Jimma University

Jimma Institute of Technology

Faculty of Electrical and Computer Engineering

Masters of Science in Computer Engineering

Machine Learning Assignment

Back propagation Test using Matlab

Prepared by:

Misganaw Aguate

Submitted to: - Dr. Getachew A. (PHD)

We use back propagation to create model our solution for a given problem and make prediction.

To do this the following crucial step are necessary:

a. Initialize a random input signals and weights the normalize
b. Assign desired (actual) output
c. Assign learning rate alpha to modify weight and beta to apply sigmoid function
d. Apply feed forward propagation starting from input layer up to output layer
e. Find the error which is the difference of actual output value and predicted output value. If the difference is almost approach to zero so our weight is correct to train the neural network and finish our training, otherwise go to next step
f. Apply back propagation starting from output layer to input layer. This modifies the initial weight. The return to d to apply feed forward propagation by the updated weight.

1. Create a random input signals and weights, in our case

- [x1,x2,x3,x4,x5,x6,x7,x8,x9,x10,x11,12] which each has 5 input elements

```
x1=randi([-1 1],1,5);          %initialize  input x1
x2=randi([-2 4],1,5);          %initialize  input x2
x3=randi([-4 3],1,5);          %initialize  input x3
x4=randi([-1 1],1,5);          %initialize  input x4
x5=randi([-2 4],1,5);          %initialize  input x5
x6=randi([-1 1],1,5);          %initialize  input x6
x7=randi([-4 3],1,5);          %initialize  input x7
x8=randi([-2 4],1,5);          %initialize  input x8
x9=randi([-1 1],1,5);          %initialize  input x9
x10=randi([-4 3],1,5);         %initialize  input x10
x11=randi([-4 3],1,5);         %initialize  input x11
x12=randi([-1 1],1,5);         %initialize  input x12
xtt=[x1;x2;x3;x4;x5;x6;x7;x8;x9;x10;x11;x12;];  %put all input signal in one input vector
x=normalize(xtt,'range');                %normalize input vector
```

	Element 1	Element2	Element 3	Element 4	Element 5
X1	X11	X12	X13	X14	X15
X2	X21	X22	X23	X24	X25
X3	X31	X32	X33	X34	X35
X4	X41	X42	X43	X44	X45
X5	X51	X52	X53	X54	X55
X6	X61	X62	X63	X64	X65
X7	X71	X72	X73	X74	X75
X8	X81	X82	X83	X84	X85
X9	X91	X92	X93	X94	X95
X10	X101	X102	X103	X104	X105
X111	X111	X112	X113	X114	X115
X12	X121	X122	X133	X134	X135

- W1 which is feed to our input layer of neural network

```
for i=1:4
   for j=1:5
      w1t=rand(i,j);        %initialize weight for input x at layer1 (total 20 weigts)
      w1=normalize(w1t,'range');
   end
end
fprintf('The initial W1 \n');
disp(w1);
```

W1 at input signal of layer 1

	Column 1	Column 2	Column 3	Column 4	Column 5
Row 1	w11	w12	w13	w14	w15
Row 2	w21	w22	X23	w24	w25
Row 3	w31	w32	w33	w34	w35
Row 4	w41	w42	w43	w44	w45

- W2 which is feed to our output layer of neural network

```
for i=1:3
   for j=1:4
      w2t=rand(i,j);               %initialize weight at layer 2
      w2=normalize(w2t,'range');
   end
end
fprintf('The initial W2 \n');
disp(w2);
```

W2 at at input signal of layer 2

	Column 1	Column 2	Column 3	Column 4
Row 1	w11	w12	w13	w14
Row 2	w21	w22	X23	w24
Row 3	w31	w32	w33	w34

2. Assign desired (actual) output to evaluate the correctness of predicted output by making comparison.

```
dt=[-2.6 3.81 0.69];
d=normalize(dt,'range');          %initialize and normaliz dired output at output layer
```

