ATHENS UNIVERSITY OF ECONOMICS AND BUSINESS

SCHOOL OF INFORMATION SCIENCES & TECHNOLOGY

DEPARTMENT OF INFORMATICS

Deep Learning models for Machine Learning tasks:

Predicting Stock Price using LSTM

Savvas Drosos
Nikolaos Karagkounis

Source code:
https://github.com/savvasdr/Predict-Stock-Price-LSTM

Lecturer: Prof. Prodromos Malakasiotis

A project submitted in fulfilment

of the requirements for the "Deep Learning" course

Master of Science in Computer Science

AUEB, ATHENS, GREECE

FEBRUARY 2019

Contents

Chapter 1.

Introduction

1.1. Motivation

A very interesting problem would be to be able to accurately predict the price of a currency or share in the near future. Traders trying to do this, use a variety of financial indices, and make technical analyzes to invest their money accordingly.

The goal is to be able to predict the future price of a currency/share using Artificial Intelligence. More specifically, having the history of Euro and Dollar exchange rates, we will build an LSTM[1] Neural Network[2] which will predict the price of the next day.

Practically we have a timeseries and we want to predict the next step. Time series predictive problems are a difficult type of predictive modeling problem. Unlike regression predictive modeling, the time series also adds the complexity of a sequence dependence among the input variables.

A powerful type of neural network designed to handle sequence dependence is called recurrent neural networks. The Long Short-Term Memory network or LSTM network is a type of recurrent neural network used in deep learning because very large architectures can be successfully trained.[3]

[1] LSTM stands for long short-term memory. It is a model or architecture that extends the memory of recurrent neural networks. (*https://hub.packtpub.com/what-is-lstm/*)

[2] A neural network is a type of machine learning which models itself after the human brain. This creates an artificial neural network that via an algorithm allows the computer to learn by incorporating new data. (*https://www.techradar.com/news/what-is-a-neural-network*)

[3] Time Series Prediction with LSTM Recurrent Neural Networks in Python with Keras (*https://machinelearningmastery.com/time-series-prediction-lstm-recurrent-neural-networks-python-keras/*)

1.2. Dataset

Of course, we will need a dataset to be able to achieve our goal. Fortunately, it was easy to find the history of the EUR / USD exchange rates. The dataset we downloaded contained the data: Date, Price, Open, High, Low, Change%. We have also calculated[4] and added 11 indicators used for technical analysis.

Our dataset begins in 1980 and reaches the end of 2018. However, the Euro came into circulation in 1999. The data we have for previous years concern the Euro predecessor, the ECU[5]. The ECU has existed since 1979 and was mainly used between states and large companies. It did not have a physical existence, that is, it never existed as a banknote, and in 1999 it was renamed to Euro, where it became the official currency of several EU Member States. So the data we have of the EUR/USD exchange rate is for the last 38 years.

Other exchange rates or general economic indicators could also be used in the dataset, as some other currency may affect the Euro and Dollar prices. However, because the problem is by nature difficult, we preferred to stay only at that rate and its own indices, in order to be able to train our model very well at that rate.

[4] Investopedia.com provided all the information we needed. (*https://www.investopedia.com/articles/trading/11/indicators-and-strategies-explained.asp*)
[5] Stands for European Currency Unit (*https://en.wikipedia.org/wiki/European_Currency_Unit*)

Chapter 2.

Methodology

The source code is available on GitHub: https://github.com/savvasdr/Predict-Stock-Price-LSTM

2.1. Data-Preprocessing

Before we use our dataset on our network, we will first need to process it in the appropriate way. All of our features are numeric except dates. So, we initially convert the dates into corresponding numbers. The reason for this is because stock exchanges are open from Monday to Friday. Thus, e.g. on Monday there may be a larger volume of transactions and a more prone price change.

We also need to implement a feature scaling so that all of our data is on the same scale. Trying different techniques, we decided to do Normalization.

