Intelligent Information Management, 2014, 6, 45-54
Published Online March 2014 in SciRes. http://www.scirp.org/journal/iim
http://dx.doi.org/10.4236/iim.2014.62007

Scientific Research

Evolutionary Algorithm Based Approach for Modeling Autonomously Trading Agents

Anil Yaman, Stephen Lucci, Izidor Gertner

Department of Computer Science, The City College of New York, New York, USA
Email: anilyaman00@gmail.com, lucci@cs.ccny.cuny.edu, csicg@cs.ccny.cuny.edu

Received 12 February 2014; revised 5 March 2014; accepted 24 March 2014

Abstract

The autonomously trading agents described in this paper produce a decision to act such as: buy, sell or hold, based on the input data. In this work, we have simulated autonomously trading agents using the Echo State Network (ESNs) model. We generate a collection of trading agents that use different trading strategies using Evolutionary Programming (EP). The agents are tested on EUR/USD real market data. The main goal of this study is to test the overall performance of this collection of agents when they are active simultaneously. Simulation results show that using different agents concurrently outperform a single agent acting alone.

Keywords

Artificial Intelligence; Autonomous Agents; Artificial Life; Evolutionary Computation; Neural Networks; FOREX

1. Introduction

Foreign Exchange Market (FOREX) is a complex adaptive system involving a large number of dynamic networks of interacting agents. It reflects the collective behavior of participants whose behaviors evolve and self-organize corresponding to a series of historical events. Due to this complexity, the problem of predicting the future price is one of the most challenging problems for the machine learning community. There are two theories suggested concerning the viability of the predictability of the future price of a financial instrument. Efficient Market Hypothesis (EMH) states that no information can be accountable since the price is immediately absorbed by all the new information [1]. Random Walk Theory (RWT) states that the past price cannot be used to predict the price in the future, because the price movement is random and completely independent from the past [2]. These theories offer a challenge to researchers who try to extract features and make predictions. Recent developments in computational tools allow researchers to design algorithms that can trade online. Another question

arises, which is whether a computer can do a better job in predicting prices. As a result, researchers have started to apply machine learning algorithms and test the efficiency of these algorithms.

Artificial Neural Networks (ANNs) are often-used tools in prediction algorithms [3]-[6]. These algorithms usually use fundamental and/or technical analysis as inputs and produce the prediction of the price value as output. The networks are trained using the back-propagation algorithm (BP) [7] [8]. However, BP may get stuck in a local minimum due to its use of gradient descent information. Additionally, the error function should be differentiable. Autonomously trading neural network based agents on the other hand, are simulations of real world traders with limited capabilities [5] [9]-[11]. They produce action outputs such as: buy, sell or hold based on the stream of input data. In this framework, evolutionary algorithms are more suitable than BP since the error function is not differentiable.

In this work, we used Echo State Networks (ESNs) in the decision mechanisms of the autonomous agents. We evolved the network weights using Evolutionary Programming (EP). The evolutionary algorithm can provide a variety of agents that use different trading strategies. The problem lies in the selection of the agent that is likely to be successful on the test data. This is a difficult decision because even the best agent may often be wrong. Intuitively speaking, simultaneous activation of a variety of successful agents may reduce the risk and outperform the best agent. Therefore, we tested the performance of a collection of successful agents that are selected from the agent pool which is generated as a result of many evolutionary runs.

In Section 2, the Foreign Exchange Market and the methods used in prediction algorithms are briefly discussed. In Section 3, a detailed description of Echo State Networks is given. The evolutionary approach for optimizing neural networks is discussed in Section 4. A detailed description of the algorithm implemented is provided in Section 5 and the results are discussed in Section 6.

2. Overview of Foreign Exchange Market

Foreign Exchange Market (FOREX or FX) is a global and decentralized financial market where the traders exchange currency pairs in order to make profits. Trading is open 5 days a week, 24 hours a day. According to the Bank for International Settlements, daily average turnover is 4 trillion US dollars.

There are two main approaches used in market analysis. The first one is Fundamental Analysis which deals with the factors that affect supply and demand. The main goal in fundamental analysis is to gather and interpret information in order to acquire intuition on the future performance of a business. In FOREX, traders interpret the overall state of the economy as well as other factors such as: gross domestic product (GDP), interest rates, employment, earnings, housing, production and manufacturing. All this information is released periodically. When an event occurs, the traders look for trading opportunities and try to act before anyone else does. Technical analysis on the other hand, is based on the hypothesis that the factors that affect a financial instrument are embedded into the price. It is believed that these factors exhibit patterns. And these patterns may repeat. Thus, the past price trends and patterns can be studied and used to make predictions.

