Preliminary:

All categorical variables such as income bracket, age bracket, etc. are converted into dummy variables, as the feedback form HW3 suggests. Also this is because numbers representing a category have no real value, thus it is meaningless to use them for training.

All NN models have 32 neurons for each hidden layer if any. From the last homework I found that logistic regression performed very well, suggesting that the relationship between data is not that complex, thus not many neurons are needed.

For all NN models, I used Adam optimizer with learning rate = weight decay = 1e-3 because it is quick and effective. For classification NN models, nn.Sigmoid() is applied after the output layer so that the AUC can be more easily computed. Also, doing this does not affect the model's predictions. For regression models, no function is applied after the output layer, as it is not necessary to do so.

BCEloss is used for classification because the models are for binary classification. torch.sqrrt(MSEloss()) is used for regression as the questions require.

For all NN models, the full data set is split into 3 groups: train, test, and validation, with 3:1:1 ratio, or 0.6: 0.2: 0.2. For training, I compared the losses and AUCs of the training set and test set to decide how many epochs to train the model. The criteria is the epoch that produces low training loss and test loss. What determines "low" is of my choice because I could not find literature on that matter. Furthermore, for all models, I found that training for more than 15 epochs makes the models overfit alot. The model is then tested by the validation set. The AUC of the model is the AUC of the model on the validation set.

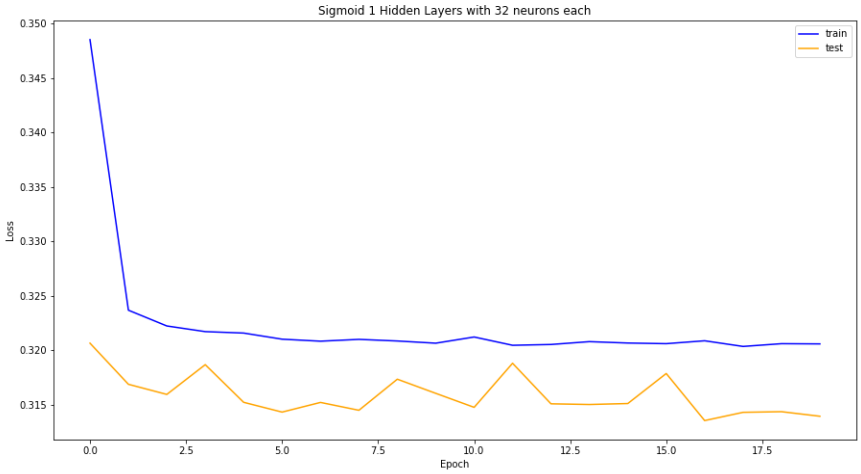
1. Build and train a Perceptron (one input layer, one output layer, no hidden layers and no activation functions) to classify diabetes from the rest of the dataset. What is the AUC of this model?

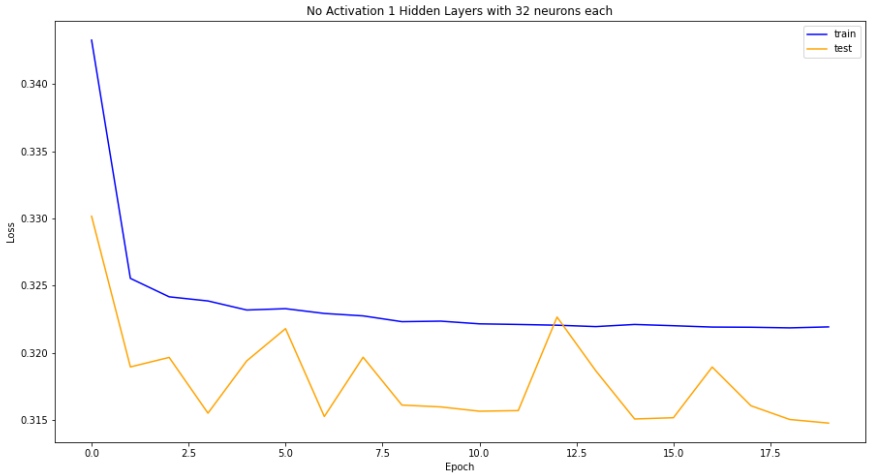
I used sklearn's Perceptron. For this question I tried splitting the dataset with random states 42, 41, and 628. These are arbitrary choices since I only wanted to see the results of different splits. I found that the AUCs of random states 42, 41, 628 to be 0.5312, 0.7875, and 0.7657, respectively. Such a large difference in the AUCs from

