Multilayer Backpropagation Neural Networks for Implementation of Logic Gates

Santosh Giri and Basanta Joshi

Department of Electronics & Computer Engineering, Institute of Engineering, Pulchowk Campus, Tribhuvan University.
Lalitpur, Nepal

Abstract. ANN is a computational model that is composed of several processing elements (neurons) that tries to solve a specific problem. Like the human brain, it provides the ability to learn from experiences without being explicitly programmed. This article is based on the implementation of artificial neural networks for logic gates. At first, the 3 layers Artificial Neural Network is designed with 2 input neurons, 2 hidden neurons & 1 output neuron. after that model is trained by using a backpropagation algorithm until the model satisfies the predefined error criteria (e) which set 0.01 in this experiment. The learning rate (α) used for this experiment was 0.01. The NN model produces correct output at iteration (p)= 20000 for AND, NAND & NOR gate. For OR & XOR the correct output is predicted at iteration (p)=15000 & 80000 respectively.

Keywords: Machine Learning, Artificial Neural Network, Back propagation, Logic Gates.

1 Introduction

Machine Learning algorithms automatically build a mathematical model using sample data also known as 'training data' to make decisions without being specifically programmed to make those decisions.Machine learning is applicable to fields such as Speech recognition[7], text prediction[1], handwritten generation[8], genetic algorithms[4], artificial neural networks, natural language processing[6].

1.1 Artificial Neural Network

Artificial neural networks are systems motivated by the distributed, massively parallel computation in the brain that enables it to be so successful at complex control and classification tasks. The biological neural network that accomplishes this can be mathematically modeled by a weighted, directed graph of highly interconnected nodes (neurons)[3]. An ANN initially goes through a training phase where it learns to recognize patterns in data, whether visually, aurally, or textually. During this training phase, the network compares its actual output produced with what it was meant to produce the desired output. The difference between both outcomes is adjusted using a set of learning rules called back propagation[5]. This means the network works backward, going from the output unit to the input units to adjust

DOI: 10.5121/ijcses.2021.12101

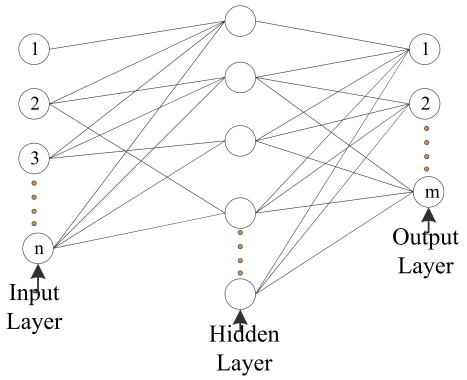


Fig. 1.1: Architecture of typical ANN

the weight of its connections between the units until the difference between the actual and desired outcome produces the lowest possible error. The structure of a typical Artificial Neural Network is given in Fig. 1.1.

Neuron An ANN has hundreds or thousands of processing units called neurons,which are analogous to biological neurons in human brain. neurons interconnected by nodes. These processing units are made up of input and output units. The input units receive various forms and structures of information based on an internal weighting system, and the neural network attempts to learn about the information presented to it[2]. The neuron multiplies each input $(x_1, x_2, ..., x_n)$

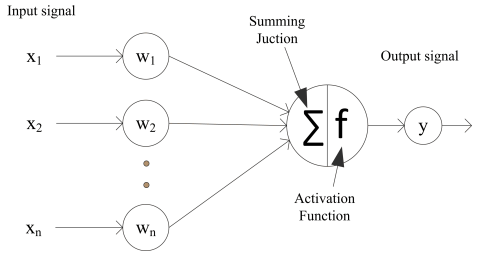


Fig. 1.2: Architecture of single neuron of an ANN

by the associated weight (w ,w ,.....,w) and then sums up all the results i.e. computes a weighted sum of the input signals as:

$$X = \sum_{i=1}^{n} x_i w_i \tag{1.1}$$

Then, the result is then passed through a non-linear function(f) called activation function. An activation function is the function that describes the output behavior of a neuron[9], and has a threshold value $'\theta'$. Then the result is compared with a threshold value which gives the output as either '0' or '1'. If the weighted sum is less than $'\theta'$ then neuron output is '1' otherwise 0. In general, the neuron uses *step* function (1.2) as activation functions.

