Predictive Portfolio Optimization Group Work Report

Jiyu Liu, Jundong Zhou, Zhaoyu Zeng
Shanghai Jiao Tong University
{520030910342, 520030910341, 520030910344}

Abstract

Recently, we have seen a lot of applications of self-supervised neural networks on portfolio optimization problems. Based on the works of predecessors, our works integrated prediction and optimization and incorporated the constraints into the network architecture and computational operators. Furthermore, by the implementation of self-attention layers and MogLSTM, we enhanced the neural network's economic performance. In addition, we attempted to explain why combining optimization layers with neural networks does not result in performance enhancement.

1. Introduction

In recent years, deep learning-based methods have achieved great success in different areas such as computer vision, natural language processing, and also, research and applications in the finance and banking (F&B) field, including portfolio optimization problems [20]. Portfolio optimization is a substantial problem in quantitative finance based on the modern portfolio theory proposed by [22]. It can be considered a combinatorial optimization (CO) problem of maximizing the expected return while minimizing risk.

The portfolio assignment consists of prediction and data-based optimization. Although it's straightforward to forecast and solve optimization problems separately in order, the joint-prediction-and-optimization method shows better performance and practicability. Works combining prediction with the feedback from optimization problem [7], [31] has implemented portfolio problem in various approaches, among which [3] provides a systematical pipeline which encodes cardinality constraints into the network architecture.

In this project, we conduct our research based on [3]. The contributions are listed as follows:

- **Jiyu Liu:** Research about CVXPY solvers (see Section 2.1) and implementation of differentiable convex optimization layer (see Section 3.2 and Section 4.1).
- **Jundong Zhou:** Implementation of MogLSTM (see Section 2.2, 3.3 and 4.2), expansion to the A-share stock market (see Section 3.5, 4.4), and augmentation of more realistic constraints (see Section 3.6, 4.5).
- **Zhaoyu Zeng:** Implementation of self-attention layers (see Section 2.2, 3.4 and 4.3), and other explorations (see Section 4.6).

2. Related Work

2.1. CVXPY Solvers

Due to widely application of convex optimization [4], many optimization solvers have been released in the last decade, including the interior-point solver ECOS [14], and the first-order method-based solver SCS [26]. While ECOS is competent at solving small or medium-sized problems (up to tens of thousands of variables) quickly and precisely, SCS can handle the problem on large scale with modest accuracy. ECOS is designed for second-order cone programming (SOCP), and its main algorithm is a standard primal-dual Mehrotra predictor-corrector method with Nesterov-Todd scaling and self-dual embedding. SCS is well-suited for almost every problem that can be expressed in the form of a cone program, including linear programming (LP), SOCP, semidefinite programming (SDP), etc. [26] proposes an operator splitting method named the alternating direction method of multipliers (ADMM) for solving the homogeneous self-dual embedding problem. Compared to other first-order methods, SCS does not rely on any explicit algorithm parameters and outperforms them on computing cost.

CVXPY [13] is a python-embedded modelling language for convex optimization. It uses a domain-specific language (DSL) which allows its users to express convex optimization problems in a natural syntax following the mathematics. This specification is then automatically converted into the standard form required by generic solvers, such as the above-mentioned open-source solvers ECOS and SCS, as well as the commercial solvers GUROBI [18] and MOSEK [24]. Based on CVXPY, the *cvxpylayer* [2] provides an interface that combines CVXPY solvers [13] with deep learning frameworks like PyTorch [27], TensorFlow [1] and Jax

[5], enabling its users to construct differentiable convex optimization layers. Besides, [2] introduces the disciplined parameterized programming (DPP) as a grammar and the affine-solver-affine form (ASA) which make it possible to differentiate through DSLs for convex optimization. So far, ECOS and SCS are the only two available solvers in *cvxpylayer*.

2.2. Neural Networks

Based on timing properties, neural network applications in finance require the ability to remember. In 1997, [19] proposed *long-short-term memory* (LSTM) as a special kind of recurrent neural network (RNN). Following this, a great deal of work has been devoted to improving LSTM, and also applying it in different areas, for example, natural language processing (NLP) [23, 33] and stock market prediction [9, 25, 30]. The common structure of a general LSTM unit contains a cell, an input gate, an output gate and a forget gate. The cell remembers values over arbitrary time intervals and the three gates regulate the flow of information in and out of the cell. In practice, hundreds of cells are connected to process a long sequence of data.