3. Echo State Network Model

Echo State Networks area type of recurrent network consisting of three layers: the input, hidden and the output layer. The connections between layers are represented as weight matrix notations: $\boldsymbol{W}_{\text{in}}$, $\boldsymbol{W}_{\text{hidden}}$, $\boldsymbol{W}_{\text{out}}$ and $\boldsymbol{W}_{\text{back}}$. The weight matrix $\boldsymbol{W}_{\text{in}}$ represents the connections from the input layer to the hidden layer; the weight matrix $\boldsymbol{W}_{\text{hidden}}$ denotes the internal connections; the weight matrix $\boldsymbol{W}_{\text{out}}$ denotes the concatenation of the connections from the input and the hidden layer to the output layer; and the weight matrix W_{back} represents the feedback connections which are from the output layer to the hidden layer.

The activation of the hidden nodes A^{t}_{hidden} at time t, can be computed by using the formula (3.1) [12]-[14].

$$\boldsymbol{A}^{t+1}_{\text{hidden}} = f_{\text{hidden}}\left(W_{\text{in}}\left(\boldsymbol{A}^{t+1}_{\text{in}}\right)+W_{\text{hidden}}\left(\boldsymbol{A}^{t}_{\text{hidden}}\right)+W_{\text{back}}\left(\boldsymbol{A}^{t}_{\text{out}}\right)\right) \tag{3.1}$$

where f_{hidden} is the activation function of the hidden nodes. The activation of output nodes can be computed as:

$$\boldsymbol{A}_{\text{concat}} = \left(\boldsymbol{A}^{t+1}_{\text{in}}, \boldsymbol{A}^{t+1}_{\text{hidden}}, \boldsymbol{A}^{t}_{\text{out}}\right) \tag{3.2}$$

$$\boldsymbol{A}^{t+1}_{\text{out}} = f_{\text{out}}\left(W_{\text{out}}\left(\boldsymbol{A}^{t+1}_{\text{in}}, \boldsymbol{A}^{t+1}_{\text{hidden}}, \boldsymbol{A}^{t}_{\text{out}}\right)\right) \tag{3.3}$$

where f_{out} is the activation function of the output nodes and $\boldsymbol{A}_{concat}$ is the concatenation of the activations of the input and hidden nodes at time $t + 1$ and the output nodes at time t [13] [14].

One of the important properties of ESNs is that only the connections to the output layer ($\boldsymbol{W}_{out}$) are trained. The rest of the connections are fixed; and are initialized before the training process. The initialization is performed by ensuring that the echo state property exists. This is achieved by sparsely connecting the hidden layer and assigning random weights to them; and then scaling the hidden connections to have spectral radius less than one [13] [14].

$$W_{\text{hidden}} = \frac{\alpha W'_{\text{hidden}}}{|\lambda_{\max}|} \tag{3.4}$$

where α is the spectral radius ($0 < \alpha < 1$) and λ_{max} is the maximum eigenvalue of the initial weight matrix [14]. The spectral radius can be chosen to be between 0 and 1. The closest the spectral ratio to one, the longer the memory persists in the network.

The architecture of an ESN is shown in **Figure 1**. All the possible connections between layers are illustrated (also recurrent connections of the output layer can be included), however all of the connections are not required. The fixed connections are displayed using solid lines, and trained connections are shown as dotted lines. The context layer is a representation of the activations of the units in the hidden layer at time t – 1. In each time step t, the activations of the nodes in the hidden layer are copied to the context layer.

4. Evolving Neural Network Model

The design of ANN based autonomous agents is a complex task that involves much time and effort. Different network structures and neuron wiring diagrams may exhibit completely different responses. Therefore, trial and error becomes an important part of the process of finding the correct settings of the network that is used in the design of an agent. Evolutionary algorithms [15]-[20] are computational models of Darwinian evolution [21]. They have often been used to find optimum solutions to the problems that have these kinds of design difficulties [22]-[24].Therefore, a number of researchers suggested that it might be advantageous to use them in the design process of neural networks [5] [25]-[37].

Evolutionary algorithms can be used to evolve both the topology and connection weights of a network. The representation of the topology and/or connection weights is called a genotype of an individual. The neural networks are decoded by mapping the genotypes to their phenotypes. There are a number of methods in the literature which can be classified into two groups: direct encoding and indirect encoding [30] [38].

