Stock Price Prediction Using Q-learning

Project Report

PADAMATA KANISHKA SAI
200001058

CS432
Reinforcement Learning Course Project
IIT INDORE

Submitted To: Dr. Surya Prakash

EMAIL : cse200001058@iiti.ac.in

PROJECT REPORT

1. Introduction:

Predicting stock prices involves forecasting continuous values, making regression techniques suitable for the task. Linear regression, a commonly used method, is effective for predicting continuous values based on historical data. Another approach is the use of time series models like LSTM, which leverage neural networks for stock price predictions. However, reinforcement learning presents an alternative method for stock price prediction.

Reinforcement learning differs from traditional supervised and unsupervised learning methods. It operates through an agent interacting with an environment, aiming to maximize cumulative rewards. Unlike supervised learning, reinforcement learning does not rely on labeled data and can perform well with limited historical data. By employing value functions and policies to guide decision-making, reinforcement learning can effectively predict stock price movements based on the current market environment.

In this project, we explore the application of Q-learning, a reinforcement learning algorithm, for predicting stock price movements. Using historical closing prices, we train a Q-learning agent to make buy, sell, or hold decisions, with the objective of maximizing profits within a simulated trading environment.

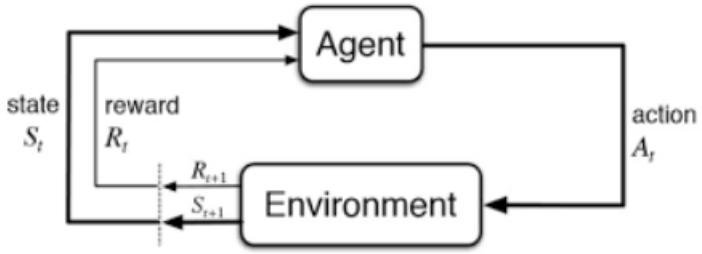
2. Objectives:

- Train a Q-learning agent on historical closing stock prices obtained from Yahoo! Finance.
- Evaluate the agent's performance on unseen test data, comparing its profitability with a buy-and-hold baseline strategy.
- Analyze the strengths and weaknesses of Q-learning for stock prediction.

3. Environment for Stock Prediction

MDP (Markov Decision Process) for Stock Price Prediction:

- Agent – An Agent A that works in Environment E
- Action – Buy/Sell/Hold
- States – Data values
- Rewards – Profit / Loss



4. The Role of Q – Learning:

Q-learning is a model-free reinforcement learning algorithm to learn the quality of actions telling an agent what action to take under what circumstances. Q-learning finds an optimal policy in the sense of maximizing the expected value of the total reward over any successive steps, starting from the current state.

$$NewQ(s,a) = Q(s,a) + \alpha[R(s,a) + \gamma max\, Q'(s',a') - Q(s,a)]$$

New Q value for that state and that action

Current Q value

Learning Rate

Reward for taking that action at that state

Discount rate

Maximum expected future reward **given the new s' and all possible actions at that new state**

5. Reinforcement learning model:

To design and implement the model using python we follow these below steps

- Import the libraries
- Create the agent who will make all decisions
- Define basic functions for formatting the values, sigmoid function, reading the data file etc.
- Train the agent
- Evaluate the agent performance

5.1 Data Preprocessing

The data used for this case study will be the standard and poor's 500. We start by loading historical stock price data, which includes various features such as open, high, low, and close prices, as well as volume. For simplicity, we focus on the closing prices. Any missing values in the dataset are filled using the last available value to ensure continuity in the data.

5.2 Agent Definition

The trading agent is defined using a neural network model. The agent's neural network consists of several fully connected layers, with the number of nodes and layers determined through experimentation and optimization.

The neural network within the agent serves as the Q-function approximator. It takes the state of the environment (e.g., historical stock price data) as input and outputs Q-values for each possible action. These Q-values represent the expected future rewards for taking each action in the given state. The agent then selects the action with the highest Q-value according to its policy.

Additionally, the agent includes other functions essential for its operation

i. **Act**: This function selects an action based on the current state and the agent's policy. It can either choose the action with the highest Q-value (exploitation) or explore a new action with some probability (exploration).
ii. **Memory**: The agent maintains a memory buffer to store experiences, typically in the form of tuples containing (state, action, reward, next_state, done). This memory is used for experience replay, where past experiences are randomly sampled during training to improve learning efficiency and stability.
iii. **Reward Calculation**: After taking an action, the agent receives a reward from the environment. This reward is used to update the Q-values and train the neural network.
iv. **Policy Update**: The agent's policy may evolve over time, such as through the use of an epsilon-greedy strategy for balancing exploration and exploitation. This policy update mechanism guides the agent's decision-making process.

