Question 1

Implement 8x3x8 Neural network and show it can solve autoencoder problem.

In [1]:
```
from hw4 import neuralnetwork as NN
import numpy as np
import timeit
```

These are my inputs for the autoencoder:

Each row is a separate training example.

In [2]:
```
inputs = ['10000000','01000000','00100000','00010000','00001000','00000100'
,'00000010','00000001']
for item in inputs:
    print(NN.transform_string(item))
```

```
[ 1.  0.  0.  0.  0.  0.  0.  0.]
[ 0.  1.  0.  0.  0.  0.  0.  0.]
[ 0.  0.  1.  0.  0.  0.  0.  0.]
[ 0.  0.  0.  1.  0.  0.  0.  0.]
[ 0.  0.  0.  0.  1.  0.  0.  0.]
[ 0.  0.  0.  0.  0.  1.  0.  0.]
[ 0.  0.  0.  0.  0.  0.  1.  0.]
[ 0.  0.  0.  0.  0.  0.  0.  1.]
```

In [3]:
```
######start time
start = timeit.default_timer()
#####
model = NN.NeuralNet(8,3,8,2.0,0.01)
iterations = 800
for i in range(iterations):
    model.clear_deltas()
    for item in inputs:
        model.prop_forward(NN.transform_string(item),NN.transform_string(it
em))
        model.prop_backward()
#######stop time
stop = timeit.default_timer()
```

Here is the runtime for 800 iterations which solves the autoencoder problem.

It runs in under one second to train this NN so code seems reasonably implemented.

In [4]:
```
print('Time: ', stop - start)
```

```
Time:  0.6838353159982944
```

Here are the outputs of the autoencoder model.

Clearly it is the same as the inputs, which is what we expect since we built this as an autoencoder (ie outputs = inputs).

In [5]:
```
for item in inputs:
    model.prop_forward(NN.transform_string(item),NN.transform_string(item))
    print(np.round(model.layer3_activation))
```

```
[ 1.  0.  0.  0.  0.  0.  0.  0.]
[ 0.  1.  0.  0.  0.  0.  0.  0.]
[ 0.  0.  1.  0.  0.  0.  0.  0.]
[ 0.  0.  0.  1.  0.  0.  0.  0.]
[ 0.  0.  0.  0.  1.  0.  0.  0.]
[ 0.  0.  0.  0.  0.  1.  0.  0.]
[ 0.  0.  0.  0.  0.  0.  1.  0.]
[ 0.  0.  0.  0.  0.  0.  0.  1.]
```

In [ ]:

2)

The architecture for my machine learning algorithm I am using is a 68X8X1 neural network.

I chose these numbers as follows:

68 = 17 base pairs * 4 (one hot encoding generates 4 nodes per 1 base pair because there are 4 categories)

8 as the hidden layer because I am reducing the number of features by a square root roughly to compress information.

1 as output layer because we want a prediction score of whether or not it is a true Rap1 binding site.

For my output layer I will use 1 as true positive Rap1 site, 0 as true negative Rap1 site.

The encoding for my machine learning algorithm is based on one hot encoding.

For each of the 17 bases, I have encoded each base into 4 bit, with a 1 for each bit indicating that it is true for that base.

For example, for base 1, A = 1 0 0 0

if base 2 were T, the encoding would then be 0 1 0 0

I then take each 4 bit chunk and concatenate them to get 17 * 4 = 68 bits of information which are used as input to the input nodes. That is why I have 68 input nodes.

3)

The number in the positive data set is 137 examples and low in number compared to the reduced set of negatives. Therefore, I added sequences to the positive data set by including reverse complements of the positive sequences. The total of positive training examples is now 274.

There was an overwhelming amount of negative data. I reasoned that if there were sequences in the negative data that closely matched the positive data, then it would slow down the learning process as well as decrease the resolution of the NN.

Therefore I chose to eliminate sequences from the negative data based on similarity to sequences in the positive data. I aligned all the sequences in the positive data and negative data, and filtered out those sequences in the negative data that had high alignment scores to any positive sequence.

