Multimodal Deep Reinforcement Learning for Portfolio Optimization

Sumit Nawathe **Ravi Panguluri** **James Zhang** **Sashwat Venkatesh**
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 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 Algorithmic and Analytical Challenge

1.2.1 Reinforcement Learning

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, 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). (See Section 2.1 for further explanation.) The goal is to find a trading 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.

1.2.2 Existing Literature

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

- [1] 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.

- [2] 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.
- [3] 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.
- [4] 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.
- [5] 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.
- [6] 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 Alternative Dataset Summary

All of the reinforcement learning agents that we create will have access to historical company price data, as it broadly reflects the market's perceived value of a given company. However, we believe that using alternative sources will enhance our agents' decision-making process and provide value to our portfolio strategy. We aim to use two primary types of alternative data: news headlines and SEC filings. We discuss our data sourcing, curation and cleaning process at length in Chapter 3. However, here we briefly motivate our usage of each in composing our multimodal dataset.

1.3.1 SEC Filings Data

SEC filings include detailed information on a company's financial health and external risk factors directly from executives [7]. SEC filings are filed under a single standard format by all publicly listed companies on a quarterly and yearly basis. Given the imposed structure of the documents and regular reporting periods, these filings provide a consistent source of external information. Further, we believe that these filings could provide valuable future-looking insight into a company's operations that might not be directly immediately reflected in its stock price. The parts of SEC reports that we use are discussed in Section 3.1.3. Section 3.3 discusses how use the Loughran-McDonald sentiment dictionary to compute sentiment scores for each company on the date of filing release, our use exponential day when forward-filling these scores to future dates, and the quality of the data.

1.3.2 News Headline Data

We incorporate company-specific news headlines in our agents' environment because they can reflect real-time shifts in investor perceptions that may take longer to be reflected in a company's price. Positive news such as acquisitions can drive stock prices up, while negative news such as leadership

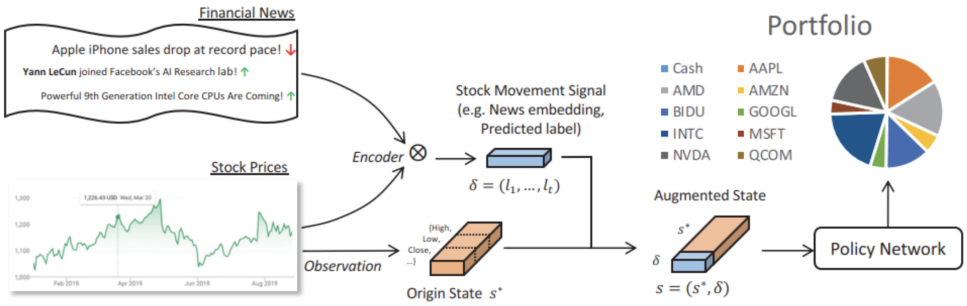
changes can have adverse effects. Therefore, having up-to-date sentiment information on each company in the trading universe could help our agent outperform its benchmarks. Acquisition of our news data is discussed in Section 3.1.2. Section 3.2 discussed how we obtain FinBERT sentiment scores, our novel function for creating sentiment embeddings, our process for forward-filling sentiment data using exponential decay, and the quality of the news data.

1.4 Methodology

We will be implementing, combining, and improving on the methodologies of several of the above papers. Our plan is to develop an reinforcement learning system that utilizes multiple time periods to achieve strong out-of-sample trading performance. Our final architecture is most similar to papers [3] and [4].

1.4.1 Markov Decision Process Problem Formulation

Paper [3] 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 as an additional channel to S^* to create the full state tensor $S = (S^*, \delta)$.

In our architecture, the signal information δ will be composed of process SEC and sentiment news indicators. The structure of the state S will remain a 3D tensor, as further described in Sections 2.4 and 4.1.2. Each row of S will represent a different stock in our universe, and along that row will be all of the price and alternative data for the past several days. (As discussed in the literature, the straightforward concatenation of price data and news embeddings does 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 [3], as well as the Differential Sharpe Ratio developed in paper [1]. 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. More information can be found in Section 4.1.1.

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 [3], 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.

1.4.2 Use of Libraries

We mainly use 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. We experiment with various combinations of state spaces, rewards, and policy types, as well as the incorporation of alternative data not used in the reference papers.

1.4.3 Strategy Benchmarking

Our final model architecture is compared against several benchmark financial portfolio selection models. Among these will be a naive equally weighted portfolio, a naive buy-and-hold portfolio, and holding the asset with the best historical sharpe. These simple strategies are defined in Section 5.1. In addition, we test our agents against two more advanced benchmark strategies: OLMAR and WMAMR, which are defined in Sections 2.5 and 2.6 respectively.

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 of the papers we discuss in Section 1.2.2 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 $U([r_0, r_1, \ldots]) = \sum_{t=0}^{\infty} \gamma^t r_t \leq \frac{\sup_t r_t}{1-\gamma}$. 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 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 m tradeable stocks in our universe, and one risk-free asset. The action space $A = \left\{a \in \mathbb{R}^{m+1} \mid \sum_i a_i = 1\right\}$ 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 2.2), which urges the agent to make portfolio allocations to maximize its total Sharpe ratio.

2.2 Differential Sharpe Ratio

[1] 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 [8] and [9]. 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:

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

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_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

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

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

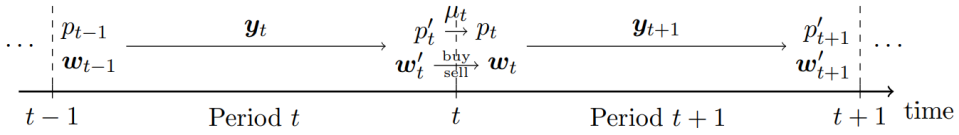
[4] contains an excellent walkthrough of the mathematics for modeling transaction costs in RL environment updates. We provide a shortened version here.

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 $(\cdot)^+ = \max\{0, \cdot\} = \text{ReLU}(\cdot)$. 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 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.

2.4 EIIE Policies

The second main contribution of [4] 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.

