ECE 592 course work

Name : Zhezhong Jiang(Harvey)

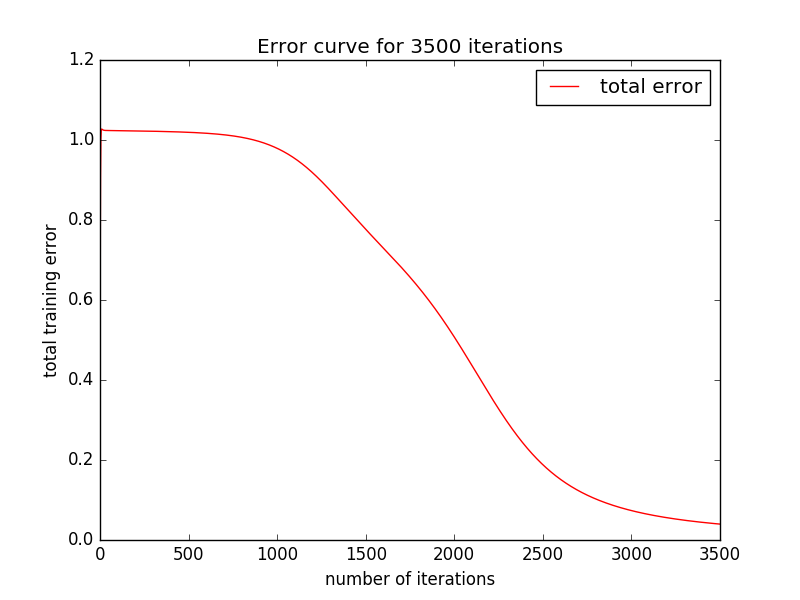
Student ID : 46311130

Part 1 Backpropagation

*1) Set up your network in a 2-input, 4-hidden and 1-output configuration. Apply the XOR training set. Initialize weights to random values in the range -0.5 to +0.5 and set the learning rate to 0.2 with momentum at 0.0.*

*a) Define your XOR problem using a binary representation. Draw a graph of total error against number of epochs. On average, how many epochs does it take to reach a total error of less than 0.05? You should perform many trials to get your results, although you don’t need to plot them all.*

Graph:

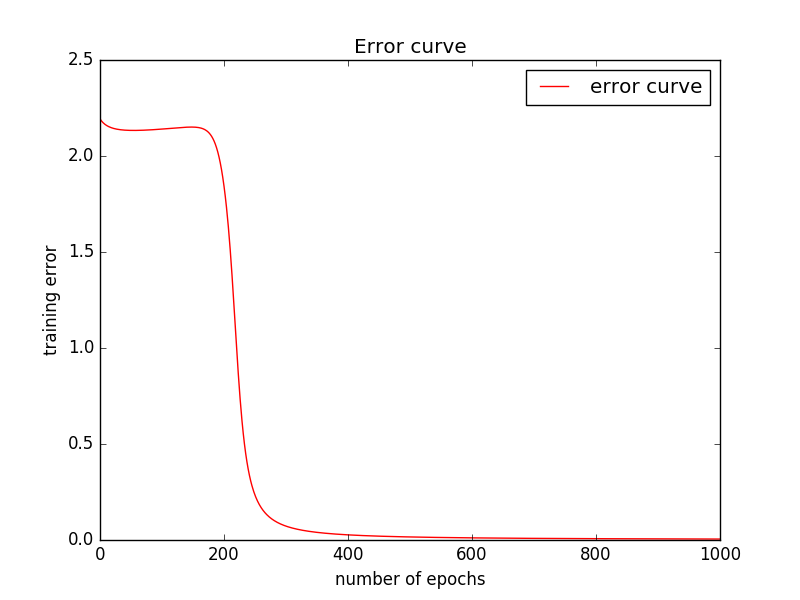


Graph 1(a)

I ran a total of 100 training sessions and find out the average number of iterations needed to reach an error of 0.05 is 4331.