This filtered the negative data from ~50,000 examples to ~600 examples.

Furthermore, I am only taking the first 17 base pairs of the negative examples. I could do a tiling approach in the future.

In this way I have increased the number of positive examples using the supplied information and decreased the amount of negative examples by reducing redundancy and increasing signal.

4)

I used the Accuracy definition used in ROC curves. This is calculated as follows:

(TP + TN) / (P + N)

The maximum value of this measurement is 1. Intuitively this measure defines the ratio of true values (pos and neg) versus called values (pos and negative. The closer it is to 1, the better the accuracy of the system is. This value is suitable for our purposes because it takes into account accuracy in both positive and negative directions.

I am using K-Fold cross validation to minimize the effect of bias and maximize accuracy. K fold validation works by splitting the total dataset into multiple chunks and then holding out one chunk at a time and doing iterative training on these k-1 chunks.

I optimized three main parameters using the accuracy metric:

Number of Iterations

Step size of the NN algorithm

Number of Chunks in K-fold cross validation

Based on the plot shown in Fig1, you can see that for most of the step sizes they converge (in terms of accuracy) by iteration eight. However, for the best step sizes, they converge early ~ iteration 5.

Therefore I will set iterations = 5 for testing the remaining parameters.

Based on the plot shown in Fig2, you can see that the best step size parameter was step size = 0.2

The best number of chunks for K-fold validation appears to be K = 7.

Shown in Printout A is sample output using iterations=5, step size=0.2, k=7

In [19]:
```
# Load libraries
import pandas as pd
import copy
import numpy as np
import timeit
import matplotlib.pyplot as plt
####
from Bio import pairwise2
from Bio.Seq import Seq
from Bio.Alphabet import generic_dna, generic_protein
####
from hw4 import neuralnetwork as NN
from hw4 import encoding as EN
```

Question 3. Positive and Negative Data.

In [2]:
```
####read in
pos = pd.read_csv('rap1-lieb-positives.txt',header=None)[0].tolist()
test = pd.read_csv('rap1-lieb-test.txt',header=None)[0].tolist()
neg = []
#####
f = open('yeast-upstream-1k-negative.fa')
#####
for line in f.readlines():
    if line[0] != '>' :
        neg.append(line.split('\n')[0])
#####
```

In [10]:
```
####build training set.
####Because there are much more positives than negatives, maximize the numb
er of positives in training data
####by also adding reverse complements of positive sequences
pos_total = copy.deepcopy(pos)
for seq in pos:
    my_dna = Seq(seq, generic_dna)
    rc = my_dna.reverse_complement()
    pos_total.append(str(rc))
```

In [12]:
```
####check for alignments between pos and negs; throw those out from negs if
there are any
max_score = pairwise2.align.localxx(pos[0],pos[0])
max_score = max_score[0][2]
cutoff = 0.9*max_score
####look for alignments between pos and neg
print('before:',len(neg))
i=0
for seq_pos in pos_total:
    for seq_neg in neg:
        if pairwise2.align.localxx(seq_pos,seq_neg)[0][2] > cutoff:
            ###this is if they do have significant overlap
            neg.remove(seq_neg)
print('after',len(neg))
```

```
before: 53744
after 603
```

In [13]:
```
####save filtered negatives
with open('negs_filtd.txt', 'w') as file:
    for seq in neg:
        file.write(seq)
        file.write('\n')
```

In [14]:
```
####save updated positives
with open('pos_filtd.txt', 'w') as file:
    for seq in pos_total:
        file.write(seq)
        file.write('\n')
```

Question 2. Encoding and NN architecture.

