Determining the Number of Neurons in Artificial Neural Networks for Approximation, Trained with Algorithms Using the Jacobi Matrix

Kostadin Yotov, Emil Hadzhikolev, Stanka Hadzhikoleva

Plovdiv University Paisii Hilendarski, Faculty of Mathematics and Informatics,
236 Bulgaria Bul., Plovdiv, Bulgaria

Abstract **– How can we determine the optimal number of neurons when constructing an artificial neural network? This is one of the most frequently asked questions when working with this type of artificial intelligence. Experience has brought the understanding that it takes an individual approach for each task to specify the number of neurons. Our method is based on the requirement of algorithms looking for a minimum of functions of type $S(z) = \sum_{i=1}^{m}[\varphi_i(z)]^2$ that satisfy the inequality $p \leq m$, where p is the dimensionality of the argument z, and m is the number of functions. Formulas for an upper limit of the required neurons are proposed for networks with one hidden layer and for networks with r hidden layers with an equal number of neurons.**

Keywords **– neural networks for approximation, number of neurons, neural network design.**

1. Introduction

One of the most common questions that arise when configuring artificial neural networks is related to the possibility of preliminary assessment of the topology, which would provide the desired accuracy or simply decrease the time spent in modeling.

DOI: 10.18421/TEM94-02
https://doi.org/10.18421/TEM94-02

Corresponding author: Stanka Hadzhikoleva,
Plovdiv University Paisii Hilendarski.
Email: stankah@uni-plovdiv.bg

Received: 09 September 2020.
Revised: 19 October 2020.
Accepted: 23 October 2020.
Published: 27 November 2020.

The people asking that question usually want to get a formula, a rule that would guide them to determining the number of neurons or the network layers based on certain quantities, such as the number of inputs or the number of training examples. For example:

$$N_h = f_1(N_i)\ ;\ N_h = f_2(N_t), \text{ or}$$

$$N_l = f_3(N_i)\ ;\ N_l = f_4(N_t),$$

where N_h is the number of neurons in the hidden layer; N_l – the number of hidden layers; N_i – the number of inputs; N_t – the number of training examples. A similar one-parameter approach is described in [1], [2], [3].

Other scientists offer functions of several variables. For example:

$$N_h = f_1(N_i, N_t).$$

Research in this area is described by Xu and Chen [4]. Shibata and Ikeda look for a connection between the number of neurons and the number of inputs in the dendritic system of the network and the number of output axons [5].

Here we must keep in mind that the questions do not make much sense when asked this way. For each set of training examples of the set

$$(1)\ \{x_{1j}, x_{2j}, \dots x_{N_i j}; y_{1j}, y_{2j}, \dots y_{N_0 j}\}_{j=1}^{N_t}$$

for $N_i, N_0, N_t \in N$ (N_0 is the number of output examples), in the end, an approximating neural network can be built with any number of neurons in the hidden layer.

The question is not whether it is possible to construct a neural network – this is quite feasible. The question is how effective these networks are. For the same set consisting of examples of the type (1), we can construct a neural network with 1, 2, 3, ... 100, 10 000 and generally with any number of neurons. 6hus, the question "what is the number of

neurons that make up the neural network" seems to naturally lose its meaning. A sensible question would be the following: ***How can we determine the optimal number of neurons ensuring high efficiency of the neural network?***

2. Preliminaries and Problem Description

In recent years, many scientists have worked on the task of determining the optimal number of neurons needed to construct a neural network. Table 1. synthesizes different formulas for the number of neurons in the hidden layer N_h. The following parameter notations are used: N_i – number of inputs (in some of the scientific literature interpreted as input neurons); N_t – number of network training examples; N_o – number of output neurons.

The presented formulas define the number of neurons in specific individual cases. For example, from the proposal of Li and team [1], $N_h = \frac{\sqrt{1+8\mathrm{N_i}}-1}{2}$, it follows that the number of neurons is determined only by the number of input variables. This in turn means that any function of n variables will be able to be successfully approximated by a neural network consisting of $N_h = \frac{\sqrt{1+8n}-1}{2}$ neurons in the hidden layer, regardless of the type of function, the specific selection of training examples, the training algorithm or the data transfer function. According to the authors, any function of three variables would be most successfully approximated by a network with two neurons in the layer. But with standard activation functions and training methods used by software libraries, it is clear that linear functions of the type $y = ax_1 + bx_2 + cx_3, a, b, c \in R$ can be easily approximated by even one neuron in the hidden layer, with a very low order of the error $MSE = 1E-20$ to $1\mathrm{E}-30$. On the other hand, for nonlinear functions such as $y = ax_1^m + bx_2^p + cx_3^q, a, b, c, m, p, q \in R$ or for example $y = Log\left|Tan(\frac{ax_1^m+bx_2^p+cx_3^q}{2})\right| + \sum_{i=1}^{3}\cos x_i$, the use of only two neurons is extremely insufficient in most cases, even with appropriately selected training examples, which, in fact, we often do not have in reality.

The situation is similar with other options for calculating the number of neurons, when using only the number of inputs or only the number of neural network outputs, as proposed in many scientific publications [2], [6], [3].

What is the solution then? At first glance, it seems that the function representing the number of neurons should cover all significant variables:

(2) $N_h = F(N_l, N_i, N_o, N_t, T_l, G, E, W, B, V, I)$,where

- N_h is the number of neurons in the network;
- N_l – number of hidden layers;
- N_i – number of inputs / dimension of the input vector;
- N_o – number of output neurons;
- N_t – number of training examples;
- T_l – type of training;
- G – type of activation function;
- E – selected maximum number of epochs in training;
- W and B – the corresponding matrices of initially initialized weights and thresholds;
- V – degree of scatter of the training examples – type of the correlation field;
- I – Interval of the input data.

