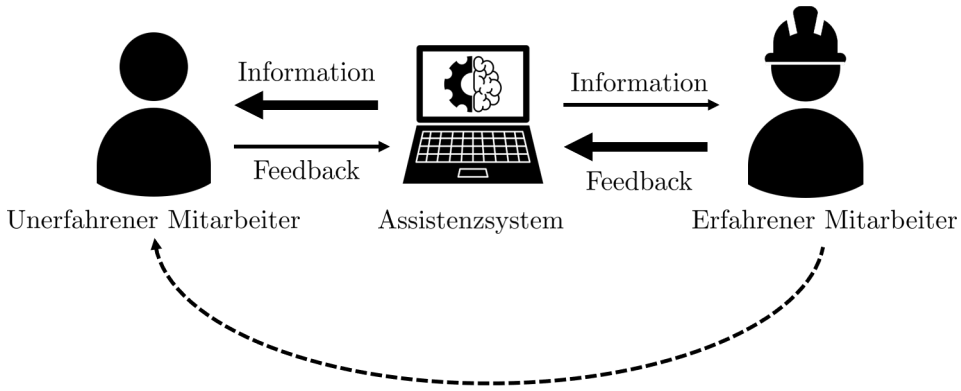


Abbildung 1.1: HETM: Das Assistenzsystem ist in einen auszuführenden Prozess miteingebunden und interagiert mit den beteiligten Mitarbeitern. Mit höherer Erfahrungsstufe des Mitarbeiters steigt der Input in das System, anhand dessen es sich adaptieren kann. Ein unerfahrener Mitarbeiter dagegen erhält mehr Output, um ihn bei der für ihn noch neuen Tätigkeit zu unterstützen.

lässt. Zum anderen gehört implizites Wissen dazu, welches sich auf Erfahrungen stützt, die sich ein erprobter Mitarbeiter im Verlauf seiner Tätigkeit angeeignet hat. Hierzu zählen insbesondere Kenntnisse außergewöhnlicher Situationen und Aspekte betriebseigener Umstände, die den gewohnten Ablauf von Prozessen beeinflussen. Das Erfahrungswissen stellt daher eine entscheidende Komponente des beruflichen Könnens dar [76]. Die damit einhergehende Adaptivität lässt sich nur unter großem Aufwand in bestehende Systeme und Algorithmen von Hand integrieren. Wünschenswert wäre also ein System, welches sich autonom an betriebs- und prozesseigene Umstände und Bedingungen anpassen kann.
Ein Framework zur Realisierung eines solchen Systems bietet dabei das **Human-Experience-Transfer-Model** (HETM) (s. Abb. 1.1) [67]. Hierbei fungieren erfahrene Mitarbeiter als Experten in Bezug auf eine auszuführende Tätigkeit. Ausgeführte Vorgänge werden dabei von einem technischen System erfasst und verarbeitet. Dieses soll das demonstrierte Verhalten reproduzieren und unerfahrenen Mitarbeitern vermitteln können. Der Informationsfluss ist dabei antiproportional: je erfahrener ein Mitarbeiter ist, desto mehr Daten fließen in das technische System ein und weniger Anweisungen werden nach außen ausgegeben. Ein unerfahrener Mitarbeiter dagegen erhält mehr Unterstützung seitens des technischen Systems. Um ein solches Lernverhalten technischerseits implementieren zu können, bieten sich insbesondere Ansätze des **maschinellen Lernens** an [107]. Die Erfahrungen, die ein solches System im Einsatz macht, werden dabei als Daten bereitgestellt, anhand derer gelernt und adaptiert werden soll. Als Resultat soll dabei ein technisches System realisiert werden können, welches auf Basis der gesammelten Erfahrungen das Erfahrungswissen von Mitarbeitern repräsentieren und anwenden kann.

1.2 Verpackungsprozess

Beim Verpackungsprozess handelt es sich um einen intralogistischen Prozess, das heißt, er findet in der internen Logistik eines Betriebes statt. Die Aufgabe hierbei ist es, eine Anzahl **Packstücke** in oder auf einem **Ladungsträger** zu platzieren. Beispiele dafür können das Verpacken von bestellter Ware in einen Versandkarton bei einem Online-Händler oder auch das Beladen einer Palette mit Gütern für die Lagerung innerhalb des betriebseigenen Lagers sein.
Zu unterscheiden ist der Verpackungsprozess von der verwandten Kommissionierung [96][92]. Letztere befasst sich mit der Zusammenstellung von bestimmten Artikeln aus einem Sortiment auf Basis eines Auftrags, wie einer Kundenbestellung. Mit anderen Worten, die für den Auftrag relevanten Artikel werden aus der Gesamtheit des Lagers zu einer Einheit zusammengetragen. Der Verpackungsprozess dagegen befasst sich mit der darauffolgenden geometrischen Anordnung der zum Auftrag gehörigen Artikel zum Zwecke der anschließenden Lagerung oder des Versandes.
Wie genau ein Packer die Packstücke auf dem Ladungsträger anordnet, welche **Packstrategie** er also verfolgt, hängt dabei stark vom angestrebten Ziel, der anschließenden Weiterverarbeitung oder anderer Faktoren, wie der eigenen Lagerverwaltung ab. Betrachten wir beispielsweise erneut den Use Case der Online-Bestellung, so könnte der Packer ein minimales Volumen der Beladung beabsichtigen, wodurch er die Kosten für den Transport reduzieren kann. Ein anderes Beispiel könnte die Beladung einer Palette sein, die innerhalb des betriebseigenen Lagers in hochgelegene Regale gestapelt werden soll. Um einen möglichst sicheren Transport auf den Gabeln des Gabelstaplers zu ermöglichen und das Kippen der Palette zu unterbinden, spielt in diesem Fall die gleichmäßige Gewichtsverteilung eine größere Rolle als beispielsweise das Volumen der Beladung.

1.3 Problemdefinition

Um das Verpackungsproblem mathematisch greifbar zu machen, bedarf es einer Formulierung der Aufgabenstellung. Das zugrunde liegende Problem lässt sich hierbei auf das **Bin-Packing-Problem** (BPP) zurückführen. In unserem Szenario betrachten wir dabei insbesondere die dreidimensionale Variante des Problems. Wir nehmen darüber hinaus an, dass uns lediglich ein Ladungsträger zur Verfügung steht, der in Breite und Länge beschränkt ist. Da in der Praxis häufig keine physikalische Einschränkung der Höhe vorliegt, wie etwa bei einer Palette, wird diese dagegen als unbeschränkt betrachtet. Bei den Packstücken soll es sich dabei um quaderförmige Objekte handeln, die orthogonal auf dem

Ladungsträger platziert werden. Formal lässt sich das betrachtete Problem wie folgt definieren [51]:

Input: Ladungsträger Breite B und Länge L mit $B, L \in \mathbb{R}_{>0}$
n Packstücke mit Breite b_i, Länge l_i, Höhe h_i und Gewicht w_i
mit $b_i, l_i, h_i, w_i \in \mathbb{R}_{>0}$ für $i \in \{0, ..., n\}$

Output: Koordinaten $x_i, y_i, z_i \in \mathbb{R}_{>0}$ für $i \in \{0, ..., n\}$, so dass:

$$
\begin{aligned}
&\min && F(x, y, z) && \\
&\text{s.t.} && x_i + b_i \leq B && \text{(I)} \\
& && y_i + l_i \leq L && \text{(II)} \\
& && && \\
& && s_{ij} + d_{ij} + u_{ij} = 1 && \text{(III)} \\
& && (x_j - x_i) \cdot s_{ij} \geq b_i \cdot s_{ij} && \text{(IV)} \\
& && (y_j - y_i) \cdot d_{ij} \geq w_i \cdot d_{ij} && \text{(V)} \\
& && (z_j - z_i) \cdot u_{ij} \geq h_i \cdot u_{ij} && \text{(VI)} \\
& && && \\
& && s_{ij}, d_{ij}, u_{ij} \in \{0, 1\} && \text{(VII)} \\
& && x_i, y_i, z_i \geq 0 && \text{(VIII)} \\
& && && \\
& && \text{für } i, j \in \{0, ..., n\}. &&
\end{aligned}
\tag{1.1}
$$

Gegeben sind dabei die Dimensionen und Gewichte der Packstücke, als auch die durch den Ladungsträger definierten Beschränkungen. Die resultierenden Platzierungen der Packstücke ergeben sich dabei aus den Ausgabevariablen x_i, y_i und z_i für $i \in \{0, ..., n\}$, wobei n der Anzahl der zu verpackenen Elemente entspricht. Die Koordinaten geben die Position der vorderen linken Ecke des Pakets auf dem Ladungsträger an, wobei dessen vordere linke Ecke den Punkt $(0, 0, 0)^T$ definiert. Die Variablen s_{ij}, u_{ij}, and d_{ij} sind Indikatorvariablen, die angeben, ob sich ein Packstück p_i **s**eitlich links von einem anderen Packstück x_j, **u**nter oder **d**ahinter befindet. Die Abb. 1.2 zeigt ein einfaches Beispiel, welches das gegebene Problem im Zweidimensionalem visualisiert.
Die Bedingungen (I) und (II) stellen sicher, dass jedes platzierte Packstück nicht über die Abmessungen des Ladungsträgers hinausragt. Die Bedingungen (III)-(VI) hingegen stellen sicher, dass sich die Dimensionen der Packstücke nicht überschneiden.
Besondere Aufmerksamkeit gilt dabei der Optimierungsfunktion. Diese wird in unserem Fall durch die abstrakte Funktion F repräsentiert, welche durch die Koordinaten x, y, z der

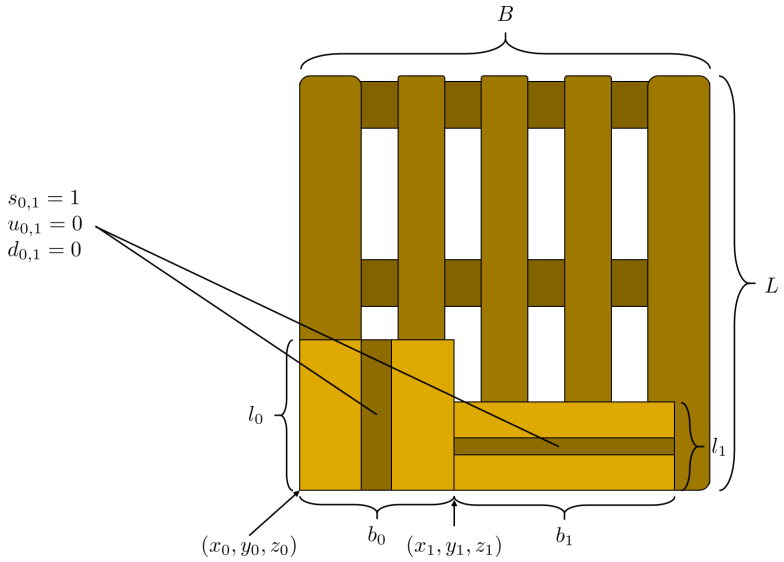


Abbildung 1.2: BPP-Visualisierung: Die Abbildung visualisiert das Problem aus zweidimensionaler Perspektive.

Packstücke parametrisiert ist. Betrachtet man ähnliche Problemstellungen im Bereich Bin-Packings, lassen sich hier beispielsweise Funktionen, die die Höhe oder das eingenommene Volumen der Beladung, wiedergeben. Da wir in unserem Anwendungsfall die angewandte Strategie und die damit verbundene Optimierung am Verhalten des Experten orientieren wollen, wird die tatsächliche Definition der Funktion F offengelassen.
Es sei zu erwähnen, dass wir das Gewicht eines Packstücks als gegeben annehmen, jedoch hierzu keine Restriktionen betrachten. In der Praxis würden sich diese beispielsweise zur Vermeidung von Kippeffekten oder Schaden durch Überladung einsetzen lassen. Das Gewicht kann jedoch in unserem Fall in die zu optimierende Funktion F einfließen und damit relevant werden.
Es sei darüber hinaus zu erwähnen, dass die Definition des Problems offenlässt, in welcher Reihenfolge die Packstücke zu platzieren sind. Wir unterscheiden dabei zwischen der Online- und Offline-Variante des Problems. Bei Ersterer platzieren wir die Packstücke in einer vorgegebenen Reihenfolge, die sich aus den Indizes ergibt. Die Offline-Variante dagegen lässt die Reihenfolge offen und somit frei wählbar. Aus kombinatorischer Sicht ist diese Variante aufgrund der wählbaren Reihenfolge das komplexere Problem. Da beide Varianten eine praktische Relevanz haben, werden beide in dieser Arbeit beachtet.

1.4 Verwandte Arbeiten

Bin Packing

Das zugrundeliegende, algorithmische Problem, welches in der Verpackung betrachtet wird, lässt sich aus dem Bin-Packing ableiten. Neben des in dieser Arbeit betrachteten Use Cases,

findet das Problem zahlreiche weitere Anwendungen wie in der industriellen Zuschneidung, Frachtverladung [34] oder dem Task Scheduling in Multiprozessoren [27] oder FPGAs [103]. Abhängig von Anforderungen der Anwendung, unterscheidet sich die genaue Problemstellung des Bin-Packings, wodurch eine Vielzahl von Variationen des ursprünglichen Problems entstanden sind. Um diese kategorisieren und benennen zu können, schlagen Wäscher et al. [100] eine Topologie für derartige Probleme vor. Hierbei werden diese unter anderem anhand der Dimensionalität (1D/2D/3D), der Anzahl der Ladungsträger (einer/mehrere) oder der Vielfalt der Packstücke (heterogen/homogen) unterschieden. Folgen wir dieser Topologie anhand der oben definierten Charakteristika, lässt sich das in dieser Arbeit betrachtete Problem als 3D-Strip-Packing-Problem (3D-SPP) definieren.
Es lässt sich sowohl für das ursprüngliche BPP als auch für das 3D-SPP zeigen, dass es sich dabei um $\mathcal{NP}$-vollständige Probleme handelt [41][48]. Eine optimale Lösung in polynomieller Zeit ist daher nur möglich, wenn $\mathcal{NP} = \mathcal{P}$ gilt. Aufgrund der Komplexität wurde eine Vielzahl von Ansätzen erforscht und veröffentlicht, die der optimalen Lösung möglichst nah kommen wollen. Neben einer Reihe heuristischer und approximativer Ansätze (u.a. [64][5][15][38][14][42]), wurde zudem die Anwendung von maschinellem Lernen am BPP untersucht.
So veröffentlichten beispielsweise Mao et al. [62] einen Ansatz, bei dem Deep Learning bei der Lösung des 1D-Bin-Packing helfen soll. Die antrainierten Modelle sollen dabei Informationen über Packaufträge extrahieren können, welche dann als Hyperparameter für heuristische Verfahren genutzt werden können.
Einen weiteren Ansatz zeigen die Veröffentlichungen von Hu et al. [51] und Duan et al. [31]. Dabei interpretieren beide Verfahren das BPP als ein kombinatorisches Problem, das mit Sequenz-zu-Sequenz-Modellen beschrieben werden kann. Ein derartiger Ansatz hat bereits gute Ergebnisse bei anderen Problemen, wie z.B. dem Traveling-Salesman-Problem gezeigt [89][8].
Darüber hinaus, dass in den genannten Verfahren andere Varianten des BPP als das 3D-SPP behandelt werden, unterscheiden sich diese zudem hinsichtlich der Verwendung der Daten von der vorliegenden Arbeit. In dem hier betrachteten Fall sollen diese nämlich der Anpassung der durchgeführten Optimierung dienen, um so ein adaptives Verhalten zu ermöglichen.
Aus einer logistischen und damit ähnlichen Perspektive betrachtet die Arbeit von Ruderman das Problem [82]. Hierbei werden Beladungen mit Hilfe von Stabilitätsmerkmalen beschrieben und diese zum Anlernen eines Modells genutzt. Dieses soll anhand der Merkmale entscheiden können, ob und wie stabil eine Beladung ist. Der Unterschied zu der vorliegenden Arbeit ist damit der Fokus, der auf der Bewertung der Beladung und nicht auf der Berechnung ebendieser liegt.

Lernen von Experten

Eine Möglichkeit, das Konzept des HETM technisch umzusetzen und welches in dieser Arbeit betrachtet wird, liefert die Kombination aus Reinforcement und Imitation Learning. Diese ermöglicht das Lernen von Strategien auf Basis von Expertendemonstrationen, die beispielsweise in Form von Daten oder interaktivem Feedback zur Verfügung stehen.
Bisherige, erfolgreiche Implementierungen befassen sich unter anderem mit dem Anlernen motorischer Aufgaben in der Robotik [36][108][73], dem Meistern von Video- und Brettspielen [86][80], als auch ersten Erfolgen im autonomen Fahren [104][98][78] und Fliegen [3][81].
Anwendungen in der Logistik, insbesondere in der Verpackung, fehlen dagegen, was den Nutzen dieser Arbeit unterstreicht.

1.5 Ziel der Arbeit

Das Ziel der vorliegenden Arbeit ist es einen Ansatz zu entwickeln, der ein durch Erfahrungswissen beeinflusstes Verhalten nachbilden kann. Insbesondere sollen dabei die Adaptivität und Variabilität in Bezug auf die Durchführung eines Prozesses erfasst werden. Als Anwendungsfall soll dabei die logistische Verpackung dienen. Gesammelte Daten über Packvorgänge sollen mit Hilfe von Techniken des maschinellen Lernens, insbesondere des Reinforcement und Imitation Learnings, verarbeitet werden. Die dabei resultierenden Modelle sollen auf Grundlage ihrer Leistung in Bezug auf das demonstrierte Verhalten untersucht werden. Hierzu werden praxisnahe Szenarien nachgebildet, auf Basis derer eine Evaluierung durchführt werden kann.
Der entwickelte Ansatz soll dabei als Grundlage für einen digitalen Assistenten in dem Bereich verwendet werden können, um so beispielsweise neue Mitarbeiter mit Strategien erfahrenen Personals anlernen zu können.

1.6 Struktur der Arbeit

Dieses Kapitel sollte einen Einblick in den Hintergrund und die Motivation dieser Arbeit geben. Insbesondere sollte dabei die praktische Relevanz der Aufgabenstellung in Hinblick auf die Logistik 4.0 hervorgehen.
In Kapitel 2 werden relevante Grundlagen des maschinellen Lernens definiert und ihre Anwendbarkeit auf die Problemstellung diskutiert.
In Kapitel 3 wird die Implementierung eines Frameworks zur Lösung des in dieser Arbeit betrachteten Problems beschrieben.
Das Kapitel 4 beschäftigt sich mit der Evaluierung des im vorherigen Kapitel implementieren Frameworks. Hierzu werden die verwendeten Methoden anhand unterschiedlicher Packstrategien evaluiert und hinsichtlich ihrer Performance und Adaptivität untersucht.

Hierbei sollen Daten verschiedener Packstrategien als Grundlage dienen.
In Kapitel 5 folgt ein Resümee des Erarbeiteten und ein Ausblick auf weiteres Vorgehen.

Kapitel 2

Reinforcement Learning

In diesem Kapitel werden grundlegende Ideen des maschinellen Lernens definiert. Dazu betrachten wir zunächst in Abschnitt 2.1 die Problemstellung des Reinforcement Learnings, welches als Grundlage zur Lösung des BPP dienen soll. Darauf aufbauend werden in Abschnitt 2.2 Ideen zur Lösung der im Reinforcement Learning gestellten Problemstellung präsentiert. Im darauffolgenden Abschnitt werden die Grundlagen neuronaler Netze definiert, die eine wichtige Klasse von Modellen im Bereich des Reinforcement Learnings darstellen. Anschließend widmen wir uns in Abschnitt 2.4 dem Imitation Learning, mit dessen Hilfe wir Erfahrungen eines Experten integrieren wollen. Die in diesem Kapitel vorgestellten Ideen sollen zu einem möglichen Lösungsansatz für die betrachtete Problemstellung beitragen. Ihre Anwendbarkeit wird daher in Abschnitt 2.5 diskutiert.

2.1 Grundbegriffe

Das grundlegende Ziel der Arbeit ist es, Wissen aus Daten zu schöpfen. Als technisches Werkzeug bieten sich dafür Ansätze des maschinellen Lernens an. Hierbei dienen Daten als Grundlage zur automatischen Generierung künstlichen Wissens, welches in Form von statistischen Modellen repräsentiert wird.
Eine Methode des maschinellen Lernens stellt dabei das **Reinforcement Learning** (RL) dar. Hierbei ist das Ziel das Erlernen von profitmaximierenden Strategien auf Grundlage eigenständigen Erkundens. Das Szenario lässt sich wie folgt beschreiben:
Im Mittelpunkt steht der sogenannte **Agent**. Hierbei handelt es sich um einen Akteur, der eine Problemstellung zu lösen hat. Diese wird durch das sogenannte **Environment** repräsentiert. Zu jedem Zeitpunkt befindet sich der Agent in einem Zustand (engl. *State*) des Environments. Aus diesem heraus kann der Agent eigenständig Aktionen durchführen und bekommt dafür Feedback. Dieses besteht zum einen aus dem resultierenden Folgezustand, zum anderen aus einer Belohnung (engl. *Reward*). War die durchgeführte Aktion zielführend, fällt diese dementsprechend gut aus. Ist die Aufgabe nach Erhalt des Feedbacks noch

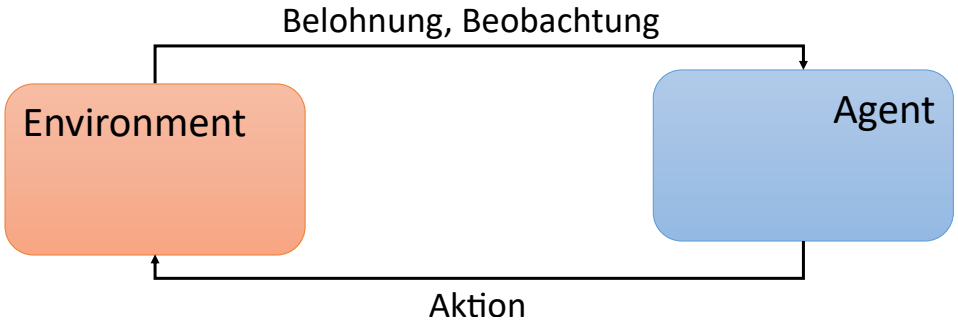


Abbildung 2.1: Szenario des Reinforcement Learnings [90]: Der Agent führt in einem iterativen Prozess Aktionen aus. Diese führen zu Feedback seitens des Environments.

nicht gelöst, startet ein weiterer Durchlauf. Der Ablauf wird in Abb. 2.1 visualisiert. Das Ziel des Agenten ist es dabei, die Problemstellung mit maximaler kumulierter Belohnung zu lösen.

Das Szenario des RL wird im Folgenden formalisiert. Die hier verwendeten Definitionen und Notationen basieren auf dem Buch von Sutton und Barto [90].

Die Problemstellung lässt sich in Form eines **Markov-Entscheidungsprozesses** (MEP) modellieren. Dieser ist wie folgt definiert:

Definition 2.1: Markov-Entscheidungsprozess

Ein **Markov-Entscheidungsprozess** $\mathcal{M}$ ist ein stochastischer Prozess, der mit Hilfe des Vier-Tupels $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R})$ beschrieben werden kann:

- Die Menge der **Zustände** $\mathcal{S}$ von $\mathcal{M}$
- Die Menge der **Aktionen** $\mathcal{A}$ von $\mathcal{M}$, wobei die Teilmengen $A_s \subseteq \mathcal{A}$ die in Zustand $s \in \mathcal{S}$ möglichen Aktionen darstellen
- Die **Transitionsfunktion** $\mathcal{P} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \mapsto [0, 1]$ von $\mathcal{M}$ gibt die Wahrscheinlichkeit an, mit der man von einem Zustand $s \in \mathcal{S}$ in einen Zustand $s' \in \mathcal{S}$ durch Ausführen einer Aktion $a \in \mathcal{A}_s$ gelangt.
- Die **Belohnungsfunktion** (oder **Reward-Funktion**) $\mathcal{R} : \mathcal{S} \times \mathcal{A} \mapsto \mathbb{R}$ von $\mathcal{M}$ weist einer Transition ausgehend eines Zustands $s \in \mathcal{S}$ durch eine Aktion $a \in \mathcal{A}_s$ einen Wert $\mathcal{R}(s, a) \in \mathbb{R}$ zu.

Die Abbildung 2.2 zeigt eine Visualisierung. Der Durchlauf eines MEP erfolgt dabei in diskreten Zeitschritten. Zum Zeitpunkt $t \in \mathbb{N}$ befinden wir uns in einem Zustand $s_t \in \mathcal{S}$ aus dem der Agent die Aktion $a_t \in \mathcal{A}_s$ ausführt. Diese wird vom Environment mit der Belohnung $\mathcal{R}(s_t, a_t) = r_t$ bewertet und versetzt dieses mit einer Wahrscheinlichkeit von

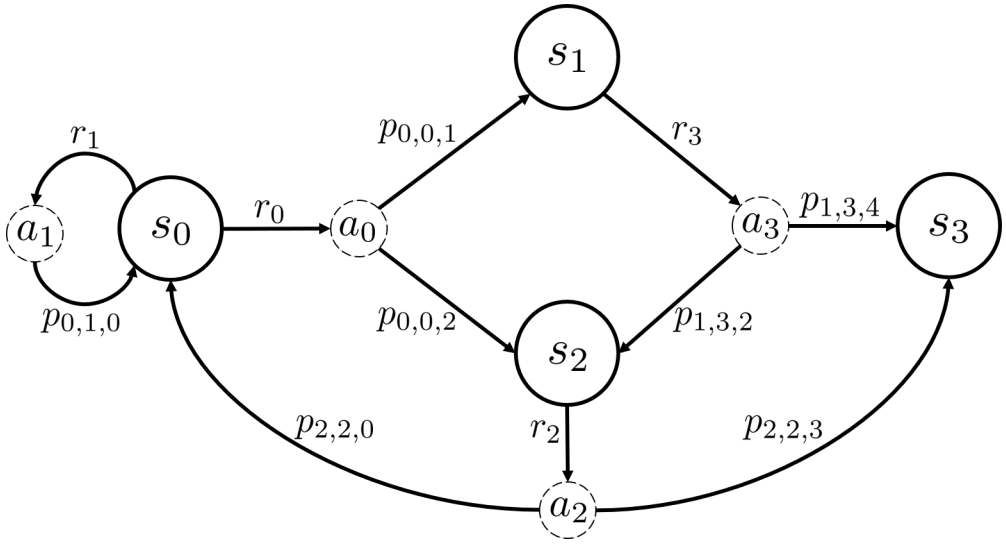


Abbildung 2.2: MEP Beispiel: Die Abbildung zeigt, wie sich ein MEP als gerichteter Graph darstellen lässt. Die Zustände eines MEPs stellen dabei die Knoten dar. Die Aktionen, die die Zustände miteinander verbinden, die Kanten. Besonderheiten ergeben sich dabei zum einen über den Reward r, der je Transition vergeben wird, als auch über die Transitionswahrscheinlichkeiten p, wodurch sich ein probabilistisches Verhalten modellieren lässt. In der Abbildung wurde der Übersicht halber die kompaktere Schreibweise $\mathcal{P}(s_x, a_y, s_z) = p_{x,y,z}$ gewählt.

$\mathcal{P}(s_t, a_t, s')$ in einen Zustand $s' \in \mathcal{S}$. Gelangen wir dabei zu einem Zeitpunkt T in einen Zustand $s_T \in \mathcal{S}$, in dem keine Aktionen mehr möglich sind, also $A_{s_T} = \emptyset$ gilt, gilt der Prozess als terminiert. Folgende Begriffe sind dabei von Relevanz:

Definition 2.2: Trajektorie, Episode, Ertrag

Eine Sequenz aus Interaktionen zwischen Agent und Environment in der Form

$$\tau =< s_t, a_t, s_{t+1}, a_{t+1}, ... >$$

wird als **Trajektorie** τ bezeichnet. Beginnt diese in einem Startzustand s_0 und endet in einem Terminalzustand s_T:

$$\tau =< s_0, a_0, s_1, a_1, ..., a_{T-1}, s_T >$$

und absolviert somit einen gesamten Durchlauf, sprechen wir von einer **Episode** des MEPs. Die gesammelten Belohnungen einer Trajektorie

$$G_t = r_t + r_{t+1} + ... = \sum_{i=0}^{\infty} r_{i+t}$$

werden als **Ertrag** G_t bezeichnet.

Die Strategie, die ein Agent beim Bearbeiten der Problemstellung anwendet, lässt sich mit Hilfe seiner Policy darstellen. Diese ist wie folgt definiert:

Definition 2.3: Policy

Eine **Policy** $\pi \in \Pi$ ist definiert als bedingte Wahrscheinlichkeit über die möglichen Aktionen $a \in \mathcal{A}_s$ für einen gegebenen Zustand $s \in \mathcal{S}$.

Die Policy des Agenten sollte dabei das Ziel verfolgen, den Ertrag einer Episode zu maximieren, den entsprechenden Aktionen also eine hohe Wahrscheinlichkeit zuordnen. Hierbei ist die Herausforderung, nicht nur die nächste Aktion möglichst gewinnbringend zu planen, sondern auch nachfolgende Zustände und Belohnungen zu berücksichtigen. Es sollte also über einen weitreichenden Horizont geplant werden um keine kurzfristige, sondern eine kumulierte Maximierung der Belohnung ermöglichen zu können.
Es bedarf dafür einer Priorisierung der Belohnungen in Bezug auf die zeitliche Nähe.

Definition 2.4: Diskontierter Ertrag, Diskontierungsfaktor

Der **diskontierte Ertrag** $G_{t,\gamma}$ ergibt sich aus

$$G_{t,\gamma} = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + ... = \sum_{i=0}^{\infty} \gamma^i r_{i+t}$$

wobei $\gamma \in [0, 1]$ den **Diskontierungsfaktor** (engl. *Discountrate*) darstellt.

Wählen wir also für γ einen Wert nahe 1, gewichten wir zukünftige Rewards mehr, wodurch wir den Agenten zu vorausschauender Planung bewegen wollen. Ein Wert nahe 0 würde ihn dagegen dazu bringen, die situationsbedingt lukrativste Entscheidung zu treffen.
Um Policies miteinander vergleichen zu können, bedarf es eines Maßes für die Performance einer solchen. Dieses lässt sich über folgende Werte definieren:

Definition 2.5: Zustandswert, Aktionswert

Der **Zustandswert** $V_\pi(s)$ eines Zustands $s \in \mathcal{S}$ bezüglich einer Policy $\pi \in \Pi$ entspricht dem diskontierten Ertrag, der aus dem Zustand zu erwarten ist:

$$V_\pi(s) = \mathbb{E}[G_{t,\gamma}|s_t = s] = \mathbb{E}[\sum_{i=0}^{\infty} \gamma^i r_{i+t+1}|s_t = s]$$

Die Funktion V_π bezeichnen wir als **Wertefunktion** (engl. *value function*).
Der **Aktionswert** $Q_\pi(s, a)$ einer Aktion $a \in \mathcal{A}_s$ in einem Zustand $s \in \mathcal{S}$ bezüglich einer Policy $\pi \in \Pi$ entspricht analog dem diskontierten Ertrag, der aus der ausgeführten Aktion zu erwarten ist:

$$Q_\pi(s, a) = \mathbb{E}[G_{t,\gamma}|a_t = a, s_t = s] = \mathbb{E}[\sum_{i=0}^{\infty} \gamma^i r_{i+t+1}|a_t = a, s_t = s]$$

Die Funktion Q_π bezeichnen wir als **Aktionswertefunktion** oder **Q-Funktion** (engl. *action-value function* oder *Q-value function*).

Die Wertefunktion gibt also anhand der zu *erwartenden* Belohnungen an, wie *gut* ein Zustand des MEP unter einer bestimmten Policy ist. Die Aktionswertefunktion betrachtet analog dazu die Aktionen, die innerhalb des Zustands durchgeführt werden können. Der Aktionswert entspricht dabei der Belohnung für die betrachtete Aktion zuzüglich der erwarteten Belohnung des nächsten Zustands. Für Terminalzustände entsprechen dabei beide Werte 0.

2.2 Lösungsansätze

Das Ziel des Reinforcement Learnings ist das Erlernen einer Policy, die die durch den MEP definierte Problemstellung mit maximaler Belohnung lösen kann. Auf Basis der im vorherigen Abschnitt definierten Wertefunktionen, lässt sich die gesuchte Policy wie folgt definieren:

Definition 2.6: Optimale Policy

Für zwei Policies $\pi, \pi' \in \Pi$ gilt:

$$\pi \geq \pi' \Leftrightarrow \forall s \in \mathcal{S} : V_\pi(s) \geq V_{\pi'}(s)$$

Das Supremum, der aus der Relation resultierenden Halbordnung, wird **optimale Policy** π_* genannt, mit Wertefunktion

$$V_*(s) = \max_\pi V_\pi(s)$$

und Aktionswertefunktion:

$$Q_*(s,a) = \max_\pi Q_\pi(s,a)$$

Es sei dabei zu erwähnen, dass die optimale Policy nicht zwingend eindeutig sein muss und es daher mehrere Policies in Π geben kann, die die genannte Bedingung erfüllen.
Zur Ermittlung einer optimalen Policy spielen die Wertefunktionen eine entscheidende Rolle. Eine Optimierung dieser entspricht einer Optimierung der Policy. Eine entscheidende Beobachtung dafür liefert dabei folgende Umrechnung:

$$\begin{aligned}
V_\pi(s) &= \mathbb{E}[G_{t,\gamma}|s_t = s] \\
&= \mathbb{E}[\sum_{i=0}^{\infty} \gamma^i r_{i+t+1}|s_t = s] \\
&= \mathbb{E}[r_{t+1} + \sum_{i=1}^{\infty} \gamma^i r_{i+t+1}|s_t = s] \\
&= \mathbb{E}[r_{t+1} + \gamma G_{t+1}|s_t = s] \\
&= \sum_{a\in\mathcal{A}_s} \pi(s,a) \sum_{s'\in\mathcal{S}} \mathcal{P}(s,a,s')(\mathcal{R}(s',a) + \gamma\mathbb{E}[G_{t+1}|s_{t+1} = s']) \\
&= \sum_{a\in\mathcal{A}_s} \pi(s,a) \sum_{s'\in\mathcal{S}} \mathcal{P}(s,a,s')(\mathcal{R}(s',a) + \gamma V_\pi(s')) \qquad (2.1)
\end{aligned}$$

Die Gleichung 2.1 entspricht der **Bellman-Gleichung**. Sie verdeutlicht durch ihre rekursive Formulierung insbesondere die Beziehung eines Zustands zu seinen Nachfolgezuständen. Darauf aufbauend lässt sich eine optimale Wertefunktion *unabhängig* einer bestimmten Policy definieren:

Definition 2.7: Optimale Wertefunktion

Für einen MEP $\mathcal{M} =(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R})$ ergibt sich aus 2.1 die **optimale Wertefunktion** V_*:

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

Analog ergibt sich die **optimale Aktionswertefunktion** Q_*:

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

Die optimalen Wertefunktionen sind dabei gültig für alle optimalen Policies. Eine solche Policy agiert dabei *greedy* in Bezug auf V_*. Das heißt, sie wählt von einem Zustand immer den Folgezustand aus, der laut V_* am lukrativsten erscheint. Ein Ansatz zur Lösung eines MEP ist daher die Berechnung von V_* um daraus eine optimale Policy π_* abzuleiten.
Aus der optimalen Wertefunktion ergibt sich für jeden Zustand des MEP eine Gleichung. Haben wir n Zustände, entspricht dies also n Gleichungen mit n Unbekannten. Aus der Lösung des daraus resultierenden nicht-linearen Gleichungssystems ergibt sich die optimale Wertefunktion und damit eine optimalen Policy π_*.
Insbesondere für MEPs mit vielen Zuständen stellt dies jedoch aufgrund des hohen Rechenaufwands keine praktikable Lösung dar. Methoden des Reinforcement Learnings zielen daher darauf ab, die optimalen Wertefunktionen *empirisch* zu approximieren, anstatt sie exakt zu errechnen. Ein Agent, der durch ein statisches Modell implementiert wird, interagiert dafür mit einem Environment. Die dabei entstehenden Beobachtungen dienen als Datengrundlage zur Anpassung der Modellparameter und der damit repräsentierten Policy des Agenten. Das Ziel ist durch genügend eigenständige Interaktion mit dem Environment eine Policy zu approximieren, die einer optimalen Policy möglichst nahe kommt.

