Reinforcement learning für intelligente Brownsche Dynamik

Bearbeitung von Luis Gindorf und Daniel Müller

Antestat am 02.03.23

Inhaltsverzeichnis

1 Einleitung

In diesem theoretischen Versuch im Rahmen des Fortgeschrittenenpraktikums sollen die Grundlagen des Reinforcement Learning anhand des *Q-learning* Algorithmus erlernt werden. Das zu untersuchende Problem kann dabei als *Random Walk* modelliert werden. Dieses Modell ist besonders relevant für typische Probleme im Bezug auf Diffusion im Bereich der *weichen Materie*.
Zunächst wird in Aufgabe 1 ein Agent implementiert, der sich in diskreten Schritten zufällig bewegt. Dessen Verhalten wird statistisch untersucht, wobei insbesondere das *Mean Squared Displacement* (MSD) betrachtet wird, das mit der Diffusionskonstante zusammenhängt. Außerdem wird ein Weg gefunden, diese Diffusionskonstante als Parameter des Agenten anzupassen.
In Aufgabe 2 wird der Lernalgorithmus implementiert und dessen Strategie validiert, in der anschließenden Aufgabe 3 werden die Hyperparameter des Algorithmus untersucht und optimiert.
Abschließend wird in Aufgabe 4 ein stochastisches Hindernis implementiert, das der Algorithmus zu Umgehen lernen soll.

2 Theorie

Zur Untersuchung der Diffusion eines Teilchens (im physikalischen Kontext häufig als *Schwimmer* bezeichnet) werden zunächst sowohl Raum als auch Zeit diskretisiert, um die informatische Modellierung realisieren zu können. Die Position kann dadurch zu einer endlichen Anzahl an Zuständen abstrahiert werden, während die Bewegung durch Aktionen dargestellt wird, die den Übergang zwischen den Zuständen in den einzelnen Zeitschritten beschreiben. Es gilt also: Eine Aktion j beschreibt den Übergang von einem Ausgangszustand vor dem Zeitschritt i in einen durch die Aktion eindeutig festgelegten nächsten Zustand i'. Im Q-Learning-Algorithmus wird nun eine namensgebende Matrix Q definiert, die zu jedem Indextupel (i, j) angibt, wie erstrebenswert die Akton j im Zustand i ist, um ein fest definiertes Ziel zu erreichen. Nach dieser Matrix kann dann für jeden Zustand die optimale Aktion als $\operatorname{argmax}_j(Q_{ij})$ gewählt werden. Die Hauptaufgabe des Algorithmus ist es nun, geeignete Werte für die Einträge von Q zu finden, sodass die entsprechende Strategie zum Erfolg führt. Dies geschieht mit einer Update-Gleichung, die nach jeder ausgeführten Aktion die Werte in Q

anpasst:

$$Q_{ij}^{\text{neu}} = Q_{ij}^{\text{alt}} + \alpha(R + \gamma \max_k(Q_{i'k}) - Q_{ij}^{\text{alt}}) \tag{1}$$

Dabei bezeichnet α die *learning rate*, R die Belohnung für das Durchführen der Aktion und γ den sogenannten *discout factor*, der beschreibt, inwieweit mögliche gute Entscheidungen in den nächsten Schritten den aktuellen Schritt beeinflussen. Der Einfluss von α und γ wird in Aufgabe 3 näher untersucht.

Im einfachsten Fall, der in Aufgabe 1 betrachtet wird, bewegt sich der Agent zufällig in einem beschränkten Intervall in einer Dimension, das heißt, es gibt Zustände $[1, N_{states}]$ und mit einer Wahrscheinlichkeit $p = 1/2$ macht der Agent einen Schritt nach rechts und mit der Gegenwahrscheinlichkeit $q = 1 - p = 1/2$ einen Schritt nach links. Die Verschiebung m nach N Schritten ausgehend von der Startposition, die festgelegt ist durch die Anzahl der Schritte nach rechts und links, kann über die Binomialverteilung ausgedrückt werden als

$$P_N(m) = \frac{N!}{(m+N)/2!(N-(m+N)/2)!} 2^{-N} \tag{2}$$

Im Übergang zum Kontinuum $m \to x, N \to t$ ergibt sich daraus die Gauß-Verteilung:

$$P(x,t) = \frac{1}{\sqrt{4Dt}} \exp\left(-\frac{x^2}{4Dt}\right) \tag{3}$$

mit der Diffusionskonstanten $D = \frac{a^2}{2\tau}$. Durch die Diskretisierung und konkrete Implementierung sind hier $a = |\Delta x| = 1$ und $\tau = 1$ festgelegt.

3 Aufgabe 1

Durch die Setzung $a = \tau = 1$ ist die Diffusionskonstante auf den Wert $D = 1/2$ festgelegt. Unser Ziel ist es, eine effektive Diffusionskonstante einzuführen, die gesteuert werden kann. Dazu verfolgen wir den Ansatz, dass sich der Agent in jedem Zeitschritt nur mit einer gewissen Wahrscheinlichkeit bewegt, sodass die Diffusion effektiv verlangsamt wird. Bewegt er sich beispielsweise mit einer Wahrscheinlichkeit $P_{diffstep} = 0.5$, also im Schnitt in jedem zweiten Zeitschritt, wird der effektive Zeitschritt verdoppelt und damit die Diffusionskonstante halbiert. Im Fall $P_{diffstep} = 1$, wo also in jedem Zeitschritt tatsächlich ein Schritt gemacht wird, beträgt die Diffusionskonstante wie oben berechnet den Wert 1/2. Größere Werte sind mit dieser Methode nicht möglich, da dafür dann die Bewegungswahrscheinlichkeit größer als eins sein müsste. Dies ließe sich nur durch größere Schrittweiten erreichen.

Mit dieser Implementierung wurden drei verschiedene effektive Diffusionskonstanten untersucht. Dabei wurden jeweils $N_{episodes} = 20.000$ Random Walks simuliert und darüber

gemittelt das Mean Squared Displacement (MSE) in Abhängigkeit der diskretisierten Zeit bestimmt. Aus den entsprechenden Plots, die im Folgenden dargestellt sind, lässt sich die effektive Diffusionskonstante bestimmen, da für die Normalverteilung $\langle x^2 \rangle = 2Dt$ gilt. Die Graphen sind für großes $N_{episodes}$ also linear, wobei die Steigung das Doppelte der Diffusionskonstante beträgt. So ließen sich folgende Werte bestimmen:

$D_{Vorgabe}$	0.01	0.05	0.45
$D_{Messung}$	0.01022	0.04931	0.44643

Tabelle 1: Eingestellte und in Simulation bestimmte Diffusionskonstanten

Wie man leicht erkennen kann, entsprechen die gemessenen Werte relativ gut den Vorgegebenen. Die Abweichungen sind durch den inhärenten Zufall in den einzelnen Simulationen zu erklären. Für $N_{epsiodes} \to \infty$ ist zu erwarten, dass $D_{Messung}$ gegen $D_{Vorgabe}$ konvergiert.

