Deep Reinforcement Learning in Trading

Ashwini Patil, Saeed Rahman, Padmanabha Guddeti

Advisor: Dr. Khaldoun Khashanah

Abstract

Can a self-learning agent take on the role of a human trader? Can Reinforcement Learning (RL) effectively trade to maximize reward? In this paper, we address this challenge using Deep Q-Network (DQN), Double DQN (DDQN) and Dueling Double DQN (DDDQN) agents in a simulated trading environment. Our model is inspired by off-policy learning and its application in video games. We trained and tested these agents with all S&P 500 stocks and show that just by using price and volume information deep-reinforcement learning agent can make money. Moreover, we also introduce this new workflow in algo-trading strategy development by incorporating the risk modelling and strategy optimization into the reward engineering of the agent.

Key Words

Deep Q-Network (DQN), Double Deep Q-Network (DDQN), Dueling Deep Q-Network (DDDQN), Deep Reinforcement Learning, Q-Learning, Finance, Trading, Neural Network, Technical Analysis, ADX, RSI, CCI

Introduction

Trading using intelligent agents has been widely researched since 1990s (Moody and Wu, Optimization of trading systems and portfolios 1997). When trading, investor's goal is to optimize reward, usually profits. Online trading is usually depicted as a two-step decision-making process: 1 Analyzing Market Condition & 2 Taking Optimal Action. In this report, we describe Deep Learning (DL) methods as a solution to automate this online trading process. We present an algorithm called Deep Reinforcement Learning (DRL) where the agent learns the environment, improves itself based on the rewards earned and takes correct action (makes trades) simultaneously. With the success of Deep Q-Network (DQN) in Atari games (Mnih, Kavukcuoglu and David, et al. 2015) as an example, we applied DRL in stock markets to train a single stock trading agent with the goal of maximizing income in short and long term. To imitate a human trader, we also introduced technical indicators and unrealized returns as part of input states to the agent. We compared the DQN algorithm with Double DQN and Dueling Double DQN to find the best agent to maximize profits of the investment strategy.

Literature Review

Machine learning in trading started with macroeconomic models to predict market prices (Meese and Kenneth, The out of sample failure of empirical exchange rate models 1997). However, these models usually fail when trying to predict the prices short term (Meese and Kenneth, Empirical exchange rate models of the seventies: Do they fit out of sample? 1983). But recent research has focused on Reinforcement Learning methods. Q-learning method has been used to estimate discounted future rewards with higher profit earnings as compared to supervised learning models (Moody and Matthew, Reinforcement learning for trading 1999). To optimize trade execution, (Nevmyvaka, Feng and Kearns 2006) considered variables pertaining to the investor's strategy and situation as well as the market condition to develop state-based strategies. This resulted in large performance gains in limit order markets. Furthermore, studies have also used least-squares temporal difference to perform generalized policy iteration (Cumming, Alrajeh and Dickens 2015) to apply RL to algorithmic trading problems. Another approach incorporates a convolutional neural network trained with a variant of Q-learning into policy learning (Mnih, Kavukcuoglu and Silver, et al. 2013).

Technical analysis methods have also been extensively researched to predict market movements. Technical traders follow rules when using technical indicators because these indicators have shown predictive ability (Allen and Karjalainenb 1999) (Neely and Weller 2003). Technical Analysis has become a dominant forecasting method and is the area we have concentrated.

Deep Q-Network (DQN) although extensively tested on real world problems and video games has not been widely researched on algorithmic trading thus our motivation for the project. However, DQN is known to overestimate action values (Hasselt, Double Q-learning 2010) due to noise and function approximation. To solve this problem, we also implemented the Double DQN algorithm (Hasselt, Guez and Silver, Deep Reinforcement Learning with Double Q-Learning 2016). We also implemented Dueling Double DQN algorithm (Wang, et al. 2015) for better approximation of the state values and in turn improves learning and performance of the DRL system.

Methodology

Reinforcement Learning

Reinforcement Learning (Sutton and Barto 1998) is a self-taught architype (Barto and Sutton 1998) which was developed to solve the issue faced due to Markov decision (Puterman 2014). Actor-based reinforcement learning (also known as Q-learning) is more suitable for trading because it has flexible objective for optimization and can replicate the continuous nature of market condition (Moody and Saffell, Learning to trade via direct reinforcement 2001) (Deng, et al. 2015). However, deep learning (Bengio 2009) is an evolving new method which allows feature learning from big data. Deep Learning has been used successfully in projects such as speech recognition (Graves, Mohamed and Hinton 2013) and image categorization (Lee, et al. 2009).

