CAN DEEP MACHINE LEARNING OUTSMART THE MARKET? A COMPARISON BETWEEN ECONOMETRIC MODELLING AND LONG- SHORT TERM MEMORY

Eva DEZSI, Ioan Alin NISTOR[1]

Abstract:
Using long-short term memory (LSTM) recurrent neural network (RNN) architecture, we analyse data from the Romanian stock markets in the attempt to forecast its future trend. Then we try to compare the results using the classical statistical modelling tools, further employing back testing to prove our findings. We believe that the LSTM should be the next tool in balancing portfolios and reducing market risk.

Key words: LSTM, RNN, Neural Networks, Deep learning, Stock prices prediction

1. Introduction

In this paper we propose to study the possibility of tracking and predicting the price of a stock return within the main index of the Romanian stock market. The goal of the paper is to explore the possibility of employing a machine learning mechanism for the Romanian market, and to observe the results of the actual versus predicted prices via a machine learning algorithm. A class of Recurrent Neural Networks (RNN) was chosen, namely LSTM. Backpropagation was introduced by Rumelhart, Hinton and Williams (1986) as a network learning procedure, together with RNN, a family of neural networks designed to process sequential data. RNN's can be viewed as a class of artificial neural networks where the connections between neurons create a directed acyclic graph. Unlike feed-forward networks, RNN's have the advantage to persist the information about the past, as a consequence taking into consideration also long-term dependencies.

Are stock prices predictable? The investigation of random walk can be traced back to Luis Bachelier (1990), and from that moment in the literature a two sided debate emerged, whether stock-price movements are random, semi-random or predictable[2]. Firstly, Malkiel (2003, 2016) argues that stock prices are best described by a random walk process, meaning each day's deviations from the central value are random and unpredictable. On the other side Lo and MacKinlay (1988, 1999); Ang and Bekaert (2007) consider that the random walk hypothesis of stock-prices is false, and that markets are predictable to some degree.[3] From these two taught of schools emerged the belief that investors can't beat the market based on the present and historical information, in this case markets being efficient. In the second case, investors can outperform the market,

[1]Author 1 teaches at the Babes-Bolyai University - Faculty of Business of Cluj, e-mail address: eva.dezsi@tbs.ubbcluj.ro; Author 2 teaches at the Babes-Bolyai University - Faculty of Business of Cluj, e-mail address: ioan.nistor@tbs.ubbcluj.ro

[2] For a full review of the literature see Dragota and Oprea (2014), Degustis ans Novickyte, (2014), Malkiel (2003).

[3] For a review of stock market forecasting techniques see Atsalakis and Valavanis (2009, 2013).

depending on the level of market efficiency. From the machine learning point of view, as Chiyuan et al. (2017) and Perez (2016) underline, when maximum entropy[1] is present in the data, randomness can be assumed as the gerenaring process. In this case the machine learning algorithm can't find those characteristics that could help generate the process, thus machine learning will be equal with memorizing, not learning. So if not maximum entropy is characteristic for the data, the machine learning alhorithm can learn. In this was, by analyzing the performance of the network, we can make interfereces about the predictability of the market. In our case, we employed a deep neural network to generate the next day stock prices by observing the previous day's Open, High, Low and Closing Prices. In this manner, without taking a stand beside one or the another shools of taught, we can apply stock price forecasting. If markets follow a random walk, the neural network will memorize the prices of the test data, but on the training set will perform badly. On the other side, if there is a pattern in the stock price movement, the network is powerfull enough to extract these patters, and will appy these on the test set.

But what is a neural network? They are usually viewed as black box approaches, as these networks mimic the human brain as it performs computation and learning processes. A neural network is built in a similar manner, with cells that are connected to each other, each cells has several inputs, performs computations, and then passes along these results to another cells, while in the end an output result is obtained. Neural networks are very similar to functions, but they are very flexible to recreate any linear or non-linear function. In traditional statistics the parameters of a function is estimated through different econometric estimators. In the case of neural networks the parameters are learned and fine-tuned by the network itself, and hence the similarity with the brain. The learning process is done via an optimization algorithm, which starts with random weights, than it is learning by trial and error, trying to minimize the difference between the estimated output and the real output. This is the 'training' phase of the network, when the network is told what the real values are, and so the weights of the neurons are calibrated until the best results are obtained thought back-propagation. This is the case of supervised machine learning algorithms. One of the main issues of back-propagation is the vanishing gradient problem, together with the more rarely observed exploding gradient. As exemplified by Goodfellow, Bengio and Courville (2016), the issue is that the long-term dependencies are forgotten by the network when performing the multiplication of the Jacobian matrices. Goodfellow, Bengio and Courville (2016) view the LSTM and other gated RNN's the most effective sequence models that are capable to solve the vanishing or exploding gradient problem, in the same time being able to be employed in sequence modelling.

Also, as Yudong and Lenan (2009) and Oh and Kim (2002) underline, stock markets are dynamic systems, characterized by non-linearity and chaotic nature. A multi layered neural network is well suited for a system where complex patterns emerge, as are stock markets. Also the Romanian stock market is viewed according to Dragota and Oprea (2014); Todea and Plesoianu (2011), less efficient and more prone to predictability. These all determined us to apply a machine learning algorithm on the Romanian stock market.

Related to the current machine learning algorithm employed for stock market forecasting, Yudong and Lenan (2009) implemented an improved bacterial chemotaxis optimization method into a back-propagating neural network in order to forecast the S&P

[1] Information entropy can be defined as the average information of all possible outcomes.

500 index. The employed model was an adaptive BP ANN, with the weight updated according to the IBCO tool. The results were evaluated for forecasting the S&P 500 index one and fifteen days in advance, comparing the results with a traditional ANN. They concluded that with the inclusion of the IBCO optimization method better prediction accuracy, less training time and less computational complexity was achieved.

Oh and Kim (2002) implement a stock trading model based on a back-propagating neural network. The model was set up to determine the next day's price of a price index based on the current price and to generate Buy or Sell signals. The model was employed on the daily Kospi 200 index from January 1990 to August 2000. The results of the authors suggest that predictability of stock price indexes improves significantly using a neural net.

Kim (2006) applied an artificial neural network with an evolutionary instance for the Korea stock price index, between 1991 and 1998. The data was divided into eight data sets, while the experiment was repeated eight times for each data set, so that knowledge is accumulated by the network as time passes. In this study not the price was predicted, but the direction of the price movement, namely will the next day trading price will be lower or higher than today. As inputs were given not just the price of the index, but a number of technical indicators as well. Average predictive ratio for training data, for all years, was 64.13% while for the holdout data it was 59.23% for GANN, whilst the performance of the training data for the GAIN was 74.87%, while for the holdout was 65.45%

Kuo et al. (2001) propose a genetic algorithm based on a fuzzy neural network (GFNN), to integrate fuzzy inference rules that can measure and estimate the macroeconomic or political qualitative effects on the stock market. Also an Artificial neural network was incorporated with the GFNN, to include also the index of the market in the decision of the network. The network was trained and tested on the Taiwan stock market. The logic behind their rationale was that fuzzy logic systems can simulate the stock markets experts' knowledge. This knowledge combined with the market index resulted in an intelligent stock trading decision support system. The expert knowledge was mapped as a political dimension combined with a financial dimension, together with economic, international, message and technical dimensions. The created system has three parts, firstly factor identification part, secondly a qualitative model GFNN, and thirdly decision integration in the form of an ANN. The results of Kuo et al. (2001) suggest that a neural network that looks at both qualitative and quantitative factors outperforms a network whose learning is only based on quantitative factors. The performance was evaluated based on buying-selling points as on buying-selling performance.

