

Name : Yun Zhou
ID: 300442776

This report contains part1 and part2, please just look at this report, thanks.

PART 1:

1. Report the output and predicted class of the first instance in the dataset using the provided weights.

```
<terminated> a2Part1 [Java Application] C:\Users\11973\Downloads\Compressed\jdk-11.0.8\bin
[[0.15636363636363626, 0.48148148148148157, 0.34545454545454546, 0.18
First instance has label Adelie, which is 0 as an integer, and [1, 0,
-----
Predicted label for the first instance is: Chinstrap
which is: false
```

The screenshot above indicates the output and predicted class of the first instance in the dataset using the provided weights.

For the first instance, it has the label Adelie, which is 0 as an integer, and [1, 0, 0] as a list of outputs. predicted labels. For my algorithm, it predicts the label of the first instance is Chinstrap, but the real label is Adelie, which is **incorrect**. Due to this part only does the forward_pass and the sigmoid activation function, so this result indicates that the provided weights need to be updated. (The sampleoutput.txt can also be found in directories.)

2. Report the updated weights of the network after applying a single back-propagation based on only the first instance in the dataset.

```
double[][] initial_hidden_layer_weights =
    new double[][] { { -0.28, -0.22 }, { 0.08, 0.20 }, { -0.30, 0.32 }, { 0.10, 0.01 } };

double[][] initial_output_layer_weights =
    new double[][] { { -0.29, 0.03, 0.21 }, { 0.08, 0.13, -0.36 } };
```

The above screenshot indicates the initial weights before training.

And below screenshot indicates updated weights of the network after applying a single back-propagation based on only the first instance in the dataset.

We can see that each weight value is updated so weights are updated successfully!

Name : Yun Zhou
ID: 300442776

```
Predicted label for the first instance is: Chinstrap
which is: false

OL betas: [0.5237282814572664, -0.5212308085921598, -0.47617240962812674]
HL betas: [-0.06672924754636447, 0.03629970824107195]
Placeholder
Weights after performing BP for first instance only:
Hidden layer weights:
[[-0.28052064067333937, -0.219718347073908], [0.07839682135470441, 0.200867277528664],
[-0.30115025265040085, 0.3206222564646219], [0.09937879128267824,
0.010336057595779677]]
Output layer weights:
[[-0.27752533507224475, 0.017579233592740794, 0.19865828140016373],
[0.09419939713573042, 0.11586195332955135, -0.37290981100762555]]
```

3. Report the final weights and accuracy on the test set after 100 epochs. Analyse the test accuracy and discuss your thoughts.

```
OL betas: [-0.4060403247774603, 0.5738617967639754, -0.4104645429964902]
HL betas: [0.28192962371597796, 0.27282040103510186]
Placeholder
Hidden layer weights
[[0.9332845187717153, -9.811201473880523], [-7.287868745252471, 5.203579457295289],
[2.3884053602160016, -1.4066374812628024], [2.4705404968671454, 1.4293400774025082]]
Output layer weights
[[-9.675231574564958, -2.4444027521516194, 3.241704546503655], [4.909408052058938,
-2.8731616123986408, -11.650203888258744]]
For 100 th Epoch, the correct predicted Number is: 222.0
The size of instances is 268
which means, we got: 222.0 out of 268
acc = 82.84 %

After training:
Hidden layer weights:
[[0.9332845187717153, -9.811201473880523], [-7.287868745252471, 5.203579457295289],
[2.3884053602160016, -1.4066374812628024], [2.4705404968671454, 1.4293400774025082]]
Output layer weights:
[[-9.675231574564958, -2.4444027521516194, 3.241704546503655], [4.909408052058938,
-2.8731616123986408, -11.650203888258744]]
After 100 Epochs on training,
For the TEST set, we got 53.0 correct predicted test instances
The size of test set is 65
which means, we got: 53.0 out of 65
acc = 81.54 %
Finished!
```

From the screenshot above, we can see the final weights and accuracy after 100 Epochs.

By observing the accuracy data output, we can see that on 100th epochs, we've got 82.84% on the Training dataset. After that, for the test set, we've got 81.54% accuracy, the accuracy on the test set is a little bit lower than on training, but it's in the fine since it's pretty close.

Name : Yun Zhou
ID: 300442776

Since I've got around 82% accuracy, so I think this accuracy result is a little bit low, I think this might be the problem of my network does not have Bias, so the network can not predict correctly for some particular instances. Therefore, if we have the bias, then it can handle these particular instances and thus we should get a much better accuracy result on both sets.

For instance, without bias, in the OR case of perceptron, it can not predict the instance with the coordination (1,0) on the x-y axis since the function always goes through the origin(0,0), so although (1,0) and (0,0) belongs to different class label, it will always predict they belong to the same class label.

4. Discuss how your network performed compared to what you expected. Did it converge quickly? Do you think it is overfitted?

My network performance does not perform pretty well as I expected.

First, I can see that for each epoch, the weights are updated successfully, but it only got 82.84% on the training set and 81.54% on the test set, lower than I expected. Thinking about the reason, it should be the lack of Bias. Bias is a very important variable since it decides where the intersection point is on the x-axis of the function. Without bias means the lack of flexibility on function, thus, can't predict on some particular instances and will predict it to the wrong class label. (p.s. The example is shown in Q4, above.)

Also, it might be the problem of the gradient descent lead to the local minima and thus stuck on the local minima, but the probability of this is pretty low since we've run for 100 epochs and the online learning is stochastic descent. The most possible problem should be the lack of Bias.

Did it converge quickly?

```
0.35450550955391996, -0.8132280387588087, -0.67735889387  
For 1 th Epoch, the correct predicted Number is: 122.0  
The size of instances is 268  
which means, we got: 122.0 out of 268  
acc = 45.52 %
```

```
For 3 th Epoch, the correct predicted Number is: 134.0  
The size of instances is 268  
which means, we got: 134.0 out of 268  
acc = 50.00 %
```

```
For 11 th Epoch, the correct predicted Number is: 213.0  
The size of instances is 268  
which means, we got: 213.0 out of 268  
acc = 79.48 %
```

Screenshots above are the accuracy after implement bias.

Name : Yun Zhou
ID: 300442776

Did it converge quickly? (cont.)

I think it converges quickly since the learning rate of our network is 0.2, it's a good point and it performs well and converges quickly. By debugging to see, I can see that it does not cause slow convergence and oscillating behaviour, which means it's not a too small learning rate and a too-large learning rate.

Do you think it is overfitted?

By looking at the accuracy results, we can see that 82.84% on the training set and 81.54% on the test set. Although the test accuracy is a little bit poorer than the training accuracy, but it's not too poor, I think it's fine and within the range.

Thus, I think it might have a little bit overfitted since the training is a little bit too long, but it's fine and in the range, can address this by reducing the training times.

PART 1, **Bonus Question:** implement the feature of add bias Nodes into my network.

```
Placeholder
Hidden layer weights
[[-2.2642011098720185, -10.886486164212831], [-8.148159426853677,
5.6528086573906755], [3.1322479761709143, -1.2922861445471252],
[3.9457565026201458, 1.9553289577763402]]
Output layer weights
[[-6.172496125492204, -3.976498492529592, 4.299073392243547], [4.73362019325906,
-3.6342137394257112, -11.276018451761495]]
For 100 th Epoch, the correct predicted Number is: 246.0
The size of instances is 268
which means, we got: 246.0 out of 268
acc = 91.79 %

After training:
Hidden layer weights:
[[-2.2642011098720185, -10.886486164212831], [-8.148159426853677,
5.6528086573906755], [3.1322479761709143, -1.2922861445471252],
[3.9457565026201458, 1.9553289577763402]]
Output layer weights:
[[-6.172496125492204, -3.976498492529592, 4.299073392243547], [4.73362019325906,
-3.6342137394257112, -11.276018451761495]]
After 100 Epochs on training,
For the TEST set, we got 59.0 correct predicted test instances
The size of test set is 65
which means, we got: 59.0 out of 65
acc = 90.77 %
Finished!
```

The screenshot above indicates the accuracy result after I implement the feature of adding the Bias into my network.

We can see that, I got 91.79% on the training set and got 90.77% on the test set. Before add bias nodes into the network, I only got 82.84% on the training set and 81.54% on the test set, we can see that it increase about 10% accuracy which also proves my thoughts that lacking the bias causes the network can not adjust some particular instances, within use the bias, we can get a much better accuracy result.

Name : Yun Zhou
ID: 300442776

PART 2:

```
this.R = new Constant(a_conf: config, a_type: CommandGene.DoubleClass, a_value: new Random().nextDouble());
```

1. Justify the terminal set you used for this task.

The screenshot on the right is the terminal Set that I use for my task, which are 20 x as the input variables. Except these x variables, a random constant R is also belongs to the terminal set that I use for this task, which is shown at the above screenshot.

The terminal sets is a very important component in the genetic programming. It is the alphabet of the programs to be made. The terminal set consists of the a set of terminals which can be represented as variables(in our assignment, it is x) and constants(Random generated, since it's constant, will not be changed) of the programs.

