Reinforcement learning für intelligente Brownsche Dynamik

Dawid Al-Bazaz
Heinrich-Heine University Düsseldorf

28. Juli 2022

Inhaltsverzeichnis

1 Einführung **3**

2 Theoretische Grundlagen **3**

2.1 Brownische Dynamik 3

2.2 Q Learning 4

3 Durchführung **5**

4 Bearbeitung und Auswertung **6**

4.1 Mittleres Verschiebungsquadrat 6

4.2 Implementation des Lernalgorithmus 6

4.3 Wahl der Hyperparameter 8

4.3.1 Discount factor 8

4.3.2 Learning rate 9

4.4 Stochastisches Hindernis 11

5 Fazit **12**

1 Einführung

Gegenstand dieses Versuchs ist die Anwendung des Q-Learning Algorithmus u.a. an einem physikalischen Kontext. Das Q-Learning im Allgemeinen stellt die Grundlage von einer Vielzahl von Lernalgorithmen dar, die numerische Lösungen für bestimmte Probleme geben können. Diese finden sowohl innerhalb als auch außerhalb der Physik Anwendungen [1, 2]. Konkret soll das Q-Learning in diesem Versuch im Kontext der brownscher Dynamik eingesetzt werden, um Q-Learning kennenzulernen. Zuerst kommt die Implementierung eines Random Walks, das später mit Q-Learning ergänzt wird. Danach wird das Q-Learning separat betrachtet, um die Bedeutung dessen Parameter genauer zu verstehen. Schließlich wird das Problem des Random Walks samt Q-Learning in einer weiterentwickelten Form untersucht.

2 Theoretische Grundlagen

2.1 Brownische Dynamik

Wenn ein größeres Teilchen von vielen kleineren Teilchen umgeben ist, dann stoßen die kleineren Teilchen aufgrund ihrer thermodynamischen Anregung mit dem größeren Teilchen zusammen. Die daraus resultierende Bewegung des größeren Teilchen wird als brownsche Bewegung bezeichnet [3]. Das Verhalten dieser Bewegung ist chaotisch und erscheint daher als zufällig. Auf der Grundlage dieser pseudozufälligen Bewegung kann die brownsche Bewegung als Random Walk approximiert werden [4]. Im Rahmen dieses Versuchs wird ein eindimensionaler Random Walk in diskreten Schritten als Modell gewählt. Die Wahrscheinlichkeit, dass das Teilchen sich l viele Schritte nach links und r viele Schritte nach rechts bewegt, ist gegeben durch

$$\omega_N(r) = \frac{N!}{r!l!}p^r q^l = \frac{N!}{r!(N-r)!}p^r(1-p)^{N-r}, \tag{1}$$

wobei p die Wahrscheinlichkeit für ein Schritt nach rechts, q die Wahrscheinlichkeit für ein Schritt nach links und $N = l + r$ die Gesamtzahl der Schritte ist.

Des Weiteren kann die Position $m = r - l$ eingeführt werden, wodurch aus $\omega_N((m+N)/2)$

$$P_N(m) = \frac{N!}{((m+N)/2)!((m-N)/2)!}p^{\frac{m+N}{2}}(1-p)^{\frac{N-m}{2}} \tag{2}$$

wird.

Für ein großes N nähert sich die Verteilung (2) aufgrund des zentralen Grenzwertsatzes einer Gaußverteilung, da diese eine Binomialverteilung darstellt [5]. Diese lautet allgemein

$$\mathcal{P}(x,N) \approx \frac{1}{\sqrt{2\pi\sigma^2}}\exp\left[-\frac{(x-\langle x\rangle)^2}{2\sigma^2}\right]. \tag{3}$$

Bezüglich (3) ist der Mittelwert aller möglichen Random Walks

$$\langle x\rangle = \sum_{i=1}^{N}\langle x_i\rangle = N\langle x_1\rangle = N(pa-(1-p)a) \tag{4}$$

$$= Na(2p-1) \tag{5}$$

und die Varianz aller möglichen Random Walks

$$\sigma^2 = \langle x^2 \rangle - \langle x \rangle^2 = N\sigma_1^2 = N\left(\langle x_i^2 \rangle - \langle x_1 \rangle^2\right) \tag{6}$$

$$= N\left(pa^2 + (1-p)a^2 - \langle x_1 \rangle^2\right) \tag{7}$$

$$= 4Na^2p(1-p), \tag{8}$$

wobei $a = |\Delta x|$ die Schrittweite ist.

Für $p = \frac{1}{2}$ wird (3) aufgrund (5) und (8) zu

$$\mathcal{P}(x,t) \approx \frac{1}{\sqrt{4\pi Dt}} \exp\left[-\frac{x^2}{4Dt}\right]. \tag{9}$$

Hierbei wurde N durch $\frac{t}{\tau}$ ersetzt, wobei t die Zeit und τ die Zeitspanne zwischen zwei Schritten ist, und zudem wurde der Diffusionskoeffizient

$$D = \frac{a^2}{2\tau} \tag{10}$$

eingeführt. Ein Teilchen, das der brownschen Dynamik unterliegt und ein Eigenantrieb besitzt, wird als (Mikro-)Schwimmer bezeichnet.

