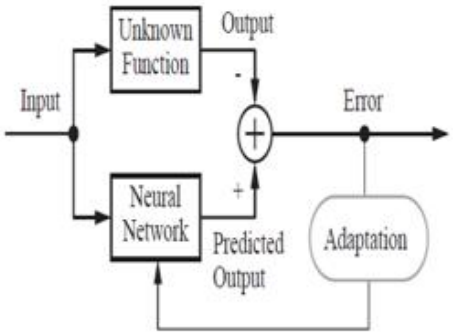
Artificial Neural Network Application in Logic System

SiddharthSaxena, JigarSingh,MaharanapratapSingh,Divya Sharma
Thakur College of Engineering and Technology
Thakur Village, Kandivali (East), Mumbai 400 101

ABSTRACT
The purpose of this paper is to provide a quick overview of neural networks and to explain how they can be used in controlsystems. We introduce the multilayer perceptron neural network and describe how it can be used for function approximation.The backpropagation algorithm (including its variations) is the principal procedure for training multilayer perceptrons; it is briefly described here. Care must be taken, when training perceptron networks, to ensure that they do not overfit the training data and then fail to generalize well in new situations.We have implemented the XOR logic using neural network tool in MATLAB and defined its use in control system.How system error can be used as feedback to the training network is also demonstrated in this paper.The paper gives a brief introspect into the neural network implimentation of control systems which in our case is the XOR network with its standard set of inputs and the respective standard set of outputs.

Categories and Subject Descriptors
Intelligent Systems
General Terms
Control system, Network architecture,Transfer function
Keywords
Perceptron, Backpropagation algorithm,Neuron model,XOR logic,Network Data Manager

1. INTRODUCTION
In this paper we want to give a brief introduction to neural networks and their application in control systems. The paper is written for readers who are not familiar with neural networks but are curious about how they can be applied to practical control problems. The field of neural networks covers a very broad area.
It is not possible in this paper to discuss all types of neural networks. Instead, we will concentrate on the most common neural network architecture–the multilayerperceptron[2].We will describe the basics of this architecture, discuss its capabilities and show how it has been used in several different control system configurations.For the purposes of this paper we will look at neural networks as function approximators.



Neural Network as Function Approximator

As shown in Figure 1, some unknown function that we wish to approximate.We want to adjust the parameters of the network so that it will produce the same response as the unknown function, if the same input is applied toboth systems.For our applications, the unknown function may correspond to a system we are trying to control, in which case the neural networkwill be the identified plant model. The unknown function could also represent the inverse of a system we are trying tocontrol, in which case the neural network can be used to implement the controller. At the end of this paper we will present severalcontrol architectures demonstrating a variety of uses for function approximator neural networks. Fig 1 Neural Network as Function Approximator.We will describe the basics of this architecture, discuss its capabilities and show how it has been used in several different control system configurations.For the purpose of this paper we will look at neural networks as function approximators.

2. MULTILAYER PERCEPTRON ARCHITECTURE
2.1 Neuron Model
The multilayer perceptron neural network is built up of simple components. We will begin with a single-input neuron, whichwe will then extend to multiple inputs.We will next stack these neurons together to produce layers. Finally, we will cascade thelayers together to form the network.
A single-input neuron is shown in Figure 2. The scalar input is

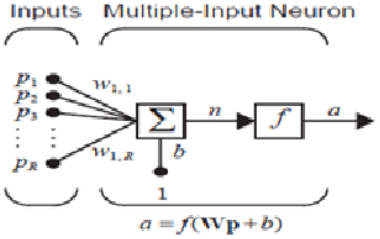


Fig 2 : Single Input Neuron

multiplied by the scalar *weight* to form one of the terms that is Sent to the summer and the other input is biased and then passed to the summer. The summer output often referred to as the *net input*, goes into a *transfer function* which produces the scalar neuron output a.

