FP-Versuch

Reinforced learning für intelligente Brownsche Dynamik

Maxim Root, Thomas Bengardt

28. April, 2023

1 Einleitung

Dieser FP-Versuch handelt vom Q-Learning Algorithmus, welcher dazu dient, Maschinen gewisse Aufgaben beizubringen. Dieser Lernprozess geschieht ganz ohne exemplarische Daten, anhand derer die Maschine lernen könnte, wie bspw. beim super-/unsupervised learning, sondern beruht rein auf der Strategie, mit der man die Maschine trainiert.
Das grundlegende Ziel in unserem Fall ist, dass der sogenannte *Agent* sich in einem zuvor generierten *Environment* zu einem zuvor definierten *Target* bewegt. Die Aufgabe des Q-Learning Algorithmus ist also, eine Strategie für den *Agent* zu entwickeln, schnellstmöglich zum *Target* zu gelangen. Der Aspekt der Brownschen Dynamik kommt ins Spiel durch eine zusätzliche Zufallsbewegung des Agents, wessen Stärke durch eine effektive Diffusionskonstante variiert wird. Das vorliegende Problem ist diskretisiert in einem eindimensionalen Raum.
Im Folgenden werden die Ergebnisse der vier Aufgaben dieses Projektes vorgestellt und diskutiert.

2 Aufgabe 1 - MSD

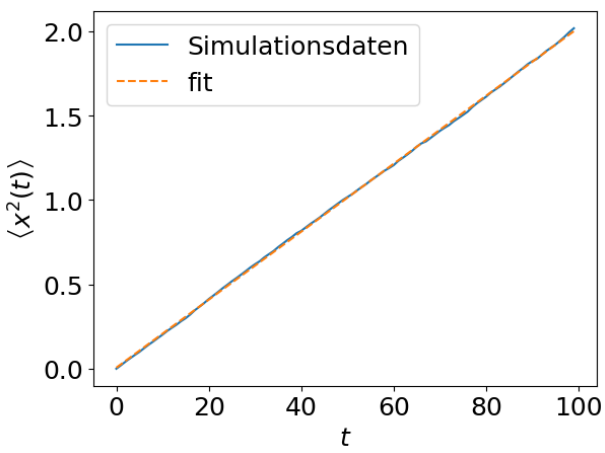
In dieser Aufgabe untersuchten wir das *mean square displacement* (MSD) $\langle x^2(t)\rangle$, welches ein Maß für die Diffusion des Agents darstellt. Aufgrund der festgelegten Größen Δx (Schrittweite) und τ (Zeit zwischen zwei Schritten) wäre die Diffusion über die unten stehende Formel bereits festgelegt. Um dies zu umgehen und unsere Diffusionskonstante D variieren zu können, führen wir eine Wahrscheinlichkeit für Diffusion ein. Hiermit ist es uns möglich, die effektive Diffusionskonstante zu variieren, mittels

$$D := \frac{\Delta x^2}{2\tau} \Rightarrow D_{\text{eff}} = \frac{\Delta x^2}{2\langle\tau\rangle} = \frac{\Delta x^2}{2} p, \qquad p = \frac{1}{\langle\tau\rangle}. \tag{1}$$

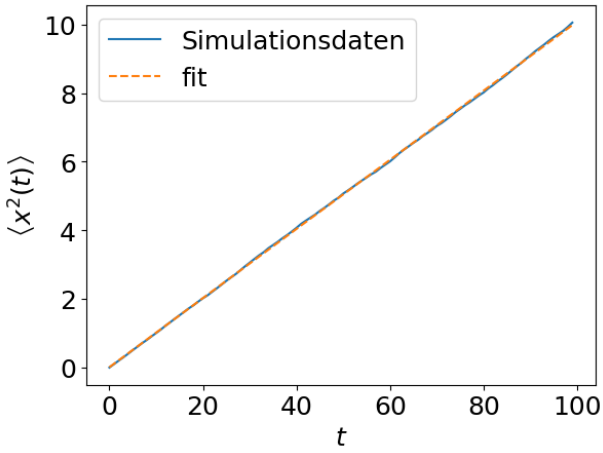
Denn τ stellt hier die Zeit zwischen zwei Diffusionsschritten dar und kann im Mittel durch p verändert werden. Dies gibt uns für die Schrittweise $\Delta x \equiv 1$ den Zusammenhang zwischen Diffusionskonstante und Diffusionswahrscheinlichkeit,

