1) Since the g(u) function is bounded between 0 and 1, it will be necessary to scale the output so that they correspond to the ranges for the desired outputs (total sales in millions of dollars, inventor level in days, and percent yield). The outputs could be interpreted as a fraction with respect to some max value, such as maximum inventory storage capacity in number of days in the case of inventory level. When updating weights, we can compare each tk to its appropriate, scaled zk; we would then rescale the difference to a value between 0 and 1. Issues may arise if the g(u) function is too steep and many values lie near the max or min Decreasing the rate of transition from 0 to 1 may help in this case. Ideally, a reasonable amount of historical data is present in order to train and test the network.

5) Node weights were initialized as suggested in class notes: hidden weights were given small values around zero and output weights were set pseudo-randomly to either -1 or 1, unless there were an odd number of total weights, in which case all bias weights were set to 0. I decided to update all weights after each training case using the delta rule described in the class notes. I trained the NN with 100 cases, and then tested it with another 100 cases and calculated *E* for the test cases. To test which parameter were best, I varied the step size, Lambda, the function type used (g( ) or h( )), and the number of hidden nodes used. For each parameter set, 10 training and test sets were run, with the *E* values from each of the 10 test sets recorded and averaged to obtain and average *E* value for the parameter set. From observing the *E* scores, the best g( ) and best h( ) cases perform similarly; on average, though, the g( ) function performs much better than the h( ) function, with the average *E* value from all g( ) cases an order of magnitude smaller than that for all h( ) cases (see right-most column of table below). The best case found used 5 hidden nodes and a Lambda value of 1, though other cases with similar parameters (2,3,4,10 hidden nodes with Lambda value 1, and the Lambda value of 0.75 and 1.5 cases) did not perform significantly worse. The data table below includes the test results.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ***Function*** | ***Hidden Nodes*** | ***Lambda Value*** | ***Average E (10 Trials)*** | |
| g( ) | 2 | 0.025 | 0.0642 |  | |
| g( ) | 3 | 0.025 | 0.0605 |  | |
| g( ) | 4 | 0.025 | 0.0845 |  | |
| g( ) | 5 | 0.025 | 0.0605 |  | |
| g( ) | 10 | 0.025 | 0.0957 |  | |
| g( ) | 2 | 0.05 | 0.0680 |  | |
| g( ) | 3 | 0.05 | 0.0600 |  | |
| g( ) | 4 | 0.05 | 0.0682 |  | |
| g( ) | 5 | 0.05 | 0.0573 |  | |
| g( ) | 10 | 0.05 | 0.0800 |  | |
| g( ) | 2 | 0.75 | 0.0419 |  | |
| g( ) | 3 | 0.75 | 0.0425 |  | |
| g( ) | 4 | 0.75 | 0.0417 |  | |
| g( ) | 5 | 0.75 | 0.0456 |  | |
| g( ) | 10 | 0.75 | 0.0446 |  | |
| g( ) | 2 | 1 | 0.0420 |  | |
| g( ) | 3 | 1 | 0.0439 |  | |
| g( ) | 4 | 1 | 0.0438 |  | |
| g( ) | 5 | 1 | ***0.0410*** |  | |
| g( ) | 10 | 1 | 0.0447 |  | |
| g( ) | 2 | 1.5 | 0.0430 |  | |
| g( ) | 3 | 1.5 | 0.0426 |  | |
| g( ) | 4 | 1.5 | 0.0437 |  | |
| g( ) | 5 | 1.5 | 0.0465 | Mean Val. | |
| g( ) | 10 | 1.5 | 0.0447 | ***0.05*** | |
| h( ) | 2 | 0.025 | 0.1156 |  | |
| h( ) | 3 | 0.025 | 0.1381 |  | |
| h( ) | 4 | 0.025 | 0.3108 |  | |
| h( ) | 5 | 0.025 | 0.1083 |  | |
| h( ) | 10 | 0.025 | 0.0833 |  | |
| h( ) | 2 | 0.05 | 0.0842 |  | |
| h( ) | 3 | 0.05 | ***0.0679*** |  | |
| h( ) | 4 | 0.05 | 0.1796 |  | |
| h( ) | 5 | 0.05 | 0.0642 |  | |
| h( ) | 10 | 0.05 | 0.1030 |  | |
| h( ) | 2 | 0.75 | 0.2628 |  | |
| h( ) | 3 | 0.75 | 0.5296 |  | |
| h( ) | 4 | 0.75 | 0.7471 |  | |
| h( ) | 5 | 0.75 | 0.8273 |  | |
| h( ) | 10 | 0.75 | 0.8473 |  | |
| h( ) | 2 | 1 | 0.7481 |  | |
| h( ) | 3 | 1 | 0.8270 |  | |
| h( ) | 4 | 1 | 0.8228 |  | |
| h( ) | 5 | 1 | 0.8637 |  | |
| h( ) | 10 | 1 | 0.8813 |  | |
| h( ) | 2 | 1.5 | 0.8371 |  | |
| h( ) | 3 | 1.5 | 0.7911 |  | |
| h( ) | 4 | 1.5 | 0.8729 |  | |
| h( ) | 5 | 1.5 | 0.8883 | Mean Val. | |
| h( ) | 10 | 1.5 | 0.8834 | ***0.52*** | |

