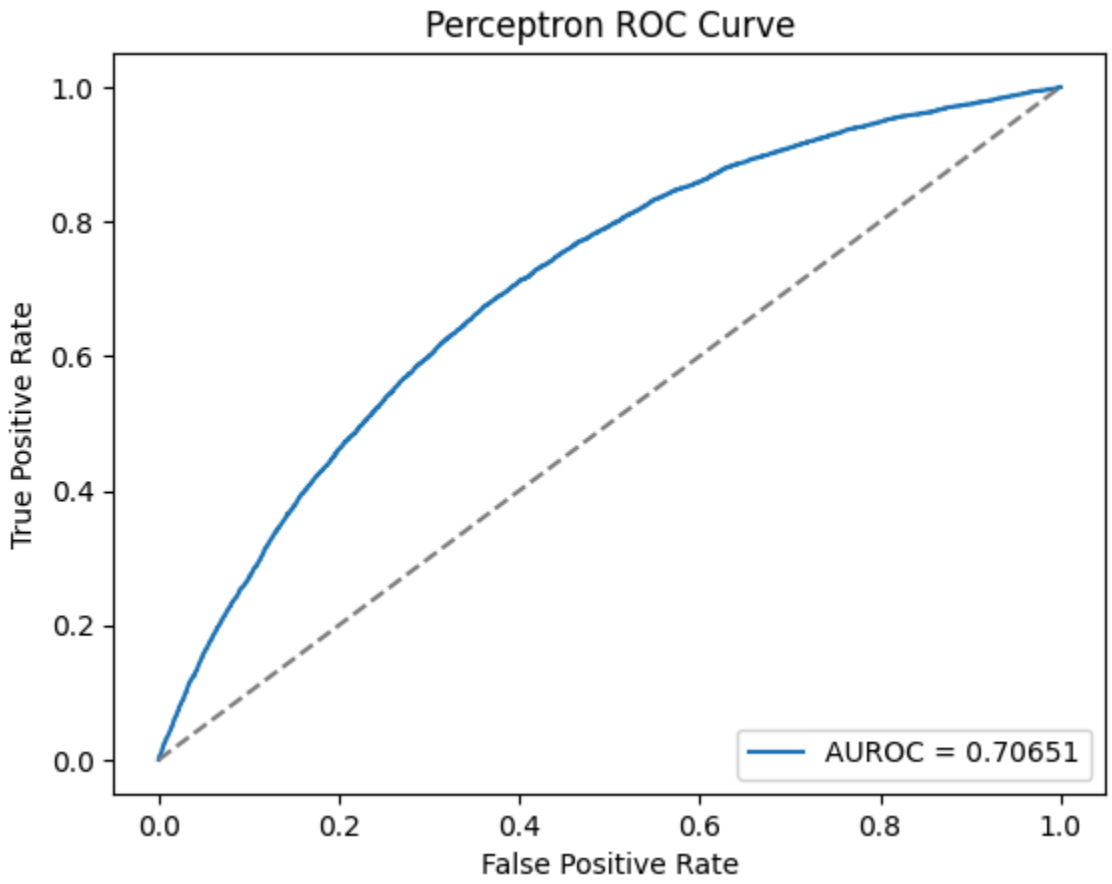
Question 1

1) & 2)

In the train test split, I specified the stratify=y parameter to maintain the same distribution of target classes between the training and testing sets since our dataset is imbalanced. It ensures that both the training and testing sets have roughly the same proportion of 0s and 1s as the original dataset so that the model learns from and generalizes well to data with similar distributions. Without stratification, we might end up with a training or testing set that doesn't accurately represent the overall distribution of classes, potentially leading to biased model performance evaluations. Since this is a perceptron with 1 input layer, 1 output layer, no hidden layers and no activation functions, there is no need to specify any parameters. I set the random states as my NetID (3959) and built a perceptron. I calculated the AUROC and plotted a graph to visualize it.



