Forecasting Financial Time Series with Artificial Neural Networks

Elizaveta Okorokova

June 21, 2014

Abstract

Financial forecasting has always been one of the biggest challenges in theoretical and practical Finance. Because of the specific nature of most financial time series, which tend to be non-Gaussian, non-stationary, chaotic and extremely noisy, most traditional models fail to accomplish the task of efficient data fitting and forecasting and, therefore, need to be replaced by a more powerful analytical tool. Recently, there has been a growing belief that Neural Networks, which are used effectively is pattern recognition and classification, might as well be a panacea for at least some problems in financial data analysis.

In this paper we apply feed-forward multilayer Neural Network to forecast stock prices of five largest companies in Russia and evaluate the possible gains and losses of trading using our Network. We find that, despite the outstanding ability of the Network to approximate the data, the standard MLP is unable to generate abnormal profits to an investor when it is used to forecast one-month-ahead stock returns. Moreover, we find that increasing the running time of the cost-minimization algorithm does not necessarily improve out-of-sample performance of the Network.

Contents

1 Introduction

Artificial Neural Networks (ANNs) are a universal analytical tool for parallel data processing. The idea behind the first Artificial Neural Networks was to create a highly intelligent system, which would mimic the enormous abilities of the human brain. Neural Networks have a number of distinguishing features, which makes them extremely attractive to researchers in many fields.

Firstly, as opposed to most traditional econometric models, which require accurate prior specification as well as careful data adjustments, Neural Networks are adaptive, self-driven systems, which need just a small number of highly flexible assumptions.

The ability of ANNs to learn by example makes them skilful at finding internal relationships between variables, which may not be evident at first sight. Because of this feature ANNs, have proved to be a useful tool in data analysis, especially dealing with non-linear, highly volatile and noisy data.

Secondly, Artificial Neural Networks are universal approximates. It can be shown ([18]) that even a simplest 3-layer network can approximate any functional relationship with a high degree of accuracy. This is a goal, traditional models often fail to achieve, in that they require exact knowledge of the relationship between the variables, which are often very complex and, thus, hard to specify.

Thirdly, Neural Networks have an important ability to generalize data. Once the approximation is fit, the relationship can be extrapolated out-of-sample. Because of these valuable features of Neural Networks, it is not surprising that they have become an increasingly popular method in time series forecasting. Since the first works in the area ([35],[39]) many new methods in artificial intelligence have been explored and successfully developed

In this paper we build a multilayer feed-forward network to forecast time series of the Russian stock market. We use stock prices of the five largest corporations in Russia to test the ability of our network to correctly forecast stock prices out-of-sample. We analyse performance of our network using several indicators, which are found in recent literature.

The particular interest of our work is the attempt to analyse practical importance of accurate forecasting. After fitting the network, we evaluate possible prospects of technical trading based on MLP Network. We use the rolling window approach to construct the Profit and Loss account for 12 periods of 2013, based on the out-of-sample forecasting with our model. The main hypothesis we test is, whether trading rules based on the Network forecast can outperform the market.

Further, we investigate how changing the accuracy of training algorithm influences the ability of the Network to forecast stock prices out-of-sample. We suppose that increasing the number of iterations in the training algorithm may lead to a better forecast performance. To test this hypothesis we compare 2 models with different training times, using the Diebold-Mariano test.

A great deal of this paper also relates to introducing the theory of Artificial Neural Networks to a reader, which might be a good starting point for anyone who wish to understand the underlying concepts of ANNs.

The paper is structured as follows. Section two provides a brief overview of recent literature in the field of forecasting using ANNs. The next section is optional and is written to introduce the reader to Artificial Neural Networks as well as their background from biology. Those already familiar with the topic can proceed directly to section four, which gives an overview of the data, descriptive statistics and figures. Section five introduces the model and methods used in data analysis. The further section analyses the performance of our Network using several goodness-of-fit indicators and evaluates the two competing models with Diebold-Mariano test. Finally, the last section summarizes the main results of the study.

In the Appendix we provide the framework for coding the multilayer feed-forward network with variable dimensions, using R programming language(B), as well provide supplementary graphs, figures and tables (A).

2 Literature Review

Artificial Neural Networks were known for at least half a century and have gained popularity in many fields, including artificial intelligence, medicine and neuroscience. Literature on Neural Network application is wide and fruitful and here, we would summarize only a small fraction of all the available works on the topic. An interested reader may find it helpful to consult some wider literature reviews in [42], [?], [?], [32] including guides to AANs in finance [41] and management [22]

Despite its background in neurobiology, the early development of Neural Networks had significant contribution from physics, because ANN seemed to be a prospective tool when dealing with non-linearities, which are one of the main issues in physics [29]. Later on, Neural Networks became widely known as powerful tool, applicable to a large set of problems, Therefore, its application spread speedily to almost all spheres [?].

Of course, ANNs are used in neurobiology to model real processes happening in human and animal brain, such as the study of bird-song acquisition. The conjunction between natural and artificial neural networks is a bit tough, since they appear to be very different, but with the help of the latter, biologists may have a better understanding of how processes in the brain are modelled.

One of the most interesting areas where ANNs are widely applied is Artificial Intelligence. Modern algorithms for pattern recognition and classification enable scientists from this field to construct highly intelligent machines and systems, which are capable of dealing with tasks such as understanding hand-written text or mimicking sounds of the environment. For many exciting papers in the field consult Andrew NG [1]and others.

Nowadays Neural Networks have also gained popularity in marketing [11],[17] and [19] and predicting consumer behaviour [40]

Neural Networks are used in bank performance [36] and bankruptcy prediction [37], [25] and macroeconomic variables, such as predicting recessions and inflation [33].

Among the interesting applications is are problems from sociology and education, including a few works on graduate student performance prediction based on ANNs, with a number of indicators such as GMAT scores, average GPA. attendance and work experience as the training variables of the network [13],[31], [12].

Since this paper deals mostly with forecasting financial time series, it

[1]urlhttp://cs.stanford.edu/people/ang/

would be useful to relate to some previous research in this direction.

One of the first works on forecasting with ANNs was made by Lapedes and Farber (1987) [24], who used two chaotic time series and the Glass-Mackey equation to prove the ability of ANNs to mimic the non-linear time series with impressive accuracy. However, comparing to most other methods used in data forecasting, neural Networks are rather new, and according to [7] are one of the most prospective tools in the future of time series forecasting.

Adya and Collopy [1] and Hill et al [16] provide a comparative study of all the available works in forecasting with ANNs, stressing whether the researchers found neural networks useful in such framework.

Charkraborty at al (1992)[5] and Balkin et al [3] presented the ANN approach to analysis of the multivariate time-series, De Groot [8] concentrate on ANNs based on univariate time series. Ghiassi et al [6] forecast time series events with neural networks.

A popular topic in publications is comparing ANN performance with other statistical methods, including vast literature on empirical comparison of Box-Jenkins ARIMA model with a Neural Network model. Foe example, Khashei and Bijari [21] found that ANNs can significantly outperform the linear ARIMA(p,d,q) model.

McNelis (2005) provides a good book for artificial neural networks in Financial Applications [28].

There is also a number of good books and theoretical notes, highly recommended for anyone interested in understanding Neural Networks [34], [15], [27].

3 Neural Networks: From Neuron to Intelligent Systems

The history of Artificial Neural Networks goes back to 1943, when a neurophysiologist Warren McCulloch and a logician Walter Pitts built their first computational model of neural interaction [30], also called the threshold logic. Since then, mathematical models of neural networks have been largely extended and modified, but the idea behind them remains inviolable. Idealistically, it is to build a tool, no less intelligent and adept than the human brain.

Just as their name suggests, Artificial Neural Networks were inspired by biological nervous systems. This section serves as an introductory chapter to Artificial Neural Networks and, hopefully, gives a reader an essence of this fascinating topic.

Finally, there is a number of introductory textbooks on Neural Networks, which are useful for anyone wishing t understand Artificial Neural Networks in action.

3.1 Inspiration

3.1.1 Biological Neuron

A basic unit of a biological nervous system is called an nerve cell, or simply a neuron. There are more than 100 billion neurons in a human body.

Neurons are very different in their shape, size and functions. Most neuron bodies are very small and invisible to a human eye, however, ramifications from the cell body can be disproportionately long. For example, nerve cells in a human sciatic nerve, which starts at the bottom of the spinal cord and goes through the buttock down the lower limb, reaching the toe, contains cells which may have axons of up to a meter long.

Despite an enormous diversity of nerve cells in a human body, most of them have very similar structure. Figure1 shows the main parts of a generalized neuron.

The main processor of the nerve cell is the cell body. It contains a nucleus, which stores the cell's genetic material organised in long DNA molecules, and other important organelles, such as mitochondria and Golgi apparatus, which are responsible for keeping the cell alive.

The dendrites are numerous ramifications from the cell body, which serve as the receiving wires. They pick up signals from other neurons or from sensory organs, such as the skin, and pass them to the cell body, where information is processed and the response is formed.

The axon is the longest and the unique ramification of the neuron. It picks up the response from the cell body and takes it down its length to the

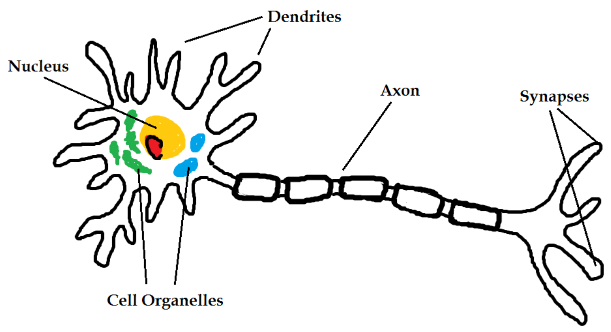


Figure 1: Generalized Neuron

axon terminals, which are also called the synapses.

3.1.2 Neural Interaction

Neurons, unlike many other types of cells, do not exist on their own. Instead, they are organized in layers or sequences, which form interconnections between each other (Figure2). In Neuroscience, such structures of neurons are called Neural Networks.

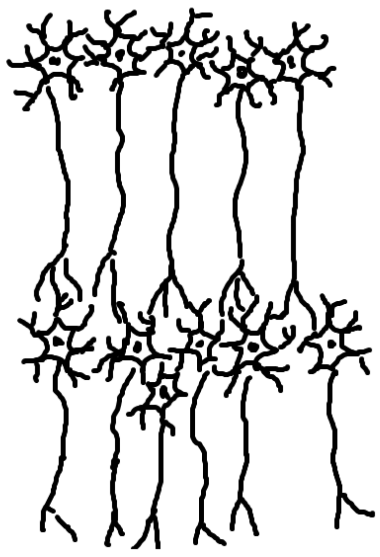


Figure 2: Neural Layers

Almost every process in a human body is governed by collective performance of neurons in the nervous system. Neural networks are capable of everything: beginning from the essential basics, like breathing and heartbeat, up to the most advanced cognitive abilities, such as writing poetry.

