

Comp307 Assignment 2

Daniel Walker - #300336857

Part 1: Neural Networks for Classification [30 marks]

1.

Learning rate: 0.2

Momentum: 0

Critical error: 0.002

Random range: 0.5

Classification accuracy: 101.0

input nodes: 4

the number of output nodes: 3

the number of hidden nodes: 3

Originally assumed only 1 hidden layer for this assignment, meaning 3 total layers; one input, one hidden and one output. Adding too many hidden layers would just make things unnecessarily complicated with a large number of epochs.

Network architecture:

I decided to use 4 input nodes as there are four classifications for the iris: sepal length in cm, sepal width in cm, petal length in cm, and petal width in cm. Using all of the attributes will increase the accuracy of our classification output. I then decided on 3 output nodes, one for each type of iris: Setosa, Versicolour and Virginica. For hidden nodes, with only 1 node we maxed out at 1,000,000 epochs, with 2 nodes we get much better, reliable and consistent results around 1100 epochs. Any more hidden nodes and we start to see overfitting.

2.

Learning parameters:

As told in the lectures, a good learning rate to start with was 0.2. I noticed if the LR was too high the weights will diverge and never find the ideal values. Any lower than 0.2 and I notice the algorithm becomes much slower. Thus 0.2 works well for this assignment. Changing momentum didn't increase the accuracy or decrease epochs so I left it as 0. The random range didn't have much effect either, Setting the random range to 0.5 allows the weights start as a random value between -0.5 and 0.5. I left it at 0.5 as the weights change very quickly anyway.

3. Network training termination criteria:

Stopping accuracy: 101% Critical error 0.002

Setting stopping accuracy at 100% would stop the program quicker than I thought. I really wanted the critical error to be the cause of the stoppage as this would allow for more accurate results to generate. At 100%, the test data had many more errors, with a MSE of about 0.08. At 101%, we terminate at the critical error of 0.002 giving us an average of 2/75 mis-classifications.

4. Report of my results

Trial	Training MSE	Incorrect (Training)	Test MSE	Incorrect (Testing)
1	0.001	0/75	0.018	2/75
2	0.006	1/75	0.014	2/75
3	0.002	0/75	0.016	2/75
4	0	0/75	0.020	3/75

Trial	Training MSE	Incorrect (Training)	Test MSE	Incorrect (Testing)
5	0.006	1/75	0.014	2/75
6	0.008	2/75	0.011	1/75
7	0	0/75	0.021	3/75
8	0.001	0/75	0.025	3/75
9	0.017	2/75	0.011	1/75
10	0	0/75	0.028	4/75
Avg	0.0041	0.6/75	0.0178	2.3/75

Features to note:

- Best training runs equaled worst testing runs i.e 0/75 for train, 4/75 for test etc
This may be due to overfitting data
- Our training MSE average was 0.004, with 0.6 mis-classifications. The unseen (test) data averaged 2/75 mis-classifications giving over 97% accuracy.

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

	KNN	Neural Network
Success Rate	92.0% (69/75)	97% (72.7/75)

KNN was consistent with its success rate whereas Neural network varied, so I was sure to take the average of 10 runs for comparison with KNN.

NN outperforms KNN by 5%, which can be a very significant amount depending on the data. I would definitely want to use neural networks on any other sets of data in the future, as I know it will be more accurate than KNN when it comes to the more important, testing data.

Neural networks has a much more efficient processing time than k-nearest neighbour. Considering KNN requires n distance comparisons $O(n^2)$. If we had 1000 instances, KNN will need to perform 1000×1000 distance calculations for every instance, whereas Neural Networks only needs to provide the inputs of each instance, traverse the NN and retrieve the outputted classification.

Part 2: Genetic Programming for Symbolic Regression

Make sure ecs computer has JGAP library 3.6.2 found using IntelliJ Maven Repository

Some code for my implementation of part 2 may have been influenced using this resource I found online:

<http://www.hakank.org/jgap/SymbolicRegression.java>

1. Determine a good terminal set for this task:
 - Variable "X"
 - Terminal (an integer between 2 and 10)

2. Determine a good function set for this task.

- Multiply
- Divide
- Subtract
- Add
- Power

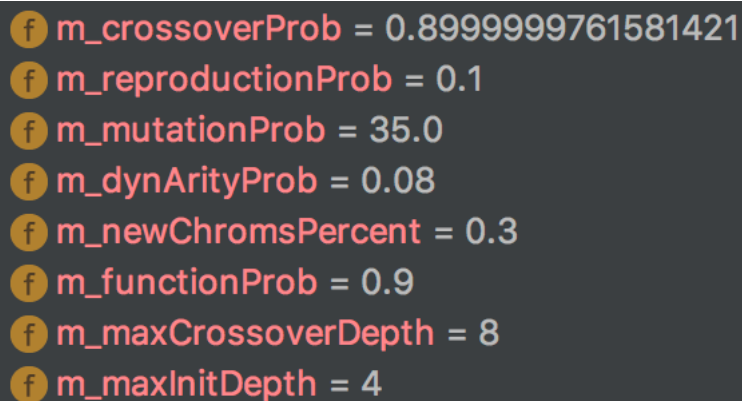
Determined within the MathProblem JGAP example

3. Construct a good fitness function and describe it using plain language:

To evaluate the performance of a given program I created a fitness function.

The function can evaluate the performance by integrating over the length of inputs, in my case 20 inputs and for every input, set var x to be an $x[i]$ value. This allows us to use a value that we know when evaluating the performance. We run the program and it will show us what the determination of y is. We then compare this to the true value of $y[i]$. Getting an estimation from these differences is done by taking a sum of the total error over all 20 inputs. The lower the total error, the better performance we will see from the program.

4. Describe the relevant parameter values and the stopping criteria:

A screenshot of a text editor showing JGAP parameter settings. Each line starts with a yellow circle containing a lowercase 'f' followed by a parameter name and its value. The parameters are: m_crossoverProb = 0.8999999761581421, m_reproductionProb = 0.1, m_mutationProb = 35.0, m_dynArityProb = 0.08, m_newChromsPercent = 0.3, m_functionProb = 0.9, m_maxCrossoverDepth = 8, and m_maxInitDepth = 4.

```
f m_crossoverProb = 0.8999999761581421
f m_reproductionProb = 0.1
f m_mutationProb = 35.0
f m_dynArityProb = 0.08
f m_newChromsPercent = 0.3
f m_functionProb = 0.9
f m_maxCrossoverDepth = 8
f m_maxInitDepth = 4
```

Max initial depth = 4
Max crossover depth = 8
Crossover prob = 0.9
Mutation prob = 35.0
Reproduction prob = 0.2
Population size: 1000
Strict program creation = true

I found **max initial depth** to usually be set to 4, this worked well here so I let it at 4

Following this resource: <http://www.hakank.org/jgap/SymbolicRegression.java>

I used the same **crossover depth of 8**. Any more than 8 and I noticed a larger effect of the bias on the results.

A **crossover probability** of 0.9 is recommended for most applications. 0.9 allowed for a "good mate" which is enough to generate new formulas from two candidates.

Strict program creation was required to ensure evolving does not encounter a problem. We also did not want the program to just try new random numbers if the solution doesn't contain a terminal value.

Mutation Probability

Reproduction Probability of 0.2 passes on some solutions to the candidate pool ensuring they continue onto the next generation

Population size 1000. This has generated sufficient results for me without being too big of a population.

Stopping Criteria:

Max evolutions = 1000 Minimum Acceptable Error = 0.001 (0.1%)

The program is at its most developed once it provides an accurate transformation of x to y (within 0.01% across all 20 inputs) using the best program.

Stopping once we have reached the min acceptable error was much more efficient than stopping at max evolutions as I noted many runs of the program completed with far less evolutions than the maximum. The max evolutions is more there to terminate the training so it does not infinitely try to evolve.

5. List three different best programs evolved by GP

<i>Trial</i>	<i>Evolutions</i>	<i>Fitness</i>	<i>Function</i>
1	59	0.00	$(4.0 / 4.0) + (((X * X) - X) * ((X * X) - X))$
2	137	0.00	$(X * X) + ((X * (X * (((X * X) - X) - X))) + (3.0 - 2.0))$
3	60	0.00	$((X * X) + (4.0 / 4.0)) + (((X * X) - X) - X) * (X * X)$

6. Bonus:

Examining the best function would be trail 3, with 59 evolutions, the function expands out to:

$$\underline{\underline{y = x^4 - 2x^3 + x^2 + 1}}$$