In addition to LSTM, [32] proposed *Transformer* relying entirely on the attention mechanism rather than recurrence. This innovation has caused quite a stir and a large number of works sprang up in recent years [6, 12, 28, 29]. Meanwhile, the attention mechanism also demonstrates outstanding capabilities in computer vision (CV) after [15] proposed *Vision Transformer* (ViT), for instance, the Swin transformer [21], the Pyramid vision transformer [34], and the Crossvit [8], etc. We can see that numerous studies have proven the capabilities of the attention mechanism.

3. Method

3.1. Preliminary

Problem setting Assume $Y = \{y^1, \cdots, y^m\}$ is a dataset of uncertain quantities of interest, such as assets. $y^i \in \mathbb{R}^{n_d}$ represents the closing price of asset i in n_d days. $X \in \mathbb{R}^m$ implies the investment decision on the m assets. $c = f(X, Y)$ is a function to evaluate the result of decision X on assets Y.

To eliminate the complex relation caused by the different prices of all assets, the closing price is translated into the rise and fall rate of stock by day, so as the feature sequence of each asset. $Z = \{Z^1, ..., Z^m\}$ is used to represent the translated dataset that contains all m assets.

Based on Z, μ and Σ are used to conclude the data in Z in a period. $\mu \in \mathbb{R}^m$ represents the average return ratio of an asset, and $\Sigma \in \mathbb{R}^{m \times m}$ represents the covariance matrix of the assets.

The Pipeline of baseline method We illustrate our baseline method in figure 1 and introduce the pipeline in the following paragraph.

Firstly, the aforementioned translated dataset Z is fed to the problem encoder and gets the prediction of prices in the future and a hidden state h. Then the probability of selecting each vector s is calculated from h through a fully-connected layer: $s = \text{fully-connected}(h)$. After that, enforce the cardinality constraint by a Sinkhorn layer [10] and get the normalized probability T. Then solve the optimization problem with the predicted price and the probability T as parameters to get the weight vector x. Finally, check the Sharpe ratio of selected decision x in ground truth prices and calculate the loss.

3.2. Convex Optimization Layer

In this section, we introduced a method that combines the neural network with convex optimization solvers.

Overview In the Objective Estimator step, we determine the weight of each asset (designated as $\mathbf{x}$) by solving the convex optimization problem below.

$$\max_x \frac{(\mu - r_f)^\top \mathbf{x}}{\sqrt{\mathbf{x}^\top \Sigma \mathbf{x}}}, \qquad s.t. \quad \sum_{i=1}^{m} x_i = 1, x \geq 0, \|x\|_0 \leq k$$

where r_f means risk-free return, e.g. U.S. treasury bonds. Note μ and Σ are unknown at the time of decision-making, and they are predicted by our network. The main contribution of our work is to convert the closed-form solution of $\mathbf{x}$ to the cvxpy solution, allowing us to directly calculate $\mathbf{x}$ with cardinality constraints.

Closed-form Solution Our origin method is estimating $\mathbf{x}$ by leveraging a closed-form solution of unconstrained Eq. $\mathbf{x} = \Sigma^{-1}(\mu - r_f)$, and then enforcing constraints: $\mathbf{x} = \text{ReLU}(\mathbf{x} \odot \widetilde{\mathbf{T}}_i[2, :])$, $\mathbf{x} = \mathbf{x}/sum(\mathbf{x})$. After obtaining $\mathbf{x}$, we compute Sharpe ratio based on $\mathbf{x}$ and on ground truth μ^{gt}, Σ^{gt}, and use this Sharpe ratio as supervision: $\widetilde{J}_i = ((\mu^{gt} - r_f)^\top \mathbf{x})/(\sqrt{\mathbf{x}^\top \Sigma_{gt} \mathbf{x}})$.

Although a closed-form solution requires no iteration and is fast to calculate, finding the optimal solution without constraints and then projecting it onto the constraint plane will unavoidably result in a loss of accuracy.

