MCD4710 – Introduction to Algorithms and Programming
Assignment 2 (10%)

Trimester 2 2021

Due: Thursday, September 2, 2021, 11:55pm (week 10)

Objectives

The objectives of this assignment are:
- To demonstrate the ability to implement algorithms using basic data structures and operations on them.
- To gain experience in designing an algorithm for a given problem description and implementing that algorithm in Python.
- To explain the computational problems and the challenges they pose as well as your chosen strategies and programming techniques to address them.

Submission Procedure

Your assignment will not be accepted unless it meets these requirements:

1. Your name and student ID must be included at the start of every file in your submission.
2. Save your file(s) into a zip file called YourStudentID.zip
3. Submit your zip file containing your entire submission to Moodle.

Late Submission

Late submission will have 5% deduction of the total assignment marks per day (including weekends). Assignments submitted 7 days after the due date will not be accepted.

Important Notes

- Please ensure that you have read and understood the university's procedure on plagiarism and collusion available at https://www.monashcollege.edu.au/__data/assets/pdf_file/0005/1266845/Student-Academic-Integrity-Diplomas-Procedure.pdf . You will be required to agree to these policies when you submit your assignment.
- Your code will be checked against other students' work and code available online by advanced plagiarism detection systems. Make sure your work is your own. Do not take the risk.
- Your program will be checked against a number of test cases. Do not forget to include comments in your code explaining your algorithm. If your implementation has bugs, you may still get some marks based on how close your algorithm is to the correct algorithm. This is made difficult if code is poorly documented.
- Where it would simplify the problem you may not use built-in Python functions or libraries (e.g. using list.sort() or sorted()). Remember that this is an assignment focusing on algorithms and programming.

Assignment code interview

Each student will be interviewed during a lab session regarding their submission to gauge your personal understanding of your Assignment code. The purpose of this is to ensure that you have completed the code yourself and that you understand the code submitted. Your assignment mark will be scaled according to the responses provided.

Interview Rubric

0	The student cannot answer even the simplest of questions. There is no sign of preparation. They probably have not seen the code before.
0.25	There is some evidence the student has seen the code. The answer to a least one question contained some correct points. However, it is clear they cannot engage in a knowledgeable discussion about the code.
0.5	The student seems underprepared. Answers are long-winded and only partly correct. They seem to be trying to work out the code as they read it. They seem to be trying to remember something they were told but now can't remember. However, they clearly know something about the code. With prompting, they fail to improve on a partial or incorrect answer.
0.75	The student seems reasonably well prepared. Questions are answered correctly for the most part but the speed and/or confidence they are answered with is not 100%. With prompting, they can add a partially correct answer or correct an incorrect answer.
1	The student is fully prepared. All questions were answered quickly and confidently. It is absolutely clear that the student has performed all of the coding themselves.

Marking Criteria

This assignment contributes to 10% of your final mark.

- Task 1: Select a Pixel (5 marks)
- Task 2: Adversarial Image (5 marks)
- Decomposition: 1 mark
- Variable names and Code documentation: 2 marks

Total: 13 Marks

Introduction

In this assignment, you will write an algorithm that generates an adversarial image to an ANN based on an input image (Figure 1). More formally, you will solve the computational problem of minimally modifying the pixels of an input image such that the number classified by the ANN for the adversarial image is different than that of the input image. Therefore, you will write functions to "attack" an ANN. That is, to try to generate inputs that will fool the network to make a wrong classification.

```
0000000000000000000000000000
0000000000000000000000000000
0000000000000000000000000000
0000000000000000000000000000
0000000000000000000000000000
0000000000000000000000000000
0000000000000000000000000000
0000000000110000000110000000
0000000000110000000110000000
0000000001100000000110000000
0000000011000000000110000000
0000000011000000001110000000
0000000110000000011000000000
0000000110000000011100000000
0000000110000000011100000000
0000000110000000011000000000
0000000111000011110000000000
0000000011111111111000000000
0000000000000000111000000000
0000000000000000011000000000
0000000000000000111000000000
0000000000000000011000000000
0000000000000000111000000000
0000000000000000110000000000
0000000000000000000000000000
0000000000000000000000000000
0000000000000000000000000000
0000000000000000000000000000
```

Figure 1: Visualization of the image with a resolution of 28x28 pixels that is classified to be number 4 by the ANN.