Given the theorem proven by Cybenko [7], we can fix:

$$N_l = 1,$$

and set the condition for sigmoid character of the continuous activation function:

$$G(x) = \begin{cases} 1, x \to +\infty \\ 0, x \to -\infty \end{cases}$$

Thus, as far as approximating neural networks are concerned, we have two variables that in certain practical cases can be omitted – the number of hidden layers N_l and the type of activation function G. Under these conditions (2) becomes:

(2′) $N_h = F(N_i, N_o, N_t, T_l, E, W, B, V, I)$.

Is it possible to get rid of the influence of other of the described factors? The answer to this question is still open. Even the fact that developers of software designed for modeling of artificial neural networks have left it up to users to determine the number of neurons, without intervention and suggestions for the optimal option, indirectly but eloquently speaks of the lack of an accurate answer. Still, there are several different ways in which we can optimize both the training and the efficiency of the neural network.

Table 1. Basic approaches for determining the number of neurons in a neural network

Author, year	Formula
Li, J. Y., Chow, T. W. S., Yu, Y. L., 1995	$N_h = \frac{\sqrt{1+8\mathrm{N_i}}-1}{2}$
Tamura, S., Tateishi, M., 1997	$N_h = N_i - 1$ refers to 4-layer neural networks
Xu, S., Chen, L., 2008	$N_h = \frac{1}{2}\frac{N_t}{N_i \log \mathrm{N_t}}, \frac{N_t}{N_i} > 30$ $N_h = \frac{N_t}{N_i}, \frac{N_t}{N_i} \leq 30$
Shibata K., Ikeda, Y., 2009	$N_h = \sqrt{N_i N_o}$
Hunter, D., Yu, H. , Pukish III, M. S., Kolbusz, J., 2012	$N_h = \log(N_i - 1) - N_o$
Sheela, K., Deepa, S. N., 2013	$N_h = \frac{4\mathrm{N_i^2}+3}{\mathrm{N_i^2}-8}$

3. Influence of the Number of Inputs on the Number of Neurons in the Hidden Layer

In the process of research, some neural networks were constructed, approximating different types of functions with the same random sample generated in the closed interval [-1000,1000]. To provide ourselves with reasons for making comparisons, all studies were performed under the same conditions: in all cases the sample consists of 286 examples, 200 (75%) of which are used for training and 43 (15%) – for testing and validity, respectively.

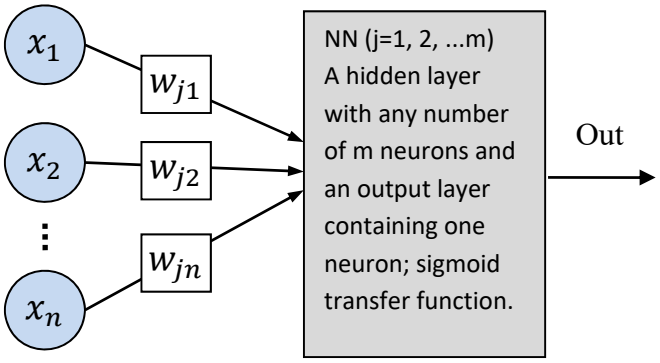


Figure 1. Neural network approximating the function (3)

Let's take a look at an n-dimensional linear function

$$(3)\ y = \sum_{i=1}^{n} a_i x_i + b\,, a_i, b \in R, n \in N$$

and an approximating neural network that meets Cybenko's requirements [7] (Fig. 1.).

In the experiments conducted we tried to determine which formula from those presented in Table 1. is most suitable for different functions and with different numbers of inputs and training examples.

3.1. Approximating Neural Networks on a Linear Function

The main parameters in the conducted experiments on neural networks approximating a linear function are the number of inputs N_i and training examples N_t. Specific values for some of them and the expected number of neurons N_h, according to various authors, are presented in Table 2. The training of the neural network in all experiments is performed using the Levenberg-Marquardt algorithm, the number of epochs in training – 1000, and for the activation function $Tanh\ (x) = \frac{2}{1+e^{-2x}} - 1$ is selected.

Table 2. Number of N_h neurons, according to existing theories for a specific number of N_i inputs and training examples N_i

Formula	$N_i = 4$ $N_t = 200$	$N_i = 1$ $N_t = 200$	$N_i = 50$ $N_t = 200$
$N_h = \frac{\sqrt{1 + 8\mathrm{N_i}} - 1}{2}$	2; 3	1	9; 10
$N_h = N_i - 1$	3	-	49
$N_h = \frac{1}{2}\frac{N_t}{N_i \log \mathrm{N_t}}$, $\frac{N_t}{N_i} > 30$, $N_h = \frac{N_t}{N_i}, \frac{N_t}{N_i} \leq 30$	4; 5	18; 19	4
$N_h = \sqrt{N_i N_0}$	2	1	7; 8
$N_h = \log(N_i - 1) - N_o$	1	-	2; 3
$N_h = \frac{4\mathrm{N_i^2} + 3}{\mathrm{N_i^2} - 8}$	8; 9	-	4; 5

In the case $\boldsymbol{N_i = 4, N_t = 200}$, (Table 3.), Sheela and Deepa's method has been the most successful in predicting the optimal number of neurons [3], where $N_h = \frac{4\mathrm{N_i^2}+3}{\mathrm{N_i^2}-8}$.

Table 3. Mean Square Errors (MSE) of networks with different numbers of hidden neurons, when $N_t = 200, N_i = 4$

Number of neurons	Mean Square Errors
1	4.46e-2
2	6.62e-3
3	4.12e-3
4	1.76e-5
5	1.23e-5
6	4.89e-4
7	9.67e-4
8	2.34e-5
9	1.01e-6
10	5.15e-4
11	4.81e-5
12	3.77e-3
13	5.81e-4

Similarly, at $\boldsymbol{N_i = 1, N_t = 20,}$ the optimal approximation is reached at 19 neurons in the hidden layer, which is predicted by the Xu and Chen model [4] $N_h = \frac{1}{2}\frac{N_t}{N_i \log \mathrm{N_t}}$.

In the case $\boldsymbol{N_i = 50, N_t = 200}$ with a significant increase in the dimensionality of the input vector, deterioration of the network performance is observed, leading to new requirements for the number of neurons. Despite the deteriorating performance of the network, it reaches its optimal status at 2 and 3 neurons, as assumed by Hunter and team [6] $N_h = \log(N_i - 1) - N_o$, in which is from MSE=1e-2 to 1e+2.

Another part of the conducted experiments show that the increased number of neurons does not significantly improve the efficiency of neural

networks in approximation. Moreover, with a significant increase in the number of neurons, the network error increases by several orders of magnitude. For example, in a network of 300 to 1000 neurons, MSEs of the order of +2 to +8 are observed. Also, in networks with 50 input variables, a significant decrease in efficiency is observed when the number of neurons increases above 7. The errors reach up to 1e + 6 and 1e + 8.

Table 4. Optimal number of neurons depending on the dimensionality of the input vector

Function	Dimensionality of the input vector	Optimal number of neurons
$y = \sum_{i=1}^{50} a_i x_i$	50	2;3
$y = \sum_{i=1}^{25} a_i x_i$	25	4
$y = \sum_{i=1}^{15} a_i x_i$	15	6
$y = \sum_{i=1}^{4} a_i x_i$	4	9
$y = x$	1	19

Ultimately, when approximating a linear function, the following trend is observed – as the number of inputs increases, the optimal number of neurons decreases (Table 4.).

3.2. Approximating Neural Networks on a Nonlinear Function

Similar conclusions are reached in attempts to approximate other nonlinear functions we have experimented with, such as:

- graded: $y = \sum_{i=1}^{n} a_i x_i^i, n \in N$;
- trigonometric: $y = \sum_{i=1}^{n} a_i \sin(x_i)$, $y = \sum_{i=1}^{n} a_i \cos(x_i), n \in N$, $y = \sum_{i=1}^{n} a_i tg(x_i)$, $y = \sum_{i=1}^{n} a_i cotg(x_i), n \in N$;
- logarithmic: $y = \sum_{i=1}^{n} a_i \log(x_i), n \in N$;
- exponential: $y = \sum_{i=1}^{n} a_i e^{x_i}, n \in N$.

3.3. Findings

The conducted experiments show that ***the dimensionality of the input vector influences the optimal number of neurons in the hidden layer. To achieve optimal network efficiency, as the number of input parameters increases, the number of neurons decreases.***

At first glance, as the number of input parameters increases, we expect an increase in the complexity of the topology of neural networks and automatically focus on finding networks with more neurons. However, as we have seen, this may prove to be the wrong strategy. For the considered types of functions, with the selected training algorithm and number of epochs, we found that ***the optimal number of neurons is inversely proportional to the dimension of the input vector:***

$$N_h \sim \frac{1}{N_i}.$$

4. Influence of the Number of Training Examples on the Number of Neurons in the Hidden Layer

When considering the influence of the number of training examples on the number of neurons, the approach is similar. Since the study of any of the factors that are thought to affect the number of neurons must be performed under equal conditions, the training of the neural network was again performed using the Levenberg-Marquardt algorithm, with the activation function being a hyperbolic tangent, the number of epochs in training – 1000, range of input data from -1000 to 1000. The number of examples is distributed as follows: 70% for training, 15% for testing, and 15% for validating.

Several neural networks for training were considered, for which samples with different number of examples were used but in the same segment. To avoid the influence of the bandwidth, the boundaries of the definition area were always present in the training examples, which leads to the replacement of the input vector with one whose coordinates are in the range [-1,1]. The results for a linear function are shown in Table 5.

Table 5. Optimal number of neurons depending on the number of training examples

Number of examples N_t	Optimal number of neurons N_h
1000	13
600	12
300	11
150	10
75	9
35	4
N_t<30	-

According to these results, the increase in the number of the training sample leads to the requirement for a larger number of neurons in the network. Similar results are also observed in the other cases of functions of different types: power, logarithmic, trigonometric, mixed.

The conclusion from the experiments conducted is that there is a tendency for ***proportional relationship between the number of training examples and the required neurons:***

$$N_h \sim N_t.$$

5. Optimization of the Number of Neurons in the Hidden Layer in Training Based on the Gauss and Newton Algorithm

We found that, although it seems strange, the increase in the number of inputs leads to the need to shrink the topology, expressed in a decrease in the number of neurons in the hidden layer, in order to maintain or increase the efficiency off the network. In other words, according to the conducted experiments, we found that the relationship between the number of signals on the input dendritic tree of the whole network and the required number of neurons is inversely proportional:

$$N_h \sim \frac{1}{N_i}.$$

On the other hand, increasing the number of used training examples necessitates increasing the number of neurons to meet efficiency requirements. I.e., the relationship here seems to be directly proportional:

$$N_h \sim N_t.$$

How can we logically explain the dependencies found?

