CS461 Artificial Intelligence
Final Report
Group 11

Özgür Karadağ
21902801
EE

Berk Demirkan
21802124
EE

I. Introduction

In the field of financial trading, traditional decision-making relied heavily on human intuition, experience, and fundamental analysis due to the dynamic and unpredictable nature of the sector. However, with the advent of AI and cutting-edge technology, the industry is transforming towards algorithmic trading. Our project is at the forefront of this evolution, exploring and evaluating various types of reinforcement learning to refine and enhance trading strategies. We specifically look into Double Deep Q-Learning and Soft Actor-Critic (SAC) methodologies.

Double Deep Q-learning is a sophisticated approach that addresses the overestimation bias often seen in standard Q-learning, making our AI's decision-making process more reliable and efficient. On the other hand, SAC is an advanced model in reinforcement learning that emphasizes learning a policy that maximizes both the expected return and entropy. This encourages a more exploratory and robust strategy in the constantly fluctuating stock market.

We aim to develop an AI agent capable of assimilating historical and contemporary financial data to maximize profits while minimizing losses. To achieve this, we utilized diverse data sources, including historical stock prices, technical indicators, and financial news. This rich dataset serves as the training material for our AI, which is tailored to be compatible with the OpenAI Gym environment. This compatibility is crucial as it provides a standardized and versatile platform for testing and refining our algorithms.

The core of our project lies in teaching AI agents to adeptly navigate the tumultuous waves of the stock market, learn from its intricacies, and adapt to its volatile nature. By leveraging Double Deep Q-Learning and Soft Actor-Critic methodologies, we aim to harness the full potential of AI in revolutionizing traditional trading strategies. This will move us beyond reliance on human judgment towards a more data-driven, algorithmic approach.

II. Background and Previous Work

Reinforcement learning is a crucial aspect of artificial intelligence. It involves training entities (known as agents) such as computer programs to make strategic decisions that maximize their rewards. This technique is particularly useful in high-stakes fields like financial trading. Here, agents learn to identify and execute the most advantageous trading strategies through trial and error, continuously refining their decisions based on the outcomes.

Reinforcement learning has seen significant advancements and notable contributions. For instance, Huang developed a specialized model called the Deep Recurrent Q-Network (DRQN) that is tailored to the complexities of trading. This model stands out due to its action augmentation technique, which equips the AI with the ability to learn effectively without extensive random experimentation. This method streamlines the learning process and provides additional guidance, enhancing the agent's capacity to assimilate knowledge and apply it proficiently.

Moreover, the realm of open-source contributions has also seen remarkable developments. ClementPerroud's project on GitHub involved creating a unique Deep Q-Network that demonstrated its practical utility by generating profits in real-world trading scenarios, even when accounting for a transaction fee of 0.1%. Such endeavors underscore the dynamic and rapidly evolving nature of reinforcement learning, especially in its application to complex and consequential fields such as financial trading.

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 thirty-minute interval Bitcoin/USD and Ethereum/USD data from January 1st, 2023, until December 6Th, 2023. The downloaded data was then resampled with offsets to get six-hour interval data, which increased data variation and overall data available. The Gym-Trading-Env module provides data in an 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 data frame, 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.

1) Action Space: During the training of our DDQN model, we utilized a discrete action space to define possible portfolio positions. The values ranged from 0 to 1 and were incremented in steps of 0.1. This approach helped quantify the portfolio value allocation between two distinct assets: U.S. Dollars and Ethereum (ETH). When the position value was set to 0, the entire portfolio value was held in Dollars, indicating a conservative, non-exposure stance towards cryptocurrency. On the other hand, a position value of 1 indicated a total commitment to Ethereum, reflecting a portfolio entirely invested in this cryptocurrency.

However, when it came to implementing the Soft Actor-Critic (SAC) algorithm, we chose to use a finer gradation in the action space by reducing the step size to 0.01. This adjustment was crucial for a couple of reasons. Firstly, the smaller step size allowed the SAC algorithm to operate more like a continuous action space. This was particularly advantageous in the SAC framework, which thrives in environments where actions can be adjusted more precisely. Secondly, this finer gradation allowed for more nuanced decisions by the AI agent, facilitating a more sophisticated and potentially more profitable portfolio management strategy. Allowing the AI to make smaller adjustments between full Dollar or full Ethereum holdings could better navigate the volatile cryptocurrency market, identifying and capitalizing on subtle market movements that a coarser action space might miss.