In [2]:
```
####read in filtered negatives
with open('negs_filtd.txt', 'r') as file:
    neg = file.read().splitlines()
####read in positives
with open('pos_filtd.txt', 'r') as file:
    pos = file.read().splitlines()
```

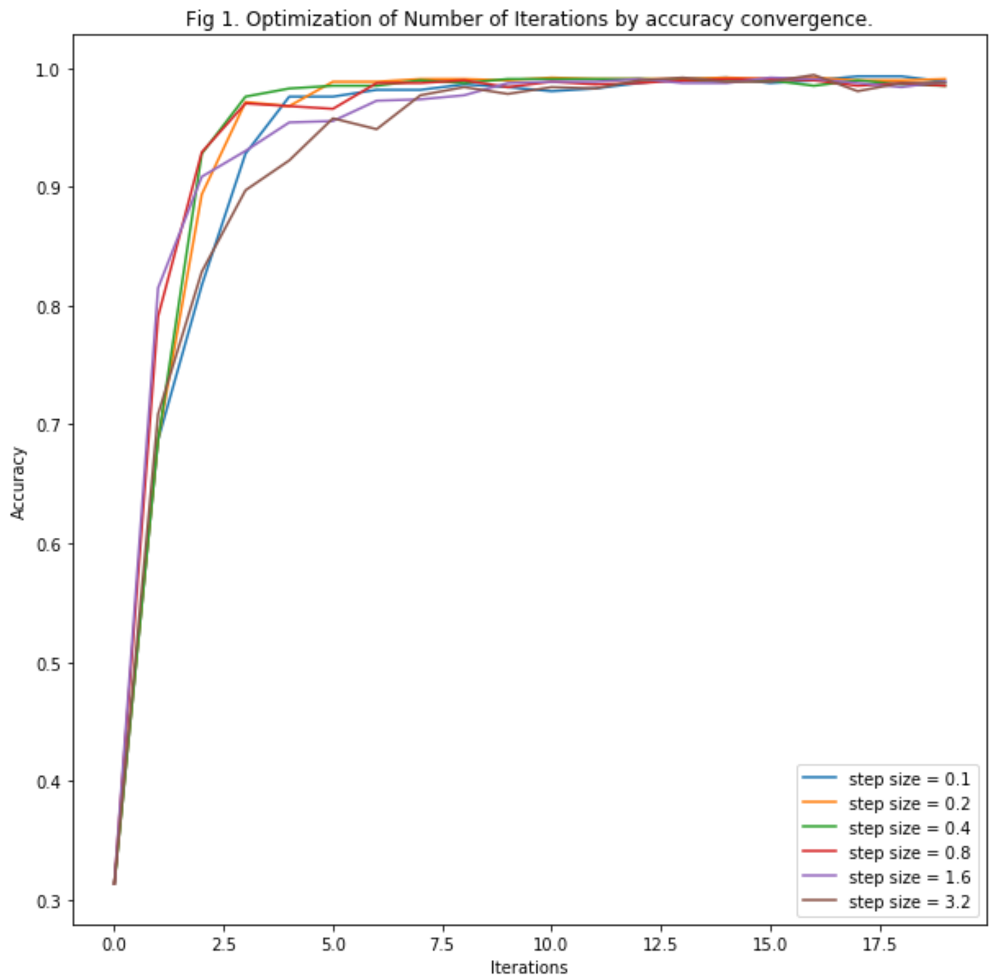
In [3]:
```
neg_encoded,pos_encoded = EN.encode_pos_neg(neg,pos,17)
```

Question 4. K-fold cross validation.

In [40]:
```
####define parameters
num_parts = 3
#####split up data
neg_chunks,pos_chunks = NN.chunk_kfold(num_parts,neg_encoded,pos_encoded)
#####format training data and get outputs
training_inputs,training_outputs = NN.format_trainingdata(neg_chunks,pos_ch
unks)
```