2.2 Q Learning

Maschinelles Lernen ist ein Bereich der Informatik, der sich mit Algorithmen auseinandersetzt, die automatisch Muster in Daten finden sollen, indem sie auf frühere Erfahrungen zurückgreifen. Erfahrung bedeutet hier, dass die Algorithmen so konzipiert sind, dass Informationen aus sogenannten Trainingsdaten in den Algorithmen eingespeist werden, um eine Verallgemeinerung der zugrundeliegenden Eigenschaften der Daten zu erhalten. Die Extraktion von Informationen mit dem Ziel der Verallgemeinerung wird als Lernen bezeichnet.

Dabei gibt es drei große Bereiche des maschinellen Lernens: supervised, unsupervised und reinforcement learning. Beim supervised learning sind die Trainingsdaten gekennzeichnet, sodass dem Lernprozess im Voraus vorgeschrieben wird, welche Daten zu welchen Ergebnissen führen sollen. Beim unsupervised learning sind die Daten nicht gelabelt und der Algorithmus muss selbst herausfinden, wie er die Daten clustern kann. Dies kann auch als Suche nach einer vorteilhafteren Darstellung der gegebenen Daten verstanden werden.

Beim Reinforcement Learning geht es darum, zu lernen, was zu tun ist, also welche Handlung, auf welche Situation folgen soll, um eine numerische Belohnungsfunktion zu maximieren. Dem lernenden Individuum, der sogenannte Agent, wird nicht gesagt, welche Handlungen er ergreifen soll, sondern muss durch Ausprobieren innerhalb einer Lernumgebung herausfinden, welche Handlungen die größte Belohnung mit sich bringen. Unter bestimmten Fällen können Handlungen nicht nur die unmittelbare Belohnung beeinflussen, sondern auch die nächste Situation und damit alle nachfolgenden Belohnungen. Diese beiden Merkmale - Versuch-und-Irrtum-Suche und verzögerte Belohnung - sind die beiden wichtigsten Merkmale des Reinforcement Learnings.

Q-Learning ist ein Reinforcement Learning Algorithmus, der versucht, die beste Aktion für den aktuellen Zustand zu finden. Er versucht, eine Strategie zu lernen, die die Gesamtbelohnung maximiert. Der zentrale Teil des Q Learning Algorithmus ist eine Matrize Q, die jedem möglichen Zustand jeweils ein

Wert zu jeder möglichen Aktion zuordnet. So stellt jede Zeile von Q ein Zustand in der Lernumgebung dar und jede Spalte eine Aktion, die vom Agenten ausgeführt werden kann.

Für den Versuch-und-Irrtum Aspekt des Lernens führt der Agent mit der Wahrscheinlichkeit ϵ eine zufällige Aktion aus. Demnach wird aber mit einer Wahrscheinlichkeit von $1 - \epsilon$ eine Aktion j so ausgewählt, dass für diese

$$\arg\max(Q_{ij}) \tag{11}$$

gilt, wenn der Agent sich im Zustand i befindet. Letztere stellt die Aktion dar, die vom Agent als Strategie gelernt wurde.

Nach jeder ausgeführten Aktion kommt der Lernprozess zum Einsatz: In der Matrix Q wird der alte Eintrage Q_{ij}^{alt} zu Q_{ij}^{neu} geändert, wobei dies nach der Vorschrift

$$Q_{ij}^{\text{neu}} = Q_{ij}^{\text{alt}} + \alpha(\mathcal{R} + \gamma \max(Q_{i'j}) - Q_{ij}^{\text{alt}}) \tag{12}$$

passiert. Dabei ist j die performierte Aktion, i der Zustand vor der Aktion, i' der Zustand nach der Aktion, α die Lernrate und γ der discount factor. Die Hyperfaktoren α und γ werden dabei später erläutert.

Wenn der Agent sein Ziel erreicht, ist der Durchlauf beendet. Insgesamt soll der Durchlauf N mal wiederholt werden. Die Durchläufe werden als Epochen oder Episoden bezeichnet. Bei jeder neuen Episode wird der Agent mit einem neuen beliebigen Zustand initialisiert.

Die Matrix Q, welche die Strategie des Agenten darstellt, soll im Idealfall am Ende des Lernprozesses zu einer vernünftigen Strategie konvergieren.