For our project, we use Deep Learning with Q-learning to design a trading agent that replicates a human trader. Figure 1 and 2 gives an insight into the process flow of our DRL architecture. We first input the Open-High-Low-Close (OHLC) stock data into the trading environment via the data generator module. Also, indicator values are calculated in the data generator which are then fed to the Agent. Part of this data is used in the training phase where the agent learns which action to take based on the reward it earns by interacting with the environment. Then the agent is tested on out of sample data.

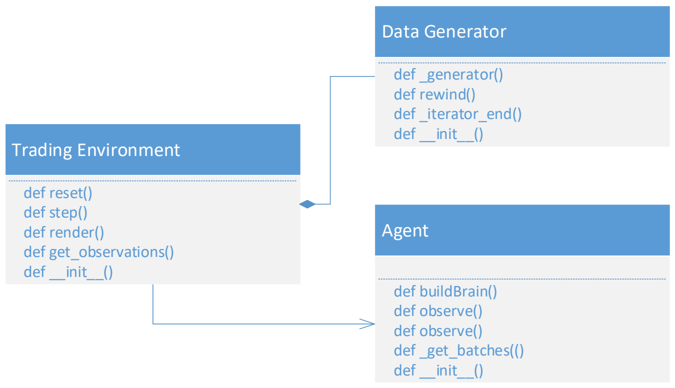

Figure 1: RL Architecture

Trading Environment

Inspired by OpenAI gym framework for video games, we created an environment where the agent can decide when to long or short a single stock. We allowed the agent to have the freedom to make the decision whenever it wants except that it can only hold one position at a time. For instance, if it has a long position, it cannot have a short position simultaneously. It needs to close its long position and then enter a short position. Furthermore, we also added in a flat position in the architecture where the agent can just observe the market without having a long or short position.

We implemented a deep artificial neural network that represents the agent's brain. To stabilize the reinforcement learning process and to train the neural network with more information at training period, we implemented experience replay – a biologically inspired mechanism (Mnih, Kavukcuoglu and David, et al. 2015). To perform experience replay, the agent's experience while training is stored at each time step in the agent's memory. At each update of the neural network weights, Q-value updates are applied on randomly drawn sample of these experiences.

State-Action-Reward Structure

State

We started with giving the Agent the following state:

$$[price_{(t-1)}, price_{(t)}, position_{price}, position] \qquad (a)$$

$t = time$
$positionprice =$
$The\ price\ at\ which\ the\ agent\ has\ entered\ the\ position$
$position = long, short\ or\ flat$

However, when we ran the algorithm, we realized that the agent could not learn a lot of information from just two data price points. When a trader looks at price chart, they look at 100-1000 price points before taking a position. Inspired by (Allen and Karjalainenb 1999) (Neely and Weller 2003)

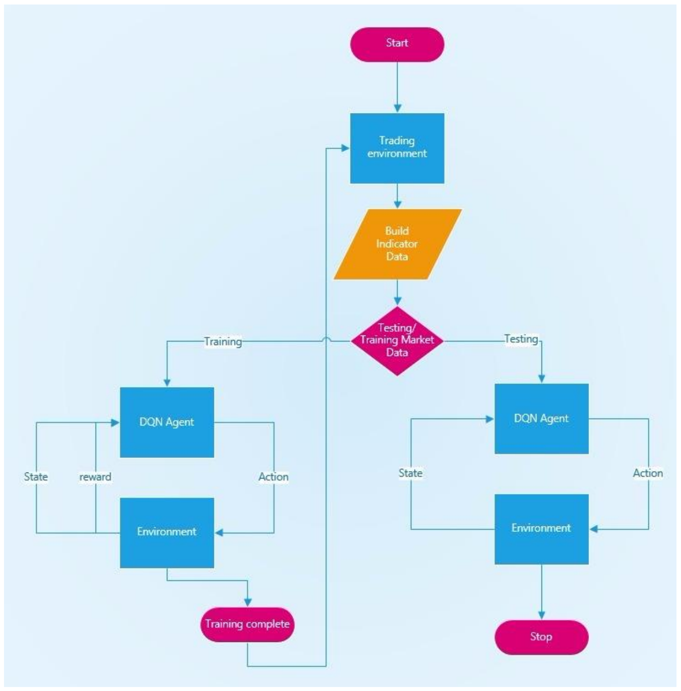

Figure 2: Process Flow

which showed that technical indicators have predictive ability, we decided to change our state to the following:

$$[ADX_{(t)}, RSI_{(t)}, CCI_{(t)}, position] \qquad (b)$$

$ADX = Average\ Directional\ Index$
$RSI = Relative\ Strength\ Index$
$CCI = Commodity\ Channel\ Index$