*b) This time use a bipolar representation. Again, graph your results to show the total error varying against number of epochs. On average, how many epochs to reach a total error of less than 0.05?*

Graph shown below : for 1000 iterations

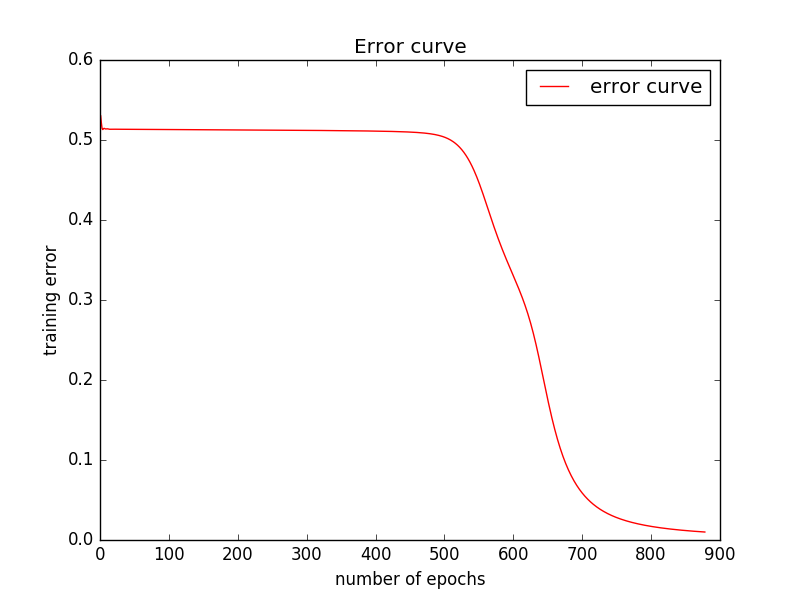


Graph 1(b)

I ran a total of 100 training sessions ,and find the average number of iterations needed to reach an error of 0.05 is 347.

*c) Now set the momentum to 0.9. What does the graph look like now and how fast can 0.05 be reached?*

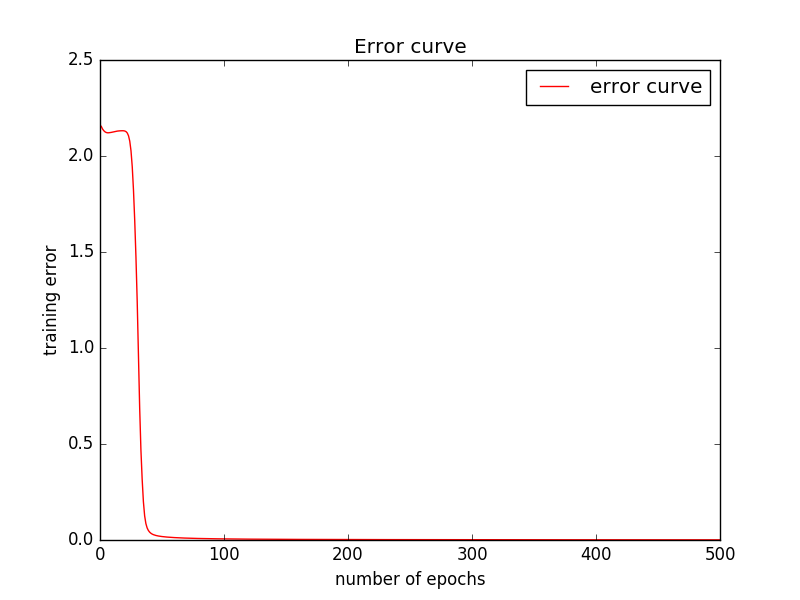
Graph for Binary representations(900 iterations) :



Graph 1(c) a.

I ran a total of 100 training sessions and the reported Average number of iterations needed to reach 0.05 error is 506

Graph for bipolar representation(500 iterations) :



Graph 1(c).b

I ran a total of 100 training sessions and the reported average number of iterations needed to reach an error of 0.05 is 52.

