Cam Olssen
300492582
cam.olssen@gmail.com

COMP307 Assignment 2 Report:
Part 1 (Neural Network)

a. The screenshot for the first instance, run using the provided weights, shows that it has the label Adelie. However, my algorithm predicts that it will have the label Chinstrap, which is incorrect.

```
C:\Users\camol\.jdks\corretto-1.8.0_362\bin\java.exe ...
[[0.15636363636363626, 0.48148148148148157, 0.34545454545454546, 0.1865671641791045], [0.258181
First instance has label Adelie, which is 0 as an integer, and [1, 0, 0] as a list of outputs.
Predicted label for the first instance is: Chinstrap
```

b. The first attached screenshot shows what the initial weights are set to, which is unchanged from the provided skeleton code.

```
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 } };
```

After applying a single back-propagation instance and updating the weights, these are the output weights.

```
Weights after performing BP for first instance only:
Hidden layer weights:
[[-0.2805206406733937, -0.219718347073908], [0.07839682135470441, 0.200867277528664], [-0.30115025265040085, 0.3206222564646219], [0.09937879128267824, 0.01033605759577967]]
Output layer weights:
[[-0.27752533507224475, 0.017579233592740794, 0.19865828140016373], [0.09419939713573042, 0.11586195332955135, -0.3729098110076255]]
```

c. The screenshots show the final weights and accuracy for the test set and for the final epoch after 100 epochs of training.

```
Hidden layer weights
[[0.9332845187717151, -9.811201473880523], [-7.287868745252472, 5.203579457295289], [2.3884053602160025, -1.4066374812628022], [2.4705404968671454, 1.4293400774025078]]
Output layer weights
[[-9.675231574564956, -2.44440275215162, 3.241704546503655], [4.909408052058938, -2.8731616123986403, -11.650203888258744]]
222.0 out of 268 correct results.
Accuracy for epoch 99 = 82.83582089552239%
```

```
After training:
Hidden layer weights:
[[0.9332845187717151, -9.811201473880523], [-7.287868745252472, 5.203579457295289], [2.3884053602160025, -1.4066374812628022], [2.4705404968671454, 1.4293400774025078]]
Output layer weights:
[[-9.675231574564956, -2.44440275215162, 3.241704546503655], [4.909408052058938, -2.8731616123986403, -11.650203888258744]]
For the test set:
53.0 out of 65 correct results.
Accuracy for test set = 81.53846153846153%
```

It can be seen from this data that we get an accuracy of about 82.83% on the training set and 81.53% on the test set. The accuracy on the test set is slightly lower than the accuracy on the training set, but close enough to be acceptable. This general accuracy is lower than I would like, likely due to the fact that I have not yet implemented bias nodes into my network.

d. My neural network does not perform particularly well, as I stated in the previous question. The weights are updated successfully on each epoch, but the final accuracy is still only 82.83% for the training set and 81.53% for the test set. This is likely due to

Cam Olssen
300492582
cam.olssen@gmail.com

my not having implemented bias nodes meaning that the function is inflexible and cannot predict correctly for some instances.

```
139.0 out of 268 correct results.
Accuracy for epoch 0 = 51.865671641791046%
```

```
178.0 out of 268 correct results.
Accuracy for epoch 4 = 66.41791044776120%
```

```
213.0 out of 268 correct results.
Accuracy for epoch 9 = 79.47761194029852%
```

As this data indicates, the neural network converges quite quickly. With a learning rate of 0.2, the network performs well and learns quickly. Convergence is at an acceptable speed, and there are no major issues, so I think that the learning rate is at a good level.

By comparing training and testing accuracy, we can see that the testing accuracy is a little over 1% lower than the training accuracy. While I consider this an acceptable difference, it is possible that the network is slightly overfitting.

