David Schulz

CS 4981 011

3/23/2022

**Lab 2: Signals Option**

**Approach**

Because I was given the option, and because I am much more comfortable and familiar with Python and TensorFlow than MATLAB, those are what I decided to work with for this lab. I also decided to just work with the 1-D data from UCI and keep it in the time domain. Conveniently, they already had the training and testing data split into separate files.

For starters, I had to figure out how to normalize the input data so that all values were between 0 and 1. This had to be done because the range of values for every sample was very different, with some samples being in the single digits and others being in the ten thousands. After trying several approaches that ended with models that just made 50/50 guesses, I found that using the keras Softmax layer as the first layer in the network was the perfect method for normalizing this type of data because not only does it scale all of the values between 0 and 1, but it emphasizes the hill or valley part of the signal.

For the rest of the network structure, after experimenting with some different layouts, I got the best test accuracy with 3 fully-connected hidden layers (64, 16, and 5 nodes, in that order) and a fully-connected output layer with one node. The 3 hidden layers used the ReLU activation function, and the output layer used the sigmoid activation function to push the output value toward either 0 or 1.

The optimizer I went with was the Adam optimizer with a learning rate of 0.001. Because of this small learning rate, a large number of epochs was required to get to the best test accuracy. When using the noiseless data, 1000 epochs was sufficient, but an extra 100 epochs was necessary for the noisy data. Any additional epochs for either dataset made their test accuracies worse. The loss function I used was the BinaryCrossentropy function, since the expected output is a binary classification to either 0 or 1.

**Results**

The output of the model is a single value from 0 to 1. If the value is closer to 0, the model was more confident that the input was a valley. If it’s closer to 1, the model was more confident that the input was a hill.

The best test accuracy I could get with the noiseless data was 0.9967. During training, the training accuracy got all the way to 1.0 for some epochs. The best test accuracy I could get with the noisy data was 0.7739. The training accuracy got up to 0.92.

**Difficulties**

I had a few small challenges, but they were easily overcome after some research. The first was figuring out how to normalize the data before training it. Before using the Softmax layer at the beginning of the model, I tried multiple different other approaches that ended with models that just made a 50/50 guess. They never scaled the data to a consistent range. After the Softmax layer started showing some improvement in the model, I had to adjust the network structure and hyperparameters until I got as best of an accuracy as I could.