CS 5600/6600: Intelligent Systems
Assignment 1

Vladimir Kulyukin
Department of Computer Science
Utah State University

August 31, 2019

Learning Objectives

1. Perceptrons
2. Perceptron Networks

Introduction

This assignment has one technical reading/writing problem and two coding problems. The first problem is the first in a series of reading/writing problems that ask you to read a paper and then write its summary. These papers are meant to give you additional information and background on the issues discussed in class and, possibly, give you some ideas for your final project or a possible long-term research direction. The two coding problems ask you to code up simple perceptrons in Python (please use Py3), which will drive home the material discussed in lecture 1 and make sure that your Python installation is up and ready for the remainder of the semester. Note that I am not expecting from you an in-depth understanding and mastery of every issue discussed in the paper. The point is to get you to think about various AI-related problems.

Problem 1 (1 point)

Read the paper "A Logical Calculus of the Ideas Immanent in Nervous Activity" by McCulloch and Pitts and write a one page summary of this paper. Your summary should include four components: 1) a brief statement of the problem addressed in the paper; 2) what you liked in the paper and why; 3) what you did not like in the paper and why; 4) any inspirations you found in the paper. Each of these points should be addressed in a separate paragraph. Save your summary in `mcculloch_and_pitts_paper.pdf`.

Problem 2 (1 point)

Implement the binary `and`, `or`, and `not` perceptrons as Python classes. In each perceptron the `__init__()` function initializes the `weights` and `bias` attributes to appropriate weights. As we discussed in class, the actual values of the weights and biases are up to you. There are multiple ways of building these perceptrons. The `output()` takes a 2-element binary numpy array as input and outputs 0 or 1. The starter code is given in `logical_perceptrons.py`.

```
class and_perceptron:

    def __init__(self):
        pass
    def output(self, x):
        pass

class or_perceptron:
    def __init__(self):
        pass
    def output(self, x):
        pass

class not_perceptron:
    def __init__(self):
        pass
    def output(self, x):
        pass
```

Here are some unit tests.

```
>>> andp = and_perceptron()
>>> orp = or_perceptron()
>>> notp = not_perceptron()
>>> import numpy as np
>>> x00 = np.array([0, 0])
>>> x01 = np.array([0, 1])
>>> x10 = np.array([1, 0])
>>> x11 = np.array([1, 1])
>>> andp.output(x00)
0
>>> andp.output(x10)
0
>>> andp.output(x11)
1
>>> orp.output(x01)
1
>>> notp.output(np.array([0]))
1
>>> notp.output(np.array([1]))
0
```

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 discussed in lecture 1. 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 `logical_perceptrons.py`.

```
class xor_perceptron:
    def __init__(self):
```

```
        pass
    def output(self, x):
        pass
```

Here are some unit tests.

```
>>> xorp = xor_perceptron()
>>> import numpy as np
>>> x00 = np.array([0, 0])
>>> x01 = np.array([0, 1])
>>> x10 = np.array([1, 0])
>>> x11 = np.array([1, 1])
>>> xorp.output(x00)
0
>>> xorp.output(x01)
1
>>> xorp.output(x10)
1
>>> xorp.output(x11)
0
```

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 also available in `logical_perceptrons.py`.

```
class xor_perceptron2:
    def __init__(self):
        pass
    def output(self, x):
        pass
```

Here are a few unit tests.

```
>>> import numpy as np
>>> xorp2 = xor_perceptron2()
>>> xorp2.output(x00)
array([0])
>>> xorp2.output(x01)
array([1])
>>> xorp2.output(x10)
array([1])
>>> xorp2.output(x11)
array([0])
```

What To Submit

Save your summary document for Problem 1 in `mcculloch_and_pitts_paper.pdf`. Save and code your solutions to problems 2 and 3 in `logical_perceptrons.py`. Zip these two files as `hw01.zip` and submit the zip in Canvas.