Reinforcement Learning Project - RL4Trading

Julien GADONNEIX, Samuel GAUDIN, Mathias GRAU

Abstract—**This project is an attempt to train an autonomous agent using a deep Q-network. The agent learns to make trading decisions through interaction with a simulated stock trading environment, employing strategies like epsilon-greedy for action selection and using a replay buffer to facilitate learning from a diverse set of experiences. The goal is to maximize the total wealth over time through effective trading strategies. This problem is already tackled by extremely competitive and ruthless companies which possess almost infinite computation resources. However, it was a great opportunity for us to deal with a real-world problem while trying to make the most of what we learned through the INF581 course and some additional sources. The complexity of our project increased linearly and we tried to adapt it to our results and comments that we got. While it was evolving, we split ourselves into different tasks to have a final version with an overview of the trading problem.**

I. Introduction

We decided to look at a specific, real-world environment, and implement different strategies, mainly based on deep neural networks. The chosen environment the market stock exchange. We designed a simulation of daily prices of various famous stocks which can be seen as time series. In this project, we mostly looked at deep-Q network (DQN) learning whose learning is based on the off-policy temporal difference. More precisely, we looked at a more recent and effective version of this reinforcement learning (RL) strategy which is the double DQN (DDQN) and some variants of it such as Dueling DDQN (DDDQN). This method is indeed well-suited for our problem where we can easily discretize our action spaces while having a continuous state space by definition of the market prices. As a trader would do, our agent is supposed to achieve the best choices in terms of buying and selling in order to maximize its wealth thanks to Q-learning with a gradient descent.

The challenges lie in finding the right structure for our neural networks and hyper-parameters to automatize this job that is often maths-based. The definition of the observation or time-range of the study, reward and state as well as conventional trading strategies such as the risk management are also important stakes. Finally, a crucial challenge is to reach a good trade-off between computational cost and model complexity.

Although certain myths build an idea of instinct around this job, mathematical models and algorithms are hidden behind. Traditional trading algorithms often rely on fixed rules and historical data but RL has gained prominence for its ability to optimize trading strategies through continual learning and adaptation. They mostly use DQN and policy gradient strategies [1].

In this project, we brought an answer to the challenges involved. We firstly implemented the environment from scratch. It required us to define the observation, state and reward. Then, we adapted it to the needs of our plans and our increasing knowledge of this real-world problem. Another important feature of our work lies in the agent implemented. It was based on the idea of DQN, but we added the following options:

- A Double Deep-Q Network with a replay buffer;
- An epsilon-greedy strategy;
- Neural layers adapted to time series such as RNN, LSTM or GRU;
- An attention mechanism to predict the time series instead of the preceding mentioned layers;
- The progressively additions of new stocks;
- The consideration of daily press reviews.

Our interest focused on the final wealth of the agent. We trained the bot with these different options and the results were not always pretty convincing. There were often no visible increase through the training with simulated episodes of market prices. We tried to vary the stocks, the number of stocks considered at the same time and the time-range considered for the episodes. This implies that this environment is indeed very competitive and intricate. There are many parameters that can influence the strategy to take into account and finding the best trade-off between computational cost and model complexity is quite hard. Finally, even if mathematical and statistical model can help to approximate the market stock exchange, this environment possesses a human bias which is almost impossible to correctly predict.

If we had more time, we would have liked to explore other RL methods. One of them, which is almost the state-of-the-art in its category is the Proximal Policy Optimization. We do not believe that this would give groundbreaking results but it would be interesting from an academic point of view. Evolutionary strategies could also be interesting RL methods for our environment. Perhaps we were stuck in local optima, which evolutionary algorithms would have helped us get out of. A more interesting approach would be the one of imitation learning which could help us to directly learn from experts with imitation reinforcement learning for example.

Here is the link to our GitHub repository: https://github.com/julien-gadonneix/RL4Trading.

II. Background

A. Recurrent layers

In this project we studied three variants of the recurrent neural network [4] that are illustrated in Figure 1.