2.3 Neuronale Netze

Im Bereich des maschinellen Lernens dienen trainierbare Modelle dazu, das gewonnene Wissen zu repräsentieren und spielen damit eine entscheidende Rolle. So wird auch im Reinforcement Learning der Agent durch ein solches Modell repräsentiert. Neuronale Netze stellen eine wichtige Klasse von Modellen dar, die insbesondere in den vergangenen Jahren zunehmend an Bedeutung gewonnen haben. Viele moderne Algorithmen im Bereich des Reinforcement Learnings bauen auf diesen auf, weshalb wir diese im Folgenden näher betrachten. Die Definitionen und Notationen entstammen dem Buch von Niemann [72].

2.3.1 Künstliches Neuron

Ein *künstliches* Neuron bildet die Basiskomponente eines künstlichen neuronalen Netzes. Dieses ist durch sein biologisches Pendant motiviert, welches eine elementare Rolle in der Informationsverarbeitung des Menschen spielt. Das künstliche Neuron lässt sich dabei wie folgt definieren:

Definition 2.8: Künstliches Neuron

Ein **künstliches Neuron** ist definiert über seine Gewichte $w \in \mathbb{R}^n$, seinen Bias $b \in \mathbb{R}$ und seine Aktivierungsfunktion $\varphi : \mathbb{R} \mapsto \mathbb{R}$.

Der Aufbau eines solchen Neurons wird in Abb. 2.3 visualisiert. Die Auswertung eines Inputs $x \in \mathbb{R}^n$ erfolgt dabei wie folgt: Der Input wird mit Hilfe der Werte $w \in \mathbb{R}^n$ gewichtet und kumuliert. Das Ergebnis wird anschließend mit Hilfe der Aktivierungsfunktion φ ausgewertet, heißt:

$$f = \varphi(y) = \varphi(\langle x, w \rangle + b) = \varphi[\sum_{i=1}^{n}(x_i \cdot w_i) + b]$$

Die Funktion φ hat dabei einen Schwellwertcharakter. Das heißt, dass diese, falls y einen gewissen Schwellwert überschreitet, einen positiven Wert ausgibt. Man spricht dann davon, dass das Neuron *feuert*. Beispiele zu Aktivierungsfunktionen sind in Abb. 2.4 abgebildet. Der Bias dient dabei dazu, den Schwellwert variieren zu können, ohne die Schwellwertfunktion selbst modifizieren zu müssen. Dies erleichtert insbesondere die Implementierung derartiger Modelle.
Ein Neuron stellt durch seine Formulierung eine Hyperebene dar und ermöglicht damit eine lineare Trennung des betrachteten Raums.

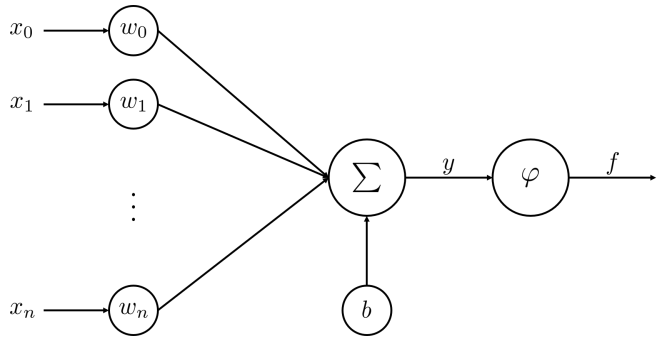
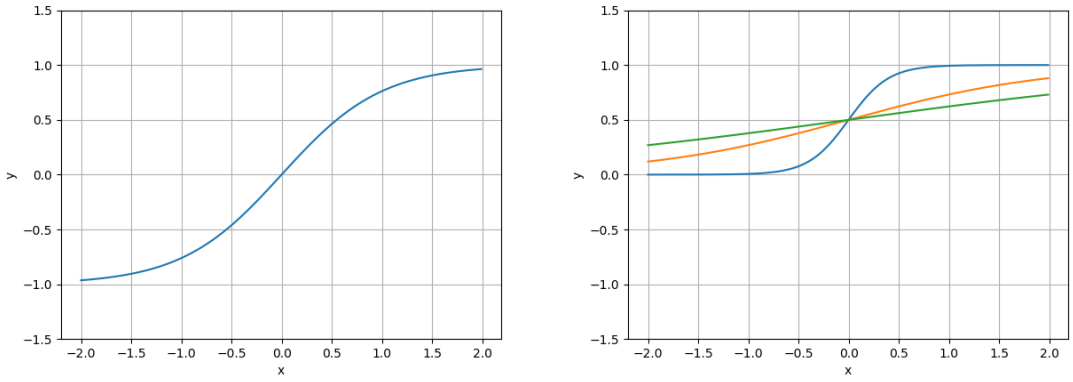
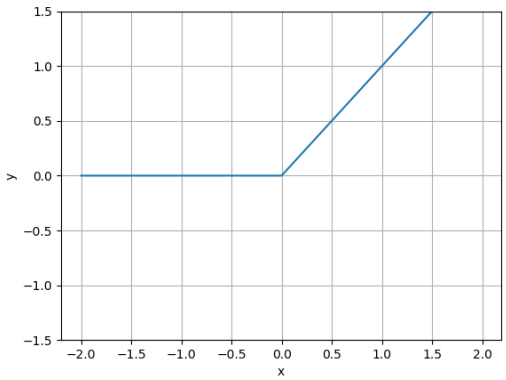


Abbildung 2.3: Aufbau eines künstlichen Neurons [35]: Die Eingabe x wird mit Hilfe w gewichtet und anschließend summiert. Anschließend wird die gewichtete Summe durch die Aktivierungsfunktion ausgewertet, was dem Output des Neurons entspricht.

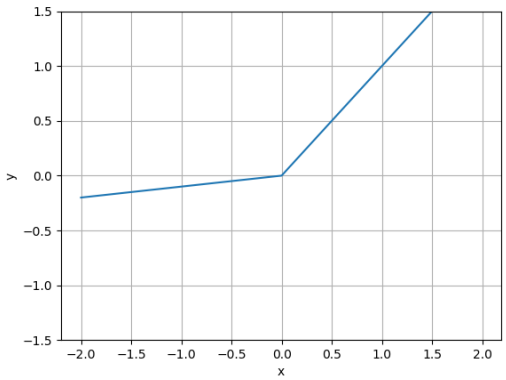


(a) $\tanh(x) = \frac{e^y - e^{-y}}{e^y + e^{-y}}$

(b) $sig_\alpha(x) = \frac{1}{1+e^{-\alpha x}}$



(c) $relu(x) = \max(0, x)$



(d) $leakyrelu(x) = \max(0.1x, x)$

Abbildung 2.4: Beispiele verbreiteter Aktivierungsfunktionen: (a) Tangens hyperbolicus, (b) Sigmoid für $\alpha = 5$ (blau), $\alpha = 1$ (orange), $\alpha = 0.5$ (grün), (c) Rectified Linear Unit [71], (d) Leaky Rectified Linear Unit [43].

2.3.2 Multi Layer Perceptron

Um deren Mächtigkeit zu erhöhen, lassen sich künstliche Neuronen miteinander verbinden. Dazu werden diese in Schichten angeordnet und die Schichten fortlaufend miteinander verbunden. Man spricht hierbei von einem **Multi Layer Perceptron** (MLP). Ein solches MLP besteht aus $L + 1$ Schichten, wobei jede Schicht aus M^l Knoten besteht für $l \in \{0, ..., L\}$. Ein derartiger Aufbau wird in Abb. 2.5 visualisiert. Wollen wir einen Input $x \in \mathbb{R}^n$ auswerten, so wird dieser zunächst über die Eingabeschicht mit Index $l = 0$ aufgenommen und anschließend von Schicht zu Schicht durchpropagiert, bis das Ergebnis schließlich über die Ausgabeschicht mit Index $l = L$ ausgegeben wird. Die Neuronen zweier aufeinander folgender Schichten sind dabei so miteinander verbunden, dass der Output

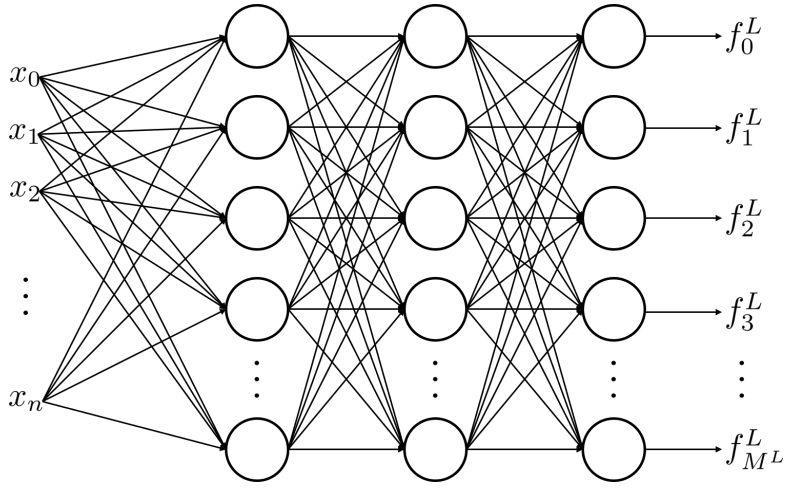


Abbildung 2.5: Multi Layer Perceptron: Neuronen werden in Schichten anordnet und diese anschließend miteinander verbunden. Der Output einer Schicht entspricht dabei dem Input der darauffolgenden Schicht.

eines Neurons i in Schicht l zum Input des Neurons j in Schicht $l+1$ mit Gewicht w_{ij}^{l+1} beiträgt. Der Output eines Neurons j in Schicht $l+1$ lässt sich damit wie folgt beschreiben:

$$f_j^{l+1} = \varphi(y_j^{l+1}) = \varphi\left(\sum_{i=0}^{M^l} w_{ij}^{l+1} f_i^l\right), \forall j \in \{1, ..., M^{l+1}\} \text{ und } \forall l \in \{0, ..., L-1\}$$

Eine vereinfachte Schreibweise ergibt sich dabei aus der Anordnung der Gewichte der l-ten Schicht in einer Matrix:

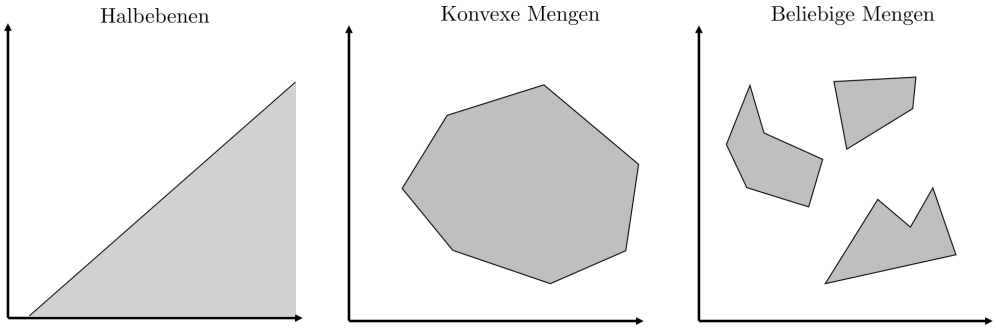
$$W^l = \begin{bmatrix} w_{01}^l & w_{11}^l & \cdots & w_{M^{l-1}1}^l \\ w_{02}^l & w_{12}^l & \cdots & w_{M^{l-1}2}^l \\ \vdots & \vdots & \ddots & \vdots \\ w_{0M^l}^l & w_{1M^l}^l & \cdots & w_{M^{l-1}M^l}^l \end{bmatrix}$$

Daraus ergibt sich für den Output einer Schicht $l+1$ folgende Ausgabe:

$$f^{l+1} = \varphi(W^{l+1} f^l), \forall l \in \{0, ..., L-1\}$$

Der Output des Netzes entspricht dabei der Ausgabe der letzten Schicht, also $f_L \in \mathbb{R}^{M^L}$. Da der Input von vorne nach hinten durch das Netz verarbeitet wird, spricht man bei der Auswertung auch von *Forward Propagation*.

Wie bereits im vorherigen Abschnitt angemerkt, ist ein einzelnes Neuron in der Lage den Raum linear durch eine Hyperebene zu trennen. Eine Schicht von Neuronen stellt dementsprechend eine Menge an Hyperebenen dar. Fügen wir eine weitere Schicht hinzu, die die durch diese Hyperebenen definierten Mengen schneidet, erhalten wir als Resultat eine konvexe Menge. Erweitern wir dieses Konstrukt zudem um eine weitere Schicht, die diese konvexen Mengen vereinigt, so können wir bereits mit einem Netz, bestehend aus drei



(a) Eine Schicht **(b)** Zwei Schichten **(c)** Drei Schichten

Abbildung 2.6: Multi Layer Perceptron (MLP) Mächtigkeit in 2D [35]: (a) Ein Neuron kann stellt eine lineare Trennung des Raums dar (b) Werden Hyperebenen geschnitten, ergibt dies konvexe Mengen (c) die Vereinigung konvexer Mengen ermöglicht die Darstellung beliebiger Mengen

Schichten, *beliebige* Mengen abbilden. Die Abb. 2.6 visualisiert dies im Zweidimensionalen. Aufbauend auf dieser Beobachtung lässt sich zeigen, dass ein neuronales Netz mit bereits drei Schichten beliebige Funktionen approximieren kann [49].

2.3.3 Training

Um eine Funktion approximieren zu können, muss ein neuronales Netz an diese angepasst werden. Man spricht hierbei vom *Training* des Netzes. Bei den dabei anzupassenden Parametern handelt es sich um die Gewichte. Die Idee ist hierbei diese in einem iterativen Prozess so anzupassen, sodass der Fehler zwischen der Ausgabe des Netzes und des gewünschten Ergebnisses minimiert wird.
Die Gewichte sollen durch Gradientenabstieg in die Richtung des minimalen Fehlers trainiert werden. Als Updateregel für die Gewichte ergibt sich damit:

$$w_{ij}^l \leftarrow w_{ij}^l - \beta \frac{\partial \mathcal{L}}{\partial w_{ij}^l}.$$

Der Gradient $\frac{\partial \mathcal{L}}{\partial w_{ij}^l}$ beschreibt dabei den Anteil, den das Gewicht w_{ij}^l in Bezug auf den Fehler $\mathcal{L}$ hat. Der Parameter β stellt die Lernrate dar, die empirisch während des Trainings festzulegen ist.
Um den Gradienten bestimmen zu können, können wir dessen Ausdruck mit Hilfe der Kettenregel umformulieren:

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial w_{ij}^l} &= \frac{\partial \mathcal{L}}{\partial f_j^l} \cdot \frac{\partial f_j^l}{\partial w_{ij}^l} \\ &= \underbrace{\frac{\partial \mathcal{L}}{\partial f_j^l}}_{(1)} \cdot \underbrace{\frac{\partial f_j^l}{\partial y_j^l}}_{(2)} \cdot \underbrace{\frac{\partial y_j^l}{\partial w_{ij}^l}}_{(3)}\end{aligned}$$

Je nach Wahl der Aktivierungsfunktion, lassen sich die Terme (2) und (3) durch Differenzieren errechnen. Besondere Aufmerksamkeit gilt dabei dem Term (1). Hier wollen wir den Anteil der Ausgabe eines Neurons f_j^l am gesamten Fehler $\mathcal{L}$ ermitteln. Für die Ausgaben der letzten Schicht kann dies durch direktes Differenzieren errechnet werden, da gerade diese direkten Einfluss auf den Fehler haben. Die Schwierigkeit entsteht allerdings in den inneren Schichten des Netzes. Da diese nur indirekt in den Fehler miteinwirken, bedarf es zusätzlicher Überlegungen, um deren Anteil zu ermitteln. Hierbei hilft die verallgemeinerte Kettenregel:

$$\frac{\partial \mathcal{L}}{\partial f_j^l} = \sum_{k=1}^{M^{l+1}} \frac{\partial \mathcal{L}}{\partial f_k^{l+1}} \cdot \frac{\partial f_k^{l+1}}{\partial f_j^l}$$

Daraus folgt, dass wir den Fehler einer Ausgabe f_j^l in Abhängigkeit der Fehler der nächsten Schicht $l+1$ angeben können. Der Fehler wird damit von der letzten Schicht in Richtung der ersten Schicht durchgereicht. Man spricht daher von der *Back Propagation* [83].

2.3.4 Deep Learning

Ein Problem, das neuronale Netze mit sich bringen, ist, dass die Anzahl interner Parameter mit zunehmender Inputgröße wachst. Das heißt, dass falls wir hochdimensionale Eingaben verarbeiten wollen, wir konsequenterweise eine proportional höhere Parameterzahl in Kauf nehmen müssen. Eine hohe Parameterzahl kann allerdings zu führen, dass ein Netz schlecht auf neuen Daten generalisiert. Man spricht hierbei von *Overfitting*.
Dies tritt insbesondere bei Bildern auf, die bereits bei geringen Auflösungen eine für ein neuronales Netz hohe Anzahl von Pixeln aufweisen. Es ist daher üblich, derartige Eingaben vor der eigentlichen Verarbeitung im Modell anzupassen. Man spricht hierbei von *Merkmalsextraktion*. Dabei ist das Ziel, Eingaben auf die relevanten Bereiche zu reduzieren, um so insbesondere die Dimension zu verkleinern.
Eine Möglichkeit, eine solche Merkmalsextraktion im Zusammenhang mit neuronalen Netzen zu realisieren, bietet das *Deep Learning*. Hierbei betrachtet man spezielle Architekturen, die MLPs um zusätzliche Schichten erweitern, durch welche Eingaben vorverarbeitet werden sollen. Ein Vorteil solcher Netze gegenüber anderer Verfahren zur Merkmalsextraktion ist, dass diese zusammen mit dem MLP anhand der Daten trainiert werden können. Dies heißt insbesondere, dass zwar die Wahl einer passenden Architektur vom Entwickler zu fällen ist, die dazugehörigen Parameter sich jedoch aus dem Training ergeben.
Ein hierfür verbreitetes Modell stellen *Faltungsnetze* dar [39][57]. Hierbei handelt es sich um neuronale Netze, die um sogenannte Faltungsschichten erweitert werden, die den Input vor Eingang ins MLP verarbeiten.
Die Idee stammt dabei aus der digitalen Bildverarbeitung: ein Eingabebild $I \in \mathbb{R}^{h \times w \times c}$

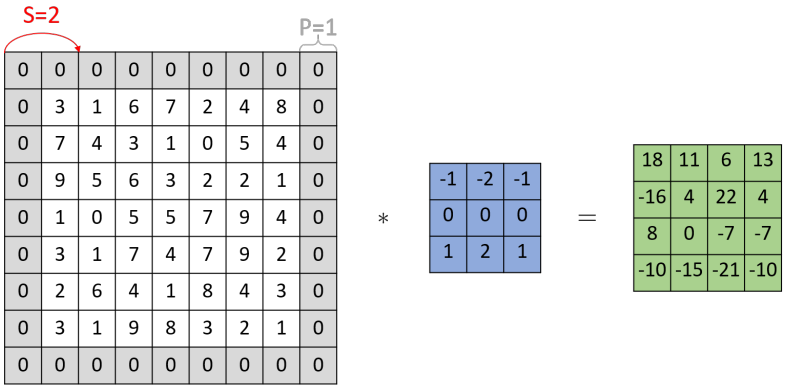


Abbildung 2.7: Beispiel für die Faltung: Das Beispiel zeigt einen $9 \times 9 \times 1$ Input, der mit einem $3 \times 3 \times 1$ Filter gefaltet werden soll. Bei dem hier dargestellten Filter handelt es sich um den Sobel-Filter, der in der Bildverarbeitung insbesondere in der Kantenerkennung verwendet wird [87]. Es soll eine Stride $S = 2$ und ein Padding $P = 1$ verwendet werden. Es ergibt sich dadurch ein Aktivierungsfeld der Größe $4 \times 4 \times 1$.

wird mit Hilfe eines *Filters* gefaltet. Dieser kann durch eine Matrix $W \in \mathbb{R}^{n \times n \times c}$ dargestellt werden, dessen Dimension deutlich geringer als die des Bildes ist. Die (diskrete) Faltung lässt sich wie folgt formulieren:

$$(I * W)(x, y) = F(x, y) = \sum_{c=1}^{K} \sum_{i=1}^{n} \sum_{j=1}^{n} I(x + i, y + j, c) W(i, j, c)$$

Der Filter wird über das Bild „geschoben“, sodass nach jeder Verschiebung ein anderer Bildausschnitt betrachtet wird, die Werte des Filters selbst bleiben dabei gleich. Daraus ergeben sich für jede Position (x, y) des Filters Ausgabewerte, die in einer Matrix F festgehalten werden. Diese Matrix wird *Aktivierungsfeld* genannt. Wie genau ein Filter über die Eingabe geht, wird mittels zwei Parameter eingestellt: Stride S und Padding P. Die Stride gibt dabei an, wie groß wie Sprünge zwischen zwei Auswertungen sind. Das Padding dient dazu, die Eingabe um einen Rand aus Nullwerten zu erweitern. Ein Beispiel im Zweidimensionalen zeigt die Abb. 2.7. Die neuen Dimension w' und h' des resultierenden Aktivierungsfeldes ergibt sich damit aus:

$$h' = \frac{h - n + 2P}{S},$$
$$\text{bzw. } w' = \frac{w - n + 2P}{S}$$

Eine Faltungsschicht verfügt dabei für über m Aktivierungsschichten, die sich aus m verschiedenen Filtern ergeben. Anders als es beispielsweise in der Bildverarbeitung üblich ist, werden die Werte in den Filtern nicht von Hand ermitteln, sondern ergeben sich direkt aus dem Training mit den Daten. Da eine Aktivierungsschicht aus lediglich einem Filter entsteht, sich also die Gewichte teilt, bleibt die Anzahl der freien Parameter in Faltungsschichten im Vergleich zu vollvernetzten Schichten eines MLPs relativ gering, wodurch

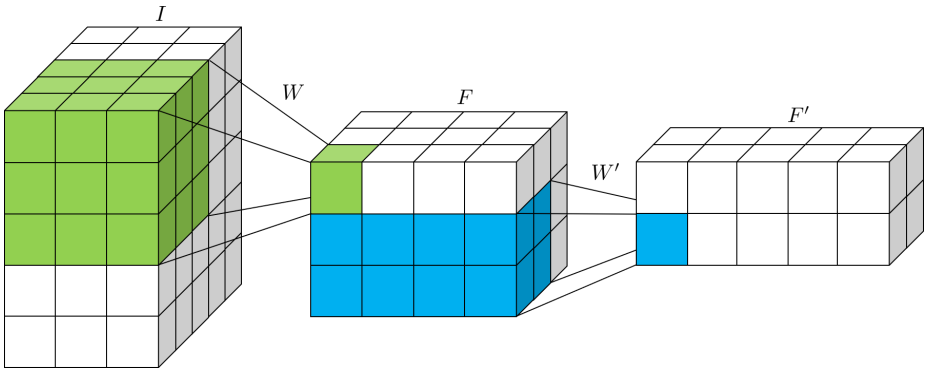


Abbildung 2.8: Aufbau von Faltungsschichten [35]: Die Abbildung visualisiert wie Faltungssschichten in einem Faltungsnetz aufgebaut sein können. Hier soll der Input I vorverarbeitet werden. Der Filter, der sich aus den Gewichten W ergibt, läuft dabei über die gesamte Eingabe und betrachtet Fenster fixer Größe. Die Auswertung eines Fensters ergibt einen Wert in einer Aktivierungsschicht. In diesem Beispiel haben wir ein Padding von 0 und eine Stide von 1. Daraus ergeben sich für den Input mit Dimensionen 5×5 und einem 3×3 Filter eine Aktivierungsschicht der Dimension 3×3. Wenden wir vier verschiedene Filter an, so ergeben sich dementsprechend vier Aktivierungsfelder in der Faltungsschicht. Der dabei entstandene Output wird als Input für die darauffolgende Schicht verwendet.

Overfitting eingedämmt werden kann.
Ähnlich wie auch schon in einem MLP, werden auch Faltungsschichten in sequentieller Reihenfolge miteinander verbunden, wobei der Output einer Schicht, den Input der darauffolgenden Schicht darstellt (siehe Abb. 2.8). Ein Unterschied liegt dabei insbesondere in der Tiefe dieser Vernetzungen: während bei MLPs bereits drei Schichten genügen, um beliebige Funktionen zu approximieren, so können bei Faltungsschichten deutlich mehr Schichten zum Einsatz kommen.
Der Einsatz von neuronalen Netzen hat zu jüngsten Erfolgen in Bereichen der Bildverarbeitung geführt. Insbesondere Faltungsnetze zeigen dabei bemerkenswerte Resultate und gewannen dadurch zunehmend an Bedeutung [106][40][50]. Dies beeinflusste auch das Reinforcement Learning. So lassen sich unter anderem Funktionen wie V_* oder Q_* mittels tiefer neuronaler Netze approximieren und zeigten dabei bemerkenswerte Resultate in Bereichen wie der Robotik oder Brett- und Videospielen. Derartige Ansätze werden unter dem Begriff *Deep Reinforcement Learning* zusammengefasst [56]. Ein konkretes Verfahren dazu soll in Kapitel 3.3 vorgestellt werden.

2.4 Imitation Learning

Anders als beispielsweise beim Supervised Learning, gehen wir beim Reinforcement Learning nicht davon aus, dass uns zu Beginn des Lernprozesses eine Datenmenge zur Verfügung steht. Stattdessen werden Daten aus der direkten Interaktion mit dem Environment

verwendet, die während des Lernprozesses gewonnen werden. Dennoch kann es in einem solchen Lernprozess Sinn ergeben, Gebrauch von *externen* Daten zu machen. Ein möglicher Anreiz dafür kann die Verwendung von **Demonstrationsdaten** sein.
Das **Imitation Learning** umfasst Methoden, die ein demonstriertes Verhalten möglichst genau imitieren wollen. Hierbei gehen wir davon aus, dass uns Demonstrationsdaten zur Verfügung stehen, die beispielsweise einem menschlichen Experten oder einer kostenintensiven Optimierung entstammen können. Das Szenario ist dabei wie folgt:
Die Problemstellung kann ähnlich wie beim Reinforcement Learning im Rahmen eines MEP $\mathcal{M} =(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R})$ modelliert werden. Es steht zudem ein Datensatz bestehend aus demonstrierten Trajektorien zur Verfügung

$$\mathcal{D} = \{\tau_i \mid i \in \{1, ..., N\}\} = \{\{(s_i^j, a_i^j) \mid j \in \{1, ..., |\tau_i|\}\} \mid i \in \{1, ..., N\}\} \sim \pi_E$$

wobei π_E der Policy des zu imitierenden Experten entspricht. Das Ziel ist es, auf Basis der Daten $\mathcal{D}$ eine Policy $\hat{\pi}_*$ zu finden, für die gilt:

$$\hat{\pi}_* = \underset{\hat{\pi}}{\operatorname{argmin}}\, \mathbb{E}[D(\hat{\pi}, \pi_E)]$$

wobei $D(.,.)$ ein geeignetes Ähnlichkeitsmaß zwischen zwei gegebenen Policies ist.
Das **Behaviour Cloning** (BC) stellt die direkteste Variante des Imitation Learnings dar. Die Problemstellung wird dabei als Supervised Learning interpretiert. Dies bedeutet konkret, dass die gegebene Menge an Demonstrationsdaten $\mathcal{D}$ als unabhängige und identisch verteilte Trainingsdaten (i.i.d.) behandelt werden. Ein Zustand $s_i \in \mathcal{S}$ stellt dabei den Input und die Aktion $a_i \in \mathcal{A}$ das dazu passende Label dar. Ein Modell mit trainierbaren Parametern θ soll dann auf Basis der Daten trainiert werden, sodass die Differenz zwischen der Expertenpolicy π_E und der eigenen $\hat{\pi}_\theta$ minimiert wird. Die Differenz lässt sich dabei für einen Zustand $s \in \mathcal{S}$ mit Hilfe einer Lossfunktion $\mathcal{L}(\pi_E(s), \hat{\pi}_\theta(s))$ beschreiben. Die gesuchte Policy kann damit wie folgt definiert werden:

$$\hat{\pi}_* = \underset{\theta}{\operatorname{argmin}}\, \mathbb{E}[\mathcal{L}(\pi_E(s), \hat{\pi}_\theta(s))],\ \forall s \in \mathcal{S}$$

Für ein solches Modell kann beispielsweise ein neuronales Netz verwendet werden, welches im vorherigen Abschnitt beschrieben wurde. Ein Beispiel für eine erfolgreiche Umsetzung ist das Projekt ALVINN (Autonomous Land Vehicle In a Neural Network), in welchem Ende der 1980er Jahre eines der ersten autonomen Fahrzeuge realisiert wurde [78]. Das Fahrzeug wurde durch ein dreischichtiges neuronales Netz gesteuert, dessen Input aus Bilddaten der aktuell befahrenen Straße bestand und dessen Output diskreten Aktionen entsprach, die an das Steuerungssystem des Fahrzeugs weitergeleitet wurden. Das neuronale Netz wurde auf Basis von Daten, die aus einem Simulator stammen, trainiert, bevor es auf die Straße gelassen wurde.
Ein großer Vorteil, den das BC mit sich bringt, ist seine Einfachheit. Rahmenbedingungen,

die durch das MEP gestellt werden, wie Belohnungen und Transitionsfunktionen, werden dabei vollständig ausgeblendet und das Problem damit auf das Mapping von Zuständen auf Aktionen reduziert.
Ein wesentlicher Nachteil dagegen ist das Verhalten in Situationen, die nicht im Trainingsdatensatz vorkommen. Da beim Training nur die Zustände im Demonstrationssatz $\mathcal{D}$ berücksichtigt werden, ist das Verhalten eines solchen Modells auf ungesehenen Daten nicht definierbar und stellt daher eine Unsicherheit dar. In vielen realen Anwendungen ist das Sammeln von Demonstrationsdaten zeitaufwändig und teuer, was zu kleineren Demonstrationssets führt und damit das Risiko erhöht, in unbekannte Situationen zu geraten, aus denen sich der Agent nicht mehr befreien kann.
Eine alternative Technik des Imitation Learning stellt dabei das **Inverse Reinforcement Learning** (IRL) dar. So wie das BC eine enge Beziehung zum Supervised Learning hat, so hat das IRL, wie der Name andeutet, eine solche zum Reinforcement Learning. Die Idee ist hier wie folgt: Das Ziel des RL ist es, eine Policy zu erlernen, die die kumulative Belohnung über eine Reihe von Schritten maximiert. Die Reward-Funktion $\mathcal{R}$ weist dafür einer Aktion a in einem Zustand s einen reellen Wert zu. Basierend auf dieser Funktion wird das Verhalten des Agenten optimiert, womit sein Verhalten vollständig durch $\mathcal{R}$ erklärt werden kann. Diese Idee wird beim IRL verwendet, um das Verhalten eines Experten zu reproduzieren. Konkret bedeutet dies, dass derartige Methoden aus gegebenen Expertendaten eine *Reward-Funktion* konstruieren, die das beobachtete Verhalten erklären soll. Anders als beim BC ist hier also nicht die Frage *wie* der Experte Entscheidungen fällt, sondern *warum*.
IRL-Algorithmen betrachten dazu ein modifiziertes MEP-Modell, das wie folgt definiert ist:

Definition 2.9: MEP ohne Belohnung

Ein **Markov Entscheidungsprozess ohne Belohnung** $\mathcal{M}_{\mathcal{R}}$ (MEP/R) ist ein stochastischer Prozess, der durch das Tripel $(\mathcal{S}, \mathcal{A}, \mathcal{P})$ beschrieben werden kann:

- Die Menge der **Zustände** $\mathcal{S}$ von $\mathcal{M}$
- Die Menge der **Aktionen** $\mathcal{A}$ von $\mathcal{M}$, wobei die Teilmengen $A_s \subseteq \mathcal{A}$ die in Zustand $s \in \mathcal{S}$ möglichen Aktionen darstellen
- Die **Transitionsfunktion** $\mathcal{P} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \mapsto [0, 1]$ von $\mathcal{M}$ gibt die Wahrscheinlichkeiten an, mit denen man von einem Zustand $s \in \mathcal{S}$ in einen Zustand $s' \in \mathcal{S}$ durch Ausführen einer Aktion $a \in \mathcal{A}_s$ gelangt.

Während beim Reinforcement Learning eine Policy gesucht wird, ist diese im Fall des IRL bereits gegeben. Diese wird durch den Datensatz des Experten repräsentiert. Es wird dabei angenommen, dass die Expertenpolicy optimal ist. Daraus folgt, dass die gesuchte Reward-Funktion $\mathcal{R}^*$ durch die Policy maximiert wird, sich also durch folgenden Ausdruck beschreiben lässt:

$$\pi_E = \underset{\pi \in \Pi}{\operatorname{argmax}} \mathbb{E}[\mathcal{R}^*(s,a)], \forall s \in \mathcal{S} \text{ und } \forall a \in \mathcal{A}_s$$

Mit Hilfe von Reinforcement Learning lässt sich schließlich eine Policy $\hat{\pi}$ auf Grundlage der durch IRL gewonnenen Reward-Funktion $\hat{\mathcal{R}}$ lernen. Das Zusammenspiel beider Disziplinen wird als **Apprenticeship Learning** bezeichnet [4].
Die Problematik beim IRL besteht darin, dass eine Policy in Bezug auf beliebig viele Belohnungsfunktionen optimal sein kann. Eine eindeutige Lösung existiert daher im Allgemeinen nicht. Das Problem ist damit *ill-posed* (dt. inkorrekt gestellt) [4]. Veröffentliche Lösungsansätze unterscheiden sich daher insbesondere darin, wie sie die Auswahl einer geeigneten Belohnungsfunktion begründen. Ein konkretes Verfahren dazu wird in Kapitel 3.2 präsentiert.