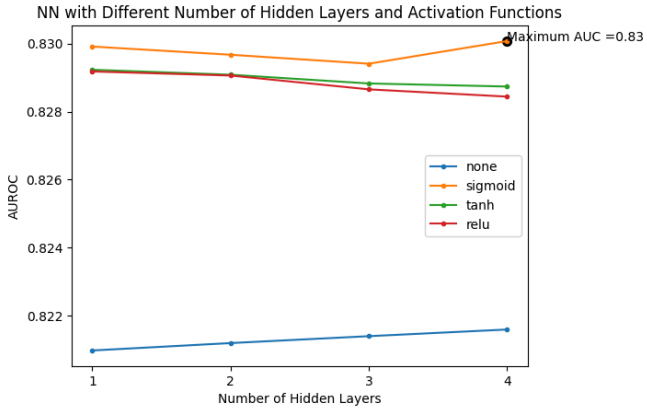
3) & 4)

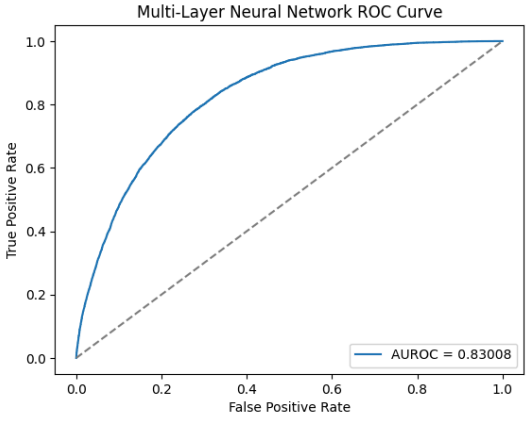
The AUROC of this perceptron is 0.706. Generally, such an AUROC is considered moderately good. It performs better than random guessing to some extent but still has room for improvement. It suggests that the model can effectively distinguish between the two classes (i.e., diabetes or not), but it may not be highly accurate. We could expect it to correctly classify instances but it may still misclassify some instances.

Question 2

1) & 2)

I built and trained a feedforward neural network with at least one hidden layer (multi-layer perceptron) to classify diabetes from the rest of the dataset. I tried different numbers of hidden layers (1, 2, 3, 4) and different activation functions (none, sigmoid, tanh, relu) to find the best combination that gives the highest AUC. I set the numbers of neurons as 10, 4, 3 to follow the same procedure in the lab. The lab built a neural network with three layers with 10, 4, 3 neurons each. Also, I tested and tried different numbers of neurons for a single layer (since a single-layer neural network runs the fastest and is the most time-efficient way for testing). I tried from 10 to hundreds. It turns out that the time needed to run is exponentially growing as the number of neurons grows. Plus, more neurons don't necessarily produce higher AUCs. Sometimes more neurons give lower AUCs. And the difference is not big. So I keep the neuron numbers as 10, 4, 3 to keep it fast and simple as well as following the lab. The logistic function used in scikit-learn's MLPClassifier is the same as the sigmoid function. Both terms refer to the same mathematical function, which is the logistic sigmoid function. The identity function is the same as no activation function. Both retain the input as the output.





3) & 4)

