ETGG 4803 **Sigmoid Neural Network**
Lab #9 (60 points) **Due Date: 26 April**

This lab roughly follows Section 18.7 of *Artificial Intelligence* by Russell and Norvig.

Create necessary objects, methods, and functions for each of the following parts.

1. Network initialization
- Begin with one input layer, one hidden layer, and one output layer.
- Initialize network with three parameters: *number_of_inputs*, *number_of_hidden_nodes*, and *number_of_outputs*.
- Each neuron in the hidden layer has *number_of_inputs* + 1 weights. (Why?)
- Output layer has *number_of_outputs* neurons, each with *number_of_hidden_nodes* + 1 weights.
- We typically begin with a fully connected network in which each neuron at every level is adjacent to every neuron in the next level.
- Weights initially can be randomly assigned values between 0 and 1.
- Bias can be assigned an input value of 1 with the weight originally set randomly.

2. Neuron activation
- We will use the sigmoid activation function $g(z) = 1 / (1 + e^{-z})$, where *z* is the dot product of the input values and the input weights for a neuron. Create an activation function that gives the output of a neuron given the weights and the inputs to a neuron.
- Other activation (or *transfer*) functions such as the hyperbolic tangent and Rectified Linear Unit (ReLU) are also commonly used.

3. Forward propagation
- Proceed through each layer of the network calculating outputs for each neuron. Outputs from one layer are inputs to the neurons at the next level.

4. Backpropagation (Figure 18.24, p 734 in Russell and Norvig)
- Backpropagation works backward through the network, calculating error between the expected outputs and the actual outputs forward propagated from the network.
- Weights are updated accordingly as the process moves back from the output layer through the hidden layer(s) to the input layer.
- Create a function *neuron_derivative*(*input*) to calculate the rate of change of the output from a neuron. (Hint: Calculus tells us that the derivative of the sigmoid function $g(x) = 1 / (1 + e^{-x})$ is $g(x)(1 – g(x))$.)
- Calculate the error for each output neuron *j*, where *error_j* = (*expected_j – output_j*) * *neuron_derivative*(*input_j*).
- Calculate the error for each neuron *i* in the hidden layer, where *error_i* =*neuron_derivative*(*input_i*) * Sum (*weight_i_to_j* * *error_j*)

5. Network training (Figure 18.24)

- Errors are used to update weights.
- Calculate each new weight: *weight_i_to_j = weight_i_to_j + learning_rate * error_j * output_i*.
- This is also used to update bias by considering bias as a fixed input of 1 and updating its weight.
- Learning rate should be low. For example, *learning_rate* = 0.2 means the weight will update 20% of what it could do. Slower learning over a large number of training iterations is preferred over premature convergence that results from quickly finding a set of weights that minimizes error. This type of premature convergence corresponds to local optimization in GAs.
- We usually increase the learning rate when we have a small training set.
- Use stochastic gradient descent, looping through the network a fixed number of times or until no changes are seen, updating weights as we process each input. (This is *online learning*. If errors were accumulated and weights updated at the end of each cycle, then we have *batch learning* using batch gradient descent.)

6. Prediction

- Input a test point into the neural network and receive the outputs.
- For classification, select the category corresponding to the output neuron with largest output.