Mostafa ElGhandour

Machine learning engineer Nano-degree

May 9th, 2019

Stock prediction using long-short term memory neural networks

Project overview

Predicting how the stock market will perform is one of the most difficult things to do. There are so many factors involved in the prediction – physical factors vs. physiological, rational and irrational behavior, etc. All these aspects combine to make share prices volatile and very difficult to predict with a high degree of accuracy, which makes it a very project to implement.

In this projected, I created an algorithm that can predict the price of the Egyptian pound relative to the US dollar.

The data dataset I got was from : https://www.investing.com/currencies/usd-egp-historical-data

Problem statement

The goal was to create an algorithm that can learn and predict the pricing of the EGP-USD those were the tasks I followed for doing so:

1. Download and preprocess the EGP-USD pricing historical data.
2. Create a persistence model to use it as a benchmark.
3. Create a LSTM neural networks algorithm that predicts pricing.

4. Compare and tune the parameters for the LSTM to try to improve it more than the persistence model.

Metrics

Mean squared error was used as an accuracy metric for both of the persistence model and the LSTM model, because it's the most simple metric and we don't have any unexpected data in our dataset, so it makes MSE perfect for this dataset.

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2.$$

I created a persistance model to be used as a benchmark model by doing the following :

1.Transform the univariate dataset into a supervised learning problem.

2.Establish the train and test datasets for the test harness

3.Define the persistence model.

4.Make a forecast and establish a baseline performance

5.Plot the output

After creating the persistence model I calculated its MSE and used it as a reference for my model.

Analysis

Data exploration

The data I used was downloaded from a stock market website and I downloaded the data from 2000 but I used only the data after the floating (December 2016) of the Egyptian currency to remove any bias and outliers.

Here is a sample of the data:

	Date	Price	Open	High	Low
5686	04-Nov-16	15.500	15.000	15.760	14.895
5687	05-Nov-16	15.550	15.500	15.850	15.200
5688	07-Nov-16	16.800	16.300	17.700	16.100
5689	08-Nov-16	18.000	17.450	18.100	16.250
5690	09-Nov-16	17.025	17.725	18.025	16.775

The data also had a mean of 17.781 and standard deviation of 0.469

The data had the following fields:

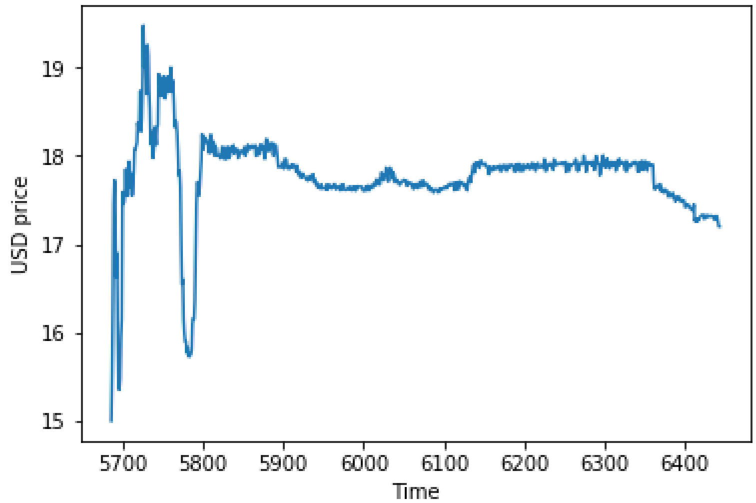
- Date: it's the date of the current USD-EGP price.
- Price: the price of the conversion.
- Open: the opening price of the stock.
- High: the highest price the stock reached that day.
- Low: the lowest price the stock reached that day.
- Change: the change between high and low.

I disregarded the open, high, low, and change features and used the price feature only to simplify the problem, other features can be used in a later version.

Exploratory visualization

The plot below shows the price of the USD-EGP starting from December 2016 to our current day, where the Y-axis represents the price of the Egyptian pound and the X-axis represents the time(day).

The days are simply translated into a sequenced number for easier understanding of the data, graphs and learning of the algorithm.



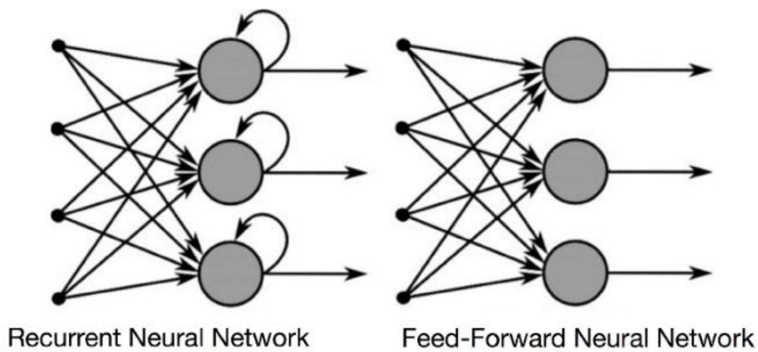
It can be seen that:

- The price doesn't change instantaneously (it takes a few days to spike) which is very good in our case in using the short memory technique.

