Analysis of Neural Network Back-Propagation Algorithm

ANEKE JUDE I. [1], EZECHUKWU O. A. [2], UWAECHI P. C. [3]
[1,2,3] *Nnamdi Azikiwe University, Awka, Nigeria*

Abstract -- Artificial neural networks simulate the neural systems behaviour by means of the interconnection of the basic processing units called neurons. The neurons can receive external signals or signals coming from the other neurons affected by a factor called weight. The output of neuron is the result of applying a specific function, known as transfer function, to the sum of its inputs plus threshold value called bias. This paper demonstrates the practical analysis of neural network back-propagation algorithm. It shows the mathematical process of how the neural network manages the data fed to it for it to be trained to recognize patterns, classify data and forecast future events. Feed forward networks have been employed along with back propagation algorithm for the pattern recognition process.

Indexed Terms: Feed forward back propagation algorithm, neural network, Neuron, Local gradient, Synaptic weight, Activation function

I. INTRODUCTION

A typical Back Propagation Neural Network is a non-linear regression technique which attempts to minimize the global error. Its training includes both forward and backward propagation, with the desired output used to generate the error values for back propagation to iteratively improve the output. The back propagation neural network must have at least one input layer and one output layer. The hidden layers are optional. A Typical back propagation neural network is shown in the Figure 1 [1]:

The Back propagation neural network consists of four layers: an input layer with two neurons, hidden layers with three and two neurons respectively and an output layer with one neuron. In the Figure 1, we see that the output of a neuron in a layer goes to all neurons in the following layer and each neuron has its own weights. Initially the weights of the input layer are assumed to be 1 for each input. The output of the back propagation neural network is reached by applying input values to the input layer [2], passing the output of each neuron to the following layer as input.

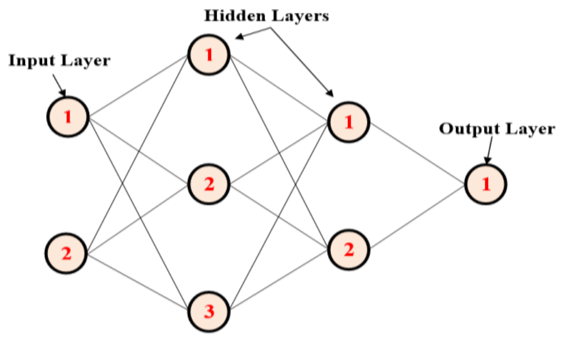

Fig. 1: Back propagation neural network

The number of neurons in the input layer depends on the number of possible inputs we have, while the number of neurons in the output layer depends on the number of desired outputs. The number of hidden layers and how many neurons in each hidden layer cannot be well defined in advance, and could change per network configuration and type of data. In general, the addition of a hidden layer could allow the network to learn more complex patterns, but at the same time decreases its performance [3]. Ideally, we could start a network configuration using a single hidden layer, and add more hidden layers if we notice that the network is not learning as well as we like.

The Back-propagation training algorithm could be summarized as follows: The Input data sample is first presented to the network and then the network's output taken from the output layer is compared with the desired output and the error is calculated in each output neuron. And now for each neuron, a scaling factor called the local error is calculated which indicates how much higher or lower the output must be adjusted to match the desired output. The weights are modified to lower this local error. This process gets repeated until the error falls within the acceptable value (pre-defined threshold) which would indicate that the neural network has been trained successfully. On the other side, if the maximum number of iterations is reached, then it indicates that the training was not successful [4].

II. RESEARCH METHODOLOGY

2.1 Analysis of Back-Propagation Algorithm:

The basic concept behind the successful application [4] of neural networks in any field is to determine the weights to achieve the desired target and this process is called learning or training. The network weights are modified with the prime objective of minimization of the error between a given set of inputs and their corresponding target values. However, in this work, the back-propagation algorithm is employed in training all the neural networks [5]. The steps below are mathematical calculation and illustration of network back-propagation algorithm. For this purpose, a NN architecture of (2-2-2) has been adopted as shown in Figure 2.

Step 1:

Calculate the local gradients $(\delta o1, \delta 02, \delta h1\ and\ \delta h2)$ for the four nodes n1, n2, n3 and n4 in the network of Figure 2.

