On Practical Constraints of Approximation Using Neural Networks on Current Digital Computers

Michal Puheim[1], Ladislav Nyulászi[2], Ladislav Madarász[3], Vladimír Gašpar[4]
Department of Cybernetics and Artificial Intelligence
Technical University of Kosice
Letná 9, 042 00 Košice, Slovak Republic
[1]michal.puheim@tuke.sk, [2]ladislav.nyulaszi@tuke.sk, [3]ladislav.madarasz@tuke.sk, [4]vladimir.gaspar@tuke.sk

Abstract—**Goal of this paper is to highlight the most common problems and constraints which accompany the implementation of artificial neural networks on current digital computers. We focus on feed-forward multilayer neural networks, i.e. multilayer perceptrons, in role of function approximators. Multiple constraints of approximation by neural networks are discussed within the paper, taking into account research from the previous two decades. We address the issues of structural construction of feed-forward neural networks, learning and data pretreatment. Conclusions stated by universal approximation theorem cannot be blindly applied to implementations on real hardware without considering the limitations such as finite accuracy of floating point operations and data type overflow issues. This fact is emphasized in the paper.**

Keywords—artificial neural network, multilayer perceptron, approximation constraints, universal approximation theorem, digital computers.

I. Introduction

Artificial neural networks (ANN) are software programs, which in highly abstract way imitate the function of certain parts of biological nervous system. During the last two decades they became the centerpiece of artificial intelligence methods due to their capability to approximate unknown nonlinear functions or relations between elements of various complex systems. However, there always is a possibility that ANNs will fail to accomplish its stated objectives. It is troublesome that some scientists and technical engineers try to overcome these situations using sudden ad hoc solutions instead of systematic approach. Apart from that, there is evident lack of understanding of fundamental background of ANN theory, especially in case of younger researchers, who often use “all in one” or “ready to deploy” frameworks (e.g. FANN [1]) for their applications [2]. The “universal approximation” proofs are commonly used to justify the idea that neural networks can “do anything” [3]. However, these proofs consider calculations with unlimited accuracy (!), which is not achievable on current computers. Furthermore, there is often no interest in pretreatment of training data or proper normalization, let alone implications resulting from implementation on real hardware.

The rest of the paper is organized as follows: In the second section, we briefly explain the fundamentals of neural networks and the universal approximation theorem. In the third section we focus on problems of design and learning of neural networks from theoretical standpoint. In the fourth section we address the issues resulting from implementation on digital computers. In the conclusion we summarize the consequences of problems from previous sections and try to outline the instructions for proper design and implementation of ANN.

II. Neural Network as Universal Approximator

An ANN is a well established alternative to conventional deterministic computational methods. According to the universal approximation theorem, which is discussed later, ANN can approximate arbitrary continuous smooth bounded function, using only samples of input and output data to learn to perform the approximation. It is also able to solve variety of other similar tasks such as signal transformation, memory simulation, association problems, classification, etc. [4][5]. In this section we provide the basic foundation for ANNs along with approximation theorems, which support the use of ANNs as function approximators. We also provide references to proofs for these theorems.

A. Introduction to Sigmoidal Feed-Forward Neural Networks

ANN is basically a parallel processor, which is able to store knowledge in connections between its core elements and use it later for various computations. Basic structural element of ANN is the artificial neuron (see Fig. 1).

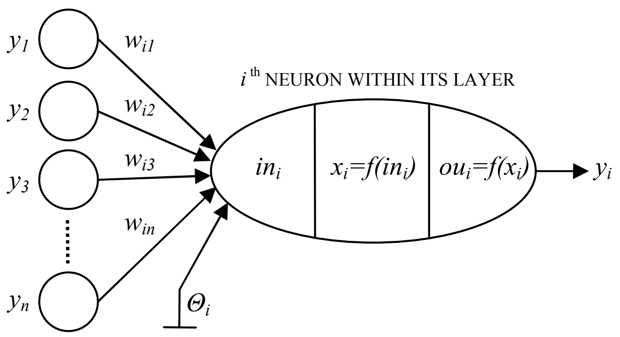

Fig. 1. The artificial neuron.

