COMP 307/AIML420 — *Introduction to AI*

Assignment 2: Neural and Evolutionary Learning

12% of Final Mark — Due: 11:59pm Wednesday 27 April 2022

1 Objectives

The goal of this assignment is to help you understand the basic concepts and algorithms of neural and evolutionary learning, use these algorithms to perform classification and regression tasks, and analyse the results to draw some conclusions. In particular, you should be familiar with the following topics:

- Multilayer feed forward neural network architectures and applications,
- Calculating the output of a neural network given a set of inputs (i.e. a feedforward pass),
- Back (error) propagation algorithm and its variations,
- Evolutionary computing and learning paradigms,
- Genetic programming for solving real world applications particularly for regression problems, and
- Tackling a problem with an existing genetic programming package.

These topics are (to be) covered in lectures 08–12. The online materials can also be checked.

In this assignment, neural networks refer to the standard multilayer feed forward neural networks trained by the back propagation algorithm. Genetic programming refers to the standard genetic programming approach with the tree-like structure for the evolved programs.

2 Question Description

Part 1: Classifying Pingu with a Neural Network (50 Marks)

In this part, you are required to implement a simple neural network to perform classification on the *penguins* data set described below, and answer questions on its performance.

Problem Description

The original dataset contained 344 penguins, with species corresponding to their: species, island, bill length (mm), bill depth (mm), flipper length (mm), body mass (g), sex, and the year of observation. To simplify the task, we have processed the dataset to remove all missing values and keep only the four real-valued numerical features, plus the species as the class label.

The processed dataset consists of three species (classes) of penguins: 146 of class "Adelie", 119 of class "Gentoo", and 68 of class "Chinstrap". The features are *bill length (mm), bill depth (mm), flipper length (mm) and body mass (g)*.

Requirements

You should implement a simple neural network (as described in class) to classify the Penguins dataset into its three classes. You should then use the training set `penguins307-train.csv` to learn/train your neural network, then apply the learned/trained neural network on the test set `penguins307-test.csv`. Note that the *final* column in these files lists the class label for each instance.

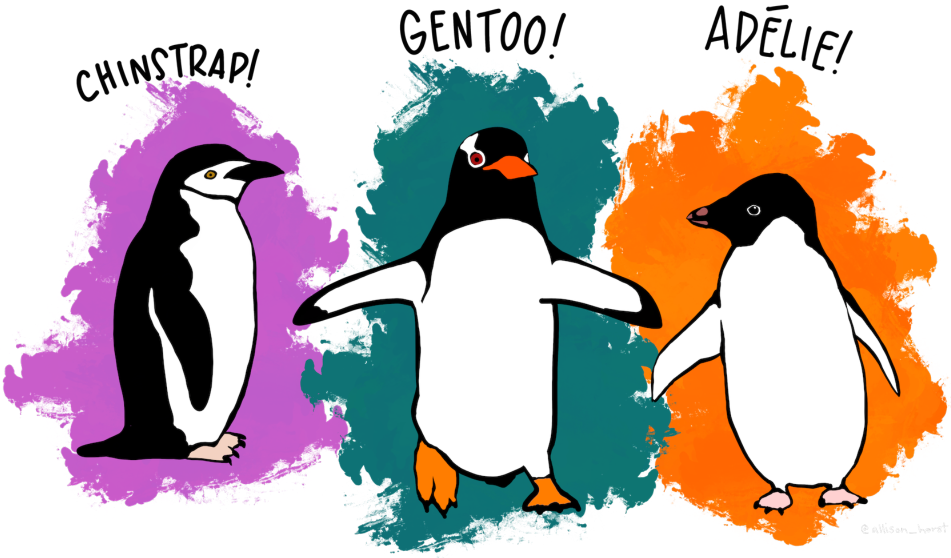

Figure 1: The Palmer Penguins Dataset. Horst AM, Hill AP, Gorman KB (2020). palmerpenguins: Palmer Archipelago (Antarctica) penguin data. https://allisonhorst.github.io/palmerpenguins/

To simplify things as much as possible, we have pre-defined the network for you to implement, as well as the parameter settings. As representing a neural network in code is not a straightforward data structure, we have also provided **skeleton code** in both Python and Java. The parts of this code that you will need to complete are marked with "TODO" statements in the `a2Part1.py` and `NeuralNetwork.py`; or `a2Part1.java` and `NeuralNetwork.java` files, respectively.

As always, you are more than welcome to use other languages — in this case, you will need to rewrite the code yourself, as we cannot support every possible language!