1) Recurrent Neural Network (RNN): They process sequential data by maintaining a hidden state that catches information from previous time steps. They suffer from the vanishing gradient problem in long sequences where gradients diminish exponentially as they are backpropagated through time. Thus, they struggle to capture long-range dependencies in the data. In our case, the time range was about 20 days which is not too big.

2) Long Short-Term Memory (LSTM): They introduce a memory cell that allows the network to retain information over long periods of time. Thanks to three ”gates” (input, forget and output) to control the flow of information into and out of the memory cell, it enables the network to selectively update and forget information as needed. Thus, LSTMs can catch long-range dependencies in the data by addressing the vanishing gradients problem.

3) Gated Recurrent Unit (GRU): They combine the forget and input gates into a single update gate and merge the memory cell state and hidden state into a single state vector. GRUs have thus fewer parameters compared to LSTMs, which makes them computationally more efficient.

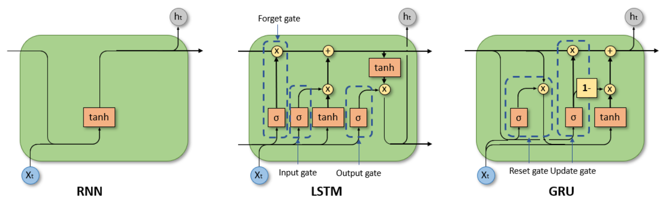

Fig. 1. Schemes of RNN, LSTM and GRU.

Despite their architectural differences, all three architectures are designed to handle sequential data. They have been applied successfully with approximately the same results. In the following, we will thus only consider LSTM.

B. The attention mechanism

In our latest version of the DDQN [2], we wanted to use the state-of-the-art deep neural model to catch interactions and dependencies within a sequence of data. For this purpose, we used the self-attention mechanism of the transformer model [5]. It is based on an encoder-decoder model containing the forward-looking attention mechanism:

$$softmax(\frac{Q \times K^T}{\sqrt{d_K}}) \times V \tag{1}$$

The matrices Q, K and V are obtained from the input via linear layer and this calculation enables to catch the interactions within the sequence. and to transmit information across data.

The encoder-decoder model is pretrained to predict the following element of a sequence. In our model, we will use it to try to predict the price of the following day conditioned by the knowledge of the price of the previous days. Therefore, we will include these results in our DDQN model.

C. Dueling Double Deep Q-Networks

In DDDQN [6], the agent's observation yields two different estimates. A value estimator (V) for the current state which is the value of the agent being in a state and an advantage estimator (A) that decides how much a specific action in a state will benefit the agent. The goal is still to estimate the Q-value and for a state s and an action a, the formula is:

$$\mathcal{Q}(s,a) = V(s) + A(s,a) - \frac{1}{\mathcal{A}} \sum_{a' \in \mathcal{A}} A(s,a')$$

D. News and sentiment analysis

We wanted to slightly complicate our environment to make it closer to the real-world problem. Sentiment analysis of the press is a useful tool for traders. We used a package [7] from GitHub repository to parse articles from the press available on the Internet. The parsing API uses GDELT Project databases which also contains sentiment analysis of the articles and their magnitude which are famous features in natural language processing (NLP). It can thus easily be added as an input to our model.

However, the database was not large enough and the NLP model was not efficient enough to be useful in our project. Nearly all articles were not linked to stock exchange and as the NLP model had to parse the articles, it was slowing down too much the training of our model. Finally, the very few results that we got with it were not convincing. Therefore, we decided to give up temporarily this part of the project and to leave it for future plans.

