Auswertung FP-Versuch: Reinforcement learning für intelligente Brownsche Dynamik

von Edgar Hartmann (2814524)

Einleitung

In diesem Versuch wurden Grundlagen des reinforcment leanrings anhand des Q-learning Algorithmus angewendet auf die Brownsche Dynamik erlernt.

Dabei wurde dem sogenannten Agenten auf einem 1-dimensionalen Zahlenstrahl beigebracht von einer Startposition zu einem Ziel zu gelangen, während dieser der zufälligen brownschen Dynamik unterlag. Zunächst von einer festen, dann von einer zufälligen Startposition aus und zuletzt wurde ein stochastisches Hindernis eingebaut.

Der Code ist in python3 geschrieben. Es wurden mit dem Versuch Grundlagen des objektorientierten Programmierens gelehrt.

Aufgaben

3.1 Mittleres Verschiebungsquadrat

In diesem Teil der Aufgaben wird die brownsche Dynamik im Code implementiert und mit dem mittleren Verschiebungsquadrat (mean squared displacement (MSD)) analysiert.

Der 1-dim. Zahlenstrahl wird durch das Intervall $[0{,}99]$ (festgelegt durch self.N_states) beschrieben und besitzt periodische Randbedingungen.

Die brownsche Dynamik wird mithilfe des Zufalls-Pakets *np.random* von *numpy* realisiert.

2.

Um auf die Definition der Variablen *self.P_diffstep* in Abhängigkeit von *self.D* zu kommen, wurde folgende Gleichung der Diffusionskonstanten D verwendet:

$$D \coloneqq \frac{a^2}{2\tau} \tag{1}$$

wobei $a = |\Delta x|$ die Schrittweite und τ die Zeitspanne zwischen zwei Schritten ist. Es wird die Wahrscheinlichkeit (*self.P_diffstep* $:= P$) eingeführt, sodass in jedem Zeitschritt nur eine Wahrscheinlichkeit P existiert, dass ein Diffusionsschritt passiert. Daraus folgt für D:

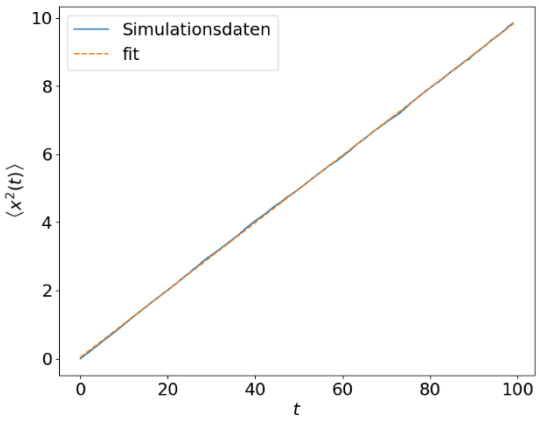
$$D = \frac{a^2}{2*\frac{1}{P}*\tau}. \tag{2}$$

Mit $\tau = 1$ und $a = 1$ folgt:

$$P = \frac{2\tau}{a^2} D = 2D. \tag{3}$$

Gleichung (3) wurde im Code eingebaut.

6. - 8.



Auf den Abb. 1 und 2 ist der MSD $\langle x^2 \rangle$ gegen die Zeit t aufgetragen für verschiedene Diffusionskonstanten. Innerhalb des Codes wurde die Diffusionskonstante anhand

$$\langle x^2 \rangle = 2Dt \leftrightarrow D = \frac{\langle x^2 \rangle}{2t} \qquad (4)$$

berechnet. Wobei $\frac{\langle x^2 \rangle}{t}$ die Steigung des linearen Fits ist.

Es ist zu erkennen, dass die Simulationsdaten und der jeweilige lineare Fit gut übereinstimmen

Abbildung 1: MSD gegen die Zeit mit eingestellter Diffusionskonstanten $D_0 = 0.05$ und berechnetem $D_1 = 0.049$

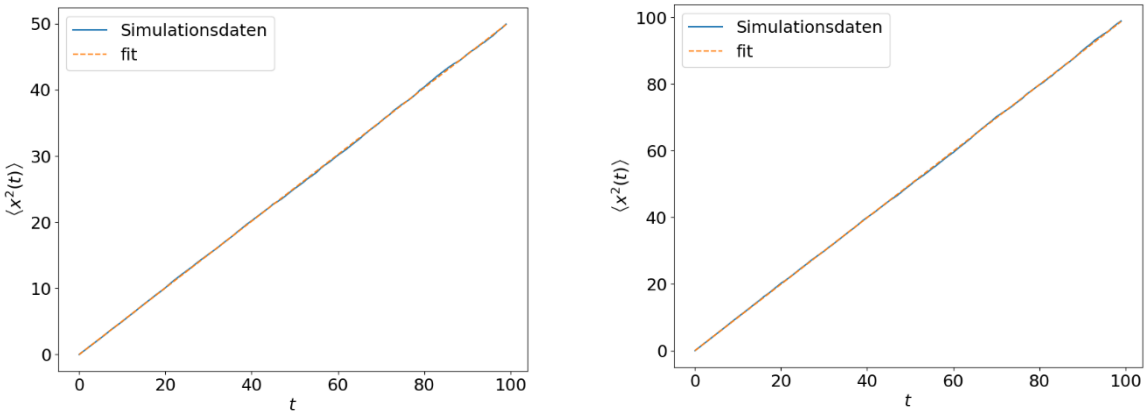


Abbildung 2: MSD gegen die Zeit mit $D_0 = 0.25$ und berechnetem $D_1 = 0.252$ (links). $D_0 = 0.5$ und $D_1 = 0.499$ (rechts)

3.2 Implementation des Lernalgorithmus

In diesem Teil wird der Lernalgorithmus implementiert.

