SUMMER 2022

FINAL REPORT

BOA Reinforcement Learning Capstone

Deep Reinforcement Learning in Quantitative Wealth and Investment Management

Nicole Zhao (xz2616)
Teresa Duan (rd3111)
Tianjing Cao (tc3385)
Wenqing Gao (wg2125)
Zhuotian Tang (zt2017)

Supervised By
Dr. Cristian Homescu , Dr. Francisco Rubio

Contents

1 Introduction

Quantitative strategy is a significant topic in modern Finance. Among distinct topics in quantitative finance, asset allocation, which is the process of finding optimal weights for component assets in a portfolio, is one of the most important areas . By tilting toward exposure to different kinds of assets, investors could obtain great diversified portfolio. For decades, the research on asset allocation problems to guide investment actions is always a blooming direction.

This paper is inspired by Frederik (2021)[1] that applied Deep Reinforcement Learning (DRL) to overcome the hurdle of Markowitz Modern Portfolio Theory, which determines the optimal portfolio policy by estimating the expected returns and covariance matrix. Based on their research, we mainly focuse on combining Deep Q-Network (DQN) on global indices' asset allocation and performing experimental tests to observe whether or not our model will further prove the profitability and generalization.

The input data in this project is time series data (years of daily) of 8 different market indices. We combine reinforcement learning with three different neural network strategies — Convolutional Neural Network, Recurrent Neural Network, and Long Short-Term Memory Neural Network — to train the optimal allocation. The reward function is assigned as portfolio's sharpe ratio at reallocation spot, so that the trading agent will be driven to the direction of maximizing sharpe ratio. The logic here is to search for the allocation that strikes some sort of balance between maximizing expected return and reducing the risk of the portfolio. Also, we utilize the Equal Weighted Portfolio (EWP), where each asset has an equal weight in the portfolio, as our benchmark. In addition to comparing the returns of different neural network approaches to the benchmark, we also made other comparisons, such as normalizing the loss, replacing Sharpe ratio with Sortino ratio to measure the effectiveness of the different allocation policies, and comparing with traditional portfolio allocation models — Markowitz Model and Black-Litterman Model.

Based on our analysis, we can conclude that there is no clear winner among the three DRL strategies versus benchmark, depending on how we choose the loss function and whether it is normalized or not. Compared to classic portfolio allocation approaches, DRL models stands out for its ability to capture arbitrage space when there is a clear trend in the market.

2 Related Work

It has long been challenging to design a comprehensive strategy for capital allocation optimization in a complex and dynamic stock market. With development of Artificial Intelligence, machine learning coupled with fundamentals analysis and alternative data has been in trend and provides better performance than conventional methodologies. Reinforcement Learning (RL) as a branch of it, is a blooming field famous for addressing a wide scope of complex decision-making tasks. The agent is able to learn from interactions with environment, during which it continuously absorbs information, takes actions, and learns to improve its policy regarding rewards or losses obtained.

As illustrated by Das [2], a widely used dynamic programming known in RL literature is solving the "planning problem", which works backward from all values of mean-variance pairs to achieve the best outcomes. The approach is model-free and may be applied to problems with path-dependency and extremely large state spaces, which is often seen in the finance world. To deal with the commonly high dimensional data in wealth management tasks, Dixon introduced G-Learner as an RL algorithm that operates with explicitly defined one-step rewards, does not assume a data generation process, and is suitable for noisy data[3].

Researchers also paid efforts applying these techniques in Goal-Based Investing (GBI). According to Das [4], Sanjiv developed a dynamic programming methodology with investor preferences to maximize investor outcomes over multiple, potentially competing goals, even when financial resources are limited. Two years later, he reported an improved algorithm to construct an optimal portfolio trading strategy that maximizes the probability of attaining an investor's specified target wealth at the end of a designated time horizon, given a set of efficient or even inefficient portfolios [5].

Moreover, for solving GBI problems, Das et al., proposed two other improvements. The first smoothes and improves the convergence of the approximate solutions towards the underlying, continuous solution either by using analytic solutions at the penultimate time point or by adjusting the wealth grid. The second suggestion pertains to fast matrix products and is purely computational but has a large impact on the time required to solve the problem [6].

On top of that, within a Deep Reinforcement Learning (DRL) framework, the RL agents construct and learn their knowledge through deep learning of neural networks. As pointed out by Arulkumaran [7], DRL can learn from raw sensors directly as input, which makes it an efficient solution to the traditional portfolio management problem. As demonstrated by [8], an RL method being used to find a risk-adjusted reward function to evaluate the expected total rewards for policy, was combined with GRUs, one of the deep recurrent neural network (RNN) models, which decides how earlier states and actions influence the policy optimization in non-Markov decision processes, and resulted in an optimal model.

Much work has also been done on choices of different DRL models. Katongo and Bhattacharyya in [9], seeking to increase the portfolio's risk-adjusted returns by capturing market trends and anomalies and reallocating the component assets, employed DRL Algorithms (PPO, A2C, and DDPG) as well as Neural Network Autoencoders. Similarly, Yang et al. in [10] reveals a DRL scheme that automatically integrates the three actor-critic algorithms. Focusing on Convolutional Neural Network, Benhamou et al., in [11] used three individual networks as inputs, including portfolio returns, asset features, and the previous portfolio allocation.

3 Asset Allocation Problems

Our team aim at using the RL techniques to solve asset allocation problem.

Typically, an asset allocation problem can be described as a mean-variance optimization (MVO) problem.

$$w^* = \arg\max_{w \in \mathcal{C}} \{w' \mathbb{E}[r] - \frac{\lambda}{2} w' var[r] w\} \tag{1}$$

where
$w-$ the portfolio weights / holdings
$\mathbb{E}[r]-$ the vector of expected returns
$var[r]-$ the covariance matrix of returns
$\lambda-$ the risk aversion
$\mathcal{C}-$ the set of constraints (linear equality/inequality constraints, etc.)

This economic theory has been evolving for several decades, and has an enormous application on related areas. During the time MVO shows its flaws. First, the efforts takes to estimate the covariance matrix grow dramatically with the increase of assets. Second, the optimized results are highly sensitive to the expected returns and the covariance matrix, so that a small deviation of estimate could lead to a large bias of the allocation. Furthermore, the MVO is constructed on the assumption of market equilibrium, which can not be reached in reality. Several techniques were introduced to decrease these influence, the Black Litterman model is a good example. On the other hand, with the blooming of machine learning, more studies focus on solving asset allocation via reinforcement learning.

Reinforcement learning allows us to solve these dynamic optimization problems in an almost model-free way, relaxing the assumptions often needed for classical approaches. However, how to formulate the expected utility maximization problems using modern machine learning techniques remains a huge challenge for us. We also notice that the goal of asset allocation is to obtain the weights of the assets that maximize the rewards in a given state of the market considering risk and transaction costs. So the first step should be solving a prediction problem for the vector of expected returns and covariance matrix, which is exactly the framework of Deep Q-Network (DQN).

4 Reinforcement Learning Theory

Different from ML, RL develops from the idea of "interacting with the environment", which means the RL training agent gains experience and improves performance by constantly interacting with an environment. More specifically, the basic framework of RL problem consists of agent (the learner and decision maker)and environment that is defined by the Markov Decision Process (MDP) which is denoted by $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R})$. $\mathcal{S}$ is the set of all possible states (state space), $\mathcal{A}$ is collection of all available actions (action space), and $\mathcal{R}$ decides the reward received after each transformation between states.The goal of agent is to implement the optimal policy π^* which directs to take actions and gets the maximum total amount of reward (expected return) in the whole trajectory starting from any state.

Another important concept is value function. Formally, value function v under policy π is defined by:

$$v_\pi(s) = \mathbb{E}_\pi[\sum_{k=0}^{\infty} d^k R_{t+k+1}]$$

This state-value function tells the value of state s. Similarly, the action-value function that tells the value of taking action a at state s is given by:

$$q_\pi(s,a) = \mathbb{E}[\sum_{k=0}^{\infty} d^k R_{t+k+1} | S_t = s]$$

A fundamental property of value function is the recursive relationship it satisfies.The value under policy π at state s can be expressed by summation of the value of all possible successor states weighted by the transformation probability:

$$v_\pi(s) = \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)[r + d v_\pi(s')]$$

It is also called the Bellman equation for v_π. The value function is a powerful tool to measure the performance of policies, and a better policy is defined to be the one that has no less value at any state than the other one. From this well defined order we can find the optimal policy which takes the maximum value at every state. Thus the evaluation of a policy can be naturally transformed to the estimation of its value function v_π, and performance of actions can be directly illustrated as the difference of value in next state and value in present state. In the training process, the goal-seeking agent trades off be- tween exploration (taking actions it has not known the reward) and exploitation (taking actions with the greatest expected reward) to decide the next action, and gradually drives the value function at each state towards the maximum. The value function under optimal policy in the form of Bellman equation can be expressed by:

$$v_{\pi^*}(s) = \max_{a \in A(s)} \sum_{s',r} p(s', r|s, a)[r + dv_{\pi^*}(s')]$$

Equivalently, for the action-value function under optimal policy it is:

$$q_{\pi^*}(s, a) = \sum_{s',r} p(s', r|s, a)[r + dv \max_{a' \in A(s')} q_{\pi^*}(s', a')]$$

In the most of cases, people do not have the complete knowledge of the environment, as the state transformation probability $p(\cdot)$ is always unknown. The different approaches to approximate the solution to optimal Bellman equation forms the basis of a number of RL algorithms.

4.1 Q-Learning Algorithm

Q-learning is a model-free RL algorithm and it can handle problems with stochastic transitions and rewards without requiring adaptations. For any finite Markov decision process (FMDP), Q-learning finds an optimal policy in the sense of maximizing the expected value of the total reward over any and all successive steps, starting from the current state. Q-learning can identify an optimal action-selection policy for any given FMDP, given infinite exploration time and a partly-random policy. "Q" refers to the function that the algorithm computes – the expected rewards for an action taken in a given state.

After Δt steps into the future the agent will decide some next step. The weight for this step is calculated as $\gamma^{\Delta t}$, where γ (the discount factor) is a number between 0 and 1 $(0 \leq \gamma \leq 1)$and has the effect of valuing rewards received earlier higher than those received later (reflecting the value of a "good start"). γ may also be interpreted as the probability to succeed (or survive) at every step Δt.