2.5 Diskussion

In diesem Kapitel haben wir uns mit grundlegenden Ideen des Reinforcement und Imitation Learnings befasst.
Das Reinforcement Learning eignet sich insbesondere zur Lösung sequentieller Problemstellungen. Das Problem der logistischen Verpackung kann als ein solches sequentielles Problem interpretiert werden. Dazu muss diese zunächst im Rahmen eines MEPs formuliert werden. Entscheidend ist dabei das Design der Reward-Funktion. Diese steuert maßgeblich das Verhalten eines Agenten und kann daher als Mittel zur Adaptivität genutzt werden. Mit Hilfe von Inverse Reinforcement Learning und geeigneten Expertendaten kann eine zum demonstrierten Verhalten adäquate Reward-Funktion gewonnen werden. Ein derartig definiertes MEP kann als Grundlage für das Training eines RL-Agenten dienen, dessen Policy sich dem Verhalten des Experten annähern soll (s. Abb. 2.9).
Um ein solches System realisieren zu können, bedarf es also neben der Implementierung eines Environments und eines RL-Agenten, zudem eine IRL-Methode. Im folgenden Kapitel werden dazu zu den hier vorgestellten Ideen konkrete Verfahren und eine mögliche Implementierung dieser präsentiert.

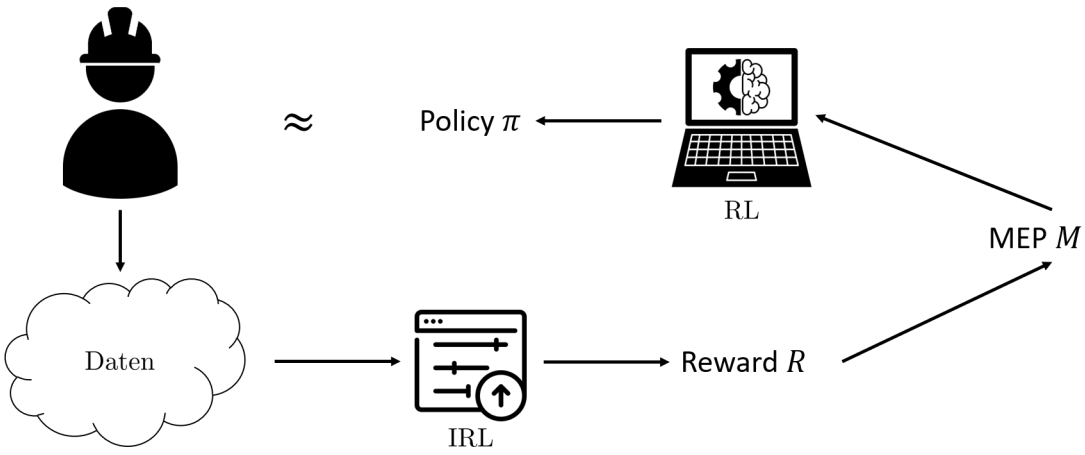


Abbildung 2.9: IRL-RL-Pipeline: Das Verhalten des Experten wird in Form von Daten bereitgestellt. Diese werden mit Hilfe einer IRL-Methode zu einer Reward-Funktion verarbeitet, die in ein MEP eingesetzt werden kann. Aufbauend darauf lässt sich ein RL-Agent trainieren, dessen Policy am Verhalten des Experten angelehnt ist.

Kapitel 3

Implementierung

In diesem Kapitel befassen wir uns mit einer möglichen Implementierung auf Basis der im vorherigen Kapitel vorgestellten Ideen. Um das Problem im Rahmen des (Inverse) Reinforcement Learnings angehen zu können, bedarf es zunächst der Realisierung des Environments, welche in Kapitel 3.1 vorgestellt wird. In unserem Fall fordert die Reward-Funktion besondere Aufmerksamkeit. Wie diese repräsentiert und ermittelt werden kann, wird im darauffolgenden Abschnitt 3.2 vorgestellt. Anschließend betrachten wir die Architektur und das Training des Agenten in Kapitel 3.3. Abschließend werden die präsentierten Verfahren in Kapitel 3.4 resümiert.

3.1 Environment

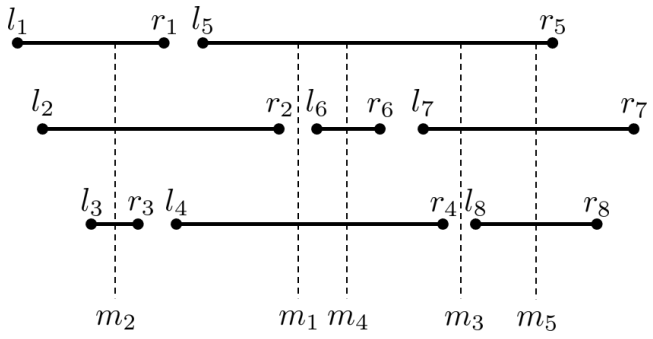
Das Environment verkörpert im Rahmen des Reinforcement Learning das zu lösende Problem. Im Fall der Verpackung muss also diese durch das Environment abgebildet werden. Dazu muss die Problemstellung im Rahmen eines MEP interpretiert und umgesetzt werden.
Ein Packauftrag beinhaltet, neben dem zur Verfügung stehenden Ladungsträger, eine zu platzierende Menge an Packstücken. Den Startzustand stellt dabei der leere Ladungsträger dar. Platzieren wir ein Packstück, so verändern wir die Beladung. Dies entspricht einer Transition, wobei das Platzieren des Packstücks die Aktion und die Beladungen vor und nach der Platzierung die beteiligten Zustände darstellen. Der Zustandsraum $\mathcal{S}$ des MEP setzt sich also aus den möglichen Beladungen des Ladungsträgers zusammen. Diese sind durch Aktionen, also die Platzierungen von Packstücken verbunden, welche durch Auswahl und Position des Packstücks bestimmt sind. Das Entfernen eines Packstücks wird dabei jedoch ausgeschlossen. Darüber hinaus werden weitere nachträgliche Veränderungen der Position von Packstücken, wie beispielsweise durch Kippeffekte, ignoriert. Wir nehmen also an, dass die gewünschte Position eines Packstücks auch der tatsächlichen Position entspricht. Die Transitionsfunktion ist damit deterministisch. Die Herleitung der

Reward-Funktion erfordert für das Vorhaben besondere Aufmerksamkeit. Diese wird daher in Kapitel 3.2 gesondert betrachtet.
Das Environment ist softwareseitig in zwei Komponenten aufgeteilt: dem **Geometriemodell** und dem **Bewerter**. Bei dem Geometriemodell handelt es sich um eine Datenstruktur zur Darstellung einer Beladung im Computer. Diese soll zudem Platzierungen verwerten und daraus resultierende Beladungen erstellen. Der Bewerter dagegen soll die Bewertung einer Beladung realisieren. Der Aufbau beider Komponenten wird im Folgenden beschrieben.

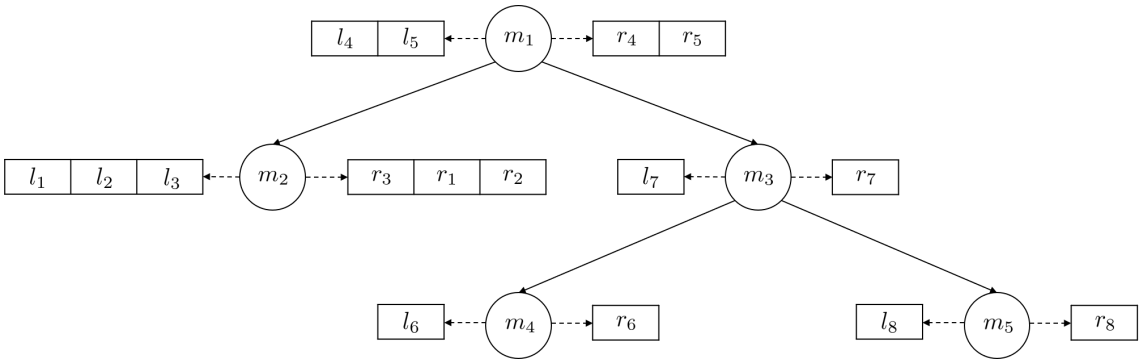
3.1.1 Geometriemodell

Um die Dimensionen und Positionen von Packstücken auf einem Ladungsträger repräsentieren zu können, bedarf es eines geeigneten Geometriemodells. Dieses soll durch eine adäquate Datenstruktur im Rechner repräsentiert werden. Die Anforderungen an diese sind eine speicherplatzeffiziente Darstellung der Beladung, als auch eine möglichst laufzeiteffiziente Erweiterung durch neue Platzierungen. Wünschenswert ist darüber hinaus, dass neue Platzierungen bezüglich ihrer Umsetzbarkeit überprüft werden können. Das heißt, die Datenstruktur soll insbesondere Überschneidungen mit bereits platzierten Objekten ermitteln und darauf aufbauend entscheiden können, ob eine geplante Positionierung valide ist. Damit soll gewährleistet werden, dass alle durchgeführten Platzierungen korrekt und tatsächlich umsetzbar sind.
Eine Möglichkeit zur Realisierung eines solchen Geometriemodells bieten **Intervallbäume** [58][28][9]. Hierbei handelt es sich um eine Datenstruktur, welche Intervalle in einer Baumstruktur speichert und für angefragte Intervalle alle Überlappungen der im Baum befindlichen Elemente ermitteln kann. Betrachten wir hierbei zunächst den eindimensionalen Fall.
Ein Intervallbaum kann als erweiterter Binärbaum realisiert werden. Das heißt, dass jeder Knoten im Baum, neben dem für Binärbäume üblichen Schlüssel, weitere Informationen trägt. In unserem Fall verfügt jeder Knoten $v \in B$ neben seinem Schlüssel und Zeigern zu seinen beiden Kindknoten, über zwei Hilfsarrays *leftArray* und *rightArray*. Der Aufbau eines solchen Baums lässt sich dabei wie in Algorithmus 3.1 dargestellt rekursiv definieren und wie folgt beschreiben:
Gegeben ist eine Menge von Intervallen $S = \{I_i | i \in \{1, 2, ..., n\}\}$ mit Intervallen $I_i = [l_i, r_i]$ für $l_i, r_i \in \mathbb{R}, l_i \leq r_i, i \in \{1, 2, ..., n\}$. Sei m der Median aller $2|S|$ in S gespeicherten Werte. Abhängig von m lassen sich nun drei Mengen definieren:

1. $S_l = \{I_i \mid r_i < m\}$, alle Intervalle die komplett links von m liegen
2. $S_r = \{I_i \mid l_i > m\}$, alle Intervalle die komplett rechts von m liegen
3. $S_m = \{I_i \mid l_i \leq m \leq r_i\}$, alle Intervalle die m beinhalten



(a) Intervalle



(b) Intervallbaum

Abbildung 3.1: 1D-Intervallbaum [58]: (a) zeigt beispielhafte Intervalle (b) zeigt den dazu gehörigen Intervallbaum. Die Knoten sind dabei über die Mediane der jeweiligen Intervallmengen definiert.

Wir verwenden m als Schlüssel für die Wurzel des Baums. Das linke Kind des Wurzelknotens ergibt sich aus dem rekursiven Aufruf der Konstruktion mit S_l als Intervallmenge. Das rechte Kind analog dazu mit S_r. Die beiden Hilfsarrays des Wurzelknotens sollen nun dazu dienen, die Intervallgrenzen von S_m zu speichern. Das Array *leftArray* beinhaltet dabei die linken Grenzen in aufsteigender Sortierung, das Array *rightArray* analog die rechten Endpunkte. Ein Beispiel für einen auf diese Weise konstruierten Baum zeigt Abbildung 3.1.

Da jeder rekursive Aufruf mit maximal der Hälfte der Intervalle erfolgt, ist die Höhe des Baumes durch $\mathcal{O}(\log n)$ beschränkt. Jedes Intervall wird zudem lediglich einmal abgespeichert, wodurch der Speicherplatz des Intervallbaums $\mathcal{O}(n)$ beträgt.

Wollen wir nun ein Intervall $V = [r_V, l_V]$ anfragen, also alle im Baum vorkommenden Überlappungen ermitteln, können wir dafür wie in Algorithmus 3.2 beschrieben vorgehen: Wir beginnen mit einer leeren Ergebnismenge F. Ist der aktuelle Knoten leer, geben wir F leer zurück. Ist dem nicht so, überprüfen wir, ob der im Schlüssel gespeicherte Median ins Anfrageintervall V fällt. Ist dem so, wissen wir dadurch, dass auch jedes im aktuellen

Algorithmus 3.1 InterallTreeConstruct

Input: Intervallmenge $S = \{I_i | i \in \{1, 2, ..., n\}\}$
Output: Baum B

1: **if** $S = \emptyset$ **then**
2: **return** null
3: **end if**
4: $m \leftarrow \text{Median}(S)$
5: $S_l \leftarrow \{I_i \mid r_i < m\}$
6: $S_r \leftarrow \{I_i \mid l_i > m\}$
7: $S_m \leftarrow \{I_i \mid l_i \leq m \leq r_i\}$
8: $x \leftarrow Node()$
9: $x.key \leftarrow m$
10: $x.leftArray \leftarrow \text{sorted}(\{l_i | I_i \in S_m\})$
11: $x.rightArray \leftarrow \text{sorted}(\{r_i | I_i \in S_m\})$
12: $x.leftChild \leftarrow \text{InterallTreeConstruct}(S_l)$
13: $x.rightChild \leftarrow \text{InterallTreeConstruct}(S_r)$
14: **return** x

Knoten gespeicherte Intervall mit V überlappt, weshalb wir diese dann der Ergebnismenge hinzufügen. Da weitere Intervalle in den Unterbäumen auch überlappen können, starten wir zwei rekursive Aufrufe auf den Kinderknoten. Liegt der Schlüssel nicht im angefragten Intervall, prüfen wir ob V rechts vom Median liegt. Ist dem so, prüfen wir für alle in $leftArray$ befindlichen Grenzen, ob diese kleiner sind als die rechte Grenze des Intervalls und damit überlappen. Aufgrund der Vorsortierung der Grenzen kann hierfür eine binäre Suche eingesetzt werden. Da auch weitere Überlappungen im linken Teilbaum möglich sind, starten wir einen rekursiven Aufruf auf dem Baum. Das Vorgehen wird analog angewendet, wenn V rechts vom Median liegt, also s_V größer als der Median ist.
Da die Höhe des Baums durch $\mathcal{O}(\log n)$ beschränkt ist und wir bei Verwendung der binären Suche insgesamt $\mathcal{O}(\log n)$ aufwenden müssen, um die Arrays zu durchsuchen, ergibt sich inklusive der Ausgabe von k Ergebniselementen eine Laufzeit von $\mathcal{O}(k + \log n)$ [58].

Auf Basis der Intervallbäume lässt sich nun ein für den betrachteten Use Case passendes Geometriemodell definieren. Betrachten wir Intervalle im dreidimensionalen Raum, fällt dabei auf, dass es sich hierbei geometrisch um Quader handelt (s. Abb. 3.2). Wir können also unsere Packstücke als dreidimensionale Intervalle betrachten, welche durch die Kanten der Objekte definiert werden. Die Werte der Intervalle ergeben sich dabei aus der Position des Packstücks und dessen Dimensionen.
Um Intervallbäume auch in 3D nutzen zu können, verwenden wir für jede Dimension einen eigenen Baum. Die Platzierung eines Packstücks wird damit über drei Bäume verteilt: der x-Baum verwaltet dabei das vom Packstück belegte x-Intervall, der y-Baum das ent-

Algorithmus 3.2 1D-Lookup

Input: Baum B, Intervall V

Output: $F = \{I \in B \mid V \cap I \neq \emptyset\}$

```
1: x ← B.root
2: r ← ∅
3: if x = null then
4:    return F
5: end if
6: if x.key ∈ V then
7:    for all I_i ∈ x do
8:       F ← F ∪ I_i
9:    end for
10:   F = F ∪ 1D-Lookup(x.leftChild.Tree, V)
11:   F = F ∪ 1D-Lookup(x.rightChild.Tree, V)
12: end if
13: if e_V < x.key then
14:   for all I_i ∈ x such that l_i ≤ r_V do
15:      F ← F ∪ I_i
16:   end for
17:   F ← F ∪ 1D-Lookup(x.leftChild.Tree, V)
18: end if
19: if l_V > x.key then
20:   for all I_i ∈ x such that r_i ≤ s_V do
21:      F ← F ∪ I_i
22:   end for
23:   F ← F ∪ 1D-Lookup(x.rightChild.Tree, V)
24: end if
25: return F
```

sprechende y-Intervall und der z-Baum entsprechend das z-Intervall. Um eine Platzierung wieder rekonstruieren und zuordnen zu können, wird zu jedem Eintrag in dem jeweiligen Baum zusätzlich eine ID gespeichert, welche das Packstück eindeutig identifiziert.
Die Kombination der drei Bäume kann dementsprechend eine Beladung repräsentieren. Wollen wir nun wissen, ob ein Kandidat für eine Platzierung valide ist, können wir dazu vorgehen, wie in Algorithmus 3.3 beschrieben. Eine Platzierung ist valide, wenn es keine Überschneidungen mit bereits gespeicherten Platzierungen gibt. Ein Quader überschneidet dabei mit einem anderen Quader, wenn sich die Körper definierenden Intervalle in jeder Dimension überschneiden. Darauf aufbauend, können wir also jede Dimension separat auf Überschneidungen testen und die Resultate zusammenführen. Wie in Zeile 4 gezeigt, kön-

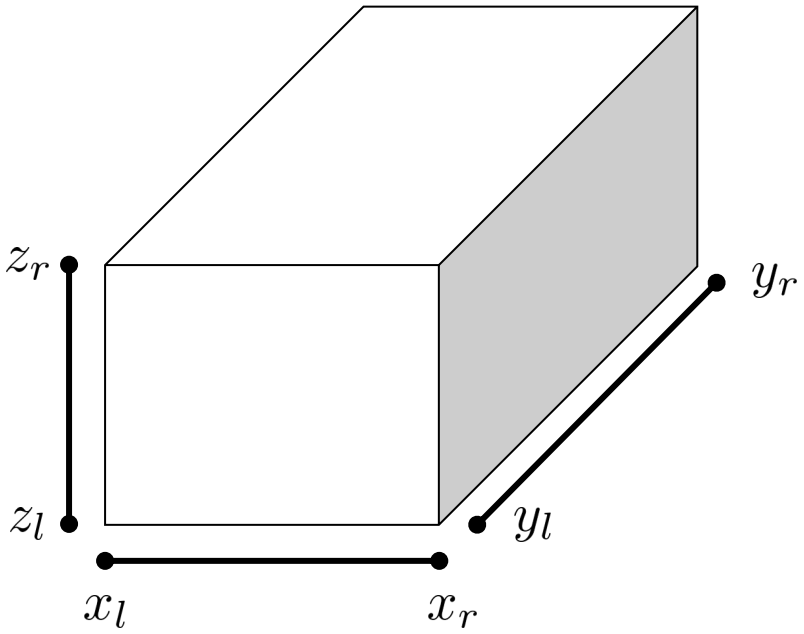


Abbildung 3.2: 3D-Intervall: Ein dreidimensionales Intervall ist über ein Intervall je Dimension definiert. Geometrisch lässt sich dies als Quader interpretieren.

nen wir die IDs der separaten 1D-Anfragen schneiden. Ist eine ID in jeder der Mengen enthalten, überlappt das korrespondierende Paket damit in jeder Dimension und damit auch im 3D-Raum. Die Laufzeit für Lookups steigt dabei um den konstanten Faktor 3, bleibt also asymptotisch weiterhin $\mathcal{O}(k + \log n)$.
Es sei hierbei zu erwähnen, dass die Methode der Intervallbäume aufgrund der Einfachheit der geometrischen Objekte und der orthogonalen Platzierungen anwendbar ist. Betrachten wir andere geometrische Formen oder flexiblere Rotationen der Packstücke, müssen für das Geometriemodell alternative Verfahren verwendet werden. Eine Möglichkeit stellen dabei unter anderem konvexe Hüllen, definiert über die Kanten der Packstücke, dar [10]. Überlappungsanfragen können dann über Schnittmengen der jeweiligen Hüllen beantwortet werden. Da diese jedoch komplexer und aufwendiger sind, wird hier auf die effizientere Lösung mittels Intervallbäumen zurückgegriffen.

Algorithmus 3.3 3D-Lookup

Input: x-Baum xB, y-Baum yB, z-Baum zB, Quaderdimensionen $d = (l, b, h)$, Position $p = (x, y, z)$
Output: $F = \{I \in B \mid V \cap I \neq \emptyset\}$

1: $xOverlap \leftarrow$IDs(1D-Lookup($xB, [x, x + l]$))
2: $yOverlap \leftarrow$IDs(1D-Lookup($yB, [y, y + b]$))
3: $zOverlap \leftarrow$IDs(1D-Lookup($zB, [z, z + h]$))
4: **return** $xOverlap \cap yOverlap \cap zOverlap$

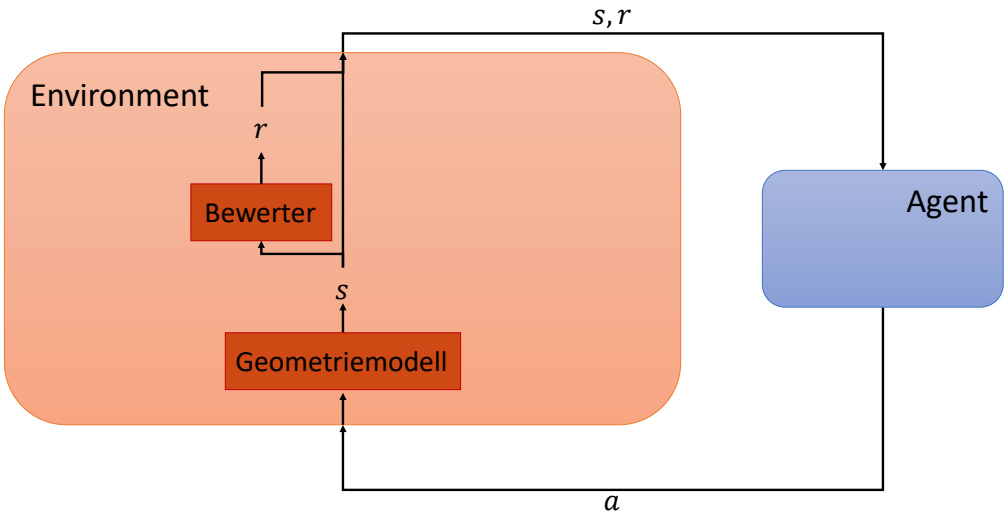


Abbildung 3.3: Environment-Architektur: Das Environment besteht aus zwei Komponenten: dem Geometriemodell und dem Bewerter. Während das Geometriemodell die neue Beladung berechnet, bewertet der Bewerter diese. Zusammen ergibt dies das Feedback des Environments.

3.1.2 Bewerter

Neben des neuen Folgezustands soll das Environment auch mit einer entsprechenden Belohnung auf eine Aktion des Agenten reagieren. Dazu muss die durch das Geometriemodell berechnete Beladung anhand der durchgeführten Platzierungen bewertet werden. Softwareseitig übernimmt dies in unserem Fall der Bewerter. Durch das IRL, welches im folgenden Abschnitt näher betrachtet wird, soll eine Reward-Funktion gewonnen werden, welche der Bewerter übernimmt, auf Beladungen anwendet und dem Agenten als Feedback übermittelt. Der Aufbau des Environments lässt sich damit wie in Abb. 3.3 zu sehen darstellen.

3.2 Reward

Um eine adäquate Reward-Funktion zu einer bestimmten Packstrategie finden zu können, bedarf es eines geeigneten IRL-Algorithmus. Dieser wird im folgenden Abschnitt vorgestellt.

3.2.1 Relative Entropy Inverse Reinforcement Learning

Um eine zur ausgeführten Strategie zugehörige Reward-Funktion zu gewinnen, bedarf es einer passenden IRL-Methode. Wie bereits in Kapitel 2.4 erwähnt, handelt es sich hierbei allerdings um ein *ill-posed* Problem, da keine eindeutige Lösung für das Problem existiert. Aus diesem Grund kam es zu einer Reihe verschiedener Ansätze, die die Auswahl einer Reward-Funktion anhand bestimmter Kriterien begründen wollen.
In dieser Arbeit soll der Ansatz von Boularias et al. implementiert werden [16]. Die Idee

hierbei ist es, die relative Entropie (Kullback-Leibler-Divergenz) zwischen einer empirischen und einer gelernten Verteilung zu minimieren. Das **Relative Entropy Inverse Reinforcement Learning** (REIRL) geht dabei vom in Kapitel 2.4 eingeführten MEP/R aus. Bei der fehlenden Reward-Funktion handelt es sich um eine Linearkombination verschiedener Features, die einen Zustand des MEP/R beschreiben. Die Entwicklungskoeffizienten der Linearkombinationen stellen dabei die Gewichte der einzelnen Features dar. Die gesuchte Reward-Funktion lässt sich damit wie folgt definieren:

$$\mathcal{R}^*(s,a) = \sum_{i=1}^{k} \theta_i f_i(s,a) = \theta f(s,a)$$

Die Features $f_i(s,a)$ für $s \in \mathcal{S}$ und $a \in \mathcal{A}_s$ beschreiben dabei numerisch bestimmte Eigenschaften eines Zustands. Die Entwicklungskoeffizienten θ_i gewichten diese Features.
Die in den Expertendemonstrationen auftretenden Features lassen sich wie folgt ermitteln:

$$\hat{f} = \frac{1}{|\mathcal{D}|} \sum_{(s,a)\in\mathcal{D}} f(s,a)$$

Tritt ein Feature $\hat{f}_i$ besonders häufig in den Experten-Demonstrationen $\mathcal{D}$ auf, so soll dieses auch entsprechend gewichtet werden. Der Vorteil einer solchen Darstellung der Reward-Funktion, im Gegensatz zu beispielsweise einer Darstellung mittels neuronaler Netze [102][36][46] oder Gauß-Prozesse [59], ist deren Einfachheit und die daraus resultierende Interpretierbarkeit.
Das Verfahren nimmt dabei an, dass die Wahrscheinlichkeit einer Trajektorie proportional zu dem erlangten Reward ist:

$$Pr(\tau|\theta,\mathcal{P}) \propto \exp(\theta f^\tau) \prod_{t=1}^{|\tau|} \mathcal{P}(s_t,a_t,s_{t+1}) = \exp(R(\tau)) \prod_{t=1}^{|\tau|} \mathcal{P}(s_t,a_t,s_{t+1}) \tag{3.1}$$

Eine Trajektorie ist also umso wahrscheinlicher, je höher der erlangte Reward ist. Ein schlechtes Verhalten des Experten wird damit als eher unwahrscheinlich angenommen.
Um die Entwicklungskoeffizienten θ zu finden, nutzt das Verfahren die relative Entropie zweier Verteilungen. Die relative Entropie ist ein Maß für die Ähnlichkeit zweier Verteilungen und ist wie folgt definiert [55]:

Definition 3.1: Relative Entropie (Kullback-Leibler-Divergenz)

Die relative Entropie zweier (diskreter) Verteilungen P und Q ist für die Menge X gegeben durch:

$$RE(P,Q) = \sum_{x\in X} P(x) \cdot \ln\left(\frac{P(x)}{Q(x)}\right)$$

Je höher dabei das Maß $RE(.|.)$ ist, desto unterschiedlicher sind die beiden Verteilungen. Das REIRL geht von zwei Verteilungen P und Q aus. P entspricht dabei der Verteilung der

Trajektorien $\tau \in \mathcal{T}$ des MEP/R, wobei $\mathcal{T}$ die Gesamtheit aller Trajektorien darstellt. Die Verteilung Q bezieht sich ebenfalls auf die Trajektorien des MEP/R, allerdings in Bezug auf eine sogenannte Baseline-Policy mit Transitionsmatrix T^Q. Die Problemstellung des REIRL lässt sich darauf aufbauend wie folgt definieren:

$$\begin{aligned} \min_P \quad & \sum_{\tau \in \mathcal{T}} P(\tau) \ln\left(\frac{P(\tau)}{Q(\tau)}\right) \\ \text{s.t.} \quad & \sum_{\tau \in \mathcal{T}} |P(\tau) f_i^\tau - \hat{f}_i| \leq \epsilon, \quad \forall i \in \{1, ..., k\} \qquad \text{(I)} \\ & \sum_{\tau \in \mathcal{T}} P(\tau) = 1 \qquad \text{(II)} \\ & P(\tau) \geq 0, \qquad \forall \tau \in \mathcal{T} \quad \text{(III)} \end{aligned} \tag{3.2}$$

Wir suchen also eine Verteilung P über die Trajektorien des Raumes, die die relative Entropie zur Verteilung Q minimiert. Dabei wird mit (I) gefordert, dass die Verteilung P expertennahe Features bevorzugt, der Abstand zu den demonstrierten Features im Erwartungswert also gering ist. Die Kriterien (II) und (III) stellen zudem sicher, dass die Wahrscheinlichkeiten sich zu 1 summieren und zudem nicht-negativ sind.
Die Lösung des Optimierungsproblem wird dabei durch eine Lagrange-Optimierung hergeleitet. Das Problem lässt sich wie folgt in einer Lagrange-Gleichung darstellen [32]:

$$\begin{aligned} L(P, \theta, \eta) = \sum_{\tau \in \mathcal{T}} P(\tau) \cdot \ln\left(\frac{P(\tau)}{Q(\tau)}\right) & - \sum_{i=1}^{k} \theta_i \left(\sum_{\tau \in \mathcal{T}} P(\tau) f_i^\tau - \hat{f}_i \right) \\ & - \sum_{i=1}^{k} |\theta_i| \epsilon_i \\ & + \eta \left(\sum_{\tau \in \mathcal{T}} P(\tau) - 1 \right) \end{aligned} \tag{3.3}$$

Es gilt:

$$\begin{aligned} \frac{\partial L(P, \theta, \eta)}{\partial P(\tau)} &= \ln\left(\frac{P(\tau)}{Q(\tau)}\right) - \sum_{i=1}^{k} \theta_i f_i^\tau + \eta + 1 \\ &\stackrel{!}{=} 0 \end{aligned}$$

Durch Nullsetzen des Terms ergibt sich folgende Beziehung:

$$\begin{aligned}
\ln\left(\frac{P(\tau)}{Q(\tau)}\right) - \sum_{i=1}^{k} \theta_i f_i^\tau + \eta + 1 = 0 \\
\ln\left(\frac{P(\tau)}{Q(\tau)}\right) = \sum_{i=1}^{k} \theta_i f_i^\tau - \eta - 1 \\
\frac{P(\tau)}{Q(\tau)} = \exp(\sum_{i=1}^{k} \theta_i f_i^\tau - \eta - 1) \\
P(\tau) = Q(\tau) \exp(\sum_{i=1}^{k} \theta_i f_i^\tau - \eta - 1) \\
P(\tau) = Q(\tau) \frac{\exp(\sum_{i=1}^{k} \theta_i f_i^\tau)}{\exp(\eta - 1)}
\end{aligned}$$

Für die Gesamtheit der Trajektorien in $\mathcal{T}$ gilt damit:

$$\begin{aligned}
\sum_{\tau \in \mathcal{T}} P(\tau) &= \sum_{\tau \in \mathcal{T}} \left(Q(\tau) \frac{\exp(\sum_{i=1}^{k} \theta_i f_i^\tau)}{\exp(\eta - 1)} \right) \\
1 &= \sum_{\tau \in \mathcal{T}} \left(Q(\tau) \frac{\exp(\sum_{i=1}^{k} \theta_i f_i^\tau)}{\exp(\eta - 1)} \right) \\
\exp(\eta - 1) &= \sum_{\tau \in \mathcal{T}} \left(Q(\tau) \exp(\sum_{i=1}^{k} \theta_i f_i^\tau) \right) \qquad \stackrel{def}{=} Z(\theta)
\end{aligned}$$

Durch Einsetzen von $Z(\theta)$ erhalten wir dadurch:

$$P(\tau|\theta) = \frac{1}{Z(\theta)} Q(\tau) \exp(\sum_{i=1}^{k} \theta_i f_i^\tau)$$

Die dazu duale Problemstellung ergibt sich zu:

$$g(\theta) = \sum_{i=1}^{k} \theta_i \hat{f}_i^\tau - \ln Z(\theta) - \sum_{i=1}^{k} |\theta_i| \epsilon_i$$

Die zu maximierende Funktion g ist konkav und stetig differenzierbar, außer bei $\theta = 0$. Diese kann mit einem Subgradientenverfahren optimiert werden, welches sich zur Optimie-

rung nicht vollständig differenzierbarer Funktionen eignet [17]. Der Subgradient ergibt sich in diesem Fall durch:

$$\frac{\partial g(\theta)}{\partial \theta_i} = \hat{f}_i - \sum_{\tau \in \mathcal{T}} P(\tau) f_i^\tau - \alpha_i \epsilon_i$$

$$\text{mit } \alpha_i = \begin{cases} 1 & \theta_i \geq 0 \\ -1 & \theta_i < 0 \end{cases}$$

Da eine analytische Bestimmung des Gradienten aufwendig und nur dann möglich ist, wenn die Transitionsfunktionen bekannt ist, stellen die Autoren zudem eine Methode vor, diesen empirisch zu schätzen. Dazu wird die Verteilung Q mit Hilfe zweier Ausdrücke dargestellt:

$$Q(\tau) = D(\tau)U(\tau)$$

$$\begin{aligned} \text{mit } D(\tau) &= \prod_{t=1}^{H} T(s_t, a_t, s_{t+1}) \\ U(\tau) &= \frac{1}{|\mathcal{A}|^H} \\ \tau &= s_1 a_1, ..., s_H a_H \end{aligned}$$

Der Term D beschreibt dabei die Transitionswahrscheinlichkeiten, U dagegen die Wahrscheinlichkeiten der gewählten Aktionen. Letztere werden hierbei als gleichverteilt angenommen. Damit lässt sich P wie folgt ausdrücken:

$$P(\tau|\theta) = \frac{D(\tau) \exp\left(\sum_{i=1}^{k} \theta_i f_i^\tau\right)}{\sum_{\tau \in T} D(\tau) \exp\left(\sum_{i=1}^{k} \theta_i f_i^\tau\right)}$$

Die Autoren wollen diesen Ausdruck durch Samples approximieren. Dazu schlagen diese vor, **Importance Sampling** zu verwenden. Hierbei handelt es sich um eine Technik aus dem Bereich des Monte-Carlo-Samplings.

Exkurs: Importance Sampling

Gegeben sei eine Zufallsvariable X mit zugehöriger Verteilung $P(X)$ und einer Funktion $f(X)$. Der Erwartungswert von $f(X)$ ergibt sich dabei wie folgt:

$$\mathbb{E}[f(X)] = \int_{x \in X} f(x)P(x)dx$$

Ist es nicht möglich diesen analytisch zu ermitteln, können wir den Erwartungswert mit Hilfe von Stichproben empirisch abschätzen. Im einfachsten Fall würde sich dieser wie folgt ermitteln lassen:

$$\hat{\mathbb{E}}[f(X)] = \frac{1}{N}\sum_{i=1}^{n} f(x_i),\ x_i \sim P(x)$$

wobei n die Stichprobengröße und x_i die Elemente der Stichprobe darstellen. Letztere werden dabei ausgehend der Verteilung $P(X)$ gezogen. Dies kann jedoch ineffizient sein, wenn der für den Erwartungswert relevante Wertebereich von X klein ist und unzureichend von P abgedeckt wird (s. Abb. 3.4). Wünschenswert wäre es die Stichprobenelemente proportional zum Einfluss in den Erwartungswert zu ziehen. Das **Importance Sampling** stellt dafür eine mögliche Lösung dar [74][63]. Diese lässt sich wie folgt herleiten:

$$\begin{aligned}
\mathbb{E}[f(X)] &= \int_{x\in X} f(x)P(x)dx \\
&= \int_{x\in X} f(x)P(x)\frac{Q(x)}{Q(x)}dx \\
&= \underbrace{\int_{x\in X} f(x)\frac{P(x)}{Q(x)}Q(x)dx}_{\approx\hat{\mathbb{E}}[f(X)\frac{P(X)}{Q(X)}]}
\end{aligned}$$

Statt also nun von P zu samplen, können wir zur Abschätzung des Erwartungswertes eine alternative Verteilung Q nutzen. Idealerweise ist diese so zu wählen, dass $Q \propto f$ gilt. Asymptotisch erhalten wir zwar mit steigender Stichprobengröße dasselbe Resultat wie beim naiven Sampling, können allerdings schneller zu einem zufriedenstellenden Resultat konvergieren [63].

