Formal Neuron Based on Adaptive Parametric Rectified Linear Activation Function and its Learning

Yevgeniy Bodyanskiy[1][0000-0001-5418-2143], Anastasiia Deineko[2][0000-0002-3279-3135],

Iryna Pliss[1][0000-0001-7918-7362] and Valeriia Slepanska[2][0000-0002-0465-8593]

[1] Control systems research laboratory, [2] Artificial Intelligence department,
Kharkiv National University of Radio Electronics, Kharkiv, Ukraine
yevgeniy.bodyanskiy@nure.ua
anastasiya.deineko@gmail.com
iryna.pliss@nure.ua
valeriia.slepanskaia@gmail.com

Abstract. The paper proposes an adaptive activation function (AdPReLU) for deep neural networks which is generalization of rectified unit family, differing by opportunity of online tuning its parameters during the learning process of neural network. The learning algorithm of formal neuron with adaptive activation function which is generalization of delta-rule and in which parameters of the function tune simultaneously with synaptic weights, based on error back-propagation is developed. The proposed algorithm of tuning is optimized for increasing of operating speed. Computational experiments confirm the effectiveness of the approach under consideration.

Keywords: deep neural network, adaptive activation function, delta-rule, synaptic weights, rectified linear unit, learning algorithm.

1 Introduction

At the present time artificial neural networks are widely used for solving Data Science tasks, due to their possibility to tune parameters and architecture during the process of information processing and their universal approximative abilities. These properties provide effective solving the tasks of pattern recognition (classification), time series processing (prediction), complex non-linear objects and processes emulation (identification and adaptive control).

The most widely used are multilayer perceptrons whose nodes-neurons usually are the Rosenblatt`s elementary perceptrons with sigmoidal activation functions. Besides traditional σ-functions [1] the most widespread are tanh, SoftSign, Satlin [2, 3], polynomial activation functions of special type [4] and another squashing functions.

Based on the classical multilayer perceptrons the deep neural networks (DNN) were created [5-8]. This has led to increasing of processing images, audio signals, arbitrary time series, and intelligent text analysis effectiveness. However, there are

2019 DCSMart Workshop.

significant computational problems connected with to the so-called vanishing and exploding gradients connected with specific form of sigmoidal activation functions.

Consequently, so called rectified unit family [9] is used in DNN as activation functions. There can be noted such functions as leaky rectified linear unit (LReLU), parametric restified linear unit (PReLU), randomized leaky rectified linear unit (RReLU), noisy rectified linear unit (NReLU), exponential linear unit (ELU) [7-12], except rectified linear unit (ReLU) itself.

The functions listed above are piecewise linear functions with fixed parameters chosen by empirical considerations. The advantage is that their derivatives do not vanish, so they overcome the problem of vanishing gradient and permit to optimize the speed of learning process. However, these functions do not satisfy G.Cybenko`s theorem [1] conditions, so for providing required quality of approximation it is necessary to increase the number of hidden layers in the DNN. It causes increasing of DNN`s computational complexity and learning process speed decreasing.

Accordingly, it is expedient to introduce in consideration adaptive parametric rectified linear activation function (AdPReLU) within rectified unit family, whose parameters can tune during learning process like usual neuron`s synaptic weights do, optimizing adopted learning criterion and improving approximating properties both individual neuron and neural network in general.

2 Architecture of Neuron with Adaptive Parametric Rectified Linear Activation Function

Rosenblatt's perceptron as node of any neural network implements nonlinear mapping as:

$$\hat{y}_j(k) = \psi_j\left(\theta_{j0} + \sum_{i=1}^{n} w_{ji} x_i(k)\right) =$$

$$= \psi_j\left(\sum_{i=0}^{n} w_{ji} x_i(k)\right) = \psi_j\left(w_j^T x(k)\right) = \psi_j(u_j(k))$$

where $\hat{y}_j(k)$ - output signal of j-th neuron of network in the moment of discrete time $k = 1,2,\ldots$; $x(k) = \left(1, x_1(k),\ldots,x_i(k),\ldots,x_n(k)\right)^T \in R^{(n+1)}$ – input vector signal, $\theta_{j0} \equiv w_{j0}$ – bias signal, $w_j = \left(w_{j0}, w_{j1},\ldots,w_{ji},\ldots,w_{jn}\right)^T \in R^{(n+1)}$ – synaptic weights vector, adjusting in learning process, $u_j(k)$ – signal of internal activation, $\psi_j(\cdot)$ – activation function of j-th neuron, chosen usually by empirical considerations during the process of learning and functioning of neural network.

