Technology
Arts Sciences
TH Köln

Regelextraktion aus Deep Reinforcement Learning Modellen

Bachelorarbeit

ausgearbeitet von

Marc Oedingen

zur Erlangung des akademischen Grades
Bachelor of Science (B.Sc.)

vorgelegt an der

Technischen Hochschule Köln
Campus Gummersbach
Fakultät für Informatik und
Ingenieurwissenschaften

im Studiengang
Allgemeine Informatik

Erster Prüfer/in:	Prof. Dr. Wolfgang Konen Technische Hochschule Köln
Zweiter Prüfer/in:	Prof. Dr. Lutz Köhler Technische Hochschule Köln

Gummersbach, im Januar 2022

Adressen: Marc Oedingen
Offermannsheiderstraße 100
51515 Kürten
marc.oedingen@smail.th-koeln.de

Prof. Dr. Wolfgang Konen
Technische Hochschule Köln
Institut für Informatik
Steinmüllerallee 1
51643 Gummersbach
wolfgang.konen@th-koeln.de

Prof. Dr. Lutz Köhler
Technische Hochschule Köln
Institut für Informatik
Steinmüllerallee 1
51643 Gummersbach
lutz.koehler@mac.com

Inhaltsverzeichnis

1. Einleitung **3**
- 1.1. Motivation . . . 3
- 1.2. Problemstellung und Aufbau . . . 4
- 1.3. Aktueller Forschungsstand . . . 4
- 1.4. Forschungsfragen . . . 5

2. Einführung Machine Learning **6**
- 2.1. Überblick . . . 6
- 2.2. Supervised Learning . . . 7
- 2.3. Reinforcement Learning . . . 8
- 2.4. Unsupervised Learning . . . 9

3. Grundlagen **11**
- 3.1. Markov-Entscheidungsprozess . . . 11
- 3.2. Lösungsverfahren Reinforcement Learning . . . 16
- 3.3. Deep Learning . . . 19
- 3.4. Lösungsverfahren Deep Reinforcement Learning . . . 21

4. Methoden zur Regelextraktion **23**
- 4.1. Decision Tree . . . 23
- 4.2. DeepRED . . . 25

5. Aufbau Experimente **26**
- 5.1. Reinforcement Learning Probleme . . . 26
- 5.2. Implementierung . . . 28
- 5.3. Leistung Deep Reinforcement Learning Modelle . . . 29

6. Experimente Regelextraktion aus Deep Reinforcement Learning Modellen **30**
- 6.1. Validierung: XOR Problem . . . 30
- 6.2. Decision Tree . . . 32
- 6.3. DeepRED . . . 54

7. Zusammenfassung und Ausblick **61**

Abbildungsverzeichnis **64**

Tabellenverzeichnis **65**

A. Anhang **66**

Literaturverzeichnis **86**

Kurzfassung

Deep Reinforcement Learning Modelle haben sich in der Vergangenheit als leistungsstarke Algorithmen zur Lösung von komplexen Entscheidungsproblemen erwiesen. Diese weisen aufgrund der hohen Komplexität ein geringes Maß an Interpretierbarkeit auf und werden als Blackbox bezeichnet. Die Opazität solcher Modelle, auch Orakel genannt, reduziert die Akzeptanz in vielen Anwendungsbereichen. Zur Erhöhung der Akzeptanz von Deep Reinforcement Learning Modellen in den Bereichen liegt es im Interesse der Wissenschaft, diese Modelle näher zu beleuchten, um komplexe Entscheidungsprozesse aus dieser Blackbox für den Menschen verständlicher darzustellen.

In der vorliegenden Arbeit untersuchen wir die Interpretierbarkeit von verschiedenen Deep Reinforcement Learning Modellen in OpenAI-Gym Problemen. Mithilfe von Decision Trees extrahieren wir Regeln aus den Daten der Interaktion des Modells mit seiner Umgebung, um so die Entscheidungen nachvollziehen zu können. Wir stellen einen weiteren Ansatz vor, indem wir die Entscheidungen des Orakels mit einem Deep Neural Network lernen und Regeln mit dem DeepRED Modell extrahieren.

Im Rahmen dieser Arbeit konnten Regeln durch Decision Trees gefunden werden, welche die Entscheidungen des Orakels erklären und die Umgebungen lösen. Die Regelextraktion aus einem für ein Reinforcement Learning Problem angelernten Deep Neural Network mithilfe des DeepRED Modells war nur bedingt erfolgreich. Es konnten teilweise Regeln gefunden werden, welche zur Lösung eines Problems geführt haben. Diese waren jedoch nur partiell interpretierbar.

1. Einleitung

1.1. Motivation

Das Deep Reinforcement Learning (**DRL**) ist eines der vielversprechendsten Bereiche des Machine Learnings (**ML**) und konnte zahlreiche Erfolge in den letzten Jahren verzeichnen (Silver u. a., 2017b). Der Begriff des DRLs wird aus dem Deep Learning (**DL**) und dem Reinforcement Learning (**RL**) zusammengesetzt, welche ebenfalls Teilgebiete des MLs sind. Einen der größten Erfolge erzielte AlphaGo (Silver u. a., 2017a) mit dem Sieg über den damaligen Weltmeister Lee Sedol (GOBASE, 2010) im Spiel *Go* (Cho, 2018). In diesem zweiten historischen Spiel gegen den Weltmeister spielte AlphaGo einen Zug, welcher als nicht menschlich klassifiziert wurde, jedoch der entscheidende und gewinnbringende Zug war (Samek u. a., 2019). Derartige übermenschliche Leistungen wurden auch in anderen Bereichen wie Computerspielen (Mnih u. a., 2015), Poker (Moravcík u. a., 2017), anderen komplexen Brettspielen und auch in Steuerungs- und Reglungsproblemen (Brockman u. a., 2016) reproduziert. Dieser Fortschritt hat die Aufmerksamkeit von Forschern auf sich gezogen, sodass das Verständnis der Funktionalität, das Verhalten des Lernens und die Einsetzbarkeit solcher Modelle genauer analysiert wird (Botvinick u. a., 2019). Ziel von DRL Algorithmen ist es, das Verhalten eines Agenten in einer bestimmten Umgebung zu optimieren. Der Agent erhält als einzige Rückmeldung seiner zu unterschiedlichen Zeitpunkten ausgeführten Aktionen Belohnungen, welcher er maximieren möchte. Dabei folgen die Aktionen stets einer bestimmten Strategie, die der Agent über die Zeit versucht zu optimieren (Lapan, 2018). Die getroffenen Entscheidungen des Agenten sind meist für den Menschen nicht intuitiv und daher schwer interpretierbar. In weniger komplexen Spielen wie Tic-Tac-Toe (Zaslavsky, 1982) ist die gewählte Strategie des Agenten leichter nachzuvollziehen als bei weitaus komplexeren Umgebungen (Wiering u. van Otterlo, 2012).

Unter den bisher wichtigsten Aspekten des DRLs wird die Performance gegenüber der Interpretierbarkeit und dem Verständnis priorisiert. Die Konsequenzen eines gescheiterten Versuchs bei der Gesichtserkennung oder eines falsch übersetzten Textes mithilfe von DRL Algorithmen erweisen sich als unspektakulär und nicht gefährlich, sofern diese keine verheerenden Konsequenzen nach sich ziehen. Beim Einsatz in lebensgefährlichen Situation wie dem autonomen Fahren oder der Erstellung von medizinischen Diagnosen, ist diese Opazität der Modelle ein Entscheidungsgrund für die Ablehnung des Einsatzes solcher Systeme (Mourad Chebila, 2021). Daher ist die Intransparenz, auch bezeichnet als Blackbox, wichtig zu erforschen, um so mehr Vertrauen in künstliche Intelligenzen zu erhalten (Samek u. a., 2019). Dazu werden Entscheidungen eines transparenten Systems gerechtfertigt, sodass diese als fair und ethisch erachtet werden können (Puiutta u. Veith, 2020).

1.2. Problemstellung und Aufbau

Die Theorie der Explainable Artificial Intellegence (**XAI**) ist kein neues Gebiet, sondern ist so alt wie die künstliche Intelligenz (engl. Artifical Intellegence) (**AI**) selbst. Frühere Modelle der AI wurden aufgrund ihrer Transparenz als Glasbox bezeichnet. Hingegen sind heutige Algorithmen, wie zum Beispiel die des DRLs, auf komplexen probabilistischen Modellen aufgebaut, welche für den Menschen nicht durchschaubar sind (Holzinger, 2018). Daher stellen wir uns die Frage, ob es trotz so einer hohen Komplexität möglich ist, für den Menschen interpretierbare Regeln aus DRL Algorithmen zu extrahieren. XAI im RL wird auch als Explainable Reinforcement Learning (**XRL**) bezeichnet. In der vorliegenden Arbeit wollen wir uns mit XRL beschäftigen.

Dazu verwenden wir die gewonnenen Modelle des vorangegangenen Praxisprojekts (Oedingen, 2021), welche in den durch das OpenAI-Gym (Brockman u. a., 2016) gegebenen Umgebungen trainiert wurden. Eine generelle Einführung in das ML ist durch Kapitel 2 gegeben. In Kapitel 3 konkretisieren wir die vorgestellten Verfahren des MLs und führen die erforderliche Theorie ein, bevor wir die Verfahren zur Regelextraktion in Kapitel 4 vorstellen. Weiterhin wollen wir mit Kapitel 5 einen kurzen Überblick über die zu lösenden Umgebungen und die dafür eingesetzten Modelle geben. Letztlich stellen wir die durchgeführten Experimente mit den beschriebenen Verfahren zur Regelextraktion in Kapitel 6 vor und ziehen ein Fazit dieser Arbeit mit Kapitel 7.

1.3. Aktueller Forschungsstand

Aufgrund der zahlreichen Erfolge des DRLs in den letzten Jahren ist die Motivation für das Verständnis solcher Modelle gestiegen. Daraus resultierend existieren zahlreiche Forschungsarbeiten, die sich mit dem XRL beschäftigen (Heuillet u. a., 2021). Grundsätzlich wird bei den verschiedenen Ansätzen zwischen Post-Hoc und transparenten Methoden differenziert. Transparente Methoden besitzen anders als DRL Algorithmen eine durchschaubare Architektur und liefern ein hohes Maß an Interpretierbarkeit. Hingegen werden Post-Hoc Methoden als Ansätze, welche Erklärungen für eine spezifische Lösung und nicht für das gesamte Modell liefern, beschrieben (Holzinger, 2018). Einige interessante Ansätze sind zum Beispiel durch die Entwicklung einer höheren Programmiersprache (Verma u. a., 2018), die für den Menschen interpretierbare Regeln erstellt oder die Erklärbarkeit durch Generierung von Heatmaps (Greydanus u. a., 2017) gegeben. Weitere Arbeiten zum Thema XAI, beziehungsweise XRL, sind in folgenden Übersichtsartikeln hinterlegt: (Puiutta u. Veith, 2020; Heuillet u. a., 2021). Trotz vieler Forschungsarbeiten lässt sich zu diesem Zeitpunkt keine allgemeingültige und formale Definition für die Theorie des XRLs festlegen. Daher werden Forscher weiterhin daran interessiert sein, diese Opazität zu durchbrechen (Dazeley u. a., 2021).

1.4. Forschungsfragen

Aufgrund der Problemstellung und des aktuellen Forschungsstands, formulieren wir das Ziel der vorliegenden Arbeit noch präziser. Daraus resultierend ergeben sich Forschungsfragen, welche wir mit dieser Arbeit beantworten wollen. Der aktuelle Forschungsstand stellt ein offenes Gebiet dar, mit dem Ziel eine allseits anerkannte Methode zu finden, welche die Blackbox Modelle durchleuchtet (Holzinger, 2018).

Im Rahmen dieser Arbeit wollen wir uns dieser Problematik annehmen. Dazu stellen wir zwei Ansätze vor, zum einen die Regelextraktion aus einem Decision Tree (**DT**) (Kozak, 2018) und aus einem Deep Neural Network (**DNN**) (Kruse u. a., 2015, S.5 ff.). Beide Ansätze basieren auf den erzeugten Daten der Interaktion des DRL Modells mit der Umgebung und stellen Probleme des Supervised Learnings dar. Als Umgebungen bezeichnen wir die von uns untersuchten Probleme des OpenAI-Gyms. Die Entscheidungen des DRL Modells werden, wie im späteren Verlauf dieser Arbeit vorgestellt, durch ein oder mehrere DNNs bestimmt. Im ersten Ansatz nutzen wir die Entscheidungen des DRL Modell internen DNNs und die Observationen der Umgebung, um einen DT auf diesen Daten zu trainieren. Da es sich bei einem DT um ein transparentes Modell handelt (Heuillet u. a., 2021), können Regeln sofort extrahiert werden. Der zweite Ansatz beinhaltet die Erstellung eines weiteren DNNs, welches auf Grundlage der Interaktionsdaten trainiert wird. Die Regelextraktion erfolgt dabei über das DeepRED Modell (Zilke u. a., 2016). Daher stellen wir uns die folgenden Forschungsfragen:

1. Ist es möglich, die Entscheidungen eines komplexen DRL Modells durch einen DT darzustellen?
   a) Wird das vorliegende Problem mit den extrahierten Regeln gelöst?
   b) Ab welcher Tiefe liefert ein DT Regeln, welche das vorliegende Problem lösen?
   c) Sind die gewonnenen Regeln interpretierbar und für den nicht fachkundigen Leser verständlich?
   d) Stellen die identifizierten Regeln qualitative Ergebnisse im Vergleich zu Regeln aus vorherigen Forschungsarbeiten dar?
2. Erlaubt uns das DeepRED Modell, Regeln aus einem separat trainierten DNN zu extrahieren, das auf den Interaktionsdaten des DRL Modell beruht?
   a) Wird das vorliegende Problem mit den extrahierten Regeln gelöst?
   b) Ab welcher initialen Tiefe gilt das vorliegende Problem als gelöst?
   c) Sind die gewonnenen Regeln interpretierbar und für den nicht fachkundigen Leser verständlich?
   d) Stellen auch diese qualitative Ergebnisse im Vergleich zu Regeln aus vorherigen Arbeiten dar?

Im Falle einer negativen Antwort auf einer dieser Fragen, werden wir versuchen die Ursache zu erklären und dafür relevante Daten darzustellen.

2. Einführung Machine Learning

2.1. Überblick

Der Begriff Intelligenz ist eines der Schlagwörter im Bereich des Machine Learnings. Die allgemeine Definition lässt sich als die Fähigkeit, aus Erfahrung zu lernen, abstrakte Probleme zu lösen und subjektives Wissen zur Anpassung an neue Situationen einzusetzen, auffassen. Mithilfe eines Intelligenztests kann diese Fähigkeit anhand eines numerischen Wertes repräsentiert und mit anderen Individuen verglichen werden (Myers u. Hoppe-Graff, 2014, S.400). Im Machine Learning wird der Terminus Intelligenz als eine Optimierung des Verhaltens des jeweiligen Agenten verstanden. Der Unterschied zur allgemeinen Definition der Intelligenz liegt dabei in der kontinuierlichen Verbesserung von artifiziellem Verhalten durch Belohnungen. Die Leistung eines DRL Modells werden als die kumulierten Belohnungen einer Episode bezeichnet. Damit ist die Leistung eines Agenten in einer bestimmten Umgebung vergleichbar zu anderen agierenden Agenten in dieser Umgebung, ähnlich zu Intelligenztests bei Menschen (Lorenz, 2020, S.3).

Ein weiteres wichtiges Leitwort ist die Kreativität von Agenten. Diese wird allgemein als die Fähigkeit interpretiert, neuartige und wertvolle, beziehungsweise nützliche Ideen hervorzubringen (Myers u. Hoppe-Graff, 2014, S.406). Computer sind logikbasierte Maschinen, die genau die Aufgabe erfüllen sollen, welchen man ihnen aufgetragen hat. Kreativität innerhalb eines Algorithmus wird als Indikator für den Einsatz des Machine Learnings genutzt:

> *„Computers aren't supposed to be creative; they're supposed to do what you tell them to do. If you tell them to do is be creative, you get machine learning.“* (Domingos, 2015)

Vor dem Einsatz des Machine Learnings war der Computer als ein Rechenautomat zu verstehen, welcher deterministische Ergebnisse unter Einfluss von festgelegten Regeln erzeugte. Dabei galt ein abwegiges Verhalten von der definierten Vorgehensweise als fehlerhaft und nicht verwertbar. Im Gegensatz dazu stehen die verschiedenen Methoden des Machine Learnings, bei denen es sogar priorisiert wird, andere oder außergewöhnliche Lösungen zu finden. Das Erforschen neuer Möglichkeiten wird als autonome Erkundung bezeichnet und ist nicht als trivial anzusehen. In Unterabschnitt 3.2 werden die Probleme der Erkundung (engl. Exploration) und Ausbeutung (engl. Exploitation), welche sich mit dem vorliegenden Problem beschäftigen, weiter vertieft (Lorenz, 2020, S. 167). Machine Learning lässt sich in drei Kategorien einteilen: Überwachtes Lernen (engl. Supervised Learning), Bestärkendes Lernen (engl. Reinforcement Learning) und Un-überwachtes Lernen (engl. Unsupervised Learning). Diese sind in den nachfolgenden Unterabschnitten definiert (Frochte, 2018, S. 21).

2.2. Supervised Learning

Das Supervised Learning beinhaltet verschiedene Methoden, welche Probleme der Klassifikation und Regression lösen. Im Folgenden wird die Klassifikation verdeutlicht. Das Modell erhält einen Datensatz der folgenden Form: $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$ mit $x_i \in \mathcal{X}$, der Menge der zu klassifizierenden Objekte, und $y_i \in \mathcal{Y} \subseteq \mathbb{R}$, der Menge der Sollwerte, mit $i = 1, \ldots, n$ und $n \in \mathbb{N}$. Die Sollwerte sind als eine diskrete Menge von Klassen zu beschreiben. Wir sind an einer Funktion $f : \mathcal{X} \to \mathcal{Y}$ mit $f \in \mathcal{F}$, der Menge möglicher Funktionen, interessiert, welche die zu klassifizierenden Objekte den Sollwerten zuweist. Dabei ist uns die Zuweisung von $x_i \to y_i$ durch den gegeben Datensatz bereits bekannt. Im Lernvorgang können vorläufig fehlerhafte Ausgaben erzeugt werden. Durch eine Anpassung der Funktion f in die Richtung des vorgegebenen Wertes y_i, wird dieser Fehler über mehrere Iterationen minimiert. Die Anzahl der Iterationen bestimmen wir über eine Teilmenge des vorliegenden Datensatzes. Dieser wird in drei Mengen separiert: Trainingsdaten, Validierungsdaten und Testdaten. Die Anpassung der Funktion f findet dabei über die Menge der Trainingsdaten statt. Testdaten sind unabhängig von den Trainingsdaten und prüfen die Qualität der Funktion mit Daten, welche dem Modell noch nicht bekannt sind. In einem letzten Schritt werden die Validierungsdaten präventiv genutzt, um Hyperparameter abzustimmen und das Overfitting der Daten zu verhindern. Häufig genutzte Modelle sind künstliche neuronale Netze, welche ihren Fehler mithilfe von Backpropagation minimieren (Lorenz, 2020, S.4).

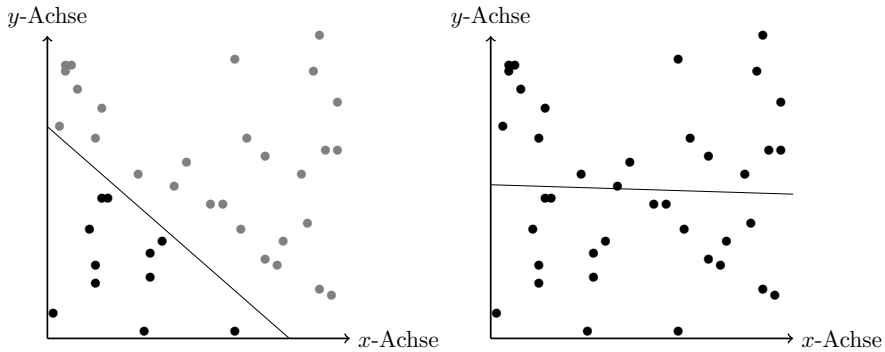
Die Regression läuft analog zur Klassifikation ab, mit dem Unterschied, dass die Sollwerte kontinuierlich sind. Mithilfe von Regression ist es möglich, die Beziehung zwischen den Eingangs- und Ausgangsvariablen in Form eines kontinuierlichen Wertes herzustellen, beziehungsweise zu schätzen. Wir sind an einer Funktion $g : \mathcal{X} \to \mathcal{Z}$ interessiert mit $\mathcal{X}$ wie oben definiert, $\mathcal{Z} \subseteq \mathbb{R}^n$ einer Menge von kontinuierlichen Werten und $g \in \mathcal{F}$ einer aus dem Funktionsraum enthaltenen Funktion. Dabei unterliegt die Regression einem Minimierungsproblem, bei welchem die Differenz zwischen den Sollwerten und den durch die Näherung approximierten Werten minimiert wird. Formal bestimmen wir den Mean-Squared-Error (**MSE**) als

$$\frac{1}{n} \sum_{i=1}^{n} \left(\mathcal{Z}_i - g(\mathcal{X}_i)\right)^2 = \min_{\vec{\alpha}} \|\vec{z} - \vec{g}\|_2^2 , \tag{2.1}$$

mit $\vec{\alpha} \in \mathbb{R}^m$ demjenigen Parameter, welcher die Summe minimiert und $\vec{z}, \vec{g} \in \mathbb{R}^n$, Vektoren aus den Soll- und angenäherten Werten. In der Realität kann keine perfekte Approximation gefunden werden, da zu viele Einflüsse existieren, welche die Qualität beeinflussen (Frochte, 2018, S.23).

In Abbildung 2.1 können wir den Unterschied zwischen einer linearen Klassifikation und linearen Regression deutlich erkennen. Während die Klassifikation versucht, Objekte zu separieren, ist es das Ziel der Regression, eine Beziehung zwischen den Sollwerten und den approximierten Werten zu finden. Diese Unterscheidung ergibt Sinn, da wir zwischen der Zuordnung von Klassen bei der Klassifikation und der Voraussa-

ge stetiger Werte bei der Regression unterscheiden. Beide Methoden können auch in nicht-linearer Form auftreten, dabei bleiben die vorgestellten Ansätze zu den Methoden gleich.



(a) Lineare Klassifikation (b) Lineare Regression

Abbildung 2.1.: Supervised Learning Methoden - Eigene Erstellung LaTeX

2.3. Reinforcement Learning

Bei vielen Problemen ist eine optimale Strategie nicht bekannt oder lässt sich nur schwer aufgrund deren Komplexität formulieren. Jedoch sind wir uns bewusst, welches ein erstrebenswertes Ziel ist. Diese Art von Problemen lassen sich mithilfe des Reinforcement Learnings lösen. Unter der Verwendung von Agenten-Systemen, die mit ihrer jeweiligen Umgebung interagieren, lassen sich Strategien zur Lösung des Problems formulieren (Frochte, 2018, S.24).

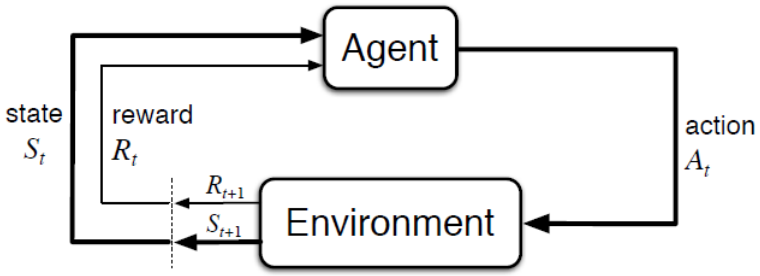


Abbildung 2.2.: Agent-Umgebungsinteraktion (Sutton u. Barto, 1998, S.54)

Der Agent ist der Akteur des Reinforcement Learnings. Dieser versucht zu gegebenen Zeitpunkten, $t = 0, 1, \ldots m$ mit $m \in \mathbb{N}$ von dem derzeitigen Zustand der Umgebung $S_t \in \mathcal{S}$, wobei $\mathcal{S}$ die Menge der möglichen Zustände ist, eine Aktion zu wählen $A_t \in \mathcal{A}(S_t)$, mit $\mathcal{A}(S_t)$ den möglichen Aktionen im Zustand S_t, um so in einen neuen Zustand S_{t+1} zu gelangen. Dabei erhält jede durchgeführte Aktion eine numerische Belohnung $R_{t+1} \in \mathcal{R} \subset \mathbb{R}$. Dieser Ablauf ist in Abbildung 2.2 dargestellt (Sutton u. Barto, 1998, S.53 ff.).

Das Verhalten eines Agenten in einem Zeitpunkt t wird durch seine Strategie $\pi_t\,(a|s)$ bestimmt. Diese Strategie stellt den Zustandsübergang dar, mit der Wahrscheinlichkeit eine Aktion $A_t = a$ in einem Zustand $S_t = s$ zu wählen. Für jeden Zeitpunkt t werden verschiedene Folgezustände S_{t+1} betrachtet, welche der Strategie π_t folgen. Jede Aktion unter Verwendung der Strategie soll die Maximierung der zu erhaltenen Belohnung anstreben (Sutton u. Barto, 1998, S.54).

Im Vergleich zum Supervised Learning wird der Agent im klassischen Reinforcement Learning in seiner Umgebung trainiert und erhält keine Sollwerte, die ihn auf seine Fehler aufmerksam machen. Dabei werden Entscheidungen im Direkten-RL während der Interaktion mit der Umgebung getroffen und sind nicht von Beginn an festgelegt. Den Unterschied zwischen Direktem- und Indirektem-RL erläutern wir in Unterabschnitt 3.2 (Lorenz, 2020, S.4). Eine genauere Spezifikation des Agenten und seiner Umgebung ist in (Oedingen, 2021, S.6 ff.) zu finden.

2.4. Unsupervised Learning

Beim Unsupervised Learning sind wir an einer Repräsentation der Daten interessiert, bei welcher diese in Cluster unterteilt werden. Ein Cluster wird als homogene Gruppe gleichartiger Elemente bezeichnet. Der Unterschied zum überwachten Lernen besteht im Datensatz, welcher keine Sollwerte beinhaltet (Lorenz, 2020, S.4). Der Datensatz besitzt folgende Form: $(x_1, x_2, \ldots, x_n)$ mit $x_i \in \mathcal{X}$, die Menge der zu gruppierenden Objekte, mit $i = 1, \ldots, n$ und $n \in \mathbb{N}$. Sei $\mathcal{M} = \bigcup_{j=0}^{k} \mathcal{M}_j$, die Menge aller Objekte unseres Datensatzes, und die $\mathcal{M}_j$ paarweise disjunkte Teilmengen, dann suchen wir eine Funktion $f : x_i \to \mathcal{M}_j$. Dabei ist $j = 1, \ldots, k$, $k \in \mathbb{N}$ die Anzahl der Cluster und $f \in \mathcal{F}$, die Menge der möglichen Funktionen. Aufgrund der nicht vorhandenen Sollwerte kann kein Fehler während der Prozedur des Lernalgorithmus berechnet werden. Dadurch ist es nicht möglich, Lösungen des Computers direkt zu bewerten (Frochte, 2018, S.24).

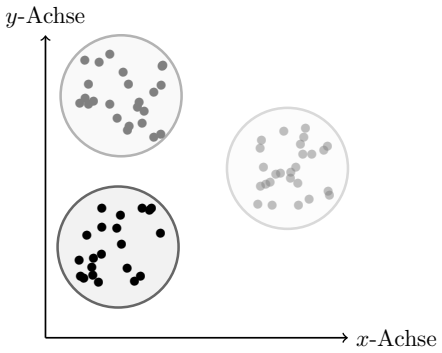


Abbildung 2.3.: Unsupervised Learning Clustering - Eigene Erstellung LaTeX

In Abbildung (2.3) wird die Verwendung des k-Means Algorithmus für ein Clustering-Verfahren genutzt. Pro Cluster, also gefülltem Kreis mit Punkten, gibt es einen Repräsentanten. Die Qualität des Algorithmus wird durch die Summe der Abweichungen der Cluster-Repräsentanten ξ_j in der gewählten Metrik, in unserem Fall die euklidi-

sche Metrik mit der Norm $\rho = 2$, berechnet. Diese sollte minimal sein, sodass ein Minimierungsproblem vorliegt. Mathematisch lässt sich dies unter Verwendung des MSE folgendermaßen formulieren:

$$\begin{aligned} \mathcal{J} &= \sum_{j=1}^{k} \sum_{x_i \in M_j} d(x_i, \xi_j) \\ &= \sum_{j=1}^{k} \sum_{x_i \in M_j} \|x_i - \xi_j\|_2^2. \end{aligned} \tag{2.2}$$

Wobei $i \in \mathbb{N}$ die Laufvariable über die Punkte innerhalb eines Clusters M_j und d die Distanz der gewählten Metrik ist. $\mathcal{J}$ kann als Varianzminimierung interpretiert werden, da die Summe der Varianzen der Cluster minimiert und jeder Punkt durch den Einsatz der euklidischen Distanz dem am nächsten liegenden Repräsentanten zugeordnet wird (Frochte, 2018, S.429). Offensichtlich gibt es noch weitere Algorithmen des Unsupervised Learnings (Barlow, 1999). Das Unsupervised Learning wurde lediglich der Vollständigkeit halber erwähnt, wird in dieser Arbeit jedoch nicht weiter benötigt.

3. Grundlagen

In diesem Kapitel wollen wir die grundlegenden und notwendigen Methoden eines DRL-Modells vorstellen. Dazu präzisieren wir insbesondere die Sektionen des Supervised- und Reinforcement Learnings aus Kapitel 2. Durch einen Markov-Entscheidungsprozess wollen wir ein RL-Problem formulieren. Basierend auf den Formulierungen stellen wir Grundzüge der Lösungsansätze vor, wobei wir auf signifikante Eigenschaften einzelner Methoden eingehen. Weiterhin bilden wir die Brücke zwischen RL und DRL Algorithmen mithilfe des DLs. Letztlich beschreiben wir kurz die von uns eingesetzten DRL Modelle.

3.1. Markov-Entscheidungsprozess

Durch die Voraussetzungen an die Umgebung und die Verhaltensweise des Agenten formulieren wir ein RL Problem als einen Markov-Entscheidungsprozess (**MEP**). Diese Repräsentation ermöglicht es die von uns untersuchten Steuerungsprobleme als einen vollständig beobachtbaren MEP darzustellen. Eine ausführlichere Beschreibung der vorzustellenden Bereiche ist in (Oedingen, 2021, S.7 ff.) zu finden. Weiterhin existieren partiell beobachtbare MEPs (**PBMEP**), welche nicht weiter berücksichtigt werden.

Definition 3.1.1 (Markov-Entscheidungsprozess)
Sei $\mathcal{M}$ ein Markov-Entscheidungsprozess, welcher durch ein Modell von Entscheidungsproblemen beschrieben wird, dann charakterisieren wir diesen durch ein 5*-Tupel $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma)$:*

- *$\mathcal{S}$ ist ein Zustandsraum mit $S_t \in \mathcal{S}$, wobei jeder Zustand die Markov-Eigenschaft, siehe Definition 3.1.2, erfüllt.*
- *$\mathcal{A}$ ist ein Aktionsraum mit $A_t \in \mathcal{A}(S_t)$, wobei die möglichen Aktionen gegebenenfalls vom Zustand abhängen.*
- *$\mathcal{P}$ ist eine Zustandsübergangmatrix mit $\mathcal{P}^a_{ss'} = \mathbb{P}\left[S_{t+1} = s' | S_t = s, A_t = a\right]$, welche die Wahrscheinlichkeit in den Zustand s' zu gelangen, wenn der Agent in Zustand s ist und die Aktion a wählt, beschreibt.*
- *$\mathcal{R}$ ist eine Belohnungsfunktion mit $\mathcal{R}^a_s = \mathbb{E}\left[R_{t+1} | S_t = s, A_t = a\right] \in \mathbb{R}$, welche die zu erwartende Belohnung für eine Aktion a im Zustand s berechnet.*
- *γ ist ein Diskontierungsfaktor mit $\gamma \in \mathbb{R}^+$, wobei $0 < \gamma \leq 1$ gilt.*

Ein MEP stellt ein Optimierungsproblem dar. Der Agent kann sich in diesem nur optimal verhalten, genau dann, wenn er sich ab dem aktuellen Zeitpunkt t optimal verhält. Daraus resultierend folgt die Irrelevanz der Vergangenheit und damit die Definition der Markov-Eigenschaft als ein Merkmal von großer Bedeutung in einem MEP (François-Lavet u. a., 2018, S.17).

