**Assignment 4**

1. Use one hidden layer with 4 hidden units. Plot the error as a function of the number of iterations. Remember the initial weights.

Solution:

clear all

close all

clc

%% input vector

X = [ 0,0;

0,1;

1,0;

1,1];

%% vector %%

t = [0;

1;

1;

0];

input\_size = 2;

Hidden\_unit = 4;

output\_unit = 1;

bias\_v = 0.5;

bias\_w = 0.5;

max\_iteration = 200;

%% choosing the weights in range of -0.5 to 0.5

a = 0.5;

b = -0.5;

V = (b-a).\*rand(input\_size,Hidden\_unit) + a;

W = (b-a).\*rand(output\_unit,Hidden\_unit) + a;

i = 1;

for iteration = 1:max\_iteration

Error\_value = 0;

for i = 1: 4

%% forward multiplication.

%% level1

Z\_In = (X(i,:)\*V) + bias\_v;

z = 1.0 ./ (1.0 + exp(-Z\_In));

z = z';

%% level2

Y\_In = W\*z + bias\_w;

y = 1.0 ./ (1.0 + exp(-Y\_In));

%% Backward propagation.%%

%% Find error

Error = t(i) - y;

alpha = 0.9;

%% level 2

Derivative\_y = y'\*(1-y);

Delta\_y = Error \* Derivative\_y;

Delta\_w = alpha \* Delta\_y .\* z;

Delta\_w\_bias = alpha \* Delta\_y;

%% level 1

Derivative\_z = z' \*(1-z);

Delta\_z = Derivative\_z .\* (W \* Delta\_y');

Delta\_v = alpha\* Delta\_z' \* X(i,:);

Delta\_v\_bias = alpha \* sum(Delta\_z);

%% update the weights

W = (W' + Delta\_w)' ;

bias\_w = bias\_w + Delta\_w\_bias;

V= V + Delta\_v';

bias\_v = bias\_v + Delta\_v\_bias ;

%% incrementing loop

Error\_value = Error\_value + Error;

end

Error\_plot(iteration) = abs(Error\_value);

end

plot(Error\_plot);

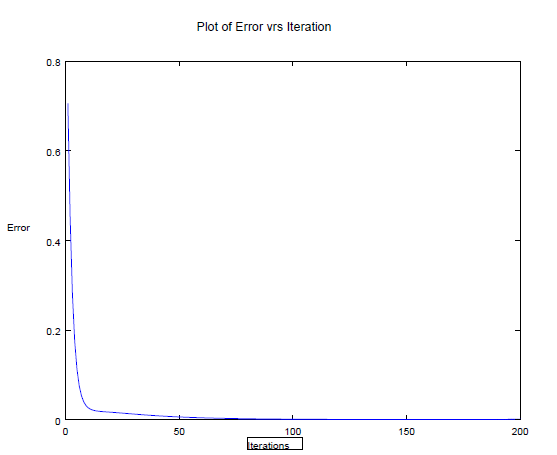


fig 1: Plot Error vrs Iteration using Back propagation method.

1. Verify that momentum term indeed improves convergence ( = 0.5). Use initial weights remembered in (1).

Solution:

clear all

close all

clc

%% input vector

X = [ 0,0;

0,1;

1,0;

1,1];

%% vector %%

t = [0;

1;

1;

0];

input\_size = 2;

Hidden\_unit = 4;

output\_unit = 1;

bias\_v = 0.5;

bias\_w = 0.5;

max = 60;

%% choosing the weights in range of -0.5 to 0.5

a = 0.5;

b = -0.5;

V = (b-a).\*rand(input\_size,Hidden\_unit) + a;

W = (b-a).\*rand(output\_unit,Hidden\_unit) + a;

i = 1;

past\_w = 0;

past\_w0 = 0;

past\_v = 0;

past\_v0 =0;

present\_w= W;

present\_w0= bias\_w ;

present\_v= V;

present\_v0= bias\_v;

for iteration = 1:max

Error\_value = 0;

for i = 1: 4

%% forward multiplication.

