Double Dueling Deep Q Reinforcement Based Trading Agent:

A Sentiment, Search Trends, Financial Data, and Blockchain Information based Approach

Eric B Martin
Department of Computer Science
Colorado State University
Fort Collins, CO, USA
Email: ebmartin@colostate.edu

Abstract

This paper presents a reinforcement learning-based trading agent designed to navigate the volatile cryptocurrency market, with a focus on the Ethereum market. Building upon the foundation laid by David Gallo's thesis [1], which utilized Double Deep Q-Networks (DDQN) and Dueling Deep Q-Networks, the proposed trading agent incorporates several improvements to enhance its performance and adaptability. Key upgrades to the architecture include the implementation of LeakyReLU activation functions and batch normalization, which contribute to the stability and generalization performance of the underlying neural networks. The agent is also equipped with a more diverse set of features, such as sentiment data, search trends, financial indicators, and blockchain data, which allow it to make more informed decisions in the face of market volatility. To further improve the agent's decision-making capabilities, a curriculum learning approach is employed, progressively exposing the agent to increasingly complex market scenarios and fostering better adaptability. The results of the experiments show the promise of the trading agent in generating profitable trading strategies and adapting to prevailing market trends. By building upon and enhancing the original agent proposed in Gallo's thesis, this research demonstrates the potential of reinforcement learning and artificial intelligence in developing robust and adaptive trading strategies in the ever-changing cryptocurrency market.

CCS Concepts

Computing methodologies → Reinforcement learning
Computing methodologies → Q-learning
Computing methodologies → Neural networks
Computing methodologies → Artificial intelligence
Applied computing → Finance
Applied computing → Electronic commerce

Keywords

Ethereum, Reinforcement Learning, Deep Q-Networks (DQN), Double Deep Q-Networks (DDQN), Dueling Deep Q-Networks, Prioritized Experience Replay, Cryptocurrency Trading, Algorithmic Trading, Trading Bots, Curriculum Learning, Artificial Intelligence

I. Introduction

The rapid growth of cryptocurrencies and their adoption in financial markets combined with the surge in artificial intelligence and machine learning adoption have garnered significant interest in the development of automated trading strategies. In this paper, I present a reinforcement learning-based trading agent that is specifically designed to navigate the volatile and unpredictable nature of the cryptocurrency market. My primary focus is on the Ethereum market, which is the second-largest cryptocurrency by market capitalization, but I have only limited the bot to this asset for simplicity as I plan on introducing multiple assets in the future. The proposed trading agent's base architecture is based on a thesis by David Gallo at the Toulouse Business School which utilized a combination of the reinforcement learning techniques, Double Deep Q-Networks (DDQN) and Dueling Deep Q-Networks. [1] Starting from his base architecture, I implemented various improvements as well as identified and corrected a mistake in original trading functionality of the thesis. I started by implementing a Prioritized Experience Replay to effectively estimate the optimal trading actions in a dynamic market environment. I then improved the network architecture by implementing Leaky ReLU activation functions, dropout, and batch normalization, to improve the stability and generalization performance of the underlying neural networks. Furthermore, since the original thesis only provided a proof of concept that was not applicable in a real-world scenario, my approach addressed this by incorporating curriculum learning to gradually expose the agent to increasingly complex market scenarios, fostering better adaptability, decision-making skills, and real-world integration. By supplying the agent with a variety of important features such as sentiment, search trends, and blockchain data the agent will hopefully be able to better predict the volatile environment. The results of my experiments thus far demonstrate the potential effectiveness of the trading agent in generating profitable trading strategies, even in the face of highly volatile market conditions.

II. Product Description

In this paper, I introduce a sophisticated Ethereum trading bot that leverages the power of deep reinforcement learning to make informed trading decisions based on a plethora of market features. The trading bot is designed to operate autonomously, utilizing historical price, twitter and telegram sentiment, Google search trends, financial data, and Ethereum blockchain [2] data to optimize its trading strategy. The core of the trading bot is a double dueling deep Q-network (DDDQN) agent, which has shown promise to be effective in solving complex decision-making problems. To improve the learning efficiency and overall performance of the trading bot, I employ curriculum training,

a technique that progressively introduces more challenging tasks as the agent becomes more proficient. This allows the agent to be able to learn how to trade, one step at a time, in order to gradually adapt to the dynamics of the Ethereum market and develop a robust trading strategy that can generalize well to unseen market conditions. The trading bot is designed to interact with the Coinbase API, enabling seamless integration with the popular cryptocurrency exchange. This integration allows the bot to execute trades in real-time, capitalizing on market opportunities as they arise. Furthermore, the trading bot takes trading fees into consideration, ensuring that its trading decisions account for the costs associated with each transaction. By incorporating trading fees into the decision-making process, the bot aims to maximize its net returns while minimizing the impact of transaction costs on its overall performance.