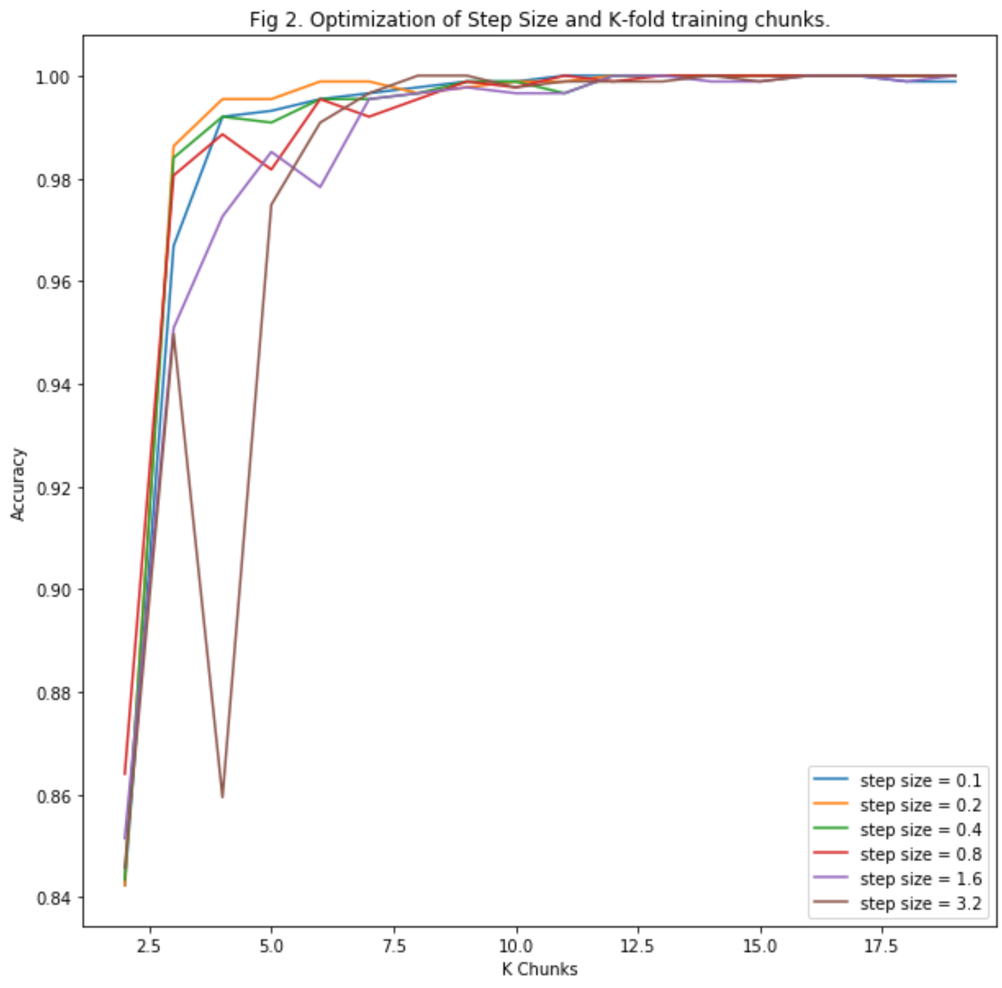
In [ ]:
```
accuracies = []
step_sizes = [0.1,0.2,0.4,0.8,1.6,3.2]
for step_size in step_sizes:
    tmp_accuracies = []
    for iterations in range(20):
        ######start time
        start = timeit.default_timer()
        #####setup model
        model = NN.NeuralNet(68,8,1,step_size,0.01)
        #####perform k-fold cross validation
        for i in range(len(training_inputs)):
            train_inputs = []
            train_outputs = []
            for j in range(len(training_inputs)):
                if i != j:
                    for item in training_inputs[j]:
                        train_inputs.append(item)
                    for item in training_outputs[j]:
                        train_outputs.append(item)
            #######
            model = NN.train_model(iterations,train_inputs,train_outputs,mo
del)
        ######
        total_inputs = []
        total_outputs = []
        for i in range(len(training_inputs)):
            for j in range(len(training_inputs[i])):
                total_inputs.append(training_inputs[i][j])
                total_outputs.append(training_outputs[i][j])
        pos_predicted,pos_true,neg_predicted,neg_true = NN.call_results(tot
al_inputs,total_outputs,model)
        accuracy = NN.evaluate_accuracy(pos_predicted,pos_true,neg_predicte
d,neg_true)
        tmp_accuracies.append(accuracy)
        #######stop time
        stop = timeit.default_timer()
        ####print time
        print('Time: ', stop - start)
    accuracies.append(tmp_accuracies)
```

