RNN-LSTM LEARNING AND NEWS SENTIMENT ON SP500 STOCK MARKET PREDICTION

Shan Zhong Xiran Wang Yang He

2020-05-04

Abstract

Machine learning has a great potential in algorithmic trading. We propose two approaches by using reinforcement learning and recurrent networks on traditional time series stock price data, and the proposed procedure combines news processed by text summarization and sentiment analysis with knowledge graph to predict stock prices in near future. By converting text news to sentiment score, we reduce the computation time needed. In the work, not only do we demonstrate the prediction result but comparisons and discussions on two ptoposed models.

I. Introduction

The Long Short-Term Memory (LSTM) models reference is an extremely powerful model for predicting time-series type of data. Compared with standard and traditional moving average models, it can predict an arbitrary number of steps into the future, which most moving average models can't do. Moreover, it is developed from the recurrent neural network (RNN), in fact, RNNs use previous state of the hidden neurons to learn the current state given the new input, while LSTM has an extra cell added to help RNN better memorize the long-term context. In the stock prediction problems, it helps store past short-term and long-term prices information, it is significantly essential since the previous price of the stock is crucial in predicting its future price.

In this work, we present the application of LSTM on S&P500 stock market prediction. Here are some significant contributions of this work:

1. We created highly flexible stacked LSTM algorithm capturing dynamic stock data every minute from API and Wikipedia and making stock price prediction.
2. Tuned the 2-stacked LSTM model with optimal hyper-parameters among 20+ candidate models with TensorFlow.
3. Scraped data from online news to help make predictions.
4. Our alogirthm can track all S&P500 comapnies at the same time

Also, four main sections are contained later in this paper. Firstly, we introduce the methodology and training steps behind our optimal 2-stacked LSTM model. Then, we demonstrate our experimental results from multiple types of RNN-LSTM candidate models. Moreover, we will also present the error analysis based on them. a. Candidate models b. Error analysis 3. We have some discussions about our works.

II. Data

Quantitative Dataset:

The time series daily stock price data comes from a python API package which can track stock price to a minute interval. The input dataset contains the 4976 days' prices information in S&P500 stock market and currency exchange rate. More specifically, daily closed index of S&P500, daily close price of 505 companies in S&P500 stock market, daily closed price of Bitcoin and daily closed exchange rate of Chinese Yuan to US Dollars. The timeline is from 2000-01-03 to 2019-10-10. We store this data into the "josn" file. Lastly, by using the proposed model, we predict S&P500 index of a specific day.

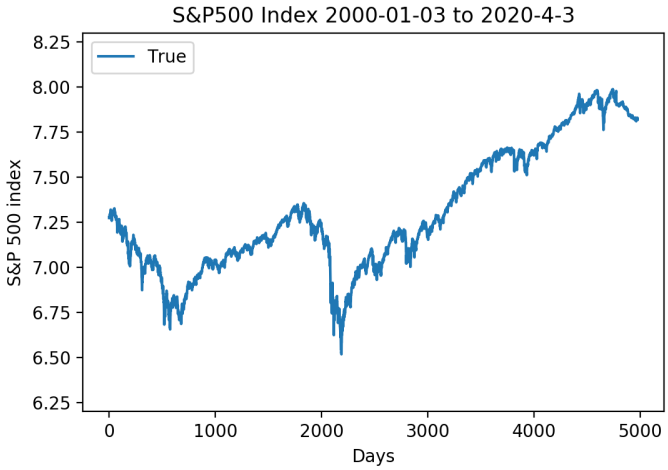


Figure 1: S&P 500 Over all

Sentiment Dataset

The sentiment news data comes from another python API which track political and financial news on a 15 minutes interval. We used the summarization of raw text. The data contains 3 million news from year 2015 to 2020. [2]. The sentiment dataset is mainly about political news. These news coverage from Associated Press,BBC, Washington Post, New York Times ,Google News, etc. this API filtered non-political news. The news is then classified with different actors and actions in the news. Such as Police Forces, Government, Military, and etc. Action such as appeal. After that, a sentiment score is calculated as tonal assessment of an article based on positive or negative.

