DOI 10.2478/ama-2020-0017

PREDICTIVE NEURAL NETWORK IN MULTIPURPOSE SELF-TUNING CONTROLLER

Oleksiy BONDAR*

*Electronic Microwave Devices Department, Institute of Radio Astronomy of the National Academy of Sciences of Ukraine,
4 Mystetstv St., Kharkiv, 61002, Ukraine

bondarrian@gmail.com

received 16 April 2020, revised 20 July 2020, accepted 22 July 2020

Abstract: A very important problem in designing of controlling systems is to choose the right type of architecture of controller. And it is always a compromise between accuracy, difficulty in setting up, technical complexity and cost, expandability, flexibility and so on. In this paper, multipurpose adaptive controller with implementation of artificial neural network is offered as an answer to a wide range of tasks related to regulation. The effectiveness of the approach is demonstrated by the example of an adaptive thermostat. It also compares its capabilities with those of classic PID controller. The core of this approach is the use of an artificial neural network capable of predicting the behaviour of controlled object within its known range of parameters. Since such a network, being trained, is a model of a regulated system with arbitrary precision, it can be analysed to make optimal management decisions at the moment or in a number of steps. Network learning algorithm is backpropagation and its modified version is used to analyse an already trained network in order to find the optimal solution for the regulator. Software implementation, such as graphical user interface, routines related to neural network and many other, is done using Java programming language and Processing open-source integrated development environment.

Keywords: Artificial neural network, adaptive regulator, backpropagation algorithm, system modelling

1. INTRODUCTION

Nowadays, the use of artificial neural networks in control processes and other activities is a very popular area of research. And this is understandable, since these structures are a good tool in the modelling of complex systems, for which it is difficult to find a simple mathematical solution. It should also be remembered that artificial neural networks are capable of providing high quality data processing even in the conditions of their incompleteness. Such a predicting model is successfully being using for weather forecasting in short term for some localities (Wica et al., 2019). It can also be useful to provide neural classification mechanism in genetics researches (Liu et al., 2019; MacLean, 2019). The usage of artificial neural network (hereinafter referred to as ANN) is a reasonable solution for power flow regulators (Ma et al., 2018). Also, ANN can be used as a tool for detecting stable equivalent series resistance (ESR) in voltage regulator characterization (Zaman et al., 2018), in mechatronic hydraulic drive regulation (Burennikov et al., 2017) or autopilot (Zhao et al., 2018). However, researchers are particularly curious about the possibility of using artificial neural networks in the automatic tuning of PID regulators (Ayomoh and Ajala, 2012; Hernández-Alvarado et al., 2016; Pirabakaran and Becerra, 2002; Zhang et al., 2016; Du et al. 2018; Han et al., 2017). The method proposed in this paper excludes the PID section from the controller system. As will be shown, a trained neural network with a multi-step error estimation module is sufficient for high quality control of a wide range of systems providing flexible controls. Moreover, the network does not require any specific or detailed data, rather accurate system data in the regulatory range. To demonstrate the principle operation of the multipurpose controller as thermostat, a software model of the solid body temperature under the influence of external factors was created. Such an example is simple to understand and easily portable to a real thermostat or to another type of system. For the software implementation, Processing integrated development environment (IDE) was selected, which uses a Java programming language. It speeds up and facilitates both writing and debugging of programs for which it is important to have as many graphical evaluation options as possible.

2. DESCRIPTION OF MULTIPURPOSE NEURONAL NETWORK-BASED CONTROLLER

In order to understand the principle of operation of a multipurpose adaptive controller, it is necessary to consider its work in stages with a detailed analysis of the work of each element. We list these stages. Primary training of predictive ANN, during which the regulator acquires the ability to assess the state of regulated object in the future, based on the state of that object at the present moment. Further, the already trained ANN is embedded in a multi-step planning mechanism, where, thanks to the neural network's ability to approximate functions, we obtain a model of the object's behaviour depending on the actions of the controller itself on a certain number of steps forward. The predicted sequence of states of the control object over time is used to estimate the error relative to the desired control result for each of these states. Depending on the approach chosen in the assessment, these error values are used to optimize the controller's actions in the appropriate way. First of all, we will look

at the core of the whole system – one-step predictive artificial neural network.