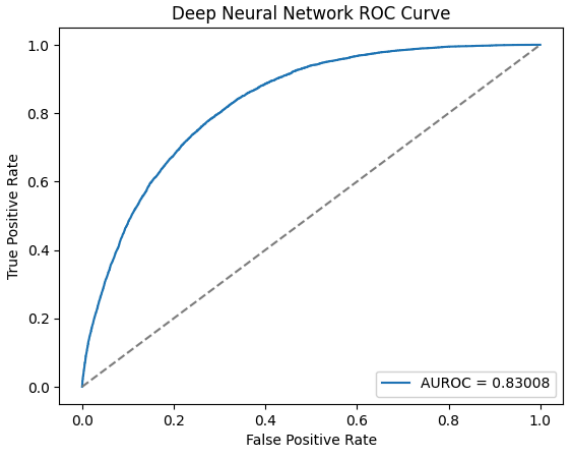
As we can see, the AUC increases as the number of layers increases for no activation function. For the other three activation functions, the AUC decreases as the number of layers increases. Across functions, the logistic sigmoid function has the highest AUC than the other three. The highest AUC is 0.83, achieved with the logistic sigmoid function with 4 hidden layers. This is a lot higher than the perceptron's AUC, meaning this model is performing better and classifying more accurately than the perceptron. It makes perfect sense because a neural network with an activation function and multiple hidden layers ought to be better than a perceptron with no activation function and no hidden layer.

Question 3

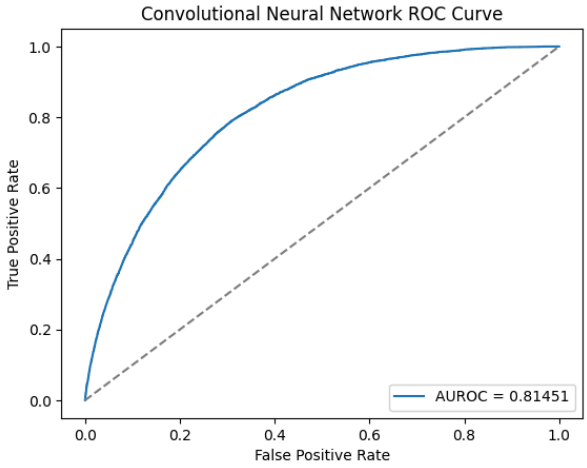
1) & 2)

I built and trained a deep neural network with at least 2 hidden layers to classify diabetes from the rest of the dataset. I used the neural network that has the highest AUC from Question 2, which has the logistic sigmoid as the activation function and 4 hidden layers with 10,4,3,2 neurons. I intentionally split to run and show the AUC after each of the 10 epochs. I also built a CNN with 3 convolutional layers to see if there is a benefit to using a CNN for the classification.

Epoch 1, AUROC: 0.8158406895629292
Epoch 2, AUROC: 0.8232298597446991
Epoch 3, AUROC: 0.8249959489866374
Epoch 4, AUROC: 0.8258480880177463
Epoch 5, AUROC: 0.8259395445908426
Epoch 6, AUROC: 0.8261773815704194
Epoch 7, AUROC: 0.8265121176169044
Epoch 8, AUROC: 0.8267997793314967
Epoch 9, AUROC: 0.8270631020841793
Epoch 10, AUROC: 0.8273138390699222



Epoch 1/10
6342/6342 [==============================] - 13s 2ms/step - loss: 0.3374 - auc: 0.7871 - val_loss: 0.3291 - val_auc: 0.8029
Epoch 2/10
6342/6342 [==============================] - 12s 2ms/step - loss: 0.3298 - auc: 0.8005 - val_loss: 0.3288 - val_auc: 0.8058
Epoch 3/10
6342/6342 [==============================] - 12s 2ms/step - loss: 0.3282 - auc: 0.8034 - val_loss: 0.3256 - val_auc: 0.8081
Epoch 4/10
6342/6342 [==============================] - 12s 2ms/step - loss: 0.3269 - auc: 0.8056 - val_loss: 0.3244 - val_auc: 0.8104
Epoch 5/10
6342/6342 [==============================] - 12s 2ms/step - loss: 0.3260 - auc: 0.8070 - val_loss: 0.3257 - val_auc: 0.8094
Epoch 6/10
6342/6342 [==============================] - 12s 2ms/step - loss: 0.3252 - auc: 0.8084 - val_loss: 0.3235 - val_auc: 0.8125