III. Motivation

This project was motivated by my artificial intelligence and machine learning graduate curriculum at Colorado State University. The concept for the architecture was inspired by the thesis by David Gallo at the Toulouse Business School which utilized a DDDQN and twitter sentiment to produce a trading bot. [1] The bot Gallo proposed in his thesis was profitable but was not applicable for real trading scenarios. In order to build off of this project I implemented various improvements which are seen in this paper.

IV. Project Complexity/Challenges

Training a reinforcement learning trading bot poses numerous challenges, primarily due to the complex and dynamic nature of financial markets. One major obstacle in developing an effective RL trading bot is the design of an appropriate reward function, which should effectively guide the agent towards profitable trading decisions while avoiding undesirable behaviors. Crafting a reward function that accurately captures the intricacies of the trading process, such as transaction costs, risk management, and market fluctuations, can be an arduous task. To address this issue, I employ a curriculum-based approach, progressively introducing the bot to more complex trading scenarios. This method enables the trading bot to gradually learn realistic trading techniques by mastering simpler tasks before tackling more challenging ones. For example, to learn how to buy low sell high, a simple curriculum was created that allows the bot to buy whenever it wants as long as the purchase is less than its balance. The balance does not change after a buy though. The balance only changes when the agent sells. This dramatically increased the agent's ability to learn profitable techniques. By adopting a curriculum-based approach, I aim to enhance the bot's learning efficiency and improve its ability to adapt to the ever-changing market conditions, ultimately achieving a more robust and reliable trading strategy.

V. Project Goals

The primary objectives of this research project are to develop, implement, and evaluate a reinforcement learning-based trading agent capable of navigating the complex and dynamic landscape of financial markets.

The goals of this project are as follows:

1. **Curriculum-based Learning:** Design and implement a curriculum-based approach to progressively train the trading agent, enabling it to acquire profitable trading strategies by first mastering simpler tasks and gradually advancing to more complex scenarios. This approach will enhance the agent's learning efficiency and adaptability, ensuring that it can effectively handle a diverse range of market conditions.

2. **Data Acquisition and Processing:** Develop an integrated system to gather and process the necessary financial data, including historical and real-time information, by utilizing APIs and web scraping techniques. This system will provide the trading agent with access to hourly and daily data, empowering it to make informed decisions based on accurate and up-to-date market information.

3. **Real-time Trading Integration:** Connect the trading agent to the Coinbase Pro API, enabling it to execute real-time trades within the Coinbase Pro testing sandbox. This integration will provide the agent with an environment in which to refine and validate its trading strategies before deployment in a live market setting.

4. **Live Market Deployment:** Deploy the trained trading agent in the real trading market, allowing it to apply the knowledge and strategies acquired during the curriculum-based training phase to real-world trading scenarios, ultimately aiming to achieve consistent profitability.

5. **Asset and Trading Pair Expansion:** Expand the scope of the trading agent to include additional assets and trading pairs, further enhancing its versatility and adaptability. This expansion will enable the agent to capitalize on a broader range of market opportunities, increasing its potential for generating profits and mitigating risks.

6. **Feature Expansion:** Expand the current features the agent has available to encompass buy/sell walls and other useful fintech information in order to further increase the agent's potential.

VII. Features

1. **Google Search Trends:**

   There may be correlations between Google search trends and price movements for Ethereum and Bitcoin. A trading bot can analyze historical search trends data alongside price data to identify patterns or relationships that might be useful for predicting future price movements. Search trends can sometimes also act as early indicators of significant events or news related to a company or industry. For example, a sudden spike in searches for Coinbase could signal an upcoming wave of new

investors looking into entering the crypto space. A trading bot can leverage this information to make more informed decisions before the broader market reacts to the news.

Utilizing the unofficial Google Trends API called pytrends [7], daily and hourly trend data is obtained for the following queries:

Bitcoin

Ethereum

ETH

Crypto

Cryptocurrency

Coinbase

BTC

2. Sentiment Data:

Daily Twitter and Telegram sentiment was obtained from IntoTheBlock. [4] The daily sentiment was used in the training of the agent because obtaining hourly sentiment was too cost prohibitive at this time. Therefore, the agent will only have yesterday's sentiment to help with daily hourly decisions. Hourly data would be preferred as it would allow for more real-time decision making.