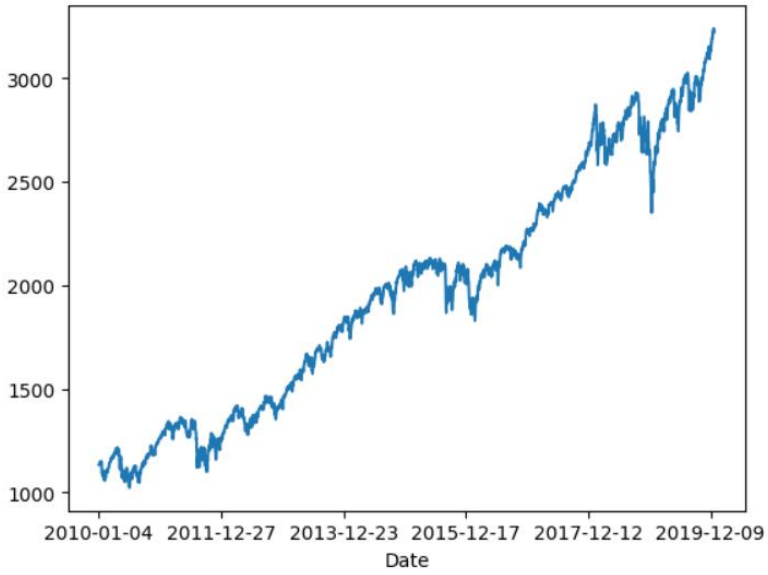
By integrating these components, the agent learns to make optimal decisions in the trading environment, aiming to maximize cumulative rewards over time.

5.3 Model Training

The model is trained using a Q-learning algorithm, which involves iteratively updating the Q-values of state-action pairs based on the rewards received. During training, the agent interacts with the environment (historical stock price data) by observing the current state, taking actions, receiving rewards, and updating its Q-values.

Along with these a hyperparameter called window size is included, which represents the number of days of closing prices (previous n-days) that is used as input for the current state to decide on which actions should to maximize the reward. This parameter is crucial for capturing temporal patterns in the data and guiding the agent's decision-making process.

The below picture represents the actual closing prices before training of the model



Where the y-axis represents the closing prices and x-axis refers to the time (dates)

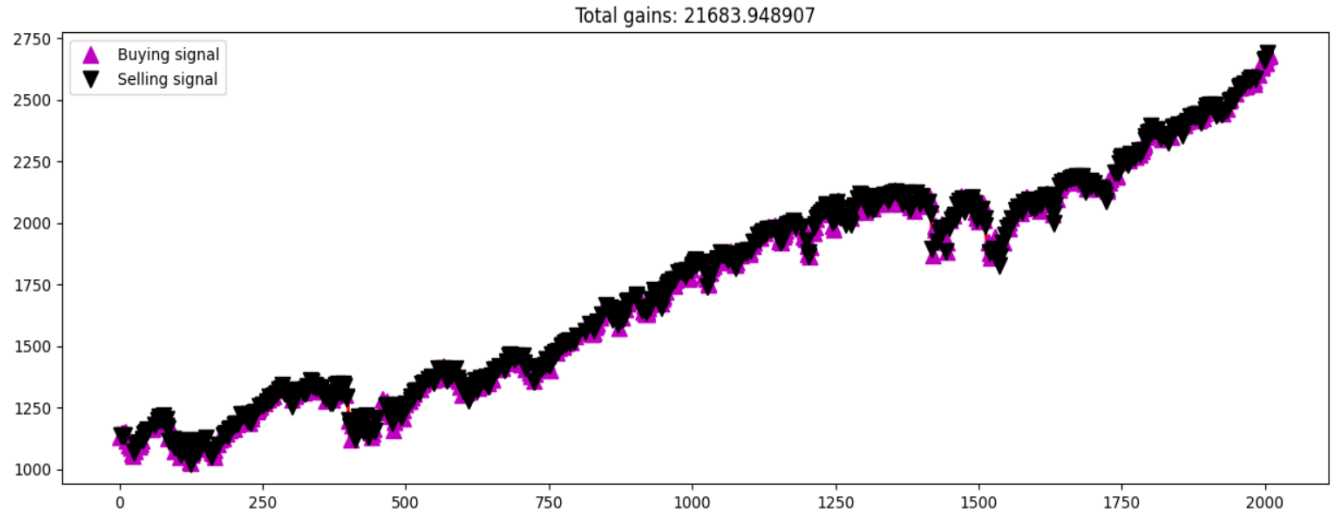
5.3.1 Training Process

1. **Initialization**: The agent initializes its Q-table (in our case, the neural network weights) and other parameters such as the learning rate, discount factor (gamma), exploration rate (epsilon), and batch size.
2. **Episode Execution**: Each episode represents a complete pass over the training data. For each time step in an episode, the agent observes the current state, selects an action using an epsilon-greedy policy, and receives a reward based on its action.

