Collectively Intelligent Prediction in Evolutionary Multi-agent System

Joanna Kijak, Piotr Martyna, Aleksander Byrski,
Łukasz Faber, Kamil Piętak, and Marek Kisiel-Dorohinicki

***Abstract*—In the paper a summary of our previously realized and published work connected with constructing collective intelligent evolutionary multi-agent systems for time series prediction, based on multi-layered perceptrons is shown. Besides recalling our past papers, we describe the whole concept, present an implementation in a contemporary, component-oriented software framework AgE 3.0 and we conduct a number of experiments, finding different optimal parametrization for the considered instances of the problems (popular Mackey-Glass chaotic time series). The paper may be useful for a practitioner willing to use our meatheuristic algorithm (EMAS) along with the idea of collective agent-based system in order to realize prediction tasks.**

***Index Terms*—evolutionary neural networks, agent-based computing, time series prediction, collective intelligence, metaheuristic optimization**

I. Introduction

FOR a long period of time we authors have been tackling neural network optimization problem using an a evolutionary approach in agent-based environment. These approaches, involving the hybridization of EMAS (Evolutionary Multi-Agent System [9]) and neural-based collective intelligence, also involvement of immunological inspirations lead to obtaining interesting results, calling for further exploration and extension.

We have started with developing a hybrid of neural network based prediction system working as an ensemble in a multi agent system EMAS [9] where the agents contained genotypes describing the parameters of the networks, and the whole system worked as a prediction system, at the same time evolving the structure and parameters of the predictors [2], [7].

In most of our works we used Multi-Layered Perceptrons, keeping in mind that these neural networks are universal approximators and are able to learn any possible function, providing that appropriate configuration with regard to the number of neurons in the hidden layers and the parameters of the training are used [12]. Later we have also used Radial Basis Function Networks, being also universal approximator [5].

We have also designed an ensemble predictor according to PREMONN (PREdictive MOdular Neural Networks) defined by Petridis and Kehagias [17], treating our agents in EMAS as parts of collective intelligence applied to enhancing the parameters of neural networks, at the same time giving better and better predictions [3].

Finally we have also considered immunological inspirations in speeding-up the search for optimal neural network by prematurely discarding non-promising ones [6], [8].

We have also approached classification on a similar basis (modified PREMONN for ensemble creation) [4].

In this paper we would like to summarize the already realized research, to present a recently implemented collectively-intelligent prediction system and to discuss some extensively planned and realized experimental results.

The concept of the system relies heavily on using neural networks for prediction, leveraging their main advantage: the ability to learn from examples and generalize acquired knowledge to new cases in such a way that no explicit problem-dependent knowledge is needed. A proper design of neural network, may call for applying optimization techniques, such as evolutionary optimization, as still the expert needs to define network architecture, which should be suitable for the given problem. This requires carrying out numerous experiments, so it is a very time consuming job and can be performed only by the specialists.

When using metaheuristics, and in particular evolutionary computation on properly encoded network structure (the genotype can contain the parameters of the network training, not only its structure), the genotype is translated to phenotype by constructing and training the network on available data, checking the error measures. These parameters can be optimized using any general-purpose algorithm, such as evolutionary methods, including EMAS [9]. As EMAS is a quite complex system, and the configuration of the search has a significant number of degrees of freedom, it will be wise to use a dedicated, flexible, component-oriented framework to build the experimental environment. An appropriate candidate for this is AgE (a component-oriented computing framework developed at AGH University of Science and Technology, devoted to supporting development of population-based metaheuristics, i.a. agent-based ones), especially the recently released ver. 3.0 [18].

In the state of the art, agent-based approaches are naturally found in the cases when agent-based modelling of such phenomena as evacuation or disease spread are considered (see, e.g., [14]), based on such models of course certain predictions can be made. Particular ensemble predictors were also constructed and applied e.g. for weather forecasting [20], but without predictor optimization like the one presented in this paper (and a number of our previous work). Thus, the presented approach combining collective intelligence,

J. Kijak, P. Martyna, A. Byrski, Ł. Faber, K. Piętak and M. Kisiel-Dorohinicki are with Department of Computer Science, Faculty of Computer Science, Electronics and Telecommunications, AGH University of Science and Technology, Al. Mickiewicza 30, 30-059 Krakow, Poland (e-mails: {kijak.shia,pmartyna}@gmail.com, {olekb,faber,kpietak,doroh}@agh.edu.pl).

ISSN 2080-8755

prediction system and search for optimal parameters of the predictors is unique according to our best knowledge and it is hard to be compared to existing solutions. One possible aspect to be compared is the prediction ability, however this is not the main goal of the system—the main goal is combined prediction and search for optimal predictor, thus even a little bit worse results can be accepted, if leading to better architecture found. Even if no better architecture for particular neural network is found, the collective prediction can still prevail.

In the next section the state of the art regarding EMAS and EMAS-based prediction is shown, leading to discussion of multi-agent optimization and prediction system. Later the AgE 3.0 environment is sketched out, and a broad range of experiments are shown and discussed.

II. Neural EMAS for Time-series Prediction

The configuration of the agents in a predicting MAS (kind of specialization or method of cooperation) is often difficult to specify. What is more, when dynamic changes of the characteristics of the signal are possible, the configuration of the agents should reflect these changes, automatically adapting to the new characteristics. The mechanisms of evolution may help to transform the whole population of agents (by means of mutation and/or recombination) so as it fits best current profile of the input signal (proper selection/reproduction) – this evolutionary development of predicting MAS meets the general idea of an evolutionary agent system (EMAS).

A. Evolutionary Multi-Agent Systems

Following neodarwinian paradigms, two main components of the process of evolution are *inheritance* (with random changes of genetic information by means of mutation and recombination) and *selection*. They are realized by the phenomena of death and reproduction, which may be easily modeled as actions of *death* which results in the elimination of the agent from the system, and *reproduction* which is simply the production of a new agent from its parent(s) (see Fig. 1).

Selection is the most important and most difficult element of the model of evolution employed in *EMAS*. This is due to assumed lack of global knowledge (which makes it impossible to evaluate all individuals at the same time) and autonomy of agents (which causes reproduction to be achieved asynchronously). The proposed principle of selection corresponds to its natural prototype and is based on the existence of non-renewable resource, called *life energy*. The energy is gained and lost when the agent executes actions in the environment. An increase in energy is a reward for "good" behavior of the agent (e.g. attaining better solution of a certain task), a decrease – a penalty for "bad" behavior (which behavior is considered "good" or "bad" depends on the particular problem to be solved). At the same time the level of energy determines actions the agent is able to execute. In particular, low energy level should increase possibility of death and high energy level should increase possibility of reproduction [9, and other].

B. Time series prediction fundamentals

Prediction (or *forecasting*) is a generation of information about the possible future development of some process from data about its past and present behaviour [13]. A predicting system may be considered as a box with some input sequences, and predictions of successive values of (some of) these sequences as output. A neural network may be used as a mechanism to model the characteristics of a signal in a system for a time-series prediction [16]. The choice of a particular architecture of the network is to a large extent determined by a particular problem. Usually the next value of the series is predicted on the basis of a fixed number of the previous ones. Thus the number of input neurons correspond to the number of values the prediction is based on, and the output neuron(s) give prediction(s) of the next-to-come value(s) of the series. The multi-layer perceptron (MLP) in should predict t_{n+1} value

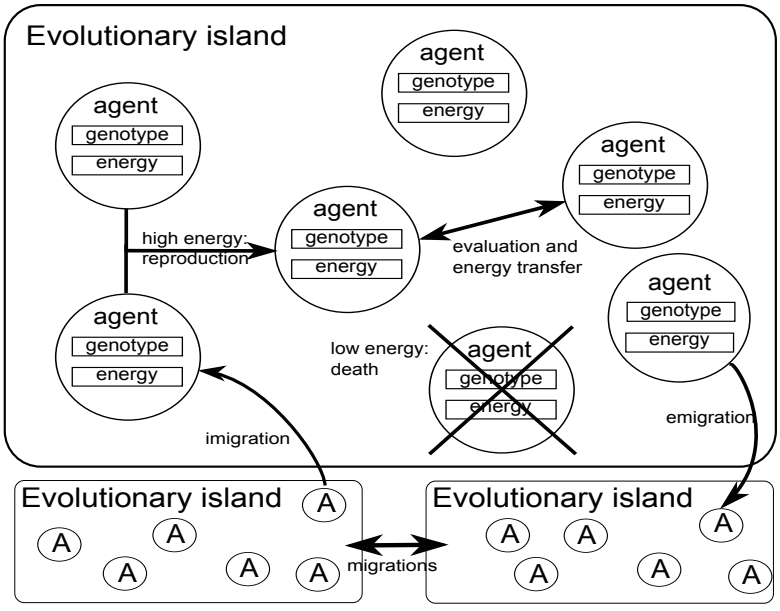


Fig. 1. Structure of EMAS

of the series, basing on some previous values, which are given on the inputs of the first layer. When t_{n+1} value is predicted, the inputs are shifted, and the value t_{n+1} is given as the input to the last neuron of the first layer.