The algorithm, therefore, has a function that calculates the quality of a state–action combination:

$Q : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$

Before learning begins, Q is initialized to a possibly arbitrary fixed value (chosen by the programmer). Then, at each time t the agent selects an action a_t, observes a reward r_t, enters a new state s_{t+1} (that may depend on both the previous state s_t and the selected action), and Q is updated. The core of the algorithm is a Bellman equation as a simple value iteration update, using the weighted average of the old value and the new information:

$$Q^{new}(s_t, a_t) \leftarrow \underbrace{Q(s_t, a_t)}_{\text{old value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \overbrace{\Bigg(\underbrace{\underbrace{r_t}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(s_{t+1}, a)}_{\text{estimate of optimal future value}}}_{\text{new value (temporal difference target)}} - \underbrace{Q(s_t, a_t)}_{\text{old value}} \Bigg)}^{\text{temporal difference}}$$

where r_t is the reward received when moving from the state s_t to the state s_{t+1}, and α is the learning rate $(0 < \alpha \leq 1)$.

Note that $Q^{new}(s_t, a_t)$is the sum of three factors. The first one is $(1 - \alpha)Q(s_t, a_t)$, the current value weighted by the learning rate. Values of the learning rate near to 1 make the changes in Q more rapid. The second one is $\alpha\, r_t$, the reward $r_t = r(s_t, a_t)$ to obtain if action a_t is taken when in state s_t (weighted by learning rate). The third one is $\alpha\gamma \max_a Q(s_{t+1}, a)$, the maximum reward that can be obtained from state s_{t+1}(weighted by learning rate and discount factor).

An episode of the algorithm ends when state s_{t+1} is a final or terminal state. However, Q-learning can also learn in non-episodic tasks (as a result of the property of convergent infinite series). If the discount factor is lower than 1, the action values are finite even if the problem can contain infinite loops. For all final states s_f, $Q(s_f, a)$ is never updated, but is set to the reward value r observed for state s_f. In most cases, $Q(s_f, a)$ can be taken to equal zero.

4.2 Q-Learning Implementation

From the pseudo code Algorithm 1, we can see the only difference is the update rule for value function. In the process of Q-learning, generated actions in a trajectory does not follow the action selected in Q's updating. This implies that the learning data is derived from the policy that is "off" the policy being estimated, and this kind of algorithms represented by Q-learning is termed "off-policy". Q-learning algorithms directly learns the optimal value function thus it usually converges in faster speed, and the result of Q-learning may be affected by the large variance of training data and could be trapped in the local optimal control.

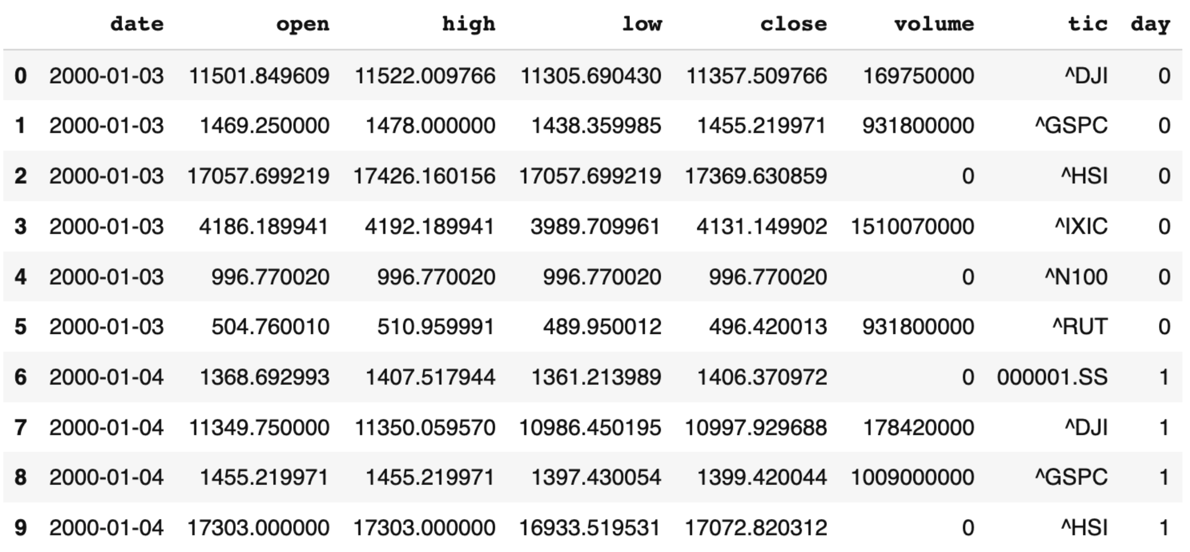
Algorithm 1 Q-Learning Algorithm

```
Require: State space $\mathcal{S}$, action space $\mathcal{A}$,discount rate $d$ and learning rate $\alpha$
  Initialize $Q(\mathcal{S}, \mathcal{A})$ with zeros
  repeat forever (for each episode):
    Initialize $S$
    Generate action $A$ under $\epsilon-$soft policy from present $V(A)$
    repeat forever (for each episode):
      Take action $A$, observe $R, S'$
      $Q(S, A) \leftarrow Q(S, A) + \alpha[R + d \max_a Q(S', a) - Q(S, A)]$
      $S \leftarrow S'$
      Choose $A$ from $S$ using $\epsilon$-soft based on the present $Q$
  until S is terminal
```

5 Data Description

A market index is a collection of investments, such as stocks, that are grouped to track the performance of a particular segment of the financial market. Stock market indices around the world are powerful indicators for global and country-specific economies. In the United States the S&P 500, Dow Jones Industrial Average, and NASDAQ Composite are the three most broadly followed indexes by both the media and investors. Additionally, there are approximately 5000 others that make up the U.S. equity market. Indices can be constructed in a wide variety of ways but they are commonly identified generally by capitalization and sector segregation (Investopedia). Investment managers use indices as benchmarks for performance reporting. Meanwhile, investors of all types use indexes as performance proxies and allocation guides. Indexes also form the basis for passive index investing often done primarily through exchange-traded funds that track indices specifically.

We use the data for gold and 7 indices including global indices and US indices for the period from January 01, 2000 to June 30, 2022 giving us a total of 38,793 data points. The data source is Yahoo Finance. The list of 7 indices are S&P 500 (ˆGSPS), Dow 30 (ˆDJI), Nasdaq (ˆIXIC), Russell 2000 (ˆRUT), HANG SENG INDEX (ˆHSI), SSE Composite Index (000001.SS) and Euronext 100 Index (ˆN100). The table below shows the head of the downloaded data frame.



	date	open	high	low	close	volume	tic	day
0	2000-01-03	11501.849609	11522.009766	11305.690430	11357.509766	169750000	^DJI	0
1	2000-01-03	1469.250000	1478.000000	1438.359985	1455.219971	931800000	^GSPC	0
2	2000-01-03	17057.699219	17426.160156	17057.699219	17369.630859	0	^HSI	0
3	2000-01-03	4186.189941	4192.189941	3989.709961	4131.149902	1510070000	^IXIC	0
4	2000-01-03	996.770020	996.770020	996.770020	996.770020	0	^N100	0
5	2000-01-03	504.760010	510.959991	489.950012	496.420013	931800000	^RUT	0
6	2000-01-04	1368.692993	1407.517944	1361.213989	1406.370972	0	000001.SS	1
7	2000-01-04	11349.750000	11350.059570	10986.450195	10997.929688	178420000	^DJI	1
8	2000-01-04	1455.219971	1455.219971	1397.430054	1399.420044	1009000000	^GSPC	1
9	2000-01-04	17303.000000	17303.000000	16933.519531	17072.820312	0	^HSI	1

Figure 1: Data Frame

The S&P 500, which is short for Standard & Poor's 500, was introduced in 1957 as a stock market index to track the value of 500 corporations that have their stocks listed on the New York Stock Exchange (NYSE) and the NASDAQ. Its index symbol is ˆGSPS. The collection of stocks that make up the S&P 500 is designed to represent approximately 80% of the total value of the U.S. stock market. The S&P 500 Index is a market-weighted index (also referred to as capitalization-weighted). Therefore, every stock in the index is represented in proportion to its total market capitalization. The value of the S&P 500 and various stocks within the index is closely watched by market participants since their performance represents a gauge of the health of the U.S. economy. That is why we choose S&P 500 as the first index to test our strategy.

Dow Jones Industrial Average (DJIA) is one of the oldest, most well-known, and most frequently used indices in the world. It includes the stocks of 30 of the largest and most influential companies in the United States. Its index symbol is ˆDJI. The DJIA is a price-weighted index and represents about a quarter of the value of the entire U.S. stock market. A change in the Dow represents changes in investors' expectations of the earnings and risks of the large companies included in the index. In general, the Dow is known for its listing of the U.S. markets best blue-chip companies with regularly consistent dividends.

The NASDAQ Composite Index is known for being a heavily tech weighted index, which includes software, biotech, semiconductors, and more sub-sectors across the tech market. The NASDAQ Composite Index is a market-capitalization-weighted index of all the stocks traded on the NASDAQ stock exchange and it also includes some non-US based companies. Its index symbol is ˆIXIC. Unlike the S&P 500 and the Dow Jones Industrial Average, NASDAQ also includes many speculative companies with small market capitalization. As a result, its movement generally indicates the performance of the technology industry as well as investor's attitudes toward more speculative stocks.

Russell 2000 Index refers to a stock market index that measures the performance of the 2,000 small-cap to mid-cap companies included in the Russell 3000 Index. It represents approximately 10% of the total Russell 3000 total capitalization. The index was launched in 1984 by the

Frank Russelll Company and is now managed by FTSE Russell. As of Q1 2022, the top 10 holdings in the Russell 2000 Index by market capitalization included: Ovintiv Inc. (OVV), Antero Resources Co. (AR), Chesapeake Energy (CHK), Southwestern Energy (SWN), Range Resources (RRC), Biohaven Pharmaceutical (BHVN), BJ's Wholesale Club Hdlg. (BJ), Avis Budget Group Inc. (CAR), PDC Energy (PDCE), and Willscot Mobile (WSC). Many investors compare small-cap mutual funds against the index's movement as it is seen as a reflection of opportunities in that entire subsection of the market than narrower indices, which may contain biases or more stock-specific risks tht can distort performance.

The Hang Seng Index (HSI) is a free-float market capitalization-weighted index of the sixty largest companies that trade on the Hong Kong Exchange (HKEx). The index is composed of four sub-sector indices in industry, finance, utilities, and real estate investment trusts and the index is a benchmark for blue-chip stocks traded on the Hong Kong stock exchange.

The SSE Composite, short for the Shanghai Stock Exchange Composite Index, is a stock market composite made up of all the A- and B-shares on the Shanghai Stock Exchange, which is the largest stock exchange in mainland China. The index is calculated by using a. base period of 100 and the first day of reporting was July 15, 1991. Much of the total market cap of the SSE is made up of formerly state-run companies like major commercial banks and insurance companies (Investopedia).

The Euronext 100 Index is the blue chip index of the pan-European exchange, Euronext NV. The index comprises the largest and most liquid stocks traded on Euronext. Each stock must trade more than 20 percent of its issued shares over the course of the rolling one year analysis period. The index is reviewed quarterly through a size and liquidity analysis of the investment universe. Each stock in the index is given a sector classsification (Wikipedia).

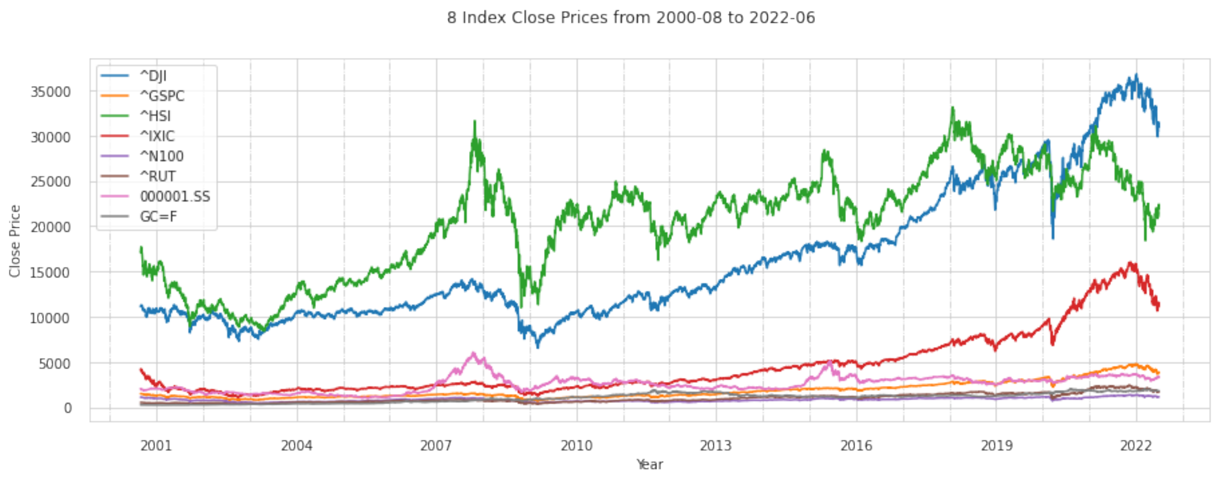


Figure 2: Indices Close Prices from 2000 to 2022

Diversification of a portfolio means varying the asset classes. The value of a company's stock or index goes up or down, changes with the market, and the paper certificates can be worth nothing. The value of gold goes up and down and changes with the market but is never worth nothing (Investopedia). Therefore, we decided to include gold into our portfolio.

Based on the pre-processed data-set which has all information of all indexes in one form, we only extracted out the close prices for each index to build a price matrix. After data cleaning process, the final price matrix was expressed with 8 columns representing the 8 indexes, and 5677 rows being the prices on business days throughout the time, as shown in Figure 2. On top of that, we calculated a corresponding return matrix by computing the daily returns of indexes on each day, which would be used in the DRL model.

In train-test-split, we took rolling windows of close prices as the input X, and the daily return on the last day of each window be the corresponding target y. After normalization and splitting, the train and test datasets were appropriate for the DRL model training.

6 Design of Experiment

Inspired by the work from Frederik [1], we decide to implement an application of multi-indices asset allocation based on Deep Q-learning Network.

Our implementation is described as follows:

- **Training Models**: We combine the neutral networks CNN, RNN, and LSTM, with reinforcement learning process. And we define the sharpe ratio as our reward function, so that the model's optimizer could drive the model to the direction of maximizing sharpe ratio.
- **Number of Assets and Lags**: This is explicit in RNN and LSTM models. In CNN, the dimensionality of the "image" is determined from the number of assets - in this case 8. We choose the number of lags in this project to be 10.
- **Benchmarks**: We use the Equal Weighted Portfolio (EWP) as benchmark.
- **Data Input**: We use the indices' close price within last 10 days as our input and investment allocation weights as model's output vector. We also make the values of price series standardized. The data is split into training, validation, and test data. The training data is used to train the models.The validation data is used for tuning hyperparameters. The hyperparameters resulting in the model with best performance on the validation data is then tested on the test data for the final performance measurement.

Our overall framework is described as follows:

- **Step1**: Model the asset allocation problem into the reinforcement learning framework. Determine settings of necessary details such as the choice of reward function, time period, etc.
- **Step2**: Retrieve historical daily-basis price of multiple important global indices using package *yfinance*. Clean the data by standard data-processing methods.
- **Step3**: Implement a series of commonly used neutral networks, including CNN, RNN, LSTM, etc. Predict indices' future price at each time spot.
- **Step4**: Train a q-learning RL agent on historical price from 2010-2018 and backtest the trading agent's performance on 2019-2022.
- **Step5**: Analyze NN model's predictions under different network structures. Using metrices such as volatility, Sharpe Ratio (SR), and maximum drawdown to compare the back-testing performance between strategy from mean-variance optimization and deep-Q-network.

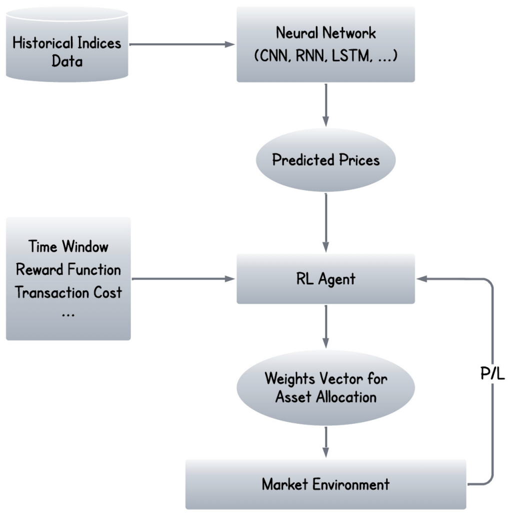


Figure 3: Flow Chart of Trading System

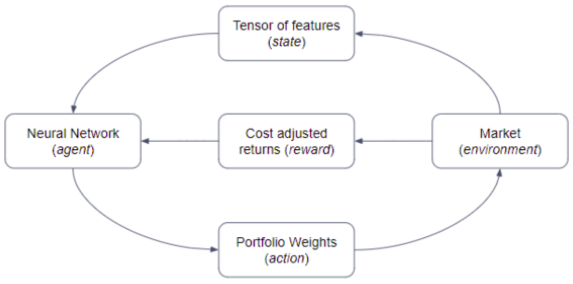

Tensor of features (state)
Neural Network (agent)
Cost adjusted returns (reward)
Market (environment)
Portfolio Weights (action)

Figure 4: MVO in a RL framework

6.1 Classic Method: Markowitz Model

Section 6.1 and 6.2 aimed at using traditional models, such as the Mean-variance optimization model and the Black-Litterman model, to construct optimal portfolios and apply them to real world data. The real world data applied in this section, which has been demonstrated in section 5, is exactly the same data we applied to the Deep Reinforcement Learning model in section 7.

Markowitz Portfolio Optimization also known as Mean-variance optimization, is a kind of classical portfolio optimization technique introduced by Markowitz. This approach can be summarized as maximizing an objective function known as the utility function while subjecting to various restrictions. Here we consider a quadratic optimization problem and rebalance the optimal portfolio over 20 days. To illustrate the principal clearly, let w_i denote the weight of asset i throughout the period, p_i denote the relative price change of asset i over the period, μ and Σ denote the mean and covariance of p respectively. Therefore, the overall return on the portfolio is $r = p^T w$ with mean $\mu^T w$ and variance $w^T \Sigma w$. The optimization problem can be described as a trade-off between the mean of the return, and the variance of the return:

$$\text{minimize } w^T \Sigma w$$

$$\text{subject to } \mu^T w \geq r_{min};\ 1^T w = 1;\ w > 0,$$

where r_{min} is the minimum required return. We consider the risk-free rate as 0.03 (annualized) in this section.

Computations of optimal w are based on *CVXPY*, an open source Python-embedded modeling language for convex optimization problems, and executed in the function named ‘MarkowitzOpt’. The basic structure of this function can be summarized as:

Step 1: Generate the mean matrix (μ) from the input.

Step 2: Generate nearest positive definite covariance matrix (Σ) to make sure the covariance matrix is symmetric positive semidefinite.

Step 3: Compute the optimal w which maximizes portfolio return and minimizes risk restricted to long-only portfolio with minimal return $r_{min} = 0.03$.

We compute the optimal w in each separate period (20 days) and calculate the cumulative return over the whole trading period. We can see the in-sample and out-sample results of portfolio returns versus time by Mean-variance optimization method separately in Figure 15 and Figure 16 (shown in orange lines).

6.2 Classic Method: Black-Litterman Model

Black-Litterman Model is a type of portfolio allocation model developed based on Mean-variance optimization modern portfolio theory. One main difference between Black-Litterman Model and Mean-variance optimization Model is that Black-Litterman Model allows portfolio investment analysts to incorporate their subjective views on performance of one or several components in the asset portfolio. This model offers a systematic method to estimate the mean and covariance of asset returns by combining analyst perspectives and equilibrium returns as opposed to solely relying on historical asset returns. Since it considers both 'neutral' ('implied') returns and 'subjective' return, the BL model consistently produces better out-of-sample performance, such as Sharpe ratios and Omega measures, than MV, BS, and Minimum-variance optimized portfolios. In addition, sensitivity analyses imply that the BL model's superior out-of-sample performance over MV is attributable to the incorporation of additional information on the dependability of return estimates, which led to more accurate and stable return estimations and, as a result, decreased portfolio turnover[12].