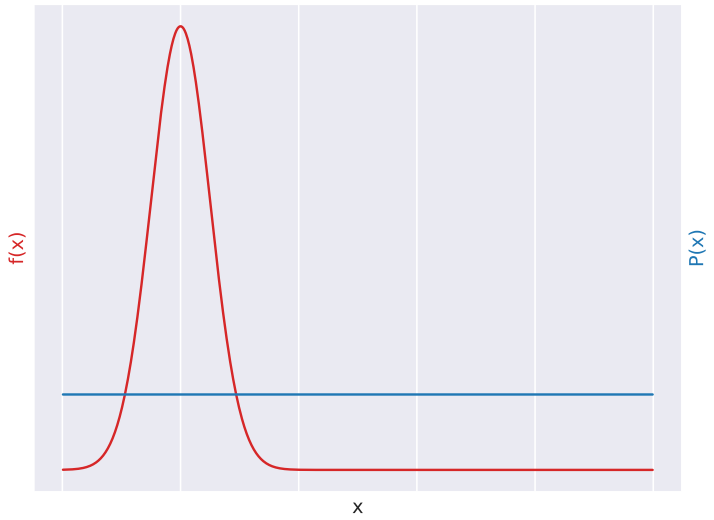


Abbildung 3.4: Negativbeispiel für naives Sampling: Nehmen wir als Sampling-Verteilung eine Gleichverteilung an (blau), so treffen wir mit den Stichproben nur selten in relevante Bereiche der abgebildeten Funktion $f(x)$ (rot). Um einen zufriedenstellenden Erwartungswert schätzen zu können, brauchen wir daher viele Stichproben. Wünschenswert wäre eine Verteilung proportional zur Relevanz des Wertes.

Dazu werden Sample-Trajektorien $\mathcal{T}_{\pi_N}$ auf Basis einer Policy π_N generiert, mit $|\mathcal{T}_{\pi_N}| = M$. Eine Schätzung des Subgradienten kann auf Basis derer wie folgt beschrieben werden:

$$
\begin{aligned}
\frac{\partial \hat{g(\theta)}}{\partial \theta_i} &= \hat{f}_i - \frac{1}{M} \sum_{\tau \in \mathcal{T}_{\pi_N}} \frac{P(\tau|\theta)}{D(\tau)\pi_N(\tau)} f_i^\tau - \alpha_i \epsilon_i \\
&= \hat{f}_i - \frac{1}{M} \sum_{\tau \in \mathcal{T}_{\pi_N}} \frac{\left(\frac{D(\tau) \exp\left(\sum_{i=1}^k \theta_i f_i^\tau\right)}{\sum_{\tau \in T} D(\tau) \exp\left(\sum_{i=1}^k \theta_i f_i^\tau\right)} \right) f_i^\tau}{D(\tau)\pi_N(\tau)} - \alpha_i \epsilon_i \\
&= \hat{f}_i - \frac{1}{M} \frac{\left(\sum_{\tau \in \mathcal{T}_{\pi_N}} \frac{D(\tau) \exp\left(\sum_{i=1}^k \theta_i f_i^\tau\right)}{D(\tau)\pi_N(\tau)} \right) f_i^\tau}{\sum_{\tau \in T} D(\tau) \exp\left(\sum_{i=1}^k \theta_i f_i^\tau\right)} - \alpha_i \epsilon_i \\
&\overset{IS}{=} \hat{f}_i - \frac{\frac{1}{M}\left(\sum_{\tau \in \mathcal{T}_{\pi_N}} \frac{D(\tau) \exp\left(\sum_{i=1}^k \theta_i f_i^\tau\right)}{D(\tau)\pi_N(\tau)} \right) f_i^\tau}{\frac{1}{M}\left(\sum_{\tau \in \mathcal{T}_{\pi_N}} \frac{D(\tau) \exp\left(\sum_{i=1}^k \theta_i f_i^\tau\right)}{D(\tau)\pi_N(\tau)} \right)} - \alpha_i \epsilon_i \\
&= \hat{f}_i - \frac{\frac{1}{M}\left(\sum_{\tau \in \mathcal{T}_{\pi_N}} \frac{D(\tau)}{D(\tau)} \frac{\exp\left(\sum_{i=1}^k \theta_i f_i^\tau\right)}{\pi_N(\tau)} \right) f_i^\tau}{\frac{1}{M}\left(\sum_{\tau \in \mathcal{T}_{\pi_N}} \underbrace{\frac{D(\tau)}{D(\tau)}}_{=1} \frac{\exp\left(\sum_{i=1}^k \theta_i f_i^\tau\right)}{\pi_N(\tau)} \right)} - \alpha_i \epsilon_i \\
&= \hat{f}_i - \frac{\left(\sum_{\tau \in \mathcal{T}_{\pi_N}} \frac{\exp\left(\sum_{i=1}^k \theta_i f_i^\tau\right)}{\pi_N(\tau)} \right) f_i^\tau}{\left(\sum_{\tau \in \mathcal{T}_{\pi_N}} \frac{\exp\left(\sum_{i=1}^k \theta_i f_i^\tau\right)}{\pi_N(\tau)} \right)} - \alpha_i \epsilon_i
\end{aligned}
$$

$$
\text{mit } \pi_N(\tau) = \prod_{i=1}^{H} Pr(a_t|s_t)
$$

Indem wir den Gradienten anhand von Samples schätzen, können wir schlussendlich die Gewichte der Reward-Funktion entsprechend trainieren. Das Verfahren wird in Algorithmus 3.4 zusammengefasst.

Algorithmus 3.4 Relative Entropy Inverse Reinforcement Learning

Input: Expertendaten $D_E \sim \pi_E$, Sample-Trajektorien $\mathcal{T}_{\pi_N} \sim \pi_N$, Lernrate β, Stopkriterium μ

Output: Gewichte θ

1: $\theta^0 \leftarrow \vec{0}$
2: $j \leftarrow 1$
3: $\hat{f} = \frac{1}{|D|} \sum_{(s,a) \in D} f(s,a)$
4: **repeat**
5: **for all** $\tau \in \mathcal{T}_{\pi_N}$ **do**
6: $P(\tau|\theta^j) = \frac{\exp\left(\theta^j f^\tau\right)/\pi_N(\tau)}{\sum_{\tau \in T_B} \exp(\theta f^\tau)/\pi_N(\tau)}$
7: **end for**
8: **for** $i = 0$ **to** k **do**
9: $\Delta\theta_i^j \leftarrow \hat{f}_i - \sum_{\tau \in \mathcal{T}_{\pi_N}} P(\tau|\theta^j) f_i^\tau - \alpha_i \epsilon_i$
10: $\theta_i^{j+1} \leftarrow \theta_i^j + \beta \left(\Delta\theta_i^j\right)$
11: $j \leftarrow j + 1$
12: **end for**
13: **until** $||\theta^j - \theta^{j-1}||_2 < \mu$
14: **return** θ

Die ersten drei Zeilen des Algorithmus dienen zur Initialisierung. Dabei werden die Expertenfeatures $\hat{f}$ aus den Expertendaten berechnet, der Gewichtsvektor mit Nullen initialisiert und ein Zähler j gestartet. Anschließend startet das Gradientenverfahren in einer Schleife. Dazu schätzen wir zunächst $P(\tau|\theta^j)$ anhand der aktuellen Gewichte für jede der durch π_N gesampelten Trajektorien. In der Zeile 9 wird dann darauf aufbauend der Gradient für jedes Feature f_i berechnet und in Zeile 10 in die Gewichte verrechnet. Dabei gehen wir, wie bei einem Gradientenverfahren üblich, nur eine vorgegebene Schrittweite β, die während des Trainings auch variiert werden kann. Unterschreitet die relative Veränderung der Gewichte einen Threshold μ, werden die Gewichte ausgegeben.
Das REIRL konnte bereits erfolgreich in einigen Domänen eingesetzt werden. Dazu zählt unter anderem das Erlernen von Strategien erfahrener Tischtennis-Spieler [65], in Videospielen [26], aber auch in der Robotik [36][16].
Um das Verfahren auch auf den in dieser Arbeit betrachteten Use Case anwenden zu können, bedarf es neben der Experten- und Samplingdaten zudem Features, die einen Zustand beschreiben. Wir benötigen also Merkmale, anhand derer wir eine Beladung beschreiben können.

3.2.2 Features

Um die Arbeitsweise eines Packers beschreiben zu können, bedarf es Merkmale, anhand derer wir eine Beladung charakterisieren können. Wir bezeichnen diese Merkmale als **(Pack-)Features**. Hierbei handelt es sich um Eigenschaften der Beladung, die mit Hilfe numerischer Kennzahlen beschrieben werden. Im folgenden Abschnitt wird eine Anzahl von Features definiert, anhand derer wir die Strategie eines Arbeiters erklären wollen. Die Features sind dabei so gewählt, dass sie folgende Eigenschaften erfüllen: zum einen sollen die Werte der Features im selben Wertebereich liegen, damit eine Vergleichbarkeit zwischen diesen gewährleistet ist. Hierbei bietet sich das Intervall $[0, 1] \in \mathbb{R}$ für die Kennzahlen an. Zudem sollen die Kennzahlen relativ zum Auftrag berechnet werden, sodass eine Vergleichbarkeit zwischen verschiedenen Packaufträgen möglich ist. Absolute Werte können je nach Packauftrag variieren, sodass zwischen verschiedenen Packaufträgen große Differenzen entstehen können. Eine Kennzahl soll zudem einen Packauftrag als Ganzes beurteilen und nicht auf Eigenschaften einzelner Pakete eingehen.

Relative Höhe

Ein wichtiges Merkmal einer Beladung stellt die resultierende Höhe des Paketstapels dar. Idealerweise lässt sich die resultierende Höhe in Relation zur optimalen Höhe setzen. Da allerdings das Ermitteln der optimalen Höhe mit der Lösung des BPP und damit eines $\mathcal{NP}$-schweren Problems einher geht, ist dies nicht praktikabel. Wir müssen daher auf eine alternative Abschätzung setzen. Eine Möglichkeit besteht dabei darin, das Resultat in Relation zu einer Worst- bzw. Best-Case Schranke zu setzen. Die Höhe einer Beladung ergibt sich dabei wie folgt:

$$\text{height} = \max_i(z_i + h_i).$$

Die Worst-Case-Höhe einer Beladung ergibt sich, wenn alle Pakete auf einer Stelle übereinandergestapelt werden:

$$\text{height}_{max} = \sum_{i=1}^{n} h_i.$$

Eine mögliche Best-Case-Höhe gibt die maximale Pakethöhe des Packauftrags an:

$$\text{height}_{min} = \max_i h_i.$$

Diese kann zwar nicht immer erreicht werden, stellt aber eine recheneffiziente Approximation dar.
Eine Kennzahl für die Höhe kann damit wie folgt formuliert werden:

$$RH = 1 - \frac{(\text{height} - \text{height}_{min})}{(\text{height}_{max} - \text{height}_{min})} \tag{3.4}$$

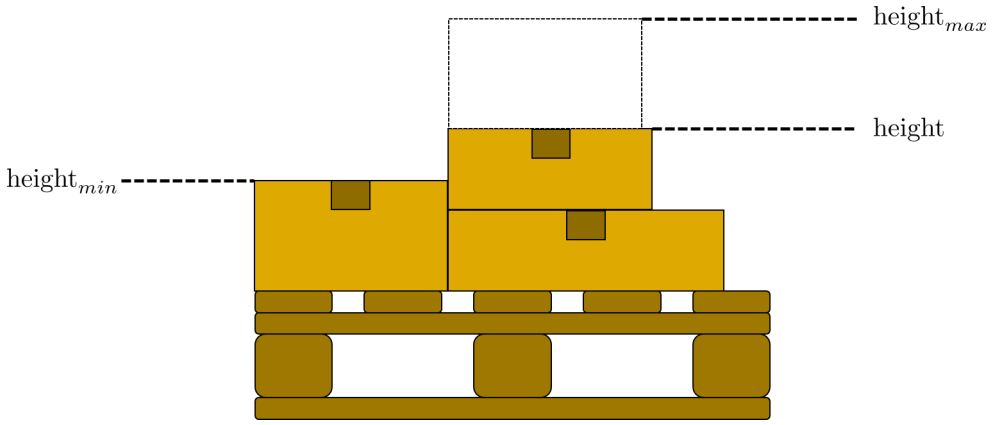


Abbildung 3.5: Veranschaulichung der relativen Höhe: Die minimale Höhe, die ein Paketstapel theoretisch erreichen kann, ist durch die Höhe des höchsten Pakets beschränkt. Die Summe der Höhen dagegen liefern eine obere Schranke. Die tatsächliche Höhe eines Paketstapels kann in Relation der beiden Schranken bewertet werden.

Je näher RH[1] dabei an 1 herankommt, desto näher ist die erreiche Höhe height am möglichen Optimum (s. Abb. 3.5).

Relatives Gewicht

Neben der Höhe der Beladung, stellt auch die Verteilung des Gewichts eine wichtige Eigenschaft dar. Für dieses lässt sich analog ein ähnliches Feature definieren. Als Grundlage dient dabei das maximale Gewicht, welches auf eine Stelle des Ladungsträgers einwirkt:

$$\text{weight} = \max_{x,y} w_{x,y}$$

wobei $w_{x,y}$ das Gewicht an der Position (x, y) angibt. Auch hier ergibt sich eine Worst-Case-Schranke dadurch, alle Pakete aufeinander zu stapeln:

$$\text{weight}_{max} = \sum_{i=1}^{n} w_i.$$

Der Best-Case ergibt sich analog aus dem Gewicht des schwersten Packstücks:

$$\text{weight}_{min} = \max_{i} w_i.$$

Das daraus resultierende Feature kann damit wie folgt bestimmt werden:

$$RW = 1 - \frac{(\text{weight} - \text{weight}_{min})}{(\text{weight}_{max} - \text{weight}_{min})}. \tag{3.5}$$

Auch hier gilt, je näher der Wert RW [2] an 1 kommt, desto näher ist man einer optimalen Verteilung des Gewichts auf dem Ladungsträger.

[1] RH = Relative Height
[2] RW = Relative Weight

Relative Ränder

Die Ränder einer Beladung stellen den Abstand der Packstücke zum Ladungsträgerrand dar. Ein gewisser Rand ist wünschenswert, da dieser eine Polsterung der Beladung durch beispielsweise Luftpolsterfolie ermöglicht, ohne die Maße des Ladungsträgers zu überschreiten. Die Ränder ergeben sich dabei aus den maximalen bzw. minimalen Positionen der Packstücke auf dem Ladungsträger (s. Abb. 3.6):

$$UM = \frac{L - \max_i(y_i + l_i)}{L}$$

$$BM = \frac{\min_i(y_i)}{L}$$

$$RM = \frac{B - \max_i(x_i + b_i)}{B}$$

$$LM = \frac{\min_i(x_i)}{B}$$

Diese sind allerdings absolut und nicht auf den betrachteten Auftrag bezogen. Um diese zu relativieren, können wir diese ins Verhältnis mit dem optimalen Wert setzen, welcher sich aus der maximalen Packstückdimension ergibt:

$$RUM = \frac{UM}{\max_i l_i} \tag{3.6}$$

$$RBM = \frac{BM}{\max_i l_i} \tag{3.7}$$

$$RRM = \frac{RM}{\max_i b_i} \tag{3.8}$$

$$RLM = \frac{LM}{\max_i b_i} \tag{3.9}$$

Je näher dabei einer der Werte [3][4][5][6] an 1 liegt, desto größer ist der relative Rand zur jeweiligen Kante des Ladungsträgers.

Relative Packungsdichte

Die Dichte eines Paketstapels ist definiert als das von ihm aufgebrauchte Volumen [82]. Hierbei ist es wünschenswert, dieses zu minimieren, also einen Paketstapel mit möglichst wenig Lücken zu packen. Um die Dichte zu relativieren, lässt sich die resultierende Dichte mit der minimalen Dichte in Relation setzen. Letztere ergibt sich dabei aus der Summe der

[3] RUM = Relative Upper Margin
[4] RBM = Relative Bottom Margin
[5] RRM = Relative Right Margin
[6] RLM = Relative Left Margin

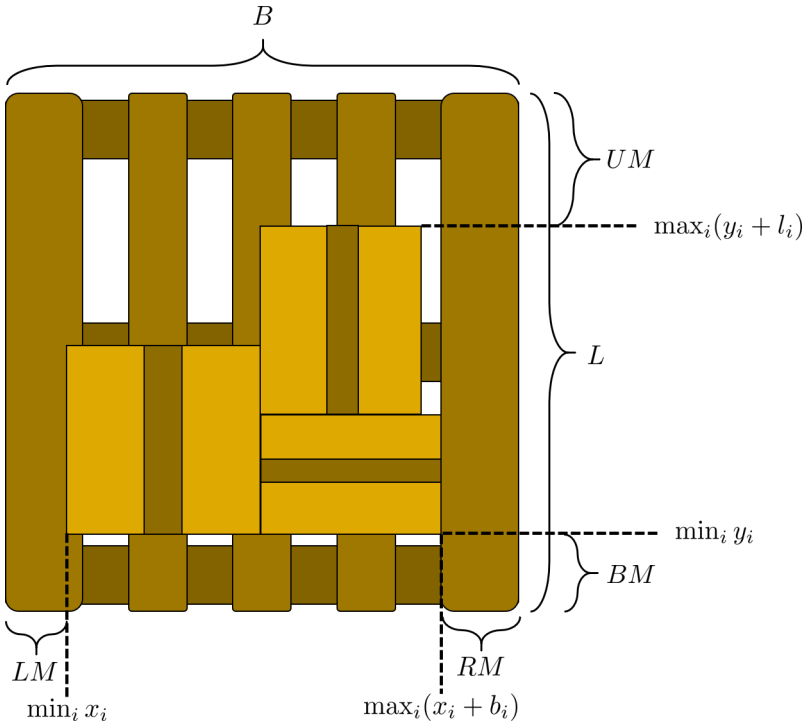


Abbildung 3.6: Veranschaulichung des Randes: Der Rand einer Beladung ergibt sich aus den Extremstellen der Beladung. Das Bild soll andeuten, wie die absoluten Ränder ermittelt werden. Diese lassen sich anhand der maximalen Werte, die sich aus den Paketdimensionen ergeben, relativieren.

Volumina aller im Packauftrag befindlichen Packstücke. Die relative Packungsdichte lässt sich damit wie folgt definieren:

$$RD = \frac{\sum_{i=i}^{n} V_i}{(\max_x - \min_x) \cdot (\max_y - \min_y) \cdot \text{height}} \tag{3.10}$$

Der Nenner ergibt sich dabei aus dem tatsächlichen Volumen der Beladung. Dieses kann über die minimalen und maximalen Punkte der Packstücke bestimmt werden. Der Zähler gibt die minimale Dichte an. Je näher die tatsächliche Dichte an diese herankommt, desto näher kommt der Wert des Ausdrucks RD [7] an 1.

Relativer Überbauquotient

Bei Packaufträgen einer gewissen Größe, lässt es sich nicht vermeiden, Pakete übereinanderstapeln zu müssen. Hierbei ist wichtig, dass für die Pakete in höheren Lagen ein stabiler Untergrund gewährleistet ist. Hierfür ist es wünschenswert, dass ein Packstück durch möglichst viele Packstücke unter diesem gestützt wird [82][13]. Eine mögliche Kennzahl, um dies für einen Packstapel beurteilen zu können, liefert der Überbauquotient. Dieser setzt die

[7] RD = Relative Density

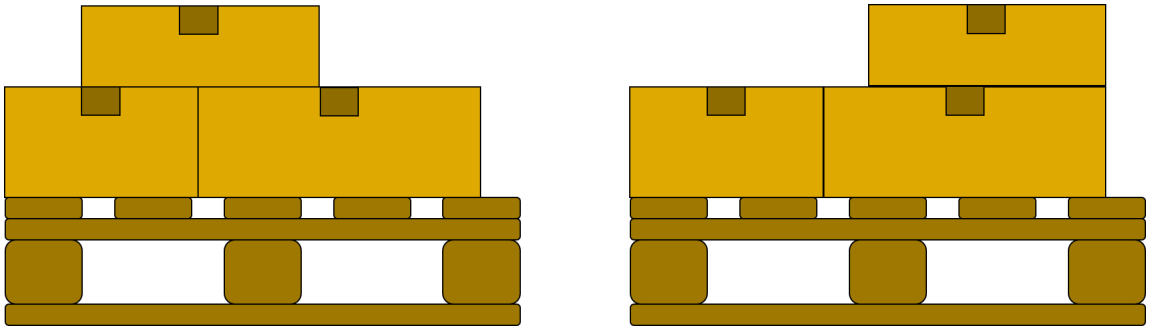


Abbildung 3.7: Veranschaulichung des Überbauquotienten: Die beiden Abbildungen zeigen zwei unterschiedliche Anordnungen derselben Packstücke. Dabei ist die linke Anordnung zu bevorzugen, da hierbei das obere Pakete mehr Stützflächen in Anspruch nimmt und dadurch stabiler steht.

Anzahl der Pakete, also möglicher Stützflächen, in Relation zu den tatsächlich genutzten Stützflächen und lässt sich wie folgt definieren:

$$OBR = 1 - \frac{\#\text{Pakete} - \#_1\text{Pakete}}{\#\text{Stützflächen} - \#_1\text{Pakete}} \tag{3.11}$$

Die Werte $\#_1$Pakete bzw. $\#_1$Stützflächen geben dabei die Pakete bzw. Stützflächen in der ersten Schicht an. Da hier allerdings der Ladungsträger als Stützfläche dient, werden diese hier nicht miteinbezogen. Je höher die Anzahl der Stützflächen in den übrigen Schichten ist, desto näher ist der Ausdruck OBR[8] an 1 und damit an einem wünschenswerten Ergebnis.

Relativer Überhangquotient

Ragt ein Packstück über seine Stützflächen hinaus, hängt also mit einem Teil seiner Fläche in der Luft, spricht man vom Überhang. In dieser Arbeit werden zwar Kippeffekte von Packstücken außer Acht gelassen, es ist jedoch in Hinblick auf eine tatsächliche Anwendung wünschenswert, den Überhang zu minimieren. Dieser lässt sich mit folgendem Ausdruck beschreiben:

$$OHR = 1 - \frac{\#\text{Überhängende Pakete} - \#_1\text{Pakete}}{\#\text{Pakete} - \#_1\text{Stützflächen}} \tag{3.12}$$

Hierbei wird die Anzahl der überhängenden Pakete in Relation mit der Gesamtzahl der Pakete gesetzt. Der Wert OHR [9] kommt dabei näher an 1, je weniger überhängende Pakete auf dem Ladungsträger sind.

Relative Höhe des gemeinsamen Schwerpunkts

Bezüglich des Gewichtes der Beladung ist es im Allgemeinen erwünscht, dass schwere Packstücke unten und leichtere oben Platz finden [82]. Andernfalls kann es nämlich dazu kommen, dass die Packstücke in den unteren Schichten durch das Gewicht Schaden nehmen.

[8] OBR = Overbuilt Ratio
[9] OHR = Overhang Ratio

Eine mögliche Kennzahl, um einen solchen Sachverhalt zu beschreiben, bietet die relative Höhe des gemeinsamen Schwerpunkts. Der Gewichtsschwerpunkt des Stapels ergibt sich dabei aus dem Superpositionsprinzip. Das Gewicht eines Pakets wird dabei als punktuelle Kraft betrachtet, die auf Höhe des platzierten Objekts wirkt. Anders ausgedrückt, entspricht die Höhe des gemeinsamen Schwerpunkts dem Mittelwert der Höhen der Pakete, gewichtet mit dem jeweiligen Gewicht des Pakets. Dieses lässt sich wie folgt formulieren:

$$c_z = \frac{1}{\sum_{i=1}^{n} w_i} \cdot \sum_{i=1}^{n} z_i \cdot w_i$$

Die Höhe des Schwerpunkts c_z ist allerdings absolut und muss auf den spezifischen Auftrag relativiert werden. Als Referenz kann dabei die resultierende Höhe des Paketstapels dienen:

$$RBC = 1 - \frac{c_z}{\text{height}} \tag{3.13}$$

Der Wert RBC [10] kommt dabei näher an 1, je niedriger der Schwerpunkt in Relation zur erreichten Höhe ist. Das Gewicht ist in diesem Fall also möglichst weit unten und damit wünschenswert.

3.3 Agent

3.3.1 Deep Deterministic Policy Gradient

Prinzip

Aufbauend auf dem durch das IRL vervollständigte Environment, können wir nun einen möglichen Agenten definieren und implementieren. Dieser soll auf Basis von Interaktionen mit dem Environment eine möglichst gewinnbringende Policy entwickeln. Das Reinforcement Learning bietet dazu eine Reihe möglicher Algorithmen und Verfahren an. In dieser Arbeit wurde dafür das **Deep Deterministic Policy Gradient**-Verfahren (DDPG) für stetige Environments von Lillicrap et al. implementiert [61]. Bei dem 2016 veröffentlichtem Verfahren handelt es sich um einen Algorithmus des Actor-Critic-Paradigmas. Hierfür charakteristisch ist, dass der Agent intern in zwei Rollen aufgeteilt wird: dem Actor und dem Critic. Der Actor nimmt den Zustand des Environments s_t entgegen, wertet diesen aus und entscheidet sich für eine durchzuführende Aktion a_t. Der Critic bewertet die Entscheidung des Actors in Bezug auf den zu erwarteten Reward. Das Ziel des Actors ist es, die Bewertung des Critics möglichst zu maximieren, die des Critics dagegen eine möglichst genaue Prognose abzugeben. Die Autoren des Papers schlagen dazu vor, den Critic die Q-Funktion repräsentieren zu lassen. Diese kann, abhängig des Zustands des Environments und der vom Actor ausgewählten Action, den zu erwartenden Reward bestimmen und stellt somit ein

[10] RBC = Relative Barycenter

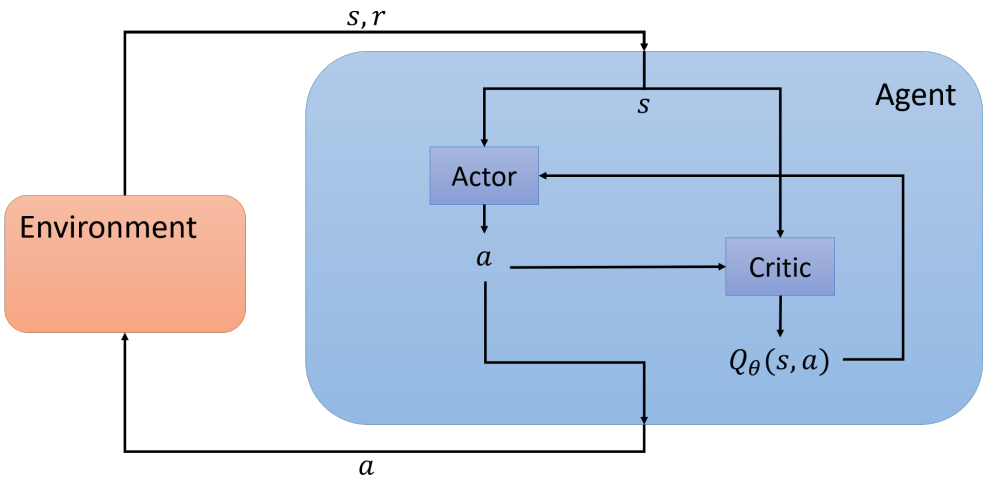


Abbildung 3.8: Agent-Architektur: Der Actor wählt die durchzuführende Aktion, während der Critic diese in Abhängigkeit des Zustands bewertet.

geeignetes Kriterium zur Bewertung des Actors dar. Das Prinzip wird in Abb. 3.8 visualisiert. Die beiden Rollen werden dabei mit Hilfe von neuronalen Netzen implementiert. Diese gilt es so anzutrainieren, dass der Critic eine möglichst genaue Approximation der Q-Funktion darstellt und der Actor diese mit seinen ausgewählten Aktionen maximiert. Die Modellparameter des Critics werden mit θ^Q und die des Actors mit θ^π beschrieben.
Die für das Training benötigten Daten entstehen durch die direkte Interaktion des Agenten mit dem Environment. Hierbei entstehen Transitionen $(s_t, a_t, r_t, s_{t+1}, d_t)$, wobei s_t den Ausgangszustand, a_t die ausgeführte Aktion, r_t den jeweiligen Reward, s_{t+1} den Folgezustand und d angibt, ob in s_{t+1} die Episode beendet ist. Die auftretenden Transitionen werden in einem endlichen Buffer $\mathcal{B}$ abgelegt. Ist dieser voll, werden die jeweils ältesten Einträge durch neue Transitionen ersetzt (*FIFO*-Prinzip). Die im Buffer anfallenden Transitionen dienen dann als Trainingsdaten für die beiden Netze. Bei jedem Trainingsschritt wird dafür eine Menge an Transitionen, ein *Batch*, aus dem Buffer gleichverteilt gezogen. Dadurch wird ein unabhängig und identisch verteiltes Lernen der Netze ermöglicht.
Die Loss-Funktionen der beiden Netze ergeben sich dabei aus den jeweiligen Aufgaben. Der Critic soll die Q-Funktion möglichst genau approximieren, die zugehörige Loss-Funktion ergibt sich für einen Batch b aus der Bellman-Gleichung:

$$\mathcal{L}_C = \frac{1}{|b|} \sum_{(s,a,s',r,d)\in b} \left(Q(s,a|\theta^Q) - y(r,s',d)\right)^2, \tag{3.14}$$

$$\text{mit } y(r,s',d) = r + \gamma(1-d)\cdot Q(s', \pi(s'|\theta^\pi)|\theta^Q)$$

Wir vergleichen also die Vorhersage des Critics mit dem tatsächlichen Resultat des Environments anhand ihrer mittleren quadratischen Abweichung. Man spricht hierbei vom

Q-Learning [99][66].
Der Actor soll eine Policy lernen, die den erwarteten Ertrag maximiert. Da wir von einem stetigen Environment ausgehen, wird angenommen, dass die Q-Funktion in Bezug auf eine Aktion $a \in \mathcal{A}$ differenzierbar ist. Die Loss-Funktion des Actors kann damit wie folgt für einen Batch b beschrieben werden:

$$\mathcal{L}_A = \frac{1}{|b|} \sum_{s \in b} Q(s, \pi(s|\theta^\pi)|\theta^Q) \tag{3.15}$$

Ein Problem, das beim Anlernen beider Netze auf diese entsteht, ist eine mögliche Divergenz des Trainings. So haben Mnih et al. gezeigt, dass das Training eines Netzwerks instabil ist, wenn die Zielwerte ebenfalls anhand der aktuellen Netzwerkparameter ermittelt werden (s. Gl. 3.14) [66]. Als Lösung schlagen die Autoren vor, sowohl für Actor, als auch für den Critic jeweils zwei Netzwerke zu verwalten. Ein Netzwerk dient dabei wie bislang als aktives *Policy*-Netz. Das zweite Netzwerk wird als *Target*-Netz bezeichnet und dient dazu die Zielwerte der Loss-Funktionen zu bestimmen. Die Parameter der beiden Netzwerke werden in regelmäßigen Abständen synchronisiert. Die DDPG-Autoren schlagen dafür eine *Polyak*-Synchronisierung vor [77]. Hierfür werden die Parameter des Policy-Netzwerks zu einem Anteil von $1 - \rho$ übernommen:

$$\theta^Q_{target} = \rho\theta^Q_{target} + (1-\rho)\theta^Q_{policy}$$
$$\text{bzw.} \qquad \theta^\pi_{target} = \rho\theta^\pi_{target} + (1-\rho)\theta^\pi_{policy}$$

Die Einführung der Target-Netzwerke stabilisieren die Zielwerte und damit auch das Training der Netze.
Einem Problem, dem sich alle RL-Algorithmen stellen müssen, ist das *Exploration-Exploitation-Dilemma* (dt. Erkundungs-Ausbeutungs-Dilemma). Im Idealfall probiert ein Agent alle möglichen Pfade, die das Environment bietet, aus, um den lukrativsten zu finden. Dies ist allerdings in großen Environments aufgrund der hohen Anzahl möglicher Pfade nicht praktikabel. Es gilt daher einen Trade-Off zwischen Ausbeutung und Erkundung zu finden. Der Agent soll dabei lukrativen Pfaden folgen, aber auch das Environment genügend erkunden können. Um dies zu ermöglichen, führen die Autoren des DDPG zudem eine *Action-Noise* ein. Dabei werden die Aktionen, die der Actor wählt, verrauscht, sodass auch ungeplante Aktionen und Pfade erkundet werden können. Eine Möglichkeit, dies zu realisieren, stellt die ϵ-Greedy-Policy dar. Hierbei wird die Aktion des Agenten mit einer Wahrscheinlichkeit von ϵ durch eine zufällige Aktion ausgetauscht. Die Wahrscheinlichkeit nimmt dabei im Laufe des Trainings ab, sodass der Agent mit fortlaufender Zeit mehr ausbeutet, als erkundet.
Der Lernalgorithmus für DDPG wird in Algorithmus 3.5 zusammengefasst.

