In [34]:
```
import glob
import os
import sys
import pandas as pd
import numpy as np
from random import sample
from fasta_reader import read_fasta
from NN import NeuralNetwork as NeuralNetwork, rand, KFold_split
import random
from i_o import encoder
```

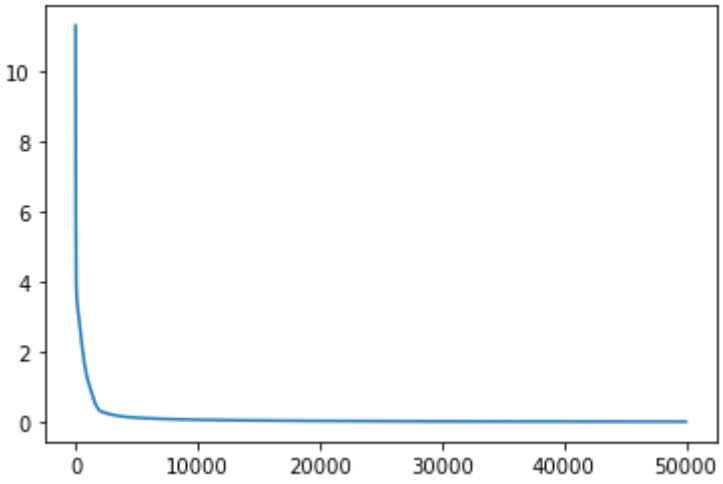
Part 1: Autoencoder implementation

In [36]:
```
# stimutate a 8 elements vector randomly to build the training set

empty_vec = np.zeros(8)
indexs = range(0,80)




# generate a dataset with 100 samples
inputs = []
groundtruth = []
for i in range(0,8):
    k =i %8
    vec = empty_vec.copy()
    vec[k] =1

    inputs.append(vec)
    groundtruth.append(vec)

```

In [50]:
```
k(input_layer=8, hidden_layer= 3, output_layer=8, batch_size= 8, print_frequ

groundtruth)
```

```
Epoch: 24000 Error 0.02742
Epoch: 25000 Error 0.02627
Epoch: 26000 Error 0.02521
Epoch: 27000 Error 0.02423
Epoch: 28000 Error 0.02332
Epoch: 29000 Error 0.02247
Epoch: 30000 Error 0.02168
Epoch: 31000 Error 0.02094
Epoch: 32000 Error 0.02025
Epoch: 33000 Error 0.01960
Epoch: 34000 Error 0.01899
Epoch: 35000 Error 0.01842
Epoch: 36000 Error 0.01787
Epoch: 37000 Error 0.01736
Epoch: 38000 Error 0.01687
Epoch: 39000 Error 0.01641
Epoch: 40000 Error 0.01598
Epoch: 41000 Error 0.01556
Epoch: 42000 Error 0.01517
Epoch: 43000 Error 0.01479
```

In [51]:
```
# training curve
Autoencoder.viz()
```



In [52]:
```
print("the test for [1,0,0,0,0,0,0,0] is", Autoencoder.test(np.array([1
print("the test for [0,1,0,0,0,0,0,0] is", Autoencoder.test(np.array([0
print("the test for [0,0,1,0,0,0,0,0] is", Autoencoder.test(np.array([0
print("the test for [0,0,0,1,0,0,0,0] is", Autoencoder.test(np.array([0
print("the test for [0,0,0,0,1,0,0,0] is", Autoencoder.test(np.array([0
print("the test for [0,0,0,0,0,1,0,0] is", Autoencoder.test(np.array([0
print("the test for [0,0,0,0,0,0,1,0] is", Autoencoder.test(np.array([0
print("the test for [0,0,0,0,0,0,0,1] is", Autoencoder.test(np.array([0
```

```
the test for [1,0,0,0,0,0,0,0] is [9.59933465e-01 3.06392126e-05 2.643339
50e-02 1.93939693e-02
 1.36030698e-02 2.54099024e-02 1.68514995e-02 4.95765667e-03]
the test for [0,1,0,0,0,0,0,0] is [2.03130041e-04 9.59888679e-01 2.852964
85e-02 1.85634394e-02
 2.24322464e-02 3.03753294e-02 1.73803470e-02 1.05489456e-02]
the test for [0,0,1,0,0,0,0,0] is [1.86644619e-02 2.01489478e-02 9.597617
92e-01 1.73522212e-02
 6.29671757e-04 8.77292785e-04 2.27718406e-02 1.08425733e-02]
the test for [0,0,0,1,0,0,0,0] is [2.78533998e-02 2.07218377e-02 1.178956
79e-02 9.67026297e-01
 1.90853705e-02 8.94954575e-04 3.00380087e-04 2.14392642e-02]
the test for [0,0,0,0,1,0,0,0] is [1.85047584e-02 8.59594395e-03 9.271930
21e-05 5.88620295e-03
 9.65170671e-01 9.70496000e-03 1.74897549e-02 2.60581396e-03]
the test for [0,0,0,0,0,1,0,0] is [2.46031876e-02 2.54989179e-02 1.011349
77e-03 1.94887706e-04
 1.38616705e-02 9.64170704e-01 9.07322276e-03 1.56569067e-02]
the test for [0,0,0,0,0,0,1,0] is [2.12785498e-03 1.29343006e-02 1.755743
10e-02 1.44394855e-04
 1.54539296e-02 1.70238442e-03 9.61793322e-01 1.94960869e-02]
the test for [0,0,0,0,0,0,0,1] is [0.00338813 0.00567007 0.00417008 0.018
30194 0.01238566 0.0201941
 0.01904226 0.96563836]
```