Hierbei wurde das Q-learning verwendet, wobei die Q-Matrix (Q_{ij}) erstellt wird, die ständig aktualisiert wird. Anhand dieser Matrix navigiert sich der Agent durch das Problem.

Der Zeilenindex i in Q_{ij} steht für jeden möglichen Zustand, in welchem sich der Agent befinden kann, in diesem Fall die Position x auf dem Zahlenstrahl. Der Spaltenindex j steht für die mögliche Aktion, welcher der Agent ausführen kann. Hier entweder nach links laufen, stehen bleiben oder nach rechts laufen (←= 0, ↓= 1, →= 2). Der Agent verwendet die Strategie, dass er die Aktion j wählt, welche das Maximum von Q_{ij} im Zustand i ist.

Der Agent führt die Aufgabe, das Ziel zu erreichen, N mal durch. Dieses Durchführen wird als Episode oder Epoche bezeichnet. Dabei wählt der Agent eine zufällige Aktion mit einer gewissen Wahrscheinlichkeit ϵ aus und mit einer Wahrscheinlichkeit $1 - \epsilon$ eine Aktion mit zuvor genannter Strategie. Der Agent durchläuft viele Episoden (hier *agent.N_episodes* $= 10^4$) bis Q_{ij} so konvergiert ist, dass der Agent zuverlässig zum Ziel findet. ϵ wird von Episode zu Episode kleiner bis $\epsilon = 0$ ist, danach werden Aktionen nur noch auf Basis von Q_{ij} getroffen.

Q_{ij} wird dann nach jeder Aktion gemäß

$$Q_{ij}^{neu} = Q_{ij}^{alt} + \alpha(R_{ij} + \gamma \max_{j}(Q_{i'j}) - Q_{ij}^{alt}) \quad (5)$$

aktualisiert. Hier ist R_{ij} die Belohnung, die der Agent bekommt, wenn er in Zustand i die Aktion j ausführt. α ist die learning rate und γ der discount factor, genauere Betrachtung in Abschnitt 3.3 und für diesen Abschnitt stets: $\alpha = 0.01$ und $\gamma = 0.9$.

In dieser Aufgabe startet der Agent immer im Zustand $i = 30$ (bzw. der Position $x = 30$) und das Ziel ist immer an der Zielposition *self.target_position* $= 8$.

3.

Damit $P = 1/4$ gilt, folgt aus Gl. (3), dass $D = 1/8$ sein muss.

12.

In dieser Aufgabe wird Q_{ij} am Ende einer Simulation in eine Text-Datei geschrieben. Anhand Q_{ij} in diesem Zustand kann der Agent mit zuvor beschriebener Strategie sicher zum Ziel finden. Zu erkennen ist das daran, dass sich in der Zeile des Zielzustandes die bevorzugte Aktion ↓= 1 ist. Links davon, in Q_{ij} also oberhalb des Zielzustandes, ist die bevorzugte Aktion →= 2 und rechts neben dem Zielzustand ist die bevorzugte Aktion ←= 0.

Außerdem ist zu erkennen, dass der Agent nur etwa die Hälft der Zustände überhaupt einmal betreten hat, weil der Agent durch die feste Start- und Zielposition sowie der periodischen Randbedingungen nie die Entscheidung trifft dort hinzugehen.

In der ausgegebenen GIF-Datei ist zu erkennen, dass der Agent, der Hund Lasse, zuverlässig den direkten Weg zum Ziel, dem Hundeknochen, gefunden hat.

3.3 Wahl der Hyperparameter

Die Hyperparameter müssen auf jedes Problem angepasst werden und sind ausschlaggebend für die Konvergenz des Algorithmus gegen eine sinnvolle Strategie. Im Folgenden wird das Verhalten der Parameter an dem Beispiel der brownschen Dynamik untersucht.

3.3.1 Discount factor γ

Mit dem *discount factor* kann kontrolliert werden, wie schnell die Belohnung (Reward) über die Zustände hinweg propagiert. Werte des *discount factors* sind etwa $0.5 < \gamma < 1$.

1.

Die Q-Matrix hat in diesem Modell die Form 4×2.

2.

