The use of Deep Reinforcement Learning in Tactical Asset Allocation

Musonda Katongo

Worldquant University
Katongolive@gmail.com

Abstract

The Tactical Asset Allocation (TAA) problem is a problem to accurately capture short to medium term market trends and anomalies in order to allocate the assets in a portfolio so as to optimize its performance by increasing the risk adjusted returns. This project seeks to address the Tactical Asset Allocation problem by employing Deep Reinforcement Learning Algorithms in a Machine Learning Environment as well as employing Neural Network Autoencoders for selection of portfolio assets. This paper presents the implementation of this proposed methodology applied to 30 stocks of the Dow Jones Industrial Average (DJIA). In (1), the Introduction to the project objectives is done with the Problem Description presented in (2). Part (3) presents the literature review of similar studies in the subject area. The methodology used for our implementation is presented in (4) whilst (5) and (6) presents the benchmark portfolios and the DRL portfolios development respectively. The evaluation of the performance of the models is presented in (7) and we present our conclusions and the future works in (8).

Keywords: *Tactical Asset Allocation, Reinforcement Learning, Neural Networks, Markov Decision Process, Asset Portfolio.*

1. Introduction

The major challenge faced with Asset Allocation strategies is estimation errors in forecasting expected returns used in the construction of an investment portfolio. This estimation error tends to be amplified even more in an optimization problem which in most cases causes optimized strategies to be outperformed by naïve strategies which allocates capital equally across all assets (Clark, Feinstein and Simaan, 2020)[3]. For a Tactical Asset Allocation (TAA) strategy, the added challenge is with the transaction costs associated with the frequent rebalancing of a portfolio. Cloutier, Djatej and Kiefer, 2017[5], highlights how trading in and out positions to take advantage of short to medium term market movements tend to undermine the long term objectives of a portfolio by increasing trading costs and thereby reducing returns.

The TAA problem is a problem to accurately capture short to medium term market trends and anomalies in order to allocate the assets in a portfolio so as to optimize its performance by increasing the risk adjusted returns.

The project investigates TAA by application of Deep Reinforcement Learning (DRL) Models to achieve an optimal asset allocation in a portfolio. The portfolio allocation problem is modeled as a Markov Decision Process (MDP) and DRL is used to optimize the allocation strategy.

Three DRL models are explored and we have proposed the use of the Advantage Actor Cretic (A2C) Model because it is stable, cost effective, faster and works better with large batch sizes (Yang, Liu, Zhong and Walid, 2020)[18]. The performance of the DRL models

Copyright ⓒ XXXX SERSC

is compared with the portfolio benchmark index and the naïve equal weight allocation as well as the traditional maximum Sharpe strategy (mean-variance optimization).

2. Problem Description

The problem is described as an optimization problem that seeks to maximize the portfolio risk adjusted returns for a given portfolio asset allocation. We set up our problem with an initial capital investment which is invested in a set of assets. The initial strategic allocation is determined, which in our case is the equal weight allocation, and all of the capital is invested in the set of assets.

At each point in time, the assets are reallocated according to the allocation which will increase the portfolio value. It is proposed to model the asset allocation problem as a Markov Decision Process (MDP). The MDP is solved using a Deep Reinforcement Learning (DRL) model whose environment can be represented as shown in the figure below:

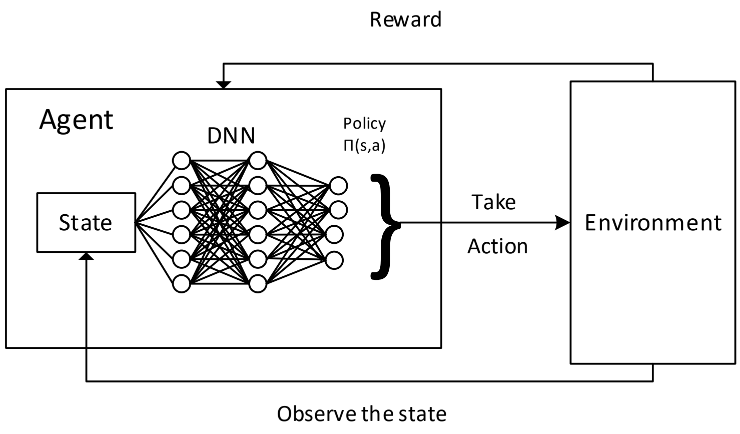


Fig. 1: Basic Representation of a Deep RL Framework

A Deep Neural Network is proposed as the agent which finds an optimal policy that acts on the state of the environment to produce actions that maximize the reward function. The DRL process is modeled as follows:

We will consider a portfolio of n number of assets with a vector of close prices at time t given by V_t. The Normalized vector of the price is given by element-wide division of vector at time t by the vector at time $t-1$:

$$Y_t = V_t \oslash V_{t-1} \dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots. \dots\dots\dots (1)$$

$$Y_t = [1, \frac{V_{1,t}}{V_{1,t-1}}, \frac{V_{2,t}}{V_{2,t-1}}, \dots \frac{V_{n,t}}{V_{n,t-1}}]^T \dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots.. (2)$$

The portfolio vector at time t is determined by the element-wise product of the normalized price vector and the weights at time $t-1$ divide by their dot product.

$$W_t' = \frac{Y_t \odot W_{t-1}}{Y_t \cdot W_{t-1}} \dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots. \dots\dots. (3)$$

The portfolio weight will evolve from W_t' to W_t due to transaction costs. Then the price at time t will be $P_t = \mu P_t'$ where $\mu \in [0,1]$ the transaction costs factor.

The portfolio value at time t is given by:

$$P_t = \mu_t P_{t-1} . Y_t . w_{t-1} \dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots . (4)$$

The portfolio log return is given by:

$$R_t = \ln \frac{P_t}{P_{t-1}} = \ln \mu_t Y_t . w_{t-1} - 1 \ \dots\dots\dots\dots\dots\dots\dots\dots (5)$$

The final portfolio value at time T is given by:

$$P_T = P_0 . \exp\left(\sum_{t=1}^{T} R_t\right) = P_0 \prod_{t=1}^{T} \mu_t Y_t w_{t-1} \dots\dots\dots\dots\dots .. (6)$$

P_0 is the investment at time zero that we start with. The objective function in the optimization problem will be to maximize this cumulative portfolio value at time T.

We define the Reinforcement Learning Configuration of our model as follows:

1. The state (S) includes the High, Low, Close, the covariance matrix of close prices and the identified market financial indicators that are used as inputs;
2. The action (A) is the desired weights allocation. The action at time t will be represented by the weights vectors at time $t-1$: $A_{t-1} = W_{t-1}$. As a result of the action at A_{t-1} and the state inputs, we will get an action or weights allocation at time t which is $A_t = W_t$. Taking into consideration the transaction costs, our model should make a decision on how the weights are rebalanced from W_{t-1} weights W_t. This is done to maximize the accumulative portfolio value.
3. The reward function is given based on the total portfolio value adjusted for the transaction costs. The reward function adjusted for transaction costs is given by:

$$R_t(S_{t-1}, A_{t-1}) = \ln(A_{t-1} \cdot Y_{t-1} - \mu \sum_{i=1}^{n} \left| A_{i,t-1} - W_{i,t-1} \right| \dots\dots (7)$$

   The reward function is fed back to the agent to reinforce the policy of making actions so that future actions are optimized to reach the objective function.

4. **Policy, $\boldsymbol{\pi(s, a)}$**. The policy determines the action to take that maximizes the reward at each state. In our case the policy is made by a Deep Neural network using a Reinforcement Learning algorithm.

In implementation of the model the following assumptions are taken into consideration:

- The market is liquid and it is possible to trade at any time;
- Transactions are small enough not to affect the market price of the assets which means the number of assets traded is small enough compared to the overall market liquidity.
- Transaction costs have been assumed at 0.1% of each trade total value.

3. Literature Review

Tactical Asset Allocation is an active research area with a number of research focusing on optimizing the tactical allocation of assets in a portfolio to take advantage and profit from market movements and anomalies. The main focus of many research works has to do with development of models that can accurately capture market movements and predict the expected future returns so as to rebalance the portfolio optimally based on the current and future market trends.