3.1.3 The Power of Ions

Neurons "talk" to each other by sending electrical impulses, shortly called the spikes. But, unlike metal wires, which conduct electricity with the help of free electrons, neural membranes have a completely different chemical structure.

The human body mainly consists of water, which fill the space inside and outside of the cells. Salts, which are naturally present in this water solution, dissociate into ions - charged atoms, which have either lost some electrons on their outer energy level (positive ions), or, on the other hand, have more electrons than necessary to balance the number of protons in their structure (negative ions). For example, sodium chloride, mixed with water, dissolves into positive sodium ions (Na^+) and negative chloride ions (Cl^-). Other ions, like potassium (K^+) and calcium (Ca^{2+})are also widely present in the nervous system environment.

Importantly, the cell membrane, which is a double layer of lipids protecting the cell, does not let ions flow freely between the inside and the outside of the cell. Instead, the membrane contains a limited amount of open channels for each specific ion. This guarantees that the cell membrane would not allow more ions to flow in, and, therefore, keep their concentration stable.

When the cell is at rest, meaning that there is no stimulation from the environment, it is said to be negatively charged at about -70mV with respect to the inter-cellular space. Such condition is called a resting membrane potential.

However, neurons are very rarely at rest. More realistically, they constantly receive stimulation from other neurons, or from sensory organs of the body. The signals between neurons travel by electrical impulses, called the Action Potentials. They are so fast that they can only be detected as spikes on the oscilloscope screen (Figure3). The more stimulation the neuron gets, the higher is the frequency of those spikes on the screen.

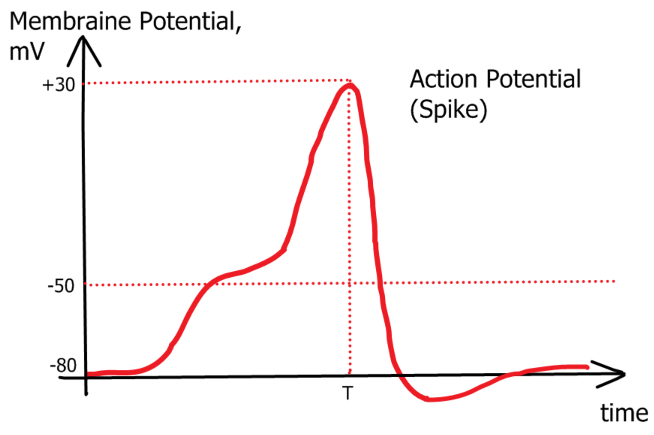


Figure 3: Action Potential on Oscilloscope screen

3.1.4 Learning through Synapses

Action Potentials play a very important role in leaning, which is the striking feature of Neural Networks that makes them so unique and interesting.

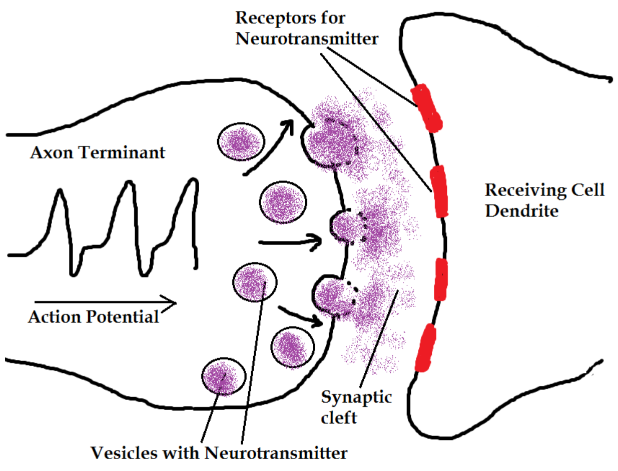


Figure 4: Neural Layers

Figure4 shows an enlarged synapse - an area, where the axon of one neuron meets the dendrite of the other. This is a space, where communication between neurons happen.

When the action potential travels down the axon, it generates the emergence of bubbles, called vesicles, which are filled with a special chemical - a neurotransmitter. Under electrical force, vesicles fuse with the cell membrane at the end of the axon and gradually release the neurotransmitter to the synaptic cleft, which, in turn, reaches the other cell's membrane and permeates through special channels inside the receiving neuron.

Depending on the type of neurotransmitter, released in the cleft, the receiving neuron can be either aroused or inhibited. In the former case, excitatory influence would increase the probability of action potential to arise at the receiving cell. In the latter, the inhibitory signal would weaken the ability of the cell to generate the action potential in response to stimulation.

Normally, neurons receive more than one signal at a time. Therefore, the ability of the cell to generate an action potential depends on the combination of signals coming from all dendrites. If the sum of all signals of different intensity forms a large enough stimulation, the neuron will fire. Otherwise, the stimulation would not have any effect. Normally, for a neuron to fire, the potential inside a cell has to rise to about +30mV.

Mathematically, this can be represented by a so-called threshold stimulation. Suppose signals are arrived at N channels, and each channel is associated with some weight (w_i), i.e. the strength of stimulation is different at each synapse. Then the total signal received by the cell is simply a weighted sum of all synapses:

$$TotalSignal = \sum_{i=1}^{N}(w_i * Signal_i) \tag{1}$$

The threshold logic is then summarized by a simple activation function of a type:

$$Activation = \begin{cases} Fire, & TotalSignal \geq \widetilde{S} \\ Rest, & TotalSignal < \widetilde{S} \end{cases} \tag{2}$$

Because, synapses are so important in signal processing, they are said to be the main storage of information in Neural Networks. What is more, synaptic connections are not stable and can change with time.

Suppose, there is a synapse connecting neuron A to neuron B, and suppose that, initially, an impulse from neuron A was too weak to stimulate neuron B on its own. However, if neuron A continues firing in direction of neuron B, the amount of neurotransmitter released into the synaptic cleft would eventually be enough to generate an action potential in neuron B. Such process would strengthen the synapse between the neurons. As a result, even a minor release in neurotransmitter from cell A into the synapse, would give rise to action potential in neuron B.

This idea of synaptic plasticity was first postulated by Donald Hebb [14] and is still considered a fundamental principle of learning. If the synapse is used more, it gets strengthened, and the weight associated with this synapse increases. Similarly, synapses, which are not used for a long time, lose their potency to pass the signal, so their weights in the network decrease.

3.2 An Artificial Neuron

Figure5 shows a single artificial neuron.

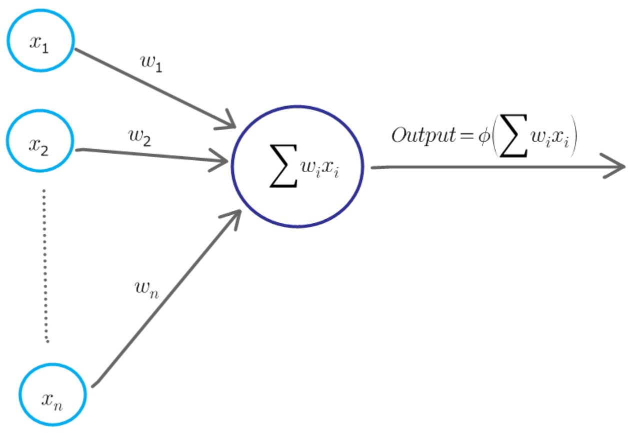


Figure 5: A Single Artificial Neuron

Just as its biological precursor, an artificial neuron receives inputs from N incoming sources: x_1, x_2,..., x_N. Each input enters the body of a neuron with a weight, associated with it: w_1, w_2,...,w_3. When the weighted summation happens in the body processor, the neuron puts it through an activation function $\phi(.)$. Thus, we can calculate the output of an artificial neuron with the following expression:

$$Ouput = \phi(\sum_{1}^{N}(w_i * x_i)) \tag{3}$$

In fact, activation does not need to be a binary function, described for simplicity in the previous subsection. A typical choice of activation functions for ANN design is a sigmoid-type function, because it has some particularly attractive properties, such as variable marginal effect and differentiability.

3.3 Network Design

A single neuron is just an elementary processing unit, which has a built-in rule by which to treat the signals. However, just as in biological networks, a single neuron cannot effectively operate on its own. To create a powerful processing system we need to form a neural network: a combination of neurons connected to each other by a set of predefined rules.

A typical network is shown on Figure 6. Each node illustrates a single processing unit, which gathers signals from other units and produces a response with a primitive activation function. Each arrow, connecting the two nodes is associated with some weight, which is a variable parameter of the model. By changing the weights in the network, we can adjust the model throughout a learning algorithm to produce a desired response.

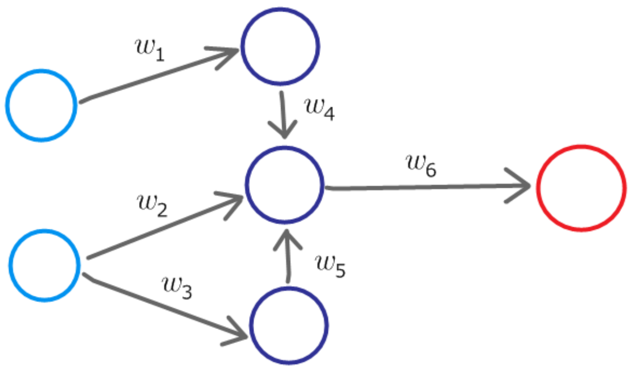


Figure 6: A typical ANN

The main fascination of Neural Networks is that they can produce a desired response simply by training on set of examples. A researcher dealing with a Neural Network does not have to specify all the parameters, since they are found by a network on its own in the process of training.

Mathematically, the problem, solved by any ANN can be treated as mapping from the set of inputs from an n-dimensional space to a set of outputs from a k-dimensional space. Because of this Neural Networks are often called the black boxes[4], which serve as a transfer function from inputs to outputs of the model, but keep the internal processes hidden.