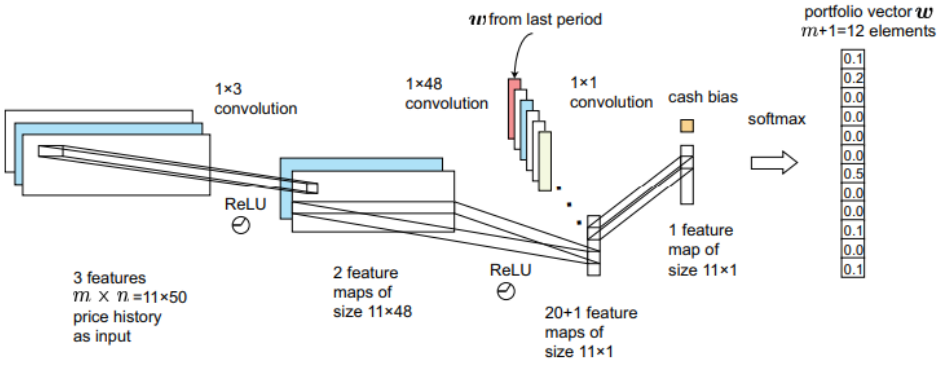
Written mathematially: let $X_{i,t}$ be the historical price information for asset i at time t, and let $w_{i,t}$ represent the portfolio weights for asset i at time t. Let α, β, and γ be trainable parameters. First, we define a function f_α to extract features from each asset's price data $X_{\times,t}$. 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 2.3. We define a second function g_β that takes the features produced by f_{θ_1}, as well as the previous portfolio weights, to produce new weights. Then the portfolio weights for the next time period are

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

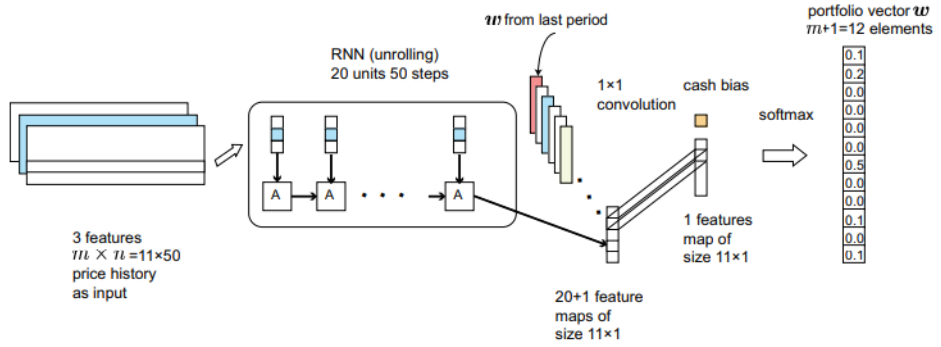
(Note that there are m tradeable assets in our universe, and that $w_{m+1,t}$ is the weight for the risk-free asset).

The form of g_β is fairly arbitrary; the authors take it to be an MLP neural network. The form of f_α is more interesting. The authors of [4] 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.

2.5 OLMAR Benchmark Strategy

Online Portoflio Selection with Moving Average Reversion (OLMAR), introduced in [10], is used by both [3] and [4] as a benchmark strating strategy. Our codebase includes both custom and library implementations of OLMAR; we provide a brief description of the method here.

Consider a universe of m tradeable assets (and no risk-free asset). Let

$$\Delta_m = \left\{ b_t \in \mathbb{R}^m \mid b_{t,i} \geq 0 \, \forall i, \sum_{i=1}^{m} b_{t,i} = 1 \right\}$$

be the space of possible portfolios, which is a simplex. Let $p_t \in \mathbb{R}^m$ be the price at time t, and and let $x_t \in \mathbb{R}^m$ be the price-relative vector for time t, computed as $x_{t,i} = p_{t,i}/p_{t-1,i}$ for all i; that is, x_t is the element-wise division of p_t by p_{t-1}. The goal of the algorithm is to produce a good portfolio b_{t+1} given $p_t, p_{t-1}, ...p_{t-w+1}$ (that is, historical stock prices over a lookback of period w).

If we believe that the price of an asset is mean-reverting, then a good prediction for the next price-relative vector $\tilde{x}_{t+1}$ is

$$\tilde{x}_{t+1} = \frac{\mathrm{MA}_t(x)}{p_t} = \frac{1}{w}\left(\frac{p_t}{p_t} + \frac{p_{t-1}}{p_t} + \cdots + \frac{p_{t-w+1}}{p_t}\right)$$

To obtain a good return over the next time period, we want $b_{t+1} \cdot \tilde{x}_{t+1}$ to be high. However, to keep transaction costs down, we do not want to be too far away from the previous portfolio b_t. Therefore, we formula the optimization problem:

$$b_{t+1} = \arg\min_{b \in \Delta_m} \frac{1}{2}\|b - b_t\| \text{ such that } b \cdot \tilde{x}_{t+1} \geq \epsilon$$

for some positive threshold value ϵ. This optimization problem can be solved numerically using standard constrained solvers.

Alternatively, the authors present the following solution: if we ignore the non-negativity constraint, then the solution is

$$b_{t+1} = b_t + \lambda_{t+1}(\tilde{x}_{t+1} - \bar{x}_{t+1}\mathbb{1}) \quad \bar{x}_{t+1} = \frac{\mathbb{1} \cdot \tilde{x}_{t+1}}{m} \quad \lambda_{t+1} = \max\left\{0, \frac{\epsilon - b_t \cdot \tilde{x}_{t+1}}{\|\tilde{x}_{t+1} - \bar{x}_{t+1}\mathbb{1}\|^2}\right\}$$

To enforce the non-negativity constraint, we can project this solution back into the simplex Δ_m.

2.6 WMAMR Benchmark Strategy

Weighted Moving Average Mean Reversion, introduced in [11], is another trading strategy benchmark used by [3]. It is a relatively simple modification to OLMAR, so we will not restate the content of Section 2.5.

The definitions of all terms remain the same. Define the ϵ-insensitive loss function

$$l_{1,\epsilon}(b, \tilde{x}_{t+1}) = \begin{cases} 0 & b \cdot \tilde{x}_{t+1} \leq \epsilon \\ b \cdot \tilde{x}_{t+1} - \epsilon & \text{otherwise} \end{cases}$$

The authors formula the optimization problem as

$$b_{t+1} = \arg\min_{b \in \Delta_m} \frac{1}{2}\|b - b_t\|^2 \text{ such that } l_{1,\epsilon}(b, \tilde{x}_{t+1}) = 0$$

As with OLMAR, this optimization problem can be solved numerically. If we ignore the non-negativity constraint, then an analytic solution is

$$b_{t+1} = b_t - \tau_t(\tilde{x}_{t+1} - \bar{x}_{t+1} \cdot \mathbb{1}) \qquad \tau_t = \max\left\{0, \frac{l_{1,\epsilon}}{\|\tilde{x}_{t+1} - \bar{x}_{t+1}\|^2}\right\}$$

To recover the non-negativity constraint, we project this solution back into the simplex Δ_m.

Chapter 3

Data Collection and Processing

We outline the process of collecting, storing, and preprocessing all of the price data and alternative textual data used in our trading strategies. Implementations for each part can be found on our project GitHub repository.

3.1 Data Sourcing

3.1.1 Stock Price Data

The BUFN Computational Finance Minor program provided us with access to Wharton Research Data Services (WRDS), an in particular, data from the Center for Research in Security Prices (CRSP). Pursuant to the ideas and implementation of [4], we downloaded basic stock price data (close/high/low price, volume, and metadata) for all stocks in the S&P100 index from 2010 to 2020. We also downloaded data for the S&P500 value-weighted and equally-weighted indices for benchmark comparison [12].