3 Durchführung

Für diesen Versuch wurde zwei Programmgerüste jeweils in eine eigene .py Datei bereitgestellt. In der Datei *fp_classes.py* sind zwei Klassen implementiert: agent und enviroment. Diese stellen den Agenten und dessen Lernumgebung vom Q-Learning dar. Die andere Datei *fp_reinforcement_learning_main.py* stellt das main Skript dar. Dort werden die Klassen aus *fp_classes.py* importiert und für die beiden jeweils Objekte erstellt. Darauf folgen verschiedene Programmteile: Der erste soll Werte vom mittleren Verschiebungsquadrat (MSD) berechnen und dann grafisch darstellen. Der zweite stellt die Trainingsschleife des Agenten dar, wo die Implementierung des Q-Learnings angewandt werden soll. Der letzte Teil erzeugt eine .gif Datei, welche das Ergebnis des Q-Learnings in einer visuellen Animation präsentiert.

Im Versuch soll ein Agent in einer eindimensionalen Umgebung trainiert werden. Die Umgebung hat N Zustände, von denen einer die Startposition und einer die Zielposition ist, und soll periodische Randbedingungen haben. Der Agent startet an der Startposition der Umgebung und soll ab an folgende Aktionen ausführen können: nach links gehen ($\leftarrow$), nach rechts ($\rightarrow$) gehen oder verweilen ($\downarrow$). Wenn der Agent auf der Zielposition der Umgebung verweilt ($\downarrow$), dann wird der Agent belohnt und der Durchlauf beendet.

Der erste Teil des Versuchs besteht darin, die restliche Implementierung der brownschen Dynamik zu vervollständigen, sodass der effektive MSD berechnet und dieser mit der Diffusionskonstante in Relation gesetzt werden kann. Dieser Teil wird später dann auskommentiert. Im zweiten Teil erfolgt die Implementierung des Q-Learning Algorithmus. So wird der Entscheidungsprozess

und die Belohnung des Agenten samt Berücksichtigung der Randbedingungen implementiert und eingesetzt. Dabei bleibt die Startposition zunächst immer gleich. Danach folgt die Analyse der Hyperparameter im dritten Teil des Versuchs. Zuerst wird die Auswirkung des Parameter γ beim Lernen in einer neuen Umgebung theoretisch untersucht. Die Analyse des Parameters α findet wieder in der alten Lernumgebung ohne brownsche Dynamik statt. Dort wird α im Hinblick auf die Effizienz des Lernprozesses bewertet, indem die benötigte Anzahl der Aktionen bis zum Ziel und dessen Verhältnis zur minimalen Anzahl der Aktionen bis zum Ziel betrachtet werden. Dabei wird auch die zufällige Initialisierung der Startposition verwendet. Zum Schluss des Versuchs wird die Lernumgebung um ein stochastisches Hindernis erweitert, wobei die initialisierte Startposition gleich bleibt. In einem Intervall der Umgebung soll der Agent unmittelbar nach seiner Aktion mit einer Wahrscheinlichkeit P_o nach links verschoben werden. Das ganze Programm wird dann in einer Schleife mit unterschiedlichen P_o gesetzt, um herauszufinden, bei ungefähr welchem P_o der Agent beginnt, den Weg über die Randbedingungen gegenüber den kürzeren direkten Weg zu präferieren.

4 Bearbeitung und Auswertung

4.1 Mittleres Verschiebungsquadrat

Für die Berechnung des mittleren Verschiebungsquadrats der brownschen Bewegung muss die Diffusionskonstante zu der Wahrscheinlichkeit P_{diff} übersetzt werden. Diese beschreibt die Wahrscheinlichkeit, dass sich der Agent einen zufälligen Schritt nach rechts oder links geht. Aus dem Ausdruck (10) folgt, dass die Wahrscheinlichkeit gegeben ist durch

$$P_{diff} = \frac{1}{\tau} = 2D, \tag{13}$$

wobei die Schrittweite hier auf a=1 gesetzt wird. Nachdem der zufällige Schritt als Funktion random_step implementiert wurde, kann $\langle x^2 \rangle$ numerisch berechnet werden, indem der Agent in einer Schleife insgesamt 100 zufällige Schritte für jeweils 20000 Episoden geht und dabei x^2 gespeichert wird. Für $\langle x^2 \rangle$ gilt die Relation $\langle x^2 \rangle = 2Dt$, sodass aus $\langle x^2 \rangle$ der effektive Diffusionskoeffizient D_{eff} mit einer linearen Regression bestimmt werden kann, da die Steigung der linearen Regression $2D$ entspricht. Die Diagramme 1 stellen die berechneten Werte für drei verschiedene eingestellte D dar. Es ist dabei gut zu erkennen, dass die Daten sehr gut durch eine lineare Regression beschreiben lassen. Zudem liegen die effektiven Diffusionskoeffizienten D_{eff} sehr nah an den im Programm eingestellten D.

