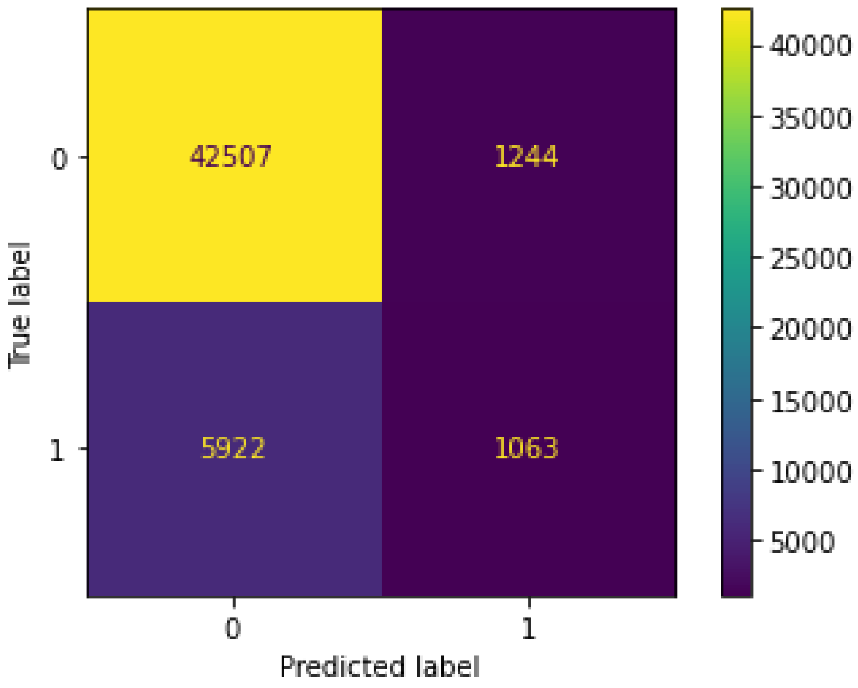
Homework #4

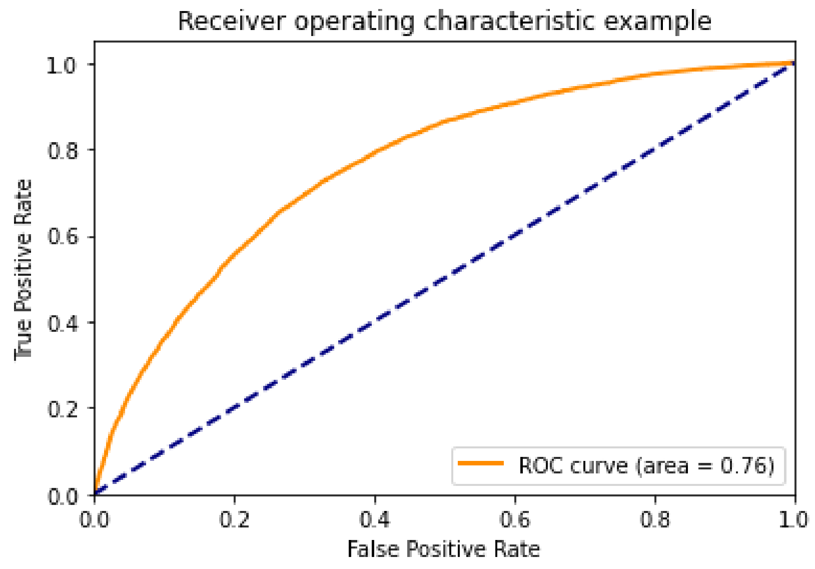
Alejandro Zaccour
az1715

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?

The first step I took to approach this task was to normalize the data. In order to do so I applied this formula over all the data points (x-dmin)/(dmax-dmin) which inherently gave me values between 0-1 with the same weights as before. To check that the data hadn't been incorrectly normalized I checked the correlation coefficients before the normalization and after and they were both the same. After this step I started building my model. First of all I split my data into a test set and a validation set. After this I went ahead and built plus trained a perceptron to see how accurately I could calculate if a patient has diabetes or not. After building the model I went ahead and did a confusion matrix plus graphed the AUC. Needless to say, the results were pretty good. I was able to capture almost 85% of the individuals who have diabetes, with very few false positives and false negatives. In addition the AUC ended up being quite good with a value of 0.76.

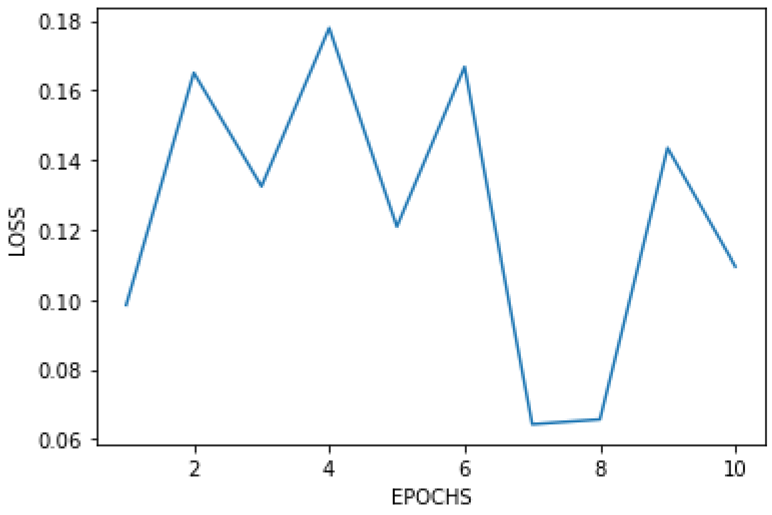


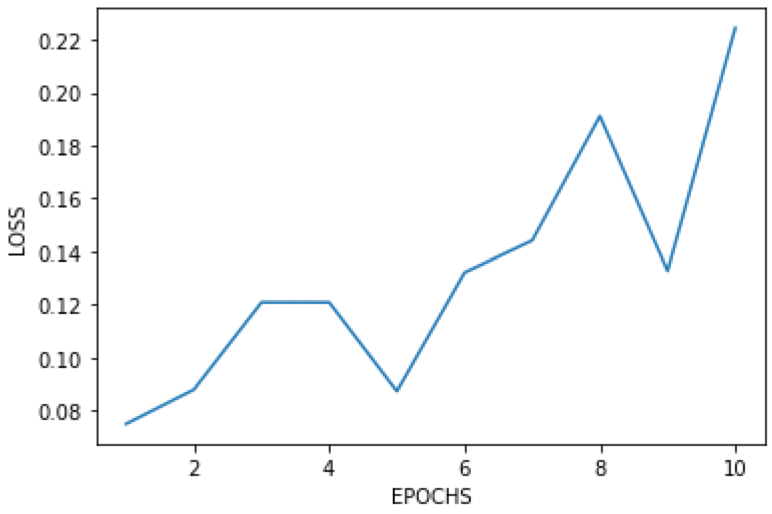
```
Accuracies:
(0.8587590665405235, 'Diabetes')
Confusion Matrix:
Precision:
(0.46077156480277415, 'Diabetes')
Recall:
(0.15218324982104509, 'Diabetes')
AUC:
(0.7621546934433912, 'Diabetes')
```

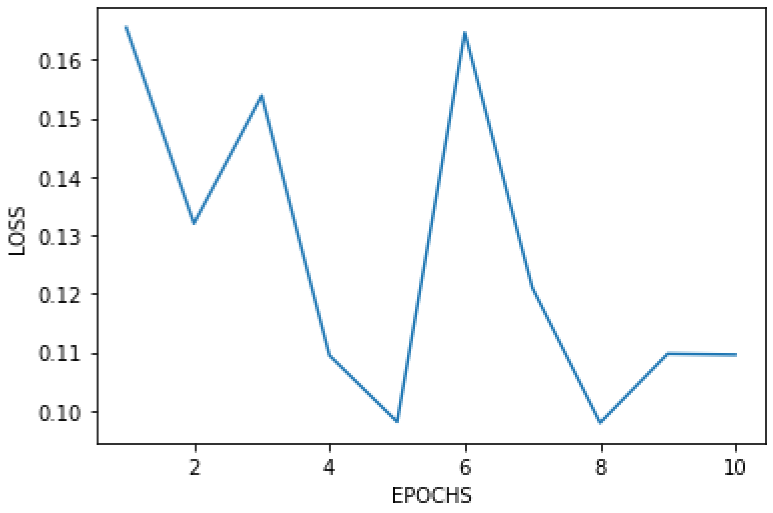


2. Build and train a feedforward neural network with at least one hidden layer to classify diabetes from the rest of the dataset. Make sure to try different numbers of hidden layers and different activation functions (at a minimum reLU and sigmoid). Doing so: How does AUC vary as a function of the number 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?

