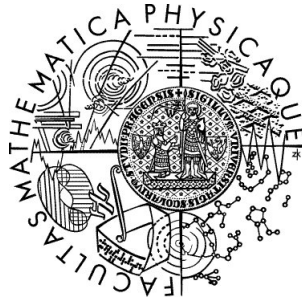


Charles University in Prague  
Faculty of Mathematics and Physics

## Master thesis



Larysa Aharkava

## Artificial neural networks and self-organization for knowledge extraction

Department of Theoretical Computer Science and Mathematical Logic

Thesis supervisor: doc. RNDr. Iveta Mrazova, CSc.,

Study program: Informatics

2010

I would like to thank my supervisor, doc. RNDr. Iveta Mrazova, CSc., for her patience and valuable comments.

I would also like to express my gratitude to all those people who created great tools - such as Latex, MetaTrader platform, MQL language, Viscovery SOMine and Visual Studio - that simplify people lives and make research a pleasure.

I declare that I wrote that Master thesis by myself and using only referenced sources. I agree with the thesis lending.

In Prague

Larysa Aharkava

# Contents

<b>1</b>	<b>Introduction</b>	<b>7</b>
1.1	Motivation and goals . . . . .	7
<b>2</b>	<b>Artificial neural networks</b>	<b>10</b>
2.1	Biological neural networks . . . . .	11
2.2	Mathematical model of the neuron . . . . .	12
2.3	Multi-layer perceptron . . . . .	14
2.3.1	Back propagation network training algorithm . . . . .	15
2.3.2	Training algorithm - practical aspects . . . . .	18
2.4	RBF networks . . . . .	19
2.4.1	RBF network architecture . . . . .	20
2.4.2	Learning of the RBF networks . . . . .	22
2.5	Conclusions . . . . .	24
<b>3</b>	<b>Kohonen's self-organizing maps</b>	<b>26</b>
3.1	Self-organizing maps general overview . . . . .	27
3.2	Self-organizing maps training algorithm . . . . .	28
3.2.1	Initialization process . . . . .	29
3.2.2	Competition process . . . . .	29
3.2.3	Cooperation process . . . . .	30
3.2.4	Adaptation process . . . . .	31
3.2.5	Parameters' selection . . . . .	33
3.2.6	Training algorithm . . . . .	33
3.3	Other clustering methods . . . . .	35
3.3.1	K-means clustering . . . . .	35
3.3.2	Fuzzy c-means clustering . . . . .	36
3.4	Conclusions . . . . .	37

<b>4</b>	<b>A general introduction to financial markets</b>	<b>38</b>
4.1	Financial market basics . . . . .	38
4.2	Market analysis techniques . . . . .	39
4.2.1	Fundamental analysis . . . . .	39
4.2.2	Technical analysis . . . . .	40
4.3	Introduction to trading . . . . .	42
4.4	MetaTrader platform . . . . .	48
4.4.1	MQL4 language . . . . .	49
<b>5</b>	<b>Neural networks for Forex trading</b>	<b>51</b>
5.1	Selection of input and output variables . . . . .	51
5.2	Data preprocessing . . . . .	53
5.3	Model construction . . . . .	54
5.4	Other forecasting techniques . . . . .	57
<b>6</b>	<b>Self-organizing maps and Forex</b>	<b>59</b>
6.1	Example: Advisors' profiling . . . . .	59
6.1.1	Data for the profiling . . . . .	60
6.1.2	Results . . . . .	60
6.1.3	Conclusions . . . . .	63
6.2	Example: SOM-based Forex advisor . . . . .	63
6.2.1	Data for classification . . . . .	63
6.2.2	Clusters . . . . .	64
6.2.3	Results . . . . .	67
6.2.4	Conclusions and future work . . . . .	72
<b>7</b>	<b>FXANN - A Forex trading model</b>	<b>73</b>
7.1	FXANN model . . . . .	73
7.1.1	Input data . . . . .	74
7.1.2	Output data . . . . .	75
7.1.3	Training and testing sets . . . . .	77
7.1.4	Adjusted learning . . . . .	77
7.2	FXANN tool . . . . .	78
7.2.1	FXANN settings . . . . .	80
7.2.2	FXANN report . . . . .	81
7.2.3	FXANN data export . . . . .	83
7.2.4	FXANN requirements . . . . .	83
7.3	Experimental simulation . . . . .	83
7.3.1	Observations . . . . .	84

7.3.2	Selected model . . . . .	85
7.3.3	Prediction results . . . . .	87
7.3.4	Simulation results . . . . .	90
7.3.5	Model evaluation . . . . .	91
7.4	Conclusions . . . . .	92
<b>8</b>	<b>Conclusions and future work</b>	<b>93</b>
<b>Appendices</b>		
<b>A</b>	<b>Technical indicators</b>	<b>95</b>
A.1	Commodity Channel Index (CCI) . . . . .	95
A.2	Relative Strength Index (RSI) . . . . .	95
A.3	Average Directional Movement Index (ADX) . . . . .	95
A.4	Momentum . . . . .	99
A.5	Standard Deviation (StdDev) . . . . .	99
A.6	Average True Range (ATR) . . . . .	99
A.7	Bill Williams' Accelerator/Decelerator (AC) . . . . .	103
A.8	Bill Williams' Awesome oscillator (AO) . . . . .	103
A.9	Bill Williams' Accumulation/Distribution (AD) . . . . .	105
	<b>Bibliography</b>	<b>106</b>

Title: Artificial neural networks and self-organization for knowledge extraction

Author: Larysa Aharkava

Department: Department of Theoretical Computer Science and Mathematical Logic

Thesis supervisor: doc. RNDr. Iveta Mrázová, CSc.,

Supervisor's email address: iveta.mrazova@mff.cuni.cz

**Abstract:** Neural networks are widely used for financial time series prediction. However, the future values' prediction has its drawbacks and often cannot be converted to the effective and profitable trading system. In that thesis I will describe several different types of neural networks. Then, I will propose and evaluate on real series data two different approaches based on Kohonen's self-organizing maps and back propagation networks of how to use those networks for creating successful and profitable trading models. Also, I will give a general overview of the Forex market (Foreign exchange market) and neural networks' usage within that market.

Keywords: neural networks, self-organizing maps, financial series prediction, Forex

Název práce: Samoorganizace a uměle neuronové sítě pro extrakci znalostí

Autor: Larysa Aharkava

Katedra: Katedra teoretické informatiky a matematické logiky

Vedoucí: doc. RNDr. Iveta Mrázová, CSc.,

Email vedoucí: iveta.mrazova@mff.cuni.cz

**Abstrakt:** Neuronové sítě jsou často využívány k predikci finančních časových řad. Předpovídání budoucích hodnot však skrývá řadu problémů, které často znemožní vytvoření profitabilního obchodního systému. V této diplomové práci jsem popsala dva typy neuronových sítí - Kohonenovy mapy a sítě založené na algoritmu zpětného šíření. Na základě těchto dvou přístupů jsem zkonstruovala a následně otestovala dva obchodní modely. V obecnější rovině jsem se také věnovala popisu specifik Forexu (Foreign exchange market) a možnostem využití neuronových sítí na těchto trzích.

Klíčová slova: neuronové sítě, Kohonenovy mapy, algoritmus zpětného šíření, predikce finančních časových řad, Forex.

# Chapter 1

## Introduction

### 1.1 Motivation and goals

In recent years artificial neural networks (ANN) converted from a theoretical approach to the widely-used technology with successful applications to different problems. The most of these applications are related to the pattern recognition and neural networks are often used as non-linear approximators that exhibit a high tolerance to imprecision and perform well even in noisy environments. Those qualities also make neural networks suitable for financial problems domains, and particularly for financial trading.

There have been numerous attempts to apply neural networks to the developing of the trading systems that can not only predict future values of some goods, but also extract profits from those predictions. Most of those attempts were formally and statistically successful, but because of different reasons did not perform well on real-time financial trading and therefore remained just a pure theoretical research. But there is at least one result that proved to be extremely efficient on the real financial market.

Olexandr Topchylo is the winner of the *Automated trading championship 2007*. He developed an ANN-based automated trading program that was able to generate more than 1200% of gross profit in three months. The program started with 10 000\$ (standard starting amount for the championship) and in less than three month ended up with more than 130 000\$. After the championship, he started using that program for individual automated trading. And according to his trading reports, it is still very profitable.

Details of the methodology that Olexandr has used are unknown - as every successful trader, he will not reveal his secrets. But his example shows that creating a profitable advisor using ANN is possible. And that fact became the basis for the main objectives of that thesis.

Therefore, the main goals of that thesis can be stated as:

1. Explain and evaluate different types of neural networks.

2. Explore the possibilities of using neural networks in automated Forex trading.
3. Create and evaluate ANN-based Forex trading model with respect to noise robustness and its ability to approximate the input space. Compare that model to alternative models.
4. Examine different approaches to automated trading that are based on different types of neural networks.

The thesis is divided into eight chapters. The second chapter gives an introduction to the one of the oldest models of artificial neural networks - a multi-layer perceptron (MLP). There I will explain the model of the artificial neuron, how can artificial neurons be organized into network and how that network can be trained to be able to recognize patterns in some particular domain. In that chapter I will also give an overview of the similar model - so-called RBF network and compare it to MLP.

Chapter 3 deals with the complete different types of ANN - networks that do not recognize patterns, but rather group input space items and divide them into clusters. In that chapter I will describe three well-known methods that perform that task - Kohonen's self-organizing maps, K-means clustering and Fuzzy c-means clustering. Kohonen's maps will be described in details as I will use that method in later chapters.

Backgrounds of the financial markets are explained in Chapter 4. That chapter gives an overview of the terminology and processes that are happening on the financial markets and is inevitable for understanding the financial trading domain and problems that I will try to solve.

Chapter 5 provides an overview of how multi-layer neural networks can be applied to the financial trading. It describes common problems that must be solved and based on the cited sources suggest different approaches of how to use neural networks in that domain.

In Chapter 6 I will suggest two Kohonen's maps-based approaches that can be used for the financial market analysis. The first approach will show how to learn from the successful traders by finding out what are their behavioral patterns. The second method will propose how to classify market situations and then use that information for the profit extraction. Both approaches will be tested and analyzed based on real financial data.

In Chapter 7 I will introduce, analyze and implement the framework for the automated financial trading. Its efficiency will also be tested on real financial data.

Finally, in the last chapter I will discuss results of my work and propose some directions for the future research.

The part of the thesis is also the tool that implements the suggested framework and make it easy to use and evaluate. Description of that tool is the part of the Chapter 7 and the



tool itself (including the source code) can be found on the attached CD. The CD content description is also the part of the CD.

# Chapter 2

## Artificial neural networks

A biological neural network is an essential part of human brain. It is a highly complex system with the ability to process large amounts of information simultaneously. Biological networks are able to recognize and process different visual inputs much faster than any modern high-end computer. For example, human brain is able to recognize familiar face in about 100-200 ms, while modern computer requires minutes or even hours for solving the same problem. In general, it is a known fact that in many tasks human brain is much more efficient than computers.

Based on examples and feedback from the "teacher", our brain allows us learning how to distinguish an apple from an orange or recognize letters. And even without the "teacher", we are still able to group similar patterns together. Those and other strengths of human brain challenged scientists to emulate those processes by researching how to use machines for tasks that are common for humans. And one of the concepts that appeared as the result of that research, is the **Artificial neural network (ANN)** concept.

First steps of the ANN theory were made in 1943 by neuro-physiologist Warren McCulloch and mathematician Walter Pitts. They introduced artificial neurons with the threshold<sup>1</sup>, that could be arranged into networks. Several years later, in 1949, a psychologist Donald Hebb designed the first learning rule for artificial neural networks - *Hebb's rule*. Its premise was that if two neurons were active simultaneously, then the strength of the connection between them should be increased.

In subsequent years, the ANN theory made significant progress: in 1960s more neuron architectures - *perceptron* and *adaline* - were introduced. Next years, jointly with more powerful computers, brought even more different types of ANN - self-organizing maps, associative networks, RBF-networks and more. Nowadays, artificial neural networks are widely adopted

---

<sup>1</sup>Later called McCulloch-Pitts neuron. By threshold they meant the following idea: the unit fires only if the input to the neuron is greater than the "threshold value".

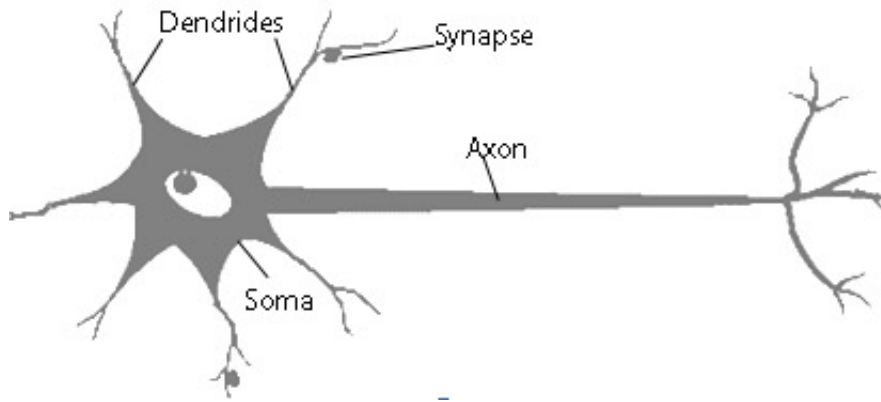


Figure 2.1: **Model of the neuron.**

by different scientific and industrial fields - from medicine to speech recognition. More about ANN history can be found in [9].

## 2.1 Biological neural networks

Scientists started their attempts in discovering human brain secrets long time ago. But despite the fact that in previous centuries the neuroscience made a huge breakthrough, there are still numerous mysteries about how human brain works ([7], [26]).

One of the very first scientific attempts of discovering the main principle of brain functioning was made by Santiago Ramon y Cajal<sup>2</sup> in 1911. He used the method of Camillo Golgi<sup>3</sup> for silver nitrate preparation, and then dyed brain neurons using that substance. Cajal discovered that brain has a cell structure and described brain cells as a polar-sensitive cells that receive signals using ramified sprouts (called **dendrites**) and send signals using formation called **axon**. Axon contacts dendrites of other neurons through special connection points - **synapses** - that adjust strength of the input signal (Fig. 2.1).

Golgi method helped to discover a great variety of neurons which differ in dendrites' branching and axons length. Cajal found numerous differences between the types of neurons. But despite all those distinctions, they all process signals in the same way: signals are represented by electric impulses that are transmitted across a synaptic gap and dendrites to the **soma**. Soma, neuron's body, then somehow sums up the incoming signal. If the result is greater than some threshold value, the cell fires and transmits a signal over its axon to other

---

<sup>2</sup>**Santiago Ramon y Cajal** (1852 - 1934) was a Spanish histologist, physician, pathologist and Nobel laureate.

<sup>3</sup>**Camillo Golgi** (1843 - 1926) was an Italian physician, pathologist, scientist and Nobel laureate.

cells. Also, the synaptic behavior can change in time, so the neuron's behavior changes as well. Cajal came to the conclusion that synapses play an important role in learning. Basically, the whole learning process is based on synaptic weights adjustments.

## 2.2 Mathematical model of the neuron

Biological neural networks are very complex. In contrast, the mathematical model of the network is much more simplified and is based on several assumptions:

1. All neurons are synchronized. That means that the signal passing from one neuron to another takes the same time for all connections. Signal processing is also synchronized and is the same for all neurons.
2. Every neuron has a so-called transfer function which determines neuron's output signal depending on the input signal strength. That function is time-independent.
3. When the signal passes the synapse, it changes linearly, i.e., the signal value is multiplied by some number. That number is called **synaptic weight**.

The very important property of the synaptic weight is that it changes in time. That feature makes it possible for brain to react differently on the same input in different moments. Or, in other words, to learn.

Of course, those assumptions simplify the initial biological neural network very much. For example, brain signal transmission time naturally depends on the distance between neurons. But despite those simplifications, artificial networks still preserve the most important characteristics of biological networks - adaptability and ability to learn.

The first mathematical model of the neuron was introduced more than a half century ago, but did not change much since then. First of all, the neuron is seen as a simple "automate" that transforms input signals into the output signal (Fig. 2.2).

The model functions as follows: inputs of the neuron's synapses receive  $N$  signals  $[X_1, \dots, X_n]$ . Then every synapse makes a linear modification of the signal using its synaptic weight. After that neuron's body (soma) receives signals  $[X_1w_1, \dots, X_nw_n]$  (where  $w_i$  is the corresponding synaptic weight) and sums those signals:

$$S = \sum_{i=1}^n X_i w_i \quad (2.1)$$

Then it applies some given function  $F$  (that is also called **activation function**) and sends the final signal

$$Y = F(S) \quad (2.2)$$

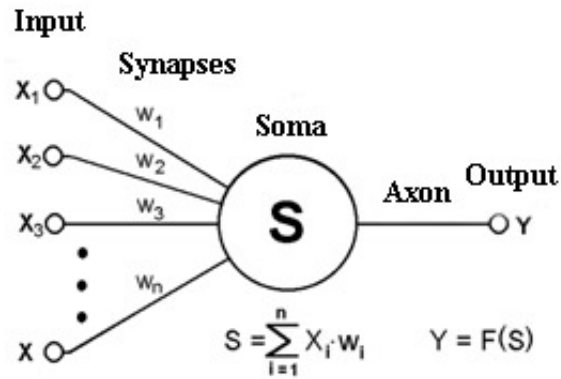


Figure 2.2: **Artificial neuron model.**

to the output.

There are different functions, that are commonly used as activation functions. In a network, it is not necessary to use the same function for all the neurons. However, it's a common practice.

In most cases activation functions are non-linear. Otherwise, the whole network will implement some linear transformation and will be equivalent to only one artificial neuron - perceptron.

Most common activation functions are listed below:

Function name	Formula	Values range
Linear	$F(S) = kS, k \in \mathbb{R}_+$	$(-\infty, \infty)$
Semi linear	$F(S) = \begin{cases} kS, S > 0, k \in \mathbb{R}_+ \\ 0, S \leq 0 \end{cases}$	$[0, \infty)$
Sigmoid	$F(S) = \frac{1}{1+e^{-\alpha S}}$	$(0, 1)$
Bipolar sigmoid	$F(S) = \frac{2}{1+e^{-\alpha S}} - 1$	$(-1, 1)$
Hyperbolic tangent	$F(S) = \frac{e^{\alpha S} - e^{-\alpha S}}{e^{\alpha S} + e^{-\alpha S}}$	$(-1, 1)$
Exponential	$F(S) = e^{-\alpha S}$	$(0, \infty)$
Sinusoidal	$F(S) = \sin(S)$	$[-1, 1]$
Fractional	$F(S) = \frac{S}{\alpha +  S }$	$[-1, 1]$
Step	$F(S) = \begin{cases} 1, S \geq 0 \\ 0, S < 0 \end{cases}$	$[0, 1]$
Signature	$F(S) = \begin{cases} 1, S \geq 0 \\ -1, S < 0 \end{cases}$	$[-1, 1]$
Binary step	$F(S) = \begin{cases} -1, S \leq -1 \\ S, -1 < S < 1 \\ 1, S \geq 1 \end{cases}$	$[-1, 1]$

Where  $\alpha \in \mathbb{R}$ .

## 2.3 Multi-layer perceptron

That section describes one of the fundamental neural networks' types - **multi-layer perceptron** (or **MLP**, or **back propagation network**). Lots of other neural networks' types - such as *RBF networks* or *probabilistic networks* - are based on that model.

In general, an artificial neural network is a set of artificial neurons. Arrangement and types of those neurons depend on the network type. Multi-layer perceptron contains three types of neurons:

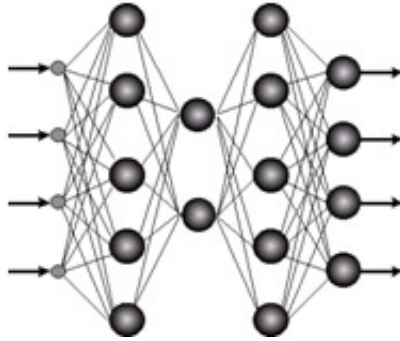


Figure 2.3: **Multi-layer perceptron.**

1. *Input neurons.* Those neurons are taking input vector that encodes some action or information about the external environment. Input neurons don't perform any type of computation, but only pass the input vector to subsequent neurons.
2. *Output neurons* receive signals from the preceding neurons and transform it using formulas 2.1 and 2.2. Those values represent output of the whole neural network.
3. *Hidden neurons* are the basis of the neural network. Those neurons receive the signal from the input neurons or preceding hidden neurons, process it in accordance with formulas 2.1 and 2.2 and then pass result signals to the subsequent (hidden or output) neurons.

In multi-layer perceptron neurons are divided into layers. Input and output neurons form separate layer each - **input layer** and **output layer**. Hidden neurons form one or several **hidden layers**. Every MLP neuron, with the exception of input neurons, is connected via synapses with all neurons of the previous layer. Example of the MLP architecture is shown on Fig. 2.3.

That network receives 4-dimensional input (as it has 4 input neurons). The result is represented by the 4-dimensional output vector. It also contains 3 hidden layers. In publications such network is often referred as *4-5-2-5-4*.

### 2.3.1 Back propagation network training algorithm

Neural networks are the powerful tool for solving different problems - such as classification or prediction. However in order to solve some particular problem, the network should be properly set up and then trained.

The very *first step* is the network architecture selection - e.g., how many input and output neurons should the network contain, how many hidden neurons and layers. Number of input and output units often follows the context. If the network is expected to produce boolean values, like "Yes" or "No", it is clear that it should have only one output. Similar thoughts can be applied to the input neurons' number.

In contrast, selecting the number of hidden units and layers is not simple. In most situations, there is no obvious way of how to determine the best number of hidden neurons without training several networks and estimating the generalization error for each of them<sup>4</sup>. Too few hidden units will produce high training error due to underfitting. Too many hidden units lead to low training error but still have high generalization error due to overfitting. Some discussions about number of hidden units and layers can be found, for example, in [5] or [21].

The *second step* is training itself. Back propagation algorithm belongs to the group called **supervised learning**. It means that to be trained, the network should learn on pairs (*input, expected output*) which all together form the **training set**. In contrast, some classes of networks, like Kohonen's self-organizing maps, belong to the **unsupervised learning** group. It means that the training set does not contain expected outputs for the given inputs. Unsupervised learning is often used for clustering purposes.

At the beginning of the MLP's learning, network's weights are initialized randomly (often by small random values). Then the network

1. processes one by one every item in the training set,
2. compares its output with the desired output,
3. computes error value and

After checking the whole training set, the network modifies its synaptic weights in order to minimize the error.