Thus, in the Cybenko's theorem σ-function is used:

$$\hat{y}_j(k) = \psi_j(u_j(k)) = \frac{1}{1 + \exp\left(-\gamma_j(u_j(k))\right)} \tag{1}$$

where γ_j is a gain parameter which determines the form of this function.

It should be noticed that derivative of sigmoidal function has the form

$$\psi_j(u_j(k)) = \gamma_j \hat{y}_j(k)\left(1-\hat{y}_j(k)\right)$$

that means that it has a form of bell-shaped function. Therefore, the more value of $\hat{y}_j(k)$ is closer to 0 or 1, the closer the value of derivative to 0, which originates the vanishing gradient.

In general form, rectified unit family can be written as

$$\psi_j(u_j(k)) = \begin{cases} u_j(k) if\ u_j > 0, \\ a_j u_j(k) otherwise \end{cases} \quad (2)$$

where the a_j parameter is chosen by empirical considerations and stays constant during the learning process. In the standart ReLU a_j equal to 0, so:

$$\psi_j(u_j(k)) = 0 \text{ if } u_j(k) < 0.$$

This may lead to learning process being frozen because of negative values of internal activation function`s signal.

The generalization of activation function (2) has the form

$$\psi_j(u_j(k)) = \begin{cases} a_j^R u_j(k) \ if\ u_j > 0, \\ a_j^L u_j(k) otherwise, \end{cases} \quad (3)$$

however, there is a problem with the a_j^R and the a_j^L parameters` values choosing. So, the solution is to introduce extra procedure of tuning these parameters to the neuron`s learning process. This makes the learning process more sophisticated and leads to necessity to tune n+3 parameters instead of n+1 adjustable parameters which are within the w_j vector. In spite of that, improvement of approximating properties is provided, due to the fact that (3) can be performed in different forms, for example:

$$\psi_j(u_j(k)) = |u_j(k)|.$$

There is the schema of neuron with adaptive parametric rectified linear activation function (shown on fig. 1) where parameters w_j, a_j^R, a_j^L are tuned during the learning process.

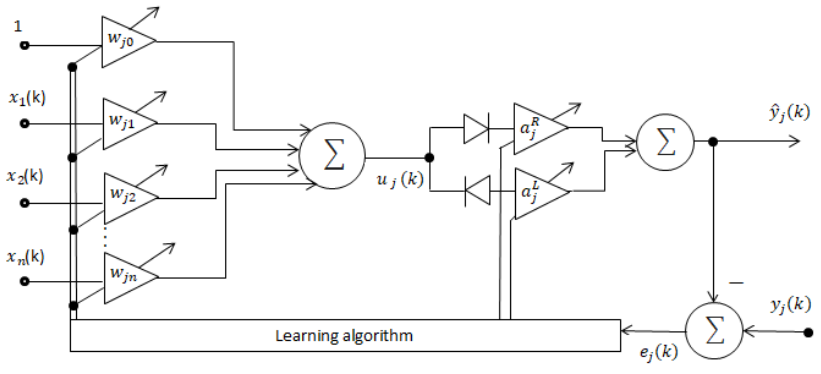

Fig. 1. Neuron with adaptive parametric rectified linear unit (AdPReLU).

In Fig. 1 – $y_j(k)$ – external reference signal, $e_j(k) = y_j(k) - \hat{y}_j(k) = y_j(k) - \psi_j\left(u_j(k)\right)$ – learning error.

3 Learning Procedure

As learning criterion standard quadratic function is used in the form:

$$E_j(k) = \frac{1}{2}e_j^2(k) = \frac{1}{2}\left(y_j(k) - \psi_j\left(u_j(k)\right)\right)^2 = \frac{1}{2}\left(y_j(k) - \psi_j\left(\sum_{i=0}^{n} w_{ji}x_i(k)\right)\right)^2.$$