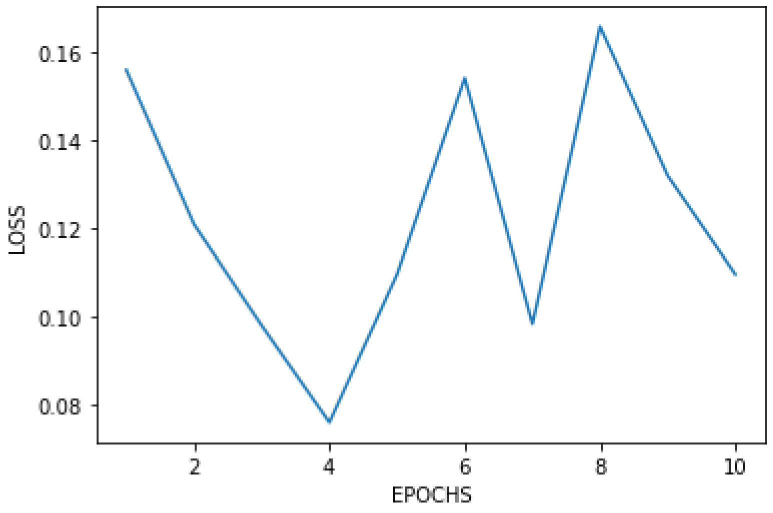
In order to build and train a feedforward neural network I approached the question with the same data division as in the last question. As I built the network I decided to try both ReLU and Sigmoid activation functions. In addition, I did my first network without an activation function to correctly compare them to each other. I did 11 networks, 1 with no activation function, 5 with ReLU, and 5 with Sigmoid. I did the networks in a loop so that I could change the number of hidden layers, with the variations being [2,5,10,20,50,2,5,10,20,50]. In the first five I used the ReLU function and in the other I did the Sigmoid with their respective hidden layers. In order to understand a little bit more of the data I decided to plot the loss to see which combination is the best (Graphs below).



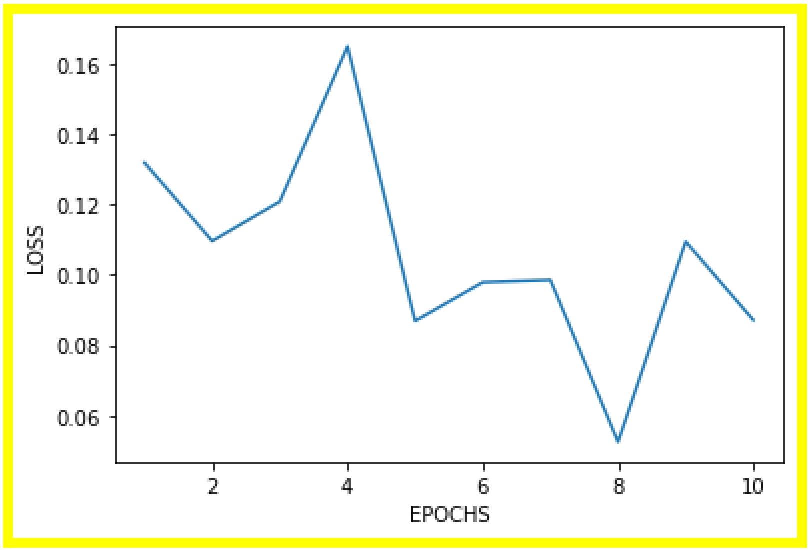




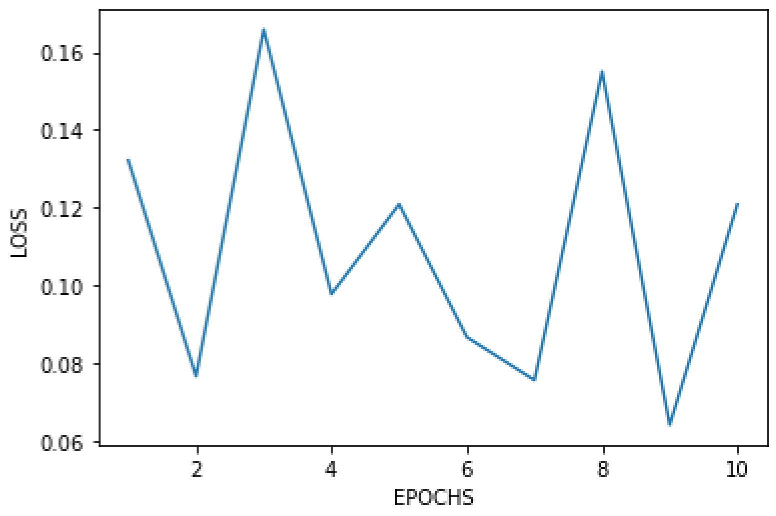
LOSS
EPOCHS
0.10
0.11
0.12
0.13
0.14
0.15
0.16
2
4
6
8
10

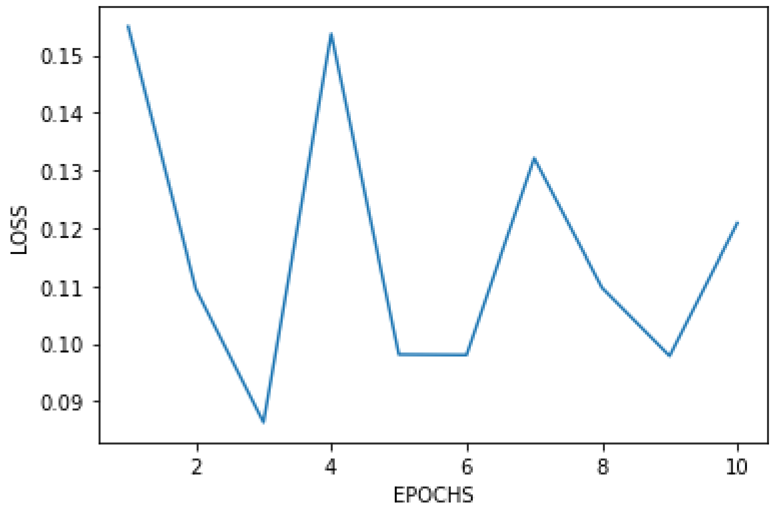


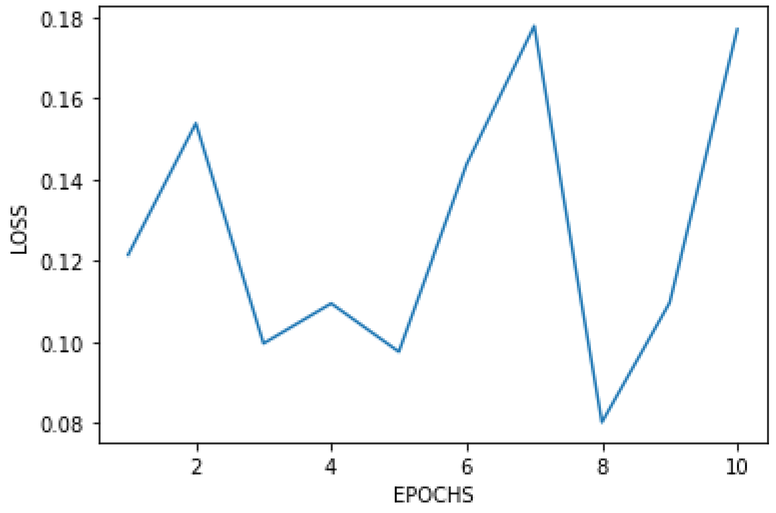
LOSS
EPOCHS
0.08
0.10
0.12
0.14
0.16
2
4
6
8
10