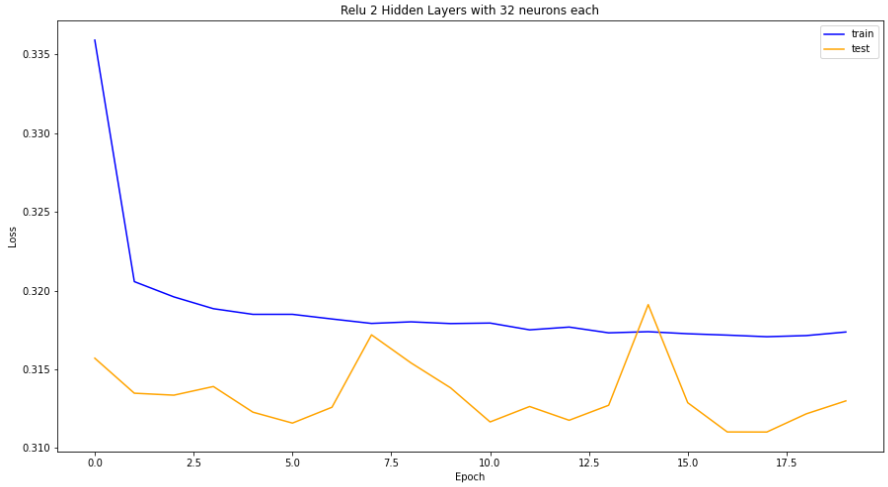
different splits may be due to the imbalances in the dataset,as only 13.933% of the samples have diabetes, which makes a split having very few diabetes in the training set and many more diabetes in the testing set not unlikely.
To find a “better” representation of the performance of the model, I used 5-fold cross validation because this would better represent the true performance of the model than a single split. The result is that the AUC of the model is 0.7892.

2. Build and train a feedforward neural network with at least one hidden layer to classify diabetes from the rest of the dataset. Make sure to try different numbers of hidden layers and different activation functions (at a minimum reLU and sigmoid). Doing so: How does AUC vary as a function of the number of hidden layers and is it dependent on the kind of activation function used (make sure to include “no activation function” in your comparison). How does this network perform relative to the Perceptron?

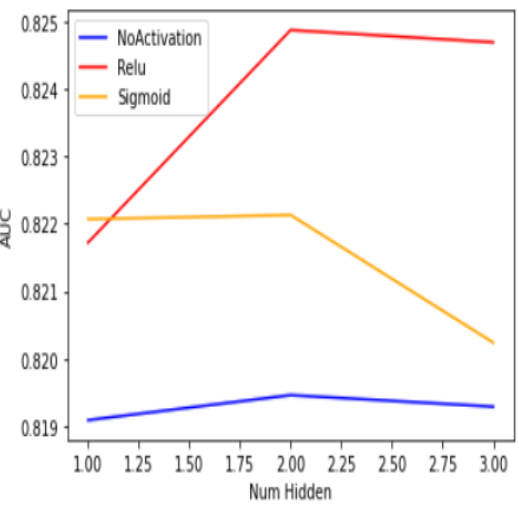
For this question, I tried relu, sigmoid, and no activation, each with 1-3 hidden layers. Since the purpose of this question is to explore the difference hidden layers and activation functions make, all models have the same number of neurons per hidden layer, 32. Through training and testing, I found that for all these models, training for 10 epochs produces the best results, meaning that the models produce both low training loss and testing loss. Below are examples of #epochs vs. loss graph

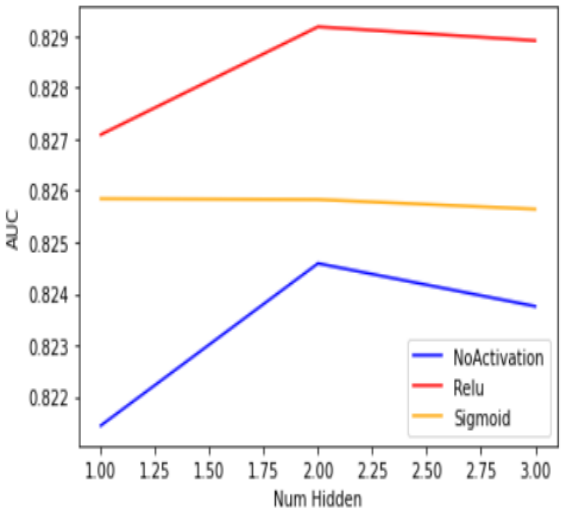


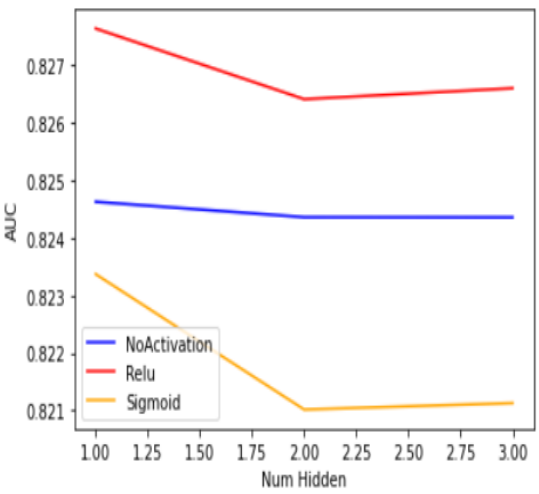




I also tried 3 different splits: random states of 128, 42, 68. Plots that present the difference between activations and #hidden layers for each of the random states in order are below:







It is clear that relu activation produces the highest AUC of all of the ones tested. However, sigmoid out performs no activation for RS = 128, 42 but not 68. One possible explanation is that the classes are linearly separable, which also explains the decent performance of no activation NN.

