HARRISBURG UNIVERSITY
OF SCIENCE AND TECHNOLOGY

CISC 340

Introduction to Artificial Intelligence

Lab Booklet #05

Lab Title: Artificial Neural Networks

Instructors

Brian Grey M.S.

Chad Van Chu M.S.

Objectives

Artificiail neurons and the neural networks that they create are the root of deep learning and much of the AI explosion which has happened over the last 5 years. While neural networks can be used for both supervised and unsupervised learning, we have been examining it from only a supervised paradigm. In this lab, we will be examing different approaches to creating and training neural networks.

After completing this lab, you will be able to:

- **Understand the intrinsic differences of different artificial neurons**
- **Understand how different neurons behave**
- **Understand the gradiant descent training algorithm**

Resources

While the lab will explain everything that you need to know, you may want to consult the following sources for additional knowledge:

- https://en.wikipedia.org/wiki/Perceptron (https://en.wikipedia.org/wiki/Perceptron)
- https://en.wikipedia.org/wiki/Artificial_neuron (https://en.wikipedia.org/wiki/Artificial_neuron)
- https://en.wikipedia.org/wiki/Artificial_neural_network (https://en.wikipedia.org/wiki/Artificial_neural_network)

Deliverables

For this lab, you will need to submit:

- Jupyter notebook files (.ipynb), named and coded as instructed.
  - CISC 340 Lab 05 FA19.ipynb

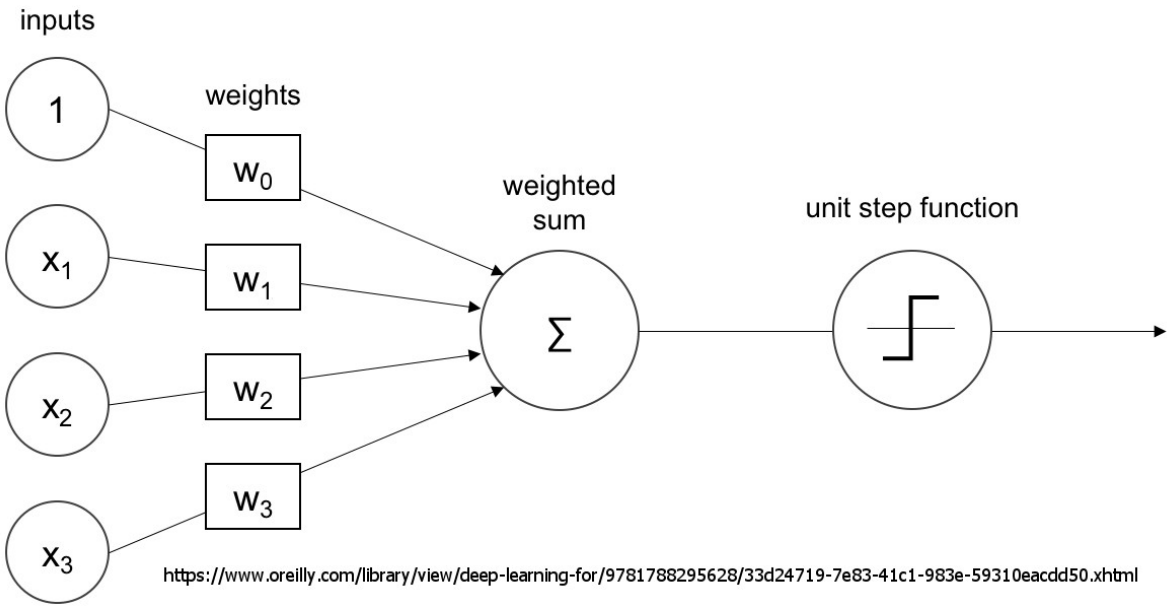
Part 1: Artificial Neurons

We have discussed 4 different approaches to artificial neurons: step-function perceptrons, linear units, squashing sigmoid units, and squashing tanh units.

Step-Function Perceptrons

In [1]:
```
from IPython.display import Image
Image(filename="./images/Perceptron_Diagram.jpg", format="jpg")
```

Out[1]:



Above is the diagram for the structure of a step-function perceptron as we covered in class.