Here we have constructed BL model as another kind of traditional portfolio allocation model. By combining the market-implied returns with views on specific assets in the portfolio, we expect to produce a posterior estimate of expected returns. Similar as Mean-variance optimization model, assets weights w are rebalanced in a cycle of 20 days. Consider the following Black-Litterman formula:

$$E(R) = [(\tau\Sigma)^{-1} + P^T\Omega^{-1}P]^{-1}[(\tau\Sigma)^{-1}\Pi + P^T\Omega^{-1}Q],$$

where
$E(R)$ is a $N \times 1$ vector of expected returns, where N is the number of assets
Q is a $K \times 1$ vector of view
P is the $K \times N$ picking matrix which maps views to the universe of assets
Ω is the $K \times K$ uncertainty matrix of views
Π is the $N \times 1$ vector of prior expected returns
Σ is the $N \times N$ covariance matrix of asset returns
τ is a scalar tuning constant.

Optimal w are calculated using *'PyPortfolioOpt'* library in the function named 'blacklitter'. This library can implement a variety of portfolio optimization techniques, such as the traditional efficient frontier methods and Black-Litterman allocation, as well as shrinkage and Hierarchical Risk Parity. The basic structure of 'blacklitter' can be summarized as:

Step 1: Set absolute views. We derive the views by observing the historical returns. For example, from Figure 2, we expected '^GSPC' will return 1% and '^DJI' will return 2% in the future.

Step 2: Set the market implied risk-aversion parameter using formular

$$Delta = \frac{R - R_f}{\sigma^2},$$

where
R is the market return
R_f is the risk-free rate
σ^2 is the volatility of market.

For example, if we expect market has excess returns of 10% a year with 5% variance, the risk-aversion parameter is 2. Here we set the risk-aversion parameter as 0.7.

Step 3: Compute the corresponding optimal w.

In-sample and out-sample results of portfolio returns versus time by Black-Litterman (BL) Model are displayed separately in Figure 15 and Figure 16 (shown in green lines).

6.3 Back-tested System

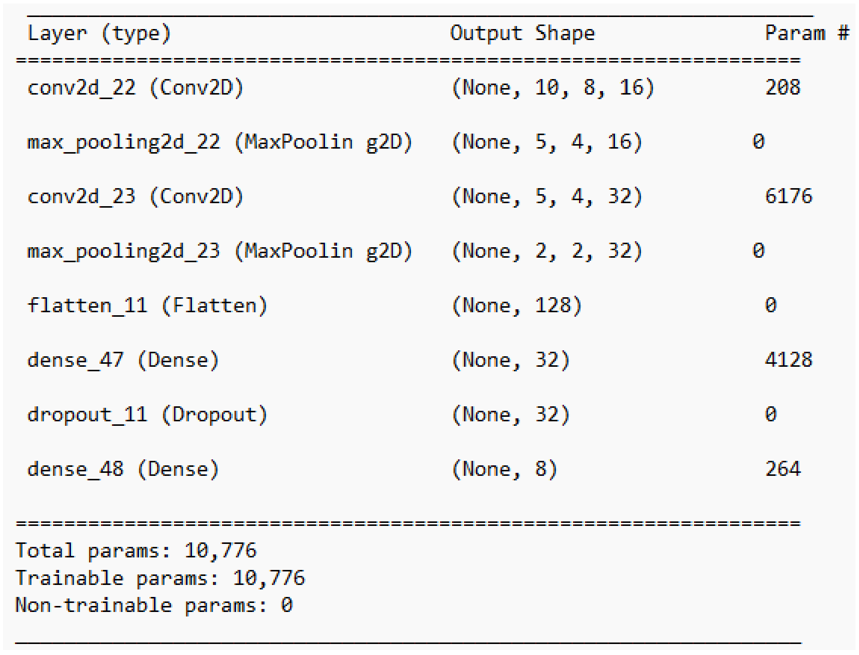
The whole train-test trading process is briefly shown by a flowchart in Figure 3. In order to obtain well balance between portfolio return and risk control, we will calculate its volatility, Sharpe ratios etc. as our risk metrics. We will compare the result with our benchmark, the equal weighted buy-and-hold strategy.

7 Experimental Analysis

7.1 Convolutional Neural Network

Convolutional neural network (CNN) is a class of artificial neural network. CNNs are regularized versions of multilayer perceptrons. Multilayer perceptrons usually mean fully connected networks, which is that each neuron in one layer is connected to all neurons in the next layer. This can lead to overfitting data easily. A convolutional neural network consists of an input layer, hidden layers, and an output layer. In any feed-forward neural network, any middle layers are called hidden layers because their inputs and outputs are masked by the activation function and final convolution.

We use a CNN with two hidden convolutional layers. First, one takes the (10 x 10 x 16) price tensor as input and performs convolution 2D of size (5 x 4 x 16) to create a hidden layer, and maximum pooling layer to filter out the maximum status, which is followed by another convolution 2D and maximum pooling that feeds into the portfolio vector memory.



```
 Layer (type)                      Output Shape              Param #
=================================================================
 conv2d_22 (Conv2D)                (None, 10, 8, 16)         208

 max_pooling2d_22 (MaxPoolin g2D)  (None, 5, 4, 16)         0

 conv2d_23 (Conv2D)                (None, 5, 4, 32)          6176

 max_pooling2d_23 (MaxPoolin g2D)  (None, 2, 2, 32)         0

 flatten_11 (Flatten)              (None, 128)               0

 dense_47 (Dense)                  (None, 32)                4128

 dropout_11 (Dropout)              (None, 32)                0

 dense_48 (Dense)                  (None, 8)                 264

=================================================================
Total params: 10,776
Trainable params: 10,776
Non-trainable params: 0
```

Figure 5: CNN Model Structure

1. NUMBER OF NODES: there is not a uniform number on how many nodes should be used and it depends on specific problem. While increasing the number of nodes (with regularization techniques) within a layer can increase accuracy, fewer number of nodes may cause underfitting. We use 16, 32 and 32 for number of nodes of each layer.

2. DROPOUT: every layer should be accompanied by a dropout layer. The dropout layer helps avoid overfitting in training by bypassing randomly selected neurons, which can reduce the sensitivity to specific weights of the individual neurons. The dropout value should be kept small and 20% is a widely accepted number.

3. ACTIVATION FUNCTION: activation functions are what defines the output of a node should be passed to next layer or not. We use the activation functions to include no-linearity to models and allowing models to learn non-linear prediction boundaries. Additionally, choice of activation layer depends on individual application and we use ReLU in our case. ReLU stands for the rectified linear activation function, which is a piecewise linear function that will output the input directly if it is positive, otherwise, it will output zero.

4. LEARNING RATE: the learning rate defines how quickly the network updates it parameters. A higher learning rate accelerates the learning by the model may not converge while a lower rate will slow down the learning but the model can converge smoothly. A decaying learning rate is preferred. Our learning rate range is [0.0005, 0.05]

5. MOMENTUM: momentum allows the accumulation of the gradients of the past steps to determine the direction to go with, instead of using the gradient of only the current step to guide the search. Typically, the value is between 0.5 to 0.9.

6. NUMBER OF EPOCHS: the number of epochs sets how many complete iterations of the dataset is to be run. The number of epochs should be increased until the validation accuracy starts to decrease. Additionally, we can also use early stopping method to specify a large number of training epochs firstly and stop training once the model performance stops improving.

7. BATCH SIZE: the batch size hyperparameter defines the number of samples to work on before the internal parameters of the model are updated. Large sizes make large gradient steps compared to smaller ones for the same number of samples “seen”. The default value is 32 and 64, 128 and 256 are also accepted. Our optimizer range is [16, 512].

8. OPTIMIZER: to better handle the complex training dynamics of recurrent neural networks, we choose between Adam and SGD in our case.

The investing weight of Hang Seng Index (HIS) has been keeping as 1 through the entire training process. This indicates that agent invests all of his/her money into this single index, which is unrealistic. Therefore, we want to look more detailed structure to analyze this scenario. By printing each layer of CNN and weights of each node, we guess that during model training process, optimizer leads network falling into local minimum.

Hyperparameters	Range of Tuning	Best Choice
Network Type	Convolutional Neural Network	
Training Date Range	'2002-01-01' to '2016-04-04'	
Validate Date Range	'2016-04-5' to '2019-06-20'	
Testing Date Range	'2019-06-21' to '2022-06-30'	
Layers	2	
Samples	4508	
Learning Rate	[0.0005, 0.05]	0.005834
Batch Size	[16, 512]	252
Optimizer	Adam/SGD	Adam

Table 1: Configuration of the CNN Network

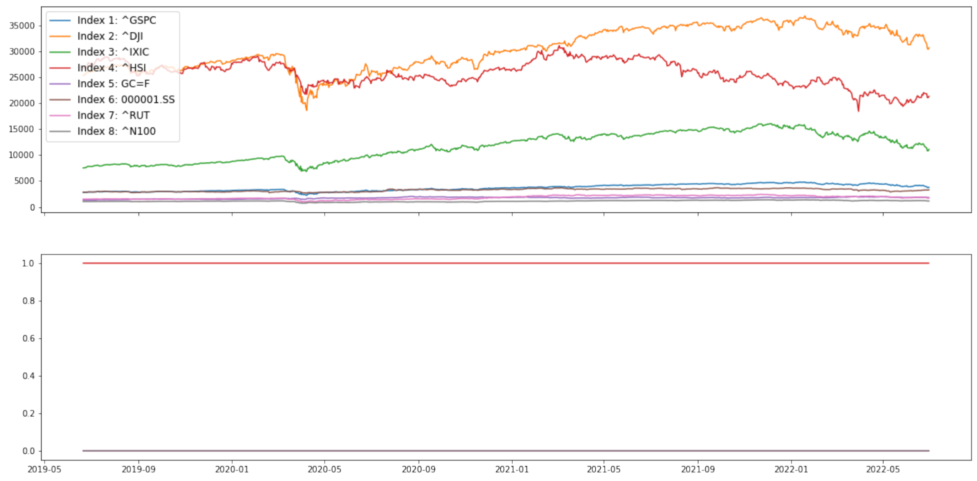


Figure 6: Investment Allocation Predicted by CNN

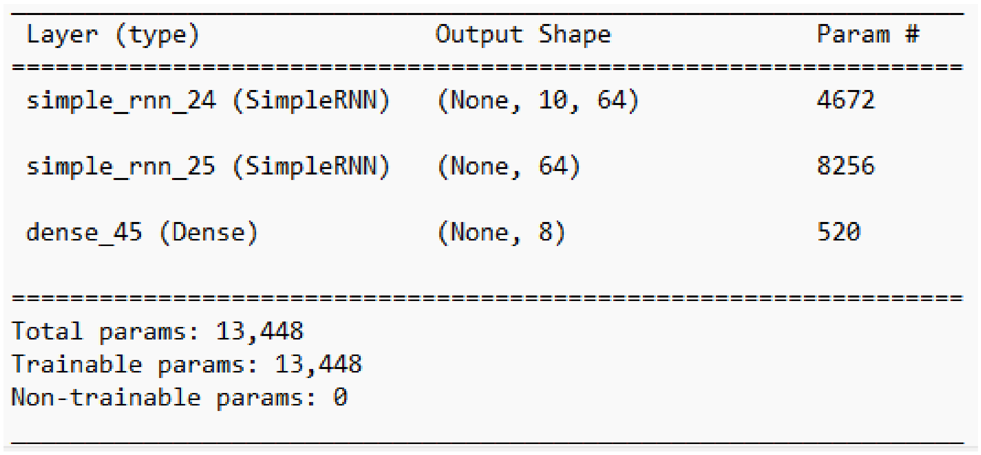
7.2 Recurrent Neural Network

In traditional neural network models, the input layer is only connected to the hidden layer and then to the output layer, when the nodes between each layer are unconnected. Derived from those, Recurrent Neural Network (RNN), is a class of artificial neural networks where connections between nodes form a directed or undirected graph along a temporal sequence. In this case, the current output of a sequence is related to the previous output. The network will memorize all previous information and apply it to the calculation of the current output, which means the nodes between the hidden layers are no longer unconnected but connected. This enables RNN models be extremely effective for processing data with sequence characteristics, while it can exploit the time series information or semantic information hidden in data.

