DEEP GRAPH CONVOLUTIONAL REINFORCEMENT LEARNING FOR FINANCIAL PORTFOLIO MANAGEMENT

Eetu Honkanen, Khoa Pharm-Dinh, Long Nguyen

DATA.ML.330 Media Analysis
Final Report

ABSTRACT

In the financial markets, effective portfolio management is a challenging task. This paper introduces an optimal approach that combines Deep Graph Convolutional Networks (DGCNs) with Reinforcement Learning (RL) for financial portfolio management. The proposed method uses extracting features from financial data, capturing the relationship between different assets in a portfolio as a Graph component aspect. The RL component then uses these features to make optimal decisions about portfolio weight, aiming to maximize the expected return while minimizing the loss values. The approach is data-driven, making it adaptable to changing market conditions. However, during the research, we encountered challenges in the successful results of our model. These challenges, due to factors such as parameter settings, and model architectures, led to the final performance that did not meet our initial expectations. This paper discusses the mathematical approach, model architectures, results, and challenges in detail, providing insights into the difficulty and complexities of applying advanced machine learning techniques in the domain of financial portfolio management. Despite the current results, we believe that with further research and refinement, this approach holds promise for the success of this approach.

***Index Terms*—** Convolutional Neural Network, Reinforcement Learning, Graph Convolutional Network, Portfolio Management, Machine Learning, Pytorch

0. AUTHORS' CONTRIBUTIONS

Long Nguyen was responsible for preparing the sections Abstract, Model Architecture - Feature Extraction and Normalization, Restricted stack autoencoder, Deep reinforcement learning framework, taking part in the implementation and training of the model

Khoa Pham-Dinh was responsible for preparing the sections Introduction, Problem Formulation & Mathematical Model, Results, and Conclusion of this paper, taking part in the training model and investigating various loss functions.

Eetu Honkanen was responsible for preparing sections Related Work, Graph Convolution Network, and Results of this paper, taking part in the implementation and training of the model, preparing result figures

1. INTRODUCTION

Assets in financial markets exhibit intricate interrelations, shaped by a multitude of economic, geopolitical, and market forces. Economic indicators such as GDP growth, unemployment rates, inflation, and interest rates play a pivotal role in determining asset valuations, highlighting how macroeconomic conditions influence financial instruments. Market sentiment, encapsulating the collective optimism or pessimism of investors, further steers the direction of asset prices, revealing the psychological underpinnings of market trends. Geopolitical events, including political instability, conflicts, and shifts in trade policies, exert profound impacts on global markets, underscoring the significance of international affairs in investment decisions. The performance of specific industry sectors often moves in correlation, driven by sector-specific factors, emphasizing the interconnections within markets. Monetary policies enacted by central banks directly affect stock prices and bond yields, showcasing the influence of institutional decision-making on financial landscapes. Common risk factors such as credit, liquidity, and operational risks create linkages between different assets, illustrating the shared vulnerabilities within financial systems. The globalized nature of today's economy means that regional events can have worldwide market implications, highlighting the extensive reach of globalization. Additionally, arbitrage opportunities lead to the alignment of prices across different markets as traders seek to exploit these discrepancies, demonstrating the market's self-correcting mechanisms. Collectively, these dynamics underscore that assets in financial markets are deeply interconnected through various economic, geopolitical, and market factors, shaping investment strategies and risk management practices.

A primary objective for economists and portfolio managers is to assess the connections among financial assets, influenced by the aforementioned economic factors. Following this analysis, their subsequent aim is to develop investment portfolios that capitalize on these relationships. Historically, correlations between assets were determined

using model-based approaches like Pearson or Spearman correlation. Furthermore, mathematical frameworks, notably the Markowitz model, have been employed for portfolio construction. These traditional methods offer several benefits, including established formulas, ease of application, and mathematical validation. However, despite their structured approach and straightforward implementation, these model-based techniques reveal limitations in the context of the modern era, characterized by vast data volumes and advanced machine learning technologies. Moreover, they often rest on assumptions, such as the normal distribution of returns, which do not accurately reflect the behavior of financial time series, pointing to a disconnect between theoretical models and real-world financial markets.

