Korea No.1 University

KYUNGPOOK NATIONAL UNIVERSITY

Data Science

CHO YOONSANG

Stock Price Prediction Using Deep-Learning with RNN and LSTM

Cho YoonSang
Dept of Computer Science and Engineering, KyungPook National University

Abstract—The art of forecasting stock prices has been a difficult task for many of the researchers and analysts. In fact, investors are highly interested in the research area of stock price prediction. For a good and successful investment, many investors are keen on knowing the future situation of the stock market. Good and effective prediction systems for stock market help traders, investors, and analyst by providing supportive information like the future direction of the stock market. In this work, we present a recurrent neural network (RNN) and Long Short-Term Memory (LSTM) approach to predict stock market indices.

Index Terms—Deep Learning, Machine Learning, RNN, LSTM, TensorFlow

1 Introduction

There are a lot of complicated financial indicators and also the fluctuation of the stock market is highly violent. However, as the technology is getting advanced, the opportunity to gain a steady fortune from the stock market is increased and it also helps experts to find out the most informative indicators to make a better prediction.

The prediction of the market value is of great importance to help in maximizing the profit of stock option purchase while keeping the risk low. Recurrent neural networks (RNN) have proved one of the most powerful models for processing sequential data. Long Short-Term memory is one of the most successful RNNs architectures. LSTM introduces the memory cell, a unit of computation that replaces traditional artificial neurons in the hidden layer of the network. With these memory cells, networks are able to effectively associate memories and input remote in time, hence suit to grasp the structure of data dynamically over time with high prediction capacity.

2 LSTM Architeture

2.1 RNN and LSTM

Long Short Term Memory networks – usually just called "LSTMs" – are a special kind of RNN, capable of learning long-term dependencies.

LSTMs are explicitly designed to avoid the long-term dependency problem. Remembering information for long periods of time is practically their default behavior, not something they struggle to learn!

All recurrent neural networks have the form of a chain of repeating modules of neural network. In standard RNNs, this repeating module will have a very simple structure, such as a single tanh layer.

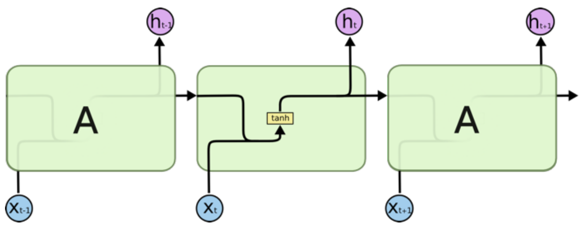


Figure 1. *The repeating module in a standard RNN contains a single layer.*

LSTMs also have this chain like structure, but the repeating module has a different structure. Instead of having a single neural network layer, there are four, interacting in a very special way.

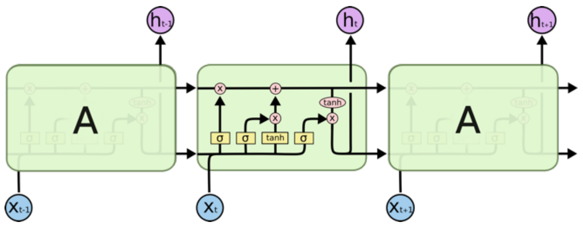


Figure 1. *The repeating module in an LSTM contains four interacting layers.*

Don’ t worry about the details of what’ s going on. We’ ll walk through the LSTM diagram step by step later. For now, let’ s just try to get comfortable with the notation we’ ll be using.

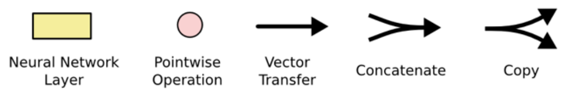


Figure 3. *The meaning of each Symbol*

In the above diagram, each line carries an entire vector, from the output of one node to the inputs of others. The pink circles represent pointwise operations, like vector addition, while the yellow boxes are learned neural network layers. Lines merging denote concatenation, while a line forking denote its content being copied and the copies going to different locations.