In [42]:
```
i = 0
plt.figure(figsize=(10,10))
####
for entry in accuracies:
    printme = 'step size = '+str(step_sizes[i])
    #####
    plt.plot(range(20),entry,label=printme)
    ##
    i+=1
####
plt.legend(loc=0)
plt.ylabel('Accuracy')
plt.xlabel('Iterations')
plt.title('Fig 1. Optimization of Number of Iterations by accuracy converge
nce.')
plt.show()
```



In [ ]:
```
accuracies = []
step_sizes = [0.1,0.2,0.4,0.8,1.6,3.2]
for step_size in step_sizes:
    tmp_accuracies = []
    for num_parts in range(2,20):
        #####split up data
        neg_chunks,pos_chunks = NN.chunk_kfold(num_parts,neg_encoded,pos_en
coded)
        #####format training data and get outputs
        training_inputs,training_outputs = NN.format_trainingdata(neg_chunk
s,pos_chunks)
        ######
        iterations = 5
        ######start time
        start = timeit.default_timer()
        #####setup model
        model = NN.NeuralNet(68,8,1,step_size,0.01)
        #####perform k-fold cross validation
        for i in range(len(training_inputs)):
            train_inputs = []
            train_outputs = []
            for j in range(len(training_inputs)):
                if i != j:
                    for item in training_inputs[j]:
                        train_inputs.append(item)
                    for item in training_outputs[j]:
                        train_outputs.append(item)
            #######
            model = NN.train_model(iterations,train_inputs,train_outputs,mo
del)
        ######
        total_inputs = []
        total_outputs = []
        for i in range(len(training_inputs)):
            for j in range(len(training_inputs[i])):
                total_inputs.append(training_inputs[i][j])
                total_outputs.append(training_outputs[i][j])
        pos_predicted,pos_true,neg_predicted,neg_true = NN.call_results(tot
al_inputs,total_outputs,model)
        accuracy = NN.evaluate_accuracy(pos_predicted,pos_true,neg_predicte
d,neg_true)
        tmp_accuracies.append(accuracy)
        #######stop time
        stop = timeit.default_timer()
        ####print time
        print('Time: ', stop - start)
    accuracies.append(tmp_accuracies)
```

In [37]:
```
i = 0
plt.figure(figsize=(10,10))
####
for entry in accuracies:
    printme = 'step size = '+str(step_sizes[i])
    #####
    plt.plot(range(2,20),entry,label=printme)
    ##
    i+=1
####
plt.legend(loc=0)
plt.ylabel('Accuracy')
plt.xlabel('K Chunks')
plt.title('Fig 2. Optimization of Step Size and K-fold training chunks.')
plt.show()
```



In [43]:
```
####define parameters
num_parts = 7
#####split up data
neg_chunks,pos_chunks = NN.chunk_kfold(num_parts,neg_encoded,pos_encoded)
#####format training data and get outputs
training_inputs,training_outputs = NN.format_trainingdata(neg_chunks,pos_ch
unks)
######define parameters
step_size = 0.2
iterations = 5
######start time
start = timeit.default_timer()
#####setup model
model = NN.NeuralNet(68,8,1,step_size,0.01)
#####perform k-fold cross validation
for i in range(len(training_inputs)):
    train_inputs = []
    train_outputs = []
    for j in range(len(training_inputs)):
        if i != j:
            for item in training_inputs[j]:
                train_inputs.append(item)
            for item in training_outputs[j]:
                train_outputs.append(item)
    #######
    model = NN.train_model(iterations,train_inputs,train_outputs,model)
######
total_inputs = []
total_outputs = []
for i in range(len(training_inputs)):
    for j in range(len(training_inputs[i])):
        total_inputs.append(training_inputs[i][j])
        total_outputs.append(training_outputs[i][j])
pos_predicted,pos_true,neg_predicted,neg_true = NN.call_results(total_input
s,total_outputs,model)
accuracy = NN.evaluate_accuracy(pos_predicted,pos_true,neg_predicted,neg_tr
ue)
tmp_accuracies.append(accuracy)
#######stop time
stop = timeit.default_timer()
####print time
print('Time: ', stop - start)
```

```
Time:  1.9254704262129962
```