Data Processing

We scale the data by using natural log transformation, for this sequence data, we chose 120 days as our prediction window. In a word, prices in a 120 days window are used to predict the S&P500 index on the 121st day. In addition, our numerical dataset does no have missing values, we pad sentiment dataset toa fix amount per day. Saturdays and Sundays are removed from our sequence. This setting is flexible and can be changed easily. Both sequence dataset and sentiment dataset are transformed into tenosr inputs. We developed a pipline that can automatically update our dataset. Then we jump over the data cleaning step.

```
In [40]: sentiment_data['20200223'][0:5]
Out[40]:
[{'SQLDATE': '20200223',
  'Actor1': 'GENEVA',
  'Actor1 location': 'Geneva Gen Switzerland',
  'Actor2': 'DINESH GUNAWARDENE',
  'Actor2 location': 'Geneva Gen Switzerland',
  'Action': ' Host a visit',
  'Action Code': '04',
  'Action subsection Code': '043',
  'Sentiment Score': -1.8575851393188798,
  'Original url': 'http://www.dailynews.lk/2020/02/25/local/212496/sri-lanka-ready-face-any-chal
geneva'},
 {'SQLDATE': '20200223',
  'Actor1': 'CHRISTIAN CHURCH',
  'Actor1 location': 'Amsterdam NoordHolland Netherlands',
  'Actor2': 'AMSTERDAM',
  'Actor2 location': 'Amsterdam NoordHolland Netherlands',
  'Action': ' Consult, not specified below',
  'Action Code': '04',
  'Action subsection Code': '040',
  'Sentiment Score': 3.07692307692308,
  'Original url': 'http://www.coptsunited.com/Details.php?I=2603&A=40870'},
```

Figure 2: Example of processed sentiment data

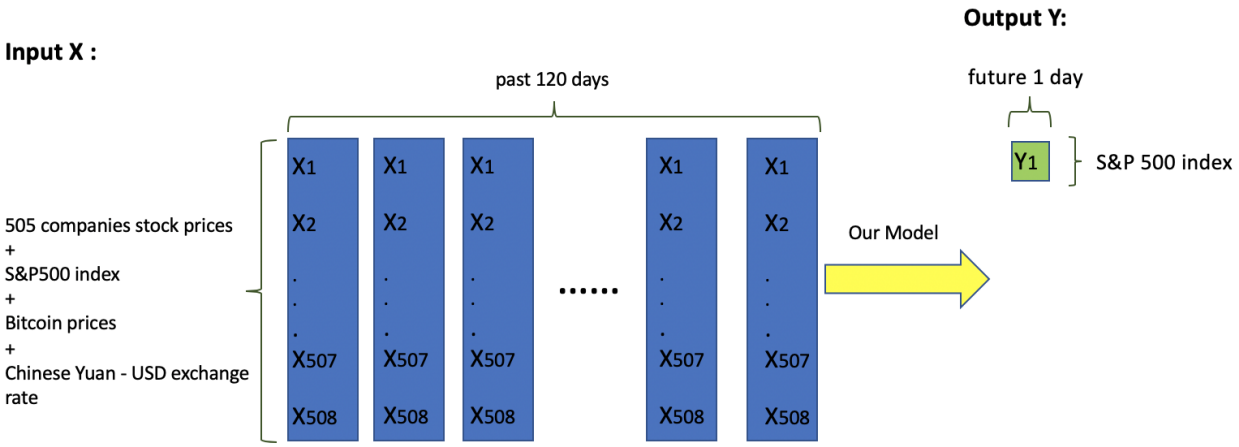


Figure 3: Quantitative Data processing

III. Methodology and Architecture

3.1 Training Step

Before we moved to sentiment dataset, firstly, we want to setup a baseline model which only includes the numeric dataset. Then, We incroporate the text dataset in the later work, as the sentiment prediction model is built upon our baseline model. A RTX 2080 Super is used during the training process. The training time for each baseline model is within 5 minutes, which we considered fast, and time affordable.

