,

Computer Assignment 3 Report by Pongsakorn Rattanapan. Student ID: 630610749

1 Wisconsin Diagnostic Breast Cancer (WDBC)

Classification to diagnosis malignant or benign cancer

1.1 problem

ให้ทำการทดลองกับ wdbc.data (Wisconsin Diagnostic Breast Cancer (WDBC) จาก UCI Machine learning Repository) โดยที่ data set นี้ มี 2 classes และ 30 features ซึ่งในแต่ละ sample จะมีทั้งหมด 32 ค่าโดยที่

1) ID number

2) Diagnosis (M = malignant, B = benign) -> class

3-32) เป็นค่า features ทั้ง 30

ให้ทำการทดลองโดยใช้ 10% cross validation เพื่อทดสอบ validity ของ network ที่ได้ และให้ทำการเปลี่ยนแปลงจำนวน hidden layer และ nodes

1.2 Design

ในการออกแบบระบบ Genetic algorithm นั้นเป็นการเลียนแบบธรรมชาติของสิ่งมีชีวิต โดยที่หากสิ่งมีชีวิตตัวไหนมี Chromosome ที่ทำได้ดีในสถานะการนั้นๆ ก็จะมีโอกาสอยู่รอด และสร้างลูกสร้างหลานได้มากกว่า โดยที่ลูกหลานก็อาจจะมีพฤติกรรมที่ที่มาจาก Genes ของพ่อหรือแม่ รวมถึงอาจเกิดการ mutation ของ Genes ได้เอง

โดยในการทดลองจะมี population 50 individual โดยในแต่ละ individual คือ 1 neural network หรือ Multi-Layer Perceptron ซึ่งเป็นตัวแทนของ 1 chromosome โดยมี Genes แทน weight ในแต่ละเส้นของ Multi-Layer Perceptron

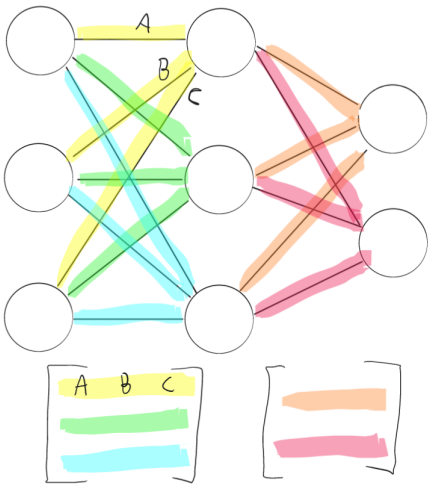
individual แต่ละตัวก็จะมี unique weight ที่ทำให้การทำนายผล Tumor ของแต่ละ 30-input ว่าเป็น malignant หรือ benign ซึ่งจะมีค่า error เมื่อนำไปผ่านสมการ scaling

$$Fitness = 1/(error + 0.01) \tag{1}$$

จะได้ว่าหาก individual ตัวไหนที่มีค่า error น้อยก็จะมีค่า fitness ที่มาก และเป็น idividual ที่ดี

1.2.1 Chromosome Encoding

เราใช้ matrix (3D array)ในการเก็บค่า weight ของแต่ละเส้นของ MLP โดย ในแต่ละ row ของ matrix แทน node หนึ่งใน MLP และ ในแต่ละ col ของ row นั้นๆ แทน weight อันที่เชื่อมกับ node นั้นๆ



ในการทำ Crossover หรือ Mutation จึงทำได้โดยการ แก้ไขและแลกเปลี่ยนค่าใน matrix

1.3 process

แบ่งข้อมูล 90 % train และ 10 % blind-test สำรับการ 10% cross validation (selecting 1 line data every 10 lines of data to distribute the grouping of data)

$$\varphi(x) = \begin{cases} x, & \text{if } x > 0, \\ 0.01, & \text{otherwise.} \end{cases} \tag{2}$$

The activation function used Leaky ReLU (1)
โดย MLP ที่ใช้ในการทดลองนี้จะเป็น 30-16-2 โดยมี input 30 features และ output 2 node หาก output node 0 > node 1 classify to [malignant] else [benign]
โดยมี Crossover rate 0.3 และ Mutation probability 0.01

โดยในแต่ละ generation ทำตามขั้นตอนต่อไปนี้

1.3.1 Initial Population

หากยังไม่มี individual ใน population ทำการสร้าง individual mlp โดยสุ่ม weights ในแต่ละเส้นของ mlp ในช่วง [-1,1] โดยขนาดของ population_pool คือ 50 individual

1.3.2 individual evaluate fitness

ในแต่ละ individual รับ train_dataset มาและทำ feed forward neural network เป็นค่า predict ของ individual และคิด error (Mean squared error) จาก train_desired_dataset ซึ่งยิ่งมีค่า error น้อยแสดงว่า predict ได้อย่างแม่นยำ

นำค่า error ไปเข้า scaling function - (1) จะได้ค่า fitness ในแต่ละ individual หากตัวไหน predict ได้แม่นยำก็จะมี fitness ที่สูง

1.3.3 selection

ใช้วิธี random tournament selection สุ่มเลือกมา 2 individual แล้วตัวไหนมีค่า fitness ที่มากกว่าเลือกตัวนั้นไป selection_pool ทำแบบนี้ 50 รอบ by this code:

```
        for(int i = 0 ; i < 50 ; i++) {
            select1 =  uniquerandom
            select2 =  uniquerandom

            if(select1.fitness  > select2.fitness  )
                selection_pool[i] = select1;
            else
                selection_pool[i] = select2;
        }
```

1.3.4 mating pool & crossover

จาก seletion_pool สุ่มคู่ของ individual ในที่นี้คือ 25 คู่
โดยจากพ่อแม่ 50 individual ใน seletion_pool จะได้ 25 individual ที่เป็น offspring จาก 2 individual ของพ่อแม่

Pair mating by this code:

```
    for(int pair = 0 ; pair< 25 ; pair++){
            father_individual = uniquerandom from selection_pool
            mother_individual = uniquerandom from selection_pool

            individual offspring =  crossover( father_individual, mother_individual );

            offspring_pool.add(offspring);
        }
```

