VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

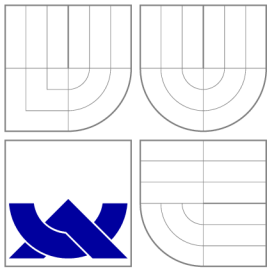
STROJOVÉ UČENÍ V ELEKTRONICKÉM OBCHODOVÁNÍ

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

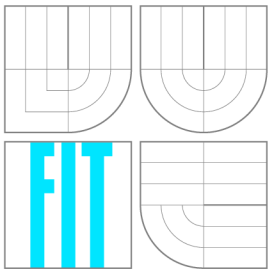
AUTOR PRÁCE
AUTHOR

PETR HUF

BRNO 2014



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

MACHINE LEARNING STRATEGIES IN ELECTRONIC TRADING

STROJOVÉ UČENÍ V ELEKTRONICKÉM OBCHODOVÁNÍ

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

PETR HUF

VEDOUCÍ PRÁCE
SUPERVISOR

Doc. Dr. Ing. JAN ČERNOCKÝ

BRNO 2014

Abstrakt

Úspěšné obchodování na trzích je snem mnoha lidí. Zajímavým odvětvím tohoto byznysu je elektronické obchodování, kde obchodní strategie běží na počítači bez jakéhokoliv zásahu člověka. Tento způsob obchodování poskytuje spoustu volného času a vysoké příjmy. Tato práce je zaměřena na využití neuronových sítí při stavbě takovéto obchodní strategie. Jako základ byla použita již existující rekurentní neuronová síť, která byla postupně modifikována podle potřeb pro obchodování. Výsledkem je neuronová síť předpovídající budoucí pohyby trhu. Obchodní strategie používající tuto neuronovou síť dokáže na burze úspěšně obchodovat.

Abstract

Successful stock trading is a dream of many people. Eletronic trading is an interesting branch of this business. The trading strategy runs on the computer all the time without any human intervention. This way of trading provides a lot of free time and high earnings. This thesis is aimed at usage of neural networks in building this type of trading strategy. An already existing recurrent neural network was used as a basis and was modified for the needs of trading. The result is a neural network which predicts future market moves. The trading strategy based on this neural network is able to perform a successful trading.

Klíčová slova

neuronová síť, rekurentní neuronová síť, burza, obchodování, trh, profit, model

Keywords

neural network, recurrent neural network, stock exchange, trading, market, profit, model

Citace

Petr Huf: Strojové učení v elektronickém obchodování, bakalářská práce, Brno, FIT VUT v Brně, 2014

Strojové učení v elektronickém obchodování

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Doc. Dr. Ing. Jana Černockého. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Petr Huf
May 16, 2014

Poděkování

Tímto bych chtěl poděkovat svému vedoucímu bakalářské práce, Doc. Dr. Ing. Janu Černockému, za odborné vedení práce, poskytnutí mnoha rad a nápadů a příjemnou atmosféru. Poděkování patří také Ing. Františku Skálovi, který mi velmi pomohl při řešení problému spojených s neuronovými sítěmi. Také děkuji své rodině a přítelkyni za podporu a trpělivost při psaní této práce.

Contents

Chapter 1

Introduction

Trading exists since a long time ago. In the beginning it was just „piece by piece“. Almost everything was used as a currency – from eggs, milk and other food to cows, goats and other animals, mostly livestock. The establishment of currency has changed this concept and in these days, in almost every trade, money got its place.

People bartered and still barter goods, money and services. In every trade, always two participants take part–the seller and the buyer. Seller offers some goods or service to the buyer. Everything is happening for the purpose of a profit. The seller usually obtains the money from the buyer, which he can barter further. The buyer obtains required goods or service. Most people imagine trading as a visit of some shopping center, supermarket or perhaps cafe. But in this thesis, we will focus on stock trading.

Stock exchange is a regulated market, where can be bought or sold various shares of companies, commodities or diverse financial derivatives. The word „regulated“ is very important in this context. It means that there is a supervisor who controls the whole market[1]. Most often it is some government subject. I do not want to discuss exactly how stock exchange works and describe everything, what can be bought or sold there. I will confine myself to simple description of facts, that are useful for understanding this thesis, see chapter 2.

1.1 Motivation

Stock trading provides huge potential, but also big danger. The issue is that stock trading is a very specific kind of business. Stock exchange provides possibility to make big money and also lose a lot of money. A great advantage is that stock trading can be done by everybody in these days. All you need is just input capital. Great emphasis should be put on this condition, because undercapitalized traders are losing their money very often. Higher capital gives more possibilities and gives bigger space to mistakes.

In this thesis, I will concentrate on using neural networks in stock trading. Neural networks are usually used in artificial intelligence as data structures capable of learning. The basic unit of neural networks is a neuron, which models human neuron with simple abstraction. Neural network is a kind of simplified model of a human brain. And how will it look like in stock trading?

The purpose will be, simply said, to teach neural network the market activity and then use it as a predictor of future prices. With this knowledge, a trading system should be built,

[1] For example, this does not apply to forex market, where so-called market-maker can be broker.

with two goals: safety and profitability. This system should be able to run without any user input, perform trades and make profit, of course. It should be noted that a necessary condition to reach this goal is not only the right prediction, but also trade management, execution planning, money management of the whole trade system etc. As we see, building trade system is like assembling a mosaic, which consists of many details and problems, that need to be solved. But if the whole mosaic is successfully compound, the reward will be a profitable and safe trading system.

1.2 Structure of thesis

In chapter 2, trading is described in a form that is important for this thesis. All information, necessary to ensure overview and put thesis to the context of trading, are mentioned there.

The following chapter 3 describes principles of general networks. This chapter does not cover full neural network theory, as architectures, constructions and optimization of neural network are not subject of this thesis (in detail).

The next chapter 4 is describing how the network had to be modified for using it in price prediction.

Chapter 5 contains description of trading system architecture and interfacing the neural network to this process is described in detail.

The most interesting chapter 6 documents the testing process and its results.

The last chapter 7 summarizes the results and mentions a few ideas for future work.

Chapter 2

Overivew of trading

Stock exchange provides information about buying and selling prices. These prices are established on a principle of supply and demand. If supply prevails demand, then prices are falling. And vice versa, if demand prevails supply, then prices are rising. This information can be mostly seen as charts. Trading itself is very simple. A trade can be opened as long or as short. Long trade is opened, when trader expects rising prices. Because of that, he buys relevant commodity and waits until price grows. At the moment he wants to stop the trade, he simply sells the commodity. The resulting difference between buying price and selling price is trader's profit. If selling price is lower than buying price (price of commodity has fallen instead of expected rise), then the trader received a loss. Short trade is the exact opposite of long trade. Stock exchange allows selling a commodity even if trader did not buy it before. It means that trader can speculate on commodity prices decrease too. For example, trader expects falling prices, so he decides to sell a commodity. If he wants to stop the trade, then he has to buy commodity back. His resulting profit is the difference between selling and buying price again. If price of the commodity has grown up instead of expected decrease, then the trader received a loss.

As a defense against higher losses than the trader is able to tolerate, the trader can use *stop loss* command [3], which allows defining a certain price. When the price of commodity intersects this price, the trade is automatically canceled. So the trader can define his maximum loss in one trade this way[1].

To be able to do stock trading, a broker is necessary. Broker has usually access to stock exchange, so he ensures the execution of trades for his clients. As a reward for his services, he is charging commissions for every trade. This costs the client roughly $5–$10 per trade. Another trader's expense is a *slippage*. Slippage is a situation, when the trader sends a command to his broker to buy or sell some commodity. Meanwhile, the price can rise or fall a little and the trade is opened on price different from price in time, when the command was sent. Time delay plays a significant role here. It is also good to realize that for the trade, two participants1 are needed – the seller and the buyer. And it can take some time to find counterparty for the required trade.

2.1 Chart types

As it was already mentioned in chapter's introduction, the most common visualization of actual and historical prices of a commodity are charts. There are many types of charts, but

[1] Just approximate loss. For example, it may come to slippage. This term is described further.

we will introduce just a few basics. At the beginning, it is necessary to explain something about price movements. Price of a commodity sometimes changes very quickly. For generating a chart, the price is drawn in certain intervals. And just the way of defining of this interval gives rise to create various types of charts. To keep the most information from this interval, every drawn line (unit) consists of 4 parts:

- Open – The price at which a commodity was traded at the time of a drawing the beginning of the line.
- High – The maximum price at which a commodity was traded in the time of a drawing the line.
- Low – The minimum price at which a commodity was traded in the time of a drawing the line.
- Close – The price at which a commodity was traded in the time of a drawing the end of the line.

For the chart reading, colors are also used. Green lines represent growing lines. It means *Close* is greater than *Open*. Red lines represents falling lines. It means *Close* is lower than *Open*. Let's finally introduce a few basic chart types.

- Time – This is the most common type of chart. A line is drawn every second or minute or 5 minutes etc. The user can choose any time period he wants.
- Range – When this type of chart is chosen, line is always drawn when price moves about given number of points. So the difference between *High* and *Low* is still the same. If the price do not move much, line drawing can take a longer time. As a result, every line represents different time interval.
- Volume – In this type of chart, every line is drawn when given number of contracts (not trades) was traded[2].
- Tick – This type of chart is similar to *Volume* chart. Only difference is that interval is not defined by number of contracts, but by number of trades.
- Renko – This type of chart is made by lines of the same height. This height is given by the user. If descending line is followed by another descending line, then drawing behavior is normal. But if a descending line is followed by a rising line, the price has to move twice more than the height of lines to draw this rising line. Open price is drawn higher to keep the height of line still the same. In case of rising lines the principle does not change. Figure 2.1 should explain it better.

2.2 Money management

Money management is an integral part of every successful trade strategy [9]. Let's start with the balance. The height of the balance is directly linked to market which we want to trade. Good choice could be *share indexes*, which are very liquid[3] and cheap. For successful trading, input capital of $10 000 should be enough.

[2]Number of trades is not the same thing as number of traded contracts. In one trade, it is possible to buy or sell more contracts. So every trade consists of one or more contracts

[3]Sufficient number of trades are made. It is no problem to open and close trade whenever we want.

Figure 2.1: Time based chart vs. renko chart.

- NQ – E-mini NASDAQ 100. It is the cheapest share index on market for trading. The smallest move is 0.25 points, it is $5 in dollars. This market is suitable for the beginners also because of its small volatility[4]. That is the reason, why small stop losses can be used.
- YM – E-mini DOW. This market moves in entire points. One point symbolizes $5. Compared with NQ, this market is not so much liquid, but more volatile. This leads to more frequent slippage and a possibility to reach greater profits.
- YM – E-mini S&P 500. The smallest move of this market is 0.25 points, which is $12.5. This is more expensive market meant more likely for professional traders.
- TF – E-mini Russel2000. The smallest move is just 0.1 points, which is equal to $10. This market is the most volatile of this share indexes. Because of that, it is suitable for professional traders.

Stop losses $100 high should be sufficient on these markets. For safe trading, it is recommended to define 1% of balance as loss per one trade. It follows that the input

[4]Market's moves are more likely small

capital should be $10 000 high at least, as was already mentioned above.

2.2.1 Trade more contracts

To increase the success rate, trading with two contracts is nothing unusual. But do not forget about *margin*. Margin is some kind of refundable deposit, which is saved by broker while trader has opened trade. The amount of margin depends on the number of contracts bought by the trader. To be able to open a trade, the balance has to be as high as the required margin at least.

Trading with more contracts can be used for increasing the trading stability. One of well-known techniques is buy more contracts and sell them one by one during the trade. Let's show it on an example. A trader sees a nice opportunity for a long trade. He buys two contracts and then he is waiting until price reaches profit target that he has determined. After the price reached this target, he sells one contract. Now he is holding just one contract. This contract is used to maximize profit as much as possible. This simple method showed to be very highly efficient.

2.2.2 Diversification

Diversification is a powerful tool and every successful businessman should know about it. In trading, there are several ways how a trader is able to diversify his behavior. The first simple way how to diversify is trading on more markets. However, it is not recommended to novices, as every market has his own characteristics and going from one to another leads to worse perception of these characteristics which leads to more losses. Because of that, it is necessary to well consider this step and be ready for this situation. This step is much easier in case of automatic trading.

Another way is to develop more trading strategies. It does not mind when some strategy fails if more strategies are traded. The other strategies will cover losses caused by currently non-functional strategy. Meanwhile there is an opportunity to fix the problem. But there is one thing you should care about. Strategies should not correlate with each other[5]. The result would be the same as trading one strategy multiple times. In case of manual trading loss of concentration is quite possible and then following rules of strategies has worse quality. Automatic trading is making it simpler again.