III. Methodology/Approach

A. The environment

The environment simulates a simplified stock exchange market for trading. We can consider one or more stock at a time. In this environment, the agent can buy new shares with its own money, hold its shares for the future, or sell its shares. The action space is thus:

$$\mathcal{A} = \{buy, hold, sell\} := \{0, 1, 2\}$$

The amount of shares exchanged is also chosen by the agent which will be detailed later (see section III-B). All the choices among the possible actions are based on the observed state. This observed state includes the closing prices of the considered stocks for a predefined number of previous days (`T`), the account balance (available money) (`balance`), and the floating number of shares held (`shares_held`). We chose to normalize the closing prices which might not be realistic as we don't know in advance the future prices. However, this step avoids divergences and an upper bound of the maximum price over a finite number of days can almost always be predicted. In Figure 2, there is an example of the prices of SPY over a finite time range and here is the state space:

$$\begin{aligned}\mathcal{S} &= norm\_close\_price \times account\_balance \times shares\_held \\ &:= \mathbb{R}^T \times \mathbb{R} \times \mathbb{R}\end{aligned}$$

The environment handles the action taken and updates the state accordingly, providing rewards based on the change in total wealth. A simulated episode ends if the agent goes bankrupt without available cash nor shares or if the agent explored the complete range of data initially provided. As for the reward, it is equal to 0 if the end of the data initially provided is reached. Otherwise, the reward is equal to the account balance plus the value of held shares, which is equal to the floating number of held shares times the current value of one share. Thus, the reward is dense and distributed at each step according to the following formula, which approaches real-world methodologies:

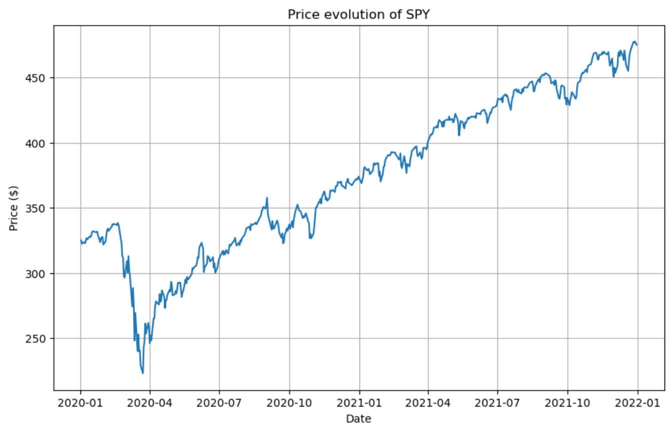

Fig. 2. Prices of SPY from January 2020 to January 2022.

$$R_t = \begin{cases} 0 & \text{if end of data} \\ wealth(t) - wealth(t-1) & \text{otherwise} \end{cases} \tag{2}$$

$$\texttt{wealth} = \texttt{balance} + \texttt{cur\_price} * \texttt{share\_held}$$

Let's delve deeper into the code which draws inspiration from the package gymnasium [3]:

- `_next_observation`: This function returns the observed state as described previously.
- `_take_action`: This function takes as input the action to operate as well as a proportion of shares. If the chosen action is buy or sell, this proportion represents the amount of shares to exchange:

$$shares\_sold = prop * shares\_held$$
$$total\_possible = \frac{balance}{current\_price}$$
$$shares\_bought = prop * total\_possible$$

  The function then updates the account balance and the floating number of shares held by looking at the current price of the stock considered (`current_price`).
- `step`: This function computes the wealth possessed by the agent and calls the function `_take_action` with the parameters of the action and the proportion also received as input. Then, if the episode does not end, it computes the new wealth possessed by the agent and computes the reward.
- `reset`: This function reinitializes all the parameters of the environment.

B. *Our agent*

Our agent interacts with the environment, making decisions based on the current state and learning from the outcomes through the training over simulated episodes of stock prices. We mainly focused on the DDQN [2] method which employs a model for decision-making, a target model for stable Q-value estimation and a replay buffer to store and sample experiences. An ϵ-greedy strategy for action selection was implemented to reach a good trade-off between exploration and exploitation. Epsilon decays over time to shift from exploration to exploitation. Moreover, in order to force or speed up the convergence of the model, there is a learning rate scheduler that will reduce the learning rate over time.

