Application of Deep Reinforcement Learning in Algorithmic Trading & Portfolio Management

Hieu Trung Le
Stanford University
Computer Science Department
leric@stanford.edu

Abstract

Algorithmic trading is a financial problem where an agent constantly executing orders utilizing automated and pre-programmed trading instructions. This project explores an innovative approach based on deep reinforcement learning (DRL) to solve the traditional algorithmic trading problem. Known as Trading Deep Q-Network algorithm (TDQN) [7] and adapted from the Deep Q-Network (DQN) [5] algorithm, we will teach a reinforcement learning (RL) agent to optimally execute a series of trading orders that not only can generate profit, but also can efficiently mitigate the risk associated with trading

1. Introduction

Algorithmic trading is the use of process and rules based algorithms to employ strategies for executing trades. It has been employed in the early 1980s and is used by institutional investors and large trading firms for a variety of purpose including managing portfolio, cutting down on costs associated with trading and generate positive return for wealthy client. Algorithmic trading also allows for faster and easier execution of orders, making it attractive for traders and investors who would like to quickly make profits off small changes in price. [1]

Instead of determining trading policy using traditional trading algorithms, we will frame the problem as follow: Consider a portfolio consists of one single stock together with the agent cash, the input to our algorithm is the historical data of OHCLV (Open/High/Close/Low/Volume) prices. A RL trading agent interacts with the stock market through an order book, which contains the entire set of buying orders (bids) and selling orders (asks). An order represents the willingness of a market participant to trade and is composed of a price p, a quantity q and a side s (bid or ask). We then train the trading agent to know what, when, how, at which price and which quantity to trade the stock in hand. After successfully being trained, the agent will be able to determine when what, when and how much does it need to execute trading decision to maximize profit.

2. Related works

2.1. Classical Techniques

Some of the earlier technique in algorithmic trading are developed by mathematicians, economists and traders who do not exploit AI. One example of such classical techniques are called trend following and mean reversion strategies [2].

2.2. Deep Learning Techniques

Bao et all [3] later presents a novel deep learning framework where wavelet transforms (WT), stacked autoencoders (SAEs) and long-short term memory (LSTM). First, the stock price time series is decomposed by wavelet transforms to eliminate noise. Second, stacked autoencoders is applied to generate deep high-level features for predicting the stock price. Third, high-level denoising features are fed into long-short term memory to forecast the next day's closing price.

2.3. Reinforcement Learning Techniques

Later, Moody et all [6] presents methods that use an adaptive algorithm called recurrent reinforcement learning (RRL) for discovering investment policies which eliminates the need to build forecasting models and result in better trading performance. Another example of reinforcement learning in algorithmic trading is Deng et al [9], who introduced recurrent deep neural network structure to obtain a technical-indicator free trading system The technique combines deep learning and reinforcement learning.

3. Dataset and Features

3.1. Datasets

We obtain the historical data of Apple, Google, Microsoft, Coca-cola, Tesla stocks from 01/01/2012 to

12/31/2020 from Yahoo Finance `https://finance.yahoo.com/`. This period is selected as the representative of the current market condition. They are split it into into training, validation and testing. We will trained the algorithm on Apple, Google, Microsoft stock which is fairly stable and evaluate on Apple, Google, Microsoft, Coca-cola, Tesla stocks.

- Training: 01/01/2012 to 12/31/2018
- Testing: 01/01/2019 to 12/31/2020

Data pre-processing & normalization: In order to reduce the high-frequency noise present in the trading data, a low-pass filtering operation is applied to the training data set. This pre-processing step has a downside because it modifies or destroys some potentially useful trading patterns. Once the data is passed through the filter, it was then transformed in order to convey more meaningful information about market movements. For example, instead of the raw prices, the daily evolution of prices is computed. The data are then normalised using batch normalisation layers which results in adjusting and scaling the activation functions.

3.2. Features

In this project, we will frame this problem as a state-based model in order to use reinforcement learning. Here, we give the definition of our states, actions, rewards and policy:

3.2.1 State

The state space consists of internal state and market observations.

- Internal state: which is defined by the current portfolio value, how many shares it has and the net long or short position.
- Market observations: current state of the market.

