

# Assignment 4 Report

Tanat Tangun 630610737

November 2022

This report is about the result of my implementation of Particles Swarm Optimization (PSO) for optimizing the MLP on Rust language for 261456 - INTRO COMP INTEL FOR CPE class assignment. If you are interested to know how I implement PSO and use it to optimize the MLP, you can see the source code on my Github repository or in this document appendix.

## Problem

Given the AirQualityUCI dataset from UCI machine learning repository which has 9358 samples and 14 attributes as follows:

1. Date (DD/MM/YYYY)
2. Time (HH.MM.SS)
3. True hourly averaged concentration CO in  $\text{mg}/\text{m}^3$  (reference analyzer)
4. PT08.S1 (tin oxide) hourly averaged sensor response (nominally CO targeted)
5. True hourly averaged overall Non Metanic HydroCarbons concentration in  $\text{microg}/\text{m}^3$  (reference analyzer)
6. **True hourly averaged Benzene concentration in  $\text{microg}/\text{m}^3$  (reference analyzer)**
7. PT08.S2 (titania) hourly averaged sensor response (nominally NMHC targeted)
8. True hourly averaged NOx concentration in ppb (reference analyzer)
9. PT08.S3 (tungsten oxide) hourly averaged sensor response (nominally NOx targeted)
10. True hourly averaged NO2 concentration in  $\text{microg}/\text{m}^3$  (reference analyzer)
11. PT08.S4 (tungsten oxide) hourly averaged sensor response (nominally NO2 targeted)
12. PT08.S5 (indium oxide) hourly averaged sensor response (nominally O3 targeted)
13. Temperature in  $^{\circ}\text{C}$
14. Relative Humidity (%)
15. AH Absolute Humidity

We want to use the underlined attributes 4, 7, 9, 11, 12, 13, 14, 15 to predict attribute 6 (benzene concentration) in next 5 days and next 10 days.

## Dataset Preparation

We will refer to the underlined attributes as “features” and attribute 6 (benzene concentration) as “desired output”. The dataset preparation process will follow these steps (implementation on source code 7):

1. Load the dataset from the csv file and removing any samples that has a missing value (missing value are tagged with -200 value).
2. Match each features with desired output of the next 5 and 10 days, then we will get an array of tuple (features, next 5 days desired output) and tuple (features, next 10 days desired output).

The array of tuple (features, next 5 days desired output) and tuple (features, next 10 days desired output) will be used for training our MLP.

# Particles Swarm Optimization

## Particle Representation

A particle is represented by a list of weights and biases of MLP. We use weights and bias of top node to bottom node of each layer to create one individual, for an example: from 3-2-1 network in fig. 1 a particle is represented by (w1, w2, w3, b1, w4, w5, w6, b2, w7, w8, b3).

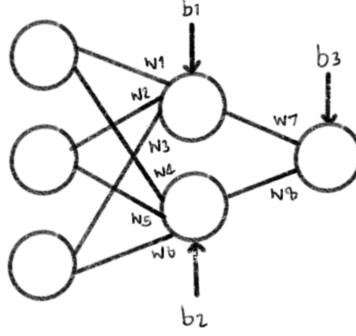


Figure 1: The 3-2-1 network.

## Full Process

Using 10% cross-validation, and preprocess each iteration training and validation set with min-max normalization. The min-max normalization process is done by for each feature  $f$  on training set we find  $\max(f)$  and  $\min(f)$  then for each datapoint  $f_x$  we compute new datapoint on both training set and validation set  $f'_x = \frac{f_x - \min(f)}{\max(f) - \min(f)}$ , this will guarantee that we applied the min-max normalization using  $\min$  and  $\max$  from training set on both training set and validation set. Next, for each cross-validation iteration we follow the local best method described on [Aue13] page 138 which is (implementation on source code 5 and 6):

1. Initialize the particles population  $P(t)$  at  $t = 0$  which has 5 groups of 4 particles and for each particle we set the weights to a random number in range  $[-1.0, 1.0]$ , and bias of each node to 1.0.
2. For each group  $j$  and each particle  $i$  in group  $j$  do:
  - (a) Evaluate its performance  $F$  using its current position  $x_i(t)$  through all samples in training set.
  - (b) Compare evaluation result from (a) with its best evaluation result  $\text{pbest}_i$ .
    - if**  $F(x_i(t)) < \text{pbest}_i$  **then**
    - $\text{pbest}_i = F(x_i(t))$
    - $x_i^{\text{pbest}} = x_i(t)$
    - end if**
  - (c) Compare evaluation result from (a) with the group's  $\text{lbest}_j$ .
    - if**  $F(x_i(t)) < \text{lbest}_j$  **then**
    - $\text{lbest}_j = F(x_i(t))$
    - $x_{\text{lbest}_j} = x_i(t)$
    - end if**
  - (d) Update the speed of  $i$  by using following equation:

$$v_i(t) = v_i(t-1) + \rho_1(x_i^{\text{pbest}} - x_i(t)) + \rho_2(x_{\text{lbest}_j} - x_i(t))$$

where  $\rho_1 = r_1 c_1$  and  $\rho_2 = r_2 c_2$  with  $c_1 = 1.0$ ,  $c_2 = 1.5$  and  $r_1, r_2$  are a random number from uniform distribution of  $(0, 1)$

- (e) Update  $x_i$  by  $x_i(t) = x_i(t-1) + v_i(t)$  and set  $t = t + 1$

3. Repeat step 2. until  $t = 100$ .

## Training Result

For both next 5 days dataset and next 10 days dataset, we will experiment with 3 models to see if their training result will have any significant differences in training time and MAE (mean absolute error). The 3 models (implementation on source code 5) are

- **air-8-4-1**: The base model that contains 8 input nodes, 1 hidden layer with 4 nodes, and 1 output node. The result is shown on fig. 2 and fig. 3
- **air-8-1-1**: A smaller model with 8 input nodes, 1 hidden layer with 1 nodes, and 1 output node. The result is shown on fig. 4 and fig. 5
- **air-8-8-4-1**: A larger model with 8 input nodes, 2 hidden layers with 8 and 4 nodes, and 1 output node. The result is shown on fig. 6 and fig. 7

which the output node use linear activation function and other nodes use relu activation function. We use Rust compiler with release profile to build and run all training.

## Analysis

From table 1 and table 2, we can see that every model train on both next 5 days dataset and next 10 days dataset have a similar validation set MAE and similar training process as we can see in fig. 2, fig. 3, fig. 4, fig. 5, fig. 6 and fig. 7. The training process from those figures shows that every model seem to converge to MAE  $\approx 5$  in less than  $t = 25$  and can't find a position to make MAE lower. The reason why every model can not make MAE lower maybe is because this dataset need a much more complex MLP structure to create a better regression model. Next, the training time of each model is less with the less complex it is and more with the more complex it is, showing that PSO training times correlate with MLP complexity.

Model	Training Time (seconds)	Mean Absolute Error (MAE)
air-8-4-1	62.233	5.194
air-8-1-1	58.504	5.184
air-8-8-4-1	81.596	5.216

Table 1: Training time and validation set MAE of next 5 days dataset (red line on fig. 2b, fig. 4b, and fig. 6b) of each model.

Model	Training Time (seconds)	Mean Absolute Error (MAE)
air-8-4-1	61.401	5.107
air-8-1-1	53.990	5.122
air-8-8-4-1	78.985	5.256

Table 2: Training time and validation set MAE of next 10 days dataset (red line on fig. 3b, fig. 5b, and fig. 7b) of each model.

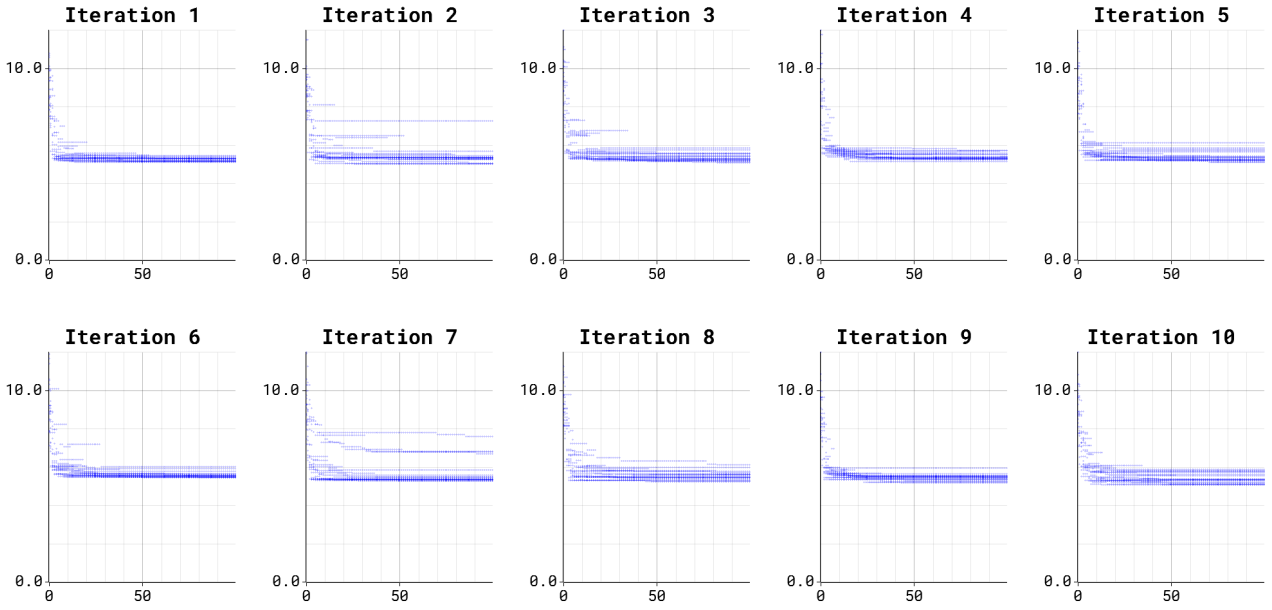
## Summary

Particles Swarm Optimization (PSO) can be used for training MLP with an okay performance within a reasonable times as shown on our experiment. Training the regression MLP for AirQualityUCI dataset seem to be a challenging problem with PSO, this may need a further investigation. Finally, Rust is a great language to implement PSO with how fast it is and how easy it is to write a memory-safe program.

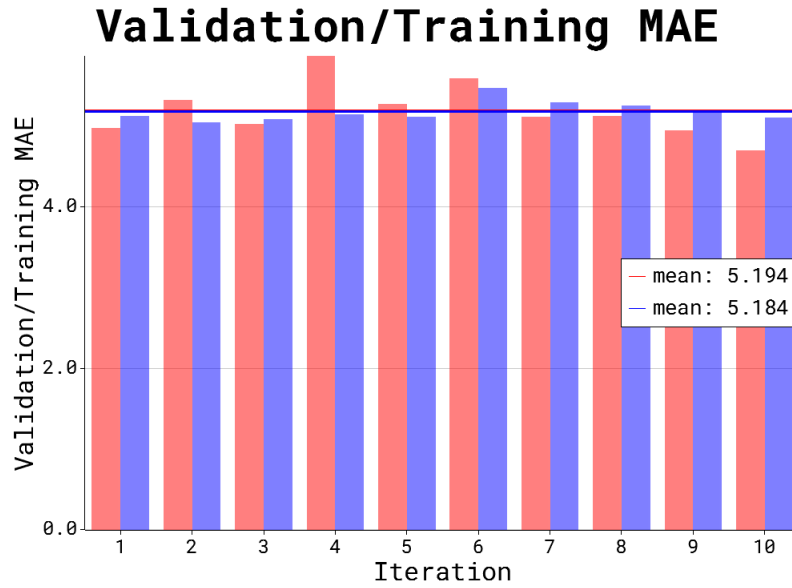
## References

- [Aue13] Sansanee Auephanwiriyaikul. *Introduction to Computational Intelligence for Computer Engineering*. 2013. URL: [http://myweb.cmu.ac.th/sansanee.a/Intro\\_CI\\_withwatermark.pdf](http://myweb.cmu.ac.th/sansanee.a/Intro_CI_withwatermark.pdf). (accessed: 8.11.2022).