Epoch 7/10
6342/6342 [==============================] - 12s 2ms/step - loss: 0.3244 - auc: 0.8098 - val_loss: 0.3224 - val_auc: 0.8134
Epoch 8/10
6342/6342 [==============================] - 12s 2ms/step - loss: 0.3238 - auc: 0.8108 - val_loss: 0.3242 - val_auc: 0.8137
Epoch 9/10
6342/6342 [==============================] - 12s 2ms/step - loss: 0.3233 - auc: 0.8115 - val_loss: 0.3222 - val_auc: 0.8138
Epoch 10/10
6342/6342 [==============================] - 12s 2ms/step - loss: 0.3230 - auc: 0.8120 - val_loss: 0.3220 - val_auc: 0.8143

3) & 4)

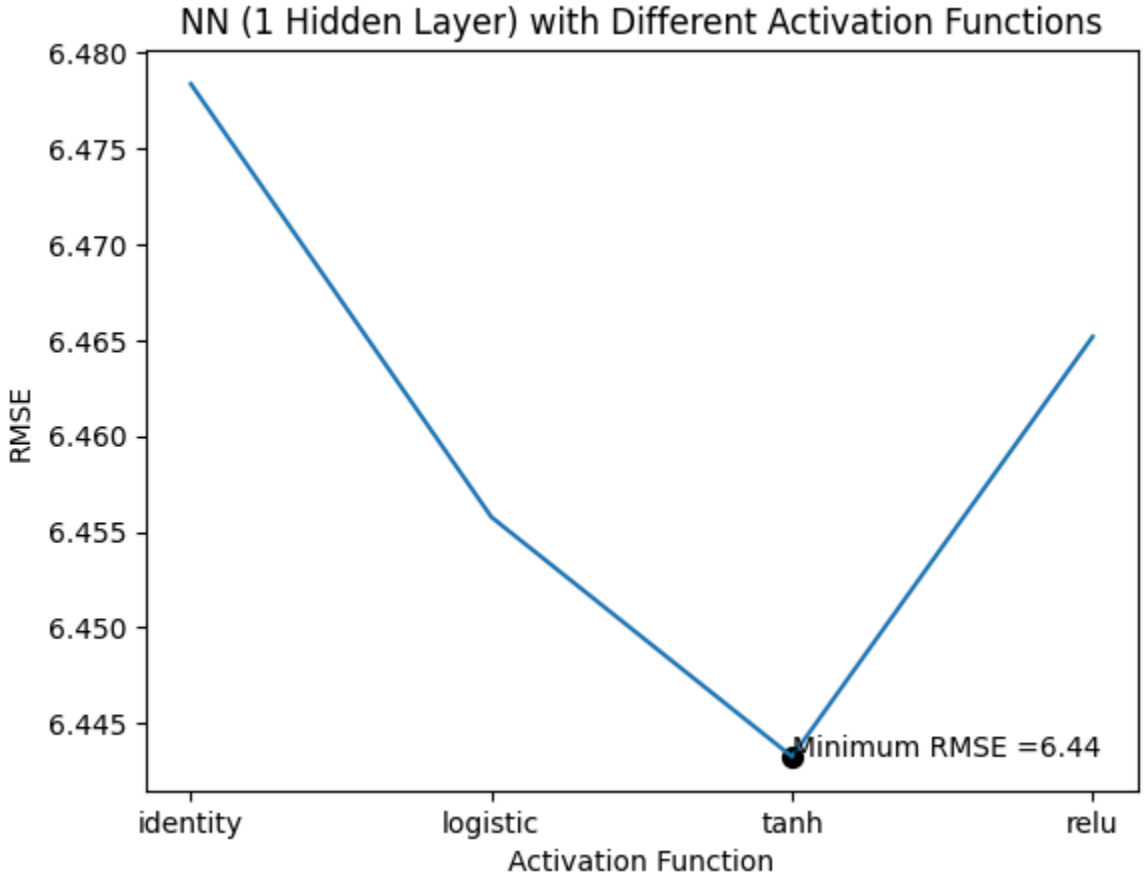
As we can see, for both DNN and CNN, AUC increases after each epoch. However, CNN is not a good fit here. There is no benefit to using a CNN for the classification. The AUC of the CNN is lower than the DNN, meaning that the CNN performs worse than the DNN. Meanwhile, the CNN runs a lot slower than the DNN. This makes sense because CNN is more suitable for grid-like

data such as images. The convolutional layers in the CNN here are not as useful because they are used to scan and capture spatial hierarchies and patterns.