Zu Beginn:

$Q_{ij} = 0 \ \forall \ i,j \quad (i \in \{0,1,2,3\}, \ j \in \{0 = \leftarrow, 1 = \rightarrow\})$, Startposition:0, Ziel:3 und $R_{21} = R$

Epoche (1): Beliebige Aktionsfolge: →→←→→ und nach Gl. (5):

→: $Q_{01}^{neu} = Q_{01}^{alt} + \alpha \left[R_{01} + \gamma \max_j(Q_{1j}) - Q_{01}^{alt}\right] = 0 + \alpha[0 + \gamma 0 - 0] = 0$

Analog folgt für →: $Q_{11}^{neu} = 0$, für ←: $Q_{20}^{neu} = 0$ und für →: $Q_{11}^{neu} = 0$
Zuletzt:

→: $Q_{21}^{neu} = Q_{21}^{alt} + \alpha \left[R_{21} + \gamma \max_j(Q_{3j}) - Q_{21}^{alt}\right] = 0 + \alpha[R + \gamma 0 + 0] = \alpha R$

Also: $$Q_{(1)} = \begin{pmatrix} 0 & 0 \\ 0 & 0 \\ 0 & \alpha R \\ 0 & 0 \end{pmatrix}$$

3.
Epoche (2): Beliebige Aktionsfolge: →←→→→
Wie in 2. gilt für →: $Q_{01}^{neu} = 0$, für ←: $Q_{10}^{neu} = 0$ und für →: $Q_{01}^{neu} = 0$

→: $Q_{11}^{neu} = Q_{11}^{alt} + \alpha \left[R_{11} + \gamma \max_j(Q_{2j}) - Q_{11}^{alt}\right] = 0 + \alpha[0 + \gamma\alpha R + 0] = \gamma\alpha^2 R$

Und zuletzt:

→: $Q_{21}^{neu} = Q_{21}^{alt} + \alpha \left[R_{21} + \gamma \max_j(Q_{3j}) - Q_{21}^{alt}\right] = \alpha R + \alpha[R + \gamma 0 + 0] = 2\alpha R$

Also: $$Q_{(2)} = \begin{pmatrix} 0 & 0 \\ 0 & \gamma\alpha^2 R \\ 0 & 2\alpha R \\ 0 & 0 \end{pmatrix}$$

4.
Epoche (3): Aktion: →

→: $Q_{01}^{neu} = Q_{01}^{alt} + \alpha \left[R_{01} + \gamma \max_j(Q_{1j}) - Q_{01}^{alt}\right] = 0 + \alpha[0 + \gamma\gamma\alpha^2 R - 0] = \gamma^2\alpha^3 R$

Also: $$Q_{(3)} = \begin{pmatrix} 0 & \gamma^2\alpha^3 R \\ 0 & \gamma\alpha^2 R \\ 0 & 2\alpha R \\ 0 & 0 \end{pmatrix}$$

5.
Der Agent hat nach 3 Epochen in Zustand 0 bereits von dem Ziel in Zustand 3 gelernt, denn durch den Algorithmus lernt der Agent in jeder Epoche jeweils einen Schritt weiter entfernt von dem Ziel bzw. dem Reward. Da die Entfernung zum Ziel genau 3 Schritte sind, hat der Agent nach bereits 3 Epochen von dem Reward gelernt.

Für Zustand 0 wird der Agent gelernt haben nach rechts zu laufen.

6.

Nein, der Agent kann in diesem Modell nicht lernen nach links zu laufen, weil keine periodischen Randbedingungen gegeben sind und damit der kürzeste Weg drei Schritte nach rechts zu laufen ist.

3.3.2 Learning rate α

Die learning rate ist für die Dauer bis zur Konvergenz der Algorithmus verantwortlich. Allerdings führen zu große Werte von α zu einer schlechten oder gar keiner Konvergenz. Es gilt immer $\alpha < 1$.

3.

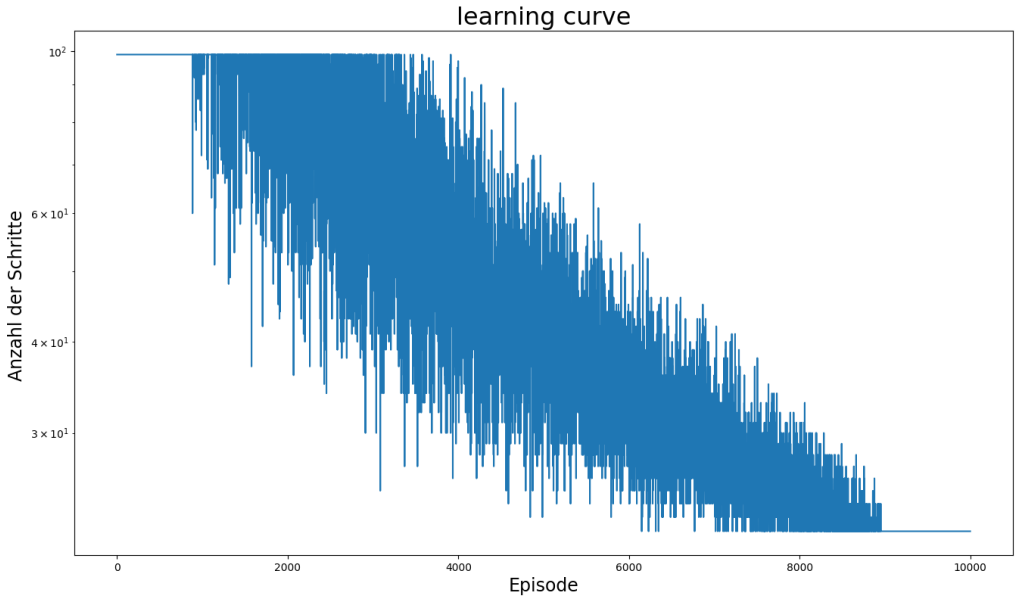


Abbildung 3: Lernkurve bei fester Start- und Zielposition

In Abb. 3 ist eine Lernkurve (learning curve) des Agenten zu sehen, wobei die Anzahl der benötigten Schritte zum Ziel zu laufen gegen die Episoden aufgetragen wird. In dieser Simulation ist die Startposition des Agenten und die Zielposition fest. Es ist zu erkennen, dass die Kurve etwa exponentiell abfällt. Außerdem ist die Kurve konstant für

Episode >= agent.zero_fraction∗agent.N_episodes = 9000,

also die Episode ab welcher der Agent nur noch Entscheidungen auf Basis der Q-Matrix trifft.