ในแต่ละคู่จะสร้าง offspring ซึ่งเป็น MLP ที่มี weight บางส่วนมาจากพ่อ และ มาจากแม่ ด้วยโอกาส prob_parent โดยในแต่ละ non-output node ของ mlp หากมาจากพ่อ นำค่า weight ของ input ทั้งหมดที่เชื่อมกับ non-output node นั้น

Crossover by this code:

```
             foreach(non-output node) {
                double q = uniform_random(0.0,1.0);
                  if (q < prob_parent) {
                      offspring_weight_@node = father_weight;
                  } else {
                      offspring_weight_@node = mother_weigh;
                  }
                }
```

1.3.5 Mutation

ในแต่ละ non-output node ของ mlp จะมีโอกาสที่ weight จะเปลี่ยนค่าด้วยโอกาส p ดังนี้

Generate a random number q from U(0, 1)

- q < p, mutate
- q >= p, don't mutate

mutation by this code:

```
    mutation(node){
        for (int line = 0 ; line < node.input ; line ++ ){
            double e = uniform_random(-1.0,1.0);
            node.line +=  e;
        }
    }
```

1.3.6 increase offspring

ใน generation ต่อไปจะต้องมีจำนวน individual คงที่ จึงต้องทำการเพิ่มจำนวนของ offspring ที่จะไปใน generation ถัดไปให้เท่ากับ generation นี้ โดยสุ่ม copy individual จากพ่อแม่ใน population_pool มาเติมใน offspring_pool
All by this code:

```
        while(offspring_pool.size() < size_population){
            individual copy = uniquerandom from  population_pool
            offspring_pool.add(copy);
        }
```

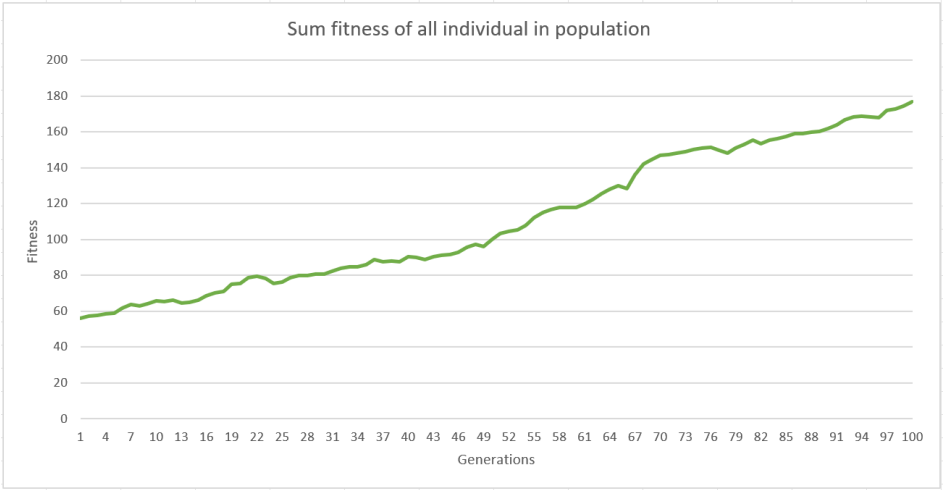
1.3.7 next_population

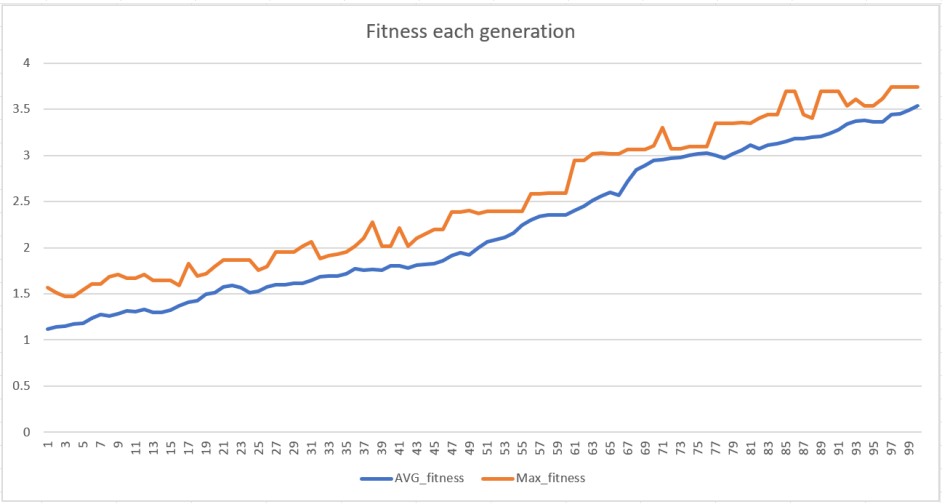
individual แต่ละตัวที่อยู่ใน offspring_pool ที่ผ่านการ crossover จากพ่อแม่ การ mutation จะถูกย้ายไปเป็น generation ถัดไป
by this code:

```
    population_pool =  offspring_pool;
```

1.4 result

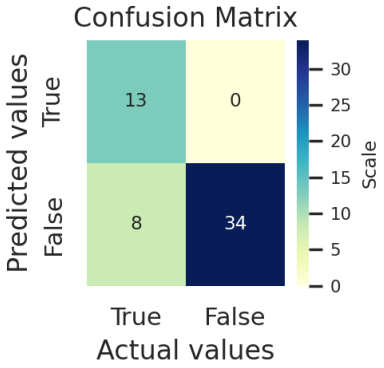
เมื่อผ่านไปในแต่ละ generations จะเห็นได้ว่า ค่า fitness ปรับตัวสูงขึ้น กล่าวคือ แต่ละ individual เข้าใกล้ solution weight ที่ต้องการมากขึ้น ดังตาราง



