A LSTM-CNN hybrid approach to predicting stock market prices

Edward Zhang
University of California, Los Angeles
edzhang@ucla.edu

Jack Zhang
University of California, Los Angeles
dvorjackz@gmail.com

Abstract

Neural networks have a variety of potential applications to the stock market. In this paper, we attempt to find an ideal model architecture to predict a stock's closing price one day into the future. To avoid problems such as the one-step lag predictor and extremely noisy data, we found that a parallel CNN-LSTM architecture was able to achieve the best performance and produce the most sensible results.

1. Introduction

1.1. Architecture

Forecasting stock prices requires not just a single previous price, but a history of several previous prices. Thus, it is appropriate to pursue a machine learning architecture that includes a recurrent layer. For our model, we experimented with several architectures.

We first tested a simple architecture of an LSTM layer feeding into a fully connected layer. When we encountered issues with this architecture, we tried adding additional LSTM layers, additional FC layers, as well as a dropout layer after the first LSTM layer with no discernible effect.

Inspired by similar attempts to harness convolutional layers to augment our model[1][3], we first experimented with using just a convolutional layer without a LSTM. This model was able to perform better.

Combining the convolutional neural network with the LSTM in parallel proved to be the most effective, achieving the lowest test error by far.

A full description of the architectures used can be found at the end of this paper.

1.2. Loss function

The model uses mean-squared error (MSE) as the loss function, since we were looking to minimize the difference between the actual price and the predicted price.

1.3. Data

Stock data was pulled from Alpha Vantage. Our data included the 20-year daily price time series for all (3563) NASDAQ stocks. For our training data, we used three stock price metrics: close price, price range, and volume. We chose a sequence length of 122 (a third of a year) for each data point. These sequences formed our inputs, and our outputs consisted of the close price following each sequence.

Data augmentation was performed by using a step size of one instead of stepping by the sequence length. For example, our inputs would be price[0:122], price[1:123], price[2:124], etc. instead of price [0:122], price[123:245], price[246, 368], etc, where price is an array of all historical prices for a stock. Each price is normalized using the statistics of the sequence that it belongs to.

We used a rough 90/10 split for training and validation data. The data was split by stock ticker to ensure there was no data leakage between our training and validation data.

2. Results

2.1. Naive LSTM solution

Initially our model architecture consisted of a single LSTM layer followed by a fully connected layer. With this architecture, training converged in about 1 epoch, with any subsequent training resulting in overfitting and an increase in validation loss. With this barebones LSTM solution, we were able to achieve a MSE of approximately 0.151.

While this seems to be reasonable performance at first glance, upon inspecting a plot of several predictions compared to actual data (shown in the top-right graph of Figure 1), it becomes clear that our model has devolved to a "one step lag predictor" (shown in the top-left graph of Figure 1), which simply outputs the last data point in the input sequence. If we compare the loss of a one step lag predictor to our trained model, they match almost exactly, with both models having similar validation loss.

Adding additional features such as extra LSTM and dropout layers did not improve performance or solve the problem.

