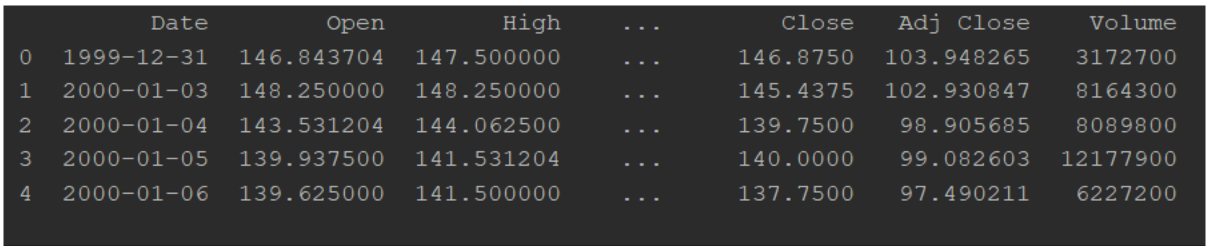
Trading stocks

Developing trading strategy is quite challenging task because of variety of approaches that could be applied. Some of the mostly used are: momentum strategy, reversion strategy and forecasting strategy.

For this assignment, I will use forecasting strategy combined with deep learning. The main idea is to predict the next value of a stock based on some historical factors (data).

Data

In this case, our historical data are prices of a SPY stock from 1st Jan 2010 to 31st Dec 2015. This data contains information for opening, closing, highest and lowest prices for each day, as presented on the Figure 1. In this project, only *CLOSE* prices were used for prediction and the graphic is showed on Figure 2. This data contains 4025 days for the given period.



```
         Date        Open        High  ...       Close   Adj Close    Volume
0  1999-12-31  146.843704  147.500000  ...    146.8750  103.948265   3172700
1  2000-01-03  148.250000  148.250000  ...    145.4375  102.930847   8164300
2  2000-01-04  143.531204  144.062500  ...    139.7500   98.905685   8089800
3  2000-01-05  139.937500  141.531204  ...    140.0000   99.082603  12177900
4  2000-01-06  139.625000  141.500000  ...    137.7500   97.490211   6227200
```

Figure 1: Data for period from 1st Jan 2010 to 31st Dec 2015

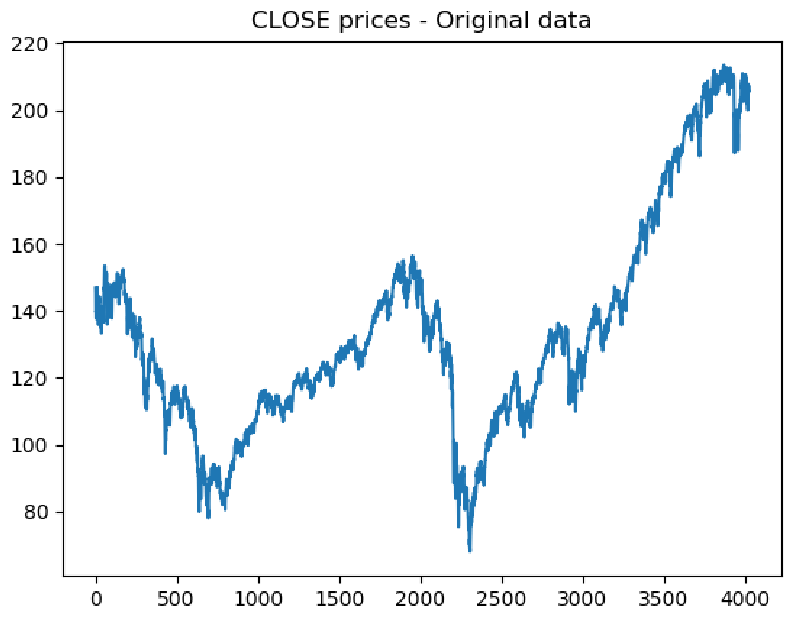


Figure 2: *CLOSE* prices data

This data should be preprocessed so to put it into the right range. Normalization is problem here because trading data tends to change its statistics (mean, variance) over time, so we can't simply subtract the mean from this whole dataset and divide it by its variance. Instead, we split the data into adjacent windows of size *input_length*, and values in each window divide with the value of last sample in previous window. After this, we get normalized data as shown on the Figure 3. On this figure, normalized data is averaged across windows where *input_length*=3. From this original data 90% is train data and the rest 10% is validation data.

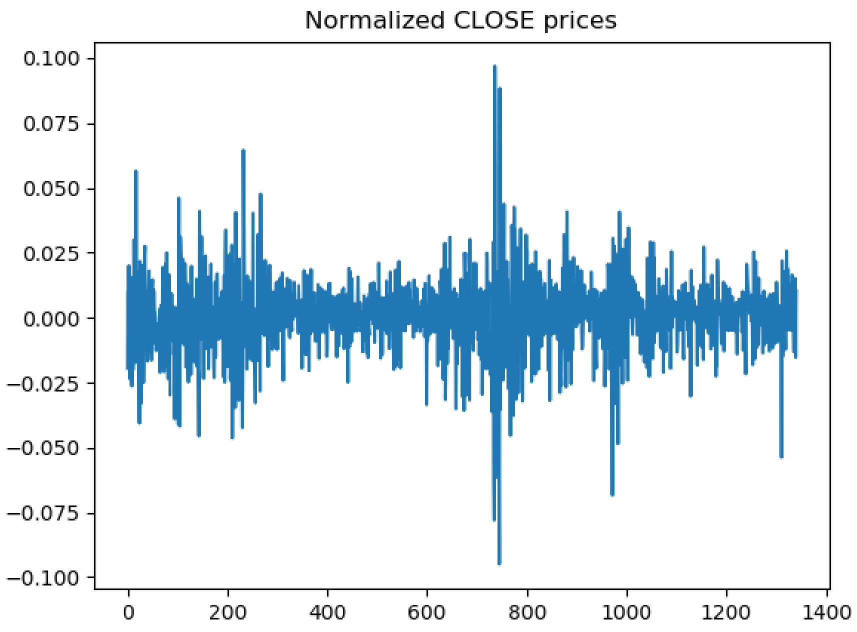


Figure 3: Normalized *CLOSE* prices

Prediction with ML model

Before choosing the right machine learning model for prediction, I have read some articles so to get sense of what kind of model to use. After this short research, I decided to use *LSTM* recurrent neural network for prediction. This is one kind of recurrent neural networks which allows working with sequential data and gives quite good results when working with time-series signals. It overcomes some standard problems in vanilla recurrent networks such as vanishing/exploding gradients. This problem occurs when gradient increases to much or decreases to 0, so it can not propagate back through the network and update its parameters. On the other side, *LSTM* solves this problem with its unique architecture shown on the Figure 4.

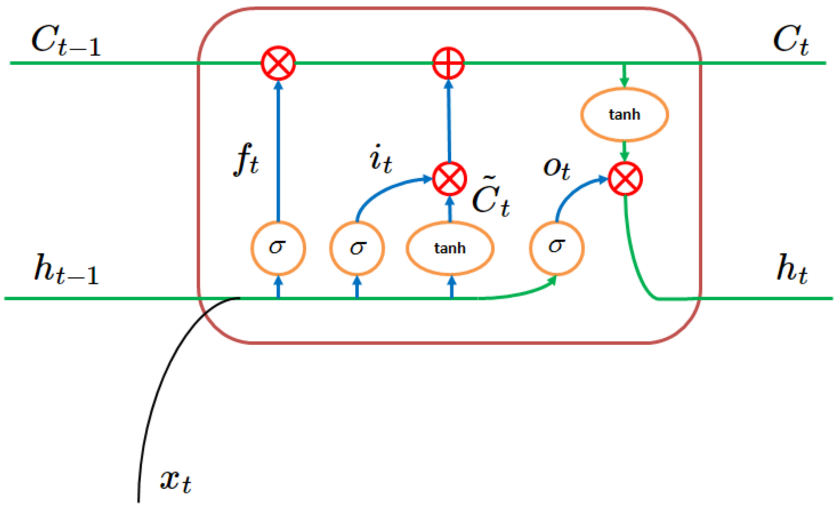


Figure 4: *LSTM* cell

It contains 4 gates (*F* – forget, *I* – input, *C'* – cell, *O* - output) which decide how information is passed through the cell. A single LSTM has hidden state (h) and cell state (c). These states store all the logic and memory needed for prediction.

- Forget gate (*F*) decides what information will be thrown away from the cell state. Its equation is $f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$ and depends on previous hidden state h_t and the current input x_t. When f_t is small (close to 0) it will keep this information, otherwise it will reject.
- Input gate (*I*) is given as $i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$ and decides whether the input will be propagated through the cell and affect the cell hidden state.
- Cell gate (*C'*) is defined as $C_t' = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$ and this represents possible new cell values.
  Using last two gates new cell state is specified as $C_t = f_t \cdot C_{t-1} + i_t \cdot C_t'$
- Output gate (*O*) decides what is going to be on the output. First, this gate is computed as $O_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o)$ and it decides how much of a cell state will affect the new hidden state. Therefore, new hidden state is calculated as $h_t = O_t \cdot \tanh(C_t)$.

In this implementation, 3 *LSTM* cells are stacked in a row and their hidden state size is specified with parameter *lstm_size*. This is shown on the Figure 5.

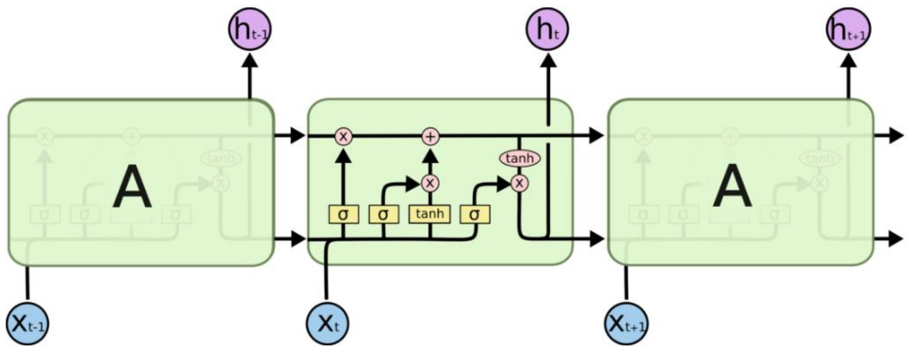


Figure 5: Stacked *LSTM* cells

Each *LSTM* cell has dropout wrapper which prevents them from overfitting, as well as L2 regularization. After last cell, simple multiplication of its hidden layer and matrix *W* is used to give output with proper dimension. Loss is calculated as a mean squared error between targets and predicted values and during training both train and validation loss is observed so to stop before overfitting occurred. The original *CLOSE* values were split in training (90%) and validation (10%) dataset.

While tweaking the parameters these were changed: $input\_length \in [1, 3, 5], lstm\_size \in [50, 250, 400], num\_epochs \in [10, 30, 50]$. Parameter $learning\_rate$ has 5e-3 initial value and this is decayed by the factor 0.7 after every 2 epochs. Decreasing of $learning\_rate$ through epochs is shown on the Figure 6.

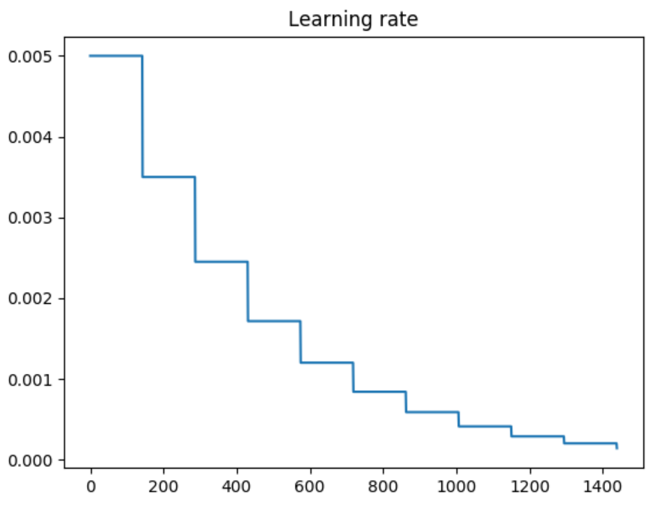


Figure 6: $learning\_rate$ decrasing

Inputs to the network contains last num_time_steps of grouped values and targets are the next value after these.

Trading strategy

I have used very simple trading strategy. Whenever the network predicts that in the future price is going up, I would buy some number of stocks if there is enough money. When the network predicts that price is going down, I would sell all the bought stocks. This is represented with the next equation

$$strategy = \begin{cases} buy,\ if\ y_{p(t+n)} > y_t \\ sell\ all\ stocks,\ if\ y_{p(t+n)} \leq y_t \end{cases},$$

where y_p represents predicted prices and n is number of days for which to look ahead in the future. This parameter n is changed in range $[0, 5]$. Amount of money when buying stock is fixed to 5000. Even though this value could be variable depending on the predicted value, for simplicity it is taken as a constant.

