SAKI SS19 Homework 4

Author: Jannis Wolf

Program code: https://github.com/JannisWolf/deep_q_learning_trader

Summary

In der 4. Aufgabe musste ein smarter Aktien Trader mit Hilfe von Deep Q Learning entwickelt werden. Der grundlegende Aufbau des Frameworks beinhaltet eine Börse an der zwei verschiedene Aktien gehandelt werden. Täglich erhält der Trader von der Börse sein Portfolio übergeben und kann eine Orderliste zurückgeben um Aktien zu kaufen und verkaufen. Da die Aktienkurse alleine nicht genügend Informationen enthalten, gibt für jede Aktie ein Experte eine Empfehlung über dessen zukünftigen Verlauf in der Form Aktien kaufen, verkaufen oder halten ab.

Der Deep Q Learning Algorithmus wurde in der durch das Interface ITrader vorgeschriebene *trade()* Methode implementiert, welche einmal pro virtuellen Tag von der Börse aufgerufen wird. Zunächst wird der 'tägliche' *State* definiert. Durch die *Q-Funktion* wird eine *Action* ausgewählt, dessen *Reward* am nächsten Börsentag ausgerechnet werden kann.

State S: Array (Länge 2) mit den Expertenmeinungen konvertiert in numerische Werte {'Buy': 1, 'Sell': 2, 'Hold': 3}.

Action A: Der Aktionsraum für die Aktien A und B hat die Größe 4 und wurde wie folgt definiert: {(Buy A, Buy B), (Buy A, Sell B), (Sell A, Buy B), (Sell A, Sell B)}, wobei immer mit allem verfügbaren Geld Aktien gekauft wurde, bzw. alle vorhandenen Aktien verkauft wurden.

Q-Funktion: Ein einfaches Neuronales Netz (NN) mit 3 Schichten approximiert die Q-Funktion. Zu einem Input-State wird ein Q-Wert für jede Aktion ausgegeben. Die Aktion mit dem höchsten Q-Wert wird ausgeführt.

Reward R: Das Design der Reward Funktion kann sehr verschieden gestaltet werden. Die Funktion R_1 erzielte sehr gute Ergebnisse:

$$R_1 = \begin{cases} 1 \text{ falls Portfoliowert}_{alt} \leqq \text{ Portfoliowert}_{neu} \\ 0 \text{ sonst} \end{cases}.$$

Um nicht durch die anfangs zufällige Initialisierung des neuronalen Netze in einem lokalen Minimum zu landen und das 'Exploration vs. Exploitation' Problem zu lösen, wird eine *Epsilon Greedy Policy* verfolgt. Anfangs wird mit einer Wahrscheinlichkeit Epsilon = 1 eine zufällige Aktion ausgeführt. Epsilon wird nach jedem Training des Neuronalen Netz sukzessive verringert. Somit wird langsam immer mehr die Policy des NN ausgeführt.

Das Training des neuronalen Netzes wurde mit Hilfe von Experience Replay implementiert. Jeden Tag wird ein Tupel (S_t, A, R, S_{t+1}) in einer begrenzten Warteschlange gespeichert. Bei genug Erinnerungen kann das neuronale Netz mit zufälligen Batches in jedem *trade()* Aufruf trainiert werden. Dabei werden die Q-Werte der nicht vollzogenen Actions gleich gehalten, während sich der Target Q-Wert der ausgeführten Action $Target_A$ durch folgende Formel berechnet:

$$Target_A = R + \gamma \cdot max_a Q(S_{n+1}).$$

Der Parameter Gamma steht für die Weitsichtigkeit des Traders. Je näher γ -> 0 gewählt wird, desto mehr versucht der Trader den Reward für den Tag zu maximieren, was in diesem Szenario gewünscht war.

Evaluation

Als Performance Metrik wurde der Portfoliowert nach dem 'traden' auf den ungesehenen Testdaten der Aktienkurse gewählt, welcher mit zwei anderen virtuellen Tradern verglichen werden kann. Diese folgen einem 'Buy-and-Hold'-Schema, bei dem nach einem ersten Aktienkauf diese immer gehalten werden und einem

‘Trusting’-Schema, bei welchem immer die Experten Empfehlung durchgeführt wird.

