Assignment 2 : Neural and Evolutionary Learning Report

Ritesh Patel

300362131

Part: 1

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.*

Input Nodes = 4. There are 4 numeric attributes in the iris data set (sepal length, sepal width, petal length, petal width) which is equal to the amount of input nodes.

Output Nodes = 3. There are 3 classes (Iris Setosa, Iris Versicolour, Iris Virginica) which is equal to the amount of output nodes.

Hidden Nodes = 5. I have checked the accuracy and number of epochs required to train the network by using different amounts of nodes in the hidden layer. After testing, I found that increasing the number of hidden nodes increases the accuracy. I also found that the number of epochs increased as I increased the number of hidden layer nodes. The increase in the number of epochs is the reason why I reduced the number of hidden nodes to 5. The value is also smaller than 2x the number of input nodes to minimize overfitting.

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.*

Learning Rate = 0.1. Since the learning rate (LR) scales the change of weights, the LR value of 0.1 changes the weights in the network by 10% of the estimated weight error every time the weights are updated.

Momentum = 0.1. Similarly, to the LR, the momentum value scales the past weight change. I have selected 0.1 as it is enough to change the current weights with influence from the past change.

Random Range = 0.5. I set the random range to 0.5, this means that the initial weight range is between -0.5 and 0.5. The random range of 0.5 worked fine for my tests as the results were ok.

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

Percent = 101.0. I wanted to use the critical error as the network training termination criteria, I changed the classification accuracy to 101% so the classification accuracy could not be used for termination. When the classification accuracy was set to 100%, the training algorithm would usually achieve the classification accuracy within 150 epochs. This usually resulted in a mean squared error(mse) of around 0.050. When the training was stopped at 100% classification accuracy the number of incorrect classifications was on average 14/75.

Critical Error = 0.01. I used the critical error as the network training termination criteria. By setting the critical error to 0.01 the network was trained until a critical error of 0.01 was achieved. This resulted in an average of 2/75 incorrect classifications. This is much better result than the 100% classification accuracy termination which resulted in 14/75 incorrect classifications.

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.*

i	Training Results	Test Mean Squared Error	Correct Classifications
1	Epoch = 264 | MSE = 0.010 |74/75	0.020	73/75
2	Epoch = 268 | MSE = 0.010 |74/75	0.019	73/75
3	Epoch = 308 | MSE = 0.010 |74/75	0.017	73/75
4	Epoch = 285 | MSE = 0.010 |74/75	0.019	73/75
5	Epoch = 262 | MSE = 0.010 |74/75	0.019	73/75
6	Epoch = 262 | MSE = 0.010 |74/75	0.020	73/75
7	Epoch = 305 | MSE = 0.010 |74/75	0.019	73/75
8	Epoch = 264 | MSE = 0.010 |74/75	0.019	73/75
9	Epoch = 280 | MSE = 0.010 |74/75	0.018	73/75
10	Epoch = 267 | MSE = 0.010 |74/75	0.019	73/75
AVG	276.5	0.0189	73/75 (97.3%)

From the results I can see that the test data averaged 97.3% accuracy with an average MSE of 0.189. The averaged accuracy was caused by all tests averaging 73/75 correct classifications. The same number of correct classifications indicate that there is a potential outlier in the test set since there are always 2 incorrect classifications, I noticed that the same two classifications failed every time in the dataset. I also noticed that using the critical error as the termination criteria increased the number of epochs than that of the percentage termination criteria. This led me to believe that the percentage termination criteria was causing under-fitting since there was a significantly lower number of epochs.

5. *Compare the performance of this method (neural networks) and the nearest neighbor methods.*

KNN (10 Runs AVG) = 72/75(For All Tests) => 96.00%

Neural Network (AVG) = 97.3%

97.3% - 96% = 1.3%

From my tests I found that the accuracy of the neural network is 1.3% higher than my KNN algorithm. From an accuracy point of view the neural network performs better than KNN algorithm. The KNN algorithm can be computationally intensive depending on the training data set. This is due to the fact that the KNN algorithm stores the entire training dataset for prediction making the algorithm slow and costly. This is not the case for neural networks as it does not require the entire training data set to be stored for predictions. The neural network does not require the training set once the neural network is trained which can be a time-intensive process. Neural networks outperform the nearest neighbor methods when there is a large amount of data or the structure is complex.

Part: 2

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

A good terminal set for this task would be the 'X' values which is the input values. The constants used was the range 0.0 to 10.0.

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

For this task, I used Add, Subtract, Divide and Multiply. This resulted in the most simplified function along with best fit to the test data. This resulted in reasonable accuracy. Other functions were not used as we are trying to get the relation between X & Y and would not be as simple. The selected functions were enough for this task.

