CS 5600/6600: F24: Intelligent Systems
Assignment 1

Vladimir Kulyukin
Department of Computer Science
Utah State University

August 31, 2024

Learning Objectives

1. Perceptrons

2. Perceptron Networks

Introduction

Like most of the assignments for this course, this assignment has two components: technical reading/writing and coding. The technical reading/writing problem in this assignment is the first one in a series of the reading/writing assignments that will ask you to read scientific papers and write their short technical analyses. All these papers are written by AI masters. While some language may be occasionally dense and may strike you as old fashioned, their ideas are profound and still relevant. Many of these ideas continue to influence the bleeding edge AI research and development.

These papers will give you additional information and background on the issues discussed in class and, possibly and more importantly, some ideas for your final projects or long-term research directions, especially for graduate students. I am not expecting from you an in-depth understanding and mastery of every issue discussed in the paper. My objective is much more modest – to get you to think about various AI-related concepts, problems, and challenges, and ethical issues.

Do not think that if a paper is written a few decades ago and uses some unfamiliar terminology, it is somehow less relevant than a paper full of recent buzzwords (e.g., deep learning, data science, cloud computing, data mining, ChatGPT, generative AI, etc.) and published within the past few years. The key concepts of calculus, classical mechanics, and number theory that are driving the developments in such branches of science as fluid dynamics, aviation, and cryptography are centuries old. In AI, if a paper discusses a problem that you have not heard about, more often than not it means that AI, as a field, has failed to solve it, ignored it, and moved on to other lower hanging fruit, as frequently happens in many branches of science due to the cut-throat publish-or-perish and getting-funding-at-all-costs attitude.

Some students, when writing their analyses, start to recite the paper to fill up space. This problem is not specific to students. I regularly write peer reviews for different journals and observe this problem in many submissions that come across my desk. And, I have done it on occassion, too, to meet the strict page requirements for some conferences or journals. But, do your best to avoid reciting the paper just for the sake of filling up space. You will have only one page to write. If you need to cite some sentence/paragraph/section, by all means do so, but do it to support the point you are trying to make. You can always use phrases like "I agree with the authors' point in paragraph X on page Y, because ..." or "I disagree with the point the authors make in Section Z, because ..." etc. I am (and as my fellow scientists/researchers you should be, too) interested in your analyses of the authors' arguments. If you agree/disagree with a point, briefly explain why. If you find something inspiring, say so and outline the things that you feel inspired to do after reading the paper.

The coding problems ask you to code up simple perceptrons in Python, which will drive home some of the material we discussed in Lectures 1 and 2.

Problem 1 (2 points)

Read the article "A Logical Calculus of the Ideas Immanent in Nervous Activity" by McCulloch and Pitts (McCullochPittsPaper.pdf in the zip) and write a one page analysis of this paper. Your font size should be 11 or 12 points (no larger) and your analysis should include four components: **1)** a brief statement of the problems addressed in the paper

in your own words; **2)** what you agree with/like in the paper and why; **3)** what you disagree with/dislike in the paper and why; **4)** any inspirations you found in the paper. As you read these papers, you should keep a final project in mind. You may get an inspiration for your final project. Each of these points should be addressed in a separate paragraph. Save your summary in `mcculloch_and_pitts.pdf` for Canvas submission.

Read this paper for big ideas. Do not get bogged down in the somewhat archaic notation and terminology. It was one of the very first attempts to formulate a mathematical toolkit that matches nervous activity with logical reasoning. I come back to this paper for inspiration all the time and realize every time I reread it how little I understand and appreciate the depth of this 1943 publication.

Problem 2 (1 point)

Implement the binary `and`, `or`, and `not` perceptrons as Python classes. In each class the `__init__()` function initializes the `weights` and `bias` attributes to appropriate values. As we discussed in class, there are multiple ways of building these perceptrons by using different weights and biases. The `output()` takes a 2-element binary numpy array as input and outputs 0 or 1 as computed by the perceptron. The starter code is in `cs5600_6600_f24_hw01.py`. The unit tests are in `cs5600_6600_f24_hw01_uts.py`.