2. RELATED WORK

The problem of portfolio management has been extensively investigated [1][2]. State-of-the-art approaches rely heavily on Deep Reinforcement Learning (DRL) [2] [1][3]. Recent approaches to portfolio management use often restricted autoencoders to extract features from trading data[2][1][4]. In addition, graph-convolutional networks (GCNs) are used in many cases to represent interconnected relationships between institutions. From reinforcement learning techniques, especially the actor-critic network has shown good performance in determining the optimal course of action [1] [4]. There are differences on what data to include to the model. For example, in the paper [3] additional information was used on the business relations between different companies, which increased the performance of the model. The most intuitive and common evalution metric for the task is the Return of Investment (ROI) for obvious reasons. Because of the highly stochastic nature of the markets it is crucial to avoid overfitting. The model has to show consistency with the returns with different periods of time. However, financial volatility is also an important metric to consider. Lower volatility is better, being correlated with lower risks.[2]

Our main reference is the paper of Soleymani and Paquet [1]. They have presented a portfolio management model called DeepPocket, that includes a restricted stacked autoencoder (RSA), GCN, and actor-critic RL agent. The DeepPocket utilizes only public trading data from the market. The authors of the paper [1] used trading data of 28 stocks of large companies from New York Exchange (NYSE). The model outperformed market indexes in every five 90-days test datasets. Furthermore, the model managed to yield positive ROI even from the beginning of the COVID-19 crisis. DeepPocket seems to work especially for medium-term portfolio management, since the portfolio value peaked in most cases after 60 days of the test period.

3. PROBLEM FORMULATION & MATHEMATICAL MODEL

The portfolio management problem can be formulated into an optimization problem and solved by advanced machine learning methods. The sophistication of the machine learning model allows it to capture both the complex patterns in financial market data and the dynamic nature of asset prices, enabling more accurate predictions and strategic decision-making. This approach integrates cutting-edge technologies to tackle the challenges of portfolio optimization, setting a new standard for managing investments in the rapidly evolving financial landscape.

3.1. Problem formulation

In the context of portfolio optimization, leveraging techniques from graph convolution and reinforcement learning, we address an essential problem: determining the optimal asset weights in a portfolio based on historical price data to optimize a given objective function (e.g., maximizing return, Conditional Value at Risk (CVaR), Sharpe ratio, etc.).
The formulation of this optimization problem can be stated as follows: we aim to optimize a function $f(\mathbf{w}, \boldsymbol{\theta})$, subject to the constraints that each weight $w_i \geq 0$ and the sum of all weights equals 1 ($\sum_{i=1}^{n} w_i = 1$). Here, $\mathbf{w}$ represents the vector of weights assigned to each asset in the portfolio, while $\boldsymbol{\theta}$ denotes a set of parameters derived from historical data. The goal is to identify the weight vector $\mathbf{w}$ that either maximizes or minimizes the objective function f, thereby tailoring the portfolio to the investor's risk-return preferences.

$$\begin{aligned} \underset{\mathbf{w}}{\text{Optimize}} \quad & f(\mathbf{w}, \boldsymbol{\theta}) \\ \text{subject to} \quad & w_i \geq 0, \ \forall i \in \{1, 2, \ldots, n\}, \\ & \sum_{i=1}^{n} w_i = 1. \end{aligned} \tag{1}$$

where $f(\mathbf{w}, \boldsymbol{\theta})$ is a function for maximizing return, Conditional Value at Risk (CVaR), Sharpe ratio, etc.

3.2. Mathematical model

Besides implementing the machine learning model to solve portfolio optimization problems. There are other aspects of traditional finance, particularly for a financial market that we have to take into consideration such as how to calculate portfolio return and transaction fee model.
The final value of every asset within the portfolio is represented by the vector V_t. This vector also includes the total cash on hand. The normalized price return vector at period t is described as:

$$\mathbf{Y}_t := \mathbf{V}_t \oslash \mathbf{V}_{t-1} = \left[1, \frac{\mathbf{V}_{1,t}}{\mathbf{V}_{1,t-1}}, \frac{\mathbf{V}_{2,t}}{\mathbf{V}_{2,t-1}}, \cdots, \frac{\mathbf{V}_{m,t}}{\mathbf{V}_{m,t-1}}\right]^T. \tag{2}$$

At the start, it is assumed that the portfolio holds full cash, implying that the weight assigned to cash is one (1), while the weights for other assets are zero (0).

$$\mathbf{w}_0 = [1, 0, \cdots, 0]^T. \tag{3}$$

Following every instance of the agent executing a purchase or sale of an asset, the commission fee is factored in. Each transaction, whether it involves buying or selling, is associated with a transaction factor that diminishes the portfolio value. The portfolio weights vector, adjusted to incorporate transaction fees, becomes:

$$\mathbf{w}'_t = \frac{\mathbf{Y}_t \odot \mathbf{w}_{t-1}}{\mathbf{Y}_t \cdot \mathbf{w}_{t-1}}, \tag{4}$$

At the outset of period (t), the portfolio possesses a weight vector denoted by $\mathbf{w}_{t-1}$. As the agent adjusts asset allocations according to the policy, a new weight vector $\mathbf{w}'_t$ is formulated by selling and purchasing required assets. Transactions incur a commission fee denoted by $\mu \in (0, 1]$, ultimately leading to the derivation of the final weight vector $\mathbf{w}_t$.

