CS274P

Neural Networks and Deep Learning

Final Report

Winter 2019

1 Team Details

1.1 Members

Group #8:

Li-Yan Wang (Hayden)	11004461	liyanw3@uci.edu
Tzu-Yu Chao (Zoe)	50002066	tzuyuc@uci.edu
Pin-Ying Wu (Iris)	37723788	pinyingw@uci.edu
Alex Chu	73362242	alexc8@uci.edu
Kuo-Wei Chiao (Michael)	17365926	kmchiao@uci.edu

2 Project Summary

Stock prediction has always been a trending financial topic, but the prediction performance varies depending on the techniques people used. In this project, we aim to apply neural network knowledge based on what we've learned in class to this problem. We focus on predicting the following index: Taiwan Semiconductor Manufacturing Company Limited (TSM), with the help of dependent indices, iShares PHLX Semiconductor ETF (SOXX) and VanEck Vectors Semiconductor ETF (SMH), that we believe will have a great impact to our result. As for our model, we decide to use RNN as our base architecture, and we implement it with its successors, LSTM and GRU, to further increase the performance on predicting our target. Finally, we introduce hyperparameter to our model for hidden unit optimization. Though the number of days we can predict in the future for our target is currently too short to be commercialized for general purpose, our prediction has been proved to outweigh most of others with a relative high probability.

3 Introduction

The successful prediction of a stock's future price could yield significant profit, as well as avoiding potential losses. The goal of this project is to predict the "Taiwan Semiconductor Manufacturing Index" (TSM). TSM is an exchange-traded fund that tracks the performance of Taiwan's semiconductor industry. TSM is often viewed as a indicator of the technology market of Taiwan. This insight provides investors information on stock trading, and successful predictions could generate significant profit. However, general public does not have an easy access to accurately predict stock price. Therefore, we decide to create a stock prediction application to bridge this gap; we are interested on whether we could predict the trend of the technology market successfully using a model trained on past data, as well as models trained on other stock index data in the semiconductor/technology industries.

4 Model Architecture and Approach

4.1 Data

4.1.1 Source

We use a decade of stock history from Yahoo Finance as our sample size, which is equivalent to 2266 days. Yahoo Finance provides 5 parameters summarizing the daily movement of each stocks: Open, Close, High, Low prices, and Volume. Those information are fed into our neural network to predict the stock movement of TSM.

4.1.2 Input and Output Stock Indices

Taiwan stock market is highly impacted from external factors such as the U.S. stock market. We pick multiple exchange-traded funds to support our TSM prediction. The indices are Nasdaq 100 ETF (QQQ), Phoenix Semiconductor ETF (SOXX), Direxion Semiconductor ETF (SOXL), VanEck Vectors Semiconductor ETF (SMH), ProShares Ultra Semiconductors (USD). In this project, we want to see how these inputs influence our target, TSM, and furthermore, we expect to predict the trends of TSM for the near future.

The inputs of our neural networks are the past ETF and TSM data. Our output target is the future TSM price 5 days ahead, which is the average of high and low price of TSM. After evaluating the individual ETF effect on average TSM price, we observed that SMH ETF has the most significant contribution to the accuracy. Therefore, we decided to use only SMH ETF as our neural network inputs.

4.2 Preprocessing

4.2.1 Data Shifting

In order to predict the stock data, we have to do align the input to the corresponding predicted target. The input are the past data of TSM and ETF, and the output is future TSM average price numbers of day ahead. For example, if we are forecasting the price two days ahead. The input of the first day is predicting the price two days later, which is the third day. The second day of data predicts the fourth day of TSM price. The graph below (Figure 4-2.a) illustrates the relationship of input and outputs, which is offset by 2 days apart. The input needs to be shifted to the left from

the target data by n days in order to match the input to the target prediction.

