Machine Learning Engineer

Nanodegree

Capstone Project

Juan Roberto Honorato

March 6th, 2018

Index

I. Definition 3

Project Overview 3

Problem Statement 5

Metrics 6

II. Analysis 7

Data Exploration 7

Exploratory Visualization 8

Algorithms and Techniques 11

Benchmark 13

III. Methodology 13

Data Preprocessing 13

Implementation 14

Refinement 16

IV. Results 17

Model Evaluation and Validation 17

Justification 18

V. Conclusion 19

Visualization 19

Reflection 20

Improvement 21

References 23

Appendix 24

Metatrader code to export indicators to a csv file 24

I. Definition

Project Overview

Forex trading, or more formally the foreign exchange market, is a global and decentralized market for the trading of currencies. This market has a long history and can be traced back to ancient times[1]. But modern Forex trading is said to have started in a free and open manner since 1973 [1]. Today, the foreign exchange market is the most liquid financial market in the world [1] and accounted for $5.09 trillion per day in April 2016 [1].

Autotrading, on the other hand, originates at the emergence of online retail trading since about 1999 [1]. As its name suggests, autotrading refers to algorithms that either operate the market on their own or signal for a human to manually take action. The possibility of entirely autonomous systems operating with real money, and (hopefully) *making* real money is a fascinating prospect for me, and this is why I have chosen this idea to be my capstone project.

Picture 1: Typical Forex trading chart with japanese candlesticks describing the price in the upper band and a pair of stochastic indicators in the lower band.

Certain specific terminology will be necessary to describe the work done in the project. The following is a very brief glossary of the most essential terms to be used, mostly taken from [2] and [3]:

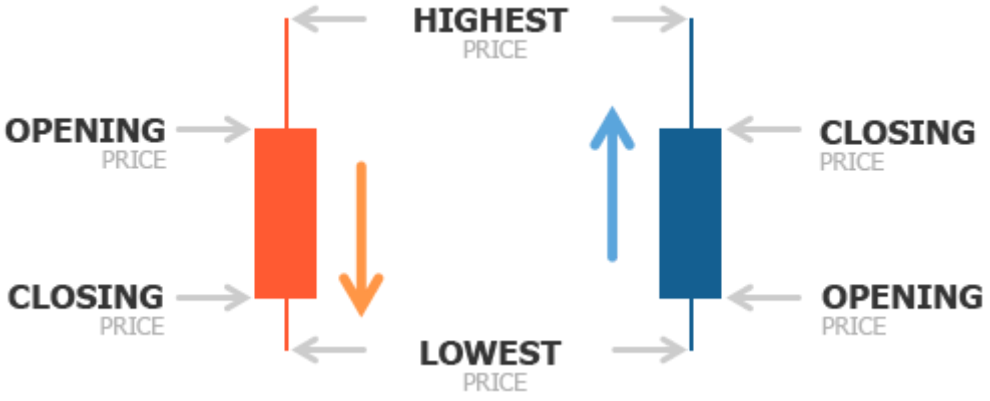
Currency pair: Currency pairs are when two types of money are traded for one another. One can trade nearly any kind of currency against nearly any other kind, provided someone in the Forex market has it available. For example, one can trade US dollars versus Japanese yen, or Euros versus Great British pounds. In this project the EURO versus USD currency pair is used.

Spread: The spread is the difference between the bid or buying price for a currency and the ask or selling price for it. An individual trading currencies has to use a broker, and every broker attaches a spread to the currency they trade, which is where they make their profit.

Pip: A pip is a measurement of how far the price has moved. It is an acronym for the phrase “percentage in point”. If you think of the exchange rate of the pound and the dollar (GBP/USD), you might think of it as say, 1.57, where only two units follow the decimal point. However, in the foreign exchange markets, this is broken down even further and we observe the price as 1.5700. The last number – the last 0 – is the pip. If the value of that currency pair moves from 1.5700 to 1.5701, it has moved by a single “pip”. Pips are how traders generally measure their profit.

Opening/closing a position: Buying and selling is sometimes called entering a position. It is the same as “entering the market”. When the trader exits the market, they are said to have closed their position.

Entry: An entry is when a trader decides to open a position, either by buying or selling a financial instrument.



Picture 2: Visual explanation of the japanese candlestick.

Exit: An exit is when a trader decides to close their open position in the market for either a profit or a loss.

Long position: When we refer to something being long, we think of it as going up. In forex trading, if the trader believed that the currency pair is going to rise, and they bought that pair, they would be described as "entering a long position".

Short selling: If you think that the price will go down and you have not entered the market, you can also click the sell button and you will enter into what is called a “short sell” position. You have, in fact, sold something you do not own, to buy it back at a different price. So if you short sell an asset and the price goes down, you make money; if the price goes up, you lose money.

Problem Statement

The goal for the project is not to develop a fully autonomous autotrader agent, but rather, have a predictor that can support manual decisions made by a human being. More specifically, I will train a machine learning model called LSTM network to predict whether the next candle will close higher or lower than the previous one, and how strong the movement will be. Thus, we will predict four classes: way down, shy down, shy up and way up. We will feed the model all the information that the japanese candlestick provides along with some previously calculated indicators. For the output we will ask of the model the 4 classes previously described. This problem is, by extension, a classification problem.

