**113EC0199 SAGNIK BASU**

### Channel Equalization using MLP

**Matlab Code :**

**clc;**

clear all;

close all;

c=input('Channel order');

experiments =50.0;

%noise=2\*rand(1,samples)-1; %%noise(bias)

%%channel 1

%y1=inp+noise;

samples=1000;

x=2\*rand(1,samples)-1;

inp=zeros(1,samples);

inp=zeros(1,samples);

for i=1:length(x) %%generation of inputs

if(x(i)<0)

inp(i)=-1;

else if(x(i)>0)

inp(i)=1;

else

inp(i)=0;

end

end

end

SNR=5;

%y1=awgn(inp,SNR);

y2=[inp(2:length(inp)) inp(1)];

y4=[inp(3:length(inp)) inp(1:2)];

y3=(0.364\*inp)+(0.86\*y2)+(0.364\*y4);

r=awgn(y3,SNR);

%weights=2\*(rand(1,c))-1;

%bias=2\*rand(1,1)-1; %%bias for the perceptron

weights=-1+2.\*rand(2,2\*c);

weights\_b= -1+2.\*rand(1,2\*c);

weightsb\_out= -1+2.\*rand(1,1);

bias = [-1 -1 -1 -1 ];

%iterations = constant;

coeff=0.5;

y=zeros(1,samples);

output=zeros(1,samples);

error=zeros(1,samples);

err\_train=zeros(1,samples);

%%final\_err\_train=0;

for j=1:samples-c

%% input1(:,j)=input(:,r);

% y(1,j)=y1(1,j:j+c-1)\*(transpose(weights))+bias;

%%out(1,j) = (1/(1+exp(-y(1,j))));

%%e=d\_out(r)-out(j);

%output(1,j)=hardlims(y(1,j));

%MSE Calculation for 50 experiments

final\_err\_mse2=0;

H1 = bias(1,1)\*weights\_b(1,1)+r(1,j)\*weights(1,1)+ r(1,j+1)\*weights(1,2);

% Send data through sigmoid function 1/1+e^-x

% Note that sigma is a different m file

% that I created to run this operation

x2(1) = tanh(H1);

H2 = bias(1,2)\*weights\_b(1,2) + r(1,j)\*weights(1,3) + r(1,j+1)\*weights(1,4);

x2(2) = tanh(H2);

%H3 = bias(1,3)\*weights\_b(1,3) + x(i)\*weights(1,5) + y(i)\*weights(1,6);

%x2(3) = tanh(H3);

%H4 = bias(1,4)\*weights\_b(1,4) + x(i)\*weights(1,7) + y(i)\*weights(1,8);

%x2(4) = tanh(H4);

%H2 = bias(1,4)\*weights(1,4) + x(i)\*weights(2,2) + y(i)\*weights(2,3);

%x2(3) = tanh(H2);

% Output layer

x3\_1 = bias(1,4)\*weightsb\_out(1,1)+ x2(1)\*weights(2,1)+x2(2)\*weights(2,2);%+x2(3)\*weights(2,3);%+x2(4)\*weights(2,4);

out(j) =tanh(x3\_1);

for k=1:experiments

y\_mse2(1,k)=bias(1,4)\*weightsb\_out(1,1)+ x2(1)\*weights(2,1)+x2(2)\*weights(2,2);

output\_mse2(1,k)=tanh(y\_mse2(1,k));

error\_mse2(1,k)=inp(1,k)-output\_mse2(1,k);

final\_err\_mse2=final\_err\_mse2+error\_mse2(1,k)\*error\_mse2(1,k);

end

mse\_2(j)=final\_err\_mse2/experiments;

delta3\_1 = (1-out(j)\*out(j))\*(inp(j)-out(j));

%delata3\_1=(output(i)-out(i));

% Propagate the delta backwards into hidden layers

%delta2\_1 = x2(1)\*(1-x2(1))\*weights(3,2)\*delta3\_1;

%delta2\_2 = x2(2)\*(1-x2(2))\*weights(3,3)\*delta3\_1;

delta2\_1 = (1-x2(1)\*x2(1))\*weights(2,1)\*delta3\_1;

delta2\_2 = (1-x2(2)\*x2(1))\*weights(2,2)\*delta3\_1;

% delta2\_3 = (1-x2(2)\*x2(2))\*weights(2,3)\*delta3\_1;

weights\_b(1,1) = weights\_b(1,1) + coeff\*bias(1,1)\*delta2\_1;

weights\_b(1,2) = weights\_b(1,2) + coeff\*bias(1,2)\*delta2\_2;

