,

Computer Assignment 4 Report by Pongsakorn Rattanapan. Student ID: 630610749

1 Prediction Benzene concentration using Particle Swarm Optimization

prediction Benzene concentration 5 day , 10 day

1.1 problem

Train Multilayer Perceptron โดยใช้ Particle Swarm Optimization (PSO) สำหรับการทำ prediction Benzene concentration โดยเป็นการ predict 5 วันล่วงหน้า และ 10 วันล่วงหน้า โดยให้ใช้ attribute เบอร์ 3,6,8,10,11,12,13 และ 14 เป็น input ส่วน desire output เป็น attribute เบอร์ 5 ให้ทำการทดลองกับ AirQualityUCI (Air Quality Data Set จาก UCI Machine learning Repository) โดยที่ data set นี้มีทั้งหมด 9358 sample และมี 14 attribute ดังนี้

0 Date (DD/MM/YYYY)

1 Time (HH.MM.SS)

2 True hourly averaged concentration CO in mg/m^3 (reference analyzer)

3 PT08.S1 (tin oxide) hourly averaged sensor response (nominally CO targeted)

4 True hourly averaged overall Non Metanic HydroCarbons concentration in microg/m^3 (reference analyzer)

5 True hourly averaged Benzene concentration in microg/m^3 (reference analyzer)

6 PT08.S2 (titania) hourly averaged sensor response (nominally NMHC targeted)

7 True hourly averaged NOx concentration in ppb (reference analyzer)

8 PT08.S3 (tungsten oxide) hourly averaged sensor response (nominally NOx targeted)

9 True hourly averaged NO2 concentration in microg/m^3 (reference analyzer)

10 PT08.S4 (tungsten oxide) hourly averaged sensor response (nominally NO2 targeted)

11 PT08.S5 (indium oxide) hourly averaged sensor response (nominally O3 targeted)

12 Temperature in °C

13 Relative Humidity %

14 AH Absolute Humidity

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

1.2 Design

ในการออกแบบระบบ Particle Swarm Optimization โดยในการทดลองของ Search space จะมีทั้งหมด 50 Particle โดยในแต่ละ Particle จะแทน weight ของ neural network หรือ Multi-Layer Perceptron

ดังนั้น position of Particle in search space จึงเป็นการแทน weight แต่ละเส้นของ Multi-Layer Perceptron ด้วยตำแหน่งใน dimension ต่างๆของ Particle ภายใน search space และหาก Particle อันไหน อยู่ในตำแหน่งที่ทำให้ได้ผลลัพธ์ที่ดี ก็จะทำให้ Particle ที่อยู่รอบๆ เข้ามาหาได้มากกว่า โดยที่ Particle แต่ละตัวก็จะมี position (unique weight) ที่ทำให้การทำนายผล AirQualityUCI ของทั้ง 8-input ว่า Benzene concentration ในอีก 5 วันล่วงหน้า และ 10 วันล่วงหน้า เป็นเท่าไหร่ ซึ่งจะมีหากทำนายค่าได้แม่นยำ ก็จะมีค่า error ที่น้อย เมื่อนำไปผ่านสมการ scaling เพื่อแปลงเป็นค่า fitness

$$Fitness = MeanAbsoluteError \tag{1}$$

จะได้ว่าหาก Particle ในตำแหน่งไหน ที่มีค่า error น้อยก็จะมีค่า fitness ที่น้อย และเป็น Particle ที่เข้าใกล้ solution

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)
MLP ที่ใช้ในการทดลองนี้จะเป็น 8-5-1 โดยมี input 8 features และ output 1 node
โดย MLP จะรับค่า weight จาก dimension ของ Particle แต่ละตัวใน search space เพื่อหาค่า MAE , fitness
ซึ่งจำนวน dimension ของ Particle ขึ้นอยู่กับจำนวนของเส้น weight ใน MLP
(Particle for 8-5-1 have 45 dimension)

by van den Bergh (2002) ใช้ Cognitive coefficient = 1.496180 , Social coefficient = 1.496180 และ Inertia coefficient = 0.729844