3. **Q-Value Update**: The Q-value of the current state-action pair is updated using the Bellman equation, which incorporates the reward received and the estimated future rewards. In the case of a neural network, this involves updating the weights of the network through backpropagation.
4. **Memory Replay**: In some Q-learning implementations, a memory replay mechanism is used to store and replay past experiences. This helps stabilize and improve the training process by reducing correlations between consecutive experiences.
5. **Model Update**: The neural network weights are updated using backpropagation to minimize the difference between the predicted Q-values and the target Q-values.
6. **Exploration vs. Exploitation**: The epsilon value is gradually decayed over time to balance exploration (trying new actions) and exploitation (using learned actions).

In our case we implemented for 100 episodes with window size as 10, learning rate at 0.01, discount factor at 0.95 and epsilon at 0.1 and batch size of 32 for better performance of the model.
Below picture depicts the model that is the state -action pair graph after 100 episodes of training and the total profit is also calculated after maximizing the rewards through the learning process.

```
Running episode 100/100
--------------------------------
Total Profit: $21683.95
--------------------------------
```



Where the y-axis represents the closing prices (states) and x-axis refer to the actions that is buy, sell or hold.

5.4 Model Testing and Evaluation

Once the model is trained, it is tested on unseen data to evaluate its performance. The test data is similar to the training data but does not overlap with it. In our case we use a validation split of 0.2 representing 20 percent of the given dataset as testing dataset and 80 percent as training.

Performance metrics used for the evaluation:

1. **Total Profit**: Total profit is the cumulative financial gain or loss generated by a trading strategy over a specified period. It is calculated by summing the profits from successful trades (selling at a higher price than buying) and losses from unsuccessful trades (selling at a lower price than buying).
2. **Sharpe Ratio**: The Sharpe ratio quantifies the risk-adjusted return of an investment or trading strategy. It is calculated by dividing the average return of the strategy by its standard deviation. The formula for Sharpe ratio is:

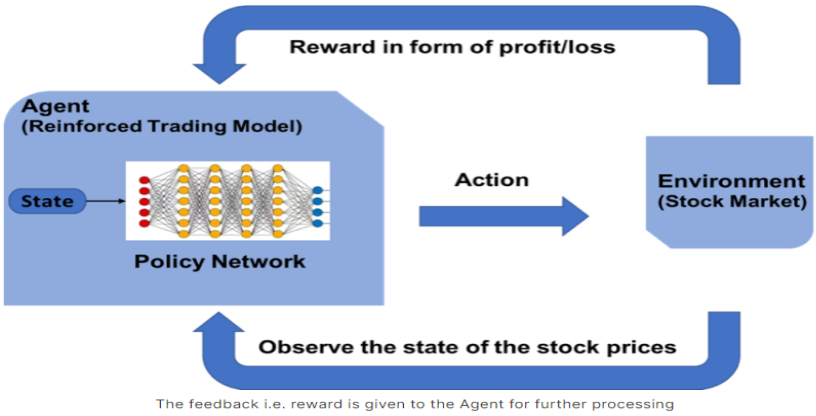
$$Sharpe\ Ratio = \frac{R_p - R_f}{\sigma_p}$$

   where R_p is the average return of the strategy, $RfRf$ is the risk-free rate of return, and $\sigma_p$is the standard deviation of the strategy's returns.
3. **Win Rate**: The win rate is the percentage of profitable trades out of the total number of trades executed by the model. It measures the model's accuracy in making successful predictions and generating profits consistently. The win rate is calculated as the number of profitable trades divided by the total number of trades, multiplied by 100 to express it as a percentage.
4. **Baseline metrics:** The baseline metrics offer a comparison to a passive investment approach, like buy and hold. ***Baseline Profit*** calculates the profit from buying and selling without active trading. ***Baseline Sharpe Ratio*** assesses risk-adjusted return, providing a benchmark for the trading strategy's performance. ***Baseline Win Rate*** indicates the percentage of profitable trades, serving as a consistency reference relative to passive investment.

Performance metrics such as total profit, Sharpe ratio, win rate, and baseline metrics are calculated to assess the model's effectiveness in making profitable trades and managing risk.

