POZNAN UNIVERSITY OF TECHNOLOGY

Denis Sheremetov

Using recurrent neural network LSTMs for the analysis of financial time series data

Master's Thesis

Supervisor: dr hab. inż. Mikołaj Morzy, prof. nadzw.

Poznań, 2018

Contents

Abstract

The use of recurrent and deep neural networks is becoming very popular for solving many problems that were considered complex or unresolvable for a long time.

A huge number of problems in the field of machine learning is associated with the analysis of the change of some events in time (time series data) and prediction of events. A well-known example of time series data are stock prices which represent time-dependent variability of financial markets.

Predicting and building an effective trading strategy for trading on stock markets is still an unresolved problem. This work is an attempt to solve it by applying modern methods and approaches to working with time series data to solve it.

The main motivation and idea of the research is to study the actual approaches and methods of machine learning for working with financial time series data, designing and testing a model capable of making meaningful decisions in the stock market, building a profitable strategy of a virtual trader. In a broader sense, we are interested in building solution which will enable the user to formulate the problem for the machine, and the machine, using it's computing power, will be able to offer the solution.

Chapter 1

Introduction

1.1 Problem statement

The financial market is a system of relations in the process of which economic benefits are exchanged using money as an intermediary. The participant submit applications for the purchase and sale of assets and at the moment when demand meets the offer, a transaction takes place.

The foreign exchange market (Forex, FX, or currency market) is a global decentralized financial market for currency pairs trading. The currency trading market is interesting for our research for a number of reasons:

- The huge volume of trading of one of the most liquid assets.
- Continuous trading 24 hours a day except for weekends.
- A lot of external factors affecting the exchange rate.

In our research, we started from the assumption that it is impossible to predict the state of the assets or currencies, due to the influence of a large number of random factors on the market. The sum total of these factors at each moment of trading causes the price to change. Our goal is not to predict the cost of asset or currency but rather to build a model capable of making the most profitable decision at every moment of trading, relying only on historical data of the price.

Technical analysis is a methodology for forecasting prices, based on an analysis of price movements in the past. It was interesting to explore how the model could generalize the patterns in price fluctuations in the market and learn how to make the right decisions, just like professional traders do.

While working on the model, in the following primary assumptions were made:

1. Transactions are made instantaneously, at a closed price for the period in which the transaction is made
2. The trading volume of the model is negligible, and does not affect the price change in the market
3. On the order, the opening and closing of the trading position takes place at a price

of Close [1]

The problem we are trying to solve is complicated by the fact that we do not know what we want to teach the network. There is no algorithm for successful trading that we could implement, and thus no target feature / value by which we can classify or regress. Instead of predicting a future price we want to use deep recurrent neural networks for building an algorithm of smart opening and closing trading positions to maximize profits of trading on financial stock market.

In other words, the problem we are trying to solve is finding the goal function reflecting the behavior of the "ideal trader" and to build a model capable of learning from historical data and giving recommendations as close as possible to the results of the goal function.

1.2 Research process

In a situation where the result is not always known (we can't be sure our goal function is correct), testing our hypothesis is going to be complicated as well.

Errors can occur at each stage: data collection, data preprocessing, target result generation, neural network configuration, errors within the network, and errors during the validation or interpretation of results.

To minimize the risk of errors and the impact of errors in one of components on other parts of the system, the work on this project was organized in the following way:

- The work was done in iterations.
- Each iteration solves a particular problem or tests a particular hypothesis.
- The solution was tested and validated in several ways.
- The results were analyzed.
- The results of the analysis made changes in the existing approach to modeling, or new hypotheses were used to adjust the existing approach.
- Each new iteration included writing new code and was tested together with the previous version for being sure new design doesn't break anything.

Before the research, we studied the existing scientific works in this field [2], [3], [9] that explore a similar problem and can be used as a starting point for our research. Based on research results [10], [11] a decision was made to use the LSTM architecture as the state of the art for the time series data.

To solve the problem of the lack of "goal function", that could be used for training neural network, it was decided to move in two directions:

1. try to determine the desired behavior of the algorithm and to teach the network to generalize it
2. allow the model itself to find the optimal solution, rewarding the network for profitable transactions and penalizing for unprofitable ones

[1] The market price is represented by four values: the minimum price (low), the maximum price (high), the opening price (open) and the close price (close).

1.3 Original contribution

The original contribution of the author presented in this thesis includes:

- Data collecting
- Development of the pre-processing workflow
- Design and training neural networks
- Development the validation workflow
- Perform the data analysis

Chapter 2

Preliminary

2.1 Initial data set

For any data exploration or machine learning task, it is critically important to have a quality data set for training and testing the model. In our work, we examine historical data of currency exchange market Forex for the currency pair EURUSD for the period from 2001 to 2015. The positive quality of this data set is the accuracy for analysis and simulation without preliminary preparation and cleaning of the data. An example of data set is provided in table 2.1.

Since training on full data set requires a huge amount of computational resources, the following methodology was used: hypotysis testing was performed on a data set in one month (for example: Jan 2005 training, Feb 2005 testing), the most successful hypotheses were tested on an annual data set (for example: 2005 for training, 2006 for testing).

2.2 Good strategy limitations

The first step of the study was to determine what a good strategy is and what we would like to get as a result. There are a lot of trade-offs. For example, one strategy can be to make more short transactions, which is more profitable, but also more risky. Another strategy is to play on long transactions, which is more expensive in terms of the cost of holding positions (finance leverage). The "naive approach" of determining the most effective strategy is trying to make all possible trading orders and choosing a combination of transactions that gives the maximum profit. Suppose a virtual trader is able at any

<TICKER>	<DATE>	<TIME>	<OPEN>	<HIGH>	<LOW>	<CLOSE>
EURUSD	20090501	000100	1.3233000	1.3233000	1.3225000	1.3229000
EURUSD	20090501	000200	1.3228000	1.3231000	1.3223000	1.3223000
EURUSD	20090501	000300	1.3225000	1.3227000	1.3223000	1.3226000
EURUSD	20090501	000400	1.3226000	1.3228000	1.3222000	1.3225000
EURUSD	20090501	000500	1.3225000	1.3230000	1.3222000	1.3224000
EURUSD	20090501	000600	1.3223000	1.3223000	1.3219000	1.3222000

Table 2.1: An example of data set

moment to perform one of three actions: buy, sell, wait, deciding what action to take once every 10 minutes (144 times a day). Then to find the best strategy by looking through all possible options, we should sort out 2924064 strategies:

$$\mathsf{C}_n^k = \frac{n!}{k!(n-k)!} = \frac{144!}{(144-3)!} = 2924064$$

As the price on the market changes more often than once a minute (in the data set used by us for research there is every minute price change), it could take months and years of historical data to be analyzed. Finding the efficient strategy by straightforward search would be a challenging computational task.

2.3 Strategy generation and optimization

Instead of looking through all possible options, it was decided to analyze the effectiveness within a small "time window", which, on the one hand, minimizes the costs of holding positions and on the other hand, simplifies the complexity of the algorithm, making it possible to create a multi-threaded implementation.

Another, optional restriction, is to consider only one opened transaction per unit of time (mentioned in section problem statement). This will significantly simplify the computational complexity of the algorithm. Technically, it is no problem to consider several independent strategies, each doing separate trading orders independently, assuming that at some point in time a virtual trader will have several open transactions. However, the restriction makes sense for simplification, better predictability of modeling and testing.

