Entwicklung eines RL-Agenten

Verfasserin: Elisabeth Barar

Kurs: WWI22DSB

Matrikelnummer: 2825002

Inhaltsverzeichnis

Abbildungsverzeichnis

1. Vorstellung des Projektes

Ziel dieses Projekts ist es, einen Reinforcement-Learning (RL) Agenten zu entwickeln, der in einer geeigneten Umgebung trainiert und evaluiert werden kann. Dabei sollen die in den Vorlesungen behandelten Konzepte und Methoden angewandt werden. In diesem Bericht wird zunächst die gewählte RL-Umgebung vorgestellt und begründet, warum sie ausgewählt wurde. Anschließend wird erläutert, welche RL-Konzepte zur Entwicklung des Agenten verwendet wurden und welche Ergebnisse erzielt werden konnten. Zum Schluss werden die wichtigsten Learnings und Herausforderungen des Projekts beschrieben**.**

2. RL-Umgebung

Als RL-Umgebung wurde Mountain Car aus der Classic-Control-Kategorie von OpenAI Gymnasium ausgewählt.

2.1 Beschreibung

In dieser Umgebung besteht das Ziel darin, ein Auto (Mountain Car), das sich in einer Senke zwischen zwei Hügeln befindet, so zu beschleunigen, dass es genug Schwung aufbaut, um die Flagge auf dem rechten Hügel zu erreichen. Dies soll in möglichst wenigen Schritten geschehen. Zusätzlich sollte die Endposition des Autos möglichst nahe an der Flagge bei Position 0.5 liegen [1].

Wie in allen RL-Umgebungen haben wir auch hier einen Aktionsraum, Observationsraum und eine Belohnungsfunktion. Der Aktionsraum ist diskret und hat die folgenden Aktionen: Aktion 0, beschleunige nach links; Aktion 1, beschleunige gar nicht; Aktion 2, beschleunige nach rechts. Der Observationsraum ist ein Array der Länge 2. An der Position 0 wird die Position des Autos entlang der x-Achse angegeben. Diese kann zwischen -1,2 und 0,6 sein. Die Flagge befindet sich dabei an der Position 0,5. An der Position 1 des Arrays wird die Beschleunigung des Autos angegeben, welche Werte zwischen -0,07 und 0,07 annehmen kann. Die Zustandsdynamik ergibt sich aus den folgenden Gleichungen:
Position(t+1) = Position(t) + Geschwindigkeit(t+1)
*Geschwindigkeit(t+1) = Geschwindigkeit(t) + (Aktion - 1) * Kraft - cos(3 * Position(t)) * Schwerkraft*
Die Startposition des Autos wird zufällig gewählt, während die Anfangsgeschwindigkeit immer 0 ist. Die Belohnungsfunktion ist wie folgt: Für jeden Schritt, den das Auto bis zum Ziel benötigt, erhält es eine Belohnung von -1. Ziel ist es daher, das Ziel möglichst schnell zu erreichen [1].

2.2 Begründung

Die Wahl der Mountain-Car-Umgebung wurde aus mehreren Gründen getroffen. Als erstes ist es eine klassische Umgebung, an der die grundlegenden Konzepte des Reinforcement Learnings gut geübt werden können, ohne mit einer zu hohen Komplexität konfrontiert zu werden. Zusätzlich wurden ähnlichen Umgebungen wie dem Pendulum in der Vorlesung vorgestellt. Außerdem ist Mountain Car durch die geringere Komplexität relativ ressourcenschonend.

Daher, das Training des RL-Agenten dauert nicht zu lange und lässt sich gut auf einem normalen Laptop durchführen.

3. Entwicklung des RL-Agenten

Für die Entwicklung des RL-Agenten wurden sich entschieden zwei unterschiedliche Methoden zu verwenden. Als erstes wurde der Actor-Critic Algorithmus verwendet und als zweites das Deep Q-Learning. Im Folgenden wird auf die Umsetzung der beiden Methoden eingegangen. Der vollständige Code kann auf Github angeschaut werden unter dem Link https://github.com/Ellibab/Abgabe_Aktuelle_Data_Science_Entwicklungen/tree/main.