Question 4

1) & 2)

I rearranged the data for predicting BMI. I did the train test split again with the new data. I then built and trained feedforward neural networks with one hidden layer to predict BMI from the rest of the dataset. I tried different activation functions to see if RMSE depends on the activation function used.



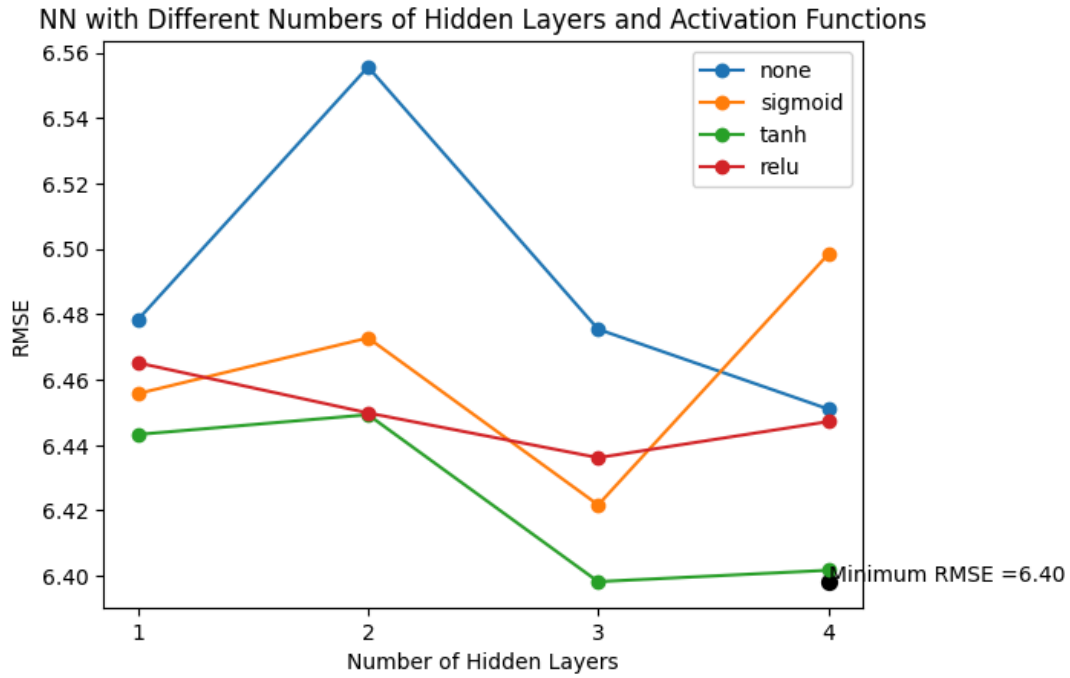
3) & 4)