$$P_t = \mu_t P'_t. \tag{5}$$

The portfolio value at the end of period (t) is

$$P_t = \mu_t P_{t-1} \cdot \mathbf{Y}_t \cdot \mathbf{w}_{t-1}. \tag{6}$$

Then the return rate and logarithmic return rate for period (t) are given by

$$\rho_t := \frac{P_t}{P_{t-1}} - 1 = \frac{\mu_t \cdot P'_t}{P_{t-1}} - 1 = \mu_t \mathbf{Y}_t \cdot \mathbf{w}_{t-1} - 1, \tag{7a}$$

$$R_t := \ln \frac{P_t}{P_{t-1}} = \ln(\mu_t \mathbf{Y}_t \mathbf{w}_{t-1}) - 1. \tag{7b}$$

The portfolio at final time t_f is calculated by

$$P_f = P_0 \cdot \exp\left(\sum_{t=1}^{t_f+1} R_t\right) = P_0 \prod_{t=1}^{t_f+1} \mu_t \mathbf{Y}_t \mathbf{w}_{t-1}. \tag{8}$$

Buying and selling assets result in spending and gaining cash [5], which implies that:

$$C_g = (1 - c_s) P'_t \sum_{i=1}^{m} \text{ReLU}(\mathbf{w'}_{t,i} - \mu_t \mathbf{w}_{t,i}) \tag{9}$$

$$C_s = (1 - c_b)\Big[\mathbf{w'}_{t,0} + (1 - c_s) \sum_{i=1}^{m} \text{ReLU}(\mathbf{w'}_{t,i} - \mu_t \mathbf{w}_{t,i}) - \mu_t \mathbf{w}_{t,0}\Big] = \sum_{i=1}^{m} \text{ReLU}(\mu_t \mathbf{w}_{t,i} - \mathbf{w'}_{t,i}) \tag{10}$$

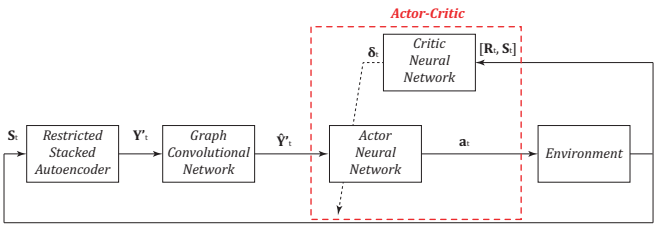

Fig. 1. Architecture of DeepPocket [1]

where the selling commission rate $0 < c_s < 1$, the buying commission rate $0 < c_b < 1$ and $\text{ReLU}(x) = \max(0, x)$. The available cash $P'_t \mathbf{w}'_{t,0}$ is $\mu_t P'_t \mathbf{w}'_{t,0}$. The transaction factor μ_t is calculated from equation

$$\mu_t = \frac{1}{1 - c_b \mathbf{w}_{t,0}}\Big[1 - c_b \mathbf{w'}_{t,0} - (c_s + c_b - c_s c_b) \sum_{i=1}^{m} \text{ReLU}(\mathbf{w'}_{t,i} - \mu_t \mathbf{w}_{t,i})\Big] \tag{11a}$$

$$\mu_t = \mu_t(\mathbf{w}_{t-1}, \mathbf{w}_t, \mathbf{Y}_t) \tag{11b}$$

The equation 11 cannot be solved analytically. [6] presents a theorem guaranteeing convergence for the iterative method of solution. The mathematical model is built on two assumptions:

-Trading is feasible at any point during business hours.

-The volume of financial instruments traded by the agent is relatively insignificant compared to the total volume of assets traded within a day.

4. MODEL ARCHITECTURE

The DeepPocket architecture can be divided into four main parts: feature extraction and normalization, restricted stacked autoencoder, GCN, and actor-critic RL agent. This is illustrated in Figure 1. This section aims to give, a comprehensive overiview of its parts.

4.1. Feature Extraction and Normalization

Due to the asynchrony of the data files and the necessary feature extraction, the preprocessing setup provides a robust method for preparing financial time series data for further tasks. It ensures that the data is clean, properly formatted, and includes a range of potentially informative technical indicators. Each asset is processed by converting the float data type, removing missing values, and calculating the financial values according to the mathematical financial function as indicators. There are a total of twelve features, which can also be understood as financial indicators, categorized by five indicators: opening, closing, low, and high prices for us to extract, which can be illustrated in Table 1 below.