3.1.2 News Data

Scraping Attempts

We attempted to scrape a wider corpus of headline data from financial news providers including Yfinance and Bloomberg. However, we encountered numerous issues including rate limiting and IP blocking when we scraped at too high of a frequency. We also explored using APIs that provide real-time and historical news data pipelines for business and professional use including Event Registry, newsapi.ai, and Alpha Vantage. However, all of them require a significant payment to get data at a velocity that we would need, and historical data is often even more expensive. While some have a free tier that allows for limited data collection, it is not sufficient in quantity or scope for our purposes.

Daily Financial Headlines Dataset

The dataset we use in this project is Daily Financial News for 6000+ Stocks that was downloaded via Kaggle [13]. 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 publisher, the date and time of publication, and the stock ticker symbol.

We concatenate the headline data from each file to create a single unified dataset that contains all available news headlines in our trading period for all S&P 100 stocks.

3.1.3 SEC Data

We used the EDGAR database [14] to downloaded 10-K and 10-Q SEC filings for S&P100 for the last 30 years. The results are a set of HTML files taking up roughly 115GB of storage space, which we stored in Google drive. We built parsers to extract the key sections from both types of filings; in particular, Item 7/7A from the 10-K and Item 2 from the 10-Q. This is the Management's Discussion and Analysis (MD&A) section, which allows the company management to discuss "the company's exposure to market risk, such as interest rate risk, foreign currency exchange risk, commodity price risk or equity price risk,", and "how it manages its market risk exposures" [7].

3.2 News Data Processing

3.2.1 FinBERT Sentiment Scores

Over the full trading period (2010-2020), headlines for S&P 100 companies are fed into 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:

$$\texttt{Value}_{\texttt{Embedding}} = \tanh\left(\frac{\frac{\texttt{positive sentiment probability}}{\texttt{negative sentiment probability}}}{\texttt{neutral sentiment probability}}\right) \tag{3.1}$$

This approach captures the sentiment polarity by measuring the ratio between positive and negative sentiment in the numerator term. Dividing this ratio by the neutral sentiment probability imposes a penalty in the case that a headline is likely neutral. In that case, even if the the ratio between negative and positive sentiment probabilities are high, we lose the information that the sentiment of the headline is likely neutral. Finally, our approach uses the tanh for normalization changing the domain of sentiment scores to be between -1 and 1. 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 interpreted as negative.

An issue that we run into with news data is irregular reporting dates and significant gaps in data reporting, which is described in more detail section 3.2.3. To address some of the gaps in news data reporting, we apply exponential decay to the sentiment scores on report dates. Formally,

$$y = a(1-\gamma)^t \tag{3.2}$$

where a represents the company's sentiment score on the most recent 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. In our training process, we tune γ as a hyperparameter to see what rate of decay yields the best performing agents; we found that $\gamma \approx 0.8$ worked well for us.

3.2.2 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`.

3.2.3 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. Table 3.1 displays some summary statistics on the distribution of news reports across the tickers.

Table 3.1: 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 sub-optimal 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.

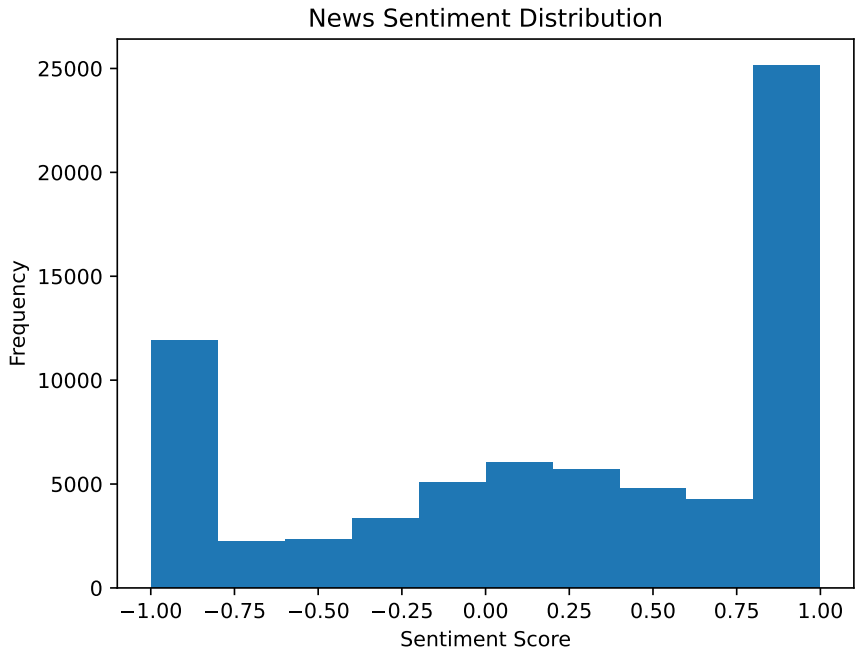


Figure 3.2: Frequency distribution of our novel news sentiment scores.

Figure 3.2 shows the distribution of sentiment scores across the articles. 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 distributed. This indicates that the headlines that display strong enough sentiment that they could inform and change the actions of our reinforcement learning agents.

3.3 SEC Data Processing

Creating SEC Tensors

To extract meaningful values from the text, we first parse and clean the SEC filing HTML documents so we can extract the raw text. 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 attempted to replicate the FinBERT sentiment score procedure explained in Section 3.2.1 for SEC filings. However, issues were encountered both with the size of the dataset making applying FinBERT to these extracted sections too computationally intensive. There were also parsing issues due to the way the formatting irregularities in the filings. Therefore, we use a modified process to create the sentiment tensors. We extract positive, negative, and neutral words as specified by the Loughran-McDonald sentiment dictionary, and then utilize the proportions in a similar fashion to the news embeddings using Equation (3.1).

The Loughran-McDonald sentiment dictionary is an academically maintained dictionary that lists business-specific words used to gauge the status of a firm. As documented in their 2011 paper in the *Journal of Finance*, the dictionary contains a list of over 80,000 words, which each word flagged with a particular sentiment, such as "positive", "negative", "litigious", etc. We parse the SEC filings and tokenize them, then determine the proportion of positive, negative, and neutral words in the total filing, and then use (3.1), substituting in positive word proportion, negative word proportion, and neutral word proportion for positive sentiment probability, negative sentiment probability, and neutral sentiment probability, respectively. For this investigation, we utilize the 2023 version of the Master Sentiment Dictionary.

An issue that we run into when incorporating SEC filings data is that they are recorded on a annual or quarterly basis, which creates significant gaps between reporting dates. To help fill these, we again use exponential decay, defined in section 3.2, and tune the γ parameter during model training; once again, $\gamma \approx 0.8$ yielded good results.

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. Table 3.3 shows some reported summary statistics on the distribution of SEC filings across the tickers:

Table 3.3: 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

Since there are only 4 filings per year, we use forward filling with decay to fill in the "missing" dates as in Equation 3.2. Giving the addition and dropping of companies from the S&P100, as well as some newer public companies joining, each company does not have the same number of filings over the time period.

