STOCK PRICE PREDICTION

Using Linear Regression &

Long-short term Memory (LSTM)

Florian Harsch

05.12.2019

Udacity Machine Learning Nanodegree Capstone Project

Table of content

I. Definition

Project Overview

In this project I will examine the financial market, especially the so called German "DAX". DAX is the abbreviation for "Deutscher Aktien Index". He is the most important German stock index. It measures performance of the 30 largest and (in terms of free float market capitalization) most liqiud companies. He also represents around 80% of the market capitalization of listed companies in Germany. As additional source I will use the VDAX-NEW. The VDAX-NEW Index expresses the variation margin – the implied volatility – of the DAX anticipated on the derivatives market. The VDAX indicates in percentage points the volatility to be expected in the next 30 days for the DAX. The basis for the calculation of this index is provided by the DAX option contracts.

Stock trading has a long history. The first share was published in 1288 („Stora Kopparbergs Bergslags Aktiebolag“ in Falun"). Since then traders have been trying to make the best possible profit for themselves. Some were and some are still sure that they can predict the devolpment of th market.

Already a lot of statistical and historical data is used to predict the development of the market. There are also some research using Machine Learning to predict some parts of the market, e.g.:

- *Prediction Stock Price Direction using Support Vector Machines by Saahil Madge and Swati Bhatt*
- *Support Vector Machine for prediction of futures prices in Indian Stock Market, Shom Prasad Das and Sudarsan Padhy*

As a shareholder of shares, it is very interesting for me to optimize my portfolio and thus my profits. I hope to be able to optimize my purchase or sales decision with the help of machine learning approaches

Problem Statement

The object of this project is to check if machine learning techniques can be used to predict the DAX. The task is to create an stock price predictor that takes historical daily traiding infos over a certain period of time and deliver as output prediction for the future.

Metrics

As this problem is a regression task, I suggest to R-Squared and Root Mean Square Error (RMSE), additionally Mean Absolute Error (MAE).

R-squared (R2) is a statistical measure that represents the proportion of the variance for a dependent variable that's explained by an independent variable or variables in a regression model. 6

Root Mean Square Error (RMSE) is the standard deviation of the residuals (prediction errors). Residuals are a measure of how far from the regression line data points are; RMSE is a measure of how spread out these residuals are. In other words, it tells you how concentrated the data is around the line of best fit. Root mean square error is commonly used in climatology, forecasting, and regression analysis to verify experimental results.

Mean Absolute Error (MAE) is a measure of difference between two continuous variables. Assume X and Y are variables of paired observations that express the same phenomenon. Examples of Y versus X include comparisons of predicted versus observed.

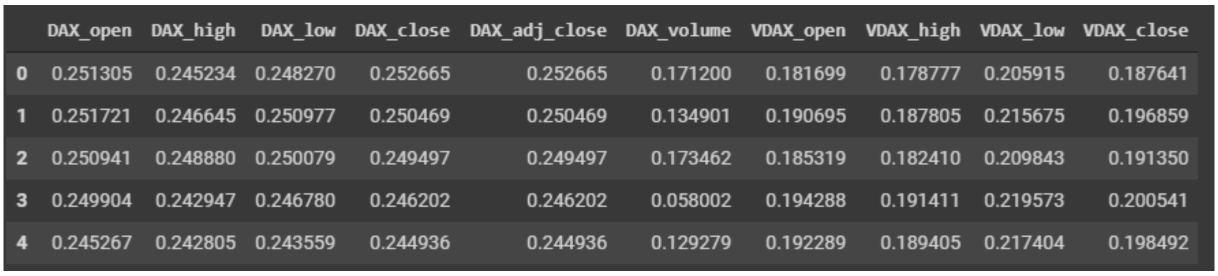
II. Analysis

Data Exploration

The data used in this project is of the German Stock Index, called DAX from 15. June 2002 up to 17. September 2019 and the VDAX for the same period.

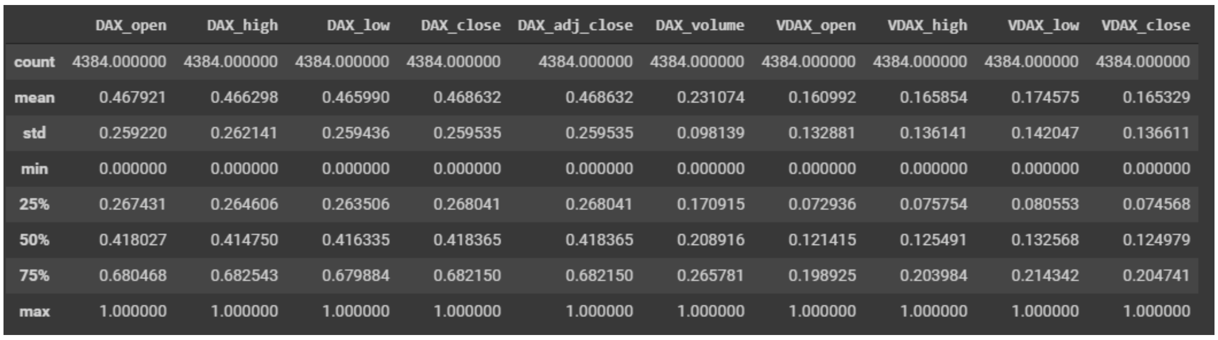
I merged both datasets to one dataset. As I want to create a predictor for the DAX_adj_close for every timestep, the DAX_adj_close is the interesting column for me. I worked with it in my following analysis.