2.1. One-step predictive artificial neural network

The module is so called Feedforward Backpropagation Neural Network. Quite traditional architecture of network with neurons (knots) grouped into layers (see Fig.1). There are three types of layers: input, hidden, and output. The input layer is a set of corresponding signals that the module must process to obtain the output signal. In this case, the following signals are: RV – regulated value, RA – regulator action. Regulated value is the parameter that should be controlled such as temperature (for thermostats), speed or torque (for motor controller), distance, pressure or voltage and so on. Regulator action is any relevant action that affects a regulated value such as change of voltage, PWM, power, or any other parameter that the regulator may manipulate during control.

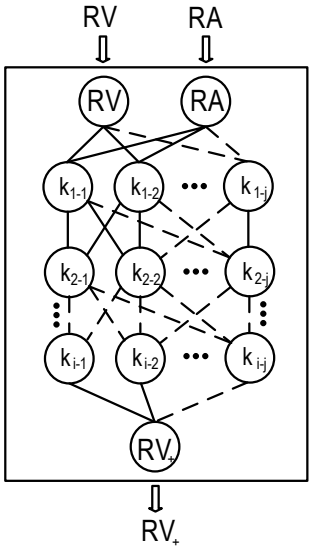

Fig. 1. The scheme of one-step predictive ANN

Hidden layers can contain a certain number of neurons in each layer and consist of a certain number of layers. The number of hidden layers depends on the complexity of the behaviour of the control object, but in most cases, one hidden layer is sufficient (Heaton, 2008). The number of neurons in each layer is defined as a balance between the precision with which the neural network as a model is able to reproduce the behaviour of the system on which it is trained and the complexity of implementing such a system in a software sense. Each additional neuron in the layer increases the accuracy with which the system can make predictions as a whole, but it also increases the computational power requirements of the equipment on which this network must operate. For each neuron of the hidden or output layers, the internal state of the neuron is first calculated based on the signals coming from the previous layer. As it was stated, the prediction module trains with the backpropagation algorithm, but there are no restrictions on using other algorithms such as genetic or imperialist competitive or others (Elsisi, 2019). Equation 1 shows how to calculate the internal state of the hidden or output neuron in layer with i neurons and previous layer with l neurons. All other neurons in this layer are calculated similarly by replacing the corresponding index i with the desired one.

$$state_{k_i} = (\sum_l (w_{li} \cdot out_l)) + b_{k_i} \quad (1)$$

where: $state_{k_i}$– internal state of neuron in a row of 1 to i of hidden or output layer, l – the number of neurons in the previous layer, w_{li} – the weight of the connection between the calculated neuron and the output of neuron in a row of 1 to l from the previous layer (out_l), b_{k_i} – offset of the calculated neuron, makes it possible for the internal state of the neuron not to be 0 when the input signals are 0. After that, the output of the neuron is calculated. To do this, the resulting internal state of the neuron must be passed through the activation function. There are different variants of the activation function, but in this paper, a linear activation function is used for the output neuron, and a fast sigmoid (https://stackoverflow.com/questions/10732027/fast-sigmoid-algorithm) for the hidden layers. The linear activation function simply transmits to the output of the neuron its internal state and allows the output value of the neuron to not be limited by any asymptotes in its range. Which is useful for a network whose output values may reach values that other activation functions do not allow. The sigmoid, as a function of activating the hidden layers, on the contrary, limits the output values in the range from 0 to 1 and provides nonlinearity in the operation of the network. Nonlinearity is necessary where the behaviour of the system to be simulated cannot be reduced to the sum of linear functions, which is any more or less complex system. The fast sigmoid (see Eq. 2) is close to the original but significantly reduces the time to calculate the activation function since it does not require a floating point exponent.

$$out_{k_i} = \begin{cases} \frac{0.5}{1+state_{k_i}^2}, & when \quad state_{k_i} < 0 \\ 1 - \frac{0.5}{1+state_{k_i}^2}, & when \quad state_{k_i} \geq 0 \end{cases} \quad (2)$$

where: out_{k_i} – the output value of the neuron in a row of 1 to i of some hidden layer (from 1 to j, see Fig.1).

The output layer of this model consists of one neuron and forms the final result of the network. The output RV_+ itself is the prediction of a regulated value (RV) on the next step of regulation. It is worth noting that such a network may also contain some additional inputs and outputs, which on the one hand allow to expand the number of parameters being monitored for regulation, and on the other hand create additional opportunities for optimizing the controller's actions. But it should also be remembered that increasing the number of inputs and outputs of the network requires an increase in its complexity, and thus, the requirements for hardware to process it (Heaton, 2008).