Definition 3.1.2 (Markov-Eigenschaft)
Sei $S_t \in \mathcal{S}$ der aktuelle Zustand in einem MEP, dann wird die Umgebung durch diesen Zustand vollständig charakterisiert, sodass alle relevanten Informationen durch den aktuellen Zustand dargestellt werden und die Historie zu vernachlässigen ist:

$$\mathbb{P}\left[S_{t+1}|S_t\right] = \mathbb{P}\left[S_{t+1}|S_1, \ldots, S_t\right]. \tag{3.1}$$

Das durch den MEP beschriebene Modell ist vollständig beobachtbar und in jedem Zustand gilt die Markov-Eigenschaft. Daher sind die kumulierten Wahrscheinlichkeiten der möglichen Aktionen in einem Zustand $S_t \in \mathcal{S} \setminus \{S_\mathcal{T}\}$ gleich eins (Silver, 2015). $S_\mathcal{T}$ beschreibt den terminierenden Zustand. Mit der Zustandsübergangsmatrix $\mathcal{P}$ beschreiben wir dies formal als (Sutton u. Barto, 1998, S.57 ff.):

$$\mathcal{P} = \begin{bmatrix} \mathcal{P}_{11} & \ldots & \mathcal{P}_{1n} \\ \vdots & \ddots & \vdots \\ \mathcal{P}_{n1} & \ldots & \mathcal{P}_{nn} \end{bmatrix}, \tag{3.2}$$

wobei für jeden Eintrag $\mathcal{P}_{ij}$ mit $i, j = 1, \ldots, n$ und $n \in \mathbb{N}$ gilt:

$$\sum_{j=1}^{n} \mathcal{P}_{ij} = \sum_{j=1}^{n} \mathbb{P}\left[S_{t+1} = j | S_t = i\right] = 1. \tag{3.3}$$

Definition 3.1.3 (Strategie)
Sei $A_t \in \mathcal{A}(S_t)$ die Wahl einer Aktion des Agenten, dann beschreiben wir eine Strategie als deterministisch oder stochastisch:

- *eine deterministische Strategie $\pi(s) : \mathcal{S} \to \mathcal{A}$ wählt die selbe Aktion für einen gegebenen Zustand.*
- *eine stochastische Strategie $\pi(s, a) : \mathcal{S} \times \mathcal{A} \to [0, 1]$ beschreibt die vollständige Wahrscheinlichkeitsverteilung über die möglichen Aktionen $A_t \in \mathcal{A}(S_t)$.*

Eine Strategie beschreibt vollständig das Verhalten des Agenten und verfolgt zu jedem Zeitpunkt $t \in [0, \mathcal{T})$ das Ziel der langfristigen Maximierung der zu erwartenden Belohnungen. Dabei wollen wir es dem Agenten ermöglichen, alle in der Zukunft gelegenen Belohnungen zu berücksichtigen. Wir setzen voraus, dass sich das vorliegende RL Problem in Episoden unterteilen lässt und diese einen terminierenden Zustand $S_\mathcal{T}$ besitzen (Lapan, 2018, S.30 ff.).

Definition 3.1.4 (Rückgabe)
Sei $t \in \mathbb{N}$ der aktuelle Zeitpunkt, dann beschreiben wir die Rückgabe als die diskontierte zu erwartende Belohnung ab t bis zum finalen Zeitpunkt $\mathcal{T}$:

$$G_t = \sum_{k=0}^{\mathcal{T}} \gamma^k R_{t+k+1}. \tag{3.4}$$

Der Agent wird für einen Wert des Diskontierungsfaktor von $\gamma \to 0$ kurzsichtig und belohnt Aktionen A_t sofort, um R_{t+1} zu maximieren. Hingegen wird der Agent für einen Wert von $\gamma \to 1$ weitsichtig, sodass zukünftige Belohnungen weiterhin stark berücksichtigt werden. Unterschieden wird zwischen einem episodischen und kontinuierlichen Problem, beschrieben in (Oedingen, 2021, S.9).

Das Ziel des Agenten ist es eine Strategie $\pi(s, a) \in \Pi$, mit Π der Menge der Strategien, zu finden, welche die Rückgabe maximiert. Dazu müssen wir definieren „wie gut es ist“ sich einem Zustand zu befinden (Sutton u. Barto, 1998, S.70 ff.).

Definition 3.1.5 (Wertfunktion)
Sei $S_t \in \mathcal{S}$ der aktuelle Zustand und $A_t \in \mathcal{A}(S_t)$ eine Aktion in diesem Zustand, dann beschreiben wir eine Wertfunktion als die Auskunft über die Rückgabe, welche wir ab dem aktuellen Zeitpunkt t erwarten können. Wir unterscheiden zwischen einer Zustands- und Aktionswertfunktion:

- *eine Zustandswertfunktion $v_\pi(s) : \mathcal{S} \to \mathbb{R}$ liefert die in Abhängigkeit der Strategie π definierte Rückgabe, wenn wir im Zustand S_t starten und anschließend der Strategie π folgen:*

$$v_\pi(s) = \mathbb{E}_\pi \left[G_t | S_t = s \right] = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \middle| S_t = s \right]. \tag{3.5}$$

- *eine Aktionswertfunktion $q_\pi(s, a) : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ liefert die in Abhängigkeit der Strategie π definierte Rückgabe, wenn der Agent im Zustand S_t startet, eine Aktion A_t wählt und anschließend der Strategie π folgt:*

$$q_\pi(s, a) = \mathbb{E}_\pi \left[G_t | S_t = s, A_t = a \right] = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \middle| S_t = s, A_t = a \right]. \tag{3.6}$$

Eine fundamentale Beziehung im RL stellt die Dekomposition der Wertfunktion dar. Dies resultiert in einer Reduktion der Komplexität durch die Zerlegung in Teilprobleme (Sutton u. Barto, 1998, S.70 ff.).

Definition 3.1.6 (Bellman-Gleichung)
Sei $S_t \in \mathcal{S}$ der aktuelle Zustand, dann können wir eine Zustandswertfunktion rekursiv in eine direkte Belohnung R_{t+1}, plus den diskontierten Wert des Folgezustands $\gamma v_\pi(s')$ darstellen. Dabei folgen die Entscheidungen stets einer Strategie π:

$$\begin{aligned} v_\pi(s) &= \mathbb{E}_\pi \left[G_t | S_t = s \right] \\ &= \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \middle| S_t = s \right] \\ &= \mathbb{E}_\pi \left[R_{t+1} + \gamma \sum_{k=0}^{\infty} \gamma^k R_{t+k+2} \middle| S_t = s \right] \\ &= \mathbb{E}_\pi \left[R_{t+1} + \gamma v_\pi(s') | S_t = s \right]. \end{aligned} \tag{3.7}$$

In Definition 3.1.6 haben wir die rekursive Darstellung der Zustandswertfunktion gezeigt, welche analog für die Aktionswertfunktion formuliert werden kann. Offensichtlich existieren die Wertfunktionen in Abhängigkeit voneinander und lassen sich als eine Ein-Schritt-Voraussicht schreiben, sodass wir die Zustandswertfunktion als

$$v_\pi(s) = \sum_{a \in \mathcal{A}(s)} \pi(a|s) q_\pi(s,a) \tag{3.8}$$

$$= \sum_{a \in \mathcal{A}(s)} \pi(a|s) \left(\mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_\pi(s') \right) \tag{3.9}$$

und die Aktionswertfunktion als

$$q_\pi(s,a) = \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_\pi(s') \tag{3.10}$$

$$= \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a \sum_{a' \in \mathcal{A}(s')} \pi(a'|s') q_\pi(s',a') \tag{3.11}$$

formulieren können. Mithilfe der Dekomposition und der Voraussichten können wir theoretisch optimale Lösungen für einen MEP definieren (Silver, 2015).

Definition 3.1.7 (Optimale Wertfunktion)
Gegeben sei eine Wertfunktion aus Definition 3.1.5, dann definieren wir eine optimale Wertfunktion als diejenige, welche die höchstmögliche Leistung in einem MEP liefert.

- *Eine optimale Zustandswertfunktion bestimmt die bestmögliche Zustandswertfunktion über allen Strategien:*

$$v_*(s) = \max_{\pi \in \Pi} v_\pi(s) \tag{3.12}$$

$$= \max_{a \in \mathcal{A}(s)} \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_*(s'). \tag{3.13}$$

- *Eine optimale Aktionswertfunktion bestimmt bestmögliche Aktionswertfunktion über allen Strategien:*

$$q_*(s,a) = \max_{\pi \in \Pi} q_\pi(s,a) \tag{3.14}$$

$$= \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a \max_{a' \in \mathcal{A}(s')} q_*(s',a'). \tag{3.15}$$

In einem MEP können mehrere optimale Lösungen existieren, da für unterschiedliche Trajektorien $\tau = (S_t, A_t, S_{t+1}, \ldots, S_\mathcal{T}) \sim \pi_*$ die selbe Belohnung erzielt werden kann. Die optimale Wertfunktion muss nicht mehr alle Aktionen in einem Zustand betrachten $\pi(a|s)$, sondern nur die Aktion wählen, welche die Belohnung maximiert (Silver, 2015; Sutton u. Barto, 1998, S.75 ff.).

Die Leistung einer Strategie legen wir mithilfe der zu erwartenden Belohnung fest, welche die jeweilige Strategie in der Umgebung erzielt. Wir bezeichnen eine Strategie π' besser als eine Strategie π, wenn gilt:

$$v_{\pi'}(s) \geq v_{\pi}(s), \forall s \in \mathcal{S}. \tag{3.16}$$

Mit Gleichung (3.16) und Definition 3.1.7 formulieren wir das Theorem über die Existenz einer optimalen Strategie (Silver, 2015).

Theorem 3.1.1 (Existenz Optimale Strategie)
Sei $\mathcal{M}$ ein beliebiger MEP, dann gilt für $\mathcal{M}$:

- $\pi_* \geq \pi, \forall \pi \in \Pi$*, mit der Existenz einer optimale Strategie π_*, welche besser als oder gleich allen anderen Strategien ist.*
- $v_{\pi_*}(s) = v_*(s)$ *und* $q_{\pi_*}(s,a) = q_*(s,a)$*, sodass alle optimalen Strategien die Eigenschaft der optimalen Wertfunktionen besitzen.*

Mit Theorem 3.1.1 und Definition 3.1.7 können wir eine optimale Strategie beschreiben.

Definition 3.1.8 (Optimale Strategie)
Sei $\pi(a|s)$ eine Strategie, dann beschreiben diese als optimal, wenn der Agent über die optimale Aktionswertfunktion $q_*(s,a)$ maximiert:

$$\pi_*(a|s) = \begin{cases} 1 & \text{wenn } a = \arg\max_{a \in \mathcal{A}} q_*(s,a) \\ 0 & \text{sonst.} \end{cases} \tag{3.17}$$

Letztlich können wir einen MEP als gelöst bezeichnen, wenn wir eine optimale Wertfunktion gefunden haben. Instinktiv sind wir an der optimalen Aktionswertfunktion interessiert, da diese die bestmögliche Aktion in einem Zustand liefert. Eine optimale Strategie haben wir gefunden, wenn wir in jedem Zustand die Aktion wählen, welche die Rückgabe maximiert (Silver, 2015).

In vielen praxisrelevanten Problemen sind optimale Lösungen nicht erreichbar, da reale Modelle zu viele Einflüsse besitzen, um von einer perfekten oder optimalen Lösung zu sprechen. Wie auch schon ein berühmter Physiker feststellte:

> *„The universe doesn't allow perfection."* (Hawking, 1988)

Daher sind es Abstraktionen von komplexen Problemen, beziehungsweise nur deren stark vereinfachte Modelle, welche wir in der Lage sind, als optimal gelöst zu bezeichnen. Unter Berücksichtigung der von uns untersuchten Steuerungsproblemen gilt eine Umgebung als gelöst, wenn ein gewisser Schwellwert überschritten wird.

3.2. Lösungsverfahren Reinforcement Learning

In einem RL Problem, welches durch einen MEP formuliert werden kann, differenzieren wir zwischen verschiedenen Lösungsmethoden. Um den Umfang dieser Arbeit nicht zu überschreiten, werden die Lösungsmethoden nur kurz vorgestellt.

Die folgende Abbildung gibt Auskunft über die Klassifikation von RL Algorithmen. An dieser wollen wir uns orientieren und Bezug auf einige Charakteristiken nehmen.

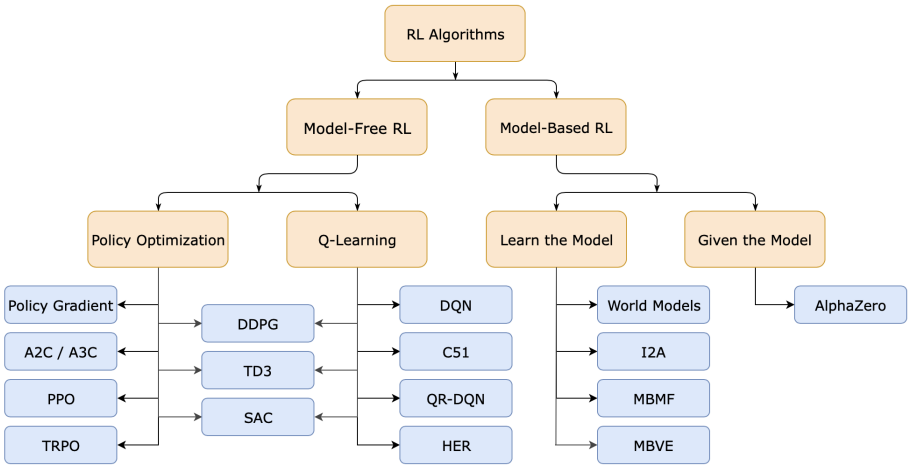


Abbildung 3.1.: Taxonomie Reinforcement Learning Algorithmen (Achiam, 2018)

Mit Abbildung 3.1 klassifizieren wir RL Algorithmen als modellbasierte (engl. model-based) oder modellfreie (engl. model-free) Algorithmen. Der Unterschied ist durch das Wissen über die Dynamik des Systems gegeben. Ein modellbasierter Algorithmus verwendet ein vermeintlich approximiertes Modell eines MEPs zur Lösung des Problems. Dieses Modell beinhaltet das Wissen über eine geschätzte Zustandsübergangsmatrix und Belohnungsfunktion. Hingegen basiert ein modellfreier Algorithmus auf der Unwissenheit der Umgebung. Das Modell wird durch Interaktion des Agenten mit der Umgebung erzeugt (Hao u. a., 2020, S.127 ff.).

Wir verwenden Methoden der Dynamischen Programmierung (**DP**), wenn das Modell für das jeweilige Problem verfügbar ist. Dazu überführen wir die Bellman-Gleichung aus Definition 3.1.6 in eine iterative Aktualisierung und planen einen Pfad in dem Modell. Weiterhin wird ein alternierender Prozess, auch Generalisierte-Strategie-Iteration (**GSI**) genannt, zwischen der Evaluation und Verbesserung einer Strategie ausgeführt. Mit Banachs Fixpunkt Theorem können wir eine γ-Kontraktion für diesen Vorgang zeigen (Silver, 2015). Bei mehrfacher Anwendung des Bellmanschen Optimalitätsoperator erhalten wir eine Cauchy-Folge, welche gegen den Fixpunkt, sprich die optimale Zustandswertfunktion, konvergiert (Fedorenko, 1969). Die von uns eingesetzten Modelle lassen sich als modellfreie Algorithmen einordnen. Daher werden modellbasierte Algorithmen nicht weiter berücksichtigt, sind jedoch in (Oedingen, 2021, S.22 ff.) ausführlich beschrieben.

Erste Ansätze zur Lösung von RL Problemen, bei welchen kein Modell verfügbar ist, sind durch die Monte-Carlo (**MC**) Methoden gegeben. Unter der Voraussetzung einer episodischen Aufgabe können wir die Approximationen der Wertfunktionen definieren (Silver, 2015).

Definition 3.2.1 (Monte-Carlo Sampling)
Sei $n \in \mathbb{N}$ die Anzahl der gesammelten Trajektorien $\tau = (S_t, A_t, S_{t+1}, \ldots, S_\mathcal{T}) \sim \pi$, dann werden die Wertfunktionen für ein genügend großes n approximiert.

- *Die approximierte Zustandswertfunktion mit der Rückgabe in Abhängigkeit einer Trajektorie approximieren wir als:*

$$V_\pi(s) = \mathbb{E}_\pi \left[G_t | S_t = s \right] \approx \frac{1}{n} \sum_{e=1}^{n} G_t(\tau). \tag{3.18}$$

- *Die approximierte Aktionswertfunktion mit der Rückgabe in Abhängigkeit einer Trajektorie approximieren wir als:*

$$Q_\pi(s, a) = \mathbb{E}_\pi \left[G_t | S_t = s, A_t = a \right] \approx \frac{1}{n} \sum_{e=1}^{n} G_t(\tau). \tag{3.19}$$

Mit dem Gesetz der großen Zahlen (Bewersdorff, 2011, S.109 ff.) folgt für $n \to \infty$: $V_\pi(s) \to v_\pi(s)$ und $Q_\pi(s, a) \to q_\pi(s, a)$ (Silver, 2015).

Aus Definition 3.2.1 ergibt sich ein Dilemma in modellfreien RL Algorithmen, namentlich dem Kompromiss zwischen Exploration und Exploitation. Eine optimale Wertfunktion können wir bestimmen, wenn der Agent alle Pfade kennt und nicht nur jene, welche zum aktuellen Zeitpunkt optimal sind.

Definition 3.2.2 (ϵ-Greedy)
Sei $\epsilon > 0$ mit $\epsilon \in \mathbb{R}$ und $\pi(a|s) > 0$ für alle $s \in \mathcal{S}$ und $a \in \mathcal{A}(s)$, dann wählen wir entweder eine optimale oder zufällige Aktion:

$$\pi(a|s) = \begin{cases} 1 - \epsilon + \frac{\epsilon}{|\mathcal{A}(s)|} & \text{wenn } a_* = \underset{a \in \mathcal{A}(s)}{\arg\max}\, Q(s, a) \\ \frac{\epsilon}{|\mathcal{A}(s)|} & \text{sonst.} \end{cases} \tag{3.20}$$

Resultierend aus Definition 3.2.2 und der gezeigten Greedy in the Limit with Infinite Exploration (**GLIE**) Eigenschaft in (Oedingen, 2021, S.31), können wir eine optimale Wertfunktion für die MC Methoden bestimmen (Sutton u. Barto, 1998, S.124 ff.).

In MC-Methoden werden Trajektorien vom Agenten gesammelt. Dieser verbessert seine Strategie erst nach Beendigung einer Episode, sodass direkte Veränderungen in der Umgebung nicht erkennbar sind. Eine im RL zentrale Methode zur Wahrnehmung von unmittelbaren Ereignissen in der Umgebung ist durch das Temporal Difference Learning (**TDL**) gegeben. TDL Methoden lernen direkt aus ihrer gesammelten Erfahrung, sprich einer partiellen Trajektorie, um die Wertfunktionen anzupassen (Sutton, 1988).

Definition 3.2.3 (Temporal Difference Learning Aktualisierung)
Sei $s \in \mathcal{S}$ der aktuelle Zustand und $\alpha \in (0, 1]$ die Lernrate, dann stellen wir den TD Error mithilfe der Bellman-Gleichung (3.1.6) dar und führen eine Aktualisierung in Richtung des TD Errors mit Schrittweite α aus:

$$V(s) \leftarrow V(s) + \alpha \left[r + \gamma V(s') - V(s) \right], \tag{3.21}$$

mit $\delta := r + \gamma V(s') - V(s)$, dem TD Error, welcher den temporalen Unterschied wiedergibt.

Analog zu Gleichung (3.21) können wir auch die Aktionswertfunktion als eine Methode des TDLs darstellen. Insbesondere beschreibt die referenzierte Gleichung die TD(0) Methode, welche eine Ein-Schritt-Voraussicht charakterisiert. Für eine unendliche Anzahl von Voraussichten TD(∞) betrachten wir ganze Trajektorien und erhalten wieder eine MC Methode(Sutton u. Barto, 1998, S.153). Des Weiteren existiert eine Methode TD(λ), welche den Kompromiss zwischen MC und TD Methoden darstellt, beschrieben in (Oedingen, 2021, S.37 ff.)

Bei der Lösung eines MEP mithilfe einer TDL Methode unterscheiden wir zwischen On- und Off-Policy Algorithmen. In einem On-Policy Algorithmus trifft und lernt der Agent Entscheidungen, welche einer Strategie π folgen. Anders bei Off-Policy Algorithmen, welche zwei Strategien berücksichtigen. Durch eine Strategie π wird das Verhalten des Agenten beschrieben, während eine weitere Strategie μ entscheidet, welche alternativen Entscheidungen getroffen hätten werden können (Silver, 2015). Im Folgenden betrachten wir den SARSA (Sutton u. Barto, 1998, S.154) und den Q-Learning Algorithmus (Watkins u. Dayan, 1992).

Definition 3.2.4 (SARSA und Q-Learning)

- *SARSA ist ein On-Policy Algorithmus zur Lösung eines MEPs mit einer TDL Methode und aktualisiert die Wertfunktion für einen Zustand $s \in \mathcal{S}$, einer Aktion $a \in \mathcal{A}(s)$ und einer Lernrate $\alpha \in (0, 1]$ wie folgt:*

$$Q(s, a) \leftarrow Q(s, a) + \alpha \left[r + \gamma Q(s', a') - Q(s, a) \right]. \tag{3.22}$$

- *Q-Learning ist ein Off-Policy Algorithmus zur Lösung eines MEPs mit einer TDL Methode und aktualisiert die Wertfunktion für einen Zustand $s \in \mathcal{S}$, einer Aktion $a \in \mathcal{A}(s)$ und einer Lernrate $\alpha \in (0, 1]$ wie folgt:*

$$Q(s, a) \leftarrow Q(s, a) + \alpha \left[r + \gamma \max_{a' \in \mathcal{A}(s')} Q(s', a') - Q(s, a) \right]. \tag{3.23}$$

Im Gegensatz zu Algorithmen, welche ein RL Problem durch die Verbesserung einer Wertfunktion erlernen, existieren Verfahren zur direkten Manipulation der Strategie, auch Strategie Optimierung (engl. Policy Optimization) genannt. Diese erfordern ein grundsätzliches Verständnis von DNNs und sind im Folgenden beschrieben.

3.3. Deep Learning

Das DL stellt einen wesentlichen Bestandteil im DRL dar und wird als Voraussetzung für den DeepRED Algorithmus benötigt. Wir wollen einen kurzen Überblick über die Eigenschaften eines DNNs geben und verweisen für weitere Informationen auf (Kruse u. a., 2015, S.5 ff.).

3.3.1. Deep Neural Network

Ein DNN besteht aus einer Eingabeschicht $x = (x_1, x_2, \dots x_m)^\intercal$, einer oder mehreren versteckten Schichten $h_1, h_2, \dots, h_L$, wobei jede Schicht eine verschiedene Anzahl von Neuronen $(m_1, m_2, \dots, m_L)$ enthalten kann, und einer Ausgabeschicht $y = (y_1, y_2, \dots y_n)^\intercal$. Dieses kann als azyklischer gerichteter Graph (engl. Directed acyclic Graph) (**DAG**) beschrieben werden, indem Neuronen als Knoten und deren Verbindungen als Kanten dargestellt werden (Zhou u. a., 2018). Jeder Knoten der k-ten Schicht ist mit allen Knoten der $k+1$-ten Schicht, ausgeschlossen dem Bias, verbunden, mit $k = 1 \dots L + 1$. Die Gewichte der Kanten sind in Gewichtsmatrizen $W_k \in \mathbb{R}^{i \times j}$, mit i der Anzahl der Neuronen der k-ten Schicht und j der Anzahl der Neuronen der $k + 1$-ten Schicht.

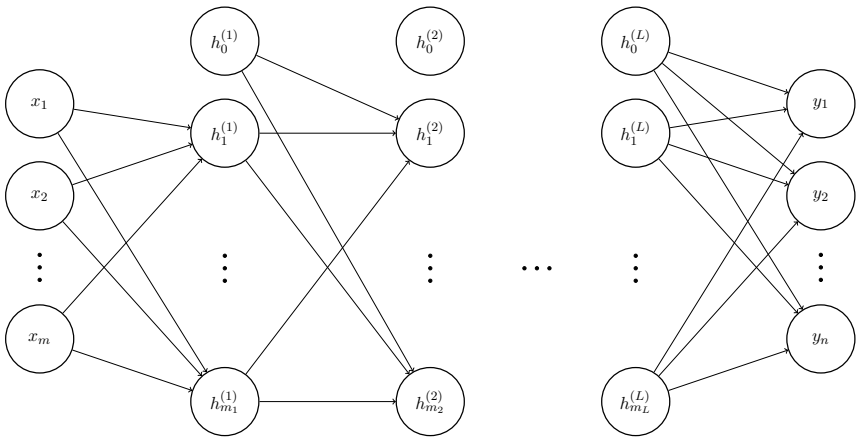


Abbildung 3.2.: DNN mit einer Eingabeschicht von m Neuronen, L versteckten Schichten mit $m1, m2, \dots, m_L$ Neuronen pro Schicht und einer Ausgabeschicht mit n Neuronen. Der Bias wird durch das erste Neuron der einzelnen Schichten repräsentiert. - Eigene Erstellung LaTeX

Eingehende Signale, werden vorwärts durch das Netz geleitet. Die Aktivierung der k-ten Schicht wird mithilfe der $k-1$-ten Schicht bestimmt:

$$h^{(k)} = \sigma\left(W_k \times h^{(k-1)} + h_0^{(k-1)}\right), \tag{3.24}$$

wobei $\sigma : \mathbb{R} \to \mathbb{R}$ eine Aktivierungsfunktion darstellt (Goodfellow u. a., 2016, S.168 ff.). In Abbildung 3.3 haben wir verschiedene Aktivierungsfunktionen dargestellt.

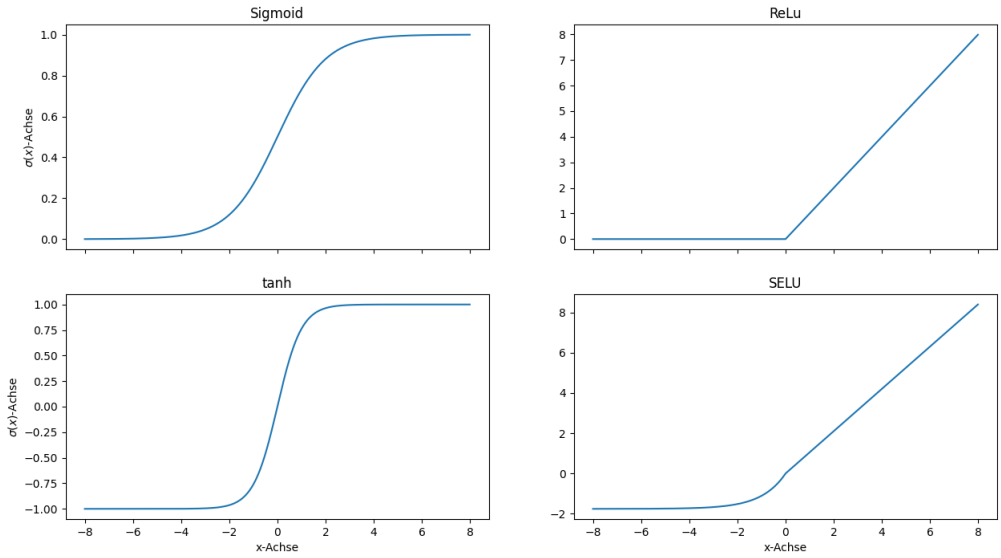


Abbildung 3.3.: Aktivierungsfunktionen nach (Abadi u. a., 2016)

Mithilfe eines DNNs können Probleme des Supervised Learnings gelöst werden. Wir suchen einen Parameter $\theta \in \mathbb{R}^n$, welcher eine Verlustfunktion $\mathcal{L} : \mathbb{R}^n \to \mathbb{R}$ für einen gegebenen Datensatz $\mathcal{D}$ minimiert. Für ein Problem der Regression nutzen wir den in Gleichung (2.1) vorgestellten MSE und für die Klassifikation die Cross-Entropy (**CE**) Methode als:

$$\mathcal{L}(\theta) = \sum_{i=0}^{n} y_i \log(\hat{y}_i), \tag{3.25}$$

mit y dem wahren und $\hat{y}$ dem vorhergesagten Wert. Die Optimierung des Parameters θ erfolgt über das Gradientenabstiegsverfahren (Arens u. a., 2013). Die schrittweise Aktualisierung des Parameters θ formulieren wir als:

$$\Delta\theta = -\alpha \frac{\partial \mathcal{L}(\theta)}{\partial \theta}. \tag{3.26}$$

Alle Funktion innerhalb eines DNNs sind differenzierbar, daher bestimmen wir den Fehler durch den Backpropagation Algorithmus, welcher die rekursive Anwendung der Kettenregel beschreibt (Frochte, 2018, S.184 ff.):

$$\frac{\partial \mathcal{L}(\theta)}{\partial W_k} = \frac{\partial \mathcal{L}(\theta)}{\partial y} \times \frac{\partial y}{\partial h^{(L)}} \times \frac{\partial h^L}{\partial h^{(L-1)}} \times \cdots \times \frac{\partial h^{(k)}}{\partial W_k}. \tag{3.27}$$

Weiterhin existieren verschiedene Optimierer, welche das Gradientenverfahren beschleunigen und verfeinern, beschrieben in (Abadi u. a., 2016).

3.4. Lösungsverfahren Deep Reinforcement Learning

DRL Modelle werden durch die Vereinigung von RL und DL Methoden formuliert. Insbesondere werden DNNs zur Approximation von Wertfunktionen und Strategien genutzt, da oftmals der Aktions- und Zustandsraum zu groß ist, um alle Informationen zu speichern (Silver, 2015). Die folgenden Definitionen und Formulierungen sind detaillierter in (Oedingen, 2021, S.41 ff.) beschrieben.

Definition 3.4.1 (Wertfunktion-Approximation)
Seien $v_\pi(s)$ und $q_\pi(s,a)$ Wertfunktionen in Abhängigkeit einer Strategie π, dann wählen wir einen Parameter $\psi \in \mathbb{R}^n$ als Repräsentanten für die Gewichte eines DNNs, mit $n \in \mathbb{N}$, sodass für die Approximation gilt:

$$v_\pi(s) \approx \hat{v}(s;\psi) \text{ und } q_\pi(s,a) \approx \hat{q}(s,a;\psi). \tag{3.28}$$

Die Optimierung des Parameters erfolgt über eine Verlustfunktion $\mathcal{J} : \mathbb{R}^n \to \mathbb{R}$, welche das stochastische Gradientenabstiegsverfahren[1] (**SGD**) nutzt. Dabei können wir die vorgestellten Methoden aus Definition 3.2.3 und 3.2.4 mithilfe der Approximation von Wertfunktionen formulieren, beschrieben in (Li, 2017).

Definition 3.4.2 (Strategie-Approximation)
Sei $\pi(a|s) \in \Pi$ eine Strategie, dann wählen wir einen Parameter $\theta \in \mathbb{R}^n$ als Repräsentanten für die Gewichte eines DNNs, mit $n \in \mathbb{N}$, sodass wir die Strategie mit $\pi(a|s;\theta)$ optimieren.

Im Gegenteil zur Approximation von Wertfunktionen versuchen wir den Parameter θ so zu wählen, dass er die höchstmögliche Rückgabe liefert, sprich das Maximum (Silver, 2015).

Definition 3.4.3 (Gradienten-Aktualisierung)
Sei $\hat{v}_\pi(s;\psi)$ oder $\hat{q}_\pi(s,a;\psi)$ eine zu approximierende Wertfunktion und $\pi(a|s;\theta)$ eine zu optimierende Strategie.

- *Wir aktualisieren eine Wertfunktion mit:*

$$\Delta\psi = -\frac{1}{2}\alpha\nabla_\psi\mathcal{J}(\psi), \text{ mit } \mathcal{J}(\psi) = \left(\frac{\partial\mathcal{J}(\psi)}{\partial\psi_1},\ldots,\frac{\partial\mathcal{J}(\psi)}{\partial\psi_n}\right)^\intercal. \tag{3.29}$$

- *Wir aktualisieren eine Strategie mit:*

$$\Delta\theta = \alpha\nabla_\theta\mathcal{J}(\theta), \text{ mit } \mathcal{J}(\theta) = \left(\frac{\partial\mathcal{J}(\theta)}{\partial\theta_1},\ldots,\frac{\partial\mathcal{J}(\theta)}{\partial\theta_n}\right)^\intercal. \tag{3.30}$$

[1] Das stochastische Gradientenverfahren unterscheidet sich vom standardmäßigen Gradientenverfahren in der Betrachtung zufälliger Datenpaare, hingegen zum gesamten Datensatz.