Its minimization by gradient procedure leads to algorithm of synaptic weights` tuning that can be written in the form:

$$w_{ji}(k) = w_{ji}(k-1) - \eta(k)\frac{\partial E_j(k)}{\partial e_j(k)}\frac{\partial e_j(k)}{\partial w_{ji}} = w_{ji}(k) - \eta(k)e_j(k)\frac{\partial e_j(k)}{\partial w_{ji}} =$$

$$= w_{ji}(k-1) - \eta(k)e_j(k)\frac{\partial e_j(k)}{\partial u_j(k)}\frac{\partial u_j(k)}{\partial w_{ji}} =$$

$$= w_{ji}(k-1) + \eta(k)e_j(k)\psi_j(u_j(k))x_i(k) =$$

$$= w_{ji}(k-1) + \eta(k)\delta_j(k)x_i(k)$$

or in the vector form:

$$w_j(k) = w_j(k-1) + \eta(k)\delta_j(k)x_i(k)$$

where $\eta(k)$ – is a learning rate parameter, $\delta_j(k) = e_j(k)\psi_j'\left(u_j(k)\right)$ – δ-error.

For standard hyperbolic tangent function it can be written as:

$$\frac{\partial\psi(u_j)}{\partial u_j} = \gamma_j\left(1 - \left(\tanh\gamma_j u_j\right)^2\right) = \gamma_j\left(\operatorname{sech}\gamma_j u_j\right) = \gamma_j\left(1 - \hat{y}_j^{\,2}\right)$$

$$w_j(k) = w_j(k-1) + \eta(k)e_j(k)\gamma_j\left(1 - \hat{y}_j^{\,2}(k)\right)x(k). \quad (4)$$

Obviously, if $\hat{y}_j(k) \to \pm 1$ «vanishing gradient» effect is appeared. For improvement of algorithm (4) convergence in [13] it was proposed to tune gain parameter γ_j according to the procedure:

$$\frac{\partial\psi(u_j)}{\partial u_j} = \gamma_j\left(1 - \left(\tanh\gamma_j u_j\right)^2\right) = \gamma_j\left(\operatorname{sech}\gamma_j u_j\right) = \gamma_j\left(1 - \hat{y}_j^{\,2}\right)$$

$$w_j(k) = w_j(k-1) + \eta(k)e_j(k)\gamma_j\left(1 - \hat{y}_j^{\,2}(k)\right)x(k). \quad (5)$$

that also leads to «vanishing gradient» effect.

Neuron`s learning scheme, which is shown in Fig. 1, using backpropagation procedure, begins with tuning of a_j^R and a_j^L parameters. To simplify the transformations lets skip the R and L indexes temporarily.

Then

$$a_j(k)=a_j(k-1)-\eta_a(k)\frac{\partial E_j(k)}{\partial a_j}=a_j(k-1)+$$
$$+\eta_a(k)\big(y_j(k)-a_j(k-1)u_j(k)\big)u_j(k)= \tag{6}$$
$$=a_j(k-1)+\eta_a(k)\big(y_j(k)-a_j(k-1)w_j^T(k-1)x(k)\big)w_j^T(k-1)x(k).$$

The parameter learning process of AdPReLU activation function (6) can be optimized for the increasing of operating speed. So it can be provided following transformations:

$$a_j(k)=a_j(k-1)+\eta_a(k)\big(y_j(k)-a_j(k-1)u_j(k)\big)u_j(k),$$
$$a_j(k)u_j(k)=a_j(k-1)u_j(k)+\eta_a(k)\big(y_j(k)-a_j(k-1)u_j(k)\big)u_j^2(k),$$
$$y_j(k)-a_j(k)u_j(k)=y_j(k)-a_j(k-1)u_j(k)-\eta_a(k)e_j(k)u_j^2(k),$$
$$\tilde{e}_j(k)=e_j(k)-\eta_a(k)e_j(k)u_j^2(k),$$
$$\tilde{e}_j^{\,2}(k)=e_j^{\,2}(k)-2\eta_a(k)e_j^2(k)u_j^2(k)+\eta_a^2(k)e_j^2(k)u_j^4(k),$$
$$\frac{\partial\tilde{e}_j^{\,2}(k)}{\partial\eta_a}=-2e_j^{\,2}(k)u_j^2(k)+2\eta_a(k)e_j^2(k)u_j^4(k)=0,$$

which suggest that optimal value of learning rate parameter $\eta_a(k)$ is determined by expression:

$$\eta_a(k)=u_j^{-2}(k). \tag{7}$$

Then by substitution (7) into (6) and returning to R and L indexes the next result was gotten:

$$\begin{cases} a_j^R(k)=a_j^R(k-1)+\big(y_j(k)-a_j^R(k-1)u_j(k)\big)u_j^{-1}(k)\,if\,u_j(k)>0, \\ a_j^L(k)=a_j^L(k-1)+\big(y_j(k)-a_j^L(k-1)u_j(k)\big)u_j^{-1}(k)\,otherwise. \end{cases} \tag{8}$$