Based on the predefined RNN layer function provided in the tensor-flow package, we created our initial RNN model with two equally weighted simple RNN layers. The model contains 13,448 total parameters and all of them are trainable, as shown in Figure 7. The model takes input with shape of (#training length, 10, 8), and will return a tensor vector with shape (#testing length, 8), indicating the investment allocation of each index in percentage.

For this RNN model, our ranges of tuning the parameters are similar to the CNN one, and the best choices after tuning are displayed in Table 2.

1. From the top picture in Figure 8, we saw that due to different constituents and computation methods of different indices, the 8 indices display quite disparate prices; among them,



```
Layer (type)                 Output Shape              Param #
=================================================================
simple_rnn_24 (SimpleRNN)    (None, 10, 64)            4672

simple_rnn_25 (SimpleRNN)    (None, 64)                8256

dense_45 (Dense)             (None, 8)                 520

=================================================================
Total params: 13,448
Trainable params: 13,448
Non-trainable params: 0
_________________________________________________________________
```

Figure 7: RNN Model Structure

Hyperparameters	Range of Tuning	Best Choice
Network Type	Recurrent Neural Network	
Training Date Range	’2002-01-01’ to ’2016-04-04’	
Validate Date Range	’2016-04-5’ to ’2019-06-20’	
Testing Date Range	’2019-06-21’ to ’2022-06-30’	
Layers	2	
Samples	4508	
Learning Rate	[0.0005, 0.05]	0.009741
Batch Size	[16, 512]	57
Optimizer	Adam/SGD	

Table 2: Configuration of the RNN Network

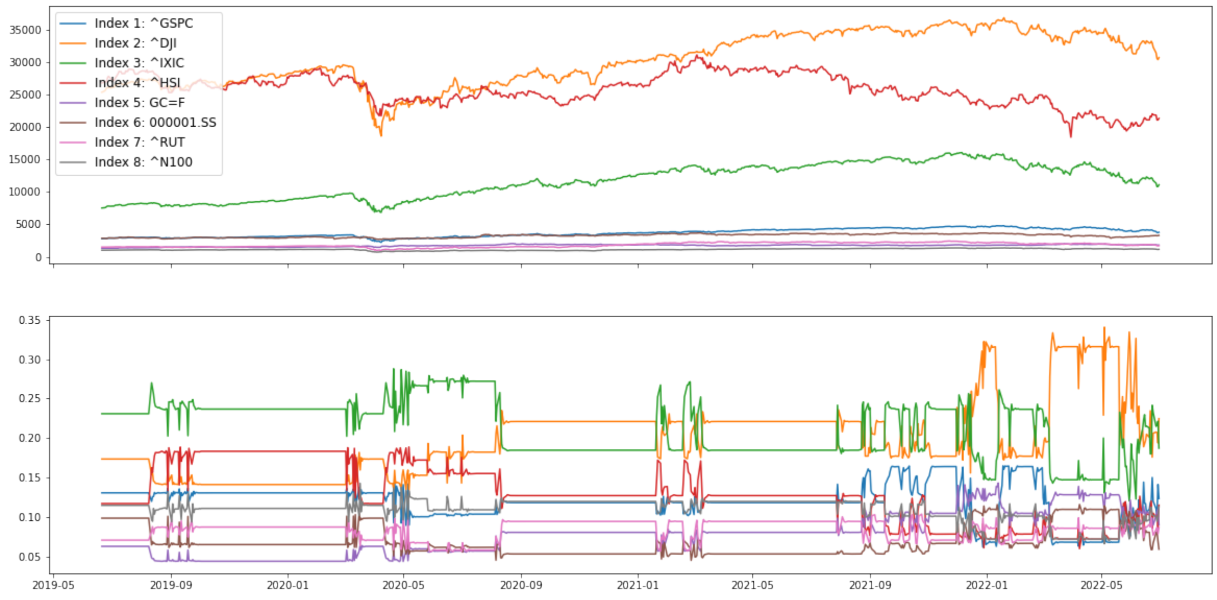


Figure 8: Investment Allocation Predicted by RNN

Dow Jones Industrial Average (DJI) and Hang Seng Index (HSI) ranked top 1 and 2, while NASDAQ Composite Index (IXIC) got the 3rd place.

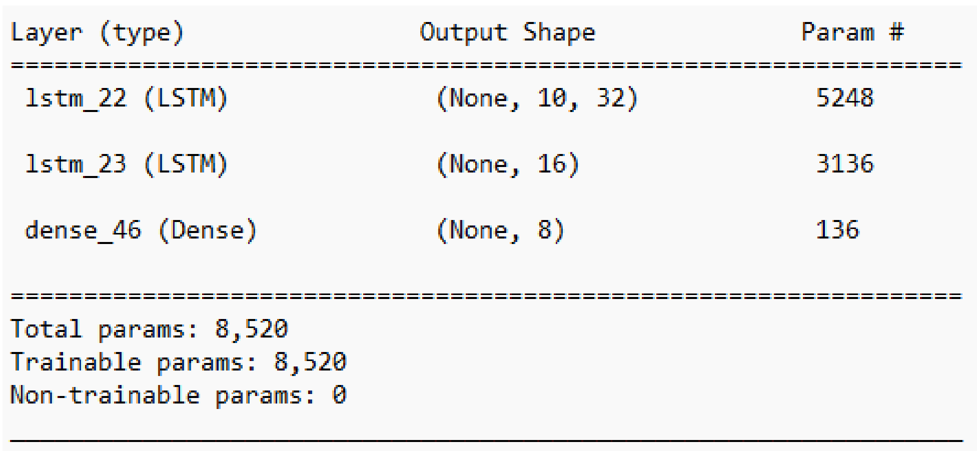
2. Despite the diverse prices, the bottom figure shows that the IXIC and DJI index almost got the highest weights through all the time, and even demonstrated inverse directions in trend. One of the reason might be that the DJI index represents the 30 largest and most influential companies, while IXIC also includes many speculative companies with small market capitalization and places extra emphasis on the technology industry. For instance, in 2022 spring, the RNN-trained trading agent allocated much more weights to the DJI index while reducing IXIC weights in the meantime, suggesting it considered the traditional market as more profitable with larger arbitrage space compared with emerging technology industry. Besides, GSPC and HSI ranked 3 and 4 in composition, since their performance represents a gauge of the health of the U.S. and Hong Kong economy.

3. By the plunge of all the index prices around April 2020, we can deduce that the Coronavirus Disease 2019 (COVID-19) has brought huge influences on the financial markets around the globe, and in later recovery stage, the index prices increased slowly and there was little fluctuation in the wealth allocation, until late 2021.

7.3 Long Short Term Memory Neural Network

A long short term memory network is a recurrent neural network with feedback loop, which is useful for processing both single data points and sequences of data and can be applied in natural language processing and speech research. A standard LSTM unit is composed of a cell, an input gate, an output gate, and a forget gate, which supports read, write and reset operations for the cells. The cell remembers values over arbitrary time intervals, and flow of information into and out of the cell are regulated by the three gates. LSTMs were designed to solve the vanishing and exploding gradient problems typical in RNNs, and its advantages of relative insensitivity to gap length making them suitable for time series processing.

We use a two-layer simple long short term memory network with 10 steps. We also add a choice of range for hyperparameters for the following step of tuning model. For example, we get the best Learning Rate of 0.00132 from 0.0005 to 0.05, the best Batch Size of 42 from 16 to 512 after tuning the LSTM model. The output from this network goes to the portfolio vector memory.



```
Layer (type)                Output Shape              Param #
=================================================================
 lstm_22 (LSTM)              (None, 10, 32)            5248

 lstm_23 (LSTM)              (None, 16)                3136

 dense_46 (Dense)            (None, 8)                 136

=================================================================
Total params: 8,520
Trainable params: 8,520
Non-trainable params: 0
_________________________________________________________________
```

Figure 9: LSTM Model Structure

Hyperparameters	Range of Tuning	Best Choice
Network Type	Long Short Term Memory Neural Network	
Training Date Range	'2002-01-01' to '2016-04-04'	
Validate Date Range	'2016-04-5' to '2019-06-20'	
Testing Date Range	'2019-06-21' to '2022-06-30'	
Layers	2	
Samples	4508	
Learning Rate	[0.0005, 0.05]	0.00132
Batch Size	[16, 512]	42
Optimizer	Adam/SGD	

Table 3: Configuration of the LSTM Network

Compared with the RNN model, the index weights of the LSTM model are distributed more evenly and the portfolio of the LSTM model is more diversified. And from the two weight plots, we can see that the two models adjust the weights at about the same time.