ADX indicator is used to determine whether the market is trending or ranging. It also informs how strong the trend is based on its value. RSI indicator is a momentum indicator used to determine the speed and changes in price movements. We also included CCI, another momentum indicator, because along with the price movements, it helps to determine when change in the price may be coming. All these indicators were normalized between -1 to +1 because the Neural Network requires all features to be stationary and within common range. When traders look at technical indicators, they do not look at the past values of the indicators but at the current frame. Thus, we believed by giving the agent indicator data, it could glean a better understanding of the market status and take more informed decisions. Although, this helped improve the results, the agent is only getting a reward when exiting a position. However, in real world a trader looks at the unrealized Profit and Loss (PnL) before closing their positions. To imitate this, we also included unrealized returns as one of the state factors:

$$[ADX_{(t)}, RSI_{(t)}, CCI_{(t)}, position, unrealized\ return] \quad (c)$$

In addition, we have also added volume one of the state variable making the state:

$$[ADX_t, RSI_t, CCI_t, Vol_t, position, unrealized\ return]\ (d)$$

This made the trading environment as close to a trading system which traders currently utilize.

Action

The agent could take three actions – Buy, Sell or Hold

Reward

The reward objective set is to maximize PnL from a position. It also includes the Trading Fee required for each transaction. This is like the commission we pay our brokers when we initiate a trade. We also included a Holding Fee which is like the interest we pay our brokers when we hold an open position. These rewards are what controls and optimize the agents during the training phase that determines the trading behavior.

Agents

The goal of reinforcement learning is to find a series of actions from certain state which lead to reward (Sutton and Barto 1998). Thus, an agent in state *s* chooses from a set of actions one action, *a* and receives a reward, *r*. The agent then comes to a new state *s'*. This process of choosing an action is called policy. Following, we see the three different agents we deployed in the above-mentioned trading environment.

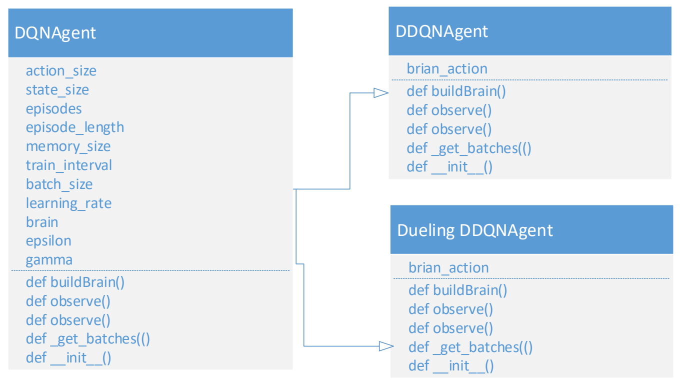

Figure 1: Agents Class Diagram

1. Deep Q-Network (DQN)

   We define Q(s, a) function such that for given state s and action a, it returns an estimate of a total reward we can achieve should we start at this state. Using the well-known Bellman equation (3) as introduced by (Watkins 1989), a DQN agent chooses an action according to the greedy policy – maximizing the Q* function.

$$Q^*(s_t, a) \rightarrow r(s_t, a_t) + \gamma max_a Q^*(s_{t+1}, a) \qquad (1)$$

$Q^* = Optimal\ Q\ Value$

$\gamma = discount\ factor$

$s = state, a = action, r = reward, t = time$

This equation converges to desired *Q** given that there are finite number of states and each state-action pair is presented repeatedly. However, this is not possible for market conditions. All state-action pair may not be presented repeatedly or even once in the training section. Thus, the Q function is generalized and approximated with several neural networks – Deep Q-Network (DQN). However, using neural network to represent Q function is unstable (Mnih, Kavukcuoglu and David, et al. 2015). Thus, we use Experience Replay (Lin 1992) to stabilize the training. Here, a gradient descend step is performed with each memory held by the agent. Thus, more and more truth is introduced into the system and causes the system to converge.