After a_j^R and a_j^L parameters` are tuned, it can be possible to return to w_j synaptic weights learning. In this case the learning criterion is based on $\hat{e}_j(k)$ error, i.e.:

$$\tilde{E}_j(k)=\frac{1}{2}\tilde{e}_j^{\,2}(k)=\frac{1}{2}\big(y_j(k)-a_j(k)w_j^Tx(k)\big)^2. \tag{9}$$

The gradient minimization (9) by w_j leads to the procedure:

$$w_j(k)=w_j(k-1)-\eta(k)\nabla_{w_j}\tilde{E}_j(k)=w_j(k-1)+\eta(k)\tilde{e}_j(k)a_j(k)x(k)=$$
$$=w_j(k-1)+\eta(k)\big(y_j(k)-a_j(k)w_j^T(k-1)x(k)\big)a_j(k)x(k)= \tag{10}$$
$$=w_j(k-1)+\eta(k)\big(y_j(k)-w_j^T(k-1)\tilde{x}(k)\big)\tilde{x}(k)$$

where $\tilde{x}(k)=a_j(k)x(k)$.

It`s simple to notice, that algorithm (10) is basically a learning procedure of neuron-Adaline [2], what means that it can be optimized by operating speed. As a result, we obtain optimized one-step Kaczmarz-Widrow-Hoff learning algorithm [14, 15] in the form:

$$w_j(k) = w_j(k-1) + \frac{y_j(k) - w_j^T(k-1)\tilde{x}(k)}{\left\|\tilde{x}(k)\right\|^2}\tilde{x}(k) = \\ = w_j(k-1) + \tilde{e}_j(k)\tilde{x}^{+T}(k) \quad (11)$$

where $(\cdot)^+ -$ is a symbol of pseudoinversion.

For preventing from "exploding gradient", regularized version of (11) can be considered:

$$w_j(k) = w_j(k-1) + \left(\tilde{x}(k)\tilde{x}^T(k) + \alpha I\right)^{-1}\tilde{e}_j(k)\tilde{x}(k)$$

where $\alpha > 0$ – is a momentum term. Using matrix inversion lemma we can finally obtain expression:

$$w_j(k) = w_j(k-1) + \frac{\tilde{e}_j(k)\tilde{x}(k)}{\alpha + \left\|\tilde{x}(k)\right\|^2},$$

that coincides with the additive form of Kaczmarz`s algorithm.

For providing additive filtering properties to learning algorithm (12), the procedure [16-18] can be used:

$$w_j(k) = w_j(k-1) + \frac{\tilde{e}_j(k)\tilde{x}(k)}{r(k)} = w_j(k) = w_j(k-1) + \frac{\tilde{e}_j(k)\tilde{x}(k)}{\beta r(k-1) + \left\|\tilde{x}(k)\right\|^2}$$

(where $0 \le \beta \le 1$ – is a forgetting factor), which coincides with algorithm (11) if β=0. However, if β=1 it coincides with stochastic approximation algorithm of Goodwin-Ramadge-Caines [19], which provides convergence in the conditions of stochastic disturbances and noises.

Consequently, the resulting synaptic weights learning procedure can be written as:

$$w_j(k) = \begin{cases} w_j(k-1) + \dfrac{\left(y_j(k) - a_j^R(k)w_j^T(k-1)x(k)\right)a_j^R(k)x(k)}{r^R(k)}, \\ r^R(k) = \beta r^R(k-1) + (a_j^R(k))^2\left\|x(k)\right\|^2 \text{ if } w_j^T(k-1)x(k) > 0, \\ w_j(k-1) + \dfrac{\left(y_j(k) - a_j^L(k)w_j^T(k-1)x(k)\right)a_j^L(k)x(k)}{r^L(k)}, \\ r^L(k) = \beta r^L(k-1) + (a_j^L(k))^2\left\|x(k)\right\|^2 \text{ otherwise.} \end{cases} \quad (13)$$

Algorithms (8), (13) describe learning process of neuron with adaptive parametric rectified linear activation function in general.