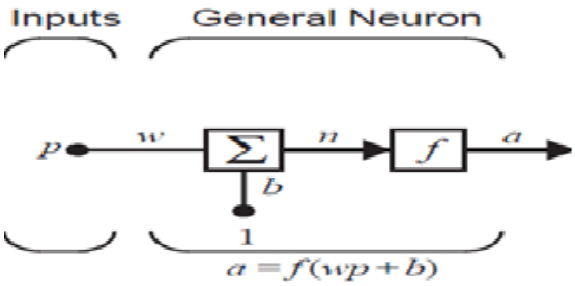


Fig4:Multiple Input Neuron

The neuron output is calculated as

$$a = f(wp + b) .$$

Note that and are both adjustablescalar parameters of the neuron. Typically the transfer function is chosen by the designer and then the parameters are adjusted by some learning rule so that the neuron input/output relationship meets some specific goal.The transfer function in Fig 2 may be a linear or a nonlinear function of n . One of the most commonly used functions is the log-sigmoid transfer function, which is shown in Figure below.

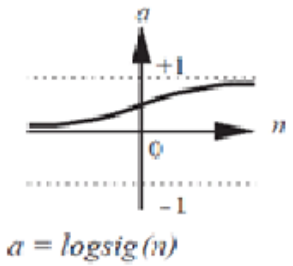


Fig 3: Log Sigmoid Transfer Function

This transfer function takes the input (which may have any value between plus and minus infinity) and squashes the output intothe range 0 to 1, according to the expression:

$$a = \frac{1}{1 + e^{-n}} .$$

The log-sigmoid transfer function is commonly used in multilayer networks that are trained using the Back Propagation algorithm ,in part because this function is differentiable.

Typically, a neuron has more than one input. A neuron with***R***inputs is shown in Fig 4. The individual inputs$\boldsymbol{p_1, p_2, \ldots, p_R}$areeach weighted by corresponding elements$\boldsymbol{w_{1,1}, w_{1,2}, \ldots, w_{1,R}}$of the *weight matrix* **W.**The neuron has a bias***b***which is summed with the weighted inputs to form the net input***n*****:** $\boldsymbol{n = w_{1,1}p_1 + w_{1,2}p_2 + \ldots + w_{1,R}p_R + b}$ .This expression can be written in matrix form: $\boldsymbol{n = Wp + b}$**,**where the matrix W for the single neuron case has only one row. Now the neuron output can be written as$\boldsymbol{a = f(Wp + b)}$ .Figure shown below represents the neuron in matrix form

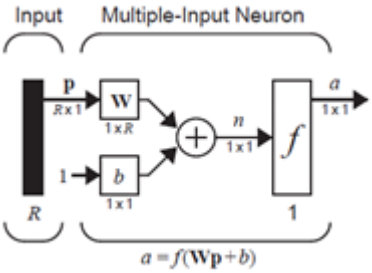


Fig 5: Neuron with R Inputs ;Matrix Notation

2.2 Network Architectures

Commonly one neuron, even with many inputs, is not sufficient. We might need five or ten, operating in parallel, in what is called a *layer*. A single-layer network of *S*

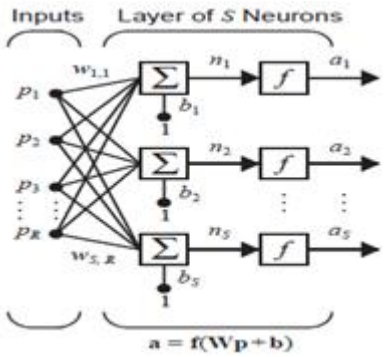


Fig 6: Layers Of S Neurons

is shown in Figure 6. Note that each of the *R* inputs is connected to each of the neurons and that the weight matrix now has *S* rows. The layer includes the weight matrix**W**,the summers, the bias vector **b**,the transfer function boxes and the output vector **a**.