The optimization problem of TAA can be presented as a Markov Decision Process (MDP) which can be solved using Dynamic Programming (DP) (Neuneier, R., 1996)[13]. In his paper, Neuneier, formalizes this representation and proposes the use of DP or Reinforcement Learning algorithms in solving such an optimization problem. (Yang, Liu, Zhong and Walid, 2020)[18] have proposed a Deep Reinforcement Learning (DRL) Library that uses various DRL algorithms for portfolio construction and stock trading. The methodology in this paper adopts this approach and make use of DRL models based on the Stable Baselines library which is a library for DRL algorithms.

Chakravorty et al., 2018 in their paper, Deep Learning based Global Tactical Asset Allocation, show the use of deep neural networks with macro-economic data in conjunction with price-volume data in a walk-forward setting to do tactical asset allocation[2]. They used macroeconomic indicators and price-volume features to perform tactical asset allocation and optimizing the weights using a custom utility function of a single metric. Further research in this regard can be done by using a utility function which takes into consideration a number of matrices for optimization. Their strategy showed a significant performance improvement across different runs. This demonstrates the advantage of using RL models for TAA as the model leans iteratively and improve the policy for each run.

Obeidat et. al., 2018 used a methodology for TAA based on prediction of future returns using the Long Short Term Memory (LSTM) Neural Network (NN)[15]. The predicted future returns where then used to optimize the asset allocation. As the main challenge with any asset allocation problem is accuracy with which expected returns can be estimated, their model showed better performance compared to the traditional passive approaches. However, the model performed worse in certain market trends. Also, the rebalancing period was fixed on a monthly basis. This can be optimized by use of RL in which we include the optimization period as part of the cost function.

4. Methodology

The methodology proposed for the asset allocation problem follows the following steps:

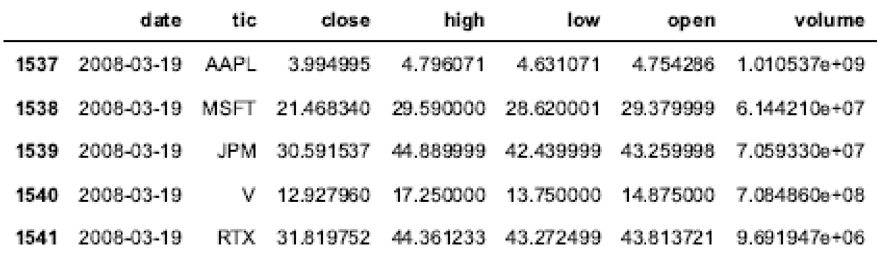
1. **Select the constituent stocks or assets that will form the portfolio.** We have considered the 30 stocks of Dow Jones Industrial Average (DJIA) to form part of our portfolio. Perform stock selection from the total stock constituent. This process ensures that the stocks considered for the portfolio are less volatile. Neural Network Auto Encoders are proposed to perform Stock Selection;
2. **Feature Selection and Data Pre-processing.** Technical indicators are added to the selected stocks to aid in the performance of the agent;
3. **Feature Reduction.** We perform dimensionally reduction of the added features (technical indicators) in order to optimize the learning process of the model. Neural Network Autoencoders are used to perform feature reduction.

4. **Train and Test Data Split.** We split the data into train-data and test-data which is used to train the model and test its performance respectively;
5. **Benchmark Portfolios.** We construct portfolios based on equal weights allocation and mean-variance (maximum Sharpe) allocation which are used to benchmark the DRL model;
6. **Deep Reinforcement Learning Model.** We develop a DRL models to perform asset allocation which are based on the A2C, DDPG and PPO RL algorithms.
7. **Backtesting of the DRL Model.** We test the performance of the DRL models against the mean variance and the equal weights portfolios. The DRL model based on the A2C algorithm is further benchmarked with the DJIA Index.

4.1. Stock Selection

We use the data for the 30 stocks of the DJIA for the period from 19 March 2008 to 01 January 2020 giving us a total of 3,221 data points. The table below shows the head of the downloaded data frame:

Table 1: Stocks Data Frame



	date	tic	close	high	low	open	volume
1537	2008-03-19	AAPL	3.994995	4.796071	4.631071	4.754286	1.010537e+09
1538	2008-03-19	MSFT	21.468340	29.590000	28.620001	29.379999	6.144210e+07
1539	2008-03-19	JPM	30.591537	44.889999	42.439999	43.259998	7.059330e+07
1540	2008-03-19	V	12.927960	17.250000	13.750000	14.875000	7.084860e+08
1541	2008-03-19	RTX	31.819752	44.361233	43.272499	43.813721	9.691947e+06

Below is a list of all the stocks in the DJIA:

```
Index(['date', 'WBA', 'TRV', 'MMM', 'RTX', 'DD', 'KO', 'V', 'XOM', 'CAT', 'PG',
       'AXP', 'WMT', 'MSFT', 'VZ', 'JPM', 'UNH', 'DIS', 'HD', 'AAPL', 'CVX',
       'JNJ', 'CSCO', 'PFE', 'IBM', 'MRK', 'INTC', 'MCD', 'BA', 'NKE', 'GS'],
      dtype='object')
```

From the 30 stocks, 20 with lower volatility are selected using Autoencoders. We construct an Autoencoder with the total number of stocks in our input layer, we have an encoder layer with dimension 5 and a decoder layer with dimension 30. The stock data is reconstructed by passing it through the Autoencoder and then the reconstruction error of each stock is calculated. We then pick the 20 stocks with the lowest reconstruction errors. Below is the summary of the Autoencoder model:

```
-------------------------Define autoencoder model
Model: "functional_11"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_6 (InputLayer)         [(None, 30)]              0
_________________________________________________________________
Encoder_Input (Dense)        (None, 5)                 155
_________________________________________________________________
Decoder_Input (Dense)        (None, 30)                180
_________________________________________________________________
Decoder_Activation_function  (None, 30)                0
=================================================================
Total params: 335
Trainable params: 335
```

```
Non-trainable params: 0
```

Fig. 2: Model Summary for Stock Sellection

The following is the list of the selected stocks:

```
Index(['JNJ', 'PG', 'MMM', 'KO', 'IBM', 'VZ', 'MCD', 'PFE', 'RTX', 'WMT',
       'MRK', 'V', 'DIS', 'MSFT', 'XOM', 'HD', 'TRV', 'INTC', 'AXP', 'NKE'],
      dtype='object', name='stock_name')
```

4.2. Feature Selection and Data Pre-processing

The following technical indicators are generated to be used as feature:

1. Volatility Average True Range (ATR)
2. Volatility Bollinger Band Width (BBW)
3. Volume On-balance Volume (OBV
4. Volume Chaikin Money Flow (CMF)
5. Trend Moving Average Convergence Divergence (MACD)
6. Trend Average Directional Index (ADX)
7. Trend Fast Simple Moving Average (SMA)
8. Trend Fast Exponential Moving Average (EMA)
9. Trend Commodity Channel Index (CCI)
10. Momentum Relative Strength Index (RSI)

Dimensionality reduction using unsupervised stacked restricted Autoencoder model is performed to reduce the number of input features from ten to four. Autoencoders consists of two parts which are; the encoder which reduces the dimensionality of the input data in the latent hidden layer and the decoder which reconstructs the data from the hidden layer (Soleymani and Paquet, 2020)[16].