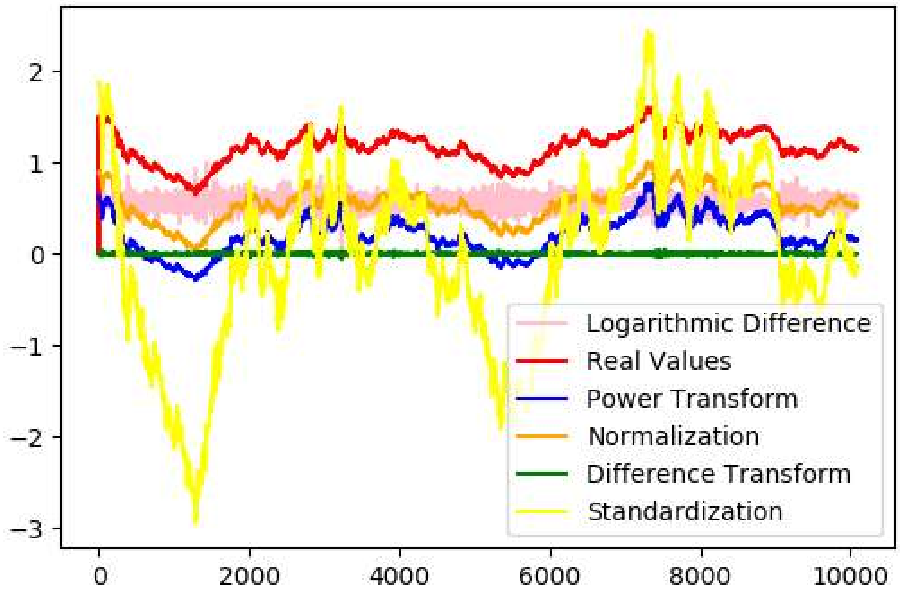

Image 1: Comparing different feature scaling techniques on the Price feature

2.2. LSTM Model

Our dataset is practically a huge timeseries. So, in order to be able to train our network, we will create smaller n-size timeseries. Each exchange rate refers to one day, so a timeseries step refers to one day. The input of the network will be the timeseries and the output will be the predicted value for the next day.

Suppose that the size of each timeseries is n = 3 days and t = 1 is the first day. The timeseries ts_1 will contain days t={1,2,3}. This is going to be the input of our neural network: $x_1 = ts_1^{t=\{1,2,3\}}$

Having this specific timeseries, our purpose is to predict the price of the next day. The output will be: $y_1 = price_4^{t=\{4\}}$. That is, the value precedes the timeseries by +1 step (1 day in our case).

Day	1	2	3	4
Price	101	102	103	104

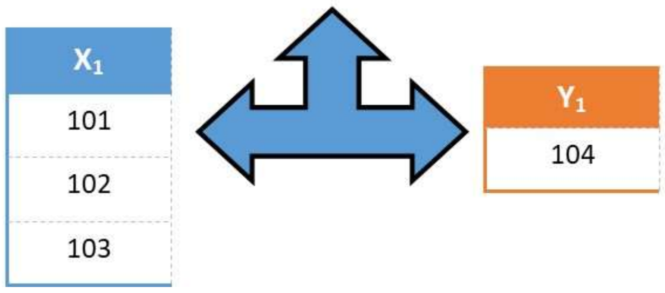

Image 2: A simple example of creating Timeseries₁

To create the LSTM Neural Network, we have used Keras[6] and TensorFlow[7] in Python. The code is written in such a way that the various parameters we want to test easily and massively can be set, which greatly facilitates Hyperparameters tuning. For each set of parameters, the corresponding Epochs / Error chart is also printed.

The code also keeps the network and the parameters that gave the smallest error, and very easily can predict the result for new timeseries and their error.

We have kept a constant number of epochs for all tests and we chose the Mean Squared Error Loss Function, as it is usually better for regression problems, but also to have the same measure of error comparison.

[6] Keras: The Python Deep Learning library (*https://keras.io/*)
[7] TensorFlowAn open source machine learning framework for everyone (*https://www.tensorflow.org/*)

Chapter 3.

Evaluation

3.1. Hyperparameters Tuning

Having properly transposed our network architecture into code, our purpose was then to optimize our model. We started by choosing the optimizer. We decided to use Adam as it seems to be a more sophisticated optimizer than Gradient Descent, as it looks at the momentum and has an adaptive learning rate. Also, our tests showed that Adam converged faster and better than Stochastic Gradient Descent. We used the Mean Squared Error Loss function as it is used extensively in the prediction of numerical values.

Initially, the model was not converging, probably because there was a local optimum. So, we started tuning the learning rate, ending in choosing the price: lr = 0.001