Since the range and the extrema of each feature are unknown, we would normalize the indicators by dividing the 'close', 'low', and 'high' with the open values as

$$\mathbf{V}_t^{(\mathrm{Lo})} = \left[\frac{\mathrm{Lo}(t-n-1)}{Op(t-n-1)}, \cdots, \frac{\mathrm{Lo}(t-1)}{\mathrm{Op}(t-1)}\right]^T,$$
$$\mathbf{V}_t^{(\mathrm{Cl})} = \left[\frac{\mathrm{Cl}(t-n-1)}{\mathrm{Op}(t-n-1)}, \cdots, \frac{\mathrm{Cl}(t-1)}{\mathrm{Op}(t-1)}\right]^T, \quad (12)$$
$$\mathbf{V}_t^{(\mathrm{Hi})} = \left[\frac{\mathrm{Hi}(t-n-1)}{\mathrm{Op}(t-n-1)}, \cdots, \frac{\mathrm{Hi}(t-1)}{\mathrm{Op}(t-1)}\right]^T.$$

Then, the financial indicators are normalized by correlating the relationship of their value to the previous day

$$\mathbf{V}_t^{(\mathrm{FI})} = \left[\frac{\mathrm{FI}(t-n)}{\mathrm{FI}(t-n-1)}, \cdots, \frac{\mathrm{FI}(t)}{\mathrm{FI}(t-1)}\right]^T. \quad (13)$$

4.2. Restricted stack autoencoder

The RSA is used to obtain highly informative abstract features to get access to the features efficiently and compatibly with the model while reducing the computational cost and time required. The RSA is a type of neural network designed for unsupervised learning of efficient codings. The architecture of the RSA is symmetric with respect to the central hidden layer (the code layer). The RSA consists of two main parts: an encoder and a decoder. Figure 2 can be used to illustrate the RSA effectively

4.2.1. Encoder

The encoder is responsible for mapping the input data to a lower-dimensional representation. It is a feed-forward neural network that consists of three linear layers. The first layer

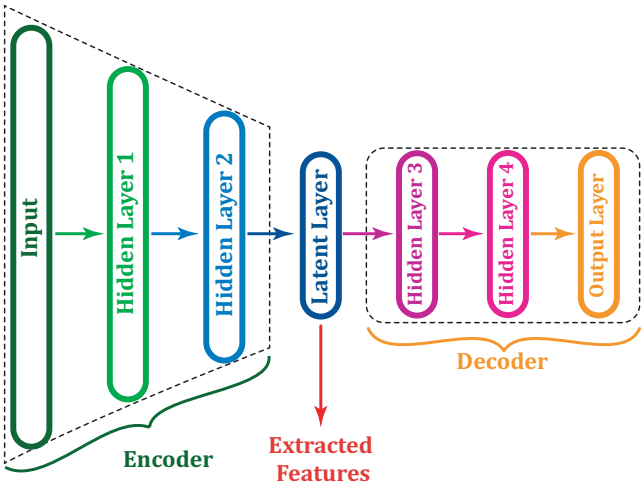

Fig. 2. Restricted stacked autoencoder [1]

Table 1. Financial Indicators Employed as Features. [1]

Financial Indicators	Definition
Average True Range	The average true range is a technical indicator that assess the volatility of an asset in the market through a certain period.
Commodity Channel Index	The commodity channel index is a momentum-based oscillator used to help determine cyclical trend of an asset price as well as strength and direction of that trend.
Commodity Selection Index	The commodity selection index is a momentum indicator that evaluates the eligibility of an asset for short term investment.
Demand Index	The demand index is an indicator that uses price and volume to assess buying and selling pressure affecting a security.
Dynamic Momentum Index	The dynamic momentum index determines if an asset is an asset is overbought or oversold.
Exponential Moving Average	An exponential moving average is a technical indicator that follow the price of an asset while prioritizing on recent data points.
Hull Moving Average	The hull moving average is a more responsive alternative to moving average indicator that focus on the current price activity whilst maintaining curve smoothness.
Momentum	The momentum in a technical indicator that determines the speed at which the price of an asset is changing.

takes the input data and transforms it to a higher-dimensional space. The second layer further processes this transformed data. The final layer of the encoder maps the processed data to the code layer. The dimensionality of the code layer is less than the input data, forcing the encoder to learn a compressed representation of the input. The first two layers use the hyperbolic tangent (tanh) activation function, and the output layer uses the softmax function to produce a probability distribution over actions.

4.2.2. Decoder

The decoder takes the compressed representation produced by the encoder and attempts to reconstruct the original input data. Like the encoder, the decoder is a feed-forward neural network and consists of three linear layers. The first layer takes the code produced by the encoder and transforms it to a higher-dimensional space. The second layer further processes this transformed data. The final layer of the decoder maps the processed data back to the original input space. They used the Rectified Linear Unit (ReLU) activation function in the convolutional layers.

