Reinforced learning für intelligente Brownsche Dynamik

Einleitung:

In diesem Projekt wurden die Grundlagen des Reinforced learnings anhand des Q-learning Algorithmus behandelt. Das Q steht hier für eine Matrix nach dem der „Agent“ handeln soll. Das Ziel war es, einem intelligenten Agenten, hier ein Hund, den schnellsten Weg zu seiner Belohnung, einem Knochen, selbstständig finden zu lassen. Da es sich nur um ein recht kleines Projekt handelt, wurde hier nur der eindimensionale Fall betrachtet. Es war außerdem nicht kontinuierlich, d.h. es gab begrenzte Felder (zunächst 100) und es galten periodische Randbedingungen. Falls der Hund links rauslief, kam er rechts wieder rein. Der Hund hatte „gewonnen“ , wenn er auf das Feld mit dem Knochen kam und als nächste Aktion „nicht bewegen“ wählte.

Theorie des Lernprozesses:

Die Matrix hatte die Form (Anzahl der Felder) *(Anzahl Aktionen), hier meist 100*3. Die drei Aktionen sind hier 0: nach links bewegen, 1: auf dem Feld bleiben, 3: nach rechts bewegen. Es gibt zwei Arten, wie der Hund sich bewegen konnte:

- den „Randomwalk“, hier bewegt der Hund sich mit einer gewissen Wahrscheinlichkeit nach links oder rechts und soll so die Diffusion simulieren
- die andere Art ist „Chosenaction“, hier sucht sich der Hund seine Aktion selber anhand der Matrix aus, in dem der maximale Zeilenwert des Feldes genommen wird auf dem er sich befindet.

Nach jedem Schritt wird die Matrix durch die nachfolgende Formel aktualisiert:

$$Q_{ij}^{new} = Q_{ij}^{old} + \ \alpha(R + \gamma * \max(Q'_{ij}) - Q_{ij}^{old})$$

i' bezieht sich hier auf das Feld auf welches der Hund sich hin bewegt hat.

R ist hier die Belohnung, die erhalten wird, wenn der Hund auf dem Belohnungsfeld ist und dort bleibt.

α ist die „learningrate“, sie bestimmt, wie schnell der Lernprozess ist, darf jedoch nicht zu groß sein damit der Algorithmus noch funktioniert.

γ ist der „discount factor“ und spielt bei diesem Problem keine große Rolle, da es nur ein Belohnungsfeld gibt.

Umsetzung:

Zunächst wurde die Diffusion des Teilchens simuliert, hierzu wurde eine Funktion namens randomstep benutzt. Diese macht einen Schritt nach links oder rechts mit einer Wahrscheinlichkeit in Abhängigkeit der Diffusionskonstante D.

Für die Diffusionskonstante gilt:

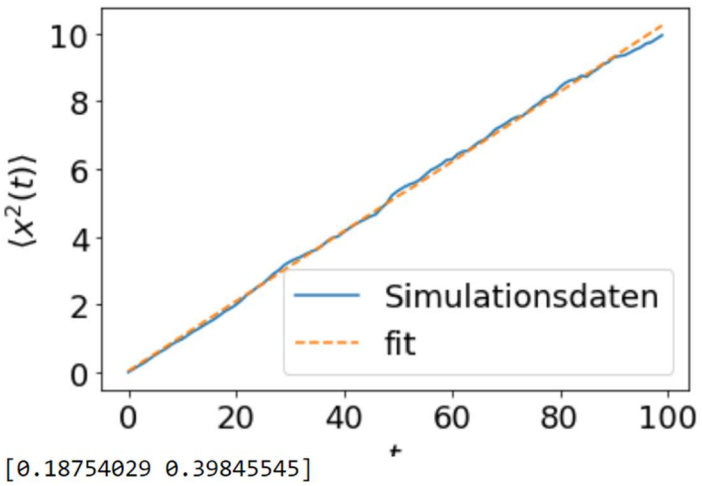
D=$\frac{a^2}{2\tau}$. a ist die Schrittweite (hier immer 1) und τ ist die Zeit zwischen zwei Schritten (hier auch immer 1). Somit ist D eigentlich festgelegt. Um D trotzdem variabel zu halten wird der Zufallsschritt nur mit der Wahrscheinlichkeit 2D ausgeführt, womit die effektive Zeit zwischen zwei Schritten entsprechend angepasst wird. Um zu testen, ob die Simulation gut gelingt, wurden mit unterschiedlichen Diffusionskonstanten die Diffusion simuliert und auf einem Graph aufgetragen.