Let's look again at the neural network with one hidden layer and a sigmoid transfer function (Fig. 1.). Let's accept we have m training examples:

$$\left\{[x_{j1}, x_{j2}, \dots x_{jn};\ t_j = F(x_{j1}, x_{j2}, \dots x_{jn})]\right\}_{j=1}^{m}$$

Each of them consists of an n-tuple of input data $(x_{j1}, x_{j2}, \dots x_{jn})$ and its corresponding value of the objective function $t_j = F(x_{j1}, x_{j2}, \dots x_{jn})$. In the consecutive submission of training examples:

$$[x_{11}, x_{12}, \dots x_{1n}\ ;\ t_1 = F(x_{11}, x_{12}, \dots x_{1n})]$$
$$[x_{21}, x_{22}, \dots x_{2n}\ ;\ t_2 = F(x_{21}, x_{22}, \dots x_{2n})]$$
$$\dots$$
$$[x_{m1}, x_{m2}, \dots x_{mn}\ ;\ t_m = F(x_{m1}, x_{m2}, \dots x_{mn})]$$

at the neural network output we have:

$$Out_1 = f(x_{11}, x_{12}, \dots\ x_{1n}, w_1, w_2, \dots\ w_p\ ),$$
$$Out_2 = f(x_{21}, x_{22}, \dots\ x_{2n}, w_1, w_2, \dots\ w_p\ )$$
$$\dots$$
$$Out_m = f(x_{m1}, x_{m2}, \dots\ x_{mn}, w_1, w_2, \dots\ w_p\ ),$$

where p is the number of possible connections between neurons in the network.

Network errors when submitting examples are:

$$e_1 = t_1 - f(x_{11}, x_{12}, \dots\ x_{1n}, w_1, w_2, \dots\ w_p\ )$$
$$e_2 = t_2 - f(x_{21}, x_{22}, \dots\ x_{2n}, w_1, w_2, \dots\ w_p\ )$$
$$\dots$$
$$e_m = t_m - f(x_{m1}, x_{m2}, \dots\ x_{mn}, w_1, w_2, \dots\ w_p)$$

The root mean square error (RMSE) of the grid is represented as follows:

$$E = \frac{\sqrt{e_1^2 + e_2^2 + \dots + e_m^2}}{m} = \frac{1}{m}\sqrt{\sum_{i=1}^{m} e_i^2}\ ,\ \text{i.e.}$$

$$E = \frac{1}{m}\sqrt{\sum_{i=1}^{m}[t_i - f(x_{i1}, x_{i2}, \dots\ x_{in}, w_1, w_2, \dots\ w_p\ )]^2}$$

As

$$\forall\ x_{ji}, t_j \in \left\{[x_{j1}, x_{j2}, \dots x_{jn};\ t_j = F(x_{j1}, x_{j2}, \dots x_{jn})]\right\}_{j=1}^{m}, i = 1,2, \dots n,$$

then each

$$\varphi_i = [t_i - f(x_{i1}, x_{i2}, \dots\ x_{in}, w_1, w_2, \dots\ w_p\ )]^2$$

for certain x_{ji} can be considered as a function of all possible weights in the network:

$$\varphi_i = \varphi_i(w_1, w_2, \dots\ w_p).$$

The task of training algorithms is to find such a set of weights $w_1, w_2, \dots\ w_p$, at which the error function is at its minimum:

$$E_{min} = \frac{1}{m}\sqrt{\sum_{i=1}^{m}[t_i - f(x_{i1}, x_{i2}, \dots\ x_{in}, w_1, w_2, w_3, \dots\ w_p\ )]^2}$$

In the best case $w_1, w_2, w_3, \dots\ w_p$ must be such that

$$E_{min} = 0, \text{ i.e.}$$
$$\sum_{i=1}^{m}[t_i - f(x_{i1}, x_{i2}, \dots\ x_{in}, w_1, w_2, \dots\ w_p\ )]^2 = 0.$$

One of the most commonly used methods for finding the minimum of the error function is the Gauss and Newton algorithm. In its essence, this is an iterative procedure aimed at finding the minimum of functions of the type:

$$S(z) = \sum_{i=1}^{m}[\varphi_i(z\ )]^2$$

In our case z are p-dimensional points

$$z = (w_1, w_2, w_3, \dots\ w_p), \text{ and}$$
$$\varphi_i(z\ ) = t_i - f(x_{i1}, x_{i2}, \dots\ x_{in};\ w_1, w_2, \dots w_p\ ), \text{ for each } i = 1,2, \dots m.$$

At initial network initialization with a random set of weights

$$z_0 = z_0(w_{01}, w_{02}, w_{03}, \dots\ w_{0p}),$$

for each iteration with a number k we have:

$$(4)\ z_{(k)} = z_{(k-1)} - \left[J_\varphi\left(z_{(k-1)}\right)^T J_\varphi\left(z_{(k-1)}\right)\right]^{-1} J_\varphi\left(z_{(k-1)}\right)^T \varphi(z_{(k-1)}),$$

where $\varphi = (\varphi_1, \varphi_2, \dots, \varphi_m), \forall\ \varphi_i(z)$:

$\varphi_i(z) = t_i - f(x_{i1}, x_{i2}, \dots x_{in}, w_1, w_2, \dots w_p)$, and

$J_\varphi\left(z_{(k-1)}\right)$ is the Jacobi matrix in p. $z_{(k-1)}$.

For the algorithms based on the Gauss and Newton method to work correctly, there is one important condition: the number of variables in the function φ_i in the sum $S(z) = \sum_{i=1}^{m}[\varphi_i(z)]^2$ must be less than the number of these function [8], [9], i.e.

$$p \leq m.$$

In cases when this condition is violated, the matrix $J_\varphi\left(z_{(k-1)}\right)^T J_\varphi\left(z_{(k-1)}\right)$ is irreversible and the iterations (4) cannot return unambiguous results.

But the dimension of z in our case is determined by the number of weights and thresholds in the neural network

$$z = (w_1, w_2, w_3, \dots w_p),$$

And the number of functions

$$\varphi_i(z) = t_i - f(x_{i1}, x_{i2}, \dots x_{in}, w_1, w_2, \dots w_p)$$

is determined by the number of training examples. This means something very important about the problem we are looking at: ***The number of all possible connections between the neurons in the network cannot be greater than the number of examples with which we train this network.***

Let's look at a network with one hidden layer in which there are q neurons and one output neuron, to which n inputs are applied (Fig. 2.). Let the neural network be trained through a method based on the Gauss-Newton algorithm using m training examples:

$$\left\{[x_{j1}, x_{j2}, \dots x_{jn};\ t_j = F\left(x_{j1}, x_{j2}, \dots x_{jn}\right)]\right\}_{j=1}^{m}.$$

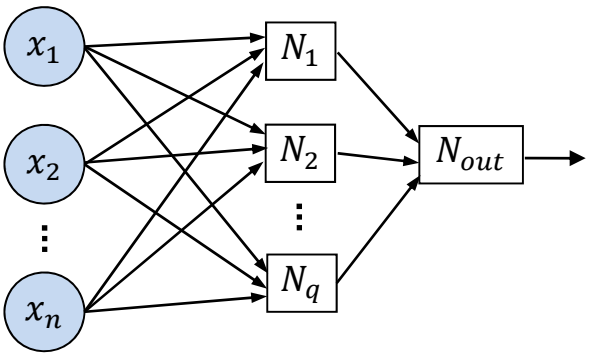


Figure 2. Artificial neural network with one hidden layer and one output neuron

Then, if p is the number of all possible connections consisting of weights and thresholds of neurons, since:

- the number of connections between inputs and neurons from the hidden layer is nq;
- the number of thresholds of the neurons in the hidden layer is q;
- the number of connections between the hidden neurons and the output neuron is q;
- the threshold of the output neuron is one, then

$$p = nq + q + q + 1,$$

whence

$$(5)\ p = q(n+2) + 1.$$

But the Gauss-Newton algorithm requires the number of variables in the function φ_i in the sum

$$S(z) = \sum_{i=1}^{m}[\varphi_i(z)]^2$$

to be lower than the number of these functions, i.e.

$$p \leq m.$$

It then follows from equation (5) that the number of neurons in the hidden layer must have an upper limit, and in particular:

$$(6)\ q \leq \frac{m-1}{n+2}$$

The requirement $p \leq m$ from which we derived the upper limit (6) is embedded in many optimization algorithms [9]. This condition also exists in the Levenberg-Marquardt method [10], [11], which we used when testing our idea.

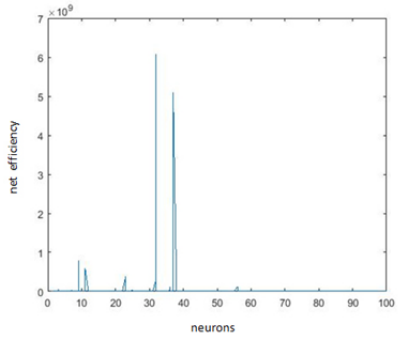
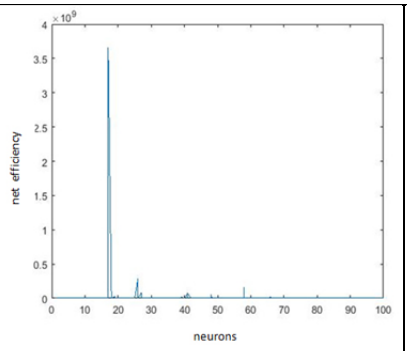
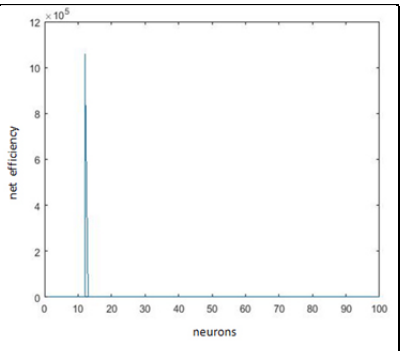
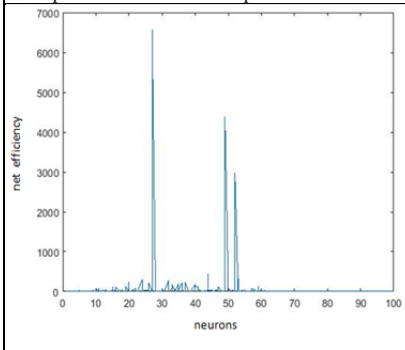
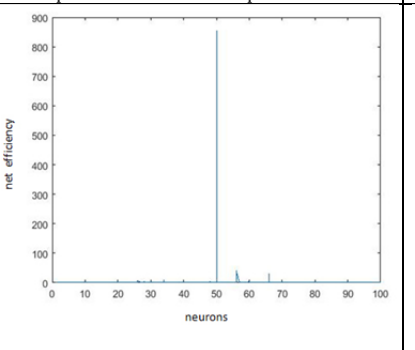
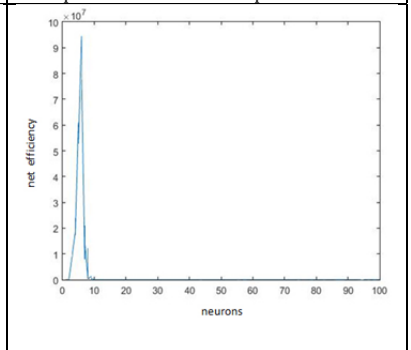
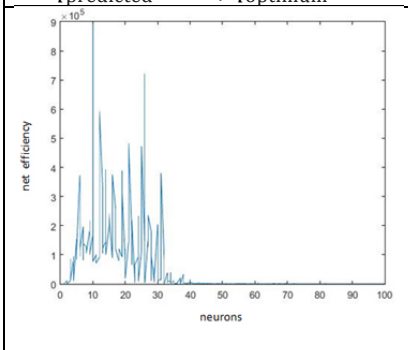
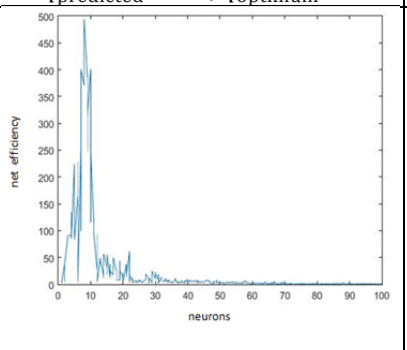
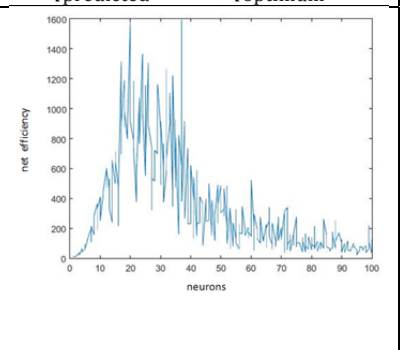
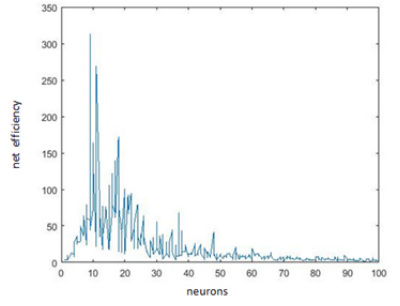
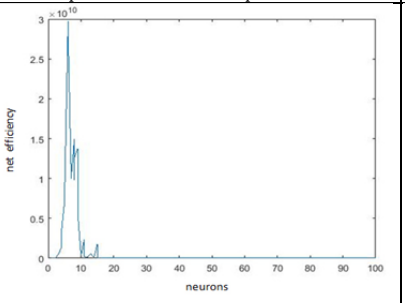
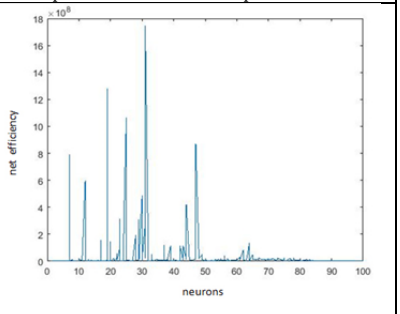
6. Experimental Verification of the Possibility of Optimal Number of Neurons Below the Established Upper Limit

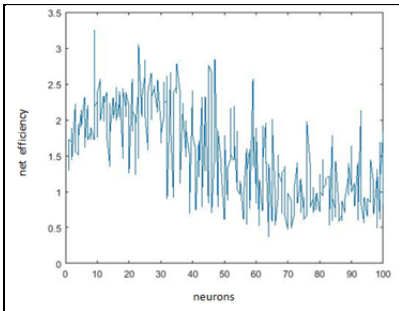
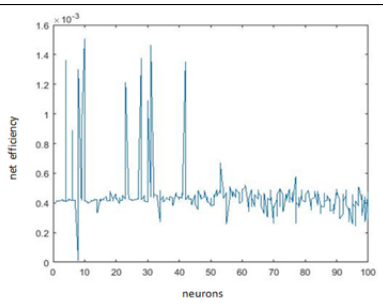
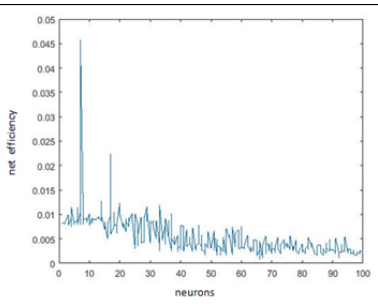
To check the existence of this upper limit outlining the required number of neurons, we introduced the network efficiency indicator:

$$net\ efficiency = \frac{1}{performance},$$

where $performance$ is the total performance of the network during the training, validation and testing. Studies were performed for a possible prediction of the maximum number of required neurons, for the most optimal network according to (6), for the following types of functions: linear, square, power, logarithmic, exponential and trigonometric. We used the Levenberg-Marquardt method to train the neural networks. Table 6. presents some of the results for specific functions, values for m and n,

Table 6. Peaks in the efficiency off neural networks depending on the number of neurons, when approximating randomly selected functions

		
$F = 2x_1 + 3x_2 - 4x_3 + 1,$ n= 3, m = 200, $q_{\text{predicted}} \leq 39$, $q_{\text{optimum}} = 32$	$F = 2x_1 + 3x_2 - 4x_3 + 5x_4 + 1,$ n = 4, m = 280, $q_{\text{predicted}} \leq 46$, $q_{\text{optimum}} = 17$	$F = 2x_1^2 + 3x_1 - 1,$ n = 1, m = 70, $q_{\text{predicted}} \leq 23$, $q_{\text{optimum}} = 12$
		
$F = 3x_1 + 2x_2^2,$ n = 2, m = 200, $q_{\text{predicted}} \leq 48$, $q_{\text{optimum}} = 27$	$F = 2x_1^2 + 5x_2^2 + 3x_3^2 + 1,$ n = 3, m = 280, $q_{\text{predicted}} \leq 55$, $q_{\text{optimum}} = 50$	$F = log(x_1),$ n = 1, m = 11, $q_{\text{predicted}} \leq 55$, $q_{\text{optimum}} = 6$
		
$F = log(x_1 + x_2),$ n = 2, m = 105, $q_{\text{predicted}} \leq 26$, $q_{\text{optimum}} = 10$	$F = log(x_1 + x_2x_3),$ n = 3, m = 49, $q_{\text{predicted}} \leq 9$, $q_{\text{optimum}} = 8$	$F = log(x_1x_2 + x_3x_4),$ n = 4, m = 245, $q_{\text{predicted}} \leq 40$, $q_{\text{optimum}} = 37$
		
$F = \log[(x_1x_2 + x_3x_4)x_5],$ n = 5, m = 140, $q_{\text{predicted}} \leq 19$, $q_{\text{optimum}} = 9$	$F = \sin(x_1),$ n = 1, m = 70, $q_{\text{predicted}} \leq 23$, $q_{\text{optimum}} = 6$	$F = \cos(x_1 + x_2),$ n = 2, m = 280, $q_{\text{predicted}} \leq 69$, $q_{\text{optimum}} = 31$