In the ϵ-greedy strategy, we decided to implement a specificity. Firstly, we had a hyper-parameter determining the quantity of share to buy (resp. sell) when the action picked was buy (resp. sell). However, the agent is supposed to improve over time and this hyper-parameter quickly happened to be a limitation. This ϵ-greedy strategy thus returns two numbers: an integer representing the action and a float between 0 and 1 representing the proportion of the maximum amount to possibly buy (resp. sell). Here is the definition of this float:

$$p = \begin{cases} 0.05 & \text{if random (coin} \leq \epsilon) \\ \frac{biggest\_Q - second\_biggest\_Q}{biggest\_Q} & \text{otherwise} \end{cases} \tag{3}$$

The value 0.05 in the random case is a hyper-parameter that can be optimized. With this value of the proportion, if the risk is high, for example if the expected action-value of buying is not much greater than the one of selling, the proportion to buy will be close to 0. On the contrary, if the risk is low, the proportion will be close to 1.

The agent trains over a specified number of episodes, updating its models to maximize rewards, which correspond to effective trading strategies.

C. *The network architectures*

1) DDQN with recurrent layers: This neural network model used for approximating the action-value (Q) function consists of a recurrent layer to process sequences of stock prices, followed by fully connected layers to integrate additional information such as account balance and shares held. The network outputs a value for each possible action, guiding the agent's choices. Recurrent layers seemed well suited at first for this model environment because it possesses an inherent temporal feature. In Figure 3, we can observe the architecture with the LSTM layer.

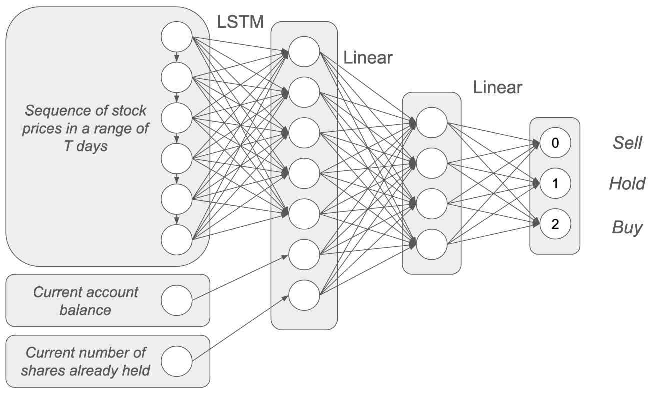
LSTM
Linear
Linear
Sequence of stock prices in a range of T days
Current account balance
Current number of shares already held
0
1
2
Sell
Hold
Buy

Fig. 3. Structure of the DDQN with an LSTM layer.

2) DDQN with the attention mechanism: Since the outcomes of the previous strategy were not particularly persuasive, we decided to explore a closely related approach involving a transformer model. This model consists of an initial transformer applied to the sequence of stock prices. The transformer's objective is to predict future stock values based on a sequence of past prices. It comprises both source and target components and leverages attention mechanisms to understand and forecast the sequence effectively. Additionally, we introduced a masking technique to facilitate progressive learning in the transformer model. Furthermore, we incorporated positional encoding to retain the temporal order of the stock sequence, given the critical importance of time in our context. The architectural depiction of this model can be found in Figure 7.

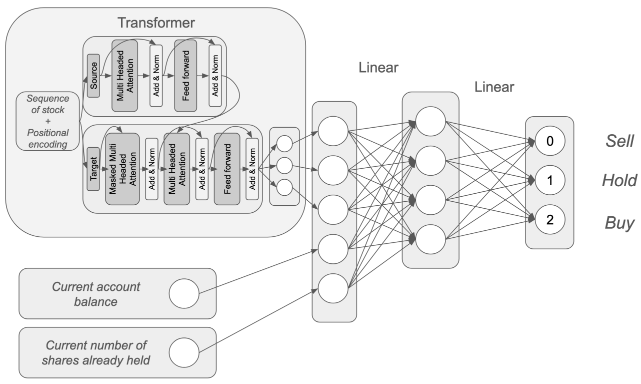

Fig. 4. Structure of the DDQN with an attention layer thanks to the transformer.

