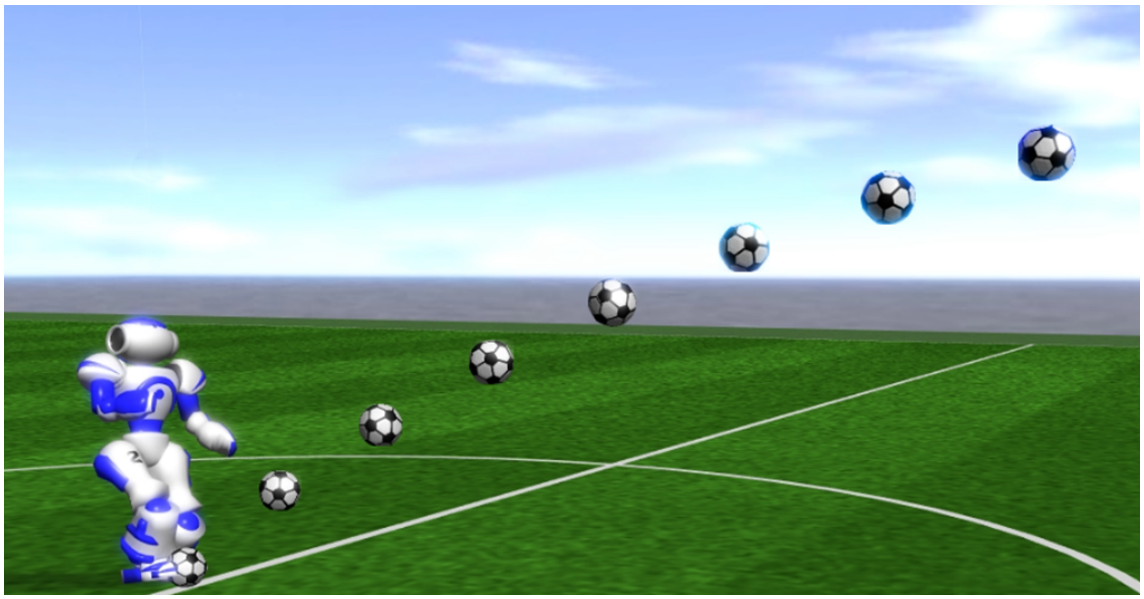
Hochschule Offenburg
offenburg.university

Analyse des Deep Reinforcement Learning Algorithmus PPO2 in der RoboCup Umgebung



Martin Spitznagel

BACHELORARBEIT

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

Studiengang Angewandte Informatik

Fakultät Elektrotechnik, Medizintechnik und Informatik
Hochschule für Technik, Wirtschaft und Medien Offenburg

01.04.2020 - 01.10.2020

Betreuer

Prof. Dr. rer. nat. Klaus Dorer, Hochschule Offenburg

Prof. Dr.-Ing. Janis Keuper, Hochschule Offenburg

Spitznagel, Martin:
Analyse des Deep Reinforcement Learning Algorithmus PPO2 in der RoboCup Umgebung / Martin Spitznagel. –
BACHELORARBEIT, Offenburg: Hochschule für Technik, Wirtschaft und Medien Offenburg, 2020. 87 Seiten.

Spitznagel, Martin:
Analysis of the deep reinforcement learning algorithm PPO2 in the RoboCup environment / Martin Spitznagel. –
BACHELOR THESIS, Offenburg: Offenburg University, 2020. 87 pages.

Eidesstattliche Erklärung

Hiermit versichere ich eidesstattlich, dass die vorliegende Bachelor-Thesis von mir selbstständig und ohne unerlaubte fremde Hilfe angefertigt worden ist, insbesondere, dass ich alle Stellen, die wörtlich oder annähernd wörtlich oder dem Gedanken nach aus Veröffentlichungen, unveröffentlichten Unterlagen und Gesprächen entnommen worden sind, als solche an den entsprechenden Stellen innerhalb der Arbeit durch Zitate kenntlich gemacht habe, wobei in den Zitaten jeweils der Umfang der entnommenen Originalzitate kenntlich gemacht wurde. Ich bin mir bewusst, dass eine falsche Versicherung rechtliche Folgen haben wird.

Ich bin damit einverstanden, dass meine Arbeit veröffentlicht wird, d. h. dass die Arbeit elektronisch gespeichert, in andere Formate konvertiert, auf den Servern der Hochschule Offenburg öffentlich zugänglich gemacht und über das Internet verbreitet werden darf.

Offenburg, 01.04.2020 - 01.10.2020

Martin Spitznagel

Zusammenfassung

Analyse des Deep Reinforcement Learning Algorithmus PPO2 in der RoboCup Umgebung

Seit 2009 nimmt das Team "magmaOffenburg" an der 3D-Simulationsliga des RoboCups teil. Für das erfolgreiche Abschneiden in Turnieren ist die Qualität der erlernten Bewegungsabläufe ein zentraler Faktor. Bisher wurden genetische Algorithmen verwendet, um verschiedenste Aktionen zu erlernen sowie zu optimieren. In dieser Arbeit wird der Deep Reinforcement Learning Algorithmus Proximal Policy Optimization für das Erlernen bestimmter Bewegungen verwendet. Um ein Verständnis für dessen einflussreichen Parameter zu erhalten, werden Größen wie paralleles Lernen, Hyperparameter, Netzwerktopologie, Größe des Observationspace sowie asynchronem Lernen anhand dem Kicken aus dem Stand evaluiert. Durch die Ergebnisse der Evaluierung konnte der erlernte Kick signifikant verbessert werden und sein genetisch erlerntes Gegenstück im Spiel ablösen. Drüber hinaus wurden die Erkenntnisse anhand dem Laufen lernen evaluiert und Zusammenhänge bzw. Unterschiede der zwei Lernprobleme festgestellt.

Abstract

Analysis of the deep reinforcement learning algorithm PPO2 in the RoboCup environment

The team ”magmaOffenburg” has been taking part in the 3D simulation league of the RoboCup since 2009. The quality of the learned behaviors is a central factor for the successful performance in tournaments. So far, genetic algorithms have been used to learn and optimize various behaviors. In this work the deep reinforcement learning algorithm Proximal Policy Optimization is used for learning behaviors. Its configuration consisting of parallel learning, hyperparameters, network topology, size of the observation space and asynchronous learning is evaluated using kicking from a standing start. The results of the evaluation made it possible to significantly improve the kick learned and replace its genetically learned counterpart in the game. Also, the results were evaluated based on learning to walk and differences between the two learning problems were determined.

Inhaltsverzeichnis

1 Einleitung **1**
1.1 Motivation . . . 1
1.2 Ziele . . . 2
1.3 Aufbau . . . 2

2 RoboCup-Domäne **4**
2.1 RoboCup3D . . . 4
2.2 Simulator . . . 4
2.3 Robotermodell . . . 5

3 Grundlagen **7**
3.1 Reinforcement Learning . . . 7
3.1.1 Markow-Entscheidungsproblem . . . 8
3.1.2 Policy . . . 9
3.1.3 Value Function . . . 10
3.1.4 Q Function . . . 10
3.1.5 Rewardfunktion und Return . . . 11
3.1.6 Discount Factor . . . 12
3.1.7 Generalized Advantage Estimator . . . 13
3.2 Proximal Policy Optimization . . . 16
3.2.1 Actor-Critic Modell . . . 16
3.2.2 Trust Region Methode . . . 17
3.2.3 Clipped Objective Function . . . 17
3.2.4 Updatezyklus . . . 21
3.2.5 Implementierung von Stable-Baselines . . . 22
3.2.6 Hyperparameter . . . 22
3.3 Lernprobleme . . . 24
3.3.1 Kicken aus dem Stand . . . 24
3.3.2 Kicken mit Ausholen . . . 25
3.3.3 Laufen . . . 26
3.3.4 Keep . . . 26
3.4 Observation- und Actionspace . . . 27
3.4.1 Observationspace . . . 27
3.4.2 Actionspace . . . 28

4 Evaluierung anhand Kickwalk-Problem **29**
4.1 Sequentiell/Parallel 30
4.1.1 Sequentielle Herangehensweise 30
4.1.2 Parallele Herangehensweise 32
4.2 Synchron/Asynchron 35
4.2.1 Herangehensweise 35
4.2.2 Ergebnisse 36
4.2.3 Einfluss der Dummy-Values auf das Lernen 38
4.2.4 Weitere Optimierung und Ergebnisse 41
4.3 Hyperparameter 44
4.3.1 Lernrate 45
4.3.2 nSteps 47
4.3.3 Minibatch 50
4.3.4 Episoden 53
4.3.5 Entropy Coefficient 55
4.3.6 Value Function Coefficient 59
4.3.7 Gamma 60
4.3.8 Lambda 62
4.3.9 Clip 63
4.4 Netzwerktopologie 65
4.4.1 Netzwerkgröße 65
4.4.2 Aktivierungsfunktion 68
4.5 Observation-/Actionspace 69
4.6 Pretraining 71
4.7 Vergleich mit Baseline Kicks 73
4.8 Varianz der Startposition 74

5 Vergleich mit Walk-Problem **79**
5.1 Asynchrones Lernen 79
5.2 Einfluss der Dummy-Elemente 81
5.3 Konsistenz 82
5.4 Discount Factor 83

6 Zusammenfassung und Ausblick **85**
6.1 Zusammenfassung 85
6.2 Ausblick 86

Abkürzungsverzeichnis

Tabellenverzeichnis **i**

Abbildungsverzeichnis **ii**

Literatur **iv**

1. Einleitung

Seit 2009 nimmt das Team ”magmaOffenburg” der Hochschule Offenburg an der 3D-Simulationsliga des RoboCups teil. Diese Liga des RoboCups steuert auf Softwarebasis dem hochgesteckten Ziel der RoboCup-Föderation bei. Dabei soll es möglich sein im Jahre 2050 mit eigenständigen humanoiden Robotern, echte Menschen im Fußball spielen zu übertreffen. Die Simulationsliga ist wie ihr echter Gegenpart realitätsnah gehalten. Jeder Roboter hat bis zu 24 Gelenke, welche einzeln angesteuert werden müssen.

In der Vergangenheit konnte das Team ”magmaOffenburg” durch genetisch gelernte Bewegungsabfolgen gute Ergebnisse in weltweiten Turnieren erzielen. In den vergangenen Jahren haben sich Deep Reinforcement Learning Algorithmen als sehr kompetent im eigenständigen Lernen und Lösen bestimmter Aufgaben herausgestellt. Diese neue Herangehensweise könnte eine Alternative zu den bisher verwendeten Algorithmen darstellen.

1.1 Motivation

Deep Reinforcement Learning Algorithmen lernen selbständig bestimmte Aufgabenstellungen zu lösen und können somit meist bessere Ergebnisse erzielen als eine manuelle Implementierung der Aktionen. Durch dessen Verwendung konnten bspw. professionelle Esport-Teams von künstlichen neuronalen Netzen im kompetitiven Multiplayerspiel Dota2 geschlagen werden [1]. Der verwendete Algorithmus war dabei die Proximal Policy Optimization und bewies, dass Reinforcement Learning eine Herangehensweise sein könnte, um das gesetzte Ziel des RoboCups zu erreichen. Die Proximal Policy Optimization wies sich als effizienter und einfach zu optimierenden Algorithmus heraus und könnte ein optimaler Kandidat sein, um einen weiteren ausschlaggebenden Schritt in Richtung Ziel zu machen.

Seit Ende 2019 kann dieser Deep Reinforcement Learning Algorithmus beim Team ”magmaOffenburg” ebenfalls eingesetzt werden, um bestimmte Bewegungsmuster zu erlernen. Derzeit gibt es vier Lernprobleme: Kicken aus dem Stand, Kicken mit Ausholen, Laufen und Torwarthecht. Im Team werden die Lernläufe aber bisher unsystematisch und wenig explorativ gestartet. Der Einfluss der verschiedenen Hyperparametern wurde bisher kaum untersucht. Deshalb werden alle Lernprobleme von einem gemeinsamen Setup gestartet, unabhängig von dessen Begebenheiten. Weitere Optimierungsmöglichkeiten der gesamten Lernmethode wurden bisher auch nicht untersucht.

1.2 Ziele

Das Ziel der Arbeit ist es, diese verschiedenen Größen zu evaluieren und deren Einfluss auf das Lernen zu beobachten. Hierbei soll untersucht werden, ob die Proximal Policy Optimization einfach zu optimieren ist und unabhängig von einem optimalen Setup gute Ergebnisse liefert. Zudem soll der gesamte Lernprozess evaluiert werden und gegebenenfalls optimiert werden.

In der Arbeit werden folgende Größen genauer betrachtet:

- Gegenüberstellung von sequentiellem gegenüber parallelem Lernen
- Einfluss der Hyperparameter der Proximal Policy Optimization
- Pre-Training des Netzwerks
- Einfluss des Rewards auf das Lernziel
- Netzwerktopologie
- Einfluss der Hyperparameter auf verschiedene Lernprobleme

1.3 Aufbau

Die Arbeit ist wie folgt aufgebaut:

- In Kapitel 2 wird der RoboCup3D, sowie Begebenheiten der Simulationsumgebung und das dort verwendete Robotermodell erläutert.

- In Kapitel 3 werden die grundlegenden Reinforcement Learning Basics geschaffen um den Einfluss verschiedener Größen auf das Lernen verstehen zu können.
- Kapitel 4 bildet den Hauptteil der Thesis. Dieser beinhaltet die Evaluierung der genannten Ziele der Thesis, sowie Interpretation der Ergebnisse auf den Lernfortschritt anhand des Kickens aus dem Stand.
- In Kapitel 5 werden einige Erkenntnisse von Kapitel 4 auf das Laufen lernen übertragen und Ähnlichkeiten sowie Unterschiede aufgezeigt.
- Kapitel 6 schließt die Thesis mit einer Zusammenfassung und Ausblick auf mögliche zukünftige Verbesserungen ab.

2. RoboCup-Domäne

Im folgenden Abschnitt wird ein Einblick in die RoboCup-Domäne und dessen verwendeten Physik Simulations- und Visualisierungsumgebung, sowie das Robotermodell gegeben.

2.1 RoboCup3D

Die RoboCup 3D Simulated Soccer League realisiert eine Plattform, um humanoide Roboter gegeneinander im Fußballspiel zu messen. Dabei wird auf eine realistische Simulation, sowie Regeln der Physik geachtet. Der RoboCup3D beschäftigt sich damit mit dem Software-Part des gestellten Ziels der RoboCup Föderation. Im Jahre 2050 soll es möglich sein mittels eigenständigen humanoiden Robotern Menschen im echten Fußball zu schlagen. Beim RoboCup3D treten daher zwei Teams, mit jeweils elf Robotern, gegeneinander an. Das Team mit dem best funktionierenden Komplettpaket gewinnt das Turnier. Durch verbesserte Bewegungsabfolgen, Entscheidungslogiken sowie Kommunikationssysteme können andere Teams übertroffen werden, um dem Sieg ein Schritt näherzukommen. Das Team Magma aus Offenburg konnte in den letzten Weltmeisterschaften und Turnieren gute Ergebnisse erzielen und ist derzeitig Vizeweltmeister.

2.2 Simulator

Der RoboCup3d basiert auf dem Simulationsserver Simspark[1]. Simspark ist eine multiagent Physiksimulation für dreidimensionale Umgebungen, welches auf dem Spark Framework [2] aufbaut. Die grundlegende Physiksimulation wird dabei von

[1] https://gitlab.com/robocup-sim/SimSpark

der Open Dynamic Engine [2] (ODE) bereitgestellt. SimSpark wird seit dem RoboCup 2004 als offizieller Simulationsserver für die simulierte 3D-Liga des RoboCups verwendet. Mit dem Monitorprogramm RoboViz[3] können alle laufenden Instanzen auf dem Simulationsserver visualisiert werden. Abbildung 2.1 zeigt ein Beispiel eines laufenden Spiels.

Abbildung 2.1: Visualisierung eines Fußballspiels in der Visualisierungsumgebung RoboViz [3].

Jeder Spieler auf dem Spielfeld wird als eigener Prozess realisiert und kommuniziert mit dem Server mittels TCP-Verbindung. Der Server liefert für einen Spieler dessen Wahrnehmung. Der Roboter muss darauf hin zwischen Aktionen entscheiden, um im Endeffekt möglichst viele Tore zu erzielen. Die Tickrate des Servers ist dabei 20ms und liefert jedem Agenten in dieser Periode die aktuelle Wahrnehmung, sowie weitere Informationen über das Spiel. Der Client muss seine auszuführenden Aktionen an den Server schicken, welche daraufhin im nächsten Tick ausgeführt werden. Falls der Roboter seine Entscheidungen nicht rechtzeitig zum Server schickt, werden die vorher gesendeten Bewegungen der Aktoren, in gesetzter Winkelgeschwindigkeit, fortgesetzt. Die komplette Wahrnehmung ist nicht deterministisch und mit einem gewissen Rauschen behaftet.

2.3 Robotermodell

Das verwendete Robotermodell basiert auf der echten Version des Nao-Roboters der Firma Softbank-Robotics[4] und verfügt über 24 Gelenke, Drucksensoren in den Fü-

[2] https://www.ode.org
[3] https://github.com/magmaOffenburg/RoboViz
[4] https://www.softbankrobotics.com

ßen, sowie einem Gyroskop und Beschleunigungssensor. In dieser Thesis wird das Modell mit Zehen verwendet. Diese Untervariante hat sich in vorherigen Arbeiten des Team Magmas als positiv für verschiedene Bewegungsmuster herausgestellt [3], [4].

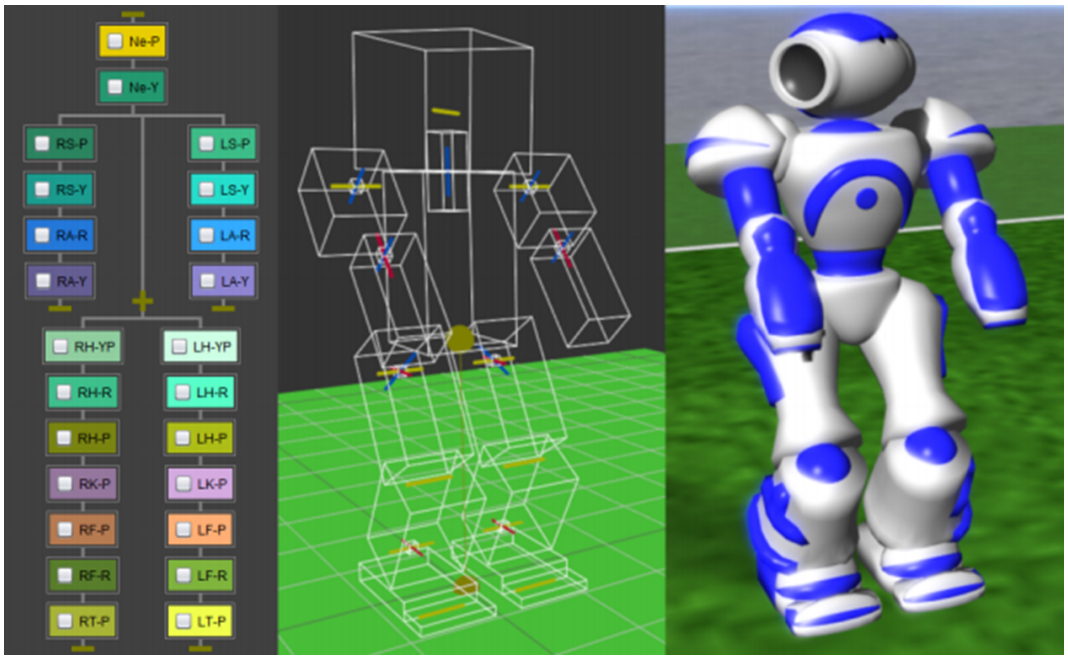


Abbildung 2.2: Robotermodell des Naos anhand den Gelenken (links), Boxmodell (mitte) und Darstellung im RoboViz (rechts) [3].

In Abbildung 2.2 (links) werden die Gelenke anhand ihrer Position des Naos visualisiert. Die Gelenke sind in Geschwindigkeit sowie Winkel in verschiedene Richtungen ansteuerbar. Eine Besonderheit des echten Naos ist, dass die YawPitch-Gelenke der Hüfte nur aus einem Motor bestehen. Somit ist es in der Realität nicht möglich beide Gelenke unabhängig voneinander anzusteuern [5]. Bei der simulierten Version des Naos ist das nicht der Fall. Zusätzlich sind bei der echten Version Motoren mit unterschiedlicher Stärke verbaut. In der Simulation wird dies ignoriert und allen Aktoren dieselbe Kraft zugewiesen.

Äußerlich sieht der simulierte Nao (rechts) der echten Version ähnlich. Dennoch unterscheiden sie sich in der eigentlichen physikalischen Form. Wie in Abbildung 2.2 (mitte) zu sehen ist, sind die einzelnen Körperteile des Naos als Boxen repräsentiert. Die Kollisionen zwischen den physikalischen Objekten ist deshalb sehr vereinfacht dargestellt.

3. Grundlagen

In den folgenden Kapiteln werden die Grundlagen für den eigentlich Hauptteil der Thesis geschaffen und die zugrundeliegenden Reinforcement Learning Basics erläutert. Zudem soll die Funktionsweise der Proximal Policy Optimization als eigentlicher Hauptfokus der Thesis verdeutlicht werden. Darauf aufbauend werden die existierenden Lernprobleme beim Team Magma in Offenburg aufgezählt und deren Beschaffenheiten erläutert.

3.1 Reinforcement Learning

Sei es nun ein Kind, welches die ersten Schritte vollbringt, oder der Leistungssportler/in der seine Konkurrenten übertreffen möchte. Alle lernenden Individuen haben eines gemeinsam: Alle lernen in dem sie mit ihrer Umgebung interagieren und erkennen, welche Aktionen sie einen Schritt näher zu ihrem Ziel bringt. Jegliche Fortschritte sowie Rückschritte im Erreichen des Ziels wirken auf das Individuum und formen somit das eigentliche Wissen zwischen Aktionen und Reaktion des Umfelds. Diese gewisse Verbindung zwischen Fortschritt und Aktionen beschreibt das Wissen des Individuums über dessen Umgebung [6]. Ob nun das Kleinkind oder der Leistungssportler/in, beide werden geleitet aufgrund irgendeines Triebs, um das gesetzte Ziel zu erreichen. Beim Kind wohl die Unabhängigkeit von den Eltern, wobei jeder einzelne Schritt eine Errungenschaft in Richtung Selbständigkeit ist. Beim Leistungssportler wohl eher das jahrelange Training und das Übertreffen des Konkurrenten und des daraus resultierenden Prestiges.

Das Reinforcement Learning (RL), oder auch bestärkendes Lernen, baut auf diese grundlegende Tatsache des Lernens auf und überträgt sie in den Bereich der Informatik. Wobei nun der Fokus auf das Lösen eines gewissen Lernproblems und die Maximierung einer Belohnung gelegt wird. Eine Herangehensweise ist es, das

menschliche Gehirn als künstliches neuronales Netz zu abstrahiert, um eine Struktur zu bilden, welche Unmengen an Zusammenhänge und Wissen speichern kann. Ähnlich wie beim Menschen muss das Netz unterschiedlichste Zusammenhänge an Daten herauskristallisieren und davon lernen, um zukünftige Problematiken zu lösen. Dabei entwickelt sich das System weiter, in dem es mit seiner Umgebung interagiert und sozusagen "einfach ausprobiert".

Dabei unterscheidet sich das Reinforcement Learning maßgeblich von anderen Lernmethoden wie zum Beispiel dem überwachten Lernen (engl. supervised learning). Die vier folgenden Tatsachen definieren das Reinforcement Learning und grenzen es von anderen Lernmethoden ab [7]:

- Es gibt keinen Supervisor der vorgibt welche Aktionen gut oder schlecht sind. Die lernende Instanz muss selbst Zusammenhänge zwischen Aktionen und deren Belohnungen feststellen.
- Die Rückmeldung der gewählten Aktionen erreicht die Instanz oft verspätet. Das Lernproblem ist dabei keine fest definierte Funktion mit einem einzelnen Eingabeparameter und dessen Belohnung. Beim Reinforcement Learning werden Abfolgen von Aktionen durchgeführt, welche nach einem bestimmten Zeitrahmen im Erhalten einer Belohnung enden.
- Die gesammelten Daten sind abhängig vom derzeitigen Zustand, in welchem sich die Umgebung und das Individuum befindet.
- Bei anderen Lernmethoden verändert sich der zugrundeliegende Datensatz nicht. Beim Reinforcement Learning ist der erzeugte Datensatz abhängig vom Fortschritt des Systems und somit nie gleich oder fest definiert.

Alle Reinforcement Learning Probleme können unter dem Markow-Entscheidungsproblem zusammengefasst werden. Das Markow-Entscheidungsproblem beschreibt eine sequentielle Aneinanderreihung von Entscheidungen, Aktionen und kommenden Rewards.

3.1.1 Markow-Entscheidungsproblem

Im folgenden Abschnitt wird das Markow-Entscheidungsproblem (engl. markov decision process (MDP)) und dessen grundlegende Elemente sowie Funktionsweise erläutert. Dabei wird zwischen Agent und Umgebung unterschieden. Die lernende

Instanz, welche Entscheidungen treffen muss, wird dabei Agent genannt. Diese beiden Instanzen stehen dauerhaft in Kontakt und interagieren miteinander. Der Agent führt eine Aktion aus worauf die Umgebung dem Agenten eine neue Situation darbietet [8]. Zusätzlich bekommt der Agent eine bestimmte Belohnung abhängig von der Güte der gewählten Aktion, den sogenannten Reward [6].

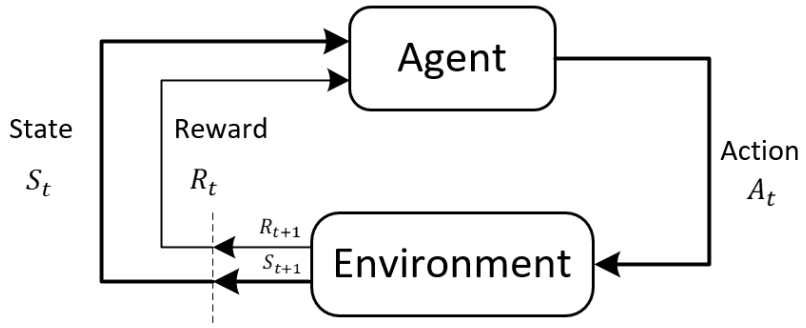


Abbildung 3.1: Die Interaktionen zwischen Agent und Umgebung in einem Markow-Entscheidungsproblem nach [6]

Wie in Abbildung 3.1 visualisiert, interagiert der Agent mit der Umgebung zu bestimmten Zeitpunkten t. Zu jedem dieser Zeitintervalle erhält der Agent eine Wahrnehmung der Umgebung, $S_t \in \mathcal{S}$ und wählt daraufhin eine Aktion $A_t \in \mathcal{A}(s)$. Zum Zeitpunkt $t + 1$ bekommt der Agent daraufhin den Reward $R_{t+1} \subset \mathbb{R}$ und befindet sich nun in einem neuen State S_{t+1} [8].

3.1.2 Policy

Diese Übergangslogik von S_t zu S_{t+1} ist abhängig von der Umgebung und der ausgeführten Aktion A_t. Dabei muss die Aktion abhängig von dem derzeitigen Zustand der Umgebung gewählt werden. Die Instanz, welche Aktionen für bestimmte States vorschlägt und entscheidet, wird Policy genannt. Die Policy beschreibt dabei eine Funktion, welche als Input einen State bekommt und eine Aktion liefert.

Dabei gibt es zwei unterschiedliche Ansätze. Die deterministische Policy π, welche eine fest definierte Aktion auf einen gewissen State zurückgibt mittels $\pi : S \rightarrow A$. Zusätzlich gibt es den Ansatz der stochastischen Policy. Hierbei wird nicht genau eine Aktion als Antwort auf einen State geliefert, sondern die Wahrscheinlichkeit für eine gewisse Aktion, $\pi : S \times A \rightarrow [0, 1]$. Die Summe aller Aktionen liefert eine Wahrscheinlichkeit von 1 zurück und ist als $\sum_{a \in A} \pi(s, a)$ definiert.

Der Ablauf für das Anwenden dieser Policy anhand eines RL Problems ist wie folgt:

- Eine Startdistribution vom initialen State s_0 wird erstellt.
- Die Policy π schlägt eine Aktion $a_0 = \pi(s_0)$ vor.
- Der Übergang zu s_1 wird durchgeführt und ein Reward r_0 berechnet.

Dieser Ablauf wird wiederholt wobei eine Reihenfolge $s_0, a_0, r_0, s_1, a_1, r_1, ...$ entsteht, welche bei einem gewissen Endzustand terminiert und wieder beim Initialstate s_0 anfängt. Die abgeschlossene Abfolge der verschiedenen States und Aktionen wird Trainingssequenz genannt. Die Policy ist Teil des Agents und versucht die Umgebung im Vorteil des Agents mittels Ausführen der richtigen Aktionen zu manipulieren [9].

3.1.3 Value Function

So gut wie alle RL Algorithmen verwenden eine Value Function, die eine Aussage darüber trifft, wie gut der Agent sich fühlen darf in einem gewissen State zu sein und mit welchen kommenden Rewards er vom derzeitigen Zustand rechnen kann [6].

Die zukünftigen Rewards, definiert als G_t, sind abhängig von den Aktionen, die der Agent wählen wird, weshalb diese Value Function $V_\pi(s)$ in Abhängigkeit zur Policy π definiert wird. Sie spiegelt die kommenden Rewards ab dem State s wider, falls der Agent anhand der Policy π handelt und liefert somit einen Erwartungswert. Diese sogenannte State-Value Function ist wie folgt definiert:

$$V_\pi(s) = \mathbb{E}_\pi[G_t|S_t = s] \tag{3.1}$$

Das Ziel eines jeden MDP ist es die beste Policy zu finden welche die höchste Summe der Rewards erhält. Dabei soll der Wert von $V_\pi(s)$ für alle $s \in \mathcal{S}$ maximal sein [9].

3.1.4 Q Function

Mit dem Erwartungswert kann eine Aussage getroffen werden, wie gut der aktuelle State anhand der derzeitigen Policy π ist. Er trifft eine Aussage über die Güte des States für den Agenten in seiner Umgebung. Eine weitere Größe, welche hier eingeführt werden muss, ist die sogenannte State-Action-Value-Function, oder auch Q

Function genannt. Diese Q Function (Formel 3.2), definiert wie gut eine Aktion für den Agent ausgehend von einem bestimmten State ist [10].

$$Q_\pi(s, a) = \mathbb{E}_\pi[G_t|S_t = s, A_t = a] \tag{3.2}$$

Dabei wird der erwartete Rückgabewert von einem State s anhand Policy π, falls Aktion a gewählt wird, zurückgeliefert. Die Value Function liefert eine Aussage wie gut der derzeitige State für den Agenten ist, die Q Function hingegen wie gut die Wahl einer spezifischen Aktion ausgehend von diesem State ist. Somit ist es einfacher für den Agenten die nächste und beste Aktion zu wählen, da er den Nutzen jeder Aktion in den Zuständen kennt. Dies ist mit der State-Value Function nicht ohne Weiteres möglich.

3.1.5 Rewardfunktion und Return

Spezifischer im Anwendungsfeld des DRL ist der Reward eine numerische Zahl in einem bestimmten Wertebereich, welche durch die Rewardfunktion definiert wird. Der Agent hat dabei als grundlegendes Ziel die Summe der erhaltenen Rewards zu maximieren.

Diese Rewardfunktion kann und muss bei unterschiedlichen Lernproblemen verschiedene Formen annehmen und dementsprechend angepasst werden. Dabei können Rewards pro Zeitintervall oder auch nur bei Erreichen eines festgelegten Ziels vergeben werden [6]. Der Reward sollte dabei nicht verwendet werden, um dem Agenten bestimmte Aktionsfolgen vorzulegen. Würde man zum Beispiel ein Agent das Schach spielen beibringen wollen, wäre es kontraproduktiv im Rewards für gewisse Spielzüge zu vergeben, da er so eventuell einen Weg findet diese Spielzüge häufig auszuführen ohne das eigentliche Hauptziel, das Spiel zu gewinnen, zu erreichen [6].