Overall, this approach aimed to balance the benefits of discrete and continuous action spaces. By harnessing both strengths, we enhanced the AI agents' ability to optimize portfolio allocations in a highly dynamic and uncertain financial environment.

B. Double Deep Q-Learning Network

In our approach to reinforcement learning, we employed a Double Deep Q-Network (DDQN) model, which distinguishes itself from a basic deep Q-learning network through its dual-network architecture. This model comprises two separate neural networks: the online network, which evaluates the current states, and the target network, which assesses potential actions. This bifurcation is a strategic move to mitigate the value overestimation issue commonly encountered in simple deep Q-learning networks.

1) Network Update Strategy:
In a typical DDQN model, the target network is not trained as frequently as the online network. Initially, we synchronized the two networks every five episodes. However, the frequency of updates was a critical hyper-parameter requiring careful tuning to ensure the model's efficiency and accuracy. Too many updates could lead to overestimating values, undermining the DDQN's advantage over traditional Q-Learning. Conversely, infrequent updates could result in a lack of correlation between the two networks, hindering effective score optimization.

We adopted a soft update method between the online and target networks to address this issue instead of periodic synchronizations. This method involved using a tau value of 0.005 to update the target network gradually and continuously toward the online network's parameters. This approach ensures a more stable learning process and reduces the risk of over-estimations.

2) Action Selection:
We initially considered using Thompson sampling as the method for action selection in our Double Deep Q-Network (DDQN) agent. Thompson sampling is known for effectively balancing exploration and exploitation, making it a promising approach. However, implementing this method posed a significant challenge in re-implementation, as it required frequent data transfers between the GPU and the CPU for every action selection. This introduced computational inefficiency and significantly increased the overall training time.

Therefore, we decided to pivot to the epsilon-greedy strategy for action selection. The primary advantage of this approach was its ease of development and implementation, which seamlessly fit into our existing framework. Epsilon-greedy is also known for its straightforward yet effective mechanism in managing the trade-off between exploring new actions and exploiting known rewarding actions. To optimize this method further, we implemented a multiplicative epsilon decay with a factor of 0.95. This adjustment was made to accelerate the decay rate of epsilon, allowing for a quicker transition from exploration to exploitation.

This faster decay rate was particularly crucial given that the environment was the limiting factor in our implementation. By speeding up the epsilon decay, we ensured that our DDQN agent quickly learned to focus more on exploiting the best-known actions, making the learning process more efficient and effective. The decision to employ epsilon-greedy, coupled with the adjusted decay rate, was instrumental in enhancing the performance of our DDQN agent, enabling it to adapt more rapidly to the complexities of the environment while reducing overall training times.

3) Optimizer and Model Saving:
We opted for the Adam optimizer instead of a custom learning rate scheduler due to its efficiency in adjusting the learning rate. The Adam optimizer is known for its adaptability as it adjusts the learning rate throughout the training process based on the model's behavior, eliminating the need to tune the rate manually. We ensured that all relevant data, including environment variables, agent parameters, and auxiliary data that could influence the learning outcome, were meticulously maintained in a comprehensive log during the learning phase. To achieve this, we used the "dill" module, which is proficient in serializing complex Python objects. After each training episode, the system automatically saved all the data to a designated file, allowing us to resume training from the last saved state in case of any unexpected errors that could cause the training process to halt prematurely. This practice proved particularly beneficial in reducing downtime and ensuring the continuity and efficiency of our model's training regime.

C. Soft Actor-Critic Algorithm

SAC, which stands for Soft Actor-Critic, is an off-policy algorithm that optimizes a stochastic policy in an entropy-augmented reinforcement learning context. The algorithm introduces an element of randomness into the actions the policy selects to encourage more extensive exploration and, ultimately, maximize the expected return from the environment.