Chen et al. (2003) also on the Taiwan stock market implement a probabilistic neural network. They motivated the choice of the market by the emerging nature of it. They model the excess return on the Taiwanese market on 3, 6 and 12 months horizon, and choose short-term interest rate, government consumption level lagged on 12 months, gross domestic product lagged on 12 months, consumer price and production level lagged on 12 months. Their findings suggest that the PNN neural network is able to outperform from the point of view of predictive power the MM-Kalman filter and random walk forecasting models. According to the authors view, the PNN possesses the capabilities to identify outliers and erroneous data, together with the ability of estimating the underlying probability density function without prior assumptions.

Guresen at al. (2011) also employ an ANN for stock market prediction. They analyse different models: Multi-Layer Perceptrons, Dynamic Artificial Neural Network

and Hybrid Neural Network with GARCH to extract new input variables. The created models were evaluated by MSE and MAD indicators. NASDAQ daily quotes from 2008 to 2009 were employed. The authors conclude that MLP model has the best performance, while hybrid and dynamic models have underperformed the rest of the models.

Moghaddam et al. (2016) investigate the predictability of the NASDAQ index through a feed forward ANN. The input for the ANN consisted from the short-term historical prices, together with the days of the week. Four and nine days historic prices were added as separated inputs to the network, with several activation functions. Moghaddam et al. (2016) obtained R^2 above 0.9622, which validates the predictive capability of the ANN.

Dixon et al. (2016) implement a five hidden layered DNN for algorithmic trading purposes. Their dataset consisted from 5 minute mid-prices for 43 CME listed commodity and FX futures from 1991 to 2014. The data was normalized before feed into the network, and divided into 25.000 consecutive training set, together with 15.000 testing set. The dataset included lagged price differences from 1 to 100 lags, moving price averages with windows size 5 to 100, and pair-wise correlation coefficients between the returns. Dixon et al. (2016) argued that the motivation of introduction of these engineered features emerged from the fact that in this way the network should be able to capture beside the historical movements also the co-movements. Dixon et al. (2016) also illustrate a trading strategy based on DNN, demonstrating how prediction accuracy can lead to profitability.

Khaidem et al. (2016) proposed a novel approach in the area of forecasting of stock market prices, namely the minimization of forecasting error by the treatment of it as a classification problem. Technical indicators such as Relative Strength Index, Stochastic Oscillator, William %R, Moving Average Convergence Divergence, Price Rate of Change, On Balance Volume were employed as input for the model, while the learning model was an ensemble of multiple decision trees. The output is not the stock price, but a buy or a sell signal, generated by the model. Three data sets were employed, Samsung traded on the Korean Stock Exchange, together with the Apple and General Electric stocks, traded on NASDAQ. Their results are encouraging, accuracy of 85% to 94% was obtained, depending on dataset and trading period.

Arteta et al. (2016) propose an ANN structure with grammatical swarm, while the raining process is achieved by the use of HydroPSO. Also the authors employ a grammatical swarm algorithm to generate trading rules. The IBEX stock index was chosen from 2002 to 2014, with prices of 30 min. A total of 55.490 patterns were employed, while the dataset consisted from 176 inputs, 16 trading indicators with one to ten lags, two price changes, for one and five days. They compare their results with the results of tradingmotion.com, and they conclude that the proposed grammatical swarm behaves, in general, better than many automatic system described there.

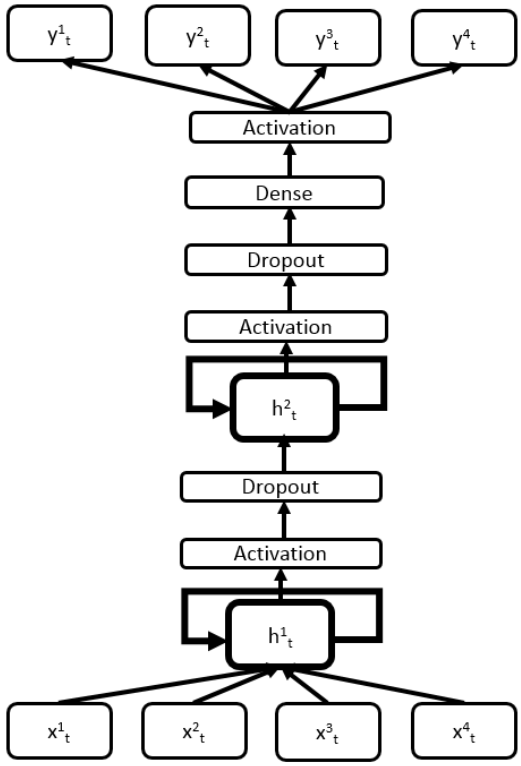
Qiu et al. (2016) applied an ANN on the Nikkei 225 index, with 71 input variables consisting of financial indicators and macroeconomic data. To improve prediction accuracy Genetic Algorithm and Simulated Annealing search techniques were employed. Their results suggest that an ANN based on GA and SA improves the prediction accuracy, while outperforming a traditional BP training algorithm.

For a review of stock market forecasting techniques using soft computing methods see Atsalakis and Valavanis (2009) and Guresen et al. (2011).

2. Research Methodology

Deep learning networks are just a variation of neural networks, where more than one hidden layer is present in the network. Usually, as mentioned by Goodfellow, Bengio and Courville (2016), a computation of a network can be separated in three phases, first from the input data to the hidden state of the network, secondly, from the hidden stated to another hidden state, and thirdly, from the last hidden state to the output. As demonstrated by Graves, Mohamed and Hinton (2013) together with Pascanu, Gulcehre, Kyunghyun, Bengio, (2014) networks perfom better with multiple hidden layers. In our network two layers of LSTM are employed, together with Droput, Activation and Dense Layers.

FIGURE 1. Applied recurrent LSTM network.[1]



In traditional recurrent neural networks, gradient back-propagation is employed during learning. In this phase the gradient is multiplied a large number of times, usually the number of time steps, with the weight matrix which identifies the connections between the neurons of the recurrent layer. In this way gradient back-propagation influences the learning process, as the magnitude of weights of neur impacted by the continuous multiplication. For example, if the leading eigenvalue of the weight matrix is smaller than 1, which can be translated in weights of the transition matrix between 0 and

[1]Source: The authors own processing after Goodfellow, Bengio and Courville (2016), Karpathy (2015), Paszke (2015), Olah (2015) representation of LSTM and RNN networks.

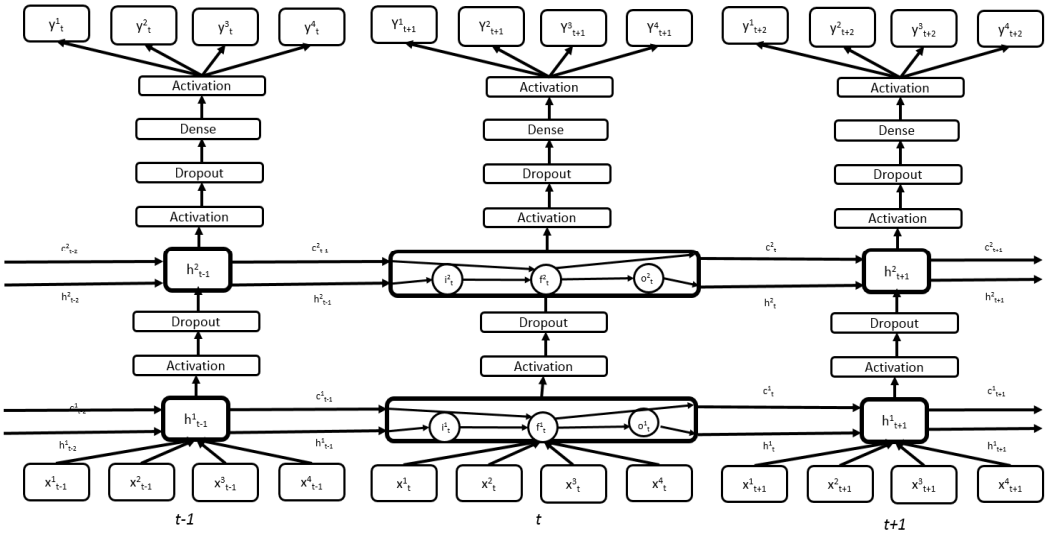
1, then the learning rate of the networks becomes very slow or inexistent. This phenomenon is known as the vanishing gradient problem, which also translates in a more troublesome learning of long-term dependencies. LSTM's or long short-term memory networks were proposed by Hochreiter and Schmidhuber (1997) to avoid the vanishing or exploding gradient problem. This methodology is a class of deep machine learning algorithms, which in the recent years caught a lot of attention. In the below figure the network that was employed in the learning process can be visualized. Two hidden LSTM layers were included, each followed by an Activation and a Dropout Layer, as in Figure 1. Before the output is calculated by the network, a Dense layer in combination with an Activation layer was employed.