One more specific of the "greedy" algorithm and opening of only one transaction at a time is the ability to reduce the number of transactions that are being searched. After finding the best solution within the window, the algorithm should no longer go over the possible solutions within the window, just beyond.

The basic idea of the algorithm that determines the "best strategy" is that the algorithm goes through a training set with historical data, opening transactions for buying and selling with variable durations (limited by size of the time window), selects the most profitable transaction, and divides the timeframe in half and algorithm applies to the both parts (timeframes before and after), recursively.

The result of the algorithm is a list of signals to buy / sell. The intensity of the signal being proportionate to the profit of the transaction on the signal. Below is a fragment of the code 2.1, that illustrates the idea of the algorithm.

The graph 2.2 reflects the dependence of the window size on time based on historical data for the year 2001 (more than 200K records). The linear nature of the graph indicates that algorithm has been optimized to O(n) complexity.

2.4 Strategy analysis

Investigating the work of the algorithm, a relation was found between the size of the window, the number of transactions, and the profits in different years. Examples are shown in the graphs 2.3, 2.4, reflecting the dependence of the model's efficiency on the

```python
def best_action(start_pos, steps_in_forward):
    end_pos = start_pos + steps_in_forward
    if SIZE < end_pos:
        end_pos = SIZE

    max_profit = 0
    best_action = None
    for state in [State.BID, State.ASK]:
        for close_pos in range(start_pos+1, end_pos):
            profit = estimate(state, start_pos, close_pos)
            if profit > max_profit:
                max_profit = profit
                best_action = Action(state, start_pos, close_pos, profit)
    return best_action

def search_the_best_action(pos_start, pos_end):
    the_best_action = None
    for i in range(pos_start, pos_end):
        steps = STEPS_FORWARD
        if pos_end - i < STEPS_FORWARD:
            steps = pos_end - i

        best_action = best_action(i, steps_in_forward=steps)

        if best_action is not None:
            if the_best_action is None:
                the_best_action = best_action
            elif best_action.profit > the_best_action.profit:
                the_best_action = best_action
    return the_best_action

def search_best_actions(pos_start, pos_end):
    action = search_the_best_action(pos_start, pos_end)
    if action is None:
        return []

    left = search_best_actions(pos_start, action.pos_from)
    right = search_best_actions(action.pos_to, pos_end)
    result = left + [action] + right

    return result
```

Figure 2.1: Fragment of the code that represents the algorithm of search for optimal strategy

