Quantitative Trading with Deep Reinforcement Learning

Reinforcement Learning and Sequential Decision Making

Farhanur Rahim Ansari
ansari.f@northeastern.edu

Shubhi Saxena
saxena.shu@northeastern.edu

Abstract

The financial market is a complex system, influenced by news, political views and situations, economy, etc. It makes them subject to much uncertainty. Here, the shares of public companies are traded, meaning that investors worldwide can buy and sell their stock shares. Here the trader can "hold," "buy," or "sell" the stocks, based on estimating if the value of the stocks in the market would increase or decrease. With so many factors, agents, and traders, each with different goals and returns in the account, it is nearly impossible to comprehend the market's state at a given time. All this makes Reinforcement Learning's applications in the Finance domain one of the most popular and widely researched areas.

Since the market space is never fully observable, it becomes an appropriate problem that can be efficiently tackled by reinforcement learning principles. Since the environment here is partially observable, we can use reinforcement learning's model-free algorithms and approaches. In real-world trading, the traders develop an intuition to "buy" or "sell" stocks based on their previous knowledge and the profit or loss they earned. We can replicate this in reinforcement learning since the agent "learns" to "buy" or "sell" stocks based on the returns (profit or loss) to produce an optimal policy that tells when to "buy" and "sell" in a simulated environment. Ideally, an effective agent must derive a representation of the environment from high-dimensional input and generalize experience to future and new situations as output. Keeping in mind the complexity and nuances of the market, it is beyond this class's scope to create a simulator from scratch; instead, we make sting simulators handling portfolio optimization.

Problem Statement

This project aims to implement a deep reinforcement learning technique, deep deterministic policy gradient (DDPG), to trade a portfolio of multiple stocks. Our end goal is to maximize the user's total assets and optimize the portfolio by deciding how many stock units to trade for each of them. To better understand the domain and check our model's performance, we will also be implementing traditional Modern Portfolio Theory and a Deep Learning approach (RNN-LSTM) for optimal trading.

Dataset and Preprocessing

We downloaded the dataset using the quandl library. It consists of typical stock trading data from Jan 2009 to Feb 2019. The dataset consists of features: date, open, high, low, close, adjusted close, and volume (OHLCV). We have stored information about different stocks as additional CSV files. Other contextual data, such as DJIA(Dow Jones Industrial Average), S&P 500, etc., are used as well. The agent is trained on a training dataset sliced from 2009 - 2017 and a testing dataset sliced from 2017 - 2019. We have maintained this distribution throughout the project, across various models.

The next step after acquiring the dataset was to perform data preprocessing and feature engineering. It was essential, as, without a cleaned data, the agent may perform poorly. The preprocessing step included normalizing the time series data, cleaning up the mismatching timestamps and date formats. After this, we aggregated all the stocks' data from different files to a single dataframe for each day and brought down the features to the same data types. We implemented functions to calculate benchmark values and other variables, which are essential to understand trading results like yearly-return, gain-to-pain-ratio, Sharpe-ratio, etc., using time series data as input a step towards feature engineering.

Since the amount of data we are using is huge, and not all features are required and essential, the dataset is filtered to only provide the agent with high importance features. Collinear variables in the dataset with a correlation value more significant than a specific correlation coefficient are removed/dropped. The feature selection step selects about 6-8 features out of 41 OHLCV and technical data. In total, there are 121 features.

A collection of 40+ companies for doing project analysis. These companies belong to diverse sectors of business. It is decided to keep the number of portfolio stocks to be 5 for the optimization, keeping in mind the time taken to run the algorithm on each stock and at the same time having a reasonable estimate to compare the results.

Algorithms

There are multiple approaches from different domains that are used in solving the problem of Portfolio Management. We are going to work on three major strategies used by experts. One is the traditional approach of Modern Portfolio Theory, which is mainly used by stockbrokers and traders, the other is a supervised approach using Deep Learning, and the last and the most important one is based on Deep Reinforcement Learning.

I. Modern Portfolio Theory

Modern Portfolio Theory (MPT) is a theory on how risk-averse investors can construct portfolios to maximize expected return based on a given market risk level. MPT can also build a portfolio that minimizes risk for a given level of expected return. MPT is handy for investors trying to build efficient portfolios using Exchange Trade Funds (ETFs). There are many strategies and mathematical models under MPT, which the traders use while performing portfolio optimization.