Auf der y-Achse ist $<\triangle x^2>$ aufgetragen und auf der x-Achse ist t aufgetragen da D=$\frac{a^2}{2\tau}$= $\frac{<\triangle x^2(t)>}{2\tau}$ und τ=1 gilt 2D=$<\triangle x^2>$ da $<\triangle x^2(t)>$ linear ist, ist D die Hälfte der Steigung.

Beispiele mit unterschiedlichem D:

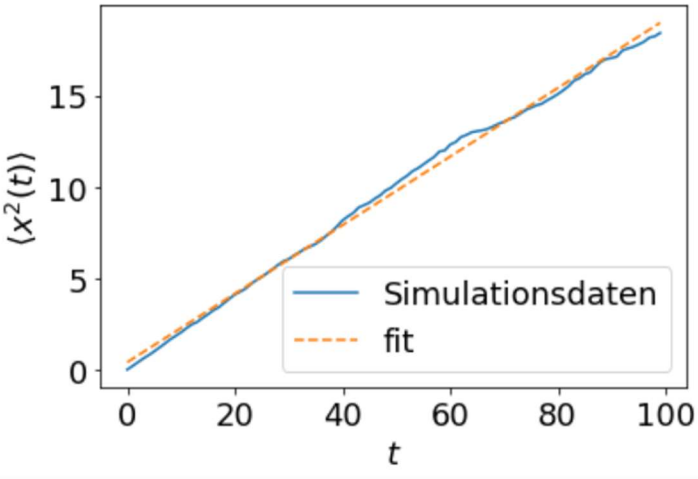
D = 0.05, gemessen 0,052:

```
[0.10309926 0.03212683]
```



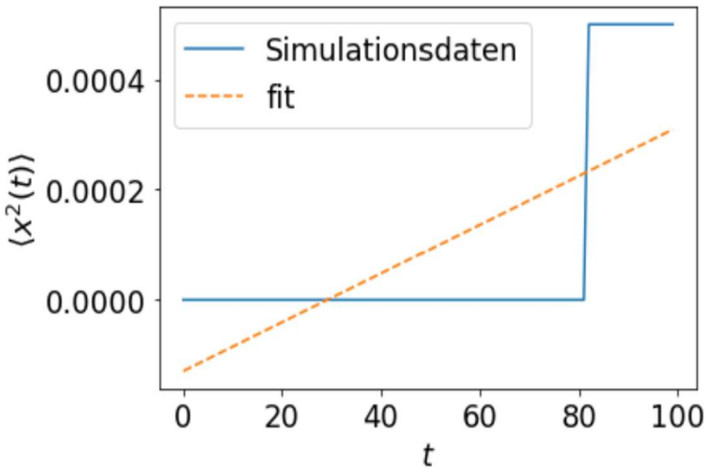

```
[0.18754029 0.39845545]
```

D= 0.1, gemessen 0,934:



*D= 5e-6, gemessen 2.21*10e-6:*

```
[ 4.42844284e-06 -1.29207921e-04]
```



Wie man leicht erkennen kann, funktioniert es mit sehr kleinem D nicht mehr, bei recht großem D ist auch eine größere Abweichung zu erkennen, aber es funktioniert noch recht gut. Man muss dazu anmerken, dass es statistische Abweichungen gibt, die hier nicht näher behandelt werden.

Als nächstes wurde der Lernalgorithmus implementiert. Dafür wurde zunächst eine Matrix initialisiert, deren Einträge am Anfang alle 0 waren. Anschließend wurde eine Funktion geschrieben, die die Matrix wie oben beschrieben nach jedem Schritt aktualisiert. Um den Hund zu trainieren, sollte er 10000 Episoden durchlaufen. Eine Episode fängt bei einem Startfeld an und endet auf dem Belohnungsfeld, wenn er sich entscheidet sich nicht mehr zu bewegen. Hier wurde $\alpha = 0{,}01\ und\ \gamma = 0.9$ gesetzt. Er startet auf Feld 30 und das Belohnungsfeld ist auf Feld 8. Am Ende der 10000 Episoden, hatte sich die Matrix so angepasst, dass links vom Belohnungsfeld die Aktion nach rechts bevorzugt wurde, auf dem Belohnungsfeld wurde die Aktion daraufbleiben bevorzugt und rechts vom Belohnungsfeld wurde die Aktion nach links bevorzugt allerdings nur bis zum 59ten Feld. Ab hier wird aufgrund der periodischen Randbedingungen wieder nach rechts gegangen. Zum besseren Verständnis, warum der Algorithmus funktioniert, wird hier in einem vereinfachten Szenario ein Beispiel vorgerechnet. Es gibt nur vier Felder 0,1,2,3. Gestartet wird auf Feld 0. Eine Episode endet, sobald Feld 4 erreicht wird. α, γ und R sind hier beliebig. Die Option auf dem Feld zu bleiben, gibt es hier nicht. Die Schritte werden immer vorgegeben und sind nicht zufällig oder nach der Matrix ausgeführt.