1. Create a `Perceptron` object with the following, publicly-visible methods:

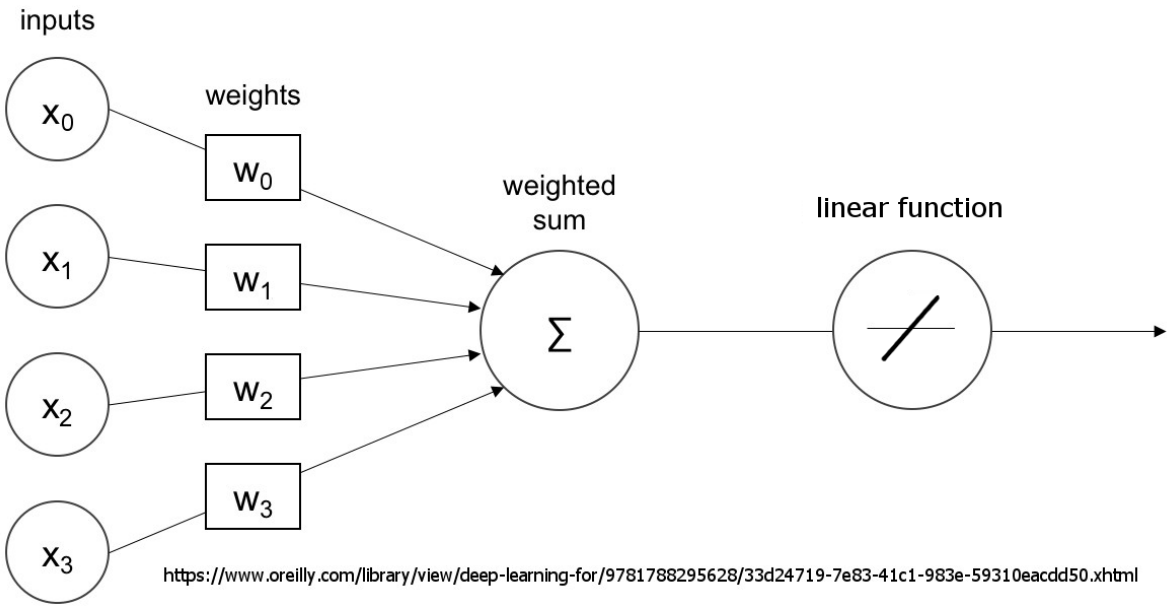
- **`__init__` : Initializer method which takes number of inputs this `Perceptron` expects and configures it appropriately. Note that this number does not include the "bias" input.**
- **`classify(record)` : Method which takes a record which the method will classify. It should return the classification (i.e. the output from the `Perceptron`).**
- **`train(expected_output, output, record)` : Method which takes the expected output of the `Perceptron`, the actual output of the actual output of the `Perceptron`, and the input record which created the output. The method then adjusts the weights of the `Perceptron` using the gradiant descent equation.**

In [1]:
```
# Code Goes Here
```

Linear Units

In [2]:
```
from IPython.display import Image
Image(filename="./images/Linear_Diagram.png", format="png")
```

Out[2]:



Above is the diagram for the structure of a linear unit as we covered in class.