Artificial Neural Networks can look very differently, and it is up to the engineer to specify the organization of nodes and connections as well as the rules by which the network will learn. A typical algorithm for ANN construction would look as follows:

1) Define the topology of a network

2) Define the primitive function of each unit

3) Specify a set of training rules for the network

4) Choose a training and a test sample

5) Feed the Network with training examples and find the parameters

6) Use the parameters to estimate the Network goodness-of-fit on the test sample

Not only the structure on neural networks matter, but the direction in which information flows from one unit of the network to the other. There are 2 major classes of ANNs: feed-forward networks (Figure 7), in which information flows in one direction only, and recurrent (or feed-back) networks (Figure 8), in which signals may loop back and generally go in any direction.

A multi-layer perceptron is a typical example of a feed-forward network (Figure 7), while a Hebbian Network can demonstrate the structure of a simple recurrent network. In this paper we will concentrate on feed-forward networks only.

3.4 Training the network

Neural Networks are so unique because of their ability to learn by example. A learning algorithm is an adaptive method which adjusts the parameters of the Network until it is capable of performing a task with high accuracy.

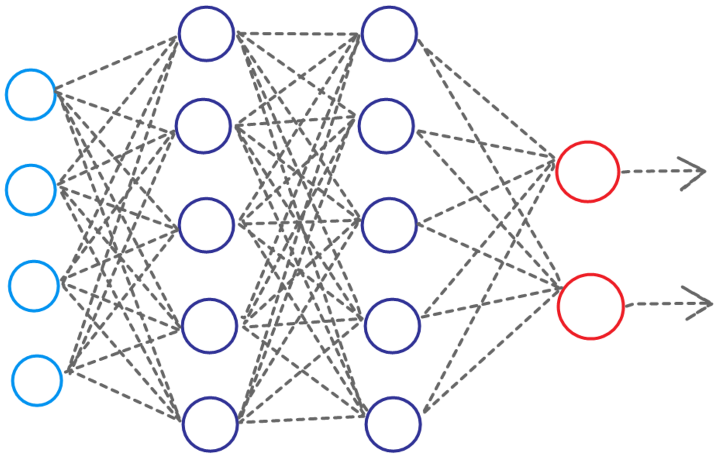


Figure 7: A Feed-Forward Network

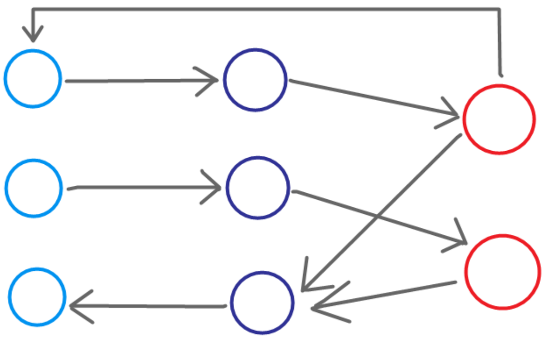


Figure 8: A Recurrent (Feed-back) Network

There are two classes of learning algorithms: supervised learning and unsupervised learning.

In the former, the Network is presented with both input and output values and it learns the rules by which the relationship between input and output can be formed.

In unsupervised learning, only input data is fed to the network, and it learns to identify patterns in this data and classify it by some hidden characteristics. In this paper we would deal with supervised learning only.

Figure 9 shows a typical cycle of a supervised learning algorithm. After the topology of a network is chosen it is fed with a sample of inputs, on which the Network will learn to classify the data. At the beginning of the algorithm, an array of random weights is assigned to the Network, i.e. each connection in the network is defined by chance.

Using the random weights and an input array, the network computes output and compares it to the target value by calculating a squared deviation of output from the target. It is called a Cost of the Network.

If the Cost is very large, output is very far from the target level, so the network is performing its task poorly, and the weights have to be adjusted. The learning algorithm, thus, changes the weights by a small fraction, in a direction of minimizing the error. Then the new weights are fed back into the network and the new output is calculated. The loop goes until the error is small enough to be tolerable.

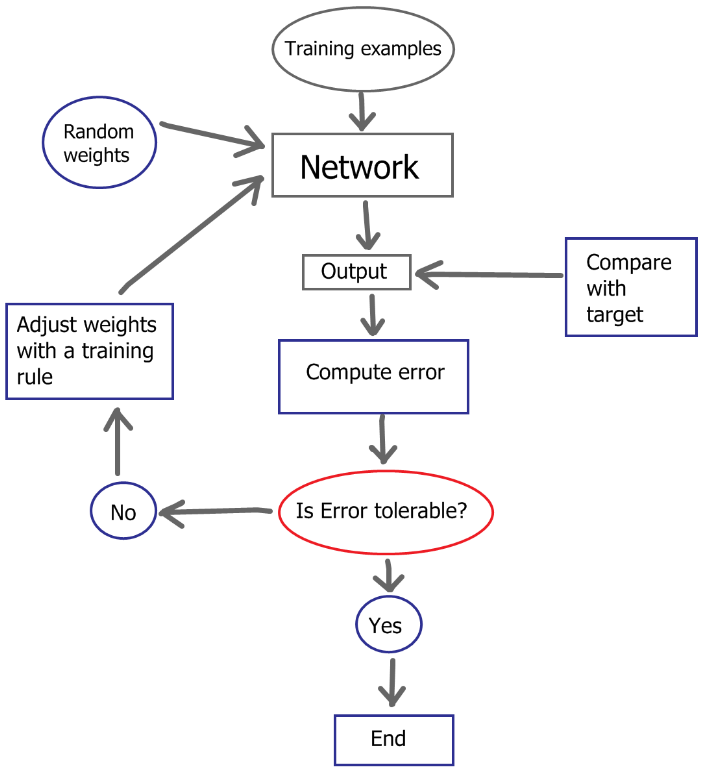


Figure 9: Supervised Learning Algorithm

3.5 The Multi-Layer Perceptron

One of the most popular networks in the theory of ANNs is a multi-layer perceptron (MLP). Because of its simplicity and universality it is often a very good starting point to understanding how ANNs work. It is also a type of a network considered in the empirical part of this paper. Therefore we provide the basic topology of such networks as well as the essential mathematical formulas for training such type of networks.

3.5.1 Topology

A multilayer perceptron (7) is a feed-forward network, which is organized in layers. Each layer consists of a set of units (neurons), each of which is connected to all units in the previous layer as well as all units on the subsequent layer. The layers in the middle of the network, between the input layer and the output layer, are called hidden layers, and the neurons in those layers, are by analogy, hidden units. A network with only one hidden layer is also known as a single-layer perceptron (SLP).

Each node in the MLP Network, except for the inputs, is associated with an activation function, which is computed, once all the inputs are accumulated at the node. A typical activation functions for such networks are: threshold, linear or sigmoid-type function (logistic or hyperbolic tangent). Importantly, the units inside one layer are not connected to each other and perform their basic computation in parallel.

3.5.2 Information processing

To understand the learning algorithm of a multilayer network, consider a 3-layer Network, with 4 input units (x_1,x_2,x_3,x_4) , 3 hidden units ($a_1^{(2)}$,$a_2^{(2)}$, $a_3^{(2)}$) and one output unit ($a_1^{(3)}$). For the sake of simplicity, assume for the first time that we are dealing with only one training example, i.e. we run the network only once to determine one the output of a unique example. Latter on we will extend the example to multiple training examples.

Note that information received at each neuron is not the same as the information passed by it to the subsequent layer, so be careful not to mix the notations. In this example, information **received** at each ith node in layer

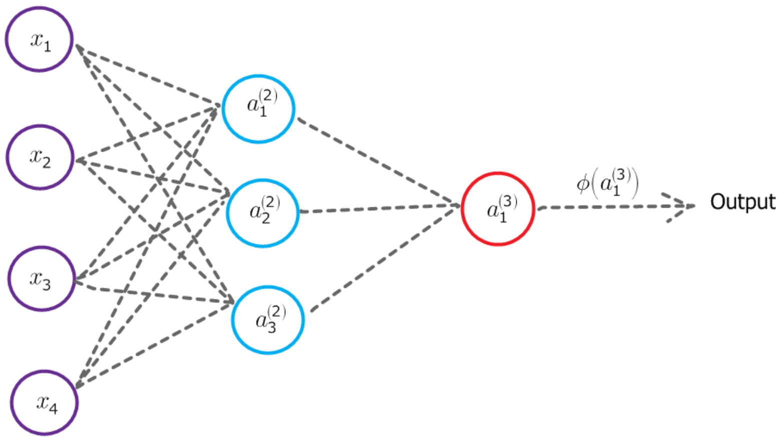


Figure 10: Our Network

l, is defined as $a_i^{(l)}$, while information **passed** by same node, except for the nodes in the input layer, is denoted by $\varphi(a_i^{(l)})$, where $y = \varphi(a)$ is a primitive activation function of the network.

Also, remember that each two connected nodes are associated with a weight, which is an unknown parameter before the network is trained.

Let $w_{ij}^{(l)}$ be a weight parameter, which projects unit i in layer l to unit j in layer $(l+1)$. Then, denote a projection matrix from input layer to hidden layer by w^1. In our example, the dimension of w^1 is 4 by 3, because 4 inputs have to be projected to 3 hidden units, while the dimension of w^2 is 3 by 1, since 3 hidden units are projected to one output unit. Concretely,

$$w^{(1)} = \begin{pmatrix} w_{11}^{(1)} & w_{12}^{(1)} & w_{13}^{(1)} \\ w_{21}^{(1)} & w_{22}^{(1)} & w_{23}^{(1)} \\ w_{31}^{(1)} & w_{32}^{(1)} & w_{33}^{(1)} \\ w_{41}^{(1)} & w_{42}^{(1)} & w_{43}^{(1)} \end{pmatrix} \quad \text{and} \quad w^{(2)} = \begin{pmatrix} w_{11}^{(2)} \\ w_{21}^{(2)} \\ w_{31}^{(2)} \end{pmatrix} \tag{4}$$

Now, when the notations are specified, we start computing the network with the first layer - the input layer. Information, which goes from the input layer is not modified by the activation function, so

$$a^{(1)} = \begin{pmatrix} a^1_{(1)} \\ a^{(1)}_2 \\ a^{(1)}_3 \\ a^{(1)}_4 \end{pmatrix} = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} \tag{5}$$

All inputs project to each of the three hidden units with a corresponding weight. The incoming information at each hidden unit, therefore, accumulates as a weighted sum of all the incoming signals

$$\begin{gathered} a^{(2)}_j = \sum_{i=1}^{4} w^{(1)}_{ij} * a^{(1)}_i \\ \text{or, in vector form} \\ a^{(2)} = (w^{(1)})^T * a^{(1)} \end{gathered} \tag{6}$$

At each node in the hidden layer, information passes through an activation function and goes out of a neuron to the third layer, the output layer. The unit in the output layer gathers information through all units in the hidden layer and therefore receives a total signal of

$$a^{(3)}_1 = \sum_{i=1}^{3} (w^{(2)})^T_{i1} * \varphi(a^{(2)}_i) \tag{7}$$

An output layer produces a response with an activation function:

$$Output = \varphi(a^{(3)}) \tag{8}$$

Carefully plugging expression 6 into 7 and 7 into 8 we get an expression for the response of the output layer:

$$Output = \varphi[(w^{(2)})^T * \varphi((w^{(1)})^T * x)] \tag{9}$$

Then if we are interested in how far the target output (the actual one) is from the one estimated through the network, we can compute the Error $E(w^{(1)}, w^{(2)})$ with the expression:

$$E(w^{(1)}, w^{(2)}) = (Output(w^{(1)}, w^{(2)}) - Target)^2 \tag{10}$$

If we now want to extend calculations to multiple training examples, the Cost of the Network would be computed as an average error for each training example. Thus, the Error function for a sample of size m would be as follows:

$$E_m(w^{(1)}, w^{(2)}) = \frac{1}{2m} * \sum_{i=1}^{m} (Output_m(w) - Target_m)^2 \tag{11}$$

Note that adding "2" in the denominator of the error function in no more than a convenient scaling of the expression. Later on, we would be dealing with the derivatives of the Error function, so "2s" would cross out and lead a neater derivative expression. Such update is just a tradition in neural networks computing, but is definitely not a rule, since it does not change anything except for the scale.

