LSTM Recurrent Neural Network for Forecasting S&P 500 Post Financial Crisis

March 8, 2020

Authors: Leiwen Gao (UID: 804681806, email: gaoleiwen@ucla.edu), Redmond Xia (UID: 405435340, email: redxia@g.ucla.edu), William Lee (UID: 904814655, email: williamjlee1@gmail.com)

Abstract

We explore the Recurrent Neural Network (RNN) for problems in financial predictions. Financial prediction problems often involve large data sets that often basic economics models cannot handle. For this, we turn to deep learning hierarchical models in conjunction with some finance theory to help solve the prediction problems that standard/traditional methods in finance cannot provide. In particular, we explore the prediction accuracies of S & P 500.

1. Introduction

The global economy seems to be heavily around the S & P 500. Basic finance theory (Capital Asset Pricing Model, CAPM) tells us that if the market is perfect then no investor could beat the market. As a result, all investors should hold the market portfolio. The usual benchmark for the market is the S & P 500 because it best represents the U.S. stock market and is assumed to be one of the most perfect markets. When a recession hits, the S & P 500 drops in correspondence. Thus, when one could accurately predict the S & P 500, we can predict the global economy and at the same time profit from this. A deep learning framework could make an accurate prediction if we assume a perfect market, i.e. prices of the stock market reflect all public and private information and its observed market price is taken as a true reflection of the market. As we break away from the S & P 500, such as predicting an individual company, we start to lose more of the perfect market assumptions, implying the prices reflect less and less of the information. We use the Long Short Term Memory (LSTM) RNN to model the S & P 500 due to its markov assumptions that naturally applies to stocks. We also compare the recurrent temporal nature of these RNN networks with a more spatial associated network, the Convolutional Neural Network (CNN) in performing this same task of stock prediction.

1.1 Data & Methodology

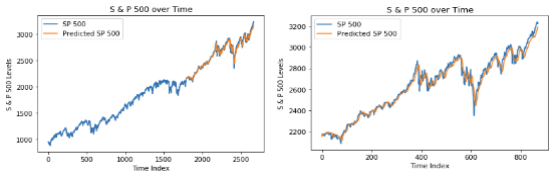
In this project, we initially aim to forecast the S&P 500 post financial crisis (06/01/2009-12/31/2019, giving 2666 data points) using data provided from Center for Research in Security Prices (CRSP) through Wharton Research Data Services (WRDS).[3] We transform the dataset using the keras MinMaxScaler method and used Adam optimizer for all methods. Details about methodology (forward chaining) and model settings are described in the Appendix.

2. Results

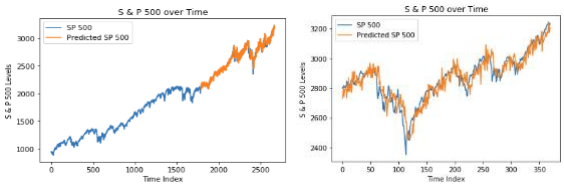
Overall the model seems to predict the S & P 500 really well. The predictions tend to follow the trend really well. We provide results of the CNN model, simplest RNN model, simplest RNN V2, adding forward chaining addition to it, and tuning it. The forward training method can be found in the appendix. Initially for training, we chose the first ⅔ of the dataset to be the training set and the last ⅓ to be the test set (giving us a dataset size of 1786 and 880, respectively) to run the model with a 2:1 train test split. This corresponds to 06/01/2009-07/05/2016 dates for the training set and 07/06/2016-12/31/2019 for the test set. We summarize our models below.