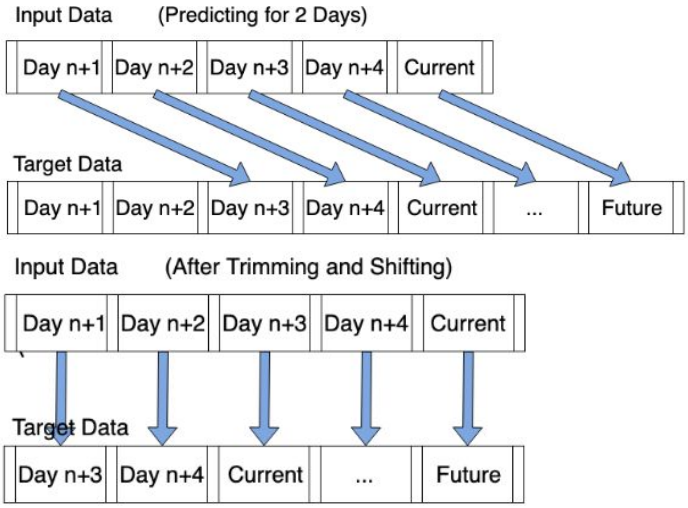


Figure 4-2.a Relationship between input data and target data.

4.2.2 Normalization

The table below (Figure 4-2.b) are the inputs of our parameters. The range of each parameters varies widely. The price of QQQ is 5 times more than TSM's price. The gap between volume and stock price is even greater. Without normalization, volume would have greater influence to the network, diminishing other features' influences. Therefore, we use MinMaxScaler to normalize each feature individually to a range of 0 to 1. We use fit function to find the maximum and minimum value from the training data. We then transform the data on both training and test data using the range we obtained from the fit function.

Date	QQQ Open	QQQ Close	QQQ High	QQQ Low	QQQ Volume	... (SOXX, TSM)	TSM Avg
2014-02-24	90.14	90.41	90.82	90.04	31500400	...	17.72

Figure 4-2.b Input parameters.

4.2.3 Tensor

After we get the scaled value, we can create tensor for our input of our neural network. Our tensor has 3 dimensions: features, sequence length, and batch size. Features are volume and different stock prices. Sequence length is how many continuous days we are going to evaluate at once, which is the window size. The graph below (Figure 4-2.c) illustrates the relationship between sequence length and batch size. Our total sample size contains 2261 days of stock data. We decide to evaluate 100 continuous days of data together as our window size, which is illustrated by the orange lines. The batch size indicates the number of windows selected. Each window has a random starting point with the fix length. After selecting our windows, the 3 dimensions tensor is fed into our neural network as our input.

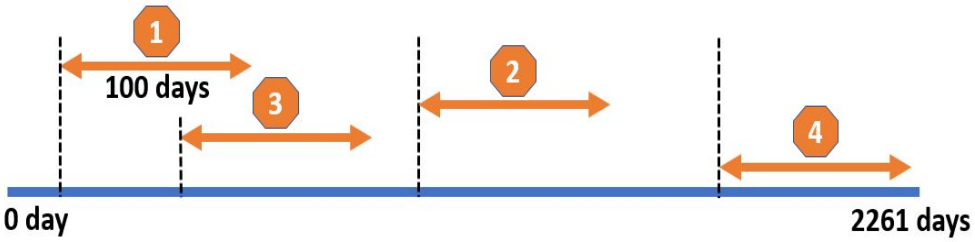


Figure 4-3.c Relationship between sequence length and batch size.

4.3 Model Architecture: RNN

Originally we thought the RNN model would be a good fit for us to use since our input is time-series data and we know that RNN is designed to make use of the sequential type of data. However, the RNN model suffers from the vanishing gradient problem: once a sequence is long enough, they'll have difficulties to carry information from earlier time steps to later ones, which occurs during back propagation. Fortunately, there are several established models that are able to resolve this problem, and we will use LSTM and GRU for this project.

4.3.1 LSTM

Long Short-Term Memory networks (LSTM) is able to regulates what to remember and what to forget with more gates: it has a forget gate layer that removes what we don't need, an input gate layer that updates our cells with new information, and a layer that filters out the most important part of information we need and pass it to the next cell.