2.2.3 Risk reward ratio

It is a very important term which you should know. Before entering a trade, you should be able to evaluate the risk potential of the trade. If stop loss is set to $50 and profit target is $10, then RRR (risk reward ratio) is only 1:0.2. It means that we can gain just one fifth of money we risk. Trading strategies based on higher RRR have better chance in general. Of course, higher success rate could be achieved with lower RRR, but more consecutive losses may seriously endanger the balance. It takes much longer to cover losses. Higher RRR allows lower success rate, but only a few successful trades increase the height of the balance. Why risk more than you can get? Testing is the basis of everything and it should show importance of this term.

[5]They should realize dissimilar trades.

2.3 Types of trading

There are very different types of trading [6]. In this chapter, some basic types of trading will be described. In the end, we will select a type of trading for this thesis.

- Position traders – This kind of traders open a trade, stay in for a few days or weeks and then close it. This attitude leads to necessity to have higher input capital, less opportunity for trades, higher commissions, but also it is less time-consuming and mentally demanding. We have plenty of time to make analysis of the market. Another advantage is free data, because there is no need to have real-time data which are paid.

- Intraday traders – A trade made by these traders takes several hours, minutes or even seconds. This style of trading is quite demanding. Quick decision has to be made, you have to sit and watch market a few hours a day and real-time data are paid. But it brings great benefits. There are enough trade opportunities, you do not need high input capital and balance growth can be very fast (as well as a decrease). Psyche plays a very important role here.

- Trend-following traders – These traders are waiting for a trend in market. Trend can be explained as still growing or decreasing market. This kind of trader tries to make a profit on a trend. How? He „simply“ participates in the trend. Trading with a trend is simpler than trading against a trade. Not for nothing it is said: „Trend is your friend.“

- Trading against the trend – The basis and the most difficult task is to determine when the current trend is over and an opposite trend is starting. If a trader is able to do it, then he opens a trade just at the beginning of the trend. It makes larger profits than opening a trade in the middle of the trend. This style of trading is more demanding and a prediction of a change of trend ends with no success very often.

- Scalpers – These traders are opening a trade just for a very short time – a few seconds or even milliseconds. They make hundreds of trades for a minimum profit, but with high success rate. This is one of the most difficult trading styles. It requires high concentration, speed and very fast internet connection.

- Fundamental traders – This kind of traders make decisions according to messages published by various news and economic servers, according to weather and weather forecast, actual events, simply according to information from the whole world.

- Technical analysis – This is an opposite of fundamental traders. Traders, using technical analysis makes decisions, using charts only. These charts display price evolution in time and allow drawing various lines, indicators and another marks useful for making decisions. They are able to read significant information from charts and use them for market's movements prediction.

- Discretionary traders – These traders are sitting at a computer, watching price and making decisions about opening and closing trades. They can have some rules for opening and closing trades, but they always use actual „mood“ of market in decisions.

- Electronic trading – This term can be understood in two ways. First, it can be a situation, when a trader uses a computer for sending trading commands to his broker

and he sends it directly to stock exchange's servers. This style of trading prevails older pit trading, where instead of servers, every broker has his man on stock exchange and this man makes trades according to commands from his broker. The second way how this term can be understood is a situation, when a trader has his own trading strategy which is assembled from simply defined rules. The trader programs rules into trading platform and let trading strategy run on a computer. Then the strategy is running and making trades without any intervention from the trader. Finding working strategy can take some time, but after this part is done, trader can do almost nothing with running trading strategy.

In this thesis, trading strategies based on intraday trading will be used, because we can make more trades with lower input capital on these data. Decisions about opening and closing trade will be based on a price history – technical analysis will be used. Strategies will be programmed and then evaluated by trading platform – we will use electronic trading.

2.4 Electronic trading

Creating computer program trading without any intervention of the trader, is difficult. Thinking that having such a program would reward you with a lot of free time, is correct only to a certain extent. The creation of a good trading algorithm is not easy. The trading strategies, which are easy to trade manually, are not necessarily programmed. Programmed strategies do not provide overall view of the market and make trades in situations, when the author of the strategy would evaluate a situation as very risky. The creation and finding of functional strategy can be very time-consuming. Other aspects, which have to be taken into consideration, are problems outside of the computer program. Internet connection can fail anytime and opened trade can stay opened and the trader will not have any chance to close it via the internet. Nevertheless, he does not have to even know about it, because the strategy is running whole day and the trader does not watch it all the time. This situation can leads to large losses. A solution could be to buy a virtual server, where a provider guarantees backup internet connection, backup power supply and other services.

A great advantage of electronic trading is the possibility to test a trading strategy very quickly. In case of discretionary trading, it is necessary to go through all charts in a certain period and mark all trades, which would be made according to trading strategy. The whole trading strategy can be evaluated only after this process and this process can take days or weeks. And if any change in this strategy is made, the whole process has to be repeated. A programmed trading strategy does not have this problem. Any change in the trading strategy can be tested immediately in a few seconds. Trading platforms provides evaluation of the trading strategy in details – drawdown[6], profit per trade, success rate etc.

Another great advantage is unequivocal rules of a trading strategy. The program evaluates a situation always as expected. Because of that, the results of testing and real trading do not differ much. This is a problem in discretionary trading, where the results of testing and real trading often do differ very much. It is because decisions depends on a human and this decision making is very influenced by actual mood, psyche, results of a few last trades and many others factors. These factors are one of the most common reasons, why traders are losing their money.

[6] Maximum decline of a balance

2.4.1 Approach to creating algorithms

Many approaches can be used to creating trading strategies. One of the most simplest attitudes is watching price movements and other indicators and making decisions about opening or closing a trade based on it. Experiences from discretionary trading can be useful here (see section 2.3). A trader can program trading strategies, which were profitable when he traded them manually. The trader wants to do some optimization very often. Strategies and various indicators have their own parameters and proper combination of these parameters can turn a loosing trading strategy into a profitable one. And this takes us to using genetic algorithms. On the market, tools exist, allowing finding proper combination of input parameters using genetic algorithms, which speeds up finding of the best combination considerably.

Another approach will be tested in this thesis. Involvement or neural networks to electronic trading introduces a different view on the creation of profitable trading strategies. There is no need to think about describing strategies for opening and closing trade. It is assumed, that the neural network will learn market moves itself and thanks to this ability, the neural network will be able to predict future prices. How reliably a neural network can do that, will be shown in this thesis.

This application in electronic trading can be seen as straightforward at first, but many hidden important aspects need to be solved as was mentioned in section 1.1. The neural network will be trained on historical data. When being used, actual price of market will be send to the neural network and predicted price will be returned. This only information is not enough for successful trading. It is just a clue, which can be the base of sophisticated trading strategy and this trading strategy should exploit it as much as possible.

2.4.2 Overfitting

Overfitting is a problem of many trading strategies? The trading strategy is always tested on historical data and tuned to make maximum profit on them. After deployment to a real account, the trading strategy can stop working. It is because the strategy was adapted to historical data too much. There are several procedures to prevent this situation.

One of basic procedures is a separation of historical data into two parts – training data and evaluation data. Training data are used for development of a strategy and tuning of parameters. Evaluation data are used just for evaluation. If the strategy fails on evaluation data, it means, that the strategy is ovefitted and whole process has to be repeated. Ideally the results on training data and evaluation data should be similar.

Let us mention one more procedure, which is related to building automatic trading strategies. If you are looking for a really robust system, there are many possibilities, how to test the system. You can test the system on different timeframes, different chart types (time, volume, tick etc.), markets etc. Requirements on the robustness do not have to be stringent and it is possible to lower these requirements. For example, if the strategy works very well on five minute timeframe and just well on one hour timeframe, it is still acceptable.

2.4.3 Long-term functionality

When we develop a trading system, the most basic requirements include working as long as possible. It is necessary to test the strategy on sufficient number of trades [7]. If the system

is tested on a few dozens of trades, then positive results of testing can be just a short-term deviation of losing trading strategy. Thousands of trades can be considered as sufficient.

2.4.4 Platforms

It is advantageous to develop the strategy on the trading platform, that will be used for the real trading – by keeping the environment fixed, we will be sure that the strategy will work the same way as in the development phase. The trading platform also offers detailed information about the strategy when the strategy is tested. That helps the evaluation of the strategy and allows for focusing on logic of the system and not on creating mechanisms evaluating the strategy. Tradestation and Multicharts are widely used trading platforms [5], which define logic of the strategy with their own programming language – Easy language. It allows using for DLL libraries and this greatly extends the usage. The second big player on the field of trading platforms is NinjaTrader. NinjaTrader uses C# for writing a code of the strategy. C# provides many libraries and allows for connecting of many different technologies. NinjaTrader is free when using demo account. Data feed connection is also free from CQG [7] and can be used for watching market in real-time and testing your strategies. For testing purposes, NinjaTrader will be used.

[7] http://www.ampclearing.com/ninjatrader_cqg.php

Chapter 3

Recurrent neural networks

We recognize two main types of neural networks – „feed forward“ and „recurrent“ neural networks. Section 3.1.1 explains, why recurrent neural network was chosen.

3.1 Neural network functionality

Let us describe feed forward neural network first. As was already mentioned in chapter 1.1, the core of the neural network is a model of neuron. Neurons are organized in layers, usually input layer, hidden layer and output layer. More hidden layers can be used. Every layer has a fixed number of neurons. Every neuron is always connected to all neurons in neighbouring layers with synapses and every synapse has a defined weight determining its „importance“. This architecture can be seen in figure 3.1. A neuron defines his value from input synapses and transfer function (sigmoid or softmax).

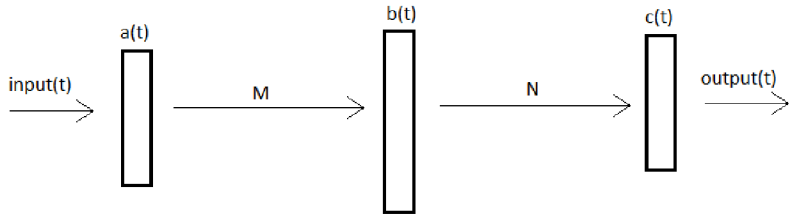


Figure 3.1: Architecture of feed forward neural network.

Let us describe, how the feed forward neural network is used. At first, neural network has to be trained. In the training phase, the neural network is fed with training data multiple times. When an input vector is presented to the neural network, it is copied to the input layer $a(t)$. Then, neurons in all layers are calculated from weights of synapses:

$$b_j(t) = f\left(\sum_i a_i(t) m_{ji}\right) \tag{3.1}$$

$$c_k(t) = g\left(\sum_j b_j(t) n_{kj}\right) \tag{3.2}$$

Function $f(x)$ is sigmoid:

$$f(x) = \frac{1}{1+e^x} \tag{3.3}$$

Function $g(x)$ is softmax function:

$$g(z_i) = \frac{e^{z_i}}{\sum_k e^{z_k}} \tag{3.4}$$

Layers are evaluated from the input layer to the output layer. After that, the result of the neural network is stored in output layer and is used for calculation of error: comparison correct value with the result of the network. Gradient of errors e_o in the output layer is calculated using cross entropy criterion as

$$e_o(t) = d(t) - c(t), \tag{3.5}$$

where $d(t)$ is a correct output which comes in next step $t+1$. e_h is backpropagated error gradient in the hidden layer.

The next step is backpropagation – the error is propagated back from the output layer to the input layer. In this moment, weights of synapses are adjusted in order to minimize the error.

$$n_{jk}(t+1) = n_{jk}(t) + b_j(t)e_{ok}(t)\alpha - n_{jk}\beta \tag{3.6}$$

$$m_{ij}(t+1) = m_{ij}(t) + a_i(t)e_{hj}(t)\alpha - m_{ij}\beta \tag{3.7}$$

This is the way, how the network is learned. The training is stopped, when the error is acceptable. Now, the network is ready for use. We will feed it with the test data and read results from the output layer. It is the same procedure as the first part of the learning. Adaption (described in more detail in 3.2) is optional – the neural network can learn from test data too.

3.1.1 Recurrent neural network

The neural network described above responds to same inputs in the same way and ignores the context of inputs. To eliminate this undesirable effect, a recurrent neural network is used. A recurrent neural network has the output of the hidden layer connected not only to the output layer, but its copy is also concatenated to the input layer (Fig 3.2), so the input layer is assembled from two parts. The first part of the input layer is initialized according to input sample. The second part is copy of hidden layer from the previous step and this is how the feedback is implemented. The reaction of the neural network depends not on a new input only, but on previous inputs too[1].

The computations are almost the same as in the feed forward neural network. The biggest difference is, that hidden layer is not calculated from the input layer only, but from previous state of the hidden layer too. The hidden layer is now calculated as

[1]Note that in feed forward neural network this is solved by enlargement of the input and output layer [8]. Anyway this solution is not perfect, the context of inputs is formed by fixed number of inputs. This is not problem of recurrent neural networks.