2.2. Multi-step planning mechanism

After considering the one-step predictive ANN as a basic element of the planning mechanism, we move on to the multi-step mechanism as a whole.

As can be seen from Fig. 2, the mentioned mechanism is composed as a sequence of one-step forecasts made using the previously described neural network. Moreover, since the trained network is capable of making predictions for the original managed object within the range of values in which it was trained, only one forecasting network is needed for any long-range forecast in time. The first step in forecasting is based on the current data of regulated value (RV_0) and regulator action (RA_0). After this, the

DOI 10.2478/ama-2020-0017

forecast for the first adjustment step RV_1 becomes the input value for the prediction of the next step RV_2, and so on, down to some step n which limits the planning horizon. Increasing the planning horizon on one hand improves the dynamic characteristics of the controller, such as stability and the absence of overshoots in operation, on the other hand, it increases the time to calculate all the steps. Regarding the values of future control actions $(RA_1 \ldots RA_{n-1})$, prior to optimization, they may be equal to the current control or may be random in a certain range. Further, all the results predicted at each step are sent for estimation of a regulation error in the corresponding module.

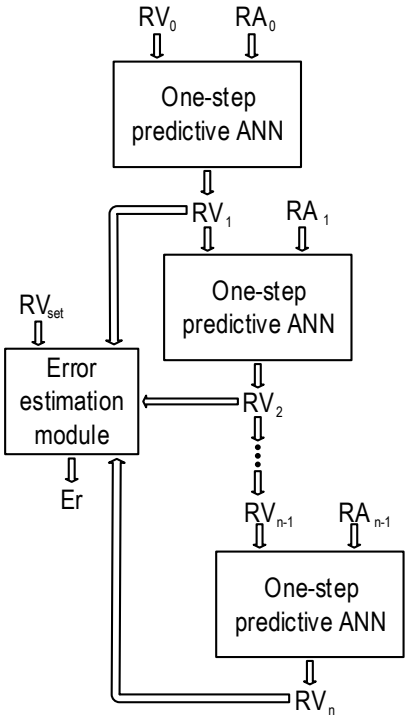

Fig. 2. The scheme of multi-step planning mechanism

2.3. Error estimation module

As the proposed multi-step controller must be able to make current decisions based on certain predictions of both behaviour of the system it manages and its actions, it must also adequately assess how its actions are approaching in the future achievement of the goal, avoiding oscillations and overshoots. This function is performed by the error estimation module. Taking into account the target value to be reached by the control object (RV_{set}), the controller calculates an error for each predicted step. The equation to calculate the regulation error in the first step of forecasting is:

$$Er_1 = \frac{(RV_1 - RV_{set})^2}{2} \tag{3}$$

where: Er_1 – regulation error in the first step of forecasting.

It should also be remembered that each new step of the prediction is based on the previous one, and the error of the neural network modelling of the source system will accumulate. Therefore, the regulator should not equally evaluate the result at each step. Equation 4 shows how it calculates the values of the regulation errors in the range from 2nd to n-th steps.

$$Er_n = \frac{(RV_n - RV_{set})^2}{2 \cdot (n-1) \cdot d} \tag{4}$$

where: Er_n – regulation error in the n-th step of forecasting, n – step number, d – initial depreciation, RV_n – regulated value in the n-th step of forecasting. As we can see, from the second step, the significance of each subsequent regulation error decreases in arithmetic progression. The initial depreciation is the factor in how many times the second forecast is less important for the regulator than the first one. This approach helps to increase the stability of the regulator and indicate that the first step is the most important to control because it influences further events by domino effect.