## Appendix

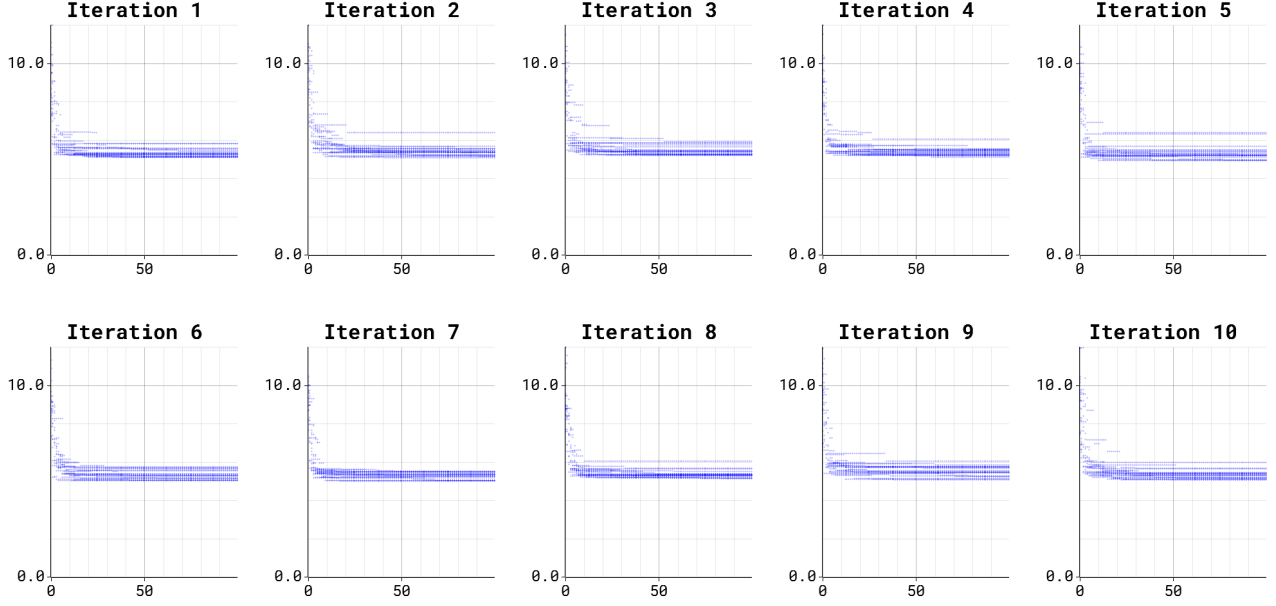


(a) The training process of each cross-validation iteration: x-axis is  $t$  value, y-axis is the evaluation result (MAE), and each blue dot is a particle at time  $t$  with its MAE.

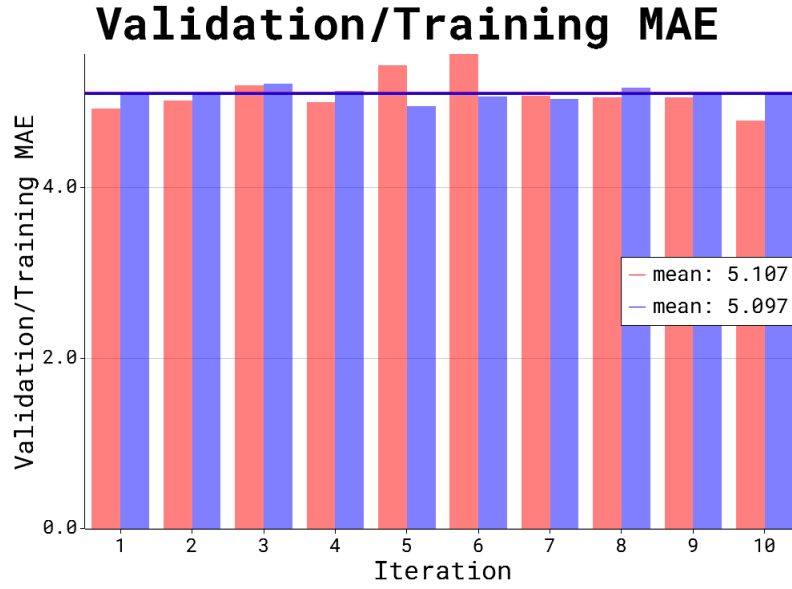


(b) The best particle from each cross-validation iteration MAE on training set (blue) and validation set (red).

Figure 2: Training result of air-8-4-1 of next 5 days dataset with 63.233 seconds used for training.

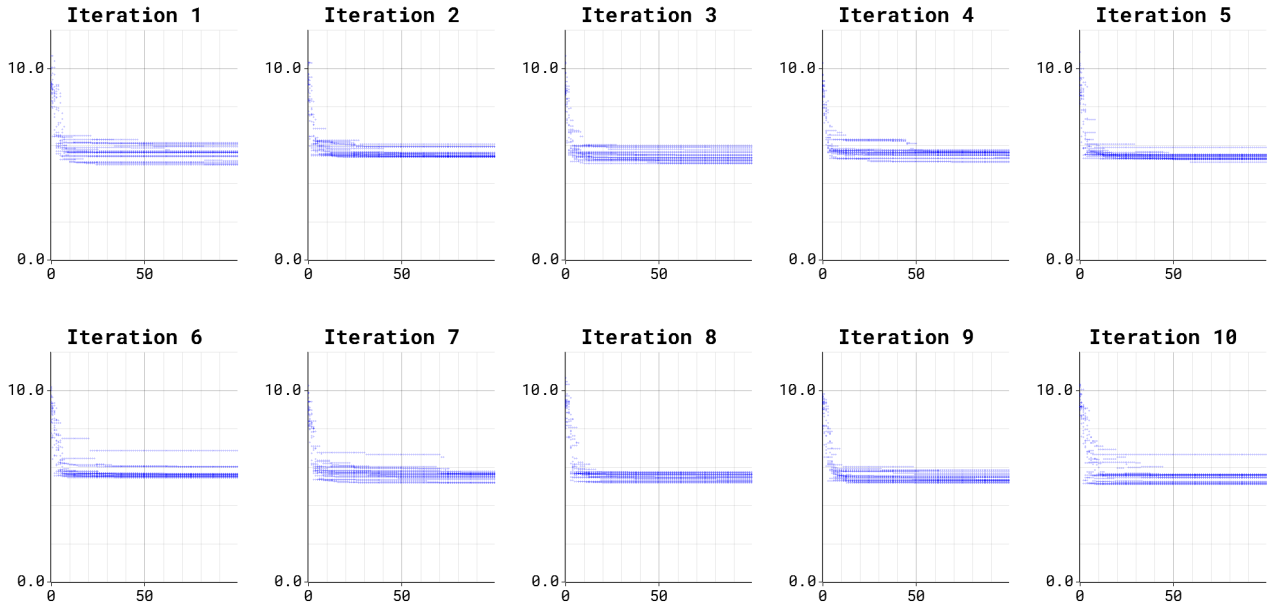


(a) The training process of each cross-validation iteration: x-axis is  $t$  value, y-axis is the evaluation result (MAE), and each blue dot is an particle at time  $t$  with its MAE.

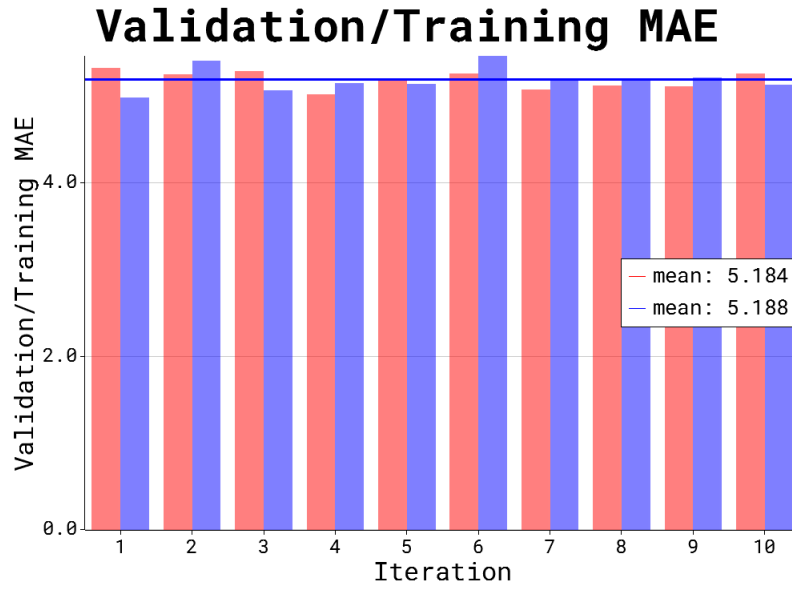


(b) The best particle from each cross-validation iteration MAE on training set (blue) and validation set (red).

Figure 3: Training result of air-8-4-1 of next 10 days dataset with 61.401 seconds used for training.

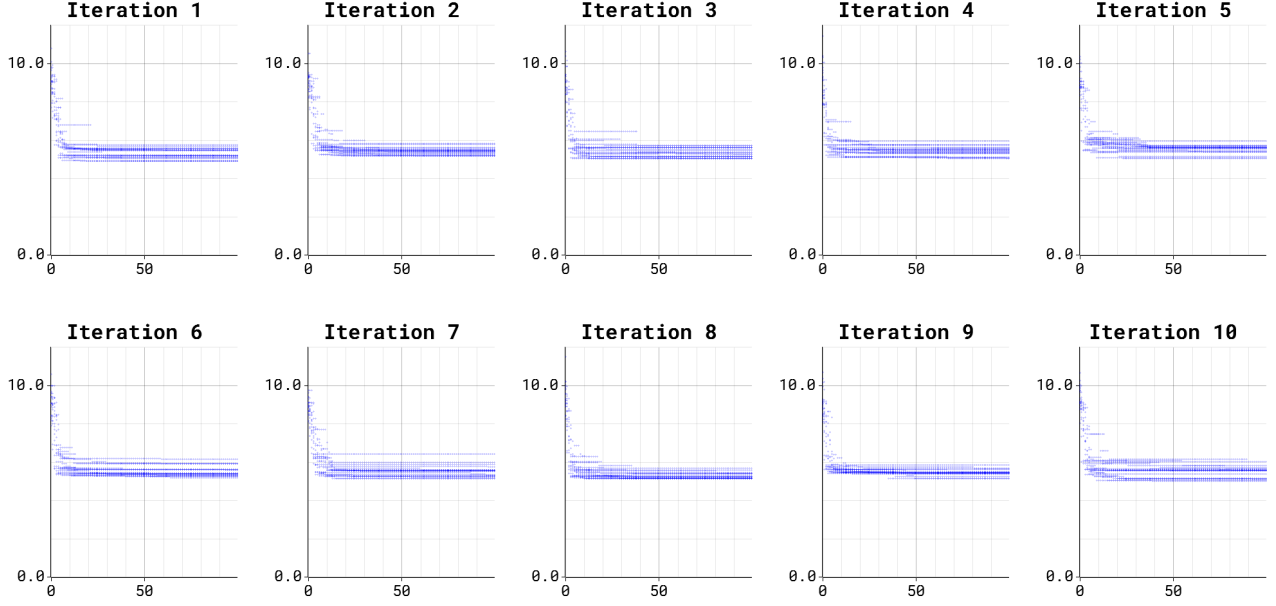


(a) The training process of each cross-validation iteration: x-axis is  $t$  value, y-axis is the evaluation result (MAE), and each blue dot is a particle at time  $t$  with its MAE.

