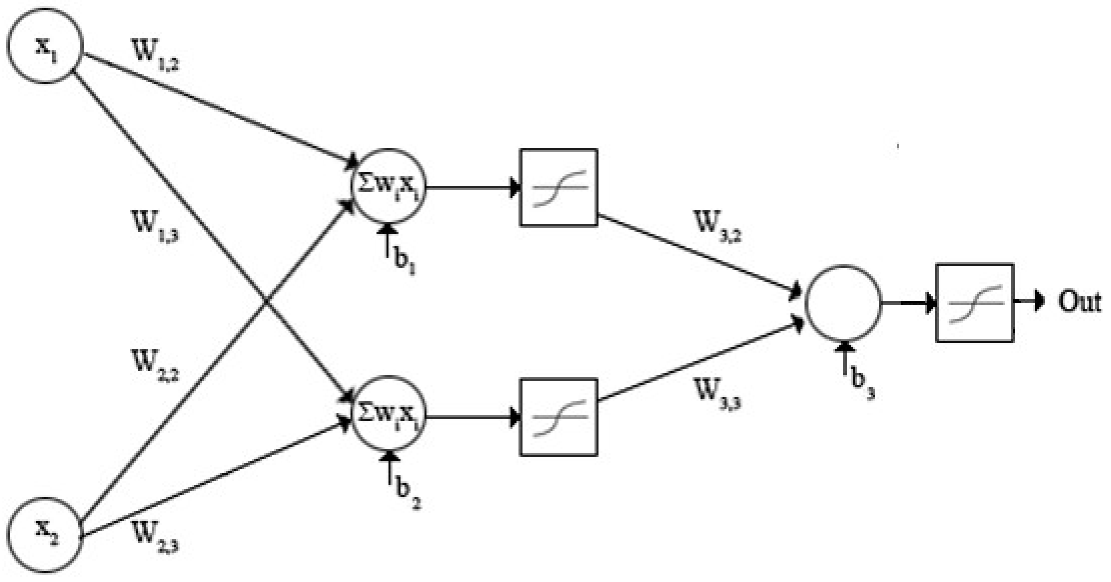
**SAGNIK BASU 113EC0199**

## 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;
```

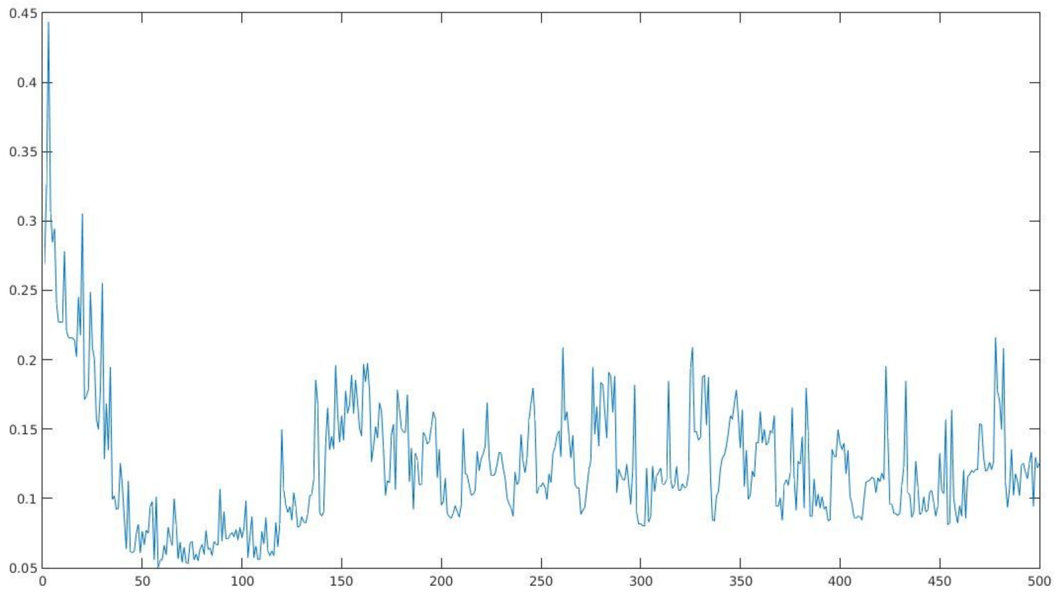
```
weights=-1+2.*rand(2,2*input_layer);
weights_b= -1+2.*rand(1,input_layer);
weightsb_out= -1+2.*rand(1,1);