%weights\_b(1,3) = weights\_b(1,3) + coeff\*bias(1,3)\*delta2\_3;

%weightsb(1,4) = weightsb(1,4) + coeff\*bias(1,4)\*delta2\_4;

weightsb\_out = weightsb\_out + coeff\*bias(1,4)\*delta3\_1;

weights(1,1) = weights(1,1) + coeff\*r(1,j)\*delta2\_1;

weights(1,2) = weights(1,2) + coeff\*r(1,j+1)\*delta2\_1;

weights(1,3) = weights(1,3) + coeff\*r(1,j)\*delta2\_2;

weights(1,4) = weights(1,4) + coeff\*r(1,j+1)\*delta2\_2;

weights(2,1) = weights(2,1) + coeff\*x2(1)\*delta3\_1;

weights(2,2) = weights(2,2) + coeff\*x2(2)\*delta3\_1;

%%training

%error(1,j)=inp(1,j)-output(1,j);

%%err\_train(j)=error(1,j)\*error(1,j);

%bias=bias+error(1,j);

%weights=weights+error(1,j)\*y1(1,k:k+c-1);

%weights\_array(j,:)=weights;

%%weights\_final(j,:,k)= weights;

%%bias\_final(j,1,k) = bias;

%%error(j,1,k) = e;

end

%%Testing

testing\_size=1000;

y\_test=2\*rand(1,testing\_size)-1;

input=zeros(1,50);

final\_err=0;

mse=zeros(1,50);

SNR=1;

for k=1:100

y\_test=2\*rand(1,testing\_size)-1;

input=zeros(1,testing\_size);

for i=1:length(y\_test) %%generation of inputs

if(y\_test(i)<0)

input(i)=-1;

else if(y\_test(i)>0)

input(i)=1;

else

input(i)=0;

end

end

end

final\_err=0;

SNR\_arr(k)=SNR+k/10;

y1=awgn(input,SNR\_arr(k));

BER=0;

for i=1:testing\_size-c+1

% y1\_test(1,i)=y1(1,i:i+c-1)\*(transpose(weights))+bias;

% percp\_out(1,i)=hardlims(y1\_test(1,i));

% error\_test(i)=percp\_out(1,i)-input(1,i);

H1 = bias(1,1)\*weights\_b(1,1)+y(1,j)\*weights(1,1)+ y(1,j+1)\*weights(1,2);

% Send data through sigmoid function 1/1+e^-x

% Note that sigma is a different m file

% that I created to run this operation

x2(1) = tanh(H1);

H2 = bias(1,2)\*weights\_b(1,2) + y(1,j)\*weights(1,3) + y(1,j+1)\*weights(1,4);

x2(2) = tanh(H2);

x3\_1 = bias(1,4)\*weightsb\_out(1,1)+ x2(1)\*weights(2,1)+x2(2)\*weights(2,2);%+x2(3)\*weights(2,3);%+x2(4)\*weights(2,4);

out(i) = hardlims(tanh(x3\_1));

error\_test(i)=out(1,i)-input(1,i);

if(error\_test(i)==0)

else

BER=BER+1;

end

end

%final\_err=final\_err+error\_test(i)\*error\_test(i);

BER\_arr(k)=BER/1000;

end

%mse(k)=final\_err/1000.0;

%axis([-3 3 -3 3]);

%w1=-bias/weights(1,1);

%w2=-bias/weights(1,2);

%plot([w1,0],[0,w2]);

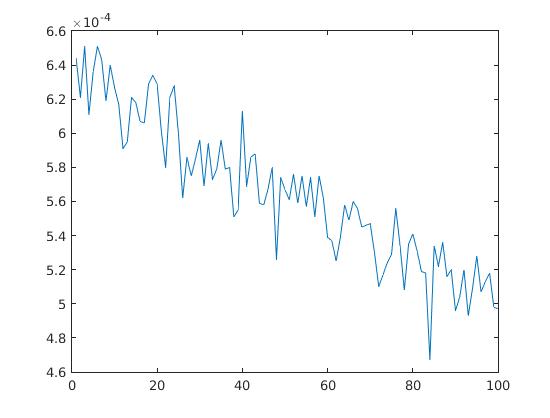
%hold on;