A trend in AUC vs. #Hidden is that AUC generally increases as #Hidden increases. An explanation is that more hidden layers allow the models to learn more complex relationships. However, this is also not the case for RS = 68. Maybe RS = 68 makes the data split more linearly separable, which makes more complex models (more hidden layers) overfit the data more. Furthermore, RS = 68 generally makes the models learn better, as shown in the AUC comparison table.

The highest AUC of 0.8292 was achieved by 2 hidden layers with ReLU activation.

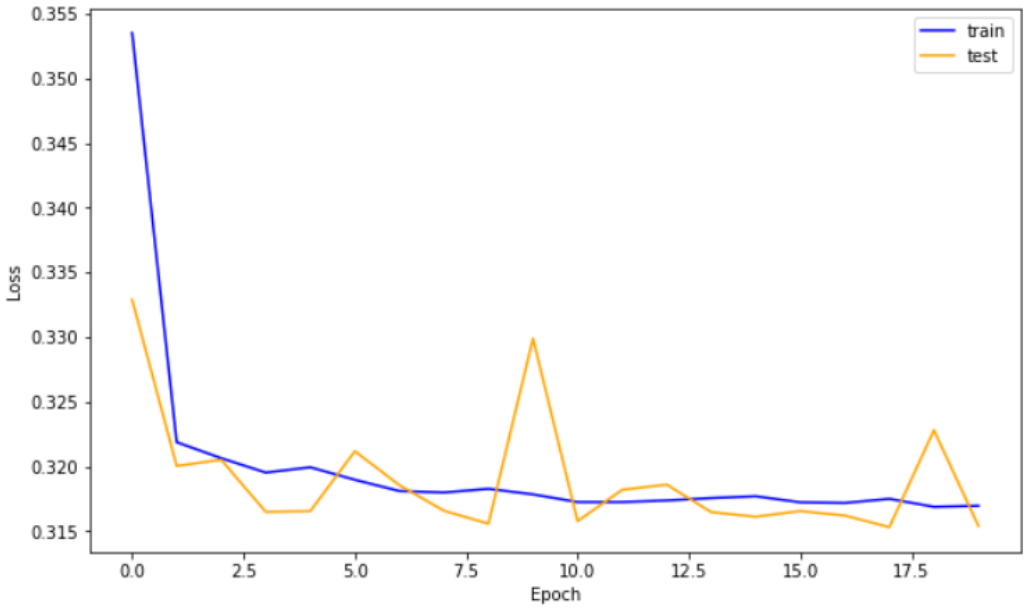
AUCs sorted in descending order is below: AUC comparison table

	Loss	AUC
NN2Relu_RS_68	0.312587	0.829181
NN3Relu_RS_68	0.31286	0.828912
NN1Relu_RS_128	0.314845	0.827644
NN1Relu_RS_68	0.312814	0.827087
NN3Relu_RS_128	0.317991	0.826605
NN2Relu_RS_128	0.316931	0.826414
NN1Sig_RS_68	0.314531	0.825845
NN2Sig_RS_68	0.315231	0.825829
NN3Sig_RS_68	0.315806	0.825643
NN2Relu_RS_42	0.31312	0.824868
NN3Relu_RS_42	0.31337	0.824686
NN1_RS_128	0.318701	0.824629
NN2_RS_68	0.317783	0.824589
NN2_RS_128	0.319572	0.824361
NN3_RS_128	0.318936	0.824359
NN1Sig_RS_128	0.314361	0.823372
NN0_RS_128	0.318686	0.823086
NN2Sig_RS_42	0.31584	0.822127
NN1Sig_RS_42	0.315164	0.822066
NN1Relu_RS_42	0.31521	0.821718
NN1_RS_68	0.318898	0.821441
NN3Sig_RS_128	0.317253	0.821128
NN2Sig_RS_128	0.316097	0.821015
NN0_RS_68	0.320561	0.820776
NN3Sig_RS_42	0.318283	0.820238
NN2_RS_42	0.318296	0.81946
NN3_RS_42	0.325157	0.81929
NN1_RS_42	0.323567	0.81909
NN0_RS_42	0.320945	0.815176

All these models outperform perceptrons. This is expected because these models have more hidden layers that allow them to learn more complex relationships in the data.