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;
```

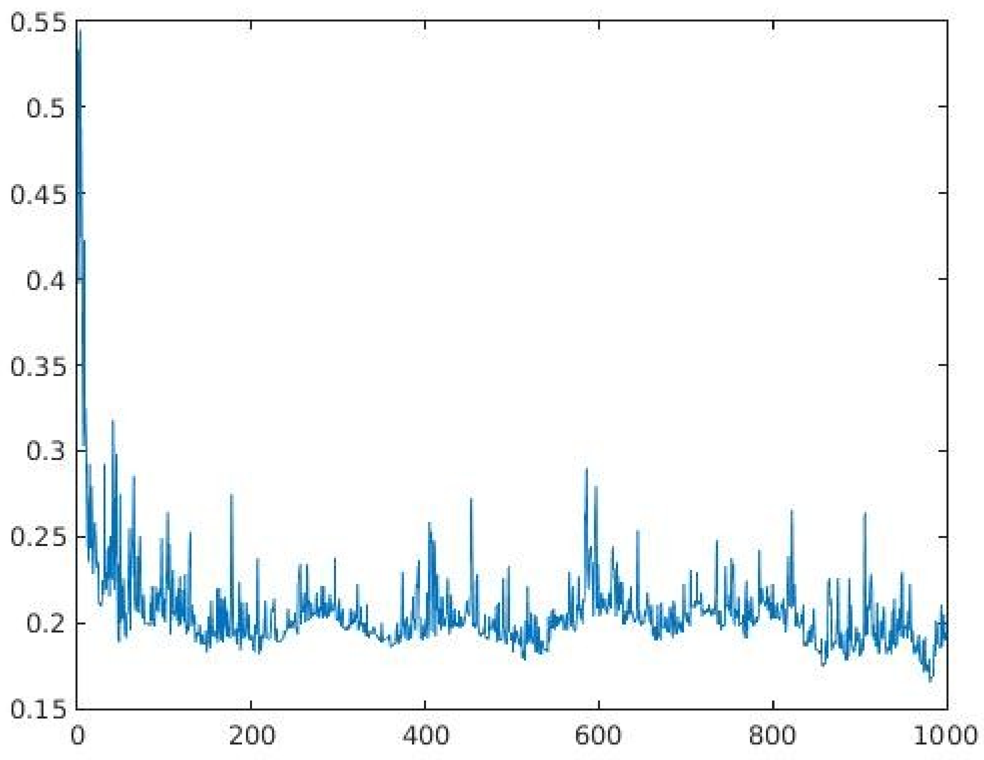
```
        % Adjust delta values of weights
        % For output layer:
        % delta(wi) = xi*delta,
        % delta = (1-actual output)*(desired output - actual output)
        delta3_1 = (1-out(i)*out(i))*(output(i)-out(i));
        %delata3_1=(output(i)-out(i));
        % Propagate the delta backwards 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;
```

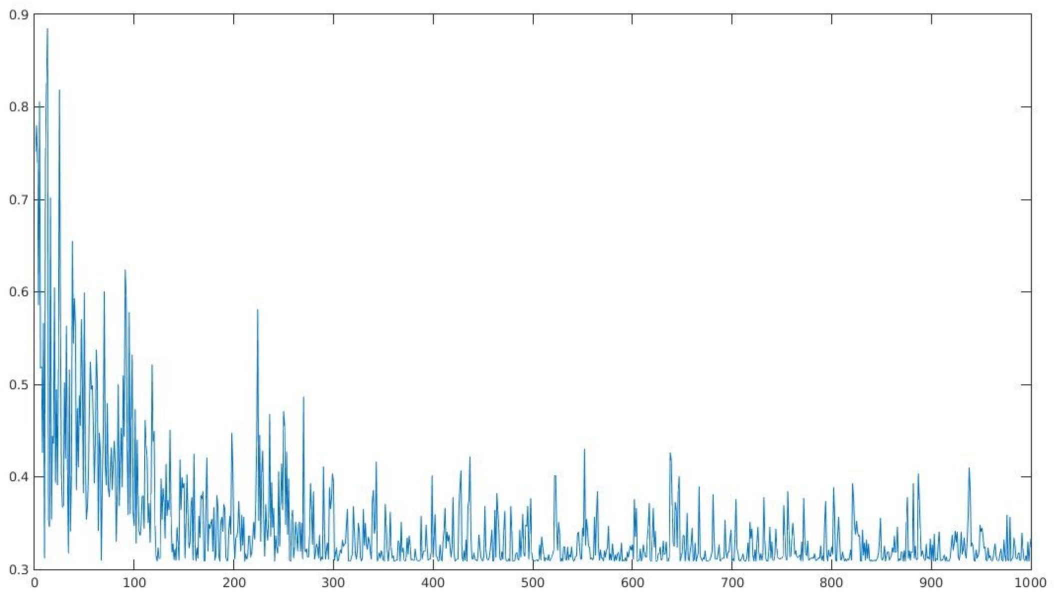
```
for i=1:1000