Finally, we will try a very simple trading strategy to see if our model could indeed make money or not, aside from any loss or accuracies that our model’s metrics could report. We will simulate, with a series of simple rules, the operations of a human using what the model predicts, and by taking into account the spread, we will know whether the combination of this strategy paired with the accuracy of our model, would actually be profitable in the forex market.

In order to achieve this goal, we will need historical data from a currency pair. I chose the EURUSD currency pair because of the high volume of trade that this pair receives and because I found another study that uses an SVM model with which we can compare our results against. The data collected also includes a series of indicators. This are calculations over the raw price data that, when plotted, are intended to give enough information to the human operator so it can make profitable

decisions. A crude example would be “when this two lines cross, and this other block is red, enter short”.

Metrics

The metrics for the model are very important for every problem to be solved, and in this domain it is no different. For this problem I’m going to measure the accuracy of predicting the four classes. I’m solving this problem as a classification task instead of a regression one because I can extract the certainty of the model for its prediction, and this is valuable information for the human operator. As a regression task the model would predict a price for the currency pair without a degree a certainty for the prediction.

As stated in [4] and [5], in a binary classification task (of predicting only up or down movement) it is considered a good result to achieve close to 60% accuracy in the predictions. There are papers were higher accuracies -higher than 70%- are reported, such as [6], but those are typically reported in the stock market rather than the forex market. Arguably, I think the results can be compared within these two markets.

The accuracy metric is used because it is the most widely used in the literature found, so the comparisons are easier to make. Also, an F1 score is not chosen because for this particular project it would arguably not give a better metric over the plain accuracy, this is because the classes are highly balanced and thus we will not have a deceptively high accuracy score.

II. Analysis

Data Exploration

The data used is exported from the Metatrader 4 software. The script used to export the data is shown in the first appendix section. The data consists in 438,676 data points of the EURUSD index given in 5 minute japanese candlesticks. A sample of the data can be seen in table 1.

time	close	open	high	low	atr	cci	macd	rsi	stoch_stock	wpr
2015-12-08 18:55:00	1.09	1.09	1.09	1.09	0.00048	-1.0	-0.00015	45.0	30.0	-52.0
2014-04-17 13:00:00	1.39	1.39	1.39	1.39	0.00036	-83.0	0.00014	50.0	27.0	-74.0
2014-06-26 23:35:00	1.36	1.36	1.36	1.36	0.00011	109.0	0.00006	60.0	86.0	-9.0
2017-06-22 06:50:00	1.12	1.12	1.12	1.12	0.00016	-30.0	0.00008	52.0	20.0	-53.0
2013-09-19 21:45:00	1.35	1.35	1.35	1.35	0.00030	70.0	0.00014	56.0	38.0	-25.0

Table 1: Some example data points of the gathered data.

As can be seen, there are 11 columns, of which the *time* column acts as an ordered index, while the immediately next *close* column will be the number that we will use to calculate the correct class for our model to predict.

The columns, aside from *close*, *open*, *high* and *low* are calculated from the flow of the price data and are called *indicators*. This *indicators* could be custom arbitrary functions that the trader has somehow in his (virtual) hands, but in this case we are using 6 indicators that come as standard functions in Metatrader 4, and hence everyone has access to them. They do have some parameters to be set, and those can be seen in the code used to export the values. The code is in the *Appendix* section 1.

Using the handy pandas *describe* method, we can easily obtain important statistics about our data. This statistics can be seen in the table 2 below. From the table we can see that there are no clear outliers or errors in our data, and once we plot the data -shown in picture 3 in the “Exploratory Visualization” section below- we can see that the graphs and ranges look like non abnormal data distributions, except for the *atr* and *macd* features, which we will further explore in the next section. Also, note that there are no missing values.

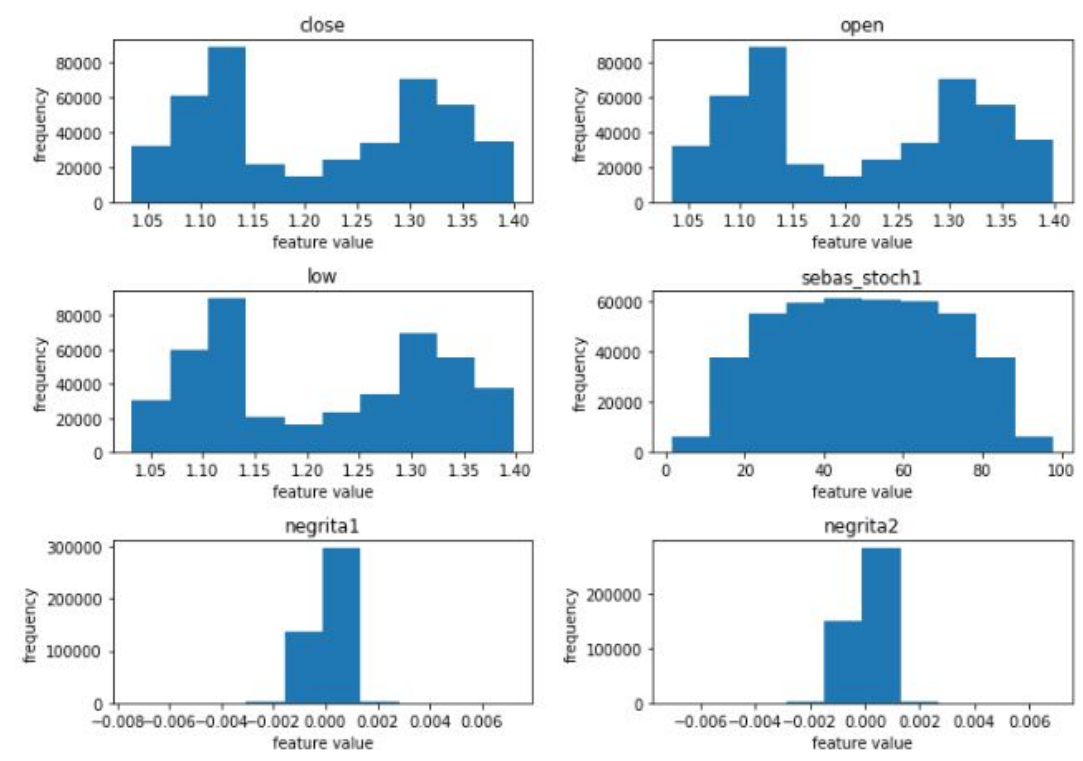
	close	open	high	low
count	438676.000000	438676.000000	438676.000000	438676.000000
mean	1.214805	1.214806	1.215025	1.214591
std	0.108290	0.108290	0.108275	0.108305
min	1.030000	1.030000	1.040000	1.030000
25%	1.110000	1.110000	1.120000	1.110000
50%	1.220000	1.220000	1.220000	1.220000
75%	1.320000	1.320000	1.320000	1.320000
max	1.400000	1.400000	1.400000	1.400000