A network may be supervisory trained, using the comparison between values predicted and received as an error measure.

C. Neural evolution in EMAS

In EMAS training of a neural network may be entrusted to an agent while the search for a suitable network architecture may be realized as the process of evolution occurring in the whole population. To achieve this goal, each agent simply possesses some vector of parameters, which describes the configuration of its neural network. This vector plays role of agent's genotype, and as such may be modified by genetic operators when inherited by its offspring. The evaluation of agents is based on the quality of prediction obtained from a trained network by means of gained/lost life energy.

One may notice that such system performs not only search for the optimal neural network structure, but also exhibits collective intelligence at the agent population level since agents are able to cooperate and provide even better solutions to the given problem.

D. A population of agents as a dynamic modular neural network

In the above-described system every agent contains a neural network, which acts as a computational model for the given (prediction) problem (see Fig. 2). Entrusting the task of solving the problem to the complex system, one may expect to obtain more accurate answers. This is similar to the approach of modular neural networks such as the model of PREMONN. PREMONN is a group (team) of neural networks, which solve the same problem, and their responses are combined together to yield the final result [17].

Applying PREMONN algorithm, every prediction of the given time-series may be assigned a certain probability, which can be used to determine the answer of the whole group of predicting individuals. After every prediction step, every individual based on its predictions and errors:

$$y_t^k = f_K(y_{t-1}, y_{t-2}, \ldots, y_{t-M})$$
$$e_t^k = y_t - \hat{y}_t^k$$

computes its credit function:

$$p_t^k = \frac{p_{t-1}^k \cdot e^{-\frac{|e_t^k|^2}{2\sigma^2}}}{\sum_{n=1}^K p_{t-1}^n \cdot e^{-\frac{|e_t^k|^2}{2\sigma^2}}}$$

Based on this function the response of the group of individuals can be a weighted combination of the answers or even can be the result of the winner-take-all combination:

1) Weighted combination:

$$\hat{y}_t = \sum_{k=1}^K p_{t-1}^k y_t^k$$

2) Winner-take-all combination:

$$\hat{y}_t = y^{\hat{z}_t}, \text{ where } \hat{z}_t = \arg\max_{k=1,2,\ldots,K} p_{t-1}^k$$

Using the above-mentioned approach a group of agents can produce the answer, which will be more accurate than the prediction of one arbitrarily chosen agent, as it comes from the group of agents using at least simple voting, or a more sophisticated bayesian stochastic scheme.

The system constructed in this way is also adaptive – its adaptation abilities base on its stochastic features. The probabilities of correct answers of agents are dynamically changed by the gating expert, so in the group of the agents, the answer of the whole group as the (somehow) weighted answer of every agent, is reliable, and the agents which produce worse answers should be replaced with new agents, in this way, globally, the system can adapt to the new features of the environment.

E. Evolving collective intelligence of agent populations

In such a complex organisation of agents, which was described above as a dynamically changing predictive modular neural network, it will be very difficult to determine correct values of additional parameters, which describe particular mechanisms of interaction. In order to improve the process of refining of such an organisation evolutionary processes may be also used (*meta-evolution*).

In an EMAS evolution is usually realized at the level of individual agent actions [7] (as each of them contains a chromosome which can be recombined, mutated, inherited etc.). At the same time perceiving the whole system as a group of modular neural networks (groups of agents, every

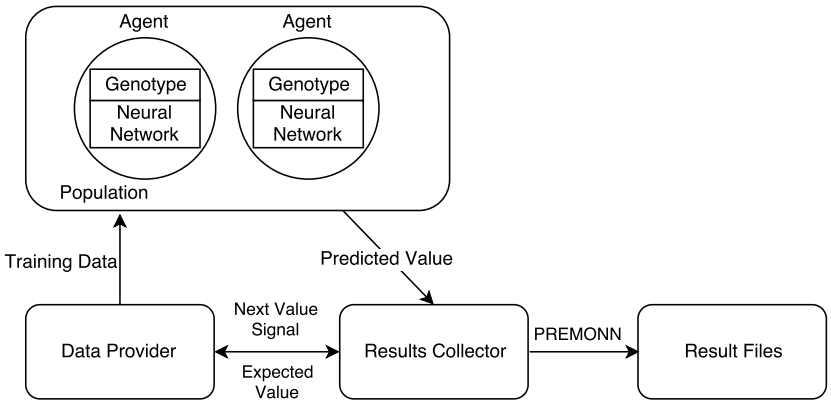


Fig. 2. Overview of implemented system

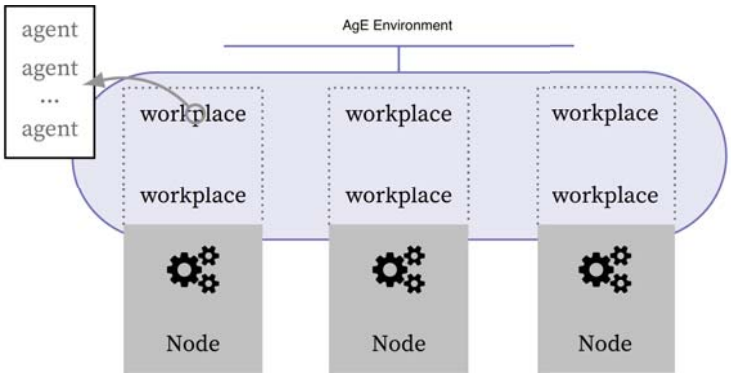


Fig. 3. AgE 3.0 architecture. Gray rectangles show cluster nodes. Each node is executing the EMAS module and each one has two local workplaces. Workplaces together create a single EMAS environment. Each workplace contains its own subpopulation of agents.

group can be perceived as a single being), it seems natural to propose the way of evolving such „complex beings". Every modular network must have its own parameters, characterising its behavior, such as parameters of credit function, and the parameters defining the way of evolving individual agents (amount of rewards and punishments [7]), which can be a subject of the evolution process.

Such two-level evolution may lead to automatic determination of the system parameters, making it more reliable and adaptive to the changes of the work conditions (e.g. in time-series prediction, to the changes of the time-series to be predicted).

III. Optimization of Neural Networks in Agent-based Environment

Optimization and simulation tasks requiring significant computation power are usually realized using dedicated software environments. Let us focus for a moment on agent-oriented ones [1], like REPAST HPC [11], FLAME [10], PDES-MAS [19] or Pandora framework [21] that are devoted mostly for simulation purposes. A renown framework for computing purposes, also HPC enabled is Paradiseo [15]. All the mentioned platforms use quite standard technologies and languages (like MPI, C++, Java), which make possible development by anyone, but does not to a great extend ease the process of creation the software itself. Moreover, it would be desirable to leverage certain flexibility, extensibility and reusability mechanisms (like software components, or at least Dependency Injection/Inversion of Control design pattern). Thus we have usually a very good performance, but steep learning curve and little flexibility. This situation created an opportunity to develop novel, dedicated solutions leveraging contemporary, high-level languages and technologies that would ease the development, debugging, deploying and modification process. Therefore in order to be able to utilize a flexible and extensible environment, instead of using the existing ones we have been developing our own solution for about 15 years, namely AgE.

A. Agent-based Environment – AgE 3.0

AgE[1] is a distributed computational platform written in Java, and designed for agent-based computational systems. Its third version provides a more modern, lightweight and flexible architecture than previous ones. At the bottom, so-called *core* layer, AgE manages communication, fail-tolerance and other services required by a distributed environment. This layer heavily depends on Hazelcast[2] technology. The *core* layer provides utilities needed to built computational modules that are easily scalable and can hide the distributed nature of the software from the end user.