4.

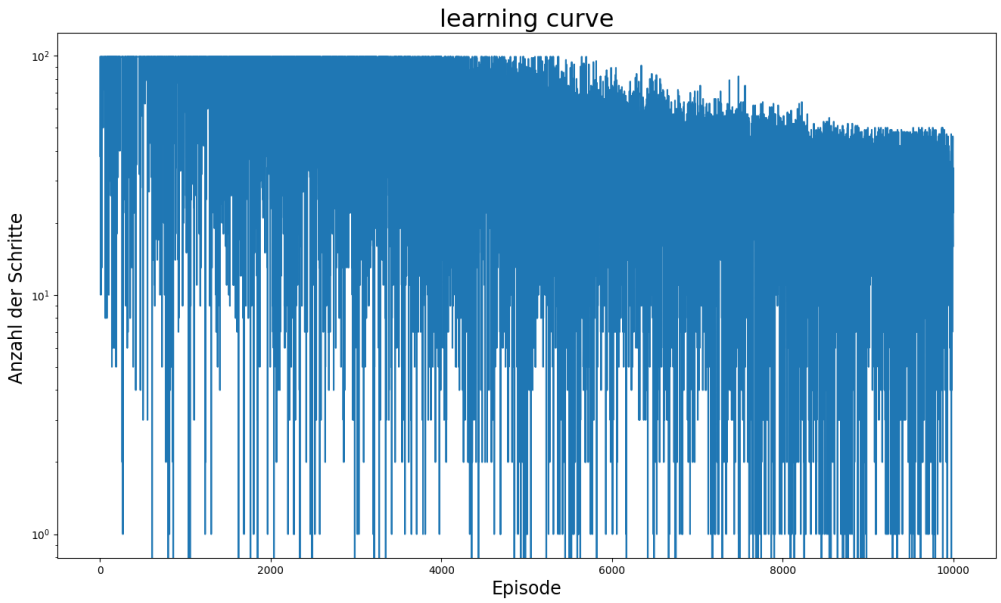


Abbildung 4: Lernkurve mit zufälliger Startposition des Agenten

In Abb. 4 ist eine Lernkurve mit zufälliger Startposition des Agenten zu sehen. Zu erkennen ist, dass die Kurve zu hohen Episoden hin leicht abfällt, aber nicht zu einem konstanten Wert konvergiert.

7.

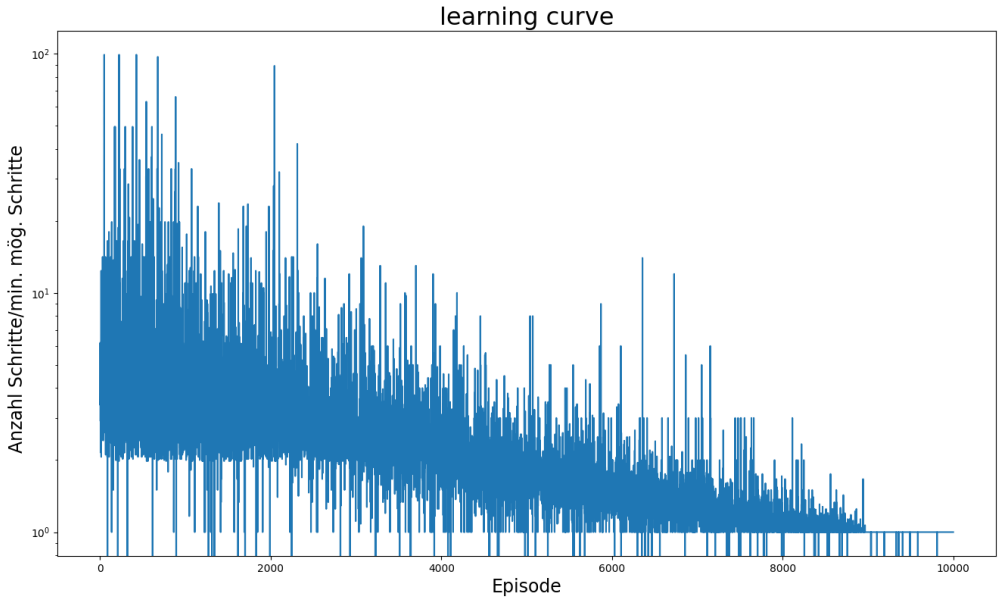


Abbildung 5: Lernkurve mit zufälliger Startposition Anzahl der Schritte auf minimal mögliche Schritte normiert

In Abb. 5 ist zu erkennen, dass die Lernkurve nun gegen 1 konvergiert, weil die Anzahl der Schritte (im Code *count_steps*) auf die minimal möglichen Schritte (*min_actions*) normiert wurde.