2.2 The Core Idea Behind LSTMs

The key to LSTMs is the cell state, the horizontal line running through the top of the diagram.

The cell state is kind of like a conveyor belt. It runs straight down the entire chain, with only some minor linear interactions. It’ s very easy for information to just flow along it unchanged.

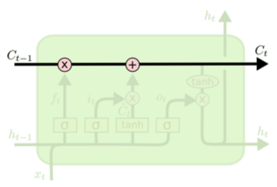


Figure 4. *The LSTM Diagram*

The LSTM does have the ability to remove or add information to the cell state, carefully regulated by structures called gates.

Gates are a way to optionally let information through. They are composed out of a sigmoid neural net layer and a pointwise multiplication operation.

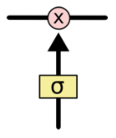


Figure 5. *The Gates.*

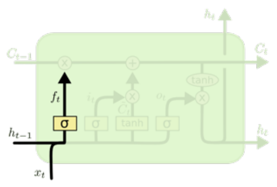
The sigmoid layer outputs numbers between zero and one, describing how much of each component should be let through. A value of zero means “let nothing through,” while a value of one means “let everything through!”

An LSTM has three of these gates, to protect and control the cell state.

2.3 Step – by -Step LSTM Walk Through

The first step in our LSTM is to decide what information we’ re going to throw away from the cell state. This decision is made by a sigmoid layer called the “forget gate layer.” It looks at ht−1ht−1 and xtxt, and outputs a number between 00 and 11 for each number in the cell state Ct−1Ct−1. A 11 represents “completely keep this” while a 00 represents “completely get rid of this.”

Let’ s go back to our example of a language model trying to predict the next word based on all the previous ones. In such a problem, the cell state might include the gender of the present subject, so that the correct pronouns can be used. When we see a new subject, we want to forget the gender of the old subject.

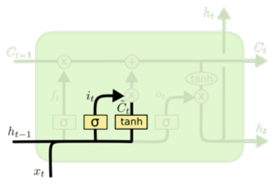


$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \ + \ b_f\right)$$

Figure 6. *The first step – Forget Gate*

The next step is to decide what new information we’ re going to store in the cell state. This has two parts. First, a sigmoid layer called the “input gate layer” decides which values we’ ll update. Next, a tanh layer creates a vector of new candidate values, C˜ tC~t, that could be added to the state. In the next step, we' ll combine these two to create an update to the state.

In the example of our language model, we' d want to add the gender of the new subject to the cell state, to replace the old one we' re forgetting.

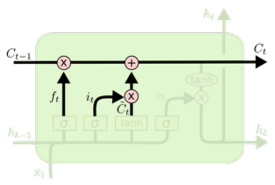


$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \;+\; b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \;+\; b_C)$$

Figure 7. *The Second step – Update Gate.*

It' s now time to update the old cell state, Ct−1Ct−1, into the new cell state CtCt. The previous steps already decided what to do, we just need to actually do it.
We multiply the old state by ftft, forgetting the things we decided to forget earlier. Then we add it∗C˜ tit∗C~t. This is the new candidate values, scaled by how much we decided to update each state value.

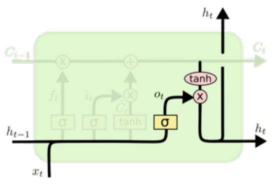
In the case of the language model, this is where we' d actually drop the information about the old subject' s gender and add the new information, as we decided in the previous steps.