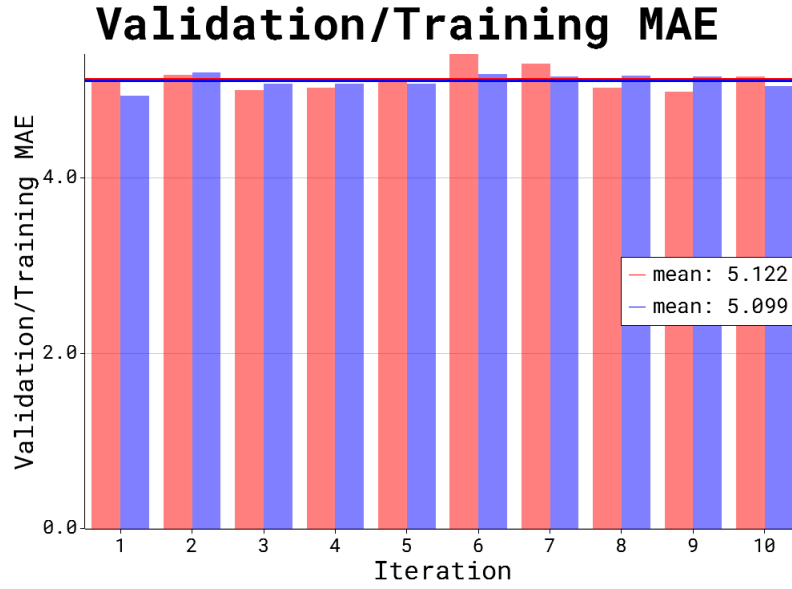


(b) The best particle from each cross-validation iteration MAE on training set (blue) and validation set (red).

Figure 4: Training result of air-8-1-1 of next 5 days dataset with 58.504 seconds used for training.

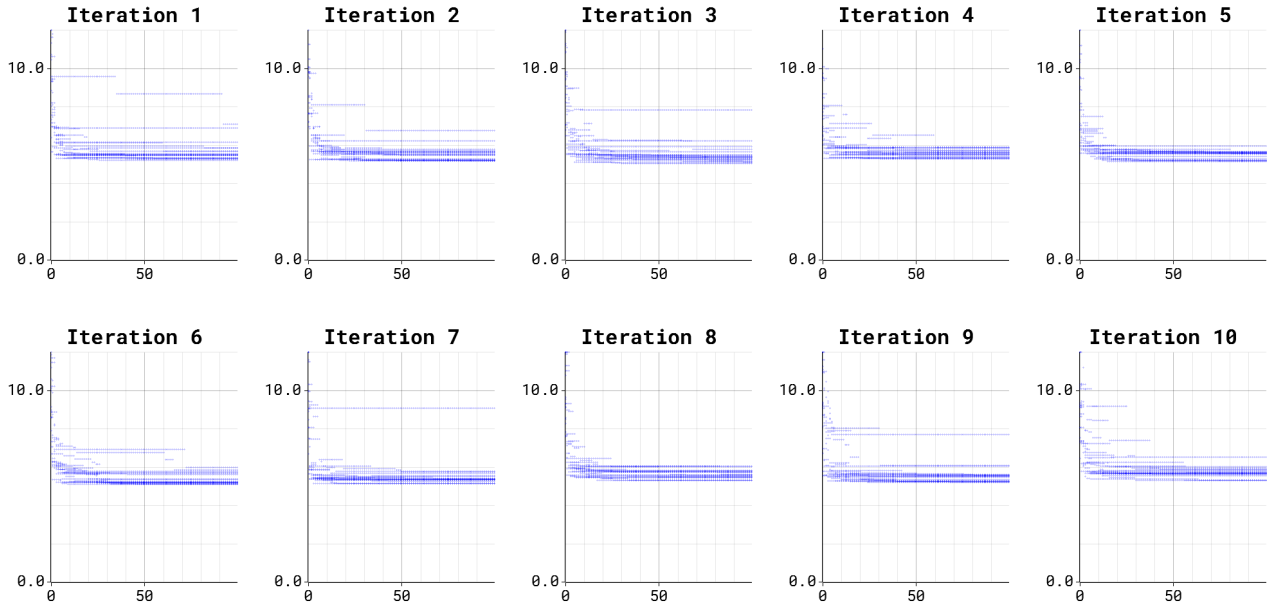


(a) The training process of each cross-validation iteration: x-axis is  $t$  value, y-axis is the evaluation result (MAE), and each blue dot is a particle at time  $t$  with its MAE.

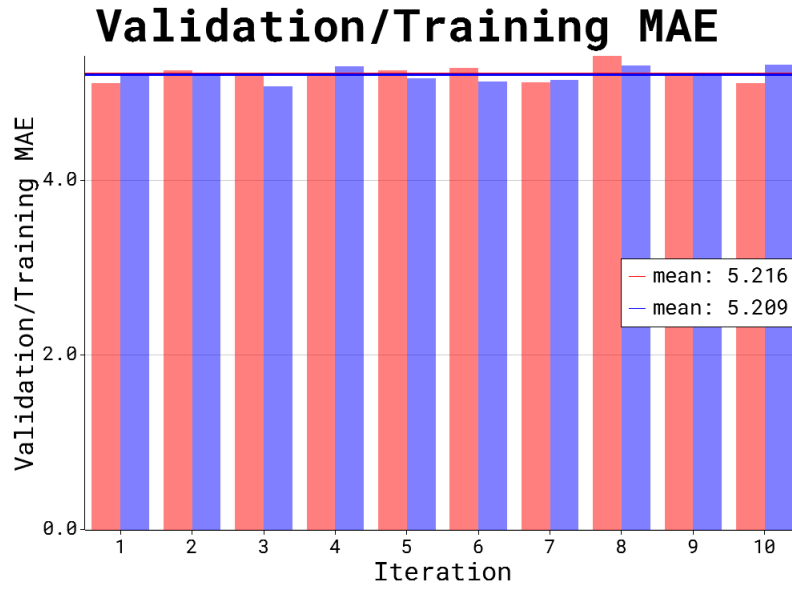


(b) The best particle from each cross-validation iteration MAE on training set (blue) and validation set (red).

Figure 5: Training result of air-8-1-1 of next 10 days dataset with 53.990 seconds used for training.



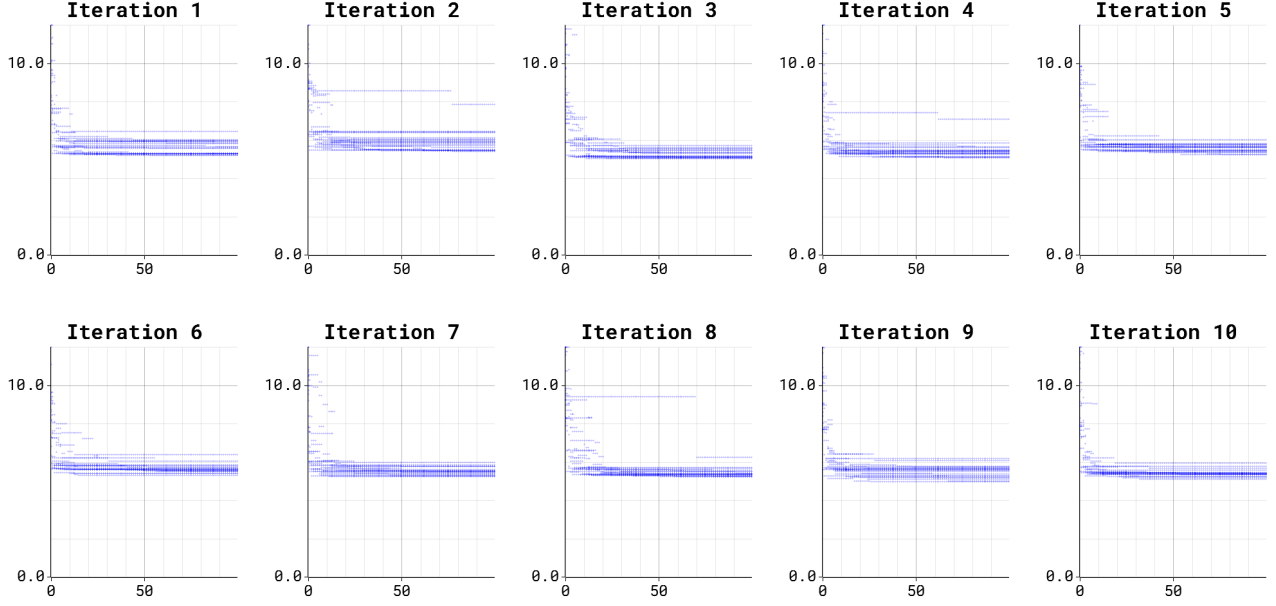
(a) The training process of each cross-validation iteration: x-axis is  $t$  value, y-axis is the evaluation result (MAE), and each blue dot is a particle at time  $t$  with its MAE.



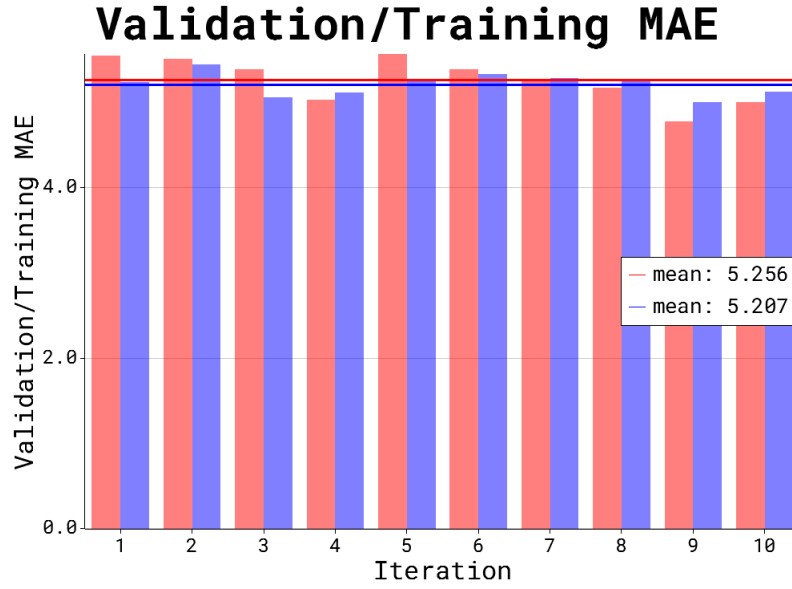
(b) The best particle from each cross-validation iteration MAE on training set (blue) and validation set (red).

Figure 6: Training result of air-8-8-4-1 of next 5 days dataset with 81.596 seconds used for training.





(a) The training process of each cross-validation iteration: x-axis is  $t$  value, y-axis is the evaluation result (MAE), and each blue dot is an particle at time  $t$  with its MAE.



(b) The best particle from each cross-validation iteration MAE on training set (blue) and validation set (red).

Figure 7: Training result of air-8-8-4-1 of next 10 days dataset with 78.985 seconds used for training.