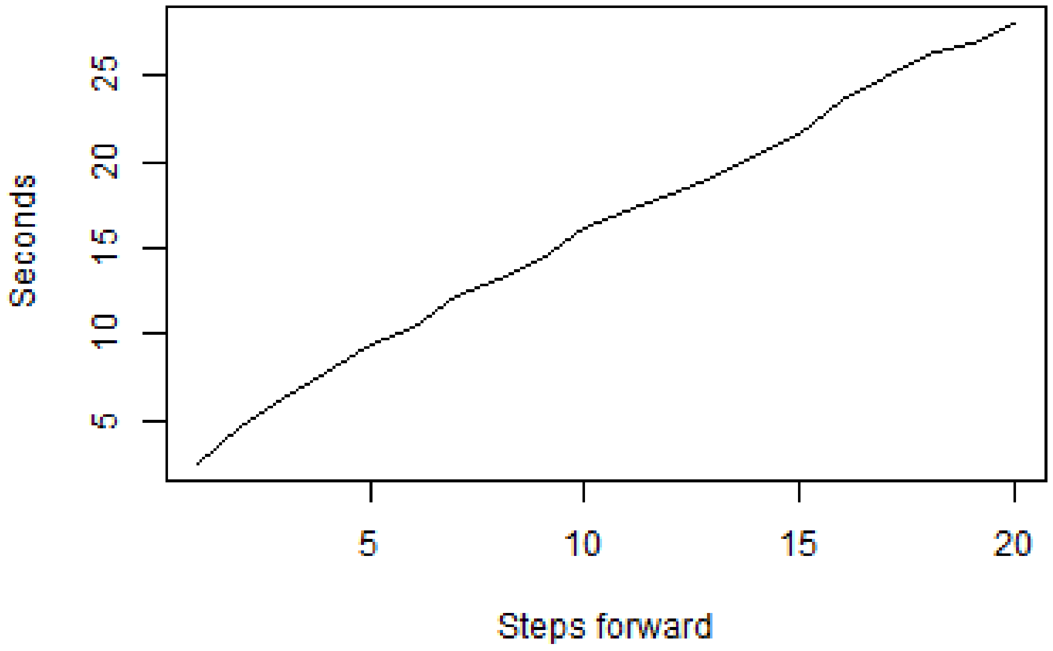

Figure 2.2: Algorithm performance by time window size

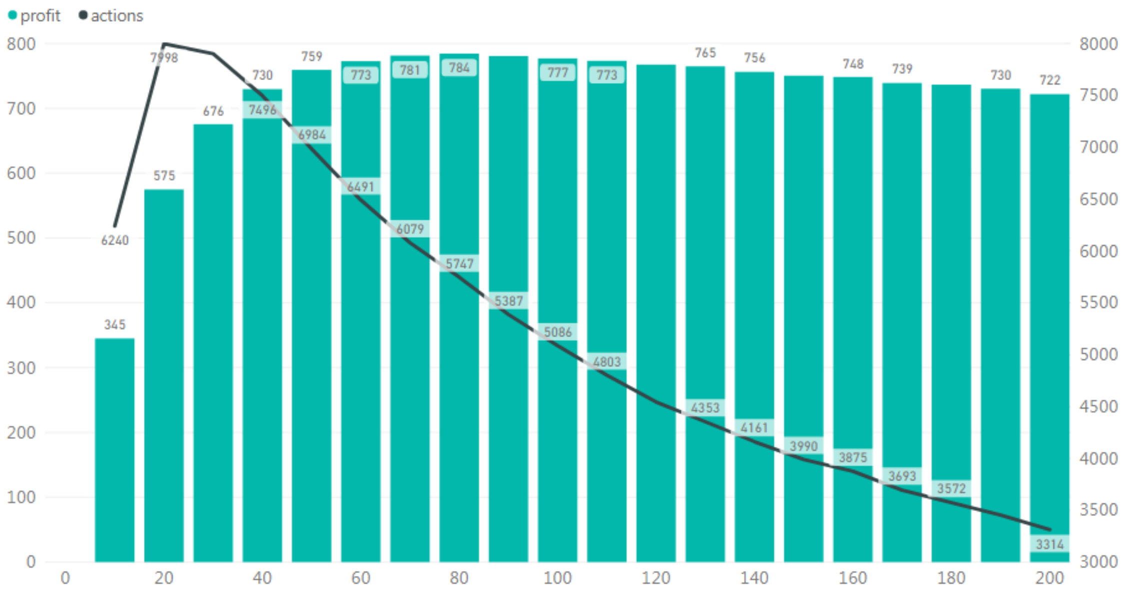

Figure 2.3: Relation between the count of trades and yearly profit in 2004

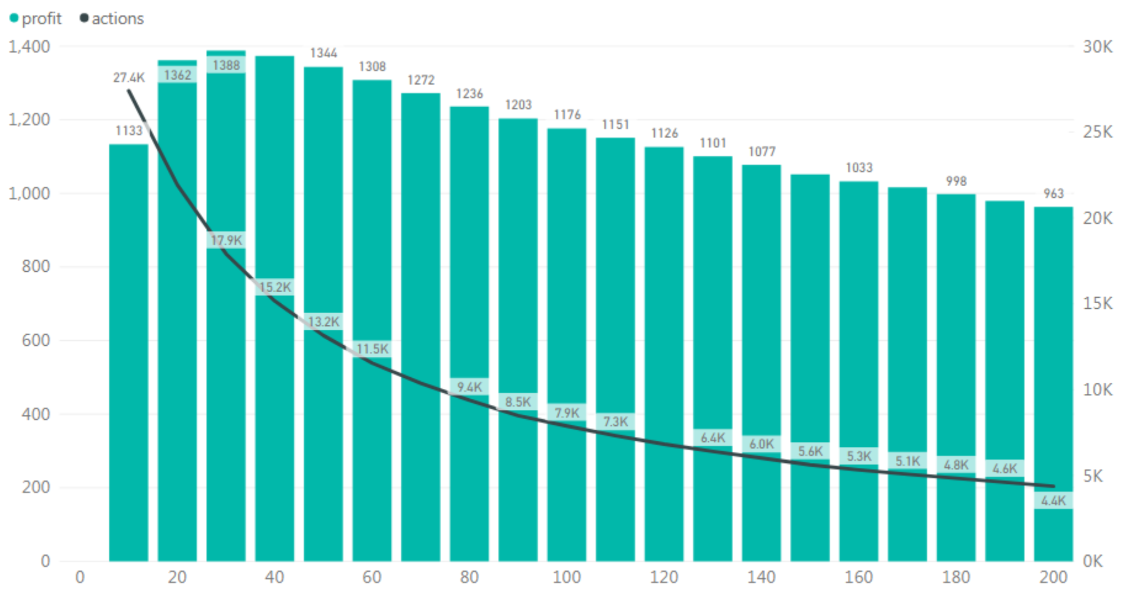

Figure 2.4: Relation between the count of trades and yearly profit in 2015

window size and the number of actions in 2004, 2015. So, in 2004 it is clear that the profit with the increase in the size of the window degrades poorly, while the number of actions decreases significantly. Thus we believe that it makes sense to use bigger time windows for achieving almost the same result, but with fewer orders (less risky). In 2015, the situation is different, and the maximum efficiency model shows at a window size of 50 minutes, as seen in the figure 2.6.

Analyzing the result of the algorithm for Forex historical data from 2001 to 2016 on the graph 2.5, it is clear that in most cases the large window is not so effective (the algorithm shows the maximum efficiency on time window sizes from 40 to 60 minutes), although from year 2003 up to 2007 a larger window size shows shows better algorithm performance.

The reason for this non-linear dependence of the size of the window on trade efficiency is that, with a small window size, the strategy does not open long enough trades, and misses the opportunity of more profitable, "long transactions". It can be seen for the time windows of 10-20 minutes for the years 2004 and 2015. By increasing the window size, we increase efficiency until those long transactions begin to "swallow" short transactions. And at the moment when "swallowed" short transactions in the sum give a better result than the long ones found, the efficiency of the model begins to degrade.

In 2005 the profit starts to fall after the window is increased by over 30 minutes, in 2004 the degradation occurs much later (primarily because of the smaller number of transactions in 2004), after increasing the window size over 80 minutes. The figure 2.7 shows how the number of transactions decreases significantly and the profitability increases with increasing the window size. The graph 2.6 shows the dependence with the window size increase, the

2001	2002	2003	2004	2005	2006	2007	2008	2009	2010	2011	2012
50	80	200	200	200	200	200	50	40	50	50	60

Table 2.2: Optimal value of window size for the period 2001-2012

number of transactions made, and the profit from the transactions. It is seen that the maximum profit with a minimal time window is reached somewhere in the region of 30-50 minutes of transactions.

The table 2.2 contains the optimal window values for annual periods from 2001 to 2012.

Therefore, as soon as the situation on the market changes over time and the window size significantly affects the efficiency of the algorithm, the window size should be one of the metaparameters of the final model.

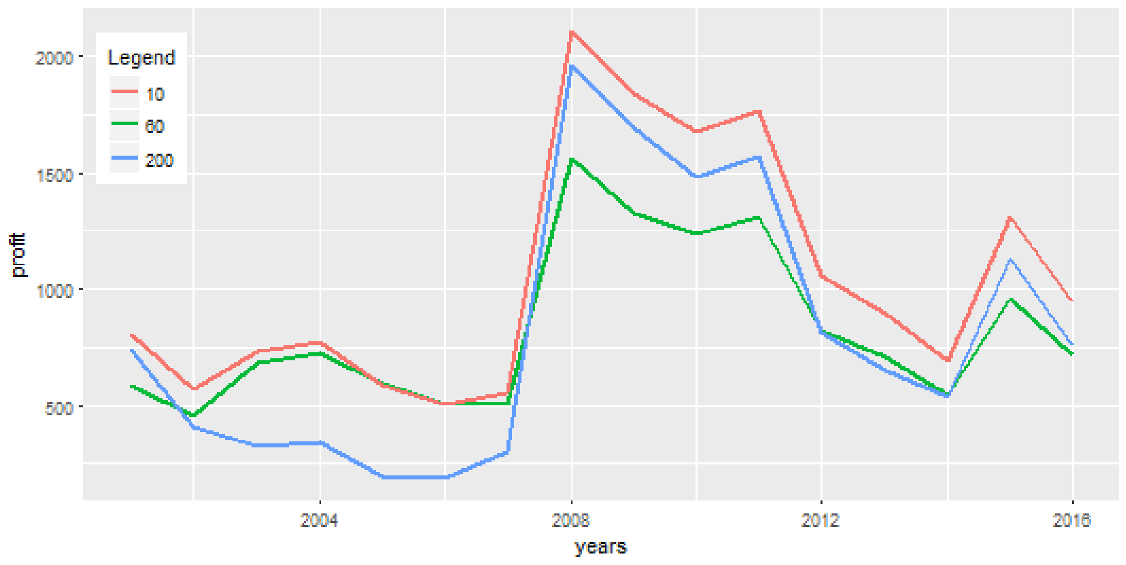

Figure 2.5: Profit in 2001-2016 calculated by the algorithm with lookup for 10, 60 and 200 steps forward