$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Figure 8. *Updating the cell*

Finally, we need to decide what we' re going to output. This output will be based on our cell state, but will be a filtered version. First, we run a sigmoid layer which decides what parts of the cell state we' re going to output. Then, we put the cell state through tanhtanh (to push the values to be between −1−1 and 11) and multiply it by the output of the sigmoid gate, so that we only output the parts we decided to.
For the language model example, since it just saw a subject, it might want to output information relevant to a verb, in case that' s what is coming next. For example, it might output whether the subject is singular or plural, so that we know what form a verb should be conjugated into if that' s what follows next.



$$o_t = \sigma\left(W_o \; [h_{t-1}, x_t] \;+\; b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

Figure 8. *The Last step – Output Gate*

3 Proposed FrameWork

3.1 Deep Learning

Deep learning has revolutionized artificial intelligence by helping us build machines and systems that were only dreamt of in the past. In true essence, Deep Learning is a sub-sect of Machine Learning that uses *deep* artificial neural networks

A Deep Neural Network is just a Neural Network with many layers stacked on top of each other – greater the number of layers, deeper the network. The growing need for Deep Learning, and, consequently, training of Deep Neural Networks gave rise to a number of libraries and frameworks dedicated to Deep Learning.

3.2 What is TensorFlow?

TensorFlow is an open-sourced library that' s available on GitHub. It is one of the more famous libraries when it comes to dealing with Deep Neural Networks. The primary reason behind the popularity of TensorFlow is the sheer ease of building and deploying applications using TensorFlow. The sample projects provided in the GitHub repository are not only powerful but also written in a beginner-friendly way.

TensorFlow excels at numerical computing, which is critical for deep learning. It provides APIs in most major languages and environments needed for deep learning projects: Python, C, C++, Rust, Haskell, Go, Java, Android, IoS, Mac OS, Windows, Linux, and Raspberry Pi.

Moreover, TensorFlow was created keeping the processing power limitations in mind. Implying, we can run this library on all kinds of computers, irrespective of their processing powers. It can even be run on a smartphone

Some of the major applications of TensorFlow are:

- Tensorflow has been successfully implemented in DeepDream – the automated image captioning software – uses TensorFlow.
- Google' s RankBrain, backed by TensorFlow, handles a substantial number of queries every minute and has effectively replaced the traditional static algorithm-based search.
- If you' ve used the Allo application, you must' ve seen a feature similar to Google' s Inbox – you can reply to the last message from a few customized options. All thanks to Machine Learning with TensorFlow. Another feature analyses the images sent to you in order to suggest a relevant response.

3.3 What is Keras?

Keras is a high-level library that' s built on top of Theano or TensorFlow. It provides a scikit-learn type API (written in Python) for building Neural Networks. Developers can use Keras to quickly build neural networks without worrying about the mathematical aspects of tensor algebra, numerical techniques, and optimization methods.
The key idea behind the development of Keras is to facilitate experimentations by fast prototyping. The ability to go from an idea to result with the least possible delay is key to good research.

Salient Features of Keras

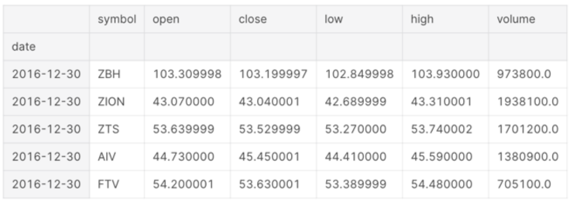
- Keras is a high-level interface and uses Theano or Tensorflow for its backend.
- It runs smoothly on both CPU and GPU.
- Keras supports almost all the models of a neural network – fully connected, convolutional, pooling, recurrent, embedding, etc. Furthermore, these models can be combined to build more complex models.
- Keras, being modular in nature, is incredibly expressive, flexible, and apt for innovative research.
- Keras is a completely Python-based framework, which makes it easy to debug and explore.

4 Methodology

4.1 Stage 1 : Load Data

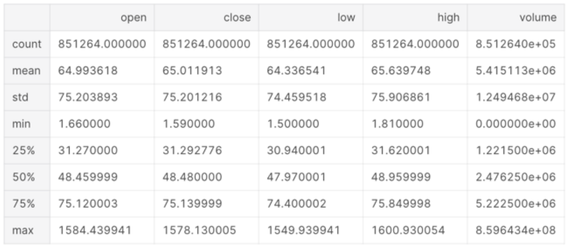
Load stock prices from Raw data, prices – split – adjused.csv and comfrim table.