3. THE CONTROL ALGORITHM

After a schematic description of the proposed regulator, we proceed to the disclosure of the algorithm of its operation. First of all, let's say that the given controller can work on different algorithms of optimization of control action. This paper describes one such algorithm. A common feature of such algorithms is their similarity to the backpropagation method in neural network training. They are also iterative and use a gradient descent method with dividing into the forward and backward pass. But if backpropagation in a step-by-step training changes the parameters of the neural network, then the control algorithms below do not change the network itself, but use it to detect a correlation between the action of the controller and the error of regulation. Mathematically, the optimization step in general is shown in Equation 5.

$$RA' = RA - \lambda \cdot \frac{\partial Er}{\partial RA} \tag{5}$$

where: RA' – regulator action after one optimisation step, λ – optimization rate, Er – predicted regulation error (depends on the optimization strategy). Thus, at each step of the optimization of the control action, it changes by a value proportional to the instantaneous speed of change of the predicted regulation error with the change of the regulatory action. Moreover, λ should be small enough to ensure the smoothness and accuracy of the process, but not too small, as this will require a large number of iterations for a successful result. Further work is to calculate $\frac{\partial Er}{\partial RA}$. According to the chain rule, this task can be divided into two simpler ones (see Eq. 6).

$$\frac{\partial Er}{\partial RA} = \frac{\partial Er}{\partial RV} \cdot \frac{\partial RV}{\partial RA} \tag{6}$$

where: RV – predicted regulated value. So, calculation of $\frac{\partial Er}{\partial RV}$ allows the error estimation module. And the forecasting network allows to calculate $\frac{\partial RV}{\partial RA}$. But for this, we need to decide on a strategy by which we will evaluate regulatory errors and optimize the regulation action. So, let's describe the strategy of optimization of the maximum predicted regulation error.

3.1. Maximum Predicted Error Reduction Strategy (MAPERS)

The essence of the Maximum Predicted Error Reduction Strategy (hereinafter referred to as MAPERS) is to look for that step in the planning mechanism that predicts more regulation error than others, then change the planned action in this step

to the side, which should reduce the error in that step (see Equation 5).

As can be seen from Fig. 3, in each forward pass of the optimization cycle, the multi-step planning mechanism generates a chain of forecasts from RV_1 to RV_n, which together with RV_{set} go to the input of the error estimation module. Here, the error of regulation of each of the predicted steps is calculated separately by Equations (3) and (4), after which the step with the maximum error is selected from all the steps (Er_{max}). The Er_{max} computation ends the forward pass and the back pass begins, the purpose of which is to change the step with the predicted Er_{max} to reduce it. The way of back pass is shown in the Fig. 4 (arrow pointing from Er_{max} to RA_{max}).

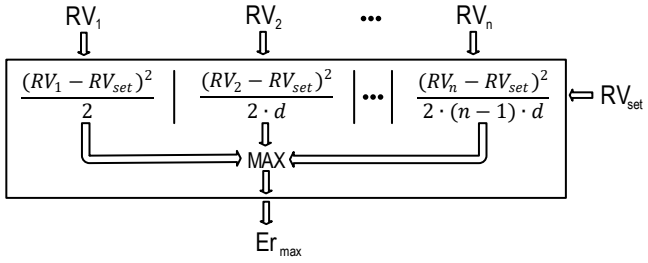

Fig. 3. Schematic diagram of the error estimation module for the Maximum Predicted Error Reduction Strategy (forward pass)

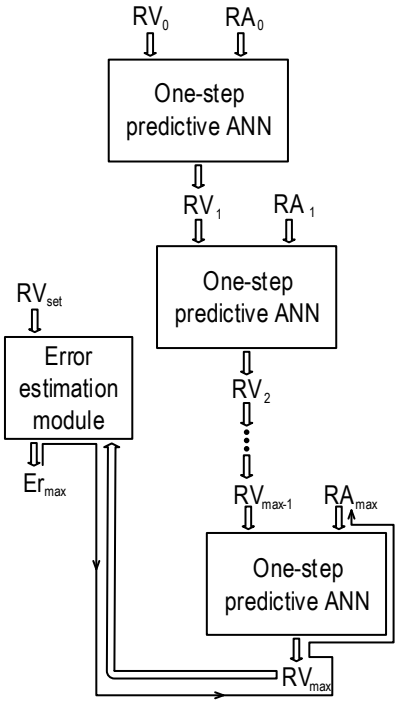

Fig. 4. Schematic diagram of the way of back pass for MAPERS

Therefore, Equation 5 for MAPERS can be rewritten according to the Fig. 4 as:

$$RA'_{max} = RA_{max} - \lambda \cdot \frac{\partial Er_{max}}{\partial RA_{max}} \tag{7}$$

where: RA_{max} – regulator action in step with maximum value of predicted regulation error.