Um genauer zu sein, beschreiben wir den REINFORCE Algorithmus (Williams, 1992), wenn wir den Parameter θ wie folgt aktualisieren:

$$\Delta\theta = \alpha\nabla_\theta \log(\pi(A_t|S_t;\theta))(G_t - B_t(S_t)), \tag{3.31}$$

wobei $B_t(S_t)$ eine Grundlinie (engl. Baselines) darstellt. Bei der Verwendung der approximierten Zustandswertfunktion $\hat{v}_\pi(S_t;\psi)$ als Baseline erhalten wir die Vorteilsfunktion.

Definition 3.4.4 (Vorteilsfunktion-Approximation)
Seien $\hat{v}_\pi(S_t;\psi)$ und $\hat{q}_\pi(S_t, A_t;\psi)$ approximierte Wertfunktionen, dann erhalten wir den Vorteil einer Aktion durch:

$$\hat{A}(A_t, S_t;\psi) = \hat{q}_\pi(S_t, A_t;\psi) - \hat{v}_\pi(S_t;\psi). \tag{3.32}$$

Grundsätzlich gibt die Vorteilsfunktion Auskunft über den relativen Vorteil einer Aktion und werden in Akteur-Kritiker (engl. Actor Critic) Methoden verwendet (Hao u. a., 2020, S.164 ff.).

Für die angestellten Experimente zur Regelextraktion aus DRL Algorithmen[2] haben wir folgende Modelle als Orakel genutzt:

- Deep Q-Network (**DQN**) - Das DQN Modell basiert auf der Verwendung des Q-Learning Algorithmus. Dabei werden Wertfunktionen durch ein DNN approximiert. In der Regel wird dieses mit einem Experience Replay eingesetzt, in welchem zufällige Daten vorheriger Episoden gespeichert und wiederum zum Lernen genutzt werden. Eine Optimierung folgt einem in der Vergangenheit liegenden Stand des Q-Networks, auch frozen target network genannt, welches nach C Schritten auf den aktuellen Stand gesetzt wird, mit $C \in \mathbb{N}$ einem Hyperparameter (Mnih u. a., 2013).

- Proximal Policy Optimization (**PPO**) - Das PPO Modell ist ein strategiebasierter Algorithmus, welcher in zwei Varianten existiert: PPO-Penalty und PPO-Clip. Wir haben uns für den PPO-Clip Algorithmus entschieden und stellen diesen vor. Ziel des Algorithmus ist die direkte Optimierung der Strategie mittels SGD. Dazu maximieren wir eine Funktion derart, dass sich die neue Strategie höchstens um den Faktor ϵ von der alten Strategie unterscheidet, mit $\epsilon \in \mathbb{R}$ einem Hyperparameter (Schulman u. a., 2017).

- Advantage Actor Critic (**A2C**) - Das A2C Modell nutzt sowohl strategiebasierte als auch wertbasierte Algorithmen, um ein RL Problem zu lösen. Dieses ist eine synchrone Version des (**A3C**) Modells. Der Unterschied liegt in der Verwendung einer synchronen Parallelisierung mehrerer Agenten. Alle Agenten interagieren mit der Umgebung und treffen unterschiedliche Entscheidungen. Nach Terminierung aller Episoden der Agenten werden die gesammelten Trajektorien zusammengetragen und eine einzelne Strategie formuliert (Mnih u. a., 2016).

Die vorgestellten Modelle beinhalten weitere theoretische Ansätze, welche in (Oedingen, 2021) nachzulesen sind.

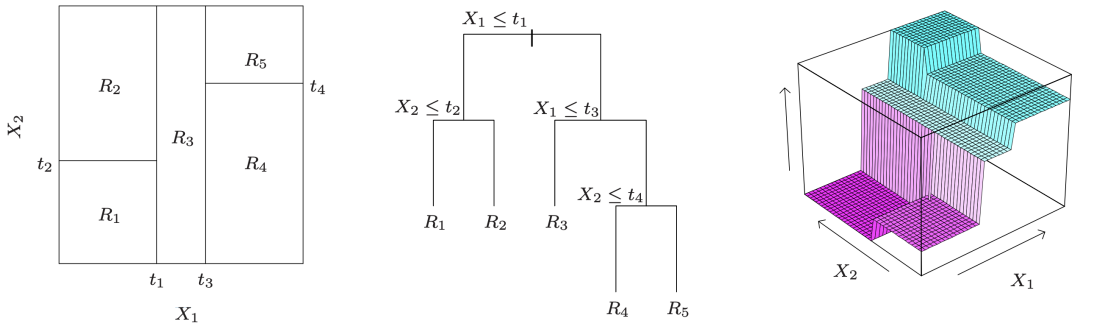
[2]Für alle vorgestellten Modelle befindet sich im Anhang eine Formulierung der Funktionsweise in Form eines Pseudocodes.

4. Methoden zur Regelextraktion

In diesem Kapitel wollen wir die eingesetzten Verfahren zur Regelextraktion vorstellen. Dazu konkretisieren wir insbesondere das Modell eines binären Decision Trees und den DeepRED Algorithmus, welcher zur Regelextraktion aus DNNs dient.

4.1. Decision Tree

Ein DT ist ein Modell zur Klassifikation von Entscheidungsproblemen. Dieses Modell wird durch eine Methode des Supervised Learnings beschrieben und behandelt sowohl Probleme der Klassifikation, als auch der Regression. Dabei repräsentiert ein DT eine Funktion $\mathcal{F} : \mathcal{X} \to \mathcal{Y}$, welche die Eingabewerte $\mathcal{X}$ auf eine Menge von Entscheidungsklassen $\mathcal{Y}$ abbildet. In der Graphentheorie wird ein DT als ein azyklisch gerichteter Graph betrachtet, wobei Entscheidungen als Knoten, Entscheidungspfade als Kanten und Entscheidungsklassen als Blätter dargestellt werden. Die von uns untersuchten DTs sind in Form eines Binärbaums (Necaise, 2010, S.369 ff.). In diesem Fall besitzt jeder Knoten, außer die Blätter, jeweils zwei Kinder. Solche Bäume werden mithilfe des Classification and Regression Trees (**CART**) Algorithmus[1] erstellt (Kozak, 2018, S.3 ff.).



(a) Zweidimensionale Treemap

(b) Binärer Decision Tree mit einer Tiefe von drei

(c) Dreidimensionale Treemap

Abbildung 4.1.: Die drei Abbildungen geben Auskunft über verschiedene Möglichkeiten der Interpretierbarkeit eines binären Decision Trees. In Abbildung (a) und (c) wird eine Einteilung in fünf Bereiche $R_1, R_2, \ldots, R_5$ beschrieben. Die Eingaben X_1 und X_2 werden durch die Entscheidungen des binären Decision Trees $t_1, t_2, \ldots, t_4$ in Abbildung (b) in die Bereiche eingeordnet (Hastie u. a., 2009, S.325).

[1] Der CART Algorithmus wurde von (Breiman u. a., 1984, S.246 ff.) entwickelt. Die Implementierung ist in (Pedregosa u. a., 2011) zu finden.

Unsere Experimente berücksichtigen nur binäre DTs, daher beschreiben wir formal die identifizierten Regeln, beziehungsweise die Entscheidungen eines binären DTs, wie folgt (Hastie u. a., 2009, S.307 ff.):

Definition 4.1.1 (Binärer Decision Tree Split)
Sei $x_i \in \mathbb{R}^n, i = 1 \dots l$ ein Vektor als Eingabe und $y \in \mathbb{R}^l$ ein Vektor als Ausgabe. Weiterhin sei Q_m der Repräsentant des Knoten m mit N_m Daten, dann unterteilen wir diesen in zwei Hälften:

$$Q_m^{links}(\xi) = \{(x, y) | x_j \leq t_m\} \tag{4.1}$$

und

$$Q_m^{rechts}(\xi) = Q_m \setminus Q_m^{links}(\xi), \tag{4.2}$$

wobei $\xi = (j, t_m)$, mit j einem Datenpunkt und t_m einem Schwellwert.

Eine Verbesserung des Parameters ξ erreichen wir durch die Minimierung einer Verlustfunktion. Diese rufen wir rekursiv auf die jeweiligen Teilmengen des aktuellen Repräsentanten Q_m auf. Der Prozess wird solange ausgeführt, bis die maximale Tiefe erreicht, $N_m < \min_{samples}$ oder $N_m = 1$ ist.

Definition 4.1.2 (Binärer Decision Tree Split Qualität)
Sei Q_m der Repräsentant des Knoten m, ξ der zu minimierende Parameter und H eine Verlustfunktion, siehe Definition 4.1.3, dann minimieren wir diesen über:

$$G(Q_m, \xi) = \frac{N_m^{links}}{N_m} H\left(Q_m^{links}(\xi)\right) + \frac{N_m^{rechts}}{N_m} H\left(Q_m^{rechts}(\xi)\right), \tag{4.3}$$

wobei das Optimum $\xi_* = \arg\min_\xi G(Q_m, \xi)$ ist. Der rekursive Aufruf erfolgt über die Teilmengen $Q_m^{links}(\xi_*)$ und $Q_m^{rechts}(\xi_*)$.

Letztlich unterscheidet sich die Verlustfunktion in einem Problem der Klassifikation und Regression. Bei der Klassifikation nutzen wir die Gini Impurity-Funktion und bei einer Regression den MSE (Pedregosa u. a., 2011).

Definition 4.1.3 (Decision Tree Klassifikation und Regression)
Sei das vorliegende Problem eine Klassifikation, dann drücken wir die Verlustfunktion wie folgt aus:

$$H(Q_m) = \sum_k p_{mk}(1 - p_{mk}), \text{ mit } p_{mk} = \frac{1}{N_m} \sum_{y \in Q_m} I(y = k), \tag{4.4}$$

wenn ein Problem der Regression vorliegt, dann formulieren wir die Verlustfunktion als:

$$H(Q_m) = \frac{1}{N_m} \sum_{y \in Q_m} (y - \hat{y}_m)^2, \text{ mit } \hat{y}_m = \frac{1}{N_m} \sum_{y \in Q_m} y. \tag{4.5}$$

4.2. DeepRED

DeepRED ist ein Algorithmus[2] (Zilke u. a., 2016) zur Regelextraktion aus DNNs mithilfe von DTs. Dieser basiert auf dem CRED Algorithmus (Sato u. Tsukimoto, 2001) und nutzt den in (Quinlan, 1993) vorgestellten C4.5 Algorithmus[3] zur Erzeugung eines DTs. CRED ermöglicht die Regelextraktion aus einem Neural Network (**NN**) mit einer versteckten Schicht, wohingegen DeepRED die Extraktion aus einem DNN mit k versteckten Schichten erlaubt, $k \in \mathbb{N}$. Wir betrachten ein Problem der Klassifikation mit der Eingabe $x \in \mathbb{R}^n$, wobei jedes x_j einer Entscheidungsklasse, sprich der Ausgabe, $\lambda_v \in \{\lambda_1, \ldots, \lambda_u\}$ zugeordnet wird, mit $v = 1, \ldots, n$ und $n, u \in \mathbb{N}$. Eine versteckte Schicht $h_i \in \{h_1, \ldots, h_k\}$ enthält h_{i,H_i} Neuronen. Der Übersicht halber setzten wir h_0 als die Eingabe und h_{k+1} als die Ausgabe (Zilke u. a., 2016).

Der DeepRED Algorithmus bildet DTs und darauf basierende Regeln, die durch Berücksichtigung der Aktivierungen aller Schichten erzeugt werden. Dabei wird für jede Entscheidungsklasse λ_v wie folgt vorgegangen: Beginnend mit der Erstellung eines DTs der letzten Schicht h_{k+1}, werden Split-Punkte bezüglich der Werte der vorherigen Schicht h_k erzeugt. Diese Split-Punkte bilden die Regeln $R^v_{h_k \to h_{k+1}}$ für die h_{k+1}-te Schicht. Die benötigten Daten zur Erzeugung eines DTs werden vorwärts durch das Netz propagiert. In einem nächsten Schritt gehen wir analog zu der vorherigen Schicht vor, sprich wir bestimmen Regeln $R^v_{h_{k-1} \to h_k}$, indem wir einen DT mithilfe des C4.5 Algorithmus für die h_k-te Schicht erstellen. Diese Prozedur führen wir solange aus, bis wir die eigentliche Eingabe des Netzes verwenden und somit die Menge an Regeln $\{R^v_{h_0 \to h_1}, R^v_{h_1 \to h_2}, \ldots, R^v_{h_k \to h_{k+1}}\}$ für die einzelnen Schichten erhalten. Die Formulierung von Regeln, welche das gesamte DNN für eine Entscheidungsklasse beschreiben $R^v_{h_0 \to h_{k+1}}$, erfolgt über die Zusammenführung der gesammelten Regeln. Dazu starten wir mit den Regeln der letzten Schicht und verfahren wie folgt:

$$R^v_{h_{j-1} \to h_{k+1}} \leftarrow merge(R^v_{h_{j-1} \to h_j}, R^v_{h_j \to h_{k+1}}), \text{ mit } j = k, k-1, \ldots, 1, \tag{4.6}$$

wobei $merge : R \times R \to R$ eine Funktion ist, die zwei Regeln zusammenführt. Eine genaue Beschreibung der merge-Funktion ist durch das referenzierte Paper nicht gegeben. Gewonnene Regeln, die nicht erfüllbar oder redundant sind, entfernen wir. Der beschriebene Prozess wird für alle Entscheidungsklassen wiederholt, sodass wir letztlich die Ausgabe des DNNs durch einen DT mit der Menge an Regeln $\{R^1_{h_0 \to h_{k+1}}, R^2_{h_0 \to h_{k+1}}, \ldots, R^u_{h_0 \to h_{k+1}}\}$ beschreiben können (Zilke u. a., 2016).

Weitere Informationen über das vorgestellte Modell sind in dem referenzierten Paper oder in der Masterarbeit (Zilke, 2015) des Entwicklers zu finden.

[2] Der DeepRED Algorithmus befindet sich im Anhang als Pseudocode.

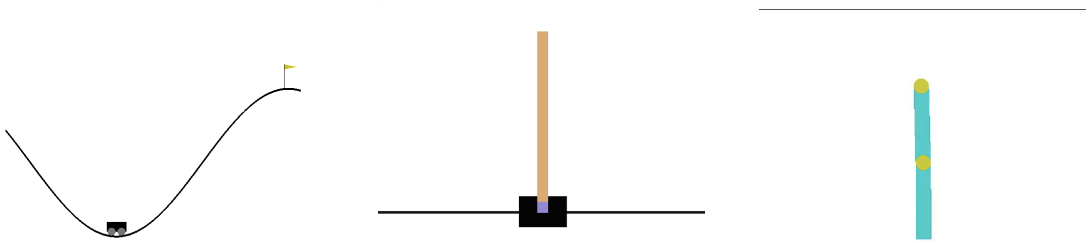
[3] Der C4.5 Algorithmus ist in der Lage Probleme der Klassifikation, jedoch keine der Regression, zu lösen. Der Code ist in (Pedregosa u. a., 2011) zu finden.

5. Aufbau Experimente

Nachdem wir die Grundlagen des RLs und die eingesetzten Modelle zur Extraktion von Regeln aus DRL Algorithmen vorgestellt haben, wollen wir den Aufbau sowie die Voraussetzungen für die Anwendung dieser besprechen. Dazu stellen wir zunächst die untersuchten Umgebungen, beziehungsweise die RL Probleme, vor. Aufbauend auf dem vorangegangenen Praxisprojekt (Oedingen, 2021, S.62 ff.) entnehmen wir die darin identifizierten Hyperparameter für die jeweiligen DRL Algorithmen in den einzelnen Umgebungen. Weiterhin gehen wir auf die verschiedenen Bibliotheken für die genutzte Implementierung und die Verwendung der identifizierten Regeln ein.

5.1. Reinforcement Learning Probleme

In dieser Arbeit haben wir vier verschiedene RL Probleme untersucht, welche auch als die klassischen Kontrollprobleme bekannt sind. Darunter befinden sich die folgenden Probleme: CartPole-v1 (**CP**), MountainCar-v0 (**MC**), MountainCarContinuous-v0 (**MCC**) und Acrobot-v1 (**AB**). Diese sind in Abbildung 5.1 dargestellt. Bei den folgenden Beschreibungen verweisen wir auf (Brockman u. a., 2016) und für weitere Spezifikationen auf (Oedingen, 2021, S.56 ff.).



(a) MountainCar/Continuous-v0 (b) CartPole-v1 (c) Acrobot-v1

Abbildung 5.1.: Reinforcement Learning Probleme (Brockman u. a., 2016)

MC/C beschreibt ein Problem, indem ein Auto in einem Tal steht und die Fahne auf dem rechten Berg erreicht werden soll, siehe Abbildung (a). Aufgrund der unzureichenden Leistung des Motors ist der Wagen nicht in der Lage, den Berg direkt hinaufzufahren. Daher muss der Agent genügend Schwung aufbauen, um den Berg zu erklimmen und die Fahne zu erreichen. Das MC (Moore, 1990) und MCC (Singh u. a., 1995, S.143) Problem unterscheiden sich durch ihren diskreten und kontinuierlichen Aktionsraum. Dabei werden fest definierte Aktionen durch einen kontinuierlichen Wert, im Weiteren als „Leistungs-Koeffizient“ bezeichnet, ersetzt.

Das in Abbildung (b) dargestellte CP Problem wurde erstmals in (Barto u. a., 1983) formuliert und beschreibt das Balancieren einer Stange, welche an einem Wagen befestigt ist. Ziel des Agenten ist es, die Stange vor dem Herunterfallen zu bewahren, indem er den Wagen nach links oder nach rechts bewegt. Dabei darf der Wagen nicht die Grenzen der eindimensionalen Strecke überschreiten.

Abschließend beschreiben wir das Problem des Aufschwingens eines Doppelpendels mit dem AB Problem, siehe Abbildung (c). Das Ziel des Agenten ist es, das Doppelpendel hochzuschwingen, indem ein Drehmoment auf das untere Gelenk ausgeübt wird, sodass das untere Pendel die schwarze Linie berührt. Erstmals wurde das Problem in (Sutton, 1996) formuliert.

Im Folgenden geben wir einen Überblick über die Observationen[1] in Tabelle 5.1 und die Aktionen[2] in Tabelle 5.2, welche ein Agent wahrnimmt, beziehungsweise in den vorgestellten Umgebungen ausführen kann.

Umgebung	**Nummer**	**Observation**	**Minimum**	**Maximum**
MC/C	0	Auto Position	-1.2	0.6
	1	Auto Geschwindigkeit	-0.07	0.07
CP	0	Wagen Position	-4.8	4.8
	1	Wagen Geschwindigkeit	$-\infty$	∞
	2	Stange Winkel	$-24°$	$24°$
	3	Stange Winkel Geschwindigkeit	$-\infty$	∞
AB	0	$\cos(\theta_1)$	-1.0	1.0
	1	$\sin(\theta_1)$	-1.0	1.0
	2	$\cos(\theta_2)$	-1.0	1.0
	3	$\sin(\theta_2)$	-1.0	1.0
	4	v_1	$-\infty$	∞
	5	v_2	$-\infty$	∞

Tabelle 5.1.: Reinforcement Learning Probleme Observationen (Brockman u. a., 2016; Verma u. a., 2018)

Eine Umgebung gilt in den meisten Fällen als gelöst, wenn die Belohnung in 100 aufeinander folgenden Episoden einen bestimmten Schwellwert über- oder unterschreitet. Die Schwellwerte sind für die jeweilige Umgebung in Tabelle 5.3 zusammen mit den erreichten Belohnung der von uns eingesetzten Modelle hinterlegt.

[1] Im Acrobot-v1 Problem wird der Winkel zwischen dem äußeren Glied und dem äußeren Gelenk als θ_1 und der Winkel zwischen dem unteren Glied und dem unteren Gelenk als θ_2 bezeichnet. Weiterhin stellt $v1$ die gemessene Geschwindigkeit am oberen Gelenk und $v2$ die gemessene Geschwindigkeit am unteren Gelenk dar.

[2] Die Aktion im MCC Problem stellt einen kontinuierlichen Wert dar, welcher ein Minimum und Maximum besitzt.

Umgebung	Nummer	Aktion	Minimum	Maximum
MC	0	Beschleunige Links	-	-
	1	Beschleunige Nicht	-	-
	2	Beschleunige Rechts	-	-
MCC	0	Leistungs-Koeffizient	-1.0	1.0
CP	0	Schiebe Links	-	-
	1	Schiebe Rechts	-	-
AB	0	Drehe Links	-	-
	1	Drehe Nicht	-	-
	2	Drehe Rechts	-	-

Tabelle 5.2.: Reinforcement Learning Probleme Aktionen (Brockman u. a., 2016)

5.2. Implementierung

Bei der Implementierung der DRL Modelle haben wir uns für Stable Baselines3 (**SB3**) (Raffin u. a., 2019) entschieden. SB3 bietet eine Auswahl von vertrauenswürdigen und robusten DRL Algorithmen, welche unter Verwendung von PyTorch (Paszke u. a., 2019) entwickelt wurden. Dadurch vermeiden wir Instabilitäts- und Leistungsmängel, die durch ineffiziente Implementierungen auftreten können. Die Umgebungen aus Abbildung 5.1 werden durch das OpenAI-Gym (Brockman u. a., 2016) bereitgestellt und sind kompatibel mit SB3. Dabei können die vorgestellten Umgebungen als vollständig beobachtbaren MEP formuliert werden.

In der vorangegangenen Arbeit (Oedingen, 2021, S.62 ff.) haben wir uns intensiv mit der Modifikation der Hyperparameter beschäftigt, da diese eine signifikante Rolle in der Leistung eines Agenten spielen. Diese entscheiden, ob es sich bei dem trainierten Modell um ein außergewöhnlich gutes oder ein eher moderates Modell handelt. Zusätzlich zu unser Suche nach optimierten Hyperparametern, haben wir die des RL Baselines3 Zoos (Raffin, 2020) genutzt, welche unter anderem von Optuna (Akiba u. a., 2019) erzeugt wurden. Optuna ist ein Programm zur Optimierung von Hyperparametern in einem DRL Algorithmus. Daraus resultierend haben wir DRL Modelle[3] mit hohen Leistungen in ihren Umgebungen erhalten. Den verwendeten Python-Code für die Erstellung der eingesetzten Modellen haben wir in diesem *Repository* hinterlegt.

Basierend auf den performantesten Modellen aus Tabelle 5.3 haben wir die Experimente zur Regelextraktion aufgebaut. Unter der Verwendung der bereitgestellten Bibliothek eines DTs von scikit-learn (Buitinck u. a., 2013) haben wir versucht, Regeln zu extrahieren. Für die weiteren Versuche bezüglich des DeepRED Algorithmus haben wir zur Erstellung eines DNNs[4] das TensorFlow Framework (Abadi u. a., 2016) genutzt. Der DeepRED Algorithmus selber ist von diesem *Github Benutzer* (Fanta, 2019) im

[3]Die verwendeten Hyperparameter für die eingesetzten DRL Modelle sind im Anhang zu finden.
[4]Die genutzten Hyperparameter zur Erstellung der DNNs befinden sich im Anhang.

Rahmen einer Masterarbeit implementiert worden. Weitere genutzte Bibliotheken für die Erstellung von Abbildungen und zur Analyse von Daten sind im Anhang zu finden.

Grundsätzlich können wir Regeln aus einem DT als `IF-ELSE` Anweisungen extrahieren. Die vorgestellten Algorithmen aus Kapitel 4 liefern genau einen DT für ihr vorliegendes Problem, sodass alle notwendigen Konditionen zur Einteilung in einen bestimmten Bereich vorliegen. Betrachten wir den DT aus Abbildung 4.1, dann formulieren wir diesen mithilfe von `IF-ELSE` Anweisungen als: `IF` $X_1 \leq t_1$ `THEN (IF` $X_2 \leq t_2$ `THEN` R_1 `ELSE` R_2`) ELSE (IF` $X_1 \leq t_3$ `THEN` R_3 `ELSE (IF` $X_2 \leq t_4$ `THEN` R_4 `ELSE` R_5`))`. Durch diese Regeln lässt sich der gesamte DT beschreiben. Für diese Form der Darstellung haben wir einen rekursiven Algorithmus genutzt, welcher den DT in Preorder (Necaise, 2010, S.377) traversiert. Der genutzte Code für die im nächsten Kapitel vorzustellenden Experimente ist in diesem *Repository* hinterlegt.

5.3. Leistung Deep Reinforcement Learning Modelle

In unser vorherigen Arbeit haben wir leistungsstarke DRL Modelle erstellt. In der nachfolgenden Tabelle[5] befinden sich die kumulierten Belohnungen der DRL Modelle in der jeweiligen Umgebung. Für einen DRL Algorithmus haben wir jeweils vier Modelle erstellt, trainiert und für 100 Episoden mit der Umgebung interagieren lassen. Durch vier Durchläufe eines Modells unter gleichen Voraussetzung können wir die erreichten Belohnungen in Tabelle 5.3 verifizieren und haben somit jede Umgebung gelöst.

Umgebung	**Modell**			**Gelöst**
	DQN	PPO	A2C	
MC	-99.65 ± 7.21	-98.41 ± 7.10	-107.17 ± 20.04	≥ -110
	-98.59 ± 7.66	-99.08 ± 6.46	-100.56 ± 21.38	
	-98.71 ± 7.59	-98.62 ± 7.87	-109.94 ± 20.71	
	-96.98 ± 4.58	-98.65 ± 7.31	-113.34 ± 25.03	
MCC	-	92.22 ± 2.33	93.24 ± 0.79	≥ 90
	-	93.02 ± 1.40	93.23 ± 0.58	
	-	92.33 ± 1.96	93.13 ± 0.94	
	-	92.72 ± 1.89	93.20 ± 0.46	
CP	500 ± 0	500 ± 0	500 ± 0	≥ 475
	500 ± 0	500 ± 0	500 ± 0	
	500 ± 0	500 ± 0	500 ± 0	
	500 ± 0	500 ± 0	500 ± 0	
AB	-72.13 ± 12.79	-80.84 ± 10.06	-83.70 ± 23.10	-
	-72.92 ± 13.08	-82.41 ± 11.92	-77.79 ± 9.53	
	-72.72 ± 12.45	-79.77 ± 6.91	-86.62 ± 30.09	
	-71.18 ± 11.18	-83.33 ± 22.11	-79.81 ± 19.64	

Tabelle 5.3.: Deep Reinforcement Learning Modelle Leistung

[5] Die Spalte „Gelöst“ gibt an, ab welchem Schwellwert nach 100 Episoden ein Problem als gelöst bezeichnet wird. Für die AB Umgebung ist kein Schwellwert definiert.

6. Experimente Regelextraktion aus Deep Reinforcement Learning Modellen

Dieses Kapitel bildet den Hauptteil unserer Arbeit und beinhaltet die durchgeführten Experimente zur Regelextraktion aus DRL Modellen. Dazu untersuchen wir die vorgestellten Umgebungen aus Sektion 5.1 mit den aus Sektion 3.4 zuletzt etablierten DRL Algorithmen. Weiterhin extrahieren wir die Regeln mit den eingeführten Methoden aus Kapitel 4.

Für jedes RL Problem versuchen wir, Regeln zu finden, welche die Entscheidungen des Agenten erläutern. Diesbezüglich starten wir jede Sektion mit einer Übersicht über die erreichten Belohnungen der DTs in der jeweiligen Umgebung. Basierend auf den erbrachten Leistungen und genutzten Tiefen selektieren wir diejenigen Modelle, welche hauptsächlich zur Regelextraktion untersucht werden. Nichtsdestotrotz vergleichen wir signifikante Unterschiede zwischen den primär zu untersuchenden und den restlichen Modellen für einzelne Umgebungen. In einem zweiten Ansatz stellen wir die Ergebnisse der angelernten DNNs dar, welche zur Regelextraktion des DeepRED Modells benötigt werden. Nachfolgend widerlegen wir die Interpretierbarkeit der identifizierten Regeln des DeepRED Modells auf RL Problemen.

6.1. Validierung: XOR Problem

Bevor wir mit der Extraktion von Regeln aus einem komplexen DRL Modell beginnen, wollen wir die Korrektheit der vorgestellten Methoden aus Kapitel 4 am XOR Problem (Brutzkus u. Globerson, 2019) zeigen.

A	B	$A \oplus B$
0	0	0
0	1	1
1	0	1
1	1	0

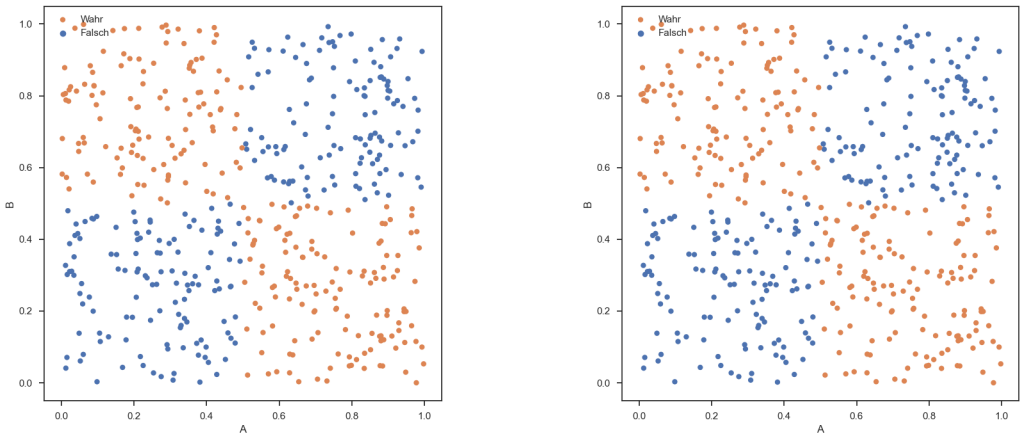
Tabelle 6.1.: XOR Problem Wahrheitstafel (Arens u. a., 2013, S.34)

Das XOR Problem beschreibt ein zweidimensionales Problem und ist nicht linear separierbar. Daher suchen wir eine nicht-lineare Funktion, um das Problem zu lösen. Dieses lässt sich für zwei Aussagen A und B mit booleschen Ausdrücken als: $A \oplus B \Leftrightarrow ((A \vee B) \wedge \neg(A \wedge B))$ formulieren (Arens u. a., 2013, S.34). Das Ergebnis dieses Ausdrucks ist also nur wahr, wenn entweder A oder B wahr ist und nicht beide. Für das XOR

Problem betrachten wir die Wahrheitstafel in Tabelle 6.1. Dementsprechend suchen wir Regeln, welche zwei eingehende Aussagen korrekt zuordnen. Unsere Ergebnisse befinden sich in Tabelle 6.2 und sind visualisiert in Abbildung 6.1.

Methode	**Training**	**Genauigkeit Training**	**Test**	**Genauigkeit Test**
DNN	2×10^3	100%	500	100%
DT	2×10^3	-	500	100%
DeepRED	1×10^5	-	500	100%

Tabelle 6.2.: XOR Problem Ergebnisse - Training und Test sind als die Anzahl der Datenpaare aus (A, B) zu verstehen, welche für den jeweiligen Datensatz genutzt wurden.



(a) Wahre Lösung nach selbst gemachten Regeln

(b) Modellbasierte Lösung nach DNN/DT/ DeepRED

Abbildung 6.1.: XOR Problem Ergebnisse - Der Ergebnisraum besteht aus vier Quadranten, welche die möglichen Eingaben widerspiegeln: 1. Quadrant $(1, 1)$, 2. Quadrant $(0, 1)$, 3. Quadrant $(0, 0)$ und 4. Quadrant $(1, 0)$. Die selbst gemachten Regeln, welche ebenfalls vom DeepRED Modell gefunden wurden, lauten: `IF` $A \geq 0.5$ `THEN (IF` $B \geq 0.5$ `THEN Falsch ELSE Wahr) ELSE (IF` $B < 0.5$ `THEN Falsch ELSE Wahr)`. Auf die Regeln des DTs verzichten wir, da diese zu komplex sind.

Mit Abbildung 6.1 bestätigen wir die Identität der wahren und modellbasierten Lösung aus Tabelle 6.2. Demzufolge schließen wir auf eine korrekte Verwendung der bereitgestellten Bibliotheken des DNNs (Abadi u. a., 2016), des DTs (Buitinck u. a., 2013) und des DeepRED Modells (Fanta, 2019).

6.2. Decision Tree

Unser erster Ansatz beschäftigt sich mit der Regelextraktion aus DRL Modellen mithilfe von DTs. In den folgenden Sektionen beschreiben wir die von uns identifizierten Regeln und versuchen diese zu interpretieren. Zur Erstellung und Evaluierung eines DTs haben wir die performantesten Modelle aus Tabelle 5.3 selektiert. Diese haben wir mit der jeweiligen Umgebung interagieren lassen und die Daten[1] der Interaktion gespeichert. Aufbauend auf diesen Daten haben wir den DT erstellt und mit den gewonnenen Regeln ausgewertet. Alle vorzustellenden DTs in dieser Sektion wurden mit dem CART Algorithmus erstellt. Die dargestellten Regeln sind im Anhang zu finden.