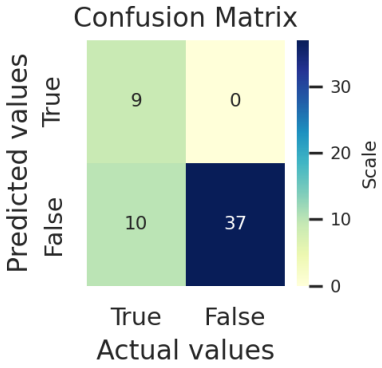


เมื่อผ่านไปครบ 100 generations individual ที่ให้ค่า fitness สูงที่สุดจากทั้งหมดคือ best solution mlp โดยเมื่อนำ solution ไปทำการทดสอบกับ test_dataset ซึ่งเป็น blindtest จาก 10% cross validation ได้ผลลัพธ์ดังนี้

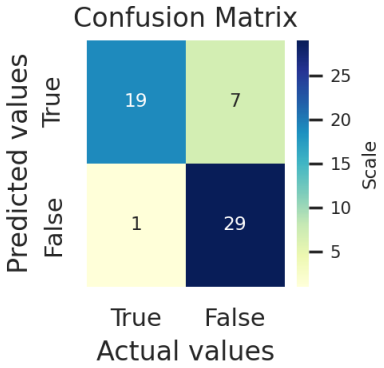
test_dataset 1



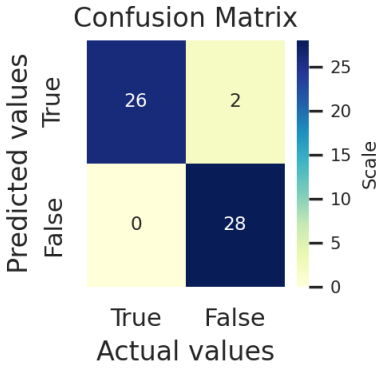
test_dataset 2



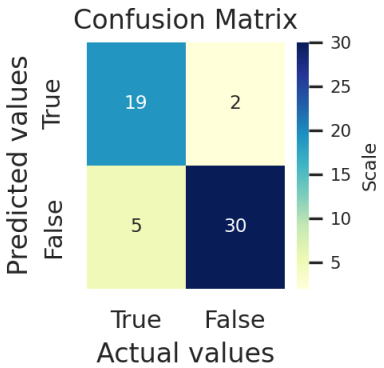
test_dataset 3



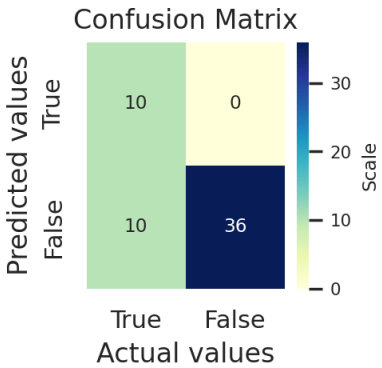
test_dataset 4



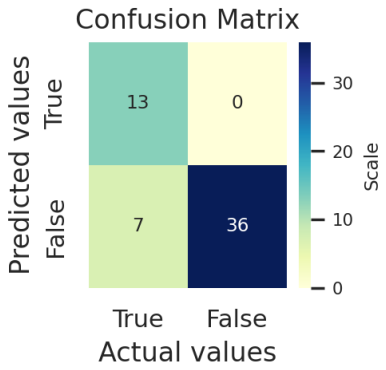
test_dataset 5



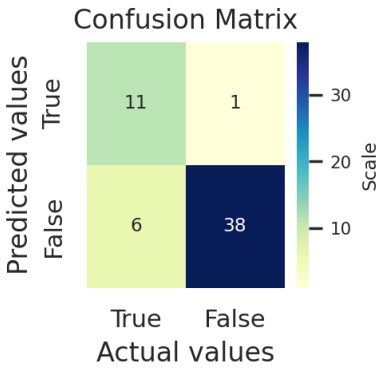