We can also rewrite Equation 6 for MAPERS as:

$$\frac{\partial Er_{max}}{\partial RA_{max}} = \frac{\partial Er_{max}}{\partial RV_{max}} \cdot \frac{\partial RV_{max}}{\partial RA_{max}} \tag{8}$$

where: RV_{max} – predicted regulated value in step with maximum value of predicted regulation error.

To calculate $\frac{\partial Er_{max}}{\partial RV_{max}}$, we need to refer to Equations 3 and 4, mentioned earlier. After calculations, we get the corresponding derivatives:

$$\frac{\partial Er_{max}}{\partial RV_{max}} = \begin{cases} RV_1 - RV_{set}, \ if \ Er_{max} = Er_1 \\ \frac{RV_n - RV_{set}}{(n-1)\cdot d}, \ if \ Er_{max} = Er_n \ and \ n > 1 \end{cases} \tag{9}$$

Now to calculate $\frac{\partial RV_{max}}{\partial RA_{max}}$, we have to dive into the work of the one-step predictive ANN with the algorithm of backpropagation of the effect of the signal. Since this algorithm is involved in the regulator strategy, we are considering that it is worth exploring it in detail.

3.2. Backpropagation of the Effect of the Signal(BES)

Having a trained neural network, we can use it to determine the direction and value in which the output parameter will change when the input is changed. The algorithm of Backpropagation of the Effect of the Signal (hereinafter referred to as BES) resembles a backpropagation training algorithm but works without changing the network settings. Instead, it allows to calculate $\frac{\partial out}{\partial in}$ for the specific status of the inputs and outputs of this network. And that is just what we have left to do to complete Equation 8. Refer to Fig. 1 for an explanation of the BES algorithm, but somewhat simplify the network structure. Let's leave one input (no matter which one) and one output. Hidden neurons are located in two layers, two in each. This network configuration makes it possible to trace possible signal propagation options and can be easily scaled to any number of layers with any number of neurons in each.

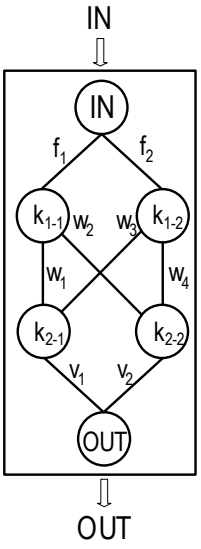

Fig. 5. The scheme of simplified one-step predictive ANN in BES algorithm

First of all, it should be understood that this scheme provides for three types of connections between elements. Namely, input-hidden neuron, hidden neuron-hidden neuron and hidden neuron-output neuron. Therefore, a description of these three types of communication enables a general description of the scheme. For

DOI 10.2478/ama-2020-0017

simplicity, each type of communication has its own letter in the scheme: f for input-neuron, w for neuron-neuron, v for neuron-output. Although they correspond to the weights for Equation 1. The BES algorithm starts with a forward pass through the network, in order to establish all the internal states and outputs of its nodes. After that, the back passage begins, where the effect of network input on the output must be calculated. Using a chain rule, find the effect on the output signal of a neuron from the last hidden layer, for example, k_{2-1}:

$$\frac{\partial out}{\partial state_{k_{2-1}}} = \frac{\partial out}{\partial out_{k_{2-1}}} \cdot \frac{\partial out_{k_{2-1}}}{\partial state_{k_{2-1}}} = \mu_{k_{2-1}} \quad (10)$$

where: $\mu_{k_{2-1}}$– effect of the neuron k_{2-1} state on the output signal. As for neuron k_{2-2}, its effect is calculated in the same way. Remembering Equations (1) and (2), we unpack the derivatives.

$$\mu_{k_{2-1}} = out_{k_{2-1}}\left(1 - out_{k_{2-1}}\right) \cdot v_1 \quad (11)$$

As we go further, we come across a neuron-neuron type connection, so we write down an equation describing the effect of neuron k_{1-2} on the output signal (with k_{1-1} everything is the same).

$$\frac{\partial out}{\partial state_{k_{1-2}}} = out_{k_{1-2}}\left(1 - out_{k_{1-2}}\right) \cdot \left(\mu_{k_{2-1}} \cdot w_3 + \mu_{k_{2-2}} \cdot w_4\right) \quad (12)$$

Finally, we pack the influence of the neurons of the first hidden layer and write the equation for the input signal.