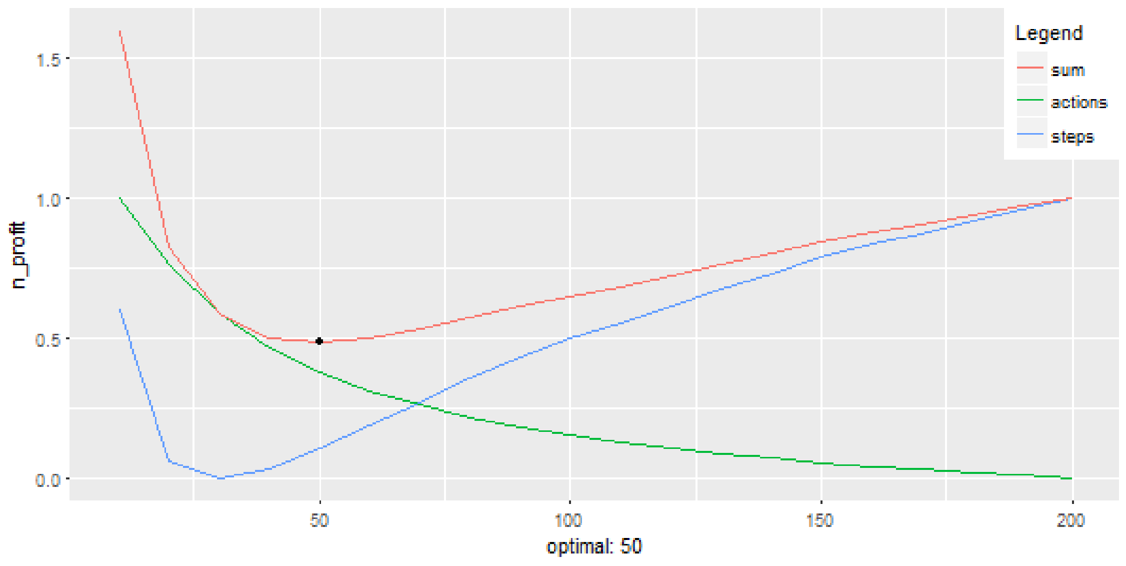

Figure 2.6: Relation between count of trades, profit and size of the time window in 2015 and optimal size of the time window (getting max profit with min orders)

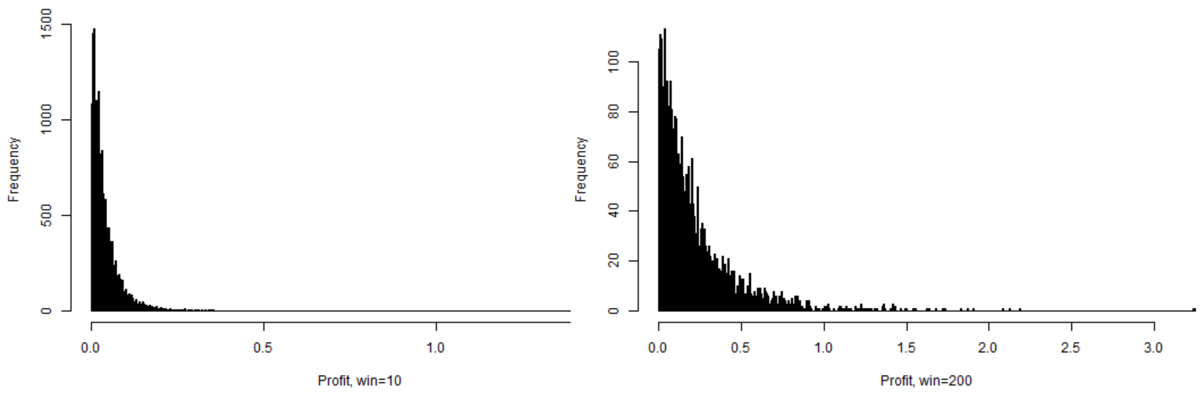

Figure 2.7: figure
Redistribution of profit in 2015 with the time window of 10 and 200 min

Chapter 3

Model

3.1 Data and pre-processing

The methodology proposed above was adopted as the basis for determining the goal function for training the model. In addition, the following methods of normalizing input data were tested:

- Normalization:
$$x_i = \frac{n_{i+1} - n_i}{n_i}$$
- Difference:
$$x_i = n_{i+1} - n_i$$
- Difference of differences:
$$x_i = (n_i - n_{i-1}) - (n_{i+1} - n_i)$$
- Tangents:
$$x_i = \frac{n_i - n_{i-1}}{0.0001}$$
- Angle:
$$x_i = atan\left(\frac{n_i - n_{i-1}}{0.0001} \times \frac{180}{\pi}\right)$$
- Variations of Moving Average with various windows.
- Financial functions RSI, FSTD, ADX

In addition, after analyzing the frequency of price change types on the market, the following pattern was observed: there are a few standard types that come up frequently. The types of price changes are shown in histogram 3.1 Having tried to allocate a cluster of values using the k-means algorithm, several classes were determined (table 3.1). Based on this discovering the changes of price could be divided in this way:

- $-\infty \cdots - 70$.
- $-70 \cdots - 50$.
- $-50 \cdots - 30$.

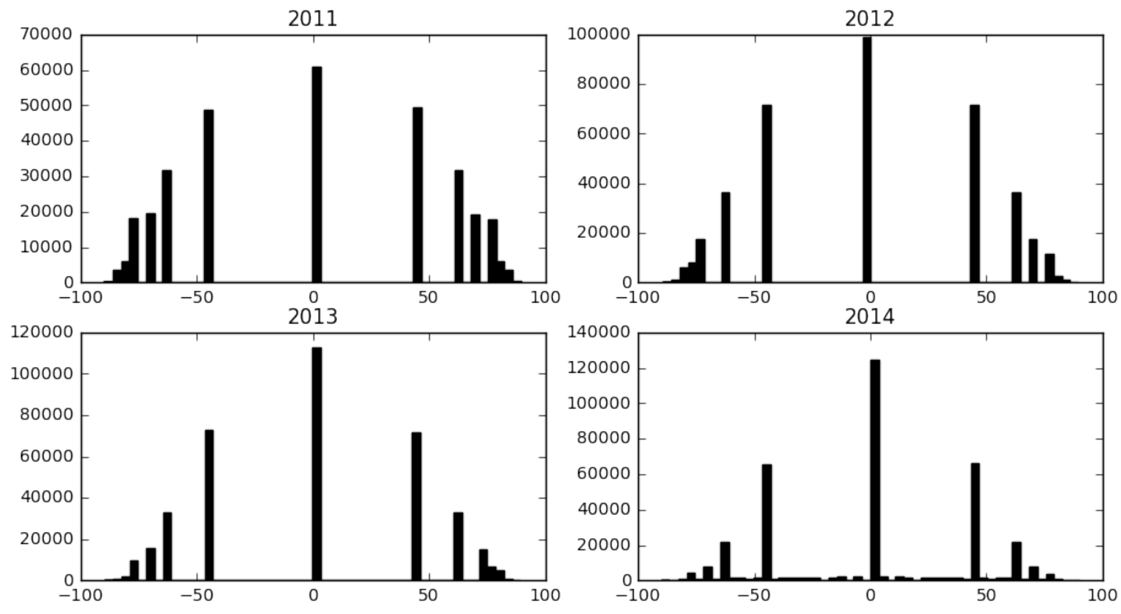

Figure 3.1: Frequency of price changes types during the year

- $-30 \ldots 30$.
- $30 \ldots 50$.
- $50 \ldots 70$.
- $70 \ldots \infty$.

These classes formed the basis of another type of normalization, Simplify.

In addition to the mentioned types of preprocessing, different types of training data set (only Close price, Close price with Low and High prices) were tested. Moreover, various values of the time window for training the neural network, as well as batch size[1], were tested.

