COMP307/AIML420 — Fundamentals of AI

Assignment 2:
Neural Networks and Probability Basics

100 marks = 15% of Final Mark for COMP307,
150 marks = 14% of Final Mark for AIML420

Due date: 11:59 PM - May 01, 2024 (Wednesday)

01-04-2024

Objectives

The main 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:

- Perceptron and its implementation;
- 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 propagation algorithm.

In addition, we review some probability basics, which are helpful for understanding cross entropy and the Bayes networks that we will discuss later on.

These topics are covered in lectures from weeks 5–6. The textbook and 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.

IMPORTANT. You must not use external libraries (like PyTorch, sklearn, or any others) or any AI Tool (like ChatGPT, CoPilot) to complete any part of this assignment, except for 3.4.

1 Reasoning Under Uncertainty Basics

In the following, unless explicitly specified, *capital letters* (e.g., A, B, X, Y) represent a *random variables*, and *lower-case letters* (e.g., a, b, x, y) represent values (numbers).

This part contains several questions about the basics of reasoning under uncertainty. You need to write your answers to each of these questions in your report, and **show your reasoning.**

For calculations, you need to show the steps in the form like $P(A = 0|B = 1) = \frac{P(A=0,B=1)}{P(B=1)}$, to demonstrate that you *know how to calculate* them.

For proofs, you must clearly show each step.

1.1 Question 1 [15 marks]

The tables below give the prior distribution $P(X)$, and two conditional distributions $P(Y|X)$ and $P(Z|Y)$. It is also known that Z *and* X *are conditionally independent given* Y. All the three variables (X, Y, and Z) are binary variables.

X	$P(X)$
0	0.35
1	0.65

X	Y	$P(Y\|X)$
0	0	0.10
0	1	0.90
1	0	0.60
1	1	0.40

Y	Z	$P(Z\|Y)$
0	0	0.70
0	1	0.30
1	0	0.20
1	1	0.80

1. Compute the table of the joint distribution $P(X, Y, Z)$. **Show the rule(s) you used, and the steps of calculating each joint probability**.
2. Create the full joint probability table of X and Y, i.e., the table containing the following four joint probabilities $P(X = 0, Y = 0)$, $P(X = 0, Y = 1)$, $P(X = 1, Y = 0)$, $P(X = 1, Y = 1)$. **Show the rule(s) used, and the steps of calculating each joint probability**.
3. From the above joint probability table of X, Y, and Z, calculate the following probabilities. **Show your derivation**.
   (a) $P(Z = 0)$,
   (b) $P(X = 0, Z = 0)$,
   (c) $P(X = 1, Y = 0|Z = 1)$,
   (d) $P(X = 0|Y = 0, Z = 0)$.

1.2 Question 2 [10 marks]

Consider three Boolean variables A, B, and C (can take t or f). We have the following probabilities:

- $P(B = t) = 0.7$
- $P(C = t) = 0.4$
- $P(A = t|B = t) = 0.3$
- $P(A = t|C = t) = 0.5$
- $P(B = t|C = t) = 0.2$

We also know that A *and* B *are conditionally independent given* C. Calculate the following probabilities. **Show your derivation.**

1. $P(B = t, C = t)$
2. $P(A = f|B = t)$
3. $P(A = t, B = t|C = t)$
4. $P(A = t|B = t, C = t)$
5. $P(A = t, B = t, C = t)$

1.3 Question 3 [AIML420 only: 10 marks]

135+10 160 Prove the following statements. **Show your derivation.**

1. If $P(A|B, C) = P(B|A, C)$, then $P(A|C) = P(B|C)$
2. If $P(A|B, C) = P(A)$, then $P(B, C|A) = P(B, C)$
3. If $P(A, B|C) = P(A|C) * P(B|C)$, then $P(A|B, C) = P(A|C)$

2 Perceptron

This part of the assignment involves writing a program that implements a perceptron that learns to distinguish between two classes of radar data for the ionosphere data set.

Data Set. The ionosphere data set is taken from the UCI Machine Learning Repository (https://archive.ics.uci.edu/ ml/datasets/ionosphere). The data set contains 351 instances in 2 classes, of which 225 are "good" (g) radar samples and the remaining 126 are "bad" (b). Each instance has 34 features, consisting of 17 complex number pairs of pulse numbers. These samples were collected by a system in Goose Bay, Labrador. The system consists of a phased array of 16 high-frequency antennas with a total transmitted power on the order of 6.4 kilowatts. The file ionosphere.data consists of the 351 instances, with a header line on the first line of the file.

Simple Perceptron Algorithm. A perceptron with n features is represented by a set of real valued weights, $\{w_0, w_1, ..., w_n\}$: one weight for the threshold (w0), and one for each feature. Given an instance with features $f_1, ..., f_n$, the perceptron will classify the instance as a positive instance (i.e., a member of the class) if:

$$\sum_{n=0}^{n} w_i f_i > 0$$

where f_0 is the "dummy" feature that is always 1 (providing the "bias").