3.1 Actor-Critic Algorithmus

Beim Actor-Critic Algorithmus wird der Agent sowohl mit einer Policy als auch mit einer Value Funktion trainiert. Der Actor (Policy) lernt, wie man die Entscheidung trifft daher welche Aktion gewählt werden muss. Als Eingabe bekommt er einen bestimmten Zustand in der Umgebung und gibt eine Verteilungsfunktion aus, welche die Aktion sagt, den der Agent durchführen soll. Die Strategie des Actors wird geupdated anhand des Feedbacks der Umgebung und der Evaluation vom Critic. Der Critic (Value Funktion) schaut wie gut die Entscheidung des Actors war. Er soll daher für ein Zustands-Aktion Paar oder nur einen Zustand die zukünftige Belohnung vorhersagen. Als Exploration wird eine Softmax-Aktivierungsfunktion verwendet [2, 3].

Sowohl Actor als auch Critic wurden als separate neuronale Netze mit jeweils zwei linearen Schichten und ReLU-Aktivierungsfunktionen implementiert. Der Actor verwendet zusätzlich eine Softmax-Funktion im letzten Layer zur Exploration. Für ein schnelleres Lernen werden die Erfahrungen einer gesamten Episode gespeichert, die Gewinne berechnet und normalisiert. Daraus werden die Vorteile abgeleitet, die angeben, wie viel besser eine Aktion war als der erwartete Wert. Basierend auf diesen Vorteilen aktualisiert der Actor seine Policy, und der Critic verbessert seine Vorhersage der Zustandswerte. Die Belohnungsfunktion wurde manuell angepasst, um den Agenten für das Erreichen einer Position von 0,5 oder mehr stark zu belohnen. Ein frühes Beenden ohne Zielerreichung wurde bestraft, während eine kontinuierliche positive Belohnung gegeben wurde, je näher das Auto der Zielposition kam. Der Agent wurde insgesamt über 500 Episodentrainiert, um die Trainingszeit kurz zu halten.

3.2 Deep Q-Learning

Beim Deep Q-Learning verwendet man ein neuronales Netzwerk, um die Q-Werte zu approximieren für jedes State Action Paar. Die Erfahrungen, die der Agent sammelt, werden im Memory Buffer gespeichert. Dazu zählt der aktuelle Zustand, die durchgeführte Aktion, die Belohnung und den nächsten Zustand. Im Gegensatz zum Actor-Critic Algorithmus wird hier nur eine Value Funktion benutzt [4].

Das implementierte Target neuronales Netzwerk besteht aus drei linearen Schichten. Zwischen diesen wird die ReLU-Aktivierungsfunktion verwendet, um Nichtlinearität mit reinzubringen.

Als Input bekommt die Layers, die Space-Dimension und zum Schluss gibt der letzte Layer die Q-Werte für alle möglichen Aktionen zurück. Zum Updaten der Werte wird die Bellman Gleichung verwendet. Um eine Balance zwischen Exploration und Exploitation zu finden, daher zufällig eine Aktion ausprobieren oder die mit dem höchsten Q-Wert zu nehmen, wurde die Epsilon-Greedy Strategie implementiert. Den kleinsten Wert welchen Epsilon annehmen darf ist 0,01 und den größten ist 1. Der Epsilon Decay ist dabei 0,995. Zusätzlich wurde die Belohnungsfunktion angepasst, dass wenn Mountain Car das Ziel erreicht bzw. seine Endposition größer oder gleich 0,5 ist, er eine Belohnung von 100 bekommt. Insgesamt wurde der Agent über 1000 Epochen trainiert, mit einer Lernrate von 0,001 und als Optimizer wurde der Adam Loss verwendet. Zur Orientierung bei der Programmierung wurden folgende Quellen verwendet [5, 6].

4. Bewertung des RL-Agenten

Nachdem die Implementierung der RL-Agenten erklärt wurde, sollen diese im Folgenden evaluiert werden.

4.1 Actor-Critic Algorithmus

Beim Actor-Critic Algorithmus kann man in Abb. 1 erkennen, dass sowohl der Loss vom Critic als auch vom Actor sinkt und sich immer mehr der null annähert. Dabei ist das Sinken des Losses vor allem beim Critic zu beobachten.