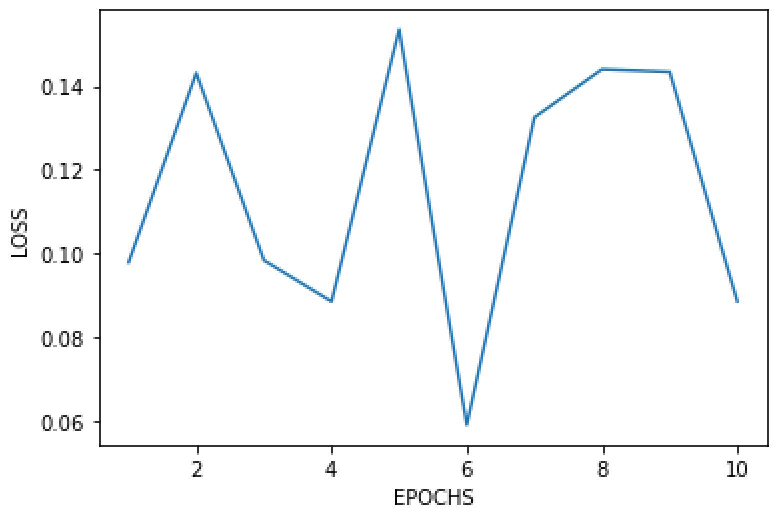


LOSS
EPOCHS
0.06
0.08
0.10
0.12
0.14
0.16
2
4
6
8
10

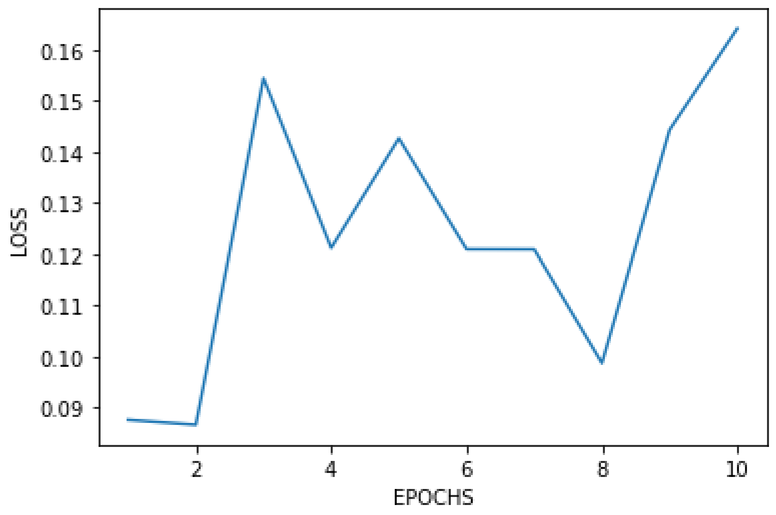

0.16
0.14
0.12
0.10
0.08
0.06
LOSS
2
4
6
8
10
EPOCHS


0.15
0.14
0.13
0.12
0.11
0.10
0.09
LOSS
2
4
6
8
10
EPOCHS

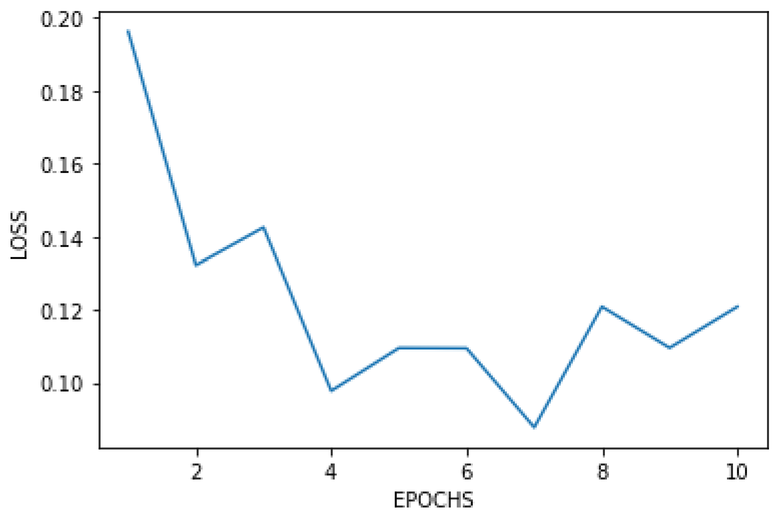

0.18
0.16
0.14
0.12
0.10
0.08
LOSS
2
4
6
8
10
EPOCHS



0.14
0.12
0.10
0.08
0.06
LOSS
2
4
6
8
10
EPOCHS

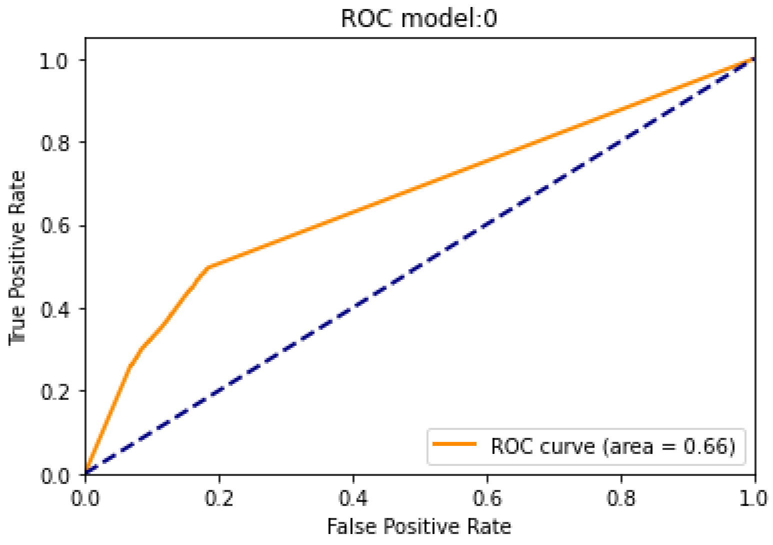


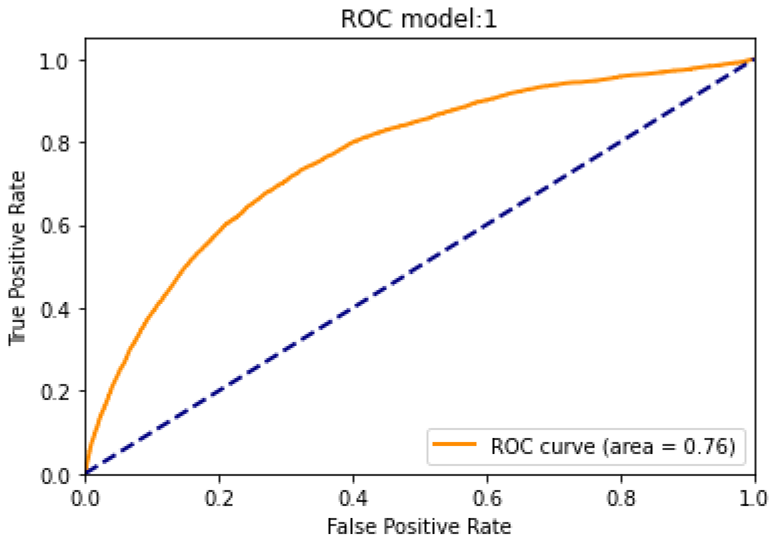
0.16
0.15
0.14
0.13
0.12
0.11
0.10
0.09
LOSS
2
4
6
8
10
EPOCHS

