IJREAT International Journal of Research in Engineering & Advanced Technology, Volume 1, Issue 5, Oct-Nov, 2013
ISSN: 2320 - 8791
www.ijreat.org

Neural Network Model for Load Forecasting

Amit Shrivastava[1], Dr Anand Khare[2]

[1]Research Scholar, CSE, Bhagwant University, Ajmer, Rajasthan, India

[2]Research Director, the MRPC Company, Hyderabad, Andhra Pradesh, India

ABSTRACT
Neural Network technique works on the working of Human Brain which contains billions of neurons in several layers. Each neuron is connected with all the neurons of previous layer neurons and also with all the neurons of output layer neurons i.e. it is a complete bipartite interconnection. Similar is the case with every neuron. All the input which a neuron receives is summed and if it reaches certain threshold value then neuron gives output which is called the activation of that neuron. In this paper we have used 3-layer neural network for load forecasting. Also the results as obtained are also shown.
The data used here for training and testing purpose is the actual Evening Hours Peak Load in "Mega Watt" taken from Madhya Pradesh Power Transmission Company Ltd. – State Load Despatch Centre , Jabalpur from Monthly System Performance Report – September 2012.

KEYWORDS: Neural Network, Activation, Linearly Separable Functions, Sigmoid Function, Hyperplane, Threshold Functions, Extend Gradient, HMI(Human Machine Interface), RMS-Root Mean Square Error

INTRODUCTION

The proposed model is using the logic like this- It has 7 neurons acting as input which will be holding the peak loads of mon - sun of 1st week of sep-2012, next 7 neurons are there in hidden layers, after this 7 neurons are there in output layer which will be giving the predicted loads of next week mon to Sunday. At output layer the comparisons is done with the actual peak loads of next week (i.e. 2nd week of Sep 2012) and adjust the weights accordingly to minimize the error. Once the difference of predicted and actual loads become less then 0.00 the weights are fixed. This completes the training. Then it is given the 2nd week peak loads as input and prediction is done for the 3rd week. Results were compared and % accuracy was calculated. The contemporary work done in this field is illustrated below.

[1] K. Y. Lee et.al. Has proposed Artificial Neural Network (ANN) non linear model based on two distinct load patterns namely weekday and weekend day load patterns. Past Loads are given as input and current day load is forecasted using 2 and 3 hidden layer network Model. Results are shown and % error was less then 3 %.

[2] Amera Ismail et.al. Had done the short term load forecasting of the next day based on the two years actual load data obtained from Duhok ELc. Control in Iraq. Error was less then 3.7 %.

[3] Mr. Rajesh Deshmukh et.al had used the 3 layer neural network model for 24 hour advanced load forecasting. They have used historical load data and real time load data of May 2011 from state load dispatch centre Jabalpur. Maximum % error was 9%.

[4] Author has discussed one hour ahead short load forecasting for reducing the complexity of neural network. In the proposed prediction method, the forecasted load power is obtained by adding a correction to the selected similar day data. Results show that the forecasted errors have gone to maximum of 14% and 1.63% on average.

[5] In this paper Author has proposed two different hybrid approaches based on Artificial Neural Network (ANN) models with Autoregressive (AR) method and Weighted Frequency Bin Blocks (WFBB) for doing next day load forecasting. He has used ANN structure having two layers composed of 49 and 24 neurons for input and output layers, respectively. The forecasting results were obtained from AR, ANN and the two hybrid models which are compared to each other in the sense of root mean square error (RMSE). It is observed that the RMSE values for the hybrid approaches are smaller than the conventional models.

[6] Author has used the levenberg-marquardt optimization technique which has one of the best learning rates as a back propagation algorithm for the multilayer feed forward ANN model using Matlab r2008b ANN toolbox. The forecasted next day 24 hourly peak loads are obtained based on the stationary output of the ANN with a performance mean squared error (MSE) of 5.84 e^{-6} and compares favorably with the actual power utility data. The results have shown that the proposed technique is robust in forecasting future load demands for the daily operational planning of power system distribution sub-stations in Nigeria.

[7] In this website a complete MATLAB program is given for load forecasting on the basis of past loads, Temperature of the day and holiday knowledge.
[8] This website was used for reference purpose.

THE COMPLETE THEORY OF NEURAL NETWORK

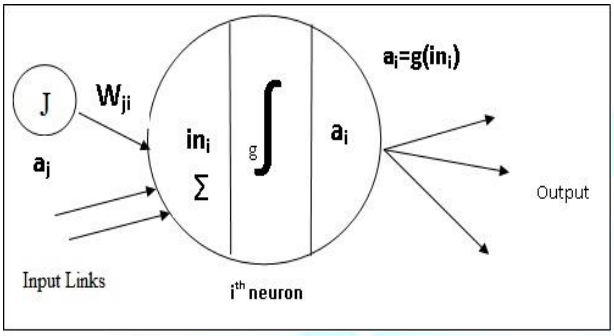

Figure 1- Relationship between I and J neuron.

