A C LANGUAGE APPLICATION FOR OBTAINING AN INCREASED FAULT TOLERANCE FOR MLP NEURAL NETWORKS

Alexandru ENE, Cosmin ŞTIRBU
FECC, University of Pitesti, Romania
alexandru.ene@upit.ro, cosmin.stirbu@upit.ro

Keywords: MLP neural networks, increased fault tolerance, C application.

Abstract: *We present a C application through which we train a MLP (multilayer perceptron) neural network, and, using the backpropagation algorithm, we obtain more valid sets of weights. Although each from these sets of weights assures the convergence of the network, we choose only one set of weights, the one for which the neural network has the best fault tolerance. We analyse all the p - fault hidden neurons combinations , where p is a natural number less than the number of neurons from the hidden layer.*

1. INTRODUCTION

Artificial neural networks are made from interconnected processing units (artificial neurons). There are many architectures of neural networks, and one of the most popular architecture is MLP. They are typically used in pattern recognition applications (hand written text recognition, diagnosis in medical or industrial field, military targets recognition) or for the estimation of the trend of a process (financioal, industrial, etc) [2],[4].

The first step in designing a MLP neural network is to determine the number of input neurons, the number of layers of hidden neurons, the number of neurons in each hidden layer and the number of output neurons. Some of these parameters are experimentally chosen (the number of hidden layers and the number of neurons in each hidden layer) and some are determined directly from the problem to be solved (the number of input and output neurons). For instance if we have a problem in which we have to recognise hand written digits that are presented to the network as a 10x10 pixels matrix (each pixel is either 0 either 1), we use a neural network with 10x10=100 input neurons and 10 output neurons, as we have 10 different digits. In opther applications we have more choices for the input number of neurons and for the output number of neurons. For example, in a diagnosis problem in which an anlysed object has to be classified as "good" or "bad", we can use a single output neuron (if its output is over a threshold, the object is "good", if its output is below a threshold, the object is classified as "bad") or, we can have two output neurons (If the output of the first output neuron is high and the output of the second outpuit neuron is low, then the object is "good" . If the output of the first output neuron is low and the output of the second outpuit neuron is high, then the object is "bad" .)

For the hidden layer and for the output layer there are typically used sigmoidal artificial neurons. The output y of a sigmoidal neuron has the following formula:

$$y = \frac{1}{1 + e^{-activation}} \quad (1)$$

and

$$activation = \sum x_i w_i \quad (2)$$

So the activation is the weighted sum of all outputs of neurons from the previous layer, that are connected with the current neuron, (each connection between two neurons has a real

numbere associated with it, called weight). In order to solve a specific problem, after the architecture is chosen, the network is traine using a set of training examples. For the MLP architecture the training is typically done with the backpropagation algorithm. When the training succeeds (the network learned all training examples within a specified global error), we say that the nework converged. As a result of the training, a set of weights is obtained. If we repeat the training, due to the fact that the backpropagation algorithm starts with a random set of weights, when the neural network converges, we obtain another set of weights, that is different from the previously obtained set of weights.

After the training is done, the neural network is checked to see if it continues to generate correct answers for input patterns that are different from the training examples. All these stages: training and training validation are typically done through a software simulator [1].

In order to obtain an increase in the speed of answering of a neural network that is used to solve a scertain problem, the neural network could be hardware implemented. The fault tolerance [3] of a neural network analyzed in this article is for the hardware implementatuion of a MLP neural network.

2. THE METHOD USED FOR OBTAINING AN INCREASED FAULT TOLERANCE

Using a certain set of weights for which the neural network converges, we analyse the fault tolerance of the network, in the presence of faulty hidden neurons. We consider as a typical faulty neuron, the neuron whose output was short circuit, as illustrate in the next figure.

So, because the neuron H_k is faulty, we have $yH_k=0$. When the neural network is used to solve a specific problem, it computes all the outputs (yO_0, yO_1,…,yO_N) starting from the input pattern (this is the forward propagation of the inputs). The faulty output yH_k appears only in the computation of the activation for each neuron from the output layer. The activation of a neuron is the weighted sum of all its inputs.

($activation = \sum x_i w_i$).

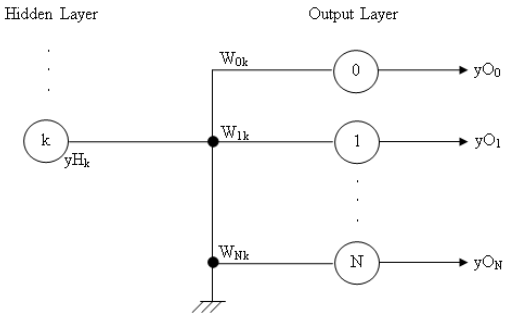

Fig. 1 A hidden neuron that has a short circuit at its output

In all cases the yH_k term appears multiplied with the corresponding weight, when the activation is computed. So, we have the following products:

$w_{0k}\ yH_k$, $w_{1k}\ yH_k$, $w_{2k}\ yH_k$, ,,,

Because $yH_k=0$, all these products are 0. The idea is that instead of thinking $yH_k=0$, it is equivalent with considering every weight of every connection that connects the hidden neuron k with all the output neurons is 0 ($yH_k=0$ is equivalent with: $w_{0k}=0$, $w_{1k}=0$, etc.). This result is important for our simulations.

