**Purpose:** Perceptron Learning

**Summary:** You’ll experiment with the perceptron learning algorithm and examine the effects of altering the learning rate, size of the training set, etc.

**Details:**

**CREATE THE PERCEPTRON**

In Python, design a simple perceptron class. It should have three instance variables:

- an integer $n$, denoting the number of inputs
- an array $w$ of size $n$, containing the weights to be applied to each of the $n$ inputs
- a weight bias to be applied to the “bias bit” of 1.

When you create a new perceptron, you should be able to specify whether the initial weights are random (values between -1 and +1) or fixed to a prespecified initial value, such as 0 or 1.

Your perceptron model should have an “eval” function that returns the value $bias + \sum_i x_i w_i$, where the vector $x$ of input values is passed to the `eval` function as a parameter. NOTE: the output should NOT be “thresholded”, it should just be a real number. We will write the thresholding function separately.

Write two different threshold function, `step` and `sigmoid`. The first should return 0 if its input parameter $x$ is $\leq 0$ and should return 1 otherwise. The second should return the value of the function $1/(1 + \exp(-x))$.

For instance, we might use this class as follows:

```python
if __name__=='__main__':
    p = perceptron(4,"random")
    print step(p.eval([1,0,0,1]) #will print some random value, e.g, -1.453
    print sigmoid(p.eval([0,1,0,1]) #will print some random value, e.g, 2.7372
    q = perceptron(5,"zero")
    print step(q.eval([1,1,0,0,1]) #will print .5
    print sigmoid(q.eval([1,1,1,1,0]) #will print .5
```
Now implement the perceptron training algorithm as given in class. The value of $\alpha$ should be a parameter (between 0 and 1). There should be a cutoff of the training if the perceptron does not achieve perfect training after a certain number of cycles.

**Hand in**

To be announced