The most important example of the computational module implementation is the multi-agent framework and EMAS library provided with it. Figure 3 shows the architecture of a sample experiment running.

EMAS in AgE 3 is implemented in a functional-like form and uses a stream-based processing where agents are entities passing through a stream. An experiment is defined in terms of stages of a *pipeline*. A pipeline is similar in style to streams known, for example, from the Java language. A user defines pipeline within a *step* that describes one iteration of an algorithm. A step is then passed to a workplace that is a container for a population of agents. It is possible to create multiple workplaces and they can have different configuration and steps, if needed. Agents located in different workplaces are independent in terms of their interactions. The only way for an agent to have interaction with a different workplace is to migrate to it from its current one. In algorithmic terms, workplaces are separate islands with subpopulations.

A user can define the following stages of a pipeline that correspond to the EMAS actions:

- selection,
- reproduction,
- fight,
- evaluation,
- mutation,
- migration,
- death.

All of these stages are implemented as functions with the number of parameters and return values dependent on a stage. It is also possible to pass any other function to execute at any point of the pipeline.

AgE already provides several ready-to-use operators that can be used with EMAS but the user can define new ones when defining the step. As operators are simply functions, they can be defined inline.

[1]Project home pagehttps://age.agh.edu.pl/

[2]https://hazelcast.org/

Both an instance of EMAS experiment and AgE nodes can be configured in two ways: using static XML files or with dynamic JavaScript-based script files. Both approaches have their advantages, but the latter makes it possible to generate more dynamic and differentiated configurations for experiments.

B. Implementation of neural based prediction system with AgE 3.0

AgE platform can be used to implement a neural based prediction system in which agent's genotype refers to the parameters of the neural network. Usage of EMAS allows to find better parameters of the neural network, which results in more precise predictions afterwards. The neural network used for prediction is a multilayer perceptron with two hidden layers. Required functionality was provided by a use of popular Java library deeplearning4j.

To adjust the system to the problem we had to define a proper solution, which contains the genotype and the neural network itself. To perform an experiment it was needed to create a set of stages which were used on a population pipeline.

We used predefined stages like selection, fight, migration and death, but we had to provide the rest to meet the needs of a given problem. To deliver the missing stages we implemented specified interfaces. We created our own recombination with mutation and evaluation. As a recombination we used uniform crossover, as a mutation we decided to use random gene mutation. The evaluation stage calculated the fitness of each agent from the population based on their prediction result. In addition, we defined a new stage - learning phase, which purpose was to train neural networks of agents. Moreover, we provided the sequencer which responsibilities were to deliver the data set at each step and also to indicate expected value of the function.

In the after step we defined a new task to gather all populations and extract the results from the whole system. As a strategy for extracting the result from the whole system, PREMONN algorithm was used. The overall prediction result is calculated based on the predictions of populations and then persisted to the result file. At the end we had to prepare configuration file in which we defined the components and parameters to be used.

IV. Experimental Results

We carried out some experiments in which we tested Mackey-Glass time series prediction. Each of the experiments was performed 10 times. The plots present an average of measured results for each experiment.

Every agent from initial population has a genotype based on values randomly chosen from defined ranges. This genotype corresponds to the neural network parameters like:

1) learning rate,
2) momentum,
3) drop out,
4) number of neurons in first hidden layer,
5) number of neurons in second hidden layer.

After agent is created his neural network is initialized with proper parameters specified by the genotype. Example values ranges for agent's genotype are presented in Table I.

TABLE I
SAMPLE GENOTYPE VALUE RANGES

	From	To
Learning rate	0.01	0.10
Momentum	0.80	0.95
Drop Out	0.10	0.40
Neurons Number	32	256

Table II presents default configuration which was used in the experiments. Configuration was slightly different for each experiment and changes will be described in corresponding subsections.

TABLE II
DEFAULT CONFIGURATION PARAMETERS

Number of workplaces	2
Number of training dataset in each step	50
Starting population size	50
Required energy for reproduction	75
Agent initial energy	30
Fight energy transfer	5
Reproduction energy transfer	25%
Recombination type	Uniform crossover
Mutation type	Random gen mutation
Number of repetitions	10

The Mackey-Glass equation was used to calculate the expected value:

$$\frac{dx}{dt} = \beta \frac{x_\tau}{1 + x_\tau^n} - \gamma x, \qquad \gamma, \beta, n > 0$$

where β, γ, τ, n are real numbers and x_τ represents the value of the variable x at time $(t - \tau)$. Parameters of the equation were different in each experiment.

A. Experiment 1

The parameters of the time series for the first experiment are as follows: $\beta = 0.2$, $\gamma = 0.1$, $\tau = 50$, $n = 10$.

Obtained optimal network parameters are as follows: Learning rate = 0.043, Momentum = 0.881, Drop out = 0.297, Neurons in first hidden layer = 112, Neurons in second hidden layer = 145.

As we can observe in the first steps error rate of the whole system is decreasing to the acceptable level. Afterwards, there are some peaks which are caused by creation of new agents whose neural networks are not trained yet or by the fact the function is more complicated in that point. We can also notice that the number of neurons in the individual layers start to oscillate between different value ranges.

B. Experiment 2

The parameters for this experiment are as follows $\beta = 0.5$, $\gamma = 0.25$, $\tau = 40$, $n = 10$.

Obtained optimal network parameters are as follows: Learning rate = 0.051, Momentum = 0.881, Drop out = 0.314, Neurons in first hidden layer = 110, Neurons in second hidden layer = 162.

At the beginning the error rate is decreasing to the appropriate level and we can notice same peaks as before. The overall error rate of the whole system is bigger than the one before because the function has smaller time period and

is more complicated. Still we can observe some trends like splitting value ranges of neuron numbers in neural networks' layers.

C. Experiment 3

The parameters for this experiment are as follows: $\beta = 2.0$, $\gamma = 1.0$, $\tau = 10$, $n = 10$.

The obtained optimal network parameters are as follows: Learning rate = 0.063, Momentum = 0.876, Drop out = 0.209, Neurons in first hidden layer = 94, Neurons in second hidden layer = 132.

This was the most difficult function we have measured. The system cannot properly predict values of the Mackey Glass time series. The error rate of the whole system is almost the same through the whole experiment. It is difficult to notice any big changes of the genotype values trend.

D. Experiment 4

The parameters for this experiment are as follows: $\beta = 1.0$, $\gamma = 0.5$, $\tau = 30$, $n = 10$.

The obtained optimal network parameters are as follows: Learning rate = 0.058, Momentum = 0.9, Drop out = 0.31, Neurons in first hidden layer = 132, Neurons in second hidden layer = 147.

In this part of experiment we did not use PREMONN algorithm to get the overall result from the system. Instead of that we decided to choose different strategy like taking the average result of 10% of the best agents in whole population.

The obtained optimal network parameters are as follows: Learning rate = 0.061, Momentum = 0.91, Drop out = 0.28, Neurons in first hidden layer = 123, Neurons in second hidden layer = 192.

Finally, we decided to check how the system behaves when the result is determined only by the best agent in the current step.

Obtained optimal network parameters are as follows: Learning rate = 0.073, Momentum = 0.901, Drop out = 0.332, Neurons in first hidden layer = 122, Neurons in second hidden layer = 177.

The strategy of extracting result from the system will have only effect on the error rate of the whole system. As we can see the 10% of the best neural network is slightly better then the PREMONN strategy.

E. Experiment 5

The parameters for this experiment are as follows: $\beta = 1.0$, $\gamma = 0.5$, $\tau = 30$, $n = 10$.

In this one we can observe how the system reacts when the training phase is stopped after passing the the first half of steps measured in the experiment.

The obtained optimal network parameters are as follows: Learning rate = 0.053, Momentum = 0.89, Drop out = 0.294, Neurons in first hidden layer = 130, Neurons in second hidden layer = 185.

As we can notice the overall error rates are peaking high after stopping the learning process. The reason of that is the death of the agents whose neural networks have been trained. Another important fact is that new genotypes will be always worse than those whose the neural networks were trained.

F. Experiment 6

The parameters for this experiment are as follows: $\beta = 1.0$, $\gamma = 0.5$, $\tau = 30$, $n = 10$.

In the next three parts of experiments we wanted to check how the system behaves when the amount of energy transfer will change. We used following values: 3, 10 and 15.