CVXPY Solution Cxvpylayer allows us to solve the optimization problem with constraints while returning the derivative of the solution with respect to the parameter. To pass the problem to CVXPY solvers, we rewrite the problem to find a portfolio weight vector $\mathbf{x}$ that minimizes the sum of squares of the product of Σ and $\mathbf{x}$ subject to the constraints below.

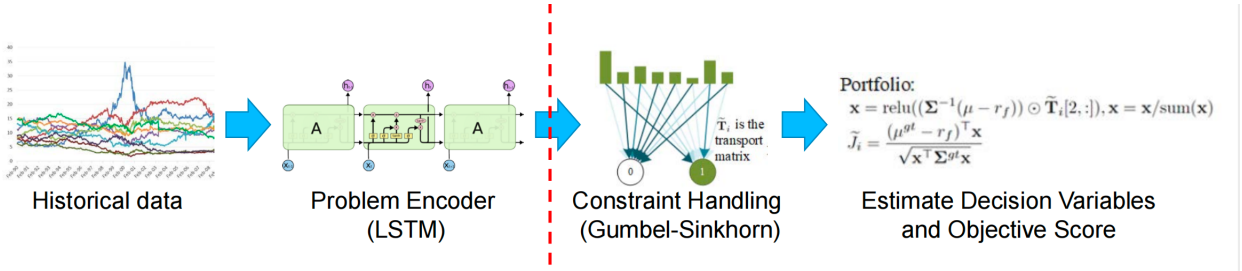


Figure 1. The pipeline of the baseline algorithm. We improve the second part in the pipeline in Sec. 3.3 and 3.4, expand the third part in Sec. 3.6 and promote the last part in Sec. 3.2.

(1) Limiting the concentration of the portfolio to any individual asset.

(2) Requiring the weights to be non-negative.

(3) Fixing the sum of all returns of the portfolio.

Formulating these problems in mathematical form as below:

$$\min_{x} \quad \sum_{i}^{m}(\sum_{j}^{m}\Sigma_{ij}\mathbf{x}_j)^2$$

$$s.t. \quad \mathbf{x} \leq sum(\mathbf{x}) \cdot \widetilde{\mathbf{T}}_i[2,:], \quad \mathbf{x} \geq 0, \quad \mathbf{x}^T(\mu - r_f) = 1$$

3.3. MogLSTM

MogLSTM [23] noticed that the input and the hidden layer interacted with each other only in the LSTM cell, which may lead to losses of information. To tackle this problem, MogLSTM makes the two inputs x and h_{prev} modulate one another in an alternating function before the usual LSTM computation takes place (see Fig. 3). By adding this amendment, the new LSTM achieved state-of-the-art performance in various datasets ranging from small case to large case.

Inspired by the previous work, I replace LSTM in the pipeline with MogLSTM. In this work, x and h represents the price of assets and the constraint on the proportion of assets respectively. To improve the precision of price prediction and constraints, I use MogLSTM to make the current price interact with previous constraints before entering the LSTM cell.

3.4. Self-Attention Layer

According to [32], the *scaled dot-product attention* swallows the input queries and keys (of dimension d_k) and values (of dimension d_v) and then computes the attention function. Denoting the packed queries as a matrix Q, and the packed keys and values as matrices K and V, we have the outputs computed as:

$$O_{\text{Attention}}(Q,K,V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V. \qquad (1)$$

Based on the *scaled dot-product attention*, we can implement the *multi-head attention* for jointly attending to information from different representation subspaces at different position [32]. With the denotation of parameter matrices $W_i^Q \in \mathbb{R}^{d_{\text{model}} \times d_k}$, $W_i^K \in \mathbb{R}^{d_{\text{model}} \times d_k}$, $W_i^V \in \mathbb{R}^{d_{\text{model}} \times d_v}$, $W^O \in \mathbb{R}^{hd_v \times d_{\text{model}}}$, we have the *multi-head attention* in the form of

$$O_{\text{head}_i} = O_{\text{Attention}}(QW_i^Q, KW_i^K, VW_i^V);$$
$$O_{\text{MultiHead}}(Q,K,V) = \text{Concat}(O_{\text{head}_1}, \ldots, O_{\text{head}_h})W^O.$$

