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.

Preprint. Under review.

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 data the states matrix of the agent could allow 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 textual corpus sufficient for us to build a robust reinforcement learning agent required us to pull and combine stock price data with SEC filings and news data. The proceeding section gives motivates our usage for each type of data we use and explains our data cleaning process.

1.2.1 Stock Data

We retrieve data from Wharton Research Data Services (WRDS). WRDS is a pre-eminent source of financial data that we have experience utilizing in the Computational Finance program's other courses. Specifically, we want to utilize data from the Center for Research in Security Prices (CRSP), which has security price, return, and volume data for stocks listed in the NYSE, AMEX and NASDAQ exchanges. We created a our trading universe with the S&P 100 stocks. For this dataset, limited cleaning was required and we discuss how it is incorporated into the states matrix in Section 1.3.

1.2.2 News Data

An important direction of our research is to explore how the performance of reinforcement learning trading agents are influenced by news data. We believe that news data will impart an external understanding of how well a given stock is performing. Further, it could inform our agent on a stock's exposure market or sector risks. This could provide our agent a better view of the trading environment, which can help it make better decisions to maximize the reward.

We initially attempted to query data from news APIs to get a more expansive news corpus. However, we found that due to the cost of obtaining the data, the computational power needed to process the headlines, and the storage needed to manage all of the data, that this would not be a feasible pursuit.

The dataset we use in this project is Daily Financial News for 6000+ Stocks that was downloaded via kaggle [**?**]. This dataset contains scraped headline data for over 6000 stocks listed on the NYSE exchange from 2009-2020. There are two main files within this dataset that we use. The first is `raw_analyst_ratings.csv`, which only contains scraped data from a prominent financial news publisher Benzinga. The other file `raw_partner_headlines.csv` contains scraped headline data from other smaller publishers that partner with Benzinga. Each row of the datasets contains a headline, the base article URL, the date and time of publication, and the stock ticker symbol. The `raw_analyst_ratings.csv` file does not contain publisher information, as it is already implied that Benzinga is the publisher. Meanwhile, the table in `raw_partner_headlines.csv` has a column indicating the publisher of each article. We concatenate the headline data from each file to create a single unified dataset that contains all headlines for each stock in our universe.

SEC Filings

To enrich our dataset, we utilize SEC filings data for each of the S&P100 companies. 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 and 10-Q filings. 10-K filings are an annual report on a company's performance, and include information. They are divided into items that show a company's financial statements, stock projections, and pertinent information for shareholders in sections that are denoted as items. 10-Q filings are similar to 10-K filings, with the key differences being less detail given the quarterly frequency, along with unaudited financial statements.

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 approach, which 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

Over the full trading period (2010-2020), we will feed all of the headlines and selected SEC filings text for S&P 100 companies to pre-trained FinBERT. The model then generates probabilities of the content having a positive, negative, or neutral sentiment. For the news headlines, we developed a novel function to extract a single embedding for a stock on a given day.

The function that we created is:

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

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. Thus, a sentiment score close to 1 can be interpreted as a positive sentiment, a score close to 0 can be interpreted as neutral, and a score close to -1 can be intepreted as negative.

An issue that we run into when incorporating SEC filings data is that they are recorded on a annual or quarterly basis, which is far more infrequent than our samples for price and news data. Thus, to fill the gaps between SEC filings dates, we apply exponential decay to the sentiment scores on report dates. Formally,

$$y = a(1-\gamma)^t$$

Where a represents the company's sentiment score on the reporting date, t represents time (in days) between the last report date and the current day, and γ is a constant between 0 and 1 representing the daily decay factor. We test $\lambda = 0.1, 0.5, 0.9$ in our models to see what performs optimally. Note that we test a wide range of γ values because we are unsure of how fast the signal from SEC filings decay. We also run experimented with using this same routine to fill some gaps between news data.

1.2.4 Finalized Data Processing Pipeline

We dedicate this section of the paper to outline the pre-processing pipeline for both our SEC filings and news data.

Creating News Tensors

From the concatenated dataset of news headline data from each publisher as described in the "News Data" section, we feed the dataset (loaded into a pandas dataframe) through a multi-stage pipeline. The first step is to scrape the current $S\&P$ 100 companies and then filter the dataset down to only include headlines from companies in the $S\&P$ 100. We introduce a custom dataset class called "NewsHeadlines," implemented in PyTorch framework, designed for efficiently handling news headline data. The class takes a dataset and a user-defined tokenizer which will pre-process headlines in batches to be fed into FinBERT. In the class, we implement an iterator function `_getitem`, which takes the raw headline data as input and returns an encoding for the batch of headlines after tokenization. Then given the large size of the dataset, we use a create a "Dataloader" object, implemented in PyTorch, which feeds our dataset into the model in small batches.