$$Y = \begin{cases} +1, if X \geqslant \theta \\ 0, if X < \theta \end{cases} \tag{1.2}$$

1.2 Back-propagation multilayer neural networks

In the backpropagation neural network, input signals are propagated on a layer-by-layer basis from input to output layer for forward propagation and output to input for backward propagation. The hidden layer is stacked in between inputs and outputs, which allows neural networks to learn more complicated features. The learning in multi-layer neural networks processed the same way as in perceptron. At first, a training set of input patterns is presented to the network and then the network computes its output, if there is a difference between the desired output and actual output (an error) then the weights are adjusted to reduce the difference.

1.3 Backpropagation Algorithms

To train multi-layer Artificial Neural Network designed here, the backpropagation algorithm[5] is used. Backpropagation is the set of learning rules used to guide artificial neural networks. The working mechanism of this algorithm is as follows:

Step 1: Initialization:
Initialization of initial parameters such as weights (W), threshold (θ), learning rate (α) & bias value (b) and number of iterations (p). Initialize all the initial parameters inside a small range.

Step 2: Activations:
Activate the networks by applying all sets of possible inputs $x_1(p)$, $x_2(p)$, $x_3(p), ...x_n(p)$ & desired outputs $y_{d1}(p), y_{d2}(p), y_{d3}(p), ..., y_{dn}(p)$.
i. Calculate the actual outputs in hidden layer as:

$$y_j(p) = sigmoid[\sum_{i=1}^{n} x_i(p) * w_{ij}(p) - \theta_j] \tag{1.3}$$

where,
n = number of inputs of neuron j in hidden layer and Activation function used here is *sigmoid* function defined as:

$$Y^{sigmoid} = \frac{1}{1 + e^{-X}} \tag{1.4}$$

ii. Calculate theactual outputin outputlayer as:

$$y_k(p) = sigmoid[\sum_{j=1}^{m} x_j(p) * w_{jk}(p) - _k] \tag{1.5}$$

Where,
m = number of inputs of neuron k in output layer.

Step 3: Error gradient and weights:
Calculate the error gradient and update the weights by propagating backward through the networks.
i. Calculate the error gradient δ for neurons in output layer:

$$\delta_k(p) = y_k(p) * [1 - y_k(p)] * e_k(p) \tag{1.6}$$

$$\text{Where, } e_k(p) = y_{d,k}(p) - y_k(p)$$

ii. Calculate the weight corrections as:

$$\bigtriangledown w_{jk}(p) = \alpha * y_j(p) * \delta_k(p) \tag{1.7}$$

iii. Update the weights of neurons at output layer as:

$$w_{jk}(p+1) = w_{jk}(p) + \bigtriangledown w_{jk}(p) \tag{1.8}$$

iv. Calculate the error gradient δ for neurons in hidden layer:

$$\delta_j(p) = y_j(p) * [1 - y_j(p)] * \sum_{k=1}^{l} \delta_k(p) * w_{jk}(p) \tag{1.9}$$

v. Calculate the weight corrections as:

$$\bigtriangledown w_{ij}(p) = \alpha * x_i(p) * \delta_j(p) \tag{1.10}$$

vi. Update the weights of neurons at hidden layer as:

$$w_{ij}(p+1) = w_{ij}(p) + \bigtriangledown w_{ij}(p) \tag{1.11}$$

Step 4:
Increase the value of p(iteration) by 1, go to Step 2 and repeat this process until predefined error criteria is fulfilled.