2.1 CNN Results

For 2 epochs, batch size of 10, and look back of 10, the model only took about 2.8 seconds to train. The results also achieved a promising 44.17 RMSE when tested on the last ⅓ of the dataset (note: that the testing done on the last part of the dataset meant that we ran the model on all sets of 10 consecutive stock prices within the last ⅓ of the dataset and comparing the predicted next stock price with the actual price). The following compare the predicted and actual prices (right plot zooms in on the testing set):

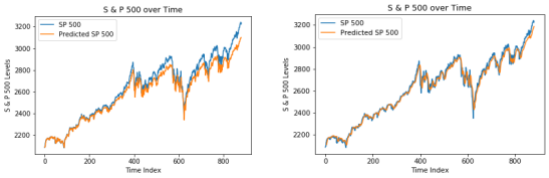


The next method we used to evaluate the CNN model is the forward chaining method. Here, we just start training on the first ⅔ of data, predict the next timestep, and expand the training set one time step to predict the next time step and continue in this fashion. We trained the model with batch size of 10 and 3 epochs at time to get an RMSE of about 50.10 and took only about 46 minutes. The following is a plot of the predicted vs actual stock prices (the right plot zooms in on the test set).



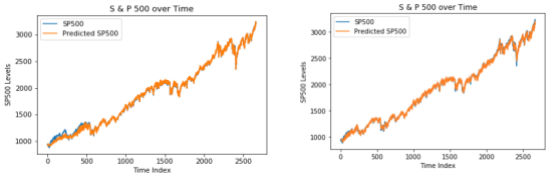
2.2 Simplest RNN Model V1 and V2

The simplest models we start off from using the same split of training set and test set as CNN. We find that the Simplest Model V2 gets better predictions with smaller overall RMSE of 36.55 compared to V1 of 57.3. Plots of the test set are shown below, with V1 on the left and V2 on the right.



2.2 Forward Chaining V1 and V2

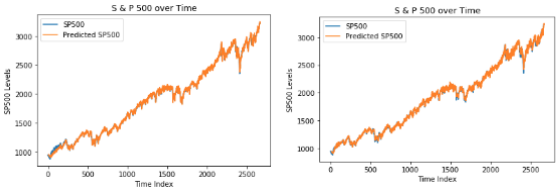
Forward Chaining V1 provides a longer prediction interval from the 6th day of data points with lag 1 data. The predicted values for the first 500 observations performs poorly, but performs well for the following time points, resulting in an overall RMSE of 28.87. Then we move to lag 5 data as Forward Chaining V2. The prediction interval starts from the 10th day of data points with the overall RMSE of 33.54 which is larger than V1, with longer running time. The left plot is for the model V1 and the right one is for V2.



To optimize the performance of Forward Chaining V1 and V2 for different lags, we added more nodes to the hidden layer, having 64 nodes and see if the models as V3 and V4 improved in the following sections.

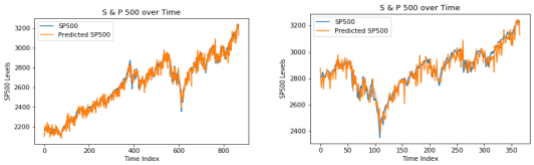
2.3 Forward Chaining V3 and V4

Forward Chaining V3 (plot on the left) achieved the smallest RMSE, decreasing from 28.87 to 25.28. For Forward Chaining V2 (plot on the right), the overall RMSE decreases from 33.54 to 30.89 but still larger than V1 and V3.



2.4 Forward Chaining V5

We fitted the Forward Chaining V5 on the test size back in simple V2 to reduce computation time. The overall RMSE was 44.5 which is even worse. The left plot is for the test sets and the right one zooms in on the last 370 observations.



3. Discussion

Dealing with high frequency of data is difficult. More often than not, the data have complex and nonlinear interactions that are not well specified in financial economic theories e.g. CAPM is a linear method thus is prone to outliers. Thus, we have proven CNN and RNN as a powerful method to capture more information in the complexities of the data. We discuss using CNN and RNN below.