I cleaned it, if there were missing values. And as there were big deviations in DAX_adj_close, I decided to normalize the data. At all I have 4,384 valid datapoints in the dataset.



	DAX_open	DAX_high	DAX_low	DAX_close	DAX_adj_close	DAX_volume	VDAX_open	VDAX_high	VDAX_low	VDAX_close
0	0.251305	0.245234	0.248270	0.252665	0.252665	0.171200	0.181699	0.178777	0.205915	0.187641
1	0.251721	0.246645	0.250977	0.250469	0.250469	0.134901	0.190695	0.187805	0.215675	0.196859
2	0.250941	0.248880	0.250079	0.249497	0.249497	0.173462	0.185319	0.182410	0.209843	0.191350
3	0.249904	0.242947	0.246780	0.246202	0.246202	0.058002	0.194288	0.191411	0.219573	0.200541
4	0.245267	0.242805	0.243559	0.244936	0.244936	0.129279	0.192289	0.189405	0.217404	0.198492

Picture 1: First 5 rows of merged dataset

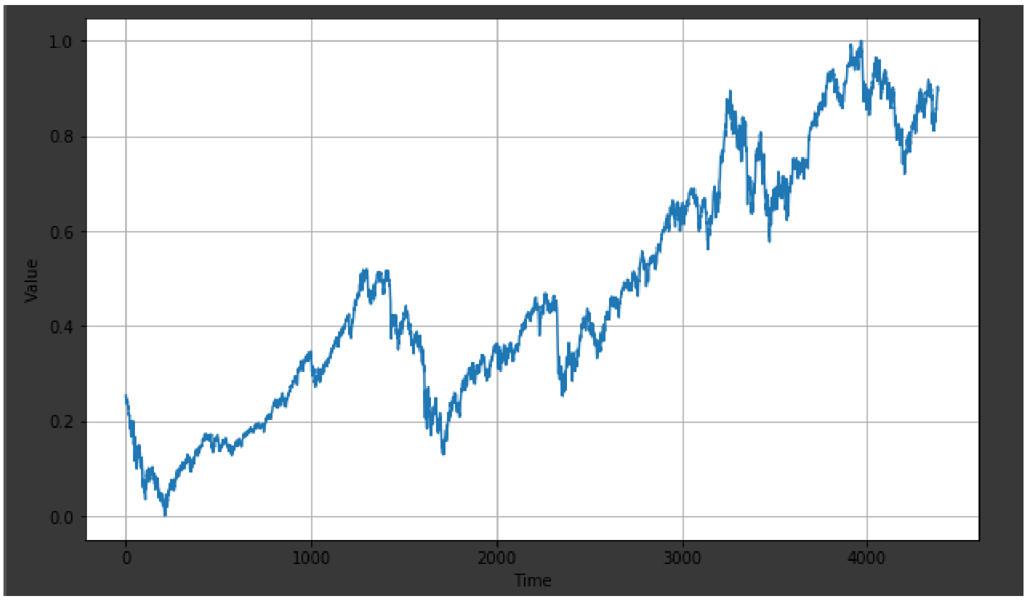


	DAX_open	DAX_high	DAX_low	DAX_close	DAX_adj_close	DAX_volume	VDAX_open	VDAX_high	VDAX_low	VDAX_close
count	4384.000000	4384.000000	4384.000000	4384.000000	4384.000000	4384.000000	4384.000000	4384.000000	4384.000000	4384.000000
mean	0.467921	0.466298	0.465990	0.468632	0.468632	0.231074	0.160992	0.165854	0.174575	0.165329
std	0.259220	0.262141	0.259436	0.259535	0.259535	0.098139	0.132881	0.136141	0.142047	0.136611
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	0.267431	0.264606	0.263506	0.268041	0.268041	0.170915	0.072936	0.075754	0.080553	0.074568
50%	0.418027	0.414750	0.416335	0.418365	0.418365	0.208916	0.121415	0.125491	0.132568	0.124979
75%	0.680468	0.682543	0.679884	0.682150	0.682150	0.265781	0.198925	0.203984	0.214342	0.204741
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000

Picture 2: Statistics of Dataset

Exploratory Visualization

Here I will visualize the development of DAX_adj_close over the time period. Therefore I used matplotlib.



Picture 3: Development of DAX_adj_close

Used Algorithms and Techniques

As previous explained the goal is to predict the value for each time step. During the my research and coding I found a Udacity course called “Intro to TensorFlow for DeepLearning”. There were a lot of different techniques explained. As I mentioned in my proposal I will use Linear Regression and a special Recurrent Neural Network (RNN), a so called Long-short Term Memory Network (LSTM).

For the Linear Regression I used more or less the approach of the course and second approach just using Sklearn. Here I just tuned the learning rate.