Algorithmus 3.5 Deep Deterministic Policy Gradient

Input: Environment E, Initiale Actor-Parameter θ^{π}, Initiale Q-Parameter θ^{Q}, Polyak ρ, Critic-Lernrate β^{Q}, Actor-Lernrate β^{π}, Batch-Size m, Synchronisationsfrequenz f, Epsilon ϵ, Stoppkriterium μ

1: $\mathcal{B} \leftarrow \emptyset$
2: $\theta^{Q}_{policy} \leftarrow \theta^{Q}$
3: $\theta^{\pi}_{policy} \leftarrow \theta^{\pi}$
4: $\theta^{Q}_{target} \leftarrow \theta^{Q}$
5: $\theta^{\pi}_{target} \leftarrow \theta^{\pi}$
6: $s \leftarrow E.\text{init}()$
7: $i \leftarrow 0$
8: **while** $!\mu$ **do**
9: **if** $x \sim [0,1] < \epsilon$ **then**
10: $a \leftarrow a \sim \mathcal{A}_s$
11: **else**
12: $a \leftarrow \pi(s_t|\theta^{\pi}_{policy})$
13: **end if**
14: $s', r, d \leftarrow E.\text{step}(a)$
15: $\mathcal{B} \leftarrow \mathcal{B} \cup \{(s, a, s', r, d)\}$
16: $s \leftarrow s'$
17: $b = \{(s, a, s', r, d)\} \sim \mathcal{B}$
18: $y(r, s', d) = r + \gamma(1 - d) \cdot Q(s', \pi(s'|\theta^{\pi}_{target})|\theta^{Q}_{target})$
19: $\Delta_C = \frac{1}{|b|} \sum_{(s,a,s',r,d) \in b} \left(Q(s, a|\theta^{Q}_{policy}) - y(r, s', d) \right)^2$
20: Update Critic-Policy-Network mit $\beta^{Q} \Delta_C$
21: $\Delta_A = \frac{1}{|b|} \sum_{s \in b} Q(s, \pi(s|\theta^{\pi}_{policy})|\theta^{Q}_{policy})$
22: Update Actor-Policy-Network mit $\beta^{\pi} \Delta_A$
23: **if** $i \bmod f == 0$ **then**
24: $\theta^{Q}_{target} = \rho\theta^{Q}_{target} + (1 - \rho)\theta^{Q}_{policy}$
25: $\theta^{\pi}_{target} = \rho\theta^{\pi}_{target} + (1 - \rho)\theta^{\pi}_{policy}$
26: **end if**
27: $i \leftarrow i + 1$
28: Update ϵ
29: **end while**

Die Zeilen 1 bis 7 initialisieren die Netzwerkwerkparameter, den Buffer, das Environment und einen Schleifenzähler. Die in Zeile 8 beginnende Schleife läuft bis das Stoppkriterium μ erfüllt ist, welches sich beispielsweise auf die Performance der Modelle oder auf die Anzahl der Durchläufe beziehen kann. Innerhalb der Schleife wird je ein Schritt im Environment ausgeführt. Die daraus resultierende Transition wird in Zeile 15 im Buffer gespeichert.

Anschließend werden die Netzwerke aktualisiert. Dazu wird zunächst ein Batch der Größe m aus dem Buffer gezogen. Anhand der Transitionen werden die Loss-Funktionen der Netzwerke bestimmt und dementsprechend aktualisiert. Anschließend werden, falls nötig, die Target-Netzwerke synchronisiert und der Trainingsvorgang startet erneut.

Eingabe

Um für einen DDPG-Agenten unsere Zwecke verwenden zu können, bedarf es zunächst einer geeigneten Repräsentation einer Beladung, um es in einem neuronalen Netz verarbeiten zu können. Wünschenswert wäre dabei eine Repräsentation, die unabhängig der Anzahl bereits und noch zu platzierender Packstücke, eine feste Dimension beibehält.
Da nicht nur das DDPG, sondern auch das Deep Reinforcement Learning im Allgemeinen erfolgreich auf das Lernen anhand von Pixeln angewandt werden konnten, bietet sich daher eine Darstellung in Form eines Bildes als mögliche Repräsentation an.
Eine geeignete Bilddarstellung muss dabei insbesondere den Ladungsträger und die sich darauf befindlichen Packstücke abbilden können. Die Dimensionen des Ladungsträgers lassen sich über die Dimensionen des Bildes darstellen. Der Einfachheit halber fixieren wir diese auf 100x100 Pixel. Packaufträge verschiedener Dimensionen können entsprechend skaliert werden. Die Packstücke, die sich auf der Palette befinden, können über die Intensität der Pixel repräsentiert werden. Dazu gehen wir von einem Farbkanal mit einem Intensitätsintervall pro Pixel von {0,...,255} aus. Die Höhe an einer Stelle der Palette kann dabei über die Intensität kodiert werden: je höher die Beladung an einer Stelle ist, desto höher ist dabei der Pixelwert. Dies kann als Aufnahme einer idealen Tiefenkamera, die über dem Ladungsträger angebracht wird, interpretiert werden. Analog lässt sich eine solche Darstellung für das Gewicht auf der Palette erstellen, wodurch zwei *Feature-Maps* die Geometrie der Beladung beschreiben können. Die Abbildung 3.9 zeigt dafür ein einfaches Beispiel. Neben der aktuellen Beladung sollte das Netzwerk zudem wissen, was es Packen soll. In der Online-Version des Bin-Packings muss dazu nur das nächste Packstück kodiert werden. Dieses kann ähnlich wie die gesamte Beladung über zwei Feature-Maps jeweils für Höhe und Gewicht kodiert werden. Damit die Dimensionen gleich bleiben, werden die Maße dazu

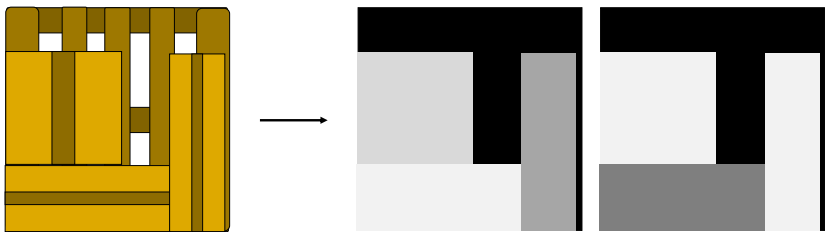


Abbildung 3.9: Bilddarstellung eines Ladungsträgers: Die geometrischen Merkmale der Beladung können als Bild dargestellt werden. Die Intensität des Pixels gibt dabei an, wie hoch bzw. schwer die Objekte an dieser Stelle sind.

Abbildung 3.10: Bilddarstellung eines Packstücks: Ähnlich wie der Ladungträger, kann auch ein Packstück mit Hilfe von Bildern dargestellt werden.

ebenfalls auf einem 100x100 Bild eingetragen. Die Abbildung 3.10 zeigt dazu ein Beispiel. Die jeweils vier Feature-Maps können konkateniert und als Input für das Actor-Netzwerk verwendet werden. Dieses gibt dann eine entsprechende Position des zu verpackenden Packstücks auf der Palette an. Die Platzierung wird dann, nachdem es vom Geometriemodell verarbeitet wurde, entsprechend im Bild eingetragen.

Um die Bilddaten verarbeiten zu können, greift ein solches Netzwerk auf die in Kapitel 2.3.4 thematisierten Faltungsschichten zurück. Dabei sind neben Stride und Padding insbesondere die Tiefe und die Anzahl der Filter pro Schicht einstellbar. Nach Verarbeitung des Inputs durch die Faltungsschichten gehen die Daten in ein MLP über.

Für die Offline-Variante der Problemstellung müssen alle noch zu packenden Packstücke in den Input kodiert werden. Packaufträge weisen jedoch eine variable Länge auf, was zu einem Problem in Hinblick auf einen Input fester Größe wird. Ein Workaround, um ein solches Netz dennoch für die Offline-Variante des Problems verwenden zu können, wird in Kapitel 3.3.2 beschrieben.

Output

Der Output kodiert die vom Netz gewünschte Position für die Platzierung eines Packstücks. Die Autoren sehen eine stetige Ausgabe des Netzwerks vor. Diese wird realisiert indem der Output der finalen Schicht durch eine tanh-Funktion ausgewertet und damit auf ein Intervall von [-1,1] abgebildet wird.
Für unsere Zwecke lassen sich hierfür zwei Ausgabeneuronen verwenden, die jeweils X- und Y-Koordinaten der vom Netz errechneten Platzierung darstellen. Die durch die tanh-Funktion auf den Bereich [-1,1] abgebildeten Werte lassen sich auf eine entsprechende Position auf dem Ladungsträger umrechnen. Die Z-Koordinate ergibt sich aus der minimalen Höhe, bei der keine Überlappung mit bereits platzierten Packstücken entsteht. Diese kann leicht über das Geometriemodell ermittelt werden, wodurch auch eine valide Platzierung gewährleistet ist.
Da das DDPG jedoch eine stetige Position als Aktion ausgibt und die vorgestellte Bilddarstellung inhärent diskret aufgelöst ist, muss eine Aktion zusätzlich diskretisiert werden. Dazu kann die stetige Position auf die nächste durch das Pixelraster definierte Position gerundet werden [33]. Gehen wir von einem 100x100 Raster, der tanh-Funktion und dem Netzwerk-Output f^L aus, ergibt sich damit für den Output:

$$x = [\tanh(f_0^L) + 1) \cdot 50]$$
$$\text{bzw. } y = [\tanh(f_1^L) + 1) \cdot 50].$$

Behaviour Cloning

Ein Problem, das insbesondere in großen Environments wie unserem auftritt, ist es eine initial gute Policy zu finden. Die Parameter der Netzwerke werden häufig zufällig initialisiert, wodurch zu Beginn eine zufällige Policy implementiert wird. Durch die Action-Noise soll das Environment erkundet werden. Bis dadurch allerdings eine brauchbare Policy entsteht, kann es insbesondere bei einem großen Aktionsraum dauern.
Die vom Experten erzeugten Demonstrationsdaten können neben ihrer Funktion im IRL auch zur Beschleunigung des Trainings des Agenten verwendet werden. Vor allem die initiale Suche nach einer guten Policy kann dadurch deutlich beschleunigt werden [70][105]. Für das DDPG lässt sich beispielsweise der Ansatz von Nair et al. implementieren [70]. Dies ist durch das in Kapitel 2.4 aufgefasste Behaviour Cloning motiviert. Die Demonstrationsdaten werden dabei als Trainingsdaten interpretiert und mit Hilfe einer zusätzlichen Loss-Funktion in das Training des DDPG-Agenten integriert. Die Demonstrationsdaten $\mathcal{D}$ werden dazu in einem separatem Buffer $\mathcal{B}_\mathcal{D}$ abgelegt. Zusätzlich wird im Trainingspro-

zess ein Batch aus diesem Buffer gezogen und der Actor anhand folgender Loss-Funktion trainiert:

$$\mathcal{L}_{BC} = \frac{1}{|b|} \sum_{s,a \in b} \left(\pi(s|\theta^{\pi}_{policy}) - a\right)^2$$

Es wird also der mittlere quadratische Abstand zwischen Ausgabe des Actors und der Expertendemonstration berechnet. Hierbei sind Parallelen zum Supervised Learning erkennbar: die Zustände in den Expertendemonstrationen stellen die Eingaben dar, während die vom Experten durchgeführte Aktion das dazu passende Label repräsentiert.
Die in Kapitel 2.4 genannten Schwächen des BC sind hierbei zu vernachlässigen, da das Training des Agenten sich nicht nur auf die Demonstrationsdaten stützt, sondern auch ungesehene Zustände der eigenständigen Interaktion mit dem Environment berücksichtigt.
Die Autoren schlagen vor, die zwei Loss-Funktionen des Actors jeweils zu gewichten. Die Parameter $\lambda_1, \lambda_2 \in [0, 1]$ steuern dabei, wie stark die jeweiligen Loss-Funktion in das Training des Actors miteingehen.
Der DDPG-Lernalgorithmus mit Demonstrationsdaten wird in Algorithmus 3.6 zusammengefasst.
Der grundsätzliche Ablauf bleibt im Vergleich zum Algorithmus 3.5 gleich. Anders ist jedoch, dass nun zusätzlich der Demonstrationsbuffer $\mathcal{B}_{\mathcal{D}}$ und die Gewichtungen λ_1 und λ_2 übergeben werden. Diese werden dazu genutzt, um in den Zeilen 22 und 23 den zusätzlichen Behaviour Cloning-Loss zu bestimmen und auf den Actor anzuwenden. Zu beachten ist dabei, dass wir diesen minimieren und den ursprünglichen Actor-Loss maximieren. Daher unterscheiden sich in Zeile 24 die Vorzeichen der beiden Loss-Funktionen.

Algorithmus 3.6 Deep Deterministic Policy Gradient from Demonstrations

Input: Environment E, Initiale Actor-Parameter θ^π, Initiale Q-Parameter θ^Q, Polyak ρ, Critic-Lernrate β^Q, Actor-Lernrate β^π, Batch-Size m, Synchronisationsfrequenz f, Epsilon ϵ, Stoppkriterium μ, Demonstrationsbuffer $\mathcal{B}_\mathcal{D}$, Gewichtungen λ_1, λ_2

$\mathcal{B} \leftarrow \emptyset$
$\theta^Q_{policy} \leftarrow \theta^Q$
$\theta^\pi_{policy} \leftarrow \theta^\pi$
$\theta^Q_{target} \leftarrow \theta^Q$
$\theta^\pi_{target} \leftarrow \theta^\pi$
$i \leftarrow 0$
$s \leftarrow E.\text{init}()$
while $!\mu$ **do**
  if $x \sim [0,1] < \epsilon$ **then**
    $a \leftarrow a \sim \mathcal{A}_s$
  else
    $a \leftarrow \pi(s_t|\theta^\pi_{policy})$
  end if
  $s', r, d \leftarrow E.\text{step}(a)$
  $\mathcal{B} \leftarrow \mathcal{B} \cup \{(s, a, s', r, d)\}$
  $s \leftarrow s'$
  $b = \{(s, a, s', r, d)\} \sim \mathcal{B}$
  $y(r, s', d) = r + \gamma(1 - d) \cdot Q(s', \pi(s'|\theta^\pi_{target})|\theta^Q_{target})$
  $\Delta_C = \frac{1}{|b|} \sum_{(s,a,s',r,d)\in b} \left(Q(s, a|\theta^Q_{policy}) - y(r, s', d)\right)^2$
  Update Critic-Policy-Network mit $\beta^Q \Delta_C$
  $\Delta_A = \frac{1}{|b|} \sum_{s\in b} Q(s, \pi(s|\theta^\pi_{policy})|\theta^Q_{policy})$
  $b_\mathcal{D} = \{(s, a, s', r, d)\} \sim \mathcal{B}_\mathcal{D}$
  $\Delta_{BC} = \frac{1}{|b_D|} \sum_{s,a\in b_D} \left(\pi(s|\theta^\pi_{policy}) - a\right)^2$
  Update Actor-Policy-Network mit $\beta^\pi(\lambda_1 \Delta_A - \lambda_2 \Delta_{BC})$
  if $i \bmod f == 0$ **then**
    $\theta^Q_{target} = \rho\theta^Q_{target} + (1 - \rho)\theta^Q_{policy}$
    $\theta^\pi_{target} = \rho\theta^\pi_{target} + (1 - \rho)\theta^\pi_{policy}$
  end if
  $i \leftarrow i + 1$
  Update ϵ
end while

3.3.2 Erweiterung auf Offline-Problem

In den vorherigen Abschnitten wurde ein Agent auf Basis von DDPG vorgestellt. Dieser bekommt als Input neben Informationen zur aktuellen Beladung lediglich Angaben zu *einem* Packstück. Dadurch eignet sich der vorgestellte Agent in erster Linie für die Online-Variante des betrachteten BPP, in der wir also die Packstücke nacheinander bekommen und nicht alle von Beginn an verfügbar sind. In der Praxis werden Packaufträge zunächst in der Kommissionierung zusammengestellt und dann zum Packer befördert. Einige Kommissioniersysteme verschicken dabei die Packstücke einzeln, beispielsweise über ein Fließband, und eben nicht als Gesamtheit [93]. Aufgrund solcher Systeme hat auch die Online-Variante des Problems seine praktische Relevanz. Dennoch ist auch die Lösung des Offline-Problems in der Logistik ein relevantes Thema, weshalb auch diese hier betrachtet werden soll.
Ein Problem, das im Zusammenhang mit der Offline-Variante entsteht, ist die variable Auftrags- und damit Eingabegröße für das RL-Modell. Zwar existieren Ansätze dafür, neuronale Netze mit Input variabler Länge zu verwenden (bspw. rekurrente Netze), diese erhöhen die Komplexität derartiger Modelle und erschweren das Training. Wünschenswert wäre es, die vorgestellte Online-Variante wiederverwenden zu können. Im Folgenden wird dafür ein Ansatz auf Basis von **Monte Carlo Tree Search** (MCTS) vorgestellt.

Exkurs: Monte Carlo Tree Search

Bei der MCTS handelt es sich um einen heuristischen Suchalgorithmus. Die Idee hierbei ist es, einen Suchbaum für das betrachtete Problem aufzubauen. Die Knoten des Baums repräsentieren dabei die Zustände eines MEPs. Die Wurzel stellt dabei den aktuellen Startzustand dar. Die Verbindung von einem Eltern- zu einem Kindknoten entsteht durch eine Aktion, die in den Zustand des Kindknotens führt. Neben des eigentlich Zustands verwaltet ein Knoten zusätzliche Meta-Informationen, die bei der Auswertung eines Knotens helfen sollen. Der Baum wird anhand von Simulationen befüllt und ausgebaut. Wurden genug Simulationen durchgeführt, lässt sich anhand dessen eine geeignete Aktion auswählen.
Konkret lässt sich das Verfahren in vier Phasen unterteilen:

- Selection

  In der Selection-Phase soll ein Knoten des Baums ausgesucht werden, von dem aus der Baum weiter ausgebaut werden soll. Dazu werden die Knoten des Baums ebenenweise von der Wurzel aus anhand eines Kriteriums ausgewertet. Ein gängiges Kriterium ist dabei die *upper confidence bound for trees* [54]:

  $$UCT(s,a) = \frac{\hat{R}(s,a)}{N(s,a)+1} + \sqrt{2}\sqrt{\frac{\log(N(s))}{N(s,a)+1}}$$

  Der Wert $\hat{R}(s,a)$ beschreibt dabei den durchschnittlichen Reward, der beim Folgen der Kante erlangt wurde. $N(s,a)$ gibt dagegen an, wie oft die Kante verwendet wurde. Steigt dieser Wert, so wird die Kante unattraktiver, wodurch andere Pfade bevorzugt werden und der Baum besser erkundet wird. Der rechte Summand beschreibt dabei wie oft die Kante vom Elternknoten aus verwendet wurde.
  Entlang des lukrativsten Knotens je Ebene, wird weitergesucht, bis ein Blatt, welcher keinen Terminalzustand darstellt, erreicht wurde.

- Expansion

  Der im vorherigen Schritt ausgewählte Knoten wird um einen oder mehrere Kindknoten erweitert.

- Simulation

  Von jedem der neu erzeugten Kindknoten wird eine Simulation gestartet. Von den jeweiligen Zuständen aus werden die Episoden zu Ende simuliert, dabei allerdings keine neuen Einträge im Baum erzeugt.

- Backpropagation

  In der letzten Phase der MCTS werden die Ergebnisse der Simulationen durch den Baum propagiert. Die Werte innerhalb der Knoten entlang zum Wurzelknoten werden anhand der neuen Werte aktualisiert. Anschließend startet eine weitere Iteration.

Das Verfahren terminiert nach einem vom Nutzer definierten Stoppkriterium. Nach Terminierung wird die vom Wurzelknoten aus am häufigsten gewählte Aktion ausgeführt und ein neuer Durchlauf startet für den Folgezustand.

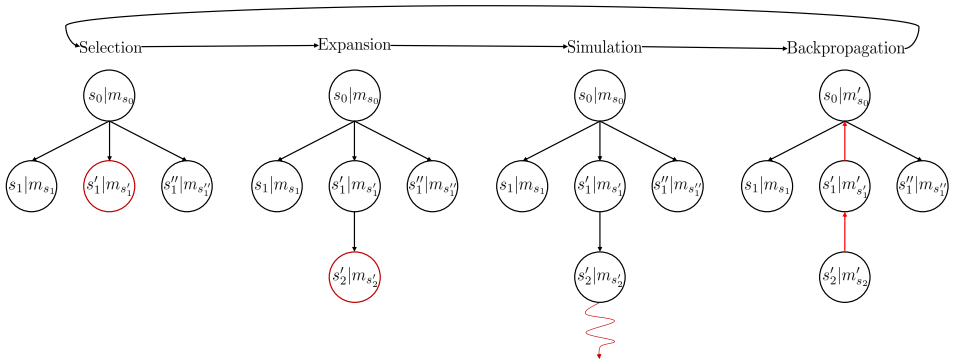


Abbildung 3.11: MCTS [24]: Der Ablauf ist in vier Phasen unterteilt. In der ersten Phase wird ein Knoten ausgewählt, an dem der Baum erweitert werden soll. in der zweiten Phase wird am gefunden Blattknoten ein neues Blatt erzeugt. Von diesem aus wird in der dritten Phase eine Simulation bis Episodenende simuliert. In der letzten Phase werden die Meta-Informationen der Knoten entlang des Pfades zur Wurzel aktualisiert. Anschließend startet eine neue Iteration.

Die MCTS findet insbesondere bei Brettspielen wie Schach oder Dame ihre Anwendung [19]. Zudem konnte das Verfahren erfolgreich mit Ideen des Reinforcement Learnings kombiniert werden und so Erfolge in komplexen Spielen wie Go erlangen [86].
Wir greifen die Ideen letzterer Entwicklungen auf, um die MCTS für die Offline-Variante des BPP verwenden zu können. Wir bauen dabei darauf auf, dass uns ein DDPG-Agent für das Online-Problem, bestehend aus Actor π mit Parametern θ^π und ein Critic Q mit Parametern θ^Q, zur Verfügung steht. Die Idee ist dabei wie folgt:
Die Wurzel des Baums stellt den Startzustand dar, aus dem wir die MCTS starten. Die Kindknoten der Wurzel ergeben sich aus den möglichen Packstücken, die dem Agenten im Startzustand zur Verfügung stehen. In der Selection-Phase wird einer dieser Knoten ausgewählt. Als Kriterium lässt sich die UCT wie folgt erweitern [6]:

$$UCT(s, a|\theta^\pi, \theta^Q) = UCT(s, a) + \frac{Q(s, \pi(s|\theta^\pi)|\theta^Q)}{N(s, a) + 1}$$

Wir addieren zusätzlich die Bewertung des Critics. Dadurch sollen Pfade, die laut dem Critic lukrativ sind, hervorgehoben werden. Der Q-Wert bewertet dabei die Qualität der Aktion, bevorzugt also lukrative Pfade.
Die Auswahl eines Kindknotens entspricht der Auswahl eines zu packenden Packstücks. Mit Hilfe des Online-Agenten wird von dem ausgewählten Kindknoten aus die Platzierung berechnet, woraus ein neuer Zustand bzw. eine neue Beladung entsteht.
Vom ausgewählten Kindknoten aus, gilt es nun diesen in der zweiten Phase zu expandieren. Dazu werden für jedes übrige Packstück jeweils ein Kindknoten erzeugt.

In der Simulation-Phase wird dann eine Episode zu Ende simuliert. Die Reihenfolge der Packstücke ist dabei zufällig.
Ist die Episode beendet, wird der erlangte Reward zur Wurzel durchpropagiert und die entsprechenden Werte aktualisiert. Anschließend folgt ein weiterer Durchlauf.
Für einen Packauftrag mit n Packstücken beinhaltet der maximal große Baum $n!$ Knoten. Da der damit aufkommende Rechenaufwand insbesondere für große Packaufträge nicht praktikabel ist, wird der Berechnungsaufwand vom Nutzer limitiert. Dazu kann beispielsweise ein Zeit- oder Knotenlimit genutzt werden.

3.4 Diskussion

Auf Basis der in Kapitel 2 vorgestellten Ideen, wurden in diesem Kapitel konkrete Verfahren und Implementierungen präsentiert. Anhand derer soll eine Realisierung eines Systems zur Lösung des in dieser Arbeit betrachteten Problems entwickelt werden können. Besondere Aufmerksamkeit galt beim Environment zum einen der Repräsentation einer Beladung. Hierbei helfen Intervallbäume zu einer effizienten und kompakten Darstellung. Zum anderen galt die Aufmerksamkeit der Reward-Funktion, da diese die entscheidende Schnittstelle zur Imitation des Experten darstellt. Die definierten Features sollen dabei helfen, das Verhalten eines Arbeiters zu beschreiben, auf Basis dessen das REIRL eine passende Reward-Funktion definieren soll.
Das damit definierte MEP dient als Grundlage des anschließenden Reinforcement Learnings. Der vorgestellte DDPG-Algorithmus soll in der Lage sein, empirisch eine Policy zu finden, welche die durch das REIRL definierte Reward-Funktion maximiert.
Es ist jedoch fraglich, wie erfolgreich sowohl die einzelnen Komponenten, als auch deren Zusammenspiel in Bezug auf die Problemstellung arbeiten. Dieses soll im Rahmen einer Evaluierung, die im folgenden Kapitel vorgestellt wird, untersucht werden.

Kapitel 4

Evaluierung

Im folgenden Kapitel wird die vorgestellte Methode evaluiert. Dazu wird zunächst in Kapitel 4.1 der Aufbau der Experimente beschrieben. Dies schließt neben den zu untersuchenden Kriterien zudem eine Beschreibung der verwendeten Daten ein. Anschließend werden in Kapitel 4.2 die Ergebnisse der Evaluierung präsentiert und interpretiert.

4.1 Aufbau

Ziel

Im Fokus der Evaluierung soll dabei die Fähigkeit zur Adaptivität der Methode stehen. Um diese überprüfen zu können, bedarf es verschiedener Packstrategien, um ein variables Verhalten der Methode feststellen zu können. Aus diesem Grund werden im Rahmen der Evaluierung drei unterschiedliche Packstrategien untersucht:

1. Strategie

   Bei der ersten Strategie soll die gesamte Fläche des Ladungsträgers genutzt werden, um die Packstücke möglichst gut verteilen zu können. Eine gleichmäßige Verteilung der Objekte soll dazu führen, dass zum einen das Gewicht über den Ladungsträger verteilt wird, aber auch die maximale Höhe gering bleibt.

2. Strategie

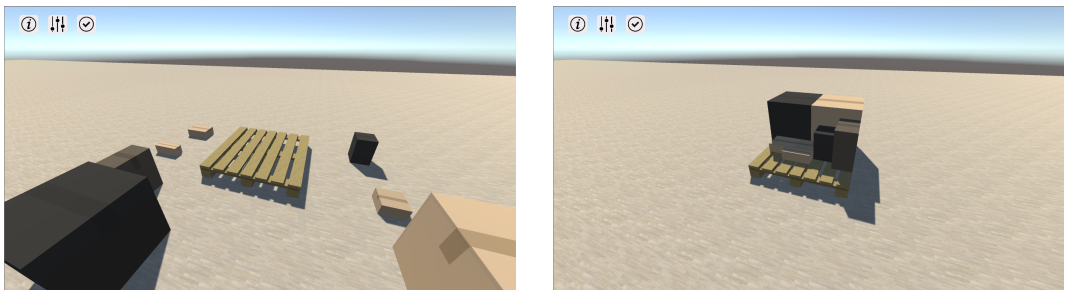
   Die zweite Strategie soll sich ähnlich verhalten wie die erste, mit dem Unterschied, dass ein kleiner Rand zu den Ladungsträgerkanten gelassen werden soll. Dies ist in der Praxis insbesondere dann interessant, wenn die Ladung nachträglich mit Polstermaterial geschützt werden soll. Ohne einen Rand würde diese über die Dimensionen des Ladungsträgers hinausgehen und so zu Problem bei der Lagerung führen.

3. Strategie

   Die dritte Strategie orientiert sich an der Diplomarbeit von Ruderman [82]. Hier war es das Ziel, einen Klassifikator zu konstruieren, der die Stabilität eines Paketstapels bewerten kann. Das Ziel ist es mit dieser Strategie eine Beladung zu erzeugen, die nach Rudermans Kriterien wünschenswert, also möglichst stabil, ist. Dabei ist allerdings anzumerken, dass, wie bereits in Kapitel 3.2 erwähnt, eine Palette als Gesamtheit bewertet wird. Ruderman dagegen nutzt darüber hinaus Features, die sich auf einzelne Packstücke beziehen. Die Ergebnisse sind also nur bedingt übertragbar.

Datenaufnahme

Um das Verhalten der Methode anhand der drei Strategien untersuchen zu können, bedarf es für jede der betrachteten Strategien geeignete Demonstrationsdaten. Diese wurden im Rahmen dieser Arbeit synthetisch erzeugt. Da es nach aktuellem Wissen des Autoren kein System gibt, welches Packvorgänge in Echtzeit aufzeichnen und in einem geeigneten Format ausgeben kann, wurde für die Datenaufnahme ein Pack-Simulator implementiert. Hierbei handelt es sich um eine grafische 3D-Anwendung, in welcher ein Nutzer virtuell Packaufträge verarbeiten kann (s. Abb. 4.1). Dazu wird diesem ein Ladungsträger in Form einer Palette und eine Auswahl an Packstücken zur Verfügung gestellt. Schließt der Nutzer den Packauftrag erfolgreich ab, platziert also alle Packstücke valide auf dem Ladungsträger, speichert die Anwendung das Resultat in Form einer XML-Datei (s. Abb. 4.2).
Der Simulator wurde genutzt, um die Demonstrationsdaten der Packstrategien zu erzeugen.



(a) Simulator zu Beginn eines Packauftrags.

(b) Simulator nach Beendigung eines Packauftrags.

Abbildung 4.1: Pack-Simulator: Der Nutzer der Software kann virtuell Packaufträge bearbeiten. Dazu lassen sich Packstücke per Drag-and-Drop auf dem Ladungsträger platzieren. Befinden sich alle Pakete auf der Palette, kann der Nutzer die Beladung bestätigen und damit abspeichern.

Während die Ladungsträgerdimensionen auf 100x100cm fixiert waren, wurden die Packstücke für jeden Auftrag neu gezogen. Je Auftrag wurde dabei zunächst die Anzahl der zu verpackenden Elemente gleichverteilt zwischen acht und zwölf Paketen gewählt. Hierbei handelt es sich um relativ kleine Aufträge. Damit soll die Komplexität, insbesondere in Hinblick auf das Trainieren von Agenten, vorerst gering gehalten werden. Neben der Anzahl der Packstücke müssen zudem deren Dimensionen gesampelt werden. Dazu orientiert wir uns an dem von Dowsland gegebenen Verhältnis zwischen Kantendimensionen [30]. Dies besagt, dass die Dimensionen eines Packstücks in der Praxis durch ein Verhältnis von 4:1 zueinander limitiert sind. Dazu wurde zunächst eine Länge gesampelt, die zweite in Abhängigkeit dieser und die dritte analog, sodass die Verhältnisse nicht überschritten wurden.
Auf diese Weise wurden je Packstrategie 100 Packaufträge erzeugt, bearbeitet und die entsprechenden Resultate extrahiert.

```
<Result xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
      <Meta>
            <OS>Windows 10  (10.0.0) 64bit</OS>
            <Date>08.08</Date>
            <Errors />
      </Meta>
      <Order>
            <orderSize>2</orderSize>
            <loadCarrierX>100</loadCarrierX>
            <loadCarrierY>100</loadCarrierY>
      </Order>
      <Placements>
            <BoxID>0</BoxID>
            <BoxSize>
                  <x>50</x>
                  <y>40</y>
                  <z>50</z>
            </BoxSize>
            <Weight>160</Weight>
            <PlacementPosX>0</PlacementPosX>
            <PlacementPosY>0</PlacementPosY>
            <PlacementPosZ>0</PlacementPosZ>
      </Placements>
      <Placements>
            <BoxID>1</BoxID>
            <BoxSize>
                  <x>40</x>
                  <y>30</y>
                  <z>30</z>
            </BoxSize>
            <Weight>100</Weight>
            <PlacementPosX>50</PlacementPosX>
            <PlacementPosY>0</PlacementPosY>
            <PlacementPosZ>0</PlacementPosZ>
      </Placements>
</Result>
```

Abbildung 4.2: Beispiel einer Ergebnis XML: Neben Meta-Informationen, die der Simulator zusätzlich speichert, finden sich in der Ergebnis-XML Informationen zum Ladungsträger und den Packstücken. Insbesondere beinhalten letztere ihre Platzierungen. In dem gezeigten Beispiel handelt es sich um lediglich zwei Pakete, die auf eine 100x100cm Ladungsträger nebeneinander platziert wurden.

Disziplinen

Die Evaluierung wird in drei Bereiche unterteilt:

1. Inverse Reinforcement Learning

   Das Ziel des IRL ist es, eine zu den Demonstrationsdaten passende Reward-Funktion zu finden. Im Rahmen der Evaluierung wollen wir also folgenden Fragestellungen nachgehen:

   - Sind die Besonderheiten der Strategien am Resultat erkennbar?
   - Eignen sich die vorgestellten Features zur Repräsentation einer komplexen Packstrategie?
   - Wie viele Demonstrationsdaten werden benötigt, um eine passende Reward-Funktion zu finden?

2. Reinforcement Learning - Online

   In diesem Teil der Evaluierung wollen wir uns dem Reinforcement Learning widmen. Dieses soll die im vorherigen Teil gewonnenen Reward-Funktionen nutzen, um darauf aufbauend eine geeignete Policy zu lernen. Dazu soll zunächst die Online-Variante des BPP untersucht werden. Die Fragestellungen sind dabei wie folgt:

   - Kann das DDPG erfolgreich eine Policy zur Lösung des BPP lernen?
   - Lassen sich damit unterschiedliche Strategien implementieren?

3. Reinforcement Learning - Offline

   Im letzten Teil der Evaluierung wollen wir die im vorherigen Abschnitt antrainierten Agenten mit der MCTS verbinden, um auch das Offline-BPP lösen zu können. Dazu wollen wir folgende Fragestellungen untersuchen:

   - Steigert sich die Leistung im Vergleich zur Online-Variante?
   - Wie gut skaliert das Verfahren?
   - Wie gut ist das Verfahren in Relation zu alternativen Algorithmen?