โดยในแต่ละ $timestep$ ทำตามขั้นตอนต่อไปนี้

1.3.1 Initial Particle

หากยังไม่มี Particle ใน Search space ทำการสร้าง Particle mlp โดยสุ่ม Position ของแต่ละ Particle ใน dimension ต่างๆ ของ Search space โดยขนาดของ Particle_pool คือ 50 Particle

1.3.2 Particle evaluate fitness

ในแต่ละ Position ของ particle นำ train_dataset ทำการ evaluate feed forward neural network โดยใช้ตำแหน่งที่ Particle นั้นอยู่ใน Search space เป็นค่า weight ได้ผลลัพธ์เป็นค่า predict ของ individual และคิด error Mean Absolute Error (MAE) จาก train_desired_dataset ซึ่งยิ่งมีค่า error น้อยแสดงว่า predict ได้อย่างแม่นยำ

```
        error_MAE =   engine.eval( train_dataset  , particle.position);
```

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

Update if Particle self better ในแต่ละ Particle หากค่า ค่า fitness ของ Particle ใน position ตอนนี้ นั้นดีกว่า pbest (fitness ที่ดีที่สุดของตัวเอง) ทำการ update pbest และ position pbest ของตัวเอง by this code:

```
            if(particle.fitness <  particle.pbest){
                particle.pbest = particle.fitness;
                particle.position_pbest = particle.position;
            }
```

Update if Particle are better in Search space หาก Particle มีค่า fitness ที่ดีกว่า gbest (fitness จากตำแหน่งดีสุดในตอนนี้) ทำการ update gbest และ position gbest by this code:

```
            if(particle.fitness < gbest){
                gbest = particle.fitness;
                 position_gbest = particle.position;
            }
```

1.3.3 Update Vector

ในแต่ละ Particle จะพยายามเข้าหา position ที่ทำให้ค่า fitness ดีที่สุดในที่นี้คือ pbest และ gbest โดยคำนวณทิศทาง(ความเร็ว) ของแต่ละ $Particle_i$ ได้จาก

$$V_i(t) = wV_i(t-1) + p_1(x_i.pbest - x_i.position) - p_2(gbest - x_i.position))$$

- $V_i(t)$: ความเร็วของ particle i ใน dimension ต่างๆ ใน next time step
- $V_i(t-1)$: ความเร็วของ particle i ใน dimension ต่างๆ ในตอนนี้
- w : Inertia coefficient
- p_1 : Cognitive coefficient
- p_2 : Social coefficient
- $x_i.position$: ตำแหน่งปัจจุบันของ particle i
- $x_i.pbest$: ตำแหน่ง particle i ที่มีค่า fitness ดีสุด
- $gbest$: ตำแหน่งที่ให้ค่า fitness ดีสุดในตอนนี้

1.3.4 Update Particle Position

ในแต่ละ Particle จะขยับเข้า position ที่ดีขึ้นใน dimension ต่างๆของ Search space ด้วย velocity $V_i(t)$ จาก step ก่อนหน้า และทำการ update ด้วยสมการดังนี้

- $X_i(t) = X_i(t-1) + V_i(t))$

1.3.5 Next time step

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:

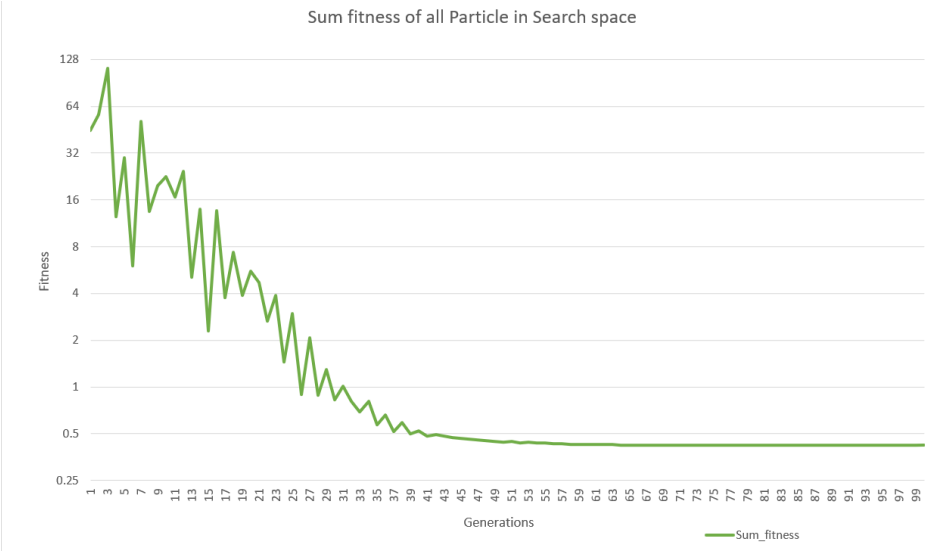
```
        t = t + 1
```

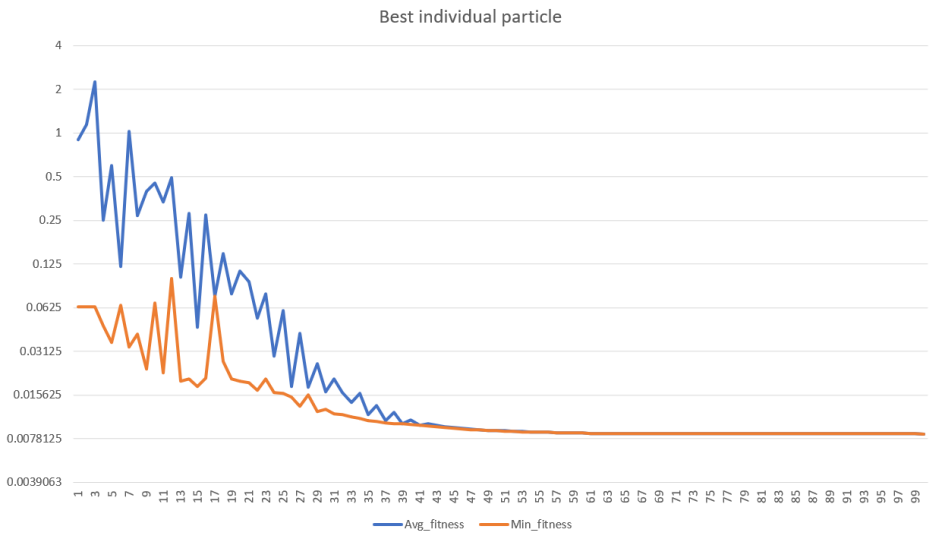
1.3.6 until coverage

จนกว่า Particle ใน Search space จะเจอ solution position , velocity ของ particle ไม่ขยับ , ได้ค่า fitness ที่รับได้ หรือ ทำถึงจำนวนรอบที่ต้องการ ทำการเริ่มรอบใหม่ไปเรื่อยๆ

