Technische Hochschule Ostwestfalen-Lippe
University of Applied Sciences and Arts

Wintersemester 2022/2023

Vergleich verschiedener Reinforcement Learning Algorithmen

Modularbeit

Vorgelegt im Kontext des Moduls „Anwendung des maschinellen Lernens“

am Fachbereich Technische Informatik und Elektrotechnik
im Studiengang Data Science

Veranstalter:	Prof. Dr. Burkhard Wrenger
Vorgelegt von:	Bjarne Seen
	Liebigstraße 130
	32657 Lemgo
	bjarne.seen@stud.th-owl.de
Matr. Nr.:	15467085
Vorgelegt von:	Joshua Henjes
	Hanseweg 11
	32657 Lemgo
	joshua.henjes@stud.th-owl.de
Matr. Nr.:	15467024
Abgabetermin:	02.03.2023

1. März 2023

Inhaltsverzeichnis

Abkürzungen

MDP	Markov Decision Process
TD	Temporal Difference

1 Einleitung

Reinforcement Learning ist neben Supervised und Unsupervised Learning eins der elementaren Felder des maschinellen Lernens. Im Gegensatz zu den anderen Feldern benötigt Reinforcement Learning keine Trainingsdaten, denn der Algorithmus lernt durch wiederholtes Interagieren mit einer dynamischen Umgebung eine Strategie, um eine Belohnungsmetrik zu maximieren. Es wird daher auch als bestärktes lernen oder verstärktes lernen bezeichnet.

Bekannt wurde das Reinforcement Learning vor allem durch das Meistern von bekannten Brett- und Computerspielen, so ist Googles „AlphaGo“ in der Lage, die besten Go Spieler der Welt zu schlagen. Trotz dieser beeindruckenden Erfolge findet RL in der Industrie bisher nur geringe Anwendung.

Immer kürzer werdende Produktzyklen und steigende Produktvielfalt stellen für die heutigen Produktionsprozesse eine große Herausforderung dar. Zukünftige Produktionen müssen immer anpassungsfähiger werden. Zeitgleich soll der Personalaufwand aufgrund des anhaltenden Fachkräftemangels möglichst gering ausfallen. Maschinelles Lernen, insbesondere das Reinforcement Learning, kann bei der Bewältigung dieser Herausforderungen eine relevante Rolle übernehmen.

Auch bei der Bekämpfung des Klimawandels kann Reinforcement Learning unterstützen. Um unsere Klimaziele zu erreichen, ohne unseren Lebensstandard signifikant zu senken, ist eine Optimierung des Ressourcenbedarfs nötig. Mit ausreichender Trainingszeit sind RL-Algorithmen sehr gut in der Optimierung von Prozessen und somit auch in dessen Ressourcenverbrauches. Google, als einer der Vorreiter im Gebiet des maschinellen Lerners, konnte durch ML-Algorithmen den Energieverbrauch der Kühlung ihrer Rechenzentren um bis zu 40 Prozent reduzieren.

Mittlerweile existiert eine Vielzahl an unterschiedlichen Reinforcement Learning Algorithmen. Während die mathematischen und strukturellen Unterschiede meist gut dokumentiert und einsehbar sind, ist ein direkter Vergleich der Leistungsfähigkeit der Algorithmen in verschiedenen Umgebungen nur schwer zu finden. Aus diesem Grund beschäftigt sich diese Ausarbeitung mit dem Vergleich von beliebten RL-Algorithmen anhand von Umgebungen mit geringer Komplexität.

Während die Zeit, welche ein RL-Algorithmus zum Lernen benötigt, bei der Anwendung auf Brett- und Computerspielen eher eine untergeordnete Rolle spielt, ist sie in der Anwendung in industriellen Umgebungen deutlich relevanter. Zum einen verlangsamen hohe Trainingszeiten den Entwicklungsprozess deutlich, was wiederum zu höheren Lohn- und Entwicklungskosten führt. Zum anderen ist es in vielen Anwendungsfällen nötigt, dass die Umgebung während des Trainingsprozesses dem Algorithmus zur Verfügung steht. Im Fall von Produktionsanlagen ist Trainingszeit somit sehr kostspielig. Aus diesem Grund wird neben der Leistungsfähigkeit auch die Lerngeschwindigkeit der Algorithmen im Folgenden untersucht.

2 Grundlagen

2.1 Definitionen

Um Reinforcement Learning im Folgenden besser beschreiben zu können, ist zunächst die Klärung einiger Grundbegriffe nötig. Diese sind aus der englischen Sprache entstanden, auf eine Übersetzung dieser Begriffe in das Deutsche wurde verzichtet, um eine Vergleichbarkeit zu anderen Werken in diesem Themenbereich zu gewährleisten.

1. **Markov Decision Process**
   Ein Markov Decision Process (MDP) ist ein formales Modell, das zur Beschreibung von Entscheidungsproblemen verwendet wird, bei denen eine Entscheidungsträgerin oder ein Entscheidungsträger (oft als Agent bezeichnet) in einer Umgebung handelt und dabei versucht, eine bestimmte Zielsetzung zu erreichen. Ein MDP basiert auf dem Konzept eines Markov-Prozesses, der ein stochastischer Prozess ist, bei dem der zukünftige Zustand nur vom gegenwärtigen Zustand abhängt und nicht von früheren Zuständen.

   Ein MDP besteht aus den folgenden Komponenten:

   (a) *Agent*
   Der Agent ist der Entscheidungsträger, welcher Aktionen in einem Szenario/Umfeld ausführt und dafür eine Belohnung bekommt.

   (b) *Environment*
   Das Environment ist, wie die deutsche Übersetzung schon vermutet lässt, die Umgebung in dem sich der Agent befindet. Das Environment legt dabei die grundlegenden Regeln fest und definiert, welche Aktionen möglich sind. Das Environment trägt somit ausschlaggebend zur Komplexität der zu lösenden Aufgabe bei. In vielen Fällen, so auch in den in dieser Ausarbeitung folgenden Versuchen, ist das Environment eine Simulation. Dies ermöglicht einen deutlich schnelleren Lernprozess, da jegliche Interaktion ohne nennenswerte Verzögerung ausgeführt werden kann. Bei komplexen Aufgabestellungen ist es so zudem möglich, mehrere Agenten parallel zu trainieren.

   (c) *Action*
   Als Action $\boldsymbol{A}$ wird eine Interaktion des Agent mit dem Environment beschrieben. Die Lösung eines Problems kann somit als Abfolge bestimmter Actions angesehen werden. Welche Actions der Agent ausführen kann, hängt dabei von den Grundregeln des Environments ab.

   (d) *State*
   Der State $\boldsymbol{S}$ ist der eindeutige und vollständige Beschreibung des Zustands, in welchem sich das Environment befindet. Aus technischer Sicht ist der State meist ein Vektor, eine Matrix und ein Tensor, welcher alle relevanten Information des aktuellen Zustands enthält.

   (e) *Reward*
   Der Reward $\boldsymbol{R}$ ist die unmittelbare Belohnung, welche der Agent als Feedback zu einer Action erhält. In der Praxis ist dies ein numerischer Wert, welche entweder erhöht oder reduziert werden kann. Der Agent kann so für eine Action belohnt oder bestraft werden, dabei versucht er sein Handeln so auszurichten, dass er die größte mögliche Belohnung erreicht. Die Art und Weise, wie der Reward vergeben wird, bestimmt somit das Verhalten des Agents.

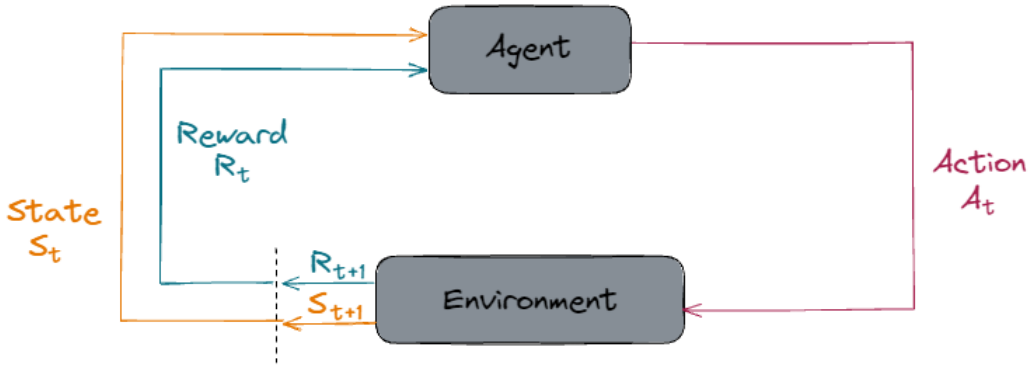


Abbildung 1: Markov Decision Process

Das Ziel des Agenten in einem MDP besteht darin, eine Strategie zu entwickeln, die ihm dabei hilft, die maximale kumulierte Belohnung im Laufe der Zeit zu erhalten. Eine Strategie ist eine Abbildung von Zuständen auf Aktionen, die angibt, welche Aktion der Agent in jedem Zustand ausführen sollte. Die optimale Strategie maximiert die erwartete zukünftige Belohnung über alle Zustände und Aktionen.