Most of the MLP training methods are based on the gradient descent approach - iterative change of the network's synaptic weights that gradually decreases the error on the training set. Algorithm that implements that idea is called **error back propagation algorithm** and can be described as follows:

1. **Initialization.** The first step is selecting the network architecture and setting its initial synaptic weights. The most common method of the synaptic weights initialization is choosing small random numbers with the zero mean value and uniform distribution.

---

<sup>4</sup>The **generalization error** is a function that somehow measures how far the network is from the "teacher".



2. **Training pairs presentation.** During that phase, elements of the training set are passed one by one to the network. For every training sample, there are two steps - **forward computation** and **backward computation** - that are taking place.
3. **Forward computation.** Assume that the training set consists of  $N$  of pairs  $(\mathbf{x}_i, \mathbf{d}_i)$ :

$$T = \{(\mathbf{x}_i, \mathbf{d}_i)\}, i = 1, \dots, N, \mathbf{x}_i \in \mathbb{R}^n, \mathbf{d}_i \in \mathbb{R}^m \quad (2.3)$$

where  $\mathbf{x}_i$  is an input vector and  $\mathbf{d}_i$  is the desired output vector.

The network passes the input vector from layer to layer and sequentially computes the outputs for all the layer's neurons. Those outputs form an input vector for the next layer. For every neuron  $j$  in the layer  $l$  (with the exception of input neurons) the output  $y_j^l$  signals are computed from the previous layer output signals  $y_i^{l-1}$  as follows:

$$S_j = \sum_{i=1}^{m_l} w_{ij}^l y_i^{l-1} \quad (2.4)$$

$$y_j^l = F(S_j) \quad (2.5)$$

Here  $m_l$  is the number of synaptic weights for all the neurons of the layer  $l$ . That number is determined by the number of neurons of the previous layer and is the same for every layer's neuron.

At the first step the neuron calculates the linear combination of outputs of the neurons of the previous layer (2.4). The output value is calculated by applying neuron's activation function to the value computed in the first step (2.5).

For input layer - layer 0 - the formula is simpler:

$$y_j^0 = x_{ij} \quad (2.6)$$

where  $x_{ij}$  is the  $j^{th}$  element of the input vector  $\mathbf{x}_i$ .

After the input vector  $\mathbf{x}_i$  passes all the layers, the network produces the output vector  $\mathbf{o}_i = [y_1^{out}, \dots, y_m^{out}]$ , that is used for the **error signal** computation:

$$\mathbf{e}_i = \mathbf{d}_i - \mathbf{o}_i \quad (2.7)$$

4. **Backward computation.** During that step synaptic weights are adjusted in order to produce smaller error value. Those adjustments are derived from the error signal.

For each output unit  $j$  calculate  $\delta_j^{out}$ :

$$\delta_j^{out} = e_{ij} F'(S_j) \quad (2.8)$$

where  $e_{ij}$  is the  $j^{th}$  element of the error vector  $\mathbf{e}_i$  and  $F'$  is the derivative function for the function  $F$ .

For hidden neuron  $j$  of the layer  $l$  calculate  $\delta_j^{hid}$ :

$$\delta_j^{hid} = F'(S_j) \sum_k \delta_k^{l+1} w_{jk} \quad (2.9)$$

where summation is done over all neurons  $k$  of the following layer, and  $w_{jk}$  is the synaptic weight that connects neurons  $j$  and  $k$ .

When all  $\delta$  are calculated, synaptic weights are adjusted using (2.10) and (2.11):

$$w_{jk}^{t+1} = w_{jk}^t + \Delta w_{jk}^t \quad (2.10)$$

$$\Delta w_{jk}^t = \eta \delta_j y_j^{l-1} + \mu (w_{jk}^t - w_{jk}^{t-1}) \quad (2.11)$$

where  $\eta$  is called *learning rate*,  $y_j^{l-1}$  is the output of the neuron  $j$  of the previous layer and  $\mu$  is *momentum*. Learning rate and momentum are numbers between 0 and 1. They will be discussed later.

### 2.3.2 Training algorithm - practical aspects

In the process of back propagation, two parameters are available to the modeler:

1. **Learning rate.** Learning rate is a number between 0 and 1 that determines how fast the neural network adjusts itself to the patterns in the training data. It does not have to be a constant and can also dynamically decrease or increase with time. That parameter must be chosen carefully - too small one will make the learning process slow, and too large one might lead to the divergence. Jacobs in [15] suggest modifying learning rate dynamically and increasing it as long as gradient keeps pointing the same direction, but lowering it when the gradient changes its sign.

2. **Momentum.** Momentum term influences the way of how previous weights affect the current one. It helps the algorithm in preventing being stuck in a local minimum. That value must also be chosen carefully and should be determined experimentally. The use of momentum can be omitted. However it may improve neural network's performance greatly and therefore it is commonly used.

The back propagation process can suffer from several problems:

1. Reaching of the global error function minimum is not guaranteed by the method. As there can be several local minima, the algorithm might possibly remain at one of them.
2. Back propagation method might lead to overfitting: in cases where learning was performed for too long or where the training examples are rare, the network may adjust to very specific random features of the training data, that have no causal relation to the target function. The other possible cause of overfitting is having too many hidden neurons.
3. There is no exact way of how to determine that the algorithm has found the best solution (i.e., global minimum). That problem can be generally approached by restarting the algorithm several times with some changes (e.g., shuffling training set) and therefore increase the probability that the reached minimum is the global one.

Despite those problems, back propagation is a very popular algorithm that seems to be useful in many areas ([3], [1]). Moreover, other training methods are also available - conjugate gradient descent, Quasi-Newton algorithm, Levenberg-Marquardt algorithm. However, those are not widely used.

Another alternative training method is based on the genetic approach. The main principle is the following: weights of the network are not adjusted by the back propagation algorithm, but by the genetic algorithm. At the beginning, the initial population contains randomly generated sets of weights. An evolution algorithm is then applied to the population - it generates set of new weights and keeps only the most appropriate ones. That solution generally prevents the algorithm from being stuck in a local minimum and also shows a good performance. Networks that employ that approach are generally called *Genetic Algorithm Neural Network* or *GANN*. Kim in [17] gives an example and evaluates those networks in financial forecasting.

## 2.4 RBF networks

Radial basis function networks (RBF networks) is the another neural network model that contains several connected layers and shares some other principles with the MLP.

RBF networks are often used for solving approximation and interpolation problems. Commonly, approximation or interpolation solution itself is represented by linear combination of some basis functions (e.g., polynomials) that approximates the objective function with some precision. The principle is the same for RBF networks. The difference is that the RBF network uses so-called *radial basis functions* instead of polynomials.

Formally, interpolation of a set of  $N$  data points requires all the input vectors  $X = \{\mathbf{x}_i | i = 1, \dots, N\}$  to be mapped onto the corresponding target output vectors  $T = \{\mathbf{t}_i | i = 1, \dots, N\}$ . So the goal is to find a function  $f(\mathbf{x})$  so that

$$f(\mathbf{x}_i) = \mathbf{t}_i, \forall i = 1, \dots, N \quad (2.12)$$

The RBF approach employs functions, each of those functions takes the form  $\phi(\|\mathbf{x} - \mathbf{x}_i\|)$  where  $\phi(\cdot)$  is some radial basis function and  $\|\mathbf{x} - \mathbf{x}_i\|$  is some metrics, normally Euclidean distance.

The output of the mapping is then takes the form:

$$f(\mathbf{x}) = \sum_{i=1}^N w_i \phi(\|\mathbf{x} - \mathbf{x}_i\|) \quad (2.13)$$

The last step is finding the weights  $w_i$  such that function goes through all the data points.

Radial basis function networks implement the idea described above. In order to do so, they employ a particular neural network architecture that is described below.

### 2.4.1 RBF network architecture

Radial basis function networks are built using so-called *RBF units*, and each of them implements RBF function. Every RBF unit has  $n$  inputs and one output. It also has two additional parameters that are used for the corresponding RBF function computation - *function center*  $c$  and *function width*  $b$ .

Then the output of the unit for the given input  $\mathbf{x} = (x_1, x_2, \dots, x_n)$  can be expressed as

$$\phi(\xi) = \phi\left(\frac{\|\mathbf{x} - \mathbf{c}\|}{b}\right) \quad (2.14)$$

where  $\phi(\cdot)$  is a radial basis function. The most common radial basis functions are listed below:

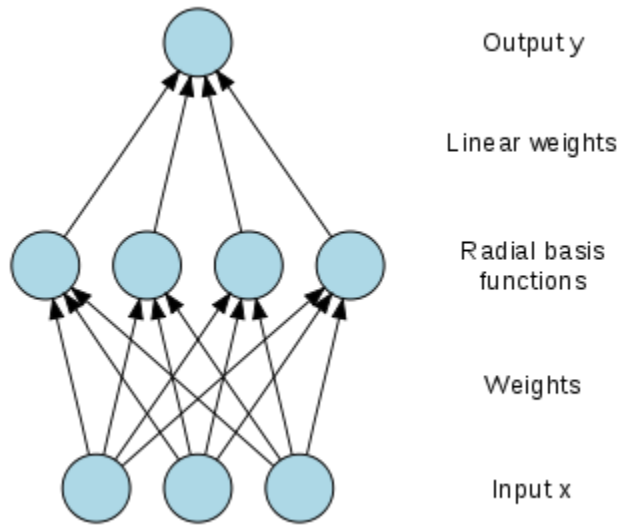


Figure 2.4: **RBF network.** Source: Wikipedia

Function name	Formula
Gaussian	$\phi(\xi) = \exp\frac{-\xi^2}{2\sigma^2}, \sigma > 0$
Multi-quadric	$\phi(\xi) = (\xi^2 + \sigma^2)^{1/2}, \sigma > 0$
Inversed multi-quadric	$\phi(\xi) = (\xi^2 + \sigma^2)^{-1/2}, \sigma > 0$
Generalized multi-quadric	$\phi(\xi) = (\xi^2 + \sigma^2)^\beta, \sigma > 0, \beta \in (0, 1)$
Thin plate spine	$\phi(\xi) = \xi^2 \ln(\xi)$
Cubic	$\phi(\xi) = \xi^3$
Linear	$\phi(\xi) = \xi$

RBF network itself is has three layers - *input layer*, *RBF layer* and *output layer* - and is shown on Figure 2.4. The input layer passes the input vector  $\mathbf{x} = (x_1, x_2, \dots, x_n)$  to the RBF layer. That layer - which is a hidden layer of  $H$  RBF units - using 2.14 transforms the input

into new vector  $\mathbf{y} = (y_1, y_2, \dots, y_h)$  that is subsequently passed to the output layer using the linear transformation. Then the final result  $\mathbf{f} = (f_1, f_2, \dots, f_n)$  can be expressed as

$$f_k(\mathbf{x}) = \sum_{j=1}^h w_{jk} y_j = \sum_{j=1}^h w_{jk} \phi\left(\frac{\|\mathbf{x} - \mathbf{c}_j\|}{b_j}\right) \quad (2.15)$$

where  $f_k$  is the output of the  $k^{th}$  unit of the output layer.

## 2.4.2 Learning of the RBF networks

Similarly to MLP, RBF networks use supervised learning for the training. So the main idea remains the same: based on the training set, at every step the error value is calculated and subsequently used for the network parameters adjustment.

RBF networks employ two different approaches of how to train them:

1. **Gradient training**
2. **Hybrid training**

In that section I will describe the main idea of both of those methods. More information can be found in [20].

### Gradient training

Because RBF networks are very similar to MLP, the same training idea can also be applied to the RBF learning. The only difference is that RBF networks parametrize not only their weights, but also RBF units (by setting custom centers and widths). The main goal of the training process can be stated as setting all required parameters in order to minimize overall training error. Then the common rule for that step can be expressed as:

$$p(n+1) = p(n) - \alpha \frac{\delta E}{\delta p} \quad (2.16)$$

where  $p$  is the value of some parameter during the iteration  $n$ ,  $\alpha$  is the learning rate and  $n$  is an iteration. That approach reminds the MLP gradient descent learning algorithm.

### RBF gradient training outline

1. **Initialization.** Set  $n = 0$ . Set random values to  $w_{ij}(0)$ ,  $c_i(0)$ ,  $b_i(0)$  for all  $i = 1, \dots, h$ ,  $j = 1, \dots, m$ , where  $m$  is the number of outputs.

2. **Next iteration.** Set  $n = n + 1$ .
3. **Error calculation.** Calculate error  $E$ .
4. **Parameters' update.** Update network's parameters:

$$w_{ij}(n + 1) = w_{ij}(n) - \alpha \frac{\delta E}{\delta w_{ij}} \quad (2.17)$$

$$c_i(n + 1) = c_i(n) - \alpha \frac{\delta E}{\delta c_i} \quad (2.18)$$

$$b_i(n + 1) = b_i(n) - \alpha \frac{\delta E}{\delta b_i} \quad (2.19)$$

5. **Testing stop condition.** If stop condition is met (e.g., the overall error is small enough or maximum number of iterations is reached), then stop. Otherwise, continue at step 2.

That algorithm is accumulative, i.e., the error is calculated for the whole training set, not for the individual (*input, output*) pairs.

### Hybrid training outline

Hybrid training is based on the fact, that RBF network parameters can be divided into several groups - parameters that determine location (centers), parameters that determine size (widths) and weights. Hybrid training is using that classification in order to set those parameters separately in three steps.

1. **Step 1.** That step sets RBF units' centers. The idea behind that process is very simple: at some parts of the input space, there is a lot of items and therefore that part should be "covered" by the large number of the RBF units. In contrast, few items may be "covered" by a significantly smaller number of the RBF units.

For finding those confluence regions, the training algorithm employs one of the clustering methods - such as Kohonen's self-organizing maps or K-means. Then it uses found clusters for setting the appropriate centers' values.

2. **Step 2.** The next step is widths  $b_i$  selection. If centers of RBF units define their "location", then the width can be seen as the "size" of the unit. It is important to set those parameters not too small as that poor selection may create areas into the input space that are not covered by RBF units at all. At the same time, widths should not be too large, because in that case RBF units will collide and that is also not desirable.

The one of the possibilities of how to set width is based on a simple idea - the width should be in a direct proportion to the distance between the unit and its closest neighbor. The closer is the closest RBF unit, the smaller width value will suffice.

- Step 3.** While steps 1 and 2 are setting parameters of the hidden layer, the last step sets up weights that are connecting hidden and output layers. That can be done, for example, by solving the system of linear equations. Because for every  $\mathbf{x}_i$  the corresponding output should be:

$$f_k(\mathbf{x}_i) = \sum_{j=1}^h w_{jk} y_j = \mathbf{t}_i^k \quad (2.20)$$

That condition can also be expressed in matrix notation:

$$\begin{bmatrix} y_1(\mathbf{x}_1) & \dots & y_h(\mathbf{x}_1) \\ \vdots & \ddots & \vdots \\ y_1(\mathbf{x}_k) & \dots & y_h(\mathbf{x}_k) \end{bmatrix} \begin{bmatrix} w_{11} & \dots & w_{1m} \\ \vdots & \ddots & \vdots \\ w_{h1} & \dots & w_{hm} \end{bmatrix} = \begin{bmatrix} t_1^1 & \dots & t_1^m \\ \vdots & \ddots & \vdots \\ t_h^1 & \dots & t_h^m \end{bmatrix} \quad (2.21)$$

or

$$\mathbf{Y}\mathbf{W} = \mathbf{T} \quad (2.22)$$

Because the number of the training pairs is normally greater than the number of the RBF units, equations above will have more than one solution. Then the solution with the smallest error should be chosen.

The other approach of how to determine weight values is based on the least square method and is described in details in [20].

## 2.5 Conclusions

In that chapter I have described two models of neural networks: multi-layer perceptrons and RBF networks. Those networks both share a lot of similarities: both of them contain several layers of neurons and they can employ the same principle of the error back propagation learning. They also both play role of the non-linear mappers between multidimensional spaces. In both cases mappings are expressed in terms of composition of functions of one variable.

At the same time, there are substantial differences between those two types:



1. Network architectures may be different: while RBF networks have only one hidden layer, MLP networks may have one or more hidden layers.
2. MLP nodes usually use the same activation function, while RBF nodes use different functions that are parametrized by centers' and widths' values.
3. MLP learning process determines all the parameters at the same time, while RBF learning requires two or more stages of learning to set up the network.

The other substantial difference is that - MLP constructs global approximations, while RBF construct local approximations. That means that for a given input vector, in MLP many units contribute to determine the output value. By contrast, RBF networks form a representation in the space of hidden units which is local. So for a given input vector, only few of the hidden RBF-units will have significant activations.

Choosing between RBF and MLP networks should be motivated by the nature of the input space. For more "clustered" input spaces where some inputs may be considered less relevant than the others, RBF networks may perform better. While in more uniformly distributed spaces, MLP may be the better choice. However, there is no exact rule or algorithm for choosing between those two alternatives and some researchers suggest trying both. The other alternative is the combined RBF-MLP network that tries to take advantage of both MLP and RBF networks ([10]).

For the purpose of my research, I have chosen MLP networks. The first reason for that choice was the greater architecture flexibility provided by the MLP networks. The second and more important reason is the nature of the problem I will try to solve. Using clustering methods, in Chapter 6 I will show that the nature of the input space I will use can not be naturally divided into a set of localized clusters.

# Chapter 3

## Kohonen's self-organizing maps

**Self-organizing maps (SOM)** were invented by *Tuevo Kohonen* in 1984. That neural networks type proved to be a very powerful clusterization and visualization tool and is known as one of the most popular models in the unsupervised learning category.

The main idea of the self-organized neural network was inspired by nature: SOM is based on the same principle of self organization as the human brain. Human visual perception and brain together make a very complex cognition system. More than 90% of the information is entering brain through the visual inputs. Those inputs are then mapped onto corresponding areas of the cerebral cortex, so that neurons dealing with the closely related information are close to each other and can interact via short synaptic connections.

Self-organizing model employs the same principle: it perceives some complex input and maps it onto lower dimensional surface of artificial neurons by activating some of them. Also, same as it happens in brain, similar signals activate nearby neurons. As a result, self-organizing maps can, in some sense, simplify multi-dimensional input space and visualize it in human-friendly two or three dimensions.

It is worth mentioning that Kohonen's model was not the first one inspired by the brain self-organization features. Its predecessor - **Willshaw-von der Malsburg model** - was introduced in 1976 ([30]). It contained two two-dimensional lattices (also called layers) of the exactly same size - one for the input and one for the output (in contrast, the SOM's input is one-dimensional, as we will see later in that chapter). Those lattices were interconnected using synaptic weights and nearby neurons on the input layer activated nearby neurons on the output layer (Fig. 3.1).

Kohonen's model appears to be more general then the Willshaw-von der Malsburg model (that is specialized to mapping inputs and outputs of the same dimensions) [31]. What Kohonen proposed was a much simplified learning mechanism that is capable of performing data compression and don't require the same dimensions of the input and the output. That

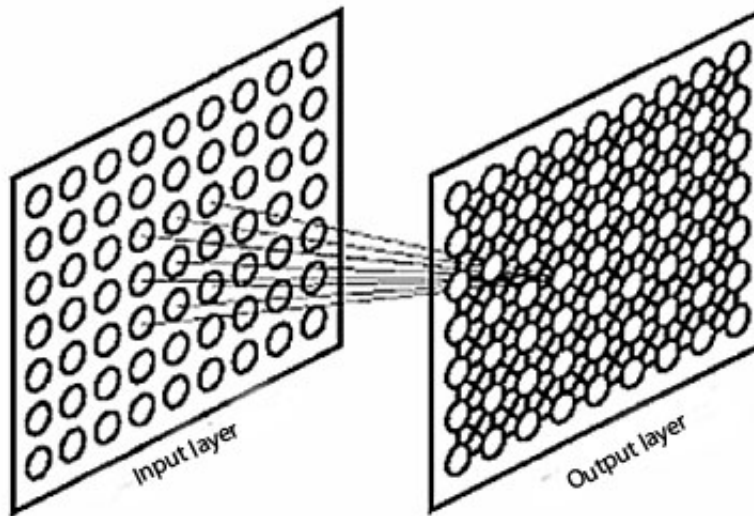


Figure 3.1: **Willshaw-von der Malsburg model**. Input and output layers are connected. There are also interconnections within output sheet [31].

is the reason why that model became much more popular than the previous one.

### 3.1 Self-organizing maps general overview

SOM's architecture differs from the multi-layer perceptron. Same as Willshaw-von der Malsburg model, it contains only two layers. The first layer is the input layer that only passes input vector to the next layer. The second layer is represented by the two-dimensional<sup>1</sup> lattice of interconnected neurons. Every neuron of the input layer is connected to the every neuron of the output layer. Typical SOM architecture is shown on Fig. 3.2.

Output lattice is often two-dimensional and is organized as a rectangular grid. Every neuron of that layer represents some class of possible inputs. That is done by assigning it a numerical  $n$ -dimensional **weight vector** that is also sometimes called **synaptic weight**. Also, every output neuron is connected with its closest neighbors.

Like most artificial networks, SOM operates in two modes: training mode and mapping mode.

1. **Training mode**. At the beginning, newly created SOM does not know anything about the input space it should represent and therefore can not work correctly. So the first step

---

<sup>1</sup>Higher or lower dimensions are also possible, but not common.

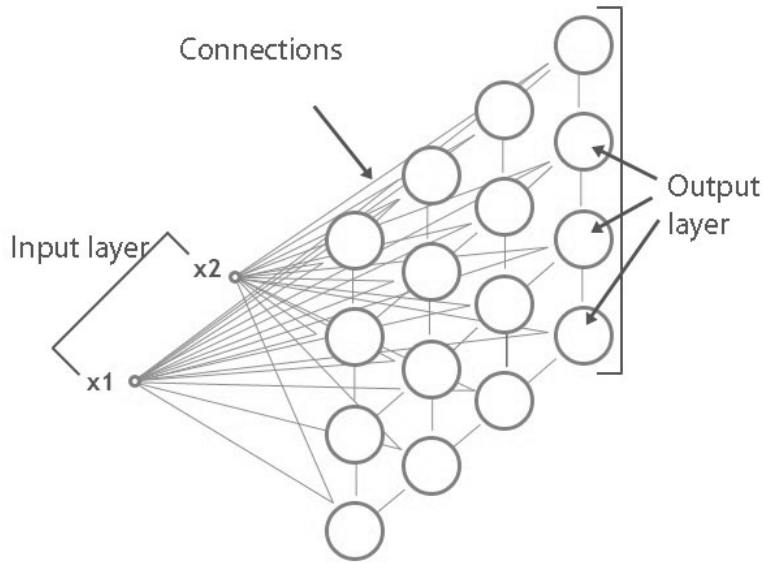


Figure 3.2: **SOM architecture.** Input layer contains two neurons:  $x_1$  and  $x_2$ . Those neurons are connected with all the output layer neurons. Neurons of the output layer are also connected.

is to teach the network of how to classify possible inputs. That process is called *self-organization* or *SOM training* and is very important. Training phase will be described in details later in that chapter.