## Source Code 1: main.rs

```

1  pub mod activator;
2  pub mod ga;
3  pub mod loss;
4  pub mod mlp;
5  pub mod models;
6  pub mod utils;
7
8  use std::error::Error;
9  fn main() -> Result<(), Box<dyn Error>> {
10     models::airquality::air_8_4_1();
11     models::airquality::air_8_1_1();
12     models::airquality::air_8_8_4_1();
13     Ok(())
14 }

```

## Source Code 2: mlp.rs

```

1  use crate::activator;
2
3  #[derive(Debug)]
4  pub struct Layer {
5      pub inputs: Vec<f64>,
6      pub outputs: Vec<f64>, // need to save this for backward pass
7      pub w: Vec<Vec<f64>>,
8      pub b: Vec<f64>,
9      pub grads: Vec<Vec<f64>>,
10     pub w_prev_changes: Vec<Vec<f64>>,
11     pub local_grads: Vec<f64>,
12     pub b_prev_changes: Vec<f64>,
13     pub act: activator::ActivationContainer,
14 }
15
16 impl Layer {
17     pub fn new(
18         input_features: u64,
19         output_features: u64,
20         bias: f64,
21         act: activator::ActivationContainer,
22     ) -> Layer {
23         // initialize weights matrix
24         let mut weights: Vec<Vec<f64>> = vec![];
25         let mut inputs: Vec<f64> = vec![];
26         let mut outputs: Vec<f64> = vec![];
27         let mut grads: Vec<Vec<f64>> = vec![];
28         let mut local_grads: Vec<f64> = vec![];
29         let mut w_prev_changes: Vec<Vec<f64>> = vec![];
30         let mut b_prev_changes: Vec<f64> = vec![];
31         let mut b: Vec<f64> = vec![];
32
33         for _ in 0..output_features {
34             outputs.push(0.0);
35             local_grads.push(0.0);
36             b_prev_changes.push(0.0);
37             b.push(bias);
38
39             let mut w: Vec<f64> = vec![];
40             let mut g: Vec<f64> = vec![];
41             for _ in 0..input_features {
42                 if (inputs.len() as u64) < input_features {
43                     inputs.push(0.0);
44                 }
45                 g.push(0.0);
46                 // random both positive and negative weight
47                 w.push(2f64 * rand::random::<f64>() - 1f64);
48             }
49             weights.push(w);
50             grads.push(g.clone());
51             w_prev_changes.push(g);
52         }
53         Layer {
54             inputs,

```

```

55         outputs,
56         w: weights,
57         b,
58         grads,
59         w_prev_changes,
60         local_grads,
61         b_prev_changes,
62         act,
63     }
64 }
65
66 pub fn forward(&mut self, inputs: &Vec<f64>) -> Vec<f64> {
67     if inputs.len() != self.inputs.len() {
68         panic!("forward: input size is wrong");
69     }
70
71     let result: Vec<f64> = self
72         .w
73         .iter()
74         .zip(self.b.iter())
75         .zip(self.outputs.iter_mut())
76         .map(|((w_j, b_j), o_j)| {
77             let sum = inputs
78                 .iter()
79                 .zip(w_j.iter())
80                 .fold(0.0, |s, (v, w_ji)| s + w_ji * v)
81                 + b_j;
82             *o_j = sum;
83             (self.act.func)(sum)
84         })
85         .collect();
86
87     self.inputs = inputs.clone();
88     result
89 }
90
91 pub fn update(&mut self, lr: f64, momentum: f64) {
92     for j in 0..self.w.len() {
93         let delta_b = lr * self.local_grads[j] + momentum * self.b_prev_changes[j];
94         self.b[j] -= delta_b; // update each neuron bias
95         self.b_prev_changes[j] = delta_b;
96         for i in 0..self.w[j].len() {
97             // update each weights
98             let delta_w = lr * self.grads[j][i] + momentum * self.w_prev_changes[j][i];
99             self.w[j][i] -= delta_w;
100             self.w_prev_changes[j][i] = delta_w;
101         }
102     }
103 }
104
105 pub fn zero_grad(&mut self) {
106     for j in 0..self.outputs.len() {
107         self.local_grads[j] = 0.0;
108         for i in 0..self.grads[j].len() {
109             self.grads[j][i] = 0.0;
110         }
111     }
112 }
113 }
114
115 #[derive(Debug)]
116 pub struct Net {
117     pub layers: Vec<Layer>,
118     pub parameters: u64,
119 }
120
121 impl Net {
122     pub fn from_layers(layers: Vec<Layer>) -> Net {
123         let mut parameters: u64 = 0;
124         for l in &layers {
125             parameters += (l.w.len() * l.w[0].len()) as u64;
126             parameters += l.b.len() as u64;
127         }
128
129         Net { layers, parameters }
130     }

```

```

131
132 pub fn new(architecture: Vec<u64>) -> Net {
133     let mut layers: Vec<Layer> = vec![];
134     for i in 1..architecture.len() {
135         layers.push(Layer::new(
136             architecture[i - 1],
137             architecture[i],
138             1f64,
139             activator::sigmoid(),
140         ))
141     }
142     Net::from_layers(layers)
143 }
144
145 /// Set this network parameters from flattened parameters.
146 pub fn set_params(&mut self, params: &Vec<f64>) {
147     if self.parameters != params.len() as u64 {
148         panic!["The neural network parameters size is not equal to individual size"];
149     }
150     let mut idx: usize = 0;
151
152     for l in self.layers.iter_mut() {
153         l.w.iter_mut().for_each(|w_j| {
154             w_j.iter_mut().for_each(|w_ji| {
155                 *w_ji = params[idx];
156                 idx += 1;
157             })
158         });
159
160         l.b.iter_mut().for_each(|b_i| {
161             *b_i = params[idx];
162             idx += 1;
163         });
164     }
165 }
166
167 pub fn zero_grad(&mut self) {
168     for l in 0..self.layers.len() {
169         self.layers[l].zero_grad();
170     }
171 }
172
173 pub fn forward(&mut self, input: &Vec<f64>) -> Vec<f64> {
174     let mut result = self.layers[0].forward(input);
175     for l in 1..self.layers.len() {
176         result = self.layers[l].forward(&result);
177     }
178     result
179 }
180
181 pub fn update(&mut self, lr: f64, momentum: f64) {
182     for l in 0..self.layers.len() {
183         self.layers[l].update(lr, momentum);
184     }
185 }
186 }
187
188 #[cfg(test)]
189 mod tests {
190     use super::*;
191
192     #[test]
193     fn test_linear_new() {
194         let linear = Layer::new(2, 3, 1.0, activator::linear());
195         assert_eq!(linear.outputs.len(), 3);
196         assert_eq!(linear.inputs.len(), 2);
197
198         assert_eq!(linear.w.len(), 3);
199         assert_eq!(linear.w[0].len(), 2);
200         assert_eq!(linear.b.len(), 3);
201
202         assert_eq!(linear.grads.len(), 3);
203         assert_eq!(linear.w_prev_changes.len(), 3);
204         assert_eq!(linear.grads[0].len(), 2);
205         assert_eq!(linear.w_prev_changes[0].len(), 2);
206         assert_eq!(linear.local_grads.len(), 3);

```

```

207     assert_eq!(linear.b_prev_changes.len(), 3);
208 }
209
210 #[test]
211 fn test_linear_forward1() {
212     let mut linear = Layer::new(2, 1, 1.0, activator::sigmoid());
213
214     for j in 0..linear.w.len() {
215         for i in 0..linear.w[j].len() {
216             linear.w[j][i] = 1.0;
217         }
218     }
219
220     assert_eq!(linear.forward(&vec![1.0, 1.0])[0], 0.9525741268224334);
221     assert_eq!(linear.outputs[0], 3.0);
222 }
223
224 #[test]
225 fn test_linear_forward2() {
226     let mut linear = Layer::new(2, 2, 1.0, activator::sigmoid());
227
228     for j in 0..linear.w.len() {
229         for i in 0..linear.w[j].len() {
230             linear.w[j][i] = (j as f64) + 1.0;
231         }
232     }
233     let result = linear.forward(&vec![0.0, 1.0]);
234     assert_eq!(linear.outputs[0], 2.0);
235     assert_eq!(linear.outputs[1], 3.0);
236     assert_eq!(result[0], 0.8807970779778823);
237     assert_eq!(result[1], 0.9525741268224334);
238 }
239
240 #[test]
241 fn test_set_params() {
242     let mut layers: Vec<Layer> = vec![];
243     layers.push(Layer::new(2, 2, 1.0, activator::relu()));
244     layers.push(Layer::new(2, 1, 1.0, activator::linear()));
245     let mut net = Net::from_layers(layers);
246     net.set_params(&vec![1.0, 1.0, 1.0, 1.0, 2.0, 2.0, 1.0, 1.0, 2.0]);
247
248     assert_eq!(net.layers[0].w[0], vec![1.0, 1.0]);
249     assert_eq!(net.layers[0].w[1], vec![1.0, 1.0]);
250     assert_eq!(net.layers[0].b, vec![2.0, 2.0]);
251 }
252 }
253

```

Source Code 3: activator.rs

```

1  #[derive(Debug)]
2  pub struct ActivationContainer {
3      pub func: fn(f64) -> f64,
4      pub der: fn(f64) -> f64,
5      pub name: String,
6  }
7
8  pub fn sigmoid() -> ActivationContainer {
9      fn func(input: f64) -> f64 {
10         1.0 / (1.0 + (-input).exp())
11     }
12     fn der(input: f64) -> f64 {
13         func(input) * (1.0 - func(input))
14     }
15     ActivationContainer {
16         func,
17         der,
18         name: "sigmoid".to_string(),
19     }
20 }
21
22 pub fn relu() -> ActivationContainer {
23     fn func(input: f64) -> f64 {
24         return f64::max(0.0, input);
25     }

```

```

26     fn der(input: f64) -> f64 {
27         if input > 0.0 {
28             return 1.0;
29         } else {
30             return 0.0;
31         }
32     }
33     ActivationContainer {
34         func,
35         der,
36         name: "relu".to_string(),
37     }
38 }
39
40 pub fn linear() -> ActivationContainer {
41     fn func(input: f64) -> f64 {
42         input
43     }
44     fn der(_input: f64) -> f64 {
45         1.0
46     }
47     ActivationContainer {
48         func,
49         der,
50         name: "linear".to_string(),
51     }
52 }
53
54 #[cfg(test)]
55 mod tests {
56     use super::*;
57
58     #[test]
59     fn test_sigmoid() {
60         let act = sigmoid();
61
62         assert_eq!((act.func)(1.0), 0.7310585786300048792512);
63         assert_eq!((act.func)(-1.0), 0.2689414213699951207488);
64         assert_eq!((act.func)(0.0), 0.5);
65         assert_eq!((act.der)(1.0), 0.1966119332414818525374);
66         assert_eq!((act.der)(-1.0), 0.1966119332414818525374);
67         assert_eq!((act.der)(0.0), 0.25);
68     }
69
70     #[test]
71     fn test_relu() {
72         let act = relu();
73
74         assert_eq!((act.func)(-1.0), 0.0);
75         assert_eq!((act.func)(20.0), 20.0);
76         assert_eq!((act.der)(-1.0), 0.0);
77         assert_eq!((act.der)(20.0), 1.0);
78     }
79 }
80

```

Source Code 4: loss.rs

```

1  use crate::mlp;
2
3  pub struct Loss {
4      outputs: Vec<f64>,
5      desired: Vec<f64>,
6      pub func: fn(f64, f64) -> f64,
7      pub der: fn(f64, f64) -> f64,
8  }
9
10 impl Loss {
11     /// Absolute Error
12     pub fn abs_err() -> Loss {
13         fn func(output: f64, desired: f64) -> f64 {
14             (output - desired).abs()
15         }
16         fn der(output: f64, desired: f64) -> f64 {
17             if output > desired {

