CS461 Artificial Intelligence

Intermediate Report

Group 11

Berk Demirkan EE 21802124 - Özgür Karadağ EE 21902801

I. Introduction

In financial trading, where things are constantly changing and hard to predict, people used to rely on their feelings, experience, and fundamental analysis to make investment choices. But now, with AI and advanced tech improving, algorithmic trading is becoming a big deal. Our project is about testing different reinforcement learning types to determine the best way to improve trading strategies. We want to make an AI agent to learn from old and new financial data to make more money and avoid losing it. We used data like past stock prices, technical stuff, and financial news to train our AI in a way that is compatible with OpenAI Gym. This is all about teaching the AI to get good at handling the ups and downs of the stock market.

II. Background and Previous Work

Reinforcement learning is *a valuable* part of AI, where agents, like computer programs, learn to make decisions to get the most rewards. It is helpful for many critical applications, *such as financial trading*, where you discover the best moves through trial and error. There has been some informative work in this area already. For example, in one research [1], Huang came up with a Deep Recurrent Q-Network (DRQN) made for trading. It *uses* an action augmentation technique that helps the AI learn without having to try out some random stuff, and it gives extra tips to the agent to learn better. Also, there is this project [2] by ClementPerroud on GitHub, who made their own Deep Q-Network. It actually made money, with a transaction fee of 0.1%.

III. Methodology

A. The Environment

We gathered historical stock price data from multiple sources (Binance, Bybit, Huobi). We could download and use historical cryptocurrency data for training our agent in an OpenAI Gym using CelementPerroud's Gym-Trading-Env repository [3]. We downloaded historical five-minute interval Bitcoin/USD and Ethereum/USD data from January 1st, 2023, until November 20th, 2023. The downloaded data was then resampled with offsets to get thirty-minute interval data, which increased data variation and overall data available. Gym-Trading-Env module provides data in a OpenAI-Gym compatible, reinforcement learning oriented way, with observations that the agent needs to take action on. The environment also offers methods to get multiple-step observations for agents with recurrent networks. While our implementation does not have recurrent neural networks, we used this feature by increasing the input dimensions of our networks.

The environment is initialized using the initial portfolio value, a list of positions for the agent to choose from, and the trading fee amount. A list of positions provided to the environment means this task is discrete. We decided on an initial portfolio value of 1000 US dollars, a trading fee of 0.18%, corresponding to BTCTurk trading fees, and a position list from 0 to 1 in 0.1 increments. A position of 0 means that all the portfolio value is

kept as dollars, and a position of 1 means that all the portfolio value is kept as cryptocurrency.

The observation space of the environment depends on the data it is initialized with. The Gym-Trading-Env requires a pandas dataframe, in which the columns with the text "feature" are used for observation. The feature columns of our data included the percentage change of the closing price, the ratio of opening price and closing price, the ratio of highest price and closing price, the ratio of the lowest price and closing price, and the ratio of market volume and the running average of market volume over a week. Because the close feature is the percentage change since the last interval, and all of the other features are taken with respect to the closing price, we can get a reasonable estimate of how the market is doing from only these five features.

The rewards from the environment are calculated from the following equation, with p_t being the portfolio value at timestep t.

$$r_t = \ln\left(\frac{p_t}{p_{t-1}}\right)$$

B. Double Deep Q-Learning Network

In contrast to a simple deep Q-Learning network, DDQN models use two neural networks: one to evaluate states (online network) and the other to assess actions (target network). This minimizes the value overestimation problem of a simple Deep Q-learning network. The target network is not trained at every training cycle. Instead, we decided to synchronize the two networks periodically every five episodes. The update frequency is an essential hyperparameter of the DDQN model. If the updates are too frequent, the model tends to overestimate values, negating the purpose of the DDQN. If the updates are too infrequent, the two networks are not correlated enough to ensure score optimization.

C. Action Selection

We decided to use epsilon-greedy as an action selection method. Thompson sampling required data transfer between the GPU and the CPU for every action selection for our specific implementation. Thus, we decided to use epsilon-greedy due to the ease of implementation. A multiplicative epsilon decay with 0.95 was used to speed up the decay since the environment is the limiting part of our implementation.

D. Optimizer and Model Saving

We selected the Adam optimizer to optimize the learning rate instead of using a custom learning rate scheduler. During the learning process, all the environment variables, agent parameters, and other auxiliary data were saved to a file using the "dill" module after every episode. This enabled us to easily continue training our model if an error that caused the training to stop occurred.

IV. Results

We tested the Double Deep Q-Learning Network (DDQN) throughout this study and obtained informative results. The first stage created the foundation for a strong training environment by collecting and preprocessing historical stock price data for Ethereum and Bitcoin. We processed and divided this data into thirty-minute intervals. This improved the variety of our dataset and increased the amount of data available for the DDQN.

The agent had multiple episodes during training, offering a distinct set of experiences from the historical data. As the events went on, there was an apparent upward trend despite the early negative returns, with the portfolio return frequently lowest at -100%, which indicates the financial market's complexity and learning curve. This pattern is seen in the portfolio return, which improved over time, showing the agent's capacity for adaptation and learning.

The epsilon-greedy action selection approach turned out to be the correct choice, enabling a successful balance between investigating new approaches and applying profitable actions. The agent moved from exploration to exploitation more quickly thanks to the multiplicative epsilon decay method, which also helped it adjust its policy over time to take more beneficial actions as training continued.

The portfolio return, a critical success indicator, significantly improved over time. After several episodes, the DDQN agent started to beat the market constantly and achieved a maximum portfolio return of 72.78%. In addition, as time passed, the position changes measure dropped, showing that the agent's trading approach was stabilizing and that there was

less excessive trading, frequently resulting in transaction costs and reduced earnings.