Der MDP besitzt, wie eben beschrieben, eine Menge an States $\boldsymbol{S}$, eine Menge an Aktionen $\boldsymbol{A}$ und eine Menge an $\boldsymbol{R}$. In dem Prozess werden die Schritte $\boldsymbol{t = 0,1,2,...}$ durchlaufen und der Agent befindet sich jeweils in einem State $\boldsymbol{S_t} \in \boldsymbol{S}$. Basierend auf diesem State kann der Agent eine Action $\boldsymbol{A_t} \in \boldsymbol{A}$ wählen. Dies ergibt dann das State-Action Paar ($\boldsymbol{S_t}$, $\boldsymbol{A_t}$).

In dem nächsten Schritt $\boldsymbol{t + 1}$ wird das Environment in den State $\boldsymbol{S_{t+1}} \in \boldsymbol{S}$ überführt. Hier bekommt der Agent nun den entsprechenden Reward $\boldsymbol{R_{t+1}} \in \boldsymbol{R}$ für die Action $\boldsymbol{A_t}$, welcher er zuvor in State $\boldsymbol{S_t}$ genommen hat. Dieser Prozess ist in der Abbildung 1 abgebildet.

2. **Episode**
   Als Episode wird ein vollständiger Durchlauf während des Trainings bezeichnet. Jede Episode startet mit dem Anfangszustand des Environments und kann auf mehreren Wegen enden. Im besten Fall wird die Episode beendet, weil die gestellte Aufgabe vom Agent gelöst worden ist. In vielen Fällen wird eine Episode jedoch abgebrochen, weil die maximale Anzahl an Actions überschritten wurde. Eine solche Grenze wird implementiert, um sicherzustellen, dass das Training effektiv und effizient abläuft. Wenn keine Obergrenze festgelegt wird, kann der Agent endlos versuchen, das Ziel zu erreichen, ohne jemals erfolgreich zu sein. Zudem kann so verhindert werden, dass der Agent in einer Schleife von kleinen Belohnungen feststeckt. Die letzte Möglichkeit, wie eine Episode enden kann, ist von Environment definiert, in vielen Fällen kann der Agent durch bestimmte Fehlentscheidungen die Episode beendet.

3. **Policy**
   Im Reinforcement Learning bezeichnet eine Policy $\boldsymbol{\pi}$ eine Funktion, die Entscheidungen trifft, um eine bestimmte Aufgabe zu lösen. Eine Policy entscheidet, welche Aktion ein Agent in einer bestimmten Situation ausführen soll, basierend auf den Informationen, die der Agent in der Vergangenheit gesammelt hat.

   Die Policy wird durch das Optimierungsproblem des Reinforcement Learning bestimmt, das darin besteht, die optimale Strategie zu finden, um die Belohnung des Agents zu maximieren. Die optimale Policy ist diejenige, die in jeder Situation die Aktion empfiehlt, die die höchste erwartete Belohnung ergibt.

Es gibt verschiedene Arten von Policies im Reinforcement Learning, wie beispielsweise deterministische Policies, stochastische Policies und epsilon-greedy Policies. Eine deterministische Policy gibt für jede Situation genau eine Aktion vor, während eine stochastische Policy eine Wahrscheinlichkeitsverteilung über alle Aktionen in einer Situation bereitstellt. Die epsilon-greedy Policy ist eine Mischung aus deterministischen und stochastischen Policies und wählt die Aktion mit der höchsten erwarteten Belohnung mit einer Wahrscheinlichkeit von 1-epsilon und eine zufällige Aktion mit einer Wahrscheinlichkeit von epsilon aus.

4. **Value Function**
   Bei dem Begriff Value Function handelt es sich um eine eine Funktion, die den erwarteten Wert einer bestimmten State-Action-Kombination oder eines States wiedergibt. Der Wert gibt an, wie nützlich es ist, sich in diesem State oder dieser Kombination zu befinden, um das Gesamtziel zu erreichen, also die maximale Belohnung zu erhalten.

   Die Value Function kann verwendet werden, um die optimale Policy zu finden, die die maximale Belohnung im Laufe der Zeit liefert. In Reinforcement Learning gibt es zwei Arten von Value Functions: Die State-Value Function und die Action-Value Function.

   Die State-Value Function $\boldsymbol{V}$ gibt den erwarteten Wert der Gesamtbelohnungen an, den ein Agent in einem bestimmten Zustand erzielen kann. Mit anderen Worten, sie gibt an, wie nützlich es ist, sich in einem bestimmten Zustand zu befinden, um das Ziel der maximalen Belohnung zu erreichen. Die Zustandswertfunktion wird oft als $\boldsymbol{V(s)}$ bezeichnet, wobei s der Zustand ist.

   Die Action-Value Function $\boldsymbol{Q}$ gibt den erwarteten Wert des Gesamtbelohnungen an, den ein Agent in einem bestimmten Zustand erreichen kann, wenn er eine bestimmte Aktion ausführt. Mit anderen Worten, sie gibt an, wie nützlich es ist, in einem bestimmten Zustand eine bestimmte Aktion auszuführen. Die Aktionswertfunktion wird oft als $\boldsymbol{Q(s,a)}$ bezeichnet, wobei s der Zustand und a die Aktion sind.

   Value Functions können auf verschiedene Weise geschätzt werden, wie z.B. mit Hilfe von Monte-Carlo-Methoden und Temporal Difference Learning.

   Im weiteren Verlauf der Arbeit wurde sich mit der Action-Value Function $\boldsymbol{Q}$ und dem Temporal Difference Learning auseinandergesetzt.

5. **Temporal Difference Learning**
   Temporal Difference (TD) Learning [**mediumTemporalDifference**] ist eine Methode des Reinforcement Learning, die es einem Agenten ermöglicht, aus Erfahrungen zu lernen, indem er die erzielte Belohnung mit der erwarteten Belohnung vergleicht. Im Gegensatz zu Monte-Carlo-Methoden, die die Gesamtrabatte aus der Erfahrung berechnen, werden bei TD-Learning die Value Functions schrittweise durch den Vergleich von aufeinanderfolgenden Schätzungen aktualisiert.

   Die TD-Learning Methode verwendet dabei die Bellman-Gleichungen, um die Value Functions zu aktualisieren. Diese Gleichungen sind eine Reihe von Gleichungen, die die Beziehung zwischen den Zustands- und Aktionswerten und der optimalen Policy beschreiben.

   Wenn der Agent eine Aktion ausführt und in einen neuen Zustand gelangt, wird die erhaltene Belohnung zusammen mit dem geschätzten zukünftigen Wert des nächsten Zustands verwendet, um eine neue Schätzung der Value Function zu berechnen. Diese Schätzung wird dann mit der vorherigen Schätzung der Value Function kombiniert, um die Value Function schrittweise zu aktualisieren.

   Im TD-Learning gibt es zwei wichtige Methoden: SARSA (State-Action-Reward-State-Action) und Q-Learning.

6. **Optimalität**

   (a) *Optimale Policy*
   Das Ziel beim Reinforcement Learning ist es, eine Policy $\boldsymbol{\pi}$ zu finden, welche die Belohnung des Agenten maximiert. Dafür muss die Policy $\boldsymbol{\pi}$ gefunden werden, welche dem Agenten mehr Belohnung liefert, als jede andere Policy $\boldsymbol{\pi'}$.
   Mathematisch formuliert muss folgendes gelten:

$$\boldsymbol{\pi} \geq \boldsymbol{\pi'} \text{ if } q_\pi(s,a) \geq q_{\pi'}(s,a) \text{ for all } s \in S \text{ and } a \in A \tag{1}$$

   Hierbei ist $q_\pi(s,a)$ die Action-Value Function, welche der optimalen Policy $\boldsymbol{\pi}$ folgt.

   (b) *Optimale Action-Value Function*
   Für die optimale Action-Value Function gilt folgende Gleichung $\boldsymbol{q_*}$:

$$q_*(s,a) = \max_\pi q_\pi(s,a) \tag{2}$$

   Somit liefert $\boldsymbol{q_*}$ die größten erwarteten Belohnungen für jede Policy $\boldsymbol{\pi}$ für jedes mögliche State-Action Paar.

   (c) *Bellman Optimality Equation*
   Die Bellman Optimality Equation für die Action-Value Function beschreibt den optimalen Wert eines State-Action Paares ***(s,a)*** unter der Annahme, dass der Agent die optimale Policy verfolgt.

$$q_*(s,a) = E[R_{t+1} + \gamma \max_{a'} q_*(s',a')] \tag{3}$$

   Sie kann genutzt werden um in einem iterativen Prozess die optimale Action-Value Function $\boldsymbol{q_*}$ zu finden.