To obtain the output tensors corresponding to the sentiment probabilities, we iterate over the batches, applying FinBERT to classify each headline and from the raw logits using the softmax activation function to a vector of probabilities. Then for each batch, we save off the tensors to separate files. A finalized version of this process is available in `utilities`
`news_data_processing.py` Then in the `tensor_data_loading.ipynb` script, we merge the tensors with the original concatenated dataset and create our the sentiment embedding as described in the "FinBERT Sentiment Scores" section. This finalized dataset is then written to `news_sentiment_data.csv`

News Dataset Statistics

Our dataset contains data for 84 out of the total 100 tickers in the $S\&P$ 100, and it contains 70,872 entries containing the sentiment embedding of news for a company on a given day. Below are some reported summary statistics on the distribution of news reports across the tickers:

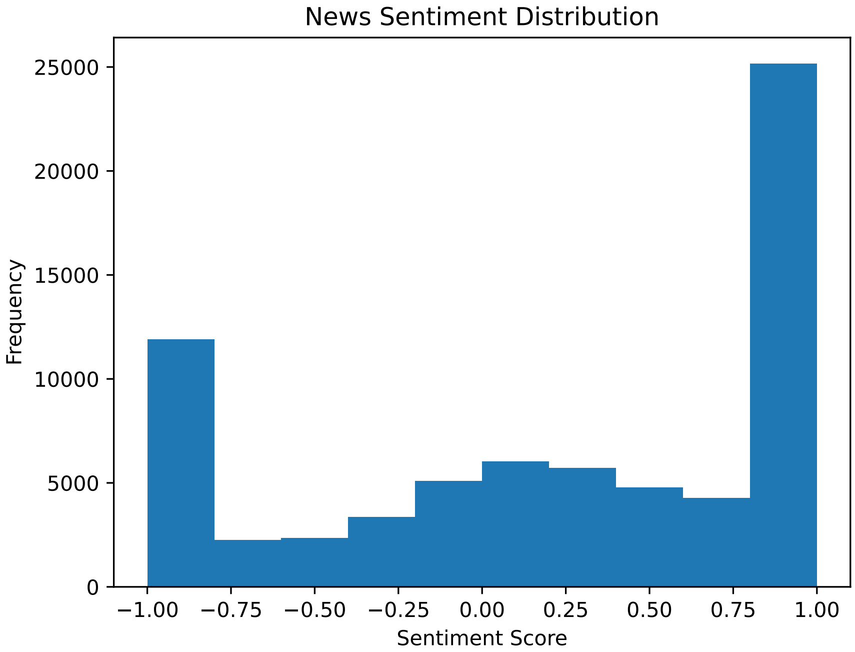
Table 1.1: Company News Reporting Date Distribution

Statistic	Value
Count	84
Mean No. of Reporting Dates	843.714
Standard Deviation	508.209
Minimum Observations	1
25th Percentile	393.250
Median	905.000
75th Percentile	1198.500
Maximum	1829

Note that given our median ticker only has news reports on 905 of the total trading dates and because there are 16 tickers for which we have no sentiment data, our dataset is still suboptimal for developing an agent. Our forward filling process, does address some of the gaps in our data, however our coverage is still not complete. This is an important consideration when examining the results of our work.

Now, we will examine the distribution of sentiment scores across the articles to examine key facts about their distribution.

The figure highlights that news sentiment has a bimodal distribution. Much of the headlines are interpreted as either negative or positive, but news headlines that are relatively neutral, or closer to 0,



or more evenly distriubuted. This indicates that the headlines that display strong enough sentiment that they could inform and change the actions of our reinforcement learning agents.

Creating SEC Tensors

To create tensors, we initially attempted to scrape the SEC's EDGAR database for all 10-K and 10-Q filings in the database's history, ranging from 1994 through 2023. The filings from the EDGAR database are returned as HTML files which we stored in a cloud database. The size of this database is roughly 108.3 GiB, or 116.3 GB.