1	x
2	-----
3	-2.00
4	-1.75
5	-1.50
6	-1.25
7	-1.00
8	-0.75
9	-0.50
10	-0.25
11	0.00
12	0.25
13	0.50
14	0.75
15	1.00
16	1.25
17	1.50
18	1.75
19	2.00
20	2.25
21	2.50
22	2.75
23	

2. Justify the function set you used for this task.

```
new Add(a_conf: config, a_returnType: CommandGene.DoubleClass),  
new Multiply(a_conf: config, a_returnType: CommandGene.DoubleClass),  
// add multiply & subtracts  
new org.jgap.gp.function.Subtract(a_conf: config, a_returnType: CommandGene.DoubleClass),  
new org.jgap.gp.function.Multiply(a_conf: config, a_returnType: CommandGene.DoubleClass)
```

The function set consists of a set of functions or operators, and it form the root and the internal nodes of the tree representation of our assignment program.

The above screen shot indicate the function set that I use for this task, which are $\{+, -, *, \div\}$ addition, subtraction, division and multiplication.

Name : Yun Zhou
ID: 300442776

3. Formulate the fitness function and describe it using plain language (and mathematical formula, or other formats you think appropriate, e.g. good pseudo-code).

My fitness function is within MyFitnessFunction.java file and it actually came from the provided JGAP tutorial that is attached in the correspond comp307 course website.

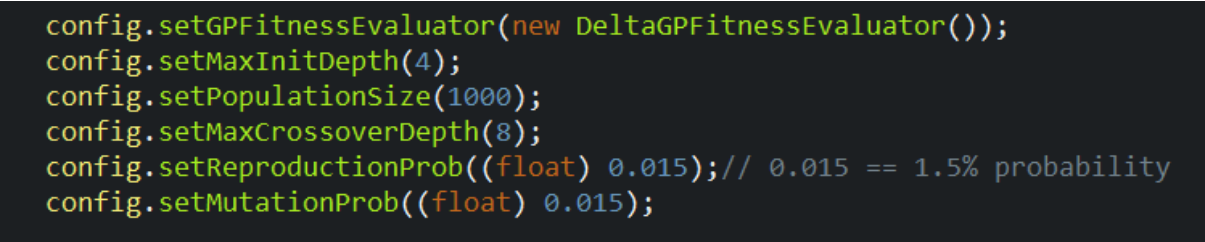
The logic of the fitness function is:

iterative through all inputs, calculate and predict the corresponding output value of this input via the provided algorithm within the IGPPProgram.class. Then, calculate the error between the real output value and this predicted output value. Increment each error into the long result and finally return it.

The pseudo-code is shown below:

```
evaluate(IGPPProgram program){  
    result = 0;  
    Iterative each input which is each x variable:{  
        //calculate the predicted_output_value by provided tool class  
        predict_Value = program.execute_double();  
  
        //get the difference error between predictValue & real Value  
        Error = predictValue - realValue(x);  
        If (Error<0){Error *= -1;}//get the absolute value  
        result += Error;  
    }  
    return result;  
}
```

4. Justify the relevant parameter values and the stopping criteria you used.

A screenshot of a code editor showing the configuration of a Genetic Programming (GP) system using the JGAP library. The code is in Java and sets various parameters for a DeltaGPFitnessEvaluator, including maximum initial depth, population size, maximum crossover depth, reproduction probability, and mutation probability.

```
config.setGPFitnessEvaluator(new DeltaGPFitnessEvaluator());  
config.setMaxInitDepth(4);  
config.setPopulationSize(1000);  
config.setMaxCrossoverDepth(8);  
config.setReproductionProb((float) 0.015);// 0.015 == 1.5% probability  
config.setMutationProb((float) 0.015);
```

The above screenshot indicate the parameter values that I use for my program.

Name : Yun Zhou
ID: 300442776

For the termination criteria, it is shown in the below screenshot, I set the termination stop criteria to be Either it reaches the 200th generation OR the fitness value is 0 which is lower enough.

```
/*
 * loop 200 generation times, stop it till it's in the 200th generation
 * OR the fitness value reaches 0
 */
int generation_time = 0;
while (++generation_time < 200) {
    gp.evolve(generation_time); // evolve it
    double fitnessValue = gp.getFittestProgramComputed().getFitnessValue();
    // check whether the fitness value is smaller than a predefined value,
    // if so, then meet the stopping criteria
    if (fitnessValue == 0) {
        // System.out.println("stop time" + time);
        break;
    }
}
```

5. Different GP runs will produce a different best solution. List three different best solutions evolved by GP and their fitness values (you will need to run your GP system several times with different random seeds).

```
Final Formula:
Best solution fitness: 0.0
Best solution: (((X * X) - X) - X) * (X * X)) + ((X * X) + 1.0)
Depth of chrom: 5
```

```
Final Formula:
Best solution fitness: 0.0
Best solution: ((X * X) - X) * ((X * X) - X)) + 1.0
Depth of chrom: 4
```

```
Final Formula:
Best solution fitness: 0.0
Best solution: (X * ((X * ((X * X) - X)) - ((X * X) - X))) + 1.0
Depth of chrom: 6
```

6. (optional, bonus, 5 marks) Analyse one of the best programs from above and explain how different parts the tree "work together" to solve the task.