Neural Computing and Applications (2023) 35:20663–20674
https://doi.org/10.1007/s00521-023-08862-w

ORIGINAL ARTICLE

Graph neural networks for deep portfolio optimization

Ömer Ekmekcioğlu[1] · Mustafa Ç. Pınar[1]

Received: 25 April 2022 / Accepted: 5 July 2023 / Published online: 22 July 2023

Abstract
There is extensive literature dating back to the Markowitz model on portfolio optimization. Recently, with the introduction of deep models in finance, there has been a shift in the trend of portfolio optimization toward data-driven models, departing from the traditional model-based approaches. However, deep portfolio models often encounter issues due to the non-stationary nature of data, giving unstable results. To address this issue, we advocate the utilization of graph neural networks to incorporate graphical knowledge and enhance model stability, thereby improving results in comparison with state-of-the-art recurrent architectures. Moreover, we conduct an analysis of the algorithmic risk-return trade-off for deep portfolio optimization models, offering insights into risk for fully data-driven models.

Keywords Deep learning · Portfolio optimization · Graph neural network

1 Introduction

The portfolio optimization problem is a time-honored problem that has been studied extensively since the 1960s [1]. Starting with the Markowitz model, many optimization models have been proposed to deal with different objectives [2]. Recently, the success of deep learning methods in various fields, especially computer vision and natural language processing, has laid the basis for deep learning tools that could be applied to various problems. In addition, deep learning models have significantly improved the time-series prediction task, paving the way for improvements in financial studies [3].

In contemporary portfolio optimization literature, data-driven methods have begun gaining prominence over classical model-based approaches. Data-driven methods range from reinforcement learning to natural language processing models that predict market movements from text data [4, 5]. Various deep learning methods have been tested for this problem, depending on the dataset structure. Earlier methods focused on the time-series prediction task. There also exist other approaches that rank the potential of assets via graph neural networks (GNN) [6, 7]. However, recent studies show that it is not always clear how to allocate resources for the portfolio. To bridge this gap, [8] proposes a framework where forecasting is performed first, and allocations are made afterward in a two-stage method. Similarly, multiple-stage models are proposed in the literature [9, 10]. Advanced two-stage models incorporate joint loss functions and create a pipeline where the allocations could be calculated [11]. Other examples of two-stage approaches involve the use of dynamic programming to improve the performance of the neural networks [12]. The latest approaches in the literature involve the usage of an end-to-end deep learning framework to directly learn the asset allocation [13]. Despite the advantages deep end-to-end models provide, they still present computational problems due to a lack of stability in the algorithms employed. In the literature, particularly dense layers have been shown to be very noisy, causing performance and reliability issues. Furthermore, the datasets used in the field are generally smaller than those used in other fields of deep learning literature. This fact makes networks harder to train, due to the kind of sample sizes obtainable from daily market data. We propose to use a graphical neural network approach to address these problems, and we compare our results to those of state-of-the-art recurrent models. We also leverage the classical graphical lasso algorithm to

✉ Mustafa Ç. Pınar
mustafap@bilkent.edu.tr

Ömer Ekmekcioğlu
omer.ekmekcioglu@bilkent.edu.tr

[1] Department of Industrial Engineering, Bilkent University, 06800 Ankara, Turkey

generate the inverse covariance matrix, which saves us from handcrafting all the adjacency information and makes our algorithm easy to implement.

2 Literature review

In the last few years, the computational finance literature has started to leverage deep for many different purposes. These deep models are not limited to forecasting. Generative models and deep reinforcement learning are also heavily practiced. Some examples include estimation of parameters for better and faster option pricing [14] and hedging [15, 16], market simulations and financial data generation [17, 18] and portfolio management with reinforcement learning [19, 20]. Reinforcement learning approaches have been expanded recently with studies allowing a dynamic number of assets [21], multi-agent environments for better risk management [22] and graph convolutions to account for in-depth company interdependencies [23]. Portfolio optimization, in the classical sense, remains a problem that could benefit from further deep learning practices similar to the aforementioned studies.

End-to-end methods are preferred over their alternatives due to the flexibility of their loss functions. These functions enable the neural network to be adjusted toward different goals, such as return maximization or Sharpe ratio minimization. Furthermore, constrained portfolio problems can be integrated into the end-to-end deep portfolio network with a slight change in loss functions, making the end-to-end approach to portfolio optimization quite versatile [24]. Additional features such as sparsity are often desired in portfolio optimization problems [25]. The flexibility of imposing additional features such as sparsity just by changing the loss function shows the power of the framework.

In classical time-series prediction or trend prediction problems, GNNs have proven to be successful [26–28]. This also makes GNNs a strong candidate for the stock movement classification and prediction tasks [29]. GNNs have also started to become popular in various subfields of finance, such as forecasting realized volatility [30]. However, these models generally require handcrafted adjacency matrices and are generally used for classical regression and classification tasks. We propose leveraging the effectiveness of these graphical models by incorporating the graph knowledge into the end-to-end portfolio optimization framework proposed in [24].

In the field of finance, graph neural networks are used to exploit the graphical information that underlies the market. This information may include sector connections, supply chains, supply chains [6], social media [31] or any other imaginable implicit structure. This approach gave rise to studies using different methods for the generation of adjacency graphs. While the common thread in generating graphs is to estimate the covariance matrix of the assets [32], there exist more complex ways to embed social media information or the sector indices [6]. Also, statistical analysis of graph construction is studied in [33].[1]

3 Our contributions

The goal of this paper is to reinforce the end-to-end portfolio optimization framework that was recently proposed in [13]. Our novel contribution originates from utilizing graph neural networks as a tool to solve some of the inherent consistency issues of the framework. We aim to provide a compact alternative to the LSTM layers in such architectures to stabilize performance issues that might arise in real-life portfolio datasets. The methods proposed in this study overlap with the models used in state-of-the-art asset management and forecast studies, such as e.g., [6, 26], with the difference being the framework that we are trying to build upon. The main contributions of this paper can be summarized as follows:

- Adapting GNN models to the end-to-end portfolio optimization framework: GNNs yield higher portfolio returns compared to the state-of-the-art recurrent models in [13].
- Recurrent models suffer from stability issues. We address this problem by using GNNs and analyzing the algorithmic risk-return trade-off. This result also parallels the implicit outcome of using graph neural networks in financial forecast studies [6].
- Instead of a handcrafted adjacency matrix, we use the inverse covariance matrix of the returns, which is a method practiced in optimization literature but not commonly deployed in deep learning literature. As a result, our framework benefits from the sparsity of the adjacency matrix, while requiring no expert knowledge to create it.

4 Background

In time-series analysis, standard neural network architectures generally incorporate recurrent layers, various convolutional layers, such as dilated convolutions, and attention mechanisms [34]. These layers allow models to effectively use sequential data. Attention mechanisms and

[1] We became aware of this reference while our paper was under review.

GNNs are also effectively used in recent literature for time-series prediction tasks.

4.1 Convolution

In addition to the success of convolutional layers in computer vision and image processing literature, one-dimensional convolutional layers are used in the literature on time-series analysis [35]. Methods such as dilated convolutions are effective in increasing the receptive field, yielding results comparable to recurrent models.