2. Double DQN

   One problem faced with DQN agent is that the agent tends to overestimate the Q function (Hasselt, Double Q-learning 2010) due to the max in the formula (1). When estimating a Q function for a certain state, the estimate is noisy and differs from the true value. Due to the max function, the action with the highest positive error is selected and propagated to other states. This leads to positive bias overestimation. To solve this issue, we introduced a Double DQN (DDQN) agent. In DDQN, two separate Q functions are independently learned. One network is used to determine the maximizing action while the other estimates its value. By decoupling the maximizing action from its value, we can eliminate the maximization bias. The change in estimation is as follows:

$$Q^*(s_t, a) \rightarrow r(s_t, a_t) + \gamma Q'^*(s'_{t+1}, argmax_a Q^*(s'_{t+1}, a')) \qquad (2)$$

$Q^* = Optimal\ Q\ Value$

$\gamma = discount\ factor$

$s = state, a = action, r = reward, t = time, Q'\ \&\ s'\ \&\ a' = target\ network\ Q - value, state\ and\ action$

This helps improve stability of the learning process to learn complicated tasks which in turn translates to improvement in performance (Hasselt, Guez and Silver, Deep Reinforcement Learning with Double Q-Learning 2016).

3. Dueling Double DQN

To further improve the learning process and performance, we also compared our results with a Dueling Double DQN (DDDQN) agent. A DDDQN agent presents a change in the network structure comparing to DQN as follows:

$$Q^*(s_t, a; \theta, \beta, \alpha) = \hat{V}(s_t; \theta, \beta) + \left(\hat{A}(s_t, a; \theta, \alpha) - \frac{1}{|\hat{A}|} \sum_{a'} \hat{A}\,(s_t, a'; \theta, \alpha) \right) \quad (3)$$

$Q^* = Optimal\ Q\ Value$

$\gamma = discount\ factor$

$\hat{V} = Value\ network, \beta = parameter\ specific\ to\ Value\ network$

$\hat{A} = Advantage\ network, \alpha = parameter\ specific\ to\ Value\ network$

$s = state, a = action, r = reward, t = time, Q'\ \&\ s'\ \&\ a' = target\ network\ Q - value, state\ and\ action$

By separating the Q Head into an advantage stream and Value stream, the agent is better able to differentiate actions from one another. This significantly improves learning. Furthermore, in DQN, on each iteration, for each state in the batch, we update the Q-Values only for the action taken in the state. This results in slower learning since Q-values for actions which are not taken are not learned. On the dueling architecture, learning is faster as we start learning the state-value even if just a single action has been taken.

Dueling architecture proves useful in a trading system where the value stream learns to pay attention to the indicator data while the advantage stream learns to pay attention only when there is an action to be taken to reap rewards. This is useful when the market is trending as we want the agent to ignore any price movements during the trend and only exit its position when the trend is over.

Data

We have used two data-set. One set comprising End of Day (EOD) information two stocks [Symbols-AAPL, XOM] from January 2010 – April 2018 to find the best agent (out of DQN, DDQN and DDDQN).

The second dataset comprises the End of the Day (Open, High, Low, Close, Volume) information of all S&P 500 stocks from February 2013 to February 2018 to test the performance of best agent across all the stocks.

Experimental Setup

The training was done on 75 percentage and tested the performance of the agent in the 25 percentage out of sample data. For example, for the first data set we trained the agents on data from 2010 to 2016 and tested the performance on data from 2016 to 2018 data.

No hyperparameter optimization was done on the agent's or the environment parameters.

These are the list of parameters:

- Training Epochs -100 & 150 (in DDDQN for comparing agents)
- validation_split = 0.1
- batch_size = 64
- train_interval = 10
- gamma = 0.96
- learning_rate = 0.001
- memory_size = 3000
- trading_fee = 0.0001
- time_fee = 0.001

Results

We tested the agents in scenario where the agents see the states as in (c)

$$[ADX_{(t)}, RSI_{(t)}, CCI_{(t)}, position, unrealized\ return]$$

As a benchmark for comparing different agents, we use a random agent which takes an action randomly in the same environment. The table below gives the trade statistics of the different agents:

Agent	Reward (Total PnL)	Risk Adjusted Return	Avg Holding Period
Random	-81	-0.76	3
DQN (AAPL)	42	0.8	36
Double DQN(AAPL)	69	1.02	35
Dueling-Double DQN(AAPL)	100.51	1.21	16
Dueling-Double DQN(AAPL)	29	1	24

Table 1: Comparing Agents' trade statistics

In machine learning, a cost/loss function measures the model performance. We used training loss to measure the performance of the model. Training loss measures the error on the training data which we want to minimize for optimization. Validation loss measures the loss on a subset of the training data which the model has not used. We want to make the gap between the training testing loss small. An over-fitting model is when the gap between the two is large while an under-fitting model is when the model cannot achieve a low training error (Li 2017).