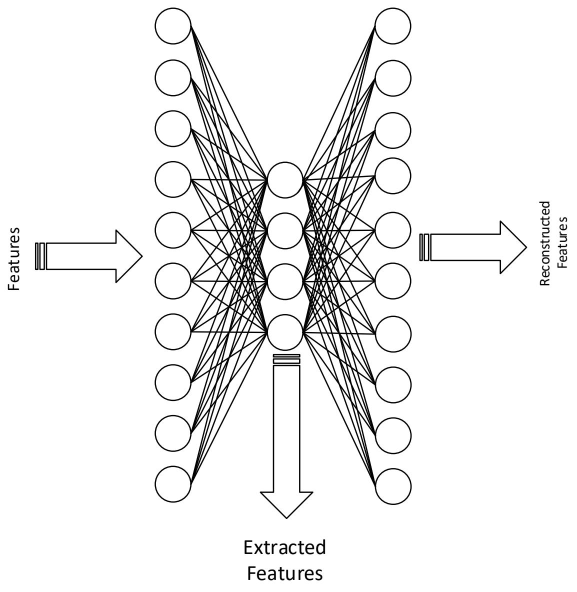


Fig. 2: Structure of the Autoencoder

The following is a graphical representation of an Autoencoder which we propose to use. It consists of an Input Layer, Two Hidden Layers and One Output Layer.

```
Model: "sequential_28"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_50 (LSTM)               (None, 4)                 240
_________________________________________________________________
repeat_vector_22 (RepeatVect (None, 20, 4)             0
_________________________________________________________________
lstm_51 (LSTM)               (None, 20, 100)           42000
_________________________________________________________________
time_distributed_18 (TimeDis (None, 20, 10)            1010
=================================================================
Total params: 43,250
Trainable params: 43,250
Non-trainable params: 0
_________________________________________________________________
```

Fig. 3: Model Summary for Feature Reduction

4.3. Train and Test Data Split

The data for the selected stocks is split into training data and testing data as follows:

```
Training Data:  from 2009-03-20 to 2018-08-23
Testing Data:  from 2018-08-24 to 2020-12-31
```

5. Benchmark Portfolios

Two benchmark portfolios are constructed based on the equal weights allocation and the mean-variance allocation which maximizes the portfolio Sharpe ratio.

1. **Equal Weighted Portfolio.** An equally weighed portfolio of assets with the capital invested equally among the total number of assets is constructed. With n number of assets in a portfolio, the weight for each asset in the portfolio is given by:

$$w_1 = w_2 = \cdots w_n = \frac{1}{n} \dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots . (18)$$

2. **Mean-Variance Optimization Portfolio.** Given n number of assets with expected return vector of R and the covariance matrix of $\sum$ the mean-variance optimization is the solution to the following quadratic problem:

$$\arg_w \min \left(\frac{1}{2}\right) w^T \sum w \dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots (19)$$

Subject to:

$$R^T w \geq R_b \dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots (20)$$

R_b is the acceptable baseline expected rate of return.

6. Deep Reinforcement Learning Models

Three DRL models are implemented using the following three RL algorithms:

1. Deep Deterministic Policy Gradient (DDPG). This algorithm constantly learns the State-action function (Q function) and uses it to determine the best policy;

2. Advantage Actor Critic (A2C) Algorithm. The algorithm consists of the policy which is the actor and acts on the environment. The critic calculates the value function based on a set reward function and this helps the actor to determine the optimal policy.

3. Proximal Policy optimization (PPO). The PPO algorithm is an on-policy algorithm that can be used for discrete or continuous action space environments.

The environment setup within which we implement the algorithms is made up of the stock constituents, the prices, the technical indicators (features) and the covariance matrix derived from the one-year period data at each time step. The DRL agent interacts with the environment in an exploration-exploitation manner. This involves making decisions which will maximize rewards by considering whether to make previous good decisions (exploitation) or to try new decisions with potential for greater rewards (exploitation).

We make the assumptions that there are no short sales and all of our capital is used to invest in the portfolio stocks. The following algorithm gives a summary of the interaction within our environment.

Algorithm 1: Portfolio Allocation Using Deep Reinforcement Learning (DRL)

```
1.  Input:      $s$, state space which includes covariance matrix for stocks and technical indicators
2.  Output:     Final portfolio value
3.  Initialize  $P_0 = \$1{,}000{,}000, w_0 = \left(\frac{1}{m}, \dots, \frac{1}{m}\right)$, $P_0$ is the initial portfolio value and $w_0$ is the initial portfolio weights and $m$ is the number of assets in the portfolio;
4.  for         t = 1, …, $n$ do
5.                  Observe the state $s$ and output portfolio weights vector $w_t$ at time $t$;
6.                  Normalize the weights $w_t$ to sum to 1;
7.                  Calculate stock returns vector $r_t = \left(\frac{v_{1,t}-v_{1,t-1}}{v_{1,t-1}}, \dots, \frac{v_{m,t}-v_{m,t-1}}{v_{m,t-1}}\right)$, $v$ is the closing price;
8.                  Portfolio returns for the period: $w_t^T r_t$;
9.                  Update portfolio value $P_t = P_{t-1} \times (1 + w_t^T r_t)$;
10. End
```

The actions which the agents make are the portfolio weights and the policy is learned at each step that maximizes the portfolio value (reward function).

7. Evaluation and Backtesting of the Models

The weights for the equal weight portfolio and the mean-variance optimization are determined using the train data and these are applied on the test data. The DRL models, however, learns a policy for allocation and the train data and continuously rebalances the portfolio based on the state of the portfolio at every time space.

7.1. Portfolio Weight Allocations

The figure below shows the weight allocation for the equal weights and the maximum Sharpe (mean-variance) portfolios:

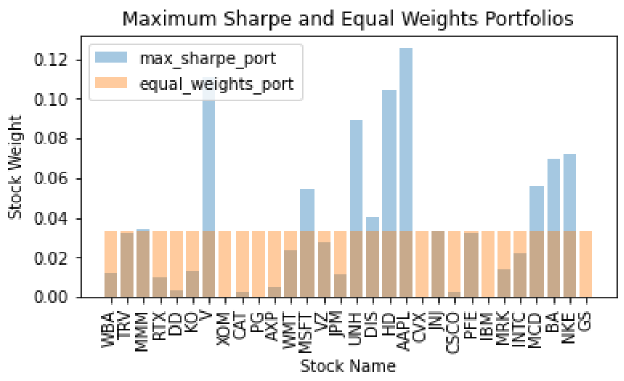
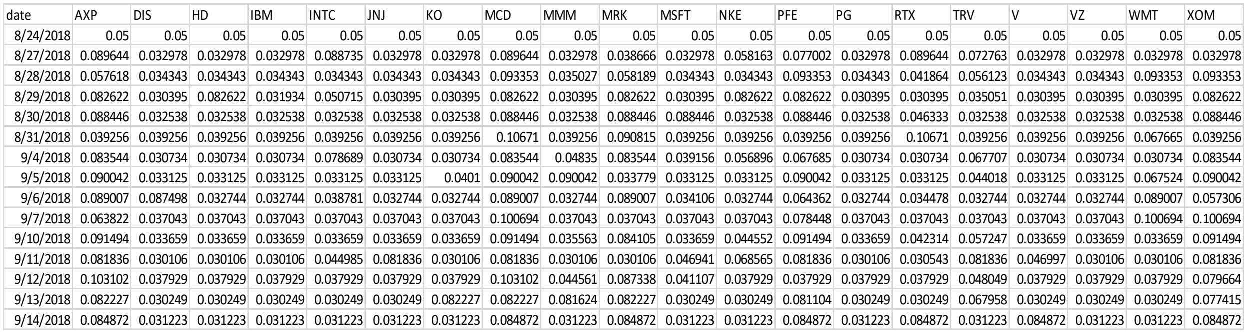