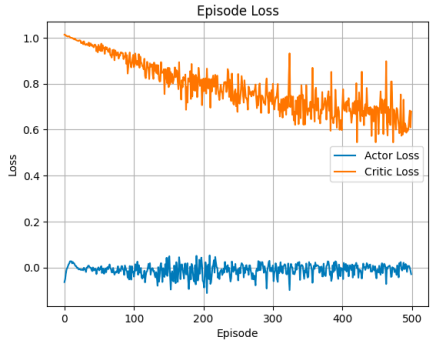

Abbildung 1 Verlauf des Losses von Actor und Critic

Die Belohnung dagegen steigt mit der Zeit deutlich an wie man in Abb. 2 sehen kann. Daher der Agent lernt mit der Zeit sich dem Ziel anzunähern.

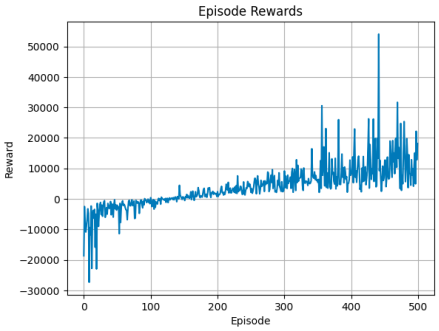

Abbildung 2 Verlauf der Belohnung

Jedoch konnte beobachtet werden, dass sogar der beste Agent nicht das Ziel (die Flagge auf dem Hügel) erreicht hat. Daher kann abschließend gesagt werden, dass der Actor-Critic Algorithmus nicht das beste RL-Konzept für die Umgebung Mountain Car ist. Ein Grund dafür könnte sein, dass trotz des manuellen Anpassens der Belohnungsfunktion immer noch zu spärlich ist, um ein gutes Training zu ermöglichen. Ein anderer Grund könnte sein, dass es im lokalen Optimum stecken bleibt. Daher der Agent lernt dann beispielsweise, einfach keine Kraft anzuwenden und am unteren Ende des Tals zu verharren, um die negativen Belohnungen zu minimieren, statt den aufwendigen Weg zum Ziel zu finden. Auch ist die Actor-Critic Methode besser geeignet für kontinuierliche Aktionen, jedoch wurde in diesem Projekt Mountain Car mit einem diskreten Aktionsraum verwendet.

4.2 Deep Q-Learning

Abb. 3 zeigt, dass die Anzahl der Schritte zur Erreichung des Ziels (der Flagge) nach etwa 50 Episoden sehr schnell von rund 50.000 auf unter 200 Schritte sank und über den Rest des Trainings konstant niedrig blieb. Der Verlauf der Belohnung, dargestellt in Abbildung 4, spiegelt den Graphen der Schrittanzahl an der x-Achse, da die Höhe der Belohnung direkt von der Anzahl der benötigten Schritte abhängt.

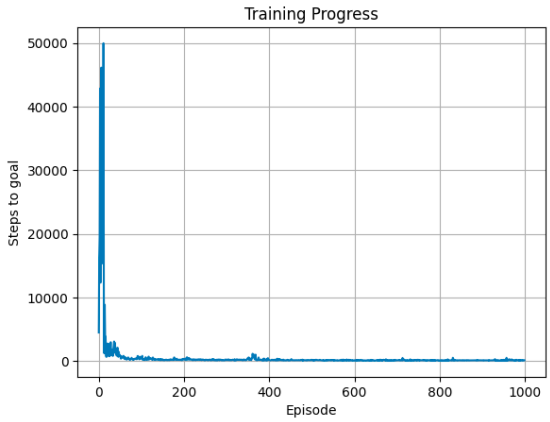

Abbildung 3 Anzahl der Schritte zur Erreichung des Ziels pro Episode

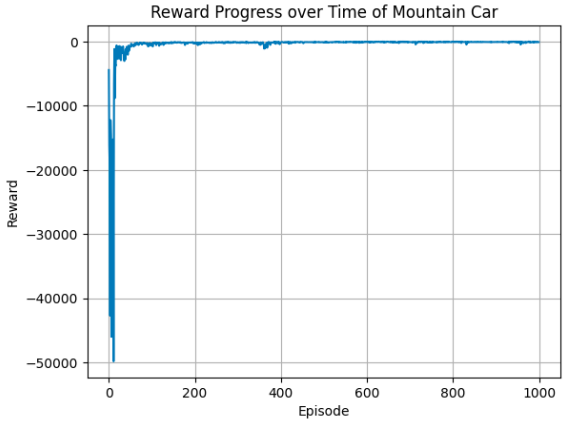

Abbildung 4 Verlauf der Belohnung

In Abb. 5 sieht man, dass der durchschnittliche Loss pro Episode gesunken ist von über 25 auf ca. 4 am niedrigsten Punkt, den er bei knapp unter 1000 Episoden erreichte. Man sieht, dass der Loss bis ca. Episode 500 sank bis er wieder bis ca. Episode 750 anstieg, um dann wieder zu fallen.