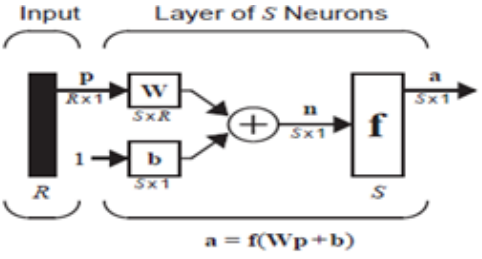


Fig 7: Layers of S Neurons, Matrix Notation

Some authors refer to the inputs as another layer, but we will not do that here. It is common for the number of inputs to a layer to be different from the number of neurons (i.e., $R \neq S$) .The S-neuron, R-input, one-layer network also can be drawn in matrix notation, as shown in Figure 7

2.2.1 Multiple Layers of Neurons

Now consider a network with several layers. Each layer has its own weight matrix W, its own bias vector b , a net input vector n and an output vector a . We need to introduce some additional notation to distinguish1 between these layers. We will use superscripts to identify the layers. Thus, the weight matrix for the first layer is written as $W^1$and the weight Matrix[3] for the second layer is written as W^2. Thi2s notation is used in the three-layer network shown in Fig 8. As shown,there are R inputs, S^1 neurons in the first layer, S^2 neurons in the second layer, etc. As noted, different layers can have different numbers of neurons. The outputs1of layers one and two are the inputs2for l1ayers two and three. Thus layer 2 can be viewed as a one-layer network with R=S^1inputs, S=S^2 neurons, and an $S^2 \times S^1$ weight matrix W^2 . The input to layer 2 is a^1 and the output is a^2.A layer whose output is the network output is called an output layer. The other layers are called hidden layers. The network shown in Fig 8 has an output layer (layer 3) and two hidden layers (layers 1 and 2).

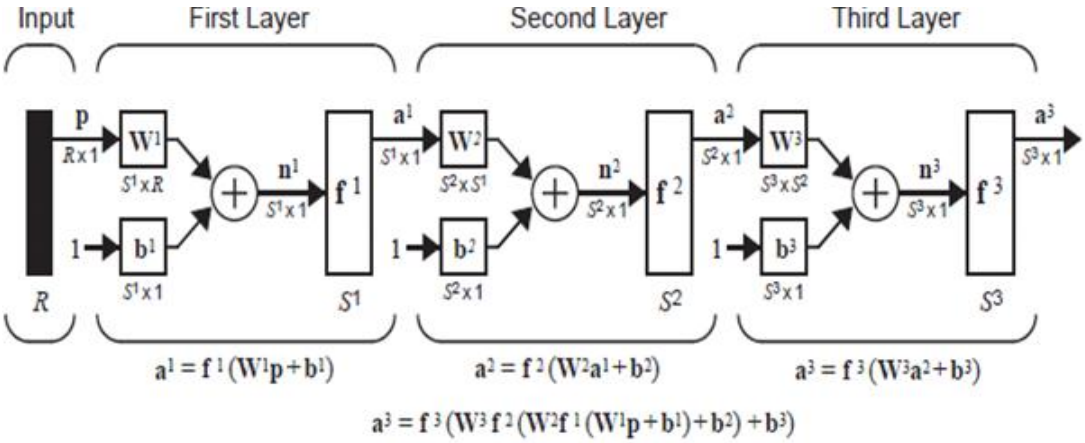
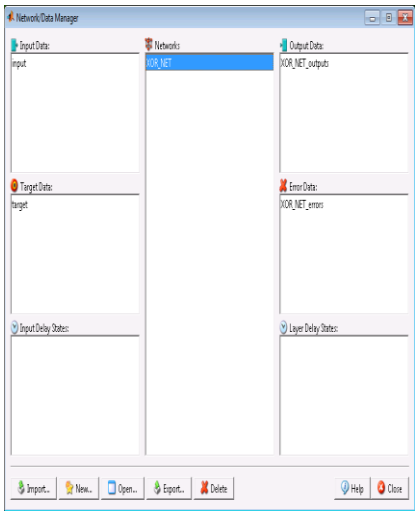


Fig8 :Three Layer Network

3. NEURAL NETWORK SIMULATION OF XOR LOGIC USING MATLAB

The XOR logic gives low output when both inputs are either high or low and gives high otherwise or it can also be stated that it gives low output for even parity of high inputs.