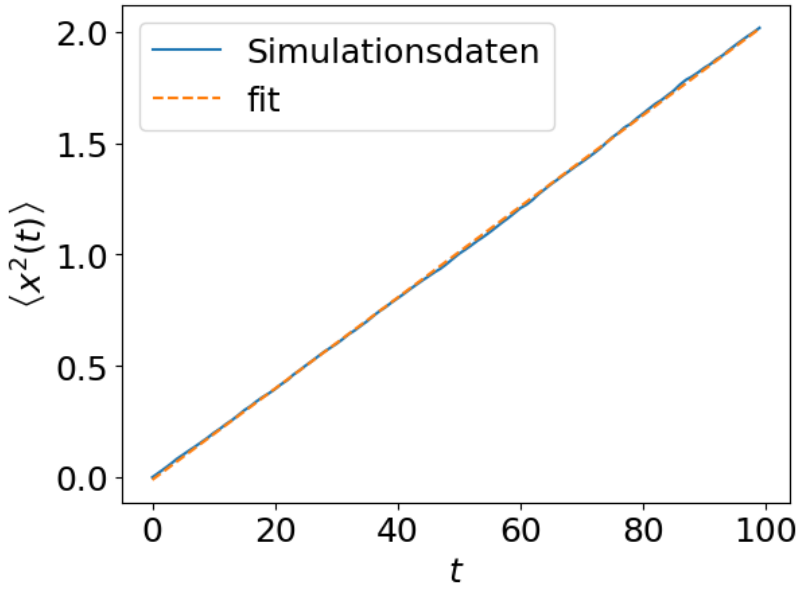


Abbildung 1: Simulation des MSDs mit $D_{Vorgabe} = 0.01$

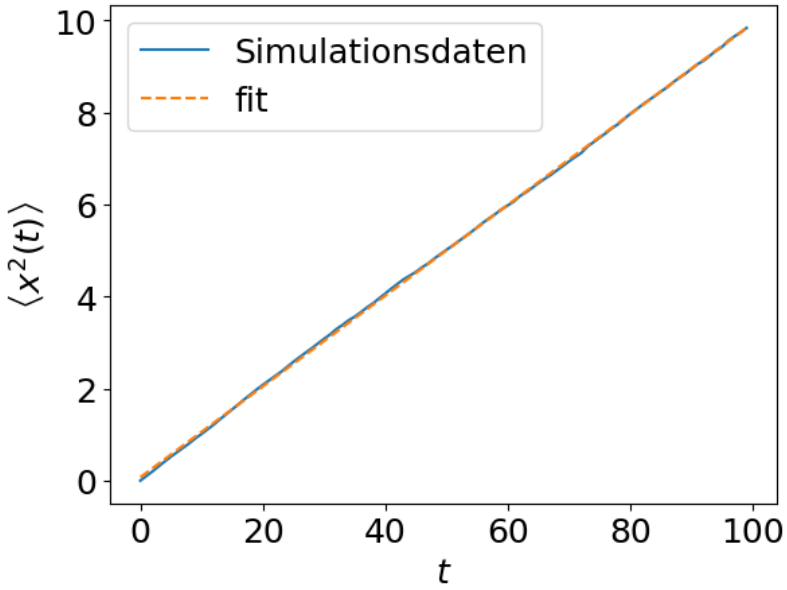


Abbildung 2: Simulation des MSDs mit $D_{Vorgabe} = 0.05$

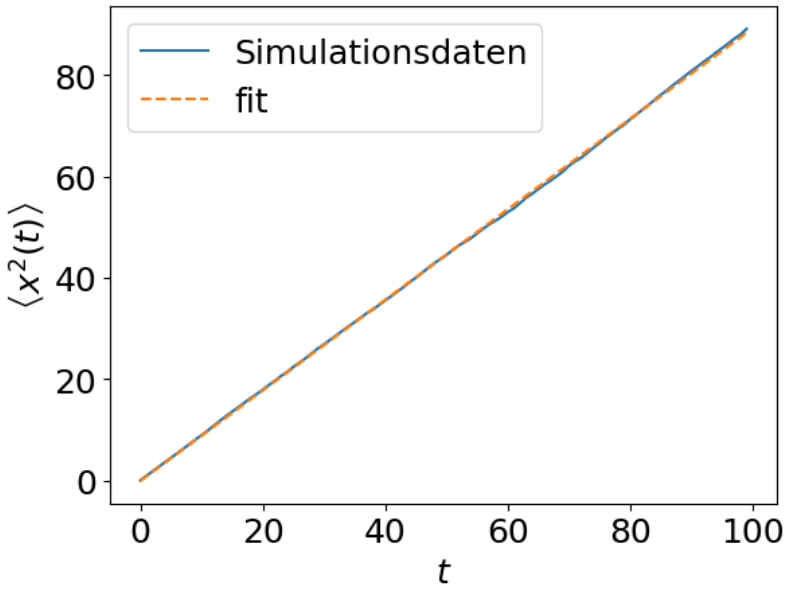


Abbildung 3: Simulation des MSDs mit $D_{Vorgabe} = 0.045$

4 Aufgabe 2

Zunächst beschreiben wir hier kurz die Funktionsweisen der einzelnen implementierten Methoden:*adjust_epsilon* dekrementiert mit jedem Aufruf um einen konstanten Wert, sodass er nach dem Aufruf zero_fraction × N_episodes auf 0 abgefallen ist. Weiter kann er nicht sinken. In dieser Implementation ist es nicht nötig, die aktuelle Episode zu übergeben. erstellt ein Array der möglichen Aktionen, je nachdem ob der Algorithmus zufällig entscheidet oder anhand von Q. Aus diesem Array wird in beiden Fällen ein Element ausgewählt. Alternative Implementierung wären möglicherweise effizienter, aber das spielt bei einer maximalen Auswahl aus drei Aktionen keine Rolle und die gewählte Umsetzung ist unserer Ansicht nach schön übersichtlich. *perform_action* führt die Aktion dann aus, indem x um den Index der Aktion minus Eins inkrementiert wird. Die periodische Randbedingung wird mittels des modulo-Operators umgesetzt. In der Methode *update_Q* werden zunächst die Zustände, die gewählte Aktion und die Belohnung als lokale Variablen mit den Bezeichnungen aus Gleichung 1 bestimmt, um die abschließende Berechnung des neuen Q_{ij} zu vereinfachen.
Aufgrund des *random_step*, der zwischen *choose_action* und *perform_action* ausgeführt wird, gilt Folgendes im Allgemeinen **nicht**:
Die Anwendung von Aktion j auf Zustand i führt zu Zustand j'. Die Belohnung erfolgt also nicht immer aufgrund der tatsächlich gewählten Aktion, was eine direkte Folge der Diffusion ist, die zusätzlich zum Lernprozess stattfindet.
Die folgende Animation des GIFs namens *LAST_TRAJECTORY_23-03-08T19_53_09.gif* zeigt die Trajektorie der letzten Lernepisode, also dem besten gefundenen Weg. Augenscheinlich entspricht dies der optimalen Lösung. An der Matrix kann man dies bestätigen, da dort zu erkennen ist, dass für der Zielzustand $x = 8$ die Aktion $\downarrow$ am besten ist, während die Zustände links davon die Aktion $\rightarrow$ bevorzugen und andersherum. Zur korrekten Anzeige muss ein animationsfähiger PDF-Viewer benutzt werden (Windows: Adobe Reader, Linux: Okular).

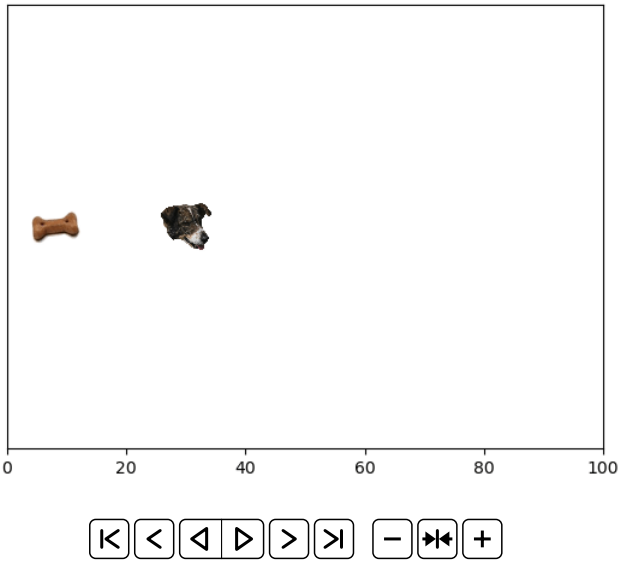


Abbildung 4: GIF mit Name *LAST_TRAJECTORY_23-03-08T19_53_09.gif*.

Teilweise kann man beobachten, dass der Agent durch die Diffusion mehrmals vom Ziel heruntergeschoben wird. Da dieser Effekt für größere Diffusionskonstanten stärker ist, haben wir dies einmal im GIF *LAST_TRAJECTORY_23-03-08T19_58_51.gif* für $D = 1/4$ festgehalten.

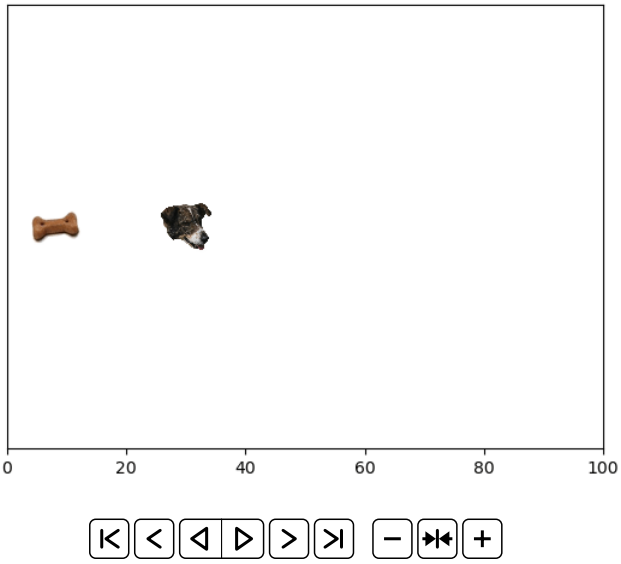


Abbildung 5: GIF mit Name *LAST_TRAJECTORY_23-03-08T19_58_51.gif*.

