**CSCE 823: Machine Learning**

**Summer 2020**

**HW1**

In this assignment, you will explore using perceptrons for classification. You will be starting with the provided jupyter notebook code in this assignment. Each step listed below should correspond to code and/or markdown in the notebook.

You should practice applying techniques and methodology, and present evidence and draw conclusions about the methods and models.

This assignment uses Keras (with Tensor\_Flow backend) to model networks quickly. Be sure to read through the online Keras documentation for the function calls used here, and understand the different options from which you choose. Time spent understanding Keras now will help in future assignments and especially your project.

Caution: You will not *need* the GPU backend working for this assignment, however if you test many models inside a loop running on CPU only (or an older GPU) it may take a significant amount of time. For example, if one model takes 60 sec to train, and you try all possible combinations of 4 hidden layer sizes, 4 quantities of hidden layers, 4 learning rates, and 3 activation functions, you will be running for 4\*4\*4\*3 = 192\*60 = 11520 seconds (3.2 hours!)

**Simple Perceptron (code provided; student should examine and discuss results)**

In steps 1-7 keras code is provided which uses a simple perceptron to solve several different datasets including logic gates and the NUT database from Fyfe’s paper. Because these datasets are extremely small, we are reusing the training set as the validation/test set – which is normally a violation of the golden rule.

Notice that in step 8 the simple perceptron fails to learn the XOR function.

TODO: Run (and rerun) this code with different learning rates & maximum epochs of training, look at the results and make observations.

**Multilayer Perceptron (student code required):**

In step 9 your job is to build a multilayer perceptron (using keras) which can, with high probability, consistently learn the XOR function using as few layers and as few neurons per layer as possible. Can you achieve the goal using only 2 neurons in the first layer and 1 neuron in a second layer?

Note: Because 2d XOR has only 4 possible observations, creating a sequestered test set of even 1 example makes the other 3 points in the training set linearly separable. For this activity we will temporarily suspend the rule of not reporting performance on anything but the test set. Instead, use the training set for both training and evaluation of the performance of the network.

Step 9: Build a *new* keras model with multiple perceptrons to handle the XOR problem. This model should have 2 inputs and one (Boolean) output (hard\_sigmoid activation) but *will need more than one layer* to solve XOR. Report performance of a model on the XOR dataset - in substeps 9a through 9d (below). You should consider using different model architectures (number of layers, layer widths, inner-layer activation functions), optimizers, and learning rates and see how these choices affect the model fitting process and performance (perhaps determined empirically inside a cross-validation loop). Your goal is to achieve 100% training accuracy on XOR. How well can you do? What is the smallest number of layers and nodes per layer you can achieve success with?

* 1. **Define your multilayer models in Keras**: Instantiate at least one model in keras. You could use a grid search to search over multiple models (different activations, different layer counts, different numbers of hidden nodes per layer)
  2. **Fit the model(s) on the XOR dataset**: fit the model on the training data. Start with a batch\_size of 1, but consider increasing during your exploration. You will need to decide on the number of epochs to train for (and select an appropriate learning rate). Alternately, you can use the keras EarlyStopping callback to stop once you’ve obtained 100% accuracy. Capture the model history for reporting in a later step.
  3. **Report the performances of the multilayer models on XOR**: Use keras model.evaluate, to obtain and display the score of your model on the full training dataset (batch\_size = 4). Display graphical plots of the model history over epochs (training & validation accuracy vs epoch; loss vs epoch).   
     Repeat the training/evaluation process under different conditions to see how quickly or slowly your model converges and present performance graphs.   
     Explore different layer widths, number of layers, activation functions, learning rates, optimizers. Use keras get\_weights() on the model architecture and report the results. Discuss the performance as a function of your design choices.
  4. **Plot the decision boundary for the *best* multilayer perceptron on XOR**: Explain design choices that worked the best to obtain & report the final weights of the ANN. Using the instructor-provided code, generate the final decision boundary makeDecisionBoundaryBool2(). Describe the shape of the decision boundary. Describe how the weights of the network relate to this decision boundary.

**Optional Challenge:** Make a movie of the decision boundary changing over time as XOR is learned by your network. This movie should contain 1 image frame per epoch.