**CSCE 823 Summer 2020 – Artificial Neural Networks:**

**Learning Activity - Perceptrons**

From Fyfe’s Perceptrons

Ingredients:

Python; Numpy

Matplotlib for display.

**Simple Perceptron**

A reminder about “design matrices” – although we will be working with 2-feature observations for this activity, our design matrix will prepend a bias feature value (1) to each observation. Thus, we will have design matrices of size N x 3 and corresponding weight vectors of size 3.

A file with a partially completed perceptron class (PerceptronModel) is provided. You will need to finish defining the class and you will also need to flesh out the function used to train the perceptron (.fit). visualization functions are provided to display both the decision boundary and the training history for the perceptron’s weights

1. **Student Coding**: In the PerceptronModel class, complete the function definition for the function predict(self,X) to implement the (simple) perceptron forward prediction. This function should accept a matrix of observations of shape (*n*, 3) and return a column vector of predictions yhat. Your code should implement the weighted, biased sum of inputs, and returns the results of the activation function as described in reading 01: Fyfe section 3.3. Specifically, your perceptron should allow two inputs and a bias term as inputs to the activation as described in equation 3.3:

and your activation function should produce outputs which will be a 0 or 1 (the activation function is a step function known as modified “heaviside”):

1. **Student Coding**: In the PerceptronModel class, complete the function definition for the function computeWeightUpdate(self,X,D) which accepts a training example (x), and a Desired output for that training example (). This function will then compute and return the weight update based on the perceptron learning rule described in Fyfe Section 3.4. This learning rule should compute the update to the 2-number weight vector update of the perceptron based on the correctness of the single desired output and the single perceptron produced output:

where

is a small positive number – the learning rate

is the *j*th feature for the training example

is the desired output

is the output produced by the perceptron

1. Review the function fit(self, X, y, alpha=0.1, maxSteps = 200, errorTolerance = 0.) which accepts a training dataset set of observations (matrix X), a vector of target values (y), a learning rate (alpha), and a max number of training steps maxSteps and executes a training loop to train a perceptron on a specific function until it has met some threshold of correctness (on the training data) or exceeds maxSteps. This function should be able to return an object of the model class (including history info on the training process: the final training weights; the history of weights at each step of the training process; the total training error on the whole dataset at each step; and anything else you want to track. Weights are instantiated () for the perceptron with small random values (using numpy.random.randn). Then, in each iteration of the loop, one example is randomly selected from the dataset and the perceptron input weights are updated as described in Fyfe’s five-step process on Fyfe page 33/34, calling the functions you implemented previously. **You may need to tweak your learning rate** (alpha) to find one that works well – but this should probably be static over the course of a training session. Indicate how you determined the learning rate.

1. Use the small dataset with two inputs and one output representing the input and output of a Logical **AND** function. Your inputs will be either 0 or 1 and your output will be either -1 (for false) or 1 (for true). Since the AND function has 2 Boolean inputs, there are 4 observations (rows) in the dataset (provided). Each row in the dataset design matrix X will have 3 elements: [1 ] where *x*1 and *x*2 are inputs to the AND function. Train the network and explore its training performance using different learning rates. It should train pretty fast. Plot the training performance (error) over the steps of the algorithm.
2. Use the small dataset with two inputs and one output representing the input and output of a Logical **OR** function (4 observations are provided). Train the perceptron and report on performance.
3. Use the a small dataset with two inputs and one output representing the input and output of a Logical **XOR** function (4 observations are provided). You should find that your perceptron cannot converge on correct weights for this function. What happens?
4. For each dataset, train and display the results using the displayResults(model,X,y,title) function (use an appropriate title for each dataset)