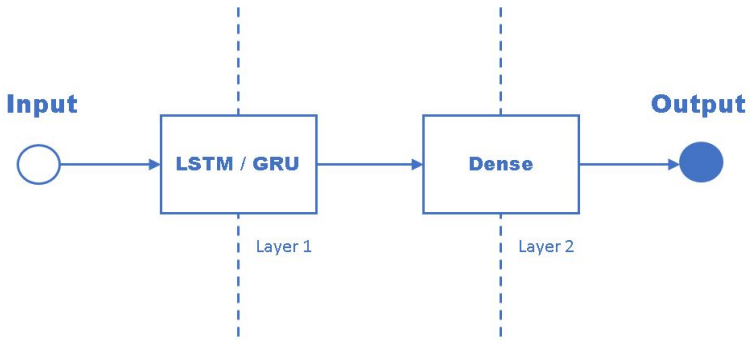
4.3.2 GRU

Another popular model that we will use is GRU, which stands for Gated Recurrent Units. The update gate acts similar to the forget and input gate of an LSTM; it decides what information to throw away and what new information to add. And then a following gate, called reset gate, is used to decide how much past information to forget.

GRU's has fewer tensor operations, so theoretically, they should be faster to train than LSTM. We implement these 2 models to compare the effect of these architecture on our prediction result.

4.3.3 Model Structure

We begin our project with simpler models, which consists 2 layers: the first layer is LSTM or GRU, and the second layer is our output layer, as you can see in this graph, and later we try some variations for this structure to find out which performs better in stock prediction based on our existing data sets.



4.4 Model Implementation

4.4.1 Model Input

To train our model, we split our data into training set and testing set, 80% for training and 20% for testing. In our first layer of our LSTM and GRU models, we use the tanh activation function and try to experiment with different units of neurons in the layer, such as 64, 128, 256, 512, to see how it affects our model's performance. In our second layer, which is the dense output layer, we experiment with two activation functions, sigmoid and ReLU, to see how it impacts the performance of the model.

4.4.2 Model Configuration

To configure our model, we need the loss function and an optimizer to compile our model. We use Mean Squared Error (MSE) as our loss function to measure how our model's output matches the true output. For the optimizer, we use RMSProp, which is commonly used in RNN networks. Furthermore, we use certain callback functions to track on the internal states and statistics of our model. We have a early stopping callback function that monitors our test loss and stops training when the model's performance worsens and a learning rate reduction callback function that reduces our learning rate when our test loss has not improved after certain amount of epochs. We adopt these 2 callback functions to make sure that our model does not overtrain.

4.4.3 Model Training

After we finish building our model, we train our model for 20 epochs, 100 steps for each epoch. Because we have the early stopping callback function, our model does not fully train for 20 epochs and stops at a certain time when the test loss does not improve. One thing to note is that we ignore the first 50 time steps of data because the model has seen little data at that point and the prediction might not be accurate. We call these first 50 time steps as warm-up steps.

4.4.4 Model Evaluation

Finally, we evaluate the performance of our model not only based on the computed test loss but also another metric called Explained Variance Score, known as EVS. Initially, we want to use accuracy to measure our model's performance and the reason why we didn't use accuracy is because we find out that accuracy is used for classification problems and that does not apply to our time series regression model. Therefore, we did some investigation on the metrics for evaluating the regression model and out of the metrics we found, we think that EVS would be a good option for us. The EVS calculates the ratio between the variance of the error and the variance of the true value. The highest score EVS can achieve is 1. So, the smaller the variance between the true value and the predicted value, the higher score our model achieves, and hence the better performance our model has.