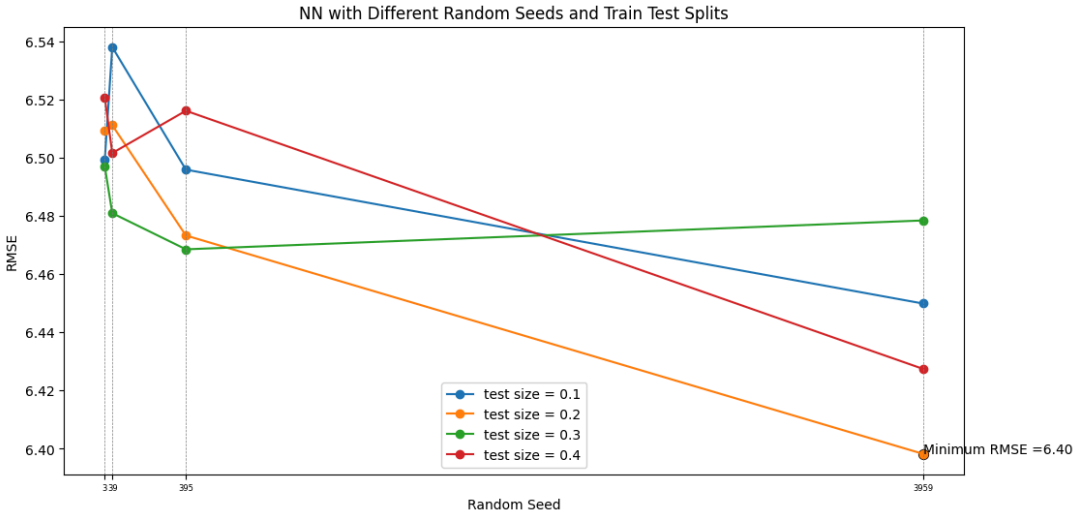
Indeed, RMSE depends on the activation function used. The tanh function gives the lowest RMSE of 6.44. Notice the second lowest is logistic sigmoid, the best activation function from Question 2. Tanh is similar to sigmoid, but it centers at 0 and ranges from -1 to 1 instead of from 0 to 1, which makes optimization easier. Tanh when used in hidden layers of neural networks help reduce the vanishing gradient problem compared to sigmoid. In this case, it's better than sigmoid.

Question 5

1) & 2)

Like Question 2, I built and trained a feedforward neural network with different numbers of hidden layers and different activation functions to see which give the lowest RMSE in predicting BMI from the rest of the dataset. I then tried different combinations of random seeds and train test split sizes to see if that affects RMSE and how low it can get.





3) & 4)