The artificial neuron y_i is composed of the following components [5]:

- *synaptic weights* (w_{ij}) and *bias weight* (Θ_i), which are basically carriers of knowledge stored within the ANN,

- *input function* (in_i) of the neuron y_i, which is calculated as sum of product of outputs ou_j from previous neurons y_j and corresponding weights w_{ij}:

$$in_i = \sum_{j=1}^{N} w_{ij} ou_j + \Theta_i , \quad (1)$$

- *activation function* (x_i) of the neuron y_i, which squashes the result of the input function to interval (0,1); most used activation function is sigmoid:

$$x_i = f(in_i) = \frac{1}{1+e^{-\alpha\, in_i}} , \quad (2)$$

but any other bounded function can be used, e.g. threshold function:

$$x_i = f(in_i) = \begin{cases} 1 & in_i \geq 0 \\ 0 & in_i < 0 \end{cases} , \quad (3)$$

- *output function* (ou_i) of the neuron y_i, which is often the identity function:

$$ou_i = f(x_i) = x_i . \quad (4)$$

With regard to its basic components, we can also define artificial neuron y_i mathematically as:

$$y_i = \varphi(\sum_{j=1}^{N} w_{ij} y_j + \Theta_i) , \quad (5)$$

where y_j are signals from preceding neurons and φ is the activation function used, e.g. (2) or (3). Such neuron (with threshold activation function) is also referred to as the *McCulloch-Pitts neuron* [6][7].

In order to form the ANN, the artificial neurons are mutually interconnected into more complex structures with certain defined topology. Generally, the ANN can have an arbitrary structure, which can be described as an oriented graph, but properties of such network are difficult to analyze. Therefore, the majority of used networks have regular topologies, often layered. The most common is the multilayered structure with one input layer, one output layer and one or more hidden layers in between, where every neuron within the layer is connected with all neurons from the previous layer and the following layer. This layout is often referred to as *feed-forward artificial neural network* or *multilayer perceptron* (see Fig. 2).

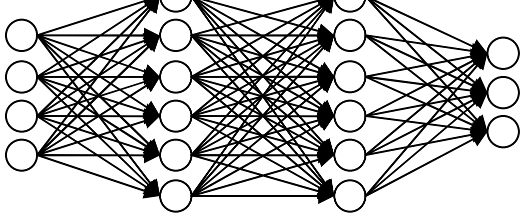

Fig. 2. Multilayer perceptron with two hidden layers.

B. Universal Approximation Theorem

Applicability of ANN comes from the fact that the ANN is a universal approximator of an arbitrary bounded function. The *universal approximation theorem* states [6] that:

"An arbitrary continuous bounded function can be uniformly approximated by a multilayer feed-forward neural network with one hidden layer with finite number of neurons with arbitrary bounded activation functions".

The theorem can be also formulated mathematically:

Let φ be a monotonically-increasing bounded continuous function. Let I_m be the m-dimensional unit hypercube $[0, 1]^m$. The space of continuous functions on I_m is denoted as $C(I_m)$. Then, for any given function $f \in C(I_m)$ and $\epsilon > 0$, there exist such an integer N and real constants α_i, $b_i \in \mathbf{R}$, $w_i \in \mathbf{R}^m$, where $i = 1, ..., N$, that we may define:

$$F(x) = \sum_{i=1}^{N} \alpha_i \varphi(w_i x + b_i) , \quad (6)$$

as an approximation of the function f independent of φ; that is:

$$\left| F(x) - f(x) \right| < \varepsilon , \quad (7)$$

for all $x \in I_m$. In other words, functions of the form $F(x)$ are dense in $C(I_m)$.