4.3. Graph convolution network

In many Machine Learning (ML) problems, the data has some kind of grid structure. For instance, images are grids of pixels. In these Euclidean spaces, the convolution of operation is properly defined, which gives the basis for convolutional neural networks. [1]

The financial data concerning one instrument is often time series data at a fixed time intervals and thus embeds also in Euclidean space. However, time series data of several instruments and their mutual interrelations are better represented as a graph instead of as a Euclidean representation. Financial data can be expressed as a weighted graph $G = \{V, E, \boldsymbol{W}\}$ in which nodes V correspond to financial instruments and E edges correspond to some correlation function between pairs. The graph can be characterized by a weight adjacency matrix $\boldsymbol{W} = [w_{ij}]$ where w_{ij} is the weight between node i and j defined as

$$w_{ij} = 1 - corr(V_i, V_j)_{t-n+1,t} \tag{14}$$

where the correllation is evaluated on the interval $[t-n+1, t]$. [1]

Since the standard convolution is restricted into only Euclidean geometries, the difficulty comes up, how to encode these features into the ML algorithm. Fortunately, there is a way to extend the convolution theorem also to non-Euclidean spaces, under one condition. The theorem remains applicable, if a well-defined Fourier transform exist in the given space. By constructing a symmetric Laplacian matrix

$$\boldsymbol{L}_{sym} = \boldsymbol{D}^{-\frac{1}{2}}(\boldsymbol{D} - \boldsymbol{W})\boldsymbol{D}^{-\frac{1}{2}} \text{ ,where } \boldsymbol{D} = [d_{i,j}] = \Sigma w_{ij}, \tag{15}$$

, solving its eigenvectors $\{\phi_0, ..., \phi_{n-1}\}$ and collecting them into matrix $\boldsymbol{\Phi} = [\phi_0, ..., \phi_{n-1}]$, it is possible to define Fourier transform of a signal $\boldsymbol{x}$ on the graph

$$\mathscr{F}(\boldsymbol{x}) = \boldsymbol{\Phi}^T = \tilde{\boldsymbol{x}}. \tag{16}$$

Convolution of a signal $\boldsymbol{x} \in R^n$ with a parametric filter g_θ is defined as

$$\boldsymbol{x} *_G g_\theta = \mathscr{F}^{-1}[\mathscr{F}(\boldsymbol{x}) \odot \mathscr{F}(g_\theta)], \tag{17}$$

where $\theta \in R^n$ is a parametrization of the filter and $\odot$ is the element-wise or Hadamard product. Thus, by using the formulation of graph Fourier transform 16 spectral graph convolution can be formulated as

$$\boldsymbol{x} *_G g_\theta = \boldsymbol{\Phi}(\boldsymbol{\Phi}^T \boldsymbol{x}) \odot (\boldsymbol{\Phi}^T g_\theta). \tag{18}$$

This allows the application of CNNs to graphs. In deep GCN, multiple layers of graph convolution layers are stacked. The propagation rule for such a network is given by

$$\boldsymbol{f}_i^{l+1} = \sigma\left(\sum_{j=1}^{p} \boldsymbol{\Phi}\hat{\boldsymbol{G}}_{i,j}\boldsymbol{\Phi}^T \boldsymbol{f}_i^l\right), \quad i = 1, ..., q; j = 1, ..., p \tag{19}$$

where σ is the nonlinear activation function and $\hat{\boldsymbol{G}} = diag(g_\theta(\lambda))$. The number of features is given by q while number of assets is p. [1]

The output of the GCN is used to train the actor-critic agent.

4.4. Deep reinforcement learning framework

Reinforcement Learning (RL) is a powerful paradigm in machine learning that allows an agent to learn how to behave in an environment by performing actions and receiving rewards. [7] It has been successfully applied to various domains, from game playing to autonomous driving.

In this implementation, we focus on a specific type of RL algorithm known as the Actor-Critic method. These methods leverage two neural networks: the Actor, which decides which action to take, and the Critic, which evaluates the chosen action. The Actor's policy is improved using the gradient of the expected reward, which is estimated by the Critic. This allows the algorithm to take advantage of both policy gradient methods and value function approximation.

4.4.1. Actor

The Actor is responsible for deciding which actions to take, given the current state of the environment. It is implemented

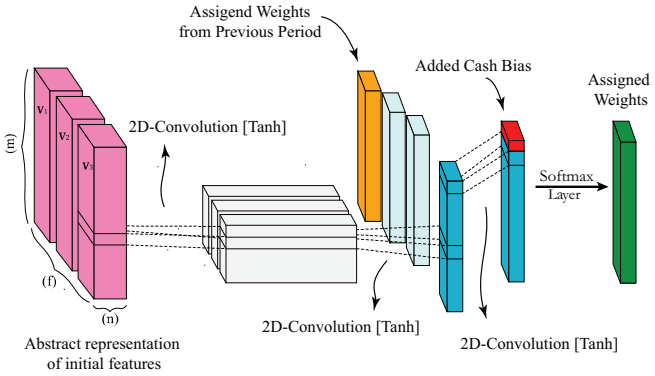

Fig. 3. Architecture of Actor [1]