An activation layer in Keras is behaving as a function to transform an input of a neuron to an output signal. As a consequence, as indicated by Goodfellow, Bengio and Courville (2016), activation functions are used to compute the value of hidden layer, or the output, from the input values.

Dropout was introduced by Srivastava et al. (2014), as a technique of avoiding overfitting of a neural network to the trained data. Dropout is a randomly applied deletion mechanism, which drops units of a network together with its connections, in this way preventing the network of adapting too much to the training data. In our model dropout was applied to the LSTM layers of the model. We assured in this way that the network becomes more insensitive to the training data, and so it is more robust. This means that even if a shock of the market was introduced to the training data, for example the financial crisis of 2007, the model was forced to 'forget' some details, in order to avoid memorizing specific characteristics of the financial crisis, which may be present only in this shock, and not it others. Scheau (2016) views dropout as an ensemble algorithm, where the training phase of a neural network with dropout can be translated as training a collection of individual networks with parameter sharing, where in the end all the separate networks are merged into one. Dropout was employed in our network as well, as a regularisation technique within deep networks, where network units are randomly dropped during training, in order to avoid overfitting of the data.

A dense layer is a classic fully connected neural network layer: each input node is connected to each output node, with an activation function. In this way, the dense layer is similar to a dropout layer, with the exception that the activations are set to zero for some random nodes in the case of a dropout layer. The unfolded structure of the resulted neural network is illustrated in Figure 2. h_{t-1}^{1} and h_{t-1}^{2} represent the hidden layer LSTM 1, respective LSTM 2 at time *t-1*.

FIGURE 2. Unfolded LSTM network.[1]



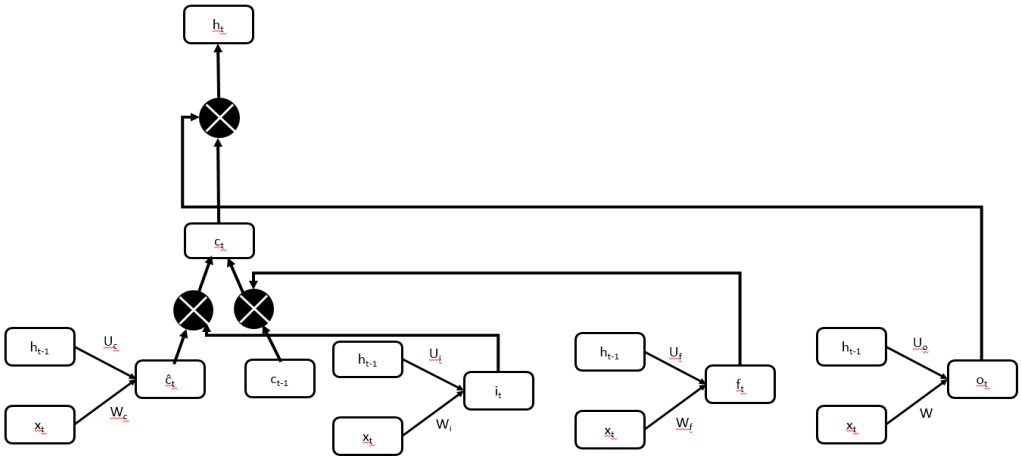
x^1_{t-1} represents the Opening stock prices return at time step *t-1*, x^2_{t-1} the Highest stock prices return for time step *t-1*, x^3_{t-1} the Lowest stock price return for time step *t-1*, x^4_{t-1} the Closing stock prices return for time step *t-1*. As a conclusion, with x_{t-1} was illustrated the input given to the network at time *t-1*, for predicting the output y_t at time *t*. Similarly to the input, the output of the network is represented by the Opening stock prices return at time *t,* denoted by y^2_t , the Highest stock prices return at time *t*, y^3_t the Lowest stock price return at time *t*, y^4_t the Closing stock prices return at time *t.*

In our case the networks is a many to many recurrent network, where 4 inputs - Open, High, Low and Close returns are feed to the network, while the network is predicting based on time step *t* time step *t+1* Open, High, Low and Close returns.

The fascinating characteristic of the LSTM network is illustrated in Figure 2, as in Figure 3. This is given by c_t, which is the candidate value computed by the network for a memory cell, or the internal memory of the cell. For the ease of illustration, the LSTM memory cell was visualized only with one input. In Figure 3. we can consider under x_t all the inputs – Open, High, Low and Close returns as a vector feed to the memory cell layer at time *t*.

[1]Source: authors own processing after Goodfellow, Bengio and Courville (2016), Karpathy (2015), Paszke (2015), Olah (2015) representation of LSTM and RNN networks.

FIGURE 3. LSTM cell.[1]



As a general notation, after DeepLearning 0.1 documentation, we express with W_i, W_f, W_o, W_c, U_i, U_f, U_o, U_c, V_o the weight matrices, while b_i, b_f, b_o, b_c represent the bias vectors. First through the input gate the values of i_t are computed, as in equation (1), translated this can be viewed as how much information of the newly computed state for the current input is let through the cell. The input, forget and output gates have the same equation for computation purposes, just different parameter matrices are employed for each. These functions are traditionally named as gates, as the sigmoid function, as in equation (7), supresses the values of these vectors between the values 0 and 1. Then the forget gate f_t, exemplified in equation (2), determines how much from the input is kept, and how much is forgotten from the previous state. All the gates have the same dimension. $\hat{c}_t$ is the candidate value for the states of the memory cell at time *t*, given by equation (3). The candidate hidden state is calculated based on the current input and the previous hidden state, through a tangent activation function.

The input i_t together with the candidate value of the present state $\hat{c}_t$, with the forget gate f_t and the previous states memory c_{t-1} form the internal memory unit of the present state c_t. Equation (4) can be viewed as a combination of how much previous memory is combined with present inputs. For example, if the network learns that it is better to forget all previous memory, as the current input is the best predictor of the current output, then the forget gate will be set to 0. At the other extreme would be the case where the network finds that it is better to ignore the newly computed state, and base its prediction just on the previous memory, then the input gate will be 0. The output gate, given by equation (5) defines how much from of the internal state is externalized to the hidden layers at another time step. Given c_t the memory cell for time step *t*, h_t the hidden state for time step *t* will be computed, as shown by equation (6). The gated mechanism of the LSTM is what determines LSTM to learn and model long-term dependencies.