To extract meaningful values from the text, we first parse and clean the HTML so we can extract the raw text from each document. Then we use regular-expression based text parsing to extract text from Item 1A and 7/7A, and Item 2 in 10-Qs. We then construct a data frame, where each row contains the company ticker, the date of the filing, the extracted section name, the text of the extracted section. We then pass this into the FinBERT model to add a final column containing sentiment scores, calculated by the methodology explained in Section **??**. However, issues were encountered both with the size of the dataset being computationally intensive, along with parsing issues due to the way the filings are stored and transmitted. The process to create SEC-sourced tensors is then to extract positive, negative, and neutral words as specified by the Loughran-McDonald sentiment dictionary. We then utilize the proportions in a similar fashion to the news embeddings:

$$\texttt{Value}_{\texttt{Embedding}} = \tanh\left(\frac{\frac{\texttt{positive word count proportion}}{\texttt{negative word count proportion}}}{\texttt{neutral word count proportion}}\right)$$

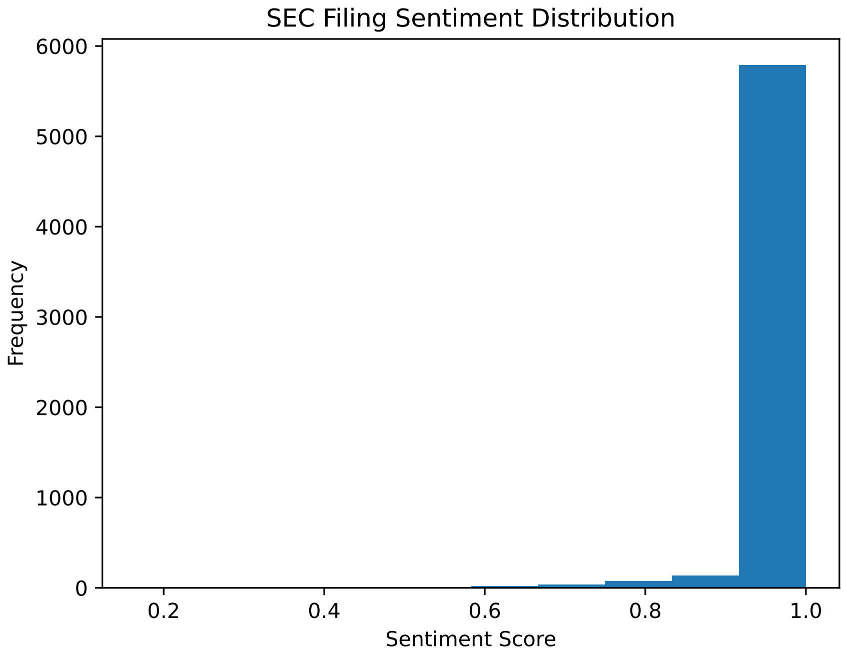
SEC Filings Dataset Statistics

Our dataset contains data for 99 out of the 100 tickers in the $S\&P$ 100, containing over 9,000 filings between 1994 and the present day, with the used subset consists of roughly 6,100 filings. Below are some reported summary statistics on the distribution of SEC filings across the tickers:

Since there are only 4 filings per year, we use forward filling with decay to fill in the "missing" dates. Giving the addition and dropping of companies from the SP100, as well as some newer public companies joining, each company does not have the same number of filings over the time period. We now look at the distribution of sentiment scores as calculated in Section **??**:

Table 1.2: Company SEC Filings Distribution

Statistic	Value
Count	99
Mean No. of Filings	61.42
Standard Deviation	10.18
Minimum Observations	20
25th Percentile	64
Median	65
75th Percentile	65
Maximum	66



There is a pronounced tail towards 1, indicating a strongly positive unimodal distribution, as compared to that of the news sentiment. Since we utilize sections of the SEC filings that are written by the companies themselves, it is likely that companies aim to provide filings that suggest strong performance and future outlook. In the dataset, we do observe some drops in sentiment, such as times of financial crisis or bad market conditions, like in 2013 for some technology-based companies.