test_dataset 6



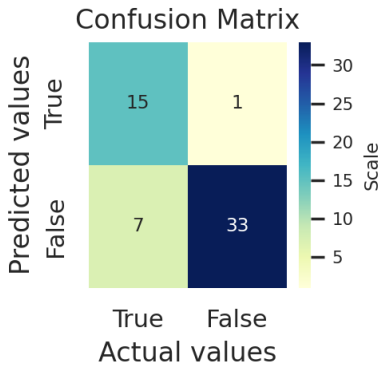
test_dataset 7



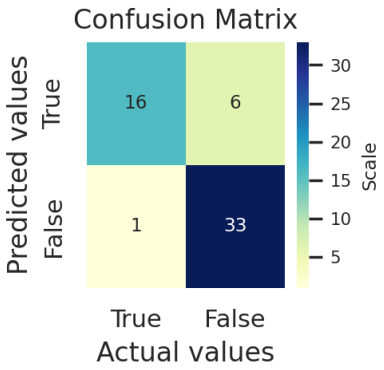
test_dataset 8



test_dataset 9



test_dataset 10



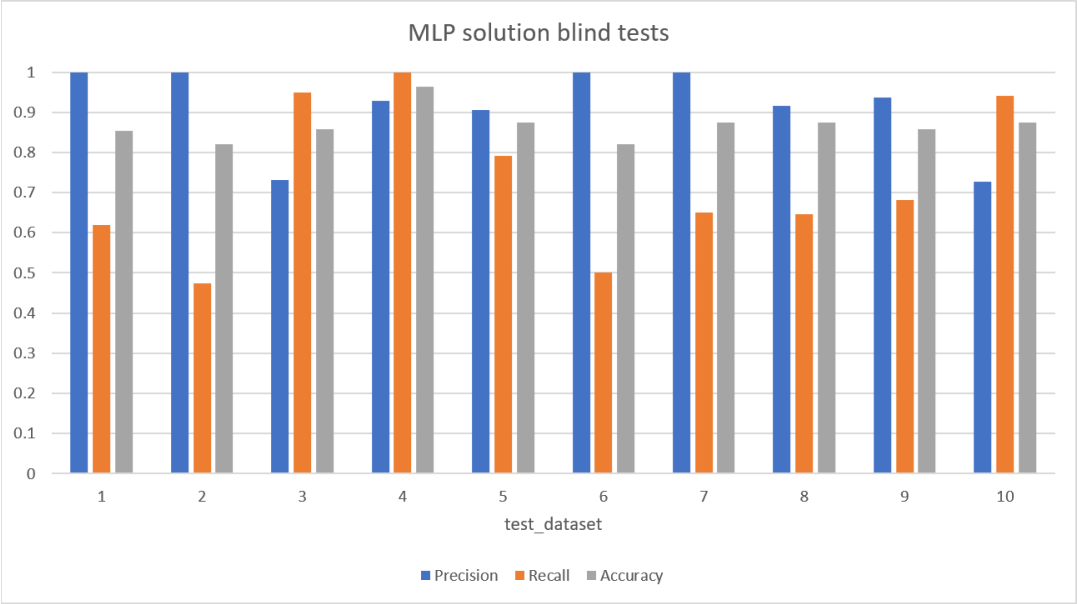
then try changes hidden nodes to test the impact. (different initialize weights between -1.0 to 1.0)

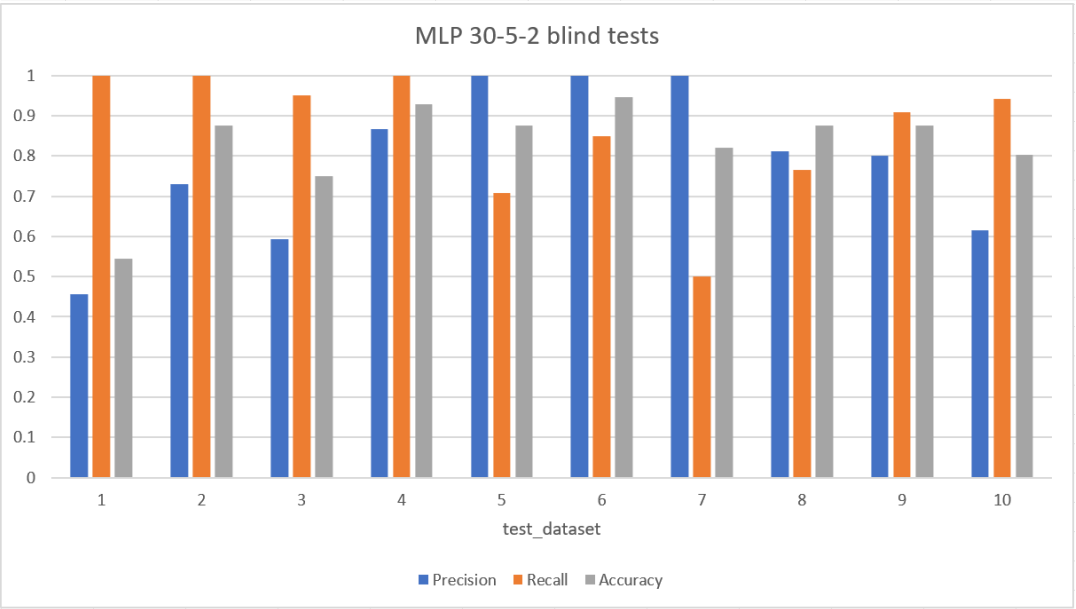
ทดสอบ validity ของ network โดยทำการเปลี่ยนแปลงจำนวน hidden layer และ nodes ดังตารางนี้

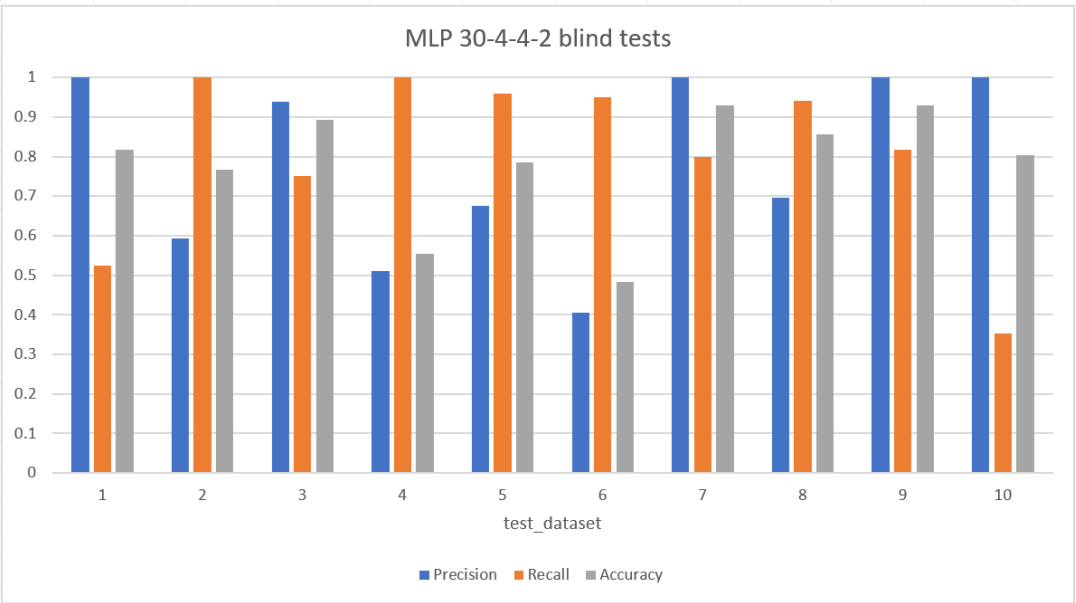
neural test	
Test	Neural Type
base	30-16-2
A	30-5-2
B	30-4-4-2
C	30-15-15-2