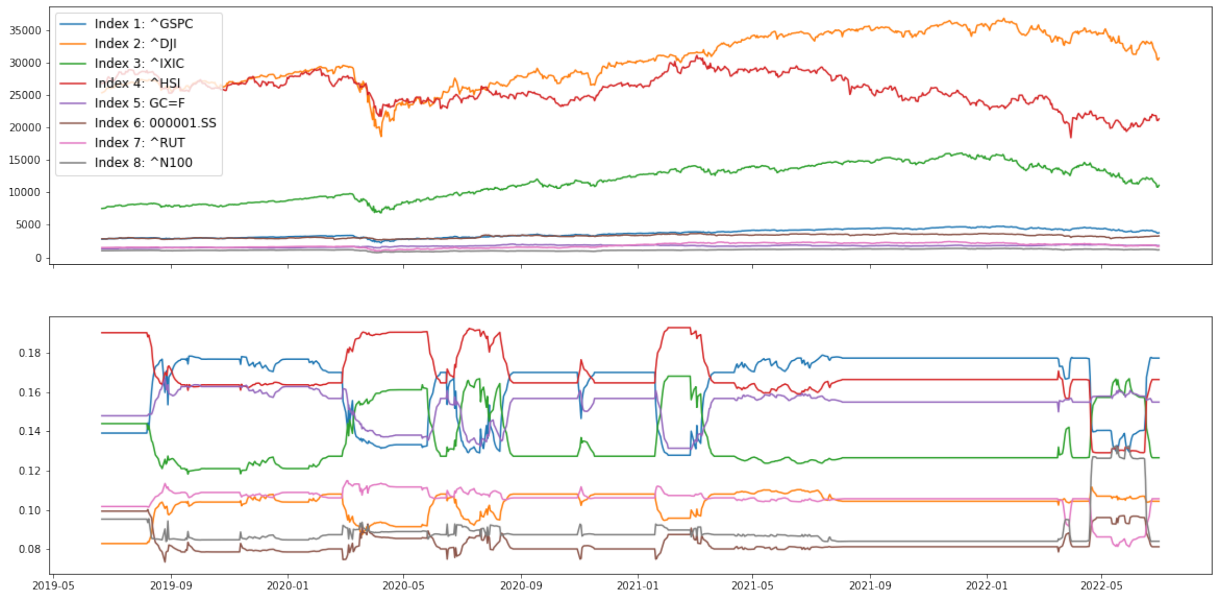


Figure 10: Investment Allocation Predicted by LSTM

1. We can see that the weights of the Hang Seng Index (HIS) and S&P 500 (GSPS) indices continue to be high, which means the trading agent traded by the LSTM model might be optimistic about the Hong Kong market and U.S. market, expecting more profitability and arbitraging space in them.

2. From April 2020 to June 2020 and January 2021 to March 2021, the weights of HSI, 000001.SS, IXIC, N100 indices became higher and the weights of GSPS, DJI became lower. It might derive from the massive outbreak of COVID-19 in the United States and the recovery of the epidemic in China. The coronavirus pandemic has hit the economy hard, forcing many industries to shut down, but some tech industries have not suffered much because of the flexibility of working methods. Thus the weights of IXIC and N100 indices have risen during this period.

3. The weights of HSI, GSPS and RUT indices decreased while the weights of N100 and 000001.SS increased in April 2022 , showing that Shanghai market and European market perform well and other markets are weak.

7.4 Comparison

The DRL model results would firstly be compared with each other, and then be compared to traditional methodologies and benchmarks.

7.4.1 Neural Networks (NNs)

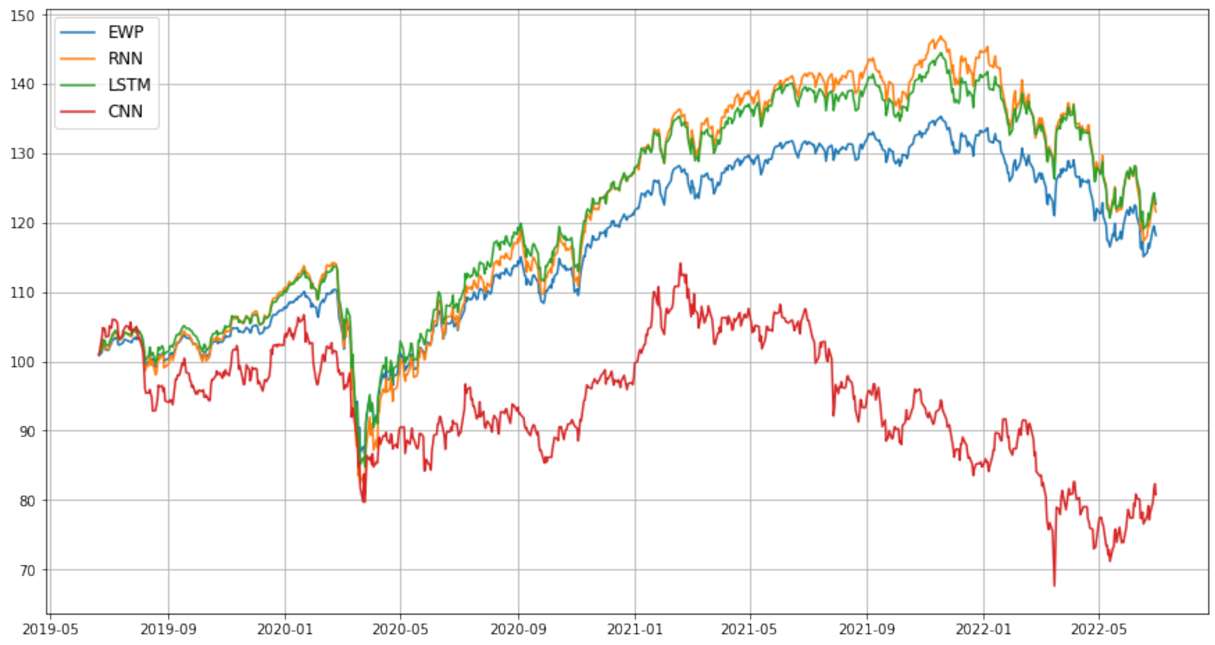


Figure 11: Portfolio Performance (Cumulative Return %) with Different Neutral Networks

From Figure 11 we can note that portfolio performance by using Convolutional Neural Network (denoted as ‘CNN’ in the red line) is the worst among the four neutral network methods. Portfolios constructed based on Long Short-Term Memory network (denoted as ‘LSTM’ in green line) and Recurrent Neural Network (denoted as ’RNN’ in orange line) are slightly better than the equally weighted portfolio (denoted as ‘EWP’ in blue line).

	Sharpe Ratio	Sortino Ratios	Maximum Drawdown	Cumulative Return
CNN	-13.5241	-18.9090	23.63%	-19.21%
RNN	22.6733	26.6112	40.58%	22.22%
LSTM	22.6983	26.1627	39.05%	23.31%
Equal weights	21.6770	25.1535	21.93%	18.18%

Table 4: Performance without Normalization

Table 4 shows the Sharpe Ratio, Sortino Ratios, Maximum Drawdown, and Cumulative Return for the four neural network constructed portfolios. The Sharpe Ratio of ‘CNN’ portfolio is negative, while the other three methods generate positive Sharpe Ratio. This also confirms that 'CNN' portfolio does not perform as well as the other three portfolios.

Since the performance is not good enough, especially the CNN, we would like to look for some other operations to refine the models.

7.4.2 Non-normalization Vs Normalization

Here we consider the comparison of non-normalization and normalization on training data. By adding the loss terms to the sequence and then dividing them by the maximum sequence length, the loss is normalized. We do this to average out the loss across the batch which will facilitate the reuse of the hyperparameters across experiments.

According to Figure 12, we can observe that there is no significant gap between the portfolio constructed by using CNN and the other three methods. Moreover, comparing this Figure 12 with Figure 11, returned by models without normalization, and Figure 13 with Figure 6, the portfolio performance given by CNN model has been improved outstandingly.

Similar as previous non-normalization result, 'LSTM' portfolio and 'RNN' portfolio perform slightly better than equally weighted portfolio. Sharpe Ratio, Sortino Ratios, Maximum Drawdown, and Cumulative Return for the four neural network constructed portfolios after normalizing the loss are shown in Table 5. One obvious difference between Table 4 and Table 5 is that the sharpe ratio of 'CNN' portfolio changes from negative to positive which indicates that normalization improves the performance of the portfolio to some extent.

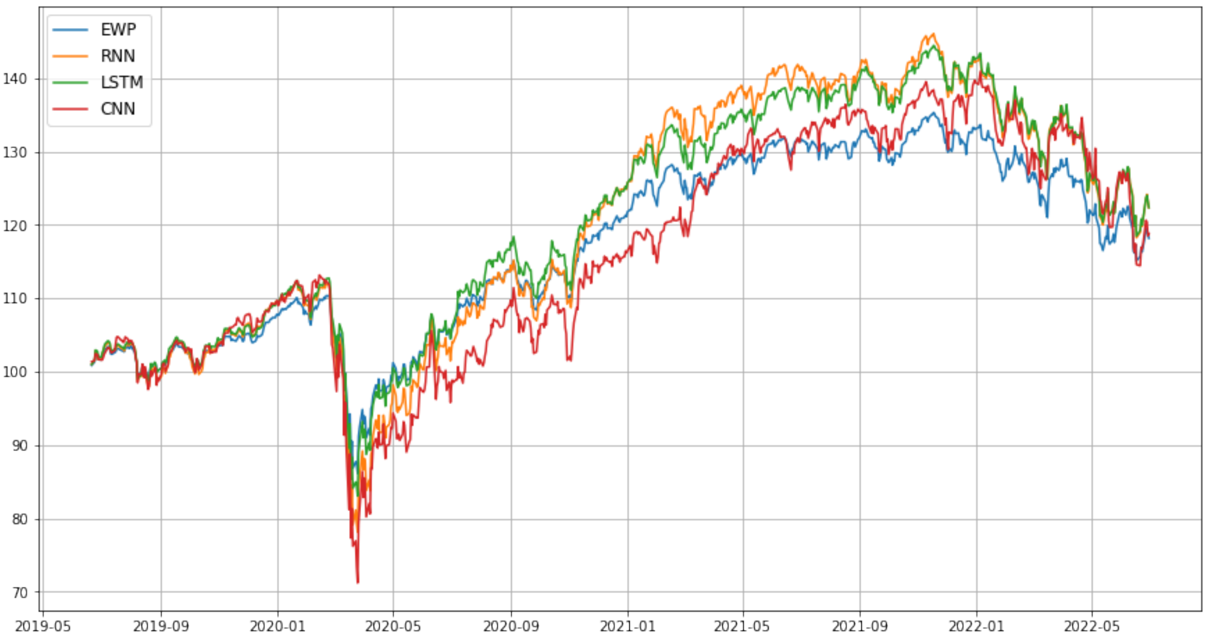


Figure 12: Portfolio Performance (Cumulative Return %) with Different Neutral Networks with Normalization

