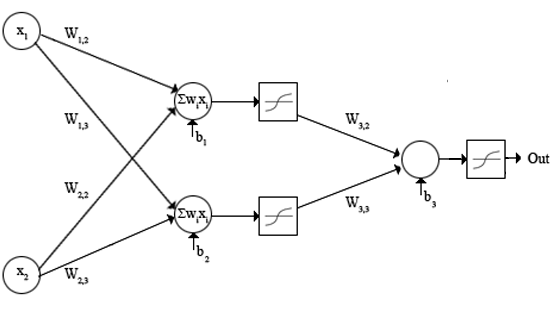
**SAGNIK BASU 113EC0199**

### **Application of MLP in Function approximation**

Design a MLP to approximate the function z=sin(pi\*x).Cos(pi\*y) where -0.5 <= x, y <= 0.5. The MLP consist of 2 inputs and one output. The network contains 2layes of neurons. One input cum hidden layer with multiple neurons. The output layer will have only one neuron. The hidden layer should be varied from 2 neurons to 5 neurons. Train the weights and threshold using back propagation algorithm. Consider a suitable activation function for each layer. Generate a training data table consisting of 900 samples spread across the input space. In a plot, show the training data points. After training, determine the mean square error (MSE) of the network for the network output over 500 samples which are different fro the input data set. Show the change in error when the network size changes from 2 to 5 neurons in hidden layer. Plot the output in a 3-D plot for each of the network considered and comment of the performance of each of the networks.

**Structure of Perceptron :**

****

**Matlab Code :**

clc;

clear all;

close all;

% Function Approximation using perceptron

%

% Function = sin(x)\*cos(y)

%

constant=1000;

x=rand(1,constant)-0.5; %training table

y=rand(1,constant)-0.5;

inp=zeros(constant,2);

for i=1:length(x)

inp(i,1)=x(i);

inp(i,2)=y(i);

end

thr=0.000001;

for j=1:20

var1=2\*rand(1,1)-1;

var2=2\*rand(1,1)-1;

w=[var1 var2];

output=sin(pi\*x).\*cos(pi\*y); %desired output

output\_plot=(sin(pi\*x)')\*(cos(pi\*y));

%=rand(1,1);

e=100000000;

weights = 1 +2.\*rand(3,3);

input\_layer = 4;

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

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

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

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

iterations = constant;

coeff=0.5;

%weight1=zeros(1,constant);

%weight2=zeros(1,constant);

%bias1=zeros(1,constant);

out=zeros(1,constant);

for i = 1:iterations

%out = zeros(4,1);

% numIn = length (input(:,1));

%for j = 1:numIn

% Hidden layer

H1 = bias(1,1)\*weights\_b(1,1)+x(i)\*weights(1,1)+ y(i)\*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) + x(i)\*weights(1,3) + y(i)\*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(i) = tanh(x3\_1);

error\_out=0;

mse\_iter=50;

for l=1:mse\_iter

% error\_in=0;

out\_mse=tanh(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));

error\_in=output(l)-out\_mse;

%1+2.\*rand(1,input\_layer)

error\_out=error\_out+error\_in\*error\_in;

end

mseo(i)=error\_out/mse\_iter;

% Adjust delta values of weights

% For output layer:

% delta(wi) = xi\*delta,

% delta = (1-actual output)\*(desired output - actual output)

delta3\_1 = (1-out(i)\*out(i))\*(output(i)-out(i));

%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(2))\*weights(2,2)\*delta3\_1;

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

delta2\_4 = (1-x2(2)\*x2(2))\*weights(2,4)\*delta3\_1;

%delta2\_5 = (1-x2(2)\*x2(2))\*weights(3,3)\*delta3\_1;

% Add weight changes to original weights

% And use the new weights to repeat process.