2 Methodology

2.1 Designing a Backpropagation multilayer neural network

In this article, we have used a multilayer neural network with two input neurons, two hidden neurons, and one output neurons as shown in Fig. 2.1. We are doing so because all the logic gates: AND, OR, NAND, NOR, and XOR being implemented here have two inputs signals and one output signals with signal values being either '1' or '0'. Here W_13, W_{14}, W_{23}, W_{24} are weights associated between neurons of input layer & hidden layer and W_{35}, W_{45} are weights associated between hidden layer and output layer. biases: b_3, b_4 & b_5 are values associated with each node in the intermediate (hidden) and output layers of the network, are treated in the same manner as other weights.

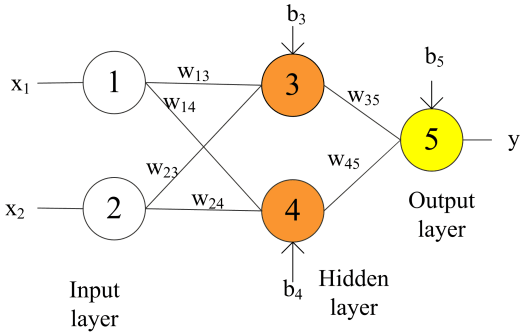


Fig. 2.1: Multi-layer back-propagation neural networks with two input neurons and single output neuron

2.2 Implementation of multi-layer neural networks.

To train the multi-layer neural network designed here we have used the back-propagation algorithm[5] describes in section(1.3). The training of the NN model takes place in two steps. In the first step of training, input signals were presented to the input layer of the network. Then network propagates the signals from layer to layer until the output is generated by the output layer. If the output is different from the desired output then an error is calculated and propagated backward from the output layer to the input layer and the weights are modified accordingly. This process is repeated until a predefined criteria (sum of squared error)is fulfilled.

3 Results and Discussions

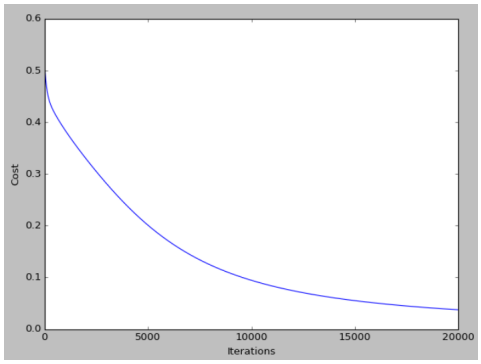
In this experiment we have set error criteria ='0.01'. Initial weights and bias values of the networks are set randomly in between small range as described in section **1.3**. The cost function or learning curve and final results for all the logic gates are presented in respective figures and tables. Among other hyper-parameters, the learning rate(α) to train the NN model is set $'0.01'$ for all logic gates. The training of the NN model was started from the iteration(p)= 5000 & final output is observed to satisfy the error criteria. If the NN model doesn't satisfy the error (predict the correct output), then the value of p is incremented by 5000 on the next run. This process is continued until the Network predicts the correct output. From the figures and tables in the **section 3**. we can conclude that given the random bias and weights values inside an small range, OR gate satisfies the threshold (error criteria=0.01) at '20000' epoch, i.e. at '20000' epoch the sum of square error (0.008) is less than threshold (0.01). similarly the model satisfies the error threshold value (0.01) for OR gate at '15000' epoch, with sum of squared error equals to '0.005', for NAND gate at '20000' epoch with sum of squared error equals to '0.01', for NOR gate at '20000' epoch with sum of squared error equals to '0.008' and for XOR gate at '80000' epoch with sum of squared error equals to '0.003'. All the other experimental results are given in more details in the respective tables.