[1]Source: The authors own processing after Goodfellow, Bengio and Courville (2016) and Paszke (2015).

$$i_t = \sigma\left(W_i \cdot x_t + U_i \cdot h_{t-1} + b_i\right) \quad (1)$$

$$f_t = \sigma\left(W_f \cdot x_t + U_f \cdot h_{t-1} + b_f\right) \quad (2)$$

$$\hat{c}_t = \tanh\left(W_c \cdot x_t + U_c \cdot h_{t-1} + b_c\right) \quad (3)$$

$$c_t = i_t \cdot \hat{c}_t + f_t \cdot c_{t-1} \quad (4)$$

$$o_t = \sigma\left(W_o \cdot x_t + U_o \cdot h_{t-1} + V_o \cdot c_t + b_o\right) \quad (5)$$

$$h_t = o_t \cdot \tanh\left(c_t\right) \quad (6)$$

Several activation functions were used, for the reason of ensuring robustness in the networks estimates. The employed activation function were the Sigmoid activation function in equation (7), the Tangent activation function in equation (8), while in equation (9) the Tangent activation function expressed as Sigmoid function is exemplified, the Softplus activation function in equation (10), the Linear activation function in equation (11) and the Softsign activation function in equation (12).

$$g(z) = \sigma(z) = 1/\left(1 + \exp^{(-z)}\right) \quad (7)$$

$$g(z) = \tanh(z) = 1/\left(1 + \exp^{(-2x)}\right) - 1 \quad (8)$$

$$\tanh(z) = 2\sigma \cdot (2z) - 1 \quad (9)$$

$$g(z) = \varsigma(z) = \log\left(1 + \exp^{x}\right) \quad (10)$$

$$g(z) = z \quad (11)$$

$$g(z) = \delta(z) = x/\left(1 + |x|\right) \quad (12)$$

The assesment of fitness of the predicted residuals was done via Root Mean Square Error – RMSE, given by equation 13, and Mean Absolute Error – MAE (illustrated by equation 14).

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(e_i)^2} \quad (13)$$

$$MAE = \frac{1}{n}\sum_{i=1}^{n}\left|\hat{y}_i - y_i\right| = \frac{1}{n}\sum_{i=1}^{n}\left|e_i\right| \quad (14)$$

As mentioned, we also choose a traditional econometric technique to model the returns. The returns R_t are calculated in the continuous compounding form by taking the logarithm of the ratio of two consecutive daily prices P_t and P_{t-1}: $R_t = \ln(P_t / P_{t-1})$. Besides modelling the mean equation, as in equation (15), the variance equation was also modelled. In equation (15) $\mu_i = \left[\mu_{Open} \quad \mu_{High} \quad \mu_{Low} \quad \mu_{Close}\right]^T$ represents an array of size 6x1, with T symbolizing the transposed matrix. Different *j* lags were chosen to model each mean equation of the returns.

$$R_{i,t} = \mu_i + \sum_{j=1}^{n} \Phi \cdot R_{i,t-j} + e_{i,t} \quad (15)$$

Vector e_t has also a 6x1 dimension, given by equation (16), and it expresses idiosyncratic shocks on the market that follows a normal distribution. We define ε_t as the vector of **residuals, given by equation (17).**

$$e_t = \left\lfloor e_{Open,t} \quad e_{High,t} \quad e_{Low,t} \quad e_{Close,t} \right\rfloor \tag{16}$$

$$\varepsilon_{i,t} = \left(R_{i,t} - \mu_i + \sum_{j=1}^{n} \Phi \cdot R_{i,t-j} \right) \tag{17}$$

With the introduction of the ARCH[1] model of Engle (1982) and the GARCH model of Bollerslev (1986) and Taylor (1986), in the economic literature there was an outburst of approaches specializing on the modelling of the variance equation. These models were specifically designed to capture the features, or stylized facts, noted in the evolution of stock markets prices and returns, such as the phenomenon of volatility clustering, leptokurtosis and leverage effect. This is the reason why the GJR-GARCH proposed by Glosten, Jagannathan and Runkle (1993), as in equation (18) model was chosen. It allows the conditional variance to respond differently to the past negative and positive innovations. α_j is the ARCH coefficienct, β_j stands for the GARCH coefficient, while γ_j is a parameter that captures the assymetry effects in the variance equations.

$$\sigma_{i,t}^2 = \omega + \sum_{j=1}^{q} \alpha_j \cdot \varepsilon_{i,t-j}^2 + \sum_{j=1}^{n} \gamma_j \cdot \varepsilon_{i,t-j}^2 \cdot I\left(\varepsilon_{i,t-j} < 0\right) + \sum_{j=1}^{p} \beta_j \cdot \sigma_{i,t-j}^2 \tag{18}$$

3. Data and Results

This section describes the data used for the analysis, and provides descriptive statistics and diagnostic tests. We collected daily returns for 'BRD' stock, listed on the Romanian stock market in 2001. The data that on which the network was trained and tested was gathered from the Bloomberg database. The sample includes the period between 16th of January 2001 to the 4th of November 2016, with a total sample size of 3809 daily observations. This data covers 15 years, and allows capturing the effects of the financial crisis which began in 2007. The fact that we included in our sample a shock allowed us to train the network on data displaying non-normal behaviour. In this was the network observed the behaviour of the stock in several different phases, and this increased the robustness of the estimates of the network. This characteristic also influenced positively the learning curve of the network, as it helped avoiding overfitting of the data, which can be translated as the network learning the prices of the daily stocks, and not by learning the dynamics that influence the evolution of the prices. The Open, Highest, Lowest and Closing price for each day of training of the sample were given to the network, as logarithmic returns. Summary statistics of the returns can be found in Table 1. According to the statistical properties of the returns, most of the indices depart from normality, with positive skewness and excess kurtosis. We can observe that the time series exhibit fat tail phenomenon.

[1] See Terasvirta (2006) and Bollerslev (2010) for a synthesis of the economic literature of the ARCH and GARCH models.

The network was built in Pyhton 2.7, where Keras Python library was employed on the top of Theano. For robustness purposes several optimizers were adopted, with combination of different activation functions. The employed optimizers were - Stochastic gradient descent – SGD, with support for momentum, learning rate decay, and Nesterov momentum.

TABLE 1: Summary Statistics of the BRD daily logarithmic returns.[1]

	BRD Open	BRD High	BRD Low	BRD Close
Mean	0.000529	0.000506	0.000524	0.000507
Median	0.000000	0.000000	0.000000	0.000000
Maximum	0.544112	0.522802	0.544112	0.531879
Minimum	-0.397097	-0.372404	-0.428846	-0.397097
Std. Dev.	0.027432	0.024225	0.026157	0.025805
Skewness	0.750709	1.759053	0.812212	1.257311
Kurtosis	65.16079	85.99398	79.06637	72.92742
JB	612634.5	1093422.	917272.6	775838.5
Note:	The Jarque-Bera (JB) test is a goodness-of-fit statistical test of whether the sample data has the skewness and kurtosis of a normal distribution.			

Root Mean Square Propagation or RMSprop was also used, as introduced by Tieleman and Hinton (2012), a method in which the learning rate is adapted for each of the parameters. Adaptive gradient algorithm or Adagrad, also employed, is a modified stochastic gradient descent with per-parameter learning rate, as presented by Duchi, Hazan and Singer (2011). Adadelta, introduced by Zeiler (2012), was also adopted, and can be viewed as an extension of Adagrad, a method that seeks to reduce its aggressive, monotonically decreasing learning rate. Adam or Adaptive Moment Estimation, as proposed by Kingma and Ba (2014), is a method that also computes adaptive learning rates for each parameter. In addition of storing an exponentially decaying average of past squared gradients similar to Adadelta and RMSprop, in Adam exponentially decaying average of past gradients, similar to momentum, are also employed. Adam, can be viewed as RMSprop with momentum, and it was also used as an optimizer for our network. Adamax, also introduced by Kingma and Ba (2014), was utilized, and it is a variant of Adam based on the infinity norm. Nesterov Adam optimizer or Nadam, is an ADAM with Nesterov momentum, was also employed. A total sample of 3803 returns was feed to the network, from which 2282 were directed for training of the network, and 1521 for testing the network. Dropout was set to 0.2, while the number of epoch was 50, with a batch size of 500. A total of 12.028.004 network parameters were estimated for each activation – optimizer combination. In Table 1., Table 2, Table 3. and Table 4. the different variants of the network RMSE and MAE measures are found. As it can be observed, the best results are obtained via the Softsign activation function, while the best optimizer is ADAM. The best combination of optimizer and activation function for all the input series was Softsign ADAM, while the least best performance was achieved by Sigmoid SGD.

