# COMP307

## Assignment 2

**Neural and Evolutionary Learning**

---

William Kilty

300473541

# Part 1:

1. Output: [0.47627171854273365, 0.5212308085921598, 0.47617240962812674]

   Predicted label for the first instance is: Chinstrap

2. 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, 0.01033605759577967]]

   Output layer weights:

   [[-0.27752533507224475, 0.017579233592740794, 0.19865828140016373], [0.09419939713573042, 0.11586195332955135, -0.37290981100762555]]

3. After training:

   Hidden layer weights:

   [[0.9332845187717129, -9.81120147388053], [-7.287868745252472, 5.203579457295295], [2.3884053602160047, -1.4066374812628015], [2.4705404968671454, 1.4293400774025078]]

   Output layer weights:

   [[-9.675231574564949, -2.44440275215162, 3.2417045465036542], [4.909408052058934, -2.87316161239864, -11.650203888258744]]

   Training Accuracy: 82.84%

   Test Accuracy: 81.54%

   The training and test accuracies are very similar, indicating that the weights are a good reflection of the actual patterns observed in the data. However, as neither are close to 100%, this likely indicates that we have not performed enough epochs, and the weights gained are not as accurate as they could be

4. The network converged rapidly to begin with, starting at 50% accuracy and reaching 78% after only 6 epochs. However, from this point progress slowed down to a crawl, and in the next 94 epochs only managed to reach 81%. A measly 3% increase in accuracy. This lack of progress is to be expected without bias nodes, as there is only so good a model can get without being able to shift its dividers around. Because of the close relationship between training and testing accuracy it seems highly unlikely that the model is overfitted. To gain a more accurate model, one will need to include bias nodes and/or extra hidden nodes and layers.

# Part 1.1:

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]]

After training:

Hidden layer weights:

[[-1.4189792045294503, -11.165881017834813, 0.0], [-6.191546481646949, 5.370284807111607, 0.0], [4.2017236362385795, -0.8391904626087715, 0.0], [4.654311004926817, 2.5110428280858144, 0.0], [-1.3527285009235708, 0.13490800029268865, 0.0]]

Output layer weights:

[[-2.7053078397343455, -7.0378764358459405, 7.54821907920647], [9.253328241444699, -8.180332809702046, -5.500190076552467], [-3.7461833336751442, 3.7828810892344857, -2.7787306001882692]]

Training Accuracy: 99.25%

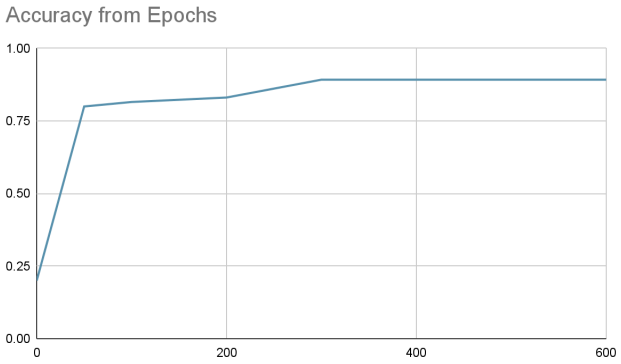
Test Accuracy: 100.00%

After retraining the network with the inclusion of bias nodes, I reached a training accuracy of 99.25% and a testing accuracy of 100%. This is significantly better than the testing accuracy of 81.54% achieved without biases. In fact this result suggests we have essentially found a perfect model for distinguishing these species of penguin from one another. This difference is likely caused by the data containing significant distinctions that aren't centred around 0, requiring our model to be able to shift around in order to match the distinctions.

# Part 1.2:

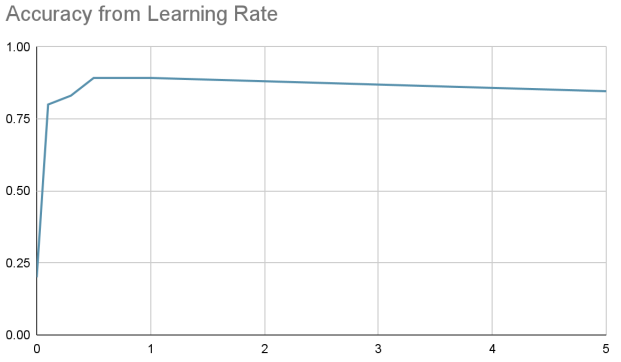
First Element to alter: Epochs

50 Epochs Accuracy: 80.00%
100 Epochs Accuracy: 81.54%
200 Epochs Accuracy: 83.08%
300 Epochs Accuracy: 89.23%
400 Epochs Accuracy: 89.23%
500 Epochs Accuracy: 89.23%
600 Epochs Accuracy: 89.23%




Second Element to alter: Learning Rate

0.10 LearnR Accuracy: 80.00%
0.20 LearnR Accuracy: 81.54%
0.30 LearnR Accuracy: 83.08%
0.40 LearnR Accuracy: 86.15%
0.50 LearnR Accuracy: 89.23%
1.00 LearnR Accuracy: 89.23%
5.00 LearnR Accuracy: 84.62%




In the epoch plot, the accuracy increases rapidly to begin with, but plateaus at 89.23%. This is likely the maximum accuracy reachable without biases, and beyond this point one would expect no further improvement, and perhaps overfitting leading to a lower accuracy after many many more epochs.