The obtained optimal network parameters are as follows: Learning rate = 0.075, Momentum = 0.9, Drop out = 0.321, Neurons in first hidden layer = 111, Neurons in second hidden layer = 123.

The obtained optimal network parameters are as follows: Learning rate = 0.042, Momentum = 0.87, Drop out = 0.325, Neurons in first hidden layer = 113, Neurons in second hidden layer = 182.

The obtained optimal network parameters are as follows: Learning rate = 0.065, Momentum = 0.907, Drop out = 0.310, Neurons in first hidden layer = 168, Neurons in second hidden layer = 193.

The conclusions we can draw are that the lower the energy transfer is the lower the system error rate is. If the energy transfer is low the population is not changing so fast. Moreover, agents have more time to train their networks. The disadvantage of low energy transfer is that the speed of searching for optimal genotype is rather small. When the energy transfer is high the total life time of the agents is shorter. We are searching for optimum result faster but the chance to drop into local minimum of function is also bigger. To achieve some kind of balance we should also increase the amount of the migrations between workplaces.

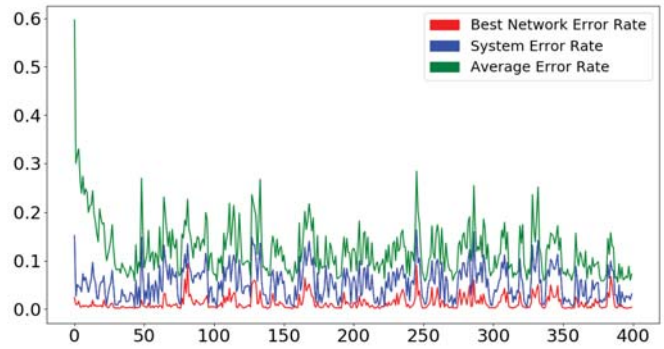
V. Conclusions

In the paper a summary of our work on Collective Intelligent Predicting Neural Networks was presented. Besides recalling our earlier works, we have prepared a novel implementation of the whole system using the AgE 3.0 software environment and we have run a broadly-planned series of experiments focusing on predicting the Mackey-Glass time series.

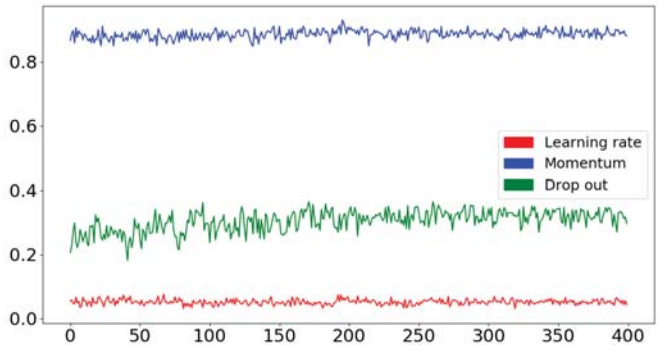
Observing the actual outcomes of the evolution process for different instances of the test problem, we have realized that different values of the genotype were generated as optimal ones. Thus the capability of collective prediction and adaptation to particular problem to be solved of the whole system was presented.

The presented paper can be used as a reference for the researchers wanting to construct similar systems—ensembles of predictors or classifiers focused also on optimization of the parameters of their basic units (like predicting neural networks in the described case).

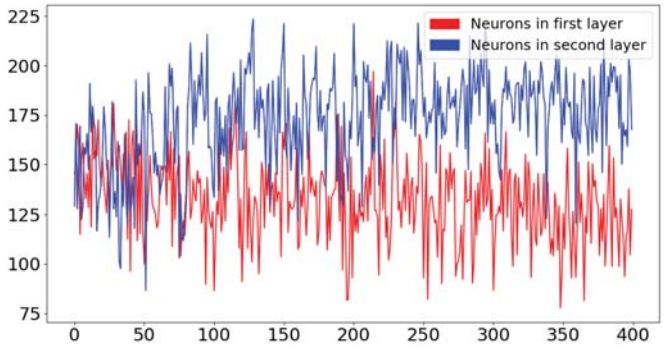
Evolution of a neural network is a time-consuming job because of a complex fitness function evaluation including construction and training of the evaluated network. In this paper we summarize our work connected with optimization of an universal approximator—Multi Layered Perceptron, and these results may be used as a reference for anybody interested in such collective-intelligent predicting. In the future we will focus on a broader selection of neural network architectures, e.g. LSTM.



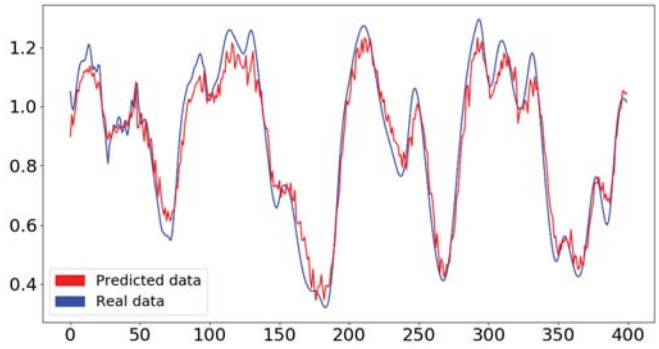
Prediction error rate in every step.



Network parameters for best agent in every step.

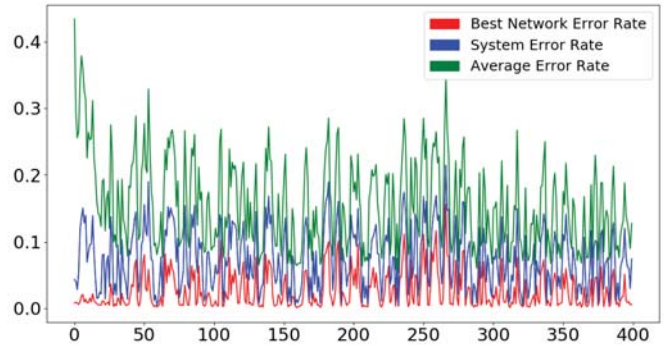


Number of neurons for best agent in every step.

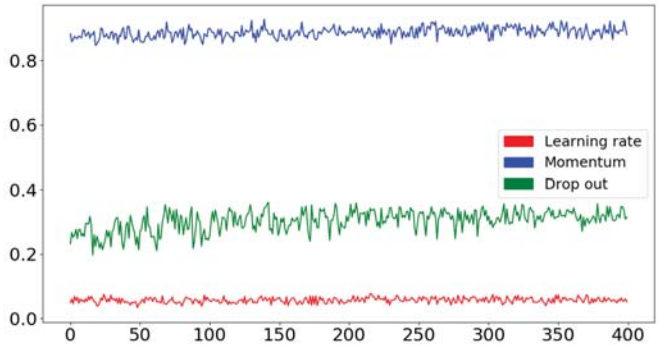


Real value and predicted value in every step.

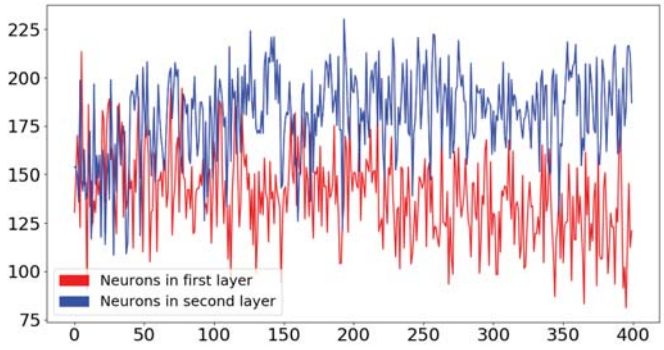
Fig. 4. Result plots from Experiment 1



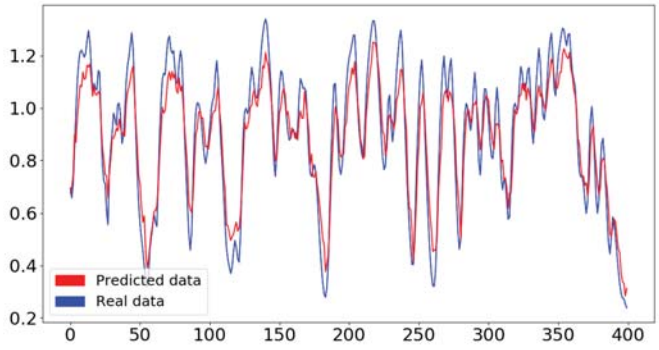
Prediction error rate in every step.