Appendix :

Code:

**package** backprop;

**import** java.io.BufferedWriter;

**import** java.io.File;

**import** java.io.FileNotFoundException;

**import** java.io.FileWriter;

**import** java.io.IOException;

**import** java.util.Arrays;

**import** java.util.Scanner;

**public** **class** bipolar\_2 {

**public** **static** **double**[][] Matrix\_trans(**double** [][] m){

**int** row\_num = m.length;

**int** col\_num = m[0].length;

**double**[][] temp = **new** **double**[col\_num][row\_num];

**for** (**int** i = 0; i < m.length; i++)

**for** (**int** j = 0; j < m[0].length; j++)

temp[j][i] = m[i][j];

**return** temp;

}

**public** **static** **double**[][] Matrix\_sub(**double** [][]m,**double** [][]n){

**int** row\_num = m.length;

**int** col\_num = m[0].length;

**double** [][]sub = **new** **double**[row\_num][col\_num];

**for**(**int** i=0; i<row\_num; i++) {

**for**(**int** j=0; j<col\_num; j++) {

sub[i][j] = m[i][j] - n[i][j];

}

}

**return** sub;

}

**public** **static** **double** [][] Matrix\_dot(**double** [][]m, **double** [][]n){

**double** [][] result = **new** **double** [m.length][n[0].length];

**for** (**int** i = 0; i < m.length; i++) {

**for** (**int** j = 0; j < n[0].length; j++) {

**for** (**int** k = 0; k < m[0].length; k++) {

result[i][j] += m[i][k] \* n[k][j];

}

}

}

**return** result;

}

**public** **static** **double** [][] Matrix\_mul(**double** [][] m, **double** [][] n){

**int** row\_num = m.length;

**int** col\_num = m[0].length;

**double** [][]result = **new** **double** [row\_num][col\_num];

**for**(**int** i=0; i<row\_num; i++) {

**for**(**int** j=0; j<col\_num; j++) {

result[i][j] = m[i][j] \* n[i][j];

}

}

**return** result;

}

**public** **static** **double** [][] initialize\_weights(**int** row,**int** col){

**double** [][]weights = **new** **double**[row][col];

**double** min = -0.5;

**double** max = 0.5;

**for**(**int** i=0; i<row; i++) {

**for**(**int** j=0; j<col; j++) {

/\*change this to random initialization\*/

weights[i][j] = (**double**)(Math.*random*() \* (max - min) + min);

}

}

**return** weights;

}

**public** **static** **double** [][] initialize\_bias(**int** row,**int** col){

**double** [][]weights = **new** **double**[row][col];

**for**(**int** i=0; i<row; i++) {

**for**(**int** j=0; j<col; j++) {

/\*change this to random initialization\*/

weights[i][j] = 0.0;

}

}

**return** weights;

}

**public** **static** **double**[][] sigmoid(**double** [][]m,String flag){

**int** row\_num = m.length;

**int** col\_num = m[0].length;

**double** [][] results = **new** **double**[row\_num][col\_num];

**for**(**int** i=0; i<row\_num; i++) {

**for**(**int** j=0; j<col\_num; j++) {

**if**(flag == "binary") {

/\*perform sigmoid activation for every element in the matrix 1/(1+e^(-x))\*/

results[i][j] = 1/(1+Math.*exp*(-1\*m[i][j]));

}

**else** **if** (flag == "bipolar") {

results[i][j] = (1-Math.*exp*(-1\*m[i][j]))/(1+Math.*exp*(-1\*m[i][j]));

}

}

}

**return** results;

}

**public** **static** **double** [][] Matrix\_add(**double** [][]m,**double** [][] n){

**int** row\_num = m.length;

**int** col\_num = m[0].length;

**double** [][] sum = **new** **double**[row\_num][col\_num];

**for**(**int** i=0; i<row\_num; i++) {

**for**(**int** j=0; j<col\_num; j++) {

sum[i][j] = m[i][j] + n[i][j];