$$p = 2D_{\text{eff}}. \tag{2}$$

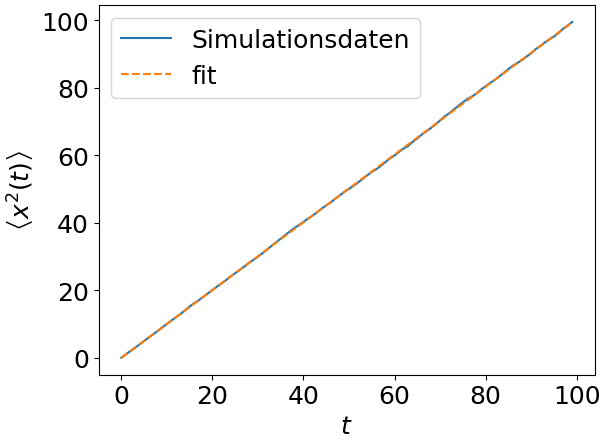
Mithilfe dieser Einführung ließen wir den Agent einen *Random Walk* ausführen und werteten für $N = 10^4$ Episoden das MSD dafür aus (siehe Abb. 1). Die analytische Lösung für das MSD eindimensionaler Brownscher Bewegung ist durch $\langle x^2(t)\rangle = 2D_{\text{eff}} \cdot t$ gegeben.



(a) $D_{\text{eff}} = 0.010070573807380736$



(b) $D_{\text{eff}} = 0.05035252730273028$



(c) $D_{\text{eff}} = 0.5023940030003$

Abbildung 1: Das MSD wurde als Funktion von der Zeit (mit diskreten Zeitabschnitten $\Delta t = 1$) für $D = 0.01$ (a), $D = 0.05$ (b) und $D = 0.5$ (c) geplottet und die effektive Diffusionskonstante D_{eff} wurde mithilfe der obigen Formel und eines linearen Fits ermittelt. Die Legende ist den Plots zu entnehmen. In allen Fällen stimmt D mit D_{eff} bis auf geringe Abweichungen (vermutlich) statistischer Art überein.

Die präzise Übereinstimmung unserer Diffusionskonstanten lässt darauf schließen, dass die Einführung von p eine erfolgreiche Methode darstellt, um trotz diskreter Zeit und Bewegung Diffusion variabel zu modellieren. Jedoch ist bei dieser Implementierung D_{eff} nach oben beschränkt (siehe Gl. (2)), da sie über eine Wahrscheinlichkeit definiert ist.

3 Aufgabe 2 - Implementation

Jetzt kommen wir dazu, die Strategie für den Agent auf Basis des Q-Algorithmus zu implementieren, um den Agent trotz der oben behandelten Diffusion schnellstmöglich zum Target zu bringen. Hierzu führen wir eine Wahrscheinlichkeit ε ein, mit der der Agent einen Zufallsschritt betätigt, um das Environment zu „erforschen“. Mit der Gegenwahrscheinlichkeit $(1-\varepsilon)$ führt der Agent einen Schritt auf Grundlage der zu dem Zeitpunkt bekannten Information über den Weg zum Target in unserem Environment aus. Es ist schlüssig, dass diese Entscheidung immer besser wird, je mehr der Agent das Environment erforscht hat. Aufgrund dessen senken wir die Wahrscheinlichkeit ε zur Zufallsaktion linear mit zunehmender Episode ab, bis sie bei letzter Episode gänzlich verschwindet; der Agent handelt dann vollkommen nach gelernter Strategie (, ist aber immer noch Diffusion ausgesetzt). Da das vorliegende Problem diskret und eindimensional ist, haben wir eine zuvor generierte feste Anzahl an Zuständen N_Z, die man sich als einen Kreisrand vorstellen kann (periodische Randbedingungen). In diesem Environment sind dem Agent die Aktionen nach links Gehen, nach rechts Gehen und Stehenbleiben gegeben. Eine Episode gilt als beendet, wenn der Agent auf dem Target verweilt. Unseren Lernerfolg halten wir in der Matrix Q fest, die Informationen über jede Aktion in jedem Zustand trägt. Demnach ist Q in unserem Fall von der Form $(N_Z)\times(3)$. Mithilfe von Q wird für den Fall, dass der Agent nicht zufällig handelt, entschieden, welche Aktion ausgeführt werden soll. Dies geschieht durch Auswählen der Handlung, die den maximalen Wert zwischen den drei Aktionen besitzt. Man möchte also am Ende des Lernprozesses die Werte in der Matrix möglichst hoch haben, die den Agent auf schnellstem Wege zum Target bringen. Um das umzusetzen, nehmen wir uns folgende Formel zur Hilfe, mit der wir bei jeder Aktion des Agents den vorliegenden Eintrag der Matrix aktualisieren müssen:

$$Q_{ij}^{\text{neu}} = Q_{ij}^{\text{alt}} + \alpha(R + \gamma \max_k Q_{i'k}^{\text{neu}} - Q_{ij}^{\text{alt}}). \tag{3}$$

Hierbei stellt i den Zustand vor der Aktion j dar und i' den Zustand danach. α beschreibt die Lernrate, γ den Discount Faktor und R (für Reinforcement) die Belohnung für das Verbleiben auf dem Target-Zustand. Die Belohnung wird also ausschließlich verliehen, wenn der Agent auf dem Target verweilt und dient dazu, den Agent darauf zu trainieren, diese Aktion zu befürworten. Des Weiteren dient der Discount-Term dazu, von benachbarten Einträgen zu lernen, denn bei Einträgen hoher Gewichtung sorgt der Discount-Term dafür, dass anliegende Zustände davon „profitieren“ (mehr dazu in Aufg. 4). Nach erfolgreich implementierter Funktionsweise wird die Matrix Q über die N Episoden hinweg aktualisiert und darauf perfektioniert, den Agent schnellstmöglich zum Target zu führen. Beurteilen lässt sich das mithilfe der endgültigen Matrixeinträge, wie in Abb. 2 zu sehen ist.

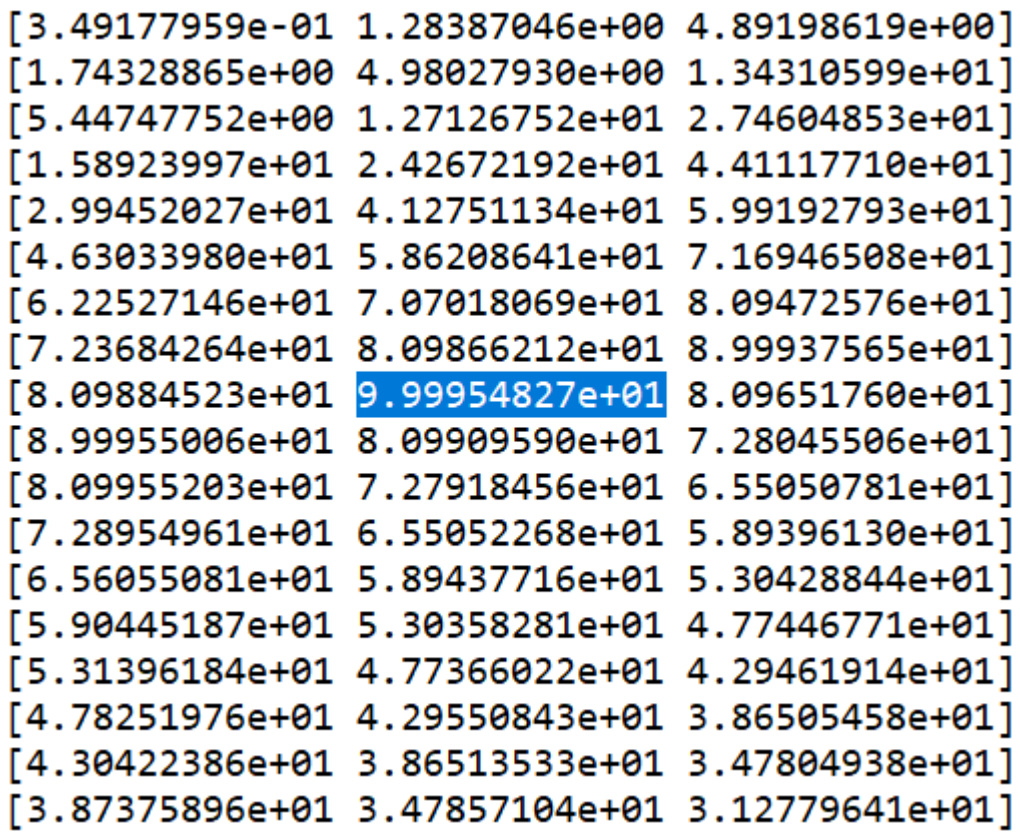


Abbildung 2: Ausschnitt der Matrix Q nach vollständigem Lernprozess mit $N = 10^4$ Episoden. Der markierte Beitrag repräsentiert den Wert für das Verweilen auf dem Target. Dieser ist maximal und Aktionen benachbarter Zustände weisen in Richtung dieses Target-Zustands, wie bei erfolgreichem Lernprozess zu erwarten ist. Der Trend setzt sich bis etwa $\frac{N_Z}{2}$ Zustände vom Target entfernt fort, wo die bevorzugten Aktionen die Richtung wechseln, weil das vorliegende Environment periodische Randbedingungen erfüllt.