Network parameters for best agent in every step.

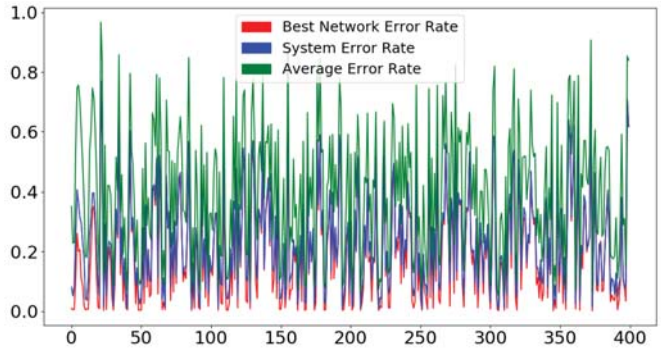


Number of neurons for best agent in every step.

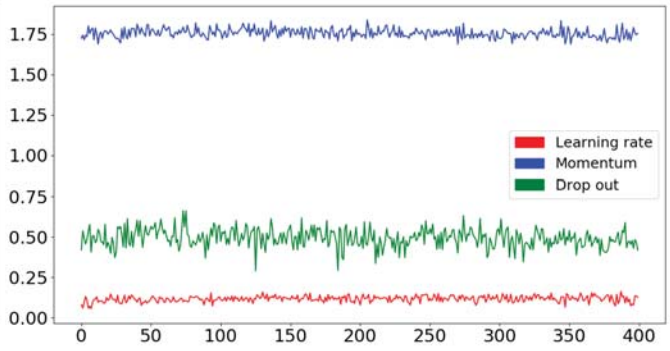


Real value and predicted value in every step.

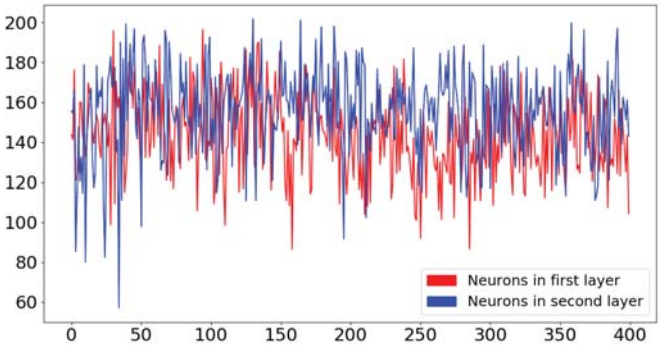
Fig. 5. Result plots from Experiment 2



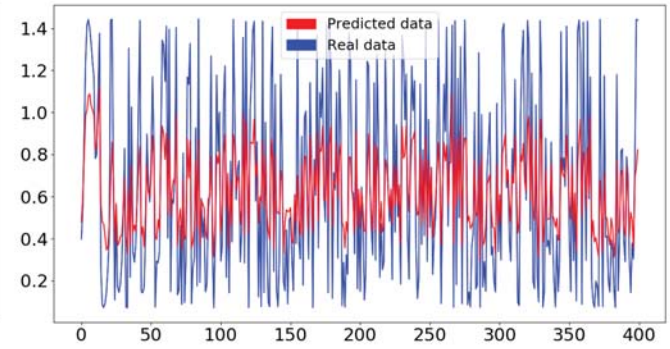
Prediction error rate in every step.



Network parameters for best agent in every step.

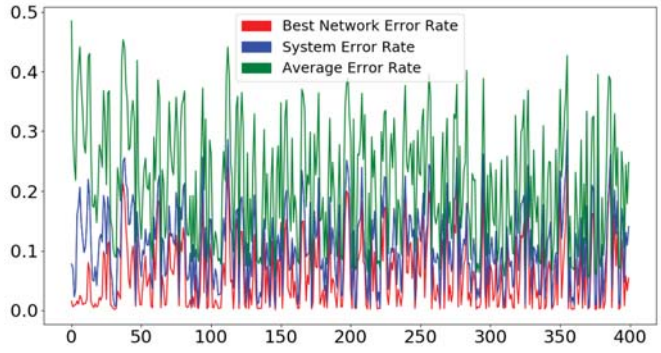


Number of neurons for best agent in every step.

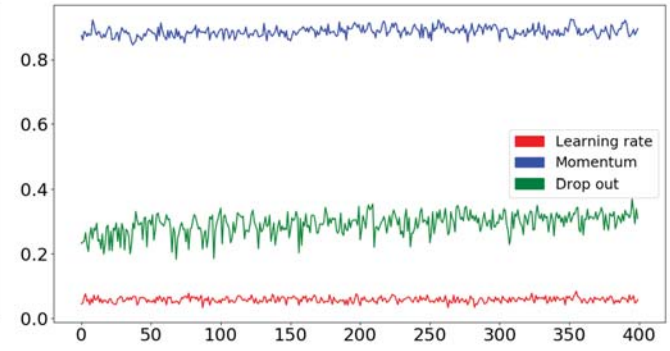


Real value and predicted value in every step.

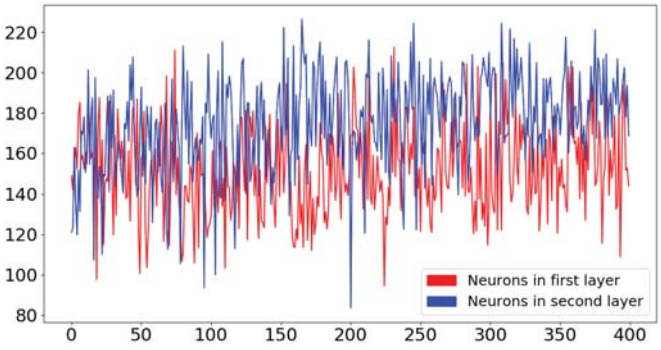
Fig. 6. Result plots from Experiment 3



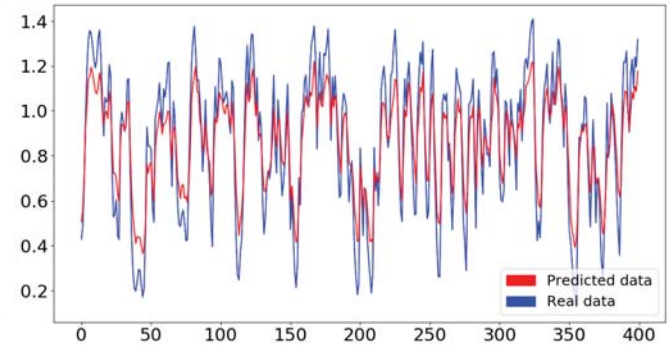
Prediction error rate in every step.



Network parameters for best agent in every step.

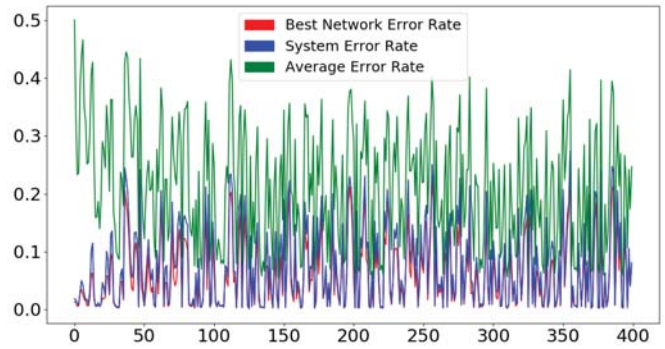


Number of neurons for best agent in every step.

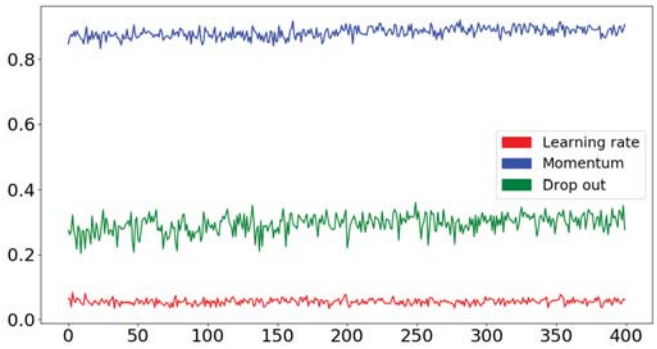


Real value and predicted value in every step.