3. Build and train a “deep” network (at least 2 hidden layers) to classify diabetes from the rest of the dataset. Given the nature of this dataset, is there a benefit of using a CNN or RNN for the classification?

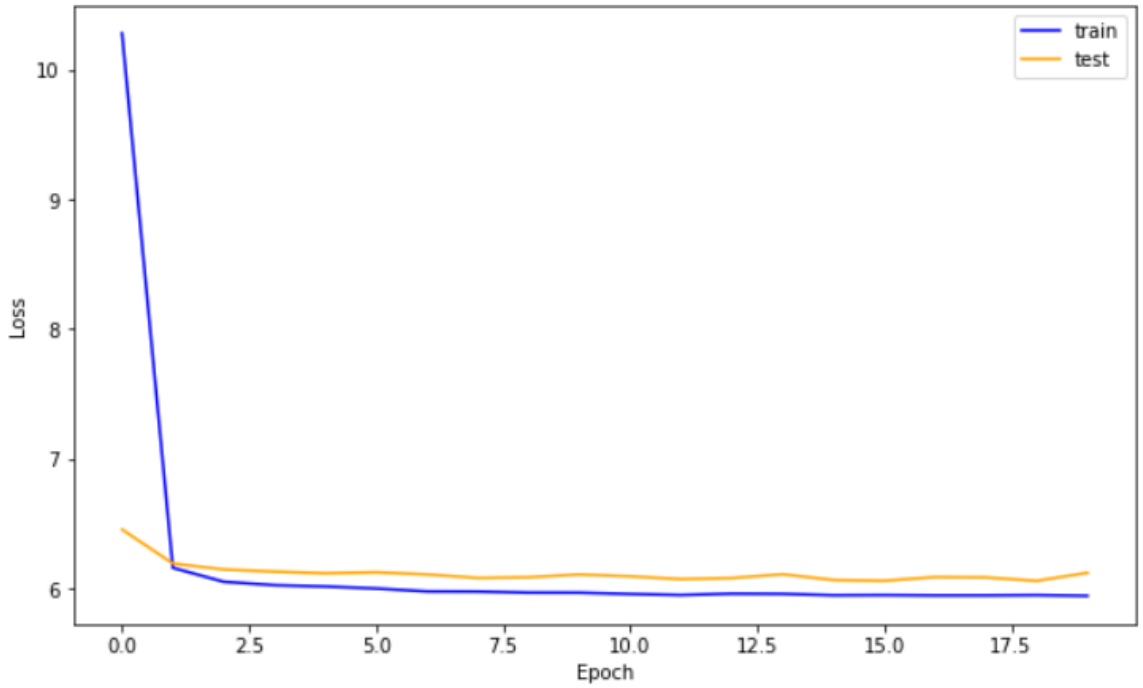
I built a NN model with 4 hidden layers and all ReLU activations because for the previous question, ReLU seems to be the best choice. The graph of Epoch vs. Loss is below:



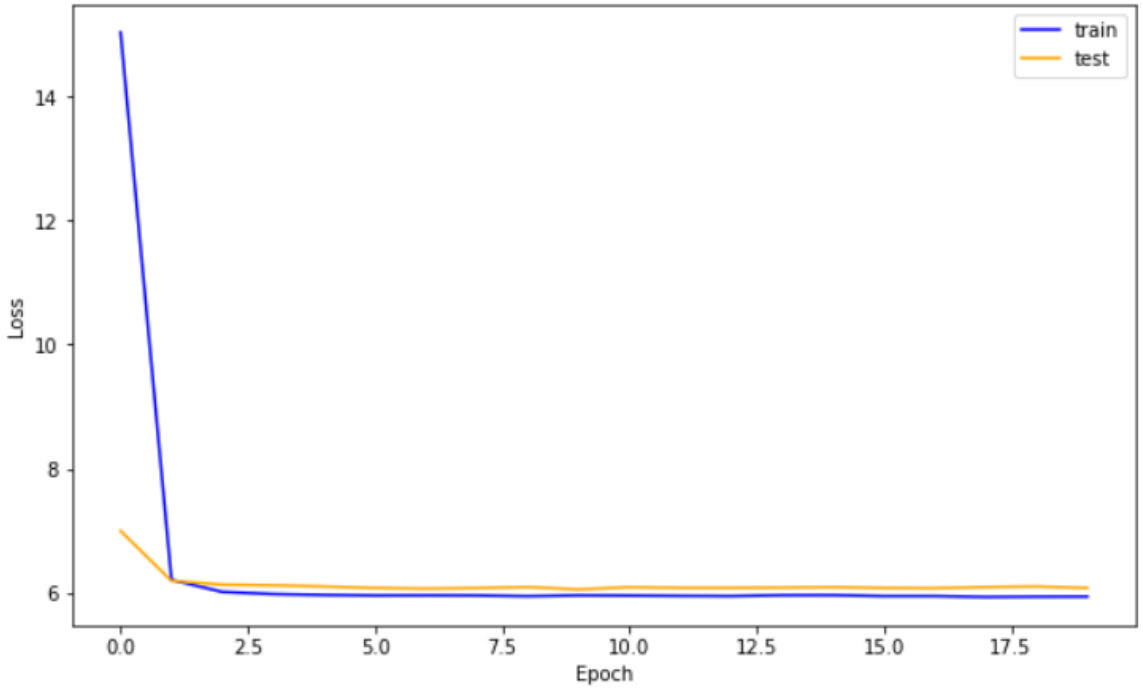
At Epoch 10 (11th training) the model produces low training and testing loss. The testing loss also becomes lower in variance. Thus I chose to train the model 10 epochs.
The AUC of the model is 0.8252.
I don't think there is a benefit of using CNN because the order of the features is not important by any means, while the benefit of CNN is that it takes into account the order of the features like pixels.
There is no benefit of using RNN either because the samples have no temporal relationship but RNN takes advantage of the temporal relationship between samples.