$$\frac{\partial out}{\partial in} = \frac{\partial out}{\partial state_{k_{1-1}}} \cdot \frac{\partial state_{k_{1-1}}}{\partial in} + \frac{\partial out}{\partial state_{k_{1-2}}} \cdot \frac{\partial state_{k_{1-2}}}{\partial in} \quad (13)$$

Solving derivatives, we have:

$$\frac{\partial out}{\partial in} = \mu_{k_{1-1}} \cdot f_1 + \mu_{k_{1-2}} \cdot f_2 \quad (14)$$

After solving this problem in a simplified version, we can generalize its solution to allow search of $\frac{\partial out}{\partial in}$ in a network with an arbitrary number of hidden layers and with an arbitrary number of neurons in each (but in all hidden layers identical).

For the input of the neural network (see Fig. 1), the effect equation will be:

$$\frac{\partial out}{\partial in} = \sum_j \left(\mu_{k_{1-j}} \cdot f_j\right) \quad (15)$$

where: $\mu_{k_{1-j}}$ – effect on output of the neuron of the first hidden layer which makes neuron in range from 1 to j, f_j – weight of the connection between k_{1-j} and input.

For hidden layers (except the last):

$$\mu_{k_{(i-1)-j}} = out_{k_{(i-1)-j}}\left(1 - out_{k_{(i-1)-j}}\right) \cdot \sum_j \left(\mu_{k_{i-j}} \cdot v_{i-j}\right) \quad (16)$$

where: $k_{(i-1)-j}$ and k_{i-j} – neurons from adjacent layers (i - the number of hidden layers) with j neurons in each, v_{i-j} – weight of connection between them.

For the last hidden layer:

$$\mu_{k_{i-j}} = out_{k_{i-j}}\left(1 - out_{k_{i-j}}\right) \cdot w_j \quad (17)$$

where: k_{i-j} – neuron from the last hidden layer, w_j – weight of connection between k_{i-j} and output.

So, using the BES algorithm, we are able to complete the calculation of the Equation 8 and, accordingly, Equation 7. Multiple repetition of MAPERS gradually reduces the largest values of the regulation error throughout the planning horizon. As a result, we get an array of optimized predicted regulator actions, including RA_0 (see Fig. 4), which is the action that will be sent for execution by the next control cycle.

In the next section, we will consider the implementation of the previously described multipurpose neuronal network-based regulator on the example of a thermostat.

4. TEMPERATURE CONTROLLER BASED ON SELF-TUNING PREDICTIVE ANN

To demonstrate the operation of the specified multipurpose controller, run it within the task of regulating the temperature of a particular object. Thus, in Fig. 2, we replace the regulated value (RA) with the temperature of the regulated object (T), and at the site of the regulator action (RA), there will be a heater power (P) that is able to heat the control object. Our control object will be virtual, so let's specify the equation according to which it functions.

$$T = T_a + \frac{P}{m} \cdot \left(1 - e^{-b \cdot t}\right) + \left(T_0 - T_a\right) \cdot e^{-b \cdot t} \quad (18)$$

where: T_a – ambient temperature, m and b – parameters that make object temperature inertia (taking into account mass and volume), t – time, T_0 – initial temperature. With each change of heater power (P), time (t) is reset to zero and T_0 equals the current temperature value (T). In this way, the attenuation processes are restarted. Set the temperature range in which the controller should operate from 0 to 50°C. In this range, it shall provide a regulation accuracy of up to 0.5°C. Since this paper describes the processes in the simulated environment, to simplify the demonstration of the principles, the error of temperature measurement is neglected.

One hidden layer with 30 neurons is sufficient to achieve the specified accuracy of controller.

4.1. Training of one-step predictive artificial neural network

As a training set, we will use the reaction of the control object on full power of the heater with consistent cooling.