$\delta o1$ is the local gradient of the first output neuron
$\delta o2$ is the local gradient of the second output neuron
$\delta h1$ is the local gradient of the first neuron in the hidden layer
$\delta h2$ is the local gradient of the second neuron in the hidden layer

The neuron activation function used is sigmoid activation function as written in (1)

$$f(v) = \frac{1}{1+\exp(-v)} \qquad (1)$$

And its derivative is

$$f'(v) = f(v)[1-f(v)] \qquad (2)$$

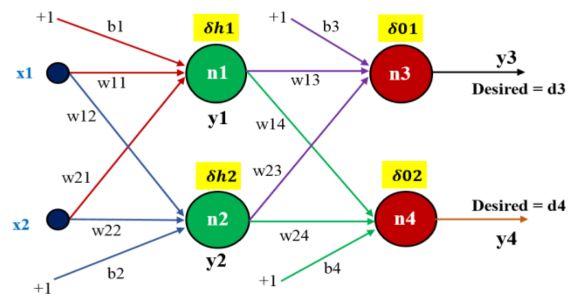

Fig. 2: Mathematical Illustration of Back-Propagation Algorithm

First the local gradients of the two output neurons in Figure 2 are determined as

$$\delta o1 = f'(1 * b3 + y1 * w31 * y2 * w32) * (d3 - y3) \quad (3)$$

$$\delta o2 = f'(1 * b4 + y1 * w41 * y2 * w42) * (d4 - y4) \quad (4)$$

Next the local gradients of the two hidden neurons are determined as

$$\delta h1 = f'(1 * b1 + x1 * w11 * x2 * w12) * (\delta o1 * w31 + \delta o2 * w41) \qquad (5)$$

$$\delta h2 = f'(1 * b2 + x1 * w21 * x2 * w22) * (\delta o1 * w32 + \delta o2 * w42) \qquad (6)$$

Where, w is the weight, b is the bias weight, y is the actual neuron output, and d is the desired (target) output, see Figure 2.

Step 2:
Adjust the weights of the network using the general learning rule:

$$w(n+1) = w(n) + \alpha * w(n-1) + \eta * \delta(n) * y \quad (7)$$

Where, $w(n+1)$ is the new weight, $w(n)$ is the current weight, $w(n-1)$ is the previous weight, α is the mobility factor, η is the learning rate or the training parameter, $\delta(n)$ is the current local gradient.

Applying (7) to Figure 2, the new weights read:

$$w13(n+1) = w13(n) + \alpha * w13(n-1) + \eta * \delta o1(n) * y1 \qquad (8)$$

$$w14(n+1) = w14(n) + \alpha * w14(n-1) + \eta * \delta o2(n) * y1 \qquad (9)$$

$$w23(n+1) = w23(n) + \alpha * w23(n-1) + \eta * \delta o1(n) * y2 \qquad (10)$$

$$w24(n+1) = w24(n) + \alpha * w24(n-1) + \eta * \delta o2(n) * y2 \qquad (11)$$

$$w11(n+1) = w11(n) + \alpha * w11(n-1) + \eta * \delta h1(n) * x1 \qquad (12)$$

$$w12(n+1) = w12(n) + \alpha * w12(n-1) + \eta * \delta h2(n) * x1 \qquad (13)$$

$$w21(n+1) = w21(n) + \alpha * w21(n-1) + \eta * \delta h1(n) * x2 \qquad (14)$$

$$w22(n+1) = w22(n) + \alpha * w22(n-1) + \eta * \delta h2(n) * x2 \qquad (15)$$

$$b3(n+1) = b3(n) + \alpha * b3(n-1) + \eta * \delta o1(n) * 1 \qquad (16)$$

$$b4(n+1) = b4(n) + \alpha * b4(n-1) + \eta * \delta o2(n) * 1 \quad (17)$$

$$b1(n+1) = b1(n) + \alpha * b1(n-1) + \eta * \delta h1(n) * 1 \quad (18)$$

$$b2(n+1) = b2(n) + \alpha * b2(n-1) + \eta * \delta h2(n) * 1 \quad (19)$$

III. RESULTS AND DISCUSSION

With application of specific values to step 1 and 2, a complete forward and backward sweep of the feed forward network (2-2-1 architecture) is performed as shown below using the back-propagation algorithm as aforementioned. In this case neural network architecture of (2-2-1) is used as shown in Figure 3. The values of the initial weights are chosen arbitrary.