7. **Exploration vs Exploitation**
   Exploration bezieht sich auf die Strategie des Agents, neue Entscheidungen zu erkunden und zu lernen, indem er Aktionen ausprobiert, die er noch nicht ausprobiert hat. Der Agent versucht, neue Möglichkeiten zu erforschen, um mehr über die Umgebung zu erfahren.

   Exploitation hingegen bezieht sich auf die Strategie des Agents, Entscheidungen auf der Grundlage der bisherigen Erfahrungen zu treffen, um den maximalen Gewinn zu erzielen. Mit anderen Worten, der Agent nutzt die bisherigen Erfahrungen, um die Aktion auszuwählen, die ihm die höchste Belohnung in der Vergangenheit gegeben hat.

   Ein Gleichgewicht zwischen Exploration und Exploitation ist wichtig, um optimale Entscheidungen zu treffen. Wenn der Agent zu sehr auf Exploration setzt, wird er immer wieder neue Entscheidungen ausprobieren und keine optimale Entscheidung treffen, um den maximalen Gewinn zu erzielen. Wenn der Agent hingegen zu sehr auf Exploitation setzt, wird er sich auf die Entscheidungen konzentrieren, die ihm in der Vergangenheit den höchsten Gewinn gebracht haben, und nicht in der Lage sein, neue Entscheidungen zu erkunden, die möglicherweise noch höhere Gewinne erzielen können.

Daher verwenden Reinforcement Learning-Algorithmen oft eine Strategie namens epsilon-greedy-Strategie, bei der der Agent mit einer Wahrscheinlichkeit von epsilon eine zufällige Aktion auswählt, um Exploration durchzuführen, und mit einer Wahrscheinlichkeit von 1 - epsilon eine Aktion auswählt, die bisher die höchste Belohnung erzielt hat, um Exploitation durchzuführen. Auf diese Weise kann der Agent gleichzeitig erkunden und von seinen bisherigen Erfahrungen profitieren, um optimale Entscheidungen zu treffen.

2.2 Algorithmen

2.2.1 Q-Learning

Q-Learning ist ein iteratives, modellfreies Reinforcement Learning Verfahren aus dem Bereich der Temporal Difference Learning Verfahren, das verwendet wird, um die optimale Action-Value Functon $\boldsymbol{Q_*(s,a)}$ zu lernen, indem es Erfahrungen sammelt und die Action-Values iterativ aktualisiert.
Das Verfahren basiert auf der Bellman Optimality Equation für die Action-Value Function $\boldsymbol{Q_*(s,a)}$, die besagt, dass der optimale Wert eines State-Action Paars $\boldsymbol{(s,a)}$ die maximale erwartete zukünftige Belohnung ist, die durch die Wahl der optimalen Actions in diesem State und danach unter Verwendung der optimalen Policy erreicht wird.
Q-Learning nutzt eine Tabelle, die die Values für jedes State-Action Paar $\boldsymbol{(s,a)}$ speichert. Das Verfahren sammelt Erfahrungen, indem es den Agenten in der Umgebung handeln lässt, und verwendet diese Erfahrungen, um die Values zu aktualisieren. Die Aktualisierung erfolgt durch die Formel:

$$Q(s,a) \leftarrow Q(s,a) + \alpha[R + \gamma \max_{a'} Q(s',a') - Q(s,a)] \tag{4}$$

wobei $\boldsymbol{Q(s,a)}$ das aktuelle Value für das State-Action Paar $\boldsymbol{(s,a)}$ ist, $\boldsymbol{\alpha}$ die Learning Rate ist, $\boldsymbol{R}$ die unmittelbare Belohnung ist, $\boldsymbol{\gamma}$ die Discount Rate ist, die die zukünftige Gesamtbelohnung gewichtet, und $\boldsymbol{\max_{a'} Q(s',a')}$ die erwartete zukünftige Gesamtbelohnung ist, die durch Wahl der optimalen Action $\boldsymbol{a'}$ im nächsten Zustand $\boldsymbol{s'}$ erreicht wird.
Der Agent verwendet eine Exploration-Exploitation Strategie, um die Umgebung zu erkunden und gleichzeitig die bereits bekannten Informationen zu nutzen. Die Exploration wird durch zufällige Actions gesteuert, während die Exploitation durch die Wahl der Actions mit den höchsten Action-Value erreicht wird.

2.2.2 SARSA

Ähnlich wie bei Q-Learning basiert SARSA auf der Bellman Optimality Equation, aber im Gegensatz zu Q-Learning lernt SARSA direkt die Aktionswerte der Policy, die tatsächlich ausgeführt wird. Dies bedeutet, dass SARSA bei der Aktualisierung der Action-Values die nächste Action $\boldsymbol{a'}$ basierend auf der aktuellen Policy wählt, anstatt die Action mit dem höchsten Action-Value zu wählen, wie dies bei Q-Learning der Fall ist. Daher spricht man bei Q-Learning auch von einem off-Policy Algorithmus, während SARSA ein on-Policy Algorithmus ist.
Die Aktualisierungsregel von SARSA ist wie folgt:

$$Q(s,a) \leftarrow Q(s,a) + \alpha[R + \gamma Q(s',a') - Q(s,a)] \tag{5}$$

wobei der Unterschied in dem Term $\gamma Q(s',a')$ liegt. Denn hierbei wird wieder das epsilon-greedy Verfahren benutzt um den Action-Value zu bestimmen, im Vergleich zu dem maximalen Value, welcher bei Q-Learning gewählt wurde.

2.3 Hyperparameter

Unter Hyperparametern versteht man Einstellungen oder Konfigurationen für einen Maschine-Learning-Algorithmus, um das Verhalten und die Leistung zu steuern. Diese werden vor Beginn des Trainings festgelegt. Da die Wahl der Hyperparameter einen großen Einfluss auf die Fähigkeit der Modelle hat, ist es für den späteren Vergleich wichtig diese Parameter für jeden Algorithmus zu optimieren, um einen fairen Vergleich gewährleisten zu können.

- **Anzahl an Episoden**
  Wie bereits bei 2.1 Definitionen beschrieben, stellt eine Episode einen vollständigen Durchlauf des Environments dar. Die Anzahl dieser Durchläufe währende des Trainingsprozesses wird als Hyperparameter festgelegt. Umso mehr Durchläufe der Agent zur Verfügung hat, um so häufiger kann er dazulernen und sich anpassen, somit hat dieser Hyperparameter einen großen Einfluss auf die Leistung des Agent. Die Anzahl der Episoden steht zudem im direkten Zusammenhang zu der für das Training benötigte Zeit, umso mehr Episoden durchlaufen werden, umso länger dauert der gesamte Trainingsprozess. Die optimale Anzahl der Episoden hängt von verschiedenen Faktoren wie der Komplexität der Umgebung, der Anzahl der States sowie dem verwendeten Lernalgorithmus ab. Bei zu wenigen Episoden ist der Algorithmus nicht in der Lage sein volles Potenzial zu entfalten. Zu viele Episoden können, besonders bei komplexen Modellen, zu Overfitting führen, dabei lernt der Agent die Umgebung auswendig und kann so nicht auf neue Situationen reagieren.

- **Discount Faktor**
  Der Discount Faktor bestimmt, wie stark der Agent eine Belohnung in der Zukunft gewichtet im Vergleich zu einer sofortigen Belohnung. Ein höherer Discount-Faktor bedeutet, dass der Agent die zukünftigen Belohnungen stärker gewichtet und daher eher langfristige Entscheidungen trifft. Ein niedrigerer Discount-Faktor führt dagegen zu kurzfristigeren Entscheidungen, da zukünftige Belohnungen weniger stark gewichtet werden. Der Wertebereich für diesen Hyperparameter liegt zwischen null und eins. Im ersten Moment könnte man jetzt denken, dass der höchstmögliche Discount Faktor der beste sei, da so der komplette Fokus auf dem Endergebnis der Episode liegt. Dies ist aber in den meisten Fällen nicht korrekt. Denn nicht jede Entscheidung hat einen Effekt, der bis zum Ende der Episode reicht, der Agent wäre dann nicht in der Lage solche Entscheidungen zu treffen. Zudem würden bei jeder Entscheidung eine enorme Menge an irrelevanten Informationen mit berücksichtigt, welche den Lernprozess enorm erschweren. Wie auch die maximale Anzahl an Episoden, hängt auch der optimale Wert für diesen Hyperparameter von der Komplexität der Problemstellung und dem verwendeten Lernalgorithmus ab. Auch die zeitliche Entfernung zwischen den Aktionen und den Belohnungen ist relevant.