```

```

18         1.0
19     } else {
20         -1.0
21     }
22 }
23
24 Loss {
25     outputs: vec![],
26     desired: vec![],
27     func,
28     der,
29 }
30 }
31
32 /// Squared Error
33 pub fn square_err() -> Loss {
34     fn func(output: f64, desired: f64) -> f64 {
35         0.5 * (output - desired).powi(2)
36     }
37     fn der(output: f64, desired: f64) -> f64 {
38         output - desired
39     }
40
41     Loss {
42         outputs: vec![],
43         desired: vec![],
44         func,
45         der,
46     }
47 }
48
49 /// Binary Cross Entropy
50 pub fn bce() -> Loss {
51     fn func(output: f64, desired: f64) -> f64 {
52         -desired * output.ln() + (1.0 - desired) * (1.0 - output).ln()
53     }
54     fn der(output: f64, desired: f64) -> f64 {
55         -(desired / output - (1.0 - desired) / (1.0 - output))
56     }
57
58     Loss {
59         outputs: vec![],
60         desired: vec![],
61         func,
62         der,
63     }
64 }
65
66 pub fn criterion(&mut self, outputs: &Vec<f64>, desired: &Vec<f64>) -> f64 {
67     if outputs.len() != desired.len() {
68         panic!("outputs size is not equal to desired size");
69     }
70     let loss = outputs
71         .iter()
72         .zip(desired.iter())
73         .fold(0.0, |ls, (o, d)| ls + (self.func)(*o, *d));
74     self.outputs = outputs.clone();
75     self.desired = desired.clone();
76     loss
77 }
78
79 pub fn backward(&self, layers: &mut Vec<mlp::Layer>) {
80     for l in (0..layers.len()).rev() {
81         // output layer
82         if l == layers.len() - 1 {
83             for j in 0..layers[l].outputs.len() {
84                 // compute grads
85                 let local_grad = (self.der)(self.outputs[j], self.desired[j])
86                     * (layers[l].act.der)(layers[l].outputs[j]);
87
88                 layers[l].local_grads[j] = local_grad;
89
90                 // set grads for each weight
91                 for k in 0..(layers[l - 1].outputs.len()) {
92                     layers[l].grads[j][k] =
93                         (layers[l - 1].act.func)(layers[l - 1].outputs[k]) * local_grad;

```

```

94         }
95     }
96     continue;
97 }
98 // hidden layer
99 for j in 0..layers[l].outputs.len() {
100     // calculate local_grad based on previous local_grad
101     let mut local_grad = 0f64;
102     for i in 0..layers[l + 1].w.len() {
103         for k in 0..layers[l + 1].w[i].len() {
104             local_grad += layers[l + 1].w[i][k] * layers[l + 1].local_grads[i];
105         }
106     }
107     local_grad = (layers[l].act.der)(layers[l].outputs[j]) * local_grad;
108     layers[l].local_grads[j] = local_grad;
109
110     // set grads for each weight
111     if l == 0 {
112         for k in 0..layers[l].inputs.len() {
113             layers[l].grads[j][k] = layers[l].inputs[k] * local_grad;
114         }
115     } else {
116         for k in 0..layers[l - 1].outputs.len() {
117             layers[l].grads[j][k] =
118                 (layers[l - 1].act.func)(layers[l - 1].outputs[k]) * local_grad;
119         }
120     }
121 }
122 }
123 }
124 }
125
126 #[cfg(test)]
127 mod tests {
128     use super::*;
129
130     #[test]
131     fn test_mse_func() {
132         assert_eq!((Loss::square_err().func)(2.0, 1.0), 0.5);
133         assert_eq!((Loss::square_err().func)(5.0, 0.0), 12.5);
134     }
135
136     #[test]
137     fn test_mse_der() {
138         assert_eq!((Loss::square_err().der)(2.0, 1.0), 1.0);
139         assert_eq!((Loss::square_err().der)(5.0, 0.0), 5.0);
140     }
141
142     #[test]
143     fn test_mse() {
144         let mut loss = Loss::square_err();
145
146         let l = loss.criterion(&vec![2.0, 1.0, 0.0], &vec![0.0, 1.0, 2.0]);
147         assert_eq!(l, 4.0);
148
149         loss.criterion(
150             &vec![34.0, 37.0, 44.0, 47.0, 48.0],
151             &vec![37.0, 40.0, 46.0, 44.0, 46.0],
152         );
153         assert_eq!(l, 4.0);
154     }
155
156     #[test]
157     fn test_bce_func() {
158         println!("{}", (Loss::bce().func)(0.9, 0.0));
159         println!("{}", (Loss::bce().func)(0.9, 1.0));
160     }
161 }
162

```

Source Code 5: models/airquality.rs

```

1 use std::time::Instant;
2
3 use crate::{

```



```

4     activator, loss,
5     mlp::{Layer, Net},
6     swarm::{self, gen_rho},
7     utils::{
8         data::{self, DataSet},
9         graph,
10    },
11 };
12
13 const IMGPATH: &str = "report/assignment_4/images";
14
15 pub fn air_8_4_1() {
16     fn model() -> Net {
17         let mut layers: Vec<Layer> = vec![];
18         layers.push(Layer::new(8, 4, 1.0, activator::relu()));
19         layers.push(Layer::new(4, 1, 1.0, activator::linear()));
20         Net::from_layers(layers)
21     }
22     air_particle_swarm(&model, "air-8-4-1");
23 }
24
25 pub fn air_8_1_1() {
26     fn model() -> Net {
27         let mut layers: Vec<Layer> = vec![];
28         layers.push(Layer::new(8, 1, 1.0, activator::relu()));
29         layers.push(Layer::new(1, 1, 1.0, activator::linear()));
30         Net::from_layers(layers)
31     }
32     air_particle_swarm(&model, "air-8-1-1");
33 }
34
35 pub fn air_8_8_4_1() {
36     fn model() -> Net {
37         let mut layers: Vec<Layer> = vec![];
38         layers.push(Layer::new(8, 8, 1.0, activator::relu()));
39         layers.push(Layer::new(8, 4, 1.0, activator::relu()));
40         layers.push(Layer::new(4, 1, 1.0, activator::linear()));
41         Net::from_layers(layers)
42     }
43     air_particle_swarm(&model, "air-8-8-4-1");
44 }
45
46 pub fn validation_test(net: &mut Net, validation_set: &DataSet, training_set: &DataSet) -> (f64, f64) {
47     let mut mae = 0.0;
48     for data in validation_set.get_datas() {
49         let result = net.forward(&data.inputs);
50         let abs_err = loss::Loss::abs_err().criterion(&result, &data.labels);
51         mae += abs_err;
52     }
53     mae = mae / validation_set.len() as f64;
54
55     let mut t_mae = 0.0;
56     for data in training_set.get_datas() {
57         let result = net.forward(&data.inputs);
58         let abs_err = loss::Loss::abs_err().criterion(&result, &data.labels);
59         t_mae += abs_err;
60     }
61     (mae, t_mae/training_set.len() as f64)
62 }
63
64 pub fn pso_fit(model: &dyn Fn() -> Net, dataset: &DataSet, folder: String) -> f32 {
65     let mut loss = loss::Loss::abs_err();
66     let max_epoch = 100;
67     let mut train_proc: Vec<Vec<(i32, f64)>> = (0..10).into_iter().map(|_| vec![]).collect();
68     let mut valid_mae: Vec<f64> = vec![];
69     let mut train_mae: Vec<f64> = vec![];
70
71     let start = Instant::now();
72     for (j, dt) in dataset.cross_valid_set(0.1).iter().enumerate() {
73         let (training_set, validation_set) = dt.minmax_norm(&dt.1);
74
75         let mut net = model();
76         let mut groups = swarm::init_particles_group(&net, 5, 4);
77
78         for i in 0..max_epoch {
79             for (k, g) in groups.iter_mut().enumerate() {

```

```

80         for (_, x) in g.particles.iter_mut().enumerate() {
81             net.set_params(&x.position);
82             let mut run_loss = 0.0;
83             for data in training_set.get_shuffled() {
84                 let result = net.forward(&data.inputs);
85                 run_loss += loss.criterion(&result, &data.labels);
86             }
87             let mae = run_loss / training_set.len() as f64; // Mean Absolute Error,  $F(x_i(t))$ 
88             if mae < x.f {
89                 x.f = mae;
90                 x.best_pos = x.position.clone();
91             }
92             if mae < g.lbest_f {
93                 g.lbest_f = mae; // set gbest
94                 g.lbest_pos = x.position.clone();
95             }
96             x.update_speed(&g.lbest_pos, gen_rho(1.0), gen_rho(1.5));
97             x.change_pos();
98             train_proc[j].push((i, x.f));
99         }
100         println!("{}", {} lbest : {:.5e}", k, i, g.lbest_f);
101     }
102 }
103
104 let best_group = &groups
105     .iter()
106     .reduce(|best, x| if best.lbest_f < x.lbest_f { best } else { x })
107     .unwrap();
108
109 let gbest = best_group
110     .particles
111     .iter()
112     .reduce(|best, ind| if best.f < ind.f { best } else { ind })
113     .unwrap();
114
115 net.set_params(&gbest.best_pos);
116 //io::save(&net.layers, "models/air/air-8-4-1.json".into()).unwrap();
117 let (v_mae, t_mae) = validation_test(&mut net, &validation_set, &training_set);
118 valid_mae.push(v_mae);
119 train_mae.push(t_mae);
120 }
121
122 let duration = start.elapsed();
123
124 graph::draw_ga_progress(
125     &train_proc,
126     format!("{}", {}/train_proc.png", IMGPATH, folder),
127     12.0,
128 )
129 .unwrap();
130
131 graph::hist::draw_2hist(
132     [&valid_mae, &train_mae],
133     "Validation/Training MAE",
134     ("Iteration", "Validation/Training MAE"),
135     format!("{}", {}/mae.png", IMGPATH, folder),
136 ).unwrap();
137
138 duration.as_secs_f32()
139 }
140
141 pub fn air_particle_swarm(model: &dyn Fn() -> Net, folder: &str) {
142     let (dataset_five, dataset_ten) =
143         data::airquality_dataset().expect("Something wrong with airquality_dataset");
144
145     let t1 = pso_fit(model, &dataset_five, format!("5days/{}", folder));
146     let t2 = pso_fit(model, &dataset_ten, format!("10days/{}", folder));
147
148     println!("t1: {:.3} sec, t2: {:.3} sec", t1, t2);
149 }
150
151

```

Source Code 6: swarm/mod.rs

```

1 use rand::{distributions::Uniform, prelude::Distribution};
2
3 use crate::mlp::Net;
4
5 #[derive(Debug, Clone)]
6 pub struct Individual {
7     pub best_pos: Vec<f64>,
8     pub position: Vec<f64>,
9     pub f: f64, // evaluation of this individual
10    pub speed: Vec<f64>,
11 }
12
13 impl Individual {
14     pub fn new(position: Vec<f64>) -> Individual {
15         let mut rand = rand::thread_rng();
16         let dist = Uniform::from(-1.0..=1.0);
17         let speed: Vec<f64> = position.iter().map(|_i| dist.sample(&mut rand)).collect();
18         Individual {
19             best_pos: position.clone(),
20             position,
21             f: f64::MAX,
22             speed,
23         }
24     }
25
26     /// Individual best speed updater
27     pub fn ind_update_speed(&mut self, rho: f64) {
28         self.speed
29             .iter_mut()
30             .zip(self.best_pos.iter().zip(self.position.iter()))
31             .for_each(|(v, (x_b, x))| {
32                 *v = *v + rho * (*x_b - *x);
33             });
34     }
35
36     /// Speed upator with social component included
37     pub fn update_speed(&mut self, other_best: &Vec<f64>, rho1: f64, rho2: f64) {
38         let w = 1.0;
39         self.speed
40             .iter_mut()
41             .zip(
42                 self.position
43                     .iter()
44                     .zip(self.best_pos.iter().zip(other_best.iter())),
45             )
46             .for_each(|(v, (x, (x_b, x_gb)))| {
47                 *v = w * *v + rho1 * (*x_b - *x) + rho2 * (*x_gb - *x);
48             });
49     }
50
51     pub fn change_pos(&mut self) {
52         self.position
53             .iter_mut()
54             .zip(self.speed.iter())
55             .for_each(|(x, v)| {
56                 *x = *x + v;
57             });
58     }
59 }
60
61 pub fn gen_rho(c: f64) -> f64 {
62     let mut rand = rand::thread_rng();
63     let dist = Uniform::from(0.0..=1.0);
64     dist.sample(&mut rand) * c
65 }
66
67 /// Create initial particles of MLP from layers
68 ///
69 /// return: particles
70 pub fn init_particles(net: &Net, amount: u32) -> Vec<Individual> {
71     let mut individuals: Vec<Individual> = vec![];
72     for _ in 0..amount {
73         let mut position: Vec<f64> = Vec::with_capacity(net.parameters as usize);
74         for l in net.layers.iter() {
75             for output in l.w.iter() {
76                 for _ in output.iter() {