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

Screenshot of my fitness function.

```
for (int i = 0; i < input.size(); i++) {
    xVar.set(input.get(i));

    try {
        double value = program.execute_double(0, NO_ARGS)
        error += Math.abs(value - output.get(i));
        if (Double.isInfinite(error)) {
            return Double.MAX_VALUE;
        }
    } catch (ArithmeticException e) {
        throw e;
    }
}

if (error < 0.001D) {
    error = 0.0D;
}
return error;
```

How it works:

For each X value in the dataset,

> Calculate the value of the output which results in using 'X' as the input into the function described by the current chromosome(chrom).
>
> Add the absolute (abs) value of the error of the output. This is done by subtracting the ideal output from the value returned by the function, to the total error.

The return value represents the total distance from the ideal values of that current chrom.

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

~Screenshot of Parameters

```
config.setGPFitnessEvaluator(new DeltaGPFitnessEvaluator());
config.setPopulationSize(500);
config.setFitnessFunction(new FitnessFunction(xx_Vals, yy_Vals, x_Var));
```

Parameter Values:

a. **MaxInitDepth:** I used the default value by JGAP = 7.

b. **GPEFitnessEvaluator:** DeltaGPFitnessEvaluater (DGPFE)– I chose the DGPFE since the DefaultGPFitnessEvaluator uses large number to represent better fit. This would make it incompatible with the fitness function.

c. **PopulationSize:** After testing, I set the population size to 500, since populations under 500 would find a function.

d. **Crossover:** Unchanged => Stock rate of 35%, This means the population size multiplied by 35% crossover operations per generation.

e. **Mutation:** Unchanged => Stock rate of 12%, In this case the probability of a mutation occurring is 1/12 (8.3%). The probability can then be applied to each gene of each element in the population that was not produced by the crossover. This results in a probability of ((1/12) *(populationSize * chromSize))

Termination criteria:

The termination criteria is to receive a distance (fitness function) of zero for a chromosome within 200 generations. Termination occurs when a successful chromosome within 200 generations is found. The chromosome is returned with the smallest distance.

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).

	FITNESS VALUE	FUNCTION
1	0.0	(((X * X) - X) * ((X * X) - X)) + (4.0 / 4.0)
2	0.0	(((X * X) - X) * ((X * X) - X)) + (9.0 / 9.0)
3	0.0	((4.0 / 4.0) + ((X * X) * (X * (X - 2.0)))) + (X * X)

Part: 3

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

A good terminal set for this task would need to include 9 different attributes. These attributes consist of the following: Clump Thickness (CT), Uniformity of Cell Size (USz), Uniformity of Cell Shape (UShp), Marginal Adhesion (MA), Single Epithelial Cell Size (SESz), Bare Nuclei (BN), Bland Chromatin (BC), Normal Nucleoli (NN), Mitoses (M). Along with all 9 'inputs' the constants range used was between 1.00 and 10.00.

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

Initial function set = Add, Subtract, Multiply, Divide, Power, Exponent, Cosine, Tangent.

After testing the initial function set thoroughly, I found that the last chromosome only included the Add, Subtract, Multiply and DIVIDE. This led me to remove the Power, Exponent, Cosine and Tangent. In doing so, there was no loss in accuracy.

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

Screenshot of my fitness function.

```
protected double evaluate(IGPProgram program)
{
  double truePositive = 0.0D;double trueNegative = 0.0D;double falsePositive = 0.0D;double falseNegative = 0.0D;
  for (int i = 0; i < instances.size(); i++)
  {
    clumpThickness.set(((Instance)instances.get(i)).getAttributes().get(0));
    uniformityOfCellSize.set(((Instance)instances.get(i)).getAttributes().get(1));
    uniformityOfCellShape.set(((Instance)instances.get(i)).getAttributes().get(2));
    marginalAdhesion.set(((Instance)instances.get(i)).getAttributes().get(3));
    singleEpithelialCellSize.set(((Instance)instances.get(i)).getAttributes().get(4));
    bareNuclei.set(((Instance)instances.get(i)).getAttributes().get(5));
    blandChromatin.set(((Instance)instances.get(i)).getAttributes().get(6));
    normalNucleoli.set(((Instance)instances.get(i)).getAttributes().get(7));
    mitoses.set(((Instance)instances.get(i)).getAttributes().get(8));
    try
    {
      int classLabel = 0;
      double result = program.execute_double(0, NO_ARGS);

      if (result > 0.0D) {
        classLabel = 4;
      } else {
        classLabel = 2;
      }
      if (classLabel == ((Instance)instances.get(i)).getClassLabel()) {
        if (result > 0.0D) {
          truePositive += 1.0D;
        } else {
          trueNegative += 1.0D;
        }
      }
      else if (result > 0.0D) {
        falsePositive += 1.0D;
      } else {
        falseNegative += 1.0D;
      }
    }
    catch (ArithmeticException e)
    {
      e.printStackTrace();
    }
  }
  double measure = (truePositive / (truePositive + falseNegative) + trueNegative / (trueNegative + falsePositive)) / 2.0D;

  return measure * 100.0D;
`
```