Weight adjustment rule for 2 layers Neural Network is: - $\mathbf{W_{ji} = W_{ji} + \alpha * I_j * Err}$ (shown only for one weight. It had to be repeated for all the weights.). See Fig. 1
Where **α** – is the fraction or learning rate. So effectively we are adding the fraction of the total error to the weight to adjust it.
$\mathbf{W_{ji}}$ → weight of the link between j^{th} neuron and i^{th} neuron
$\mathbf{I_j}$ → Input to ith neuron = input * $\mathbf{W_{ji}}$ = activation of jth neuron * $\mathbf{W_{ji}}$
Err →T – O = correct output – predicted output
→ 2 layer Neural network is good in learning Linearly separable function

E.g. 1 We want to learn AND function
Truth table of "**AND**" Function

Table 1

I_1	I_2	Output
0	0	0
0	1	0
1	0	0
1	1	1

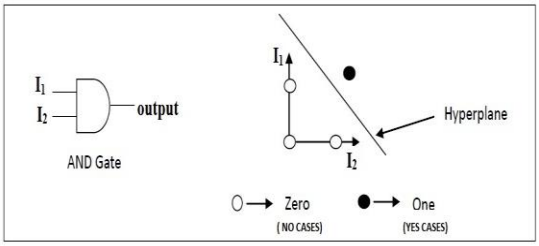

Figure 2 - AND Gate and its output

In figure 2 it can be seen that "yes" and "no" output's of AND Gate can be separated by a single Hyper plane. Such functions are known as "Linearly Separable Functions". Similarly OR Gate also comes into the category of Linearly Separable Functions. These kinds of functions can be easily learnt by 2-layer Neural Network.

E.g. 2 - We want to learn Ex-OR Function
Truth Table of "Ex-OR" Function

Table 2

I_1	I_2	Output
0	0	0
0	1	1
1	0	1
1	1	0

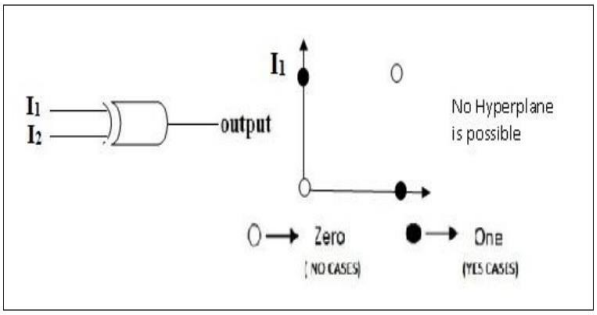

Figure 3 – EX-OR Gate and its Output

As can be seen in Figure 3 no single Hyper plane is possible which can separate out "yes" cases with "No" cases. This comes under the category of Non-Linear Function. These functions can be learnt effectively by 3 Layer neural network.

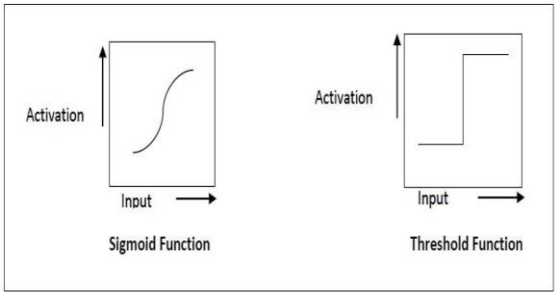

Figure 4 – Sigmoid and Threshold Functions

In Sigmoid function Input changes gradually and so is the activation level. In most of the cases Sigmoid functions are used for 3 layer neural network. However in certain cases Threshold functions are also required where output remains Zero up to a certain threshold value and then it switches to 1 after input gains certain threshold value. In threshold case g' (in) (derivative of g) is dropped down as design choice. See figure 4.

Sigmoid function = $\frac{g^2}{2} - \frac{g^3}{2}$

$\frac{d\,(\,Sigmoid\ Function\,)}{dg} = \frac{d\,(\frac{g^2}{2} - \frac{g^3}{2})}{dg} = \frac{2g}{2} - \frac{3g^2}{3} =$

$= g - g^2 = g\,(1\text{-}g)$

$g = \frac{1}{1+e^{-WX}}$ where W, X are vectors i.e.

$W.X = \sum_{i=1}^{n} W_i X_i$

Basic Reason for going to 3 layer Network is shown in Figure 5.

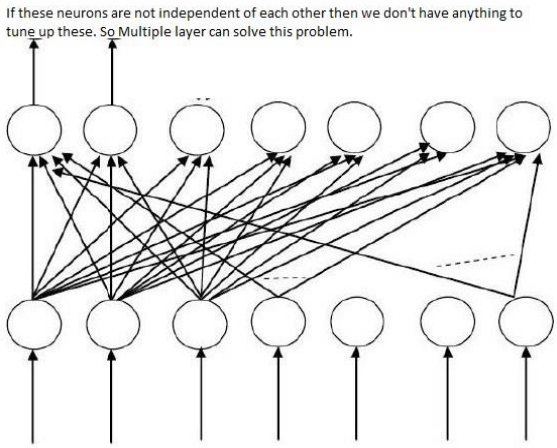

Figure 5 – Interdependency among Output Layer Neurons

If we tune up W_k then we can tune up all output neurons. See figure 6.