% delta weight = coeff\*x\*delta

% for k = 1:3

% if k == 1 % Bias cases

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;

% else % When k=2 or 3 input cases to neurons for k=1:4

weights(1,1) = weights(1,1) + coeff\*x(i)\*delta2\_1;

weights(1,2) = weights(1,2) + coeff\*y(i)\*delta2\_1;

weights(1,3) = weights(1,3) + coeff\*x(i)\*delta2\_2;

weights(1,4) = weights(1,4) + coeff\*y(i)\*delta2\_2;

weights(1,5) = weights(1,5) + coeff\*x(i)\*delta2\_3;

weights(1,6) = weights(1,6) + coeff\*y(i)\*delta2\_3;

weights(1,7) = weights(1,7) + coeff\*x(i)\*delta2\_4;

weights(1,8) = weights(1,8) + coeff\*y(i)\*delta2\_4;

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

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

weights(2,3) = weights(2,3) + coeff\*x2(3)\*delta3\_1;

weights(2,4) = weights(2,4) + coeff\*x2(4)\*delta3\_1;

%end

%weights21= weights21 + coeff\*x2(1)\*delta3\_1;

%weights22 = weights22 + coeff\*x2(2)\*delta3\_1;

%weights23= weights23 + coeff\*x2(3)\*delta3\_1;

end

end

x\_test=rand(1,constant)-0.5; %training table

y\_test=rand(1,constant)-0.5;

for i=1:1000

% test(i)= tanh(bias(1,3)\*weights(3,1)+x\_test(i)\*weights(3,2)+ y\_test(i)\*weights(3,3));

H1 = bias(1,1)\*weights\_b(1,1)+x\_test(i)\*weights(1,1)+ y\_test(i)\*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) + x\_test(i)\*weights(1,3) + y\_test(i)\*weights(1,4);

x2(2) = tanh(H2);

H3 = bias(1,3)\*weights\_b(1,3) + x\_test(i)\*weights(1,5) + y\_test(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\_test(i) = tanh(x3\_1);

end

z\_test=[x\_test ;y\_test ;out\_test];

z=[x;y;output];

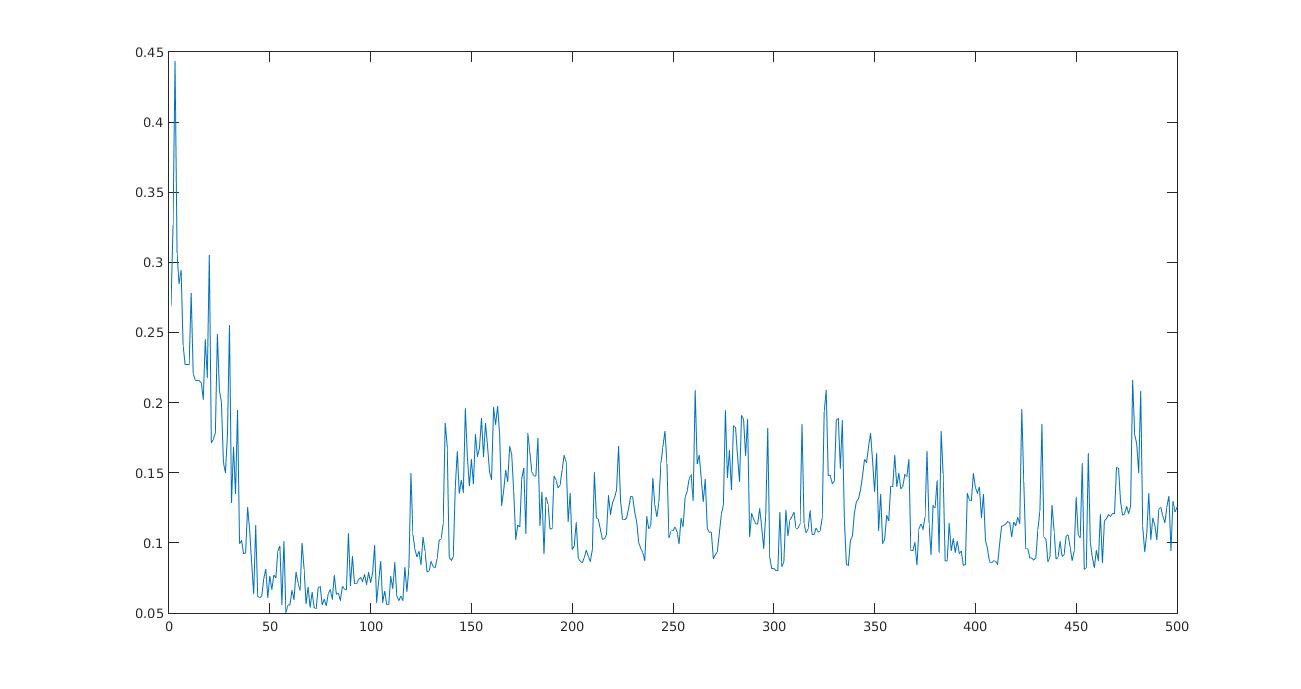
scatter3(x\_test,y\_test,out\_test);