%% level1

Z\_In = (X(i,:)\*V) + bias\_v;

z = 1.0 ./ (1.0 + exp(-Z\_In));

z = z';

%% level2

Y\_In = W\*z + bias\_w;

y = 1.0 ./ (1.0 + exp(-Y\_In));

%% Backward propagation.%%

%% Find error

Error = t(i) - y;

alpha = 0.85;

%% level 3

Derivative\_y = y'\*(1-y);

Delta\_y = Error \* Derivative\_y;

Delta\_w = alpha \* Delta\_y .\* z;

Delta\_w\_bias = alpha \* Delta\_y;

%% level 2

Derivative\_z = z' \*(1-z);

Delta\_z = Derivative\_z .\* (W \* Delta\_y');

Delta\_v = alpha\* Delta\_z' \* X(i,:);

Delta\_v\_bias = alpha \* sum(Delta\_z);

**%% update the weights**

**myu = 0.5;**

W = (W' + Delta\_w)' + myu \* (present\_w - past\_w) ;

bias\_w = bias\_w + Delta\_w\_bias + myu\* (present\_w0 - past\_w0);

V= V + Delta\_v' + myu\*(present\_v - past\_v) ;

bias\_v = bias\_v + Delta\_v\_bias + myu\* (present\_v0 - past\_v0) ;

past\_w = present\_w;

past\_w0 = present\_w0;

past\_v = present\_v;

past\_v0 =present\_v0;

present\_w= W;

present\_w0= bias\_w ;

present\_v= V;

present\_v0= bias\_v;

%% Finding the total error

Error\_value = Error\_value + Error;

end

Error\_plot(iteration) = abs(Error\_value);

end

plot(Error\_plot);

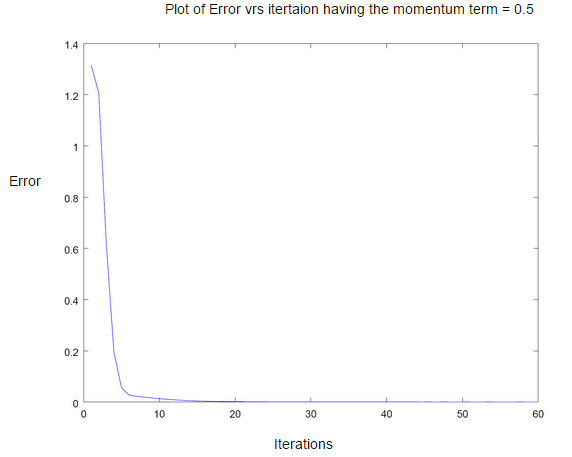


fig 2 Plot of Error vrs iteration having the momentum term = 0.5

1. Repeat (1) using bipolar sigmoid and bipolar representation for X-OR function. The network should converge faster.

Solution:

clear all

close all

clc

%% input vector

X = [ 0,0;

0,1;

1,0;

1,1];

%% vector %%

t = [0;

1;

1;

0];

input\_size = 2;

Hidden\_unit = 4;

output\_unit = 1;

bias\_v = 0.5;

bias\_w = 0.5;

max = 50;

%% choosing the weights in range of -0.5 to 0.5

a = 0.5;

b = -0.5;

V = (b-a).\*rand(input\_size,Hidden\_unit) + a;

W = (b-a).\*rand(output\_unit,Hidden\_unit) + a;

i = 1;

for iteration = 1:max

Error\_value = 0;

for i = 1: 4

%% forward multiplication.

%% level1

Z\_In = (X(i,:)\*V) + bias\_v;

z = Bipolarsigmoid(Z\_In);

z = z';

%% level2

Y\_In = W\*z + bias\_w;

y = Bipolarsigmoid(Y\_In);

%% Backward propagation.%%

%% Find error

Error = t(i) - y;

alpha = 0.9;

%% level 3

Derivative\_y = y'\*(1-y);

Delta\_y = Error \* Derivative\_y;

Delta\_w = alpha \* Delta\_y .\* z;

Delta\_w\_bias = alpha \* Delta\_y;

%% level 2

Derivative\_z = z' \*(1-z);

Delta\_z = Derivative\_z .\* (W \* Delta\_y');

Delta\_v = alpha\* Delta\_z' \* X(i,:);

Delta\_v\_bias = alpha \* sum(Delta\_z);

%% update the weights

W = (W' + Delta\_w)' ;

bias\_w = bias\_w + Delta\_w\_bias;

V= V + Delta\_v';

bias\_v = bias\_v + Delta\_v\_bias ;

%% incrementing loop

Error\_value = Error\_value + Error;

end

Error\_plot(iteration) = abs(Error\_value);

end

plot(Error\_plot);