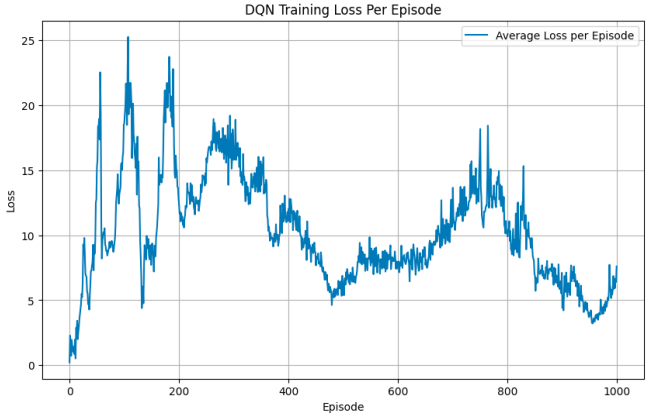

Abbildung 5 Durchschnittlicher Loss pro Episode

In seinem besten Durchlauf erzielte der Agent folgende Werte: eine Gesamtbelohnung von -148, benötigte 148 Schritte und erreichte eine Endposition von 0,5135. Die Abweichung von der genauen Zielposition betrug somit nur 2,2%. Diese Ergebnisse zeigen, dass der Agent sehr gut performte, was Deep Q-Learning für diese Umgebung als sehr geeignet erscheinen lässt. Das Training war zudem mit etwa 10 Minuten kurz.

Als Erweiterung könnte das Double Q-Learning implementiert werden. Jedoch wurde sich für diese Umgebung dagegen entschieden. Der Grund dafür ist, dass sich dies nur bei komplexen Umgebungen lohnt, um deutliche Verbesserungen zum normalen Deep Q-Learning festzustellen.

5. Fazit

Obwohl die Mountain Car-Umgebung auf den ersten Blick einfach erscheint, ist das Training doch komplexer als gedacht. Außerdem wurde erkannt, dass nicht jeder RL-Ansatz für jede Aufgabe geeignet ist. Dies kann vor allem an den unterschiedlichen Ergebnissen von Actor-Critic und DQN gesehen werden. Auch war ein Learning und gleichzeitig eine Herausforderung die Anpassung der Belohnungsfunktion. Diese musste für beide Ansätze angepasst werden, um bessere Ergebnisse zu erzielen beziehungsweise wie beim Actor-Critic es wenigstens zu versuchen.

6. Literaturverzeichnis

[1] Open AI Gymnasium Documentation Mountain Car, URL: https://gymnasium.farama.org/environments/classic_control/mountain_car/

[2] Geeks for Geeks. Actor-Critic Algorithm in Reinforcement Learning. 05.06.2025. URL: https://www.geeksforgeeks.org/machine-learning/actor-critic-algorithm-in-reinforcement-learning/

[3] Dhanoop Karunakaran. The Actor-Critic Reinforcement Learning algorithm. 30.09.2025. URL: https://medium.com/intro-to-artificial-intelligence/the-actor-critic-reinforcement-learning-algorithm-c8095a655c14

[4] Samina Amin. *Deep Q-Learning (DQN)*. 14.09.2024. URL: https://medium.com/@samina.amin/deep-q-learning-dqn-71c109586bae

[5] *DQN for Mountain Car RL Training*. 23.09.2025. URL: https://medium.com/@310274movie/dqn-for-mountain-car-rl-training-3b169a264534

[6] Ha Nguyen. *Playing Mountain Car with Deep Q-Learning*. 13.03.2021. URL: https://ha-nguyen-39691.medium.com/playing-mountain-car-with-deep-q-learning-9bdce3715159