- **Learning Rate**
  Die Learning Rate gibt an, wie stark das Verhalten des Agents nach jeder Episode aufgrund der Geschehnisse in der Episode angepasst wird. Bei einer hohen Learning Rate werden die Gewichte des Agent stark verändert, und somit das Verhalten stark verändert. Auf der einen Seite ist das gut, um das Training zu beschleunigen, und so schneller zum Ergebnis zu kommen. Gleichzeitig führt eine zu hohe Learning Rate dazu, dass das Modell instabil wird und keine Konvergenz erreicht, somit nie das Optimum erzielt. Eine zu niedrige Learning Rate führt dagegen dazu, dass das Modell zu langsam lernt und möglicherweise in einem lokalen Minimum stecken bleibt. Dies führt dazu, dass das Modell eine sehr lange Zeit zum Konvergieren benötigt oder möglicherweise nie die optimale Leistung erreicht. Um die Vorteile der hohen sowie der niedrigen Learning Rate nutzen zu können, wird eine abnehmende Learning Rate verwendet. Der Trainingsprozess wird mit einer hohen Learning Rate begonnen, um die Grundlagen schnell

zu erlernen und lokale Minima zu vermeiden. Umso weiter das Training fortschreitet, umso mehr wird die Learning Rate reduziert, somit wird das Model stabiler und auf die optimale Leistung fein abgestimmt.

Es gibt verschiedene Methoden, um die Learning Rate zu reduzieren. Eine einfache Methode ist die Verwendung einer konstanten Decay-Rate, die angibt, um wie viel die Learning Rate nach jeder Episode oder nach einer bestimmten Anzahl von Iterationen reduziert wird. Eine weitere Methode ist die Verwendung von adaptiven Decay-Methoden, bei denen die Learning Rate dynamisch anhand von Metriken wie der Performance des Modells angepasst wird. Auch eine exponentielle Abnahme der Learning Rate ist möglich.

- **Exploration Rate**

3 Implementierung

3.1 Probleme

1. Taxi
   Für das Taxiproblem wurde ein fünf mal fünf großen Feld implementiert, der Agent (Taxifahrer) hat dabei die Möglichkeit, sich in alle vier Himmelsrichtungen zu bewegen. Die Aufgabe besteht darin, den Passagier, welcher sich in einer zufälligen Ecke des Feldes befindet, abzuholen und ihn dann zu seinem gewünschten Ziel zu bringen. Das gewünschte Ziel des Passagiers ist immer einer der verbleibenden Ecken des Spielfeldes. Um diese Aufgabe noch etwas zu erschweren, wurden zusätzlich noch Wände in das Feld mit integriert. Diese befinden sich immer an den gleichen Positionen und verhindern Bewegungen in bestimmte Richtungen.

   Für das Bewältigen dieser Aufgabe kann der Agent zu jedem Zeitpunkt zwischen sechs Actions entscheiden; Bewegung in jeweils einer der vier Himmelsrichtungen, aufnehmen des Passagiers und das Absetzen des Passagiers. Natürlich sind nicht alle Actions zu jedem Zeitpunkt sinnvoll.

   Der Zustand des Feldes kann durch 500 diskrete States beschrieben werden. Diese Anzahl ergibt sich aus der Multiplikation der 25 möglichen Position des Taxis, der fünf möglichen Positionen des Passagiers (beinhaltet den Fall, dass sich der Passagier im Taxi befindet), mit den vier möglichen Zielorten.

   Da das Abliefern des Passagiers an dem richtigen Strandort das Ziel der Aufgabe ist, bekommt der Agent dafür auch die höchste Belohnung. Damit die Erfüllung der Aufgabe so effizient wie möglich durchgeführt wird, bekommt der Agent für jede Action, die er durchführt und die nicht zu einer Belohnung führt, eine kleine Strafe. Zuletzt wird der Agent stark bestraft, wenn er den Passagier an dem falschen Ort absetzt oder versucht, einen Passagier an einer falschen Position aufzunehmen.

2. Cliff
   Wie für das Taxiproblem wurde auch für das Cliff Walking Problem ein Feld implementiert, dieses Mal allerdings in der Größe von vier mal zwölf. Die Aufgabe des Agenten besteht darin, von einer Seite des Feldes zur anderen zu gelangen, ohne dabei in die auf dem Feld befindliche Kippe zu fallen. Als Klippe wurden alle Felder definiert, welche sich auf dem direkten Weg zwischen Agent und Ziel befinden. Der Agent hat somit die Aufgabe, um diese Klippe herum zu navigieren und so das Ziel zu erreichen. Um die Herangehensweise der verschiedenen Algorithmen besser vergleichen zu können, befinden sich bei diesem Experiment alle Objekte (Agent, Klippen, Zielpunkt) zu Beginn jeder Episode an derselben Position.

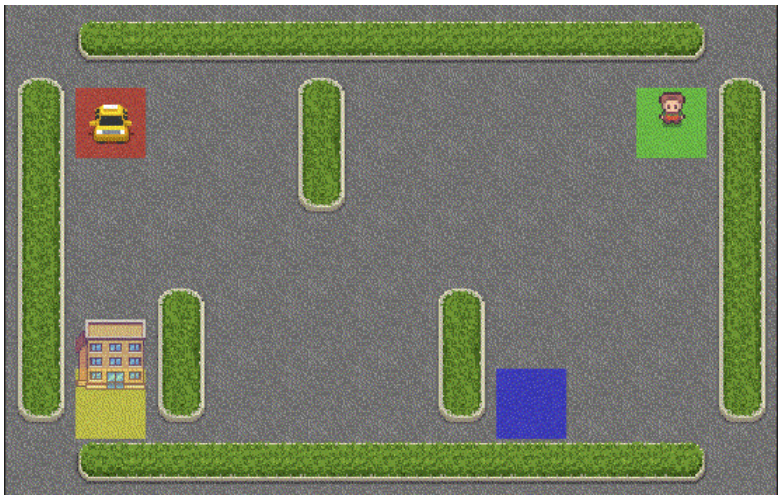


Abbildung 2: Taxi Environment

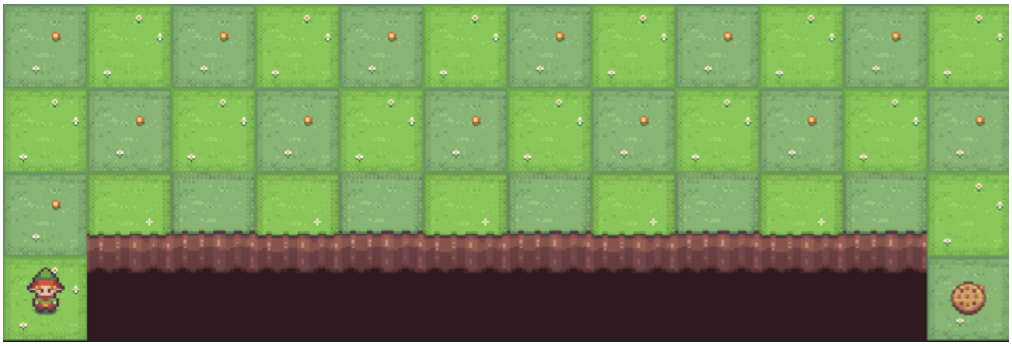


Abbildung 3: Cliff Environment

Damit sich der Agent auf dem Feld bewegen kann, stehen ihm vier verschiedene Actions zur Verfügung, welche den Agent jeweils um ein Feld in einer der Himmelsrichtungen verschiebt.

Um den Zustand des Environments zu beschreiben, ist die aktuelle Position des Agent ausreichend. Insgesamt gibt es 48 (4×12) verschiedene Positionen auf dem Feld. Da das Betreten des als Kippe definierten Bereiches jedoch zum Ende der Episode führt, sind diese Positionen kein gültiger State. Gleiches gilt auch für die Zielposition. Bei zehn Klippen und einem Ziel ergeben sich so 37 States.

Neben des Erreichen des Ziels ist es besonders wichtig, dass der Agent nicht in die Klippe fällt, daher ist dies mit einer hohen Bestrafung für den Agent versehen. Zudem soll das Ziel so schnell wie möglich erreicht werden, daher ist, wie auch bei Taxi Problem, jede Action mit einer kleinen Strafe belegt. Eine explizite Belohnung des Agent ist für diesen Anwendungsfall nicht nötig, da das Ziel zur Beendung der Episode führt und das beste Ergebnis somit das ist, welches zur geringsten Bestrafung führt.

3. Frozen Lake
   Auch das Frozen Lake Problem ist auf einem Gitternetz aus Feldern implementiert. Einige Felder sind Eisplatten, die der Agent sicher betreten kann, während andere Felder Löcher darstellen, in die der Agent fallen kann und damit das Spiel verliert. Das Ziel des Agenten besteht darin, sicher zum Ziel zu gelangen, welches sich auf der anderen Seite des Sees (Gitternetz) befindet.

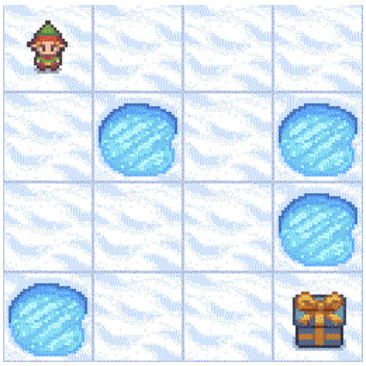


Abbildung 4: Frozen Lake Environment

Die besondere Herausforderung bei diesem Problem sind die Eisplatten, bewegt sich der Agent auf einer dieser Platten in eine bestimmte Richtung besteht eine Change, dass der ausrutscht und sich in eine andere Richtung bewegt.