Now we will implement this logic using neural network simulation.For this MATLAB provides NNtool.In this simulation we put in use the Network/Data Manager GUI which is shown in Figure 9.The inputs given are [0 0 1 1;0 1 0 1] as shown in Figure 10 and the target output expected is [0 1 1 0]



9:Network Data Manager

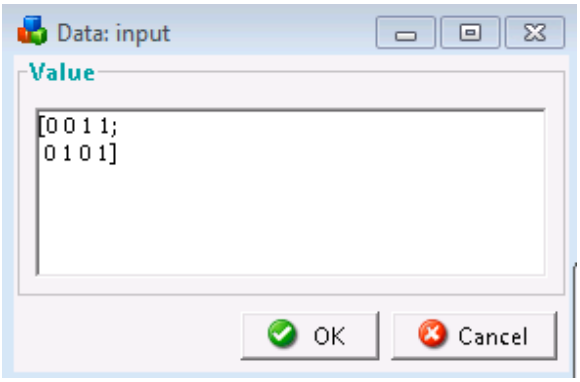


Fig 10:XOR network input

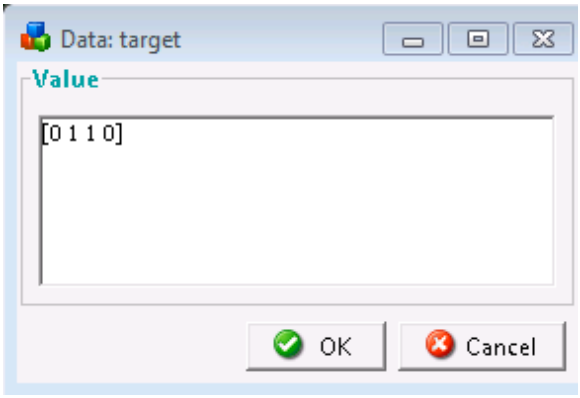


Fig 11:XOR network target

which is displayed in Figure 11.Since we are simulating a neural network the adaptive feature of neural network would tend to bring the practical output as close as possible to the actual or in our case the target output.
Once the input and target values are entered then we select the network characteristics. Here we have selected 2 layer neuron model for the XOR network.The two inputs are provided to the first layer by passing through a block of weights which help to keep the input-output relation linear. Also a bias is provided which is analogous to dc offset provided to an operational amplifier. These are summed up and provided as input to the first layer transfer function.Then the output of the first layer goes as input to the second layer and finally there is a single output to the system.The neural network for the XOR logic is provided in Figure 12.The derived output of this simulation is not exactly the
target output but an output with acceptable deviation.The neural network for XOR is trained under certain parameters so as to adapt to any undesirable change or variance in processing circumstances.

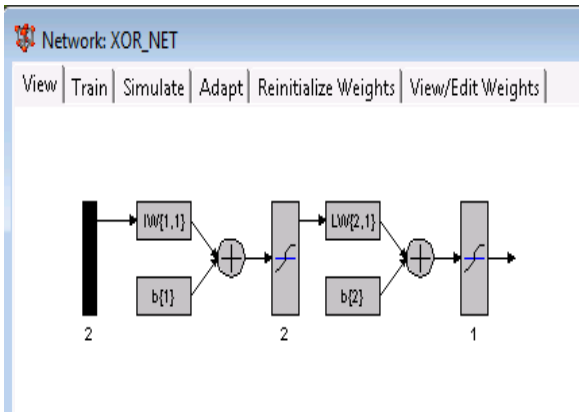


Figure 12:XOR network

The XOR network output after simulation is displayed in the output GUI as shown in Figure 13.Here the training parameters are set and the system of XOR network is trained to get the desired output for the given standard set of inputs.The error in the output comes out as a tolerable value thus the network has performed as per the expectation and we have a successful simulation of XOR logic.