1.3 Algorithmic and Analytical Challenge

Our primary model technique is deep reinforcement learning, which is a branch of machine learning that operates in a game-theoretic-like system. 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:

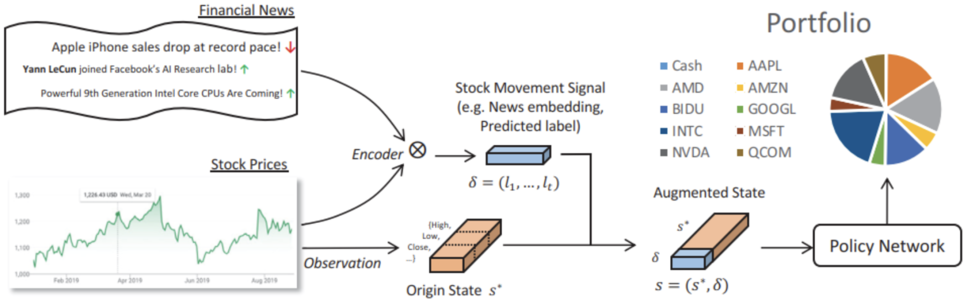
- [?] Deep Reinforcement Learning Comparison with Mean-Variance Optimization: Using a lookback of recent past returns and a few market indicators (including 20-day volatility and the VIX), this paper implements a simple algorithm for portfolio weight selection to maximize the Differential Sharpe Ratio, 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 δ, 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 δ, 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 [**?**], as well as the Differential Sharpe Ratio developed in paper [**?**].

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.

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).
- $T : S \times A \times S \to [0, 1]$ 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.)
- $R : S \times A \times S \to \mathbb{R}$ is the reward function. When the environment changes state from s to s' due to action a, the agent recieves 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 $\pi : S \to A$ that dictates actions to take at a given state. A solution to an MDP problem is an optimal policy π^* 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 $U([r_0, r_1, \ldots]) = \sum_{t=0}^{\infty} \gamma^t r_t \le \frac{\sup_t r_t}{1-\gamma}$. 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 desireable; we include it because it is common in RL literature.)

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

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

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

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

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:

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

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 $A = \left\{a \in \mathbb{R}^{n+1} \mid \sum_i a_i = 1\right\}$ 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 udpates.)

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' 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

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

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:

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

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_η 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 η. 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 η; expanding the Sharpe ratio estimator formula in Taylor series gives

$$S_t \approx S_{t-1} + \eta \frac{\mathrm{d}S_t}{\mathrm{d}\eta}\bigg|_{\eta=0} + o(\eta^2)$$

If η 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

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

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

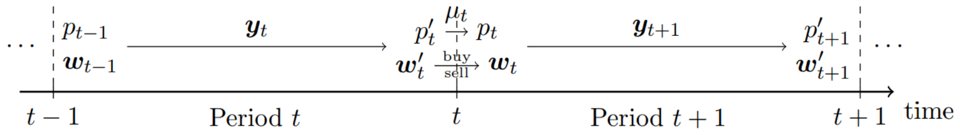
[**?**] contains an excellent walkthrough of the mathematics for modeling transaction costs in RL environment updates. We provide a shortened version here.

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 portfoliio 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 μ_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 $p'_t w'_{i,t} > p_t w_{t,i}$, 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 $(\cdot)^+ = \max\{0, \cdot\} = \mathrm{ReLU}(\cdot)$. 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 commision 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 μ_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.

Chapter 3

Documentation

3.1 Gym Environments

WIP

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.