For the LSTM I took over the approach I learned in the course. I tuned some more hyperparameters:

- Learning Rate
- Number of epocs
- Batch size

I didn’t tuned other hyperparameters, e.g.:

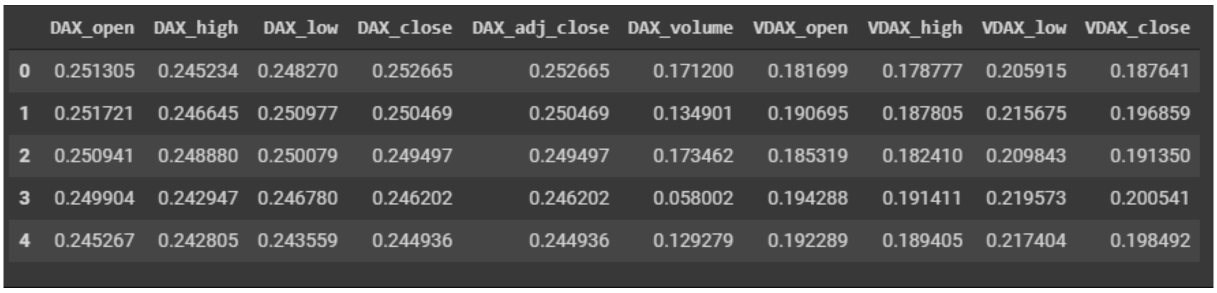
- Percentage training, test split
- Loss function
- Optimizer Function
- Number of layers
- Kind of layers (e.g. dropout)
- Number of Nodes

III. Methodology

Data Preprocessing

I did the following steps:

1. I downloaded the development of DAX starting 15/05/2002 to 18/09/2019 from Yahoo finance and stored it as csv. Also downloaded the development of VDAX for the same time frame from ariva and stored it as csv.
2. Renamed the columns names for both dataset.
3. Merged both datasets on date
4. Checked for any missing values using describe and isna function.
5. Stored the dataset as df_clean.csv for future reuseability.
6. Normalized the data using MinMaxScaler function from Sklearn.
7. Stored the dataset as df_norm.csv for future reuseability.



	DAX_open	DAX_high	DAX_low	DAX_close	DAX_adj_close	DAX_volume	VDAX_open	VDAX_high	VDAX_low	VDAX_close
0	0.251305	0.245234	0.248270	0.252665	0.252665	0.171200	0.181699	0.178777	0.205915	0.187641
1	0.251721	0.246645	0.250977	0.250469	0.250469	0.134901	0.190695	0.187805	0.215675	0.196859
2	0.250941	0.248880	0.250079	0.249497	0.249497	0.173462	0.185319	0.182410	0.209843	0.191350
3	0.249904	0.242947	0.246780	0.246202	0.246202	0.058002	0.194288	0.191411	0.219573	0.200541
4	0.245267	0.242805	0.243559	0.244936	0.244936	0.129279	0.192289	0.189405	0.217404	0.198492

Picture 4: Final dataset

8. Split the data into tr Normally I would split my dataset as mentioned in the 6roposal in three sets:
   - Training
   - Validation and
   - Test set.

   But in the meantime I learned there that it’s unusual to do this. Because the test set is not reliable as in regular machine learning and because we need to use it for training before deploying on model, it is common to just use a training period and a validation period. In other words, the test period is in the future. So I will do an 80% Training to 20% Validation set split. As I have at al 4384 points, that means:
   - Training 80% = 3,507
   - Validation 20% = 876

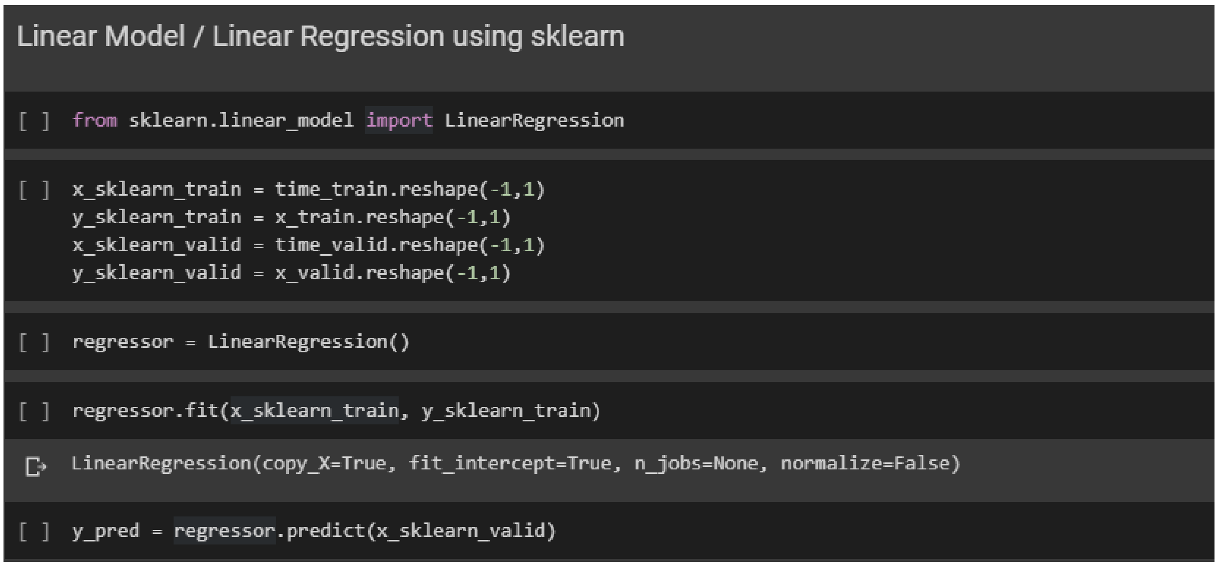
Implementation

As I want to plot and evaluate the results for each model and each try, I decided to create to functions doing this:

- Plot_series function to plot
- Evaluaton metrics function to evaluate the three metrices:
  - Mean Absolute Error
  - R**2
  - Root Mean Square Error

As mentioned before I implemented two Linear Regression models.

The first one was using Keras LinearRegression. For that I also need to reshape the train and valid data as another shape is therefore demanded.

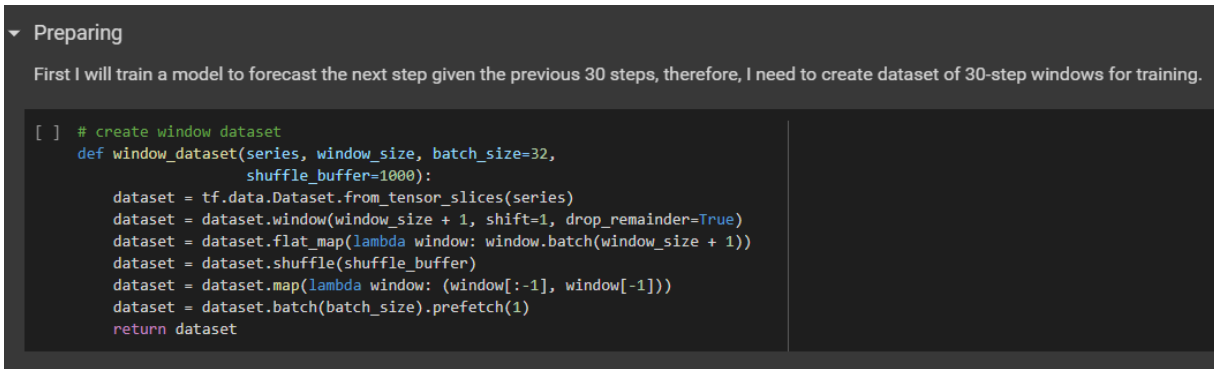


Linear Model / Linear Regression using sklearn

```
[ ] from sklearn.linear_model import LinearRegression

[ ] x_sklearn_train = time_train.reshape(-1,1)
    y_sklearn_train = x_train.reshape(-1,1)
    x_sklearn_valid = time_valid.reshape(-1,1)
    y_sklearn_valid = x_valid.reshape(-1,1)

[ ] regressor = LinearRegression()

[ ] regressor.fit(x_sklearn_train, y_sklearn_train)

    LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)

[ ] y_pred = regressor.predict(x_sklearn_valid)
```

Picture 5: Code LinearRegression using sklearn

As I was not happy with the result on the one hand and learned in the course to use TensorFlow, I created a second LinearRegression Model. To do this I first needed to create a dataset of 30-step windows, as the mode should forecast the next step, given the previous 30 days.



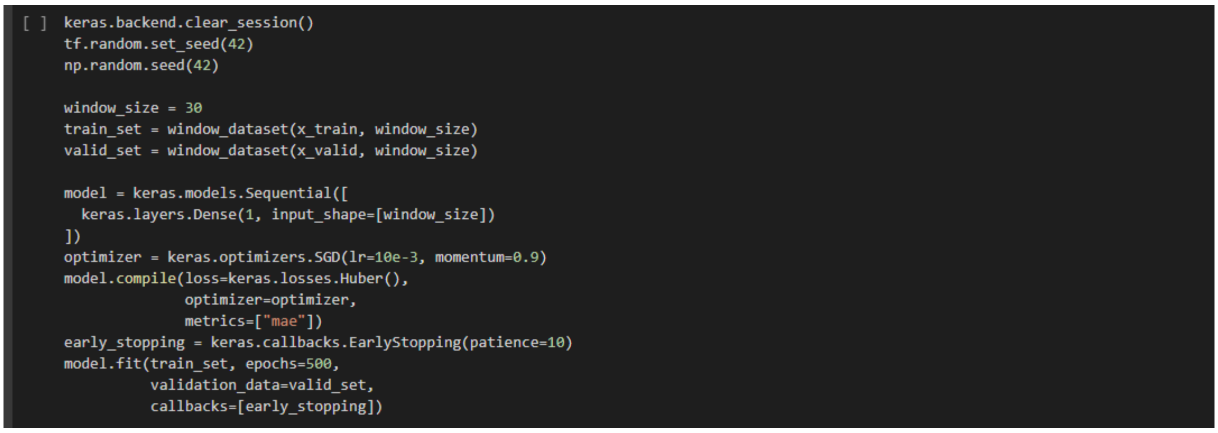
Preparing

First I will train a model to forecast the next step given the previous 30 steps, therefore, I need to create dataset of 30-step windows for training.

```
[ ] # create window dataset
    def window_dataset(series, window_size, batch_size=32,
                       shuffle_buffer=1000):
        dataset = tf.data.Dataset.from_tensor_slices(series)
        dataset = dataset.window(window_size + 1, shift=1, drop_remainder=True)
        dataset = dataset.flat_map(lambda window: window.batch(window_size + 1))
        dataset = dataset.shuffle(shuffle_buffer)
        dataset = dataset.map(lambda window: (window[:-1], window[-1]))
        dataset = dataset.batch(batch_size).prefetch(1)
        return dataset
```

Picture 6: Preparing window dataset for TensorFlow

Next steps were to train the model, find the right learning rate and finally execute the model with the optimized learning rate and more epochs.