Let o_t represents the state space at present time t

$$o_t = S(t), D(t), I(t), M(t), N(t), E(t)$$

where

- S(t) represents the internal state the RL agent at time t
- D(t) is the current information gathered about current OHCLV (Open/High/Close/Low/Volume) prices where $D(t) = p_t^O, p_t^H, p_t^L, p_t^C, V_t$
- T(t) is the trading time step t (date, weekday, time)
- I(t) is the information regarding multiple technical indicators about the stock market at time step t
- M(t) is the macroeconomic information at time step t.
- N(t) is the news information gathered by the agent at time step t.

3.2.2 Action

At each time step, the agent executes an action a_t and making decision of whether to go long, short or do nothing (hold). $a_t \in \mathcal{A}$ where $\mathcal{A}$ is the action space. The agent also makes decision on how many share to either long/short or hold. Let Q_t defines the agent action at time step t. We have

- $Q_t > 0$: The RL agent buys or go long
- $Q_t < 0$: The RL agent sells or go short
- $Q_t = 0$: The RL agent holds or do nothing

Let v_t^c be the cash value on hand at time t and p_t is the current market price of each share. We have the first constraint

$$v_{t+1}^c = v_t^c - Q_t p_t$$

This means the cash value on hand v_t^c has to remain positive for every trading time steps t.

Also, $v_{t+1}^c \geq 0$. This means after the agent buy Q_t shares, the cash value on hand will changes but it should always remain positive. This gives an upper bound to the number of shares that the agent can buy.

The second constraint is

$$v_{t+1}^c \geq -n_{t+1} p_t (1 + \epsilon)$$

where $\epsilon \geq \frac{p_{t+1} - p_t}{p_t}$ is relative change in market prices and n_{t+1} is the number of shares at t+1. This constraint means that in the short position, the number of shares that the agent can trade is limited so that the agent will have enough cash value to pay back share lenders.

3.2.3 Reward

The RL rewards is the daily return that the agent get from the strategy. Let r_t is the reward at time step t. R is the discounted reward of all future return from 0 to infinity

$$r_t = \frac{v_{t+1} - v_t}{v_t}$$

$$R = \sum_{t=0}^{\infty} \gamma^t r_t$$

Another objective function the agent need to maximize is called the Sharpe ratio. Sharpe ratio is the ratio of the excess return of the portfolio, relative to the risk-free rate to the standard deviation of the portfolio's excess returns.

$$\text{Sharpe ratio} = \frac{R_p - R_f}{\sigma_t}$$

where R_p is the return of the portfolio, R_f is the risk free rate and σ_t is standard deviation of the portfolio's excess returns a time t.

3.2.4 Policy

We will view trading strategy the agent trading execution as a programmed policy $\pi(a_t|i_t)$ which outputs a trading action a_t according to the information available to the trading agent i_t at time step t. An agent executing its trading strategy sequentially applies the following steps:

- Update available market information i_t
- Execution of the policy $\pi(a_t|i_t)$ to get action a_t.
- Apply action a_t
- Next time step $t \rightarrow t+1$, go back to step 1.

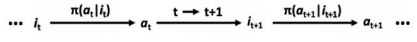

Figure 1: Illustration of a trading strategy execution

The resulting optimal policy π^* is expressed as:

$$\pi^* = \arg max_\pi E[R|\pi]$$

where R is the discounted reward. This means the optimal policy will produce an action that aims to maximize the expected discount reward given all the policy

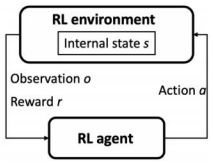

Figure 2: RL core building blocks

4. Methods

First, the Deep Q-Network (DQN) [5] algorithm was chosen as starting point for the our trading strategy. The Deep Q-Network (DQN) is an example of Deep Reinforcement Learning algorithm which is capable of successfully learning control policies from high-dimensional sensory inputs. The algorithm is model-free, meaning that a complete model of the environment is not required and that trajectories are sufficient. It is based on the learning of an approximation of the state-action value function. The algorithm is known to be off-policy as it exploits in batch mode previous experiences $e_t = (s_t, a_t, r_t, s_{t+1})$ collected at any point during training

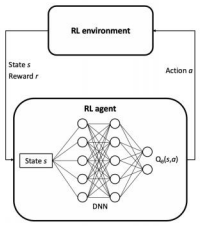

Figure 3: Illustration of the DQN algorithm

We have made several modification to the existing Deep Q-Network (DQN) [5] algorithm to adapt better with algorithmic trading decision:

- neural network architecture: convolutional neural network (CNN) will be replaced by a classical feedforward DNN (Deep Neural Network) with leaky rectified linear unit (Leaky ReLU) activation functions.
- Double DQN (DDQN) [8]: instead of Deep Q-Network (DQN), we will use Double DQN (DDQN) [8] as a modification which tackles the overestimation bias of DQN
- Adam Optimizer [4]: the classical DQN algorithm implements the RMSProp optimiser. Instead we will use ADAM optimiser
- Huber loss: the DQN algorithm implements a mean squared error (MSE) loss. For our model, we will implement the Huber loss because experimentally it improves the stability of the training phase.
- Gradient clipping: implemented in order to solve the gradient exploding problem
- Batch normalisation: normalising the input layer by adjusting and scaling the activation functions which leads to more robust training
- Regularisation: three regularisation techniques are implemented: Dropout, L2 regularisation and Early Stopping to reduce overfit

We will implement the TQDN algorithm [7] on Appendix 7.1

5. Experiments/Results/Discussion

5.1. Experiments

In order to assess the strengths and weaknesses of the TDQN algorithm, some benchmark were selected for comparison purposes. Despite the fact that TQDN algorithm is active strategy, we select both passive and active strategies for comparison for the sake of fairness. The first two benchmark trading strategies (BH and SH) are passive trading strategies, because there are no changes in trading position. The other two benchmark strategies (TF and MR) are active trading strategies because it requires multiple changes in trading positions over the trading horizon

We selected multiple performance indicators to assess the performance of a trading strategy. The complete list of the performance indicators we selected are in Appendix 7.2

The most important performance indicators, however, is the Sharpe rati as it is widely used in the field of algorithmic trading and is particularly informative as it combines both profitability and risk. In order to estimate the expected performance as well as the variance of the TDQN algorithm, the same RL trading agent is trained multiple times. We trained the TQDN algorithm on the sample data from 01/01/2012 to 12/31/2018 of Apple, Google, Microsoft stocks. We train the model using Google Colab GPU on 100 episodes and below is the graph of Performance(Sharpe ratio) vs episode.

5.2. Results

Due to the space limit, we will only be able to show the result of Apple and Tesla stock. Below is the graph that show Sharpe ratio on training and testing on Apple stocks

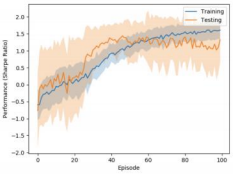

Figure 4: Sharpe ratio on training and testing on Apple stocks vs episode

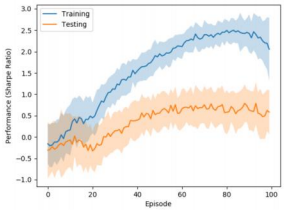

Figure 5: Sharpe ratio on training and testing on Tesla stocks vs episode

Perf Indicator	B&H	S&H	TF	MR	TDQN
Sharpe ratio	1.182	-1.682	1.291	-0.781	1.603
Profit & loss	81371	-84382	64232	-38920	107922
Ann. return (%)	22.38	-107.71	28.19	-13.92	38.12
Ann. volatility (%)	28.39	48.12	27.76	26.12	28.96
Prof. ratio (%)	96.89	0.83	41.98	59.83	50.37
P&L ratio	∞	0.27	3.193	0.378	3.672
Sortino ratio	1.367	-3.971	1.716	-0.692	1.973
MD (%)	31.83	88.91	13.12	58.63	18.19
MD duration	67	236	31	214	19

Table 1. Performance assessment on Apple

Perf Indicator	B&H	S&H	TF	MR	TDQN
Sharpe ratio	0.213	-0.281	-0.789	0.412	0.293
Profit & loss	26921	-27420	-72921	8913	78
Ann. return (%)	25.91	-5.29	-96.19	18.32	14.83
Ann. volatility (%)	51.87	44.92	54.92	55.82	54.82
Profitability ratio (%)	95.82	0.93	32.19	62.02	34.02
P&L ratio	∞	0.92	0.792	0.920	1.391
Sortino ratio	0.928	-0.339	-1.283	0.792	0.620
MD (%)	59.12	58.39	89.32	78.31	59.95
MD duration	289	189	238	2178	390

Table 2. Performance assessment on Tesla

Below is the cumulative capital following the execution of the algorithm on Apple and Tesla test set