Abbildung 1 in den Screenshots zeigt den Portfoliowert der 3 Trader im Testzeitraum von 2012-01 bis 2016-01. Der mit Deep Q Learning trainierte Trader kann hier beide ‘herkömmlichen’ Trader deutlich überragen mit einem Portfoliowert von 67630 gegenüber 3828 des ‘Buy-and-Hold’-Traders und 22452 des ‘Trusting’-Traders. Die Parameterwahl ist äußerst einfach gehalten, mit 4 möglichen Aktionen, Gamma = 0, und der Reward Funktion R_1.

Der Deep Q Learning Algorithmus beinhaltet eine Fülle an Parametern welche optimiert werden können um die beste Performance zu erhalten. Es stellte sich heraus, dass komplexere Aktionsräume und Reward Funktionen zwar immer besser als der ‘Trusting Trader’ war, jedoch keine signifikante Verbesserung gegenüber den relativ trivialen Parametern darstellte. Die folgende Tabelle zeigt einige Konfigurationen mit Ergebnis.

#	Gamma	Reward Function	# Actions	# Episodes	Final Portfolio Value
1	0	R_1	4	5	67630
2	0	R_1	9	10	35072
3	0.1	R_1	4	10	63247
4	0.5	R_1	4	10	65676
5	0.9	R_1	4	10	60424
6	0	R_2	4	5	48951

Größere Aktionsräume (in Tabelle Zeile #2): Es macht Sinn, dass das einführen einer ‘Aktien halten’ Aktion keine Verbesserung bringen kann. Meint der Trader der Kurs wird stagnieren, kann er genauso gut seine Aktien verkaufen und später wieder hinzu kaufen. Das Ergebnis ist eine signifikant schlechtere Performance. Bei Einführung einer Handelsgebühr könnte diese Option wichtiger werden. Eine Anpassung des Gamma Wertes wäre dann auch nötig.

Verschiedene Gamma Werte (in Tabelle Zeilen #3-5): Der Gamma Wert von 0 sorgt dafür, dass der Trader den Reward kurzsichtig (hier für einen virtuellen Tag) maximieren will. Das ist sinnvoll an dieser Börse, an der ‘täglich’ einmal Aktien gekauft und verkauft werden können. Bei mehr Trainingsepisoden konnte auch bei höheren Gamma Werten der Deep Q-Algorithmus Strategien berechnen, die ähnlich gute Ergebnisse lieferten wie im $\gamma = 0$ Fall.

Komplexere Reward Funktionen (in Tabelle Zeile #6): Die intuitive Idee in die Reward Funktion die relative Änderung der Portfoliowerte einzubringen, wurde mit der Funktion R_2 getestet:

$$R_2 = \frac{Aktueller\ Wert - Letzter\ Wert}{Letzter\ Wert}.$$

Eine naheliegende Erklärung wieso kein besseres Ergebnis erzielt wurde ist, dass schlechte Aktionen anfangs häufig vorkommen und in der langen Trainingszeit schnell durch niedrige Rewards wegtrainiert werden, während es nun bei gute Aktionen nicht mehr auf einen genauen Reward Wert ankommt um den höchsten Q Value bei so wenig möglichen Aktionen zu approximieren.

Fazit: Wie eben dargelegt kann der Deep Q Learning Algorithmus in dem gewählten Setting schon mit trivial gewählten Parametern eine sehr gute Performance bringen. Während im Supervised Learning die Affinität Neuronaler Netze das Rauschen von Datensätzen zu lernen unerwünscht ist, kann diese Charakteristik hier genutzt werden um die virtuell gestörte Funktion der Experten zu approximieren. Anhand der guten Performance des ‘Trusting Traders’ ist ersichtlich, dass die Expertenmeinung nur kleinere Fehler in der Vorhersage enthält. Das Neuronale Netz im Deep Reinforcement Learning kann diese gut erkennen, wodurch ein smartes Handeln des Traders möglich ist.

Screenshot

Abbildung 1: