An Implementation of a Generic Scheme of an Artificial Neural Network and the Backpropagation algorithm in C++

Done Stojanov*, Jasmina Veta Buralieva** and Aleksandar Velinov***
Faculty of computer science, University „Goce Delčev"–Štip, Republic of North Macedonia
*done.stojanov@ugd.edu.mk, **jasmina.buralieva@ugd.edu.mk, ***aleksandar.velinov@ugd.edu.mk

Abstract – Due to the learning capabilities of artificial neural networks (ANNs), they are commonly used for solving complex problems, such as: prediction, optimization, approximation and recognition. To be able to solve such complex problems, an artificial neural network (ANN) has to be trained. The commonly used algorithm for that purpose is the backpropagation algorithm, which is a supervised learning approach. Therefore in this paper we present a generic scheme of one-layer artificial neural network and we apply the backpropagation algorithm. To carry out an analysis, we coded the structure of the artificial neural network and the backpropagation algorithm in C++.

Keywords - backpropagation, algorithm, neural, network, C++

I. INTRODUCTION

Artificial neural networks (ANNs) are biologically inspired computing models, approximating the way brain functions. Structurally an artificial neural network consists of a set of connected artificial nodes called neurons. Throughout connections, usually regarded as synaptic edges, an artificial neuron takes the inputs and computes the output as a non-linear function of the sum inputs. Note that edges simulate real life synapses, passing electrical signals from one cell to another. Usually there are several layers of neurons built into one neural network. Each layer is a vector of neurons computing the output that is taken as an input to the next layer. A weight that is a real number is assigned to each synaptic edge. Weights are dynamically updated (increased or decreased) throughout training and learning in order to be able to reach the output for a specific input.

The very early beginnings of this topic can be traced back to 1943 when McCulloch and Pitts [1] designed the first model of an artificial neuron. Few years later, in 1949, Donald Hebb published his work known as "The Organization of Behavior" [2] which introduced the law for neuron learning, based on straight-forward synaptic propagation. Later this law became known as Hebbian Learning. Two years later Marvin Minsky created the first ANN. John von Neumann's book "The Computer and the Brain" [3] published in 1958 had big impact on the topic and radically changed the whole approach. Frank Rosenblatt [4] created the Perceptron in 1958. However, the Perceptron works only fine for linearly separable input data sets and this constrain was first reported by Marvin Minsky and Seymore Papert [5] in 1969. Minsky and Papert reported that the Perceptron in not able to classify non-linearly separable input data sets. However, the technology of Perceptron was not a fail but rather a case of an upgrade for more advance application. A three layers architecture that can be applied for non-linearly separable data was presented in 1992 by Hecht-Nielsen [6].

There are several types of artificial neural networks that can be applied to specific problems. Feed forward neural networks are commonly used in computer vision and speech recognition, because they can easily cope with noisy data. There is no backpropagation and the neuron fires an output if the activation passes certain threshold. Radial basis function neural networks distinguish other neural networks due to their fast learning speed. These neural networks consist of three layers. The first layer corresponds to the input, the second layer consists of units with non-linear radial basis activation function and the third layer corresponds to the output. This type of neural network is commonly used for classification, system control and function approximation. Self-organizing neural networks [7], [8] are best for pattern classification. Input patterns are compared to known patterns and they are associated to the best match. Once the best match has been found, cluster's weights needs to be updated. Metrics is based on the square of minimum Euclidean distance. Recurrent neural networks utilize the fact that neurons are able to memorize

some of the information that had in the previous step. Feeding back the output to the input, these neural networks are commonly used for text into speech conversion. Most of nowadays computer vision technologies explore convolutional neural networks due to their high accuracy in image and signal processing. Neurons of this type of networks have learnable weights and biases. However, sometimes we have to combine different types of neural networks that will work independently towards the output and here comes the concept of modular neural networks. The main advantage of this type of neural network is the ability to decompose large computational process into smaller tasks, making them suitable for implementation into multi-module decision systems.

The backpropagation algorithm is a well-known and popular algorithm for training artificial neural networks. This algorithm was invented in the early 60's and it was implemented to run on computers in 70's. Even thought Werbos [9] in 1974 was the first that showed that this algorithm can be applied to neural networks, it took few decades until this algorithm was popularized again in a paper called "Learning representations by back-propagating errors" written by Rumelhart, Hinton and Williams [10]. The backpropagation algorithm is a supervised learning technique which is based on Widrow-Hoff learning rule. The core of this algorithm is that it starts with random weights assigned to the synaptic edges and the goal is to adjust them until the artificial neural network learns from training data. Note that this is done in thousands and sometimes millions of iterations of error updates.

In this paper we introduce a generic scheme of an artificial neural network and the application of the backpropagation algorithm. To be able to simulate the execution of this algorithm, we programmed it in C++.

II. MATERIALS AND METHODS

The general architecture of a neural network is shown on Figure 1. It consists of three layers: an input layer, hidden layer and output layer. To understand how neural networks work, we have to consider the artificial neuron, Figure 2. The neuron takes N inputs $A_1 A_2 \dots A_N$ throughout N synaptic edges weighted from W_1 to W_N plus threshold θ – Figure 2. The output of the neuron equals the activation function of the sum of products x such as : $x = \sum_{i=1}^{N} A_i W_i + \theta$, Figure 2. The activation function makes the decision if a given input signal is considered relevant or not and thus firing (activating) the neuron or not.

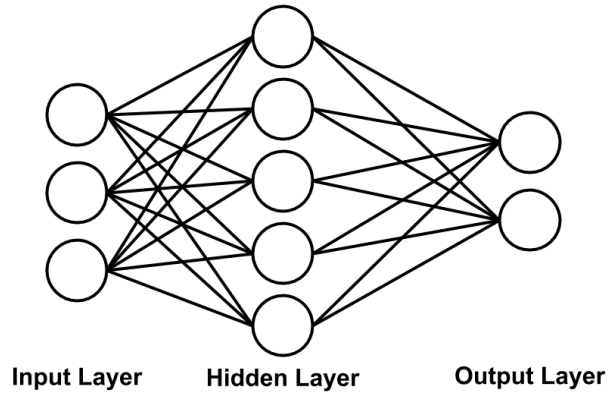

Figure 1. General architecture of an artificial neural network

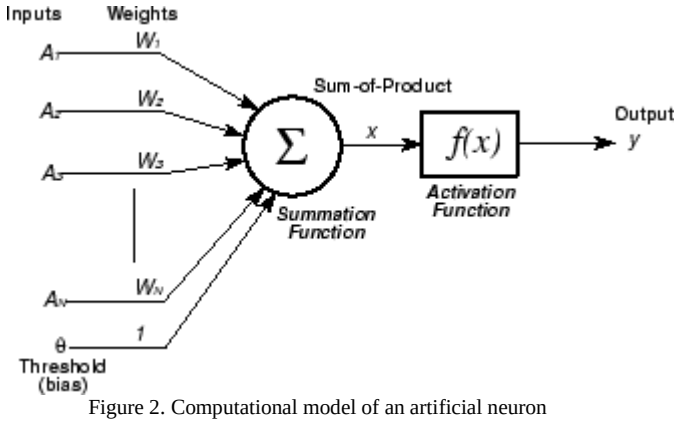

Figure 2. Computational model of an artificial neuron

Even though there are different types of activation functions, some of them, such as: the binary step, linear, sigmoid and tangent hyperbolic activation functions are commonly used. The simplest of all activation functions is the binary step activation function. This function serves as a threshold classifier. The neuron is activated ($y = 1$) if that the sum of products x is greater than certain threshold, otherwise not ($y = 0$), Figure 3.

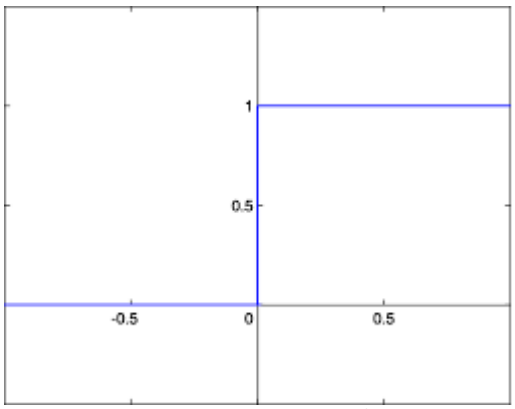

Figure 3. Binary step activation function

The equation of the linear activation function is $y = cx$ or the sum of products x is multiplied by a

constant c, Figure 4. This activation function is used in the output layers.

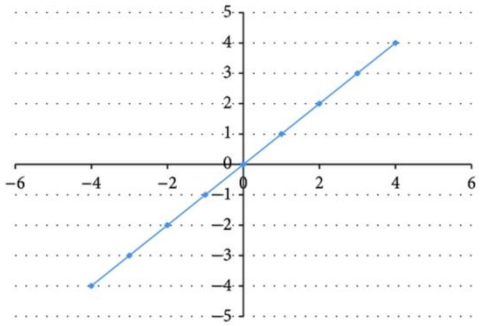

Figure 4. Linear activation function

Sigmoid activation function is a non-linear function. The equation of this function is $y = 1/(1 + e^{-x})$ and it can be plotted as 'S' shaped graph, Figure 5. The output of the neuron ranges between 0 and 1 and small changes of x around 0 will result in major change of y. This activation function is also used in the output layers.

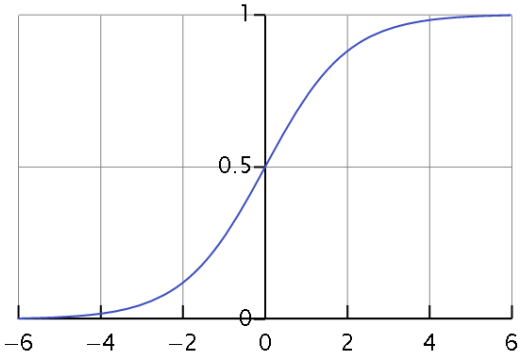

Figure 5. Sigmoid activation function

Tangent hyperbolic activation function (tanh) is basically shifted sigmoid faction, Figure 6. The output of a neuron using this type of activation ranges between -1 and 1 and it is usually used in the hidden layers of neural networks.

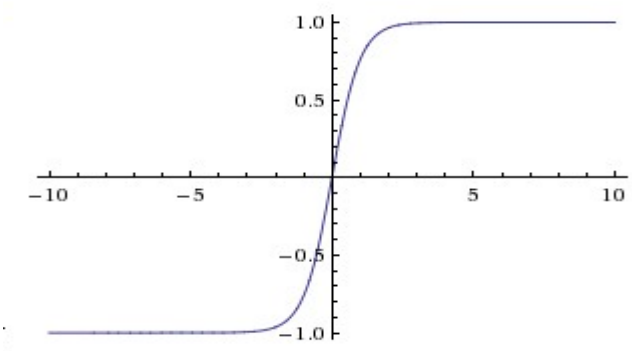

Figure 6. Tanh activation function

Figure 7 shows the structure of the neural network on which we applied the backpropagation algorithm. It takes k inputs $i_1 i_2 \dots i_k$ and consists of n neurons $n_1 n_2 \dots n_n$ that generate also n outputs $o_1 o_2 \dots o_n$. All neurons use sigmoid activation function, thus calculating the output o_r as $o_r = 1/(1 + e^{-x_r})$ such as the sum of products $x_r = \sum_{i=1}^{k} i_i w_{i,r}$. Note that the notation $w_{i,r}$ denotes the weight of the synaptic edge connecting the $i'th$ input to the $r'th$ neuron, Figure 7. Our goal is to obtain a vector of desired outputs $[do_1, do_2, \dots, do_n]$ for an input vector $[i_1 i_2 \dots i_k]$ applying the backpropagation algorithm that does so in a process of iterative update of the weights of synaptic edges $w_{i,r}$. The way the backpropagation algorithm does this is after computing the output of the $r'th$ neuron o_r an error e_r is computed such as $e_r = (do_r - o_r)(1 - o_r)o_r$ and this is done for all neurons in the architecture. Once the error for the $r'th$ neuron was computed, the weight of synaptic edge $w_{i,r}$ must be updated to $w_{i,r} + e_r i_i$ providing that the output o_r in the following iteration will come closer to the desired output do_r than it was in the previous iteration. This mechanism allows network convergence and the whole process is known as network training or learning.

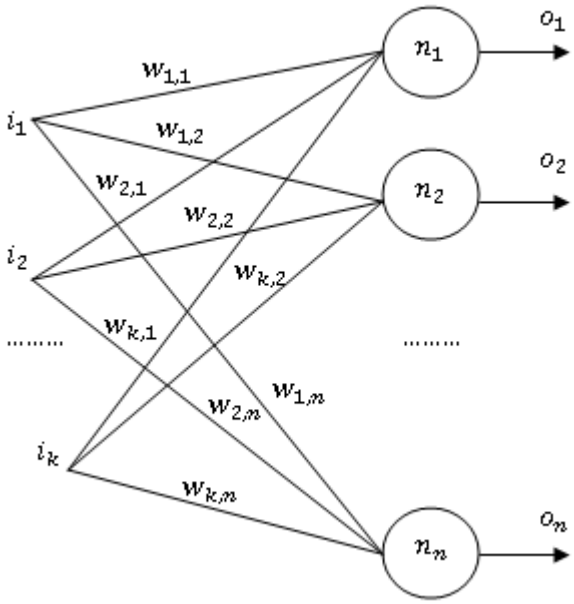

Figure 7. Structure of the test network

III. RESULTS

We used object-oriented programming in C++ to implement the network and the training algorithm. The structure of the neural network that is shown on Figure 7 was abstracted by a class *NeuralNetwork* with inputs, outputs and desired outputs being implemented as float arrays and a set of initial synaptic weights $w_{i,r}$ being implemented in form of matrix. The initial set up of the synaptic weights is done by a constructor, while the backpropagation algorithm is implemented as a special function within the class based on the explanation in the previous section. The main program accepts input and desired output vectors Figure 8 and performs training calls upon the network, having previously set up the initial weights of the synaptic edges, Figure 9. In order to understand what happens throughout the learning process, we printed the results on the screen.

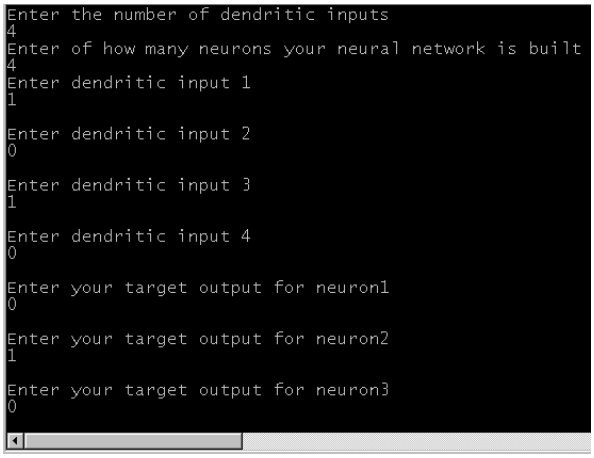

Figure 8. Setting the input and desired output vector

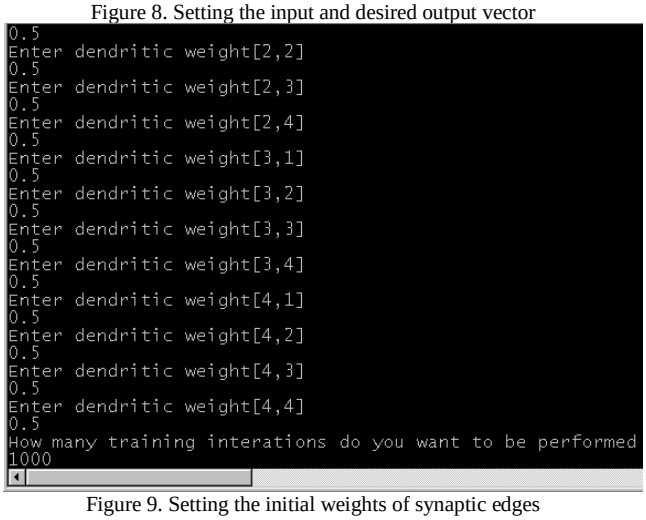

Figure 9. Setting the initial weights of synaptic edges

Our neural network was tested for $[1,0,1,0]$ input vector and $[0,1,0,1]$ desired output vector Figure 8, i.e. we wanted to train the network to work as an inverter of the input. Therefore we analyzed what happened throughout the training process in: 100, 500, 1000, 5000 and 10000 iterations, Table 1. At the end of each epoch we calculated the absolute error between the current outputs of the neurons and the goal values. To improve the uniformity of the process, all $w_{i,r}$ of the synaptic edges were initially set up to $0,5$.

For this scenario, from the obtained results we can see that most of the convergence of the neural network to the goal happens in the first 100 and 500 iterations, Table 1 and Figure 10. The absolute error between the current output and the desired output continues to decrease having increased the number of training iterations (1000, 5000 and 10000) but the rate of this change is not as sharp and significant as it was throughout the first couple of training iterations. As results clearly show, after 10000 iterations the absolute error between the output and the goal is about 0,005 or we can consider that our neural network was almost trained to function as a digital inverter, Table 1, Figure 10 and Figure 11.

TABLE I. Outputs of the neurons for different number of training iterations

Number of iterations	Neuron	Output of neuron	Desired Output	Abs. error
100	n1	0,0565899	0	0,0565899
	n2	0,946133	1	0,053867
	n3	0,0565899	0	0,0565899
	n4	0,946133	1	0,053867
500	n1	0,0235008	0	0,0235008
	n2	0,976719	1	0,023281
	n3	0,0235008	0	0,0235008
	n4	0,976719	1	0,023281
1000	n1	0,0163614	0	0,0163614
	n2	0,983715	1	0,016285
	n3	0,0163614	0	0,0163614
	n4	0,983715	1	0,016285
5000	n1	0,00717572	0	0,00717572
	n2	0,992831	1	0,007169
	n3	0,00717572	0	0,00717572
	n4	0,992831	1	0,007169
10000	n1	0,00505166	0	0,00505166
	n2	0,994951	1	0,005049
	n3	0,00505166	0	0,00505166
	n4	0,994951	1	0,005049