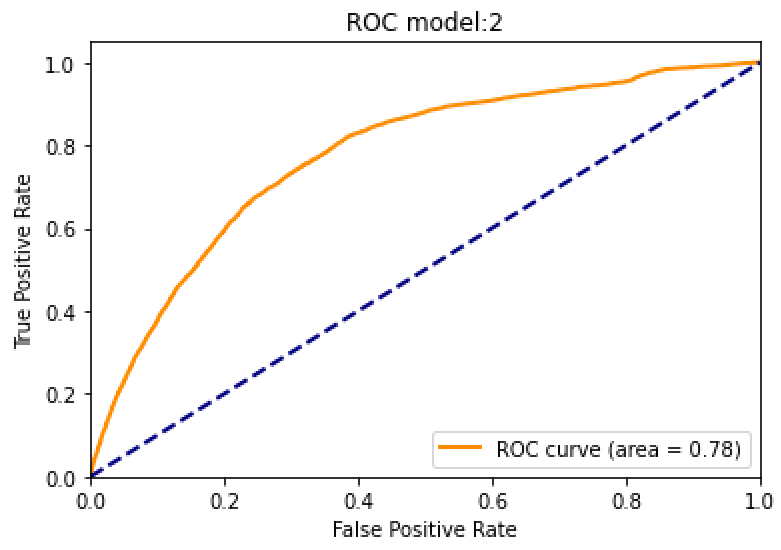


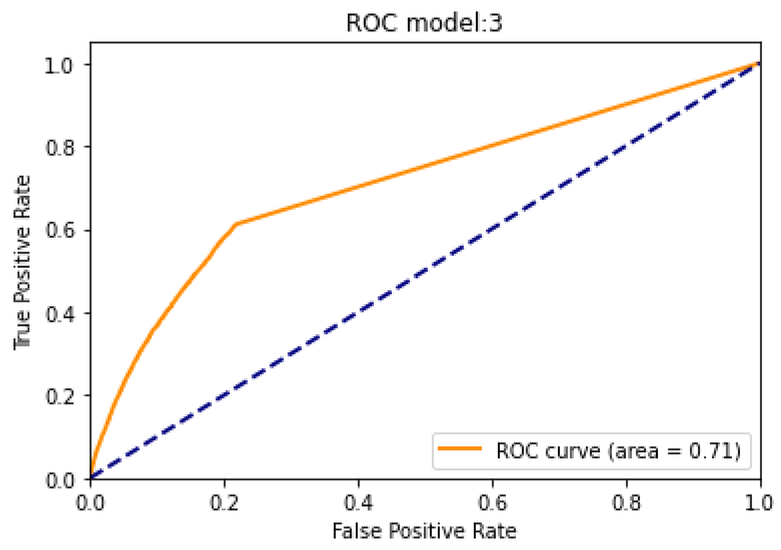
0.20
0.18
0.16
0.14
0.12
0.10
LOSS
2
4
6
8
10
EPOCHS

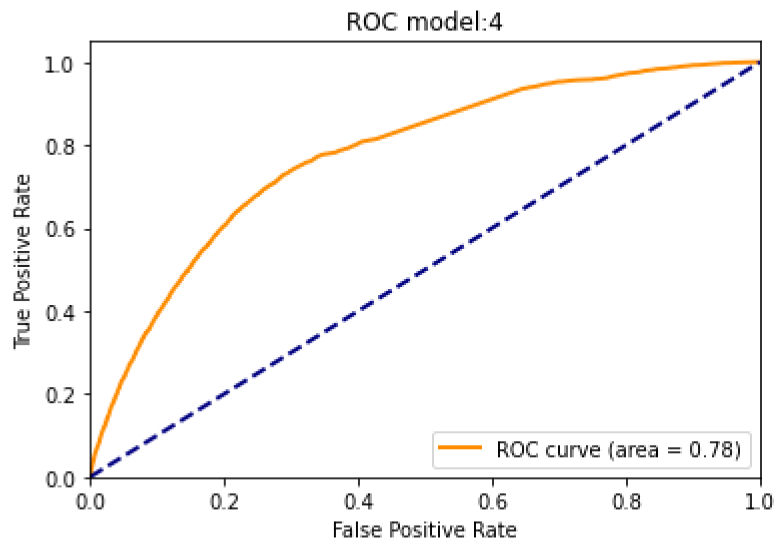
After finding which combination produced the least loss(highlighted) I was ready to calculate the AUC. I proceeded to calculate the same 11 combinations mentioned before and the best AUC I got was 0.79. Both the Sigmoid and ReLU gave me similar results but in general the ReLU performed constantly better. In addition, due to ReLU natural tangent properties it computes faster than Sigmoid due to it being a less complex function. Relative to the perceptron network this model works slightly better. With the perceptron being .76 and this one being .79.

