SAGNIK BASU 113EC0199

Simulate a 2input XOR gate using MLP with back-propagation algorithm. Use a MLP with 2 neural layers. The second layer should have only one neuron. Tune the number of neurons in the first layer starting with 2neurons to achieve the result. Repeat the experiment for 3input XOR gate also.

MATLAB CODE :

1) For two input xor gate

```
% multi layer perceptron

% XOR input for x1 and x2
input = [0 0 ; 0 1 ; 1 0 ; 1 1 ];
% Desired output of XOR
output = [0;1;1;0];
% Initialize the bias
bias = [-1 -1 -1];
% Learning coefficient
coeff = 0.7;
% Number of learning iterations
iterations = 1000;
% Calculate weights randomly using seed.
%rand('state',sum(100*clock));

weights = 5 +2.*rand(3,3);



for i = 1:iterations
   out = zeros(4,1);
   numIn = length (input(:,1));
   for j = 1:numIn
```

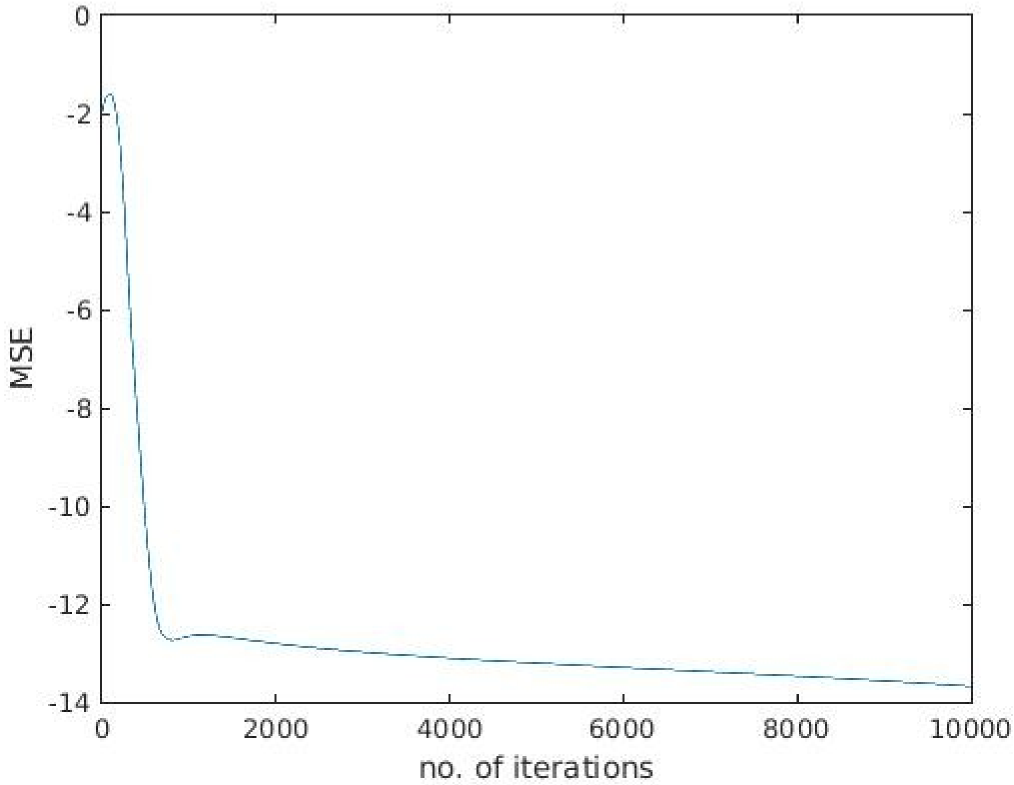
```
    % Hidden layer
    H1 = bias(1,1)*weights(1,1)+input(j,1)*weights(1,2)+ input(j,2)*weights(1,3);

    % 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) = sigma(H1);
    H2 = bias(1,2)*weights(2,1) + input(j,1)*weights(2,2) + input(j,2)*weights(2,3);
    x2(2) = sigma(H2);

    % Output layer
    x3_1 = bias(1,3)*weights(3,1)+ x2(1)*weights(3,2)+x2(2)*weights(3,3);
    out(j) = sigma(x3_1);
    error_out=0;
    mse_iter=50;
    for l=1:mse_iter
       error_in=0;
       for j2=1:numIn
          out_mse=sigma(bias(1,1)*weights(1,1)+input(j2,1)*weights(1,2)+
input(j2,2)*weights(1,3));
          error_in=error_in+output(j2)-out_mse;
       end
       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 = out(j)*(1-out(j))*(output(j)-out(j));

    % 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;

    % 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(1,k) = weights(1,k) + coeff*bias(1,1)*delta2_1;
        weights(2,k) = weights(2,k) + coeff*bias(1,2)*delta2_2;
        weights(3,k) = weights(3,k) + coeff*bias(1,3)*delta3_1;
      else % When k=2 or 3 input cases to neurons
        weights(1,k) = weights(1,k) + coeff*input(j,1)*delta2_1;
        weights(2,k) = weights(2,k) + coeff*input(j,2)*delta2_2;
        weights(3,k) = weights(3,k) + coeff*x2(k-1)*delta3_1;
      end
    end
  end
end


%% Plotting in matlab


p = [0 0 1 1; 0 1 0 1];
t = [0 1 1 0];
%plotpv(p,t);
%plotpc(weights,bias)
```