Table 3.1: Final result(AND gate) at iteration(p)=5000

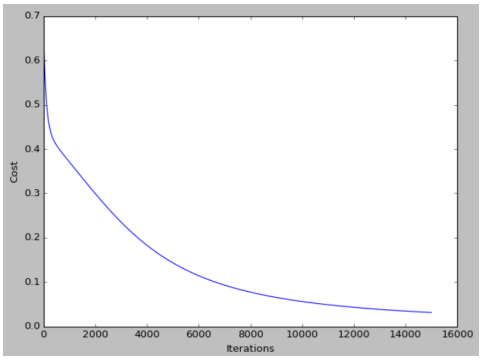
Inputs		Desired Output	Actual Output	Error	Sum of squared error
0	1	*0*	0.1395	-0.1395	0.17521154
1	0	*0*	0.1655	-0.1655	
1	1	*1*	0.643	0.357	
0	0	*0*	0.0302	-0.0302	

Table 3.2: Final result(AND gate) at iteration(p)=10000

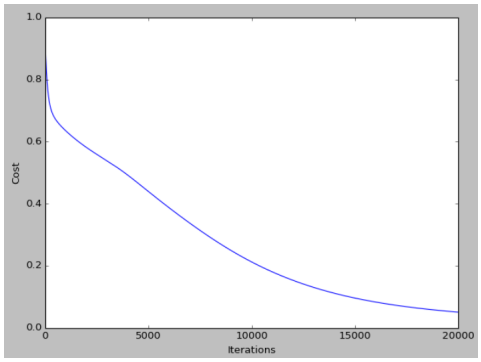
Inputs		Desired Output	Actual Output	Error	Sum of squared error
0	1	*0*	0.0637	-0.064	0.04672867
1	0	*0*	0.0956	-0.096	
1	1	*1*	0.8171	0.1829	
0	0	*0*	0.0089	-0.009	



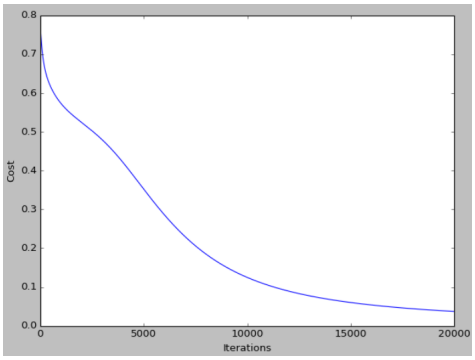
(a) Final learning curve(AND gate) at iteration (p)=20000



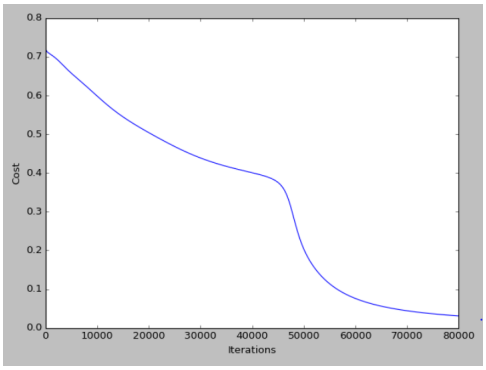
(b) Final learning curve(OR gate) at iteration (p)=15000



(c) Final learning curve(NAND gate) at iteration (p)=20000



(d) Final learning curve(NOR gate) at iteration (p)=20000



(e) Final learning curve(XOR gate) at iteration (p)=80000

Fig. 3.1: The Final learning curve for AND, OR, NAND, NOR and Exclusive OR(XOR) gates at which the model satisfies the error criteria (0.01) is shown in above figures (a), (b),(c),(d),(e) respectively.