4 Aufgabe 3 - Hyperparameter

Wie man sich bereits denken kann, ist die richtige Wahl der Hyperparameter γ und α essenziell für die erfolgreiche Entwicklung einer Lernstrategie. Deshalb schauen wir uns in den kommenden zwei Unterabschnitten genauer an, welche Rolle im Algorithmus der Discount Faktor und die Lernrate jeweils spielen und welche Tatsachen dabei entscheidend sind.

4.1 Discount Faktor γ

Um den Discount Faktor γ zu verstehen, wird exemplarisch ein vereinfachter Fall unseres obigen Modells angenommen. Man gehe davon aus, das Environment bestehe aus 4 Zuständen mit den Indizes 0,1,2,3 und die erlaubten Aktionen sind das nach links Gehen (Aktion 0) sowie das nach rechts Gehen (Aktion 1). Dementsprechend hat Q die Form $(4) \times (2)$. Unser Agent startet jedes Mal in Zustand 0 und das Target ist der Zustand 3. Periodische Bedingungen liegen nicht vor, sodass die Entscheidung, einen Schritt nach links von Zustand 0 aus zu machen, direkt die Episode

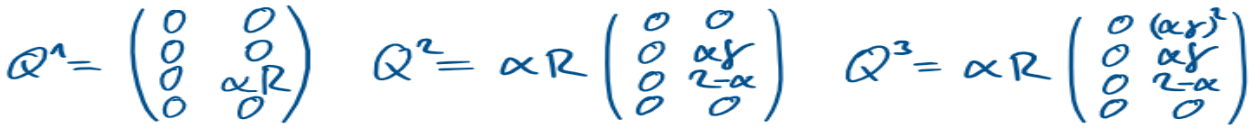


Abbildung 3: Matrix Q für drei Episoden mit Aktionsfolgen $(\rightarrow\rightarrow\rightarrow)$, $(\rightarrow\leftarrow\rightarrow\rightarrow\rightarrow)$ und $(\rightarrow)$. Die Matrix in der dritten Episode ist gezielt bereits nach der ersten Aktion ausgewertet, um die Fortpflanzung des Discount-Terms zu veranschaulichen. Dieser pflanzt sich nach jeder Episode mit jeweils einem weiteren Faktor $(\alpha \cdot \gamma)$ fort.