An den vielen Nulleinträgen der Q-Matrix lässt sich erkennen, dass viele Zustände scheinbar gar nicht erreicht werden. Aus Interesse und um zu überprüfen, ob die periodische Randbedingung klappt, haben wir einmal den Startwert $x = 95$ ausprobiert. Hier hat der Agent trotzdem den (jetzt längeren) Weg nach links gewählt. Erst nach einer Erhöhung von *N_episodes* auf 10^5 wählt der Agent den Weg mit Intervallsprung, wie in *LAST_TRAJECTORY_23-03-08T20_07_38.gif* zu erkennen ist.

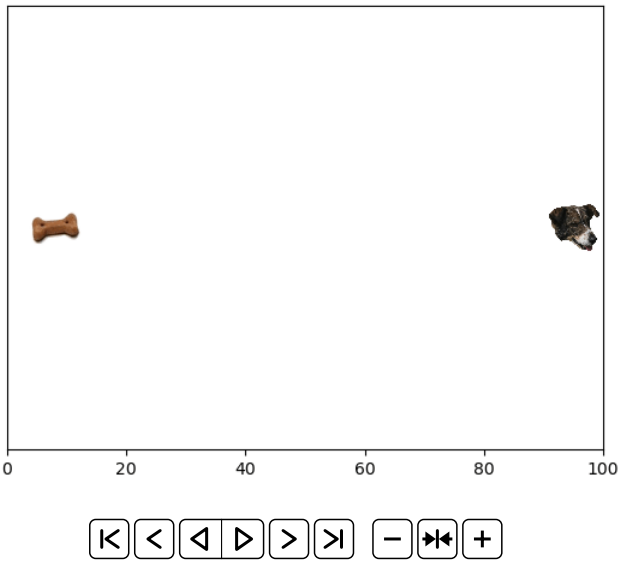


Abbildung 6: GIF mit Name *LAST_TRAJECTORY_23-03-08T20_07_38.gif*.

5 Aufgabe 3

5.1 Discount factor γ

Wir betrachten ein vereinfachtes Modell mit eindimensionalem Bewegungsraum auf vier Zuständen $[0, 1, 2, 3]$ und zwei möglichen Aktionen: Nach links 0 und nach rechts 1. Der Start ist in Zustand 0 und das Ziel erfüllt durch Erreichen des Zustands 3. Die Q-Matrix hat eine Zeile für jeden Zustand und eine Spalte für jede Aktion. Sie wird also initialisiert als $np.zeros((4, 2))$. In der ersten Episode sind alle $Q^{neu} = 0$ unabhänig von der Aktionsfolge, da alle $Q^{alt} = 0$ sind. Die Ausnahme bildet Q_{21}^{neu}, da hier das Ziel erreicht wird und somit die Belohnung $R \neq 0$ ist. Hier gilt also entsprechend $Q_{21}^{neu} = \alpha R$. In der zweiten Episode setzen wir als Aktionsreihenfolge $\rightarrow \leftarrow \rightarrow \rightarrow \rightarrow$ voraus. Die Q-Updates sehen folgendermaßen aus:

$$
\begin{aligned}
Q_{01}^{neu} &= Q_{01}^{alt} + \alpha(0 + \gamma \underset{k}{max}(Q_{1k}^{alt}) - Q_{01}^{alt}) = 0 \\
Q_{10}^{neu} &= Q_{10}^{alt} + \alpha(0 + \gamma \underset{k}{max}(Q_{0k}^{alt}) - Q_{10}^{alt}) = 0 \\
Q_{01}^{neu} &= Q_{01}^{alt} + \alpha(0 + \gamma \underset{k}{max}(Q_{1k}^{alt}) - Q_{01}^{alt}) = 0 \\
Q_{11}^{neu} &= Q_{11}^{alt} + \alpha(0 + \gamma \underset{k}{max}(Q_{2k}^{alt}) - Q_{11}^{alt}) = \alpha^2 \gamma R \\
Q_{21}^{neu} &= Q_{21}^{alt} + \alpha(R + \gamma \underset{k}{max}(Q_{3k}^{alt}) - Q_{21}^{alt}) = (2 - \alpha)\alpha R
\end{aligned} \tag{4}
$$

Wird anschließend in Episode 3 zunächst ein Schritt nach rechts gemacht, updated sich noch Q_{01} gemäß

$$
Q_{01}^{neu} = Q_{01}^{alt} + \alpha(0 + \gamma \underset{k}{max}(Q_{1k}^{alt}) - Q_{01}^{alt}) = \alpha^3 \gamma^2 R
$$

Man kann also erkennen, dass sich in diesem Setup der Reward αR in jeder Lernepisode um einen Zustand in Richtung Start fortpflanzt und dabei jeweils mit einem Faktor $\alpha\gamma$ versehen wird. Nach der dritten Episode wird also schon $Q_{01} \neq 0$ sein, sodass gilt: $\underset{k}{argmax}(Q_{0k}) = 1$. Der Algorithmus hat also bereits gelernt, im ersten Schritt nach rechts zu gehen. Auch in den nächsten Schritten wird er immer nach rechts gehen, er hat also bereits die richtige Lösung gefunden und wird diese weiter verstärken. Selbst wenn er viele zufällige Schritte nach links macht, wenn $\epsilon < 1$ ist, macht er mindestens gleich viele nach rechts in jedem Zustand, um zum Ziel zu gelangen, sodass die Bewegungen nach rechts immer stärker belohnt werden. Der Algorithmus wird also nie lernen, nach links zu laufen.