Table 3.3: Final result(AND gate) at iteration(p)=15000

Inputs		Desired Output	Actual Output	Error	Sum of squared error
0	1	*0*	0.0353	-0.0353	0.01718587
1	0	*0*	0.0613	-0.0613	
1	1	*1*	0.8897	0.1103	
0	0	*0*	0.004	-0.004	

Table 3.4: Final result(AND gate) at iteration(p)=20000

Inputs		Desired Output	Actual Output	Error	Sum of squared error
0	1	*0*	0.0228	-0.023	0.00823433
1	0	*0*	0.0438	-0.044	
1	1	*1*	0.9239	0.0761	
0	0	*0*	0.0022	-0.002	

Table 3.5: Final result(OR gate) at iteration(p)=5000

Inputs		Desired Output	Actual Output	Error	Sum of squared error
0	1	*1*	0.8703	0.1297	0.09218
1	0	*1*	0.898	0.102	
1	1	*1*	0.9639	0.0361	
0	0	*0*	0.2523	-0.2523	

Table 3.6: Final result(OR gate) at iteration(p)=10000

Inputs		Desired Output	Actual Output	Error	Sum of squared error
0	1	*1*	0.9444	0.0556	0.01604
1	0	*1*	0.9558	0.0442	
1	1	*1*	0.9876	0.0124	
0	0	*0*	0.1041	-0.1041	

Table 3.7: Final result(OR gate) at iteration(p)=15000

Inputs		Desired Output	Actual Output	Error	Sum of squared error
0	1	*1*	0.9686	0.0314	0.00518
1	0	*1*	0.9742	0.0258	
1	1	*1*	0.9934	0.0066	
0	0	*0*	0.059	-0.059	

Table 3.8: Final result(NAND gate) at iteration(p)=5000

Inputs		Desired Output	Actual Output	Error	Sum of squared error
0	1	*1*	0.7141	0.2859	0.55312
1	0	*1*	0.7746	0.2254	
1	1	*0*	0.6286	-0.6286	
0	0	*1*	0.8405	0.1595	

Table 3.9: Final result(NAND gate) at iteration(p)=10000

Inputs		Desired Output	Actual Output	Error	Sum of squared error
0	1	*1*	0.8134	0.1866	0.19074
1	0	*1*	0.8446	0.1554	
1	1	*0*	0.3623	-0.3623	
0	0	*1*	0.9774	0.0226	

Table 3.10: Final result(NAND gate) at iteration(p)=15000

Inputs		Desired Output	Actual Output	Error	Sum of squared error
0	1	*1*	0.9139	0.0861	0.04745
1	0	*1*	0.9157	0.0843	
1	1	*0*	0.1814	-0.1814	
0	0	*1*	0.9954	0.0046	

Table 3.11: Final result(NAND gate) at iteration(p)=20000

Inputs		Desired Output	Actual Output	Error	Sum of squared error
0	1	*1*	0.9539	0.0461	0.0142
1	0	*1*	0.9538	0.0462	
1	1	*0*	0.0997	-0.0997	
0	0	*1*	0.9981	0.0019	

Table 3.12: Final result(NOR gate) at iteration(p)=5000

Inputs		Desired Output	Actual Output	Error	Sum of squared error
0	1	*0*	0.1861	-0.1861	0.41447
1	0	*0*	0.2115	-0.2115	
1	1	*0*	0.1233	-0.1233	
0	0	*1*	0.4344	0.5656	

Table 3.13: Final result(NOR gate) at iteration(p)=15000

Inputs		Desired Output	Actual Output	Error	Sum of squared error
0	1	*0*	0.0332	-0.0332	0.02021
1	0	*0*	0.0604	-0.0604	
1	1	*0*	0.0114	-0.0114	
0	0	*1*	0.8762	0.1238	

Table 3.14: Final result(NOR gate) at iteration(p)=20000

Inputs		Desired Output	Actual Output	Error	Sum of squared error
0	1	*0*	0.0197	-0.0197	0.00805
1	0	*0*	0.0398	-0.0398	
1	1	*0*	0.006	-0.006	
0	0	*1*	0.9223	0.0777	

Table 3.15: Final result(XOR gate) at iteration(p)=5000

Inputs		Desired Output	Actual Output	Error	Sum of squared error
0	1	*1*	0.5197	0.4803	0.93131
1	0	*1*	0.5259	0.4741	
1	1	*0*	0.5905	-0.5905	
0	0	*0*	0.3566	-0.3566	