Fig. 3: Portfolio Weight Allocation

The DRL portfolios rebalances the weights at each time step using a policy that would maximize the portfolio cumulative value. Below we show the weights for the a2c model:



date	AXP	DIS	HD	IBM	INTC	JNJ	KO	MCD	MMM	MRK	MSFT	NKE	PFE	PG	RTX	TRV	V	VZ	WMT	XOM
8/24/2018	0.05	0.05	0.05	0.05	0.05	0.05	0.05	0.05	0.05	0.05	0.05	0.05	0.05	0.05	0.05	0.05	0.05	0.05	0.05	0.05
8/27/2018	0.089644	0.032978	0.032978	0.032978	0.088735	0.032978	0.032978	0.089644	0.032978	0.038666	0.032978	0.058163	0.077002	0.032978	0.089644	0.072763	0.032978	0.032978	0.032978	0.032978
8/28/2018	0.057618	0.034343	0.034343	0.034343	0.034343	0.034343	0.034343	0.093353	0.035027	0.058189	0.034343	0.034343	0.093353	0.034343	0.041864	0.056123	0.034343	0.034343	0.093353	0.093353
8/29/2018	0.082622	0.030395	0.082622	0.031934	0.050715	0.030395	0.030395	0.082622	0.030395	0.082622	0.030395	0.082622	0.082622	0.030395	0.030395	0.035051	0.030395	0.030395	0.030395	0.082622
8/30/2018	0.088446	0.032538	0.032538	0.032538	0.032538	0.032538	0.032538	0.088446	0.032538	0.088446	0.088446	0.032538	0.088446	0.032538	0.046333	0.032538	0.032538	0.032538	0.032538	0.088446
8/31/2018	0.039256	0.039256	0.039256	0.039256	0.039256	0.039256	0.039256	0.10671	0.039256	0.090815	0.039256	0.039256	0.039256	0.039256	0.10671	0.039256	0.039256	0.039256	0.067665	0.039256
9/4/2018	0.083544	0.030734	0.030734	0.030734	0.078689	0.030734	0.030734	0.083544	0.04835	0.083544	0.039156	0.056896	0.067685	0.030734	0.030734	0.067707	0.030734	0.030734	0.030734	0.083544
9/5/2018	0.090042	0.033125	0.033125	0.033125	0.033125	0.033125	0.0401	0.090042	0.090042	0.033779	0.033125	0.033125	0.090042	0.033125	0.033125	0.044018	0.033125	0.033125	0.067524	0.090042
9/6/2018	0.089007	0.087498	0.032744	0.032744	0.038781	0.032744	0.032744	0.089007	0.032744	0.089007	0.034106	0.032744	0.064362	0.032744	0.034478	0.032744	0.032744	0.032744	0.089007	0.057306
9/7/2018	0.063822	0.037043	0.037043	0.037043	0.037043	0.037043	0.037043	0.100694	0.037043	0.037043	0.037043	0.037043	0.078448	0.037043	0.037043	0.037043	0.037043	0.037043	0.100694	0.100694
9/10/2018	0.091494	0.033659	0.033659	0.033659	0.033659	0.033659	0.033659	0.091494	0.035563	0.084105	0.033659	0.044552	0.091494	0.033659	0.042314	0.057247	0.033659	0.033659	0.033659	0.091494
9/11/2018	0.081836	0.030106	0.030106	0.030106	0.044985	0.081836	0.030106	0.081836	0.030106	0.030106	0.046941	0.068565	0.081836	0.030106	0.030543	0.081836	0.046997	0.030106	0.030106	0.081836
9/12/2018	0.103102	0.037929	0.037929	0.037929	0.037929	0.037929	0.037929	0.103102	0.044561	0.087338	0.041107	0.037929	0.037929	0.037929	0.037929	0.048049	0.037929	0.037929	0.037929	0.079664
9/13/2018	0.082227	0.030249	0.030249	0.030249	0.030249	0.030249	0.082227	0.082227	0.081624	0.082227	0.030249	0.030249	0.081104	0.030249	0.030249	0.067958	0.030249	0.030249	0.030249	0.077415
9/14/2018	0.084872	0.031223	0.031223	0.031223	0.031223	0.031223	0.031223	0.084872	0.031223	0.084872	0.031223	0.031223	0.084872	0.031223	0.084872	0.031223	0.084872	0.031223	0.031223	0.084872

Fig. 4: Portfolio Weight Allocation for A2C Model

7.2. Backtesting of the Portfolios

The figure below shows the cumulative returns of the DRL models benchmarked against the uniform weight and mean-variance portfolios using the training data:

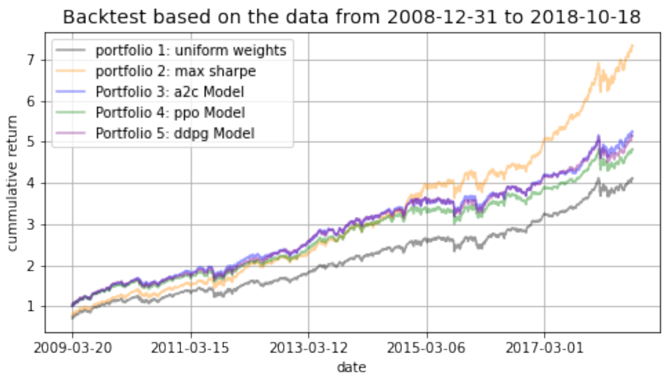


Fig. 4: cumulative Portfolio Returns – Train Data

We can see that the mean-variance (portfolio) outperforms all the other portfolios on the training data. The three DRL models however, performs better than the equal weight allocation portfolio with the A2C model slightly doing better than the other two DRL models.

Figure 5 below shows the benchmarking of the portfolios using the test dataset:

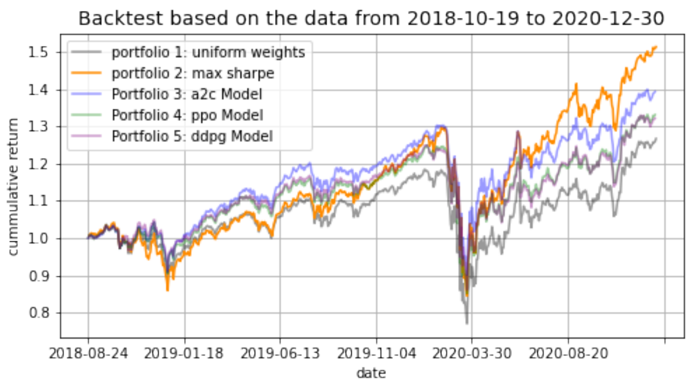
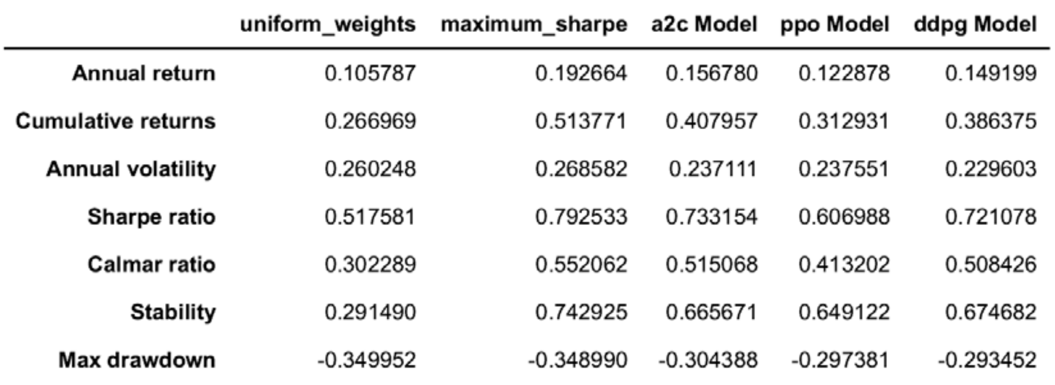


Fig. 5: cumulative Portfolio Returns – Test Data

With the test data we see that the mean-variance portfolio outperforms all the other portfolios during the period starting 2020. The DRL models however, performs better before this period with the A2C model with the overall best performance amongst the three. Overall we can note that there is significantly improved performance of the DRL models on the test data resulting in outperforming the mean-variance during the period before the 2020 year. The underperformance of the DRL models during the period starting 2020 could be attributed to the market fall during the period of the COVID-19 pandemic.

The following table presents the performance matrices of the portfolios with the mean-variance presenting the highest Sharpe ratio followed by the A2C model and the equal weigh allocation having the lowest Sharpe Ratio.

