Eine Einführung in Reinforcment Learning mit Gym von Open AI

vorgelegt von: Fabian Grüterich  
Matrikel-Nr.: 11125738  
Adresse: Sonnenschein 2  
 51688 Wipperfürth  
 fabian.grueterich@smail.th-koeln.de

eingereicht bei: Prof. Dr. Wolfgang Konen

Wipperfürth, 13.07 .2020

Erklärung

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

|  |  |  |
| --- | --- | --- |
|  |  |  |
| Ort, Datum |  | Rechtsverbindliche Unterschrift |

Inhalt

[Erklärung I](#_Toc47788348)

[Abbildungsverzeichnis IV](#_Toc47788349)

[Einleitung 1](#_Toc47788350)

[1 Was ist Gym? 2](#_Toc47788351)

[1.1 Die Plattform 2](#_Toc47788352)

[1.2 Testumgebungen 2](#_Toc47788353)

[1.3 Leaderbord 3](#_Toc47788354)

[2 Einführung in Reinforcment Learning 4](#_Toc47788355)

[2.1 Begriffserklärung Reinforcement Learning 4](#_Toc47788356)

[2.2 Arten von Reinforcement Learning Algorithmen 4](#_Toc47788357)

[2.3 Q-Learning 5](#_Toc47788358)

[3 Installation und Nutzung von Gym 7](#_Toc47788359)

[3.1 Installation 7](#_Toc47788360)

[3.2 Frozen Lake 8](#_Toc47788361)

[3.2.1 Problembeschreibung 8](#_Toc47788362)

[3.2.2 Leaderboard Algorithmus 8](#_Toc47788363)

[3.2.3 Entwicklung eines eigenen Algorithmus 9](#_Toc47788364)

[3.2.3 Vergleich der Algorithmen 10](#_Toc47788365)

[3.3 Mountain Car 11](#_Toc47788366)

[3.3.1 Problembeschreibung 11](#_Toc47788367)

[3.3.3 Q Learning Algorithmus von Genevieve Hayes 11](#_Toc47788368)

[3.3.4 Parametertests 12](#_Toc47788369)

[3.3.5 Test des trainierten Models 16](#_Toc47788370)

[3.3.6 Mountain Car Continuous 17](#_Toc47788371)

[3.4 Cart Pole 18](#_Toc47788372)

[3.4.1 Problembeschreibung 18](#_Toc47788373)

[3.4.2 Anwenden des Q-Algorithmus 19](#_Toc47788374)

[4 Fazit zu Gym 20](#_Toc47788375)

[Literaturverzeichnis 22](#_Toc47788376)

[Anhang 23](#_Toc47788377)

Abbildungsverzeichnis

[Abbildung 1: Eine Übersicht über die Einteilung von Reinforcment Learning Algorithmen 5](#_Toc519677586)

[Abbildung 2 : Parametertest der Episoden (blau=500, orange=1500, grün=2500, rot=10000) 11](#_Toc519677587)

[Abbildung 3: Parametertest der Lernrate (blau=0.1, orange=0.2, grün=0.5, rot=0.7) 11](#_Toc519677588)

[Abbildung 4: Parametertest des Discountfaktor (blau=0.5, orange=0.7, grün=0.9, rot=0.95) ………..12](#_Toc519677589)

[Abbildung 5: Parametertest der ε -Minwerte (blau=0.3, orange=0.5, grün=0, rot=-0.3) 12](#_Toc519677588)

[Abbildung 6: Parametertest des ε-Wertes(lila=0.1,blau=0.3, orange=0.5, grün=0.7, rot=0.8) 13](#_Toc519677588)

[Abbildung 7: Parametertest des ε-Wertes bei 10.000 Episoden (blau=0.1, orange=0.8)………….13](#_Toc519677588)

[Abbildung 8: Untersuchung der (blau=1, orange=2, grün=4, rot=8, lila=10) 14](#_Toc519677588)

[Abbildung 9: Durchschnittliche Belohnung mit dem trainierten Model mit der gleichen Q-Tabelle 1](#_Toc519677588)5

[Abbildung 10: Versuchsdurchlauf des Q-Algorithmus in der Mountain Car Continuous Umgebung 16](#_Toc519677588)

[Abbildung 11: Versuchsdurchlauf in der Mountain Car Continuous Umgebung ohne Bestrafung 16](#_Toc519677588)

Einleitung

An der TH Köln gab es bislang noch keine Plattform, mit der man Reinforcement Learning lernen oder Reinforcement Learning Agenten testen konnte.

Ziel des Praxisprojektes war es Gym als eine solche Plattform an der TH Köln einzurichten. Aufgrund der Covid-19-Pandemie und der damit eingehenden Einschränkungen musste dieses Ziel auf die Installation auf einem privaten Rechner beschränkt werden.

Neben der Installation war ein weiteres Ziel dieses Projekts, Testumgebungen der Plattform mit Reinforcement Learning Agenten zu lösen. Damit sollte überprüft werden, wie gut sich Gym als Lernplattform von Reinforcement Learning eignet.

# Was ist Gym?

## Die Plattform

Gym ist ein von Open AI entwickeltes Toolkit für Reinforcement Learning. Es stellt neben einer Softwarebibliothek mit Testumgebungen eine Website zur Verfügung, auf der die Nutzer die Ergebnisse ihrer Reinforcement -Learning-Algorithmen teilen können.[[1]](#footnote-2)

## Testumgebungen

Es werden verschiedene Testumgebungen bereitgestellt, welche mit Reinforcement Learning gelöst werden können. Sie werden in die Bereiche Algorithmic, Atari, Box2d, Classic control, MuJoCo, Robotics und Toy text eingeteilt und im Folgenden kurz erläutert.[[2]](#footnote-3)

Die Algorithmic Problemstellungen umfassen Aufgaben zur Verarbeitung einer Eingabe, wie das Kopieren eines Strings oder dem Addieren von Zahlen.

Der Atari Bereich beinhaltet Atari 2600 Spiele, welche mithilfe von Reinforcement Learning gelöst werden können.

Box2D beschäftigt sich mit dem Bewegen eines 2D Objekts über Hindernisse oder in einen Zielbereich.

In Classic Control geht es auch um das Bewegen eines 2D Objekts, jedoch zum Balancieren eines Gegenstands oder Beschleunigen des Objekts, um einen Überschlag zu erreichen oder mit Schwung eine Anhöhe zu erklimmen.

Bei MuJoCo werden verschiedene Aufgaben in einer Physiksimulation gestellt. Sie beinhalten neben dem Ausbalancieren eines Gegenstandes das Erlernen menschlicher Bewegungen wie Gehen, Aufstehen und Schwimmen mit 2- oder 3-dimensionalen Figuren.

Die Robotics-Umgebungen stellen verschiedene Aufgaben für einen Roboterarm und eine Roboterhand zur Verfügung.

Der Toy Text Bereich behandelt Textaufgaben, nach deren Anweisungen ein Ziel in der dazugehörigen Umgebung erreicht werden soll. Die Umgebungen sind nicht 2- oder 3-dimensional simuliert, sondern besteht aus Buchstabenfeldern, welche für Objekte aus dem Text stehen und entsprechende Eigenschaften besitzen.

Neben diesen Umgebungen ist es möglich, eigene Umgebungen zu erstellen und diese den anderen Nutzern bereitzustellen.[[3]](#footnote-4) Diese werden unter dem Punkt „Third Party Environments“ auf der Website

<https://github.com/openai/gym/blob/master/docs/environments.md#third-party-environments> aufgeführt.

Der Code der einzelnen Umgebungen ist in Python geschrieben und im Repository „openai/gym“ auf Github einzusehen. Zudem ist es möglich dort Änderungsvorschläge für die Umgebungen einzubringen und Probleme zu melden, die beim Nutzen dieser Umgebungen aufgekommen sind.

Eine Übersicht aller Umgebungen ist unter <https://github.com/openai/gym/wiki/Table-of-environments> zu finden.

