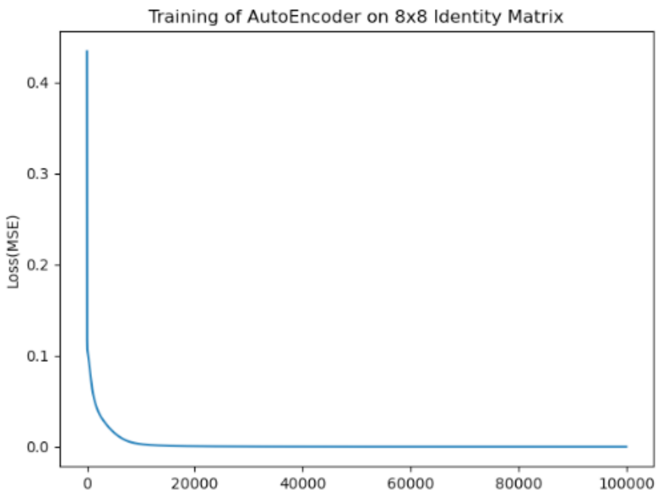
BMI 203 Project 3 Miguel Guardado Write Up

1) To confirm that your implementation functions correctly, demonstrate its ability to correctly solve the 8 x 3 x 8 autoencoder task. Specifically, set up an autoencoder network consisting of an input layer (8 nodes), a hidden layer (nodes), and an output layer (8 nodes), all with sigmoidal units.



I was successful in creating an autoencoder task, as well as creating a neural network such that you can specify any amount of layers you want for a task. To show I was able to run this network. I created an [8x3x8] neural network where the input layer is an 8x8 identify matrix and the predictor is the same row in the identity matrix, with sigmoid function determining the activation and final output layer decisions. There is one hidden layer of size 3, and an input and output later of 8. The graph on the right shows a figure of the loss per epoch as I trained my data though 100000 iterations. I used a learning rate of 0.1 and ran on a np.seed of 1, all of which can be inputted in my project under the main class. This graph resembles and asymptomatic curve, for which the loss of the network per epoch continuously gets closer and closer to zero. I can derive a limit of 0(thanks calculus!) and to show that the network creates a model of the identify matrix that has little error and is able to replicate the input matrix.

2) Describe your process of encoding your training DNA sequences into input vectors in detail. Include a description of how you think the representation might affect your network's predictions.

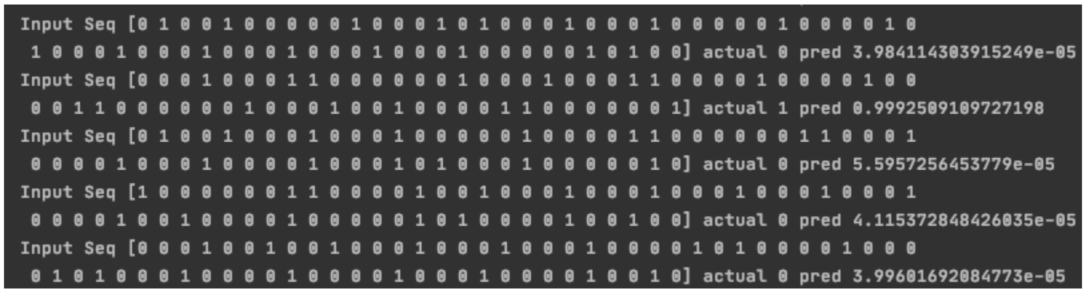
For encoding my DNA sequence into my neural network, I will implement one hot encoding, having my input layer for each base pair is encoded as a 4 index array of all zeros except one in the specific position of the base in. I will encode my training data via one hot encoding since machine algorithms cannot work with categorical data directly. For example, ATTAC =
[[1,0,0,0], [0,1,0,0], [0,1,0,0], [1,0,0,0],[0,0,0,1]]

3) Design a training regime that will use both positive and negative training data to train your predictive model. Describe your training regime. How was your training regime designed so as to prevent the negative training data from overwhelming the positive? training data?

For my training data of predicting transcription binding sites, I had a dataset of 137 17-mer positive controls, and 3163 1000bp negative controls to train. If we input our training data as is, we will have major overfitting for negative cases, so we need to limit the number of negative controls we feed into the neural network. This negative controls need to be converted from a 1000 bp regions to a 17 bp region, to accomplish this I created a 17-base subset of the 1000 base region and assigned that as the negative control. To control for the number of negative controls that are inputted, I will randomly subset the 3163 negative controls and input the first N number of input nodes to be trained on. N will be based on any number of sites that is desirable. For my initial analysis I simply used an equal amount of positive and negative controls (137 each) as a sanity check, I then gradually added more negative controls until all negative controls were made. I noticed that as I increased the number of negative samples, the model was quicker to asymptomatic convergence, which could lead to potential overfitting of the data, so for my later analysis I used 500 negative samples, and 137 positive sequences.

4) Modify your implementation to take as input positive and negative examples of Rap1 binding sites (using your encoding from Q2)

a. Provide an example of the input and output for one true positive sequence and one true negative sequence.



```
Input Seq [0 1 0 0 1 0 0 0 0 0 1 0 0 0 1 0 1 0 0 0 1 0 0 0 1 0 0 0 0 0 1 0 0 0 0 1 0
 1 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 0 0 1 0 1 0 0] actual 0 pred 3.984114303915249e-05
Input Seq [0 0 0 1 0 0 0 1 1 0 0 0 0 0 0 1 0 0 0 1 0 0 0 1 1 0 0 0 0 1 0 0 0 0 1 0 0
 0 0 1 1 0 0 0 0 0 0 1 0 0 0 1 0 0 1 0 0 0 0 1 1 0 0 0 0 0 0 1] actual 1 pred 0.9992509109727198
Input Seq [0 1 0 0 1 0 0 0 1 0 0 0 1 0 0 0 0 0 1 0 0 0 0 1 1 0 0 0 0 0 0 1 1 0 0 0 1
 0 0 0 0 1 0 0 0 1 0 0 0 0 1 0 0 0 1 0 1 0 0 0 1 0 0 0 0 0 1 0] actual 0 pred 5.5957256453779e-05
Input Seq [1 0 0 0 0 0 0 1 1 0 0 0 0 1 0 0 1 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 1
 0 0 0 0 1 0 0 1 0 0 0 0 1 0 0 0 0 0 1 0 1 0 0 0 0 1 0 0 1 0 0] actual 0 pred 4.115372848426035e-05
Input Seq [0 0 0 1 0 0 1 0 0 1 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 0 1 0 1 0 0 0 0 0 1 0 0 0
 0 1 0 1 0 0 0 1 0 0 0 0 1 0 0 0 0 1 0 0 0 1 0 0 0 0 1 0 0 1 0] actual 0 pred 3.99601692084773e-05
```

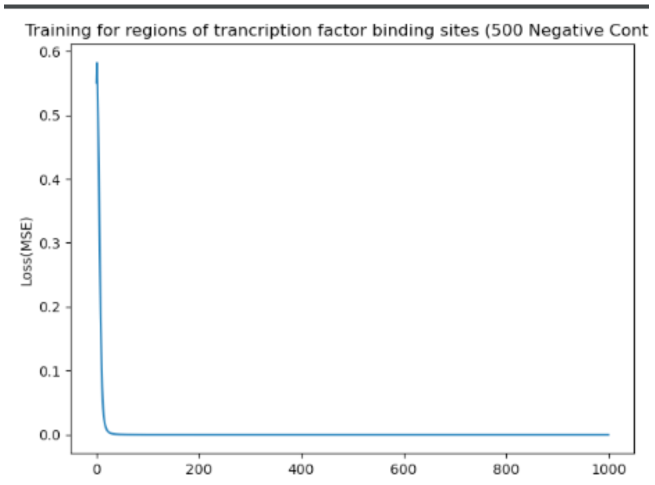
This is an example input and output of my training data, after the model was trained on, this should give some insight into how the model was preforming it's input and output layers in the neural network. In this example you see the 17-mer region one hot encoded, where the input layer is binary. And the output layer is a single value of the sigmoid activation function for the network. You can also observe that my model preforms really well on the test data, woo hoo!!!

b. Describe your network architecture, and the results of your training. How did your network performs in terms of minimizing error?

For my network architecture, I decided to implement a [67,17,4,1] layer neural network. This means that the input layer is a 67 one hot encoded 17-mer, we will have 2 layers of size 17 and 4, and an output layer of 1, which is all activated with a sigmoid function. The results of my training on this model did phenomenal, which makes me anxious. Plots shown below look at the loss per epoch of my training model, looking at the 1000 iterations of the system. With the cost function being a Mean Squared Error, this graph will show the average squared pairwise difference between the predicted and actual values for the training set. This plot shows really fast convergence (plot shown in part c), showing that this network performed well minimizing the error of the training data. This could lead to overfitting, so being able to test this data on a test dataset and look at the AUROC will give us a better picture of the model's prediction accuracy, which was shown to be 0.99877, meaning it predicted the test data near-perfectly, which can show overfitting on the data I used, and would need an external validation data set to test on.

c. What was your stop criterion for convergence in your learned parameters? How did you decide this?