		
$F = \sin(x_1) + \cos(x_2 x_3)$, $n = 3, m = 126$, $q_{\text{predicted}} \leq 25$, $q_{\text{optimum}} = 9$	$F = \text{tg}(x_1 + x_2 x_3) + \sin(x_1 x_4)$, $n = 4, m = 350$, $q_{\text{predicted}} \leq 58$, $q_{\text{optimum}} = 10$	$F = \sin(x_1 + x_2) + \cos(x_3 + x_4) + ctg(x_5)$, $n = 5, m = 350$, $q_{\text{predicted}} \leq 49$, $q_{\text{optimum}} = 10$

calculated value for the upper limit of $q - q_{\text{predicted}}$ – by formula (6), as well as the corresponding optimal value q_{optimum} obtained in the experiments.

If the number of the required variables $(w_1, w_2, w_3, \ldots w_p)$ and the number of functions φ_i in the sum are equal

$$S(z) = \sum_{i=1}^{m} [\varphi_i(z)]^2$$

for $z = (w_1, w_2, w_3, \ldots w_p)$, i.e. at

$$p = m,$$

the Jacobian of the function φ is a square nonsingular matrix:

$$\det J \neq 0$$

and with an appropriate set of initial weights from which to start the iteration process, a square rate of convergence is observed [12], [13]. Thus, the choice of the number of neurons according to the equation:

$$q = \frac{m-1}{n+2},$$

where m is the number of training examples and n is the number of inputs, will provide optimal conditions for operation of the algorithms with appropriate initial initialization of the weights.

If we consider the case

$$q > \frac{m-1}{n+2},$$

it should be noted that the Levenberg-Marquardt optimization algorithm is able to work even under conditions that in practice lead to

$$p > m.$$

Here, however, the greater number of neurons, although not an obstacle to finding a set of weights, can easily degrade the quality of the created neural network. In these cases, there is a risk of network overtraining. With a higher number of neurons, overtraining is often observed, in which the quality of training is significantly improved, but this is at the expense of severely deteriorated test results. Although very high values of the network training efficiency are observed and it covers the sample with high accuracy, in fact its prognostic abilities are very weak, taking into account very low levels of efficiency in the validation and test.

Ultimately, our proposal for optimizing the search is not to construct an infinite number of neural networks, but ***to limit the modeling process to networks with number of neurons***

$$\boldsymbol{q \leq \frac{m-1}{n+2}.}$$

This formula sets an upper limit on the required neurons, which will save time and resources in the search for the optimal structure.

7. Upper Limit of the Number of Neurons in Neural Networks With r Hidden Layers Containing an Equal Number of Neurons

Let a neural network be given with r hidden layers containing an equal number of neurons – q (Fig. 3.).

It can be seen that the number of all connections between the neurons from different layers (in the absence of feedback) with added thresholds is:

$$p = (r-1)q^2 + (r+n+2)q + 1,$$

hence the number of neurons q in each layer is a solution of the quadratic equation

$$(r-1)q^2 + (r+n+2)q + 1 - p = 0.$$

Its square roots are

$$q_{1,2} = \frac{-(r+n+2) \pm \sqrt{(r+n+2)^2 - 4(r-1)(1-p)}}{2(r-1)},$$

And one of them

$$q = \frac{-(r+n+2) - \sqrt{(r+n+2)^2 - 4(r-1)(1-p)}}{2(r-1)}$$

has no real physical meaning for neural networks with more than one layer (r>1), due to the positive values of n. Thus, the relationship between the number of neurons q in each layer and the number p

of all possible weights and thresholds in the network is presented by the expression

$$q_1 = \frac{-(r+n+2)+\sqrt{(r+n+2)^2-4(r-1)(1-p)}}{2(r-1)},$$

which can also be written like this:

$$q = \frac{\sqrt{(r+n+2)^2+4(r-1)(p-1)}-(r+n+2)}{2(r-1)}$$

Taking into account the condition $p \le m$, under which the algorithms optimizing the error function are considered, we can determine the upper limit of the number of neurons in each layer, expressed by the inequality

$$q \le \frac{\sqrt{(r+n+2)^2+4(r-1)(m-1)}-(r+n+2)}{2(r-1)}.$$

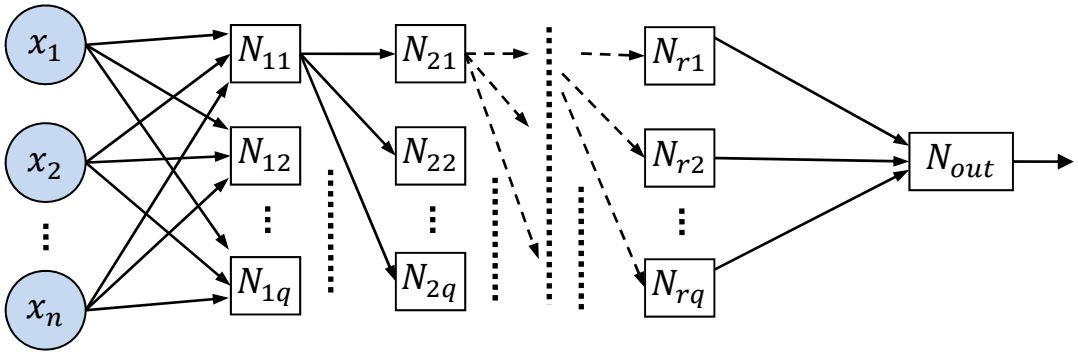


Figure 3. An artificial neural network with r hidden layers containing an equal number of neurons

The total number of neurons for the entire network must be chosen by the condition:

$$Q \le \frac{\sqrt{(r+n+2)^2+4(r-1)(m-1)}-(r+n+2)}{2(r-1)} r.$$

With the number of neurons in each layer performing the equations:

$$q = \frac{\sqrt{(r+n+2)^2+4(r-1)(m-1)}-(r+n+2)}{2(r-1)},$$

we have the number of all possible connections in the network $p = m$, which guarantees us a square and non-singular Jacobi matrix. This in turn means that an appropriate choice of the initial weights will allow quadratic convergence of training algorithms.