3) Our approach of the DDDQN: Ultimately, despite marginal improvements from previous strategies, we opted to test one last method suggested during the mid-term reviews by a specific group. This method involves employing a Dueling Double Deep Q-Network, specifically designed to mitigate issues of overestimation. The problem of overestimation arises from the utilization of the maximum Q-value for the subsequent state in the Q-learning update equation. This reliance on the max operator in the Q-value introduces a bias towards maximization, which can lead to suboptimal performance in environments where the true maximum value is zero, but the agent's estimated maximum is positive. The architecture of this approach is illustrated in Figure 5.

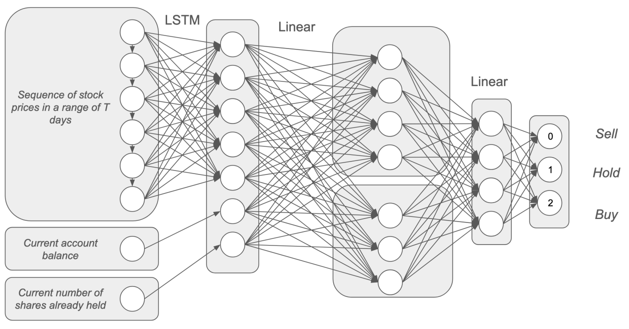
LSTM
Linear
Linear
Sequence of stock prices in a range of T days
Current account balance
Current number of shares already held
0
1
2
Sell
Hold
Buy

Fig. 5. Structure of the Dueling Double DQN.

IV. Results and Discussion

A. Fine tuning and hyperparameters

A significant aspect of the project involved optimizing the models we had developed, a task that proved challenging. Initially, we quickly recognized the potential benefit of adjusting the gamma (discount factor), as the model frequently exhibited a tendency to exclusively pursue either buying or selling, indicating a correlation with a low gamma value.

Subsequently, we realized the necessity of fine-tuning the epsilon decay parameter. This adjustment was crucial in balancing exploration and exploitation, preventing the model from fixating solely on exploitation and neglecting exploration, which could lead to a narrow focus on a single action.

Additionally, we encountered challenges related to T, which represents the size of the stock price list considered. A larger T necessitated a significant memory capacity and extended training time, which was deemed unacceptable.

Finally, we encountered challenges in adjusting the learning rate, a process complicated by its dependence on the specific characteristics of the models. Tuning the learning rate often proved difficult, resulting in significant disparities in rewards even after extensive training and notable peaks in reward outcomes.

B. DDQN with LSTM

With this model, the results were very unstalble from one training to another. Sometimes, it was satisfying like in Figure 6.

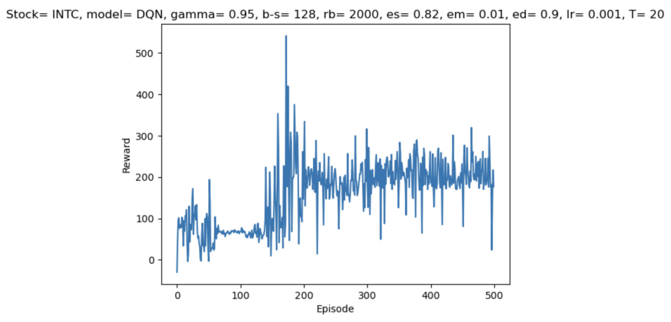

Fig. 6. Application to Intel stock prices with the DDQN model using an LSTM layer.

In this situation, the final reward, which is also the overall reward because of the telescopic sum, seem to have increased throughout training. Here we considered only one stock over two years and the reward reaches a convenient plateau. However, the results were very unstable from one training to another and we can not generalize from this training. As explained before, we tried with various hyerparameters but it was really hard to stabilize the training.