After the dataset is transformed into tensors, it is then divided into training and testing sets to evaluate. The split rate is 0.8, 80 % data used to train the LSTM model and 20 % used to validate the LSTM model. In this work, we mainly focus on 5 hyperparameters of the LSTM model, such as epoch, batch sizes, dropout rate, LSTM- units, and number of stacks.

Candidate Models

We have 27 LSTM candidate models with different values of hyperparameters. We tested the combinations of the batch size, LSTM units, dropout rate and LSTM stack:

Epoch, the number of times that we useded all data in training: from 1 to150

Batch size, the number of datasets we feed into the algorithm to update weight each time: [64,128,256]

LSTM units, the number of nodes in the lstm layer: [80,120,160]

Dropout rate, the percentage of nodes be randomly setted to 0 weight during training: [0,0.25,0.5]

The number of stack,the number of LSTM layers are stacked together: [1,2,3]

3.2 Error Analysis for Training and Testing sets:

For each hyper-parameters, we recorded the model performance. In our work, we used the mean square error (MSE) to evaluate and compare models. We recorded the MSE after each number of epoch run, then plot the MSE with respect to number of epochs run under each hyper-parameters settings. Based on the error analysis, which comes from training vs. validating loss result after each epoch, we judged the performance of our models under different hyper-parameters settings. Then we choosed the model with the hyper-parameters in the experiment as our baseline model.

Expriement 1: Examine Batch Size Effect

First, we exam the best batch size values during the training process. There are a lot of paper talking about how to choose an optimal batch size. A empirical rule is to start out with 32, then double it or cut it in half. We fixed other hyer-parameters to LSTM units120, Dropout rate 0 % , and a stack of two lstm layers. Then compare the effects of just batch size. From the graph on next page, we can see, when batch size is 256, the train and validation losses are both overall at a very low level. Also, the volatility of validation loss is smaller when batch size equal 256. Thus, batch size=256 has the best performance among all.

Expriement 2: Examine Drop-out Effect

Then, we tried dropout rate 0.5 and 0.25 both for different batch size 64,128, and 256 to train the data. With dropout added, we can see the validation loss has a significant decrease for all situations (batch size = 68,128, and 256). Consistently, batch size = 256 still has a lowest loss when dropout rate is 0.5 or 0.25. However, when batch size = 256 and dropout rate = 0.5, according to the plot, it shows an hight overall loss, and volitility. Therefore, we concluded that LSTM with dropout rate 0.25 batch size 256 may be a good fit for our data. Notice here, because of dropout effects, the training loss we recorded are sometimes larger than validation loss. As when training, a certain percentage of nodes are set with weight 0.