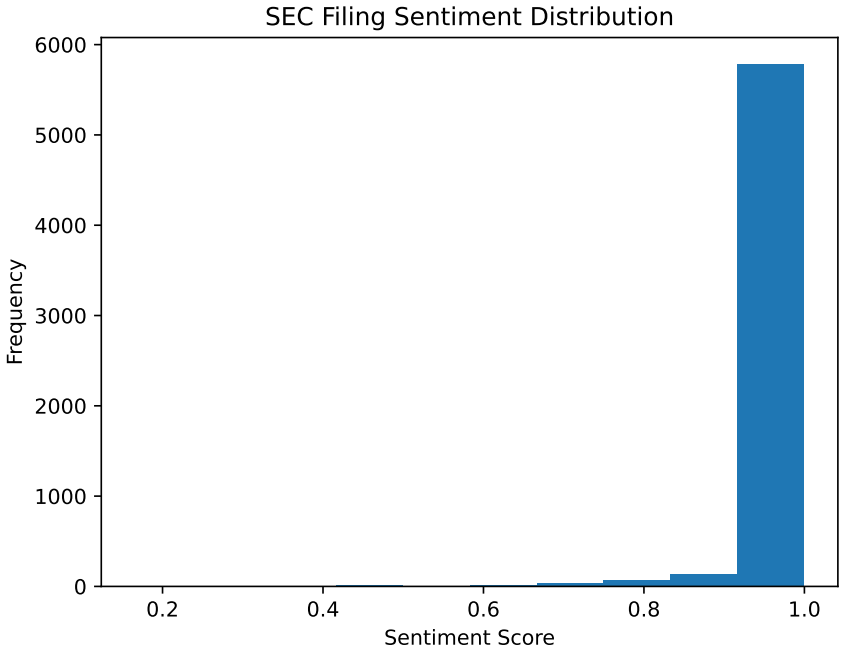


Figure 3.4: Frequency distribution of our novel SEC sentiment scores.

Figure 3.4 shows the distribution of sentiment scores. 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.

Chapter 4

Implementation

While complete documentation of our code would take us too far afield in this report, an overview of our implementation is beneficial in understanding our experimental results. For a more complte explanation, please see our GitHub repository. For more information about the libraries we use, please see the Gymnasium and Stable Baselines 3 documentation sites.

4.1 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.

4.1.1 `AbstractRewardManager`

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

There are two subclasses: `DifferentialSharpeRatioReward` and `ProfitReward`. The former implements the rewards as described in Section 2.2, while the later simply outputs the difference between the beginning and end portfolio values over each time step.

4.1.2 `AbstractDataManager`

This abstract class manages reading and iterating through the state and price data for the environment. In addition to boilerplate, it has the following methods:

- `get_data()`: responsible from loading the necessary price and alternative text data into memory and organizing it for future use. Often, these make use of the functions is `data_utils.py` to read the data from CSV's and perform basic preprocessing.
- `get_state(t, w)`: creates the next state tensor to be given to the agent at the end of the current time step. `t` denotes what the next time step is (for use in indexing the pulled data), and `w` are the portfolio weights from the current time step (will be useful to the agent as they shouldn't stray to far for fear of transaction costs).
- `get_prices(t)`: obtains the prices for all securities at time `t`. Is separate form `get_state` for generalizeability, but they likely access the same price data.

For each experiment that we run, we create separate subclasses of `AbstractDataManager` for the train and test periods. Inside each, we load the appropriate data for the time period and the experiment we are running (that is, whether or not we are using SEC and/or news sentiment data). The `get_state()` method handles combining all the relevant data into a single tensor for the lookback window.

For the state itself, we take the approach of paper [4]. The state observation provided to the policy consists of two parts: the previous portfolio weights `w`, and a 3D tensor of past data. The tensor has

shape $m \times L \times C$, where there are m assets in our universe, the lookback period is L days, and there are C "channels" of information. Each row of the tensor is a different asset, each column is a different day in the past, and each channel is a different type of data (such as price data, SEC data, and news sentiment data); see the figures in Section 2.4 for reference.

The form of state illustrated in [4] was chosen over the state forms presented in [1] and [4] due to its simplicity to implement, and its compatibility with the EIIE system from the same paper which has strong training performancee.

4.1.3 `PortfolioEnvWithTCost`

This class is the main Gymnasium environment and uses that library's standard interface. It takes instances of `AbstractDataManager` and `AbstractRewardManager` as arguments to its constructor.

It iterates through each trading day in the data provided by `AbstractDataManager`. At each step, it takes an action (portfolio weights) from the policy, computes the change in portfolio value, and uses `AbstractRewardManager` to compute a reward to return to the policy before incrementing the day counter. When updating the portfolio value, it employs the transaction costs-cognizant approach described in Section 2.3.

4.2 RL Policies

While we tested both the Proximal Policy Optimization (PPO) and Deep Deterministic Policy Gradient (DDPG) policy training algorithms provided by Stable Baselines 3, DDPG was found to be most effective. We believe this to be because it does a better job of exploring the space of possible portfolios, and thereby avoids getting stuck in local minima.

4.2.1 MLP Policy

For our MLP policies, we simply utilize the standard implementation provided by Stable Baselines 3. The library automatically has functions to handle combining the data and construction a feed-forward neural network within the defaults of its learning algorithm, so we simply make use of this.

4.2.2 CNN EIIE Policy

See the figure in Section 2.4 for reference. We take the 3D tensor part of the state observation and apply a row-wise convolutional neural network; that is, the same CNN is applied to each stock's data separately to produce a feature map for that stock, without referencing any other stock. We then concatenate the previous portfolio weights to this set of features, after which we pass the entire dataset to an MLP model with a Softmax at the end that produces our portfolio weights. The benefit of this method is that much of the feature extraction is performed by the CNN as opposed to an MLP the whole way through, which greatly reduces the complexity of the model as compared to the pure MLP policy.

4.2.3 RNN EIIE Policy

This is very similar to the CNN EIIE policy, except that we apply a row-wise LSTM model to the state tensor to produce the feature map (that is, an LSTM is applied to each stock's data seperately). The concatenation with the previous portfolio weights and the back half of the model is the same. This provides the same benefit as the simplicity of the CNN EIIE policy, but now with the added benefit of LSTM memory.

4.3 Benchmarks and Comparisons

Most of the benchmarks trivial to implement within our RL framework. Their discussion is omitted; the code can be viewed on our GitHub.

For OLMAR and WMAMR, there are two ways to implement that are roughly equivalent. The optimization problems formulated in Sections 2.5 and 2.6 can be solved using standard constrained

solvers, such as those in the `scipy` library. Alternatively, we can utilize the analytical solutions presented in the same section and project them back into the space of allowed portfolios.

