Machuria Johnson
ITAI 2376
Adam Blank

Module 2: Assignment - Python Programming Refresher for Deep Learning

The purpose of this module is to reinforce lessons learned during Module 1. We created a neural network that uses the MNIST training data to train a model to identify handwritten numbers and classify them correctly.
This module uses the TensorFlow library and the Keras library within TensorFlow.

I created a docker image containing my Python environment to make installation and configuration more straightforward and efficient. This allows me to recreate the image should I corrupt it or want to use it on a different computer. I created a GitHub repository for the docker container and a sub-repository for the notebooks directory that contains my Jupyter notebooks to allow portability.
The first step in the process is to import the needed libraries into the Python environment. With the libraries loaded, we create and load the train and test datasets from the mnist using mnist.load_data(). Once that has been completed, we normalize the image. The dataset uses 0-255 greyscale values, but the TensorFlow library uses float values, so we divide the images by 255 to create a 0.0-1.0 float datatype value. Next, we convert the labels to categorical; this converts the datasets into a more efficient format for neural networks and is needed when using softmax by providing a clear target for the model to aim for each class.
We now create our model and define the visible and hidden layers. The first layer is the Flatten layer, Flatten(input_shape=(28, 28)); this takes the 28x28 2d array of the MNIST dataset and flattens it into a 1D array of 784 pixels. Then we run Dense(128, activation='relu'), which defines a densely connected neural array; each neuron is connected to all neurons in the previous layer, and the layer has 128 neurons as specified in the command. Activation is the parameter that determines the activation function that the neurons will use; in this case, "real" is used, which stands for Rectified Linear Unit. This will output the input directly if it is greater than zero; otherwise, it will output zero. Finally, we reach our output layer, Dense(10,activation=’softmax'). Again, this is fully connected; as specified by Dense, there are ten neurons related to our ten classes (0-9), and softmax is our activation. This converts the outputs of the last layers into probabilities that will sum to 1. The highest probability is the prediction of the model.
Now, the model is compiled with these parameters: optimizer='adam', loss='categorical_crossentropy,' metrics=['accuracy']. The Optimizer, adam, is Adaptive Moment Estimation. This adjusts the learning rate to improve the weighting accuracy and speed. Loss, categorical_crossentropy, guides the optimizer by showing how far the predictions are from the actual results. Metrics, accuracy, these are used to monitor the training and testing steps.
After the mode has been compiled, we can begin the training with model.fit(). The training and testing datasets are passed as a parameter, the number of epochs and the batch size. Using too high a training epoch count can lead to overfitting, where the model will identify training data very well but struggle with data not included in the training. Too low an epoch count will undertrain and lead to poor overall performance.
Once training has been completed, the model can be evaluated to determine its accuracy against the testing data. The training data has the correct classification, and the model makes its prediction,

ignoring the actual classification. Then, the prediction is compared against the classification to determine how many times the model predicted correctly, and that is the model's accuracy.

GitHub repos:
https://github.com/ablank74/ITAI-2376
https://github.com/ablank74/itai-2376-Modules