For my criteria I used a loss function to look at the total loss per epoch. My model was learning the data very fast, with convergence found under 100 epochs. You will also need to consider that my neural network implementation utilized stochastic gradient descent to update my weights, which means the weights were updated after each observation passed. This means for a dataset of 300 training observation, 100 epochs of data resulted in 3000 weight updates, which can make more sense why the data was able to learn the data so fast. This led me to believe that my data was learning very accurately to determine if a 17kb region was a transcription binding site. In picking a stop criterion I chose to run only 1000 epoch of training samples, with a learning rate of 0.01, since it showed fast convergence.



5) Evaluate your model's classification performance via k-fold cross validation.

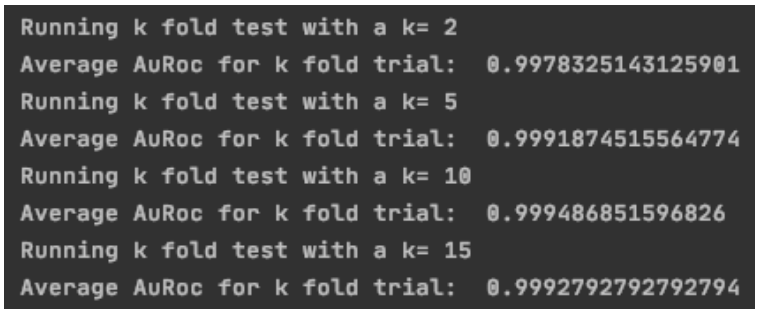
a.) How can you use k-fold cross validation to determine your model's performance?

K-fold cross validation is a resampling procedure that is used to evaluate a machine learnings model on an imbalanced training set, which can be perfectly utilized in out models training of only 647 samples. K fold validation requires only a single parameter, k, which will partition and split the data into the defined number of groups to test and train on. Usually, the first fold is used as a validation set, while the rest of the k-1 folds are used to train the model. The motivation to use this method on limited data is to limit the amount of data we input into the model, which will produce less biased estimates of the model, since all observations will get tested and trained on, rather than potentially overfitting with a train/test split.

b.) Given the size of your dataset, positive and negative examples, how would you select a value for k?

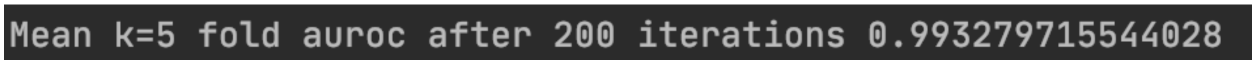
Choosing the value of k is very crucial in the model's performance, which can result in a model with high bias or variance. There is no formal rule to choose the best k from a study, with most people starting with a k=10 and adjusting from there depending on how well the model will do. The general intuition is that as we increase the number of k's, we will get less bias towards estimating the true expected error, additionally you will get an accurate confidence interval. For my analysis I will test different values of k to determine which will get us the best fit, I will test 4 different values of k [2,5,10,15] to see which one will perform the best, via an auroc metric.

c.) Using the selected value of k, determine a relevant metric of performance for each fold. Describe how your model performed under cross validation.



```
Running k fold test with a k= 2
Average AuRoc for k fold trial:  0.9978325143125901
Running k fold test with a k= 5
Average AuRoc for k fold trial:  0.9991874515564774
Running k fold test with a k= 10
Average AuRoc for k fold trial:  0.999486851596826
Running k fold test with a k= 15
Average AuRoc for k fold trial:  0.9992792792792794
```