MATLab Code for training NN:

TestFunc = inline('(x1^2+x2^2-1)^2','x1','x2');

% gFunc = inline('1/(1+exp(-u))', 'u');

% hFunc = inline('2/(1+exp(-u))-1','u');

if input.Func == 2

Func=inline('2/(1+exp(-u))-1','u');

else

Func=inline('1/(1+exp(-u))', 'u');

end

TrainingSet = input.TrainingSet;

T = size(TrainingSet); % [numTrainingSets numInputs]

I = T(2); % numInputs

J = input.HiddenNodes;

K = 1; % numOutPutNodes

Lam = input.Lam;

tol = input.tol;

% Initialize weights. a's around 0, b's [-1,0,1]

HiddenWeights = (rand(J,I+1)-.5)/2; % make J by I+1; a0 + aK for each input node; a's

OutPutWeights = round(rand(K,J+1)); % make K by J+1; b's

OutPutWeights(OutPutWeights==0)=-1;

if mod(J,2)==1, OutPutWeights(:,1) = 0; end

% E = 999;

dE = 999;

t = 1;

while t <= T(1) && dE > tol

% Calculate ANN output for given test case

% 1) Calc values of hidden nodes

TempTrain = ones(J,1)\*TrainingSet(t,:);

% Temp is the output value for each hidden node

Temp = HiddenWeights(:,1) + sum(TempTrain.\*HiddenWeights(:,2:end),2);

Temp = Func(Temp); % this is an J by 1 vector of values for each hid node

if size(Temp,1) ~= J, Temp=Temp';end

% size(Temp),size(OutPutWeights),pause

% 2) Calc values of output nodes

% zTemp is the output value for each output node

zTemp = OutPutWeights(:,1) + sum((ones(K,1)\*Temp').\*OutPutWeights(:,2:end));

zTemp = Func(zTemp); % this is K by 1 vector of ANN outputs

if size(zTemp,1) ~= K, zTemp=zTemp';end

% 3) Update weights on nodes based on output for this test case

ETemp = zTemp - TestFunc(TrainingSet(t,1),TrainingSet(t,2));

if input.Func == 1

pk = (ETemp).\*zTemp.\*(1-zTemp); % size K by 1

elseif input.Func == 2

pk = ETemp.\*(zTemp+1).^2;

else

pk = (ETemp).\*zTemp.\*(1-zTemp); % size K by 1

end

% create vector that is sum(p sub k times b sub jk), size J by 1

pbj = sum((pk\*ones(1,J)).\*OutPutWeights(:,2:J+1),1)';

if input.Func == 1

qj = pbj.\*(Temp).\*(1-Temp); % J by 1

elseif input.Func == 2

qj = pbj.\*(Temp+1).^2;

else

qj = pbj.\*(Temp).\*(1-Temp); % J by 1

end

% Find partial dir. of E with respect to weights

dEdb = (pk\*ones(1,J+1)).\*[ones(K,1),(ones(K,1)\*Temp')]; % K by J+1 matrix

dEda = (qj\*ones(1,I+1)).\*[ones(J,1),(ones(J,1)\*TrainingSet(t,:))]; % J by I+1 matrix

% Do weight update using delta method

HiddenWeights = HiddenWeights - Lam\*dEda;

OutPutWeights = OutPutWeights - Lam\*dEdb;

% 4) Update stopping conditions

t = t + 1;

% ETemp = .5\*(ETemp)^2;

% dE = abs(E-ETemp);

% E = ETemp;

end