Kahu Mulligan 300572890

# AI Assignment 2

*Part 1 Report.*

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

output: [0.47627171854273365, 0.5212308085921598, 0.47617240962812674]

Predicted label for the first instance is: Chinstrap

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

Hidden initial weights: [[-0.28, -0.22], [0.08, 0.2], [-0.3, 0.32], [0.1, 0.01]]

Output initial weights: [[-0.29, 0.03, 0.21], [0.08, 0.13, -0.36]]

OL betas: [0.5237282814572664, -0.5212308085921598, -0.47617240962812674]

HL betas: [-0.06672924754636447, 0.03629970824107195]

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 7]]

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.*

test predicted labels: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 0, 0, 0, 0, 0, 0, 0, 2, 0, 0, 0, 0, 1]

test hidden layer weights:

[[0.9332845187717169, -9.811201473880526], [-7.2878687452524735, 5.203579457295294], [2.388405360216003, -1.4066374812628042], [2.4705404968671445, 1.4293400774025085]]

test output layer weights:

[[-9.67523157456495, -2.44440275215162, 3.2417045465036542], [4.909408052058931, -2.87316161239864, -11.650203888258746]]

Test set accuracy: 0.8153846153846154

The test accuracy of 0.8153846153846154 suggests that the model has learned to generalize well and is performing well on previously unseen data. However, it is important to note that this accuracy may not be representative of the model's performance on all future data. It is possible that the model may not generalize as well on new data that is significantly different from the test set.

Overall, the test accuracy suggests that the model is performing well and is likely to be useful for classification similar data sets for penguin classification.

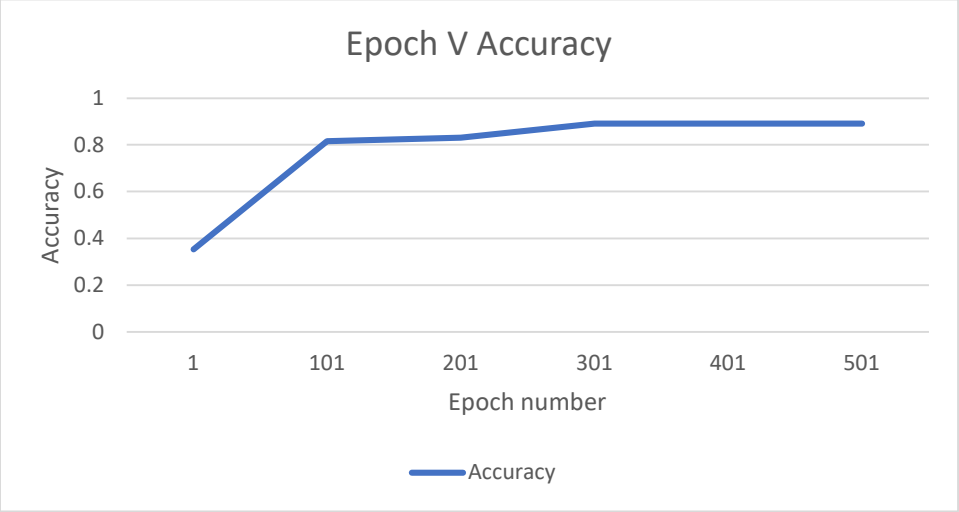
(d) *Discuss how your network performed compared to what you expected. Did it converge quickly? Do you think it is overfitting, underfitting or neither?*

The neural network's accuracy was consistent with my expectations, given the simplicity of the dataset, which only had four attributes for classification. There was no evidence of underfitting or overfitting since both the test and training data produced similar results in terms of accuracy, suggesting a balanced fit. By adding bias nodes, the neural network gained additional complexity and increased its capacity to fit a broader range of functions, which enabled it to match the desired classifications more closely.

(e) *Sensitivity Testing: Plot the test accuracy vs parameter value for each parameter. Briefly describe the pattern shown in the plot and discuss why this pattern may have occurred.*

Epoch vs Accuracy with no bias node.

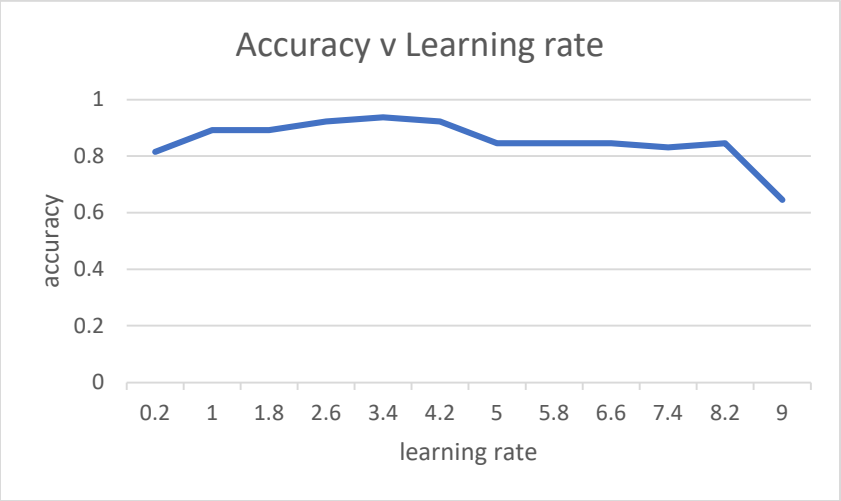
| Epoch Number | Accuracy |
|---|---|
| 1 | Test set accuracy: 0.35384615384615387 |
| 101 | Test set accuracy: 0.8153846153846154 |
| 201 | Test set accuracy: 0.8307692307692308 |
| 301 | Test set accuracy: 0.8923076923076924 |
| 401 | Test set accuracy: 0.8923076923076924 |
| 501 | Test set accuracy: 0.8923076923076924 |