3.5.3 Training

Now that we know how information is processed throughout the Network, it is useful to understand the logics behind training the network.

As already mentioned, a supervised learning algorithm is aimed at minimizing the cost of the ANN, i.e. the mean-squared error of computation. Therefore, any learning algorithm, dealing with cost minimization would actually serve as training technique for an ANN.

However, Neural Networks most often have a large set of parameters, which make a problem of function minimization a tough one. Even in our example with only 3 layers and a small number of neurons, there are 15 weights we would have to find with the learning algorithm. Dealing with such problems is technically hard, time consuming and simply daunting for a researcher. Therefore, there need to be powerful tool, which can deal with such kind of problems as multi-variable function minimization.

There wasn't much progress in implementation of Artificial Neural Networks before the discovery of the back-propagation algorithm [35]. It proves to be a rather simple method of computing feed-forward networks, which is used in combination with any of the optimization methods known up to date.

In recent years there have been numerous extensions proposed to classical back-propagation algorithm, which try to deal with its imperfections. Such extensions normally propose new and more reliable methods of error minimization. In this section we consider 2 optimization methods, which are applicable to back-propagation method. The first is the widely known gradient descent. The other one is an alternative method, called the Newton-type optimization, which we implement later in the empirical part of the paper.

Consider the same network topology we used in the previous subsection. To simplify notation, again consider only one training example. Then, the cost function of the network is summarized by 10.

The first step to finding a critical point of the function is to compute its gradient, which is an array of partial derivatives of the function with respect to each of its parameters. In out example, the gradient of the Error function consists of 15 partial derivatives computed for each weight in the network.

$$\nabla E = (\frac{\partial E}{\partial w_{11}^{(1)}}, \frac{\partial E}{\partial w_{12}^{(1)}}, \frac{\partial E}{\partial w_{13}^{(1)}}, ..., \frac{\partial E}{\partial w_{31}^{(2)}}) \tag{12}$$

Since 10 is a composite function of weights, we have to use the chain rule to calculate the partial derivatives for each weight. Concretely, each element of the gradient vector is split into several partial derivatives.

For example, a derivative with respect to a weight connecting neuron j in hidden layer of our network to the unique neuron in output layer, would be calculated in 3 steps:

$$\frac{\partial E}{\partial w_{j1}^{(2)}} = \frac{\partial E}{\partial Output} * \frac{\partial Output}{\partial(\varphi(a_1^{(3)}))} * \frac{\partial(\varphi(a_1^{(3)}))}{\partial w_{j1}^{(2)}} \tag{13}$$

Computing weights, which connect the input layer and the hidden layer is a bit more complex, but the logics remains the same.

Generalizing, for the each weight connecting layer l to layer $(l + 1)$, we need to compute $2 * (L - l) + 1$ partial derivatives in order to apply the chain rule, where L is the total number of layers in the network.

Once we have computed the derivatives, which are no more than the rate of change of the function with respect to each of the parameters, we can use them to form a learning rule, which would minimize the error function.

In Gradient Descent method, each weight is adjusted proportionately to the negative derivative of error with respect to this weight, i.e.

$$\triangle w_{ij}^{(l)} = -\lambda * \frac{\partial E}{\partial w_{ij}^{(l)}} \tag{14}$$

The Newton-type minimization has a rather modified version of weight adjustment:

$$\triangle w_{ij}^{(l)} = -\delta * \frac{\partial (Error)}{\partial w_{ij}^{(l)}} / \frac{\partial^2 Error}{\partial {w_{ij}^{(l)}}^2} \tag{15}$$

In both methods, λ and δ are learning rates, which are normally predefined by a researcher. The algorithm runs until the error becomes small enough to be tolerable.

4 Data

4.1 Time Series Description

In the present paper we consider five Russian stocks, which are among the most liquid [2] assets, listed on the Moscow Interbank Currency Exchange (MICEX). The five chosen companies represent the major sectors of the Russian economy and therefore contribute a significantly to the Nation's income.

The three internationally recognized corporations specialize in resource extraction: Gazprom (gaz), Lukoil (oil) and MMC Norilsk Nickel (nickel and palladium). The other two companies are Mobile TeleSystems (MTS), which is one of the leading telecommunication agent, and Sberbank Rossii - the largest bank in Russia and Eastern Europe. All five companies were on the list [3]of 100 most capitalized public companies in 2014 in Russia.

Table 1 includes the basic descriptive statistics of the 5 time series, taken from 1st January 2011 until 31st December 2013. Plots of tte time series, histograms and QQ-plots are available in the Appendix A.

The analysis of the histograms and the QQ plots, as well as the results of the Jarque-Berra tests indicate that none of the five time series is normal. The fluctuations of the time series (graphs) suggest that they are non-stationary.

[2] `http://moex.com`

[3] `http://riarating.ru/corporate_sector_rankings/`

Table 1: Descriptive Statistics and Normality Test

Stock	GAZPROM	LUKOIL	MTS	SBERBANK	GMKN
Min	107.2	1537	169.5	62.65	4105
Median	156.2	1894	241.8	95.44	5269
Mean	163.0	1877	247.0	94.48	5649
Max	244.0	2136	351.5	110.70	7818
St.Dev	29.5059	129.488	36.1867	8.67248	990.6903
Skewness	0.50728	-0.30526	0.66326	-0.66729	0.7843
Kurtosis	2.56156	2.22768	3.362481	3.13559	2.22347
Jarque-Bera	38.3261	30.409	59.3318	56.4596	96.1176
Probability	4.76e-09	2.493e-07	1.307e-13	5.494e-13	2.2e-16

4.2 Training and Test Samples

A total period of 3 years is considered in this paper, starting from 1st January 2011 until 31st December 2013.

We use a rolling window approach to yield 12 test samples, each one month long. The procedure of fitting and testing the model is as follows. We train the network on 2 years of data and then use the estimated parameters of the Network to forecast one subsequent month after of the training period. The number of observations in the test sample, thus, depend on the number of trading days in a corresponding month.

We repeat the procedure 12 times, each time shifting the training and the test periods one month ahead. For example, we start with the training period from 1st January 2011 to 31st December 2012, and test the model on the data from 1st January to 31 January. In the next trial, the training period would be from 1st February 2011 until 31 January 2013, while the testing period would be from 1st February 2013 until 28th February 2013. Generally, we provide the forecast for the whole year 2013.

5 Model

5.1 Training Algorithm and Software

Using R software environment [4], we developed a set of functions to compute a feed-forward multilayer network with flexible parameters, in which the amount of layers and neurons can be freely chosen, depending on the context of the problem (Appendix B).

The initial version of the program was designed for fitting univariate models, with lagged values as inputs. However, the program can be easily extended to multivariate time series as well. The program works the following way:

Step 1: A researcher sets the topology of a network: number of inputs, outputs, hidden layers and neurons in each hidden layer.

Step 2: A time series is introduced to the model. Normally about 80% of the data is used to train the network (steps 3-6), while the remaining data is used for verifying goodness-of-fit of the network (step 7).

Step 3: The program sets an array of random weights according to the dimension of the network. The output value is forward-propagated using the array of random weights.

Step 4: Output of the network (step 3) is compared to the target value (real data output) and the mean squared error (MSE) is calculated as a function of network weights:

$$MSE = \sum_{1}^{N} (Output_i - Target_i)^2 \tag{16}$$

Step 5: An algorithm optimizes the weights of the network, decreasing MSE on each subsequent iteration of the algorithm. Minimization problem is solved by a Newton-type optimization algorithm. A maximum of number of iterations can be set exogenously through the function. After exceeding the limit of iterations, the algorithm stops to determine the final array of weights, which are considered the optimal weights.

Step 6: Optimal weights are used to fit the expected output for the

[4]`http://www.r-project.org/`

training set, then both real values of the time series and the estimated values are visualized on the graph for comparison.

Step 7: Verification data (remaining 20%) is used to determine the goodness of fit of the network, out-of-sample.

Note that the activation function of the network is a sigmoid, which is defined on the interval (0,1). Therefore, initial time series has to be adjusted before it is introduced to the network. In this paper we adjusted the data by first normalizing the data set and then putting it through the sigmoid transformation:

$$\widetilde{x} = \frac{1}{1 + e^{-(\frac{x-\mu_x}{\sigma_x})}} \tag{17}$$

For further comments and coding instructions, consult the Appendix B.

5.2 Optimal Model Configuration

Choosing a network dimension is one of the most challenging parts of ANN modelling. Some empirical rules have been previously proposed([20],[38],[26]) to determine the optimal number of hidden layers and neurons in the network.

In this paper we avoid empirical rules available in literature on Neural Networks and instead use a standard approach of sorting out models based on their performance. We called this approach "The Network Grid Search".

We fit several networks with different configurations, each time either increasing the dimension of the network, or the number of neurons in the layers.

We compare the models based on 3 criteria: mean squared error of the training sample, which is a standard measure of goodness-of-fit, mean-squared error of the test sample, which is an indicator of forecasting ability of the network, and the running time of the algorithm - one of the most daunting part of network fitting.