2. **Mapping mode.** Mapping is performed by the trained network. It automatically classifies a new input vectors.

## 3.2 Self-organizing maps training algorithm

As mentioned before, SOM training is the most important phase because it de facto determines quality of the whole network and its ability to process input data correctly. In that section that process is explained in details.

SOM training algorithm can be divided into four main steps that can be summarized as:

1. **Initialization.** During that phase synaptic weights are being initialized. That can be done by assigning them some small random values. Also, it's important to select the right size of the output layer that will represent the input space properly.

2. **Competition.** For each input pattern from the **training set**, neurons are competing with each other by computing their output (i.e., as scalar product of the input vector and output neuron weight vector) and on the ground of this computation the particular neuron is declared as a winner of the competition.
3. **Cooperation.** During that step topological neighborhood of the winning neuron is being determined. Thereby providing the basis for cooperation.
4. **Synaptic adaptation.** Weight vectors of the winning neuron and neurons from its topological neighborhood are being adjusted under the impact of the input vector.

### 3.2.1 Initialization process

During the initialization process network parameters (e.g., number of output layer neurons and synaptic weights) are set. It's important to select good initial values to make training more effective.

The very first step is the output map size selection. On the one hand it's important to select number of neurons that will represent the input space, on the other hand if the map is too large it may not be efficient. The other important step is to select neurons' initial weights. There are three main possibilities how to do so:

1. **Random initialization.** Vectors are being initialized with the small random values.
2. **Sample initialization.** Initial neurons' weights values are selected among sample values.
3. **Linear initialization.** In that case initial weights are being initialized on the basis of the training set principal eigenvectors (that could be found by using Gram-Schmidt procedure). The main goal of that method is to distribute initial synaptic weights uniformly.

Other important initialization parameters - initial constants, neighborhood and initial radius - will be discussed later.

### 3.2.2 Competition process

Let

$$X = \{\mathbf{x}_i | \mathbf{x}_i = [x_{i1}, x_{i2}, \dots, x_{in}]^T\}, i = 1, 2, \dots, N \quad (3.1)$$

be the training set which contains  $N$  of  $n$ -dimensional vectors.

Then let

$$S = \{j | \mathbf{w}_j = [w_{j1}, w_{j2}, \dots, w_{jn}]^T\}, j = 1, 2, \dots, M \quad (3.2)$$

be the set of the output lattice neurons. There are  $M$  neurons on the output layer (their order is not important at that moment), and every neuron  $j$  is assigned a weight vector  $\mathbf{w}_j$  of the same dimensionality as every of the input vectors.

During the competition process vectors from the training set are being submit, one by one, to the network's input layer that passes it to the output layer. The next step is finding a **winning neuron** (also called a **best-matching neuron** or simply a **winner**) - the neuron of the output layer, that, in some sense, corresponds the given input vector best of all. The most established approach for the winner selection is based on the inner product criterion: neuron  $j$  with the maximum value of inner product  $\mathbf{w}_j^T \mathbf{x}$  is selected as a winning neuron.

As known from the geometry, inner product is equivalent to the distance between two vectors in some Euclidean space, so the inner product condition can be rewritten as:

$$i(\mathbf{x}) = \underset{j}{arg \min} \|\mathbf{x} - \mathbf{w}_j\|, j = 1, 2, \dots, M \quad (3.3)$$

where  $i$  is a winning neuron and  $\|\cdot\|$  stands for Euclidean distance.

### 3.2.3 Cooperation process

One of the basic SOM properties is that in trained network, neurons that are spatially located close to each other also represent similar classes of input space patterns. That property can be achieved during the learning process when the winning neuron is taken as the center of some topological neighborhood where all neurons are cooperating. When the winner is selected and its weight vector changes, all neighborhood neurons' weights are adjusted as well. Those adjustments depend on how far the adjusted neuron is located from the winner.

That section explains the process of cooperation in details and also answers the question: what is the topological neighborhood and how it changes in time.

**Definition.** Let  $i$  and  $j$  be two neurons on the output layer. Then the **lateral distance** between those two neurons  $d_{i,j}$  can be defined as  $d_{i,j}^2 = \|\mathbf{r}_i - \mathbf{r}_j\|^2$  where  $\mathbf{r}_i$  defines the position of the neuron  $i$ , and  $\mathbf{r}_j$  defines the position of the neuron  $j$ . Both are in the discrete  $n$ -dimensional output space.

For example, for one-dimensional space  $d_{i,j}$  is an integer equal to  $|i - j|$  where  $i, j$  are neurons' coordinates on the one-dimensional line. For two-dimensional case,  $d_{i,j}$  turns into Euclidean distance between neurons  $i$  and  $j$ , where neuron's position is defined by its coordinates on the lattice.

**Definition. Neighborhood** (or **topological neighborhood**) of the neuron  $i$  with the **neighborhood radius**  $\sigma$  (sometimes also called **effective width**) is the set of neurons with the lateral distance  $d_{i,j}$  smaller than  $\sigma$ :

$$N(i, r) = \{j | d_{i,j} \leq \sigma\}, j = 1, 2, \dots, M \quad (3.4)$$

For the topological neighborhood we can also define a **neighborhood function**.

**Definition.** Let  $N(i, \sigma)$  is the topological neighborhood, centered in the neuron  $i$ . Then **neighborhood function**  $h_{i,j}$  is a function of the lateral distance  $d_{i,j}$  defined on the  $N(i, \sigma)$ , such that satisfies:

1. The topological neighborhood  $h_{i,j}$  is symmetric to the central neuron for which  $d_{i,j} = 0$  and  $h_{i,j}$  has its maximum at that neuron.
2. Topological function  $h_{i,j}$  decreases monotonically with the lateral distance  $d_{i,j}$  increasing and  $\lim_{d_{i,j} \rightarrow \infty} h_{i,j} = 0$ . That condition is necessary for convergence.

A typical example of the topological function is the Gaussian function:

$$h_{i,j} = e^{-\frac{d_{i,j}^2}{2\sigma^2}} \quad (3.5)$$

One of the features of the SOM training algorithm is that topological neighborhood shrinks with time. It means that parameter  $\sigma$  is a function that is decreasing in time. A popular choice for the  $\sigma(n)$  is the exponential decay:

$$\sigma(n) = \sigma_0 e^{-\frac{n}{\tau_1}} \quad (3.6)$$

where  $\sigma_0$  indicates initial neighborhood radius and  $\tau_1$  is a time parameter.

Therefore, topological neighborhood function can now be written as:

$$h_{i,j}(n) = e^{-\frac{d_{i,j}^2}{2\sigma^2(n)}} \quad (3.7)$$

### 3.2.4 Adaptation process

The last part of the learning process is the **synaptic adaptation**. That means that for every neuron within winner's neighborhood  $N(i, \sigma)$ , its synaptic weights need to be adjusted with a glance of the input vector  $\mathbf{x}$ . That adjustment could be made using generalized Hebb's rule:

$$\Delta \mathbf{w}_j = \eta y_j \mathbf{x} \quad (3.8)$$

where  $\eta$  is the **learning rate parameter** and  $y_j$  is the response of the neuron  $j$  to the input  $\mathbf{x}$ .

However, the Hebb's rule in its initial form is not good for the unsupervised learning and should be modified by including the **forgetting term**  $g(y_j)\mathbf{w}_j$ , where  $\mathbf{w}_j$  is a synaptic weight of the neuron  $j$  and  $g(y_j)$  is some positive scalar function of the response  $y_j$ . The only requirement on the function  $g(y_j)$  is

$$y_j = 0 \Rightarrow g(y_j) = 0 \quad (3.9)$$

Given such function, the synaptic change of the output neuron  $j$  can be expressed as follows:

$$\Delta\mathbf{w}_j = \eta y_j \mathbf{x} - g(y_j)\mathbf{w}_j \quad (3.10)$$

The simplest and the most obvious choice for the forgetting term is the linear function:

$$g(y_j) = \eta y_j \quad (3.11)$$

It's obvious that it satisfies the condition 3.9.

3.10 can be also simplified by setting

$$y_j = h_{i,j} \quad (3.12)$$

where  $i$  is the winning neuron. Then, using 3.11 and 3.12, 3.10 can be rewritten as:

$$\Delta\mathbf{w}_j = \eta h_{i,j}(\mathbf{x} - \mathbf{w}_j) \quad (3.13)$$

And finally, adding discrete time:

$$\mathbf{w}_j(\mathbf{n} + 1) = \mathbf{w}_j(\mathbf{n}) + \eta(n)h_{i,j}(n)(\mathbf{x} - \mathbf{w}_j(\mathbf{n})) \quad (3.14)$$

that is applied to all the neurons that are inside of the topological neighborhood of the winning neuron  $i$ .

In 3.14 the learning-rate parameter  $\eta(n)$  should be time varying. In particular, it should start at initial value  $\eta_0$ , and then decrease gradually with increasing time  $n$ . That requirement can be satisfied by choosing

$$\eta(n) = \eta_0 e^{-\frac{n}{\tau_2}} \quad (3.15)$$

Here  $\tau_2$  is another SOM algorithm constant.

Equation 3.14 visually means moving of the winner's synaptic weights vector  $\mathbf{w}_i$  towards the input vector  $\mathbf{x}$ . Neurons in the winner neighborhood are impacted as well, but in proportion to the neighborhood function value. Repeating presentation of the training data means that synaptic neuron weights tend to follow input vector distribution. The algorithm therefore leads to a SOM layer topological ordering in sense that neurons that are adjacent in the lattice will tend to have similar synaptic weights vectors.



### 3.2.5 Parameters' selection

It is important to select initial training parameters well. In [13] Haykin argues for the following selection:

1. The learning-rate parameter  $\eta(n)$  should begin with value close to 0.1 and then decrease gradually, but remain above 0.01 and not allowed to be decreased to zero. The values of the initial parameters for the formula 3.15 that satisfy that demand may be set as:

$$\eta_0 = 0.1$$

$$\tau_2 = 1000$$

2. The neighborhood function  $h_{i,j}(n)$  should initially include all or almost all the neurons of the lattice, centered in the winning neuron  $i$  and then diminish slowly with time. To satisfy that, in 3.6 we may choose

$$\tau_1 = \frac{1000}{\log \sigma_0}$$

At the end of the training, the neighborhood function should contain only the closest neighbors of the winning neuron.

### 3.2.6 Training algorithm

Now we can describe the SOM algorithm. The first step is initialization of the key parameters that were discussed in the previous section. In short, we need to initialize:

- **Training patterns.** Those vectors should be selected carefully and reflect the input space distribution.
- **Network topology** i.e., number of the output neurons, their locations and initial weights.
- **Time-varying neighborhood function**  $h_{i,j}(n)$  that decreases in time and was described in details above.
- **A learning-rate parameter**  $\eta(n)$  that starts at initial value  $\eta_0$  and then decreases.

The algorithm is then can be summarized as Algorithm 1.

In Algorithm 1 **while** cycle tests stopping condition. It can be, for example, whether maximum number of iterations is reached or there are no significant changes in the map.

---

**Algorithm 1** SOM training algorithm

---

**Require:** Training set  $X$

**Initialization:**

Initialize synaptic weights  $w_{ij}$

Set topological neighborhood parameters

Set learning rate parameters

**Processing:**

**while** stopping condition is *false* **do**

**for all** input vector  $\mathbf{x}$  in  $X$  **do**

**for all** output neuron  $j$  **do**

      compute  $j(\mathbf{x}) \leftarrow \|\mathbf{x} - \mathbf{w}_j\|$

**end for**

    winning neuron  $i(\mathbf{x}) \leftarrow$  neuron with the smallest value of  $j(\mathbf{x})$

**for all** neuron  $j$  within a winner's neighborhood **do**

      update synaptic weights  $w_j$ :

$\mathbf{w}_j(\mathbf{new}) \leftarrow \mathbf{w}_j(\mathbf{old}) + \eta(\mathit{old})h_{i,j}(\mathbf{x} - \mathbf{w}_j(\mathbf{old}))$

**end for**

**end for**

  update learning rate  $\eta(\mathit{old}) \leftarrow \eta(\mathit{new})$

  reduce topological neighborhood radius  $\sigma$

**end while**

---

## 3.3 Other clustering methods

Kohonen's SOM is not the only one clustering method and there are different alternatives to it. Two of them - K-means clustering and Fuzzy c-means clustering (*FCM*) - will be in short described in that section.

### 3.3.1 K-means clustering

K-means clustering is one of the simplest and the oldest clustering methods. It is called **K-means**, because the number of clusters  $K$  should be fixed "a priori".

In general, the K-means clustering idea can be described as following: given a set of  $N$  data points (observations)  $X = \{\mathbf{x}_1, \dots, \mathbf{x}_N\}$ ,  $\mathbf{x}_i \in \mathbb{R}^n$ , the goal is to partition those observations into  $K$  sets  $S = \{S_1, \dots, S_K\}$  so that the value of the function 3.16 is minimal.

$$F(X) = \sum_{i=1}^K \sum_{x_j \in S_i} \|x_j - c_i\|^2 \quad (3.16)$$

Here  $c_i$  is the mean of all points in  $S_i$ .

The algorithm that fulfills that idea is shown below:

---

**Algorithm 2** K-means algorithm

---

1. Select initial  $K$  points from the set  $X$  as representatives for each cluster (centroids)  $C = \{\mathbf{c}_1, \dots, \mathbf{c}_K\}$ . Those points can be selected randomly, or using some heuristics.
2. Assign each point from  $X$  to the cluster, that has the closest centroid.
3. When all points from  $X$  are assigned, recalculate centroids using the following formula:

$$c_i^{new} = \frac{1}{|S_i|} \sum_{x_j \in S_i} x_j \quad (3.17)$$

4. Repeat Steps 2 and 3 until centroids no longer move.
- 

The K-means clustering algorithm is simple and is relatively undemanding on the resources (comparing to Kohonen's SOM, for example). However, it has some weaknesses. Among them:

- The ways of how to initialize centroids is not specified. However, the result may depend on that selection.

- Number of clusters  $K$  should be fixed in advance and the result depends on that value. On the other hand, determining that number for a complex data sets may become a very non-trivial task.
- The result also depends on the metric, that is used to measure  $\|x_j - c_i\|$ .

### 3.3.2 Fuzzy c-means clustering

Fuzzy c-means clustering employs a very similar idea to the K-means algorithm - it divides a set of observations into several clusters. However, in contrast to the previous method it also maintains one additional function - a degree of belonging to clusters. Therefore, each data item may belong to more than one cluster and there is a function  $u_i(\mathbf{x}_j) = u_{ij}$  that measures the degree of being in the cluster  $S_i$  for the item  $\mathbf{x}_j$ . Also, the sum of all the coefficients must be designed to be equal 1:

$$\sum_{i=1}^K u_{ij} = \sum_{i=1}^K u_i(\mathbf{x}_j) = \mathbf{1}, \forall \mathbf{x}_j \in \mathbf{X} \quad (3.18)$$

Then the main goal of the FCM algorithm is to minimize the the following objective function:

$$F(X) = \sum_{i=1}^K \sum_{x_j \in S_i} u_{ij}^m \|x_j - c_i\|^2, 1 \leq m \quad (3.19)$$

Here  $c_i$  is the mean of all points in  $S_i$ .

The algorithm of the fuzzy partitioning is iteratively optimize the objective function 3.19 by updating parameters  $u_{ij}$  and  $c_i$  using the following rules:

$$c_i = \frac{\sum_{j=1}^{|X|} u_{ij}^m x_j}{\sum_{j=1}^{|X|} u_{ij}^m} \quad (3.20)$$

$$u_{ij} = \frac{1}{\sum_{k=1}^{|S|} \left( \frac{\|x_i - c_j\|}{\|x_i - c_k\|} \right)^{\frac{2}{m-1}}} \quad (3.21)$$

Then the FCM algorithm is described below (3).

---

**Algorithm 3** FCM algorithm

---

1. Select number of clusters  $K$  and initialize  $u_{ij}$  randomly or using some heuristics.
  2. Calculate clusters' centers using 3.20
  3. Update  $u_{ij}$  using 3.21
  4. If  $\max_{ij} |u_{ij}^{new} - u_{ij}^{old}| < \epsilon$ , where  $\epsilon$  is termination criterion, then stop. Otherwise return to the step 2.
- 

### 3.4 Conclusions

In that chapter I've described tree methods: Kohonen's self-organizing maps, K-means and Fuzzy c-means clustering. All those algorithms are used for similar tasks - grouping similar input space items together.

K-means and FCM clustering methods are very similar. They both are starting with the predefined number of random clusters and then gradually recalculate those clusters. The only difference between them is one additional parameter - degree of belonging - for FCM. Both those algorithms are rather simple and effective when the number of result clusters is known in advance. Those algorithms also have their drawbacks - for example, clusters depend on the initial state and the order of the input. But the main obstacle for their practical usage is the need to know number of clusters in advance.

In contrast, Kohonen's SOM employs different principle - it maps highly dimensional input space into lower dimensional output space. Basically Kohonen's maps rearrange the data in order to preserve some topological characteristics of the input space. That method does not operate with clusters directly and therefore knowing number of clusters in advance is not necessary.

Because my research is based on highly dimensional and complex data, I have decided to use Kohonen's maps. Mainly because they can operate on data without additional input information (such as number of clusters).

# Chapter 4

## A general introduction to financial markets

In recent years financial markets became very popular. The main reason for that shift is that joining such market is incredibly easy nowadays: there are a lot of banks, individuals and web pages that offer those services. Because of that accessibility, information availability and potential profitability, I have selected financial markets as the domain for my research.

That chapter introduces a general concept of the financial market and explains how it works.

### 4.1 Financial market basics

Financial market is closely related to the **trading** concept. By trading here I mean doing "buy" and "sell" actions by using a special terminal (usually computer). Participants of the market are often called **traders** or **players**.

Financial markets can be divided into several types:

1. **Stock markets.** That kind of market operates with stocks and shares of the large companies like Google or Microsoft.
2. **Foreign exchange markets** or **Forex** facilitate exchange of national currencies. Those markets use currency pairs as a basic tool. The most common example of such pair is a relationship between US dollar and euro (which is denoted as EUR/USD or USD/EUR, depending on the data provider).
3. **Commodity markets** operate with commodities like oil, gold and natural gas.

4. **Futures and derivatives markets.** Those markets provide tools for buying and selling futures and derivatives respectively. The most common object of the transaction is a deferred commercial agreement.
5. Finally, lots of different types of financial markets have appeared in recent years. For example, **index markets** that remind betting agency. Traders are not buying particular goods, but stake on different conditions in the future (e.g., Dow Jones index value next week).

On financial markets all operations are performed "virtually" and do not imply the actual movement of funds and goods. Basically, any trader can perform "buy" or "sell" operation at the current market price anytime (except some days when the market is closed). At any moment, equilibrium market price is calculated as the result of the supply and demand comparison.

Every market type has its own distinctive characteristics. But the main principle remains the same: recent price values form a time series that is called **price curve** or **rate curve**. Traders analyze that curve and make their decisions about the actions that should be taken.

## 4.2 Market analysis techniques

There are two basic approaches that can be used for the market analysis:

1. First method is based on the analysis of external factors that could effect current price and is called **fundamental analysis**. It is generally known, that decisions that the board of directors makes can effect their company's stock prices. Similarly, political instability and high inflation reflect in the country's currency strength. Traders are gathering that type of information from newspapers or annual reports in effort to make better future predictions.
2. In contrast, **technical analysis** is based on mathematical, statistical or numerical analysis of the price curve. That approach is based on the *Dow theory*. It is worth mentioning that other curves also can be used sometimes in addition. For example, US dollar rate and US industrial index are correlated.

### 4.2.1 Fundamental analysis

The main drawback of the *fundamental analysis* approach is its complexity. Every fundamental indicator movement often effects at least several another indicators (e.g., high unemployment rate commonly leads to lower inflation rate). In case when the country has over

50 fundamental indicators with their own cause-effect relations (that can be inconsistent with other indicators), detailed analysis becomes almost impossible. That is the main reason why only 15-25% of traders use fundamental methods, and most of them perform analysis superficially.

The main classes of basic fundamental indicators are:

1. Indicators that describe country's financial market
2. National measures (e.g., gross national product)
3. Inflation rates
4. Monetary and government regulations indicators
5. Index of economical activity and overall economic optimism

More information about fundamental analysis can be found in [19].

## 4.2.2 Technical analysis

*Technical analysis* is the study of how prices behave, which uses only one source of information - price curve. That technique is based on three postulates, that were stated by Dow:

### 1. **Price movements discount all news.**

It means that every external factor that can effect the price (e.g., political, economical, psychological factors) is taken into account as soon as it becomes available. Once news is released, prices will change to reflect this new information immediately. Hence, there is no need in fundamental analysis because all the known information is reflected in the price already.

That statement is a cornerstone of the whole technical analysis. And here is also the main difference from the fundamental analysis. Fundamental techniques state that for the future price prediction, it is necessary to predict all factors that can effect supply and demand of the product. In contrast, technical techniques assume that if the price has changed already - there was "something" (e.g., political or economical news) that effected the price. And that "something" doesn't have to be considered again, because prices have changed already. In financial terms, *market is adoptive to fundamental factors*.



That statement also agrees with the *efficient market hypothesis (EMH)*, that asserts that financial markets are "informational efficient". That means that fundamental analysis doesn't lead an advantage, because other traders have analyzed all known factors and their actions have changed the price already.<sup>1</sup>

## 2. **Market movements are directed.**

**Trend**, or directed price movement, is one of the central ideas of the technical analysis.

Trends can be divided into three categories:

- Bullish Trend - price is going up
- Bearish Trend - price is going down
- Trading Range (sometimes also called Flat or Sideway) - price does not change much

All those types of trends almost doesn't appear per se as a motion along a straight line. Therefore trend can be defined as a prevalent movement.

In terms of trends, goals of technical analysis can be formulated as:

- Recognizing new trends at an early stage
- Using those trends for profit deriving
- Discovering ending of the trend

## 3. **History repeats itself.**

That statement acts on the premise that human psychology doesn't change much. In similar cases the crowd behaves identically. As a result, technical analysis can also be used for patterns detection.

At present, technical analysis is the most popular method for the future rate prediction. Various sources estimate that approximately 60-70% of trades use technical analysis as the main method (other use fundamental analysis or simply intuition).