}

}

**return** sum;

}

**public** **static** **double** [][] initialize\_ones(**double** [][]m){

**int** row\_num = m.length;

**int** col\_num = m[0].length;

**double** [][] ones = **new** **double**[row\_num][col\_num];

**for**(**int** i=0; i<row\_num; i++) {

**for**(**int** j=0; j<col\_num; j++) {

ones[i][j] = 1.0;

}

}

**return** ones;

}

**public** **static** **double** [][] sigmoid\_deriv(**double** [][]m,String flag){

**int** row\_num = m.length;

**int** col\_num = m[0].length;

**double** [][] results = **new** **double**[row\_num][col\_num];

**double** [][] ones = **new** **double** [row\_num][col\_num];

**if** (flag == "binary") {

ones = *initialize\_ones*(m);

results = *Matrix\_mul*(m,*Matrix\_sub*(ones,m));

}

**else** **if** (flag == "bipolar") {

ones = *initialize\_ones*(m);

results = *constant\_mul*(0.5,*Matrix\_mul*(*Matrix\_add*(ones,m),*Matrix\_sub*(ones,m)));

}

**return** results;

}

**public** **static** **double** [][] constant\_mul(**double** c,**double**[][]m){

**int** row\_num = m.length;

**int** col\_num = m[0].length;

**double** [][] results = **new** **double**[row\_num][col\_num];

**for**(**int** i=0; i<row\_num; i++) {

**for**(**int** j=0; j<col\_num; j++) {

results[i][j] = c\*m[i][j];

}

}

**return** results;

}

**public** **static** **double** [][] update(**double** lr, **double**[][]weights,**double**[][]d\_weights){

/\* return the updated weights matrix\*/

**int** row\_num = weights.length;

**int** col\_num = weights[0].length;

**double** [][] results = **new** **double**[row\_num][col\_num];

**for**(**int** i=0; i<row\_num; i++) {

**for**(**int** j=0; j<col\_num; j++) {

results[i][j] = weights[i][j]-lr\*d\_weights[i][j];

}

}

**return** results;

}

**public** **static** **double** [][] lr\_product(**double** lr,**double**[][]d\_weights){

/\* return the updated weights matrix\*/

**int** row\_num = d\_weights.length;

**int** col\_num = d\_weights[0].length;

**double** [][] results = **new** **double**[row\_num][col\_num];

**for**(**int** i=0; i<row\_num; i++) {

**for**(**int** j=0; j<col\_num; j++) {

results[i][j] =-lr\*d\_weights[i][j];

}

}

**return** results;

}

**public** **static** **double**[][] create\_v(**double**[][]m){

**int** row\_num = m.length;

**int** col\_num = m[0].length;

**double** [][] results = **new** **double**[row\_num][col\_num];

**for**(**int** i=0; i<row\_num; i++) {

**for**(**int** j=0; j<col\_num; j++) {

results[i][j] =0.0;

}

}

**return** results;

}

**public** **static** **double**[][] create\_mu(**double**[][]m,**double** mu){

**int** row\_num = m.length;

**int** col\_num = m[0].length;

**double** [][] results = **new** **double**[row\_num][col\_num];

**for**(**int** i=0; i<row\_num; i++) {

**for**(**int** j=0; j<col\_num; j++) {

results[i][j] = mu;

}

}

**return** results;

}

**public** **static** **double**[][] update\_v(**double**[][]w,**double** [][]dw,**double** mu,**double**[][]v){

**double** [][] mu\_matrix = *create\_mu*(w,mu);

**return** *Matrix\_add*(*Matrix\_mul*(mu\_matrix,v),*lr\_product*(0.2,dw));

}

**public** **static** **double** [][] update\_weights(**double**[][]w,**double**[][]v){

**return** *Matrix\_add*(w,v);

}

**public** **static** **int** train(**double**[][][]X,**double**[][][]Y,**double**[][]w1,**double**[][]b1,**double**[][]w2,**double**[][]b2,**int** num\_iterations, **double** error\_threshold,**double**[][]v1,**double**[][]v2,**double**[][]v3,**double**[][]v4,**double** mu,**double** cost,String flag) {