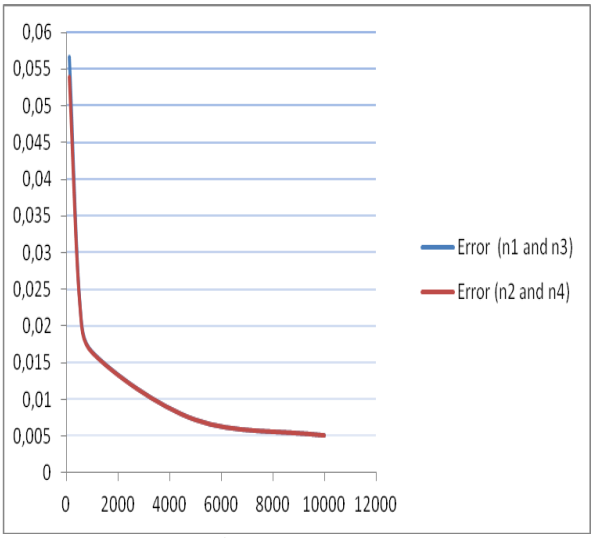

Figure 10. Convergence of the neural network throughout the training process

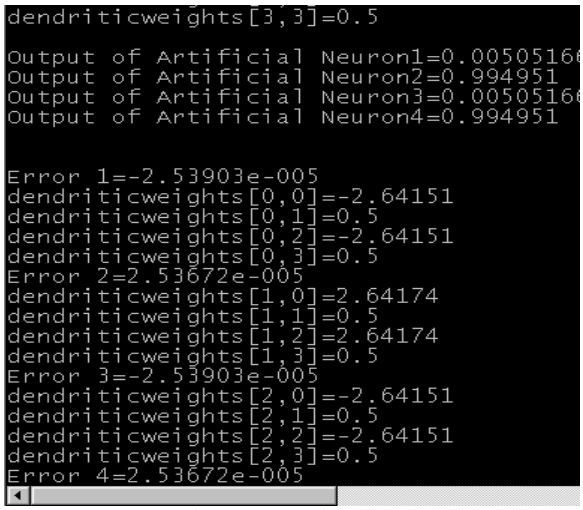

Figure 11. Output of the network after 10000 training iterations

IV. CONCLUSION

In this paper we analyzed the structure and the training of an artificial neural network. This cutting-edge technology emulates the way our brains work and recognize everyday's patterns upon ours experience. Neural networks map this concept in the domain of the artificial intelligence to construct machines that can actually act intelligently in unpredicted conditions. However, in order to achieve that, the structure needs to be trained and here comes the main challenge, i.e. more training is performed the more accurate the recognition is. In this paper we used the backpropagation algorithm and we showed that in the case of uniform initial weights setup, the network learns most of the things in the first couple of training iterations.

REFERENCES

[1] McCulloch, S. Warren, and W. Pitts. "A logical calculus of the ideas immanent in nervous activity," The bulletin of mathematical biophysics, vol.5.4, pp. 115-133, 1943.

[2] H.D. Olding. The organization of behavior: A neuropsychological theory. Psychology Press, 2005.

[3] J.V. NEUMANN. "The Computer and the Brain–The Silliman lectures," 1958.

[4] F. Rosenblatt. "The perceptron: a probabilistic model for information storage and organization in the brain," Psychological review, vol.65.6, pp. 386-408, 1958.

[5] M. Minsky, and P.A. Seymour. Perceptrons: An introduction to computational geometry. MIT press, 2017.

[6] R. Hecht-Nielsen. "Theory of the backpropagation neural network," in Neural networks for perception, Academic Press, 1992, pp.65-93.

[7] T. Kohonen . Self-organizing and associative memory, 3rd ed., Berlin:Springer-Verlag, 1989.

[8] T. Kohonen. "A self-learning musical grammar, or 'Associative memory of the second kind'," in International Joint Conference On Neural Networks. 1989, Chap. 1: pp. 1 – 5.

[9] P. Werbos. "Beyond regression:" new tools for prediction and analysis in the behavioral sciences," Ph.D. dissertation, Harvard University, 1974.

[10] D.E. Rumelhart, G.E. Hinton, and R.J. Williams. "Learning representations by back-propagating errors," Nature, vol.323.6088, pp. 533-536, 1986.