5.2 Learning Rate α

Wir setzen für diese Aufgabe die Diffusion außer Kraft ($D = 0$). In jeder Episode eines Lernprozesses berechnen wir die Anzahl Schritte, die notwendig ist, um vom randomisierten Start das Ziel zu erreichen und darauf zu verweilen mit der Funktion *minimal_steps_needed*. Außerdem bestimmen wir die Zahl der tatsächlich benötigten Schritte. Das Verhältnis dieser beiden Größen aufgetragen über die aktuelle Episode ergibt die sogenannte *learning curve*. Für $N_{\text{Episodes}} = 10000$ und $\alpha = 0.999999$ erhalten wir einen recht normalen, erfolgreichen Lernverlauf:

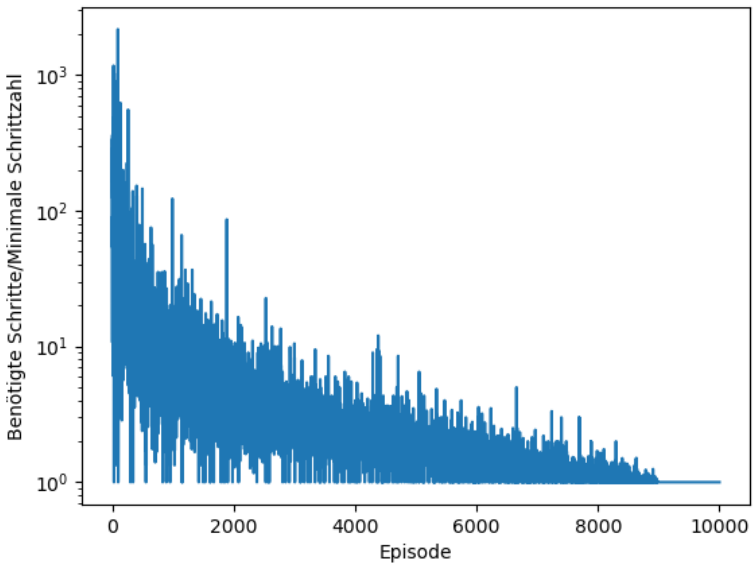


Abbildung 7: Lernkurve für normale Simulation

Das Verhältnis aus theoretisch und tatsächlich benötigten Schritten nähert sich im Mittel schnell an 1 an, wobei die Varianz im Lernverlauf immer weiter abnimmt, weil weniger zufällige Schritte stattfinden. Ab dem Punkt *episode >= agent.zero_fraction* $\times$ *N_episodes* ist $\epsilon = 0$, also verfolgt der Agent die erlernte Strategie und macht keinen Schritt mehr als benötigt, was zeigt, dass die Aufgabe gemeistert wurde. Man kann also sehen, dass für solch eine einfache Aufgabe selbst eine *learning rate* von fast 1 zur Konvergenz führt.
Um den Einfluss der *learning rate* sichtbar zu machen, verringern wir also die Zahl der Episoden auf 800. Hier erhalten wir beispielsweise für $\alpha = 0.4$ Konvergenz:

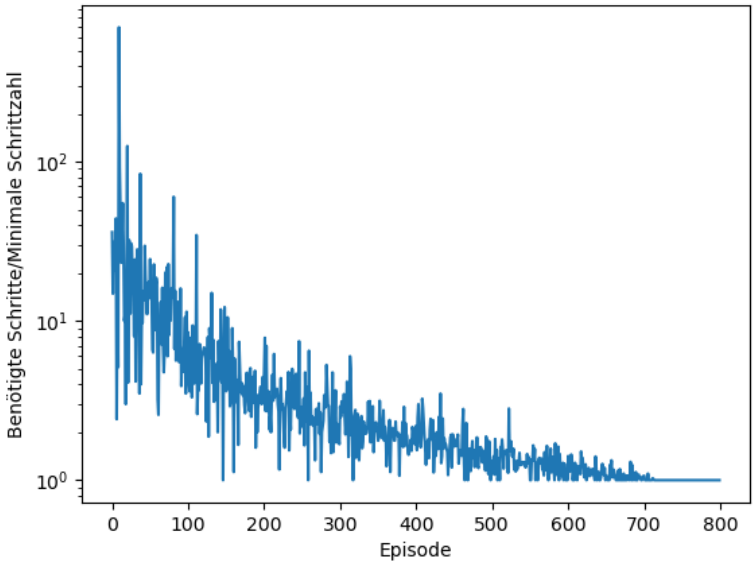


Abbildung 8: Lernkurve für wenige Episoden und normale Lernrate

und für $\alpha = 0.999999$ keine Konvergenz:

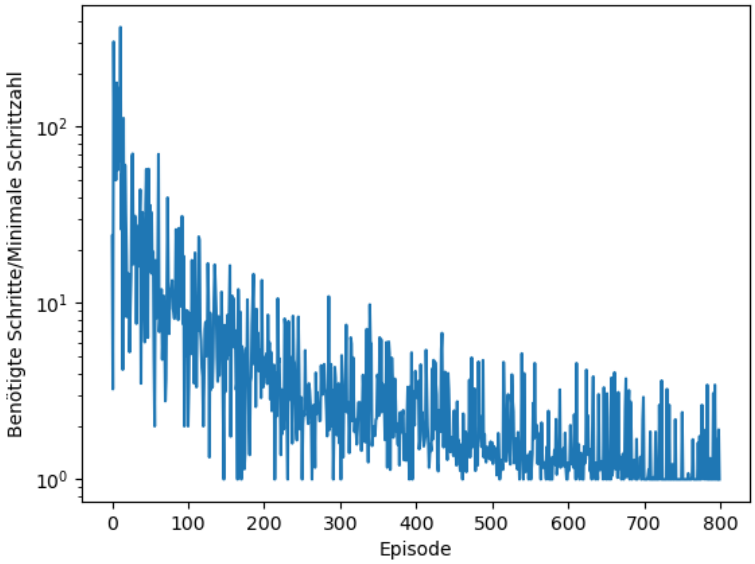


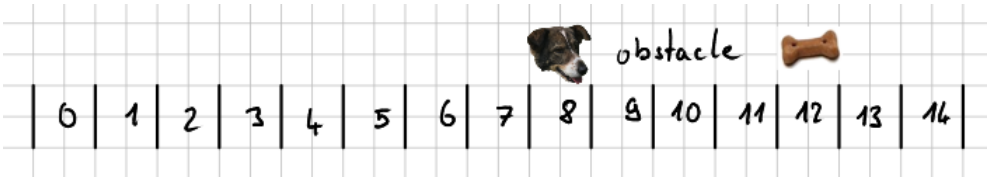
Abbildung 9: Lernkurve für wenige Episoden und hohe Lernrate

Man kann also sehen, dass die hohe *learning rate* hier die Konvergenz verhindert. Dazu muss

jedoch gesagt werden, dass durch die randomisierte Natur des Algorithmus auch einzelne Durchläufe mit hohem α konvergiert und mit niedrigem α nicht konvergiert sind.

6 Aufgabe 4

Wir setzen nun die Parameter zurück auf $D = 0.125$, $N_{\text{states}} = 15$, $\text{target}_{\text{position}} = 12$, $\text{starting}_{\text{position}} = 8$, $N_{\text{epsidoes}} = 1000$, $\alpha = 0.4$, $\gamma = 0.9$ und $\text{obstacle}_{\text{interval}} = \text{np.arange}(9, 12)$. Die Situation für den Learner ist also wie folgt:



Für die nun folgende theoretische Betrachtung vernachlässigen wir die Diffusion. Ist das Hindernis nicht vorhanden, ist die Situation einfach:
Auf direktem Weg benötigt Lasse nur 4 Schritte zum Erreichen des Ziels, auf dem indirekten Weg 11 Schritte, also 7 mehr. Für Lasse ist es also genau dann günstig, zu probieren, das Hindernis zu überwinden, wenn er erwarten kann, dass er dazu weniger als $\Delta = 7$ Extra-Schritte machen muss. Das Hindernis hat eine Länge von 3 und auf jedem Feld wird Lasse mit einer Wahrscheinlichkeit P_{obstacle} ein Feld zurückgeworfen, was einen Extraschritt für ihn bedeutet. Wir betrachten zunächst die Hindernislänge $l = 1$ und suchen die Wahrscheinlichkeit p^* für P_{obstacle}, sodass im Mittel genau Δ Extraschritte benötigt werden. Die Bedingung ist also, dass der Erwartungswert für das Weiterkommen (also gerade nicht zurückgeworfen zu werden) 1 ist. Entsprechend der Binomialverteilung für dieses Ereignis mit Wahrscheinlichkeit $1 - p^*$ soll also gelten: $\Delta(1 - p^*) = 1$. Umgestellt ergibt sich $p^* = 1 - 1/\Delta$. Wir können nun das Hindernis der Länge l als l unabhängige Einzelhindernisse beschreiben, wobei nun bei jedem Hindernis im Schnitt nur $\Delta' = \Delta/l$ Extraschritte gemacht werden dürfen. Abschließend ergibt sich also $p^* = 1 - l/\Delta = 1 - 3/7 = 4/7 \approx 0.571$
Diesen Wert haben wir anschließend versucht, numerisch zu bestimmen, indem wir die Simulation für unterschiedliche Werte von P_{obstacle} durchgeführt und die Laufrichtung bestimmt haben. Abweichend von der Versuchsanleitung haben wir zusätzlich jede Simulation $n = 11$ mal laufen lassen und die entsprechenden Werte $\kappa \in \{-1, +1\}$ aufaddiert, sodass wir für jeden Wahrscheinlichkeitswert einen ganzzahligen Wert in $[-n; n]$ erhalten, der angibt, wie häufig welcher Weg gewählt wird.