Einige Werte sind hier kleiner als 1, was theoretisch nicht möglich sein sollte, da aber *min_actions* $= 0$ sein kann, würde bei der Normierung, dann durch 0 geteilt werden. Daher wird im Code geprüft ob *min_actions* $== 0$ und für diesen Fall *min_actions* $= 1$ gesetzt. Der Wert ist egal, weil der Agent 0 Schritte ausführt und dann ist immer *count_steps* / *min_actions* $= 0$. Daher sind alle Werte < 1 in Abb. 5 genau $= 0$, was nicht weiter störend ist, da diese Episoden keinen Einfluss auf den Lernprozess haben.

8.

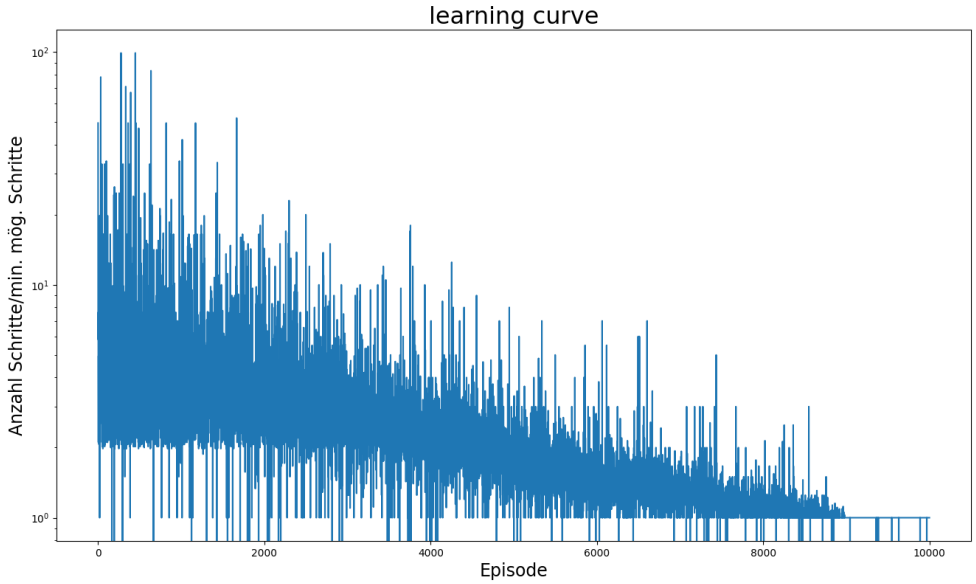


Abbildung 6: Lernkurve normiert mit α=0.999999

In dieser Aufgabe sollte das Verhalten der Lernkurve mit hohen Werten für α untersucht werden. Jedoch unterscheidet sich das Ergebnis hier (Abb. 6) nicht von den Ergebnissen zuvor.

Daher wurde im Folgenden die Anzahl der Episoden reduziert.

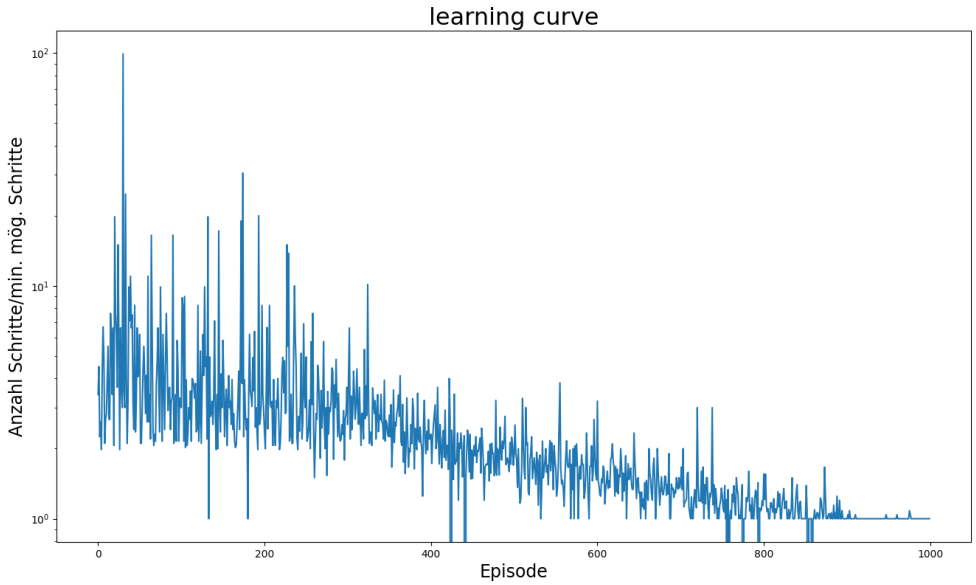


Abbildung 7: Lernkurve normiert und mit α=0.999999 und 1000 Episoden

Mit 1000 Episoden in Abb. 7 ist kaum ein Unterschied im Vergleich mit Abb. 5 zu sehen. Die Konvergenz ist nach

$$Episode >= agent.zero\_fraction * agent.N\_episodes = 900$$

noch fast konstant gegen 1.

