

---

# BUFN403 Project Report

---

**Sumit Nawathe   Ravi Panguluri   James Zhang   Sashwat Venkatesh**  
Computational Finance Minor Capstone  
University of Maryland College Park  
{snawathe, rpangulu, jzhang72, sashvenk}@terpmail.umd.edu

## Abstract

We aim to create a reinforcement learning agent that utilizes historical price data as well as sentiment and topical embeddings of news articles to optimally trade on S&P100 stocks. After implementing multiple papers on financial reinforcement learning for equity trading, our contribution will be in experimenting and synthesizing an improved combination of the state space and reward function for our environment. We compare our best strategy against other standard equity portfolio benchmarks.

# Contents

# Chapter 1

## Project Overview

### 1.1 Introduction

Our group is seeking to develop a reinforcement learning agent to support portfolio management and optimization. Utilizing both empirical stock pricing data along with alternative data, we look to create a more well-informed portfolio optimization tool.

Our primary motivations for pursuing a reinforcement learning-based approach are as follows:

1. Reinforcement learning lends itself well to learning/opening in an online environment. The agent can interact with its environment, providing real-time feedback/ responsiveness to allow for better results.
2. Our approach involves incorporating alternative data to support the agent's decision making process. Encoding this alt-data into the states matrix of the agent allows for the agent to make better decisions when it comes to adjusting portfolio weights.
3. Given that a reinforcement learning agent's decisions are modeled by a Markov Decision Process, we can easily provide different reward functions to account for a variety of investor preferences or restrictions.

### 1.2 Dataset Creation

Creating a multimodal dataset of tabular and language data to train a robust reinforcement learning agent will require pulling data from a wide variety of sources. We aim to use three primary types of data: historical stock data, news headlines, and SEC filings.

#### 1.2.1 Stock Data

For historical price data, we will use data from the Wharton Research Data Services (WRDS). Specifically, we aim to use data from the Center for Research in Security Prices (CRSP), which has security price, return, and volume data for the NYSE, AMEX and NASDAQ stock markets. We will begin by creating a universe with just the S&P 100 stocks. If time permits, we hope to expand to the S&P 500 and/or stocks on all publicly-listed exchanges available through WRDS.

#### 1.2.2 News Data

The inclusion of news data into the reinforcement learning agent's state is important to integrate an external understanding of how well a given stock is performing at a given time and how it is exposed to certain market or sector risks. This can provide our agent a better view of the environment and knowledge of our ticker companies in context, which can help it make better decisions to maximize our chosen reward function.

## Scraping Financial News Sources

While attempts have been made to scrape news data from financial sources including Yfinance and Benzinga, we have primarily been rate limited/and or IP blocked for scraping at too high of a frequency. From Kaggle, we have found a source of pre-scraped news data from Benzinga — a financial news source — with daily news headline data spanning from 2014-2020 for over 6000 companies. From this dataset, we plan to start with using headline data companies in the S&P 100. However, we could expand the dataset if time and data storage capacity permits.

## Utilizing Paid News APIs

Seeing a gap in real-time and historical data pipelines for business and professional use, many companies have created paid news APIs that allow institutions to query a wide range of news sites for current event information including financial news. Many of these APIs such as Event Registry, newsapi.ai, and Alpha Vantage provide this data. However, in examining most of these APIs there are a few significant issues. Many of them require a significant payment to get data at a velocity that we would need, and historical data is often even more expensive.

However, within the free tier of these services we can make some API calls to query information for some tickers that could be additive to our analysis. It would be difficult to rely on this as a long term solution for data though.

## SEC Filings

To enrich our dataset, we are considering utilizing SEC (10-K and 10-Q) filings data for all NYSE/-NASDAQ companies (available through WRDS) that we plan to pull data for. These filings contain essential information regarding a company's financial performance, governance, and compliance that could enhance our measure of company outlook. Specifically we aim to use data from 10-K (annual) and 10-Q (quarterly) filings. 10-K's are an annual report on a company's performance, and include information. For our project, the textual data that is most likely to capture a company's sentiment is Item 1A, Risk Factors. This item is a company issued statement on the risk factors that could affect the operations of the business for the next fiscal year. We will extract this data from 10-Q statements for every company in our trading universe to create sentiment indicators for our RL agent to use.

### 1.2.3 Sentiment Analysis

Using the news sources and SEC filings data described above, we wish to generate embeddings from which we can extract sentiment related features to provide to our reinforcement learning agent. Our initial approach, which we have conducted some basic testing on, is to utilize the pre-trained FinBERT model, fine-tuned to recognize the sentiment of financial text to create embeddings for us [? ].