Dabei wird der erwartete Reward G_t aus der Summe der darauf folgenden Rewards bis zum letzten Intervall T berechnet und wie folgt definiert:

$$G_t = R_{t+1} + R_{t+2} + R_{t+3} + ... + R_T \tag{3.3}$$

3.1.6 Discount Factor

Bei der Berechnung dieses erwarteten Rewards gibt es aber einige Schwierigkeiten. Normalerweise ist die Umgebung unsicher und nicht deterministisch. Dementsprechend sind zukünftige Rewards unsicher und deshalb für den Agenten weniger wertvoll. Um diese Problematik zu adressieren wird im Folgenden das Prinzip des Discount Factors erläutert. Eine weitere Funktionalität die in den meisten gängigen DRL Algorithmen seinen Einsatz findet.

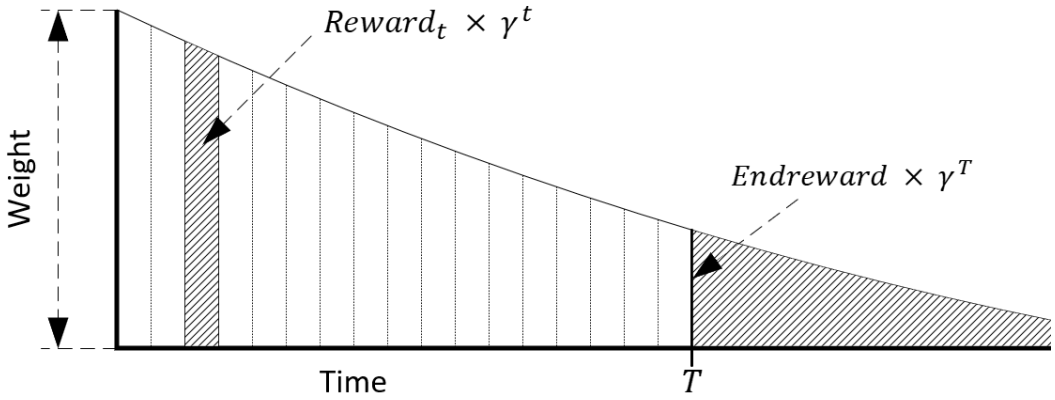


Abbildung 3.2: Veranschaulicht die Gewichtung der erwarteten Rewards abhängig von γ nach [6]

Der Discount Factor ist der sogenannte Abzinsfaktor γ. Ganz nach dem Motto "Besser ein Spatz in der Hand als eine Taube auf dem Dach" werden somit abhängig von γ Aktionen, die früher zum gewünschten Reward führen höher gewertet, als jene welche erst in weiter Zukunft zu einem Reward führen. Abbildung 3.2 visualisiert hierbei die Wichtigkeit der nachfolgenden Rewards ab einem gewissen Startpunkt.

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ... \tag{3.4}$$

Der Rückgabewert G_t bildet die Summe der Discounted Rewards vom Zeitpunkt t. Dabei ist der Wert von Reward R_{k+1} Schritten das Resultat aus $R \times \gamma^k$. Gamma kann dabei Werte von 0 bis 1 annehmen. Damit werden unmittelbare Rewards bevorzugt im Vergleich zu Rewards die erst in Zukunft erreicht werden. Werte von Gamma nahe zu 0 führen zu einem sehr kurzsichtigen Netzwerk, wohingegen ein Gamma Wert von nahe 1 ein sehr weitsichtiges System erzeugt [6].

$$G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \tag{3.5}$$

Nimmt man beispielsweise für $\gamma = 0.95$ und ein Reward von $R = 10$ der in 13 Schritten erreicht wird, ist dieser zukünftige Reward ausgehend vom derzeitigen Standpunkt nun noch $(0.95^{13}) \cdot 10 \approx 5$. Somit ist die Halbwertszeit bei 13 Schritten, also $\gamma^{13} = 0.5$, erreicht.

Da mit der Berechnung von G_t die Summe aller Discounted Rewards vom Zeitpunkt t fortschreitend berechnet wird, muss Gamma in der Praxis entweder kleiner als 1 sein oder es ist ein hartes Endkriterium in der Umgebung vorhanden, welches eine Episode definitiv beenden lässt. Sollte dies nicht der Fall sein, würde diese unendliche Summe der zukünftigen Rewards sich Unendlich annähern. Mit den derzeitigen Lernproblemen im RoboCup von Team Magma gibt es keines ohne hartes Endkriterium, weshalb diese Problematik nicht beachtet werden muss.

Dabei liefert das Einführen des Discount Factors und die entsprechende Summe der Discounted Rewards als Rückgabewert nur eine Einschätzung auf State-Übergänge, die der Agent über eine Episode wahrgenommen hat. Der Agent achtet nicht darauf was hätte passieren können, sondern nur darauf was wirklich passiert. Er achtet nicht auf Rewards, die er eventuell erreichen hätte können, hätte er andere Aktionen ausgeführt.

3.1.7 Generalized Advantage Estimator

Eine weitere Ausführung der Q Function ist die sogenannten Advantage Funktion. Sie definiert wie gut oder schlecht die Wahl einer bestimmten Aktion in einem gewissen State ist. Diese Advantage Function ist in der grundlegenden Form definiert als:

$$A_\pi(s, a) = Q_\pi(s, a) - V_\pi(s) \tag{3.6}$$

Hierbei wird eine Differenz zwischen Q Function und State-Value Function gebildet. Der Vorteil wird berechnet mittels der Aktion a gegenüber des erwarteten Rewards des States s abhängig der derzeitigen Policy π. Dieser Wert für $A_\pi(s, a)$ ist demnach positiv, falls a besser als erwartet für den Endreward war, oder negativ, falls die Wahl der Aktion im Endeffekt den erwarteten erhaltenen Endreward verringert hat [10].

Der Generalized Advantage Estimator (GAE) ist dabei eine Erweiterung des grundlegenden Erwartungswerts [11], baut jedoch nur auf der eigentlichen Idee der Advantage Funktion auf. Hierbei gibt es zunächst zwei unterschiedliche Ansätze für die Berechnung der Q Function und deren Schätzwerte, welche im Folgenden erläutert werden.

Monte-Carlo Estimate

Beim Monte-Carlo Estimate wird eine gesamte abgeschlossene Episode betrachtet. Dabei wird ein Erwartungswert $R(s, a)$ für jeden Step t durch die bevorstehenden Rewards bei jedem kommenden Step errechnet. Alles in Hinsicht auf den erreichten Endreward dieser Episode [11].

$$R_t(s, a) = \sum_{t}^{\infty} \gamma^t r_t \tag{3.7}$$

Die Berechnung des Erwartungswertes bildet sich aus der Summe der Discounted Rewards in Abhängigkeit zum derzeitigen Zeitpunkt t. Hierbei wird der vorher besprochene Discount Factor γ angewandt. Nimmt man das Beispiel des Torwarts in der RoboCup3D Umgebung würde man eine komplette abgeschlossene Episode betrachten. Dabei ist das Endziel das Halten des Balls. Falls der Roboter dies bewältigt erhält er einen entsprechenden Reward dafür. Da sich der Agent in einer stochastischen Umgebung befindet mit einem gewissen Rauschen, kann es dazu kommen, dass er einmal den Ball hält und einmal durch die vorhandene Varianz nicht. Bei der Monte-Carlo Methode ergeben sich dabei sehr unterschiedliche Erwartungswerte, obwohl die gewählten Aktionen relativ ähnlich sein können. Zum Beispiel deutet eine Beobachtung darauf hin, dass nach rechts zu hechten schlecht für den Endreward ist. Da er knapp den Ball mit den Armen verfehlt. Im Gegensatz dazu deutet die andere Beobachtung jedoch darauf, dass somit der Ball gehalten werden kann und der Reward signifikant erhöht wird.

Bei der Monte-Carlo Herangehensweise geht man davon aus, dass sich bei einer großen Anzahl von Aktionsverläufen, die Aktionen welche für den echten Reward der Umgebung zuständig sind, herausstellen und maßgeblich für den Lernfortschritt sein werden.

Temporal Difference Estimate

Eine weitere Herangehensweise für das Errechnen des Erwartungswerts ist die Temporal Difference Methode (TD). Dabei wird nicht auf das Endergebnis der Episode gewartet, sondern der Erwartungswert $R(s, a)$ kann mittels des unmittelbaren Rewards addiert mit einem gelerntem Schätzwert vom darauf folgenden Zeitpunkt $t+1$ bestimmt werden. Dieser Wert von $V(s_{t+1})$ wird, wie bei der Monte-Carlo Herangehensweise, mit dem vorher eingeführten Discount Factor γ multipliziert.

$$R_t(s, a) = r_t + \gamma V(s_{t+1}) \tag{3.8}$$

Dieser Schätzwert ist nie zu 100% akkurat, weshalb das System sehr voreingenommen ist. Deshalb stellt sich dieser Schätzwert bei der Temporal Difference Methode als problematisch heraus.

Im Falle des Torwart-Beispiels, kann es mit dieser Herangehensweise sein, dass der Agent eine Abfolge von Aktionen als weniger wichtig ansieht, da er sie mit Abfolgen vergleicht, die er viel zu wichtig einschätzt. Der Lernfortschritt kann somit negativ beeinflusst werden, falls die optimale Herangehensweise anhand des erwarteten Rewards unterschätzt wird.

Da beide Herangehensweisen Vor- wie auch Nachteile vorweisen wird bei der Proximal Policy Optimization eine Mischung aus beiden verwendet und in einem gewissen Verhältnis angewandt. Deshalb wird hier die Größe λ eingeführt welche das Verhältnis der Interpolation zwischen Monte-Carlo- und TD-Ansatz regelt. Die Herleitung der Interpolation nach [11] wird im Rahmen der Thesis nicht weiter erläutert, da ein grundlegendes Verständnis für die folgenden Ergebnisse der Arbeit ausreicht. Die beiden Extrema für $\lambda = 0, 1$ werden definiert als:

$$GAE(\gamma, 0) : \hat{A}_t := r_t + \gamma V(s_{t+1}) - V(s_t) \tag{3.9}$$

$$GAE(\gamma, 1) : \hat{A}_t := \sum_{l=0}^{\infty} \gamma^l r_{t+1} - V(s_t) \tag{3.10}$$

Beide Herangehenweisen für die Berechnungen von $R_t(s, a)$ werden subtrahiert mit der State-Value Function um den Vor- bzw. Nachteil zu ermitteln. Ist $\lambda = 0$ wird der Vorteil ausschließlich aus Temporal Difference berechnet. falls $\lambda = 1$ gesetzt wird, wird nur die Monte-Carlo-Methode verwendet. Werte zwischen diesen zwei Ex-

trema liefern eine gewichtete Mischung aus beiden Herangehensweisen und regeln das Verhältnis von hoher Varianz und hohem Bias des Netzwerks [12]. Durch dieses Verhältnis der beiden Methoden konnten die Ergebnisse bzw. der Lernfortschritt in gängigen Lernproblemen wie zum Beispiel dem Cart-Pole Problem signifikant verbessert werden [11].

3.2 Proximal Policy Optimization

Aufbauend auf die Grundlagen des Reinforcement Learnings in den vorherigen Kapiteln, wird nun die Funktionalität der Proximal Policy Optimization (PPO) erläutert.

3.2.1 Actor-Critic Modell

PPO basiert dabei auf einer Kombination des Value- sowie Policy-basierten RL Ansatzes: das sogenannte Actor-Critic-Modell. Bei diesem Ansatz wird jeweils die Value Function sowie Policy erlernt. Die Q-Function wird in folgender Form als Critic definiert:

$$Q(s, a, w) \approx Q_\pi(s, a) \tag{3.11}$$

Der Critic nimmt dabei die Funktion der Q Function ein und beurteilt die gewählten Aktionen anhand der Parameter der Action-Value-Function w [7]. Der Actor optimiert die Policy π anhand den Empfehlungen des Critics. Hierbei werden zwei separate neuronale Netze eingesetzt, um den optimalen Actor sowie Critic zu finden [10].

In Abbildung 3.3 sind die Relationen der einzelnen Komponenten eines Actor-Critic-Modells veranschaulicht. Im Gegensatz zu einem MDP wird der Reward nicht unmittelbar zum Actor geführt, sondern wird vorher vom Critic eingeschätzt. Der Critic evaluiert die Policy, während der Actor sie optimiert.

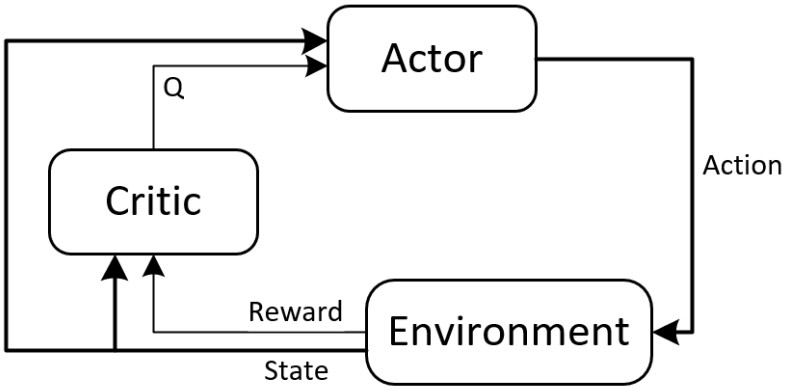


Abbildung 3.3: Visualisierung der Beziehungen zwischen Agent, Critic und Umgebung im Actor-Critic-Modell nach [6].

3.2.2 Trust Region Methode

Eine der größten Herausforderungen bei der Policy Gradient Optimization ist, dass der Agent in seiner Umgebung nicht nur gute Aktionen findet, sondern sich auch in schlechte Richtungen entwickeln kann und die Varianz dementsprechend groß ausfallen kann. Da es sich bei PPO um einen on-policy Algorithmus handelt, ist es hierbei besonders wichtig, dass die neue Policy von der alten nicht zu weit abweicht. Der neue Datensatz, auf dem der Agent trainiert wird, ist abhängig von der neu erzeugten Policy. Falls die neue Policy zu weit von der alten abweicht und einen großen Sprung in die falsche Richtung gemacht hat, kann das Lernen fehlschlagen und das System sich nicht wieder erholen. Ein erster erfolgreicher Ansatz für diese Herangehensweise ist die Trust Region Policy Optimization (TRPO). Hierbei wurde eine weitere Beschränkung eingesetzt, die sogenannte Kullback-Leibler (KL)-Divergenz. Dabei wird die Differenz zwischen alter und neuen Policy beschränkt. Durch diesen Parameter kann die neue Policy nicht weiter von der alten Policy abweichen als einen festgelegten Wert δ [10]. Dabei konnte TRPO in einem weiten Anwendungsbereich eine stetig monotone Verbesserung der Policy und sehr gute Endergebnisse erzielen [13].

3.2.3 Clipped Objective Function

Die Proximal Policy Optimization erweitert den grundsätzlichen Ansatz von TRPO, umgeht aber die intensive Rechenleistung die TRPO benötigt, um festzustellen, ob die KL-Divergenz zwischen der alten und neuen Policy kleiner als δ ist [10]. PPO

verändert die ursprüngliche Objective Function von TRPO. Dabei wird zuerst $r_t(\theta)$ als ein gewisses Verhältnis zwischen alter und neuen Policy definiert. Die daraus folgende Objective Funktion ist definiert als:

$$L^{CPI}(\theta) = \hat{\mathbb{E}}_t[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}\hat{A}_t] = \hat{\mathbb{E}}_t[r_t(\theta)\hat{A}_t] \tag{3.12}$$

Falls nun L^{CPI} maximiert werden sollte, würden die Policy Updates uneingeschränkt groß sein, gegensätzlich dem eigentlichen Ansatz der Trust Region Methode. TRPO fügt dabei einige weitere Beschränkungen ein, welche dies verhindern sollen, wie zum Beispiel die KL-Divergenz, die zu einem stetig monotonem Wachstum der Policy führt. PPO fügt diese Beschränkungen nicht dem ganzen System, sondern der eigentlichen Objective Function hinzu. Deshalb wird hier die Objective Function erweitert. Die daraus folgenden Objective Function ist definiert als:

$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t[min(r_t(\theta)\hat{A}_t, clip(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t)] \tag{3.13}$$

Der erste Part dieser Objective Function ist sozusagen der gleiche wie bei L^{CPI} und bildet auch die Objective Function von TRPO [13]. Der neu hinzugekommene Clipping-Term ϵ beschränkt den Wert des Verhältnis $r_t(\theta)$. Falls dieses Verhältnis von alter zu neuer Policy einen zu großen Schritt machen würde, wird dieser zwischen den Values $1-\epsilon$ und $1+\epsilon$ beschränkt [14]. Dabei ist 1 der Startwert der alten Policy.

Es entstehen zwei grundlegende Fälle:

Fall 1: $\hat{A}_t > 0$

Ist die Advantage Function positiv haben die gewählten Aktionen besser als erwartet abgeschnitten, weshalb deren Wahrscheinlichkeit im Verhältnis zu allen anderen Aktionen erhöht werden sollte. Der Wert für diese Aktion wird unter $r_t(\theta)$ erhöht, bleibt jedoch in den festgelegten Clipping-Werten. Wie in Abbildung 3.4 zu sehen, stagniert der Wert ab der festgelegten Grenze von $1+\epsilon$.

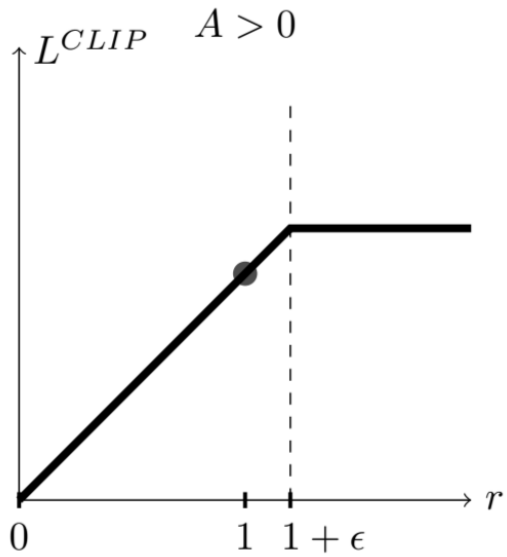


Abbildung 3.4: Plot zeigt einen Zeitpunkt der Funktion L^{CLIP} entlang des Wahrscheinlichkeitsverhältnis r, falls die Advantage Function einen positiven Wert zurückgeliefert hat und die Wahrscheinlichkeit der gewählten Aktion erhöht werden soll. Der graue Punkt auf der Linie zeigt dabei den Startpunkt der Optimierung.

Fall 2: $\hat{A}_t < 0$

Falls der Wert der Advantage Function negativ ist, ist die gewählte Aktion schlechter als erwartet und liefert keinen Vorteil für das eigentliche Ziel des Systems. Deshalb wird der Wert dieser Aktion unter $r_t(\theta)$ verringert. Wie bei einem positiven Wert der Advantage Function wird wieder dieser maximale Wert beschränkt. Dabei kann der Wert dieser Aktion nicht kleiner als $1 - \epsilon$ werden.

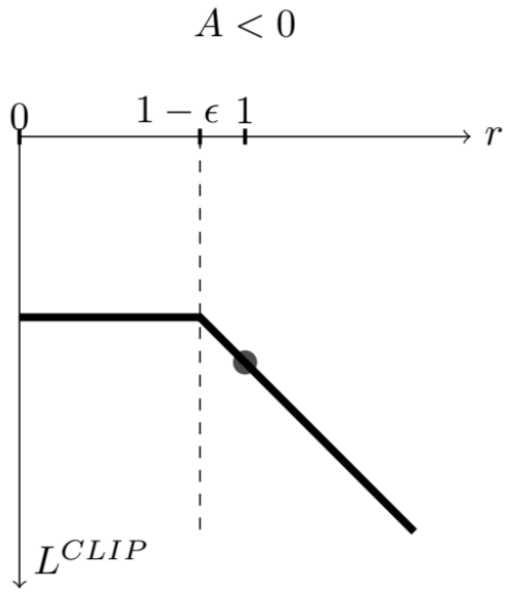


Abbildung 3.5: Plot zeigt einen Zeitpunkt der Funktion L^{CLIP} entlang des Wahrscheinlichkeitsverhältnis r, falls die Advantage Function einen negativen Wert zurückgeliefert hat und die Wahrscheinlichkeit der gewählten Aktion verringert werden soll. Der graue Punkt auf der Linie zeigt dabei den Startpunkt der Optimierung.

Durch den Mininimum-Operator wird das Minimum des Unclipped und Clipped Objective zurückgeliefert. Somit werden die Beschränkungen der Wahrscheinlichkeit nur ignoriert, falls diese das Objective verschlechtern würden [14].

Die neue Objective Function wird darauf hin mit den Ansätzen des Actor-Critic-Modells kombiniert, wobei zwei neuronale Netze als Policy- sowie Value Function verwendet werden. Die komplette Objective Function findet sich daraufhin in folgender Form:

$$L^{CLIP+VF+S}(\theta) = \hat{\mathbb{E}}_t[L_T^{CLIP}(\theta) - c_1 L_t^{VF}(\theta) + c_2 S[\pi_\theta](s_t)] \tag{3.14}$$

$L_T^{CLIP}(\theta)$ beschreibt dabei die vorher eingeführte Clipped-Objective Function. Der zweite Term L_t^{VF} ist der quadrierte Fehler der Value Function. Zusätzlich wird mittels des dritten Terms $S[\pi_\theta](s_t)$ das weitere Erforschen der Umgebung angeregt, in dem es ein Bonus für eine große Entropie vergeben wird [15]. $c1$ und $c2$ sind dabei Koeffizient der Gewichtung der jeweiligen Terme. ϵ beschreibt die Clipping-Reichweite und kann weiter adjustiert werden um die möglichen Abweichungen von alter zu neuer Policy weniger oder mehr zu beschränken.

3.2.4 Updatezyklus

PPO folgt einem gewissen Updateschema, welches im folgenden Abschnitt erläutert wird. Hierbei werden wichtige Größen wie Zeitschritte, Minibatchsize und Epochen behandelt.

Algorithmus 1 zeigt die grundsätzliche Funktionsweise von PPO anhand eines Iterationsschrittes. Zuerst werden für alle Agents Daten gesammelt, indem jeder Agent seine Umgebung mit der alten Policy $\pi_{\theta_{old}}$ erkundet. Daraufhin wird für jeden Datensatz der entsprechende Advantage $\hat{A}$ berechnet. Der Ansatz der parallelen Agents und Umgebungen welcher im A2C Paper veröffentlicht wurde, hat sich hier als Standard durchgesetzt [16]. Die darauf folgende Optimierung wird anhand K Epochen und einer Minibatchgröße von M durchgeführt. Nach Abschluss der Optimierung wird die alte Policy auf die neue gesetzt und daraufhin eine neue Iteration des Updatezyklus gestartet.

Algorithm 1 Eine Iteration der Proximal Policy Optimization

1: **for** $Agent = 1, 2, ..., N$ **do**
2: Führe $\pi_{\theta_{old}}$ in Umgebung aus über T
3: Berechne Erwartungswerte $\hat{A}_1, ..., \hat{A}_T$
4: **end for**
5: Optimiere L^{CLIP} über K Epochen und Minibatchgröße $M \leq NT$
6: $\theta_{old} \leftarrow \theta$

Theoretisch wäre es optimal einen gesammelten Datensatz mehrfach zu bearbeiten, um das größtmögliche Potenzial herauszuholen. Jedoch ist dies bei Standard Policy Gradient Methoden nicht möglich, da somit die Sprünge der alten zur neuen Policy zu groß werden könnten. Die neue Policy könnte somit zu weit von der alten abweichen und sich selbst zerstört haben, da die Änderungen grundlegend immer noch auf einem Schätzwert basieren. Da PPO mittels der Clipped Objective Function immer in einer gewissen Trust Region bleibt, kann die derzeitige Policy anhand eines Datensatzes über mehrere Episoden optimiert werden. Hierbei wird aus dem Datensatz die größtmögliche Information extrahiert. Sobald die neue Policy von der alten zu groß abweichen würde, setzt der Clipping-Operator ein und verhindert einen zu großen Sprung in eine eventuell falsche Richtung.

Bei der Optimierung gibt es unterschiedliche Ansätze wie genau die gesammelten Trainingsdaten für ein Update verwendet werden. Stochastic Gradient Descent ist dabei die grundlegende Form und nimmt für die Optimierung jeweils einen Daten-

satz. Hierbei lernt das Modell sozusagen online, da jeder gesammelte Trainingsdatensatz für die Optimierung verwendet wird. Da dies sehr rechenintensiv ist, gibt es eine Alternative, welche erst das Modell optimiert, sobald eine Epoche abgeschlossen und deren Daten gesammelt wurden. Dieser Ansatz nennt sich Batch Gradient Descent.

Bei PPO wird wie in Algorithmus 1 ein Minibatch Ansatz verwendet, welcher eine Kombination zwischen Stochastic und Batch Gradient Descent bildet. Dabei nimmt Minibatch Gradient Descent (MGD) nicht einzelne Datensätze oder die Gesamtheit der Trainingsdaten für die Durchführung eines Updates, sondern kleinere Abschnitte mit festgelegter Größe. MGD ist die meistverwendete Variante der drei oben genannten Herangehensweisen im Deep Learning Bereich [17]. Bei diesem Ansatz muss jedoch eine weitere Größe als Parameter eingeführt werden.

3.2.5 Implementierung von Stable-Baselines

Derzeitig wird im Team Magma von Offenburg die Implementierung von PPO von Stable-Baselines[1] verwendet. Stable-Baselines ist eine Sammlung von optimierten Algorithmen, aufbauend auf der OpenAI[2] Baseline. Dieses Toolset beinhaltet eine optimierte Implementierung aller gängigen RL Algorithmen, welche zusätzlich eine einheitliche Architektur aufweisen [18]. Zwischen dem originalen Paper [14] und der Implementierung von Stable-Baselines gibt es einige Unterschiede, weshalb diese Version unter dem Namen PPO2 veröffentlicht wurde. Bei dieser neuen Implementierung sind einige Veränderungen wie das Clipping der Value Function nicht dokumentiert. Viele Veränderungen werden erst durch das Studieren des Codes ersichtlich. PPO2 arbeitete dabei mit vektorbasierten Umgebungen und wurde hauptsächlich für die Optimierung auf GPUs entwickelt. Als Optimierer wird Adam als MGD im Updatezyklus eingesetzt. PPO2 ist die derzeitig verwendete Version des Algorithmus von Team Magma.

3.2.6 Hyperparameter

Hyperparameter beschreiben alle einstellbaren Parameter, welche vor dem eigentlichen Start des Lernens eingestellt werden [19]. Dabei werden diese Hyperpara-

[1] https://stable-baselines.readthedocs.io/en/master/
[2] https://openai.com

meter meist manuell definiert und aus gesammelter Erfahrung optimiert. Die Hyperparameter sind ausschlaggebend für den Erfolg der RL Aufgabe. Dabei ist es ein gängiges Problem diese Hyperparameter zu optimieren, da von Lernproblem zu Lernproblem ein unterschiedliches Set an Parametern optimale Ergebnisse erzielen kann [20]. PPO2 soll dabei eine sehr benutzerfreundliche Implementierung des Algorithmus sein, bei dem selbst mit geringfügigem Hyperparametertuning akzeptable Ergebnisse erzielbar sind.

Im Folgenden werden alle Hyperparameter der Implementierung von PPO2 von Stable-Baselines, sowie deren Standardwert und kurze Definition, aufgelistet.

learning_rate (Default: 0.00025)
Die Lernrate definiert wie stark sich die Gewichtungen innerhalb eines Netzwerks anhand des errechneten Werts der Fehlerfunktion anpassen.

n_steps (Default: 128)
Beschreibt die Anzahl der Zyklen, welche für ein Update gesammelt werden.

nminibatches (Default: 4)
Für MGD wird die Anzahl der gesammelten Zyklen in n Minibatches unterteilt. Die Größe von n_steps muss dabei durch den Wert von nminibatches teilbar sein.

episodes (Default: 4)
Beschreibt wie viele Episoden ein gesammelter Datensatz für die Optimierung in Betracht gezogen wird.

lam (Default: 0.95)
Steht für die eingeführte Größe welche das Verhältnis zwischen Monte-Carlo-Rollout und TD-Methode im GAE regelt. Hierbei steht die Abkürzung lam für Lambda λ.

gamma (Default: 0.99)
Gamma definiert den Abzinsfaktor γ der Summe des erwarteten Rewards. Hierbei wird definiert wie kurzsichtig bzw. weitsichtig das System lernen soll und wie wichtig zukünftige Rewards für den Agenten sind.

ent_coef (Default: 0.01)
Der Entropy Coefficient beschreibt die Größe des Entropiebonus während dem Lernen. Hierbei ist ent_coef der Parameter $c2$ in Formel 3.14 und wird Multipliziert mit der Entropie um einen gewissen Bonus für eine hohe Entro-

pie zu vergeben. Das System wird angeregt viele Aktionen auszuführen und sich nicht auf eine zu fokussieren.

vf_coef (Default: 0.5)
Hierbei ist vf_coef der Parameter $c1$ in Formel 3.14 und wird multipliziert mit dem Loss der Value Function.

cliprange (Default: 0.1)
Der Clip Parameter ist für den eigentliche Trust-Region-Ansatz von PPO zuständig. Hier wird mittels cliprange die maximale Abweichung von alter zu neuer Policy festgelegt. Dabei ist der Hyperparameter entsprechend ϵ in der Clipped Objective Function von PPO, unter Formel 3.13.

Jeder dieser Hyperparamter hat in der Evaluierung unter 4.3 einen eigenen Abschnitt in dem dessen Einfluss auf das Lernverhalten verdeutlicht wird.

3.3 Lernprobleme

Beim Lernen der einzelnen Bewegungsabläufe müssen verschiedene Szenarien kreiert werden, um das Lernsetup zu erstellen. Derzeitig gibt es beim Team Magma in Offenburg folgende vier Lernprobleme mit ihren eigenen Spezifikationen.

3.3.1 Kicken aus dem Stand

Beim Kicken aus dem Stand soll der Roboter so weit wie möglich aus dem Stand den Ball in Richtung gegnerisches Tor schießen. Dabei ist die Startposition des Balls fix und der Spieler wird innerhalb eines kleinen Bereichs vor den Ball platziert. Dieser Bereich kann mittels eines Parameters in x- und y-Achse festgelegt werden. Unmittelbar vor dem Ausführen des Kicks führt der Agent noch einige Schritte auf der Stelle aus, um sozusagen eine künstliche Abweichung zu erzeugen und es realitätsnaher an einem echten Spiel zu machen.

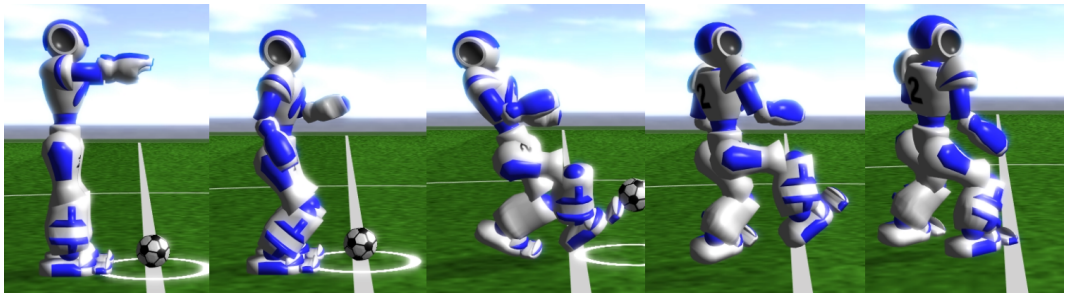


Abbildung 3.6: Veranschaulichung des Bewegungsablaufs des KickWalk-Problems.