     % 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

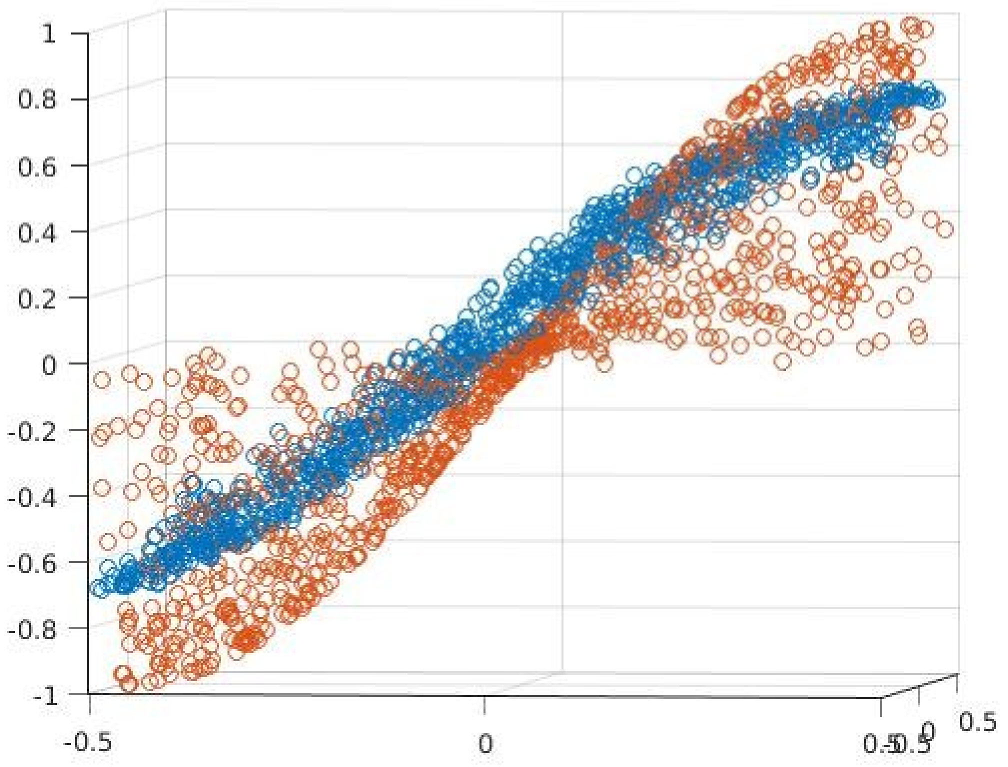

0.45
0.4
0.35
0.3
0.25
0.2
0.15
0.1
0.05
0
50
100
150
200
250
300
350
400
450
500