Figure 6: TDQN algorithm total capital on testing on Apple stocks

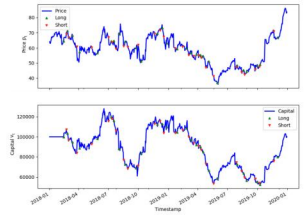

Figure 7: TDQN algorithm execution for the Tesla stock

5.3. Discussion

As we have noted from Figure 4, both Sharpe ratio of training and test set of Apple stock increase with the number of episode. The test set Sharpe ratio plot rapidly increase at first and seem to be superior comparing to the training set. It then levels out and stay below the training plot. The earlier superior performance of test set can be explained by noticing that the trading period of test set is the profitable market for the Apple stock. Due to different market condition, this also shows the difficulty of algorithmic trading problem where training and test sets do not share the same distributions and therefore can be different from one another in term of Sharpe ratio

Figure 6 shows 2 graphs. The first graph is the stock market price for each timestamp. The second graph is the portfolio value for each timestamp. Both graph has indicator with the actions at outputted by the TDQN algorithm. It can be observed that the deep reinforcement learning trading strategy is capable of accurately detecting and benefiting from major trends in the market (and go long or buy). It is also able to detect market behavioural shifts when the volatility increases and go short or sell. Overall, we see by implementing the trading strategies based on the TDQN algorithm, our portfolio value kept increasing overtime and double in value over the period of 2 years.

As we have observed from Table 1, the TDQN algorithm achieves good results from both an earnings and a risk mitigation point of view for the case of Apple stock, clearly outperforming all the benchmark active and passive trading strategies

In the case of Tesla stock, Figure 5 shows the Sharpe ratio of training and test set of Tesla stock. We notice that the Sharpe ratio of training first increase similar to test set and end up outperforming the test set. This suggests that our model did not perform as well during testing as comparing to during training in term of Sharpe ratio. This shows that there is a limitation in the TDQN algorithm. It only works well with low variance stock. With stocks that have high variance, the algorithm could end up selecting poorly performing policies compared to the expected performance. Also, we notice there is significantly higher performance in term of Sharpe ratio achieved on the training set. This means the TDQN algorithm is subject to overfitting in certain cases, despite the regularisation we have applied during training. The overfitting might be explained due to the limited observation space $\mathcal{O}$ that does not efficiently apprehend the Tesla stock. To reduce the overfitting, it might be worth to expand the observation space $\mathcal{O}$ to encapsulate other information about the stocks. Overfitting in this case can be indicative that the TDQN algorithm may lead to poor performance for other stocks with similar characteristic as Tesla stocks.

Figure 7 shows the graph of price and cumulative capital for each timestamp. Even though we start with $\$100,000$, we end up with exactly the same amount. This suggest that the Tesla stock is quite difficult to trade, which is partly due to its significant volatility. Even though the TDQN algorithm achieves a positive Sharpe ratio, almost no profit is generated. Also, we notice that the risk level associated with this trading activity is abnormally high. For example, the maximum drawdown duration is particularly large. From Figure 7, we can be observe that the volatility of the Tesla stock induces a higher trading frequency which increases even more the riskiness of the DRL trading strategy

6. Conclusion/Future Work

In this project we explores an innovative approach based on deep reinforcement learning (DRL) to solve the traditional algorithmic trading problem. Known as Trading Deep Q-Network algorithm (TDQN) [7]. We obtain the historical stock data from Yahoo Finance from 2012-2020 as input. We then frame the problem as state-based model in order to use reinforcement learning. The Deep Q-Network (DQN) [5] algorithm was chosen as ur trading strategy. We then apply some modification to adapt better with algorithmic trading decision such as the Double DQN, Adam Optimizer and Huber Loss that mildly penalize large errors, batch normalization and regularization. We trained the TQDN algorithm on the sample data from 01/01/2012 to 12/31/2018 of Apple, Google, Microsoft stocks. The TDQN algorithm achieves good results from both an earnings and a risk mitigation point of view for the case of Apple stock. With a rigorous performance assessment, the TDQN algorithm achieves promising results, surpassing on average the benchmark trading strategies. The TDQN algorithm also demonstrates multiple benefits compared to classical approaches such as versatility and robustness to trading costs.