Bei diesem Lernproblem wird nur ein Endreward vergeben der sich aus relativer Ballendposition zum Startpunkt und seitlicher Abweichung zusammensetzt. Der Agent wird zusätzlich darauf belohnt, falls er nach dem Ausführen der Bewegung stehen bleibt und nicht hinfällt. Der Roboter wird durch den Reward also dahin getrimmt den Ball möglichst weit und gerade nach vorne zu schießen und dabei einen stabilen Stand zu bewahren. Der Nao soll also auf der Stelle laufen, kicken und daraufhin wieder weiterlaufen können. Das Kicken aus dem Stand liegt auch in verschiedenen Untervarianten vor, wie zum Beispiel das Kicken in eine bestimmte Richtung, oder das Kicken auf eine festgelegte Zieldistanz.

Da sich mit diesem Problem das Lernen als sehr konstant herausgestellt hat, wird das grundlegende Lernproblem im Hauptteil der Evaluierung von PPO2 in den meisten Experimenten verwendet.

3.3.2 Kicken mit Ausholen

Das Kicken mit Ausholen ist sehr ähnlich zum Kicken aus dem Stand. Der Hauptunterschied liegt aber in der Startposition des Spielers. Dabei werden nicht unmittelbar vor dem Kick ein paar Schritte auf der Stelle ausgeführt, sondern bei der Startposition balanciert der Roboter mit angewinkeltem Bein (Abbildung 3.7, drittes Bild von links). Ab dieser Position soll die eigentliche Bewegung erlernt werden und das Schwung holen beim Kicken vorgelegt werden.

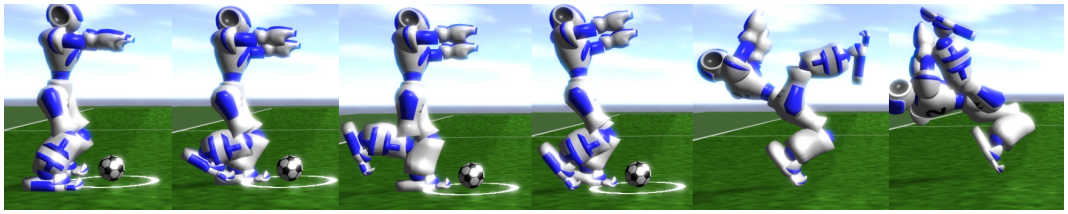


Abbildung 3.7: Veranschaulichung des Bewegunsablaufs des StabilizedKick-Problems.

Der Reward bildet sich wie beim Kicken aus dem Stand mittels zurückgelegter Distanz des Balls, abgezogen von der seitlichen Abweichung. Dieser wird auch erst am Ende einer Episode vergeben. Beim stabilisierten Kick wird aber nicht darauf geachtet ob der Spieler nach dem Ausführen des Kicks auch stehen bleibt. Dafür erhält der Agent keinen zusätzlichen Reward. Dabei ist das Ziel den Ball so weit wie möglich nach vorne zu schießen, unabhängig ob der Spieler nun hinfällt oder nicht. Es wird davon ausgegangen, dass der Ball ohnehin so weit geschossen wird, damit der Spieler genug Zeit hat wieder aufzustehen.

3.3.3 Laufen

Das Walk-Problem ist sehr trivial gehalten und soll dem Agenten das Laufen beibringen. Dabei wird der Spieler auf Höhe des Tors platziert. Im Team Magma gibt es hier unterschiedliche Ansätze der Rewardfunktion. Zum einen die Variante bei der Agent nur ein Endreward für die zurückgelegte Distanz erhält. Zusätzlich gibt es eine Variante, bei der er basierend auf den Bewegungen in den Füßen einen entsprechenden Reward zugeteilt bekommt. Er erhält dabei entsprechende Belohnungen, falls er den richtigen Fuß verwendet oder den Oberkörper relativ gerade hält.

3.3.4 Keep

Das Keep-Problem soll dem Agenten das Halten des Balls bei einem Torschuss beibringen. Dabei wird von unterschiedlichen Positionen auf das Tor geschossen. Der erzeugte Schuss auf das Tor trifft definitiv in das Tor. Der Nao muss also versuchen mit seinen Bewegungen diesen Schuss abzuhalten, um eine gewisse Belohnung zu erhalten.

3.4 Observation- und Actionspace

Wie vorangehend erläutert erhält der Roboter von seiner Umgebung eine bestimmte Wahrnehmung auf die er reagieren muss indem er gewisse Aktionen ausführt. Diese Wahrnehmung und Aktionen werden mittels des Observation- bzw. Actionspace definiert.

3.4.1 Observationspace

Der Observationspace beschreibt dabei die Wahrnehmung des Naos. Diese Informationsmenge kann grundsätzlich von Lernproblem zu Lernproblem unterschiedlich sein. Informationen wie Stellung der einzelnen Gelenke, sowie Position des Roboters und Sichtfeld sind hier festgehalten. Die Wahrnehmung ist dabei realistisch gehalten. Der Agent bekommt somit nicht alle Informationen von dem gesamten Spielfeld, sondern auch nur wirklich diese, welche er normalerweise wahrnehmen kann. Falls der Ball nicht im Blickfeld ist, kann er somit nicht auf den Ball reagieren, da er ihn logischerweise nicht sieht.

Insgesamt ergibt sich dabei eine Sammlung von bis zu 120 Variablen, welche die Umgebung für den Agenten beschreiben. Da es sich hierbei um unterschiedlichste Werte handelt wie zum Beispiel Winkel der Gelenke, oder Position des Balls, werden alle Werte auf -1 bis 1 normalisiert bevor sie den Agenten erreichen. Werte für Counter, Ballposition, Torso und Zielposition sind keine Informationen die direkt der Sensorwahrnehmung des Naos entsprechen.

Dimension	Anzahl	Observation
0	1	Counter Variable
1-4	4	Kopf Gelenke
5-20	16	Oberkörper Gelenke (Schulter, Arme)
21-48	28	Unterkörper Gelenke (Hüfte, Knie, Füße, Zehen)
49-54	6	Ballposition in x,y,z sowie Momentum
55-102	48	Fuß- und Zehensensoren
103-108	6	Beschleunigungssensoren
109-114	6	Gyroskop
115-117	3	Torso
118-119	2	Zielposition

Tabelle 3.1: Spezifikation des Inhalts des Observationspace

3.4.2 Actionspace

Der Actionspace beschreibt die Aktionen, welcher der Nao während eines Spiels wählen kann. Dabei hat jedes Gelenk in seiner unterschiedlichen Orientierung einen Wert für den Winkel sowie Geschwindigkeit mit bestimmten Limitierungen und ist somit ein Continuous Action Space. Daraus ergibt sich beim Modell mit Zehen eine Größe des Actionspace von 48 Variablen. Diese Werte sind zudem Teil des Observationspace. Der Actionspace ist in grundlegender Form der Output des Actors, der den Nao auf dem Spielfeld steuert. Das zugrundeliegende Policy Netz muss auf die Observations mit einem entsprechenden Actionspace reagieren. Diese gewählten Aktionen werden daraufhin als Motorbefehle an die Simulationsumbebung geschickt und ausgeführt. Tabelle 3.2 zählt die einzelnen Variablen mit ihrer Orientierung auf.

Dimension	Anzahl	Gelenk	Orientierung
0 - 3	4	Kopf	Yaw, Pitch
4 - 7	4	Linke Schulter	Yaw, Pitch
8 - 11	4	Linker Arm	Yaw, Roll
12 - 15	4	Rechte Schulter	Yaw, Pitch
16 - 19	4	Rechter Arm	Yaw, Roll
20 - 25	6	Linke Hüfte	YawPitch, Roll, Pitch
26 - 27	2	Linkes Knie	Pitch
28 - 31	4	Linker Fuß	Roll, Pitch
32 - 33	2	Linker Zeh	Pitch
34 - 39	6	Rechte Hüfte	YawPitch, Roll, Pitch
40 - 41	2	Rechtes Knie	Pitch
42 - 45	4	Rechter Fuß	Roll, Pitch
46 - 47	2	Rechter Zeh	Pitch

Tabelle 3.2: Spezifikation des Inhalts des Actionspace

4. Evaluierung anhand Kickwalk-Problem

Das KickWalk-Problem hat sich als am besten geeignetes Lernproblem für die Evaluierung von PPO2 herausgestellt, da es relativ konstant zu guten Ergebnissen führt. Damit können mit einer gewissen Sicherheit Aussagen und Erkenntnisse gezogen werden, welche bei unsicheren Lernproblemen wie dem Walk erschwert wären. Zudem ist der KickWalk vom Agent in einer hinreichenden Zeit zu lernen, weshalb die große Menge an Einstellungsmöglichkeiten der Hyperparameter sowie der Lernmethode nicht über den Rahmen dieser Arbeit hinausgehen. Im Verlaufe der Evaluierung werden zuerst grundlegende Begebenheiten der Lernmethode verglichen und optimiert. Daraufhin wird der Einfluss unterschiedlichster Hyperparameter sowie der Netzwerktopologie auf das Lernen evaluiert. Um das Kapitel abzuschließen werden die gesammelten Erkenntnisse und das beste gelernten Netzwerks mit einem genetisch gelernten KickWalk verglichen der derzeitig im Team Magma des RoboCups verwendet wird.

Bei dieser Evaluierung ist die maßgebliche Größe um Lernläufe zu vergleichen der durchschnittliche Reward und die durchgeführten Episoden, sowie Netzwerkupdates. Eine Episode beschreibt einen durchgeführten Kick und dessen erhaltenen Reward sowie Ergebnis. Eine Episode besteht, im Falle des KickWalks, aus einer festen Anzahl von 18 Zyklen. Bei jedem Zyklus muss der Agent anhand seiner Umgebung eine gewisse Aktion ausführen. Sobald genügend Zyklen gesammelt wurden, wird das Netzwerk optimiert. Solange die Hyperparameter nicht explizit angepasst werden, ist die Anzahl der durchgeführten Updates eine Messgröße des Durchsatzes.

4.1 Sequentiell/Parallel

Um die Serverhardware optimal auszunutzen ist es unerlässlich mehrere Umgebungen und Agenten gleichzeitig trainieren zu lassen. Die Anzahl an parallel laufenden Threads welche an dem Lernproblem arbeiten ist dabei eine wichtige Einstellung für den ganzen Lernprozess. Es gibt zwei unterschiedliche Ansätze, welche im Folgenden näher erläutert und anhand dem Kickwalk-Problem evaluiert werden.

Zum einen gibt es die Möglichkeit mehrere Lernläufe sequentiell zu starten. Hier werden n-Umgebungen und n-Agenten mit einem eigenen Netzwerk erzeugt, welche unabhängig voneinander fortschreiten. Als Endresultat wird das Netzwerk mit dem besten Endergebnis ausgewählt.

Zum anderen werden n-Umgebungen und dessen Agenten mit einem gemeinsamen Netzwerk erzeugt. Dabei sammelt PPO2 die Action-State-Observation von allen Agenten und Umgebungen ein und optimiert darauf hin das dahinter stehende neuronale Netz. Somit profitieren alle parallelen Instanzen von dem Fortschritt eines Einzelnen. Durch diese Herangehensweise sollten sich die Ergebnisse deutlich verbessern, je mehr parallele Instanzen nebenläufig ausgeführt werden [21].

Im Folgenden werden die Ergebnisse dieser zwei Herangehensweisen gegenübergestellt und anhand des KickWalk-Problems evaluiert.

4.1.1 Sequentielle Herangehensweise

Durch Analyse der Serverhardware hat sich eine maximale Auslastung auf 24 Instanzen herausgestellt. Bei der sequentiellen Herangehensweise werden 24 sequentielle Instanzen mit 24 separaten Netzwerken gestartet. Alle Instanzen wurden nach Erreichen des harten Endkriteriums von 48 Stunden Lernzeit gestoppt. Abbildung 4.1 zeigt die Ergebnisse der 24 Instanzen nach diesem Lernlauf. Eine Episode ist definiert als durchgeführten Kick.

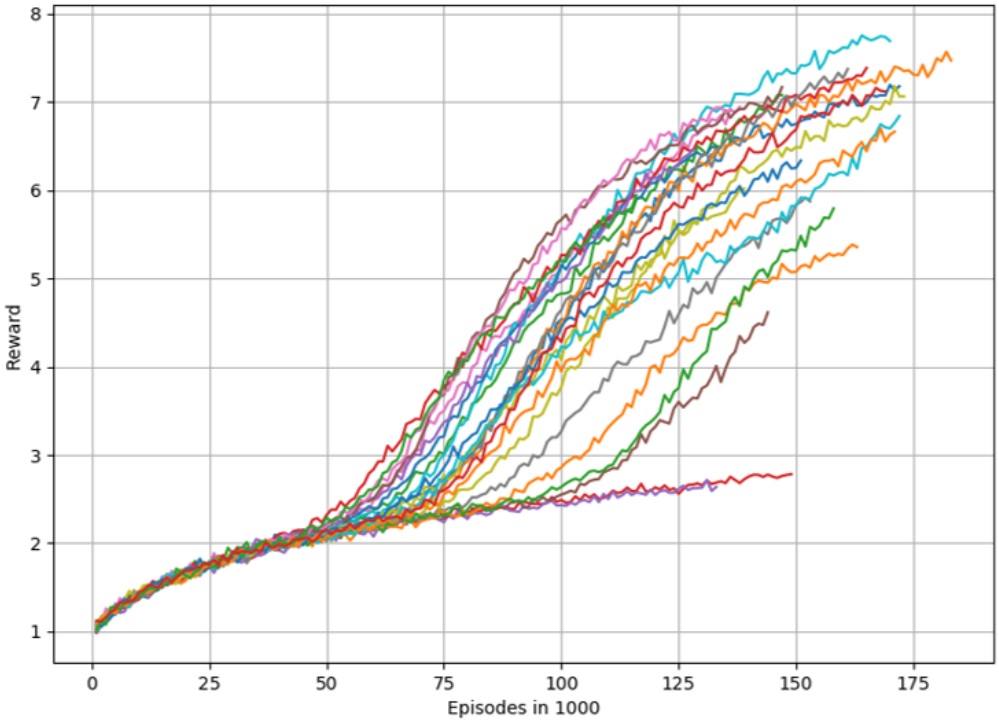


Abbildung 4.1: Sequentieller Ablauf des KickWalk-Lernproblems mit 24 parallel laufenden Instanzen welche unabhängig voneinander Lernen. Jede Linie entspricht einer Instanz. Dabei ist das Voranschreiten der durchgeführten Episoden anhand des durchschnittlichen Rewards gegenübergestellt.

Wie deutlich zu erkennen ist, schreitet jede Instanz unabhängig voneinander fort. Nach Ablauf der festgelegten Zeit erreicht jede Instanz unterschiedliche Endergebnisse des Rewards. Dies kann durch die indeterministische Umgebung erklärt werden. Der Agent probiert explorativ unterschiedlichste Aktionen aus, um eine höhere Belohnung zu bekommen und den Ball weiter zu kicken. Je nach Zufall kommt er dabei früher, später oder auch gar nicht auf Bewegungsabläufe die zu einem höheren Reward führen. Das Endergebnis des Lernlaufs wäre dabei die hellblaue Kurve in Abbildung 4.1 mit einem durchschnittlichen Reward von ca. 7.8. Die unterschiedliche Anzahl der durchgeführten Episoden kann damit erklärt werden, dass eine Episode keine fest definierte Zeitspanne hat. Fällt der Agent direkt hin wird die Episode gestoppt, weshalb diese Episode relativ kurz ist. Falls der Agent beispielsweise jedoch seine Beine ineinander gebracht hat, muss er neu gestartet werden, weshalb viel Overhead-Zyklen erzeugt werden und eine Episode relativ lang braucht.

Deshalb lässt sich feststellen, dass bei einer selbst sehr hohen Anzahl an sequentiellen Läufen, die Varianz relativ hoch ausfällt. Somit kann anhand eines Lernlaufes nur wenig über das aktuelle Setup, die Rewardfunktion oder das Hyperparameterset

ausgesagt werden. Theoretisch wäre es damit möglich, bei einem optimalen Hyperparameterset, dennoch schlechtere Ergebnisse zu erhalten, als bei einem suboptimalen Set.

Die Problematik des unterschiedlichen Voranschreiten des Lernfortschrittes und dessen große Varianz lässt sich mit einem parallelen Ansatz vermindern.

4.1.2 Parallele Herangehensweise

Im Gegensatz zum sequentiellen Ansatz werden bei der parallelen Herangehensweise n-parallele Instanzen mit einem Modell gestartet. Falls nun zum Beispiel eine Instanz eine Bewegungsabfolge entdeckt, welche zu einem verbesserten Reward führt, profitieren alle Agenten davon, da das Modell von der Gesamtheit der gesammelten Zyklen aktualisiert wird.

Hierbei ist es zusätzlich interessant, wie eine steigende Anzahl an parallelen Lernläufen, den Lernfortschritt sowie Endergebnis beeinflusst. Deshalb wird eine unterschiedliche Anzahl ($n = 6, 8, 12, 24$) an parallelen Instanzen gestartet und gegenüber gestellt.

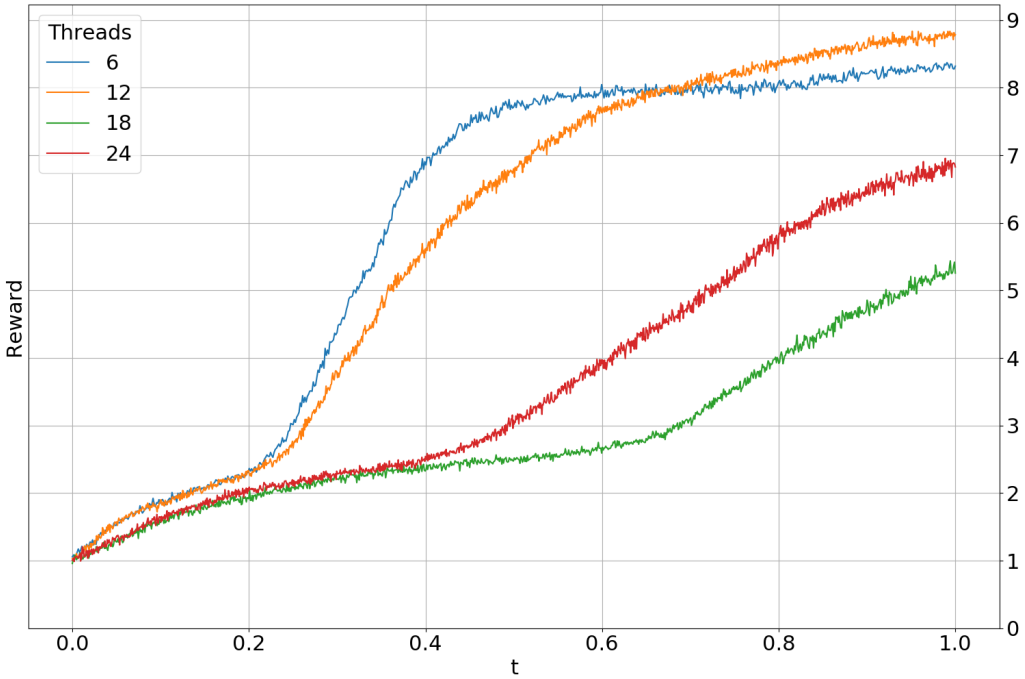


Abbildung 4.2: Paralleler Ablauf des KickWalk-Lernproblems mit parallelen Instanzen welche gemeinsam anhand eines Netzwerks lernen. Jede Linie hat dabei eine unterschiedliche Anzahl an Instanzen. Dabei ist das Voranschreiten der durchgeführten Episoden anhand des durchschnittlichen Rewards der letzten 1000 Episoden gegenübergestellt.

Normalerweise würde man eine Steigerung der Performanz im Faktor n erwarten, jedoch ist dies nicht der Fall wie in Abbildung 4.2 zu erkennen ist. Generell gilt eher: je größer die Anzahl der parallelen Threads, desto langsamer der Lernfortschritt. Dies spiegelt sich auch in der Anzahl an Updates wider die in dem Zeitfenster von 48h durchgeführt wurden, wie in Tabelle 4.1 zu sehen ist. Die Anzahl der Updates ist abhängig der durchgeführten Episoden und gibt eine Aussage über den Durchsatz des Lernlaufs.

Threads	Updates
1	808
6	545
12	481
18	262
24	274

Tabelle 4.1: Anzahl der Updates des Netzwerks in 48 Stunden anhand der parallelen Instanzen.

Obwohl die Anzahl der Threads steigt, sinkt die Anzahl der durchgeführten Updates signifikant. Theoretisch würde man einen Performanzanstieg im Faktor n erwarten, da bei 24 Threads die CPU Auslastung der Server maximal ist. Dies ist jedoch nicht der Fall, da die Implementierung von PPO2 je nach Anzahl der parallel laufenden Umgebungen mehr Observations sammelt. Der Standardwert liegt dabei bei 2048 Cycles die durchgeführt werden müssen. Bei $n=24$ werden somit 49152 Cycles pro Update benötigt. Da trotz der erhöhten parallelen Abarbeitung weniger Updates durchgeführt werden, lässt sich auf ein bestehendes Synchronisationsproblem schließen.

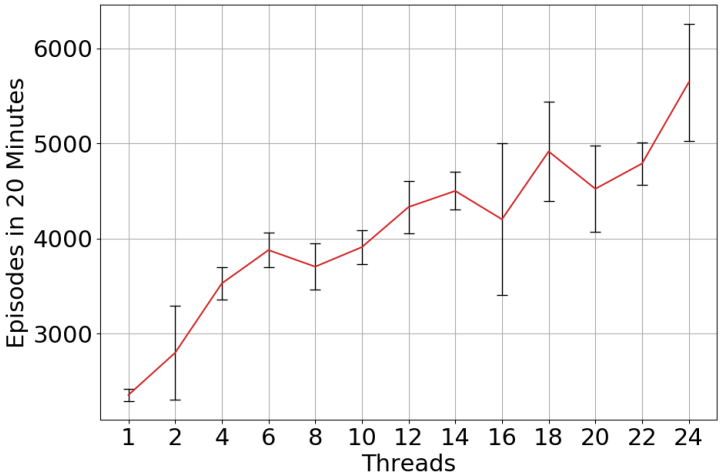


Abbildung 4.3: Anstieg des Durchsatzes anhand steigender Anzahl an parallelen Instanzen.

Deshalb wurden Lernläufe mit unterschiedlicher Threadanzahl gestartet und die durchgeführten Episoden nach 20 Minuten verglichen. Die Ergebnisse sind in Abbildung 4.3 zu sehen. Wie sehr gut zu erkennen ist, steigt die Anzahl der Episoden nicht im Faktor n, ähnlich wie im vorherigen Vergleich der Updates. Da nun bei steigender Anzahl an parallelen Threads mehr Cycles benötigt, aber weniger durchgeführt werden können, wird der ganze Lernprozess signifikant verlangsamt.

Eine Feststellung, die auch nicht außer Betracht gelassen werden sollte, ist die immer noch hohe Varianz der Läufe. Wie in Abbildung 4.2 und Tabelle 4.1 zu sehen ist, lernt die Instanz mit 24 Threads schneller als die mit 18 Threads und kommt somit auch auf ein besseres Endergebnis nach Ablauf der Zeit. Alle Ergebnisse sind abhängig vom Fortschritt des Agenten. Der Agent kann zufällig wichtige Erkenntnisse früher, später oder gar nicht erkennen.

4.2 Synchron/Asynchron

Da die Serverhardware jedoch maximal ausgelastet werden sollte, um das best möglichste Endergebnis zu erzielen, wäre ein Lauf mit 24 parallelen Threads erwünscht. Deshalb wird hier im Folgenden die Herangehensweise und Implementierung eines pseudo-asynchronen Ansatzes erläutert, welcher die genannten Probleme der vorangehenden Kapitel lösen soll. Dieser Ansatz war schon vor Beginn dieser Arbeit implementiert, wurde jedoch noch nicht ausreichend evaluiert. Infolgedessen wird eine optimierte Variante dieses Prototyps implementiert und mit dem ursprünglichen Ansatz verglichen.

4.2.1 Herangehensweise

Das grundlegende Problem der Synchronisation liegt daran, dass PPO2 pro Zeitintervall immer von jeder parallelen Instanz eine Rückmeldung braucht. Da aber je nachdem manche Instanzen mit Overhead-Zyklen beschäftigt sind, wie zum Beispiel das Platzieren des Balls oder Zurücksetzen des Spielers nach einer Episode, können diese keine Observation zurückliefern. Die anderen Threads müssen daraufhin auf die nicht bereiten Instanzen warten. Somit ergibt sich ein großer Performance-Verlust mit steigender paralleler Abarbeitung des Lernproblems.

Ein rein-asynchroner Ansatz mit der PPO2-Implementierung von Stable-Baselines würde eine komplette Umimplementierung bedeuten und ist deshalb im Umfang der Thesis zu aufwendig. Zusätzlich würde eine rein-asynchrone Implementierung die zukünftige Verwendung der Stable-Baselines Algorithmen sowie dessen Funktionalität verhindern. Deshalb wird das asynchrone Verhalten mit Dummy-Observations simuliert, welche zurückgegeben werden, falls eine Instanz derzeit anderweitig beschäftigt ist.

In der ursprünglichen Implementierung von Team Magma wartet PPO2 auf die Observation der ersten parallelen Instanz. Falls diese bereit ist eine Observation zu liefern, sammelt die Python-Anwendung diese ein und fragt bei allen zusätzlichen Instanzen, ob sie ihre Ergebnisse bereitstellen können. Hat die Instanz derzeit keine, da sie sich mit anderen Zyklen beschäftigt, wird eine Dummy-Observation in den Datensatz eingebaut. Diese Dummy-Observations bestehen aus lauter Nullen und liefern keinerlei brauchbare Informationen. Somit ist die Synchronisation der einzelnen Threads entkoppelt und jeder Thread kann nahezu unabhängig voneinander

Episoden durchführen und dessen Resultate zurückliefern. Der Fall, in dem zufällig auf die erste Instanz gewartet wird, da sie sich in Overhead-Zyklen befindet, wird ignoriert, da dieser bei einer hohen Anzahl an parallelen Instanzen vergleichsweise selten vorkommt.

4.2.2 Ergebnisse

Durch diese Entkoppelung kann eine signifikante Steigerung im Lernfortschritt, den durchgeführten Updates und Episoden erzielt werden.

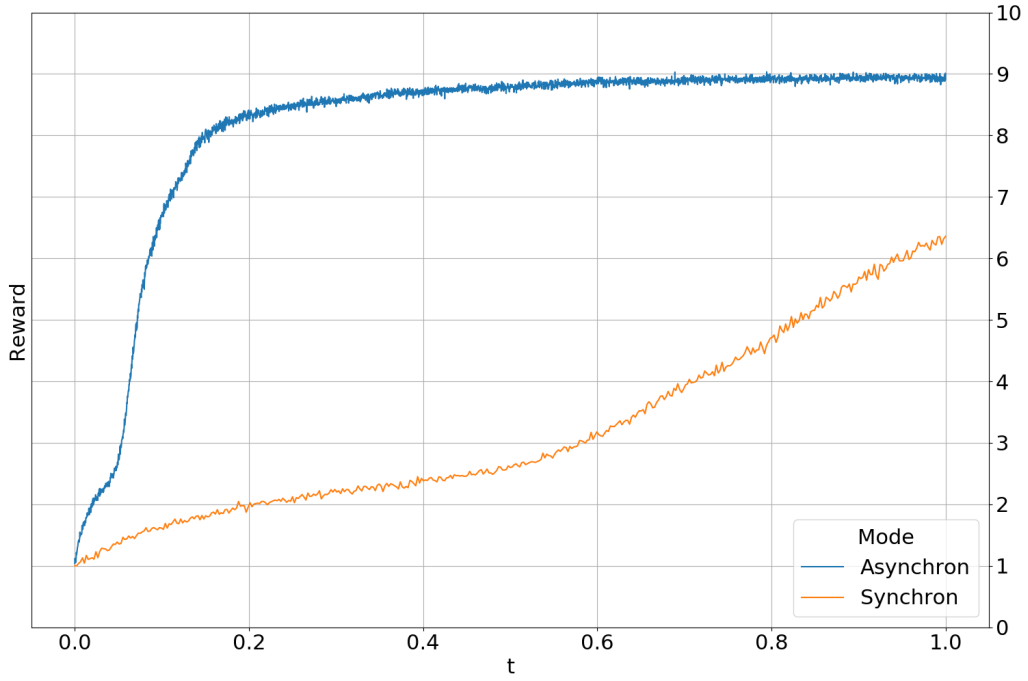


Abbildung 4.4: Vergleich 24 paralleler Instanzen mit asynchroner und synchroner Lernmethode. Dabei ist das Voranschreiten der Zeit anhand des durchschnittlichen Rewards der letzten 1000 Episoden gegenübergestellt. t = 1 steht für die 48 Stunden Grenze.

Abbildung 4.4 zeigt den Vergleich zwischen synchronem und asynchronem Lernlauf mit ansonsten gleichem Setup. Die Ergebnisse zeigen deutlich, dass durch den asynchronen Modus das Endergebnis deutlich schneller erreicht werden kann.

	Synchron	Asynchron
Average Reward	6.83423	8.96709
Updates	166	1627
Episoden in 20 Minuten	5644	18627

Tabelle 4.2: Vergleich des Durchsatzes der asynchronen sowie synchronen Lernmethode.

Im Vergleich schneidet das asynchrone Lernen deutlich besser ab als das Synchrone. Die wichtigsten Größen wie durchschnittlicher Reward und Updates nach 48 Stunden, sowie die Anzahl der durchgeführten Episoden nach 20 Minuten sind in Tabelle 4.2 gegenüber gestellt. Die Ergebnisse des asynchronen Ansatzes sind signifikant besser und beweisen, dass es unter den parallelen Instanzen eine hohe Synchronisation gibt, welche den Lernfortschritt durch Overhead-Zyklen sehr stark verlangsamt.

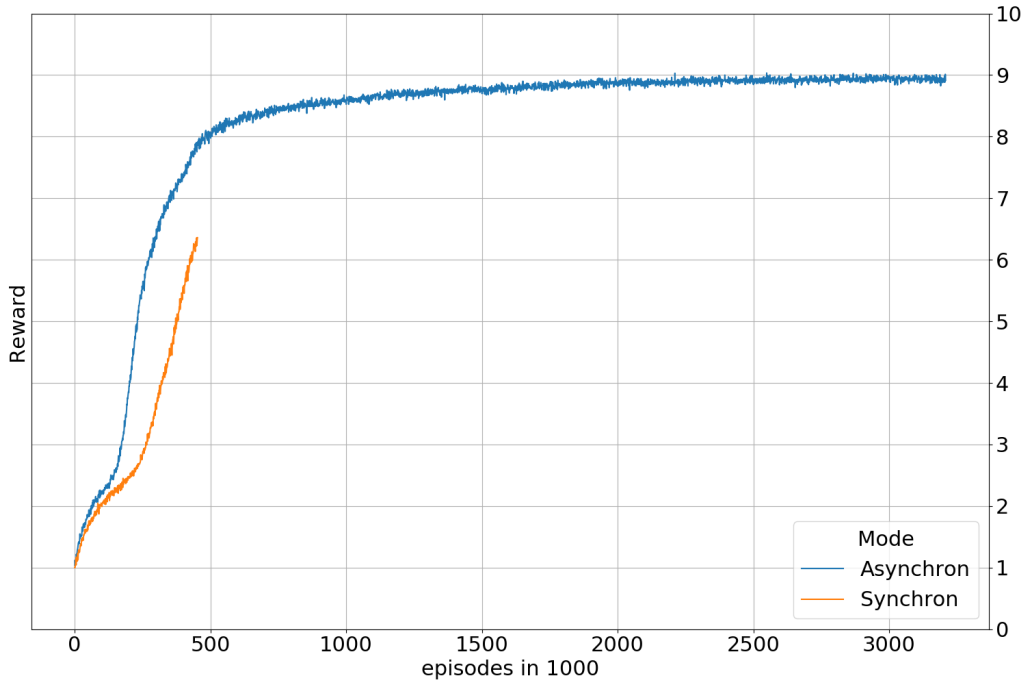


Abbildung 4.5: Vergleich 24 paralleler Instanzen mit asynchroner und synchroner Lernmethode anhand der durchgeführten Episoden beim Erreichen des harten Endkriteriums.