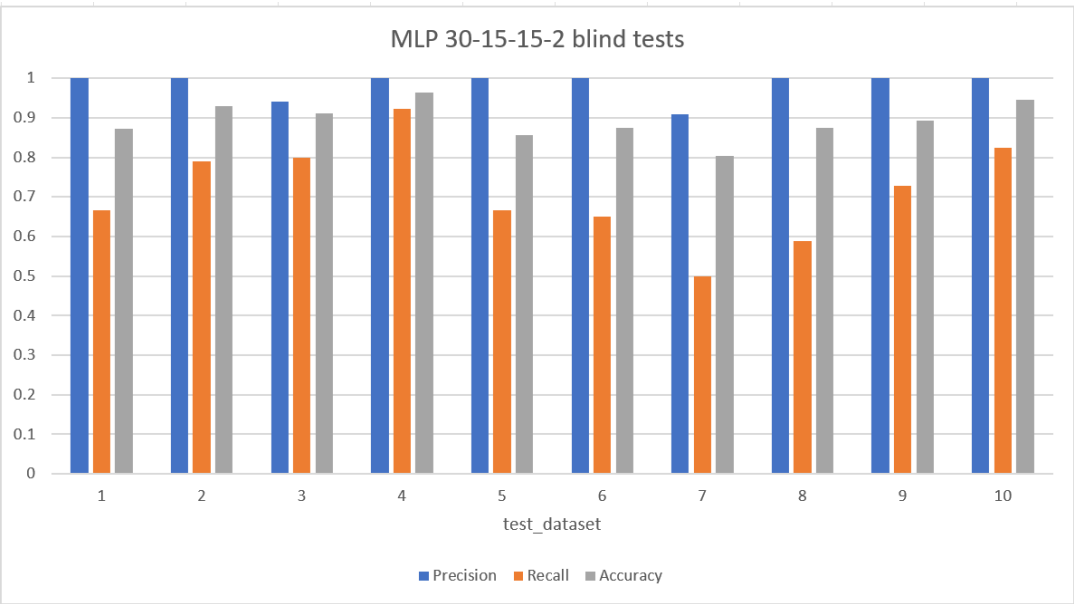
จากนั้นวัดค่า

-Precision as a comparison of the correct prediction is true and it true (TP) and the prediction is true but is not true (FP).
-Recall as the accuracy of the prediction ”true” compared to the number of times the event both predicted and actual as ”true”.
-Accuracy as the accuracy of predict matches actually correctly


MLP solution blind tests
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
1 2 3 4 5 6 7 8 9 10
test_dataset
Precision
Recall
Accuracy


MLP 30-5-2 blind tests
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
1 2 3 4 5 6 7 8 9 10
test_dataset
Precision
Recall
Accuracy


MLP 30-4-4-2 blind tests
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
1 2 3 4 5 6 7 8 9 10
test_dataset
Precision
Recall
Accuracy



1.5 Genetic analysis

จะเห็นได้ว่า MLP ที่มีโครงสร้าง 30-16-2 และมีความแม่นยำคงที่ ที่ประมาณ 0.8 ขึ้นไปในทุก test case ซึ่งอาจจะเป็นเพราะว่าเป็นจำนวนที่ไม่น้อยเกินไป เหมือน MLP 30-5-2 ที่อาจจะทำให้ ค่า เด่นๆของ input นั้น หล่นหายไปได้ถึงแม้จะมี layer ที่น้อยกว่า 30-15-15-2 ซึ่ง 30-16-2 กับ 30-15-15-2 มีความแม่นยำคงที่ ที่ประมาณ 0.8 ขึ้นไป อาจจะเป็นเพราะจำนวน node ในแต่ละ layer ที่มากเพียงพอที่จะส่งค่าเด่นๆ ไปยัง layer ต่อไปได้ ไม่เหมือน 30-4-4-2 ที่ถึงแม้จะมี layer เท่ากับ 30-15-15-2 แต่ไม่ได้มีความแม่นยำคงที่ในทุกๆ test case

1.6 Conclusion

genetic algorithms ในการเพิ่มค่า fitness ของ MLP โดยการพยายามเลือกส่วนที่ดีของพ่อหรือแม่ นำมารวมกันได้ MLP ตัวใหม่ และมีการ Mutation เพื่อป้องกัน local solution จากการ ทดลองพบว่า Best individual ตัวที่ดีที่สุดจาก population ทั้งหมดมีความแม่นยำที่รับได้ โดยอยู่ที่ 0.8 ถึง 0.9

โดยใน โปรเจคนี้มี ค่าหลายอย่างที่สามารถปรับได้ ไม่ว่าจะเป็น biased , activation function ที่ใช้ ในส่วนการ crossover ผมเองก็รู้สึกทึ่งเหมือนกัน ที่แค่ถ่ายทอด weight มาจาก individual จะทำให้ MLP มีความแม่นยำมากขึ้นได้ ซึ่งผมคิดว่าการหา local gradient และการทำ backpropagation ยังดูมีหลักการมากกว่า