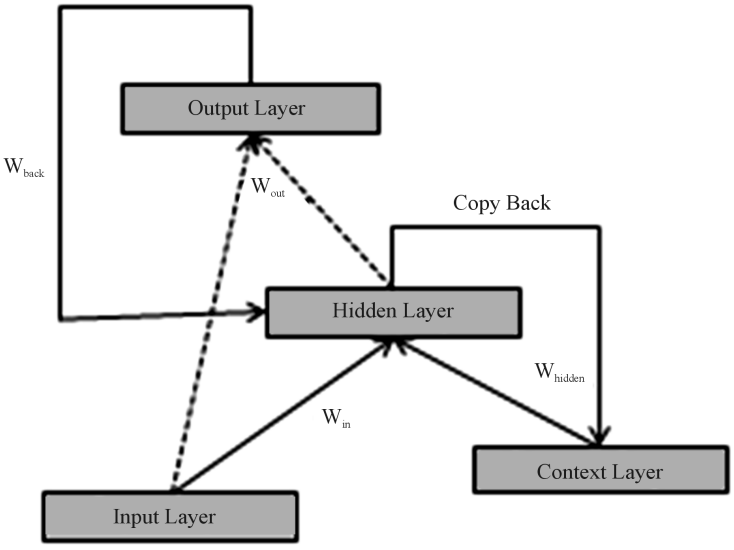

Figure 1. The architecture of an ESN with all connections. The connections that are trained are shown as dotted lines; connections that are fixed are depicted as solid lines. (Redrawn from Tong *et al*., 2007 [14]).

In direct encoding there is a one-to-one correspondence between the genotype and the parameters of the phenotype. Usually, this method is used to evolve only the parameters of the network [27] [34] [37] [39]. Whereas in indirect encoding, the mapping between the genotype and the weights (and the topology) of the networks is described as a function, rules or a growth model [35] [40]. Selection of the evolution method depends on the problem type. Using direct encoding on large neural networks might yield bad convergence performance since there is a large amount of connection weights to be optimized.

In echo state networks, the inputs, the recurrent and the feedback connections are independent of the problem since they are initialized randomly before the training process begins. Therefore, the number of weights to be optimized is minimal. It is thereby possible to explore large network structures on complex problems without worrying about the convergence problem of evolutionary algorithms.

The structure of the echo state network used in this work consists of 25 input nodes, 50 hidden nodes and 1 output node. In this case, the connection weights to be optimized can be represented as a 1 by 75 real-valued vector (concatenation of the connections from the input layer to the output layer and from the hidden layer to the output layer 25 + 50). We used evolutionary programming to evolve these 75 real-valued connection weights by directly mapping to the representation of the individuals (objective vectors) in the population.

Evolutionary Programming (EP) was first developed by Fogel, Owens and Walsh [41], and used to evolve finite state machines as predictors. The state transition tables of these finite state machines were modified by random mutations. Later, this model was extended by Fogel [42] to operate on real-valued vectors.

In EP, the individuals in the population consist of two components: an objective vector and a variance (self-adaptation) vector. Only the objective vectors are evaluated, however, both components are subject to evolution. The evolution of the variance vector allows the variance parameters to adapt during the search process. This is called self-adaptation.

Only the mutation operator is used in EP. The mutation operator is applied by first mutating the objective vector using a Gaussian distribution which has an independent variance for each component of the objective vector [43]. The variance vector is then updated.

$$x_i' = x_i + \sqrt{\mathrm{var}_i} \cdot N_i(0,1) \tag{4.1}$$

$$var_i' = var_i + \sqrt{\alpha \cdot var_i} \cdot N_i(0,1) \tag{4.2}$$

where $X = \{x_1, x_2, \cdots, x_n\}$ is the objective vector, $V = \{var_1, var_2, \cdots, var_n\}$ is the variance vector. The parameter α ensures that var_i remains positive [43] [44].

We use the plus selection ($\mu + \mu$) method. In plus selection, μ number of offspring is generated from μ individuals. Then, the individuals in the whole population ($\mu + \mu$) are ordered according to their fitness values, and the best μ individuals are selected. In this selection mechanism, parents can survive until offspring become more adapted [45].

5. The Algorithm to Create a Collection of Autonomously Trading Agents

The evolutionary programming approach is a basis to create autonomously trading agents. Its fitness function is designed to optimally select agents. In this section we describe this process in detail.