```

```

77         // new random weight in range [-1, 1]
78         position.push(2f64 * rand::random::<f64>() - 1f64);
79     }
80 }
81     for bias in l.b.iter() {
82         position.push(*bias);
83     }
84 }
85     inidividuals.push(Individual::new(position));
86 }
87     inidividuals
88 }
89
90 pub struct IndividualGroup {
91     pub particles: Vec<Individual>,
92     pub lbest_f: f64,
93     pub lbest_pos: Vec<f64>,
94 }
95
96 impl IndividualGroup {
97     pub fn add(&mut self, individual: Individual) {
98         self.particles.push(individual);
99     }
100 }
101
102 pub fn init_particles_group(net: &Net, group: usize, group_size: u32) -> Vec<IndividualGroup> {
103     (0..group)
104         .into_iter()
105         .map(|_| {
106             let particles = init_particles(&net, group_size + 1);
107             IndividualGroup {
108                 particles: particles[1..].into(),
109                 lbest_f: f64::MAX,
110                 lbest_pos: particles[0].position.clone(),
111             }
112         })
113         .collect()
114 }
115
116 #[cfg(test)]
117 mod tests {
118     use crate::{activator, mlp::Layer};
119
120     use super::*;
121
122     #[test]
123     fn test_update_speed() {
124         fn f(pos: &Vec<f64>) -> f64 {
125             pos[0].powi(2) + 2.0 * pos[1]
126         }
127
128         let mut p1 = Individual::new(vec![1.0, 1.0]);
129         p1.f = 4.0;
130         p1.speed = vec![0.5, 0.5];
131
132         let gbest = vec![0.5, 1.0];
133
134         // training
135         let eval_result = f(&p1.position);
136         if eval_result < p1.f {
137             p1.f = eval_result;
138             p1.best_pos = p1.position.clone();
139         }
140
141         p1.update_speed(&gbest, 1.0, 1.0);
142         p1.change_pos();
143
144         assert_eq!(p1.speed, vec![0.0, 0.5]);
145         assert_eq!(p1.position, vec![1.0, 1.5]);
146     }
147
148     #[test]
149     fn test_split() {
150         let mut layers: Vec<Layer> = vec![];
151         layers.push(Layer::new(4, 2, 1.0, activator::sigmoid()));
152         layers.push(Layer::new(2, 1, 1.0, activator::sigmoid()));

```

```

153         let net = Net::from_layers(layers);
154
155         let groups = init_particles_group(&net, 3, 3);
156         assert_eq!(groups.len(), 3);
157         assert_eq!(groups[2].particles.len(), 3);
158
159         let groups = init_particles_group(&net, 2, 5);
160         assert_eq!(groups.len(), 2);
161         assert_eq!(groups[0].particles.len(), 5);
162     }
163 }
164

```

## Source Code 7: utils/data.rs

```

1  use super::io::read_lines;
2  use chrono::{DateTime, Duration, TimeZone, Utc};
3  use rand::prelude::SliceRandom;
4  use serde::Deserialize;
5  use std::error::Error;
6
7  pub fn max(vec: &Vec<f64>) -> f64 {
8      vec.iter().fold(f64::NAN, |max, &v| v.max(max))
9  }
10
11  pub fn min(vec: &Vec<f64>) -> f64 {
12      vec.iter().fold(f64::NAN, |min, &v| v.min(min))
13  }
14
15  pub fn std(vec: &Vec<f64>, mean: f64) -> f64 {
16      let n = vec.len() as f64;
17      vec.iter()
18          .fold(0.0f64, |sum, &val| sum + (val - mean).powi(2) / n)
19          .sqrt()
20  }
21
22  pub fn mean(vec: &Vec<f64>) -> f64 {
23      let n = vec.len() as f64;
24      vec.iter().fold(0.0f64, |mean, &val| mean + val / n)
25  }
26
27  pub fn standardization(data: &Vec<f64>, mean: f64, std: f64) -> Vec<f64> {
28      data.iter().map(|x| (x - mean) / std).collect()
29  }
30
31  pub fn minmax_norm(data: &Vec<f64>, min: f64, max: f64) -> Vec<f64> {
32      data.iter().map(|x| (x - min) / (max - min)).collect()
33  }
34
35  #[derive(Debug, Clone)]
36  pub struct Data {
37      pub inputs: Vec<f64>,
38      pub labels: Vec<f64>,
39  }
40  #[derive(Clone)]
41  pub struct DataSet {
42      datas: Vec<Data>,
43  }
44
45  impl DataSet {
46      pub fn new(datas: Vec<Data>) -> DataSet {
47          DataSet { datas }
48      }
49
50      pub fn cross_valid_set(&self, percent: f64) -> Vec<(DataSet, DataSet)> {
51          if percent < 0.0 && percent > 1.0 {
52              panic!("argument percent must be in range [0, 1]")
53          }
54          let k = (percent * (self.datas.len() as f64)).ceil() as usize; // fold size
55          let n = (self.datas.len() as f64 / k as f64).ceil() as usize; // number of folds
56          let datas = self.get_shuffled().clone(); // shuffled data before slicing it
57          let mut set: Vec<(DataSet, DataSet)> = vec![];
58
59          let mut curr: usize = 0;
60          for _ in 0..n {

```

```

61         let r_pt: usize = if curr + k > datas.len() {
62             datas.len()
63         } else {
64             curr + k
65         };
66
67         let validation_set: Vec<Data> = datas[curr..r_pt].to_vec();
68         let training_set: Vec<Data> = if curr > 0 {
69             let mut temp = datas[0..curr].to_vec();
70             temp.append(&mut datas[r_pt..datas.len()].to_vec());
71             temp
72         } else {
73             datas[r_pt..datas.len()].to_vec()
74         };
75
76         set.push((DataSet::new(training_set), DataSet::new(validation_set)));
77         curr += k
78     }
79     set
80 }
81
82 pub fn data_points(&self) -> Vec<f64> {
83     let mut data_points: Vec<f64> = vec![];
84     for mut dt in self.datas.clone() {
85         data_points.append(&mut dt.inputs);
86         data_points.append(&mut dt.labels);
87     }
88     data_points
89 }
90
91 pub fn len(&self) -> usize {
92     self.datas.len()
93 }
94
95 pub fn standardization(&self, valid_set: &DataSet) -> (DataSet, DataSet) {
96     let size = self.datas[0].inputs.len();
97     let features: Vec<(Vec<f64>, Vec<f64>)> = (0..size)
98         .into_iter()
99         .map(|i| {
100             let feature = self.get_feature(i);
101             let v_feature = valid_set.get_feature(i);
102             let mean = mean(&feature);
103             let std = std(&feature, mean);
104             (
105                 standardization(&feature, mean, std),
106                 standardization(&v_feature, mean, std),
107             )
108         })
109         .collect();
110
111     let datas: Vec<Data> = self
112         .datas
113         .iter()
114         .enumerate()
115         .map(|(i, dt)| Data {
116             labels: dt.labels.clone(),
117             inputs: features.iter().map(|x| x.0[i]).collect(),
118         })
119         .collect();
120
121     let v_datas: Vec<Data> = valid_set
122         .datas
123         .iter()
124         .enumerate()
125         .map(|(i, dt)| Data {
126             labels: dt.labels.clone(),
127             inputs: features.iter().map(|x| x.1[i]).collect(),
128         })
129         .collect();
130
131     (DataSet::new(datas), DataSet::new(v_datas))
132 }
133
134 pub fn minmax_norm(&self, valid_set: &DataSet) -> (DataSet, DataSet) {
135     let size = self.datas[0].inputs.len();
136     let features: Vec<(Vec<f64>, Vec<f64>)> = (0..size)

```

```

137     .into_iter()
138     .map(|i| {
139         let feature = self.get_feature(i);
140         let v_feature = valid_set.get_feature(i);
141         let min = min(&feature);
142         let max = max(&feature);
143         (
144             minmax_norm(&feature, min, max),
145             minmax_norm(&v_feature, min, max),
146         )
147     })
148     .collect();
149
150     let datas: Vec<Data> = self
151         .datas
152         .iter()
153         .enumerate()
154         .map(|(i, dt)| Data {
155             labels: dt.labels.clone(),
156             inputs: features.iter().map(|x| x.0[i]).collect(),
157         })
158         .collect();
159
160     let v_datas: Vec<Data> = valid_set
161         .datas
162         .iter()
163         .enumerate()
164         .map(|(i, dt)| Data {
165             labels: dt.labels.clone(),
166             inputs: features.iter().map(|x| x.1[i]).collect(),
167         })
168         .collect();
169
170     (DataSet::new(datas), DataSet::new(v_datas))
171 }
172
173 pub fn get_datas(&self) -> Vec<Data> {
174     self.datas.clone()
175 }
176
177 pub fn get_feature(&self, i: usize) -> Vec<f64> {
178     if i >= self.datas[0].inputs.len() {
179         panic!("i should not exceed inputs feature size");
180     }
181
182     self.datas.iter().map(|data| data.inputs[i]).collect()
183 }
184
185 pub fn get_label(&self, i: usize) -> Vec<f64> {
186     if i >= self.datas[0].labels.len() {
187         panic!("i should not exceed inputs feature size");
188     }
189
190     self.datas.iter().map(|data| data.labels[i]).collect()
191 }
192
193 pub fn get_shuffled(&self) -> Vec<Data> {
194     let mut shuffled_datas = self.datas.clone();
195     shuffled_datas.shuffle(&mut rand::thread_rng());
196     shuffled_datas
197 }
198 }
199
200 pub fn confusion_count(
201     matrix: &mut [[i32; 2]; 2],
202     result: &Vec<f64>,
203     label: &Vec<f64>,
204     threshold: f64,
205 ) {
206     if result[0] > threshold {
207         // true positive
208         if label[0] == 1.0 {
209             matrix[0][0] += 1
210         } else {
211             // false negative
212             matrix[1][0] += 1

```

```

213     }
214 } else if result[0] <= threshold {
215     // true negative
216     if label[0] == 0.0 {
217         matrix[1][1] += 1
218     }
219     // false positive
220     else {
221         matrix[0][1] += 1
222     }
223 }
224 }
225
226 pub fn un_standardization(value: f64, mean: f64, std: f64) -> f64 {
227     value * std + mean
228 }
229
230 pub fn xor_dataset() -> DataSet {
231     let inputs = vec![[0.0, 0.0], [0.0, 1.0], [1.0, 0.0], [1.0, 1.0]];
232     let labels = vec![[0.0], [1.0], [1.0], [0.0]];
233     let mut datas: Vec<Data> = vec![];
234     for i in 0..4 {
235         datas.push(Data {
236             inputs: inputs[i].to_vec(),
237             labels: labels[i].to_vec(),
238         });
239     }
240
241     DataSet::new(datas)
242 }
243
244 pub fn flood_dataset() -> Result<DataSet, Box<dyn Error>> {
245     #[derive(Deserialize)]
246     struct Record {
247         s1_t3: f64,
248         s1_t2: f64,
249         s1_t1: f64,
250         s1_t0: f64,
251         s2_t3: f64,
252         s2_t2: f64,
253         s2_t1: f64,
254         s2_t0: f64,
255         t7: f64,
256     }
257
258     let mut datas: Vec<Data> = vec![];
259     let mut reader = csv::Reader::from_path("data/flood_dataset.csv")?;
260     for record in reader.deserialize() {
261         let record: Record = record?;
262         let mut inputs: Vec<f64> = vec![];
263         // station 1
264         inputs.push(record.s1_t3);
265         inputs.push(record.s1_t2);
266         inputs.push(record.s1_t1);
267         inputs.push(record.s1_t0);
268         // station 2
269         inputs.push(record.s2_t3);
270         inputs.push(record.s2_t2);
271         inputs.push(record.s2_t1);
272         inputs.push(record.s2_t0);
273
274         let labels: Vec<f64> = vec![f64::from(record.t7)];
275         datas.push(Data { inputs, labels });
276     }
277     Ok(DataSet::new(datas))
278 }
279
280 pub fn cross_dataset() -> Result<DataSet, Box<dyn Error>> {
281     let mut datas: Vec<Data> = vec![];
282     let mut lines = read_lines("data/cross.pat")?;
283     while let (Some(_), Some(Ok(l1)), Some(Ok(l2))) = (lines.next(), lines.next(), lines.next()) {
284         let mut inputs: Vec<f64> = vec![];
285         let mut labels: Vec<f64> = vec![];
286         for w in l1.split(" ") {
287             let v: f64 = w.parse().unwrap();
288             inputs.push(v);