The algorithm for learning the weights of the perceptron was given in lectures as:

```
Until the perceptron is always right (or some limit):
    Present an example (+ve or -ve)
    If perceptron is correct, do nothing
    Else if -ve example and wrong:
        (i.e. weights on active features are too high)
        Subtract feature vector from weight vector
    Else if +ve example and wrong:
        (i.e. weights on active features are too low)
        Add feature vector to weight vector
```

-ve = negative and
+ve = positive

Your program should implement this algorithm. It should present the entire sequence of training examples several times over, until either the perceptron is correct on all the training examples, or it stops converging (e.g., it has presented all the examples 100 times without any progress). It should then use the perceptron to classify all the examples (without changing the weights anymore) and print the final classification accuracy.

Requirements. The program should take one file name (the data file) as a command line argument, and then:

- load the set of instances from the data file;
- construct a perceptron that uses the features as inputs;
- train the perceptron until either it is correct on all instances, or it stops converging (at least 100 iterations);
- report on the number of training iterations to convergence, or the number of instances that are still classified wrongly; and
- print out the final set of weights the perceptron learned.

You can use any programming language, **as long as it can be easily run on the ECS systems**. For this part you have to hand in two or three components as listed below.

2.1 Program code [15 marks]

Program for your perceptron (the source code as well as the executable program that runs on the ECS School machines). Submit also a **readme.txt** describing how to run your program.

2.2 Report in .pdf format [10 marks]

The report should:

1. Report on the accuracy of your perceptron. For example, did it find a correct set of weights? Did its performance change much between different runs? ("run" means training and testing from scratch, with a different random initialization of the weights.)

2. Explain why evaluating the perceptron's performance on the training data is not a good measure of its effectiveness. You should split the dataset, retrain, and perform a fairer evaluation.

2.3 Analysis [AIML420 only: 10 marks]

Use a perceptron to solve the classification problem shown in Table 1. Run the program code of perceptron to solve this problem five times and report the averaged accuracy and the standard deviation. Submit the program code and add an analysis of the result and conclusions to your report.

Table 1: Dataset (for AIML420 students)

Instance	F1	F2	F3	Class
1	0	0	1	0
2	0	1	0	1
3	1	0	1	1
4	1	1	0	0
5	1	1	1	0
6	1	0	0	1
7	0	1	1	1
8	0	0	0	0

3 Classifying Pinguins with a Neural Network

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). **IMPORTANT:** You don't need to worry about further preprocessing the dataset.

Requirements. You should implement a simple neural network *including backpropagation* (as described in class) to classify the Penguins dataset into its three classes. Note that implementing the network is simplest with the sigmoidal activation. You should use the training set **penguins307-train.csv** to learn/train your neural network, then apply the learned/trained neural network on the

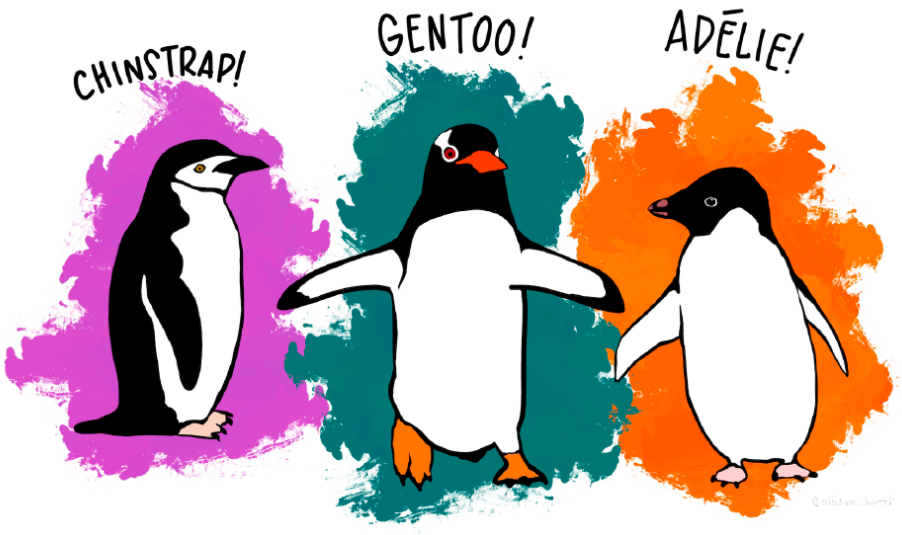

Figure 1: The Palmer Penguins Dataset. Horst AM, Hill AP, Gorman KB (2020). palmerpenguins: Palmer Archipelago (Antarctica) penguin data. https://allisonhorst.github.io/palmerpenguins/

test set **penguins307-test.csv**. Note that the final column in these files lists the class label for each instance.

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 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);
- Objective function: squared error (yes, not so good);
- 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 according to Table 1.