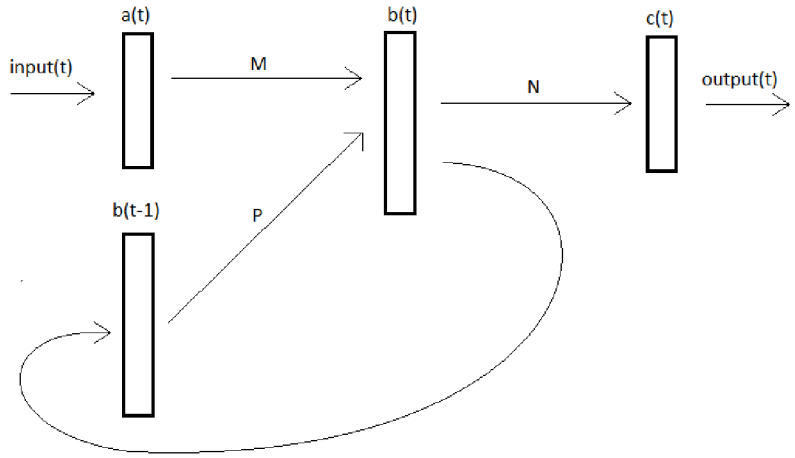


Figure 3.2: Architecture of recurrent neural network.

$$b_j(t) = f\left(\sum_i a_i(t)m_{ji} + \sum_l b_l(t-1)p_{jl}\right) \tag{3.8}$$

In the training, the weights between the second part of input layer and the hidden layer are updated as follows:

$$p_{lj}(t+1) = p_{lj}(t) + b_l(t-1)e_{hj}(t)\alpha - p_{lj}\beta \tag{3.9}$$

The training and testing take place in the same way as in feed forward neural network.

3.2 Parameters

Neural network written in RNN LM toolkit offers many parameters for setting the network. In this section, parameters used and adjusted in testing, will be described. We concentrate only on parameters influencing the network, and do not mention other configuration parameters like train file, valid file etc. All parameters are described in [2].

- Alpha – Sets the starting learning rate. Learning rate defines the speed of the training. Large value can leads to divergence and small value to slow process of learning. This parameter is applied when adjusting weights as can be seen in equations 3.6, 3.7 and 3.9.
- Beta – L2 regularization used while adjusting weights in order to ensure training stability. Its influence is very small.
- Hidden – Sets size of the hidden layer.
- Compression – Sets size of the second hidden layer. Setting it to zero turns the second hidden layer off.

- Direct – Sets size of the hash for direct connections. This parameter is not much important as it reports almost no improvements on small tasks [2].
- Direct-order – Sets the n-gram order for direct connections.
- Bptt – Set amount of steps to propagate error back in time. Setting it to zero is equal to simple RNN (simple backpropagation). Backpropagation through time allows to simulate deep feedforward neural network with N hidden layers (with the same dimensionality) with recurrent neural network with one hidden layer which is used for M time steps back. Backpropagation through time is illustrated in figure 3.3. It is necessary to save state of N hidden layers and an error has to be propagated over M time steps. This leads to more time-consuming and memory intensive training, which should result in more accurate network.
- Bptt-block – This parameter speeds up the training (at the expense of accuracy) as an error is backpropagated through time only after every P steps.
- Gradient-cutoff – Specifies maximal absolute gradient value in all layers in order to improve training stability. It can be turned off by setting it to zero.
- Dynamic – This is only parameter for testing the network. It sets learning rate for testing phase and represents adaption in run-time. It can be turned off by setting it to zero. When turned on, the network learns while testing – dynamic model. When turned off, the network does not learn while testing – static model.
- Min-improvement – Default value is set to 1.003 and it represents minimal relative entropy improvement (discrete input) or minimal error improvement (real input) in the training phase.

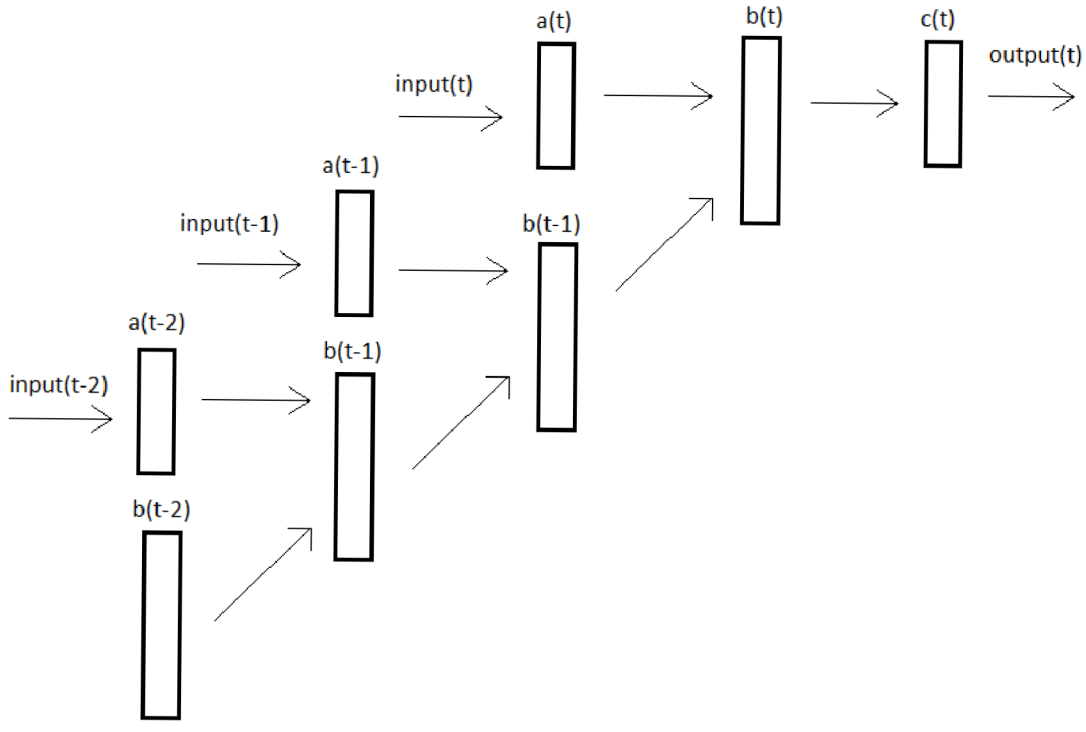


Figure 3.3: Bppt set to 3 time steps back in time.

Chapter 4

Use of RNNs for trading

Neural networks are used in many industry sectors. Their application can be found in medicine, transport, finance etc. One of the most important applications is application in language modelling where neural networks successfully compete with n-gram models [2]. This self-learning data structure is exploitable everywhere where slight inaccuracy does not mind. Neural networks can often replace otherwise complex decision-making mechanism. These properties of neural networks make them the ideal candidate for use in automatic trading strategies as a predictor of future prices.

As will be mentioned in section 5.1.2, used neural network is based on Mikolov's toolkit [2]. It was designed for statistical language modeling and because of that, there was a big effort to achieve high speed and avoid high memory consumption. Also, it contains many functions useful for language modeling but not for electronic trading. This resulted to some code reduction and simplification. But let us take it from the beginning.

4.1 Network inputs

Default RNN LM toolkit is adapted to work with words and large dictionaries. At the first sight, this state is completely unsuitable for usage in trading, nevertheless this neural network can be still used. It is just necessary to sample continuous quantity (real numbers) to discrete quantity (words). It can be done in a few ways, which will be described in the following sections.

4.1.1 Discrete sampling

The first way, how sampling can be done is very simple and fast. From numbers, data are transformed to symbols (= words) representing high growth, lower growth, no change, lower decrease and higher decrease. Limit for every single symbol can be found just by looking at the data on a few average days. This view can tell us, which growth can be considered as high or as lower, etc. For example, growth higher than 0.4 points will be a high growth and growth lower or equal than 0.4 points will be low. This approach is tested in section 6.1.1.

The second way is more demanding but also more accurate. Growths and decreases (in points/ticks) are exported from data. Histogram of these values is generated afterwards (Fig. 4.1). As decreasing values has almost the same number of occurrences as opposite growing values, decreases are transformed by changing their sign and added to growths. From this modified histogram, limits for symbols are chosen (Fig. 4.2). With decreasing

numbers of occurrences, intervals for symbols become wider. Negative values are distinguished in real usage by adding negative sign as first character of symbol. Final symbol distribution is illustrated in figure 4.3. Tests with this sampling can be found in section 6.1.1.

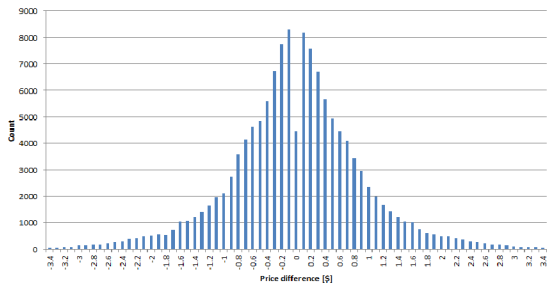


Figure 4.1: Example of histogram of growths and decreases.

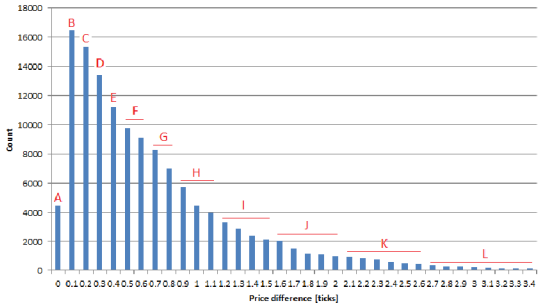


Figure 4.2: Symbol distribution over histogram.

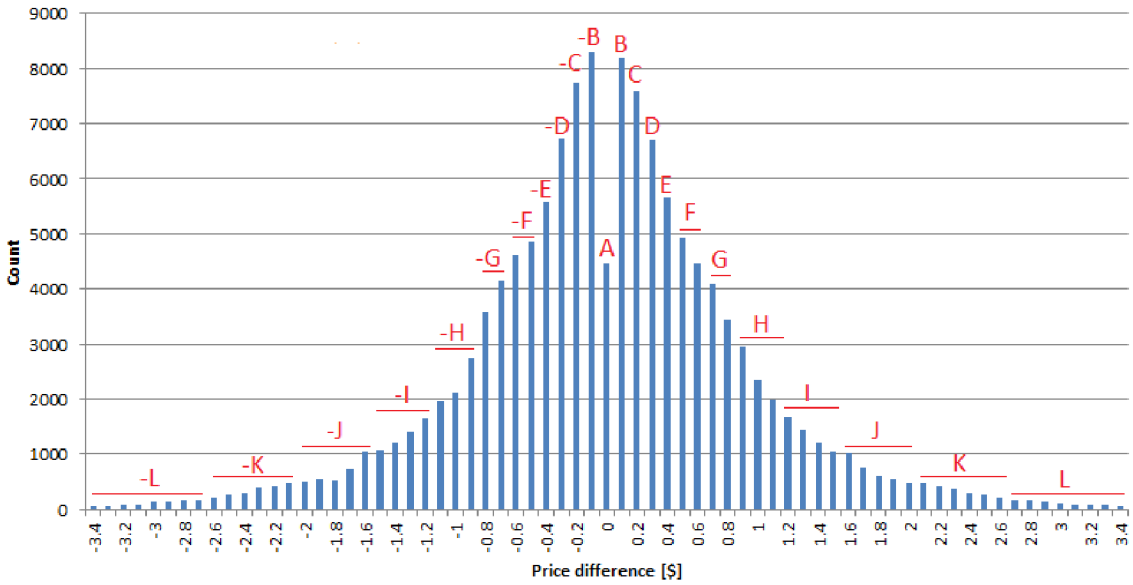


Figure 4.3: Symbol distrbution for growths and decreases is the same.

4.1.2 Continuous sampling

The neural network from RNN LM toolkit in its default state is not able to work with real numbers. Because of that, many changes had to be made, which are described in section 4.2. After these changes, growths and decreases in points can be sent to the network. The network returns prediction of next value of growth or decrease. The results of this approach are analyzed in section 6.1.2.

4.2 Changes in RNN LM toolkit

The input layer consists of the input vector, which has the size of the dictionary, and the output of the hidden layer. The input vector is always full of zeros except for one neuron (corresponding to input word), which has value one. It means that just connections between the active word and the hidden layer are active while computing the output. Neurons in the input layer corresponding to hidden layer are simply initialized by copying previous values from the hidden layer.

Hidden layer is placed between the input and the output layer. Two hidden layers can be chosen. When computing the network output, a sigmoid (3.3) function is applied to values at the outputs of hidden layers in order to introduce non-linearities into the network, otherwise, the network could learn linear functions only.