To avoid haphazard results, each configuration was run 10 times on the same stock and the same estimation period and then the average MSE and running time were stated for each network configuration. Then we chose a suitable model based on the optimal cost and benefit combination.

Table 2 presents an extract from our grid search analysis, showing performance of several combinations of networks run on MTS stock. The number of digits in brackets indicate the number of layers of the network, while each digit shows the amount of neurons in a corresponding layer. For example, a network indicated by (3,2,1) is a three-layer network with 3 input units, 2 hidden units and one output unit.

The sample estimation period runs from 1st January 2011 until 31st December 2012 and the forecast period is from 1st January 2013 to 31st January 2013. In-MSE is the mean squared error computed on the estimation period, while out-MSE is the mean squared error for out-of-sample forecast. The number of parameters indicate the dimension of the weight vector, which is estimated to fit the network. Running time estimates the amount of time in seconds or minutes, needed for the CPU to handle the Error Minimization problem and determining the parameters of the network.

Table 2: Network Configuration Grid Search

Configuration	Parameters	In MSE	Out MSE	Running Time
(3.1.1)	4	278.635	258.3314	12.02656 sec
(3.2.1)	8	12.3410	5.8673	52.7406 sec
(4.3.1)	15	12.34661	5.73758	1.54732 min
(5.3.1)	18	12.28980	5.95265	1.84865 min
(4.4.1)	20	12.24035	5.64919	2.01464 min
(3.6.1)	24	12.15218	5.58951	2.45013 min
(4.2.2.1)	14	12.26358	6.19697	1.87956 min
(3.3.3.1)	21	12.21255	5.81322	2.82013 min
(5.5.5.1)	55	12.0738	5.6294	7.1624 min

The purpose of the grid search was to determine an optimal model, which would provide a forecast with a good fit (low In-MSE) and forecasting power (low Out-MSE) and at the same time would not overload the CPU. In other words, the desire to optimize the MSE by increasing the number of layers and units may only seem reasonable as long as we can tolerate the running time of the algorithm.

We were interested in network configurations with no less than 3 regressors (long memory) and at least one hidden layer. Generally, the grid search for our data (Table 2) shows that any configuration with two or more hidden units does a good job in fitting and forecasting the data. A network with just one hidden unit stands out significantly and provides evidence that a single neuron is not capable of efficient problem solving in this context.

Increasing the number of parameters, while keeping the number of layers fixed, does not necessarily increase efficiency, while the estimation time, surely, rises. Increasing the number of layers does not provide evidence of better fit, either, and, moreover, prolongs estimation. For example, compare (4.2.2.1) with 14 parameters to estimate and (4.3.1) with 15 parameters. Despite the decrease in the number of unknown parameters, the estimation time has increased, which shows that the algorithm actually deals better with fewer layers.

Following the results of the grid search, we found it reasonable to avoid undue complexity and optimize the running time of the algorithm without, actually harming the results.

A network considered in this paper is a three layer feed-forward network with 4 input units and a single hidden layer with 3 units in it (Figure10) A total number of parameters to be estimated is 15: 12 weighs for mapping from input layer to hidden layer and 3 weights to associate the hidden layer with the output layer.

5.3 Performance Measures

To evaluate performance of our Network we propose several goodness-of-fit indicators, which are also found widely in literature on financial forecasting [23],[2].

To begin with, we examine out-of sample performance by mean-squared-error (MSE) measure, which is one of the most widely used verification method in model fitting. For a network with training period from from 1 to N and a test period from N+1 to N+k, MSE would be calculated as:

$$MSE = \sum_{N+1}^{N+k} (Forecast_i - Price_i)^2 \tag{18}$$

However, calculating MSE might not give a good overview of how effective the forecast is.

In financial forecasting it is often very useful to correctly identify the direction of the forecast rather than the concrete value. Therefore, we propose a binary function, which we called the Hit Sequence, which can take either of the 2 values: 1, if the forecast guesses the direction of the time series correctly, 0 otherwise:

$$Hit_i = \begin{cases} 1, & Forecast_i * Price_i \geq 0 \\ 0, & otherwise \end{cases} \tag{19}$$

Then, we find a proportion of correct guesses of the forecast direction and call this measure the Hit Ratio. For a forecast window of length k, the Hit Ratio would be

$$HitRatio = \sum_{i=1}^{k} (Hit_i)/k \tag{20}$$

Finally, our work would have been useless if we do not consider practical implication of our forecasts. Therefore, we introduce a trading strategy and evaluate Profit-and-Loss account for a hypothetical trader, who pursues this strategy.

The details of trading are as follows. A trader enters the market on 1st January 2013 with hypothetical money of 100. At the beginning of each month, he builds a network based on a learning sample of 2 years of data, and then uses the parameters of the network to evaluate one month ahead forecast. Then, using the forecast values he uses the following strategy for each trading day in the current month: he goes long when the model forecasts the price to rise and to goes short otherwise. At the end of each month, cumulative profit (of loss) is calculated as a percentage of the starting money.

Trading performance is evaluated for each month in 2013, independently for each of the 5 assets considered in this paper.

5.4 Comparing Forecasts

The second hypothesis of our paper deals with comparing the models with different iterations in the learning algorithm. Denote MSE_{100} the mean-squared error of the model with 100 iterations in the learning algorithm, and MSE_{100} the MSE of the model with 200 iterations.

We use the standard Diebold-Mariano test [10] for forecast comparison. The Error of each forecast is denoted by mean-squared error. Then the null hypothesis tested is

$$\begin{gathered} H_o : MSE_{100} = MSE_{200} \\ \text{againnst the alternative} \\ H_1 : MSE_{100} \neq MSE_{200} \end{gathered} \tag{21}$$

We denote d the difference between the MSE of the first model and MSE of the second model, i.e.

$$d = MSE_{100} - MSE_{200} \tag{22}$$

Diebold-Mariano statistics is calculated as

$$\begin{gathered} DM = \frac{\overline{d} * \sqrt{n}}{s_d}, \\ \text{where} \quad \overline{d} = \frac{1}{n}\sum_{i=1}^{n} d_i \quad \text{and} \\ s_d = \sqrt{\frac{1}{n-1} * \sum_{i=1}^{n} (d_i - \overline{d})^2} \end{gathered} \tag{23}$$

Then the result of the test is compared to student t-statistics with $n - 1$ degrees of freedom at a predefined level of significance.

In our test, we had MSE estimates for each of the 12 months of 2013, thus the sample consists of only 12 observations.

6 Results

We first consider the Network trained with 100 iterations of the Error function adjustments. Figures 16,17 in The Appendix A show the results of training such a network on the 5 stocks.

Clearly, the in-sample fit appears to be outstandingly accurate (red dotted line as opposed to black line of the real time series). The graphs 17b, 16b show almost one-to-one relationship between the forecast and the stock price. However, such seemingly amazing results do not grantee that we can reproduce the direction of the time series with great certainly using the forecast. A relatively low MSE(Figure3) is attained for all stocks out-of-sample, but may be a rather confusing sign of accuracy. While being a good indicator in theory, it does not grant practical success.

In fact, if we build a graph, showing the relationship between the change of time series $x_t - x_{t-1}$ and the change in forecast value $f_t - f_{t-1}$, we see that there is almost no pattern in the data. Observations tend to fill the graph randomly, and the best-fitting line has a slope close to zero and an R-squared of below 1 %. Such tendency is shown for each of the 5 considered stocks. What this means for the data analysis, is that we cannot confidently predict the time path of the time series, judging by the time path of the forecast.

The relatively poor forecasting power of the ANN is supported by the results of Hit ratio (Figure 4) and the P&L Account of a trader (Figure 5). The average Hit Ratio throughout the year 2013 appears to be close to 0.5 for all five time series, which means that the network correctly predicts the direction of the time series almost randomly. This is equivalent to forecasting with tossing a fair coin, which would definitely be a quicker and simpler way to attain the same accuracy of results.

The P&L Account measure is somewhat more complicated to understand at first sight, since there are both positive and negative profit values throughout stocks and time periods. In each forecast window, which lasts for about 20 trading days (one month), the forecast correctly predicts the stock price direction for about half of the days, in which the P&L is increased. The remaining half of the month leads to a loss in the P&L Account. The cumulative value earned by the end of each month is determined primarily by the magnitude of price change in each of the 2 periods: when the forecast predicts correctly - good days, and when it does not - bad days. If the price fluctuations on the good days is higher than those of the bad days, then the trader is expected to enjoy his gain, otherwise, the bad days would play havoc.

After yielding such unsatisfactory results with our initial model, we tried to investigate the possible ways to increase the forecasting ability of the network. Going back to the theory of ANNs, it may be a good idea to review the learning algorithm of the network. In most cases, minimizing the error of the cost function may lead to an infinite loop, since the dimension of the parameter vector is very large for the algorithm to easily cope with error minimization. Therefore, an ANN engineer should always specify the maximum number of iterations , which the algorithm runs in the error-minimization problem. Clearly, the higher the number of iterations, the more precise would be the in-sample network fit.

Our idea at this step was to analyse the change of forecasting power of the ANN with the increase in the number of iterations of the learning algorithm. In other words, would a better training give us neater forecasts?

The forecasting power was estimated as a coefficient of determination in the regression, in which delta time series $x_t - x_{t-1}$ is regressed on delta forecast values $f_t - x_{f-1}$. For each stock we associated the number of learning steps with the R-squared (figure 18 in Appendix A). For all 5 stock we clearly see an increase in forecasting power with the increase in training iterations (even though R-squared still remains very low).

Nevertheless, such idea forced us to evaluate the same performance measures as before, but for a network with a higher number of iterations. We repeated the calculations for 200 iterations of the learning algorithm and computed the MSE of each forecasting window [5]. Then we apply Diebold-Mariano test to compare the results with those gathered with the initial set-up.

Table 6 shows the results of the tests. Clearly, the test does not reject the hypothesis that the forecasts are the same. So, under this framework, changing the quality of the training is not a remedy for a poor forecasting

[5]The results for P&L, Hit Ratio and MSE are available on demand

model.

7 Conslusion

In this paper we considered performance of an MLP Neural Network on forecasting stock time series of the 5 Russian companies. It is clear that a network of this kind cannot be an ideal tool for practical problems, such as constructing a trading strategy. However, we still see the undeniable ability of such networks to fit the data with great precision, which, I am sure, can be exploited with a wiser and deeper analysis of the problem. The possible directions of the study of financial forecasting through ANNs might be different from merely applying the conventional networks such as the MLP. The basic feed-forward network can be extended and modified to serve the researcher's needs.