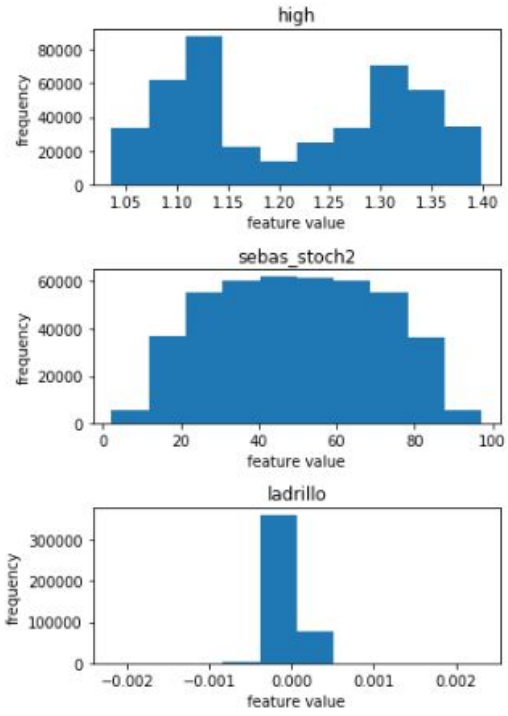
	atr	cci	macd	rsi	stoch_stock	wpr
count	438676.000000	438676.000000	438676.000000	438676.000000	438676.000000	438676.000000
mean	0.000463	-0.054915	-0.000002	49.965209	49.737740	-50.329487
std	0.000291	106.169792	0.000432	11.033709	24.832979	28.812995
min	0.000000	-467.000000	-0.007440	0.000000	0.000000	-100.000000
25%	0.000270	-80.000000	-0.000170	43.000000	28.000000	-76.000000
50%	0.000390	-1.000000	0.000000	50.000000	50.000000	-50.000000
75%	0.000590	80.000000	0.000170	57.000000	71.000000	-25.000000
max	0.006050	467.000000	0.007210	96.000000	100.000000	0.000000

Table 2: Statistics of our data

Exploratory Visualization

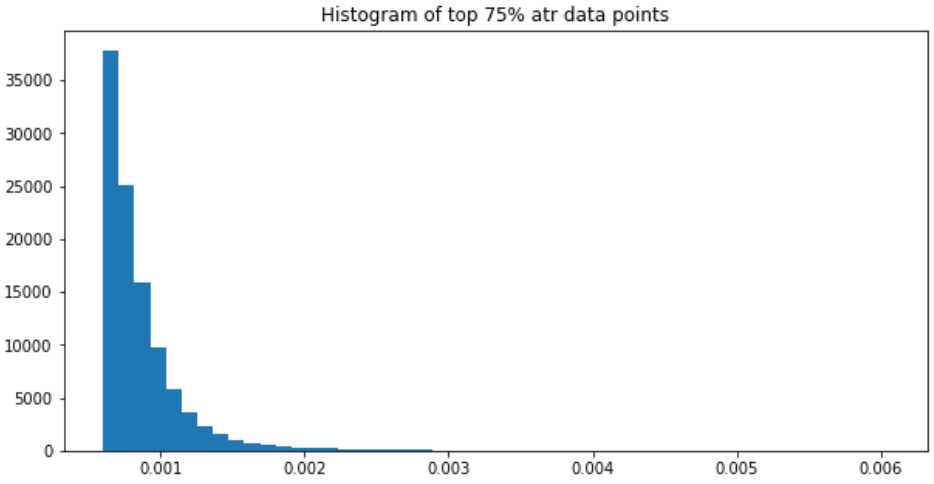
It is important to understand if for some reason our data has outliers, and how frequent they are. If they are frequent and too far off, they could introduce important noise that could really damage our model's performance. In order to see this, a histogram plot is really helpful as a first approximation.