2) Three input xor gate

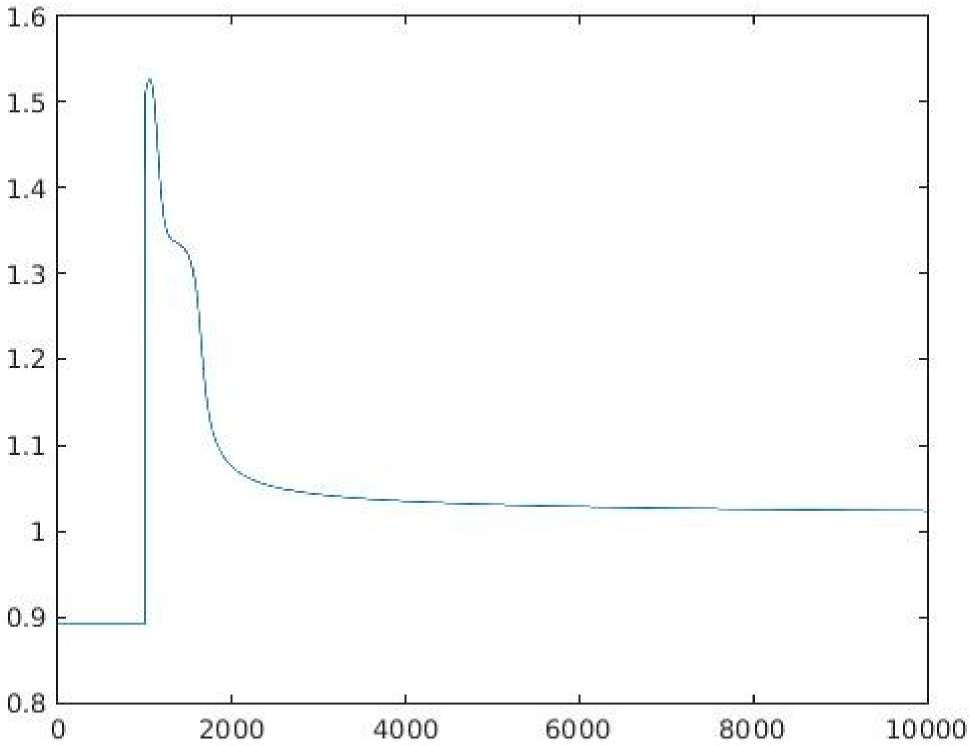
```
% multi layer perceptron

% XOR input for x1 and x2
input = [0 0 0; 0 1 0; 1 0 0; 1 1 0;1 0 1;1 1 1;0 1 1;0 0 1];
% Desired output of XOR
output = [0;1;1;0;0;1;0;1];
% Initialize the bias
bias = [-1 -1 -1 -1];
% Learning coefficient
coeff = 0.7;
% Number of learning iterations
iterations = 2000;
% Calculate weights randomly using seed.
%rand('state',sum(100*clock));

weightsb = -1 +2.*rand(1,4);
weights11= -1 +2.*rand(1,4);
weights12= -1 +2.*rand(1,4);
weights13= -1 +2.*rand(1,4);
weights21=  -1 +2.*rand(1,1);
weights22= -1 +2.*rand(1,1);
weights23= -1 +2.*rand(1,1);




for i = 1:iterations
   out = zeros(4,1);
   numIn = length (input(:,1));
   for j = 1:numIn
      % Hidden layer
      H1 = bias(1,1)*weightsb(1,1)+input(j,1)*weights11(1,1)+
input(j,2)*weights11(1,2)+input(j,3)*weights11(1,3);

      % 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) = sigma(H1);
      H2 = bias(1,2)*weightsb(1,2) + input(j,1)*weights12(1,1) +
input(j,2)*weights12(1,2)+input(j,3)*weights12(1,3);
      x2(2) = sigma(H2);

      H3 = bias(1,3)*weightsb(1,3) + input(j,1)*weights13(1,1) +
input(j,2)*weights13(1,2)+input(j,3)*weights13(1,3);
      x2(3) = sigma(H3);
```

```
    % Output layer
    x3_1 = bias(1,4)*weightsb(1,4)+ x2(1)*weights21+x2(2)*weights22+x2(3)*weights23;
    out(j) = sigma(x3_1);
    error_out=0;
    mse_iter=50;
    %for l=1:mse_iter
       test=xor_test(50);
       for l=1:50
         error_in=0;

          out_mse=sigma(bias(1,3)*weightsb(1,4)+test(l,1)*weights21+