Expriement 3: Examine LSTM units Effect

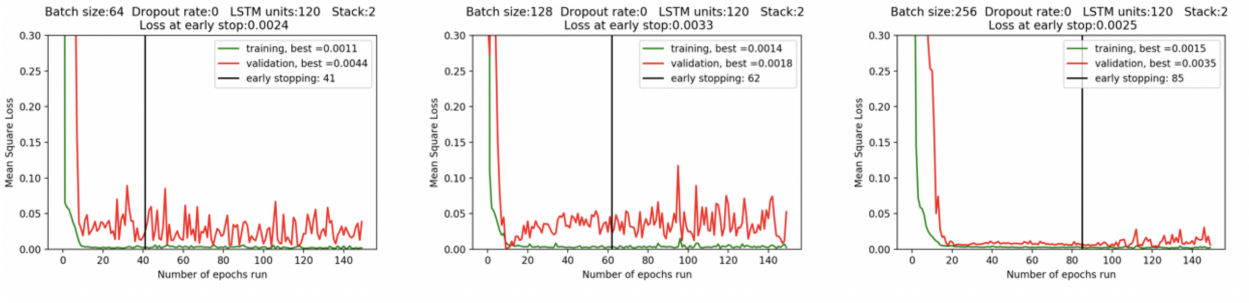


Figure 4: Batch size effect on traning/validating loss

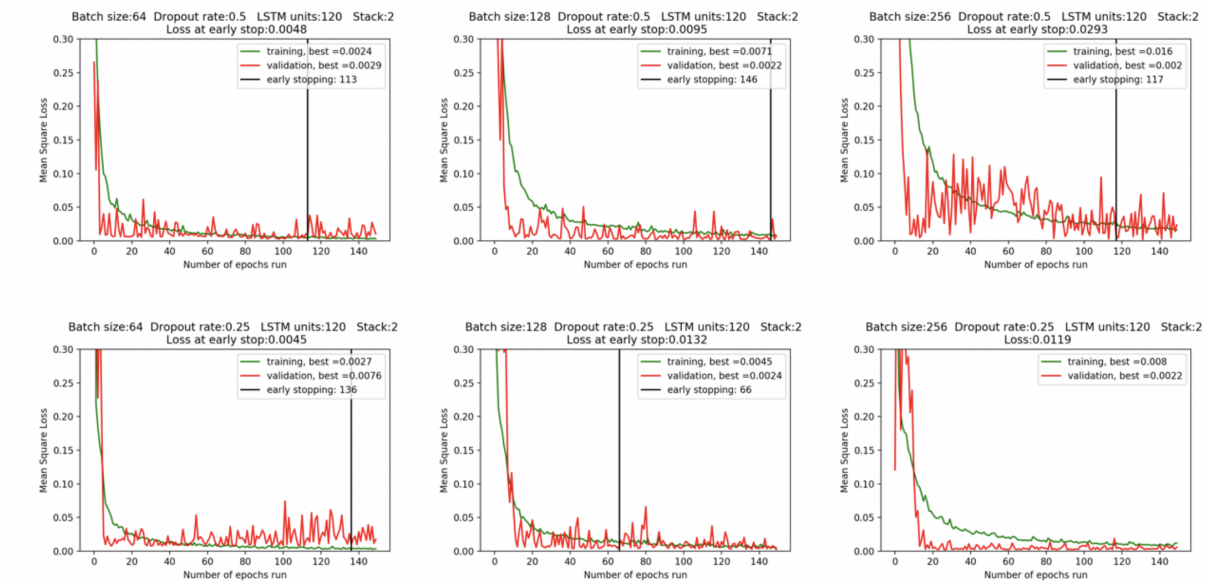


Figure 5: Dropout rate 0.5 or 0.25 effect on traning/validating loss

In our experiment, we also examed the LSTM units. To be notice, initially follow we chose 120 as our LSTM units, while at the same time, some papers suggest that $\frac{2}{3(N_i+N_0)}$ could be a basic reference for choosing LSTM hidden nodes, where N_i is the number of input neurons, and N_0 is the number of output neurons. So in our case, $\frac{2}{3(N_i+N_0)} = 80$ unites. So we also train the LSTM model with units 80, In addtion, for exploring purpose, we tried 160 too. It turns out that LSTM units = 120 is still the best choice, since both LSTM=80 and 160 had a higher training and validating loss.