The output layer consists of two parts. First part is the vector, which has the size of the dictionary, again. Second part represents classes. Classes are implemented to increase speed of the network. This attitude significantly increases speed and decreases accuracy just a little. Nevertheless, we will not working with large dictionaries, so classes were completely removed. Results of the neural network are probabilities of words. Probability is obtained when softmax (3.4) is applied to class and word part of the output layer. Softmax in the output layer ensures that the sum of probabilities will be one [1].

This state of the neural network was used for some of tests as described in section 6.1.1. For usage as regression, some modifications had to be made. Direct connections between input and output layer were removed, as they are useless for our testing. The following modification solves transformation from word-based neural network to number-based neural network. Almost all neurons representing dictionary in the input and the output layers were removed. Just one neuron remained for representing all input numbers. When an input number comes, the activation of neuron is set to value of input. But before that, the value of input is normalized with mean-variance normalization:

$$x' = \frac{x - \mu}{\sigma}, \tag{4.1}$$

where μ is the mean and σ is the variance. Error in the output layer is calculated from difference between the predicted and real value. MSE[1]

$$MSE = E[(\hat{\theta} - \theta)^2] \tag{4.2}$$

is used as an objective function applied to this difference. Another way of evaluating the error (also shown in the toolkit's log files) is to compute the average of absolute values of differences between the predicted and the real values:

$$er = \frac{1}{N} \sum_{i=1}^{N} \mid \hat{\theta}_i - \theta_i \mid \tag{4.3}$$

Now the neural network is ready for number-based input. Tests with this input are described in 6.1.2.

[1] Mean squared error

4.3 Trading strategy

Having prediction of prices is not enough for successful trading. The second step is creating a trading strategy, which will use predictions in decision making.

Principle of basic trading strategy used in testing (chapter 6) is simple. Strategy does not look at the time as the vast majority of data are from trading hours where volume is high. But it is important to consider this when testing it on complete data or using it in a real trading.

There are two limits defined – limits for opening and closing a trade. When the network predicts big price move which crosses the opening limit, the strategy opens a trade on a relevant side (long or short). The trade is closed when the network predicts big price move against the position which crosses the closing limit. But this is not only way how the trade can be closed. The trade can be closed by hitting a stop loss or at the end of the day. Closing at the end of the day ensures, that the trade does not remain open after trading hours as volume goes down and broker can charge higher commissions for trading outside of trading hours.

Chapter 5

Implementation

In this chapter, design and implementation of programs and date structures (necessary for embedding the neural network into building of trading system and its testing), will be described. The resulting concept aims at simple usage in real trading with the smallest differences between testing and real trading.

5.1 Used software

The entire architecture is divided into two main parts. The first is the neural network and the second is NinjaTrader platform. The logic of the trading strategy is defined in the trading platform. The strategy communicates with the neural network with simple messages. The strategy sends the actual price to the neural network and gets back the predicted next price. Details of this communication are described in section 5.2. Trading strategy responds to predicted value and buys, sells or does nothing according to defined rules.

5.1.1 Ninja trader

Implementation of trading strategy is written in C#. NinjaTrader's interface offers many functions for automatic trading. Entry point of algorithm is „OnBarUpdate()“ method, which is called on every new drawn line in the chart. A line is drawn according to chosen type of chart (Fig. 2.1). Chosen type of chart has strong influence on strategy performance and should not be underestimated.

A trade is opened on chosen direction by calling functions „EnterLong()“ or „EnterShort()“. Function „SetStopLoss()“ is there for setting stop loss. The trade is closed by calling functions „ExitLong()“ or „ExitShort()“. Actual and historical[1] prices[2] are accessible as values of various indicators. These tools are sufficient for building any trading strategy.

Testing in NinjaTrader is hidden under Strategy Analyzer tab. This tab contains trading strategy, market, tested period etc. I consider settings for commission and slippage simulation very useful. When these settings are used, the results are much closer to real results.

[1] Just fixed count of values backwards.
[2] Open, High, Low, Close

5.1.2 RNN LM toolkit

Creating already created does not have any sense, therefore the RNN library designed by Mikolov [2] was used. He achieved great results in language modelling with this library. Basic design is not very suitable for our usage so many changes had to be made as was already mentioned in section 4.2.

5.2 Integration

Connecting the trading platform and the neural network is not trivial. Neural network is written in C++ language and that is the reason why it cannot be simply added as a trading strategy in NinjaTrader. One solution would be to rewrite neural network into C#, which would make everything much easier. This was not done though because of lack of the time and the neural network was kept as separate program. To make a connection between two programs, an inter-process communication – named pipe – was used. Trading strategy creates new named pipe „rnnlm“ in a role of server, then it starts the neural network as normal binary file. The neural networks loads trained network and connects to created named pipe. A simple communication follows – the strategy sends new price to the neural network and the neural network sends the predicted price back to the strategy. As the price is sent every time *OnBarUpdate()* is called, the *close* value of just drawn line is sent. The whole communication can be seen in figure 5.1.

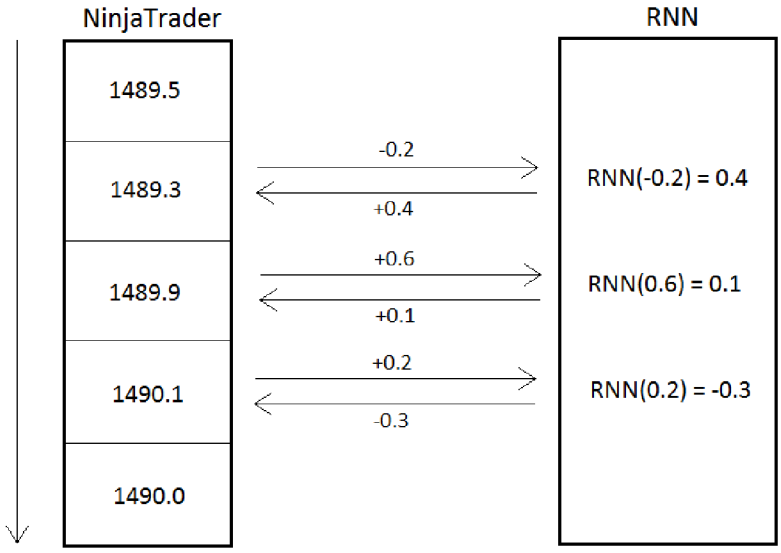


Figure 5.1: Diagram of communication between NinjaTrader and recurrent neural network (RNN).

Chapter 6

Testing and results

Modifications of neural network took place gradually. During these modifications, experiments were performed. Data were divided into training data and evaluation data. The final profit of the strategy was evaluated and was the key for comparing the results.

6.1 Data

Data from market TF (E-mini Russell2000) from 2011 to 2013 was chosen as reference data. It is a shares bundle of two thousand smallest companies, which are contained in Russell3000. The names of companies are not much important. Contract specifications are more important. The size of one tick is 0.1 points. This tick has value of $10. The value of a whole point is $100. Contracts are traded in quarterly cycle (March, June, September and December) [1].

Raw data were tick data in a simple text file, where every line responds to single trade and defines actual price in time of a relevant trade. These data had to be converted to NinjaTrader format and imported with NinjaTrader Historical Data Manager. From this point, NinjaTrader does all data sampling necessary for displaying various kinds of charts. It is important to have tick data as it contains information about trades/volume and thanks to it, volume-based chart can be displayed (volume chart, tick chart etc.).

We should note that the data is not 100% complete. Very often, the data does not contain prices from the whole day, but only from trading hours. Sometimes, we are missing whole days. These problems were not solved and all data sampling stayed in NinjaTrader's hands as only important data are from trading hours and data from other hours would be cut anyway.

Training data are from 16.12.2010 to 31.12.2012 and evaluation data are from 1.1.2013 to 25.9.2013. In experiments, a commission of $5 is included per trade and also one tick of slippage is set. These settings should simulate real market conditions. Timeframe was set 5 minutes to reduce small market movements and preserve sufficient amount of data. Initial balance is $0 to see profitability at first sight and keep independence on input capital.

The network was trained on data from 16.12.2010 to 30.6.2012. On data from 1.7.2012 to 31.12.2012, the network was tested and all parameters (learning rate, number of neurons in hidden layer etc.) were set to reach balance as high as possible. The performance of trading strategy is shown on evaluation data.

[1] See contract description https://www.theice.com/productguide/ProductSpec.shtml?specId=86

6.1.1 Discrete values

In this section, the strategy with word-based neural network will be tested. Discrete sampling is described in section 4.1.1.

Sampling by hand

The first experiments use the default version of the toolkit. Data are sampled to representing basic market movements. Word A represents higher growth (more than 0.5 points), word B represents moderate growth (less than 0.5 points), word C represents no change (still the same price), word D represents moderate decrease (less than 0.5 points) and word E represents higher decrease (more than 0.5 points). Why 0.5 points is a limit? That is because the average value of absolute values of growths and decreases is approximately 0.5 points.

The initial strategy opens a trade when A or E is expected (has the highest probability) and closes the trade when opposite direction is expected (D or E when long or A or B when short). Stop loss was set to $150 and minimal improvement in entropy stayed set to 1.003. Important training parameters were *-alpha 0.1 -beta 0.0000001 -compression 0 -gradient-cutoff 15 -hidden 30 -bptt 4 -bptt-block 10 -direct-order 3 -direct 2* and parameters for testing were *-dynamic 0.1.* All parameters were gradually tuned.

In this configuration, the whole strategy ended $-77860 in loss. At first, exit signal (from D or E when long and A or B when short) was changed to close a trade when E (when long) or A (when short) was expected. The number of trades was highly reduced and the final loss was „only“ $-28680 now. The following parameter was adaption in run-time. Changing it did not make any difference, even when the adaption in run-time was completely turned off. A change of stop loss caused an improvement. The influence of stop loss on the resulting profit can be seen in figure 6.1.

The best profit $-17925 was obtained with stop loss $230 high. A surprise happened when the sizes of both hidden layers were tuned. Changing these values did not make any change in profit. Too coarse sampling (ABCDE) can be the reason. And this reason probably stays behind results of changing all the other parameters. *Alpha, beta, bppt, bppt-block, direct, direct-order, gradient-cutoff* – changing any of them did not change any trade and profit. The strategy tuning ended with final profit $-17925 and profit $-9.24 per trade on training data. This result was expected and can be considered as good. Sampling was very coarse and without commissions and slippage, the strategy would end in a little profit. It corresponds to the fact that this sampled data are very close to random data and expected result without commissions and slippage should be around zero. The results on testing data can be seen in figures 6.3 and 6.2. Let us mention the final parameters. Training parameters are *-alpha 0.1 -beta 0.0000001 -compression 0 -gradient-cutoff 15 -hidden 30 -bptt 4 -bptt-block 10 -direct-order 3 -direct 2* and test parameters are *-dynamic 0.1.*

From a trader's view, this setup is unsuitable. As we can see from figure 6.3, the strategy is absolutely unprofitable. The number of trades is very high and it leads to a lot of money drowned in commissions and slippage. But as the most likely reason, we can consider very limited sampling. Thanks to it, the data are very close to random data. The result on test data is worse than the result on training data, but this fact is expected.

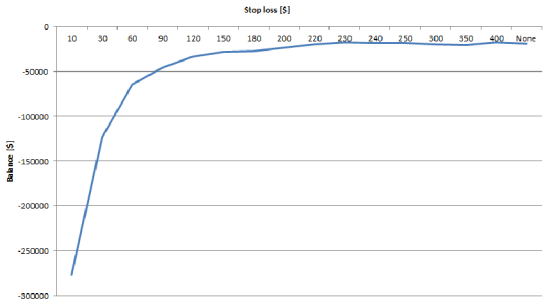


Figure 6.1: Influence of stop loss with ABCDE sampling.

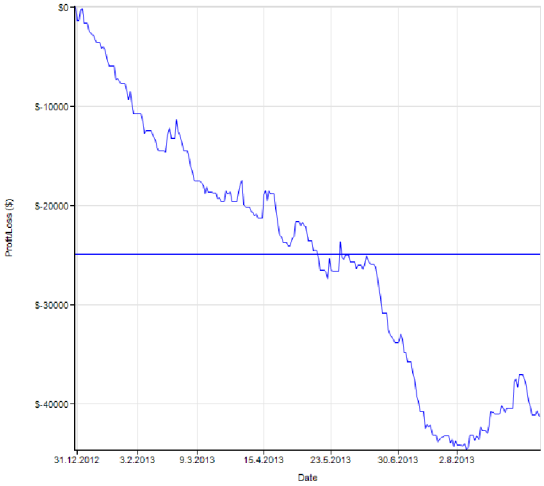


Figure 6.2: Equity with ABCDE sampling.