beendet. Ebenso endet die Episode, nachdem der Agent sich von Zustand 2 aus auf das Target zubewegt, da keine Aktion zum Verweilen vorliegt. Diese Aktion wird dann mit R belohnt. Hierfür wurde die Matrix für drei Episoden mit willkürlichen Aktionsfolgen ausgewertet, um die Funktion von γ erklären zu können: Anhand der Entwicklung in Abb. 3 kann man erkennen, wie der Agent mithilfe des Discount-Terms in Gl. (3) über benachbarte Zustände vom Target-Zustand lernt. In diesem Beispiel ist das Fortschreiten zum Ziel erfolgreich, also sollte das Fortschreiten zum Zustand neben dem Ziel auch in gewisser Weise erfolgreich sein. Demnach wird dies über den Discount-Term auch belohnt, indem man die Belohnung mit $\alpha \cdot \gamma$ skaliert, wobei $\alpha \cdot \gamma$ zwischen 0 und 1 liegt. Jede Epoche pflanzt sich dieser Wert mit erneuter Skalierung von $\alpha \cdot \gamma$ fort, also immer weniger Gewichtung, während der Eintrag, der belohnt wird, linear mit R wächst. Dadurch wird nach genügend Epochen eine Tendenz zum Ziel gebildet; der Agent lernt. Anhand dieser grundlegenden Funktionsweise lässt sich jedes Pfad-Finde-Problem verstehen und umsetzen. In diesem Modell kann man ebenfalls sehen, dass kein Lernprozess in die linke Richtung vorliegt, was darauf hinweisen sollte, dass bei vorliegenden periodischen Randbedingungen (bspw. Kugeloberflächen, o.Ä.) diese wichtig sind zu implementieren, um die bestmögliche Lernstrategie zu entwickeln. Denn hier wäre theoretisch das Erreichen des Ziels durch einen Schritt nach links am schnellsten. Allgemein setzt man den Discount Faktor auf Werte $0.5 \lesssim \gamma \lesssim 1$ und orientiert sich dabei an der Ähnlichkeit des erlernten Verhaltens zwischen den Zuständen.

4.2 Lernrate α

Bei der Untersuchung der Lernrate α stellt man fest, dass diese sich ähnlich zu einer finiten Zeitdifferenz Δt in der numerischen Lösung einer Differenzialgleichung verhält. Zu hohe Werte verhindern eine Konvergenz des Algorithmus, zu niedrige lassen den Algorithmus zu lange rechnen. Um α zu untersuchen, wurde die Diffusion in diesem Unterabschnitt entfernt. Eine sogenannte *Lernkurve* in Abb. 4 wurde bei gleichem Start-Zustand des Agents ausgewertet. Man erkennt einen exponentiellen Abfall sowie Konvergenz der Kurve gegen den kürzesten Weg. Für den Fall eines zufälligen Start-Zustandes lässt sich, wie zu erwarten, keine Konvergenz feststellen, wenn man die totale Anzahl an Schritten untersucht (siehe Abb. 5), denn das Target und damit auch die kleinstmögliche Schrittzahl variieren. Deshalb wurden (in Abb. 6) die Schrittzahlen jeder Episode in Abhängigkeit von der jeweils minimalen Schrittzahl geplottet und es wurde Konvergenz von der Lernkurve gegen den Wert 1 beobachtet. Dies bedeutet, dass der Agent in der Lage ist, das Target in beliebigem Zustand auf kürzestem Wege zu finden. Wenn man nun den Lern-Parameter sehr hoch setzt, wie in Abb. 6b gemacht wurde, kann man erkennen, dass der Algorithmus schlechter konvergiert, weil die Schwankungen höher sind und der Agent mehr Episoden braucht, bis ein konstantes Regime an Schritten erreicht wird.

Um α effizient anzupassen, kann man es, ähnlich zur Wahrscheinlichkeit ε, mit zunehmender Episodenanzahl abfallen lassen, um den Algorithmus nach gewisser Zeit auf eine Strategie zu verankern und diese zu perfektionieren.