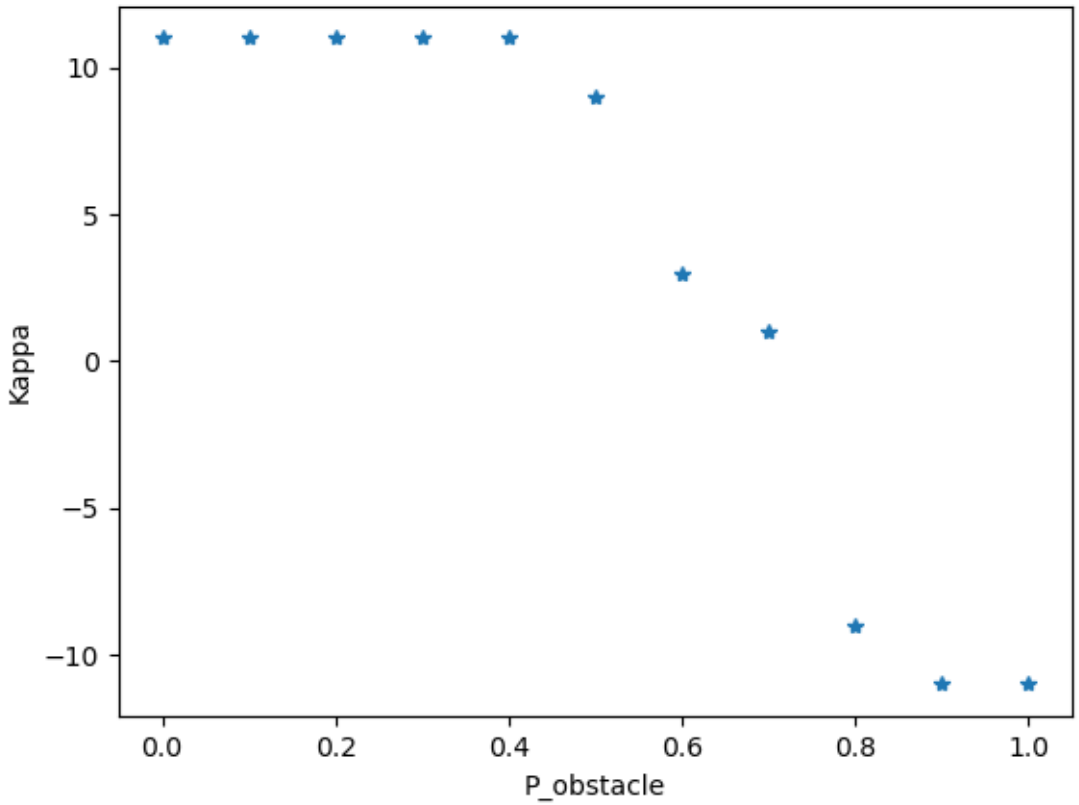


Abbildung 10: Kumulierte Laufrichtung in Abhängigkeit der Hindernis-Stärke

Wie zu erwarten erhalten wir für kleine P_{obstacle} den maximalen Wert für $\sum \kappa$ (das heißt, alle Agenten lernen, den direkten Weg zu laufen) und für große P_{obstacle} genau den gegenteiligen Fall, da das Hindernis dann nahezu unüberwindbar wird. Der Übergang zwischen den Extremfällen ist in diesem Plot mit 11 Datenpunkten auf $[0; 1]$ nicht genau zu bestimmen, sodass wir einen weiteren Durchlauf auf $[0.5; 0.7]$ mit 21 Punkten durchgeführt haben, der im folgenden Diagramm zu sehen ist:

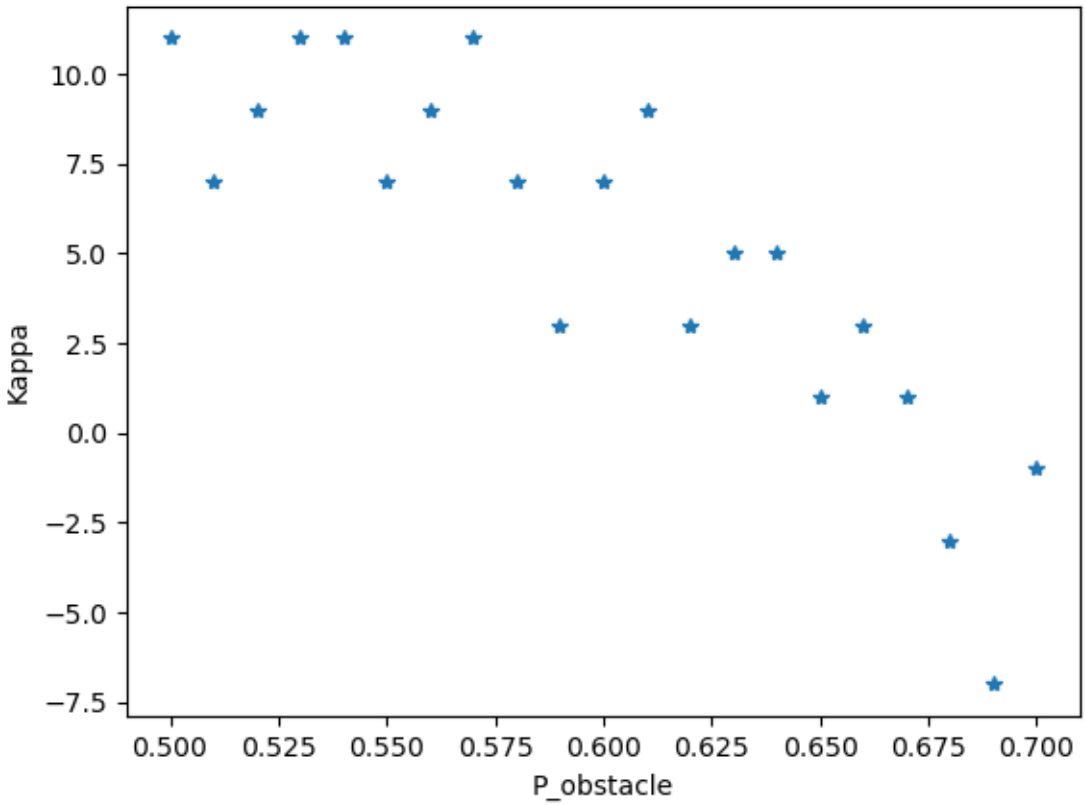


Abbildung 11: Kumulierte Laufrichtung in Abhängigkeit der Hindernis-Stärke im Bereich des Richtungswechsels

Hier fluktuieren die Werte etwas, da die Wahrscheinlichkeit in dem Bereich ist, wo der Unterschied zwischen den Wegen nur geringfügig ist. Der Vorteil der Addition mehrerer Durchläufe für jeden Datenpunkt ist, dass diese Fluktuation hier dargestellt werden kann und nicht dazu führt, dass die Werte direkt zwischen den beiden Extremen wechseln.
Den Wert für p^* kann man hier immer noch nicht genau ablesen, er liegt prinzipiell beim Nulldurchgang der dargestellten Daten, also circa bei 0.650 bis 0.675. Die Abweichung vom theoretischen Wert kann durch statistische Abweichungen, aber vor allem auch durch den Einfluss der Diffusion erklärt werden, die wir in der theoretischen Rechnung vernachlässigt hatten. Schließlich ist ein zufälliger Schritt in Richtung des Ziels hilfreicher, wenn dabei ein Stück des Hindernisses übersprungen wird.