Buy and Hold is a long-term passive strategy where investors keep a relatively stable portfolio over time, regardless of short-term fluctuations. Buy and Hold investors tend to outperform active management, on average, over longer time horizons and after fees, and they can typically defer capital gains taxes. Critics, however, argue that buy-and-hold investors may not sell at optimal times.

The **Sharpe ratio** adjusts a portfolio's past performance, or expected future performance, for the investor's excess risk. A high Sharpe ratio is reasonable when compared to similar portfolios or funds with lower returns. The Sharpe ratio has several weaknesses, including an assumption of normally distributed investment returns.

The formula for calculation of Sharpe ratio:

$$Sharpe\ Ratio\ =\ (R_p - R_f)/\sigma_p$$

where,

R_p : return on Portfolio

R_f : risk-free rate

σ_p : standard deviation of the portfolio's excess return

The **Dow Jones Industrial Average (DJIA)**, or only "the Dow," is a widely-watched benchmark index in the U.S. for blue-chip stocks. The DJIA is a price-weighted index that tracks 30 large, publicly-owned companies trading on the New York Stock Exchange and the NASDAQ.

$$DJIA = SUM(Component\ stock\ prices) / Dow\ Divisor$$

Here, the Dow Divisor is a predetermined constant used to determine the effect of a one-point move in any of the approximately thirty stocks that comprise the Dow. Its value keeps on changing over time. Its current value is 0.14748. Critics believe that factoring only stock prices in the calculation does not accurately reflect a company, as much as considering a company's market cap would. In this manner, a company with a higher stock price but a smaller market cap would have more weight than a company with a smaller stock price but a larger market cap, which would poorly reflect its actual size.

MPT is the most widely used strategy by investors. However, it is calculated only based on stock returns. Suppose we want to consider other relevant factors, for example, some of the technical indicators like Moving Average Convergence Divergence (MACD) and Relative Strength Index (RSI). In that case, MPT may not be able to combine this information well. As a result, it is susceptible to outliers and has a very volatile prediction. Overall it does not perform so well for out-of-sample data. Although it gives a decent estimate for portfolio optimization, it comes with an overhead that can be solved using the power of Artificial Intelligence and Reinforcement learning.

II. Deep Learning

Deep Learning is a subset of machine learning which uses artificial neural networks to learn non-linear functions or relationships from data. Artificial neural networks are designed to identify underlying trends in the data and generalize from it. They usually consist of three layers, namely, input, hidden layer, and output layer. Non-linear activation functions are used in all layers of the network except for the input layer. A deep learning network is built by connecting each neuron in a layer to every neuron in the next layer.

Recurrent Neural Networks (RNN) take input from two sources; one is from the present and the other from the past. Information from these two types of input is used to decide how they react to unseen data. This is done with the help of a feedback loop. Each input sequence has plenty of information and is stored in the hidden state of the recurrent network. This hidden information is recursively used in the network as it sweeps forward to deal with new examples.

Long Term Short Memory (LSTM) is a particular type of RNN. They are specially built to learn about long term dependencies. Conventional RNN has a very simple architecture with a feedback loop, but LSTM consists of memory blocks instead of a single neural network layer. Each block has three gates, and a cell state tends to regulate the flow of data information.

RNNs and LSTMs are very popular in the time series forecasting domain. RNNs are good at processing sequential data, which holds true in our context of stock market prediction. LSTMs are specially designed to help RNN better memorize the long-term context.

We built the RNN-LSTM model using Keras library, with two hidden layers and a dropout of 0.5, the loss function as the mean absolute error (MAE), optimizer with decay value of 0.00001, rho as 0.95, and metrics defined with MAE as well as mean squared error (MSE).

III. Reinforcement Learning

Reinforcement Learning is one of the three approaches to machine learning techniques. It trains an agent to interact with the environment by sequentially receiving states and rewards from the environment and taking actions to reach better rewards. Deep Reinforcement Learning approximates the Q-value with a neural network. Using a neural network as a function approximator would allow reinforcement learning to be applied to large data. Bellman Equation is the guiding principle to design reinforcement learning algorithms. Markov Decision is used to model the environment.