## FinBERT Sentiment Scores

We query article headlines for all stocks in our universe over our entire dataset time period. Note that if we use a source like Benzinga, articles are already tagged with relevant stock tickers. We will feed headline data to pre-trained FinBERT. The model then will preprocess the text and generate probabilities of the content being positive, negative, or neutral. From there, we can assign each headline a numerical score based on its maximum probability class. The numerical map could look something like the following: positive: 1, neutral: 0, negative: -1 Over a trading period, we can take some aggregate of these class labels for each stock and feed these class labels to our agent's states matrix; at each time step, this value will be appended to the row corresponding to the stocks' price data. (The state matrix will be defined in greater detail in the Algorithmic and Analytical Challenges section.) We will experiment to find an optimal aggregate function; potential options include providing all logits, taking the mean, or designing a custom function to extract value heuristically. One such function could be

$$\text{Value}_{\text{Embedding}} = \tanh\left(\frac{\frac{\text{positive sentiment probability}}{\text{negative sentiment probability}}}{\text{neutral sentiment probability}}\right)$$

This approach combines the “log likelihood” (ratio of probabilities of positive and negative sentiment) along with a penalty for high neutral sentiment (a measure of uncertainty), using the tanh for normalization. This approach would allow us to adequately detect strong positive/negative sentiment. We will compare this against other aggregate functions in training experiments.

We will test the exact same preprocessing pipeline on the SEC 10-K and 10-Q filings for each company in our universe and integrate them into our states matrix. We will experiment with the same options mentioned previously for the optimal aggregation function. An issue that incorporates SEC filings is that they are recorded on a relatively infrequent basis compared to news and price data. Preliminarily, we will assume that there is no decay in the sentiment between reports. That means, the sentiment embeddings for each company will only be updated on dates where a new filing was reported. On non-reporting dates, the embeddings will be filled forward from the last filing date.

Alternatively, we intend to use a logistic decay function to generate a multiplier that represents the decaying effect of sentiment over time from a document like a 10-K or 10-Q, which is not reported every day. The function we’ve designed is:

$$Multiplier = 2 \left( \frac{-1}{1 + e^{-0.25x}} + 1 \right)$$

Generating a multiplier, then combining this multiplier with the sentiment score generated by FinBERT will allow us to show that some  $x$  days after a 10-K or 10-Q filing is published, the actual effective sentiment wanes as the news becomes older. At 12 days, the sentiment is modeled to be roughly 10 percent as “effective” as it was on day 0. At day 0, the multiplier will always be 1, to indicate that news is most impactful on the day that it is published.

## Topic Modeling

In a similar manner to the Financial Statement Analysis assignment from Andy Chakraborty’s lecture, we can also utilize FinBERT to categorize news headlines and content into financial-related topics. As demonstrated in that assignment, the correlations between such scores and the performance of companies can be useful to our RL agent by similarly incorporating such embeddings as a tensor. For every news headline and SEC filing related to each stock ticker, we will use the same pre-trained topic classification model as we did in our assignment to give us a numeric mapping of each text’s most probable topic. Using this, we will append a column of topic embeddings for each ticker on each day to the states matrix.

## 1.3 Algorithmic and Analytical Challenge

Our primary algorithmic technique is deep reinforcement learning, which uses deep neural networks to learn an optimal policy to interact with an environment and optimize performance towards a goal. Formally, a reinforcement learning problem is an instance of a Markov Decision Process, which is a 4-tuple  $(S, A, T, R)$ :  $S$  the state space (matrix of selected historical stock price and news data available to our model at a given time; see Methodology section),  $A$  the action space (portfolio weights produced by our model, under appropriate constraints),  $T$  the transition function (how the state changes over time, modeled by our dataset), and  $R$  (the reward function). The goal is to find a policy (function from  $S \rightarrow A$ ) that maximizes future expected rewards. Most reinforcement learning research is spent on providing good information in  $S$  to the model, defining a good reward function  $R$ , and deciding on a deep learning model training system to optimize rewards.

### 1.3.1 Existing Literature

Much of the literature applying RL to portfolio optimization has arisen in the last few years. Some relevant papers are:

- [?] Deep Reinforcement Learning Comparison with Mean-Variance Optimization: Using a lookback of recent past returns and a few market indicators (including 20-day volatility and the VIX), this paper implements a simple algorithm for portfolio weight selection to maximize the Differential Sharpe Ratio, a (local stepwise) reward function which approximates

(global) Sharpe Ratio of the final strategy. They compare their model with the standard mean-variance optimization across several metrics.

