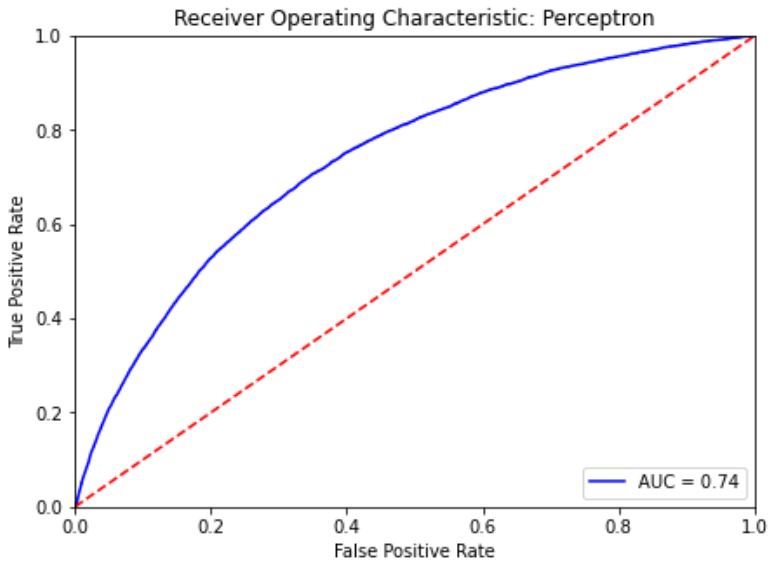
Simone Rittenhouse
Prof. Pascal Wallisch
Into to Machine Learning
23 April 2022

Intro to Machine Learning: Homework 4

General Note: My graphics chip is incompatible with CUDA, so I had to run the models on my CPU instead of sending the data to a GPU to train. Because of this, some decisions in terms of hyperparameter values and the number of neurons in a model were made for the sake of runtime and not necessarily to optimize performance.

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?**
    a. In order to build a Perceptron to classify diabetes, I first set a random seed and pre-processed the dataset. I recoded gender to be (0 = 'Female', 1 = 'Male'). I then z-scored the numeric variables 'BMI', 'GeneralHealth', 'MentalHealth', and 'PhysicalHealth.' I also encoded the categorical variables ('Zodiac' and age, income, education brackets) into binary dummies. After this pre-processing, I split the dataset into a training and test set, with 30% of the data being withheld for testing. Then, I built a Perceptron model using sklearn.linear_model.Perceptron. I also ran my code trying out various values for tol, but found little difference between the values tested (1e-1, 1e-2, 1e-3). I set tol equal to 1e-3 and class_weight to 'balanced' since there was an imbalance between the positive and negative classes. Lastly, I fit the model to the training data, found AUC and the confusion matrix using the test data, and plotted the ROC curve to visualize my results.
    b. Firstly, I recoded gender to ensure its values were in the same range as the other binary variables. I then normalized the numeric variables because z-scored data can lead to a faster training time for neural networks as well as help prevent the model from getting stuck in local minima. Lastly, I encoded categorical variables since their values were not strictly meaningful in their unprocessed form. I chose a 70/30 train/test split because (after also trying 80/20 and 75/25) it led to the highest AUC for my Perceptron. However, it should be noted that both the train-test split value and random seed used did affect my results. I set tol to 0.001 because it's the sklearn default value and my other tested values did not have an affect on the model's performance. Importantly, I set class_weight to 'balanced' to reweight the positive and negative class since few individuals were diagnosed with diabetes. Lastly, I found AUC using the exclusively the test data to ensure that the model can generalize its performance to new data (and avoid overfitting).

c. The Perceptron model had an AUC of 0.73707. This can also be seen in the ROC curve below. The confusion matrix also shows that more individuals were predicted as having diabetes than in actuality - showing there were more false positive errors than false negatives.