The universal approximation theorem is based on a classical theorem on function approximation, the Kolmogorov's theorem [8]. It provides the theoretical support for the existence of ANNs that implement arbitrary functions with more than one independent variable and motivates the usage of multilayer feed-forward ANN as function approximators. The theorem was rigorously mathematically proven independently by Cybenko [9], Hornik [10] and Funahashi [11]. Hornik has also shown, that the type of the activation function is not relevant and that it is the multi-layer feed-forward architecture, which gives neural networks the potential of being universal approximators.

III. Theoretical Constraints of Approximation by Neural Networks

The universal approximation theorem states that ANNs are universal approximators, but it does not give any advice how to find the parameters for the ANN which would solve the particular approximation problem. The parameters of ANN are number of layers, number of neurons and values of synaptic weights in between. These parameters are usually determined in the learning process using the training data, which correspond to the function which is the target of approximation. In the next paragraphs, we will point out typical problems which accompany the design and learning of ANNs.

A. Black Box Objection

Common argument against using an ANN is that the knowledge stored in synaptic weights is hard to extract and understand by human experts. ANN is often compared to black box, because it is difficult to visually represent the stored knowledge. Another shortcoming is that after the ANN has finished its learning on the initial set of training data, *it is difficult to learn to cover new examples* of training data, i.e. to approximate previously unknown parts of the target function. If we try to do incremental learning, we may end up *losing the precision* on already learned examples and consequently loose the average precision for all examples.

Therefore, a typical application scenario of an ANN in solving the approximation task is the one, in which the approximated function or corresponding input and output data is known beforehand. In this case, the ANN is used as a black box, because we only apply some of the existing learning algorithms to automatically find appropriate parameters (values on the synaptic weights), but we cannot deduce anything from these parameters alone. We are rarely interested in doing so. The problem arises when the target function changes because we *cannot do the incremental learning* due to loss in general precision of the ANN. The only option is to start the learning process from the beginning using all the available data. However, this may be very time consuming and due to the stochastic nature of ANN, the learning may not lead to any acceptable results at all. This issue will be further discussed in the next section.

B. Stochastic Nature of Learning

The universal approximation theorem only states that when given appropriate parameters, simple neural networks can represent a wide variety of functions, but it does not grant the algorithmic possibility to find desired parameters. The only option to find these parameters is to try to use one of available learning algorithms. Most algorithms focus only on the problem of finding the suitable weights and do not take the structural parameters of ANN into account, such as the number of hidden layers and number of neurons within these layers. In this section we only discuss the weight setting algorithms. The neuron/layer count problem is covered in the next section.

When we aim to train the ANN to approximate an unknown function, we usually have to use samples of input data X and output data Y, which determine the target function $f : X \to Y$. Type of learning, which utilizes such data is referred to as *supervised learning* or *data-driven learning*. There are three basic types of supervised learning [5]:

- *error correction learning*, which proposes the change of weights as function of output error of a given neuron,
- *stochastic learning*, which changes weights quasi-randomly and accepts the change only if it improves the approximation precision,
- *reinforcement learning*, which is similar to error correction learning and the difference rests on global mechanism to change weights according to overall state of whole network, not only one neuron.

The main problem of all of these methods is that they are *not guaranteed to converge* to optimal solution or they may not find any suitable solution at all. Common complication is convergence to suboptimal solution (or local minima). Even though the error correction learning methods [13] are usually deterministic in the way that they search for the solution, they depend upon random weight initialization at the beginning of ANN training. The problem of finding optimal weights also grows with the size of the network. In case of multiple hidden layers (so called deep learning [14]), we encounter the saturation problem on sigmoid neurons, which slows learning considerably. Another problem is the gradient explosion/ vanishing of the error, propagated through more than one layer. This happens because back-propagated error is not being squashed by activation function unlike feed-forward signal [15].