3. Price Data:

The agent has access to the hourly price data (low, high, close, open, volume) as well as the daily data from the previous day (low, high, open, close, volume). The agent also has access to the delta values of the previous day and hourly prices.

The agent also has access to the volume and volatility data of the two leading stable coins, USDC and USDT since their volumes and activity can have a correlation with the cryptocurrency economy.

4. Financial Data:

Various exponential moving averages (3-day, 5-day, 7-day, 14-day, 20-day, and 50-day) are supplied to the agent as well as the relative strength index (RSI).

Exponential Moving Averages (EMAs): EMAs are a type of moving average that gives more weight to recent price data, making them more responsive to recent price changes. Providing the agent with different timeframes of EMAs (3-day, 5-day, 7-day, 14-day, 20-day, and 50-day) allows the agent to analyze short-term, medium-term, and long-term trends in the market. These EMAs can help the agent identify:

Trend direction: EMAs can help the agent determine if the market is in an uptrend, downtrend, or sideways trend by analyzing the direction of the EMAs. In general, if the short-term EMAs are above the longer-term EMAs, it may indicate an uptrend, while the opposite scenario may suggest a downtrend.

Support and resistance levels: EMAs can act as dynamic support and resistance levels. When the price is above the EMA, it can act as a support level, while when the price is below the EMA, it can act as a resistance level. The agent can use this information to make more informed buy-and-sell decisions.

Trend reversals and crossovers: When shorter-term EMAs cross above or below longer-term EMAs, it can signal a potential trend reversal. These crossover points can be valuable for the agent in timing its entry and exit points in the market.

Relative Strength Index (RSI): The RSI is a momentum oscillator that measures the speed and change of price movements. It ranges from 0 to 100 and can help the agent identify:

Overbought and oversold conditions: RSI values above 70 typically indicate overbought conditions, suggesting that the asset may be overvalued and due for a price correction. Conversely, RSI values below 30 usually signal oversold conditions, implying that the asset may be undervalued and due for a price rebound. The agent can use these signals to time its buying and selling decisions.

Divergences: When the price and RSI show diverging trends, it may signal a potential reversal in the market. For example, if the price is making higher highs while the RSI is making lower highs, it could indicate that the bullish momentum is weakening, and a trend reversal might occur. The agent can use these divergences to adjust its trading strategy accordingly.

I have limited the indicators to these two for training's sake but once the agent curriculum is solved, I would like to implement Bollinger bands, stochastic oscillator, average true range, on-balance volume, Ichimoku cloud, and moving average convergence divergence indicators.

5. Blockchain Data:

Since the blockchain is a public ledger, I am able to find and supply various blockchain data to the agent. There is a plethora of data but for my current progress, I have limited the data supplied to the agent to the following daily data:

Daily Active Ethereum Addresses

Daily Active ERC20 Token Transfers

Transactions Volume

Average Transaction Fee

Average Block Size

This information can be used to supply the agent with information about the blockchain ecosystem that runs Ethereum and ERC20 tokens. If this information could be provided hourly, the bot would be able to improve substantially. Access to hourly data is currently out of my budget at the moment but I intend to have access to this data in the future.

6. Trading Flags:

I currently do not have trading flags implemented as an input to the agent as it may inadvertently cause the agent to trade based on traditional buy-sell signals as opposed to identifying new trading strategies. This will be looked into later though.

VII. Methodology

This project utilized Meta's PyTorch [5] due to its ease of use and the simple GPU support as opposed to Google's Tensorflow [6].

1. Dueling Double Deep Q-Network (DDDQN):

The Dueling Double Deep Q-Network (DDDQN) is an advanced variant of the standard Deep Q-Network (DQN) that aims to improve the performance and stability of reinforcement learning algorithms. It combines two key techniques: Dueling Architecture and Double Q-Learning.

Dueling Architecture:

The dueling architecture introduces two separate streams within the Q-network: a value stream (V(s)) and an advantage stream (A(s, a)). The value stream estimates the value of being in a particular state, while the advantage stream estimates the advantage of taking a specific action in that state. These two streams are combined to produce the Q-values (Q(s, a)). This separation enables the DDDQN to learn which states are valuable without having to learn the effect of each action in every state. [3] The Q-values are computed as follows:

$$Q(s,a) = V(s) + A(s,a) - mean(A(s,a))$$

In the code provided, the dueling architecture is implemented using separate layers for the value and action hidden layers (dv1, da1) and the value and action output layers (dv2, da2). The Q-values are computed by combining the value and advantage streams in the forward method.