Sehr deutlich werden die Unterschiede der zwei Modi in Abbildung 4.5 in der die durchgeführten Episoden gegenüber gestellt werden. Hierbei kommt der synchrone Modus gerade mal auf 500.000 Episoden, wohingegen der asynchrone Ansatz nach Ablauf der 48 Stunden $3.250.000$ Episoden durchläuft. Der durchschnittliche

Reward ist beim asynchronen Ansatz logischerweise höher, da das Lernproblem nach 48 Stunden deutlich gegen ein lokales Optimum konvergiert. Beim synchronen Lernlauf sieht man deutlich, dass das System sich noch im Lernprozess befindet und der Reward mit der Zeit noch weiter steigen würde.

Anzumerken ist auch, dass das Verhältnis zwischen Updates und durchgeführter Episoden nicht übereinstimmt. Dies liegt an der künstlichen verkleinerten Anzahl an Cycles die PPO2 pro Update einsammelt, da eine Dummy-Observation auch als normaler Cycle mit validen Werten angesehen wird. Somit wird ein Update mit einem gewissen Anteil an nicht validen Werten durchgeführt.

Beispielsweise wurden somit für das erste Update des Lernlaufs 20.136 Cycles mit Ergebnissen und 29.016 Dummy-Elemente gesammelt. Daraus ergibt sich im Durchschnitt ein Verhältnis von Values zu Dummys von 40/60%. Durch diesen Einschnitt in der gesammelten Daten muss im Folgenden verifiziert werden, ob womöglich diese Dummy-Elemente ein Effekt auf das Endergebnis eines solchen Lernlaufs haben.

4.2.3 Einfluss der Dummy-Values auf das Lernen

Um den Einfluss der Dummy-Observations auf den Lernfortschitt beim KickWalk-Problem zu evaluieren, wurde ein Threshold-Mode implementiert, der eine bestimmte Prozentzahl an validen Observations sicherstellt. Dieser Prozentsatz kann vor dem Start eines Lernlaufs gesetzt werden.

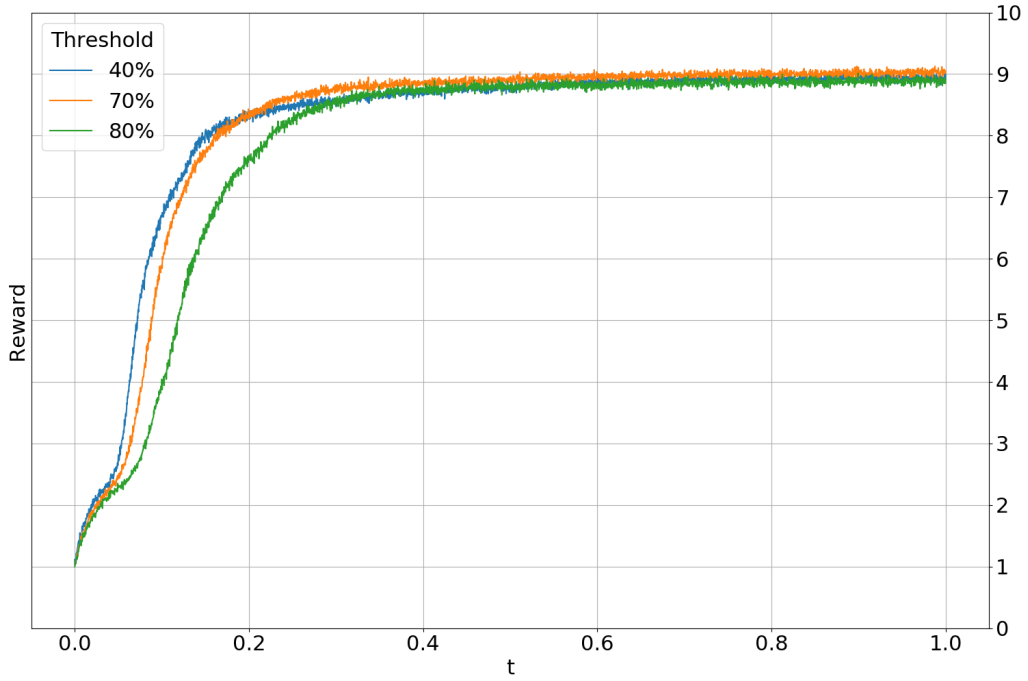


Abbildung 4.6: Vergleich unterschiedlicher Grenzwerte für die Mindestanzahl der validen Observations eines Updatezyklus entlang der Zeit und des durchschnittlichen Rewards der letzten 1000 Episoden.

Abbildung 4.6 zeigt die Ergebnisse der zwei zusätzlich gestarteten Lernläufe im Vergleich vom Standardverhalten des asynchronen Modus. Hierbei wurde 70% bzw. 80% an validen Werten sichergestellt. Wie deutlich zu sehen ist, hat ein kleinerer Anteil an Dummy-Elementen keinen Einfluss auf das lokale Optimum welches vom Lernlauf gefunden wird. Jedoch wird in der Anfangsphase bei steigender Synchronisation der Lernfortschritt etwas abgebremst. Auch wenn dies auf das Endergebnis keinen Einfluss nimmt.

Dennoch muss beachtet werden, dass die gesammelten Informationen für ein Update mit der Zeit variieren können. Bei verschiedenen Lernproblemen kann zu Beginn, also wenn der Agent noch nichts Sinnvolles ausführen kann, der Overhead höher sein als gegen Ende des Lernens. Beim derzeitigen Stand des asynchronen Modus ist dies nicht limitiert.

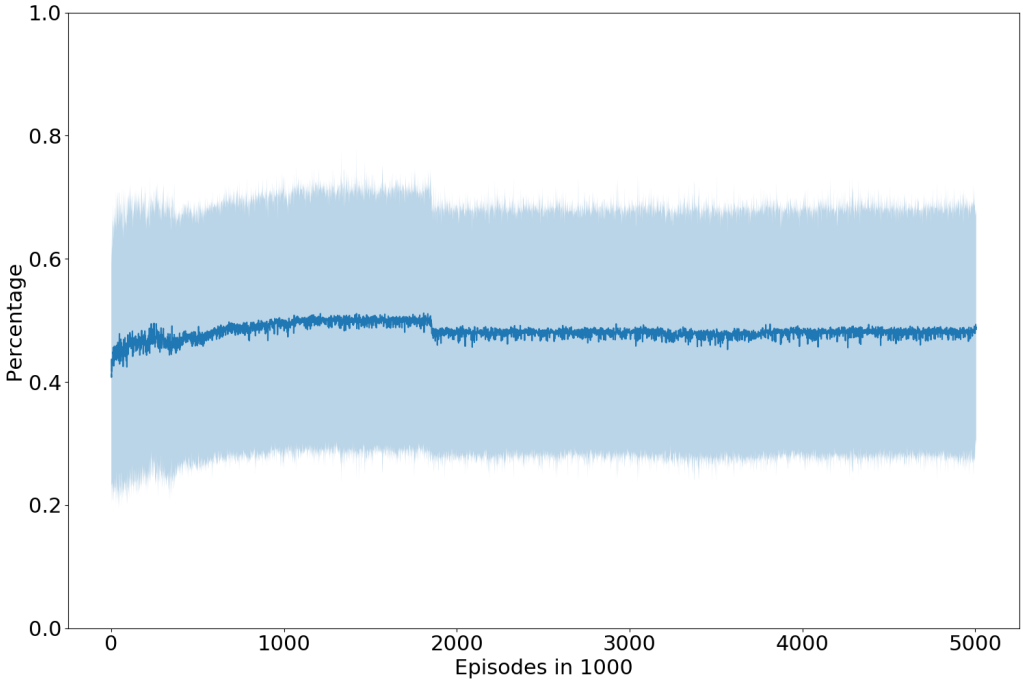


Abbildung 4.7: Durchschnittlicher prozentualer Anteil an validen Observations eines Update Zyklus während der asynchronen Lernmethode entlang der durchgeführten Episoden.

Abbildung 4.7 zeigt die Anzahl der validen Observations während eines Zyklus. Dabei trainierten 24 Threads parallel auf einem gemeinsamen Netzwerk. Der Graph zeigt den prozentualen Anteil an validen Observations gegenüber Dummy-Werten entlang der Zeit. Hier wurde immer über 1000 Zyklen mit 24 Observations der Durchschnitt und die Standardabweichung berechnet. Wie zu sehen ist, schwankt dieser Anteil von validen Werte sehr stark. Dabei kann es häufig vorkommen, dass bei einem Zyklus der Observation nur 20% der eigentlichen Werte bei PPO ankommen.

Der Lernfortschritt wird nur geringfügig beeinflusst, jedoch das Endergebnis kann durch eine künstlich verkleinerten Horizont signifikant verschlechtert werden. Ergebnisse mit einem kleineren Horizont konvergieren zwar früher, die gefundenen Ergebnisse sind jedoch nicht so gut wie mit einem größeren Horizont. Beim asynchronen Modus wird zu Beginn der Kick theoretisch nur mit 40% der Daten trainiert, welche mittels Hyperparameter festgelegt wurden. Dies könnte zu unerwünschten Endergebnissen führen und das Lernen nicht nachvollziehbar machen.

4.2.4 Weitere Optimierung und Ergebnisse

Deshalb wird der vorher implementierte Ansatz weiter verfeinert, um diese Problematik zu lösen. Beim alten Ansatz wartet die Python-Anwendung auf den ersten Agenten für die gesammelte Observation. Wie in Abbildung 4.7 zu sehen, hat sich beim Ausführen des alten asynchronen Modus der Anteil von validen Observations auf ca. 40% gemittelt. Der Rest wird mit Dummy-Elementen aufgefüllt. Dabei wird immer auf die erste Instanz gewartet und danach alle anderen Instanzen nach Ergebnissen abgefragt. Es hat sich bei weiterer Analyse der gesammelten Daten herausgestellt, dass der erste Agent dabei häufig auch der Langsamste oder der Schnellste sein kann. In diesen zwei Fällen wird entweder der asynchrone Modus aufgehoben, da auf die eine sehr langsame Instanz gewartet wird welche derzeit mit Overhead-Zyklen beschäftigt ist. Des Weiteren kann diese Instanz auch eine der Schnellsten sein welche Ergebnisse bereitstellt. Infolgedessen kann es dazu kommen, dass die zurückgegebenen Observations sehr viele Dummy-Values enthalten. Die im Weiteren angewandten Veränderungen lösen dieses Problem, in dem sie nicht immer auf die erste Instanz warten, sondern auf alle gleichzeitig. Sobald ein gewisser Grenzwert erreicht wurde, werden die Ergebnisse von den parallel laufenden Instanzen angefordert. Hier ist ein gewisser Grenzwert an Dummy-Observations definitiv sichergestellt. Somit kann es nicht mehr vorkommen, dass eine zurückgegebene Sammlung an Observations nur aus nicht relevanten Dummy-Werten besteht.

Im Folgenden werden die Ansätze wieder gegenübergestellt, um den Durchsatz der einzelnen Implementierungen zu vergleichen und den Vorteil der neuen Herangehensweise zu verdeutlichen.

	Synchron	asynchroner Protoyp	Asynchron
Episoden in 20 Minuten	5644	18627	26136

Tabelle 4.3: Gegenüberstellung des Durchsatzes der verschiedenen Herangehensweise anhand den durchschnittlich durchgeführten Episoden in 20 Minuten.

Wie die Ergebnisse in Tabelle 4.3 zeigen, haben die Verbesserungen zu einer signifikanten Steigerung der durchgeführten Episoden in 20 Minuten geführt. Dabei muss jedoch beachtet werden, dass die Synchronisation der beiden Ansätze unterschiedlich ist. Bei der verbesserten Version werden mindestens 40% an Observations gesammelt, wohingegen beim alten Modus im Durchschnitt 40% gesammelt werden. Dies spricht zusätzlich für die Effizienz der neuen Implementierung.

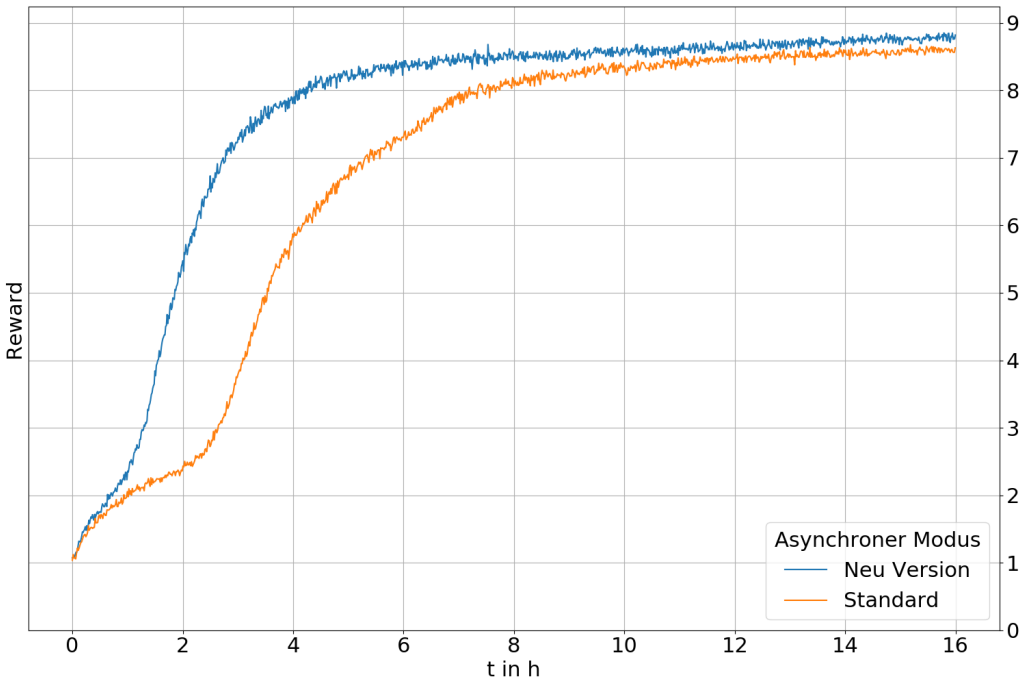


Abbildung 4.8: Vergleich des neuen asynchronen Modus gegenüber des alten Prototyps anhand des durchschnittlichen Rewards entlang der ersten 16 Stunden.

Abbildung 4.8 zeigt die Ergebnisse der ersten 16 Stunden zweier Lernläufe mit denselben Hyperparametern und demselben Threshhold. Hierbei wieder bei der alten Version im Durchschnitt und bei der neuen Version mindestens 40% an validen Observations pro Zyklus. Es ist deutlich zu sehen wie der neue Modus in den ersten Phasen schneller lernt und früher auf ein etwas besseres Ergebnis kommt. Dies liegt zum einen am erhöhten Durchsatz, sowie des größeren Horizonts während eines Updates in den frühen Phasen. Die neue Version hat nach 16 Stunden Laufzeit etwa 923.000.000 Episoden durchgeführt, wohingegen der alte asynchrone Modus nur 780.000.000 ausführen konnte.

Zusätzlich kann die neue Variante dazu genutzt werden, um verschiedene Grenzwerte für die Asynchronität festzulegen. Beim Kickwalk-Problem hat die große Anzahl an Dummy-Obervations und die künstliche Verkleinerung des gesammelten Horizonts nicht zu einem schlechteren Endergebnis geführt, jedoch kann dies von Lernproblem zu Lernproblem unterschiedlich sein, weshalb eine solche Funktionalität in Zukunft dennoch nützlich sein kann.

Grenzwert	Durchschnittliche Episoden in 20 Minuten
40%	26136
70%	20781.6
80%	16769.5
90%	13442.6
100%	5643.8

Tabelle 4.4: Durchsatz der asynchronen Implementierung anhand unterschiedlicher Grade an Synchronisation. Dabei ist für jeden Grenzwert die durchschnittlichen durchgeführten Episoden in 20 Minuten gegeben.

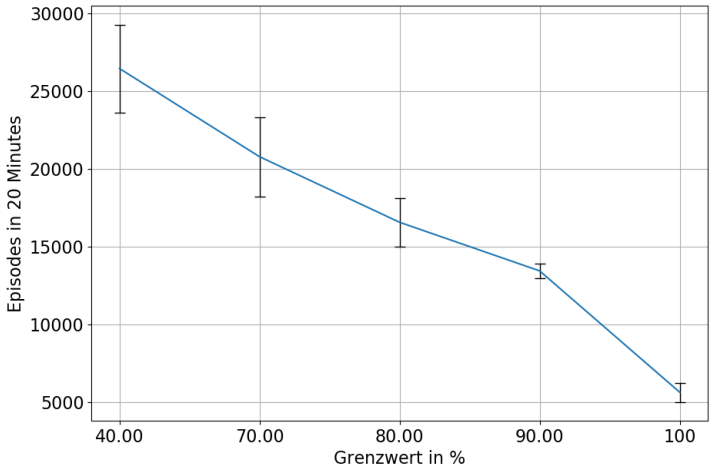


Abbildung 4.9: Durchgeführte Episoden in 20 Minuten entlang der Synchronisation in %

Um diese Funktionalität zu testen, wurden unterschiedliche Grenzwerte definiert und anhand der durchschnittlichen Episoden in 20 Minuten gegenüber gestellt. Tabelle 4.4 zeigt, dass die Ergebnisse auch bei einem höheren Grad der Synchronisation noch akzeptabel sind. Bei einer Synchronisation von 100% liegen die Werte wieder bei einem kompletten synchronen Ansatz und dementsprechend auch bei den Ergebnissen von Tabelle 4.2. Der große Sprung des Durchsatzes von 100% zu 90% zeigt, dass die Problematik des synchronen Lernens an nur wenigen blockierenden Instanzen liegt. Bei einem Grenzwert von 90% kann trotzdem eine signifikante Steigerung des Durchsatzes erreicht werden, wobei die Anzahl der gesammelten validen Observations nur geringfügig verringert wird. Ein Kompromiss den man für einen in etwa dreimal so hohen Durchsatz eingehen kann.

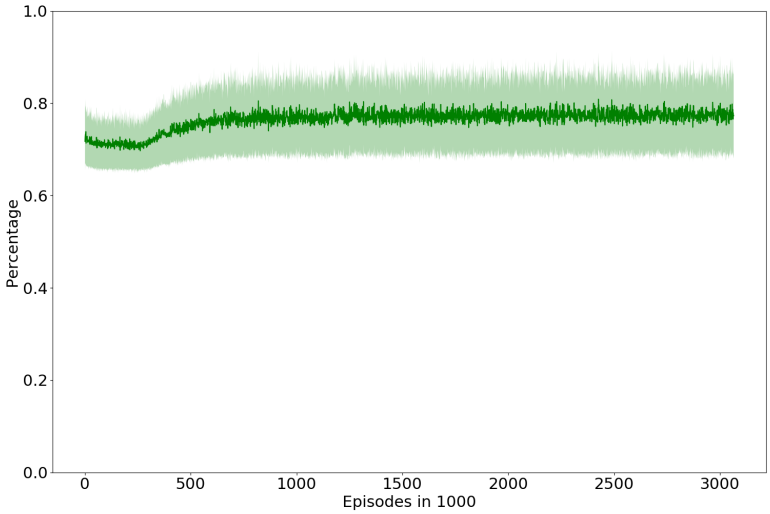


Abbildung 4.10: Durchschnittlicher prozentualer Anteil an validen Observations eines Update Zyklus während der asynchronen Lernmethode entlang der durchgeführten Episoden mit einem Grenzwert von 75%.

Durch diesen festgelegten Grenzwert verändert sich demnach auch der prozentuale Anteil der validen Observations entlang des Lernfortschritts. In Abbildung 4.10 wurde der Lernlauf von Abbildung 4.7 mit einem Grenzwert von 75% wiederholt. Dabei kann man gut erkennen wie die Standardabweichung des gesammelten Anteils deutlich kleiner geworden ist. Bei der Standardabweichung muss jedoch beachtet werden, dass der prozentuale Anteil nie kleiner des festgelegten Grenzwertes sein kann.

Die Evaluierung des asynchronen Modus hat deutlich gezeigt, dass durch dessen Verwendung die gewünschten Ergebnisse schneller und effizienter erreicht werden können. Die vorher beschriebenen Verbesserungen und die Probleme des asynchronen Modus haben sich erst im Laufe der Experimente herausgestellt, weshalb ein Großteil der Lernläufe mit dem alten asynchronen Modus ausgeführt wurden. Dies beeinflusst jedoch nicht die Erkenntnisse und Ergebnisse der folgenden Kapitel.

4.3 Hyperparameter

Die verwendeten Hyperparameter sind für den Erfolg und das Ergebnis eines Lernlaufs sehr entscheidend. PPO2 ist sehr benutzerfreundlich und preist eine sehr einfache Optimierung, sowie gute Ergebnisse mit dem Standardparameterset an. Im Folgenden werden daraufhin alle Hyperparameter, welche in den Grundlagenkapiteln

vorgestellt wurden, anhand dem KickWalk-Problem evaluiert und deren Einfluss auf den Lernfortschritt diskutiert und erläutert.

Bei der Evaluierung wird für die ersten zwei Hyperparameter (Lernrate, Nsteps) die sequentielle Herangehensweise angewandt. Die darauf folgenden Experimente bauen dann jedoch auf dem Prototyp des asynchronen Ansatzes auf, um die Endergebnisse in einem akzeptablen Zeitrahmen darzubieten.

Bei der Evaluierung wird jeder Lernlauf anhand des Kickwalk-Problems in einem festen Zeitfenster von 48 Stunden gegenüber gestellt. Die maßgebliche Größe ist der durchschnittliche Reward der letzten 1000 durchgeführten Episoden. Dabei werden alle Experimente von einem gemeinsamen Standard-Hyperparameterset gestartet, welches beim Startpunkt der Thesis von Team Magma festgelegt wurde:

Hyperparameter	Thesis-Standard	Stable-Baselines-Standard
learning_rate	0.0003	0.00025
n_steps	2048	128
nminibatches	32	4
episodes	10	4
lambda λ	0.95	0.95
gamma γ	0.99	0.99
ent_coef	0.0	0.01
vf_coef	0.5	0.5
cliprange	0.2	0.1

Tabelle 4.5: Standard Hyperparameterset der Thesis im Vergleich zu der Hyperparameterkonfiguration von Stable-Baselines.

4.3.1 Lernrate

Die Lernrate ist der grundlegendste Hyperparameter, weshalb bei der Evaluierung damit gestartet wird. Die Lernrate definiert wie stark sich die Gewichtungen innerhalb eines Netzwerks anhand des errechneten Werts der Fehlerfunktion anpassen. Im Folgenden wurden daher verschiedene Werte für die Lernrate verwendet und anhand dem sequentiellen Ansatz gegenübergestellt.

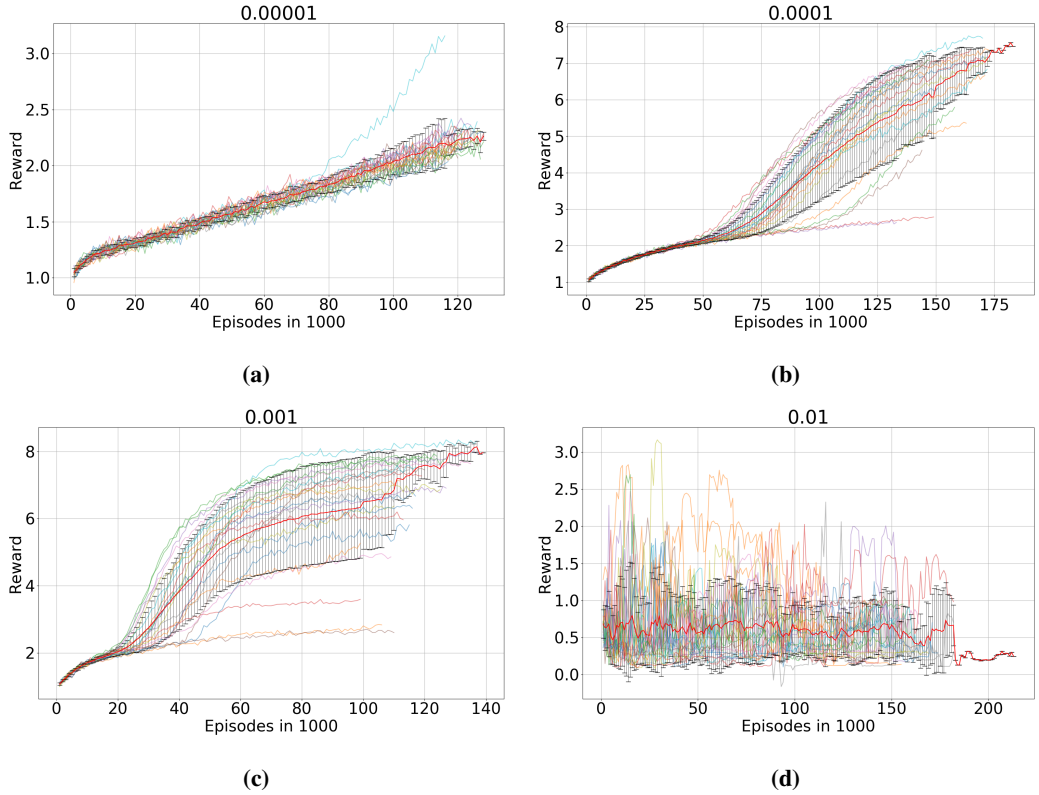


Abbildung 4.11: Vergleich unterschiedlicher Lernraten anhand der sequentiellen Lernmethode und 24 Instanzen. Dabei ist der durchschnittliche Reward entlang den durchgeführten Episoden dargestellt. Die rote Linie beschreibt den durchschnittlichen Reward aller Instanzen und dessen Standardabweichung.

In den Abbildungen 4.11 sind die Ergebnisse der vier verwendeten Werte zu sehen. Bei diesen Werten lässt sich der Einfluss der Lernrate auf das Lernen beobachten. Bei einer sehr kleinen Lernrate wie in Abbildung 4.11a lernt das System nur sehr langsam da sich das Netzwerk pro Update nur minimal oder zu wenig anpasst. Die Varianz der Läufe ist jedoch sehr konstant und alle bleiben in einem gewissen Rahmen. Nur ein Lauf spaltet sich ab und erreicht deutlich bessere Fortschritte als der Durchschnitt. Dennoch sind die Ergebnisse nach einer Laufzeit von 48 Stunden bei einer so kleinen Lernrate nicht zu gebrauchen.

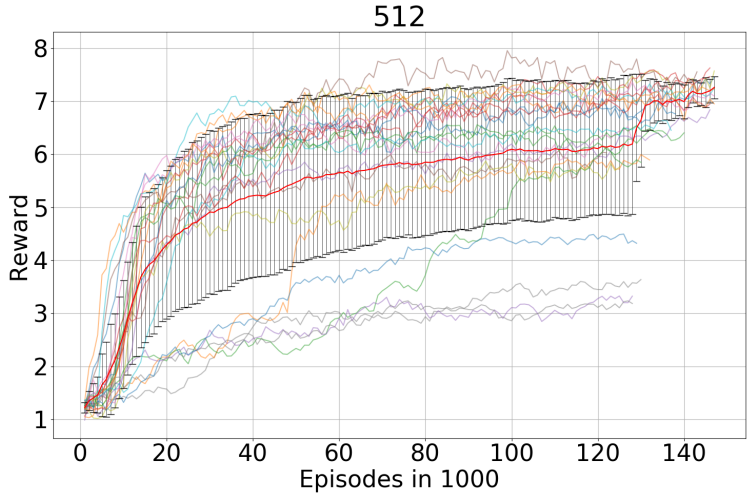
Erhöht man die Lernrate wie in Abbildung 4.11b oder 4.11c kann man signifikante Lernfortschritte entlang der Zeit erkennen. Die Ergebnisse von einer etwas größeren Lernrate von 0.001 liefern nach Ablauf der Zeit die deutlich besseren Ergebnisse mit einem durchschnittlichen Reward von etwa 8. Hierbei ist jedoch anzumerken, dass die Varianz der sequentiellen Läufe sehr hoch ist. Wie in allen Läufen zu erkennen ist, schreiten alle Instanzen unabhängig voneinander fort, weshalb manche zurückbleiben und überhaupt keine Lernerkenntnisse erreichen. Dies ist jedoch auch ab-

hängig von der Lernrate. Im Vergleich der Lernrate von 0.0001 und 0.001 ist die Varianz der größeren Lernrate deutlich höher.

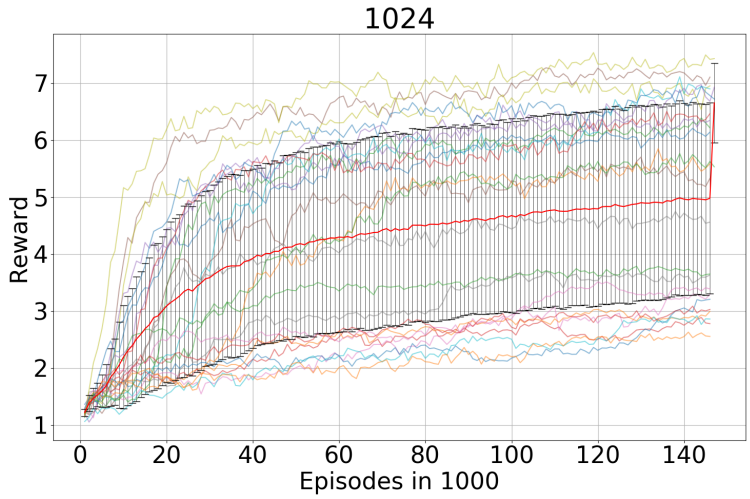
Bei einer zu großen Lernrate, wie in Abbildung 4.11d, können keine brauchbaren Ergebnisse erreicht werden. Die Änderungen am neuronalen Netzwerk werden dabei von Update zu Update zu groß, weshalb ein Optimum nie erreicht werden kann.

4.3.2 nSteps

Eine weitere interessante Größe ist die Anzahl der gesammelten Zyklen, welche für ein Update in Betracht gezogen werden. Mit der nSteps-Größe wird nicht die Anzahl der durchgeführten Episoden definiert, sondern der Anzahl der einzelnen Zyklen welche durchgeführt wurden. Beim sequentiellen Ansatz ist diese Größe sehr entscheidend für das erreichte Endergebnis nach dem festen Zeitlimit von 48 Stunden.



(a)



(b)

Abbildung 4.12: Vergleich unterschiedlicher Nsteps-Größen anhand der sequentiellen Lernmethode und 24 Instanzen. Dabei ist der durchschnittliche Reward entlang den durchgeführten Episoden dargestellt. Die rote Linie beschreibt den durchschnittlichen Reward aller Instanzen und dessen Standardabweichung.

Abbildung 4.12 zeigt den Vergleich zwischen der Nsteps-Größe von 512 und 1024 anhand derselben Anzahl an Episoden. Bei dieser Gegenüberstellung ist deutlich der Einfluss dieser Größe zu erkennen. Bei 4.12a werden pro Update des Netzwerks und dessen Policy weniger Episoden ausgeführt. Dabei werden 512 Zyklen pro Update gesammelt, was in etwa 30 durchgeführten Kicks entspricht. Anhand diesen gesammelten Daten wird daraufhin das System verbessert und in eine bestimmte Richtung gelenkt. Bei einer kleineren Nsteps-Größe sieht man deutlich, dass der

durchschnittliche Reward (rote Linie) etwas höher liegt als bei 1024 gesammelten Zyklen pro Update. Dies liegt daran, dass deutlich mehr der sequentiellen Instanzen Fortschritte machen und nur ein Paar zurückbleiben. Der eigentliche Einfluss dieser Größe wird jedoch erst beim parallelen Lernen und dem asynchronen Ansatz deutlich.

