Harrison Bacordo

Part 1

DISCLAIMER: I created my own neural network library for this, as Yi allowed me to do so. Because of this, I may get slightly different results in comparison to the results of other people.

1. **Determine and report the network architecture, including the number of input nodes, the number of output nodes, the number of hidden nodes (assume only one hidden layer is used here). Describe the rationale of your choice.**

   The neural network architecture used for this dataset was (4, 5, 3), 4 input nodes, 1 hidden layer consisting of 5 nodes, and 3 output nodes. I used 4 input nodes since there are only 4 discrete features for each entry in the dataset, therefore we don't need to worry about prioritization of features (like when training a neural network on a game like Flappy Bird), and we also don't need to worry about convoluting the dataset like we would for an image input. The output layer contains 3 nodes since this is a classification ternary classification, and therefore only 3 possible outcomes exist. Any more output nodes and there's the possibility of an unassigned output node. Any less output nodes and you lose the possibility of classifying one of the irises. I chose the number of hidden nodes based on online research, which consistently provided very similar answers: either to choose a value between the number of input nodes and number of output nodes, and also to veer toward adding more hidden nodes. The reason given for using more hidden nodes during model building is that the additional nodes will likely do no harm, whereas too few nodes may prevent convergence entirely. I experienced this when training my neural network and seeing that smaller hidden layers sometimes resulted in the weights not converging and hovering around a 0.2-0.4 cost. However, upon increasing the number of hidden nodes I found that the weights consistently converged.

2. **Determine the learning parameters, including the learning rate, momentum, initial weight ranges, and any other parameters you used. Describe the rationale of your choice.**

   <u>Learning rate:</u> 0.05; I initially chose a learning rate of 0.2 as per the lecture slides but eventually found that it would at times jump the local minimum entirely. I found that 0.05 helped keep the cost within a minimum more consistently.

   <u>Momentum:</u> 0.8; I initially chose a lower momentum since I researched that a high learning rate should be paired with a lower momentum. Once I found that this didn't work in my favour, I started twisting the dials of both the learning rate and the momentum until I found a combination that gave consistently good scores

   <u>Initial weight range:</u> -1 to 1; I was deciding between 0 to 1 and -1 to 1. I decided on -1 to 1 since it gives the network more persuasion to compute a negative intermediate output, which helps gauge a better starting point for weights that will eventually converge on a negative value.

3. **Determine your network training termination criteria. Describe the rationale of your decision.**

   My network training termination criteria had two parts: either if it reached at least 98% or if the it reached the end of the modified dataset. I chose 98 percent because I thought that was a relatively decent accuracy given the dataset. Since I created my own neural network library, I decided to clone and shuffle the iris dataset 7 times, and then adding that to the original dataset, giving my resulting dataset 19200 rows, or x*2^n rows, where x is the number of rows in the original dataset (150), and n is the number of times I cloned and shuffled it (in this case, 7). Since I thought 19200 rows was enough to train the network on (especially since it I all completely shuffled), I decided to use that as a stopping criteria as well.

4. **Report your results (average results of 10 independent experiment runs with different random seeds) on both the training set and the test set. Analyse your results and make your conclusions.**

   <u>Learning rate</u> = 0.05
   <u>Momentum</u> = 0.8

Training Cycle	Cost	Correct/Total
1	0.0742	141/150
2	0.0728	146/150
3	0.0774	145/150
4	0.0747	146/150
5	0.0801	142/150
6	0.0838	142/150
7	0.0603	147/150
8	0.0654	145/150
9	0.0637	144/150
10	0.0851	140/150

   I believe the results are quite consistent around an average cost of 0.07. Given that this neural network is self-created, I am satisfied with the results. I believe if I had iterated through the data more it would converge even further on its local minimum, but decided that a slightly lower cost wouldn't be worth the time it would take to train the network 10 times with even more iterations. It looks as if training cycle 10 may have jumped a minimum towards the end, since it has a considerably worse cost and correct/total value than the remaining 9.

