Stock Price Prediction With Convex Neural Nets

Matthew Aguilar msa013@ucsd.edu
Jake Millhiser jmillhis@ucsd.edu
John O'Boyle jboboyle@ucsd.edu
William Sharpless wsharpless@ucsd.edu

Abstract

Stock prediction has long been a subject of interest among traders and statisticians alike. Recent advances in machine learning have sparked interest in utilizing neural networks to do stock prediction and uncover the hidden trends that run the stock market. However, despite this, stocks are still well-known to be difficult to predict due to vast number of socio-economic and political influences. Neural network training is still not without its issues. Networks can still get stuck in locally optimum weights due to non-smooth, non-convex objective functions. Motivated by Ergen's and Pilanci's recent work, this paper studies the application of a reformulated convex neural network on stock data to determine if better stock prediction performance could be obtained. Experiments were run comparing the convex and non-convex networks on various metrics. Additionally, a trading bot was created to simulate the feasibility of returning a profit with the convex network.

1. Introduction

Stock prediction has long been studied by statisticians throughout history as a means to maximize profits. Yet, stocks remain one of the most difficult signals to predict due to rapid cascades induced by socio-economic and political actions. With the rise of the perceptron [21], and the view of multi-layer neural nets as *universal function approximiators* [12], the idea of applying machine learning to stock market prediction became sensible. Additionally, recent advances in computation have allowed for training deep neural networks on GPU's to reach relatively fast convergence. However this training is still not without its faults. Often, the optimization landscapes of neural networks are non-smooth and non-convex, allowing for the presence of many local optima for network training to converge to. Initial weights can greatly change which set of optimal weights a network yields. This makes it unclear which set of weights would lead to a network that generalizes well to unseen data. Multiple networks are often required to be trained with different sets of randomly initialized weights, with the best model being selected amongst those. In the domain of Optimization, non-smooth and non-convex objectives are often replaced with surrogate *convex functions* which provide a single global optimum, avoiding the potential of getting trapped in a non-optimal solution and making the problem more computational tractable. The use of these convex objectives in neural networks could provide these same benefits to the training procedure. Recent work by Ergen and Pilanci have demonstrated that a 2-layer linear network with rectified linear unit (ReLU) activation could indeed be reformulated as a convex program which will have the same globally optimal weights [19].

Motivated by this work, this paper implements Pilanci's and Ergen's novel convex neural network approach to find the hidden characteristics in historical stock data. We hypothesize that applying this result to stock data may provide better predictive power in predicting financial patterns than implementing standard a 2-layer ReLU on the same data. By combining the power of neural networks with tools of the finance industry, a trading bot is created to take the knowledge of historical stock data and predict the future movement of the price. The network predictions will be used buy bot to drive buy/sell decisions, and we will finalize the results by comparing the profits made by the two network implementations.

1.1. History

Although evidence of stocks and bonds being traded can be found before the 1600's, the first highly profitable stock exchange began in 1611 Amsterdam with the Dutch East India Company. Since its inception, investors attempted to predict which stocks would yield the highest return and which were not worth the investment. The opportunity to make money from investments has birthed countless methods to game the market over the past few centuries, and the first is seen is in 1688 with the beginning of technical analysis [24].

The stock market is known for being difficult, and often impossible, to predict. This ideology is driven by what is called the Efficient Market Hypothesis (EMH), which states that the current price of a stock is the reflection of all available information with nothing left to predict [7]. According to the EMH, stocks always trade at their fair value, so it is

impossible to outperform the market since there is no belief in the existence of under or overvalued stocks. However, this is contrary to the methods of many popular investors, such as Warren Buffet, who believe in "value investing," where the approach is to invest in undervalued companies [11] and have a proven record of making profitable investments and trades. Value investing suggests that there may be characteristics of a stock that indicates its future movements. It is these characteristics that our method strives to discover.