Adding Bias Nodes

```
Hidden layer weights
[[-2.2642011098720167, -10.886486164212833], [-8.14815942685368, 5.652808657390677], [3.1322479761709148, -1.2922861445471259], [3.9457565026201453, 1.955328957763413]]
Output layer weights
[[-6.172496125492204, -3.976498492529591, 4.299073392243546], [4.73362019325906, -3.6342137394257117, -11.276018451761498]]
246.0 out of 268 correct results.
Accuracy for epoch 99 = 91.7910447761194%

After training:
Hidden layer weights:
[[-2.2642011098720167, -10.886486164212833], [-8.14815942685368, 5.652808657390677], [3.1322479761709148, -1.2922861445471259], [3.9457565026201453, 1.955328957763413]]
Output layer weights:
[[-6.172496125492204, -3.976498492529591, 4.299073392243546], [4.73362019325906, -3.6342137394257117, -11.276018451761498]]
For the test set:
59.0 out of 65 correct results.
Accuracy for test set = 90.76923076923077%
Finished!
```

These screenshots show the training and testing accuracy for my network after the addition of bias nodes. This has led to an accuracy of 91.79% on the final epoch of the training set and an accuracy of 90.76% on the test set, for an increase of about 9% accuracy. This is likely due to the addition of bias nodes giving the function greater flexibility and allowing the algorithm to make more accurate predictions for each instance.

Part 2:

a. For the terminal set for this task, I chose to use the x variables from the provided data in regression.txt, displayed in screenshot 1 below. I also added a random constant R, which is displayed in screenshot 2. The terminal set is very important to genetic programming, as it specifies the candidate solutions that can be generated by the algorithm.

Cam Olssen
300492582
cam.olssen@gmail.com

x
-2.00
-1.75
-1.50
-1.25
-1.00
-0.75
-0.50
-0.25
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
2.25
2.50
2.75

```
this.R = new Constant(configuration, CommandGene.DoubleClass, new Random().nextDouble());
```

b. For the function set, I used the following operators (*+-/) - addition, multiplication, subtraction and division. The code for this is shown in the attached screenshot.

```
new Add(configuration, CommandGene.DoubleClass),
new Multiply(configuration, CommandGene.DoubleClass),
new Subtract(configuration, CommandGene.DoubleClass),
new Divide(configuration, CommandGene.DoubleClass)
```

c. My fitness function is stored in the FitnessFunction.java file, and based on the JGAP tutorial provided in the assignment brief (https://cvalcarcel.wordpress.com/2009/08/04/jgap-a-firstsimple-tutorial/). The logic of the fitness function iterates through all inputs, calculates and predicts the output via the JGAP-provided algorithm in the IGPProgram class. It then calculates the error between the real output and predicted output and adds each error into the long result, and then returns the long result.

d. The first screenshot below shows the parameters that I used for my program.

```
configuration.setFitnessEvaluator(new DeltaGPFitnessEvaluator());
configuration.setMaxInitDepth(4);
configuration.setPopulationSize(1000);
configuration.setMaxCrossoverDepth(8);
configuration.setReproductionProb((float)0.015);
configuration.setMutationProb((float)0.015);
```

The screenshot below shows the stopping criteria for my program. My stopping criteria was either after the 500th generation, or when the fitness reached 0.

Cam Olssen
300492582
cam.olssen@gmail.com

```
    int generation = 0;
    while(++generation < 500){
        genotype.evolve( a_evolutions: 1);
        double fitness = genotype.getFittestProgramComputed().getFitnessValue();
        if(fitness == 0){
            break;
        }
    }
    System.out.println("Best Formula: ");
    genotype.outputSolution(genotype.getAllTimeBest());
}
```

e. Three different best solutions generated are:

```
Formula:
[JGAP][12:44:07] INFO GPGenotype - Best solution fitness: 0.0
[JGAP][12:44:07] INFO GPGenotype - Best solution: (((X * X) - X) * ((X * X) - X)) + 1.0
[JGAP][12:44:07] INFO GPGenotype - Depth of chrom: 4
```

```
Best Formula:
[JGAP][13:06:05] INFO GPGenotype - Best solution fitness: 0.0
[JGAP][13:06:05] INFO GPGenotype - Best solution: (1.0 + (X * X)) + ((X * X) * (((X * X) - X) - X))
[JGAP][13:06:05] INFO GPGenotype - Depth of chrom: 5
```

```
Best Formula:
[JGAP][13:06:32] INFO GPGenotype - Best solution fitness: 0.0
[JGAP][13:06:32] INFO GPGenotype - Best solution: (X - X) + (1.0 + (((X * X) - X) * ((X * X) - X)))
[JGAP][13:06:32] INFO GPGenotype - Depth of chrom: 5
```