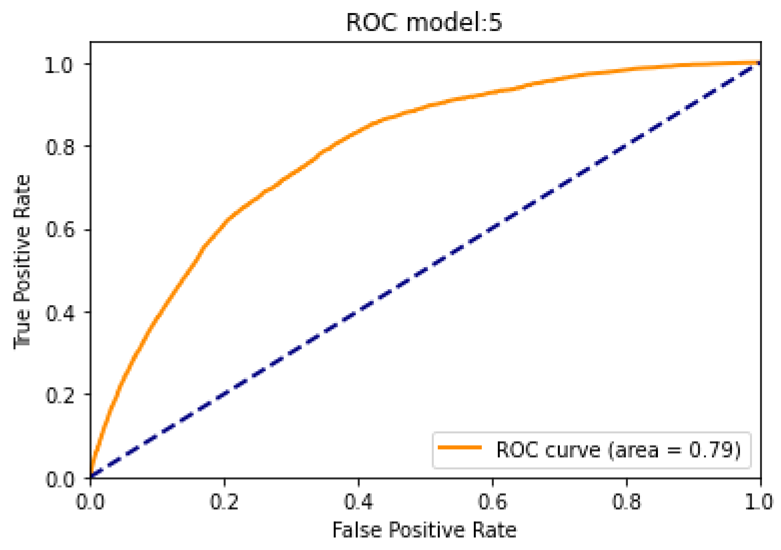


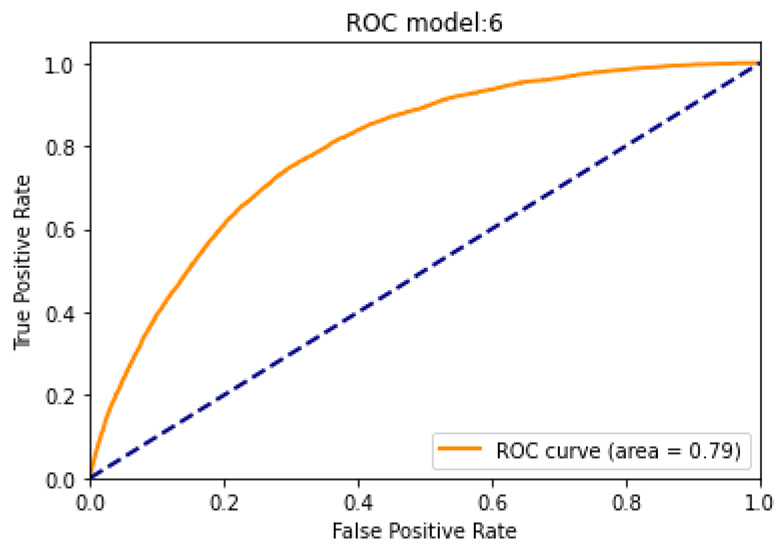


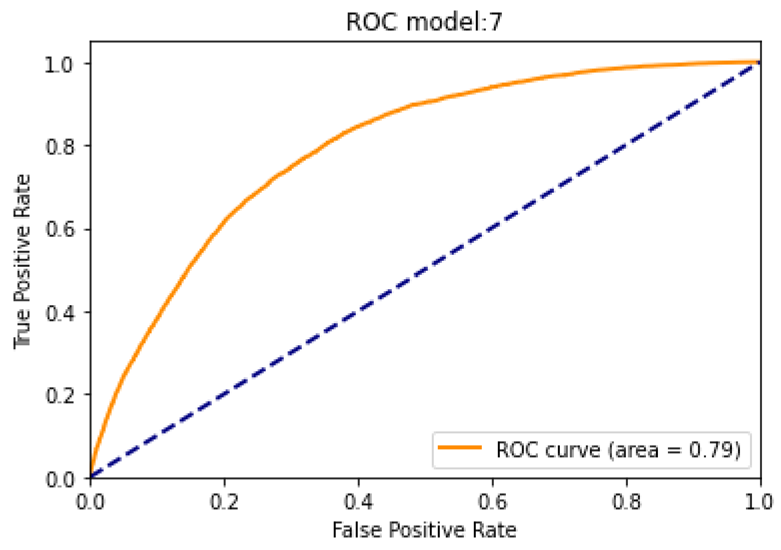

ROC model:2
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
ROC curve (area = 0.78)

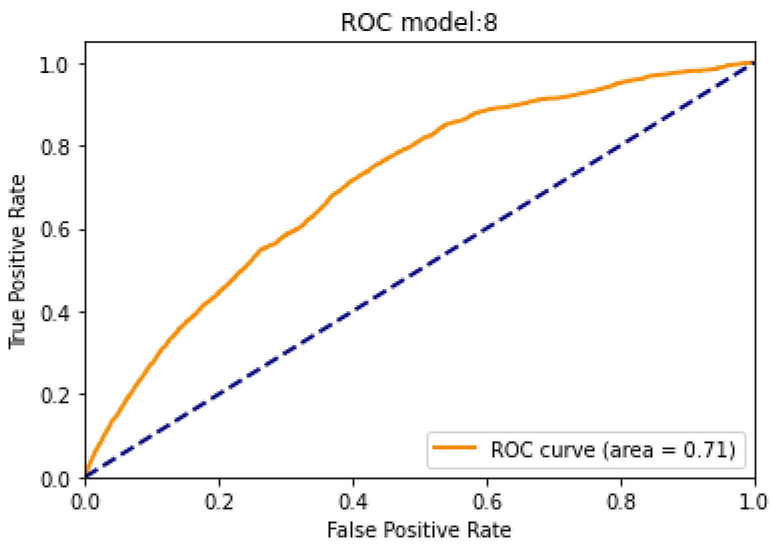

ROC model:3
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
ROC curve (area = 0.71)

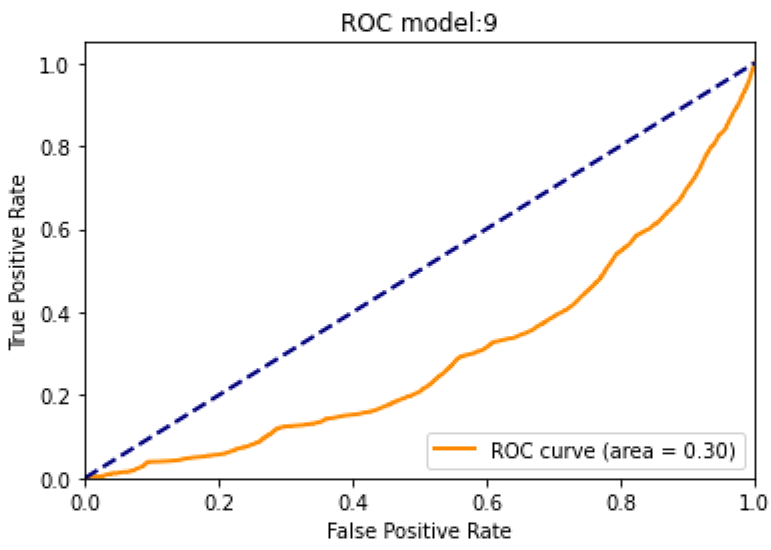

ROC model:4
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
ROC curve (area = 0.78)

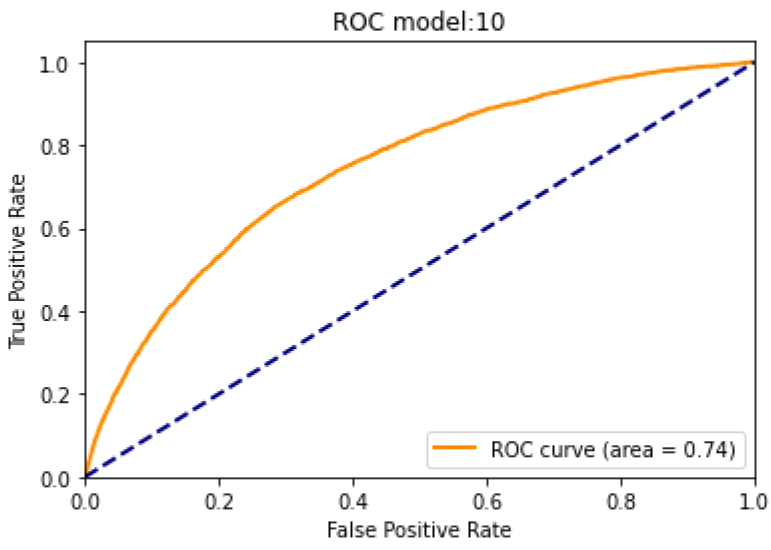

ROC model:5
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
ROC curve (area = 0.79)


ROC model:6
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
ROC curve (area = 0.79)


ROC model:7
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
ROC curve (area = 0.79)


ROC model:8
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
ROC curve (area = 0.71)


ROC model:9
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
ROC curve (area = 0.30)

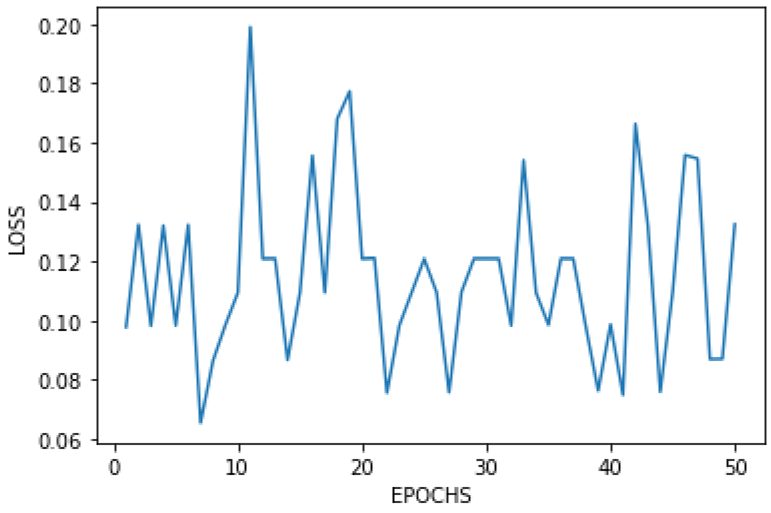

ROC model:10
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
ROC curve (area = 0.74)

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?