**int** count = 0;

**int** sum = 0;

/\* outer loop for iterations \*/

**for** (**int** j =0; j<num\_iterations;j++) {

**if** (cost >= error\_threshold) {

cost = 0.0;

/\* iterating through examples\*/

**for** (**int** i = 0; i <4;i++) {

/\* Z1 = W1\*X[0]+bias\_1\*/

**double** [][] Z1 = *Matrix\_add*(*Matrix\_dot*(w1,X[i]),b1);

**double** [][] A1 = *sigmoid*(Z1,flag);

/\* Z2 = W2\*A1 + bias\_2\*/

**double** [][] Z2 = *Matrix\_add*(*Matrix\_dot*(w2,A1),b2);

**double** [][] A2 = *sigmoid*(Z2,flag);

cost += (Y[i][0][0]-A2[0][0])\*(Y[i][0][0]-A2[0][0]);

/\*back prop\*/

/\* dZ2 = (A2-Y[0])\*(A2)\*(1-A2)\*/

**double** [][]sigmoid\_deriv\_1 = *sigmoid\_deriv*(A2,flag);

**double** [][]dZ2 = *Matrix\_mul*(*Matrix\_sub*(A2,Y[i]),sigmoid\_deriv\_1);

/\*dW2 = np.dot(dZ2,A1.T)\*/

**double** [][]dW2 = *Matrix\_dot*(dZ2,*Matrix\_trans*(A1));

v2 = *update\_v*(w2,dW2,mu,v2);

w2 = *update\_weights*(w2,v2);

/\* db2 = dZ2\*/

**double** [][]db2 = dZ2 ;

v4 = *update\_v*(b2,db2,mu,v4);

b2 = *update\_weights*(b2,v4);

/\*dZ1 = np.dot(W2.T,dZ2)\*(A1)\*(1-A1)\*/

**double** [][]dZ1 = *Matrix\_mul*(*Matrix\_dot*(*Matrix\_trans*(w2),dZ2),*sigmoid\_deriv*(A1,flag));

/\*dW1 = np.dot(dZ1,X.T)\*/

**double** [][]dW1 = *Matrix\_dot*(dZ1,*Matrix\_trans*(X[i]));

v1 = *update\_v*(w1,dW1,mu,v1);

w1 = *update\_weights*(w1,v1);

/\*db1 = dZ1\*/

**double** [][]db1 = dZ1 ;

v3 = *update\_v*(b1,db1,mu,v3);

b1 = *update\_weights*(b1,v3);

}

cost = cost\*0.5;

count+=1;

sum = j;

System.***out***.println(count + " "+cost);

} /\* if loop\*/

} /\* this bracket closes the outer most loop\*/

**return** sum;

}

**public** **static** **void** predict(**double**[][][]X,**double**[][][]Y,**double** [][] weights\_1,**double** [][] bias\_1,**double**[][] weights\_2,**double** [][] bias\_2,String flag) {

/\* predict \*/

**for** (**int** i = 0; i <4;i++) {

**double** [][] Z1 = *Matrix\_add*(*Matrix\_dot*(weights\_1,X[i]),bias\_1);

**double** [][] A1 = *sigmoid*(Z1,flag);

/\* Z2 = W2\*A1 + bias\_2\*/

**double** [][] Z2 = *Matrix\_add*(*Matrix\_dot*(weights\_2,A1),bias\_2);

**double** [][] A2 = *sigmoid*(Z2,flag);

System.***out***.println(Arrays.*deepToString*(A2));

}

}

**public** **static** **void** save (String file, **double**[][]weights) **throws** IOException{

**try** {

BufferedWriter outputWriter = **null**;

outputWriter = **new** BufferedWriter(**new** FileWriter(file));

**for** (**int** i=0;i<4;i++) {

**for** (**int** j=0;j<2;j++) {

outputWriter.write(String.*valueOf*(weights[i][j]));

outputWriter.newLine();

}

}

outputWriter.flush();

outputWriter.close();