4.2 LSTM

LSTM is among the most important layers in the deep sequence models due to its success in natural language processing and time-series data. Even though the inner structure of the layer is quite convoluted, it essentially learns to remember and forget sequential information to improve the performance of the classical recurrent layers [36].

4.3 Graph neural networks

Graph convolutions are the generalized version of the convolutional networks. Instead of using the transformation of the adjacent features, a graph structure is used to generalize the adjacency notion. Most basic two-layer graph convolutional network is represented as follows:

$$H = f(X, A) = \phi(AReLU(AXW_1)W_2),$$

where $X \in \mathbb{R}^{p \times l}$ with p denoting the company number (for the assets) and l denotes the lookback window (features). $A \in \mathbb{R}^{p \times p}$ is the (normalized) adjacency matrix. $W_i \in \mathbb{R}^{l \times h_i}$ and h_i denote the number of filters used in each layer i.

Some prominent layers used in GNNs are GAT, GCN and ARMA.

- GAT:

  In [37], it was proposed to use an attention mechanism in GNNs. For every node, this layer concatenates the joint information of its neighbors while weighting the information important to that node. The operation could be described as follows:

  $$\alpha_{ij} = \frac{exp(LeakyReLU(a^T[XW_i \| XW_j]))}{\sum_{\mathcal{N}_i} exp(LeakyReLU(a^T[XW_i \| XW_j]))}$$
  $$X' = \alpha XW + b$$

  where $\|$ operator denotes concatenation and $\mathcal{N}$ denotes the adjacency matrix with self-loops. Due to its performance in node classification problems, this layer has been used in most of our architectures.

- GCS:

  This is a simple graph convolutional layer with residual connections. The operation performed by this layer is as follows:

  $$\tilde{A} = D^{-1/2}AD^{-1/2}$$
  $$X' = D^{-1/2}AD^{-1/2}XW_1 + XW_2 + b,$$

  where D denotes the degree matrix, and the adjacency matrix A does not contain self-loops [38]. With the residual connections, this layer is very easy to train and could be considered as the baseline graph convolutional layer.

- ARMA:

  This layer carries the spirit of the original auto-regressive models and GCS layers [39]. For a given order K, the model performs the following:

  $$X^0 = X$$
  $$\tilde{A} = D^{-1/2}AD^{-1/2}$$
  $$X_k^{t+1} = \sigma(\tilde{A}X^tW^t + XV^t)$$
  $$X' = \frac{1}{K}\sum_{k=1}^{K} X_k^t.$$

4.4 Optimization problem

The mean-variance portfolio problem has been extensively explored in the literature. There are various analyses involving different loss functions and different constraints. The classical optimization problem targets maximizing future returns while keeping a cap on the variance of portfolio returns using the first- and second-order information of the time-series data:

$$\begin{aligned} \max_w \quad & \mu^T w - \alpha w^T \Sigma w \\ \text{s.t.} \quad & w_i \geq 0 \quad \forall i \in \{1, \ldots, p\} \\ & e^T w = 1, \end{aligned}$$

or minimizing the variance of portfolio return while ensuring a minimum expected portfolio return:

$$\begin{aligned} \min_w \quad & w^T \Sigma w - \beta \mu^T w \\ \text{s.t.} \quad & w_i \geq 0 \quad \forall i \in \{1, \ldots, p\} \\ & e^T w = 1 \end{aligned}$$

where e denotes a vector with all components equal to one and α, β are real scalars that reflect the balance between risk and returns (the models are to be equivalent under judicious choices of parameters α and β). There are many variants to the above problems that relax the non-negativity constraint or change the objective function to a fractional

function that maximizes the Sharpe ratio [40]. Essentially, these problems are similar in nature, and the goal is to obtain the highest return with the least amount of portfolio risk. In this study, we will focus on the notion of algorithmic risk with less attention paid to portfolio risk, as there are different robustness problems that arise in fully data-driven approaches.

5 Our method

Let $\mathcal{G} = (V, E)$ be a graph. In our model, V denotes the companies, and edges, E, are weightless and bidirectional to denote the relations between the companies. We represent the data by a matrix $D \in \mathbb{R}^{n \times p}$ where there are n time steps, i.e., trading days, and p companies. The first step is to create a graph representation using the given dataset. To infer the adjacency relation, we use the graphical lasso algorithm to calculate the inverse covariance matrix. Since this sparse matrix gives the conditional relationship between each company, it is an intuitive model-based approach we can use freely. The inverse covariance matrix is often used in optimization-based portfolio allocation studies as the precision matrix is included in the analytical solution [41]. Further improvements to the matrix estimation with distributionally robust estimates are proposed in [42]. Inverse covariance estimates also appear in recent studies examining the market structure in the implementation of dynamic portfolio optimization models [43]. In the literature, there are studies that manually construct graphs using market information available [26]. However, to have an easy-to-use framework, we refrain from using expert knowledge and utilize a graphical lasso. The graphical lasso algorithm solves the following problem.

$$\hat{\Theta} = \arg\min_{\Theta} Tr(S\Theta) - \log\det(\Theta) + \lambda \sum_{i \neq j} |\Theta|,$$

where S denotes the covariance matrix. This optimization problem is a well-known and well-studied problem in the literature, and its adaptive extensions are also available [44]. These adaptive extensions are important in cases where the change in the adjacency graph is crucial, such as car sensor data. In the present study, due to the structure of the real-life dataset that we have worked on, this extension is not used. However, especially in large datasets, an adaptive version could be preferred to have more precise adjacency matrices.

Combining the return data with the adjacency graph obtained, we obtain our input graph, $\mathcal{G}$, and we use our proposed GNN model to predict the portfolio allocation by modifying the previous formulation using flattening and dense layers:

$$out = softmax(flatten(H)W_3),$$

where $W_3 \in \mathbb{R}^{h_f p \times p}$ since the flattened graph will be a vector of length $h_f \times p$, and the output will be p-dimensional. The "softmax" activation is very useful to directly obtain the allocations in the portfolio optimization problem that is nonzero and summed to one. Furthermore, allocation constraints could be implemented or relaxed by modifying the output activation of the network [24].

In this study, we mainly focus on maximizing returns. In [45], it is shown that the Sharpe ratio criterion can effectively minimize the variance of the returns, which is the portfolio risk. In addition to the portfolio risk, the stability of the algorithm is an additional source of concern for investors in data-driven methods. We believe that having a robust and stable model is instrumental in achieving high returns while effectively minimizing algorithmic risk. We aim to obtain results with less algorithmic risk by manipulating the network architecture of the deep learning approaches so that we can maximize the return per risk notion in the fully data-driven case. We also look at the algorithmic risk-return trade-off for multiple algorithms.

6 Architecture

We have tested multiple GNN layers and dense layers. However, due to the dimensionality of the dataset, we prefer to stick to a simple architecture similar to the ones used in node classification problems [37]. Input to our model is the past return information, the first and second moments of the returns, as well as the adjacency matrix. This input is fed into three layers of graph convolutions. Afterward, we flatten the data and feed it into two dense layers. In these dense layers, we use residual connections to improve training performance. We also observed that utilizing the GeLu activation function instead of the classical ReLu in the convolutional layers yields better results. On different tests, we have used different graphical convolutions, and the results are reported in the results section.

In Fig. 1, we represent our GNN architecture. The number of filters represents the number of attention heads used in the GAT model. We also used a very similar architecture in our ARMA and GCS models; however, the hyperparameters slightly vary. Essentially, in all our architectures, we stack multiple graphical convolutional layers and follow up with dense layers with drop-out and residual connections.

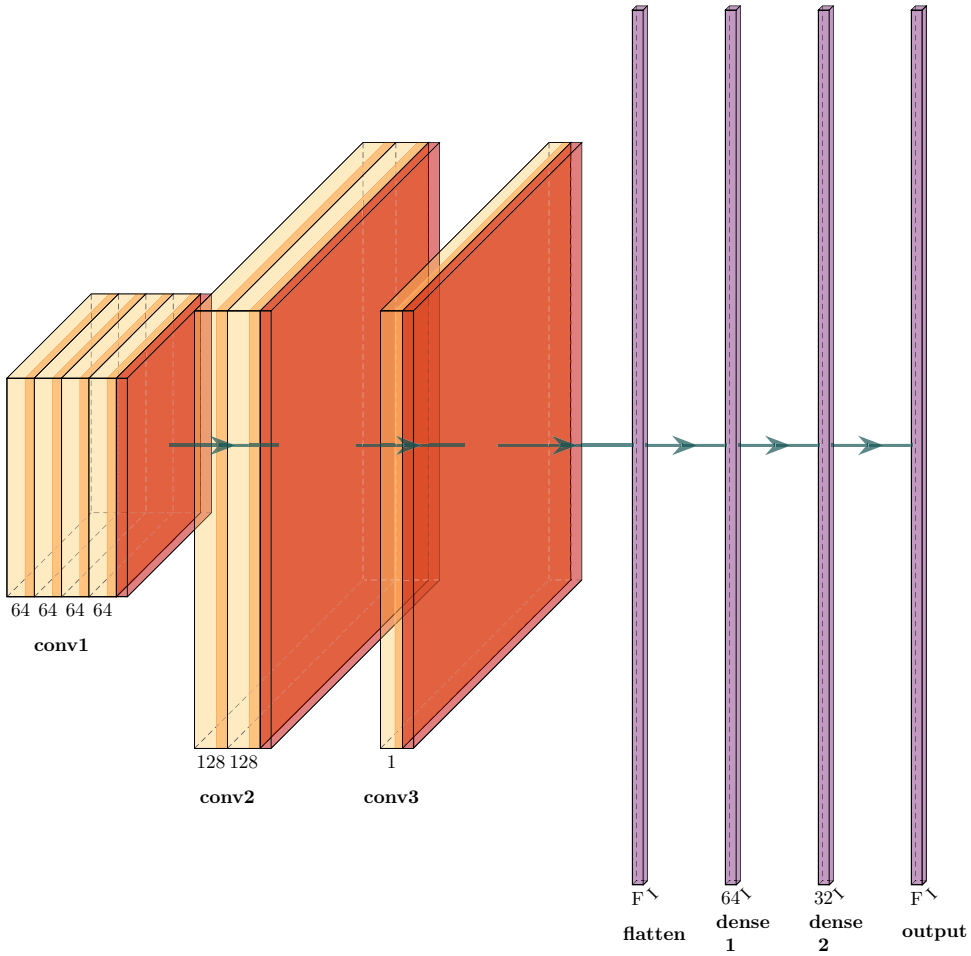

Fig. 1 Proposed GNN architecture

7 Results

We have used datasets obtained from https://host.uniroma3.it/docenti/cesarone/DataSets.htm, which are also used in [46]. Among the available datasets, we have used Dow Jones 2005 (Dataset 1), Euro Bonds (Dataset 2), Commodities and Italian Bonds Mix (Dataset 3) and World Bonds Mix (Dataset 4). These datasets contain 1564 days of asset data, with sizes varying from 11 to 104 securities. We also reproduce from [46] the data features in Table 1.

Twelve trading instances were created from the datasets, each with its own set of training and test data. Each instance's training data were a combination of the testing and training data from the preceding instance. The initial instance consisted of 400 days of training and 100 days of prediction. Up until all of the available data were covered, the datasets were combined to provide training data for the future mini-data. This approach is consistent with the methodology used in [24]. To evaluate algorithmic risk, we ran each model 10 times for each mini dataset and reported the average results. Our proposed models were named *ARMA*, *GAT* and *GCS*. The *LSTM* architecture was proposed in [13] and [24]. The classical long-only mean-variance return maximization problem was denoted as *CVX*. We validated and tuned our models using a $90\% - 10\%$ split in training data. Throughout all the experiments, TensorFlow and Spektral libraries are used to train and evaluate the neural networks [38]. In the implementation of the optimization-based models, CVXPY is used [47].

Table 1 Dataset structure

Datasets	Assets	Days	Range
DowJones2005 (1)	21	1564	1/2013-12/2018
EuroBonds (2)	62	1564	1/2013-12/2018
ItBondComodities (3)	11	1564	1/2013-12/2018
WorldMixBonds (4)	104	1564	1/2013-12/2018

Our primary goal in terms of the performance criteria was to achieve high returns. Therefore, Table 2 summarizes our primary results. The best results achieved for each dataset is highlighted in bold. In order to understand the risk associated with the returns, we have also looked into

Table 2 Evaluation of total returns on datasets

Dataset	ARMA	GAT	GCS	LSTM	CVX
1	1.3601	1.5557	1.3921	1.2708	**1.8715**
2	1.6936	0.8172	**2.1713**	0.8418	1.1609
3	**0.5046**	0.3610	0.4217	0.3898	0.4112
4	1.2971	0.7250	**1.7297**	0.8156	1.2526

the Sharpe ratios derived from the model outputs. We report these results in Table 3. The Sharpe ratios were not our primary concern, and deep models are not trained for that purpose. It is solely used to gain an understanding of the performance of our models. Our secondary focus was to reduce algorithmic risk, which we defined as the risk of not converging to similar local optimal points consistently.

The results in Table 2 indicate that the proposed GCS and ARMA networks yield results surpassing the returns obtained by the state-of-the-art LSTM networks. Moreover, the GAT network fixes the stability problem of the deep models. Due to the limited observations in the financial datasets, it is quite difficult to train a robust deep architecture. This limitation is also applicable to the model-based architectures, as in the fourth dataset, the CVX model failed to give meaningful results when the dataset was small. We omitted the first two time instances because of this problem. As a result of these challenges, simpler models like GCS are effective in those datasets. Furthermore, GAT results appear to be consistent in most instances. The significant stability difference between the LSTM model and our proposed GAT model is apparent in the figures. The difference in the error bars highlights the observation that the algorithmic risk decreases with the use of the GAT models. Figures 2 and 3 show the standard deviation of the different returns obtained from the first dataset for every instance of each run. Similarly, Figs. 4 and 5 show the maximum difference encountered in every evaluation period. We present the detailed results obtained using the first dataset in Table 4. The rest of the tables regarding the other datasets are in appendices. In general, the error bars in the figures representing the LSTM models are larger, indicating a larger standard deviation in the obtained results, whereas the GAT model is more consistent and the error bars reflect only a small amount of variation. Figures 6 and 7 highlight how the variance shrinks with the adoption of graphical models.

Table 3 Evaluation of Sharpe ratios on datasets

Dataset	ARMA	GAT	GCS	LSTM	CVX
1	0.6164	0.6142	0.6337	0.5820	**0.7612**
2	0.7542	0.4543	**1.0111**	0.5772	0.0146
3	0.5407	0.5373	0.4396	0.5205	**0.7260**
4	0.4957	0.4015	**0.7360**	0.4859	0.0815

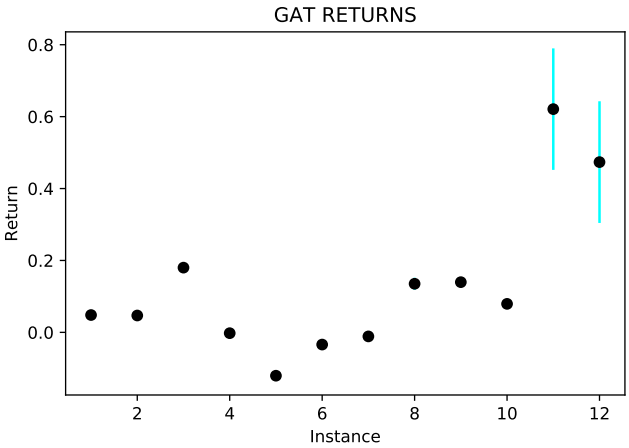

Fig. 2 GAT returns with std. error on Dataset1

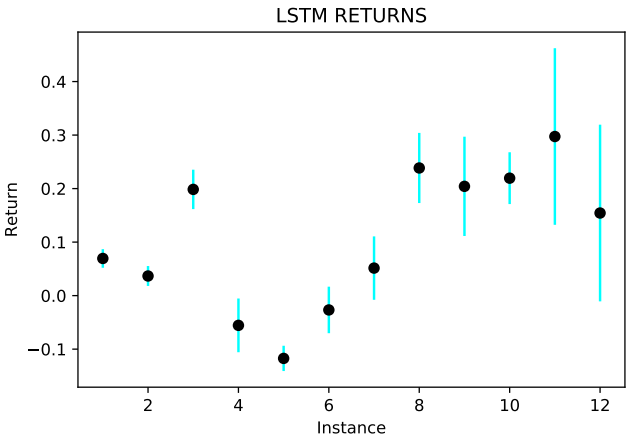

Fig. 3 LSTM2 returns with std. error on Dataset1

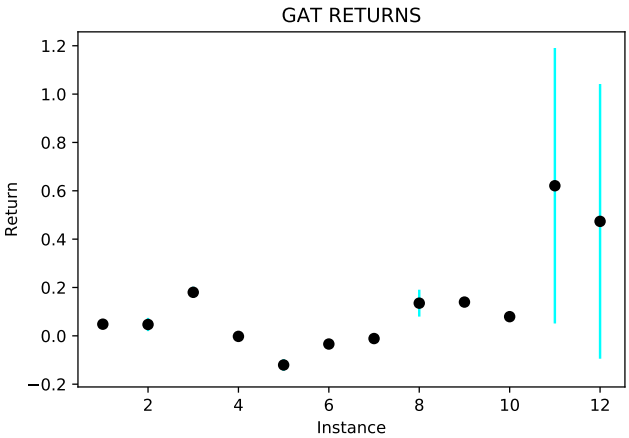

Fig. 4 GAT returns with max error on Dataset1

In some datasets, LSTM mean returns are slightly higher than GAT model returns. However, the variance in the returns is significantly high for every training period, making the models quite unreliable. Similarly, if we want to obtain higher returns with higher risk, the GCS and ARMA models could be used to outperform the returns of the

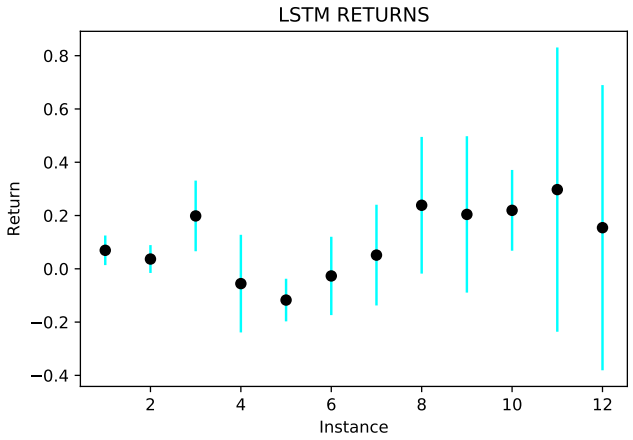

Fig. 5 LSTM2 returns with max error on Dataset1

Table 4 Return statistics on every instance for Dataset 1

GAT mean	GAT std.	LSTM mean	LSTM std.
0.048215	0.00012	0.069446	0.01740
0.046904	0.00815	0.036681	0.01861
0.180032	0.00672	0.198491	0.03683
− 0.002247	0.00260	− 0.055583	0.05020
− 0.120561	0.00771	− 0.117312	0.02362
− 0.033928	0.00001	− 0.026766	0.04347
− 0.011228	0.00029	0.051438	0.05923
0.135172	0.01676	0.238524	0.06555
0.139644	0.00026	0.204196	0.09279
0.079272	0.00017	0.219471	0.04841
0.620897	0.16890	0.297301	0.16500
0.473544	0.16920	0.154340	0.16508

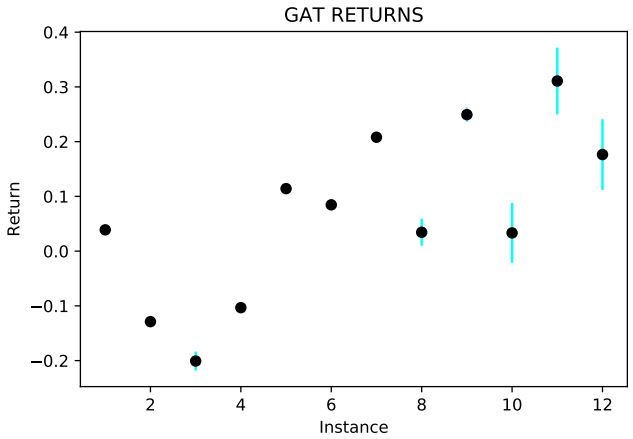

Fig. 6 GAT returns with std. error on Dataset2

recurrent models. There are occasions where the standard deviation of the LSTM and graphical models is equal to zero. In those cases, they yield the same return, converging to the same local optima. Figures 8 and 9 demonstrate this

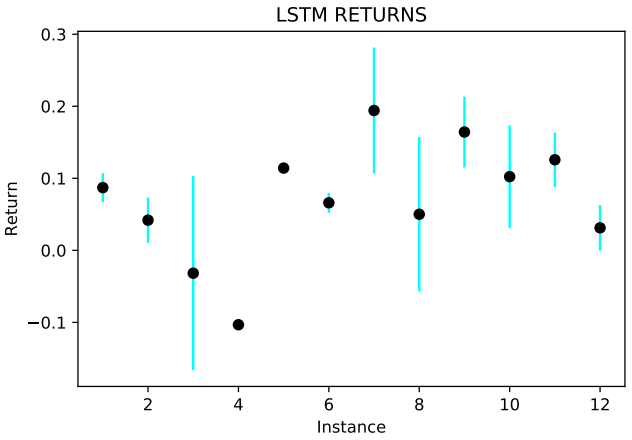

Fig. 7 LSTM2 returns with std. error on Dataset2

phenomenon. However, this is not the case for WorldMix-Bonds dataset. The difference in return patterns in Figs. 10 and 11 indicate that the models converge to a different allocation policy. In other points, depending on the dataset, LSTM and graph models are competing in returns and standard deviations. Since it is difficult to come across a model strongly dominating another in each instance, it is meaningful to compare the cumulative returns as shown in Table 2. These results show that graph models are superior to the LSTM model in various scenarios, both in terms of returns and in terms of Sharpe ratios.