4. Build and train a feedforward neural network with one hidden layer to predict BMI from the rest of the dataset. Use RMSE to assess the accuracy of your model. Does the RMSE depend on the activation function used?

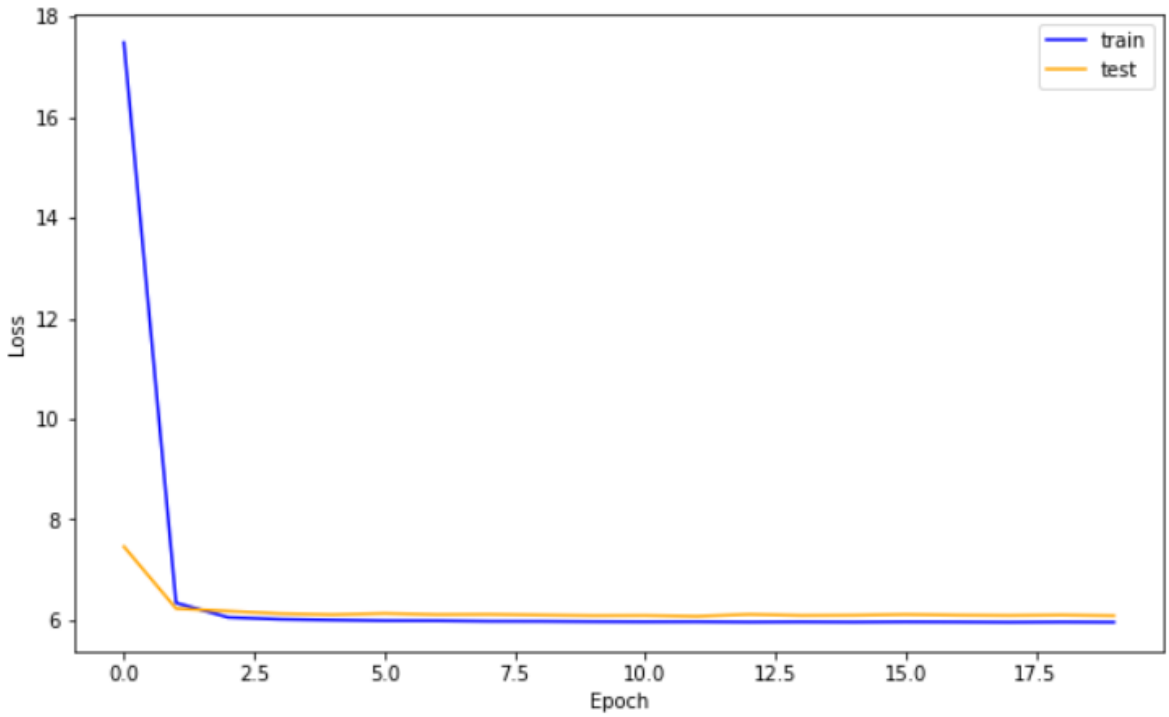
As it requires, the NN I built has 1 hidden layer. Below is #Epoch vs. Loss for ReLU:



For tanh:



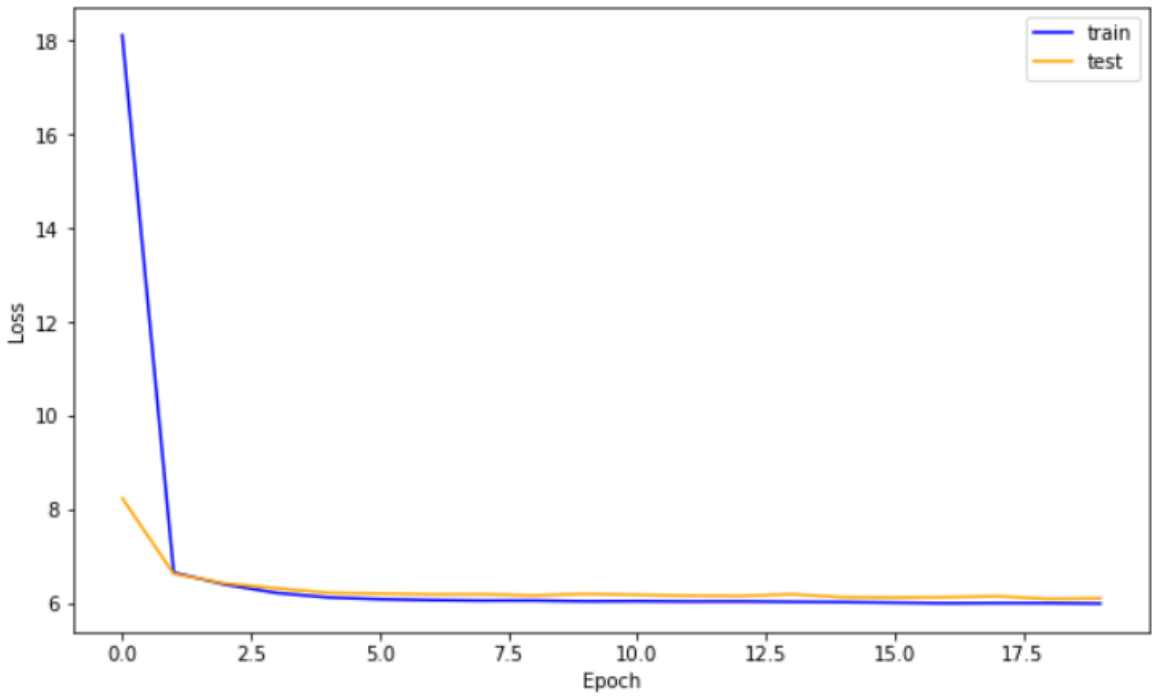
For Sigmoid:



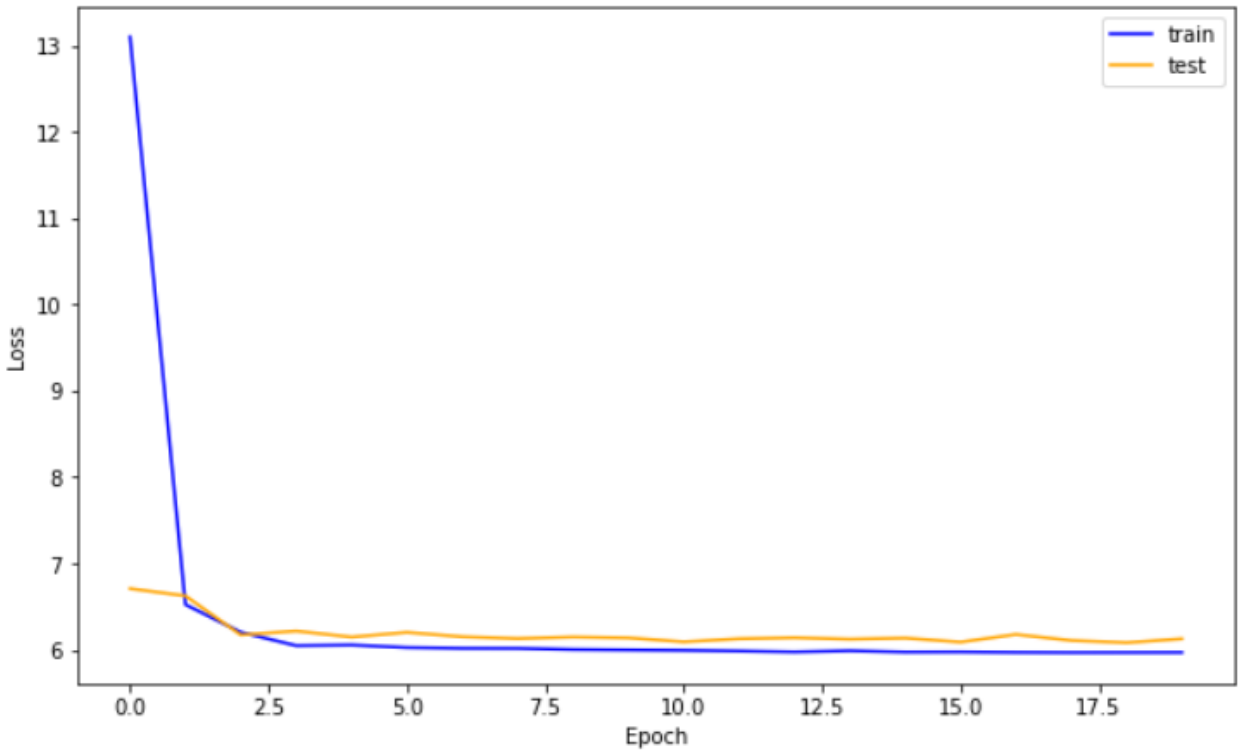
After Epoch = 2 (3 trainings), for all activations, the testing loss stays above training loss, an indication of overfitting. Thus I trained the models for 3 Epochs. The RMSEs of the model with ReLU, Tanh, and Sigmoid are 6.0002, 5.9421, and 5.9313, respectively. RMSE does depend on the activation function, and Sigmoid produces the lowest RMSE out of the three activation functions. This indicates that one hidden layer with Sigmoid suits the dataset the best out of these three for this dataset.