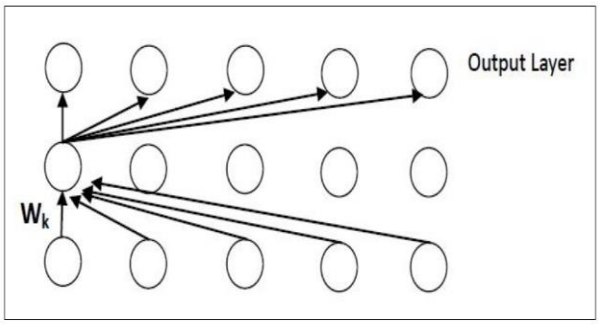

Figure 6 – Tuning up the output layer Neurons

PROPOSED MODEL FOR LOAD FORECASTING

In this technique we have used 7 neurons for the input, 7 in the hidden layer and 7 in the output layer. The input layer gets the input of peak loads of Monday to Sunday of the current week. Then the weights are assigned to the links randomly in such a manner that summation of weights of all the links which are falling on the hidden layer and which are falling in the output layer is equal to 1. Then activation is calculated for all hidden layer neurons and based on this, further we calculate the activations of output layer neurons.

In output layer also we have used 7 neurons which will predict peak load for the next week. First neuron predicted load will be treated as predicted peak load for Monday of next week, second for Tuesday and so on.

Next we compare the predicted load with the true output and do the weight adjustments until we get the predicted value close to true output.

We can use these predicted values for taking the decision that if any one load increases then one of the load that is connected out of many is disconnected.

The sample software is developed which tries to estimate the peak load of the future on the basis of previously stored peak loads. Neural Network self learning technique is used for Load forecasting.

The two layered feed-forward artificial neural network which has the feature of memory and learning is constructed and Back Propagation learning algorithm is used to obtain load forecasting value. The load parameter setting is chosen to set load forecasting error limit and the like.

Back propagation is a systematic method for training multi layer artificial neural network. It has a strong mathematical foundation. It is a multi layer forward network using extend gradient based delta learning rule, commonly known as back propagation rule. See Fig 7.

The Neural network model for load forecasting is developed in C++ language and HMI (Human Machine Interface) for controlling the instrument is developed in LabView. It's a completely connected bipartite graph. It is first trained and then tested for predicting peak loads. For this we need to have actual peak loads of each day for at least 3 weeks. Following steps are followed for training and testing the network:-

1. First part consists of training the neural network and in second part we do the testing of our neural network.
2. We train the neural network with first week data as true input and second week data as true output.
3. The output given by output layer neurons is compared with true output and does the weight adjustment until the predicted value become close enough to the true output.
4. We fix the weights and then give second week true peak loads as input and predict the third week load.
5. The predicted loads given by trained neural network are compared with the actual third week peak loads.
6. We calculate the efficiency/effectiveness of our trained network by comparing the output of the neurons with actual peak loads of 3rd week.
7. Above steps can be repeated for further weeks.

8. As we adjust the weights with more and more weeks i.e. as we train our neural network with more and more of data the weights begin to fine tune more and more resulting into greater accuracy.
9. Also we can consider temperature data of that particular day for further increasing our accuracy.

$\mathbf{a_i} = \mathbf{g}(\sum_j W_{ji} a_j)$ $\mathbf{a_j} = \mathbf{g}(\sum_k W_{kj} X_k)$

Activation function can be sigmoid or simple linear function.

Weights are adjusted as per following formula for linear function activation (i.e. without using sigmoid or any other activation function):-

$W_{ji} = W_{ji} + \alpha * a_j * Err_i$ [These weight adjustments have to be done with every W_{ji} connected with j^{th} neuron]

$W_{kj} = W_{kj} + \alpha * \sum [(yi-ai)*W_{ji}] * x_k$ [This had to be repeated with all inputs feeding into j^{th} neuron.]

See fig 8. Above weights adjustments were found out by Partial derivation of error difference w.r.t. weight. It will be different for sigmoid function.

α → Learning Rate

W_{ji} → weight between hidden layer jth neuron and output neuron i.

W_{kj} → It is Weight between input neuron k and hidden layer neuron j.

$\mathbf{a_i}$ → It is activation for output neuron i.

$\mathbf{a_j}$ → It is activation for hidden layer neuron j.

Err_i → True output – Output generated by the output layer neuron

Yi → True output

x_k → input value for the input layer neuron k.

Where $\sum [(yi-ai)*W_{ji}]$ → is error summation over all i (output neurons) for error generated due to j^{th} neuron.

All W_{kj} are to be adjusted which are feeding into j.