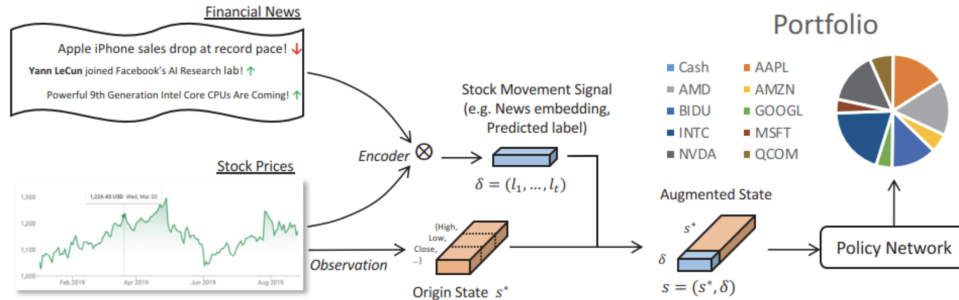
- [?] DRL for Stock Portfolio Optimization Connected with Modern Portfolio Theory: This paper applies reinforcement learning methods to tensors of technical indicators and covariance matrices between stocks. After tensor feature extraction using 3D convolutions and tensor decompositions, the DDPG method is used to train the neural network policy, and the algorithm is backtested and compared against related methods.
- [?] RL-Based Portfolio Management with Augmented Asset Movement Prediction States: The authors propose a method to augment the state space  $S$  of historical price data with embeddings of internal information and alternative data. For all assets at all times, the authors use an LSTM to predict the price movement, which is integrated into  $S$ . When news article data is available, different NLP methods are used to embed the news; this embedding is fed into an HAN to predict price movement, which is also integrated into  $S$  for state augmentation. The paper applies the DPG policy training method and compares against multiple baseline portfolios on multiple asset classes. It also addresses challenges due to environment uncertainty, sparsity, and news correlations.
- [?] A Deep Reinforcement Learning Framework for the Financial Portfolio Management Problem: This paper contains a deep mathematical and algorithmic discussion of how to properly incorporate transaction costs into an RL model. The authors also have a GitHub with implementations of their RL strategy compared with several others.
- [?] Stock Portfolio Selection Using Learning-to-Rank Algorithms with News Sentiment: After developing news sentiment indicators including shock and trends, this paper applies multiple learning-to-rank algorithms and constructs an automated trading system with strong performance.
- [?] MAPS: Multi-agent Reinforcement Learning-based Portfolio Management System: This paper takes advantage of reinforcement learning with multiple agents by defining a reward function to penalize correlations between agents, thereby producing multiple orthogonal (diverse) high-performing portfolios.

### 1.3.2 Methodology

We will be implementing, combining, and improving on the methodologies of several of the above papers. Our plan is to develop an RL system that utilizes multiple time periods to achieve strong out-of-sample trading performance. As of this writing, we have partial implementations of papers [?], [?], and [?]. Our final architecture will be most similar to papers [?] and [?].

#### Markov Decision Process Problem Formulation

Paper [?] includes the following diagram, which is very close to our desired architecture:



An explanation of this diagram: at time  $t$ , the origin state  $S^*$  is a 3D tensor of dimensions  $U \times H \times C$  which contains historical price data.  $U$  is the size of our universe (for example, for the S&P100,  $U = 100$ ).  $H$  is the size of history we are providing (if we are providing 30 day history, then  $H = 30$ ).  $C$  is a categorical value representing the close/high/low price. This format of  $S^*$  allows us to store, for example, the last 30 days of stock price data for all companies in the S&P100, for any given day. In addition to this, we have news information  $\delta$ , obtained from financial news headlines

for that day, processed through a pre-trained encoder. This information is added to  $S^*$  to create the full state  $S = (S^*, \delta)$

In our architecture, for  $S^*$ , we will experiment with the lookback period size and likely reduce it to a 2D array by flattening along the C index, but will otherwise keep  $S^*$  largely the same. For  $\delta$ , we plan to utilize better feature extraction via sentiment scores and topic modeling; we also plan to use different alternative data sources, as described in the Dataset Creation section. In addition, we will extract what company each headline refers to, so our features can be changed over time independently for each company as news articles enter through our environment. The final state  $S$  will likely be a 2D matrix, where each row represents a different company (ticker), and along that row we find, concatenated, the following: (1) the past month-or-so of stock price data from  $S^*$ , and (2) numerical features extracted from recent news data pertinent to that company (as described in the Dataset section). (The straightforward concatenation of price data and news embeddings did not affect the ability of the neural network-based agent to learn.)

Regarding the reward function  $R$ , we plan to experiment with both the profit reward function used in paper [? ], as well as the Differential Sharpe Ratio developed in paper [? ]. The former is simply the change in the portfolio value over the last time period based on the weights (action) provided by the agent; the latter attempts to make the cumulative reward approximate the Sharpe ratio over the entire time period.