Performance	All Trades	Long Trades	Short Trades
Total Net Profit	$-41305.00	$5640.00	$-46945.00
Gross Profit	$159495.00	$53360.00	$106135.00
Gross Loss	$-200800.00	$-47720.00	$-153080.00
Commission	$13295.00	$3210.00	$10085.00
Profit Factor	0.79	1.12	0.69
Cumulative Profit	$-41305.00	$5640.00	$-46945.00
Max. Drawdown	$-45300.00	$-5400.00	$-46945.00
Sharpe Ratio	-1.21	0.31	-1.67
Start Date	1.1.2013		
End Date	25.9.2013		
Total # of Trades	2659	642	2017
Percent Profitable	32.61%	38.32%	30.79%
# of Winning Trades	867	246	621
# of Losing Trades	1792	396	1396
Average Trade	$-15.53	$8.79	$-23.27
Average Winning Trade	$183.96	$216.91	$170.91
Average Losing Trade	$-112.05	$-120.51	$-109.66
Ratio avg. Win / avg. Loss	1.64	1.80	1.56
Max. conseq. Winners	6	4	5
Max. conseq. Losers	16	14	15
Largest Winning Trade	$1645.00	$1645.00	$1475.00
Largest Losing Trade	$-245.00	$-245.00	$-245.00
# of Trades per Day	10.19	2.46	7.73
Avg. Time in Market	39.8 min	61.0 min	33.1 min
Avg. Bars in Trade	5.3	5.5	5.2
Profit per Month	$-4826.83	$659.08	$-5485.91
Max. Time to Recover	257.06 days	125.19 days	260.25 days

Figure 6.3: Strategy performance with ABCDE sampling.

Sampling based on histogram

The following experiment solves the problem with very limited sampling. Data were sampled according to histogram of data. The histogram can be seen in figure 6.4. Table 6.1 shows how data were sampled.

Interval/Value	Word
0.1	„0.1“
0.2	„0.2“
0.3	„0.3“
0.4	„0.4“
0.5	„0.5“
0.6	„0.6“
<0.7;0.8>	„0.75“
<0.9;1.0>	„0.95“
<1.1;1.3>	„1.20“
<1.4;1.8>	„1.60“
<1.9;2.6>	„2.25“
<2.7;inf>	„4.00“

Table 6.1: Histogram sampling

Data with high numbers of occurrences were not sampled and other data were sampled to intervals. The system works similarly to the system in the first experiment. When expected value represents high increase or decrease, then a trade is opened and stays opened until the expected value is too bad for keeping the trade open.

The basic strategy had a limit for opening a trade set to 2 points. The limit for closing a trade was set to -0.2 points. These values are valid for long trades. For short trades, opposite values are used. Stop loss was set to $150 again and minimal improvement in entropy stayed set to 1.003 also. Important training parameters were *-alpha 0.1 -beta 0.0000001 -compression 0 -gradient-cutoff 15 -hidden 30 -class 1 -bptt 4 -bptt-block 10 -direct-order 3 -direct 2* and parameters for testing were *-dynamic 0.1.* All parametere were tuned in the same order as in the previous strategy.

The profit of the basic strategy was $4530 ($26.65 per trade). Even this basic strategy with no optimization has a great result. Tuning has started with setting the limit value for opening a trade. Tests showed that values less than 1.8 give lower profit. Other higher values give still the same profit – $4530, so we have stayed at the 2 points. Another parameter is the limit value for closing a trade. This parameter was changed to zero as this configuration has raised performance of the strategy to $6650. The influence of this parameter on the balance can be seen in figure 6.5

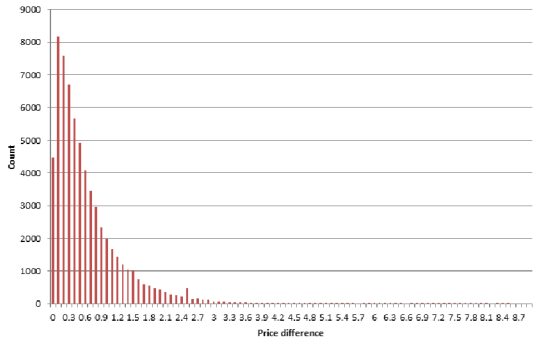


Figure 6.4: Histogram of growths and decreases.

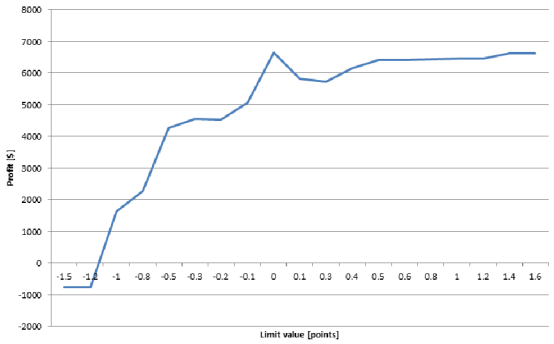


Figure 6.5: Influence of a limit for closing a trade.

Tuning has continued with setting the learning rate in the adaption phase. Thanks to it, the performance has risen again and the profit has reached $7210. Figure 6.6 shows, how this learning rate affects the performance of the strategy. It is obvious, that the impact of this parameter is very strong.

The same situation should arise with setting a stop loss. See figure 6.7 for quite striking results. This figure shows that using higher stop loss leads to worse results. The exact opposite is usual as small movements of the market against opened positions do not close the trade with a loss. The fact, that this strategy requires lower stop loss, makes many things easier. A trader is able to start with a lower input capital and many losses in a row do not have serious negative consequences on the balance or the trader himself. By the way, using no stop loss resulted in the worst results.

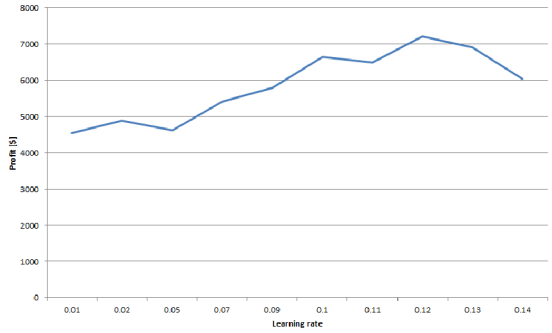


Figure 6.6: Influence of the adaption in run-time.

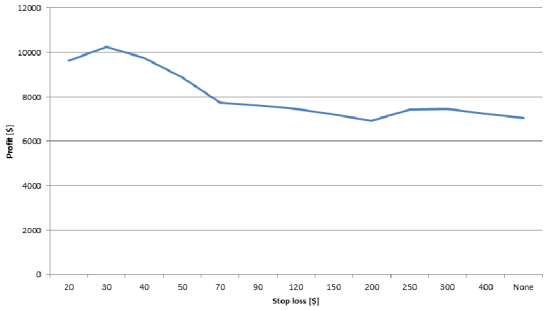


Figure 6.7: Influence of a stop loss.

Stop loss optimization was the last one that significantly increased the profit. Changing sizes of both hidden layers did some differences, but only in negative ones as can be seen in figures 6.8 and 6.9. In case of adaption in run-time, it is the same situation. Setting it to any different value just worsened the performance, see figure 6.10.

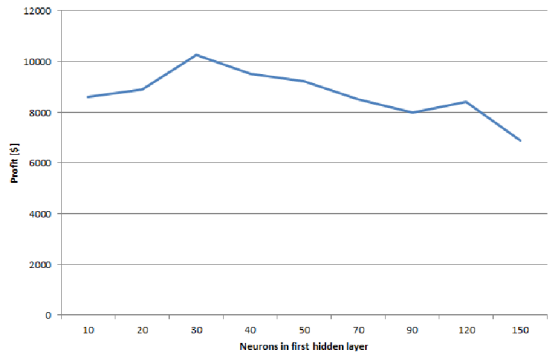


Figure 6.8: Influence of the size of the first hidden layer.

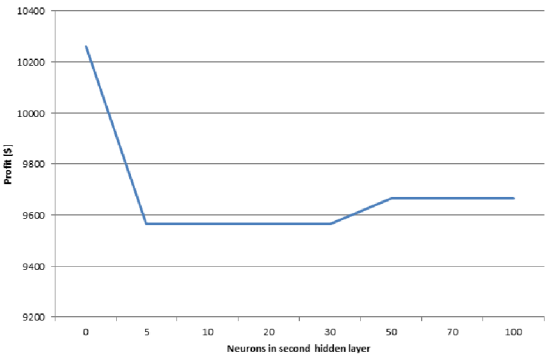


Figure 6.9: Influence of the size of the second hidden layer.

Changing beta had almost none impact to the profit. Results have differed only in the order of dozens of dollars and the profit was worse anyway. The following parameters that were optimized with no success are *bppt*, *bppt-block* and *direct*. The profit were moving down with any change of one of these parameters, but never moved under $7500. A little improvement was brought by a change of *direct-order* parameter. As can be seen from figure

6.11, the impact of this parameter was sometimes very strong, however the improvement is just $200 in the end. $200 difference is probably caused by just a one different trade, thus these little changes do not affect result on testing data much. We can consider these changes as just noise.

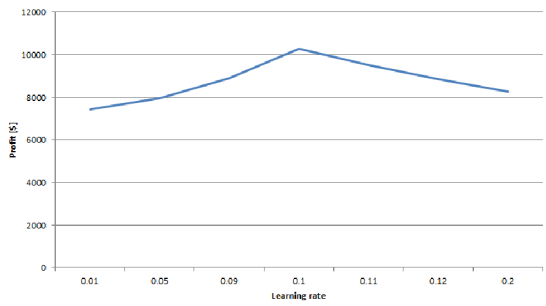


Figure 6.10: Influence of the adaption in run-time.

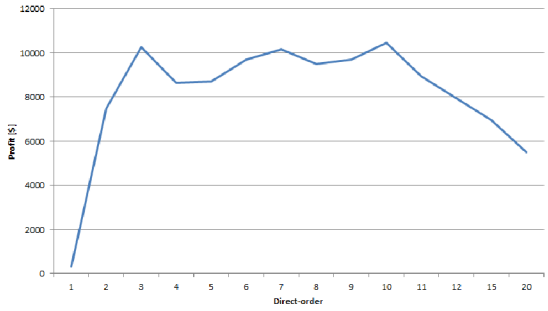


Figure 6.11: Influence of a direct-order parameter.

The last optimization is very similar to previous ones. *Gradient-cutoff* parameter was changed and improved the performance by $500. It should be said, that changes of this parameters were causing only little changes in the profit. Differences while optimizing this parameter were not larger than $1000.

The optimization ended with training parameters *alpha 0.1 -beta 0.0000001 -gradient-cutoff 0.3 -hidden 30 -compression 0 -bptt 4 -bptt-block 10 -direct-order 10 -direct 2* and test parameters *-dynamic 0.12*. The results of this experiment are promising, but not perfect. The system is finally profitable. Even if balance is positive in the end of testing, the whole system is not suitable for real trading. When analyzing the balance equity in figure 6.13, it is obvious that there is no stability. And the average profit per trade is very poor. Bigger slippage can appear and the profitability will be gone. The drawdown is over one third of earned money, which is an indicator that this strategy can be very risky. Nevertheless, there is a huge improvement in performance that tells us that we are moving in the right direction.

6.1.2 Continuous values

In this section, testing the neural network with real-valued input and output will be described. All necessary changes from the original RNN LM toolkit to the neural network, which is able to work with real numbers, are described in 4.2. Tuning and adjusting all the parameters took place in the same way as for other tests before. Only some parameters were removed while changing the network.

Fixed parameters

The first test is based on tuning parameters and letting the strategy run with these fixed values. The strategy logic is still the same as in previous tests.

The initial values of parameters were almost the same as in previous tests. Only default learning rate has been changed, because the network diverged with the default learning rate. Important training parameters were *-alpha 0.01 -beta 0.0000001 -hidden 30 -compression 0 -bptt 4 -bptt-block 10 -gradient-cutoff 15* and parameters for testing were *-dynamic 0.01*.

Performance	All Trades	Long Trades	Short Trades
Total Net Profit	$5700.00	$3850.00	$1850.00
Gross Profit	$12200.00	$7730.00	$4470.00
Gross Loss	$-6500.00	$-3880.00	$-2620.00
Commission	$1000.00	$600.00	$400.00
Profit Factor	1.88	1.99	1.71
Cumulative Profit	$5700.00	$3850.00	$1850.00
Max. Drawdown	$-2060.00	$-1155.00	$-1240.00
Sharpe Ratio	0.60	0.68	0.47
Start Date	1.1.2013		
End Date	25.9.2013		
Total # of Trades	200	120	80
Percent Profitable	24.00%	25.00%	22.50%
# of Winning Trades	48	30	18
# of Losing Trades	152	90	62
Average Trade	$28.50	$32.08	$23.12
Average Winning Trade	$254.17	$257.67	$248.33
Average Losing Trade	$-42.76	$-43.11	$-42.26
Ratio avg. Win / avg. Loss	5.94	5.98	5.88
Max. conseq. Winners	3	5	2
Max. conseq. Losers	20	19	15
Largest Winning Trade	$1625.00	$1625.00	$485.00
Largest Losing Trade	$-45.00	$-45.00	$-45.00
# of Trades per Day	1.09	0.66	0.46
Avg. Time in Market	59.2 min	61.6 min	55.7 min
Avg. Bars in Trade	0.9	1.0	0.8
Profit per Month	$950.00	$641.67	$326.16
Max. Time to Recover	59.24 days	60.92 days	72.27 days

Figure 6.12: Strategy performance with advanced sampling.