A single actor-network outputs a probability distribution over potential actions in the SAC architecture. This feature is crucial as it helps the algorithm balance exploring new actions and exploiting known rewards. Additionally, SAC uses two critic networks that work in tandem to compute the action-value function. These critic networks evaluate the potential future rewards of actions the actor-network takes, guiding it towards more effective strategies.

Combining these elements makes SAC particularly effective in complex environments where the balance between exploration and exploitation is critical for achieving optimal performance.

D. Important Metrics and Custom Reward Function

Important metrics to evaluate the performances of the models are as follows:

1) Portfolio Return Percentage:
Demonstrates how much profit the agent made during the episode.

2) Return Difference:
Demonstrates the difference between portfolio and market returns during the episode. It helps us to understand whether the specific actions taken by the agent are optimal compared to the change in the market.

3) Position Change Percentage:
Demonstrates how frequently the agent changes its position in the market. It helps us to analyze the stability of the model.

E. Reward Function

We used a custom reward function to evaluate the performances of our models:

1) Log Portfolio Change: Natural logarithm of the current portfolio value divided by the previous portfolio value. This is calculated using the following formula, where r_n is the market value at the time step n.

$$r_{n,portfolio} = ln(\frac{p_n}{p_{n-1}})$$

2) Log Market Change: Natural logarithm of the current market value divided by the previous market value. This is calculated using the following formula, where m_n is the market value at the time step n.

$$r_{n,market} = ln(\frac{m_n}{m_{n-1}})$$

The final reward function is calculated as follows:

$$r_n = 1000 * (r_{n,portfolio} - r_{n,market})$$

IV. CHALLENGES

In this project, we faced several challenges that tested the limits of our approaches and our understanding of the stock market environment. One of the most important challenges was the financial markets' complexity and stochastic nature, which made it difficult to design algorithms that could generalize well to unseen data. We also struggled with the high dimensionality of market data, which required careful feature selection and data pre-processing to ensure meaningful input to our models.

The computational demands were significant with both DDQN and SAC algorithms. Parameter tuning was crucial because the performance of the algorithms heavily depends on the fine-tuning of hyper-parameters. Additionally, the risk of over-fitting was a big problem as models that performed well on training data can fail to show that success in live trading scenarios. Another notable challenge was the balance between exploration and exploitation, particularly for SAC, which consistently struggled to capitalize on profitable opportunities.

V. RESULTS

A. Double Deep Q-Learning Network

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%, indicating 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 was the correct choice, enabling a successful balance between investigating new processes 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.

We obtained the highest training return with 1626.98% on episode 950, corresponding to a return difference of 1264.22% with a position changes metric of 51.58%. We got the highest evaluation return with 7.65% on episode 532, corresponding to a return difference of -10.42%, with a position changes metric of 48.26%.

Overall, 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. return differences" and "episode vs. scores" graphs that we obtained are as follows.

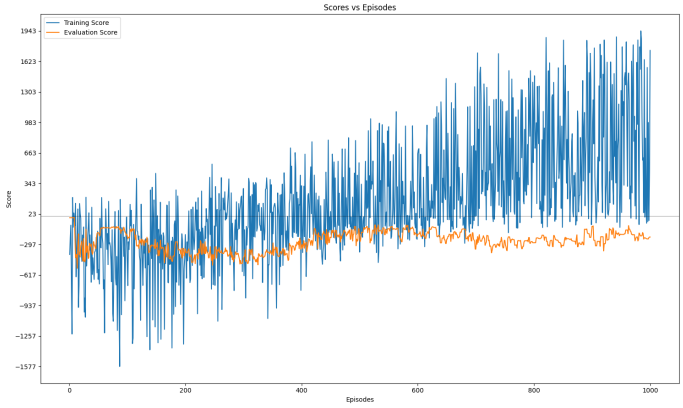

Fig. 1. DDQN Episode vs. Scores (%)