Chapter 4

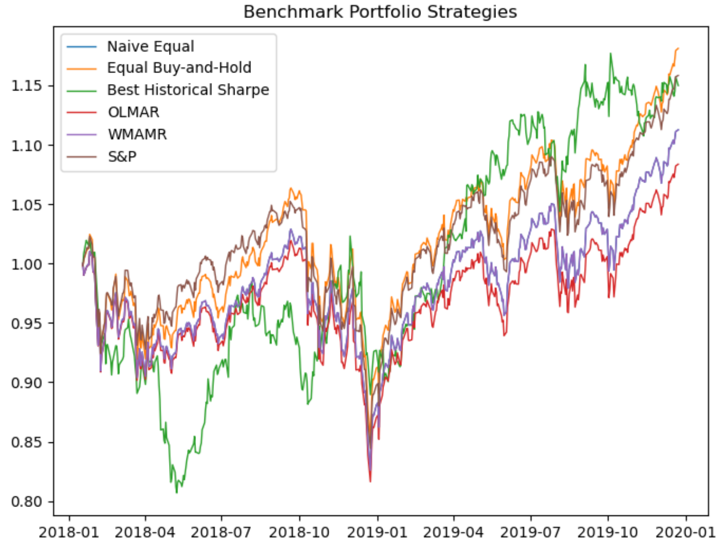
Results

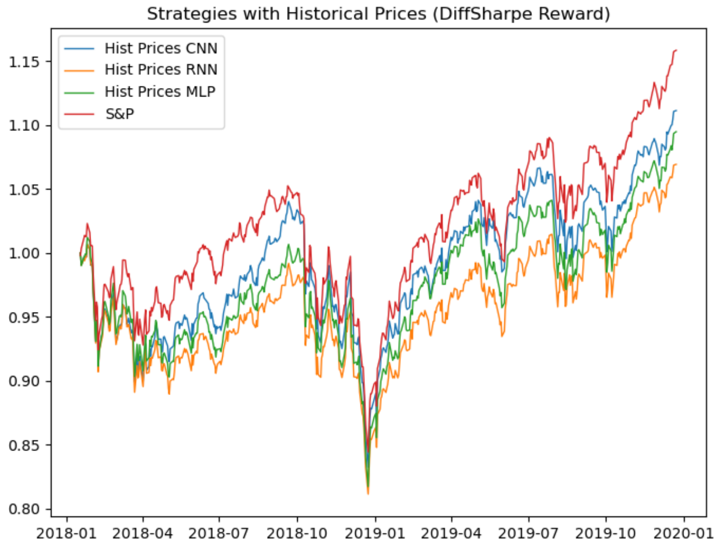
4.1 Benchmarks Portfolios

To inform our discussion of our agents' trading performance, we want to compare each of our benchmark portfolios against the S&P 100 index performance.

Table 4.1: Benchmark Portfolio Strategies

	Net Profit	Sharpe Ratio	Sortino Ratio	Max Drawdown
Naive Equal	0.112583	0.392352	-14.464945	0.197294
Equal Buy-and-Hold	0.181138	0.595225	-14.009800	0.198039
Best Historical Sharpe	0.149857	0.475577	-16.749871	0.210250
OLMAR	0.083634	0.300906	-14.344610	0.199240
WMAMR	0.112565	0.392310	-14.464168	0.197294
S&P	0.158347	0.552242	-14.756129	0.197728





As we can from figures **??** and **??** most of our benchmarks underperform the S&P index over the trading period in terms of profitability. Further, the only benchmark that does outperform the S&P (Naive Equal) has a significantly lower sharpe ratio (0.39) than the S&P (0.55) indicating that after adjusting for portfolio deviation, the S&P outperforms it. Further, the maximum drawdown and sortino ratios for most of our benchmarks compared to the S&P are fairly similar indicating that they have similar downside risks.

4.2 RL Portfolios

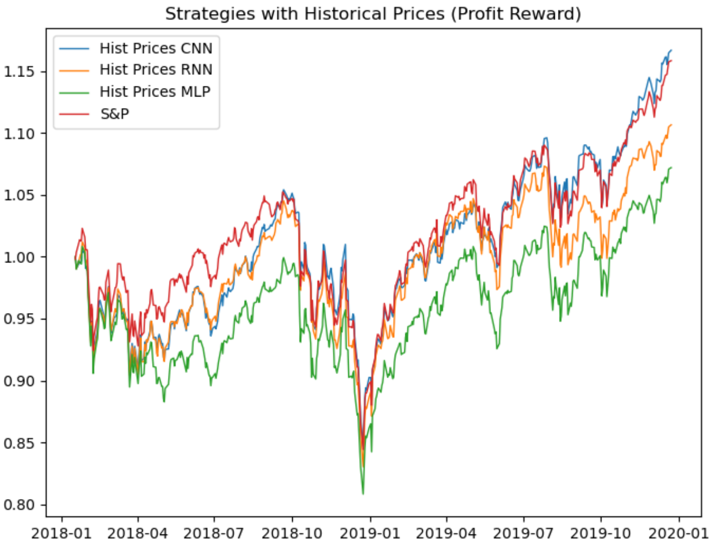
In this section, we describe the returns of our reinforcement learning agent portfolios in an environment including only historical price and returns data. We CNN, RNN, and MLP based policy networks and employ DDPG to optimize the policy. We also test across each both of our rewards, differential sharpe ratio and profit.

4.2.1 Differential Sharpe Reward

Table 4.2: Strategies with Historical Prices (Differential Sharpe Reward)

	Net Profit	Sharpe Ratio	Sortino Ratio	Max Drawdown
Hist Prices CNN	0.111205	0.396336	-14.459686	0.199721
Hist Prices RNN	0.069154	0.262906	-14.991369	0.194177
Hist Prices MLP	0.094686	0.338142	-14.739489	0.188082
S&P	0.158347	0.552242	-14.756129	0.197728

From **??** it is evident that our strategies are underperforming the S&P index over the trading period. The sharpe ratio of each of the reinforcement learning portfiolios is markedly lower than that of the S&P. This indicates that our policy gradient optimization process is not effectively learning the differential sharpe ratio reward effectively ove It is also of note that the CNN appears to perform better than the MLP and RNN policy networks with higher profit and sharpe ratio. Figure **??** also reinforces these takeaways.