Below is unit test 1 that tests the and-perceptron. Unit tests 2 and 3 test the or- and not-perceptrons.

```
def test_assgn_01_ut_01(self):
    print('\n***** Assign 01: UT 01 ************')
    x00 = np.array([0, 0])
    x01 = np.array([0, 1])
    x10 = np.array([1, 0])
    x11 = np.array([1, 1])
    andp = and_percep()
    assert andp.output(x00) == 0
    assert andp.output(x01) == 0
    assert andp.output(x10) == 0
    assert andp.output(x11) == 1
    print('Assign 01: UT 01: passed...')
```

Use your implementations of the above perceptrons to implement the binary `xor` perceptron that consists of 2 `and`-perceptrons, 1 `or`-perceptron, and 1 `not`-perceptron, as we discussed in class in Lecture 2. The `__init__()` function constructs the four perceptrons as attributes and the `output()` function takes a 2-element binary numpy array and uses the `output` attributes of the four attribute perceptrons to produce the required output (i.e., 0 or 1). The starter code is in `cs5600_6600_f24_hw01.py`. Unit test 4 (see below) in `cs5600_6600_f24_hw01_uts.py` will test your implementation of the xor-perceptron.

```
def test_assgn_01_ut_04(self):
    print('\n***** Assign 01: UT 04 ************')
    x00 = np.array([0, 0])
    x01 = np.array([0, 1])
    x10 = np.array([1, 0])
    x11 = np.array([1, 1])
    xorp = xor_percep()
    assert xorp.output(x00) == 0
    assert xorp.output(x01) == 1
    assert xorp.output(x10) == 1
    assert xorp.output(x11) == 0
    print('Assign 01: Unit Test 04: passed...')
```

Problem 3 (1 point)

Now implement another `xor` perceptron class, call it `xor_perceptron2`, that implements the `xor` logic directly, i.e., in terms of cells, connections, and biases, and not in terms of other perceptrons. This perceptron takes as input a 2-element binary numpy array and outputs a 1-element binary numpy array. The starter code is in `cs5600_6600_f24_hw01.py`. Unit test 5 (see below) in `cs5600_6600_f24_hw01_uts.py` will test your implementation of the xor-perceptron.

```
def test_assgn_01_ut_05(self):
    print('\n***** Assign 01: UT 05 ************')
    x00 = np.array([0, 0])
    x01 = np.array([0, 1])
    x10 = np.array([1, 0])
    x11 = np.array([1, 1])
    xorp2 = xor_percep2()
    assert xorp2.output(x00)[0] == 0
    assert xorp2.output(x01)[0] == 1
    assert xorp2.output(x10)[0] == 1
    assert xorp2.output(x11)[0] == 0
    print('Assign 01: UT 05: passed...')
```

Problem 4 (1 point)

I hope that this problem will drive home one of the points in McCulloch and Pitts' article that networks of perceptrons can be built to simulate arbitrary logical formulats. Toward that end, implement the class `percep_net` that runs a small network of perceptrons to simulate the logical expression $(((x_0 \lor x_1) \land \neg x2) \lor x_3)$, where $\lor$ stands for logical **or**, $\land$ for logical **and**, and $\neg$ for logical **not**.

The starter code is in `cs5600_6600_f24_hw01.py`. Unit test 6 (see below) in `cs5600_6600_f24_hw01_uts.py` will test your implementation of your perceptron network.

```
def test_hw01_ut06(self):
    print('\n***** Assign 01: UT 06 ************')
    pn = percep_net()
    x0000 = np.array([0, 0, 0, 0])
    x0100 = np.array([0, 1, 0, 0])
    x1100 = np.array([1, 1, 0, 0])
    x1101 = np.array([1, 1, 0, 1])
    x1110 = np.array([1, 1, 1, 0])
    assert pn.output(x0000) == 0
    assert pn.output(x0100) == 1
    assert pn.output(x1100) == 1
    assert pn.output(x1101) == 1
    assert pn.output(x1110) == 0
    print('Assign 01: UT 06 passed...')
```

What To Submit

Save your 1-page analysis for Problem 1 in `mcculloch_and_pitts.pdf`. Save and code your solutions to problems 2, 3, 4 in `cs5600_6600_f24_hw01.py`. Zip these two files as `hw01.zip` and submit the zip in Canvas.

Happy Reading, Writing, and Hacking!