2. Create a `LinearUnit` object with the following, publicly-visible methods:

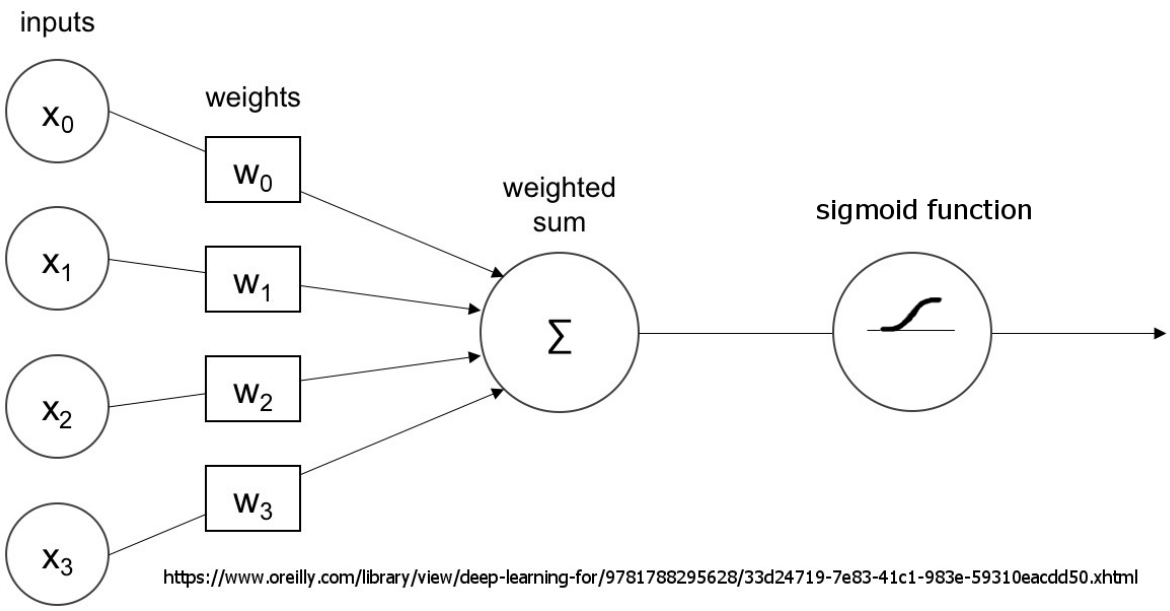
- **`__init__` : Initializer method which takes number of inputs that this `LinearUnit` expects and configures it appropriately.**
- **`classify(record)` : Method which takes a record which the method will classify. It should return the classification (i.e. the output from the `LinearUnit`).**
- **`train(expected_output, output, record)` : Method which takes the expected output of the `LinearUnit` , the actual output of the actual output of the `LinearUnit` , and the input record which created the output. The method then adjusts the weights of the `LinearUnit` using the gradiant descent equation.**

In [2]:
```
# Code Goes Here
```

Squashing Sigmoid Units

In [3]:
```
from IPython.display import Image
Image(filename="./images/Sigmoid_Diagram.png", format="png")
```

Out[3]:



Above is the diagram for the structure of a squashing sigmoid unit as we covered in class.

3. Create a `SigmoidUnit` object with the following, publicly-visible methods:

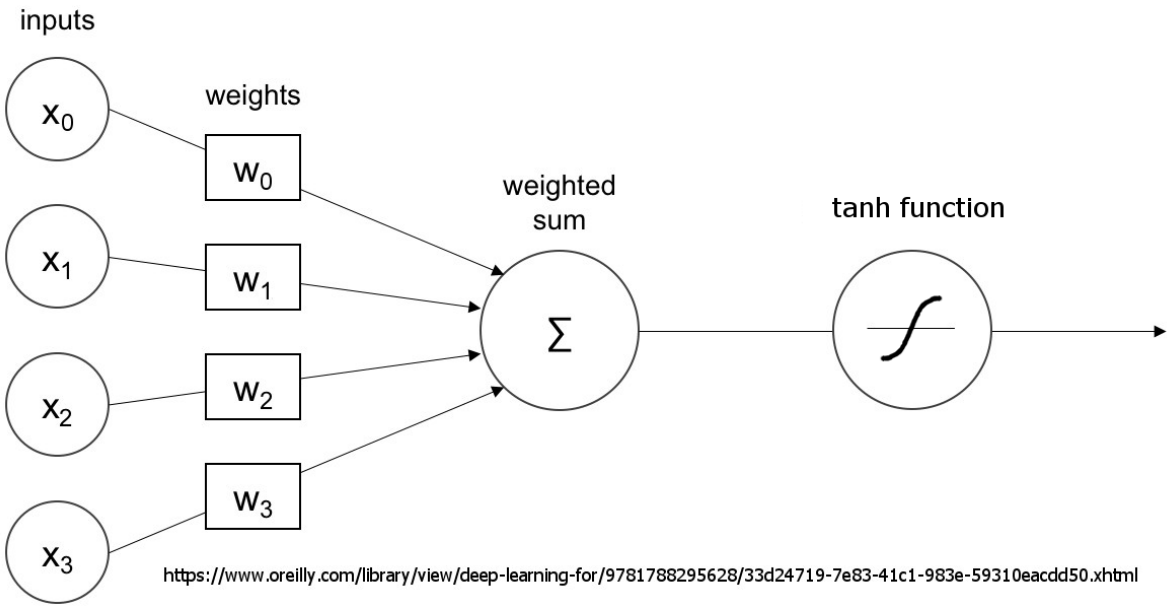
- **`__init__` : Initializer method which takes number of inputs that this `SigmoidUnit` expects and configures it appropriately.**
- **`classify(record)` : Method which takes a record which the method will classify. It should return the classification (i.e. the output from the `SigmoidUnit`).**
- **`train(expected_output, output, record)` : Method which takes the expected output of the `SigmoidUnit`, the actual output of the actual output of the `SigmoidUnit`, and the input record which created the output. The method then adjusts the weights of the `SigmoidUnit` using the gradiant descent equation.**