4.2.2 Profit Reward

In contrast, we see more encouraging results when we change to using the the profit reward. We see from **??**, that with the CNN, we actially outperform the S&P by 1 percent over the trading period with a higher sharpe ratio and lower maximum drawdown.

We hypothesize that this is because the profit reward is easier to learn. However, we will investigate if these findings hold up when integrating our alternative data.

Table 4.3: Strategies with Historical Prices (Profit Reward)

	Net Profit	Sharpe Ratio	Sortino Ratio	Max Drawdown
Hist Prices CNN	0.166717	0.561471	-14.923095	0.196557
Hist Prices RNN	0.106518	0.369764	-15.113139	0.205465
Hist Prices MLP	0.071937	0.261355	-15.265520	0.198329
S&P	0.158347	0.552242	-14.756129	0.197728

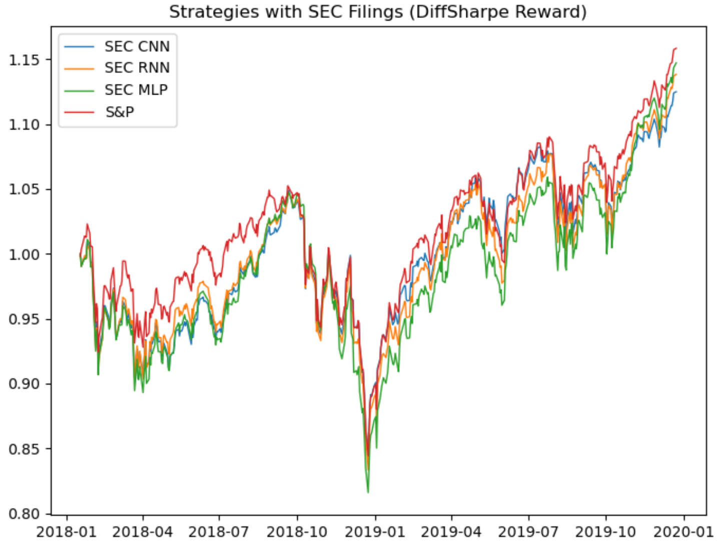
4.3 RL Portfolio With Alternative Data

4.3.1 News Data

4.3.2 SEC Data

Differential Sharpe Reward

Profit Reward



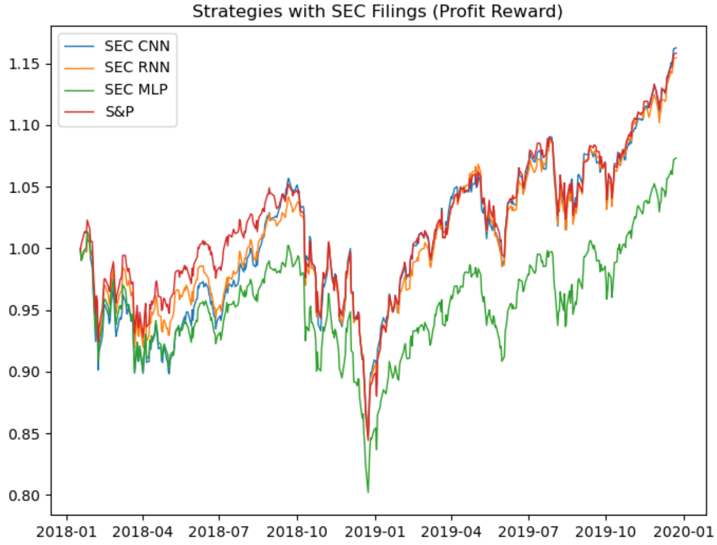


Figure 4.1: Strategies with SEC Filings (Profit Reward)

Table 4.4: Strategies with SEC Filings (Differential Sharpe Reward)

	Net Profit	Sharpe Ratio	Sortino Ratio	Max Drawdown
SEC CNN	0.124870	0.439863	-14.397950	0.195347
SEC RNN	0.138309	0.464039	-14.439768	0.204159
SEC MLP	0.147103	0.482788	-14.519536	0.220906
S&P	0.158347	0.552242	-14.756129	0.197728

Table 4.5: Strategies with SEC Filings (Profit Reward)