Based on the plotted graph, it appears that the test accuracy increases as the parameter value increases. The initial accuracy at epoch 1 is low at 0.35384615384615387, but as the number of epochs increases, the accuracy steadily improves. At epoch 101, the accuracy jumps to 0.8153846153846154 and continues to increase with each epoch until it reaches a plateau at around epoch 301, with an accuracy of 0.8923076923076924. The accuracy remains constant from epoch 301 to epoch 501, indicating that the model has reached its maximum capacity to learn from the given data.

Learning rate V Accuracy, no bias node.

| Learning rate | Accuracy |
|---|---|
| 0.2 | 0.8153846153846154 |
| 1 | 0.8923076923076924 |
| 1.8 | 0.8923076923076924 |
| 2.6 | 0.9230769230769231 |
| 3.4 | 0.9384615384615385 |
| 4.2 | 0.9230769230769231 |
| 5 | 0.8461538461538461 |
| 5.8 | 0.8461538461538461 |
| 6.6 | 0.8461538461538461 |
| 7.4 | 0.8307692307692308 |
| 8.2 | 0.8461538461538461 |
| 9 | 0.6461538461538462 |




As the learning rate increases from 0.2 to 3.4, the accuracy increases gradually. However, as the learning rate continues to increase from 3.4 to 5, there is a sudden spike in the accuracy, with the highest accuracy achieved at a learning rate of 3.4 with an accuracy of 0.9384615384615385.

Beyond a learning rate of 5, the accuracy drops sharply, and the model performs poorly. This drop in accuracy suggests that the model is not learning effectively and may be overshooting the minimum in the cost function.

This pattern can be attributed to the impact of the learning rate on the optimization process of the model. A learning rate that is too low can cause the model to learn slowly, while a learning rate that is too high can cause the model to overshoot the minimum and not converge. The optimal learning rate lies in the range where the model can learn quickly but not overshoot the minimum.

# *Part 2*

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

The choice of terminal set is dependent on the problem being solved and the representation of the individuals in the population. In the context of this task, the terminal set consists of a range of values between -99.99 and 99.99. This range provides a large search space for the genetic algorithm to explore and discover new solutions.

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

The function set used in this task consists of four arithmetic functions: Add, Multiply, Divide, and Subtract, and a terminal set with a range of values between -99.99 and 99.99. The use of arithmetic functions allows for the manipulation of numerical values and the creation of new solutions.

The arithmetic functions used in this function set provide a wide range of operations that can be used to construct new individuals. Add and subtract can be used to adjust the values of two terminals, while Multiply and Divide can be used to create new values by combining two or more terminals.

*(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 evaluates the deviation between an individual's output and the expected output for a given set of input values. This deviation is determined by summing the absolute differences between the expected and actual output values. We add one to the error so that better solutions get higher fitness scores.

fitness score = 1 / (1 + error)

where

error = sum(absolute value(expected output - actual output) for input in inputs)

expected output = input * input

actual output = evaluate individual(individual, input)

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

A maximum initial depth of 10 was chosen to allow for a wide range of initial program complexity. A population size of 1000 was selected to ensure sufficient exploration of the search space. A maximum crossover depth of 8 was chosen to prevent overly complex individuals.

My stopping criteria is when the fitness value of the fittest program in the population drops below 0.01. This means that the program has reached a level of accuracy that is indistinguishable from the actual solution.

(e) *(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).*

Generation: 103 Fitness: 3.7500000000124487E-5

Best solution: ((x * x) + (97.0 / 97.0)) + ((((x * x) - (x + x)) * x) * x)

Generation: 42 Fitness: 3.7500000000124487E-5

Best solution: (56.0 / 56.0) + (((x * x) - x) * ((x * x) - x))

Generation: 84 Fitness: 3.7500000000124487E-5

Best solution: ((x * (x * ((x * x) - (x + x)))) + (x * x)) + (8.0 / 8.0)

Even if the fitness values are the same, the solutions obtained through genetic programming algorithm may differ. This is because the algorithm is stochastic, which means that it can generate different results even with the same settings when run multiple times. The initial population of programs is generated randomly, and the algorithm uses mutation and crossover operations to search for better solutions.