Bipolarsigmoid.m

function g = Bipolarsigmoid(z)

%SIGMOID Compute sigmoid functoon

% J = BIPOLAR SIGMOID(z) computes the sigmoid of z.

g = (1.0 - exp(-z)) ./ (1.0 + exp(-z));

end

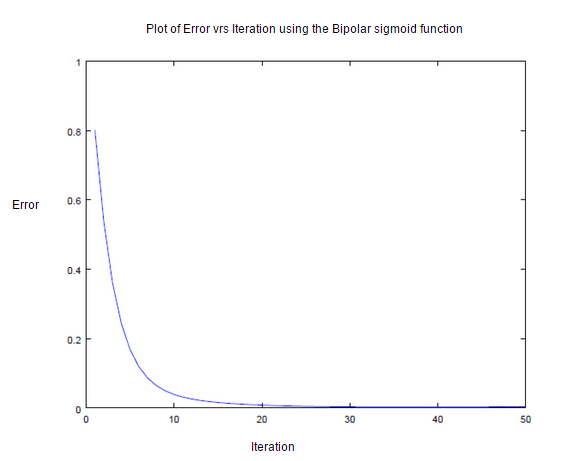


fig 3: Error vrs iteration using Bipolar sigmoid function.

1. Verify that Nguyen – Widrow approach of assigning initial weights improves convergence.

**Solution:**

Set vij to random values between [-0.5, +0.5].

Calculate β = 0.7(h)1/n, where n and h are numbers of input and hidden units, respectively.

Reinitialize weights as vij(new) = β vij(old) / ||vj||. Set bias to a number between [- β, + β].

**Obtained beta value is**  β = 0.7(h)1/n

**h = 4**

**n = 2**

**beta = 1.4;**

clear all

close all

clc

%% input vector

X = [ 0,0;

0,1;

1,0;

1,1];

%% vector %%

t = [0;

1;

1;

0];

input\_size = 2;

Hidden\_unit = 4;

output\_unit = 1;

bias\_v = 1;

bias\_w = 1;

max = 200;

%% choosing the weights in range of -0.5 to 0.5

a = 0.5;

b = -0.5;

V = (b-a).\*rand(input\_size,Hidden\_unit) + a;

%% Applying Nguyen-Widrow Formula.

beta = 0.7 \* (Hidden\_unit)^(1/input\_size)

V = (beta \* V) / norm(V)

W = (b-a).\*rand(output\_unit,Hidden\_unit) + a;

i = 1;

for iteration = 1:max

Error\_value = 0;

for i = 1: 4

%% forward multiplication.

%% level1

Z\_In = (X(i,:)\*V) + bias\_v;

z = 1.0 ./ (1.0 + exp(-Z\_In));

z = z';

%% level2

Y\_In = W\*z + bias\_w;

y = 1.0 ./ (1.0 + exp(-Y\_In));

%% Backward propagation.%%

%% Find error

Error = t(i) - y;

alpha = 0.9;

%% level 3

Derivative\_y = y'\*(1-y);

Delta\_y = Error \* Derivative\_y;

Delta\_w = alpha \* Delta\_y .\* z;

Delta\_w\_bias = alpha \* Delta\_y;

%% level 2

Derivative\_z = z' \*(1-z);

Delta\_z = Derivative\_z .\* (W \* Delta\_y');

Delta\_v = alpha\* Delta\_z' \* X(i,:);

Delta\_v\_bias = alpha \* sum(Delta\_z);

%% update the weighs

W = (W' + Delta\_w)' ;

bias\_w = bias\_w + Delta\_w\_bias;

V= V + Delta\_v';

bias\_v = bias\_v + Delta\_v\_bias ;

%% incrementing loop

Error\_value = Error\_value + Error;

end

Error\_plot(iteration) = abs(Error\_value);

end

plot(Error\_plot);