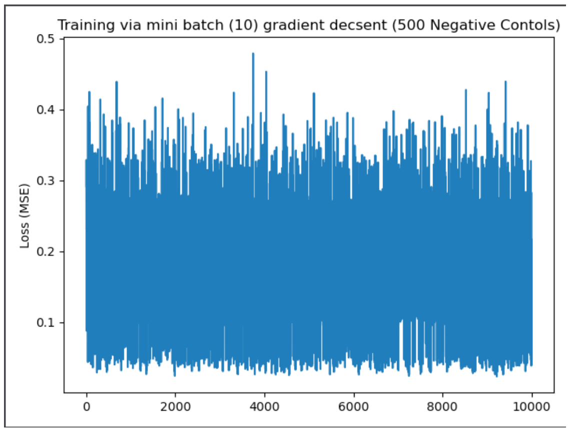
To determine the best k to use on cross fold validation, I ran 4 cross validation tests increasing the amount of k by a factor of 5. I added 3 negative samples into my prediction model so we could have an evenly balanced groups of data for testing and training. I ran this under a neural network with a 64 node input layer, 17 and 4 length input layer, with one output layer all activated via a sigmoid functions. Additional input parameters include a learning rate of 0.001, seed of 420, and running for 200 epochs, since the figure in 4c showed convergence around 200 iterations for a similar model. This graph showed that all different K performed very well. So, I will choose a K=5 to test, to save the number of tests that need to be done for a test in the next step. Now that we have a K to test, I will run 200-fold CV on out training data with a k=5, meaning I will run 200 test of a k=5 cross validation. We run additional test so we can create a distribution of roc scores to determine if our high auroc is just an artifact of the simulation or not. I additionally train the model on a different random seed for each iteration, for similar reason as the k-fold. The results of the mean of each test are shown below. After 200 iterations of testing the data, I was a able to achieve average auc score of .993, which is an amazingly high auroc curve, which shows that my was able to train and perform well on the entire training data alone. I should still be cautions that my model is overfitting my training dataset, but across multiple cross validation sets, as well as additional analysis from earlier sections, I will go forward with my model of the testing and training data.

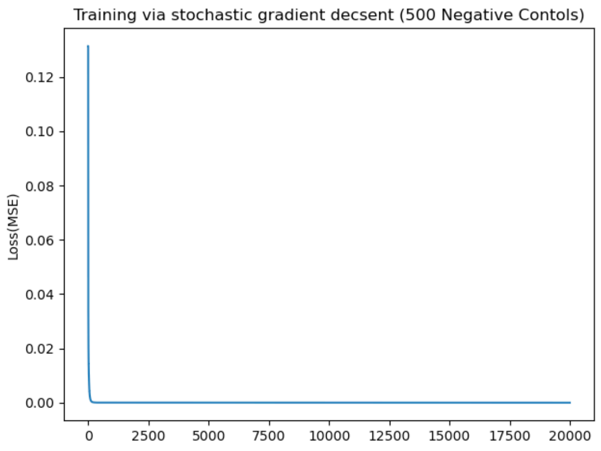


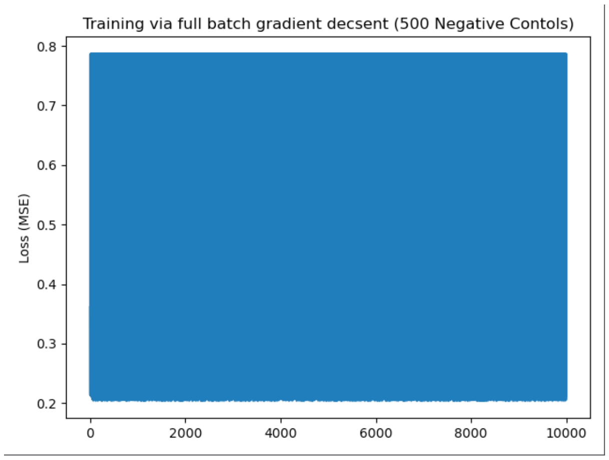
```
Mean k=5 fold auroc after 200 iterations 0.993279715544028
```

Part 4: Extension

For my extension, I chose to not seek an optimization method for my data since my neural network was able to train very well on the data based on the initial parameters I selected. Cross validation also confirmed this was not an artifact of the test/training data presented, at least in considering I only have the training data to assess the model on. I will instead, extend my network by making it more generalizable and reusable, which can in turn be used to assess more versions of neural networks. My network will be able to modify the version of gradient descent you want to run by either implementing a Batch, Mini Bach, or Stochastic Gradient Descent. This will be done by an extra parameter of “Batch”, “Mini Batch”, or “Stochastic” in my network initialization. Gradient Descent is used as a first order approximation algorithm for finding the minimum of a function, which in our case, finding the minimum loss of our neural network, based on the training dataset. We use this procedure to update the weights of layer to correct for the error, or loss, of the model. To give a brief overview of each of the 3 descent functions implemented. Stochastic gradient descent is used to update the weights one observation at a time per epoch. For example, if you were training a dataset of 500 observations, for one epoch you would need to update the weights 500 times based off the loss of each individual observation. This method is optimal when dealing with large dataset and updating one batch at a time can be too costly on the computer. Using a Full Batch Gradient descent will update the weights one epoch at a time based on the average loss of all the training’s predictions. This utilizes NumPy’s vector optimization by updating the all the observations at once in the feedforward and backpropagation function, thanks linear algebra!! It is great for convex or smooth error manifolds, converging at a faster and more computationally efficient in smaller datasets. Mini Batch is as Hannah Montana would say, the best of both worlds[1], since SGD can be used when datasets are large, but BGD converges better to the minima, mini batch gradient descent is used to combine both of the strengths of gradient descent. In this method, we won’t input the whole batch at once but rather subset of the observations, updating our weights parameters quickly while also being able to utilize vectorized implementations for faster computations. This means we can avoid the issue of underfitting based of the errors of the entire dataset compared to mean error of a smaller subset of the training data. In the case of our model for evaluating transcription binding motifs, I found that training the model for stochastic gradient descent performed better on the testing dataset, but minibatch performed well too, needing much more time to converge. For the remainder of my analysis, I chose to use stochastic gradient descent for the remainder of the analysis. And showed to train the model much and much faster. Unsure why this is but it is interesting to observe.