		Predicted	
		0	1
Actual	0	36610	28847
	1	2309	8338

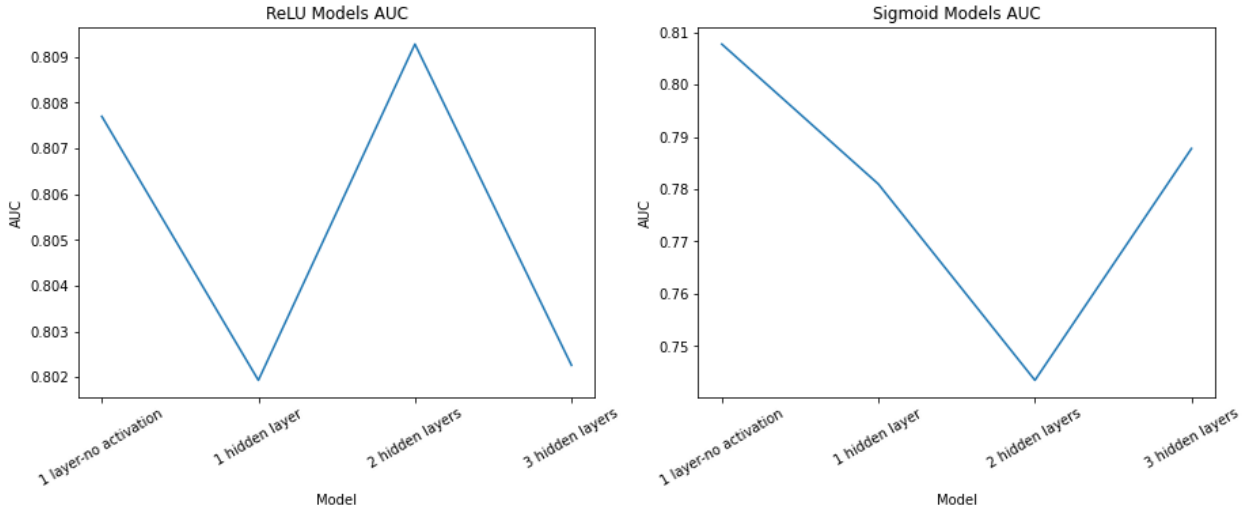
d. The AUC of the Perceptron model classifying diabetes was 0.737. This is better than random chance (AUC = 0.5); however, it's not as high as the AUC of a perfect classifier, which would be 1.0. Additionally, the model had a high false positive rate, meaning that individuals were more likely to be incorrectly predicted as having diabetes than to be incorrectly predicted as not having diabetes. Importantly, my conclusions and results varied slightly when trying different random seeds and train/test splits. This may be because the classes are imbalanced and the dataset is small. Therefore, my results are conditional on the random seed and train/test split I used.

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?

a. In order to see the effect of different activation functions and the number of hidden layers on a feedforward network, I built and trained seven different models using PyTorch nn.Sequential and using the same 70/30 train/test split from Question 1. I had 56 input neurons (one for each feature in my training dataset), and I chose to use 30 hidden layer neurons and a single output layer neuron. In this case, an individual would be predicted to have diabetes if the output neuron's activation is >=0.5. I used BCEWithLogitsLoss as the loss criterion, and stochastic gradient descent as the optimizer with a learning rate of 0.01 and a weight_decay of 0.001. Additionally, I up-weighted the positive class to correct for class imbalance in my loss criterion. The first model had a single hidden layer and no activation function. I then kept 1 hidden layer and an activation function between the input and first hidden layer. I used both a ReLU and sigmoid activation function, and then added a second and third hidden layer to these two models (with no additional activation functions). I trained all seven models on the training data for 1000 epochs, and then computed AUC for each model using torch.sigmoid(model(X_test)) to find the predicted class probabilities. I plotted AUC across the ReLU and sigmoid models to compare performance as the number of hidden layers changed.
b. I used 30 hidden layer neurons because I wanted to decrease the runtime of my models and avoid overfitting on the training data. I used BCEWithLogitsLoss as my loss criterion because it implements binary cross entropy loss (which is suitable for a single output neuron in a binary classification task) and implements a sigmoid function (unlike BCELoss), so it's not necessary to manually add nn.Sigmoid to the models' forward methods. Importantly, I also upweighted the positive class in my loss criterion because I found that without doing so many of the models were only predicting the negative class. I used a learning rate of 0.01 and a weight_decay of 0.001 in my optimizer because, after manually testing several values, I found that this combination was suitable both in terms of runtime and model performance. I also chose to use stochastic gradient descent because, as discussed in lecture, it's more computationally efficient and avoids local minima better than regular gradient descent. Lastly, I used a single activation function in my models after the input layer because I wanted to assess the effect of adding hidden layers alone, so I didn't add additional activation functions with second and third hidden layers.
c. I found that all seven models outperformed the Perceptron from Question 1 with AUCs > 0.74. The model with a single hidden layer and no activation function