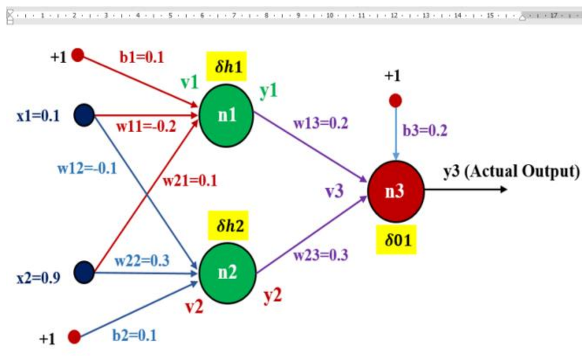

Fig. 3: A 2.2.1 NN Architecture with the corresponding weights shown

Assumptions:

Let the target output (d) = 0.9 and actual output (y) is unknown,

Learning rate, $\eta = 0.25$,

Mobility Factor, $\alpha = 0.0001$

The Forward Pass:

$$v1 = 1 * b1 + x1 * w11 + x2 * w12$$

$$= 1 * 0.1 + 0.1 * (-0.2) + 0.9 * 0.1 = 0.17$$

$$y1 = f(v1) = f(0.17) = \frac{1}{1+\exp(-0.17)} = 0.542$$

$$v2 = 1 * b2 + x1 * w21 + x2 * w22 \quad = 1 * 0.1 + 0.1 * (-0.1) + 0.9 * 0.3 = 0.36$$

$$y2 = f(v2) = f(0.36) = \frac{1}{1+\exp(-0.36)} = 0.589$$

$$v3 = 1 * b3 + y1 * w31 + y2 * w32 = 1 * 0.2 + 0.542 * (0.2) + 0.589 * 0.3 = 0.485$$

$$y3 = f(v3) = f(0.485) = \frac{1}{1+\exp(-0.485)} = 0.619$$

Please note the activation function, $f(v)$ used in the forward pass (feed-forward) and not its derivative $f^{'}(v)$ which is used during the backward pass (back-propagation).

Now,

The error(e) = target output $(d3)$ − actual output $(y3) = 0.9 - 0.619 = 0.281$

Therefore, after the forward pass, there is an error of 0.281 which means the back-propagation is required to adjust the weights in other to get the weights that will reduce the error to the global minimum value. The idea is that the actual output should be equal to the target output.

The Backward Pass:

Here, we need to go backward to find out the new weights of the network. This is achieved by applying back-propagation algorithm. First the local gradients of the neurons (nodes) are calculated as was done before but in this case, there are specific values. The local gradients of the three neurons are as follows starting with that of the output neuron.

$$\begin{aligned}\delta o1 = f'(v3) * (d3 - y3) &= f'(0.4851) * 0.281 \\ &= f(0.4851)[1 - f(0.4851)] * 0.281 \\ &= 0.619[1 - 0.619] * 0.281 = 0.0663\end{aligned}$$

$$\begin{aligned}\delta h1 = f'(v1) * (\delta o1 * w31) &= f'(0.17) * (0.0663 * 0.2) \\ &= f(0.17)[1 - f(0.17)] * 0.01362 \\ &= 0.542[1 - 0.542] * 0.0136 \\ &= 0.0033\end{aligned}$$

$$\begin{aligned}\delta h2 = f'(v2) * (\delta o1 * w32) &= f'(0.36) * (0.0663 * 0.3) \\ &= f(0.36)[1 - f(0.36)] * 0.01989 \\ &= 0.589[1 - 0.589] * 0.0198 \\ &= 0.0049\end{aligned}$$

Next is to adjust the weights of the network using the learning rule general expression of (7):