In all of the papers, the action space  $A$  is a length  $m + 1$  vector such that the sum of all element is 1, where there are  $m$  stocks in our universe (the other weight is for the risk-free asset). Each action represents the agent’s desired portfolio for the next time period, and is computed based on the state at the end of the previous time period. We will experiment with short-selling and leverage restrictions (which put lower and upper bounds on the weight components, respectively).

In summary, our project aims to implement and replicate the approach used in [? ], with some modifications to  $S$  and  $R$  as previously described. We will conduct experiments alternative data sources, feature extraction methods, and reward functions (both custom and from other papers listed) to find a good combination that allows this approach to work well on S&P100 stocks; this comprises our novel extension/contribution.

## Use of Libraries

We will mainly be using the Gymnasium library to implement the reinforcement learning environments. The Stable Baselines 3 library provides several policy learning techniques that we will experiment with, including Proximal Policy Optimization (PPO) and Deep Deterministic Policy Gradients (DDPG). The papers above discuss the advantages and disadvantages of multiple reward functions and constraints, which we will make improvements upon and provide as options to the user, if applicable.

## Strategy Benchmarking

Our final model architecture will be compared against several benchmark financial portfolio selection models. Among these will be the CAPM, an exponential moving average strategy, linear factor models such as the Fama French 3/5-factor models, and the QMJ model. We will compare our returns in-sample and out-of-sample plots, as well as our relative performance on portfolio statistics including cumulative return, Sharpe Ratio, Sortino Ratio, drawdown, etc. The experiment sections in the above papers provide a strong reference for our methodological comparison.

## 1.4 User Interface and Visual Analytics

As our project is primarily oriented towards data curation and algorithms, we will not directly have a user interface for a final end-user. However, in order to experiment with and visualize the performance of our various models, we will create a small suite of visualizations in an interactive manner that allows a researcher to choose methods and hyperparameters for our architecture. We will also create a notebook to demonstrate the infrastructure and usage of our deep reinforcement learning model, similar to those on TensorFlow and Huggingface.

## Chapter 2

# Literature Notes and Commentary

### 2.1 Reinforcement Learning Overview

The reader may not be readily familiar with reinforcement learning (RL). Thus, we here provide a brief overview of the terminology, basic definition, essential results, and common algorithms in RL theory.

#### 2.1.1 Markov Decision Process

A Markov Decision Process (MDP) problem is a framework for modeling sequential decision-making by an agent in an environment. A problem is formally defined as a 4-tuple  $(S, A, T, R)$ .

- $S$  is the state space, which is the set of all possible states of the environment.
- $A$  is the action space, which contains all possible actions that the agent can take (across all possible states).
- $T : S \times A \times S \rightarrow [0, 1]$  is the transition function in a stochastic environment. When the environment is in state  $s$  and the agent takes action  $a$ , then  $T(s, a, s')$  is the probability that the environment transitions to state  $s'$  as a result. (In a deterministic environment, this function may not be necessary, as there may be only one possible state due to the taken action.)
- $R : S \times A \times S \rightarrow \mathbb{R}$  is the reward function. When the environment changes state from  $s$  to  $s'$  due to action  $a$ , the agent receives reward  $R(s, a, s')$ .

The environment is Markov, which means that the distribution of the next state  $s'$  conditioned on the current state  $s$  and action  $a$  is independent from the time step.

A policy is a function  $\pi : S \rightarrow A$  that dictates actions to take at a given state. A solution to an MDP problem is an optimal policy  $\pi^*$  that maximizes the agent's utility, however that is defined.

#### 2.1.2 RL Terminology

In RL, the agent's utility is generally defined as the total expected discounted reward. Let  $\gamma \in [0, 1]$  be a constant discount factor. The utility from a sequence of reward  $\{r_t\}_{t=0}^{\infty}$  is thus commonly defined as  $U([r_0, r_1, \dots]) = \sum_{t=0}^{\infty} \gamma^t r_t \leq \frac{\sup_t r_t}{1-\gamma}$ . The benefit of this formulation is that (1) utility is bounded if the rewards are bounded, and (2) there is a balance between small immediate rewards and large long-term rewards. (The use of the discount factor depends on the actual reward function. For custom reward functions, it may not be necessary or even desirable; we include it because it is common in RL literature.)