4 Computer Experiments

To demonstrate the efficiency of the proposed neuron and its learning procedure it was implemented a simulation test based on approximation of reference signal $y_j(k)$ defined by expression:

$$y_j(k) = \tanh\left(0,1x_1(k) + 0,2x_2(k) + 0,3x_3(k) + 0,4x_4(k)\right) = \tanh\left(u_j(k)\right)$$

where $x_i(k)-$ is a uniformly distributed random variable on the interval: $-1 \le x_i(k) \le 1$. The results of the proposed approach were compared with the results obtained using a neuron-Adaline, neuron with standart ReLU activation function and neuron with classical $\tanh\left(u_j(k)\right)$ activation function.

In Fig.2 it is shown how the mean square error is changing

$$\overline{e_j}^2(N) = \frac{1}{N}\sum_{k=1}^{n} e_j^2(N-1) = \overline{e_j}^2(N-1) + \frac{1}{N}\left(e_j^2(N) - \overline{e_j}^2(N-1)\right).$$

In this experiment the best results were gotten by neuron with AdPReLU activation function. So it surpasses neuron with Adaline, neuron with ReLU and another one with $\tanh\left(u_j(k)\right)$. As reference signal was chosen such expressions as:

$$y_j(k) = \sin\left(0,5\pi u_j(k)\right),$$

$$y_j(k) = \begin{cases} \tanh u_j(k), if\ u_j(k) > 0, \\ u_j^3(k), othherwise, \end{cases}$$

$$y = \tanh\left(u_j(k)\right)$$

the proposed neuron also overperforms Adaline, ReLU and $\tanh\left(u_j(k)\right)$.

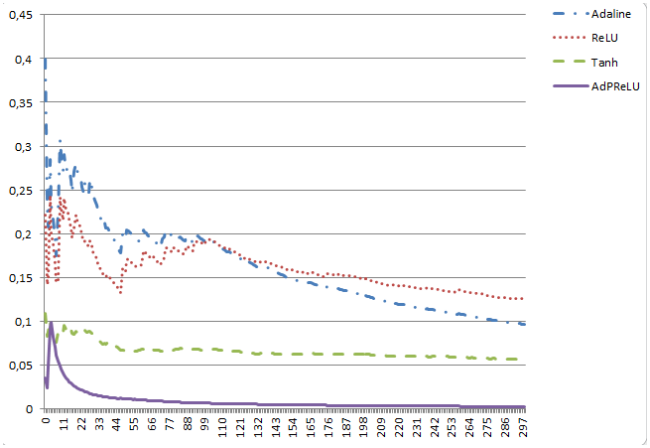

Fig. 2. Convergence curves for training neurons with functions: $y_j(k) = \sin\left(0,5\pi u_j(k)\right)$;

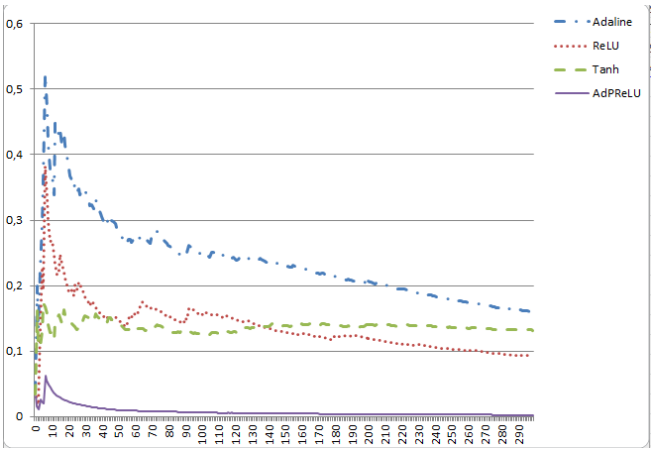

Fig. 3. Convergence curves for training neurons with functions

$$y_j(k)=\begin{cases}\tanh u_j(k), if\ u_j(k)>0;\\ u_j^{\,3}(k), otherwise;\end{cases}$$

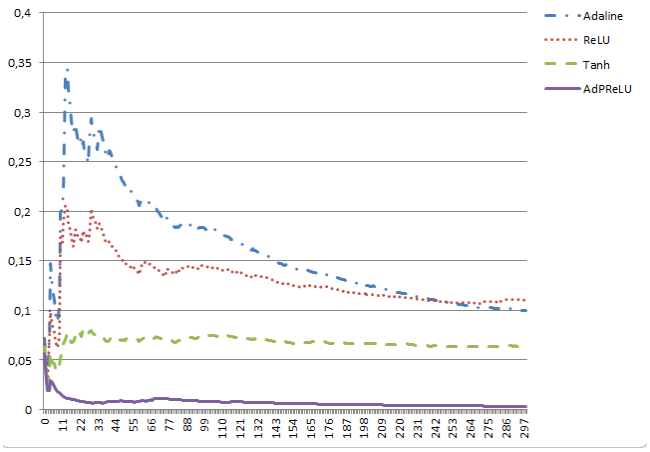

Fig. 4. Convergence curves for training neurons with functions $y=\tanh\left(u_j(k)\right)$.