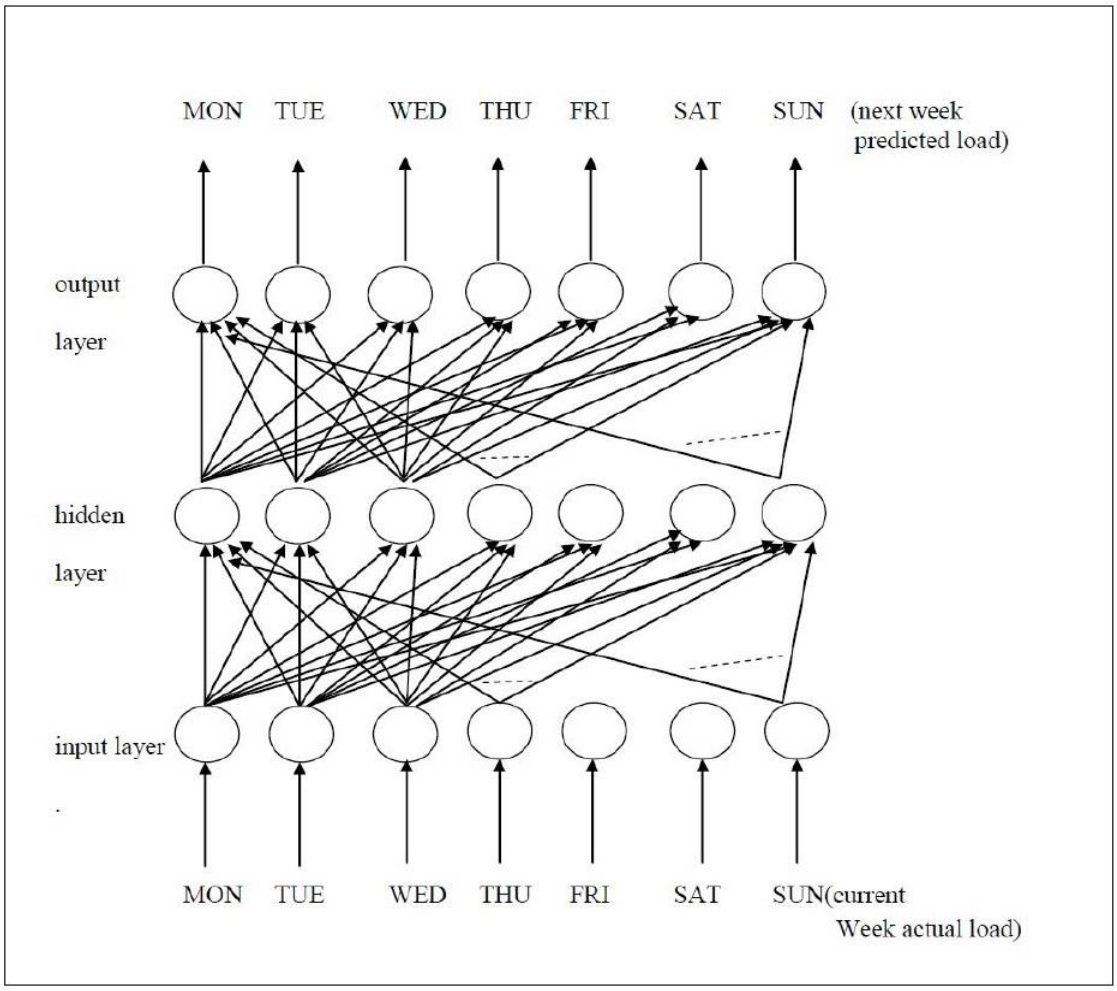

Figure 7: Neural Network model for Load forecasting

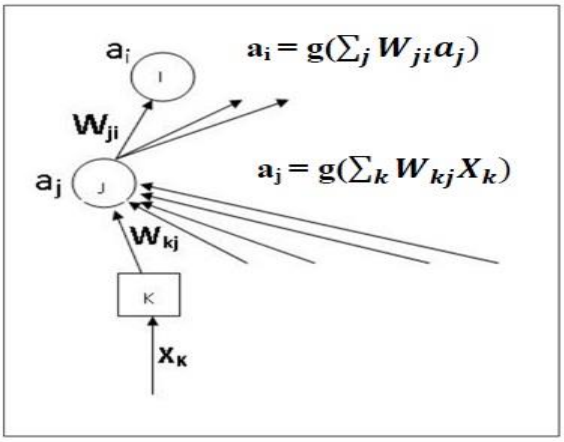

Figure 8:- Weight adjustment of a single path

DERIVATION-

Root Mean Square error is the most common error which is used.

RMS = $\frac{1}{2} E_{rr}^2 = \frac{1}{2} (Y - a_i)^2$

Where **Y**- is the true output, a_i – activation of i^{th} neuron

Now for wt adjustment for W_j **i.e**. wts between hidden layer and output layer, we see the change in error with respect to change in wt i.e. Output error is partially derivates w.r.t wt W_j. Then fraction of that term is added to the weight W_j to reduce the error.

$\frac{\partial E}{\partial W_j} = \frac{\partial(\frac{1}{2}E^2)}{\partial W_j}$ = E * $\frac{\partial E}{\partial W_j}$ = E * $\frac{\partial(Y - a_i)}{\partial W_j}$ = E * $\frac{\partial(Y - g(\sum_j W_j X_j))}{\partial W_j}$

Since Y is a constant so when derivates, it gets eliminated.