Given a policy  $\pi : S \rightarrow A$ , we define the value function  $V^\pi : S \rightarrow \mathbb{R}$  and the  $Q$ -function  $Q^\pi : S \times A \rightarrow \mathbb{R}$  as

$$V^\pi(s) = \mathbb{E}_\pi \left[ \sum_{t=0}^{\infty} \gamma^t R_t \mid s_0 = s \right] \quad Q^\pi(s, a) = \mathbb{E}_\pi \left[ \sum_{t=0}^{\infty} \gamma^t R_t \mid s_0 = s, a_0 = a \right]$$

$V^\pi(s)$  is the expected utility from starting at  $s$  and following policy  $\pi$ , and  $Q^\pi(s, a)$  is the expected utility from starting at  $s$ , taking action  $a$ , and then following policy  $\pi$  thereafter. The goal of RL is to find the optimal policy  $\pi^*$ , from which we have the optimal value function  $V^*$  and optimal  $Q$ -function  $Q^*$ . These optimal values can further be defined as follows:

$$Q^*(s, a) = \mathbb{E}_{s'} [R(s, a, s') + \gamma V^*(s')] = \sum_{s'} T(s, a, s') [R(s, a, s') + \gamma V^*(s')] \\ V^*(s) = \max_a Q^*(s, a) = \max_a \sum_{s'} T(s, a, s') [R(s, a, s') + \gamma V^*(s')]$$

The second equation is known as the Bellman equation.

There are two main branches of RL: model-based and model-free. In model-based RL, the agent attempt to build a model of the environment transition function  $T$  and reward function  $R$ . Based on these model, it then attempts to directly maximize the total expected reward. In model-free RL, the agent does not attempt to model the environment, but instead attempts to learn either the value function or  $Q$ -function. Once it has one of these, it can derive an optimal policy from it as:

$$\pi^*(s) = \arg \max_a Q^*(s, a) = \arg \max_a \sum_{s'} T(s, a, s') [R(s, a, s') + \gamma V^*(s')]$$

We proceed with model-free RL in this project.

### 2.1.3 Specialization to Our Application

In the portfolio optimization setting, our RL agent seeks to produce an optimal set of portfolio weights given all the information it knows. Assume that there are  $n$  tradeable stocks in our universe, and one risk-free asset. The action space  $A = \{a \in \mathbb{R}^{n+1} \mid \sum_i a_i = 1\}$  is the set of all possible portfolio weights.

The state space  $S$  encompasses all information available to the agent when it is asked to make a portfolio allocation decision at a given time. Depending on what information is provided, this could include past performance of the strategy, historical stock prices, encoded news information for select/all tickers in the universe, or some combination of these. For most of the scenarios we consider,  $S \in \mathbb{R}^{n \times N}$  is a matrix, where each row corresponds to a different stock ticker, and along that row we find the past few weeks of historical price data as well as some aggregate of news sentiment indicators/scores.

The transition function  $T$  is a delta function since state transitions are deterministic. The environment uses the weights provided by the agent to reallocate the portfolio, computes the new portfolio value, and reads in new historical stock data and news datapoints to form the next state (for the next time period) which is provided to the agent. (The exact for of  $T$  is not needed; it is implicitly defined by the deterministic environment updates.)

The reward function  $R$  should be such that it encourages the agent to produce good portfolio weights. One simple reward function is pure profit:  $R(s_t, a_t)$  is how much profit is gained to portfolio allocation  $a_t$  during time interval  $[t, t + 1)$ . Another possible reward function is the Differential Sharpe ratio (as described in section ??), which urges the agent to make portfolio allocations to maximize its total Sharpe ratio.

## 2.2 Differential Sharpe Ratio

[?] utilizes the Differential Sharpe Ratio to implement and evaluate a reinforcement learning agent. The Differential Sharpe Ratio is based on Portfolio Management Theory, and is developed in the author's previous works [?] and [?]. We briefly review the theory developed in both sources.

The traditional definition of the Sharpe Ratio is the ratio of expected excess returns to volatility. If  $R_t$  is the return of the portfolio at time  $t$ , and  $r_f$  is the risk-free rate then

$$S = \frac{\mathbb{E}_t[R_t] - r_f}{\sqrt{\text{Var}_t[R_t]}}$$

This works well to analyze a strategy once all data is collected. The goal of traditional portfolio theory is to maximize the Sharpe Ratio over the given time period (equivalently, to maximize the mean-variance utility function).

Unfortunately, this will not work for a reinforcement learning agent. The agent must be given a reward after every time step, but the traditional Sharpe ratio is only calculated at the end.

The Differential Sharpe Ratio attempts to remedy this by approximating a change in the total Sharpe ratio up to that point. By summing together many of these incremental changes (though approximate), the cumulative rewards is an approximation of the total Sharpe ratio over the complete time period.

The approximation works by updating moment-based estimators of the expectation and variance in the Sharpe Ratio formula. Let  $A_t$  and  $B_t$  be estimates of the first and second moments of the return  $R_t$  up to time  $t$ . After time step  $t$ , having obtained  $R_t$ , we perform the following updates:

$$\begin{aligned} \Delta A_t &= R_t - A_{t-1} & A_t &= A_{t-1} + \eta \Delta A_t \\ \Delta B_t &= R_t^2 - B_{t-1} & B_t &= B_{t-1} + \eta \Delta B_t \end{aligned}$$

where  $A_0 = B_0 = 0$  and  $\eta \sim 1/T$  is an update parameter, where there are  $T$  total time periods. These updates are essentially exponential moving averages.