## Leaderbord

Auf dem Leaderboard unter <https://github.com/openai/gym/wiki/Leaderboard> werden Ergebnisse von Nutzern veröffentlicht, welche eine Testumgebung gelöst haben. In den meisten Fällen ist auch der Code der Agenten dazu veröffentlicht, sodass dieser auch von anderen getestet und genutzt werden kann. Die Veröffentlichung der Agenten soll einen Vergleich der Lerngeschwindigkeit der Agenten bieten, mit der diese das Problem lösen.[[4]](#footnote-5)

# Einführung in Reinforcment Learning

## Begriffserklärung Reinforcement Learning

Die folgende Begriffserklärung von Reinforcement Learning beruht auf den Ausführungen von [Genevieve Hayes](https://towardsdatascience.com/@gkhayes?source=post_page-----c289aca874f----------------------) in ihrem, auf der Plattform Medium veröffentlichten, Artikel „Reinforcement Learning:The Other Type of Machine Learning“[[[5]](#footnote-6)] :

Bei Reinforcement Learning Problemen liegt eine unbekannte Umgebung vor, über die Informationen gesammelt wird, indem Aktionen in dieser Umgebung ausgeführt werden und die Konsequenzen beobachtet werden. Dabei können die gesamten Konsequenzen einer Aktion auch erst im späteren Verlauf der Problemlösung bekannt werden.

Ziel ist es eine Aktionsfolge zu finden, welche das Problem optimal löst.

## Arten von Reinforcement Learning Algorithmen

Eine der wichtigsten Unterscheidungspunkte von Reinforcement Learning ist die Unterscheidung in Model-Free und Model-Based Algorithmen.[[6]](#footnote-7)

Model-Based Algorithmen besitzen eine Funktion, welche Zustandsübergänge und Konsequenzen von Aktionen vorhersagen können. Diese Funktion ist in den meisten Fällen nicht gegeben und muss daher von dem Algorithmus gelernt werden. Die Funktion kann zu einer signifikanten Verbesserung der Ergebnisse gegenüber Model-Free Algorithmen führen, aber auch zu schlechteren Ergebnissen, wenn ein Modell gelernt wird, das nicht mit der eigentlichen Umgebung übereinstimmt.[[7]](#footnote-8)

Bei Model-Free Algorithmen gibt es eine Unterteilung in Policy Optimization und Q-Learning. Dabei nutzen Policy Optimization Algorithmen immer nur die Aktionen, welche nach der aktuellen Strategie die beste Wahl sind. Q-Learning hingegen nutzt auch zufällige Aktionen. Diese scheinen bei der aktuellen Strategie nicht der optimale Wert zu sein, können sich jedoch im späteren Verlauf als beste Wahl herausstellen, da über sie die optimale Strategie gefunden wurde. Dabei scheint Policy-Optimization stabiler zu laufen, jedoch weniger effizient als Q-Learning.[[8]](#footnote-9)

Ein Beispiel soll den Unterschied der beiden Vorgehen noch genauer erläutern.

Ein Agent muss sich zwischen zwei Wegen von seinem Startpunkt zu einem Zielpunkt bewegen. Auf diesen Wegen erhält er zwei Belohnungen, von denen nur die erste bekannt ist.

Auf dem ersten Weg erhält er zunächst eine Belohnung von 10 und danach eine Belohnung von 1. Auf dem zweiten Weg erhält er zuerst eine 1 als Belohnung und als zweites eine 1000.

Bei der Policy Optimization würde bei jedem Durchlauf die erste Option gewählt, da die Belohnung von 10 größer ist als die 1.

Bei einem Q-Learning-Algorithmus würde zunächst auch die erste Option gewählt. Wird bei einem zufälligen Schritt jedoch die zweite Strategie ausgewählt, wird erkannt, das sie insgesamt eine höhere Belohnung liefert und bei den nicht zufälligen Schritten als Option gewählt.

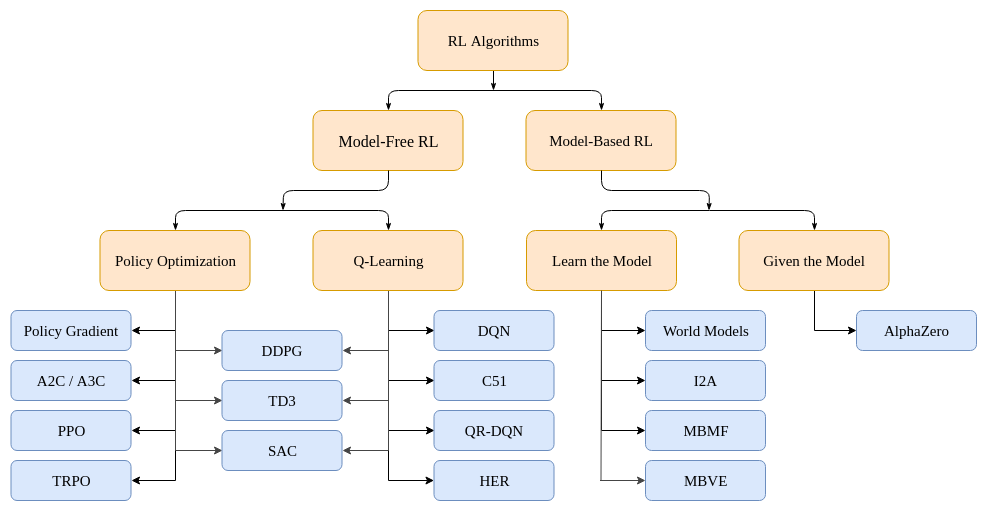


Abbildung 1: Eine Übersicht über die Einteilung von Reinforcement Learning Algorithmen [[[9]](#footnote-10)]

## Q-Learning

Da das Mountain Car Problem im Abschnitt 3.3 mit einem Q-Learning Algorithmus gelöst wurde, wird das Konzept im Folgenden genauer erläutert. Die Erklärungen entstammen den Ausführungen von [Genevieve Hayes](https://towardsdatascience.com/@gkhayes?source=post_page-----c289aca874f----------------------) in ihrem Artikel „Reinforcement Learning: The Other Type of Machine Learning“:

Für Q-Learning Algorithmen werden Tupel *Q(s,a)* aus den möglichen Zuständen der Umgebung *s* und den möglichen Aktionen *a* in diesen Zuständen gespeichert, zum Beispiel in einer Tabelle oder einem Neuronalen Netz.

Nach jeder Aktion wird der Zustand *s* vor der Aktion, die Aktion *a,* die Belohnung *r* und der neue Stand s‘ betrachtet. Mit diesen Informationen wird der Wert des Tupels *Q(s,a)* nach folgender Regel in der Tabelle mit einem neuen Wert belegt:

*Q‘(s,a) = (1 — w) \* Q(s, a) + w \*(r + d \* Q(s’, argmax a’ : Q(s’, a’)))*

Dabei steht *w* für die learning rate, welche festlegt, wie hoch neue Informationen für die Gesamtstrategie gewertet werden. *d* steht für die discount rate.

Sobald eine optimale Aktionsfolge gefunden wurde, findet sich die optimale Aktion *a* an einer Stelle *s* in der Q-Tabelle bei dem Eintrag *argmax a: Q(s, a).*

Um diese optimale Aktionsfolge zu finden, muss eine Strategie gewählt werden, wann die besten Ergebnisse untersucht und wann neu Aktionsfolgen ausprobiert werden.[[10]](#footnote-11) Für das Mountain Car Problem wurde dafür die Epsilon-Greedy-Strategie verwendet.

# Installation und Nutzung von Gym

Nach den Erläuterungen zu Gym und Reinforcement Learning folgt nun die Dokumentation des praktischen Teils des Projektes, wo Gym installiert und getestet wurde.

## Installation

Um Gym installieren zu können, muss auf dem Rechner Pyhton in der Verion 3.5 oder in einer höheren Version installiert sein.[[11]](#footnote-12) Zudem wird eine Entwicklungsumgebung benötigt, in welcher die Testumgebungen aufgerufen und die Reinforcement Learning Agenten geschrieben werden können. Eine Installationsanleitung für Gym wird unter „<https://gym.openai.com/docs/>“ angeboten.