Am Anfang sieht die Matrix so aus:

$$\begin{pmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{pmatrix}$$

Nach 3 Schritten nach rechts sieht die Matrix so aus

$$\begin{pmatrix} 0 & 0 \\ 0 & 0 \\ 0 & \alpha \cdot R \\ 0 & 0 \end{pmatrix}$$

Als nächstes werden die Schritte R, L, R, R, R ausgeführt. Die Matrix sieht dann so aus:

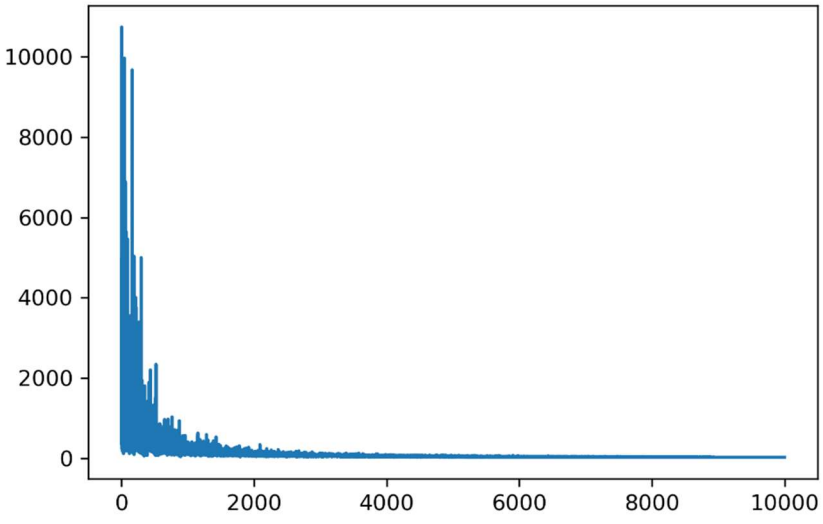
$$\begin{pmatrix} 0 & 0 \\ 0 & q^2 \gamma \cdot R \\ 0 & 2\alpha \cdot R - q^2 \cdot R \\ 0 & 0 \end{pmatrix}$$

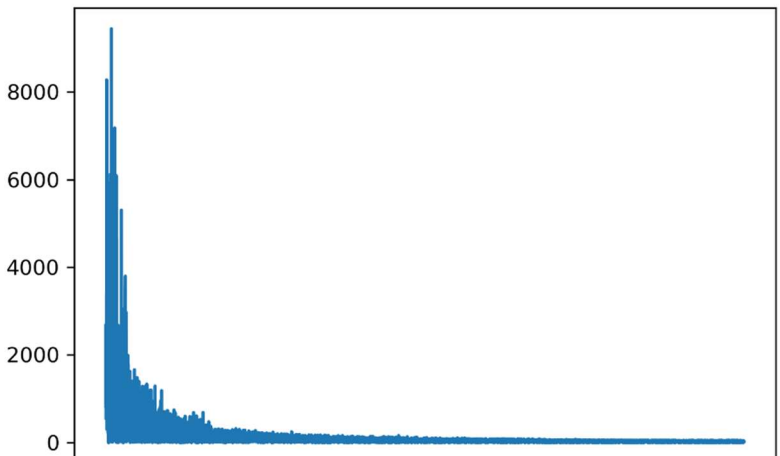
Wenn nun in der nächsten Episode ein Schritt nach recht gemacht wird, sieht die Matrix so aus:

$$\begin{pmatrix} 0 & \alpha^3\gamma^2 \cdot R \\ 0 & \alpha^2 \cdot \gamma \cdot R \\ 0 & 2\cdot\alpha\cdot R - \alpha^2 \cdot R \\ 0 & 0 \end{pmatrix}$$