That popularity of technical methods has its reasons. Among them:

- Price history is often easier to access then fundamental data
- Most of the technical indicators are the part of the trading terminal
- Technical analysis is much simpler then fundamental analysis

---

<sup>1</sup>Despite the fact that EMH is very popular among financial market players, it has never been proved or disproved.

## 4.3 Introduction to trading

In that thesis I am going to analyze artificial neural networks through their application to **Foreign Exchange market** (or simply **Forex**). Main arguments for that decision are:

- Easy access to the historical data and related information
- Large number of published researches and papers
- Variety and functional capabilities of available tools
- Market stability, especially presence of fundamental patterns
- Market liquidity and transactions simplicity
- Simple performance metric: loss or profit amount

### Dealing centers

Dealing center is a company, that works as an intermediary between the financial market and individual trader. It provides on-time data delivery to the client. Dealing center also fulfills orders like "buy X"/"sell Y", which the trader gives through the trading terminal.

Nowadays there are a lot of companies that operate as dealing centers. To enter the financial market, the trader should select some dealing center, register an account, download trading software and make an initial deposit (often up to several hundreds dollars). All those operations can be performed via Internet.

### Positions and orders

**Opened position** is a presence of certain amount of goods that the trader bought for the speculative purposes. The process of buying those goods is called a **position opening**. Selling of available goods is called a **position closing**.

On many markets, including Forex, it is possible to sell certain goods without opening corresponding position before that (i.e., without buying those goods previously). Thereby, we can describe two types of positions: **long position** and **short position**.

- **Long position** is the position that was opened by the buying of a security such as a stock, commodity or currency, with the expectation that it will rise in value. For example, the trader can buy USD dollars, planning to hold it for some time and then sell it with profit for himself. Sometimes that type of position is also called a **bullish position** or **bull transaction**.

- In contrast, **short position** is the sale of a *borrowed* stock, commodity or currency with the expectation that the asset will fall in value. The trader doesn't have to buy that asset before the transaction. In case that trader's prediction was fulfilled, he just takes all the profit of the transaction and "returns" borrowed asset<sup>2</sup>. In opposite case, the trader has to cover the loss. That type of position is also sometimes called a **bearish position** or **bear transaction**.

Commands "buy X" or "sell Y", that the trader gives through the terminal, are called **orders**. There are several types of orders. The simplest and the most common one - **immediate market order** - is executed immediately at the current market price. More complex orders could be postponed until some condition (like a certain price or when maximum allowed loss is reached) is met.

## Prices and ticks

The asset's price at any moment is determined as the result of matching supply and demand. For many markets, that price is set by individual dealing centers separately and therefore may differ from one center to another. However, it is important to state that the price doesn't differ much. In other case traders can simply buy some asset at lower price and immediately sell it to other dealer at higher price. Therefore dealing centers watch over other centers' prices carefully and price differences are insignificant.

One iteration of the price update is called a **market tick**. Market tick is a random event and there is no regular time interval for it. On Forex, number of ticks can reach hundreds per minute during active trading hours, and then fall to just several ticks per minute for night hours, when activity is low.

## Buy and sell price

As mentioned before, the equilibrium market price is the result of matching current supply and demand. Every moment there is a number of subjects willing to buy or sell certain assets and offering the price that they are going to buy or sell for. Then maximum of buying prices is called **BID** price, and minimum of selling prices is called **ASK** price. Those two values form equilibrium price (which is represented as a pair BID/ASK). Generally, BID and ASK prices can be very far one from another. In such case, there is no single price chart, but two charts - BID price chart and ASK price chart. But on active markets, such as Forex, those two prices are always very close.

It is also important to state, that the price can not be effected by individual traders (unless the trader is buying a really huge amount of some goods), but by the bigger players

---

<sup>2</sup>It is done automatically by the terminal.

like banks or event countries. Individual traders usually only buy or sell at the price that is given already.

The difference between the BID and ASK prices is called **spread**. Normally the spread is rather low - for example, on Forex it often differs in the fourth decimal place.

It is also important to state, that dealing centers include their charges into spread and there is no extra fee.

## Price charts

Market price chart is determined by several components:

1. **Time frame**

Time frame specifies a particular period that is used for the price visualization. The most common time frames are: month (MN), day (D1), 4 hours (H4), 1 hour (H1), 30 minutes (M30), 15 minutes (M15), 10 minutes (M10), 5 minutes (M5), 1 minute (M1), one tick. Every chart that is based on a time frame bigger than one tick, contains aggregative information about the whole time frame period.

2. **Opening price**

Price at the beginning of the period.

3. **Closing price**

Price at the end of the period<sup>3</sup>.

4. **High**

The highest price during the whole period.

5. **Low**

The lowest price during the whole period.

6. **Tick volume**

Number of ticks during the period.

Price chart can be displayed in several forms: *tick chart*, *bar chart* and *line chart*.

1. **Tick chart.**

Tick is the smallest scale for the chart. The chart is represented as the sequence of vertical lines (Fig. 4.1). The upper part of each line indicates ASK price, and the lower part - BID price. The tick chart is not used by traders often.

---

<sup>3</sup>Closing price of the one period and opening price of the next period may not be the same.

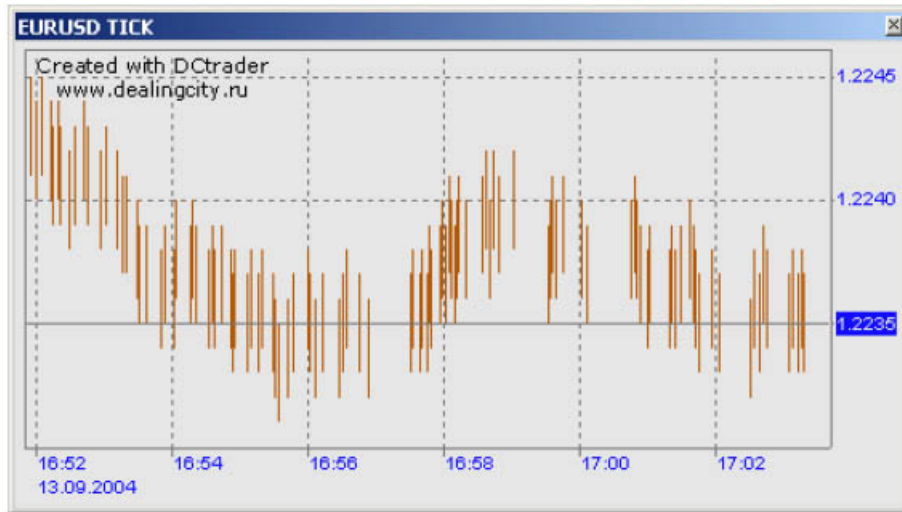


Figure 4.1: Tick chart example.

## 2. Bar chart.

Bar chart is based on a series of vertical bars. Every bar represents a certain period (Fig. 4.2). Lowest point of the bar indicates the lowest price during the period - *Low*. Topmost point is the maximum price value during the whole period - *High*. There are also two small dashes. The left one is the period opening price (*Open*) and the right one is the closing price (*Close*).

## 3. Line chart.

In most cases, line chart is based on the closing price (*Close*). Other options - such as *Open*, *High*, *Low* - are also possible, but rather rare. Sometimes line chart can also use so-called *median price* ( $[High + Low]/2$ ) or *typical price* ( $[High + Low + Close]/3$ ) (Fig. 4.3).

Line chart has its disadvantages. For larger time frames, it may be very difficult to understand what happened inside the period - e.g., whether the price fell and rose, or was stable. On the other hand, line chart is rather "short-spoken" and does not display information that traders could take as redundant.

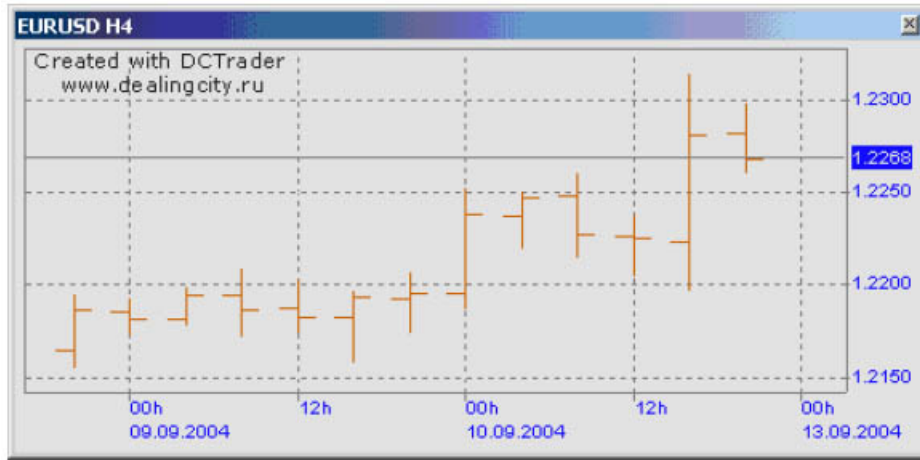


Figure 4.2: Bar chart example.

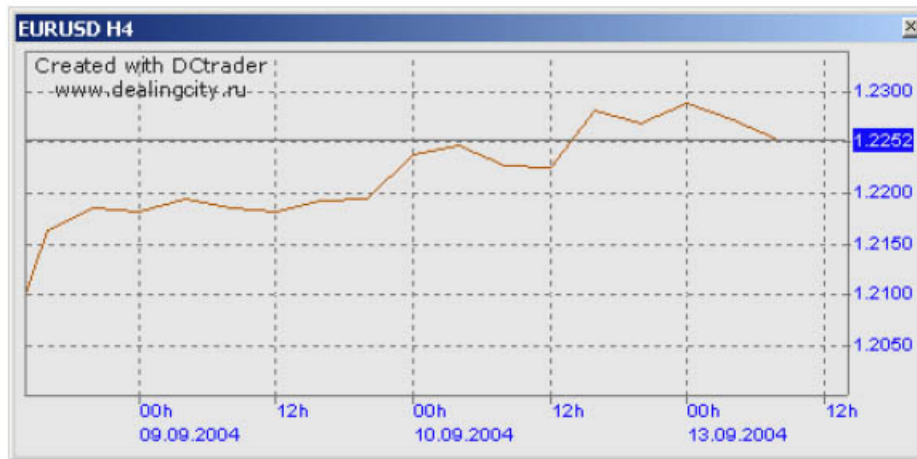


Figure 4.3: Line chart example.

## Trading terminal

As already mentioned, Forex trading is available to everyone. Any person who has a computer, Internet connection, 50\$ for initial deposit<sup>4</sup> and wants to trade, can do that. Then to start trading, the one needs only to choose a dealing center and download a trading terminal, which is basically a program.

**Trading terminal** provides trader with basic tools for working on the financial market. The most basic features of the terminal are:

- Obtaining and displaying information about the price change.
- Orders' execution.

An essential prerequisite for working with the trading terminal is an agreement with some of the dealing centers. That center will provide the data and will execute orders that are sent from the trader's terminal.

Often, trading terminals provide the user with much more advanced functionality than the only basic features - such as data analysis or automated trading tools. Most of terminals also have their own programming language (or support some common language, such as Java), so the trader can not only use built-in functions, but also implement custom behavior.

Trading terminal may be, but doesn't have to be exclusive for the given dealing center. As those centers are only data providers and orders executors, they may use the same type of the terminal.

## Technical indicators and expert advisors

There is a variety of methods that can be used for extracting useful information from the price chart - such as statistics, time series analysis or function approximation. But there is one method that is much more popular among traders than the rest - **technical indexes** (or **technical indicators**).

**Technical index** is a real-time function of the price chart. Value of that function at the given point of time is calculated based on the recent price values. Those indexes can be displayed together with the price chart and when new tick (i.e., new price value) comes, current value of the technical index is updated and immediately displayed along with the price value. Traders use technical indicators to visualize more information, than just raw series data.

There is a big number of different indexes that might be used (and every trader can develop his own indexes). Some of the most common indexes are:

---

<sup>4</sup>Even that is often not important as lots of dealing centers grant initial deposit as a bonus to newcomers.

- **Moving average** or **MA**. That indicator is an average value of the prices of the certain period. That period may vary from several ticks (or bars) to several hundreds of ticks (bars). That index is used for understanding trend.
- **Weighted moving average** or **WMA**. That index is similar to the moving average. The difference is that every price value also has a weight and more recent values have bigger impact on the resulted value.
- **Moving Average Convergence-Divergence** or **MACD**. The Moving Average Convergence Divergence was developed by Gerald Appel in the 1960s. The MACD compares two moving averages, and displays the difference between the moving averages as a single line, with positive and negative values, above and below a zero line. That index is primarily used for indicating trends.

Example of the price chart with two indexes (moving averages with the different period) is shown on Fig. 4.4.

**Expert advisor** is an automated trading system that is based on a certain price chart and contains its own trading logic. The advisor is launched every time when new tick comes. As the result, it generates *buy*, *sell* or *no change* signals, or even trades on its own.

## 4.4 MetaTrader platform

**MetaTrader platform** is one of the most popular trading platforms nowadays. It provides a very rich functionality - from its own programming language to mobile trading - and is used by many dealing centers (detailed list of the most popular of them can be found at [34]). That platform can also be used on different types of financial markets (not only Forex). Its functionality also includes:

- *Dow Jones Forex news feed* - a specialized selection of information in real time, led by professional currency traders
- *Multilingual, intuitive interface*. It supports Russian, English, German, Czech and much more other languages
- *Embedded programming language MetaQuotes Language 4* (or **MQL 4**) that allows the trader to create his own scripts
- *Comprehensive technical analysis*: a large amount of embedded indicators and line tools, the capability to write one's own scripts and indicators, and periodicity support (from minutes up to monthly charts)



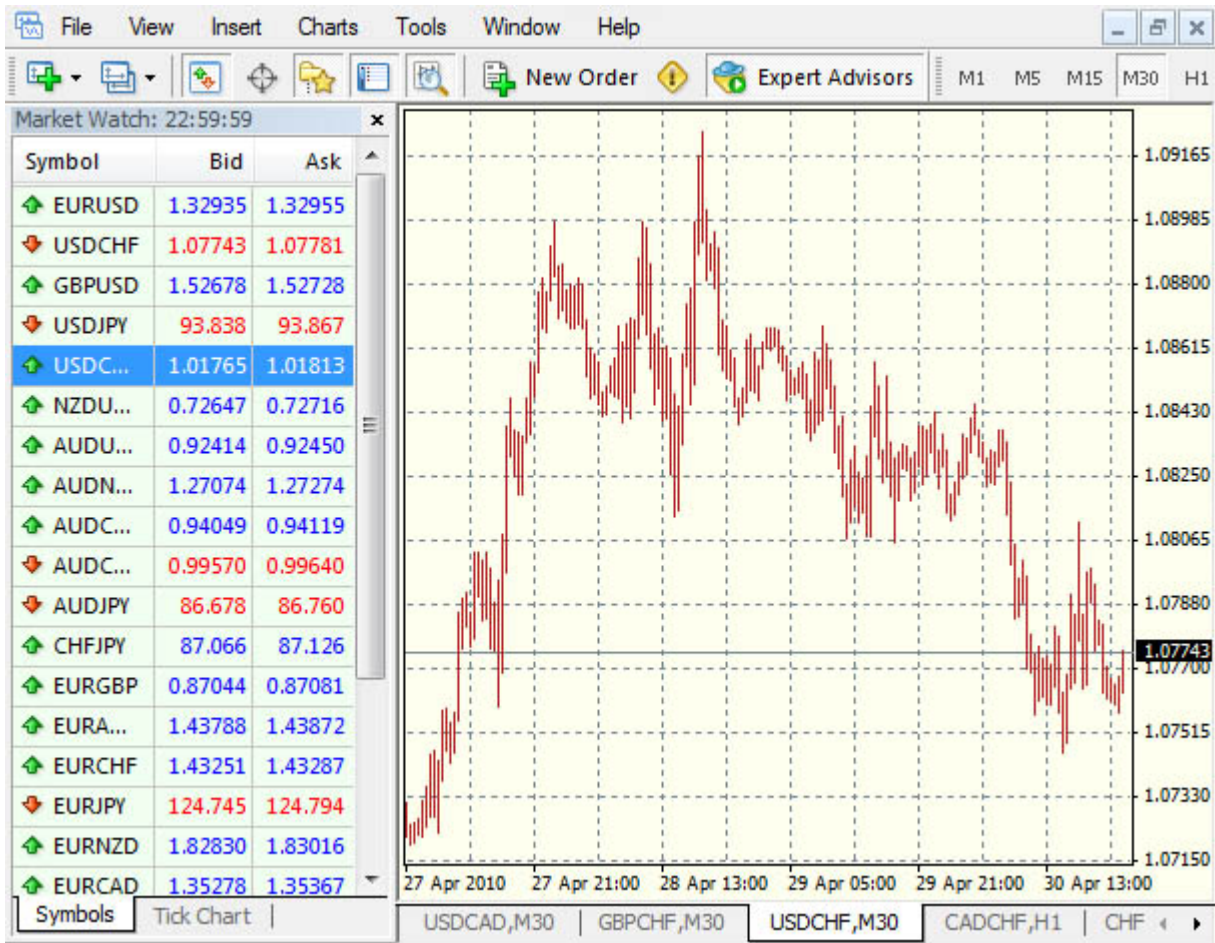


Figure 4.4: MetaTrader 4 terminal.

- *Open MetaTrader Client API* that allows creating custom interfaces and functionality
- *Transaction confidentiality* using 128-bit key data encryption
- *Multiple accounts* support

Current stable version of the platform is MetaTrader 4 (Fig. 4.4).

#### 4.4.1 MQL4 language

MQL4 is a C++-like embedded language for the MetaTrader 4 platform. It allows the trader to create his own indicators and advisors. That language defines number of the trading-

specific functions and work flows that are automatically executed when some standard event (e.g., new tick) appears.

MetaTrader 4 platform also comes with the MetaTrader editor - special environment for the MQL4 programs development. MQL4 language is also well documented - all the information is available for free on its web page.

Although MQL4 is a powerful programming tool that even supports object-oriented programming, it has certain limitations (e.g., execution efficiency) that make implementation of some (such as neural networks) strategies problematic.

# Chapter 5

## Neural networks for Forex trading

Financial series prediction is an essential part of any investment strategy. As most of the Forex transactions are speculative (i.e., made for the profit extraction), all trading participants are interested in the future rate prediction for making better decisions. And despite the efficient market hypothesis, that basically states that stock prices and rates follow a random walk and that all profits are due entirely to chance, most of the market participants believe that there are still regularities that make prediction possible. Numerous researches also share that opinion ([8], [27]).

Neural networks are widely used for the financial time series forecasts ([33], [12], [25]). Despite the efficient market hypothesis, many researches were able to extract rules for predicting future rates using neural networks. In that chapter I will discuss different stages and approaches to time series forecasting, with the main focus on neural networks' application.

### 5.1 Selection of input and output variables

Good selection of the input and output variables is fundamental for the accurate forecast. Often it requires a deep understanding of the economical background.

There are several approaches that are commonly used:

1. **Sliding window.** Financial time series can be represented as a series of vectors:

$$\mathbf{X}_t^d = [x_{t-1}, \dots, x_{t-d}] \quad (5.1)$$

where  $\mathbf{X}_t^d$  contains  $d$  values that preceded  $t^{th}$  value of the series. Then the forecasting method can be formally stated as finding a function  $f : \mathbb{R}^d \rightarrow \mathbb{R}$  such as next value  $x_{t+1}$  is determined by the previous  $d$  values of  $\mathbf{X}_t^d$ :

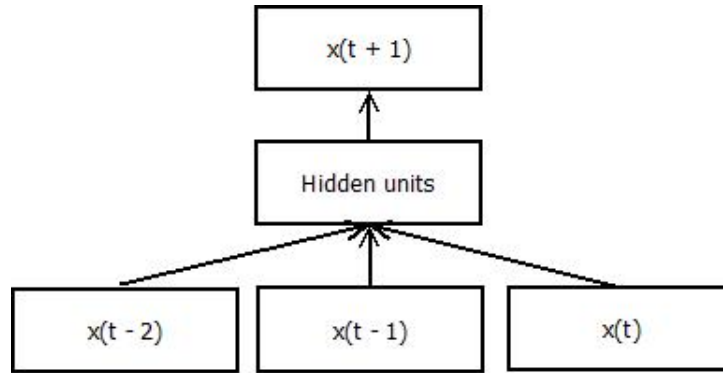


Figure 5.1: Scheme of the sliding window technique.

$$x_{t+1} = f(\mathbf{X}_t^d) \quad (5.2)$$

Neural networks perform that task by using feedforward network architecture, such as *MLP* or *RBF*. The input value is a  $d$ -dimensional vector and output is the single number that represents next value (Fig. 5.1). Multiple next values are also possible, but rare. That method is also called *sliding window* as vector  $\mathbf{X}_t^d$  "slides" over the full training set.

That method is also closely related to the Takens' theorem ([22]), which has one useful implication: if the market movements are not completely random (e.g., efficient market hypothesis is not true), then exists such  $d \in \mathbb{N}$ , that provides unambiguous prediction of the next series value.

It is necessary to take into account that even if exists, value of  $d$  can be rather big and therefore neural networks could not be able to process such input vectors in a reasonable time. Huang and others in [14] suggest method of how to overcome that by compressing the input and reducing the dimensionality. Other well-known dimensionality reduction methods (e.g., *Principal component analysis*) can be found in [3].

2. **Sliding window for trend.** For making profitable trades, it is not necessary to know exact values of the the future rate. So the previous method can be adapted for predicting the trend movement and its strength instead of the exact future rate value. Then the output can be represented by the binary signal, that encodes rate's increasing or decreasing. Such an output is rather useless, though. It should also carry some additional information such as expected length of increasing or decreasing signal. Networks can also be trained to produce signals for buying, selling or doing nothing, instead of doing the prediction.

3. **Prediction based on extracted data.** Analyzing raw series data using neural networks can be rather expensive. That is why it makes sense to analyze data that already have economical context and contain extracted information from the raw data - such as technical indicators. Selecting right set of indicators can be a challenging task and often requires a deep understanding the trading economical background. But that method requires less input data and can significantly improve computational performance.

## 5.2 Data preprocessing

Before the data are analyzed by the neural network, they have to be preprocessed in order to increase the accuracy of the output results and to facilitate the learning process. That stage is crucial and the way the data are represented influences network's behavior directly.

Sometimes it may be necessary to fill up empty or missing data ([23]). As Forex trading stops on weekends, there might be a gap between Friday and Monday data (e.g., closing price on Friday is almost always not the same as opening price on Monday). Those data may be interpolated or rebuilt by separate neural network.