Eine ausführlichere Anleitung wir von [Genevieve Hayes](https://towardsdatascience.com/@gkhayes?source=post_page-----c289aca874f----------------------) in einem Beitrag auf der Plattform Medium geliefert.[[12]](#footnote-13) Hier wird genauer erläutert wie die einzelnen Bereiche von Umgebungen installiert werden können, anstatt sie wie in der offiziellen Dokumentation eine vollständige Installation über das Klonen des Repositories vorzunehmen. Zudem wird auch eine Anleitung zur Installation von „Xming“ gegeben. Mit diesem können die gerenderten Umgebungen in einem separaten Fenster angezeigt werden.

Die Installation für das Projekts wurde auf einem Windows Rechner durchgeführt. Als Entwicklungsumgebung wurde Spyder in Verbindung mit der Data Science Plattform Anaconda gewählt.

Nach dem Befolgen der Installationsanleitung wurden alle Umgebungen mit einem Agenten getestet, welcher jeweils eine zufällige Aktion ausführt. Dieser Test ergab, dass alle Umgebungen aus den Bereichen Toy Text, Classic Control und Box 2D funktionierten.

Bei Algorihmic gab es für die Umgebungen eine Fehlermeldung, da die zufälligen Aktionen dazu führten, dass der Bereich der zu verarbeiteten Eingabe verlassen wurde. Dieses Problem ließ sich jedoch lösen, wenn man vor jedem Schritt abfragt, ob der Schritt für den Rückgabewert done true zurückgibt und in diesem Falle abbricht.

Bei den Atariumgebungen kam es zu der Fehlermeldung „module could not be found“. Für dieses Problem ließ sich unter „<https://github.com/openai/gym/issues/1726>“ eine Lösung von dem Nutzer „razzzu“ in seinem Post vom 07.11.2019 finden, sodass diese Umgebungen danach auch funktionierten.

Die Umgebungen von Mujuco und Robotics nutzen beide die Mujoco physic engine. Diese ist nur mit einer (für Studenten kostenlosen) Lizenz nutzbar und der Support für Windows wurde ab Version [1.50.1.68](https://github.com/openai/mujoco-py/blob/9ea9bb000d6b8551b99f9aa440862e0c7f7b4191/README.md#requirements). eingestellt. Daher wurden diese Umgebungen nicht weiter getestet.

## Frozen Lake

Als erste Umgebung wurde für das Frozen Lake eine Lösung gesucht. Dieses Problem ist bei den Toy Text Umgebungen eingeordnet.

### Problembeschreibung

In der Frozen Lake Umgebung liegt ein 4 mal 4 großes Feld vor. An der ersten Stelle ist ein Startpunkt und an der letzten Stelle ein Zielpunkt. Die restlichen Felder sind Loch-Felder, welche eine negative Belohnung geben, und Gefroren-Felder. Diese Einteilung ist fest und verändert sich nicht während eines Durchlaufs und auch nicht nach einem Zurücksetzen der Umgebung. Ziel ist es einen Weg von dem Startpunkt zu dem Zielpunkt zu finden. Dabei startet der Agent auf dem Startfeld und kann sich auf die angrenzenden Felder bewegen. Führt er eine Aktion aus, die ihn aus den 16 Feldern bewegen würde, wie ein Schritt nach links in der linken Spalte, bleibt er stattdessen stehen. Bewegt er sich auf ein Loch-Felder, kann sich der Agent nicht mehr von dem Feld wegbewegen. Nur durch ein Zurücksetzen der Umgebung gelangt den Agenten wieder zurück auf den Startzustand, von dem er sich wieder normal bewegen kann. Jeder ausgeführte Schritt in eine Richtung besitzt eine gleich große Wahrscheinlichkeit zu den Seiten zu gehen. So kann die Aktion „nach links gehen“ die Ergebnisse „nach links gegangen“, „nach oben gegangen“ und „nach unten gegangen“ haben. Nur das Gegenteil der Aktion (in dem Beispiel „nach rechts gegangen“) kann nicht auftreten. Nach jedem Schritt wird die Position auf dem 4x4Feld, die Belohnung, die Information ob das Ziel erreicht wurde und die Wahrscheinlichkeit, mit der man dieses Feld erreichen konnte, zurückgegeben.

### Leaderboard Algorithmus

Zu Beginn wurde versucht, selbstständig einen Algorithmus für das Problem zu entwickeln. Jedoch konnte zunächst keine Erklärung für die Abweichungen der tatsächlichen Aktionen von den gewählten gefunden werden, sodass keine Lösungsstrategie gefunden werden konnte.

Um sich einen Überblick über mögliche Problemlösungen zu verschaffen, wurde somit eine Lösung auf dem Leaderboard gesucht. Für die Frozen Lake Umgebung war auf dem Leaderboard eine Lösung veröffentlicht.[[13]](#footnote-14)

Dort wird eine Reihenfolge von vier Aktionen zufällig ausgewählt und diese werden nacheinander ausgeführt. Dabei ist die erste, zweite und vierte Aktion auf die Möglichkeiten „nach links gehen“, „nach rechts gehen“ und „nach unten gehen“ beschränkt, Die dritte Aktion ist entweder „nach links gehen“ oder „nach unten gehen“. Nachdem die vier Aktionen aufgeführt wurden, werden vier neue Aktionen ausgewählt. Dies geschieht so lange bis sich der Agent auf eine Lochposition oder die Zielposition bewegt. Im Falle des Ziels wird die Routine abgebrochen und der Algorithmus beendet. Im Falle eines Loches wird die Umgebung zurückgesetzt. Dem Algorithmus sind die Zielposition und die Lochposition zu Beginn bekannt, damit er die aktuelle Position mit diesen Positionen abgleichen kann.

### 3.2.3 Entwicklung eines eigenen Algorithmus

Nach dem Anwenden des Leaderboard-Algorithmus und mit den gewonnenen Erkenntnissen über die Umgebung wurde anschließend ein eigener Algorithmus für das Frozen Lake Problem entwickelt. Ziel war es dabei ohne vorherige Kenntnis über die Position der Löcher und des Zieles zu arbeiten und eine Aktion nur zufällig zu wählen, wenn nichts über die angrenzenden Felder bekannt ist.

Der entwickelte Algorithmus besitzt ein zweidimensionales Array, in dem jeder Eintrag für eine Position in der Umgebung steht. Dort werden die Löcher und das Ziel gespeichert, sobald diese von dem Agenten betreten wurden. Dieses Array wird zu Beginn übergeben, sodass ein Agent die Einträge des vorherigen Agenten nutzen kann. Der erste Agent bekommt ein Array ohne eingetragene Löcher oder Zielen übergeben.

Der Agent wählt zunächst eine zufällige Aktion aus. Befindet sich der Agent nach der Aktion in einem Loch, wird die Umgebung zurückgesetzt. Der Agent kann erkennen, dass er sich in einem Loch befindet, wenn die Wahrscheinlichkeit für die Position 100% beträgt. Diese Wahrscheinlichkeit wird bei dem vorherigen Schritt zurückgegeben, da der Agent sich nicht aus dem Loch bewegen kann und somit keine andere Position betreten haben kann. Seine Aktion wird überschrieben, wenn Informationen über die angrenzenden Felder in dem Array abgespeichert sind.

Dabei wird zunächst überprüft ob ein angrenzendes Feld als Loch abgespeichert ist. In diesem Fall wird als Aktion die Richtung gewählt, die in die entgegengesetzte Richtung zum Loch führt. Damit wird sichergestellt, dass die Aktion in keinem Fall in das Loch führt und die Umgebung zurückgesetzt werden müsste.

Wenn sich der Agent an der gleichen Position befindet, wie vor seinem letzten Schritt und sich nicht in einem Loch befindet, hat er sich gegen eine Wand bewegt. Daher wird über die Position bestimmt, an welcher Wand sich der Agent befindet und sich in die Richtung von der Wand wegbewegt.

Für den Fall, dass der Agent sich an einer Position befindet, an der keine Löcher oder Wände angrenzen und er das Ziel in dem Array übergeben bekommen hat, wird eine Richtung ausgewählt, in der sich das Ziel befindet.

Wird nach dem Schritt die Belohnung 1 zurückgegeben wurde das Ziel erreicht. Dann bricht der Algorithmus ab und speichert das Ziel in dem Array. Zuletzt wird dieses Array übergeben, um es anderen Agenten zur Verfügung zu stellen.