How it works:

The function takes the current chromosome, sets the variables to the data provided from each instance in a lost. It then gets the result from the function using program.execute_double. If this result is higher than 0, it will classify the instance as positive, else it classifies as negative. Once classified it checks the instance to see if this is correct. It will then increment the applicable variable: truePositive, falsePositive, true Negative, FalseNegative. Once all instances are checked, the correct classifications are divided by (themselves + the false classifications) and finally gets divided by 2 since there are 2 classes. This gives us a weighted accuracy.

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

Parameter Values:

a. **MaxInitDepth:** I used the default value by JGAP = 7.

b. **GPEFitnessEvaluator:** DeltaGPFitnessEvaluater (DGPFE)– I chose the DGPFE since the DefaultGPFitnessEvaluator uses large number to represent better fit. This would make it incompatible with the fitness function.

c. **PopulationSize:** After testing, I set the population size to 500, since populations under 500 would find a function.

d. **Generations:** I left the generations value to 200 from the previous part. This could be set to 100 since the best chromosome is normally found in under 100 generations.

e. **Crossover:** Unchanged => Stock rate of 35%, This means the population size multiplied by 35% crossover operations per generation.

f. **Mutation:** Unchanged => Stock rate of 12%, In this case the probability of a mutation occurring is 1/12 (8.3%). The probability can then be applied to each gene of each element in the population that was not produced by the crossover. This results in a probability of ((1/12) *(populationSize * chromSize))

Termination criteria:

The termination criteria I used for this part was that the program needs to find a function that is more than 99% accurate on the training set. The other termination criteria is the number of generations which is 200 (stop at 200).

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

Since the data is divisible by 3, I decided to divide the data into 3 sets. I used 2/3rd of the data for the training set and 1/3rd for test set. I could have divided the data into 4 sets and used ¾ for training but that would have been a risk to the 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.*

i	Best Solution	Training Set Accuracy	Test Set Accuracy
1	ucsi + (((((ucsh + ((ct - (6.0 - bn)) * nn)) - 5.0) * nn) * ucsi) - 4.0)	97.49%	98.14%
2	(ucsi + (((ucsi + ((ucsh * ((ct - 5.0) + bn)) - 5.0)) - 5.0) * nn)) - 6.0	97.67%	97.22%
3	(bn - 5.0) + (((nn - 5.0) + ((ct - 6.0) + ucsh)) + ucsi)	97.13%	98.52%
4	ucsh + ((ucsi - 6.0) + (((bn * ucsh) - 3.0) + ((ucsh * m) - 6.0)))	97.04%	97.96%
5	((((ct - 6.0) + (ct - 4.0)) + ucsi) + ((bn + (ct - 6.0)) - 6.0)) + bn	97.13%	97.03%
6	(bc * (bn + (bc * (bn + (((ucsh - 5.0) - 5.0) + ct))))) - 5.0	97.22%	98.89%
7	(ucsh * ((ucsh * ((ucsi + (((ct + bn) - 6.0) * nn)) - 4.0)) - 6.0)) - 6.0	97.67%	98.15%
8	((ct - 5.0) * ucsh) + ((nn + ((bn * ucsh) - 6.0)) - 5.0)	97.40%	97.22%
9	(ct * ucsh) - ((5.0 * 5.0) - ((bn * nn) + bn))	97.13%	98.79%
10	(ct - 5.0) + ((((ucsh * nn) - 5.0) + bn) + (((secs + ucsh) - 6.0) - 5.0))	97.58%	98.79%
	AVG	97.35%	98.07%

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

i	Best Solution	Training Set Accuracy	Test Set Accuracy
1	(bc * (bn + (bc * (bn + (((ucsh - 5.0) - 5.0) + ct))))) - 5.0	97.22%	98.89%
2	(ct * ucsh) - ((5.0 * 5.0) - ((bn * nn) + bn))	97.13%	98.79%
3	(ct - 5.0) + ((((ucsh * nn) - 5.0) + bn) + (((secs + ucsh) - 6.0) - 5.0))	97.58%	98.79%