Peter Kruse
CSI Lab

Overview:

This document serves as an explainer for using the *NeuralNetConstructor.py* file. The Neural Net Constructor contains a streamlined architecture for constructing neural networks; instead of fully coding an artificial neural network, the user can specify the necessary parameters to create a neural network for their data. In addition this python file, our repository includes two example jupyter notebooks, titled *fracking_constructor_test.ipynb* and *walnut_constructor_test.ipynb*. The fracking constructor notebook gives an example of the constructor being used for binary classification, while the walnut constructor notebook gives an example of the constructor's use for multiclass classification.

Before creating the network, we must first import the ANN (artificial neural network) class into our python notebook. We do so with the the line *from NeuralNetConstructor import ANN*. After importing this class and the necessary packages, then we must preprocess our data. We have four datasets: training input, training output, testing input, and testing output. In our input datasets, each row should correspond to one sample, and each column should correspond to one feature. Our output data should correspond to our input, with each row in the output representing the treatment group that the sample belongs to. For datasets with more than two treatment groups, we must apply one-hot encoding to the output data during our preprocessing steps.

In order to create a neural network, we must specify ten parameters. The first is **input shape**. The input shape of the network should be equal to the number of variables (typically taxa) in our dataset. The input shape will be equivalent to the number of neurons in our input layer. Next, we must specify the **number of neurons** in each layer. We do so by inputting a list, with each entry in the list corresponding to the number of neurons in the first, second, third layers, etc. The length of this list is unlimited, so users can create any number of hidden layers in the network. After specifying the number of layers in the network, we have to make specifications for the output of the neural network. We do so by defining the **output activation** and the number of **output neurons**. These two parameters are related, and they depend on the task at hand. Most of our tasks are binary classification tasks, which have two outputs. In this case, our output activation function should be a sigmoid function, and there should be one neuron in the output layer. For multiple output classification tasks, the output activation function should be the softmax function, while the number of output neurons should correspond to the number of classes we are trying to classify.

After specifying these required parameters, there are three optional parameters that we can include. The first is **batch normalization**. If we set the *use_batch_norm* argument to true, then the output of each layer will be normalized. Additionally, if we want to include **dropout** in our network, we can set the *use_dropout* argument to true and specify our **dropout rate**. In situations where our neural network is overfitting, we can include dropout to reduce the degree of overfitting.

After we have specified these parameters, we can train and test the neural network. We train the network by calling the *.train* method and specifying our training input and output datasets. After the model has been trained, we can test it by calling the *.test* method and specifying our test input and output data sets.