= E * ($\frac{-\partial(g(\sum_j W_j X_j))}{\partial W_j}$) = - E * g'(in) * X_j

Here, E = Y - a_i , i.e. true output – predicted output(activation of output neuron)

X_j – is Input if it is 1st layer ie Input layer neuron and it is a_j i.e. activation of jth neuron if this neuron belongs to higher layer neuron

So, wt updating rule W_{ji} now becomes

$w_{ji} \rightarrow w_{ji}$ - α * (- E * g'(in) * X_j)

$w_{ji} \rightarrow w_{ji}$ + α * (E * g'(in) * X_j)

$w_{ji} \rightarrow w_{ji}$ + α * (Yi - a_i) * X_j * g'(in)

If we are not using any function and simply relying on wt adjustment for getting the correct output then as a design choice we can safely drop derivative of g i.e. g'(in)

So, final Wt adjustment rule for w_{ji} i.e. wt between hidden layer and output layer becomes

$w_{ji} \rightarrow w_{ji}$ + α * (Yi - a_i) * X_j

⇨ $w_{ji} \rightarrow w_{ji}$ + α * (Yi - a_i) * a_j **-- Eqn 1**

This wt adjustment is only for one link wt i.e. between jth neuron and ith neuron. In a similar fashion all the wts between hidden layer and output layer are to be adjusted.

Now wt adjustment for W_{kj} i.e. link wt between hidden layer and input layer.

This W_{kj} is not only affecting a_j (activation of jth neuron of hidden layer) but all activations of output layer also.

Here also for wt adjustment for $W_{k,j}$ i.e. wts between hidden layer and Input layer, we see the change in error with respect to change in wt i.e. Output error is partially derivates w.r.t wt $W_{k,j}$. Then fraction of that term is added to the weight $W_{k,j}$ to reduce the error.

$\frac{\partial E}{\partial W_{k,j}} = \frac{\partial(\frac{1}{2}E^2)}{\partial W_{k,j}} = \frac{\partial(\frac{1}{2}(Y_i - a_i)^2)}{\partial W_{k,j}} = \frac{1}{2} * 2 * \sum_i (Y_i - a_i) * (\frac{\partial(Y_i)}{\partial W_{k,j}} - \frac{\partial(-a_i)}{\partial W_{k,j}})$

Here $\frac{\partial(Y_i)}{\partial W_{k,j}}$ gets eliminated as Y_i is constant w.r.t $W_{k,j}$.

= - $\sum_i (Y_i - a_i) * \frac{\partial a_i}{\partial W_{k,j}}$ = - $\sum_i (Y_i - a_i) * \frac{\partial g(in_i)}{\partial W_{k,j}}$ = -$\sum_i (Y_i - a_i)$ g'(in_j) $\frac{\partial(in_i)}{\partial W_{k,j}}$

= -$\sum_i (Y_i - a_i)$ g' (in_j) $\frac{\partial(\sum_j W_{j,i} * a_j)}{\partial W_{k,j}}$

Now $W_{j,i}$ is constant or not dependent on $W_{k,j}$ as these two wts are independent from each other. So, $W_{j,i}$ comes out of derivation as a constant. However a_j (activation of j^{th} neuron) is dependent on $W_{k,j}$.

So, => - $\sum_i ((Y_i - a_i) W_{j,i})$ * g' (in_j) * $\frac{\partial(a_j)}{\partial W_{k,j}}$ =

- $\sum_i ((Y_i - a_i) W_{j,i})$ g' (in_j) $\frac{\partial(\sum W_{k,j} a_k)}{\partial W_{k,j}}$

= - $\sum_i ((Y_i - a_i) W_{j,i})$ g' (in_j) a_k

Again as a design choice "g' (in_j)" can be dropped. a_k for input layer is X_k i.e. the input feeding into input layer.

= - $\sum_i [(Y_i - a_i) * W_{j,i}] * x_k$

So, Weight updating rule for $W_{k,j}$ = $W_{k,j}$ - α * (- $\sum_i [(Y_i - a_i) * W_{j,i}] * x_k$)

⇨ $W_{k,j} = W_{k,j} + \alpha * \sum_i [(Y_i - a_i) * W_{j,i}] * x_k$ **-- Eqn 2**

$\sum_i [(Y_i - a_i) * W_{j,i}]$ → summation had to be applied to all i (output neurons)

This wt updating rule had to be repeated with all inputs feeding into j^{th} neuron i.e. all $W_{k,j}$ are to be adjusted which are feeding into j.

⇨ $w_{ji} \rightarrow w_{ji} + \alpha * (Yi - a_i) * a_j$ **-- Eqn 1**

⇨ $W_{k,j} = W_{k,j} + \alpha * \sum_i [(Y_i - a_i) * W_{j,i}] * x_k$ **-- Eqn 2**

Procedure will be as follows:-

1. Apply Eqn-1 to adjust all output wts of 1st hidden layer neuron.
2. Apply Eqn-2 to adjust all input wts of 1st hidden layer neuron.
3. Repeat step 1 and 2 to adjust all input and output wts of all 7 neurons of hidden layer.
4. Assign all output wts of hidden layer to corresponding input wts of output layer.
5. Assign all input wts of hidden layer to corresponding output wts of input layer.

6. Calculate activations of all neurons of all layers.
7. Activation value of output layer neuron is our predicted value. Compare the predicted value/Load with actual Load. Store the differences.
8. Check the summation of all 7 differences (from 7 output layer neurons). Repeat the above procedure until the summation of differences becomes 0.00.
   The value of Learning rate α is very important so as to achieve the summation of differences = 0.00. We have used learning rate α= 0.00000001 for Eqn-1 & α = 0.000000001 for Eqn-2. This has been obtained by hit & trial method.
   No of passes required to reach the goal varies from 15 to 32 which is near the optimum no of passes for this particular scenario. See Figure 9.