had an AUC of 0.80770. For the models with a ReLU function after the input layer, a single hidden layer had an AUC of 0.80194, two hidden layers had an AUC of 0.80928, and three hidden layers had an AUC of 0.80226. For the models with a Sigmoid function after the input layer, a single hidden layer had an AUC of 0.78097, two hidden layers had an AUC of 0.74350, and three hidden layers had an AUC of 0.78776. The ReLU models outperformed all Sigmoid models. Additionally, the Sigmoid model with 2 hidden layers made 0 positive predictions, and the Sigmoid model with 3 hidden layers made only 2 positive predictions - signaling poor performance. For the ReLU models, performance decreased as a third hidden layer was added.



d. Feedforward networks with one or more hidden layers outperformed the Perceptron from Question 1 - achieving AUCs above 0.74. However, AUC depended on the activation function used as well as the number of hidden layers. For this dataset, ReLU models outperformed Sigmoid models. This may be because the ReLU function has an elbow below which activation is 0, which makes the model less affected by noise since low activations are dropped to 0. This also helps prevent the vanishing gradient problem. The sigmoid function, on the other hand, may still produce an activation value greater than 0 due to random noise alone. Additionally, across all models, the number of hidden layers also affected AUC. Models with a single hidden layer and no activation function performed better than models with one hidden layer and an activation function; however, performance increased as hidden layers were added. For the ReLU models, two hidden layers performed best. For the Sigmoid models, performance decreased with a second hidden layer but began to increase with a third hidden layer. This suggests that the addition of more hidden layers to create a deep network may increase the model's performance.

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?**

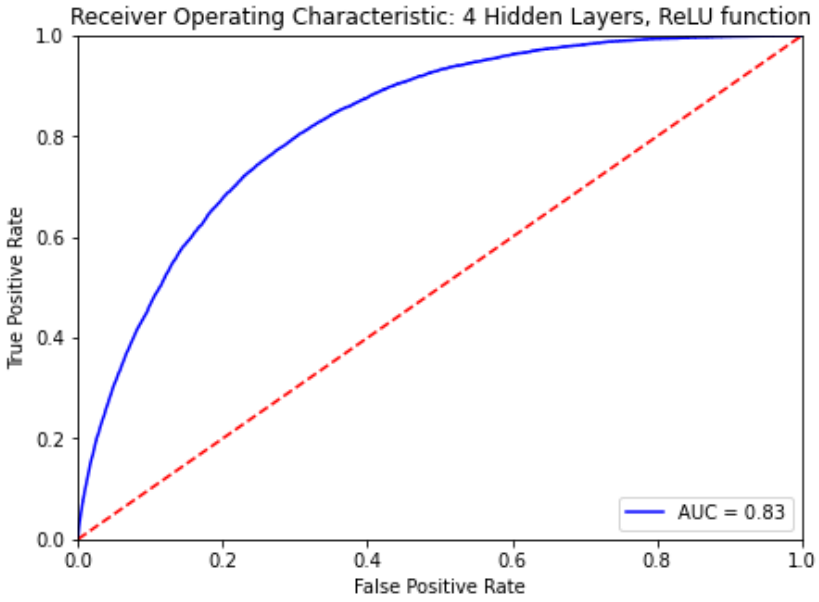
Note: For this question, I trained a deep feedforward network instead of a CNN or RNN because the course tutor suggested this approach on our course Discussion Forum on Brightspace (posts "Clarifications on Q2 & Q3" and "Question 3 'deep' network"). I hope this is an acceptable solution.

a. In order to build and train a deep network to classify diabetes, I used PyTorch's DataLoader to load in the same training data from the 70/30 train/test split in Question 1 with a batch size of 64, shuffling the data for each batch. I then built a deeper feedforward network with four hidden layers and ReLU activation functions between each layer. I again used 30 neurons for each hidden layer, 56 input neurons (for the 56 total predictors) and a single output neuron where diabetes is predicted if the neuron's activation is >=0.5. I used the same hyperparameters as in Question 2 (learning rate = 0.01, weight decay = 0.001). I then trained the model over 100 epochs using BCELossWithLogits as the criterion and stochastic gradient descent as the model's optimizer. As in the previous questions, I upweighted the positive class to correct for class imbalance. I then found AUC for the model using the withheld test data and a sigmoid function to determine the prediction probabilities of the model. Lastly, I outputted a confusion matrix for the model and plotted the ROC curve.