The rapid increase at the start is because the network is able to converge on a fairly good model very quickly, by epoch 6, from part 1. And beyond this point there are small, slow improvements until it reaches its local maximum which is likely around 89.23% as seen in the plot.

In the Learning Rate plot, the accuracy also increases rapidly to begin with, reaching the same peak of 89.23% with a learning rate of 0.5. This indicates that the starting learning rate of 0.2 used in part 1 is too low, and causes our model to adapt slower than it should be. This peak of 89.23% is maintained for a stretch of learning rates until around 5.0, where the changes each step become so large we start to lose accuracy again. Continuing this plot would likely lead to lower and lower accuracies.

The decrease in accuracy beyond 5.0 happens because at this learning rate, weight change causes such a large difference in the model's weights, that the network ends up jumping clean over the local maximum and must double back, missing the maximum again, never getting any closer to the correct result.

# Part 2:

1. I used two terminals. The variable x, and a constant between -100 and 100.

   X was chosen because we need some variable to enter into the function. It would be a pretty boring function without one.

   The constant was chosen so that the function could be shifted up or down within a range of 100 (larger than it should need)

2. The function set I used was the standard +-x/ operators. There was no obvious reason to exclude any of them, and more complicated operators like exponents can be achieved with combinations of multiplications.

3. For each x value in *regression.txt*, we run it through the given instance of the GPProblem, and subtract the result from the corresponding y value in *regression.txt*.

   These differences are summed up, and that acts as the fitness function. The closer to 0, the better the fitness.

4. The parameter values I chose were population = 1000, tree depth = 10.

   Population of 1000 ensured lots of space for mutations to occur, with a high chance that at least one of them is beneficial. This has the effect of reducing the total evolutions needed.

   Tree depth of 10 allowed the function to experiment with a lot of different options without letting it go crazy with recursive functions hundreds of operations long.

   The stopping criteria I used was a best fitness of 0.1 or less. I used this number as it ensured the produced function would at the very least match the input values very closely.

5. Best solution: (56.0 / 56.0) + ((x - (x * x)) * (x - (x * x)))
   Fitness achieved: 3.750000000124487E-5

   Best solution: (1.0 + (x * ((((x * x) - x) - x) * x))) + (x * x)
   Fitness achieved: 3.750000000124487E-5

   Best solution: (x * x) + ((93.0 / 93.0) + ((((x * x) - x) - x) * (x * x)))
   Fitness achieved: 3.750000000124487E-5

6. If we look at the third solution, we can work our way up the tree from the terminal operators.

   x*x = x^2

   93/93 = 1

   x*x = x^2

   x*x = x^2

   At this point we have

   $$x^2 + 1 + ((x^2 - x) - x) * (x^2)$$

   now,

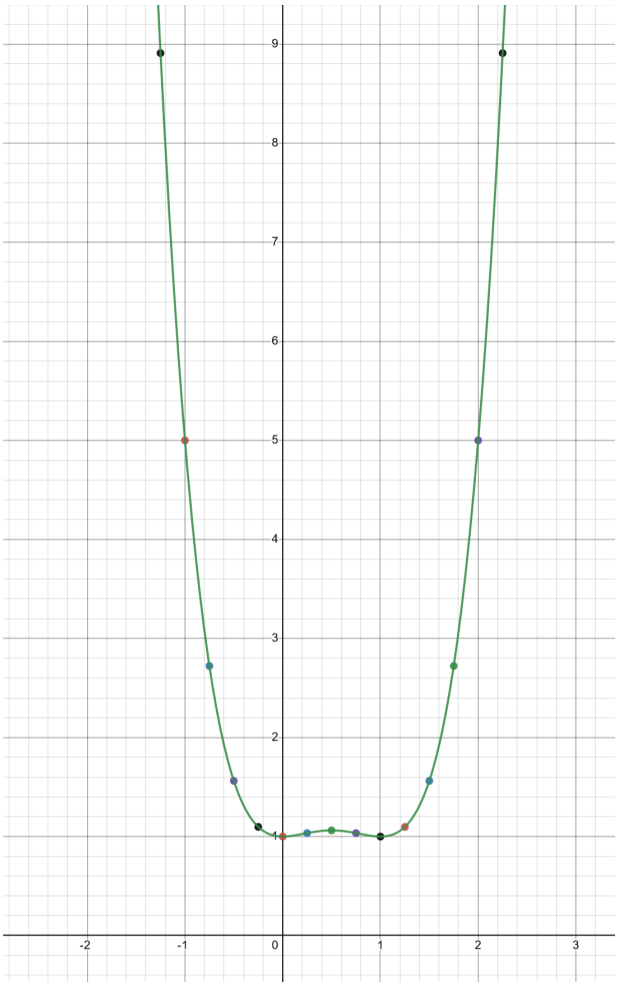
   $$((x^2 - x) - x) = (x^2 - 2x)$$

   $$(x^2 - 2x) * (x^2) = (x^4 - 2x^3)$$

   And our equation is now fully expanded. It can be rearranged as:

   $$x^4 - 2x^3 + x^2 + 1$$

   Putting this equation into a graph plotter against the points in *regression.txt*, we can observe what looks to be the correct equation:




In fact, all of our best solutions expand and rearrange into this equation, making this very likely to be the correct solution. Each branch in the tree contains information necessary to form this equation. With the terminal set chosen by the function, every operation, every branch in the tree is necessary - with no redundancy.