Portfolio optimization is often analyzed along with Pareto-optimal solutions due to the risk-return trade-off. In our results, we can see a variant of the portfolio's risk-return trade-off in terms of the algorithms' risk-return trade-off. Our proposed GAT model significantly reduces algorithmic risk while maintaining or even exceeding the return levels achieved with state-of-the-art LSTM methods. Furthermore, the GCS and ARMA models manage to beat the returns of the LSTM model, which makes the graphical models highly effective for deep portfolio optimization.

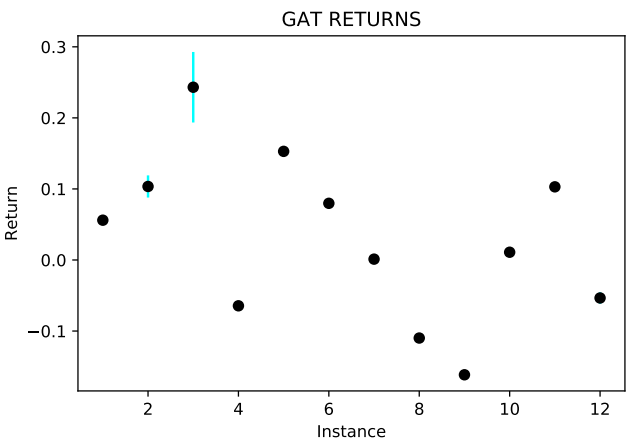

Fig. 8 GAT returns with std. error on Dataset3

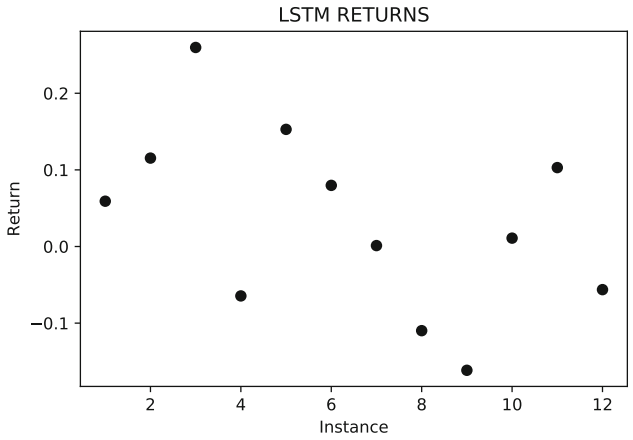

Fig. 9 LSTM2 returns with std. error on Dataset3

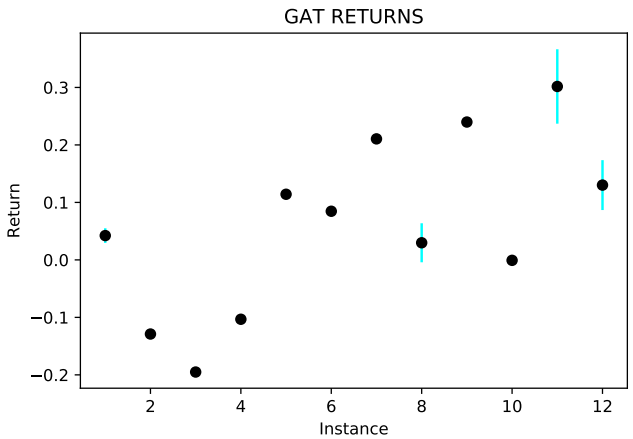

Fig. 10 GAT returns with std. error on Dataset4

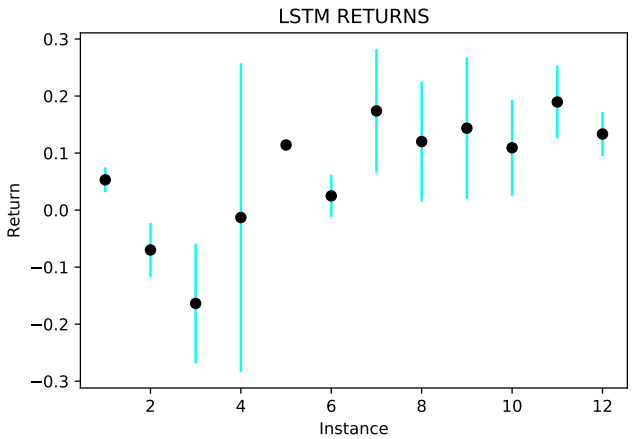

Fig. 11 LSTM2 returns with std. error on Dataset4

8 Remarks

In deep portfolio optimization, some of the traditional portfolio optimization metrics should be evaluated from a different perspective. As it could be observed from the results reported in [45], the effect of Sharpe ratio minimization effectively decreases the variance of returns. However, when minimizing the Sharpe ratio is the key objective, returns also decrease significantly. Furthermore, the resulting portfolio remains risky due to the algorithmic risk stemming from the instability of the models. We can mitigate this issue separately by focusing on the stability and robustness of the network. This will yield returns that are not far off the predicted levels, which is beneficial for evaluating the portfolio's performance.

Recent studies suggest that the first few days are important in a recurrent model [13]. Similarly, from our results, we observed that the graphical relations are effective in explaining the information within the data, which makes the model more robust than its recurrent counterparts. These insights are important for deep portfolio optimization frameworks since the notion of algorithmic "risk" should also be considered for fully data-driven approaches. We also observe that different layers excel at interpreting different return structures. For example, when returns are increasing in the data, ARMA convolutions return high rewards compared to their alternatives. However, due to its higher loss when returns are declining, the model overall may fail to return the highest rewards. These observations may prove valuable when making allocation decisions using an ensemble model, which we plan to focus on in our future work.

9 Conclusion

End-to-end deep learning methods have emerged as serious contenders against model-based portfolio optimization tools. From a deep learning perspective, we examined the classical Markowitz portfolio optimization problem and tackled one of its main challenges: stability. When using data-driven approaches, the risk that algorithms pose is a significant issue. We improved the stability of recurrent models that are at the forefront of this field by utilizing graph neural networks. Our contribution to the literature is two-fold. Firstly, we utilized the graphical lasso to generate an adjacency matrix of the companies, which enabled us to obtain an adjacency matrix without requiring expert knowledge or manual intervention. Secondly, we demonstrated that GNN results are more stable than LSTM models, and the achieved returns are comparable, if not better in some cases. These findings suggest that minimizing algorithmic risk can be improved by employing neural network techniques, which are essential in a fully data-driven architecture.