Table 2: Portfolio Performance Matrices

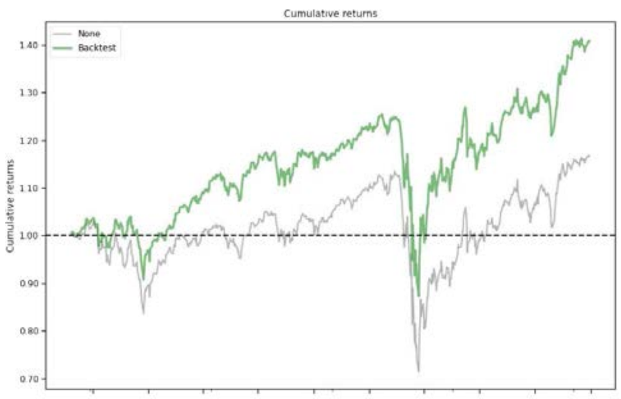


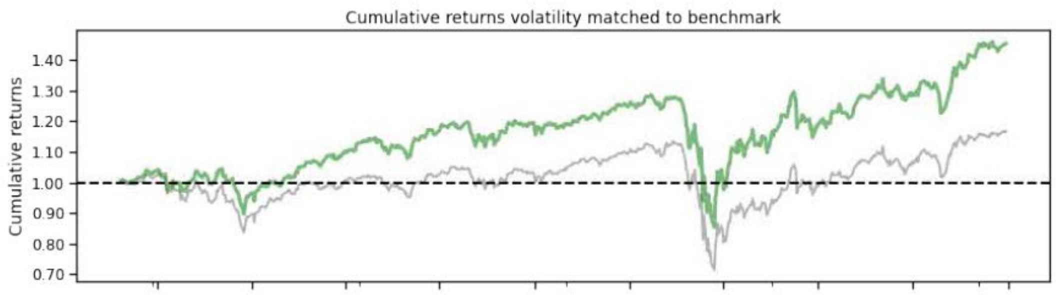
	uniform_weights	maximum_sharpe	a2c Model	ppo Model	ddpg Model
Annual return	0.105787	0.192664	0.156780	0.122878	0.149199
Cumulative returns	0.266969	0.513771	0.407957	0.312931	0.386375
Annual volatility	0.260248	0.268582	0.237111	0.237551	0.229603
Sharpe ratio	0.517581	0.792533	0.733154	0.606988	0.721078
Calmar ratio	0.302289	0.552062	0.515068	0.413202	0.508426
Stability	0.291490	0.742925	0.665671	0.649122	0.674682
Max drawdown	-0.349952	-0.348990	-0.304388	-0.297381	-0.293452

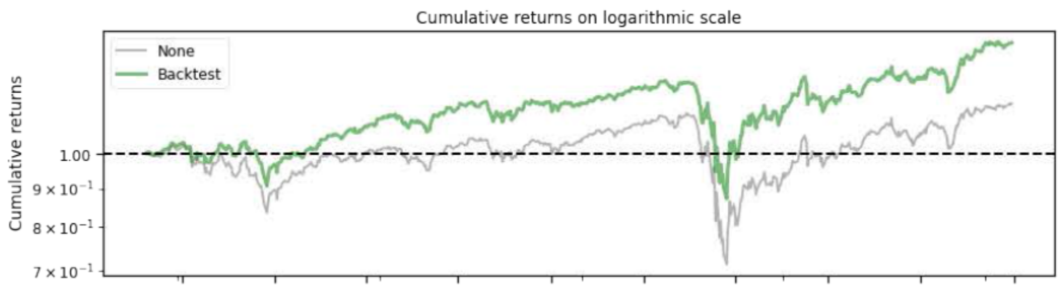
The best performing DRL model is benchmarked with the DJIA index. The following are the performance matrices for the DJIA index:

```
Annual return           0.074890
Cumulative returns      0.185235
Annual volatility       0.265505
Sharpe ratio            0.406106
Calmar ratio            0.201934
Stability               0.151322
Max drawdown           -0.370862
```

We notice that the DRL model has a better Sharpe Ratio compared to the benchmark index. Below are the benchmarking plots of the A2C model against the DJIA index. We note that the model has overall better cumulative returns as compared to the benchmark index. The waste performance is posted during the start of 2020 which has the worst drawdown. This is attributed to the market fall during the period of the COVID-19 pandemic. On average, the portfolio posted positive returns on the test data with the exception of the period of the maximum drawdown as stated above.







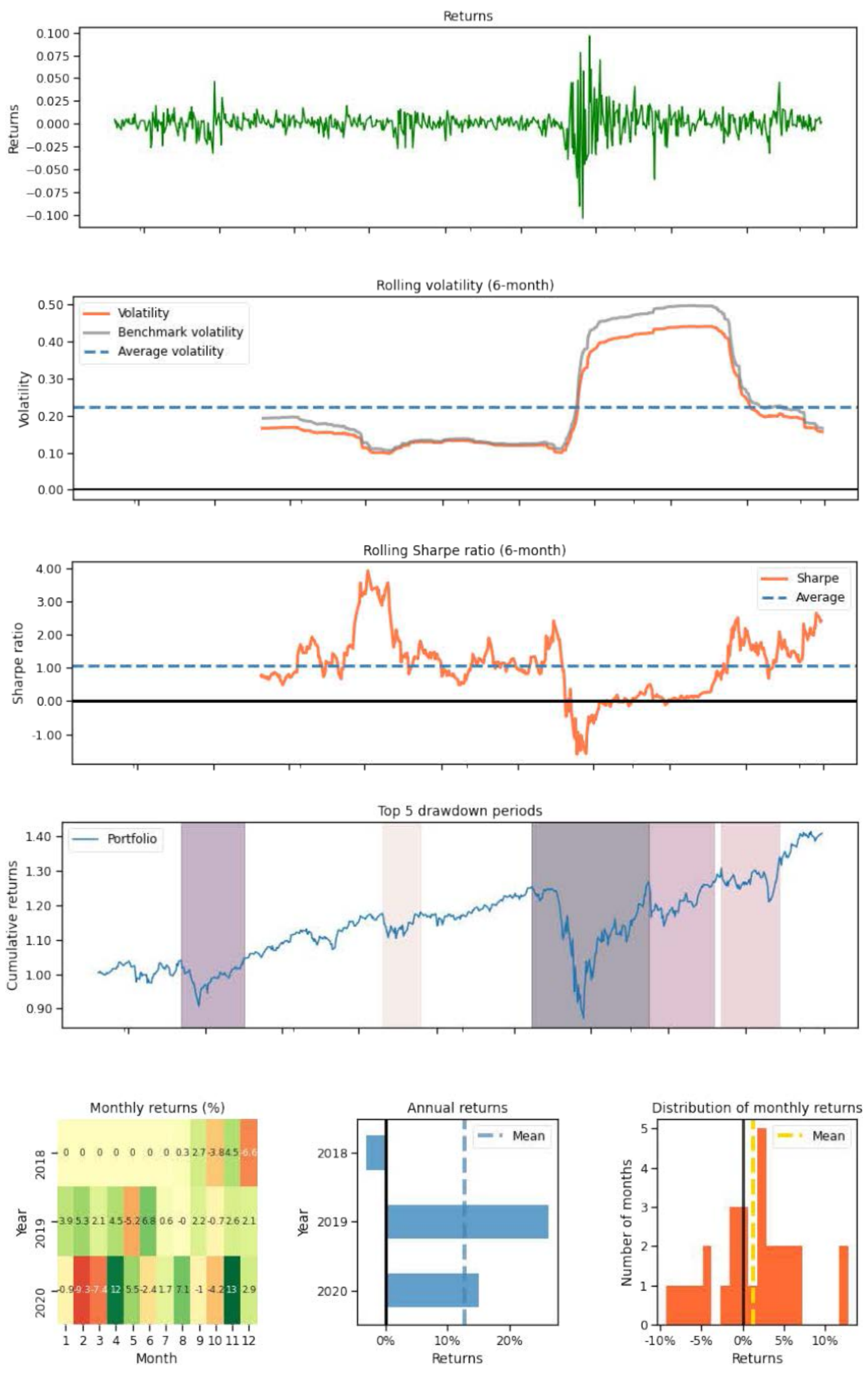
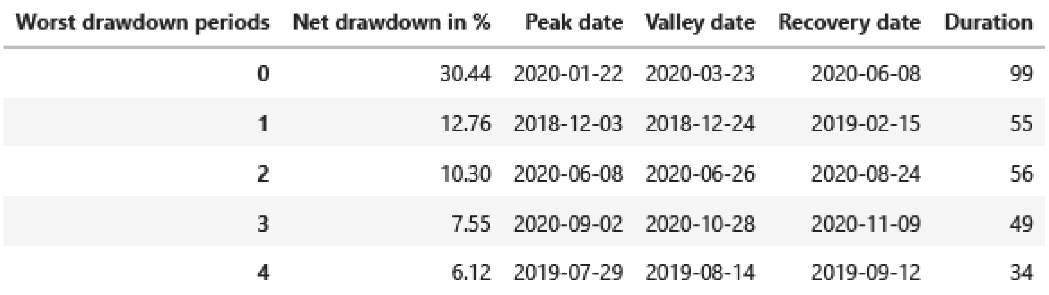