2 Source code

All the source code are in github: https://github.com/min23asdw/xxxxx

2.1 Main.java

```java
import java.io.*;
import java.util.ArrayList;

public class main {

    private static ArrayList<ArrayList<Double[]>> test_dataset = new ArrayList<>();
    private static ArrayList<ArrayList<Double[]>> test_desired_data = new ArrayList<>();

    private static ArrayList<ArrayList<Double[]>> train_dataset = new ArrayList<>();
    private static ArrayList<ArrayList<Double[]>> train_desired_data = new ArrayList<>();

    private static int NumberOftest = 10;

    public static void main(String[] args) throws IOException {
        for(int tain_i = 0 ; tain_i < NumberOftest ; tain_i ++) {
            ArrayList<Double[]> test_dataset_i = new ArrayList<>();
            ArrayList<Double[]> test_desired_data_i = new ArrayList<>();

            ArrayList<Double[]> train_dataset_i = new ArrayList<>();
            ArrayList<Double[]> train_desired_data_i = new ArrayList<>();

            FileInputStream fstream = new FileInputStream("src/wdbc.data");
            DataInputStream in = new DataInputStream(fstream);
            BufferedReader br = new BufferedReader(new InputStreamReader(in));
            String data;

            int line_i = 1;
            while ((data = br.readLine()) != null) { // each line

                    String[] eachLine = data.split(",");

                    Double[] temp_input = new Double[30];

                Double[] ans = new Double[2];

                    for(int  i = 2 ; i<eachLine.length;i++){
                        temp_input[i-2] = Double.parseDouble(eachLine[i])/6000.8;  // lazy min max norm
                    }

                    if(eachLine[1].equals("B")){ ans = new Double[]{0.0, 1.0};   // b 01
                    }
                    else if (eachLine[1].equals("M")){ans = new Double[]{1.0, 0.0};  // m 10
                    }

                        if (line_i % 10 == tain_i) {  // 10% for test
                            test_dataset_i.add(temp_input);
                            test_desired_data_i.add(ans);
                        } else {
                            train_dataset_i.add(temp_input);
                            train_desired_data_i.add(ans);
                        }

                line_i++;
            }
            test_desired_data.add(test_desired_data_i);
            train_desired_data.add(train_desired_data_i);
            test_dataset.add(test_dataset_i);
            train_dataset.add(train_dataset_i);
        }
        System.out.println("work");

        for(int test_i = 0 ; test_i < NumberOftest ; test_i ++) {
            System.out.println("==================================================");
```

```
            genetic ga = new genetic("30,16,2",  0 , 50 , 100 );
            System.out.println("train: " + test_i);
            ga.settraindata( train_dataset.get(test_i), train_desired_data.get(test_i));
            ga.run_gen();
            System.out.println("test: " + test_i);
            ga.test(test_dataset.get(test_i), test_desired_data.get(test_i));
            System.out.println("=====================================================");
        }
    }
}
```

2.2 genetic.java

```
import java.util.ArrayList;
import java.util.Random;
public class genetic {

    String[] mlp;
    String neural_type;
    double biases;
    int num_population;
    int maxGeGeneration;
    Random r = new Random();
    individual[] population_pool;

    individual[] selection_pool ;

    ArrayList<Double[]> train_dataset;
    ArrayList<Double[]> train_desired_data;

    ArrayList<Double[]> train_result = new ArrayList<>();

    ArrayList< Pair<Double , individual>> scoreBoard = new ArrayList<>();

    double prob_mul = 0.01;
    double prob_parent = 0.3;
    double best_mlp_score = 0.0;


    public genetic( String _mlp ,double _biases ,int _population , int _maxGeneration){
        this.neural_type = _mlp;
        this.mlp = _mlp.split(",");;
        this.biases = _biases;
        this.num_population = _population;
        this.maxGeGeneration = _maxGeneration;

    }

    public void settraindata(ArrayList<Double[]> _train_dataset, ArrayList<Double[]> _train_desired_data)
         {
        this.train_dataset = _train_dataset;
        this.train_desired_data = _train_desired_data;
    }

    public void  init_population(){
        individual[] init =  new individual[num_population];
        for(int indi = 0 ; indi < num_population ; indi++){
            individual indiler = new individual( neural_type ,biases );
            init[indi] = indiler;
        }
        this.population_pool = init;
        this.selection_pool = new individual[num_population];
    }

    public void run_gen(){
        System.out.println("train");
        init_population();

        for (int gen = 0 ; gen < maxGeGeneration ; gen++){

            Double[] result =  population_eval(train_dataset,train_desired_data);
            System.out.println(result[0] + "\t" + result[1] + "\t" + result[2] );
            train_result.add(result);
            selection();
            ArrayList<individual> offspring_pool =  p1();
            p2(offspring_pool);
            add2N(offspring_pool);

            if(gen != maxGeGeneration-1)  move2next_population(offspring_pool);

        }
        System.out.println("");