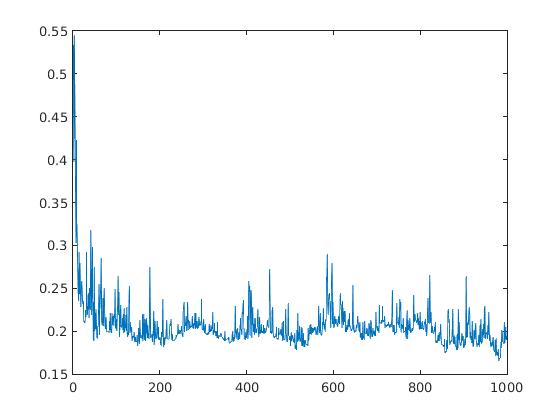
hold on;

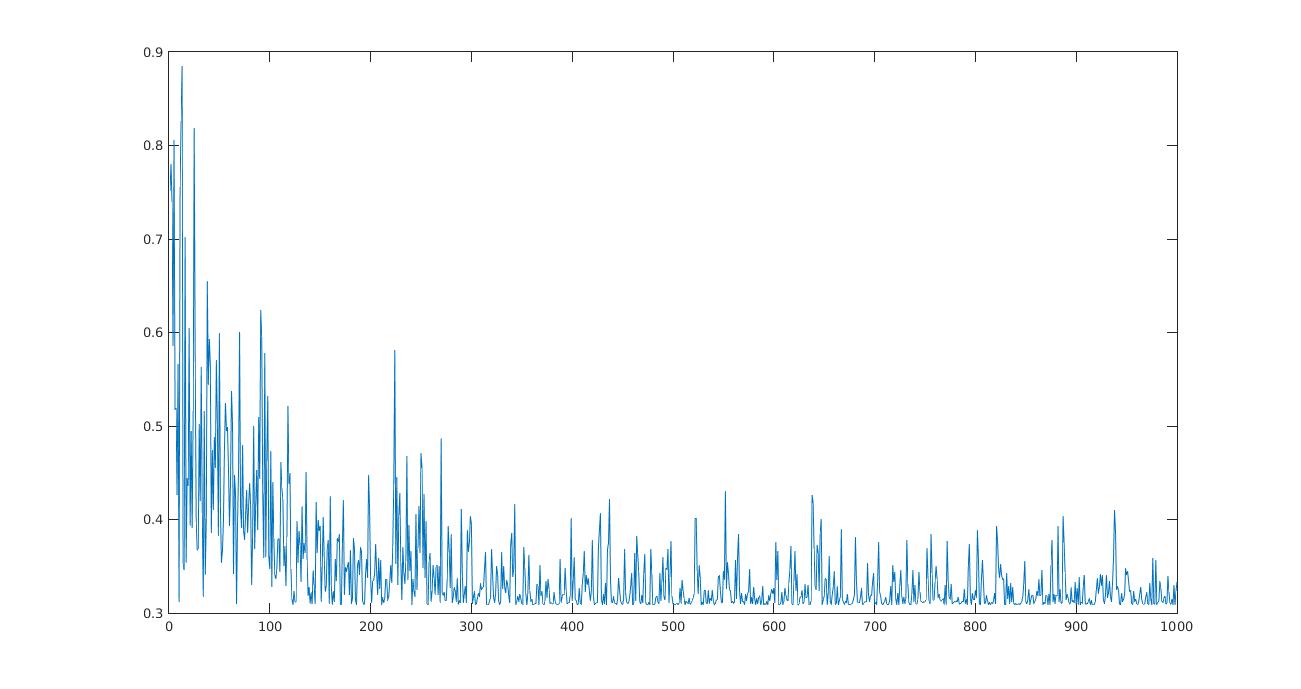
scatter3(x,y,output);