[1]The size of the mini-batch does not affect the learning result of the neural network, only the speed of learning

2011	2012	2013	2014	2015
-75	-74	-74	-67	-69
-63	-63	-63	-44	-45
-44	-44	-45	-20	-23
0	0	0	0	-2
44	44	44	20	20
63	63	63	44	46
75	74	74	67	69

Table 3.1: Centroids in clusters of price change types for the years 2011-2015

3.2 Data processing and validation framework

The work on the project began as a set of notebooks in Jupyter. When the notebooks became too difficult to support, the main functionality was rewritten to a set of functions that could conveniently manipulate the data. Later, this approach also began to get complicated and the project was rewritten in the object-oriented style, with active use of python's magic methods for a simple and fluent API that helps work with input and output data. The project continued growing complicated, more experiments made and the performance bottleneck and the possibility of a declarative configuration of experiments became a bottleneck. The next version adopted the design of the object-oriented system as its basis, using NumPy for collections, which allowed to increase the productivity 4-5 times, as well as caching all the results of preprocessing and models. These changes significantly accelerated the work on training and model validation.

The following conclusions were made:

- The API should be as simple and flexible as possible, allowing easy configuration of experiments and hide the details of the implementation from the programmer as much as possible. Any "leaking abstractions" of implementation lead to potential errors, as well as significant time-consuming debugging and experimentation.
- The detailed information about the process, the source data and the code of each experiment should be preserved for the purpose of analyzing the course and result of each experiment separately. It must be possible to restart the experiment and get the same result.
- The results of preprocessing and the trained model should be cached to ensure that the same computational work is not performed twice. This accelerates the work on experiments.
- It is important to have a declarative API, which allows generating experiments in search of the optimal solution.

3.3 Recurrent Neural Network

Recurrent neural networks (RNN) proved to be efficient in the analysis of sequential data and are widely used for machine translation, image classification, video classification, voice and emotion recognition. Because price changes are a type of time-series data, our goal was to prove its effective use for building trading strategy on highly volatile market. Since classical recurrent neural networks have a problem of vanishing (or exploding) gradient, it was decided to use the LSTM network architecture, which does not have such problems. Despite the fact that LSTM networks have been known for more than 20 years, this architecture is still actively used in the industry and finds new practical applications every day. Below is a list of papers that explored the application of RNN and LSTM networks for finding patterns and forecasting in time-series data:

- Stock Chart Pattern recognition with Deep Learning [14]

- Time Series Prediction: Predicting Stock Price [15]
- Cryptocurrency Portfolio Management with Deep Reinforcement Learning [6]
- Time Series Forecasting Based on Augmented Long Short-Term Memory [12]
- Sequence to sequence weather forecasting with long. International Journal of Computer Applications [13]

3.4 Validation

Since we are interested in the algorithm of automatic trading, the task of the algorithm will be find the optimal moment for opening and closing a trading position. The strategy generation algorithm described above generates the opening and closing signals that will be used to train the model.

The effectiveness of the model, first of all, will be estimated based on the calculated profit from the algorithm. To understand the nature of the result, the visualization of the algorithm's work makes it possible to intuitively evaluate the logic of the algorithm.

In addition to the visualization for validation, the following metrics were used:

- Confusion matrix.
- Precision and accuracy.
- The maximum possible profit in a time window.
- The total time of transactions (longer transactions mean a larger commission and risks).
- Number of successfully and unsuccessfully closed deals.
- Count of successful and unsuccessful transactions.

The analysis of metrics allows to understand how (due to what) the algorithm maximizes profit and where it fails.

In addition, the validation algorithm uses the following parameters that affect the overall performance of the algorithm:

- Maximum-minimum length of open trade position.
- Stop loss - maximum allowable loss for a trading position.
- Optimism - heuristics that makes the algorithm less cautious when closing transactions, waiting longer for a suitable market situation, to close the transaction.

3.5 Reinforcement learning

The idea of using reinforcement learning arose when we thought about how to properly interpret market information or the generalized understanding of the network we trained and the decision about buying and selling. The idea of reinforcement learning is that having an environment and being able to interact with it using a specified set of actions, an agent trying different behaviors gets a certain reward for actions that lead to a given goal. Thus, in the course of training, the algorithm optimizes the agent's behavior to maximize reward. In our case, reward represents a profit from a closed trading position. An action is buy-sell-wait, the environment is a data set of historical data from the market.

3.6 Hardware and Software

All work was written on the HP Pavilion g6 laptop and tested on the server of the Poznan University of Technology with the GeForce GTX 1080 Ti graphics card.

Python 3.5.3 and the following libraries are used as the main programming language:

- NumPy 1.13.1, for working with mathematical data
- Keras 2.0.6, as frontend for working with Tensorflow
- Matplotlib 2.1.0, to visualize the data
- TA-lib 0.4.17, for working with financial data
- pickle from the standard library, to serialize the cached values
- Jupyter 5.3.1, as a tool for prototyping and testing algorithms

Chapter 4

Experiments

4.1 Hypothesis validation and experiments strategy

Several directions were chosen in which to conduct experiments. Before, was created an environment in which the sales agent could learn from historical data from the market, validating the trained agent on new data. Also, two types of agents were created for learning and testing them agains two types of environment to compare the results of their work:

- Agent that learns multilayered neural network and use normalized market data as environment
- Agent that learns LSTM network and use normalized market data as environment
- Agent that learns multilayered neural network and use output of LSTM networke trained over data set prepared by our strategy generation algorithm
- Agent that learns LSTM network and use output of LSTM networke trained over data set prepared by our strategy generation algorithm

To simplify this part of the experiment, since it requires a maximum of computing power, a data set of 2,000 training records and a data set of 2,000 validation entries were used. Another simplification was the considering of a strategy only on price growth: in order to make a profit the agent should buy at a lower price and sell at higher.

4.2 Hyperparameters tuning

A number of experiments were made to investigate the different architectures of LSTM networks. We investigated networks with 1, 2, and 3 hidden layers and different dimensions of hidden layers, using dropout to avoid overfitting.

Two approaches to obtaining the result were used: the classification of the best possible action of the trader (buy-sell-wait), the prediction of the signal (purchase-sale) model. Having carried out experiments with the selection of optimal parameters of training epochs, activation functions and various types of preprocessing, we found out the following:

- For LSTM, tanh (Hyperbolic tangent activation function) works best as an activation function.

- The number of learning epochs is best done dynamically using the EarlyStopping callback. It is experimentally determined that the optimal value of patience is 10.
- The number of hidden layers and the dimension does not affect the result as much as data normalization.
- MovingAverage and Simplify for input data, allow the network to generalize the result more efficiently.

4.3 Hyperparameters tuning summary

Initially, the idea was to train the network in such a way that it generated a signal for the opening and closing of trading positions. Unfortunately, the model's work produces a rather noisy result, as a signal example on graph 4.1 demonstrates. An attempt to build a trade with a smoothed or filtered signal did not get good enough results. Experiments with embedding technique for a filtration technique didn't work efficiently either.

An interesting discovery was made by training the network on the values of "looking ahead" at a greater number of steps, helped enhance strong changes in the price and make the output signal more clear. An example of such a signal is shown in the graph 4.2. Despite the fact that it was possible to significantly improve the signal quality signaling that it is worthwhile to open a transaction, the algorithm could not rely on the same signal when making a decision to close the trading position due to strong noise and false / premature closing transactions. As a result, it was decided to split the logic of opening and closing a trading position. To open transactions, the algorithm uses a signal from the model optimized for this purpose.

Worth a separate mention is the algorithm for distinguishing the positive and negative signals. In the course of operation, the algorithm, each time when opening a transaction, collects statistics on the best possible position, remembering the signal level. The algorithm yields decent results based on the average value of the best signal of the last 20 transactions.

In search for the optimal algorithm for closing transactions, the following options were investigated:

- "Naive algorithm", which closes the transaction after the effectiveness of the transaction begins to fall, parameterized by the meta-parameter "optimism".
- LSTM type neural network, trained on transaction closure signals.
- Markov chain, which determines the probability of trend movement and the rationality of closing a trading position.

Considering the wide variety of parameters for experiments, we decided not to go through all possible combinations of metaparameters, but to implement a genetic algorithm that generates random candidates and mutates the most successful of them. A simplified version of the algorithm is shown in the listing 4.3.

To avoid getting into the local minimum, the algorithm performs random generation

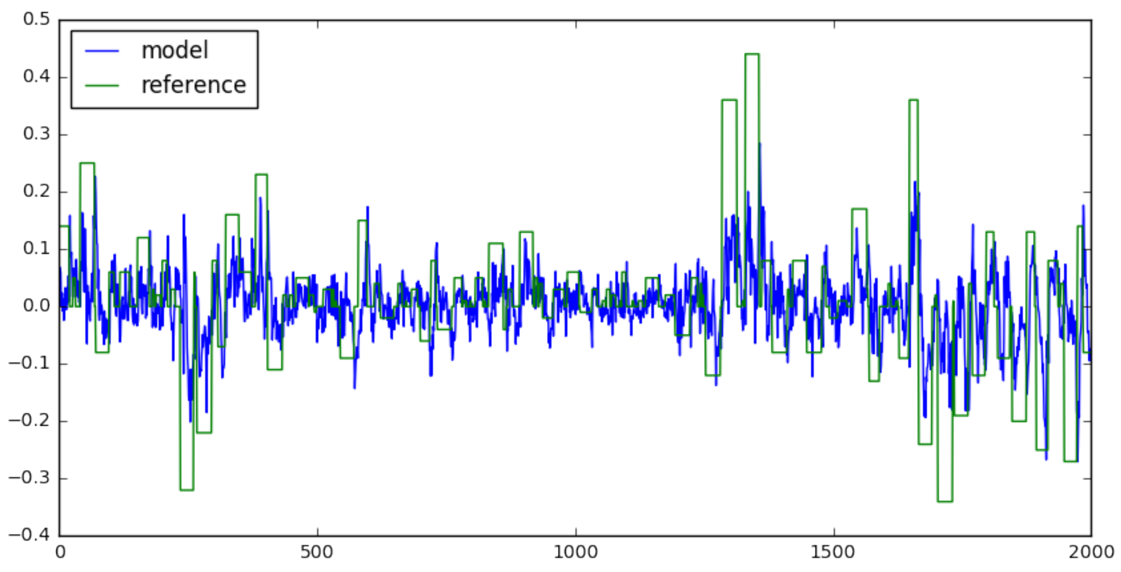

Figure 4.1: An example of reference trading signal and signal from a pre-trained model

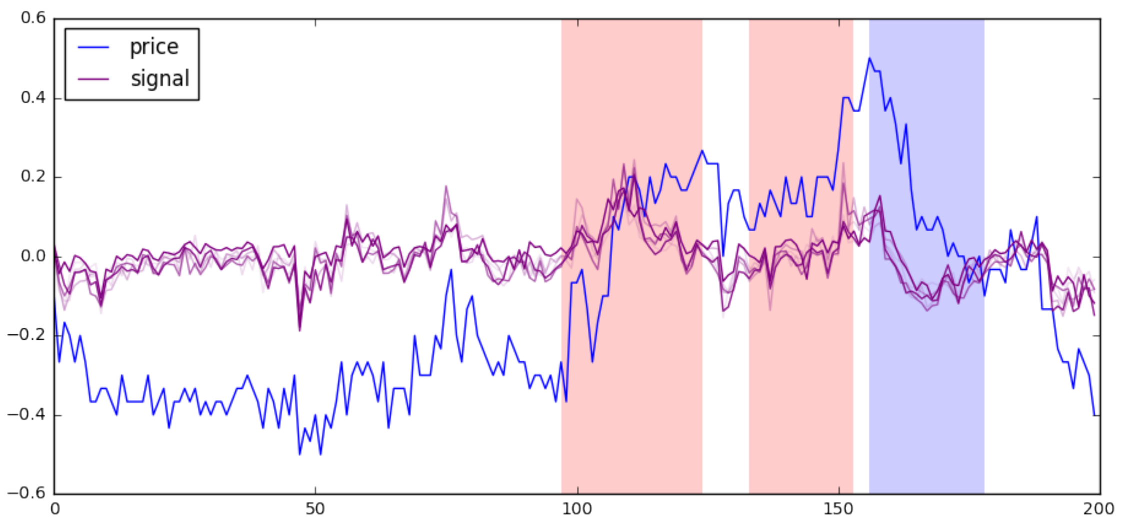

Figure 4.2: Signal amplification by increasing lookup value

```
score = -sys.maxsize # min possible value
while True:
    candidate_generation = generate_candidate()
    cur_score = validate_candidate(values)
    reward = cur_score - score
    if cur_score > score and score != 0:
        # use found candidate for further mutations
        apply_condidate(candidate_generation, reward)
        score = cur_score
    else:
        # don't use this candidate cause it's worse than existing
        discard_candidate(candidate_generation, reward)
```

Figure 4.3: Simplified fragment of code that represents the mutation algorithm

of candidates instead of more productive algorithms like Differential Evolution [1]. The performance problem is solved with caching all processed operation, which ensures that the processing is never performed for the same set of parameters twice. The same applies to processing data and calculating models, where calculated result will always be reused.

The mutation occurs for all metaparameters of the algorithm:

- Parameters of data preprocessing: normalization type and normalization parameters (for example different window size for MA smoothing), window size, lookup parameter.
- Metaparameters of neural networks and various network architectures (network type, activation function, layers, hidden layer size, etc).
- Various implementations of algorithms for opening / closing trading positions, optimistic and pessimistic strategies.

It should be noted that the algorithm is very quick to understand that the most profitable strategy of cost minimization is not to open trading positions at all. So, in the process of mutations, candidates with zero actions are immediately discarded. The estimated function for the algorithm was profit-loss and the adapted measure of accuracy (only true-positive values are taken for buying and selling and ignoring true-negative / false-negative for signals when waiting is selected instead of buying-selling).

The best parameters was found by mutation algorithm are:

- Normalization algorithm: Moving Average with window size 3 over tangent of price change and Simplify pre-processing algorithm
- Window size for input data set: 12 minutes
- Architecture of newral network: three LSTM layers with 512, 256, 128 neurons in hidden layer accordingly
- Prediction for training data set in 6 minutes ahead
- The best result was achived with building successful strategy within 30 minutes window

4.4 Reinforcement learning summary

Following the research strategy for validation the reinforcment lerning model, were chosen to validate two options of the neural network (multilayered neural network and LSTM) and two versions of the data set. In data sets were selected 2,000 entries from historical data of the Forex EURUSD currency pair for 2005 year. Unsupervised data set normalized with tangent of value change function described in section 3.1. Supervised data set preprocessed with algorithm described in section 4.3. The agent used to train and validate the model is limited to a maximum of 10 open trade positions. Each trade position opens for $1000.

[1]An example of effective searching optimal solution with differential evolution algorithm: https://pablormier.github.io/2017/09/05/a-tutorial-on-differential-evolution-with-python/

The results of the total profit and loss for each of the methods are given in the table 4.1. In detail, the results of each of the experiments are shown in graphs 4.4, 4.5, 4.6, 4.7.

The results of experiments show that an agent with an LSTM network opens longer transactions, but as noted in the section 2.4, longer transactions after some limit are less efficient than short, plus are more risky. It should also be noted that, normalization is one of the most significant factors affecting ability learning the network, and the selected methods and parameters of normalization in the section 4.3 show good results for both supervised and unsupervised data sets.

This set of tests did not show significant advantages of supervised data set over unsupervised, however, experiments on large data sets showed quite good results for a supervised approach. Unfortunately, experiments on large data take much longer and have not been sufficiently investigated.

Experiment	Profit	Loss
Unsupervised, multilayered NN	6757.8	146.7
Unsupervised, LSTM	27976.5	76.4
Supervised, multilayered NN	2679.2	411.5
Supervised, LSTM	26286.5	358.3

Table 4.1: Results of validation model with reinforcement learning and supervised/unsupervised data sets

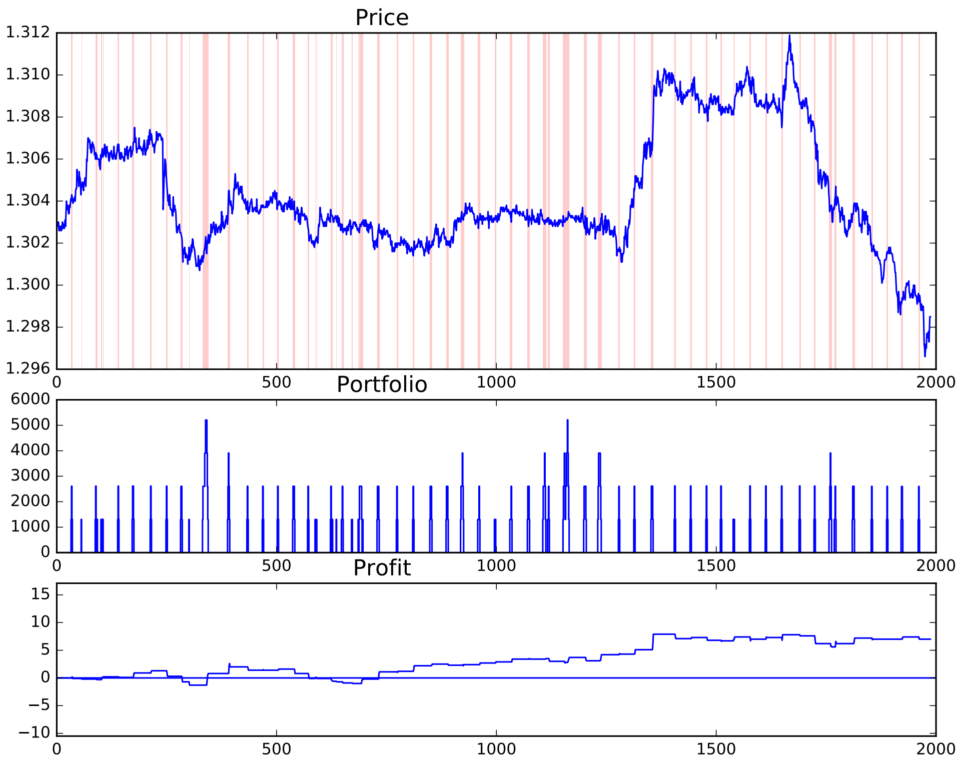

Figure 4.4: Unsupervised data, multilayered neural network

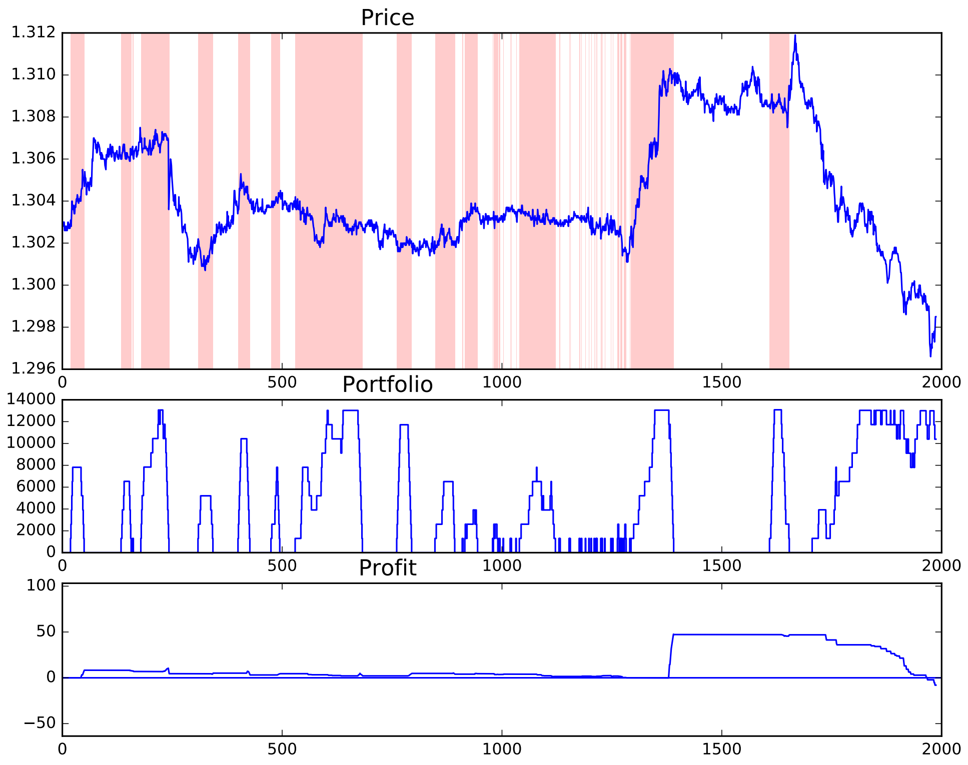

Figure 4.5: Unsupervised data, LSTM network

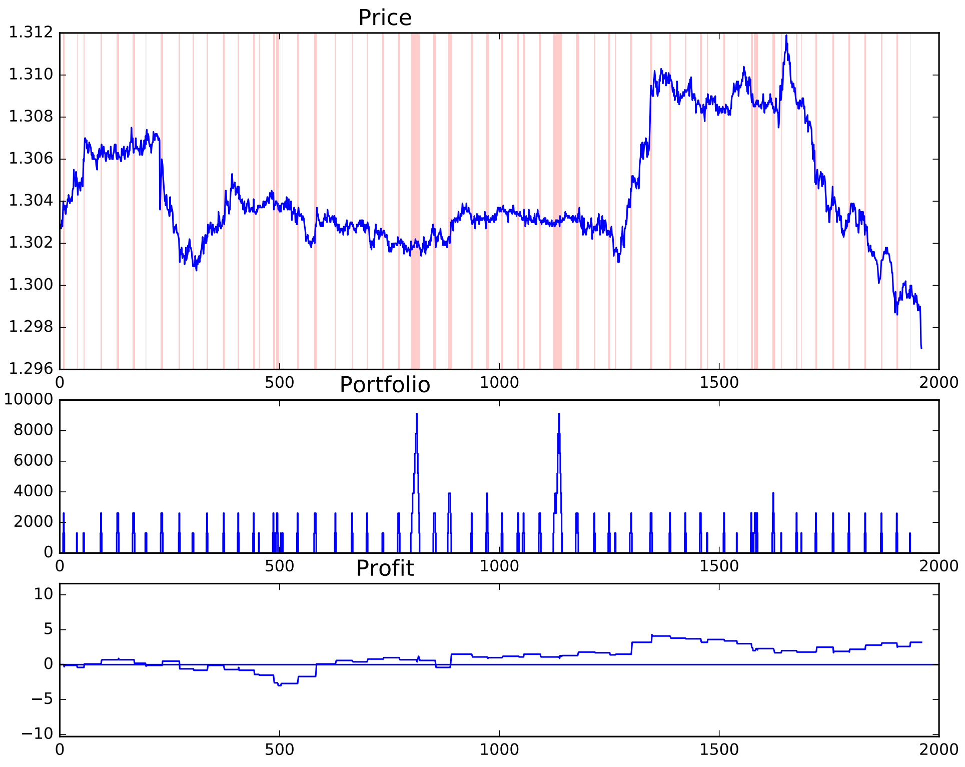

Figure 4.6: Supervised data, multilayered neural network

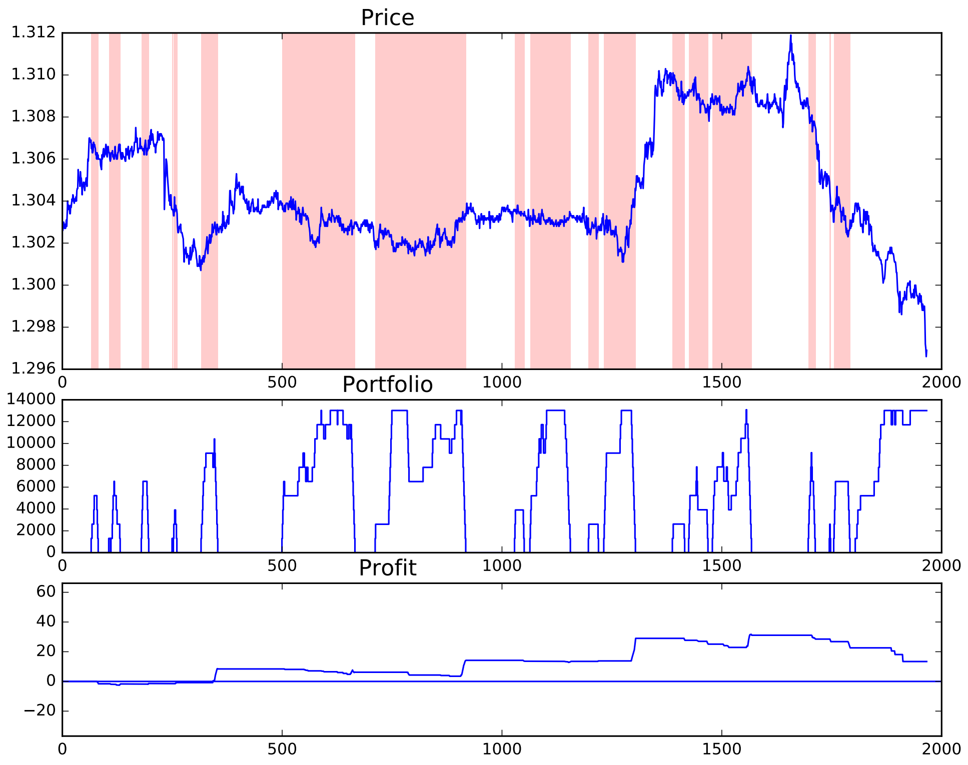

Figure 4.7: Supervised data, LSTM network

Chapter 5

Summary

The result of the work is a library that implements a set of components that allow to model strategies for arbitrary data sets, validate, visualize and save the simulation result, provide statistics and debug information, automate the search for optimal solutions.

The results of the experiments show that neural networks can definitely be used to build an effective trading strategy, especially if the network takes into account more factors, such as seasonality, media analysis [16] [17], analysis of several markets, etc. It is worth to mention that each of the areas studied by us, could be discovered deeply. A huge field for research lies in studying how the network learns from various financial indicators, how networks of different architectures are trained and how different types of normalization affect learning, as a combination of different algorithms allows to achieve better results. Lot of experiments and potential opportunities opens up modern methods of reinforcement learning in combination with deep neural networks.

Bibliography

[1] Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. Nature, 521(7553):436-444, 2015.

[2] T. Kuremoto, S. Kimura, K. Kobayashi, and M. Obayashi. Time series forecasting using a deep belief network with restricted Boltzmann machines. Neurocomputing 137 (2014): 47-56.

[3] P. Romeu, F. Zamora-Martnez, P. Botella-Rocamora, J. Pardo. Time-series forecasting of indoor temperature using pre-trained deep neural networks. In: Artificial Neural Networks and Machine Learning ICANN 2013, pp. 451-458, Springer ,2013.

[4] John Myles White. Bandit Algorithms for Website Optimization, ISBN: 978-1-449-34133-6

[5] Volodymyr Kuleshov, Doina Precup. Algorithms for the multi-armed bandit problem. Journal of Machine Learning Research 1 (2000) 1-48

[6] Zhengyao Jiang, Jinjun Liang. Cryptocurrency Portfolio Management with Deep Reinforcement Learning

[7] John Moody and Matthew Saffell. Reinforcement Learning for Trading Systems and Portfolios.

[8] Yuriy Nevmyvaka. Yi Feng. Michael Kearns. Reinforcement Learning for Optimized Trade Execution

[9] Philip Arvidsson. Tobias Ånhed. Seqence-to-Seqence Learning of Financial Time Series in Algorithmic Trading

[10] Sepp Hochreiter. Jurgen Schmidhuber. Long short-term memory

[11] Felix A. Gers. Jurgen Schmidhuber. Fred Cummins. Learning to Forget: Continual Prediction with LSTM

[12] Daniel Hsu. Time Series Forecasting Based on Augmented Long Short-Term Memory

[13] Zaytar, M. A. and Amrani, C. E. (2016). Sequence to sequence weather forecasting with long. International Journal of Computer Applications, 143(11).

[14] Marc Velay and Fabrice Daniel. Stock Chart Pattern recognition with Deep Learning (2018)

[15] Aaron Elliot, Cheng Hua Hsu. Time Series Prediction: Predicting Stock Price (2017)

[16] Johan Bollen, Huina Mao, Xiao-Jun Zeng. Twitter mood predicts the stock market (2010).

[17] Venkata Sasank Pagolu, Kamal Nayan Reddy Challa, Ganapati Panda, Babita Majhi. Sentiment Analysis of Twitter Data for Predicting Stock Market Movements (2016).