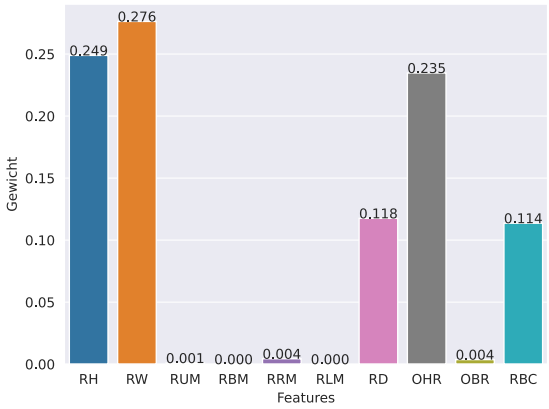
4.2 Ergebnisse

Inverse Reinforcement Learning

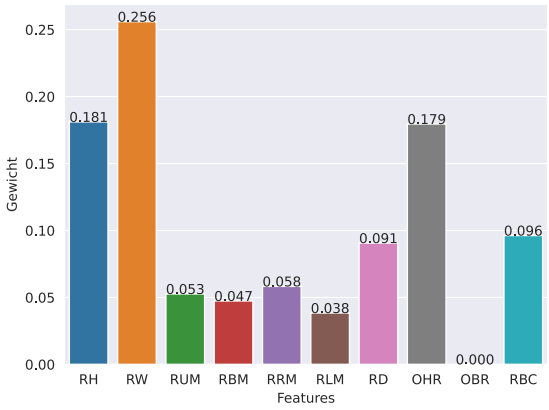
Um eine Reward-Funktion aus den vorhandenen Demonstrationsdaten zu generieren, wurde im Rahmen der Arbeit der Ansatz von Boularias et al. implementiert [16]. Je Packstrategie standen 100 Trajektorien zur Verfügung. Um das Verfahren nutzen können, bedarf es einer nicht-optimalen Policy für das verwendete Importance Sampling. Um diese zu generieren,

orientieren sich die Experimente dabei am Vorgehen, welches die Autoren angewandt haben. Dazu wird zunächst die vorhandene Datenmenge in zwei gleichgroße Mengen geteilt. Die Daten, die dann als nicht-optimale Policy verwendet werden sollen, werden manipuliert. Dazu wird eine demonstrierte Platzierung mit Wahrscheinlichkeit $\frac{1}{|\tau|}$ durch eine Zufällige ersetzt. Es wird angenommen, dass eine solche Platzierung in Hinblick auf die ausgeführte Packstrategie schlechter gewertet wird als die des Demonstrators. Dadurch lassen sich diese als nicht-optimale Daten verwenden. Die Gewichte der Features werden darüber hinaus auf 1 normiert. Dadurch, und dass die Features ebenfalls im Intervall von [0,1] liegen, resultiert daraus ein Reward im selben Bereich. In Hinblick auf das anschließende RL-Training, kann dies als sogenanntes *Reward Scaling* interpretiert werden, welches das Training von DDPG-Agenten verbessern soll [44]. Das Verfahren terminiert, wenn die Änderungen der Gewichte einen Threshold von $\mu = 1 - e4$ unterschreiten. Als Schrittweite wurde für das Subgradientenverfahren $\beta = 1 - e3$ verwendet.
Die Abbildung 4.3 zeigt die resultierenden Gewichte der jeweiligen Packstrategien. Auf der X-Achse sind dabei die Pack-Features aufgetragen, die Y-Achse gibt das jeweilige Gewicht des Features an. Diese ergeben sich aus dem Durchschnitt über 10.000 Ausführungen des REIRL.
Zu erkennen ist, dass bei allen drei Strategien sowohl die Höhen- als auch die Gewichtsverteilung (RH und RW) einen hohen Stellenwert haben, was auch wünschenswert ist. Auch haben die Werte von Overhang Ratio (OHR), Relative Density (RD) und Relative Barcenter (RBC) in allen drei Strategien einen nennenswerten Anteil.
Im Vergleich zu Strategie 1, ist bei Strategie 2 die Relevanz des Randes erkennbar. Die vier Features, die diesen beschreiben (RUM, RBM, RRM, RLM), zeigen jeweils ein etwa gleiches Gewicht, was für eine Gleichmäßigkeit des Randes in den Demonstrationsdaten spricht. Dass der Rand im Gegensatz zu Strategie 1 erkannt wurde, spricht für die angewandte Methode.
Die dritte Strategie stellt die komplexeste dar. Ruderman nutze dabei insbesondere die Relative Density (RD), Overbuid Ratio (OBR) und Relative Barycenter (RBC) um die Stabilität einer Beladung zu bewerten. Die drei Werte gilt es dabei möglichst zu maximieren. An den Ergebnissen ist zu erkennen, dass vor allem die OBR im Vergleich zu den beiden anderen Strategien Gewicht bekommen hat. Auch das RBC kann einen höheren Anteil aufweisen. Lediglich die RD ist geringer als in den anderen beiden Experimenten. Wünschenswert wäre, dass hier diese auf Kosten der RW oder RH höher ausfallen würde. Dennoch lässt sich sagen, dass sich die Kerneigenschaften der Strategien grundsätzlich in den Gewichten ablesen lassen. Hier profitieren wir von der einfachen Darstellung der Reward-Funktion. Auf Kosten der Interpretierbarkeit ließen sich hier alternativ komplexere Darstellungen, z.B. mittels neuronaler Netze, implementieren. Der Autor sieht dafür jedoch keinen Grund, da dabei insbesondere die subjektive Bewertung und Interpretation

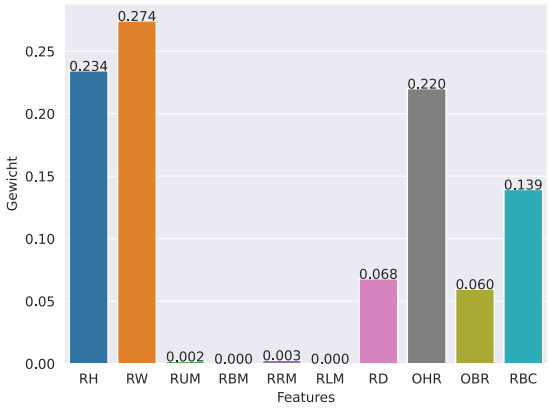
erschwert werden würde.



(a) 1. Strategie



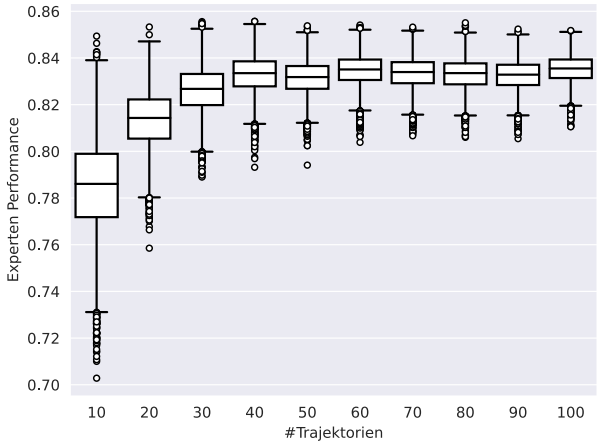
(b) 2. Strategie



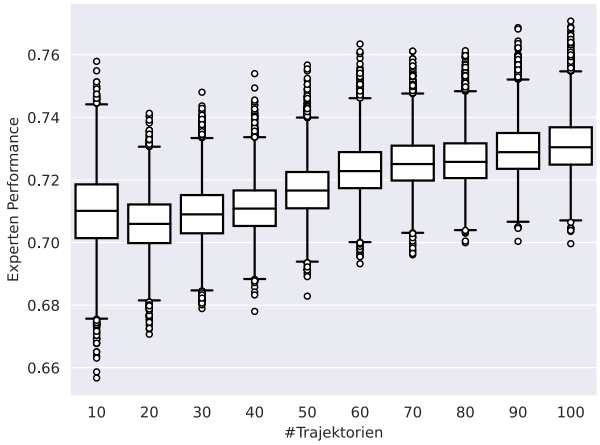
(c) 3. Strategie

Abbildung 4.3: IRL Gewichte: Resultierende Gewichte nach Ausführung des REIRL je Strategie.

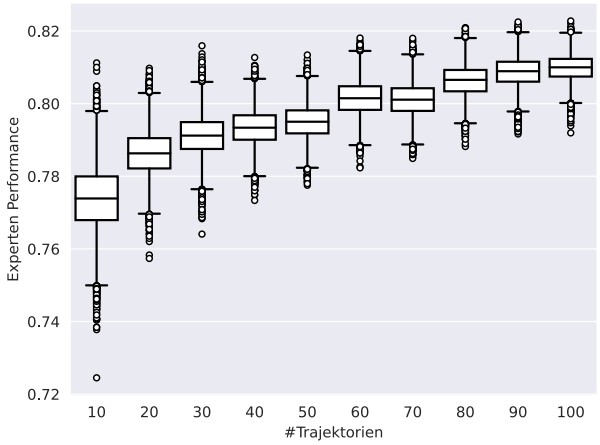
Insbesondere für mögliche Anwender einer solcher Methode, ist es interessant zu wissen, wie viele Daten notwendig sind, um eine adäquate Reward-Funktion generieren zu können. Die Abbildung 4.4 zeigt die Ergebnisse eines Experiments, welches dieser Fragestellung nachgeht. Im Rahmen dessen wurden die Datenmengen reduziert. Die X-Achse gibt dabei an, wie viele der vorhandenen Trajektorien genutzt wurden. Die angegebene Menge wurde dabei gleichverteilt aus der Gesamtheit gezogen. Für jede Teilmenge wurden 10.000 Experimente durchgeführt. Auf der Y-Achse ist die durchschnittliche Experten-Performance aufgetragen, welche sich aus der Bewertung der Gesamtheit der Trajektorien anhand der gefundenen Reward-Funktion ergibt.
Die Box-Plots (oder auch Whisker-Plots) zeigen die Verteilungen der Ergebnisse. Bei der ersten Strategie ist eine starke Streuung bei geringen Datenmengen zu erkennen. Erhöhen wir diese, steigt zum einen die durchschnittliche Performance des Experten, zum anderen reduzieren wir zudem die Streuung. Die Strategien 2 und 3 zeigen in der Tendenz ein ähnliches Bild, jedoch ist insbesondere bei Strategie 2 die Streuung auch bei hohen Datenmengen größer als bei den anderen beiden Strategien. Zudem lässt sich hier erkennen, dass die Strategie 1 nicht nur eine höhere Performance als die anderen beiden Strategien erlangt, sondern dafür auch weniger Daten benötigt.
Dennoch zeigen die Experimente, dass das Ergebnis generell mit steigender Datenmenge präziser wird und die durchschnittliche Leistung des Experten steigt. So zeigt die Abbildung 4.5 zusätzlich die durchschnittliche Varianz der Gewichte und Abbildung 4.6 die durchschnittliche Performance des Experten. Diese verdeutlichen die sinkende Varianz und steigende Leistung mit höherer Datenmenge.
Die genauen Zahlen und Konfigurationen der Experimente finden sich im Anhang (s. Anh. A.2). Auf den ermittelten Reward-Funktionen aufbauend, können wir nun untersuchen, ob das Lernen einer Policy möglich ist.



(a) 1. Strategie

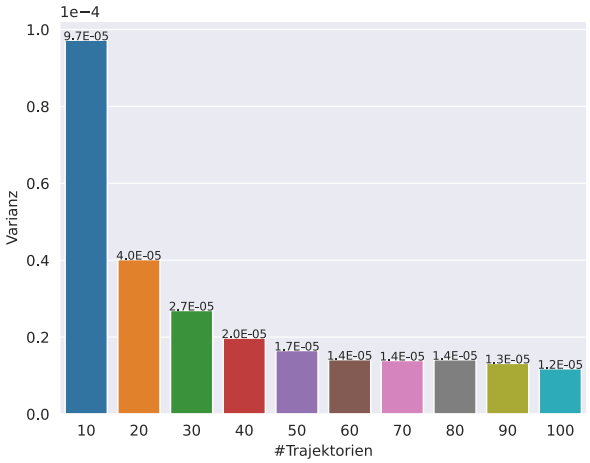


(b) 2. Strategie

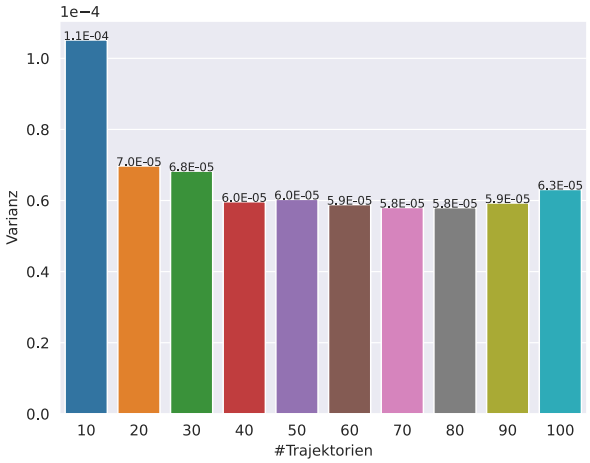


(c) 3. Strategie

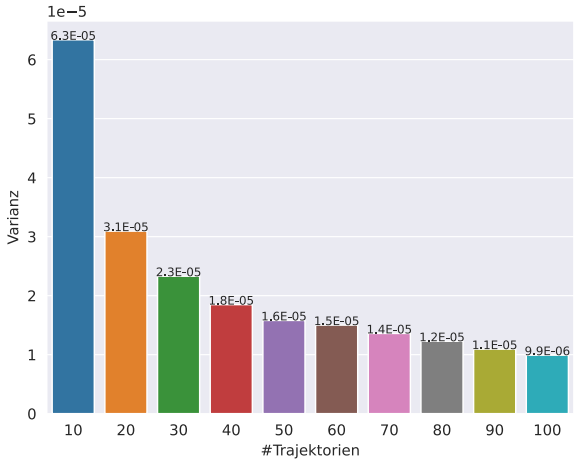
Abbildung 4.4: IRL Performance Box-Plots: Die Plots zeigen die Performance über verschieden große Datenmengen beim REIRL. Zur besseren Visualisierung des Verlaufs wurden hier unterschiedliche Skalen verwendet.



(a) 1. Strategie

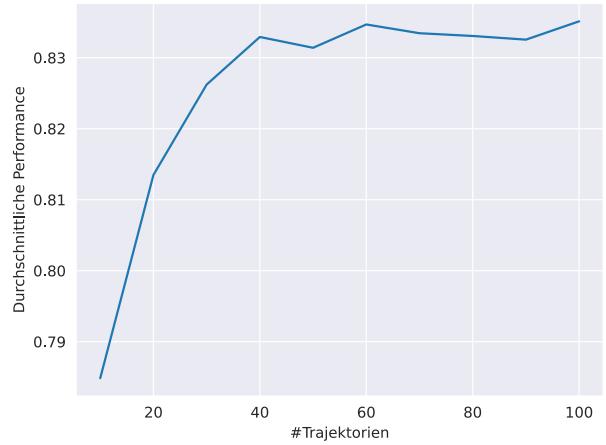


(b) 2. Strategie

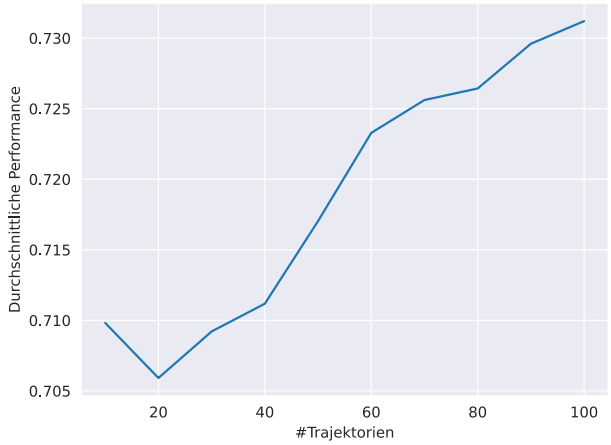


(c) 3. Strategie

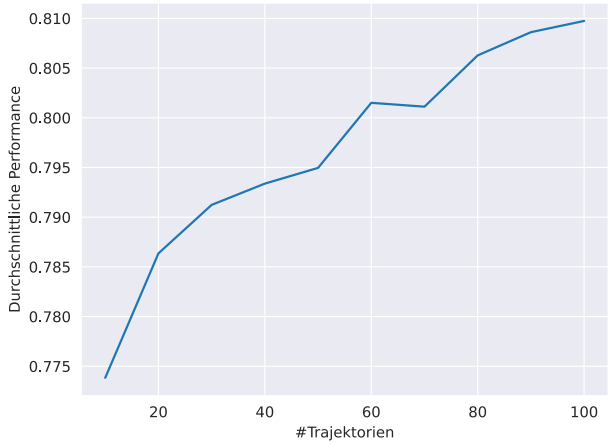
Abbildung 4.5: IRL Gewichts-Varianz: Die Plots zeigen die durchschnittliche Varianz der Gewichte mit steigender Datenmenge. Auch hier wurden unterschiedliche Skalen zur Darstellung verwendet.



(a) 1. Strategie



(b) 2. Strategie



(c) 3. Strategie

Abbildung 4.6: IRL Experten-Performance: Die Plots zeigen die durchschnittliche Experten-Performance über verschieden große Datenmengen. Zu beachten sind auch hier die unterschiedlichen Skalen.

Reinforcement Learning - Online

Aufbauend auf den durch das REIRL gewonnenen Reward-Funktionen, lassen sich nun Agenten trainieren, welche diese maximieren sollen. Wie in Kapitel 3.3 beschrieben, werden hierfür DDPG-Agenten trainiert. Der Erfolg des Trainings ist dabei stark von den freien Hyperparametern abhängig. Diese gilt es so zu wählen, dass das Training möglichst effektiv und konvergent verläuft. Da eine analytische Bestimmung der Parameter nicht praktikabel ist, werden hier heuristische Verfahren zur Hyperparameter-Wahl herangezogen. Man spricht hierbei von *Hyperparameter-Tuning*. Hierzu werden Konfigurationen gewählt, die entsprechenden Trainings durchgeführt und eine Auswahl anhand der Resultate getroffen. Im Rahmen der Evaluierung wurde dazu für jede der Strategien ein separates Tuning durchgeführt. Dazu wurden jeweils 120 Konfigurationen erzeugt, die in Summe elf Parameter variiert haben. Das Tuning wurde im Rahmen des Frameworks *Tune* durchgeführt [60]. Die Konfigurationen der Parameter wurden mit Hilfe des Suchalgorithmus *HyperOpt* gewählt [12]. Die optimalen Parameter werden hierbei auf Basis einer baumbasierten Parzen-Schätzung geschätzt [11]. Eine Übersicht der Hyperparameter, deren betrachtete Wertebereiche und die durch das Hyperparameter-Tuning gefundenen Konfigurationen zeigt Tabelle 4.1. Diese werden im Folgenden näher betrachtet.
Eine Episode entspricht dabei einem Packauftrag und wird mit jeder Iteration neu gezogen, um das Verhalten über möglichst verschiedene Instanzen untersuchen zu können. Dabei werden dieselben Bedingungen, wie bereits im vorherigen Abschnitt beschrieben, verwendet. Die Wahl der Packaufträge hat dabei einen großen Einfluss auf die Leistung des Agenten. Da es sich um einen stochastischen Prozess handelt und eine für den Agenten unglückliche Wahl der Packaufträge die Performance stark beeinträchtigen kann, kann die Leistung nicht an einem Durchlauf bemessen werden [44]. Aus diesem Grund wurden zu jeder Parameter-Konfiguration fünf Wiederholungen durchgeführt. Die finalen Resultate ergeben sich dabei aus dem Durchschnitt der Wiederholungen.
Die internen Rollen des DDPG-Agenten werden mit Hilfe neuronaler Netze realisiert. Bei diesen spielt insbesondere die Architektur eine entscheidende Rolle in Bezug auf die Leistung des Agenten. Eine umfangreiche und ausgiebige Suche nach einer individuellen Architektur würde über den Rahmen dieser Arbeit hinausgehen. Für das Training wurde daher eine Architektur, die sich an der des ursprünglichen Papers orientiert, gewählt [61]. Diese hat über verschiedene Disziplinen zufriedenstellende Leistungen gezeigt, stellt also einen guten Ausgangspunkt dar. Das Actor-Netz verwendet dabei zur Auswertung eines Zustands drei Faltungsschichten, von denen jede 32 Filter aufweist. Die erste Schicht implementiert einen 8x8 Filter mit einer Stride von 4, die zweite einen 4x4 Filter mit einer Stride von 2 und die letzte einen 3x3 Filter mit einer Stride von 1. In allen drei Schichten wird auf Padding verzichtet. Gefolgt werden die Faltungsschichten von zwei versteckten Schichten, dessen Anzahl von Neuronen im Rahmen der Evaluierung variiert wurden. Zudem

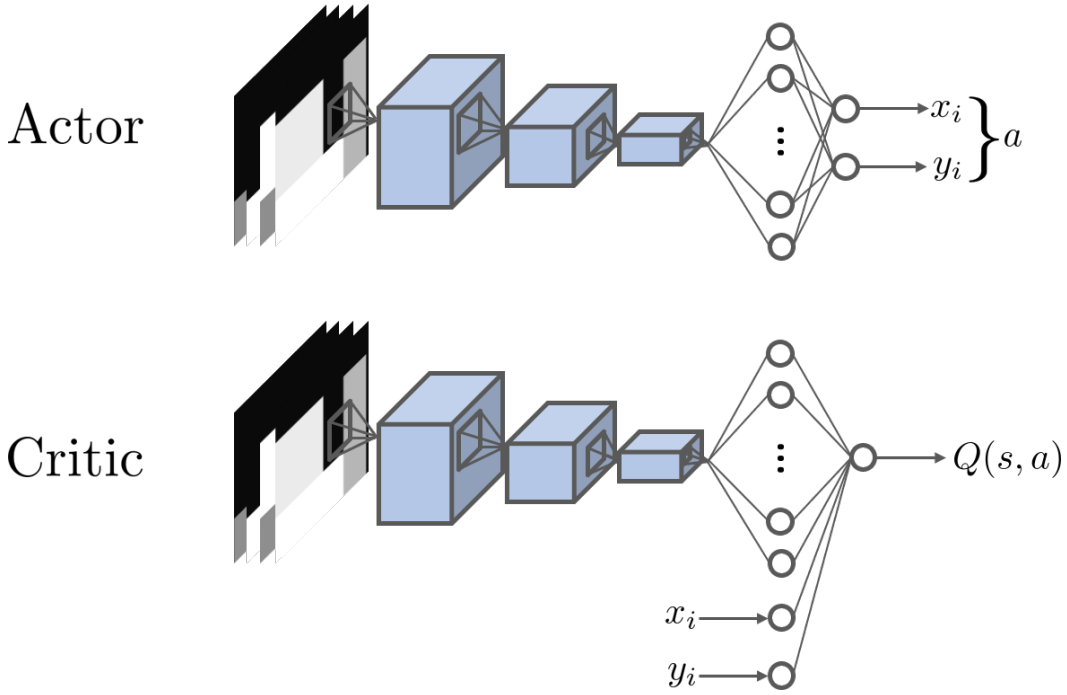


Abbildung 4.7: DDPG Netzwerk-Architektur: Die zwei Netzwerke folgen demselben Schema, mit dem Unterschied, dass der Critic zusätzlich den Output des Actors in den internen Schichten des MLPs verarbeitet.

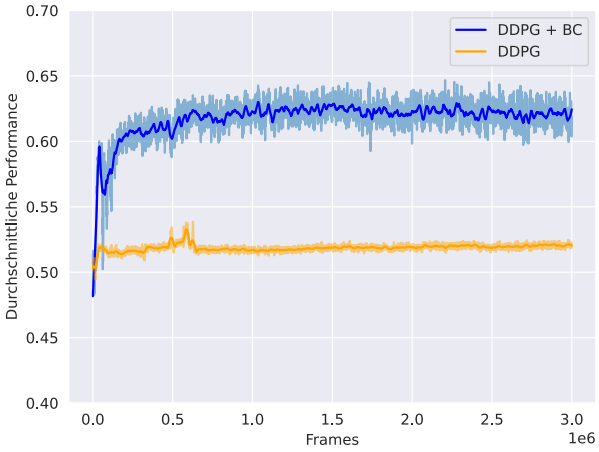
wurden die in den Neuronen verwendeten Aktivierungsfunktionen variiert. Die Ausgabe erfolgt über lediglich zwei Ausgangsneuronen, jeweils für die X- und Y-Koordinaten der Ausgabe. Diese wurden mit Hilfe einer tanh-Aktivierungsfunktion jeweils auf einen Bereich von [-1,1] begrenzt. Die entsprechende Z-Koordinate einer Platzierung ergibt sich, wie in Kapitel 3.3.1 beschrieben, direkt über das Geometriemodell.
Das Critic-Netz ist identisch zum Actor-Netz aufgebaut, mit dem Unterschied, dass der Output der Faltungsschichten mit der Aktion des Actors, repräsentiert durch einen zweidimensionalen Vektor, konkateniert wird. Eine Visualisierung der Netzarchitekturen zeigt Abbildung 4.7.
Das Training der Netze wurde mit Hilfe des Adam-Optimierers realisiert [53]. Hierbei gilt es sowohl für den Actor, als auch für den Critic die Lernrate einzustellen. Darüber hinaus wurde im Rahmen der Evaluierung die L_2-Regularisierung variiert, welche Overfitting auf den Demonstrationsdaten verhindern soll [97][45]. Die Buffer-Größe je Experiment ist auf 10^6 beschränkt, welche sich aus dem maximal verfügbaren Speicher der Evaluierungsplattform ergibt. Die Größe des Batches, die aus dem Buffer während des Trainings gezogen wird, wurde im Rahmen der Evaluierung variiert.
Neben der Netzwerke und deren Training, verfügt auch der DDPG-Algorithmus selbst über zu wählende Parameter. Dazu zählen neben der Gewichtungen der Loss-Funktionen λ_1 und λ_2 darüber hinaus die Frequenz, in welcher Policy- und Target-Netze synchronisiert wer-

Parameter	Wertebereich	Strategie 1	Strategie 2	Strategie 3
Batch Size	$\{16, 32, 64\}$	16	128	32
Lernrate Actor	$\{1e-3, 1e-4, 1e-5, 1e-6\}$	$1e-4$	$1e-3$	$1e-4$
Lernrate Critic	$\{1e-3, 1e-4, 1e-5, 1e-6\}$	$1e-3$	$1e-6$	$1e-4$
Hidden Units Actor	$\{128, 256, 512, 1024\}$	128	256	512
Hidden Units Critic	$\{128, 256, 512, 1024\}$	256	128	128
Aktivierungsfunktion	{sigmoid, relu, leakyrelu}	leakyrelu	relu	leakyrelu
L_2-Regularisierung	$\{1e-3, 1e-4, 1e-5\}$	$1e-4$	$1e-5$	$1e-4$
λ_1	[0,1]	0.53	0.20	0.15
λ_2	[0,1]	0.66	0.05	0.61
Sync.-Frequenz	$\{1000, 10000\}$	1000	1000	1000
Noise	$[0.1, 0.5]$	0.244	0.307	0.3

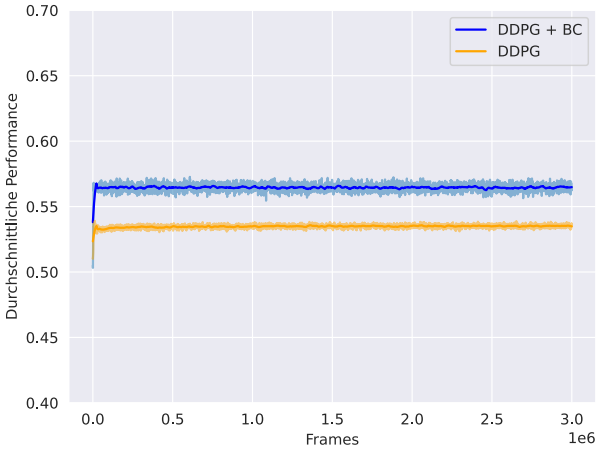
Tabelle 4.1: RL-Evaluations-Parameter: Die Tabelle zeigt die im Rahmen der Evaluierung variierten Hyperparameter, deren jeweiligen gewählten Wertebereiche und die jeweiligen Konfigurationen, die in Kapitel 4 vorgestellt wurden.

den. Hierzu wurde ein festes $\rho = 0.999$ aus dem DDPG-Paper übernommen. Auch der Diskontierungsfaktor von $\gamma = 0.99$ entstammt den Experimenten der Autoren. Das initiale ϵ dagegen wurde variiert. Dieses wird über ein festes Zeitfenster von 1 Mio. Frames auf 0 reduziert, wobei ein Frame einem als Bild verarbeiteten Zustand entspricht. Das Training des Agenten ist dabei auf 3 Mio. Frames pro Durchlauf limitiert.
Die verwendete Implementierung, sowie die Wahl der Parameter und derer Wertebereiche orientiert sich an der DDPG-Implementierung der *OpenAI-Baselines* [29].
Die Abbildungen 4.8 zeigen die Performance während des Trainings. Auf der X-Achse sind die Frames aufgetragen. Die Y-Achse gibt die durchschnittliche Performance des Agenten zum jeweiligen Frame an. Die jeweils blauen Graphen zeigen die Performance des DDPG inklusive des Behaviour Clonings auf den Demonstrationsdaten. Abgebildet ist dabei die Performance der Parameterkonfiguration, die über die fünf Wiederholungen die beste durchschnittliche Leistung gezeigt hat. Gemessen wurde dies anhand der Leistung der jeweils letzten 100 Packaufträge bzw. Episoden. Die orangene Kurve zeigt dagegen die Leistung derselben Konfiguration über fünf Wiederholungen, jedoch ohne Verwendung des Behaviour Clonings.
Die Abbildungen 4.9 zeigen die dem Training entsprechenden Verläufe der Loss-Funktionen. Hier wird jedoch aus Gründen der Übersicht auf die BC-freie Variante verzichtet. Diese können dem Anhang entnommen werden (s. Anh. A.3).
Für die Strategien 1 und 3 sind deutliche Lerneffekte im Vergleich zum Beginn des Trainings erkennbar. Mit steigender Interaktion mit dem Environment steigt auch die entsprechende Leistung des Agenten. In Hinblick auf die BC-freien Kurven lässt sich hierbei vermuten, dass insbesondere das BC einen großen Einfluss auf das Lernen der Agenten hat. Auch zeigen die jeweiligen Loss-Kurven ein konvergentes Verhalten. Lediglich der Actor-Loss

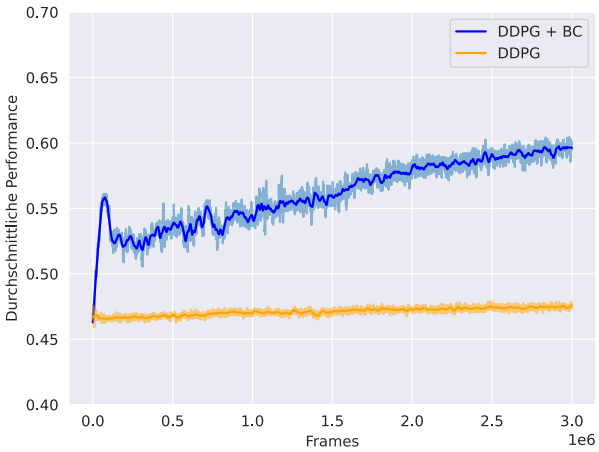
scheint in beiden Strategien nicht gegen Null, sondern gegen einen Wert nahe diesen zu laufen. Dies kann daher kommen, dass in beiden Konfigurationen $\lambda_2 > \lambda_1$ gilt und somit der BC-Loss dominiert.
Ein anderes Bild zeichnet sich dabei bei Strategie 2 ab. Hier erzielt zwar auch die Variante inklusive BC die bessere Performance, jedoch ist hier kein Lerneffekt über das Training erkennbar. Dies kann darauf hindeuten, dass sich der Agent in einem lokalen Optimum festgelaufen hat. Eine besondere Entwicklung zeigt hier der Actor-Loss: dieser scheint zum Ende des Trainings zu divergieren. Dies kann auf eine mögliche Entwicklung aus dem vermeidlichen lokalen Optimum hindeuten.
Generell zeigte sich, dass die Auswahl der Parameter in allen Strategien großen Einfluss auf die Performance des Agenten hat. So zeigten sich insbesondere bei der Wahl der Aktivierungsfunktion merkbare Performance-Unterschiede. Sowohl durch die leakyrelu- als auch die relu-Funktionen konnte das Training im Vergleich zur klassischen sigmoid-Funktion beschleunigt werden. Dies könnte auf das Sättigungsproblem zurückzuführen sein, welches in tiefen Netzen bei derartigen Funktionen auftreten kann [47]. Zudem favorisierten erfolgreiche Trainingsläufe eine eher geringe Zahl versteckter Neuronen. Dies könnte auf ein Overfitting durch zu viele freie Parameter bei einer hohen Neuronenanzahl hindeuten. Eine zu hohe Lernrate der Netzwerke wirkte sich hierbei nicht negativ auf den Lernprozess aus, führte also zu keiner Divergenz. Interessant wäre hier das Verhalten noch höherer Lernraten zu untersuchen. Eine hohe Gewichtung des BC-Losses λ_2 wirkte sich positiv auf das Training aus, da dadurch die Demonstrationsdaten stärker in den Trainingsprozess einfließen konnten. Dies führt schneller zu einer der Strategie entsprechenden Policy. Die häufigere Synchronisation der Netze, mit einem Update alle 1.000 Frames, konnte zudem das Training besser stabilisieren, als die Variante mit einem Update alle 10.000 Frames. Auch eine geringe Action-Noise wirkte sich eher positiv auf das Training aus. Eine hohe Wahrscheinlichkeit für Rauschen sorgte dabei dafür, dass der Buffer mit vielen zufälligen Aktionen gefüllt wurde. Dies ist insbesondere dann nicht nötig, wenn Demonstrationsdaten zur Verfügung stehen, die die Auswirkungen des Exploration-Exploitation-Dilemmas durch Aufzeigen erfolgreicher Pfade reduzieren.



(a) 1. Strategie

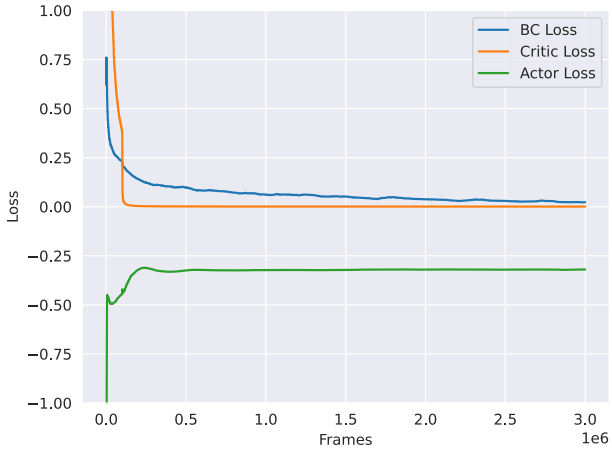


(b) 2. Strategie

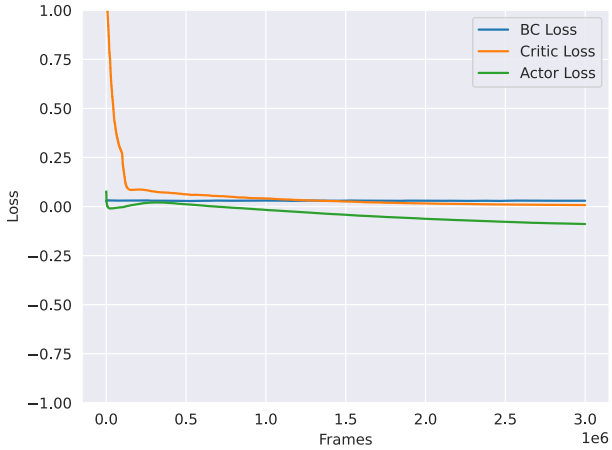


(c) 3. Strategie

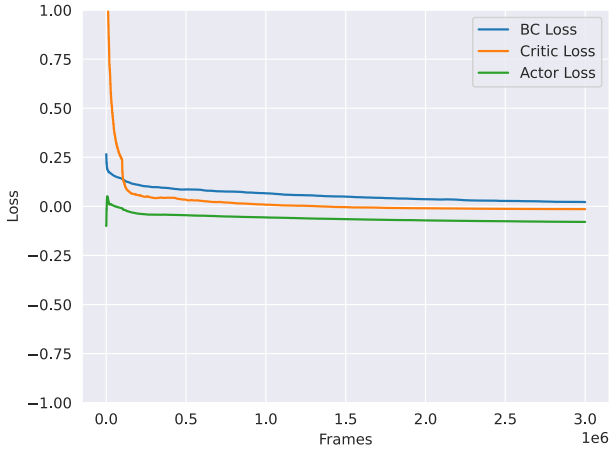
Abbildung 4.8: RL-Online-Performance: Auf der X-Achse sind die Frames des Trainings aufgetragen. Die Y-Achse gibt den durchschnittlichen Reward an, den der Agent zu einem bestimmten Frame erzielt hat. Die jeweils dunklere Linie zeigt die mit einem Savitzky-Golay-Filter verarbeiteten Performance [84]. Dies soll den Trend des Trainings besser veranschaulichen.