Another important preprocessing operation is data normalization. First and foremost, it should be noted that before starting the ANN training, the training set must be properly prepared. Prices and quotations should not be chosen as inputs and outputs themselves. ([28]). The most common preprocessing methods are:

1. **Scaling.** Here we are assuming that every input variable belongs to some interval  $x \in [Min, Max]$ . Then the most simple normalization is:

$$x_{new} = \frac{x - Min}{Max - Min} \quad (5.3)$$

After that transformation, every input value is over the range 0 to 1 (Fig. 5.2).

Linear scaling is a good method when input values are distributed uniformly within [Min, Max] interval. If not, values can be normalized using statistical indicators:

$$x_{new} = \frac{x - \bar{x}}{\sigma} \quad (5.4)$$

where  $\bar{x}$  is an average and  $\sigma$  is a variance of the initial data.

2. **Increments.** Let  $C_t$  be the rate value at the time  $t$ . Then the next value  $C_{t+1}$  can be expressed as:

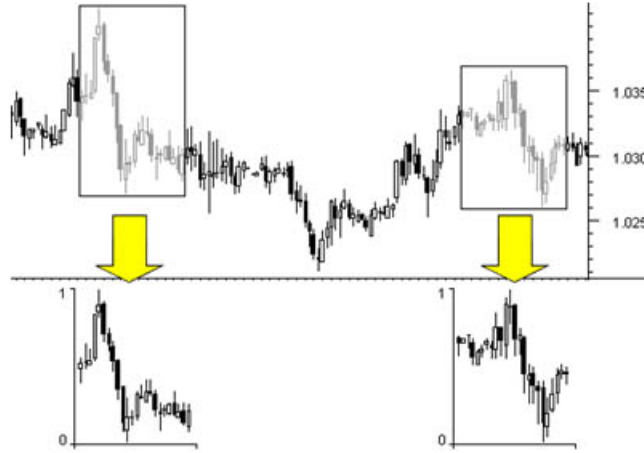


Figure 5.2: **Data transformation to the unit interval.**

$$C_{t+1} = C_t + \Delta C_t \quad (5.5)$$

where  $\Delta C_t$  is called *rate increment*.

As those incrementations are much smaller than rates values, there is a strong correlation between the next and previous rate values. The most probable value of the next exchange rate is the same as the previous one:  $C_{t+1} = C_t$ .

To avoid that correlation, input series can be represented not by rates, but by their increments  $\Delta C_t$  (Fig. 5.3) or relative logarithm increments  $\log(C_t/C_{t-1}) \approx \Delta C_t/C_{t-1}$ .

3. **Decomposition.** Sometimes time series are also built from separate components such as trend, seasonality, cycles and noise ([23]). In that case, series data may be preprocessed using those components - input data are divided into those components and prediction is taking place within the scope of each component separately. If the series contain a lot of regularities, it is much easier to predict it. However, Forex (and other financial markets) data do not contain such obvious patterns and also do not seem to be autocorrelated.

### 5.3 Model construction

There are three types of neural networks that are commonly used by economists and researchers in market prediction:

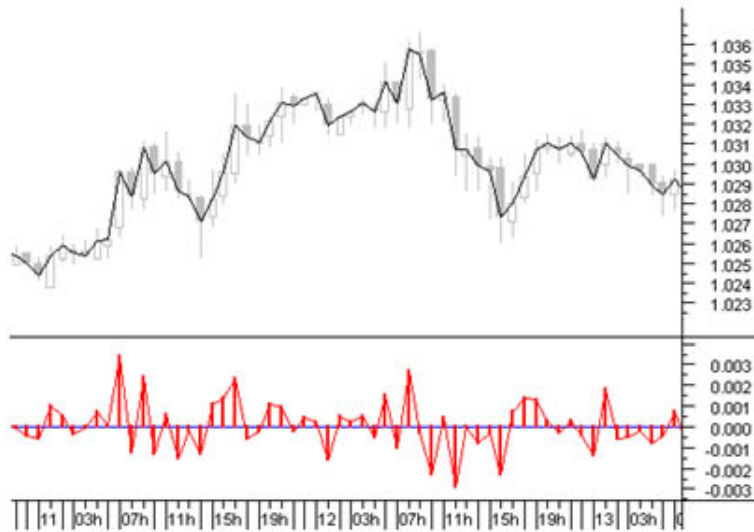


Figure 5.3: Quote increments as input variables.

1. **Multi-layer perceptron.** That layered feed-forward network was explained in details in previous chapter.
2. **Radial basis function network (RBF).** RBF is a hybrid multi layered network that contains a single layer of processing units. Those units use some RBF function (most commonly Gaussian transfer function) rather than standard sigmoid function. That type of network is explained in details in [3] and [13].
3. **Generalized feed-forward network.** That network is a MLP generalization that allows jumping over one or more layers.

The right type of neural network is often selected by developing and comparing numerous possibilities and by keeping only the best ones. However, some researchers approached the problem of the right model selection and extracted some advices which model can be more appropriate for which task ([28], [12]). It is necessary to take into account that those advices do not guarantee better results in any case.

After the model is selected, it should be instanced using particular parameters, e.g., number of inputs and outputs and number of hidden layers.

1. **Number of input neurons.** That number is determined easily when the previously defined steps have been done. As every input variable is treated by a single input neuron, input neurons' count is determined by the size of the input vector (which, as we seen

before, can be found using Takens theorem with the following input compression, or selecting a set of input indicators).

2. **Number of hidden layers.** Number of hidden layers can not be clearly defined because the optimal number strongly depends on the data and may differ from case to case. However, some researches approached that problem and developed common advices ([21], [5]). In most cases, one or two hidden layers proved to be enough, and increasing the number of those layers also increases the danger of overfitting ([4]).

Due to the nature of the quote or stock movements, number of hidden layers can't be predicted easily and training of networks with the different number of hidden layers for the following comparison may be required.

3. **Number of hidden neurons.** There are no clear rules of how to select the right number of hidden neurons. The most common technique is to determine appropriate number during the experimentation phase. But it also necessary to keep in mind that too big number is demanding on resources, and too small number may not reflect all the variety of the input data. Kaastra and Boyd in [16] suggest always prefer the network that performs well on the testing set and has the least number of hidden neurons.
4. **Number of output neurons.** Most neural networks use only one output neuron for the next value forecasting. However, as predicting exact price may not be the best approach for the Forex trading, it may be useful to consider several output neurons.
5. **Number of neural networks.** Different networks have different strengths and weaknesses and individual network, even when it has good results on the testing data, may not perform well in all real situations. Impact of that factor may be decreased by using a set of  $L$  different networks (also called a *committee of neural networks*).

It is easy to see that the average error of the whole committee is smaller then the average expert from the set. Let  $\epsilon_i(\mathbf{x})$  be the error for the  $i^{th}$  network for the input  $\mathbf{x}$ . Then using *Cauchy-Bunyakovsky-Schwarz inequality*, it is easy to prove that:

$$\left(\frac{1}{L} \sum_{i=0}^L \epsilon_i\right)^2 \leq \frac{1}{L} \sum_{i=0}^L \epsilon_i^2 \tag{5.6}$$

Error decreasing can be rather significant. If errors of individual networks do not correlate, the committee error can be up to  $\sqrt{L}$  times less than the average error of the average network error. That is why predictions can be improved significantly by using the average predicted value of the whole committee.





Figure 5.4: MACD advisor.

Another approach that uses several neural networks can be based on the following idea: every network from the committee receives the same amount of resources at the beginning. During the trading process, resources are being redistributed - profitable networks receive more resources that are taken from unprofitable networks from the same set.

## 5.4 Other forecasting techniques

There is a huge number of different forecasting approaches - some are based on statistics (e.g., ARIMA model), some are based on heuristics (e.g., Elliot waves) and others are based on neural networks. Among the most commonly used techniques (besides neural networks) are<sup>1</sup>:

1. **Moving average.** That method is based on two moving averages - one is for short period (usually 9-12 bars), and other is for a long period (usually 15-25 bars). Buy and sell signals are determined by the lines intersection: buy signal is generated when the short period moving average is going above the long period moving average and vice versa (Fig. 5.5).

---

<sup>1</sup>I will compare suggested neural networks' approaches to those methods later



Figure 5.5: Moving average advisor.

2. **Moving Average Convergence/Divergence or MACD.** That advisor is based on MACD indicator which is calculated as the difference between the fast (i.e., with the smaller period) and slow moving averages values (MACD indicator is explained in Chapter 4). The other part of the MACD advisor is so-called *signal line* which is also a moving average line with the period close to 10.

MACD advisor works with the difference between MACD indicator and the signal line. When the difference is 0, it generates buy or sell signal. Buy signal is generated when the MACD indicator is increasing, and sell signal is generated when the indicator is decreasing (Fig. 5.4).

# Chapter 6

## Self-organizing maps and Forex

That section shows by example how self-organizing maps can be used for the useful data extraction and trading strategy implementation. Here I will present two different applications - traders' profiling and SOM-based automated advisor for Forex trading.

There is a number of books and articles that advice traders to be more or less active, open more long or short positions, be risky or use more conservative strategies. Therefore the first part's goal is to explore the set of different strategies and try to answer the question: "Are there some behavior patterns of the successful strategies?". If such patterns are found, that could provide the trader with a very useful information.

Second part of the section proposes different approach to the ANN-based Forex trading then those described in Chapter 5. I will show how to create advisor that is based on self-organizing maps. That advisor does not try to predict the future, but recognize situations that are appropriate for trading and then using them.

### 6.1 Example: Advisors' profiling

*Automated trading championship* is a yearly competition, organized and supported by companies related to Forex - dealing centers, banks and financial software providers. To compete, every trader has to develop an automated advisor using MQL4 language. That advisor then trades on its own on real-time data for the period of approximately three month. Once submitted, the advisors can not be changed. One trader can only submit one advisor.

At the beginning, every advisor receives 10 000 fictitious US dollars (for the simulation purposes only). Absolute profit is always the winning criterion.

After the championship, the organizer publishes participants' statistics on the championship web page. For the analysis, I have extracted information about 186 participants of the last competition - Automated trading championship 2008. Final profits of those participants

vary from approximately 159 500\$ for the winner to -2 926\$.

### 6.1.1 Data for the profiling

The profiling is based on the following parameters:

1. **Profit.** The first and the most important parameter is a total profit amount. Advisor's efficiency evaluation is based on that parameter.
2. **Total trades.** That parameter indicates how active was the advisor. Some strategies made only tens of trades, while others did hundreds.
3. **Expected payoff.** Expected payoff is the average profit per trade in a deposit currency (USD).
4. **AvgWin/AvgLoss factor.** Ratio of an average winning trade to an average losing trade.
5. **Relative wins.** That parameter indicates the percent of the winning trades.
6. **Short positions or shorts.** Percentage of the short positions among all trades.

Some of those variables were preprocessed (e.g., total profit was normalized to the interval  $[0, 1]$ ) in order to give more accurate results. For the clustering I have used squared network  $7 * 7$  with 49 neurons.

### 6.1.2 Results

The result of the self-organizing algorithm application to the data is shown of Fig. 6.1 and Fig. 6.2.

Algorithm divided trading strategies into 4 clusters:

- **Cluster 1 and Cluster 2.** Those clusters contain the most of all advisors - unprofitable ones.
- **Cluster 3 and Cluster 4.** In contrast to previous two clusters, strategies from Cluster 3 and 4 are profitable. Cluster 4 contains the most prosperous advisors and Cluster 3 contains not the most successful, but still profitable advisors.

Based on the results of the Kohonen's clustering, I have made the following conclusions:

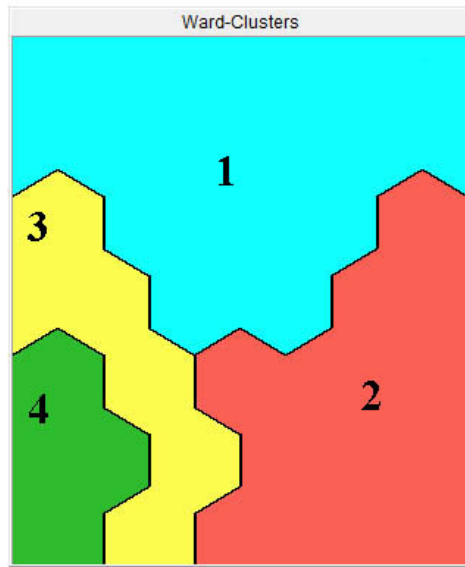
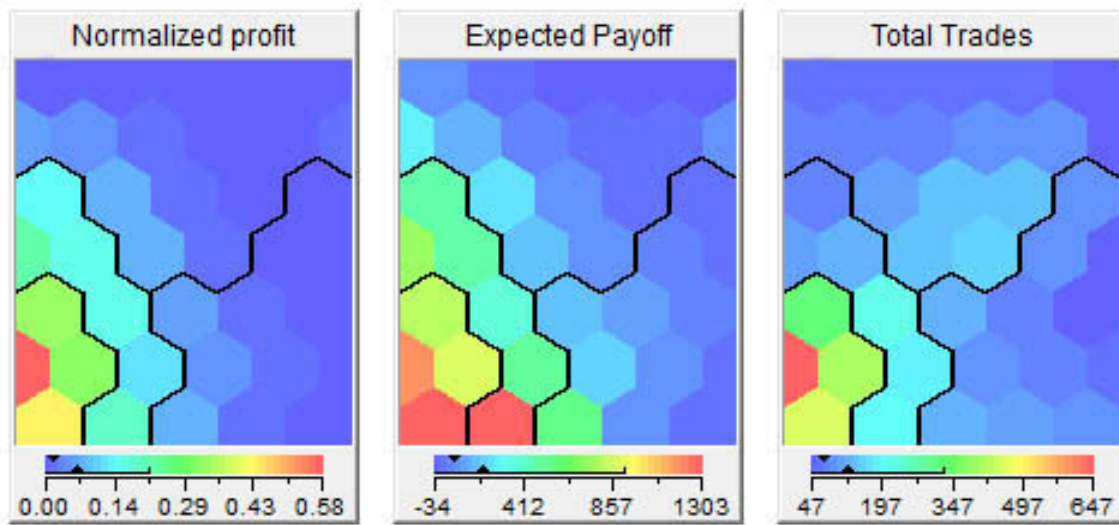


Figure 6.1: Advisors' clustering.



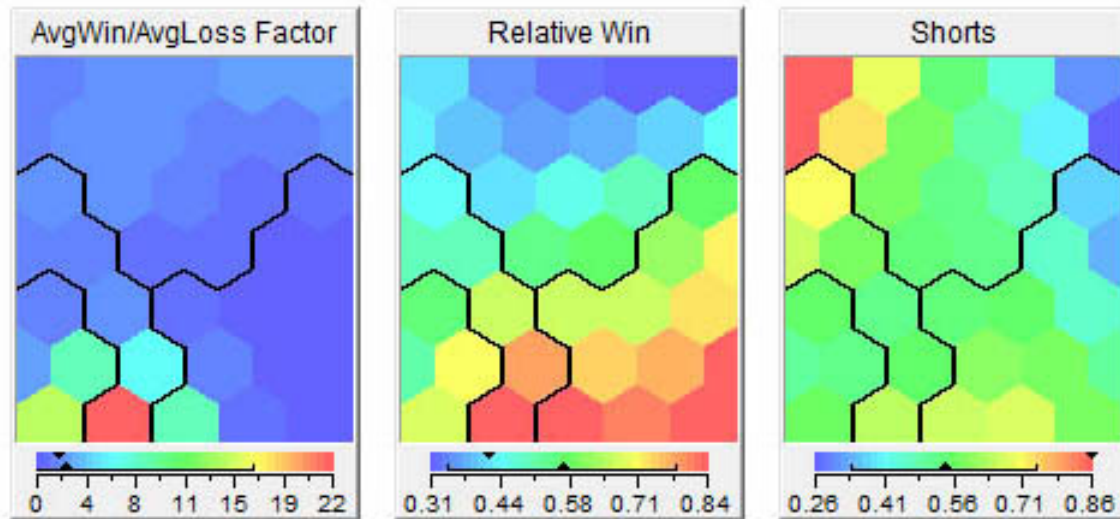


Figure 6.2: Clustering of the individual parameters.

1. The most successful strategies were also the most active ones - they did more trades than strategies from other clusters.
2. Successful strategies also had the highest expected payoff rate. However, that result is not unexpected - the correlation between total profit and average profit per trade is very high.
3. The most successful advisors do not have the highest - but slightly above average - winning rate. It means that those traders had to make several very profitable trades to compensate relatively low percent of the winning trades.
4. Best strategies did not prefer short or long positions and opened both of them equally.
5. Expected payoff values are rather close for both "profitable clusters". On the other hand, strategies from the Cluster 4 earned approximately five times more money than those from the Cluster 3. That can be explained by the fact, that advisors from the winning cluster were much more active.
6. Strategies from the Cluster 2 had the highest win rate. It means that those strategies were probably optimized for making winning trades. However, it did not grant them profit.

7. Profitable strategies also had large AvgWin/AvgLoss factor. Surprisingly, automated advisors with the largest values of the factor do not belong to the winning cluster.

### 6.1.3 Conclusions

Championship table indicates that only 122 of more than 700 traders ended with some profit. That can be explained by several factors:

First of all, participants had to prepare automated advisors for the rather long period of time - three month. And it could be difficult, to create an adaptable strategy that will adjust itself to all nonstandard changes that will happen.

Secondly, clustering indicated that superior profits of the winning strategies could be due to several large winning trades. And those could be just matter of luck. As participating automated advisors are not shared, that conclusion can not be proved or disproved.

And finally, having a good strategy that will have a big percentage of winning trades still does not grant profit.

## 6.2 Example: SOM-based Forex advisor

Almost every neural network's application to Forex prediction is based on attempt to predict the future of the market. The most common predictions are exact price values or trend direction and its strength. As an alternative, prediction can be also based on the market situation classification: what if there are situations that appear to be favorable for opening long or short positions? If so, those situations could be found using a classification tool - such as self-organizing maps.

### 6.2.1 Data for classification

For prediction, I have chosen pair *USDCHF* - US dollar and Swiss franc. For the network training I've used *M30* quotes data for years *2007* and *2008* - 24 587 bars in total. Year 2009 was used for evaluation. The rates data are obtained from the Alpari dealing center through the MetaTrader4 terminal.

The most simple and natural way of how to describe current market situation is to use of different indicators. So I've also extracted satellite information - some indicators.

The extracted data contain following information:

1. *Bar information*. Standard information about every bar - date, time, high value, low value and close value (those parameters are explained in first chapter).

2. *Information about quote behavior in the future.* There are two values - for convenience called *BUY* and *SELL* - that indicate position of the current price in comparison with maximum and minimum prices within the scope of some *sliding window*. In that example the window has size of 20 next bars.

The idea of those values is based on following empirical observation: if the price has grown since the first observation, then buying (i.e., opening long position) would be profitable, especially in the situation when the growth was significant. And vice-versa for short positions - big price drops indicate sell signal. So high BUY values indicate the situation when long position should be opened. And low SELL values state for short position.

Those signals will be used as basis for self-organizing maps analysis.

3. *Indicators values for every bar.* For classification, I've used following indicators (all those indicators are explained in detail in Appendix A):
  - (a) Relative strength index - *iRSI*
  - (b) Momentum index - *iMomentum*
  - (c) Bill Williams' Accelerator/Decelerator oscillator - *iAC*
  - (d) Movement directional index - *iADX*
  - (e) Indicator of the average true range - *iATR*
  - (f) Bill Williams' Awesome oscillator - *iAO*

## 6.2.2 Clusters

SOM algorithm used the map with 1000 nodes. No input data transformation was performed. Training data contained information about BUY and SELL signals and all the indicators listed above. Result map indicates that there are two areas - one is favorable for opening long positions, and another is favorable for opening short positions (Fig. 6.3).

Selected areas indicate situations when the price change was more then 100 points. Application of those areas to indexes' maps shows that none of them indicates BUY or SELL situation explicitly (Fig. 6.4).

Further analysis shows that BUY and SELL areas mark out same intervals for almost all of the indicators. That fact suggests the following idea: BUY and SELL indicators' intervals, when put together, can recognize the situation when some action (BUY or SELL) should take place. As type of the position that should be opened - long or short - is unknown at that moment, the advisor should open both long and short positions. Every of those two positions should have set two values:



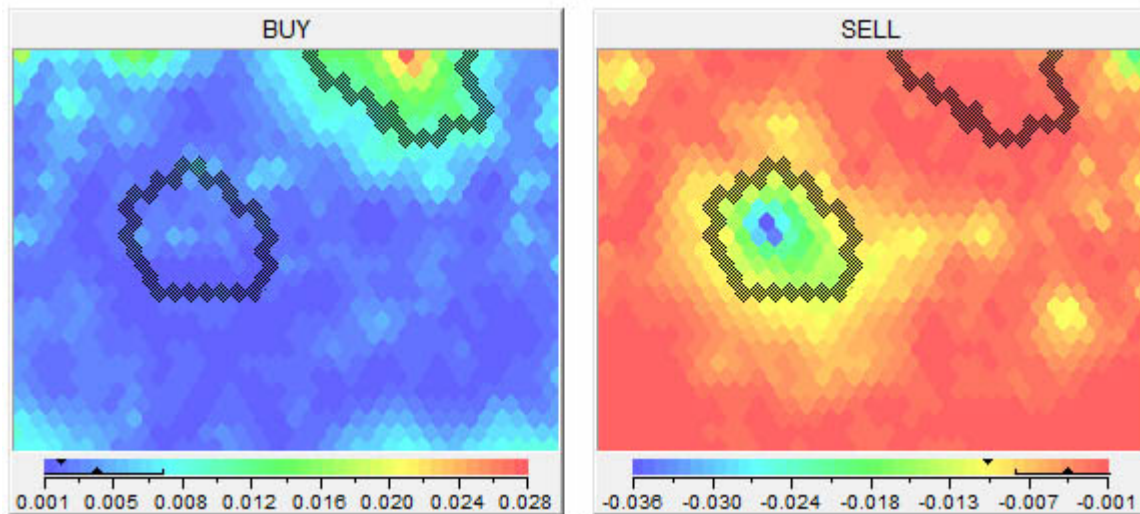


Figure 6.3: BUY and SELL signals.

1. *Stop loss level.* That parameter limits losses if the market does not move in the preferred direction. When the stop loss value is reached (that could be certain amount of money or quote value), the position is closed automatically at the current market value in order no to lose any more resources.
2. *Take profit level.* Similar to the previous one, a take profit order is an order that closes the position once it reaches a certain level of profit.

As both orders start simultaneously, stop loss level should be much less than take profit level. Thereby, unprofitable order will be closed rather quickly while profitable one will be opened until desired profit level is reached.

At that point it is important to understand that proposed method does not guarantee that the order that is closed by the stop loss signal will not become profitable in near future. As well as order that remains opened could finally appear unprofitable and take profit level will never be reached.

