Program 3 Report

For my program, I decided to do two different networks with a few adjustments between each (please read comments in each network script). I switched my language to Python because it offered a lot more libraries to make writing this program a lot easier. I initially did try to make my own back propagation function and network from scratch (there is a folder in my source code that contains a tutorial on that). But that required a lot of work and I switched to using libraries instead. A lot of my development process involved repeated testing and reformulating data.

How I prepared my data was slightly different between each version. How I split the data was the same; the first 13 columns were the independent variables and the last column was the target variable. For version 1, which went through many iterations… git version history can help with seeing my progression, I finally ended up with one-hot encoding the variable chest pain type, since it had the most categories. I also normalized all variables for version 1 since it was a recommendation for input variables to be within the same range. How I built this network is with 16 input neurons (after one-hot encoding) and 6 neurons for the hidden layers. The output layer had one neuron with a hyperbolic tangent activation function. This was a total of 4 layers. The loss function was a bit tricky to figure out initially. I knew that this network involved classification, since I am training the network to figure out which patient has a presence of heart disease. But, the values associated with that in the data were split across 4 labels (numbers 0 to 4). I looked up the loss function parameters for Keras Sequential, a library I used, and found two loss functions that were appropriate: binary-cross entropy and categorical-cross entropy. I tried categorical since I thought that was best but couldn’t get that to work, so I decided to change the target variable’s values to binary data (I don’t know if this is allowed since I directly manipulated the data before it was fit to the network, but I couldn’t figure out another solution. I did a validation data split (70/10/10) after each epoch and then predicted from test data. The difference between version 2 and version 1 is the absence of one-hot encoding and the number of neurons for the hidden layers. I put 7 neurons instead of 6 in the hidden layers and 14 input neurons for the input layer. For the rest, it was the same. I do realize I could have adjusted more aspects about it, but I ran out of time.

My results were less than ideal. This was my first time making a neural network, especially in Python. So a lot of the libraries I wasn't familiar with. The version with one-hot encoding got better results (75% accuracy) compared to the one without (65% accuracy). There were some surprises such as running these multiple times, the accuracy went up for both. But at some point, it started to level off around 80%. I think what went wrong was overfitting the data. I was making a validation split after each epoch which isn’t really recommended. I found this out after I made the change, but I thought my results were acceptable enough that I understood what was going on. If I knew how to adjust the learning rate of this network using the libraries and changed the way I made validation data, I believe my results would have been a lot better.