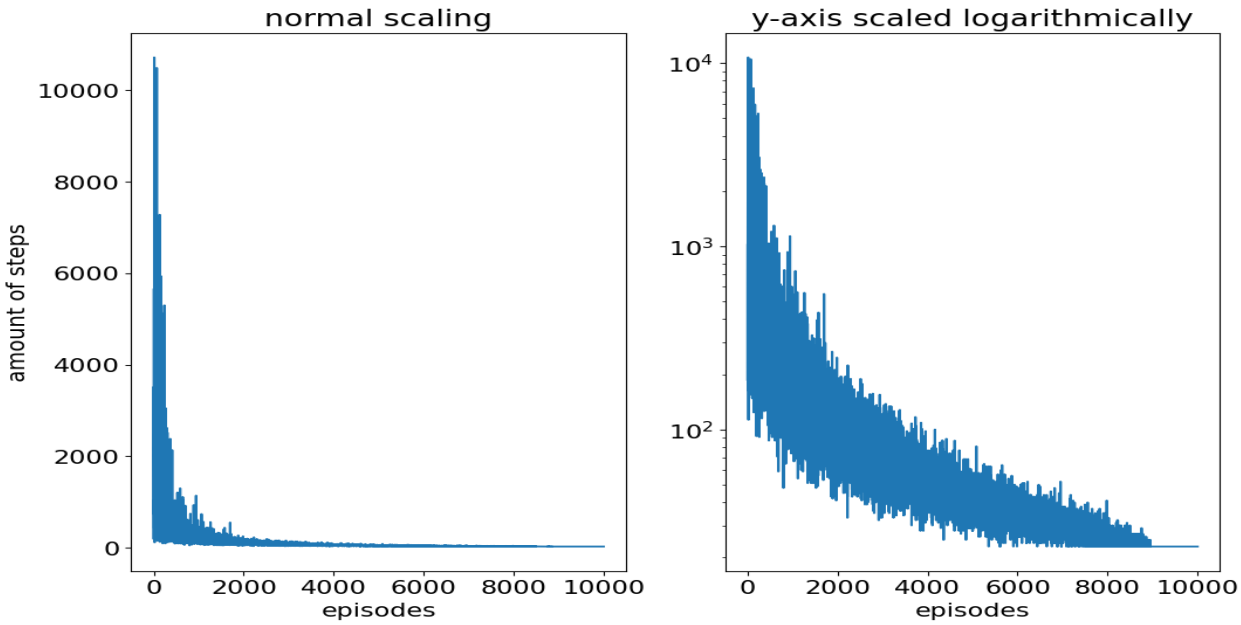


Abbildung 4: Lernkurve für den Fall $\alpha = 0.01$. Der Start-Zustand des Agents sowie der Target-Zustand sind für alle $N = 10^4$ Episoden konstant. Der Algorithmus konvergiert gegen die minimale Anzahl an Schritten (hier: 22). Mithilfe logarithmischer Skalierung der y-Achse (rechts) kann man einen ungefähr exponentiell abfallenden Zusammenhang zwischen Episodenanzahl und Schrittzahl feststellen.