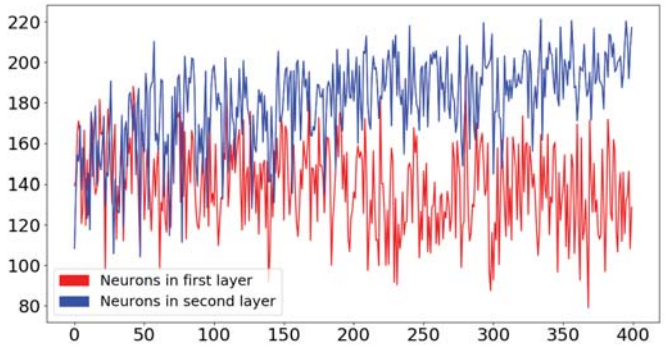
Fig. 7. Result plots from Experiment 4



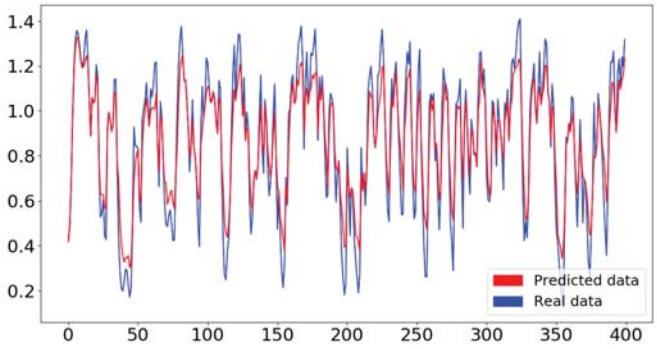
Prediction error rate in every step.



Network parameters for best agent in every step.

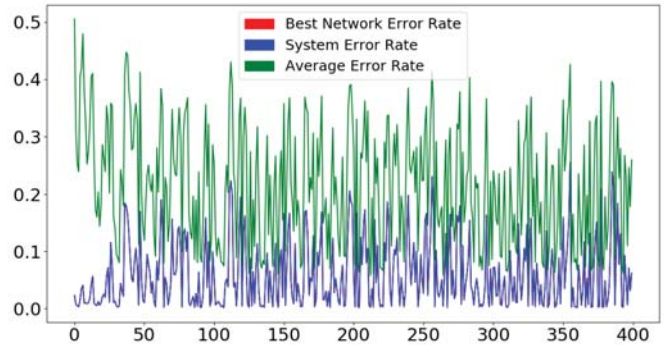


Number of neurons for best agent in every step.

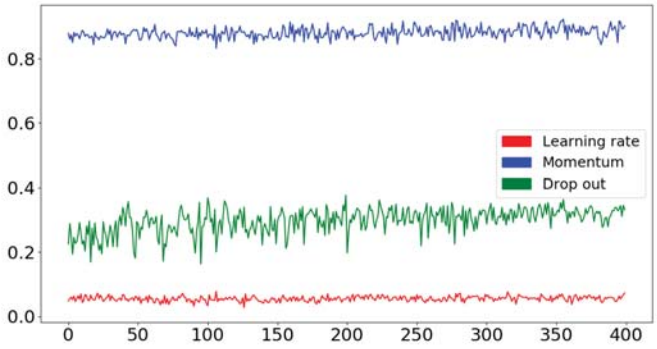


Real value and predicted value in every step.

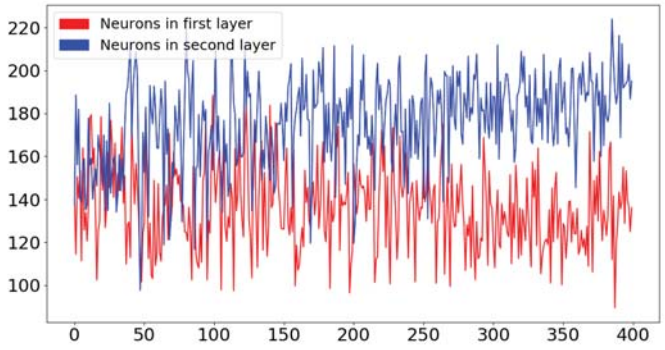
Fig. 8. Result plots from Experiment 4 with the average result of 10% of best agents



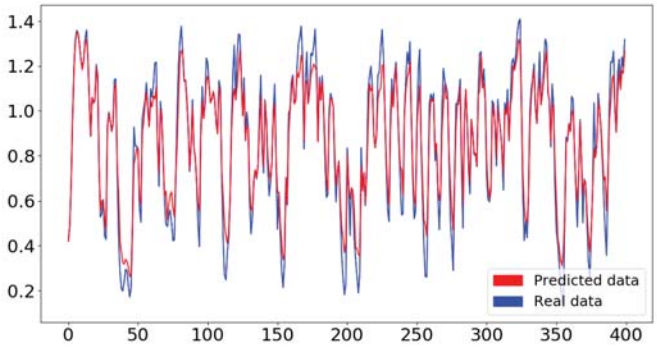
Prediction error rate in every step.



Network parameters for best agent in every step.

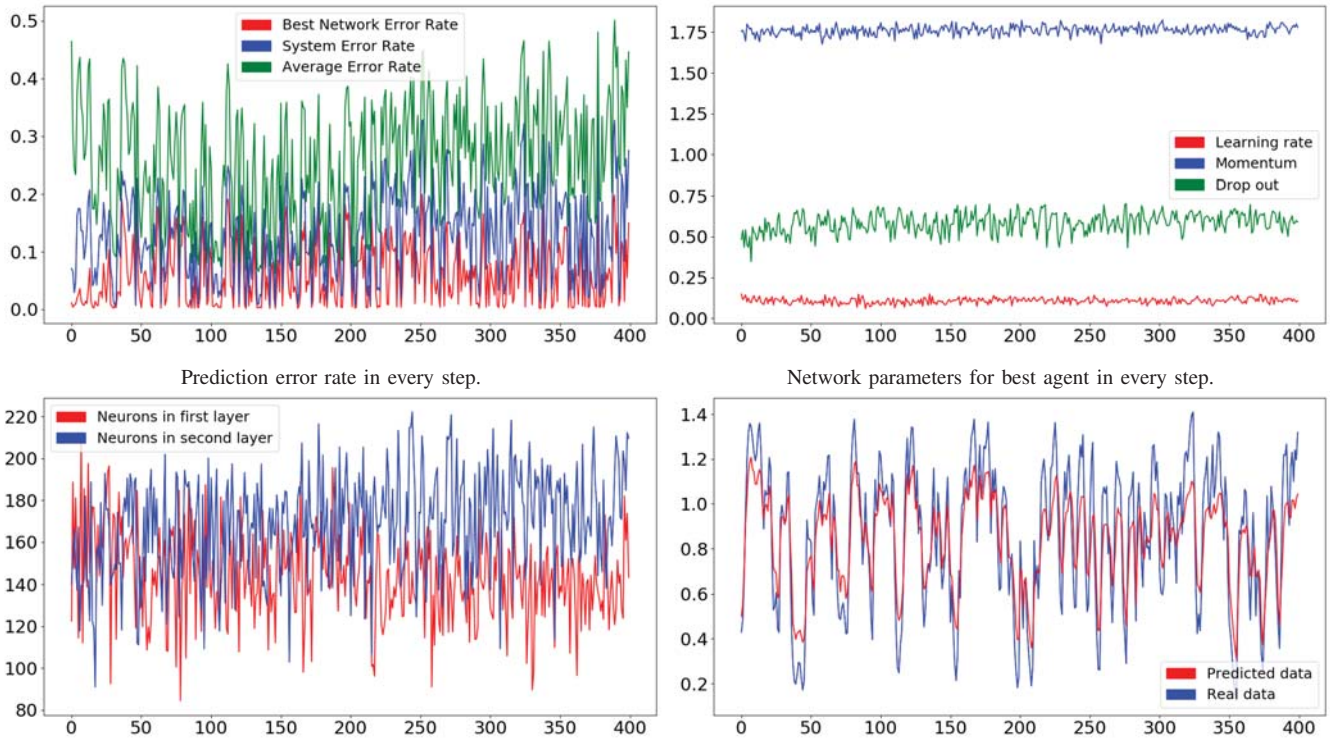


Number of neurons for best agent in every step.