	Net Profit	Sharpe Ratio	Sortino Ratio	Max Drawdown
SEC CNN	0.162534	0.532298	-14.478839	0.195766
SEC RNN	0.154506	0.513757	-14.728491	0.187671
SEC MLP	0.073242	0.266018	-15.909619	0.208675
S&P	0.158347	0.552242	-14.756129	0.197728

4.3.3 SEC and News Data Combined

Differential Sharpe Reward

Table 4.6: Strategies with Combined Data (DiffSharpe Reward)

	Net Profit	Sharpe Ratio	Sortino Ratio	Max Drawdown
SEC+News CNN	0.094546	0.342485	-14.805659	0.193996
SEC+News RNN	0.128636	0.444354	-14.533568	0.195274
SEC+News MLP	0.053514	0.205204	-15.149644	0.200069
S&P	0.158347	0.552242	-14.756129	0.197728

Profit Reward

4.4 Summary of Results

Our principal observations indicate a disparity in learning complexity between the profit reward function and the differential Sharpe ratio for our agent, resulting in consistently superior portfolio performance when optimizing with the former. Notably, policy networks based on CNNs and RNNs exhibit enhanced performance under the profit reward, while the MLP policy network seems to overfit to S&P returns.

Furthermore, our analysis underscores the challenge of integrating news data, revealing a diminished capacity of the model to learn the differential Sharpe ratio reward. We attribute this difficulty to the sparse and inconsistent nature of the dataset. However, it is noteworthy that the infusion of news data appears to minimally impact performance concerning the profit reward function, which again highlights the profit reward's relative simplicity.

Upon integrating SEC filings data, notable performance enhancements are observed across both reward functions compared to baseline models. Despite the sparsity of SEC filings, their regularity and consistency across all tickers facilitate improved learning for our policy algorithms, thus contributing to heightened performance.

Perhaps most intriguingly, optimal performance is achieved through hyperparameter tuning when integrating both news and SEC data into the states matrix. This outcome underscores the untapped potential of research avenues exploring comprehensive datasets capturing nuanced company sentiment.

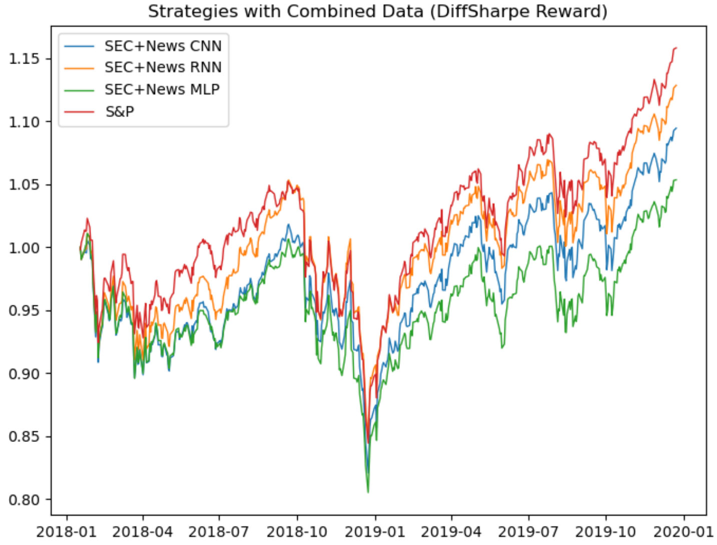
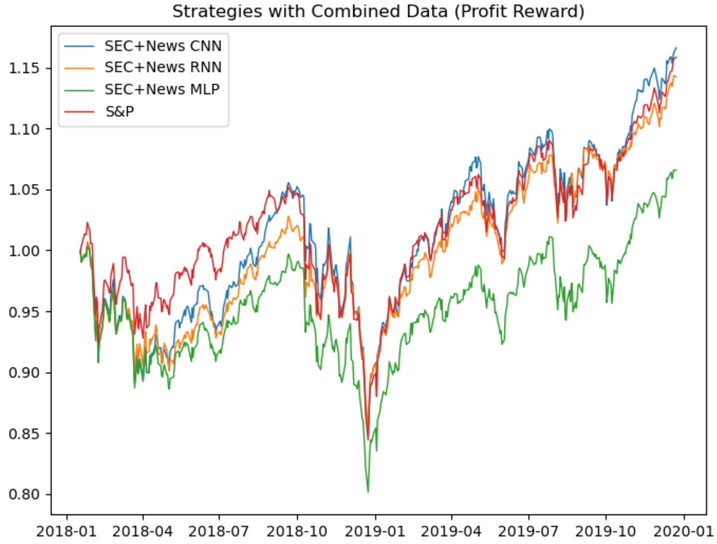