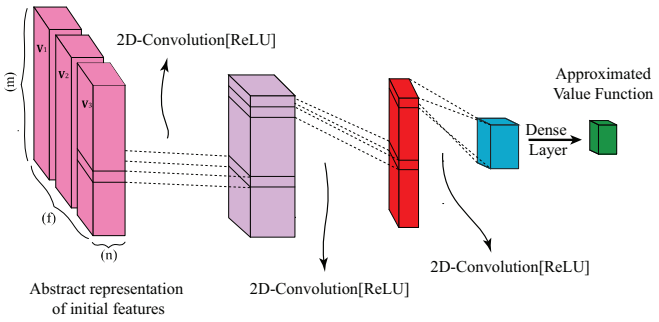

Fig. 4. Architecture of Critic [1]

as a Convolutional Neural Network (CNN) with three layers. The first two layers are convolutional layers with Tanh activation functions, while the third layer is a convolutional layer without an activation function. The output of the Actor is a probability distribution over actions, obtained by applying a softmax function to the output of the third layer. The Actor also includes a mechanism for adding a cash bias and saving the actions from the previous timestep. A demonstration of the architecture of actor can be represented in Figure 3

4.4.2. Critic

The Critic evaluates the actions taken by the Actor by estimating the value function of the current policy. Like the Actor, the Critic is implemented as a CNN. The first two layers are convolutional layers with ReLU activation functions, followed by another convolutional layer without an activation function. The output of the Critic is a single value, representing the estimated value of the current state under the current policy. A demonstration of the architecture of critic can be represented in Figure 4

4.4.3. Training Process

Since reinforcement learning will be used to update the portfolio weight distribution and minimize the loss values from the value function, the training process involves sampling batches of data from the memory buffer and updating the Actor and Critic networks based on this data. The Critic is trained to minimize the difference between its estimated state-expected values and the actual return received, while the Actor is trained to maximize the expected return based on its current portfolio, guided by the Critic's value estimates. We will use value-based methods (like Q-learning) and policy-based methods to achieve efficient and stable learning. In the context of a trading system, the goal of the agent is to learn a policy that maximizes the expected portfolio return.

5. RESULTS

This section presents the experimental results obtained from our study. We also discuss the findings in detail.

5.1. Experimental results

In our experimental setting, we used public trading data sourced from the New York Stock Exchange. The portfolio contained 28 stocks. The composition was the same compared to [1], with one exception. The stock of Royal Dutch Shell is changed to another oil company, Equinor. The data set consisted of daily trading information from an interval spanning seven years, from April 1, 2002, to April 16, 2009, and a subsequent test set spanning 90 days, from March 15, 2010, to July 21, 2010. This corresponds to Data Set 1 in reference [1]. Our implementation of the DeepPocket model has its basis in the open source code project [8].

The peak performance achieved by our model is illustrated in Figure 5. It can be easily seen that our solution follows closely to the mean price of individual stocks. The behavior of the model can be better understood by looking more closely at the weight allocations made during the test batch, which is shown in Figure 6. The model has converged to a static policy. The agent does not make trades after it has distributed equal weights to all assets at the beginning of the trading period. However, it is not the worst policy to take. The commissions are minimized when trades are not done. Volatility is also reduced compared to an individual stock.

The results shown in Figures 5 and 6 were the best results achieved. The weight distribution of another trial is shown in Figure 8 to illustrate the difficulty of ensuring that the model converges to a sensible result. During many trials, the model often converged to periodically allocate the weights, as seen in Figure 8, or modified the weights randomly.

5.2. Discussion

As it can be seen from Figure 5, while the portfolio value reported in [1] clearly outperforms the market indexes, our implementation of the model seems to fail to take advantage of the price fluctuations. Equally weighting the stocks is a trivial

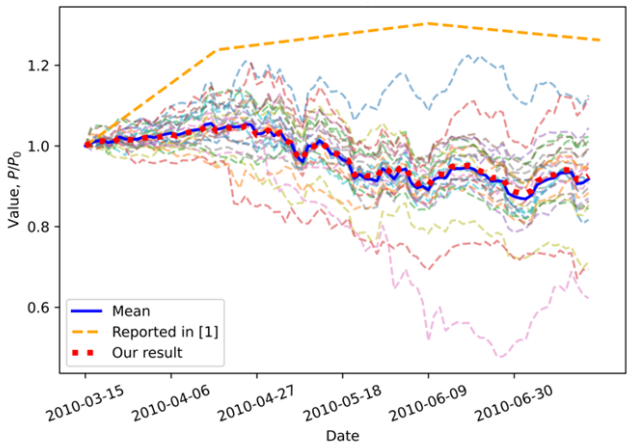

Fig. 5. Value evolution of the portfolio during the test batch. The closing prices of individual stocks are shown in the background.