A possible solution for the convergence problem of ANN is to restart the learning with different initial weights, although it might not be helpful. Another option is to change the topology of the network by adding or removing neurons or changing the number of layers.

C. Neuron/Layer Count Problem

The universal approximation theorem assumes that ANN with one hidden layer is capable of approximating arbitrary unknown continuous function. Problem is that it does not tell us anything about the number of neurons on this layer. Furthermore it does not make clear if it is not better to use multiple layers. Therefore, the question is how to choose optimal topology of ANN, i.e. to determine the ideal neuron/layer count in order to achieve the best possible approximation and cut down learning times.

The first step when designing the architecture of an ANN is to define the number of hidden layers. According to the universal approximation theorem, one hidden layer should be enough to approximate any unknown function. But for solving some particular problems (e.g. the XOR problem) it may be helpful to use more. While one hidden layer can be enough, it may require unfeasibly large number of neurons, whereas comparable multi-layered networks could be of manageable size [3]. Indeed, the results from [11] show that it is often better to use two hidden layers, where the first hidden layer processes local features of input sample and the second hidden layer processes global features [5].

If we simplify the architecture of the ANN to two hidden layers, then the only remaining problem is to determine the number of neurons on each of these layers. In general, there are three possible approaches:

- *trial and error method* – most common method, where the number of neurons is initially set ad hoc and only if ANN cannot converge to a viable solution it is randomly increased or decreased,
- *construction method* – the neurons are initially set to the lowest number possible and gradually added, until the ANN reaches satisfactory approximation,
- *pruning method* – the ANN is initially trained using large number of neurons, which are later discarded depending on their contribution to the total error of the network [12].

The power of ANN depends on the ability to generalize new data after the initial training, i.e. the ability to interpolate and extrapolate data not seen in the past [12]. The generalization ability largely depends on the number of hidden layers and neurons. The ANN with *large number of hidden neurons* is able to learn faster and has better chance of avoiding the local minima but its *ability to generalize is usually weak* as over-fitting may occur. On the contrary, the ANN with *lower number of hidden units* is more difficult to train, but has *better generalization capability* [12]. Smaller networks are also more efficient performance-wise. Therefore we should always use as simple architecture as possible.

D. Input/Output Normalization Issues

One of the requirements for the approximated function is that it is *bounded,* which means that its input parameters and output values are set within an enclosed interval. Likewise, the corresponding ANN needs to have its inputs and outputs bounded, usually to the interval [0, 1]. It is evident that in order to process the data by the ANN, we need to transform the values of the data to the interval [0, 1] in the first place. In other words, the *data need to be normalized* [16].

If the domain of some I/O value x is bounded within the interval $[a, b]$, it is possible to normalize it to value x_n from the interval [0, 1] using the formula:

$$x_n = \frac{x-a}{b-a}\,. \tag{8}$$

The task of normalization may look trivial, especially if domains of input and output data are clearly given. However it can become more challenging in some cases [16], especially if:

- *parameter domain is unbounded* or half-bounded (e.g. distance, which can obtain values from 0 to infinity),
- *parameter domain is bounded*, but we *do not have data* for a large part of the domain, or we do not need to use the whole domain,
- *parameter domain is unknown*, i.e. we do not possess the information about boundaries a and b.

In case that domain is unbounded, it is necessary to perform a transformation, which will delimit it within defined interval. Therefore, we need to carry out suitable transformation:

$$\mathrm{T}:[-\infty, \infty]\rightarrow[0, 1]\,, \tag{9}$$

For the purpose of transformation, it is possible to use:

- *logistic*, *sigmoid* or *inverse tangent* functions for *unbounded domains*,
- *logarithmic* functions for *half-bounded domains* (e.g. for distance).