Recent application of deep reinforcement learning in financial markets consider discrete or continuous state and action spaces and employ one of these learning approaches: critic only approaches, actor-only approach, or actor-critic approach.

The critic-only approach's idea is to use a Q-value function to learn the optimal action selection policy that maximizes the expected future reward given the current state. Instead of calculating the state-action value table, Deep Q-learning (DQN) minimizes the mean squared error between the target Q-values and uses a neural network to perform optimization. The critic-only approach's primary limitation is that it only works with discrete and finite state and action spaces.

In the Actor-only approach, the agent directly learns the optimal policy itself. Instead of having a neural network to learn the Q-value, the neural network learns the policy. The actor only approach can handle continuous action space.

The Actor-critic approach's idea is to simultaneously update the actor-network representing the policy and the critic network representing the value function. The critic estimates the value function, while the actor updates the policy probability distribution, guided by the policy gradients' critic.

Environment Setup

We developed a trading environment using the OpenAI gym platform to achieve our objectives. The environment has the basic Reinforcement Learning structure with the defined state, action, and rewards.

States - The state of the trading environment is defined by account balance, unrealized profit, and loss, relevant stock technical data & current stock holdings.
Actions - Buy and Sell operation for each stock along with the quantity of trading.
Rewards - The total asset gain/loss by the end of each day.
Episode - An episode ends when the timestamp reaches the last day in the feature dataset. The environment will reset after the end of each episode. A gamma value of 0.99 was applied to calculate the cumulative rewards.

Trading's separate class was created, which performs trading for a given stock while calculating the remaining stocks after trading, creating a trading book, and finding non-correlated stocks from a portfolio of stocks. When trading is executed, 121 features along with total asset, current asset holdings, and unrealized profit and loss will form a complete state space for the agent to trade and learn.

Deep Deterministic Policy Gradient (DDPG)

DDPG is an actor-critic-based model-free off-policy algorithm for learning continuous actions, which we have implemented to maximize the investment return. DDPG combines the framework of both Q-learning and policy gradient and uses neural networks as function approximators. It uses Experience Relay and slow-learning target networks from DQN, and it is based on policy gradient, which can operate over continuous action spaces. In contrast with DQN that learns indirectly through Q-values tables and suffers the curse of the Dimensionality problem, DDPG learns directly from the observations through policy gradient. It is proposed to deterministically map states to actions to fit the continuous action space environment better.