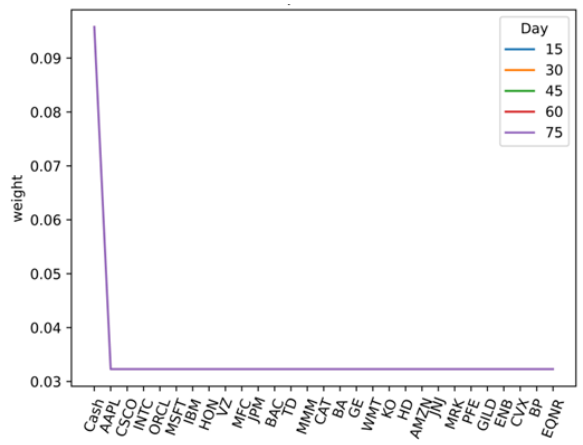

Fig. 6. The weights allocated for different stocks and cash during the test batch

solution to the portfolio management problem, and the model can be considered to be failing. Next, some speculations for our poor success are discussed.

First of all, the model is complex, which makes it by nature difficult to teach and debug. The two separate networks of actor and critic are mutually interacting while relying on the implementation of the GCN and the autoencoder. The parts are not easily separable, which forces one to work with the whole model, making diagnostics complicated.

Furthermore, documentation of some crucial details of implementation was lacking in the reference [1]. Getting the model working was difficult even though the base model [8] was comprehensive in the sense it contained some kind of implementation of all necessary parts of the model.

The evolution of the loss function during the training process is shown in Figure 7. The actor loss converges quickly,

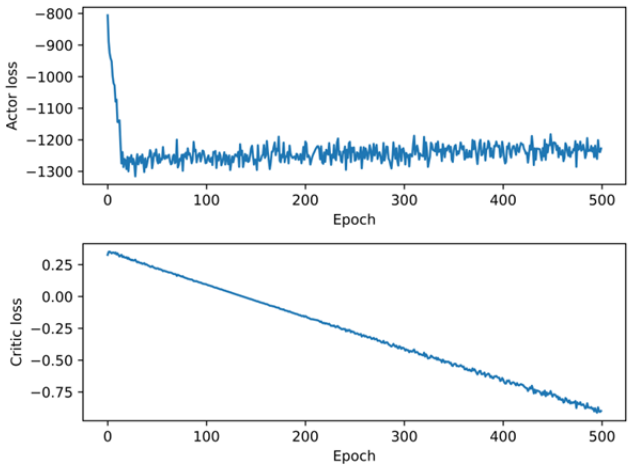

Fig. 7. The values of loss functions during the training

while the loss of the critic network steadily decreases. This could indicate that the actor converges quickly to policy in which only one action is made regardless of the state. This results in a situation in which the action space is not explored basically at all, which is the basis of the actor-critic algorithm. The agent needs variety in its memory of states and actions to evaluate their probabilities. This problem can be due to several things.

Firstly, the sampling and batching of the previous states and actions are not trivial and the documentation of this was a bit lacking in the reference. From what is inferred from the documentation, only a partial of the batch data are trained during each episode, making the environment not fully explored.

Secondly, the definition of the loss functions may have had some issues. Depending on the loss function, the training may converge to different policies. We tried a couple of different versions yielding different results. In some implementations, we got the periodic weight of assets as in 8 which is not correct, since the weight should not depend on the position of the assets listed. We also try the loss function implemented by online repository [8] and the weights do not converge during episode training but fluctuate randomly at every epoch of an episode in 9.

Additionally, our work explored the application of stacked autoencoders. There are no officially pre-trained weights available. The only pre-trained weights we have discovered are from [8]; these pertain to a basic autoencoder model featuring hidden linear layers (instead of stacked autoencoder with various convolution layers as mentioned in the original paper). There are also no specific hyperparameters of the training process for autoencoders. Consequently, in the absence of official pre-trained weights and lack of documentation, while we can decode data into time series as described in Section 4.2.2, the latent space utilized for feature extraction might significantly differ. Therefore, this variance in