TABLE 1
NY Stock prices (tail)



	symbol	open	close	low	high	volume
date						
2016-12-30	ZBH	103.309998	103.199997	102.849998	103.930000	973800.0
2016-12-30	ZION	43.070000	43.040001	42.689999	43.310001	1938100.0
2016-12-30	ZTS	53.639999	53.529999	53.270000	53.740002	1701200.0
2016-12-30	AIV	44.730000	45.450001	44.410000	45.590000	1380900.0
2016-12-30	FTV	54.200001	53.630001	53.389999	54.480000	705100.0

TABLE 2
NY Stock prices (describe)



	open	close	low	high	volume
count	851264.000000	851264.000000	851264.000000	851264.000000	8.512640e+05
mean	64.993618	65.011913	64.336541	65.639748	5.415113e+06
std	75.203893	75.201216	74.459518	75.906861	1.249468e+07
min	1.660000	1.590000	1.500000	1.810000	0.000000e+00
25%	31.270000	31.292776	30.940001	31.620001	1.221500e+06
50%	48.459999	48.480000	47.970001	48.959999	2.476250e+06
75%	75.120003	75.139999	74.400002	75.849998	5.222500e+06
max	1584.439941	1578.130005	1549.939941	1600.930054	8.596434e+08

4.2 Stage 2 : Analyze Data

Analyze the imported data and visualize the stock price and the stock volume respectively.

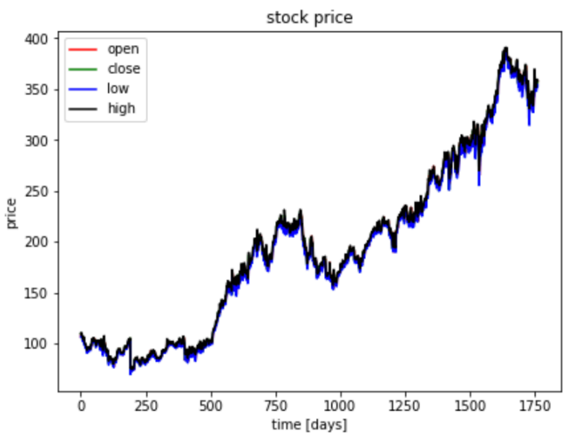


Figure 9. *Stock Price*

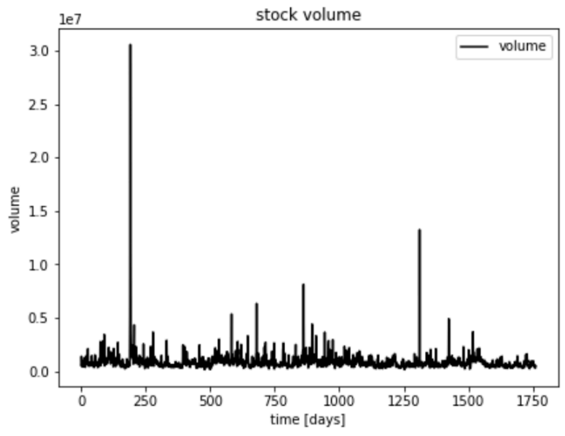


Figure 10. *Stock Volume*

4.3 Stage 3 : ManiPulate Data

only the features which are to be fed to the neural network are chosen. We will choose the feature from volume and special stock. And we also normalize stock data, Create train and validation and test data sets.

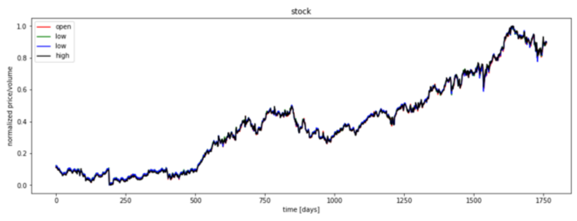


Figure 11. *Stock by feacture volume*

4.4 Stage 4 : Model and validate data

Training Neural Network: In this stage, the data is fed to the neural network and trained for prediction assigning random biases and weights. Our LSTM model is composed of a sequential input layer followed by 3 LSTM layers and dense layer with activation and then finally a dense output layer with linear activation function.