1.4 result

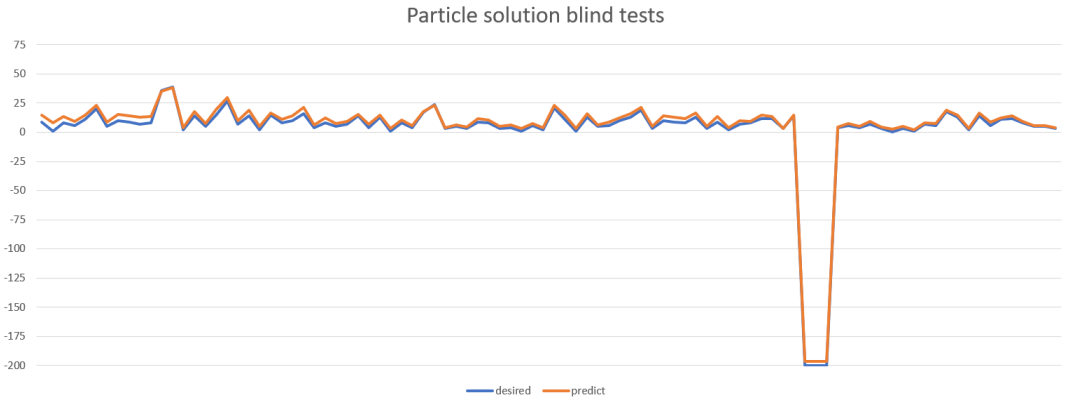
เมื่อผ่านไปในแต่ละ Time step จะเห็นได้ว่า ผลรวมของ fitness จากทุก Particle มีค่าลดลง กล่าวคือ แต่ละ Particle เข้าใกล้ position ของ solution ที่ต้องการมากขึ้น ดังตาราง





จะเห็นได้ว่าค่า Average fitness ของ Particle ทั้งหมดนั้นค่อยๆลดลง ในทางเดียวกับที่ Minimum fitness ที่ลดลง เนื่องจากมี Particle ที่มี Position ที่ให้ค่า fitness ที่ดีคอยดึงให้ Particle อื่นๆ ขยับเข้ามาใกล้ จึงทำให้ค่า Average fitness นั้นค่อยๆลดลง

และเมื่อผ่านไปครบ 100 Time step จะได้ Particle ที่ให้ค่า fitness ต่ำที่สุดจากทั้งหมดคือ best solution mlp โดยเมื่อนำ solution ไปทำการทดสอบกับ test_dataset ซึ่งเป็น blindtest จาก 10% cross validation ได้ผลลัพธ์ดังนี้

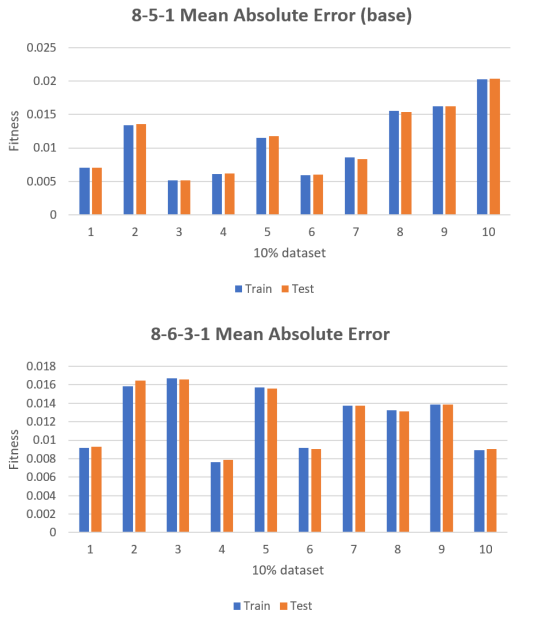


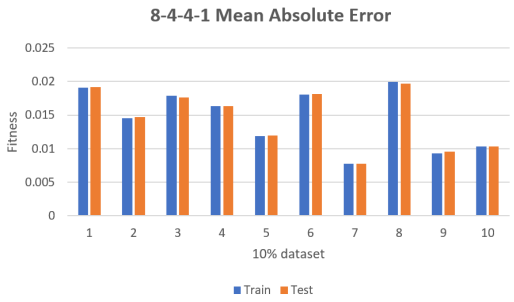
then try changes hidden nodes to test the impact. (different initialize weights between -1.0 to 1.0)