Aside from solving the digit classification task which serves as an intuitive demonstrative example, ANNs are mainly designed to solve critical tasks such as making cancer diagnosis from mammogram images, driving vehicles from sensory input etc. Therefore, it is extremely important to ensure the safety of such ANNs. One common practice of ensuring the safety of an ANN is to attack it, which is typically carried out by minimally modifying the input (e.g., the image) to see if the ANN makes a different prediction (e.g., digit classification). For instance, Figure 2 visualizes an adversarial image generated by attacking the ANN based on the original input image that is previously visualized in Figure 1. That is, when the adversarial image is fed to the network the ANN could be led into thinking the adversarial image is more likely to be number 9 (instead of number 4) based on its following output list

```
[-2.102353, -5.111125, -1.799243, -4.036013, 4.205424, -5.937777, -2.942028, 0.794603,
-2.199816, 4.538266].
```

```
0000000000000000000000000000
0000000000000000000000000000
0000000000000000000000000000
0000000000000000000000000000
0000000000000000000000000000
0000000000000000000000000000
0000000000000000000000000000
0000000000110010000110000000
0000000000110010000110000000
0000000001100000000110000000
0000000011000000000110000000
0000000011000000001110000000
0000000110000000001100000000
0000000110000000011100000000
0000000110000000011100000000
0000000110000000011000000000
0000000111000011111000000000
0000000011111111111000000000
0000000000000000111000000000
0000000000000000011000000000
0000000000000000111000000000
0000000000000000011000000000
0000000000000000111000000000
0000000000000000110000000000
0000000000000000000000000000
0000000000000000000000000000
0000000000000000000000000000
0000000000000000000000000000
```

Figure 2: Visualization of the adversarial image with a resolution of 28x28 pixels that is classified to be number 9 by the ANN. Note that the only difference between the adversarial image and the original image visualized in Figure 1 is the two pixels that are highlighted by the red square.

Select a Pixel (5 marks)

The pixel that is to be flipped should be selected based on its overall impact on the output list of the ANN. For example, the output of the original image is given by

```
[-2.166686, -5.867823, -1.673018, -4.412667, 5.710625, -6.022383, -2.174282, 0.378930,
-2.267785, 3.912823]
```

modifying the pixel at position 238 produces the following output

```
[-2.087871, -5.475471, -1.621999, -4.213924, 4.903939, -6.095823, -2.653019, 0.697776,
-2.230024, 4.242473]
```

which decreases the score for number 4 by around 0.8, while increasing the score for number 9 (i.e., the second most likely number) by around 0.3. Therefore, the overall impact of flipping pixel at position 238 would be around 1.1.

Note:

1. Reducing the prediction of the original number and increasing the prediction of the second largest number consider a positive impact in attacking the ANN.
2. The other way around, i.e. increasing the prediction of the original number and reducing the prediction of the second largest number is considered as a negative impact in attacking the ANN.
3. The function should choose the pixel that will cause the maximum possible impact.

Write function select_pixel(x, w, b) that can be used to either (i) select which pixel to flip that will cause the maximum possible impact or (ii) conclude that no further modifications can be made (i.e., when the overall impact of flipping is negative for all pixels), by adhering to the following specification:

Input: A list of inputs (`x`), a list of tables of weights (`w`) and a table of biases (`b`).

Output: An integer (i) either representing the pixel that is selected to be flipped, or with value -1 representing no further modifications can be made.