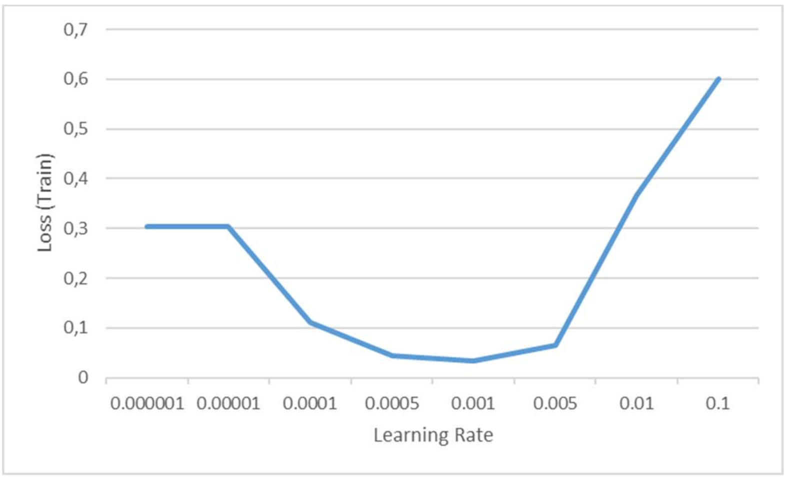

Image 3: Error using different learning rates

We used timeseries with 60 timesteps, i.e. the last 60 days. We also used 3 hidden layers, 50 units per layer and 20% Dropout regularization. However, it was observed that while the train error was satisfactorily improved, the test error remained at the same level, and indeed it was much smaller than the train error. This was an indication that our model has overfitting and how we should reduce its variance. In this case, we usually have two solutions: use more data or use regularization.

We have decided to increase Dropout regularization from 20% to 40%, but this has aggravated our problem. Instead, the test error behaved normally (that is, not always stable and not much less than the train error) when we did not use Dropout regularization at all.

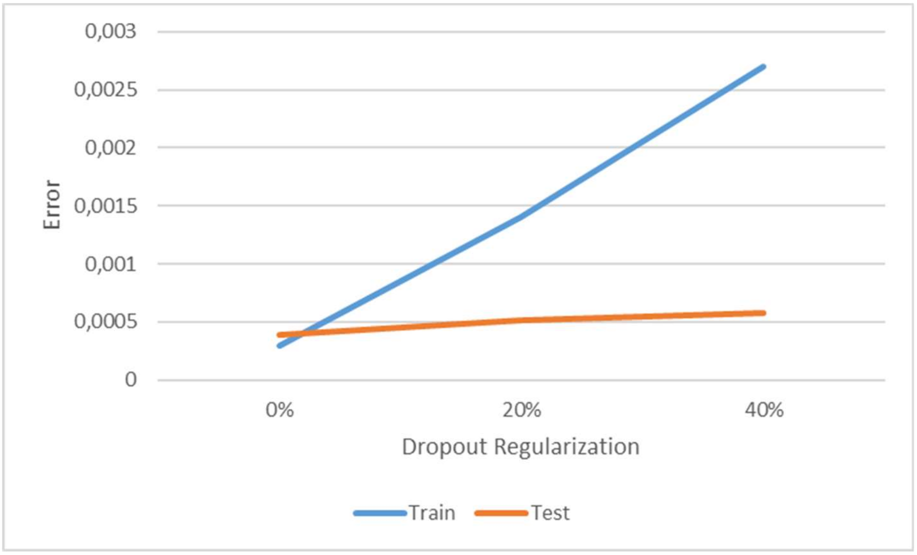

Image 4: Error using Dropout regularization

This may happen because LSTM's are good for long terms but an important thing about them is that they are not very well at memorizing multiple things simultaneously. The logic of drop out is for adding noise to the neurons in order not to be dependent on any specific neuron. By adding drop out for LSTM cells, there is a chance for forgetting something that should not be forgotten.[8] Note that on some problems it is normal for the test error to be less than the train error.

The next solution to reduce the high variance was to use a larger dataset. Unfortunately, there is no larger dataset, so the solution was to double the size of the timeseries to 150 timesteps. And finally this solution yielded.

[8] Answer on Stack Exchange: https://datascience.stackexchange.com/questions/38205/dropout-on-which-layers-of-lstm

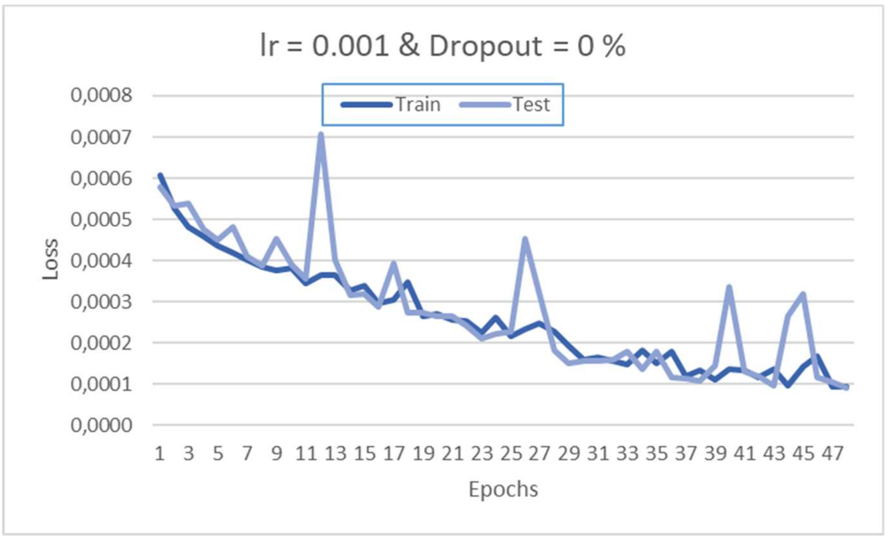

Image 5: Errors without dropout regularization

By also trying to increase the depth of the network with 6 hidden layers and 200 units per layer, we seemed to get just as good results. But because there was a lack of time and computational power, it took more than 15 minutes per epoch. So we decided to keep some intermediate parameters.