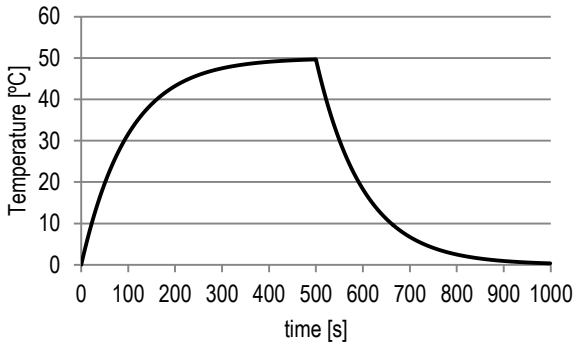

Fig. 6. Training set of temperature for ANN

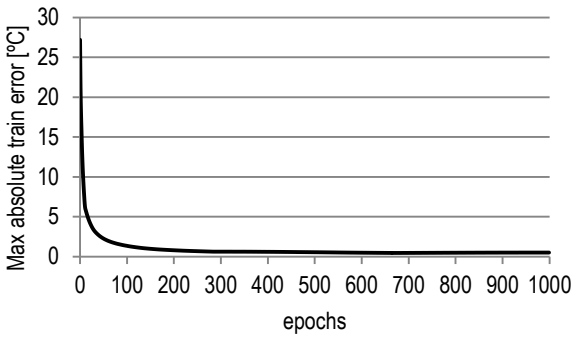

Fig. 7. Training process of ANN

Fig. 6 shows that for half of the training set points (500), the object is heated at P = 100%, after which the heater switches off (P = 0) and the object cools the other half of the set to ambient temperature ($T_a = 0°C$). Regarding the shape of the curve, it can be changed, the slope can be reduced and split into a larger number of time intervals with different heater power. It is not important for training but may be important for maintaining the control object.

As can be seen from Fig. 7, the ANN training process is non-linear. The first 100 epochs reduce the maximum absolute prediction error for the training set to 2 degrees. And after 1000 epochs, the level of the specified error decreases under 0.5 degrees. This means that the network is able to operate with the desired accuracy within the specified range. After training, the regulator is ready to go. Now let's compare it with a PID controller.

4.2. The result of the operation of the self-tuned temperature controller

To evaluate the performance of the previously described controller (with MAPERS), consider it together with a regular PID controller. Fig. 8 shows the response of the system to the change in the desired temperature of the control object (T_{set}).

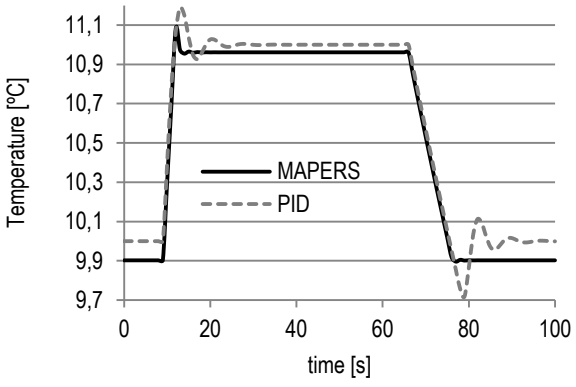

Fig. 8. Comparison of adaptive controller operation with classic PID

At the beginning of measurement, T_{set} is equal to 10 degrees, after which it switches to 11 degrees and after the transition processes returns to 10 degrees. So, we can see the response of regulators to the single step up and down. From what we have seen, we can conclude that the self-tuning controller is capable of operating in a given range with a given accuracy ($0.5°C$), and even significantly outperforming it (the maximum absolute error for steady state during measurements was $0.097°C$). It is worth recalling that in modelling, we ignore measurement errors and focus on the behaviour of regulators under given conditions. Also, the self-tuning controller performed better than the manual tuned PID controller in dynamics, avoiding oscillations and large overshoot. At the same time, the PID controller was more accurate in steady state. It is possible to increase the accuracy of the self-tuning controller by increasing the number of hidden layer neurons and increasing the learning time. But the main advantage of an adaptive controller, as opposed to the manual tuned PID, is automatic tuning. With having a training set that characterizes the behaviour of the control object, we can build a controller with arbitrary complexity and precision without human intervention. The given example of a temperature regulator is the most primitive and clear, therefore, it is necessary to emphasize the possibility for the described adaptive regulator to operate much more complex, multidimensional processes (such as autopilot, industrial control systems, automotive onboard systems, etc.) with the involvement of many input and output signals. The use of a classic PID controller in such processes may not be appropriate and sometimes possible.