For example, the function `select_pixel(x, w, b)` can behave as follows for input list `x`, list of tables of weights w and table of biases b:

```
>>> x , (w, b) = read_image('image.txt'), read_weights_biases('weights_biases.txt')
>>> pixel = select_pixel(x, w, b)
>>> pixel
238
>>> x[pixel] = int(not x[pixel])
>>> pixel = select_pixel(x, w, b)
>>> pixel
210

>>> x , (w, b) = read_image('another_image.txt'), read_weights_biases('weights_biases.txt')
>>> pixel = select_pixel(x, w, b)
>>> pixel
343
>>> x[pixel] = int(not x[pixel])
>>> pixel = select_pixel(x, w, b)
>>> pixel
469
```

You must clearly explain your design and implementation choices for function `select_pixel(x,w, b)` in docstring.
Note: two functions `compute_difference` and `print_image` are provided in the template file to help you with this task and the next one.

Adversarial Image (5 marks)

Write function `adversarial_image(image_file_name,weights_biases_file_name)` that solves the adversarial image generation task.

Input: A string (i.e., `image_file_name`) that corresponds to the image file name, a string (i.e., `weights_biases_file_name`) that corresponds to the weights and biases file name.

Output: A list of integers representing the adversarial image or -1 if the algorithm is unsuccesful in finding an adversarial image.

For example, `adversarial_image` can behave as follows:

```
    >>> file_name = 'image.txt'
    >>> orig_image, adv_image = read_image(file_name), adversarial_image(file_name, 'weights
_biases.txt')
    >>> if adv_image == -1:
    ...     print('Algorithm failed.')
    ... else:
    ...     print('An adversarial image is found!, with ', compute_difference(orig_image, ad
v_image), ' pixels change')
    ...     print_image(adv_image)

    An adversarial image is found!, with  2  pixels change

    <BLANKLINE>
    0000000000000000000000000000
    0000000000000000000000000000
    0000000000000000000000000000
    0000000000000000000000000000
    0000000000000000000000000000
    0000000000000000000000000000
    0000000000000000000000000000
    0000000001100100000110000000
    0000000001100100000110000000
    0000000011000000000110000000
    0000000110000000000110000000
    0000000110000000001110000000
    0000001100000000001100000000
    0000001100000000011100000000
    0000001100000000011100000000
    0000001100000000011000000000
    0000001110000011111000000000
    0000000111111111111000000000
    0000000000000000111000000000
    000000000000000011000000000
    000000000000000111000000000
    000000000000000011000000000
    000000000000000111000000000
    000000000000000110000000000
    000000000000000000000000000
    000000000000000000000000000
    000000000000000000000000000
    000000000000000000000000000

    >>> file_name = 'another_image.txt'
    >>> orig_image, adv_image = read_image(file_name), adversarial_image(file_name, 'weights
_biases.txt')
    >>> if adv_image == -1:
    ...    print('Algorithm failed.')
    ... else:
    ...    print('An adversarial image is found!, with ', compute_difference(orig_image, adv
_image), ' pixels change')
    ...    print_image(adv_image)
    An adversarial image is found!, with  2  pixels change
    <BLANKLINE>
    000000000000000000000000000
    000000000000000000000000000
    000000000000000000000000000
    000000000000000000000000000
    000000000000000000000000000
    000000000000000000000000000
    000000000011111111110000000
    000000001111111111111000000
    000000011100000000111000000
    000000111000000000111000000
    000000111000000000111000000
    000000111000000001111000000
    000000011000000001110000000
    000000111111111111100000000
    000000011111111111100000000
    000000111111111111110000000
    000001111100000111111000000
    000001110000000001110000000
    000001110000000001110000000
    000001110000000001110000000
    000001111000000111100000000
    000000111111111111000000000
    000000011111111100000000000
    000000000000000000000000000
    000000000000000000000000000
    000000000000000000000000000
    000000000000000000000000000
    000000000000000000000000000
```

You must clearly explain your design and implementation choices for the function `adversarial_image` in docstring.

Hints:

1- You may want to keep calling select_pixel until you find a solution, what would be your input image in each case?

2- You may need a way to avoid infinite loop, think what will happen if select_pixel keep picking the same pixel every time.

3- You should not try every possibility, i.e., your code should not have a factorial complexity or an exponential complexity.