Results

The network with the biggest profit has parameters:

(This doesn't mean it is the most optimal network, but for the networks that are tried gives the best results)

Parameter $input\_length$ does not affect the final result significantly. This is mostly used for averaging prices as part of data visualization.

Parameter $lstm\_size$ is quite important so the network can have enough capacity to learn from data. If this parameter is to low, then underfitting could happen. Also, if this parameter is to high, overfitting is possible. In this implementation the best results are for the $lstm\_size$ = 250.

Parameter num_epochs is quite important and overfitting occurs mostly because of its too big value. The one used in the final model is num_epochs = 30.

Parameter $learning\_rate$ is constant for all models and it is as described in previous chapter.

On Figure 7 train and validation loss are given.

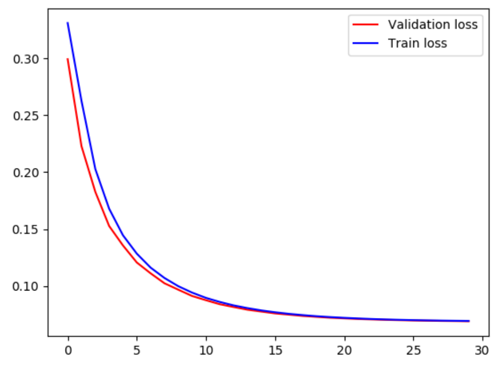


Figure 7: Train and validation loss

On Figure 8 prediction on the validation data is given.

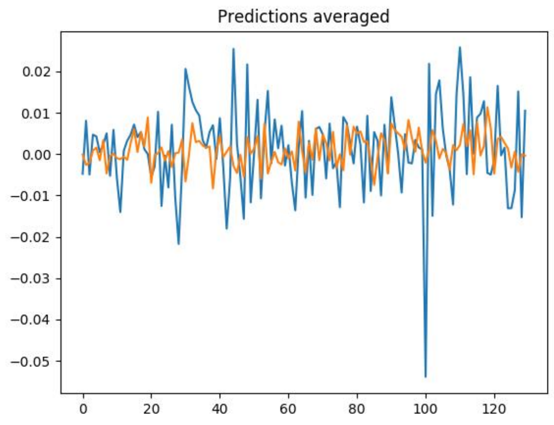


Figure 8: Averaged predictions on validation data (blue color real values, orange predicted)

On Figure 9 prediction on the test data from 01st Jan 2016 to 7th Sept 2018 is shown.

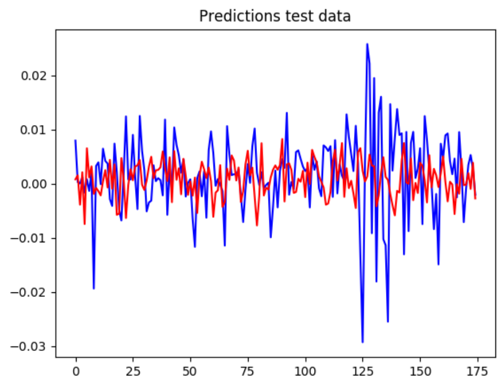


Figure 9: Averaged predictions on test data (blue color real values, red predicted)

Total profit on the train data + validation data: 750 $

Total profit on the validation data: -1073 $

Total profit on the test data: -1192 $

For all these n = 2 which means how many steps ahead to look for prediction.

Conclusion

This project showed that approach with *LSTM* recurrent neural networks could have bigger impact in predicting stock prices. However, since trading data is not only dependent on its historical value, but instead on many other factors from real world, these results are not so good. The right data for this kind of prediction should contain newsletter articles that reflects happening in the world, social and policy aspects, sentiment of the environment and many other so it can give accurate results. Also, there are needed some improvements in the architecture of these networks so it can detect long-term relations in time-series data. This could be done with attention models which focus to the most important part of the input. Moreover, there are often seen some approaches where reinforcement learning is combined with RNNs so that model can constantly learn from new data and not just to be based on historical data.

To conclude with, LSTM showed that they are the right approach, but it needs enriched data and changed architecture so it can become quite efficient. Anyway, there are plenty of approaches and techniques that should be investigated.