Fig. 6: Benchmarking Plots of the Best DRL Model

7.3. Performance of the DRL Model During Bearish Periods

We consider the performance of the DRL model based on the A2C algorithm during the bearish period of the 2020 market crash. The table below shows the five worst drawdown periods during the period of the test data:

Table 3: Worst Drawdown Periods



Worst drawdown periods	Net drawdown in %	Peak date	Valley date	Recovery date	Duration
0	30.44	2020-01-22	2020-03-23	2020-06-08	99
1	12.76	2018-12-03	2018-12-24	2019-02-15	55
2	10.30	2020-06-08	2020-06-26	2020-08-24	56
3	7.55	2020-09-02	2020-10-28	2020-11-09	49
4	6.12	2019-07-29	2019-08-14	2019-09-12	34

We note that the period starting end of March 2020 to July 2020 had the maximum drawdown during the test period with the longest drawdown period recorded. This is the period of the market crash which was the result of the outbreak of the COVID-19 pandemic.

From the plot of the cumulative returns in figure 6 above, we note a steep drop in the cumulative returns of the model as well as the benchmark index. However, we note that the DRL model recovers at a faster rate to record positive returns as compared to the benchmark index. This demonstrates the ability of the model to quickly learn the market structure and adjust its performance to maximize the rewards (portfolio cumulative value). See the figure below for the comparison of the recovery rate of the DRL model and the benchmark DJIA index.

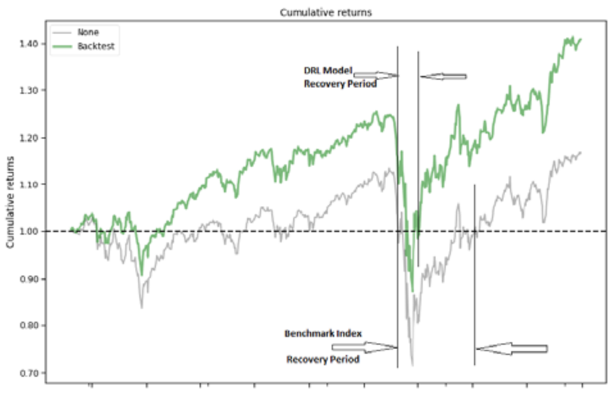


Fig. 7: Performance During Bearish Period

8. Conclusions and Further Work

The three implemented DRL models do perform relatively better than the mean variance portfolio during the period before 2020. We can conclude that the models work better in times of less market uncertainties and performance goes down during market crashes. The best model selected which is based on the A2C RL algorithms outperforms the benchmark index by far in all market conditions considered.

An analysis of the performance of the model during market down times as observed during the 2020 market crash shows that the DRL model has the ability to recover at a relatively quicker rate than the benchmark index. This is an indication of the Reinforcement Learning nature of the model to be able to explore and exploit the prevailing market environment.

Further works should be explored to improve the performance of the DRL models to be able to significantly outperform the mean-variance optimization model in all market conditions. The researcher proposes the following further works to be considered in refining the DRL model:

1. Explore the use of differential Sharpe ratio as the reward function for the DRL model;

2. Application of the DRL model on a bigger stock constituent;

3. Consideration of a number of technical indicators;

4. Consider using different portfolio initialization strategies apart from the equal weights initialization.

9. Bibliography

[1] An, B., Ang, A. and Collin-Dufresne, P., 2015. How Often Should You Take Tactical Asset Allocation Decisions?. *SSRN Electronic Journal*,.

[2] Chakravorty, G., Awasthi, A., Da Silva, B. and Singhal, M., 2018. Deep learning based global tactical asset allocation. *SSRN Electronic Journal*

[3] Clark, B., Feinstein, Z. and Simaan, M., 2020. A Machine Learning Efficient Frontier. *SSRN Electronic Journal*,.

[4] Corporate Finance Institute. 2020. *Tactical Asset Allocation (TAA) - Overview, Reasons, Example*. [online] Available at: https://corporatefinanceinstitute.com/resources/knowledge/trading-investing/tactical-asset-allocation-taa/ [Accessed 20 November 2020].

[5] Cloutier, R., Djatej, A. and Kiefer, D., 2017. A tactical asset allocation strategy that exploits variations in VIX. *Investment management and financial innovations*, (14,№ 1), pp.27-34.

[6] Faber, M., 2007. A Quantitative Approach to Tactical Asset Allocation. *The Journal of Wealth Management*, 9(4), pp.69-79. (Faber, 2007)

[7] Fa-mag.com. 2020. [online] Available at: https://www.fa-mag.com/news/why-tactical-asset-allocation-19305.html [Accessed 20 November 2020].

[8] Heidrich-Meisner, V., Lauer, M., Igel, C. and Riedmiller, M.A., 2007, April. Reinforcement learning in a nutshell. In *ESANN* (pp. 277-288).

[9] Horel, E., Sarkar, R. and Storchan, V., 2016. Dynamic Asset Allocation Using Reinforcement Learning.

[10] Liang, Z., Jiang, K., Chen, H., Zhu, J. and Li, Y., 2018. Deep reinforcement learning in portfolio management.

[11] Liu, X.Y., Yang, H., Chen, Q., Zhang, R., Yang, L., Xiao, B. and Wang, C.D., 2020. FinRL: A Deep Reinforcement Learning Library for Automated Stock Trading in Quantitative Finance. *arXiv preprint arXiv:2011.09607*.

[12] Micaletti, R., 2019. Relative Sentiment and Machine Learning for Tactical Asset Allocation. *SSRN Electronic Journal*,.

[13] Neuneier, R., 1996. Optimal asset allocation using adaptive dynamic programming. In *Advances in Neural Information Processing Systems* (pp. 952-958).

[14] Noguer i Alonso, M. and Srivastava, S., 2020. Deep Reinforcement Learning for Asset Allocation in US Equities. *SSRN Electronic Journal*,.

[15] Obeidat, S., Shapiro, D., Lemay, M., MacPherson, M.K. and Bolic, M., 2018. Adaptive portfolio asset allocation optimization with deep learning. *International Journal on Advances in Intelligent Systems*, *11*(1), pp.25-34.

[16] Soleymani, F. and Paquet, E., 2020. Financial Portfolio Optimization with Online Deep Reinforcement Learning and Restricted Stacked Autoencoder-DeepBreath. *Expert Systems with Applications*, p.113456.

[17] Stockton, K.A. and Shtekhman, A., 2010. A primer on tactical asset allocation strategy evaluation. *Vanguard Group Inc., Valley Forge, PA.*

[18] Yang, H., Liu, X., Zhong, S. and Walid, A., 2020. Deep Reinforcement Learning for Automated Stock Trading: An Ensemble Strategy. *SSRN Electronic Journal*.