Let  $S_t$  be an approximation of the Sharpe Ratio up to time  $t$  based on estimates  $A$  and  $B$ . That is,

$$S_t = \frac{A_t}{\sqrt{B_t - A_t^2}}$$

The definition here ignores the risk-free rate term.  $K_\eta$  is a normalization constant to ensure an unbiased estimator.

Pretend that at the update for time  $t$ ,  $A_{t-1}$  and  $B_{t-1}$  are constants, and  $R_t$  is also a known constant. Then the updates to  $A_t$  and  $B_t$  really only depend on the time step parameter  $\eta$ . Indeed, if  $\eta = 0$ , then  $A_t = A_{t-1}$  and  $B_t = B_{t-1}$ , so  $S_t = S_{t-1}$ . Now consider varying  $\eta$ ; expanding the Sharpe ratio estimator formula in Taylor series gives

$$S_t \approx S_{t-1} + \eta \left. \frac{dS_t}{d\eta} \right|_{\eta=0} + o(\eta^2)$$

If  $\eta$  is small, the final term is negligible, so this formula gives us an exponential-moving-average update for  $S_t$ . The Differential Sharpe Ratio is defined to be proportional derivative in that expression. With some tedious calculus, we find that

$$\begin{aligned} D_t &= \frac{dS_t}{d\eta} = \frac{d}{d\eta} \left[ \frac{A_t}{\sqrt{B_t - A_t^2}} \right] = \frac{\frac{dA_t}{d\eta} \sqrt{B_t - A_t^2} - A_t \frac{\frac{dB_t}{d\eta} - 2A_t \frac{dA_t}{d\eta}}{2\sqrt{B_t - A_t^2}}}{B_t - A_t^2} \\ &= \frac{\Delta A_t \sqrt{B_t - A_t^2} - A_t \frac{\Delta B_t - 2A_t \Delta A_t}{2\sqrt{B_t - A_t^2}}}{B_t - A_t^2} = \frac{B_t \Delta A_t - \frac{1}{2} A_t \Delta B_t}{(B_t - A_t^2)^{3/2}} \end{aligned}$$

This reward function is simple to implement in an environment. The authors of the original papers provide experimental support for the value of this reward function in a reinforcement learning setting.

## 2.3 Transaction Costs

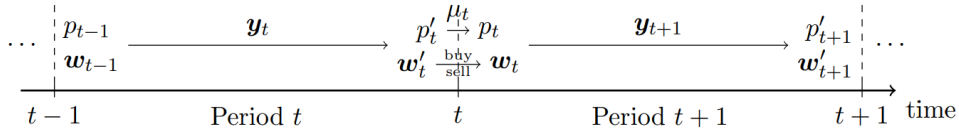
[?] contains an excellent walkthrough of the mathematics for modeling transaction costs in RL environment updates. We provide a shortened version here.

Suppose we have  $m$  tradeable assets and the risk-free asset. Let  $\mathbf{v}_t = (1, v_{1,t}, v_{2,t}, \dots, v_{m,t}) \in \mathbb{R}^{m+1}$  be the prices of the assets at time  $t$  (the first entry is the risk-free asset). The raw return vector is defined as  $\mathbf{y}_t = \mathbf{v}_t \oslash \mathbf{v}_{t-1} \in \mathbb{R}^{m+1}$ , where division is element-wise. Suppose the portfolio weight vector during time period  $t$  is  $\mathbf{w}_t \in \mathbb{R}^{m+1}$ , and let the value of the portfolio value at time  $t$  be  $p_t$ . If we were not considering transaction costs, then the portfolio return would be  $\frac{p_t}{p_{t-1}} = \mathbf{y}_t \cdot \mathbf{w}_t$ .

Unfortunately, buying and selling assets incurs transaction costs. Let  $\mathbf{w}'_t \in \mathbb{R}^{m+1}$  be the effective portfolio weights at the end of time  $t$  (it has changed from  $\mathbf{w}_t$  due to the changes in price). We have

$$\mathbf{w}'_t = \frac{\mathbf{y}_t \odot \mathbf{w}_t}{\mathbf{y}_t \cdot \mathbf{w}_t}$$

where  $\odot$  is element-wise multiplication. Between time  $t-1$  and time  $t$ , the portfolio value is also adjusted from  $p_{t-1} \in \mathbb{R}$  to  $p'_t = p_{t-1} \mathbf{y}_t \cdot \mathbf{w}_{t-1}$ . Let  $p_t$  be the value of the portfolio after transaction costs, and let  $\mu_t \in \mathbb{R}$  be the transaction cost factor, such that  $p_t = \mu_t p'_t$ . We can keep track of the relevant time of each variable with the following diagram:



In this paradigm, the final portfolio value at time  $T$  is

$$p_T = p_0 \prod_{t=1}^T \frac{p_t}{p_{t-1}} = p_0 \prod_{t=1}^T \mu_t \mathbf{y}_t \cdot \mathbf{w}_{t-1}$$

The main difficulty is in determining the factor  $\mu_t$ , since it is an aggregate of all the transaction cost penalties.