Appendix 1 Extended results

See Figs. 12, 13, 14, 15, 16, 17, 18 and 19.

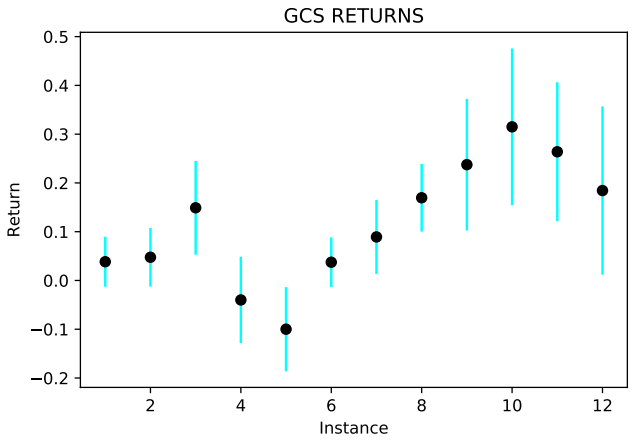

Fig. 12 GCS returns with std. error on Dataset1

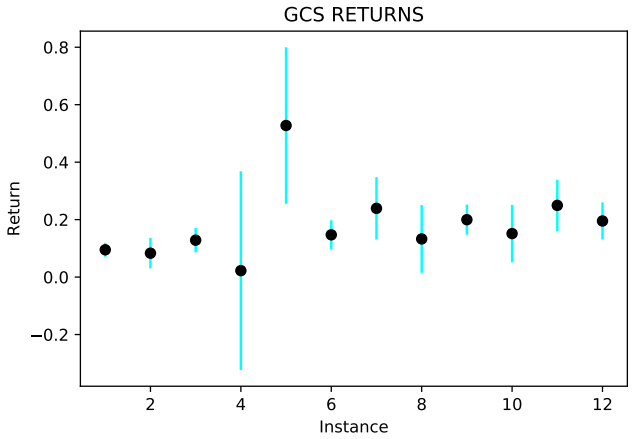

Fig. 13 GCS returns with std. error on Dataset2

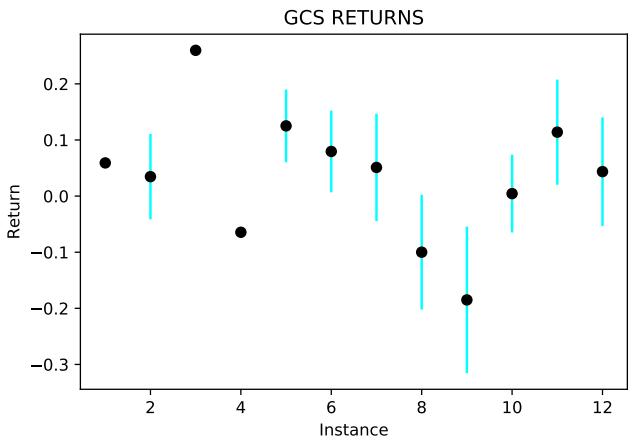

Fig. 14 GCS returns with std. error on Dataset3

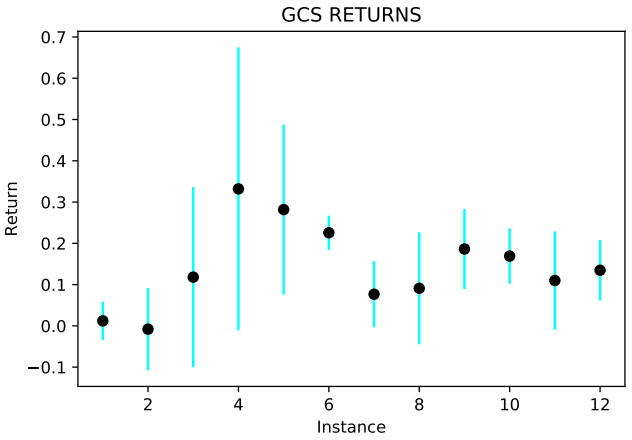

Fig. 15 GCS returns with std. error on Dataset4

Fig. 16 ARMA returns with std. error on Dataset1

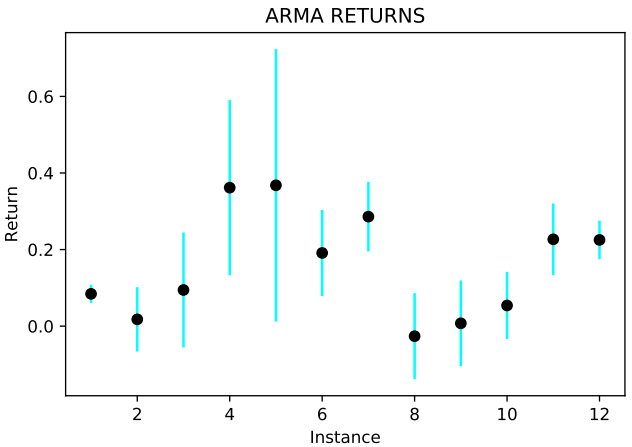

Fig. 17 ARMA returns with std. error on Dataset2

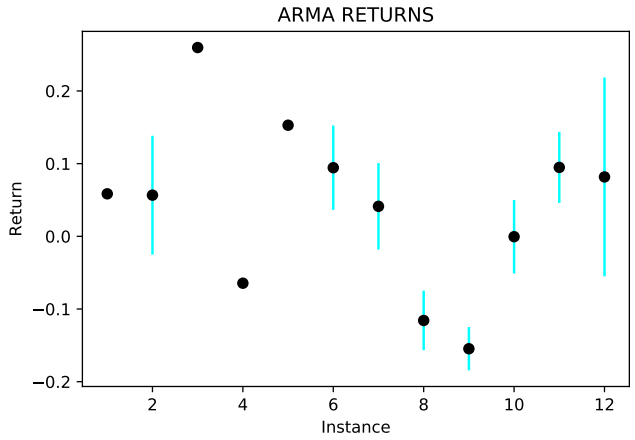

Fig. 18 ARMA returns with std. error on Dataset3

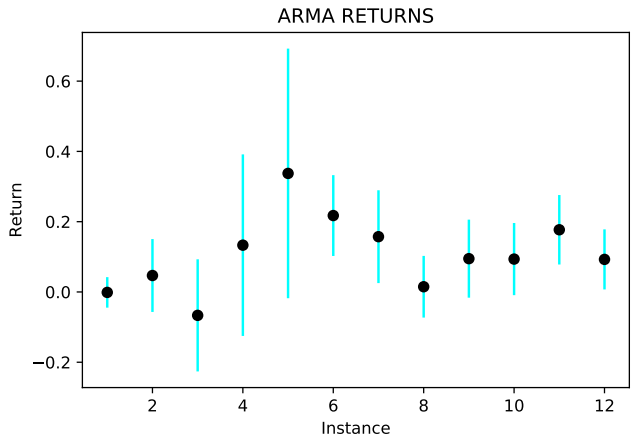

Fig. 19 ARMA returns with std. error on Dataset4

See Tables 5, 6, 7, 8, 9, 10 and 11.
We included additional test results in this section