4.2 Implementation des Lernalgorithmus

Die Implementation erfolgte gemäß der Versuchsdurchführung, wobei die Spalten j in der Q Matrix den Aktionen $\leftarrow = 0$, $\downarrow = 1$ und $\rightarrow = 2$ entsprechen. Darüber hinaus wird die Diffusionskonstante D auf 1/8 gesetzt, damit nur in allen vier Schritten ein zufälliger Schritt zu erwarten ist. Die Hyperparameter werden auf $\alpha = 0,01$ und $\gamma = 0,9$ eingestellt. In einer Schleife sollen die Funktionen adjust_epsilon, choose_action, random_step, perform_action und update_Q werden aufgerufen werden. adjust_epsilon soll ϵ so ändern, sodass bei

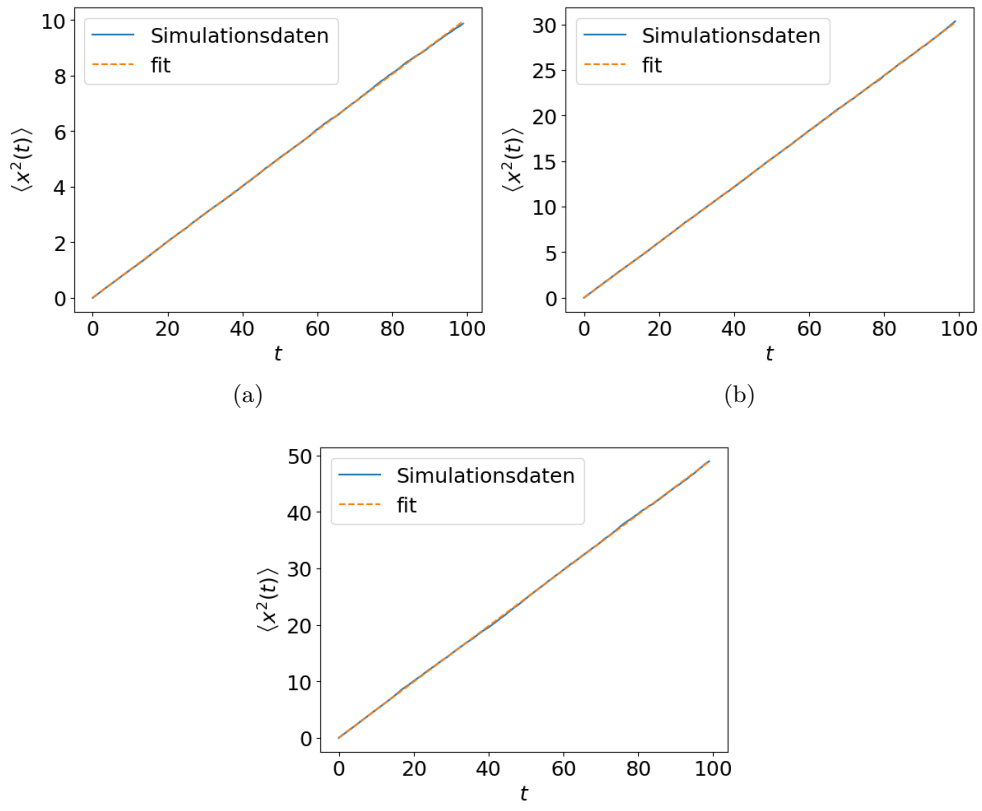


Abbildung 1: Hier wird $\langle x^2(t) \rangle$ gegen die Zeit t aufgetragen. Die Hälfte der Steigung entspricht D_{eff}.Bei (a) liegt $D_{eff} \approx 0,0502$ für $agent.D = 0,05$ vor. Bei (b) liegt $D_{eff} \approx 0,1525$ für $agent.D = 0,15$ vor. Bei (c) liegt $D_{eff} \approx 0,2469$ für $agent.D = 0,25$ vor.

der ersten Epside $\epsilon = 1$ und bei der $agent.zero\_fraction * agent.N\_episodes$ $\epsilon = 0$ ist, wobei Epsilon dazwischen linear abfallen soll und in diesem Versuch $agent.zero\_fraction = 0,9$ ist. choose_action soll entweder eine zufällige Aktion wählen (mit Wahrscheinlichkeit ϵ) oder gemäß (11) eine Aktion auswählen. Falls es mehrere Maxima für (11) gibt, dann soll auch eine zufällige Aktion gewählt werden. perform_action führt die gewählte Aktion mit Berücksichtigung der Randbedingungen aus. update_Q verändert die Matrix Q gemäß (12). Insgesamt wird die Schleife 10000-mal ausgeführt.

Das Resultat des Trainings wird in der .gif Datei ($LAST\_TRAJECTORY\_22-07-26T23\_26\_24$) grafisch zur Schau gestellt, wo die Tendenz des Agenten in Richtung Belohnung ersichtlich ist. Zudem wird in der Q Matrix (in der Datei $Q\_MATRIX\_22-07-26T23\_26\_24.txt$) die Aktionen ($\downarrow$) für die Zielposition, ($\rightarrow$) für die Position links davon und ($\leftarrow$) für die Position rechts davon „bevorzugt“, da diese jeweils in ihren Zeilen das Maximum darstellen. Diese zwei Punkte deutet in gewissen Maßen auf den Erfolg des Trainings hin.