In other cases we propose three basic approaches to I/O data normalization [16]:

- *expert oriented normalization*, where the boundaries a and b are determined by an expert or from some well known source,
- *automatic normalization*, where the boundaries a and b are determined by minimum and maximum values from training dataset,
- *semi-automated normalization*, where the boundaries a and b are initially set as minimum and maximum values from training data, but later can be modified by an expert or some optimization algorithm with goal of normalizing the data approximately to the same order of magnitude.

In general, we can say that data normalization within certain criteria, prior to the training, can help us obtain *better approximation* results, *reduce required size of the network* and also *speed up the training* process. In the best scenario, all normalized data should be approximately in the same order of magnitude, preferably in the order of one [16].

IV. Implementation-Related Constraints of Approximation by Neural Network

Implementation of ANN on physical hardware is restricted by several constraints. First of all, the computational and memory capacity of computers is limited, which means, that only finite number of neurons can be simulated in one time. Secondly, both analog and digital computers use numbers with finite magnitude. This means that weights cannot grow to infinity. And finally, the results of the approximation theory cannot be used blindly without consideration of precision limits of floating point operations, as these limitations constrain the approximation capability of neural networks [3].

A. Demand on Computing Resources

Calculation of ANN output may be quite demanding on computing resources especially in case of large networks and real-time applications such as real-time control, object recognition etc. In case that an implemented ANN consumes too many resources, optimizations are necessary. Obvious option is to use an ANN with lower number of layers/neurons, but this may come at a price of worse approximation capability of the ANN or unfeasible learning times. Other option is to gain performance increase by optimizing arithmetic calculations. If we consider the mathematical formula of a neuron (5), it is obvious that the most demanding computation is the calculation of a nonlinear activation function φ (which is mostly sigmoid). In order to gain performance increase it is possible to compute the function φ using a lookup table (LUT) [17]. Another option is to deliberately limit the range or precision used to store weight values. This possibility is covered in the following sections.

Suggestions stated above also apply to the ANN learning. However, during the learning process of large networks, it is also possible to speed up learning by using localized approximations [18], where we train n smaller networks separately, each for particular subinterval of the approximated function. This can be also done using parallelization. After the sub-networks are trained, they are merged into the final network. An advantage of this method is that if the approximated function changes, it is not necessary to retrain the whole network, only the affected parts.

B. Limited Weight Range and Overflow Issues

The ability of ANN to approximate an arbitrary continuous function (as declared by universal approximation theorem in section II.B of this paper) is based on the assumption that arbitrary large weights and a sufficient number of hidden units are available. However, in the real-world hardware implementations, both the numbers of hidden neurons and range of weights are limited [3]. Therefore some ANN structures, which should be capable of approximating certain functions by theory, would not be able to achieve this goal in real conditions, because values of some weights may need to come close to infinity (especially on bias weights). Another unwelcome implication of this problem lies in the fact, that if not treated, the change of weights during learning may potentially cause data-type overflow, which could lead to very strange and unpredictable behavior of the training algorithm. In the worst case, the whole program may crash unexpectedly. Ranges of basic floating point data-types in C programming language are shown in TABLE I.

TABLE I. FLOATING POINT TYPES IN C PROGRAMMING LANGUAGE

Type	Size	Value Range	Precision
float	4 bytes	1.2E-38 to 3.4E+38	6 decimal places
double	8 bytes	2.3E-308 to 1.7E+308	15 decimal places
long double	10 bytes	3.4E-4932 to 1.1E+4932	19 decimal places

Only way to overcome the overflow issue is to perform check at every weight update and add more neurons to the network every time the overflow occurs. Additional neurons should ensure that weights would not need to grow near infinity.

C. Limited Weight Precision

Similarly to weight range constraints, there is restriction on weight precision. This limitation further reduces approximation capabilities of ANNs implemented on digital computers. Consequences of reduction to weight precision (and range) are depicted on Fig. 3. Every line in the figure represents the input function in_i of the neuron y_i with one input ou_j from previous neuron y_j. The input function is therefore calculated as the sum of bias weight Θ_i and the product of ou_j and weight w_{ij} :

$$in_i = w_{ij} ou_j + \Theta_i . \qquad (10)$$

From the figure, we can see that the mesh of possible input functions has large interstices and covers only small portion of space, when the weight precision and range is restricted. In other words, when the weights precision and range is lower, the learning capability of the ANN is relatively weaker [19].