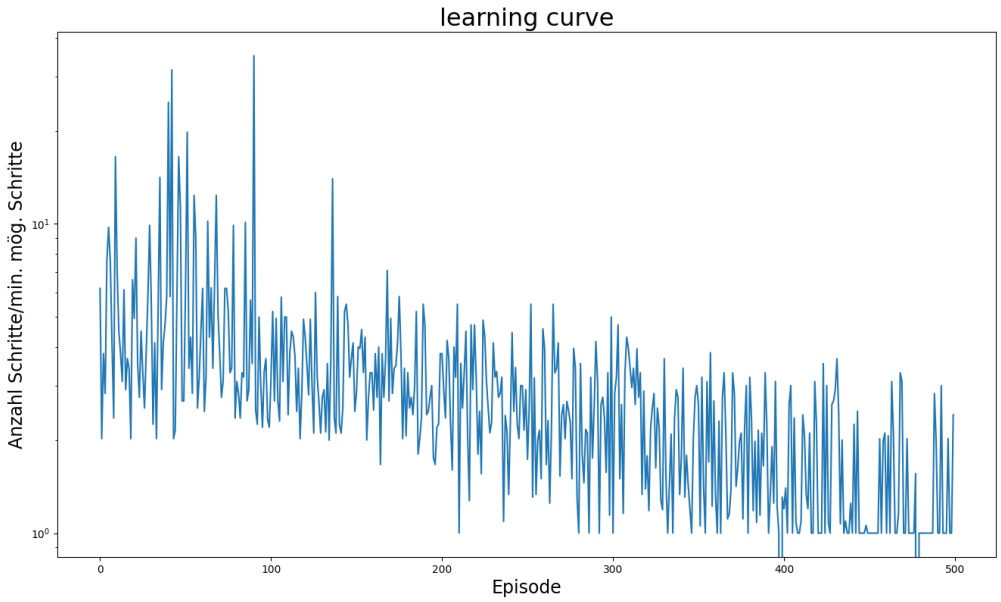


Abbildung 8: Lernkurve normiert und mit α=0.999999 und 500 Episoden

In Abb. 8 ist zu erkennen, dass bei 500 Episoden der Algorithmus schlechter konvergiert. Allerdings ist es fraglich, ob die Anzahl der Episoden nicht zu gering ist und die Wahl von α hier einen geringeren Einfluss hat. Daher wurde in Abb. 9 $\alpha = 0.01$ mit 500 Episoden untersucht.

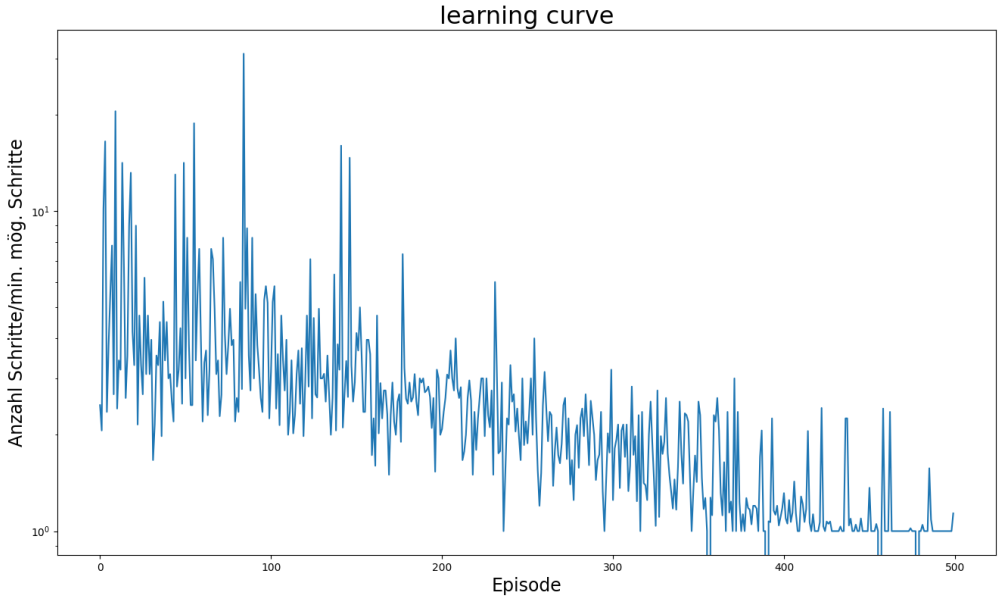


Abbildung 9: Lernkurve normiert und mit α=0.01 und 500 Episoden

In Abb. 9 ist zu erkennen, dass der Lernprozess zwar besser konvergiert im Gegensatz zu der Wahl von $\alpha = 0.999999$, aber auch nicht sicher gegen 1 für

Episode >= agent.zero_fraction∗agent.N_episodes = 450.

Wie bereits in der Aufgabenstellen beschrieben, kann dieses Problem an der Einfachheit der Simulation liegen.