4.3 Wahl der Hyperparameter

4.3.1 Discount factor

Um die Aufgabe des Hyperparameters γ zu verstehen, sollte dieser in einer vereinfachten Umgebung getrachtet werden: ein eindimensionales Pfad-Finde Problem, mit nur vier Zuständen (0,1,2,3), zwei möglichen Aktionen ($\leftarrow = 0$, $\rightarrow = 1$) und ohne periodische Randbedingungen (Aktion $\leftarrow$ im Zustand 0 resultiert in keine Zustandsänderung). Der Agent beginnt immer im Zustand 0 und das Ziel befindet sich im Zustand 3. Wenn der Agent den Zustand 3 erreicht, wird die Episode beendet. Hierfür wird dem Agenten eine Belohnung $\mathcal{R}$ gegeben, falls dieser im Zustand 2 die Aktion $\rightarrow$ wählt. Wenn beispielsweise der Agent in der ersten Episode eine beliebige Aktionsfolge (z.B. $\rightarrow\rightarrow\rightarrow$) ausführt, dann wird die Matrix

$$Q = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & \alpha\mathcal{R} \\ 0 & 0 \end{bmatrix}. \tag{14}$$

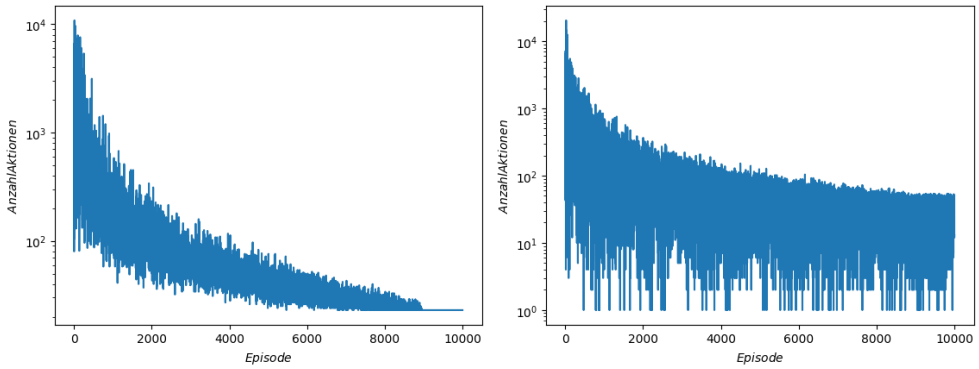
Wenn dann eine Aktionsfolge $\rightarrow\leftarrow\rightarrow\rightarrow\rightarrow$ in der zweiten Episode folgt, dann wird

$$Q = \begin{bmatrix} 0 & 0 \\ 0 & \gamma\alpha\mathcal{R} \\ 0 & 2\alpha\mathcal{R} - \alpha^2\mathcal{R} \\ 0 & 0 \end{bmatrix}. \tag{15}$$

Wird nun in der dritten Episode die Aktion $\rightarrow$ ausgeführt, dann ist

$$Q = \begin{bmatrix} 0 & \gamma^2\alpha\mathcal{R} \\ 0 & \gamma\alpha\mathcal{R} \\ 0 & 2\alpha\mathcal{R} - \alpha^2\mathcal{R} \\ 0 & 0 \end{bmatrix}. \tag{16}$$

Hier wird deutlich, dass der discount factor γ die Propagation der Belohnung im Zustand 3 auf die anderen Zustände reguliert. Dies ist ein wichtiger Teil des Q-Learning Algorithmus. Denn es ist sinnvoll bei der Entwicklung einer Strategie, dass nicht nur das Erreichen eines Zieles belohnt wird, sondern auch, wenn der Agent sich in dessen Richtung bewegt.



(a) (b)

Abbildung 2: Hier wird die Anzahl der Aktionen, die der Algorithmus bis zum Ziel braucht, für jede Episode dargestellt. Für beide Fälle gilt $\alpha = 0,01$. (a) ist die Lernkurve für ein konstanten Startzustand 8. (b) ist die Lernkurve für ein zufälligen Startzustand.

4.3.2 Learning rate

Die Learning rate α stellt die Geschwindigkeit der Konvergenz des Lernalgorithmus dar. Der Parameter α reguliert bei (12) die „Schrittweite“ der Veränderung von der Matrix Q für jeden Schritt. Ist α große, dann wird Q bei jeden Schritt stärker verändert. Dies hat den Vorteil, dass Q am Anfang sehr schnell zu einer gewünschten Strategie konvergiert. Doch wegen der großen Schrittweite kann der Algorithmus „über das Ziel hinaus schießen“, sodass Q die gewünschte Strategie verfehlt und nicht mehr zu ihr konvergiert. Ist α dagegen zu klein, dann ist die Konvergenz sicherer, doch braucht der Algorithmus insgesamt mehr Schritte und damit auch mehr Training, um die gewünschte Strategie zu erreichen. Um beide Szenarien zu vermeiden, sollte die Performance von verschiedenen α mit einer Methode bemessen werden können.