5. CONCLUSIONS

The research work aimed to consider a control system that may be an alternative to classic PID controllers in tasks that require automation of the controller setup process. A multipurpose neuronal network-based controller, discussed in this paper, may be that kind of system. We first looked at its structure and the main modules that make it up with a detailed description of the operation of each of these modules and the system as a whole. After a general analysis, we proceeded to solve a specific problem, which was the synthesis of an adaptive thermostat. The synthesized adaptive thermostat showed the set accuracy and stability of control at the simulation level. The success of this task, showed the value of the original model, its strengths, weaknesses and possible ways to improve it. At the same time, it is necessary to continue studying the capabilities of the described regulator on the examples of real control processes. It is worth to study the possible strategies (beyond MAPERS) for optimizing the control action of the regulator.

REFERENCES

1. **Ayomoh M. K. O., Ajala M. T.** (2012), Neural Network Modeling of a Tuned PID Controller, *European Journal of Scientific Research,* 71, 283–297.
2. **Burennikov Y., Kozlov L., Pyliavets V., Piontkevich O.** (2017), Mechatronic Hydraulic Drive with Regulator, Based on Artificial Neuron Network, *IOP Conference Series: Materials Science and Engineering,* 209(1):012071.
3. **Du X., Wang J., Jegatheesan V., Shi G.** (2018), Dissolved Oxygen Control in Activated Sludge Process Using a Neural Network-Based Adaptive PID Algorithm, *Applied Sciences,* 8(2):261, DOI: 10.3390/app8020261.
4. **Elsisi M.** (2019), Design of neural network predictive controller based on imperialist competitive algorithm for automatic voltage regulator, *Neural Computing and Applications,* 31, 5017–5027.

DOI 10.2478/ama-2020-0017

5. **Han G., Fu W., Wang W., Wu Z.** (2017), The Lateral Tracking Control for the Intelligent Veicle Based on Adaptive PID Neural Network, *Sensors,* 17(6):1244, DOI: 10.3390/s17061244.
6. **Heaton J.** (2008), *Introduction to Neural Networks with Java*, Heaton Research Inc., St. Louis.
7. **Hernández-Alvarado R., García-Valdovinos L.G., Salgado-Jiménez T., Gómez-Espinosa A., Fonseca-Navarro F.** (2016), Neural Network-Based Self-Tuning PID Control for Underwater Vehicles, *Sensors,* 16(9), 1429, https://doi.org/10.3390/s16091429.
8. https://stackoverflow.com/questions/10732027/fast-sigmoid-algorithm (08.02.2018)
9. **Liu B., Hussami N., Shrikumar A., Shimko T., Bhate S., Longwell S., Montgomery S., Kundaje A.** (2019), A multi-modal neural network for learning cis and trans regulation of stress response in yeast, arXiv:1908.09426.
10. **Ma H., Lang S., Wellßow W.** (2018) Fallback Solution for a Low-Voltage Regulator Control using Artificial Neural Networks, *CIRED 2018 Ljubljana WS*, http://dx.doi.org/10.34890/413.
11. **MacLean D.** (2019), A convolutional neural network for predicting transcriptional regulators of genes in Arabidopsis transcriptome data reveals classification based on positive regulatory interactions, bioRxiv 618926.
12. **Pirabakaran K., Becerra V.M.** (2002), PID autotuning using neural networks and model reference adaptive control, IFAC Proceedings, 35, 451–456.
13. **Wica M., Witkowsk M., Szumiec A., Ziebura T.** (2019), Weather forecasting system with the use of neural network and backpropagation algorithm, *Proceedings of the International Conference on Data Engineering and Communication Technology,* 2468, 37–41, DOI: 10.1007/978-981-10-1675-2_62.
14. **Zaman M.H.M., Marzuki M.M., Hannan M.A., Hussain A.** (2018), Neural Network Based Prediction of Stable Equivalent Series Resistance in Voltage Regulator Characterization, *Bulletin of Electrical Engineering and Informatics,* 7, 134–142, DOI: 10.11591/eei.v7i1.857.
15. **Zhang Z., Ma C., Zhu R.** (2016), Self-Tuning Fully-Connected PID Neural Network System for Distributed Temperature Sensing and Control of Instrument with Multi-Modules, *Sensors,* 16(10):1709, DOI: 10.3390/s16101709.
16. **Zhao D., Yang T., Ou H., Zhou H.** (2018), Autopilot Design for Unmanned Surface Vehicle based on CNN and ACO, *International Journal of Computers Communications & Control,* 13(3), 429–439, DOI: 10.15837/ijccc.2018.3.3236.