6.2.1. MountainCar-v0

Die MC Umgebung bietet aufgrund der niedrigen Dimension der Eingabe einen guten Einstieg in die Extraktion von Regeln aus RL Problemen, siehe Tabelle 5.1. Wir betrachten lediglich die Geschwindigkeit (**GA**) und die Position (**PA**) des Autos, um das Problem zu lösen. Durch den diskreten Aktionsraum betrachten wir ein Problem der Klassifikation, sodass Observationen auf fest definierte Aktionen abgebildet werden. In der nachfolgenden Tabelle wollen wir die Ergebnisse des DTs für das MC Problem vorstellen.

Tiefe	**Modell**		
	DQN	PPO	A2C
-	-96.98 ± 4.58	-96.82 ± 8.23	-100.56 ± 21.38
10	-97.76 ± 8.20	-97.26 ± 7.15	-100.36 ± 9.80
5	-98.10 ± 7.52	-97.73 ± 8.34	-102.70 ± 6.01
4	-99.78 ± 5.97	-98.58 ± 5.80	-103.33 ± 10.92
3	-101.24 ± 2.84	-101.12 ± 8.38	-104.41 ± 10.21
2	-115.84 ± 1.43	-117.22 ± 3.47	-131.99 ± 25.61
1	-119.46 ± 3.81	-123.07 ± 12.24	-134.26 ± 27.61

Tabelle 6.3.: MountainCar-v0 Decision Tree Leistung - Die evaluierten DTs werden auf dem in der ersten Zeile angegebenen Modell aufgebaut. Die Anzahl der Episoden, welche wir zur Evaluierung nutzen, beträgt 100.

Mit Tabelle 6.3 stellen wir fest, dass sich die DTs des DQN und PPO Modells aufgrund der Leistungen zur Untersuchung anbieten. Die DTs des A2C Modells hingegen weisen eine hohe Standardabweichung auf und liefern keine robusten und schlechtere Ergebnisse, sodass wir diese für das MC Problem vernachlässigen. Weiterhin erkennen wir, dass die kumulierten Belohnungen für DTs mit größere Tiefe auch größer ausfallen. Dies ergibt durchaus Sinn, da sich so Entscheidungen verfeinern lassen. Ein Nachteil ist durch das mangelnde Maß an Interpretierbarkeit für DTs mit großer Tiefe gegeben. Daher untersuchen wir DTs mit einer Tiefe von drei für das MC Problem. Trotz geringer Tiefe liefern diese Ergebnisse, welche das vorliegende Problem lösen.

[1] Die Daten werden durch die Observationen und gewählte Aktion in einem Zustand dargestellt.

Die Visualisierung der DTs erfolgte über Graphviz (Ellson u. a., 2001). Mit diesem Tool werden die Entscheidungsknoten basierend auf deren vorhergesagtem Wert und Verlust gefärbt. Daher ist der DT auch für den nicht fachkundigen Leser durch seine Färbung erklärbar. Der DT für das DQN Modell ist in Abbildung 6.2 und der DT des PPO Modells ist in Abbildung 6.3 dargestellt.

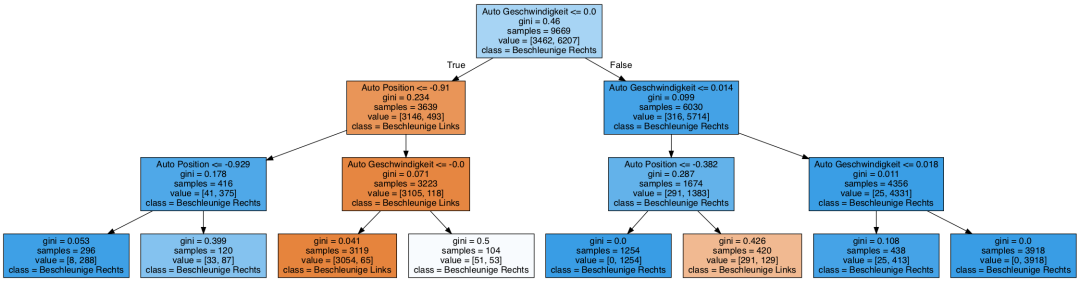


Abbildung 6.2.: MountainCar-v0 DQN Decision Tree Tiefe 3 - Eine Auffälligkeit des DTs ist, dass dieser lediglich zwei von drei möglichen Aktionen benutzt. Die Aktion „Beschleunige Nicht“ wird vom DT nicht berücksichtigt. Weiterhin enthalten nur zwei von acht Blättern die Aktion „Beschleunige Links“.

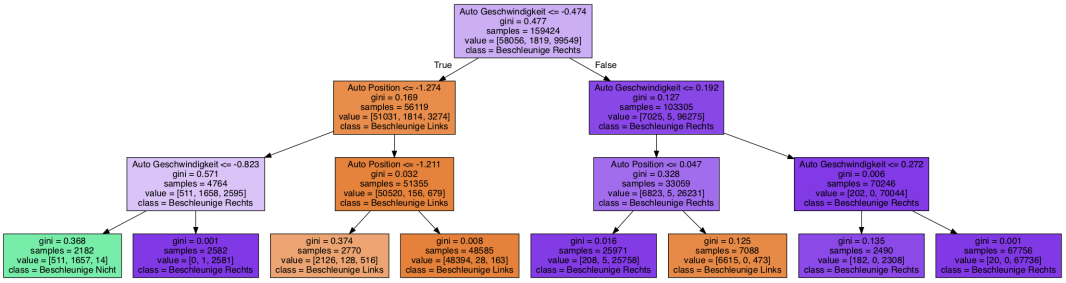
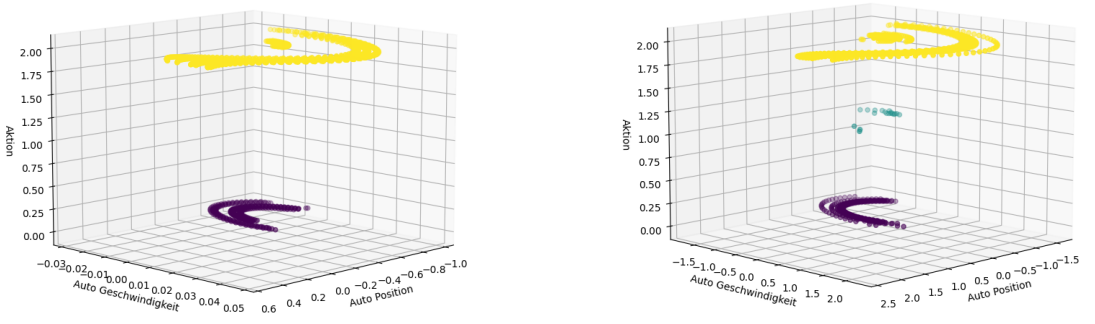


Abbildung 6.3.: MountainCar-v0 PPO Decision Tree Tiefe 3 - Alle Aktionen und Observationen werden berücksichtigt. Auffallend sind die abweichenden Werte der Observationen im Vergleich zu Tabelle 5.1. Dies resultiert aus der normalisierten Umgebung, mit welcher das PPO Modell interagiert, siehe (Oedingen, 2021, S.82).

Im Vergleich der DTs weisen die jeweils zur rechten Seite des Wurzelknotens liegenden Entscheidungsbäume identische Observationen auf. Weiterhin werden diese bei unterschiedlichen Werten der Entscheidungsknoten als die selbe Aktion klassifiziert. Hingegen herrscht im linken Entscheidungsbaum keine Übereinstimmung ab einer Tiefe von zwei. Trotz ihrer Unterschiede verfolgen diese eine ähnliche Strategie. Wie schon in Sektion 5.1 beschrieben, versucht der Agent in der MC Umgebung genügend Schwung aufzubauen, um den rechten Berg zu erklimmen und die Fahne zu erreichen. Dabei wird in einer nicht normalisierten Umgebung eine beliebige Startposition des Autos in einem Intervall von $[-0.6, -0.4]$ mit einer Geschwindigkeit von null gewählt.

Die Regeln des DTs für das DQN Modell unterscheiden grundsätzlich zwischen zwei Strategien[2]. Im ersten Fall, wobei die Startposition so weit rechts ist, dass das Auto zurück ins Tal rollt, sprich GA ≤ 0, beschleunigt das Auto weiter nach links. Diese Aktion wird solange ausgeführt, bis GA > 0, also das Auto nicht mehr ausreichend Antrieb besitzt, um den linken Berg hinaufzufahren und die Geschwindigkeit positiv wird. Das Auto rollt daher den linken Berg hinab. Resultierend wurde genug Schwung aufgebaut, um den Rest der Episode nach rechts zu beschleunigen und die Fahne zu erreichen. In einer zweiten Strategie befindet sich das Auto auf der linken Seite des Tals, sodass das Auto den linken Hügel hinab rollt, also GA > 0 gilt. Das Auto beschleunigt nach rechts bis GA ≤ 0.014 erfüllt ist und beschleunigt anschließend nach links, bis PA ≤ -0.91 ist. Mit dieser Technik wurde ebenfalls genug Schwung aufgebaut, um den Rest der Episode nach rechts zu beschleunigen und den Berg zu erklimmen. Offensichtlich berücksichtigt der in Abbildung 6.2 dargestellte DT noch weitere Fälle. Diese Formulierung sollte lediglich einen Eindruck der Verhaltensweise vermitteln.



(a) DQN MountainCar-v0 Decision Tree Interaktionsdaten

(b) PPO MountainCar-v0 Decision Tree Interaktionsdaten

Abbildung 6.4.: MountainCar-v0 DQN/PPO Decision Tree Interaktionsdaten - Deutlich zu beobachten ist die im DT des PPO Modells genutzte Aktion „Beschleunige Nicht“, welche durch Aktion 1 repräsentiert wird. Im DT des DQN Modells wird diese nicht berücksichtigt.

Der DT des PPO Modells lässt sich fast analog zur zweiten Strategie des DTs des DQN Modells beschreiben. Das Auto startet in einer beliebigen Position des Tals und fährt zunächst solange den rechten Berg hinauf, bis PA > 0.047 gilt. Daraufhin beschleunigt es nach links, mit dem Ziel genügend Schwung über den linken Hügel aufzubauen. Letztlich unterscheidet sich der weitere Verlauf von der zweiten Strategie des DTs des DQN Modells durch die Verwendung der „Beschleunige Nicht“ Aktion, siehe Abbildung 6.4. Bei Einhaltung der Grenzen von $-0.823 \leq$ GA ≤ -0.474 und PA ≤ -1.247 wird diese Aktion gewählt. Dadurch vermeidet das Auto, einen nicht notwendigen Teil des Berges zu erklimmen, da bereits genügend Schwung aufgebaut wurde, und spart somit Zeit. Aus unseren Ergebnissen geht hervor, dass die Verwendung der

[2]In diesem Szenario kann die Strategie als die resultierende Verhaltensweise aus den extrahierten Regeln verstanden werden. In einem Problem, dass durch einen MEP beschrieben wird, existiert eine Strategie, die diese Verhaltensweise abdeckt.

weiteren Aktion die Leistung des Agenten in einem DT des PPO Modells verbessert. Als durchschnittliche Belohnung in 100 Episoden für den in Abbildung 6.3 dargestellten DT haben wir -101.12 ± 8.38 erhalten. Weiterhin haben wir in den Episoden, in denen die „Beschleunige Nicht“ Aktion nicht berücksichtigt wurde, -103.37 ± 9.49 und in Episoden mit der weiteren Aktion -92.03 ± 6.98 durchschnittliche kumulierte Belohnungen erreicht. Eine optimale Lösung beschreiben wir jedoch ohne diese Aktion. Durch den unmittelbaren Einsatz der „Beschleunige Rechts“ Aktion, nach dem der linke Berg zum Teil erklommen wurde, können die bestmöglichen Ergebnisse erzielt werden, ähnlich wie beim DT des DQN Modells mit anderen Splitpunkten auf den Observationen.

In einem weiteren Schritt wollen wir die Unterschiede in den Entscheidungen zwischen den DRL Modellen und den darauf aufgebauten DTs feststellen. Dazu betrachten wir Abbildung 6.4 im zweidimensionalen Raum.

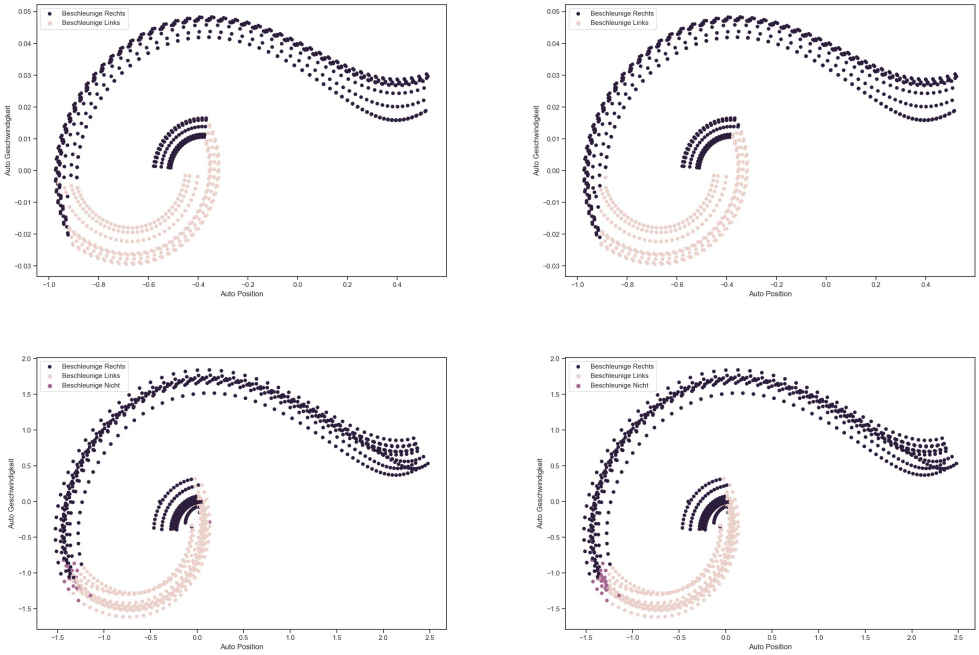


Abbildung 6.5.: MountainCar-v0 DQN/PPO Decision Tree Vergleich Interaktionsdaten - Auf der linken Seite befinden sich die DRL Modelle (oben DQN / unten PPO) und daneben die darauf aufgebauten DTs. Erreichte Genauigkeit: DQN DT $98,96\%$ und PPO DT $98,37\%$ in zehn aufeinander folgenden Episoden.

Mithilfe von Abbildung 6.5 stützen wir die in den vorherigen Abschnitten aufgestellten Thesen der Verhaltensweise der DTs. Des Weiteren können wir anhand dieser Abbildung sehen, dass keine signifikanten Unterschiede zwischen den regelbasierten und DRL Modellen existieren, welches sich auch in der Genauigkeit widerspiegelt.

Zuletzt wollen wir die von uns identifizierten Regeln mit bereits vorhandenen vergleichen, um die Qualität unserer Regeln in Tabelle 6.4 zu verifizieren. Für diesen Vergleich haben wir einen bereits vorhandenen Ansatz zur Regelextraktion durch DTs genommen, beschrieben in (Engelhardt u. a., 2021). Als Erweiterung zu üblichen DTs wurde in diesem Paper ein weiterer Ansatz besprochen, die Oblique Decision Trees (**ODT**) (Stepisnik u. Kocev, 2020). Diese ermöglichen eine Aufteilung des zweidimensionalen Raums mit zusätzlichen diagonalen Linien, anstelle der sonst achsenparallelen Trennlinien, vergleiche Abbildung 4.1. Ein weiteres Ergebnis haben wir der vorgestellten Methode zur Extraktion aus einer höheren Programmiersprache (Verma u. a., 2018) entnommen. Dabei wurden für die gewonnenen Regeln aus einem Dueling Deep Q-Network (**DDQN**) (Wang u. a., 2015) extrahiert. Zudem wollen wir einige selbst erstellte (engl. Handcrafted) (**HC**) Regeln vorstellen, die wir aus Engelhardt u. a. (2021) und Xiao (2019) übernommen haben.

Methode	**Leistung Methode**	**Modell**	**Leistung Modell**
DQN DT_3	-101.24 ± 2.84	DQN	-96.98 ± 4.58
PPO DT_3	-101.12 ± 8.38	PPO	-96.82 ± 8.23
Engelhardt DT_4	-103.5 ± 8.85	DQN	-101.9 ± 10.75
Engelhardt ODT_4	-105.0 ± 13.28	DQN	-101.9 ± 10.75
Engelhardt HC	-107.71 ± 13.95	-	-
Xiao HC	-104.58 ± 20.28	-	-
Verma	$-108.06, -143.86$	DDQN	-84.73

Tabelle 6.4.: MountainCar-v0 Regeln Vergleich - Die angehängten Indizes bezeichnen die verwendete Tiefe des jeweiligen DTs. Die Anzahl der Episoden, welche wir zur Evaluierung nutzen, beträgt 100. In Verma u. a. (2018) wurden keine Standardabweichungen hinterlegt. Die Spalte „Modell“ gibt das für den DT aufbauende DRL Modell an.

Offensichtlich gilt, dass ein Modell des Supervised Learning stark von der Qualität der zur Verfügung stehenden Daten abhängt (Li u. Liang, 2019). In unserem Fall, stehen die Daten in direktem Zusammenhang mit dem zu untersuchenden DRL Modell. Daher folgen aus einem qualitativen DRL Modell auch meist Regeln, welche hohe kumulierte Belohnungen erzielen. Eine vermeintlich sinnvolle Überlegung wäre es daher, Episoden zu filtern, welche eine kumulierte Belohnung unterschreiten, beziehungsweise überschreiten. Dies resultiert in einem DRL Modell, das eine bessere Strategie erlernt und woraus leistungsstärkere Regeln extrahiert werden können. Durch diese Änderung würden die Regeln einen sehr speziellen Fall behandeln, der theoretisch vorteilhaft wäre, praktisch jedoch nicht einsetzbar ist. Regeln sollen verschiedene Ereignisse in der Umgebung, welche auch von den Üblichen abweichen, abdecken, sodass diese für viele Eventualitäten genutzt werden können und nicht nur den „perfekten“ Fall behandeln.

6.2.2. MountainCarContinuous-v0

Im MCC gilt es das gleiche Problem zu lösen, wie auch schon in der vorherigen MC Umgebung. Die beiden Umgebungen unterscheiden sich in ihrem diskreten und kontinuierlichen Aktionsraum, sodass wir hier ein Problem der Regression betrachten. Weiterhin differieren die Umgebungen in der Vergabe von Belohnungen, siehe (Oedingen, 2021, S.57). Vorab können wir die Verwendung des DQN Modells für kontinuierliche Umgebungen vernachlässigen, da dieses eine Verlustfunktion $\mathcal{L}_i : \psi_i \to \mathbb{R}$ durch den MSE wie folgt minimiert:

$$\mathcal{L}_i(\psi_i) = \mathbb{E}_{s,a\sim\rho(\cdot),s'\sim\epsilon}\left[\left(r + \gamma \max_{a'\in\mathcal{A}} \hat{q}(s', a', \psi_{i-1}) - \hat{q}(s, a, \psi_i)\right)^2\right], \tag{6.1}$$

wobei das Maximum über eine Auswahl von diskreten Aktionen gewählt wird. Folglich sind wir mit dem vorliegenden Modell nicht in der Lage, ein RL Problem mit kontinuierlichem Aktionsraum zu lösen (Oedingen, 2021, S.61). Wir stellen in Tabelle 6.5 die Leistungen der erzeugten DTs und der genutzten DRL Modelle dar.

Tiefe	**Modell**	
	PPO	A2C
-	93.02 ± 1.40	93.23 ± 0.58
10	93.56 ± 0.59	93.89 ± 0.31
5	93.45 ± 0.79	93.73 ± 0.42
4	92.59 ± 1.72	93.69 ± 0.56
3	92.25 ± 1.85	93.61 ± 0.57
2	82.07 ± 0.71	93.48 ± 1.66
1	81.47 ± 0.85	83.43 ± 0.74

Tabelle 6.5.: MountainCarContinuous-v0 Decision Tree Leistung - Die evaluierten DTs wurden auf dem in der ersten Zeile angegebenen Modell aufgebaut.Die Anzahl der Episoden, welche wir zur Evaluierung nutzen, beträgt 100.

Für das MCC Problem konnten wir selbst für sehr geringe Tiefen DTs finden, welche gute und stabile Ergebnisse liefern, siehe Tabelle 6.5. Dabei erzielen einige DTs sogar bessere Ergebnisse als das DRL Modell selbst. Dies wird im weiteren Verlauf dieser Sektion besprochen. Aufgrund der geringen Tiefe von zwei und der daraus resultierenden Interpretierbarkeit der Leistung des DTs des A2C Modells, wählen wir dieses zur primären Untersuchung aus. Weiterhin versuchen wir, dieses mit dem DT des PPO Modells bei einer Tiefe von drei zu vergleichen. Basierend auf den Beobachtungen und Ergebnissen des MC Problems erwarten wir ähnliche Regeln, mit dem Unterschied in der Ausgabe von kontinuierlichen Werten, welche die Strategie beschreiben. Beide Modelle wurden in normalisierten Umgebungen erstellt, sodass die Observationen abweichend von den aufgeführten Werten in Tabelle 5.1 sind.

Bei der Extraktion von Regeln existieren Observationen, welche sich als relevanter erweisen als andere (Dunn u. a., 2021). In einigen Problemen ist es durchaus möglich,

dass eine einzige Beobachtung ausreicht, um dieses zu lösen. Solch eine Beobachtung bezeichnen wir als univariat (Denis, 2016, S.88 ff.). Eine univariate Beobachtung ist eine eindimensionale unabhängige Variable, mit welcher wir versuchen, eine im Kontext abhängige Variable zu beschreiben. In unserem Fall betrachten wir eine Observation und versuchen, eine Aktion damit vorherzusagen. Dafür erstellen wir einen DT mit der eindimensionalen Beobachtung und den dazugehörigen Aktionen des DRL Modells als Eingabe. Dabei teilen wir das für die Beobachtung definierte Intervall in Sektionen auf, welche durch die Konditionen in den Entscheidungsknoten des DTs gegeben sind. Auf Grundlage der Daten innerhalb einer Sektion, wollen wir eine Vorhersage treffen. Für das MCC Problem beschreiben wir die univariaten Observationen der DTs in der nachfolgenden Abbildung.

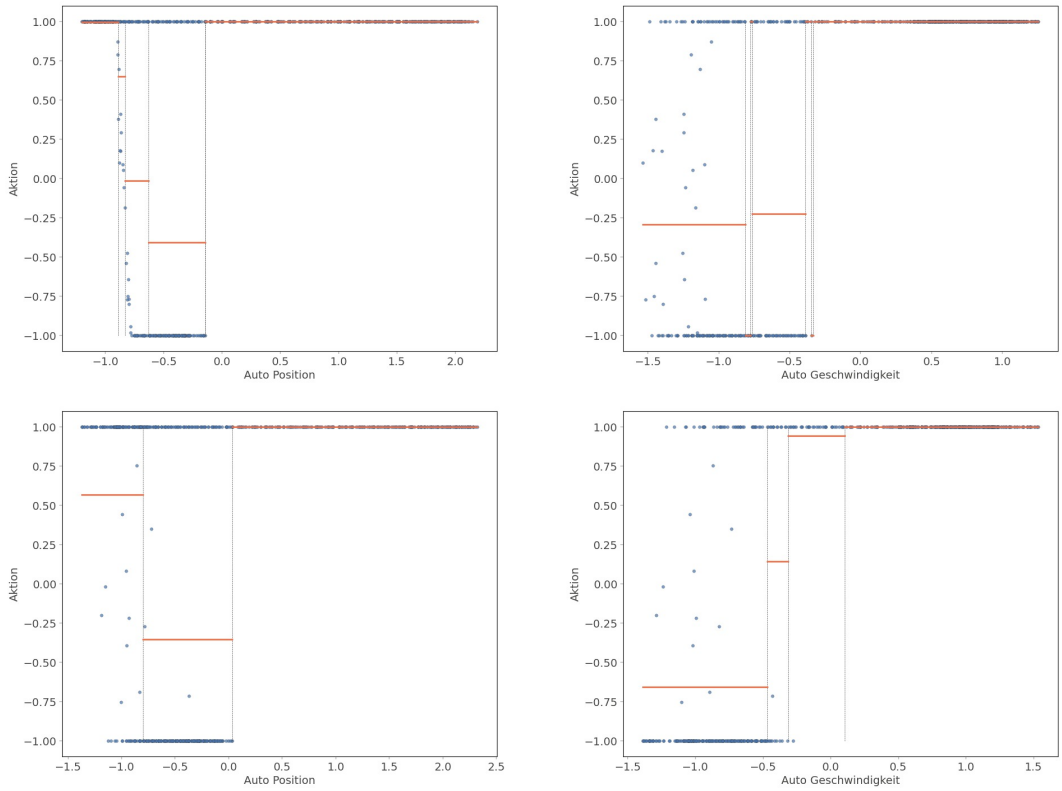


Abbildung 6.6.: MountainCarContinuous-v0 PPO/A2C Decision Trees univariate Observationen - Die oberen beiden Abbildungen beschreiben die univariaten Beobachtungen eines DTs mit Tiefe drei des PPO Modells und die unteren beiden die eines DTs mit Tiefe zwei des A2C Modells. Beide DTs wurden auf den Datenpaaren für zehn Episoden trainiert.

In einem Problem der Regression sind wir nicht in der Lage, die Genauigkeit eines DTs anzugeben. Daher verwenden wir den Coefficient of Determination R^2 (Chicco

u. a., 2021). Dieser kann als der Anteil der Varianz von einer abhängigen Variable interpretiert werden, der durch die unabhängigen Variablen vorhersagbar ist. Wir beschreiben diesen formal als:

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2}{\sum_{i=1}^{n}(Y_i - \bar{Y})^2}, \tag{6.2}$$

wobei $\hat{Y}_i$ der vorhergesagte Wert, Y_i der wahre Wert, $\bar{Y}_i = \frac{1}{n}\sum_{i=1}^{n} Y_i$ und $n \in \mathbb{N}$ die Anzahl der Daten ist. Ein hoher R^2-Wert sagt aus, dass sich die Varianz der abhängigen Variablen durch die unabhängigen Variablen erklären lässt. Im Unterschied dazu beschreibt ein niedriger Wert die Varianz der abhängigen Variable als weniger erklärbar durch die unabhängigen Variablen. Die Implementierung entnehmen wir aus Buitinck u. a. (2013).

Durch Abbildung 6.6 können wir sehen, dass Sektionen existieren, in denen eine univariate Observation die Aktion passend beschreibt. Andererseits gibt es Sektionen, in denen eine alleinige eindimensionale Beobachtung nicht ausreicht, um eine korrekte Prädiktion zu liefern. Der R^2-Wert der univariaten Beobachtungen ist in der folgenden Tabelle zu finden.

DRL Modell	Tiefe	Beobachtung	R^2-Wert	Leistung
PPO	3	Auto Position	52.23%	-14.69 ± 0.0
	3	Auto Geschwindigkeit	64.59%	-99.90 ± 0.0
A2C	2	Auto Position	36.54%	-2.98 ± 0.0
	2	Auto Geschwindigkeit	62.87%	-46.20 ± 0.0

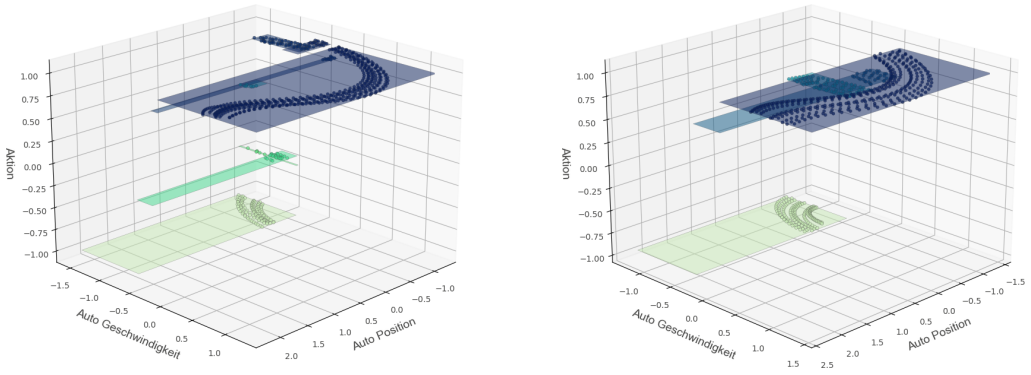
Tabelle 6.6.: MountainCarContinuous-v0 PPO/A2C Coefficient of Determination univariate Beobachtungen - Die hier untersuchten DTs wurden jeweils mit nur einer Observation trainiert. Wir stellen fest, dass die Geschwindigkeit des Auto eine signifikantere Beobachtung darstellt, als die Position des Autos, wenn wir diese interpretieren wollen, siehe R^2-Wert. Die Leistung wird allerdings von der Position des Auto dominiert. Der Kompromiss zwischen Erklärbarkeit und Leistung ist nicht gegeben.

Trotz der eher schlechten Ergebnisse in der Leistung der DTs in Tabelle 6.6, stellen die R^2-Werte eine gute Basis zur Erklärbarkeit dar. Daher wollen wir im Folgenden auf die größeren Sektionen aus Abbildung 6.6, die korrekt vorhergesagt wurden, eingehen.

Im DT des PPO Modells können wir zwei Sektionen durch die Position des Autos beschreiben. Beim Unterschreiten von PA ≈ -0.9 erhalten wir eine annähernd perfekte Prädiktion der zu wählenden Aktion. Dabei befindet sich das Auto auf dem linken Berg und maximiert den Leistungs-Koeffizienten, um den Schwung mitzunehmen. Weiterhin gilt für das Überschreiten von PA ≈ -0.1, dass die Aktion maximiert wird, mit dem Ziel den rechten Berg zu erklimmen. Ein gleiches Verhaltensmuster ist für den DT des A2C Modells zu beobachten, wobei der Schwellenwert bei PA ≈ 0.0 liegt. Bei der Beobachtung der Geschwindigkeit des Autos können wir weitere Ähnlichkeiten in

beiden Modellen erkennen. Im DT des PPO Modells wird die Aktion maximiert ab GA ≈ -0.3 und im DT des PPO Modells ab GA ≈ 0.1. Im Kontext beschreiben wir dies wieder als das Erklimmen des rechten Berges. Wir können davon ausgehen, dass sich die gewonnenen Kenntnisse aus den univariaten Beobachtungen in den Regeln der eigentlich zu untersuchenden DTs wiederfinden lassen.

Die noch nicht beschriebenen Sektionen erfordern für eine genauere Vorhersage beide Observationen. Nicht formal beschreiben wir die Sektionen, welche das Auto mehrmals passiert, als den Aufbau des Schwungs. Dabei tendiert die Vorhersage mehr in Richtung eines niedrigen Leistungs-Koeffizienten. Basierend auf der Geschwindigkeit und der in einer Sektion verbrachten Zeit, werden mehr oder weniger Datenpunkte erstellt. Aus einer geringeren Geschwindigkeit folgt ein größerer Verbrauch der Zeit, daher auch mehr Datenpunkte in einer Sektion, sodass die Prädiktion in Richtung eines niedrigeren Leistungs-Koeffizienten ausschlägt. Für eine Übersicht beider Observationen betrachten wir bivariate Beobachtungen (Denis, 2016, S.88 ff.). Diese befähigen uns, mit zwei unabhängigen Variablen eine im Kontext abhängige Variable zu beschreiben und den vollständigen Entscheidungsraum des jeweiligen DTs im MCC Problem zu erzeugen. Während wir die Vorhersagen mit univariaten Beobachtungen durch Linien darstellen konnten, verwenden wir im Dreidimensionalen Ebenen, welche durch die Konditionen in den Entscheidungsknoten aufgespannt werden. Dazu stellen wir diese in Abbildung 6.7 dar.



(a) PPO MountainCarContinuous-v0 Decision Tree bivariate Observationen; R^2-Wert 100%

(b) A2C MountainCarContinuous-v0 Decision Tree bivariate Observationen; R^2-Wert 100%

Abbildung 6.7.: MountainCarContinuous-v0 PPO/A2C Decision Tree bivariate Observationen - Der Unterschied in der Strategie wird durch die Anzahl und Größe der genutzten Ebenen sowie der Anzahl der Punkte auf diesen verdeutlicht. Beide DTs weisen einen R^2-Wert von 100% auf. Dies sagt aus, dass alle Aktionen durch die Observationen erklärt werden können. Beide Abbildungen beinhalten zehn aufeinander folgende Episoden.