3.2. Best Hyperparameters

Having done several tests, we reached the following parameters and trained a network of more epochs based on these:

Test size	20%
Timesteps per Timeseries	150
Hidden Layers	4
Units per Layer	100
Batch Size	128
Dropout	0%
Optimizer	Adam
Learning Rate	0.001
Loss Function	Mean Squared Error
Epochs	100

Image 6: Parameters used to train the final model

The results after 100 epochs:

Loss (Train): 0.000080790

Loss (Test): 0.000079705

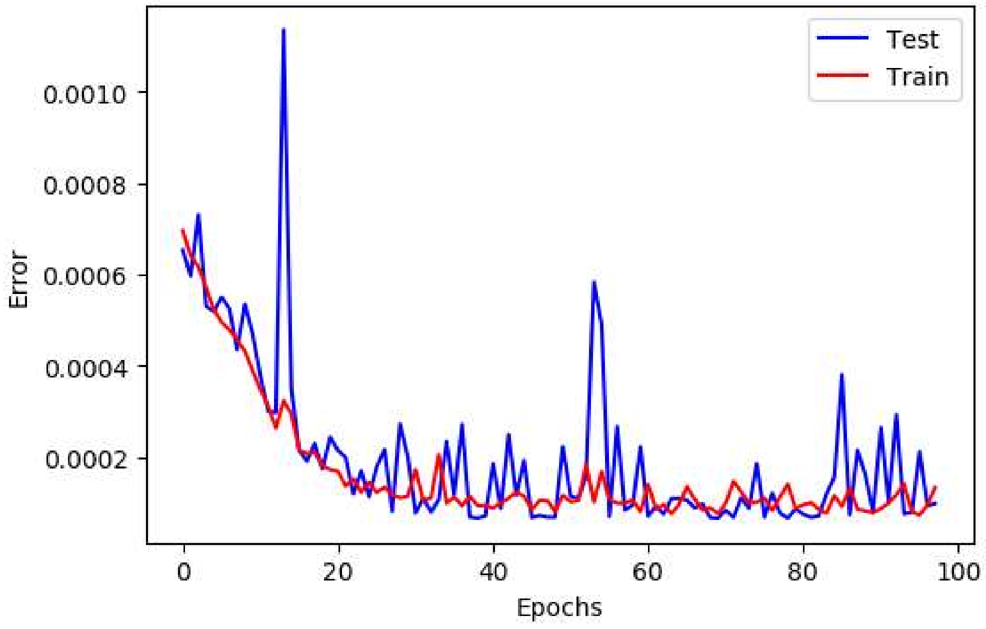

Image 7: Training our network with the best hyperparameters

The results are satisfactory, as the error is of the same order of magnitude of the daily price changes. Also, the error seems to decrease in a smooth way and finally our network converges. There are some spikes of test error, indicating that we need larger dataset to avoid the overfitting more.

Chapter 4.

Conclusion

The results appear to be satisfactory, but in any case, the model is not so trusted to risk our money! Besides, we would all be rich if the problem had a simple solution. From the technical point of view, we have seen how to build our network architecture, how to optimize it and how to evaluate our results.

Our network has been trained well, but the problem of predicting the price is much deeper. In practice, however, there is nothing special to offer tomorrow's Euro to a real trader. The reason is that stocks and currencies do not follow a steady course, but they have multiple changes. We may well predict that the dollar price, for example, will decrease over the next 5 days, but after 10 days eventually it has increased. And that's what matters. Predicting the price in the long run and not tomorrow's day.

4.1. Feature Work

Surely, we can grow our dataset by combining other parities, stocks and financial indicators. This will, however, require a deeper network to be able to train properly, and thus much more time and computational power available. We could all train a deeper network and try some improvements in our architecture.

Improvements can also be made to the construction of timeseries. As we said, the problem of equities and stocks is that they do not grow smoothly. A better approach would be to predict the trend for a longer time, not a single day. We don’t just need to predict multiple timesteps, but a momentum of the price movement. This, however, required much more economical and technical analysis, and went beyond the project needs.

List of Figures