Slovak University of Technology in Bratislava
Institute of Information Engineering, Automation, and Mathematics

PROCEEDINGS

of the 18th International Conference on Process Control

Hotel Titris, Tatranská Lomnica, Slovakia, June 14 – 17, 2011

ISBN 978-80-227-3517-9

http://www.kirp.chtf.stuba.sk/pc11

Editors: M. Fikar and M. Kvasnica

Petrík, M., Kozák, Š.: Prediction of Critical Processes in Nuclear Power Plant Using Genetically Trained Neural Networks, Editors: Fikar, M., Kvasnica, M., In *Proceedings of the 18th International Conference on Process Control*, Tatranská Lomnica, Slovakia, 77–84, 2011.

Full paper online: http://www.kirp.chtf.stuba.sk/pc11/data/abstracts/048.html

Prediction of critical processes in nuclear power plant using genetically trained neural networks

Matej Petrík * **Štefan Kozák** **

* *Matej Petrík, Institute of Applied Informatics at FIIT SUT (Tel: +421 910 904 516; e-mail: matej.petrik@gmail.com)*
** *Štefan Kozák, Institute of Automatic control systems at FEI SUT (Tel: +421 905 581 323; e-mail: stefan.kozak@stuba.sk)*

Abstract: Neural network is one of many models used in power engineering process prediction. In most cases, the accuracy of prediction models is critical in operational safety or is used to support human irreversible decisions. We use neural network, when the real model of process is unknown, or it is too difficult to identify them in domain specific environment. Neural networks training algorithms are different. Typically, measured data are divided into 2 sets called train and test set. On the train set, algorithms set neural network parameters so that network simulate process on train interval. On the test set is network tested if it can generalize the process from train set. We take a look on special genetic training and compare it with algorithm used today to generalize the process.

Keywords: neural network genetic plant energetics

1. INTRODUCTION

Nuclear power plant performance is the result of complex system composed of nuclear reactor, warm and cold water, radioactive water, turbines and electricity generator. Source of generated energy are graphite rods containing uranium. Uranium is fuel for controlled nuclear reactions, which results into non-linear amount of irradiated heat. This heat is transfered through two system of warm water to large generator turbines - figure 11. Even though we know the physical background of the nuclear chain reaction and thermodynamic laws, we are not able to compute physical model because of many (but measurable) external impacts. Instead of assembling mathematical model of physical laws affecting the outputs in nuclear power plant, we are trying to create soft-computing methods to solve the prediction problem. Performance modeling is critical in 2 cases: security and economic. To prevent and avoid critical situation, we can't generate more energy in reactor than the water system can absorb. From economic reasons we have to generate just enough energy, which is possible to consume by customers.

Described process is nonlinear dynamic system, where the current state strongly depends on previous situation and outside conditions. It is possible to model dynamic systems in many ways, but with neural networks we hold a great tool - universal approximator. Offline training (which is specific for neural networks with supervisor) bring us opportunity to research new algorithms without affecting the real world. The only important point on generated neuro-model is generalization quality - the ratio between test and train prediction error. Classical algorithms use during the training process prediction error as error signal to calculate new network parameters. In this work, we are going to describe and show special genetic algorithm for training neural network, and compare them with standard methods used nowadays in nuclear plants for training neural networks.

Main area of the proposed paper is creation of effective dynamic models for prediction of possible critical situations. Created system can be used in supervisor mode for supporting decision processes by human professionals. Measured data were obtained from collecting the real system values in nuclear plant.

Neural network models used nowadays in nuclear power plants are described in APVV project documentation for VVER 400 reactor. They are trained with iterative - newton methods provided by toolbox NNSYSID because of low prediction error in real usage.

There are many publications about neural networks, but for purposes of this paper we are using only NNSYSID toobox, which is fully described at NNSYSID homepage.

2. PROBLEM FORMULATION

Because of previously used models of neural network implementations in nuclear plant [1] we are going to train recurrent network - multilayered percepton in toolbox NNSYSID. This architecture of recurrent connections allows connect previous outputs only at input layer of neurons, not hidden layers of network in input vector, as shown in figure 1

As an activation function we are using hyperbolic tangent sigmoid:

[1] Project APVV - Application of artificial intelligent methods in modeling and control of critical processes in power industry.

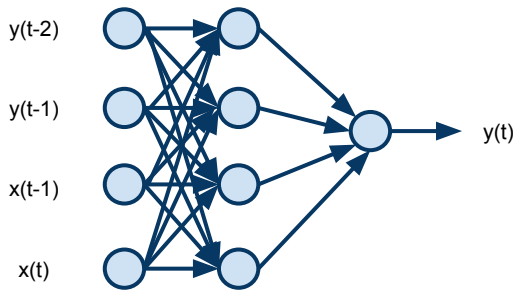


Fig. 1. Neural network architecture, x - input, y - output, t - time sample

$$y = f(\overline{x}) = \tanh\left(\sum_{i=1}^{n} w_i x_i - w_{i+1}\theta\right) \quad (1)$$

where:
$\overline{x}$ is input vector
N is number of connections to given neuron
w_i is weight, which affects input signal
θ is threshold
w_{i+1} is weight for threshold

It is possible to use another activation function, for example:

$$y = \frac{1}{1+e^{-x}} \quad (2)$$

where:
e is Euler's number

Because of the gradients of those functions are similar, it is not necessary to determine which function is better. Only the interval used to initialize network with random weights correlate with this gradient.