In order to analyse the fault tolerance of the neural network that has p – faulty hidden neurons, we have to generate all the combinations of p faulty neurons from the hidden layer. For this, we used a a backtracking algorithm

3. THE C APPLICATION. SIMULATION RESULTS

We developed a C application that has two modules. The first is responsible for training the neural network and for saving zhe set of weights in a file. Running this module we can obtain more valid sets of weights that are saved in text files weights001.txt, weights002.txt, etc.

The second module of the application analyses the fault tolerance of the network, inducing a variable number p of faulty hidden neurons (p=1, p=2, p=3, etc).

The simulation results are presented for a MLP neural network that classifies 7x7 pixels images in two classes: horizontal line and vertical line. This network has 49 input neurons, 18 neurons in the hidden layer and 2 output neurons.

We used 8 training examples and other 8 different images for network testing.

Using the first module of the C application , we obtained 3 sets of weights (weights001.txt and weights002.txt and weights003.txt.), that were validated using the 8 test patterns. One set of weights is validated only if the network answers correctly to all 8 tests [5].

We shall compare the three sets of weights analyzing the fault tolerance of the network. We start making one faulty neuron in the hidden layer. Because the hidden layer has 18 neurons, we have 18 different cases with a single faulty neuron. For each case we apply all the 8 tests to the neural networks, and for all the three sets of weights we obtained the same results:

18 correct answers from 18

The neural network answered correctly to all the tests, even though one hidden neuron was faulty. That illustrates the intrinsic fault tolerance of neural networks.

Because we obtained the same results for all three files of weights , we continue to analyze the fault tolerance, generating all possible groups of two faulty hidden neurons (153 different groups). For NH hidden neurons we have NH*(NH-1)/2 different pairs of hidden neurons.

Also, in this case with the three sets of weights we obtained the same results:

153 correct answers from 153

We continue to analyze the fault tolerance, generating all possible groups of three faulty hidden neurons (816 different combinations).

Also in this case with the three sets of weights we obtained the same results:

816 correct answers from 816

So, we then generate all possible groups of four faulty hidden neurons (3060 different cases).

For the three sets of weights we obtained the following results:

For weights001.txt we have:

3056 correct answers from a total of 3060.

For weights002.txt we have:

3060 correct answers from a total of 3060.

For weights003.txt we have:

3060 correct answers from a total of 3060.

So, at this step we have two files of weights that obtained the same maximum score: 3060 from 3060 (weights002.txt and weights003.txt) .

With these two files of weights we continue to test them for any combination of five faulty hidden neurons. We obtained the following results:

For weights002.txt :

We obtained 8541 correct answers from 8568.

For weights003.txt:

We obtained 8551 correct answers from 8568.

Now we have different numbers of correct answers. Then we choose as the best weights file from the point of view of fault tolerance, the one that has the maximum fault tolerance in respect with the faults in hidden neurons, the file weights003.txt

4. CONCLUSIONS

In case of a hardware implementation, an increased fault tolerance of a neural network, is a real advantage. The application that we presented in this paper, helps the designer of the neural network to choose from a few weight sets, the one that assures the greatest fault tolerance for any p – combinations of faulty hidden neurons. We illustrated the proposed method using a C language simulation, for a MLP neural network that is used to classify an input image of 7x7 pixels in two output classes. We choose the best weights set from three different sets of weights.

5. REFERENCES

[1] Al. Ene, Gh. Serban and R. Beloiu, "*A software environment for the analysis of the fault tolerance of multi-layer perceptron neural-networks*" RSSE'98, Second International Conference on

renewable sources and environmental electrotechnologies, Oradea, pp. 94-98, 1998.

[2] G. Toderean, M. Costeiu, M. Giurgiu, *Retele neuronale artificiale*, Editura Albastra,1995.

[3] V. Catuneanu and A. Bacivarof, "*Structuri electronice de inalta fiabilitate. Toleranta la defectari*", Ed. Militara, Bucuresti, 1989.

[4] Al. Ene and C. Stirbu, "*Retele neuronale. Teorie si aplicatii in Java*", Ed. Universitatii din Pitesti, 2008.

[5] Al. Ene and C. Stirbu, "*Weights set selection method for feed forward neural networks*", ECAI 2013.