KIP 23/24 - Praktikum 2: Reinforcement Learning

Rahmenbedingungen

Programmiersprache: Python.
Persönliche Empfehlung: PyCharm als IDE verwenden, andere Optionen wie z.B. Jupyter Notebooks aber auch legitim. Lösung, wenn z.B. kein Laptop vorhanden ist, welcher zum Praktikumstermin mitgebracht werden kann: Jupyter Notebooks auf Laborrechnern im 11. Stock oder Google Collab → Jupyter Notebooks in Google Cloud.
Implementierung: In den Aufgaben werden einige "Grundpfeiler" für die Implementierungen vorgegeben, davon ab werden euch jedoch sehr viele Freiheiten gelassen. Grundprinzipien aus Programmieren 1 und 2 und Software Engineering sollten natürlich auch in KIP Anwendung finden, so dass die Implementierung gut "lesbar", wartbar, erweiterbar etc. sein sollte.
LLM Policy: Die Verwendung von z.B. ChatGPT ist grundsätzlich erlaubt. Falls ihr dieses verwenden solltet, dokumentiert grob eure Prompts, wie gut das geklappt hat, wo ihr Verbesserungen vorgenommen habt und ob ihr meint, dadurch einen Zeitgewinn erhalten zu haben.
Persönliche Einschätzung: Der Zeitgewinn wird eher gering ausfallen, da der generierte Code auch verstanden und häufig verbessert werden muss. Dadurch, dass eben diese Konzepte auch verstanden und vermittelt werden müssen für die Abgabe (s.u.), bietet es sich für den Lernprozess ggf. eher an, den Code von Grund auf selbst zu implementieren.
Abnahme:
Grundsätzlich gilt: Beide Teammitglieder müssen sowohl den Code als auch die relevanten Konzepte vollständig erklären können. **Die Aufgaben sollten bis zu dem Praktikumstermin fertig sein und nicht erst im Praktikumstermin bearbeitet werden**.
Der generelle Ablauf ist wie folgt: Idealerweise redet ihr den größten Teil der Zeit über eure Aufgabenlösungen, führt grob durch den Code und stellt die dahinterstehenden Konzepte vor und beantwortet die Teilfragen der einzelnen Aufgaben, sodass nur noch wenige Rückfragen bleiben. Zwischendurch und danach werde ich ggf. einige Fragen zur Implementierung und den dazugehörigen Konzepten der Vorlesung stellen. Idealerweise dauert dieser Prozess etwa 20 Minuten - bereitet euch also vor!

Aufgabe

Für alle folgenden Aufgaben gilt:
Die Verwendung von fertigen Implementierungen der kompletten Algorithmen, importiert aus Packages, ist nicht erlaubt. Die Implementierung soll generell mit möglichst wenigen Imports auskommen (also eine weitgehend eigenständige Implementierung der Algorithmen darstellen).

Environments

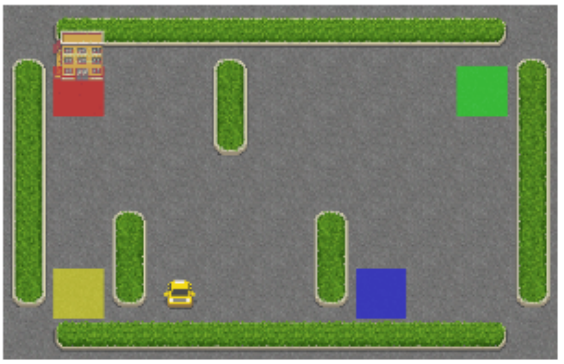
In dieser Aufgabe sollen Reinforcement Learning (RL) Algorithmen implementiert und anhand gegebener *Environments* überprüft werden. Diese Environments werden von **Farama Gymnasium** (https://gymnasium.farama.org/) zur Verfügung gestellt, müssen

also nicht von euch implementiert werden. Farama Gym ist ein Fork des inzwischen eingestellten ***OpenAI Gym***, welches eine Reihe von RL Environments (Atari-Videospiele, (Simulierte) Physikalische Simulationen, (Simulierte) Laufen-lernende Roboter,...) für u.a. das Benchmarking von RL Algorithmen zur Verfügung stellt. Die notwendigen Komponenten können mit `pip install gymnasium` in eure Entwicklungsumgebung installiert werden. Im Rahmen dieser Aufgabe soll es um die beiden Environments **Frozen Lake** und **Taxi** gehen.