All parameters were tuned in the same order as in the previous strategy. You can see, that learning rate was decreased. That is because higher values lead to divergence [4].

The initial profit of the strategy was poor $1235, but with $137.22 per trade. This result denotes that number of trades was quite small as was already discussed in section 2.4.3, it is not very suitable for testing. The first tuned parameter was the limit for opening a trade. This parameter has very strong influence on results of the strategy which is illustrated in figure 6.14. The strategy's profit was significantly improved to $6170 at the cost of profit per trade, which decreased a little to $118.65. Changing the limit for closing a trade did not make any improvement as its default value gives best result. It can be seen in figure 6.15. The same situation happened when adaption in run-time was tuned. Changing this parameter influenced profit significantly, but no improvement has been achieved as can be seen in figure 6.16. Let us move to the following parameter which is the stop loss. Stop loss brought various results and some improvement too. See figure 6.17 for more details. The profit raised to nice $7680 and the profit per trade increased almost two times to $192 with stop loss $380 high. Let us just note that high balance would be needed if trading was done with this stop loss.

The size of the hidden layer was tuned next. The influence of this parameter can be seen

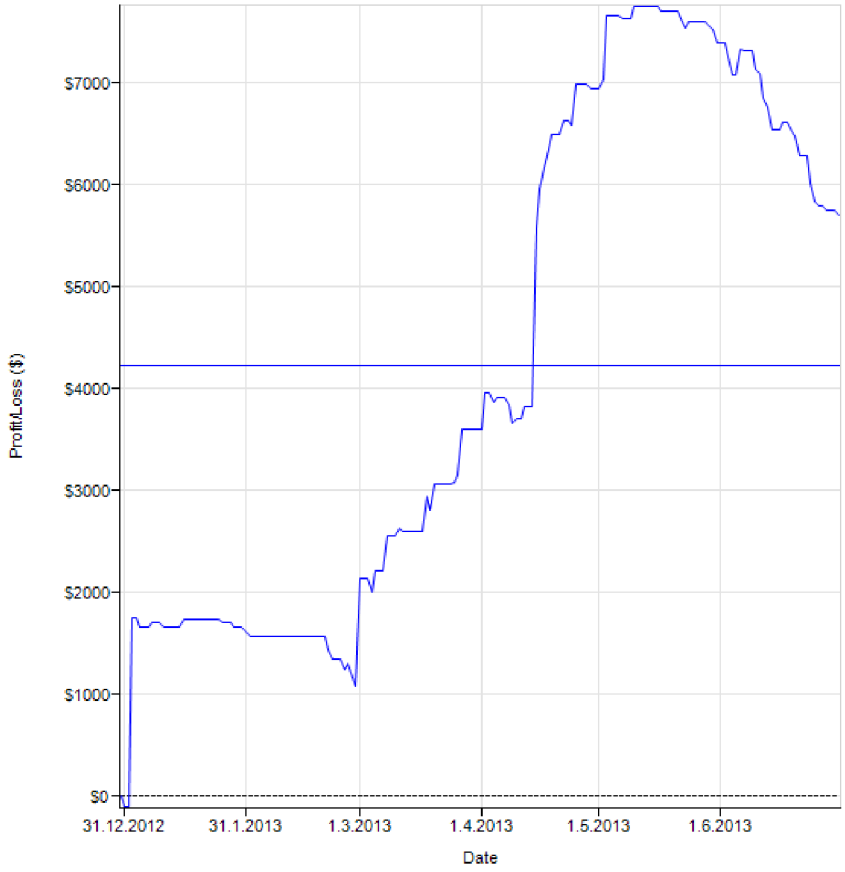


Figure 6.13: Equity with advanced sampling.

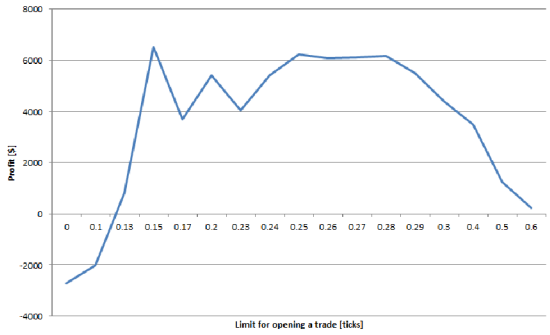


Figure 6.14: Influence of the limit for opening a trade.

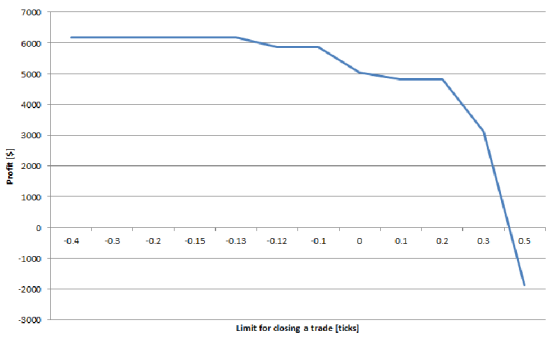


Figure 6.15: Influence of the limit for closing a trade.

in figure 6.18. It is clear that the size of the hidden layer is very important, nevertheless, the best profit was obtained by hidden layer with 27 neurons. Only 3 neurons were removed. A little surprise came with setting the size of the second hidden layer. Setting it to any value greater than zero led to zero trades executed.

The size of the hidden layer was the last parameter which caused some improvement. Setting learning rate had strong influence on the results (Fig. 6.19), but no improvement

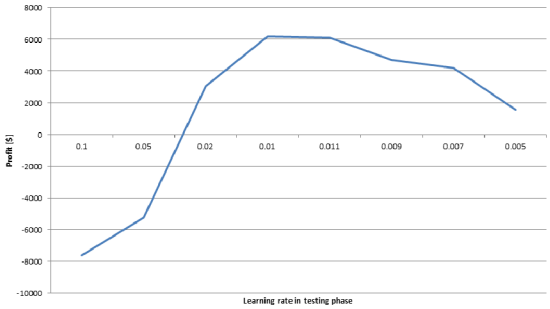


Figure 6.16: Influence of the adaption in run-time.

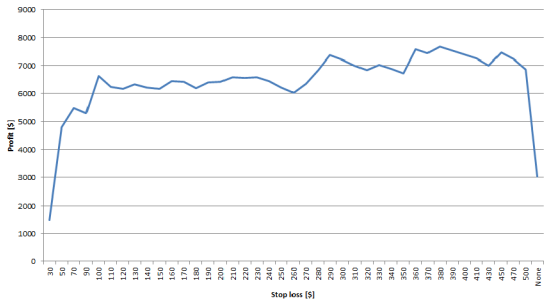


Figure 6.17: Strategy performance with various stop loss.

was obtained. Parameter beta did not any change and changing bptt and bptt-block lead to decrease of profit. Last parameter to adjust is gradient-cutoff. Its influence is similar to other tuned parameter - strong influence but no improvement (Fig. 6.20).

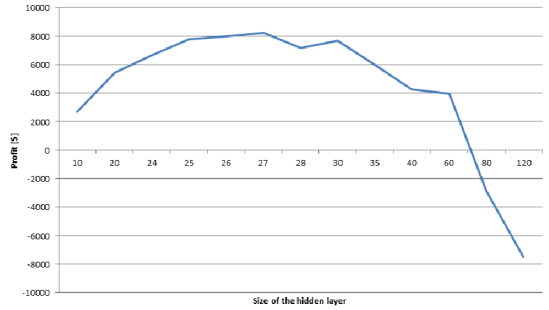


Figure 6.18: Influence of size of the first hidden layer.

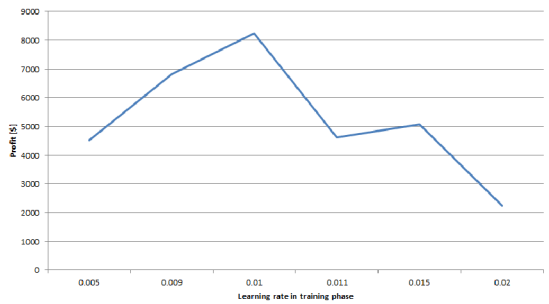


Figure 6.19: Influence of a learning rate in training phase.

The final parameters are *-alpha 0.01 -beta 0.0000001 -hidden 27 -compression 0 -bptt 4 -bptt-block 10 -gradient-cutoff 15* and test parameters are *-dynamic 0.01*. Tuning ended with profit $8235 and $222.57 per trade. Number of trades is lower than it should be. On tested data a little worse results were expected. But results of testing were worse significantly as you can see in figures 6.22 and 6.21. Only $1500 profit was achieved. Why this happened? The first reason is definitely the low number of trades. This is caused by strategy design and lack of data. As data needs to be uniform for all tests, more data will not be added. The strategy design could not be changed too, at least this not will be the first solution. The second reason are fixed parameters. For example, volatility changes through days, month and years but stop loss is firmly set to some value. Stop loss (and other parameters) should be set according to actual volatility. This improvement will be tested in the following section.

Working with volatility

The next test tries to eliminate the negative influence of fixed parameters. Volatility is changing ale the time; sometimes, stop loss $100 is high enough and sometimes it is so small that this stop loss is hit almost immediately. The same situation happens with limits for opening and closing a trade. When market moves are too big, this limit would be hit almost all the time.

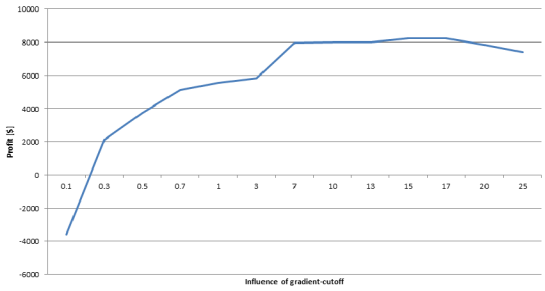


Figure 6.20: Influence of gradient-cutoff.

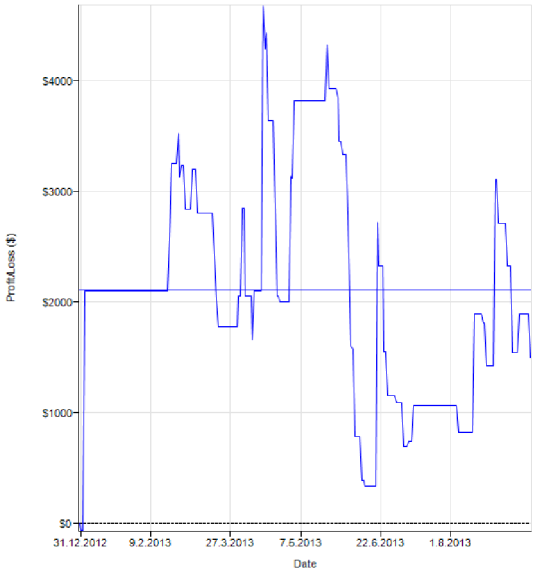


Figure 6.21: Equity with number-based neural network.

The most important input for adapting of parameters according to actual volatility is ATR (average true range) indicator:

$$ATR_t(n) = \frac{(n-1)ATR_{t-1} + TR_t}{n}, where \tag{6.1}$$

$$TR_t = max(high_t - low_t, \mid high - close_{t-1} \mid, \mid low - close_{t-1} \mid) \tag{6.2}$$

Simply put, this indicator measures volatility of N bars[2] back. The higher the value of indicator, the higher the volatility is and vice-versa. The strategy should undergo a few changes. Stop loss and opening and closing limits are multiplied by actual ATR values. This ensures higher stop loss and limits when volatility rises.

The initial values of parameters stayed set to the values from the previous testing: test parameters *-dynamic 0.01* and training parameters *-alpha 0.01 -beta 0.0000001 -hidden 27 -compression 0 -bptt 4 -bptt-block 10 -gradient-cutoff 15*. Stop loss is set to $380 and the period of ATR indicator is set to 14. This means that it is counted from 14 previous bars. 14 bars represent a history of 90 minutes as 5 minute sampling is used.