In order to answer this question I decided to use four hidden layers with ReLU and 5 hidden units. I concluded on these values after playing with them a bit to get the lowest possible loss. After playing a bit with these factors I came to the conclusion that this combination would bring the best results. In order to observe the results better I did a loop of 50 epochs to visualize where the training was moving the data. We can see in the graph below the results of this experiment, we can see that the more we train our data doesn’t necessarily correlate to a better result. In this case in epoch 7 we got the lowest loss signifying its the combination to get the best results. This further solidifies the idea that sometimes excessive training leads to overfitting. Given the data, using a CNN would not be very useful. CNNs are most commonly used to analyze spatial data(images) because of their natural property of comparison between data. For example if you are given a picture you can compare pixels that are close by to arrive at a conclusion but with this data set some columns have no correlation. RNNs will also not be very useful due that they rely on temporal data such as the speech recognition system Profesor Pascal uses in his lectures. It basically works on data that is sequential meaning that the last output will affect the upcoming one and in this data this is not the case.

```
[EPOCH]: 0, [LOSS]: 0.097803
[EPOCH]: 1, [LOSS]: 0.132254
[EPOCH]: 2, [LOSS]: 0.098203
[EPOCH]: 3, [LOSS]: 0.131959
[EPOCH]: 4, [LOSS]: 0.098321
[EPOCH]: 5, [LOSS]: 0.132217
[EPOCH]: 6, [LOSS]: 0.065430
[EPOCH]: 7, [LOSS]: 0.086455
[EPOCH]: 8, [LOSS]: 0.098577
[EPOCH]: 9, [LOSS]: 0.109471
[EPOCH]: 10, [LOSS]: 0.199076
[EPOCH]: 11, [LOSS]: 0.120852
[EPOCH]: 12, [LOSS]: 0.120870
[EPOCH]: 13, [LOSS]: 0.086674
[EPOCH]: 14, [LOSS]: 0.109489
[EPOCH]: 15, [LOSS]: 0.155610
[EPOCH]: 16, [LOSS]: 0.109483
[EPOCH]: 17, [LOSS]: 0.168044
[EPOCH]: 18, [LOSS]: 0.177345
[EPOCH]: 19, [LOSS]: 0.120852
[EPOCH]: 20, [LOSS]: 0.121014
[EPOCH]: 21, [LOSS]: 0.075580
[EPOCH]: 22, [LOSS]: 0.098374
[EPOCH]: 23, [LOSS]: 0.109623
[EPOCH]: 24, [LOSS]: 0.120850
```

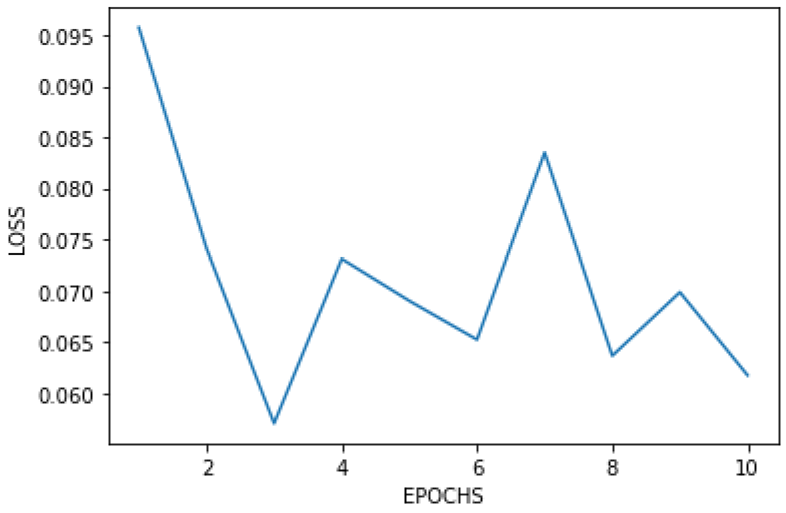
```
[EPOCH]: 25, [LOSS]: 0.109484
[EPOCH]: 26, [LOSS]: 0.075731
[EPOCH]: 27, [LOSS]: 0.109645
[EPOCH]: 28, [LOSS]: 0.120862
[EPOCH]: 29, [LOSS]: 0.120853
[EPOCH]: 30, [LOSS]: 0.120850
[EPOCH]: 31, [LOSS]: 0.098303
[EPOCH]: 32, [LOSS]: 0.154040
[EPOCH]: 33, [LOSS]: 0.109442
[EPOCH]: 34, [LOSS]: 0.098543
[EPOCH]: 35, [LOSS]: 0.120876
[EPOCH]: 36, [LOSS]: 0.120859
[EPOCH]: 37, [LOSS]: 0.098147
[EPOCH]: 38, [LOSS]: 0.076296
[EPOCH]: 39, [LOSS]: 0.098700
[EPOCH]: 40, [LOSS]: 0.074843
[EPOCH]: 41, [LOSS]: 0.166325
[EPOCH]: 42, [LOSS]: 0.132047
[EPOCH]: 43, [LOSS]: 0.075896
[EPOCH]: 44, [LOSS]: 0.109604
[EPOCH]: 45, [LOSS]: 0.155767
[EPOCH]: 46, [LOSS]: 0.154595
[EPOCH]: 47, [LOSS]: 0.086976
[EPOCH]: 48, [LOSS]: 0.087046
[EPOCH]: 49, [LOSS]: 0.132348
```

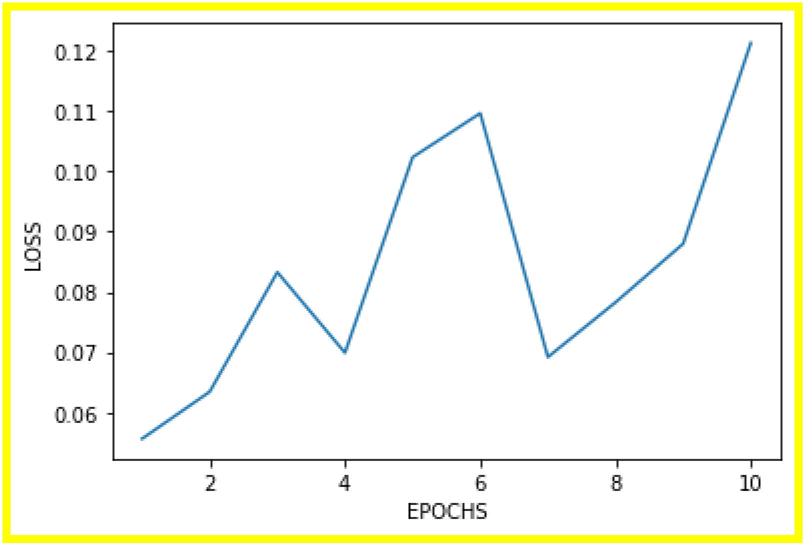


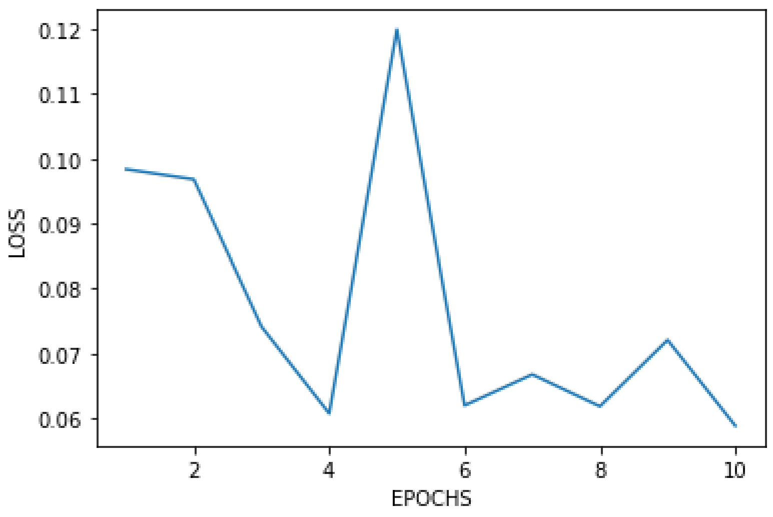
```
min loss 0.06543039530515671 in epoch 6
```