$$w(n+1) = w(n) + \alpha * w(n-1) + \eta * \delta(n) * y$$

$$w31(n+1) = w31(n) + \alpha * w31(n-1) + \eta * \delta o1(n) * y1$$

$$w31(n+1) = 0.2 + 0.0001 * 0.2 + 0.25 * 0.0663 * 0.542 = 0.2090$$

$$w32(n+1) = w32(n) + \alpha * w32(n-1) + \eta * \delta o1(n) * y2$$

$$w32(n+1) = 0.3 + 0.0001 * 0.3 + 0.25 * 0.0663 * 0.589 = 0.3098$$

$$w11(n+1) = w11(n) + \alpha * w11(n-1) + \eta * \delta h1(n) * x1$$

$$w11(n+1) = (-0.2) + 0.0001 * (-0.2) + 0.25 * 0.0033 * 0.1 = -0.1999$$

$$w21(n+1) = w21(n) + \alpha * w21(n-1) + \eta * \delta h2(n) * x1$$

$$w21(n+1) = (-0.1) + 0.0001 * (-0.1) + 0.25 * 0.0049 * 0.1 = -0.0999$$

$$w12(n+1) = w12(n) + \alpha * w12(n-1) + \eta * \delta h1(n) * x2$$

$$w12(n+1) = (0.1) + 0.0001 * (0.1) + 0.25 * 0.0033 * 0.9 = 0.1008$$

$$w22(n+1) = w22(n) + \alpha * w22(n-1) + \eta * \delta h2(n) * x2$$

$$w22(n+1) = 0.3 + 0.0001 * 0.3 + 0.25 * 0.0049 * 0.9 = 0.3011$$

$$b3(n+1) = b3(n) + \alpha * b3(n-1) + \eta * \delta o1(n) * 1$$

$$b3(n+1) = 0.2 + 0.0001 * 0.2 + 0.25 * 0.0663 * 1 = 0.2166$$

$$b1(n+1) = b1(n) + \alpha * b1(n-1) + \eta * \delta h1(n) * 1$$

$$b1(n+1) = 0.1 + 0.0001 * 0.2 + 0.25 * 0.0033 * 1 = 0.1008$$

$$b2(n+1) = b2(n) + \alpha * b2(n-1) + \eta * \delta h2(n) * 1$$

$$b2(n+1) = 0.1 + 0.0001 * 0.2 + 0.25 * 0.0049 * 1 = 0.1012$$

Now these new weights as calculated above and shown in Figure 4 are used to perform another (second) forward pass.

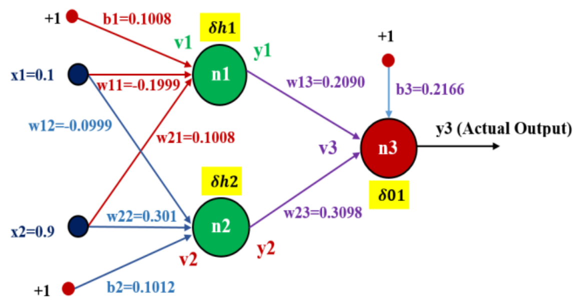

Fig. 4: New weights obtained after the first backward pass (back-propagation)

Then after one complete forward and backward pass we have new inputs and output. The results are compared with the old inputs and output as shown in Table 1.

Table 1: Comparison of the results obtained after one complete forward and backward pass

S/N	Component	Old	New (After One Iteration)
1	$v1$	0.17	0.1715
2	$y1$	0.542	0.5428
3	$v2$	0.36	0.3622
4	$y2$	0.589	0.5896
5	$v3$	0.4851	0.5127
6	$y3$	0.619	0.6254
	$e = d3 - y3$	$0.9 - 0.619 = 0.281$	$0.9 - 0.6254 = 0.2746$

From the Table 1, the error reduced after the first forward and backward pass from $0.281\ to\ 0.2746$. The forward and backward passes continue until the error becomes almost zero. The results obtained after a few more complete forward and backward passes are as shown below.

After the second pass (iteration) $e =$ 0.2683

After the third pass (iteration) $e =$ 0.2623

After the fourth pass (iteration) $e =$ 0.2565

After 100 passes (iteration) $e =$ 0.0693

After 200 passes (iteration) $e =$ 0.0319

After 500 passes (iteration) $e =$ 0.0038

Error is getting reduced after each pass until it converges (target and actual output equal each other). So, this is how neural network is trained using back-propagation algorithm.

IV. CONCLUSIONS

Effort has been made to demonstrate how the Back-propagation training algorithm could be achieved. It has been shown that neural network synaptic weights can be updated iteratively. The Input data sample was first presented to the network and then the network's output taken from the output layer was compared with the desired output and the error was calculated in each output neuron. And now for each neuron, a scaling factor called the local error was calculated which indicated how much higher or lower the output must be adjusted to match the desired output. The weights were modified to lower this local error. This process was repeated until the error falls within the acceptable value (pre-defined threshold) which would indicate that the neural network has been trained successfully. On the other side, if the maximum number of iterations is reached, then it indicates that the training was not successful.

REFERENCES

[1] Anderson, J. A. *An Introduction to Neural Networks.* Prentice Hall, 2003.

[2] Gaudrat, J., Giusiano B., and Huiart. *Comparison of the performance of multi-layer perceptron and linear regression for epidemiological data*. Computer Statist. & Data Anal., 44, 547-70, 2004.

[3] Howard Demuth, Mark Beale, Martin Hagan. *The MathWorks.*

[4] Lukowicz M., Rosolowski E. (2013). Artificial neural network based dynamic compensation of current transformer errors. Proceedings of the 8th International Symposium on Short-Circuit Currents in Power Systems, Brussels, pp. 19-24.

[5] R.P.Hasbe, A.P.Vaidya, Detection and classification of faults on 220 KV transmission line using wavelet transform and neural network, International Journal of Smart Grid and Clean Energy, August 2013.