The basic result of this strategy was $4265 and $170.6 per trade. At first, the limit for opening a trade was changing. This parameter had a very interesting influence. Because of that, the number of trades had to be taken in consideration too. As can be seen in figure 6.23, the highest profit (around $7400) was achieved a few times. The value 0.161 was chosen, because of still high number of trades. In the other cases, number of trades was very low for tuning the strategy. Profit per trade is still great – $82.33. The second tuned parameter was the limit for closing a trade. Adjusting the parameter was not so much interesting as the previous one. The profit was not changing much as can be seen in figure 6.24. This parameter has increased the profit to $9260.

[2]Bar = one drawn line, see 2.1.

Performance	All Trades	Long Trades	Short Trades
Total Net Profit	$1500.00	$-2640.00	$4140.00
Gross Profit	$16880.00	$4490.00	$12390.00
Gross Loss	$-15380.00	$-7130.00	$-8250.00
Commission	$340.00	$170.00	$170.00
Profit Factor	1.10	0.63	1.50
Cumulative Profit	$1500.00	$-2640.00	$4140.00
Max. Drawdown	$-4340.00	$-5200.00	$-2495.00
Sharpe Ratio	0.13	-0.23	0.53
Start Date	1.1.2013		
End Date	25.9.2013		
Total # of Trades	68	34	34
Percent Profitable	29.41%	23.53%	35.29%
# of Winning Trades	20	8	12
# of Losing Trades	48	26	22
Average Trade	$22.06	$-77.65	$121.76
Average Winning Trade	$844.00	$561.25	$1032.50
Average Losing Trade	$-320.42	$-274.23	$-375.00
Ratio avg. Win / avg. Los	2.63	2.05	2.75
Max. conseq. Winners	4	3	3
Max. conseq. Losers	14	12	7
Largest Winning Trade	$2575.00	$2155.00	$2575.00
Largest Losing Trade	$-395.00	$-395.00	$-395.00
# of Trades per Day	0.26	0.13	0.17
Avg. Time in Market	237.4 min	263.5 min	211.2 min
Avg. Bars in Trade	35.7	38.7	32.7
Profit per Month	$175.96	$-309.69	$634.52
Max. Time to Recover	153.73 days	194.77 days	85.00 days

Figure 6.22: Strategy performance with number-based neural network.

Changing adaption in run-time had very strong influence as always (see figure 6.25). Nevertheless, higher profit was not achieved and this parameter stayed set to 0.01. Stop loss tuning has brought nice improvement as traditionally. In figure 6.26 the ideal stop loss value of $212 can be seen. Thanks to that, the profit has risen to excellent $12105 and stop loss was lowered, which does not place so much high demands on the input capital.

Tuning the ATR period was the next move, but changing it did not bring any improvement (Fig. 6.27). The most likely reason is that the values of previous already tuned parameters are set to work with the default ATR period. In figure 6.28, almost the same influence can be seen. This time, this is influence of the size of the first hidden layer. Changing it did not increase the profit again. The influence of the second hidden layer is the same as in the previous test. With turning on the second hidden layer, no trades were made and the profit stayed at $0.

Other parameters changed the resulting profit very slightly. The influence of learning rate in the training phase can be seen in figure 6.29. As the influence is strong, no improvement was made. Changing beta, bptt, and bptt-block had very similar results – only slight influence on profit, but always in a negative way. The last improvement was made

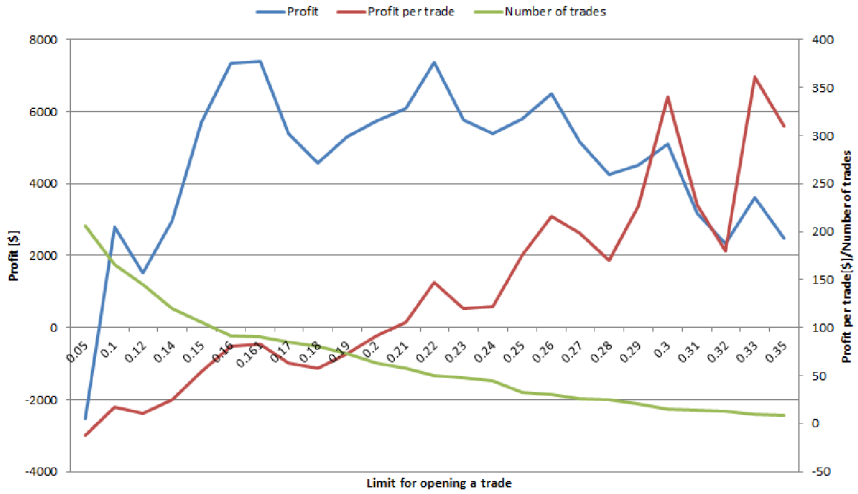


Figure 6.23: Influence of the limit for opening a trade.

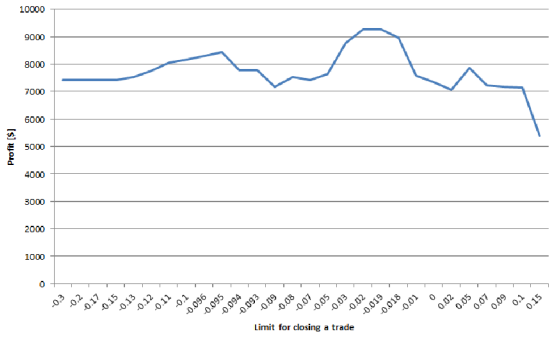


Figure 6.24: Influence of the limit for closing a trade.

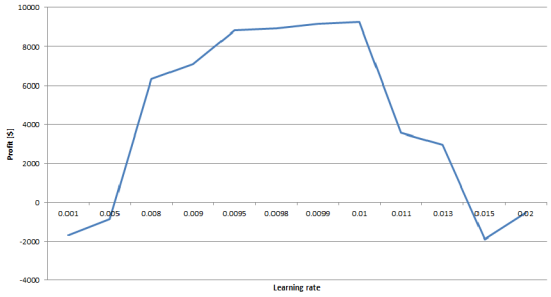


Figure 6.25: Influence of adaption in runtime.

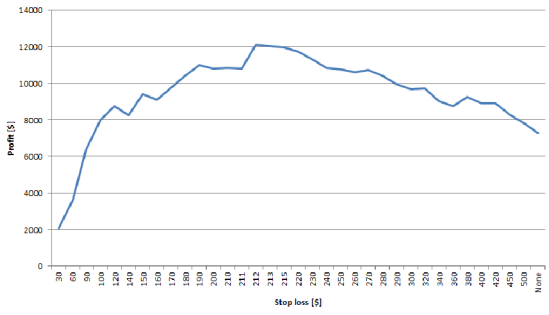


Figure 6.26: Influence of stop loss.

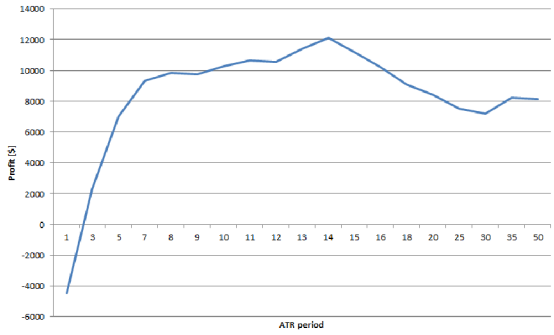


Figure 6.27: Influence of ATR period.

by gradient cutoff parameter. The improvement was very slight and improved the profit to $12565. This profit is the highest profit achieved in training and tuning the strategy.

The final training parameters are *-alpha 0.01 -beta 0.0000001 -hidden 27 -compression 0 -bptt 4 -bptt-block 12 -gradient-cutoff 6* and test parameters are *-dynamic 0.01*. The results on the test data show a big step forward. Strategy performance and balance equity

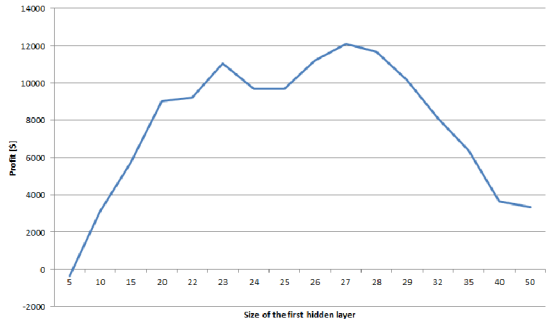


Figure 6.28: Influence of size of the first hidden layer.

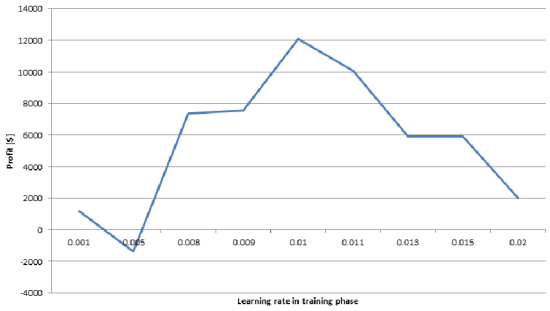


Figure 6.29: Influence of learning rate in training phase.

can be seen in figures 6.31 and 6.32. The profit is now $5170, which is five times better than profit of the previous test. Unfortunately, profit per trade is higher only slightly. This parameter should be much higher in real trading. Equity of this strategy is much nicer and shows growing trend. Also, the number of trades can be considered as appropriate.

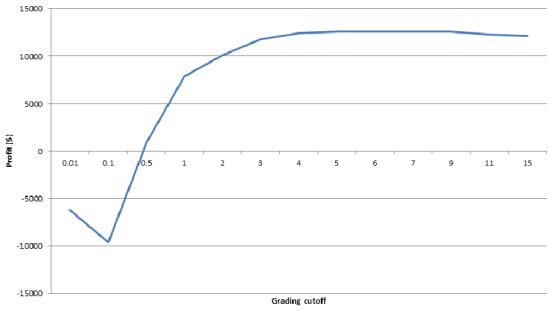


Figure 6.30: Influence of gradient cutoff.

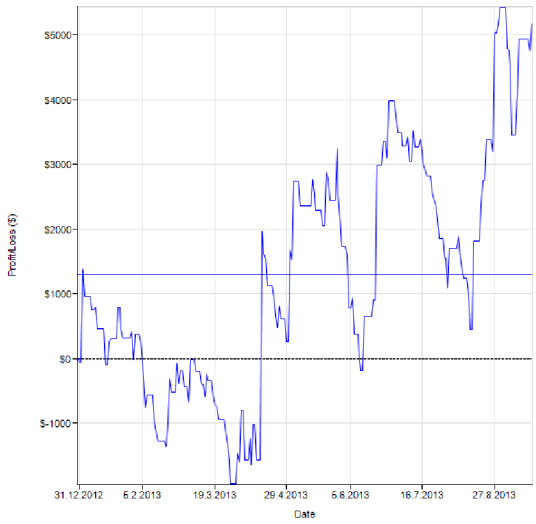


Figure 6.31: Equity with number-based neural network with ATR.

Performance	All Trades	Long Trades	Short Trades
Total Net Profit	$5170.00	$925.00	$4245.00
Gross Profit	$32330.00	$15385.00	$16945.00
Gross Loss	$-27160.00	$-14460.00	$-12700.00
Commission	$910.00	$565.00	$345.00
Profit Factor	1.19	1.06	1.33
Cumulative Profit	$5170.00	$925.00	$4245.00
Max. Drawdown	$-3725.00	$-2790.00	$-2600.00
Sharpe Ratio	0.29	0.11	0.28
Start Date	1.1.2013		
End Date	25.9.2013		
Total # of Trades	182	113	69
Percent Profitable	30.77%	30.97%	30.43%
# of Winning Trades	56	35	21
# of Losing Trades	126	78	48
Average Trade	$28.41	$8.19	$61.52
Average Winning Trade	$577.32	$439.57	$806.90
Average Losing Trade	$-215.56	$-185.38	$-264.58
Ratio avg. Win / avg. Loss	2.68	2.37	3.05
Max. conseq. Winners	4	3	4
Max. conseq. Losers	13	8	8
Largest Winning Trade	$3545.00	$1435.00	$3545.00
Largest Losing Trade	$-545.00	$-405.00	$-545.00
# of Trades per Day	0.70	0.43	0.28
Avg. Time in Market	198.3 min	209.5 min	180.0 min
Avg. Bars in Trade	26.9	28.5	24.4
Profit per Month	$604.16	$108.09	$524.18
Max. Time to Recover	99.92 days	238.04 days	89.00 days

Figure 6.32: Strategy performance with number-based neural network with ATR.

Chapter 7

Conclusion

This thesis shows that it is possible to create a functional trading strategy based on recurrent neural networks. The neural network can work with both symbolic (discrete) and real (continuous) types of inputs. When tuning the network and the strategy, the initial values of parameters can affect the result significantly. The tests have shown, that only about a half of tuned parameters (input and output limit, both learning rates and stop loss) were crucial for the performance. The second half was then adapted to previously tuned values.