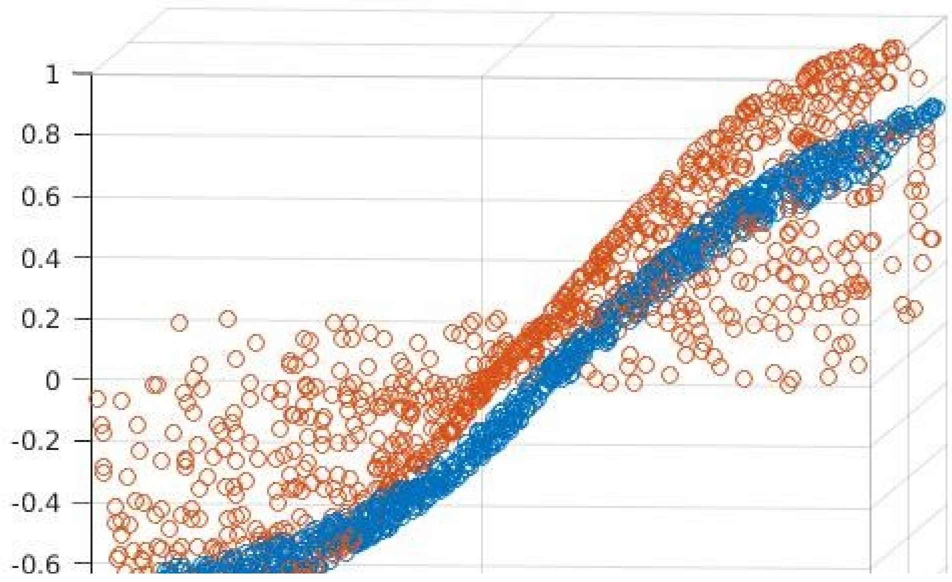


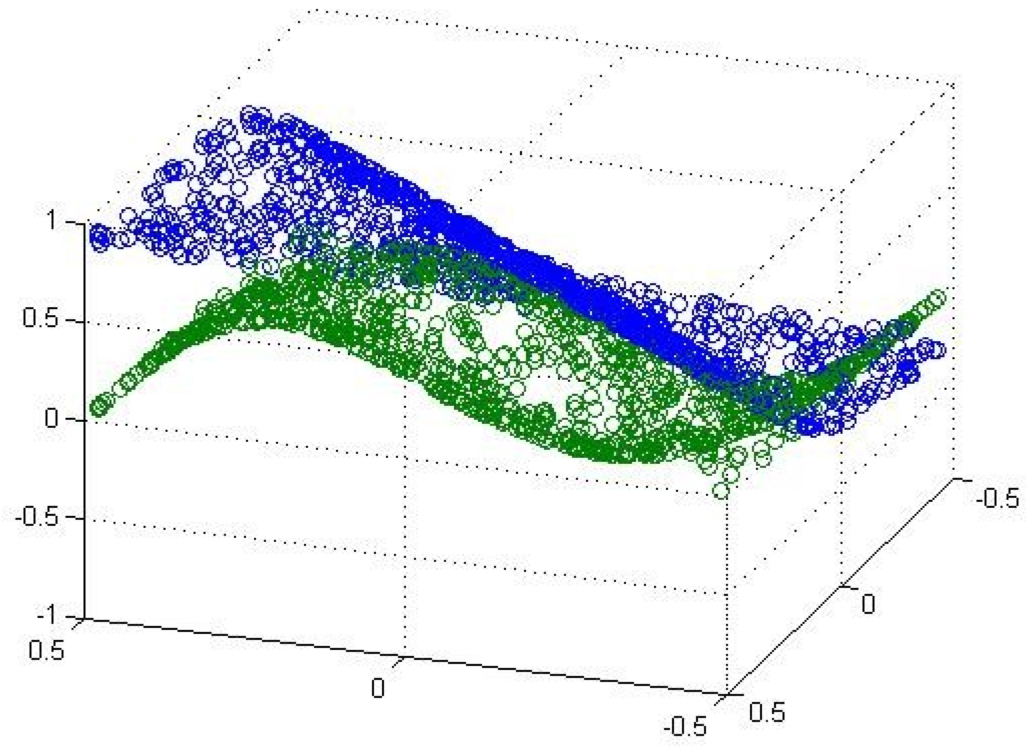