b. I used a batch size of 64 because the small batch size decreased training time while also helping to prevent the model from overfitting on the training data. I used four hidden layers to compare how a deeper model performs relative to the shallower networks in the previous question (which used a maximum of 3 hidden layers). Additionally, I used ReLU activation functions because ReLU models had consistently better performance than the Sigmoid models in Question 2. I kept the rest of the structure the same as Question 2 (i.e. the number of neurons in the hidden layers, the learning rate, weight decay, criterion, and optimizer) so that, again, I could compare how the performance changed as a function of the number of hidden layers alone. I used fewer epochs than the previous questions (100 compared to 1,000) since using batches of the training dataset allowed for more total iterations than previously - where I used the entire training set each epoch. Lastly, I computed AUC and the confusion matrix to check model performance, and plotted the ROC curve to visualize my results.

c. I found that the deeper feedforward model outperformed all of the shallow networks in Question 2. This model with a fourth hidden layer had an AUC of 0.826448, while the model with the highest performance in Question had an AUC of 0.81036. Additionally, the confusion matrix shows that the number of false positives decreased from the Perceptron model (10685 compared to 28847). This

suggests that performance has improved relative to my initial model. The number of false negatives also decreased from the Perceptron, although not as dramatically (2093 compared to 2309).



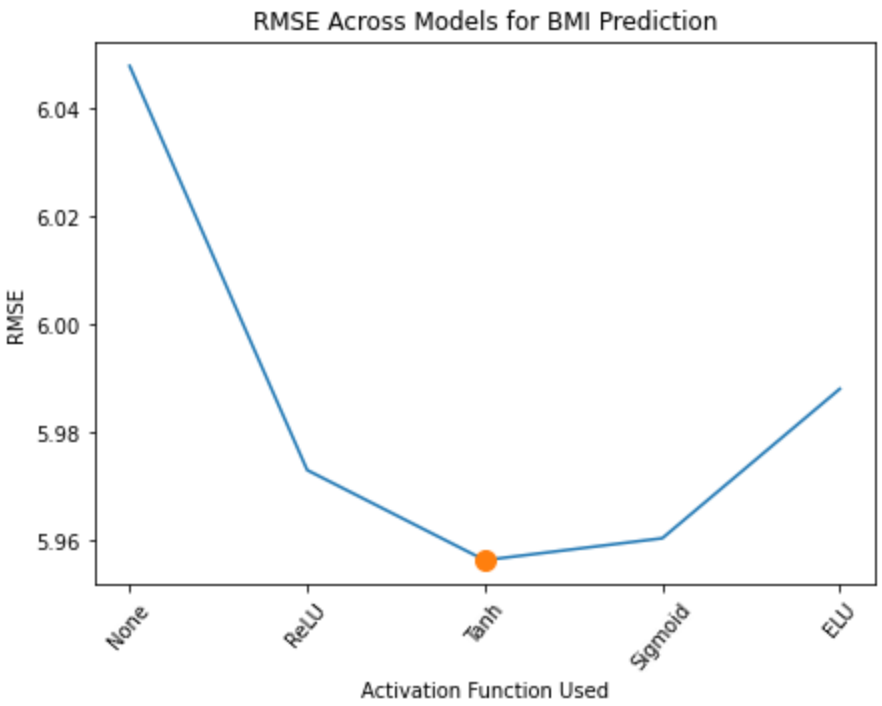
		Predicted	
		0	1
Actual	0	54772	10685
	1	2093	8554

d. A deeper feedforward network with four hidden layers and ReLU activation functions outperformed both the Perceptron and the shallower networks (with fewer than four hidden layers) - achieving an AUC of 0.826. It should be noted that this performance may have improved even more had I used more training epochs; however, I used fewer epochs for the sake of a shorter runtime.

Additionally, there does not seem to be a benefit of using a CNN or RNN for the classification over a feedforward network. We discussed CNNs in the context of image processing and RNNs in the context of time-series data; however, CNNs can be applied to any data with a spatial relationship and RNNs may be applied to any sequence prediction problem. After examining a correlation matrix of the input features, there are few linear relationships between the columns. This implies that the distance between two columns is not very meaningful for the dataset. For example, the values in the Zodiac columns are essentially unrelated to

the values of the BMI column. For this reason, there does not appear to be a benefit to using CNN, which works best on spatially-related data. Additionally, there is no sequential element to this dataset. All values are for a single collection time. Therefore, RNNs (which suppose sequential data) also don't appear to be best-suited for this classification task.