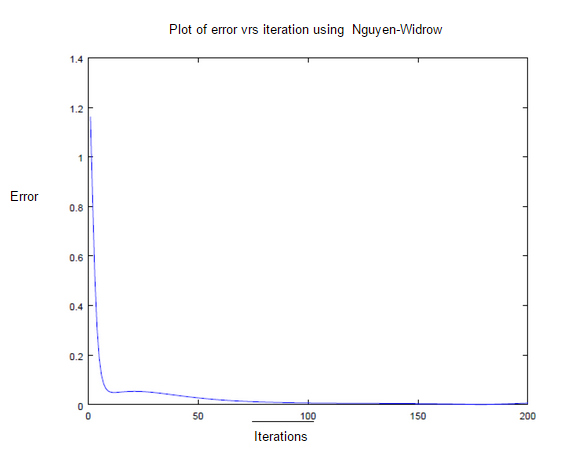


fig 4: plot of error vrs iterations using the Nguyen-Window

1. Use two hidden layers (3 units in the first hidden layer and 2 units in the second hidden layer)

Solution:

clear all

close all

clc

%% input vector

X = [ 0,0;

0,1;

1,0;

1,1];

%% vector %%

t = [0;

1;

1;

0];

input\_size = 2;

Hidden\_unit1 = 3;

Hidden\_unit2 = 2;

output\_unit = 1;

bias\_v = 0.5;

bias\_w = 0.5;

bias\_u = 0.5;

max = 200;

%% choosing the weights in range of -0.5 to 0.5

a = 0.5;

b = -0.5;

V = (b-a).\*rand(input\_size,Hidden\_unit1) + a;

U= (b-a).\*rand(Hidden\_unit1,Hidden\_unit2) + a;

W = (b-a).\*rand(output\_unit,Hidden\_unit2) + a;

i = 1;

for iteration = 1:max

Error\_value = 0;

for i = 1: 4

%% forwarrd multiplication.

%% level1

Z\_In = (X(i,:)\*V) + bias\_v;

z = 1.0 ./ (1.0 + exp(-Z\_In));

z = z';

%% level2

Y\_In = U'\*z + bias\_u;

y = 1.0 ./ (1.0 + exp(-Y\_In));

%% level3

P\_In = W\*y + bias\_w;

p = 1.0 ./ (1.0 + exp(-P\_In));

%% Backward propagation.%%

%% Find error

Error = t(i) - p;

alpha = 0.9;

%% level 3

Derivative\_p = p'\*(1-p);

Delta\_p = Error \* Derivative\_p;

Delta\_w = alpha \* Delta\_p .\* y;

Delta\_w\_bias = alpha \* Delta\_p;

%% level 2

Derivative\_y = y \*(1-y)';

Delta\_y = Delta\_p .\* W \* Derivative\_y;

Delta\_u = alpha .\* (z \* Delta\_y ) ;

Delta\_u\_bias = alpha \* sum (Delta\_y);

%% level 3

Derivative\_z = z' \*(1-z);

Delta\_z = Derivative\_z .\* (U \* Delta\_y');

Delta\_v = alpha\* Delta\_z \* X(i,:);

Delta\_v\_bias = alpha \* sum(Delta\_z);

%% update the weighs

W = (W' + Delta\_w)' ;

bias\_w = bias\_w + Delta\_w\_bias;

U = U + Delta\_u;

bias\_u = bias\_u + Delta\_u\_bias ;

V= V + Delta\_v';

bias\_v = bias\_v + Delta\_v\_bias ;

%% incrementing loop

Error\_value = Error\_value + Error;

end

Error\_plot(iteration) = abs(Error\_value);

end

plot(Error\_plot);

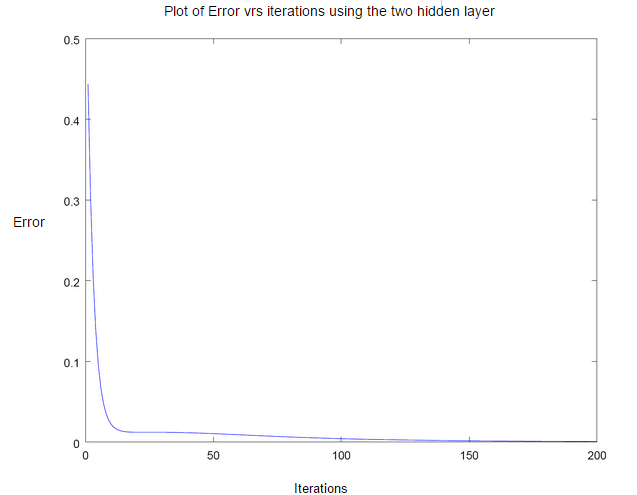


fig 5: plot of error vrs iterations using two hidden layers