Der Code ist unter „<https://github.com/Fgrueter/Praxisprojekt>“ als „FrozenLake.py“ veröffentlicht.

### Vergleich der Algorithmen

Der eigene Algorithmus und der Algorithmus des Leaderboards wurden auf die Anzahl der Schritte überprüft, die sie zum Erreichen des Zieles benötigen. Dabei wurde der eigene Algorithmus zweimal laufen gelassen. Der erste Durchlauf bekam ein leeres Array übergeben, der zweite Durchlauf das Array des Ersten.

Der Leaderboard-Algorithmus benötigte im Durchschnitt 233 Aktionen, um das Ziel zu erreichen. Dem gegenüber benötigte der eigene Algorithmus im ersten Durchlauf 104 Aktionen durchschnittlich, im zweiten Durchlauf durch das Vorwissen des vorherigen Agenten 80 Aktionen und war somit fast 3-mal schneller.

Dieser Effekt verstärkte sich, als beide Algorithmen in der Frozen-Lake-8x8-Umgebung getestet wurden. Diese entspricht der Frozen-Lake-4x4-Umgebung, besitzt nur entsprechend mehr Felder.

In dieser Umgebung wurden von dem Leaderboard-Algorithmus durchschnittlich 25.651 Aktionen benötigt. Der eigene Algorithmus benötigte dagegen 1506 und 638 Aktionen und war somit mehr als 10-mal schneller als der Algorithmus.

Neben dem schnelleren Erreichen des Zieles, ist der eigene Algorithmus auch leichter auf andere Probleme dieses Typs anwendbar. Er benötigt keine Eingabe der Löcher und Ziele, sondern nur eine rechteckige Umgebung und ein Array mit den Dimensionen der Umgebung.

Der entwickelte Algorithmus ist im engeren Sinne kein richtiger Reinforcement-Learning-Algorithmus, da die Schritte nicht nach der zu erwarteten Belohnung ausgewählt werden, sondern nur mit den Informationen über die Umgebungen arbeiten, jedoch wird die Umgebung im Laufe der Aktionen von dem Algorithmus gelernt. Zudem wäre ein klassischer Reinforcement-Learning-Algorithmus in dieser Umgebung nicht funktionsfähig. Die Belohnung ist, außer bei dem Ziel, immer gleich groß, sodass sich danach nicht gerichtet werden kann. Durch die Unsicherheit bei den Bewegungen kann zudem kein optimaler Weg bestimmt werden, da der Algorithmus selbst bei der richtigen Aktionsauswahl zufällig in einem Loch landen kann, welches bei einem klassischen Reinforcement-Learning-Algorithmus bestraft werden müsste.

In den folgenden Umgebungen waren klassische Reinforcement-Learning-Algorithmen anwendbar, da jeder Schritt ein eindeutiges Ergebnis hervorbrachte.

## Mountain Car

### Problembeschreibung

Nach den Frozen Lake Umgebungen wurde das Mountain Car Problem betrachtet. Dieses befindet sich in dem Bereich Classic Control. In dieser zweidimensionalen Umgebung befindet sich ein Auto im Tal zwischen zwei Anhöhen und muss die Spitze der rechten Anhöhe erreichen. Dabei kann es nach links, rechts und auf der Stelle beweget werden. Dabei kann sich das Auto nicht aus eigener Kraft den Berg hoch bewegen, sondern benötigt den Schwung, der sich ergibt, wenn es einen der Berge herabfährt.[[14]](#footnote-15) Für jede Aktion wird eine Belohnung von -1 gegeben. Wenn das Ziel erreicht wurde oder 200 Schritten absolviert wurden, wird die Umgebung zurückgesetzt. Nach jeder Aktion werden die aktuelle Position und Geschwindigkeit als Status, die Belohnung und die Information, ob das Ziel erreicht wurde, zurückgegeben.

### Q Learning Algorithmus von [Genevieve Hayes](https://towardsdatascience.com/@gkhayes?source=post_page-----c289aca874f----------------------)

Für das Mountain Car Problem wurde der Q-Learning-Algorithmus verwendet, welcher von [Genevieve Hayes](https://towardsdatascience.com/@gkhayes?source=post_page-----c289aca874f----------------------) auf Medium für dieses Problem veröffentlicht wurde.[[15]](#footnote-16) Da der Bereich an möglichen Zuständen kontinuierlich ist, wurde dieser auf 285 Zustände diskretisiert. Es wurde eine Q-Tabelle mit 855 möglichen Einträgen erstellt.[[16]](#footnote-17) Die 855 Einträge ergeben sich aus 19 mögliche Zustände für die Position mit jeweils 15 möglichen Geschwindigkeiten und dabei 3 möglichen Aktionen, die dort gewählt werden können. Die Q-Tabelle wurde zu Beginn gleichverteilt mit Werten zwischen -1 und 1 initialisiert.

Während des Trainings setzt der Algorithmus auf eine Epsilon-Greedy-Strategie. Dabei wird ein ε-Wert zwischen 0 und 1 festgelegt. Für jeden Schritt wird überprüft, ob eine Zufallszahl zwischen 0 und 1 in dem Bereich 0 bis 1-ε liegt. Falls dies zutrifft wird die Aktion aus der Q-Tabelle ausgewählt, welche für den aktuellen Status den maximalen Wert aufweist. Ansonsten wird eine zufällige Aktion ausgewählt. Nach jedem Zurücksetzen der Umgebung wird ε verringert. Der Wert, um den es verringert wird, hängt linear von der Anzahl der Durchläufe des Algorithmus ab. Wird zu Beginn der Durchläufe ein großes ε gewählt, besteht eine höhere Wahrscheinlichkeit dafür, eine zufällige Aktion durchzuführen. Durch die Verringerung von ε steigt die Wahrscheinlichkeit zum Ende hin, den maximalen Wert aus der Q-Tabelle auszuwählen. Damit soll erreicht werden, dass zu Beginn möglichst viele Zustände besucht werden und der Agent daraus den schnellsten Weg zum Ziel ermitteln kann. Dieser Weg besitzt nachher den maximalen Wert in der Q-Tabelle und wird zum Ende häufiger ausgewählt.

### Parametertests

Der gewählte Algorithmus besitzt fünf Parameter, die ihm übergeben werden. Diese sind die Lernrate, der Discountfaktor, ε, Min- ε und die Anzahl der Episoden. Die Lernrate und der Discountfaktor werden bei dem Update der Q-Tabellenwerte genutzt. ε wird für die Epsilon-Greedy-Strategie benötigt. Min- ε ist der Wert, auf welchen ε maximal verringert werden darf. Die Anzahl der Episoden bestimmt, wie lange der Code läuft. In jeder Episode hat der Agent maximal 200 Aktionen zur Verfügung, um das Ziel zu erreichen. Mit den übergebenen Werten für die Parameter wurde das Problem gelöst. Betrachtete man jedoch die durchschnittliche Belohnung über jeweils 100 Episoden ergab sich ein Maximum von -169.26 für die letzten 100 Episoden. Da für jede Aktion eine Belohnung von -1 vergeben wird, besagt dieser Wert, dass nach durchschnittlich 169 Aktionen das Ziel erreicht wurde. Mit der Anpassung der Parameter wurde anschließend versucht, diesen Wert zu verringern.

In den folgenden Diagrammen werden zum Vergleich der Parameter die durchschnittlichen Belohnungen über jeweils 100 Episoden zum Zeitpunkt der einzelnen Episoden dargestellt. Dabei entspricht eine Episode einem Durchlauf des Algorithmus (maximal 200 Schritte). Somit zeigt sich im Graphen wie schnell der Algorithmus über mehrere Durchläufe hinweg lernt und wie sich die durchschnittliche Schrittanzahl während den Durchläufen verbessert oder verschlechtert.

Als erster Parameter wurde die Episodenanzahl überprüft. Für diesen Parameter wurden die Werte 500,1500,2500 und 10000 getestet.