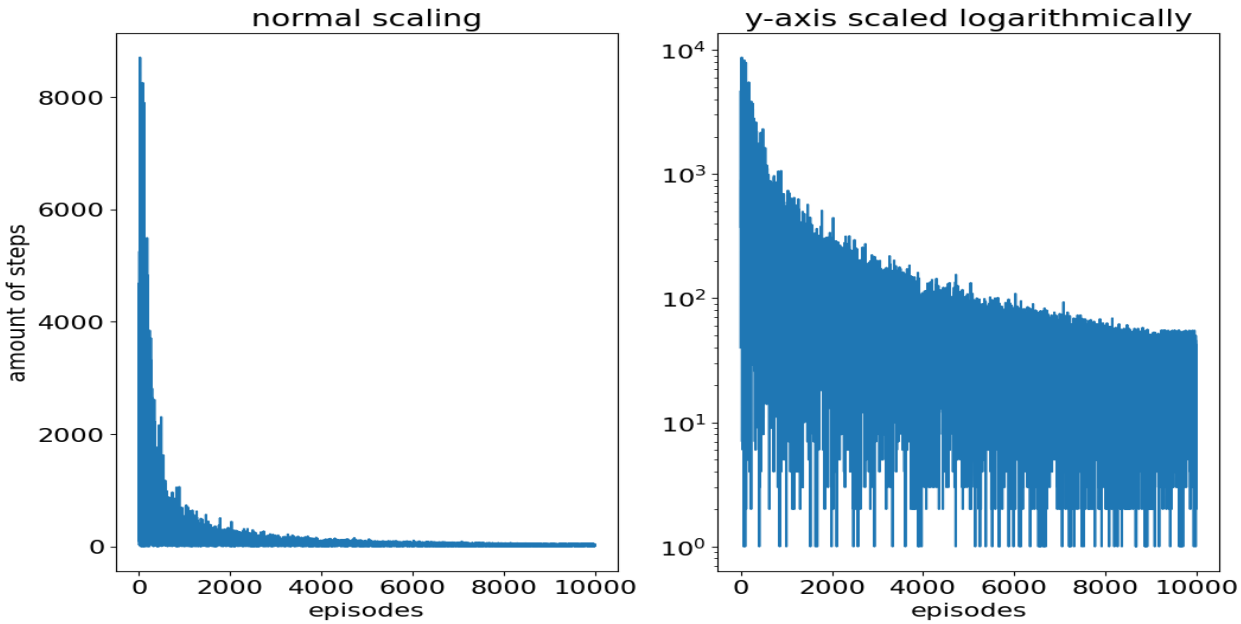
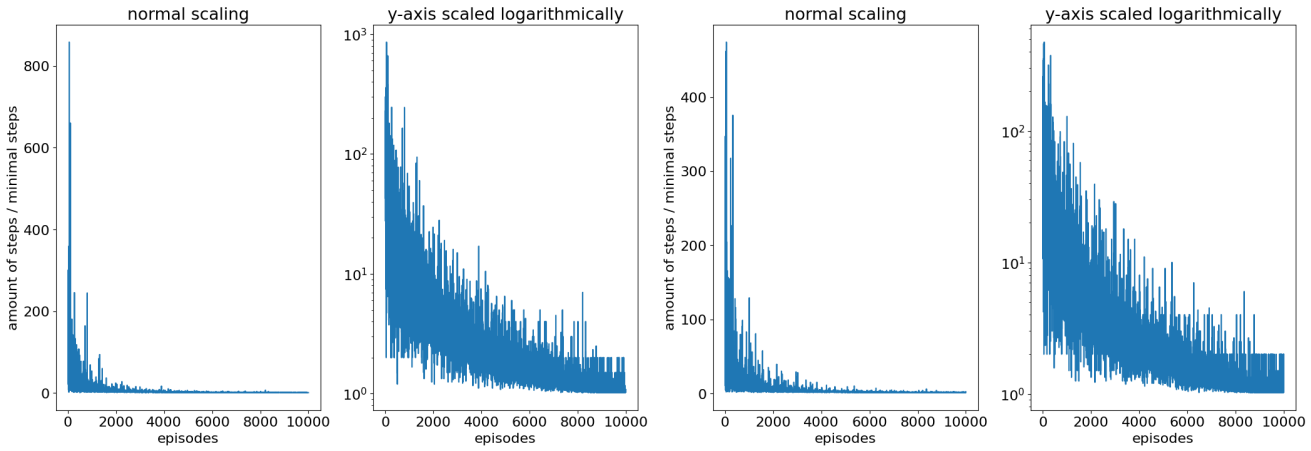


Abbildung 5: Lernkurve für zufälligen Start-Zustand des Agents ($\alpha = 0.01$, $N = 10^4$). Der Algorithmus konvergiert nicht mehr gegen konstante Anzahl an Schritten, da die kleinstmögliche Anzahl an Schritten zum Target bei jeder Episode unterschiedlich ist.