Real value and predicted value in every step.

Fig. 9. Result plots from Experiment 4 with the best agent



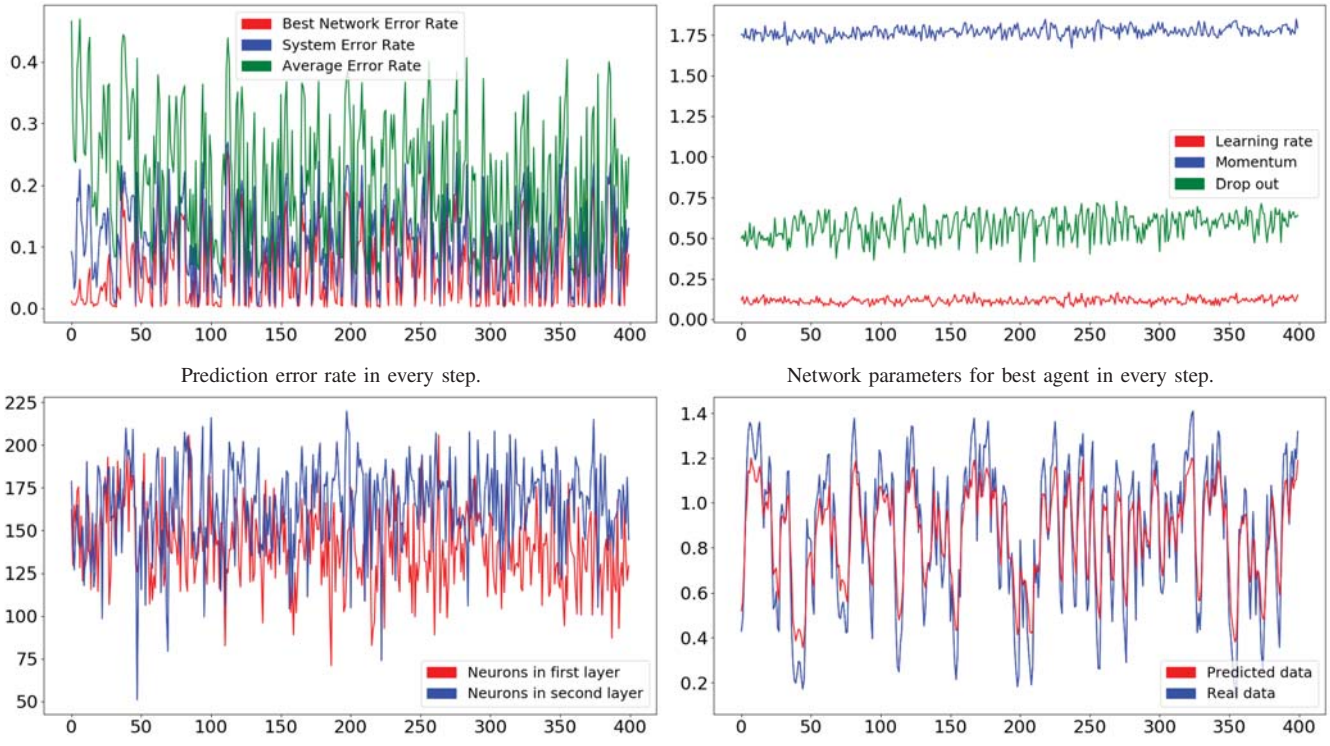
Prediction error rate in every step.

Network parameters for best agent in every step.

Number of neurons for best agent in every step.

Real value and predicted value in every step.

Fig. 10. Result plots from Experiment 5



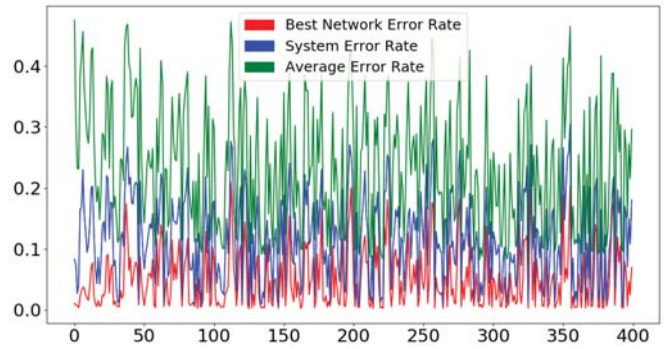
Prediction error rate in every step.

Network parameters for best agent in every step.

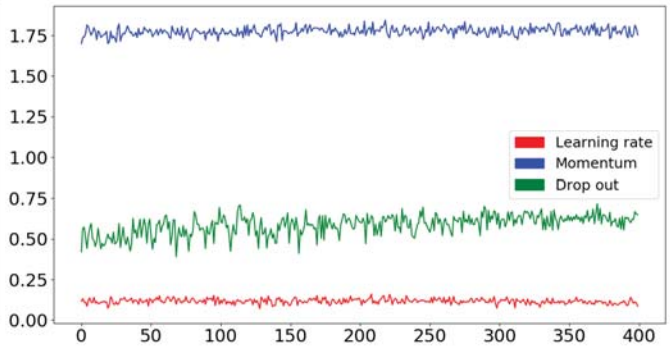
Number of neurons for best agent in every step.

Real value and predicted value in every step.

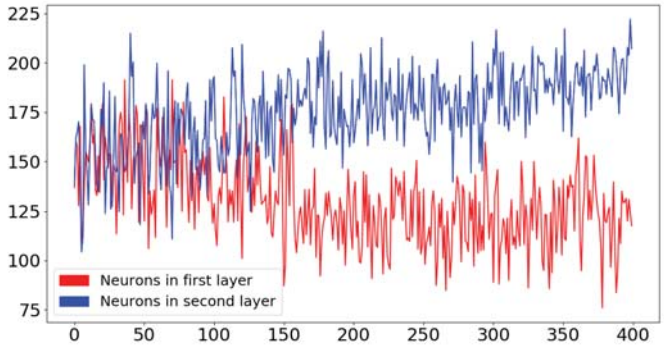
Fig. 11. Result plots from Experiment 6 with energy transfer at level 3



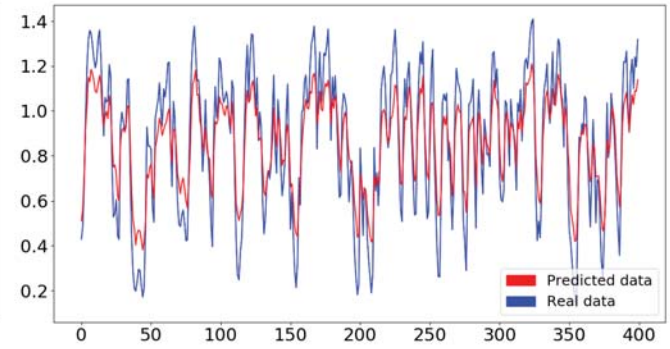
Prediction error rate in every step.



Network parameters for best agent in every step.

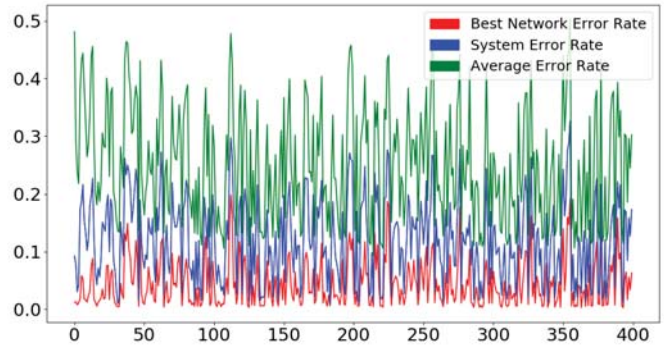


Number of neurons for best agent in every step.

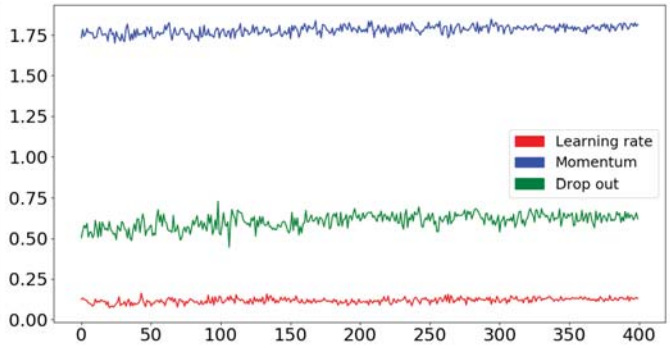


Real value and predicted value in every step.