```

```
    }

    public void test(ArrayList<Double[]> dataset, ArrayList<Double[]> desired_data){
        System.out.println("test");
        individual best_solution =  scoreBoard.get(scoreBoard.size()-1).individual;
        best_solution.test(dataset,desired_data);
        System.out.println("test");

    }

    public Double[] population_eval(ArrayList<Double[]> dataset , ArrayList<Double[]> desired_data){
        double sum_fit = 0;
        double avg_fit;
        double max_fit = -9999999;

        //eval fitness
        for (individual mlp : population_pool) {
            double error_mlp =   mlp.eval(dataset , desired_data);

            double fitness = scaling(error_mlp);
            if(max_fit<fitness){
                max_fit = fitness;
            }

            sum_fit += fitness;

            if(best_mlp_score < fitness){
                best_mlp_score = fitness;
                Pair<Double , individual> score_individual = new Pair<>(fitness, mlp);
                scoreBoard.add(score_individual);
            }

        }
        avg_fit = sum_fit / num_population;
        return new Double[]{sum_fit, avg_fit, max_fit};
    }

    public double scaling(double error){
        return 1/(error + 0.01)  ;
    }

    public void selection(){
        // random tournament selection

        for(int i = 0 ; i < num_population ; i++) {

            int select1 = r.nextInt(0,49);
            int select2 = r.nextInt(0,49);
            individual a1 = population_pool[select1].clone();
            individual a2 = population_pool[select2].clone();

            double a1_fit = scaling(a1.avg_error_n);
            double a2_fit = scaling(a2.avg_error_n);

            if(a1_fit > a2_fit  ) selection_pool[i] = a1;
            else  selection_pool[i] = a2;
        }
    }

    public  ArrayList<individual>  p1(){
        ArrayList<individual> offspring_pool = new ArrayList<>();
        //pair selection
        for(int ran = 0 ; ran < num_population/2 ; ran++){
            int select1 = r.nextInt(0,selection_pool.length);
            int select2 = r.nextInt(0,selection_pool.length);

            individual offspring =  crossover(selection_pool[select1], selection_pool[select2]);

            offspring_pool.add(offspring);
        }
        return offspring_pool;
    }
```

```
142
143     public individual crossover(individual f , individual m){
144         Matrix[] father_weight = f.clone().get_weight();
145         Matrix[] mother_weight = m.clone().get_weight();
146
147         individual offspring = new individual( neural_type ,biases );
148
149         Matrix[] offspring_weight = newWeight(f);
150         for (int layer = 0 ; layer < father_weight.length ; layer++ ) {
151             for (int node = 0; node < father_weight[layer].rows; node++) {
152                 double q = uniform_random(0.0,1.0);
153                 if (q < prob_parent) {  //
154                     offspring_weight[layer].data[node] = father_weight[layer].data[node].clone();
155                 } else {  //
156                     offspring_weight[layer].data[node] = mother_weight[layer].data[node].clone();
157                 }
158             }
159         }
160         offspring.set_weight(offspring_weight);
161
162         return offspring;
163     }
164
165     public Matrix[] newWeight(individual blueprint){
166         Matrix[] offsprong_weight = new Matrix[blueprint.neural_type.length-1];
167         for (int layer = 0; layer < offsprong_weight.length; layer++) {
168             Matrix weight = new Matrix(blueprint.neural_type[layer+1],blueprint.neural_type[layer] ,false
                );
169             offsprong_weight[layer] = weight;
170         }
171         return offsprong_weight;
172     }
173
174     public void p2(ArrayList<individual> offspring_pool){
175         for (individual offspring:offspring_pool) {
176             Matrix[] nodeofchild = offspring.get_weight();
177               for (int layer = 0 ; layer < nodeofchild.length ; layer++ ){
178                   for (int node = 0 ; node < nodeofchild[layer].rows ; node++) {
179                         //            for each non-input node
180                       double q = uniform_random(0.0,1.0);
181
182                       if( q < prob_mul){
183                           mutation(offspring,layer,node);
184                       }
185                   }
186               }
187         }
188     }
189
190     public void mutation(individual offspring , int layer , int node){
191         Matrix[] a = offspring.get_weight();
192         for (int weightline = 0 ; weightline < a[layer].cols ; weightline ++ ){
193             double e = uniform_random(-1.0,1.0);
194             offspring.add_weight(layer,node,weightline , e);
195         }
196     }
197
198     public void add2N(ArrayList<individual> offspring_pool ){
199         while(offspring_pool.size() < num_population){
200             int pick = r.nextInt(0,num_population-1);
201             individual copy = population_pool[pick].clone();
202
203             offspring_pool.add(copy);
204         }
205     }
206
207     public void move2next_population(ArrayList<individual> offspring_pool) {
208         int i = 0 ;
209         for (individual offspring : offspring_pool) {
210             population_pool[i] =  offspring.clone();
211             i++;
212         }
```

```
213     }
214
215     public double uniform_random(double rangeMin , double rangeMax ){
216         return rangeMin + (rangeMax - rangeMin) * r.nextDouble();
217     }
218
219 }
```

2.3 individual.java

```java
import java.util.ArrayList;

public class individual  implements Cloneable {
    public final int[] neural_type;

    //sum of squared error at iteration n (sse)
    private final ArrayList<Double[]> error_n = new ArrayList<>();
    public double avg_error_n  ; // average sse of all epoch
    private double biases; // threshold connected : biases
    private Matrix[] layer_weight  ;
    private Double[][]  node  ;

    public individual(   String _neural_type ,double _biases) {
        String[] splitArray = _neural_type.split(",");
        int[] array = new int[splitArray.length];
        for (int i = 0; i < splitArray.length; i++) {
            array[i] = Integer.parseInt(splitArray[i]);
        }

        this.neural_type = array;
        init_Structor();
        this.biases = _biases;

    }
    private void init_Structor(){
        node = new Double[neural_type.length][];
        for (int i = 0; i < neural_type.length; i++) {
            node[i] = new Double[neural_type[i]];
        }

        layer_weight = new Matrix[neural_type.length-1];
        for (int layer = 0; layer < layer_weight.length; layer++) {
            Matrix weight = new Matrix(neural_type[layer+1],neural_type[layer] ,true);
            layer_weight[layer] = weight;
        }
    }

    public double eval(ArrayList<Double[]> _train_dataset, ArrayList<Double[]> _train_desired_data  ){
            unique_random  uq = new unique_random(_train_dataset.size());
            for(int data = 0; data < _train_dataset.size() ; data++) {
                //random  one dataset
                int ran_dataset_i = uq.get_num();
                //setup dataset value to input node
                for(int input_i = 0 ; input_i < neural_type[0] ; input_i ++){
                    node[0][input_i] = _train_dataset.get(ran_dataset_i)[input_i];
                }
                //cal  (input x weight) -> activation_Fn  for each neuron_node
                forward_pass();

                get_error(_train_desired_data.get(ran_dataset_i));
            }

            double sum = 0.0;
            for (Double[] doubles : error_n) {
                //  E(n) = 1/2   e^2   : sum of squared error at iteration n (sse)
                double error_output = 0.0;
                for (double error:doubles) {

                    error_output += Math.pow(error, 2);
                }
                sum += error_output;
            }
            // avg_E(n) = 1/N   E(n)   : avg (sse)
            avg_error_n =  sum / (error_n.size());

        error_n.clear();
        return avg_error_n;

    }