Let  $c_s \in [0, 1)$  be the commission rate for selling. We need to sell some amount of asset  $i$  if there is more of asset  $i$  in  $\mathbf{w}'_t$  than in  $\mathbf{w}_t$  by dollar value. Mathematically, this condition is  $p'_t w'_{i,t} > p_t w_{i,t}$ , which is equivalent to  $w'_{i,t} > \mu_t w_{i,t}$ . Thus, the total amount of money raised from selling assets is

$$(1 - c_s) p'_t \sum_{i=1}^m (w'_{i,t} - \mu_t w_{i,t})^+$$

where  $(\cdot)^+ = \max\{0, \cdot\} = \text{ReLU}(\cdot)$ . This money, as well as the money from adjusting the cash reserve from  $p'_t w'_{0,t}$  to  $p_t w_{0,t}$ , is used to purchase assets according to the opposite condition. Let  $c_p \in [0, 1)$  be the commission rate for purchasing. Equating the amount of money available from selling/cash and the amount of money used for purchasing assets yields

$$(1 - c_p) \left[ w'_{0,t} - \mu_t w_{0,t} + (1 - c_s) p'_t \sum_{i=1}^m (w'_{i,t} - \mu_t w_{i,t})^+ \right] = p'_t \sum_{i=1}^m (\mu_t w_{i,t} - w'_{i,t})^+$$

Moving terms around and simplifying the ReLU expressions, we find that  $\mu_t$  is a fixed-point of the function  $f$  defined as:

$$\mu_t = f(\mu_t) = \frac{1}{1 - c_p w_{0,t}} \left[ 1 - c_p w'_{0,t} - (c_s + c_p - c_s c_p) \sum_{i=1}^m (w'_{i,t} - \mu_t w_{i,t})^+ \right]$$

The function  $f$  is a nonlinear. However, for reasonable values of  $c_s$  and  $c_p$ ,  $f$  is both monotone increasing and a contraction, so its unique fixed point can be found by iteratively computing values of  $f$ . This procedure is fairly efficient and easy to implement.

## 2.4 EIIE Policies

The second main contribution of [?] is to create the framework of Ensemble of Identical Independent Evaluators (EIIE) for a policy. The principle is to have a single evaluation function that, given the

price history and other data for a single asset, produces a scores representing potential growth for the immediate future. This same function is applied to all assets independently, and the softmax of the resulting scores become the weights of the portfolio.

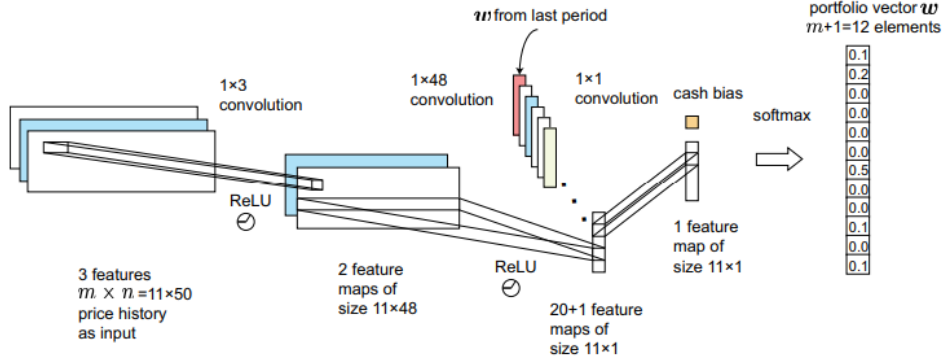
Written mathematically: let  $X_{i,t}$  be the historical price information for asset  $i$  at time  $t$ , and let  $w_{i,t}$  represent the portfolio weights for asset  $i$  at time  $t$ . Let  $\alpha$ ,  $\beta$ , and  $\gamma$  be trainable parameters. First, we define a function  $f_\alpha$  to extract features from each asset's price data  $X_{i,t}$ . This function is applied to each asset individually. The agent also needs to incorporate the previous portfolio weights into its action, in order to deal with the transaction costs as in section ?? . We define a second function  $g_\beta$  that takes the features produced by  $f_{\theta_1}$ , as well as the previous portfolio weights, to produce new weights. Then the portfolio weights for the next time period are

$$w_{t+1} = \text{Softmax}(g_\beta(f_\alpha(X_{1,t}), w_{1,t}), \dots, g_\beta(f_\alpha(X_{m,t}), w_{m,t}), g_\beta(\gamma, w_{m+1,t}))$$