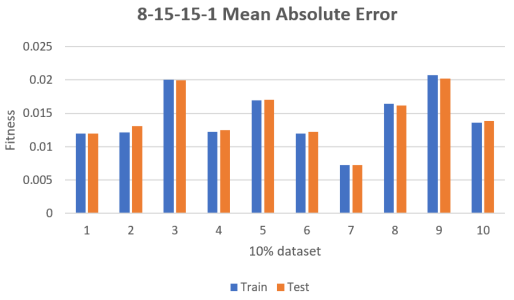
ทดสอบ validity ของ network โดยทำการเปลี่ยนแปลงจำนวน hidden layer และ nodes ดังตารางนี้

neural test	
Test	Neural Type
base	8-5-1
A	8-4-4-1
B	8-6-3-1
C	8-15-15-1

โดยเปรียบเทียบจาก Particle ที่ให้ค่า Minimum fitness ดีที่สุดในตอน train จากนั้นนำไปทำการ blind test







1.5 Particle Swarm analysis

จะเห็นได้ว่า MLP ที่มีโครงสร้าง 8-5-2 เมื่อใช้ Particle ในการ Search space จนได้ Position ที่ดีที่สุด จะได้ว่า Mean Absolute Error อยู่ที่ 0.0051 ซึ่งอาจจะเป็นเพราะว่าเป็นจำนวนโครงสร้างที่พอดี ซึ่ง 8-4-4-1 กับ 8-6-3-1 ใน test case โดยรวมทั้งหมดอาจจะยังไม่แม่นยำมากนัก ซึ่งอาจจะเป็นเพราะจำนวน layer ที่มี 2 Hidden layer เทียบกับ จำนวน node ต่อ layer ที่น้อยเกินไป ซึ่งในจำนวน Hidden layer ที่เยอะขึ้น 8-15-15-1 โดยรวมทำได้ดีกว่า อาจจะเป็นเพราะจำนวน node ในแต่ละ layer ที่มากเพียงพอที่จะส่งค่าเด่นๆ ไปยัง layer ต่อไปได้ไม่เหมือน 8-4-4-1 , 8-6-3-1

แต่โดยรวมแล้ว จำนวน Hidden layer หรือ node ต่อ layer ไม่จำเป็นต้องเยอะมาก อย่างเช่น 8-5-1 ก็มีความแม่นยำที่มากเพียงพอแล้ว

1.6 Conclusion

Particle Swarm Optimization ในการเพิ่มค่า fitness ของ MLP โดยการใช้ Particle ในการ Search space เพื่อหา Position หรือ weight dimension ที่ทำให้ MLP นั้นมีค่า fitness ที่ดีขึ้น และเมื่อมี Particle ตัวใด เจอ Position ที่ดีก็จะพยายามดึง Particle เข้ามาใกล้ๆ จนทำให้ Particle แต่ละตัวมีค่า fitness ที่ดีขึ้นตามไปด้วย

และการใช้ Particle ในการ Search space หา weight ใน dimension ต่างๆ เพื่อนำไปให้ MLP evaluate ในการหา fitness เพื่อทำให้ค่า fitness นั้นดีขึ้น จากการ ทดลองพบว่า Best individual ตัวที่ดีที่สุดจาก population ทั้งหมดมีความแม่นยำ Mean Absolute Error อยู่ที่ 0.0051 และในทุก test case MAE อยู่ที่ประมาณ 0.0109 โดยใน โปรเจคนี้มีค่าหลายอย่างที่สามารถปรับได้ ไม่ว่าจะเป็น biased , activation function ที่ใช้ ที่สามารถปรับเปลี่ยนได้

2 Source code

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

2.1 Main.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/AirQualityUCI_dataset.txt");
            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("\t");