The portfolio evaluation statistics are also relatively easy to implement, given their standard formulas. Again, we point to our GitHub implementation.

Chapter 5

Experimental Results

We run several experiments testing various combinations of alternative data usage, reward function, and policy type. We compare the performance of our best models to several standard benchmarks.

Our training time period is the start of 2010 to the end of 2017, and the test time period is the start of 2018 to the end of 2019. This 10-year time span was chosen as it is the largest intersection of all data sources available to us.

All of the strategies described in this chapter were trained and tested on these time periods; only test plots and statistics are shown. For testing, we start each trained strategy with $1 and let it run over the test period, and plot the value of the strategy's portfolio over time. Transaction costs are held at 1% throughout.

The cumulative returns plots for each group of strategies for this section have been collected in Appendix A. We choose to only display comparative summary tables in this section because they are easier to interpret.

5.1 Benchmarks Portfolios

We begin by reviewing the benchmarks highlighted in the overview:

- Naive Equal: A simple strategy that trades to maintain equal weights across all assets (and the risk-free asset).
- Equal Buy-and-Hold: It initially buys equal amounts of all assets, and then does no further trading.
- Best Historical Sharpe: Puts all money in the single asset that had the best Sharpe ratio during the training period.
- OLMAR and WMAMR: Described in Sections 2.5 and 2.6 respectively.
- S&P500: The returns of the index, as provided by CRSP (see Section 3.1.1).

The performance of the benchmarks is shown in Table 5.1 and Figure A.1.

Table 5.1: Benchmark Portfolio Strategies

	Net Profit	Sharpe Ratio	Sortino Ratio	Max Drawdown
Naive Equal	0.112583	0.392352	0.473881	0.197294
Equal Buy-and-Hold	0.181138	0.595225	0.716195	0.198039
Best Historical Sharpe	0.149857	0.475577	0.683296	0.210250
OLMAR	0.083634	0.300906	0.364244	0.199240
WMAMR	0.112565	0.392310	0.473818	0.197294
S&P	0.158347	0.552242	0.675382	0.197728

Most benchmarks underperform the S&P index over the trading period in terms of profitability. The simple Equal Buy-and-Hold is the best performer by net profit, Sharpe Ratio, and Sortino Ratio. Both of the trading strategies OLMAR and WMAMR also lag behind the index. The Best Historical Sharpe stock also underperforms the market, and as much higher volatility indicated by its significantly lower sharpe ratio compared with the S&P with similar profitability. All strategies have similar max drawdown due to the market decline in late 2018 / early 2019.

5.2 RL Portfolios: Historical Price Data

Our first set of strategies closely follows the implementation of [4] as described in Section 2.4. At each time, we provide the agent with a tensor of close/high/low stock prices for the past few weeks. We test policies of CNN EIIEs, RNN EIIE, and a standard MLP neural network on the entire tensor. We also test using both the Differential Sharpe ratio reward and the profit reward.

The results with the Differential Sharpe Ratio reward can be found in Table 5.2 and Figure A.2, while results with the Profit reward are found in Table 5.3 and Figure A.3.

Table 5.2: Performance of Strategies with Historical Prices (Differential Sharpe Reward)

	Net Profit	Sharpe Ratio	Sortino Ratio	Max Drawdown
Hist Prices CNN	0.111205	0.396336	0.477160	0.199721
Hist Prices RNN	0.069154	0.262906	0.318161	0.194177
Hist Prices MLP	0.094686	0.338142	0.409170	0.188082
S&P	0.158347	0.552242	0.675382	0.197728

Table 5.3: Performance of Strategies with Historical Prices (Profit Reward).

	Net Profit	Sharpe Ratio	Sortino Ratio	Max Drawdown
Hist Prices CNN	0.166717	0.561471	0.676669	0.196557
Hist Prices RNN	0.106518	0.369764	0.454653	0.205465
Hist Prices MLP	0.071937	0.261355	0.322155	0.198329
S&P	0.158347	0.552242	0.675382	0.197728

From Table 5.2, strategies using the Differential Sharpe ratio reward all have mediocre returns, significantly underperforming the S&P index across all statistics. This indicates that the policy gradient optimization process is not as effective to optimize the policy. Indeed, the Differential Sharpe Ratio is a difficult reward for the policy to learn given the agent's limited information. We note that the CNN appears to perform better than both the RNN and MLP policies here.

In contrast, in Table 5.3, the CNN and RNN EIIE policies show reasonable performance, but the MLP does not. In fact, the CNN actually slightly outperforms the S&P index, while the MLP policy deteriorates significantly. Profit reward is relatively easy to learn, and so the small (and therefore more noise-robust) EIIE models fare well, while the MLP model overfits. and does not see improvements.

The trends seen by statistical analysis, and also displayed visually in Figures A.2 and A.3, are common themes throughout the rest of this analysis.

5.3 RL Portfolios: Price+SEC Data

We now keep the same types of policies and reward functions, but augment the state tensor with additional channels for the SEC sentiment scores for every asset on every day in the lookback window.

The results with the Differential Sharpe Ratio reward can be found in Table 5.4 and Figure A.4, while results with the Profit reward are found in Table 5.5 and Figure A.5.

SEC data regularly available for all companies in our universe and is of high quality. As a result, we see significant improvement in performance. In Table 5.4, results are strong across the board using the Differential Sharpe reward, but none quite match the S&P index. Differential Sharpe is a difficult

Table 5.4: Strategies with SEC Filings (Differential Sharpe Reward)

	Net Profit	Sharpe Ratio	Sortino Ratio	Max Drawdown
SEC CNN	0.124870	0.439863	0.529951	0.195347
SEC RNN	0.138309	0.464039	0.568351	0.204159
SEC MLP	0.147103	0.482788	0.581065	0.220906
S&P	0.158347	0.552242	0.675382	0.197728

Table 5.5: Strategies with SEC Filings (Profit Reward)

	Net Profit	Sharpe Ratio	Sortino Ratio	Max Drawdown
SEC CNN	0.162534	0.532298	0.649254	0.195766
SEC RNN	0.154506	0.513757	0.626332	0.187671
SEC MLP	0.073242	0.266018	0.329634	0.208675
S&P	0.158347	0.552242	0.675382	0.197728

reward, so it imposes a penalty on optimal performance; however, it is difficult for a model to overfit, so none of the returns are abysmal. In Table 5.5, we see impressive results from the CNN and RNN EIIE policies, but bad performance from the MLP policy. This is likely because the overly-complex MLP policy overfits and suffers. The CNN and RNN policies using the Differential Sharpe ratio are among the strongest contenders in our experiments.

5.4 RL Portfolios: Price+SEC+News Data

Unfortunately, the news data is much less regular than the SEC data, and is not available consistently for all stocks in our universe. While it does provide some improvement over only using historical price data as in Section 5.2, it is not as significant as with SEC data.