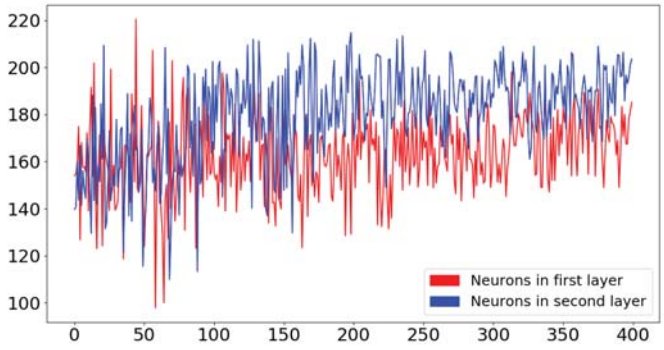
Fig. 12. Result plots from Experiment 6 with energy transfer at level 10



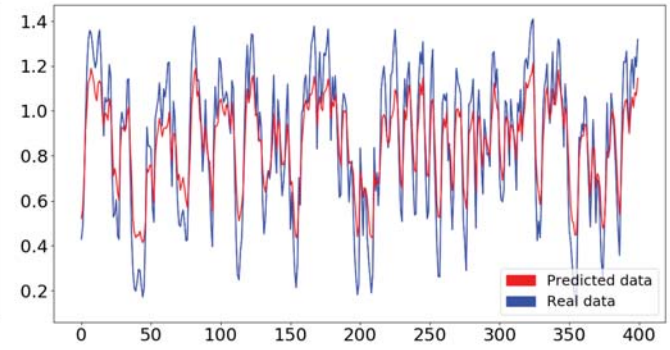
Prediction error rate in every step.



Network parameters for best agent in every step.



Number of neurons for best agent in every step.



Real value and predicted value in every step.

Fig. 13. Result plots from Experiment 6 with energy transfer at level 15

ACKNOWLEDGMENT

The research presented in this paper was supported by the AGH University of Science and Technology Statutory Project.

REFERENCES

[1] S. Abar, G.K. Theodoropoulos, P.Lemarinier, and G.M.P. O'Hare. Agent based modelling and simulation tools: A review of the state-of-art software. *Computer Science Review*, pages –, 2017.

[2] A. Byrski. Evolutionary search for optimal parameters of predicting neural networks. In *Mat. Warsztatów Naukowych Algorytmy Ewolucyjne i Optymalizacja Globalna (KAEiOG 2002) oraz Konferencji Systemy Rozmyte (KSR 2002)*. Politechnika Warszawska, Wydział Elektroniki i Technik Informacyjnych, 2002.

[3] A. Byrski and J. Bałamut. Evolutionary neural networks in collective intelligent predicting system. In L. Rutkowski, editor, *Seventh International Conference on Artificial Intelligence and Soft Computing*. Springer Verlag, 2004.

[4] A. Byrski, J. Dobrowolski, and K. Tobola. *Prace Naukowe*, pages 59–65, 2008.

[5] A. Byrski and M. Kisiel-Dorohinicki. Evolving rbf networks in a multi-agent system. *Neural Network World*, 12(2):440, 2002.

[6] A. Byrski and M. Kisiel-Dorohinicki. Immune-based optimization of predicting neural networks. In V.S. Sunderam, G. Dick van Albada, Peter M. A. Sloot, and J.J. Dongarra, editors, *Computational Science - ICCS 2005, 5th International Conference, Atlanta, GA, USA, May 22-25, 2005, Proceedings, Part III*, volume 3516 of *Lecture Notes in Computer Science*, pages 703–710. Springer, 2005.

[7] A. Byrski, M. Kisiel-Dorohinicki, and E. Nawarecki. Agent-based evolution of neural network architecture. In M. Hamza, editor, *Proc. of the IASTED Int. Symp.: Applied Informatics*. IASTED/ACTA Press, 2002.

[8] A. Byrski, M. Kisiel-Dorohinicki, and E. Nawarecki. Immunological selection in agent-based optimization of neural network parameters. In K. Wegrzyn-Wolska and P.S. Szczepaniak, editors, *Advances in Intelligent Web Mastering, Proceedings of the 5th Atlantic Web Intelligence Conference - AWIC 2007, Fontainebleau, France, June 25 - 27, 2007*, volume 43 of *Advances in Soft Computing*, pages 62–67. Springer, 2007.

[9] K. Cetnarowicz, M. Kisiel-Dorohinicki, and E. Nawarecki. The application of evolution process in multi-agent world (MAW) to the prediction system. In M. Tokoro, editor, *Proc. of the 2nd Int. Conf. on Multi-Agent Systems (ICMAS'96)*. AAAI Press, 1996.

[10] S. Coakley, M. Gheorghe, M. Holcombe, S. Chin, D. Worth, and C. Greenough. Exploitation of high performance computing in the flame agent-based simulation framework. In *2012 IEEE 14th International Conference on High Performance Computing and Communication 2012 IEEE 9th International Conference on Embedded Software and Systems*, pages 538–545, June 2012.

[11] N. Collier and M. North. Parallel agent-based simulation with repast for high performance computing. *SIMULATION*, 89(10):1215–1235, 2013.

[12] S. Haykin. *Neural Networks and Learning Machines*. Pearson, 2008.

[13] N.K. Kasabov. *Foundations of Neural Networks, Fuzzy Systems, and Knowledge Engineering*. The MIT Press, 1996.

[14] K.E. Lane-deGraaf, R.C. Kennedy, S.N. Arifin, G.R. Madey, A. Fuentes, and H. Hollocher.

[15] A. Liefooghe, L. Jourdan, and E.-G. Talbi. Technical report.

[16] T. Masters. *Neural, Novel and Hybrid Algorithms for Time Series Prediction*. John Wiley and Sons, 1995.

[17] V. Petridis and A. Kehagias. *Predictive Modular Neural Networks – Application to Time Series*. Kluwer Academic Publishers, 1998.

[18] K. Pietak and M. Kisiel-Dorohinicki. Agent-based framework facilitating component-based implementation of distributed computational intelligence systems. In Ngoc-Thanh Nguyen, Joanna Kołodziej, Tadeusz Burczyński, and Marenglen Biba, editors, *Transactions on Computational Collective Intelligence X*, pages 31–44, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[19] V. Suryanarayanan, G. Theodoropoulos, and M. Lees. Pdes-mas: Distributed simulation of multi-agent systems. *Procedia Computer Science*, 18:671 – 681, 2013.

[20] Z. Toth and E. Kalnay. Ensemble forecasting at ncep and the breeding method. *Monthly Weather Review*, 125(12):3297–3319.

[21] P. Wittek and X. Rubio-Campillo. Scalable agent-based modelling with cloud hpc resources for social simulations. In *4th IEEE International Conference on Cloud Computing Technology and Science Proceedings*, pages 355–362, Dec 2012.

Joanna Kijak obtained B.Eng. in 2017, she is currently a student at AGH University of Science and Technology in Krakow, Poland, pursuing a Master Degree in computer science. Interested in neural networks.

Piotr Martyna obtained B.Eng. in 2017, he is currently a student at AGH University of Science and Technology in Krakow, Poland, pursuing a Master Degree in computer science. Interested in cloud computing and distributed systems.

Aleksander Byrski obtanied Ph.D. in 2007 and D.Sc in 2013, he works at AGH University of Science and Technology in Krakow, Poland, he is interested in nature-inspired computing and agent-based simulation.

Łukasz Faber obtained M.Sc. in 2012 at AGH University of Science and Technology in Krakow, Poland and he is currently a Ph.D. student at the Department of Computer Science of AGH-UST. His research interests include agent based modeling and distributed systems.

Kamil Piętak obtained Ph.D. in 2017, works at AGH University of Science and Technology in Krakow, Poland, he is interested in heterogeneous computing, distributed and parallel HPC and component-based systems.

Marek Kisiel-Dorohinicki obtained Ph.D. in 2001 and D.Sc. in 2013, works at AGH University of Science and Technology in Krakow, Poland, he is interested in agent-based systems, parallel and distributed computing and simulation.