Part 2: Adapt for classification, and develop training regime

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.

The DNA sequence havs A T C G, all need to be treat differently but evenly. So I decide that for each A, T , C, G we use a vector to represent it. A as [1,0,0,0] T as [0,1,0,0] C as [0,0,1,0] G as [0,0,0,1]

For the full seuqnce we merge them together. Add them one by one based on the sequence of nucleotide.

This method assme that the contribution of each nucleotide to the prediction are equally to each other.

If A, T, C, G are encoded as 1, 2, 3, 4, then these values can affect training not equally, which is a bad idea.

However, if there are certain pattern of seuqnce which can have special results: for example, if sequence 'ATCGA' affect the result from an expontail scale, I think my encode method won't be able to detect that.

```
However, based on my following results I think my encoding method w
orks relavately well for the TF bnding site recogniation.
So I will go with this for now unless there is more information tha
t can guibe me to weight these four nucleotide differently.
```

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 building the training dataset. Since there is a huge amount of negative seuqneces I can use. I need to be careful because if I use all the possible negative sites, the compiutation will be forever, and the worst case would be the model will tend to bindly turn whatever unknwoen sequences into the negative becauuse the negative sites are too overwhleming.

To avoid this situation, I decide to use all 137 positive examples but randomly sample 137 negative samples from the negative seuqnces Following these rules: 1) all are taken from the negative seuqnces 2) length are the same 3) will at least have two mismatches with all the positive sites

To avoid the misleading from the a batch. for loading the dataset, every time once I load a positive sites I also load a negative sites. In this way I can make sure everytime I train a batch, there is 50 percent of positive sites and 50 percent of negative sites.

In [53]:

```
# Train the network to recogize the TF binding site


# positive sites
positive_sites = []
with open('../data/rap1-lieb-positives.txt','r') as F:
    lines = F.readlines()
    for line in lines:
        line = line.split('\n')[0]
        positive_sites.append(line)


# negative sites
p = 0
negative_background = []
with read_fasta("../data/yeast-upstream-1k-negative.fa") as file:
    for seq in file:

        negative_background.append(seq.sequence)


# Prepare the negative sites:


# define a function return the different nucleotide number between two
def diff_num(string1, string2):
    num = 0
    for i in range(len(string1)):
        if string1[i] != string2[i]:

            num +=1
    return num


# randomly smaple 137 17-neocletide-length seugnces from the negative c
negative_sub_backgroundsample = sample(negative_background,150,) # took
negative_sites = []
for a_negative_string in negative_sub_backgroundsample:
    length = len(a_negative_string)
    flag= False
    while (flag == False):

        start = int((length -17)*random.random())
        a_negative_string = a_negative_string[start: start + 17]
        for positive_site in positive_sites:
            flag= True
            if diff_num(positive_site, a_negative_string) <2:
                flag= False
                break

    assert flag == True, 'Something wrong!'

    negative_sites.append(a_negative_string)
    negative_sites = list(set(negative_sites))
    if len(negative_sites) == 137:
        break
```

```
57
```

In [54]:
```
positive_sites_encoded = encoder(positive_sites)
negative_sites_encoded = encoder(negative_sites)
```