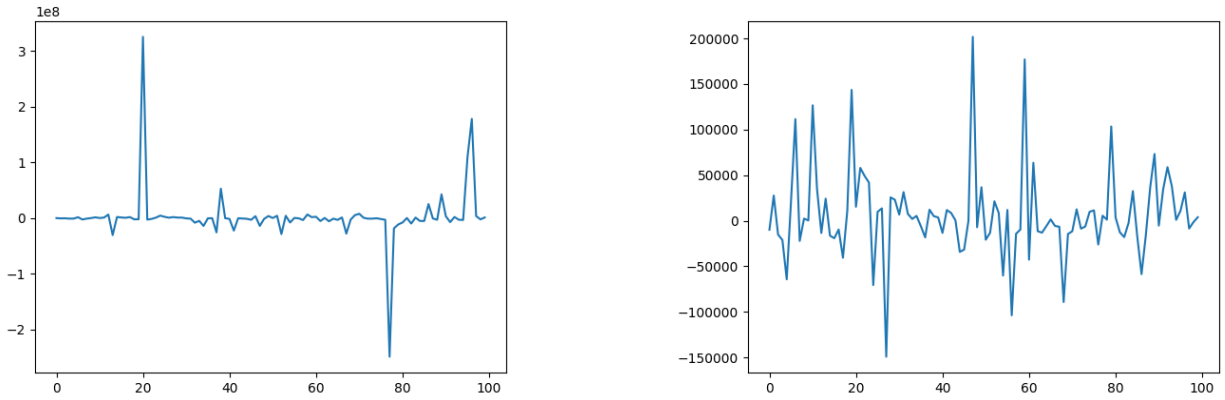
In practice, we put the multi-head self-attention layer between the encoder LSTM and the decoder LSTM to enhance observation and screening of the features. Also, we expect the self-attention layer to help the fit a more complex share price model and thus give a more accurate share price forecast.

3.5. Expansion to A-share

A-shares are closer to our lives than snp500, so the performance of the algorithm on A-shares is also very noteworthy. Compared to snp-500, A-share consists of more assets and each asset is subject to a maximum limit of 10% upside and downside, which imposes more challenge on portfolio optimization. A statistical website [11] is used to acquire the closing price of each specific asset.

3.6. Realistic Constraints

In practice, there are more political and economic constraints on the stock market. [17] listed various regulations and analyzed the impact of these constraints on portfolio return in a realistic case. Common constraints, including restriction on short selling, the limit of trading lots, transaction costs and the upper bound of rise and fall rate, generally decrease the risk of investment, but the impact on the return ratio is quite uncertain. To certify the influence of these constraints on our case, we add simulated regulations on the variables to see how the Sharpe ratio change.



(a) Sum of gradients in closed-form

(b) Sum of gradients in CVXPY model

Figure 2. Comparison between the sum of gradients in closed-form and CVXPY solution

solution method	scale	iterations	tolerance	lookback	Result
closed-form	100	3000	1e-10	3	2.1928
CVXPY	100	3000	1e-10	3	1.7686
CVXPY	100	10000	1e-8	0	1.8287
closed-form	500	3000	1e-10	3	1.8125
CVXPY	500	3000	1e-10	3	1.7116
CVXPY (topk)	500	3000	1e-10	3	1.8080

Table 1. Comparisons of result (Sharpe ratio) with different solution methods, number of assets and other solver arguments.

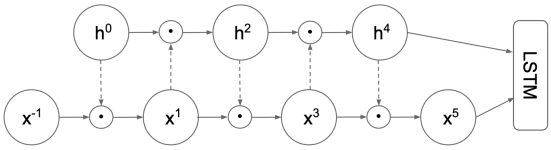


Figure 3. MogLSTM with 5 rounds. The previous hidden layer $h^0 = h_{prev}$ and $x^{-1} = x$ are the original input of LSTM. The dotted line represents a linear transform and the signal $\odot$ represents the elementwise product and activation function. After several repetitions of this mutual gating cycle, the last values of h^* and x^* sequences are fed to an LSTM cell.