Im MC Problem hat sich die Verzögerung der „Beschleunigung nach Rechts“ Aktion, nachdem genügend Schwung über den linken Berg aufgebaut wurde, als eine Strategie identifiziert, welche bessere Ergebnisse liefert. Der Kontrast zur direkten Beschleunigung ist in Abbildung 6.7 wiederzufinden. In (a) beschreiben die Regeln des DTs eine Strategie, die einen abrupten Wechsel des Drehmoments auf den Motor ausübt. In wenigen Schritten wird der Leistungs-Koeffizient vom Minimum auf das Maximum gesetzt. Während wir in (b) einen flüssigeren Übergang des Drehmoments vom Minimum auf das Maximum beobachten können. Dies ist auf die Anzahl der Punkte auf einer Ebene zurückzuführen. Während wir in (a) durch eine Tiefe von drei des DTs die Bereiche verfeinern, sprich mehr Ebenen zur Verfügung stellen, befinden sich nur wenige Punkte auf diesen. Im Unterschied dazu werden in (b), in welchem wir mit deutlich weniger Ebenen arbeiten, die Punkte jedoch gleichmäßiger verteilen. Weiterhin wollen wir die Regeln der von uns untersuchten DTs aus Tabelle 6.5 und Abbildung 6.7 vorstellen.

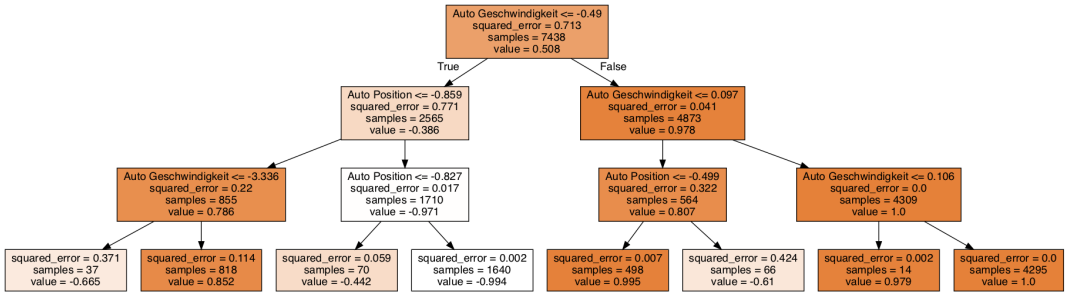


Abbildung 6.8.: MountainCarContinuous-v0 PPO Decision Tree Tiefe 3 - Alle zur Verfügung stehenden Observationen wurden genutzt.

Die Regeln für die untersuchten DTs aus Tabelle 6.5 und Abbildung 6.7 sind in Abbildung 6.8 dargestellt. Wir können für die Abbildung feststellen, dass die meisten Entscheidungspfade die Maximierung des Leistungs-Koeffizienten anstreben. Bestätigt wird dies durch die Datenpaare in den Blättern und die Menge an Punkten auf der höchsten Ebene in Abbildung 6.7a. Einen weniger großen Anteil besitzt die Minimierung der Aktion, welche nur in einem Pfad verfolgt wird. Eine Episode startet grundsätzlich mit einem niedrigen Leistungs-Koeffizienten, denn es gilt $-0.49 < \text{GA} \leq 0.097$ und $\text{PA} > -0.499$. Bei Unterschreitung von $\text{GA} \leq -0.49$ wird die Aktion minimiert. Bislang hat das Auto einen Teil des linken Bergs erklommen. Dieser wird beginnend mit einer mittleren Geschwindigkeit hinabgefahren. Dieser schmale Bereich ist in Abbildung 6.7a zu sehen. Schließlich wird der Schwung genutzt und die Episode bis zum Ende mit maximalem Leistungs-Koeffizient beendet. Damit bestätigen wir die vorab identifizierten Regeln der univariaten Beobachtungen.

Aufgrund der geringen Tiefe des DTs in Abbildung 6.9 können wir diesen vollständig beschreiben. Die Regeln beschreiben den Beginn mit minimalem Leistungs-Koeffizienten, denn es gilt $\text{GA} \leq -0.308$ und $\text{PA} > -0.817$. Dieser wird solange beibehalten, bis PA den Wert von -0.817 unterschreitet. Ab diesem Zeitpunkt ist das Auto den linken Berg weit genug hinaufgefahren, um mit einer positiven Aktion fortzuführen. Der Leistungs-

Koeffizient wird bei der weiteren Überschreitung der Werte in den Entscheidungsknoten gesteigert. Die erste Steigerung findet bei Überschreitung von GA > -0.308 statt. Final maximieren die Regeln den Leistungs-Koeffizienten ab einem Schwellenwert von GA > 0.096. Mit dieser Aktion wird, wie auch schon in den eindimensionalen Beobachtungen in Abbildung 6.6 beschrieben, eine Episode beendet.

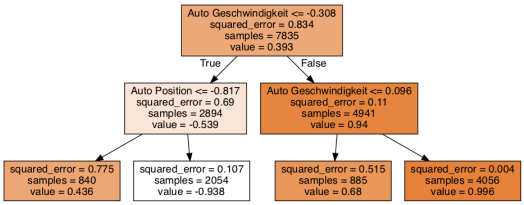


Abbildung 6.9.: MountainCarContinuous-v0 A2C Decision Tree Tiefe 2 - Die maximale Tiefe wurde in jedem Entscheidungspfad erreicht, wobei jede Beobachtung berücksichtigt wurde.

Nachdem wir die Entscheidungen der DTs nachvollzogen haben, wollen wir die Entscheidungen des Orakels mit denen der vorgestellten DTs vergleichen. Im MC Problem konnten wir bereits beobachten, dass DTs in der Lage waren, eine hohe Genauigkeit zu erzielen und die Agenten zu repräsentieren. In der folgenden Abbildung sind die Interaktionsdaten der Orakel und der DTs dargestellt.

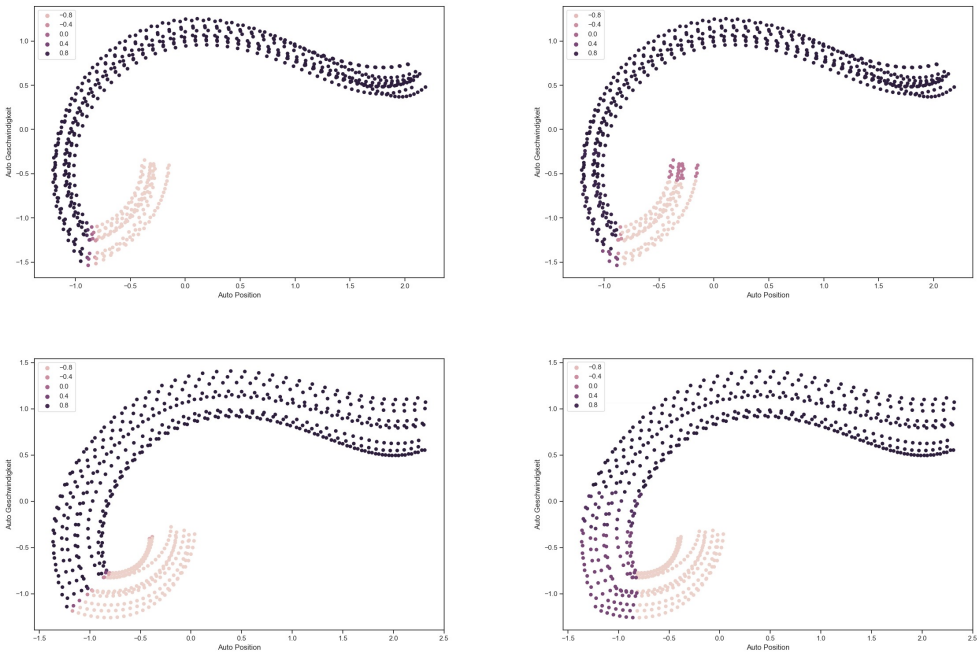


Abbildung 6.10.: MountainCarContinuous-v0 PPO/A2C Decision Tree Vergleich Interaktionsdaten - Auf der linken Seite befinden sich die DRL Modelle (oben PPO / unten A2C) und daneben die darauf aufgebauten DTs. Die Abbildungen basieren auf zehn aufeinander folgenden Episoden.

Erneut beschreiben die in Abbildung 6.10 dargestellten Trajektorien die von uns vorher interpretierten Regeln der DTs aus Abbildung 6.8 und Abbildung 6.9. Der DT des PPO Modells weist bis auf die beginnende Aktion eine fast identische Trajektorie zum Orakel auf. Hingegen stellen die Regeln des DTs des A2C Modells eine von der des DRL Modells abweichende Strategie dar. Das Orakel des A2C Modells verwendet einen abrupten Wechsel des Leistungs-Koeffizienten vom Minimum auf das Maximum, wenn genügend Schwung über den linken Berg aufgebaut wurde. Dieser wurde wie bereits beschrieben, durch einen sich leicht steigernden Wert über die verschiedenen Sektionen des DTs aus Abbildung 6.9, ersetzt. Dadurch ist es möglich, dass einige DTs bessere Ergebnisse erzielen, als das DRL Modell selbst. Sei es durch eine Verfeinerung des Leistungs-Koeffizienten in höheren Tiefen des DTs oder wie in unserem Fall, eine durch den DT abweichende erzeugte Strategie.

Letztlich wollen wir unsere identifizierten Regeln, beziehungsweise deren Leistung, mit bereits vorhandenen vergleichen. Dazu nutzen wir aus (Engelhardt u. a., 2021) den erzeugten DT des Twin Delayed Deep Deterministic Policy Gradient (**TD3**) (Fujimoto u. a., 2018) Modells und die HC Regeln. Weitere HC Regeln sind durch (Xiao, 2019) gegeben.

Methode	**Leistung Methode**	**Modell**	**Leistung Modell**
PPO DT_3	92.25 ± 1.85	PPO	93.02 ± 1.40
A2C DT_2	93.48 ± 1.66	A2C	93.23 ± 0.58
Engelhardt DT	93.90 ± 0.32	TD3	94.00 ± 0.32
Engelhardt HC	97.20 ± 0.63	-	-
Xiao HC	93.35 ± 0.05	-	-

Tabelle 6.7.: MountainCarContinuous-v0 Regeln Vergleich - Die angehängten Indizes bezeichnen die verwendete Tiefe des jeweiligen DTs. Die Anzahl der Episoden, welche wir zur Evaluierung nutzen, beträgt 100.

Bezüglich der Lösungen aus Engelhardt u. a. (2021) für das MCC Problem und unseren gewonnenen Erkenntnissen, wollen wir eine optimale Strategie formulieren. Im Gegenteil zum MC Problem, in welchem wir unmittelbar nach dem Aufbau des Schwungs über den linken Berg nach rechts beschleunigen, existiert in der MCC Umgebung eine unterschiedliche Strategie. Im MCC Problem wird die Belohnung basierend auf der Menge der verbrauchten Leistung in jedem Schritt vergeben, sodass ein hoher ≈ 1 oder niedriger ≈ -1 Leistungs-Koeffizient eine größere negative Belohnung erzielt. Daher bietet sich ein flüssiger Übergang zwischen dem Minimum und Maximums des Leistungs-Koeffizienten an. Abschließend beschreiben wir eine optimale Strategie so, dass der Schwung über den linken Berg mit einem abnehmenden Koeffizienten am Ende aufgebaut und anschließend maximiert wird, um den Berg zur Fahne hinaufzufahren.

6.2.3. CartPole-v1

Das CP Problem stellt für den Menschen ein formal weitaus schwierigeres Problem dar, als das vorherige MC oder MCC Problem. Bei dieser Aussage beziehen wir uns auf die Observationen in der vorzustellenden Umgebung. In den vorherigen Problemen konnten wir bereits feststellen, dass sich trotz einer geringen Anzahl von Observationen, viele Strategien haben ableiten lassen. Resultierend vermuten wir eine steigende Komplexität mit steigender Anzahl an Observationen. Im Folgenden bezeichnen wir die Beobachtungen im CP Problem als: Wagen Position (**WP**), Wagen Geschwindigkeit (**WG**), Stange Winkel (**SW**) und Stange Winkel Geschwindigkeit (**SWG**). Im vorliegenden Problem betrachten wir einen diskreten Aktionsraum und damit ein Problem der Klassifikation, sodass alle vorgestellten Modelle eingesetzt werden können. Die Ergebnisse für DTs verschiedener Tiefe sind in der nachfolgenden Tabelle dargestellt.

Tiefe	**Modell**		
	DQN	PPO	A2C
-	500.00 ± 0.00	500.00 ± 0.00	500.00 ± 0.00
10	500.00 ± 0.00	500.00 ± 0.00	500.00 ± 0.00
5	500.00 ± 0.00	500.00 ± 0.00	500.00 ± 0.00
4	500.00 ± 0.00	500.00 ± 0.00	500.00 ± 0.00
3	500.00 ± 0.00	500.00 ± 0.00	500.00 ± 0.00
2	201.35 ± 37.57	196.24 ± 46.89	199.07 ± 44.17
1	200.04 ± 45.24	198.23 ± 41.78	192.08 ± 51.93

Tabelle 6.8.: CartPole-v1 Decision Tree Leistung - Die evaluierten DTs wurden auf dem in der ersten Zeile angegebenen Modell aufgebaut. Die Anzahl der Episoden, welche wir zur Evaluierung nutzen, beträgt 100.

Mit Tabelle 6.8 stellen wir fest, dass zunächst alle DRL Modelle in der Lage sind, das vorliegende Problem optimal zu lösen, sprich eine kumulierte Belohnung von 500 zu erzielen. Dies gilt auch für alle DTs mit einer minimalen Tiefe von drei. Bei Unterschreitung dieser Tiefe, lässt sich das Problem nicht mehr lösen. Daher werden wir uns hauptsächlich mit DTs der Tiefe drei beschäftigen. Aus unseren Ergebnissen geht hervor, dass die eingesetzten Modelle nicht nur ähnliche Ergebnisse, sondern auch fast identische Strategien liefern. Daraus resultierend haben wir uns für die alleinige Analyse des DTs des DQN Modells im CP Problem entschieden.

Bevor wir die Ergebnisse der DTs untersuchen, wollen wir zunächst eine bereits formulierte Strategie nach Xu (2021) vorstellen. Diese beinhaltet zwei der vier möglichen Observationen, SW und SWG, um das Problem optimal zu lösen. Die Strategie wird formal beschrieben als: `IF` $|SW|$ `< 0.03 THEN (IF SWG < 0 THEN 0 ELSE 1) ELSE (IF SW < 0 THEN 0 ELSE 1)`. Informell charakterisieren wir die Strategie als zwei simple Substrategien: (1) Wenn SW positiv und die Stange nach rechts geneigt ist, schieben wir den Wagen ebenfalls nach rechts, um die Stange wieder in eine aufrechte

Position zu bringen. Für SW negativ verfolgen wir das gleiche Ziel, sodass wir den Wagen nach links schieben. (2) Wenn SWG positiv und die Stange sich vom Zentrum im Uhrzeigersinn fortbewegt, schieben wir den Wagen nach rechts und umgekehrt nach links. Entscheidend ist der absolute Winkel der Stange, welcher diese Substrategien aufruft. Bei einem kleinen Winkel wird die erste Substrategie aufgerufen, andernfalls die zweite Substrategie. Entscheidend in diesem Absatz ist, dass sich das CP Problem mit nur zwei Beobachtungen lösen lässt. Durch diese Reduktion ist es uns möglich[3], die Observationen in für den Menschen verständlichen Abbildungen darzustellen und zu interpretieren.

Im Folgenden wollen wir die oben erwähnten Regeln mit denen des DTs mit Tiefe zwei aus Tabelle 6.8 vergleichen. Dazu stellen wir den DT wie folgt dar.

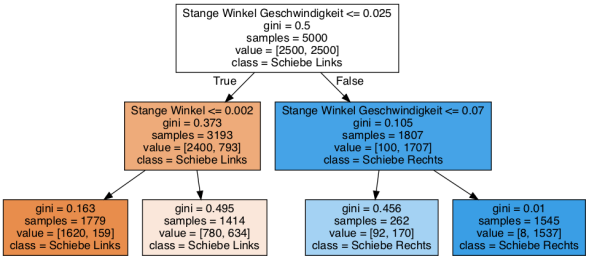


Abbildung 6.11.: CartPole-v1 DQN Decision Tree Tiefe 2 - Es wurden zwei von vier möglichen Observationen genutzt. Die Genauigkeit liegt bei 82.14%.

Anhand von Abbildung 6.11 sehen wir, dass sich die genutzten Observationen mit denen der vorgestellten Regeln decken. Außerdem lassen sich signifikante Unterschiede ermitteln. In der Strategie von Xu (2021) wird der erste Split auf dem absoluten Wert von SW gesetzt, beim DT hingegen wird auf SWG gesplittet. Der Split auf einem absoluten Wert ist in der von uns genutzten Bibliothek nicht möglich, sodass wir eine weitere Tiefe hinzufügen müssen, um diesen abbilden zu können. Das macht sich in den mittleren Blättern des DTs bemerkbar, da diese die Genauigkeit des gesamten DTs deutlich reduzieren. Die Genauigkeit des linken mittleren Blattes beträgt 55.16% und die des mittleren Rechten 64.86%.

Im Folgenden wollen wir den DT der Tiefe drei für die Ergebnisse in Tabelle 6.8 vorstellen. Wenn wir den DT aus Abbildung 6.12 nach seiner Farbgebung beurteilen, stellen wir fest, dass dieser fast symmetrisch zum Wurzelknoten ist, wenn wir die Aktionen vertauschen. Mit der Symmetrie und den identisch aufgeteilten Aktionen auf dem linken und rechten Entscheidungsbaum vom Wurzelknoten vermuten wir ein alternierendes Verhaltensmuster. Dieses beschreiben wir durch das Schieben des Wagens nach links, wenn dieser vorher nach rechts geschoben wurde und umgekehrt.

[3]Diese Möglichkeit besteht unter der Voraussetzung, dass der DT ebenfalls nur zwei Observationen benötigt, um die Entscheidungen zu beschreiben.

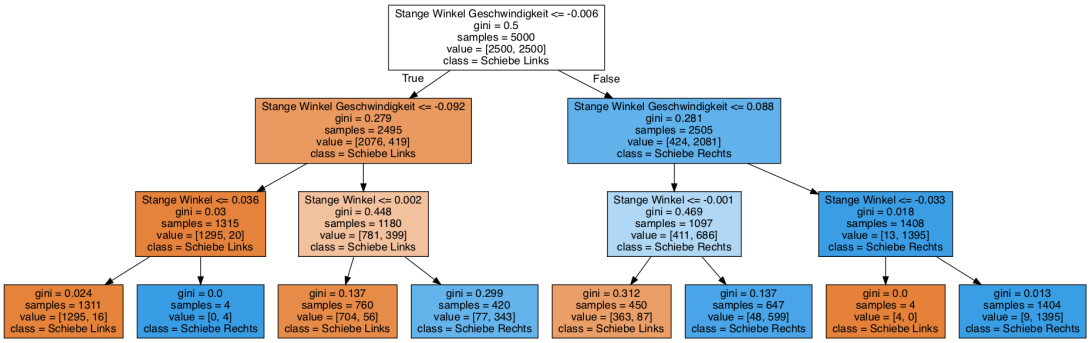


Abbildung 6.12.: CartPole-v1 DQN Decision Tree Tiefe 3 - Selbst in einem DT der Tiefe drei werden nur zwei von vier möglichen Observationen benötigt, um die Entscheidungen vorherzusagen.

Zudem ist es auffällig, dass die gleichen Observationen genutzt werden, wie in den HC Regeln (Xu, 2021) und im DT mit Tiefe zwei aus Abbildung 6.11. In der MCC Umgebung, beziehungsweise in Problemen der Regressionen, können wir das Maß an Interpretierbarkeit der Observationen anhand des R^2-Werts bestimmen. Dahingegen haben wir im MC und CP Problem, sprich einem Problem der Klassifikation, bislang nur die Genauigkeit und die kumulierten Belohnungen genutzt, um die Qualität der Observationen zu bewerten. Ein Verfahren, welches auch für Probleme der Klassifikation zur Identifikation eines weiteren Qualitätsmerkmals beiträgt, ist durch die Relevanz von Beobachtungen (engl. Feature Importance) (**FI**) (Pedregosa u. a., 2011) gegeben. Diese berechnet die durchschnittliche Abnahme in der Impurity-Funktion, siehe Gleichung (4.4), gewichtet mit der Wahrscheinlichkeit diesen Knoten zu erreichen. Die Wahrscheinlichkeit, einen Knoten vorzufinden, drücken wir als die Datenpaare die den Knoten erreichen, geteilt durch die gesamte Anzahl an Datenpaaren, aus. Formal beschreiben wir die Relevanz der Beobachtungen mit der Wichtigkeit eines Knotens, siehe Gleichung (6.4), als:

$$FI_i = \frac{\sum_m IM(Q_m^i)}{\sum_m IM(Q_m)}. \tag{6.3}$$

Dabei stellt i eine Observation und insbesondere im Zähler von Gleichung (6.3) diejenige dar, auf welcher gesplittet wird. Weiterhin gibt IM Auskunft über die Relevanz eines Knotens Q_m mit:

$$IM(Q_m) = w_m H(Q_m) - w_m^{links} H(Q_m^{links}) - w_m^{rechts} H(Q_m^{rechts}). \tag{6.4}$$

Schließlich erhalten wir den normalisierten Wert der FI, indem wir die Relevanz einer einzelnen Beobachtung durch die Summe aller relevanten Observationen teilen:

$$FI_i^{norm} = \frac{FI_i}{\sum_j FI_j}. \tag{6.5}$$

Im CP Problem können wir für verschiedene Tiefen der DTs normierte FIs feststellen und bilden diese in Tabelle 6.9 ab.

Tiefe	Relevanz Beobachtungen			
	WP	WG	SW	SWG
10	0.0%	0.0%	23.42%	76.58%
5	0.0%	0.0%	18.34%	81.66%
4	0.0%	0.0%	24.29%	75.71%
3	0.0%	0.0%	21.46%	78.54%
2	0.0%	0.0%	19.76%	80.24%
1	0.0%	0.0%	0.0%	100.0%

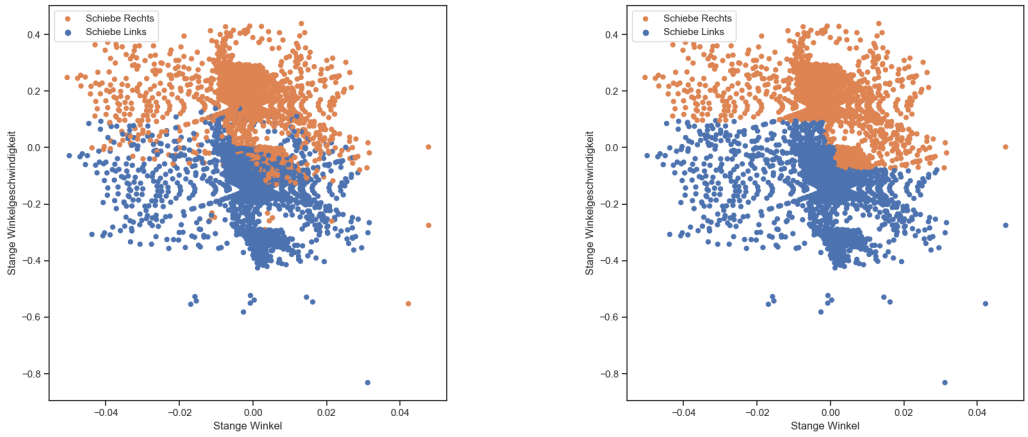
Tabelle 6.9.: CartPole-v1 DQN Decision Trees Feature Importance

Aus Tabelle 6.9 können wir entnehmen, dass die DTs mit den abgebildeten Tiefen immer nur die gleichen Observationen, namentlich SW und SWG, betrachten. Für das CP Problem erzielen diese optimale Lösungen, weisen jedoch im Hinblick auf eine größere Zeitspanne einer Episode noch Schwächen auf. Eine Episode terminiert in der CP Umgebung im besten Fall nach 500 Zeitschritten. Angenommen wir betrachten eine „CartPole-Extreme“ Umgebung mit 1×10^4 zu absolvierenden Zeitschritten je Episode, dann werden die Regeln des DTs mit hoher Wahrscheinlichkeit keine optimalen Lösungen erzeugen. Aufgrund der Vernachlässigung von WP und WG kann der DT nicht erkennen, welche Position er gerade besitzt und überschreitet die Grenzen der eindimensionalen Strecke. In einem Experiment haben wir den DT mit Tiefe drei aus Abbildung 6.12 in zehn aufeinander folgenden Episoden in der CartPole-Extreme Umgebung getestet. Dabei konnte das Orakel stets die optimale Lösung, sprich $1 \times 10^4 \pm 0.0$ kumulierte Belohnungen, erhalten. Hingegen konnten die Regeln des DTs mit durchschnittlich kumulierten Belohnungen von 4837.60 ± 3594.68 keine optimale Lösungen nachweisen. Grund dafür besteht in der bereits erwähnten Unkenntnis der aktuellen Position, sodass die Grenzen überschritten werden.

Bezüglich der gewählten Splitpunkte in den Knoten des DTs aus Abbildung 6.12 wollen wir eine Strategie formulieren. Beginnend mit dem Splitpunkt auf SWG ≤ -0.006 wird grundsätzlich die Richtung der Geschwindigkeit untersucht. Wie wir schon in den HC Regeln von Xu (2021) untersucht haben, ist das Vorzeichen von SWG ein Entscheidungsgrund, um eine Aktion verschieden zu klassifizieren. In der darunterliegenden Ebene des DTs wird dieser Entscheidungsgrund verfeinert, indem zusätzlich geprüft wird, ob im linken Pfad SWG ≤ -0.092 und im rechten Pfad SWG ≤ 0.088 gilt. Nicht formal beschreiben wir diese Entscheidungen als die Vermeidung einer zu weiten Entfernung der Stange vom Punkt des Gleichgewichts. Letztlich kann dieser nicht lange gehalten werden, da laut verfügbaren Aktionen der Wagen immer in eine Richtung geschoben werden muss. Eine „Schiebe nicht“ Aktion, die sich im MC Problem als leistungsminimierend herausstellte, wäre im CP Problem durchaus nützlich, um das Gleichgewicht halten zu können. Die vorletzte Tiefe behandelt lediglich SW. Auf dieser Ebene kann die zuvor von der SWG festgelegte Aktion als die andere gewählt werden. Alle Knoten auf der linken Seite des Wurzelknotens bis zu dieser Tiefe wurden als den Wagen nach links schiebend eingeordnet und umgekehrt. In dieser Hinsicht

ist es in den Blättern so, dass, wenn SWG negativ und SW positiv ist, die „Schiebe Rechts“ Aktion klassifiziert wird. Dieses Verhalten dient wiederum der Stabilisierung der Stange im Gleichgewicht. Eine Verfeinerung, sodass diese die Stange nicht zu weit vom Punkt des Gleichgewichts entfernt, ist durch die äußeren beiden Konten der vorletzten Schicht gegeben.

Zum Vergleich des DQN Orakels und dem darauf aufgebauten DT aus Abbildung 6.12 stellen wir deren Interaktionsdaten bezüglich der durch die FI identifizieren Observationen im zweidimensionalen Raum dar.



(a) CartPole-v1 DQN Oracle nach Tabelle 6.8

(b) CartPole-v1 Decision Tree Tiefe 3 nach Abbildung 6.12

Abbildung 6.13.: CartPole-v1 DQN DT/HC Regeln Vergleich Interaktionsdaten - Die Genauigkeit des DTs liegt bei $94,14\%$. Die Daten wurden über zehn aufeinander folgenden Episoden gesammelt.

Eine Auffälligkeit im Vergleich des Orakels zum darauf aufgebauten DT aus Abbildung 6.13 liegt in der Umgebung von SW ≈ -0.1 und SWG ≈ 0. In (a) strebt das Orakel eine diagonale Trennung zwischen den zu klassifizierenden Aktionen in diesem Bereich an. Der von uns untersuchte DT in (b) ist nur in der Lage, den zu betrachtenden zweidimensionalen Raum mit Rechtecken zu beschreiben. Aus dieser Wahl ergibt sich ein ODT, der den Raum mit diagonalen Schnitten beschreiben kann, als mögliche bessere Wahl für das CP Problem. Abgesehen von diesem Bereich liefert der DT ein nachvollziehbares Abbild des Orakels mit einer hohen Genauigkeit. Mit (b) stützen wir die von uns im Vorherigen formulierte Strategie des DTs.

In einem letzten Schritt wollen wir die von uns identifizierten Regeln mit bereits existierenden vergleichen, um auch hier die Qualität dieser zu bestätigen. Dazu stellen wir die von uns erstellten DTs sowie die HC Regeln aus Xu (2021) und Xiao (2019) vor.

Methode	Leistung Methode	Modell	Leistung Modell
DQN DT_3	500.0 ± 0.00	DQN	500.0 ± 0.00
PPO DT_3	500.0 ± 0.00	PPO	500.0 ± 0.00
A2C DT_3	500.0 ± 0.00	A2C	500.0 ± 0.00
Xiao HC	500.0 ± 0.00	-	-
Xu HC	500.0 ± 0.00		-

Tabelle 6.10.: CartPole-v1 Regeln Vergleich - Die angehängten Indizes bezeichnen die verwendete Tiefe des jeweiligen DTs. Die kumulierten Belohnungen wurden in 100 aufeinander folgenden Episoden erzielt.

6.2.4. Acrobot-v1

Die AB Umgebung stellt das abstrakteste und am schwierigsten interpretierbare der von uns untersuchten Probleme bezüglich seiner Observationen dar. Im Vergleich zum CP Problem, wird die Anzahl der Beobachtungen nochmals um zwei erhöht. Zudem erweisen sich vier der Observationen als abstrakter, da diese durch trigonometrische Funktionen (Papula, 2011, S.243) dargestellt werden. Für die Untersuchung arbeiten wir weiter mit den in Tabelle 5.1 etablierten Bezeichnungen der Beobachtungen. Die erreichten kumulierten Belohnungen der DTs für das AB Problem sind in der folgenden Tabelle hinterlegt.

Tiefe	Modell		
	DQN	PPO	A2C
-	-71.18 ± 11.18	-79.77 ± 8.91	-77.79 ± 9.53
10	-77.81 ± 11.58	-80.08 ± 11.90	-83.73 ± 14.24
5	-78.13 ± 11.45	-80.48 ± 16.27	-86.94 ± 19.65
4	-80.47 ± 12.02	-89.96 ± 22.79	-84.86 ± 22.58
3	-81.53 ± 12.23	-88.01 ± 29.41	-92.01 ± 35.01
2	-82.48 ± 13.76	-86.61 ± 37.69	-89.41 ± 45.31
1	-87.18 ± 15.39	-111.38 ± 79.43	-104.46 ± 76.03

Tabelle 6.11.: Acrobot-v1 Decision Tree Leistung - Die evaluierten DTs wurden auf dem in der ersten Zeile angegebenen Modell aufgebaut. Die Anzahl der Episoden, welche wir zur Evaluierung nutzen, beträgt 100.

Aufgrund der angegebenen Leistung in Tabelle 6.11 untersuchen wir die aufgebauten DTs des DQN Modells. Grundsätzlich stellen wir anhand der Standardabweichung fest, dass sowohl die Orakel als auch die DTs keine robuste Strategie zur Lösung der Umgebung liefern. Insbesondere streuen die Ergebnisse des PPO und A2C Modells sehr stark in geringen Tiefen, sodass wir diese vernachlässigen.

Indem wir die FI in verschiedenen Tiefen überprüfen, können wir überflüssige Beobachtungen auf der Grundlage der Menge eliminieren. Nachdem wir die prozentuale Relevanz überprüft haben, können wir entscheiden, welche Beobachtungen im folgenden Kontext analysiert werden sollen. Basierend auf den Observationen werden wir die DTs iterativ nach ihrer Höhe interpretieren. Ergo, beginnend mit der Analyse des DTs mit Tiefe eins bis zu einem DT mit Tiefe drei. Durch dieses Vorgehen ist es wahrscheinlicher, den gesamten Entscheidungsprozess, beziehungsweise zunächst Teile dieses Prozesses zu verstehen, um diesen anschließend erklärbar darzustellen und interpretieren zu können.