However, combining the price, SEC, and news sentiment data provides strong results. The performance of models trained on this dataset with the Differential Sharpe Ratio reward can be viewed in Table 5.6 and Figure A.6, while results for the Profit reward are found in Table 5.7 and Figure A.7.

Table 5.6: Strategies with Combined Data (DiffSharpe Reward)

	Net Profit	Sharpe Ratio	Sortino Ratio	Max Drawdown
SEC+News CNN	0.094546	0.342485	0.412857	0.193996
SEC+News RNN	0.128636	0.444354	0.536047	0.195274
SEC+News MLP	0.053514	0.205204	0.248546	0.200069
S&P	0.158347	0.552242	0.675382	0.197728

Table 5.7: Strategies with Combined Data (DiffSharpe Reward)

	Net Profit	Sharpe Ratio	Sortino Ratio	Max Drawdown
SEC+News CNN	0.166081	0.553644	0.664706	0.195472
SEC+News RNN	0.142287	0.519258	0.636281	0.170134
SEC+News MLP	0.065676	0.248586	0.304594	0.195750
S&P	0.158347	0.552242	0.675382	0.197728

As seen in Table 5.6, the policies trained for Differential Sharpe Ratio do not fare as well because the irregularity of the news makes a bigger impact on the harsher reward. Thus, we end up seeing similar results to Section 5.2, with the CNN and RNN policies performing better than the MLP but still mediocre.

However, for the profit reward, we do see a notable increase in Table 5.7. Indeed, our best-performing model across all three sets of experiments is the CNN EIIE model on the combined dataset using Profit reward. Again, the MLP dramatically overfits.

5.5 Comparison of RL Against Benchmarks

Figures A.8 and A.9 and Table 5.8 compare the performance of the best models we trained against select benchmarks from Section 5.1.

	Net Profit	Sharpe Ratio	Sortino Ratio	Max Drawdown
Equal Buy-and-Hold	0.181138	0.595225	0.716195	0.198039
OLMAR	0.083634	0.300906	0.364244	0.199240
WMAMR	0.112565	0.392310	0.473818	0.197294
SEC+News CNN (Profit)	0.166081	0.553644	0.664706	0.195472
SEC+News RNN (Profit)	0.142287	0.519258	0.636281	0.170134
SEC CNN (DiffSharpe)	0.124870	0.439863	0.529951	0.195347
SEC RNN (DiffSharpe)	0.138309	0.464039	0.568351	0.204159
S&P	0.158347	0.552242	0.675382	0.197728

Table 5.8: Comparison of Best Strategies against Benchmarks

Excluding Equal Buy-and-Hold, our SEC+News CNN EIIE policy with Profit reward has the highest net profit, Sharpe ratio, and Sortino ratio. Additionally, the SEC+News RNN policy with the profit reward has the lowest max drawdown of all strategies. All of our trained strategies outperform the OLMAR and WMAMR trading benchmarks.

5.6 Analysis 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, agents trained on CNN EIIE and RNN EIIE, exhibit enhanced performance under the profit reward, while the MLP policy network seems to overfit significantly. This is because compared to the MLP, both CNNs and RNNs have fewer weights and biases to learn.

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.

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

News data also provides benefit when used in combination with SEC data. However, the irregularities in the data have a significant impact when using a difficult reward (like the Differential Sharpe ratio).

Optimal performance is achieved when integrating both news and SEC data into the agent's environment. This outcome underscores the untapped potential of research avenues exploring comprehensive datasets capturing nuanced company sentiment.

With a larger and more consistent news headline dataset, we believe that we could create better-performing agents. However, the challenges in using alternative data can be somewhat reduced by choosing an appropriate reward and a policy that is sufficiently resistant to overfitting.

Furthermore, there are other routes to improve returns beyond simply improving the quality of our data. For instance, we could test different feature extractors and applying different regularization techniques. In addition, different sentiment embedding functions could potentially be a more accurate and/or usable by our agents.

Bibliography

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

[2] 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.

[3] 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].

[4] 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.

[5] 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.

[6] 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].

[7] SEC.gov | how to read a 10-k/10-q, . URL https://www.sec.gov/oiea/investor-alerts-and-bulletins/how-read-10-k10-q.

[8] 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.

[9] 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.

[10] Bin Li and Steven C. H. Hoi. On-line portfolio selection with moving average reversion. URL http://arxiv.org/abs/1206.4626.

[11] Li Gao and Weiguo Zhang. Weighted moving average passive aggressive algorithm for online portfolio selection. In *2013 5th International Conference on Intelligent Human-Machine Systems and Cybernetics*, volume 1, pages 327–330. doi: 10.1109/IHMSC.2013.84. URL https://ieeexplore.ieee.org/document/6643896.

[12] Wharton Data Research Services. CRSP daily stocks, 2010-2024. URL https://wrds-www.wharton.upenn.edu/pages/get-data/center-research-security-prices-crsp/annual-update/stock-security-files/daily-stock-file/.

[13] Miguel Aenille. Daily finaincial news for 6000+ stocks, 2020. Retreived 2024-05-05 from https://www.kaggle.com/datasets/miguelaenlle/massive-stock-news-analysis-db-for-nlpbacktests/data.

[14] SEC.gov | EDGAR | company filings, . URL https://www.sec.gov/edgar/searchedgar/companysearch.