Abschließend schauen wir uns noch das Verhalten des Algorithmus in Bezug auf das Hindernis an, wenn das α wieder sehr groß ist.

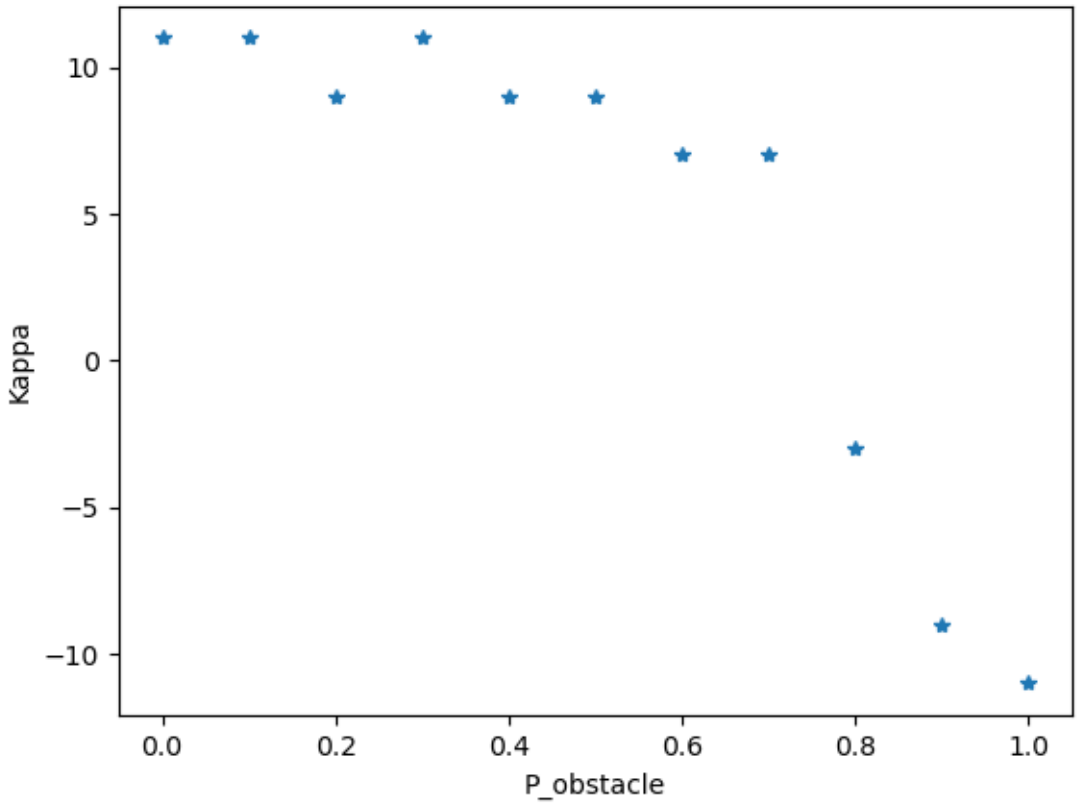


Abbildung 12: Kumulierte Laufrichtung in Abhängigkeit der Hindernis-Stärke für hohe Lernrate

Obwohl hier $N_{\text{episodes}} = 700$ und $\alpha = 0.99999999$ gesetzt wurde, ist der qualitative Verlauf des Graphen immer noch ähnlich wie zuvor. Für kleine p wählt er den direkten Weg, für große p den indirekten. Man kann jedoch auch Unterschiede beobachten:
Für $p = 0.2$ und $p = 0.4$ gibt es jeweils einen von 11 Durchläufen, in denen der Algorithmus gegen die eigentlich langsamere Strategie konvergiert. Außerdem wählt er für $p = 0.7$ immer noch 9 von 11 mal den direkten Weg, was der Algorithmus für das normale α nur noch 6 mal tat. Insgesamt scheint die Konvergenz also schlechter zu sein, auch wenn das Verhalten nicht so schlecht wie erwartet ist.

7 Fazit

Im Rahmen dieses FP-Versuchs haben wir uns das *Q-Learning* als gängigen *Reinforcement Learning*-Algorithmus angeschaut. Dazu haben wir den Algorithmus zunächst für eine konkrete Situation implementiert, uns dann die Hyperparameter angeschaut und anschließend das zu lösende Problem noch durch ein stochastisches Hindernis erweitert. Es hat sich gezeigt, dass der Parameter γ angibt, wie sehr der Lerneffekt durch die Zustände propagiert und immer an das konkrete Problem angepasst werden muss. Die Lernrate α spiegelt die Konvergenzgeschwindigkeit wieder; zu große Werte können jedoch die Konvergenz wiederum verhindern. Am Beispiel des Hindernisses habe wir gesehen, dass der Algorithmus auch erfolgreich auf Situationen anzuwenden ist, die analytisch schon relativ komplex zu untersuchen sind. Wir haben also gelernt, dass er ein geeignetes Werkzeug ist, komplexe physikalische Sachverhalte numerisch zu lösen. Nebenbei haben wir uns außerdem mit dem *Random Walk* als Modell der Diffusion und der objektorientierten Programmierung in Python beschäftigt.