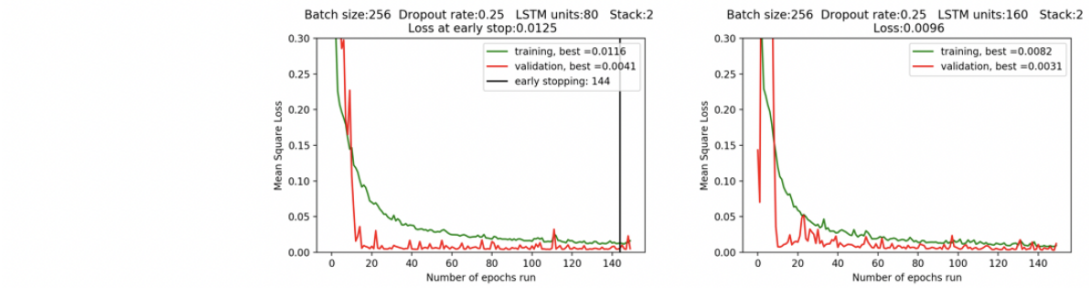


Figure 6: LSTM units effect on traning/validating loss

Expriement 4: Examine Stack Effect

As Karsten Eckhardt suggestions in his article, choosing the right Hyperparameters for a simple LSTM using Keras, generally, 2 layers have shown to be enough to detect more complex features. More layers can be better but also harder to train. As a general rule of thumb - 1 hidden layer work with simple problems, like this, and two are enough to find reasonably complex features." In our experiment, we just have performed the analysis with LSTM stack = 2, and in this section of experiment we, also exam when LSTM Stack is 1 and 3, and plots show that stack = 2 is the best choice for our model, and it provides the best loss and training and validating steps.

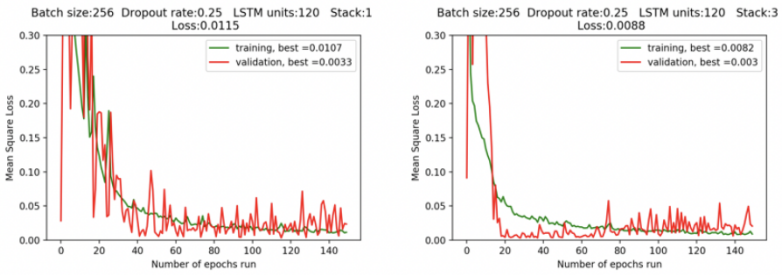


Figure 7: Stack effect on traning/validating loss

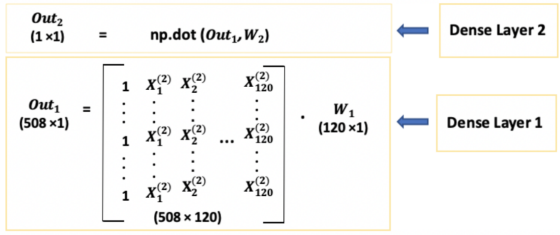
3.3 Final Quantitative Model:

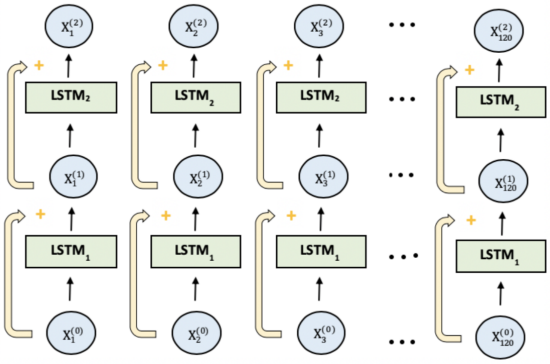
Here, we conclude our final and optimal model based on the analysis of candidate model performance comparisons. The following three figures show the statistics and architecture of the final model. The final model is a 2-stacked LSTM model with Lstm unit 120, drop out rate 0.25, past window size = 120 and future window size 1. Here, past window size is the number of days that model uses to predict and future window size is the number of days the model forecast. In Addition, the final model is trained by using batch size 256 and epoch 150.

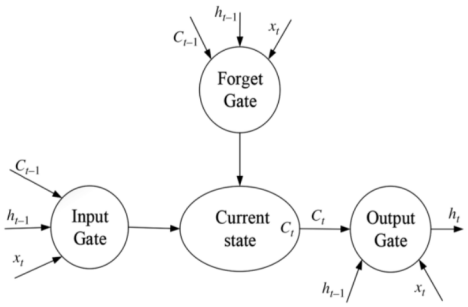
Layer (type)	Output Shape	Param #
stock_lstm_input (InputLayer	[(None, 508, 120)]	0
lstm_1 (LSTM)	(None, 508, 120)	115680
dropout_1 (Dropout)	(None, 508, 120)	0
lstm_2 (LSTM)	(None, 508, 120)	115680
dropout_2 (Dropout)	(None, 508, 120)	0
dense_1 (Dense)	(None, 508, 1)	121
flatten_1 (Flatten)	(None, 508)	0
dense_64 (Dense)	(None, 1)	509
reshape_32 (Reshape)	(None, 1, 1)	0