Optimizer

The type of optimizer used can greatly affect how fast the algorithm converges to the minimum value. Also, it is important that there is some notion of randomness to avoid getting stuck in a local minimum and not reach the global minimum. There are a few great algorithms, but I have chosen to use Adam optimizer. The Adam optimizer combines the perks of two other optimizers: ADAgrad and RMSprop.

The **ADAgrad optimizer** essentially uses a different learning rate for every parameter and every time step. The reasoning behind ADAgrad is that the parameters that are infrequent must have larger learning rates while parameters that are frequent must have smaller learning rates. In other words, the stochastic gradient descent update for ADAgrad becomes

$$\theta_{t+1,i} = \theta_{t,i} - \eta g_{t,i}$$

Where

$$g_{t,i} = \nabla_{\theta_t} J(\theta_{t,i})$$

The learning rate is calculated based on the past gradientsthat have been computed for each parameter. Hence,

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{G_t + \epsilon}} \cdot g_t$$

Where G is the matrix of sums of squares of the past gradients. The issue with this optimization is that the learning rates start vanishing very quickly as the iterations increase.

RMSprop considers fixing the diminishing learning rate by only using a certain number of previous gradients. The updates become

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} \cdot g_t$$

Where

$$E[g^2]_t = 0.9E[g^2]_{t-1} + 0.1g_t^2$$

Now that we understand how those two optimizers work, we can look into how Adam works.

Adaptive Moment Estimation, or Adam, is another method that computes the adaptive learning rates for each parameter by considering the exponentially decaying average of past squared gradients and the exponentially decaying average of past gradients. This can be represented as

$$v_t = \beta v_{t-1} + (1-\beta)g_t^2$$

$$m_t = \beta m_{t-1} + (1-\beta)g_t$$

The v and m can be considered as the estimates of the first and second moment of the gradients respectively, hence getting the name Adaptive Moment Estimation. When this was first used, researchers observed that there was an inherent bias towards 0 and they countered this by using the following estimates:

$$\hat{v}_t = \frac{v_t}{1-\beta_2^t}$$

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t}$$

This leads us to the final gradient update rule

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \cdot \hat{m}_t$$

This is the optimizer that I used, and the benefits are summarized into the following:

1. The learning rate is different for every parameter and every iteration.
2. The learning does not diminish as with the ADAgrad.
3. The gradient update uses the moments of the distribution of weights, allowing for a more statistically sound descent.

Regularization

Another important aspect of training the model is making sure the weights do not get too large and start focusing on one data point, hence overfitting. So we should always include a penalty for large weights (the definition of large would be depending on the type of regulariser used). I have chosen to use Tikhonov regularization, which can be thought of as the following minimization problem:

$$\underset{f\in\mathcal{H}}{\operatorname{argmin}} \frac{1}{n}\sum_{i=1}^{n} V(f(x_i), y_i) + \gamma ||f||_K^2$$

The fact that the function space is in a Reproducing Kernel Hilbert Space (RKHS) ensures that the notion of a norm exists. This allows us to encode the notion of the norm into our regularizer.