[1]Source: Authors own processing.

TABLE 2: Goodness of fit measures for Opening returns[1]

Activation function	Optimizer	Open			
		RMSE		MAE	
		Train sample	Test sample	Train sample	Test sample
Linear	**SGD**	0.032561671	0.01725148	0.018449621	0.011445764
Linear	**RMSprop**	0.02730019	0.015170759	0.014636572	0.010942324
Linear	**Adagrad**	0.026192363	0.013149689	0.012906953	0.0086694593
Linear	**Adadelta**	0.032514952	0.017261021	0.018482482	0.011505895
Linear	**ADAM**	0.026232895	0.013071077	0.012939851	0.0085875504
Linear	**ADAMAX**	0.026281439	0.013270446	0.013046761	0.0088148769
Linear	**NADAM**	0.02638917	0.013768448	0.013357326	0.0093746046
Sigmoid	**SGD**	0.050297394	0.042955719	0.041859273	0.039759014
Sigmoid	**RMSprop**	0.032498598	0.017199293	0.018370984	0.011390553
Sigmoid	**Adagrad**	0.032498982	0.017247599	0.018474037	0.011504208
Sigmoid	**Adadelta**	0.033004194	0.018465664	0.019433131	0.013065996
Sigmoid	**ADAM**	0.032497521	0.017237591	0.018458311	0.011483696
Sigmoid	**ADAMAX**	0.032712262	0.017828831	0.018938906	0.012308945
Sigmoid	**NADAM**	0.032560747	0.017454313	0.018674759	0.011829899
Tanh	**SGD**	0.032561671	0.01725148	0.018449621	0.011445764
Tanh	**RMSprop**	0.027368756	0.015326764	0.014781891	0.011119354
Tanh	**Adagrad**	0.026183454	0.013143085	0.012895763	0.0086618578
Tanh	**Adadelta**	0.032514948	0.017261021	0.018482482	0.011505894
Tanh	**ADAM**	0.026224531	0.013068868	0.01293469	0.0085847722
Tanh	**ADAMAX**	0.026273852	0.013263844	0.013036139	0.0088073723
Tanh	**NADAM**	0.026368354	0.013741083	0.013330089	0.0093401317
Softsign	**SGD**	0.032561466	0.017251473	0.018449765	0.011445969
Softsign	**RMSprop**	0.027084067	0.015080546	0.014496665	0.010835873
Softsign	**Adagrad**	0.026064975	0.013121966	0.012799455	0.0086258706
Softsign	**Adadelta**	0.032514948	0.017261088	0.018482637	0.011506058
Softsign	**ADAM**	0.02609976	0.013012706	0.012812519	0.0085172802
Softsign	**ADAMAX**	0.026139632	0.013186047	0.012899525	0.0087194284
Softsign	**NADAM**	0.026137119	0.013624438	0.013165363	0.0091842832
Softplus	**SGD**	0.038996916	0.028342228	0.028179031	0.024019526
Softplus	**RMSprop**	0.032501131	0.017194528	0.018350312	0.011372278
Softplus	**Adagrad**	0.032507438	0.017190216	0.018315233	0.011341261
Softplus	**Adadelta**	0.032784857	0.017993411	0.019055251	0.012504444
Softplus	**ADAM**	0.032503441	0.017192168	0.018335799	0.011359442
Softplus	**ADAMAX**	0.03249643	0.017209534	0.018401997	0.011420658
Softplus	**NADAM**	0.032502562	0.017266482	0.018500376	0.011540139
TARCH		0.027334339	0.017954525	0.018788503	0.012914461

[1]Source: Authors own processing.

TABLE 3: Goodness of fit measures for High returns[1]

Activation function	Optimizer	High			
		RMSE		MAE	
		Train sample	Test sample	Train sample	Test sample
Linear	**SGD**	0.029261706	0.013156263	0.016403452	0.0090845563
Linear	**RMSprop**	0.027364574	0.012224074	0.015421454	0.0091965944
Linear	**Adagrad**	0.027255004	0.012012178	0.015133424	0.0087771453
Linear	**Adadelta**	0.029249299	0.013180274	0.016485272	0.0091692591
Linear	**ADAM**	0.02720465	0.01188367	0.015037059	0.008667754
Linear	**ADAMAX**	0.027228696	0.011967466	0.015123575	0.0088121202
Linear	**NADAM**	0.027171087	0.012123482	0.015330522	0.0090909237
Sigmoid	**SGD**	0.047562249	0.040619485	0.040441595	0.038598347
Sigmoid	**RMSprop**	0.02933687	0.013244401	0.016569091	0.0092207231
Sigmoid	**Adagrad**	0.029335722	0.013228439	0.01653217	0.0091796992
Sigmoid	**Adadelta**	0.029636877	0.014172006	0.017471144	0.010627517
Sigmoid	**ADAM**	0.029340575	0.013272264	0.016623292	0.0092875166
Sigmoid	**ADAMAX**	0.029472193	0.013725796	0.017103778	0.010036838
Sigmoid	**NADAM**	0.029356614	0.013349646	0.016739557	0.0094414335
Tanh	**SGD**	0.029261706	0.013156263	0.016403452	0.0090845563
Tanh	**RMSprop**	0.027413961	0.012381184	0.015565428	0.0094074691
Tanh	**Adagrad**	0.027254472	0.012010427	0.015132839	0.008775793
Tanh	**Adadelta**	0.029249299	0.013180274	0.016485272	0.0091692591
Tanh	**ADAM**	0.02720423	0.011884199	0.015038697	0.0086703831
Tanh	**ADAMAX**	0.027226686	0.011964752	0.015122827	0.0088100964
Tanh	**NADAM**	0.027170589	0.012142562	0.01535084	0.0091198506
Softsign	**SGD**	0.029261826	0.01315625	0.016403157	0.0090842089
Softsign	**RMSprop**	0.027755165	0.013374306	0.016426792	0.010644337
Softsign	**Adagrad**	0.027206302	0.011959376	0.015123876	0.0087676086
Softsign	**Adadelta**	0.029249419	0.013180099	0.016484724	0.0091684861
Softsign	**ADAM**	0.027176082	0.011859636	0.015036229	0.0086665843
Softsign	**ADAMAX**	0.027182871	0.011930119	0.015114626	0.0087970076
Softsign	**NADAM**	0.027069658	0.012087381	0.015325778	0.0090870438
Softplus	**SGD**	0.036615003	0.026418429	0.027585262	0.023833308
Softplus	**RMSprop**	0.029518722	0.013857262	0.017216692	0.010219726
Softplus	**Adagrad**	0.029346619	0.013185613	0.016348131	0.0090197474
Softplus	**Adadelta**	0.029412422	0.013545742	0.016948691	0.0097745685
Softplus	**ADAM**	0.029340487	0.013191932	0.016402857	0.0090632681
Softplus	**ADAMAX**	0.029337099	0.013246626	0.016573824	0.0092263725
Softplus	**NADAM**	0.029337903	0.013253604	0.016588155	0.0092437202
TARCH		0.020655542	0.009988945	0.012854217	0.007060065

[1]Source: Authors own processing.