The network you should implement is shown below:

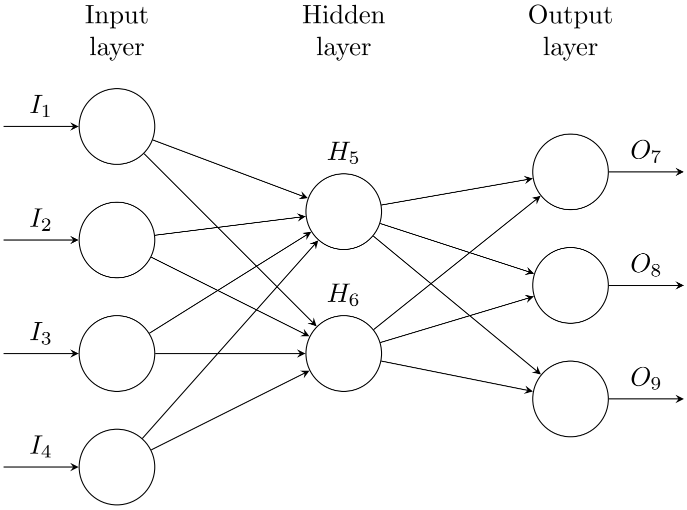

Figure 2: ANN diagram for the Penguins dataset.

And the parameter settings are:

- Input nodes: 4 (corresponding to the four features)
- Hidden layers: 1, hidden nodes: 2
- Output nodes: 3 (corresponding to the three classes)

- $\eta = 0.2$ (learning rate)
- No momentum, no bias nodes
- Activation function: Sigmoid
- Epochs to run for: 100. You should use **online** learning, i.e. update the weights after every instance.
- Initial weights should be set to:

w_{15}	w_{16}	w_{25}	w_{26}	w_{35}	w_{36}	w_{45}	w_{46}	w_{57}	w_{58}	w_{59}	w_{67}	w_{68}	w_{69}
-0.28	-0.22	0.08	0.20	-0.30	0.32	0.10	0.01	-0.29	0.03	0.21	0.08	0.13	-0.36

Writing all the code in one go will likely be stressful and make debugging very difficult. A suggested series of smaller goals with indicative marks is:

1. Implement a single feedforward pass, i.e. taking the first instance in the file and calculating the three outputs and class label (10 marks)
2. Implementing a single backpropagation update of the weights for the first instance (10 marks)
3. Extend your code to run for 100 *epochs*, reporting the performance (training accuracy) of each epoch (5 marks)
4. Use your trained neural network to classify the test set, and report the total test accuracy. (5 marks).

Your code **must** print the output of the first feed-forward pass (both the raw values, and the predicted class), as well as the new weights after the first back-propagation update. This is essential so that the tutors can give you marks! For subsequent iterations, you may prefer to print fewer updates to make the results easier to understand (e.g. overall accuracy and weights once per epoch).

You should submit the following files electronically:

- (30 marks) `Program code` based on the skeleton code, or written by yourself (the source code as well as the executable program that runs on the ECS School machines).
- a `readme.txt` describing how to run your program.
- (20 marks) `A report` in `.pdf` format. The report should:
  1. Report the output and predicted class of the first instance in the dataset using the provided weights.
  2. Report the updated weights of the network after applying a single back-propagation based on only the first instance in the dataset.
  3. Report the final weights and accuracy on the test set after 100 epochs. Analyse the test accuracy and discuss your thoughts.
  4. Discuss how your network performed compared to what you expected. Did it converge quickly? Do you think it is overfitted?

Part 2.1: Further Improving Your Network (15 marks)

Bias nodes are an important part of neural network design, as they allow the network to shift the output of the activation function to the left or the right, which is often crucial for learning.

You should now add **bias nodes** into your network from Part 2 (with the below initial weights). Train it using the same parameters as before, and report your test accuracy (**10 marks**). Compare the accuracy achieved to that of the original network, and discuss possible reasons for any performance differences (**5 marks**).

b_5	b_6	b_7	b_8	b_9
-0.02	-0.20	-0.33	0.26	0.06

Part 2.2: Sensitivity Testing (AIML420: 15 marks, COMP307: 10 *bonus* marks)

This question is *compulsory* for AIML420 students (15 marks). It is *optional* for COMP307 students (who can receive up to 10 bonus marks).

Designing a Neural Network requires (manually) setting many different parameters, such as the number of hidden layers/nodes, the learning rate, number of epochs, and whether to use online, batch, or offline learning.

Pick one or two of these parameters to perform *sensitivity testing* on. For each parameter, you should pick at least five different parameter values. For each of these values, train your network with only that parameter changed, and report the test accuracy.

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.