3. Assign learning rat alpha (eta) and beta for updating weight and calculating activation function respectively

```
beta=1.5;  % to calculat activation function by applying sigmoid function
alpha=1;   % called as (miw)or learning rate
```

4. Apply feed forward propagation to calculate prediction

```
    for j=1:5
        w1t=rand(i,j);                      %initialize weight for input x
at layer1 (total 20 weigts)

        w1=normalize(w1t,'range');
    end
end
fprintf('The initial W1 \n');
disp(w1);
for i=1:3
    for j=1:4
        w2t=rand(i,j);                        %initialize weight at layer 2
        w2=normalize(w2t,'range');
    end
end
fprintf('The initial W2 \n');
disp(w2);

beta=1.5; % to calculat activation function by applying sigmoid
function
alpha=1;   % called as (miw)or learning rate
for z=1:3000
    for i=1:3
        for j=1:4
            for k=1:12
                xt=x(k,:)';
                for l=1:5
                    hp(l)=w1(j,l).*xt(l); %store weigt and  input
signal product in vector called 'hp'
                    % disp(hp(l));
                end
                hs(k)=sum(hp)+1;     %store the same of each 5 element
of each 12 input signal vector called 'hs' bay addind bias
```

```
            end
            u(j)=sum(hs)/12;      %store the total product summation of
each 12 input  signal in vector called 'u'

            exh(j)=u(j).*beta;
            v(j)=1/(1+2.72^-exh(j));   %store the cost function of
first layer in vector called 'v

            op(j)=w2(i,j).*v(j);  %store  activation function and
weight product for second layer in vector called 'op'

        end
        os(i)=sum(op)+1; %store sum product at second layer in vector
called 'os' by adding bias 1

        ex(i)=os(i).*beta;
        y(i)=1/(1+2.72^-ex(i));   %store final output in vector called
'y
        em=1/2;
        en=(d(i)-y(i)).^2;
        E(i)=em*en;             %store each corrsponding error in vector
called 'hp

        e(i)=-(d(i)-y(i));
        delta(i)=y(i).*(1-y(i)).*e(i); % delta=-(d-y)*y*(1-y)
    end
```