Appendices

Appendix A

Image Gallery

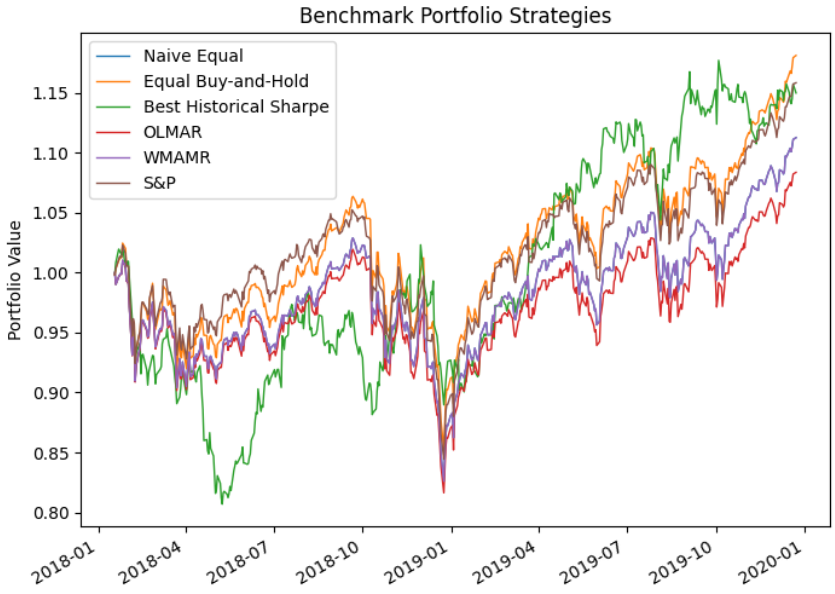


Figure A.1: Performance of benchmark strategies on test time period

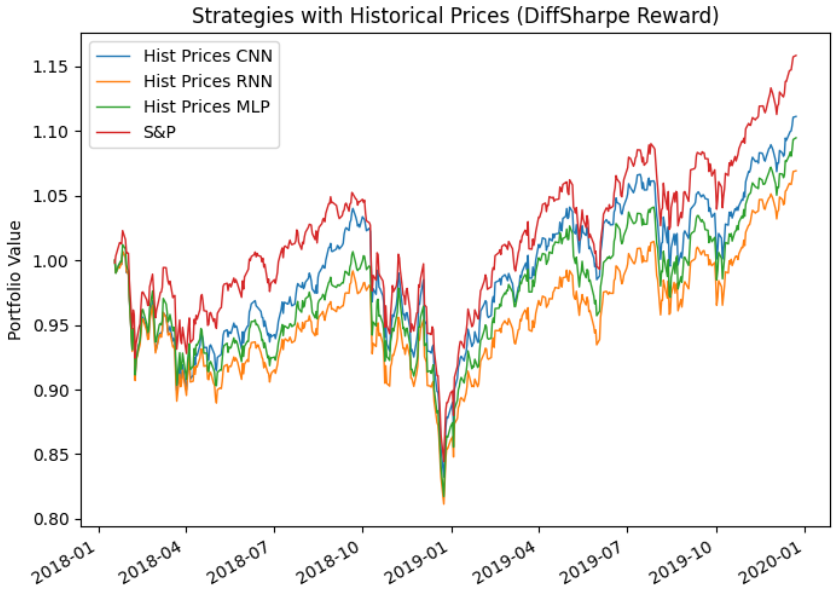


Figure A.2: Performance of strategies using historical price data with Differential Sharpe reward

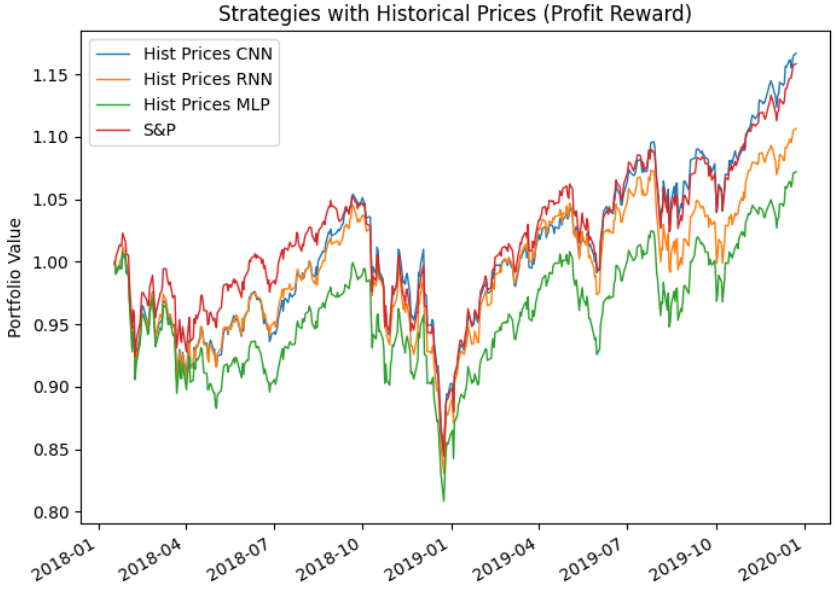


Figure A.3: Performance of strategies using historical price data with Profit reward

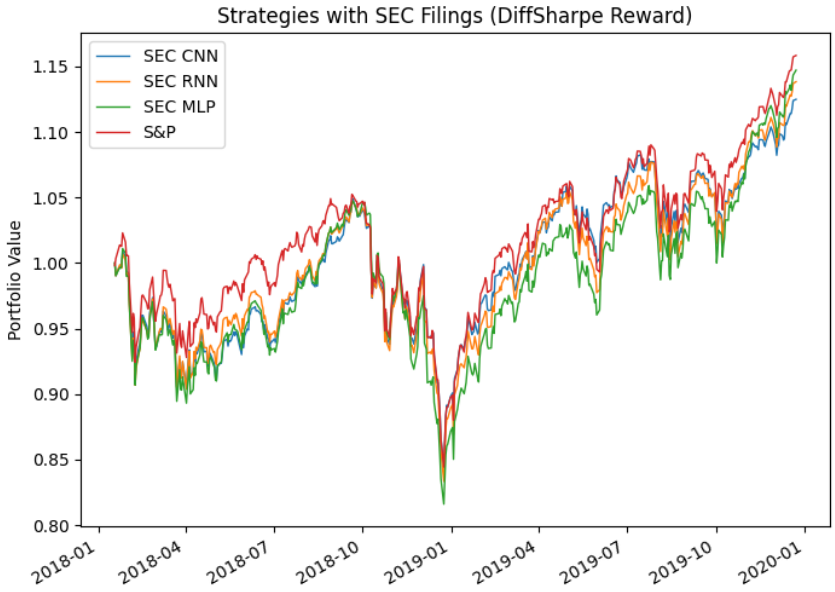


Figure A.4: Performance of strategies using price and SEC data with Differential Sharpe reward

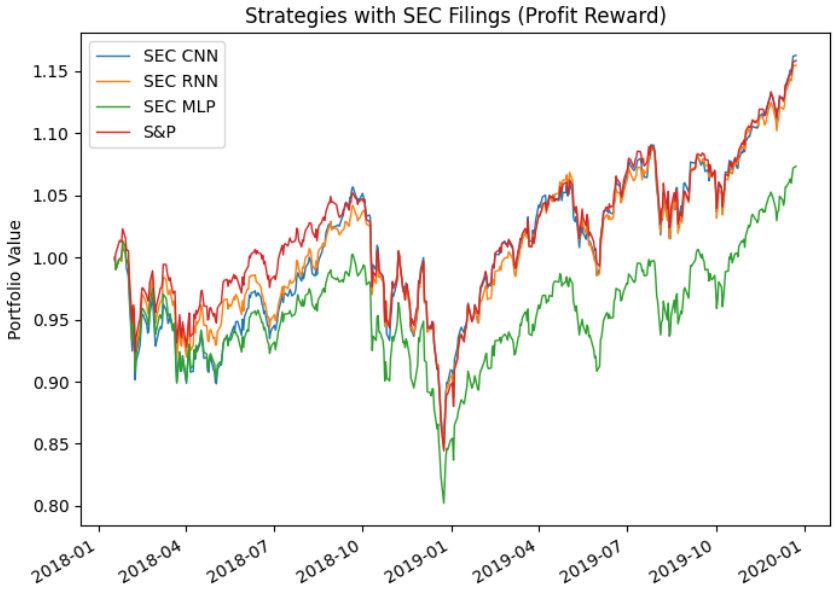


Figure A.5: Performance of strategies using price and SEC data with Profit reward