1.2. Data

Historical stock data is collected using both the python-based Robinhood API (Robin Stocks) [8] and the Alpha Vantage API [13]. These APIs allow us to pull data at different time intervals over different lengths of time depending on the interval. We were able to pull five minute interval data over a week's time and one hour interval data over a month's time for forty-five different stocks from the following sectors: technology, healthcare, finance, automotive, agriculture, defense, energy, and general exchange-traded funds. The data is organized by time and includes opening, closing, high, and low price of the stock in the chosen interval.

1.3. Related Work

Researchers from many different fields have attempted to solve the stock price prediction problem. Popular methods include signal processing, machine learning, and technical analysis. Ensemble methods that combine multiple tools together, such as a Kalman filter with a Long Short-Term Memeory (LSTM) model, have been shown to perform better than using the stand-alone counter parts [16]. Similarly, the combination of a support vector machine with information of investor sentiment found using a web scraper has shown to produce an accuracy of 89.93% when predicting a binary up/down label [20]. One can also use domain-specific information by employing the use of technical indicators and using them as feature vectors [1]. These are calculated features (moving average, rate of change, and others) of stock data used in the field of technical analysis. When using technical indicators, the best performance is seen when using a minimal set of indicators.

2. Methods

We begin our investigation by pre-processing the data and calculating *technical indicators* to avoid dealing with the raw time-series of stock prices. The technical indicators are used as the features for our networks' input vectors. The models will then output the prediction for the percent change in price that would be seen in the next time-step. We train a non-convex two-layer ReLU neural network and the equivalent convex network proposed in [19]. We then evaluate the final trained models on a set of evaluation metrics (MSE, MAE, sign accuracy) to analyze the performance of each model. Finally, we will simulate a trading scenario using a trading bot that will buy or sell informed by the predictions of our final models.

2.1. Technical Indicators

Technical indicators are used as input to the network and are calculated using the Python Technical Analysis Library [18] and the Pandas Technical Analysis Library [14]. These indicators include information on the volume, volatility, trend, and momentum of the stock, and each of them implicitly encode temporal data since they are calculated using a past subset of data. The Python version of the library can calculate 82 different technical indicators using historical stock data, but reducing this number down to less than 30 has been shown to be vital for successful predictions [3, 10]. The exact method for selecting the indicators is still an open question, but [1] gives some insights on feature selection techniques and reports the best fifteen indicators for prediction results. A subset of these will be used in this application, and we use the Pandas version of the library to hand-select them because some of the fifteen reported are missing in the Python version.

2.2. Standard 2-Layer ReLU Network

As a benchmark for the convex reformulation, a standard 2-layer ReLU network is developed and tested. The two-layer neural network is an extension of Rosenblatt's perceptron [21] that uses a cascade of perceptrons and adjusts the inner-weights using the method of back propagation. This multi-layer perceptron (MLP) was first theorized by Minsky and Papert in 1969 [15] and later formalized in 1986 by Ramelhart, Hinton, and Williams [22] to fix the issues of the single perceptron being unable to solve simple logic problems. Borrowing notation from the Pilanci paper on the convex reformulation [19] to maintain consistency with discussions in the next section, start with a two-layer network that takes in a vector input and outputs a scalar:

$$f(\boldsymbol{x}) = \sum_{j=1}^{m} \phi(\boldsymbol{x}^T \boldsymbol{u}_j)\alpha_j \tag{1}$$

where we have the input data vector, $\boldsymbol{x} \in \mathbb{R}^d$, the hidden-layer weights, $\boldsymbol{u}_j \in \mathbb{R}^d$, the output-layer weights, $\alpha_j \in \mathbb{R}$, and the ReLU activation function, $\phi(t) = \max(t,0)$.

For this paper's two-layer network, the input is the d-length vector of technical indicators, and the output is the scalar prediction for the percentage of price change between the current time and the next time step. Because of this, the squared loss objective function is used with a regularization term of the l_2-norm of the parameters. Again, notation is

borrowed from [19] for the definition of the minimization function:

$$\min_{\boldsymbol{\alpha},\mathbf{u}} \frac{1}{2}\left\|\sum_{j=1}^{m}\phi(X\boldsymbol{u}_j)\alpha_j - \boldsymbol{y}\right\|_2^2 + \frac{\beta}{2}\sum_{j=1}^{m}(||\boldsymbol{u}_j||_2^2+\alpha_j^2) \quad (2)$$

where $X \in \mathbb{R}^{n\times d}$ is the data matrix, $\boldsymbol{y} \in \mathbb{R}^n$ is the label vector, and $\beta > 0$ is a regularization parameter. The hidden weights, $\boldsymbol{u}$, and the output weights, $\boldsymbol{\alpha}$, are the minimization variables that define the model and are solved for using PyTorch tools.

2.3. Convex Formulation

With convexity definitions comes convex optimization, a field where many efficient algorithms exist to solve for the minimum of the program. A common issue with multilayer neural networks is the lack of convexity, and initial attempts to reformulate them as a convex problem required a possibly infinite amount of neurons [4]. The existence of a convex solution for a 2-layer ReLU network with a finite number of neurons was then discovered in [2] and finally formulated by Pilanci and Ergen [19].

2.3.1 Formulation for vector-input to scalar-output

Pilanci and Ergen proved that (2) can be reformulated as the following convex program:

$$\begin{aligned}\min_{\mathbf{v},\mathbf{w}} \frac{1}{2}\left\|\sum_{i=1}^{P} D_i X(\boldsymbol{v}_i-\boldsymbol{w}_i) - \boldsymbol{y}\right\|_2^2 \\ +\beta\sum_{i=1}^{P}(||\boldsymbol{v}_i||_2+||\boldsymbol{w}_i||_2) \\ s.t.\ (2D_i-I_n)X\boldsymbol{v}_i \geq 0, (2D_i-I_n)X\boldsymbol{w}_i \geq 0, \forall i\end{aligned} \quad (3)$$

where D_i are diagonal matrices with diagonal elements equal to $\boldsymbol{x}_k^T\boldsymbol{u} \geq 0\ \forall k = \{1,2,...,n\}$ and with $\boldsymbol{u}$ being an arbitrary vector in $\mathbb{R}^d$. The diagonal elements of the D matrices define hyperplanes that pass through the origin and partition the space $\mathbb{R}^d$. Using P of these D matrices allows the program to train many locally linear models separately. The optimial solutions, $\{\boldsymbol{v}_i^*, \boldsymbol{w}_i^*\}_{i=1}^P$ can then calculate the optimal solutions in the non-convex program (2) using the following relationship:

$$(\boldsymbol{u}_{j_{1i}}^*, \alpha_{j_{1i}}^*) = \left(\frac{\boldsymbol{v}_i^*}{\sqrt{||\boldsymbol{v}_i^*||_2}}, \sqrt{||\boldsymbol{v}_i^*||_2}\right) \text{ if } \boldsymbol{v}_i^* \neq 0$$

$$(\boldsymbol{u}_{j_{2i}}^*, \alpha_{j_{2i}}^*) = \left(\frac{\boldsymbol{w}_i^*}{\sqrt{||\boldsymbol{w}_i^*||_2}}, -\sqrt{||\boldsymbol{w}_i^*||_2}\right) \text{ if } \boldsymbol{w}_i^* \neq 0$$

Here, Pilanci notes that $\boldsymbol{u}_{j_{ki}}$ and $\alpha_{j_{ki}}$ follow from the non-unique convex optimal solutions $\boldsymbol{v}_i^*$ and $\boldsymbol{w}_i^*$ that can be expressed as a combination of two positively colinear solutions as $\boldsymbol{v}_i^* = (c_1+c_2)\boldsymbol{v}_i$. Though the optimal solution is non-unique, the result is proven to yield the same objective.

2.3.2 Limitations of Convex problem

One drawback with this model is the theoretically required number of D matrices, P, on any real data. Given a data set of size n with elements of dimension d that produce a data matrix of rank r, P is upper-bounded by the inequality [17, 5]

$$P_{n,r} \leq \sum_{k=0}^{r-1}\binom{n-1}{k} \quad (4)$$

The inequality becomes an equality for this convex program due to the use of hyper planes in the general position [19]. For machine learning problems, n is usually very large, however, the main issue is the exponential increase as the dimension, r, increases. Considering a data set of n=1000 and r=2, so P=2000. Increasing r to 3 yields a required P of 999,002. To address this issue, Pilanci notes that one can approximate the objective function (3) by sampling a set of D matrices by generating $u \sim (0, I_d)$ $\tilde{P}$ times. In the related python implementation [23], the P used has no relation to the sizes of the data matrices and is instead hard-coded to a value. Without properly setting P, it is not possible to create a proper mapping from the convex formulation back to the two-layer network since the number of non-zero solutions to (3) is upper bounded by the number of neurons in the two-layer net, as noted by Pilanci's result at the end of 2.3.1.

Another drawback to the convex reformulation is the overall computational complexity. Pilanci notes that (3) is solved in $O(d^3r^3(\frac{n}{r})^{3r})$ time, which quickly becomes computationally infeasible when $n > r$ and the complexity explodes to polynomial time in r. Considering this, one can consider the convex solution to be computationally constrained to instances where either $n \leq r$ or r in of a magnitude the allows for the problem to remain computationally feasible. In a practical sense, the convex solution can only be applied to problems with a limited number of training examples to meet $n \leq r$, or limited in feature dimensions r as the solution is polynomial time in r. In order to solve the convex program in a reasonable time for our application, we will later present using methods such that $5 \leq r \leq 30$, which also aligns with [1] in selecting the number of technical indicators for learning representations in stock data .

2.4. Proposed Convex Model

To combat the complexity constraints discussed in the section above, we propose solving the convex minimization

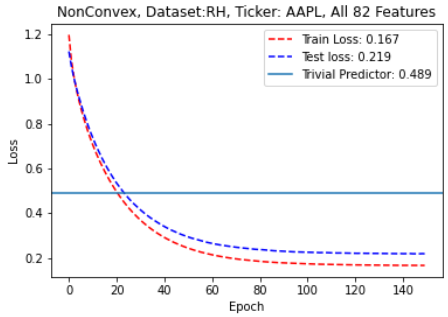

Figure 1. Training and test loss for the **non-convex** model on AAPL stock data using TI-82 feature representation

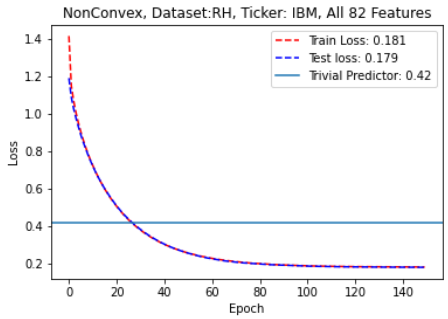

Figure 2. Training and test loss for the **non-convex** model on IBM stock data using TI-82 feature representation

program (3) by stochastic gradient decent. This, however, requires extra hyper-parameters that we do not have an optimal solution to and can only find experimentally. This approach was taken by Ergen to classify images in the CIFAR-10 data set here [23]. We also consider limiting the size of P shown in (4) to limit the complexity. Rather than using all 82 features provided by the Technical Analysis Library, a subset found by Alsubaie et al.[1] using feature selection is first taken. As an alternative approach, we perform a singular-value decomposition on the indicators, informed by Ergen et al. in [6]. In our experiments, we will compare the performance of using all 82 technical indicators (TI-82), the top technical indicators (TI-15), and sub-selecting features using the SVD (SVD-10).

2.5. Evaluation Methods

2.5.1 Metrics

Performance of the 2-layer network and the convex network are evaluated using the metrics: mean squared error (MSE), mean absolute error (MAE) and the sign accuracy (SACC)—or the percentage of predictions whose sign matched the true value. As stated before, the labels are the percent change of the stock price between the current time

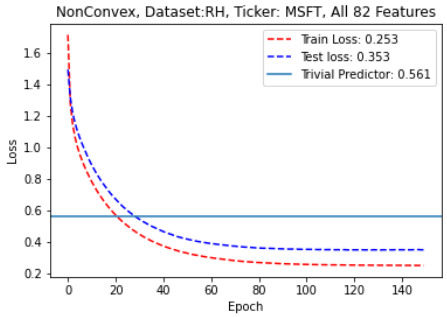

Figure 3. Training and test loss for the **non-convex** model on MSFT stock data using TI-82 feature representation

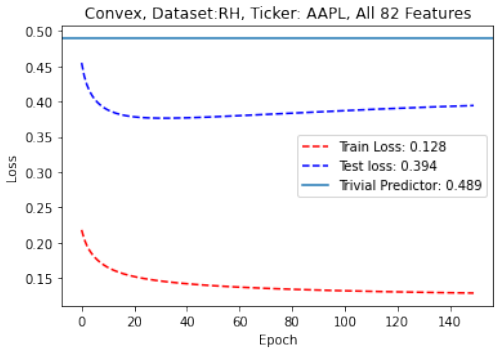

Figure 4. Training and test loss for the **convex** model on AAPL stock data using TI-82 feature representation

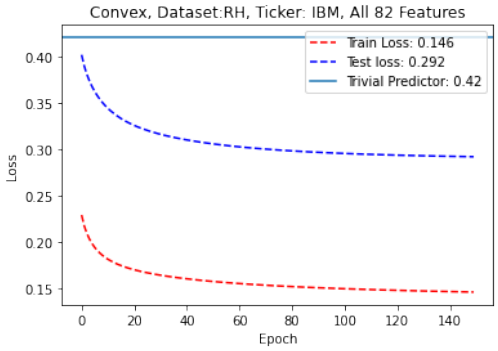

Figure 5. Training and test loss for the **convex** model on IBM stock data using TI-82 feature representation

and the next time stamp. For the purposes of comparison, we also include a *trival predictor* which will always output the mean of the training labels as its prediction.

2.5.2 Feature Representation

In our experiments, we consider three different methods of representing the input data. In the naïve method (TI-82), all

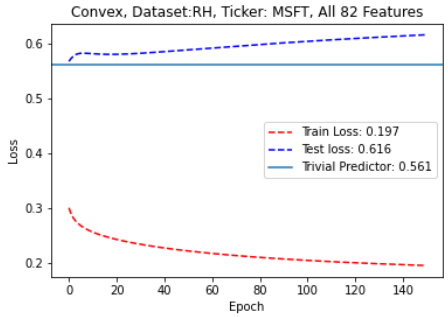

Figure 6. Training and test loss for the **convex** model on MSFT stock data using TI-82 feature representation

82 technical indicators (TIs) provided from the TA library [18] are used as features fed to the networks. Our second method (TI-15) uses the top 15 TIs selected based on the results found by Alsubai et al.[1] who developed a feature selection method for TIs. We compare these methods with Singular Value Decomposition [9] of the 82 TIs into their top 10 singular directions (SVD-10) informed by [6]. This is done by decomposing the input matrix, X, into left, right and singular matrices U, V^T and Σ and extracting the first 10 columns of U which correspond to the largest 10 singular values of Σ, resulting in the closest rank-10 matrix to the original data.

2.5.3 Deployment Feasibility

To provide a realistic application of the models, a trading bot will use the predictions of the models to drive decisions on whether or not to buy or sell some amount of stocks. The two strategies that will be employed are to "buy and hold" and to buy or sell based on the magnitude of predicted price change. The trading bot will be given an initial amount of money, and the final return on investment will be reported as a percentage of money gained or lost. The following pseudo-code in (5) shows the strategy to buy/sell based on magnitude of predicted percent change.

$$
\begin{aligned}
&\text{for } \hat{y} = \text{model}(x), \\
&\quad \text{if } \hat{y} > 0 \text{ and balance} > 0 : \\
&\quad\quad \text{buy } \hat{y} * \max(\text{purchase}; \text{balance}) \\
&\quad \text{else if } \hat{y} < 0 \text{ and stock balance} > 0 : \\
&\quad\quad \text{sell } \hat{-y} * max(\text{liquidate}; \text{stock balance})
\end{aligned} \tag{5}
$$

	Optimal Hyper-parameters		
Model	λ	β	h / r
Non-Convex (TI-82)	$1e-3$	$1e-1$	10
Convex (TI-82)	$1e-6$	$1e-2$	80
Non-Convex (SVD-10)	$1e-3$	$1e-1$	10
Convex (SVD-10)	$1e-3$	$1e-2$	80
Non-Convex (TI-15)	$1e-3$	$1e-1$	10
Convex (TI-15)	$5e-5$	$1e-2$	80

Table 1. Hyper-parameters determined via grid search

3. Experiments

3.1. Hyper-Parameter Turning

Each network's hyper-parameters were selected via a grid search which was performed for each of the three feature representations. For the non-convex network, the hyper-parameters in need of tuning consist of the learning rate, λ, the regularization strength, β, and the hidden layer size, h. The convex network also requires λ and β but includes a parameter for the number of regions, r, which defines how many samples are taken during the creation of the D matrices discussed in 2.3.2. In order to relate the amount of hidden neurons of the non-convex network with the amount of regions in the convex network, the parameter $\frac{h}{r}$ is considered as one. The optimal parameters for each feature representation and network is shown in Table 1.

3.2. Model Training

3.2.1 Non-Convex

In Table 1, we see that the non-convex network was optimized with the same set of hyper-parameters for each feature representation. This can be attributed in part to our finding that the non-convex network loss converged to the same or near-same minimum for most hyper-parameters combinations when the regularization strength β was held constant. From this we can deduce that the non-convex network's convergence is highly determined by the regularization strength as this regulates models complexity. Three example training and test loss per epoch plots can be seen in Figure 1, 2, and 3. A key takeaway should be that non-convex network loss converges for each individual stock using the same hyper-parameters where such behavior is not seen in the convex network.

3.2.2 Convex

In Table 1, we see results from the convex model differing from that of the non-convex network as optimal hyper-parameters are not consistent across all feature representations. When looking at loss convergence in Figure 4 and

Model	MSFT			AAPL			IBM		
	MSE	MAE	SACC	MSE	MAE	SACC	MSE	MAE	SACC
Non-Convex (TI-82)	0.084552	0.515968	**0.5625**	**0.06562**	**0.358062**	**0.725**	**0.057282**	**0.308709**	**0.7625**
Convex (TI-82)	0.749401	0.508237	0.25125	0.697639	0.481239	0.264531	0.648129	0.410982	0.253906
Non-Convex (SVD-10)	**0.083786**	0.508241	0.475	0.077997	0.48123	0.5625	0.072461	0.41096	0.55
Convex (SVD-10)	0.749623	0.50852	0.243125	0.697597	0.48127	0.275938	0.648108	0.410958	0.254844
Non-Convex (TI-15)	0.085079	0.546904	0.5125	0.070287	0.413291	0.7125	0.067008	0.392757	0.5875
Convex (TI-15)	0.749501	0.50819	0.249844	0.695302	0.480784	0.263125	0.648244	0.411018	0.252188
Trivial (Mean)	0.101757	**0.505574**	0.496582	0.086587	0.478981	0.486353	0.107993	0.412234	0.503374

Table 2. Evaluation of convex and non-convex

Figure 5, we also see that, not only was convergence dependent upon stock selection, but there were instances where the loss would not converge regardless of hyper-parameter selection, as seen in Figure 6. This behavior is not exhibited in any of the non-convex network experiments. One may suspect this behavior being attributed to over-fitting and could be resolved with proper hyper-parameter selection; however, a second exhaustive hyper-parameter search provided no working set of parameters capable of producing strong performing results. It should be stated the each of these discussed was not exclusive to TI-82 and were present with the convex network for each feature selection method.

A claim could be made that, due to the insufficient data set size and the inherent nature of stock prices often reflecting a distribution characterized by a Gaussian random walk, no meaningful pattern can be extracted from data that would generalize from the training set to the test set. However, the results of the non-convex network in Figure 3 would disagree with that claim as the non-convex network was able to do just that.