Double Q-Learning:

Double Q-Learning is an improvement over standard Q-Learning that addresses the issue of overestimation of Q-values. In standard Q-Learning, the same network is used to both select and evaluate actions, which may lead to over-optimistic Q-value estimates. Double Q-Learning, on the other hand, uses two separate networks: a target network and an online network. The online network is used to select actions, while the target network is used to evaluate the actions. This separation helps mitigate the overestimation issue, leading to better convergence and stability.

In the DDDQN, the target network is updated periodically (by copying the weights from the online network) to maintain a stable learning process. This updating process is not shown in the provided code but is an essential part of the overall DDDQN algorithm.

The exact architecture used in this project is seen below in figure 1.

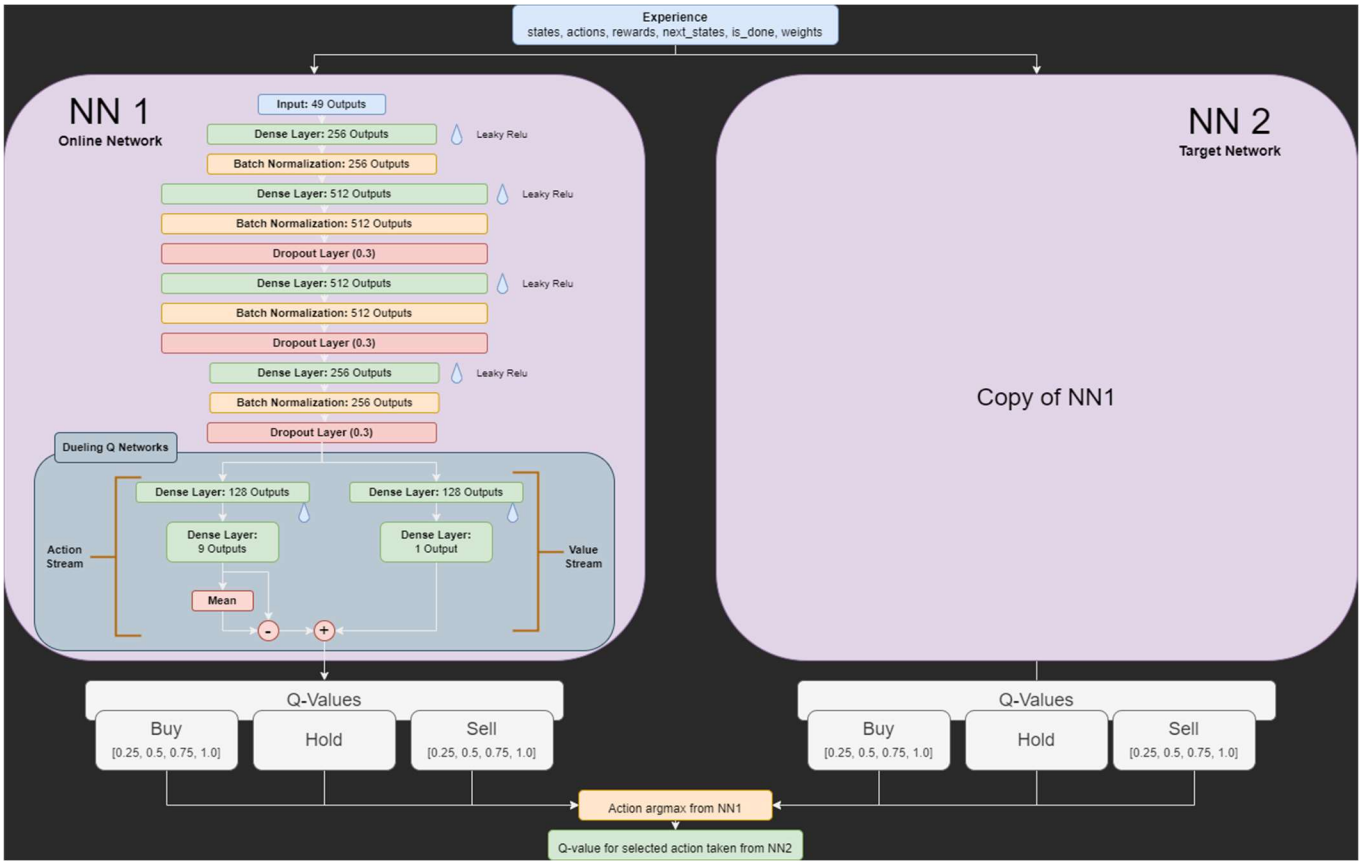

Figure 1: Architecture of the Double Dueling Deep Q-Network

2. Prioritized Experience Replay (PER):

Prioritized experience replay was introduced as an improvement over regular experience replay to address some of its limitations. In regular experience replay (seen in the thesis by Gallo), experiences are stored in a buffer and sampled uniformly at random during training. However, this approach does not take into account the fact that some experiences may be more valuable for learning than others. In order to addresses this limitation, PER was used to assign a priority to each experience based on the magnitude of the temporal-difference (TD) error. The TD error represents the difference between the predicted Q-value and the target Q-value, which is an estimate of the actual Q-value. Experiences with higher TD errors correspond to situations where the current model's predictions deviate significantly from the target values, indicating a higher potential for learning making it particularly advantageous for applications like stock trading.

Some of the advantages PER can bring a trading agent are:

Better utilization of experiences: In standard experience replay, experiences are sampled uniformly from the replay buffer, which means that all experiences have an equal chance of being selected. However, not all experiences contribute equally to learning. Some experiences might be more informative, while others might be relatively insignificant. By prioritizing experiences based on their temporal-difference (TD) errors, PER ensures that more informative experiences have a higher chance of being sampled, allowing the model to learn more effectively.

Faster convergence: By focusing on learning from significant experiences, PER can potentially speed up the learning process, helping the model to converge faster. This can be especially beneficial in stock trading applications, where timely decision-making is crucial for taking advantage of market opportunities.

Handling non-stationarity: Financial markets are highly dynamic and non-stationary. The underlying relationships between stocks and their factors can change over time. Regular experience replay treats all experiences equally, which might not be ideal when dealing with non-stationary data. PER, on the other hand, dynamically adjusts the sampling probabilities based on the TD errors, making the learning process more adaptive to changing market conditions.