**Output :**

1) MSE for a)2 b)3 c) 4 input neurons



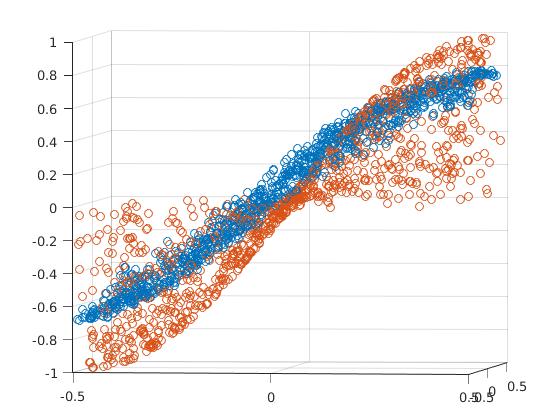


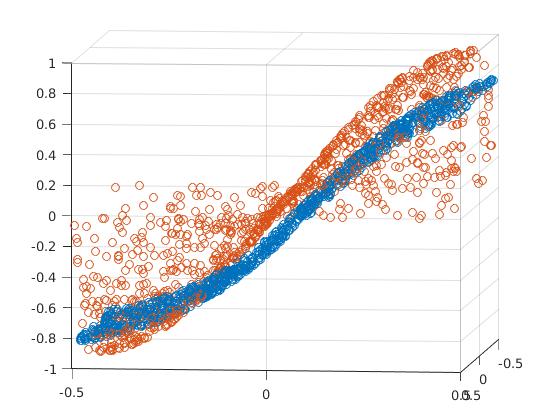


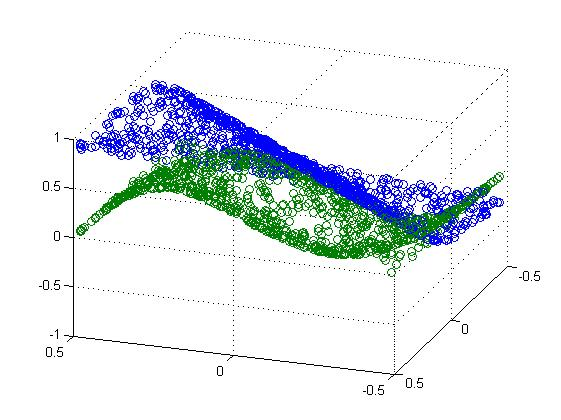
2) 3 D scatter plot for a)2 input b)3 input c)4 input neuron

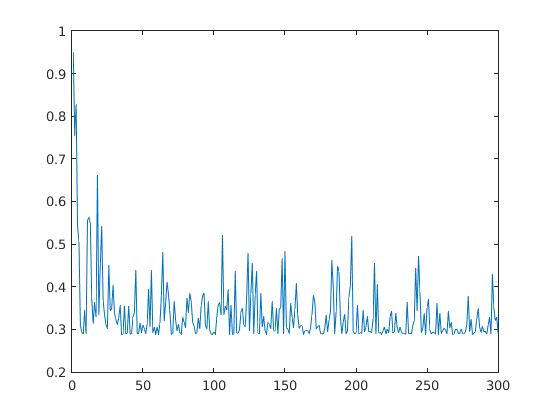
Here blue = desired function

red = approx. function

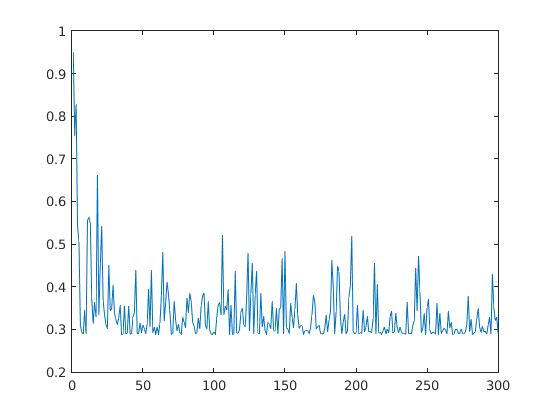






**Epoch based training **

1) MSE curve



2)3D plot