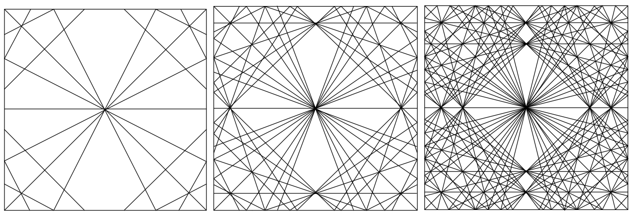

Fig. 3. Mesh of possible input functions that can be implemented with weights precision equal to 1 in the ranges [-2,2], [-3,3], and [-4,4], drawn in the square [-0.4,0.4]. Image is from [19].

Nevertheless, in some cases, we may deliberately aim for an implementation using the limited precision weights. For example implementations on embedded devices with ARM CPUs may benefit from such solutions, for the reason that it requires less memory for storing the weights and less expensive floating point units in order to perform the computations involved [19]. However, in this case it is necessary to consider the lower ANN approximation and learning capability. It is difficult for conventional training algorithms to guarantee the convergence when the weights precision is limited. In order to avoid this problem, it is possible to use higher precision during learning and lower during a runtime.

V. CONCLUSIONS

There is a raising trend of using the ANN as a black box and not taking care about its internal processes. Typical procedure is to connect the inputs, connect the outputs, set the network to largest size possible and use some predefined learning algorithm. Even though it is simple, it is a very bad practice.

During the implementation of ANN with help of third-party frameworks, it is necessary to realize that the quality of the solution is only as good as depth of understanding the problem. Optimal solution for any specified problem can only be achieved by carefully setting the adjustable parameters, taking the “no free lunch theorem” [20] into account.

Finally, to help with the design of ANN which would satisfy approximation requirements, we would recommend complying with the following points:

- understand the approximation problem, identify required inputs and outputs for the ANN,
- gather sufficient amount of training data and properly normalize this data – this is a very important step, which shortens the learning time, improves the performance of resulting approximation and reduces required size of the ANN,
- use at least 2 hidden layers of size adequate to number of inputs and outputs and expected complexity of relations between inputs and outputs,
- use construction or pruning algorithms to optimize size of the network – this helps if there are any problems with learning and also improves the resulting performance of the ANN; do not use as large network as possible, as this will lead to poor generalization and over-fitting,
- use random weight initializations, restart learning when ANN is stuck in local minima, possibly add new hidden neurons,
- use check for data type overflow if the programming language does not do it automatically,
- if there are any performance issues, try to optimize function calculations using LUT or reduce weight precision; remember, that it will affect learning and

approximation accuracy; it is also possible to use different precision for learning and runtime of ANN.

ACKNOWLEDGEMENT

The work presented in this paper was supported by VEGA, Grant Agency of Ministry of Education and Academy Science of Slovak Republic under Grant No. 1/0298/12 – "Digital control of complex systems with two degrees of freedom." The work presented in this paper was also supported by KEGA under Grant No. 018TUKE-4/2012 – "Progressive methods of education in the area of control and modeling of complex systems object oriented on aircraft turbo-compressor engines."

REFERENCES

[1] S. Nissen: "Fast Artificial Neural Network Library," [Online]. 2013. Available: http://leenissen.dk/fann/wp/news/ (Cited: 3. June 2014)

[2] S. Nissen: "Neural Networks Made Simple," [Online]. 2013. Available: http://fann.sourceforge.net/fann_en.pdf (Cited: 3. June 2014)

[3] M. Kárný, K. Warwick, V. Kůrková: "The Psychological Limits of Neural Computation," In Dealing with Complexity, Perspectives in Neural Computing, Springer London, pp. 252-263. 1998. ISBN 978-3-540-76160-0

[4] S. Haykin: "Neural Networks: A Comprehensive Foundation", Volume 2, Prentice Hall. 1998. ISBN 0-13-273350-1.