Figure 4.2: Strategies with Combined Data (DiffSharpe Reward)

Table 4.7: Strategies with Combined Data (DiffSharpe Reward)

	Net Profit	Sharpe Ratio	Sortino Ratio	Max Drawdown
SEC+News CNN	0.166081	0.553644	-14.319155	0.195472
SEC+News RNN	0.142287	0.519258	-14.759413	0.170134
SEC+News MLP	0.065676	0.248586	-14.917642	0.195750
S&P	0.158347	0.552242	-14.756129	0.197728



Bibliography

[] Sandro Gössi, Ziwei Chen, Wonseong Kim, Bernhard Bermeitinger, and Siegfried Handschuh. FinBERT-FOMC: Fine-Tuned FinBERT Model with Sentiment Focus Method for Enhancing Sentiment Analysis of FOMC Minutes. In *Proceedings of the Fourth ACM International Conference on AI in Finance*, ICAIF '23, pages 357–364, New York, NY, USA, November 2023. Association for Computing Machinery. ISBN 9798400702402. doi: 10.1145/3604237.3626843. URL `https://dl.acm.org/doi/10.1145/3604237.3626843`.

[] Srijan Sood, Kassiani Papasotiriou, Marius Vaiciulis, and Tucker Balch. Deep Reinforcement Learning for Optimal Portfolio Allocation: A Comparative Study with Mean-Variance Optimization.

[] Junkyu Jang and NohYoon Seong. Deep reinforcement learning for stock portfolio optimization by connecting with modern portfolio theory. *Expert Systems with Applications*, 218: 119556, May 2023. ISSN 0957-4174. doi: 10.1016/j.eswa.2023.119556. URL `https://www.sciencedirect.com/science/article/pii/S095741742300057X`.

[] Yunan Ye, Hengzhi Pei, Boxin Wang, Pin-Yu Chen, Yada Zhu, Jun Xiao, and Bo Li. Reinforcement-Learning based Portfolio Management with Augmented Asset Movement Prediction States, February 2020. URL `http://arxiv.org/abs/2002.05780`. arXiv:2002.05780 [cs, q-fin, stat].

[] Zhengyao Jiang, Dixing Xu, and Jinjun Liang. A Deep Reinforcement Learning Framework for the Financial Portfolio Management Problem, July 2017. URL `http://arxiv.org/abs/1706.10059`. arXiv:1706.10059 [cs, q-fin] version: 2.

[] Qiang Song, Anqi Liu, and Steve Y. Yang. Stock portfolio selection using learning-to-rank algorithms with news sentiment. *Neurocomputing*, 264:20–28, November 2017. ISSN 0925-2312. doi: 10.1016/j.neucom.2017.02.097. URL `https://www.sciencedirect.com/science/article/pii/S0925231217311098`.

[] Jinho Lee, Raehyun Kim, Seok-Won Yi, and Jaewoo Kang. MAPS: Multi-agent Reinforcement Learning-based Portfolio Management System. In *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence*, pages 4520–4526, July 2020. doi: 10.24963/ijcai.2020/623. URL `http://arxiv.org/abs/2007.05402`. arXiv:2007.05402 [cs].

[] J. Moody and Lizhong Wu. Optimization of trading systems and portfolios. In *Proceedings of the IEEE/IAFE 1997 Computational Intelligence for Financial Engineering (CIFEr)*, pages 300–307. doi: 10.1109/CIFER.1997.618952. URL `https://ieeexplore.ieee.org/document/618952`.

[] John Moody, Matthew Saffell, Yuansong Liao, and Lizhong Wu. Reinforcement learning for trading systems and portfolios: Immediate vs future rewards. In Apostolos-Paul N. Refenes, Andrew N. Burgess, and John E. Moody, editors, *Decision Technologies for Computational Finance: Proceedings of the fifth International Conference Computational Finance*, pages 129–140. Springer US. ISBN 978-1-4615-5625-1. doi: 10.1007/978-1-4615-5625-1_10. URL `https://doi.org/10.1007/978-1-4615-5625-1_10`.

[] Miguel, Aenille Daily Financial News for 6000+ Stocks URL `https://www.kaggle.com/datasets/miguelaenlle/massive-stock-news-analysis-db-for-nlpbacktests/data`.