PORTFOLIO OPTIMIZATION USING RL

Submitted by:
G Naga Jaswanth
B Hari Charan Goud

IIT Bhilai

Course Code: DS251
Course Name: Artificial Intelligence

1 Problem Statement

In the world of stock markets, Optimising portfolios creates a challenge for traders in order to maximize their profits while minimizing risks. Traditional approaches often depend on static models and heuristics, which may not be suitable for dynamic markets. Hence, it is really important to find new innovative solutions that can use advanced technologies to optimize portfolios effectively. One such technology is Reinforcement Learning. The task is to develop a reliable portfolio optimization framework adapted specifically for traders using reinforcement learning techniques like Q-Learning.

2 Methodology

(Including code explanation)

2.1 Data collection and Preprocessing

The data is collected from NASDAQ (National Association of Securities Dealers Automated Quotations), an American stock exchange based in New York City (stocks list), alongside from Yahoo finance (historical data for selected stocks). This is covered in `extract_stocks.ipynb`.

For each selected stock the preprocessing step separates 80% of historical prices up to the for training, and the rest are reserved for testing purposes. The data consists of historical stock prices includes opening, closing, high, low prices and volume of the stock, and also adjusted closing price. This step also includes adding few technical indicators like RSI, EMA, Stochastic Oscillator etc., these features are selected as per the number selected during the training of LSTM network. It also adds target column, the change in close prices of stocks, which are to be used as labels for training purposes and all this data is scaled using MinMaxScaler. Lastly, these data are converted into an array of matrices of shape $(lookback, features)$, where lookback value is the number of days in the past we wish to train the LSTM model on. This step is achieved using the `preprocess` function, in `utilities.py`. The preprocessing step finally returns train & test samples of training and labels of shapes $(len(data), lookback, features)$ and $(len(data), 1)$ respectively.

2.2 Training data using LSTM

LSTM (Long Short-Term Memory) is an improved version of recurrent neural networks (RNNs). RNNs work like how human learn. LSTMs are a kind of RNNs which remembers the information over a longer period of time, and also handles variable size inputs as it unrolls, which makes them better suited for stock price prediction.

The training data, consisting of historical stock prices along with indicators, is used to train the LSTM neural network. The LSTM model learns from the temporal patterns and dependencies in the training data to make accurate predictions for future stock prices. This is achieved using the `train` function of class object `TradeAgent` from `agent.py` module, which uses the data obtained after the preprocessing step, to train upto 30 epochs by default. This object also has functionalities to save the model and evaluate it on the test data accordingly. As expected, this model predicts the scaled difference of closing prices between current day and next day.

2.3 Deep Q-Network

The DQN framework, using experience replay, employs a neural network architecture to approximate the Q-function, enabling it to learn optimal actions for portfolio management. By integrating historical data and LSTM predictions, the DQN dynamically adjusts its policy to maximize portfolio returns while effectively managing risk. Through iterative training, the DQN refines its decision-making capabilities, leveraging insights from both past market behavior and anticipated future trends to inform daily buy, sell, or hold decisions for each stock in the portfolio. This framework is based on `DQNAgent` from deep-q-learning repository by @keon.

This DQN implemented using the `DQNAgent` also found in `agent.py` module, leverages a feed-forward fully-connected tensorflow model for the prediction of Q values.

2.3.1 State space:

State space is structured as an $k \times 1$ vector, where k represents the # of stocks and the value in each row represents the scaled difference between the present day's closing price and the previous day's closing price, which are obtained as the predictions of the LSTM model for the selected stocks. These states are obtained by reshaping the outputs produced using the model, and stacking them against each other as rows, which is achieved by the `allStates` function from the `utilities.py` module, which also takes care of creation and saving of model, in case it does not exist in `models` directory. This function finally returns this stacked prediction data along with true price data

2.3.2 Action:

The model's actions consist of three options: buying, selling, or holding assets within the portfolio for each stock, which makes it a $k \times 3$ array of probabilities of these actions for each stock. Then the actions vector is obtained by taking argmax of each row to obtain the action we wish to perform for each stock, hence this also becomes a $k \times 1$ vector. These actions are determined based on the model's analysis of the data, i.e., the prediction of DQN model via an epsilon greedy approach.

The goal is to optimize the portfolio by making decisions that maximize returns while managing risks. Through continuous learning and adaptation, the model is expected to refine and get reward using its decision-making process, leveraging insights from past market behavior and anticipated future trends to guide its actions for each stock in the portfolio.

2.3.3 Reward:

The reward is determined by the actual prices of the stocks. As a high level idea, a positive reward is attributed to price increases, while a negative reward is linked to price decreases. This is obtained using the difference between bought price and sold price if a stock has been bought before, else a zero, for each stock and is represented as $k \times 1$ vector. This reward system is produced via the actions vector taken for this state.

The model is expected to adapt to the dynamic real time markets and enhance the ability to make informed decisions for portfolio optimization.

2.3.4 Experience Replay:

The actual training of the fully connected layer occurs in this step using the values stored with the help of `store_transition` method of the agent. These transitions stored in the form of (state, actions, reward, next_state, done) in the replay_buffer as a part of agent's memory. These values are unfolded in the experience replay step one-by-one and the updated Q value is sent for training of the tf model.

2.3.5 Training and Evaluation:

The Q values produced via a prediction are applied to the Q learning formula, and updated accordingly. So it is a $k \times 3$ array. This is achieved in the `exp_replay` method of the agent. The training part executed in the `portfolio_opt.ipynb` file, which takes care of performing actions and assigning action based on them to be store in the agent's inventory, apart from further using them in train.

After integrating LSTM - DQN framework, the framework is evaluated using historical data and tested on unfamiliar data to assess its performance, earlier separated. Evaluation is done using the test data in the same file and a model portfolio is obtained using the DQN framework which consists of stocks and the position of stocks(buy/sell/hold) on a particular day for test length number of days.

3 References:

- Q-Trader
- 181_multivariate_timeseries_LSTM_GE.py
- CodeTrading
- Deep Reinforcement Learning in Action by Brandon B. and Alexander Z.