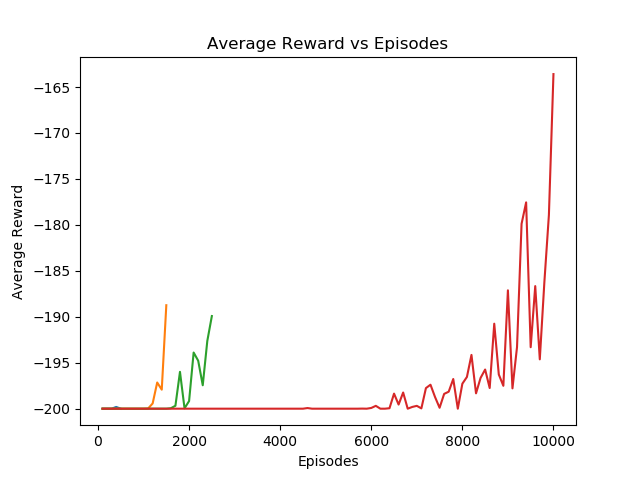


Abbildung 2: Parametertest der Episoden (blau=500, orange=1500, grün=2500, rot=10000) [[[17]](#footnote-18)]

Dabei zeigte sich für eine Episodenanzahl von 10.000 die besten Ergebnisse. Da der Graph bei diesem Wert jedoch sehr stark schwankte, wurden zur Übersichtlichkeit die anderen Parameter im weiteren Verlauf mit dem zweitbesten Ergebnis von 1500 Episoden getestet.

Der zweite untersuchte Parameter war die Lernrate.

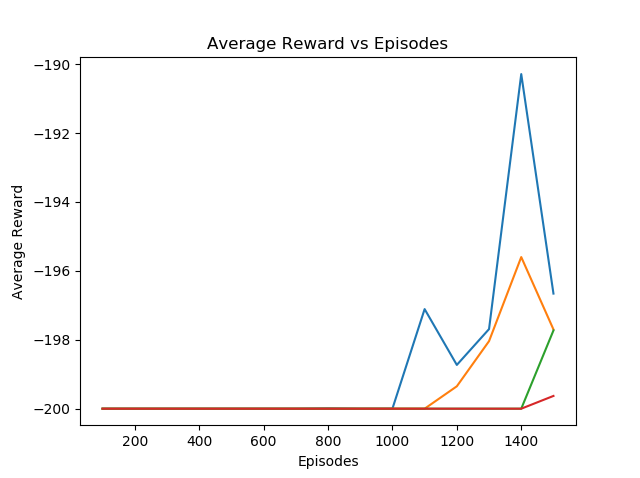


Abbildung 3: Parametertest der Lernrate (blau=0.1, orange=0.2, grün=0.5, rot=0.7) [[[18]](#footnote-19)]

Bei diesem Parameter zeigte sich, dass eine Lernrate von 0.1 die besten Ergebnisse ergab.

Der nächste Parameter war der Discountfaktor.

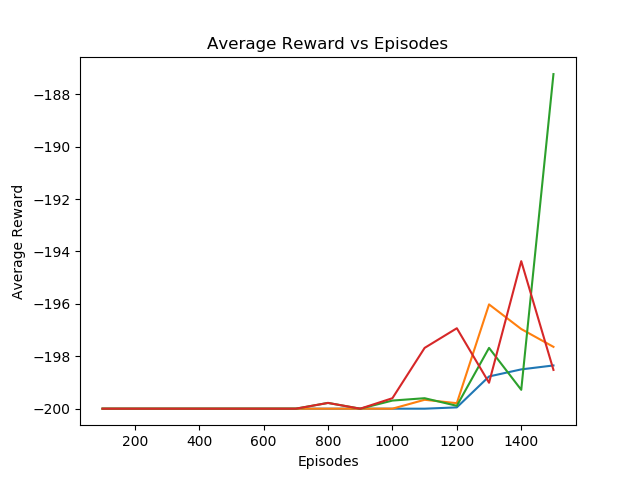


Abbildung 4: Parametertest des Discountfaktor (blau=0.5, orange=0.7, grün=0.9, rot=0.95) [[[19]](#footnote-20)]

Bei diesem Parameter ergab sich für den 0.9 die besten Ergebnisse.

Danach wurde untersucht, ob ein anderer Wert für Min- ε das Ergebnis verbessert.

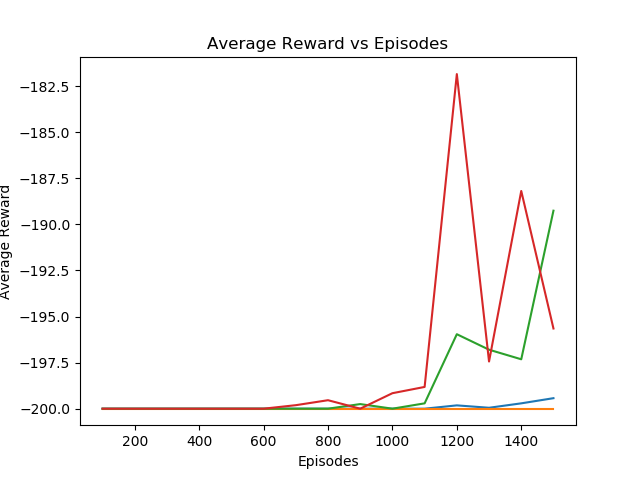


Abbildung 5: Parametertest der ε -Minwerte (blau=0.3, orange=0.5, grün=0, rot=-0.3) [[[20]](#footnote-21)]

Für einen ε-Wert über 0 gibt es auch zum Ende der Episoden noch die Auswahl von zufälligen Werten, um eine bessere Aktionsfolge zu finden. Die Untersuchung des Parameters zeigte jedoch auf, das dies zu schlechteren Ergebnissen führt.

Zuletzt wurden verschiedene ε-Werte überprüft.

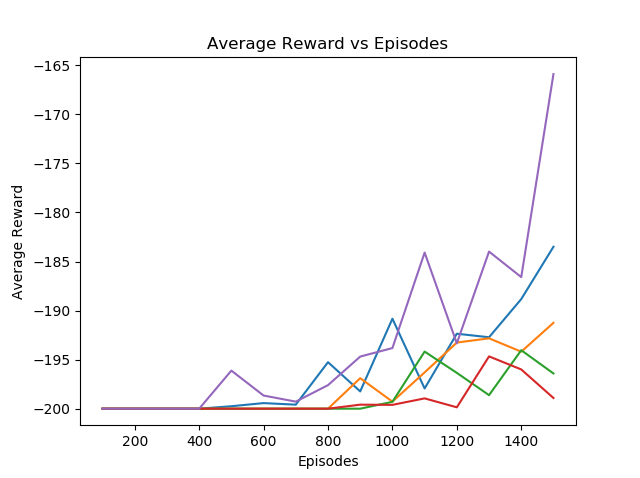


Abbildung 6: Parametertest des ε-Wertes (lila=0.1,blau=0.3, orange=0.5, grün=0.7, rot=0.8) [[[21]](#footnote-22)]

Bei einem kleineren ε-Wert wird früher der Weg mit den besten Ergebnissen befolgt und weniger nach einem alternativen Weg gesucht. Dies schien nach dem Test verschiedener Werte zu besseren Ergebnissen zu führen.

Als die ε -Wert jedoch auf 10.000 Episoden überprüft wurden, näherte sich der 0.8- ε mit seinem Ergebnis dem 0.1-ε-Wert immer näher an und besaß dabei einen robusteren Verlauf:

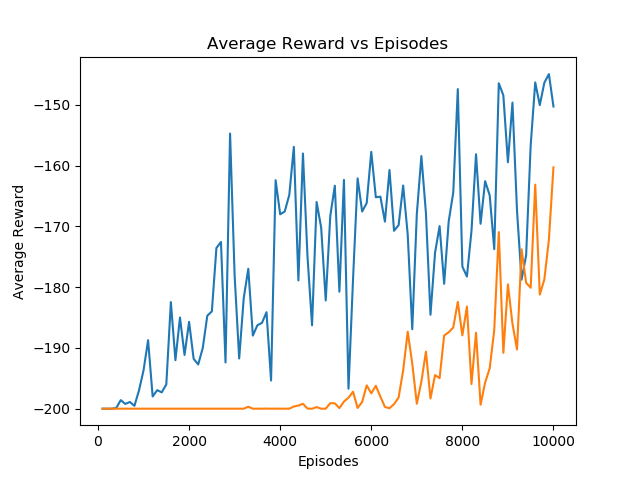


Abbildung 7: Parametertest des ε -Wertes bei 10.000 Episoden (blau=0.1, orange=0.8) [[[22]](#footnote-23)]

Somit wurde 0.8 im weiteren Verlauf gewählt.