Table 5 Return statistics on every instance for Dataset1

GCS mean	GCS std.	ARMA mean	ARMA std.
0.038412	0.05123	0.041035	0.03390
0.04756	0.06023	0.049132	0.05677
0.149021	0.09615	0.125844	0.09016
− 0.03995	0.08870	− 0.01665	0.06642
− 0.099934	0.08627	− 0.039357	0.08307
0.037406	0.05082	− 0.016773	0.06644
0.089273	0.07589	0.125617	0.13105
0.169544	0.06934	0.173572	0.05251
0.237483	0.13495	0.161368	0.12450
0.315011	0.16075	0.192065	0.10870
0.263893	0.14250	0.274490	0.18700
0.184350	0.17246	0.289794	0.18170

Table 6 Return statistics on every instance for Dataset 2

GAT mean	GAT std.	LSTM mean	LSTM std.
0.038904	0.00417	0.087058	0.02000
− 0.128798	0.00018	0.041902	0.03134
− 0.200803	0.01719	− 0.031716	0.13476
− 0.103257	0	− 0.103261	0
0.114179	0	0.114179	0
0.084612	0	0.065986	0.01382
0.20806	0.00484	0.19411	0.08750
0.03447	0.024714	0.050097	0.10716
0.249354	0.01272	0.164206	0.04918
0.033278	0.05461	0.102287	0.07098
0.310822	0.06093	0.125728	0.037847
0.176451	0.06454	0.031241	0.03167

Table 7 Return statistics on every instance for Dataset 2

GCS mean	GCS std.	ARMA mean	ARMA std.
0.094956	0.02412	0.084313	0.02368
0.083139	0.05309	0.017688	0.08396
0.128395	0.04228	0.094413	0.15007
0.022334	0.34606	0.361661	0.22888
0.527551	0.27234	0.367827	0.35571
0.14709	0.05072	0.191131	0.11247
0.239199	0.10860	0.286001	0.09095
0.132828	0.11793	− 0.026266	0.11242
0.199725	0.05234	0.00733	0.11185
0.151399	0.10003	0.053952	0.08785
0.249482	0.08962	0.226717	0.09371
0.19520	0.06458	0.225212	0.05041

Table 8 Return statistics on every instance for Dataset 3

GAT mean	GAT std.	LSTM mean	LSTM std.
0.056009	0.00611	0.059118	0
0.103474	0.01560	0.115378	0.00001
0.243125	0.04964	0.259768	0
− 0.064475	0.00002	− 0.064491	0
0.152873	0	0.152878	0
0.079757	0.00004	0.079773	0
0.001202	0	0.001199	0
− 0.109843	0	− 0.109845	0
− 0.161551	0	− 0.161555	0
0.010926	0	0.010925	0
0.102952	0.00004	0.10297	0
− 0.053466	0.00822	− 0.056293	0

Table 9 Return statistics on every instance for Dataset 3

GCS mean	GCS std.	ARMA mean	ARMA std.
0.05912	0	0.058494	0.00143
0.034659	0.07603	0.056641	0.08171
0.259772	0	0.259741	0.00002
− 0.064491	0	− 0.064491	0
0.125136	0.06479	0.152876	0
0.079529	0.07281	0.094402	0.05800
0.051029	0.09568	0.041275	0.05942
− 0.099974	0.10227	− 0.115771	0.04093
− 0.185021	0.13051	− 0.154651	0.02973
0.004418	0.06935	− 0.000504	0.05060
0.113949	0.09368	0.094835	0.04887
0.043611	0.09673	0.081722	0.13678

Table 10 Return statistics on every instance for Dataset 4

GAT mean	GAT std.	LSTM mean	LSTM std.
0.042279	0.01278	0.053117	0.02176
− 0.128956	0.00007	− 0.069907	0.04750
− 0.19507	0.00007	− 0.16372	0.10501
− 0.103258	0	− 0.013079	0.27055
0.114179	0	0.114179	0
0.084615	0	0.024966	0.03719
0.210526	0.00006	0.173922	0.10851
0.029786	0.03394	0.120206	0.10492
0.239829	0.00126	0.143658	0.12427
− 0.000763	0.00089	0.109348	0.08395
0.301698	0.06475	0.18956	0.06376
0.130126	0.04339	0.133377	0.03863

Table 11 Return statistics on every instance for Dataset 4

GCS mean	GCS std.	ARMA mean	ARMA std.
0.012129	0.04658	− 0.001259	0.04340
− 0.007884	0.09972	0.046798	0.10370
0.118145	0.21802	− 0.066621	0.15955
0.331967	0.34264	0.133187	0.25834
0.281846	0.20566	0.337363	0.35518
0.225618	0.04142	0.217437	0.11502
0.076641	0.08017	0.157311	0.13209
0.091074	0.13573	0.014852	0.08778
0.186322	0.09682	0.094761	0.11105
0.169197	0.06723	0.093558	0.10268
0.109906	0.11902	0.176996	0.09885
0.134762	0.07332	0.092671	0.08549

Acknowledgements The paper benefited from the comments of two anonymous reviewers.

Data availability The dataset we have used for this study is the same as the one used in [46], and it is publicly available by the writing of this paper https://host.uniroma3.it/docenti/cesarone/DataSets.htm. The four main datasets we have used from this archive are, Dow Jones 2005, Euro Bonds, Commodities and Italian Bonds Mix and World Bonds Mix datasets. No changes have been made to the datasets.

Declarations

Conflict of interest Both authors declare that they have no conflicts of interest.

References

1. Markowitz H (1952) Portfolio selection. J Finance 7(1):77–91. https://doi.org/10.2307/2975974
2. Gunjan A, Bhattacharyya S (2022) A brief review of portfolio optimization techniques. Artif Intell Rev 56(5):3847–3886. https://doi.org/10.1007/s10462-022-10273-7
3. Bartram SM, Branke J, Motahari M (2020) Artificial intelligence in asset management. SSRN Electron J. https://doi.org/10.2139/ssrn.3692805
4. Zhang Z, Zohren S, Roberts S (2019) Deep reinforcement learning for trading
5. Pham U, Luu Q, Tran H (2021) Multi-agent reinforcement learning approach for hedging portfolio problem. Soft Comput 25(12):7877–7885. https://doi.org/10.1007/s00500-021-05801-6
6. Pacreau G, Lezmi E, Xu J (2021) Graph neural networks for asset management. SSRN Electron J. https://doi.org/10.2139/ssrn.3976168
7. He Y, Gan Q, Wipf D, Reinert GD, Yan J, Cucuringu M (2022) Gnnrank: Learning global rankings from pairwise comparisons via directed graph neural networks. In: international conference on machine learning, pp 8581–8612. PMLR
8. Wang W, Li W, Zhang N, Liu K (2020) Portfolio formation with preselection using deep learning from long-term financial data. Expert Syst Appl 143:113042. https://doi.org/10.1016/j.eswa.2019.113042
9. Heaton JB, Polson NG, Witte JH (2018) Deep portfolio theory
10. Ma Y, Wang W, Ma Q (2023) A novel prediction based portfolio optimization model using deep learning. Comput Ind Eng 177:109023. https://doi.org/10.1016/j.cie.2023.109023
11. Yun H, Lee M, Kang YS, Seok J (2020) Portfolio management via two-stage deep learning with a joint cost. Expert Syst Appl 143:113041. https://doi.org/10.1016/j.eswa.2019.113041
12. Li X, Mulvey JM (2021) Portfolio optimization under regime switching and transaction costs: combining neural networks and dynamic programs. INFORMS J Optim 3(4):398–417. https://doi.org/10.1287/ijoo.2021.0053
13. Zhang Z, Zohren S, Roberts S (2020) Deep learning for portfolio optimization. J Financ Data Sci 2(4):8–20. https://doi.org/10.3905/jfds.2020.1.042
14. Horvath B, Muguruza A, Tomas M (2019) Deep learning volatility. SSRN Electron J. https://doi.org/10.2139/ssrn.3322085
15. Buehler H, Gonon L, Teichmann J, Wood B (2018) Deep hedging. SSRN Electron J. https://doi.org/10.2139/ssrn.3120710
16. Gierjatowicz P, Sabate-Vidales M, Siska D, Szpruch L, Zuric Z (2023) Robust pricing and hedging via neural stochastic differential equations. J Comput Finance. https://doi.org/10.21314/jcf.2022.025

17. Buehler H, Horvath B, Lyons T, Perez Arribas I, Wood B (2020) A data-driven market simulator for small data environments. SSRN Electron J. https://doi.org/10.2139/ssrn.3632431
18. Ni H, Szpruch L, Sabate-Vidales M, Xiao B, Wiese M, Liao S (2022) Sig-wasserstein gans for time series generation. In: proceedings of the second ACM international conference on AI in finance. ICAIF '21. Association for Computing Machinery, New York, NY, USA. https://doi.org/10.1145/3490354.3494393
19. Bartram SM, Branke J, Rossi GD, Motahari M (2021) Machine learning for active portfolio management. J Financ Data Sci 3(3):9–30. https://doi.org/10.3905/jfds.2021.1.071
20. Hu Y-J, Lin S-J (2019) Deep reinforcement learning for optimizing finance portfolio management. In: 2019 amity international conference on artificial intelligence (AICAI), pp 14–20. https://doi.org/10.1109/AICAI.2019.8701368
21. Betancourt C, Chen W-H (2021) Deep reinforcement learning for portfolio management of markets with a dynamic number of assets. Expert Syst Appl 164:114002. https://doi.org/10.1016/j.eswa.2020.114002
22. Lin Y-C, Chen C-T, Sang C-Y, Huang S-H (2022) Multiagent-based deep reinforcement learning for risk-shifting portfolio management. Appl Soft Comput 123:108894. https://doi.org/10.1016/j.asoc.2022.108894
23. Shi S, Li J, Li G, Pan P, Chen Q, Sun Q (2022) Gpm: a graph convolutional network based reinforcement learning framework for portfolio management. Neurocomputing 498:14–27. https://doi.org/10.1016/j.neucom.2022.04.105
24. Zhang C, Zhang Z, Cucuringu M, Zohren S (2021) A universal end-to-end approach to portfolio optimization via deep learning
25. Bertsimas D, Cory-Wright R (2022) A scalable algorithm for sparse portfolio selection. INFORMS J Comput 34(3):1489–1511. https://doi.org/10.1287/ijoc.2021.1127
26. Kim R, So CH, Jeong M, Lee S, Kim J, Kang J (2019) HATS: a hierarchical graph attention network for stock movement prediction
27. Feng F, He X, Wang X, Luo C, Liu Y, Chua T-S (2019) Temporal relational ranking for stock prediction. ACM Trans Inf Syst 37(2):1–30. https://doi.org/10.1145/3309547
28. Lim B, Arik SO, Loeff N, Pfister T (2020) Temporal fusion transformers for interpretable multi-horizon time series forecasting
29. Tian H, Zheng X, Zhao K, Liu MW, Zeng DD (2022) Inductive representation learning on dynamic stock co-movement graphs for stock predictions. INFORMS J Comput 34(4):1940–1957. https://doi.org/10.1287/ijoc.2022.1172
30. Zhang C, Pu X, Cucuringu M, Dong X (2023) Graph neural networks for forecasting realized volatility with nonlinear spillover effects. SSRN Electron J. https://doi.org/10.2139/ssrn.4375165
31. Wang X, Wang Y, Weng B, Vinel A (2020) Stock2Vec: a hybrid deep learning framework for stock market prediction with representation learning and temporal convolutional network
32. Gorduza D, Dong X, Zohren S (2022) Understanding stock market instability via graph auto-encoders
33. Zhang C, Pu X, Cucuringu M, Dong X (2022) Graph-based methods for forecasting realized covariances. SSRN Electron J. https://doi.org/10.2139/ssrn.4274989
34. Lim B, Zohren S (2021) Time-series forecasting with deep learning: a survey. Philos Trans R Soc A Math Phy Eng Sci 379(2194):20200209. https://doi.org/10.1098/rsta.2020.0209
35. Borovykh A, Bohte S, Oosterlee CW (2018) Conditional time series forecasting with convolutional neural networks
36. Hochreiter S, Schmidhuber J (1997) Long short-term memory. Neural Comput 9(8):1735–1780. https://doi.org/10.1162/neco.1997.9.8.1735
37. Veličković P, Cucurull G, Casanova A, Romero A, Liò P, Bengio Y (2018) Graph attention networks
38. Grattarola D, Alippi C (2020) Graph neural networks in tensorflow and keras with spektral
39. Bianchi FM, Grattarola D, Livi L, Alippi C (2021) Graph neural networks with convolutional arma filters. IEEE Trans Pattern Anal Mach Intell. https://doi.org/10.1109/tpami.2021.3054830
40. Kalayci CB, Ertenlice O, Akbay MA (2019) A comprehensive review of deterministic models and applications for mean-variance portfolio optimization. Expert Syst Appl 125:345–368. https://doi.org/10.1016/j.eswa.2019.02.011
41. Torri G, Giacometti R, Paterlini S (2019) Sparse precision matrices for minimum variance portfolios. Comput Manag Sci 16(3):375–400. https://doi.org/10.1007/s10287-019-00344-6
42. Nguyen VA, Kuhn D, Mohajerin Esfahani P (2022) Distributionally robust inverse covariance estimation: the wasserstein shrinkage estimator. Oper Res 70(1):490–515. https://doi.org/10.1287/opre.2020.2076
43. Wang Y, Aste T (2023) Dynamic portfolio optimization with inverse covariance clustering. Expert Syst Appl 213:118739. https://doi.org/10.1016/j.eswa.2022.118739
44. Hallac D, Park Y, Boyd S, Leskovec J (2017) Network inference via the time-varying graphical lasso
45. Tegner G (2018) Recurrent neural networks for financial asset forecasting. Master's thesis, KTH, Mathematical Statistics
46. Çağın Ararat Cesarone F, Çelebi Pınar M, Ricci JM (2021) MAD risk parity portfolios
47. Diamond S, Boyd S (2016) CVXPY: a python-embedded modeling language for convex optimization. J Mach Learn Res 17(83):1–5