Tiefe	**Relevanz Beobachtungen**					
	$\cos(\theta_1)$	$\sin(\theta_1)$	$\cos(\theta_2)$	$\sin(\theta_2)$	v_1	v_2
10	11.31%	1.01%	3.87%	0.0%	81.43%	2.38%
5	0.0%	0.75%	0.0%	3.27%	84.75%	11.23%
4	7.48%	0.0%	1.04%	0.0%	87.31%	4.17%
3	10.50%	0.0%	0.0%	0.0%	84.21%	5.29%
2	0.0%	0.0%	0.0%	0.0%	84.66%	15.34%
1	0.0%	0.0%	0.0%	0.0%	100.0%	0.0%

Tabelle 6.12.: Acrobot-v1 DQN Decision Trees Feature Importance

Mit Tabelle 6.12 können wir die Anzahl der zu beachtenden Observationen bis zu einer Tiefe von drei um die Hälfte reduzieren. Als besonders relevant erweist sich in den DTs die Observation v_1, sprich die gemessene Geschwindigkeit am oberen Gelenk. Diese Relevanz wird von den Regeln aus Verma u. a. (2018) geteilt. Die Regeln lauten: `IF (0.1357 + peek(`h_4`,`-1`)) < 0 THEN 2 ELSE 0`. Dabei verstehen wir `peek(`$X, -1$`)` als eine Funktion, welche die zuletzt beobachtete Observation X zurückgibt. In dem Fall der vorgestellten Regeln betrachten wir die fünfte Observation, da mit null gestartet wird, also v_1. Die vorgestellten Regeln liefern für das AB Problem eine durchschnittliche kumulierte Belohnung in 100 Episoden von -92.34 ± 22.01 und sind damit vergleichbar zu den Regeln eines auf dem DQN Modells aufgebauten DT mit Tiefe eins. Dieser ist in der folgenden Abbildung dargestellt.

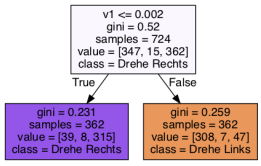


Abbildung 6.14.: Acrobot-v1 DQN Decision Tree Tiefe 1 - Der Splitpunkt wurde auf der Geschwindigkeit des oberen Gelenks gesetzt.

Der DT in Abbildung 6.14 besagt, dass das Gelenk nach rechts gedreht werden soll, wenn $v_1 \leq 0.002$ gilt, sonst linksherum. Durch das Hin- und Herschwingen wird Schwung aufgebaut, um das untere Glied über die gekennzeichnet Linie zu schwingen.

Beim Hinzufügen einer Tiefenschicht, beobachten wir zusätzlich v_2, sprich die Geschwindigkeit des unteren Gelenks. Daraus resultierend fließen alle beobachtbaren Geschwindigkeiten mit in den Entscheidungsprozess ein. Aus Abbildung 6.15 leiten wir die folgende Strategie ab: Wie auch im DT mit Tiefe eins, prüfen die Regeln grundsätzlich, ob v_1 positiv oder negativ ist. Im Fall, dass v_1 positiv und v_2 negativ ist, wird ein negatives Drehmoment auf das untere Gelenk ausgeübt. Ziel dabei ist es, das untere Glied auf die Gerade des oberen Gliedes zu schwingen, sodass auf v_1 durch die Schwingung von v_2 eine positive Geschwindigkeit ausgeübt wird. Im zweiten Fall seien beide Geschwindigkeiten positiv. Damit v_2 wieder auf die Gerade von v_1 geschwungen werden kann, muss auf v_2 ein negatives Drehmoment ausgeübt werden. Durch die negative Schwingung von v_2, wird v_1 ebenfalls negativ beschleunigt. Analog beschreiben wir die Strategie für $v_1 \leq 0.023$. Durch diese Formulierung soll eine Grundidee für die Verhaltensweise der Regeln geschaffen werden und stellt somit keine genaue Beschreibung des DTs dar.

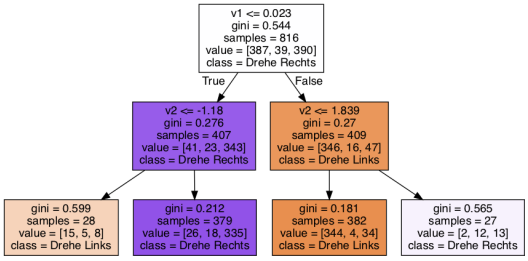


Abbildung 6.15.: Acrobot-v1 DQN Decision Tree Tiefe 2 - Alle beobachtbaren Geschwindigkeiten werden zur Entscheidungsfindung genutzt.

In einem DT der Tiefe drei wird eine weitere Beobachtung mit in den Entscheidungsprozess einbezogen. Diese wird durch den Winkel zwischen dem äußeren Glied und dem äußeren Gelenk als $\cos(\theta_1)$ beschrieben. Der DT ist durch die nachstehende Abbildung gegeben.

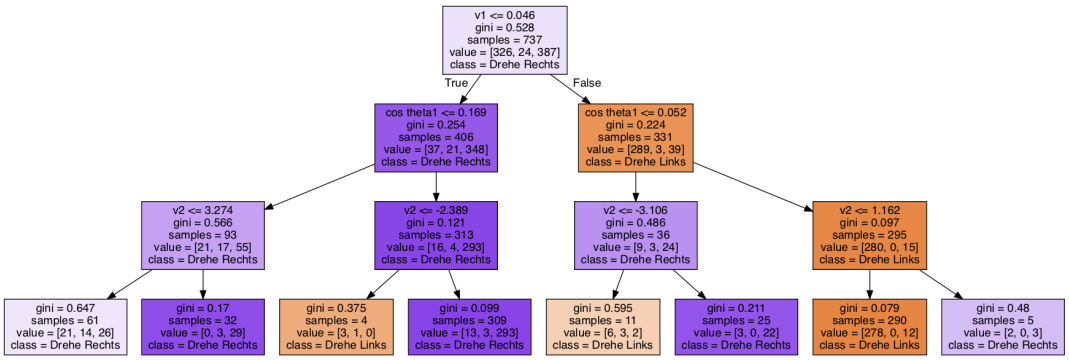
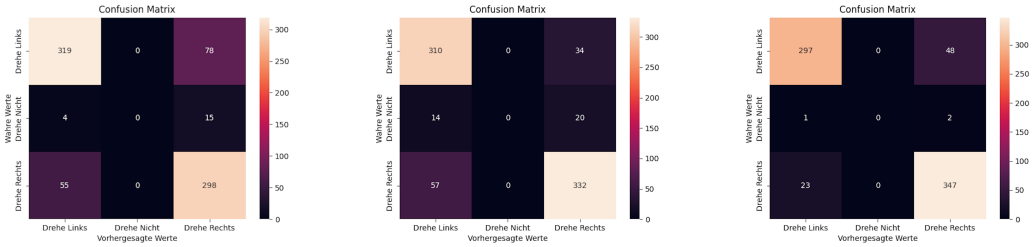


Abbildung 6.16.: Acrobot-v1 DQN Decision Tree Tiefe 3 - Zu den beobachtbaren Geschwindigkeiten wird die Observation des Winkels zwischen dem äußeren Glied und äußerem Gelenk in Abhängigkeit der Kosinus-Funktion betrachtet.

Basierend auf den gewonnenen Erkenntnissen wollen wir den DT in Abbildung 6.16 vorstellen. Anstelle der Geschwindigkeit v_2 in Tiefe eins, verwendet dieser den Winkel $\cos(\theta_1)$ und im Anschluss die beschriebene Observation v_2. Für ein besseres Verständnis des neu etablierten Parameters, wird dieser in Grad durch die Umkehrfunktion $\arccos(\cos(\theta_1))$ angegeben. Der Entscheidungsprozess des DTs beginnt wie im Vorherigen mit der Abfrage auf eine positive oder negative Geschwindigkeit, genauer gesagt, wenn $v_1 \leq 0.046$. Für $v_1 > 0.046$ betrachten wir in dem folgenden Entscheidungsknoten $\cos(\theta_1) \leq 0.052$, sprich einem kleineren oder gleich großem Winkel als 87.02°. Nicht formal beschreiben wir diesen als Indikator für die Ausrichtung der Geschwindigkeit v_1. Mit diesem weiteren Parameter ist es möglich, die Position links- oder rechtsseitig des Nullwinkels, beziehungsweise des oberen Glieds, einzuordnen. Mit der Zuordnung und den bereits beschriebenen Auswirkungen der Geschwindigkeit v_2, kann eine Aktion klassifiziert werden.

Die Genauigkeit der DTs sind durch die jeweilige Confusion Matrix $\mathcal{C}$ (**CM**) (Pedregosa u. a., 2011) dargestellt. Grundsätzlich stellen Zeilen die wahren Werte und Spalten die vorhergesagten Werte dar. Dabei sind die Einträge $\mathcal{C}_{i,j}$ auf der Hauptdiagonalen mit $i, j \in \mathbb{N}$ und $i = j$ korrekt vorhergesagt worden. Eine perfekte Vorhersage aller Werte ergibt sich durch die Spur der Matrix (Arens u. a., 2013, S.505), wobei diese gleich der Anzahl aller vorherzusagenden Werte sein muss. Fehlerhafte Prädiktionen befinden sich jenseits der Hauptdiagonalen und werden als die Aktion der jeweiligen Spalte klassifiziert.



(a) DQN Acrobot-v1 Decision Tree Tiefe 1 nach Abbildung 6.14

(b) DQN Acrobot-v1 Decision Tree Tiefe 2 nach Abbildung 6.15

(c) DQN Acrobot-v1 Decision Tree Tiefe 3 nach Abbildung 6.16

Abbildung 6.17.: Acrobot-v1 DQN Decision Trees Confusion Matrix

Die vorliegenden CMs wurden in zehn aufeinander folgenden Episoden evaluiert. Diese lassen sich auch durch ihre Färbung beschreiben. Ein helles gefärbtes Quadrat steht für eine große Anzahl an klassifizierten Aktionen und ein dunkles Quadrat für eine kleinere Anzahl. Die Genauigkeiten sind wie folgt ausgefallen: (a) 80.23%, (b) 83.70% und (c) 89.69%. Auffällig in allen CMs ist, dass die „Drehe Nicht“ Aktion nicht verwendet wird, wie wir auch in Abbildung 6.18 feststellen.

Unsere Formulierungen bezüglich der Auswirkungen der verschiedenen Observationen auf die zu klassifizierende Aktionen stellen wir in Abbildung 6.18 dar. Die Abbil-

dung nutzt weiterhin die vorgestellten DTs der Tiefe zwei und drei, um die Interaktionsdaten des DQN Orakels mit denen des DTs zu vergleichen.

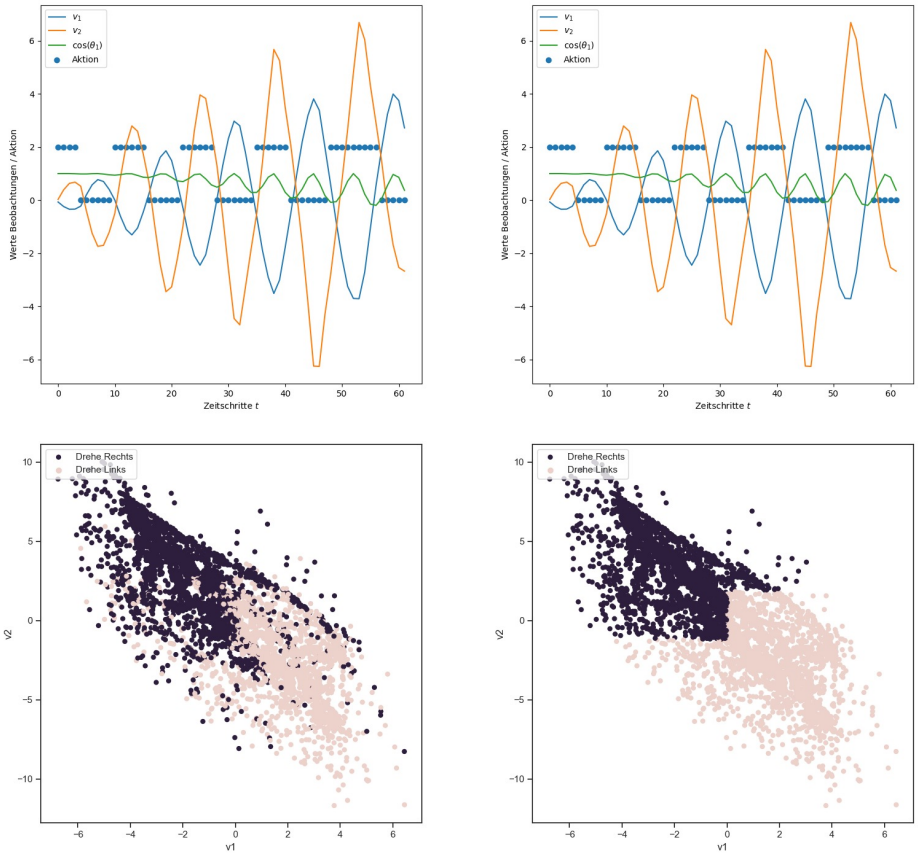


Abbildung 6.18.: Acrobot-v1 DQN/DT Decision Tree Vergleich Interaktionsdaten - Oben befinden sich die Daten des DTs mit Tiefe drei, wobei links das Orakel und rechts der DT dargestellt wird. Unten links sind die Interaktionsdaten des Orakels und unten rechts die des DTs mit Tiefe zwei beschrieben. Die oberen Abbildungen zeigen die Interaktionsdaten einer zufälligen Episode. Im Fall des DT mit Tiefe zwei wurden 100 aufeinander folgende Episoden evaluiert.

Wie schon im Vorherigen beobachtet, verwendet der DT sowie das Orakel aus Abbildung 6.18 die „Drehe Nicht“ Aktion nicht. Ähnlich zum MC Problem, basiert die kumulierte Belohnung lediglich auf der Zeit, die zur Lösung des Problems gebraucht wird. Daher ist es vermutlich nicht vorteilhaft, eine Aktion auszuführen, die vorerst keinen signifikanten Fortschritt in der Umgebung erzielt. In den von uns nicht untersuchten Tiefen könnte dies einen Vorteil bringen, wird jedoch aufgrund des Umfangs dieser Arbeit nicht weiter untersucht. Im Falle des DTs mit Tiefe zwei können wir feststellen, dass dieser zwar gute Ergebnisse liefert, jedoch mehrere Beobachtungen benötigt, um das Orakel mit einer höheren Genauigkeit zu repräsentieren.

Aufbauend auf unseren Erkenntnissen, ist es uns mit diesem Wissen nicht möglich, eine optimale Strategie für das AB Problem zu formulieren. Wir konnten jedoch gute und interpretierbare Lösungen mit den von uns beschriebenen Observationen finden, welche eine Grundlage für weitere Interpretationen mit DTs in größeren Tiefen ermöglicht. Für eine analytische Lösung des AB Problems verweisen wir auf (Chen u. a., 2018).

Für das AB Problem konnten wir als vergleichbare regelbasierte Lösung nur die in (Verma u. a., 2018) veröffentlichten Ergebnisse finden. Diese wurden bereits beschrieben. Dabei haben wir festgestellt, dass bereits mit einem DT mit Tiefe eins vergleichbare und in größeren Tiefen sogar höhere kumulierte Belohnungen erreicht werden können.

6.3. DeepRED

In dieser Sektion wollen wir uns mit der Regelextraktion aus DNNs beschäftigen. Wie schon in Sektion 3.4 erläutert, verwenden DRL Modelle DNNs, um die Strategie oder die Wertfunktion zu approximieren und Entscheidungen zu formulieren. Daraus resultierend folgt unsere Motivation, Regeln aus diesen Blackboxen zu extrahieren. Mit der Verwendung von DNNs als Funktion-Approximatoren, kann man mit ziemlich hoher Sicherheit sagen, dass DNNs, die auf den Interaktionsdaten basieren, auch in der Lage sind, eine nicht-lineare Funktion zu lernen, welche die Entscheidungen des DRL Modells repräsentieren. Dazu sammeln wir die Daten, welche durch die Interaktion der DRL Modelle in der jeweiligen Umgebung erzeugt wurden. Unser Interesse an der Verwendung eines anderen DNNs als dem internen DNN des DRL Modells, besteht darin, flexibler zu sein und vorerst Komplexität zu vermeiden. SB3 bietet einige wichtige Änderungen an den Hyperparametern der DRL Modell internen DNNs an. Jedoch erreichen wir eine höhere Flexibilität, wenn wir Modelle verwenden, die wir selbst erstellt haben und in fast jeder Hinsicht ändern können. Eine Reduktion der Komplexität erreichen wir vorerst, indem wir es vermeiden, die Ausgaben mehrerer DNNs eines DRL Modells, wie es im PPO (Schulman u. a., 2017) oder auch im A2C Modell (Hao u. a., 2020) gegeben ist, zusammenzuführen und eine gemeinsame Ausgabe zu erzeugen. Zunächst wollen wir die Leistungen der erstellten DNNs in der nachstehenden Tabelle vorstellen. Aufgrund der Maximierung über eine diskrete Auswahl von Aktionen, sind keine Leistungen des DQN Modells und dem darauf aufgebauten DNN für das MCC Problem vorhanden.

Mit Tabelle 6.13 bestätigen wir unsere Annahme, dass DNNs in der Lage sind, das Orakel auf Grundlage der Interaktionsdaten zu repräsentieren und ähnliche Leistungen hervorzubringen. Diese stellen eine solide Basis zur weiteren Untersuchung der Probleme mit dem DeepRED Modell dar. Aufgrund des genutzten C4.5 Algorithmus zur Erzeugung eines DTs für die jeweilige Schicht im DeepRED Modell, können wir nur Probleme mit einem diskreten Aktionsraum behandeln. Der Vollständigkeit halber sind die Leistungen der DNNs für das MCC trotzdem in der obigen Tabelle verzeichnet.

Umgebung	**Modell**		
	DQN	PPO	A2C
MC	-96.58 ± 4.58	-96.82 ± 8.23	-100.56 ± 21.38
-	-97.03 ± 5.21	-99.82 ± 2.35	-99.70 ± 8.36
MCC	-	93.02 ± 1.40	93.23 ± 0.58
-	-	93.35 ± 1.65	94.44 ± 1.23
CP	500.00 ± 0.00	500.00 ± 0.00	500.00 ± 0.00
-	500.00 ± 0.00	500.00 ± 0.00	500.00 ± 0.00
AB	-71.18 ± 11.18	-79.77 ± 8.91	-77.79 ± 9.53
-	-71.66 ± 11.58	-79.02 ± 9.26	-77.17 ± 9.21

Tabelle 6.13.: Deep Neural Network Leistung - Die evaluierten DNNs wurden auf dem Modell in der jeweiligen Spalte der angegebenen Umgebung aufgebaut. Die Anzahl der Episoden, welche wir zur Evaluierung nutzen, beträgt 100.

In Blackbox Modellen existiert neben der Genauigkeit ein weiteres Qualitätsmerkmal: die Vertrauenswürdigkeit (engl. Fidelity) (Yang, 2019). Fidelity beschreibt die Messung der Erklärbarkeit für eine Vorhersage in einem Blackbox Modell. In einigen Methoden, in welchen die Erklärbarkeit im Vordergrund steht, wird die Fidelity der Genauigkeit gegenüber bevorzugt (Jagielski u. a., 2020). Im DeepRED Algorithmus wird die Qualität der Erklärbarkeit in Form des Fidelity Werts angegeben. Ein hoher Fidelity Wert steht für ein nachvollziehbares Ergebnis, wohingegen ein niedriger Wert auf ein schleierhaftes Resultat hindeutet. Durch die gegebene Implementierung sind wir in der Lage, den Fidelity Wert jeder Schicht eines DNNs zu bestimmen. Inwiefern dieser Auskunft über die Leistung gibt, klären wir in den folgenden Abschnitten und betrachten die kumulierten Belohnung der identifizierten Regeln des DeepRED Modells in der Tabelle 6.14.

Mithilfe von Tabelle 6.14 stellen wir fest, dass das DeepRED Modell in der Lage ist, Regeln für RL Probleme zu finden. Bevor wir uns intensiver mit den Ergebnissen befassen, wollen wir allgemeine Aussagen zu den erzeugten Regeln des DeepRED Modells formulieren. Wie schon in der Beschreibung der referenzierten Tabelle erwähnt, setzen wir den initialen DT für die letzte Schicht auf eine bestimmte Tiefe, sodass dieser die Regeln $R^v_{h_k \to h_{k+1}}$ auch mit dieser Tiefe beschreibt. Analog werden die DTs auf den anderen k Schichten durch den C4.5 Algorithmus erzeugt. Die Wahrscheinlichkeit, dass ein DT auf einer anderen Schicht unterschiedliche Regeln und einen unterschiedlichen Splitpunkt findet, ist ziemlich hoch. Aufgrund dieser Unterschiede folgen bei der Zusammenführung der Regeln meist extrem große DTs, die nicht mehr interpretierbar sind. Im Folgenden wollen wir auf die jeweiligen Umgebungen eingehen und die dargestellten Ergebnisse erläutern.

Umgebung	**Tiefe**	**Modell**		
-	-	DQN	PPO	A2C
MC	10	-123.23 ± 38.58	-127.96 ± 12.12	-200.00 ± 0.00
	5	-129.09 ± 37.94	-130.52 ± 11.55	-200.00 ± 0.00
	4	-126.68 ± 34.70	-145.27 ± 11.29	-200.00 ± 0.00
	3	-149.26 ± 37.37	-152.84 ± 12.38	-200.00 ± 0.00
	2	-152.84 ± 40.56	-154.15 ± 11.93	-200.00 ± 0.00
	1	-161.23 ± 38.58	-163.71 ± 11.62	-200.00 ± 0.00
CP	10	500.00 ± 0.00	500.00 ± 0.00	500.00 ± 0.00
	5	500.00 ± 0.00	500.00 ± 0.00	500.00 ± 0.00
	4	500.00 ± 0.00	500.00 ± 0.00	500.00 ± 0.00
	3	497.52 ± 1.63	499.14 ± 3.61	497.17 ± 1.89
	2	197.35 ± 42.07	195.10 ± 41.10	182.86 ± 21.03
	1	183.23 ± 35.02	187.80 ± 45.00	161.39 ± 19.99
AB	10	-82.68 ± 24.95	-86.11 ± 24.81	-96.11 ± 42.18
	5	-86.44 ± 20.66	-88.23 ± 19.74	-100.69 ± 34.04
	4	-88.08 ± 18.55	-90.10 ± 25.98	-101.07 ± 59.16
	3	-90.23 ± 19.58	-89.14 ± 28.67	-124.67 ± 59.36
	2	-91.92 ± 24.71	-91.68 ± 20.04	-150.97 ± 50.99
	1	-93.36 ± 28.32	-94.68 ± 21.41	-167.53 ± 80.05

Tabelle 6.14.: DeepRED Leistung - Die evaluierten Regeln des DeepRED Modells wurden auf den DNNs in Tabelle 6.13 aufgebaut. Die Leistungen bestimmen die durchschnittliche kumulierte Belohnung von zehn Durchgängen zu je 100 evaluierten Episoden, unter den gleichen Voraussetzungen. Die Voraussetzungen sind durch das identische DRL Modell und DNN in den verschiedenen Tiefen gegeben. Die angegebene Tiefe bezeichnet die genutzte Tiefe der DTs in den einzelnen Schichten. Die zusammengefassten Regeln können diese Tiefe übersteigen.

6.3.1. MountainCar-v0

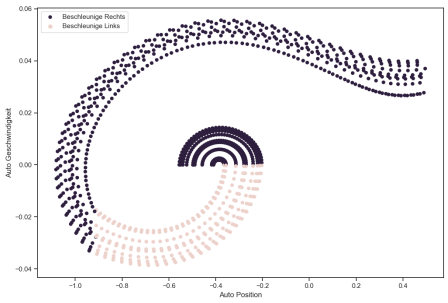
Im MC Problem können wir keine Regeln finden, mit denen wir das Problem als gelöst bezeichnen können. Ursächlich hierfür ist die partielle Funktionsweise des DeepRED Modells in diesem Problem. In den meisten Episoden erbringen die Regeln keinen Fortschritt in der Umgebung. Zudem existieren Episoden, in denen eine Strategie zur Lösung ersichtlich ist und moderate Ergebnisse erzielt werden. Dabei konnte der Fidelity Wert in jeder einzelnen Schicht diese Beobachtungen bestätigen. Daher können wir davon ausgehen, dass die Regeln korrekt über die Schichten zusammengeführt werden. Für Episoden, in welchen die schlechtmöglichste Belohnung erreicht wurde, lag der Wert häufig bei $\approx 30\%$ und in Episoden mit einer moderaten Leistung bei $\approx 90\%$. Weitere Fehlerquellen können wir über die Voraussetzung des gleichen DNNs mit einer hohen Genauigkeit und des von (Buitinck u. a., 2013) zur Verfügung gestellten C4.5 Algorithmus ausschließen. Die Genauigkeiten beschreiben wir in Tabelle 6.15. Dabei

stellen wir sowohl die Genauigkeiten des DNNs auf den Interaktionsdaten des DRL Modells vor, als auch die Episoden eines moderaten und schlechten Durchlaufs des DeepRED Modells.

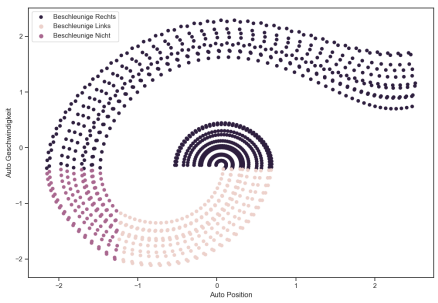
Modell	DNN	DeepRED BC	DeepRED WC
DQN	99.74%	91.82%	28.75%
PPO	98.80%	91.05%	29.78%
A2C	99.67%	50.67%	24.19%

Tabelle 6.15.: MountainCar-v0 DeepRED Genauigkeit mit (BC) Best Case und (WC) Worst Case - Angegeben werden die Genauigkeiten bezüglich der Interaktionsdaten des DRL Modells. Für die Genauigkeiten im DeepRED Modell wurde initial ein DT mit Tiefe drei gewählt. Alle Genauigkeiten wurden über 100 aufeinander folgende Episoden des jeweiligen DNNs erstellt.

Mit Tabelle 6.15 stellen wir fest, dass die identifizierten Regeln für das DQN und PPO Modell ähnliche Genauigkeiten liefern. Wegen der Ungenauigkeiten im A2C Modell, werden keine Fortschritte in der Umgebung erzielt. Ähnlich wie der Fidelity Wert, bestärkt die Genauigkeit unsere Aussagen bezüglich der Inkonsistenz in den Regeln des DQN und PPO Modells. Einen genauen Grund für das Auftreten dieses Verhaltens ist für uns zum diesem Zeitpunkt nicht nachvollziehbar. Anhand der Funktionalität und Leistungen der identifizierten Regeln in den anderen Problemen, siehe Tabelle 6.14, zeigt sich, dass die MC Umgebung ein schwer zu lösendes Problem für das DeepRED Modell darstellt. Auf eine Interpretation der gewonnenen Regeln können wir an dieser Stelle verzichten, da keine neuen Erkenntnisse dadurch gesammelt werden und diese weniger leistungsstark als die bereits vorgestellten sind. Trotzdem wollen wir diejenigen Episoden, in denen eine Strategie ersichtlich war, in der folgenden Abbildung darstellen.



(a) MountainCar-v0 DQN DeepRED nach Tabelle 6.15



(b) MountainCar-v0 PPO DeepRED nach Tabelle 6.15

Abbildung 6.19.: MountainCar-v0 DQN/PPO DeepRED Interaktionsdaten - Es wurden zehn aufeinander folgende Episoden evaluiert.

Durch Abbildung 6.19 stellen wir fest, dass sich durch Episoden, in denen der beschriebene BC aus Tabelle 6.15 eintritt, für uns zwei bekannte Strategien ergeben. In beiden beschleunigt der Wagen zunächst nach rechts und baut Schwung über den linken Berg auf. Der Unterschied ist wie auch zuvor durch die „Beschleunige Nicht“ Aktion gegeben, siehe Sektion 6.2.1. Anschließend besagen die Regeln, dass für den Rest der Episode die „Beschleunigung Rechts“ Aktion gewählt wird. In (a) werden -116 ± 2.37 kumulierte Belohnungen und in (b) -124.01 ± 1.24 erreicht.

6.3.2. CartPole-v1

Das DeepRED Modell konnte im CP Problem vergleichbare Leistungen zu den DTs aus der vorherigen Sektion hervorbringen. Dabei sind die extrahierten Regeln und die daraus resultierenden Strategien aus den jeweiligen DRL Modellen fast identisch. Einen Überblick über die erreichten Genauigkeiten und die Vertrauenswürdigkeit in den verschiedenen Schichten liefert die nachstehende Tabelle.

Modell	**DNN**	**DeepRED**	**Fidelity**$_2$	**Fidelity**$_1$	**Fidelity**$_0$
DQN	98.76%	90.99%	99.86%	98.70%	90.63%
PPO	98.16%	90.86%	99.99%	99.22%	90.37%
A2C	98.86%	90.16%	99.60%	98.96%	90.35%

Tabelle 6.16.: CartPole-v1 DeepRED Genauigkeit/Fidelity - Die Indizes an den Fidelity Werten geben an, zu welcher Schicht diese gehören. Alle Angaben wurden über 100 aufeinander folgenden Episoden des jeweiligen DNNs mit initialen DTs mit Tiefe drei erstellt.

Mit Tabelle 6.16 können wir den erfolgreichen Lernprozess des DeepRED Modells im CP Problem bestätigen. Ein großer Nachteil liegt in der Komplexität der generierten Regeln. Daher ist die Interpretation dieser nicht in einer nicht für den Menschen verständlicher Weise darzustellen. In Abbildung 6.20 stellen wir die generierten Regeln für das DQN Modell als DAG vor, mit welchem die Genauigkeiten aus Tabelle 6.16 und kumulierten Belohnungen aus Tabelle 6.14 erzeugt wurden.

Aus Abbildung 6.20 können wir keine interpretierbaren Regeln extrahieren, welche unsere Erkenntnisse erweitern. Weiterhin gehen aus unseren Experimenten keine DAGs hervor, welche die Größe des hier dargestellten DAGs für die initiale Tiefe von drei unterbieten. Eine Reduktion der Komplexität ist lediglich durch die Verwendung einer anfänglich geringeren Tiefe gegeben. Daraus ergeben sich jedoch Regeln, welche das vorliegende Problem nicht lösen, siehe Tabelle 6.14. Aufgrund der unzureichenden Leistung in geringen Tiefen und der Komplexität in höheren Tiefen, verzichten wir auf die Interpretation dieser Regeln. Im abgebildeten DAG werden zudem alle Observationen genutzt, sodass wir keine qualitativen Abbildungen bezüglich des Entscheidungsprozesses liefern können.

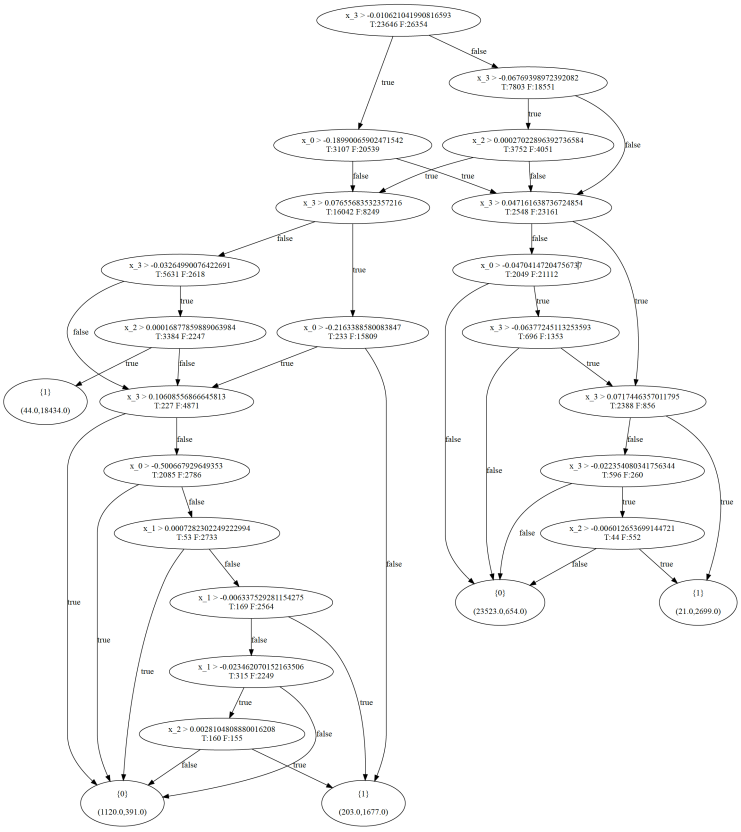


Abbildung 6.20.: CartPole-v1 DeepRED DQN DAG - Alle zur Verfügung stehenden Observationen werden genutzt. Die identifizierten Regeln folgen einer Evaluierung über 100 aufeinander folgenden Episoden des DNNs des DQN Modells. Dabei wurde ein initialer DT mit Tiefe drei genutzt.

6.3.3. Acrobot-v1

Für das AB Problem ist das DeepRED Modell in der Lage, Regeln zu finden, welche zu einem Fortschritt führen und selbst in anfänglich geringen Tiefen das vorliegende Problem lösen. Anhand Tabelle 6.14 können wir sehen, dass das PPO und DQN Modell fast identische kumulierte Belohnungen liefern. Im Gegensatz zum A2C Modell, das keine zielorientierte Strategie erlernt und daraus resultierend eine hohe Standardabweichung aufweist. Daher betrachten wir die identifizierten Regeln für das DQN und PPO Modell in geringen Tiefen.