C. DDQN with the attention mechanism

The outcome is quite disappointing as the model displays a tendency to continue purchasing assets incessantly, leading to a continuous decrease in rewards. Even after training for 150,000 epochs (equivalent to 100 training iterations per

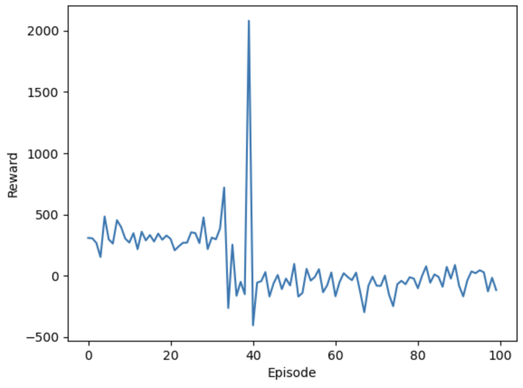

Fig. 7. Application to Intel stock prices with the DDQN model using a transformer.

episode over 1,500 episodes) and conducting 100 episodes, the results remained unsatisfactory.

D. DDDQN

For this model, the outcomes from the neural network were underwhelming. The model consistently favored a passive strategy of holding actions without taking further action, and despite efforts, we were unable to alter this behavior.

V. Conclusions

To address this real-world challenge, we continually enhanced the environment with new features to boost its realism. Within the realm of reinforcement learning (RL), we deployed advanced strategies such as Dueling Double Deep Q-Networks (DDDDQN), incorporating structures like Long Short-Term Memory (LSTM) and transformers.

Our key finding underscored the complexity of this environment, which couldn't be adequately captured using basic methodologies. Nonetheless, the project proved to be an invaluable learning experience, providing insights into real-world problem-solving, teamwork management, and the practical applications of RL. Moreover, it motivated us to explore and devise novel RL-based methodologies independently.

Looking ahead, we discussed several intriguing avenues for future exploration. These include delving into alternative RL strategies like evolutionary strategies and policy optimizations. Additionally, we considered the potential of integrating news parsing and sentiment analysis, despite being beyond the scope of the current coursework. Furthermore, optimizing the learning rate and experimenting with different values for T (the size of the stock list considered at each step) emerged as promising directions. However, achieving a balance with T is crucial, as excessively large values may not necessarily yield favorable outcomes. Therefore, precise extraction and utilization of T remain essential for future endeavors.

Exploring different scenarios for stocks and dates is also worth considering. Training on correlated stocks, such as Apple and Samsung, could reveal interesting insights, as actions in one may impact the other. This aspect of correlated training could provide a deeper understanding of how interconnected stocks behave within the market.

References

[1] Wu, Xing and Chen, Haolei and Wang, Jianjia and Troiano, Luigi and Loia, Vincenzo and Fujita, Hamido. *Information Sciences*, 2020.

[2] Mnih, Volodymyr and Kavukcuoglu, Koray and Silver, David and Rusu, Andrei A and Veness, Joel and Bellemare, Marc G and Graves, Alex and Riedmiller, Martin and Fidjeland, Andreas K and Ostrovski, Georg and others. *nature*, 2015.

[3] Brockman, Greg and Cheung, Vicki and Pettersson, Ludwig and Schneider, Jonas and Schulman, John and Tang, Jie and Zaremba, Wojciech. *arXiv*, 2016.

[4] Jonte Dancker. https://towardsdatascience.com/a-brief-introduction-to-recurrent-neural-networks-638f64a61ff4, 2022.

[5] Vaswani, Ashish and Shazeer, Noam and Parmar, Niki and Uszkoreit, Jakob and Jones, Llion and Gomez, Aidan N and Kaiser, Łukasz and Polosukhin, Illia. *Advances in neural information processing systems*, 2017.

[6] Abhishek Suran. https://towardsdatascience.com/dueling-double-deep-q-learning-using-tensorflow-2-x-7bbbcec06a2a, 2020.

[7] zrxbeijing. https://github.com/zrxbeijing/NewsTrader, 2021.

APPENDIX

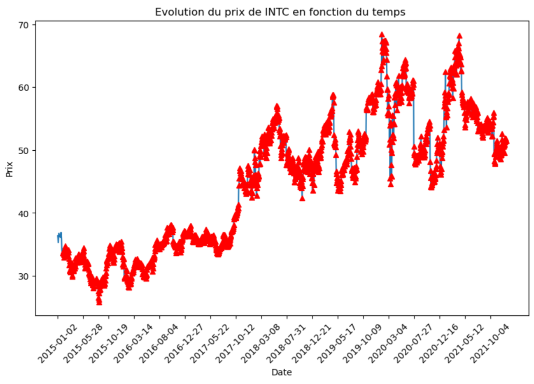

Fig. 8. Action summary for DDQN with transformer.