Balancing exploration and exploitation: In stock trading applications, it is essential to strike a balance between exploration (trying new actions) and exploitation (using learned knowledge). PER can help maintain this balance by prioritizing experiences with high TD errors, which are typically associated with uncertain or surprising outcomes. This encourages the model to explore these areas more thoroughly while still exploiting its existing knowledge.

3. Agent Class:

The `Agent` class is a reinforcement learning agent designed to learn stock trading strategies through a series of curriculums. The agent is initialized with various parameters such as window size, gamma, number of episodes, learning rate, minimum epsilon, and update interval. It also initializes the device, portfolio, epsilon, training step counter, memory, batch size, online and target networks, optimizer, scheduler, and the holding time.

The agent is trained using curriculum learning, which is divided into multiple parts (Curriculum 1, 2, 3, 4, 5). In each curriculum, the agent learns more detailed trading functionalities and is exposed to different reward systems until it is eventually able to trade on the real-time marketplace. Currently, only curriculums 1 and 2 are completed.

The Agent class has several key methods:

1. *get_action*: This method returns the action the agent should take based on the current state. It uses an epsilon-greedy approach, which balances exploration and exploitation.
2. *update_target*: This method softly updates the target network by blending the weights of the online network and target network using the parameter tau.
3. *update_epsilon*: This method linearly decays the epsilon value from its initial value to the minimum epsilon value over the course of the total training steps.
4. *train*: This method trains the online network using the samples from the prioritized experience replay memory. It updates the target network periodically based on the update_interval. It also calculates the loss, applies the importance sampling weights, and updates the priorities in the buffer.
5. *save_model* and *load_model*: These methods are used to save and load the model parameters of the online and target networks.
6. *trade:* This method simulates the trading process based on the action taken by the agent, the current timestep, and the available data. It also calculates the reward for the agent.
7. *get_state:* This method constructs the state representation for the agent by using a sliding window over the input data.

The agent's learning process is divided into multiple curriculums, with each curriculum focusing on different aspects of the trading functionalities. As the agent progresses through the curriculums, it becomes more proficient at trading and is eventually able to trade in real-time.

VIII. Results

Currently, I have only successfully written the code for three curriculums. Out of those curriculums, I have gotten the agent to successfully learn curriculum one. Curriculum one is simple and is very close to the methodology that was utilized in Gallo's thesis [1].

Curriculum 1:

The curriculum in the original thesis (curriculum 1) is as follows:

Buying:

Reward: 0

The agent has a starting balance. The agent can purchase single units of Ethereum (full coins only) as long as they cost less than the agent's balance. The agent is only allowed to purchase one coin per day. The agent's balance is not reduced after a purchase (this is where the methodology in the thesis seems to sway from real-world application).