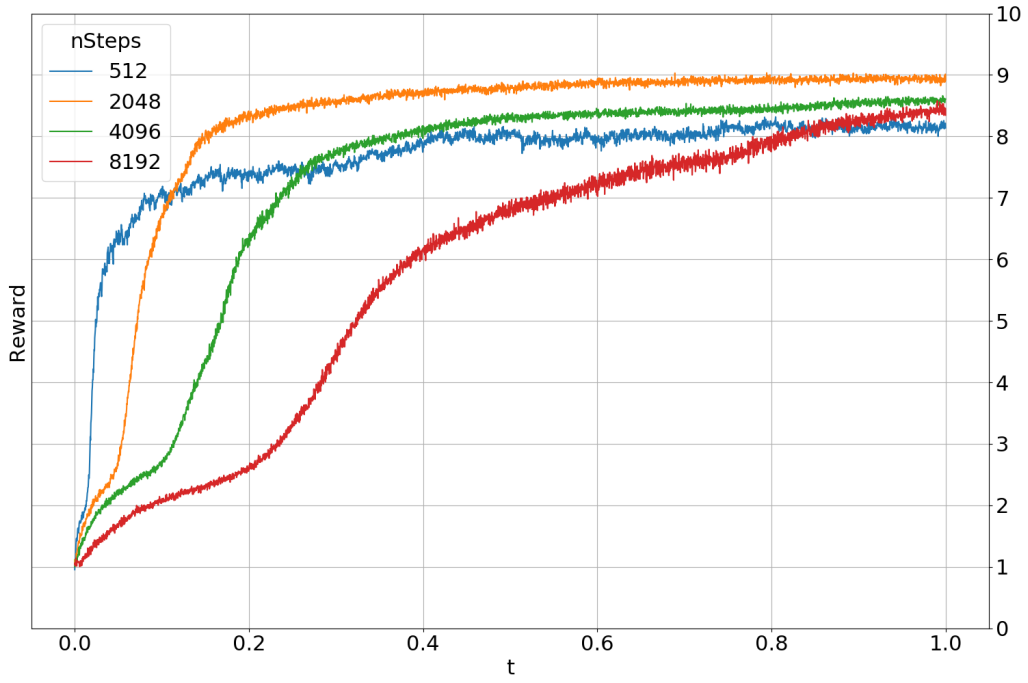


Abbildung 4.13: Vergleich unterschiedlicher nSteps-Größen anhand des KickWalk-Problems mit asynchroner Lernmethode. Dabei ist der durchschnittliche Reward entlang der Zeit bis zum Endkriterium von 48 Stunden dargestellt.

In Abbildung 4.13 sind die Endresultate verschiedener nSteps-Größen entlang des 48 Stunden Zeitfenster gegenüber gestellt. Durch die Verwendung des asynchronen Modus steigt der Durchsatz deutlich an, womit im festgelegten Zeitraum eher das Konvergieren zu einem lokalen Optimum beobachtet werden kann. Die Ergebnisse unterscheiden sich stark im Vergleich von synchronen zu asynchronen Modus. Beim synchronen Modus hat die nSteps-Größe von 512 die besten Ergebnisse geliefert, beim asynchronen Modus die Größe von 2048. Jedoch kann man in Abbildung 4.13 deutlich erkennen, dass der Lernlauf mit 512 ziemlich früh Fortschritte macht. Alle Endergebnisse nach dem 48 Stunden Zeitfenster sind signifikant besser als die vergleichbaren sequentiellen Läufe. Bei der nSteps-Größe lässt sich jedoch kein einheitliches Muster erkennen und es gilt nicht: je mehr gesammelte Daten pro Update, desto besser. Dies könnte mit der Korrelation zwischen diesem und anderen

Hyperparametern zusammenhängen. Dennoch steigt die Zeit pro Update linear mit der Größe des Horizonts, da mehr Zyklen pro Update gesammelt werden müssen.

4.3.3 Minibatch

Im Folgenden werden unterschiedliche Werte für die Minibatchanzahl verwendet. Hierbei soll beobachtet werden, wie sich die Größe auf das Endergebnis auswirkt und ob ein Muster erkannt werden kann. Die Vorgehensweise des Minibatch Gradient Descent hat gezeigt, dass sie Vorteile von den grundlegenden Stochastic und Batch Gradient Descent kombiniert, was anhand diesem Experiment gezeigt werden soll. Zudem soll geklärt werden, ob eine kleinere Minibatchanzahl immer zu einem besseren Ergebnis führt und stets die Kleinstmögliche gewählt werden sollte [17]. Zusätzlich wird der Durchsatz des Lernproblems abhängig der Minibatchanzahl betrachtet.

Die Implementierung von PPO2 von Stable-Baselines ist bei der Dokumentation dieses Hyperparameters etwas ungenau. Theoretisch würde man beim Setzen des Minibatch Hyperparameter von 32 davon ausgehen, dass die gesammelten Daten in eine gewisse Anzahl an Minibatches der Größe 32 aufgeteilt werden. Dies ist jedoch nicht der Fall. Bei der Implementierung von Stable-Baselines wird der gesammelte Datensatz in 32 gleichgroße Teilabschnitte aufgeteilt. Der Hyperparameter ist nicht zuständig für die Größe, sondern die Anzahl der Minibatches welche pro Epoche angewandt werden sollen. Da die Nsteps-Größe abhängig der parallel laufenden Instanzen ist, ist somit die Anzahl der Minibatches auch abhängig von dieser Größe. Hierbei steht die Minibatchgröße für die Anzahl der Zyklen, welche sie enthält. Die Minibatchanzahl hingegen für die Anzahl der Minibatches auf welche der gesammelte Datensatz aufgeteilt wird. Ein Wert von 8 für den Hyperparameter nminibatches, bei einer Größe von 2048 des Datensatzes, liefert dabei nicht 256 Teilabschnitte des Datensatzes mit einer Größe von 8, sondern 8 Teile mit einer Größe von 256.

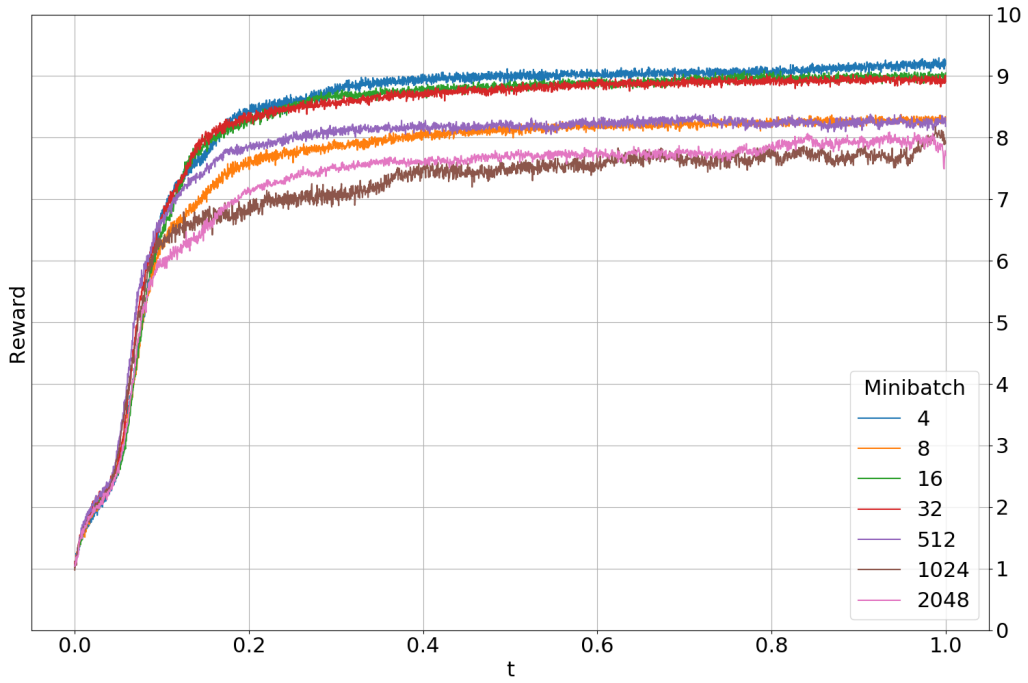


Abbildung 4.14: Vergleich unterschiedlicher Minibatch-Größen anhand des KickWalk-Problems mit asynchroner Lernmethode. Dabei ist der durchschnittliche Reward entlang der Zeit bis zum Endkriterium von 48 Stunden dargestellt.

Abbildung 4.14 zeigt die Ergebnisse der Lernläufe mit unterschiedlicher Minibatchgröße, wobei ein sehr weiter Wertebereich verwendet wurde. 32 liefert den Standardwert des Hyperparametersets dieser Thesis und die Minibatchgröße von 4 lieferte die besten Endergebnisse. Grundsätzlich bestätigt sich das Muster: je kleiner die Minibatchgröße, desto besser das konvergierende Endergebnis. Jedoch gibt es durch die allgemeine Varianz Ausnahmen. Die Minibatchgröße von 8 springt etwas aus dem Muster heraus und liefert mit einem durchschnittlichen Reward von 8,3 deutlich schlechtere Ergebnisse als eine Minibatchgröße von 32 mit einem durchschnittlichen Reward von 9. Dies scheint sehr unrealistisch für das erreichte Ergebnis, da normalerweise bei einer Minibatchgröße von 8 einen Reward im Bereich der Ergebnisse von von 4 bis 32 erwartet wird. Dies bestätigt, dass immer noch eine relativ hohe Varianz im Lernfortschritt vorhanden ist, weshalb alle Ergebnisse und Erkenntnisse kritisch zu betrachten sind.

Die Verschiebung der Anzahl an Minibatches pro Update verschiebt sozusagen das Verhältnis vom ursprünglichen Ansatz der SGD zu BGD. Hierbei ist jedoch zu beachten, dass eine Minibatchanzahl von 2048 nicht dem SGD Ansatz entspricht bei dem das Netzwerk mittels jedem gesammelten Datensatz geupdated wird. Die Mi-

nibatchgröße ist definiert durch die nSteps-Größe multipliziert mit den parallelen Instanzen, geteilt durch 2048. Somit werden bei dieser Minibatchgröße 24 einzelne Datensätze verwendet, um das Netzwerk in 2048 Schritten zu optimieren.

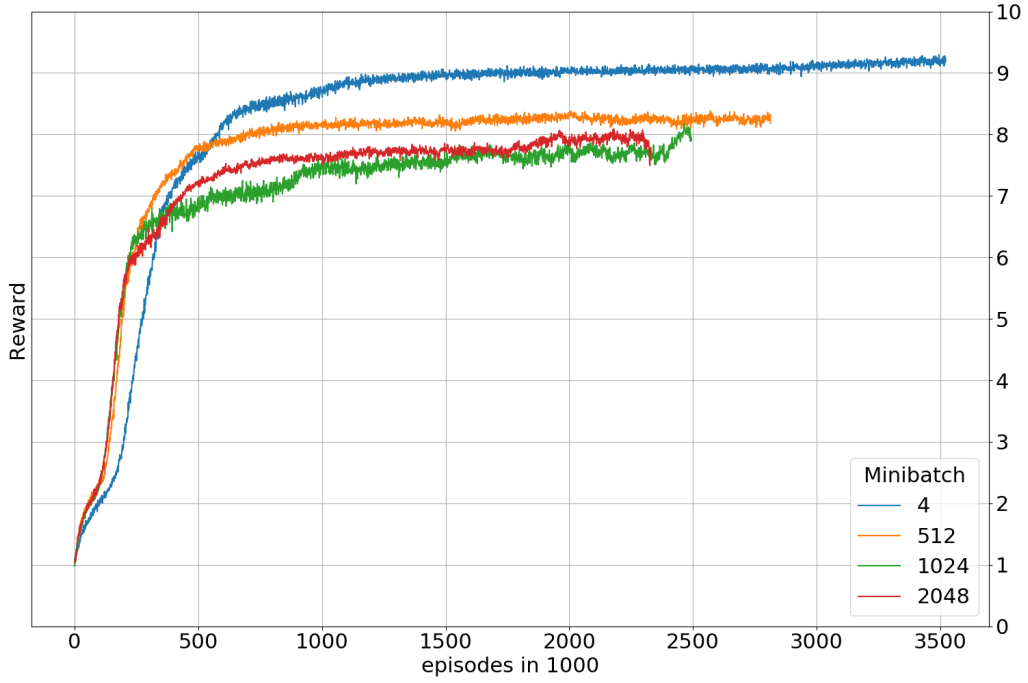


Abbildung 4.15: Vergleich unterschiedlicher Minibatchgrößen anhand des KickWalk-Problems mit asynchroner Lernmethode. Dabei ist der durchschnittliche Reward entlang der durchgeführten Episoden in 48 Stunden dargestellt.

In Abbildung 4.15 sind die Ergebnisse von Abbildung 4.14 anhand der durchgeführten Episoden gegenübergestellt. Alle Lernläufe haben exakt 48 Stunden das Netzwerk anhand des KickWalk-Problems trainiert. Hierbei zeigt sich die Annahme: je kleiner die Minibatchgröße, desto weniger Episoden konnten durchgeführt werden [22]. In den Ergebnissen kann man zudem erkennen, dass die Varianz der Ergebnisse größer wird, je mehr Minibatches pro Episode auf das Netzwerk angewandt werden. Dies liegt daran, dass die Updates des Netzwerks auf kleineren Datensätze basieren. Somit wird das Rauschen pro Update erhöht, welches deutlich in dem durchschnittlichen Reward von kleineren Minibatchgrößen sichtbar ist [23]. Auch die Hypothese, dass eine kleinere Minibatchgröße schneller zum Konvergieren der Ergebnisse führt, kann mittels den gesammelten Daten bewiesen werden. Wie zu sehen ist, wird die Anzahl der benötigten Episoden bis zum Erreichen des Endergebnis kleiner, je kleiner die Minibatchgröße ist [23]. Jedoch sind diese Un-

terschiede im Durchsatz in der ersten Lernphase minimal. Zudem sind die Endergebnisse bei größeren Minibatches deutlich besser.

Eine Problematik, welche durch den asynchronen Modus entstehen könnte, sind die schlechteren Endresultate bei einer größeren Minibatchanzahl aufgrund des Anteils an Dummy-Elementen. Bei einem durchschnittlichen Anteil an Dummy-Observations von 60% kann es vorkommen, dass der Datensatz eines Batches aus fast ausschließlich bedeutungslosen Werten besteht. Somit wird das Netzwerk aus einem noch kleinere Datensatz optimiert. Bei einer großen Minibatchgröße haben diese nicht relevanten Observations womöglich weniger Einfluss auf die Updates, da immer noch genügend valide Observations pro Update vorliegen.

Durch die vorliegenden Ergebnisse ist es jedoch von Vorteil beim KickWalk-Problem einen relativ kleinen Wert für die Anzahl der Minibatches zu wählen, da somit der Durchsatz sowie erreichtes Endergebnis gesteigert werden können.

4.3.4 Episoden

Die Anzahl der Episoden pro Update ist entscheidend für die maximale Informationsmenge, die aus einem gesammelten Datensatz extrahiert werden kann. Durch die Trust-Region Begebenheiten von PPO2 kann ein Datensatz mehrmals durchlaufen werden, da die neue Policy in einem gewissen Rahmen zur alten Policy bleiben muss. Der Standardwert hierfür sind 10 Episoden pro Update, weshalb der gesammelte Datensatz zehnmal für die Optimierung des Netzwerks verwendet wird.

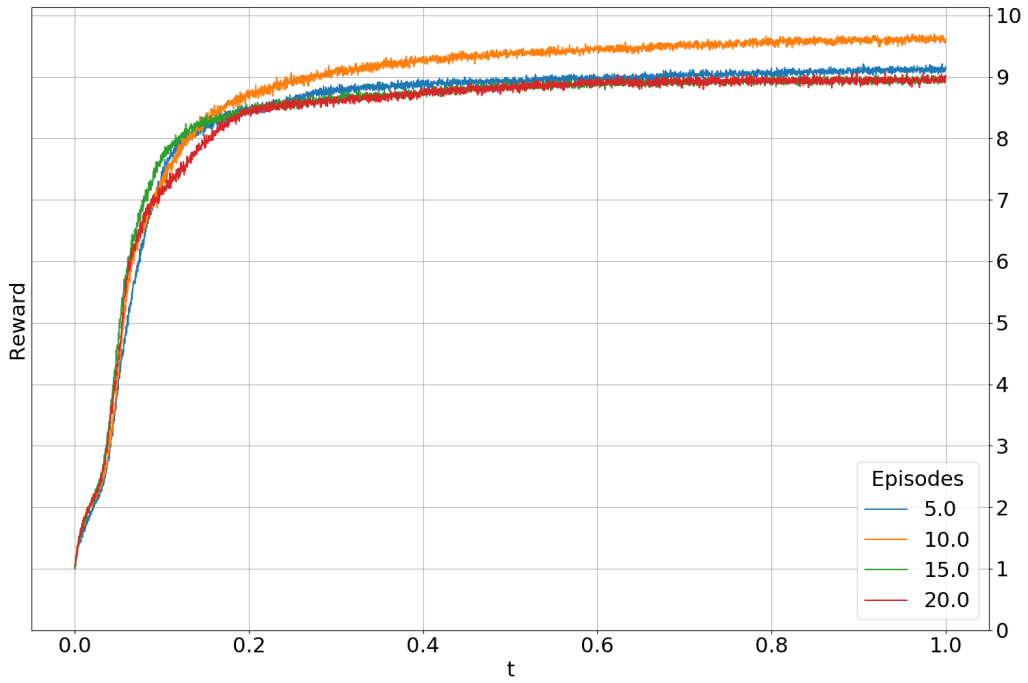


Abbildung 4.16: Vergleich unterschiedlicher Episodenzahl anhand des KickWalk-Problems mit asynchroner Lernmethode. Hier wird der durchschnittliche Reward entlang der durchgeführten Episoden in 48 Stunden dargestellt.

Wie Abbildung 4.16 zeigt, bleiben alle Ergebnisse in einem gewissen Bereich mit Ausnahme des Standardwerts von 10 Episoden. Die Größe der Episodenanzahl pro Update ist stark abhängig mit dem Wert des Clipping-Operators. Je mehr Episoden, desto wahrscheinlicher ist, dass die Clipping-Grenze von alter zur neuen Policy erreicht wird. Da hier in diesem Versuch der Standardwert 10 Episoden war, wird dieser Wert für den Standard-Clipping-Wert von 0.2 nahe am Optimum liegen. Bei einer Anzahl an Episoden kleiner als 10 werden die festgelegten Grenzen des Clippings selten erreicht, weshalb die Wichtigkeit von bedeutenden Aktionen nicht stark genug erhöht wird. Bei zu großen Episodenwerten hingegen wird vermutlich die Clipping-Grenze zu oft und auch von nicht bedeutenden Aktionen erreicht. Somit wird in beiden Fällen ein suboptimales Endergebnis erreicht, welches im durchschnittlichen Reward schlechter ist als der Standard-Episodenwert. Durch die hohe Korrelation des Clippings- sowie des Episoden-Hyperparameters, sollten diese gemeinsam optimiert werden, was jedoch den zeitlichen Rahmen dieser Thesis gesprengt hätte.

4.3.5 Entropy Coefficient

Der Entropy Coefficient ist ein Hyperparameter der einen positiven Einfluss auf das Exploration Verhalten von PPO haben soll. Hierbei wird wie im Grundlagenkapitel näher erläutert, ein gewisser Koeffizientenwert auf die Fehlerfunktion gerechnet, welcher das explorative Verhalten anregen soll. Die Policy hat eine maximale Entropie, falls alle Aktionen gleich wahrscheinlich sind. Somit wird das System dazu gebracht, eine hohe Entropie beizubehalten um nicht ständig dieselben Aktionen auszuführen um den größtmöglichen Reward zu erhalten. Falls also eine Aktion dominant wird, sinkt dementsprechend die Entropie und daraufhin auch der Bonus auf die Fehlerfunktion für die Erforschung neuer Aktionen. Hierbei können natürlich schlechtere Aktionen ausgeführt werden, jedoch auch Aktionen welche unerwarteterweise zu einem guten Reward geführt haben.

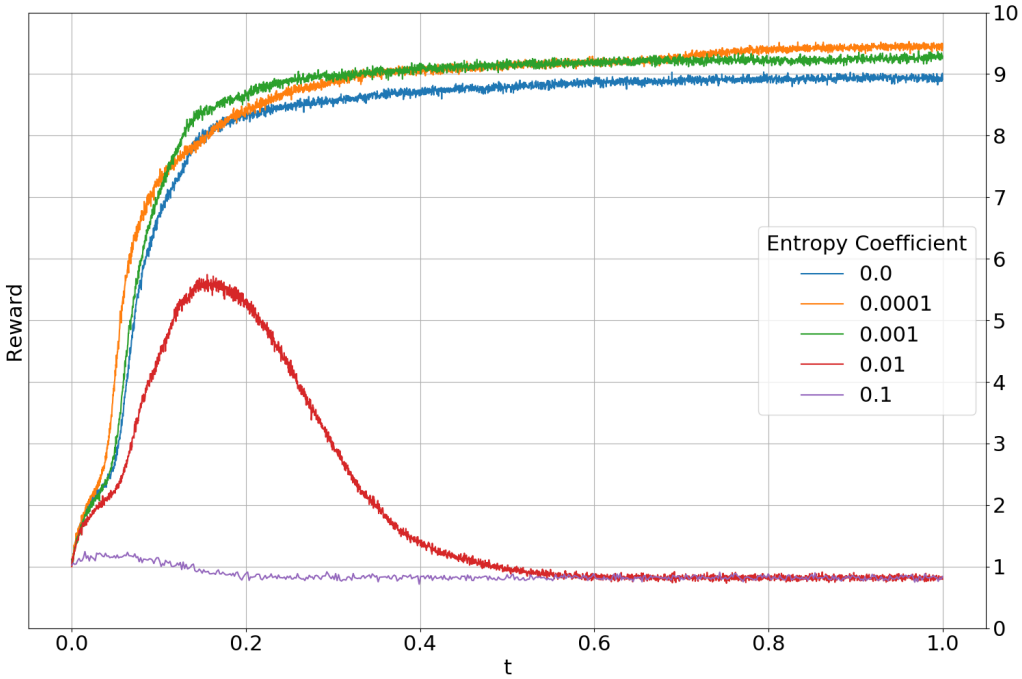


Abbildung 4.17: Gegenüberstellung unterschiedlicher Werte des Entropy Coefficient anhand des KickWalk-Problems mit asynchroner Lernmethode. Dabei ist der durchschnittliche Reward entlang der durchgeführten Episoden in 48 Stunden dargestellt.

Die Ergebnisse lassen jedoch nicht auf eine Steigerung der Exploration schließen. Wie in Abbildung 4.17 zu sehen ist, liegen die Endergebnisse von sehr kleinen Werten des Entropy Koeffizienten relativ nahe beieinander. Theoretisch würde man davon ausgehen, dass ein größerer Koeffizient die Exploration antreiben sollte und

somit bei der Konvergenz zu einem Endresultat neue Aktionen gefunden werden, da die Entropy hochgehalten wird [24]. Bei den Werten 0,0001 und 0,001 ist dies jedoch genau umgekehrt der Fall. Bei einem kleineren Wert konnte durch ein exploratives Verhalten eine neue Erkenntnis gewonnen werden, welche zu einem kleinen Anstieg im durchschnittlichen Reward geführt hat.

Der Standardwert für den Entropy Koeffizient liegt bei 0. Das bedeutet, dass die Entropy nicht in die Berechnung der Fehlerfunktion einbezogen wird. Jedoch konnte selbst durch einen kleinen Faktor des Koeffizienten das Ergebnis verbessert werden.

Dabei ist dennoch zu beachten, dass ein zu hoher Wert des Entropy Koeffizienten sehr kontraproduktiv für den Lernfortschritt sein kann. Dies ist bei Werten von 0,01 und 0,1 sehr gut zu sehen. Im Falle der roten Kurve lernt das System zwar bis zu einem gewissen Punkt, aber verlernt daraufhin wieder alles und kommt am Ende auf einen schlechteren durchschnittlichen Reward als zu Beginn. Dieses Verhalten kann mittels Verlauf der Entropie entlang des Lernfortschritts erklärt werden.

Der Standardwert für den Entropy Koeffizienten in der Implementierung von Stable-Baselines liegt bei 0,01. Durch solch einen Wert würde man wie in Abbildung 4.17 schlechte Ergebnisse erlangen. Jedoch kann dies von Lernproblem zu Lernproblem und vom eigentlichen Setup des Lernproblems unterschiedlich sein.

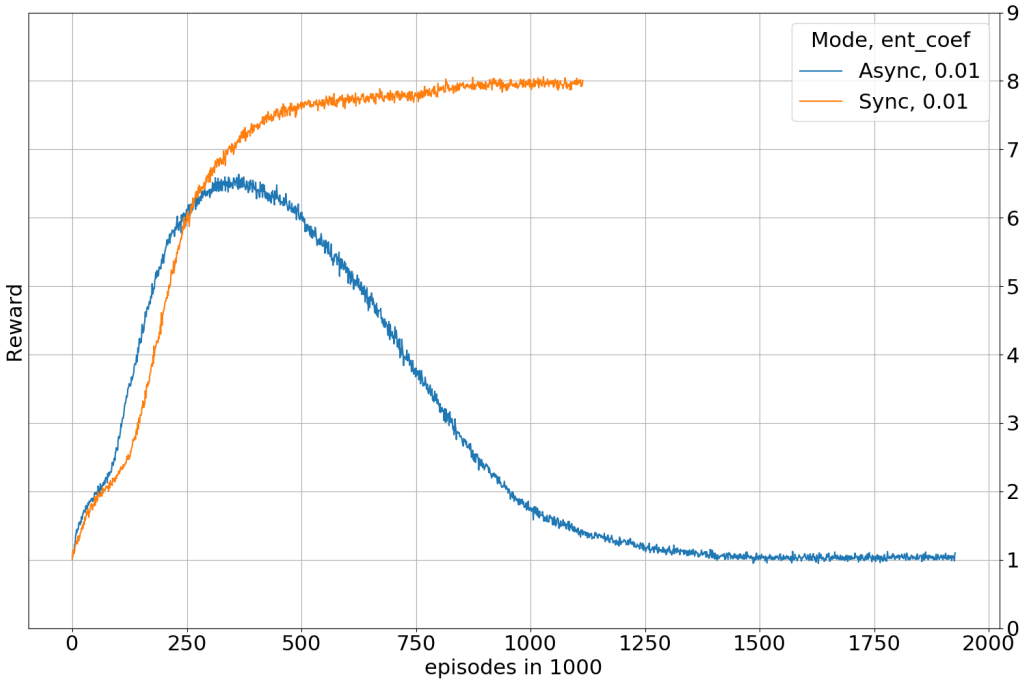


Abbildung 4.18: Vergleich des Lernfortschritts anhand Lernmodus mit gleichem Entropy Coefficient.

In Abbildung 4.18 ist ein Lauf im asynchronen Modus und im synchronen Modus gegeneinander gestellt. Beide haben einen Wert von 0.01 für den Entropy Koeffizienten und liefen auf 12 parallelen Threads. Jedoch lassen sich deutliche Unterschiede im Fortschritt sowie Endergebnis erkennen. Im synchronen Modus scheint der Koeffizient das Lernen nicht negativ zu beeinflussen, wie es beim asynchronen Ansatz der Fall ist. Dies lässt darauf schließen, dass auch durch das Setup des Lernproblems das Ergebnis der verschiedenen Hyperparameter auf das Lernen beeinflusst wird.

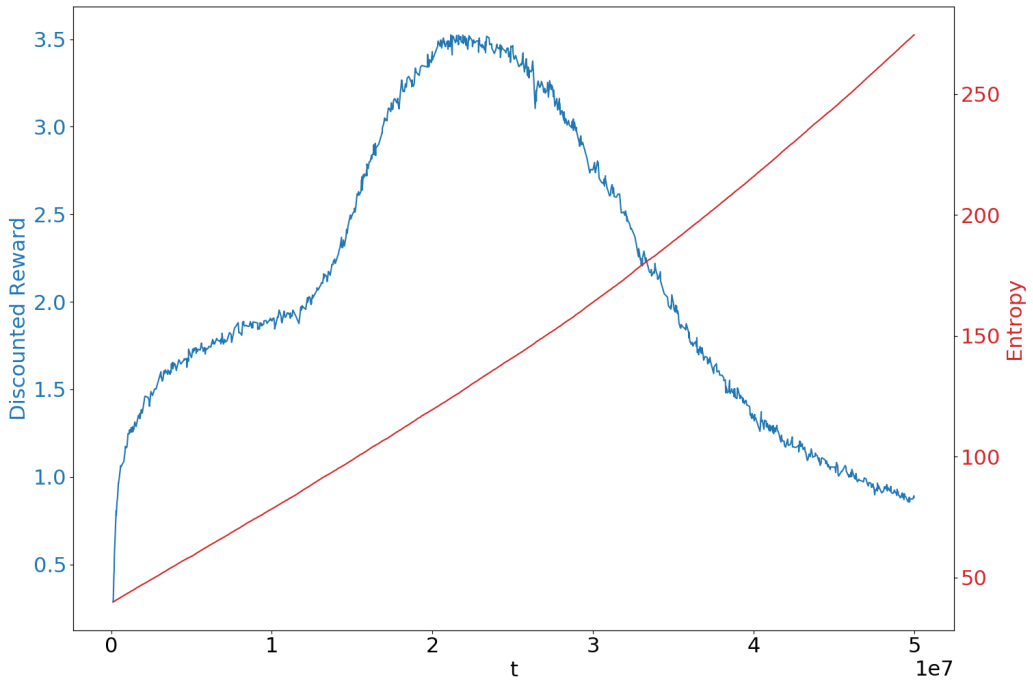


Abbildung 4.19: Gegenüberstellung des Discounted Rewards anhand der steigenden Entropy des Systems.

Das Verlernen, welches beim asynchronen Modus in Abbildung 4.18 zu sehen ist, sollte im Folgenden näher betrachtet werden. Normalerweise sinkt die Entropie entlang der Zeit, da sich der Agent immer sicherer wird welche Aktionen gut oder schlecht für den erhaltenden Reward sind. Bei einem zu großen Wert des Entropiebonus sinkt diese jedoch nicht, sondern steigt entlang der Zeit. Der Agent versucht alle Aktionen gleich oft auszuführen und die Entropie möglichst hochzuhalten, damit der Bonus für die Fehlerfunktion auch möglichst groß ausfällt. In Abbildung 4.19 sieht man, dass bei einer gewissen Grenze des Entropiewerts der Agent jeden Fortschritt verliert und der durchschnittliche Reward wieder absinkt.

Hier kann keine genaue Aussage für das "Verlernen"der Aktionen getroffen werden. Dennoch scheint es so, als wäre die Entropie ab einem gewissen Punkt so entscheidend für die Fehlerfunktion gegenüber dem Policy-Fehler, weshalb die Entropy das stärkere Mehrheitsrecht hat. Infolgedessen schlägt das System schlechte Aktionen vor, welche zu einem schlechteren Ergebnis führen und der Agent von Update zu Update erlernte Fortschritte verlernt.

Beim Hyperparametertuning empfiehlt es sich zuerst mit kleinen Werten des Koeffizienten zu beginnen oder ihn komplett zu deaktivieren, wie es bei dem Standard-

parameterset dieser Thesis der Fall ist. Der Entropy Coefficient stellt sich als ein einflussreicher Parameter heraus, der nicht nur das Endergebnis, sondern auch den gesamten Lernprozess beeinflussen kann.

4.3.6 Value Function Coefficient

Der Value Function Coefficient ist der zweite Part der vorgestellten Fehlerfunktion von PPO2 aus Formel 3.13. Dieser Koeffizient muss zwingend eingeführt werden, da es sich um ein neuronales Netz handelt welches Parameter zwischen dem Policy- und Value-Netzwerk teilt [14]. Dieser Parameter sollte bei PPO2 immer einen Wert von 0.5 haben um die besten Ergebnisse zu liefern. Dieser Wert ist auch der Standardwert über alle PPO2-Hyperparametersets von verschiedenen gängigen Lernproblemen.