[5] P. Sinčák, G. Andrejková: "Neurónové siete – Inžiniersky prístup," Dopredné neurónové siete, č. 1. Košice: Elfapress, 1996, ISBN 80-88786-38-X

[6] B. Cs. Csáji: "Approximation with Artificial Neural Networks," Master Thesis, Faculty of Sciences, Eötvös Loránd University, Hungary. 2001

[7] W. S. McCulloch, W. H. Pitts: "A Logical Calculus of the Ideas Immanent in Nervous Activity," Bulletin of Mathematical Biophysics, Vol. 5, pp. 115-133. 1943.

[8] A. N. Kolmogorov: "On the Representation of Continuous Functions of Several Variables by Superposition of Continuous Functions of one Variable and Addition," Doklady Akademii. Nauk USSR, 114, pp. 679-681. 1957.

[9] G. Cybenko: "Approximation by Superpositions of a Sigmoidal Function," Math. Control Signals Systems, 2, pp. 303-314. 1989.

[10] K. Hornik, M. Stinchcombe, H. White:. "Multilayer Feedforward Networks are Universal Approximators," Neural Networks, 2(5), pp. 359-366. 1989.

[11] K. Funahashi: "On the Approximate Realization of Continuous Mapping by Neural Networks," Neural Networks, Vol.2, pp. 183-192, 1989.

[12] M.G. Augasta and T. Kathirvalavakumar: "Pruning algorithms of neural networks — a comparative study", Central European Journal of Computer Science,Vol. 3, pp. 105-115. 2013. ISSN 1896-1533.

[13] D. E. Rumelhart, G. E. Hinton, R. J. Williams: "Learning representations by back-propagating errors," Nature, Vol. 323, pp. 533–536. 1986.

[14] X. Glorot, Y. Bengio: "Understanding the difficulty of training deep feedforward neural networks," Proc. of the 13th International Conference on Artificial Intelligence and Statistics, Chia Laguna Resort, Sardinia, Italy. 2010.

[15] M. Gregor, P. Groumpos: "Tuning the position of a fuzzy cognitive map attractor using back propagation through time," 7th International Conference on Integrated Modeling and Analysis in Applied Control and Automation. Athens, Greece, 25-27 September 2013.

[16] M. Puheim, L. Madarász: "Normalization of Inputs and Outputs of Neural Network Based Robotic Arm Controller in Role of Inverse Kinematic Model." IEEE 12th International Symposium on Applied Machine Intelligence and Informatics. Herľany, Slovak Republic. 2014.

[17] F. Piazza, A. Uncini, M. Zenobi: "Neural networks with digital LUT activation functions," Proc. of 1993 International Joint Conference on Neural Networks, vol.2, pp. 1401-1404. 25-29 Oct. 1993. DOI: 10.1109/IJCNN.1993.716806

[18] N. Hahm, B. Hong Il: "The Capability of Localized Neural Network Approximation," Honam Mathematical Journal, Vol. 35, Issue 4, pp.729-738. 2013. DOI: 10.5831/HMJ.2013.35.4.729

[19] Bao Jian, Chen Yu, Yu Jinshou: "Neural Networks with Limited Precision Weights and Its Application in Embedded Systems," International Workshop on Education Technology and Computer Science, pp. 86-91. 2010. DOI: 10.1109/ETCS.2010.448

[20] D.H. Wolpert, W.G. Macready: "No Free Lunch Theorems for Optimization", IEEE Transactions on Evolutionary Computation 1, pp. 67-82. 1997. ISSN : 1089-778X