Total params: 231,990
Trainable params: 231,990
Non-trainable params: 0

```
def make_model(lstm_units=120,dropout_rate=0.25,past_window_size=120,future_window_size=1):

    sequence_input = tf.keras.Input(shape=(len(available_tickers),past_window_size),
                                                    dtype=tf.float32,name='stock_lstm_input')

    lstm_1 = LSTM(units=lstm_units, return_sequences=True ,name = "lstm_1")(sequence_input)
    dropout_1 = Dropout(dropout_rate,name = "dropout_1")(lstm_1)

    lstm_2 = LSTM(units=lstm_units, return_sequences=True,name = "lstm_2")(dropout_1)
    dropout_2 = Dropout(dropout_rate,name = "dropout_2")(lstm_2)

    dense_1 = Dense(future_window_size,name = "dense_1")(dropout_2)
    flatten_1 = Flatten(name = "flatten_1")(dense_1)

    dense_2 = Dense(future_window_size * len(interested_y))(flatten_1)
    result = Reshape( (len(interested_y),future_window_size) )(dense_2)

    model = tf.keras.Model(inputs = sequence_input,
                           outputs = result,
                           name='model1')

    model.compile(loss='mean_squared_error', optimizer='adam')
    return model

model.fit(X_train, y_train, epochs=150, batch_size=256)
```





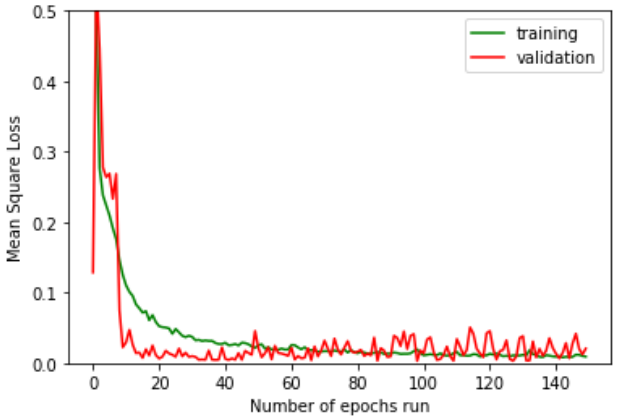


3.4 Final Text Model:

We tried using US-CHINA news, which is actors in ['US', CHINA'] to help prediction, the model structures are shown below, as well as the validation loss comparison, it seems that sentiment news does not improve loss for S&P 500. That might because of S&P500 are a lot of companies, only news of US-CHINA relation can not accuarily capture the change in the stock market. For single stock, we tried GOOGLE, and found it does improve the validation loss.

Layer (type)	Output Shape	Param #	Connected to
stock_lstm_input (InputLayer)	[(None, 508, 120)]	0	
lstm (LSTM)	(None, 508, 120)	115680	stock_lstm_input[0][0]
dropout (Dropout)	(None, 508, 120)	0	lstm[0][0]
lstm_1 (LSTM)	(None, 508, 120)	115680	dropout[0][0]
dropout_1 (Dropout)	(None, 508, 120)	0	lstm_1[0][0]
text_sentiment_score (InputLaye	[(None, 5)]	0	
text_sentiment_action (InputLay	[(None, 5, 20)]	0	
dense (Dense)	(None, 508, 1)	121	dropout_1[0][0]
dense_1 (Dense)	(None, 1)	6	text_sentiment_score[0][0]
dense_2 (Dense)	(None, 5, 5)	105	text_sentiment_action[0][0]
flatten (Flatten)	(None, 508)	0	dense[0][0]
flatten_1 (Flatten)	(None, 1)	0	dense_1[0][0]
flatten_2 (Flatten)	(None, 25)	0	dense_2[0][0]
concatenate (Concatenate)	(None, 534)	0	flatten[0][0] flatten_1[0][0] flatten_2[0][0]
dense_3 (Dense)	(None, 1)	535	concatenate[0][0]
reshape (Reshape)	(None, 1, 1)	0	dense_3[0][0]