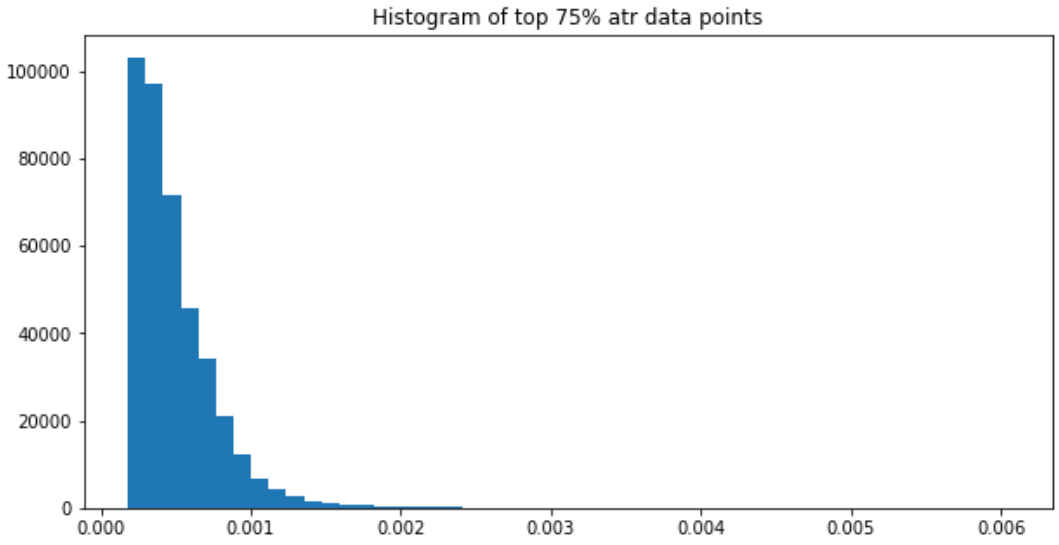


Picture 3: Distributions of data columns as histograms

From picture 3 we can see that all our feature columns are nicely distributed, except for the *atr* and *macd* columns. In order to see if this is normal behaviour or an outliers case, we plot these specific features but only taking into account the top 75% of the data points in order to be able to really see their distribution, as can be seen in pictures 4 and 5.



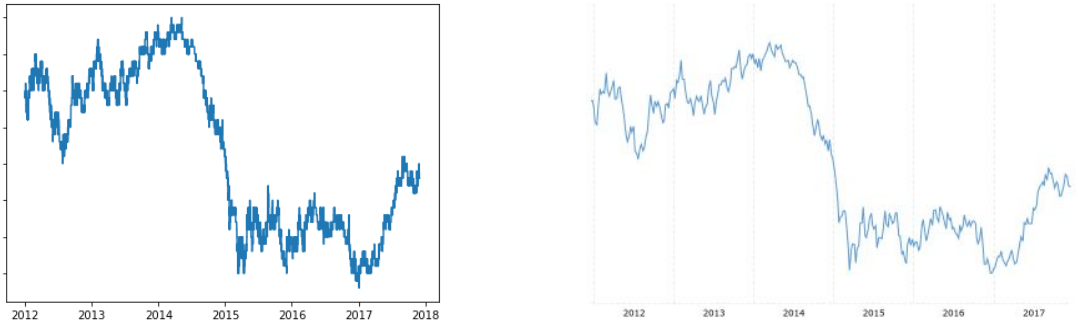
Picture 4: Distributions of top *atr* data points



Picture 5: Distributions of top *macd* data points

We can see that although the topmost points lay far away, the distribution does look like those points are not an error in the data extraction or the processing script. Thus, we will leave the data unchanged in this sense.

Another interesting and very important plot is to see if the candles' prices match what we would typically find in online data sources. For this purpose, in picture 6 we can compare the plot of the close price of our data versus the same plot for the same period but from an external source.

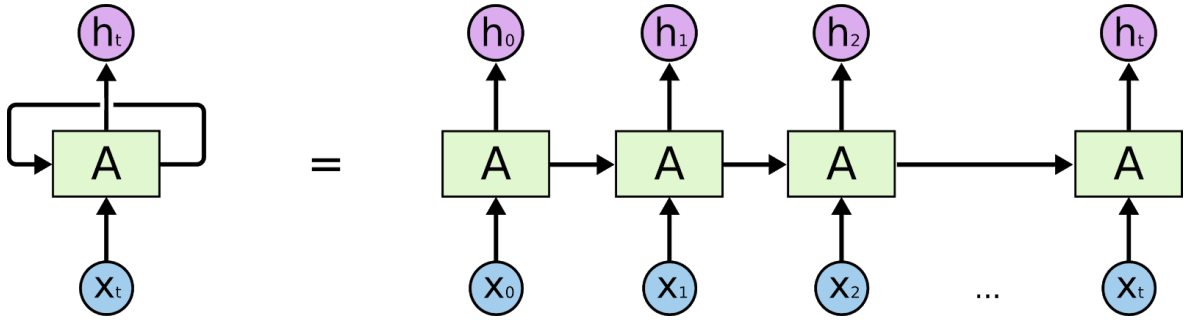


Picture 6: Plot of our data on the left. Plot of the same period extracted online on the right [source: macrotrends]

The plots do match, and considering the previous exploration and analysis, we can rest assured with the quality of the data that we are using.

Algorithms and Techniques

The data that we are going to use is clearly a time series, where the order of our inputs is very important. Also given the expected time correlation from Forex index data, it is important that our model takes into account the immediately previous data as context for the current prediction. A model that easily takes into account all this is any form of recurrent neural network, thus we are going to make use of the LSTM network. This network architecture solves the vanishing gradient problem [7], and also handles in a much better way long term memory than a vanilla recurrent neural network [8].



Picture 7: Recurrent Neural Network unrolled [Source: Colha's blog [9]]

We will feed to this network a window of candles, asking it to predict the movement of the next candle. The network will keep an internal state, a dense representation of how the market has moved recently. The hope is that this information will be enough to have a good result in predicting this future movement.

We will feed to the network the entire information of the candles and the indicators of the defined window previous to the target candle to be predicted. The prediction will be based in the following value:

```
relation = (next_candle - last_candle)/last_candle
percentage = 0.03/100
if relation > percentage:
	move = 3
elif relation >= 0:
	move = 2
elif relation > -percentage:
	move = 1
else:
	move = 0
```

If close price is less than 0.03% of previous close price, we will label it as a shy movement. This threshold was selected by trial and error. It gives, for the first 50,000 candles, a distribution of [6.402, 18.188, 19.202, 6.208] which seems about right:

Machine label	3	2	1	0
Human label	Way down	Shy down	Shy up	Way up
Distribution	12.8%	36.4%	38.4%	12.4%

Table 3: Distribution of labels with threshold value of 0.03%

This are the labels that the LSTM network will be in charge of predicting. When a candle closes, they represent the direction and the distance that the currency pair price will have in 5 minutes from when the prediction is done.

Benchmark

To compare the performance of the model, I'm going to use as a benchmark the results provided in [4]. In this blog post, the author describes how to achieve a "54% accuracy for our short trades and an accuracy of 50% for our long trades" using an SVM model. The author uses only 2 classes compared to the 4 classes we are using, but with a little post processing of the model's predictions we will be able to compare the results fairly enough. Another important point to notice is that these benchmark is also using the EURUSD price index, but the candlesticks are of 1 hour intervals.

III. Methodology

Data Preprocessing

In the previous section and through data exploration we found that there were no clear outliers, so we did not remove any data points. But there is another important step to take before we feed the data to the neural network model, that is normalizing the data.

As stated in [10], normalizing (also known as rescaling) the data when using neural network models can bring benefits with faster convergence and also help the model to not converge to a local minima.

In order to normalize the data we first compute the maximum and minimum values for each column, giving us the following values:


```
{
  "rsi": {
    "feature_max_value": 95.82226461526429,
    "feature_min_value": 0
  },
  "stoch_stock": {
    "feature_max_value": 100,
    "feature_min_value": 0
  },
  "wpr": {
    "feature_max_value": 0,
    "feature_min_value": -100
  },
  "macd": {
    "feature_max_value": 0.007210471302582233,
    "feature_min_value": -0.0074380833133858015
  },
  "atr": {
    "feature_max_value": 0.006052857142857161,
    "feature_min_value": 0
  },
  "cci": {
    "feature_max_value": 466.66666668293504,
    "feature_min_value": -466.6666666745045
  }
}
```


After we compute this values, we transform the data using the following formula:

```
normalized_value = (not_normalized_value - feature_min_value) / (feature_max_value
- feature_min_value)
```

Having our data ready, we proceed with the implementation itself.

Implementation

To build the model, I used the Keras library [11] with the Tensorflow [12] backend. Various different architectures were tested, mainly varying the number of layers and the number of neurons in each layer. The first implementation consisted in a single LSTM network with 16 hidden units and using a batch size of 32.

The most difficult part about the implementation was actually feeding the data in the right way so it would make sense as a temporal sequence and also have the right structure and dimensions for the model to correctly compile and run. The construction of the model itself is actually painless when using Keras instead of vanilla Tensorflow because it handles all the dimension matching between layers automatically and the default parameters are also very good ones.

The final implementation was given by the following code:

```
# -*- coding: utf-8 -*-
from feed_data import Data_feed
from keras.models import Sequential
from keras.layers import Dense, LSTM, Flatten, Dropout
from keras.callbacks import TensorBoard, ModelCheckpoint

# data
window = 64
batch_size = 768
train_data = Data_feed(filepath='data/EURUSD_M5_438K_Stock_Preprocessed.csv',
window=window, percentage=0.9, dataset='train', batch_size=batch_size)
test_data = Data_feed(filepath='data/EURUSD_M5_438K_Stock_Preprocessed.csv',
window=window, percentage=0.1, dataset='test', batch_size=batch_size)

# design network
model = Sequential()
model.add(LSTM(64, batch_input_shape=(batch_size, window, 10)))
model.add(Dropout(0.3))
model.add(Dense(4, activation='softmax'))
```

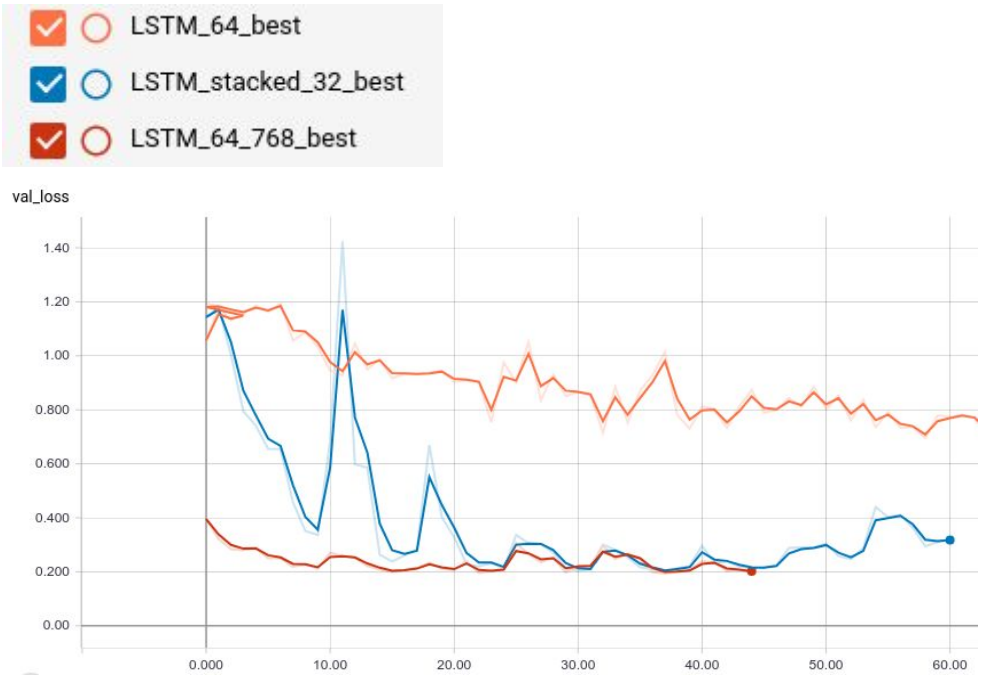
```
model.compile(loss='categorical_crossentropy', optimizer='adam',
              metrics=['acc'])

# callbacks
model_name = 'LSTM_64_512_best'
tensorboard = TensorBoard(log_dir='data/tensorboard/' + model_name,
write_graph=True)
checkpoint = ModelCheckpoint(model_name + '.h5', monitor='val_loss',
save_best_only=True)

# fit network
history = model.fit_generator(
    generator=train_data.generate(),
    steps_per_epoch=500,
    epochs=400,
    callbacks=[tensorboard, checkpoint],
    validation_data=test_data.generate(),
    validation_steps=500
)
```

The window parameter determines the amount of candles that the network will receive as history a sequence before giving its prediction. The batch size was a key factor into the convergence speed (as shown in the next section). The dropout [13] layer helps to prevent the network from overfitting by randomly turning off some neurons. Also, the model checkpoint callback in Keras helps to easily save periodically the trained models, and in this case we are only saving the model with the lowest error in the validation set.

Notice that I'm using the first 90% of the dataset as the train set and the remaining 10% as the test set.

Refinement

Several different parameter configurations were used to try and find the best configuration given the LSTM model. The error rate of the best configurations can be seen in picture 8. The very first solution was to use a single LSTM network with 64 hidden units and a batch size of 32. This turned out to give results similar to the ones obtained by the other best performing metaparameters, but the training process took much longer to reach its minimum loss error, as can be seen in picture 8.



Picture 8: Validation loss of best performing models.

The different configurations tried were as follows:

- Batch size: 32, 64, 256, 512, 768.
- Neurons in hidden layer: 32, 64, 128.
- Number of stacked LSTM layers: 1, 2, 3.

The loss errors obtained were rather similar between the more powerful configurations. Finally, the network used later on was the “LSTM_64_768_best” network, which consists in a single LSTM network with 64 hidden units and a batch size of 768. The optimizer used was Adam, with the default parameters that come in the Keras library.

Having decided on a model, different configurations were used to obtain the best monetization strategy for that model. The list of parameters used were the following:

- `stop_loss = [5, 10, 15, 20, 25]`
- `take_profit = [10, 15, 20, 25, 30]`
- `certainty = [.50, .55, .60, .65]`

IV. Results

Model Evaluation and Validation

The model used is a standard LSTM network, this network was derived to solve problems related to earlier implementations of recurrent neural networks that try to solve specific problems related to sequences of data. Thus, the model seems ideal to the problem at hand, were temporal dependency of the data is essential to model successfully.

The hyperparameters used were derived with trial and error. Different configurations were tested, changing one parameter at a time, and if the solution improved, I further tweaked the parameter until I found the optimal value. This at least was the theory and what I tried to put in practice, but as was stated before, the loss scores were mostly the same across the board, with only minimal gains from the most powerful variations.

To test that the model was robust, two changes were introduced: using a different currency pair dataset and forcing small perturbations in the input data.

Different currency pair: The initial model was trained using the EURUSD currency pair. To test the selected architecture, a model with the same parameters was trained using the GBPCAD currency pair. The results were almost identical in terms of loss error, having a minimum of 0.2027 as the best achieved score with the validation set.

Small perturbations: The input data with the EURUSD dataset was slightly perturbed to address the model's robustness to changes in the data. Every 3 timestamps, the numerical values for that candle were artificially multiplied by a random value between 0.9 and 1.1. The accuracy obtained by the model suffered a 1.03% degradation.

Given the two results just described, we can confidently state that the model is robust enough, and that the results of the model can be trusted. To be seen is the profitability of the model, which may have no resemblance whatsoever to this results.

Justification

The benchmark that we established to compare against was a blog post describing how to achieve, using an SVM, a 54% accuracy with the long positions and a 50% accuracy with the short ones.The author also claims to obtain an average of a 53% accuracy overall in this binary classification task.

With our LSTM model, to obtain a binary classification accuracy out of our 4 classes, we first have to transform our predictions with the following rule:

```
pred = model.predict(y)
if np.argmax(pred) > 1:
        y_ = [0, 1]
else:
        y_ = [1, 0]
```

Using this simple post processing rule we can compute our accuracy score, obtaining a 54% accuracy overall in the final model. This was not a significant bump from the initial trials, where the accuracy was around a 52%.

The result obtained is barely better than our benchmark, and statistically speaking, the advantage is possibly not even statistically significant. I would be inclined to say that the performance of the models are on par, neither seems to have a better result over the other.

But given that the results are comparable to our benchmark, and adding to this the fact that I was not able to squeeze any more accuracy out of the LSTM network no matter how complex I made the model, it seems that this is close to what the data can give, and maybe other sources of data could help to bump the precision the way that I would like to. For example, there seems to be successful attempts at making use of social media data to operate in real time the stock market [14].

Furthermore, and being the Forex market a potentially very lucrative opportunity, I tested a very simple strategy to see if our model could make money. It is worth noting that this is only an extremely simple strategy out of potentially millions of different ones, but this particular one makes sense given what the model predicts.

What the strategy does is open a position when the model predicts a strong movement and close it when the model predicts a movement in the opposite direction, be it strong or weak.

Several simulations were run using the test data of the model to find the optimal parameters with respect to the stop loss, take profit and certainty. This last certainty parameter refers to the certainty of the model with the prediction of the strong movement before we open the position. The best performing strategy was achieved using the following parameters:

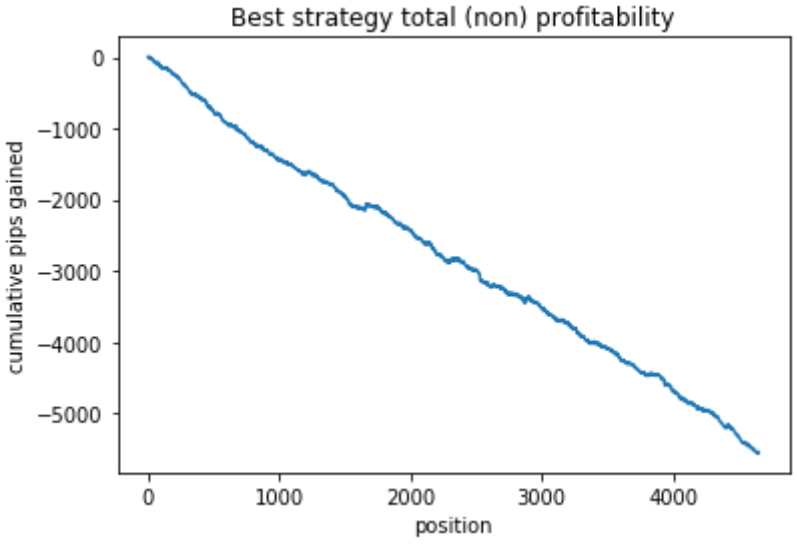
- `stop_loss = 15`
- `take_profit = 30`
- `certainty = .55`

The best simulation (sadly) achieves a negative result, this is because we have to subtract the spread. The mean pips per position achieved is of -1.65 before we add a spread of near 1.0 pips per position which can be typically found in the EURUSD index, so without the spread it means we would still be marginally below zero in average.

V. Conclusion

Visualization

The strategy can be best summarised with a plot of the cumulative gain (loss in this case) of the pips over the course of the simulation. The curve shown in picture 8 leaves no room for doubt, it is very telling.



Picture 9: Cumulative total pips with positions in best performing trading strategy.

There's really no room for debate after seeing this picture. The model may lead to a very close to break even trading strategy, but in the long term it is a complete loss in that same sense.

Reflection

The problem of predicting the movements of the Forex market is an obvious choice when thinking about applying new machine learning techniques, especially those that relate to the processing of sequences of data. This field seems to lack the open nature of other areas of machine learning research, and understandably so. If anyone found an algorithm that could truly make money in this dynamic scenario, it will most probably keep it a secret, so as to not spoil the advantage it may have by letting too many people exploit whatever the model may have found. By doing so, the market would quickly adapt and the advantage would cease to exist. This is why I didn't thought a vanilla LSTM network would truly make money in today's market. Even so, it was truly a worthy experiment which helped me to learn a lot the algorithm behind it.

In the present project, the goal of using an LSTM network to predict the Forex market was not successful, even though the network did learn what it could from the data, as evidenced by the convergence of the loss error in the validation dataset. By extension, the attempt at deriving a simple and profitable trading strategy with the

model was also not successful, even though at it best it was very close to a break even, which is by no means good enough.

The project was not particularly challenging, but even then, it consisted in a lot of steps regarding the handling of data. First, the data needed to be extracted by custom scripts in a language of very niche use, then this data had to be preprocessed so it could be safely fed to a model. A pipeline to feed the data to the model had to be written, and finally the trading strategy had to make use of this data as well as the model's predictions to calculate its profit. As can be seen, almost all of the work, and the challenge of the project, resided in the data flow realm, as a lot of data scientists will testify online. In this case the model definition was actually very simple, and the training did not take too long to complete (at most a night of training in my workstation).

I think the project was a very good example of a complete and end to end solution of a real world problem, and even though the final solution was not what we would call a success, the exercise by itself did give me confidence to tackle serious problems, and understanding much better what are the real challenges in some machine learning problems. For example, now I understand that most problems that companies face that could be easily solved with machine learning can be solved relatively quickly, given that there is enough data for the models to learn. The challenge seems to lie in the fact that most companies don't have managers or developers that know how to take advantage of these techniques and thus fail to envision their use in the problems where they would perfectly fit.

Improvement

As an improvement I would consider first and foremost, changing the paradigm used. Instead of taking the approach of building a classification or regression model to predict the future movement of the market and then acting accordingly with self derived rules, I think a far more valuable approach would be to use deep reinforcement learning so a model can try to optimize the whole process with a far wider range of action than what I could manage here. Recently there have been great advances in the field, and even though the famous Alpha Zero algorithm needs a set of clear rules about the game being played [15], there are other algorithms that could be tested like the family of Deep Q Network algorithm, as implemented here [16].

Regarding the path of predicting the market and acting based on that, I think this approach would better serve the purpose of giving support to a human professional trader rather than operating on its own, but this would probably require a degree of

transparency to the model’s prediction that neural networks notably lack at the moment. Improvements in this regard could consist of classifying a candle as a pivot point or predicting a head and shoulder movement.

References

[1.] Unknown author. (Unknown year) **Foreign exchange market,** *Unknown source* [link]
[2.] Elam, Yohay. (2011) **Forex Trading - 8 Basic Must Know Terms,** *Unknown source* [link]
[3.] Tradimo Interactive ApS. (2017) **What are the most important terms in Forex trading?,** *Unknown source* [link]
[4.] Paradkar, Milind. (2016) **Machine Learning and Its Application in Forex Markets [WORKING MODEL],** *Unknown source* [link]
[5.] Honchar, Alex. (2017) **Neural networks for algorithmic trading. Simple time series forecasting,** *Unknown source* [link]
[6.] Kara, Yakup and Boyacioglu, Melek Acar and Baykan,.... (2011) **Predicting direction of stock price index movement using artificial neural networks and support vector machines: The sample of the Istanbul Stock Exchange,** *Expert Systems With Applications* [link]
[7.] Singh, Anish. (2017) **The Vanishing Gradient Problem – Anish Singh Walia – Medium,** *Unknown source* [link]
[8.] Hochreiter, Sepp and Schmidhuber, Jürgen. (1997) **Long Short-Term Memory,** *Neural Computation* [link]
[9.] Olah, Chris. (Unknown year) **Understanding LSTM Networks -- colah's blog,** *Unknown source* [link]
[10.] Various authors. (2011) **normalization - Why do we have to normalize the input for an artificial neural network? - Stack Overflow,** *Unknown source* [link]
[11.] Chollet, Francois. (Unknown year) **Keras Documentation,** *Unknown source* [link]
[12.] Various authors. (Unknown year) **TensorFlow,** *Unknown source* [link]
[13.] Srivastava, Nitish and Hinton, Geoffrey and Krizhe.... (2014) **Dropout: A Simple Way to Prevent Neural Networks from Overfitting,** *Journal of Machine Learning Research* [link]
[14.] Tsui, Derek. (Unknown year) **Predicting Stock Price Movement Using Social Media Analysis,** *Unknown source* [link]
[15.] Silver, David and Schrittwieser, Julian and Simony.... (2017) **Mastering the game of Go without human knowledge,** *Nature* [link]
[16.] Kim, Kh. (2016) **Stock Market Reinforcement Learning,** *Unknown source* [link]

Appendix

Metatrader code to export indicators to a csv file

```
#property strict
#property indicator_separate_window

extern string file_name = "Indi2csv.csv";


int fileh =-1;
int lasterror;

//+------------------------------------------------------------------+

int init()
  {

   IndicatorShortName("Indicators2CSV");

   fileh = FileOpen(file_name,FILE_CSV|FILE_WRITE,',');
   if(fileh<1)
   {
      lasterror = GetLastError();
      Print("Error updating file: ",lasterror);
      return(false);
   }

   // file header - need to be the identifiers of the indicators to be exported


FileWrite(fileh,"time","close","open","high","low","atr","cci","macd","rsi","stoch_stock","
wpr");


   return(0);
  }

//+------------------------------------------------------------------+

int deinit()
  {
      if(fileh>0)
      {
         FileClose(fileh);
      }

   return(0);
```

```
  }

//+------------------------------------------------------------------+

int start()
  {
   int barcount = IndicatorCounted();
   if (barcount<0) return(-1);
   if (barcount>0) barcount--;

   int barind=Bars-barcount-1;

      while(barind>1)
      {
         ExportIndiData(barind);
         barind--;
      }

   return(0);

  }
//+------------------------------------------------------------------+

void ExportIndiData(int barind)
{
   datetime t = Time[barind];
   string inditime =
      StringConcatenate(TimeYear(t)+"_"+
                        TimeMonth(t)+"_"+
                        TimeDay(t)+"_"+
                        TimeHour(t)+"_"+
                        TimeMinute(t));

   // add indicators at will (do not forget to update line 31!
   FileWrite(fileh,
         inditime,
                     Close[barind],
                     Open[barind],
                     High[barind],
                     Low[barind],
                     iATR(Symbol(),0,14,barind),
                     iCCI(Symbol(),0,14,PRICE_CLOSE,barind),
                     iMACD(Symbol(),0,12,26,9,PRICE_CLOSE,0,barind),
                     iRSI(Symbol(),0,14,PRICE_CLOSE,barind),
                     iStochastic(Symbol(),0,5,3,3,MODE_EMA,0,0,barind),
                     iWPR(Symbol(),0,14,barind)
                     );

}

//+------------------------------------------------------------------+
```