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?**
    a. To predict BMI using a feedforward neural network and test the effect of various activation functions, I built five different models - each with a single hidden layer and different activation function. I used the same 70/30 train/test split as in the previous questions, and I used nn.Sequential to construct my models with 56 input neurons (for each feature in the training dataset), 30 neurons per hidden layer, and a single output neuron representing BMI prediction. I used RMSE as my loss criterion by taking the square root of nn.MSELoss(). I also used Adam as my optimizer with a learning rate of 0.01 and a weight_decay value of 0.00001. For my five models, I built one without an activation function, one using ReLU, one using tanh, one using sigmoid, and one using ELU. I trained each of these models of the training data for 1,000 epochs and then found RMSE on the withheld test data. Lastly, I plotted RMSE across the five models to compare their performance.
    b. I again used 30 hidden layer neurons to avoid overfitting as well as to help lower the time needed to train the models. I used RMSE as my loss criterion so that it would be optimized over training to result in a lower test RMSE (and higher performance). I used Adam as my optimizer because it is known to have a fast computing time. Additionally, I found that Adam slightly lowered the test RMSE of my models when compared with stochastic gradient descent. I used the same learning rate as in Question 2 and a weight_decay of 0.0001 because, again, after testing several values, I found these hyperparameters to be sufficient. Lastly, I used different activation functions to determine whether or not RMSE depends on the activation function used.
    c. All five models achieved a RMSE around 6. However, RMSE varied across the activation functions used. The model without an activation function had the highest RMSE at 6.0479. This was followed by the model using ELU (5.9881), then the model using ReLU (5.9730), the model using sigmoid (5.9604), and lastly the model using tanh (5.9563).



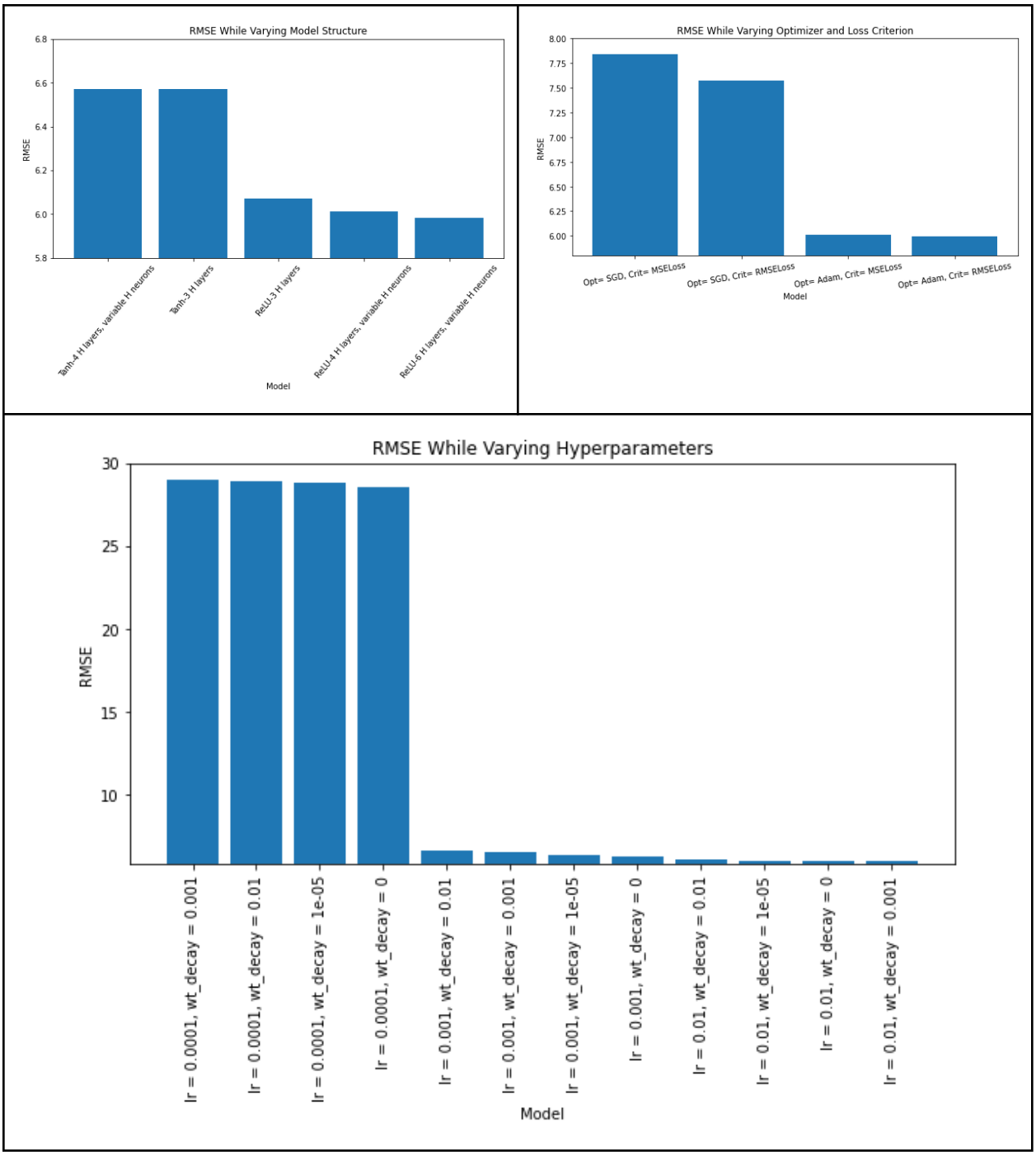
d. I found that RMSE did depend on the activation function used by my model when predicting BMI using a feedforward network with a single hidden layer. The optimal activation function as well as the lowest RMSE out of the five models depended upon several factors including the number of training epochs and the optimizer used (amongst other factors). It's therefore difficult to assess which activation function is best suited for a prediction task on this dataset. However, I found that tanh lowered the RMSE of the model the most conditioned on the hyperparameters, train/test split, criterion, and optimizer I chose. The RMSE of this model was 5.956. Additionally, the model with no activation function had the largest RMSE, which suggests that a simple linear model (i.e. linear regression) underperforms on this task relative to other models.