4. Experiment

4.1. Implementation of cvxpylayer

As we mentioned before, the cvxpylayer now provides only two kinds of solvers: SCS and ECOS. In cvxpylayer, ECOS can only be used to solve problems that don't use the positive semidefinite or exponential cone. This roughly means that your problem cannot execute any operations like exp, log, etc. or contain any positive semidefinite matrices. We are not able to use ECOS since our optimization approach inherently includes $Sigma$. Although SCS has a great capacity for handling large-scale problems, to use cvxpylayer, we have to transform the problem into DPP form, which has a stringent limitation on the number of parameters. Currently, only problems with parameter size up to a few thousand can be efficiently compiled into DPP form. When compiling a problem into DPP form for problems with larger parameter sizes, cvxpylayer may incur significant costs in performance. In the case of our 500 assets portfolio optimization problem, the parameter size reaches 2e5, which results in prohibitive compilation cost and fairly low accuracy in default maximum iterations.

During the experiment, I encountered two major problems, one of them is that the model has too many parameters so the optimization layer can't work efficiently; the other is that the return gradients in the cvxpylayer might be more unsteady than in the closed-form one.

First, due to the tail values making up a significant portion of our input parameters, I decided to cut them off when constructing cvxpylayer and then recover them before computing the Sharpe ratio. The output is very close to the original method. Secondly, I tried to reduce the number of assets to 100 so that cvxpylayer could compile it into DPP form efficiently. However, their outputs all appeared

to be unreliable since a lack of information or iterations. Third, assuming that parameter scales can only influence the running time of optimization (this has been proved to be wrong, large parameter scale have a negative influence on not only time overhead but accuracy), I decided to run the cvxpylayer training with 500 assets directly for maintaining the information, yet the result didn't converge after 100 epochs. The result of these experiments can be seen in Table 1. Besides, I visualized the Sharpe ratio of the predicted portfolio in 100 assets in Fig. 4. Then, I tried to adjust the hyper-parameters in the cvxpylayer. As recommended, I set maximum iterations to 10000, tolerance to 1e-8 and acceleration lookback to 0 .(The tolerance determines the level of accuracy or precision that is desired in the solution of the optimization problem and the acceleration lookback determines the number of previous iterations to use when constructing an accelerated iteration of the solver.) It does bring some performance improvements, even though, the CVXPY model is still inferior to the closed-form one.

After facing so many troubles when using cvxpylayer, I started to consider that reinitialization of parameters per iteration might cause the instability of gradients from the cvxpylayer. To prove my assumption, I visualized the sum of model gradients from closed-form and cvxpylayer solution in Figure 2. As we can see in the figure, the closed-form solution has a more steady gradient and the sum of gradients is typically concentrated around zero; however, the gradient of CVXPY solution is more unstable and prone to oscillate, which explains why our model with cvxpylayer has a convergence problem.

Despite numerous efforts, we have yet to discover a method for improving the performance of our model using the cvxpylayer. Upon further investigation of the optimization layer and analysis of experimental results, we have identified three potential reasons for the inferior performance of our model:

1. To be efficiently transformed into DPP form, our problem has too many parameters. The developers of CVXPY recommended a parameter number below 1e4, yet the parameter number of our 500 assets is up to 2e5.

2. Every time we run the Objective Estimator step, we need to reinitialize the cvxpylayer. This means that the gradient calculated in the back-forward step is highly dependent on the initialization parameters, which leads to instability in the training of the model.

3. Because of the tight tolerance, the optimization problem may be considered infeasible or inaccurate by the SCS solver, so the accuracy of the cvxpylayer cannot be assured.

4.2. Implementation of MogLSTM

On replacing LSTM with MogLSTM, problems with dimension occur frequently. To handle this problem, I output the shape of each tensor and reshape them into the correct dimension. Table 4 shows the best performance in 55 epochs compared to the baseline. As the table shows, mutual interaction improves the accuracy of each other and leads to better performance in general.