(a) 1. Strategie



(b) 2. Strategie



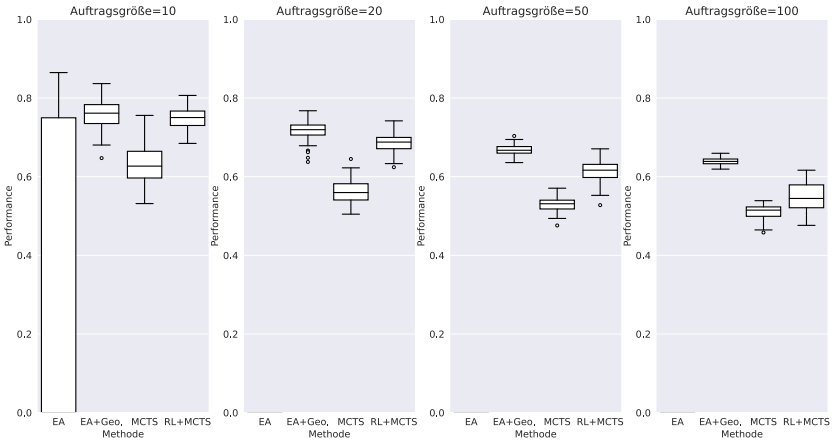
(c) 3. Strategie

Abbildung 4.9: RL-Online-Losses: Auf der X-Achse sind die Frames des Trainings aufgetragen. Die Y-Achse gibt die Entwicklung der drei Loss-Funktionen über die Dauer des Trainings an.

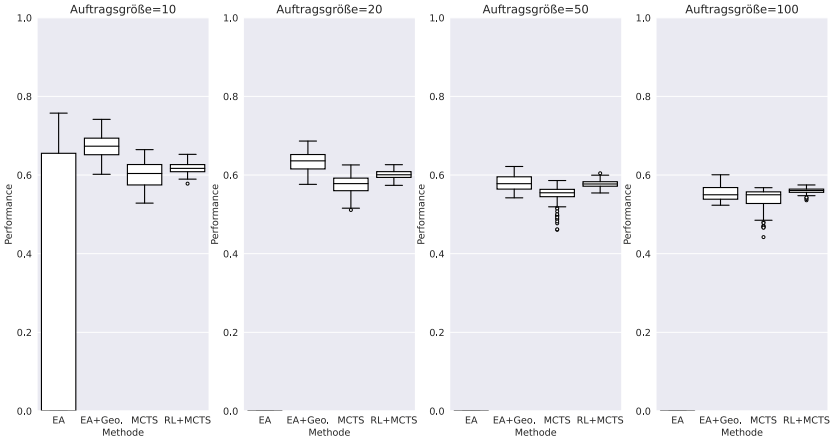
Reinforcement Learning - Offline

Die im vorherigen Abschnitt trainierten Agenten können nun genutzt werden, um eine Variante für das Offline-BPP, wie in Kapitel 3.3.2 beschrieben, zu realisieren. Dazu werden die trainierten Netze im Rahmen einer angepassten MCTS verwendet. Zum Einsatz kommt dabei je Strategie der leistungsstärkste Agent der vorangegangenen Experimente. Um das Verfahren vergleichen zu können, wurden zudem alternative Verfahren implementiert und evaluiert.
Die erste Alternative stellt eine einfache MCTS dar. Diese verzichtet auf die zusätzliche Orientierungshilfe durch die neuronalen Netze. Die Auswahl der Pakete und deren Platzierung werden hierbei zufällig gewählt.
Neben der MCTS wurde darüber hinaus eine Lösung in Form eines $(\mu+\lambda)$-**Evolutionären Algorithmus** (EA) implementiert, auf Basis dessen Optimierungsprobleme heuristisch gelöst werden können. Als Grundlage dient dabei das in Kapitel 1.3 definierte Bin-Packing-Optimierungsproblem. Die zu maximierende Funktion entspricht der Reward-Funktion des IRL. Die Individuen entsprechen gesamten Beladungen, dargestellt über die jeweiligen Koordinaten der Packstücke. Der EA mutiert die Beladungen einer Generation, indem er die Packstücke jeweils normalverteilt $\sim \mathcal{N}(0, 1)$ verschiebt. Für die Evaluierungen wurden $\mu = 10$ und $\lambda = 20$ festgelegt. In einer intensiveren Optimierung können diese Werte angepasst werden, wird jedoch hier aus Gründen der Einfachheit verworfen. Werden die durch das Bin-Packing-Problem gestellten Bedingungen verletzt, wird in Form einer Straffunktion die Fitness eines Individuums reduziert. Pro Verletzung wird dabei der Reward um 1 reduziert. Dadurch erhalten nicht valide Beladungen automatisch einen Reward ≤ 0 und sind somit immer schlechter als valide Varianten.
Eine vierte Alternative erweitert den $(\mu + \lambda)$-EA um die Nutzung des Geometriemodells. Statt ein Individuum für eine nicht valide Beladung zu bestrafen, wird stattdessen die Beladung mit Hilfe des Geometriemodells korrigiert. Überschneiden sich zwei Packstücke, wird mit Hilfe des Modells die Höhe eines der betroffenen Packstücke erhöht, bis die Packstücke nicht mehr überlappen. Auch werden über den Ladungsträger ragende Packstücke so weit verschoben, dass diese sich vollständig auf diesem befinden. Diese Korrekturen führen dazu, dass innerhalb einer Generation des EA ausschließlich valide Beladungen betrachtet werden. Dies soll zu mehr erfolgreichen Individuen führen.
Es sei hierbei zu erwähnen, dass sich sowohl die MCTS, als auch der EA parallelisieren lassen [25][88]. Dadurch lassen sich auf entsprechender Hardware mehr Auswertungen in derselben Zeit ausführen, was zu besseren Ergebnissen führen kann. Darauf wurde jedoch im Rahmen der Evaluierung zum Zwecke der besseren Vergleichbarkeit der Methoden verzichtet.
Zudem wurde hier auf das Auswerten der Netzwerke auf einer GPU aus demselben Grund verzichtet. Die Experimente wurde alle auf derselben Umgebung ausgeführt, um möglichst

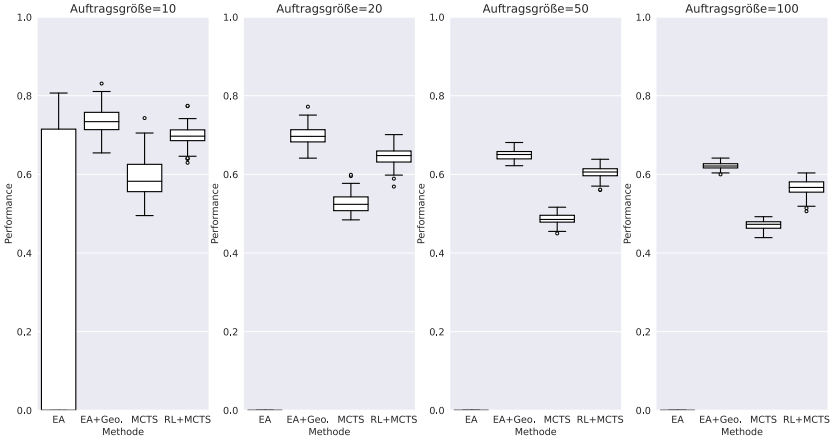
vergleichbare Resultate zu erhalten. Als Prozessor kam dabei ein Intel Xeon Prozessor E5-2643 mit einer maximalen Taktfrequenz von 3,70 GHz zum Einsatz. Der RAM war mit 64 GB bemessen und stellte für keines der Experimente eine Limitierung dar.
Auch in diesem Teil der Evaluierung werden wieder die drei Strategien separat untersucht. Die Abbildung 4.10 zeigt die Ergebnisse der Versuche. Zu jeder Strategie wurden dabei jeweils vier Packauftragsgrößen untersucht. Jeder Algorithmus berechnete dazu zu jeder Größe je 100 Packaufträge. Je Packauftrag wurde dabei die Zeit je Auftrag limitiert. Bei einer Größe von 10 Packstücken hatten die Algorithmen eine Minute, bei 20 Packstücken je zwei Minuten, bei 50 Packstücken je fünf Minuten und bei 100 Packstücken je zehn Minuten zur Verfügung.
Bei allen drei Strategien ist dabei ein ähnliches Bild zu erkennen. Die MCTS mit Hilfe der trainierten Agenten zeigt hierbei zum einen, dass der zusätzliche Input des Netzwerks zu einem Leistungsanstieg gegenüber der naiven MCTS führt und zum anderen, dass die Leistung im Vergleich zu den Online-RL-Agenten der vorherigen Evaluierung (s. Abb. 4.8) ansteigt. Aus letzterer Beobachtung lässt sich also folgern, dass sich die zusätzliche Freiheit in Bezug auf die Reihenfolge positiv auf die Packleistung auswirkt und damit die zusätzliche kombinatorische Komplexität überwiegt. Zwar sind die Leistungen gemessen am Median schlechter als die des EAs mit Geometriemodell, jedoch zeigt die MCTS mit RL-Agent häufig Ergebnisse mit geringster Streuung, insbesondere bei kleinen Auftragsgrößen.
Der naive EA hat bei einer Auftragsgröße von 10 eine große Streuung. Dabei wurden Aufträge, die aufgrund der Straffunktion negative Rewards zur Folge hatten, aus Gründen der besseren Darstellung, auf 0 abgebildet. Für größere Aufträge scheitert der naive EA jedoch und findet in der gegebenen Zeit keine validen Beladungen.
Der EA mit Unterstützung des Geometriemodells zeigt dagegen ein anderes Bild. So zeigt dieser, am Median gemessen, in allen Versuchen die beste Leistung. Darüber hinaus ist erkennen, dass mit steigender Auftragsgröße die Spannweite der Ergebnisse sinkt. Einen Vergleich dazu zeigt Tabelle 4.2. Hier ist zu erkennen, dass je größer die Aufträge sind, desto geringer wird die Spannweite der RL+MCTS, aber insbesondere auch des EAs mit Geometriemodell.
Es lässt sich zudem sagen, dass, bis auf den naiven EA, die Algorithmen angemessen skalieren. Mit steigender Auftragsgröße sind keine starken Leistungseinbrüche erkennbar, was für die Verfahren spricht.



(a) 1. Strategie



(b) 2. Strategie



(c) 3. Strategie

Abbildung 4.10: Offline-BPP-Performance: Die Boxplots bilden die Performance der verschiedenen Varianten ab. Dabei wurden für verschiedene Packauftragsgrößen jeweils 100 Versuche durchgeführt.

		Auftragsgröße			
		10	20	50	100
EA+Geo	Strategie 1	0.189	0.130	0.068	0.04
	Strategie 2	0.140	0.110	0.08	0.077
	Strategie 3	0.176	0.13	0.05	0.04
	∅	**0.168**	**0.123**	**0.066**	**0.050**
RL+MCTS	Strategie 1	0.122	0.118	0.143	0.140
	Strategie 2	0.074	0.053	0.050	0.039
	Strategie 3	0.140	0.132	0.07	0.09
	∅	**0.122**	**0.101**	**0.087**	**0.086**

Tabelle 4.2: Offline Spannweiten: Die Tabelle zeigt die Spannweiten von EA+Geo. und RL+MCTS im Rahmen der Experimente und deren Durchschnitt über die evaluierten Strategien.

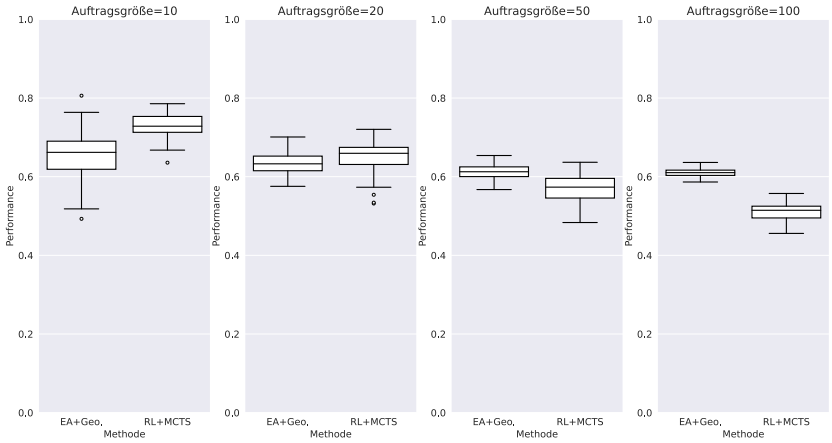
Es wurden darüber hinaus weitere Experimente durchgeführt, um das Verhalten des EAs mit Geometriemodell und der MCTS mit RL-Agent unter einem reduzierten Zeitlimit zu untersuchen. Je Auftrag standen den Algorithmen dabei jeweils ein Zehntel der Zeit der vorangegangenen Versuche zur Verfügung. Die Strategien und Auftragsgrößen sind dagegen gleich geblieben. Auch hier wurden je Einstellung 100 Wiederholungen durchgeführt. Durch das reduzierte Zeitlimit können beide Verfahren weniger Iterationen bzw. Generationen durchlaufen. Wie zu erwarten, aber durch die Experimente untermauert wird, sinkt die durchschnittliche Leistung der beiden Algorithmen durch die strengere Limitierung (s. Tabelle 4.3).
Die Abb. 4.11 zeigt die Verteilung der Experimente. Zu erkennen ist, dass die RL+MCTS-Kombination insbesondere bei geringen Auftragsgrößen eine am Mittelwert gemessen bessere Leistung im Vergleich zum EA zeigt. Auch ist hier die Spannweite der Ergebnisse deutlich geringer (s. Tabelle 4.4). Wird die Auftragsgröße erhöht, nimmt dagegen die Leistung der MCTS stärker ab als die des EAs. Auch reduziert sich die Spannweite des EAs, wodurch die Ergebnisse bei großen Aufträgen stabiler wirken.
Dies kann darauf zurückzuführen sein, dass die Repräsentation mittels des MCTS-Baums mit steigender Auftragsgröße deutlich stärker wächst, als die des EAs, in welcher ein Individuum als Matrix der Koordinaten repräsentiert werden kann. Dies hat insbesondere Auswirkung auf die Verarbeitungsgeschwindigkeit der jeweiligen Repräsentation. Die vorangegangenen Experimente haben zudem gezeigt, dass ein naiver EA bei einer größeren Menge an Packstücken Schwierigkeiten bekommt, eine valide Beladung zu ermitteln. Die zusätzliche Schwierigkeit eine valide Beladung zu finden, die durch weitere Vektoren in der Individuumsmatrix entsteht, wird in diesem Fall durch das Geometriemodell auf Kosten der nachträglichen Korrekturen umgangen.
Eine komplette Übersicht der Zahlen finden sich auch hier wieder im Anhang A.4.

		Auftragsgröße			
		10	20	50	100
EA+Geo	Strategie 1	0.759 ↘ 0.656	0.717 ↘ 0.634	0.667 ↘ 0.614	0.639 ↘ 0.61
	Strategie 2	0.671 ↘ 0.585	0.635 ↘ 0.549	0.579 ↘ 0.522	0.554 ↘ 0.516
	Strategie 3	0.734 ↘ 0.620	0.697 ↘ 0.621	0.650 ↘ 0.598	0.621 ↘ 0.592
RL+MCTS	Strategie 1	0.748 ↘ 0.730	0.686 ↘ 0.651	0.614 ↘ 0.57	0.549 ↘ 0.511
	Strategie 2	0.617 ↘ 0.606	0.601 ↘ 0.588	0.578 ↘ 0.567	0.560 ↘ 0.555
	Strategie 3	0.697 ↘ 0.673	0.645 ↘ 0.629	0.605 ↘ 0.576	0.567 ↘ 0.539

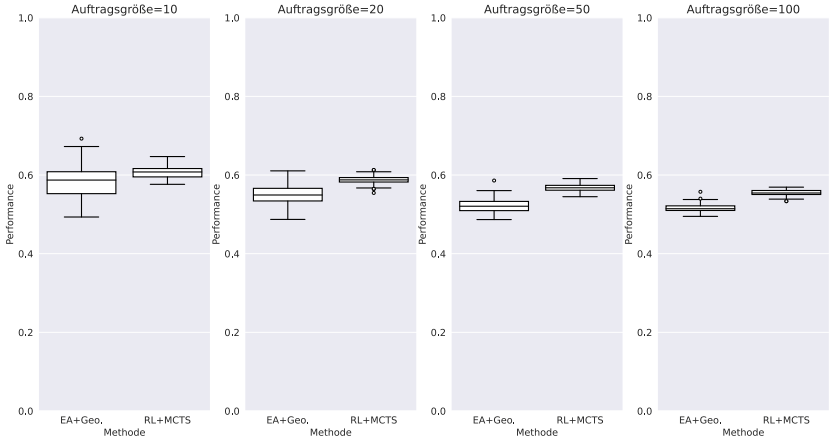
Tabelle 4.3: Offline-BPP Performance Vergleich: Die Tabelle zeigt die durchschnittlichen Bewertungen der RL+MCTS-Kombination und des EAs mit Geometriemodells für die beiden Zeitlimits. Der erste Wert einer Zeile gibt dabei die durchschnittliche Performance der jeweils 100 Experimente mit größerem Zeitlimit an.

		Auftragsgröße			
		10	20	50	100
EA+Geo	Strategie 1	0.313	0.125	0.087	0.050
	Strategie 2	0.199	0.123	0.099	0.063
	Strategie 3	0.220	0.166	0.087	0.040
	∅	**0.244**	**0.138**	**0.091**	**0.051**
RL+MCTS	Strategie 1	0.150	0.189	0.150	0.102
	Strategie 2	0.071	0.059	0.046	0.036
	Strategie 3	0.154	0.138	0.011	0.108
	∅	**0.125**	**0.128**	**0.102**	**0.081**

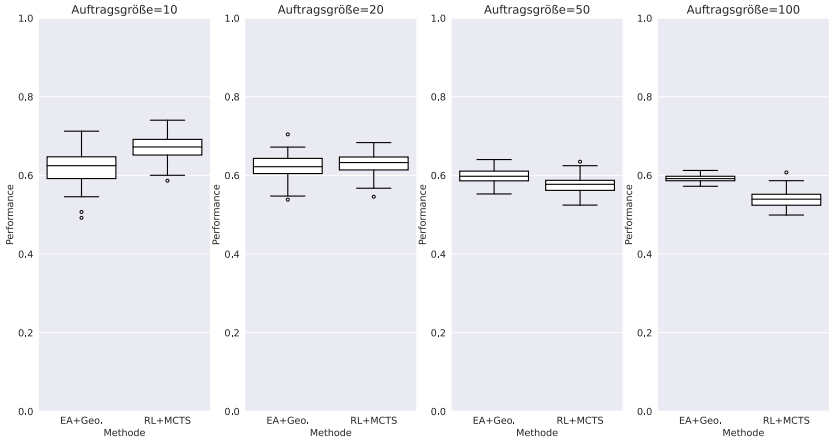
Tabelle 4.4: Offline-BPP Spannweiten mit geringerem Zeitlimit: Die Tabelle zeigt die Spannweiten von EA+Geo. und RL+MCTS im Rahmen der Experimente und deren Durchschnitt über die evaluierten Strategien. Im Vergleich zu Tabelle 4.2 wurde hier das Laufzeitlimit reduziert.



(a) 1. Strategie



(b) 2. Strategie



(c) 3. Strategie

Abbildung 4.11: Offline-BPP Performance mit geringerem Zeitlimit: Die Boxplots bilden die Performance der verschiedenen Varianten ab. Dabei wurden für verschiedene Packauftragsgrößen jeweils 100 Versuche durchgeführt. Im Vergleich zu Abb. 4.10 wurde hier das Laufzeitlimit reduziert.

4.3 Diskussion

In diesem Kapitel wurde die vorgestellte Implementierung anhand von praktischen Experimenten in verschiedenen Disziplinen evaluiert. Da das Ziel der Arbeit eine adaptive Methode zur Lösung der betrachteten BPP-Variante ist, wurden im Rahmen der Evaluierung drei unterschiedliche Strategien in Form von Daten bereitgestellt, anhand derer das Verhalten untersucht werden sollte.
Der erste Teil der Evaluierung beschäftigte sich dabei mit dem IRL. Hier sollte zu jeder Strategie eine entsprechende Reward-Funktion gefunden werden. Diese sind als Linearkombination der in Kapitel 3.2 definierten Packfeatures darstellbar. Die Evaluierung hat hierbei gezeigt, dass die Charakteristika der betrachteten Strategien in den ermittelten Reward-Funktionen erkennbar sind. Insbesondere haben die Versuche mit variierenden Datenmengen gezeigt, dass eine größere Datenmenge einen positiven Einfluss auf das Ergebnis hat.
Der zweite Teil der Experimente richtete sich an das RL. Hierbei sollten auf Basis von DDPG Agenten für die drei Strategien trainiert werden. Dazu wurde für jede der Strategien ein separates Hyperparameter-Tuning durchgeführt und die jeweils besten Ergebnisse betrachtet. Bei zwei der drei Strategien zeigte sich, dass das DDPG grundsätzlich in der Lage ist, Strategien zu erlernen. Hierbei schien das Behaviour Cloning einen wichtigen Einfluss auf die Leistung zu haben. Wünschenswert wäre hier ein ausgiebigeres Hyperparameter-Tuning, das zudem weitere Parameter in Betracht ziehen könnte, wie zusätzliche Aspekte der Netzwerkarchitekturen. Dadurch könnte die Leistung der RL-Agenten weiter gesteigert werden.
Der dritte und letzte Teil der Evaluierung betrachtete darüber hinaus die Offline-Variante des Problems. Hierzu wurden für einen aussagekräftigeren Vergleich zusätzliche Verfahren implementiert. Insbesondere bei geringem Zeitlimit konnte die im Fokus stehende Variante, die die MCTS mit den RL-Agenten kombiniert, gute Ergebnisse erzielen. Jedoch musste sie sich in Bezug auf die Leistung bei größeren Zeitlimits von einem EA in Kombination mit dem Geometriemodell geschlagen geben. Letzteres konnte dabei die Leistung im Vergleich zu einem naiven EA deutlich steigern. Bemerkenswert ist hierbei zudem, dass der EA mit Geometriemodell derartig gute Resultate ad-hoc liefern konnte und dabei nicht auf ein rechenintensives Training, wie die RL+MCTS-Kombination, angewiesen ist.
Die Evaluierungen haben gezeigt, dass die grundsätzliche IRL-RL-Pipeline funktionieren kann. Dennoch ist hier noch viel Spielraum für Verbesserungen und Variation. So könnte, wie bereits erwähnt, das Hyperparameter-Tuning des DDPG-Trainings ausgebaut werden, welches auch Einfluss auf die Leistung der Offline-Variante haben könnte. Darüber hinaus zeigte der EA in Kombination mit dem Geometriemodell sehr gute Resultate. Eine intensivere Betrachtung und eine mögliche Weiterentwicklung dessen könnte hier von größerem Interesse sein.

Kapitel 5

Fazit

Das Ziel dieser Arbeit war es einen Ansatz zu entwickeln, um Strategien erfahrener Packer in der Logistik in ein technisches System überführen zu können. Die Strategien sollen vom System auf ungesehene Probleme reproduziert werden können, sodass Charakteristika der Arbeitsweise wiedererkennbar sind.
Der in der Arbeit präsentierte Ansatz setzt dabei auf maschinelles Lernen. Auf Basis gesammelter Expertendaten wird mit Hilfe von Inverse Reinforcement Learning eine Reward-Funktion berechnet, die ein zum Experten ähnliches Verhalten belohnen soll. Für das Training eines Systems, welches schlussendlich eine solche Strategie lernen und anwenden soll, wird Reinforcement Learning verwendet. Dabei wird ein Agent, repräsentiert durch ein statistisches Modell, so trainiert, dass die vom IRL berechnete Reward-Funktion möglichst maximiert, das Verhalten des Experten also möglichst genau imitiert wird.
Der präsentierte Ansatz wurde dabei im Rahmen einer Evaluierung untersucht. Dazu wurden entsprechende Komponenten implementiert, Daten gesammelt und das Verhalten anhand unterschiedlicher Strategien untersucht, um die tatsächliche Fähigkeit zur Adaptivität prüfen zu können. Die Evaluierungen gaben vielversprechende Resultate auf, zeigten aber ebenso, dass die vorgestellte Implementierung noch ausbaufähig ist.
So könnten im Rahmen eines weitreichenden RL-Hyperparameter-Tunings sowohl größere Wertebereiche, als auch der Einfluss weiterer Hyperparameter, wie beispielsweise zusätzlicher Aspekte der Netzwerk-Architektur, untersucht werden. Zudem handelt es sich beim Deep Reinforcement Learning um ein aktives Forschungsgebiet. Neben dem hier in der Arbeit betrachteten DDPG stünden eine Vielzahl weiterer Verfahren zur Verfügung. Allein während der Planung, Konzipierens und Schreibens dieser Arbeit erschienen zahlreiche Ansätze, die interessant zu evaluieren wären [37][69][85][101][79]. Im Rahmen dieser Arbeit wäre dies jedoch nicht praktikabel gewesen. Der Autor sieht insbesondere hier mit genügend Zeit und Ressourcen noch großes Potential für Verbesserungen, welche sich auch auf die Leistung der Offline-Variante des BPP auswirken könnten.
Zudem wäre eine Evaluierung in einem realistischeren, praktischen Rahmen interessant,

da die gezeigten Evaluierungen alle auf synthetisch erzeugten Daten beruhen. So könnte beispielsweise eine Feldstudie, in realen Unternehmen und tatsächlich erfahrenen Packern, interessant sein. Darauf aufbauend wäre eine Implementierung und Evaluierung eines Systems auf Basis des Human-Experience-Transfer-Models zum Anlernen unerfahrener Mitarbeiter interessant.
Trotz des noch offenen Optimierungspotentials sieht der Autor die Arbeit als Schritt in Richtung der Realisierung eines derartigen digitalen Assistenzsystems. Auf Basis der Ergebnisse lässt sich weitere Forschung betreiben. Dies beschränkt sich dabei nicht ausschließlich auf den logistischen Packungsprozess. So können präsentierte Ideen, Stärken und Schwachstellen in die Anwendung auf weitere, komplexere Prozesse in der Logistik, wie der Kommissionierung oder dem Transport, übertragen werden.

Anhang A

Weitere Informationen

A.1 Verwendete Software

Sowohl das Environment, als auch der Agent wurden in der Programmiersprache *Python* implementiert [95]. Für die Repräsentation der Zustände und Aktionen wurde dazu Datenstrukturen aus *NumPy* und *OpenCV* verwendet [94][52]. Das Environment selbst wurde dabei im Rahmen der *OpenAI Gym*-API implementiert [18].
Der DDPG-Agent wird mit Hilfe neuronaler Netze implementiert und trainiert. Dazu kam die DL-Library *PyTorch* zum Einsatz [75]. Das Tuning der Hyperparameter wurde mit Hilfe des *Tune* Frameworks und der *Hyperopt*-Implementierung realisiert [60][12]. Die Implementierung des DDPG-Agenten basiert auf den *OpenAI-Baselines* [29].
Der Simulator, in dem die Daten für die Evaluierungen erzeugt wurden, wurde mit Hilfe der *Unity Engine* implementiert [2]. Als Scripting-Backend wurde dabei das *.NET-Framework* verwendet [1].

A.2 Evaluierung - IRL Ergebnisse

	Gewichte											Performance			
%-Anteil	**RH**	**RW**	**RUM**	**RBM**	**RRM**	**RLM**	**RD**	**OHR**	**OBR**	**RBC**	**∅-Var.**	**∅**	**Var.**	**Min**	**Max**
10	0.231	0.26	0.031	0.0	0.029	0.0	0.114	0.216	0.017	0.103	9.7E-0.5	0.785	3.97E-04	0.703	0.849
20	0.241	0.268	0.005	0.0	0.017	0.0	0.12	0.227	0.011	0.113	4.0E-0.5	0.813	1.51E-04	0.759	0.853
30	0.245	0.275	0.003	0.0	0.016	0.0	0.113	0.233	0.007	0.11	2.7E-0.5	0.826	9.67E-05	0.789	0.856
40	0.247	0.277	0.002	0.0	0.008	0.0	0.113	0.234	0.004	0.115	2.0E-0.5	0.833	6.41E-05	0.793	0.856
50	0.246	0.275	0.001	0.0	0.009	0.0	0.117	0.234	0.003	0.115	1.7E-0.5	0.831	5.16E-05	0.794	0.854
60	0.248	0.277	0.001	0.0	0.005	0.0	0.117	0.235	0.002	0.115	1.4E-0.5	0.835	4.22E-05	0.804	0.854
70	0.248	0.276	0.002	0.0	0.004	0.0	0.119	0.233	0.004	0.114	1.4E-0.5	0.833	4.43E-05	0.807	0.853
80	0.249	0.275	0.001	0.0	0.007	0.0	0.118	0.234	0.003	0.112	1.4E-0.5	0.833	4.41E-05	0.806	0.855
90	0.248	0.275	0.002	0.0	0.005	0.0	0.119	0.234	0.005	0.113	1.3E-0.5	0.833	4.09E-05	0.805	0.852
100	0.249	0.276	0.001	0.0	0.004	0.0	0.118	0.235	0.004	0.114	1.2E-0.6	0.835	3.45E-05	0.811	0.852

Tabelle A.1: Ergebnisse der IRL Evaluierung für Strategie 1.

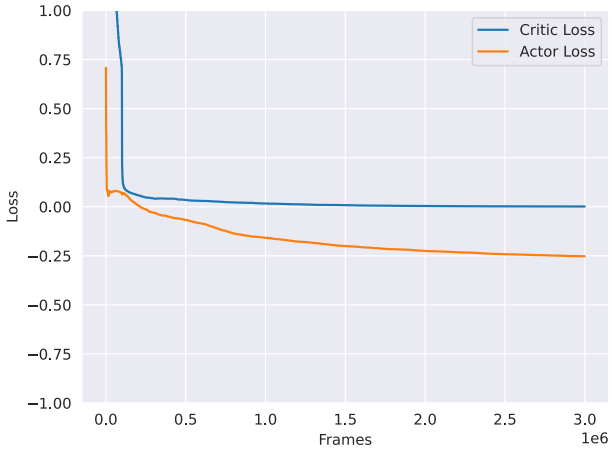
	Gewichte											Performance			
%-Anteil	**RH**	**RW**	**RUM**	**RBM**	**RRM**	**RLM**	**RD**	**OHR**	**OBR**	**RBC**	**∅-Var.**	**∅**	**Var.**	**Min**	**Max**
10	0.183	0.238	0.051	0.053	0.063	0.048	0.095	0.168	0.013	0.089	1.1E-0.4	0.785	3.97E-04	0.703	0.849
20	0.162	0.245	0.051	0.052	0.075	0.052	0.101	0.166	0.005	0.091	7.0E-05	0.813	1.51E-04	0.759	0.853
30	0.164	0.247	0.053	0.054	0.078	0.05	0.099	0.166	0.001	0.087	6.8E-0.5	0.826	9.67E-05	0.789	0.856
40	0.167	0.246	0.051	0.054	0.073	0.054	0.099	0.169	0.002	0.085	6.0E-0.5	0.833	6.41E-05	0.793	0.856
50	0.174	0.248	0.053	0.053	0.07	0.044	0.098	0.171	0.002	0.088	6.0E-0.5	0.831	5.16E-05	0.794	0.854
60	0.178	0.252	0.051	0.052	0.066	0.041	0.095	0.174	0.001	0.09	5.9E-0.5	0.835	4.22E-05	0.804	0.854
70	0.178	0.253	0.054	0.053	0.061	0.039	0.093	0.176	0.001	0.09	5.8E-0.5	0.833	4.43E-05	0.807	0.853
80	0.178	0.254	0.055	0.052	0.059	0.04	0.092	0.177	0.001	0.093	5.8E-0.5	0.833	4.41E-05	0.806	0.855
90	0.179	0.256	0.054	0.05	0.057	0.039	0.091	0.178	0.0	0.095	5.9E-0.5	0.833	4.09E-05	0.805	0.852
100	0.181	0.256	0.053	0.047	0.058	0.038	0.091	0.179	0.0	0.096	6.3E-0.6	0.835	3.45E-05	0.811	0.852

Tabelle A.2: Ergebnisse der IRL Evaluierung für Strategie 2.

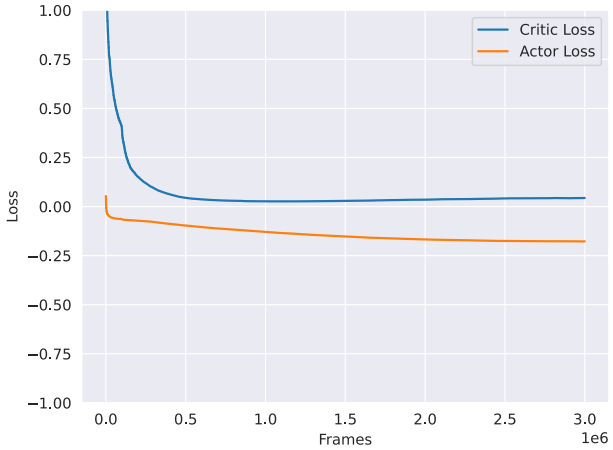
	Gewichte											Performance			
%-Anteil	**RH**	**RW**	**RUM**	**RBM**	**RRM**	**RLM**	**RD**	**OHR**	**OBR**	**RBC**	**∅-Var.**	**∅**	**Var.**	**Min**	**Max**
10	0.222	0.262	0.023	0.0	0.022	0.0	0.071	0.206	0.067	0.127	6.3E-0.5	0.774	8.84E-05	0.724	0.811
20	0.228	0.265	0.012	0.0	0.019	0.0	0.07	0.21	0.061	0.135	3.1E-0.5	0.786	3.99E-05	0.757	0.81
30	0.229	0.268	0.011	0.0	0.018	0.0	0.066	0.212	0.063	0.133	2.3E-0.5	0.791	3.23E-05	0.764	0.816
40	0.231	0.268	0.01	0.0	0.014	0.0	0.065	0.212	0.064	0.136	1.8E-0.5	0.793	2.55E-05	0.773	0.813
50	0.23	0.268	0.011	0.0	0.01	0.0	0.066	0.214	0.064	0.136	1.6E-0.5	0.795	2.24E-05	0.778	0.813
60	0.232	0.271	0.007	0.0	0.006	0.0	0.066	0.216	0.064	0.139	1.5E-0.5	0.802	2.30E-05	0.782	0.818
70	0.232	0.271	0.008	0.0	0.006	0.0	0.066	0.216	0.064	0.136	1.4E-0.5	0.801	2.19E-05	0.785	0.818
80	0.233	0.273	0.003	0.0	0.005	0.0	0.066	0.218	0.062	0.14	1.2E-0.5	0.806	1.94E-05	0.788	0.821
90	0.234	0.274	0.002	0.0	0.004	0.0	0.068	0.219	0.06	0.139	1.1E-0.5	0.809	1.68E-05	0.792	0.822
100	0.234	0.274	0.002	0.0	0.003	0.0	0.068	0.22	0.06	0.139	9.9E-0.6	0.81	1.36E-05	0.792	0.823

Tabelle A.3: Ergebnisse der IRL Evaluierung für Strategie 3.