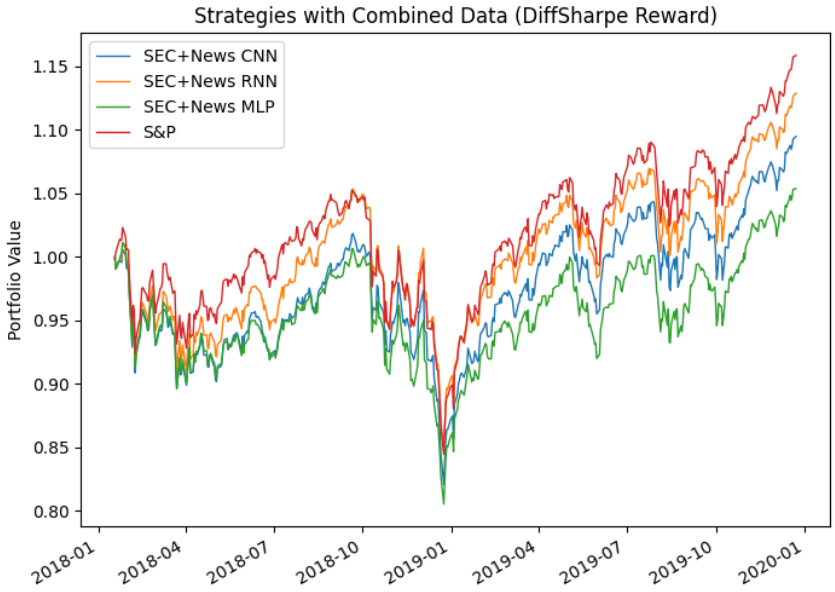


Figure A.6: Performance of strategies using price, SEC, and news data with Differential Sharpe reward

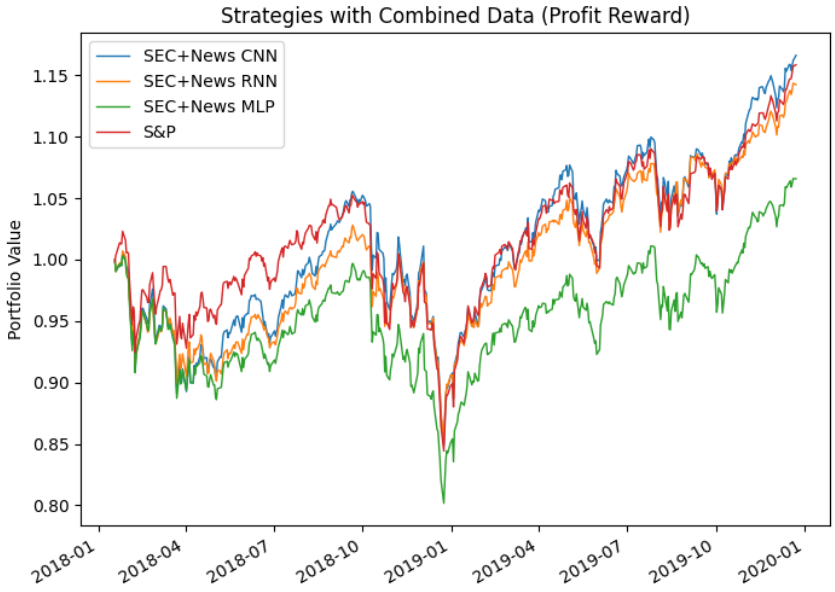


Figure A.7: Performance of strategies using all price, SEC, and news data with Profit reward

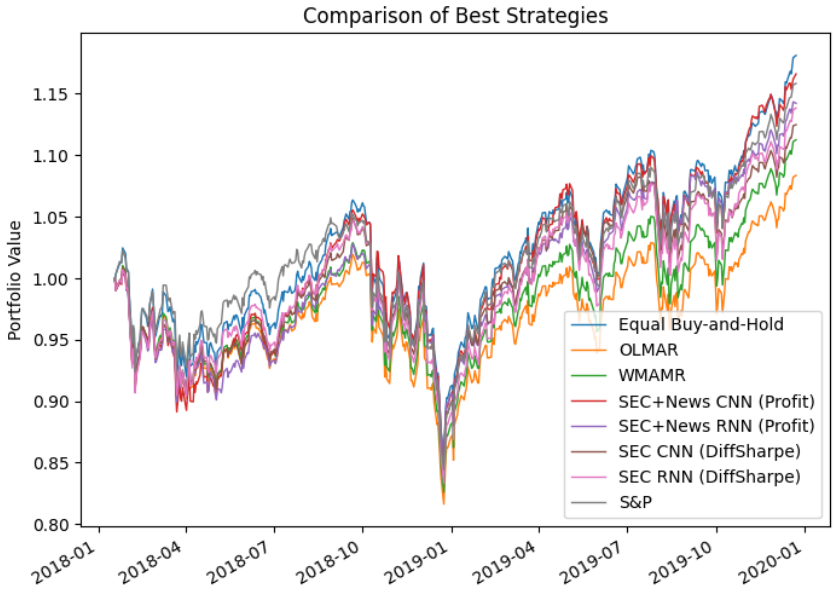


Figure A.8: Comparison of Best Strategies

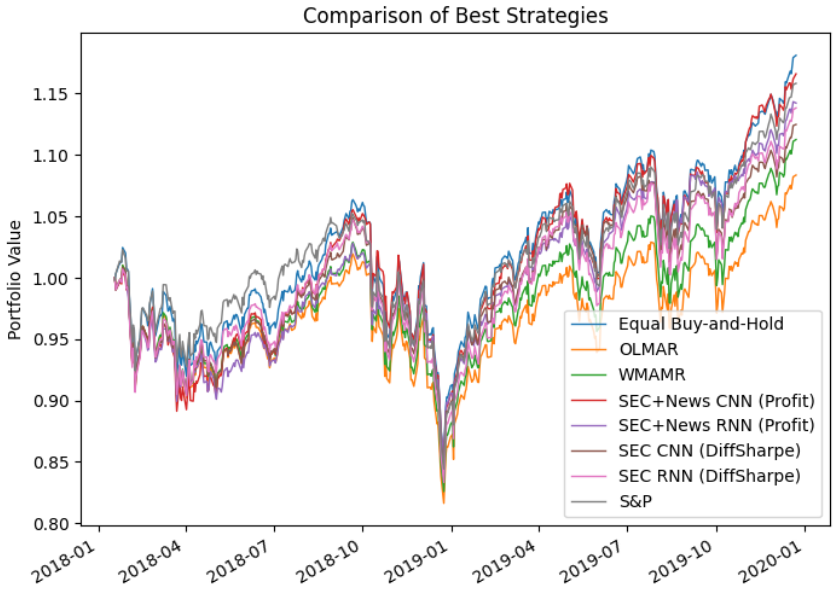


Figure A.9: Comparison of Best Strategies (with rescaled y-axis)