The agent gets zero reward for purchasing and once the agent purchases an Ethereum, it adds the coin to its inventory (a list of prices representing the cost of the coin it purchased). In this case, since the purchase is always a full coin, the inventory is simply a list of prices of the coin when it was purchased.

For example:

The agent has a $1000 balance and an inventory of [ ]

The current cost of ETH is $300

The agent purchases 1 ETH at $300

The agent adds the purchase to the inventory which is now [300]

The agent's balance is still $1000

The agent purchases a coin on the next day for $320

The agent's balance is still $1000 and its inventory is [300, 320]

Holding:

Reward: -0.1

The agent can hold its assets but will have a reward of -0.1 to influence active trading and prevent holding. This is where my method differed from Gallo's. Gallo had a reward of 0 for holding. In my experience the agent would decide that holding is the best strategy in most cases if the reward was set to 0.

Selling:

Reward: Current ETH price – Inventory.pop()

The agent could sell if its inventory was not empty. If the agent sells, it pops the most recent purchase from the inventory and takes the difference of the current Ethereum price – the most recent purchase and this is its reward. The balance is then updated based on this difference. It is here that the balance can increase or decrease.

For example:

The agent has a balance of $1000

The agent has an inventory of [300, 320] and wants to sell.

The current price of ETH is $315

The agent sells and pops the most recent item from its inventory

The agent receives a reward of 315 – 320 which is -5

The agent's balance is then updated to $9995

It is in the sell method that I identified an issue in David Gallo's thesis where he stated on page 34,

"The inventory of the agent, denoted as I, was a first-in-first-out model, where units of Ether recorded in the inventory were sold in reverse order of purchase. This meant that the first unit of Ether sold was always the most recent unit purchased." [1]

This language is contradictory as it states that the inventory is a FIFO structure but then states that the ether sold is always the most recent purchase which would entail a LIFO structure. I then assessed his code and discovered that he was performing a FIFO. I tested both FIFO and LIFO methods and identified that a LIFO structure seems to work best.

Done:

The agent is done when its balance is less than the current price of Ethereum.

Curriculum 2:

In order to create a trading agent that is applicable to real-world situations, I have added stipulations in curriculum 2 to help the agent learn to manage new actions and scenarios. In curriculum 2 I have added the following:

- Trading fees.
- Buy and Sell option are expanded to [0.25, 0.50, 0.75, 1] to allow the agent to purchase and sell varying amounts of inventory.
- Holding penalty that grows per consecutive hour held.

These stipulations allow the agent more buying and selling flexibility, makes the agent need to consider trading fees, and pushes the agent to take action rather than hold.

Curriculum 3:

The agent now will have the following new stipulations:

- Reward based on accumulation of Ethereum.

This will teach the agent that collecting more of an asset is valuable.

Curriculum 4:

- Reward agent based on overall USD value of portfolio.

Curriculum 5:

- Balance is reduced after each buy action.
- Agent can purchase more than 1 coin per hour.
- Reward based on overall portfolio value.
- Minorly rewarded for following trading flags.

After curriculum 5, the agent should hopefully be able to fully function in a live marketplace. Curriculum 5 is adding a lot of new stipulations and therefor may need to be split into four more curriculums to ensure ease of training.

Results after curriculum 1 and 2 (daily):

Prior to converting the agent to an hourly based trading bot, I had trained the agent on daily data. Here are the results after the agent was trained on curriculum 1 and 2 for 50 episodes. Shown below are the results for the last episode (index started at 0 so 49 is the last episode.

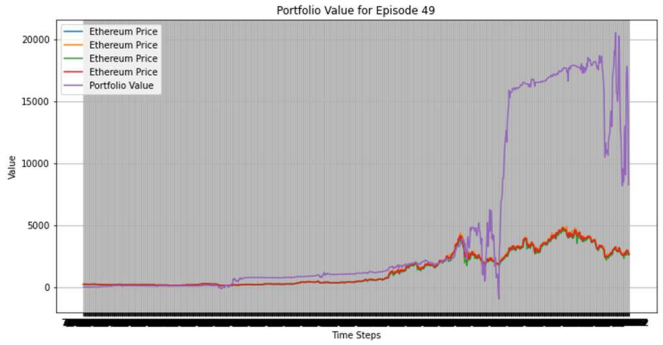

Figure 2: Portfolio value of training run on daily data after completing curriculum 1 and 2.