```



```

289     }
290     for w in l2.split(" ") {
291         let v: f64 = w.parse().unwrap();
292         // class 1 0 -> 1
293         // class 0 1 -> 0
294         labels.push(v);
295         break;
296     }
297     datas.push(Data { inputs, labels });
298 }
299 Ok(DataSet::new(datas))
300 }
301
302 pub fn wdbc_dataset() -> Result<DataSet, Box<dyn Error>> {
303     let mut datas: Vec<Data> = vec![];
304     let mut lines = read_lines("data/wdbc.txt"?);
305     while let Some(Ok(line)) = lines.next() {
306         let mut inputs: Vec<f64> = vec![];
307         let mut labels: Vec<f64> = vec![]; // M (malignant) = 1.0, B (benign) = 0.0
308         let arr: Vec<&str> = line.split(",").collect();
309         if arr[1] == "M" {
310             labels.push(1.0);
311         } else if arr[1] == "B" {
312             labels.push(0.0);
313         }
314         for w in &arr[2..] {
315             let v: f64 = w.parse()?;
316             inputs.push(v);
317         }
318         datas.push(Data { inputs, labels });
319     }
320     Ok(DataSet::new(datas))
321 }
322
323 /// Return `(desired = next five days, desired = next ten days)`
324 pub fn airquality_dataset() -> Result<(DataSet, DataSet), Box<dyn Error>> {
325     // na is not used
326     #[derive(Deserialize, Debug)]
327     struct Record {
328         #[serde(rename = "Date")]
329         date: String,
330         #[serde(rename = "Time")]
331         time: String,
332         #[serde(rename = "PT08.S1(CO)")]
333         pt_s1: i32,
334         #[serde(rename = "C6H6(GT)")]
335         benzene: f64,
336         #[serde(rename = "PT08.S2(NMHC)")]
337         pt_s2: i32,
338         #[serde(rename = "PT08.S3(NOx)")]
339         pt_s3: i32,
340         #[serde(rename = "PT08.S4(NO2)")]
341         pt_s4: i32,
342         #[serde(rename = "PT08.S5(O3)")]
343         pt_s5: i32,
344         #[serde(rename = "T")]
345         temp: f64,
346         #[serde(rename = "RH")]
347         rh: f64,
348         #[serde(rename = "AH")]
349         ah: f64,
350     }
351     #[derive(Debug)]
352     struct RecData {
353         datetime: DateTime<Utc>,
354         input: Vec<f64>,
355         output: Vec<f64>,
356         pub okay: bool,
357     }
358     /// add to input if input is true else add to output
359     fn rec_add(recdata: &mut RecData, v: f64, input: bool) {
360         if v == -200.0 {
361             recdata.okay = false;
362         }
363         if input {
364             recdata.input.push(v);

```

```

365     } else {
366         recdata.output.push(v)
367     };
368 }
369
370 impl RecData {
371     pub fn new(record: &Record) -> RecData {
372         let datetime_str = format!("{}", record.date, record.time);
373         let datetime = Utc
374             .datetime_from_str(&datetime_str, "%-m/%-d/%Y %-H:%M:%S")
375             .unwrap();
376
377         let mut recdata = RecData {
378             datetime,
379             input: vec![],
380             output: vec![],
381             okay: true,
382         };
383         rec_add(&mut recdata, record.pt_s1 as f64, true);
384         rec_add(&mut recdata, record.pt_s2 as f64, true);
385         rec_add(&mut recdata, record.pt_s3 as f64, true);
386         rec_add(&mut recdata, record.pt_s4 as f64, true);
387         rec_add(&mut recdata, record.pt_s5 as f64, true);
388         rec_add(&mut recdata, record.temp, true);
389         rec_add(&mut recdata, record.rh, true);
390         rec_add(&mut recdata, record.ah, true);
391         rec_add(&mut recdata, record.benzene, false);
392         recdata
393     }
394 }
395
396 let mut reader = csv::Reader::from_path("data/AirQualityUCI.csv")?;
397 let mut rec_datas: Vec<RecData> = vec![];
398 for record in reader.deserialize() {
399     let rec_data = RecData::new(&record.unwrap());
400     if rec_data.okay {
401         rec_datas.push(rec_data)
402     };
403 }
404
405 // Duration::days(5)
406 let mut datas_five: Vec<Data> = vec![];
407 let mut datas_ten: Vec<Data> = vec![];
408
409 for (i, x) in rec_datas.iter().enumerate() {
410     let next_five_days = x.datetime + Duration::days(5);
411     let next_ten_days = x.datetime + Duration::days(10);
412
413     let mut labels_five: Vec<f64> = vec![];
414     let mut labels_ten: Vec<f64> = vec![];
415
416     for y in &rec_datas[i..] {
417         if y.datetime == next_five_days {
418             labels_five = y.output.clone();
419         }
420         if y.datetime == next_ten_days {
421             labels_ten = y.output.clone();
422             break;
423         }
424     }
425
426     if labels_five.len() == 0 && labels_ten.len() == 0 {
427         break;
428     }
429     if labels_five.len() != 0 {
430         datas_five.push(Data {
431             inputs: x.input.clone(),
432             labels: labels_five,
433         });
434     }
435     if labels_ten.len() != 0 {
436         datas_ten.push(Data {
437             inputs: x.input.clone(),
438             labels: labels_ten,
439         });
440     }

```

```

441     }
442     Ok((DataSet::new(datas_five), DataSet::new(datas_ten)))
443 }
444
445 #[cfg(test)]
446 mod tests {
447     use super::*;
448
449     #[test]
450     fn test_airquality() -> Result<(), Box<dyn Error>> {
451         let (dt5, _) = airquality_dataset().unwrap();
452
453         let dt = &dt5.cross_valid_set(0.1)[0];
454         let (train, _) = dt.0.minmax_norm(&dt.1);
455
456         for dt in train.get_datas().iter() {
457             for v in dt.inputs.iter() {
458                 print!("{:.3e} ", v);
459             }
460             print!("{:.3e}\n", dt.labels[0]);
461         }
462         println!("{}", train.len());
463         Ok(())
464     }
465
466     #[test]
467     fn test_minmax_norm() {
468         let datas: Vec<Data> = (0..=10)
469             .into_iter()
470             .map(|i| Data {
471                 labels: vec![0.0],
472                 inputs: vec![i as f64 * 10.0],
473             })
474             .collect();
475         let v_datas: Vec<Data> = (0..=10)
476             .into_iter()
477             .map(|i| Data {
478                 labels: vec![0.0],
479                 inputs: vec![i as f64 * 5.0],
480             })
481             .collect();
482
483         let (t, v) = DataSet::new(datas).minmax_norm(&DataSet::new(v_datas));
484
485         let expected = vec![0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0];
486         for (i, x) in t.get_feature(0).iter().enumerate() {
487             assert_eq!(*x, expected[i]);
488         }
489         let v_expected = vec![
490             0.0, 0.05, 0.10, 0.15, 0.20, 0.25, 0.30, 0.35, 0.40, 0.45, 0.50,
491         ];
492         for (i, x) in v.get_feature(0).iter().enumerate() {
493             assert_eq!(*x, v_expected[i])
494         }
495     }
496 }
497

```

Source Code 8: utils/graph/mod.rs

```

1     use plotters::coord::Shift;
2     use plotters::prelude::*;
3     use std::error::Error;
4
5     pub mod hist;
6
7     const FONT: &str = "Roboto Mono";
8     const CAPTION: i32 = 70;
9     const SERIE_LABEL: i32 = 32;
10    const AXIS_LABEL: i32 = 40;
11
12    pub struct LossGraph {
13        loss: Vec<Vec<f64>>,
14        valid_loss: Vec<Vec<f64>>,
15    }

```

```

16
17 impl LossGraph {
18     pub fn new() -> LossGraph {
19         let loss: Vec<Vec<f64>> = vec![];
20         let valid_loss: Vec<Vec<f64>> = vec![];
21         LossGraph { loss, valid_loss }
22     }
23
24     pub fn add_loss(&mut self, training: Vec<f64>, validation: Vec<f64>) {
25         self.loss.push(training);
26         self.valid_loss.push(validation);
27     }
28     /// Draw training loss and validation loss at each epoch (x_vec)
29     pub fn draw_loss(
30         &self,
31         idx: u32,
32         root: &DrawingArea<BitMapBackend, Shift>,
33         loss_vec: &Vec<f64>,
34         valid_loss_vec: &Vec<f64>,
35         max_loss: f64,
36     ) -> Result<(), Box<dyn Error>> {
37         let min_loss1 = loss_vec.iter().fold(f64::NAN, |min, &val| val.min(min));
38         let min_loss2 = valid_loss_vec
39             .iter()
40             .fold(f64::NAN, |min, &val| val.min(min));
41         let min_loss = if min_loss1.min(min_loss2) > 0.0 {
42             0.0
43         } else {
44             min_loss1.min(min_loss2)
45         };
46
47         let mut chart = ChartBuilder::on(&root)
48             .caption(
49                 format!("Loss {} ", idx),
50                 ("Hack", 44, FontStyle::Bold).into_font(),
51             )
52             .margin(20)
53             .x_label_area_size(50)
54             .y_label_area_size(60)
55             .build_cartesian_2d(0..loss_vec.len(), min_loss..max_loss)?;
56
57         chart
58             .configure_mesh()
59             .y_desc("Loss")
60             .x_desc("Epochs")
61             .axis_desc_style(("Hack", 20))
62             .draw()?;
63
64         chart.draw_series(LineSeries::new(
65             loss_vec.iter().enumerate().map(|(i, x)| (i + 1, *x)),
66             &RED,
67         ))?;
68
69         chart.draw_series(LineSeries::new(
70             valid_loss_vec.iter().enumerate().map(|(i, x)| (i + 1, *x)),
71             &BLUE,
72         ))?;
73
74         root.present()?;
75         Ok(())
76     }
77
78     pub fn max_loss(&self) -> f64 {
79         f64::max(
80             self.loss.iter().fold(f64::NAN, |max, vec| {
81                 let max_loss = vec.iter().fold(f64::NAN, |max, &val| val.max(max));
82                 f64::max(max_loss, max)
83             }),
84             self.valid_loss.iter().fold(f64::NAN, |max, vec| {
85                 let max_loss = vec.iter().fold(f64::NAN, |max, &val| val.max(max));
86                 f64::max(max_loss, max)
87             }),
88         )
89     }
90
91     pub fn draw(&self, path: String) -> Result<(), Box<dyn Error>> {