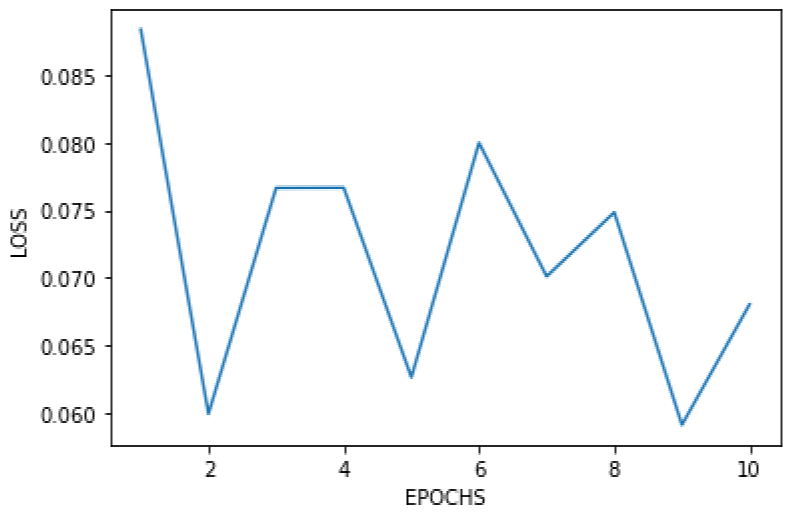
4. Build and train a feedforward neural network with one hidden layer to predict BMI from the rest of the dataset. Use RMSE to assess the accuracy of your model. Does the RMSE depend on the activation function used?

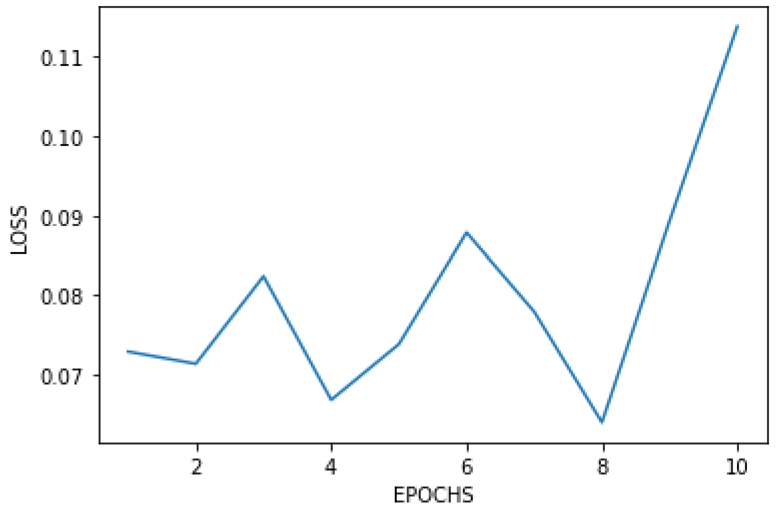
I approached this question very similar to question 2. I did 10 different combinations of hidden layers and activation functions. I did the networks in a loop so that I could change the number of hidden layers, with the variations being [2,5,10,20,50,2,5,10,20,50]. In the first five I used the ReLU function and in the other I did the Sigmoid with their respective hidden layers. After doing so I found the combination that gave me the smallest loss which was the second combination of ReLU with a loss of 0.0556. I tried both without RMSE and with RMSE and it turns out RMSE helps reduce loss. In this particular case rmse is less than 1 therefore the square root gives a bigger number. With this we observe a bigger penalization in the network which in theory lets the system learn more.