Frozen Lake

https://gymnasium.farama.org/environments/toy_text/frozen_lake/

Taxi



https://gymnasium.farama.org/environments/toy_text/taxi/

Frozen Lake entspricht von der Grundidee her der in der VL behandelten Grid World. Taxi stellt eine leicht andere Umgebung dar. Macht euch mit dem jeweils zu lösenden Problem und der Gymnasium-Bibliothek vertraut, indem ihr die oben verlinkten Dokumentationen der beiden Environments studiert, um ihre Schnittstellen in RL-Algorithmen anwenden zu können.

Model-based Learning

Die Zustände, Übergangswahrscheinlichkeiten und Rewards sind zunächst unbekannt. Erstellt ein (angenähertes) ***Modell*** der beiden Environments durch eine ausreichend große Anzahl an Zufalls*episoden,* d.h. Durchläufe mit zufälligen Aktionen. Wo liegen Gemeinsamkeiten, wo Unterschiede bei den beiden Environments? Löst den dabei erstellten *Markov Decision Process*, indem ihr die folgenden Lösungsansätze implementiert:

- Value Iteration
- Policy Iteration

Wie funktionieren diese Lösungsansätze? Was sind Gemeinsamkeiten und wo liegen Unterschiede? Untersucht verschiedene γ-Werte (Discounting-Rate), inwiefern ändern diese das Ergebnis?
Visualisiert für Frozen Lake die resultierende Gesamtpolicy (vgl. Grid World in der Vorlesung) und für Taxi die 4*3 resultierenden Optimalrouten von allen 4 möglichen Locations zu den verbleibenden 3 Destinations.

(Fortsetzung nächste Seite)

Model-free Learning

Implementiert zur Lösung von Frozen Lake und Taxi außerdem

- Q-Learning

Wichtig hier: Es handelt sich um ein *lernendes Verfahren*! Konkret soll dies hier bedeuten, dass eure Implementierung die States erst findet, während das Verfahren durchläuft. Sie sind nicht schon vorher bekannt und gegeben! Verwendet die ϵ-greedy Strategie zur Exploration. Die Explorationsrate soll zu Beginn hoch sein und während des Lernens abnehmen. Überlegt euch hierfür eine geeignete ϵ-Funktion.
Wie funktioniert Q-Learning? Was sind Gemeinsamkeiten und wo liegen Unterschiede zu Value Iteration und Policy Iteration?
Visualisiert für Frozen Lake die resultierende Gesamtpolicy (vgl. Grid World in der Vorlesung) und für Taxi die 4*3 resultierenden Gesamtrouten von allen 4 möglichen Locations zu den verbleibenden 3 Destinations.

Weitere Environments und ihre Eigenschaften (Theorieaufgabe)

Gymnasium stellt eine Reihe weiterer Environments zur Verfügung, wie z.B.:

- Cart Pole https://gymnasium.farama.org/environments/classic_control/cart_pole/
- Car Racing https://gymnasium.farama.org/environments/box2d/car_racing/

Worin unterscheiden sich diese beiden jeweils grundsätzlich von Frozen Lake und Taxi? Gemeint sind nicht die Anwendungsfälle wie z.B. “Autofahren vs. Suche in einer Grid World”, sondern technische Eigenschaften der Umgebungen.
Warum sind sie nicht ohne Weiteres mit den oben implementierten Varianten von Value Iteration, Policy Iteration oder Q-Learning lösbar? Welche Variante wurde in der VL vorgestellt, welche zumindest einen Teil des Problems (welchen Teil?) löst?
Wie kann man Cart Pole und/oder Car Racing leicht ummodellieren, um sie mit dem oben implementierten Q-Learning (annähernd) lösbar zu machen? (Also wie kann die Problemdefinition leicht geändert werden zur Lösung, nicht das Verfahren)