The agents' actions are based on the output of the echo state network as described in section 4. The connection weights from the input layer to the output layer and from the hidden layer to the output layer ($\boldsymbol{W}_{\text{out}}$) were optimized. The feedback connections ($\boldsymbol{W}_{\text{back}}$) are not used. The rest of the connections ($\boldsymbol{W}_{\text{hidden}}$ and $\boldsymbol{W}_{\text{in}}$) are fixed. The weights of the connections from the input layer to the hidden layer ($\boldsymbol{W}_{\text{in}}$) were sampled from a uniform distribution [−1, 1] with a probability of 0.2. The internal weights ($\boldsymbol{W}_{\text{hidden}}$) were randomly set to the values of 0, −1 and +1 with probabilities of 0.95, 0.025 and 0.025 respectively. They were scaled to have a 0.98 spectral radius according to the formula 3.4. The spectral radius is close to one; this means that the long term memory decay rate is small [13].

The ESN used in this work consists of 25 input neurons. The inputs to these neurons are currency pairs, technical indicators [46], an agent's status flag, an agent's return and a bias. These inputs are: EUR/USD, USD/JPY, GBP/USD, USD/CHF, AUD/USD, USD/CAD, NZD/USD, EUR/GBP, EUR/JPY, EUR/AUD, EUR/CAD, EUR/NZD. Inputs from 13 to 22 are 5 technical indicators (Stochastic Slow (%K, %D), RSI, MACD (MACD, Signal, Histogram), ATR, ADX (ADX, DI+, DI-)).

In order for an agent to sense its current status, two additional inputs are included. The first one is an agent's

status flag which indicates if an agent is currently trading. It becomes 1 if the agent is currently in a long (buy) trade, −1 if the agent is currently in a short (sell) trade, and 0 if the agent is not trading. The second one is the agent's return which indicates the price change since the beginning of a trade. It was calculated using the formula below.

$$\Delta_{\text{price}} = \left(\text{price}_{\text{now}} - \text{price}_{\text{open}}\right) \Big/ \text{price}_{\text{open}} \tag{5.1}$$

where Δ_{price} is the price percentage change and $price_{\text{open}}$ is the opening price of a position. The last input is the bias and it is fixed to be 1.

The output of the networks is a single sigmoidal (tanh) neuron. If the output becomes greater than 0.5, the agent goes long (buy). If the output becomes smaller than −0.5, the agent goes short (sell), and finally when it is between −0.5 and 0.5, the agent closes the position, if it has one, no action is performed.

The network contains 50 hidden units. The activation function of all the hidden units is the tanh activation function. As discussed earlier, all the connections except $\boldsymbol{W}_{\text{out}}$ are fixed. It means that the connections between the input layer and the output layer, and the connections between the hidden layer and the output layer should be optimized. The total count of connections to be optimized is $1x(25 + 50)$ where 25 is the number of input units, 50 is the number of hidden units and 1 is the number of output units.

The agents were trained and tested on EUR/USD hourly data. A sliding window approach, illustrated in **Figure 2**, was used. The agents were first trained on the window Train1 which consists of 900 samples, and then tested on the following 100 samples which is Test1. Then, the window was moved 100 samples and the new agents were trained on the windowTrain2 and tested on Test2. **Figure 2** depicts only the target vector which is 1 × 1000 (together with the training and the test data) samples per window. The input data on the other hand, is a matrix 25 × 1000 for each window. Each row in the input data was normalized to have 0 mean and 1 standard deviation; and normalized input sequences then scaled in the range between −1 and 1.

The beginning date of the training data is 01/01/2013 11:00 pm. The testing starts from the date 02/22/2013 01:00 am and continues until the date 12/31/2013 00:00 am by moving 100 samples in each step.

The population size was set to 5. The mutation operator produces 5 offspring from the 5 selected parents. This is called $(\mu + \mu)$ selection and more information can be found in section 4.

The *agents* = $\{(x_1, v_1), (x_2, v_2), \ldots (x_n, v_n)\}$ consist of the objective vectors $x_1, x_2, \ldots x_n$, and the variance vectors $v_1, v_2, \ldots v_n$ where n is the population size. The objective vectors and the variance vectors of an individual in the population are 75 dimensional real-valued vectors and, they are direct representations of the output connection weights to be optimized ($\boldsymbol{W}_{\text{out}}$). In this case, $\boldsymbol{W}_{\text{out}}$ the objective vectors are constructed by concatenating the

Figure 2. Sliding window approach. First, the agents are trained on 900 samples; tested on the following 100 samples and then the window is moved by 100 samples.

weights from the input nodes to the output node and the weights from the hidden nodes to the output node.

1) Randomly initialize a population of agents (the objective and variance vectors).