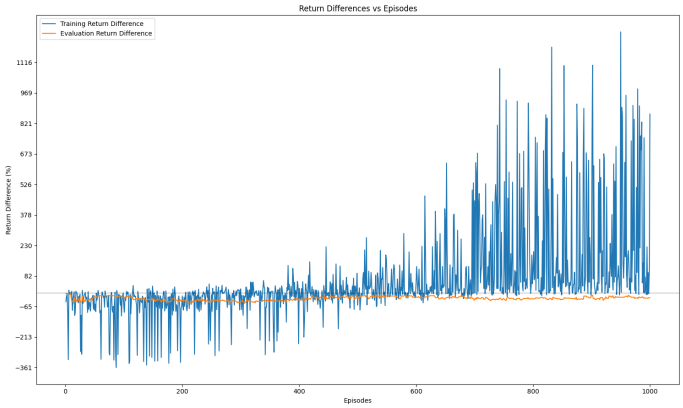

Fig. 2. DDQN Episode vs. Return Differences (%)

B. Soft Actor-Critic

During its training process, the SAC agent went through several episodes that provided unique insights from historical market data. However, the SAC agent faced considerable difficulty adapting to market complexities compared to the DDQN agent, evidenced by its lower performance. The SAC agent's initial trend of negative returns was more persistent, highlighting the heightened challenge and steeper learning curve in financial market trading.

The SAC agent's training was further complicated by its nuanced approach to action selection, which, while theoretically promising, proved less effective in practice. The agent struggled to find an optimal balance between exploring new strategies and capitalizing on profitable actions. This was partly due to its intrinsic policy adjustment mechanism, which did not facilitate as rapid a shift from exploration to exploitation compared to the DDQN agent's epsilon-greedy method.

The SAC agent's progress was less pronounced in critical indicators like portfolio return. Over several episodes, the agent failed to match or surpass market baselines. The SAC agent's trading strategy was also less stable, with higher frequently

changing positions indicative of a less settled trading approach, leading to increased transaction costs and diminished returns.

Even though there was an upward trend in the SAC agent's performance, the best training return was significantly lower, peaking at a percentage of 223.10%, corresponding to a return difference metric of -130.65% with a position changes metric of 96.83%. The SAC agent's evaluation performance also lagged, with the highest evaluation return of -9.88%, corresponding to a return difference of -27.95% and a position changes metric of 93.03%.

Overall, the SAC agent represents a valuable exploration in the realm of reinforcement learning models for financial trading; its performance in this project was low compared to the DDQN agent. The SAC agent's difficulty in training and lower overall results underline the challenges inherent in applying advanced AI techniques to the unpredictable and complex world of financial trading.

The "episode vs. return differences" and "episode vs. scores" graphs that we obtained are as follows.

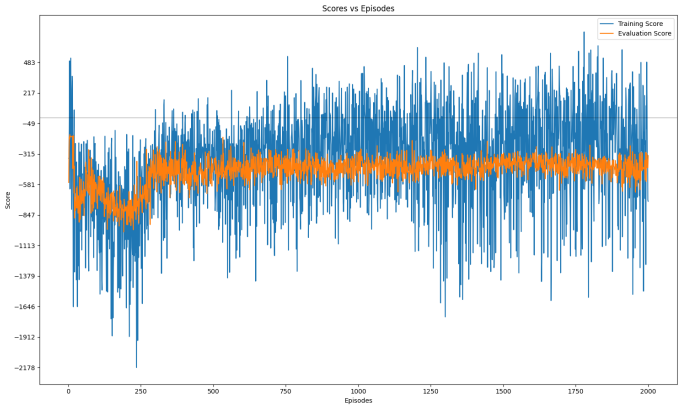

Fig. 3. SAC Episode vs. Scores (%)

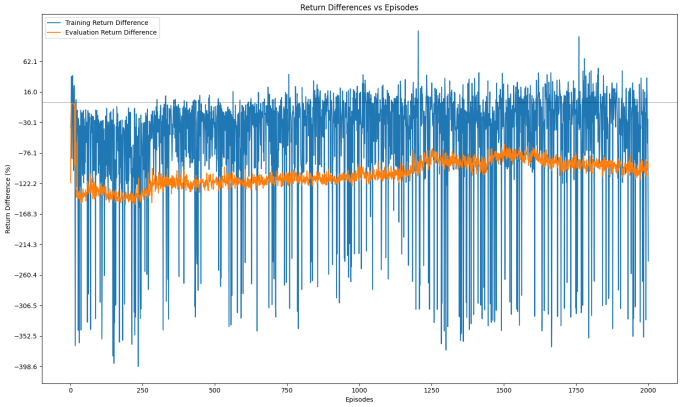

Fig. 4. SAC Episode vs. Return Differences (%)

VI. COMPARISONS

For our project, we re-implemented two advanced reinforcement learning algorithms, Double Deep Q-Learning Network (DDQN) and Soft Actor-Critic (SAC), in the context of stock market trading. The DDQN algorithm exhibited superior performance, and its ability to effectively adapt to different market conditions was better. It achieved a maximum portfolio return of 72.78%, showing its robustness and efficiency in financial trading scenarios.