}

**catch**(Exception e) {

}

}

**public** **static** **double** [][] load(String file,String flag) **throws** FileNotFoundException {

**try** {

File f = **new** File(file);

Scanner s = **new** Scanner(f);

**double** [][] results = **new** **double**[4][2];

**if**(flag == "weights\_1") {

results = **new** **double**[4][2];

**for** (**int** k = 0;k<4;k++) {

**for** (**int** j =0;j<2;j++) {

**if** (s.hasNext()) {

results[k][j] = Double.*parseDouble*(s.next());

}

}

}

}

**else** **if**(flag == "weights\_2") {

results = **new** **double**[1][4];

**for** (**int** k = 0;k<1;k++) {

**for** (**int** j =0;j<4;j++) {

**if** (s.hasNext()) {

results[k][j] = Double.*parseDouble*(s.next());

}

}

}

}

**return** results;

}

**catch**(Exception e) {

**return** **null**;

}

}

**public** **static** **void** main(String[] args) {

String flag = "binary";

**double** [][][] X = {{{0},{0}},{{0},{1}},{{1},{0}},{{1},{1}}};

**double** [][][] Y = {{{0}},{{1}},{{1}},{{0}}};

**if** (flag == "binary") {

X = **new** **double**[][][]{{{0},{0}},{{0},{1}},{{1},{0}},{{1},{1}}};

Y = **new** **double**[][][] {{{0}},{{1}},{{1}},{{0}}};

}

**else** **if** (flag == "bipolar") {

X = **new** **double**[][][]{{{-1},{-1}},{{-1},{1}},{{1},{-1}},{{1},{1}}};

Y = **new** **double** [][][]{{{-1}},{{1}},{{1}},{{-1}}};

}

**double** [][] weights\_1 = *initialize\_weights*(4,2);

**double** [][] bias\_1 = *initialize\_bias*(4,1);

**double** [][] weights\_2 = *initialize\_weights*(1,4);

**double** [][] bias\_2 = *initialize\_bias*(1,1);

/\* momentum matrix\*/

**double** [][] v1 = *create\_v*(weights\_1);

**double** [][] v2 = *create\_v*(weights\_2);

**double** [][] v3 = *create\_v*(bias\_1);

**double** [][] v4 = *create\_v*(bias\_2);

**double** [][]loaded\_weights ;

**int** sum = 0;

**int** sum\_2 = 0;

**int** num = 0;

**double** cost = 1.0;

/\* training sessions\*/

**for** (**int** k =0 ;k <=100; k++) {

cost = 1.0;

weights\_1 = *initialize\_weights*(4,2);

bias\_1 = *initialize\_bias*(4,1);

weights\_2 = *initialize\_weights*(1,4);

bias\_2 = *initialize\_bias*(1,1);

/\* momentum matrix\*/

v1 = *create\_v*(weights\_1);

v2 = *create\_v*(weights\_2);

v3 = *create\_v*(bias\_1);

v4 = *create\_v*(bias\_2);

/\* hyper parameters\*/

**double** mu = 0.9;

**int** num\_iterations = 1000;

**double** error\_threshold = 0.05;

sum = *train*(X,Y,weights\_1,bias\_1,weights\_2,bias\_2,num\_iterations, error\_threshold,v1,v2,v3,v4, mu,cost,flag);

sum\_2 = sum\_2 +sum;

num = k;

} /\* training session loop\*/

System.***out***.println("Average number of iterations needed to reach 0.05 error is" + sum\_2/num);

/\* predict\*/

*predict*(X,Y,weights\_1,bias\_1,weights\_2,bias\_2,flag);

**try** {

*save* ("weights\_1.txt",weights\_1);

}

**catch**(Exception e) {

}

**try** {

loaded\_weights = *load*("C:\\Users\\zhezhong\\Desktop\\New folder\\ece592\\backprop\\weights\_1.txt","weights\_1");

System.***out***.println("loaded weights"+Arrays.*deepToString*(loaded\_weights));

}

**catch** (Exception e) {

}

}

}