The whole network activity is computed by connected layers of neurons into one intelligent object and the resulting signal is computed by (for MISO neuro model):

$$\overline{x} = x_1, ..., x_N y_1, ..., y_{N-1} \quad (3)$$

$$y(t) = f_{act}\left(\sum_{i=1}^{i=L} w_{out} f_{act}\left(\sum_{i=1}^{i=K} w_{hid}^{i}\overline{x}\right)\right) \quad (4)$$

where:
$x_1 ... x_N$ are input values with delay
$y_1 ... y_{N-1}$ are previous network outputs
f_{act} is activation sigmoid function
w_{out} is vector of output layer weights
w_{hid} is matrix of hidden neurons weights

For simulating a dynamic system, we are searching for specific hidden and output layer weights combination. The solution can be interpolated into n-dimensional space (where n is number of weights in network). Than we can simply represent the solution or state of network with one point:

$$y(t) = F(W_{w_{1hid}}, ..., W_{w_{Khid}} W_{w_{1out}}, ..., W_{w_{Lhid}}) \quad (5)$$

where:
$F()$ is transfer function to y(t) from n-dimensional space
$\overline{W}$ is vector of all weights in network

To specify the sufficient condition for evaluating a quality of prediction we use prediction error and average prediction error for one sample:

$$\epsilon = \sum_{i=1}^{N} \frac{1}{2}(F(x) - G(x))^2 \quad (6)$$

$$\epsilon_{avg} = \frac{\sum_{i=1}^{N} \frac{1}{2}(F(x) - G(x))^2}{N} \quad (7)$$

where:
ϵ is general error
ϵ_{avg} is average error for 1 sample
N is number of samples used in test set
$F(x)$ is computed output of simulated system
$G(x)$ is real output of the system

Different training algorithms comparison should be done trough monitoring the neural network state in n-dimensional space after each iteration (if the algorithm is iterative-based, but most are). Next preference of algorithm is one step Hamming distance between the iterations. Stochastic model evaluating the next-step position in n-dimensional state is useful for measurement the probability of Hamming distance:

$$x(t+1)_{p_1 ... p_N} \in \text{sph}(x(t)_{p_1 ... p_N}; \overline{\delta}) \quad (8)$$

$$\overline{\delta} = \Delta_{max} p_1^2; ...; \Delta_{max} p_N^2 \quad (9)$$

where:
$\text{sph}(\overline{x}, \overline{y})$ is n-dimensional spheroid function
$x(t)_{p_1 ... p_N}$ are neural network state coordinates
$\overline{\delta}$ is vector of n-dimensional spheroid dimensions
$x(t+1)_{p_1 ... p_N}$ is possible state in next iteration
$\Delta_{max} p$ is maximum possible change in current weight

Main idea - nuclear plant performance prediction, should be computed from reactivity, neutron flux (in nuclear reactor) and warm and cold balance valves temperatures [2]. Progress examples of these variables are displayed at figures 2, 3, 4, 5. The output of system (which is also the expected output of our model) - performance is displayed at figure 6.

As seen on figures (and also all industry data) data are recorded with some noise (additive white noise, $1/f$ pink noise, gray noise [3]). To avoid network from learning useless noise, it is necessary to filter all input signals before putting it to neural network inputs. The mostly used filtration methods to smooth a noise are: moving average and Savitzky-Golay filter. One good example of moving average is exponential moving average:

$$EMA_N = \frac{\sum_{i=N-n}^{N} w\alpha^{N-i} m(i)}{\sum_{i=N-n}^{N} w\alpha^{N-i}} \quad (10)$$

[2] Same, as were trained models in project APVV
[3] In this case we have a white noise on our data