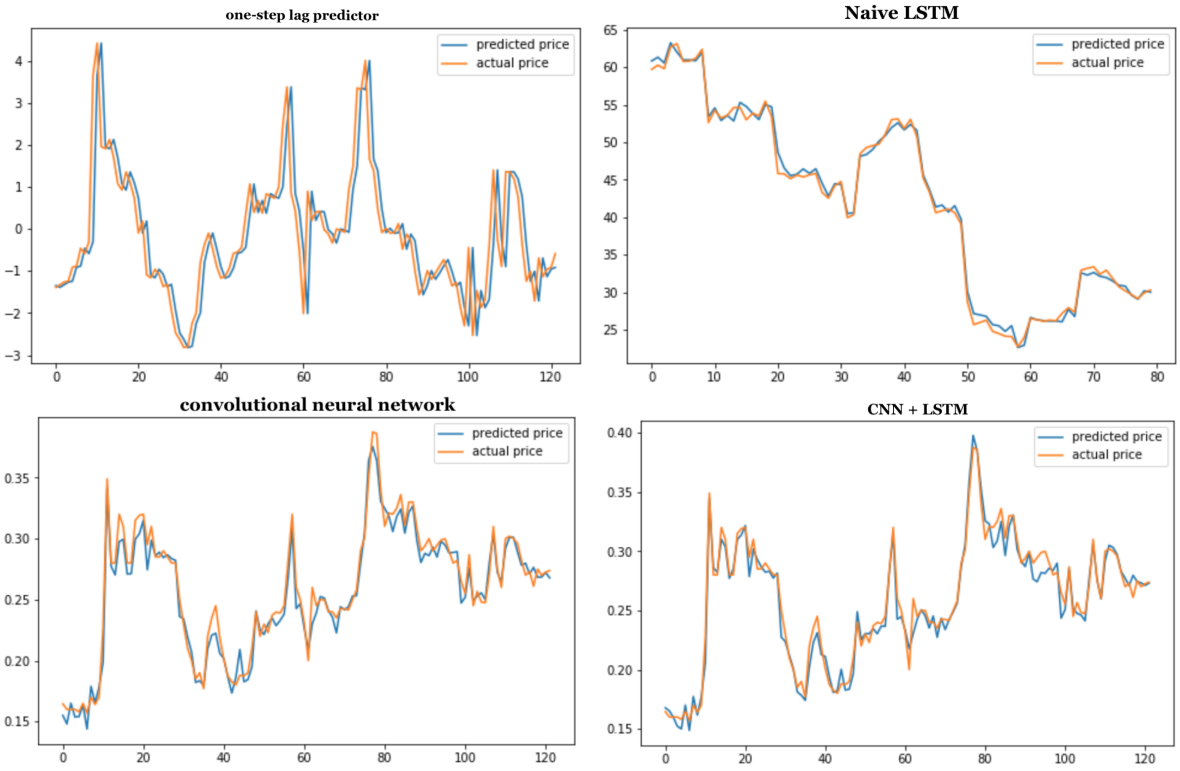

Figure 1. On the left, we have a slice of the of the naive LSTM, where the predicted prices are simply the last seen prices in the input sequence, resulting in the predicted price line looks like the actual price shifted forward one day. On the right, we have a slice of the hybrid LSTM-CNN, which visibly (right) does not exhibit this behavior as much.

2.2. Standalone CNN

Before we decided to add a convolutional layer to our LSTM architecture, we wanted to see how the CNN would perform on its own. Surprsingly, we were able to achieve a MSE of approximately 0.087, which is much better than before. Also, the predictions by the model (bottom-left graph of Figure 1) seems much less similar to those made by that of a one-step lag predictor.

2.3. Hybrid LSTM-CNN solution

We decided to increase the expressiveness of our model by adding a convolutional layer that would process the data in parallel to our existing LSTM layer. We trained this model for 2 epochs, although further training seems to improve validation loss without overfitting. This hybrid LSTM-CNN solution outperformed both the naive LSTM solution as well as the one step lag predictor with a MSE of ~ 0.067.

When observing a plot of several predictions compared to actual data (shown in Figure 1), we can see that our new model bears much less similarity to the one step lag predictor. In particular, we notice that the large peaks and troughs are predicted with great accuracy with our new model, whereas with the naive LSTM solution the peaks and troughs would nearly always be predicted at least one time step late.

2.4. Feature Engineering

In order to help our model better learn from our training data, we added two new features to each data point: volume (which was retrieved from the Alpha Vantage API), and the difference between the high and low prices. We then removed the high and low prices, as they seemed redundant. Adding these new metrics to our data and training on our naive LSTM model, we were able to outperform the one step lag predictor.

3. Discussion

Initially, our data features consisted of open price, high price, low price, and close price. However, after observing our naive LSTM devolve into a one-step lag predictor, we decided to change the features of our data, since all four features are essentially equal to one another. We removed open price, which was almost completely identical to close price, replaced high and low price with their difference, and added volume. Compared to close price, the additional fea-

tures, price range and volume, follow a different intrinsic data distribution and contain data that is less sequentially related. This helps the model form more meaningful connections and discourages the model from simply picking the previous data point in time.

As we can see from Figure 1, the predicted price looks much less like a shifted version of the actual price in the middle graph, where we made the changes to the data features, than in the left graph. This suggests that, whenever data is extremely sequentially related, we can help prevent the model from exhibiting one-step lag predictor tendencies by augmenting the data with features that are less sequentially-related and are distributed differently.

Additionally, in the presence of data prone to training models emulating the one-step lag predictor, we observe that traditional methods to improve model performance such as adding extra layers, regularization, and generalization techniques do not improve the situation. Once the model learns that it is able to easily achieve a low loss by making its prediction the previous input point, learning saturates and it is difficult to improve performance without an architectural change.

The idea behind using the convolutional neural network came from the recognition that stocks prices follow cyclic patterns and that convolution, which smoothed daily price changes into longer-period price changes, would prevent the one-step lag phenomenon. As a standalone architecture, the convolutional neural network feeding into a fully connected layer achieved a MSE of approximately 0.152, which is much better than the one-step lag predictor.

When the convolutional neural network is combined with the LSTM in parallel, as we saw above, the model had the best performance. A possible conclusion we may draw is that the convolutional neural network helped by adding activations that were much more generalized than the LSTM's activations, and as a result, decreased the likelihood that the model prioritized LSTM activations that would result in picking the previous day's price.