Copyright ⓒ XXXX SERSC

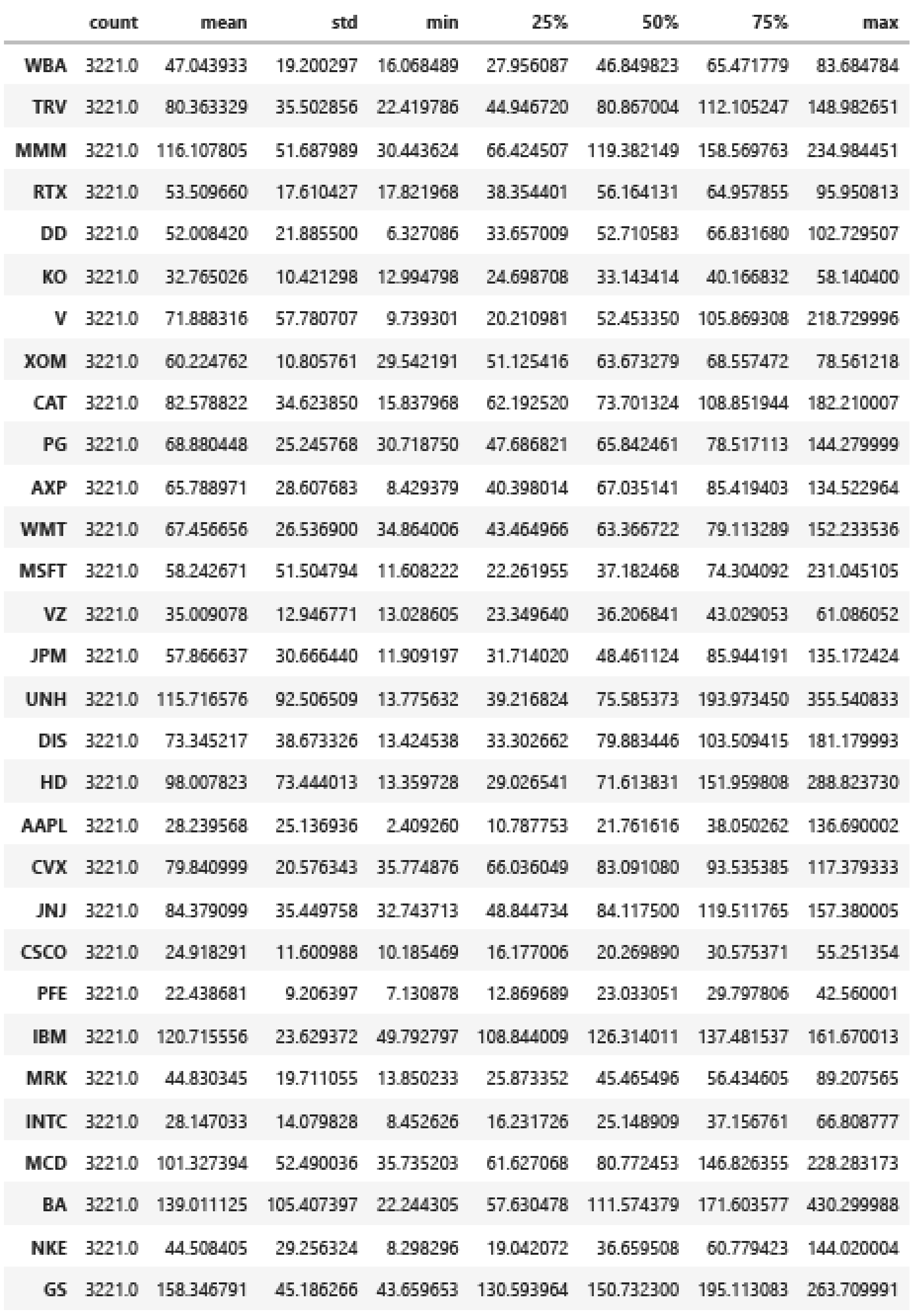
10. Appendix

A. Source Code Repository

The source code for the implementation of the project can be found on the GitHub repository Link:

https://github.com/Musonda2day/Asset-Portfolio-Management-usingDeep-Reinforcement-Learning-

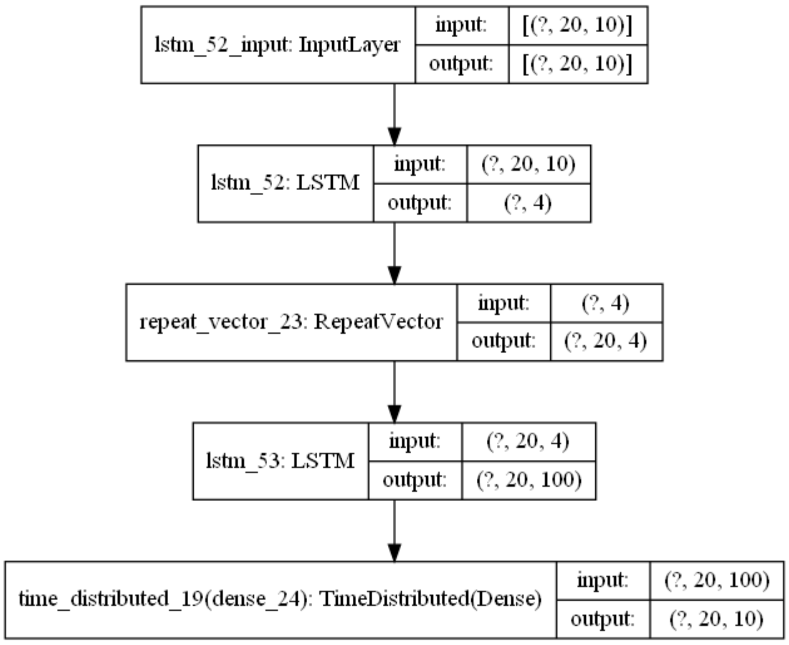
B. Descriptive Statistics of the DJIA Stocks Data



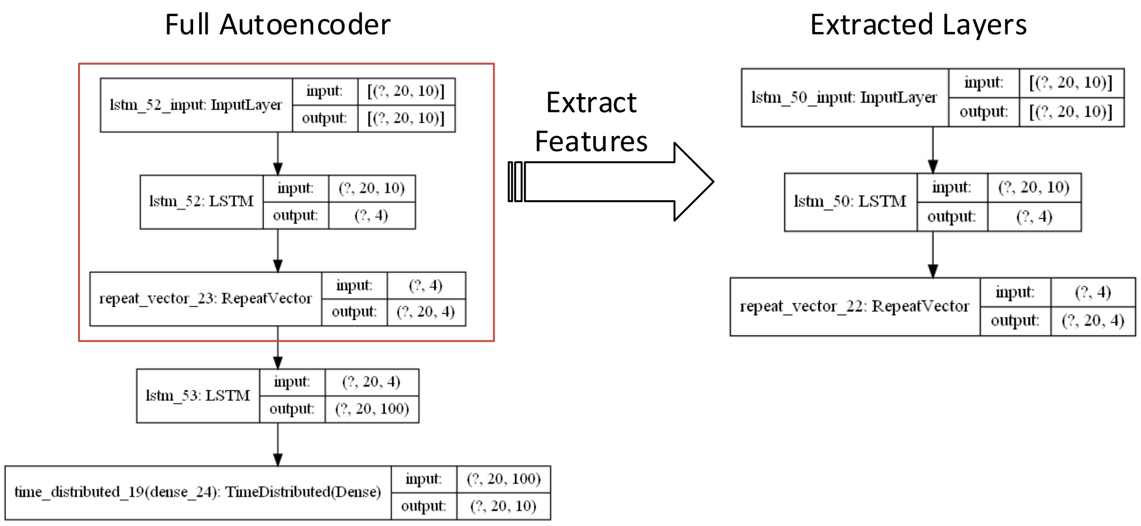
	count	mean	std	min	25%	50%	75%	max
WBA	3221.0	47.043933	19.200297	16.068489	27.956087	46.849823	65.471779	83.684784
TRV	3221.0	80.363329	35.502856	22.419786	44.946720	80.867004	112.105247	148.982651
MMM	3221.0	116.107805	51.687989	30.443624	66.424507	119.382149	158.569763	234.984451
RTX	3221.0	53.509660	17.610427	17.821968	38.354401	56.164131	64.957855	95.950813
DD	3221.0	52.008420	21.885500	6.327086	33.657009	52.710583	66.831680	102.729507
KO	3221.0	32.765026	10.421298	12.994798	24.698708	33.143414	40.166832	58.140400
V	3221.0	71.888316	57.780707	9.739301	20.210981	52.453350	105.869308	218.729996
XOM	3221.0	60.224762	10.805761	29.542191	51.125416	63.673279	68.557472	78.561218
CAT	3221.0	82.578822	34.623850	15.837968	62.192520	73.701324	108.851944	182.210007
PG	3221.0	68.880448	25.245768	30.718750	47.686821	65.842461	78.517113	144.279999
AXP	3221.0	65.788971	28.607683	8.429379	40.398014	67.035141	85.419403	134.522964
WMT	3221.0	67.456656	26.536900	34.864006	43.464966	63.366722	79.113289	152.233536
MSFT	3221.0	58.242671	51.504794	11.608222	22.261955	37.182468	74.304092	231.045105
VZ	3221.0	35.009078	12.946771	13.028605	23.349640	36.206841	43.029053	61.086052
JPM	3221.0	57.866637	30.666440	11.909197	31.714020	48.461124	85.944191	135.172424
UNH	3221.0	115.716576	92.506509	13.775632	39.216824	75.585373	193.973450	355.540833
DIS	3221.0	73.345217	38.673326	13.424538	33.302662	79.883446	103.509415	181.179993
HD	3221.0	98.007823	73.444013	13.359728	29.026541	71.613831	151.959808	288.823730
AAPL	3221.0	28.239568	25.136936	2.409260	10.787753	21.761616	38.050262	136.690002
CVX	3221.0	79.840999	20.576343	35.774876	66.036049	83.091080	93.535385	117.379333
JNJ	3221.0	84.379099	35.449758	32.743713	48.844734	84.117500	119.511765	157.380005
CSCO	3221.0	24.918291	11.600988	10.185469	16.177006	20.269890	30.575371	55.251354
PFE	3221.0	22.438681	9.206397	7.130878	12.869689	23.033051	29.797806	42.560001
IBM	3221.0	120.715556	23.629372	49.792797	108.844009	126.314011	137.481537	161.670013
MRK	3221.0	44.830345	19.711055	13.850233	25.873352	45.465496	56.434605	89.207565
INTC	3221.0	28.147033	14.079828	8.452626	16.231726	25.148909	37.156761	66.808777
MCD	3221.0	101.327394	52.490036	35.735203	61.627068	80.772453	146.826355	228.283173
BA	3221.0	139.011125	105.407397	22.244305	57.630478	111.574379	171.603577	430.299988
NKE	3221.0	44.508405	29.256324	8.298296	19.042072	36.659508	60.779423	144.020004
GS	3221.0	158.346791	45.186266	43.659653	130.593964	150.732300	195.113083	263.709991