If testing only a single parameter, you should perform a more thorough analysis (e.g. a wider range of parameter values) and discuss the results more deeply than if you pick two parameters to test.

Part 2: Genetic Programming for Symbolic Regression (35 marks)

In this part, you are required to use genetic programming to evolve a mathematical function (which is an individual program in the population) for a simple symbolic regression task. In real world applications, a test set of data is needed in many situations to evaluate the generalisability of the model. In this assignment, to make the question simple, you are not required to have a test set — you are just required to evolve a mathematical function to reveal the relationship between the input variable(s) (attributes) and output variable from a (training) set of instances.

Problem Description

The task involves mapping a single input variable x to the (single) output variable y. In a 2D (two-dimensional) space (x-y coordinates), there are 20 points (x-y pairs). The task is to find a mathematical function to describe the relationship between the input variable x and output variable y. The 20 points are as follows. They are also saved in the file `regression.txt` in plain text format.

x	-2.0	-1.75	-1.50	-1.25	-1.00	-0.75	-0.50	-0.25	0.00	0.25
y	37.0000	24.1602	15.0625	8.9102	5.0000	2.7227	1.5625	1.0977	1.0000	1.0352

x	0.50	0.75	1.00	1.25	1.50	1.75	2.00	2.25	2.50	2.75
y	1.0625	1.0352	1.0000	1.0977	1.5625	2.7227	5.0000	8.9102	15.0625	24.1602

Requirements

It is very time-consuming to write your own GP package from scratch. As many GP packages are available, this is not necessary (or recommended!). In this assignment, we recommend two main GP packages for the GP questions. They are JGAP (Java, `https://sourceforge.net/projects/jgap/`) and DEAP (Python, `https://deap.readthedocs.io/en/master/`). If you prefer a different programming language (e.g. C(++)), you can use a different package. Please clearly describe the package you used in your report.

Your job is to use any of the genetic programming packages with necessary changes of the terminal set, function set, fitness function, parameters and termination criteria to solve the task described above.

Both JGAP and DEAP have example code and/or tutorials that you may find useful. Note that you will still need to understand and justify the GP representation, fitness function, and parameter settings your method uses in order to get a good mark!

NB: There are several libraries now available (e.g. *gplearn*) that allow GP for Symbolic Regression to be performed with only a few lines of code (similar to scikit-learn for KNN/DTs). You will NOT receive many marks for using such a library as it does not show your understanding of the GP algorithm.

You should submit the following files electronically:

- `Program code` written by yourself (source code and the executable code that runs on the ECS machines).
- a `readme.txt` describing how to run your program.
- `A report` in `.pdf` format. The report should:
  1. Justify the terminal set you used for this task.
  2. Justify the function set you used for this task.
  3. Formulate the fitness function and describe it using plain language (and mathematical formula, or other formats you think appropriate, e.g. good pseudo-code).
  4. Justify the relevant parameter values and the stopping criteria you used.
  5. 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).
  6. (optional, bonus, 5 marks) Analyse one of the best programs from above and explain how different parts of the tree "work together" to solve the task.

3 Relevant Data Files and Program Files

The relevant data files and the skeleton code are included as a `.zip` file on the course homepage: `https://ecs.wgtn.ac.nz/foswiki/pub/Courses/COMP307_2022T1/Assignments/ass2_files.zip`.

4 Notes

We will endeavour to mark your work and return it to you as soon as possible, hopefully in 2 weeks. Please be patient if COVID-19 prevents us marking as promptly as we plan. The tutors will run a number of help desks to provide guidance (but won't tell you the answers!).

5 Submission Guidelines

5.1 Submission Requirements

1. Programs for all individual parts. To avoid confusion, the programs for each part should be stored in a separate directory `part1/, part2/`. Within each directory, please provide a `readme.txt` file that specifies how to compile and run your program on the **ECS school machines**. An output file called `sampleoutput.txt` should also be provided to show the output of your program running properly. If your programs cannot run properly, provide a `buglist` file which details what does and doesn't work.
2. The report as a single PDF. You should mark each of the parts clearly.

5.2 Submission Method

The programs and the report should be submitted through the web submission system from the COMP307 or AIML420 course web site **by the due time**. Please make sure you submit to the course you are enrolled in.

5.3 Late Penalties

The assignment must be submitted on time unless you have made a prior arrangement with the **course coordinator** or have a valid medical excuse. This year, we are using the ECS extension system for all extension requests. The penalty for assignments that are handed in late without prior arrangement is one grade reduction per day. Assignments that are more than one week late will not be marked.

Remember that you have three late days for this course (which can be used fractionally), but that these apply across the **whole** course, not per-assignment! Please save some late days in case you need them later on!