TABLE 4: Gopdness of fit measures for Low returns[1]

Activation function	Optimizer	Low			
		RMSE		MAE	
		Train sample	Test sample	Train sample	Test sample
Linear	**SGD**	0.030996284	0.01628959	0.0170395	0.010212585
Linear	**RMSprop**	0.028287688	0.0148629	0.015331579	0.0094937244
Linear	**Adagrad**	0.028162757	0.013750587	0.015399161	0.0095015569
Linear	**Adadelta**	0.030978149	0.015472455	0.017040571	0.010213834
Linear	**ADAM**	0.028187368	0.013671701	0.015385818	0.0094323307
Linear	**ADAMAX**	0.028201876	0.013761167	0.015344925	0.0094386935
Linear	**NADAM**	0.028147157	0.013939347	0.015323224	0.0095410068
Sigmoid	**SGD**	0.049830742	0.042173013	0.041755702	0.040343773
Sigmoid	**RMSprop**	0.031030886	0.015466766	0.017038424	0.010216351
Sigmoid	**Adagrad**	0.031036841	0.015512501	0.017150886	0.010331078
Sigmoid	**Adadelta**	0.031622395	0.016700542	0.018010946	0.012052269
Sigmoid	**ADAM**	0.031025527	0.015512314	0.017077528	0.010232378
Sigmoid	**ADAMAX**	0.031159973	0.01604566	0.01732352	0.010777064
Sigmoid	**NADAM**	0.031093078	0.01569755	0.017247874	0.010563288
Tanh	**SGD**	0.030996284	0.015465437	0.0170395	0.010212585
Tanh	**RMSprop**	0.028292662	0.014434885	0.015333695	0.0095111486
Tanh	**Adagrad**	0.028161747	0.013747923	0.015400411	0.0095017077
Tanh	**Adadelta**	0.030978151	0.015472454	0.017040571	0.010213834
Tanh	**ADAM**	0.028186832	0.013671077	0.015385038	0.009431866
Tanh	**ADAMAX**	0.028200174	0.013758266	0.015344029	0.0094376598
Tanh	**NADAM**	0.02812469	0.013921857	0.015311881	0.0094963908
Softsign	**SGD**	0.030996408	0.015465423	0.017039474	0.010212553
Softsign	**RMSprop**	0.028579975	0.014737098	0.015726246	0.010261391
Softsign	**Adagrad**	0.028132124	0.013717534	0.015406164	0.0094944015
Softsign	**Adadelta**	0.030978328	0.015472456	0.017040607	0.010213825
Softsign	**ADAM**	0.028172443	0.01364644	0.015403722	0.0094435578
Softsign	**ADAMAX**	0.028168013	0.013724489	0.015340332	0.0094337938
Softsign	**NADAM**	0.028073519	0.013892499	0.015322425	0.0095376885
Softplus	**SGD**	0.038041677	0.027543509	0.027398916	0.024116494
Softplus	**RMSprop**	0.031033996	0.015597667	0.01702683	0.010213548
Softplus	**Adagrad**	0.03103875	0.015449272	0.017012877	0.010210119
Softplus	**Adadelta**	0.031460114	0.016194006	0.01774592	0.011633051
Softplus	**ADAM**	0.031035054	0.015453763	0.017023392	0.010212717
Softplus	**ADAMAX**	0.031036491	0.015489664	0.017019005	0.010211658
Softplus	**NADAM**	0.031029942	0.015521701	0.017042574	0.010217353
TARCH		0.020072923	0.012723029	0.012418231	0.008238668

[1]Source: Authors own processing.

TABLE 5: Goodness of fit measures for Close returns[1]

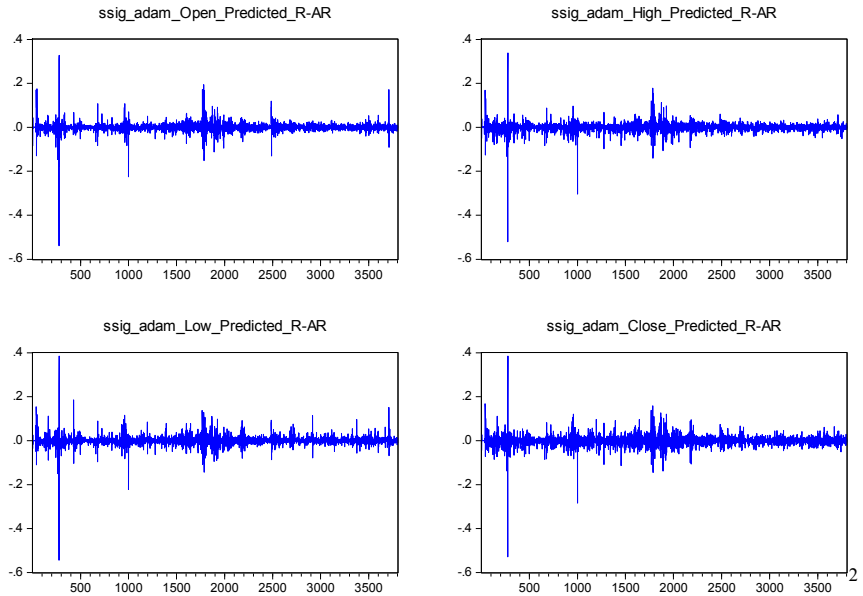
Activation function	Optimizer	Close			
		RMSE		MAE	
		Train sample	Test sample	Train sample	Test sample
Linear	**SGD**	0.031001655	0.014854879	0.018431351	0.011398439
Linear	**RMSprop**	0.030834084	0.014924763	0.018559186	0.011248262
Linear	**Adagrad**	0.030742332	0.013750587	0.018467536	0.010681178
Linear	**Adadelta**	0.031000743	0.015472455	0.01842705	0.011390971
Linear	**ADAM**	0.030778777	0.013671701	0.018472403	0.010767491
Linear	**ADAMAX**	0.030758867	0.013761167	0.018486969	0.010781307
Linear	**NADAM**	0.030735314	0.013939347	0.018429657	0.010966954
Sigmoid	**SGD**	0.049210932	0.042173013	0.041063674	0.039592892
Sigmoid	**RMSprop**	0.031007005	0.015466766	0.018397389	0.011375913
Sigmoid	**Adagrad**	0.031003131	0.015512501	0.0184872	0.011472939
Sigmoid	**Adadelta**	0.031436034	0.016700542	0.019261643	0.012829908
Sigmoid	**ADAM**	0.031012407	0.015512314	0.018552003	0.011565405
Sigmoid	**ADAMAX**	0.031240446	0.01604566	0.018959908	0.012340947
Sigmoid	**NADAM**	0.031060847	0.01569755	0.018682443	0.011796133
Tanh	**SGD**	0.031001657	0.015465437	0.018431351	0.011398439
Tanh	**RMSprop**	0.030844856	0.014434885	0.018582962	0.011315814
Tanh	**Adagrad**	0.030742018	0.013747923	0.018467527	0.010683012
Tanh	**Adadelta**	0.031000745	0.015472454	0.018427053	0.011390971
Tanh	**ADAM**	0.03077895	0.013671077	0.018472224	0.01076797
Tanh	**ADAMAX**	0.030758345	0.013758266	0.018486464	0.010782165
Tanh	**NADAM**	0.03073062	0.013921857	0.018428324	0.010961956
Softsign	**SGD**	0.031001695	0.015465423	0.018431371	0.011398445
Softsign	**RMSprop**	0.030815212	0.014737098	0.018411852	0.011048304
Softsign	**Adagrad**	0.030749395	0.013717534	0.018471962	0.010700201
Softsign	**Adadelta**	0.031000754	0.015472456	0.018427264	0.011391168
Softsign	**ADAM**	0.030774418	0.01364644	0.018467728	0.010757572
Softsign	**ADAMAX**	0.030750051	0.013724489	0.018483732	0.01077872
Softsign	**NADAM**	0.030707687	0.013892499	0.018431304	0.010998356
Softplus	**SGD**	0.037908919	0.027543509	0.027869921	0.024246063
Softplus	**RMSprop**	0.031009233	0.015597667	0.018384093	0.011363447
Softplus	**Adagrad**	0.031011961	0.014834682	0.01837042	0.011350625
Softplus	**Adadelta**	0.031175496	0.016194006	0.018862959	0.012163891
Softplus	**ADAM**	0.031004936	0.015453763	0.018413013	0.011390563
Softplus	**ADAMAX**	0.03100965	0.015489664	0.018538563	0.011545353
Softplus	**NADAM**	0.03102237	0.015521701	0.01858943	0.011625619
TARCH		0.028319724	0.013285615	0.017106774	0.009496279

We would like to mention that we estimated several versions of the models, changing the initial values of the optimization algorithm and the activation functions that were applied for all of these. In total a combination of 35 neuronal networks were trained.