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

Table 1: Initial weights

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

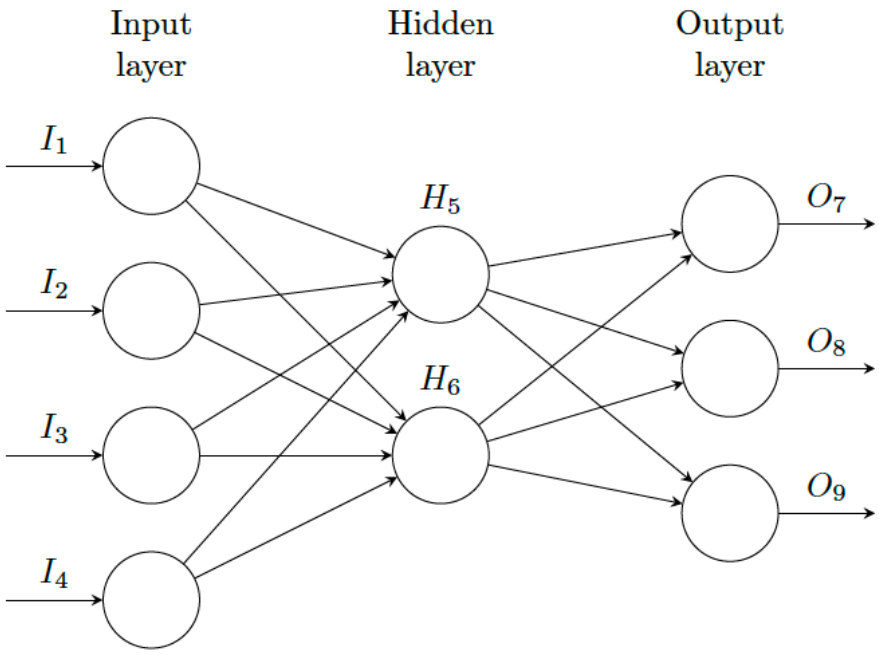

Figure 2: Neural network for the Penguins dataset.

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

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:

3.1 Program code [25 marks]

Program code (both forward pass and backpropagation) 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). Include a **readme.txt** describing how to run your program.

3.2 Report on basic program in .pdf format [10 marks]

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 overfitting, underfitting or neither?

5. You may include you answers on the remaining parts of this problems in the same report (use appropriate headings).

3.3 Refinement of Network [10 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 with the initial weights as shown in Table 2. 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

Table 2: Initial weights for the bias nodes (Further improvements to the network)

3.4 Implementation with automatic differentiation [AIML420: 15 marks, COMP307: 10 bonus marks]

While we implement backpropagation manually (without using external libraries) in problem 3.1, in this problem you implement the same network with automatic differentiation. This problem is compulsory for AIML420 students and optional for COMP307 students. You are now allowed to use external libraries and we suggest you use JAX (as shown in class), or PyTorch, but TensorFlow and Julia are also fine.

Try at least two different activation functions and report on the variation in performance with a small range for the number of hidden layers and possibly the number of neurons per hidden layer and include your analysis in your report.

3.5 Cross-entropy [AIML420 only: 10 marks]

Thus-far we have used as objective (loss) function the squared error criterion $\sum_s \|d^{(s)} - o^{(s)}\|^2$, where $d^{(s)}$ is the correct class for input s. For the squared error the slope for output neuron k can be written as $\beta_k = d_k^{(s)} - o_k^{(s)}$. This is just the derivative of the squared error (we ignored the constant factor 2). The proper objective function for classification is, in fact, cross-entropy $J_\theta = -\sum_c \sum_{x \in A_c} \log q(c|x)$ (the outer sum is over all classes and the inner sum over all inputs in each class). Explain how you would find β_k for the output neurons if your loss function is cross entropy. Hint: you need the softmax function for this.

4 Relevant Data Files

The relevant data files, information files about the data sets, and some utility program files can be found as a .zip file on the course homepage (See Assignment 2: https://ecs.wgtn.ac.nz/Courses/COMP307_2024T1/Assignments).

4.1 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.

Notice that you can submit a .zip file to preserve the subdirectory structure you might have created to organise your submission.

Please check **again** that your programs can be run on the ECS machines easily according to your readme. If the tutors can't run your code, you may **lose marks!** Each tutor has a limited amount of time (¡ 5 minutes) to get your code running, so please don't ask them to use Pycharm, IntelliJ IDEA, Visual Studio, etc to run your code. All these IDEs support exporting runnable code.

4.2 Late Submissions

The assignment must be submitted on time unless you have made a prior arrangement with the course **co-ordinator** or have a valid medical excuse. This year, we are using the ECS extension system for all extension requests. Please make a request there if you think you have a valid reason.

4.3 Plagiarism

Plagiarism in programming (copying someone else's code) is just as serious as written plagiarism, and is treated accordingly. Make sure you explicitly write down where you got code from (and how much of it) if you use any other resources asides from the course material. Using excessive amounts of others' code may result in the loss of marks, but plagiarism should result in zero marks!