COMP307/AIML420 — Fundamentals of AI

Assignment 2

Part 1: Classifying Pingu with a Neural Network

(a) Report the output and predicted class of the first instance in the dataset using the provided weights.

Outputs from first instance’s feed forward pass using the provided weights:
Hidden layer outputs: [0.47745456164517874, 0.543466856571186]
Output layer outputs: [0.47627171854273365, 0.5212308085921598, 0.47617240962812674]

That translated for the output to be [0, 1, 0] so the predicted class was Chinstrap.

The actual class was an Adelie, so the first prediction is wrong.

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

Original weights:
```
Hidden layer weights:
[[-0.28, -0.22], [0.08, 0.2], [-0.3, 0.32], [0.1, 0.01]]
Output layer weights:
[[-0.29, 0.03, 0.21], [0.08, 0.13, -0.36]]
```

Updated weights after applying a single BP on only the first instance:
```
Hidden layer weights:
[[-0.28052064067333937, -0.219718347073908],
[0.07839682135470441, 0.200867277528664],
[-0.30115025265040085, 0.32062225646462l9],
[0.09937879128267824, 0.010336057595779677]]
Output layer weights:
[[-0.27752533507224475, 0.017579233592740794, 0.19865828140016373],
[0.09419939713573042, 0.11586195332955135, -0.37290981100762555]]
```

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

Note: The weights were left adjusted after the first single back propagation of the previous task.

Final weights and accuracy of the training set after 100 epochs:

```
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]]
Accuracy = 0.8283582089552238
```

Accuracy of the Neural Network on the test set:

```
Accuracy: 0.8153846153846154
```

The test accuracy is about the same as the accuracy of the training data using the neural network after 100 epochs. I'm wondering if it's because the test data is very similar/close to the data points to the training data regarding how much of each penguin class is given. Looking deeper into the results, I notice that a trend that could be due to all the instances being grouped by class type. When it came to the first 2 classes in the test set, the predictions were 100% accurate. The third class, however, the accuracy was close to 0% with perhaps the very last prediction was successful. Is it because the third class is too similar to the previous two? Or is it more likely a side effect or what happens when the instances aren't shuffled?

It's been a few days since I wrote the last paragraph, and a thought occurred to me. Aside from the fact the biases have yet to be implemented, would having only two nodes in the hidden layer impact the neural network's ability to classify among 3 possible classifications? Considering the hidden outputs are derived from the sigmoid function the nodes are very loosely following the binery model. With 2 nodes, that's 4 possible binary combinations, so in theory it could work. Would increasing the number of hidden nodes improve the likelihood of convergence? This is probably outside the scope of the assignment, but I just thought it was interesting to think about.

(d) Discuss how your network performed compared to what you expected. Did it converge quickly? Do you think it is overfitting, underfitting or neither?

Admittedly, I thought it would converge quickly, but instead it performed similarly to how perceptron performed during the last assignment. As mentioned above, I noticed it was had little difficulty classifying 2 out of 3 possible penguins and it struggled with the third. So in this case, my take is that it is underfitting.

1.1 Further Improving Your Network

After applying the biases to the NN the accuracy lifted to 100%. That was a huge increase compared to roughly 80% accuracy of the previous network. I suspect the possible reason for the change is to do with the output of the third penguin. Based on the nature of the activation function, the further away from zero the weighted sum is, the more definitive the result. In other words, the more likely the output is to be zero or one. I suspect the weighted sums for the third penguin is hovering much closer to zero. Causing large adjustments to the weights to the point that it's not converging; and having outputs hovering around 0.5 for each of the classes.

Since adding biases shifts the sigmoid function left or right, it appears that the outputs are being pushed closer to zero or one and therefore converging and creating more definitive results.

Part 2: Genetic Programming for Symbolic Regression

(a) Justify the terminal set you used for this task

There's only one input for a given output, or we can rephrase it as "given X we get a Y result." So it made sense to only need one variable X and a random constant for the terminal set. I et a limit for the constant to between 0 and 10. It seemed like a good starting point and I planned to adjust it if needed. I noticed that while I could have restricted the constant to whole numbers, I found that the code reached the best fitness when I didn't have that restriction

(b) Justify the function set you used for this task.

I used Plus, Minus, Multiply, and Divide for the function set. The lecture notes said to use a protected divide function, but I couldn't find the feature in the JGAP library. According to the source code if the formula divides by zero it will make a check and throw an error if it came up.

I also tried out using the "power of" function, but it doesn't appear to be used much. I assumed it's due to it created extremely large error rates when it isn't used correctly

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

The fitness function basically takes the sum total of errors the genetically engineered formula makes for it to be the fitness score. Essentially, the closer the score is to zero, the better the score is. So for each input X we have an expected output Y and the output derived from the generated formula. We take the difference between the two outputs to calculate the error. We do that for each X value given, than take the sum total of all the errors given to create the fitness score.

(d) Justify the parameter values and the stopping criteria you used.

I don't have a stopping criteria, but it does stop after 100 evolutions. I tried to set up a monitor where the program stops if the fitness is below 0.01, but I must be missing a step because the code I wrote for it won't be reached. I even set a breakpoint during debugging to check. Also it appears that if the monitor work, the code would stop stop in less than 75 generations.

As for other parameter values:

```
private final static int MAX_DEPTH = 17;
private final static int POPULATION = 1000;
private static final int GENERATIONS = 100;
private static final float REPRODUCTION_RATE = 0.05f;
private static final float CROSSOVER_RATE = 0.9f;
private static final float MUTATION_RATE = 0.05f;
```

Most of these were set after seeing the example in the lecture slide. I did, however, increase the population size and the number of evolutions to bring the fitness value down. I noticed that the larger the population the less evolutions I needed and vice versa. I also noticed that adding a mutation rate (example code didn't have it originally) also brought down the number of evolutions required.

Update, some random seeds didn't converge, so I increased the evolutions to 200, and I doubled the mutation rate to 0.1. Various experiments in different seeds indicate the GP converges around 150 evolutions.

(e) 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 code several times with different random seeds).

Seed: 0
(6.776434925614476 / 6.776434925614476) + ((X * X) + ((((X * X) - X) - X) * (X * X)))
Fitness: 3.75E-5

Seed: 25
((X - (X * X)) * (X - (X * X))) + (4.745128508142928 / 4.745128508142928)
Fitness: 3.75E-5

Seed: 654
(((6.479959364057998 / 6.479959364057998) + X) - X) + (((X * X) - X) * ((X * X) - X))

Fitness: 3.75E-5

Extra notes:-

It appears to be converging to this equation:
$1 + (x^2 - x)^2 = y$

It’s safe to assume the fitness values are zero with numbers that low.
I included the seed numbers mainly so I could recreate these equations if I wanted to.
It’s clear that the desired constant is 1, perhaps it’s easier to reach there by dividing a number by itself than if I took away the divide function, like if I enforced constants to be whole numbers.