GabeReder_BMI203_FinalProject

March 24, 2017

Gabe Reder - BMI 203 - 3/24/17

```
In [1]: import pickle as pkl
        import matplotlib.pyplot as plt
```

(1) Neural Network Implementation The auto-encoder problem is implemented in the function train_autoencoder with arguments (8,3,8) in the file neural_net.py. An example call is shown in the **main**.py file.

This is generalized to DNA sequences by encoding each sequence with the following scheme. Each nucleotide in a sequence string is converted to a 4-bit number with three 0's and one 1 according to the following mapping:

A: 1000
G: 0100
T: 0010
C: 0001

This takes a DNA sequence of length 17 and turns it into a vector of length 17*4 = 68. This 68-bit vector can then be used on a neural network with input layer of size 68. The DNA neural network is implemented using the same code as the autoencoder with a 68x20x1 architecture.

(2/3) Learning Procedure and Training on Negative Data I created a 68x20x1 neural network to accept DNA input in the form described above. I then train this network on all the positive sequences in the file rap1-lieb-positives.txt and a subset of the negative sequences in the file yeast-upstream-1k-negative.fa. For every 1000bp negative sequence, I look at every 17-bp substring. If this 17-bp substring is one of the positive sequences, I do not include any 17-bp substring from this negative sequence in the negative training set. If the 1000bp negative sequence does not contain any 17-bp substring that is a positive sequence I include it in a master set of negative sequences. Once I have the master set of negative sequences, I take a random subset of the master set such that the final ratio of positive to negative training points is 10:1. This is to avoid over training on negative data since there are so many more negative data points.

(4) Cross Validation and Changing Parameters Shown below are the results of three different testing methods I used for changing parameters of the neural network. I ran cross validation by aggregating all the training data in the manner described above then partitioning the master set of training data into 5 smaller subsets. I trained a neural network for each of these 5 subsets and then tested the trained network on the combined other 4 sets. In the case of cross validation, I was hoping to see very little fluctuation in the network response between the different training sets. As a rudimentary measure, I used the ratio of the number of guessed positives from the network to the number of actual positives in the validation set and the same ratio for negative binding sites. While I was happy to see little variance between networks, I realized that this was because the network converges to uniform output values for all inputs as described below in (5).

I wrote functions to vary the hidden layer size and number of batch gradient descent iterations to test the corresponding network behavior. The same metrics are shown as for cross validation though the results once again reflect the network's convergence to uniform output values.

```
In [2]: # Print results for cross-validation
        results_0 = pkl.load(open('results_training_0.pkl', 'rb'))
        results_1 = pkl.load(open('results_training_1.pkl', 'rb'))
        results_2 = pkl.load(open('results_training_2.pkl', 'rb'))
        results_3 = pkl.load(open('results_training_3.pkl', 'rb'))
        results_4 = pkl.load(open('results_training_4.pkl', 'rb'))
        results = [results_0, results_1, results_2, results_3, results_4]
        for i in range(5):
            num_positives, num_negatives, guess_positives, guess_negatives = result
            positive_ratio = guess_positives / num_positives
            negative_ratio = guess_negatives / num_negatives
            print('--------Training Set ' + str(i) + '----------------------------
            print(positive_ratio, negative_ratio)
            print('--------------------------------------------------')
```

```
--------Training Set 0----------------------------
0.0 1.103290676416819
--------------------------------------------------
--------Training Set 1----------------------------
0.0 1.103290676416819
--------------------------------------------------
--------Training Set 2----------------------------
0.0 1.1012773722627738
--------------------------------------------------
--------Training Set 3----------------------------
0.0 1.0982711555959963
--------------------------------------------------
--------Training Set 4----------------------------
0.0 1.0938924339106655
--------------------------------------------------
```

```
In [15]: # Print results for varying hidden layer size
         results_layer_size_1 = pkl.load(open('results_layer_size_1.pkl', 'rb'))
         results_layer_size_5 = pkl.load(open('results_layer_size_5.pkl', 'rb'))
         results_layer_size_10 = pkl.load(open('results_layer_size_10.pkl', 'rb'))
         results_layer_size_20 = pkl.load(open('results_layer_size_20.pkl', 'rb'))
         results_layer_size_50 = pkl.load(open('results_layer_size_50.pkl', 'rb'))
         results_layer_size_100 = pkl.load(open('results_layer_size_100.pkl', 'rb')
         results = [results_layer_size_1, results_layer_size_5, results_layer_size_
         sizes = [1,5,10,20,50,100]
```

```
        for i in range(6):
            num_positives, num_negatives, guess_positives, guess_negatives = resul
        #     positive_ratio = guess_positives / num_positives
        #     negative_ratio = guess_negatives / num_negatives
            print('--------Hidden Layer Size ' + str(sizes[i]) + '---------------
            print(positive_ratio, negative_ratio)
            print('----------------------------------------------------')
```

```
--------Hidden Layer Size 1----------------------------
0.0 1.0938924339106655
----------------------------------------------------
--------Hidden Layer Size 5----------------------------
0.0 1.0938924339106655
----------------------------------------------------
--------Hidden Layer Size 10----------------------------
0.0 1.0938924339106655
----------------------------------------------------
--------Hidden Layer Size 20----------------------------
0.0 1.0938924339106655
----------------------------------------------------
--------Hidden Layer Size 50----------------------------
0.0 1.0938924339106655
----------------------------------------------------
--------Hidden Layer Size 100----------------------------
0.0 1.0938924339106655
----------------------------------------------------
```

```
In [14]: # Print results for varying number of iterations
         results_iterations_1 = pkl.load(open('results_iterations_1.pkl', 'rb'))
         results_iterations_10 = pkl.load(open('results_iterations_10.pkl', 'rb'))
         results_iterations_50 = pkl.load(open('results_iterations_50.pkl', 'rb'))
         results_iterations_100 = pkl.load(open('results_iterations_100.pkl', 'rb')
         results_iterations_1000 = pkl.load(open('results_iterations_1000.pkl', 'rb
         results = [results_iterations_1, results_iterations_10, results_iterations
         iterations = [1,10,50,100,1000]
         for i in range(5):
             num_positives, num_negatives, guess_positives, guess_negatives = resul
         #     positive_ratio = guess_positives / num_positives
         #     negative_ratio = guess_negatives / num_negatives
             print('--------Number Iterations ' + str(iterations[i]) + '----------
             print(positive_ratio, negative_ratio)
             print('-------------------------------------------------------')
```

```
--------Number Iterations 1----------------------------
0.0 1.0938924339106655
-------------------------------------------------------
--------Number Iterations 10----------------------------
```

```
0.0 1.0938924339106655
-------------------------------------------------------
--------Number Iterations 50----------------------------
0.0 1.0938924339106655
-------------------------------------------------------
--------Number Iterations 100----------------------------
0.0 1.0938924339106655
-------------------------------------------------------
--------Number Iterations 1000----------------------------
0.0 1.0938924339106655
-------------------------------------------------------
```

(5) Sequence Likelihood Predictions Sequence likelihood predictions can be found in the file likelihood_scores.tsv where each sequence is printed with its corresponding likelihood score seperated by a tab. Looking at the neural network output, it looks like all the scores have converged to the same value close to 0. This did not happen with my autoencoder on 8-bit numbers and I believe this is a product of the training scheme I used. By taking every substring of length 17 from the negative sequences, I may have included too much repetitive information from each negative sequence in the training data and in doing so may have washed out important features of the training set.

The final trained neural network (weights, biases, and activations) can be found in the pickle file DNA_net.pkl.

```
In [ ]:
```