The stability and reliability of the model were also shown by the agent's ability to maintain a high score despite the volatility in the market returns, which varied greatly throughout the training episodes. The DDQN agent's highest score was 0.547, showing the model's ability to produce profits even when there were fluctuations in the market.

Finally, it can be said that the DDQN agent showed potential in handling the complex world of financial trading. Over time, it improved its performance by changing its methods in response to the unstable market conditions. The increasing skill of the agent is a clear indication that algorithmic trading could undergo a revolution thanks to reinforcement learning models.

The episode vs. portfolio returns and episode vs. score graphs that we obtained are as follows:

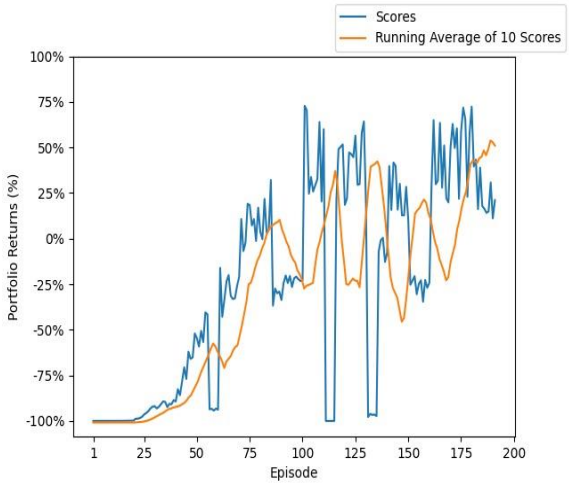

Figure 1: Episode vs. Portfolio Returns (%)

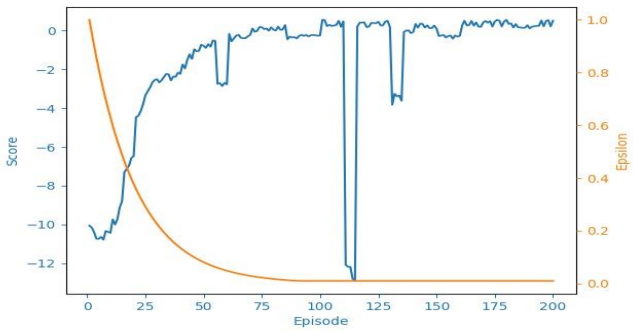

Figure 2: Episode vs. Score

V. Challenges

Integration of financial data into the DDQN model was a significant challenge. Financial markets are noisy and unpredictable, which makes it difficult for the algorithm to distinguish between random fluctuations. We had to ensure that the DDQN could respond to market changes. In addition to that, financial markets can be highly volatile. Creating a stable and reliable model under various market conditions is essential. Consequently, although the DDQN method effectively optimized financial trading strategies, overcoming these challenges required repetitive testing. While integrating financial data into the DDQN model posed a significant challenge due to financial markets' noisy and unpredictable nature, we encountered additional challenges. For instance, preprocessing the data to filter out 'market noise' was critical. We had to develop sophisticated data normalization techniques to help the DDQN algorithm focus on underlying trends rather than being misled by short-term fluctuations. In conclusion, the DDQN method provided a promising framework for optimizing financial trading strategies; however, the path to a reliable and efficient model was not easy, and there were complex challenges that needed to be solved by strategic data management, advanced algorithmic design, and repetitive testing to overcome.

VI. Conclusion

The great potential of reinforcement learning in financial applications has been made clear by this project using a Double Deep Q-network (DDQN). The course of the research, from the data preprocessing to the DDQN agent training, has demonstrated how artificial intelligence can adapt to the volatile world of the stock market. Although the initial difficulties included significant portfolio losses and a high learning curve for the agent, the DDQN showed a capacity to adapt to its environment and improve its decision-making techniques. The agent also made significant gains, as demonstrated by the highest score of 0.547 and the maximum portfolio return of 72.78%. In contrast to the market returns, these numbers showed the agent's skill at maximizing profit and lowering risk. As training continued, the agent's frequency of position adjustments decreased, suggesting a more stable and developed trading strategy. This consistency and the agent's rising returns show the learning process's development and

improved understanding of the market. Additionally, by effectively balancing the exploration of new techniques with the use of established ones, the epsilon-greedy strategy for action selection allowed for efficient learning. The reliability of the training process was further improved by the employment of the Adam optimizer and the model state preservation approach that allowed training to continue without interruption. The results of this experiment support the need for greater study on reinforcement learning in finance. It also opens the way for the possible adoption of AI-driven trading techniques in the future.

The code, data, and logs for this implementation can be found in [4].

REFERENCES

[1] C. Y. Huang, *'Financial trading as a game: A deep reinforcement learning approach,'* arXiv [q-fin.TR], 08-Jul-2018.

[2] ClementPerroud, *"Clementperroud/RL-trading-agent: Profitable Reinforcement Learning Agent on crypto markets,"* GitHub, https://github.com/ClementPerroud/RL-Trading-Agent (accessed Oct. 23, 2023).

[3] ClementPerroud, "*Clementperroud/Gym-trading-env: A simple, easy, customizable gymnasium environment for trading.,"* GitHub, https://github.com/ClementPerroud/Gym-Trading-Env (accessed Oct. 23, 2023).

[4] B. Demirkan, Ö. Karadağ, "*CS461 Project Group 11 – Google Drive*," Google Drive, https://drive.google.com/drive/folders/1DNH-oo-seQAjeVRWMVx0DdaXMFI5xnpN?usp=sharing (accessed Nov. 26, 2023