PSEUDO CODE FOR NEURAL NETWORK MODEL FOR TRAINING(WHICH WAS ACTUALLY DEVELOPED IN C++)

1. Define array of structures for input layer, hidden layer and output layer
2. The input layer neuron structure will contain 2 fields namely (1) input load (2) array of 7 output weights
3. The hidden layer neuron structure will contain 3 fields (1) array of 7 output weights (2) array of 7 input weights (3) activation of a neuron
4. The output layer neuron structure will contain 2 fields (1) activation (2) array of 7 input weights
5. Assign 7 random inputs weights to hidden layer neuron and total weights coming to one hidden layer neuron should be 1. Total 49 input weights will be assigned for 7 neurons of hidden layer.
6. Assign 7 random inputs weights to output layer neuron and total weights coming to one output layer neuron should be 1. Total 49 input weights will be assigned for 7 neurons of output layer.
7. Assign hidden layer input wts to output weights of input layer.
8. Assign output layer input neuron wts to hidden layer output wts.
9. calculate activations of hidden and output layer neurons
10. Calculate the diff between true output and predicted value.
11. Calculate the error to be back propagated to hidden layer.
12. As per the error readjust the weights.
13. Calculate the updated activations.
14. Repeat steps 7 through 13 until the difference of the predicted load and actual load becomes less then 0.00.

6 Text Files are generated from this program.

1. Diff1.txt – containing difference of the True output and Predicted Output for all passes.
2. Acti.txt – Containing the activation values of 7 output layer neurons for all passes.
3. Iwts.txt – Containing 49 output weights of 7 neurons of Input layer for all passes.
4. Hwts.txt – Containing 49 input and 49 output weights of 7 Hidden layer neurons for all passes.

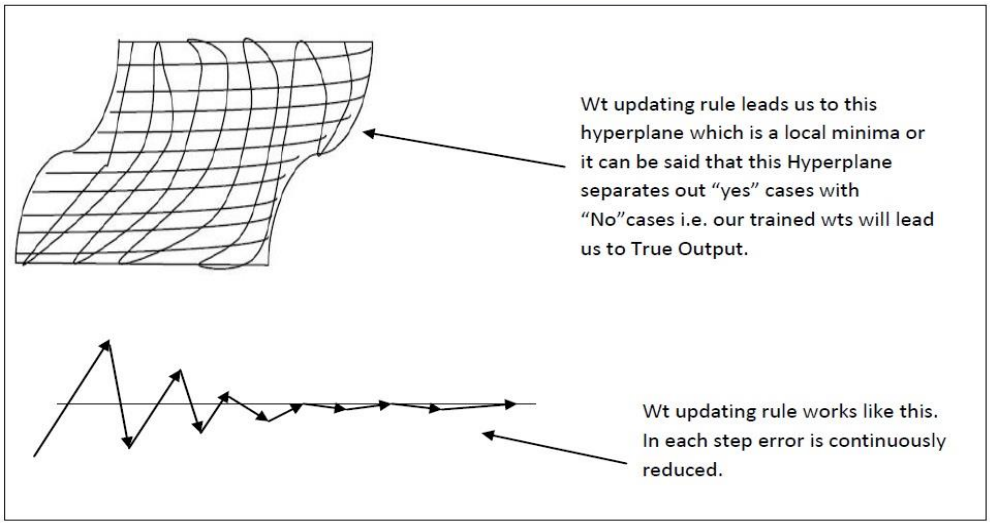

Figure 9 – Working of the Weight updating rule algorithm

5. Owts.txt – Containing 49 input weights of 7 output Layer neurons for all passes.
6. Fwts.txt – containing the final trained weights of Input, Hidden and Output layer neurons.

DIFF1.TXT

Below data was generated in one typical run. We can see that the "sum of difference" is behaving as shown in Figure 9 until it reaches 0.005175.

difference
981.311573 ,549.520073 ,1388.227402 ,1668.05084 ,1146.024601 ,1382.751927 ,1590.259518 ,
sum of difference = 8706.145935

Individual difference 1 between true and predicted value
-2084.053529 ,-1767.631439 ,-2260.168215 ,-2405.55368 ,-2029.449868 ,-2147.376608 ,-2231.996729
sum of difference 1 = -14926.230067

Individual difference 2 between true and predicted value
3923.541324 ,3614.507113 ,3930.282111 ,3989.143243 ,3630.133174 ,3668.477084 ,3660.478437 ,
sum of difference 2 = 26416.562486

Individual difference 3 between true and predicted value
-5639.05569 ,-5391.029158 ,-5415.864814 ,-5355.140693 ,-5079.857214 ,-5009.29444 ,-4875.428431
sum of difference 3 = -36765.67044
……
…….
Individual difference 12 between true and predicted value
0.068962 ,0.028012 ,0.119648 ,0.145648 ,0.094342 ,0.117803 ,0.146 ,
sum of difference 12 = 0.720415