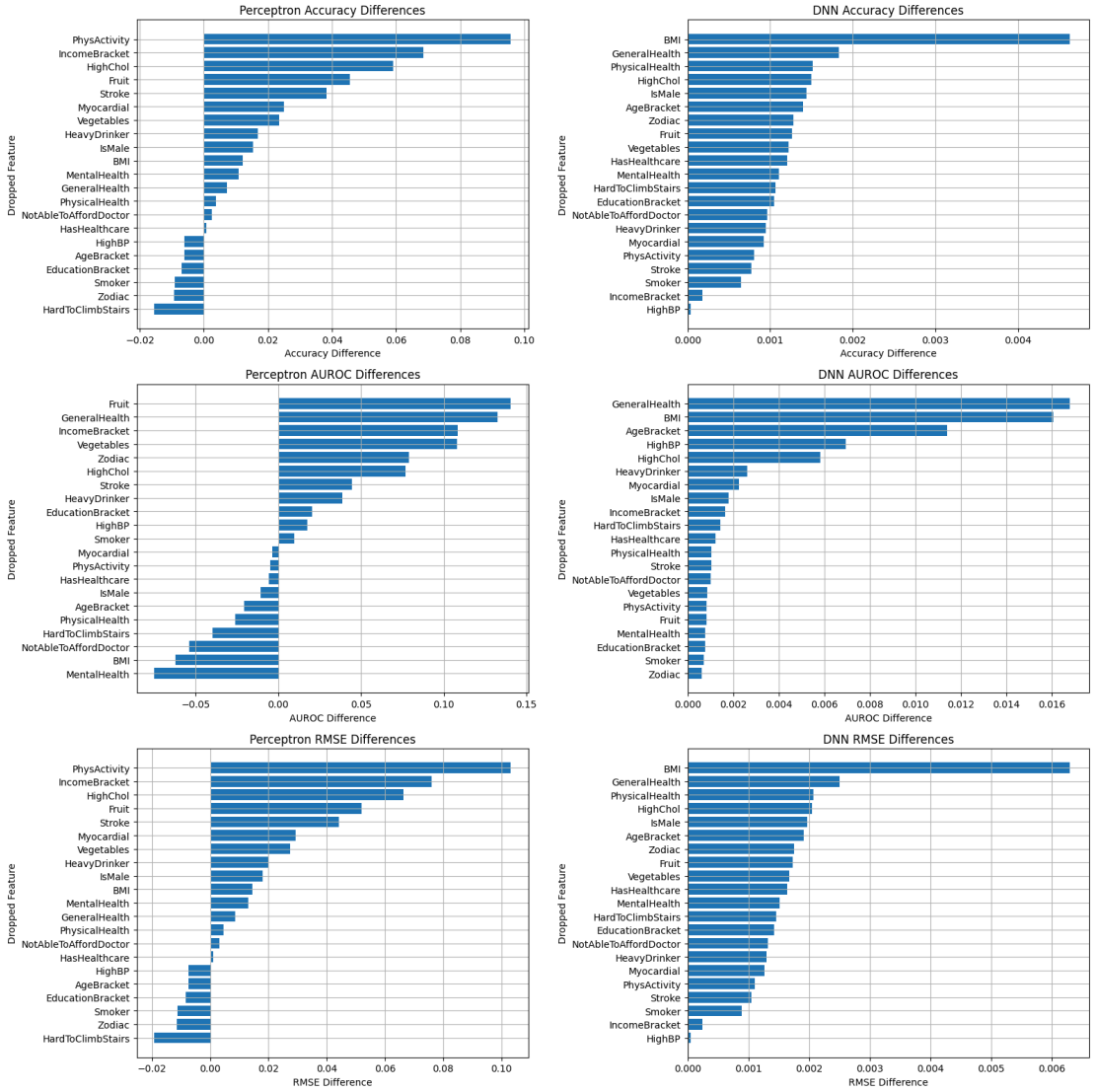
In the first part, the combination of 4 hidden layers (like the result from Question 2) and tanh function (like the result from Question 4) gives the lowest RMSE at 6.4. In the second part, the combination of 3959 as random state and 0.2 as test size gives the lowest RMSE at 6.4. I used 3959 as random state and 0.2 as test size throughout all questions. This confirms that there is

no problem with that. It seems like RMSE depends on all four things: number of hidden layers, activation function used, train test split size, and random state. An RMSE of 6.4 means that, on average, the predictions are off by approximately 6.4 units compared to the actual values.

Extra credit a

a) Are there any predictors/features that have effectively no impact on the accuracy of these models? If so, please list them and comment briefly on your findings

I dropped the features one by one to calculate the difference caused by each drop and compare these differences. Those with difference around 0 are features that have effectively no impact on the accuracy of the model.



For the perceptron, having health care has effectively no impact on the accuracy, AUC, and RMSE while myocardinal issues and physical activiities have effectively no impact only on the AUC. For the DNN, high blood pressure has effectively no impact on both accuracy and RMSE. However, there are no features that have effectively no impact on AUC. Notice that some of the differences for the perceptron are negative, meaning that the perceptron performs better after dropping. This is not a problem in the DNN.

b) Write a summary statement on the overall pros and cons of using neural networks to learn from the same dataset as in the prior homework, relative to using classical methods (logistic regression, SVM, trees, forests, boosting methods). Any overall lessons?

Comparing the values of AUROC, we can see the advantages of using neural networks. In the prior homework, the AUROC values of all five models (logistic regression, SVM, single tree, random forest, AdaBoost) are all between 0.80 to 0.82. On the other hand, the DNN we have has an AUC of 0.83+. This is because neural networks can model complex, non-linear relationships and automatically learn relevant features from the data. From the results I have, I can see more pros of using neural networks than cons. The only cons for now is the running time and computational complexity.