Insgesamt ergaben sich aus den Tests die Werte Lernrate=0.1, Discountfaktor=0.9 ε -Min=0, ε =0.8 und Episoden=10000. Diese Ergebnisse entsprachen größtenteils den Werten, welche von [Genevieve Hayes](https://towardsdatascience.com/@gkhayes?source=post_page-----c289aca874f----------------------) gewählt worden sind. Damit ergab sich jedoch auch mit einer Belohnung von -168 gegenüber -169.26 ein ähnliches Ergebnis.

Daher wurde zuletzt ein Wert getestet, welcher nicht als Parameter übergeben wurde.

Die Untersuchung des ε -Wert hatte gezeigt, dass es zu besseren Ergebnissen führen konnte, wenn früher nicht mehr zufällige Aktionen ausgewählt werden. Anstatt jedoch ein dazu einen kleineren ε -Wert zu wählen, wurde dieser während des Durchlaufs schneller verringert, indem der Wert *h* in der Reduktionsformel von ε (*reduction=* (ε *-min\_*ε*)\*h/episoden*) erhöht wurde. Dieser Wert ist im ursprünglichen Code nicht vorhanden, da er in diesem Falle 1 beträgt.

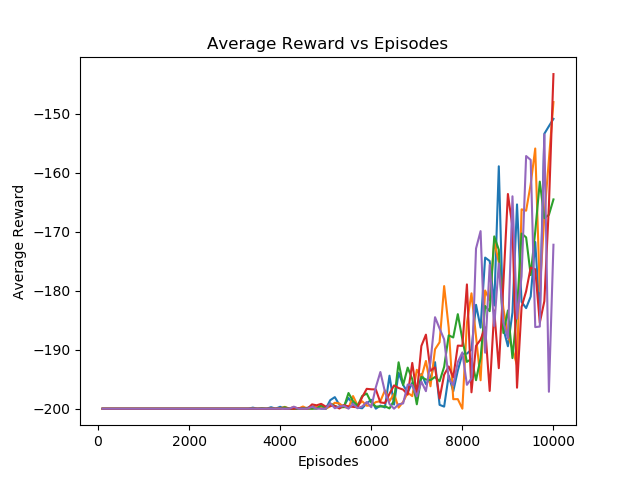


Abbildung 8: Untersuchung der (blau=1, orange=2, grün=4, rot=8, lila=10) [[[23]](#footnote-24)]

Mit dieser Änderung ließ sich bei einem Wert von h=8 ein Maximum von -143 erreichen.

### Test des trainierten Models

Nachdem der Algorithmus über 10.000 Episoden trainiert wurde, wurde im Folgenden getestet, wie gut die erstellte Q-Tabelle funktioniert. Dazu wurde eine Funktion „QModel“ geschrieben, welche die Q-Tabelle empfing. Damit wurde in dieser Funktion immer der beste Wert aus der Q-Tabelle für den Status des Mountain Cars ausgewählt und die Q-Tabelle nach einer Aktion nicht mehr geändert.

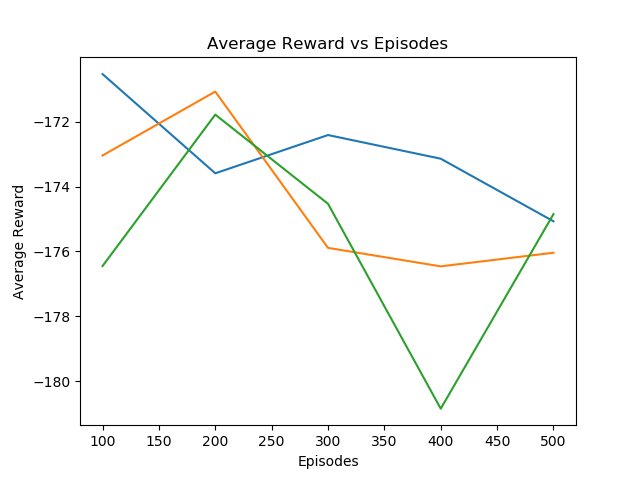


Abbildung 9: Durchschnittliche Belohnung mit dem trainierten Model mit der gleichen Q-Tabelle [[[24]](#footnote-25)]

Die Ergebnisse zeigten, dass das „QModel“ dreimal dazu in der Lage war, das Ziel zu erreichen. Dabei war auffällig, dass die Werte trotz gleicher Q-Tabelle und der jeweiligen Auswahl der besten Aktion voneinander abwichen. Diese ließ sich dadurch erklären, dass beim Zurücksetzen des Mountaincars die Position auf einen zufälligen Wert zwischen -0.6 und -0.4 gesetzt wird.[[25]](#footnote-26)

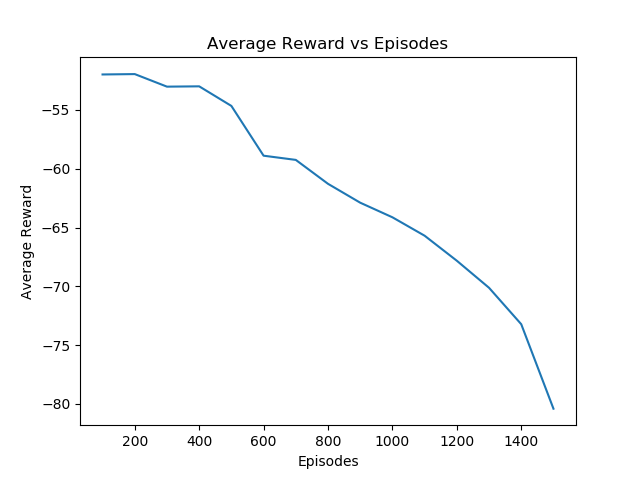
### Mountain Car Continuous

Nachdem der Q-Algorithmus in der Mountain Car Umgebung das Ziel erreichen konnte, wurde er in der Mountain Car Continuous Umgebung getestet.

Diese entspricht in den meisten Punkten der Mountain Car Umgebung. Die größten Unterschiede befinden sich bei den Aktionsmöglichkeiten und den zurückgelieferten Belohnungen.

Als Eingabe für die Aktionen des Mountaincars sind nicht mehr allein die Werte -1,0 und 1 möglich, sondern alle Zahlen im Bereich von -1 bis 1. Die Belohnung wurde von -1 auf 0 geändert und auf 100 für den Fall, dass das Ziel erreicht wurde. Um dennoch zu viele Schritte zu bestrafen, wird der Reward entsprechend der Aktion verringert.[[26]](#footnote-27)

Das Anwenden des Algorithmus ergab bei 1500 Episoden das folgende Ergebnis:

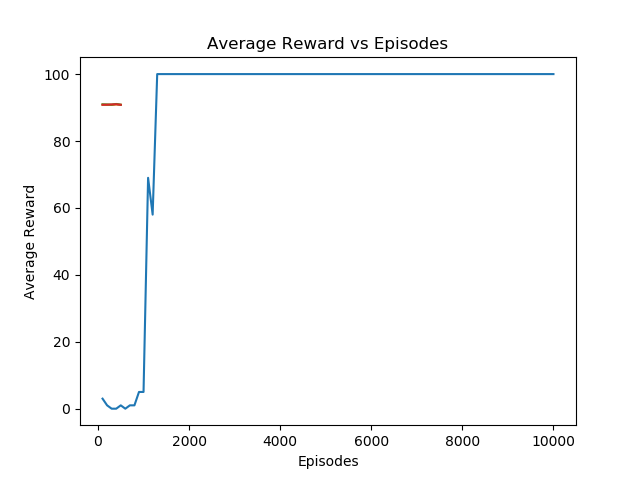


Abblíldung 10: Versuchsdurchlauf des Q-Algorithmus in der Mountain Car Continuous Umgebung [[[27]](#footnote-28)]

Durch die Bestrafung hoher Aktionswerte wurden zunehmend kleine Werte ausgewählt, sodass sich das Mountaincar nur noch in dem Tal bewegte. Dadurch führte jede Aktion zu einer weiteren Verschlechterung des Ergebnisses.