Durch eine initiale Tiefe von eins, erhalten wir folgende Regel: `IF` $v_1 \leq 0.1$ `THEN` 2 `ELSE` 0 (DQN) und `IF` $v_1 \leq 0.01$ `THEN` 2 `ELSE` 0 (PPO). Diese sind den Regeln des evaluierten DTs aus Abbildung 6.14 sehr ähnlich und ermitteln grundsätzlich, ob die Geschwindigkeit des oberen Gelenks positiv oder negativ ist. Bei einer positiven Geschwindigkeit wird nach links und bei einer negativen nach rechts gedreht.

Mit einer weiteren initialen Tiefe wird zusätzlich die Geschwindigkeit am unteren Gelenk betrachtet. Dadurch werden die Regeln wie folgt formuliert: `IF` $v_1 \leq 0.1$ `THEN (IF` $v_2 \leq -0.39$ `THEN` 0 `ELSE` 2`) ELSE (IF` $v_2 \leq 0.71$ `THEN` 0 `ELSE` 2`)` (DQN). Die auf dem PPO Modell basierenden Regeln können wir an dieser Stelle bereits vernachlässigen, da diese zu komplex sind, um weiter interpretiert zu werden. Dies gilt auch für alle größeren initialen Tiefen. Letztlich werden die vorgestellten Regeln auf den gleichen Observationen aufgeteilt, wie in Abbildung 6.15. Diese unterscheiden sich lediglich durch die Werte, auf welchen gesplittet wird.

Modell	**DNN**	**DeepRED**	**Fidelity$_2$**	**Fidelity$_1$**	**Fidelity$_0$**
DQN	98.93%	86.41%	90.48%	89.70%	86.89%
PPO	99.84%	90.86%	99.96%	99.42%	93.86%
A2C	99.78%	94.28%	99.54%	99.42%	94.45%

Tabelle 6.17.: Acrobot-v1 DeepRED Genauigkeit/Fidelity - Die Indizes an den Fidelity Werten geben an, zu welcher Schicht diese gehören. Alle Angaben wurden über 100 aufeinander folgenden Episoden des jeweiligen DNNs mit initialen DTs mit Tiefe zwei erstellt.

Bislang war der Fidelity Wert und die Genauigkeit aufgrund der Vorhersage des zu erwartenden Ergebnisses sehr zuverlässig. Beide weisen hohe Werte bezüglich der Regeln des A2C Modells in Tabelle 6.17 auf, obwohl die Leistungen sehr stark schwanken und niedrigere kumulierte Belohnungen erzeugen, als die Regeln des DQN Modells. Daraus ergibt sich, dass wir hohe Genauigkeiten und Fidelity Werte nicht unbedingt mit ähnlich hohen kumulierten Belohnungen des jeweiligen DNNs in Verbindung setzten können. Angesichts der gewonnenen Regeln durch das DeepRED Modell, können wir keine neuen Erkenntnisse im AB Problem verzeichnen. Aufgrund der Komplexität kann die Interpretation der anfänglich in höheren Tiefen gelegenen DTs vernachlässigt werden. Die Regeln der ersten beiden Tiefen sind bereits ausführlich in der Sektion des DTs 6.2.4 beschrieben.

7. Zusammenfassung und Ausblick

In der vorliegenden Arbeit haben wir uns mit der Extraktion von Regeln aus Deep Reinforcement Learning Modellen in OpenAI-Gym Problemen beschäftigt. Zur Lösung der Probleme haben wir folgende Deep Reinforcement Learning Modelle genutzt: Deep Q-Network, Proximal Policy Optimization und Advantage Actor Critic. Auf der Grundlage des vorangegangenen Praxisprojekts und den darin optimierten Hyperparametern, war es uns möglich, performante Modelle für die gegebenen Probleme zu untersuchen. Durch das OpenAI-Gym wurden die Probleme bereitgestellt und sind namentlich als einige der klassischen Kontrollprobleme bekannt: MountainCar-v0, MountainCarContinuous-v0, CartPole-v1 und Acrobot-v1. Die Regelextraktion erfolgte durch zwei Modelle: Decision Tree und DeepRED. Beide Modelle sind transparent und behandeln Probleme des Supervised Learnings. Durch die generierten Interaktionsdaten des trainierten Deep Reinforcement Learnings Modells mit den Umgebungen, konnten wir beiden Ansätzen genügend Daten bereitstellen, um Regeln zu extrahieren. Aufgrund der Transparenz der gewählten Methoden waren wir in der Lage diese Regeln direkt zu interpretieren. Um die Ergebnisse der durchgeführten Experimente vorzustellen, orientieren wir uns an den formulierten Forschungsfragen aus Sektion 1.4 und betrachten die verwendeten Ansätze zur Regelextraktion im Folgenden separat.

1. *Ist es möglich, die Entscheidungen eines komplexen DRL Modells durch einen DT darzustellen?*

   Im Rahmen dieser Arbeit war es uns möglich, die Entscheidungen eines Deep Reinforcement Learning Modells durch einen Decision Tree darzustellen. Für die Erzeugung eines Decision Trees, haben wir die Interaktionsdaten aus 100 Episoden des trainierten Deep Reinforcement Learning Modells mit und für die jeweilige Umgebung verwendet. (a) Dabei konnte jedes Problem durch die extrahierten Regeln in geringen Tiefen gelöst werden. (b) Das MountainCar-v0 und CartPole-v1 Problem galten ab einer Tiefe von drei als gelöst. Im CartPole-v1 Problem formulierten die gewonnenen Regeln eine optimale Lösung. Für die MountainCarContinuous-v0 Umgebung wurden Regeln zur Lösung mit einer Tiefe von zwei identifiziert. Das Acrobot-v1 Problem besitzt keinen Schwellenwert, ab welchem dieses als gelöst bezeichnet wird. Eine Lösung für das eigentliche Problem wurde bei einer Tiefe von eins gefunden. (c) Die hervorgebrachten Regeln lieferten für den Menschen interpretierbare Zusammenhänge, sodass nicht fachkundige Leser ein Verständnis dafür entwickeln können. Bis auf das Acrobot-v1 Problem, konnten wir aus den Erkenntnissen der Regeln für jede Umgebung eine optimale Strategie formulieren. (d) Im Vergleich zu bisherigen Forschungsarbeiten, waren die identifizierten Regeln konkurrenzfähig.

2. *Erlaubt uns das DeepRED Modell, Regeln aus einem separat trainierten DNN zu extrahieren, das auf den Interaktionsdaten des DRL Modell beruht?*

   Mit dem DeepRED Modell war es uns möglich, Regeln aus einem separat trainierten Deep Neural Network zu extrahieren. Die dafür genutzten Deep Neural Networks wurden basierend auf den in 100 Episoden erzeugten Interaktionsdaten des trainierten Deep Reinforcement Learning Modells erstellt. Das DeepRED Modell nutzt den C4.5 Algorithmus und kann Reinforcement Learning Probleme mit einem diskreten Aktionsraum behandeln. Aus diesem Aspekt konnte das MountainCarContinuous-v0 Problem nicht betrachtet werden. (a) Als gelöst konnten wir das CartPole-v1 und das Acrobot-v1 Problem bezeichnen. Zudem beschrieben die Regeln eine optimale Lösung für das CartPole-v1 Problem. Aufgrund unzureichenden kumulierten Belohnungen und einer partiellen Funktionsweise in der MountainCar-v0 Umgebung, gilt dies als nicht gelöst. (b) Mit einer initialen Tiefe von drei beschreiben wir die CartPole-v1 Umgebung als gelöst und ab vier als optimal. Für das AB Problem konnten wir das eigentliche Problem mit einer Tiefe von eins lösen. (c) Die gewonnen Regeln waren bezüglich der Interpretation unpraktisch. Selbst in geringen initialen Tiefen ist bei der Zusammenführung der Regeln ein weitaus größerer Decision Tree entstanden, als die anfängliche Tiefe angegeben hat. Die entstandenen Decision Trees waren nicht nachvollziehbar und daher auch nicht für den fachkundigen Leser verständlich. Alleinig für das Acrobot-v1 Problem konnten interpretierbare Regeln für initiale Tiefen bis zu zwei gefunden werden. (d) Unter Berücksichtigung der letztendlich verwendeten Tiefe, stellten die Regeln keine qualitativen Ergebnisse im Vergleich zu bisherigen Forschungsarbeiten dar.

Zusammenfassend können wir sagen, dass wir in dieser Arbeit interpretierbare Regeln aus Deep Reinforcement Learning Modellen extrahieren konnten, welche die zugrunde liegenden Umgebungen gelöst haben. Dabei geben wir dem Decision Tree Ansatz einen Vorteil. Dieser konnte in den von uns untersuchten Bereichen, sprich Leistung, Interpretierbarkeit und Replizierbarkeit, deutlich besser abschneiden als das DeepRED Modell. Die Wissenschaft profitiert nicht nur durch das Hervorbringen innovativer Ansätze, sondern auch durch die Falsifikation bereits bekannter Methoden. In unserem Fall konnten wir empirisch zeigen, dass die Anwendbarkeit des DeepRED Modells auf Reinforcement Learning Problemen möglich ist, jedoch nur in sehr wenigen Fällen Erkenntnisse lieferte, in denen das Blackbox Modell für den Menschen verständlich und interpretierbar dargestellt wurde.

Trotz der leistungsstarken und interpretierbaren Regeln des Decision Trees bleibt noch Platz für Verbesserungen und neue Ansätze. In dieser Arbeit haben wir drei verschiedene Deep Reinforcement Learning Modelle eingesetzt. In Engelhardt u. a. (2021) wurde im MountainCarContinuous-v0 Problem das Twin-Delayed Deep Deterministic Policy Gradient Modell genutzt, welches die von uns untersuchten Modelle in der Leistung und Robustheit übertraf. Daher wäre die Verwendung verschiedener Modelle eine möglich Maßnahme zur Steigerung der Leistung und Interpretierbarkeit. Weiterhin stellen Deep Neural Networks einen wesentlichen Bestandteil zur Formulierung

einer Entscheidung im Deep Reinforcement Learning dar. Kurz vor Drucklegung der Arbeit haben sich neue Erkenntnisse bezüglich des Programmpakets DeepRED ergeben, welche in unserem *Repository* hinterlegt sind. Daher wurden die Experimente mit dem DeepRED Modell aus zeitlichen Gründen nicht in der Tiefe besprochen, wie die Experimente der Decision Trees. Daraus ergibt sich noch weiterer Forschungsbedarf mit dem DeepRED Modell. Die verschiedenen Deep Neural Network Architekturen, der Versuch, Regeln aus dem Deep Reinforcement Learning Modell internen Netz zu extrahieren oder der Austausch des C4.5 Algorithmus durch den erfolgreich angewendeten CART Algorithmus, bilden weitere spannende zur erforschende Bereiche. Ein weiterer vielversprechender Ansatz, welcher Grundlage für neue Forschungsarbeiten darstellt, ist durch Dancey u. a. (2004) gegeben.

Abbildungsverzeichnis

2.1. Supervised Learning 8
2.2. Agent- Umgebungsinteraktion 8
2.3. Unsupervised Learning 9

3.1. Taxonomie RL Algorithmen 16
3.2. Deep Neural Network 19
3.3. Aktivierungsfunktionen 20

4.1. Decision Tree Darstellungen 23

5.1. Reinforcement Learning Probleme 26

6.1. XOR Problem Ergebnisse 31
6.2. MountainCar-v0 DQN DT 33
6.3. MountainCar-v0 PPO DT 33
6.4. MountainCar-v0 DT Interaktionsdaten 34
6.5. MountainCar-v0 Orakel/DT Vergleich 35
6.6. MountainCarContinuous-v0 DT univariat 38
6.7. MountainCarContinuous-v0 DT bivariat 40
6.8. MountainCarContinuous-v0 PPO DT 41
6.9. MountainCarContinuous-v0 A2C DT 42
6.10. MountainCarContinuous-v0 Orakel/DT Vergleich 42
6.11. CartPole-v1 DQN DT Tiefe 2 45
6.12. CartPole-v1 DQN DT Tiefe 3 46
6.13. CartPole-v1 DT/Vergleich 48
6.14. Acrobot-v1 DQN DT Tiefe 1 50
6.15. Acrobot-v1 DQN DT Tiefe 2 51
6.16. Acrobot-v1 DQN DT Tiefe 3 51
6.17. Acrobot-v1 DQN DT CM 52
6.18. Acrobot-v1 Orakel/DT Vergleich 53
6.19. MountainCar-v0 DQN/PPO DeepRED 57
6.20. CartPole-v1 DeepRED DQN DAG 59

Tabellenverzeichnis

5.1. Reinforcement Learning Probleme Observationen 27
5.2. Reinforcement Learning Probleme Aktionen 28
5.3. Deep Reinforcement Learning Modelle Leistung 29

6.1. XOR Problem Wahrheitstafel 30
6.2. XOR Problem Ergebnisse 31
6.3. MountainCar-v0 DT Leistung 32
6.4. MountainCar-v0 Regeln Vergleich 36
6.5. MountainCarContinuous-v0 DT Leistung 37
6.6. MountainCarContinuous-v0 R^2-Wert 39
6.7. MountainCarContinuous-v0 Regeln Vergleich 43
6.8. CartPole-v1 DT Leistung 44
6.9. CartPole-v1 DQN DT FI 47
6.10. CartPole-v1 Regeln Vergleich 49
6.11. Acrobot-v1 DT Leistung 49
6.12. Acrobot-v1 DQN DT FI 50
6.13. Deep Neural Network Leistung 55
6.14. DeepRED Leistung 56
6.15. MountainCar-v0 DeepRED Genauigkeit 57
6.16. CartPole-v1 DeepRED Genauigkeit/Fidelity 58
6.17. Acrobot-v1 DeepRED Genauigkeit/Fidelity 60

A.1. Hyperparameter DQN 68
A.2. Hyperparameter A2C 68
A.3. Hyperparameter PPO 69
A.4. Hyperparameter CartPole-v1 DNN 69
A.5. Hyperparameter MountainCar-v0 DNN 70
A.6. Hyperparameter MountainCarContinuous-v0 DNN 70
A.7. Hyperparameter Acrobot-v1 DNN 70

A. Anhang

Algorithmen

Algorithmus 1 DQN mit Experience Replay nach (Mnih u. a., 2013)

Voraussetzung: Initialisiere Experience Replay $\mathcal{D}$ als leeren Speicher mit Kapazität $N \in \mathbb{N}$, Aktionswertfunktion $\hat{q}$ mit zufälligen Gewichten, Schritte zur Aktualisierung der Zielwerte C und setze den Faktor $\epsilon \in \mathbb{R}$ der ϵ-Greedy Strategie.

for $e \leftarrow 1$ **to** M **do** ▷ $M \in \mathbb{N}$, Anzahl der Episoden
  Initialisiere Sequenz $S_1 = \{x_1\}$ und vorverarbeitete Sequenz $\Phi_1 = \Phi(S_1)$
  for $t \leftarrow 1$ **to** $\mathcal{T}$ **do**
    Wähle Aktion A_t mit Wahrscheinlichkeit ϵ
    Führe Aktion A_t aus und beobachte x_{t+1} und Belohnung R_t
    Setze $S_{t+1} \leftarrow \{S_t, A_t, x_{t+1}\}$ und vorverarbeite $\Phi_{t+1} = \Phi(S_{t+1})$
    Setze $\mathcal{D}_t \leftarrow (\Phi_t, A_t, R_t, \Phi_{t+1})$
    Wähle zufällige Mini-Stapel $(\Phi_j, A_j, R_j, \Phi_{j+1}) \in \mathcal{D}$
    $y_j = \begin{cases} R_j & \text{für terminierenden Zustand } \Phi_{j+1} \\ R_j + \gamma \max_{a' \in \mathcal{A}} \hat{q}(\Phi_{j+1}, a', \psi_{i-1}) & \text{für nicht-terminierenden Zustand } \Phi_{j+1} \end{cases}$
    Führe Gradienten-Aktualisierung bezüglich $(y_j - \hat{q}(\Phi_j, A_j, \psi_i))^2$ aus
    if $t \bmod C = 0$ **then**
      $\hat{q}(s', a', \psi_{i-1}) \leftarrow \hat{q}(s', a', \psi_i)$ ▷ Aktualisiere Zielwerte
    end if
  end for
end for

Algorithmus 2 PPO-Clip nach (Achiam, 2018) und (Schulman u. a., 2017)

Voraussetzung: Initialisiere Parameter der Strategie $\theta_0 \in \mathbb{R}^m$ und Parameter der Zustandswertfunktion $\psi_0 \in \mathbb{R}^{m'}$

for $k \leftarrow 1$ **to** M **do** ▷ $M \in \mathbb{N}$, Anzahl der Episoden
  Sammele Trajektorien $\mathcal{D}_k = \{\tau_i\}$ unter Verwendung von $\pi_k = \pi(\theta_k)$
  $\hat{R}_t \leftarrow \sum_{t'=t}^{\mathcal{T}} R(S_{t'}, A_{t'}, S_{t'+1})$ Berechne Belohnungen
  $\hat{A}_t \leftarrow \hat{q}(S_t, A_t, \psi_k) - \hat{v}(S_t, A_t, \psi_k)$ Berechne Vorteilsfunktion
  Aktualisiere Strategie durch Maximierung des Clips mittels stochastischem Gradienten:

$$\theta_{k+1} \leftarrow \underset{\theta \in \mathbb{R}^m}{\operatorname{argmax}} \frac{1}{|\mathcal{D}_k|\mathcal{T}} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{\mathcal{T}} \min\left(\frac{\pi_\theta(A_t|S_t)}{\pi_{\theta_k}(A_t|S_t)} \hat{A}_{\pi_{\theta_k}}(S_t, A_t), g(\epsilon, \hat{A}_{\pi_{\theta_k}}(S_t, A_t)) \right)$$

  Passe Zustandswertfunktion mit der Methode der kleinste Quadrate an:

$$\psi_{k+1} \leftarrow \underset{\psi \in \mathbb{R}^{m'}}{\operatorname{argmin}} \frac{1}{|\mathcal{D}|\mathcal{T}} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{\mathcal{T}} \left(\hat{v}(S_t, \psi) - \hat{R}_t \right)^2,$$

  mit einem Gradientenverfahren.
end for

Algorithmus 3 A2C nach (Wang u. a., 2018), (Hao u. a., 2020, S.168) und (Mnih u. a., 2016)

Voraussetzung: Initialisiere Parameter der Strategie $\theta = \theta_0 \in \mathbb{R}^m$, Parameter der Zustandswertfunktion $\psi = \psi_0 \in \mathbb{R}^{m'}$, Schrittweite η_θ, $\eta_\psi \in (0, 1]$ und eine Menge der Agenten mit $\mathcal{W} = \{w_1, w_2, \ldots, w_n\}$

Haupt-Akteur-Kritiker

for $k \leftarrow 1$ **to** M **do** ▷ $M \in \mathbb{N}$, Anzahl der Episoden
  Setze Gradienten zurück $(g_\psi, g_\theta) \leftarrow 0$
  for all $w \in \mathcal{W}$ **do**
    $(g_\psi, g_\theta) = (g_\psi, g_\theta) + w(\hat{v}_{\pi_\theta}, \pi_\theta)$ ▷ Erhalte Aktualisierung von Agenten
  end for
  Aktualisiere Parameter $\psi = \psi + \Delta_\psi; \theta = \theta + \Delta_\theta$
end for

Agent

for $i \leftarrow 1$ **to** L **do** ▷ $L \in \mathbb{N}$, Länge Trajektorie
  Sammele $\{S_t, A_t, R_t, S_{t+1}\}$ Unter Verwendung von π_θ
end for
Approximiere Vorteilsfunktion $\hat{A}_t \leftarrow R_t + \gamma \hat{v}_{\pi_\theta}(S_{t+1}, \psi) - \hat{v}_{\pi_\theta}(S_t, \psi)$
Parametrisiere Strategie $\mathcal{J}(\theta) \leftarrow \sum_t \log(\pi_\theta(A_t|S_t))\hat{A}_t$
Parametrisiere Zustandswertfunktion $\mathcal{J}(\psi) \leftarrow \sum_t \hat{A}_t^2$
return $(\nabla_\psi \mathcal{J}(\psi), \nabla_\theta \mathcal{J}(\theta))$ ▷ Sende Aktualisierung an Haupt-Akteur-Kritiker

Algorithmus 4 DeepRED nach (Zilke u. a., 2016)

Voraussetzung: DNN mit Schichten $h_0, h_1, \ldots, h_k, h_{k+1}$, mit $k \in \mathbb{N}$ und Trainingsdaten $x \in \mathbb{R}^m$

for all $\{\lambda_v \in \lambda_1, \ldots, \lambda_u\}$ **do**
  Setze $R^v_{h_k \to h_{k+1}} \leftarrow$ `IF` $h_{k+1,v} > 0.5$ `THEN` $\hat{y} = \lambda_v$ ▷ Initiale Regel zur Prädiktion
  for all $h_j \in \{h_k, h_{k-1}, \ldots, h_1\}$ **do**
    Setze $R^v_{h_{j-1} \to h_j} \leftarrow \emptyset$
    Setze $T \leftarrow$ `extractTermsFromRuleBodies`$(R^v_{h_j \to h_{j+1}})$ ▷ Extrahiere Regeln aus Schicht $j + 1$
    Entferne doppelte Regeln $T \leftarrow$ `removeDuplicateTerms`(T)
    for all $t \in T$ **do**
      $x_1^{'}, \ldots, x_m^{'} \leftarrow h_j(x_1), \ldots, h_j(x_m)$ ▷ Setze Werte der Aktivierung für alle Neuronen der $j-1$-ten Schicht
      $y_1^{'}, \ldots, y_m^{'} \leftarrow t(h_{j+1}(x_1)), \ldots, t(h_{j+1}(x_m))$ ▷ Wende t auf Aktivierung in der j-ten Schicht an
      $R^v_{h_{j-1} \to h_j} \leftarrow R^v_{h_{j-1} \to h_j} \cup$ `C4.5`$((x_1^{'}, y_1^{'}), \ldots, (x_m^{'}, y_m^{'}))$ ▷ Lerne und füge Regeln hinzu für t
    end for
  end for
  for all $h_j \in \{h_k, h_{k-1}, \ldots, h_1\}$ **do**
    $R^v_{h_{j-1} \to o} \leftarrow$ `mergeIntermediateTerms`$(R^v_{h_{j-1} \to h_j}, R^v_{h_j \to o})$ ▷ Vereinige die Regeln der Schichten
    $R^v_{h_{j-1} \to o} \leftarrow$ `deleteUnsatisfiableTerms`$(R^v_{h_{j-1} \to o})$
    $R^v_{h_{j-1} \to o} \leftarrow$ `deleteRedundantTerms`$(R^v_{h_{j-1} \to o})$
  end for
end for
return $R^1_{i \to 0}, R^2_{i \to 0}, \ldots, R^u_{i \to 0}$ ▷ Menge an Regeln, die das DNN beschreiben

Hyperparameter

Hyperparameter	**Umgebung**		
	MC	CP	AB
policy	MlpPolicy	MlpPolicy	MlpPolicy
timesteps	1.2×10^5	5×10^4	1×10^5
net_arch	$[256, 256]$	$[256, 256]$	$[256, 256]$
learning_rate	4×10^{-3}	1×10^{-3}	6.3×10^{-4}
batch_size	128	64	128
buffer_size	2×10^4	1×10^5	5×10^4
learning_starts	3×10^3	500	0
target_update_interval	600	10	250
train_freq	16	256	4
gradient_steps	8	50	-1
gamma	0.98	50	0.99
exploration_fraction	0.2	0.16	0.12
exploration_final_eps	0.0	0.04	0.1

Tabelle A.1.: Hyperparameter DQN

Hyperparameter	**Umgebung**			
	MC	MCC	CP	AB
policy	MlpPolicy	MlpPolicy	MlpPolicy	MlpPolicy
timesteps	1×10^6	1.5×10^5	5×10^5	5×10^5
net_arch	$[64, 64]$	$[64, 64]$	$[128, 128]$,$[128, 128]$	$[128, 128]$,$[128, 128]$
ortho_init	Wahr	Wahr	Wahr	Falsch
activation_fn	$\tanh(x)$	$\tanh(x)$	$ReLu(x)$	$\tanh(x)$
n_envs	16	4	8	16
learning_rate	$\text{lin}(9 \times 10^{-3})$	7×10^{-4}	$\text{lin}(3 \times 10^{-3})$	7×10^{-4}
n_steps	32	3	64	16
ent_coef	1×10^{-7}	0.0	1×10^{-6}	0.0
vf_coef	0.12	0.5	0.15	0.5
use_rms_prop	Wahr	Wahr	Wahr	Wahr
use_sde	Falsch	Wahr	Falsch	Falsch
normalize_advantage	Wahr	Falsch	Wahr	Falsch
max_grad_norm	0.6	0.5	0.6	2
gamma	0.99	0.99	0.99	0.99
gae_gamma	0.8	1.0	0.98	1.0

Tabelle A.2.: Hyperparameter A2C

Hyperparameter	**Umgebung**			
	MC	MCC	CP	AB
policy	MlpPolicy	MlpPolicy	MlpPolicy	MlpPolicy
timesteps	5×10^5	5×10^4	1×10^5	5×10^5
net_arch	[64, 64],[64, 64]	[64, 64],[64, 64]	-	[64, 64],[64, 64]
ortho_init	Wahr	Falsch	Wahr	Falsch
activation_fn	$\tanh(x)$	$\tanh(x)$	$\tanh(x)$	$\tanh(x)$
n_envs	8	4	8	16
normalized	Wahr	Wahr	Falsch	Wahr
learning_rate	1×10^{-3}	3×10^{-4}	1×10^{-3}	1.5×10^{-5}
ent_coef	1×10^{-3}	0.0	0.0	1.7×10^{-5}
vf_coef	0.2	0.5	0.5	0.34
clip_range	0.1	0.2	0.2	0.3
n_epochs	20	10	20	20
n_steps	2048	64	32	8
batch_size	64	32	256	512
max_grad_norm	0.9	0.5	0.5	1
gamma	0.99	0.99	0.98	0.99
gae_gamma	0.97	0.95	0.8	0.98
use_sde	Falsch	Wahr	Falsch	Falsch

Tabelle A.3.: Hyperparameter PPO

Hyperparameter	**DRL Modell**		
	DQN	PPO	A2C
net_arch	[64, 64]	[64, 64]	[64, 64]
activation_fn	Sigmoid	Sigmoid	Sigmoid
init	Normal	Normal	Normal
bias	Wahr	Wahr	Wahr
loss	Kategorisch	Kategorisch	Kategorisch
optimizer	RMSprop	RMSprop	RMSprop
learning_rate	0.01	0.01	0.01
batch_size	64	64	64
n_epochs	10	10	10

Tabelle A.4.: Hyperparameter CartPole-v1 DNN

Hyperparameter	**DRL Modell**		
	DQN	PPO	A2C
net_arch	[256, 256]	[256, 256]	[256, 256]
activation_fn	Sigmoid	Sigmoid	Sigmoid
init	Normal	Normal	Normal
bias	Wahr	Wahr	Wahr
loss	Kategorisch	Kategorisch	Kategorisch
optimizer	RMSprop	RMSprop	RMSprop
learning_rate	0.01	0.01	0.01
batch_size	256	256	256
n_epochs	150	150	150

Tabelle A.5.: Hyperparameter MountainCar-v0 DNN

Hyperparameter	**DRL Modell**	
	PPO	A2C
net_arch	[512, 512, 512]	[512, 512, 512]
activation_fn	Selu	Selu
init	Normal	Normal
bias	Wahr	Wahr
loss	Kategorisch	Kategorisch
optimizer	MAE	MAE
learning_rate	1×10^{-3}	1×10^{-3}
batch_size	128	128
n_epochs	100	100

Tabelle A.6.: Hyperparameter MountainCarContinuous-v0 DNN

Hyperparameter	**DRL Modell**		
	DQN	PPO	A2C
net_arch	[256, 256]	[256, 256]	[256, 256]
activation_fn	Sigmoid	Sigmoid	Sigmoid
init	Random-Uniform	Random-Uniform	Random-Uniform
bias	Wahr	Wahr	Wahr
loss	Kategorisch	Kategorisch	Kategorisch
optimizer	RMSprop	RMSprop	RMSprop
learning_rate	0.01	0.01	0.01
batch_size	256	256	256
n_epochs	150	150	150

Tabelle A.7.: Hyperparameter Acrobot-v1 DNN

Regeln

Algorithmus 5 XOR Problem Regeln für Abbildung 6.1b - Regeln des DT sind zu komplex, um dargestellt zu werden. Bei Interesse sind diese durch das gegebene Repository reproduzierbar.

Voraussetzung: Beobachtungen: A und B; Ausgabe: Wahr oder Falsch

DeepRED

```
1: if A > 0.5 then
2:     if B > 0.5 then
3:         return Falsch
4:     else
5:         return Wahr
6:     end if
7: else
8:     if A > 0.5 then
9:         return Falsch
10:    else
11:        return Wahr
12:    end if
13: end if
```

Algorithmus 6 MountainCar-v0 Problem Regeln für DQN DT_3 aus Tabelle 6.4

Voraussetzung: Beobachtungen: PA und GA; Ausgabe: 0, 1, oder 2

DQN DT_3

```
1: if GA ≤ 0.0 then
2:     if PA ≤ −0.91 then
3:         if PA ≤ −0.929 then
4:             return 2
5:         else
6:             return 2
7:         end if
8:     else
9:         if GA ≤ −0.0 then
10:            return 0
11:        else
12:            return 2
13:        end if
14:    end if
15: else
16:    if GA ≤ 0.014 then
17:        if PA ≤ −0.382 then
18:            return 2
19:        else
20:            return 0
21:        end if
22:    else
23:        if GA ≤ 0.018 then
24:            return 2
25:        else
26:            return 2
27:        end if
28:    end if
29: end if
```

Algorithmus 7 MountainCar-v0 Problem Regeln für PPO DT_3 aus Tabelle 6.4

Voraussetzung: Beobachtungen: PA und GA; Ausgabe: 0, 1, oder 2

PPO DT_3

if $GA \leq -0.474$ **then**
  if $PA \leq -1.247$ **then**
    if $GA \leq -0.823$ **then**
      return 1
    else
      return 2
    end if
  else
    if $PA \leq -1.211$ **then**
      return 0
    else
      return 0
    end if
  end if
else
  if $GA \leq 0.192$ **then**
    if $PA \leq -0.047$ **then**
      return 2
    else
      return 0
    end if
  else
    if $GA \leq 0.272$ **then**
      return 2
    else
      return 2
    end if
  end if
end if

Algorithmus 8 MountainCar-v0 Problem Verma Regeln von (Verma u. a., 2018) aus Tabelle 6.4

Voraussetzung: Beobachtungen: PA und GA; Ausgabe: 0, 1, oder 2; Die **peek**(X,Y) Funktion haben wir bereits in Sektion 6.2.4 vorgestellt. Dabei können wir die Ergebnisse in (Verma u. a., 2018) nicht bestätigen, da die Regeln weitaus niedrigere kumulierte Belohnungen zurückgeben, als im Paper angegeben. Die kumulierte Belohnung lag für die getesteten Regeln bei -164.37 ± 16.52. Im Paper sind diese als $-108.06, -143.86$ angegeben.