Another direction to look at is a careful examination of output data received from the network as well as analysing the patterns in time series before applying them to neural networks. It might also be the case that a less primitive trading strategy is available with the use of ANNs, which may yield a much more valuable result.

References

[1] Adya, M., & Collopy, F. (1998). How effective are neural networks at forecasting and prediction? A review and evaluation. J. Forecasting, 17, 481-495.

[2] de A Araújo, R. (2012). A morphological perceptron with gradient-based learning for Brazilian stock market forecasting. Neural Networks, 28, 61-81.

[3] Balkin, S. D., & Ord, J. K. (2000). Automatic neural network modeling for univariate time series. International Journal of Forecasting, 16(4), 509-515.

[4] Benítez, J. M., Castro, J. L., & Requena, I. (1997). Are artificial neural networks black boxes?. Neural Networks, IEEE Transactions on, 8(5), 1156-1164.

[5] Chakraborty, K., Mehrotra, K., Mohan, C. K., & Ranka, S. (1992). Forecasting the behavior of multivariate time series using neural networks. Neural networks, 5(6), 961-970.

[6] Ghiassi, M., Saidane, H., & Zimbra, D. K. (2005). A dynamic artificial neural network model for forecasting time series events. International Journal of Forecasting, 21(2), 341-362.

[7] De Gooijer, J. G., & Hyndman, R. J. (2006). 25 years of time series forecasting. International journal of forecasting, 22(3), 443-473.

[8] de Groot, C., & Würtz, D. (1991). Analysis of univariate time series with connectionist nets: A case study of two classical examples. Neurocomputing, 3(4), 177-192.

[9] Dennis, J. E. and Schnabel, R. B. (1983) Numerical Methods for Unconstrained Optimization and Nonlinear Equations. Prentice-Hall, Englewood Cliffs, NJ.

[10] Diebold, F. X., & Mariano, R. S. (2002). Comparing predictive accuracy. Journal of Business & economic statistics, 20(1).

[11] Dutta, S., Shekhar, S., & Wong, W. Y. (1994). Decision support in non-conservative domains: generalization with neural networks. Decision Support Systems, 11(5), 527-544.

[12] Gorr, W. L., Nagin, D., & Szczypula, J. (1994). Comparative study of artificial neural network and statistical models for predicting student grade point averages.International Journal of Forecasting, 10(1), 17-34.

[13] Hardgrave, B. C., Wilson, R. L., & Walstrom, K. A. (1994). Predicting graduate student success: a comparison of neural networks and traditional techniques. Computers & Operations Research, 21(3), 249-263.

[14] Hebb, D. O. (2005). The organization of behaviour: A neuropsychological theory. Psychology Press.

[15] Haykin, S. (1992). Neural Networks: A Comprehensive Foundation, 2nd Edition, Prentice-Hall.

[16] Hill, T., Marquez, L., O'Connor, M., & Remus, W. (1994). Artificial neural network models for forecasting and decision making. International Journal of Forecasting, 10(1), 5-15.

[17] Hippert, H. S., Bunn, D. W., & Souza, R. C. (2005). Large neural networks for electricity load forecasting: Are they overfitted?. International Journal of forecasting, 21(3), 425-434.

[18] Hornik, K., Stinchcombe, M., & White, H. (1989). Multilayer feedforward networks are universal approximators. Neural networks, 2(5), 359-366.

[19] Hruschka, H. (1993). Determining market response functions by neural network modeling: A comparison to econometric techniques. European Journal of Operational Research, 66(1), 27-35.

[20] Kang, S. Y. (1992). An investigation of the use of feedforward neural networks for forecasting.

[21] Khashei, M., & Bijari, M. (2010). An artificial neural network (p, d, q) model for time series forecasting. Expert Systems with Applications,37(1), 479-489.

[22] Krycha, K. A., & Wagner, U. (1999). Applications of artificial neural networks in management science: a survey. Journal of Retailing and Consumer Services, 6(4), 185-203.

[23] Kumar, M. (2009). Nonlinear Prediction of the Standard & Poor's 500 and the Hang Seng Index Under a Dynamic Increasing Sample. Asian Academy of Management Journal of Accounting and Finance, 5(2), 101-118.

[24] Lapedes, A., & Farber, R. (1987). Nonlinear signal processing using neural networks.

[25] Lee, K., Booth, D., & Alam, P. (2005). A comparison of supervised and unsupervised neural networks in predicting bankruptcy of Korean firms. Expert Systems with Applications, 29(1), 1-16.

[26] Lippmann, R. P. (1987). An inuction to computing with neural nets. ASSP Magazine, IEEE, 4(2), 4-22.trod

[27] MacKay, D. J. (2003). Information theory, inference, and learning algorithms (Vol. 7). Cambridge: Cambridge university press.

[28] McNelis, P. D. (2005). Neural networks in finance: gaining predictive edge in the market. Elsevier Acad. Press.

[29] Webb, A. R., & Lowe, D. (1990). The optimised internal representation of multilayer classifier networks performs nonlinear discriminant analysis. Neural Networks, 3(4), 367-375.

[30] McCulloch, W. S., & Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. The bulletin of mathematical biophysics, 5(4), 115-133.

[31] Paliwal, M., & Kumar, U. A. (2009). A study of academic performance of business school graduates using neural network and statistical techniques. Expert Systems with Applications, 36(4), 7865-7872.

[32] Paliwal, M., & Kumar, U. A. (2009). Neural networks and statistical techniques: A review of applications. Expert systems with applications, 36(1), 2-17.

[33] Qi, M. (2001). Predicting US recessions with leading indicators via neural network models. International Journal of Forecasting, 17(3), 383-401.

[34] Rojas, R. (1996). Neural networks: a systematic introduction. Springer.

[35] Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1988). Learning representations by back-propagating errors. Cognitive modelling.

[36] Sharma, S., & Shebalkov, M. (2013). Application of Neural Network and Simulation Modeling to Evaluate Russian Banks' Performance. Journal of Applied Finance & Banking, 3(5), 19-37.

[37] Tam, K. Y., & Kiang, M. Y. (1992). Managerial applications of neural networks: the case of bank failure predictions. Management science, 38(7), 926-947.

[38] Tang, Z., & Fishwick, P. A. (1993). Feedforward neural nets as models for time series forecasting. ORSA Journal on Computing, 5(4), 374-385.

[39] Werbos, P. J. (1990). Backpropagation through time: what it does and how to do it. Proceedings of the IEEE, 78(10), 1550-1560.

[40] West, P. M., Brockett, P. L., & Golden, L. L. (1997). A comparative analysis of neural networks and statistical methods for predicting consumer choice. Marketing Science, 16(4), 370-391.

[41] Wong, B. K., & Selvi, Y. (1998). Neural network applications in finance: a review and analysis of literature (1990–1996). Information & Management, 34(3), 129-139.

[42] Zhang, G., Eddy Patuwo, B., & Y Hu, M. (1998). Forecasting with artificial neural networks:: The state of the art. International journal of forecasting, 14(1), 35-62.

A Supplementary Tables, Graphs and Figures

Table 3: Normalized Out-of-Sample MSE

Month	GAZPROM	LUKOIL	MTS	SBERBANK	GMKN
January	0.0038	0.0184	0.0142	0.0473	0.0050
February	0.0046	0.0071	0.0136	0.0210	0.0066
March	0.0059	0.0183	0.0544	0.0277	0.0049
April	0.0060	0.0332	0.0331	0.0250	0.0095
May	0.0122	0.0506	0.0338	0.0549	0.0211
June	0.0333	0.0497	0.0266	0.0425	0.0117
July	0.0081	0.0229	0.0194	0.0337	0.0075
August	0.0058	0.0271	0.0336	0.0154	0.0138
September	0.0147	0.0319	0.0325	0.0329	0.0095
October	0.0090	0.0319	0.4248	0.0192	0.0099
November	0.0167	0.0174	0.0145	0.0292	0.0070
December	0.0061	0.0121	0.0144	0.0181	0.0150
Average	0.0105	0.0267	0.0596	0.0306	0.0101

Table 4: Hit Ratio

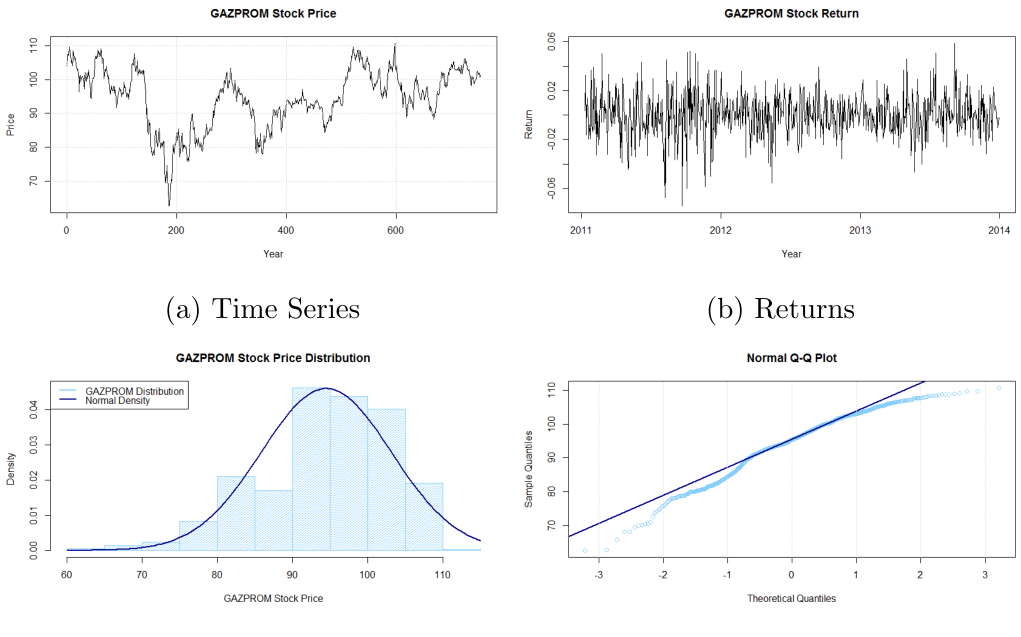
Month	GAZPROM	LUKOIL	MTS	SBERBANK	GMKN
January	0.471	0.471	0.471	0.529	0.294
February	0.526	0.474	0.474	0.474	0.368
March	0.684	0.684	0.474	0.500	0.632
April	0.619	0.524	0.571	0.524	0.476
May	0.700	0.700	0.400	0.600	0.450
June	0.556	0.556	0.500	0.556	0.667
July	0.773	0.636	0.500	0.636	0.545
August	0.429	0.476	0.667	0.810	0.714
September	0.550	0.450	0.650	0.600	0.750
October	0.409	0.545	0.409	0.364	0.545
November	0.474	0.474	0.474	0.474	0.368
December	0.800	0.650	0.579	0.600	0.600
Average	0.582	0.553	0.514	0.555	0.534