Dazu wird die Anzahl der benötigten Aktionen bis zum Ziel des Algorithmus für jede Episode gemessen, wobei $D = 0$ gesetzt wird, damit α ohne brownsche Dynamik untersucht werden kann. Daran sollte zu erkennen sein, wie gut der Algorithmus im Verlauf der Episoden lernt. Diese Darstellung der Performance gegen die Simulationszeit wird als Lernkurve bezeichnet. Die Diagramme 2 zeigen, dass die Lernkurve bei einem konstanten Startzustand bis zur Episode $zero\_fraction * N\_episodes$ (hier 9000) ungefähr exponentiell abfällt und danach einen konstanten Wert annimmt. Dagegen ist bei einem zufälligen Startzustand direkt keine Konvergenz gegen einen Wert zu erkennen, da der zufällige Startzustand immer unterschiedlich viele Schritte bis zum Ziel benötigt. Um dies mit einzubeziehen, soll die Lernkurve statt die Anzahl der Aktionen bis zum Ziel durch dessen Verhältnis zur minimal erforderlichen Anzahl von Aktionen bis zum Ziel darstellen. Dadurch wird die Performance des Algorithmus bezüglich der bestmöglichen Performance wiedergegeben, was der Lernkurve im Diagramm 3 mehr Aussagekraft gibt. Dass das Diagramm dabei gegen eins geht, bedeutet, dass die Strategie gegen die bestmöglichste Performance konvergiert.

Die Lernkurve eignet sich nun, um den Einfluss von verschiedenen Werten

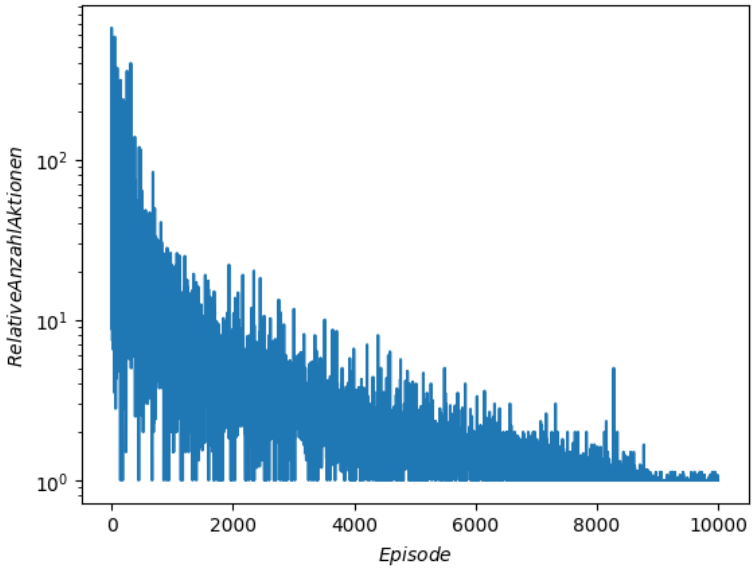
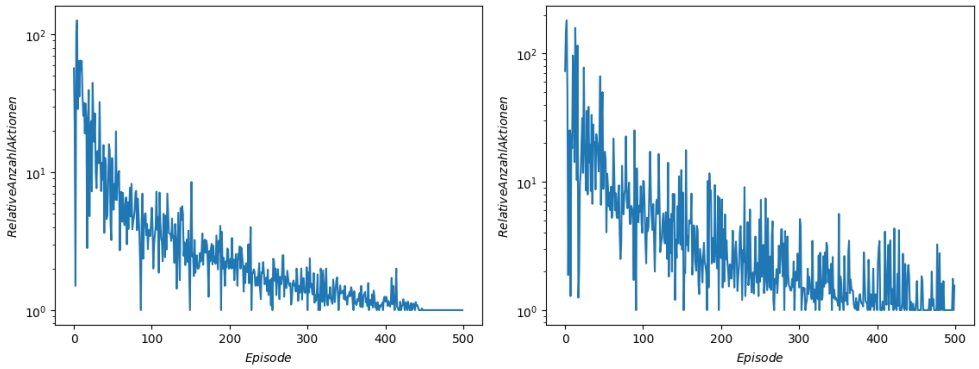
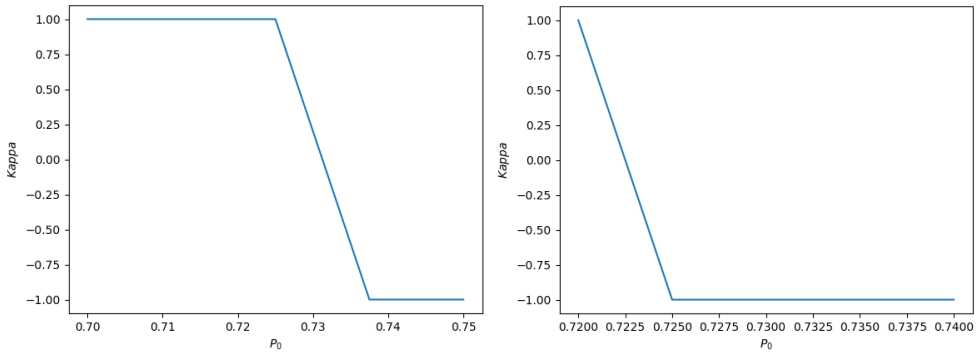