Individual difference 13 between true and predicted value
-0.042856 ,-0.050623 ,-0.027894 ,-0.020576 ,-0.030623 ,-0.023893 ,-0.015532 ,
sum of difference 13 = -0.211999

Individual difference 14 between true and predicted value
0.005376 ,0.001491 ,0.010293 ,0.012809 ,0.007937 ,0.010211 ,0.012945 ,
sum of difference 14 = 0.061062

Individual difference 15 between true and predicted value
-0.003783 ,-0.00457 ,-0.002321 ,-0.001604 ,-0.002618 ,-0.001959 ,-0.001143 ,
sum of difference 15 = -0.017997

Individual difference 16 between true and predicted value
0.00041 ,0.00004038 ,0.000887 ,0.001131 ,0.000667 ,0.000887 ,0.001153
sum of difference 16 = 0.005175

ACTI.TXT

It can be seen that after 16th pass the Activation (Predicted Output) has become almost same as true output.

0 TIME--ACTIVATIONS OF OUTPUT LAYER
1-4361.688427 , 2-4276.479927 , 3-4169.772598 , 4-4026.94916 , 5-3925.975399 , 6-3803.248073 , 7-3712.740482 ,
1 TIME--ACTIVATIONS OF OUTPUT LAYER
1-7427.053529 , 2-6593.631439 , 3-7818.168215 , 4-8100.55368 , 5-7101.449868 , 6-7333.376608 , 7-7534.996729 ,
……
…….
16 TIME--ACTIVATIONS OF OUTPUT LAYER
1-5342.99959 , 2-4825.99996 , 3-5557.999113 , 4-5694.998869 , 5-5071.999333 , 6-5185.999113 , 7-5302.998847 ,
true output
5343 , 4826 , 5558 , 5695 , 5072 , 5186 , 5303 ,
true input
5236 , 5214 , 5060 , 5122 , 5266 , 5643 , 5064 ,

HWTS.TXT

Each Neuron is associated with 7 weights. These are the finally trained weights which were used to predict 3rd week load when 2nd week load is given as Input.

SIXTEENTH TIME→INPUT WTS OF HIDDEN LAYER/ OUTPUT WTS OF INPUT LAYER
neuron 1-->
1→ -0.06552 , 2→-0.064374 , 3→-0.062901 , 4→-0.063153 , 5→-0.065965 , 6→-0.069896 , 7→0.92586 ,
neuron 2-->
1→-0.063037 , 2→-0.063613 , 3→-0.058686 , 4→-0.062035 , 5→-0.065151 , 6→-0.069954 , 7→0.911899
neuron 3-->
1→-0.063747 , 2→-0.056599 , 3→-0.061731 , 4→-0.062069 , 5→-0.057314 , 6→-0.069124 , 7→0.890868
neuron 4-->
1→-0.041113 , 2→-0.061506 , 3→-0.05441 , 4→-0.030233 , 5→-0.059147 , 6→-0.066051 , 7→0.826926 ,
neuron 5-->
1→-0.057111 , 2→-0.054565 , 3→-0.055689 , 4→-0.055707 , 5→-0.057972 , 6→-0.061813 , 7→0.819416
neuron 6-->
1→-0.05735 , 2→-0.056892 , 3→-0.054407 , 4→-0.054357 , 5→-0.050112 , 6→-0.061812 , 7→0.802961
neuron 7-->
1→0.034536 , 2→0.034634 , 3→0.034231 , 4→0.039024 , 5→0.034749 , 6→0.038036 , 7→0.877309 ,

SIXTEENTH TIME→OUTPUT WTS OF HIDDEN LAYER/ INPUT WTS OF OUTPUT LAYER

neuron 1-->
1→0.13025 , 2→0.105535 , 3→0.15081 , 4→0.164491 , 5→0.132602 , 6→0.146357 , 7→0.160419 ,

neuron 2-->
1→0.132035 , 2→0.103239 , 3→0.15442 , 4→0.162255 , 5→0.136324 , 6→0.142161 , 7→0.152933 ,

neuron 3-->
1→0.128125 , 2→0.101927 , 3→0.151184 , 4→0.159444 , 5→0.129652 , 6→0.141457 , 7→0.15171 ,

neuron 4-->
1→0.120545 , 2→0.102095 , 3→0.155515 , 4→0.168693 , 5→0.128412 , 6→0.134267 , 7→0.163406 ,

neuron 5-->
1→0.122176 , 2→0.092643 , 3→0.131908 , 4→0.144892 , 5→0.125589 , 6→0.13902 , 7→0.136428 ,

neuron 6-->
1→0.113881 , 2→0.100239 , 3→0.131792 , 4→0.14428 , 5→0.11658 , 6→0.126662 , 7→0.136224 ,

neuron 7-->
1→0.623256 , 2→0.594095 , 3→0.603684 , 4→0.597744 , 5→0.564702 , 6→0.558044 , 7→0.546545.

Another program is developed in C++ which read these finally trained weights from the file "**fwts.txt**". The 2nd week load which was treated as **true output** for 1st program for training purpose was treated as **true input** for 2nd program. Then the predicted load is found out with the help of these trained weights and true input of second Week. Finally predicted load is compared with true load of 3rd week and % accuracy was calculated.

