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

**In Class Work, Day 03**

Inspired by Deep Learning chapter 6

**ANN Design: Build a multilayer perceptron to solve the XOR problem**

A note about inputs: in this activity, the inputs should only include the raw features – not an additional column of 1 like we used in the perceptron exercise. The reason is that in a multilayer network, the biases are re-introduced in each node – thus they are part of a node instead of part of the data. This will become very important when we do the future exercise implementing backpropagation. This code uses a weight matrix for each layer instead of a single weight vector for each node. Assuming that each node in the fully-connected layer had the same activation function, the entire layer output is computed in a single matrix multiply operation, followed by the activation on the result.

A note on input matrix orientation in practice: The weight matrix for each layer as well as the inputs for each layer should be oriented to minimize the number of matrix transposes. Since the net computation at each layer will require a matrix multiplication of w and x, we would like to set up the input so that the observations of the input pass through the network without being transposed (e.g. if observations are in columns, then the observations outputs from one layer to the to the next should also be in columns). At each layer, when the dimensions of *w* are (nodecount1, featurecount) and the dimensions of *x* are (featureCount, observationCount) the multiplication of *w*1*x* yields an output to the next layer of (nodecount1, observationCount). Then, in the second layer, the multiplication *w*2*x* of weight shape (nodecount2, nodecount1) and input of shape (nodecount1, observationCount) yields a matrix shape of (nodecount2, observationCount). Therefore when we first send a normally oriented matrix (observationCount, featureCount) to the network as an input, it should first be transposed prior to running forward propagation.

1. Code the math portion of the python function for a layer compute\_foward(self,inputs) to implement the forward prediction of the layer of neurons with that layer’s activation function. This function should accept an input **matrix** where the observations are column-wise. Your code should implement the weighted, biased sum of inputs, and return the results of heavyside activation function. Obtain the weights and bias of the layer from the layer’s object attributes using self.w and self.b. Note that you should perform this action using a dot product instead of using a for loop over features. Specifically your function will efficiently compute the result of and return:

1. Run the network using the original weights. The initial weights and biases probably won’t work for the XOR problem. Try tweaking the values a few times to see if you can get a better result. If you are struggling, refer to the slides on a 3-node multilayer perceptron for XOR weights.
2. If you get this working, try implementing other networks which use relu and/or sigmoid activations in layers instead of heavyside activation. Did you have to change the weights? Why?