Further work could include training a model to predict whether a stock price will rise or fall (binary classification) or further optimizing the hyper-parameters of our current solution. We could experiment with deeper convolutional block and LSTM blocks which incorporate batch normalization, or perform further feature engineering that could extract more powerful features out of our current data.

References

[1] S. Chen and H. He. Stock prediction using convolutional neural network. *IOP Conference Series: Materials Science and Engineering*, 435:012026, 11 2018.

[2] H. D. K. Fazle, S. Majumdar and S. Chen. Lstm fully convolutional networks for time series classification.

[3] T. Kim and H. Y. Kim. Forecasting stock prices with a feature fusion lstm-cnn model using different representations of the same data. *PLOS ONE*, 14(2):1–23, 02 2019.

4. Performance

The performance of each of our model architectures is as follows:

Method	Dataset	MSE
One Step Lag Predictor	Feature-Engineered	0.142
Naive LSTM	Non-Engineered	0.152
Naive LSTM	Feature-Engineered	0.142
CNN	Feature-Engineered	0.109
Hybrid LSTM-CNN	Feature-Engineered	0.067

Table 1. Performance of each model architecture on either the non-engineered dataset, which is the original cleaned data from the Alpha Vantage API, or the feature-engineered dataset, which includes the difference between high and low prices as well as volume.

We notice first that a one step lag predictor has comparable loss to the naive LSTM models. However, the naive LSTM model sees an improvement in performance when trained on the feature-engineered dataset versus the original dataset.

We also notice that taking only the Convolutional block from the hybrid model results in a model that still outperforms the naive LSTM as well as the one step lag predictor.

5. Architecture

5.1. Naive LSTM

The naive LSTM model was a 122-unit LSTM layer, a dropout layer, and a 20-unit fully-connected layer, as shown in Figure 2.

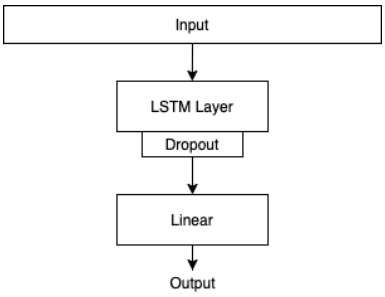

Figure 2. Naive LSTM architecture.

This architecture was modified to also have additional LSTM layers, but these augmentations showed little improvement in performance.

5.2. CNN

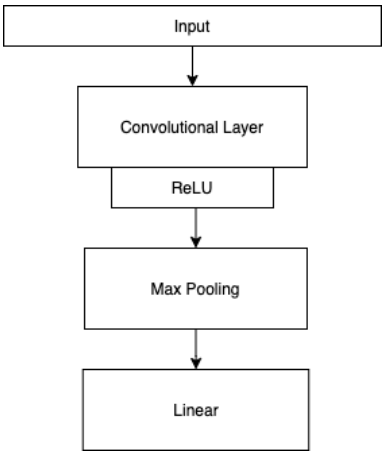

Figure 3. CNN Architecture.

The naive CNN model is identical to the CNN architecture, but without the LSTM block.

5.3. Hybrid LSTM-CNN

The LSTM-CNN parallel model consists of a 1-dimensional convolutional block and a LSTM block which process the input in parallel, as shown in Figure 4. This architecture was inspired by similar architectures that have been used to successfully classify time series sequences. [2].

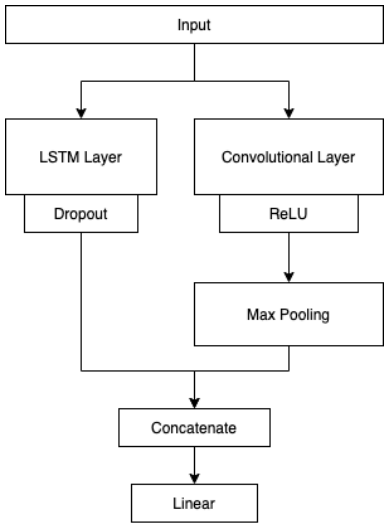

Figure 4. Hybrid LSTM-CNN architecture. The convolutional block could potentially be augmented with more convolutional layers and batch normalization layers.

The convolutional block consists of a single 1-dimensional convolutional layer with 32 filters of size 7, followed by a ReLU activation. This is then fed into a 1-dimensional max pooling layer with a filter size of 3 to reduce the number of features.

The stock time series input is simultaneously processed by a 122-unit LSTM layer. The output of the LSTM block is passed into a dropout layer, using a dropout value of 0.5.

The resulting output is then flattened and concatenated with the output of the LSTM layer. The resulting features are then passed into a linear fully-connected layer.