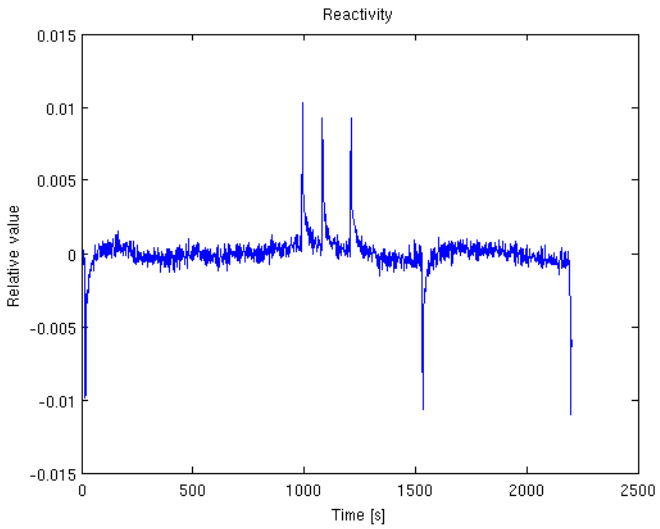


Fig. 2. Time response of reactivity

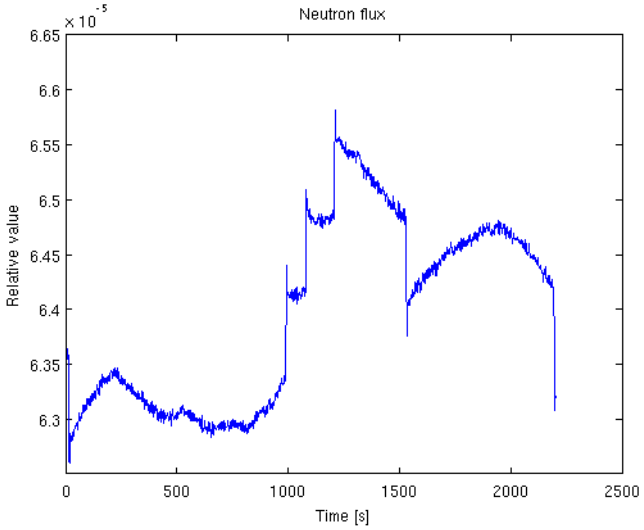


Fig. 3. Time response of neuron flux

where:
N is index of measurment
w is weight of previous measurment
α is decreasing coefficient of previous weights
m is vector of measured values
Savitzky-Golay filter is able to represent local extremes on noisy data.

3. CASE STUDY: MODEL IMPLEMENTATION

The goal is to explore new methods in neural network training, reduce known disadvantages of currently used learning algorithms. When we use Newton optimization methods, training process strongly depends on Gradients, Hessians, local extremes, process characteristic. These properties have major impact on train / test set prediction error ratio. Genetic optimizations are different. The main difference is, that the steps between iterations are the same [4] and the error signal doesn't affect the weights directly, but networks with bad results are replaced with

[4] not strictly, but comparing to newton optimization methods

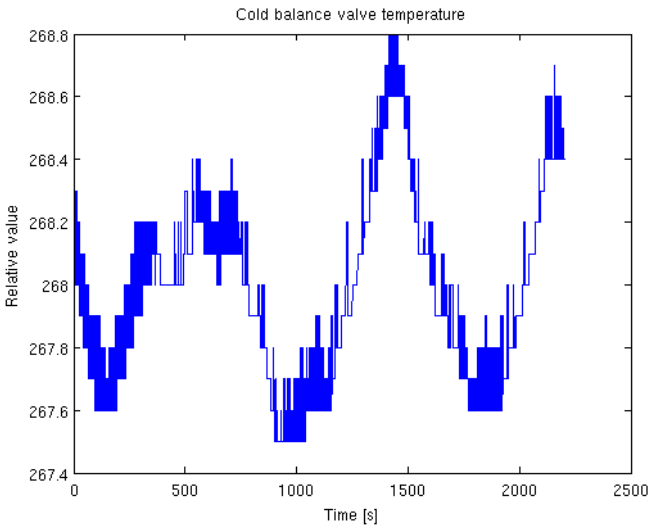


Fig. 4. Time response of cold valve temperature

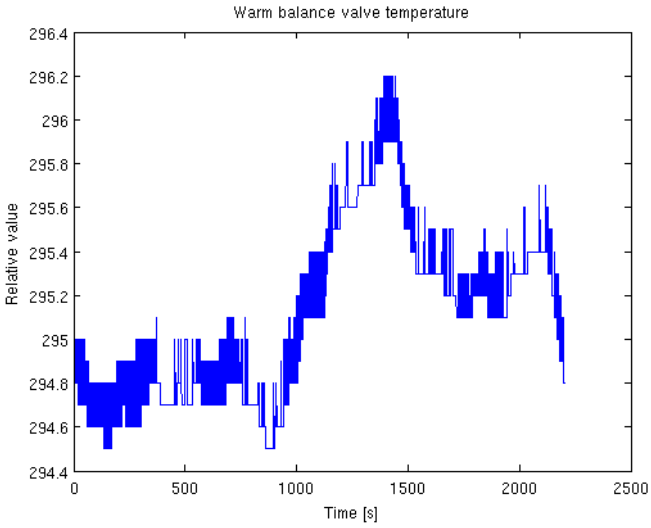


Fig. 5. Time response of warm valve temperature

better ones. Before solving an optimization problem with genetic (evolutionary) algorithm is necessary to create transfer function from problem state to gene. After a transfer function is created, we have to specify a fitness function. Fitness function is used to evaluation given gene. After that, the optimization begins with M [5] randomly initialized genes. These genes are evaluated with fitness function, and sorted from the best to worst. After sort, we take some better genes for the application of genetic methods - crossover, mutation [6]. This process is repeated in iterations while we found acceptable solution.