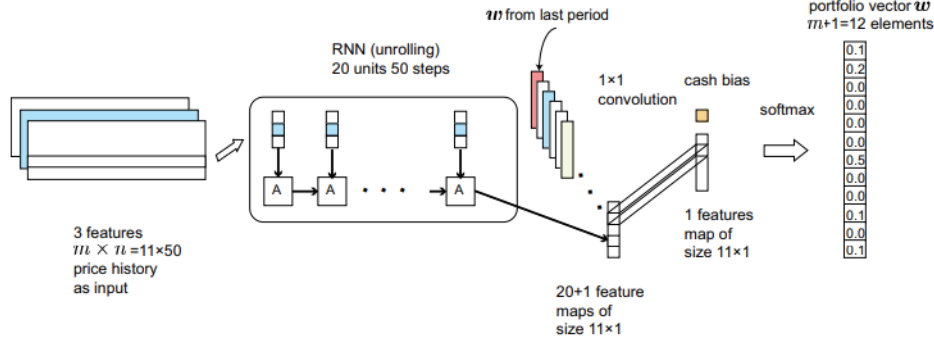
(Note that there are  $m$  tradeable assets in our universe, and that  $w_{m+1,t}$  is the weight for the risk-free asset).

The form of  $g_\beta$  is fairly arbitrary; the authors take it to be an MLP neural network. The form of  $f_\alpha$  is more interesting. The authors of [?] suggest two forms: a CNN and an RNN/LSTM.

For the CNN they provide the following diagram of the EIIE system:



For the RNN they provide a similar diagram:



The three channels in the state matrix on the left are the high, low, and closing price for each asset on each day in the historical sliding window.

The authors claim that this framework beats all of their benchmark strategies in the cryptocurrency market. The architecture remains essentially the same in our implementation. Due to the use of multiple channels in the state tensor, it makes it easy to add additional channels for alternative data sources, such as news sentiment.

## Chapter 3

# Documentation

### 3.1 Gym Environments

Gymnasium is an open-source reinforcement learning environment. Its goal is to standardize the interface between reinforcement learning algorithms and their environments.

Much of the information for this section is taken from their documentation: <https://gymnasium.farama.org/>

The extendable base class for the gym environment is `gym.Env`. A subclass must define the following two methods:

- `reset()` initializes the environment. Its return type is `tuple[ObsType, dict[str, Any]]`. The first element is the initial state, and the dictionary is for initial logging information.
- `step(a: ActType)` performs one update of the environment where the agent takes action `a`. Its return type is `tuple[ObsType, float, bool, bool, dict[str, Any]]`. In order, the elements are: (1) the next state provided to the agent, (2) the reward value for that step, (3) whether the environment has terminated, (4) whether the environment has truncated, and (5) a dictionary of logging information.

`ObsType` and `ActType` are the types of the observation (state) space and action spaces, respectively, which must set within the environment initialization. There are a few standard types of spaces supported by Gymnasium:

- `gym.spaces.Discrete(n)`: A set of `n` distinct discrete values.
- `gym.spaces.Box(low, high, shape, dtype)`: A generalized rectangle. The dimensions are specified by the `shape` parameter, and all entries have `dtype` type (generally `np.float64`). Each component can have its own upper and lower bound if they are tuples (if a single float is provided, that bound is used for all components).
- `gym.spaces.Tuple(spaces)`: A space that is a tuple (cartesian product) of other spaces.

It is possible to define custom spaces, but standard RL training algorithms and models generally only support select compositions of the standard spaces.

### 3.2 RL Framework

In order to ensure consistency and modularity, we have created a framework of classes that allows us to easily swap components of our RL environment to test various data and reward strategies.

### 3.2.1 AbstractRewardManager

This abstract class manages the calculation of rewards for the RL agent. Its interface is minimal. Subclasses can optionally have a constructor.

### 3.2.2 AbstractDataManager

This abstract class manages reading and iterating through the state and price data for the environment.

### 3.2.3 PortfolioEnvWithTCost

This class is the main gym environment. It iterates through the data and rewards using the `AbstractDataManager` and `AbstractRewardManager` provided as arguments. To compute the portfolio updates, it employs the transaction costs-cognizant approach described in section ??.

Let  $w$  be the portfolio weights. We use the convention that  $w[-1]$  is the weight for the risk-free asset, and  $w[: -1]$  are the weights for the risky assets. (Note that this is a different convention than the one used in [? ].)

We omit much of the actual code and instead provide pseudocode or equations where relevant.