2) Repeat until no improvement is observed within a certain number of iterations.

a) Decode each objective vector in the current generation and construct a corresponding ESN.

b) Evaluate each agent in the population by letting them trade starting from the first sample of the training data to the last sample of the training data.

c) Find the fitness of each agent at the end of the trading session. (The fitness value of an agent is calculated as the total gain divided by the absolute value of the total loss).

d) Select the best μ agents according to their fitness values.

e) Apply the mutation operator to the selected agents and produce μ new individuals.

f) Self-adapt the variance vectors of the new individuals.

The evaluation starts from the first sample of the training data, and moves forward one step at a time. In each time step, it presents the input data to an agent. The agent takes the input and calculates the output of the network. The agent can only perform one action at a time according to the output (output > 0.5, buy; output < −0.5, sell; $-0.5 \leq$ output ≤ 0.5, hold or close the trade). A flat spread fee ($3 per trade) is added to the price each time an agent performs a buy or sell action [5].

At the end of the training data, the evaluation function returns the performance values P = [Total Return, Total Gain, Total Loss, Total Trade, Success Trade] of an agent; and the ratio Total Gain/Total Loss is used for the selection operator.

The variables: Total Return, Total Gain and Total Loss are calculated using the raw price. Later, they are converted to the pip values. A pip (percentage in point) is a unit change in the price of a currency pair. Since, we are working with EUR/USD currency pairs, the unit change is 0.0001; and we take the unit change as the worth of a dollar.

When the training process is complete, all the variables are set to the default values, and the agents are tested on the following 100 samples. The testing process is the same as the evaluation process, but this time the evaluation function takes the test data as an argument and returns the performance values of the agents on the test data. After an evolutionary run, we obtain the performance values of 5 agents. Since we are maximizing the function Total Gain/Total Loss, these 5 agents may have become stuck at a local maximum. That is the reason that, the algorithm is run 10 times, and each time, the resulting agents are added into the agent pool. After this process, we obtained 50 agents in the agent pool. Among these 50 agents, we selected the best 5 agents according to their Succes Trade/Total Trade ratio on the training data. The results provided here are the simultaneous activation of this collection of the 5 agents selected for each time window. The steps of the algorithm are illustrated in **Figure 3**.

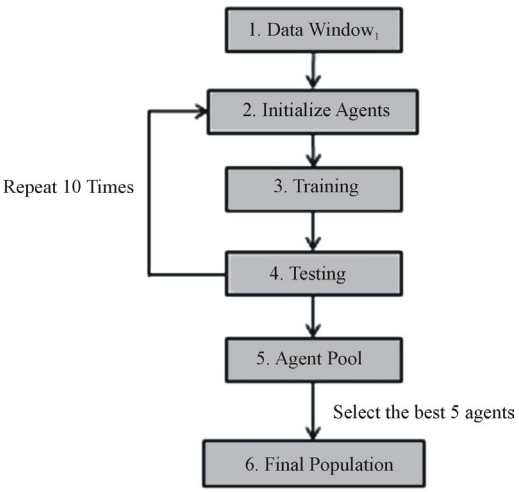

Figure 3. The steps of the algorithm. For a given time window, the evolutionary algorithm runs 10 times and generates 50 agents. The best 5 agents are selected among these agents.

The results were generated for the 54 time windows between the dates 02/22/2013 01:00 am and 12/23/2013 01:00 pm. The statistics of the all agents generated within the 54 time windows are summarized in **Table 1**, under the column header **TW54**. The row headers: **Successful Agent**, **Unsuccessful Agent** and **Neutral Agent** are the numbers of the agents whose returns were positive, negative and zero on the test data respectively. **MaxReturn** and **MinReturn** are the maximum and the minimum returns observed among all the agents on the test data. **Ave.Profit** and **Ave.Loss** are the average of all the profits gained and the average of all the losses incurred by all the agents on the test data. **Ave.Return** is the average of all the returns of the agents and **TotalReturn** is the sum of all returns of the agents. The return, profit and loss are US dollar based.

If we ignore the neutral agents, the percentage of the successful agents in the population is 0.52. The average profit is + 1.43 which is a good result. In the long run the system is not losing money. However, neither is it gaining. The return over time is shown in **Figure 4**.

Intuitively speaking, after a long up or down trend the price may change its direction and move in the opposite direction. In this case, agents may perform badly since they have been trained on trending prices in only one direction. To avoid these situations, the input windows were filtered when the price change between the starting and end points of the training data is large.

1) Start from the first sample $t_1 = 1$ and check the change in the price between the starting point t_1 and the end point $t_2 = t_1 + 899$ (change in price measured using equation 5.1).