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?**
    a. In order to find a model with low RMSE, I first tested five feedforward networks with different activation functions, numbers of hidden layers, and numbers of neurons in each hidden layer. I trained each of these for 100 epochs as in Question 4 using RMSE Loss as the criterion and Adam as the optimizer with a learning rate of 0.01 and a weight decay of 0.00001. I then plotted RMSE across the models to determine which structure was optimal. Then, to assess the optimal hyperparameters of the model, I took the strongest model from my initial assessment and again trained it for 100 epochs, varying the learning rate and

weight decay of the Adam optimizer after each training. I tested a total of twelve combinations of learning rate and weight decay values, plotting RMSE across the model as these hyperparameters were changed. Lastly, I repeated the same procedure to test different criteria and optimizers. I tested MSELoss versus the square root of MSELoss as criteria and Adam and SGD as optimizers. I used the optimal model and hyperparameters from the previous tests to do this. I then trained my final model using the optimal factors for 1,000 epochs and calculated the RMSE on the test data for this model.

b. I tested various models using 100 epochs to see how design choices affected the model without having an inordinate runtime. I first tested the overall structure of models using the same baseline hyperparameters as the previous question because I wanted to ensure that the model's design was optimal before tuning other factors. I used both Tanh and ReLU activation functions because Tanh had the best performance in Question 4 and ReLU is a common choice as it is known to have good properties like preventing vanishing gradients. I tested various learning rates and weight decay values to examine the effect of each hyperparameter. Learning rate in particular may affect model performance as a learning rate that is too large could potentially miss an optimum. Lastly, I tested the optimizer Adam and SGD as both are commonly used and have their own pros and cons. Additionally, I wanted to compare the built-in MSELoss criterion to the criterion I used previously in my Question 4 models - which was computed by taking the square root of MSELoss. I then used all the optimal factors from these tests to train a model for 1,000 epochs, as longer training time typically results in better performance, and I wanted to try and get RMSE as low as possible for my final model.

c. The optimal model used six hidden layers with 60, 90, 120, 90, 60, and 30 neurons in the respective layers. ReLU activation function models outperformed Tanh models, so the final model also had ReLU functions between all layers. The optimal hyperparameter values for such a model was a learning rate of 0.01 and a weight decay value of 0.001. Learning rate drastically changed the RMSE of the models, with lower learning rates leading to higher RMSE. Additionally, the models using Adam as an optimizer outperformed models using SGD, and the final model used Adam with a RMSE Loss criterion. The RMSE of this final model was 5.9748.