Visualisierung der Zeitentwicklung von Q-Werten

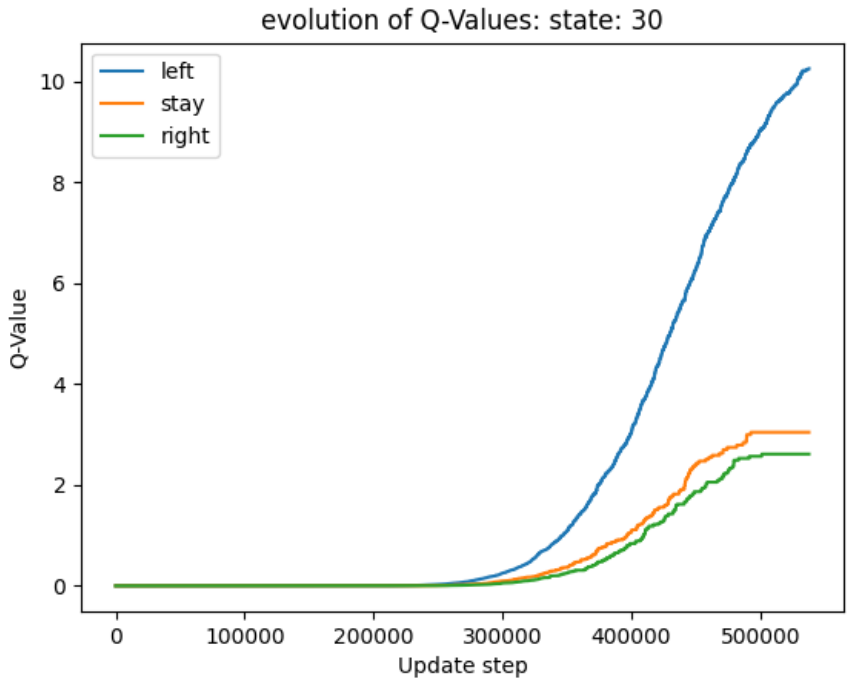


Abbildung 10: Entwicklung ausgewählter Werte aus Q-Matrix

In Abb. 10 ist die Zeitentwicklung der Q-Werte für den Zustand 30 zu sehen. Es ist klar zu erkennen, dass die Aktion „left“ den höchsten Wert hat. Grund dafür ist natürlich, dass das Ziel in Zustand 8 liegt und bei einer Zahlenstrahllänge von 100 ist der kürzeste Weg von Zustand 30 zum Ziel in Richtung links. Auf der x-Achse sind hier die insgesamt ausgeführten Schritte gewählt. Zu beachten ist hierbei, dass in den Episoden zu Beginn der Simulation noch mehr Schritte ausgeführt werden als zum Ende hin, denn dort hat der Agent bereits gelernt, wo das Ziel ist. Für *Episode >= agent.zero_fraction∗agent.N_episodes* wächst außerdem nur noch der „left“-Wert, weil nur noch Entscheidungen auf Basis der Q-matrix getroffen werden und somit immer die Aktion, nach links zu gehen, gewählt wird.

3.4 Stochastisches Hindernis

In diesem Abschnitt wurde abschließen untersucht, ob der Algorithmus auch mit einem eingebauten Hindernis sinnvoll konvergiert.

10.

In dieser Aufgabe hat der Zahlenstrahl eine Länge von 15 (Zahlen von 0 bis 14). Der Startpunkt ist bei $x_0 = 8$ und das Ziel ist bei $x_z = 12$. Da aber ein stochastisches Hindernis in dieser

Aufgabe auf dem Intervall $obstacle\_intervall = [9{,}12) = [9{,}11]$ existiert, ist für den Agent der einfachste Weg nicht einfach 4 Schritte nach rechts zu machen, denn dort befindet sich das Hindernis. Wenn sich der Agent auf einer Position innerhalb des Intervalls $obstacle\_intervall$ befindet, gibt es eine Wahrscheinlichkeit P_0, dass der Agent einen Schritt nach links macht. Das gilt also für 3 Positionen auf dem Zahlenstrahl.

Es bestehen weiterhin periodische Randbedingungen, demnach ist der Weg nach links bis zum Ziel 11 Schritte lang.

Es wird nun der Punkt gesucht, an dem die Wege zu beiden Seiten gleich lange dauern. Denn dort sollte (in etwa) der Wert κ von 1 auf -1 wechseln.

Die 3 Schritte auf dem Intervall $obstacle\_intervall$ müssen demnach „länger dauern“ als Schritte außerhalb des Intervalls.

Für ein Gleichgewicht der Wege über Links und Rechts gilt:

$$11\ Schritte\ links\ lang \equiv 3\ Schritte\ auf obstacle\ intervall + 1\ Schritt\ (von\ 11\ auf\ 12)$$

Führe Faktor m ein für das Gewicht der Schritte auf $obstacle\_intervall$:

$$\rightarrow 11 = 3 * m + 1$$

$$\rightarrow m = 4$$

Also müssen Schritte auf $obstacle\_intervall$ 4-mal „länger dauern“. Das Entspricht einer Wahrscheinlichkeit von $P_{stehen} = {}^{1}/_{m} = 1/4$, dass der Agent auf $obstacle\_intervall$ auf der Position stehen bleibt, nachdem er einen Schritt nach rechts gemacht hat. Das Stehenbleiben bezieht sich auf die Funktion *stoch_obstacle*, in welcher der Agent, nachdem er einen Schritt auf $obstacle\_intervall$ gemacht hat, durch Zufall entweder stehen bleibt oder wieder einen Schritt nach links macht. Diese Funktion ist das stochastische Hindernis in dieser Aufgabe.

Also wird ein Wert von $P_0 = 1 - P_{stehen} = 3/4$ für die Wahrscheinlichkeit, dass der Agent auf $obstacle\_intervall$ wieder einen Schritt nach links macht, nachdem er einen Schritt nach rechts gemacht hat (in Bezug auf die Funktion *stoch_obstacle*), erwartet.