Verma

if $(0.2498 - \textbf{peek}(PA, -1) > 0)$ **and** $(0.0035 - \textbf{peek}(GA, -1) < 0)$ **then**
  return 0
else
  return 2
end if

Algorithmus 9 MountainCar-v0 Problem HC Regeln von (Engelhardt u. a., 2021) und (Xiao, 2019) aus Tabelle 6.4

Voraussetzung: Beobachtungen: PA und GA; Ausgabe: 0, 1, oder 2

Engelhardt HC

if $GA = 0$ **then**
  if $PA > -0.475$ **then**
    return 0
  else
    return 2
  end if
else
  if $GA > 0$ **then**
    return 2
  else
    return 0
  end if
end if

Xiao HC

if $GA \geq \min\left(-0.09(PA + 0.25)^2 + 0.03, 0.3(PA + 0.9)^4 - 0.008\right)$ **and** $GA \leq -0.07(PA + 0.38)^2 + 0.07$ **then**
  return 2
else
  return 0
end if

Algorithmus 10 MountainCarContinuous-v0 Problem Regeln für A2C DT_2 aus Tabelle 6.5

Voraussetzung: Beobachtungen: PA und GA; Ausgabe: Leistungs-Koeffizient $\in [-1, 1]$

A2C DT_2

if $GA \leq -0.308$ **then**
  if $PA \leq -0.817$ **then**
    return 0.436
  else
    return -0.938
  end if
else
  if $GA \leq 0.096$ **then**
    return 0.68
  else
    return 0.996
  end if
end if

Algorithmus 11 MountainCarContinuous-v0 Problem Regeln für PPO DT_3 aus Tabelle 6.5

Voraussetzung: Beobachtungen: PA und GA; Ausgabe: Leistungs-Koeffizient $\in [-1, 1]$

PPO DT_3

```
if GA ≤ −0.49 then
    if PA ≤ −0.859 then
        if GA ≤ −3.336 then
            return −0.665
        else
            return 0.852
        end if
    else
        if PA ≤ −0.827 then
            return −0.442
        else
            return −0.994
        end if
    end if
else
    if GA ≤ 0.097 then
        if GA ≤ 0.106 then
            return 1.0
        else
            return 0.979
        end if
    else
        if PA ≤ −0.499 then
            return 0.995
        else
            return −0.61
        end if
    end if
end if
```

Algorithmus 12 MountainCarContinuous-v0 Problem HC Regeln von (Xiao, 2019) aus Tabelle 6.7

Voraussetzung: Beobachtungen: PA und GA; Ausgabe: Leistungs-Koeffizient $\in [-1, 1]$

Xiao HC

```
if PA > 4 · GA or PA < 13 · GA − 0.6 then
    return 1.0
else
    return −1.0
end if
```

Algorithmus 13 CartPole-v1 Problem Regeln für DQN DT_2 aus Tabelle 6.8

Voraussetzung: Beobachtungen: WP, WG, SW und SWG; Ausgabe: 0 oder 1

DQN DT_2

```
if SWG ≤ −0.025 then
    if SW ≤ 0.002 then
        return 0
    else
        return 0
    end if
else
    if SWG ≤ 0.096 then
        return 1
    else
        return 1
    end if
end if
```

Algorithmus 14 CartPole-v1 Problem Regeln für DQN DT_3 aus Tabelle 6.8

Voraussetzung: Beobachtungen: WP, WG, SW und SWG; Ausgabe: 0 oder 1

DQN DT_3

```
if SWG ≤ −0.006 then
    if SWG ≤ −0.092 then
        if SW ≤ 0.036 then
            return 0
        else
            return 1
        end if
    else
        if SW ≤ 0.002 then
            return 0
        else
            return 1
        end if
    end if
else
    if SWG ≤ 0.088 then
        if SW ≤ −0.033 then
            return 0
        else
            return 1
        end if
    else
        if SW ≤ −0.001 then
            return 0
        else
            return 1
        end if
    end if
end if
```

Algorithmus 15 CartPole-v1 Problem HC Regeln von (Xiao, 2019) und (Xu, 2021) aus Tabelle 6.10

Voraussetzung: Beobachtungen: WP, WG, SW und SWG; Ausgabe: 0 oder 1

Xiao HC

if $3 \cdot SW + SWG > 0$ **then**
    return 1
else
    return 0
end if

Xu HC

if $|SW| < 0.03$ **then**
    if $SWG < 0$ **then**
        return 0
    else
        return 1
    end if
else
    if $SW < 0$ **then**
        return 0
    else
        return 1
    end if
end if

Algorithmus 16 Acrobot-v1 Problem Regeln für DQN DT_1 und DT_2 aus Tabelle 6.11

Voraussetzung: Beobachtungen: $\cos(\theta_1), \sin(\theta_1), \cos(\theta_2), \sin(\theta_2), v_1$ und v_2; Ausgabe: 0, 1 oder 2

DQN DT_1

if $v_1 \leq 0.002$ **then**
    return 2
else
    return 0
end if

DQN DT_2

if $v_1 \leq 0.023$ **then**
    if $v_2 \leq -1.18$ **then**
        return 0
    else
        return 2
    end if
else
    if $v_2 \leq 1.839$ **then**
        return 0
    else
        return 2
    end if
end if

Algorithmus 17 Acrobot-v1 Problem Regeln für DQN DT_3 aus Tabelle 6.11

Voraussetzung: Beobachtungen: $\cos(\theta_1), \sin(\theta_1), \cos(\theta_2), \sin(\theta_2), v_1$ und v_2; Ausgabe: 0, 1 oder 2

DQN DT_3

```
if v_1 ≤ 0.046 then
    if cos(θ_1) ≤ 0.169 then
        if v_2 ≤ 3.274 then
            return 2
        else
            return 2
        end if
    else
        if v_2 ≤ −2.389 then
            return 0
        else
            return 2
        end if
    end if
else
    if cos(θ_1) ≤ 0.052 then
        if v_2 ≤ −3.106 then
            return 0
        else
            return 2
        end if
    else
        if v_2 ≤ 1.162 then
            return 0
        else
            return 2
        end if
    end if
end if
```

Algorithmus 18 Acrobot-v1 Problem Verma Regeln von (Verma u. a., 2018) aus Sektion 6.2.4

Voraussetzung: Beobachtungen: $\cos(\theta_1), \sin(\theta_1), \cos(\theta_2), \sin(\theta_2), v_1$ und v_2; Ausgabe: 0, 1 oder 2; **peek**(X,Y) Funktion

Verma

```
if (0.1357 + peek(v_1, −1)) < 0 then
    return 2
else
    return 0
end if
```

Verwendete Hard- und Software

Die nachfolgenden Informationen geben Auskunft über die verwendete Soft- und Hardware. Für die Erstellung der Experimente und der Ausarbeitung wurde folgende Hardware verwendet:

Seriennummer	-	H12G9HB8PN7C
Modell	-	iMac (Retina 5K, 27", 2020)
Prozessor	-	3,8 GHz 8-Core Intel Core i7
Arbeitsspeicher	-	64 GB 2667 MHz DDR4
Grafikkarte	-	AMD Radeon Pro 5700 XT 16 GB
Betriebssystem	-	macOS Monterey: 12.1

Verwendete Software:

Entwicklungsumgebung	-	PyCharm 2021.2.1 (Professional Edition)
LaTeX-Editor	-	Overleaf

Die genutzte Programmiersprache in diesem Projekt ist Python. Dabei haben wir die folgende Version von Python genutzt.

Version - Python 3.9.6 (v3.9.6:db3ff76da1, Jun 28 2021, 11:49:53)

Installierte Python Pakete:

Paket	**Version**	**Erscheinungsdatum**
dtreeviz	1.3.2	10.11.2021
graphviz	0.18	18.11.2020
gym	0.20.0	14.09.2021
ipython	7.28.0	25.09.2021
jolib	1.1.0	07.10.2021
matplotlib	3.4.3	13.08.2021
numpy	1.19.5	05.01.2021
optuna	2.9.1	03.08.2021
pandas	1.3.3	12.09.2021
pydotplus	2.0.2	09.12.2014
pyglet	1.5.21	16.09.2021
ruleex	0.4.1	29.10.2019
scipy	1.7.2	19.11.2020
seabron	0.11.2	16.08.2021
sklearn	1.0.2	25.12.2021
stable-baselines3	1.2.0	08.09.2021
torch	1.9.1	22.09.2021

Literaturverzeichnis

[Abadi u. a. 2016] Abadi, Martín ; Barham, Paul ; Chen, Jianmin ; Chen, Zhifeng ; Davis, Andy u. a.: Tensorflow: A system for large-scale machine learning. In: *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*, 2016, S. 265–283

[Achiam 2018] Achiam, Joshua: *Spinning Up in Deep Reinforcement Learning*. `https://spinningup.openai.com/en/latest/`. Version: 2018. – Letzter Besuch: 2022-01-05

[Akiba u. a. 2019] Akiba, Takuya ; Sano, Shotaro ; Yanase, Toshihiko ; Ohta, Takeru ; Koyama, Masanori: Optuna: A Next-generation Hyperparameter Optimization Framework. In: *Proceedings of the 25rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2019

[Arens u. a. 2013] Arens, Tilo ; Hettlich, Rolf Busam F. ; Karpfinger, Christian ; Stachel, Hellmuth: *Lehrbuch*. Bd. 1: *Analysis und lineare Algebra mit Querverbindungen*. Berlin and Heidelberg : Springer Spektrum, 2013. – ISBN 978–3–8274–2309–2

[Barlow 1999] Barlow, H B.: Unsupervised learning: introduction. Version: 1999. `http://books.google.com/books?id=yj04Y0lje4cC`. In: Hinton, Geoffrey E. (Hrsg.) ; Sejnowski, Terrence J. (Hrsg.): *Unsupervised Learning: Foundations of Neural Computation*. Bradford Company Scituate, MA, USA, 1999, 1-17

[Barto u. a. 1983] Barto, Andrew G. ; Sutton, Richard S. ; Anderson, Charles W.: Neuronlike adaptive elements that can solve difficult learning control problems. In: *IEEE Transactions on Systems, Man, and Cybernetics* SMC-13 (1983), Nr. 5, S. 834–846. `http://dx.doi.org/10.1109/TSMC.1983.6313077`. – DOI 10.1109/TSMC.1983.6313077

[Bewersdorff 2011] Bewersdorff, Jörg: *Statistik - wie und warum sie funktioniert: Ein mathematisches Lesebuch*. 2011. Vieweg+Teubner Verlag, 2011. `http://dx.doi.org/10.1007/978-3-8348-8264-6`. `http://dx.doi.org/10.1007/978-3-8348-8264-6`

[Botvinick u. a. 2019] Botvinick, Mathew ; Ritter, Sam ; Wang, Jane ; Kurth-Nelson, Zeb ; Blundell, Charles ; Hassabis, Demis: Reinforcement Learning, Fast and Slow. In: *Trends in Cognitive Sciences* 23 (2019), 04. `http://dx.doi.org/10.1016/j.tics.2019.02.006`. – DOI 10.1016/j.tics.2019.02.006

[Breiman u. a. 1984] Breiman, L. ; Friedman, J. H. ; Olshen, R. A. ; Stone, C. J. ; no (Hrsg.): *Classification and Regression Trees*. Belmont, CA : Wadsworth International Group, 1984

[Brockman u. a. 2016] Brockman, Greg ; Cheung, Vicki ; Pettersson, Ludwig ; Schneider, Jonas ; Schulman, John ; Tang, Jie ; Zaremba, Wojciech: *OpenAI Gym*. http://arxiv.org/pdf/1606.01540v1. Version: 06 2016

[Brutzkus u. Globerson 2019] Brutzkus, Alon ; Globerson, Amir: Why do Larger Models Generalize Better? A Theoretical Perspective via the XOR Problem. In: Chaudhuri, Kamalika (Hrsg.) ; Salakhutdinov, Ruslan (Hrsg.): *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA* Bd. 97, PMLR, 2019 (Proceedings of Machine Learning Research), 822–830

[Buitinck u. a. 2013] Buitinck, Lars ; Louppe, Gilles ; Blondel, Mathieu ; Pedregosa, Fabian ; Mueller, Andreas u. a.: API design for machine learning software: experiences from the scikit-learn project. In: *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*, 2013, S. 108–122

[Chen u. a. 2018] Chen, Xiaojiao ; Hu, Tommy ; Song, Chaoyang ; Wang, Zheng: Analytical Solution to Global Dynamic Balance Control of the Acrobot. In: *IEEE International Conference on Real-time Computing and Robotics, RCAR 2018, Kandima, Maldives, August 1-5, 2018*, IEEE, 2018, 405–410

[Chicco u. a. 2021] Chicco, Davide ; Warrens, Matthijs J. ; Jurman, Giuseppe: The coefficient of determination R-squared is more informative than SMAPE, MAE, MAPE, MSE and RMSE in regression analysis evaluation. In: *PeerJ Comput. Sci.* 7 (2021), e623. http://dx.doi.org/10.7717/peerj-cs.623. – DOI 10.7717/peerj–cs.623

[Cho 2018] Cho, Chikun: *Go: A Complete Introduction to the Game*. Kiseido Publishing Company, 2018. – ISBN 9784906574506

[Dancey u. a. 2004] Dancey, Darren ; McLean, David ; Bandar, Zuhair: Decision Tree Extraction from Trained Neural Networks. In: Barr, Valerie (Hrsg.) ; Markov, Zdravko (Hrsg.): *Proceedings of the Seventeenth International Florida Artificial Intelligence Research Society Conference, Miami Beach, Florida, USA*, AAAI Press, 2004, 515–519

[Dazeley u. a. 2021] Dazeley, Richard ; Vamplew, Peter ; Cruz, Francisco: Explainable Reinforcement Learning for Broad-XAI: A Conceptual Framework and Survey. In: *CoRR* abs/2108.09003 (2021). https://arxiv.org/abs/2108.09003

[Denis 2016] Denis, Daniel J.: *Applied Univariate, Bivariate and Multivariate Statistics*. Bd. 1. Wiley; 1. Edition, 2016. – ISBN 978–1118632338

[Domingos 2015] Domingos, Pedro: *The master algorithm: How the quest for the ultimate learning machine will remake our world*. New York, NY : Basic Books, 2015. – ISBN 978–0465065707

[Dunn u. a. 2021] Dunn, Jack ; Mingardi, Luca ; Zhuo, Ying D.: Comparing interpretability and explainability for feature selection. In: *CoRR* abs/2105.05328 (2021). https://arxiv.org/abs/2105.05328

[Ellson u. a. 2001] Ellson, John ; Gansner, Emden R. ; Koutsofios, Eleftherios ; North, Stephen C. ; Woodhull, Gordon: Graphviz - Open Source Graph Drawing Tools. In: Mutzel, Petra (Hrsg.) ; Jünger, Michael (Hrsg.) ; Leipert, Sebastian (Hrsg.): *Graph Drawing, 9th International Symposium, GD 2001 Vienna, Austria, September 23-26, 2001, Revised Papers* Bd. 2265, Springer, 2001 (Lecture Notes in Computer Science), 483–484

[Engelhardt u. a. 2021] Engelhardt, Raphael ; Lange, Moritz ; Wiskott, Laurenz ; Konen, Wolfgang: Shedding Light into the Black Box of Reinforcement Learning. In: *Workshop "Trustworthy AI in the Wild KI2021 - 44th German Conference on Artifical Intelligence* (2021)

[Fanta 2019] Fanta, Matěj: *Rules extraction from deep neural networks Master thesis*, Czech Technical University in Prague, Faculty of Nuclear Sciences and Physical Engineering, Diplomarbeit, 08 2019. `http://dx.doi.org/10.13140/RG.2.2.22319.28324`. – DOI 10.13140/RG.2.2.22319.28324

[Fedorenko 1969] Fedorenko, R.P.: The Cauchy problem for Bellman's dynamic programming equation. In: *USSR Computational Mathematics and Mathematical Physics* 9 (1969), jan, Nr. 2, S. 229–238. `http://dx.doi.org/10.1016/0041-5553(69)90105-0`. – DOI 10.1016/0041–5553(69)90105–0

[François-Lavet u. a. 2018] François-Lavet, Vincent ; Henderson, Peter ; Islam, Riashat ; Bellemare, Marc G. ; Pineau, Joelle: An Introduction to Deep Reinforcement Learning. In: *Found. Trends Mach. Learn.* 11 (2018), Nr. 3-4, S. 219–354

[Frochte 2018] Frochte, Jörg: *Maschinelles Lernen: Grundlagen und Algorithmen in Python*. München : Hanser, 2018. – ISBN 978–3446452916

[Fujimoto u. a. 2018] Fujimoto, Scott ; Hoof, Herke van ; Meger, David: Addressing Function Approximation Error in Actor-Critic Methods. In: *CoRR* abs/1802.09477 (2018). `http://arxiv.org/abs/1802.09477`

[GOBASE 2010] GOBASE: *Player Biography Lee Sedol*. `https://gobase.org/information/players/?pp=Lee+SeDol`. Version: 2010

[Goodfellow u. a. 2016] Goodfellow, Ian J. ; Bengio, Yoshua ; Courville, Aaron: *Deep Learning*. Cambridge, MA, USA : MIT Press, 2016

[Greydanus u. a. 2017] Greydanus, Sam ; Koul, Anurag ; Dodge, Jonathan ; Fern, Alan: Visualizing and Understanding Atari Agents. In: *CoRR* abs/1711.00138 (2017). `http://arxiv.org/abs/1711.00138`

[Hao u. a. 2020] Hao, Dong ; Zihan, Ding ; Shanghang, Zhang: *Deep Reinforcement Learning: Fundamentals, Research, and Applications*. Springer Nature, 2020. – ISBN 978–9811540943

[Hastie u. a. 2009] Hastie, Trevor ; Tibshirani, Robert ; Friedman, Jerome: *The elements of statistical learning: data mining, inference and prediction*. 2. Springer, 2009 `http://www-stat.stanford.edu/~tibs/ElemStatLearn/`

[Hawking 1988] Hawking, Stephen: *A Brief History of Time: From the Big Bang to Black Holes*. London : Bantam, 1988

[Heuillet u. a. 2021] Heuillet, Alexandre ; Couthouis, Fabien ; Rodríguez, Natalia D.: Explainability in deep reinforcement learning. In: *Knowl. Based Syst.* 214 (2021), 106685. http://dx.doi.org/10.1016/j.knosys.2020.106685. – DOI 10.1016/j.knosys.2020.106685

[Holzinger 2018] Holzinger, Andreas: Explainable AI (ex-AI). In: *Inform. Spektrum* 41 (2018), Nr. 2, 138–143. http://dx.doi.org/10.1007/s00287-018-1102-5. – DOI 10.1007/s00287–018–1102–5

[Jagielski u. a. 2020] Jagielski, Matthew ; Carlini, Nicholas ; Berthelot, David ; Kurakin, Alex ; Papernot, Nicolas: High Accuracy and High Fidelity Extraction of Neural Networks. In: Capkun, Srdjan (Hrsg.) ; Roesner, Franziska (Hrsg.): *29th USENIX Security Symposium, USENIX Security 2020, August 12-14, 2020*, USENIX Association, 2020, 1345–1362

[Kozak 2018] Kozak, Jan: *Decision Tree and Ensemble Learning Based on Ant Colony Optimization* -. Berlin, Heidelberg : Springer, 2018. – ISBN 978–3–319–93752–6

[Kruse u. a. 2015] Kruse, Rudolf ; Borgelt, Christian ; Braune, Christian ; Klawonn, Frank ; Moewes, Christian u. a.: *Computational Intelligence Eine methodische Einführung in Künstliche Neuronale Netze, Evolutionäre Algorithmen, Fuzzy-Systeme und Bayes-Netze*. 2. Wiesbaden : Springer Vieweg, 2015 (Computational Intelligence). http://dx.doi.org/10.1007/978-3-658-10904-2. http://dx.doi.org/10.1007/978-3-658-10904-2. – ISBN 978–3–658–10903–5

[Lapan 2018] Lapan, Maxim: *Deep reinforcement learning hands-on: Apply modern RL methods, with deep Q-networks, value iteration, policy gradients, TRPO,AlphaGo Zero and more*. Birmingham and Mumbai : Packt Publishing, 2018 (Expert insight). – ISBN 1788834240

[Li u. Liang 2019] Li, Yu-Feng ; Liang, De-Ming: Safe semi-supervised learning: a brief introduction. In: *Frontiers Comput. Sci.* 13 (2019), Nr. 4, 669–676. http://dx.doi.org/10.1007/s11704-019-8452-2. – DOI 10.1007/s11704–019–8452–2

[Li 2017] Li, Yuxi: Deep Reinforcement Learning: An Overview. In: *CoRR* abs/1701.07274 (2017). http://arxiv.org/abs/1701.07274

[Lorenz 2020] Lorenz, Uwe: *Reinforcement learning: Aktuelle Ansätze verstehen - mit Beispielen in Java und Greenfoot*. Berlin and Heidelberg : Springer Vieweg, 2020. – ISBN 978–3–662–61650–5

[Mnih u. a. 2016] Mnih, Volodymyr ; Badia, Adrià P. ; Mirza, Mehdi ; Graves, Alex ; Silver, David u. a.: Asynchronous Methods for Deep Reinforcement Learning. In: *CoRR* abs/1602.01783 (2016). http://arxiv.org/abs/1602.01783

[Mnih u. a. 2013] Mnih, Volodymyr ; Kavukcuoglu, Koray ; Silver, David ; Graves, Alex ; Antonoglou, Ioannis ; Wierstra, Daan ; Riedmiller, Martin A.: Playing

Atari with Deep Reinforcement Learning. In: *CoRR* abs/1312.5602 (2013). http://arxiv.org/abs/1312.5602

[Mnih u. a. 2015] Mnih, Volodymyr ; Kavukcuoglu, Koray ; Silver, David ; Rusu, Andrei A. ; Veness, Joel u. a.: Human-level control through deep reinforcement learning. In: *Nat.* 518 (2015), Nr. 7540, 529–533. http://dx.doi.org/10.1038/nature14236. – DOI 10.1038/nature14236

[Moore 1990] Moore, Andrew: Efficient Memory-based Learning for Robot Control / Carnegie Mellon University. Pittsburgh, PA, 11 1990. – Forschungsbericht

[Moravcík u. a. 2017] Moravcík, Matej ; Schmid, Martin ; Burch, Neil ; Lisý, Viliam ; Morrill, Dustin ; Bard, Nolan ; Davis, Trevor ; Waugh, Kevin ; Johanson, Michael ; Bowling, Michael H.: DeepStack: Expert-Level Artificial Intelligence in No-Limit Poker. In: *CoRR* abs/1701.01724 (2017). http://arxiv.org/abs/1701.01724

[Mourad Chebila 2021] Mourad Chebila: Predicting the consequences of accidents involving dangerous substances using machine learning. In: *Ecotoxicology and Environmental Safety* 208 (2021), 111470. http://dx.doi.org/10.1016/j.ecoenv.2020.111470. – DOI 10.1016/j.ecoenv.2020.111470. – ISSN 0147–6513

[Myers u. Hoppe-Graff 2014] Myers, David G. ; Hoppe-Graff, Siegfried: *Psychologie.* 3., vollst. überarb. und erw. Aufl. Berlin : Springer, 2014 (Springer-Lehrbuch). – ISBN 978–3–642–40781–9

[Necaise 2010] Necaise, Rance D.: *Data Structures and Algorithms Using Python.* 1st. Wiley Publishing, 2010. – ISBN 0470618299

[Oedingen 2021] Oedingen, Marc: *Anwendung von Reinforcement Learning Modellen auf OpenAI-Gym Problemen.* Technische Hochschule Köln - Campus Gummersbach, November 2021. – Forschungsbericht

[Papula 2011] Papula, L.: *Mathematik für Ingenieure und Naturwissenschaftler Band 1: Ein Lehr- und Arbeitsbuch für das Grundstudium.* Vieweg+Teubner Verlag, 2011 (Viewegs Fachbücher der Technik). – ISBN 9783834817495

[Paszke u. a. 2019] Paszke, Adam ; Gross, Sam ; Massa, Francisco ; Lerer, Adam ; Bradbury, James u. a.: PyTorch: An Imperative Style, High-Performance Deep Learning Library. Version: 2019. http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf. In: Wallach, H. (Hrsg.) ; Larochelle, H. (Hrsg.) ; Beygelzimer, A. (Hrsg.) ; Alché-Buc, F. d'(Hrsg.) ; Fox, E. (Hrsg.) u. a.: *Advances in Neural Information Processing Systems 32.* Curran Associates, Inc., 2019, 8024–8035

[Pedregosa u. a. 2011] Pedregosa, F. ; Varoquaux, G. ; Gramfort, A. ; Michel, V. ; Thirion, B. u. a.: Scikit-learn: Machine Learning in Python. In: *Journal of Machine Learning Research* 12 (2011), S. 2825–2830

[Puiutta u. Veith 2020] Puiutta, Erika ; Veith, Eric M. S. P.: Explainable Reinforcement Learning: A Survey. In: *CoRR* abs/2005.06247 (2020). https://arxiv.org/abs/2005.06247

[Quinlan 1993] Quinlan, J. R.: *C4.5: Programs for Machine Learning*. Morgan Kaufmann, 1993. – ISBN 1–55860–238–0

[Raffin 2020] Raffin, Antonin: *RL Baselines3 Zoo*. https://github.com/DLR-RM/rl-baselines3-zoo, 2020. – Letzter Besuch: 2022-01-05

[Raffin u. a. 2019] Raffin, Antonin ; Hill, Ashley ; Ernestus, Maximilian ; Gleave, Adam ; Kanervisto, Anssi ; Dormann, Noah: *Stable Baselines3*. https://github.com/DLR-RM/stable-baselines3, 2019. – Letzter Besuch: 2022-01-05

[Samek u. a. 2019] Samek, Wojciech ; Montavon, Grégoire ; Vedaldi, Andrea ; Hansen, Lars K. ; Muller, Klaus-Robert: *Explainable AI: Interpreting, Explaining and Visualizing Deep Learning*. 1st ed. 2019. Cham : Springer International Publishing and Imprint: Springer, 2019 (Lecture Notes in Artificial Intelligence). http://dx.doi.org/10.1007/978-3-030-28954-6. http://dx.doi.org/10.1007/978-3-030-28954-6. – ISBN 978–3–030–28954–6

[Sato u. Tsukimoto 2001] Sato, Makoto ; Tsukimoto, Hiroshi: Rule extraction from neural networks via decision tree induction, 2001. – ISBN 0–7803–7044–9, S. 1870 – 1875 vol.3

[Schulman u. a. 2017] Schulman, John ; Wolski, Filip ; Dhariwal, Prafulla ; Radford, Alec ; Klimov, Oleg: Proximal Policy Optimization Algorithms. In: *CoRR* abs/1707.06347 (2017). http://arxiv.org/abs/1707.06347

[Silver 2015] Silver, David: *Lectures on Reinforcement Learning*. URL: https://www.davidsilver.uk/teaching/, 2015. – Letzter Besuch: 2022-01-05

[Silver u. a. 2017a] Silver, David ; Schrittwieser, Julian ; Simonyan, Karen ; Antonoglou, Ioannis ; Huang, Aja u. a.: Mastering the game of Go without human knowledge. In: *Nat.* 550 (2017), Nr. 7676, 354–359. http://dx.doi.org/10.1038/nature24270. – DOI 10.1038/nature24270

[Silver u. a. 2017b] Silver, David ; Thomas, Hubert ; Julian, Schrittwieser ; Ioannis, Antonoglou ; Matthew, Lai u. a.: Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm. In: *CoRR* abs/1712.01815 (2017). http://arxiv.org/abs/1712.01815

[Singh u. a. 1995] Singh, Satinder ; Sutton, Richard ; Kaelbling, P.: Reinforcement Learning with Replacing Eligibility Traces. In: *Machine Learning* 22 (1995), 11. http://dx.doi.org/10.1023/A:1018012322525. – DOI 10.1023/A:1018012322525. ISBN 978–0–7923–9705–2

[Stepisnik u. Kocev 2020] Stepisnik, Tomaz ; Kocev, Dragi: Oblique Predictive Clustering Trees. In: *CoRR* abs/2007.13617 (2020). https://arxiv.org/abs/2007.13617

[Sutton 1988] Sutton, Richard S.: Learning to Predict by the Methods of Temporal Differences. In: *Mach. Learn.* 3 (1988), 9–44. http://dx.doi.org/10.1007/BF00115009. – DOI 10.1007/BF00115009

[Sutton 1996] Sutton, Richard S.: Generalization in Reinforcement Learning: Successful Examples Using Sparse Coarse Coding. In: Touretzky, David S. (Hrsg.) ; Mozer, Michael (Hrsg.) ; Hasselmo, Michael E. (Hrsg.): *Advances in Neural Information Processing Systems 8 (NIPS 1995)*, MIT Press, 1996. – ISBN 0–262–20107–0, S. 1038–1044

[Sutton u. Barto 1998] Sutton, Richard S. ; Barto, Andrew: *Reinforcement learning: An introduction*. Cambridge, Massaschusetts and London : The MIT Press, 1998 (A Bradford book). – ISBN 978–0262193986

[Verma u. a. 2018] Verma, Abhinav ; Murali, Vijayaraghavan ; Singh, Rishabh ; Kohli, Pushmeet ; Chaudhuri, Swarat: Programmatically Interpretable Reinforcement Learning. In: Dy, Jennifer G. (Hrsg.) ; Krause, Andreas (Hrsg.): *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018* Bd. 80, PMLR, 2018 (Proceedings of Machine Learning Research), 5052–5061

[Wang u. a. 2018] Wang, Jane ; Kurth-Nelson, Zeb ; Kumaran, Dharshan ; Tirumala, Dhruva ; Soyer, Hubert u. a.: Prefrontal Cortex as a Meta-Reinforcement Learning System. In: *Nature Neuroscience* (2018), 06. http://dx.doi.org/10.1038/s41593-018-0147-8. – DOI 10.1038/s41593–018–0147–8

[Wang u. a. 2015] Wang, Ziyu ; Freitas, Nando de ; Lanctot, Marc: Dueling Network Architectures for Deep Reinforcement Learning. In: *CoRR* abs/1511.06581 (2015). http://arxiv.org/abs/1511.06581

[Watkins u. Dayan 1992] Watkins, Christopher J. C. H. ; Dayan, Peter: Q-learning. In: *Machine Learning* 8 (1992), Mai, Nr. 3, 279–292. http://dx.doi.org/10.1007/BF00992698. – DOI 10.1007/BF00992698. – ISSN 1573–0565

[Wiering u. van Otterlo 2012] Wiering, Marco (Hrsg.) ; van Otterlo, Martijn (Hrsg.): *Adaptation, learning, and optimization*. Bd. Volume 12: *Reinforcement learning: State-of-the-art*. Berlin and Heidelberg u.a : Springer, 2012 http://www.loc.gov/catdir/enhancements/fy1409/2011945323-d.html. – ISBN 978–3–642–27645–3

[Williams 1992] Williams, Ronald J.: Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning. In: *Mach. Learn.* 8 (1992), 229–256. http://dx.doi.org/10.1007/BF00992696. – DOI 10.1007/BF00992696

[Xiao 2019] Xiao, Zhiqing: *Reinforcement Learning: Theory and Python Implementation*. China Machine Press, 2019

[Xu 2021] Xu, Jian: *How to beat the CartPole Game in 5 Lines*. https://towardsdatascience.com/

how-to-beat-the-cartpole-game-in-5-lines-5ab4e738c93f. Version: 2021. – Letzter Besuch: 2022-01-06

[Yang 2019] Yang, Ziqi: Fidelity: A Property of Deep Neural Networks to Measure the Trustworthiness of Prediction Results. In: Galbraith, Steven D. (Hrsg.) ; Russello, Giovanni (Hrsg.) ; Susilo, Willy (Hrsg.) ; Gollmann, Dieter (Hrsg.) ; Kirda, Engin (Hrsg.) ; Liang, Zhenkai (Hrsg.): *Proceedings of the 2019 ACM Asia Conference on Computer and Communications Security, AsiaCCS 2019, Auckland, New Zealand, July 09-12, 2019*, ACM, 2019, 676–678

[Zaslavsky 1982] Zaslavsky, Claudia: *Tic tac toe: And other three-in-a row games from ancient Egypt to the modern computer*. 1st ed. New York : Crowell, 1982. – ISBN 0–690–04316–3

[Zhou u. a. 2018] Zhou, Jie ; Cui, Ganqu ; Zhang, Zhengyan ; Yang, Cheng ; Liu, Zhiyuan ; Sun, Maosong: *Graph Neural Networks: A Review of Methods and Applications*. http://arxiv.org/abs/1812.08434. Version: 2018. – cite arxiv:1812.08434

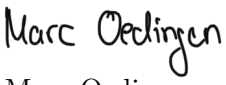
[Zilke 2015] Zilke, Jan R.: *Extracting Rules from Deep Neural Networks*, TU Darmstadt, Knowledge Engineering Group, Diplomarbeit, 10 2015. http://www.ke.tu-darmstadt.de/lehre/arbeiten/master/2015/Zilke_Jan.pdf

[Zilke u. a. 2016] Zilke, Jan R. ; Mencía, Eneldo L. ; Janssen, Frederik: DeepRED - Rule Extraction from Deep Neural Networks. In: Calders, Toon (Hrsg.) ; Ceci, Michelangelo (Hrsg.) ; Malerba, Donato (Hrsg.): *Discovery Science - 19th International Conference, DS 2016, Bari, Italy, October 19-21, 2016, Proceedings* Bd. 9956, 2016 (Lecture Notes in Computer Science), 457–473

Eidesstattliche Erklärung

Ich versichere, die von mir vorgelegte Arbeit selbstständig verfasst zu haben. Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten oder nicht veröffentlichten Arbeiten anderer oder der Verfasserin/des Verfassers selbst entnommen sind, habe ich als entnommen kenntlich gemacht. Sämtliche Quellen und Hilfsmittel, die ich für die Arbeit benutzt habe, sind angegeben. Die Arbeit hat mit gleichem Inhalt beziehungsweise in wesentlichen Teilen noch keiner anderen Prüfungsbehörde vorgelegen.

Gummersbach, 17. Januar 2022



Marc Oedingen