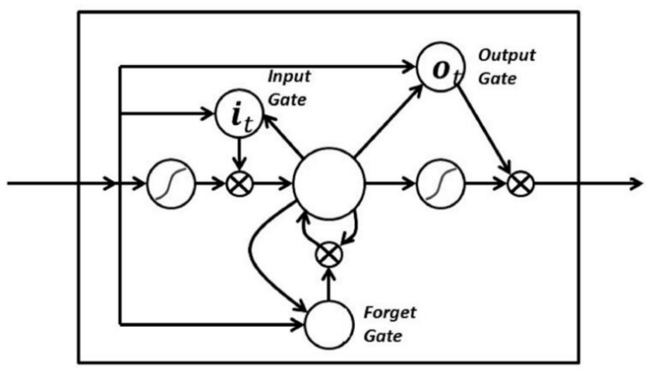
Algorithms and techniques

The algorithm I used is a LSTM neural network which is a powerful type of recurrent 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.

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, unlike feed forward networks, recurrent neural networks actually take feedback from its inputs.



A LSTM also has gates, input, forget and output gate. These gates determine whether or not to let new input in (input gate), delete the information because it isn't important (forget gate) or to let it impact the output at the current time step (output gate). You can see an illustration of a RNN with its three gates below:



The following parameters were tuned to optimize the classifier:

- Number of layers for the neural network
- Layer types
- Layer parameters
- Number of epochs, batch size.

To tune the epochs and batch size, I used grid search to optimize the parameters which in return gave me the best parameters combination from the 2 parameter arrays I entered.

Benchmark

Establishing a baseline is essential on any time series forecasting problem,
The most common baseline method for supervised machine learning is the Rule algorithm.
This algorithm predicts the majority class in the case of classification, or the average outcome in the case of regression. This could be used for time series, but does not respect the serial correlation structure in time series datasets. The equivalent technique for use with time series dataset is the persistence algorithm.

So, I created my own persistence model to use it as a benchmark,

The steps:

- Create a lagged representation of the dataset.
- Split the dataset into train and testing data.
- Train the data using the original and the lagged dataset.
- Calculate the MSE of the model and use it as a reference.

Having done these steps, I got an MSE of 0.002 which was used as a reference for my model.

Methodology

Data preprocessing

The steps were as follows:

1. The dataset was read using pandas.
2. The price column was taken into a “working data” list.

3. All data before December 2016(floating of the Egyptian pound) was removed.
4. The data was split into training and test sets with respect to time (randomizing the training and testing set would be very hard to predict in a time series model).

The data preprocessing was pretty simple and straight forward as my data is not complicated at all.

Implementation

The implementation was as follows:

1. Preprocessing the data:
    a. the dataset was read using pandas.
    b. The price "column" was put into a "working_data" list.
    c. The dataset was plotted so it could be further analyzed.
2. Creating the persistence model:
    a. A lagged representation of the "price" column was created using (concat).
    b. Splitting the data into train and test sets.
    c. The mean squared error of the data and its lagged representation was calculated
3. Creating the LSTM model:
    a. Split the data into train and test sets
    b. Create a function that creates another dataset in a lagged representation, 1-time step was chosen to be able t o compare it to the persistence model.

c. Create a neural network with 1 input layer and a hidden layer with 4 neurons and an output layer, I tried to further optimize the layers, but this was the best results I got.
d. Started with some parameters (epochs=100, batch size =5) which led to a very bad MSE (over 0.5), I tried to tune manually but failed which led to me the next step.
e. Create a gird search to find the best parameters for the following neural network
f. Use the optimized parameters (batch size and epochs) into my neural network.
g. Compare and plot the result of the trained model.

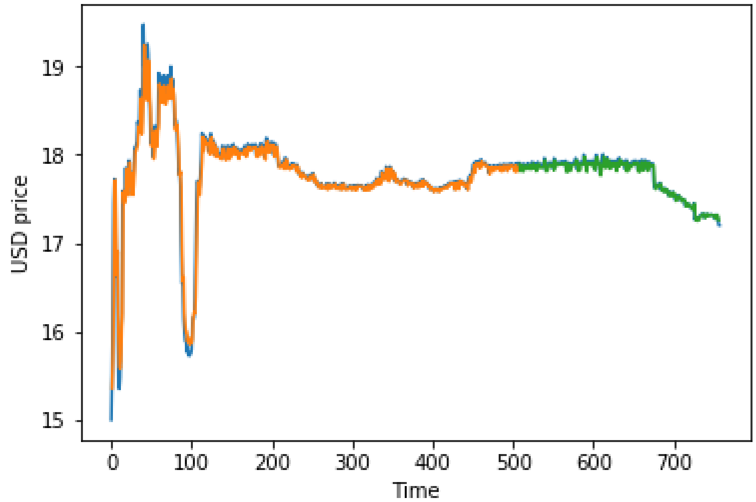
Refinement

Grid search was used to tune the parameters, as tuning them manually was almost impossible so I created a set of number of the epochs (10,25,50100, 500, 250) and a set of parameters of the batch size(1,10, 20, 40, 60, 80, 100) and using the gridsearch I got the optimal parameters which were 25 epoch and 1 batch size after 3 hours of processing.