In [6]:
```
# Code Goes Here
```

Squashing tanh Units

In [4]:
```
from IPython.display import Image
Image(filename="./images/tanh_Diagram.png", format="png")
```

Out[4]:



Above is the diagram for the structure of a squashing tanh unit as we covered in class.

4. Create a `TanhUnit` object with the following, publicly-visible methods:

- **`__init__` : Initializer method which takes number of inputs that this `TanhUnit` expects and configures it appropriately.**
- **`classify(record)` : Method which takes a record which the method will classify. It should return the classification (i.e. the output from the `TanhUnit`).**
- **`train(expected_output, output, record)` : Method which takes the expected output of the `TanhUnit` , the actual output of the actual output of the `TanhUnit` , and the input record which created the output. The method then adjusts the weights of the `TanhUnit` using the gradiant descent equation.**

In [ ]:
```
# Code Goes Here
```

Part 2: Application

Now that we have a variety of artificial units at our disposal, we can attempt to train them. Attached to the lab are three documents: `vote-dem.txt`, `vote-rep.txt`, and `vote-gen.txt`. Each contains a set of Congressional voting records from 1984 as well as a header containing the content of the file. Each voting record is contained in a single line of pairs of characters where 10 is considered a vote for, 01 is considered a vote against, and 11 is considered an abstention. The final bit character of the line contains the party affiliation for the Congressman: 0 for Republican and 1 for Democrat. All other lines are considered comments and start with the double slash (//).

Type answer here!

6. Write a function `trainNeuron` that will take an artificial neuron, a document, and a number of iterations. This function will train the artificial neuron the specified number of times on the document passed in. When complete, the function should return the trained artificial neuron.

```
In [3]: #Write code here
```

Now that we have artificial neurons and a method of training them, we need to have them learn something. Our goal is to teach a neuron so that it can predict the party of a Congressman based on a given voting pattern. However, to do this we need to determine the type of artificial neuron to use, depending on the type of artificial neuron how the output will be classified, and the number of training iterations to run.

7. What type of artificial neuron will you use? How will you classify the output of that artificial neuron? How many training iterations will you run? Provide all supporting rational for your decisions. The rational may or may not involve writing code to decide. If you do write code, please include it below.

```
In [4]: #Any supporting code goes here
```

Type answer here!

We now need to train our artificial neurons. We'll need to write code to train three different artificial neurons of the type that you chose in Question 7. One neuron should train on `vote-dem.txt`, one should train on `vote-rep.txt`, and one should train on `vote-gen.txt`. Once trained, you should test the for accuracy using the `vote-gen.txt` file.

8. Write the code described above and display the accuracy for each neuron to the screen. This accuracy information should be labeled to be understood.

```
In [5]: #Write code here
```

9. What are the comparative accuracies of the three trained neurons? Is this what you expected? Why or why not? How would you account for the differences in accuracy?

In [ ]: