Predicting Stock Prices using Recurrent Neural Networks

Gary Qian   Benjamin Hoertnagl-Pereira   David Li

Abstract

Correctly predicting stock price movements is an incredibly lucrative problem. Traditional approaches for prediction involve extensive market research and models with hand crafted signals. In this project, we investigate the suitability of recurrent neural networks for analyzing and predicting stock prices purely as raw time series data. RNNs are specialized for processing sequential data, and application to stock prices is a natural fit. For this project, we test various windowing techniques, augmentations, and "cascade" style transfer learning to attempt to develop insight into worthwhile techniques for stock prediction.

1 Introduction

Stock markets are hard to predict. There are many factors that drive supply and demand, from market-wide effects from government changes in inflation policy to localized effects from events such as new product releases. Traditional traders consider price changes as indicator signals for future price movement; however, their actions can in turn which can then trigger price changes, leading to complex feedback systems.

Recently, there has been significant work in applying machine learning to time series data in the form of recurrent neural networks. Current research includes phoneme recognition in speech audio [3], music composition [6], and stock trend prediction [5].

Neural networks can be seen as a nonlinear function approximator. In this particular setting, we can give the price of the stock as a function of time. In particular, if v_i is the price of a stock at time i, using the n most recent share values, we are trying to compute $f(v_t, v_{t-1}, \ldots, v_{t-n+1}) = v_{t+1}$. To examine ways to solve the stock price prediction problem, we make some simplifying assumptions and run experiments to empirically validate their performance.

We will discuss our data collection and processing in section 2, our methodology and results in section 3, and conclusions in section 4.

2 Data Collection

We used the **NASDAQ100 dataset**, which provides stock prices on 1 minute intervals for 104 stocks on NASDAQ. The data covers 191 days from July 26, 2016 to April 28, 2017. We also collected intra-day data from the Bloomberg Terminal, in order to verify and augment our NASDAQ dataset. We used a random 70-20-10 split on the data to create our train, validation, and test datasets, respectively.

2.1 Dataset Creation

In particular, the NASDAQ100 set had roughly 5-10% of the data missing in certain timepoints. These data points were filled in by pulling the data from Bloomberg. We realized that the NASDAQ100 dataset format and size was sufficient for the experiments we ran. We used a 70-20-10 split for our train, validation, and test datasets.

One interesting consideration that needed to be made was the ordering in which were to pass our windowed data to our model. For most problems, it is common to assume that data i.i.d. from some underlying distribution, so it is common to shuffle samples from the dataset. However, this simplifying assumption is to strong to be made for sequential stock data, especially since there is such high correlation between prices at different timesteps. Thus, we decide to keep the windows in continuous ordering.

2.2 Preprocessing

In order to ensure that predictions are not biased towards specific stocks, it is required to normalize data. Normalization could be considered for the entire price history for a given stock, or instead for the specific windows of the prices. Given our focus on generalization to arbitrary stock data, we normalize each window by subtracting the window mean and dividing by variance such that $(\mu = 0, \sigma = 1)$. This transformation eliminates the impact of the actual value of the stock and has variance stabilizing effects.

We used 12 of the 104 stocks for our actual training. Due to hardware limitations, 12 stocks (with augmentations) was the most data that would fit in the GPU memory (3GB GTX 1060) while still allowing a large enough LSTM model in memory. We attempted using Google Colab, but due to shared GPU instances, we could not reliably get access to the full GPU RAM space. In efforts to keep all of our run consistent to compare, we decided to use the abridged dataset.

2.3 Windowing Techniques

A few of the key parameters we explored for windowing were the windows size as well as augmentation through overlapping adjacent windows. To focus in on interesting window scales and overlap amounts, we did preliminary experimentation.

We found that small windows of around 10 time points were too small to produce any comprehensible result beyond noise, and very large batches were not well generalized. After the early runs, we settled on trying everything with windows of 64 and 128. These windows sizes produced.

With regard to overlap, sliding the window by one minute every window hugely increased the data size, but did not perform any better than much larger steps (smaller overlap). We did notice that the augmentation process resulted in different behavior in both validation and training loss, but the difference was not enough to warrant a full sweep. Therefore, we decided to explore the effects of half-window-size (50%) overlap versus no overlap.

The final windowing approach was a full exploration over the following parameters: [64, 128] window size, [0%, 50%] overlap, and [1, 2, 3, 4, 8, 16, 32] future prediction time steps.

3 Approach and Results

The bulk of the experiments were run with a relatively simple two layer LSTM, each layer with 64 hidden units, and two fully connected linear layers (64×64 and 64×1) to transform the hidden states from the LSTM to a real valued stock price.

This model performed relatively well with low chance of overfitting on most of the datasets we used. Deeper models tended to greatly overfit on training without improving (and often worsening) the validation loss. We used 64 hidden nodes, which was able to achieve reasonable performance on the data. Additional hidden layers did not improve the validation loss, led to behavior that we suspected was rote memorization, and greatly increased the run times of our model.

We chose the ADAM optimizer because it was one of the only optimizers that were able to produce consistent diminishing loss on both the training and validation data. SGD resulted in very smooth descent of loss but reached an asymptote that was very high. Initially, ADAM resulted in large spikes in our loss, but this was mitigated by increasing the epsilon value to 10^{-6} instead of the default 10^{-8}.

To test our model, we performed a random 70-30 split on the stock data. 70% was used as training data, and 30% was used as validation.

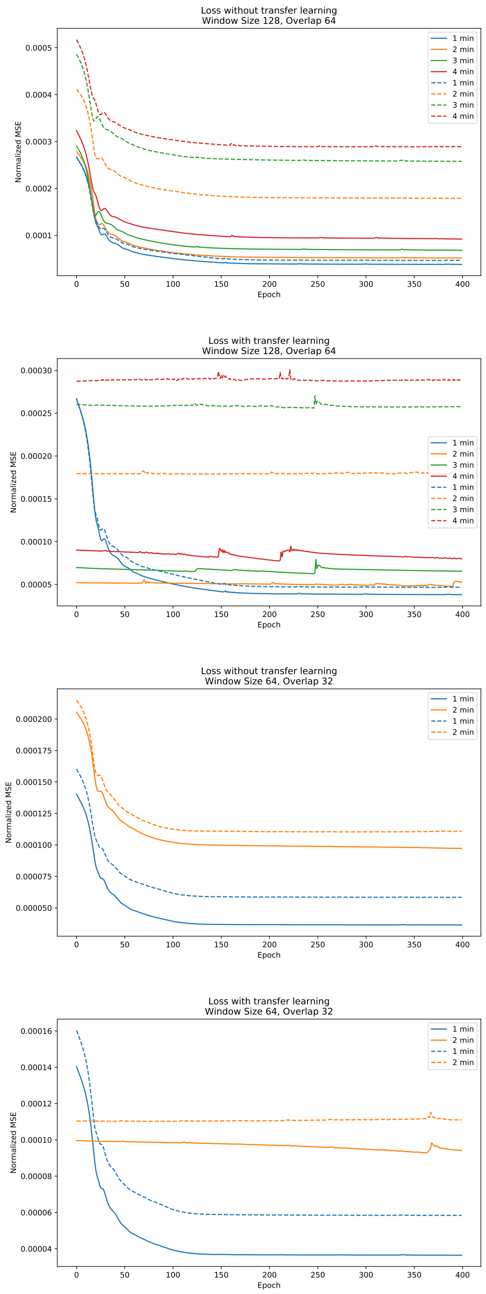
For Many-To-One prediction, each of the parameter combinations were run with both a randomly initialized neural network as well as a transferred "cascading" neural network, resulting in a total of 56 neural networks for 400 epochs each. We looked into early stopping, but it did not seem to matter as running the network for longer did not significantly increase or decrease any of the metrics once it reached the asymptotic values.

3.1 Many-To-One

First, we wanted to explore the results of predicting single future values. We configured the output of the linear layers to output to a dimension of 1. Then, we selected the ground truth y vector to be 1, 2, 3, 4, 8, and 32 time steps (minutes) into the future (Note: the loss for 32, 16, and 8 are not plotted on the same plots due to the drastically different scale of the loss. We have included these values separately). We expected that the LSTM would perform very well on the shorter future steps, and worse on the larger steps. We also repeated the experiments with windows of 64 and 128, each with both no overlap augmentation and with 50% overlap augmentation (Figure 1).

These experimental future step parameters were arrived at through a preliminary exploration of interesting timescales to test. We found that at time steps past 32 minutes into the future, the LSTMs were unable to predict with any reasonable accuracy.

We found that the LSTM was able to arrive at very small loss at only 1 minute in the future. This loss increased at steady intervals as the prediction time steps increased.


Loss without transfer learning
Window Size 128, Overlap 64

Loss with transfer learning
Window Size 128, Overlap 64

Loss without transfer learning
Window Size 64, Overlap 32

Loss with transfer learning
Window Size 64, Overlap 32

Figure 1: Solid lines are training loss and dotted lines are validation loss. Performance of 128min windows with 64min overlap and both transfer and no transfer, as well as 64min windows with 32min overlap with both transfer and no transfer. +8, +16, +32 predictions omitted

3.1.1 Many-To-One Cascade (Transfer)

To improve performance of predicting more distant future values, we attempted to use transfer learning. The weights from the previous smaller future time step were used to initialize the model for the current larger future time step. For example, we initialized the model for 4 minute prediction with the weights from 3 minute prediction. This was continued for each of the time steps we explored, producing a “cascading” model that is continuously used to initialize the next one.

The theory is based on the idea that by initializing the model with the weights of previous smaller predictions, the model may have both a better start than random initialization as well as potentially avoid local minima that previous models have encountered. In addition, even if it does not improve absolute loss, it is possible that we may be able to reach a similar loss faster than with a randomly initialized model.

After running both randomly initialized models and transferred models, we found that the transferred models were indeed able to attain similar training and validation loss to the random models faster, and the transferred models actually attained lower loss than the random models.

We also see the introduction of spikes and regressions in the performance in the transferred models. These spikes are likely less extreme versions of the spikes we encountered earlier in Adam optimizers. Overall, the models were able to recover rapidly from these disruptions.

The transferred and random models both converged to remarkably close validation and training losses and we believe that these losses are the bound by the architecture we used. However, it was interesting to note that the transfer model was almost immediately able to jump to the lowest validation losses at epoch 1. This is useful because transfer learning can be applied to increasing timescales and reduce training from hundreds of epochs to potentially a few dozen. This can lead to massive computational savings when running the networks on larger datasets.

3.1.2 Overlap Effects

Overlapped data saw similar loss when compared to non-overlap data on the small future time steps of 1 and 2 minutes. As the time steps increased to 3, 4, 8, 16, and 32, the overlap validation loss was significantly lower than non-overlapped versions (Figure 2). Since predicting further in the future is a harder task,

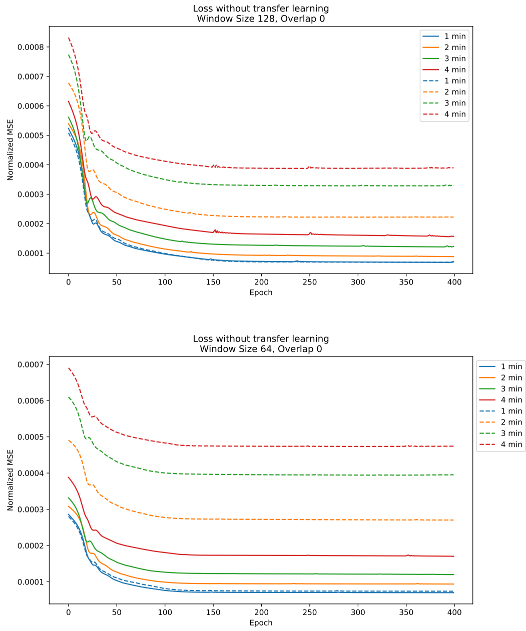


Figure 2: Effect of no overlap between windows for 128min and 64min windows. Solid lines are training loss and dotted lines are validation loss. +8, +16, +32 predictions omitted

we believe that the augmentation of the data only really had the impact visible on the harder tasks. The overlapping adds another recurrent component to the data outside of the LSTMs of the network itself.

Overall, we found that the overlapping augmentation improves performance at the cost of increased training times as it increases the dataset size significantly. However, the effects were only apparent at larger future time steps.

3.1.3 Window Size

The effects of 64 vs 128 window size was fairly consistent across the board. We saw consistently lower validation and training losses with and without overlapping. The increased window size likely performed better due to a larger input "feature" size and each iteration was able to learn from more data points. Larger window sizes had similar performances to 128, which is why we decided to focus on 128 as our primary exploration window size for hardware optimization reasons.

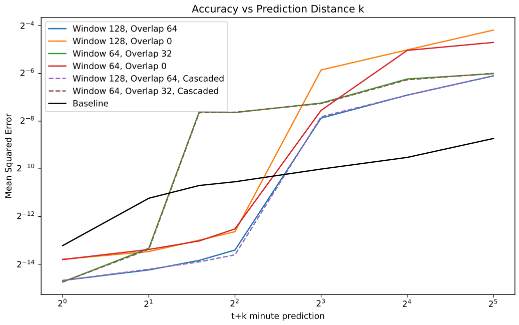


Figure 3: Prediction accuracy of various models for $f(v_t, v_{t-1}, \ldots, v_{t-n+1}) = v_{t+k}$ versus k. The black line represents the baseline for comparison and for small k, our models beat the baseline.

3.1.4 Performance vs Baseline

To compare with a more real world scenario, we calculated a baseline performance by taking the final value of the input window and using it as the prediction for the future.

We found that our LSTM models actually performed better than this simple guessing scheme for the smaller timescales. For predictions 1-4 minutes into the future, our loss from the truth is better than our baseline. After 4 minutes, we see large increases in the loss, and the validation loss is no longer beating the baseline loss. At longer future predictions, it seems that the LSTMs are unable to consistently produce reasonable predictions, which can be attributed to the highly random and volatile nature of stock data (Figure 3).

3.2 Many-To-Many

In addition to considering single price predictions at various future timesteps, we consider the sequence to sequence generation problem of predicting prices in the time range $[1, T]$ from prices in the time range $[0, T-1]$, where T is the length of the window. In this way, we are constantly predicting the next price throughout the entirety of the sequence instead of just a the very end, ensuring that hidden states are robust enough to model intermediate prices to not simply memorize a final output. Additionally, we reasoned that these intermediate predictions allow for more informative and stronger gradients for improved learning.

We began by training our baseline LSTM, and adjusted our target the sequence of prices offset by one timestep, instead of just the final price. This model had smooth losses, showing convergence relatively early within about 10 epochs, and we found that validation loss tracked the training exactly. These both suggest that the model was underfitting, so we decided to increase parameters by adding a 3rd layer to the LSTM and increasing the hidden units to 400. This deeper model had more noisy training and validation loss, with steep spikes characteristic of the Adam optimizer, and took about 300 epochs to converge. We found that validation loss tracked train almost exactly, with the lower bound on the loss essentially the same as the baseline LSTM. Again, this would suggest overfitting; however, we were unable to experiment with deeper models due to GPU limits.

It is interesting to note that the larger model did not reach as low a loss as the smaller model - more training could have been done but it seems to reach an asymptote. Additionally, the fact that train and validation loss is almost identical is concerning (Figure 4). Perhaps the inherent randomness in these stocks is too difficult to predict entire sequences reliably.

3.2.1 Stock Generation

With few epochs of training, we found that generated prices were completely sinusoidal and in phase with one another. This was interesting as the neural net was able to "derive" the properties of a Fourier transform. As training increased, the generated prices took on the form of more complicated compositions of sinusoidal functions. This can be interpreted as the hidden states learning a simple Fourier representation of the sequential data (Figure 5).

4 Conclusions

We were able to discover promising results that have potential applications in high speed (minute scale) trading. We found that in general, overlap augmentation of time series data is beneficial for producing smaller loss. 128 window size performed better than 64, but not significantly different from larger window sizes. Cascading the neural network through transferring weights allowed rapid arrival at optimal losses and resulted in slightly smaller absolute validation loss. Finally, these LSTM networks were able to beat our simple baseline of guessing the final input value. With additional work, we think this simple model

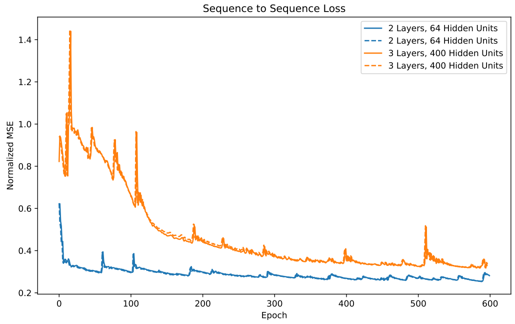


Figure 4: Many-to-Many loss: 2 layer, 64 hidden vs 3 layer, 400 hidden. Solid lines are training loss and dotted lines are validation loss