Abbildung 3: Hier wird das Verhältnis der Anzahl der Aktionen, die der Algorithmus braucht, zu der Anzahl der Aktionen, die mindestens erforderlich bis zum Ziel sind, für jede Episode dargestellt, die hier als "relative Anzahl"bezeichnet wird. Hierbei ist $\alpha = 0,01$.



(a) (b)

Abbildung 4: Hier werden die Anzahl der benötigten Aktionen relativ zur minimalen Anzahl der Aktion bis zum Ziel dargestellt. Für (a) gilt $\alpha = 0,01$. Für (b) gilt $\alpha = 0,999999$.



(a) (b)

Abbildung 5: Hier wird κ für verschiedene P_o dargestellt. Bei (a) erfolgt die Darstellung für eine größere Spannweite von P_o als bei (b). Hierbei ist $\alpha = 0,01$.

für α zu untersuchen. Die Diagramme 4 zeigen die Performance für $\alpha = 0,01$ und $\alpha = 0,999999$, wobei die Anzahl der Episoden auf 500 gestellt wurde, um deren Unterschied besser zu erkennen. Es kann beobachtet werden, dass das Diagramm mit $\alpha = 0,999999$ etwas langsamer abfällt und damit auch schlechter konvergiert als das mit $\alpha = 0,999999$. Gleichzeit liefert $\alpha = 0,999999$ keinen konstanten Wert am Ende, was auch auf eine schlechtere Performance hindeutet. Die Methode der Lernkurve konnte hier zeigen, dass ein viel zu großes α schlechter konvergiert als ein kleineres.

4.4 Stochastisches Hindernis

Der Agent wird in einer komplexeren Umgebung ausprobiert: Die Anzahl der Zustände wird auf 15 reduziert, der Startzustand wird auf 8, der Zielzustand wird auf 12 und die Hyperparameter auf $\alpha = 0,1$ und $\gamma = 0,9$ gestellt. Der Agent durchläuft 10000 Episoden. Diesmal wird ein stochastisches Hindernis hinzugefügt, sodass der Agent im Intervall von den Zuständen 8 bis 11 mit einer Wahrscheinlichkeit $P_{obstacle}$ nach links verschoben wird. Die dazu gehörende Funktion stoch_obstacle soll dementsprechend implementiert und nach random_step in der Schleife aufgerufen werden. Wenn epsilon=0 ist, dann soll eine Variable *total_action_displacement* hochgezählt werden, falls der Agent nach rechts geht, und runtergezählt werden, falls der Agent nach links geht. Dadurch kann eine Variable $\kappa = \frac{total\_action\_displacement}{abs(total\_action\_displacement)}$ definiert werden, die -1 ist, wenn der Agent insgesamt mehr nach links geht, und +1 ist, wenn der Agent insgesamt mehr nach rechts geht.

Für ein bestimmtes P_o sollte κ von +1 zu -1 übergehen, da das stochastische Hindernis so groß werden kann, dass es den zusätzlichen Weg in linker Richtung übertrifft. Um diesen Übergang einzuschätzen, wird der „zusätzliche“ Weg des stochastischen Hindernis betrachtet: Auf der Länge s_{int} des Intervalls vom stochastischen Hindernis ist zu erwarten, dass der Agent zunächst um $s_{int} \times P_o$ nach links verschoben wird und damit effektiv $s_{int} \times P_o$ mehr Schritte gehen muss, um das Ziel zu erreichen. Da sich der Agent jedoch selbst wieder im Intervall auffinden kann, nachdem er nach links verschoben wurde, erlebt der Agent auch auf dem bereits verursachten zusätzlichen Weg wieder eine Verschiebung mit einer Wahrscheinlichkeit P_o. Dadurch entsteht erneut