Results

Model evaluation, validation and justification

The final tested model with the final parameters of epochs = 25 and batch size = 5 with 1 input layer and a hidden layer with 4 neurons and an output layer gave a test score on the mean squared error of 0.05 which is more than what I got in the persistence model, because I could not improve it more than that, because if I increase the batch size I lose some information and my dataset wouldn't react well with that, and if I increase the epochs it starts memorizing the data and also takes a lot of computing power.



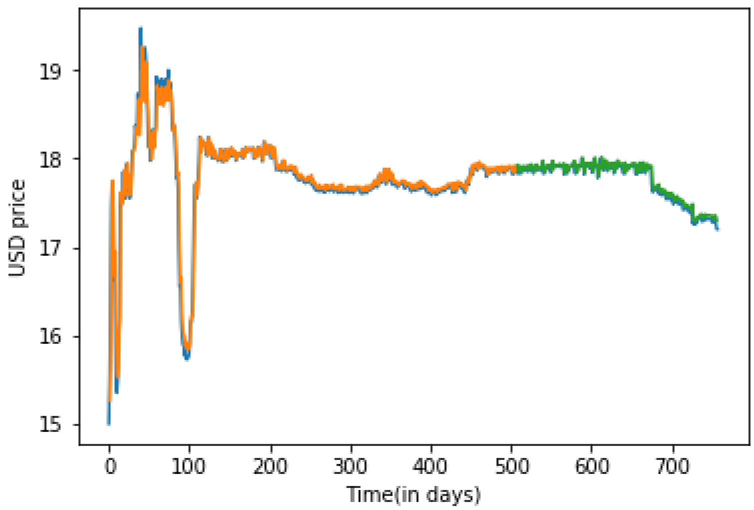
The blue line represents the original data, the orange is the predicted training set and the green is the predicted test set,

Shown in the figure above how accurately my model can predict the training and testing data.

The model can be trusted and used as a stock predication algorithm even with a little bit of error because relying on the persistence model you can get pretty accurate results but only in ranges of 1-2 days but with my model you can get far bigger range, but it needs more testing.

Conclusion

Free-Form visualization



The blue line represents the original data, the orange is the predicted training set and the green is the predicted test set.

as shown in the above figure, the model does an excellent job of fitting the train and test datasets but still with a larger error than the persistence model, it needs some parameter tuning to improve it.

Reflection

To summarize the process done in this project we can simply look at these steps:

1. An initial problem and relevant dataset were chosen.
2. The dataset was downloaded and preprocessed.
3. A persistence model was created to be used as a benchmark model and its mean squared error was calculated.
4. A LSTM neural network was created.
5. A gridsearch algorithm was implemented to tune the parameters for the neural network.
6. The LSTM was tuned using the output results from the gridsearch.

7. The mean squared error from the trained LSTM model was calculated and compared to the persistence model's.

Creation and tuning of the LSTM was the hardest part as I could not put my hands on how to split the data into labels and features, and the networks layers were really hard to play with as they would just worsen the results whenever I tried adding/removing layers.

Improvement

of course, there's a large room for improvement in this algorithm so it can be used as a legitimate algorithm for predicting stock prices, and they can be summarized in the following:

1. better hardware to ease the process of using gridsearch and using more sets of parameters.
2. More features can be used in the model to improve its accuracy.
3. More lagged time steps should be used rather the 1-time step only I used, that could help the algorithm be more accurate in predicting the far future.
4. And of course an easy GUI could be used for prediction.

References

1. "Time Series Prediction with LSTM Recurrent Neural Networks in Python with Keras." *Machine Learning Mastery*, 26 Apr. 2019, machinelearningmastery.com/time-series-prediction-lstm-recurrent-neural-networks-python-keras/.

2. "How to Grid Search Hyperparameters for Deep Learning Models in Python With Keras." *Machine Learning Mastery*, 9 May 2019, machinelearningmastery.com/grid-search-hyperparameters-deep-learning-models-python-keras/.
3. Nayak, Asutosh. "Predicting Stock Price with LSTM." *Towards Data Science*, Towards Data Science, 18 Mar. 2019, towardsdatascience.com/predicting-stock-price-with-lstm-13af86a74944.
4. Nayak, Asutosh. "Finding the Right Architecture for Neural Network." *Towards Data Science*, Towards Data Science, 18 Mar. 2019, towardsdatascience.com/finding-the-right-architecture-for-neural-network-b0439efa4587.
5. "How to Make Baseline Predictions for Time Series Forecasting with Python." *Machine Learning Mastery*, 26 Apr. 2019, machinelearningmastery.com/persistence-time-series-forecasting-with-python/.