[1]Source: Authors own processing.

For illustration purposes, under Figure 4., the difference between the predicted and actual returns by Softsign ADAM network can be visualized the for the train and the test data. Compared to the TARCH model in variance, the neural network only outperforms it with better predictions at Opening Prices, while for the other variables the TARCH model is superior taking into consideration RMSE and MAE. Another observation is that the neural network did not memorize, instead it learned. This can be inferred from comparing the training and test goodness of fit measures. As a general rule the network performed better on the test data than on the training set.

FIGURE 4. Softsign ADAM network predicted minus actual values of returns.[1]



4. Conclusions

This paper analysed different optimization strategies with several activation function on the Romanian stock market, on the BRD index. The Opening, Highest, Lowest and Closing price of each day were given to the network in the form of logarithmic returns, while the network was predicting the next day return. The performance of the neural network was measured by classical statistical modelling tools, further employing back testing to prove our findings. Our results suggest that the best activation-optimization algorithm is the Softsign ADAM combination. To our knowledge, on the Romanian market until the present time, no neural network was trained. Even if the neural network was outperformed by the TARCH model, we believe that this is due to the low number of features given as input parameters. As a consequence we propose to build in the future a network for all the stocks in the BET index, and to

[1]Source: authors own processing.
[2]The notation R is representing return, AR actual return.

analyse how a neural network can be trained to learn the dynamics of an entire market, not just the evolution of one stock price. Also a further step would be to incorporate in the networks input data technical indicators, as in Kim (2006). A much further step would be to include in the input data of the network also fundamentals indicators, together with macroeconomic indicators, and to leave the network to estimate the probable future price of the market taking all these elements into consideration. Such extensions will be left for forthcoming research.

References:

1. A Beginner's Guide to Recurrent Networks and LSTMs, https://deeplearning4j.org/lstm
2. Adam Paszke, 2015, LSTM implementation explained, https://apaszke.github.io/lstm-explained.html
3. Andrej Karpathy, 2015, The Unreasonable Effectiveness of Recurrent Neural Networks, http://karpathy.github.io/2015/05/21/rnn-effectiveness/
4. Ang Andrew, Bekaert Geert, 2007, Stock Return Predictability: Is it there?, The Review of Financial Studies, Volume 20, Issue 3, pp. 651-707.
5. An-Sing Chen, Mark T. Leung, Hazem Daouk, 2003, Application of neural networks to an emerging financial market: forecasting and trading the Taiwan Stock Index, Computers & Operations Research, pp. 901 – 923
6. Arteta Albert, A., Gómez Blas, N. & Mingo López, L.F.. Soft Comput (2016) 20: 2433. doi:10.1007/s00500-015-1652-2
7. Atsalakis S. George, Valavanis Kimon P., 2009, Surveying stock market forecasting techniques - Part II: Soft computing methods, Expert Systems with Applications, Vol. 36, pp.5932-5941 https://www.researchgate.net/profile/George_Atsalakis/publication/220216298_Surveying_stock_market_forecasting_techniques_-_Part_II_Soft_computing_methods/links/0fcfd50e84ecccfc70000000.pdf
8. Atsalakis S. George, Valavanis Kimon P., 2013, Surveying stock market forecasting techniques - Part I: Conventional methods, Computational optimization in economics and finance research compendium, New York, Nova Publ, pp. 49-103.
9. Bastien, Frédéric, Lamblin, Pascal, Pascanu, Razvan, Bergstra, James, Goodfellow, Ian, Bergeron, Arnaud, Bouchard, Nicolas, Bengio, Yoshua, 2012, Theano: new features and speed improvements. NIPS Workshop on Deep Learning and Unsupervised Feature Learning, http://www.iro.umontreal.ca/~lisa/pointeurs/nips2012_deep_workshop_theano_final.pdf
10. Bergstra, James, Breuleux, Olivier, Bastien, Frédéric, Lamblin, Pascal, Pascanu, Razvan, Desjardins, Guillaume, Turian, Joseph, Warde-Farley, David, Bengio, Yoshua, 2010, Theano: a CPU and GPU math expression compiler. In Proceedings of the Python for Scientific Computing Conference (SciPy). http://www.iro.umontreal.ca/~lisa/pointeurs/theano_scipy2010.pdf
11. Bollerslev Tim, (1986), Generalized Autoregressive Conditional Heteroskedasticity, Journal of Econometrics, Vol. 31, pp. 307–327.
12. Bollerslev Tim, (1990), Modelling the Coherence in Short-Run Nominal Exchange Rates: A Multivariate Generalized ARCH model, The Review of Economics and Statistics, Vol. 72, No. 3, pp. 498-505.
13. Bollerslev Tim, (2010), Glossary to ARCH (GARCH), Volatility and Time Series Econometrics: Essays in Honor of Robert F. Engle, Chapter 8, pp.137-163. Oxford University Press.
14. Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Rechty, Oriol Vinyals, 2017, Understanding Deep Learning requires rethinking generalization, Review as a conference paper at ICLR 2017, https://openreview.net/pdf?id=Sy8gdB9xx

15. Christopher Olah, 2014, Deep Learning, NLP, and Representations, http://colah.github.io/posts/2014-07-NLP-RNNs-Representations/
16. Christopher Olah, 2015, Understanding LSTM Networks, http://colah.github.io/posts/2015-08-Understanding-LSTMs/
17. Chung Junyoung, Gulcehre Caglar, Cho KyungHyun, Bengio Yoshua, 2014, Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling, https://arxiv.org/abs/1412.3555
18. Cristina Scheau, 2016, Regularization in deep learning, https://chatbotslife.com/regularization-in-deep-learning-f649a45d6e0#.q8mm0d6xk
19. Degustis Augustas, Novickyte Lina, 2014, The efficient market hypothesis: a critical review of the literature and methodology, Ekonomika, Vol 93(2), pp. 7 -23. http://www.zurnalai.vu.lt/ekonomika/article/download/3549/2564
20. Denny Britz, WildML, http://www.wildml.com/2015/10/recurrent-neural-network-tutorial-part-4-implementing-a-grulstm-rnn-with-python-and-theano/
21. Diederik P Kingma, Ba Jimmy Lei, 2014, Adam: A method for stochastic optimization, ILCR, https://arxiv.org/pdf/1412.6980v8.pdf
22. Dozat Timothy, 2015, Incorporating Nesterov Momentum into ADAM, http://cs229.stanford.edu/proj2015/054_report.pdf
23. Engle Robert F., (1982), Autoregressive Conditional Hetersoscedasticity with Estimates of Variance of United Kingdom Inflation, Econometrica, Vol.50, No.40, pp.987-1007.
24. Erkam Guresen, Gulgun Kayakutlu, Tugrul U.Daim, 2011, Usin artificial neural network models in stock market index prediction, Expert Systems with Applications, Vol. 38, pp. 10389-10397
25. François Chollet, Keras: Deep Learning library for Theano and TensorFlow, https://keras.io/, https://github.com/fchollet/keras
26. Gal Yarin, Ghahramani Zoubin, 2016, A Theoretically Grounded Application of Dropout in Recurrent Neural Networks, University of Cambridge. https://arxiv.org/pdf/1512.05287v5.pdf
27. Gers Felix A, Schmidhuber Jurgen, Fred Cummins, 2000, Learning to Forget: Continual Prediction with LSTM, Neural Computation, Vol. 12, No. 10 , pp. 2451-2471
28. Glosten L.R., Jagannathan R., Runkle D., (1993), On the Relation Between the Expected Value and the Volatility of the Nominal Excess Return on Stocks, Journal of Finance, Vol. 48, pp. 1779-1801.
29. Graves Alex, and Jaitly Navdeep, 2014, Towards end-to-end speech recognition with recurrent neural networks, Proceedings of the 31st International Conference on Machine Learning, Beijing, China, JMLR: W&CP volume 32.
30. Graves Alex, 2012, Supervised Sequence Labelling with Recurrent Neural Networks. Textbook, Studies in Computational Intelligence, Springer http://www.cs.toronto.edu/~graves/preprint.pdf
31. Graves Alex, 2013, Generating Sequences With Recurrent Neural Networks, https://arxiv.org/abs/1308.0850
32. Graves Alex, Mohamed Abdel-rahman, Hinton Geoffrey, 2013, Speech recognition with deep recurrent neural networks, Department of Computer Science, University of Toronto, https://www.cs.toronto.edu/~fritz/absps/RNN13.pdf
33. Ian Goodfellow, Yoshua Bengio, Aaron Courville (2016), Deep Learning, The MIT Press
34. Jason Brownlee, Machine Learning Mastery, http://machinelearningmastery.com/
35. John Duchi, Elad Hazan, Yoran Singer, 2011, Adaptative Subgradient Methods for Online Learning and Stockastic Optimization, Journal of Machine Learning Researh, Vol.12, pp. 2121-2159 http://www.jmlr.org/papers/volume12/duchi11a/duchi11a.pdf
36. Keras Optimizers, https://github.com/fchollet/keras/blob/master/keras/optimizers.py#L114