test(l,2)*weights22+test(l,3)*weights23);
          error_in=error_in+output(j2)-out_mse;
       %end
       error_out=error_out+error_in*error_in;
    end
    %mse(i)=error_out/8;

    % Adjust delta values of weights
    % For output layer:
    % delta(wi) = xi*delta,
    % delta = (1-actual output)*(desired output - actual output)
    delta3_1 = out(j)*(1-out(j))*(output(j)-out(j));

    % Propagate the delta backwards into hidden layers
    delta2_1 = x2(1)*(1-x2(1))*weights21*delta3_1;
    delta2_2 = x2(2)*(1-x2(2))*weights22*delta3_1;
    delta2_3 = x2(3)*(1-x2(3))*weights23*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
         weightsb(1,1) = weightsb(1,1) + coeff*bias(1,1)*delta2_1;
         weightsb(1,2) = weightsb(1,2) + coeff*bias(1,2)*delta2_2;
         weightsb(1,3) = weightsb(1,3) + coeff*bias(1,3)*delta2_3;
         weightsb(1,4) = weightsb(1,4) + coeff*bias(1,4)*delta3_1;
     %  else % When k=2 or 3 input cases to neurons
         for k=1:4
         weights11(1,k) = weights11(1,k) + coeff*input(j,1)*delta2_1;
         weights12(1,k) = weights12(1,k) + coeff*input(j,2)*delta2_2;
         weights13(1,k) = weights13(1,k) + coeff*input(j,3)*delta2_3;
         end
         weights21=  weights21 + coeff*x2(1)*delta3_1;
         weights22 =  weights22 + coeff*x2(2)*delta3_1;
         weights23=  weights23 + coeff*x2(3)*delta3_1;

        % weights14(4,k) = weights11(3,k) + coeff*bias(1,3)*delta3_1;
      %end
   % end
  end

end
```

OUTPUT :



1) For two input xor gate



2) For three input xor gate