4.4 Stage 5 : Model and validate data

Estimate expected stock prices using RNNs with basic, LSTM, and GRU cells

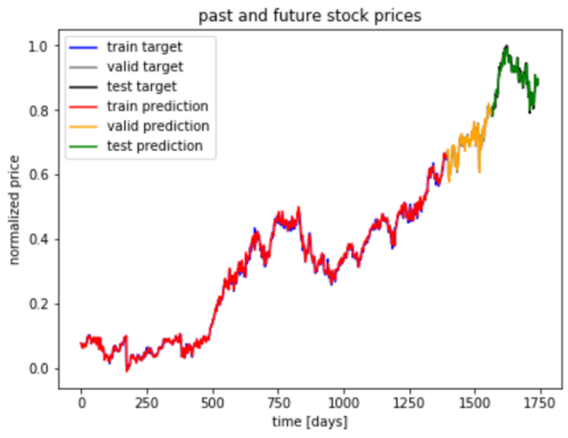


Figure 12. *Past and future stock prices*

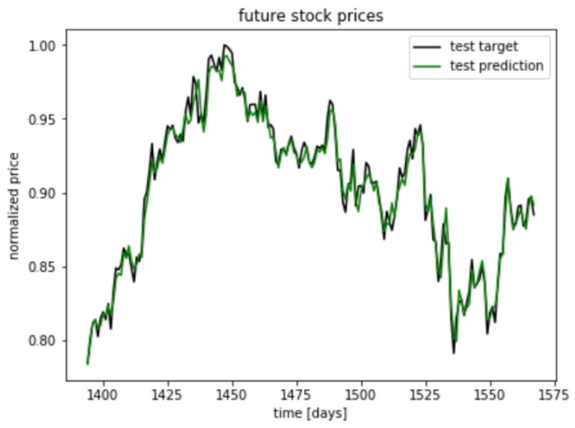


Figure 12. *future stock prices*

5 Conclusion

The popularity of stock market trading is growing rapidly, which is encouraging researchers to find out new methods for the prediction using new techniques. The forecasting technique is not only helping the researchers but it also helps investors and any person dealing with the stock market. In order to help predict the stock indices, a forecasting model with good accuracy is required.

In this work, we have used one of the most precise forecasting technology using Recurrent Neural Network and Long Short-Term Memory unit which helps investors, analysts or any person interested in investing in the stock market by providing them a good knowledge of the future situation of the stock market.

Written down as a set of equations, LSTMs look pretty intimidating. LSTMs were a big step in what we can accomplish with RNNs.

5 References

〔1〕 https://www.kaggle.com/pablocastilla/predict-stock-prices-with-lstm
〔2〕 https://www.facebook.com/groups/TensorFlowKR/permalink/447678885573175/
〔3〕 http://qiita.com/yuyakato/items/ab38064ca215e8750865

〔4〕 https://groups.google.com/forum/#!topic/keras-users/9GsDwkSdqBg
〔5〕 https://www.jakob-aungiers.com/articles/a/LSTM-Neural-Network-for-Time-Series-Prediction
〔6〕 https://github.com/jaungiers/LSTM-Neural-Network-for-Time-Series-Prediction
〔7〕 https://github.com/jaungiers/LSTM-Neural-Network-for-Time-Series-Prediction
〔8〕 https://machinelearningmastery.com/time-series-prediction-lstm-recurrent-neural-networks-python-keras/
〔9〕 https://mjgim.icim.or.kr/2017/08/02/LSTM.html
〔10〕 https://tykimos.github.io/2017/09/09/Time-series_Numerical_Input_Numerical_Prediction_Model_Recipe/
〔11〕 https://tykimos.github.io/
〔12〕 https://tykimos.github.io/2017/09/30/Book_Python_DeepLearning_Keras_with_Block/
〔13〕 https://datascienceschool.net/view-notebook/1d93b9dc6c624fbaa6af2ce9290e2479/
〔14〕 https://tykimos.github.io/lecture/
〔15〕 https://tbacking.com/2017/08/18/%ec%88%9c%ed%99%98-%ec%8b%a0%ea%b2%bd%eb%a7%9d-lstm-%ed%99%9c%ec%9a%a9-%ec%a3%bc%ea%b0%80-%ec%98%88%ec%b8%a1/
〔16〕 http://colah.github.io/posts/2015-08-Understanding-LSTMs/
〔17〕 https://www.youtube.com/watch?v=odMGK7pwTqY
〔18〕 https://www.kaggle.com/benjibb/lstm-stock-prediction-20170507/notebook