Wie auch beim Cliff Problem kann der Agent nur durch Bewegung mit dem Environment interagieren und hat somit vier Actions zur Verfügung.

Eine weitere Herausforderung besteht darin, dass die Positionen der Löcher nicht fest sind, sondern bei Beginn jeder Episode zufällig festgelegt werden. Dies muss beim Abbilden des Zustandes des Environments als States berücksichtigt werden. Ein vier mal vier großes Environment mit drei Löchern hat somit 4368 mögliche States. Dies setzt sich zusammen aus den 12 möglichen Positionen des Agenten (16 Felder minus die Löcher und des Ziels) multipliziert mit den 364 möglichen Kombinationen, die 3 Löcher auf 14 freie Flächen zu verteilen.

Die Struktur des Rewards ist für dieses Problem recht simpel, der Agent wird belohnt, wenn er das Ziel erreicht und bestraft, wenn er in ein Loch fällt.

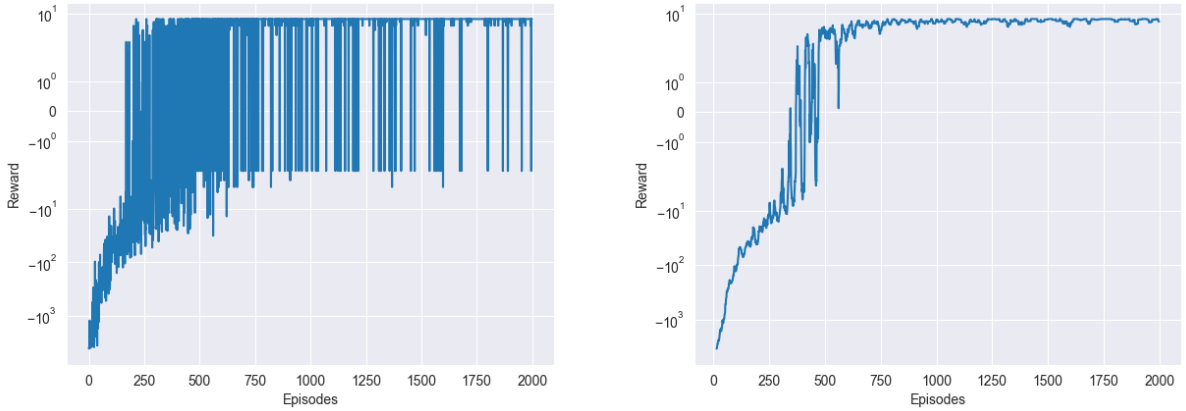
3.2 Optimierung der Hyperparameter

Die Theoretischen Grundlage der Hyperparameter wurde bereits in 2.3 Hyperparameter erläutert, um konkreten Werte für die im Rahmen dieser Arbeit untersuchten Problem zu finden, wurden einige Versuche durchgeführt. Als Ausgangslage wurde die in Hyperparameter Ausgangslage dargestellten Werte angenommen. Im Folgenden werden diese initialen Werte anhand von Experimenten für die verschiedenen Algorithmen und Probleme optimiert.

Tabelle 1: Hyperparameter Ausgangslage

Paramater	Wert
num_episodes	1000
max_steps_per_episode	1000
learning_rate	0.1
discount_rate	0.99
exploration_rate	1
max_exploration_rate	1
min_exploration_rate	0.01
exploration_decay_rate	0.005

- Anzahl an Episoden
  Als optimale Anzahl an Episoden wird der Punkt gewählt, ab wann sich der Agent nicht mehr verbessert, so wird ein unnötig langes Training vermieden. Dieser Punkt lässt sich sehr gut in einer grafischen Darstellung der erreichten Belohnung des Agenten im Vergleich zu den während des Trainings durchlaufenen Episoden ablesen. Die Rohdaten aus dem Training sind aufgrund der Exploration sehr verrauscht, und der eigentliche Trainingsfortschritt ist nur schwer zu erkennen. Um die Grafik lesbarer zu machen wurde ein gleitender Durchschnitt auf die Daten angewandt, der Effekt dieser Operation wurde in der Abbildung Trainingsverlauf Taxiproblem veranschaulicht.



(a) Rohdaten (b) Gleitender Durchschnitt

Abbildung 5: Anwendung gleitender Durchschnitt

- Maximale Anzahl an Steps pro Episode
  Für die Festlegung dieses Hyperparameters wurde zunächst der Trainingsverlauf der Basiswerte mit dem Trainingsverlauf von deutlich erhöhten und reduzierten Werten verglichen. So konnte der Einfluss des Parameters analysiert werden und eine erste Tendenz wurde festgestellt. Darauf folgenden weitere Vergleiche, um den optimalen Wert zu bestimmen.

- Learning Rate
  Wie auch bei der Festlegung der maximalen Anzahl an Steps pro Episode wurde auch für die Learning Rate der initiale Wert verändert und die Auswirkungen anhand von Grafen analysiert.

- Discount Rate
  Die Discount Rate wurde ebenfalls ermittelt, indem die Auswirkungen einer Veränderung des Hyperparameters untersucht wurden. Der initial gewählte Wert für die Discount Rate ist sehr nah am Maximum des möglichen Wertebereichs. Das Erhöhen dieses Hyperparameter führte daher in keinem Versuch zu einer Veränderung der Ergebnisse. Die Reduzierung des Parameters wirkte sich jedoch auf das Training des Agenten aus.

- Exploration Rate
  Damit sich der Wert der Exploration Rate über den Verlauf des Trainings verändert, sind insgesamt vier Parameter implementiert. Die Obergrenze für den Wert, sowie der Startwert, sind initial auf Eins festgelegt. Zu Beginn hat der Agent noch kein Wissen über das Environment, eine Exploitation ist somit nicht sinnvoll. Eine Reduzierung des Wertes führt somit nur zu einem

langsamen Trainingsverlauf. Die Obergrenze soll den höchsten erreichbaren Wert beschreiben, da die Exploration Rate während des Trainings ausschließlich reduziert wird, muss dieser Wert mit dem Startwert identisch sein. Die Abnahmerate und der minimale Wert sind durch Experimente werten, wie die vorherigen drei Hyperparameter durch Versuche ermittelt.

3.3 Vergleichen von Algorithmen

Nach dem im vorangegangen Kapitel die Implementation des Vergleichens der Hyperparameter beschrieben wurde, befasst sich dieser Abschnitt mit dem Verlgeichen der beiden Algorithmen Q-Learning und SARSA. Um den bestmöglichen Vergleich zu gewährleisten wurden geeignete Parameter für beide Algorithmen und die jeweiligen Probleme gewählt. Bei der Analyse wurden alle drei Probleme, welche in 3.1 bereits erläutert wurden, angewendet.

- Bei dem Taxiproblem wurden die in Tabelle 2 dargestellten Paramater für einen sinnvollen Vergleich der beiden Algorithmen gewählt. Das Taxi-Problem hat im Vergleich zu den anderen Algorithmen viele unterschiedliche States, daher brauche es eine hohe Anzahl an Episoden um diese zu lernen bzw. zu optimieren.

- Bei dem Cliffproblem handelt es sich um ein einfacheres Problem, daher wurden die Anzahl der Episoden etwas zurückgesetzt. Die für dieses Problem genutzten Parameter sind in Tabelle 3 beschrieben.

  Die Besonderheit bei dem Cliffproblem ist, dass es lediglich negative Belohnungen gibt und die Klippe eine vielfach höhere Bestrafung hat, als ein normaler Schritt, welcher nicht die Klippe hinunter führt.

- Bei dem Frozen Lake Problem handelt es sich widerum um ein komplexeres Problem, welches mehr Episoden als das Cliffproblem benötigt, um zu einem Optimum zu konvergieren. Wichtig bei diesem Problem ist ebenfalls eine höhere Anzahl an maximalen Steps, bevor der Algorithmus in die nächste Episode geht. Der Grund dafür ist, dass es lediglich für das Zielfeld als Belohnung $+1$ gibt, während alle anderen Aktionen im Spiel mit dem Reward 0 belegt sind. Somit muss der Agent es über die Exploration erstmal schaffen das Zielffeld zu entdecken, um sich dann im nachhinein dafür zu optimieren. Erschwert wird das ganze zusätzlich durch die Tatsache, dass neben der Exploration Rate (epsilon) auch das Environment für zufällige Bewegungen sorgt. So ist die Wahrscheinlichkeit, wenn der Agent nach links gehen möchte lediglich P(links) = 1/3 und die Chancen für die Richtungen oben und unten auch jeweils P(oben) = 1/3 und P(unten) = 1/3. Die Tabelle 4 führt die genutzten Parameter für den Vergleich von Q-Learning und SARSA dieses Problems auf.

