FinSearch EndTerm Report

Using Deep Reinforcement Learning (RL) to Optimise Stock Trading Strategy and thus Maximise Investment Return

Name: Aadit Sule
Roll Number: 23B4204
Team Number : D3

1. Reinforcement Learning

1.1. What is Reinforcement Learning?

Reinforcement learning is a part of machine learning where agents are self-taught in different mechanisms of rewards and punishment. It's about taking the best possible action or path to gain maximum rewards and minimum punishment through observations in a specific situation. It acts as a signal toward positive and negative behaviors. What is developed, in concrete terms, is an agent or a plurality of agents.

Some formal definitions of RL are:

Reinforcement learning, a type of machine learning, in which agents take actions in an environment aimed at maximizing their cumulative rewards – NVIDIA

It is a type of machine learning technique where a computer agent learns to perform a task through repeated trial and error interactions with a dynamic environment. This learning approach enables the agent to make a series of decisions that maximize a reward metric for the task without human intervention and without being explicitly programmed to achieve the task – Mathworks

In simple terms, over a series of Trial and Error paths, the agent continues to learn continually within an interactive environment through its own actions and experiences. In simple words, it has only one target: to find the right model of actions that will raise the general cumulative reward of an agent. It learns by interaction and feedback.

1.2. How is Reinforcement Learning different from Supervised Learning?

There are three major types of Learning Processes:

- **Supervised Learning:** Relies on labeled data. Each data point has a pre-defined output or label (e.g., classifying emails as spam or not spam). The model learns the mapping between the input data and the desired output.
- **Unsupervised Learning:** Deals with unlabeled data. The goal is to identify patterns or structures within the data itself (e.g., grouping customers with similar purchase history). No pre-defined output is provided.
- **Reinforcement Learning:** Doesn't use labeled data. The agent interacts with the environment and receives feedback in the form of rewards (positive, negative, or neutral). The agent learns through trial and error to maximize future rewards.

In supervised learning, the model gets trained with a dataset that has a correct answer key. The decision remains independent of each other and is done on the initial input given as it has all the data that's required to train the machine. Each decision is represented through a label. Whereas Reinforcement learning (RL) offers a diverse toolbox of algorithms. Some popular examples include Q-learning, policy gradient methods, and Monte Carlo methods, along with temporal difference learning. Deep RL takes things a step further by incorporating powerful deep neural networks into the RL framework. One such deep RL algorithm is Trust Region Policy Optimization (TRPO).

1.3. Deep Learning

Deep learning is a subdomain of machine learning, which is constructed based on an artificial neural network architecture. Artificial neural network or ANN uses layers of interconnected nodes called neurons, where every neuron works together to process the input data and learn from it.

In a fully connected Deep neural network, there is an input layer and one or more hidden layers connected one after another. Every neuron takes an input from the previous layer neurons or directly from the input layer. The output obtained from one neuron will become the input to other neurons in the next layer of the network, and so the process continues until the final layer produces the output of the network. The layers of a neural network perform a series of nonlinear transformations on the input data, which helps the network in learning complex representations of the input data.

Artificial neural networks are essentially built using principles on the structure and operation of the human neuron. The first artificial neural network layer is the input layer, which receives the input from external sources to the hidden layer, the second layer. In the hidden layer, each neuron receives information from the previous layer neurons, computes the weighted total, and then transfers it to the neurons in the next layer. These are weighted links; thus, the influences of the inputs from the previous layer are more or less optimized by giving each input a different weight. Such weights are adjusted in the training phase to maximize the model's performance.

In the case of a fully connected artificial neural network, there would be an input layer and one or more hidden layers connected to each other one after the other. Every neuron gets an input from the previous layer neurons or the input layer. Then, the output from one neuron serves as an input to other neurons in the next layer of a network, and so the process goes on until the last layer gives an output for a network. This data passes through one or more hidden layers, which transforms it into valuable data for the output layer. Finally, the output layer produces the result in the form of the response of the artificial neural network to the data passed in. Most neural networks have the units fully connected to each other from one layer to another. And all these links have weights which determine how much one unit affects another. The neural network gets to learn more and more about the data as it moves from one unit to another, till the output is produced.

2. Deep Q-Networks (DQN)

2.1. What is DQN?

Deep Q-Network (DQN) is an advanced reinforcement learning (RL) algorithm that combines Q-Learning with deep neural networks to handle environments with high-dimensional state spaces. Developed by researchers at DeepMind, DQN was a breakthrough in the field as it demonstrated the ability to learn complex policies directly from raw sensory input (e.g., pixels from video games).

2.2. Theory Behind DQN

Q-Learning is a value-based method of RL that aims to learn the optimal action-selection policy by estimating the action-value function, $Q(s, a)$. This function represents the expected return (future rewards) of taking action a in state s and following the optimal policy thereafter.

The Q-Learning update rule is:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left(r + \gamma \max_{a'} Q(s', a') - Q(s,a) \right),$$

where:

- s and s' are the current and next states,
- a and a' are the current and next actions,
- r is the reward received after taking action a,
- α is the learning rate,
- γ is the discount factor.

In DQN, a neural network (Q-network) approximates the Q-values, and the network weights are updated based on the Temporal Difference (TD) error:

$$\delta = r + \gamma \max_{a'} Q(s', a'; \theta^-) - Q(s, a; \theta),$$

where:

- θ are the weights of the Q-network,
- θ^- are the weights of the target network, which are periodically copied from θ to stabilize training.

2.3. Core Concepts of DQN

- **Experience Replay:** The agent stores its experiences, (s, a, r, s'), in a replay buffer. During training, random samples from this buffer are used to break the correlation between consecutive experiences and stabilize the training process.

- **Target Network:** A separate target network is used to generate the target Q-values. This network's weights (θ^-) are updated less frequently to provide a stable target for the Q-learning updates.

- **Q-Network:** A deep neural network that approximates the Q-value function $Q(s, a; \theta)$. The network takes the state as input and outputs Q-values for all possible actions.

2.4. Algorithm

1. Initialize the replay buffer D.
2. Initialize the Q-network with random weights θ.
3. Initialize the target network with weights $\theta^- = \theta$.
4. For each episode:
   - Initialize the state s.
   - For each time step:
     - With probability ϵ, select a random action a; otherwise, select $a = \arg\max_a Q(s, a; \theta)$.
     - Execute action a and observe reward r and next state s'.
     - Store transition (s, a, r, s') in the replay buffer D.
     - Sample a random mini-batch of transitions from D.
     - Compute the target $y = r + \gamma \max_{a'} Q(s', a'; \theta^-)$.
     - Perform a gradient descent step on $(y - Q(s, a; \theta))^2$ with respect to the network parameters θ.
     - Update $s = s'$.
5. Periodically update the target network $\theta^- = \theta$.

2.5. Advantages of DQN

- **High-Dimensional Input Handling:** DQN can learn policies directly from raw sensory inputs like images, which traditional Q-learning cannot handle efficiently.

- **Stabilized Training:** Experience replay and target networks significantly stabilize the training process, allowing DQN to learn in complex environments.

- **Scalability:** The algorithm scales well to large state spaces due to the use of deep neural networks.

2.6. Limitations of DQN

- **Sample Inefficiency:** DQN requires a large number of training samples and interactions with the environment, making it computationally expensive.
- **Overestimation Bias:** The max operator in the Q-learning update can lead to overestimation of Q-values, resulting in suboptimal policies.
- **Limited to Discrete Actions:** DQN is designed for discrete action spaces and requires modifications for continuous action spaces.

2.7. Differences Between DQN and DDPG

Deep Deterministic Policy Gradient (DDPG) is another RL algorithm designed to handle continuous action spaces, unlike DQN. Here are the key differences:

- **Action Space:**
  - DQN: Works with discrete action spaces.
  - DDPG: Handles continuous action spaces using deterministic policy gradients.
- **Algorithm Type:**
  - DQN: Value-based algorithm that estimates the action-value function.
  - DDPG: Actor-Critic algorithm combining value-based and policy-based methods. It uses an actor network to map states to actions and a critic network to evaluate the action-value function.
- **Policy Representation:**
  - DQN: Does not explicitly represent a policy; derives the policy indirectly from the Q-values.
  - DDPG: Uses an actor network to explicitly represent the policy, making it suitable for continuous actions.
- **Exploration Strategy:**
  - DQN: Uses ϵ-greedy exploration.
  - DDPG: Uses noise processes (e.g., Ornstein-Uhlenbeck process) to explore the action space.

3. Building the Model

3.1. Introduction

The goal of this project is to optimize stock trading strategies to maximize investment returns. We experiment with two distinct models: a benchmark Long Short-Term Memory (LSTM) based model and a Deep Reinforcement Learning (DRL) based model. The LSTM model is designed to predict future stock prices using historical data, while the DRL model takes a decision-making approach where the agent buys, sells, or holds stocks to maximize profits. We use the Indian stock market index (NSEI) data for this purpose.

3.2. Data Collection and Preprocessing

We used Yahoo Finance's API to retrieve daily data for the NSEI index from January 1, 2020, to January 1, 2024. The dataset includes features like the adjusted closing price, returns, volatility, moving averages (50-day, 20-day), exponential moving averages (12, 26, 50), and other technical indicators such as RSI and MACD.

After the raw data is collected, feature engineering is performed, where log returns, volatility, and moving averages are calculated. These features help to capture the underlying trends and momentum in stock prices, critical for making informed predictions. Missing values are handled by dropping rows where data is unavailable.

Once the features are created, the data is scaled using MinMaxScaler to bring all features within the range of 0 and 1, making the dataset ready for modeling.

3.3. Training the Model and Testing

The LSTM model is trained using a sliding window approach. The window size is 200 days, and each window predicts the next day's closing price. The model architecture includes two LSTM layers, followed by dropout layers for regularization, and two dense layers for final predictions. Hyperparameter tuning is performed using Keras Tuner to optimize LSTM units, dropout rates, and learning rates.

The model is trained for 25 epochs with early stopping to avoid overfitting. After training, the model is tested on a separate test dataset using the same preprocessing steps.

The Reinforcement Learning model employs Deep Q-Learning (DQN). Here, the agent learns by interacting with the environment (stock market data) and making decisions to buy, sell, or hold stocks. It uses a neural network to approximate Q-values, which represent the expected future rewards for a given state-action pair.

3.4. Benchmark LSTM-based Model

The LSTM model is trained to predict stock prices using technical indicators. After training, the model's performance is evaluated based on its ability to predict stock prices. The mean squared error (MSE) and mean absolute error (MAE) are used as evaluation metrics, which measure the average error between the predicted and actual prices.

Additionally, the model's effectiveness is evaluated by applying a simple trading strategy based on the predicted prices. Returns are calculated using the log of the predicted price changes, and the performance is compared with the actual market returns. The model's predicted returns are cumulatively plotted against the actual market returns to gauge its effectiveness in generating positive returns.

Key performance metrics for the LSTM model:

- MSE: Measures how close the predicted prices are to actual prices.
- MAE: Measures the average magnitude of prediction errors.
- Volatility: Annualized volatility, calculated as the standard deviation of returns.
- Maximum Drawdown: Measures the largest drop from peak to trough in stock price during the trading period.

3.5. Deep Reinforcement Learning-based Model

In the DRL model, a Deep Q-Learning agent is trained to make buy, sell, or hold decisions based on the stock price movements. The agent interacts with the environment, learning through exploration and exploitation of various stock trading actions. The reward is based on the profit or loss from trading decisions. The agent updates its Q-values and progressively becomes more adept at maximizing cumulative rewards (profits).

After training the agent for 10 episodes on the NSEI index data, the model was evaluated on unseen data from January 2024 onwards. The agent's performance was measured by the total profit it could generate over the testing period.

Key concepts in the DRL model:

- Actions: The agent can either buy, sell, or hold stocks.
- Rewards: Profit from selling stocks or holding value.
- Exploration vs. Exploitation: The agent explores different strategies initially but gradually starts exploiting the best-performing strategies as it learns.

3.6. Comparison of the Two Models

The LSTM and DRL models were evaluated based on their ability to generate profits and minimize risk. The key comparison points are as follows:

Prediction Accuracy: The LSTM model focuses on price prediction accuracy, while the DRL model focuses on decision-making for maximizing profits.

- LSTM MSE: [Insert MSE value]
- LSTM MAE: [Insert MAE value]
- Cumulative Returns: The LSTM model uses predicted prices to simulate trading, whereas the DRL agent directly learns to maximize cumulative profits.
- LSTM Cumulative Market Returns: [Insert Value]
- LSTM Cumulative Strategy Returns: [Insert Value]
- DRL Total Profit: [Insert Total Profit Value]
- Risk and Drawdown: The LSTM model's strategy faces risk measured by volatility and maximum drawdown. The DRL agent's profit-generation capability is measured by the total profit from trading actions.
- LSTM Volatility: [Insert Value]
- LSTM Max Drawdown: [Insert Value]

Overall, while the LSTM model provides reliable stock price predictions, the DRL model excels at making effective buy-sell decisions, particularly during high market volatility.