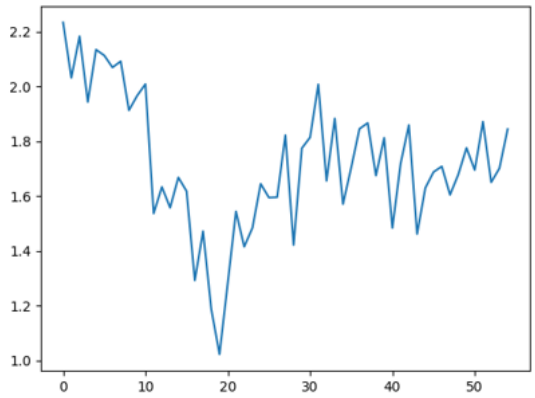


Figure 4. The Sharpe ratio curve during 55 training epochs with 100 assets. Notice that the curve is extremely unsteady and generally the training process does not help the prediction.

4.3. Implementation of Self-Attention Layer

To evaluate the performance of the inserted multi-head self-attention layer, we implement a two-stage experiment. The results and discussion are as follows.

Firstly, we try different learning rates for a model with 4 heads to select a relatively appropriate learning rate for the next experiments. Table 2 shows the results and points to the learning rate of 0.001. Secondly, we test the model on different numbers of heads under the selected learning rate. Table 3 shows the experiment results. We can see that the model achieves the highest Sharpe ratio with 8 heads but achieves the lowest prediction mean square error (MSE) with 12 heads (which points to a more accurate prediction). It appears that model with more self-attention heads gives more accurate share price forecasts, but the accuracy of the forecast seems not positively correlated with the Sharpe ratio of the final resulting portfolios.

lr	pred MSE	Sharpe ratio
0.00500	19.5652	1.2425
0.00200	1.0676	1.6305
0.00100	0.4707	**1.8786**
0.00075	0.2240	1.4399
0.00050	**0.1804**	1.5411

Table 2. Results of different learning rates with 4 self-attention heads.

num_heads	pred MSE	Sharpe ratio
4	0.4707	1.8786
6	0.2229	1.7750
8	0.2897	**2.1126**
12	**0.1819**	1.7207

Table 3. Results of different numbers of self-attention heads under learning rates set as 0.001.

4.4. Expansion to A-share

Firstly, acquire all the codes of stocks in the hanghai Stock Exchange Index from the official website [16] using a crawler program. Secondly, input these codes into the notebook of [11] by batch to get the closing price. Then get rid of assets that are listed after January 1, 2018, and delist before December 31, 2021, to make sure each asset has the same length of the sequence of price. Due to the large data volume, the data is fed into our algorithm increasingly. Table 4 illustrates that with the increasing of assets, our algorithm achieves better performance because of more choices. And MogLSTM shows more advantage on A-share than on snp-500.

Backbone	Dataset	Sharpe ratio
LSTM	snp-500	1.9202
MogLSTM	snp-500	2.2074
LSTM	600000 to 600600 of A-share	0.6320
MogLSTM	600000 to 600600 of A-share	1.4893
LSTM	600000 to 602000 of A-share	0.9824
MogLSTM	600000 to 602000 of A-share	1.6709

Table 4. Comparison of results with different backbone network structures in different datasets. The elements in the table are the best Sharpe ratio in 55 train-test iterations.

4.5. Implementation of more realistic constraints

We discuss and implement the aforementioned four constraints respectively.

1. **Ban on short selling.** It means that each element in x should be non-negative. It is guaranteed by Sinkhorn and a *ReLU* layer.

2. **Limit of trading lots.** Though this limitation mainly works on short-term trading, it impacts long-term trading with minimum volume. It is reflected as the lower bound of x in the code. Table 5 shows the influence of setting a lower bound. Compared with the result in Table 4, setting a lower bound increase the performance slightly.

3. **Transaction costs.** It also mainly works on short-term trading. The cost can be represented as $\gamma||x_{i+1} - xi||_0$ in the code where γ denotes the handling fee rate. Since this is a mandatory constraint to maintaining the stability of the market that decreases the return ratio, it is pointless to compare the result with other methods, therefore we did not implement it in the code.

4. **Upper bound of rise and fall rate.** The A-share stock market is a proper example of exploring this limit because it has an upper bound of 10% on the rise and fall rate. As Table 4 illustrates, the limit will cause a reduction in the return ratio under the same portfolio strategy.