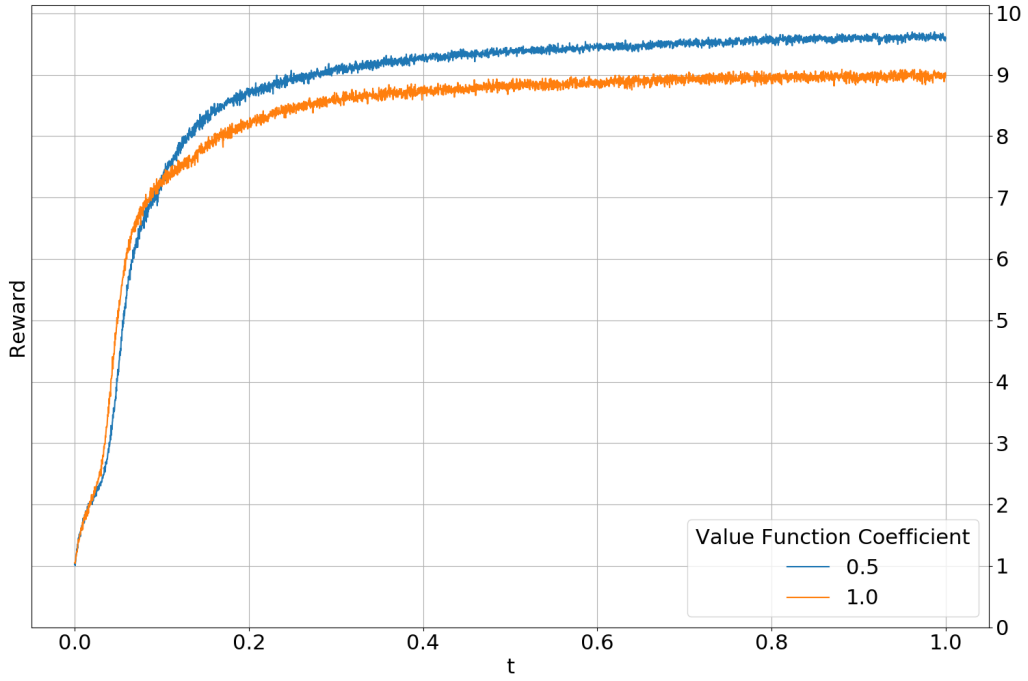


Abbildung 4.20: Vergleich Abweichung des Value Function Coefficients vom Standardwert.

Abbildung 4.20 zeigt die Ergebnisse, falls der Value Function Coefficient eine andere Gewichtung in der Fehlerfunktion erhält. Wie zu sehen ist, sind die Ergebnisse nach Erreichen des Endkriteriums schlechter als erwartet. Der Standardwert des Value Function Coefficient von 0.5 weist dabei bessere Ergebnisse auf. Dieser

Hyperparameter scheint ein Problem der zugrundeliegenden Implementierung des Algorithmus zu sein, weshalb es sich weniger lohnt ihn weiter zu untersuchen.

4.3.7 Gamma

Gamma γ ist der Abzinsfaktor, eingeführt im Grundlagenkapitel 3.1.6. Im Folgenden werden mehrere Werte im KickWalk-Problem ausgetestet und deren Einfluss auf das Lernen evaluiert.

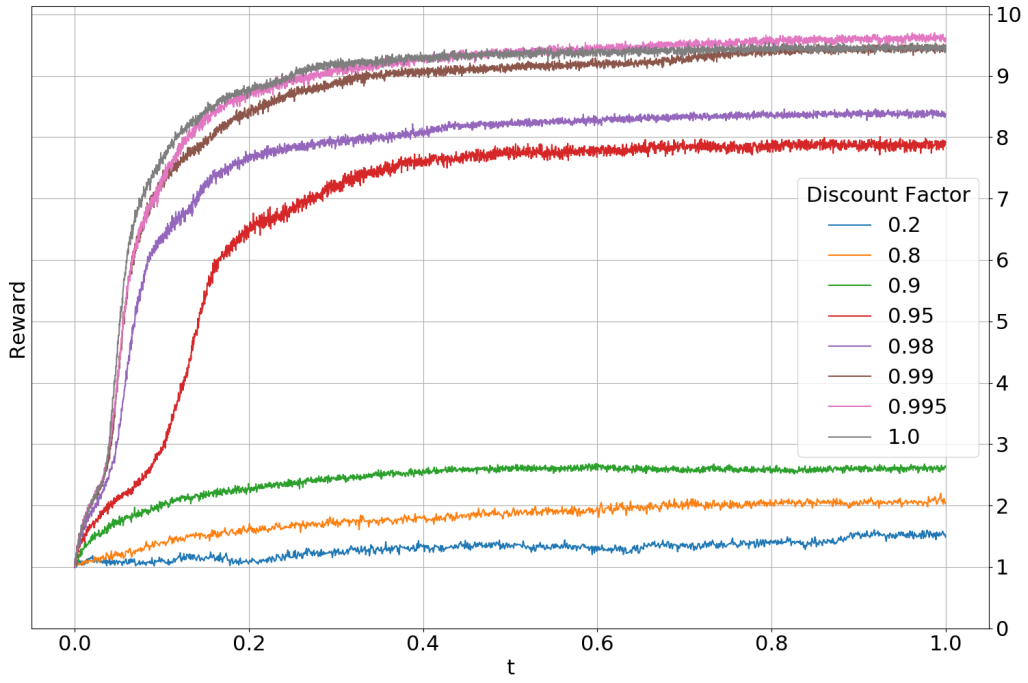


Abbildung 4.21: Vergleich für verschiedene Gamma-Werte anhand des Kickwalk-Problems und deren Ergebnisse nach 48 Stunden. Dabei ist der durchschnittliche Reward auf der y-Achse und die verstrichene Zeit auf der x-Achse abgebildet.

Wie Abbildung 4.21 zeigt, sind die Endergebnisse bei einem Wert von Gamma nahe bei 1.0 in einem recht ähnlichen Bereich, der durch die Standardabweichung des Lernfortschritts erklärt werden kann. Wird Gamma jedoch kleiner als 0.99, ist ein signifikanter Einbruch des erreichten Endergebnisses zu erkennen.

Ein Wert von Gamma nahe an 1.0 führt zu einem sehr weitsichtigen System. Je kleiner Gamma wird, desto kurzsichtiger wird der Agent und versucht unmittelbar mit der Wahl seiner Aktionen den größtmöglichen Reward zu erzielen. Beim Kickwalk-Problem ist die Bewegung auf eine feste Anzahl von 18 Cycles beschränkt. Dies ist

eine sehr kurze Ausführungszeit, weshalb die Ergebnisse von einem großen Wert von Gamma besser scheinen.

Zusätzlich gibt es beim Kickwalk-Problem nur einen Endreward, weshalb kleinere Werte von γ wenig Sinn ergeben. Die Kickbewegung wird meist durch die ersten Zyklen maßgeblich definiert, weshalb es sehr hinderlich für das Lernen ist, einen kleinen Gamma Wert zu wählen. Der Sprung vom Defaultwert von $\gamma = 0.99$ zu 0.98 senkt die Wichtigkeit des Endrewards für die ersten Zyklen um 10%.

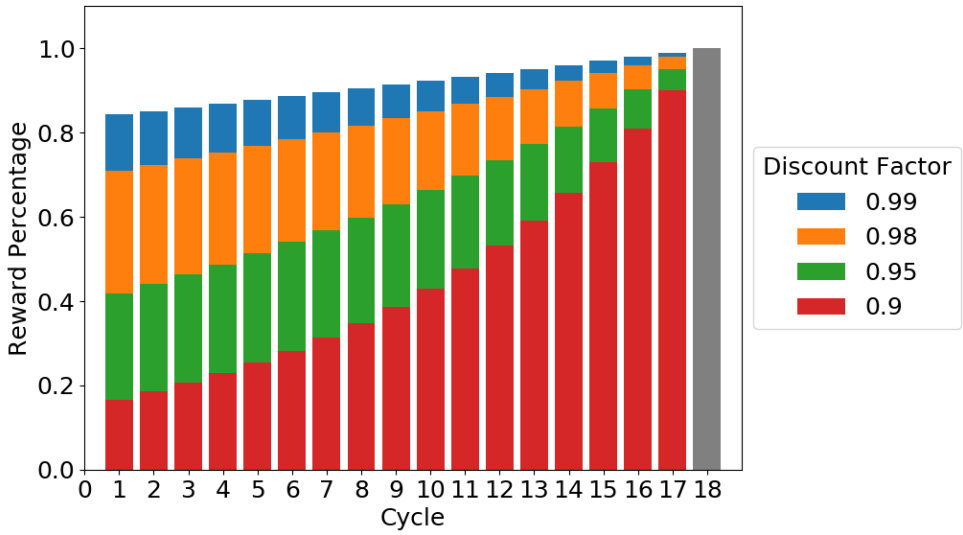


Abbildung 4.22: Visualisierung des Discount Factors und dessen Einfluss auf den erhaltenen Endreward.

Abbildung 4.22 veranschaulicht den Verfall des Endrewards und den Einfluss den er auf die frühen Aktionen hat. Es ist deutlich zu erkennen wie anhand eines kleineren Gamma-Wertes der zugeteilte Endreward der anfänglichen Aktionen sinkt. Wirft man nun einen Blick auf die vorher vorgestellten Ergebnisse der einzelnen Werte sieht man, dass sobald der Reward der ersten Aktionen zu unbedeutend wird das Netzwerk die Aufgabe kaum erfüllen kann. Bei Werten von $\gamma = 0.9$ ist es dem Agenten nahe zu unmöglich die Aufgabe hinreichend zu erfüllen, geschweige denn zu verstehen, womit er seinen erhaltenen Reward erhöhen kann. Selbst wenn der Nao per Zufall den Ball nach vorne kickt, wird er diese Erkenntnis nicht den richtigen Zyklen zuweisen können. Bei einem so kleinen Gamma-Wert werden die ersten Zyklen nur mit einer Gewichtung von 20% angesehen und fälschlicherweise die späteren Aktionen die ausgeführt wurden für diesen Reward verantwortlich ge-

macht. Deshalb lernt er ab dieser Gamma-Größe nur noch stehenzubleiben, um den Endreward dafür zu erhalten.

Die Ergebnisse dieses Testlaufs zeigen, dass bei einem Problem mit hartem Endkriterium und nur kurzer Laufzeit $\gamma = 1.0$ gesetzt werden sollte. Insbesondere falls die ersten Zyklen maßgeblich für die Bewegung sein könnte. Es ergibt wenig Sinn bei einem Kick die ersten Zyklen weniger für den Endreward zu gewichten, da diese maßgeblich für das Lernen sein sollten. Da bei einer festen Ausführungslänge von 18 Zyklen der Endreward bei einem Discount Factor von 0.99 in den ersten Aktionen noch maßgeblich sein kann, sind die Ergebnisse von Werten um diesem Standardwert relativ ähnlich und liegen in der allgemeinen Varianz des Lernproblems. Jedoch kann die Ausführungslänge von Lernproblem zu Lernproblem unterschiedlich sein und Gamma muss je nach Rewardfunktion überdacht werden, um plausible Werte zu setzen.

4.3.8 Lambda

Der Hyperparameter Lambda beschreibt das Verhältnis von Monte-Carlo- zu Temporal-Difference-Methode, wie in Kapitel 3.1.7 näher erläutert. Im Folgenden wird dieses Verhältnis verschoben, um die Auswirkungen auf das Voranschreiten des Lernproblems zu beobachten.

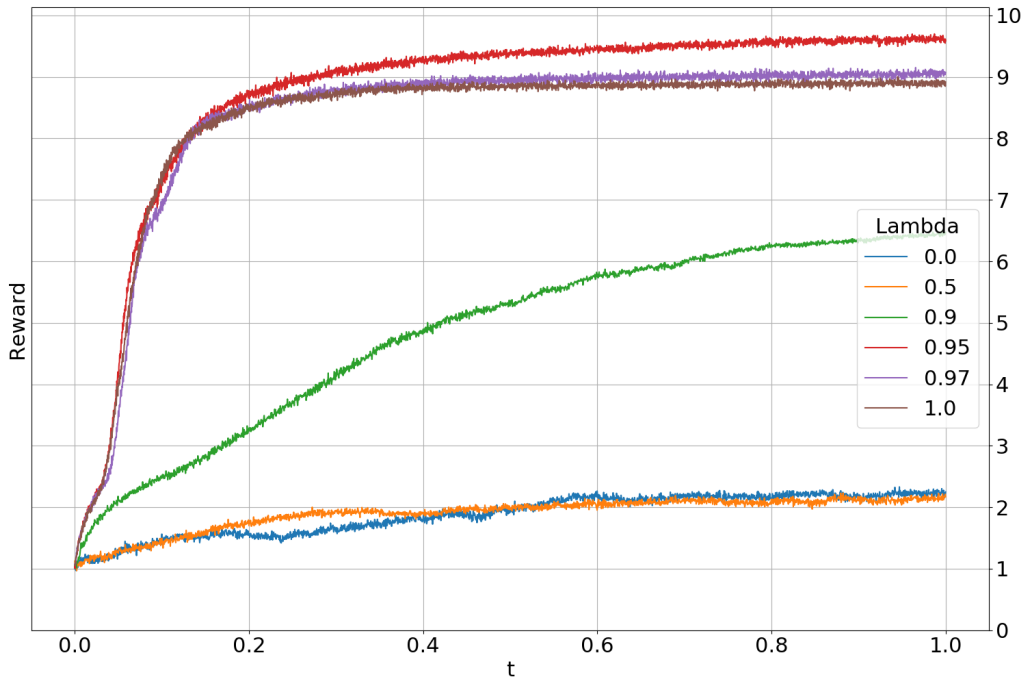


Abbildung 4.23: Vergleich unterschiedlicher Werte für den Hyperparameter λ. Dabei ist der durchschnittliche Reward entlang des Voranschreitens der Zeit dargestellt.

Abbildung 4.23 zeigt die Ergebnisse des Lernlaufs anhand verschiedenen Werten für Lambda λ. Die beiden Extrema von 0 und 1 veranschaulichen die Gewichtung von TD- zu Monte-Carlo-Methode. Falls die komplette Gewichtung auf dem TD-Ansatz liegt, können keine akzeptablen Ergebnisse erzielt werden. Selbst bei einer Gewichtung von 90% der Monte-Carlo-Methode werden die Ergebnisse sichtbar schlechter. Lambda ist dabei ein Hyperparameter des Algorithmus und unabhängig des Lernproblems, weshalb der Standardparameter der PPO2 Implementierung von Stable-Baselines auch die besten Ergebnisse liefert.

4.3.9 Clip

Der Clip-Hyperparameter ist sozusagen der grundlegende Parameter für die Trust-Region-Herangehenweise von PPO. Hierbei wird definiert, wie weit die alte Policy von der neuen abweichen kann und wie stark sich das Netzwerk pro Update verändern darf.

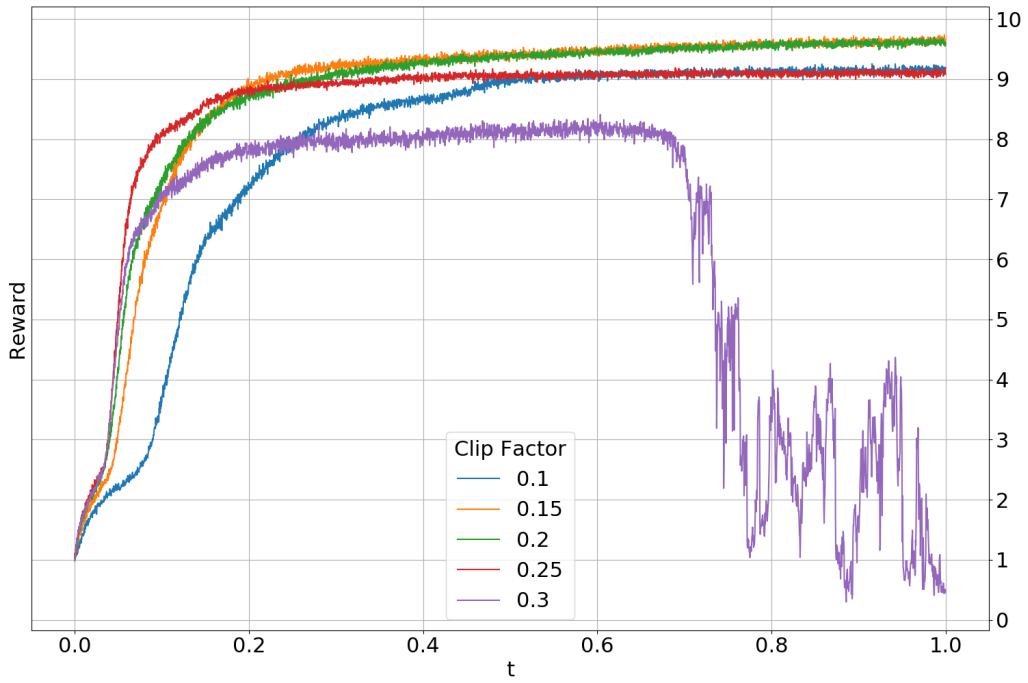


Abbildung 4.24: Vergleich unterschiedlicher Werte für den Clipping-Hyperparameter.

Die Ergebnisse für verschiedene Clipping-Werte rund um den Standardwert sind in Abbildung 4.24 zu sehen. Der Standardwert liefert den Wert 0.2, womit auch das beste Endergebniss beim Kickwalk-Problem erzielt wurde.

Falls der Clipping-Wert kleiner eingestellt wird, ist gut zu erkennen, dass der Lernfortschritt sichtlich verlangsamt wird. Die Schritte, die pro Update durchgeführt werden, werden eingeschränkt und somit können Erkenntnisse nur sehr langsam gefestigt werden. Genau das Gegenteil passiert, falls der Wert größer wird. Dabei werden die Sprünge pro Update größer und somit ist zu Beginn eine kleine Steigerung des Lernfortschritts zu erkennen. $Clip = 0.25$ lernt schneller als $clip = 0.2$.

Falls der Clipping-Wert jedoch zu groß wird, werden die Änderungen an der Policy zu groß und die Philosophie der Trust-Region-Methode ausgehebelt. Da es sich bei PPO2 um einen On-Policy Algorithmus handelt, wird das Netzwerk immer aufgrund eines lokalen Standpunkts optimiert. Der zugrundelegende Datensatz wird für ein Update verwendet und daraufhin verworfen. Der darauf folgende Datensatz wird mittels dem neu erzeugten Netzwerk gesammelt. Da nur durch einen Schätzwert ermittelt wird, ob die gewählten Aktionen besser für den Agenten waren und dieser Wert relativ ungenau ist, können die Änderungen an der Policy auch negativ ausfallen und in Zukunft einen schlechteren Datensatz für das nächste Update erzeugen.

Bei einem zu großen Wert, wie zum Beispiel 0.3, werden keine guten Ergebnisse mehr erzielt. Durch die Ergebnisse in Abbildung 4.24 wird deutlich, dass bei einer zu großen Trust-Region das Netzwerk zusammenbrechen kann und der erlernte Fortschritt verloren geht. Im Falle des Kickwalk-Problems liefern Clipping-Werte von 0.15 und 0.2 die besten Ergebnisse.

4.4 Netzwerktopologie

Die Netzwerkgröße kann die Ergebnisse signifikant beeinflussen, wie bewiesen in ähnlichen Algorithmen wie TRPO oder DDPG [25]. Zusätzlich sollten unterschiedlichste Aktivierungsfunktionen für das Policy- sowie Value-Netzwerk in Betracht gezogen werden um deren Einfluss zu beobachten [26]. Deshalb werden im Folgenden unterschiedlichste Netzwerktopologien anhand des KickWalk-Problems evaluiert.

4.4.1 Netzwerkgröße

Die Netzwerktopologie kann sehr entscheidend für den Erfolg des Lernprozesses sein, da mit der Größe des neuralen Netzes auch die Anzahl an veränderbaren Variablen steigt. Im folgenden Abschnitt werden daher verschiedene Netzwerkgrößen anhand dem Kickwalk-Lernproblem evaluiert. Dabei wurde immer dasselbe Hyperparameterset verwendet und nur die Netzwerkgröße angepasst. Die Abbildung 4.25 zeigt das Ergebnis des Versuchs.

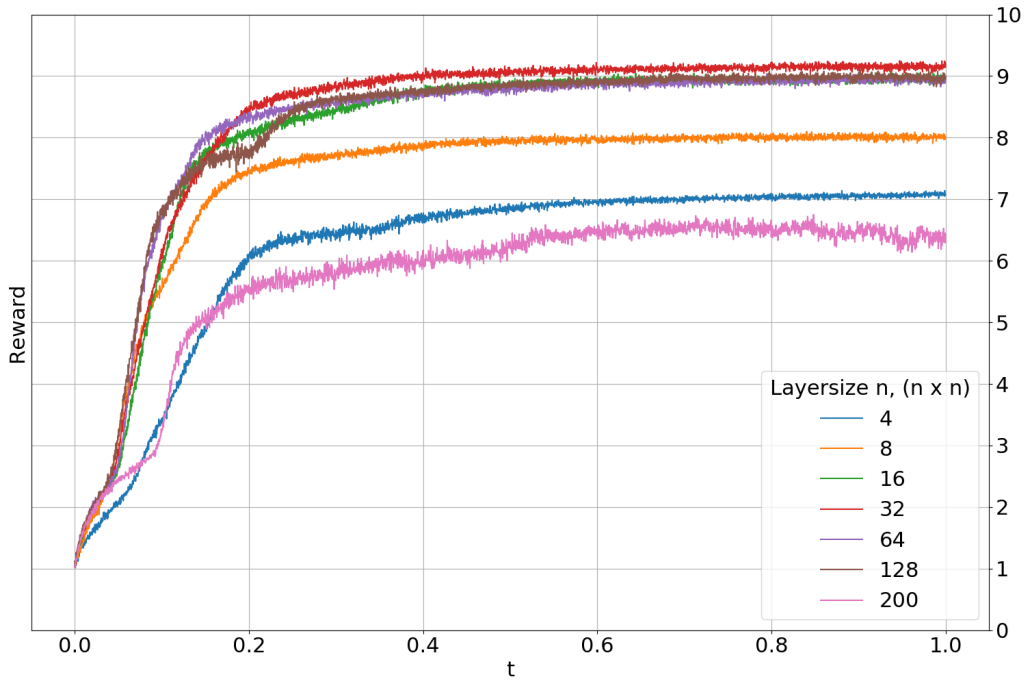


Abbildung 4.25: Vergleich unterschiedlicher Netzwerkgrößen beim KickWalk-Problem. Jeder Lernlauf wurde mit demselben Hyperparameterset gestartet. Der Default-Parameter von Stable Baselines ist die Netzwerkgröße von 64x64 Neuronen.

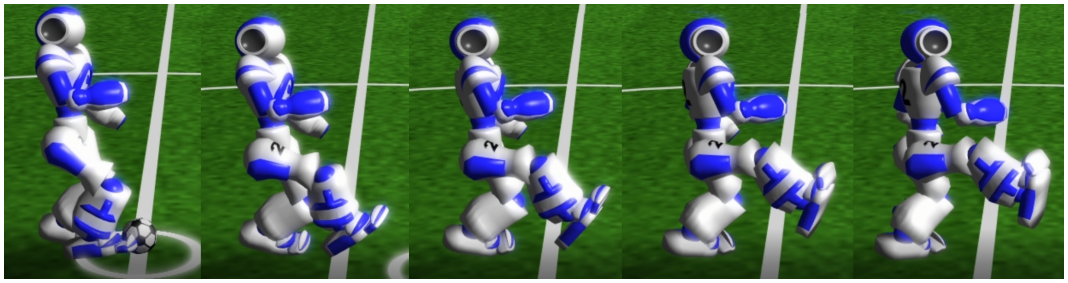
Das Netzwerk besteht immer aus zwei vollvermaschten Schichten mit einer variablen Größe. Hierbei lässt sich gut erkennen, dass die Netzwerkgröße rund um den Standard-Parameter von Stable Baselines (64x64 Neuronen) relativ gleich gute Ergebnisse liefert. Die Größe des Netzwerks definiert die Anzahl der Neuronen für das Policy sowie Value Function Netzwerk. Beide Netzwerke haben somit zwei vollvermaschte Schichten mit jeweils 64 Neuronen. Die Ergebnisse von Netzwerkgrößen von 16x16 bis 128x128 bleiben relativ konstant und in einem Bereich, welcher mit der Standardabweichung des eigentlich Lernproblems erklärt werden kann. Die Netzwerkgröße scheint in diesem Bereich kein entscheidender Faktor für den Lernerfolg des KickWalk-Problems zu sein.

Schichtgröße	Lernbare Parameter (ohne Bias)
4	688
8	1408
16	2944
32	6400
64	14848
128	37888
200	73600

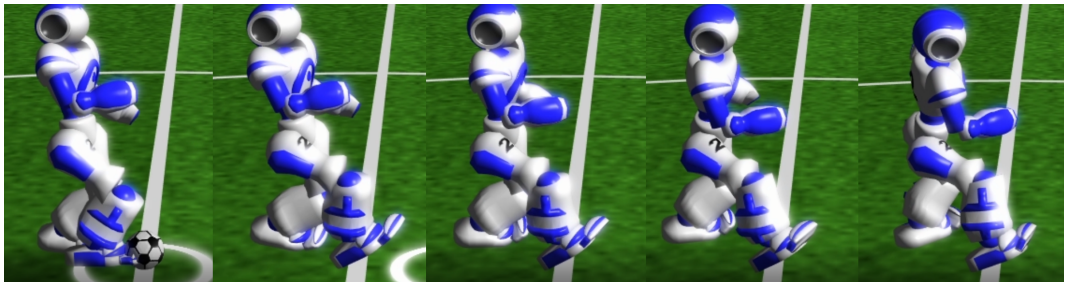
Tabelle 4.6: Vergleich der lernbaren Parameter (ohne Bias) anhand der Netzwerkgröße.

Falls das Netzwerk jedoch deutlich zu klein wird, lassen sich Einbuße im Endergebnis erkennen. Bei einer Netzwerkgröße von 4x4 und 8x8 ist das Netzwerk mit seiner geringen Anzahl an Neuronen einfach nicht mehr in der Lage die Aufgabe auszuführen. In Tabelle 4.6 ist deutlich der Unterschied zwischen den lernbaren Parametern einer Netzwerkgröße von 4 oder 8 zu 64 sichtbar. Dennoch ist es erstaunlich, dass für die relativ kleine Größe des neuronalen Netzwerks gute Ergebnisse erreicht werden können. Dies ist deutlich in der ausgeführten Bewegung zu erkennen. Bei einem größeren Netzwerk kann der Agent viel genauer mit unterschiedlichen Aktoren interagieren. Falls das Netzwerk zu klein ist, werden die Gelenke auf Anschlag ausgefahren, um den Ball möglichst weit zu kicken. Somit lässt sich eine Verschlechterung des Rewards in Kickweite, sowie auch in der Stabilität nach dem Kick erklären.

Folgende Bilderreihen sollen den Vergleich des ausgeführten Kicks nochmals bestärken und die "Grobmotorik"des kleineren neuronalen Netzes verdeutlichen.



(a) Bewegungsmuster des KickWalks bei einem sehr kleinen neuronalen Netz von einer Schichtgröße von 4 Neuronen.



(b) Bewegungsmuster des KickWalks bei der Standard Netzwerktopologie von einer Schichtgröße von 64 Neuronen.

Abbildung 4.26: Vergleich der Größe des neuronalen Netzes anhand des Bewegungsmusters des KickWalk-Problems.

Die Bewegungsabläufe in Abbildung 4.26 der beiden Netzwerkgrößen unterscheidet sich sichtlich. Das kleinere Netzwerk hat eine sehr grobe Folge von Aktionen wobei das größere Netzwerk nur die nötigen Fußbewegungen nach vorne ausführt, um nach Ausführen der Aktion nicht hinzufallen. Dem kleineren neuronalen Netz fällt es viel schwerer beide Aufgaben hinreichend auszuführen.

Zudem wurde eine dritte Schicht der gleichen Größe dem Netzwerk hinzugefügt, da die Ergebnisse möglicherweise durch die höhere gehaltene Informationsmenge besser werden könnten [27]. Dies konnte jedoch nicht bestätigt werden und die Ergebnisse mit zwei Schichten sind denen mit drei Schichten identisch.

4.4.2 Aktivierungsfunktion

Die Aktivierungsfunktion ist eine weitere einstellbare Begebenheit der Netzwerktopologie. Hierbei hat sich gezeigt, dass die Art der Aktivierungsfunktion für den Erfolg eines Lernproblems maßgeblich sein kann. Als beste Aktivierungsfunktion für PPO hat sich entweder ReLU oder Leaky ReLU herausgestellt [28]. Dies weicht jedoch mit der Standardeinstellung von PPO2 ab, da dort für das Policy-

sowie Value-Netzwerk der Hyperbeltangens tanh verwendet wird. Im Folgenden werden daraufhin unterschiedliche Aktivierungsfunktionen für die Netzwerke verwendet und anhand dem KickWalk-Lernproblem evaluiert.

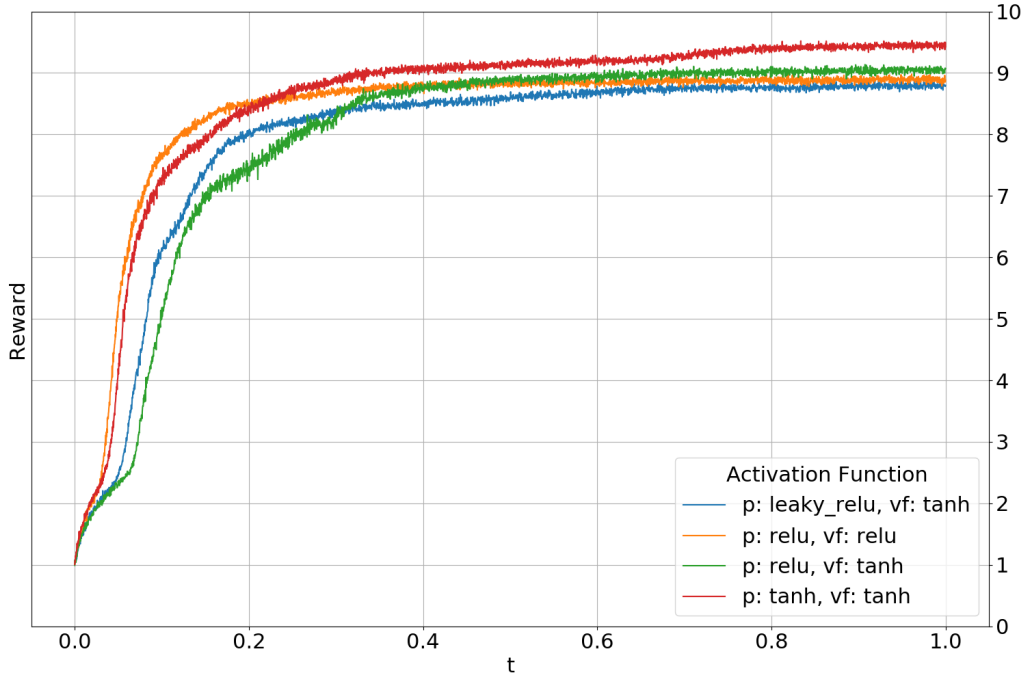


Abbildung 4.27: Vergleich zwischen unterschiedlichen Konstellationen der Aktivierungsfunktion anhand Policy- sowie Value-Netzwerk. Dabei ist der durchschnittliche Reward entlang der Zeit dargestellt.

Wie in Abbildung 4.27 zu sehen ist, liefert die Standard-Aktivierungsfunktion tanh unter PPO2 die besten Ergebnisse. Dies deckt sich mit den Ergebnissen unter vergleichbaren Lernproblemen wie HalfCheetah-v1, Hopper-v1 [28] und eines Kick-Lernproblems des RoboCup-Teams ITAndroids [29]. Die Unterschiede im durchschnittlichen Reward der Lernläufe sind jedoch minimal.

4.5 Observation-/Actionspace

Der Observation- bzw. Actionspace beschreibt die eigentliche Wahrnehmung des Agenten in seiner Umgebung und die Aktionen die er wählen kann. Der Standardobservationspace von Team Magma umfasst derzeit 120 Informationen für den Agenten, bei denen bestimmte Bereiche für Lernprobleme ausgestellt werden können.