Tabelle 2: Taxi Paramater

Paramater	Wert
num_episodes	3000
max_steps_per_episode	1000
learning_rate	0.1
discount_rate	0.99
exploration_rate	1
max_exploration_rate	1
min_exploration_rate	0.05
exploration_decay_rate	0.005

Tabelle 3: Cliff Paramater

Paramater	Wert
num_episodes	1500
max_steps_per_episode	1000
learning_rate	0.1
discount_rate	0.99
exploration_rate	1
max_exploration_rate	1
min_exploration_rate	0.05
exploration_decay_rate	0.005

Tabelle 4: Frozen Lake Paramater

Paramater	Wert
num_episodes	2000
max_steps_per_episode	10000
learning_rate	0.1
discount_rate	0.99
exploration_rate	1
max_exploration_rate	1
min_exploration_rate	0.05
exploration_decay_rate	0.005

Zum Vergleich der Algorithmen wurden die Belohnungen und Steps während des Trainings aufgenommen und anschließend in Plots anschaulich dargestellt. Dabei wurde je nach Problem eine logarithmische Skalar auf der y-Achse genutzt, um die Daten anschaulicher zu machen. Zusätzlich wird ein Rolling Window Ansatz gewählt, sodass immer 15 Werte zusammengefasst werden. Dies sorgt dafür das einzelne Peaks zwar weniger auffallen, dafür der Gesamttrend im Verlauf der Trainingsepisoden aber besser zu erkennen ist.

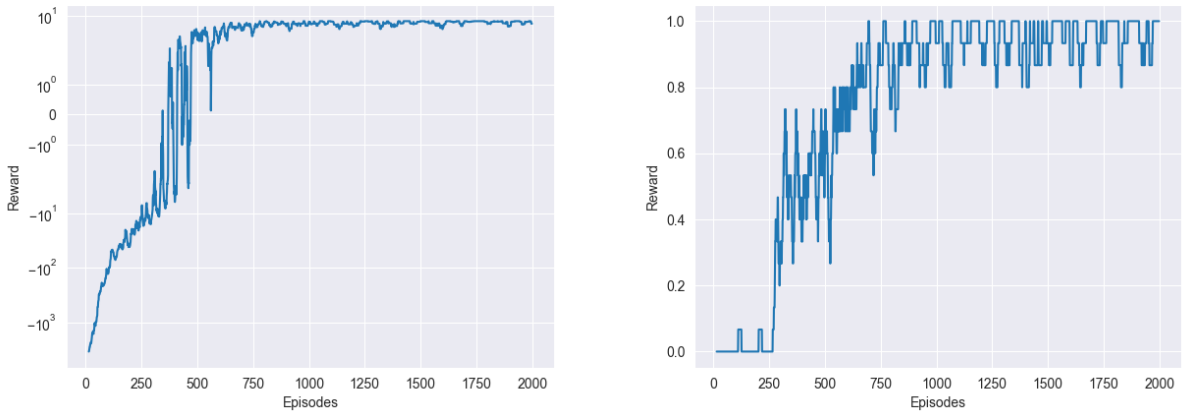
4 Ergebnisse

4.1 Optimierung der Hyperparameter

4.1.1 Anzahl an Episoden

- Q-Learning
  Der in Abbildung 6a dargestellt Graph ziegt den Trainingsverlauf von Q-Learning bei dem Taxiproblem. Dort lässt sich erkennen das sich der Reward ab Episode 1300 kaum noch verändert, der Agent dementsprechend nicht mehr dazulernt. Dieser Punkt wird als optimalen Wert für die maximale Anzahl an Episode angenommen.

  Um den optimalen Wert für das Cliff und Frozen Lake Problem zu ermitteln, wurden dieselben Schritte durchgeführt. Das Cliff Problem wird schneller von Agent erlernt, daher kann das Training schon nach etwa 1000 Episoden beendet werden. Der Trainingsverlauf des Frozen Lake Problem hat aufgrund der besonderen Rewardstruktur einen anderen Wertebereich (vgl. Abbildung 6b), ein Wert von etwa 600 benötigten Episoden lässt sich trotzdem problemlos ablesen.



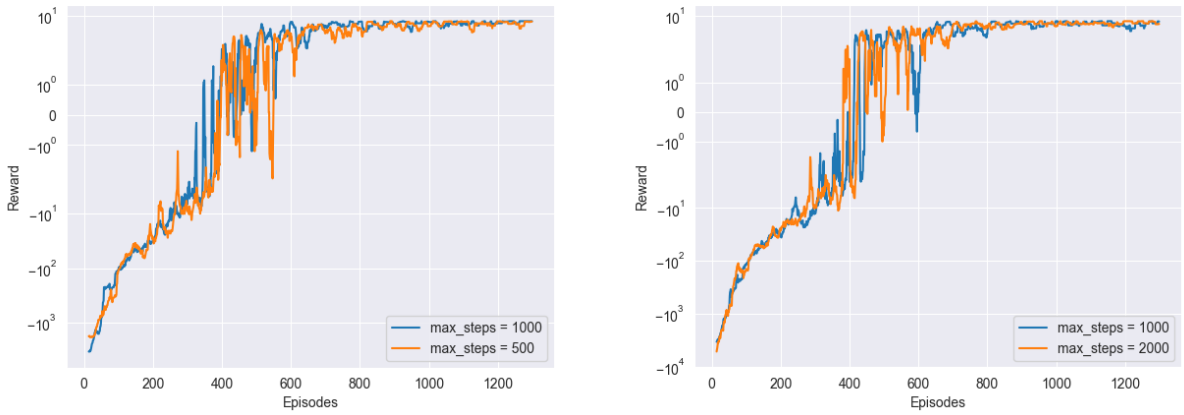
(a) Trainingsverlauf Taxiproblem (b) Trainingsverlauf Frozen Lake

Abbildung 6: Ermittlung der optimalen Anzahl an Episoden

- SARSA

4.1.2 Maximale Anzahl an Steps pro Episode

- Q-Learning
  Abbildung 7a stellt den Vergleich der Trainingsverläufe auf dem Taxiproblem zwischen den anfänglichen 1000 Steps zu 500 Steps dar. Diese deutliche Reduzierung der maximalen Steps hat zu folge, dass der Agent länger ein unstabiles Verhalten aufzeigt, somit ist diese Veränderung nicht sinnvoll. Abbildung 7b zeigt den Effekt deiner Verdopplung des Wertes auf 2000. Zu erkennen ist, dass der Agent in der Mitte des Trainings schneller lernt, diesen Vorsprung verliert er gegen Ende des Trainings jedoch wieder. Bei genauerer Betrachtung lässt sich zu dem feststellen, dass der Verlauf gegen Ende etwas weniger schwankt, und somit der Agent minimal bessere Performance nach dem Training zeigen wird. Eine weitere Erhöhung der maximalen Anzahl an Steps führte in den Versuchen allerdings nicht zu einer Verbesserung des Ergebnisses, verursachtet jedoch höhere Trainingszeiten. Zweittausend wurde somit als optimalen Wert für diesen Hyperparameter für das Taxi-Problem ermittelt.



(a) Vergleich 1000 vs 500

(b) Vergleich 1000 vs 2000

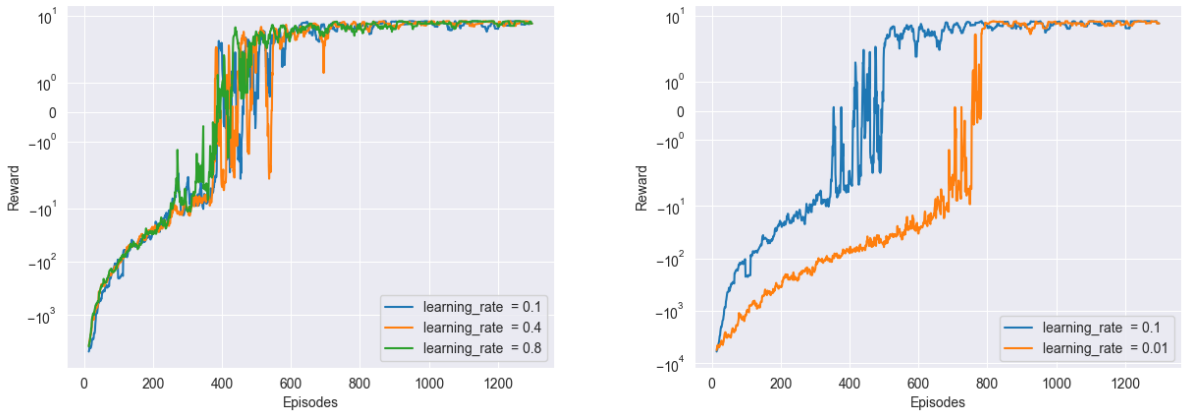
Abbildung 7: Auswirkung der maximalen Anzahl an Episode

Für das Cliff Problem kann mit einem Wert von 750 schon das beste Ergebnis erzielt werden. Dies hängt mit der durchschnittlich niedrigeren Anzahl an Steps, die zum Lösen des Problems benötigt werden, zusammen. Bei der Untersuchung des Frozen Lake Problems konnte kein signifikanter Einfluss des Hyperparameters festgestellt werden, um Trainingszeit, die Trainingszeit gering zu halten wurde 500 als Wert festgelegt.

- SARSA

4.1.3 Learning Rate

- Q-Learning
  Bei dem Vergleich verschiedener Learning Rates konnte festgestellt werden, dass Werte im Wertebereich zwischen 0,1 und 0,8 keinen Einfluss auf das Ergebnis des Trainings haben (vlg. Abbildung 8a). Es kommt zwar zu kleinen Unterschieden in der Mitte des Trainings, diese sind aber eher auf die zufällig Exploration zurückzuführen. Eine zu kleine Learning Rate, wie Abbildung 8b mit einem Wert von 0.01 veranschaulicht, führt jedoch zu einem längeren Trainingsverlauf. Für die weiteren Versuche wurde ein Wert von 0,4 festgelegt, da er sich in der Mitte des untersuchten Intervalls befindet.



(a) Einfluss Learning Rate

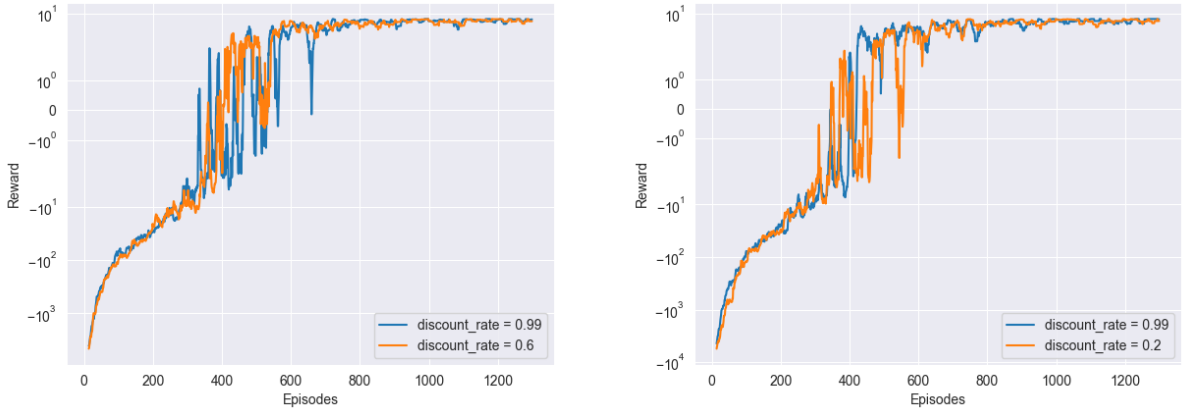
(b) Wahl einer zu geringen Learning Rate

Abbildung 8: Auswirkung der Learning Rate

- SARSA

4.1.4 Discount Faktor

- Q-Learning
  Eine Reduzierung der Discount Rate auf 0,6 führte bei Q-Learning und dem Taxiproblem zu einem konstanteren Trainingsverlauf (vgl. Abbildung x). Die Auswirkung einer weiteren Reduzierung des Parameters auf 0,2 ist in Abbildung x zu entnehmen und führte zur leichten Verschlechterung des Ergebnisses. Da der Wert von 0,6 die besten Ergebnisse lieferte, wird dieser Wert für die folgenden Versuche verwendet.



(a) Vergleich zwischen 0,99 vs 0,60

(b) Vergleich zwischen 0,99 vs 0,20

Abbildung 9: Auswirkung der Discount Rate

Bei dem Cliff Problem führt eine Absenkung des Wertes zu einer minimalen Verschlechterung der Ergebnisse, somit wird der Ausgangswert als Optimal angenommen. Dieser hohe Initialwert

verursachte bei dem Lake Problem häufiger einen Leistungseinbruch, der Agent war dann nicht mehr in der Lage, die Aufgabenstellung zu lösen. Durch eine Reduzierung des Wertes auf 0,2 konnte dieses Problem behoben werden.

- SARSA

4.1.5 Exploration Rate

- Q-Learning

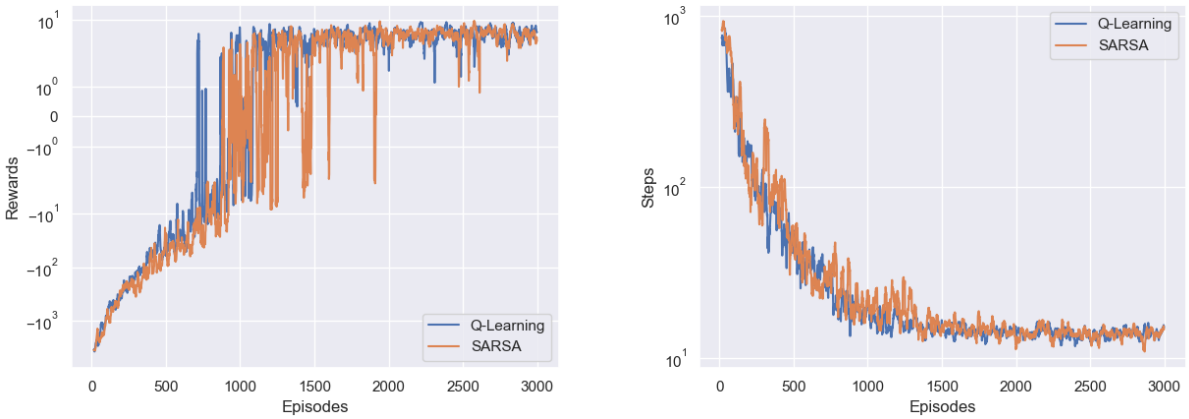
- SARSA

4.2 Vergleichen von Algorithmen

Im folgenden werden die Ergebnisse beschrieben, welche bei dem Vergleichen der Algorithmen Q-Learning und SARSA entstanden sind. Dabei wurde der Vergleich an den drei Reinforcement Learning Problemen Taxi, Cliff und Frozen Lake durchgeführt. Es wurden jeweils passende Paramater für das entsprechende Problem gewählt.

4.2.1 Taxi

In der Abbildung 10 sind die Ergebnisse für den Vergleich des Q-Learning und SARSA Algorithmus dargestellt. In der linken Abbildung 10a sind die Rewards in dem zeitlichen Verlauf über die 3000 Trainingsepisoden abgebildet. Zu erkennen ist hier, dass beide Algorithmen nach ca. 1500 Episoden Richtung Optimum konvergieren. Der Q-Learning Algorithmus trainiert jedoch ein wenig schneller, was durch die anfänglich stärkere Steigung und höheren Peaks zu Beginn verdeutlicht wird.
Zusätzlich ist es auffällig, dass SARSA im späteren Verlauf auch größere Peaks nach unten hate und generell ein wenig schlechter performiert als der Q-Learning Algorithmus. Somit lässt sich insgesamt für die Belohnung feststellen, dass Q-Learning schneller zu einem besseren Ergebnis kommt als SARSA.
In der zweiten Abbildung 10b sind die Anzahl der Steps abgebildet, welche der Agent jede Episode durchläuft, bis er entweder bei der maximalen Anzahl angekommen ist oder den Passagier erfolgreich zu seinem Zielort gebracht hat. Der Verlauf der beiden Kurven von Q-Learning und SARSA ist hier sehr ähnlich. Lediglich bei ca. 400 Episoden und 1300 lassen sich leicht höhere Peaks vom SARSA Algorithmus ablesen. Dies spricht dafür, dass SARSA in diesen Phasen etwas langsamer gelernt hat als der Q-Learning Algorithmus.
Wichtig zu erwähnen ist zusätzlich, dass beide Algorithmen nach 3000 Episoden im Test perfekt opimiert sind. Im Test nutzen die trainierten Modelle keine Exploration Rate mehr, sondern gehen lediglich den Weg mit den höchsten Q-Values. Dies führt bei dem Taxi Problem bei beiden Algorithmen zu sehr ähnlichen Ergebnissen. Starten beide in dem gleichen State, laufen sie die fast die gleichen Wege um den Passagier abzuholen und zum Ziel zu bringen.

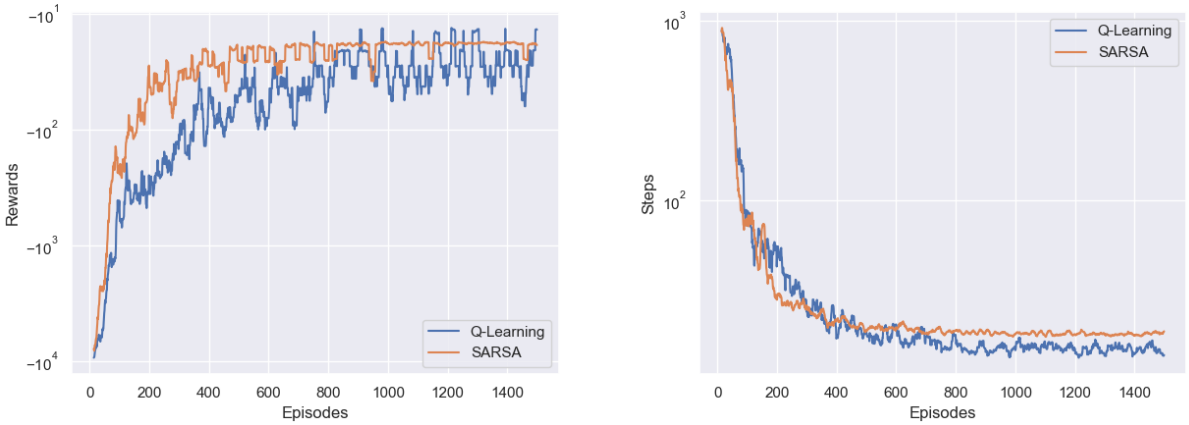


(a) Rewards (b) Steps

Abbildung 10: Taxi Problem über 3000 Episoden während dem Training

4.2.2 Cliff

Mit dem Cliff Problem lässt sich einer der besonderen Unterschiede zwischen Q-Learning und SARSA sehr gut zeigen. In Abbildung 11 sind wie in dem vorherigen Beispiel der Verlauf der Rewards (Abbildung 11a) und die Anzahl an Steps (Abbildung 11b) über die Episoden im Training dargestellt. Bei den Rewards ist zu erkennen dass SARSA zum Einen schneller lernt, dass heißt die Kurve geht zu Beginn schneller nach oben. Zum Anderen stagniert es auf einem höheren Reward zum Ende hin. Auch in der Abbildung 11b lässt sich ein Unterschied feststellen. Zu Beginn Verlaufen Sie zwar sehr ähnlich zum Ende hin ist es aber auffällig, dass SARSA bei eine höheren Anzahl an Steps stagniert, während der Verlauf bei Q-Learning deutlich niedriger, aber auch mit mehr Schwankungen vorliegt.



(a) Rewards (b) Steps

Abbildung 11: Cliff Problem über 1500 Episoden während dem Training

Dies liegt daran, dass SARSA ein on-policy Algorithmus ist. Dies bedeutet, dass es bei der Berechnung der Q-Value berücksichtigt, dass es eine Wahrscheinlichkeit gibt, zu der der Agent im nächsten Schritt explorieren statt exploiten wird. Gerade bei dem Cliff Problem ist dies besonders spannend, da eine

Exploration, welche in der Klippe endet, sehr fatal sein kann. In der Abbildung 12 sind die Wege dargestellt, welche die Algorithmen wählen. SARSA entscheidet sich aufgrund der on-policy Strategie für den “safe path” während Q-Learning für den “optimal path” geht. Beide Wege haben durchaus ihre Vor- und Nachteile und können je nach Anwendungsfall den subjektiv besseren Weg darstellen. Aus den eben genannten Gründen nennt man Q-Learning auch einen greedy-Algorithmus, da er für den möglichst kürzten Weg geht, während SARSA etwas vorsichtiger, also weniger greedy, agiert.

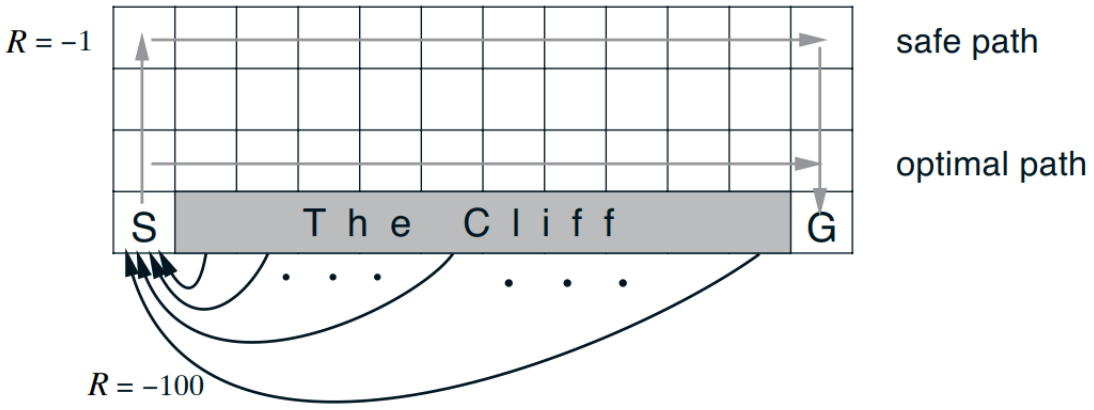
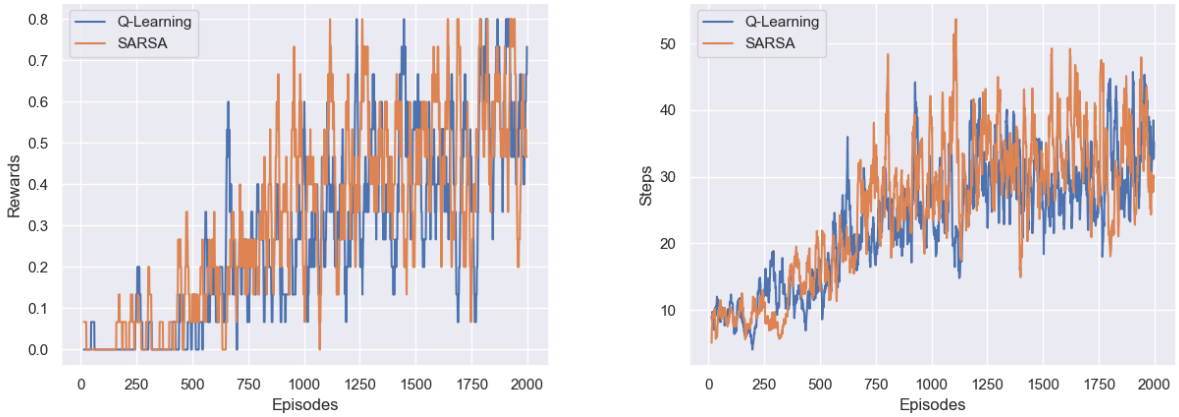


Abbildung 12: Cliff Q-Learning vs SARSA Pfad

4.2.3 Frozen Lake

Das Frozen Lake Problem hat die Besonderheit, dass es nicht zu 100% optimiert werden kann. Dies ist auch in den Abbildungen der Rewards (Abb. 13a) und der Steps (Abb. 13b) zu erkennen. Beide Graphen verlaufen zwar jeweils ähnlich und stagnieren auf einem Niveau unter dem Optimum. Bezüglich der Unterschiede lässt sich hier kein sichtbarer Unterschied zwischen der Performance der Algorithmen aufzeigen. Bei den Rewards brauchen beide Algorithtmen ca. 500 Episoden um das Zielfeld regelmäßiger zu finden. Nach 2000 Episoden schaffen beide es teilweise zu 80% während des Trainings. Anhand des Step Graphen ist zu erkennen, dass beide Algorithmen zu Beginn schnell in eines der Löcher fallen und somit die Episoden frühzeitig enden. Über die Zeit schaffen sie es aber länger auf dem Frozen Lake zu laufen. Optimalerweise würde die Anzahl der Steps gegen Ende wieder etwas sinken, dies ist aber in den ersten 2000 Episoden nicht der Fall.



(a) Rewards

(b) Steps

Abbildung 13: Frozen Lake Problem über 2000 Episoden während dem Training

Dies kann daran liegen, dass das Environment durch den Zufallsparameter $is\_slippery = True$ für sehr zufällige Episoden sorgt. Dies ist auch daran zu erkennen, dass trotz des Rolling Windows, die Graphen sehr stark schwanken.

5 Zusammenfassung und Ausblick

Im Rahmen der Arbeit konnte sich ausgiebig mit den Grundlagen des Reinforcement Learnings auseinandergesetzt werden. Es wurde anhand von drei einfacheren Szenarien zwei klassische Algorithmen für das Temporal Difference Learning getestet. Für die beiden Algorithmen gibt es eine Vielzahl von Parametern, welche je nach Anwendungsfall passend gewählt werden müssen. Im Rahmen der Arbeit konnten die Einflüsse dieser Parameter aufgezeigt werden. Für die Temporal Difference Learning Algorithmen Q-Learning und SARSA konnte ein ausführlicher Vergleich für die drei Probleme durchgeführt werden. Hierbei wurde festgestellt, dass die Algorithmen je nach Problem zu unterschiedlichen Ergebnissen führen. So hat das Q-Learning beim Taxi Problem etwas besser abgeschnitten, während SARSA den wohl sichereren Weg bei dem Cliff Problem gewählt hat.
In weiteren Arbeiten ist es von Interesse, sich mit komplexeren Problemen und dem Thema Deep Reinforcement Learning zu beschäftigen. Dies ermöglicht das Anwenden von Reinforcement Learning auf weitaus komplexere Herausforderungen, welche nur schwer mit den in dieser Arbeit genutzten Algorithmen lösbar sind.