Backbone	Lower Bound	Sharpe ratio
LSTM	1e-3	1.9917
LSTM	1e-2	1.9989
MogLSTM	1e-3	2.2375
MogLSTM	1e-2	2.2816

Table 5. Results of setting different lower bound on x with different backbone on snp-500 dataset. The result represents the best performance in 55 train-test epochs.

4.6. Other Explorations

Before evaluating the self-attention layer, we have tried other methods to deal with the features extracted by the encoder LSTM. In the origin pipeline, it is quite confusing that the way to deal with the latent feature is too simple to handle the long-term stock price sequences. As the result, the prediction MSE is huge in the original experiment result reported in [3]. To explore this problem, we designed several experiments as follows.

Firstly, we tried to use more feature dimensions than just the latest one by computing the average of the last N days' latent features as the input of the decoder LSTM. Table 6 shows the results. As we can see, the results are highly irregular and not interpretable.

N	pred MSE	Sharpe ratio
30	1.0600	1.6510
45	**0.1963**	1.5761
60	0.4943	**2.2129**
75	1.7726	1.8138
90	1.1130	1.5291
120	1.4283	1.5915

Table 6. Results of using average latent features as the input of the decoder LSTM.

Secondly, we similarly tried the weighted average method. We separate the latest 60 days of the latent fea-

ture into three parts:

$$\mathbf{z}_1 = \text{feature}[-10:,:,:],$$
$$\mathbf{z}_2 = \text{feature}[-30:-10,:,:],$$
$$\mathbf{z}_3 = \text{feature}[-60:-30,:,:].$$

Then, we calculate the weighted average in the following form:

$$\mathbf{z} = \alpha_1\bar{\mathbf{z}}_1 + \alpha_2\bar{\mathbf{z}}_2 + \alpha_3\bar{\mathbf{z}}_3,\ \alpha_1+\alpha_2+\alpha_3 = 1.$$

Table 7 shows the results, but the results are still not good. These results all seem to point to the original feature processing method being better.

α_1	α_2	α_3	pred MSE	Sharpe ratio
0.7	0.2	0.1	0.6638	1.9008
0.6	0.3	0.1	0.8669	**2.0119**
0.5	0.4	0.1	**0.4876**	1.5258
0.6	0.2	0.2	0.5228	1.7548

Table 7. Results of using weighted average latent features as the input of the decoder LSTM.

However, it is worth noting that most of the attempts based on average reach lower prediction MSE than the original method. So, we still think that there exist some better methods to deal with latent features. From this, we think about the *self-attention layers*.

5. Conclusion

In this report, We present the results of our research on this project and express some of our understanding of this project. We give our conclusion list as follows:

- **Jiyu Liu:** Due to the limitations of the existing functionality of cvxpylayer, it's very inefficient to convert our optimization problem to DPP form, thus leading to unguaranteed precision. Furthermore, to reinitialize the parameters per iteration also results in unstability of return gradients. However, We might be able to get an improvement if we use CVXPY solvers to address the problem while maintaining the gradients' form.
- **Jundong Zhou:** A plain-thinking improvement of LSTM is implemented in the algorithm to replace the original LSTM, which promotes the performance obviously showed by experiments. Dataset is expanded to A-share to check the expandability of the algorithm. More realistic constraints are imposed on the process of algorithm to enhance the generalizability in reality.
- **Zhaoyu Zeng:** Experiments shows that the *self-attention layer* indeed helps present more accurate stock price predictions while maintaining the final Sharpe ratio. However, to further increase the Sharpe ratio, more research should be done on not only the LSTM model structures but also the other regions.