Table 1. The statistics of all the agents generated. TW54 is the results of the standard algorithm, and TW47 is the results of the modified algorithm.

	TW54	TW47	TW47-1
Total Agent	270	235	47
Successful Agent	91	91	19
Unsuccessful Agent	83	74	14
Neutral Agent	96	70	14
MaxReturn	280	276	220
Ave.Profit	76	70	58
MinReturn	−269	−242	−242
Ave.Loss	−79	−67	−71
Ave.Return	1.43	5.95	2.35
TotalReturn	388	1399	111

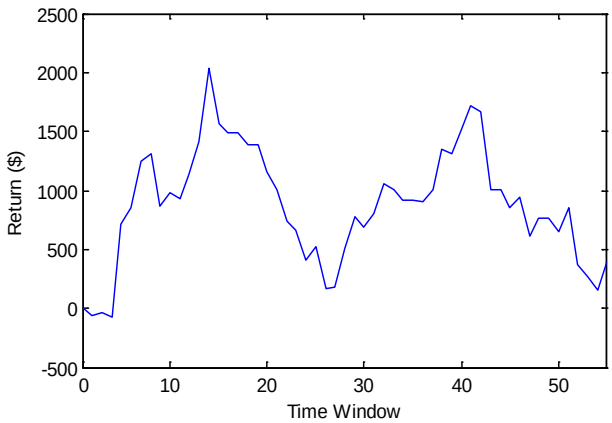

Figure 4. The cumulative return of the agents that are generated using the standard algorithm.

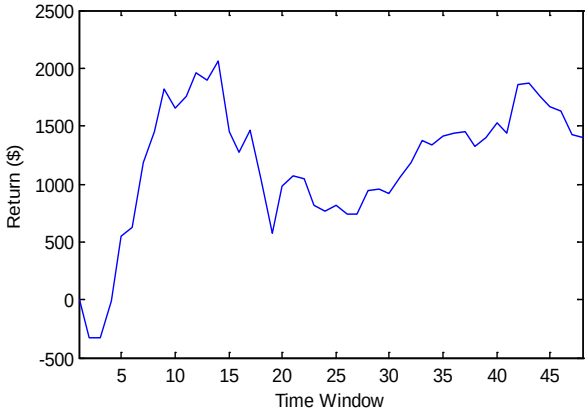

Figure 5. The cumulative return of the agents that are generated using the modified algorithm.

2) If the change is greater than 0.025, increment t_1 by one and go o step 1.
a) Train the agents on the time window between t_1 and t_2.
b) Test the agents between the time window t_2 and $t_2 + 100$.
c) Add the agents into the agent pool.
d) Increment t_1 by 100.
Go to step 1.

After this modification, 47 time windows that satisfy the percentage change rule have been found. The statistics of the agents generated by the modified algorithm is provided in **Table 1** under the header **TW47**. When the results are compared with the results that generated by the standard algorithm, a significant improvement is seen. Even though there is not much improvement in the percentage of the successful agents in the population, the average return and total return are improved by % 316 and % 260 (calculated according to equation 5.1) respectively. As a result, the cumulative return over time is trending up as illustrated in **Figure 5**.

Finally, we compared the performance of the system when only the best agent is selected versus a collection of agents at the same time. Using the modified algorithm, for each window only the best agent is selected, and the statistics of these agents are given in **Table 1** under the column header **TW47-1**. The results show that the performance of the simultaneous activation of a collection of autonomous agents that are generated by the modified algorithm is superior.

6. Conclusions

In this work, we design a system that generates autonomously trading agents. Echo State Networks are used in the decision mechanisms of these agents. The major 12 currency pairs and popular 5 technical indicators are used for predicting the optimum actions for EUR/USD currency pairs within a given time window. The connection weights of the output layers of the networks are trained using evolutionary programming.

We test whether the performance of many agents will outperform the performance of a single agent. Our results show that the total return and the average trade are better when multiple agents are used. Furthermore, a significant improvement is seen when input windows are filtered according to the price percentage.

In conclusion, the methodology described here is not limited to the financial arena, and has widespread applicability.

References

[1] Fama, E.F. (1970) Efficient Capital Markets: A Review of Theory and Empirical Work. *Journal of Finance*, **25**, 383-417.

[2] Malkiel, B.G. (1973) A Random Walk Down Wall Street. Norton, New York,

[3] Dutta, S. and Shekhar, S. (1888) Bond Rating: A Non-Conservative Application of Neural Networks. *IEEE International Conference on Neural Networks*, San Diego, 24-27 July 1998, 443-450.