In [51]:
```
print('Printout A. Sample output')
print('This is an output from a positive example:')
model.prop_forward(NN.transform_string(pos_encoded[0]),NN.transform_string(
'1'))
print(model.layer3_activation)
print('This is an output from a negative example:')
model.prop_forward(NN.transform_string(neg_encoded[0]),NN.transform_string(
'0'))
print(model.layer3_activation)
```

```
Printout A. Sample output
This is an output from a positive example:
[0.93518982]
This is an output from a negative example:
[3.10463163e-09]
```

5.

Generate output from test set for evaluation by TAs.

First train model

In [2]:
```
# Load libraries
import pandas as pd
import copy
import numpy as np
import timeit
import matplotlib.pyplot as plt
####
from Bio import pairwise2
from Bio.Seq import Seq
from Bio.Alphabet import generic_dna, generic_protein
####
from hw4 import neuralnetwork as NN
from hw4 import encoding as EN
```

In [5]:
```
####read in filtered negatives
with open('test/negs_filtd.txt', 'r') as file:
    neg = file.read().splitlines()
####read in positives
with open('test/pos_filtd.txt', 'r') as file:
    pos = file.read().splitlines()
####
neg_encoded,pos_encoded = EN.encode_pos_neg(neg,pos,17)
```

In [8]:
```
####define parameters
num_parts = 7
#####split up data
neg_chunks,pos_chunks = NN.chunk_kfold(num_parts,neg_encoded,pos_encoded)
#####format training data and get outputs
training_inputs,training_outputs = NN.format_trainingdata(neg_chunks,pos_ch
unks)
######define parameters
step_size = 0.2
iterations = 5
######start time
start = timeit.default_timer()
#####setup model
model = NN.NeuralNet(68,8,1,step_size,0.01)
#####perform k-fold cross validation
for i in range(len(training_inputs)):
    train_inputs = []
    train_outputs = []
    for j in range(len(training_inputs)):
        if i != j:
            for item in training_inputs[j]:
                train_inputs.append(item)
            for item in training_outputs[j]:
                train_outputs.append(item)
    #######
    model = NN.train_model(iterations,train_inputs,train_outputs,model)
######
total_inputs = []
total_outputs = []
for i in range(len(training_inputs)):
    for j in range(len(training_inputs[i])):
        total_inputs.append(training_inputs[i][j])
        total_outputs.append(training_outputs[i][j])
pos_predicted,pos_true,neg_predicted,neg_true = NN.call_results(total_input
s,total_outputs,model)
accuracy = NN.evaluate_accuracy(pos_predicted,pos_true,neg_predicted,neg_tr
ue)
#######stop time
stop = timeit.default_timer()
####print time
print('Time: ', stop - start)
```

```
Time:  2.1515799439512193
```

Now put true holdout data into model for testing.

In [10]:
```
testset = pd.read_csv('rap1-lieb-test.txt',header=None)[0].tolist()
neg_encoded,pos_encoded = EN.encode_pos_neg(testset,testset,17)
```

In [18]:
```
with open('test/predictions.txt', 'w') as file:
    for i in range(len(pos_encoded)):
        model.prop_forward(NN.transform_string(pos_encoded[i]),NN.transform
_string('1'))
        ####
        file.write(testset[i])
        file.write('\t')
        file.write(str(float(model.layer3_activation)))
        file.write('\n')
```