Based on the BUY and SELL clusters, following conditions were extracted: Fig. 6.5

Stop loss parameter was set to 15 points. Here one point equals to 0.0001. It means that the position is closed when rate changes on 0.0015 in unfavorable direction. For example:

- For *long position* that was opened at price 1.15856, the position is closed as soon as the price drops to 1.15706.

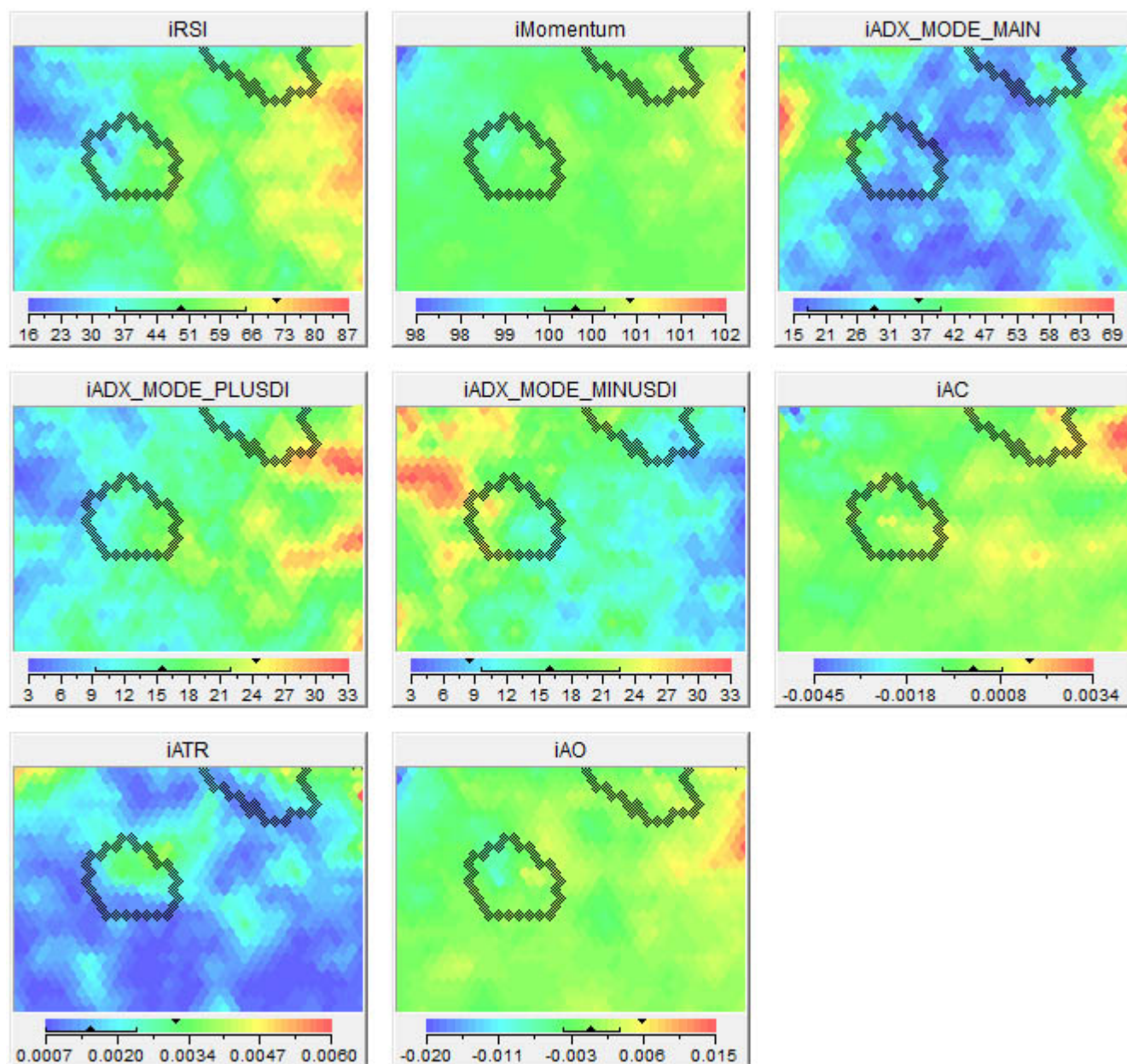


Figure 6.4: BUY and SELL signals with other indicators.

```

if((indRSI > 35 && indRSI < 68)
&& (indAC > -0.0010 && indAC < 0.0016)
&& (indAC < 0.006)
&& (indMomentum > 99.2 && indMomentum < 100.3)
&& (indADX_MAIN < 40)
&& (indADX_MINUSDI > 10 && indADX_MINUSDI < 26)
&& (indADX_PLUSDI < 27 && indADX_PLUSDI > 8)
&& (indATR < 0.001))

```

Figure 6.5: SOM-based advisor: conditions for opening new order.

- For *short position* not decreasing, but increasing is unfavorable. So for the same opening price at 1.15856, the position is closed when the price increases to 1.16006.

Similarly, take profit value was set to 46 points

### 6.2.3 Results

As already mentioned, the training set was represented by M30 bars from years 2007 and 2008. Then the advisor was tested on USDCHF M30 bars for the first 6 months of the year 2009 and then for the whole year 2009. Two other advisors - MACD and Moving average (both explained in Chapter 4) - were also tested on the same data for comparison purposes. The initial deposit was set to 10 000 US dollars.

#### Results for 6 months

Those results were received based on M30 USDCHF testing on real data. The testing period was 1.1.2009 - 30.6.2009.

	SOM advisor	MACD advisor	Moving average advisor
Total net profit	5 428.34\$	-1 907.45\$	-1 384.19\$
Total trades	100	329	285
Expected payoff	54.28\$	-5.80\$	-4.86\$
Profit trades (% of total)	31%	93.62%	24.91%
Long positions (% of total)	51%	49.5%	46.3%

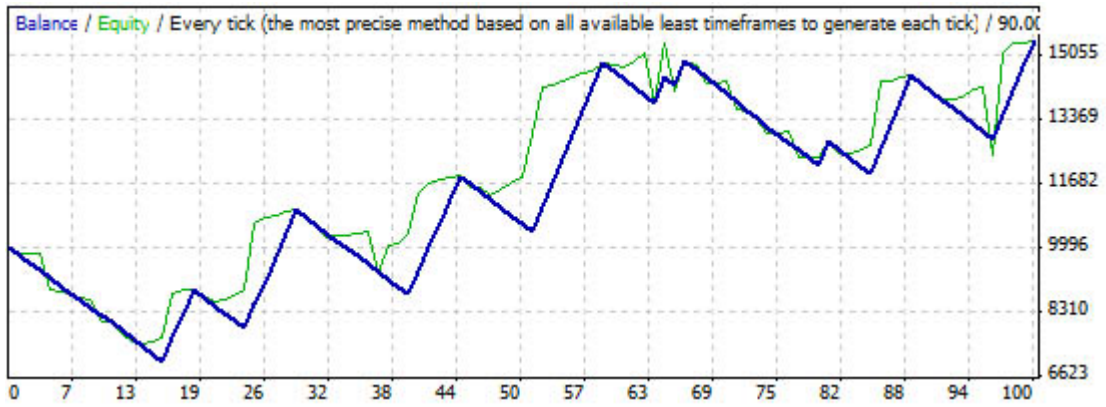


Figure 6.6: SOM-based advisor: 6 months of trading

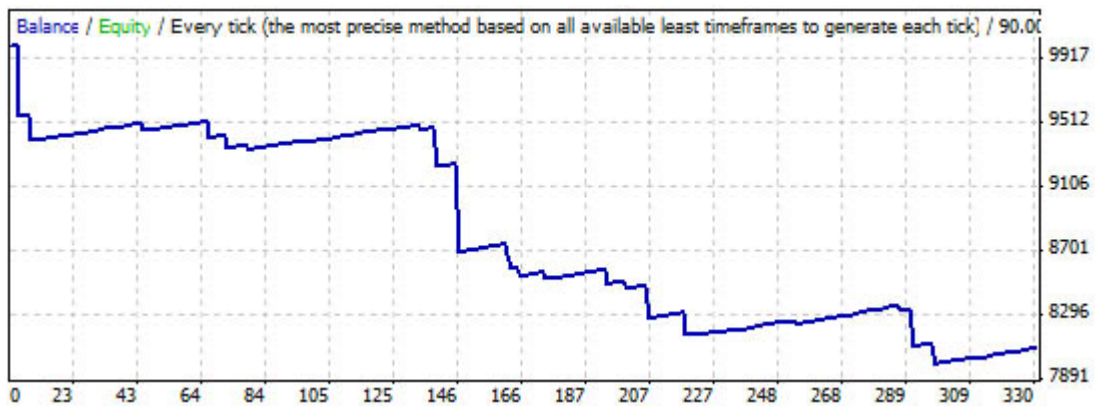


Figure 6.7: MACD advisor: 6 months of trading



Figure 6.8: Moving average advisor: 6 months of trading

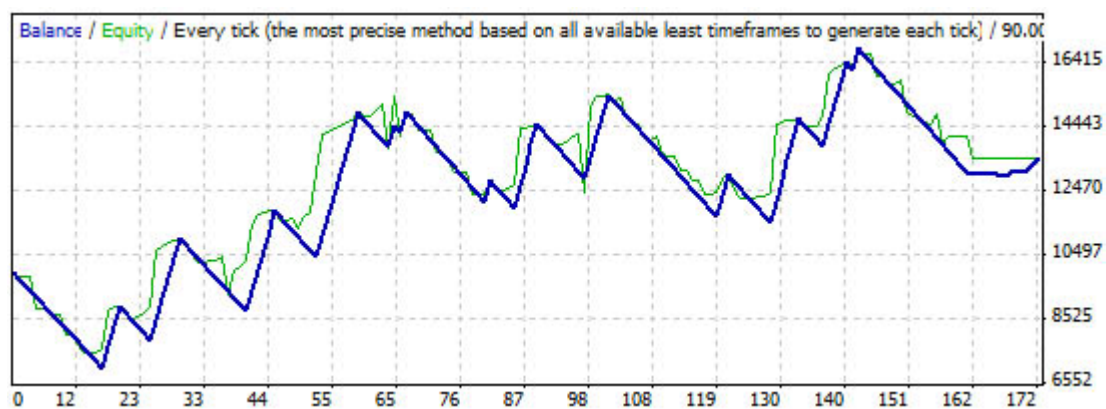


Figure 6.9: SOM-based advisor: 1 year of trading

### Results for 1 year

Those results were received based on M30 USDCHF testing on real data. The testing period was 1.1.2009 - 31.12.2009.

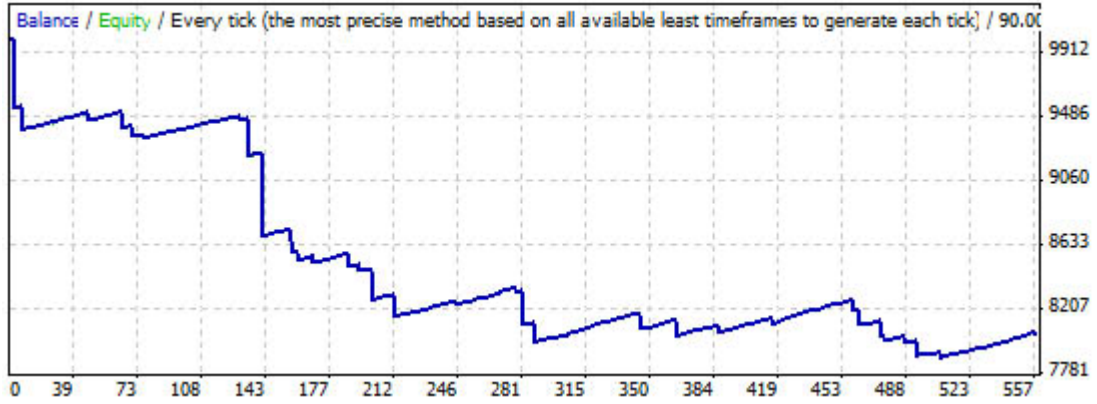


Figure 6.10: MACD advisor: 1 year of trading



Figure 6.11: Moving average advisor: 1 year of trading

	<b>SOM advisor</b>	<b>MACD advisor</b>	<b>Moving average advisor</b>
Total net profit	3 497.70\$	-1 962.69\$	-2 983.64\$
Total trades	172	556	610
Expected payoff	20.34\$	-3.53\$	-4.89\$
Profit trades (% of total)	29.65%	93.88%	21.97%
Long positions (% of total)	50.6%	51.61%	78.03%

### **Results' interpretation**

- As seen from the results' table, SOM-based advisor significantly exceed both MACD and Moving average advisors. Comparing to those two strategies, SOM-based strategy is not very active - it makes approximately 3 times less trades than other strategies.
- Results indicate that 6 months return is higher than 1 year return. There are two possible reasons for that:
  1. Indicators' intervals are dynamic and should be recalculated on less than 1-year basis. Simply put, values that worked out at the beginning of the year do not have to remain the same for the whole year. If so, SOM-based advisor's conditions should be updated in 6-months period or so.
  2. Because of economic crisis in years 2009-2010, all the financial markets (including Forex) became more dynamic and unpredictable. Those circumstances did not effect Forex at the beginning of the crisis (which was exactly first part of the year 2009) as much as during the second half of 2009. That is why extracted intervals worked in the 6 months experiment much better.

Those problems could be generally solved by updating advisor's conditions more often. But there is a possible problem there - as during the unstable period Forex is much more changeable, there might be not enough data for applying Kohonen's method. It would be reasonable to assume that our method will give much better results during the economic stability. In general, this assumption is applicable to all methods. But as self-organizing maps require much more data for training than MLP networks, they may become impracticable is the insufficient data situation.

- SOM-advisor has a rather low profit trades rate at about 30%. That fact is easily explained by the advisor's behavior: it opens a lot of unprofitable trades that are closed

quickly by the stop loss signal.

That observation could be also applicable to the advisors' profiling. Recall that the most profitable advisors also had low profit trades rate. One possible reason for that is luck. The other and more likely one is that those strategies open short and long position at the same time and then quickly close unprofitable one.

- SOM-advisor opens short and long positions equally. That result is expected as it opens two different positions at the same time.
- Self-organizing maps can be used as an evaluation tool for new indicators. If clustering results for some indicator agree with BUY or SELL clusters, that indicator can be used for recognizing BUY or SELL situation.

It is important to understand that indicators do not create new information, but rather extract and consolidate information from the existing series data. And self-organizing maps, that could hardly work with the raw series data, still prove to be an effective tool for Forex trading strategies.

#### **6.2.4 Conclusions and future work**

Self-organizing maps appear to be a very useful tool for creating a Forex advisor. Clustering of BUY and SELL signals helped in indication of conditions that are favorable for taking action. Also, SOM can be employed as evaluation tool for Forex indicators.

But there are still many directions for future research. Among them:

- That method was tested on a limited number of indicators. More indicators can be examined.
- That method can be examined on different currency pairs. USDCHF pair is partially stable: US dollar is a rather dynamic currency, while swiss franc is more stable. Testing that method with two stable or unstable currencies may give different results.
- Also, SOM could be used for finding some effective set of indicators that is still efficient in recognizing the situation when market is about to change significantly. I have used 8 indicators, but some of them could be redundant or correlated. And some important indicators could be missed.



# Chapter 7

## FXANN - A Forex trading model

In that chapter I will propose an ANN model for the Forex technical analysis and improvement of the learning method for that enhances the prediction capability of the network. I have called that model **FXANN** (*Forex + Artificial Neural Networks*). Because working with technical indicators seems to be much more efficient than using exact price rates or closely related data like rate increments, FXANN is applied to the set of extracted Forex indexes.

### 7.1 FXANN model

The overview of the proposed model for Forex trading is shown of Fig. 7.1. FXANN classifies the input pattern that consists of several technical indicators, and generates buying, selling or idle (no-change) signal.

As seen on the figure, the whole system consists of a multi-layer neural network, preprocessing unit and postprocessing unit. The preprocessing unit normalizes every input signal to the unit interval. After that the network generates output and the postprocessing unit converts result into a corresponding signal.

There are four possible alternatives of how the rate is changing over time:

- The rate grew in the recent past, but will drop in the near future. In case that those rate changes are not really small, that situation is favorable for selling (i.e., opening a short position).
- The rate movement in future is the same as it was in the past - the price will continue growing or falling.
- The rate felt in recent past, but will grow in near future. That situation is favorable for buying (i.e., opening a long position).

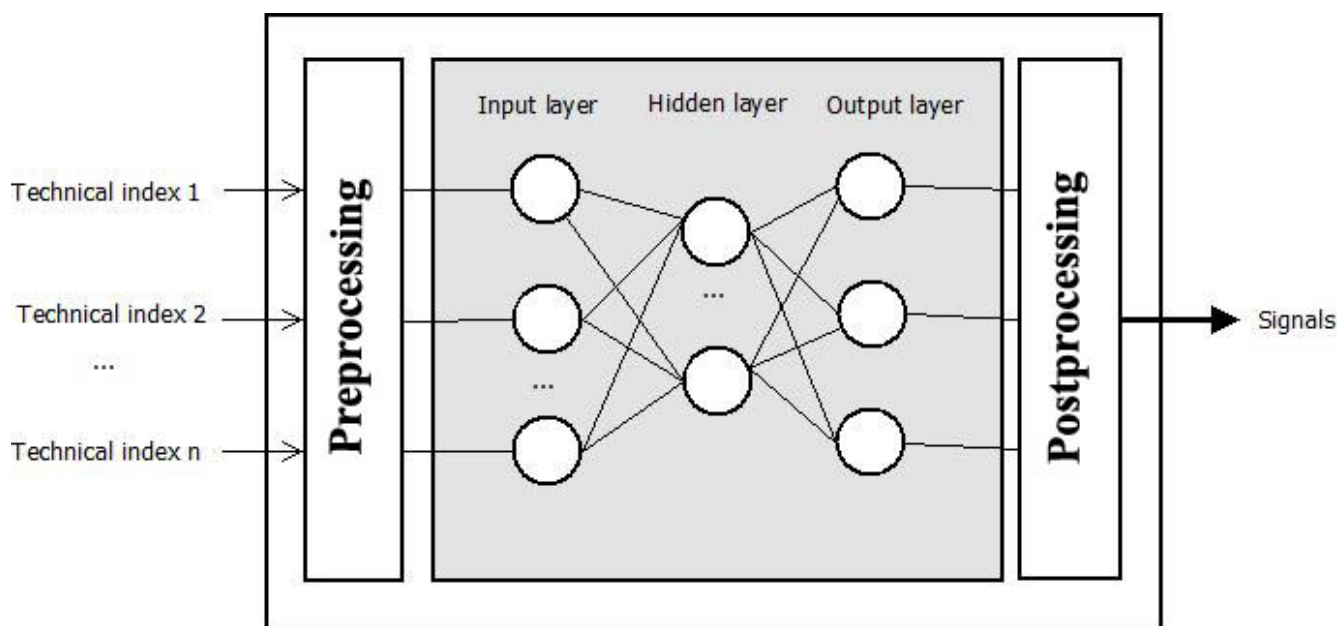


Figure 7.1: **FXANN model overview.**

For the FXANN model, I've divided rate alterations into three groups: selling, buying and no-change signals. Those groups and their possible encoding is shown on Fig. 7.2. To be considered as buying or selling signal, every rate change should be greater than some threshold value (usually threshold is set at about 0.005). If the threshold is too small, final profit may not even cover dealing center's charges.

### 7.1.1 Input data

FXANN does not work with the rates values, but with the different indicators. The main reason for that choice is similar as that was used for the SOM prediction: prediction of the exact rate is difficult and needless. While approach when the network recognizes the market situations and makes its own decisions - *buy, sell, no change* - is much closer to the trading nature itself. That is why the network should be optimized not for the next number estimation, but for making the right decision at the right moment.

As isolated rates are non-informative, the best way of how to describe current market situation is using of technical indicators. List of indicators that may be used within FXANN









Price development			Output pattern
(past)	(present)	(future)	
			(1, 0, 0) : selling signal
			(0, 1, 0) : no-change signal
			
			(0, 0, 1) : buying signal

Figure 7.2: FXANN model overview.

is shown on Fig. 7.3<sup>1</sup>.

All those indicators are relative - i.e., they measure rate's behavior, but not rate itself (example of non-relative indicator is *Moving average*).

### 7.1.2 Output data

FXANN network has three output neurons and is expected to generate *buy*, *sell* or *no-change* signals (Fig. 7.2). Because those output patterns contain only analog values - 0 and 1 - actual output pattern may not match with any of those three signals.

In this case, the postprocessing unit converts actual output to analog value by using two thresholds. In the experimental simulation (that is described further), I have used the thresholds values 0.4 and 0.6. When the output is beneath 0.4, 0 is selected. Similarly, for the output greater than 0.6 postprocessed value equals to 1.

In some situations - e.g. when some of the outputs is between 0.4 and 0.6 or final signal does not match any of three categories - the network's output can not be classified. In that case FXANN produces failure signal *NA*.

<sup>1</sup>All those indicators are explained in Appendix A

Indicators

<input type="checkbox"/> StdDev 9	<input type="checkbox"/> ADX 9	<input type="checkbox"/> AC
<input type="checkbox"/> StdDev 14	<input type="checkbox"/> ADX 14	<input checked="" type="checkbox"/> AD
<input checked="" type="checkbox"/> StdDev 25	<input checked="" type="checkbox"/> ADX 25	<input type="checkbox"/> AO
<input type="checkbox"/> StdDev 50	<input type="checkbox"/> ADX 50	<input type="checkbox"/> MACD
<input type="checkbox"/> StdDev 75	<input type="checkbox"/> ADX 75	
<input checked="" type="checkbox"/> StdDev 100	<input checked="" type="checkbox"/> ADX 100	
<input type="checkbox"/> StdDev 200	<input type="checkbox"/> ADX 200	
<input type="checkbox"/> RSI 9	<input type="checkbox"/> CCI 9	
<input type="checkbox"/> RSI 14	<input type="checkbox"/> CCI 14	
<input checked="" type="checkbox"/> RSI 25	<input checked="" type="checkbox"/> CCI 25	
<input type="checkbox"/> RSI 50	<input type="checkbox"/> CCI 50	
<input type="checkbox"/> RSI 75	<input type="checkbox"/> CCI 75	
<input checked="" type="checkbox"/> RSI 100	<input checked="" type="checkbox"/> CCI 100	
<input type="checkbox"/> RSI 200	<input type="checkbox"/> CCI 200	

Figure 7.3: Possible FXANN input indicators.

### 7.1.3 Training and testing sets

FXANN can work with any trading pair and any time frame. Extracted data should contain all necessary information about every bar: all indicators, current price, past price, future price.