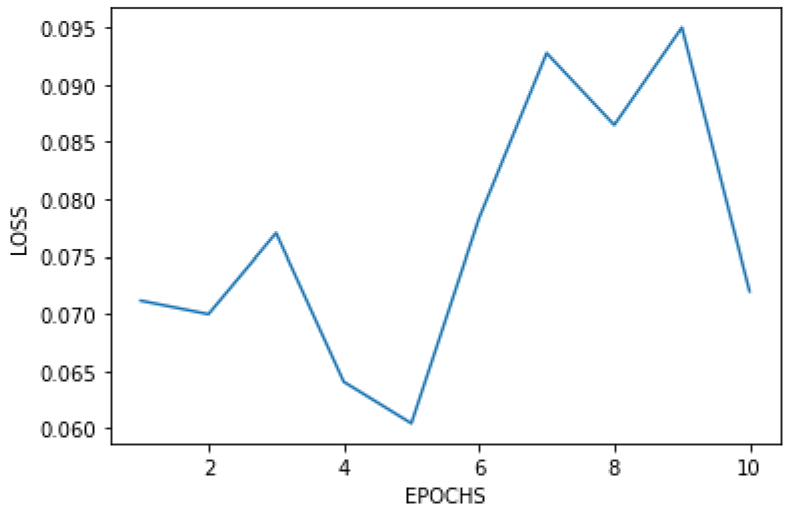


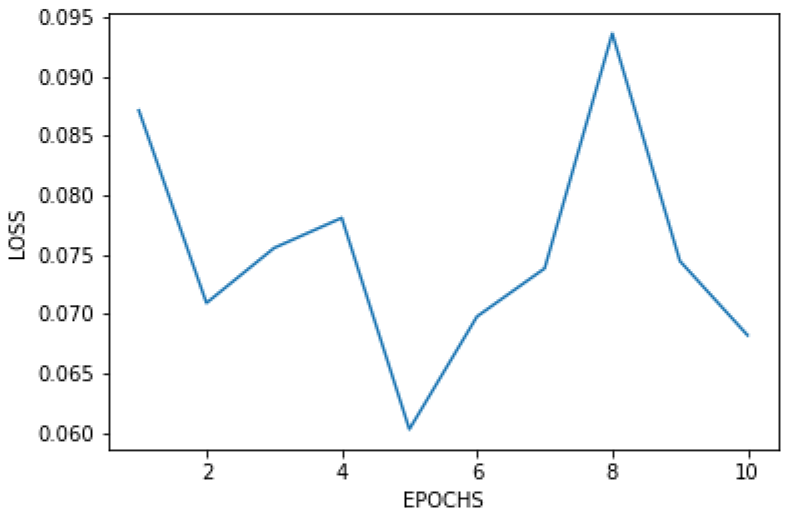

LOSS
EPOCHS
0.06
0.07
0.08
0.09
0.10
0.11
0.12
2
4
6
8
10

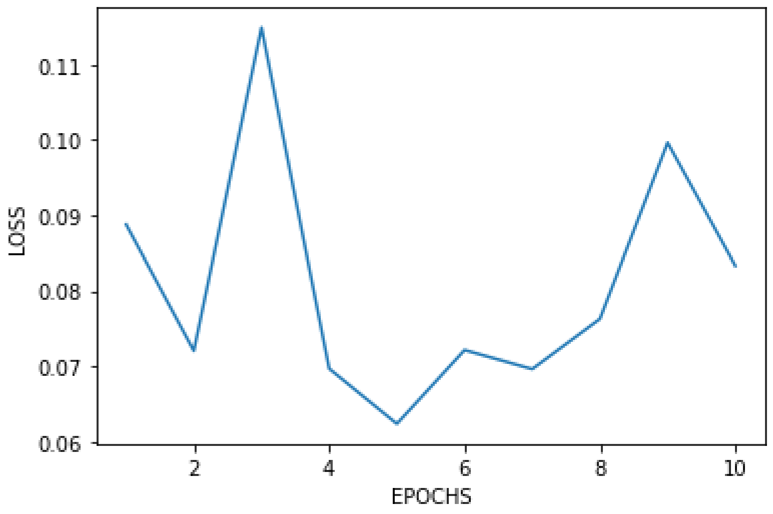

LOSS
EPOCHS
0.06
0.07
0.08
0.09
0.10
0.11
0.12
2
4
6
8
10

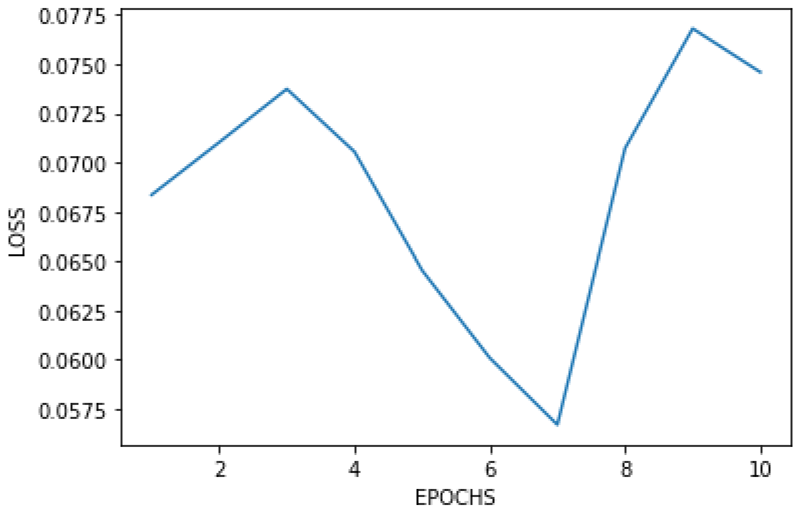

LOSS
EPOCHS
0.085
0.080
0.075
0.070
0.065
0.060
2
4
6
8
10

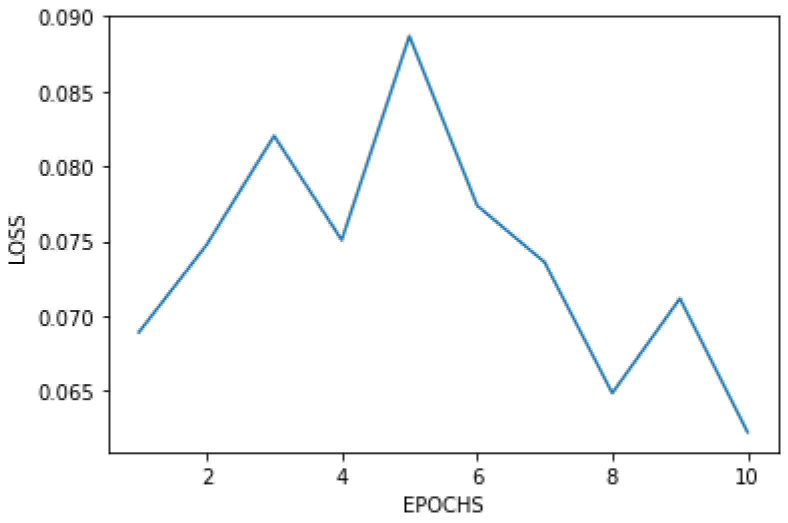

LOSS
EPOCHS
0.11
0.10
0.09
0.08
0.07
2
4
6
8
10


LOSS
EPOCHS
0.095
0.090
0.085
0.080
0.075
0.070
0.065
0.060
2
4
6
8
10


0.095
0.090
0.085
0.080
0.075
0.070
0.065
0.060
LOSS
2
4
6
8
10
EPOCHS

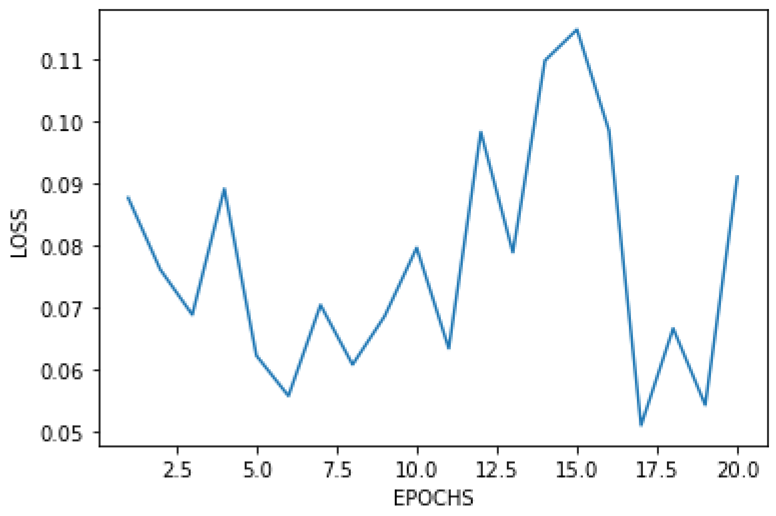

0.11
0.10
0.09
0.08
0.07
0.06
LOSS
2
4
6
8
10
EPOCHS


0.0775
0.0750
0.0725
0.0700
0.0675
0.0650
0.0625
0.0600
0.0575
LOSS
2
4
6
8
10
EPOCHS


0.090
0.085
0.080
0.075
0.070
0.065
LOSS
2
4
6
8
10
EPOCHS

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?

Based on the results of question 4 I created a loop of 20 iterations to see how low I could get the loss. We can see that in epoch 16 we get the best result with a loss of .051. We can use the values of the last question because we found out that ReLU was the best activation function to use plus we also use 4 hidden layers and 5 hidden units. From the results we can see that raining the data does reduce the error but training it a lot increases it as well. As per the graph below we can see all of these variations. In conclusion, training more doesn't necessarily mean you will get better results in the long run, sometimes the more you train the more you tend to overfit your data. In this case training the data 16 times was the best result but not always is this the case.



min loss 0.0510086826980114 in epoch 16

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.

Yes there are a lot. In the beginning of the assignment I did a correlation matrix to see how the predictors related to diabetes and found out that some were very near zero. PhysActivity, Fruit, Vegetables, HeavyDrinker, EducationBracket, IncomeBracket and Zodiac were all very close to zero and negative. This means that they will not be very efficient in impacting the accuracy in our models. I did take them out of the models sometime to see how much they affected them and it wasn't very much. I mainly did this in the first and second question and my AUCs varied by a minuscule amount signifying that these predictors helped little to nothing.

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?

First of all I learned that we are working with very very very little data. In order for neural networks to work at their best they need huge amounts of data. This comes with very big cons, the computational power to run these programs is way more than those of our older models. It took me way too long sometimes to run the models with question 2 sometimes taking me around 3-5 minutes to load. From the past assignment we can see that we got better AUCs and with way less computational power.