5 Conclusion

In this paper, formal neuron of neural network with adaptive activation function, whose parameters tune simultaneously with synaptic weights, is introdused. Proposed activation function is generalization of rectified unit family and provides improvements of approximating properties. Usage of AdPReLU in deep neural networks prevents the learning process from “vanishing and exploding gradients”. In spite of this the proposed algorithms of tuning are optimized for the increasing of operating speed,

i.e. they significantly reduce the learning time of the network in general. Computational experiments confirm the effectiveness of the proposed approach.

References

1. Cybenko, G.: Approximating by superposition of a sigmoidal function, Math.Contr. Sign. Syst, vol. 2, pp. 303–314, (1989).
2. Cichocki, A., Unbehauen, R.: Neural Networks for Optimization and Signal Processing , Stuttgart: Teubner, (1993).
3. Hornik, K.: Approximation capabilities of multilayer feedforward networks, Neural Networks, vol. 4, pp. 251–257, (1991).
4. Bodyanskiy, Ye.V, Kulishova, N.Ye, Rudenko, O.G.: One model of formal neuron, Reports of National Academy of Sciences of Ukraine, vol. 4, pp. 69–73, (2001).
5. Bengio, Y, LeCun, Y, Hinton, G.: Deep Learning, Nature, vol. 521, pp.436–444, (2015).
6. Schmidhuber, J.: Deep learning in neural networks: An overview, Neural Networks, vol. 61, pp. 82–117, (2015).
7. Goodfellow, I., Bengio, Y., Courville, A.: Deep Learning, MIT Press, (2016).
8. Graupe, D.: Deep Learning Neural Networks: Design and Case Studies, New Jersey: World Scientific, (2016).
9. Xu, B., Wang, N., Chen, T., Li, M.: Empirical evaluation of rectified activations in convolution network, arXiv preprint arXiv, 1505.00853, (2015).
10. He, K., Zhang, X, Ren, S., Sun ,J.: Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification, Proc. IEEE Int. Conf. on Computer Vision, arXiv prrprint arXiv: 1502.01852.2015, pp.1026–1034, (2015).
11. Clevert, D-A., Unterhiner, T., Hochreiter, S.: Fast and accurate deep network learning by exponential linear units (ELUs), arXiv preprint arXiv: 1511.07289, (2015).
12. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition, Proc. IEEE Int. Conf. on Computer Vision and Pattern Recognition (CVPR), pp. 770–778, (2016).
13. Kruschke, J.K., Movellan, J.R.: Benefits of gain: speeded learning and minimal layers backpropagation networks, IEEE Trans. on Syst., Man, and Cybern, vol. 21, pp. 273–280, (1991).
14. Kaczmarz, S.: Approximate solution of systems on linear equations, Int. J. Control, vol. 53, pp. 1269–1271, (1993).
15. Widrow, B., Hoff, Jr. M. E.: Adaptive switching circuits, IRE Western Electric Show and Connection Record, Part 4, pp. 96–104, (1960).
16. Bodyanskiy, Ye.V., Pliss, I.P., Solovyova, T. V.: Multistep optimal predictors of multivariable non-stationary stochastic processes, Reports of Academy of Sciences of USSR, vol. 12, pp. 47–49, (1986).
17. Bodyanskiy, Ye., Kolodyazhniy, V., Stephan A.: An adaptive learning algorithm for a neuro-fuzzy network, Ed. by B. Reusch “Computational Intelligence. Theory and Applications”, Berlin Heidelberg: Springer-Verlag, pp. 68–75, (2001).
18. Otto, P., Bodyanskiy, Ye., Kolodyazhniy, V.: A new learning algorithm for a forecasting neuro-fuzzy network, Integrated Computer Aided Engineering, vol. 10, №4, pp. 399–409, (2003).
19. Goodwin G. C., Ramadge P. J., Caines P. E.: A globally convergent adaptive predictor, Automatica, vol. 17, pp.135–140, (1981).