**Data Preparation**

In regards to preparing the data for my network, the main thing I did was to sanitize my dataset. I did this by removing any observations that had any missing data, as we won’t be able to get the full picture regarding that observation. The main columns I found these inconsistencies in were the “ca” and “thal” columns. After I removed those entries, I was left with about 296 observations of the original 300 which is still quite a bit of data to work with.

If I noticed there were a large amount of observations that had these inconsistencies (say > 5%), I would be much more curious as to how the data was gathered and potentially want to gather my own data for this network. However, since the data was provided to us for this assignment, I didn’t have to worry about that.

Additionally, I chose to not collapse or categorize any of the data for this assignment as all of it was already in a numerical format and ready to plug into the network. 1-hot encoding wasn’t necessary either for this particular dataset (or at least the way that I constructed my network).

The only other modification I made to the data was regarding the results of the diagnosis (or “num” column). Since 0 means a negative result, and 1-4 indicates a positive result of varying degree, I modified the num column to be a 0 if it equals zero, or 1 if it’s greater than 1. This allowed me to create a distribution of if the patient was diagnosed or not, rather than to what severity their diagnosis was.

**Network Configuration**

To set up my network, I ended up having 3 different hidden layers, each with 8 neurons. I tried testing with only 2 layers, different numbers of neurons, etc. but ended up finding that this configuration (3 layers, 8 neurons each) worked best with the amount of epochs I was performing (which I found the sweet spot to be around 350).

I noticed that if I decreased the amount of layers/neurons, I would then need to increase the number of epochs my network needed to learn the data, but I then had a much greater risk of my network memorizing my training data as it was being show it many more times with less neurons processing the data itself each time.

Additionally, I used Relu as my activation function for all of my hidden layers, and then sigmoid for my output layer.

**Validation Strategy**

In order to divide my training, testing, and validation datasets accordingly, I used a method from sklearn.model\_selection called train\_test\_split. This allowed me to define what two datasets I wanted to split my data into, and what percentage of data to put into each dataset. The distribution of my datasets are as follows:

* Train: 80%
* Test: 10%
* Validation: 10%

**Results**

Overall, my network worked decently, but wasn’t the most accurate. The best accuracy I was able to gain out of my network was ~80% accuracy on my validation dataset, and about 70% on my training/test datasets. As discussed previously, adding/removing layers and neurons did help, but the best balance I was able to find to achieve these results was 3 layers of 8 neurons each, running 350 epochs with a batch size of 10 observations per batch.

**Comments**

After creating this network, I would say the results make sense as there wasn’t a lot of data to work with which means it would require a lot of fine tuning in order to get this network very accurate.

My main comment out of all of this is how simple Tensorflow made this whole process. There is a lot of encapsulation going on behind the scenes to make this work, but it made it relatively straightforward to create this network. There are still a lot of other opportunities for fine tuning as well but I was just shocked by this technological feat.

**References**

The main reference I used during this project was googles documentation, as well as a YouTube tutorial to help guide me through creating the network. Both are linked below:

<https://www.tensorflow.org/guide/keras/sequential_model>

<https://youtu.be/6_2hzRopPbQ>