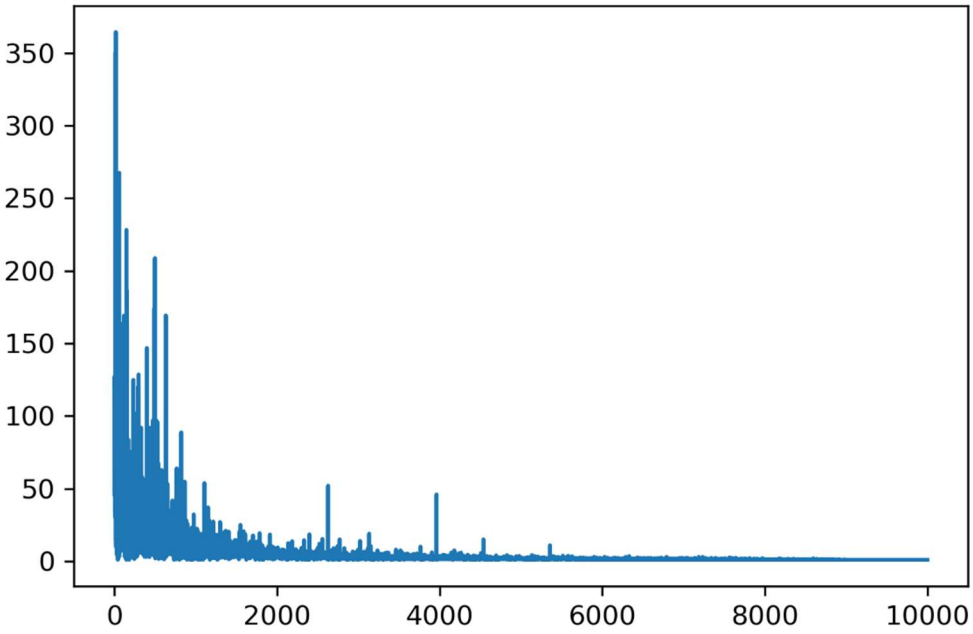
Angenommen alle Parameter sind größer 0 und $\alpha < 1$ dann lässt sich leicht erkennen, dass das Maximum einer Zeile immer rechts ist, also die Aktion nach rechts bevorzugt wird. Der Algorithmus läuft nun also von selbst nach rechts. Das R wir mit dem vor Faktor $\alpha^n \, \gamma^{n-1}$ weiter nach unten gegeben. Hierbei ist n die Entfernung zu dem Belohnungsfeld. Für $\alpha < 1$ und $\gamma < 1$ ist es nicht möglich, dass der Algorithmus lernt nach links zu laufen (Da es keine periodischen Randbedingungen gibt).

Im nächsten Schritt wurde das Problem verallgemeinert. Der Hund startet nun immer auf einem zufälligen Feld. Außerdem wird das Ergebnis auf einem Grafen aufgetragen auf der y-Achse wird gezeigt wie viele Schritte der Hund braucht und auf der x-Achse die Episode, in der er sich befindet. Das Ergebnis ist hier aufgetragen: (oben mit festem Anfangsfeld, unten mit zufälligem Anfangsfeld)