                Double[] temp_input = new Double[8];
                Double[] ans = new Double[1];
        //lazy  min max norm
                temp_input[0] = (Double.parseDouble(eachLine[3])+200)/(2050.0+200);
                temp_input[1] = (Double.parseDouble(eachLine[6])+200)/(2300.0+200);
                temp_input[2] = (Double.parseDouble(eachLine[8])+200)/(2700.0+200);
                temp_input[3] = (Double.parseDouble(eachLine[10])+200)/(2800.0+200);
                temp_input[4] = (Double.parseDouble(eachLine[11])+200)/(2600.0+200);
                temp_input[5] = (Double.parseDouble(eachLine[12])+200)/(50.0+200);
                temp_input[6] = (Double.parseDouble(eachLine[13])+200)/(90.0+200);
                temp_input[7] = (Double.parseDouble(eachLine[14])+200)/(3.0+200);

                ans[0] =  (Double.parseDouble(eachLine[5])+200)/(70.0+200);

                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("=====================================================");
            Swarm ga = new Swarm("8,5,1",  0, 50 , 100 );
            System.out.println("train: " + test_i);
            ga.settraindata( train_dataset.get(test_i), train_desired_data.get(test_i));
            ga.particle_search();
            System.out.println("test: " + test_i);
            ga.test(test_dataset.get(test_i), test_desired_data.get(test_i));
            System.out.println("=====================================================");
        }
    }
}
```

2.2 Swarm.java

```java
import java.util.ArrayList;

public class Swarm {

    String neural_type;
    double biases;

    double p1 = 1.496180;
    double p2 = 1.496180;
    double inertia_w = 0.729844;

    int num_particle;
    int maxEpoch;
    Engine_eval engine;
    double gbest = Double.MAX_VALUE;
    Matrix[] weight_gbest;

    particle[] particle_pool;

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

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

    public Swarm(String _mlp , double _biases , int _population , int _maxGeneration){
        this.neural_type = _mlp;
        this.biases = _biases;
        this.num_particle = _population;
        this.maxEpoch = _maxGeneration;
        this.engine = new Engine_eval(_mlp,_biases);

    }

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

    public void init_Particle(){
        particle[] p =  new particle[num_particle];
        for(int indi = 0; indi < num_particle; indi++){
            particle indiler = new particle( neural_type ,biases );
            p[indi] = indiler;
        }
        this.particle_pool = p;

        this.weight_gbest = newWeight(particle_pool[0]);
    }

    public void particle_search(){

        init_Particle();

        for (int epoch = 0; epoch < maxEpoch; epoch++){

            Double[] result =  particle_eval(train_dataset,train_desired_data);
            if(epoch==maxEpoch-1){
//                System.out.print(result[0] + "\t" + result[1] + "\t" + result[2] + "\t" );
                System.out.print(result[2] + "\t" );
            }

            //step 4
            updateVector();

            //step 5
            updatePosition();
        }
    }

```

```
    public void test(ArrayList<Double[]> dataset, ArrayList<Double[]> desired_data){
        particle best_solution =  scoreBoard.get(scoreBoard.size()-1).particle;
        engine.test(dataset,desired_data,best_solution.get_weight());
    }

    public void updateVector(){
        for (particle particle : particle_pool) {
            Matrix[] particle_weight = particle.get_weight();

            Matrix[] vector_weight = newWeight(particle);
            for (int layer = 0 ; layer < particle.layer_weight.length ; layer++) {
                Matrix p_w = Matrix.sub_matrix(particle.weight_pbest[layer],particle_weight[layer]);
                Matrix learn_m_p = Matrix.mul_matrix(p_w, p1);

                Matrix g_w = Matrix.sub_matrix(weight_gbest[layer],particle_weight[layer]);
                Matrix learn_m_g = Matrix.mul_matrix(g_w, p2);

                Matrix old_vector_w = Matrix.mul_matrix(particle.vector_weight[layer],inertia_w);

                Matrix vector_w = Matrix.plus_3matrix(old_vector_w,learn_m_p,learn_m_g);

                vector_weight[layer] = vector_w;
            }
            particle.set_vector_weight(vector_weight);
        }
    }

    public void updatePosition(){
        for (particle particle : particle_pool) {

            Matrix[] particle_weight = particle.get_weight();
            Matrix[] vector_weight = particle.get_vector_weight();