3.1 CNN

The high-level reasoning behind choosing the CNN model architecture was that since we used a look back of 10 (meaning that the 10 previous stock prices were used as input to train the model to predict the price at the next timestep), we needed to reduce the size of the dimensions by learning the important features of the prices of 3 consecutive time steps, then of 3 consecutive averages, and so on. This is key because features that are nearby in time are likely to be correlated and produce enough a signal to predict the output. Moreover, average pooling layers were used instead of max pool because it is more intuitive to allow all the considered features being averaged to influence the output, rather than just the maximum. Lower stock prices should also be able to influence the features that result in the output. In general, the CNN performed slightly worse than the RNN which is expected since RNNs are better designed for time series and sequential data. The LSTM being a memory, can for

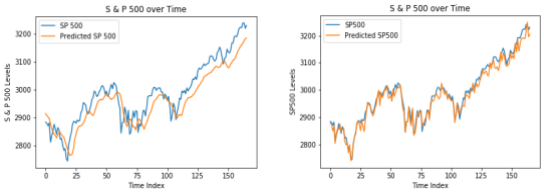
example likely take into account how far back a certain pattern was observed and how that may influence the next stock price more than fluctuations further back in the look back history.

3.2 Simple V1 vs. V2

The difference between simplest V1 and V2 was that we added one more layer and added more nodes from 4 → 64, respectively. Afterwards, we lowered the epoch from 5 → 1 and changed the lagged value from 1 → 5, respectively. The RMSE was significantly reduced by more than half from 57.3 → 36.55, respectively. Both of the plots were able to capture the shape of the curve really well. The time it took was also reduced from 57.3 → 26.25 largely due to smaller epoch's, respectively. When we zoomed in, we are able to see that V1 levels off after roughly 100 business days from 07/05/2016. The standard error in the linear regression for the test set was 98.54. Thus, our model was able to have a lower standard error than the linear standard error. When we zoomed in on the test set for V2, we observed that the neural network model often gets the average right but undershoots the volatility often.

3.3 Simple V2 vs. Forward Chaining V1

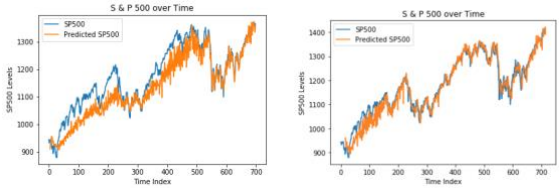
For the same test set, the forward chaining model is able to get better results and higher prediction range. This comes at a greater cost of longer run time for fitting the model n(n+1)/2 times. However, the overall RMSE is much lower, which is 28.87. For the same time frame in the simple model test set, we were able to achieve an RMSE of 15.36. We also used a much simpler model (Less nodes or layer, etc..) than the Simple V2. This is likely due to the model is learning more about the recent behavior of the S&P 500, in the forward chaining method and so it is better able to capture the volatility more. When we zoom in on the last 160, we confirm this is the case. Again, Left is Simple V2, and Right is the Forward Chaining V1 model. We would say the simple model is more biased but has less variance, and the model on the right is a little more noisy and overfits the data more, but is less biased.



We conclude that we may need to adjust the overfitting of the model through introducing drop out or regularizations. In this next section, we look at a more complicated model to confirm that we were overfitting.

3.4 Forward Chaining V1 vs. V2

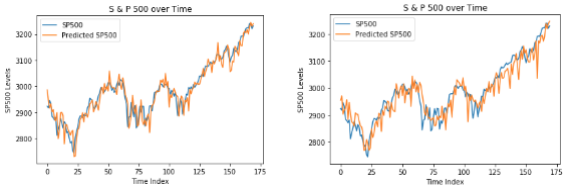
When using the forward chaining method we expected the initial training set to do poorly because we do not have a lot of input into the neural network. Usually, one would pick a neural network framework over traditional machine learning method when the input data is too large. The plots below shows the first 700 observations of the forward chaining results with Forward Chaining V1 on the left and V2 on the right. When the number of observations is not large enough for input, the model V2 performs better than V1 because it provides more information using multiple time steps (lag 5).



However, when the number of observations is large enough and the input contains enough information, the Forward Chaining V1 has better performance, captures volatility more, and results in an overall smaller RMSE. The same holds true for V3 and V4, and details are discussed in the next section.

3.5 Forward Chaining V3 vs. V4

By zooming in the plot of last 170 observations for Forward Chaining V3 (lag 1) on the left and V4 (lag 5) on the right, the plot of predictions for the model V4 looks shifted compared to the true value, which leads to larger error for lag 5 data. This is likely due to more weight in the past 5 data values, and the model is learning too much from it. This reinforces the idea that stock prices cannot learn too much on historical values through the recurrent matrix.