The occurrences of red triangles represent instances when the agent consistently opts to sell, a strategy that evidently proves ineffective. Despite attempts to adjust parameters such as gamma to extend the reward consideration to more steps, or increasing epsilon decay to encourage greater exploration through random choices for an extended period initially, we were unable to compel the agent to deviate from this suboptimal behavior.

Then, we have several plots explaining the distreibution of actions over a test simulated episode at the end of a training for the DDQN with LSTM.

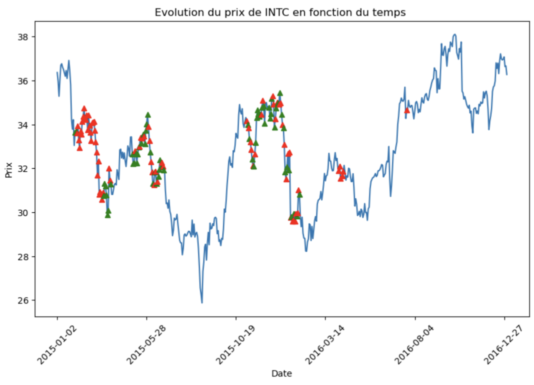

Fig. 9. Action summary for DDQN with LSTM.

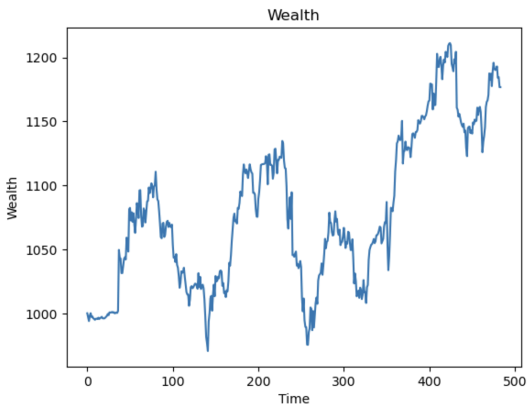

Fig. 10. Action summary for DDQN with LSTM.

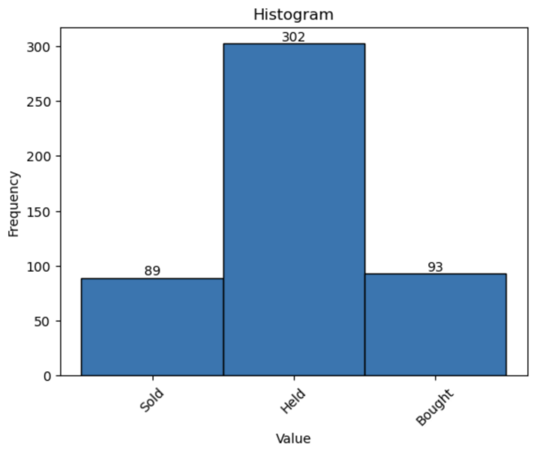

Fig. 11. Action summary for DDQN with LSTM.

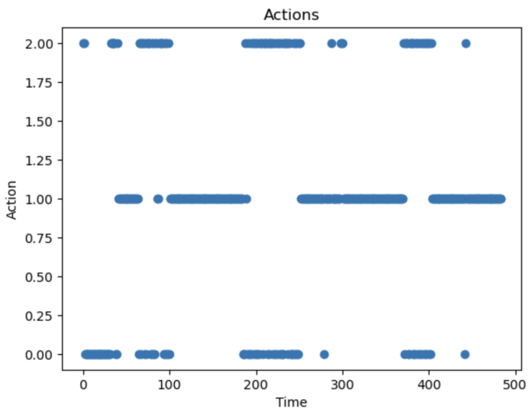

Fig. 12. Action summary for DDQN with LSTM.