            Matrix[] new_particle_weight = newWeight(particle);
            for (int layer = 0 ; layer < particle.layer_weight.length ; layer++) {
                Matrix a = Matrix.plus_matrix(particle_weight[layer], vector_weight[layer]);
                new_particle_weight[layer] = a;
            }
            particle.set_weight(new_particle_weight);
        }
    }

    public Double[] particle_eval(ArrayList<Double[]> dataset , ArrayList<Double[]> desired_data){
        double sum_fit = 0;
        double avg_fit;
        double min_fit = Double.MAX_VALUE;

        //eval fitness
        for (particle particle : particle_pool) {
            Matrix[] positions = particle.get_weight();
            double error_mlp =   engine.eval(dataset , desired_data , positions );

            double fitness = error_mlp;
            if(fitness < min_fit ){
                min_fit = fitness;
            }

            sum_fit += fitness;
            //step 3.1
            if(fitness < particle.pbest){
                particle.set_pbest(fitness);
                particle.set_weight_pbest(particle.get_weight());
            }
            //step 3.2
            if(fitness < gbest){
                gbest = fitness;
                weight_gbest = particle.get_weight();
                Pair<Double , particle> good_particle = new Pair<>(fitness, particle);
                scoreBoard.add(good_particle);
            }

        }
```

```
        avg_fit = sum_fit / num_particle;
        return new Double[]{sum_fit, avg_fit, min_fit};
    }

    public Matrix[] newWeight(particle blueprint){
        Matrix[] new_weight = new Matrix[blueprint.neural_type.length-1];
        for (int layer = 0; layer < new_weight.length; layer++) {
            Matrix weight = new Matrix(blueprint.neural_type[layer+1],blueprint.neural_type[layer] ,false
                );
            new_weight[layer] = weight;
        }
        return new_weight;
    }


}
```

2.3 particle.java

```java
public class particle implements Cloneable {
    public final int[] neural_type;

    public Matrix[] layer_weight  ;
    public Matrix[] vector_weight  ;

    public double pbest = Double.MAX_VALUE;
    Matrix[] weight_pbest;


    public particle(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();

    }
    private void init_Structor(){

        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;
        }

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

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

    }

    public  Matrix[] get_weight(){
        return  layer_weight.clone();
    }
    public  Matrix[] get_vector_weight(){
        return  vector_weight.clone();
    }

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

    public  void set_vector_weight(Matrix[] _newWeight){
        this.vector_weight = _newWeight;
    }

    public  void set_pbest(double _pbest){
        this.pbest = _pbest;
    }
    public  void set_weight_pbest(Matrix[] _weight_pbest){
        this.weight_pbest = _weight_pbest;
    }

    @Override
    public particle clone() {
        try {
            particle clone = (particle) 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 Engine_eval.java

```java
import java.util.ArrayList;

public class Engine_eval{
    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

    public Matrix[] layer_weight  ;

    private Double[][]  node  ;

    public Engine_eval(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]];
        }
    }

    public double eval(ArrayList<Double[]> _train_dataset, ArrayList<Double[]> _train_desired_data ,
        Matrix[]  _layer_weight){
        this.layer_weight = _layer_weight;
        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.abs(error);
            }
            sum += error_output;
        }
        // MAE(n) = 1/N  E(n)  : avg (sse)
        avg_error_n =  sum / (error_n.size());
        error_n.clear();
        return avg_error_n;

    }