	Sharpe Ratio	Sortino Ratios	Maximum Drawdown	Cumulative Return
CNN	12.5278	14.0198	27.19%	18.83%
RNN	18.7546	21.5518	34.10%	22.33%
LSTM	21.9296	25.3092	39.04%	22.37%
Equal weights	21.6770	25.1535	21.93%	18.18%

Table 5: Performance after Normalization

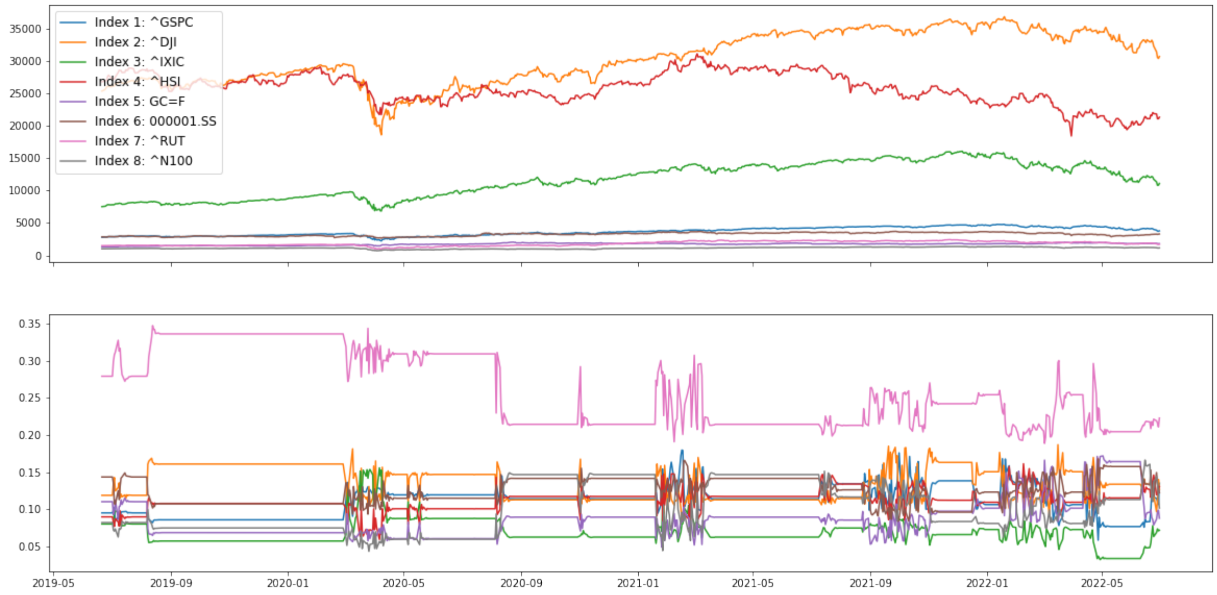


Figure 13: Investment Allocation Predicted by CNN with Normalization

7.4.3 Reward Function: Sharpe Vs Sortino

The Sharpe ratio is one of the most widely used methods for calculating risk-adjusted return. It is calculated as the excess return of the portfolio divided by standard deviation of the portfolio's excess return. Therefore, Sharpe ratio shows how well an equity investment is performing in comparison to a risk-free investment, taking into account the additional risk level involved with holding the equity investment. Sortino ratio can also measure the risk-adjusted return of a portfolio. Unlike Sharpe ratio, the Sortino ratio is a variation of the Sharpe ratio that only factors in downside risk which is the standard deviation of negative portfolio returns. For the previous cases, we train the RL model using shapre ratio as our loss function. To measure the downside risk, we switch to use sortino as our new loss function, and repreat the training process on section 7.1-7.3, to see if there is any improvement.

The results by using Sortino as loss function are shown in Figure 16 and Table 6. Sortino ratio of 'CNN' portfolio (24.529213) is higher than non-normalization (-18.909011) and normalization (14.019843) methods we performed previously. From an overall perspective, the 'CNN' portfolio is the best performer when considering downside risk. From Figure 16, it is clear that 'CNN' portfolio outperforms the other three portfolios after July 2021, and the biggest gap is near February 2022. The overall 'CNN' portfolio return trend is gradually increasing, except for a sudden drop in November 2020, when the cumulative return reached -20%. The return peaked at over 80% in March 2022, but since then the return has gradually declined to around 30% in July 2022.

In summary, best combination for each models is stored in Table 7.

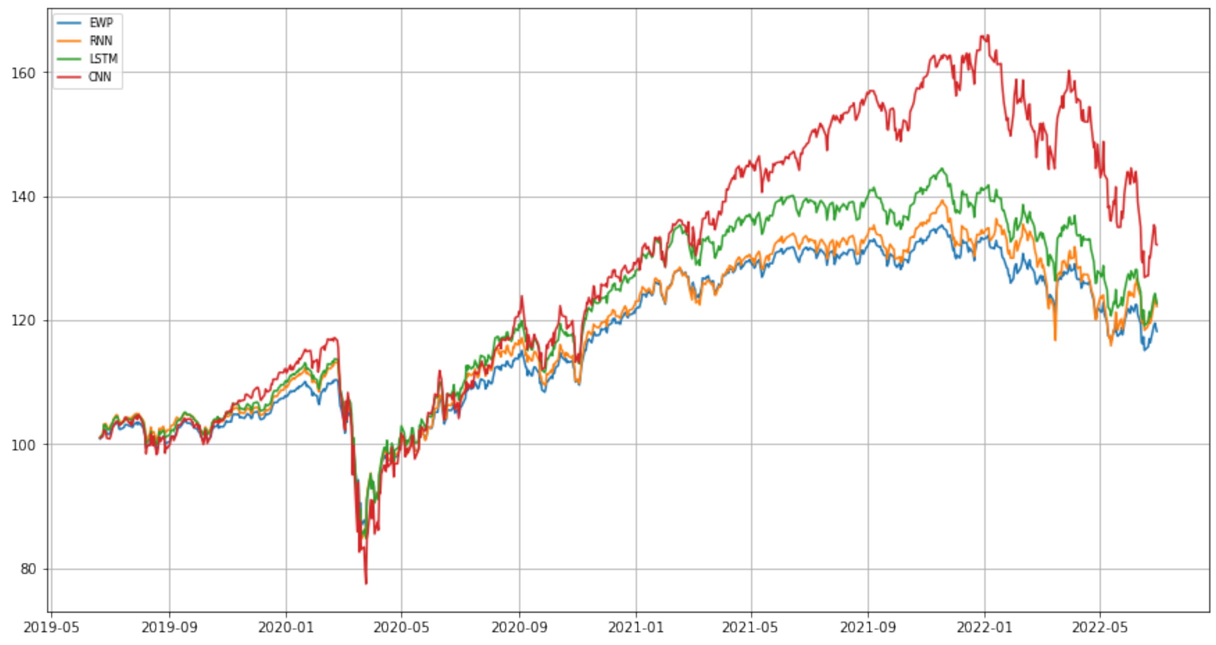


Figure 14: Portfolio Performance (Cumulative Return %) with
Different Neutral Networks Using Sortino Ratio

	Sharpe Ratio	Sortino Ratios	Maximum Drawdown	Cumulative Return
CNN	21.5988	24.5292	33.92%	32.15%
RNN	18.9310	21.6668	29.22%	21.58%
LSTM	22.7916	26.4634	25.44%	22.67%
Equal weights	21.6770	25.1535	21.93%	18.18%

Table 6: Performance Using Sortino as Loss Function

	CNN	RNN	LSTM
Normalization	Yes	Yes	No
Reward Function	Sortino	Sharpe	Sortino
Number of Nodes	16, 32, 32	64, 64	32, 16
Momentum	0.186	0.303	0.300
Learning Rate	0.00575	0.00974	0.0320
Number of Epochs	150	150	150
Batch Size	503	57	387
Optimizer	Adam	Adam	Adam

Table 7: Best Parameters for Each Model

7.4.4 NN Vs Markowitz

In Figure 15, we compare the performance of ‘MVO’ portfolio, ‘BL’ portfolio, and ‘CNN’ portfolio using the in-sample data (training data). Before 2013, the performance of the three models do not differ greatly. From 2013 onwards, MVO’s performance has gradually pulled away from the other two methods. Overall, the yield curve of ‘MVO’ remains constant, with only one large fluctuation around June 2015. ‘BL’ and ‘CNN’ both show an upward trend until June 2015, but ‘CNN’ performs slightly better than ‘BL’. The cumulative returns for each of

	Sharpe Ratio	Sortino Ratios	Maximum Drawdown	Cumulative Return
In-sample Test				
CNN	64.6558	84.0600	53.97%	74.43%
Markowitz	178.4052	178.1897	18.72%	21.04%
BL	128.8718	150.3820	39.24%	46.00%
Out-sample Test				
CNN	21.5988	24.5292	33.92%	32.15%
Markowitz	39.2843	42.4651	17.45%	29.07%
BL	13.9691	16.8286	23.67%	13.48%

Table 8: Performance Comparison

the three models are shown in Table 8, with 'CNN' reaching a maximum of 74.43%. We can also observe that MVO's Sharpe ratio and Sortino ratios are higher than the other two methods, which also shows that MVO's return is stable compared to the other two methods.