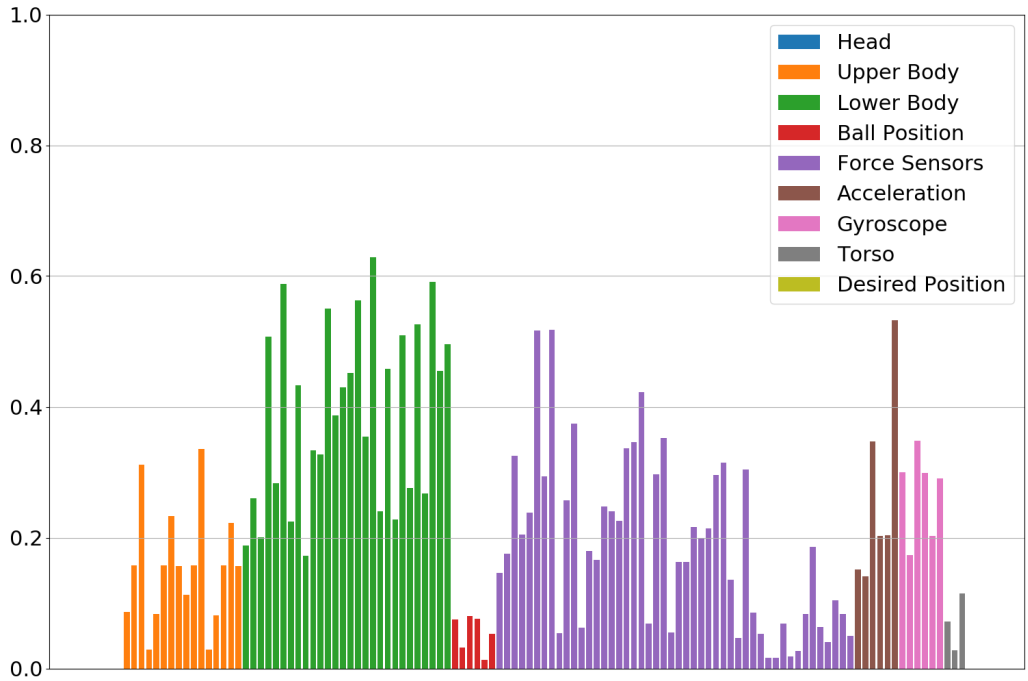


Abbildung 4.28: Visualisierung der Standardabweichung der Werte des Observationspace. Der Kopf sowie gewünschte Ballposition weisen dabei keine Werte auf, da sie für das KickWalk-Problem nicht verändert werden.

Abbildung 4.28 veranschaulicht die Wertebereiche der einzelnen Wahrnehmungen anhand 100 durchgeführten Kicks des Agenten. Hierbei ist die Standardabweichung der Werte der Gelenke, Ballposition und weiteren Größen dargestellt. Wie gut zu erkennen ist, liegt dabei die größte Abweichung der Werte im unteren Körper des Roboters. Dies ist zu erwarten, da bei einem Kick zum Großteil die unteren Gelenke des Naos in Anspruch genommen werden. Die Gelenke des Oberkörpers werden im Gegensatz zum Unterkörper eher weniger verwendet und weichen nur schwach ab. Der Kopf und dessen Gelenke sind standardmäßig für den KickWalk ausgestellt, weshalb diese nicht im Graphen angezeigt werden. Hierbei stellt sich die Frage, ob ein kleinerer Observationspace mit nur den wichtigen Informationen für eine gewisse Aktion, die Endergebnisse verbessern könnten.

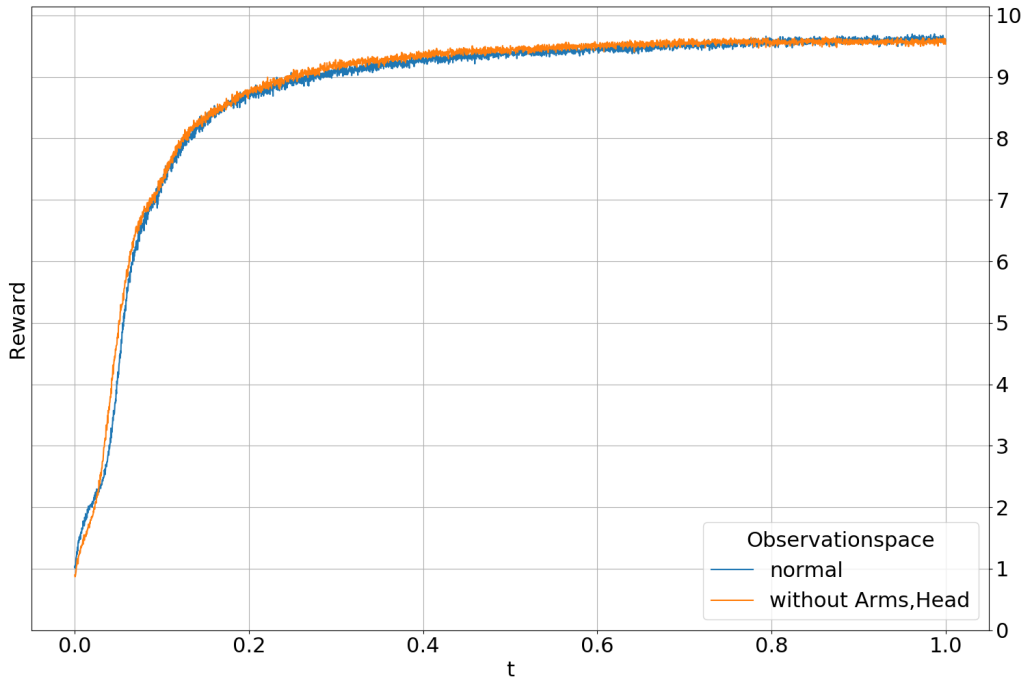


Abbildung 4.29: Vergleich des Standards zu kleinerem Observationspace und dessen Einfluss auf den Lernfortschritt sowie Endergebnis.

Die Widerlegungen dieser Hypothese finden sich in Abbildung 4.29. Hierbei wurde einmal der Standardobservationspace verwendet und einmal einen ohne die Gelenke für den Oberkörper. Der Kopf ist per Standard deaktiviert. Die Ergebnisse sind identisch, was darauf schließen lässt, dass das Netzwerk automatisch erkennen kann, welche Gelenke und Größen im Observationspace zu einem größeren Reward führen und ignoriert unwichtige Aktionsmöglichkeiten dieser Gelenke. Die Größe des Observationspace hat keinen erkennbaren Einfluss auf das erreichte Endergebnis des Agenten. Jedoch hat sich bei weiteren Experimenten herausgestellt, dass das Spiegeln gelernter Bewegungen erschwert ist, falls die Arme mitgelernt werden. Die Arme werden nicht symmetrisch bewegt, weshalb ein Rechts-Kick nicht ohne negativen Einfluss zu einem Links-Kick gespiegelt werden kann.

4.6 Pretraining

Beim Pretraining wird das erstellte Netzwerk vor dem eigentlichen Training vorinitialisiert. Das Netzwerk wird mittels eines Expertendatensatzes eines bereits gelernten Kicks vortrainiert. Normalerweise wird das Netzwerk und dessen vielen

einstellbaren Variablen zufällig initialisiert. Das vorinitialisierte Netzwerk wird damit mittels Supervised Learning und dem Expertenset trainiert. Nach dem Pretraining kann das neue Netz die Aktionen des Experten ausführen. Dabei wird nur das Policy-Netzwerk vortrainiert und nicht das neuronale Netz der Value Function. Theoretisch erhofft man sich eine Beschleunigung des Trainings, da damit von einem Experten die grundlegenden Bewegungsabläufe schon vorgelegt werden.

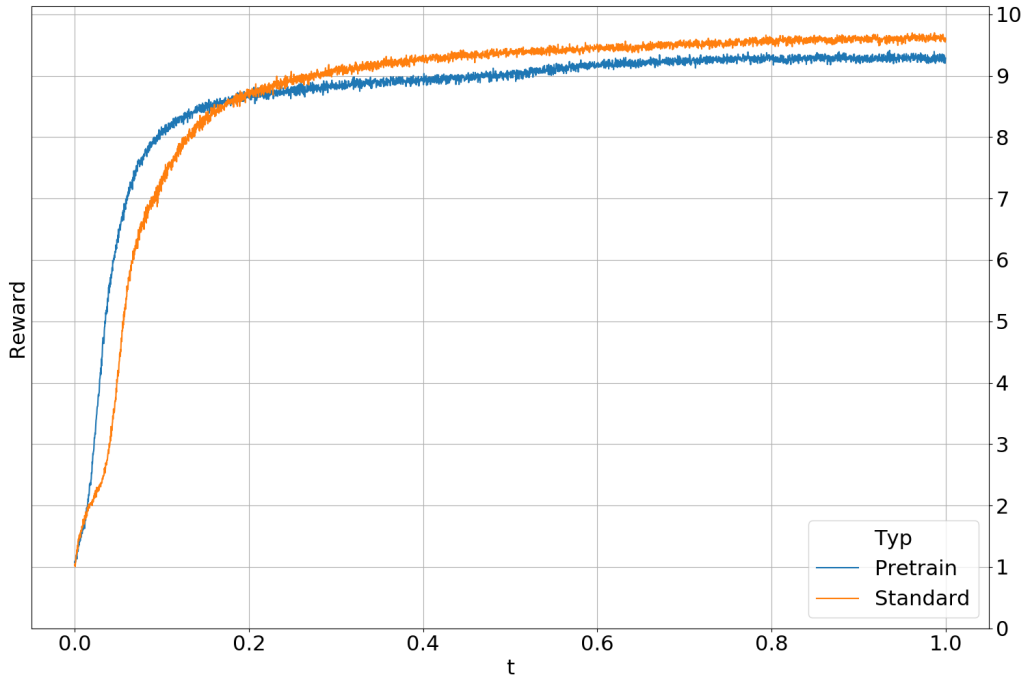


Abbildung 4.30: Vergleich des Lernfortschritts zwischen Standard und vortrainiertem Modell des KickWalk-Problems

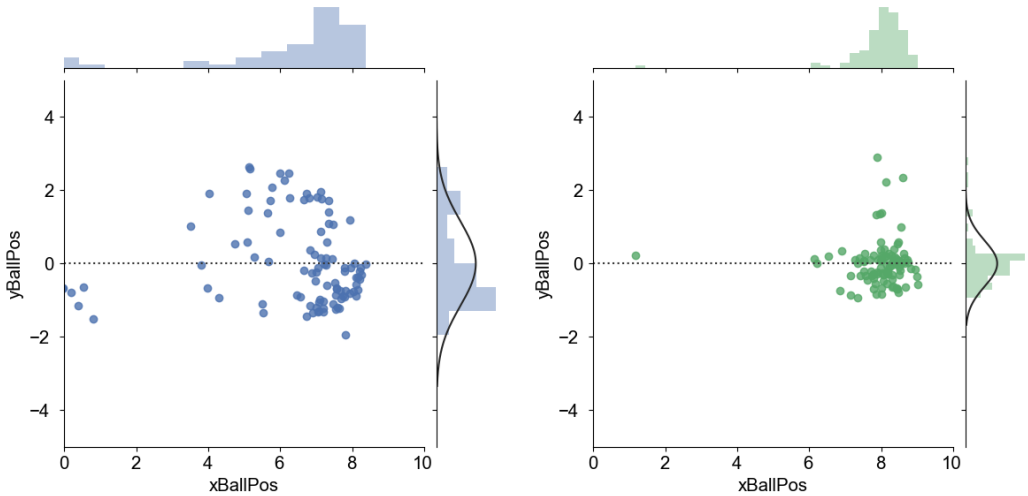
Die Ergebnisse zu einem dieser Läufe sind in Abbildung 4.30 zu sehen. Hierbei ist eine Steigerung des Lernfortschritts zu erkennen, da die Kurve schneller einen größeren Reward bekommt. Jedoch hat sich bei diesem, sowie bei weiteren ähnlichen Versuchen mit Pretraining deutlich gezeigt, dass mit zufälligem Zustand des Netzwerks signifikant bessere Ergebnisse erzielt werden können. Der Lernfortschritt ist zwar langsamer, jedoch der daraus resultierende Kick deutlich besser.

Theoretisch kann der Agent nach dem Pretraining die Aktionen ausführen. Beim Evaluieren des Netzwerkes erhält er den gleichen durchschnittliche Reward als Ergebnis. Jedoch sobald man das vortrainierte Netzwerk trainiert, erkennt man keinerlei Vorkenntnisse des Systems und der Lernprozess sieht ähnlich aus wie bei einer zufälligen Initialisierung des Netzwerks. Deshalb lässt sich hier darauf schließen,

dass das Pretraining im derzeitigen Stand nicht sinnvoll für das Lernen eingesetzt werden kann.

4.7 Vergleich mit Baseline Kicks

Im folgenden Kapitel werden die bereits vorhandenen gelernten Kicks gegenüber den gelernten Kicks der Thesis gegenübergestellt. Hierbei sollen die Ergebnisse zeigen ob sich PPO2 für das Lernen gewisser Bewegungsabläufe eignet und ob die Optimierungen der Thesis die Kicks verbessert haben. Der Baseline Kick wurde mit dem Standard-Hyperparameterset der Tabelle 4.5 gelernt.



(a) Static Kick

(b) PPO2 Kick

Abbildung 4.31: Vergleich der zwei gelernten Kicks anhand der Ballposition nach Ausführung. Auf der linken Seite ist der Baseline Kick (a) und auf der rechten Seite der von PPO2 gelernte Kick (b). Jeder Punkt ist dabei die Ballendposition in x- und y-Achse auf dem Spielfeld. Startposition des Spielers sowie des Balls ist dabei $x, y = 0$.

In Abbildung 4.31 wurden zwei Kickverhalten anhand 100 durchgeführter Kicks evaluiert und gegenüber gestellt. Jeder Punkt visualisiert die Endposition des Balls, nachdem der Kick durchgeführt wurde. Der Spieler steht mit kleinen Abweichungen auf dem Mittelpunkt des Spielfelds und schießt in x-Richtung.

In Abbildung 4.31a sind die Ergebnisse eines bereits vorhandenen Kicks von Team Magma dargestellt. Hierbei sieht man noch eine relativ hohe Streuung der Endpositionen des Balls. Daraufhin gibt es einige Kicks, welche komplett fehlschlagen und nur etwas über die Mittellinie gehen. Vergleicht man das mit dem Ergebnis der

Optimierungen in Abbildung 4.35b, können deutlich die Unterschiede der beiden Endergebnisse gesehen werden. Die Streuung beim PPO2 Kick ist deutlich kleiner und nahe zu alle Bälle Enden in einem sichtbaren Bereich. Dabei gibt es jedoch immer noch kleinere Abweichung in der x- sowie y-Achse.

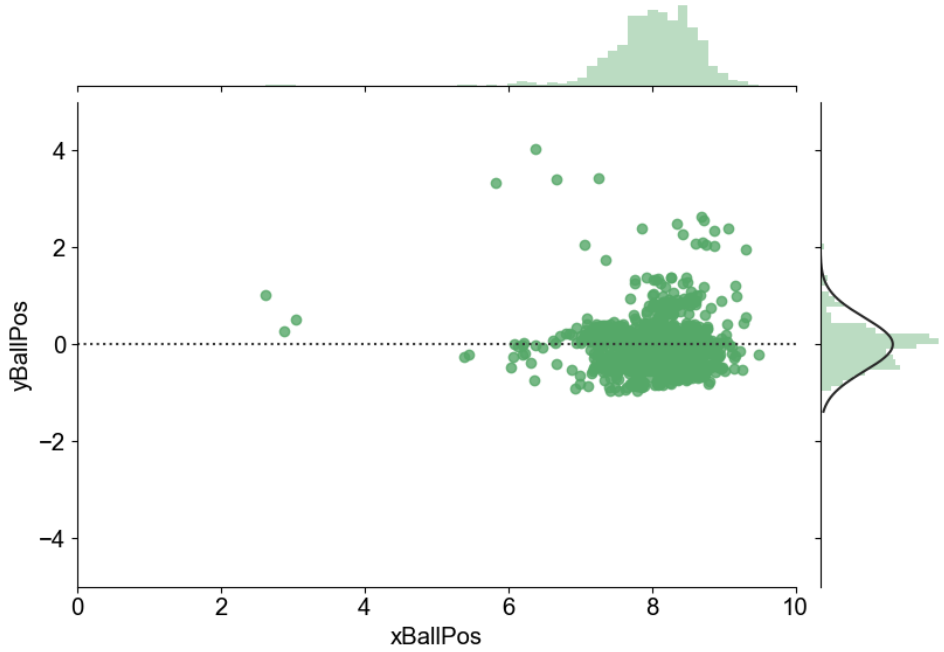


Abbildung 4.32: 1000 Ballpositionen nach Ausführen des PPO2 gelernten Kicks.

Das Modell für die PPO2 Kicks wurde daraufhin weiter evaluiert und die Ergebnisse von 1000 Ausführungen visualisiert, um einen genaueren Eindruck der Qualität des Kicks zu bekommen. Die Ergebnisse sind in Abbildung 4.32 aufgeführt. Das erneute Evaluieren mit einer größeren Anzahl an Kicks spiegelt die Ergebnisse aus den vorherigen Tests wider. Der Bereich der Endpositionen der Bälle bleibt mit steigender Anzahl der Kicks konstant. Dabei sind von 1000 Ausführung des Gelernten nur drei missglückt und haben es nur bis ca. drei Meter weit geschafft.

4.8 Varianz der Startposition

Die Endergebnisse des KickWalk-Problems sind akzeptabel, würden jedoch im Spiel nicht zu einer signifikanten Verbesserung gegenüber den genetisch gelernten Kicks führen. Um einen möglichen Vorteil gegenüber den genetischen Kicks zu erreichen,

wurde der Bereich der variablen Startposition deutlich vergrößert. Beim normalen KickWalk variiert die Startposition des Spielers in einem Bereich von $x = -0.19$ und $y = -0.06$ mit einer zufälligen Abweichung der beiden Werte von $+/-0.02$. Die Ballposition ist ständig auf dem Nullpunkt des Spielfelds und nur die Spielerposition weicht ab. Dabei wird beim neuen Lernbereich eine Fläche in x-Achse von -0.16 bis -0.25 und in y-Achse von 0 bis 0.12 abgedeckt. Bei dieser Modifikation ist es zusätzlich möglich diesen zufälligen Bereich zu vergrößern, um auch ungelernte Bereiche zu evaluieren. Hierbei wird sozusagen beim Lernproblem eine größere Varianz des zu lernenden Datensatzes erzeugt, in dem der Agent nicht blind diese Aktion ausführen kann, sondern gezielt auf die zufällige relative Ballposition achten muss. Die Ergebnisse des besten Hyperparameterset des vorherigen KickWalk-Problems haben bei diesem Lernproblem zu einer Verbesserung der Ergebnisse geführt. Der Unterschied zwischen den beiden Hyperparameterkonfigurationen liegt im Durchschnitt bei einer Verbesserung des Rewards von 2.

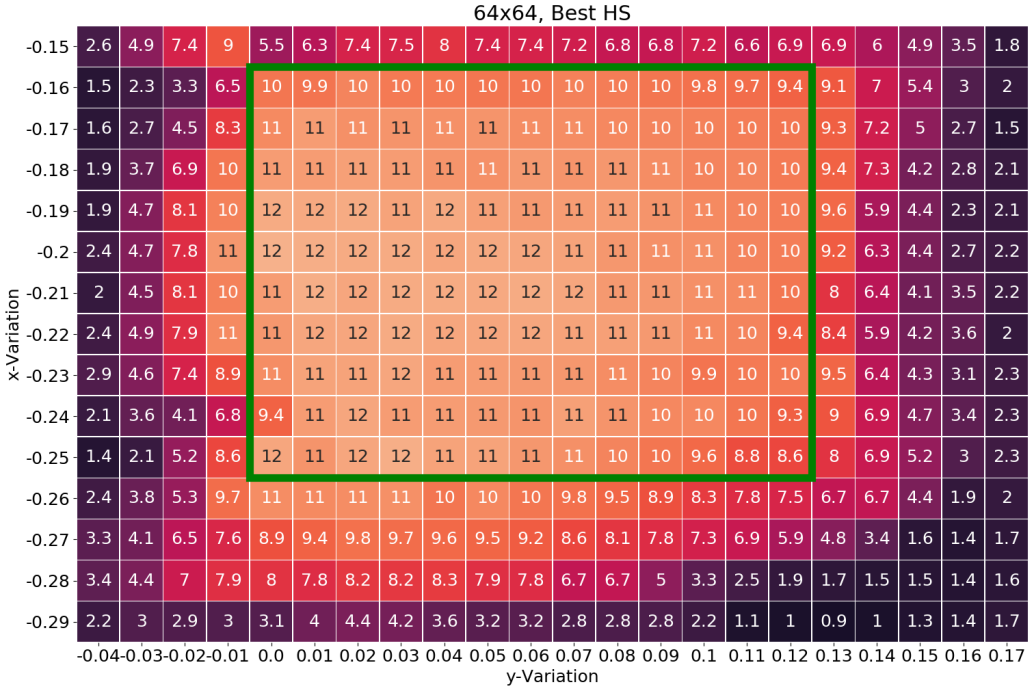


Abbildung 4.33: Heatmap des durchschnittlichen Rewards über 50 Kicks mit abweichender Startposition in x- und y-Achse. Der gelernte Bereich ist dabei grün umrandet.

Der gelernte Kick kann daraufhin anhand der variablen Startposition evaluiert werden und eine Heatmap dafür erstellt werden, welche in Abbildung 4.33 zu sehen ist. Dabei wird der Agent auf jede Position in x- und y-Richtung platziert und führt

50 Kicks bei jeder x- und y-Kombination aus. Der Wert in jeder Kachel ist der durchschnittliche erhaltene Reward. Hierbei kann man sehen, dass der Agent relativ unabhängig von der Startposition über die gesamte Fläche gute Ergebnisse liefert. Der grün markierte Bereich in der Heatmap zeigt dabei den Varianzbereich, der während des Trainings verwendet wurde. Es ist gut ersichtlich an welchen Positionen der Kick möglich ist und an Welchen nicht. Der durchschnittliche Reward liegt deutlich über dem erreichten Reward der vorherigen Lernläufe. Durch die erhöhte Varianz der Startposition wurden demnach möglicherweise bessere Erkenntnisse gezogen, weshalb der Kick übergreifend verbessert wurde.

Dieses abgewandte Lernproblem hat sich als sehr viel nützlicher erwiesen gegenüber dem ursprünglichen KickWalk, da nun abhängig der Ballposition eine leicht veränderte Bewegung ausgeführt wird. Während eines Spiels kann es oft vorkommen, dass der Spieler nicht immer optimal zum Ball steht. Durch ein Netzwerk, welches abhängig von der relativen Ballposition agiert, könnten in kommenden Turnieren gute Ergebnisse erzielt werden.

Im derzeitigen Stand von Team Magma werden hauptsächlich genetisch gelernte Bewegungsmuster ausgeführt. Hierbei stellt sich natürlich die Frage, ob der neue PPO2 Kick besser abschneidet im Vergleich mit seinem genetisch gelernten Pendant.

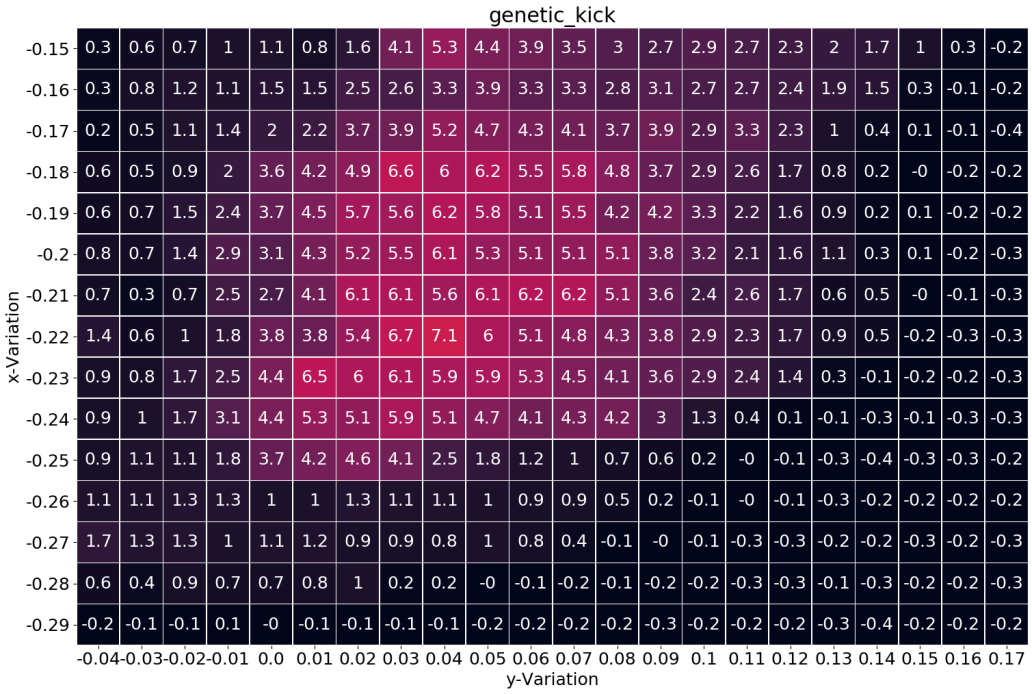
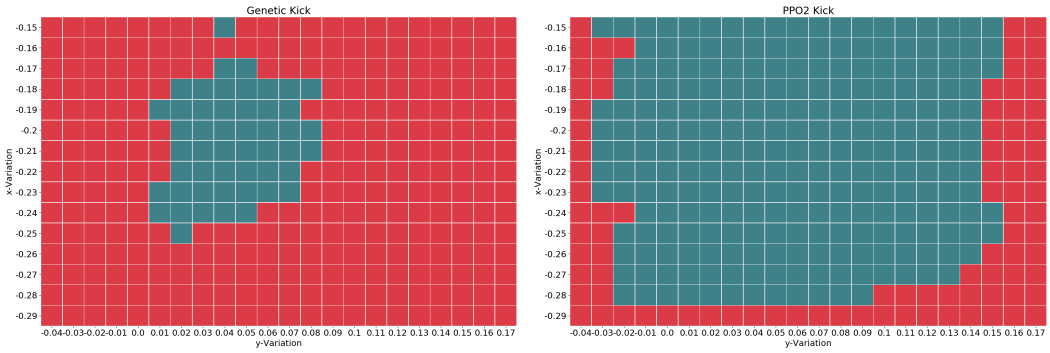


Abbildung 4.34: Heatmap des durchschnittlichen Rewards des genetisch gelernten Kicks anhand der variablen Startposition.

Abbildung 4.34 zeigt dabei die Heatmap des derzeit verwendeten genetischen Kick-Walks. Hierbei muss jedoch beachtet werden, dass die Rewardfunktion nicht völlig identisch ist mit der des PPO2 Kicks. Dennoch kann eine Aussage über den evaluierten Bereich anhand der Abweichung der Werte getroffen werden. Beim genetischen Kick sieht man deutlich, dass er gute Rewards in einem gewissen Bereich erhält. Außerhalb dieses Bereichs schneidet dieser Kick jedoch relativ schlecht ab und bleibt deutlich unter den maximalen Rewards der Heatmap. Die genetische Alternative scheitert bei der erhöhten Varianz der Startposition. Genetisch gelernte Bewegungsmuster sind immer identisch und werden immer so ausgeführt wie sie erlernt wurden. Der Vorteil der DRL Bewegungen ist, dass diese reaktiv bei jedem Schritt auf ihre Umgebung reagieren und somit die variierende Ballposition erkennen können. Legt man nun die Ergebnisse des neuen PPO2 Kicks mit dem genetischen Kick gegenüber sieht man deutlich die Verbesserungen.



(a) Genetic Kick **(b)** PPO2 Kick

Abbildung 4.35: Vergleich des kickbaren Bereichs des genetischen gegen PPO2 Kick.

Die Heatmaps in Abbildung 4.35 visualisieren den Bereich, bei dem der gelernte Kick ausführbar ist und gute Ergebnisse liefert. Hierbei sind grüne Kacheln ausführbare und Rote nicht. Wie deutlich zu erkennen ist, ist der Bereich beim PPO2 gelernten Kick signifikant größer als bei der genetischen Alternative. Um die beiden konkurrierenden Kicks zu evaluieren wurde eine Spielserie mit 200 Spielen erstellt. Ein Team verwendet dabei den genetischen Kick und das andere den PPO2 Kick. Mit dem erlernten Kick dieser Thesis wurde im Mittel 0.665 zu 0.385 gespielt. Zusätzlich wurde der PPO2 Kick durchschnittlich fünfmal so oft ausgeführt als der genetische Kick. Dies zeigt einen signifikanten Vorteil des PPO2 Kicks gegenüber dem genetisch gelernten Kick, weshalb er erstmals in der Ersatz-WM 2020 eingesetzt wird.

5. Vergleich mit Walk-Problem

Der Hauptteil dieser Arbeit wurde anhand des KickWalk-Problems evaluiert und ausgetestet. Einige Hyperparameter oder Herangehensweisen müssen jedoch zusätzlich noch an anderen Lernproblemen getestet werden. Hierbei werden im Folgenden einige Parameter anhand des Walk-Lernproblems evaluiert. Beim Walk-Lernproblem handelt es sich um ein sehr kritisches Lernproblem, da es sich um ein kontinuierliches Lernproblem handelt und damit sehr lange in der Ausführung ist. Dabei werden die Herausforderungen und Problematiken anhand den Aspekten: Konsistenz, Einfluss der Dummy-Observations sowie des Discount Factors begründet.

5.1 Asynchrones Lernen

Da die Funktionalität unabhängig von dem eigentlichen Lernproblem steht, kann diese bei jedem Problem eingesetzt werden. Von Lernproblem zu Lernproblem unterscheidet sich jedoch, wie viel Overhead entstehen kann. Also wie oft der Spieler zum Beispiel zurückgesetzt werden muss oder an bestimmte Positionen platziert wird. Während diesen Operationen kann eine Instanz keine Ergebnisse liefern. Theoretisch hat sich dabei beim KickWalk bei einem Prototyp des asynchronen Lernen ein Verhältnis von 40/60% von Validen zu Dummy-Observations herausgestellt. Dieser Grenzwert von 40% ist jedoch bei jedem Lernproblem an einer unterschiedlichen Stelle, je nachdem wie viel Overhead erzeugt wird.

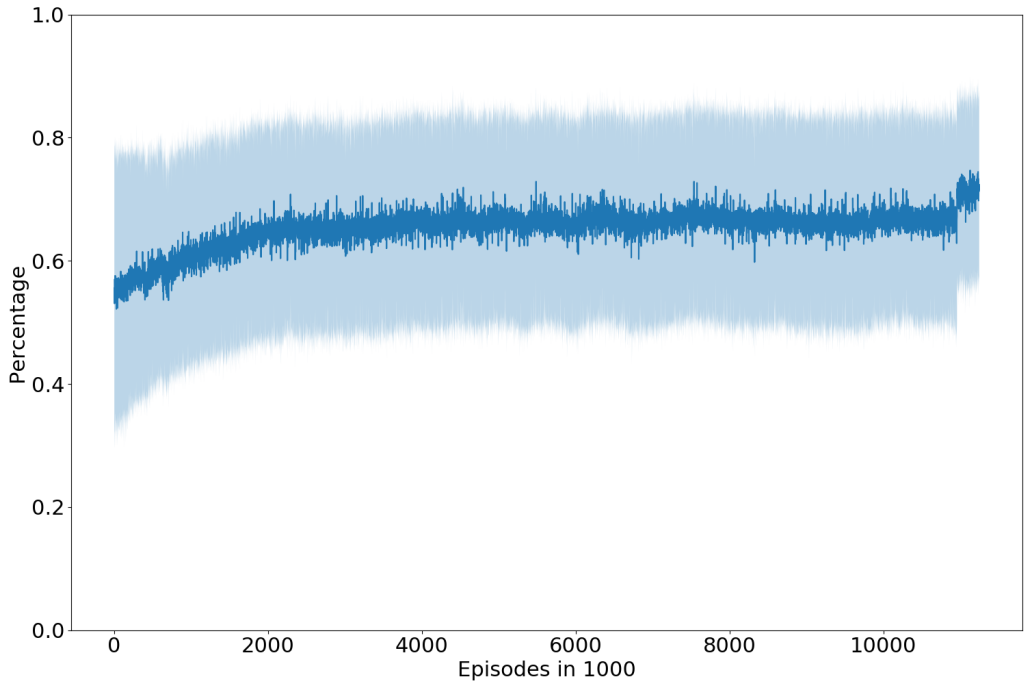


Abbildung 5.1: Zeitlicher Verlauf des Verhältnisses zwischen validen und Dummy-Observations entlang der durchgeführten Episoden des Walk-Lernproblems. Der Graph visualisiert das prozentuale Verhältnis von validen Observations und dessen Standardabweichung über 1000 Episoden.