37. Klaus Greff, Rupert Kumar Srivastava, Jan Koutnik, Bas R. Steunebrink, Jurgen Schmidhuber, 2015, LSTM: A Search Space Odyssey, The Swiss AI LAB ISDIA, https://arxiv.org/pdf/1503.04069.pdf
38. Kyong Joo Oh, Kyoung-jae Kim, 2002, Analyzing stock market tick data using piecewise nonlinear model, Expert Systems with Applications, Vol 22, pp. 249-255
39. Kyoung-jae Kim, 2006, Artificial neural networks with evolutionary instance selection for finance forecasting, Expert System with Applications, Vol. 30, pp. 519-526.
40. Lo Andrew W., MacKinlay Craig, 1988, Stock Market Prices do not Follow Random Walks: Evidence from a Simple Specification Test, The Review of Financial Studies, Vol.1, No. 1, pp. 41-66
41. Lo Andrew W., MacKinlay Craig, 1999, A Non-Random Walk Down Wall Street, Princeton University Press.
42. LSTM Networks for Sentiment Analysis, DeepLearning 0.1 documentation, http://deeplearning.net/tutorial/lstm.html#code-citations-contact
43. Luckyson Khaidem, Snehanshu Saha, Sudeepa Roy Dey, 2016, Predicting the direction of stock market prices using random forest, Applied Mathematical Finance, https://arxiv.org/pdf/1605.00003.pdf
44. Malkiel Burton G., 2003, The Efficient Market Hypothesis and Its Critics, CEPS Working Paper no.91
45. Malkiel Burton G., 2016, A Random Walk Down Wall Street: The Time-tested Strategy for Successful Investing, W.W. Norton & Company, New York
46. Matthew Dixon, Diego Klabjan, Jin Hoon Bang, 2016, Classification-based Financial Merkets Prediction using Deep Neural Networks, https://arxiv.org/pdf/1603.08604.pdf
47. Mingyue Qiu,Yu Song, Fumio Akagi, 2016, Application of artificial neural network for the prediction of stock market returns: The case of the Japanese stock market, Chaos, Solitons and Fractals, Vol. 85, pp.1-7
48. Moghaddam Amin Hedayati, Moghaddam Moein Hedayati, Esfandyari Morteza, 2016, Stock market index prediction using artificial neural network, Journal of Economics, Finance and Administrative Science, Vol.21, pp. 89 - 93.
49. Nikil Buduma, 2015, A deep dive into Recurrent Neural Nets, http://nikhilbuduma.com/ 2015/01/11/a-deep-dive-into-recurrent-neural-networks/
50. Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, Ruslan Salakhutdinov, 2014, Dropout: A Simple Way to Prevent Neural Networks from Overfitting, Journal of Machine Learning Research, No. 15, pp. 1929-1958, https://www.cs.toronto.edu/~hinton/absps/JMLRdropout.pdf
51. Paul D. McNelis, 2005, Neural Networks in Finance - Gaining Predictive Edge in the Market, Elsevier Academic Press.
52. Perez Carlos E., 2016, Deep Learning Machines are Holographic Memories, Deep Learning Patterns, Methodology and Strategy @ IntuitionMachine.com, https://medium.com/intuitionmachine/deep-learning-machines-are-holographic-memories-258272422995#.c2p2z1iue
53. R.J.Kuo, C.H.Chen, Y.C.Hwang, 2001, An intelligent stock trading decision support system throuh interaction of genetic algorithm based fuzzy neural network and artificial neural network, Fuzzy Sets and Systems, Vol. 118, pp. 21-45
54. Razvan Pascanu, Caglar Gulcehre, Kyunghyun Cho, Yoshua Bengio, 2014, How to Construct Deep Recurrent Neural Networks, ICLR 2014, https://arxiv.org/abs/1312.6026
55. Robert R. Trippi, Efraim Turban, 1996, Neural Networks in Finance and Investing, IRWIN Professional Publishing

56. Rumelhart David E., Hinton Geoffrey E., Williams Ronald J., (1986), Learning representations by back-propagating errors. Nature, Vol. 323, pp. 533–536, https://www.iro.umontreal.ca/~vincentp/ift3395/lectures/backprop_old.pdf
57. Sebastian Ruder, 2016, An overview of gradient descent optimization algorithms, http://sebastianruder.com/optimizing-gradient-descent/
58. Sepp Hochreiter and Jürgen Schmidhuber, (1997), Long Short-Term Memory, Neural Computation, Vol. 9, Issue 8, pp. 1735-1780.
59. Sutskever Ilya, Martens James, Dahl George, Hinton Geoffrey, On the importance of initialization and momentum in deep learning, http://www.cs.toronto.edu/~fritz/absps/momentum.pdf
60. Taylor Stephen J., (1986), Modelling Financial Time Series, Wiley.
61. Terasvirta Timo, (2006), An Introduction to Univariate GARCH Models, SSE/EFI Working Papers in Economics and Finance, No. 646.
62. Theano Development Team. "Theano: A Python framework for fast computation of mathematical expressions" http://deeplearning.net/software/theano/
63. Tijmen Tieleman, Hinton Geoffrey, 2012, Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. COURSERA: Neural Networks for Machine Learning http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf
64. Todea Alexandru, Plesoianu Anita, 2011, Testing the Hypothesis of Martingale on Intraday Data: the case of BET Index, Theoretical and Applied Economics, vol. 5(558)(supplement), issue 5(558)(supplement), pages 344-351, http://store.ectap.ro/suplimente/Conferinta%20FABBV%202010_engleza.pdf
65. Victor Dragota, Dragos Stefan Oprea, 2014, Informationa Efficiency tests on the Romanian Stock Market: Review of the literature, The Review of Finance and Banking, Volume 06, Issue 1, pp. 15-28. http://www.rfb.ase.ro/articole/ARTICOL_II_nr6.pdf
66. Wun-Hua Chen, Jen-Ying Shih and Soushan Wu, (2006), Comparison of support – vector machines and back propagation neural networks in forecasting the six major Asian stock markets., Int. J. Electronics Finance, Vol. 1, No. 1, pp. 49-67.
67. Zeiler Matthew D., 2012, ADADELTA: An Adaptative Learning Rate Method, https://arxiv.org/pdf/1212.5701v1.pdf
68. Zhang Yudong, Wu Lenan, 2009, Stock market prediction of S&P 500 via combination of improved BCO approach and BP neural network, Expert Systems with Applications, No.36, pp.8849-8854