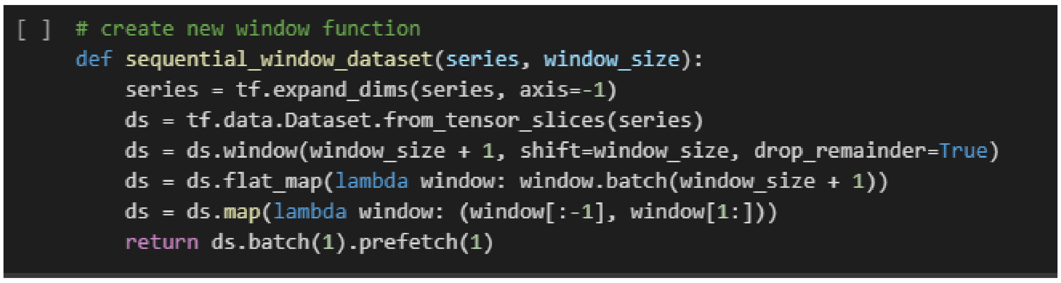


```
[ ] keras.backend.clear_session()
    tf.random.set_seed(42)
    np.random.seed(42)

    window_size = 30
    train_set = window_dataset(x_train, window_size)
    valid_set = window_dataset(x_valid, window_size)

    model = keras.models.Sequential([
      keras.layers.Dense(1, input_shape=[window_size])
    ])
    optimizer = keras.optimizers.SGD(lr=10e-3, momentum=0.9)
    model.compile(loss=keras.losses.Huber(),
                  optimizer=optimizer,
                  metrics=["mae"])
    early_stopping = keras.callbacks.EarlyStopping(patience=10)
    model.fit(train_set, epochs=500,
              validation_data=valid_set,
              callbacks=[early_stopping])
```

Picture 7: Optimized LinearRegression Model using TensorFlow

This model performed much better as the other LinearRegression Model, so I stopped any optimization here.

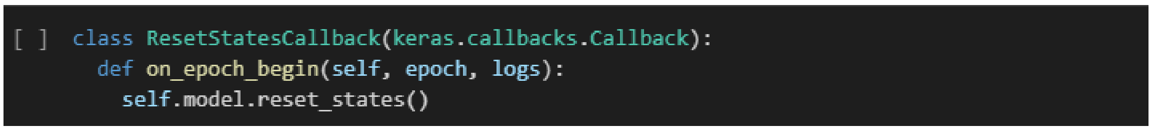
For the LSTM I needed a new data window. The last one was stateless and now I need a stateful one. That means, that it can't randomly take batches, instead it has to take them sequentially. So the LSTM can learn patterns longer than the batch size.



```
[ ] # create new window function
    def sequential_window_dataset(series, window_size):
        series = tf.expand_dims(series, axis=-1)
        ds = tf.data.Dataset.from_tensor_slices(series)
        ds = ds.window(window_size + 1, shift=window_size, drop_remainder=True)
        ds = ds.flat_map(lambda window: window.batch(window_size + 1))
        ds = ds.map(lambda window: (window[:-1], window[1:]))
        return ds.batch(1).prefetch(1)
```

Picture 8: Window Dataset for LSTM

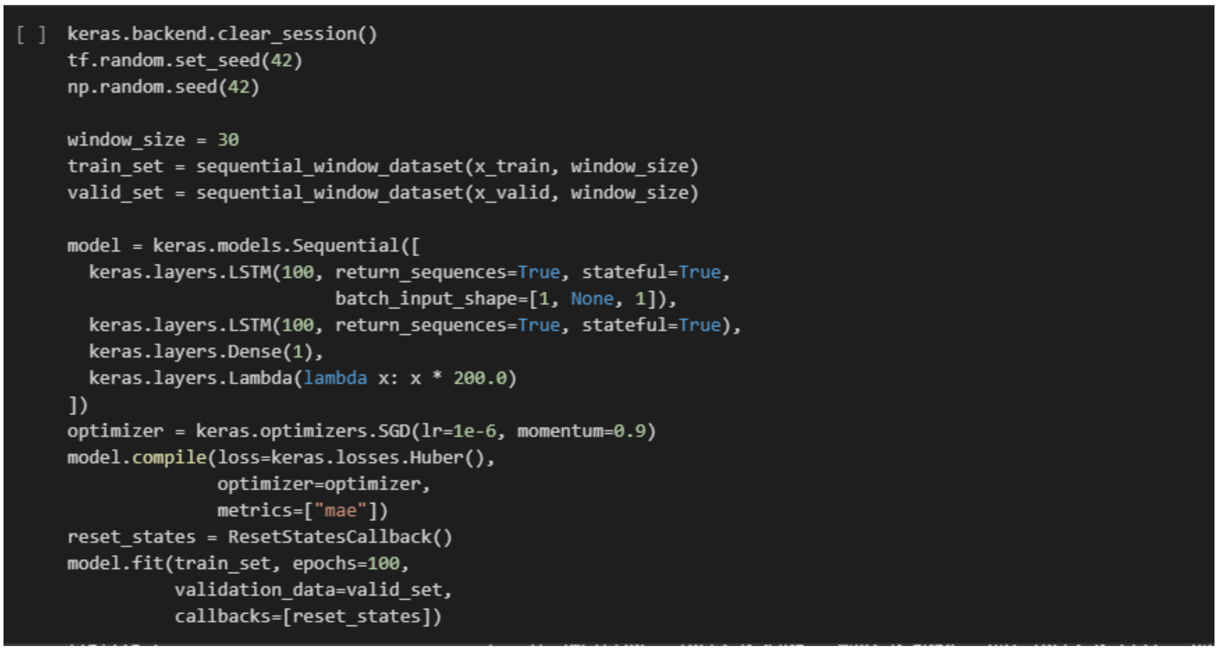
But as the LSTM don't know the end of a time series, I need to tell it. One way is to define a custom callback. Therefore I need to create a subclass of the "keras.callbacks.callback" class and also implement the "on_epoch_begin" method and make it call "self.model.reset_sates". So the LSTM states will be reset before each epoch.



```
[ ] class ResetStatesCallback(keras.callbacks.Callback):
      def on_epoch_begin(self, epoch, logs):
        self.model.reset_states()
```