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

In [60]:
```
print('Input positive sequence')
print(positive_sites[0])
print('Output positive seuqnce')
print(positive_sites_encoded[0])


print('Input negative sequence')
print(negative_sites[0])
print('Output negative seuqnce')
print(negative_sites_encoded[0])
```

```
Input positive sequence
ACATCCGTGCACCTCCG
Output positive seuqnce
[1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0,
0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0,
0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1]
Input negative sequence
TCTTCTTTTTTGAATAT
Output negative seuqnce
[0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 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, 0, 1,
1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0]
```

In [61]:
```

inputs = []
groundtruth = []
for i in range(len(positive_sites_encoded)):
    inputs.append(positive_sites_encoded[i])
    groundtruth.append([1])
    inputs.append(negative_sites_encoded[i])
    groundtruth.append([0])
```

In [62]:
```
TFs_NN = NeuralNetwork(input_layer=68, hidden_layer= 25, output_layer=1

```

In [58]:
```
TFs_NN.make_weights()
```

In [59]:
```
1 TFs_NN.train(inputs,groundtruth)
```

```
5000
Epoch: 0 Error 51.07528
Epoch: 1000 Error 1.11620
Epoch: 2000 Error 0.41760
Epoch: 3000 Error 0.27338
Epoch: 4000 Error 0.19960
```

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

For the design, I used a input layer of 68 nodes with an addtional node as the bias. For the hidden layer I used 25 nodes. For the output layer, I used one node as the probability of this binding site is a positive binding site. 1 as the True positive. 0 as the negative sequences.

I use the gradient descnet, and for the loss function I use the minimal sqaure error (MSE)

So far, it looks like the training went very well. But it needs more support from the following k flod validation.

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

I will think the error converge if the MSE doesn't decrease any more. Or the MSE is swing around a specfic value for a long time

Part 3: Cross-validation

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

k fold valdation is a powerful method if we believe that my data has a limitation Because it ensures that every observation from the original dataset has the chance of appearing in training and test set.

I will random split my dataset into several pieces. Each time hold one part for the testing and use all the other parts for the training. Use this method to evaluate the model performance and make sure that my model can be trained fairly.

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

Usually if you are having lots of dataset. The ideal cluster is 10. However, given that my dataset is not big, I decide to do k = 5. In this way, my training and testing data are 80% , 20% split. It is also acceptable.

In [63]:
```
K_fold_split = KFold_split(inputs, groundtruth, 5)
```

```
54
[46, 190, 219, 153, 162, 165, 237, 26, 75, 2, 72, 191, 225, 221, 57, 106,
80, 258, 52, 227, 173, 203, 91, 246, 144, 195, 240, 244, 148, 171, 234, 2
8, 238, 184, 29, 259, 17, 49, 138, 133, 207, 210, 253, 185, 9, 212, 100,
20, 12, 170, 146, 33, 249, 206]
[180, 6, 216, 7, 176, 231, 155, 181, 255, 40, 172, 104, 235, 134, 205, 15
4, 22, 45, 105, 61, 267, 163, 186, 56, 42, 145, 18, 4, 131, 252, 220, 20
9, 271, 273, 177, 228, 14, 140, 13, 169, 66, 113, 242, 229, 200, 51, 103,
158, 1, 202, 164, 41, 218, 63]
[214, 211, 201, 79, 71, 262, 143, 83, 189, 98, 38, 115, 135, 193, 233, 0,
125, 39, 30, 124, 179, 70, 55, 107, 149, 111, 187, 119, 199, 116, 217, 16
8, 97, 128, 87, 21, 150, 32, 117, 272, 261, 270, 68, 178, 11, 59, 78, 24,
58, 3, 256, 188, 243, 166]
[69, 84, 222, 232, 8, 147, 127, 95, 102, 241, 141, 137, 112, 268, 94, 26
6, 37, 96, 257, 74, 196, 159, 108, 35, 5, 50, 192, 23, 44, 62, 54, 121, 1
97, 19, 264, 157, 89, 109, 126, 208, 86, 239, 160, 263, 230, 265, 250, 18
3, 101, 90, 60, 142, 269, 182]
[132, 198, 64, 43, 77, 53, 130, 167, 226, 174, 92, 25, 93, 15, 114, 110,
118, 82, 88, 36, 48, 34, 215, 213, 129, 224, 136, 247, 248, 10, 139, 156,
223, 151, 120, 76, 251, 31, 16, 65, 254, 260, 73, 122, 204, 99, 236, 67,
161, 81, 47, 123, 27, 245, 85, 152, 175, 194]
```