Table 5: Profit and Loss Account

Month	GAZPROM	LUKOIL	MTS	SBERBANK	GMKN
January	-1.09%	-2.64%	2.59%	12.24%	-5.14%
February	-1.37%	-1.52%	1.39%	-0.72%	-6.50%
March	8.02%	6.52%	5.39%	1.08%	5.03%
April	9.59%	4.62%	2.78%	7.51%	-3.93%
May	5.44%	1.62%	-5.45%	-1.53%	2.37%
June	2.79%	5.15%	-5.93%	4.79%	13.18%
July	17.21%	1.14%	-3.99%	-0.67%	-1.18%
August	-3.23%	-2.20%	-0.19%	10.63%	9.12%
September	7.23%	-8.94%	5.06%	5.42%	9.13%
October	-2.36%	1.93%	-6.40%	-7.30%	6.96%
November	-4.17%	-5.17%	-2.68%	-6.18%	-1.40%
December	7.23%	6.94%	3.15%	1.02%	5.82%
Year’s PNL	52.74%	6.30%	-5.20%	27.18%	36.17%

Table 6: Diebold-Mariano Test Results

Stock	GAZPROM	LUKOIL	MTS	SBERBANK	GMKN
DM-stat	-1.042935	1.02694	1.29907	-1.01985	-1.08548
p-value	0.31936	0.32651	0.2991	0.31737	0.27268



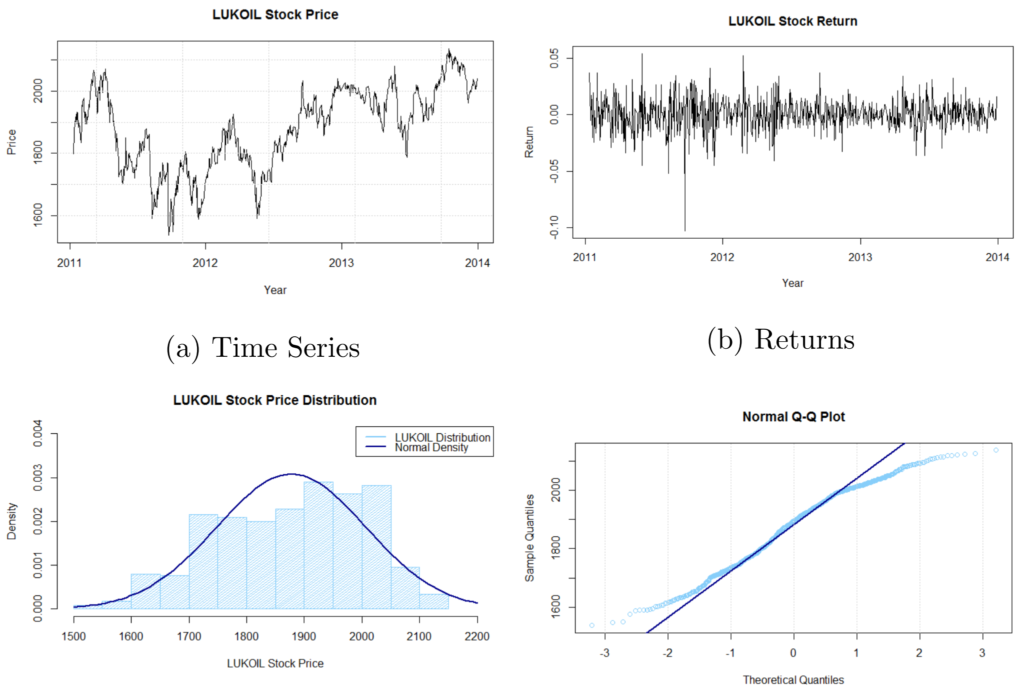
(a) Time Series

(b) Returns

(c) Histogram

(d) QQ plot

Figure 11: GAZPROM Descriptive Statistics



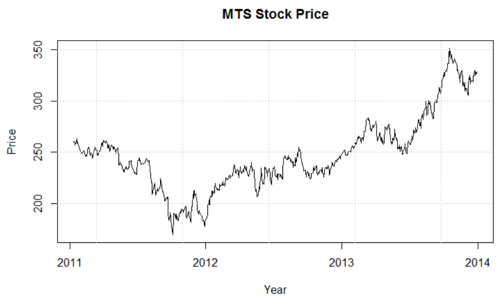
(a) Time Series

(b) Returns

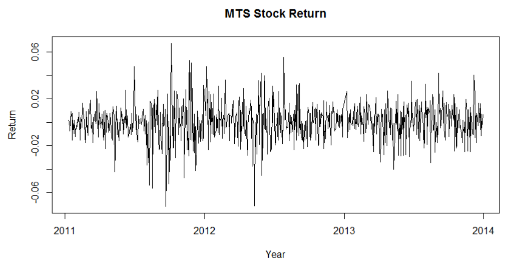
(c) Histogram

(d) QQ plot

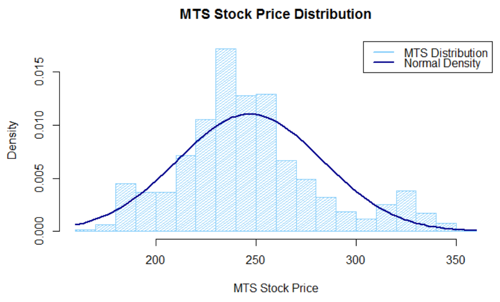
Figure 12: LUKOIL Descriptive Statistics



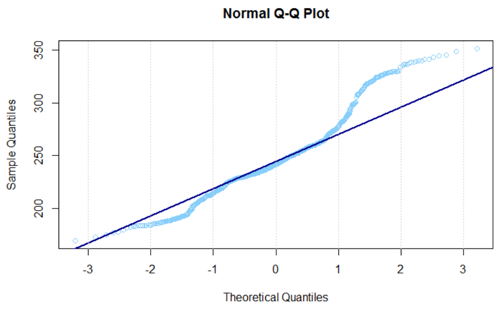
(a) Time Series



(b) Returns

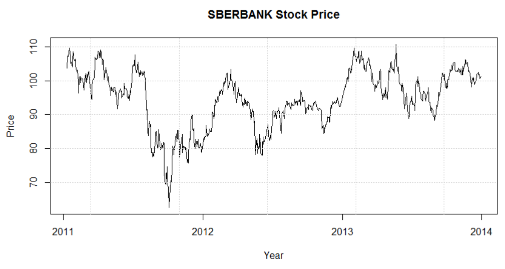


(c) Histogram

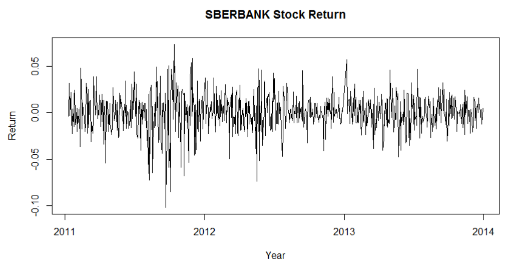


(d) QQ plot

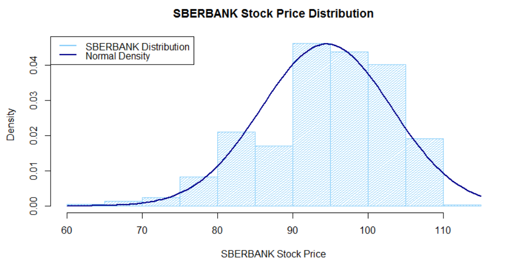
Figure 13: MTS Descriptive Statistics



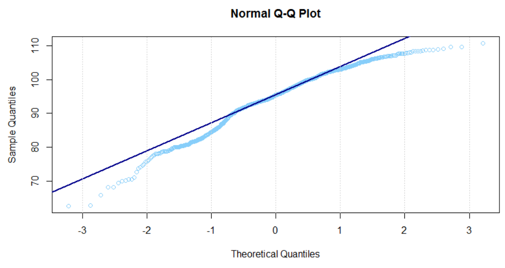
(a) Time Series



(b) Returns

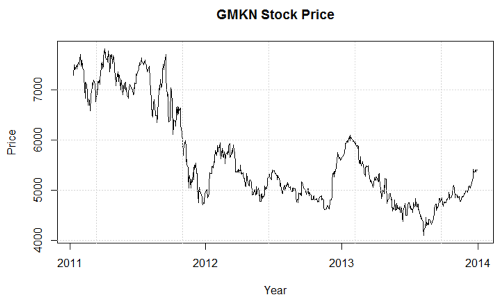


(c) Histogram

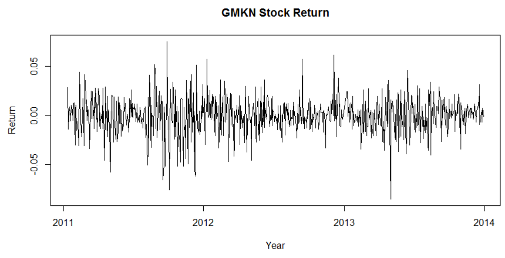


(d) QQ plot

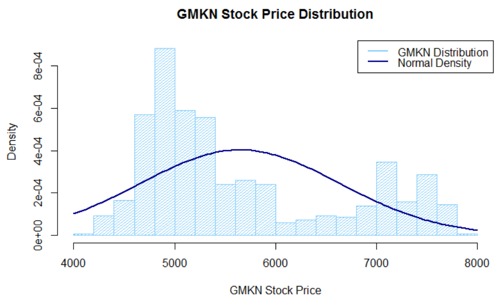
Figure 14: SBERBANK Descriptive Statistics



