**Problem**

Take an MLP Neural Network with 3 layers with:

2 Inputs, 1 output, and any number of hidden neurons

Implement sample-by-sample training and show that it works

**Design**

1 hidden neuron 2 hidden neurons

Y1= V0 + V1\*Z1 Y1= V0 + V1\*Z1 + V2\*Z2



3 hidden neurons; Y1= V0 + V1\*Z1 + V2\*Z2 + V3\*Z3

g1= u01 + x1\*u11+ x2\*u21 g2= u02 + x1\*u12+ x2\*u22 g3= u03 + x1\*u13+ x2\*u23

Z1= 1/(1+exp(-g1)) Z2= 1/(1+exp(-g2)) Z3= 1/(1+exp(-g3))

User input data or randomized at runtime:

U = V = V0

**Training Data** output = x1 + x2

|  |  |  |
| --- | --- | --- |
| x1 | x2 | d |
| -3 | -1 | -4 |
| -2 | 1.2 | -0.8 |
| -1 | 0 | -1 |
| 0 | -3 | -3 |
| 1.2 | 2.2 | 3.4 |
| 2.2 | -2 | 0.2 |
| 2.5 | 2.5 | 5 |

**Error function and BP equations**

Error = E = 2

Update function: wnew = wold –η ; where w is any U, or V value

Vh\*Zh(Zh-1) Vh\*Zh(Zh-1)\*xn

**Results and Conclusion**

I wrote my code as a function call with the following parameters:

(# hidden neurons, # epoch, hidden bias multiplier, eta)

The results reported below are for constant values:

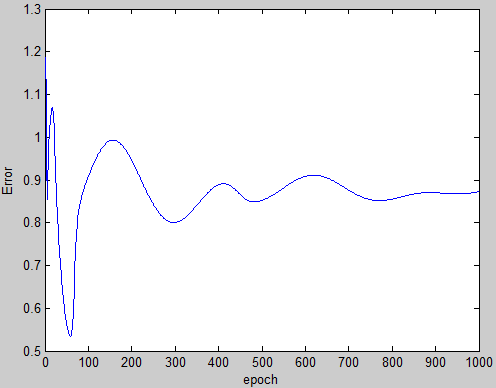
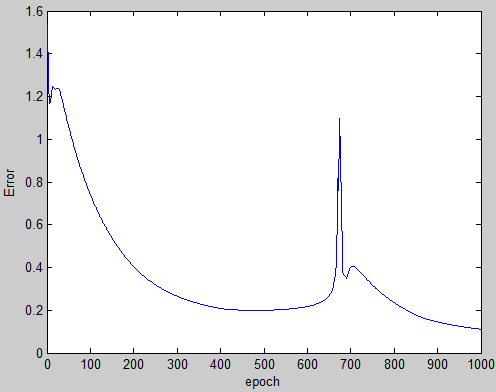
#epoch = 1000 Hidden bias multiplier = 1 eta = 0.1

Errors for an NN with 2 and 3 hidden neurons were compared with 10 iterations each.

The reported error is the absolute error Ea.

|  |  |
| --- | --- |
| 2 Hidden | 3 Hidden |
| 1.7202 | 0.7195 |
| 1.6768 | 0.1974 |
| 0.2891 | 0.2880 |
| 0.2033 | 0.6179 |
| 0.7089 | 0.1450 |
| 0.8164 | 0.8204 |
| 0.4711 | 0.4206 |
| 0.3812 | 0.3242 |
| 0.9629 | 0.1910 |
| 0.8728 | 0.1111 |

The trials I ran with my code show that my addition function works better with 3 hidden neurons. Below are two error convergence plots:

2 Hidden 3 Hidden

**Code**

function HW1(nhid, nepoch, ux, eta)

% clc;

% clear all;

%define number of input, hidden, and output neurons

% nepoch = 1000; ux = 1; eta=0.1; nhid=2; %3 hidden H

nin=2; %2 input N

nout=1; %1 output M

%define matrix for x input data:

% 1 2 3 ... number of tests

% x1 x11, x12, x13, ... x1t

% x2 x21, x22, x23, ... x2t

%

% number tests data = t= 5

t = 7;

x=zeros(2, t);

%

% %matrix of weights between input -- hidden u

% % x1 x2 ... N+1(bias)

% %u1 u11 u21 ... u01

% %u2 u12 u22 ... u02

% %u3 u13 u23 ... u03

% %

% u = zeros(nhid, nin+1)

% %BP vector for du = zeros(nhid, nin+1)

% du = zeros(nhid, nin+1);

% % x1 x2 ... N+1(bias)

% %u1 du11 du21 ... du01

% %u2 du12 du22 ... du02

% %u3 du13 du23 ... du03

% %

%matrix of weights between input -- hidden u

% u1 u2 u3

%x1 u11 u12 u13

%x2 u21 u22 u23

%N+1 u01 u02 u03

%

u = zeros(nin+1, nhid);