References

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*, 2016. 1

[2] A. Agrawal, B. Amos, S. Barratt, S. Boyd, S. Diamond, and Z. Kolter. Differentiable convex optimization layers. In *Advances in Neural Information Processing Systems*, 2019. 1, 2

[3] Anonymous. Relaxed combinatorial optimization networks with self-supervision: Theoretical and empirical notes on the cardinality-constrained case. In *Submitted to The Eleventh International Conference on Learning Representations*, 2023. under review. 1, 6

[4] Stephen Boyd, Stephen P Boyd, and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004. 1

[5] James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. JAX: composable transformations of Python+NumPy programs, 2018. 2

[6] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020. 2

[7] Andrew Butler and Roy H. Kwon. Integrating prediction in mean-variance portfolio optimization, 2021. 1

[8] Chun-Fu Richard Chen, Quanfu Fan, and Rameswar Panda. Crossvit: Cross-attention multi-scale vision transformer for image classification. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 357–366, 2021. 2

[9] Kai Chen, Yi Zhou, and Fangyan Dai. A lstm-based method for stock returns prediction: A case study of china stock market. In *2015 IEEE International Conference on Big Data (Big Data)*, pages 2823–2824, 2015. 2

[10] Marco Cuturi. Sinkhorn distances: Lightspeed computation of optimal transport. In C.J. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 26. Curran Associates, Inc., 2013. 2

[11] DataYes. [EB/OL], 2023. https://uqer.datayes.com. 3, 6

[12] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018. 2

[13] Steven Diamond and Stephen Boyd. Cvxpy: A python-embedded modeling language for convex optimization. *The*

Journal of Machine Learning Research, 17(1):2909–2913, 2016. 1

[14] Alexander Domahidi, Eric Chu, and Stephen Boyd. Ecos: An socp solver for embedded systems. In *2013 European Control Conference (ECC)*, pages 3071–3076. IEEE, 2013. 1

[15] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020. 2

[16] Shanghai Stock Exchange. Shares and depository receipts, 2023. http://www.sse.com.cn/assortment/stock/list/share/. 6

[17] E. Grizickas Sapkute, M. A. Sánchez-Granero, M. N. López García, and J. E. Trinidad Segovia. The impact of regulation-based constraints on portfolio selection: The Spanish case. *Humanities and Social Sciences Communications*, 9(1):310, Sept. 2022. 3

[18] LLC. GUROBI Optimization, 2023. https://www.gurobi.com/. 1

[19] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997. 2

[20] Jian Huang, Junyi Chai, and Stella Cho. Deep learning in finance and banking: A literature review and classification. *Frontiers of Business Research in China*, 14(1):1–24, 2020. 1

[21] Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. Swin transformer: Hierarchical vision transformer using shifted windows. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 10012–10022, 2021. 2

[22] Harry Markowitz. Portfolio selection. *The Journal of Finance*, 7(1):77–91, 1952. 1

[23] Gábor Melis, Tomáš Kočiský, and Phil Blunsom. Mogrifier lstm. In *International Conference on Learning Representations*, 2020. 2, 3

[24] Mosek, 2023. https://www.mosek.com/. 1

[25] David MQ Nelson, Adriano CM Pereira, and Renato A De Oliveira. Stock market's price movement prediction with lstm neural networks. In *2017 International joint conference on neural networks (IJCNN)*, pages 1419–1426. Ieee, 2017. 2

[26] Brendan O'Donoghue, Eric Chu, Neal Parikh, and Stephen Boyd. Conic optimization via operator splitting and homogeneous self-dual embedding. *Journal of Optimization Theory and Applications*, 169(3):1042–1068, June 2016. 1

[27] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019. 1

[28] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019. 2

[29] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, Peter J Liu, et al. Exploring the limits of transfer learning with a unified text-to-text transformer. *J. Mach. Learn. Res.*, 21(140):1–67, 2020. 2

[30] Murtaza Roondiwala, Harshal Patel, and Shraddha Varma. Predicting stock prices using lstm. *International Journal of Science and Research (IJSR)*, 6(4):1754–1756, 2017. 2

[31] Sanket Shah, Kai Wang, Bryan Wilder, Andrew Perrault, and Milind Tambe. Decision-focused learning without differentiable optimization: Learning locally optimized decision losses, 2022. 1

[32] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017. 2, 3

[33] Lukai Wang and Lei Wang. A case study of chinese sentiment analysis on social media reviews based on lstm, 2022. 2

[34] Wenhai Wang, Enze Xie, Xiang Li, Deng-Ping Fan, Kaitao Song, Ding Liang, Tong Lu, Ping Luo, and Ling Shao. Pyramid vision transformer: A versatile backbone for dense prediction without convolutions. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 568–578, 2021. 2