In addition to the current rate value, every bar item contains information about the set of the past and future rates within different time frames (called *windows*). FXANN operates with 8 possible window values: *9, 14, 20, 34, 50, 75, 100 and 200* bars. Every bar item also contains information about values of different indicators for the current bar.

Based on the training data and the selected window size, the FXANN preprocessing unit creates correct output signals. For that purpose it compares previous and current rates with the current value. For example, for the 20 bars' window three values are compared - 20 bars ago value, current value and 20th bar in the future value. Those three values fit one of four possible output signals (Fig. 7.2). In addition, to be considered as buy or sell signal, all differences between prices should be greater than some threshold value. In FXANN, the threshold value is set a 0.01.

- For example, let some bar's rates are *1.0710* (past), *1.0671* (current) and *1.0696* (future). Despite the fact that those rates formally fit buying signal, that triplet is evaluated as no-change signal because rates' differences are smaller than 0.01.

### 7.1.4 Adjusted learning

As mentioned earlier, the FXANN's network tries to generate one of three output patterns for classification. Learning of the network is done by using of the MLP back propagation algorithm. As described in Chapter 2, the strategy of the algorithm is to modify the weights of the connections between neurons towards decreasing the total sum of prediction error. However, in case that the deviation of the numbers of learning samples is large among output patterns categories, the following problem arises: the neural network will have the tendency to improve prediction accuracy of the dominant pattern only (which is no-change signal). As the result, the FXANN generates less buying and selling signals than expected.

To overcome that problem, the FXANN employs so called *Adjusted learning*. The main idea is to duplicate buying and selling samples in order to achieve more uniform distribution among input patterns. So the learning process may be expressed in the following steps:

1. **Multiplication factors calculation.** All training samples are divided into three groups in accordance to Fig. 7.2. After that, according to the groups' sizes, we should define two numbers - multiplication factors (i.e., how many times to duplicate) for buying and selling groups.

- For defining the FXANN learning multiplication factors, I've used equations:

$$sellfactor = ROUND\left(\frac{|NO - CHANGE|}{2 * |SELL|}\right) \quad (7.1)$$

$$buyfactor = ROUND\left(\frac{|NO - CHANGE|}{2 * |BUY|}\right) \quad (7.2)$$

where  $|NO - CHANGE|$ ,  $|SELL|$  and  $|BUY|$  is the number of no-change, buy and sell bars correspondingly.

2. **Sample duplication.** During that step, buy and sell sets are duplicated in accordance to its multiplication factors.
3. **Shuffling training set.** After the duplication, (duplicated) buy, (duplicated) sell and no-change sets are merged into one training set that is then shuffled.
  - For example, for USDCHF M30 training for 2008 and 20 bars window, there are 308 samples that generate buying signal, 11574 samples with no-change signal and 459 samples generating selling signal. Then multiplication sell factor is 13 and multiplication buy factor is 18. After sampling adjusting, the result training set contains 22825 items.

#### 4. Back propagation training.

## 7.2 FXANN tool

FXANN tool is a program that implements FXANN model (Fig. 7.4). It allows configuring the model by setting different parameters - like window size and number of learning iterations. The FXANN tool is developed for testing different FXANN instances in order to select most effective one.

As the first step, FXANN preprocessing unit loads testing data, calculates correct output patterns and display data for every bar - past price ( $P1$ ), current price ( $P2$ ), future price ( $P3$ ) and the correct output ( $Pattern$ ) - to the user in the *Data* list. Corresponding technical indicators to the testing set are loaded as well, but only internally and are not shown to the user.

Second step is creating FXANN neural network and training it. Both network and training parameters are set buy the user.

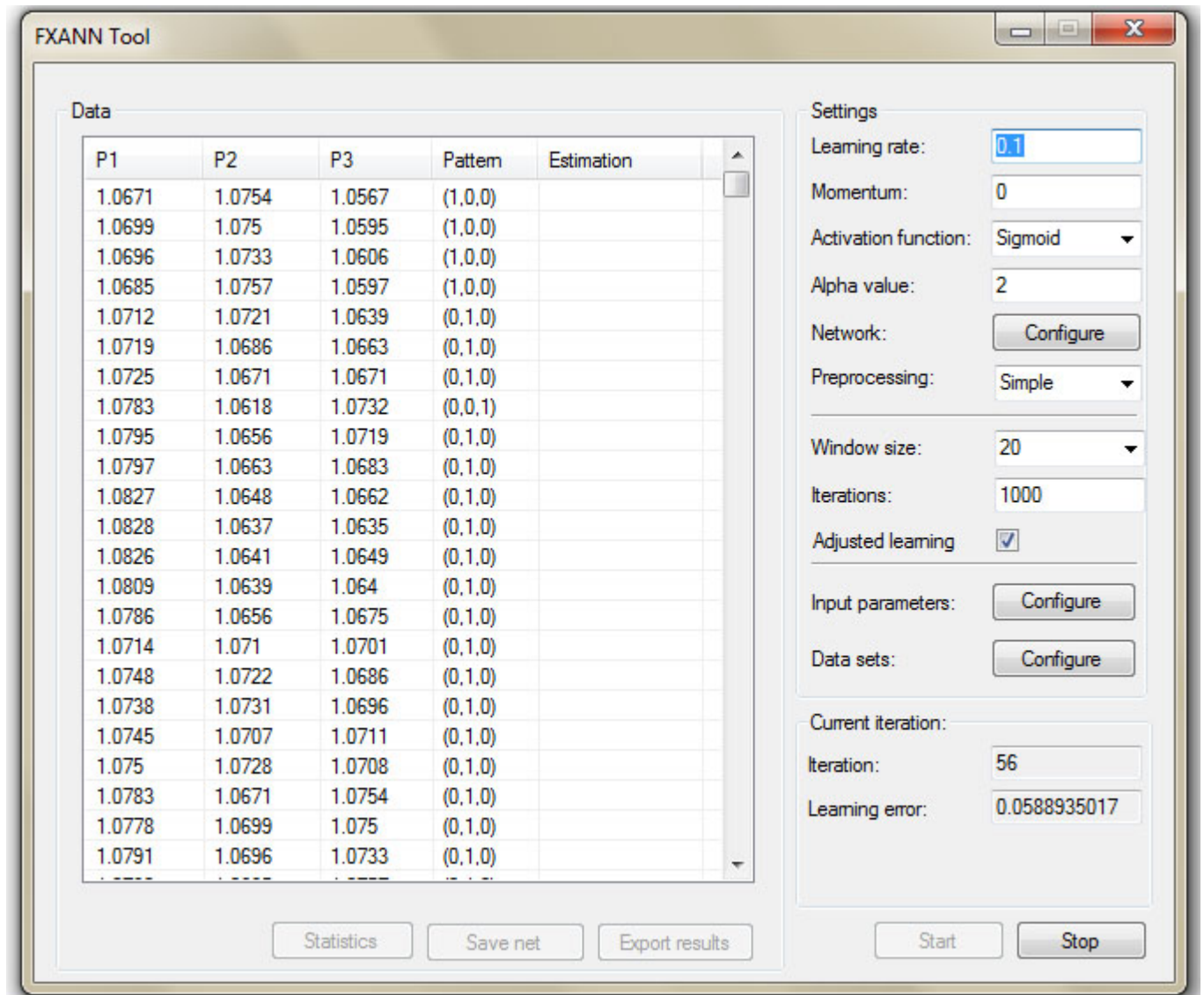


Figure 7.4: FXANN tool.

The last step is prediction itself. Trained network is launched with the testing data for prediction. After that, prediction results are displayed in the *Estimation* column of the *Data* list. If the network fails to predict pattern for some bar, *NA* value is generated. After, result prediction statistics are shown to the user.

### 7.2.1 FXANN settings

FXANN is a model that gives the user broad configuration possibilities. Parameters that may be determined using FXANN tool before training and prediction are the following:

1. **Learning rate.** Learning rate parameter is set by default on  $0.1$ , but may be changed. The value should be a number between 0 and 1, otherwise default value is used.
2. **Momentum.** Default momentum value is  $0$ . Entered value should also be a number between 0 and 1, otherwise default value is used.
3. **Activation function.** FXANN supports two activation function types: *sigmoid activation function* (called Sigmoid) and *bipolar sigmoid activation function* (called Bipolar). Default function is sigmoid.
4. **Alpha value.** Alpha value is the parameter for both sigmoid and bipolar sigmoid functions. To make learning process effective, its value should not be set too big or too small. Values smaller than 1 and greater than 2.5 are not favorable and may result in inefficient learning.
5. **Preprocessing method.** There are two options: simple preprocessing (5.3) and statistical preprocessing (5.4). By default FXANN employs simple preprocessing.
6. **Window size.** That parameter determines the size of the window for getting the past and the future rates. For example, if window size is 14, rates that are considered are: 14 bars past rate, the current rate, 14 bars future rate. By default, FXANN works with the 9 bars window.
7. **Iterations.** That parameter indicates number of training epochs. During one epoch, the network processes the whole training set. By default, that parameter is set to 1000 iterations.
8. **Adjusted learning flag.** FXANN allows indicating whether the adjusted learning should be used or not. Default flag value is *false*.



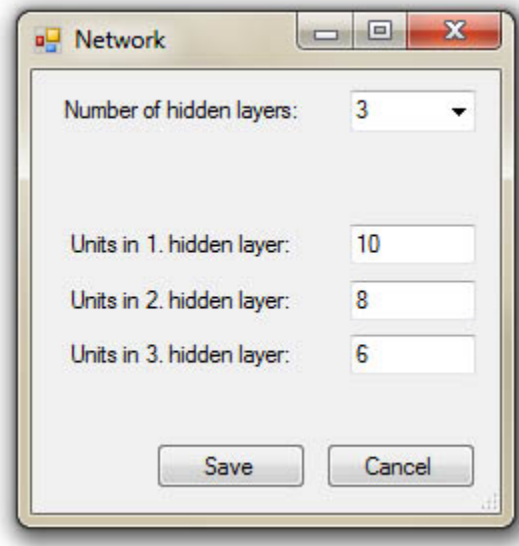


Figure 7.5: **FXANN network configuration.**

9. **Network.** The user may also define FXANN network architecture (Fig. 7.5). The network may contain up to 3 hidden layers. Default configuration is 1 hidden layer with 10 neurons.
10. **Input parameters.** FXANN tool allows the user configuring input indicators. List default indicators is shown on Fig. 7.3.
11. **Data sets.** By default, FXANN tool assumes that training and testing files are places in a special directory (".././data" directory) and are named "training.txt" and "testing.txt". That default setting may be changed by hitting "Configure" button at "Data sets".

### 7.2.2 FXANN report

After FXANN makes its prediction on the testing data, it shows general report that helps the user with the FXANN instance evaluation. Results are shown in separate window (Fig. 7.6) and contain the following parameters:

1. **Bar count.** Number of bars (samples) in testing set.
2. **Success rate.** Correct prediction rate for the whole testing set.

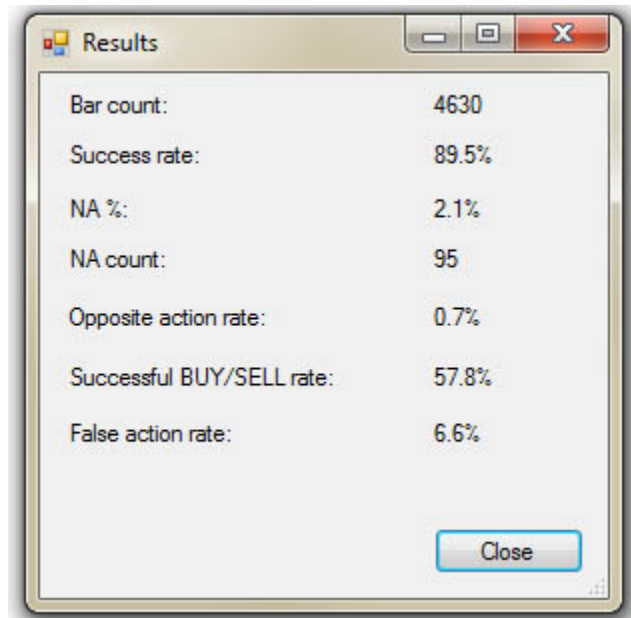


Figure 7.6: **FXANN report.**

3. **NA %.** "NA" failure signals rate among all signals.
4. **NA count.** Number of "NA" failure signals generated.
5. **Opposite action rate.** Opposite action signal is generated when instead of correct buying signal, selling signal is generated and vice versa. Those signals may potentially lead to large financial losses and knowing how many of them the FXANN instance generates may be crucial for its potential usage. Also, that rate is calculated not from the whole testing sets, but from buying and selling signals only.
  - For example, if the opposite action rate is 5%, it means that 95% of buying and selling signals were recognized correctly or as no-change signals. And 5% were recognized as opposite signal.
6. **Successful BUY/SELL rate.** That parameter shows how much buying and selling signals were recognized correctly. Successful identification of those signals is crucial for making profit. That rate is calculated from buying and selling signals only.
7. **False action rate.** That parameters shows how often no-change signal was recognized as buying or selling signal. High rate values mean that the FXANN instance may do a lot of unprofitable trades.



Figure 7.7: FXANN export buttons.

### 7.2.3 FXANN data export

FXANN tool allows exporting of both prediction results and FXANN network configuration including synaptic weights (Fig. 7.7).

1. **Results saving.** After clicking the *Export results* button and selecting text file name and location, the FXANN tool will save the FXANN instance parameters (e.g., network architecture, learning rate and momentum values) and final statistics (shown on the *Results* window).
2. **Network saving.** For saving trained network configuration - number of layers, neurons and synaptic weights of every neuron - the user should click the *Save net* button and select a text file for storing the information. The information is saved in MQL4 format and can be directly used as the part of the MQL4 advisor code.

### 7.2.4 FXANN requirements

For using FXANN tool, you must have:

1. **Operating system:** Windows XP, Windows Vista, Windows 7
2. **Software:** .Net Framework version 3.5
3. **Training and testing data:** Those files should be located in the same directory as the FXANN executable file. Their names should be *training.txt* and *testing.txt* accordingly.

## 7.3 Experimental simulation

FXANN model was tested on real data, received from the Alpari server (Fig. 7.8). Same as in previous chapter, I have used *USDCHF* M30 data. Data for year 2008 were used for training and data from 1.1.2009 to 19.05.2009 were used for prediction.

Training data contains 12321 bars. And testing set contains 4630 bars.

To select the most appropriate FXANN model, I've created more than 100 instantiations with different parameters. As the results, I've chosen several of them that gave the most accurate predictions. In that section I will discuss different parameters' impact on the system's prediction capability and then introduce and evaluate the most successful model.



Figure 7.8: USDCHF 2008-2009 data used for FXANN experiments.

### 7.3.1 Observations

After testing a big number of different models, I have made several empirical conclusions about FXANN parameters' values.

1. **Network configuration.** Regardless other parameters, the best results were achieved by using the network with only one hidden layer with rather big number of neurons - from 10 to 15. However, the neural network with two hidden layers, one of which contained approximately as many neurons as there were input indicators, and the second layer contained just a few neurons, also was able to generate acceptable results. Three-layered networks appeared to be slow and much less effective, regardless of the number of neurons in each layer. I also assume that more than three layers would be even more ineffective.
2. **Adjusted learning.** As expected, adjusted learning has brought significant improvements in recognizing buying and selling signals within all window sizes and input indicators' sets.

For example, for 20 bars window and default parameters, FXANN produced following results<sup>2</sup>:

---

<sup>2</sup>Because network's initial synaptic weights are set to small random values, different models with the same

**Table: Adjusted learning impact**

	<b>Without adjusted learning</b>	<b>With adjusted learning</b>
Success rate	91.1%	89.8%
NA %	2.6%	1.7%
Opposite action rate	0%	0%
Successful B/S rate	<b>29.9%</b>	<b>59.5%</b>
False action rate	3.3%	6.5%

3. **Window size.** If the window is too small or too large, technical indicators may not reflect current market situation comprehensively. When the window is too small, past, present and future values may be correlated. When the window is too large, rate changes are reflected by fundamental indicators much more than by technical indicators. Therefore, the use of technical indicators as input parameters may be less effective. As the result of experimentation, the window size of 20 bars was considered the most appropriate.
4. **Technical indicators.** To make learning and predicting more effective, the input parameters should cover the market situation as fully as possible. Also, number these parameters must be neither too high nor too low. Experimental results show that, for example, such input set may be made by two indicators with the small and large scope from each of four indicators' groups (*StdDev*, *RSI*, *ADX*, *CCI*). Example of the selection that worked well is shown on Fig. 7.3.
5. **Iterations.** Optimal number of iterations depends on the network architecture and the number of input indicators. However, experimental results show that the number of iterations should be in the range from 500 to 4000 or 5000. For the network with one hidden layer and 10 hidden neurons, optimal number of iterations is about 1000.

### 7.3.2 Selected model

For experimental simulation, I've selected the following model:

---

parameters may produce slightly different results.

**Table: Selected FXANN model**

Parameter	Value	Parameter	Value
Learning rate	0.1	Momentum	0
Activation function	Sigmoid	Preprocessing type	Simple
Network architecture	1 hidden layer with 10 neurons	Adjusted learning	Yes
Window size	20 bars	Alpha value	2

I have also selected the following indicators: *Deviation 25, Deviation 100, RSI 25, RSI 100, ADX 25, ADX 100, CCI 25, CCI 100, AD*.

As a currency pair, I have selected *USDCHF*. Whole year 2008 M30 historical rates were used for training, and M30 data from 1.1.2009 to 19.05.2009 were used for testing. The data characteristics are the following:

**Table: 20 bars window data**

	Training samples	Testing samples
Selling signal	308 (2.4%)	129 (2.7%)
Buying signal	439 (3.6%)	172 (3.7%)
No-change signal	11574 (93.8%)	4329 (93.6%)
Total	12321	4630

The low amount of buying and selling signals is expected. Because 20 bars is a rather small time frame, most rate changes within that interval are not significant (i.e., rates dispersion is low). For larger window there will be more action signals. For example, characteristics of the same data with the window size of 100 bars are the following:

**Table: 100 bars window data**

	Training samples	Testing samples
Selling signal	1099 (8.9%)	744 (16.0%)
Buying signal	1469 (11.9%)	703 (15.2%)
No-change signal	9753 (79.2%)	3183 (68.8%)
Total	12321	4630

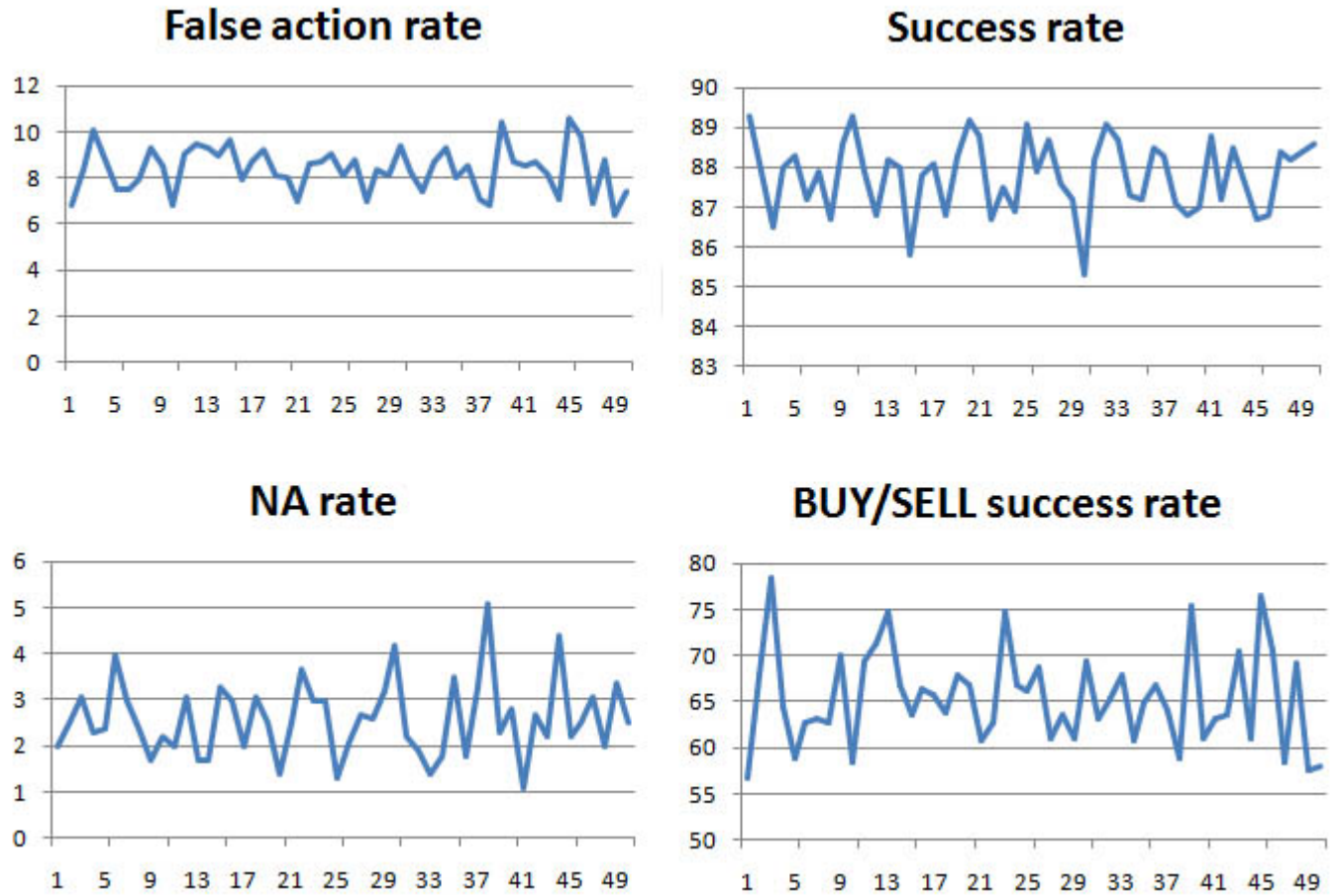


Figure 7.9: FXANN results of 50 different instances with the same parameters.

### 7.3.3 Prediction results

I have tested 50 FXANN instances with the same parameters (described in previous section). Because at the beginning synaptic weights are initialized randomly, all those models produced slightly different predictions. The results' summary is shown on Fig. 7.9.