%BP vector for du = zeros(nhid, nin+1)

du = zeros(nin+1, nhid);

% u1 u2 u3

%x1 du11 du12 du13

%x2 du21 du22 du23

%N+1 du01 du02 du03

%

%derfine vector for gamma = g(size H)

g=zeros(nhid,1);

%define vector for Z = Z(size H)

Z=zeros(nhid, 1);

%define vector for V = V(size H)

V=zeros(nhid, 1);

%V0(nout, 1) = output bias

V0 = zeros(nout, 1);

%BP vector for dV = zeros(nhid,1)

dV = zeros(nhid,nout);

%BP vector for output bias dV0 = zeros(nout,1)

dV0 = zeros(nout,1);

%define output vector y = zeros(M,t)

y=zeros(nout,t);

%desired output vector d=(M,1)

d = zeros(nout, t);

%%\_\_\_\_Define input matrix\_\_\_ (test data input)

%define matrix for x input data:

% 1 2 3 ... number of tests

% x1 x11, x12, x13, ... x1t

% x2 x21, x22, x23, ... x2t

%

%x = [-1.0 -0.5 0 0.5 1; -1.0 -0.5 0 0.5 1.0];

x= [-3 -2 -1 0 1.2 2.2 2.5; -1 1.2 0 -3 2.2 -2 2.5];

%define test data desired output

%d = [-1 -0.5 0 0.5 1.0];

%d = [0 0.25 0.5 0.75 1.0];

d= [-4 -0.8 -1 -3 3.4 0.2 5];

%randomize weights of links input -- hidden

%and randomize hidden layer bias (nin+1)

%matrix of weights between input -- hidden u

% u1 u2 u3

%x1 u11 u12 u13

%x2 u21 u22 u23

%... ... ...

%N+1 u01 u02 u03

%

for i = 1:(nhid);

for j = 1:1:(nin+1);

u(j, i)= rand\*ux;

end

end

%u = [0.1985 0.1985 0.1985; 0.1979 0.1979 0.1979; 0.199 0.199 0.199];

%u = [0.1985; 0.1985; 0.1985];

u1 = u;

%Randomize V = zeros(H, M)

for j= 1:1:nhid;

for i=1:nout;

V(j,i) = rand;

end

end

% V = [0.997; 0.996; 0.995];

V1 = V;

%randomize output bias V0 = zeros(nout,1)

for j = 1:1:nout

V0(j, 1) = rand;

end

% V0=0.99;

V01 = V0;

%create Error vector E = zeros(nout, #tests

E=zeros(nout,t);

%dE = dE/dY = zeros(#out, training size)

dE = zeros(nout, t);

Eavg = zeros(nepoch, 1);

%\*\*\*\*\*\*\*\*Epochs

%nepoch = 10000;

for epoch = 1:1:nepoch

for test = 1:t;

%+++++++++++ Feed-forward

%calculate gamma vector

%per testNUMBER test

%per x(i, test)

%for test = 1:1:t --> run through all test cases

for j = 1:nhid

g(j,1) = 0;

for i = 1:nin+1;

if i >nin;

g(j,1) = g(j,1) + u(i,j);

else

g(j,1) = g(j,1) + u(i,j)\*x(i,test);

end

end

end

%calculate Z vector Z(H,1)

for j = 1:1:nhid;

Z(j,1)=1/(1+exp(-g(j,1)));

end

%calculate output vector y(j,test) = V0(j,1)+Z(j)\*V(j)

y(1,test) = V0;

for j = 1:1:nhid

y(1,test) = y(1,test)+Z(j)\*V(j);

end

%+++++++++++END feed-forward

%Error Calculations 'E' = zeros(1, test)

%columns = number of training data

%e1=0.5\*((y-d(1,test))^2);

%E(1, test)=0.5\*((y(1,test)-d(1,test))^2);

%dE/dY = (y-d);

for i = 1:nout

dE(nout, test) = (y(1,test)-d(1,test));

end

%====== Back Propagation

%find dV and dV0

for j=1:nhid

dV(j) = Z(j);

end

for j = 1:nout

dV0(j) = 1;

end

%find du = zeros(nin+1, nhid);

for j=1:(nin+1)

for i=nhid

if j >nin %hidden bias

du(j, i) = V(i)\*Z(i)\*(1-Z(i));

else %hidden link (non-bias)

du(j,i) = V(i)\*Z(i)\*(1-Z(i))\*x(j,t);

end

end

end

%update weights

%w=w(-eta\*dE/dW); eta = 0.1

%eta = 0.1;

u=u-eta\*(dE(nout,test))\*du;

V=V-eta\*(dE(nout,test))\*dV;

V0=V0-eta\*(dE(nout,test))\*dV0;

end

E = abs((y-d)./d);

Eavg(epoch)= sum(E)/numel(E);

end

u1

u

V1

V

V01

V0

x

d

y

Eavg(nepoch)

plot(1:1:nepoch, Eavg);

xlabel('epoch');

ylabel('Error');

end