3.3. Results

3.3.1 Testing Dataset

Table 2 provides the evaluation metrics for final model performance on the testing stock data. We provide the results for Microsoft (MSFT), Apple (AAPL), and IBM stocks. We also include the performance of a trival predictor which always guesses the mean percentage change of the training stock data. The best result for each metric is in bold font. Our results found the opposite result that was found by Alsubaie [1] where it was reported that a lower amount of indicators yield more accurate results. As seen in Table 2, we see that the best performance is achieved when using the most amount of technical indicators.

Overall, we have seen the unfortunate result that the non-convex network using all technical indicators generally gave the best performance. On AAPL and IBM stock, the non-convex networks by far gave the best sign accuracy, meaning they were more accurately able to guess whether the stock was going up or down. The convex networks typically have higher MSE, even more than the trivial mean predictor. A possible explanation for this is the requirement for $\tilde{P}$ on our dataset. As discussed earlier, the true value for P can be extremely large, so it is often more feasible to sample a set of D_i matrices. However, it is possible that we still needed to sample many more D_i matrices. As result, our sampled approximation of the convex problem could have been very poor, which could make the global optimum far off from that of (2) or (3).

3.3.2 Trading Bot Performance

After being trained with the convex and non-convex programs, the trading bot employed the aforementioned strategy (5) for 90 1-hour intervals. The results displayed for the three stocks (Table 3) demonstrate that the convex-trained model outperforms the non-convex in all scenarios and all neural nets beat the naive buy-and-hold strategy. This is interesting given that the non-convex achieved much lower training and test errors in longer training runs, however, in the trading bot training, the convex program reaches lower training loss in the given time. The result is consistent with that found in [1] where the models that achieved better testing loss actually performed worse in the stock market. It is worth noting that in 2/3 stocks, the SVD-reduced input data predicts percent changes that yield better trading performance, suggesting that many of the technical indicators are likely redundant and antagonistic. Both full and SVD-reduced data significantly outperform the handpicked features. In future work, it would be beneficial to explore more advanced trading strategies which can leverage falls in the market. In summary, the trading bot demonstrates how best-loss models are not necessarily the top performing in practice and that convex optimization of neural nets may be a better approach.

Model	Return of Investment % MSFT	AAPL	IBM
Non-Convex (TI-82)	-3.83	0.17	2.2
Convex (TI-82)	-3.73	-0.54	**2.62**
Non-Convex (SVD-10)	-3.56	0.55	0.18
Convex (SVD-10)	**-0.44**	**2.42**	-2.03
Buy-and-Hold	-6.10	-5.49	-7.44

Table 3. Trading Bot Performance (90 hours)

4. Future Improvements

4.1. Data

While daily-price stock data is available for the past few decades, it only provides the opening and closing prices for each day. Our approach in this paper was to use more finely-grained time intervals. These are provided by the two utilized APIs, however, they limit the amount of historical data they provide depending on the time interval the user chooses. Because of this, we were limited to only the past two months of data when choosing to have one hour time intervals. Since stocks are famously difficult to predict, it would be preferred to have more finely-grained data over a longer period of time. This would require making weekly requests from the APIs but would allow for a specific stock to have enough data to allow for a generalized solution.

4.2. Feature Selection

Rather than using a naïve SVD decomposition to select the best technical indicators or those reported by other papers, it may be beneficial to perform a technical indicator search that maximizes prediction error within a given window for a given stock. Since the "best" indicators may change depending on which stock one is trying to predict, a new search may be necessary for each individual stock or sector.

5. Conclusion

The main advantage of the convex formulation is that we are guaranteed to find the global optimum. Doing so is invaluable when dealing with noisy data distributions, such as stock data, since standard gradient descent is susceptible to local minima. However, the two-layer ReLU may be too limiting as an estimator when it comes to stock data due to the amount of external socio-economic and political influences on the prices. Here, we demonstrated that the non-convex program is still able to outperform the convex program in training and test losses, yet consistent with previous results this does not generalize well when employed in the market. Thus, the convex training program ultimately yielded better profits even with simple strategies in a tough market.

References

[1] Yazeed Alsubaie, Khalil El Hindi, and Hussain Alsalman. Cost-sensitive prediction of stock price direction: Selection of technical indicators. *IEEE Access*, 7:146876–146892, 2019.