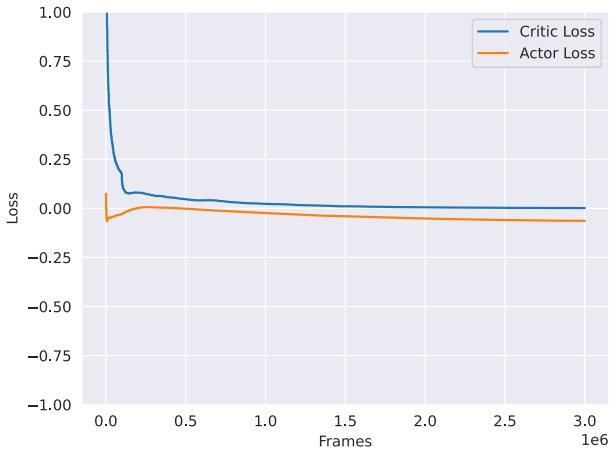
A.3 Evaluierung - RL Ergebnisse



(a) 1. Strategie



(b) 2. Strategie



(c) 3. Strategie

Abbildung A.1: RL-Online-Losses (Ohne BC): Auf der X-Achse sind die Frames des Trainings aufgetragen. Die Y-Achse gibt die Entwicklung der zwei Loss-Funktionen über die Dauer des Trainings an. In allen drei Abbildungen ist ein konvergentes Verhalten des Critic-Losses zu erkennen. Der Actor-Loss dagegen scheint insbesondere in der ersten und zweiten Strategie zum Ende des Trainings zu divergieren. Dies könnte auf eine Bewegung aus einem lokalen Optimum hindeuten.

A.4 Evaluierung - Offline Ergebnisse

Algorithmus	∅	Var.	Min	Max	Spannweite
EA	0.249	1.33E-01	0.0	0.865	0.865
EA+Geo.	0.759	1.46E-03	0.647	0.837	0.189
MCTS	0.632	2.50E-03	0.532	0.756	0.224
RL+MCTS	0.748	6.89E-04	0.685	0.807	0.122

(a) Auftragsgröße: 10

Algorithmus	∅	Var.	Min	Max	Spannweite
EA	0.0	0.00E+00	0	0	0
EA+Geo.	0.717	5.88E-04	0.637	0.768	0.13
MCTS	0.562	8.62E-04	0.505	0.645	0.14
RL+MCTS	0.686	5.22E-04	0.624	0.742	0.118

(b) Auftragsgröße: 20

Algorithmus	∅	Var.	Min	Max	Spannweite
EA	0.0	0.00E+00	0	0	0
EA+Geo.	0.667	1.57E-04	0.636	0.704	0.068
MCTS	0.529	2.88E-04	0.476	0.571	0.095
RL+MCTS	0.614	6.50E-04	0.528	0.671	0.143

(c) Auftragsgröße: 50

Algorithmus	∅	Var.	Min	Max	Spannweite
EA	0.0	0.00E+00	0	0	0
EA+Geo.	0.639	8.93E-05	0.619	0.66	0.04
MCTS	0.51	3.36E-04	0.458	0.539	0.081
RL+MCTS	0.549	1.07E-03	0.476	0.616	0.14

(d) Auftragsgröße: 100

Tabelle A.4: Offline-BPP Ergebnisse für Strategie 1.

Algorithmus	∅	Var.	Min	Max	Spannweite
EA	0.248	1.06E-01	0.0	0.757	0.757
EA+Geo.	0.671	9.26E-04	0.602	0.742	0.14
MCTS	0.601	1.10E-03	0.529	0.665	0.136
RL+MCTS	0.617	2.04E-04	0.578	0.653	0.074

(a) Auftragsgröße: 10

Algorithmus	∅	Var.	Min	Max	Spannweite
EA	0.0	0.00E+00	0	0	0
EA+Geo.	0.635	6.35E-04	0.576	0.686	0.11
MCTS	0.575	5.61E-04	0.511	0.626	0.115
RL+MCTS	0.601	1.46E-04	0.574	0.626	0.053

(b) Auftragsgröße: 20

Algorithmus	∅	Var.	Min	Max	Spannweite
EA	0.0	0.00E+00	0	0	0
EA+Geo.	0.579	4.08E-04	0.542	0.622	0.08
MCTS	0.548	7.04E-04	0.461	0.586	0.125
RL+MCTS	0.578	8.04E-05	0.554	0.604	0.05

(c) Auftragsgröße: 50

Algorithmus	∅	Var.	Min	Max	Spannweite
EA	0.0	0.00E+00	0	0	0
EA+Geo.	0.554	2.91E-04	0.523	0.601	0.077
MCTS	0.537	8.13E-04	0.442	0.568	0.126
RL+MCTS	0.56	4.60E-05	0.536	0.575	0.039

(d) Auftragsgröße: 100

Tabelle A.5: Offline-BPP Ergebnisse für Strategie 2.

Algorithmus	∅	Var.	Min	Max	Spannweite
EA	0.231	1.20E-01	0.0	0.807	0.807
EA+Geo.	0.734	1.21E-03	0.655	0.831	0.176
MCTS	0.591	2.37E-03	0.495	0.743	0.248
RL+MCTS	0.697	6.67E-04	0.63	0.774	0.145

(a) Auftragsgröße: 10

Algorithmus	∅	Var.	Min	Max	Spannweite
EA	0.0	0.00E+00	0	0	0
EA+Geo.	0.697	5.68E-04	0.641	0.772	0.131
MCTS	0.527	6.04E-04	0.484	0.599	0.115
RL+MCTS	0.645	5.07E-04	0.569	0.701	0.132

(b) Auftragsgröße: 20

Algorithmus	∅	Var.	Min	Max	Spannweite
EA	0.0	0.00E+00	0	0	0
EA+Geo.	0.65	1.71E-04	0.622	0.681	0.059
MCTS	0.486	1.60E-04	0.45	0.516	0.067
RL+MCTS	0.605	2.46E-04	0.56	0.638	0.078

(c) Auftragsgröße: 50

Algorithmus	∅	Var.	Min	Max	Spannweite
EA	0.0	0.00E+00	0	0	0
EA+Geo.	0.621	6.33E-05	0.6	0.641	0.041
MCTS	0.47	1.59E-04	0.439	0.492	0.053
RL+MCTS	0.567	3.95E-04	0.506	0.604	0.097

(d) Auftragsgröße: 100

Tabelle A.6: Offline-BPP Ergebnisse für Strategie 3.

Algorithmus	∅	**Var.**	**Min**	**Max**	**Spannweite**
EA+Geo.	0.656	2.91E-03	0.493	0.806	0.313
RL+MCTS	0.73	8.23E-04	0.636	0.786	0.15

(a) Auftragsgröße: 10

Algorithmus	∅	**Var.**	**Min**	**Max**	**Spannweite**
EA+Geo.	0.634	7.92E-04	0.575	0.701	0.125
RL+MCTS	0.651	1.31E-03	0.532	0.72	0.189

(b) Auftragsgröße: 20

Algorithmus	∅	**Var.**	**Min**	**Max**	**Spannweite**
EA+Geo.	0.614	3.32E-04	0.567	0.654	0.087
RL+MCTS	0.57	1.13E-03	0.483	0.637	0.153

(c) Auftragsgröße: 50

Algorithmus	∅	**Var.**	**Min**	**Max**	**Spannweite**
EA+Geo.	0.61	9.35E-05	0.587	0.636	0.05
RL+MCTS	0.511	4.44E-04	0.456	0.557	0.102

(d) Auftragsgröße: 100

Tabelle A.7: Offline-BPP Ergebnisse mit geringerem Zeitlimit für Strategie 1.

Algorithmus	∅	**Var.**	**Min**	**Max**	**Spannweite**
EA+Geo.	0.585	1.67E-03	0.493	0.693	0.199
RL+MCTS	0.606	2.14E-04	0.576	0.647	0.071

(a) Auftragsgröße: 10

Algorithmus	∅	**Var.**	**Min**	**Max**	**Spannweite**
EA+Geo.	0.549	5.82E-04	0.487	0.611	0.123
RL+MCTS	0.588	1.11E-04	0.554	0.613	0.059

(b) Auftragsgröße: 20

Algorithmus	∅	**Var.**	**Min**	**Max**	**Spannweite**
EA+Geo.	0.522	3.07E-04	0.487	0.586	0.099
RL+MCTS	0.567	1.05E-04	0.545	0.591	0.046

(c) Auftragsgröße: 50

Algorithmus	∅	**Var.**	**Min**	**Max**	**Spannweite**
EA+Geo.	0.516	9.27E-05	0.495	0.557	0.063
RL+MCTS	0.555	5.34E-05	0.533	0.569	0.036

(d) Auftragsgröße: 100

Tabelle A.8: Offline-BPP Ergebnisse mit geringerem Zeitlimit für Strategie 2.

Algorithmus	∅	**Var.**	**Min**	**Max**	**Spannweite**
EA+Geo.	0.62	1.85E-03	0.492	0.712	0.22
RL+MCTS	0.673	8.72E-04	0.586	0.74	0.154

(a) Auftragsgröße: 10

Algorithmus	∅	**Var.**	**Min**	**Max**	**Spannweite**
EA+Geo.	0.621	8.42E-04	0.538	0.704	0.166
RL+MCTS	0.629	6.32E-04	0.546	0.683	0.138

(b) Auftragsgröße: 20

Algorithmus	∅	**Var.**	**Min**	**Max**	**Spannweite**
EA+Geo.	0.598	2.93E-04	0.553	0.64	0.087
RL+MCTS	0.576	4.27E-04	0.524	0.635	0.11

(c) Auftragsgröße: 50

Algorithmus	∅	**Var.**	**Min**	**Max**	**Spannweite**
EA+Geo.	0.592	8.49E-05	0.572	0.612	0.04
RL+MCTS	0.539	3.87E-04	0.499	0.607	0.108

(d) Auftragsgröße: 100

Tabelle A.9: Offline-BPP Ergebnisse mit geringerem Zeitlimit für Strategie 3.

Akronymverzeichnis

HETM Human-Experience-Transfer-Model

BPP Bin-Packing-Problem

RL Reinforcement Learning

MEP Markov-Entscheidungsprozess

MLP Multi Layer Perceptron

BC Behaviour Cloning

IRL Inverse Reinforcement Learning

REIRL Relative Entropy Inverse Reinforcement Learning

RH Relative Height

RW Relative Weight

RUM Relative Upper Margin

RBM Relative Bottom Margin

RRM Relative Right Margin

RLM Relative Left Margin

RD Relative Density

OBR Overbuilt Ratio

OHR Overhand Ratio

RBC Relative Barycenter

DDPG Deep Deterministic Policy Gradient

MCTS Monte Carlo Tree Search

EA Evolutionärer Algorithmus

Notationsverzeichnis

Symbol	**Bedeutung**
b_i, l_i, h_i, w_i	Länge, Breite, Höhe, Gewicht eines Packstücks i
x_i, y_i, z_i	X-,Y- und Z-Koordinaten eines Packstücks i
$\Pr(x)$	Wahrscheinlichkeit von x
$\Pr(x\|y)$	Wahrscheinlichkeit von x unter der Bedingung y
$\mathbb{E}[x]$	Erwartungswert von x
$x \sim X$	Element x entstammt aus X
$x \propto y$	x ist proportional zu y
$\mathcal{M}$	Markov-Entscheidungsprozess (MEP)
$\mathcal{S}$	Menge der Zustände eines MEP
s	Zustand, Element von $\mathcal{S}$
$\mathcal{P}$	Transitionsmatrix eines MEP
$\mathcal{R}$	Reward-Funktion eines MEP
r	Reward, Ergebnis der Auswertung von $\mathcal{R}$
τ	Trajektorie von Zuständen und Aktionen eines MEP
G	Ertrag einer Trajektorie
Π	Menge der Policies eines MEP
π	Policy, Element von Π
γ	Diskontierungsfaktor eines MEP
G_γ	diskontierter Ertrag einer Trajektorie
V_π	Wertefunktion bezüglich einer Policy π
Q_π	Aktionswertefunktion bezüglich einer Policy π
π^*	Optimale Policy einer Menge Π
ρ	Polyak-Parameter
$\mathcal{B}$	Buffer
$\mathcal{D}$	Demonstrationsdatensatz
$\mathcal{L}$	Loss-Funktion
θ	Modellparameter
β	Lernrate

Abbildungsverzeichnis

Algorithmenverzeichnis

Danksagungen

Ich möchte meinen Betreuerinnen Prof. Dr. Katharina Morik und Dr. Helena Kotthaus vom Lehrstuhl VIII für Künstliche Intelligenz für die stetige Unterstützung während der Bearbeitung der Arbeit herzlich danken. Auch möchte ich Dr. Maryam Tavakol für die fachliche Unterstützung bei der Konzipierung des Themas danken.
Darüber hinaus danke ich dem Fraunhofer-Institut für Materialfluss und Logistik, insbesondere in Person von Benedikt Mättig, für die fachliche und technische Unterstützung.
Die erforderlichen Berechnungen der Arbeit wurden auf dem Linux-HPC-Cluster der Technischen Universität Dortmund (LiDO3) durchgeführt, welches in Teilen durch die Forschungsgroßgeräte-Initiative der Deutschen Forschungsgemeinschaft (DFG) unter der Projektnummer 271512359 gefördert wird. Für die Möglichkeit möchte ich mich herzlich bedanken.

Literaturverzeichnis

[1] *Microsoft .NET*. `https://docs.microsoft.com/de-de/dotnet/`. Accessed: 2020-08-19.

[2] *Unity Engine*. `https://unity.com/`. Accessed: 2020-08-19.

[3] Abbeel, Pieter, Adam Coates und Andrew Ng: *Autonomous Helicopter Aerobatics through Apprenticeship Learning*. I. J. Robotic Res., 29:1608–1639, 11 2010.

[4] Abbeel, Pieter und Andrew Y. Ng: *Apprenticeship Learning via Inverse Reinforcement Learning*. In: *Proceedings of the Twenty-First International Conference on Machine Learning*, ICML '04, Seite 1, New York, NY, USA, 2004. Association for Computing Machinery.

[5] Allen, S.D., E.K. Burke und Graham Kendall: *A hybrid placement strategy for the three-dimensional strip packing problem*. European Journal of Operational Research, 209:219–227, 03 2011.

[6] Anthony, Thomas, Zheng Tian und David Barber: *Thinking Fast and Slow with Deep Learning and Tree Search*. CoRR, abs/1705.08439, 2017.

[7] Bauernhansl, Thomas: *Die Vierte Industrielle Revolution – Der Weg in ein wertschaffendes Produktionsparadigma*, Seiten 5–35. Springer Fachmedien Wiesbaden, Wiesbaden, 2014.

[8] Bello, Irwan, Hieu Pham, Quoc V. Le, Mohammad Norouzi und Samy Bengio: *Neural Combinatorial Optimization with Reinforcement Learning*, 2016.

[9] Berg, Mark de, Otfried Cheong, Marc van Kreveld und Mark Overmars: *Computational Geometry: Algorithms and Applications*, Seiten 212–218. Springer-Verlag TELOS, Santa Clara, CA, USA, 2. Auflage, 2000.

[10] Berg, Mark de, Otfried Cheong, Marc van Kreveld und Mark Overmars: *Computational Geometry: Algorithms and Applications*, Seiten 235–250. Springer-Verlag TELOS, Santa Clara, CA, USA, 2. Auflage, 2000.

[11] Bergstra, James, Rémi Bardenet, Yoshua Bengio und Balázs Kégl: *Algorithms for Hyper-Parameter Optimization.* NIPS'11, Seite 2546–2554, Red Hook, NY, USA, 2011. Curran Associates Inc.

[12] Bergstra, James, Daniel Yamins und David Cox: *Making a Science of Model Search: Hyperparameter Optimization in Hundreds of Dimensions for Vision Architectures.* Band 28 der Reihe *Proceedings of Machine Learning Research*, Seiten 115–123, Atlanta, Georgia, USA, 17–19 Jun 2013. PMLR.

[13] Bischoff, Eberhard E.: *Stability aspects of pallet loading.* Operations-Research-Spektrum, 13(4):189–197, Dec 1991.

[14] Bischoff, Eberhard E. und Michael D. Marriott: *A comparative evaluation of heuristics for container loading.* European Journal of Operational Research, 44(2):267 – 276, 1990. Cutting and Packing.

[15] Bortfeldt, Andreas und Daniel Mack: *A heuristic for the three-dimensional strip packing problem.* European Journal of Operational Research, 183(3):1267 – 1279, 2007.

[16] Boularias, Abdeslam, Jens Kober und Jan Peters: *Relative Entropy Inverse Reinforcement Learning.* In: Gordon, Geoffrey, David Dunson und Miroslav Dudík (Herausgeber): *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, Band 15 der Reihe *Proceedings of Machine Learning Research*, Seiten 182–189, Fort Lauderdale, FL, USA, 11–13 Apr 2011. PMLR.

[17] Boyd, Stephen und Jaehyun Park: *Subgradient Methods*, May 2014.

[18] Brockman, Greg, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang und Wojciech Zaremba: *OpenAI Gym.* CoRR, abs/1606.01540, 2016.

[19] Browne, C. B., E. Powley, D. Whitehouse, S. M. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis und S. Colton: *A Survey of Monte Carlo Tree Search Methods.* IEEE Transactions on Computational Intelligence and AI in Games, 4(1):1–43, 2012.

[20] Bundesargentur für Arbeit: *Aktuelle Entwicklungen in der Zeitarbeit*, 2020.

[21] Bundesministerium für Wirtschaft und Energie: *Sensor, Tablet, RFID: Digitale Technologien in der Produktion*, 2017.

[22] Bundesministerium für Wirtschaft und Energie: *Was ist Industrie 4.0?*, 2020.

[23] Capgemini: *The Current and Future Stateof Digital Supply Chain Transformation.* Technischer Bericht, Capgemini Consulting, 2016.

[24] Chaslot, Guillaume, Sander Bakkes, Istvan Szita und Pieter Spronck: *Monte-Carlo Tree Search: A New Framework for Game AI.* 01 2008.

[25] Chaslot, Guillaume, Mark Winands und H. Herik: *Parallel Monte-Carlo Tree Search.* Seiten 60–71, 09 2008.

[26] Choi, Jaedeug und Kee eung Kim: *Nonparametric Bayesian Inverse Reinforcement Learning for Multiple Reward Functions.* In: Pereira, F., C. J. C. Burges, L. Bottou und K. Q. Weinberger (Herausgeber): *Advances in Neural Information Processing Systems 25*, Seiten 305–313. Curran Associates, Inc., 2012.

[27] Coffman, Jr., E. G., M. R. Garey und D. S. Johnson: *An Application of Bin-Packing to Multiprocessor Scheduling.* SIAM Journal on Computing, 7(1):1–17, 1978.

[28] Cormen, Thomas H., Charles E. Leiserson, Ronald L. Rivest und Clifford Stein: *Introduction to Algorithms, Third Edition*, Seiten 348–355. The MIT Press, 3. Auflage, 2009.

[29] Dhariwal, Prafulla, Christopher Hesse, Oleg Klimov, Alex Nichol, Matthias Plappert, Alec Radford, John Schulman, Szymon Sidor, Yuhuai Wu und Peter Zhokhov: *OpenAI Baselines.* `https://github.com/openai/baselines`, 2017.

[30] Dowsland, Kathryn A.: *Determining an upper bound for a class of rectangular packing problems.* Computers and Operations Research, 12(2):201 – 205, 1985.

[31] Duan, Lu, Haoyuan Hu, Yu Qian, Yu Gong, Xiaodong Zhang, Jiangwen Wei und Yinghui Xu: *A Multi-Task Selected Learning Approach for Solving 3D Flexible Bin Packing Problem.* In: *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems*, AAMAS '19, Seite 1386–1394, Richland, SC, 2019. International Foundation for Autonomous Agents and Multiagent Systems.

[32] Dudík, Miroslav und Robert E. Schapire: *Maximum Entropy Distribution Estimation with Generalized Regularization.* In: Lugosi, Gábor und Hans Ulrich Simon (Herausgeber): *Learning Theory*, Seiten 123–138, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[33] Dulac-Arnold, Gabriel, Richard Evans, Peter Sunehag und Ben Coppin: *Reinforcement Learning in Large Discrete Action Spaces.* CoRR, abs/1512.07679, 2015.

[34] Eliiyi, Ugur und Deniz Eliiyi: *Applications of Bin Packing Models Through The Supply Chain.* International Journal of Business and Management Studies, 1:11–19, 01 2009.

[35] Fink, Gernot A.: *Word Spotting in the Era of Deep Learning.* Tutorial (invited) presented at École Pratique des Hautes Études, Paris, France, 2019.

[36] Finn, Chelsea, Sergey Levine und Pieter Abbeel: *Guided Cost Learning: Deep Inverse Optimal Control via Policy Optimization.* CoRR, abs/1603.00448, 2016.

[37] Fu, Justin, Aviral Kumar, Ofir Nachum, George Tucker und Sergey Levine: *D4RL: Datasets for Deep Data-Driven Reinforcement Learning*, 2020.

[38] Fujiyoshi, Kunihiro, Hiroyuki Kawai und Koichi Ishihara: *A Tree Based Novel Representation for 3D-Block Packing.* Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on, 28:759 – 764, 06 2009.

[39] Fukushima, Kunihiko: *Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position.* Biological Cybernetics, 36(4):193–202, April 1980.

[40] Garcia-Garcia, Alberto, Sergio Orts-Escolano, Sergiu Oprea, Victor Villena-Martinez, Pablo Martinez-Gonzalez und Jose Garcia-Rodriguez: *A survey on deep learning techniques for image and video semantic segmentation.* Applied Soft Computing, 70:41 – 65, 2018.

[41] Garey, Michael R. und David S. Johnson: *Computers and Intractability - A Guide to the Theory of NP-Completeness.* W. H. Freeman and Co., USA, 1990.

[42] Gehring, H. und A. ortfeldt: *A genetic algorithm for solving the container loading problem.* International Transactions in Operational Research, 4(5):401 – 418, 1997.

[43] He, Kaiming, Xiangyu Zhang, Shaoqing Ren und Jian Sun: *Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification.* In: *Proceedings of the 2015 IEEE International Conference on Computer Vision (ICCV)*, ICCV '15, Seite 1026–1034, USA, 2015. IEEE Computer Society.

[44] Henderson, Peter, Riashat Islam, Philip Bachman, Joelle Pineau, Doina Precup und David Meger: *Deep Reinforcement Learning that Matters.* CoRR, abs/1709.06560, 2017.

[45] Hester, Todd, Matej Vecerík, Olivier Pietquin, Marc Lanctot, Tom Schaul, Bilal Piot, Andrew Sendonaris, Gabriel Dulac-Arnold, Ian Osband, John P. Agapiou, Joel Z. Leibo und Audrunas Gruslys: *Learning from*

Demonstrations for Real World Reinforcement Learning. CoRR, abs/1704.03732, 2017.

[46] Ho, Jonathan und Stefano Ermon: *Generative Adversarial Imitation Learning.* CoRR, abs/1606.03476, 2016.

[47] Hochreiter, Sepp: *The Vanishing Gradient Problem During Learning Recurrent Neural Nets and Problem Solutions.* Int. J. Uncertain. Fuzziness Knowl.-Based Syst., 6(2):107–116, April 1998.

[48] Hopper, E. und Brian Turton: *A Review of the Application of Meta-Heuristic Algorithms to 2D Strip Packing Problems.* Artificial Intelligence Review, 16:257–300, 12 2001.

[49] Hornik, Kurt, Maxwell Stinchcombe und Halbert White: *Multilayer feedforward networks are universal approximators.* Neural Networks, 2(5):359 – 366, 1989.

[50] Hossain, MD. Zakir, Ferdous Sohel, Mohd Fairuz Shiratuddin und Hamid Laga: *A Comprehensive Survey of Deep Learning for Image Captioning.* 51(6), Februar 2019.

[51] Hu, Haoyuan, Xiaodong Zhang, Xiaowei Yan, Longfei Wang und Yinghui Xu: *Solving a New 3D Bin Packing Problem with Deep Reinforcement Learning Method*, 2017.

[52] Itseez: *The OpenCV Reference Manual*, 2.4.9.0 Auflage, April 2014.

[53] Kingma, Diederik und Jimmy Ba: *Adam: A Method for Stochastic Optimization.* International Conference on Learning Representations, 12 2014.

[54] Kocsis, Levente und Csaba Szepesvári: *Bandit Based Monte-Carlo Planning.* In: Fürnkranz, Johannes, Tobias Scheffer und Myra Spiliopoulou (Herausgeber): *Machine Learning: ECML 2006*, Seiten 282–293, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[55] Kullback, S. und R. A. Leibler: *On Information and Sufficiency.* Ann. Math. Statist., 22(1):79–86, 03 1951.

[56] Lapan, Maxim: *Deep Reinforcement Learning Hands-On.* Packt Publishing, 2018.

[57] LeCun, Y., B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard und L. D. Jackel: *Backpropagation Applied to Handwritten Zip Code Recognition.* Neural Computation, 1(4):541–551, Dezember 1989.

[58] Lee, D. T.: *Computational Geometry I*, Seite 1. Chapman and Hall/CRC, 2 Auflage, 2010.

[59] Levine, Sergey und Vladlen Koltun: *Continuous Inverse Optimal Control with Locally Optimal Examples.* CoRR, abs/1206.4617, 2012.

[60] Liaw, Richard, Eric Liang, Robert Nishihara, Philipp Moritz, Joseph E Gonzalez und Ion Stoica: *Tune: A Research Platform for Distributed Model Selection and Training.* arXiv preprint arXiv:1807.05118, 2018.

[61] Lillicrap, Timothy P., Jonathan J. Hunt, Alexander Pritzel, Nicolas Manfred Otto Heess, Tom Erez, Yuval Tassa, David Silver und Daan Wierstra: *Continuous control with deep reinforcement learning.* CoRR, abs/1509.02971, 2016.

[62] Mao, Feng, Edgar Blanco, Mingang Fu, Rohit Jain, Anurag Gupta, Sebastien Mancel, Rong Yuan, Stephen Guo, Sai Kumar und Yayang Tian: *Small Boxes Big Data: A Deep Learning Approach to Optimize Variable Sized Bin Packing.* CoRR, abs/1702.04415, 2017.

[63] Mazonka, Oleg: *Easy As Pi: The Importance Sampling Method.* Journal of Reference, 06 2016.

[64] Miyazawa, F. K. und Y. Wakabayashi: *An Algorithm for the Three-Dimensional Packing Problem with Asymptotic Performance Analysis.* Algorithmica, 18(1):122–144, Mai 1997.

[65] Mülling, Katharina, Abdeslam Boularias, Betty Mohler, Bernhard Schölkopf und Jan Peters: *Learning strategies in table tennis using inverse reinforcement learning.* Biological cybernetics, 108, 04 2014.

[66] Mnih, Volodymyr, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra und Martin A. Riedmiller: *Playing Atari with Deep Reinforcement Learning.* CoRR, abs/1312.5602, 2013.

[67] Mättig, Benedikt und Hermann Foot: *Approach to improving training of human workers in industrial applications through the use of Intelligence Augmentation and Human-in-the-Loop.* International Conference on Computer Science and Education, 15(1):201–213, 7 2020.

[68] Mättig, Benedikt und Veronika Kretschmer: *Einsatz digitaler Assistenzsysteme in der Logistik 4.0.* In: *Handbuch Industrie 4.0*, Seiten 1–25. Springer Berlin Heidelberg, Berlin, Heidelberg, 2019.

[69] Nair, Ashvin, Murtaza Dalal, Abhishek Gupta und Sergey Levine: *Accelerating Online Reinforcement Learning with Offline Datasets*, 2020.

[70] Nair, Ashvin, Bob McGrew, Marcin Andrychowicz, Wojciech Zaremba und Pieter Abbeel: *Overcoming Exploration in Reinforcement Learning with Demonstrations.* CoRR, abs/1709.10089, 2017.

[71] Nair, Vinod und Geoffrey E. Hinton: *Rectified Linear Units Improve Restricted Boltzmann Machines.* In: *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ICML'10, Seite 807–814, Madison, WI, USA, 2010. Omnipress.

[72] Niemann, Heinrich: *Klassifikation von Mustern*, Kapitel 4, Seiten 383–395. Springer Verlag, Berlin, 1983.

[73] Osa, Takayuki, Naohiko Sugita und Mamoru Mitsuishi: *Online Trajectory Planning in Dynamic Environments for Surgical Task Automation.* 07 2014.

[74] Owen, Art B.: *Monte Carlo theory, methods and examples*, Kapitel 9. 2013.

[75] Paszke, Adam, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga und Adam Lerer: *Automatic differentiation in PyTorch.* In: *NIPS-W*, 2017.

[76] Plath, H.-E.: *Erfahrungswissen und Handlungskompetenz - Konsequenzen fürdie berufliche Weiterbildung.* In: *IAB-Kompendium Arbeitsmarkt- und Berufsforschung.Beiträge zur Arbeitsmarkt- und Berufsforschung*, 2002.

[77] Polyak, Boris und Anatoli Juditsky: *Acceleration of Stochastic Approximation by Averaging.* SIAM Journal on Control and Optimization, 30:838–855, 07 1992.

[78] Pomerleau, Dean A.: *ALVINN: An Autonomous Land Vehicle in a Neural Network.* In: *Advances in Neural Information Processing Systems 1*, Seiten 305–313. San Francisco, CA: Morgan Kaufmann, 1989.

[79] Reddy, Siddharth, Anca D. Dragan und Sergey Levine: *SQIL: Imitation Learning via Regularized Behavioral Cloning.* CoRR, abs/1905.11108, 2019.

[80] Ross, Stephane, Geoffrey J. Gordon und J. Andrew Bagnell: *A Reduction of Imitation Learning and Structured Prediction to No-Regret Online Learning*, 2010.

[81] Ross, Stephane, Narek Melik-Barkhudarov, Kumar Shaurya Shankar, Andreas Wendel, Debadeepta Dey, J. Andrew Bagnell und Martial Hebert: *Learning Monocular Reactive UAV Control in Cluttered Natural Environments*, 2012.

[82] Ruderman, Mykhaylo: *Entwicklung eines Verfahrens zur statischen Stabilitätsanalyse beim Roboterpalettieren.* Diplomarbeit, Universität Dortmund, 2005.

[83] Rumelhart, David E., Geoffrey E. Hinton und Ronald J. Williams: *Learning representations by back-propagating errors.* Nature, 323(6088):533–536, Oktober 1986.

[84] Savitzky, A. und M. J. E. Golay: *Smoothing and Differentiation of Data by Simplified Least Squares Procedures.* Analytical Chemistry, 36:1627–1639, 1964.

[85] Siegel, Noah Y., Jost Tobias Springenberg, Felix Berkenkamp, Abbas Abdolmaleki, Michael Neunert, Thomas Lampe, Roland Hafner, Nicolas Heess und Martin Riedmiller: *Keep Doing What Worked: Behavioral Modelling Priors for Offline Reinforcement Learning*, 2020.

[86] Silver, David, Aja Huang, Christopher J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel und Demis Hassabis: *Mastering the game of Go with deep neural networks and tree search.* Nature, 529:484–503, 2016.

[87] Sobel, Irwin: *An Isotropic 3x3 Image Gradient Operator.* Presentation at Stanford A.I. Project 1968, 02 2014.

[88] Sudholt, Dirk: *Parallel Evolutionary Algorithms*, Seiten 929–959. Springer Berlin Heidelberg, Berlin, Heidelberg, 2015.

[89] Sutskever, Ilya, Oriol Vinyals und Quoc V. Le: *Sequence to Sequence Learning with Neural Networks*, 2014.

[90] Sutton, Richard S. und Andrew G. Barto: *Reinforcement Learning: An Introduction.* A Bradford Book, Cambridge, MA, USA, 2018.

[91] ten Hompel, Michael und Michael Henke: *Industrie 4.0 in Produktion, Automatisierung und Logistik: Anwendung · Technologien · Migration*, Seiten 615–624. Springer Fachmedien Wiesbaden, Wiesbaden, 2014.

[92] ten Hompel, Michael, Volker Sadowsky und Maria Beck: *Kommissionierung: Materialflusssysteme 2 - Planung und Berechnung der Kommissionierung in der Logistik.* Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.

[93] ten Hompel, Michael, Volker Sadowsky und Maria Beck: *Kommissionierung: Materialflusssysteme 2 - Planung und Berechnung der Kommissionierung in der Logistik*, Seiten 27–28. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.

[94] Van Der Walt, Stefan, S Chris Colbert und Gael Varoquaux: *The NumPy array: a structure for efficient numerical computation.* Computing in Science & Engineering, 13(2):22, 2011.

[95] Van Rossum, Guido und Fred L. Drake: *Python 3 Reference Manual.* CreateSpace, Scotts Valley, CA, 2009.

[96] VDI-Gesellschaft Produktion und Logistik: *VDI-Richtlinie 3590: Kommissioniersysteme*, 1994.

[97] Vecerík, Matej, Todd Hester, Jonathan Scholz, Fumin Wang, Olivier Pietquin, Bilal Piot, Nicolas Heess, Thomas Rothörl, Thomas Lampe und Martin A. Riedmiller: *Leveraging Demonstrations for Deep Reinforcement Learning on Robotics Problems with Sparse Rewards.* CoRR, abs/1707.08817, 2017.

[98] Wang, Tianqi und Dong Eui Chang: *Improved Reinforcement Learning through Imitation Learning Pretraining Towards Image-based Autonomous Driving.* CoRR, abs/1907.06838, 2019.

[99] Watkins, Christopher und Peter Dayan: *Technical Note: Q-Learning.* Machine Learning, 8:279–292, 05 1992.

[100] Wäscher, Gerhard, Heike Haussner und Holger Schumann: *An improved typology of cutting and packing problems.* European Journal of Operational Research, 183(3):1109 – 1130, 2007.

[101] Wu, Yifan, George Tucker und Ofir Nachum: *Behavior Regularized Offline Reinforcement Learning*, 2019.

[102] Wulfmeier, Markus, Peter Ondruska und Ingmar Posner: *Deep Inverse Reinforcement Learning.* CoRR, abs/1507.04888, 2015.

[103] Yuh, Ping-Hung, Chia-Lin Yang und Yao-Wen Chang: *Temporal Floorplanning Using the T-Tree Formulation.* In: *Proceedings of the 2004 IEEE/ACM International Conference on Computer-Aided Design*, ICCAD '04, Seite 300–305, USA, 2004. IEEE Computer Society.

[104] Zhang, Jiakai und Kyunghyun Cho: *Query-Efficient Imitation Learning for End-to-End Autonomous Driving.* CoRR, abs/1605.06450, 2016.

[105] Zhang, Xiaoqin und Huimin Ma: *Pretraining Deep Actor-Critic Reinforcement Learning Algorithms With Expert Demonstrations.* 01 2018.

[106] Zhao, Zhong-Qiu, Peng Zheng, Shou tao Xu und Xindong Wu: *Object Detection with Deep Learning: A Review*, 2018.

[107] Zheng, Nanning, Ziyi Liu, Pengju Ren, Yongqiang Ma, Shitao Chen, Siyu Yu, Jianru Xue, BD Chen und Feiyue Wang: *Hybrid-augmented intelligence: collaboration and cognition.* Frontiers of Information Technology and Electronic Engineering, 18:153–179, 01 2017.

[108] Zhu, Yuke, Ziyu Wang, Josh Merel, Andrei A. Rusu, Tom Erez, Serkan Cabi, Saran Tunyasuvunakool, János Kramár, Raia Hadsell, Nando de Freitas und Nicolas Heess: *Reinforcement and Imitation Learning for Diverse Visuomotor Skills.* CoRR, abs/1802.09564, 2018.