13.

Der Wert κ beschreibt im Folgenden die Richtung, für die sich der Agent entscheidet zum Ziel zu laufen, nachdem $\epsilon = 0$ erreicht wurde. Dass bedeutet ab dem Zeitpunkt, ab welchem der Agent nur noch Entscheidungen auf Basis der Q-Matrix trifft. Dabei entspricht $\kappa = -1$ der Richtung links und $\kappa = 1$ der Richtung rechts.

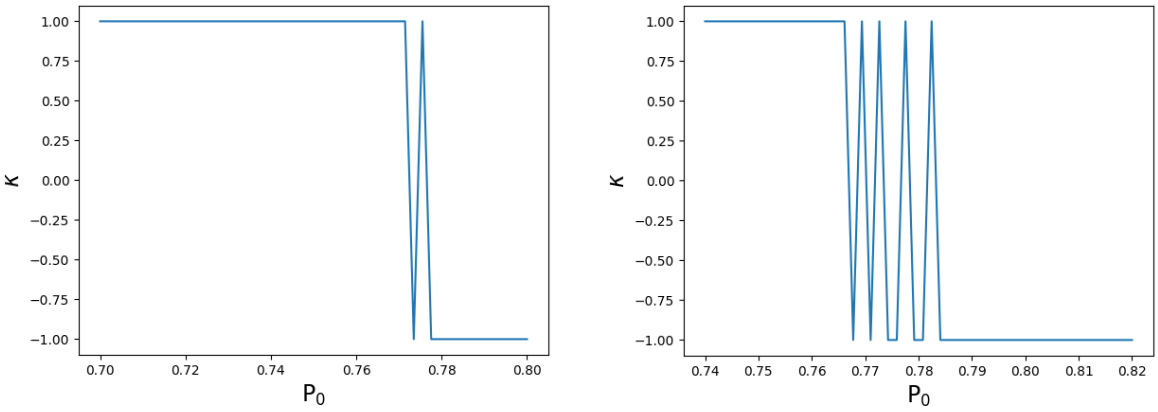


Abbildung 11: κ aufgetragen gegen einige Werte von P_0 in zwei verschiedenen Durchläufen der Simulation *(hier 50 verschiedene Werte von P_0 auf gegebenem Intervall)*

In Abb. 11 ist zu sehen, dass der Übergangen von $\kappa = 1$ auf $\kappa = -1$, aus der numerischen Auswertung, nicht genau bei dem erwarteten Wert von $P_0 = 0.75$ ist. Der Wert für den Übergang liegt demnach etwa zwischen $0.77 < P_0 < 0.78$.

Gezeigt sind außerdem zwei verschiedene Durchläufe der Simulation. Der Übergang ist bei dem rechten Diagramm nicht so klar wie bei dem linken, der Grund dafür sind statistische Fluktuationen.

14.

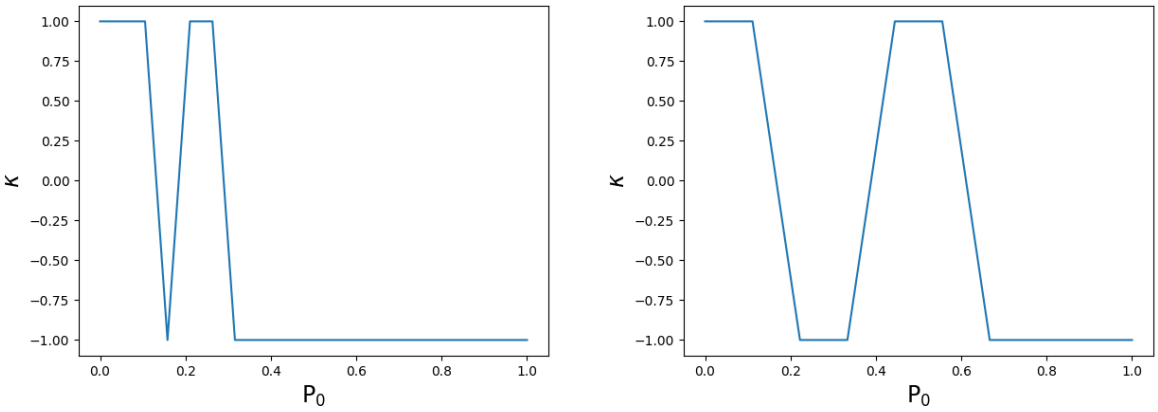


Abbildung 12: κ aufgetragen gegen einige Werte von P_0 mit $\alpha = 0.999999$ in zwei verschiedenen Durchläufen der Simulation

In Abb. 12 wurde der Übergang von κ untersucht, wobei $\alpha = 0.999999$ gewählt wurde. Es ist in beiden Durchläufen klar zu erkennen, dass der Algorithmus gegen eine schlechte Strategie konvergiert. Denn das Intervall für den Übergang ist in beiden Fällen deutlich breiter als zuvor (links etwa $0.1 < P_0 < 0.3$ und rechts etwa $0.1 < P_0 < 0.7$) und der erwartete Wert von $P_0 = 0.75$ sowie die Werte aus 13. liegen in keinem der beiden Intervalle aus dieser Aufgabe.