In the case of Tesla, it does not perform quite as well without any profit gained. We discovered there is a limitation in the TDQN algorithm. It only works well with low variance stock and not so much with high variance stock. For future work, I would like to explore extending the observation space $\mathcal{O}$ to enhance the observability of the trading environment. I also would like to explore relaxing some constraints about the action space $\mathcal{A}$ to enable new trading possibilities. I also would like to explore looking at distributions instead of expected values in the TDQN algorithm to see if that would improve the result in case of Tesla or other high variance stock.

7. Code

The code for this project can be found at `https://github.com/leeric92/cs229-final-project`

8. Appendices

8.1. TDQN Algorithm

Algorithm 1 TDQN algorithm

Initialise the experience replay memory M of capacity C.
Initialise the main DNN weights θ (Xavier initialisation).
Initialise the target DNN weights $\theta^- = \theta$.
for episode = 1 **to** N **do**
  Acquire the initial observation o_1 from the environment $\mathcal{E}$ and preprocess it.
  for t = 1 **to** T **do**
    With probability ϵ, select a random action a_t from $\mathcal{A}$.
    Otherwise, select $a_t = \arg\max_{a \in \mathcal{A}} Q(o_t, a; \theta)$.
    Copy the environment $\mathcal{E}^- = \mathcal{E}$.
    Interact with the environment $\mathcal{E}$ (action a_t) and get the new observation o_{t+1} and reward r_t.
    Perform the same operation on $\mathcal{E}^-$ with the opposite action a_t^-, getting o_{t+1}^- and r_t^-.
    Preprocess both new observations o_{t+1} and o_{t+1}^-.
    Store both experiences $e_t = (o_t, a_t, r_t, o_{t+1})$ and $e_t^- = (o_t, a_t^-, r_t^-, o_{t+1}^-)$ in M.
    if t % T' = 0 **then**
      Randomly sample from M a minibatch of N_e experiences $e_i = (o_i, a_i, r_i, o_{i+1})$.
      Set $y_i = \begin{cases} r_i & \text{if the state } s_{i+1} \text{ is terminal,} \\ r_i + \gamma\ Q(o_{i+1}, \arg\max_{a \in \mathcal{A}} Q(o_{i+1}, a; \theta); \theta^-) & \text{otherwise.} \end{cases}$
      Compute and clip the gradients based on the Huber loss $H(y_i,\ Q(o_i, a_i; \theta))$.
      Optimise the main DNN parameters θ based on these clipped gradients.
      Update the target DNN parameters $\theta^- = \theta$ every N^- steps.
    end if
    Anneal the ϵ-Greedy exploration parameter ϵ.
  end for
end for

8.2. Performance Indicators

Performance indicator	Description
Sharpe ratio	Return of the trading activity compared to its riskiness.
Profit & loss	Money gained or lost at the end of the trading activity.
Annualised return	Annualised return generated during the trading activity.
Annualised volatility	Modelling of the risk associated with the trading activity.
Profitability ratio	Percentage of winning trades made during the trading activity.
Profit and loss ratio	Ratio between the trading activity trades average profit and loss.
Sortino ratio	Similar to the Sharpe ratio with the negative risk penalised only.
Maximum drawdown	Largest loss from a peak to a trough during the trading activity.
Maximum drawdown duration	Time duration of the trading activity maximum drawdown.

References

[1] Algorithmic trading/ automated trading education - investopedia. https://www.investopedia.com/terms/a/algorithmictrading.asp, note = Updated: Oct 26, 2020.

[2] M. Babayev, F. Lotun, G. T. Mumvenge, and R. Bhattacharyya. Short term trading models – mean reversion trading strategies and the black swan events, 2020.

[3] R. Y. Bao W, Yue J. A deep learning framework for financial time series using stacked autoencoders and long-short term memory, 12 2017.

[4] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization, 2017.

[5] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller. Playing atari with deep reinforcement learning, 2013.

[6] J. Moody and M. Saffell. Learning to trade via direct reinforcement. *IEEE transactions on neural networks*, 12 4:875–89, 2001.

[7] T. Théate and D. Ernst. An application of deep reinforcement learning to algorithmic trading, 2020.

[8] H. van Hasselt, A. Guez, and D. Silver. Deep reinforcement learning with double q-learning, 2015.

[9] Y. K. Z. R. Yue Deng, Feng Bao and Q. Dai. Deep direct reinforcement learning for financial signal representation and trading, 2016.