Ludwig-Maximilians-Universität München
Institut für Informatik
Lehrstuhl für Mobile und Verteilte Systeme
Prof. Dr. Claudia Linnhoff-Popien

mobile and distributed systems group

Praktikum Autonome Systeme
Sommersemester 2022
Übungsblatt 2 – (Tabular) Reinforcement Learning

Ziel der heutigen Praxisveranstaltung sind Implementierung und Evaluation von SARSA und Q-Learning, womit ein Agent ein einfaches Navigationsproblem lösen soll.

Aufgabe 1: SARSA

Laden Sie für dieses Übungsblatt das ZIP-Archiv `asp_ue2.zip` runter. In diesem Archiv finden Sie die aktualisierten Quellcode-Dateien aus dem Übungsblatt 1.

Implementieren Sie einen SARSA Agenten mit ϵ −greedy Exploration. Orientieren Sie sich am Gerüst der Klasse `SARSALearner`:

1. Implementieren Sie die Methode `update` mit den Parametern `state`, `action`, `reward`, `next_state` in der Sie den Eintrag `self.q_values[state][action]` nach der SARSA-Lernregel aktualisieren (**Hinweise:** `self.q_values` ist diesmal ein Dictionary und kein Array! Sie erhalten das Q-Array zu `state`, wenn Sie `self.q_table(state)` aufrufen und a_{t+1}, indem Sie die Methode `self.policy(next_state)` aufrufen):

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha(r_t + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t))$$

2. Implementieren ein ϵ −Decay, indem Sie in der Methode `update` das Attribut `self.epsilon`, um einen konstanten Wert verringern. Achten Sie dabei, dass in jedem Fall `self.epsilon` > 0 gilt.

Lassen Sie den SARSA Agenten über mehrere *Rooms* Episoden (> 100) für verschiedene ϵ −Decays und Lernraten α laufen. Wie sehen die Lernkurven im Plot aus?

Wie verhält sich SARSA, wenn die ϵ −greedy Exploration in der Methode `policy` durch Boltzmann Exploration (mit $\tau \in \{0.1, 1, 10\}$) ersetzt wird? Verwenden Sie dazu die Implementierungen aus `multi_armed_bandits.py`.

Aufgabe 2: Q-Learning

Implementieren Sie einen Q-Learner mit ϵ −greedy Exploration. Orientieren Sie sich am Gerüst der Klasse `QLearner`:

1. Implementieren Sie die Methode `update` mit den Parametern `state`, `action`, `reward`, `next_state` in der Sie den Eintrag `self.q_values[state][action]` nach der Q-Lernregel aktualisieren (**Hinweis:** `self.q_values` ist diesmal ein Dictionary und kein Array. Sie erhalten das Q-Array zum `state`, wenn Sie `self.q(state)` aufrufen.):

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha(r_t + \gamma \max_{a_{t+1} \in \mathcal{A}} Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t))$$

2. Implementieren ein ϵ −Decay analog zu Aufgabe 1.2.

Lassen Sie den Q-Learner über mehrere *Rooms* Episoden (> 100) für verschiedene ϵ −Decays und Lernraten α laufen. Wie sehen die Lernkurven im Plot aus?

Wie verhält sich Q-Learning, wenn die ϵ −greedy Exploration in der Methode `policy` durch Boltzmann Exploration (mit $\tau \in \{0.1, 1, 10\}$) ersetzt wird? Verwenden Sie dazu die Implementierungen aus `multi_armed_bandits.py`.