CS138 Project Proposal

Joe Ferraro and Alex Rus

November 2024

1 Problem Statement

Our project aims to solve a financial forecasting problem by utilizing reinforcement learning (RL). Our goal is to train an RL agent to predict how a stock is going to move (i.e., determine the next day's closing price of a certain stock or group of stocks) and make decisions to either buy, hold, or sell a stock based on historical market data. The initial training will take place in a simulated trading environment, using historical closing price data and RL algorithms.

Our findings will allow us to answer the question of whether a reinforcement learning policy trained in a simulated environment with historical data can be effectively generalized to live trading on a typical market day. If the answer to this question is negative, we will consider the enhancements needed in financial modeling to improve stock price prediction accuracy. Conversely, if the answer is affirmative, we will have developed a powerful predictive capability. Consequently, we must then address how this capability can be deployed in real-life trading and determine which safeguards need to be in place to ensure safe usage.

2 Policy Gradient Methods

In our project, the agent's observations will be determined by market data, specifically in the form of historical closing prices and technical indicators. To make predictions, we will implement an approximate solution method, where the agent learns a policy directly via policy gradient methods. The policy will be determined by the following function:

$$\pi(a|s, \theta) = P(A_t = a|S_t = s, \theta_t = \theta)$$

In this policy function, π is the actual policy, modeling the probability distribution over a series of actions and states. Here, a represents the action (i.e., Buy, Hold, or Sell), s represents the state (i.e., the current market conditions based on the observed data), and θ is a parameter vector used to support this model. As we will discuss, θ will represent the confidence value assigned to each action.

To optimize the policy, we will update the parameter vector θ via gradient ascent. The agent's running reward function (i.e., the total adjusted profit and loss of the agent) will be determined by the function $J(\theta)$ below:

$$\nabla J(\theta) \propto E_\pi \left[\sum_a q^\pi(S_t, a) \nabla \pi(a|S_t, \theta) \right]$$

The above equation represents the gradient of the function $J(\theta)$. Here, $q^\pi(S_t, a)$ is the action-value function the agent will use, which represents the expected return when action a is taken in state S_t under policy π. The gradient points in the direction that will improve the agent's profit or reward, allowing it to make continually improving predictions.

As prefaced above, θ represents the level of confidence associated with either buying, selling, or holding the stock, based on the observed state s. Under this policy, the agent will select the action with the highest confidence level. Since deciding between exploration and exploitation of the market environment is essential, an epsilon-greedy algorithm will be employed to ensure a balance between exploration and exploitation, optimally preventing the agent from getting stuck in a suboptimal decision-making path.

The reward r_t will be determined at each timestep t to encourage optimal buy/sell/hold decisions. Depending on the success and implementation path of our model, r_t can be simply defined as the change in portfolio value after taking each action. A more refined approach we are considering is using the Sharpe Ratio to evaluate performance. The Sharpe Ratio is an industry-standard performance metric that balances a portfolio's returns with its volatility, creating a smoother learning signal and encouraging more stable actions.

3 Deep Q-Networks (DQN)

Deep Q-Networks (DQNs) are a category of artificial neural networks specifically tailored for reinforcement learning scenarios. DQNs consist of an input layer, one or multiple hidden layers, and an output layer. Each layer computes the weighted sum of inputs from the prior layer, using an activation function to generate an output from the input. In our RL problem, this neural network structure allows the agent to learn the estimated "Q-values" for each action in each state, which enables the agent to maximize cumulative rewards.

The DQN combines Q-learning with deep learning to approximate a function $Q(s, a)$, which represents the optimal action-value function. This function will be updated as follows:

$$Q(s, a) = r + \gamma \max_{a'} Q(s', a')$$

In the above function:

- r is the immediate reward,
- γ is the discount factor,
- s' is the next state, and
- $\max_{a'} Q(s', a')$ is the highest possible Q-value for the next state s'.

As described in our experiment and implementation plan, we do not have sufficient computational power to calculate and store the Q-values for each state-action pair in this problem, especially when training and testing on a large set of market data. Therefore, our approach will be to approximate the Q-function, allowing the agent to generalize across similar states.

The DQN has two important aspects useful for our forecasting:

1. **Experience Replay:** Instead of updating the Q-function immediately after each action, the agent samples a subset of its experiences from a "replay buffer" (i.e., a tuple (s, a, r, s')) and updates the neural network in mini-batches. This makes the model more stable by reducing correlation between consecutive samples.
2. **Target Network:** The DQN uses a second neural network, known as the target network, which is a copy of the main Q-network and remains frozen for a certain number of iterations. This provides a more stable Q-value reference and prevents unstable updates, better ensuring algorithmic convergence.

DQN typically uses the Stochastic Gradient Descent method to minimize the loss function for the neural network. In an artificial neural network, the loss function determines the error between the predicted Q-value and the target Q-value derived from our Q-function. Specifically, the loss function is defined as follows:

$$L(\theta) = E_{(s,a,r,s') \sim ReplayBuffer} \left[\left(r + \gamma \max_{a'} Q(s', a'; \theta^-) - Q(s, a; \theta) \right)^2 \right]$$

In the above, θ is the parameter of the Q-Network, and θ^- is the parameter of the target network. By minimizing this loss, the DQN will more accurately estimate Q-values, allowing us to choose the best possible actions to maximize future rewards.

In the context of our financial forecasting, the DQN will use market data as inputs—specifically in the form of historical prices (and, depending on the initial performance of our model, technical indicators such as moving averages). As these inputs pass through the hidden layers of the neural network, we will learn to identify patterns needed to predict stock price movements. The output of the neural network will be the Q-value for each possible action (i.e., Buy, Sell, Hold). The agent will select the action for which the Q-value is the highest.

4 Related Work

It is no secret that the stock price prediction problem is a difficult one to solve, even with the power of reinforcement learning. While other common RL applications such as video games, controlled simulations, etc., have a more fixed environment that allows for swift policy-making, the market environment is influenced by factors ranging from market maker quoting to geopolitical events and investor sentiment. Many studies have shown that the dynamic nature of markets makes it challenging to create accurate simulations that mirror real-life trading (see Li, Yang, & Tan, 2020).

Thus, while our goal is to train our model based on historical data, we aim to avoid the dilemma of overfitting—i.e., the classic conundrum where the agent performs well on known training data but becomes ineffective when introduced to new data in the test set. This leads to our need to approximate the market environment in our model, as opposed to attempting to mimic it one-to-one in our implementation.

To bridge the gap between historical simulations and real-world trading, researchers have turned to *transfer learning* techniques. This approach aims to adapt models from simulations to real market activity. In order to effectively transfer learning between simulation and real-life, researchers have focused on risk-reward structures for optimal policy-making. For instance, the FinRL framework was established to integrate principles of financial trading with RL algorithms, including the DQN structure we intend to implement. This framework has proven feasible and adaptable to live data by retraining the model on rolling windows of stock data (Liu et al., 2021). This study especially highlights the risk-reward principles used in the FinRL library, where financial metrics such as the Sharpe Ratio are incorporated into the risk-reward structure in the RL implementation.

An equally important part of the implementation of our solution for financial forecasting is the algorithm itself—the Deep Q-Network (DQN). Research by Huang, Sun, & Cai (2019) has shown that DQN is an ideal choice for an RL algorithm when solving financial problems and has even been proven to outperform other models used in portfolio creation. All in all, our research will leverage the findings of these studies to produce an optimized financial forecasting solution backed by established research.