On the other hand, although being creative in balancing exploitation and exploration, the Soft Actor-Critic algorithm found it difficult to maintain stable performance. It was clearly having trouble adjusting to the complex dynamics of the stock market, which led to lower returns and higher transaction costs due to frequent position changes.

The comparison of the DDQN and SAC Episode vs. Scores graphs reveals distinct performance profiles for each algorithm. The DDQN graph shows a training score that fluctuates significantly but trends upward over time. This shows learning and adaptation, with evaluation scores improving but less volatile. This suggests more consistent performance during testing. The SAC graph displays a similar fluctuation in training scores. Still, the upward trend is less, and the evaluation scores remain relatively straight, suggesting a struggle to optimize performance during testing phases. Both algorithms demonstrate the characteristic exploration and exploitation behavior of reinforcement learning but with different degrees of success in a stock market environment.

The DDQN Episode vs. Return Differences graph shows a more volatile training return difference with regular spikes. It indicates potential over-fitting to the training environment. However, the evaluation return difference remains relatively stable and close to zero. This shows that the model can generalize to unseen data consistently. In contrast, the SAC Episode vs. Return Differences graph displays less volatility in training return differences but a generally declining trend in evaluation return differences. This demonstrates that the model might not be learning effectively or generalizing well to the evaluation environment. Both graphs suggest that DDQN might be more suitable for this application, given its stability in unseen environments, which is crucial for real-world trading.

Overall, the DDQN outperformed SAC in our project, demonstrating greater stability, profitability, and adaptability in the unpredictable stock market trading environment. The significance of algorithm selection in the creation of AI-driven financial trading systems is highlighted by this contrast.

VII. CONCLUSIONS

Our project about using artificial intelligence in stock market trading has been both difficult and informative. Our goal was to evaluate and understand the effectiveness of two different reinforcement learning algorithms: the Soft Actor-Critic (SAC) and the Double Deep Q-Learning Network (DDQN).

Initially, we prepared the ground by analyzing the theoretical foundations of every algorithm. The SAC provided a state-of-the-art method with an entropy-based exploration strategy. The DDQN, with its dual network architecture, promised improved learning stability. Our techniques were tested in the context of an unstable and volatile stock market environment.

Our findings were instructive. The DDQN algorithm showed remarkable durability and adaptability. It resulted in notable portfolio returns. It demonstrated a quantitative level of ability to navigate the unstable conditions of the market. On the other hand, the SAC frequently struggled with the volatile forces of the market, and it had less-than-ideal results.

By examining different performance measures and graphical studies, we were able to understand how each algorithm behaved. Our report's challenges section provided a straightforward description of the difficulties we encountered, such as issues with data pre-processing. Trials and errors, from overfitting dangers to the difficult task of parameter tuning, it was a difficult job to adapt these algorithms to the financial domain.

As we summarized our findings, we understood the difficulties of using AI in a field as dynamic as the stock market. While the SAC's sensitivity to parameter changes and difficulty with compelling exploration raised concerns about its usage in real-world issues, the DDQN's constant performance throughout the training and evaluation phases highlighted its practical potential.

This project has set the groundwork for further research in the future. It emphasizes how important it is for trading algorithms to constantly evolve in order to become both technically and financially stable. We imagine a time when combining different AI approaches could result in hybrid models that combine the best features of SAC's exploration powers and DDQN's stability.

To sum up, the project has demonstrated the revolutionary potential of artificial intelligence in the financial industry. It has created new opportunities for innovation. For those willing to combine artificial intelligence with the skill of financial trading, the future is full of opportunity.

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).