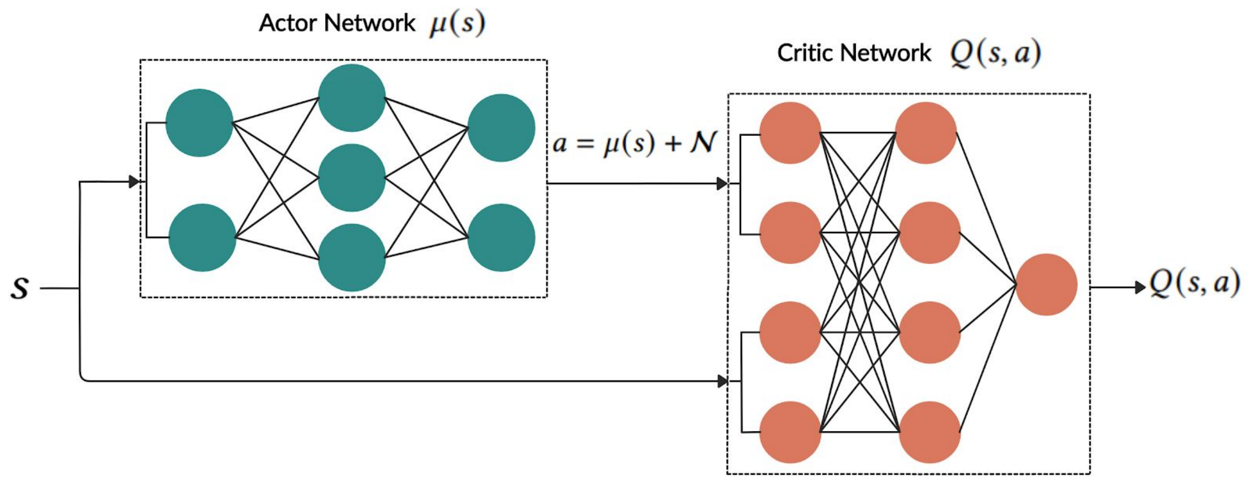


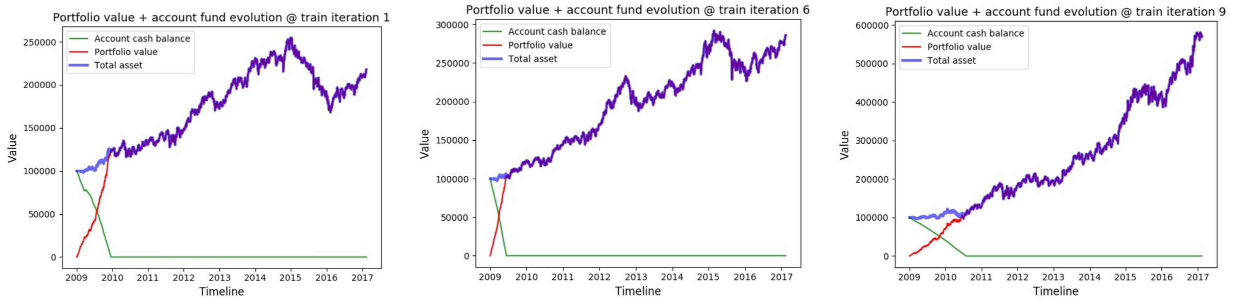
Fig 1: Deep Deterministic Policy Gradient's Actor-Critic based neural network architecture

We define our agent, memory, and noise wrappers for openAI Gym Environment. The actor takes action given states, and the critic evaluates the payoffs of the action taken by the actor. The DDPG algorithm is that it directly maps states to deterministic actions, which is a more desirable property in production, while the actor-critic algorithm outputs the probability distribution across a discrete action space.

The DDPG model's neural network architecture was built on a fully connected Multi-Layer Perceptron with two hidden layers and 64 hidden units, and a tanh activation function. Hyperparameter tuning is applied. The best results are achieved at the 1e-4 learning rate for the actor-network and 3e-4 learning rate for the critique network and an l2 regularization of 0.01 to prevent overfitting. The network is trained in a batch size of 128 with 20 epochs and 100 steps.

Results

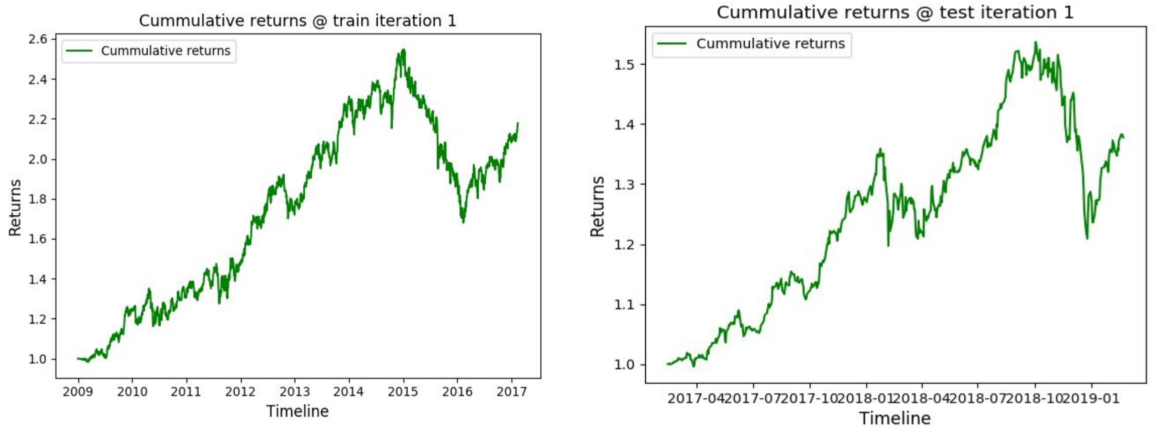
The DDPG model is trained on a portfolio of 5 non-correlated stocks (portfolio1) - American Express, Walmart, UnitedHealth Group, Apple, and Verizon Communications, for an eight-year time period, starting from 01-01-2009 till 02-22-2017. It is trained for a total of 9 epochs for better results.