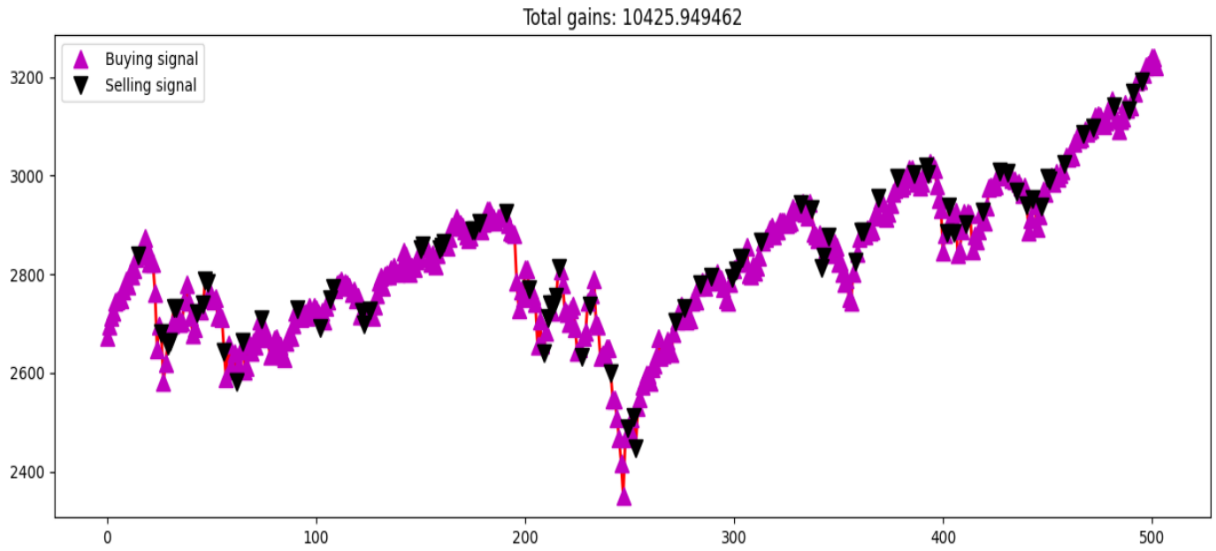
5.5 Results and Analysis

The results of the model testing are analyzed to determine its overall performance. The model's ability to generate profits and manage risk is evaluated based on the performance metrics. The results are compared against a baseline strategy (e.g., buy and hold) to assess the model's effectiveness.
So finally, this is what our model's environment looks like



After training, the model is represented as picture mentioned in section 5.3. Now after testing the results are depicted as below

```
Total Profit: $10425.95
Sharpe Ratio: 0.044723865976940634
Win Rate: 0.6705882352941176
Baseline Profit: $557.17
Baseline Sharpe Ratio: 0
Baseline Win Rate: 0
------------------------------------------
```



This graph represents state-action pairs on the testing data, where y- axis represents the states (closing prices) and the x-axis actions.

Performance metrics evaluation

Metric	Value
Total Profit	$10425.95
Sharpe Ratio	0.0447239
Win Rate	0.6705882
Baseline Profit	$557.17
Baseline Sharpe Ratio	0
Baseline Win Rate	0

The results indicate that the model achieved a total profit of $10,425.95, suggesting that it was able to generate significant returns during the trading period. However, the Sharpe ratio of 0.0447239 indicates that the risk-adjusted return may be relatively low compared to the overall volatility of the market. Additionally, the win rate of 0.6705882 suggests that the model made correct trading decisions approximately 67.06% of the time.

Comparing these results to a baseline strategy, which resulted in a profit of $557.17, the model outperformed the baseline significantly in terms of total profit. However, further analysis is needed to assess whether the model's performance is satisfactory considering the associated risks and market conditions.

6. Conclusion

In conclusion, this report provides an overview of the model's performance in predicting stock price movements using a Q-learning algorithm with a neural network. While the model demonstrates promising results in terms of profitability and risk management, it is not without its limitations. One notable limitation is the absence of experience replay, a technique that can help stabilize and improve the training process by reducing correlations between consecutive experiences. Incorporating experience replay into future iterations of the model could enhance its robustness and performance.

Looking ahead, there are several avenues for future research and improvement. One direction is the incorporation of additional features beyond just closing prices, such as volume and technical indicators, to provide the model with more information for decision-making. Furthermore, exploring more advanced deep reinforcement learning techniques could lead to even more accurate predictions and better risk management strategies. Overall, while the current model shows promise, there is still room for refinement and enhancement to further optimize its performance in real-world trading scenarios.

References

1. The data used for this case study will be the standard and poor's 500. The link to the data is : https://ca.finance.yahoo.com/quote/%255EGSPC/history?p=%255EGSPC.

2. Rani, Poonam, et al. "Stock price prediction using reinforcement learning." *International Conference on Innovative Computing and Communications: Proceedings of ICICC 2021, Volume 2*. Springer Singapore, 2022.

3. Tan, Fuxiao, Pengfei Yan, and Xinping Guan. "Deep reinforcement learning: from Q-learning to deep Q-learning." *Neural Information Processing: 24th International Conference, ICONIP 2017, Guangzhou, China, November 14–18, 2017, Proceedings, Part IV 24*. Springer International Publishing, 2017.