Based on plots in section 2.2 and 2.3, It's clear to see that adding more nodes to the hidden layer improves the model's performance for the first few observations which doesn't include enough input information, leading to smaller overall RMSE. However, referring to the summary table in the appendix, for V3, although the overall RMSE is smaller than V1, the RMSE in test set is much larger which implies the overfitting due to the extra number of nodes when there are large amounts of inputs. The poor performance of lag 5 data indicates the complexity of the model that too much input information is being used which leads to overfitting of the data.

3.5 Forward Chaining V4 vs. V5

When we increased the model complexity by adding more layers to V4, the model V5 got a noisier prediction as shown in 2.4 which implies that the more layers we use, the more we overfit the data. Dropout was used to solve the overfitting problem but doesn't seem to have enough effect. Thus, we expect the same is true if we were to use 8 layers with 8 nodes each to behave in the same way. We suspect that adding more layers will use the lag data in each layer, which in turn makes the model worse. We leave this to be a further research implementation.

Conclusion

Initially, we see that the CNN performs worse than the RNN for forecasting the S & P 500 levels. As mentioned before, this is evidence that CNN lacks recurrent connectivity. Possible future attempts at using a CNN could involve using 2D convolutions instead, and stacking the input with multiple technical indicators such as RSI, Williams %R, Simple Moving Average, etc. to generate a 2D tensor input that will be far richer as they contain more features that can be interpreted further by a CNN. However, we also showed evidence in RNN that putting larger history tends to perform worse. This implies that using longer past history is not good at predicting the future in the recurrent matrix. Thus, picking the simplest forward chaining recurrent neural networks, we were able to achieve the smallest test RMSE. When compared to the residual standard error of 107.8 in a standard linear regression model for the time frame of 06/01/2009-12/31/2019, Neural Network outperforms. There does seem to have a lot of additional benefits from running the forward chaining methods, however it comes at a computation cost. So when comparing different architectures, the model will need to run N(N+1)/2 times, which is something that doesn't seem feasible overnight. Dimension reduction should come into play here. An example would be running the model on S & P 500 on monthly data. For future research, one should utilize more information about the S & P 500, such as volume, highs, lows, open, and the volatility to see if they play a significant role as well as combining several stock indicators (i.e. RSI, Williams %R, Simple Moving Average, etc.) in the input to have a richer input space than just the stock prices. Another idea is to see if we can improve the prediction through the stationarity assumptions. That is, predict the returns of the S & P 500, since it is likely to give a pattern in the long run.

References

[1] Prof J. C. Kao, UCLA ECE, Lecture 13, slide 100

[2] Prof J. C. Kao, UCLA ECE, Lecture 13, slide 123

[3] CRSP Index / S&P 500 Indexes. (2009-2019). Available: Center For Research in Security Prices. Graduate School of Business. University of Chicago [March 8th, 2020]. Retrieved from Wharton Research Data Service.

[4] Brownlee, J. (2019, August 5). Time Series Prediction with LSTM Recurrent Neural Networks in Python with Keras. Retrieved from https://machinelearningmastery.com/time-series-prediction-lstm-recurrent-neural-networks-python-keras/

Appendix

1.Summarization of architectures

1.1 Convolution Neural Networks

Before starting with the RNN models, we first tried the CNN. In order to come up with the architecture of the model, we had to consider how to most effectively incorporate the time series data of stock prices as inputs to a CNN model. For this project, we ended up just having the CNN predict the price at the next time step given the prices of a set of previous timesteps. One intuitive way to do this is to consider a range of consecutive time slices (i.e. prices for 10 time consecutive steps) as a 1 dimensional vector that we can apply 1D convolutions on in order to predict the next price (this is what we called the look back of 10 steps). We thought this would intuitively work reasonably since the CNN is able to capture the spatial input features which are essentially temporal patterns in the stock prices since the input vector is arranged in chronological order so that the model is able to capture patterns in the stock prices around a certain time for example. The model we experimented with had the overall architecture:

`Input -> [Conv1D - AveragePooling1D]x2 -> FCx2`

The convolutions had kernel size of 3, 128 filters for the first convolution and 64 for the second. The average pooling layers both had pool size of 3 and stride of 1. We also used 64 hidden units for the 2nd to last FC and the last FC maps to the output of length 1 (a single scalar). All activations used for the convolutions and the 2nd to last FC were ReLUs.