In networks with several layers and a random number of neurons in each layer, due to the need to consider some of them as parameters, there are infinitely numerous implementations of the connection $q = f(p)$.

8. Conclusion

What is the optimal number of neurons in a neural network? This is one of the most frequently asked questions that arises when constructing this type of artificial intelligence. Many "magical formulas" are also proposed, which later turn out to have only episodic success, and that is in specific cases. Experience has brought the understanding that it takes an individual approach to specify the number of artificial neurons for each task. Our study aims, using the condition

$$p \le m,$$

where the algorithms optimizing the error function are considered, to indicate the upper limit of the number of neurons in the hidden layer. This condition, however, has a practical consequence, limiting the demand for a neural network.

When the conditions

$$q \le \frac{m-1}{n+2},$$

are met, where n is the number of inputs and m is the number of training examples for neural networks with one hidden layer, or

$$Q \le \frac{\sqrt{(r+n+2)^2+4(r-1)(m-1)}-(r+n+2)}{2(r-1)} r$$

for neural networks with r>1 layers, the search for a network does not need to continue to a very large number of neurons.

If we still need to focus on a specific choice for the upper limit of ***the number of neurons in the hidden layer, for a single-layer network, it is:***

$$\boldsymbol{q = floor\left(\frac{m-1}{n+2}\right)},$$

and for ***multilayer networks, with an equal number of neurons in the individual layer, it is:***

$$\boldsymbol{q = floor\left[\frac{\sqrt{(r+n+2)^2+4(r-1)(m-1)}-(r+n+2)}{2(r-1)}\right]},$$

where the floor function returns the argument rounded to the smaller integer. Under these

conditions we will have a Jacobi J square matrix, an easily computable inverse matrix

$$[J_{\varphi}(z_{(k-1)})^{T} J_{\varphi}(z_{(k-1)})]^{-1}$$

and fast convergence rate in algorithms using similar procedures as the Gauss-Newton or Levenberg-Marquardt. In these cases, as the number of epochs in training increases, excellent results should be expected.

Finally, we note that the increased number of neurons violating the described conditions leads to an increase in the network connections and the implementation of the inequality $p > m$. This can easily lead to an inability to determine the Jacobi inverse matrix. Although the algorithms used can create networks even under these conditions, their respective constructs may turn out over-trained: with a very small training error but severely deteriorated validation and test results, which in turn means poorer predictive capabilities and erroneous calculations outside the set of training examples.

Acknowledgements

The work is funded by the SP19-FMI-012 project at the Research Fund of the Plovdiv University "Paisii Hilendarski".

References

[1]. Li, J. Y., Chow, T. W., & Yu, Y. L. (1995, November). The estimation theory and optimization algorithm for the number of hidden units in the higher-order feedforward neural network. In *Proceedings of ICNN'95-International Conference on Neural Networks* (Vol. 3, pp. 1229-1233). IEEE.

[2]. Tamura, S. I., & Tateishi, M. (1997). Capabilities of a four-layered feedforward neural network: four layers versus three. *IEEE Transactions on Neural Networks*, *8*(2), 251-255.

[3]. Sheela, K. G., & Deepa, S. N. (2013). Review on methods to fix number of hidden neurons in neural networks. *Mathematical Problems in Engineering*, *2013*.

[4]. Xu, S., & Chen, L. (2008). A novel approach for determining the optimal number of hidden layer neurons for FNN's and its application in data mining. In *International Conference on Information Technology and Applications: iCITA* (pp. 683-686).

[5]. Shibata, K., & Ikeda, Y. (2009, August). Effect of number of hidden neurons on learning in large-scale layered neural networks. In *2009 ICCAS-SICE* (pp. 5008-5013). IEEE.

[6]. Hunter, D., Yu, H., Pukish III, M. S., Kolbusz, J., & Wilamowski, B. M. (2012). Selection of proper neural network sizes and architectures—A comparative study. *IEEE Transactions on Industrial Informatics*, *8*(2), 228-240.

[7]. Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, *2*(4), 303-314.

[8]. Niclas Börlin. (2007). Nonlinear Optimization Least Squares Problems — The Gauss-Newton method. Retrieved from: https://www8.cs.umu.se/kurser/5DA001/HT07/lectures/lsq-handouts. [accessed: 20 June 2020].

[9]. Madsen, K., Nielsen, H. B., & Tingleff, O. (2004). Methods for non-linear least squares problems. Retrieved from: http://www2.imm.dtu.dk/pubdb/edoc/imm3215.pdf. [accessed: 02 August 2020].

[10]. Bellavia, S., Gratton, S., & Riccietti, E. (2018). A Levenberg–Marquardt method for large nonlinear least-squares problems with dynamic accuracy in functions and gradients. *Numerische Mathematik*, *140*(3), 791-825.

[11]. Ranganathan, A. (2004). The Levenberg-Marquardt Algorithm. Retrieved from: http://www.ananth.in/Notes_files/lmtut.pdf. [accessed: 05 August 2020].

[12]. Yamashita, N., & Fukushima, M. (2001). On the rate of convergence of the Levenberg-Marquardt method. In Topics in numerical analysis (pp. 239-249). Springer, Vienna.

[13]. Ahookhosh, M., Artacho, F., Fleming, R. Vuong, P. (2010). Local convergence of the Levenberg–Marquardt method under Holder metric subregularity, Advances in Computational Mathematics, vol. 45, 2771–2806.