Aggregated results are also shown in the following table:

**Table: Selected model results (window size = 20 bars)**

	<b>Min</b>	<b>Max</b>	<b>Mean</b>	<b>Sd deviation</b>	<b>95% confidence interval</b>
Success rate	85.3%	89.3%	87.8%	0.917	[87.54, 88.05]%
B/S success rate	56.8%	78.4%	65.43%	5.150	[64.00, 66.86]%
False action rate	6.4%	10.6%	8.38%	0.998	[8.10, 8.66]%
NA rate	1.1%	5.1%	2.60%	0.825	[2.37, 2.83]%
Opposite action rate	0%	4.7%	0.41%	0.861	[0.17, 0.65]%

For the comparison purposes, I have also selected 2 alternative FXANN models:

1. **Model 1.** That model is the same as the selected one, but instead of 20 bars window size it employs 100 bars window size.
2. **Model 2.** In addition to the selected model parameters, that model is based on more input indicators - it also adds *StdDev 9*, *ADX 9*, *RSI 9*, *CCI 9*, *AC*, *AO* and *MACD* indicators to the input set. And because of extended number of input parameters, it contains 15 instead of 10 hidden neurons.

Similarly as with the selected model, I have trained 50 instances for each of those two alternative models. Aggregated results for 50 instances are displayed in the following tables:

**Table: Model 1 results (window size = 100 bars)**

	<b>Min</b>	<b>Max</b>	<b>Mean</b>	<b>S. deviation</b>	<b>95% confidence interval</b>
Success rate	56.1%	68.7%	62.4%	2.817	[61.66, 63.22]%
B/S success rate	22.6%	73.0%	54.95%	8.834	[52.50, 57.40]%
False action rate	5.7%	32.3%	22.08%	4.590	[20.81, 23.35]%
NA rate	9.0%	29.5%	14.67%	4.099	[13.54, 15.81]%
Opposite action rate	0%	8.8%	2.70%	2.365	[2.04, 3.35]%



**Table: Model 2 results (window size = 20, additional input indicators)**

	Min	Max	Mean	S. deviation	95% confidence interval
Success rate	78.8%	92.9%	89.5%	2.608	[88.76, 90.21]%
B/S success rate	18.9%	64.1%	35.04%	9.251	[32.48, 37.61]%
False action rate	1.7%	11.1%	4.42%	1.963	[3.87, 4.96]%
NA rate	0.6%	9.2%	2.69%	1.591	[2.25, 3.14]%
Opposite action rate	0%	9.6%	1.18%	1.708	[0.71, 1.66]%

As shown in tables above, comparing to alternative models, the selected FXANN model has rather high buying and selling signals recognition rate (about 65.43%). It also has a high overall prediction rate (about 87.7%), especially comparing to the Model 1. Taking into account the *Opposite action rate* parameter, that is rather low (i.e., less than 0.5% in average), buying and selling signals that were not recognized correctly, were recognized as no-change signal or prediction was failed. That by-turn means that not all favorable buying and selling situations will be recognized, but that will not lead to somehow significant financial losses.

The other potential "loss spot" is the situation when no-change signal is recognized as an action (buying or selling) signal. That misprediction could also cause potential financial losses. However, the selected model does not have a high *False action rate* - only about 8.38%. Losses caused by that signal may be avoided by setting *stop loss signal value* that will close the position as soon as it becomes potentially highly unprofitable.

Finally, the most dangerous situation is so-called *opposite action*, when buying signal is recognized as selling signal and vice versa. But experiments shown that the overall *Opposite action rate* is insignificant that those situations will appear rarely.

Low success rates and high misprediction and failure rates of the Model 1 (comparing to the selected FXANN model) imply high dynamism of the USDCHF market. 100 bars window size appears to be too large and selected technical indicators are not able to describe all the variety of changes that may happen between the past and future rates for every bar. However, despite that 100 bars window is not applicable to the USDCHF pair, it does not mean that it is not a generally good choice - that window may work well for some other currency pair.

Furthermore, the Model 2 results show that selecting too many technical indicators as input parameters is not a good choice (at least on the USDCHF market). Additional indicators slightly improved the ability to recognize no-change situation, but significantly reduce the ability to recognize buying and selling situations. As results' table indicates, the most of those situations were recognized as no-change signals.

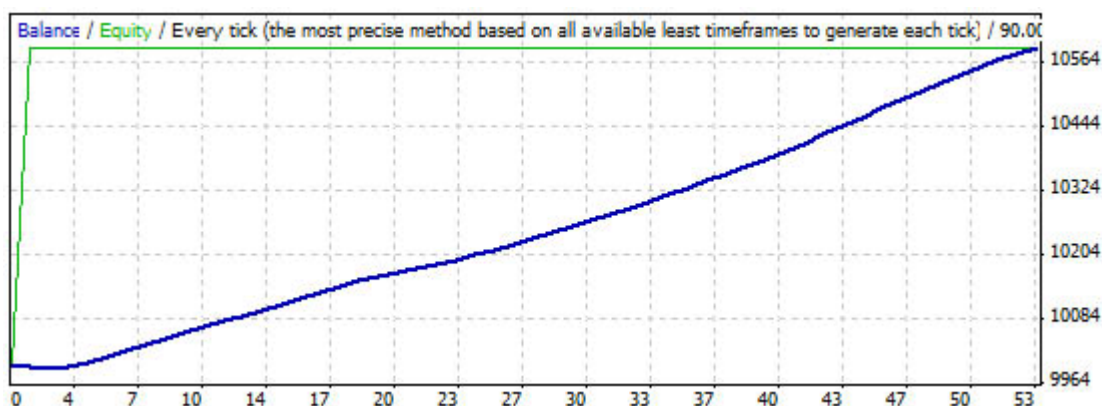


Figure 7.10: FXANN-based advisor simulation results

### 7.3.4 Simulation results

In order to test FXANN on Forex, I set up an automated advisor (*FXANN-based advisor*) that is based on a random instance of the selected model. The advisor commits financial transactions based on signals received from the FXANN model. Similarly as with the SOM-advisor, FXANN advisor also takes into account stop loss and take profit levels. Similarly as in SOM-based advisor, those values were set to 15 and 46 points accordingly.

The advisor was tested on the exactly same testing data and FXANN model - USDCHF M30 rates from 1.1.2009 to 19.05.2009.

Results of the simulation are shown in the following table:

**Table: FXANN simulation results**

	<b>FXANN advisor</b>
Total net profit	590.22\$
Total trades	53
Expected payoff	11.14\$
Profit trades (% of total)	96.23%

Fig. 7.10 shows FXANN-based advisor financial results.

### 7.3.5 Model evaluation

In order to make sure that the selected FXANN model is stable, I have also performed two additional tests - robustness test and independent data set test.

Robustness test aims to ensure that the model is stable. For that purpose I've created 50 FXANN model instances and then changed all their ANN synaptic weights. The amount of change was 5% of the original value and that change was added to or subtracted (that action was determined randomly) to/from the original synaptic weight value.

**Table: Robustness test results ( $\pm 5\%$  change)**

	Min	Max	Mean	S. deviation	95% confidence interval
Success rate	80.9%	90.5%	86.8%	2.102	[86.20, 87.36]%
B/S success rate	47.2%	83.4%	63.29%	7.140	[61.32, 65.28]%
False action rate	4.8%	13.1%	8.15%	1.644	[7.69, 8.61]%
NA rate	1.6%	11.7%	3.87%	1.772	[3.37, 4.36]%
Opposite action rate	0%	4.3%	0.43%	0.839	[0.20, 0.66]%

Results' table indicates that despite that overall prediction ability of the model was slightly decreased, the selected model is stable and is able to produce quality results even when slightly noised.

Second test was performed in order to evaluate model's prediction ability within different data set from the same domain. For that purpose, I have selected USDCHF M30 data for the following several month: from 19.5.2009 to 17.09.2009. The characteristics of the alternative testing set are the following:

**Table: Alternative testing data**

	Testing samples
Selling signal	57 (2.7%)
Buying signal	116 (5.6%)
No-change signal	1913 (91.7%)
Total	2086

Following table is based on 50 different FXANN instances and shows prediction results on the new data set:

**Table: Independent data set test results**

	Min	Max	Mean	S. deviation	95% confidence interval
Success rate	75.8%	88.9%	85.8%	2.319	[85.14, 86.43]%
B/S success rate	31.8%	72.3%	55.10%	8.493	[52.75, 57.46]%
False action rate	5%	14.2%	8.35%	1.842	[7.84, 8.86]%
NA rate	1.6%	14.8%	3.89%	2.085	[3.31, 4.47]%
Opposite action rate	0%	1.7%	0.16%	0.449	[0.04, 0.27]%

As expected, the model's performance decreased - it was able to recognize about 55.1% of buying and selling signals, comparing to 65.43% of the original results. It also ended up with higher deviation values. However, overall performance of the model did not change much and remained satisfactory.

## 7.4 Conclusions

In that chapter I have introduced and evaluated the model based on a multi-layer neural network. Trading and prediction results of the selected model shown that neural networks have the ability to successfully classify Forex trading situations. Experimental simulation applied to the real data has demonstrated that the prediction system generates buying and selling signals with the high success rate.

The model was also able to successfully classify trading situation of the completely new testing set from the same domain. Despite the fact that final results of the model were slightly worse then on the original testing set, results still remained reasonable and solid.

# Chapter 8

## Conclusions and future work

In that thesis I have presented two approaches of how to create ANN-based automated trading advisors. The first advisor was based on Kohonen's self-organizing maps and the second one was based on the back propagation network. Both models worked with technical indicators instead of currency rates. Both models tried to recognize situations that were favorable for buying or selling actions.

Kohonen's maps were not able to recognize buying or selling situations. Moreover, indicators' buying and selling clusters were very similar. That fact was used for extraction of information about indicators' intervals that were favorable for taking action. Despite the fact that the nature of action was unknown and the advisor was coerced opening both buying and selling positions, setting stop loss and take profit levels contributed to its significant profitability.

However, SOM-based advisor was developed rather as a "proof of concept" than as a proper model - only one network was trained, only one set of indicators was tested and no statistical evaluation was performed. The idea appeared promising and further developing and evaluation of the SOM-based model may become one of the possible directions for future work.

In contrast, FXANN model was able to recognize buying and selling situations with the high probability. The advisor that was based on that model also appeared to be profitable. That model was selected after more than 100 different models' tests and significantly outperformed other models. FXANN model also was tested for robustness and input space approximation and both tests results were satisfactory - the model was able to produce reasonable results on other than training set, as well as with noised weights.

Also, as the part of my project I have created a tool that can be used by traders without understanding of complex neural networks concepts. As for today, programs like the one I've developed are not available for traders and according to the fact that many of them try using

neural networks in their attempts of making profit, that tool may become very handy to them.

As future work, there are still plenty of possibilities of how to continue that research. Among them:

- As mentioned before, SOM-based model could be developed and properly tested.
- The thesis is based on the same one data set - USDCHF M30 data. Testing those models on different currency pairs and different time frames may also become a possible direction for future work.
- Both models could be tested towards different Forex parameters - such as take profit and stop loss levels.
- Different types of networks could be examined (e.g., Probabilistic neural network (PNN)).
- Different types of learning methods could be used as well (e.g., [29]). Also, error function can be optimized in order not to minimize error, but to maximize profit.

# Appendix A

## Technical indicators

### A.1 Commodity Channel Index (CCI)

CCI indicates how much current rate deviates from the average price. The index is designed such as 70% to 80% percent of CCI values fall between -100 and +100 (Fig. A.1). High values of the index point out that the price is unusually high being compared with the average one, and vice versa for low price. That index is calculated based on predefined number of bars (e.g., CCI 25 - indicator for the last 25 bars).

### A.2 Relative Strength Index (RSI)

The RSI compares the upward price movement to downward price movement over the specified time frame, and displays the result as a momentum line oscillating between 0 and 100 (Fig. A.2). It measures the current and historical strength or weakness of a market. RSI assumes that prices are higher in strong market periods and lower in weak market periods. That indicator is based on some predefined number of bars (e.g., RSI 14).

### A.3 Average Directional Movement Index (ADX)

Average Directional Movement Index Technical Indicator (ADX) helps to determine if there is a price trend as it calculates the strength of the upward or downward movement. An ADX level below 20 is considered low. It is a strong indicator of a currency trading within a range. When the ADX is at 25, the strength of a trend is growing, but still may not be strong enough yet to break out of the range (Fig. A.3).

ADX indicator exists into 3 several modes:



Figure A.1: CCI index ([35]).





Figure A.2: **RSI index** ([35]).



Figure A.3: ADX index ([35]).

1. **MAIN.** Standard indicator value (*blue line* on (Fig. A.3)).
2. **PLUSDI.** *+DI* value (*green line* on (Fig. A.3)).
3. **MINUSDI.** *-DI* value (*red line* on (Fig. A.3)).

Here *+DI* and *-DI* state for *positive/negative directional index* respectively. Those are calculated by comparing the current price with the previous price range, and displayed as an upward movement line (+DI), and a downward movement line (-DI) with the values between 0 and 100.

## A.4 Momentum

The Momentum indicator is a speed of movement (or rate of change) indicator, that is designed to identify the speed (or strength) of a price movement (Fig. A.4). Usually, the momentum indicator compares the most recent closing price to a previous closing price, but it can also be used on other indicators such as moving averages.

## A.5 Standard Deviation (StdDev)

Generally, standard deviation is a statistical measurement that shows how much variation there is from the arithmetic mean (simple average). Traders use technical indicator named Standard Deviation (StdDev) for measuring the market volatility - quantification the risk of the financial instrument over the specified time period. Thus, if the indicator value is large, the market is volatile and the bars prices are rather dispersed relating to the moving average. If the indicator value is not large, it means that the market volatility is low and the bars prices are rather close to the moving average (Fig. A.5).

## A.6 Average True Range (ATR)

ATR indicator measures predictability of the market (Fig. A.6). Average True Range can often reach a high value at the bottom of the market after a sheer fall in prices occasioned by panic selling. The indicator does not provide an indication of price direction or duration, simply the degree of price movement. ATR is also based on periods (e.g., ATR 14).



Figure A.4: Momentum index ([35]).



Figure A.5: Standard deviation ([35]).

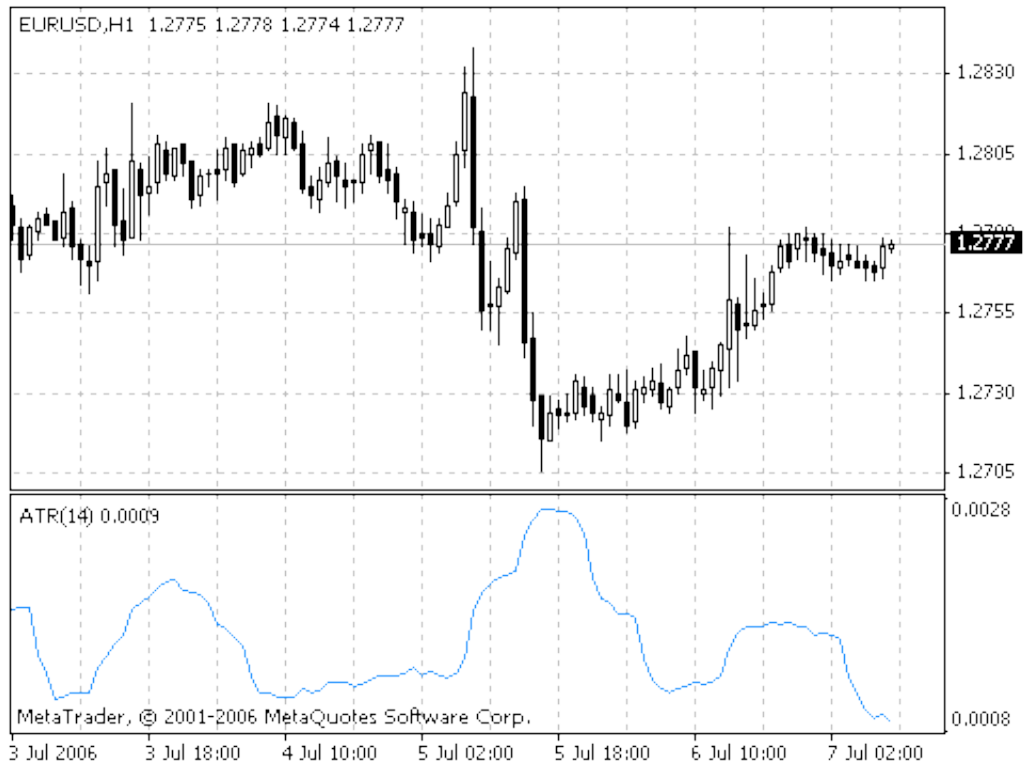


Figure A.6: ATR index ([35]).



Figure A.7: AC index ([35]).

## A.7 Bill Williams' Accelerator/Decelerator (AC)

Acceleration/Deceleration Technical Indicator (AC) measures acceleration and deceleration of the current driving force (Fig. A.7).

## A.8 Bill Williams' Awesome oscillator (AO)

Awesome Oscillator (AO) is simply the difference between the 34-period and 5-period simple moving averages of the bar's midpoints  $(H+L)/2$ . Awesome Oscillator (AO) is displayed on Fig. A.8.



Figure A.8: AO index ([35]).





Figure A.9: AD index ([35]).

## A.9 Bill Williams' Accumulation/Distribution (AD)

AD indicator that attempts to gauge supply and demand by determining whether investors are generally "accumulating" (buying) or "distributing" (selling) a certain currency by identifying divergences between stock price and volume flow (Fig. A.9).

# Bibliography

- [1] Adya M., Collopy F.: "How Effective are Neural Networks at Forecasting and Prediction? A Review and Evaluation", *Journal of Forecasting*, University of Maryland at Baltimore County, USA, pp.481-482, 1998
- [2] Berry M., Linoff G.: "Data Mining Techniques for Marketing, Sales and Customer Support", Wiley Publishing, 1997
- [3] Bishop C.M.: "Neural networks for pattern recognition", Oxford University Press, 2008
- [4] Chen S.H.: "Computationally intelligent agents in economics and finance", *Information Sciences*, vol. 177, Issue 5, pp. 1153-1168, 2007
- [5] Chester D.L.: "Why Two Hidden Layers are Better than One", *IJCNN-90-WASH-DC*, Lawrence Erlbaum, vol. 1, pp. 265-268, 1990
- [6] Coupelon O.: "Neural network modeling for stock movement prediction A state of the art" (available online), 2006
- [7] Doudge N.: "The Brain That Changes Itself: Stories of Personal Triumph from the Frontiers of Brain Science", Penguin Books, 2007
- [8] Dunis C.L., Williams M.: "Modeling and trading the EUR/USD exchange rate: Do neural networks perform better?", *Derivatives Use, Trading & Regulation*, No 8/3, pp. 211-239, 2002
- [9] Fausett L.: "Fundamentals of Neural Networks: Architectures, Algorithms And Applications" , Prentice Hall, 1993
- [10] G. W. Flake: "Square Unit Augmented, Radially Extended, Multi-layer Perceptrons", *Lecture Notes In Computer Science*, vol. 1524, "Neural Networks: Tricks of the Trade", pp. 145-163, 1998.

- [11] Flexer A.: "Statistical evaluation of Neural Network Experiments: Minimum requirements and Current practice", The Austrian Research Institute for Artificial Intelligence, A-1010, pp.1005-1008, 1994
- [12] Frank R.J., Davey N., Hunt S.P.: "Time series prediction and Neural networks", Journal of Intelligent and Robotic Systems, vol. 31, No 1-3, pp. 91-103, 2001
- [13] Haykin S.: "Neural Networks: A Comprehensive Foundation", Prentice Hall, 1998
- [14] Huang W., Wang S., Yu L., Bao Y., Wang L.: "A new computational method of input selection for stock market forecasting with neural networks", ICCS, vol. IV, LNCS 3994, pp. 308-315, 2006
- [15] Jacobs R.A.: "Increased rates of convergence through learning rate adaptation", Neural Networks, vol. 1, pp. 295-307, Amherst, Tech. Rep. 1988
- [16] Kaastra I., Boyd M.: "Designing a neural network for forecasting financial and economic time series", Neurocomputing, vol. 10, pp. 215-236, 1996
- [17] Kim K.J.: "Artificial neural networks with evolutionary instance selection for financial forecasting", Expert Systems with Applications, vol. 30(3), pp. 519-526, 2006
- [18] Kohonen T.: "Self-Organizing Maps", Springer, 3rd edition, 2001
- [19] Krantz M.: "Fundamental analysis for dummies", Wiley Publishing, Inc., 2009
- [20] Kudova P.: "Learning methods for RBF networks", Master Thesis, Charles University, Faculty of Mathematics and Physics, Prague, 2001
- [21] Lawrence S., Giles C.L., Tsoi A.C.: "Lessons in Neural Network Training: Overfitting May be Harder than Expected", Proceedings of the Fourteenth National Conference on Artificial Intelligence, AAAI-97, AAAI Press, Menlo Park, California, pp. 540-545, 1997
- [22] Packard N., Crutchfield J., Farmer D., Shaw R.: "Geometry from a time series", Physical review letters 45, pp. 712-716, 1980
- [23] Pyle D.: "Data Preparation for Data Mining", Morgan Kaufmann, 1999
- [24] Rockefeller B.: "Technical analysis for dummies", Wiley Publishing, Inc., 2009.
- [25] Sarker R.A., Kamruzzaman J.: "ANN-based forecasting of foreign currency exchange", Neural Information Processing, pp. 49-58, 2003

- [26] Schwartz J.M., Begley S.: "The Mind and The Brain", Harper Perennial, 2003
- [27] Skabar A., Cloete I.: "Neural networks, Financial trading and Efficient markets hypothesis", Conferences in Research and Practice in Information Technology Series, vol. 17, pp. 241-249, 2001
- [28] Shumsky S.A.: "Selected Lectures on Neurocomputing" (in Russian, available online), Prima Press, 1998
- [29] Tilakaratne C.D., Mammadov M.A., Morris S.A.: "Modified neural network algorithms for predicting trading signals of stock market indices", Journal of Applied Mathematics and Decision Sciences, vol. 2009, 2009
- [30] Willshaw D.J., C von der Malsburg: How patterned neural connections can be set up by self-organization, Proceedings of the Royal Society of London Series B, vol. 194, pp. 131-145, 1976
- [31] Yin H.: Learning Nonlinear Principal Manifolds by Self-Organizing Maps, Series: Lecture Notes in Computational Science and Engineering , vol. 58, pp. 431-451, 2008
- [32] Yao J., Tan C.L.: "Guidlines for financial forecasting with neural networks", Proceedings of International Conference on Neural Information Processing, pp. 14-18, 2001
- [33] Zorin A.: "Stock price prediction: Kohonen versus back propagation" (available online), 2002
- [34] "EarnForex: Serving trades" web page: <http://www.earnforex.com>
- [35] MetaTrader "Technical indicators" web page: <http://ta.mql4.com/indicators>
- [36] "Investopedia" web page: <http://www.investopedia.com>