Pseudo code for neural network model for testing(which was actually developed in C++)

1. Adjust the weights of all neurons of all layers as per the finally trained weights.
2. Give the 2nd week true peak loads as input and predict for 3rd week (by calculating the activations of output layer neurons).
3. Find the accuracy of prediction.

True data used in this program is as follows:
1st week - 5236 , 5214 , 5060 , 5122 , 5266 , 5643 , 5064 ,
2nd week - 5343 , 4826 , 5558 , 5695 , 5072 , 5186 , 5303
3rd week – 5043, 5622, 5816, 5934, 5865, 6043, 6121,
Above values are in MW.

CONCLUSIONS

It was concluded that Network model was able to predict the next week Load with 96.5% accuracy i.e. error was less then 3.5%. See figure 10. It was further fine tuned when similar day data of several weeks was taken, averaged them and then tuned up the final weights. The accuracy of a typical run was increased to almost 98% when several weeks data along with temperature of that day was taken into consideration. Most of the runs gave accuracy between 97% and 98%.

OUTPUT OF THE PROGRAM

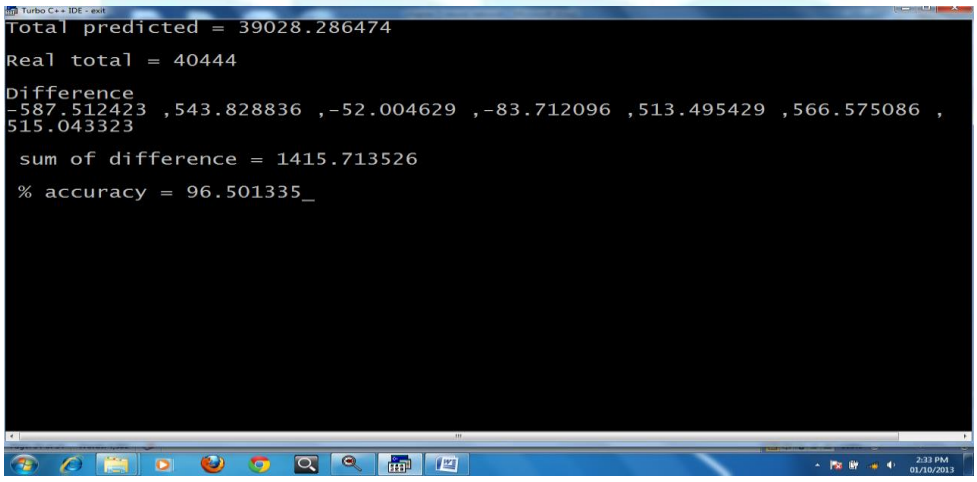

Figure 10: output of the testing program

REFERENCES

[1] K. Y. Lee, Y. T. Cha, J. H. Park , "**Short-Term Load Forecasting Using An Artificial Neural Network**", in transactions on power systems, vol. 7. No. 1, February 1992.

[2] Amera Ismail Melhum, Lamya Abd Allateefa Omar, Sozan Abdulla Mahmood, "**Short Term Load Forecasting Using Artificial Neural Network**", in international journal of soft computing and engineering (ijsce) issn: 2231-2307, volume-3, issue-1, march 2013

[3] Mr. Rajesh Deshmukh , Dr. Amita Mahor, " **Artificial Neural Network Based Approach for short load forecasting**", in International Journal of Advanced Computer Research (ISSN (print): 2249-7277 ISSN (online): 2277-7970) Volume 1 Number 2 December 2011

[4] Tomonobu Senjyu, Hitoshi Takara, Katsumi Uezato, And Toshihisa Funabashi,"**One-Hour-Ahead Load Forecasting Using Neural Network**", in IEEE transactions on power systems, vol. 17, no. 1, February 2002

[5] Mehmet Kurban And Ummuhan Basaran Filik, "**Next Day Load Forecasting Using Artificial Neural Network Models With Autoregression And Weighted Frequency Bin Blocks**", in international journal of innovative computing, information and control volume 5, number 4, April 2009

[6] Muhammad Buhari, Member, Iaenga And Sanusi Sani Adamu,"**Short-Term Load Forecasting Using Artificial Neural Network**", in IMECS 2012 , March 14-16 2012, Hong Kong.

[7]http://www.mathworks.com/matlabcentral/fileexchange/28684-electricity-load-and-price-forecasting-webinar-casestudy/content/Electricity%20Load%20 &%20 Price %20Forecasting/Load/html/LoadScriptNN. html

[8] http://www.posoco.in

THE AUTHOR

AMIT SHRIVASTAVA has obtained BE (1999), M.E (2006) in Computer Science from DAVV University, Indore India. His interests are in Computer Applications in Power Generation and Transmission Industry. Presently he is submitting thesis for his PhD at Bhagwant University, Ajmer Rajasthan India. He has published two research papers on the topics of SCADA Automation in Power-Industry, mail id - amitshrivastavaphd at the rate gmail dot com.

DR ANAND KHARE has obtained **MSc** from Wales and PhD from London. Presently he is a Research Director at MRPC Company, Hyderabad. Formerly Faculty IISc Bangalore and Hest Ham College London. Mail id – ak_mrpc at the rate hotmail dot com