Table 3.16: Final result(XOR gate) at iteration(p)=10000

Inputs		Desired Output	Actual Output	Error	Sum of squared error
0	1	*1*	0.5617	0.4383	0.82334
1	0	*1*	0.5485	0.4515	
1	1	*0*	0.5986	-0.5986	
0	0	*0*	0.2628	-0.2628	

Table 3.17: Final result(XOR gate) at iteration(p)=15000

Inputs		Desired Output	Actual Output	Error	Sum of squared error
0	1	*1*	0.6164	0.3836	0.73789
1	0	*1*	0.5717	0.4283	
1	1	*0*	0.6173	-0.6173	
0	0	*0*	0.162	-0.162	

Table 3.18: Final result(XOR gate) at iteration(p)=80000

Inputs		Desired Output	Actual Output	Error	Sum of squared error
0	1	*1*	0.9666	0.0334	0.00379
1	0	*1*	0.9677	0.0323	
1	1	*0*	0.0352	-0.0352	
0	0	*0*	0.0199	-0.0199	

Table 3.19: Weight summary (Initial and Final)

Weights	Initial	Final				
		AND p=20000	OR p=15000	NAND p=20000	NOR p=20000	XOR p=80000
W_{12}	1.62	3.1	2.67	-1.78	0.03	4.11
W_{14}	-0.61	-0.22	1.83	-2.9	-2.34	4.07
W_{23}	-0.52	-2.74	-3.3	2.84	4.24	-6.91
W_{24}	-1.07	-3.71	-3.7	1.76	3.6	-6.34
W_{35}	0.86	3.34	3.87	5.27	2.34	-8.87
W_{45}	-2.3	-7.75	-6.25	-5.42	-7.21	-9.03
b_3	0.41	-0.5	-0.81	3.01	1.2	-6.3
b_4	0.68	4.39	1.62	-2.94	-2.03	2.38
b_5	0.2	0.32	1.26	1.52	1.5	4.39

4 Conclusions

In this article, the Multi-layer artificial neural network for logic gates is implemented successfully by using the Backpropagation algorithm. The NN model implemented here satisfies error criteria (0.01) at learning rate(α=0.01), iterations(p)=20000 for logic gates: AND, NAND, NOR and for OR & XOR gate predicted correct output at p=15000 & p=80000 respectively.

References

[1] Adam Coates et al. "Text detection and character recognition in scene images with unsupervised feature learning". In: *Document Analysis and Recognition (ICDAR), 2011 International Conference on*. IEEE. 2011, pp. 440–445.

[2] JAKE FRANKENFIELD. *Artificial Neural Network*. 2020. URL: `https://www.investopedia.com/terms/a/artificial-neural-networks-ann.asp` (visited on 07/07/2020).

[3] Mohamad H Hassoun et al. *Fundamentals of artificial neural networks*. MIT press, 1995.

[4] John H Holland. "Genetic algorithms". In: *Scientific american* 267.1 (1992), pp. 66–73.

[5] Yann LeCun et al. "Backpropagation applied to handwritten zip code recognition". In: *Neural computation* 1.4 (1989), pp. 541–551.

[6] Christopher Manning and Hinrich Schutze. *Foundations of statistical natural language processing*. MIT press, 1999.

[7] Dabbala Rajagopal Reddy. "Speech recognition by machine: A review". In: *Proceedings of the IEEE* 64.4 (1976), pp. 501–531.

[8] Tamás Varga, Daniel Kilchhofer, and Horst Bunke. "Template-based synthetic handwriting generation for the training of recognition systems". In: *Proceedings of the 12th Conference of the International Graphonomics Society*. 2005, pp. 206–211.

[9] Bill Wilson. *The Machine Learning Dictionary*. 2020. URL: `http://www.cse.unsw.edu.au/billw/mldict.html/activnfn` (visited on 07/01/2020).