(a) (b) (c)

Fig 2: Total asset (Portfolio value + account cash balance) evolution of DDPG training agent with timeline. (a) train iteration 1 (b) train iteration 6 (c) train iteration 9.

In each iteration, it starts with an account cash balance of 100K$. That money is then invested in the portfolio, and the total asset value is observed over the training period. We can observe that in each of the plots of fig 2, the total asset value has a generally increasing trend, and over the iterations, we can see an increase in the range of the asset value starting from 250K$ in the first iteration to 600K$ in the 9th iteration.



(a) (b)

Fig 3 (a) and (b) shows the learning curve for cumulative returns at first iteration during training and testing

Our portfolio trading agent's testing period was for a two-year duration, starting from 02-12-2017 till 02-12-2019. Comparing our cumulative returns plots for our training and evaluation period in fig 3, we observe that the model performs better during the training period having a maximum return of 2.6 as compared to testing, having a maximum return of 1.5 only. This result was expected from our model.

Performance of the trading agent with Deep Deterministic Policy Gradient is evaluated using the Sharpe ratio and compared with both the Dow Jones Industrial Average index and the traditional buy and hold trading strategies. A thorough comparative analysis of the DDPG and RNN-LSTM models is done to better understand the optimization problem within the machine learning domain.

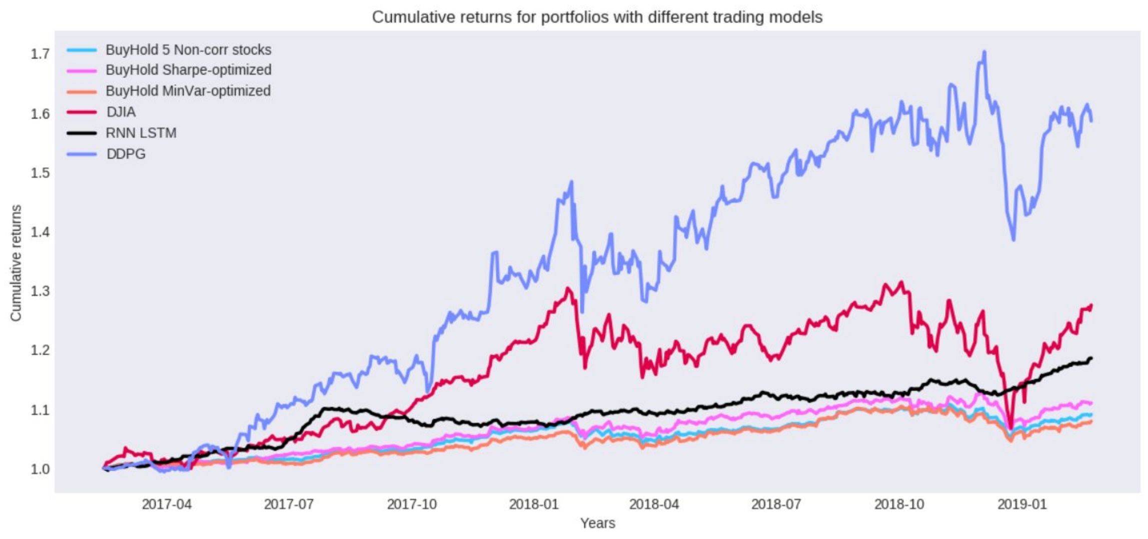


Fig 4: Cumulative return of different models on the testing dataset, with DDPG performing the best and RNN LSTM performing worst than expected

DJIA is the industry benchmark used by stockbrokers for portfolio management. The Buy and Hold strategy is the traditional approach adopted by many. RNN-LSTM is the other machine learning counterpart widely used by technical people. It is impressive to know that the DDPG approach outperformed all the other trading approaches in terms of cumulative returns. This can be verified from Fig 4. However, the RNN-LSTM model is much less volatile and has a better risk-adjusted return (Sharpe Ratio of 1.88).