5. Apply back propagation to update weights

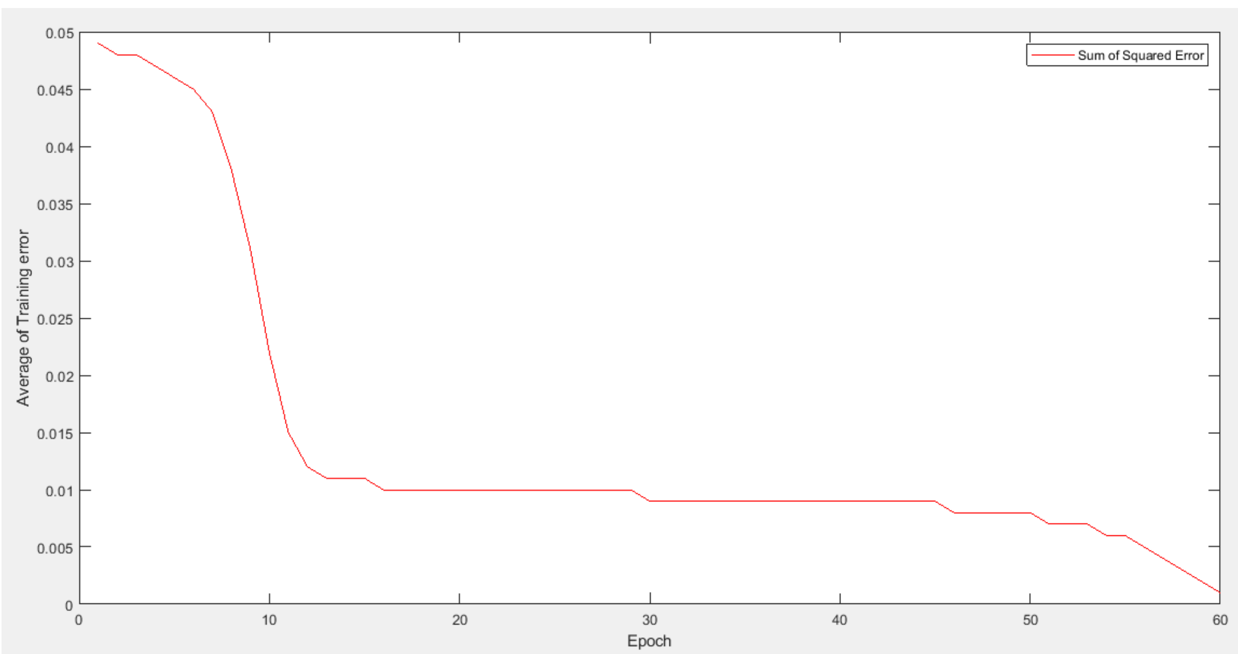
```
 if Etot(z)==0.000001
     fprintf('ther is termination point \n');
     plot(it, Etot,'r');
     hold on;
     xlabel('Epoch');
     ylabel('Average of Training error');
     legend('Sum of Squared Error');
     break;
 else
     for i=1:3
         for j=1:4
             dw2(j)=alpha*delta(i)*v(j)';  % alpha*(-(d-y)*y*(1-y)*v)
             w2(i,j)=w2(i,j)-dw2(j);  % w2 - alpha(-(d-y)*y(i)*(1-y)*v)
             e1(j)=w2(i,j)'*delta(i);     % -(d-y)*y.(1-y)*w2
          delata1(j)=v(j).*(1-v(j)).*e1(j); % -(d-y)*y*(1-y)*w2*v*(1-v)
             for k=1:12
                 xt=x(k,:)';
                 for l=1:5
dw1(l)=alpha*delata1(j)*xt(l);   % alpha*(-(d-y)*y*(1-y)*w2*v*(1-v)*x)
w1(j,l)=w1(j,l)-dw1(l);       %w1 - alpha*(-(d-y)*y*(1-y)*w2*v*(1-v)*x)
                 end
             end
         end
     end
```

```
    % disp();
        fprintf('Error At itteration %d :......... %ld',z,Etot(z));
fprintf('\n');
  if z==3000
     fprintf('The net work is not convergent \n');
 end
end
```

Remember: The neural network in our case can't reach error value of 0.0000001(10^{-7}) as given in the problem with in iteration of 3000, but can say that, predicted output (y) **almost the same** with desired output (d) with error of 0.000001 (10^{-6}).