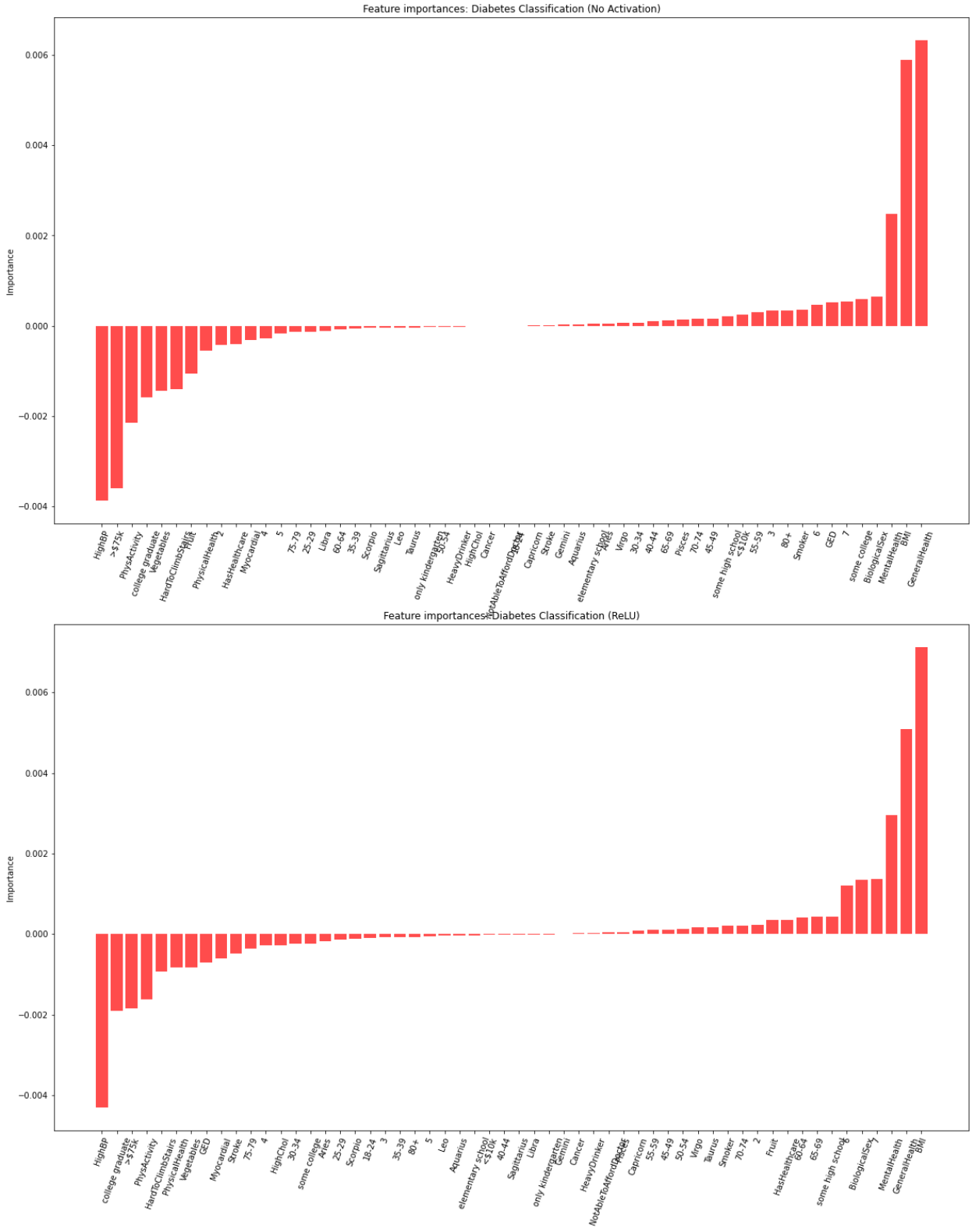
d. My final feedforward model had a RMSE of about 5.97. This is much higher than I would've liked, and is larger than some of the shallow feedforward models from Question 4 (which achieved a RMSE as low as 5.956). However, RMSE also depended on several design choices, most significantly the activation function used, the optimizer used, and the learning rate. In this case, ReLU models outperformed Tanh models regardless of the number of hidden layers or hidden layer neurons. For the ReLU models, adding extra neurons to the hidden layers