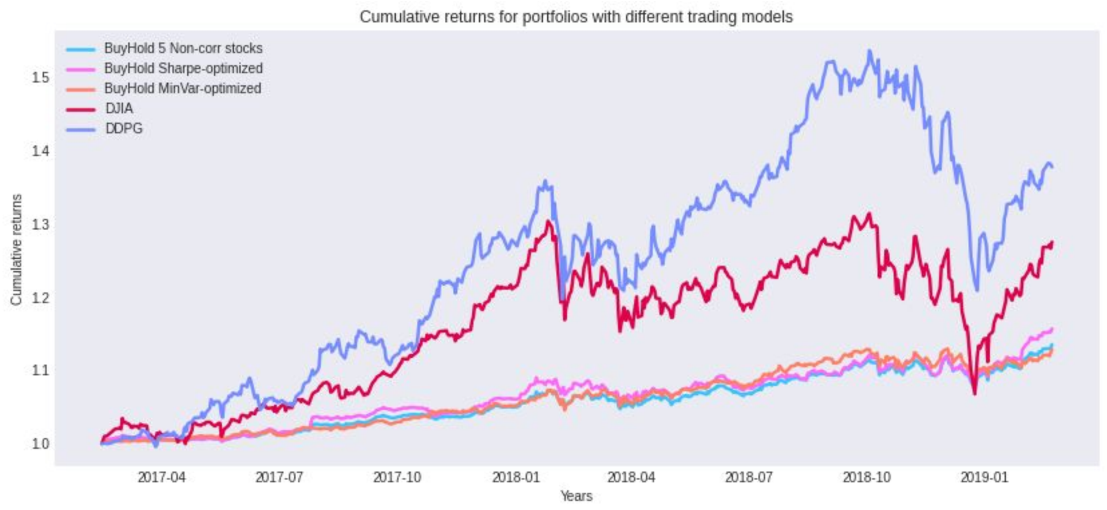


Fig 5: Different trading model's performance on an alternate portfolio

We also tried running our trading models on a different portfolio to check our solution's generality out of curiosity. An alternate portfolio of 5 non-correlated stocks was selected (portfolio2) - Boeing, Merck, Procter & Gamble, Pfizer, and Microsoft. From fig 5, we can observe out of sample data in this new case; the DDPG model outperforms the other trading models. The DDPG model, trained on portfolio1, achieves a maximum cumulative return of 1.5 on the testing dataset of portfolio2, whereas the best performing DJIA model achieves the maximum cumulative return of only 1.3. This indicates that our model is a generalized solution that performs fairly on other portfolios as well.

Discussion

Quantitative Trading is a very vast and versatile domain, with an influx of new strategies, improvements over old ones coming in each day. It's hard to understand the nuances of the domain, and very much, so a lot of pundits develop a better and clearer sense of the market after being associated with it for a few years. This made the topic and the project intellectually challenging and broadened our understanding of reinforcement learning applications in the financial domain.

In conclusion, we were curious to see how Reinforcement Learning would perform over a period of time and how efficiently it would be able to handle the volatile nature of the stock market. This is why we made sure to include as much data as possible for our trading environment. This made preprocessing and feature engineering extremely important steps apart from the traditional environment setup as our initial step. Implementation of DDPG and other benchmark metric calculations formed the crux of our project.

The results were as expected; the agent performed better in the long run than the traditional MPT algorithms. Our DDPG model even outperformed the benchmark DJIA model often used by experts for out of sample data. Although the forecast of the Deep Learning approach of RNN-LSTM gave less volatile results for comparisons within the machine learning domain, their outcome in terms of cumulative returns was still way less than the Deep Reinforcement Learning approach. Our solution even gave fairly better results for new portfolios of different stocks at the top of this.

References

1. *Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver & Daan Wierstra, Continuous control with Deep Reinforcement Learning, (2016) ICLR.*
2. *Better Exploration with Parameter Noise, openAI Gym blog post.*
3. *Hongyang Yang, Xiao-Yang Liu, Shan Zhong, and Anwar Walid, Deep Reinforcement Learning for automated stock trading: An Ensemble Strategy, (2020) SSRN.*
4. *StarTrader: Intelligent Trading Agent Development with Deep Reinforcement Learning, Github.*
5. *Modern Portfolio Theory, Buy and Hold Strategy, Sharpe Ratio, Dow Jones Industrial Average (DJIA), from Investopedia.*
6. *NSE Stock Market Prediction Using Deep Learning models, Hiransha M, GopalKrishnan E.A., Vijay Krishna Menon, Sonam K.P.*