```

```
71      public void test(ArrayList<Double[]> _test_dataset,ArrayList<Double[]> _test_desired_data){
72          //setup input data
73          double t_p =0;
74          double t_n =0;
75          double f_p =0;
76          double f_n =0;
77          for(int test_i = 0; test_i < _test_dataset.size()-1 ; test_i++) {
78
79              //set dataset value to input node
80              for (int input_i = 0; input_i < neural_type[0]; input_i++) {
81                  node[0][input_i] = _test_dataset.get(test_i)[input_i];
82              }
83              forward_pass();
84
85              // class set
86              if(activation_fn(node[node.length-1][0])  > activation_fn(node[node.length-1][1])){
87                  node[node.length-1][0] = 1.0;
88                  node[node.length-1][1] = 0.0;
89              }else {
90                  node[node.length-1][0] = 0.0;
91                  node[node.length-1][1] = 1.0;
92              }
93
94              if(node[node.length-1][0].equals(_test_desired_data.get(test_i)[0]) && node[node.length
                    -1][0].equals(1.0) ) t_p++;
95              if(node[node.length-1][0].equals(_test_desired_data.get(test_i)[0]) && node[node.length
                    -1][0].equals(0.0)  ) t_n++;
96
97              if(!node[node.length-1][0].equals(_test_desired_data.get(test_i)[0])  && node[node.length
                    -1][0].equals(1.0)  ) f_p++;
98              if(!node[node.length-1][0].equals(_test_desired_data.get(test_i)[0])  && node[node.length
                    -1][0].equals(0.0)  ) f_n++;
99          }
100         // t_p       t_n    f_p   f_n
101         System.out.println(t_p+"\t"+t_n+"\t"+f_p+"\t"+f_n);
102         System.out.println( t_p/(t_p+f_p) +"\t"+  t_p/(t_p+f_n) +"\t"+  (t_p+t_n)/(t_p+t_n+f_p+f_n)) ;
103         error_n.clear();
104     }
105
106     private void forward_pass(){
107         for(int layer = 0; layer < neural_type.length-1 ; layer++) {
108
109             // W r_c X N r_1 = N+1 r_1
110             if(   layer_weight[layer].cols != node[layer].length){
111                 System.out.println("invalid matrix");
112                 return;
113             }
114
115             double  sum_input;
116             Double[] sum_inputnode = new Double[neural_type[layer+1]];
117
118             //mutiply matrix
119             for (int j = 0; j < neural_type[layer+1] ; j++){
120                 double sum=0;
121                 for(int k=0;k<node[layer].length;k++)
122                 {
123                     //w_ji : weight from input neuron j to neron i : in each layer
124                     sum += layer_weight[layer].data[j][k]  *  activation_fn( node[layer][k])  ;
125                 }
126                 // V_j = sum all input*weight i->j + biases
127                 sum_input = sum + biases;
128                 sum_inputnode[j] = sum_input;
129             }
130             // O_k  =  output of neuron_node k in each layer
131             node[layer+1] = sum_inputnode;
132
133         }
134     }
135
136     private void get_error(Double[] desired_data) {
137
138         int number_outputn_node  =   node[node.length-1].length;
```

```
        Double[] errors = new Double[number_outputn_node];
        for ( int outnode_j = 0 ; outnode_j < number_outputn_node ; outnode_j++) {
            //train_desired_data => d_j desired output for neuron_node j at iteration N // it have "one
                data"
            //e_j  = error at neuron j at iteration N
            double desired = desired_data[outnode_j];
            double getOutput =  activation_fn( node[node.length-1][outnode_j] )  ;
            errors[outnode_j] =  desired - getOutput  ;

        }
        error_n.add(errors);
    }

    public  Matrix[] get_weight(){
        return  layer_weight;
    }

    public  void set_weight(Matrix[] _newWeight){
        this.layer_weight = _newWeight;
    }

    public void add_weight(int layer , int node , int column , double value){
        layer_weight[layer].add(node,column,value);
    }

    public double activation_fn(Double x){
        //TODO
        return Math.max(0.01,x);  // leak relu
//        return (Math.exp(x)-Math.exp(-x))/(Math.exp(x)+Math.exp(-x)); // Tanh
//        return 1.0 / (1.0 + Math.exp(-x)); //sigmoid
//         return  x; // linear
    }

    @Override
    public individual clone() {
        try {
            individual clone = (individual) super.clone();
            // TODO: copy mutable state here, so the clone can't change the internals of the original
            return clone;
        } catch (CloneNotSupportedException e) {
            throw new AssertionError();
        }
    }

}
```

2.4 My library

2.4.1 Code of unique_random.java

```java
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;
public class unique_random {
    int size;
    List<Integer> list_number;
    public unique_random(int size){
        list_number = new ArrayList<>();
        this.size = size;
        for (int i = 0; i < size; i++) {
            list_number.add(i);
        }
        Collections.shuffle(list_number);

    }
    public int get_line(){
        int temp = list_number.get(0);
        list_number.remove(0);
        return temp;
    }
}
```

2.4.2 Code of Matrix.java

```java
import java.util.Random;

public class Matrix {
    double[][] data;
    int rows,cols;

    /**
     * W ji weight form input neuron   i to j
     * @param rows j node
     * @param cols i node
     */

    public Matrix(int rows, int cols , boolean random){
        data = new double[rows][cols];
        this.rows=rows;
        this.cols=cols;
        Random generator = new Random();

        if(random){
            for(int j=0;j<rows;j++)
            {
                for(int i=0;i<cols;i++)
                {
                    double ran = 0;
                    while(ran == 0){
                        ran = generator.nextDouble(-1,1);
                        data[j][i]=ran;
                    }
                }
            }
        }
    }

    public void add(int row, int col, double value) {
        this.data[row][col] += value;
    }
}
```

2.4.3 Code of Pair.java

```java
// Pair class
class Pair<U, V>
{
    public final U score;       // the first field of a pair
    public final V individual;      // the second field of a pair

    // Constructs a new pair with specified values
    Pair(U fitscore, V individual)
    {
        this.score = fitscore;
        this.individual = individual;
    }

    @Override
    // Checks specified object is "equal to" the current object or not
    public boolean equals(Object o)
    {
        if (this == o) {
            return true;
        }

        if (o == null || getClass() != o.getClass()) {
            return false;
        }

        Pair<?, ?> pair = (Pair<?, ?>) o;

        // call `equals()` method of the underlying objects
        if (!score.equals(pair.score)) {
            return false;
        }
        return individual.equals(pair.individual);
    }

    @Override
    // Computes hash code for an object to support hash tables
    public int hashCode()
    {
        // use hash codes of the underlying objects
        return 31 * score.hashCode() + individual.hashCode();
    }

    @Override
    public String toString() {
        return "(" + score + ", " + individual + ")";
    }

    // Factory method for creating a typed Pair immutable instance
    public static <U, V> Pair <U, V> of(U a, V b)
    {
        // calls private constructor
        return new Pair<>(a, b);
    }
}
```