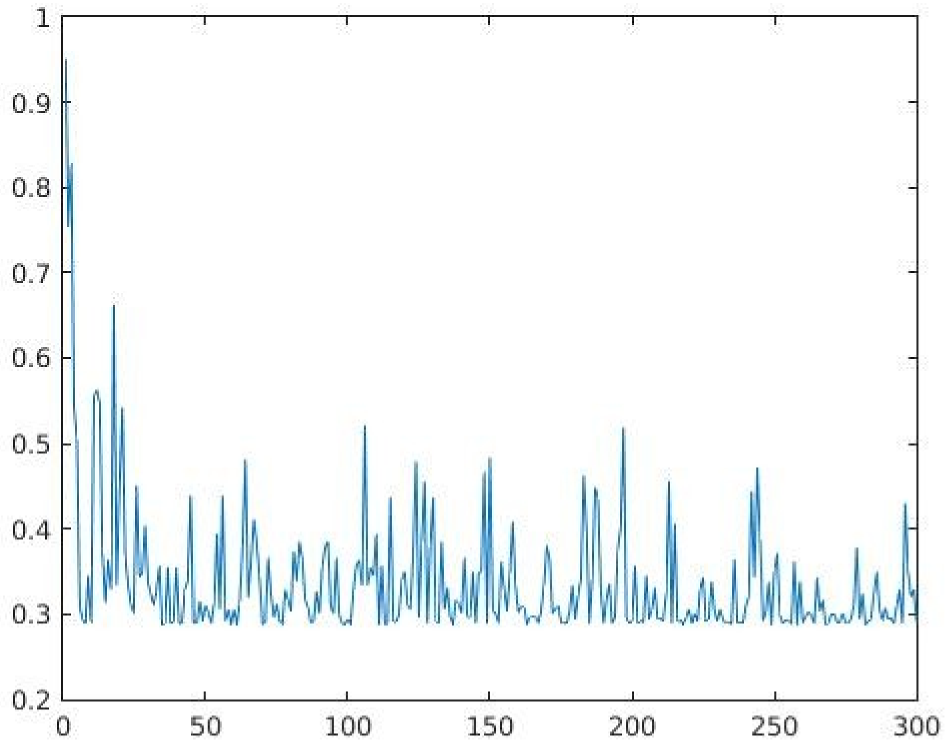
2) 3 D scatter plot for a)2 input b)3 input c)4 input neuron