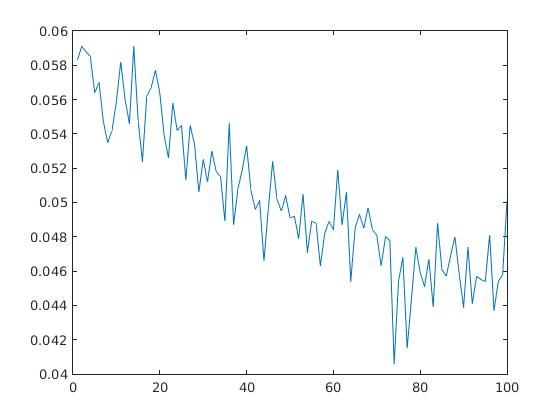
%plotpv(inp,inp);

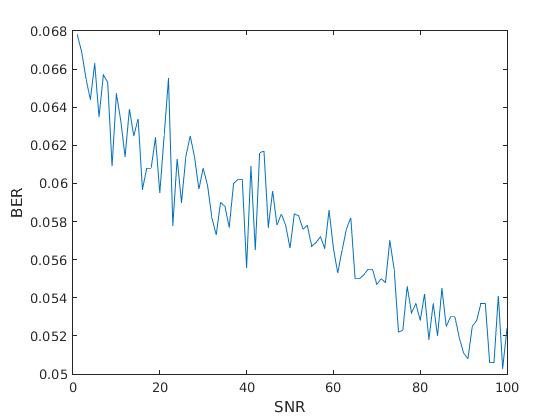
%hold on;

%plotpc(weights,bias);

**BER Curves for delay elements(-2/-3/-4)**

****

****

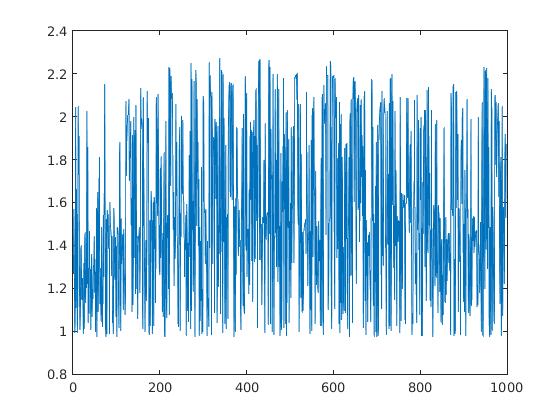
****

**MSE curves for different channel models using 2 hidden layers**

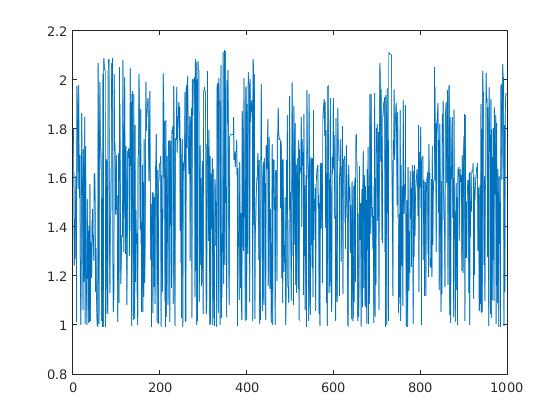
**a) y(n) = x(n) + 0.5x(n − 1) + N(n)**

**b) y(n) = 0.5x(n) + x(n − 1) + N(n)**

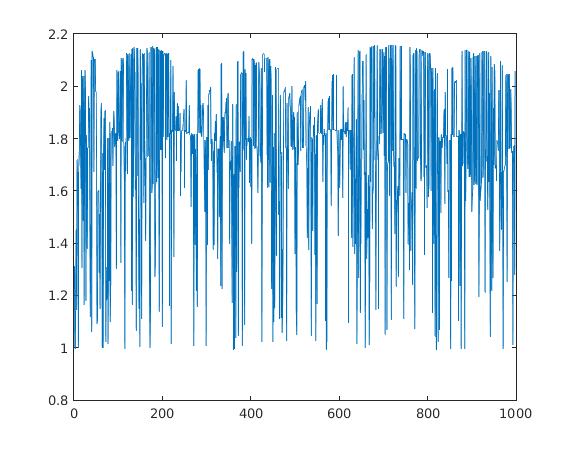
**c) y(n) = 0.364x(n) + 0.86x(n − 1) + 0.364x(n − 2)**

****

**a)**

****

**b)**

****

**c)**