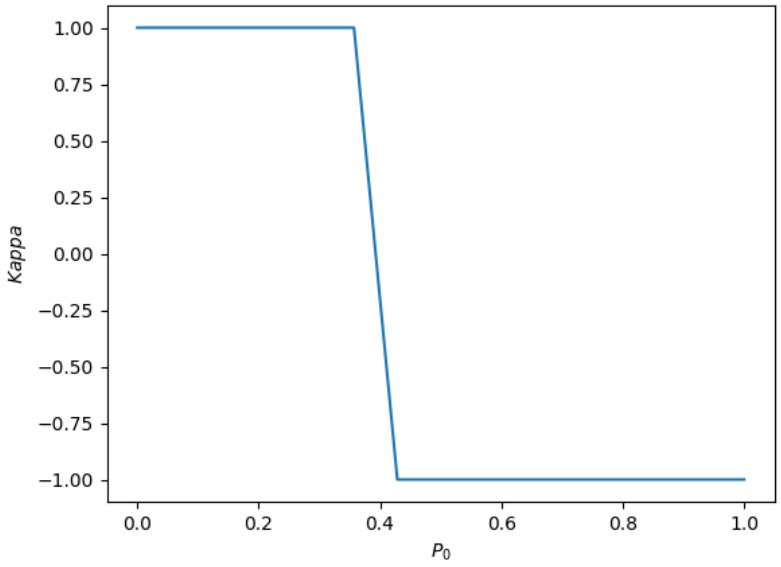


Abbildung 6: Hier wird κ für $\alpha = 0,999999$ gegen verschiedene P_0 aufgetragen. Hier ist es deutlich zu erkennen, dass der Übergang von P_0 anders ist als bei 5.

ein zusätzlicher Weg von $(s_{int} \times P_o) \times P_o = s_{int} \times P^2$, welcher wiederum mit der gleich Argumentation um $s_{int} \times P_o^3$ weiter verschoben wird, und so weiter. Damit ergibt sich der effektive Weg über das stochastische Hindernis insgesamt als

$$s_o = s_{int} \sum_{k=0}^{\infty} P_o^k = \frac{s_{int}}{1 - P_o}, \tag{17}$$

wobei hier die geometrische Reihe angewandt wird. Aus der Gleichsetzung des effektiven Wegs von links und von rechts

$$s_{links} \overset{!}{=} s_o = \frac{s_{int}}{1 - P_o} \tag{18}$$

folgt, dass der Übergang bei $P \approx 0,727$ liegen sollte, da in der Lernumgebung $s_{links} = 11$ und $s_{int} = 3$ ist.

Um den Übergang bei P_o numerisch zu finden, wird das main Skript in einer Schleife gelegt, die bei jedem Durchgang ein anderes P_o verwendet. Die dabei aufgezeichneten Werte von κ können dann in einem Diagramm gegen deren P_o grafisch darstellt werden. Das Ergebnis im Diagramm 5 zeigt, dass der erwartete Wert des Übergangs in etwa mit den numerischen Berechnungen übereinstimmen.

Hier bietet es sich auch, die Performance des Hyperparameters α zu bewerten: Wird hier $\alpha = 0,999999$ verwendet, dann ist im Diagramm 6 zu erkennen, dass auch die Strategie schlechter wird, wo deutlich früher als dem Idealfall (18) der linke Weg präferiert wird, also dass der Übergang deutlich unter $P \approx 0,727$ liegt.

5 Fazit

Q-Learning bietet eine relativ einfache Möglichkeit, Aufgaben wie Pfad-Finde Probleme numerisch zu lösen, selbst wenn sie etwas komplexer sein können. So

konnte dies im Abschnitt 4.4 des Versuchs anhand der Anpassung der Strategie an speziellen Umgebungsparameter demonstriert werden. Solche Pfad-Finde Probleme lassen sich oft mit physikalischen Interpretationen verbinden, sodass Q-Learning als Basis von komplizierteren Algorithmen in der Physik Verwendung finden kann. Der Lernalgorithmus an sich kann auf verschiedene Art und Weisen modifiziert werden: Durch mehr Zustände und möglichen Aktionen kann beispielsweise die Dimensionalität, Schrittweiten und Dynamik des Algorithmus an einem bestimmten Problem angepasst werden. So stellt Q-Learning eine alternative, numerische Lösungsmethode dar, solange eine Matrix Q sich für diese Probleme beschreiben lässt.

Im Versuch wurde in 4.3 die Bedeutung der Hyperparameter erörtert und teilweise ihre Implikationen auf die Berechnung der Strategie beobachtet. So hat α entscheidene Auswirkungen für die Konvergenzgeschwindigkeit und γ regelt, inwiefern Belohnungen auf benachbarten Zuständen sich ausweiten. Mit Lernkurven wurde zudem ein Werkzeug eingeführt, an dem die Performance der Hyperparameter gemessen wird. Basierend darauf können die Parameter quantitativ dann festgelegen werden.

Literatur

[1] J. D. Martín-Guerrero and L. Lamata, *Reinforcement Learning and Physics*, Applied Sciences **11** (2021), 10.3390/app11188589.

[2] J. Clifton and E. Laber, *Q-Learning: Theory and Applications*, Annual Review of Statistics and Its Application **7**, 279 (2020), https://doi.org/10.1146/annurev-statistics-031219-041220.

[3] *BROWNsche Bewegung*, (2020), [Online; accessed 13-July-2022].

[4] O. A. Korasj, *Brown'sche Bewegung*, (2018), [Online; accessed 13-July-2022].

[5] A. Košmrlj, *MAE 545: Lecture 17 (4/10)*, (2018), [Online; accessed 12-July-2022].