Here blue = desired function
red = approx. function



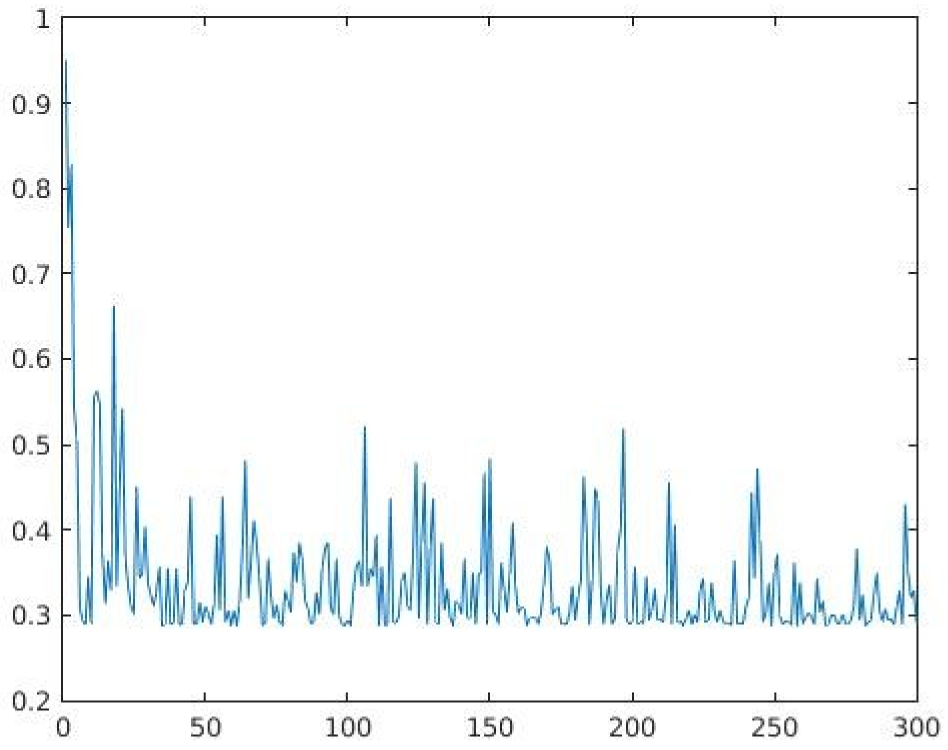

1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6


1
0.5
0
-0.5
-1
0.5
0
-0.5
0.5
0
-0.5

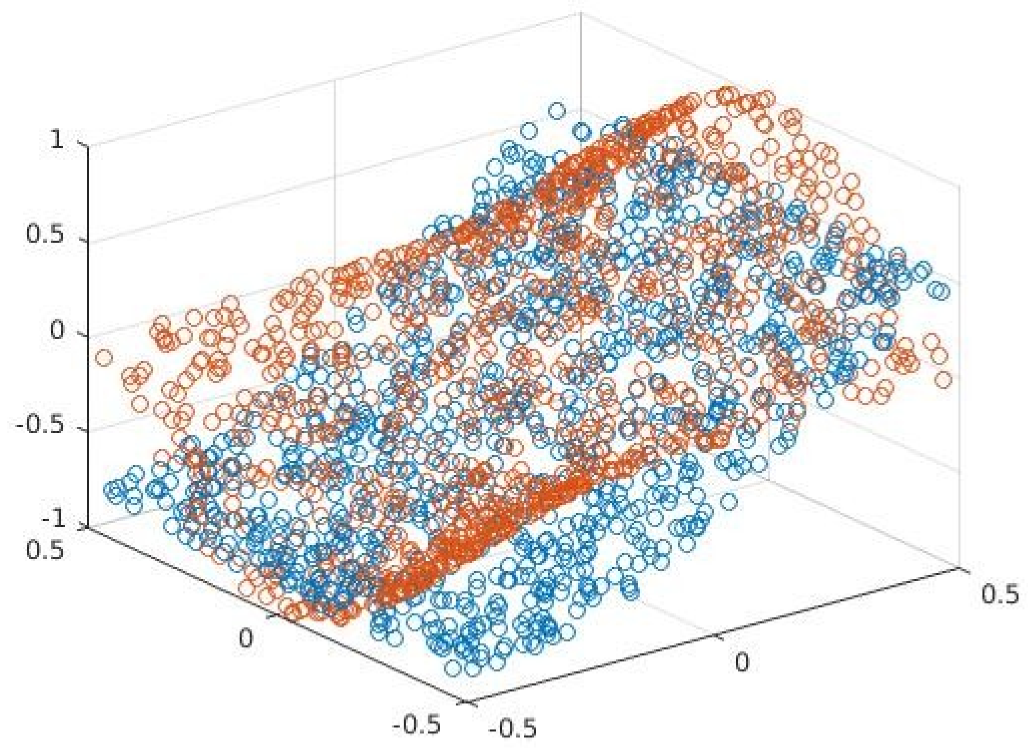


## Epoch based training

1) MSE curve



2)3D plot



1
0.5
0
-0.5
-1
0.5
0
-0.5
-0.5
0
0.5