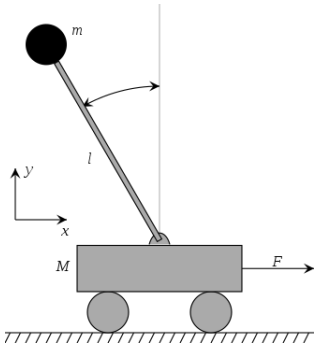
Reinforcement Learning with Python
Prof. Dr.-Ing. Nicolas Stein

Projektarbeit: Cartpole



Wir betrachten das Cartpole Regelungsproblem, auch bekannt als inverses Pendelproblem. Nach Wikipedia[1]ist ein inverses Pendel ein Pendel mit dem Schwerpunkt oberhalb der Achse. Das Pendel befindet sich in seinem höchsten Punkt in einer instabilen Ruhelage. Das inverse Pendel ist eine der Standardaufgaben der Regelungstechnik für die Stabilisierung einer instabilen Regelstrecke.
Im Alltag begegnet uns das inverse Pendelproblem beispielsweise beim Einrad- oder Segwayfahren.
Bei letzterem übernimmt ein integrierter Elektromotor die Regelungsaufgabe zum Ausbalancieren des Fahrers.

Zum Ausbalancieren des Pendels lässt sich der dazugehörige Wagen, auf dem das Pendel nach der Abbildung oben befestigt ist, hin und her bewegen. Der Aktionsraum ist durch die Aktionen Kraft aufbringen nach links oder rechts auf den Wagen bestimmt. Das Pendel startet zunächst aufrecht. Das Ziel ist es das Pendel auch aufrecht zu halten. Eine Belohnung von +1 wird für jeden Zeitschritt generiert, in der sich das Pendel in einem gewissen Spielraum des Winkels befindet. Die Episode endet, wenn das Pendel mehr als 12° von seiner Ausgangsposition ausgelenkt wird oder wenn sich der Wagen mehr als 2.4 Einheiten vom Startpunkt wegbewegt hat. Als Zustandsbeschreibung wird die Position und Geschwindigkeit des Wagens sowie der Winkel und die Winkelgeschwindigkeit des Pendels verwendet.

Ein Modell der Umgebung ist als gym Umgebung gegeben und lässt sich wie folgt instanziieren[2]

```python
import gym

env = gym.make('CartPole-v1')
```

[1] https://de.wikipedia.org/wiki/Inverses_Pendel

[2] Es wird explizit die v1 (Version 1) der Umgebung verwendet. Für viele gym-Umgebungen gibt es verschiedene Versionen, die meist die gleiche Logik beinhalten, jedoch unterschiedliche Parameterwerte aufweisen.

1. Machen Sie sich mit der Umgebung vertraut. Instanziieren Sie sie und betrachten Sie die Dimensionen des Zustands- und Aktionsraumes.

2. Wie viele Zeitschritte sind maximal pro Episode erlaubt. Wann ist das Problem als gelöst anzusehen?

3. Passen Sie ihre Implementierung des DQN Agenten aus der Programmierübung-3 auf das CartPole Problem an und lösen Sie dieses.

   Hinweis: Wenn Sie sich erinnern, hatten wir für das mountainCar Problem eine angepasste Belohnungsfunktionsimplementierung.

Der DQN-Ansatz weist einige Nachteile auf. Es gibt einige Verbesserungsvorschläge des Algorithmuses, die diese Nachteile ausgleichen und an verschiedenen Stellen den DQN Ansatz verbessern. Dies umfasst unter anderem eine Anpassung der Update-Regel, ein verbessertes Experience Replay Konzept und eine Veränderung der Architektur des neuronalen Netzes. Sie lassen sich jeweils gesondert betrachten, können jedoch auch kombiniert angewendet werden. Sie werden sich im Folgenden diese Erweiterungen konzeptuell erarbeiten, implementieren und mit dem reinen (vanilla) DQN Ansatz vergleichen.

Gruppe 1: Double Deep Q Networks (DDQN)

Der DDQN (Double Deep Q Network) Ansatz erweitert den DQN Ansatz, um das sogenannte „maximation bias“ Problem zu umgehen.

1. Recherchieren Sie, welches Problem hier betrachtet wird. Durchdenken und beschreiben Sie es.

2. Belesen Sie sich bezüglich des Konzeptes des double learnings[3], das das „maximation bias“ Problem löst und dessen Verallgemeinerung auf den DQN-Ansatz[4]. In ihrer Präsentation sollten Sie sowohl die Problematik als auch den zugehörigen Lösungsansatz für double learning sowie Double DQN ihren Kommilitonen verständlich erklären können.

3. Implementieren Sie aufbauend auf ihrem DQN Agenten ein DDQN Agent, bei dem Sie das Konzept des double Q-learnings umsetzen.