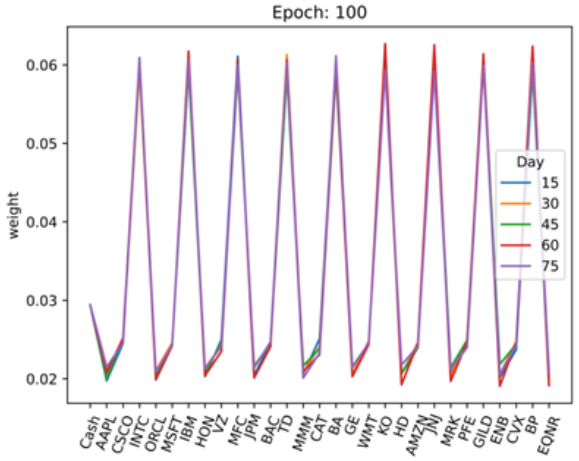

Fig. 8. Periodic distribution of weights encountered in training

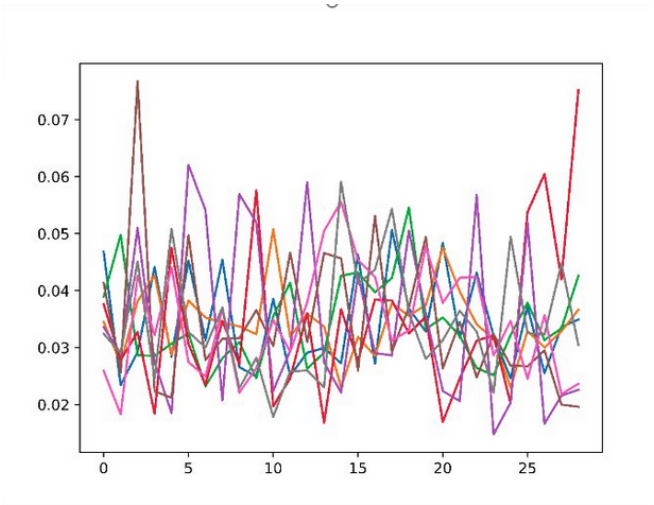

Fig. 9. Weight fluctuate randomly following loss function in [8]

the latent space could result in disparities in the way market relationships are learned.

Finally, our work only focused on Data Set 1, and it may be the case that the original paper reports the results of all Data Sets with online learning, which we did not fully implement.

6. CONCLUSIONS

In conclusion, our project aimed to replicate the results presented in [1], but we were unable to achieve the same outcomes. Despite this, the consistent convergence of our custom loss function toward an equal-weight portfolio is a promising sign that suggests that we are on the right path. This outcome, coupled with our belief that further detailed exploration will help identify and rectify the shortcomings in our methodology, fuels our optimism. The comprehensive nature of the original article, which spans multiple expertise in financial mathematics, representation learning, graph convolutional networks, reinforcement learning, and portfolio management, posed a significant challenge. The interdisciplinary approach required made it exceptionally difficult to accurately replicate the experiment within a short period without encountering errors.

Additionally, there were aspects of the implementation, such as online learning, that we did not explore. Our failure to fully operationalize the model as described in the referenced paper has, nonetheless, been a valuable learning experience. From this project, we've gained insights into the application of reinforcement learning methods and their implementation within the gym framework. We have also recognized the importance of developing robust evaluation metrics to accurately assess our research findings. Moreover, this endeavor has reaffirmed the inherent difficulties in outperforming the market using statistical methods.

Looking forward, we plan to establish new directions for our research. This includes designing baseline models, such as the Markowitz models, and testing the DeepPocket model against newer market data or within different market compositions, such as the Helsinki Stock Exchange. These future paths hold the promise of refining our approach and possibly overcoming the hurdles we faced in this initial attempt, moving us closer to achieving our research objectives.

7. REFERENCES

[1] Farzan Soleymani and Eric Paquet, "Deep graph convolutional reinforcement learning for financial portfolio management – deeppocket," *Expert systems with applications*, vol. 182, pp. 115127–, 2021.

[2] Kenniy Olorunnimbe and Herna Viktor, "Deep learning in the stock market—a systematic survey of practice, backtesting, and applications," *The Artificial intelligence review*, vol. 56, no. 3, pp. 2057–2109, 2023.

[3] Si Shi, Jianjun Li, Guohui Li, Peng Pan, Qi Chen, and Qing Sun, "Gpm: A graph convolutional network based reinforcement learning framework for portfolio management," *Neurocomputing (Amsterdam)*, vol. 498, pp. 14–27, 2022.

[4] Han Yue, Jiapeng Liu, and Qin Zhang, "Applications of markov decision process model and deep learning in quantitative portfolio management during the covid-19 pandemic," *Systems (Basel)*, vol. 10, no. 5, pp. 146–, 2022.

[5] Farzan Soleymani and Eric Paquet, "Financial portfolio optimization with online deep reinforcement learning and restricted stacked autoencoder-deepbreath," *Expert Systems with Applications*, p. 113456, 2020.

[6] Zhengyao Jiang, Dixing Xu, and Jinjun Liang, "A deep reinforcement learning framework for the financial portfolio management problem," *arXiv preprint arXiv:1706.10059*, 2017.

[7] Parag Kulkarni, "Reinforcement and systemic machine learning for decision making," *Reinforcement and Systemic Machine Learning for Decision Making*, 07 2012.

[8] MCCCSunny, "Deeppocket," https://github.com/MCCCSunny/DeepPocket, 2022.