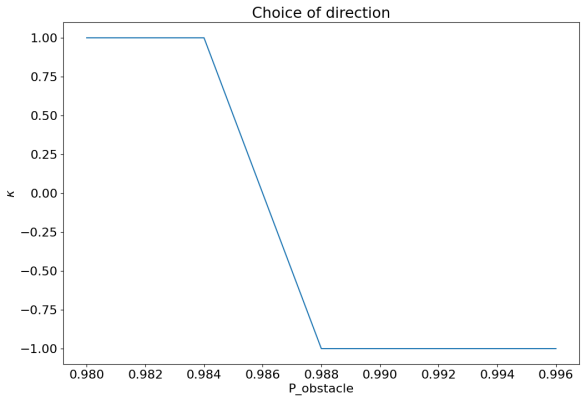


(a) $\alpha = 0.01$ (b) $\alpha = 0.999999$

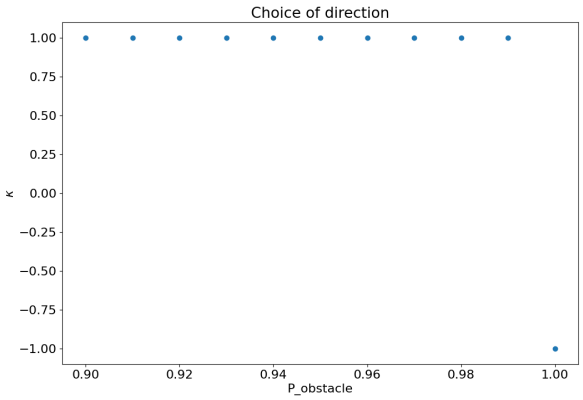
Abbildung 6: Zwei Lernkurven für zufällige Start-Zustände des Agents pro Episode ($N = 10^4$). Für jede Episode wurde vorerst die minimale Schrittzahl von Agent zu Target ermittelt und die totale Anzahl an Schritten damit skaliert. Daher konvergieren die Lernkurven gegen 1, wobei die Lernkurve in (b) aufgrund zu hoch gewählter Lernrate α schlechter konvergiert, als die Lernkurve in (a). Dies erkennt man an höherer Fluktuation sowie höherer Episodenanzahl, bei der das Schrittminimum erreicht wird.

5 Aufgabe 4 - Stochastischer Hindernis

Abschließend widmen wir uns einem Fall, bei dem in das bereits bekannte Environment ein stochastisches Hindernis eingebaut wird, das den Agent mit einer zuvor festgelegten Wahrscheinlichkeit p_{obstacle} davon abhält, dort entlangzugehen. Wir untersuchen, wie der Agent auf dieses Hindernis reagiert und wann er einen längeren Weg bevorzugt, mit dem er das Hindernis umgeht. Hierfür sind wieder Start- und Target-Zustand fest. Das Hindernis wurde implementiert, indem man in der kürzeren Richtung einen Bereich vor dem Target definierte, innerhalb dessen der Agent nach Ausführung seiner Aktion mit der Wahrscheinlichkeit p_{obstacle} um einen Zustand weiter weg vom Target verschoben wird. Um beurteilen zu können, zu welcher Richtung der Agent in Abhängigkeit von p_{obstacle} tendiert, wurde der Index $\kappa \in \{-1, 1\}$ definiert. Ist $\kappa = -1$, geht der Agent links herum, für $\kappa = 1$ geht er rechts herum. In dem kommenden Beispiel liegt der kürzere Weg rechts entlang und der längere links entlang. Dabei wird der kürzere Weg mit unserem Hindernis ausgestattet, welches drei Felder lang ist. Ein Richtungswechsel in Richtung zum längeren Weg wird dann erwartet, wenn der Mittelwert an Schritten bis zum Target in Richtung des Hindernisses gleich den Schritten bis zum Target in Richtung des längeren Weges ist. Die zugehörige Wahrscheinlichkeit p_{obstacle} wurde (in Abb. 7) numerisch angenähert. In Abb. 7b wurde für zu hohes α, bei dem der Algorithmus schlechter konvergiert, gezeigt, dass der Agent sich erst bei höherem p_{obstacle} dazu entscheidet, den längeren Weg zu gehen. In bildlicher Analogie könnte man sich einen Hund vorstellen, der verzweifelt versucht, an einen Knochen zu kommen, indem er ständig gegen einen Zaun knallt, obwohl er durch das Zauntor nebenan an den Knochen gelangen könnte.



(a) $\alpha = 0.01$



(b) $\alpha = 0.999999$

Abbildung 7: Plots für den Richtungswechsel, repräsentiert durch κ in Abhängigkeit von p_{obstacle}. (a): Der Agent ändert die Strategie auf die längere Strecke ohne Hindernis bei einer Wahrscheinlichkeit $p_{\text{obstacle}} \approx 0.986$. Dies ist der Fall, wenn der Mittelwelt an Schritten bis zum Target in Richtungen des Hindernisses etwa gleich denen in Richtung des längeren Weges ist. (b): Verlauf analog zu (a) für ein zu hoch gewähltes α. Der Scatter-Plot zeigt, dass die Richtungsänderung nicht allzu sinnvoll ist, denn nun ist p_{obstacle} für den Richtungswechsel höher, als für kleineres α in (a). (In (a) war κ bereits für 0.988 negativ und nun ist es bei 0.99 immer noch positiv.)

6 Schlussteil

Abschließend lässt sich sagen, dass der Q-Learning Algorithmus ein effektives Werkzeug ist, um intelligente Mechanismen jeglicher Art zu kreieren und zu optimieren. Jedoch ist der Erfolg und die Effizienz dieser Algorithmen sehr abhängig von den Parametern, die man (leider) selbstständig festzulegen hat.