Copyright ⓒ XXXX SERSC

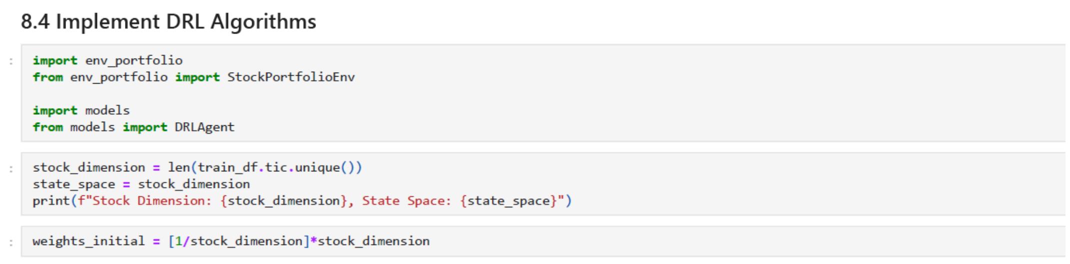
C. Stock Selection Reconstruction Autoencoder Layout Tree



D. Feature Reduction Autoencoder Layout Tree



D. Python Source Code for Implementation of DRL Models



8.4 Implement DRL Algorithms

```
import env_portfolio
from env_portfolio import StockPortfolioEnv

import models
from models import DRLAgent
```

```
stock_dimension = len(train_df.tic.unique())
state_space = stock_dimension
print(f"Stock Dimension: {stock_dimension}, State Space: {state_space}")
```

```
weights_initial = [1/stock_dimension]*stock_dimension
```

```python
env_kwargs = {
    "hmax": 500,
    "initial_amount": 1000000,
    "transaction_cost_pct": 0.001,
    "state_space": state_space,
    "stock_dim": stock_dimension,
    "tech_indicator_list": tech_indicator_list,
    "action_space": stock_dimension,
    "reward_scaling": 0,
    'initial_weights': [1/stock_dimension]*stock_dimension
}
```

```python
e_train_gym = StockPortfolioEnv(df = train_df, **env_kwargs)
```

```python
env_train, _ = e_train_gym.get_sb_env()
print(type(env_train))
```

8.4.1 Model 1: A2C

```python
# initialize
agent = DRLAgent(env = env_train)

A2C_PARAMS = {"n_steps": 5, "ent_coef": 0.005, "learning_rate": 0.0002}
model_a2c = agent.get_model(model_name="a2c",model_kwargs = A2C_PARAMS)
```

```python
trained_a2c = agent.train_model(model=model_a2c,
                                tb_log_name='a2c',
                                total_timesteps=50000)
```

```python
agent = DRLAgent(env = env_train)
PPO_PARAMS = {
    "n_steps": 2048,
    "ent_coef": 0.005,
    "learning_rate": 0.0001,
    "batch_size": 128,
}
model_ppo = agent.get_model("ppo",model_kwargs = PPO_PARAMS)
```

```python
trained_ppo = agent.train_model(model=model_ppo,
                             tb_log_name='ppo',
                             total_timesteps=50000)
```

8.4.3 Model 3: DDPG

```python
agent = DRLAgent(env = env_train)
DDPG_PARAMS = {"batch_size": 128, "buffer_size": 50000, "learning_rate": 0.001}


model_ddpg = agent.get_model("ddpg",model_kwargs = DDPG_PARAMS)
```

```python
trained_ddpg = agent.train_model(model=model_ddpg,
                             tb_log_name='ddpg',
                             total_timesteps=50000)
```

8.5 Fittng Model on Training Data

```python
# A2C Train Model
e_trade_gym = StockPortfolioEnv(df = train_df, **env_kwargs)
env_trade, obs_trade = e_trade_gym.get_sb_env()

a2c_train_daily_return, a2c_train_weights = DRLAgent.DRL_prediction(model=trained_a2c,
                        test_data = train_df,
                        test_env = env_trade,
                        test_obs = obs_trade)
```

```python
# PPO Train Model
e_trade_gym = StockPortfolioEnv(df = train_df, **env_kwargs)
env_trade, obs_trade = e_trade_gym.get_sb_env()

ppo_train_daily_return, ppo_train_weights = DRLAgent.DRL_prediction(model=trained_ppo,
                        test_data = train_df,
                        test_env = env_trade,
                        test_obs = obs_trade)
```

```
# DDPG Train Model
e_trade_gym = StockPortfolioEnv(df = train_df, **env_kwargs)
env_trade, obs_trade = e_trade_gym.get_sb_env()

ddpg_train_daily_return, ddpg_train_weights = DRLAgent.DRL_prediction(model=trained_ddpg,
                        test_data = train_df,
                        test_env = env_trade,
                        test_obs = obs_trade)
```

8.6 Trading

Assume that we have $1,000,000 initial capital at 2019-01-01. We use the DDPG model to trade Dow jones 30 stocks.

```
test_df.head(3)
```

```
# A2C Test Model
e_trade_gym = StockPortfolioEnv(df = test_df, **env_kwargs)
env_trade, obs_trade = e_trade_gym.get_sb_env()

a2c_test_daily_return, a2c_test_weights = DRLAgent.DRL_prediction(model=trained_a2c,
                        test_data = test_df,
                        test_env = env_trade,
                        test_obs = obs_trade)
```

```
a2c_test_daily_return.head()
```

```
a2c_test_weights.to_csv('a2c_test_weights.csv')
```

```
a2c_test_weights.head()
```

```
# PPO Test Model
e_trade_gym = StockPortfolioEnv(df = test_df, **env_kwargs)
env_trade, obs_trade = e_trade_gym.get_sb_env()

ppo_test_daily_return, ppo_test_weights = DRLAgent.DRL_prediction(model=trained_ppo,
                        test_data = test_df,
                        test_env = env_trade,
                        test_obs = obs_trade)
```

```
ppo_test_weights.to_csv('ppo_test_weights')
```

```
# DDPG Test Model
e_trade_gym = StockPortfolioEnv(df = test_df, **env_kwargs)
env_trade, obs_trade = e_trade_gym.get_sb_env()

ddpg_test_daily_return, ddpg_test_weights = DRLAgent.DRL_prediction(model=trained_ddpg,
                        test_data = test_df,
                        test_env = env_trade,
                        test_obs = obs_trade)
```

```
ddpg_test_weights.to_csv('ddpg_test_weights')
```