Picture 9: ResetStatesCallback Function

Then I started a basis LSTM with no optimization. I used that as a basis to compare with my optimized models.



```
[ ] keras.backend.clear_session()
    tf.random.set_seed(42)
    np.random.seed(42)

    window_size = 30
    train_set = sequential_window_dataset(x_train, window_size)
    valid_set = sequential_window_dataset(x_valid, window_size)

    model = keras.models.Sequential([
      keras.layers.LSTM(100, return_sequences=True, stateful=True,
                        batch_input_shape=[1, None, 1]),
      keras.layers.LSTM(100, return_sequences=True, stateful=True),
      keras.layers.Dense(1),
      keras.layers.Lambda(lambda x: x * 200.0)
    ])
    optimizer = keras.optimizers.SGD(lr=1e-6, momentum=0.9)
    model.compile(loss=keras.losses.Huber(),
                  optimizer=optimizer,
                  metrics=["mae"])
    reset_states = ResetStatesCallback()
    model.fit(train_set, epochs=100,
              validation_data=valid_set,
              callbacks=[reset_states])
```

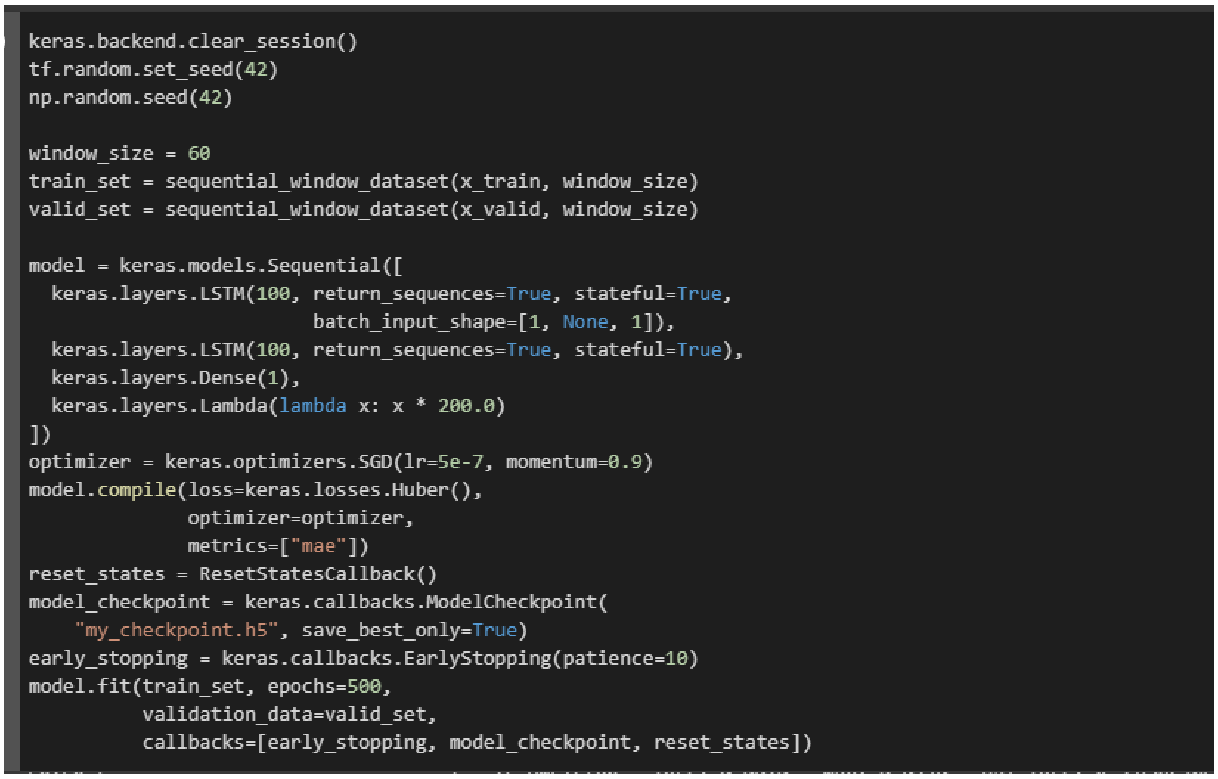
Picture 10: Basic LSTM Model

Refinement:

I tuned the following hyperparameters:

- Learning rate: start: 1e-6, optimized: 5e-7
- Batch size: start: 30, optimized: 60
- Number of epochs: start: 100, optimized: 500

The optimized model looked as follows:

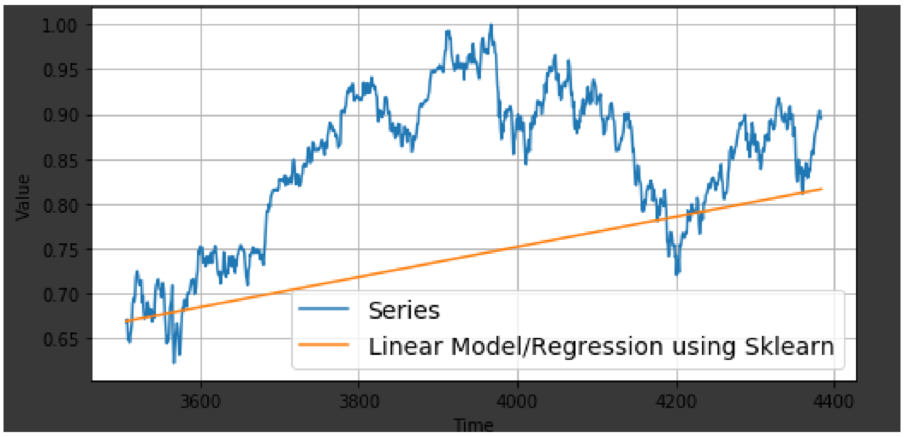


```
keras.backend.clear_session()
tf.random.set_seed(42)
np.random.seed(42)

window_size = 60
train_set = sequential_window_dataset(x_train, window_size)
valid_set = sequential_window_dataset(x_valid, window_size)

model = keras.models.Sequential([
  keras.layers.LSTM(100, return_sequences=True, stateful=True,
                    batch_input_shape=[1, None, 1]),
  keras.layers.LSTM(100, return_sequences=True, stateful=True),
  keras.layers.Dense(1),
  keras.layers.Lambda(lambda x: x * 200.0)
])
optimizer = keras.optimizers.SGD(lr=5e-7, momentum=0.9)
model.compile(loss=keras.losses.Huber(),
              optimizer=optimizer,
              metrics=["mae"])
reset_states = ResetStatesCallback()
model_checkpoint = keras.callbacks.ModelCheckpoint(
    "my_checkpoint.h5", save_best_only=True)
early_stopping = keras.callbacks.EarlyStopping(patience=10)
model.fit(train_set, epochs=500,
          validation_data=valid_set,
          callbacks=[early_stopping, model_checkpoint, reset_states])
```

IV. Results

Model Evaluation and Validation

LinearRegression using sklearn

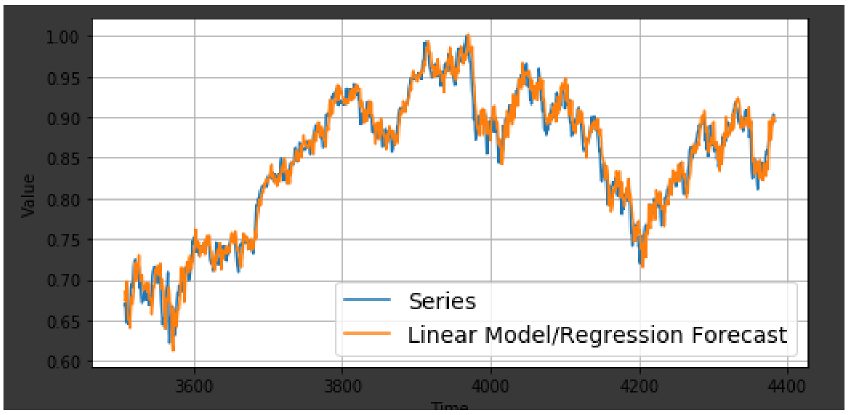


Picture 11: LinearRegression using Sklearn plot

Metrics:

Model	LinearRegression (Sklearn)
MAE	0.10624727884119481
R_2	-1.2640475644640832
RMSE	0.12750324747954495

LinearRegression using TensorFlow

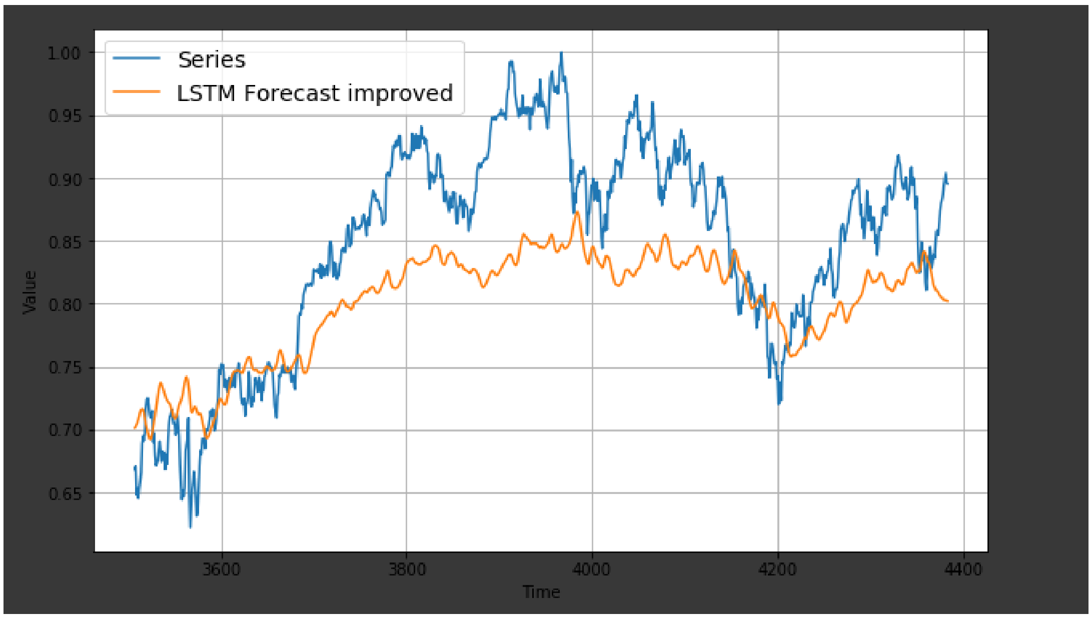


Picture 12: Linear Regression using TensorFlow plot

Metrics:

Model	LinearRegression (TensorFlow)
MAE	0.01242074316812058
R_2	0.9646051283521777
RMSE	0.015942214343151323

Long-short term memory



Picture 13: LSTM plot

Metrics:

Model	LSTM
MAE	0.0564194254292696
R_2	0.36822203892739414
RMSE	0.06735358380736665

Justification

Now it's time to compare the three models. And the winner is LinearRegression using TensorFlow. In all three metrics it performed the best.

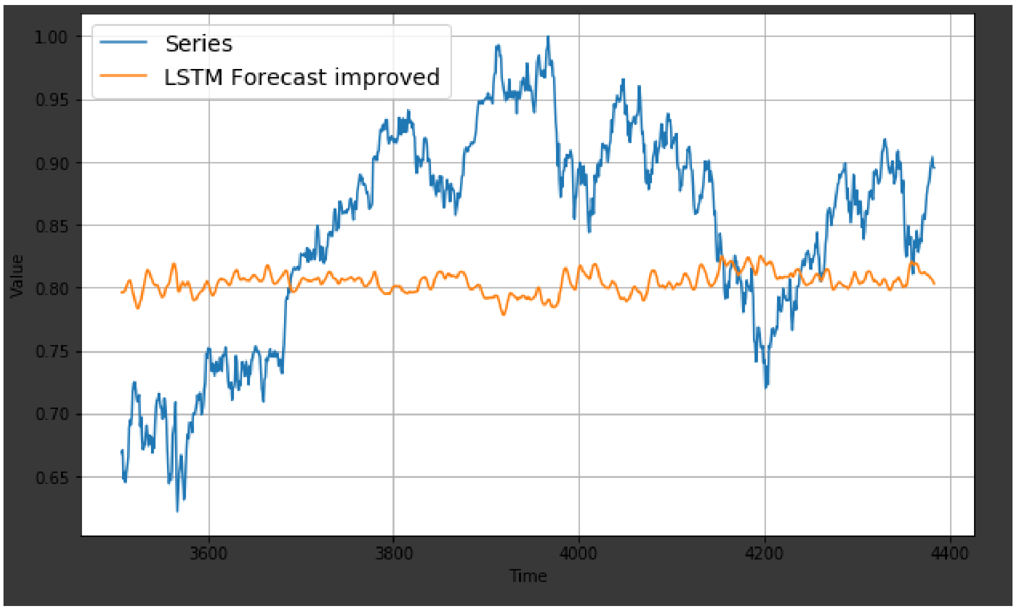
It is slight better than the LSTM and much more better than the LinearRegression using Sklearn.

Model	LinearRegression (Sklearn)	LinearRegression (TensorFlow)	LSTM
MAE	0.10624727884119481	0.01242074316812058	0.0564194254292696
R_2	-1.2640475644640832	0.9646051283521777	0.36822203892739414
RMSE	0.12750324747954495	0.015942214343151323	0.06735358380736665

V. Conclusion

Free-Form Visualization

Here I would like to show one of my improved plots of the LSTM.



Picture 14: LSTM with first optimization

I was really shocked how bad it was. It was more a less a horizontal line around 0.8.

Reflection

To recap, I will again show the path I had taken through this project:

1. Set up infrastructure
    a. Notebook
    b. install libraries
2. Data processing
    a. access raw data fron online sources
    b. incorporate data soruces
    c. check and clean data
    d. Split in Training and Testing set
3. Develop and train models
    a. Linear Regression model
    b. Tune parameters
    c. Develop LSTM Model
    d. Tune hyperparameters LSTM Model (e.g. network size and shape, learning rate, activation functions,...)
4. Document results
    a. plot values
    b. compare models
    c. write final report

As I started the project I was somehow frightened, because I had never done such thing before. Luckily I found a lot of good teaching material on udacity. Especially the course "Intro to TensorFLow for DeepLearning". I learned so much and was really excited how my models would perform. I was a little bit disappointed that the LSTM lost against the LinearRegression. But I will try to improve my LSTM further to beat the LinearRegression. I'm also not sure, if the LinearRegression is not somehow overfitting.

The major problems I was faced was as usual the starting point. As I searched how others made a stock price prediction, I really found a lot of codes. But most of them were hardly understandable. So I decided for myself, that one of the most important ToDos in coding is, to write understandable code and describe what you are doing.

Improvement

First of all, I'm really proud to finish and hopefully to pass this course. It was a lot of fun and the courses on Udacity are still amazing. I also tried several other moocs but always came back to Udacity.

I also liked the project, but I have some improvements to do. The next time I would like to compare more different models to each other. Also I want to add some interactions where the user can test and try different stuff.