In neural networks, genes should be simple all the weights, which represent the state. So the transfer function is simple - matrix form hidden and output layers weights. Fitness function should be prediction error on train data.

[5] Typically M is between 20 - 40

[6] These are the most used in evolutionary programming, typically each genetic optimization problem need separate analysis of appropriate methods

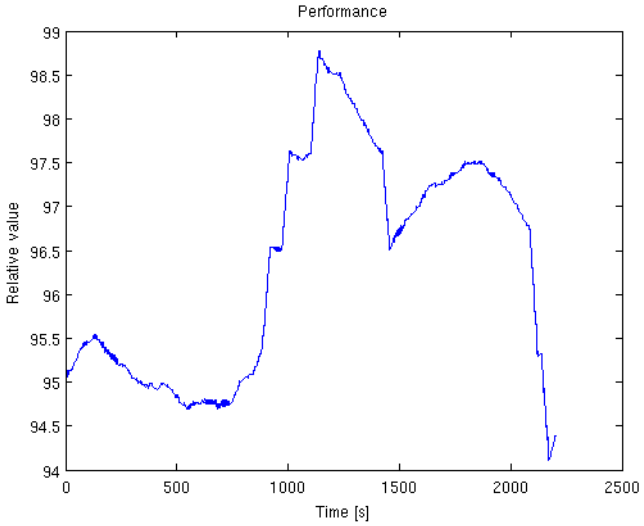


Fig. 6. Time response of performance

For better understanding both approaches, lets assume that the acceptable neuro-model solution is situated in n-dimensional space in hypercube with dimensions $[-1_{w_1}...-1_{w_N}; 1_{w_1}...1_{w_N}]$, or simply all the weights are from interval $< -1, 1 >$ [7]. In Newton optimization methods are the next-step Hamming distance of weights smaller and strongly depends on error signal, so the state where can next state occur in n-dimensional space is very irregular and small spheroid [8]. Otherwise, in genetic algorithm we use mutation - random change in random weights in some genes, so the next state, where can network occur is situated into sphere, not spheroid, and this sphere covers the whole hypercube of possible solutions [9].

Before training, we must specify how much neurons should network have, and how much earlier measured data is necessary to put on network input. Experimentally we set 40 as number of hidden layer neurons, 3 past inputs and previous outputs on input.

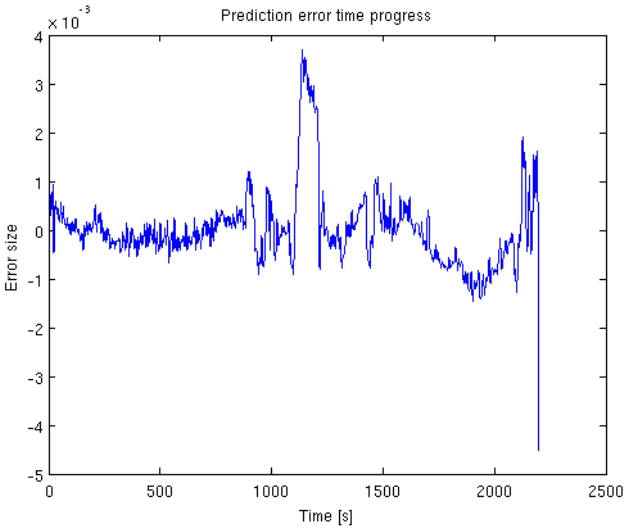
First, let's look at neural network trained with Newton optimization method - Levenberg-Marquadt. This method is implemented in toolbox NNSYSID used with Matlab environment. We took some 35 minute data as a train set [10] and we started the training process. The result is shown at figure 7.

After that, we test the model on some randomly selected data (more than 5 hours). The result on test set is shown at figure 8

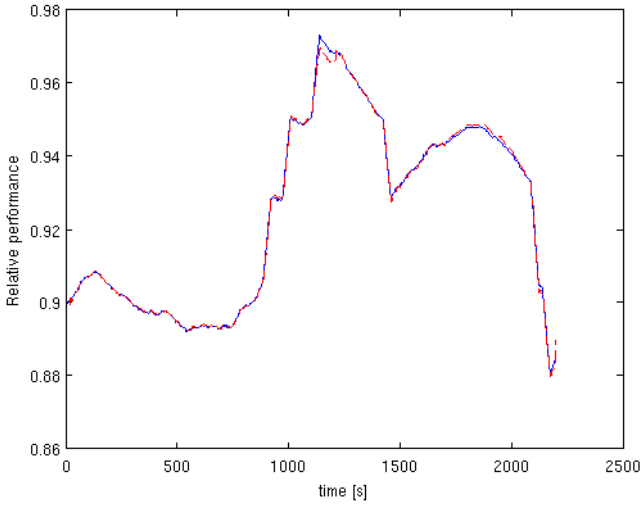
In genetic training, first we have to specify crossover function and mutation functions(s). For crossover we choose this function:

$$G_{new} = MG_{rand_1} + \text{inv}(m)G_{rand_2} \quad (11)$$

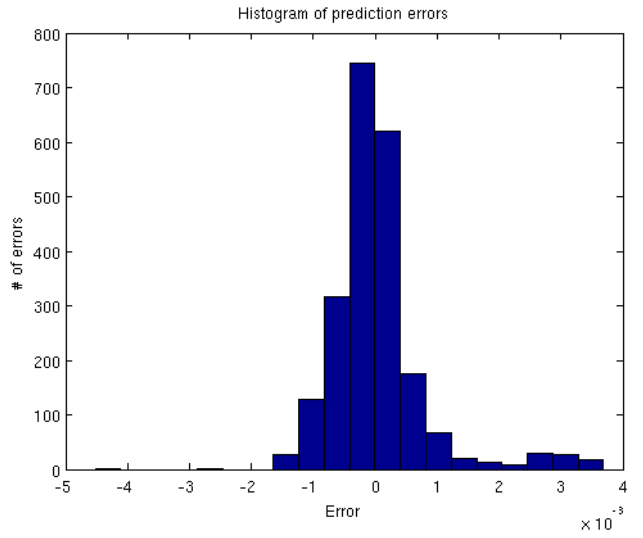
[7] It's not necessary to think about bigger interval, because of fast convergence of used sigmoid functions
[8] Something like "irregular spheroid" doesn't exist, but the radiuses of every dimensions are different - also depends on error signal
[9] The probability of next state in sphere is smaller with his increasing radius, but always possible
[10] Where were performance decreasing and increasing



(a) Time response of prediction error

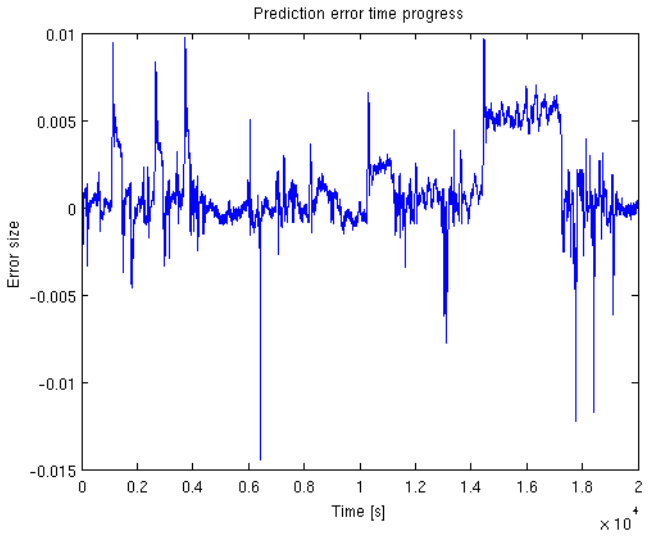


(b) Prediction (red) and reality (blue)

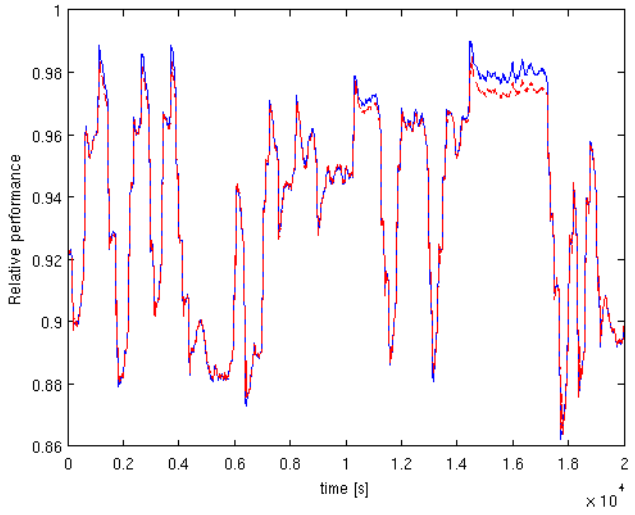


(c) Prediction error histogram

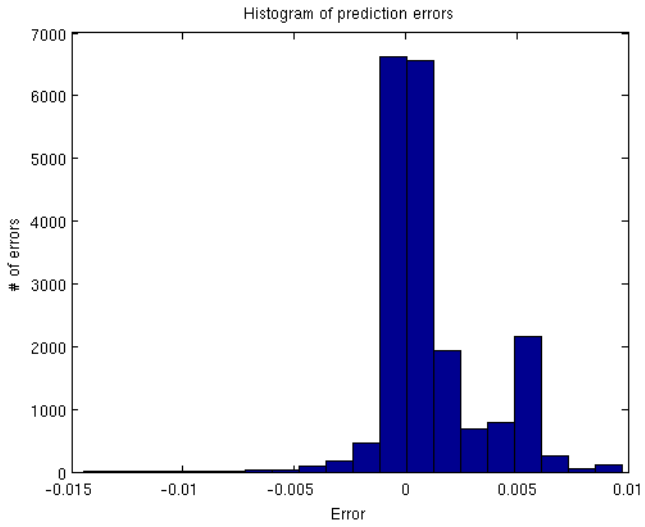
Fig. 7. Levenberg-Marquadt method, train set