4.4.5 Hyperparameters

Designing an optimal neural network architecture is time and labor intensive because many parameters needs to be tuned, such as hidden layer units and activation functions. When exploring the hidden layers, we tried 3 combinations of hidden layer units and 2 activation

functions. Manually finding the optimal combination requires constant attention and fails to reach the optimal solution. Therefore, hyperparameters are introduced to eliminate the issues. SHERPA is a hyperparameter library automating the tuning process through optimization, which is developed by Dr. Baldi's research group. We utilized SHERPA to find the optimal number of hidden units, and below is our model configurations:

- Input stock indices: TSM + SMH
- Input size: 10 years
- Model: GRU
- Activation: ReLU
- Shift days: 5
- Epoch: 10
- Trials: 10

5 Results: Model Performance Analysis

5.1 Units

Initially, we tried various different number of units for LSTM and GRU as a way to optimize and improve our model. Before we introduced hyper-parameter to obtain optimal units selection, we will manually tried 64, 128, 256 and 512 units. The results indicates that there are a significant drop in diminishing returns after 128 units in both LSTM and GRU models (Figure 5-1). We have also observed that GRU performs far better than LSTM, where GRU has around 0.6-0.7 EVS and LSTM has around 0.2-0.5 EVS in either combination of activation function (sigmoid or ReLU). Later in the report, we will introduce SHERPA to obtain an optimal hidden units. Furthermore, ReLU activation function output layer performs far better than sigmoid activation function output layer. The result can be interpreted that in our prediction model, ReLU performs better than sigmoid due to its characteristic eliminating the vanishing gradient problem, and GRU performs better than LSTM due to its simpler structure requirement and efficiency.

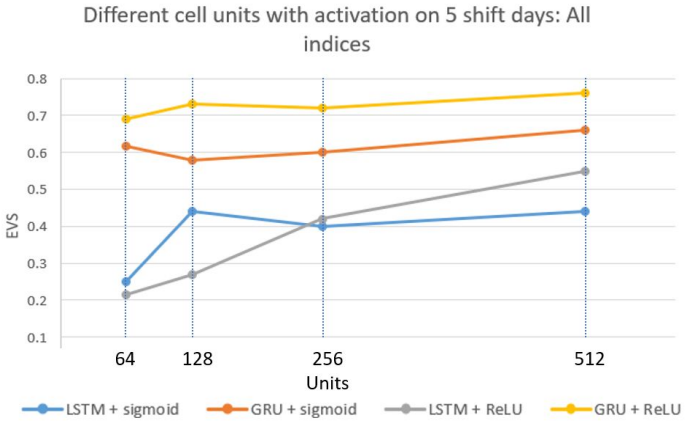


Figure 5-1. Different cell units with activation on 5 shift days for all indices (QQQ, SOXX, TSM).

5.2 Shift Days

Once we get settled down with the optimized model, which are GRU and ReLU layers, we want to see how much we can stretch shift days for the prediction. Ideally, we want to have better performance at predicting stock price for longer future. We tried various shift days ranging from 0

to 20 days with TSM index. The reason for choosing single TSM index for data input is to minimize the effect of other variables from multiple indices on measuring the effect of shift days.

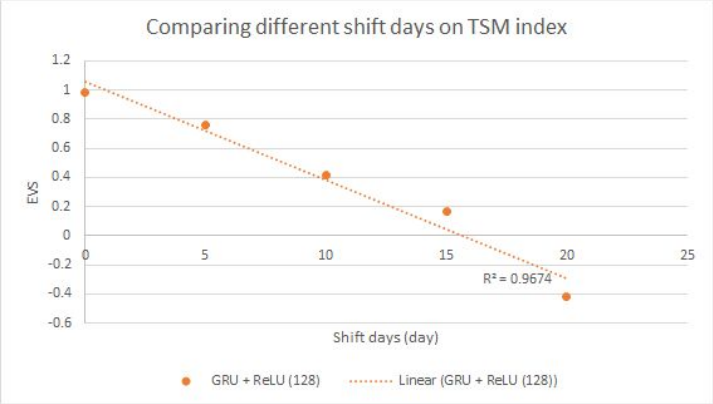


Figure 5-2. Comparison of different shift days on TSM index.

The result is interesting that there is an inverse linear relationship between the shift days and performance with R^2 value of 0.9674 (Figure 5-2). We decide to select 5 shift days for future experiments as it has a decent performance, and for the fact that the stock market opens for 5 days and closes for 2 days, giving a good future prediction for a week.

5.3 Multiple stock combinations

As we decide for the 5 days shift for stock price prediction, we now want to see how individual stock index has an effect in the multiple index data output. Individually speaking, QQQ has the worst performance around -0.6, SOXX has slight improved performance, and TSM has the best performance on predicting the stock price (Figure 5-3.a). However, in the combination of multiple indices, there are no significant improvement in the performance, in fact, there are some decrease in the performance especially with combination including the TSM index. This is possibly due to the fact that two indices acted against each other, hindering the prediction of stock price by the model.

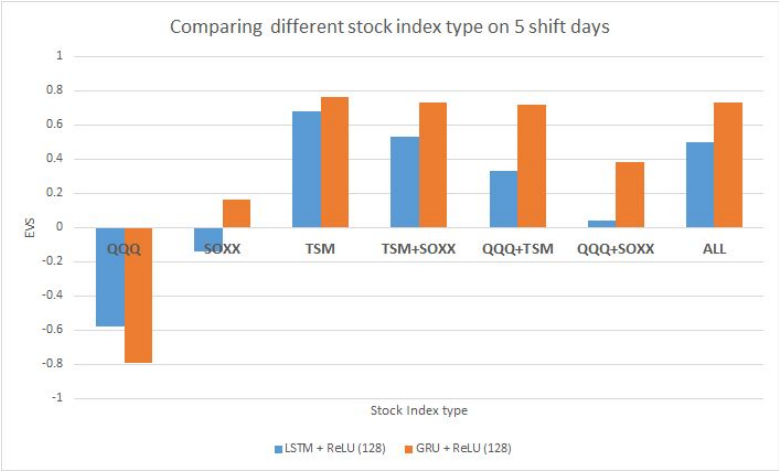


Figure 5-3.a Comparison of different stock index types on 5 shift days.

With GRU and ReLU layers, we are able to increase our EVS performance to 0.72 (Figure 5-3.b).

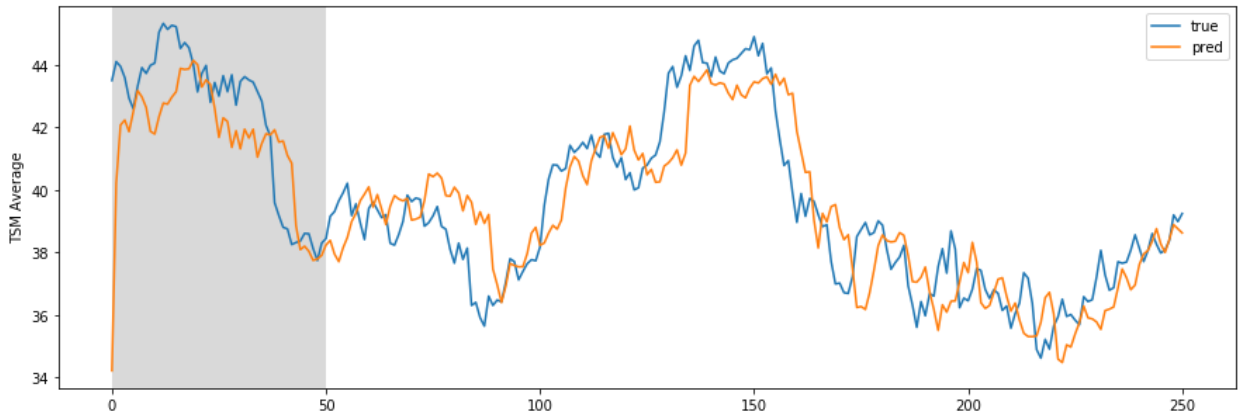


Figure 5-3.b 5-day prediction of stock price with all stock indices input with GRU model and ReLU activation output layer.

After the presentation, we added 4 more indices (PSI, USD, SMH, SOXL) and removed QQQ since it had a negative effect in our previous experiment. The graph below demonstrates the EVS value for each individual stock index. TSM and SMH produce the highest EVS result, so we will be using these two indices for our final optimization using hyperparameter.

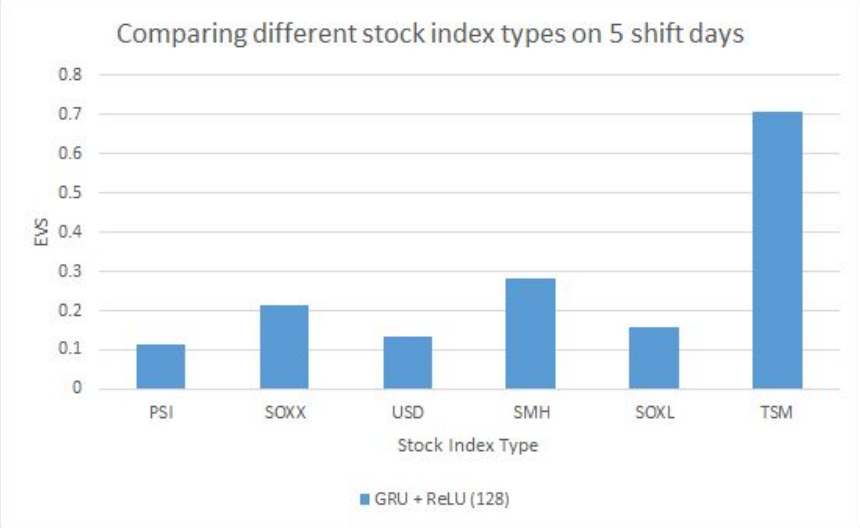
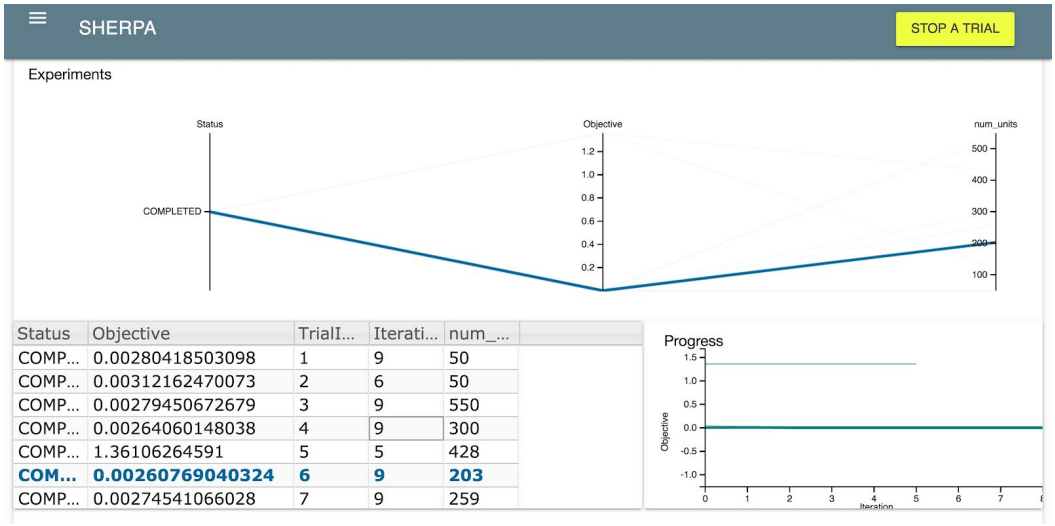


Figure 5-3.c Comparison of different stock index types on 5 shift days

5.4 Result with Hyperparameter Optimization

As shown in the picture (Figure 5-4.a) below, the highlighted row in the table has the best result: it has the lowest objective (minimize loss), and it optimizes the number of hidden units to 203. It produces the best EVS performance value of 0.8 (Figure 5-4.b). Our best EVS value through manual tuning was 0.72. Therefore, we have a 11% improvement of our EVS value of our test data after using the hyperparameter.



Status	Objective	TrialI...	Iterati...	num_...
COMP...	0.00280418503098	1	9	50
COMP...	0.00312162470073	2	6	50
COMP...	0.00279450672679	3	9	550
COMP...	0.00264060148038	4	9	300
COMP...	1.36106264591	5	5	428
COM...	**0.00260769040324**	**6**	**9**	**203**
COMP...	0.00274541066028	7	9	259

Figure 5-4.a Result of hyperparameter tuning. Trial 4th, with 203 hidden units, has the least amount of loss comparing to other trials. Therefore, 203 hidden units is the most optimal solution..

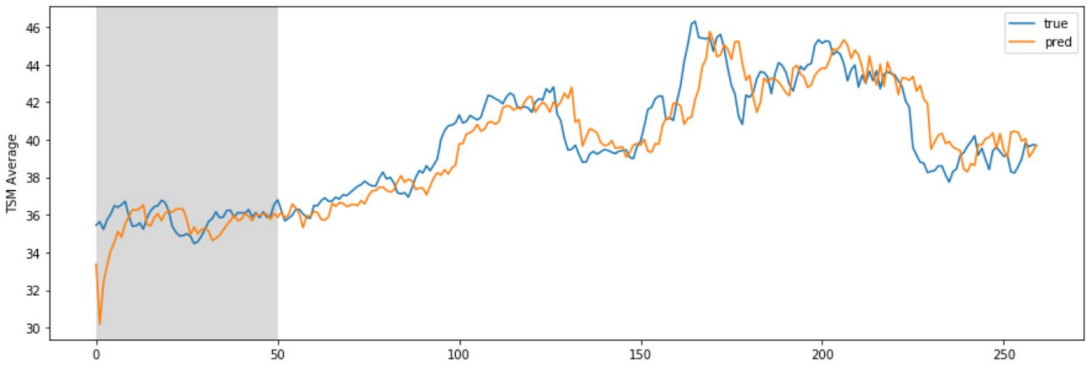


Figure 5-4.b 5-day prediction of stock price with all stock indices input with GRU model and ReLU activation output layer trained by hyperparameter.

6 Conclusion

With the optimization, we are able to improve the performance of stock price prediction on future 5 days from initial EVS performance value of 0.2 with LSTM and sigmoid layers to astonishingly high EVS performance value of 0.72 with GRU and ReLU layers. This has been further improved up to 0.8 EVS performance value with utilizing the hyperparameter library provided by Professor Baldi. We have concluded that, for our data processing and model, GRU layer with 203 units followed by ReLU function output layer made a stunning performance on stock price prediction.

Reference

- Understanding LSTM Networks
  http://colah.github.io/posts/2015-08-Understanding-LSTMs/
- How LSTM networks solve the problem of vanishing gradients
  https://medium.com/datadriveninvestor/how-do-lstm-networks-solve-the-problem-of-vanishing-gradients-a6784971a577
- Understanding GRU Networks
  https://towardsdatascience.com/understanding-gru-networks-2ef37df6c9be
- How Recurrent Neural Networks Work
  https://towardsdatascience.com/learn-how-recurrent-neural-networks-work-84e975feaaf7
- Illustrated Guide to LSTM's and GRU's: A step by step explanation
  https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21
- Tips for Training Recurrent Neural Networks
  https://danijar.com/tips-for-training-recurrent-neural-networks/
- Time-Series Prediction (Hvass-Labs)
  https://github.com/Hvass-Labs/TensorFlow-Tutorials/blob/master/23_Time-Series-Prediction.ipynb
- SHERPA: A Python Hyperparameter Optimization Library
  https://github.com/sherpa-ai/sherpa