Damit das Mountaincar auch größere Werte verwendet und sich aus dem Tal herausbewegen konnte, wurde die erhaltene Belohnung im nächsten Versuch um den Wert der Bestrafung wieder erhöht. Durch diese Änderung wird, die im Code der Umgebung festgelegte Bestrafung, ignoriert, jedoch das zu lösende Problem nicht verändert.

Mit dieser Änderung ergab sich das folgende Ergebnis:



Abblíldung 11: Versuchsdurchlauf in der Mountain Car Continuous Umgebung ohne Bestrafung [[[28]](#footnote-29)]

Es zeigt sich, dass das Ziel schnell und konstant erreicht wird. Die Belohnung des „QModels“ (in der oberen Abbildung in Rot gezeichnet) beträgt auch konstant über 90.

## Cart Pole

Nachdem der Q-Algorithmus für beide Varianten des Mountain Car Problems erfolgreich war, wurde es zuletzt in der Cart Pole Umgebung getestet.

### Problembeschreibung

Cart Pole ist eine Umgebung, welche zu Classic Control Umgebungen gehört. In dieser Umgebung muss ein Stab horizontal auf einem Cart balanciert werden. Dabei kann sich das Cart nach links und rechts bewegen. Für jeden Schritt, bei dem der Stab sich nicht um 15 Grad neigt und sich das Cart nicht aus dem Bereich der Umgebung bewegt, bekommt es als Belohnung 1 zurückgegeben.[[29]](#footnote-30) Ziel ist es über 100 Episoden eine Durchschnittsbelohnung von 195 zu erhalten, wobei eine Episode 200 Schritte dauert, und ein Schritt, wie oben beschrieben, eine Belohnung von 1 liefert. Der Status des Carts wird durch die Cart Position, Cart Geschwindigkeit, Stab Winkel und Stab Winkel Veränderung beschrieben. Dabei können die Geschwindigkeit und die Veränderung des Winkels jeden beliebigen Wert annehmen.[[30]](#footnote-31)

### Anwenden des Q-Algorithmus

Um den Q-Learning Algorithmus anwenden zu können, musste der Bereich der möglichen Zustände wieder beschränkt werden, da ansonsten unendlich viele Einträge in der Q-Tabelle enthalten sein müssten.

Bei einer Größe der Q-Tabelle von 8\*20 Einträgen für die Position, 10\*20 Einträgen für die Geschwindigkeit, 20 Einträgen für den Winkel des Stabs , 30\*20 Einträgen für die Veränderungsgeschwindigkeit des Stabs (für jeden beobachtbare Eigenschaft 20 Möglichkeiten) und 2 Aktionsmöglichkeiten, also insgesamt 768,000,000 Einträgen lieferte der Algorithmus über einen Zeitraum von 10000 Episoden nur eine durchschnittliche Belohnung von 22-24 und konnte somit weder das Ziel noch einen Lernerfolg erreichen.

Bei einer Vergrößerung der Tabelle kam es Abstürzen des Programms und Warnungen von Spyder, dass nicht genug Speicherplatz verfügbar ist.

Bei einer Verkleinerung der Tabelle auf 8\*10\*20\*30\*2 Einträge ergab sich in einem Durchlauf zum Ende eine Belohnung von 60, jedoch in weitern Durchläufen Belohnungen zwischen 10 und 30, wobei die Belohnung zum Ende hin jeweils abfiel.

Somit scheint dieses Problem aufgrund seines großen Zustandraumes nicht mit einem tabellenbasierten Q-Learning-Algorithmus lösbar zu sein.

# Fazit zu Gym

Während der Projektdurchführung zeigten sich mehrere Vor- und Nachteile von Gym, als Lernplattform für Reinforcement Learning.

Ein Kritikpunkt an Gym ist die Beschreibung der Umgebungen, welche zu Verwirrungen führen kann. So ist bei dem Mountain-Car- Continuous Problem auf der Website die gleiche Beschreibung wie bei dem generellen Mountain-Car Problem.[[31]](#footnote-32) Auch in den einleiteten Kommentaren des Codes wird nur von einer “Adaptation of the MountainCar Environment“[[32]](#footnote-33) geschrieben. Es wird jedoch nicht erwähnt, was sich geändert hat. Somit ist nur durch die Untersuchung des Codes ersichtlich, dass die Belohnung nicht nach jeder Aktion -1 ist, sondern 0. Bei dem Erreichen des Zieles ändert sich die zurückgegebene Belohnung zudem zu 100. Eine weitere, nicht dokumentierte Veränderung des Mountain-Car-Continuous-Problems ist die Änderung der Aktionsmöglichkeiten von drei diskreten Werten zu einem kontinuierlichen Zahlenraum. Durch diese Änderung können Zwischenstufen zu den diskreten Aktionen angesprochen werden und das Mountain-Car exakter gesteuert werden.

In den einleitenden Kommentaren des Mountain-Car Problems wird für die Belohnung beschrieben: „Reward of 0 is awarded if the agent reached the flag (position = 0.5) on top of the mountain“.[[33]](#footnote-34) Bei der Ausgabe der Belohnung am Ziel, erhält man jedoch, wie an den anderen Positionen, eine Belohnung von -1.

Die Beschreibung des Frozen-Lake Problems auf der Website https://gym.openai.com /envs/ FrozenLake-v0/ beinhaltet die Formulierung „However,the ice is slippery, so you won’t always move in the direction you intend“.[[34]](#footnote-35) Diese Formulierung wird als Hinweis verwendet, dass der Agent sich bei einer Aktion mit einer Wahrscheinlichkeit von 2/3 nicht an die gewählte Position bewegt. Da sie jedoch so unscharf formuliert wurde, kann auch diese Erkenntnis nur durch die Untersuchung des Codes gewonnen werden. Dort wird in Zeile 150 ein Wert zur Auswahl von 1/3 gegeben. Ob dieser Wert zur Auswahl dient oder in der aufgerufenen Methode der Oberklasse anders verwendet wird ist zwar nicht zu bestimmen, führt man jedoch die gleiche Aktion wiederholt an der Startposition aus, ergeben sich für die drei Möglichkeiten fast gleichgroße Werte. Der Code der dafür durchgeführten Tests ist unter „<https://github.com/Fgrueter/Praxisprojekt>“ als „[übergangstest.py](https://github.com/Fgrueter/Praxisprojekt/blob/master/%C3%BCbergangstest.py)“ veröffentlicht.

Dagegen gibt es als positiven Punkt anzumerken, dass der Code der Umgebungen öffentlich zugänglich ist. Somit lassen sich die Umgebungen trotz des Kritikpunktes ganz nachvollziehen.

Eine weitere Stärke von Gym ist die Einsteigerfreundlichkeit. Die Installation von Gym und den Umgebungen ist mit der bereitgestellten Anleitung schnell und unkompliziert. Für einen Überblick über mögliche Reinforcment Algorithmen und Agenten werden unter <https://github.com/openai/gym/wiki/Learning-resources> hilfreiche Links angeboten.

Insgesamt lassen die Versuche mit Gym darauf schließen, dass es sich als Lernplattform für die TH Köln nutzen lässt.

Literaturverzeichnis

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang,and Wojciech Zaremba. OpenaiGym.arXiv preprint arXiv:1606.01540, 2016.