Beim Walk-Lernproblem sieht der Overhead etwas anders aus, als der beim Kick-Walk. Wie in Abbildung 5.1 deutlich zu erkennen ist, schwankt dieser unbeschränkte Grenzwert beim asynchronen Prototyp sehr stark. Vor allem in den ersten 2.000.000 Episoden sinkt der Overhead mit der Zeit. Dies liegt daran, dass der Agent zu Beginn beim Walk relativ häufig hinfällt und die Episoden relativ kurz dauern. Durch diese Beobachtung kann darauf geschlossen werden, dass in den frühen Episoden und dessen Updates sehr viel mehr Dummy-Observations vorhanden sind, als durch den gesamten Durchschnitt angenommen.

Beim KickWalk ist die Ausführung immer 18 Zyklen lang, weshalb sich dieser Grenzwert während dem Lernen wenig verschiebt. Der Overhead des Walk-Problems ist Anfangs relativ hoch, gegen Ende, sobald der Agent etwas laufen kann jedoch sehr gering. Beim Walk hat sich jedoch herausgestellt, dass sich ein Grenzwert von 60% an richtigen Observations mittelt. Dieser prozentuale Anteil ist jedoch über das Voranschreiten des Lernproblems nicht stabil.

Bei der finalen Implementierung des asynchronen Lernens ist dies jedoch nicht entscheidend. Hierbei kann ein Minimum an gesammelten validen Observations fest-

gelegt werden, das zwingend erreicht werden muss. Dabei unterscheidet sich von Lernproblem zu Lernproblem nur der Durchsatz, also wie viele Episoden pro Zeit durchgeführt werden. Da bei jedem Lernproblem jedoch die Episodenzeit unterschiedlich ist, wird dies schwer zu ermitteln. Jedoch ist die Unterscheidung der Effizienz zwischen den einzelnen Lernproblemen auch nicht wirklich relevant. Der finale asynchrone Modus sollte jedoch unabhängig davon die Ergebnisse nachvollziehbarer und in schnellerer Zeit darbieten können.

5.2 Einfluss der Dummy-Elemente

Wie in vorherigen Kapiteln erwähnt, ist beim Walk-Lernproblem das Einsetzen des asynchronen Prototyps eventuell nicht von Vorteil für das Endergebnis. Der Anteil der Dummy-Observations in den ersten Episoden ist deutlich höher, als der Durchschnitt wie in Abbildung 5.1. Dies könnte dazu führen, dass ein schlechteres Ergebnis gefunden wird aufgrund der unzureichenden Datenmenge in den frühen Phasen des Lernens. Durch die hohe Varianz des Lernproblems ist dies relativ schwer zu beweisen. Jedoch hat sich bei mehreren Lernläufen mit einem gewissen Minimum an validen Observations gezeigt, dass es eine Tendenz für bessere Ergebnisse gibt.

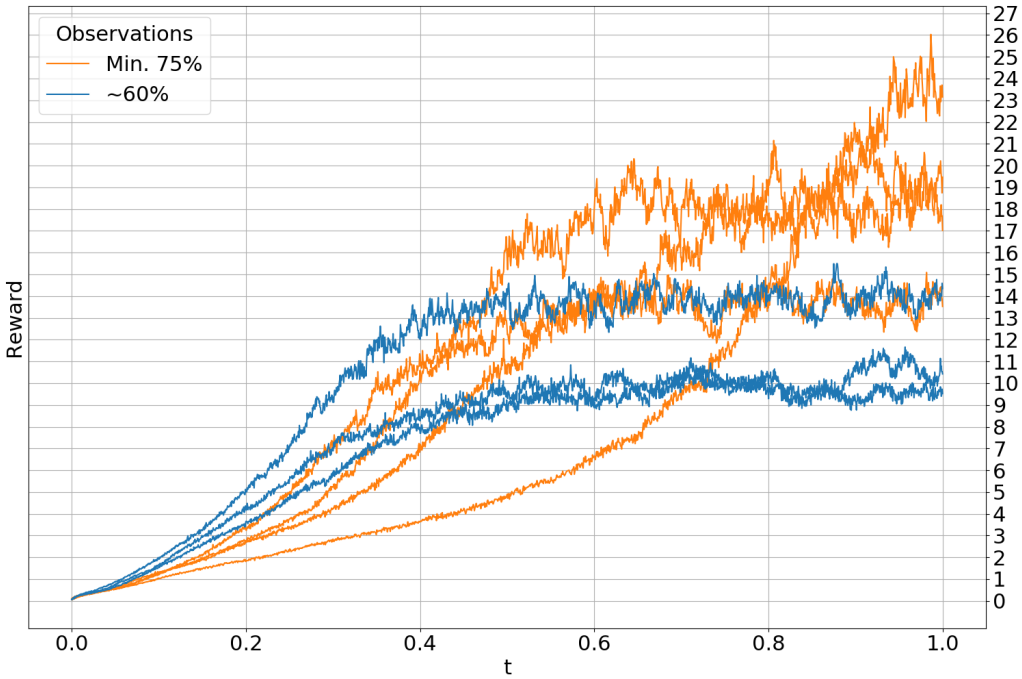


Abbildung 5.2: Vergleich des unterschiedlichen Lernfortschritts des Walk-Problems anhand Einfluss der Dummy-Elemente.

In Abbildung 5.2 ist die Gegenüberstellung mit und ohne festgelegten Grenzwert des Lernproblems visualisiert. Dabei kann man deutlich erkennen, dass die Ergebnisse mit einem festen Grenzwert von mindestens 75% an validen Observations im Durchschnitt deutlich bessere Ergebnisse erzielen. Dies könnte am Mangel an Daten in den frühen Updatezyklen liegen. Mit Grenzwert sind pro Update immer die gleiche Anzahl an validen Observations vorhanden und somit kann eventuell eine bessere Grundlage für kommende Episoden und dementsprechend ein besseres Endresultat nach den 48 Stunden erreicht werden.

5.3 Konsistenz

Das Walk-Problem hat sich als sehr inkonsistentes Lernproblem herausgestellt. Da die Netzwerke zu Beginn eines Trainings zufällig vorinitialisiert werden, lernt der Agent bei jedem Lernlauf einen unterschiedlichen Bewegungsablauf.

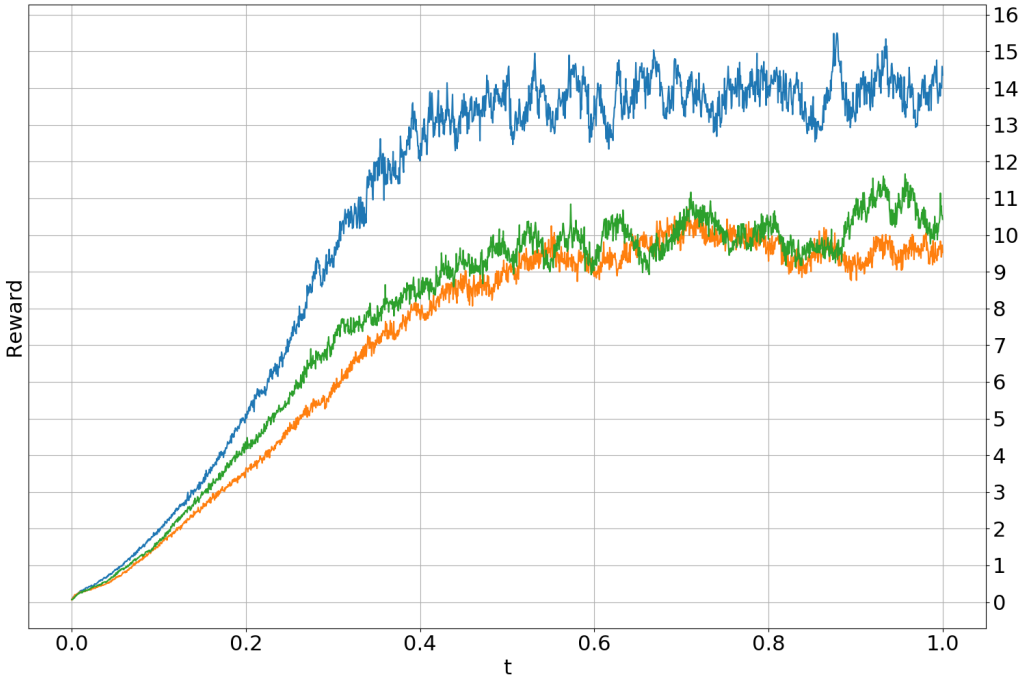


Abbildung 5.3: Vergleich des unterschiedlichen Lernfortschritts des Walk-Problems anhand identischem Setup.

In Abbildung 5.3 sind dabei die Ergebnisse des durchschnittlichen Rewards anhand dreier identischer Lernläufe dargestellt. Einer der Läufe ist in etwa 50% besser als die anderen zwei. Es hat sich im Laufe der Evaluierung anhand dieses Lernproblems herausgestellt, dass das genaue Einstellen der Hyperparameter weniger sinnvoll ist, da die Ergebnisse mit gleichem Hyperparameterset drastisch voneinander abweichen können. Somit könnte ein besseres Set an Hyperparametern dennoch schlechtere Ergebnisse liefern. Da die Zeit bis zur Konvergenz beim Walk-Lernproblem deutlich länger ist als beim KickWalk, ist es zu zeitaufwendig die Ergebnisse anhand mehrere Läufe zu mitteln. Einige erreichten ihr lokales Optimum schon nach ca. 24 Stunden. Wohingegen einige Läufe mit identischem Setup erst nach über 100 Stunden konvergierten.

5.4 Discount Factor

Der Discount Factor hat sich beim Walk-Lernproblem als sehr einflussreicher Hyperparameter herausgestellt. Dies kann unabhängig von der großen Varianz des Lernproblems festgestellt werden. Durch diesen Hyperparameter können sehr un-

terschiedliche Lernergebnisse erreicht werden. Beim KickWalk-Problem hat es sich herausgestellt, dass ein γ von 1.0 sinnig ist, aufgrund der kurzen Zeitdauer der Ausführung. Beim Laufen lernen hat dieser Wert jedoch sehr interessante Ergebnisse geschaffen. Durch das fehlende Abzinsen der zukünftigen Rewards versucht der Agent so langsam wie möglich und so stabil wie möglich zu laufen, um die größtmögliche Distanz zurückzulegen. Dies liegt daran, dass der Endreward auf die Summe der kommenden Rewards mit voller Gewichtung auf alle Aktionen gerechnet wird. Bei einem kleineren Wert des Discount Factors versucht der Agent eher die Rewards pro Zyklen zu erhöhen und möglichst schnell voran zu kommen, da die meisten Aktionen bei einer großen Ausführungsdauer den Endreward nur geringfügig zugeteilt bekommen.

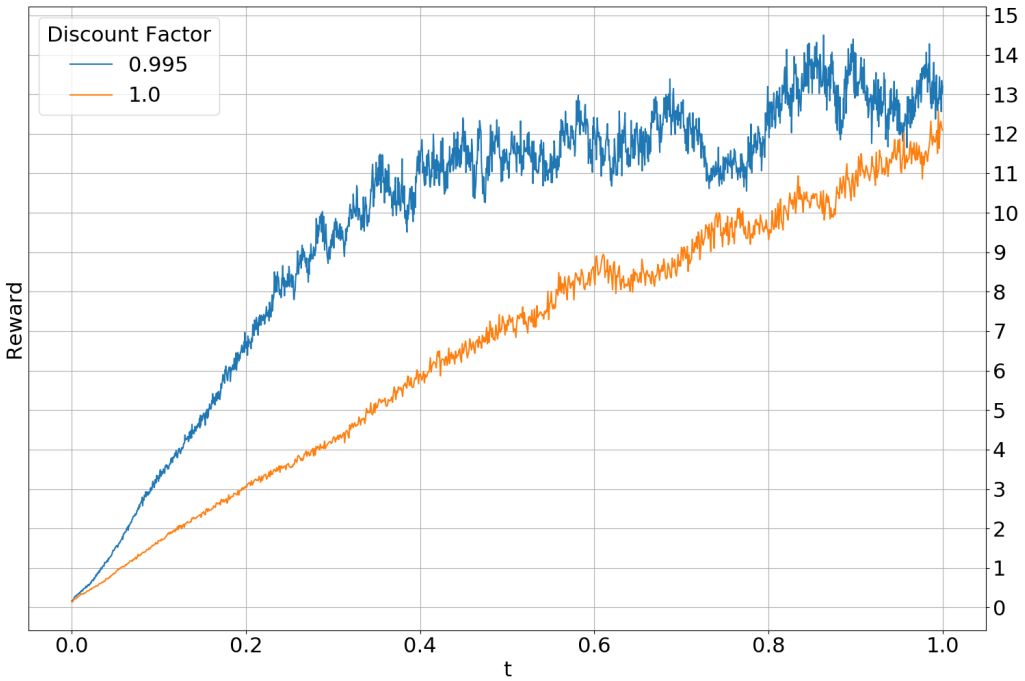


Abbildung 5.4: Vergleich des unterschiedlichen Lernfortschritts des Walk-Problems anhand des Gamma-Hyperparameters.

Beide Lernläufe mit unterschiedlichem Wert für γ sind in Abbildung 5.4 dargestellt. Dabei ist gut zu erkennen, wie beim Aushebeln des Discount Factors das Netzwerk, mit deutlich weniger Abweichung, besser im durchschnittlichen Reward wird. Die Variation bei einem geringeren Discount Factor ist deutlich höher.

6. Zusammenfassung und Ausblick

Abschließend werden nun die Ergebnisse dieser Arbeit zusammengefasst und ein Ausblick auf kommende Verbesserungsmöglichkeiten gegeben.

6.1 Zusammenfassung

Die Ergebnisse der Arbeit zeigen deutlich die Kompetenz des Deep Reinforcement Learnings um komplexe Aufgabenstellungen selbständig und effizient zu lösen. PPO2 hat sich als benutzerfreundlichen Algorithmus herausgestellt der, mit Ausnahme des Entropy Coefficient und dem Abzinsfaktor γ, recht einfach zu optimieren ist. Die Arbeit hat einen Überblick über die Vielzahl der einstellbaren Parameter gebracht und deren Einfluss auf das Lernverhalten aufgezeigt. Die Verbesserungen des asynchronen Modus haben zu einer signifikanten Verbesserung des Durchsatzes geführt. Hierbei konnte der Durchsatz bei 24 parallelen Instanzen beim KickWalk-Problem um 40% gesteigert werden. Zudem wird bei der neuen Implementierung ein Minimum an Werten sichergestellt, was beim alten Modus vermutlich zu unerwünschten Effekten geführt hat. Durch das Setzen des Grenzwertes kann ein Kompromiss zwischen Asynchronität und Durchsatz gewählt werden.

Die erzielten Ergebnisse mit PPO2 haben gezeigt, dass Bewegungsabläufe vor allem in reaktiven Verhalten verbessert werden können. Durch Erhöhen der zufälligen Startposition des Spielers, konnte das System lernen den Ball anhand der relativen Ballposition zu kicken. Die erzielten Ergebnisse sind signifikant besser als die vorher verwendeten genetischen Kicks. Durch diese Verbesserungen ist es möglich den Kick bei einer Abweichung der Ballposition seltener zu verfehlen. Zusätzlich hat sich auch gezeigt, dass das Lernen mit einer erhöhten Streuung das Ergebnis des KickWalks übergreifend verbessert hat. Die anderen Lernprobleme haben sich als weniger erfolgreich herausgestellt. Dies sollte jedoch nicht am Setup von PPO2

liegen, sondern an der Rewardfunktion sowie initiale Konfiguration des Problems. Beim Walk-Problem konnten akzeptable Ergebnisse erzielt werden. Dennoch ist die Streuung der Ergebnisse relativ hoch und der Nao fällt häufig hin. Zudem hat sich der Lernfortschritt als sehr inkonsistent herausgestellt, weshalb es nötig ist das Setup anzupassen, um gute Ergebnisse zu erhalten.

Über den Verlauf dieser Arbeit wurde die bereitgestellte Serverhardware nahezu rund um die Uhr verwendet. Die Evaluierung verschiedener Hyperparameter hat sich als sehr zeitintensiv herausgestellt. Im Folgenden werden Statistiken präsentiert, welche den Umfang dieser Evaluierung verdeutlichen sollen:

- 6.040 Stunden Serverlaufzeit
- 252 Tage Serverlaufzeit
- 127 individuelle Lernläufe
- 702.295 Netzwerk-Updates
- 327.042.313 Episoden
- 12.932.032.512 Zyklen

6.2 Ausblick

Es gibt viele Richtungen, in welche die Ergebnisse dieser Arbeit verwendet werden können. Einige Beispiele sind dabei:

- Die Optimierung des Durchsatzes können für kommende Lernprobleme eingesetzt werden und deren Fortschritt maßgeblich zu beschleunigen. Durch den neuen asynchronen Modus konnte ein Grundbaustein für die kommenden Lernprobleme geschaffen werden, um deren Entwicklung sowie Evaluierung effizienter zu gestalten.
- Der asynchrone Modus bewirkt keine rein parallele Abarbeitung. Der Durchsatz skaliert nicht linear mit der Anzahl der parallelen Instanzen. Theoretisch könnte hier durch Ansätze wie dem Hogwild-Ansatz der Durchsatz noch weiter gesteigert werden [30], [31]. Durch den großen Overhead bei manchen Lernproblemen könnte dies eine Investition sein, die sich lohnen würde.
- Die Ergebnisse des KickWalks haben gezeigt, dass die erlernten Bewegungsmuster besser sein können, als ihr genetisch erlerntes Gegenstück. Deshalb wird in Zukunft ein Fokus auf Deep Reinforcement Learning gesetzt und ei-

ne Vielzahl an verschiedenen Bewegungsmuster erlernt, wie z.B. einen Kick in beliebige Richtung oder Distanz.

- PPO2 lernt sehr gierig, weshalb es schwer ist verschiedene Bewegungsmuster zu erlernen. Beim KickWalk war es nicht möglich eine Rückwärtsbewegung des Fußes zu erlernen, obwohl der Kick vermutlich damit weiter gehen würde. Daher würde es sich lohnen einen Blick auf off-policy Alternativen zu werfen, da diese weniger gierig voranschreiten.
- Die Evaluierung der Hyperparameter hat gezeigt, dass diese mit einigen Ausnahmen weniger einflussreich für das Endresultat sind. Das Ergebnis kann zwar mit einem optimalen Set verbessert werden, jedoch erhält man auch gute Ergebnisse mit einem Standardset. Diese kritischen Größen wurden identifiziert und können in zukünftigen Lernproblem dementsprechend eingestellt werden.
- Nachfolgend dieser Arbeit, werden weitere Experimente im Aspekt der domänenseitigen Anpassung des Lernens mit PPO2 durchgeführt. Der Einfluss der Rewardfunktion sowie weiteren Größen der Domäne soll deshalb zukünftig näher evaluiert werden.

All diese Punkte sollten dazu führen, dass zukünftig mehr und mehr Bewegungsabfolgen mittels PPO2 erlernt werden und im Spiel von Team magmaOffenburg eingesetzt werden. Die Ergebnisse dieser Arbeit, sowie anderen Teams des RoboCups, haben gezeigt, dass Deep Reinforcement Learning definitiv ein Schritt in die Richtung intelligenter autonomer Roboter ist.

Abkürzungsverzeichnis

RL Reinforcement Learning
DRL Deep Reinforcement Learning
PPO2 Proximal Policy Optimization: Implementierung von Stable-Baselines
PPO Proximal Policy Optimization
ODE Open Dynamic Engine
SGD Stochastic Gradient Descent
BGD Batch Gradient Descent
MGD Minibatch Gradient Descent
MDP Markov Decision Process

Tabellenverzeichnis

3.1 Spezifikation des Inhalts des Observationspace 27
3.2 Spezifikation des Inhalts des Actionspace 28

4.1 Anzahl der Updates des Netzwerks in 48 Stunden anhand der parallelen Instanzen. 33
4.2 Vergleich des Durchsatzes der asynchronen sowie synchronen Lernmethode. 37
4.3 Gegenüberstellung des Durchsatzes verschiedener Lernmethoden . . 41
4.4 Durchsatz anhand Grenzwert 43
4.5 Standard Hyperparameterset der Thesis im Vergleich zu Stable-Baselines 45
4.6 Vergleich der lernbaren Parameter (ohne Bias) anhand der Netzwerkgröße. 67

Abbildungsverzeichnis

2.1 Visualisierung eines Fußballspiels in der Visualisierungsumgebung RoboViz 5
2.2 Robotermodell des Naos 6

3.1 Interaktionen zwischen Agent und Umgebung in einem Markow-Entscheidungsproblem. 9
3.2 Gewichtung der erwarteten Rewards abhängig von γ 12
3.3 Interaktionen zwischen Agent, Critic und Umgebung im Actor-Critic-Modell. 17
3.4 clip+e 19
3.5 clip-e 20
3.6 Veranschaulichung des Bewegungsablaufs des KickWalk-Problems . 25
3.7 Veranschaulichung des Bewegunsablaufs des StabilizedKick-Problems 26

4.1 Sequentieller Ablauf des KickWalk-Lernproblems mit 24 Instanzen 31
4.2 Paralleler Ablauf des KickWalk-Lernproblems 33
4.3 Anstieg des Durchsatzes anhand steigender Anzahl an parallelen Instanzen 34
4.4 Vergleich 24 paralleler Instanzen mit asynchronen und synchronen Lernen 36
4.5 Vergleich 24 paralleler Instanzen mit asynchroner und synchroner Lernmethode anhand durchgeführten Episoden 37
4.6 Vergleich Threshold-Modus 39
4.7 Anteil an validen Observations bei asynchroner Lernmethode 40
4.8 Vergleich finaler asynchroner Modus gegenüber Prototyp 42
4.9 Durchgeführte Episoden in 20 Minuten entlang der Synchronisation in % 43
4.10 Anteil an validen Observations bei asynchroner Lernmethode mit Grenzwert 75% 44
4.11 Vergleich unterschiedlicher Lernraten 46
4.12 Vergleich unterschiedlicher Nsteps-Größen 48
4.13 Vergleich unterschiedlicher nSteps-Größen anhand des KickWalk-Problems 49
4.14 Vergleich unterschiedlicher Minibatch-Größe 51

4.15 Vergleich unterschiedlicher Minibatch-Größe 52
4.16 Vergleich unterschiedlicher Episodenzahl anhand des KickWalk-Problems 54
4.17 Gegenüberstellung unterschiedlicher Werte des Entropy Coefficient anhand des KickWalk-Problems 55
4.18 Vergleich des Lernfortschritts anhand Lernmodus mit gleichem Entropy Coefficient 57
4.19 Gegenüberstellung des Discounted Rewards anhand der steigenden Entropy des Systems 58
4.20 Vergleich Abweichung des Value Function Coefficients vom Standardwert 59
4.21 Vergleich unterschiedlicher Werte für den Hyperparameter Gamma . 60
4.22 Discount Factor Visualisierung 61
4.23 Vergleich unterschiedlicher Werte für λ 63
4.24 Vergleich unterschiedlicher Werte für den Clipping-Hyperparameter 64
4.25 Vergleich unterschiedlicher Netzwerkgrößen beim KickWalk-Problem 66
4.26 Bewegungsmuster des KickWalks bei unterschiedlicher Netzwerkgröße 68
4.27 Vergleich der Aktivierungsfunktion 69
4.28 Visualisierung der Standardabweichung der Werte des Observationspace 70
4.29 Vergleich des Standards zu kleinerem Observationspace 71
4.30 Vergleich des Lernfortschritts zwischen Standard und vortrainiertem Modell 72
4.31 Vergleich Kickresultat zwischen Baseline und PPO2 Kick 73
4.32 1000 Ballpositionen nach Ausführen des PPO2 gelernten Kicks . . . 74
4.33 Variable Startposition des Spielers 75
4.34 Heatmap des durchschnittlichen Rewards des genetisch gelernten Kicks 77
4.35 Vergleich des kickbaren Bereichs des genetischen gegen PPO2 Kick 78

5.1 Verhältnis zwischen validen und Dummy-Observations anhand Walk-Problem 80
5.2 Vergleich des unterschiedlichen Lernfortschritts des Walk-Problems anhand Einfluss der Dummy-Elemente 82
5.3 Vergleich des unterschiedlichen Lernfortschritts des Walk-Problems anhand identischem Setup 83
5.4 Vergleich des unterschiedlichen Lernfortschritts des Walk-Problems anhand des Gamma-Hyperparameters 84

Literatur

[1] *OpenAI Five Defeats Dota 2 World Champions*, https://openai.com/blog/openai-five-defeats-dota-2-world-champions/, Accessed: 2020-09-17.

[2] O. Obst und M. Rollmann, „Spark – A Generic Simulator for Physical Multi-agent Simulations“, in *Multiagent System Technologies*, G. Lindemann, J. Denzinger, I. J. Timm und R. Unland, Hrsg., Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, S. 243–257, ISBN: 978-3-540-30082-3.

[3] J. Fischer, „Laufenlernen eines humanoiden Roboters mit Zehen mit Hilfe von Machine Learning und Behavior Morphing“, de, Master Thesis, Offenburg, 2019, S. 84.

[4] K. Dorer, „Learning to Use Toes in a Humanoid Robot“, en, H. Akiyama, O. Obst, C. Sammut und F. Tonidandel, Hrsg., Ser. RoboCup 2017: Robot World Cup XXI, Heidelberg: Springer, 2017, S. 100–111.

[5] *SoftBank Robotics. NAO - Developer Documentation*, http://doc.aldebaran.com/2-8/family/nao_technical/index_naov6.html, Accessed: 2020-09-16.

[6] R. S. Sutton und A. G. Barto, „Reinforcement Learning: An Introduction Second Edition: 2014, 2015“, 2014.

[7] L. C. Melo, *Imitation Learning and Meta-Learning for Optimizing Humanoid Robot Motions*, 2019. Adresse: https://github.com/luckeciano/deep-rl-undergrad-thesis/blob/master/tese.pdf.

[8] A. Welbers, „Zeitdiskrete Markovsche Entscheidungsprozesse zur optimalen Steuerung von Versicherungsunternehmen : Der stationäre, nichtstationäre und adaptive Fall“, Diss., Universität Hamburg, 2007.

[9] M. Otterlo und M. Wiering, „Reinforcement Learning and Markov Decision Processes“, *Reinforcement Learning: State of the Art*, S. 3–42, Jan. 2012. DOI: 10.1007/978-3-642-27645-3_1.

[10] S. Ravichandiran, *Hands-On Reinforcement Learning with Python: Master Reinforcement and Deep Reinforcement Learning Using OpenAI Gym and TensorFlow*. Packt Publishing, 2018, ISBN: 1788836529.

[11] J. Schulman, P. Moritz, S. Levine, M. Jordan und P. Abbeel, *High-Dimensional Continuous Control Using Generalized Advantage Estimation*, 2015. arXiv: 1506.02438 [cs.LG].

[12] X. B. Peng, P. Abbeel, S. Levine und M. van de Panne, „DeepMimic: Example-guided Deep Reinforcement Learning of Physics-based Character Skills“, *ACM Trans. Graph.*, Jg. 37, Nr. 4, 143:1–143:14, Juli 2018, ISSN: 0730-0301. DOI: 10.1145/3197517.3201311. Adresse: http://doi.acm.org/10.1145/3197517.3201311.

[13] J. Schulman, S. Levine, P. Moritz, M. I. Jordan und P. Abbeel, *Trust Region Policy Optimization*, 2015. arXiv: 1502.05477 [cs.LG].

[14] J. Schulman, F. Wolski, P. Dhariwal, A. Radford und O. Klimov, *Proximal Policy Optimization Algorithms*, 2017. arXiv: 1707.06347 [cs.LG].

[15] R. J. Williams, „Simple statistical gradient-following algorithms for connectionist reinforcement learning“, *Machine Learning*, Jg. 8, S. 229–256, 1992.

[16] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, D. Silver und K. Kavukcuoglu, *Asynchronous Methods for Deep Reinforcement Learning*, 2016. arXiv: 1602.01783 [cs.LG].

[17] D. Masters und C. Luschi, *Revisiting Small Batch Training for Deep Neural Networks*, 2018. arXiv: 1804.07612 [cs.LG].

[18] *Stable Baselines Documentation*, https://stable-baselines.readthedocs.io/en/master/, Accessed: 2020-09-17.

[19] X. Dong, J. Shen, W. Wang, Y. Liu, L. Shao und F. Porikli, „Hyperparameter Optimization for Tracking with Continuous Deep Q-Learning“, in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2018, S. 518–527.

[20] J. K. H. Franke, G. Köhler, A. Biedenkapp und F. Hutter, *Sample-Efficient Automated Deep Reinforcement Learning*, 2020. arXiv: 2009.01555 [cs.LG].

[21] R. M. Kretchmar, „Parallel Reinforcement Learning“, in *In The 6th World Conference on Systemics, Cybernetics, and Informatics*, 2002.

[22] M. Thoma, *Analysis and Optimization of Convolutional Neural Network Architectures*, 2017. arXiv: 1707.09725 [cs.CV].

[23] N. S. Keskar, D. Mudigere, J. Nocedal, M. Smelyanskiy und P. T. P. Tang, *On Large-Batch Training for Deep Learning: Generalization Gap and Sharp Minima*, 2016. arXiv: 1609.04836 [cs.LG].

[24] Z. Ahmed, N. L. Roux, M. Norouzi und D. Schuurmans, *Understanding the impact of entropy on policy optimization*, 2018. arXiv: 1811.11214 [cs.LG].

[25] R. Islam, P. Henderson, M. Gomrokchi und D. Precup, *Reproducibility of Benchmarked Deep Reinforcement Learning Tasks for Continuous Control*, 2017. arXiv: 1708.04133 [cs.LG].

[26] B. Xu, N. Wang, T. Chen und M. Li, *Empirical Evaluation of Rectified Activations in Convolutional Network*, 2015. arXiv: 1505.00853 [cs.LG].

[27] H. Teixeira, T. Silva, M. Abreu und L. P. Reis, „Humanoid Robot Kick in Motion Ability for Playing Robotic Soccer“, in *2020 IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC)*, 2020, S. 34–39.

[28] P. Henderson, R. Islam, P. Bachman, J. Pineau, D. Precup und D. Meger, *Deep Reinforcement Learning that Matters*, 2017. arXiv: 1709.06560 [cs.LG].

[29] L. C. Melo, M. R. O. A. Maximo und A. M. da Cunha, *Bottom-Up Meta-Policy Search*, 2019. arXiv: 1910.10232 [cs.LG].

[30] J. Keuper und F.-J. Pfreundt, *Asynchronous Parallel Stochastic Gradient Descent - A Numeric Core for Scalable Distributed Machine Learning Algorithms*, 2015. arXiv: 1505.04956 [cs.DC].

[31] B. Recht, C. Re, S. Wright und F. Niu, „Hogwild: A Lock-Free Approach to Parallelizing Stochastic Gradient Descent“, in *Advances in Neural Information Processing Systems 24*, J. Shawe-Taylor, R. S. Zemel, P. L. Bartlett, F. Pereira und K. Q. Weinberger, Hrsg., Curran Associates, Inc., 2011, S. 693–701. Adresse: http://papers.nips.cc/paper/4390-hogwild-a-lock-free-approach-to-parallelizing-stochastic-gradient-descent.pdf.