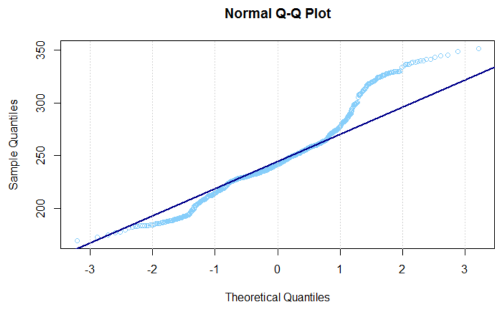
(a) Time Series



(b) Returns

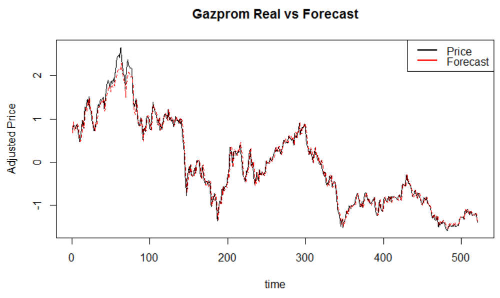


(c) Histogram

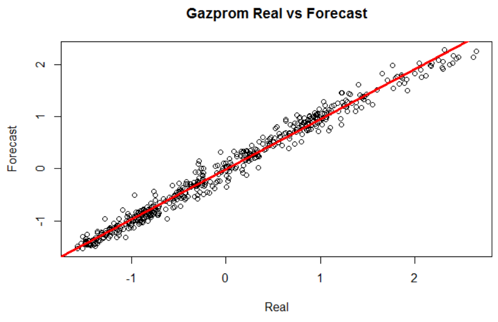


(d) QQ plot

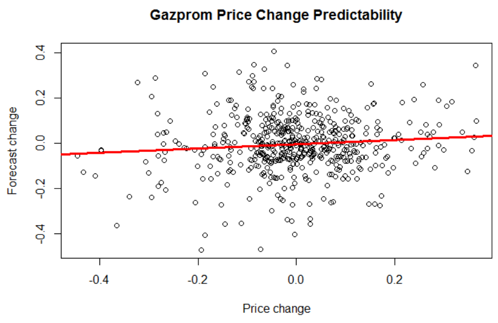
Figure 15: GMKN Descriptive Statistics



(a) Real Price vs Forecast

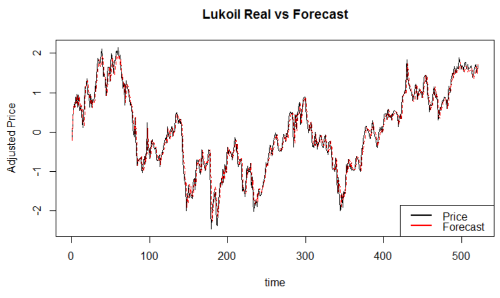


(b) Goodness of Fit

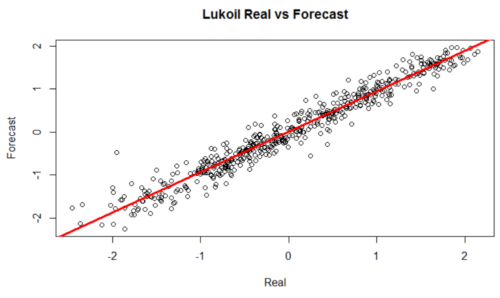


(c) Price Change Predictability

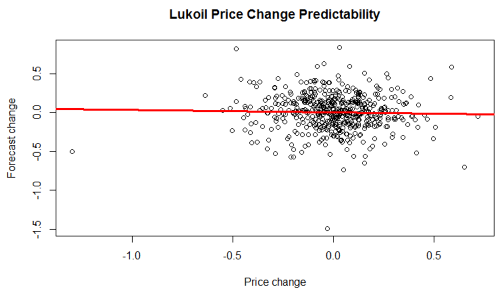
Figure 16: Gazprom Network Fit and Predictability



(a) Real Price vs Forecast

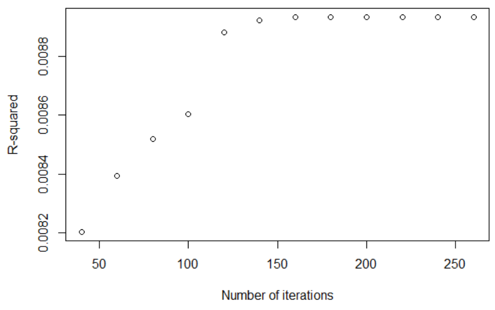


(b) Goodness of Fit

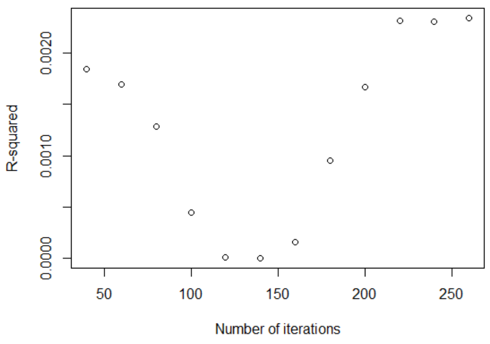


(c) Histogram

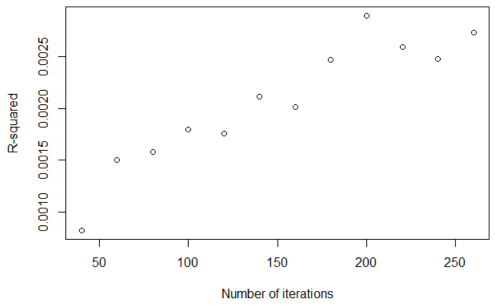
Figure 17: Lukoil Network Fit and Predictability



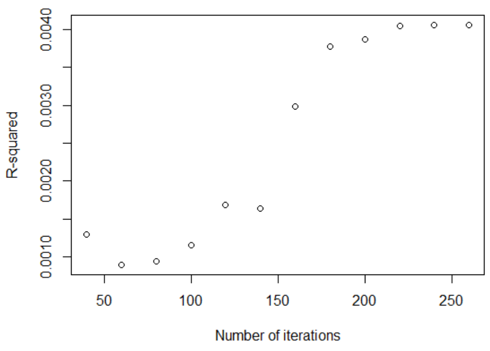
(a) Gazprom



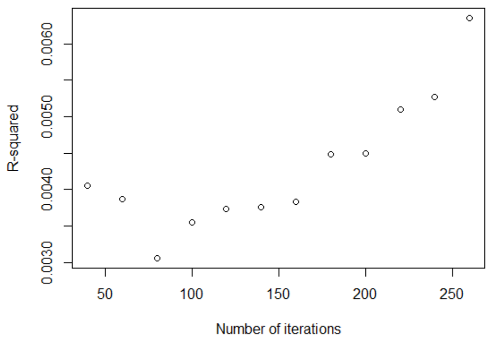
(b) Lukoil



(c) MTS



(d) Sberbank



(e) GMKN

Figure 18: Explanatory Power of the Forecast as a Function of the Number of Iterations

B Tips for Programmers

```r
### Packages: "xts", "pracma"


#################### Preliminary steps #######################


### A useful function, which transforms a vector to a matrix

vec2mat <- function(w,row,col){ matrix(w,row,col,byrow=T)}


### Error function for 1 obserbation

Error.fun.1obs <- function(input,output,w,dim.nn){

  d2 <- dim.nn[-1]

  d1 <- dim.nn[-length(dim.nn)]

  boundary <- c(0,cumsum(d1*d2))

  temp <- input
  for (m in 2:length(dim.nn)){

    temp <- sigmoid(vec2mat(w[(1+boundary[m-1]):boundary[m]],dim.nn[m],dim.
        nn[m-1])%*%temp)
  }

  Error <- sum((as.vector(temp)-sigmoid(output))^2)

  return(Error)
}


### Error function for the whole training sample

Error.fun.allobs <- function(tr.ts,w,dim.nn){

  Error <- 0

  for( i in  (dim.nn[1]+1):(length(tr.ts)-dim.nn[length(dim.nn)])){

    Error <- Error + Error.fun.1obs(tr.ts[(i-dim.nn[1]):(i-1)],tr.ts[i:(i+
        dim.nn[length(dim.nn)]-1)],w,dim.nn)
  }

  return(Error/length(tr.ts))

}
```

```r


### Forecast function

Forecast.fun <- function(input,w,dim.nn){

  d2 <- dim.nn[-1]

  d1 <- dim.nn[-length(dim.nn)]

  boundary <- c(0,cumsum(d1*d2))

  temp <- input
  for (m in 2:(length(dim.nn)-1)){

    temp <- sigmoid(vec2mat(w[(1+boundary[m-1]):boundary[m]],dim.nn[m],dim.
        nn[m-1])%*%temp)
  }

  temp <- vec2mat(w[(1+boundary[length(dim.nn)-1]):boundary[length(dim.nn)
      ]],dim.nn[length(dim.nn)],dim.nn[length(dim.nn)-1])%*%temp

  return(temp)
}


################ Feed the data to the Network ###################

data <-read.csv("here should be the directory of your csv file.csv")

dates <- as.Date(as.vector(data[,3]),"%d/%m/%y") #This makes R convert your
    dates into day-month-year format
# a third column shall be the dates in your csv file

ts <- xts(sber[,5],order.by=as.POSIXct(dates)) #This treats stock prices as
    time series related to dates
#index 5 indicates that your price series is in the 5th column of csv file

#Take some estimation window
real.training.ts<-ts["2011-01/2012-12"]

#And this is an estimation window plus one month test window
real.plot.ts <- ts["2011-01/2013-01"]


###Set the parameters of the Network

dim.nn <- c(4,3,1)

d2 <- dim.nn[-1]
d1 <- dim.nn[-length(dim.nn)]
```

```r

N <- sum(d1*d2)

w <- rnorm(N) #random weight vector


real.training.ts <-as.vector(real.training.ts)

lag<-length(real.training.ts) #number of observations in a training sample

real.plot.ts <- as.vector(real.plot.ts)

window <- length(real.plot.ts) - length(real.training.ts) #number of days to
    forecast ahead


### Normalize the data


training.ts <-(real.training.ts-mean(real.training.ts))/sd(real.training.ts)

plot.ts <-  (real.plot.ts-mean(real.training.ts))/sd(real.training.ts)


### Find the minimum of Error function (Newton-type algorithm)

min.func <- function(x,t1 = training.ts,dims=dim.nn){

  return(Error.fun.allobs(t1,x,dims))
}

global.min<-nlm(f=min.func,p=as.vector(w),print.level=2,iterlim = 100)

w.min <-  global.min$estimate #Here is the trained weight vector


### Evaluate Forecast

forecast.ts <- plot.ts

for (i in (dim.nn[1]+1):(length(plot.ts)-dim.nn[length(dim.nn)])){

  tf <- Forecast.fun(as.vector(plot.ts[(i-dim.nn[1]):(i-1)]),w.min,dim.nn)

  forecast.ts[i] <- tf[1]

}

### Now you can plot the forecast vs the real price
```

script.R