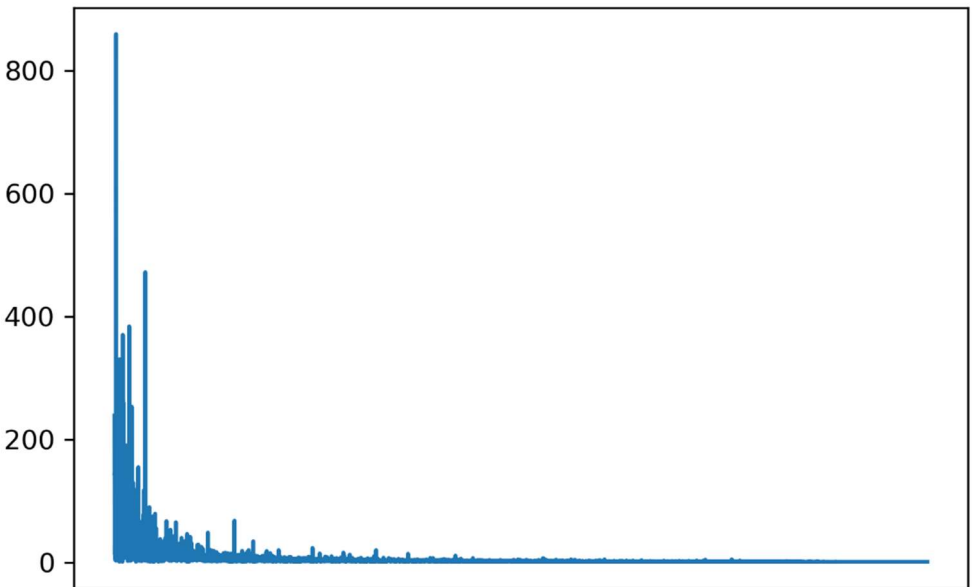


Wenn man genau hinguckt, kann man erkennen das nur in dem ersten Grafen die Schrittanzahl konvergiert. Das liegt daran, dass selbstverständlich der kürzeste Weg bei unterschiedlichen Startpositionen nicht gleich ist. Deshalb wird im nächsten Schritt der kürzeste Weg zunächst ausgerechnet und anschließend wird jede Episode auf diesen Weg normiert:

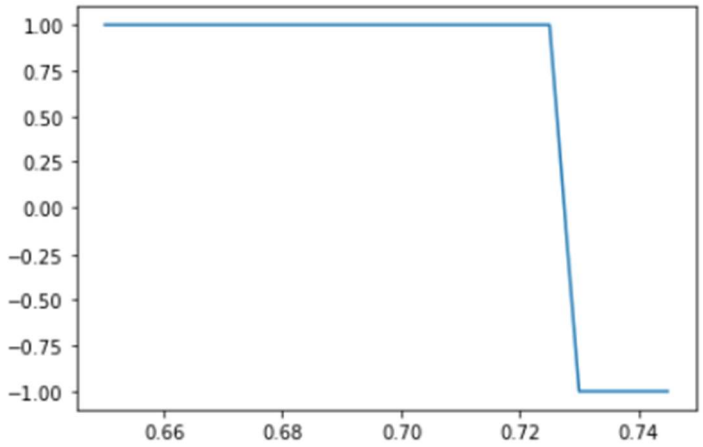


Hier kann man wieder erkennen das die Episodenzahl wieder konvergiert (gegen 1).

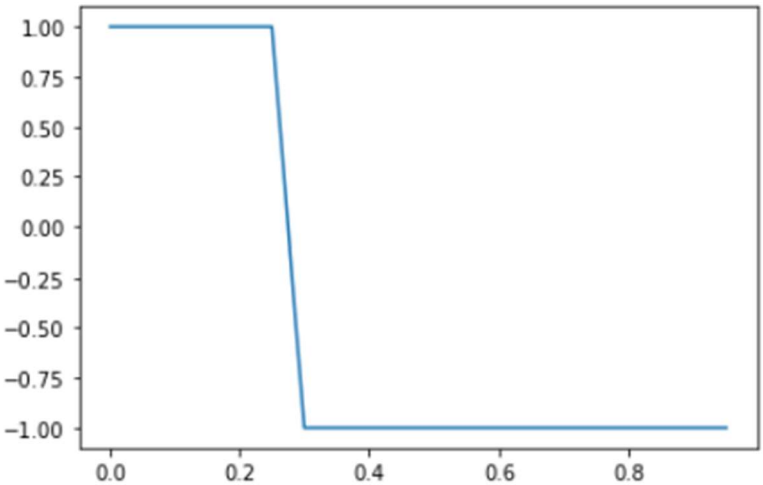
Wenn man ein sehr hohes α wählt, werden mehr Schritte benötigt, da falsche Schritte weniger „gestraft“ werden , es konvergiert allerdings trotzdem aufgrund der Einfachheit des Algorithmus.



Als nächstes wurde noch ein stochastisches Hindernis eingebaut. Um die Rechenzeit zu verringern, wurden hier nur 15 Felder statt 100 genommen. Das Startfeld wurde auf 8 gesetzt und das Belohnungsfeld auf 12. Auf den Feldern 9, 10,11 wurde das Hindernis gesetzt, hier wurde vor jedem Schritt mit einer Wahrscheinlichkeit von P ein Schritt nach links gemacht. Es wurde anschließend numerisch bestimmt bei welcher Größe von P nach rechts bzw. nach links gelaufen wird. Erwartet wurde das für P<0.7 nach rechts und sonst nach links gelaufen wird. Das Ergebnis war bei ca. 0.73. Auf der Y-Achse ist aufgetragen ob links oder rechtsrum gelaufen wurde, hier steht 1 für rechts und -1 für links:



Auch hier kann man erkennen, dass mit sehr großem α wieder Probleme auftreten wie in dem Grafen unten abgebildet:



Man kann klar erkennen, dass nach links gegangen wird lange bevor es sinnvoll ist.