5. **(optional/bonus) Compare the performance of this method (neural networks) and the nearest neighbour methods.**

Part 2

1. **Determine a good terminal set for this task**

   I believe a good terminal set is the input x, as well as a random number constant. I decided on this through following the DEAP documentation and understanding that the task at hand wasn't an overly-complicated one, since there's only one input and one output for this function. If there were multiple inputs/outputs then I believe it would be more reasonable to use more than just one random constant terminal.

2. **Determine a good function set for this task**

   I believe a good function set would be the 4 basic operators (+, -, *, /), as well as square and negation. I chose these operators based on some trial and error using trigonometric functions like sin and cosine, and came to the conclusion that certain operators didn't add much value to the end result.

3. **Construct a good fitness function and describe it using plain language (and mathematical formula, or other formats you think appropriate).**

   For this part I decided to go with the mean squared error. I chose this because it always yields a positive value, so the sum can never be zero, as well as the fact that squaring always emphasizes larger differences. This can be both a good and bad feature, since the effect that outliers have may be overexaggerated. Mean squared error simply takes the difference between the expect output and the actual output and squares the resulting number. It does this to each individual on all data points, sums the squared differences, and divides it by the number of points to get the average. Pictured below is the mathematical representation of the mean squared error function

$$\mathrm{MSE} = \frac{1}{n} \sum_{i=1}^{n} (y_i - \tilde{y}_i)^2$$

4. **Describe the relevant parameter values and the stopping criteria you used.**

   Fitness type: Minimal fitness. Since we are trying to reduce the distance from the individual's guess to the actual answer, it makes sense to have a smaller fitness be better than a bigger fitness.

   Program generation type: Half-and-half; I chose this to make the trees more diverse and test a full tree set against a grow tree set

   Selection size: 3 (Chooses the 3 best individuals in a given generation and does crossover/mutation if applicable).

Stopping criteria: The only stopping criteria in place is the number of generations that must be completed, which is 100. This would give the program 100 attempts at mating the best individuals and getting a better fitness, which should be enough for the task that this part brings.

5. **List three different best programs evolved by GP and the fitness value of them (you need to run your GP system several times with different random seeds and report the best programs of the runs).**

Program	Fitness Value
square(add(mul(x, add(-2, x)), x))	1.0000037500070311
square(sub(mul(x, add(protected_div(1, add(x, add(add(add(add(1, 1), add(x, x)), -6), mul(x, mul(x, x))))), x)), add(protected_div(1, add(add(mul(mul(mul(x, mul(x, x)), mul(add(x, x), x)), mul(mul(add(x, x), mul(x, x)), add(add(1, x), x))), x), add(x, add(add(x, x), x)))), x)))	0.027896189434054787
sub(square(neg(add(x, x))), add(add(protected_div(square(add(x, neg(square(x)))), sub(-4, protected_div(square(x), sub(-4, x)))), add(protected_div(protected_div(add(square(add(add(x, x), x)), add(x, x)), sub(-4, add(add(square(x), x), x))), add(add(x, add(x, x)), sub(add(x, x), sub(square(add(x, x)), x)))), x)), add(sub(x, add(protected_div(protected_div(x, add(x, x)), add(add(sub(add(x, x), x), add(square(x), x)), protected_div(x, sub(-4, sub(x, x))))), x)), add(sub(x, x), add(sub(x, x), add(sub(add(x, x), x), add(square(x), x)))))))	0.9938039942246462

6. **(optional, bonus) Analyse one of the best programs to reveal why it can solve the problem in the task.**

Harrison Bacordo

Part 3

1. **Determine a good terminal set for this task**

   I believe a good terminal set is the 9 feature inputs/object attributes, as well as two random number constants between 0 and 100. I determine this set to be a good terminal set on the basis of online research, including the lecture slides, stating that a terminal set similar to then I used to be suitable for classification problems in genetic programming.