[2] Raman Arora, Amitabh Basu, Poorya Mianjy, and Anirbit Mukherjee. Understanding Deep Neural Networks with Rectified Linear Units. *arXiv e-prints*, page arXiv:1611.01491, Nov. 2016.

[3] George S Atsalakis and Kimon P Valavanis. Surveying stock market forecasting techniques – part ii: Soft computing methods. *Expert systems with applications*, 36(3):5932–5941, 2009.

[4] Yoshua Bengio, Nicolas Roux, Pascal Vincent, Olivier Delalleau, and Patrice Marcotte. Convex neural networks. In Y. Weiss, B. Schölkopf, and J. Platt, editors, *Advances in Neural Information Processing Systems*, volume 18. MIT Press, 2005.

[5] Thomas M. Cover. Geometrical and statistical properties of systems of linear inequalities with applications in pattern recognition. *IEEE Transactions on Electronic Computers*, EC-14(3):326–334, 1965.

[6] Tolga Ergen, Arda Sahiner, Batu Ozturkler, John Pauly, Morteza Mardani, and Mert Pilanci. Demystifying batch normalization in relu networks: Equivalent convex optimization models and implicit regularization. 03 2021.

[7] Eugene F. Fama. Efficient capital markets: A review of theory and empirical work. *The Journal of Finance*, 25(2):383–417, 1970.

[8] Joshua Fernandez. robin_stocks. `https://github.com/jmfernandes/robin_stocks.git`, 2020.

[9] Gene H. Golub and William Kahan. Calculating the singular values and pseudo-inverse of a matrix. *Journal of the Society for Industrial and Applied Mathematics, Series B: Numerical Analysis*, 2(2), 1965.

[10] In Goo Han, Kyoung Jae Kim, John Ch, and ler. Extracting trading rules from the multiple classifiers and technical indicators in stock market. *KMIS International Conference*, pages 147–163, 1998.

[11] Adam Hayes. Value investing. *Investopedia*, 2021.

[12] Kurt Hornik, Maxwell B. Stinchcombe, and Halbert L. White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2:359–366, 1989.

[13] Alpha Vantage Inc. Alpha vantage api documentation. 2017-2022.

[14] Kevin Johnson. pandas-ta. `https://github.com/twopirllc/pandas-ta.git`, 2019.

[15] Marvin Minsky and Seymour Papert. *Perceptrons: An Introduction to Computational Geometry*. MIT Press, Cambridge, MA, USA, 1969.

[16] Kyle J. Morris, Sean D. Egan, Jorell L. Linsangan, Carson K. Leung, Alfredo Cuzzocrea, and Calvin S. H. Hoi. Token-based adaptive time-series prediction by ensembling linear and non-linear estimators: A machine learning approach for predictive analytics on big stock data. In *2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA)*, pages 1486–1491, 2018.

[17] P.C. Ojha. Enumeration of linear threshold functions from the lattice of hyperplane intersections. *IEEE Transactions on Neural Networks*, 11(4):839–850, 2000.

[18] Dario Lopez Padial. Technical analysis library in python. *Read the Docs*, 2018.

[19] Mert Pilanci and Tolga Ergen. Neural networks are convex regularizers: Exact polynomial-time convex optimization formulations for two-layer networks. In *International Conference on Machine Learning*, pages 7695–7705. PMLR, 2020.

[20] Rui Ren, Desheng Dash Wu, and Tianxiang Liu. Forecasting stock market movement direction using sentiment analysis and support vector machine. *IEEE Systems Journal*, 13(1):760–770, 2019.

[21] Frank Rosenblatt. Principles of neurodynamics: Perceptrons and the theory of brain mechanisms. *Washington: Spartan Books*, 1962.

[22] David E. Rumelhart and James L. McClelland. *Learning Internal Representations by Error Propagation*, volume 1, pages 318–362. M.I.T. Press, 1986.

[23] Ergen Tolga. convex_nn. `https://github.com/pilancilab/convex_nn.git`, 2021.

[24] Joseph De La Vega. Confusion de confusiones. 1688.