[4] Senol, D. (2008) Prediction of Stock Price Direction by Artificial Neural Network Approach. Bogazici University, Is-

tanbul.

[5] Sher, G.I. (2011) Evolving Chart Pattern Sensitive Neural Network Based Forex Trading Agents. arXiv1111.5892S.

[6] White, H. (1988) Economic Prediction Using Neural Networks: The Case of IBM Daily Stock Returns. *IEEE International Conference on Neural Networks*, San Diago, 24-27 July 1998, 451-459.

[7] Rumelhart, D.E. and McClelland, J.L. (1986) Parallel Distributed Processing: Explorations in the Microstructure of Cognition: Foundations (Volume I). MIT Press, Cambridge.

[8] Rumelhart, D.E., Hinton, G.E. and Williams, R.J. (1986) Learning Internal Representations by Error Propagation. In: Rumelhart, D.E. and McClelland, J.L., Eds., *Parallel Distributed Processing*: *Exploration in the Microstructures of Cognition*, Vol. I, MIT Press, Cambridge, 318-362.

[9] Chan, N.T., LeBaron, B., Lo, A.W. and Poggio, T. (1999) Agent-Based Models of Financial Markets: A Comparison with Experimental Markets. MIT Artificial Markets Project, 124.

[10] Tesfatsion, L. (2002) Agent-Based Computational Economics: Growing Economies from the Bottom up. *Artificial Life*, **8**, 55-82. http://dx.doi.org/10.1162/106454602753694765

[11] Fukumoto, R. and Kita, H. (2001) A Multi-Objective Genetic Algorithm Approach to Construction of Trading Agents for Artificial Market Study. Springer, Berlin Heidelberg.

[12] Jaeger, H. (2002) A Tutorial on Training Recurrent Neural Networks. Covering BPTT, RTRL, EKF and the Echo State Network Approach. German National Research Center for Information Technology.

[13] Jaeger, H. (2001) The Echo State Approach to Analysing and Training Recurrent Neural Networks. German National Research Center for Information Technology.

[14] Tong, M.H., Bicket, A., Cristiansen, E. and Cottrell, G. (2007) Learning Grammatical Structure with Echo State Network. *Neural Networks*, **20**, 424-432. http://dx.doi.org/10.1016/j.neunet.2007.04.013

[15] Holland, J.H. (1975) Adaptation in Natural and Artificial Systems. University of Michigan Press, Ann Arbor.

[16] Goldberg, D.E. (1989) Genetic Algorithms in Search, Optimization and Machine Learning. Addison-Wesley, Boston.

[17] Koza, J.R. (1992) Genetic Programming: on the Programming of Computers by Means of Natural Selection. MIT Press, Cambridge.

[18] Schwefel, H. (1981) Numerical Optimization of Computer Models. John Wiley & Sons, Hoboken.

[19] Rechenberg, I. (1965) Cybernetic Solution Path of an Experimental Problem. Royal Aircraft Establishment, Translation 1122.

[20] Lucci, S. and Kopec, D. (2012) Artificial Intelligence in the 21st Century. Mercury Learning and Information.

[21] Darwin, C. (1859) On the Origin of Species. Kindle Edition (2012).

[22] Maslov, I. and Gertner, I. (2009) Evolutionary Algorithms in Digital Image Processing: A Hybrid Approach. LAP - Lambert Academic Publishing, OmniScriptum GmbH & Co. KG, Saarbrücken.

[23] Maslov, I. and Gertner, I. (2006) Multi-Sensor Fusion: An Evolutionary Algorithm Approach. *Information Fusion*, **7**, 304-330. http://dx.doi.org/10.1016/j.inffus.2005.01.001

[24] Maslov, I. and Gertner, I. (2007) Multi-Sensor Target Recognition in Image Response Space Using Evolutionary Algorithms. In: Sadjadi, Firooz, Javidi and Bahram, Eds., *Physics of Automatic Target Recognition*, Chapter 8, Springer, Berlin, 127-141.

[25] Angeline, P.J., Saunders, G.M. and Pollack, J.B. (1994) An Evolutionary Algorithm that Constructs Recurrent Neural Networks. *IEEE Transactions on Neural Networks*, **5**, 54-65. http://dx.doi.org/10.1109/72.265960

[26] Belew, R., McInerney, J. and Schraudolph, N.N. (1990) Evolving Networks: Using the Genetic Algorithm with Connectionist Learning. CSE Technical Report CS90-174, University of California, Berkeley.