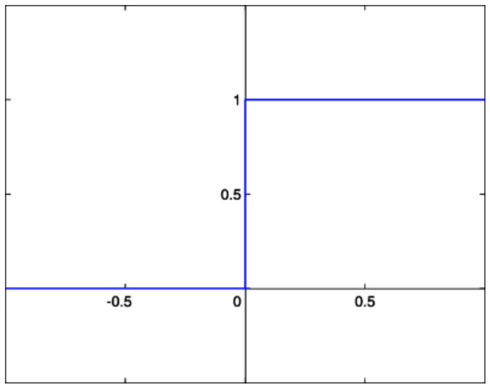
2. **Determine a good function set for this task**

   I believe a good function set would be the 4 basic operators (+, -, *, /), a power function, and trigonometric functions such as sin and cos. I determine this set to be a good function set on the basis of online research, including the lecture slides, stating that a terminal set similar to then I used to be suitable for classification problems in genetic programming.

3. **Construct a good fitness function and describe it using plain language (and mathematical formula, or other formats you think appropriate).**

   I used a simple binary step function as a fitness function for this portion of the assignment. A binary function works by naming a threshold and considering an individual's output to be 'true' if its output is greater than or equal to 0. Otherwise, its output is considered 'false'. It then checks if its output matches the expected output, and if it does it adds one to a "number correct" variable and continues to the next data point. The higher the "number correct" variable is, the better its fitness value. I believe this is a good fitness function for this dataset on the basis of it being a binary classification problem. Pictured below is the mathematical expression for binary step, as well as a graph representation of the function.

$$f(x) = \begin{cases} 0 & \text{for} \quad x < 0 \\ 1 & \text{for} \quad x \geq 0 \end{cases}$$



4. **Describe the relevant parameter values and the stopping criteria you used.**

   Fitness type: Maximal fitness. Since we are measuring the fitness based on how many correct guesses an individual makes, and the more correct guesses the better, it is logical to use a maximal fitness for this problem

Program generation type: Half-and-half; I chose this to make the trees more diverse and test a full tree set against a grow tree set

Selection size: 3 (Chooses the 3 best individuals in a given generation and does crossover/mutation if applicable).

Stopping criteria: The only stopping criteria in place is the number of generations that must be completed, which is 100. This would give the program 100 attempts at mating the best individuals and getting a better fitness, which should be enough for the task that this part brings.

5. **Describe your main considerations in splitting the original data set into a training set training.txt and a test set test.txt**

My main considerations of when splitting the dataset into training and test files is to ensure that there's a balance; there must be enough datapoints in the training set for the population to be assessed reliably, but also enough datapoints in the test set to reliably determine its accuracy on data its never seen before. Like studying for an exam, the bulk of the time is spent studying for the exam instead of actually taking it. I followed this pattern and ensured that the greater majority of the data points were used to help the individuals in the population "study", while also keeping the test set big enough to get a reliable sense of the top individual's accuracy.

6. **Report the classification accuracy (average accuracy over 10 independent experiment runs with different random seeds) on both the training set and the test set.**

Training Accuracy: 96.6%
Test accuracy: 98.3%

7. **List three best programs evolved by GP and the fitness value of them.**

Program	Fitness Training	Fitness Test
lt(sub(if_then_else(lt(abs(IN8), mul(IN5, 19.730645856768437)), protected_div(74.94461333936795, square(IN2)), square(protected_div(protected_div(74.94461333936795, square(IN2)), cos(protected_div(square(IN2), square(protected_div(IN1, IN1))))))), square(IN5)), abs(IN1))	539	137
le(49.18830959005687, mul(add(add(mul(IN6, mul(IN5, IN8)), add(3.2721219969275794, add(3.110002737028228, IN4))), add(sub(add(sub(IN4, 3.110002737028228), cos(mul(IN5, IN4))), add(IN8, IN2)), mul(IN5, IN8))), IN2))	546	138
lt(7.491860549851714, if_then_else(le(add(if_then_else(le(IN5, 3.7450587999683815), IN4, if_then_else(le(mul(IN2, IN5), 10.969309993941268), IN4, mul(IN1, IN5))), protected_div(if_then_else(le(add(IN6,	544	136

protected_div(IN8, IN6)), IN2), if_then_else(le(IN0, 10.969309993941268), IN7, IN5), mul(IN6, IN6)), IN5)), IN5), IN0, mul(mul(IN8, IN2), IN5)))		

8. **(optional, bonus) Analyse one of best programs to reveal why it can solve the problem in the task.**