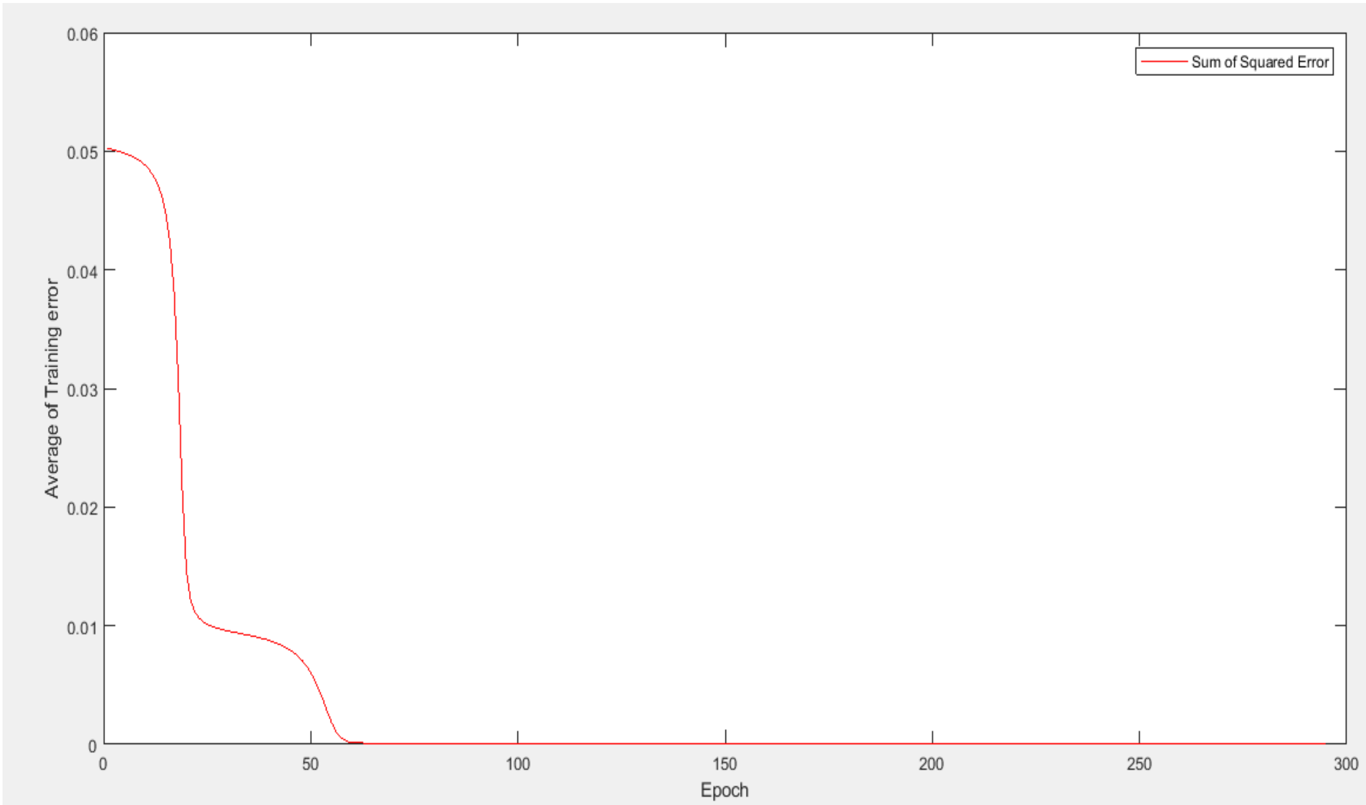
The graph with some progress is as follows