(a) Time response of prediction error



(b) Prediction (red) and reality (blue)



(c) Prediction error histogram

Fig. 8. Levenberg-Marquadt method, test set

where:
G_{new} new gene matrix, which represent neural network state
G_{rand_1}, G_{rand_2} are two randomly chosen genes
M is randomly generated binary matrix
inv is inverse function

And for mutation, we choose 2 mutation: classic and output layer moving mutation:

$$G_{new} = M\text{rand}(-1,1)G_{rand} \tag{12}$$

where:
G_{new} is new gene matrix
G_{rand} are randomly chosen gene from population
M is randomly generated binary matrix
rand is random function

$$G_{new} = rand(-1,1)\text{hid}(G_{rand}) \tag{13}$$

where:
G_{new} is new gene matrix
G_{rand} are randomly chosen gene from population
rand() is random function
hid() is function for extracting only hidden layer weights

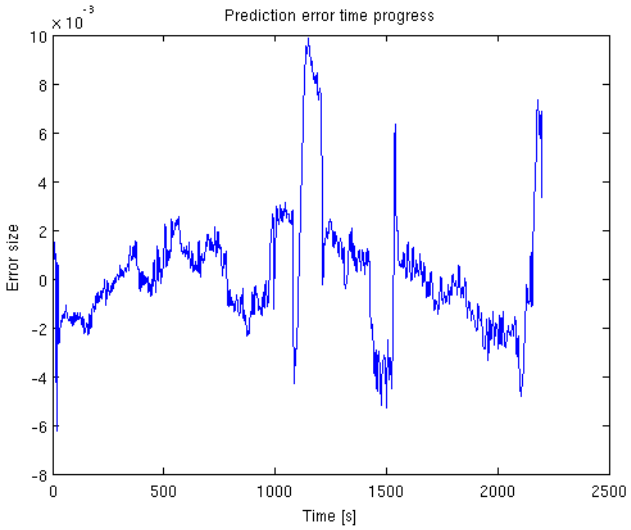
Our algorithm creates 40 randomly generated genes and sorts it in every iteration, after application of genetic methods shown above. We stop the algorithm after 1370000 iterations and it takes more than 8 days to run. To better comparison, we let the train and test set the same as in Levenberg-Marquadt method. The result of genetic method is shown at figure 9 (genetic training) and 10 (test set). Comparison of prediction errors between genetic and Levenberg-Marquadt method is shown at table 1.

Although that the prediction error with genetic training is greater, the important thing is, that the ratio between train/test set error is smaller. From this point of view we can declare, that genetic algorithm has better ability to learn process characteristic from train set and it is valuable alternative to another training methods.

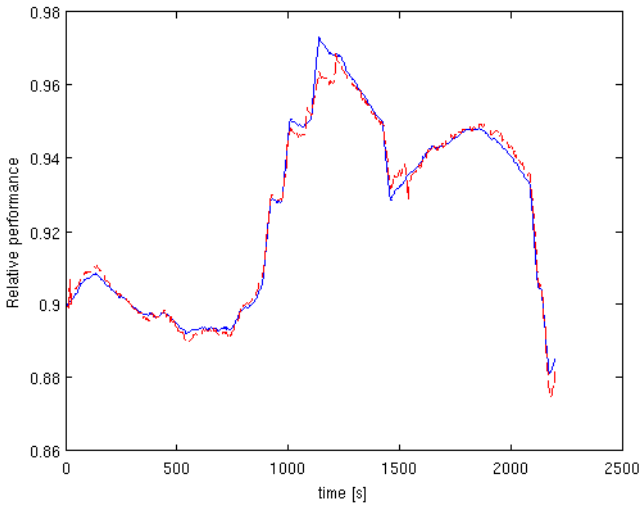
This genetic algorithm was implemented in Matlab environment, integrated with NNSYSID toolbox using the same data structures describing the neural network state as have been chosen by NNSYSID toolbox authors. Algorithm has constant memory requirements, uses all CPU resources and was tested in Matlab 2010. Both approaches were tested with NNSYSID function NNVALID.

Table 1. Levenberg-Marquadt and genetic training method comparison

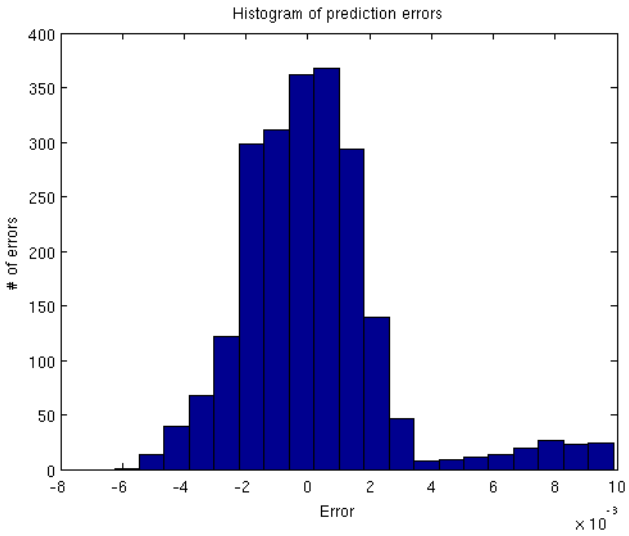
	Genetic alg.	LM alg.
Prediction error (Train)	3.0420×10^{-6}	3.0521×10^{-7}
Prediction error (Test)	1.5191×10^{-5}	3.2994×10^{-6}
Prediction error ratio	5	10
Runtime	8 days	approx. 15 min.



(a) Time response of prediction error

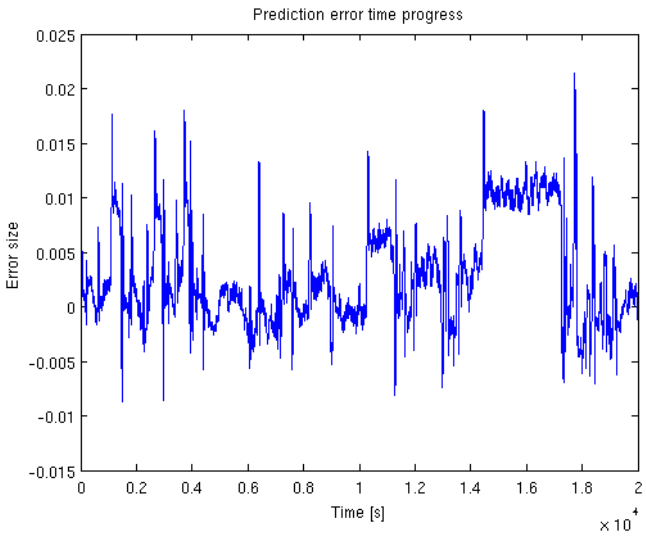


(b) Prediction (red) and reality (blue)

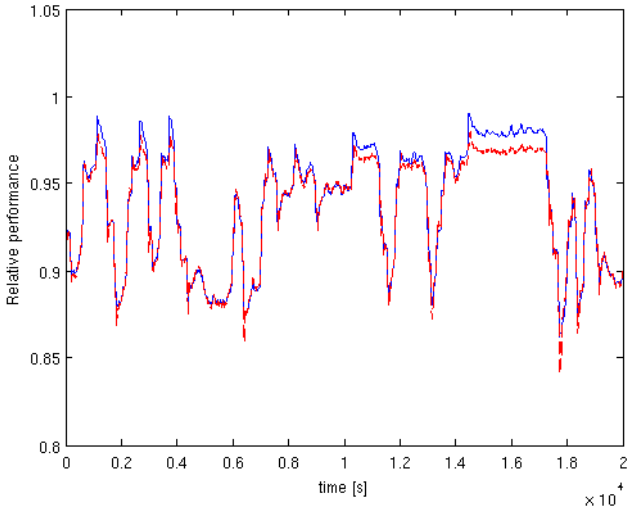


(c) Prediction error histogram

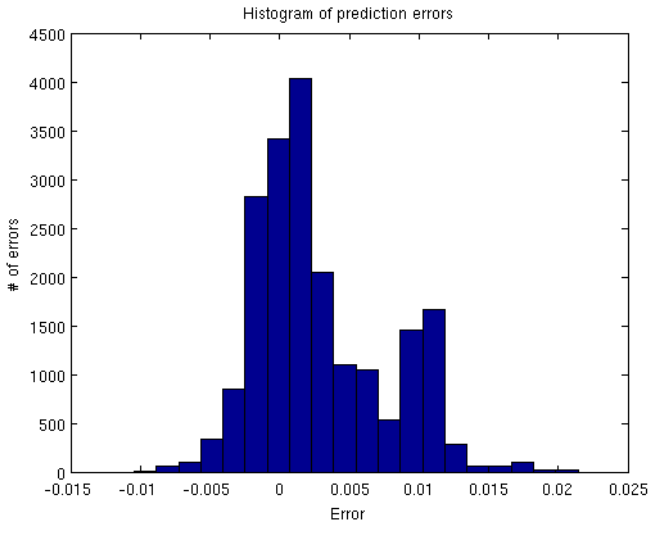
Fig. 9. Genetic method, train set



(a) Time response of prediction error



(b) Prediction (red) and reality (blue)



(c) Prediction error histogram

Fig. 10. Genetic method, test set

4. CONCLUSION

In this paper, we provide comparison of real implemented soft-computing genetic algorithm for neural network training with often used Levenbeg-Marquadt optimization method. With these methods were 2 neural networks trained with real data, on same test and train intervals. The result is, that genetic neural network training is acceptable alternative to other training methods with respect on better overlearning resistance. Modeled system provide prediction of nuclear plant power performance, and the prediction error were calculated from real data.