4. Vergleichen Sie die Lernrate des DDQN und des DQN-Agenten anhand der CartPole-Umgebung. Visualisieren Sie sich eine Lernkurve, indem Sie die kumulative Belohnung über die Anzahl der Episoden plotten.

5. Diskutieren Sie ihre Ergebnisse. Hat die Erweiterung des DDQN-Ansatzes eine Verbesserung der Lernkurve bewirkt?

6. Nach 400-600 Episoden gibt es ein plötzlich rapides Abfallen der kumulativen Belohnungen einer Episode. Überlegen und recherchieren Sie die möglichen Gründe dafür.

[3]Sutton und Barto widmen ein ganzes Unterkapitel in ihrem Buch dem Thema „maximation bias and double learning“. Alternative Quellen finden sich auch auf `https://towardsdatascience.com/` oder `https://medium.com/`

[4]Original Artikel von Hasselt et al. 2015: `https://arxiv.org/pdf/1509.06461.pdf`

Gruppe 2: Prioritized Experience Replay

Prioritized Experience Replay (PER) versucht das buffern von Erfahrungen des Agenten zu optimieren.

1. Recherchieren Sie, welches Problem PER versucht zu lösen.[5]

2. Belesen Sie sich um das Konzept des PER[6]. In ihrer Präsentation sollten Sie sowohl die Problematik als auch den zugehörigen Lösungsansatz ihren Kommilitonen verständlich erklären können.

3. Implementieren Sie die proportionale Variante von PER aufbauend auf ihrem DQN-Agenten. Führen Sie dazu eine neue Klasse Memory ein, die das Priorisieren managet und in der die Datenstruktur SumTree verwendet wird. Nutzen Sie hierfür das bereit gestellte Framework.

   (a) Machen Sie sich mit dem framework vertraut.

   (b) Implementieren Sie die Funktionen der Datenstruktur SumTree. Damit Sie nicht immer den DQN Agenten bemühen müssen, um ihre Implementierung zu testen, gibt es unit tests (`test_memory.py`), die Sie bspw. mit pytest verwenden können. Mit diesen können Sie die ersten Schritte der Implementierung ohne viel Zeitaufwand testen.

   (c) Implementieren Sie daraufhin die TODOs in der Klasse Memory (`memory.py`), um Prioritized Experience Replay zu implementieren.

4. Vergleichen Sie die Lernrate des DQN-Agenten mit dem DQN-Agent mit PER anhand der CartPole-Umgebung. Visualisieren Sie sich eine Lernkurve, indem Sie die jeweilige Belohnung der Episode über die Episodenanzahl plotten.

5. Diskutieren Sie ihre Ergebnisse. Hat die Erweiterung des DQN-Ansatzes eine Verbesserung der Lernkurve bewirkt?

[5] Es gibt einige schöne und verständliche Artikel im Internet, bspw. auf Platformen wie `https://towardsdatascience.com/` oder `https://medium.com/`. Zusätzlich ist ein zusammenfassender Überblick auf moodle gegeben.

[6] Originalartikel von Schaul et al. 2016: `https://arxiv.org/abs/1511.05952`, kurze Zusammenfassung bspw. hier: `https://github.com/bentrevett/paper-notes/blob/master/notes/prioritized-experience-replay.md`

Gruppe 3: Dueling DQN

Dueling DQN versucht die Architektur des tiefen neuronale Netz an sich zu verbessern.

1. Recherchieren Sie, was den Ansatz von Dueling DQN vom vanilla DQN Ansatz unterscheidet[7].

   In ihrer Präsentation sollten Sie die Ideen des Dueling DQN ihren Kommilitonen verständlich erklären können.

2. Implementieren Sie einen Dueling DQN-Agenten aufbauend auf ihrem DQN-Agenten. Sollten Sie noch nicht Erfahrung mit keras und tensorflow haben, empfiehlt sich die Dokumentation[8]. Es ist Ihnen dabei freigestellt das gegebene neuronale Netz zu erweitern oder eine eigene Subklasse einzuführen.

3. Vergleichen Sie die Lernrate des Dueling DQN-Agenten mit dem klassischen DQN-Agent anhand der CartPole-Umgebung. Visualisieren Sie sich eine Lernkurve, indem Sie die kumulative Belohnung über die Anzahl der Episoden plotten.

4. Variieren Sie die Parameter, um eine bestmögliche Lernkurve mit Ihrem Dueling DQN zu erreichen.

5. Diskutieren Sie ihre Ergebnisse. Hat die Erweiterung des DQN-Ansatzes eine Verbesserung der Lernkurve bewirkt?

6. **Bonus:** Zusätzlich zur Anpassung des neuronalen Netzes, können Sie die Erweiterung des Double DQN Ansatzes implementieren. Bringt dies eine weitere Verbesserung?

[7] Originalartikel: Wang et al. 2015 `https://arxiv.org/abs/1511.06581` oder Platformen wie `https://towardsdatascience.com/` oder `https://medium.com/`.

[8] `https://www.tensorflow.org/api_docs/python/tf/keras`