<https://towardsdatascience.com/the-other-type-of-machine-learning-97ab81306ce9> ,What is Reinforcment Learning“ [Hayes2019], aufgerufen am 12.07.2020

<https://spinningup.openai.com/en/latest/spinningup/rl_intro2.html#id20>, „Model-Free vs Model-Based RL“ [OpenAI2018], aufgerufen am 12.07.2020

<https://spinningup.openai.com/en/latest/spinningup/rl_intro2.html#id20>, „What to Learn in Model-Free RL” [OpenAI2018], aufgerufen am 12.07.2020

https://spinningup.openai.com/en/latest/spinningup/rl\_intro2.html#citations-below, “A Taxonomy of RL Algorithms”, Abbildung „A non-exhaustive,but useful taxonomy of algorithms in modern RL.“ [OpenAI2018], aufgerufen am 12.07.2020

<https://towardsdatascience.com/how-to-install-openai-gym-in-a-windows-environment-338969e24d30>, „How to Install OpenAI Gym in a Windows Environement“ [Hayes2019], aufgerufen am 11.07.2020

<https://towardsdatascience.com/getting-started-with-reinforcement-learning-and-open-ai-gym-c289aca874f>, „Getting Started with Reinforcement Learning and Open AI Gym“ [Hayes2019], aufgerufen am 11.07.2020

Anhang

Übersicht über die verwendeten Pythonprogramme, welche bei github im Repositorie „[Fgrueter](https://github.com/Fgrueter) / [Praxisprojekt](https://github.com/Fgrueter/Praxisprojekt)**“ veröffentlicht sind:**

## **übergangstest.py:**

## **Skript zur Überprüfung der Zustandsübergänge nach einem Schritt in der**

## **Frozen-Lake-Umgebung**

## **frozenLakeLeaderboard**.py:

Algorithmus des Leaderboards zur Lösung FrozenLake-Umgebung mit Sequenzen von zufälligen Aktionen.

## **frozenLakeLeaderboardFür8x8.py**:

Algorithmus des Leaderboards zur Lösung FrozenLake-8x8-Umgebung mit Sequenzen von zufälligen Aktionen.

FrozenLakeEigenerAlgorithmus.py:

Funktion frozenAI:

Algorithmus zur Lösung der Frozenlake-Umgebung und FrozenLake-8x8-Umgebung.

Veränderbare Parameter sind die Umgebung mit der entsprechenden Anzahl an Zeilen und Spalten und ein Array mit Einträgen der Löcher und ein Array mit den Ergebnissen vorheriger Durchläufe.

## **MountainCarQ-Learning.py:**

## **Funktion** QLearning:

## Anwendung eines tabellarischen Q-Learning-Algorithmus in der Mountain-Car-Umgebung.

Als Parameter einstellbar sind die Lernrate des Algorithmus, die Discountrate, der Epsilonwert und eine Min-Epsilonwert für die Epsilon-Greedy-Strategie und die Episodenanzahl.

**Funktion** QModel:

Funktion zur Überprüfung, wie gut die erlernte Q-Tabelle von der QLearning Funktion ohne weiteres Lernen in der Mountain-Car-Umgebung funktioniert.

Als Parameter sind die Anzahl der Episoden einzustellen.

## **MountainCarContinuous.py:**

## **Funktion** QLearning:

## Anwendung eines tabellarischen Q-Learning-Algorithmus in der Mountain-Car- Continuous-Umgebung.

Als Parameter einstellbar sind die Lernrate des Algorithmus, die Discountrate, der Epsilonwert und eine Min-Epsilonwert für die Epsilon-Greedy-Strategie und die Episodenanzahl.

**Funktion** QModel:

Funktion zur Überprüfung, wie gut die erlernte Q-Tabelle von der QLearning Funktion ohne weiteres Lernen in der Mountain-Car-Continuous-Umgebung funktioniert.

Als Parameter sind die Anzahl der Episoden einzustellen.

## **CartPoleQLearning.py:**

## **Funktion** QLearning:

## Anwendung eines tabellarischen Q-Learning-Algorithmus in der Cart-Pole-Umgebung.

Als Parameter einstellbar sind die Lernrate des Algorithmus, die Discountrate, der Epsilonwert und eine Min-Epsilonwert für die Epsilon-Greedy-Strategie und die Episodenanzahl.



1. Vgl. Greg Brockmann et al., 2016, S.1 Abstract [↑](#footnote-ref-2)
2. <https://github.com/openai/gym/blob/master/docs/environments.md#third-party-environments> , aufgerufen am 09.07.2020 [↑](#footnote-ref-3)
3. <https://github.com/openai/gym/blob/master/docs/creating-environments.md> ,aufgerufen am 09.07.2020 [↑](#footnote-ref-4)
4. Vgl. Greg Brockmann et al., 2016, S.2 Design Decisions Z.19ff. [↑](#footnote-ref-5)
5. Vgl. What is Reinforcment Learning, [Hayes2019] [↑](#footnote-ref-6)
6. Vgl. Model-Free vs Model-Based RL, [OpenAI2018] [↑](#footnote-ref-7)
7. Vgl. Model-Free vs Model-Based RL, [OpenAI2018] [↑](#footnote-ref-8)
8. Vgl. What to Learn in Model-Free RL, [OpenAI2018] [↑](#footnote-ref-9)
9. A Taxonomy of RL Algorithms, [OpenAI2018] [↑](#footnote-ref-10)
10. Vgl. What is Reinforcment Learning, [Hayes2019] [↑](#footnote-ref-11)
11. <https://gym.openai.com/docs/> , aufgerufen am 10.07.2020 [↑](#footnote-ref-12)
12. How to Install OpenAI Gym in a Windows Environement [Hayes2019] [↑](#footnote-ref-13)
13. <https://github.com/openai/gym/wiki/Leaderboard> , aufgerufen am 10.07. [↑](#footnote-ref-14)
14. Vgl. <https://gym.openai.com/envs/MountainCar-v0/> , aufgerufen am 11.07.2020 [↑](#footnote-ref-15)
15. Getting Started with Reinforcement Learning and Open AI Gym [Hayes2019] [↑](#footnote-ref-16)
16. Getting Started with Reinforcement Learning and Open AI Gym [Hayes2019] [↑](#footnote-ref-17)
17. Ergebnisse des durchgeführten Testes verschiedener Episoden [↑](#footnote-ref-18)
18. Ergebnisse des durchgeführten Testes verschiedener Lernraten [↑](#footnote-ref-19)
19. Ergebnisse des durchgeführten Testes verschiedener Discountfaktor [↑](#footnote-ref-20)
20. Ergebnisse des durchgeführten Testes verschiedener ε -Minimumwerte [↑](#footnote-ref-21)
21. Ergebnisse des durchgeführten Testes verschiedener ε -Werte bei 1500 Episoden [↑](#footnote-ref-22)
22. Ergebnisse des durchgeführten Testes verschiedener ε-Werte bei 10000 Episoden [↑](#footnote-ref-23)
23. Ergebnisse des durchgeführten Testes mit verschiedenen Werten für die Steigerung der Reduktion [↑](#footnote-ref-24)
24. Ausgabe der Belohnung von drei Durchläufen des Models [↑](#footnote-ref-25)
25. Vgl. <https://github.com/openai/gym/blob/master/gym/envs/classic_control/mountain_car.py>, Zeile 87, aufgerufen am 13.07.2020 [↑](#footnote-ref-26)
26. Vgl. <https://github.com/openai/gym/blob/master/gym/envs/classic_control/mountain_car.py>, Zeile 92, aufgerufen am 13.07.2020 [↑](#footnote-ref-27)
27. Ausgabe der Belohnung des Q-Learning-Algorithmus für Mountain Car Problems [↑](#footnote-ref-28)
28. Ausgabe der Belohnung des Q-Learning-Algorithmus für Mountain Car Problems nach der Änderung der Belohnung [↑](#footnote-ref-29)
29. Vgl. <https://gym.openai.com/envs/CartPole-v1/>, aufgerufen am 13.07.2020 [↑](#footnote-ref-30)
30. Vgl. <https://github.com/openai/gym/blob/master/gym/envs/classic_control/cartpole.py>, Z. 52-58 und Z.26-32, aufgerufen am 13.07.2020 [↑](#footnote-ref-31)
31. Vgl. <https://gym.openai.com/envs/MountainCar-v0/> und <https://gym.openai.com/envs/MountainCarContinuous-v0/> , aufgerufen am 12.07.2020 [↑](#footnote-ref-32)
32. <https://github.com/openai/gym/blob/master/gym/envs/classic_control/continuous_mountain_car.py>, aufgerufen am 12.07.2020 [↑](#footnote-ref-33)
33. <https://github.com/openai/gym/blob/master/gym/envs/classic_control/mountain_car.py>, aufgerufen am 12.07.2020 [↑](#footnote-ref-34)
34. <https://gym.openai.com/envs/FrozenLake-v0/>, aufgerufen am 12.07.2020 [↑](#footnote-ref-35)