Total params: 232,127
Trainable params: 232,127
Non-trainable params: 0



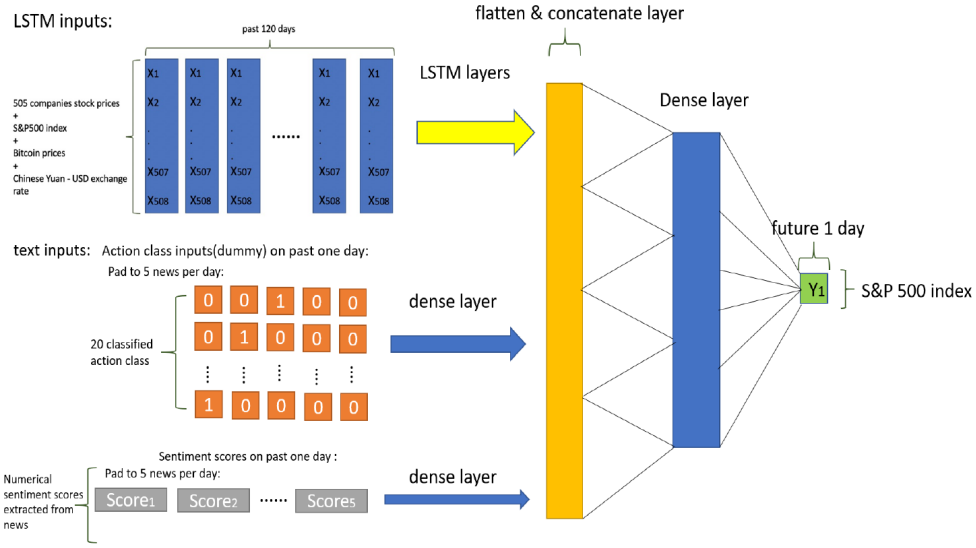


Figure 8: LSTM text data processing flow

IV. Prediction

From Quantitative Model:

In this section, we provide our prediction results for both training and testing dataset. First plot shows the prediction result when fit training is set into the model, while the second plot shows model prediction of the testing set. Our RMSE from the testing data set is 0.0024.

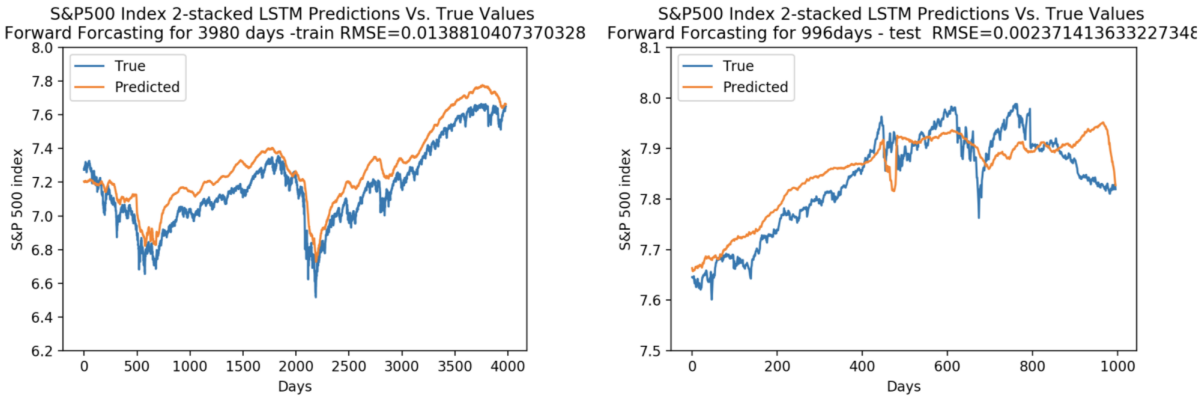


Figure 9: LSTM predictions without text

From Text Model:

Affter added the text into the LSTM model, the traning predictions get improved to 0.0113 in RMSE, however, it seems it over predicted the S&P500 index when using testing dataset. The prediction RMSE for testing dataset is 0.0207.