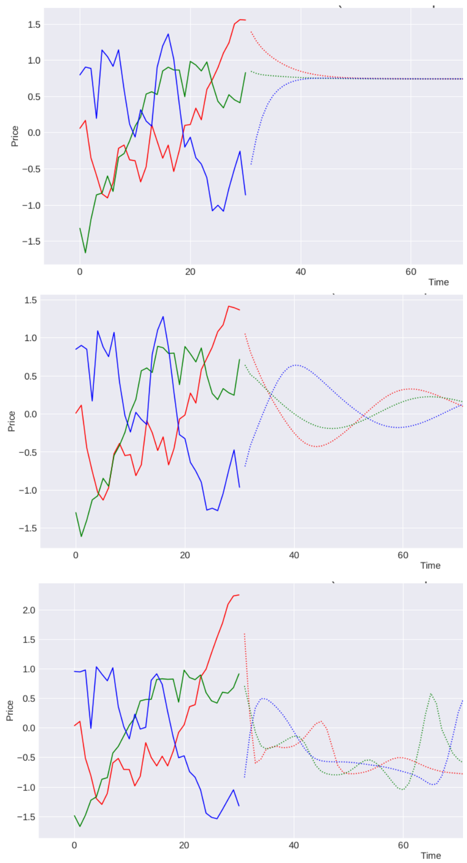


Figure 5: Generated sequence vs train time. solid lines represent true data and dotted lines represent generated data. Note the progression from dampened, to single sinusoid, to sum of sinusoids.

could lead to viable trading systems.

Future Work

Some future directions we can explore in include evaluating models using methods and metrics like profit, directional accuracy, and bull ratio [1].

Currently, we are considering stocks as independent from one another when in reality, market trends influence prices globally, or competition between companies could produce negative correlation in their price. Instead of using data where each time step as a single price for once stock, we could increase dimensionality to account for multiple prices from various stocks. Another approach to consider would be incorporating properties other than price into our data, including volume, open and close, various ratios, day of the week, and other financial metrics that are influential. We believe with these additional features, we will be able to capture much more nuanced information not evident in stock prices alone.

Another natural approach to time data is via convolutional neural networks. Wavenet [10], a recent architecture developed by Google Brain, is a generative model that uses dialated 1D convolutions to generate human speech audio sample by sample. Of course the range of human speech is more restricted than potential stock data, but it would be interesting to compare this convolutional model to more traditional recurrent models.

References

[1] Aamodt, Torkil. Predicting Stock Markets with Neural Networks. MS thesis. 2015.

[2] A Dual-Stage Attention-Based Recurrent Neural Network for Time Series Prediction. Qin, Y., Song, D., Cheng, H., Cheng, W., Jiang, G., Cottrell, G. International Joint Conference on Artificial Intelligence (IJCAI), 2017

[3] Waibel, Alexander, et al. "Phoneme recognition using time-delay neural networks." Readings in speech recognition. 1990. 393-404.

[4] Jaeger, Herbert. "The "echo state" approach to analysing and training recurrent neural networks-with an erratum note." Bonn, Germany: German National Research Center for Information Technology GMD Technical Report 148.34 (2001): 13.

[5] Saad, Emad W., Danil V. Prokhorov, and Donald C. Wunsch. "Comparative study of stock trend prediction using time delay, recurrent and probabilistic neural networks." IEEE Transactions on neural networks 9.6 (1998): 1456-1470.

[6] B. L. Sturm, J. F. Santos, O. Ben-Tal, and I. Korshunova, "Music transcription modeling and composition using deep learning," CoRR, vol. abs/1604.08723, 2016.

[7] https://lilianweng.github.io/lil-log/2017/07/08/predict-stock-prices-using-RNN-part-1.html

[8] S. Selvin, R. Vinayakumar, E. A. Gopalakrishnan, V. K. Menon and K. P. Soman, "Stock price prediction using LSTM, RNN and CNN-sliding window model," 2017 International Conference on Advances in Computing, Communications and Informatics (ICACCI), Udupi, 2017, pp. 1643-1647.

[9] D.G. Torres, H. Qiu, Applying Recurrent Neural Networks for Multivariate Time Series Forecasting of Volatile Financial Data. https://www.researchgate.net/project/Applying-Recurrent-Neural-Networks-for-Multivariate-Time-Series-Forecasting-of-Volatile-Financial-Data

[10] Aaron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, Koray Kavukcuoglu. WaveNet: A Generative Model for Raw Audio. arXiv:1609.03499.