```

```
    public void test(ArrayList<Double[]> _test_dataset,ArrayList<Double[]> _test_desired_data  , Matrix[]
          _layer_weight){
        //setup input data
        this.layer_weight = _layer_weight;
        for(int test_i = 0; test_i < _test_dataset.size()-1 ; test_i++) {

            //set dataset value to input node
            for (int input_i = 0; input_i < neural_type[0]; input_i++) {
                node[0][input_i] = _test_dataset.get(test_i)[input_i];
            }
            forward_pass();

            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++) {
                double desired = _test_desired_data.get(test_i)[0];
                errors[outnode_j] =  desired -node[node.length-1][outnode_j];
            }
            error_n.add(errors);


            double d = (_test_desired_data.get(test_i)[0]*(70+200))-200;
            double g = (node[node.length-1][0]*(70+200))-200;
//            System.out.println("desired:" + (int)d + " get: "+ g + "\t error_n: " + Math.abs(d-g));
        }
        double sum = 0.0;
        for (Double[] doubles : error_n) {
            // E(n) = 1/2   e^2   : sum of squared error at iteration n (sse)

            sum += Math.abs(doubles[0]);
        }
        // avg_E(n) = 1/N   E(n)  : avg (sse)
        avg_error_n =  sum / (error_n.size());

//        System.out.println("avg_error_n final : " + avg_error_n);
        System.out.println(avg_error_n);

        error_n.clear();
    }

    private void forward_pass(){
        for(int layer = 0; layer < neural_type.length-1 ; layer++) {

            // W r_c X N r_1 = N+1 r_1
            if(   layer_weight[layer].cols != node[layer].length){
                System.out.println("invalid matrix");
                return;
            }

            double  sum_input;
            Double[] sum_inputnode = new Double[neural_type[layer+1]];

            //mutiply matrix
            for (int j = 0; j < neural_type[layer+1] ; j++){
                double sum=0;
                for(int k=0;k<node[layer].length;k++)
                {
                    //w_ji : weight from input neuron j to neron i : in each layer
                    sum += layer_weight[layer].data[j][k]  *  activation_fn( node[layer][k])  ;
                }
                // V_j = sum all input*weight i->j + biases
                sum_input = sum + biases;
                sum_inputnode[j] = sum_input;
            }
            // O_k  =  output of neuron_node k in each layer
            node[layer+1] = sum_inputnode;

        }
    }

    private void get_error(Double[] desired_data) {

```

```
        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 =  node[node.length-1][outnode_j]   ;
            errors[outnode_j] =  desired - getOutput  ;

        }
        error_n.add(errors);
    }

    public double activation_fn(Double x){
        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
    }


}
```

2.5 My library

2.5.1 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_num(){
        int temp = list_number.get(0);
        list_number.remove(0);
        return temp;
    }
}
```

2.5.2 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 static Matrix plus_matrix(Matrix a, Matrix b) {
        Matrix temp=new Matrix(a.rows,a.cols , false);
        for(int j=0;j<a.rows;j++)
        {
            for(int i=0;i<a.cols;i++)
            {
                temp.data[j][i]=a.data[j][i]+b.data[j][i];
            }
```

```
        }
        return temp;
    }

    public static Matrix plus_3matrix(Matrix a, Matrix b , Matrix c) {
        Matrix temp=new Matrix(a.rows,a.cols , false);
        for(int j=0;j<a.rows;j++)
        {
            for(int i=0;i<a.cols;i++)
            {
                temp.data[j][i]=a.data[j][i]+b.data[j][i]+c.data[j][i];
            }
        }
        return temp;
    }

    public static Matrix sub_matrix(Matrix a, Matrix b) {
        Matrix temp=new Matrix(a.rows,a.cols , false);
        for(int j=0;j<a.rows;j++)
        {
            for(int i=0;i<a.cols;i++)
            {
                temp.data[j][i]=a.data[j][i]-b.data[j][i];
            }
        }
        return temp;
    }

    public static Matrix mul_matrix(Matrix a, double mul) {
        Matrix temp=new Matrix(a.rows,a.cols , false);
        for(int j=0;j<a.rows;j++)
        {
            for(int i=0;i<a.cols;i++)
            {
                temp.data[j][i]=a.data[j][i]*mul;
            }
        }
        return temp;
    }
    public void add(int row, int col, double value) {
        this.data[row][col] += value;
    }
}
```

2.5.3 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);
    }
}
```