At iteration 60…………error=0.001



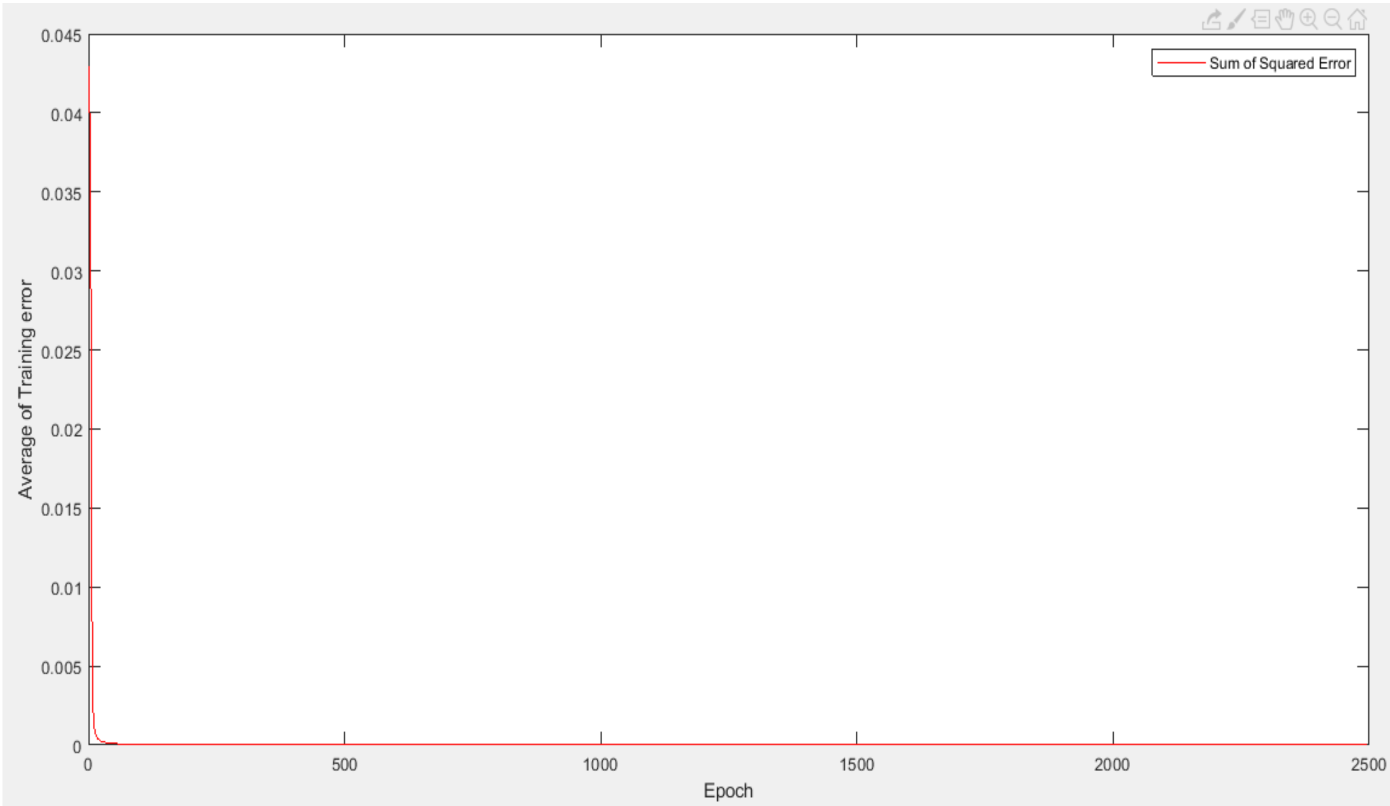
Normalized Desired output (d)	0	1.	0.5133
Predicated output (y)	0.0458	0.9971	0.68

At iteration 295…………error=0.00001



Normalized Desired output (d)	0	1.	0.5133
Predicated output (y)	0.0187	0.9972	0.5133

At iteration 2495 ………..error = 0.000001 (10^{-6}).



Normalized Desired output (d)	0	1.	0.5133
Predicated output (y)	0.006	0.9993	0.5133

From this table we can observe that even though the error 0.0000001 (10^{-7}) not found within 3000 iteration, the neural network is well trained using back propagation algorithm. If we increase number of iteration more than 3000 we will get perfect predicated value. So now we can say that we designed a model to solve a given problem using back propagation by setting the following parameters:

- Beta ==1.5
- Learning rate (alpha)== 1
- Error ==0.000001

If look the scenario properly:

- As iteration number increase we get error value almost approach to zero and predicated value almost the same with desired (actual) output
- As iteration number decrease, it is opposite of the above