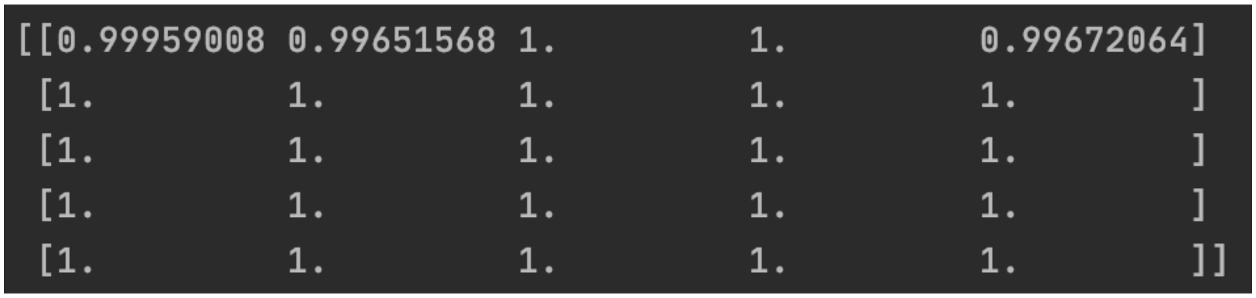






I additionally modified the network such that we can have the inclusion of more activation functions for each of the hidden and output layer, in case different activation functions will fit our predictions better. I will implement only two different activation functions, the sigmoid function as well the relu.

In addition to making my neural network more flexible, I will implement a basic grid search of different learning and hidden layers to use. I will again use AUROC as the dataset matric to test and train on. I will look at 5 different learning rates [0.01,0.1,0.2,0.5,1], I will test different network architectures, looking at 5 different architectures [[50,17,1],[17,4,1],[10,1],[4,1]] . This table below looks at the results for the grid search, with the rows representing the individual layers, and columns representing the different learning rates. The results show that all the different architectures give us a good auroc, which means it does a good job preforming on the testing data. To stay true to ol reliable, I will continue with my current model of having two hidden layers of 17 and 4 and a learning rate of 0.01.


```
[[0.99959008 0.99651568 1.         1.         0.99672064]
 [1.         1.         1.         1.         1.        ]
 [1.         1.         1.         1.         1.        ]
 [1.         1.         1.         1.         1.        ]
 [1.         1.         1.         1.         1.        ]]
```

5) Evaluate you network on the final set.

For my final model to use for analysis, I will combine all the results I found though the duration of the project. I will run a network with an input layer of 68 nodes to represent a 17 base one hot encoded sequence, and two hidden layers of 17 and 4, and an output later of 1, to predict if the 17-mer sequence inputted is a transcriptional binding site. Some additional hyper parameters I will use is having a learning rate of 0.01, I will run the model for 1000 epochs to assure convergence, finally to update the weights I utilized stochastic gradient descent. After testing this on the entire training dataset I outputted the results in the /data/ folder of my project with the name of rap1-lieb-predictions.txt which will display a text file where the first column represents the sequence and the 2nd column representing the prediction. The prediction is made such that, 1 represents a transcriptional binding site and 0 represents a non-transcriptional binding site.

References

1. Montana, Hannah. “Hannah Montana - The Best Of Both Worlds.” *YouTube*, YouTube/DisneyVevo, 16 Nov. 2010, www.youtube.com/watch?v=uVjRe8QXFHY .