1.2 Recurrent Neural Networks

Recurrent Neural Networks is a hierarchical network that address typical problems that Convolution Neural Network lack recurrent connectivity. In other words, it produces the same output given the same input, irrespective of what happened in the past. We run the Long Short Term Memory (LSTM) due to the exploding gradient problem that normal RNN have. In running the RNN, we start off with the simplest RNN model, i.e. one hidden layer and lag one input with the adam optimizer. After seeing the promising results, we switch to increase the complexity to see if the results can improve. Two simplest models were fitted by splitting the data to training set and test set with the ratio 2:1. Then we fitted five models utilizing the forward training method, which means we update our training set with the real-time current data and forecast ahead to validate our model. For example, let's say we observed the process $x_t = \{1,2,3,4\}$ for $t = 1,2,3,4$ respectively. Let's claim our estimator is the average of the past values, i.e. $\hat{\theta}_{t+1} = \sum_{i=1}^{t} x_i$. Although it is not a very good estimator but for the sake of example it does its job. For $t = 2$, we only observed $x_2 = \{1,2\}$. Based on our function, we claim $\hat{x}_3 = 1.5$. Then we update our time step to $t = 3$, and we refit our model using $x_3 = \{1,2,3\}$ to get $\hat{x}_4 = 2$. This provides an $RMSE = \sqrt{\frac{(3-1.5)^2 + (4-2)^2}{2}} = 1.77$. This is typically used when implementing time-series method, so that we do not induce an in-sample bias by using future values. We validate the model based on the RMSE, implying we check if different layers or architecture improves the RMSE. Since the forward chaining method involves evaluating the model's hyperparameters, we are using the dataset as a validation set and not as just a test set. The results were promising for the one period ahead forecast. Parameters were tuned for different models, including the number of hidden layers, number of nodes in each hidden layer, number of epochs, batch size and look back. Look back is telling the RNN how much lag data we should use to predict the next values. For example, if look back = 5, the last 5 time points are used to predict the next time step.

- Simplest Model V1: hidden layers = 1, number of nodes = 4, epochs = 5, batch size = 1, look back = 1, which is the lag 1 data.
- Simplest Model V2: hidden layers = 2, number of nodes in the 1st layer = 64, number of nodes in the 2nd layer = 64, epochs = 2, batch size = 5, look back = 5, which is the lag 5 data.
- Forward Chaining V1: hidden layers = 1, number of nodes = 8, epochs = 3, batch size = 1, look back = 1
- Forward Chaining V2: hidden layers = 1, number of nodes = 8, epochs = 3, batch size = 1, look back = 5
- Forward Chaining V3: hidden layers = 1, number of nodes = 64, epochs = 3, batch size = 1, look back = 1
- Forward Chaining V4: hidden layers = 1, number of nodes = 64, epochs = 7, batch size = 2, look back = 5
- Forward Chaining V5: hidden layers = 3, number of nodes = 64, epochs = 7, batch size = 2, look back = 5, dropout is at 15% after the first layer and the second layer.

2. Summarization of performance for all algorithms

Algorithms	Overall RMSE	RMSE for test set	Running time
CNN			
CNN 2:1 Train Test Split	-	44.17	2.8s
CNN Forward Chaining	-	50.10	46m
RNN			
Simplest V1	-	57.3	57s
Simplest V2	-	36.55	26.3s
Forward Chaining V1	28.87	15.36	6.6h
Forward Chaining V2	33.54	40.84	12 - 16h
Forward Chaining V3	25.28	28.17	10.1h
Forward Chaining V4	30.89	37.98	16.9h
Forward Chaining V5	-	44.5	6.9h