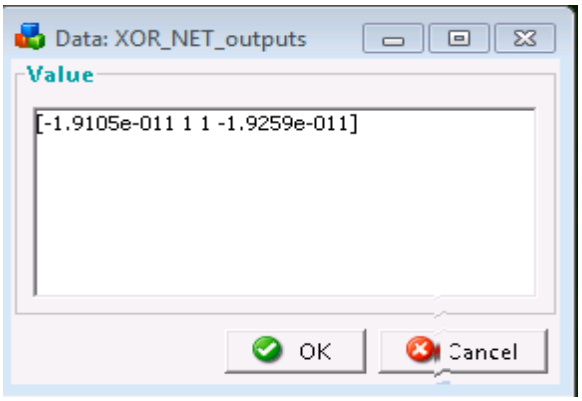


Fig 13:XOR network output

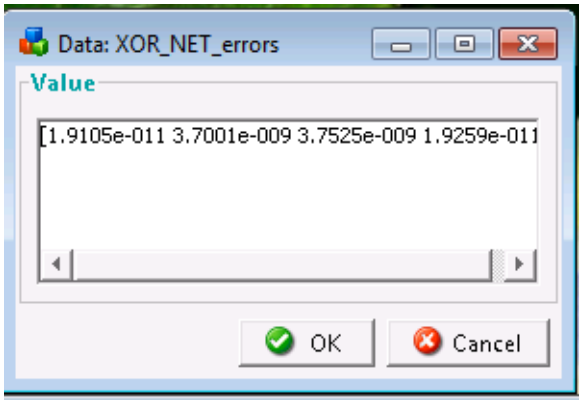


Fig 14:XOR Network Error

The neural network that we have simulated gives a control over the processing conditions which does not remain under manual supervision under various circumstances for e.g laying underground transmission lines.Now the network error which is shown in Fig13 is crucial as it gives the feedback to the training paradigm about the required variation of weights.Fig 15 shows the performance response for the epochs.

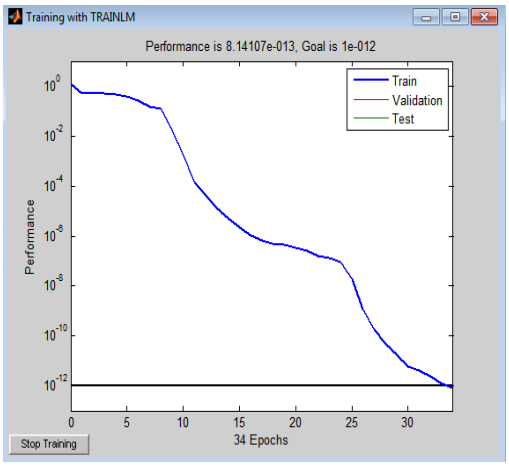


Figure 15: Performance response

4. CONCLUSION

The Neural Networks are vital when it comes to sustaining outputs under challenging conditions such as aerospace and mining fields where unexpected discrepancies could occur. The adaptivity of neural networks gives it an unique identity.

5. ACKNOWLEDGEMENTS

It is a great pleasure of us and a sense of satisfaction to present this Paper on Neyral Networks.We would take this opportunity to thank every possible person that made this Paper a success. We thank all our teachers, who have always been there to guide us to the right path, encourage us to work harder. We would like to thank Prof.SangeetaMishra at Thakur College of Engineering and Technology for their constant and valuable guidance. We Would also like to thank our Family and Friends for their Constant Support.

6. REFERENCES

[1] .Hagan, M. T., H.B. Demuth and M.H. Beale, Neural Network Design, PWS Publishing, Boston, 1996.

[2]. Bishop, C., Neural Networks for Pattern Recognition, Oxford, New York, 1995.

[3] .Haykin, S., Neural Networks: A Comprehensive Foundation, 2nd Ed., Prentice-Hall, Englewood Cliffs, NJ, 1999.

[4]. www.library.cmu.edu