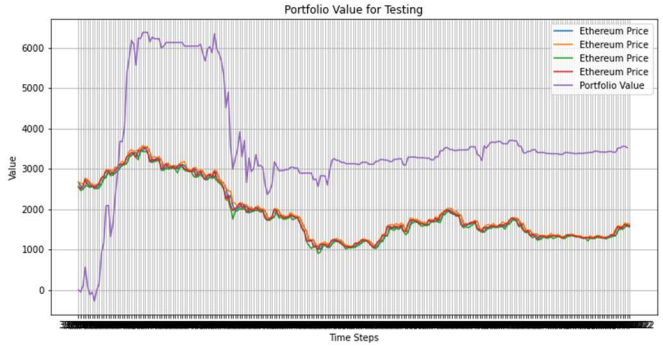

Figure 3: Profit of agent on testing data after completing curriculum 1 and 2.

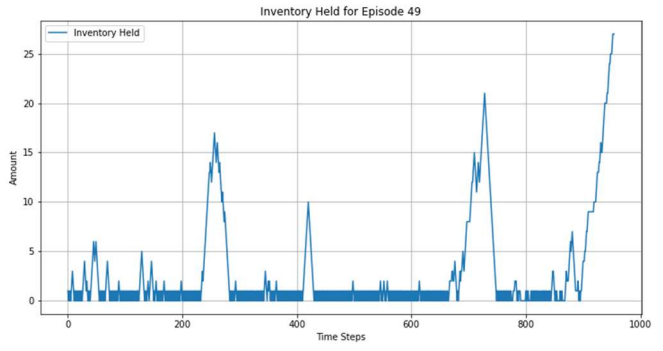

Figure 4: Inventory held by the agent during the training.

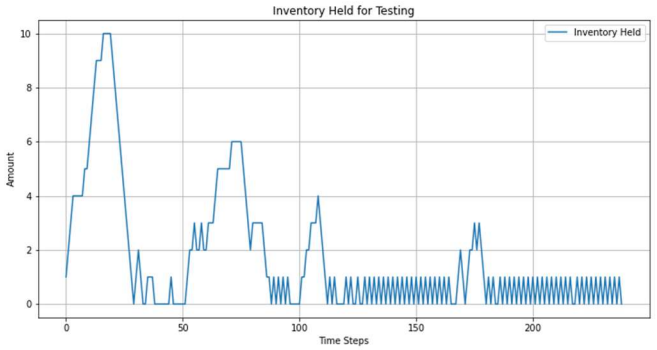

Figure 5: Inventory held by the agent during testing.

We can see that the agent made around $20,000 worth of profit in the training (figure 2) which may be symbolic of overfitting, but the market also was seeing all time highs in 2017 and 2020 which the bot was able to take advantage of. In the testing (figure 3), the agent still managed to make $3000 dollars profit even in a down trending market which is promising. By looking at the inventory held over time for both the training (figure 4) and testing (figure 5) we can see that the agent tended to accumulate inventory during the up trends and sell during downtrends. The agent also seemed to only hold 1 item in its inventory during the bear market as well which means it is actively day trading throughout the bear market.

Results after curriculum 1 (hourly):

Due to the massive increase in training time caused by the transition from 5 year historical daily data to hourly data, I have only had to opportunity to train curriculum 1. The results are as follows for the hourly agent after only being trained for 10 episodes with curriculum1:

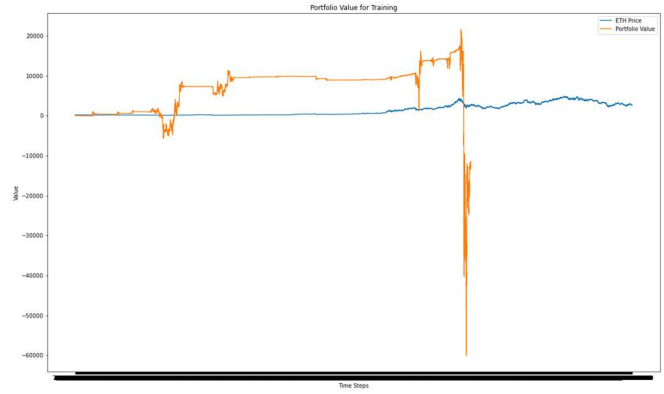

Figure 6: Training data portfolio value after training 10 episodes on hourly data with curriculum1.