Solution should help with prediction of critical situations in nuclear engineers decisions processes with supporting and precalculating the state of nuclear reactor and other power systems. Main advatage of using neural networks in proces prediction is, that there is no need to strictly define process identification parameters of models (the model shall be also very good after adding more neurons and neuro-connections) unlike in many other system identification techniques.

ACKNOWLEDGMENTS

This project has been supported by VEGA project N^{o} 1/1105/11. This support is very gratefully acknowledged.

REFERENCES

NNSYSID toolbox, http://www.iau.dtu.dk/research/control/nnsysid.html

Kvasnička, V., Pospíchal, J., Kozák, Š., Návrat, P.: *Umelá inteligencia a kognitívna veda.* 1st edition. Bratislava: Slovenská technická univerzita v Bratislave. Fakulta informatiky a informačných technológií, 2009, 457

Sekaj, I.: *Evolučné výpočty a ich využitie v praxi.* 1st edition. Bratislava: IRIS, 2005, 157

Kvasnička, V., Pospíchal, J., Tiňo, P.: *Evolučné algoritmy.* 1st edition. Bratislava: Slovenská technická univerzita v Bratislave, 2000, 215

Rosinová, D., Dúbravská, M.: *Optimalizácia.* 1st edition. Bratislava: Slovenská technická univerzita v Bratislave, 2007, 195

Návrat, P. a kol.: *Umelá inteligencia.* 1st edition. Bratislava: Slovenská technická univerzita v Bratislave, 2007, 393

Bratko, R.: *Matlab II. Optimalizácia.* 1st edtion. Praha: VŠCHT, 2008, 227

National Instruments: *Selecting a Model Structure in the System Identification Process* http://zone.ni.com/devzone/cda/tut/p/id/4028#toc6, 2.5.2010

Rail, R.: *Neural Networks A Systematic Introduction* 1st edition. New York: Springer, 1996, 722

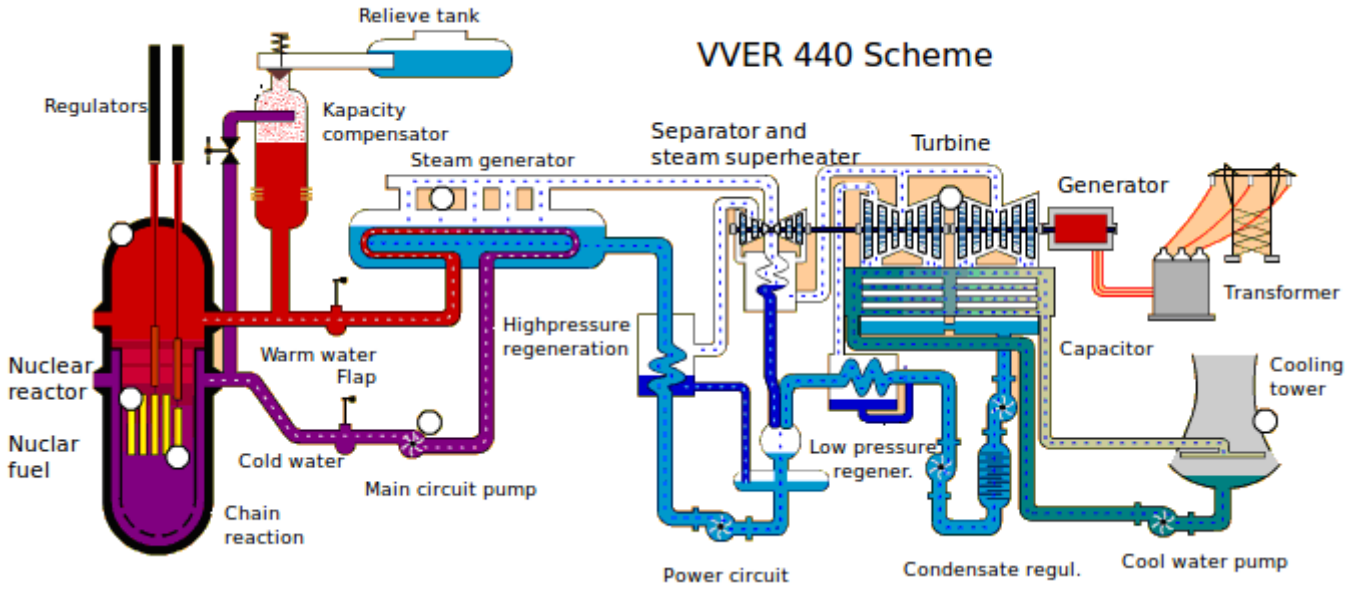


Fig. 11. Nuclear plant scheme - reactor VVER 440 Scheme