Completed mat lab program

```
clear all
x1=randi([-1 1],1,5);                  %initialize  input x1
x2=randi([-2 4],1,5);                  %initialize  input x2
x3=randi([-4 3],1,5);                  %initialize  input x3
x4=randi([-1 1],1,5);                  %initialize  input x4
x5=randi([-2 4],1,5);                  %initialize  input x5
x6=randi([-1 1],1,5);                  %initialize  input x6
x7=randi([-4 3],1,5);                  %initialize  input x7
x8=randi([-2 4],1,5);                  %initialize  input x8
x9=randi([-1 1],1,5);                  %initialize  input x9
x10=randi([-4 3],1,5);                 %initialize  input x10
x11=randi([-4 3],1,5);                 %initialize  input x11
x12=randi([-1 1],1,5);                 %initialize  input x12

dt=[-2.6 3.81 0.69];
d=normalize(dt,'range');                     %initialize and normaliz dired
output at output layer

xtt=[x1;x2;x3;x4;x5;x6;x7;x8;x9;x10;x11;x12;];  %put all input signal in one
input vector
x=normalize(xtt,'range');                           %normalize input vector
for i=1:4
    for j=1:5
        w1t=rand(i,j);                      %initialize weight for input x at
layer1 (total 20 weigts)
        w1=normalize(w1t,'range');
    end
end
fprintf('The initial W1 \n');
disp(w1);
for i=1:3
    for j=1:4
        w2t=rand(i,j);                        %initialize weight at layer 2
        w2=normalize(w2t,'range');
    end
end
fprintf('The initial W2 \n');
disp(w2);

beta=1.5; % to calculat activation function by applying sigmoid function
alpha=1;   % called as (miw)or learning rate
for z=1:3000
    for i=1:3
        for j=1:4
            for k=1:12
                xt=x(k,:)';
                for l=1:5
                    hp(l)=w1(j,l).*xt(l); %store weigt and  input signal
product in vector called 'hp'
                    % disp(hp(l));
                end
```

```
                hs(k)=sum(hp)+1;    %store the same of each 5 element of each
12 input signal vector called 'hs' bay addind bias
            end
            u(j)=sum(hs)/12;      %store the total product summation of each
12 input  signal in vector called 'u'
            exh(j)=u(j).*beta;
            v(j)=1/(1+2.72^-exh(j));   %store the cost function of first
layer in vector called 'v
            op(j)=w2(i,j).*v(j);  %store  activation function and weight
product for second layer in vector called 'op'
        end
        os(i)=sum(op)+1; %store sum product at second layer in vector called
'os' by adding bias 1
        ex(i)=os(i).*beta;
        y(i)=1/(1+2.72^-ex(i));   %store final output in vector called 'y
        em=1/2;
        en=(d(i)-y(i)).^2;
        E(i)=em*en;            %store each corrsponding error in vector
called 'hp
        e(i)=-(d(i)-y(i));
        delta(i)=y(i).*(1-y(i)).*e(i); % delta=-(d-y)*y*(1-y)
    end
    it(z)=z;
    Etotx(z)=sum(E)/12;
    Etot(z)=round(Etotx(z),3);
    if Etot(z)==0.001
        fprintf('ther is termination point \n');
        plot(it, Etot,'r');
        hold on;
        xlabel('Epoch');
        ylabel('Average of Training error');
        legend('Sum of Squared Error');
        break;
    else
        for i=1:3
            for j=1:4
                dw2(j)=alpha*delta(i)*v(j)';        % alpha*(-(d-y)*y*(1-y)*v)
                w2(i,j)=w2(i,j)-dw2(j);             % w2 - alpha(-(d-
y)*y(i)*(1-y)*v)
                e1(j)=w2(i,j)'*delta(i);            % -(d-y)*y.(1-y)*w2
                delata1(j)=v(j).*(1-v(j)).*e1(j);   % -(d-y)*y*(1-y)*w2*v*(1-
v)
                for k=1:12
                    xt=x(k,:)';
                    for l=1:5
                        dw1(l)=alpha*delata1(j)*xt(l);    % alpha*(-(d-
y)*y*(1-y)*w2*v*(1-v)*x)
                        w1(j,l)=w1(j,l)-dw1(l);           %w1 - alpha*(-(d-
y)*y*(1-y)*w2*v*(1-v)*x)
                    end
                end
            end
        end
        % disp();
            fprintf('Error At itteration %d :......... %ld',z,Etot(z));
       fprintf('\n');
```

```
        if z==3000
            fprintf('The net work is not convergent \n');
        end
    end
end
%disp(w1);
disp(y);
disp(d);
```