also dropped RMSE from a model with a fixed, lower number of hidden layer neurons (30 neurons per layer). Models using an Adam optimizer also outperformed models using SGD, and using RMSE Loss instead of MSE Loss slightly improved model performance across both optimizers. Lastly, model performance also improved as learning rate increased. This change is the most drastic, with models using a low rate of 0.0001 achieving a much larger RMSE (~28) than all other models. Lower weight decay values also tended to improve model performance, but this is less drastic. Overall, all design choices tested had some effect on model performance; however, learning rate, optimizer used, and activation function had the largest impacts.

Extra Credit:

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**
    a. To find if any features had no impact on the accuracy of the models, I calculated the feature importance of each feature in two of the models from Question 2 that classify diabetes. The first model had 1 hidden layer and no activation function, and the second used ReLU and 3 hidden layers. In order to find feature importance, I defined a function that found a baseline accuracy score using the unaltered test dataset. The function then randomly permuted the values of the feature and found accuracy using the permuted test dataset for n iterations. The importance of each feature was found by subtracting average of the permuted data accuracies from the baseline score. I iterated through each feature in both of the classification models and used the function with 50 iterations to get the feature's importance. I then plotted each model's feature importances sorted in ascending order to see which features were the most/least important.
    b. Firstly, I chose to examine two diabetes classification models as accuracy can only be computed for classification models (as opposed to the BMI regression models). I chose the model without an activation function and a ReLU model as they both had relatively high performances as measured by AUC. I found feature importance by permuting the values of a particular feature n times and taking the average accuracy because permuting the values of a feature essentially makes the feature less meaningful in the dataset by altering the correlations it may have with other data. Theoretically, if randomizing the values of a feature drops a model's performance, then that feature was important to the model's original performance. Therefore, seeing the change in accuracy from a baseline performance to the average of several trials with a random permutation of the feature helps determine the extent to which the feature affects the model's performance.

c. Both of the models had the same three most important features: BMI, GeneralHealth, and MentalHealth. Additionally, BiologicalSex had a positive importance in both models. Out of the 56 total features, the majority either had close to zero importance or a negative importance (suggesting that they drop model accuracy when included). Some of these unimportant features include the twelve Zodiac categories and the majority of the education, age, and income brackets. Additionally, PhysicalHealth, PhysicalActivity, Myocardial, HighBP, and HardToClimbStairs all had negative feature importance across the two models.



d. The vast majority of the 56 features had no positive effect on the models' performances. The features with effectively no impact on performance included the twelve Zodiac signs, some of the age bracket categories, the majority of the

income and education brackets, as well as a few other features like HeavyDrinker. Additionally, several features had negative feature importances - actively dropping model accuracy through their inclusion. These features included PhysicalHealth, PhysicalActivity, Myocardial, HighBP, and HardToClimbStairs as discussed above.

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 benefit of using neural networks is that with enough training time, data, and depth they can learn any function. This flexibility means that we can apply similar techniques to solve a variety of problems. For example, in previous assignments we used separate techniques for regression and classification tasks (with the exception of logistic regression which acted as a bridge between the two concepts). However, in this assignment I was able to build, train, and use MLP networks both for classifying diabetes and predicting BMI. Additionally, the models' performances were about the same as in the previous assignment using this dataset with classical methods (with an AUC around 0.8 for classification).

The overall drawback that I found using neural networks in this assignment is that there's more variability in performance based on how the models themselves are built. Previously, I used pre-built sklearn models and only needed to tune the hyperparameters. However, in this case, the models' performances were affected by a variety of factors including the number of hidden layers, neurons in each layer, activation function(s), optimizer, and loss criterion. Some of these, like the optimizer, came with additional hyperparameters. We also manually controlled the training time as opposed to calling a .fit() method. Therefore, it was much more difficult to determine how to optimize the model's performance as there were more factors to consider. Additionally, this dataset is relatively small, so there was already some variability based on the random seed and train/test split used - meaning that it was even more challenging to try and get optimal results.

In conclusion, neural networks have much more flexibility than classical methods. I can see their potential; however, as someone learning them for the first time, I also found that their flexibility came with the challenge of not always knowing what the optimal design choice was. It was therefore harder for me to build and implement these models than in the previous assignment.