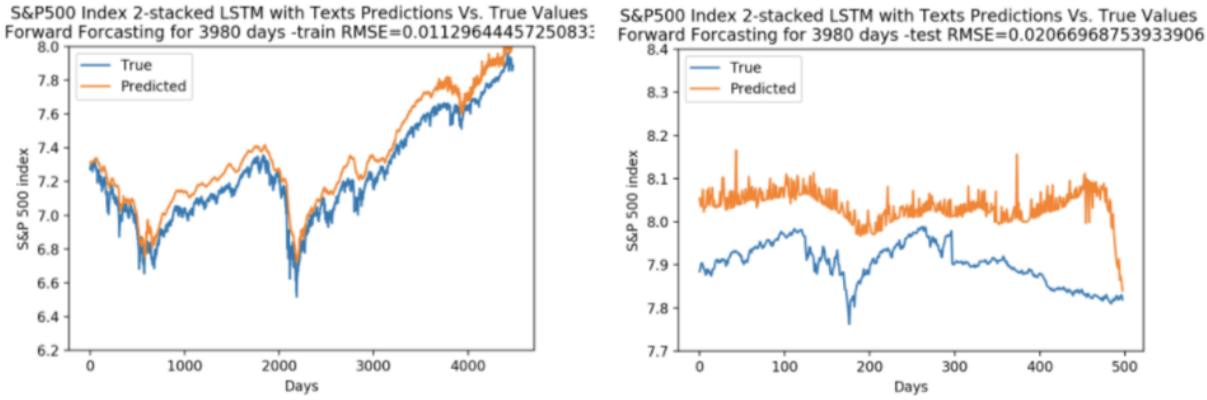
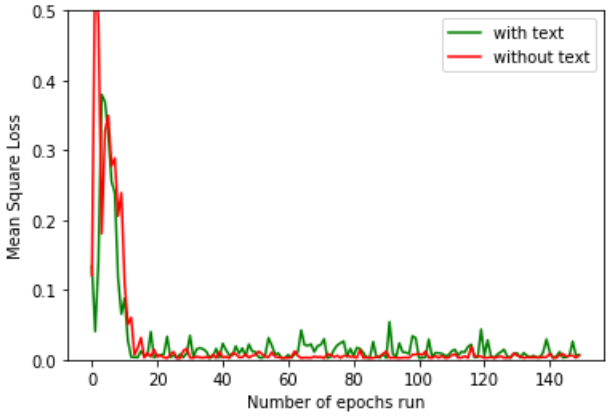


Figure 10: LSTM predictions with text

V. Discussion:

In conclusion, our final models have a good ability to predict S&P 500 index. Especially, the LSTM model with only numerical data shows a great power to learn the index future trend. Moreover, we also tried to add text input into this LSTM model with some adjustment on the orginal LSTM architecture. Bease on prediction comparsions and comparsions on loss on training steps, we don't think the extra text input can provide a significant improvment comparing only numerical data is used in our optimal LSTM.



However, we do think there are some room to improve the performance in th efuture studyies. For example, we could try a larger training data set to enhance the learning accuracy and ability of the model, and we can develop a more effeicient way to score the news information, such as, we could just grap financial news' title instead of scaning the whole news content.

References

[1] Alpha Vantage In https://github.com/RomelTorres/alpha_vantage

[2] gdeltPyR In https://github.com/linwoodc3/gdeltPyR

[3] Le, Xuan-Hien; Ho, Hung Viet Application of Long Short-Term Memory (LSTM) Neural Network for Flood Forecasting Water 2019, 11(7), 1387;, 2019.

[4] Chen, K., Zhou, Y., Dai, F. A lstm-based method for stock returns prediction: A case study of china stock market. InIEEE International Conference on Big Data (Big Data). pp. 2823–2824, 2015

[5] Abhishek Nan, Anandh Perumal, Osmar R. Zaiane Sentiment and Knowledge Based Algorithmic Trading with Deep Reinforcement Learning InarXiv:2001.09403v1, 2020