[27] Caudell, T.P. and Dolan, C.P. (1989) Parametric Connectivity: Training of Constrained Networks Using Genetic Algorithms. *Proceedings of the 3rd International Conference on Genetic Algorithms*, Fairfax, June 1989, 370-374.

[28] Cliff, D., Harvey, I. and Husbands, P. (1993) Incremental Evolution of Neural Network Architectures for Adaptive Behaviour. *Proceedings of the* 1*st European Symposium on Artificial Neural Networks*, Brussels, 7-9 April 1993, 39-44.

[29] Floreano, D. and Mondada, F. (1994) Automatic Creation of an Autonomous Agent: Genetic Evolution of a Neural Network Driven Robot. 3*rd International Conference on Simulation of Adaptive Behavior* (*SAB*'1994), Brighton, 8-12 August 1994, 421-430.

[30] Floreano, D., Dürr, P. and Mattiussi, C. (2008) Neuroevolution: From Architectures to Learning. *Evolutionary Intelligence*, **1**, 47-62. http://dx.doi.org/10.1007/s12065-007-0002-4

[31] Igel, C. (2003) Neuroevolution for Reinforcement Learning Using Evolutionary Strategies. *Congress on Evolutionary*

Strategies, **4**, 2588-2595.

[32] Jung, J.-Y. (2007) Evolutionary Design of Artificial Neural Networks Using a Descriptive Encoding Language. Doctoral Dissertation, University of Maryland, College Park.

[33] Kenneth, O.S. and Miikkulainen, R. (2002) Efficient Evolution of Neural Network Topologies. Proceedings of the 2002 Congress on Evolutionary Computation (CEC'02). In: Langdon, W.B., Cantú-Paz, E., Mathias, K.E., Roy, R., Davis, D., Poli, R., Balakrishnan, K., Honavar, V., Rudolph, G., Wegener, J., Bull, L., Potter, M. A., Schultz, A.C., Miller, J.F., Burke, E.K. and Jonoska, N., Eds., *GECCO* 2002: *Proceedings of the Genetic and Evolutionary Computation Conference*, New York, 9-13 July 2002, 569-577.

[34] Montana, D. and Davis, L. (1989) Training Feedforward Neural Networks Using Genetic Algorithms. *Proceedings of the Eleventh Joint Conference on Artificial Intelligence*, **1**, 762-767.

[35] Nolfi, S., Miglino, O. and Parisi, D. (1994) Phenotypic Plasticity in Evolving Neural Networks. Proceedings of the International Conference from Perception to Action, Lausanne, 5-9 September 1994, 146-157. http://dx.doi.org/10.1109/FPA.1994.636092

[36] Saravanan, N. and Fogel, D.B. (1995) Evolving Neural Control Systems. *IEEE Expert*, **10**, 23-27. http://dx.doi.org/10.1109/64.393139

[37] Whitley, D., Starkweather, T. and Bogart, C. (1990) Genetic Algorithms and Neural Networks: Optimizaing Connections and Connectivity. *Parallel Computing*, **14**, 347-361. http://dx.doi.org/10.1016/0167-8191(90)90086-O

[38] Yao, X. (1999) Evolving Artificial Neural Networks. *Proceedings of the IEEE*, **87**, 1423-1447. http://dx.doi.org/10.1109/5.784219

[39] Wieland, A.P. (1990) Evolving Neural Network Controllers for Unstable Systems. *IEEE International Joint Conference on Neural Networks*, **II**, 667-673.

[40] Kitano, H. (1990) Designing Neural Networks by Genetic Algorithms Using Graph Generation System. *Complex Systems*, **4**, 461-476.

[41] Fogel, L.J., Owens, A.J. and Walsh, M.J. (1966) Artificial Intelligence through Simulated Evolution. John Wiley & Sons, Hoboken.

[42] Fogel, D.B. (1991) System Identification through Simulated Evolution: A Machine Learning Approach to Modeling. Ginn, Needham Heights.

[43] Bäck, T. and Schwefel, H.-P. (1993) An Overview of Evolutionary Algorithms for Parameter Optimization. *Evolutionary Computation*, **1**, 1-24. http://dx.doi.org/10.1162/evco.1993.1.1.1

[44] Fogel, D.B. (1992) Evolving Artificial Intelligence. Doctoral Dissertation, University of California, San Diego, La Jolla.

[45] de Castro, N.L. (2011) Fundamentals of Natural Computing. Chapman and Hall/CRC, Boca Raton.

[46] Pring, M.J. (1991) Technical Analysis Explained. McGraw-Hill, New York.