```

```

92     let root = BitMapBackend::new(&path, (2000, 1000)).into_drawing_area();
93     root.fill(&WHITE)?;
94     // hardcoded for 10 iterations
95     let drawing_areas = root.split_evenly((2, 5));
96
97     let mut loss_iter = self.loss.iter();
98     let mut valid_loss_iter = self.valid_loss.iter();
99     let max_loss = self.max_loss();
100    for (drawing_area, idx) in drawing_areas.iter().zip(1..) {
101        if let (Some(loss_vec), Some(valid_loss_vec)) =
102            (loss_iter.next(), valid_loss_iter.next())
103        {
104            self.draw_loss(idx, drawing_area, loss_vec, valid_loss_vec, max_loss)?;
105        }
106    }
107    Ok(())
108 }
109 }
110
111 /// Draw confusion matrix
112 pub fn draw_confusion(matrix_vec: Vec<[[i32; 2]; 2]>, path: String) -> Result<(), Box<dyn Error>> {
113     let root = BitMapBackend::new(&path, (2000, 1100)).into_drawing_area();
114     root.fill(&WHITE)?;
115
116     let (top, down) = root.split_vertically(1000);
117
118     let mut chart = ChartBuilder::on(&down)
119         .margin(20)
120         .margin_left(40)
121         .margin_right(40)
122         .x_label_area_size(40)
123         .build_cartesian_2d(0i32..50i32, 0i32..1i32)?;
124     chart
125         .configure_mesh()
126         .disable_y_axis()
127         .disable_y_mesh()
128         .x_labels(3)
129         .label_style((FONT, 40))
130         .draw()?;
131
132     chart.draw_series((0..50).map(|x| {
133         Rectangle::new(
134             [(x, 0), (x + 1, 1)],
135             HSLColor(
136                 240.0 / 360.0 - 240.0 / 360.0 * (x as f64 / 50.0),
137                 0.7,
138                 0.1 + 0.4 * x as f64 / 50.0,
139             )
140             .filled(),
141         )
142     }))?;
143     // hardcoded for 10 iterations
144     let drawing_areas = top.split_evenly((2, 5));
145     let mut matrix_iter = matrix_vec.iter();
146     for (drawing_area, idx) in drawing_areas.iter().zip(1..) {
147         if let Some(matrix) = matrix_iter.next() {
148             let mut chart = ChartBuilder::on(&drawing_area)
149                 .caption(
150                     format!("Iteration {}", idx),
151                     (FONT, 40, FontStyle::Bold).into_font(),
152                 )
153                 .margin(20)
154                 .build_cartesian_2d(0i32..2i32, 2i32..0i32)?
155                 .set_secondary_coord(0f64..2f64, 2f64..0f64);
156
157             chart
158                 .configure_mesh()
159                 .disable_axes()
160                 .max_light_lines(4)
161                 .disable_x_mesh()
162                 .disable_y_mesh()
163                 .label_style(("Hack", 20))
164                 .draw()?;
165
166             chart.draw_series(
167                 matrix

```

```

168         .iter()
169         .zip(0..)
170         .map(|(l, y)| l.iter().zip(0..).map(move |(v, x)| (x, y, v)))
171         .flatten()
172         .map(|(x, y, v)| {
173             Rectangle::new(
174                 [(x, y), (x + 1, y + 1)],
175                 HSLColor(
176                     240.0 / 360.0 - 240.0 / 360.0 * (*v as f64 / 50.0),
177                     0.7,
178                     0.1 + 0.4 * *v as f64 / 50.0,
179                 )
180                 .filled(),
181             )
182         }),
183     )?;
184
185     chart.draw_secondary_series(
186         matrix
187         .iter()
188         .zip(0..)
189         .map(|(l, y)| l.iter().zip(0..).map(move |(v, x)| (x, y, v)))
190         .flatten()
191         .map(|(x, y, v)| {
192             let text: String = if x == 0 && y == 0 {
193                 format!["TP:{}", v]
194             } else if x == 1 && y == 0 {
195                 format!["FP:{}", v]
196             } else if x == 0 && y == 1 {
197                 format!["FN:{}", v]
198             } else {
199                 format!["TN:{}", v]
200             };
201
202             Text::new(
203                 text,
204                 ((2.0 * x as f64 + 0.7) / 2.0, (2.0 * y as f64 + 1.0) / 2.0),
205                 FONT.into_font().resize(30.0).color(&WHITE),
206             )
207         }),
208     )?;
209 }
210
211 root.present()?;
212 Ok(())
213 }
214
215 /// Receive each cross-validation vector of each individual fitness value.
216 pub fn draw_ga_progress(
217     cv_fitness: &Vec<Vec<(i32, f64)>>,
218     path: String,
219     max_y: f64,
220 ) -> Result<(), Box<dyn Error>> {
221     let root = BitMapBackend::new(&path, (2000, 1000)).into_drawing_area();
222     root.fill(&WHITE)?;
223
224     let max_x = cv_fitness[0]
225         .iter()
226         .reduce(|m, x| if m.0 > x.0 { m } else { x })
227         .unwrap()
228         .0;
229
230     // This is mostly hardcoded
231     let drawing_areas = root.split_evenly((2, 5));
232     for ((drawing_area, idx), fitness) in drawing_areas.iter().zip(1..).zip(cv_fitness.iter()) {
233         let mut chart = ChartBuilder::on(&drawing_area)
234             .caption(
235                 format!("Iteration {}", idx),
236                 (FONT, 40, FontStyle::Bold).into_font(),
237             )
238             .margin(40)
239             .x_label_area_size(20)
240             .y_label_area_size(30)
241             .build_cartesian_2d(0i32..max_x, 0.0..max_y)?;
242
243     chart

```

```

244         .configure_mesh()
245         .x_labels(3)
246         .y_labels(2)
247         .label_style((FONT, 30))
248         .max_light_lines(4)
249         .draw()?;
250
251     chart.draw_series(
252         fitness
253         .iter()
254         .map(|x| Circle::new((x.0, x.1), 1, BLUE.mix(0.25).filled())),
255     )?;
256 }
257 root.present()?;
258 Ok(())
259 }
260

```

Source Code 9: utils/graph/hist.rs

```

1  use plotters::prelude::*;
2  use std::error::Error;
3
4  use crate::utils::{graph::*, data*};
5
6  pub struct Histogram2 {
7      datas: [Vec<f64>; 2],
8      title: String,
9      axes_desc: (String, String),
10 }
11
12 impl Histogram2 {
13     pub fn new(datas: [Vec<f64>; 2], title: &str, axes_desc: (&str, &str)) -> Histogram2 {
14         Histogram2 {
15             datas,
16             title: title.into(),
17             axes_desc: (axes_desc.0.into(), axes_desc.1.into()),
18         }
19     }
20
21     pub fn mean(&self) -> Vec<f64> {
22         self.datas
23             .iter()
24             .map(|l| {
25                 l.iter()
26                     .fold(0f64, |mean, &val| mean + val / l.len() as f64)
27             })
28             .collect()
29     }
30
31     pub fn max_x(&self) -> f64 {
32         self.datas
33             .iter()
34             .fold(0f64, |max, l| max.max(l.len() as f64))
35     }
36
37     pub fn max_y(&self) -> f64 {
38         self.datas
39             .iter()
40             .fold(f64::MIN, |max, l| {
41                 let l_max = data::max(l);
42                 if l_max > max {
43                     l_max
44                 }
45                 else {
46                     max
47                 }
48             })
49     }
50
51     pub fn draw_hist(&self, path: String, max_y: f64) -> Result<(), Box<dyn Error>> {
52         let root = BitMapBackend::new(&path, (1024, 768)).into_drawing_area();
53         root.fill(&WHITE)?;
54
55         let n = self.max_x();

```

```

56     let mut chart = ChartBuilder::on(&root)
57         .caption(&self.title, (FONT, CAPTION, FontStyle::Bold).into_font())
58         .margin(20)
59         .x_label_area_size(70)
60         .y_label_area_size(90)
61         .build_cartesian_2d((1..n as u32).into_segmented(), 0.0..max_y)?
62         .set_secondary_coord(0.0..n, 0.0..max_y);
63
64     chart
65         .configure_mesh()
66         .disable_x_mesh()
67         .y_max_light_lines(0)
68         .y_desc(&self.axes_desc.1)
69         .x_desc(&self.axes_desc.0)
70         .axis_desc_style((FONT, AXIS_LABEL))
71         .y_labels(3)
72         .label_style((FONT, AXIS_LABEL - 10))
73         .draw()?;
74
75
76     let colors = [&RED, &BLUE];
77
78     for (i, dt) in self.datas.iter().enumerate() {
79         let color = colors[i % 2];
80         let offset = i as f64 * 0.4;
81         chart.draw_secondary_series(dt.iter().zip(0..).map(|(y, x)| {
82             Rectangle::new(
83                 [(x as f64 + 0.1 + offset, *y), (x as f64 + 0.5 + offset, 0f64)],
84                 Into::::into(color.mix(0.5)).filled(),
85             )
86         })))?;
87     }
88
89     let v: Vec<usize> = (0..(n + 1.0) as usize).collect();
90     let mean = self.mean();
91     for (j, m) in mean.iter().enumerate() {
92         let color = colors[j % 2];
93         chart
94             .draw_secondary_series(LineSeries::new(
95                 v.iter().map(|i| (*i as f64, *m)),
96                 color.filled().stroke_width(2),
97             ))?
98             .label(format!("mean: {:.3}", m))
99             .legend(move |(x, y)| PathElement::new(vec![(x, y), (x + 20, y)], color.filled()));
100     }
101
102     chart
103         .configure_series_labels()
104         .label_font((FONT, SERIE_LABEL).into_font())
105         .background_style(&WHITE)
106         .border_style(&BLACK)
107         .draw()?;
108
109     root.present()?;
110     Ok(())
111 }
112 }
113
114 /// Draw histogram of given datas
115 /// axes_desc - (for x, for y)
116 pub fn draw_acc_hist(
117     datas: &Vec<f64>,
118     title: &str,
119     axes_desc: (&str, &str),
120     path: String,
121 ) -> Result<(), Box<dyn Error>> {
122     let n = datas.len();
123     let mean = datas
124         .iter()
125         .fold(0.0f64, |mean, &val| mean + val / n as f64);
126
127     let root = BitMapBackend::new(&path, (1024, 768)).into_drawing_area();
128     root.fill(&WHITE)?;
129
130     let mut chart = ChartBuilder::on(&root)
131         .caption(title, ("Hack", 44, FontStyle::Bold).into_font())

```



```

132     .margin(20)
133     .x_label_area_size(50)
134     .y_label_area_size(60)
135     .build_cartesian_2d((1..n).into_segmented(), 0.0..1.0)?
136     .set_secondary_coord(1..n, 0.0..1.0);
137
138     chart
139         .configure_mesh()
140         .disable_x_mesh()
141         .y_max_light_lines(0)
142         .y_desc(axes_desc.1)
143         .x_desc(axes_desc.0)
144         .axis_desc_style(("Hack", 20))
145         .y_labels(3)
146         .draw()?;
147
148     let hist = Histogram::vertical(&chart)
149         .style(RED.mix(0.5).filled())
150         .margin(10)
151         .data(datas.iter().enumerate().map(|(i, x)| (i + 1, *x)));
152
153     chart.draw_series(hist)?;
154
155     chart
156         .draw_secondary_series(LineSeries::new(
157             datas.iter().enumerate().map(|(i, _)| (i + 1, mean)),
158             BLUE.filled().stroke_width(2),
159         ))?
160         .label(format!("mean: {:.3}", mean))
161         .legend(|(x, y)| PathElement::new(vec![(x, y), (x + 20, y)], &BLUE));
162
163     chart
164         .configure_series_labels()
165         .label_font(("Hack", 14).into_font())
166         .background_style(&WHITE)
167         .border_style(&BLACK)
168         .draw()?;
169
170     root.present()?;
171     Ok(())
172 }
173
174 pub fn draw_acc_2hist(
175     datas: [&Vec<f64>; 2],
176     title: &str,
177     axes_desc: (&str, &str),
178     path: String,
179 ) -> Result<(), Box<dyn Error>> {
180     let hist2 = Histogram2::new([datas[0].clone(), datas[1].clone()], title, axes_desc);
181     hist2.draw_hist(path, 1.0)
182 }
183
184 pub fn draw_2hist(
185     datas: [&Vec<f64>; 2],
186     title: &str,
187     axes_desc: (&str, &str),
188     path: String,
189 ) -> Result<(), Box<dyn Error>> {
190     let hist2 = Histogram2::new([datas[0].clone(), datas[1].clone()], title, axes_desc);
191     hist2.draw_hist(path, hist2.max_y())
192 }
193
194

```