The reached profit was not bad: more than $500 per month with simple tuning of parameters. The potential of this way of trading is however much greater, as there are many variables which affect the final result - neural network setting, strategy parameters, strategy logic, used indicators etc. Each of them can turn a loosing strategy into a profitable one and vice versa.

Some ideas that have a potential to improve the results are described in the following section.

7.1 Future work

At the end, let us mention a few tips and ideas for the future work.

- Involvement of genetic algorithms in the tuning phase.
- In our approach, only one parameter was tuned at the time. A better solution would be to tune several (or all) parameters at the same time, with the profit as criterion, probably using one of the gradient descent methods.
- Use profit metric in RNN. Training of RNN was done with the classical criterion of Maximum Entropy, while the final criterion is the profit. It would be nice to re-do the mathematical framework to propagate the final metric (the profit) directly to the training of the RNN.
- Using more neural network models. The prediction can be made by merging results of several models.
- Integrating all price indicators. In the current state, only *close* price is analyzed, but *open*, *high*, *low* prices are ignored.

- Using various indicators. The price is only one of possible inputs to the network. Feeding the network with the values of various other indicators, such as EMA, MACD, RSI, MFI, Williams %R, PSAR[1] etc., should improve the results.
- Choosing ideal timeframe. Chosen timeframe was 5 minutes as a compromise between financial demands and reducing data noise.
- Looking at the actual price. The trade is controlled strictly by the network outputs now. Looking at the actual price could prevent some errors caused by the network.
- Using limit orders. Every opening of a trade is made when the line is drawn at the *close* price of this line. Limit orders allows setting a price, where a trade should be open. It can prevent opening a trade too late.
- Preventing closing a trade to early. When bad price is predicted, the trade is immediately closed. Every trend consists of increases and decreases. The trade should withstand some moves against the position.

[1]EMA – exponential moving average, MACD – moving average convergence divergence, RSI – relative strength index, MFI – money flow index, PSAR – parabolic stop and reverse

Bibliography

[1] J. D. McCaffrey. Neural network classification, categorical data, softmax activation, and cross entropy error. [online]. 2012 [cit. 2014-05-04]. Dostupné z: http://jamesmccaffrey.wordpress.com/2011/12/17/neural-network-classification-categorical-data-softmax-activation-and-cross-entropy-error/.

[2] T. Mikolov. *Statistical language model based on neural networks [online]*. PhD thesis, 2012.

[3] P. Podhajský T. Nesnídal. *Obchodování na komoditních trzích*. Grada, Praha, 2 edition, 2007.

[4] G. Orr. Linear neural networks. [online]. 2013 [cit. 2014-05-04]. Dostupné z: http://www.willamette.edu/~gorr/classes/cs449/linear2.html.

[5] T. Nesnídal P. Podhajský. *Jak se stát intradenním finančníkem*. Centrum finančního vzdělávání, Praha, 2008.

[6] T. Nesnídal P. Podhajský. *Kompletní průvodce úspěšného finančníka*. Centrum finančního vzdělávání, Praha, 2009.

[7] L. Turek. Manuál technická analýzy, 2008.

[8] I. Vondrák. Neuronové sítě. [online]. 2009 [cit. 2014-05-04]. Dostupné z: http://vondrak.cs.vsb.cz/download/Neuronove_site.pdf.

[9] L. Williams. *Dlouhodobá tajemství krátkodobých obchodů*. Centrum finančního vzdělávání, Praha, 2007.

Appendix A

CD content

- /Data – Contains tick data of TF Mini Russel 2000.
- /Test – This is folder containing all sources necessary for repetition of tests. Every test contains one zip file and one folder. Zip file is strategy for Ninja Trader and folder contains project for CodeBlocks.

Appendix B

Manual

Installation and usage of the neural network and the trading strategy will be described here.

B.1 Prerequisite installation

Download and install Ninja Trader 7 (http://www.ninjatrader.com/download-registration.php) and CodeBlocks (http://www.codeblocks.org/downloads/binaries). Use Ninja Trader support and forum for troubleshooting. You will need to enter valid email to receive license key. CodeBlocks software is used for building the network binaries on Windows platform. The whole testing took place on Windows 7.

B.2 Preparation

In this section, preparation of Ninja Trader and neural network will be described.

B.2.1 Ninja Trader

Open Ninja Trader and select *Tools* → *Historical Data Manager*. Set everything according to figure B.1 and click *Start Import*. Choose file *TF ##-##.txt* from *Data* directory. Let Ninja Trader process the data. After that, select *Tools* → *Instrument Manager*. In *Name* field, enter *TF* and click *Search*. Choose Mini Russel 2000 instrument from a table. Set *Expiry* to *##-##* and click on the left arrow. Now *TF ##-##* appeared on the left in *Instrument list*. For the help, see figure B.2. Now click again on the Mini Russel 2000 instrument in the table and click on *Edit*. Choose *Misc* tab and for *Simulator* set *Minimum commission* to 2.5. This you can see in figure B.3.

For importing trading strategy, click *File* → *Utilities* → *Import NinjaScript*. Select relevant *rnnlm.zip* file from *Test* folder and the folder you want to use. After import, select *Tools* → *Edit NinjaScript* → *Strategy* and choose *RNNLM*. Opened window contains the code of the strategy. You can edit anything you want. In the end click on *Compile* icon to compile the strategy.

B.2.2 Neural network

Go to folder of the relevant test in *Test* folder. Open folder *rnnlm* and open project via *rnnlm.cbp*. This project contains the whole code of the neural network. Compile the

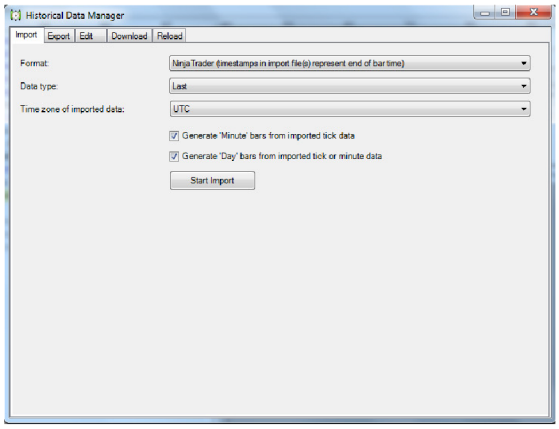


Figure B.1: Settings of *Historical Data Manager*.

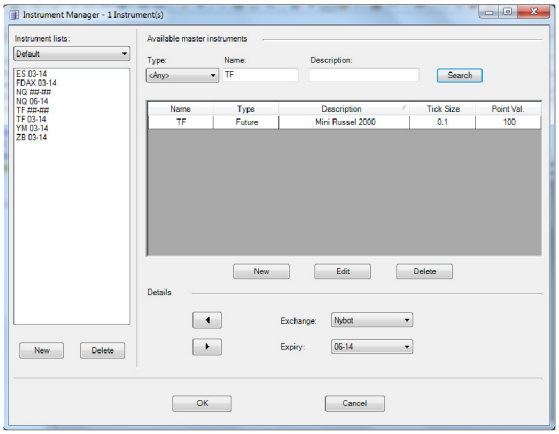


Figure B.2: *Instrument list.*

network with *Build* → *Rebuild*.

B.3 Testing

Select *File* → *New* → *Strategy Analyzer* and new window appears. Now click with the right mouse button on *TF ##-##* in the left panel and choose *Backtest*. In the right expander of *Backtest* tab, choose *RNNLM* as *Strategy*. In section *Parameters*, you can define input parameters for the strategy:

- AtrPeriod – Period for ATR indicator. This parameter is included in the last test only.
- LogFile – Path to text file where the strategy can log important informations and errors. In the strategy, you can write into log by calling *Log()* method.
- PipeName – Name of the pipe for inter-process communication between the strategy and the neural network.
- RnnlmFile – Path to exe file of the neural network. This file is the output of a CodeBlocks project.
- StopLoss – Defines height of the stop loss in dollars.
- Test – A switch between training and testing of the network. Set it to *False* for training the network. Set it to *True* and the strategy will be opening trades.
- TestParameters – Parameter used for starting the network in testing mode.
- TrainParameters – Parameter used for starting the network in training mode.

The next section *Data series* defines input data series for testing. A 5 minute timeframe was used. Section *Time frame* is important. A time range where the strategy will be tested can be set here. Just check that *Include commission* is set to *True* and *Slippage* is set to 1.

Revise all parameters, set *Test* parameter to *False* and choose time range, where the network will be trained. Click *Run Backtest* in order to train the network. After training

set *Test* parameter to *True* and choose time range, where the strategy will trade and click *Run Backtest* again. After that, you can see result of testing with current settings.

This is illustrated by figure B.4.

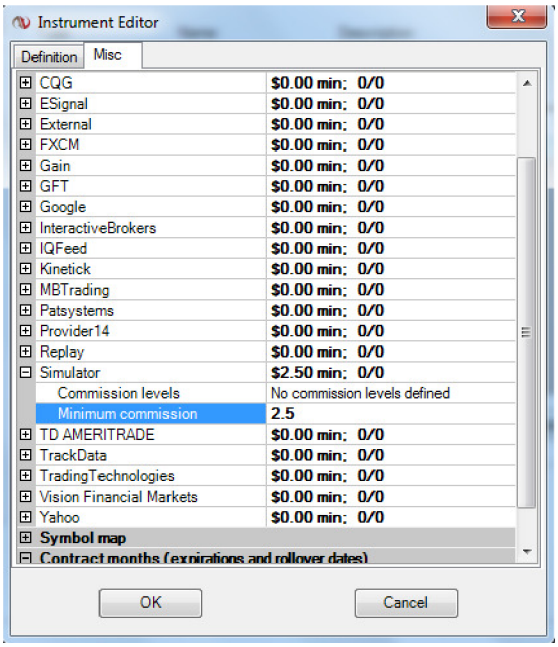


Figure B.3: Settings of *Commission*.

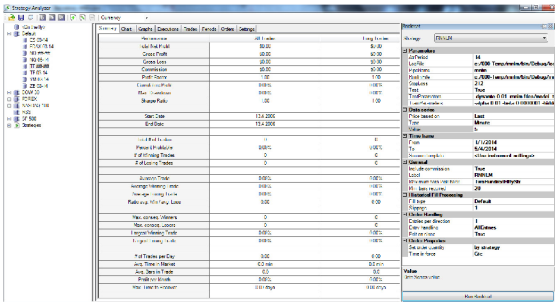


Figure B.4: *Strategy analyzer*.

B.4 Real trading usage

For real trading, you have to get some data feed, for example the free data feed from CQG [1]. Select *File* → *Connect* and choose data feed. You can select *Simulated Data Feed* for random data. Now select *File* → *New* → *Chart*. Choose *TF ##-##* and select *New*. In the right panel, set the input data series. The most important is *Period* section where you define type of chart. See figure B.5 for help. In the end, click on *OK*.

Click on *Strategies icon* select relevant strategy. Click on *New* and set strategy parameters. Do not forget to set *Min. bars required* parameter. This parameters defines after how many bars the strategy will start working. Set *Enabled* to *True* and click OK. Now the strategy should start trading.

In the main window, click on *Strategies* tab. Running strategy should by colored with green. If it is color with yellow as in figure B.6, go to *Tools* → *Options* → *Strategies* → *NinjaScript* and check settings. See http://www.ninjatrader.com/support/helpGuides/nt7/index.html?strategies_tab2.htm for help.

[1] http://www.ampclearing.com/ninjatrader_cqg.php

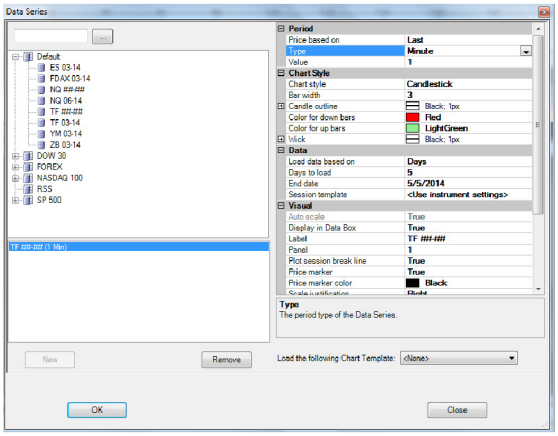


Figure B.5: Settings of strategy in real usage.

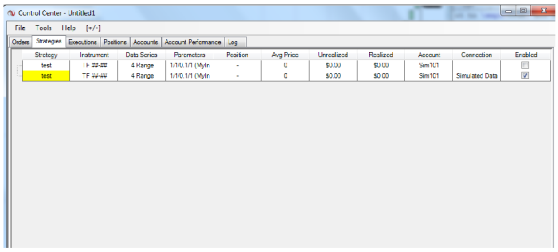


Figure B.6: Strategy state.