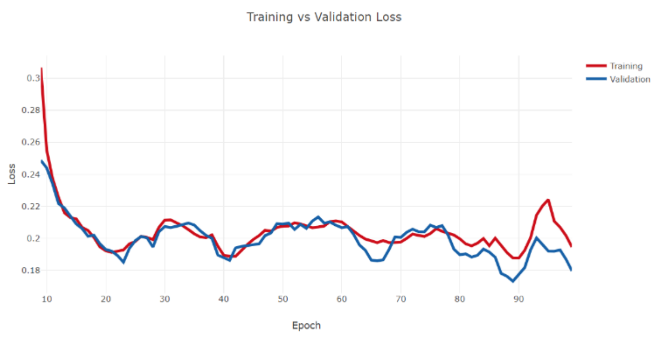

Figure 2: DQN – Loss vs Epoch

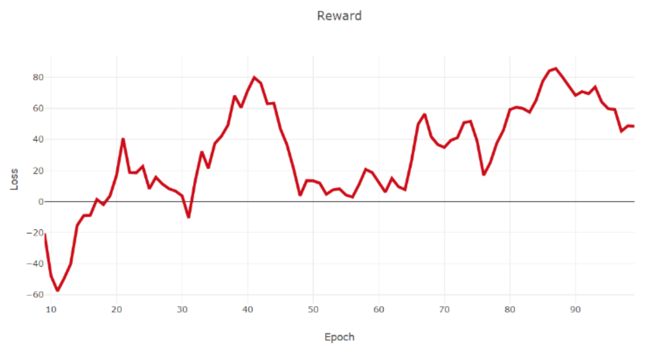

Figure 3: DQN - Reward vs Epoch

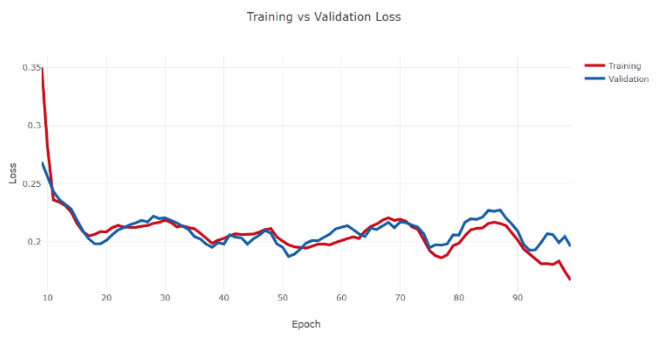

Figure 4: Double DQN – Loss vs Epoch

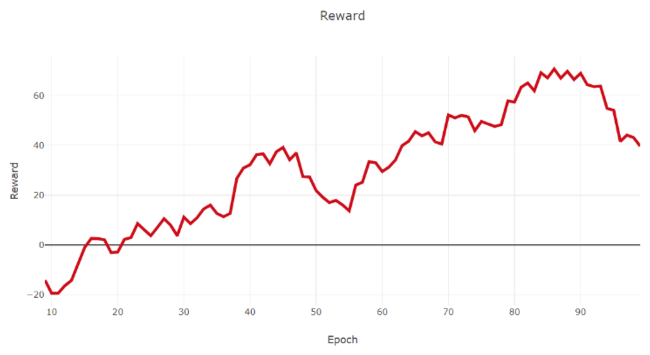

Figure 5: Double DQN - Reward vs Epoch

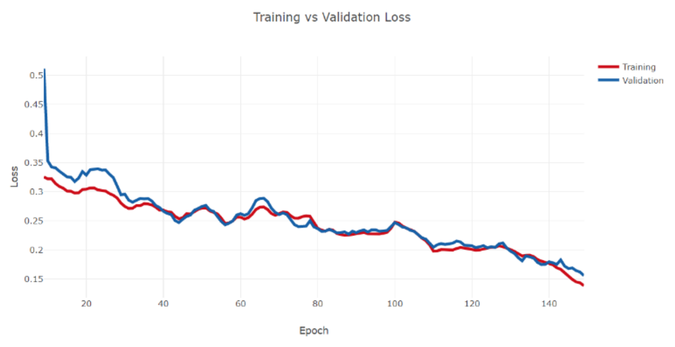

Figure 6: Dueling Double DQN– Loss vs Epoch

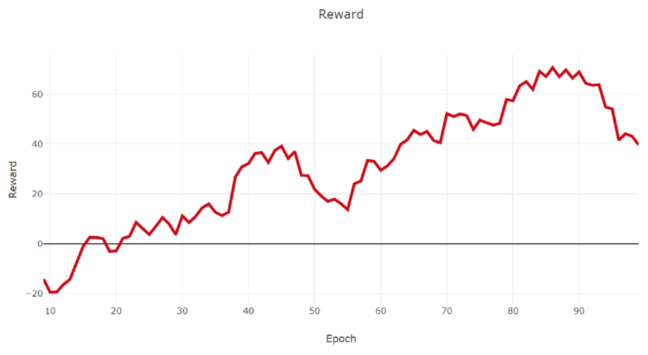

Figure 7: Dueling Double DQN - Reward vs Epoch

Comparing our charts, we can see that all three agents are learning (loss is converging, and the reward is increasing). But out of all the best results in terms of loss as well reward comes from Dueling Double DQN. We can also observe that the loss in DQN is not quite stable and this is due to the positive bias overestimation.

Once we realized that Dueling Double DQN was the best performing agent, we tested the DDDQN in the second dataset without changing the parameters and training and testing on all 505 stocks for 100 Epochs. As benchmark we compared it to the buy and hold strategy of these individual stocks during the out of sample period.

Taking the average across all 505 stocks and taking stocks with number of round trip trades greater than 2 (because the low number of trades makes DDDQN risk adjusted return ratio's positively skewed)

Strategy	Mean	Median
Buy and Hold-risk adjusted return	0.045538	0.046806
DDDQN-risk adjusted return	1.259177	1.135853

$$risk\ adjusted\ return = \frac{Mean\ (returns)}{Standard\ Deviation(returns)} \qquad (4)$$

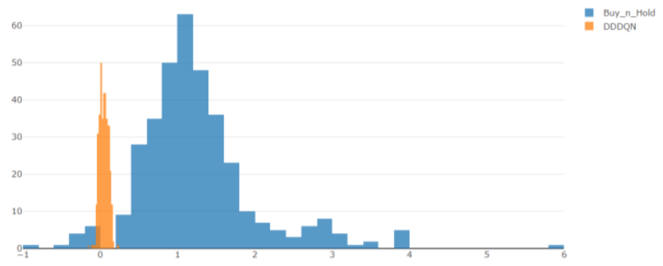

Figure 8: Risk Adjusted Return Distribution

Strategy	Mean	Median
Buy and Hold total return	17.73 %	17.75%
DDDQN-total return	21.13 %	17.6%

Conclusion

In this project, we have demonstrated that deep reinforcement learning strategies can successfully learn policies with minimal information to trade in the financial markets. Furthermore, it differentiates itself from supervised learning trading strategies, where predictions need to go through risk and execution models, whereas in reinforcement learning we can put any risk or trading constraints into the rewards structure. One improvement we believe can be implemented is a Prioritized Experience Replay (Schaul, et al. 2015) mechanism. This algorithm replays important or unlearned experiences more frequently thus improving the learning process which is similar to what we humans do. Another improvement is to utilize a Recurrent Neural Network (Gers, Schmidhuber and Cummins 1999) where the input is just the price in a time-series data. As stock market data is time-series data, a Long Short-Term Memory architecture negates the need for indicator values.

Bibliography

Allen, Franklin, and Risto Karjalainenb. 1999. "Using genetic algorithms to find technical trading rules1." *Journal of Financial Economics* 51 (2): 245-271.

Barto, Andrew G., and Richard S. Sutton. 1998. *Introduction to reinforcement learning.* Vol. 135. Cabmridge, Massachusetts, London: The MIT Press.

Bengio, Yoshua. 2009. "Learning deep architectures for AI." *Foundations and Trends® in Machine Learning* 2 (1): 1-127.

Cumming, James, Dalal Alrajeh, and Luke Dickens. 2015. ""An investigation into the use of reinforcement learning techniques within the algorithmic trading domain." *PhD diss., Master's thesis, Imperial College London, United Kiongdoms.*

Deng, Yue, Youyong Kong, Feng Bao, and Dai Qionghai. 2015. "Sparse Coding-Inspired Optimal Trading System for HFT Industry." *IEEE Transactions on Industrial Informatics* 11 (2): 467-475.

Gers, Felix A., Jürgen Schmidhuber, and Fred Cummins. 1999. "Learning to forget: Continual prediction with LSTM." *9th International Conference on Artificial Neural Networks, ICANN* 850-855.

Graves, Alex, Abdel-rahman Mohamed, and Geoffrey Hinton. 2013. "Speech recognition with deep recurrent neural networks." *Acoustics, speech and signal processing (icassp), 2013 ieee international conference* (IEEE) 6645-6649.

Hasselt, Hado V. 2010. "Double Q-learning." *Advances in Neural Information Processing Systems* 2613-2621.

Hasselt, Hado V., Arthur Guez, and David Silver. 2016. "Deep Reinforcement Learning with Double Q-Learning." *AAAI'16 Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence* 16: 2094-2100.

Lee, Honglak, Roger Grosse, Rajesh Ranganath, and Andrew Y. Ng. 2009. "Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations." *Proceedings of the 26th annual international conference on machine learning* (ACM) 609-616.

Li, Yuxi. 2017. "Deep Reinforcement Learning: An Overview." *arXiv preprint arXiv:1701.07274.*

Lin, Long-Ji. 1992. "Reinforcement Learning for Robots Using Neural Networks." *Doctoral Dissertation* (Carnegie Mellon University).

Meese, Richard A, and Rogoff Kenneth. 1983. "Empirical exchange rate models of the seventies: Do they fit out of sample?" *Journal of International Economics* 14 (1-2): 3-24.

Meese, Richard A, and Rogoff Kenneth. 1997. "The out of sample failure of empirical exchange rate models." *Exchange rates and international macroeconomics* (University of Chicago Press) 67-112.

Mnih, Volodymyr, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. 2013. "Playing atari with deep reinforcement learning." *arXiv preprint arXiv:1312.5602.*

Mnih, Volodymyr, Koray Kavukcuoglu, Silver David, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, et al. 2015. "Human-level control through deep reinforcement learning." *Nature - International Journal of Science* 518: 529-533.

Moody, John, and Lizhong Wu. 1997. "Optimization of trading systems and portfolios." *Computational Intelligence for Financial Engineering (CIFEr), 1997., Proceedings of the IEEE/IAFE 1997* (IEEE) 300-307.

Moody, John, and Matthew Saffell. 2001. "Learning to trade via direct reinforcement." *IEEE transactions on Neural Networks* 12 (4): 875-889.

Moody, John, and Saffell Matthew. 1999. "Reinforcement learning for trading." *Advances in Neural Information Processing Systems* 917-923.

Neely, C. J.., and P. A. Weller. 2003. "Intraday technical trading in the foreign exchange market." *Journal of International Money and Finance* 22 (2): 223-237.

Nevmyvaka, Yuriy, Yi Feng, and Michael Kearns. 2006. "Reinforcement learning for optimized trade execution." *Proceedings of the 23rd international conference on Machine learning* 673-680.

Puterman, Martin L. 2014. *Markov decision processes: discrete stochastic dynamic programming.* Hoboken, New Jersey: John Wiley & Sons.

Schaul, Tom, John Quan, Ioannis Antonoglou, and David Silver. 2015. "Prioritized experience replay." *arXiv preprint arXiv:1511.05952.*

Sutton, Richard S., and Andrew G. Barto. 1998. *Reinforcement Learning: An Introduction.* Vol. 1. 1 vols. Cambridge, Massachusetts, London: The MIT Press.

Wang, Ziyu, Tom Schaul, Matteo Hassel, Hado van Hasselt, Marc Lanctot, and Nando de Freitas. 2015. "Dueling Network Architectures for Deep Reinforcement Learning." *arXiv preprint arXiv:1511.06581.*

Watkins, Christopher John Cornish Hellaby. 1989. "Learning from delayed rewards." *PhD diss. King's College.*