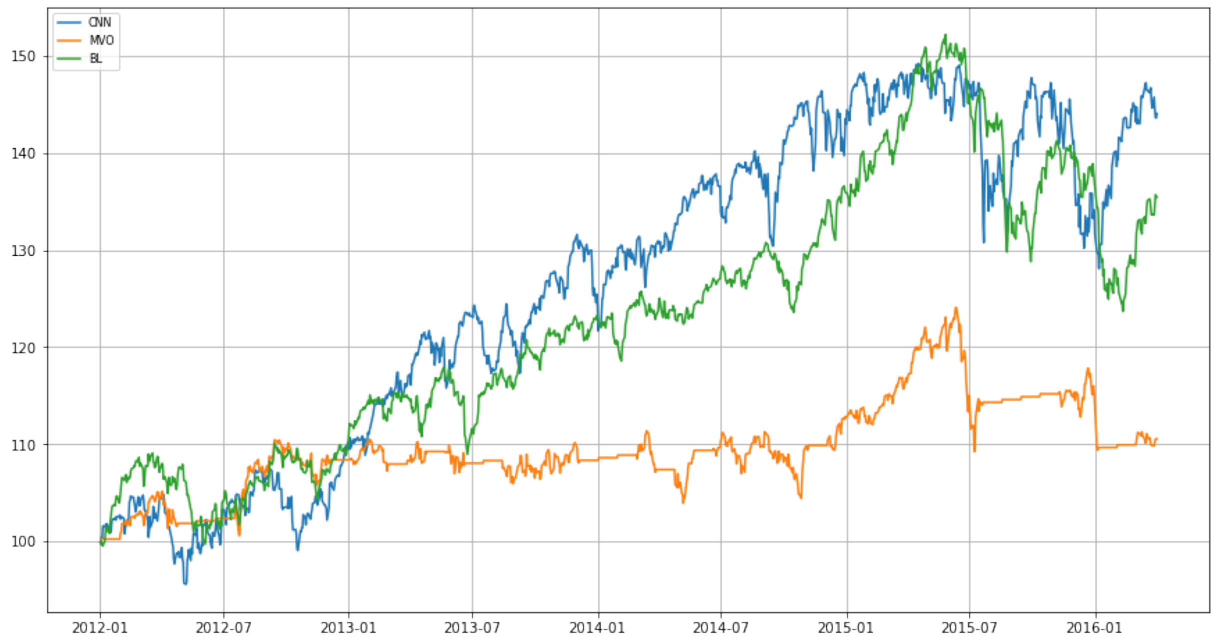


Figure 15: In-sample Portfolio Cumulative Return % of CNN and MVO

On out-sample data, Neutral Network, MVO and BL portfolio have the similar trend and the Neutral Network portfolio has the best performance with the highest cumulative return of 74.43%, while the Black-Litterman portfolio performs worst. As we can see from Figure 16, MVO portfolio has outstanding performance on some specific scenarios. For example, during April 2020, the returns of Neutral Network portfolio and BL portfolio both have a significant drop, which might be derived from the massive outbreak of the COVID-19 in the world, while the MVO model fails to fall into slump in this period. It might be part of the reason for the lowest Maximum Drawdown of the MVO portofolio, making it more stable than the other two models and has the highest Sharpe Ratio and Sortino Ratio.

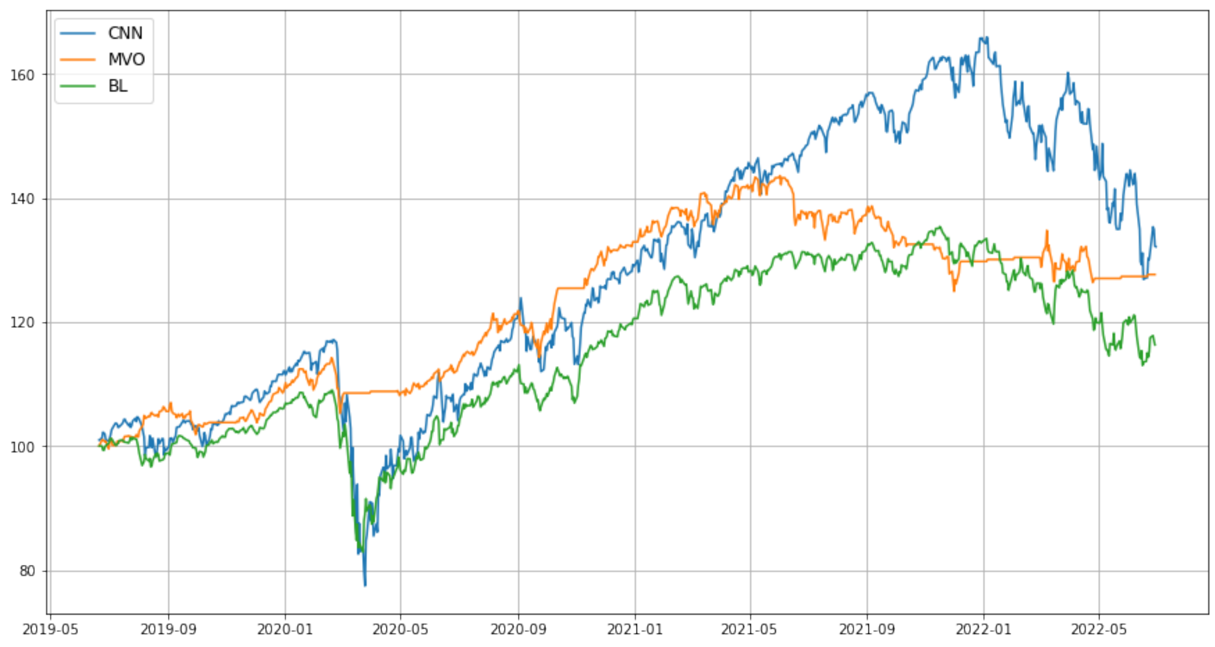


Figure 16: Out-sample Portfolio Cumulative Return % of CNN and MVO

8 Conclusion

In this project, we explored how DRL can be used to solve the optimal portfolio policy problem, by using DNNs in an RL framework, which enables the model to learn from the environment and refine its policy by each step given a target to maximize the portfolio return.

For CNN, the investing weight of Hang Seng Index (HIS) has been keeping as 1 through the entire training process, which is impossible in real life, and one reason might be that optimizer leads network falling into local minimum in model training process.

On the other side, the RNN model has allocated the highest weights to NASDAQ Composite Index (IXIC) and Dow Jones Industrial Average (DJI) index through all the time. The directions of trends in their movements were inverse which implies that the agent regard the traditional industrial market and blooming technology market as complementary to some extent.

In the LSTM model,the weights of all indexes are distributed more evenly, and among them the Hang Seng Index (HIS) and the S&P 500 (GSPS) index got the highest weight all the time, which means the trading agent traded by the LSTM model expect more profitability and arbitraging space in Hong Kong and U.S. market.

By comparing the portfolio performances resulted from the 4 models (CNN, RNN, LSTM, and EWP) using Sharpe ratio as their loss function, we can infer that the RNN and LSTM agents are good at following trends and performs well in chasing for returns, with relatively high cumulative return, Sharpe, and Sortino ratio, indicating their appropriateness in a bullish market. In contrast, the EWP agent is more adaptive to handle risk, with the lowest maximum draw-down, suggesting its capability in a bearish market. After normalization, the CNN model has been improved to a great extent, although it was stil ranked the last with the lowest return and ratios among all methods, which would be a significant point that requires further improvement.

On the other hand, when using Sortino ratio as the loss function, the CNN model was notably enhanced with the highest cumulative return, though it also got the highest maximum draw-down which indicates its instability. Besides, the advantage of RNN model has been weakened while using Sortino ratio.

Under the impact of the COVID-19, the financial markets plummeted around the world, while both the RNN and LSTM model agents displayed large fluctuations in their wealth allocation, and remained relatively stable in the following recovery stage.

Overall, the DRL models seem to outperform the traditional methods, such as the Markowitz and Black-Litterman model, specifically in following trends, as their resulted portfolio performance grows more strikingly than that of classical methods and simultaneously has larger maximum drawdowns as well. In future work, discussion on how we can train the models to deal with emergencies and stock market crashes might be a meaningful direction, exploring ways of stop trading and cutting loss in specific situations.

9 Future Work

For further improvement, we notice that using index prices to construct models will generate scalar issue, which could not be solved by normalization. It might be a better approach to take returns as model input for further study. In the future, we could also separate the forecast process and reinforcement learning process to measure the impact of different components. Besides, it will be inspiring to take emergencies and economic collapse into consideration, controlling damage for specific scenarios. We will also research in the future to explore more models such as Asynchronous Advantage Actor-Critic (A3C) or Twin Delayed DDPG (TD3), and to take more fundamental analysis indicators or ESG factors into consideration. While more sophisticated models and larger data sets are adopted, improvement of efficiency may also be a challenge.

References

[1] O. H. Frederick Spedtsberg, Oliver Bondrop, "Deep reinforcement learning for dynamic asset allocation: An analysis of the use of deep reinforcement learning for solving the optimal portfolio policy problem," https://github.com/OHellum/DRLMarkowitz/blob/main/Deep%20Reinforcement%20Learning%20for%20Dynamic%20Asset%20Allocation.pdf.

[2] S. R. Dasa and S. Varmaa, "Dynamic goals-based wealth management using reinforcement learning," *Journal Of Investment Management*, vol. 18, no. 2, pp. 1–20, 2020.

[3] M. Dixon and I. Halperin, "G-learner and girl: Goal based wealth management with reinforcement learning," *arXiv preprint arXiv:2002.10990*, 2020.

[4] S. R. Das, D. Ostrov, A. Radhakrishnan, and D. Srivastav, "Dynamic portfolio allocation in goals-based wealth management," *Computational Management Science*, vol. 17, no. 4, pp. 613–640, 2020.

[5] ——, "Dynamic optimization for multi-goals wealth management," *Journal of Banking & Finance*, vol. 140, p. 106192, 2022.

[6] M. Denault and J.-G. Simonato, "A note on a dynamic goal-based wealth management problem," *Finance Research Letters*, vol. 46, p. 102404, 2022.

[7] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath, "Deep reinforcement learning: A brief survey," *IEEE Signal Processing Magazine*, vol. 34, no. 6, pp. 26–38, 2017.

[8] Y.-J. Hu and S.-J. Lin, "Deep reinforcement learning for optimizing finance portfolio management," in *2019 Amity International Conference on Artificial Intelligence (AICAI)*. IEEE, 2019, pp. 14–20.

[9] M. Katongo and R. Bhattacharyya, "The use of deep reinforcement learning in tactical asset allocation," *Available at SSRN 3812609*, 2021.

[10] H. Yang, X.-Y. Liu, S. Zhong, and A. Walid, "Deep reinforcement learning for automated stock trading: An ensemble strategy," in *Proceedings of the First ACM International Conference on AI in Finance*, 2020, pp. 1–8.

[11] E. Benhamou, D. Saltiel, J. J. Ohana, J. Atif, and R. Laraki, "Deep reinforcement learning (drl) for portfolio allocation," in *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer, 2020, pp. 527–531.

[12] W. Bessler, H. Opfer, and D. Wolff, "Multi-asset portfolio optimization and out-of-sample performance: an evaluation of black–litterman, mean-variance, and naïve diversification approaches," *The European Journal of Finance*, vol. 23, no. 1, pp. 1–30, 2017. [Online]. Available: https://doi.org/10.1080/1351847X.2014.953699