5. Build and train a neural network of your choice to predict BMI from the rest of your dataset. How low can you get RMSE and what design choices does RMSE seem to depend on?

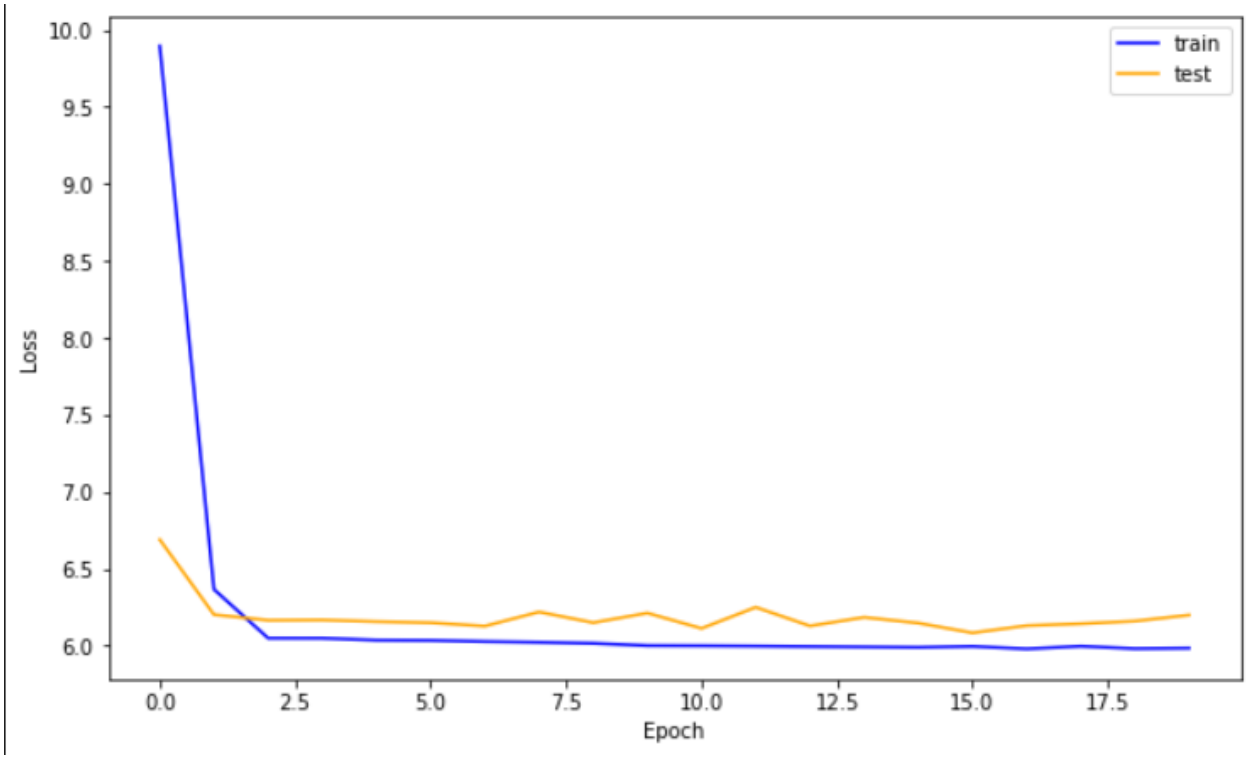
Because in the previous question, I found that Sigmoid performs the best, I built a NN with 4 hidden layers with all Sigmoid. I tried different number of neurons/hidden layer: 32, 64, and 128. Below is the #Epoch vs. Loss for 32 neurons each hidden layer:



For 64 neurons each hidden layer:



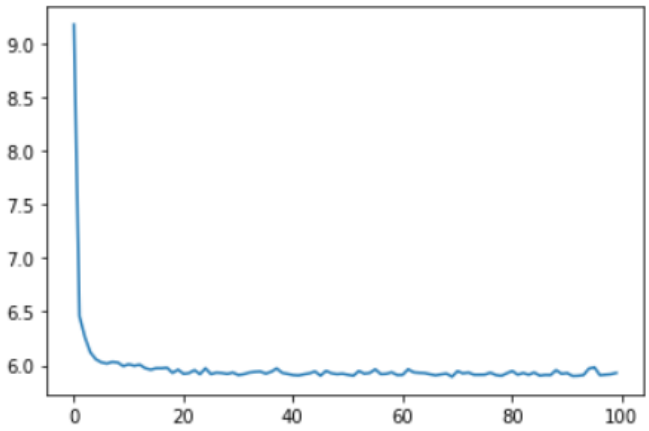
For 128 neurons each hidden layer:



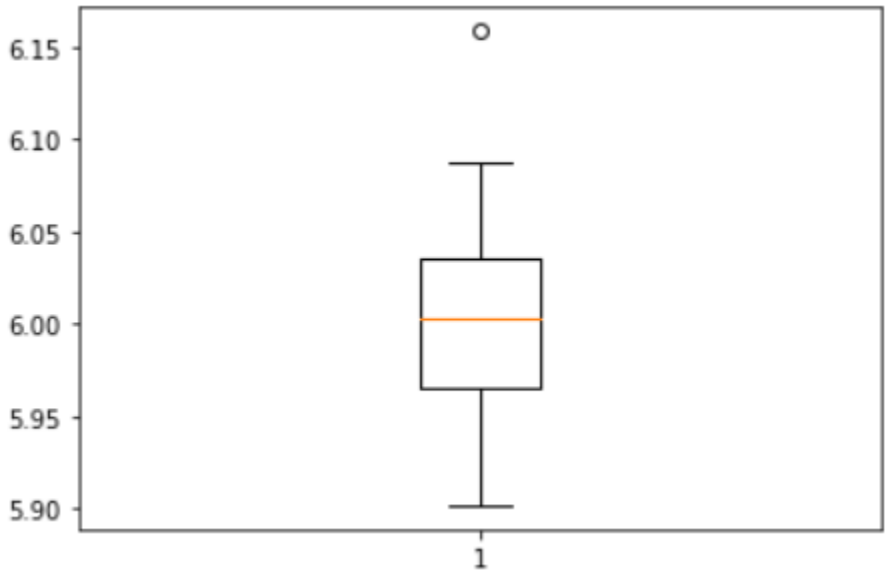
Training the models for 3 Epochs seemed to be the best choice because it has low training and testing loss. Also at 3 Epoch,training loss and testing loss start to cross, meaning that after this epoch, the model overfits.

After training for 3 Epochs, the RMSE of the model with 32, 64, 128 was 6.0319, 5.9868, 5.9858, respectively. 128 neurons performed the best.

Trying to get the lowest RMSE for 32 neurons, I trained the model for 100 times and recorded the RMSEs. The result is that at the 70th epoch, the RMSE is the lowest, 5.8935. Below is the Epoch vs. RMSE graph



Although the RMSE of this model is low, it is concerning because training for 70 epochs can make the model overfit, but not for this model, which leads me to believe that the data split are very highly balanced. To test this, I tried 50 different random states generated randomly. Below is a boxplot of the RMSEs



The mean of all RMSEs is 6.0016 with 0.0516 standard deviation, suggesting that the RMSEs are fairly consistent, which means that the model does not overfit.
So far, I have tried models with different activations, number of hidden layers, and different number of neurons/hidden layer. and they all produced different results, so RMSE seems to depend on the activation function and number of hidden layers, and number of neurons/hidden layer as we see how RMSEs vary with these variables earlier.

EC 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

To find low-impact features, I used the hint from HW3, permutation feature importance. I perform such a method using 2-hidden-layer-ReLU because this NN achieved the highest AUC out of all models tested. Training Epoch is the same as that of Q2 for consistency. The top accuracy with dropped predictor table is shown below:

FruitDropped	0.842963
ZodiacDropped	0.835806
EducationBracketDropped	0.831941
HighBPDropped	0.819684
AgeBracketDropped	0.814736
HeavyDrinkerDropped	0.813105

Dropping Fruit, Zodiac, and Education has the lowest effect on the accuracy of the model. Therefore, these are the least important features. I find this to make sense because the major cause of diabetes is insulin level, and whether a person eats fruit should not affect diabetes. For zodiac, it is obvious that the time a person is born does not affect the chance of having diabetes. Again, the level of education should not affect insulin level. However, perhaps those with higher education level understand the risk of consuming too much sugar, so they may choose to consume moderate amounts of sugar, which makes EducationBracket more predictive than previous features.

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?

The overall pros of using neural networks is that they can perform better than classical models. The highest AUC achieved by classical methods was AdaBoost: 0.8283. 2-hidden-layer-ReLU achieved 0.8292.

The overall cons of using neural networks is the longer training time and lower interpretability. For example, a simple logistic regression can produce an AUC of 0.8204 (from HW3), It took a round 1-2 seconds to train, and I could see the importance of the features from the betas. This is not the case for neural networks. This drawback of neural networks makes answering EC a) harder, for example.
An overall lessons is that avoid neural networks if possible. Simpler models, with some tuning, can achieve very similar performance with the added benefit of interpretability.