Here we can see that the agent lost too much money and triggered the 'done' break method. This is not surprising as this agent was only trained for 10 episodes whereas the daily agent was trained on 50 episodes. But, even with this bad result, the agent performed decently on the testing data making a profit of around $1100 (figure 7).

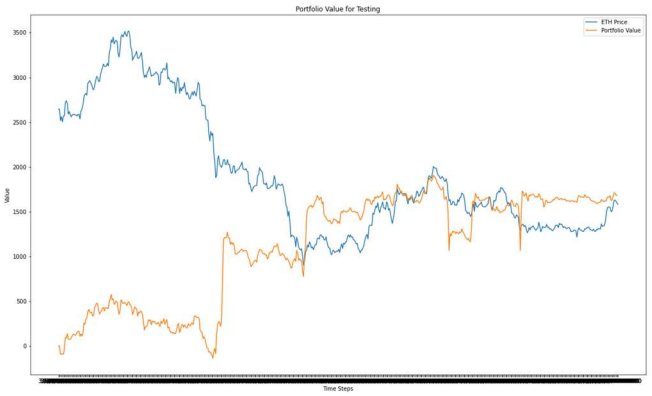

Figure 7: Testing profit of the hourly agent trained for 10 episodes with curriculum 1.

I have yet to run the hourly agent for the same number of episodes as the daily agent but based on these results, the outcome looks promising.

IX. Challenges

1. Training Time:

The current time to train effectively on a single Nvidia RTX 3080 is around 200 to 500 minutes. So, testing each tweak to the curriculum and training methodologies takes a long time. To mitigate this, I save the successful curriculum weights and load them before testing a new curriculum. Ensuring the agent is not overfit is also crucial for the success of the trading bot.

2. Curriculum Creation:

The current starting curriculum is simple, and the agent manages to learn quite well. Implementing new stipulations in the other curriculums is proving difficult for the agent to learn well. This means that the reward function or curriculum changes may need to be adjusted. This is proving very difficult due to the length of training.

3. Obtaining Hourly Data:

The data I have used in training thus far was obtained from a trial license through IntoTheBlock [4] and therefor I will not be able to supply the bot real time data once it is market ready. Finding an affordable source for hourly data is necessary for the success of this agent. I may be able to scrape the necessary data from websites, but I will need to look into this further.

X. Conclusion

The reinforcement learning-based trading agent presented in this paper has shown promising results in generating profitable trading strategies in the complex and dynamic landscape of the Ethereum market. By incorporating curriculum learning, the agent is able to adapt to a wide range of market conditions and improve its decision-making capabilities. However, there are still challenges to overcome, such as training time, curriculum creation, and obtaining hourly data. Future work will aim to address these challenges and expand the scope of the trading agent to include additional assets, trading pairs, and features. This research demonstrates the potential of reinforcement learning and artificial intelligence in developing robust and adaptive trading strategies in the ever-changing cryptocurrency market.

REFERENCES

[1] D. Gallo, “Algorithmic Cryptocurrency Trading using Sentiment Analysis and Dueling Double Deep Q-Networks,” thesis, 2022 https://www.monkeydg.com/assets/docs/David_Gallo_MSc_Thesis.pdf

[2] Buterin, V. (2014). Ethereum: A next-generation smart contract and decentralized application platform. Retrieved from https://ethereum.org/669c9e2e2027310b6b3cdce6e1c52962/ Ethereum Whitepaper - Buterin 2014.pdf

[3] Wang, Z., Schaul, T., Hessel, M., van Hasselt, H., Lanctot, M., & de Freitas, N. (2015). Dueling network architectures for deep reinforcement learning. arXiv. Retrieved from https:// arxiv.org/abs/1511.06581 doi: 10.48550/ARXIV.1511.06581

[4] “API Documentation,” Intotheblock.com, https://www.intotheblock.com/api (accessed May 10, 2023).

[5] “Pytorch Documentation,” PyTorch, https://pytorch.org/ (accessed May 10, 2023).

[6] “Tensorflow,” TensorFlow, https://www.tensorflow.org/ (accessed May 10, 2023).

[7] GeneralMills, “Generalmills/Pytrends: Pseudo API for google trends,” GitHub, https://github.com/GeneralMills/pytrends (accessed May 10, 2023).

About the Author:

Eric Burton Martin (ebmartin@colostate.edu) is a student at Colorado State University and is currently pursuing a master’s degree in computer science. He previously obtained a chemistry degree in 2013 and has worked as an analytical chemist. Currently, his interests are in AI/ML, cybersecurity, and blockchain technology.