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

In [8]:
```
import matplotlib.pyplot as plt
```

In [10]:

```
for i in range(len(K_fold_split)):
    print('The validation: '+str(i+1))
    testing_set = K_fold_split[i]
    # test_set
    test_inputs = testing_set[0]
    test_groundtruth = testing_set[1]
    training_inputs = []
    training_groundtruth = []
    # training_set
    for j in range(len(K_fold_split)):
        if i ==j:
            pass
        else:
            assert i != j, ' something wrong.'
            training_inputs = training_inputs + K_fold_split[j][0]
            training_groundtruth = training_groundtruth + K_fold_split[
    # train the model
    NN = NeuralNetwork(input_layer=68, hidden_layer= 25, output_layer=1
    NN.make_weights()
    NN.train(training_inputs, training_groundtruth)
    fig = AUROC_cruve(NN,test_inputs, test_groundtruth, Fig= True)

    print('Plot the training curve')

    f1, ax1= plt.subplots()
    NN.viz()
    #f1.show()

    fig.plot()
```

```
The validation: 1
1000
Epoch: 0 Error 52.51911
Epoch: 100 Error 17.62825
Epoch: 200 Error 7.43056
Epoch: 300 Error 4.54408
Epoch: 400 Error 3.29080
Epoch: 500 Error 2.61617
Epoch: 600 Error 2.18425
Epoch: 700 Error 1.87446
Epoch: 800 Error 1.63479
Epoch: 900 Error 1.43721
Plot the training curve
The validation: 2
1000
Epoch: 0 Error 49.28880
Epoch: 100 Error 15.08163
Epoch: 200 Error 8.74777
Epoch: 300 Error 5.66807
```

Part 4: Extension

Try something fun to improve your model performance! This should include implementation of alternative optimization methods (particle swarm, genetic algorithms, etc), you can also optionally add changes in the network architecture such as modifying the activation function, changing the architecture, adding regularization etc. For this section, we want to see a description of what you want to try and why. As long as we have this, and some effort towards implementation, you will get full points.

• What set of learning parameters works the best? Please provide sample output from your system.

• What are the effects of altering your system (e.g. number of hidden units or choice of kernel function)? Why do you think you observe these effects?

• What other parameters, if any, affect performance?

Answers:

I performed a gnetic algorthim. Using six hyperparameters: 1) nodes of hidden layer 2) learning rate 3) learning decay 4) momentum factor 5) batch size 6) times of iteration

In theory, since random generate times of interation can make the algorithm take forever. I decide to set it stable for this test mission.

ps: For this specific case, when I was running I realize with all the combination, I get a AUC score 1. In this case, there is not really something we need to optimize. It will be worthwhile to consider use the MSE as the parameter to improve in the future.

In [60]:
```
genetic_algorithm(training_inputs, training_groundtruth, test_inputs, t
                      4,5,2, [10,100], [0,1], [0,1], [0,1], [1, 274], [1
```

```
105
Epoch: 0 Error 26.59644
Epoch: 100 Error 2.60180
103
Epoch: 0 Error 49.03093
Epoch: 100 Error 0.02108
For the time 4 the best candidates and the best result is
[87, 0.29119834160481906, 0.4555150156548764, 0.017018243282391543, 24, 1
00]
1.0
100
Epoch: 0 Error 48.34411
100
Epoch: 0 Error 56.69709
87
Epoch: 0 Error 57.97561
109
Epoch: 0 Error 31.38646
Epoch: 100 Error 0.06883
114
```

[87, 0.29119834160481906, 0.4555150156548764, 0.017018243282391543, 24, 100] are the one that work the best. But I don't buy this too much since overall the model work very well.

THese parameters are definitely affect the system. But unfortunately I didn't observe these effect very obviously in this specific case.

The parameters which I didn't test are the layers of neural network, here I was only using one hidden layer. In the reall situation adding more layer will generally help the model while at the same time makes it more computational expensive.

Part 5: Evaluate your network on the final set.

Select a final model (encoding, architecture, training regime). This can be the same as your model in Part 3, Part 4, or something completely different.

Since all the model work fine. I decide to use my original model, which is 25 hidden layers, lr = 0.05, batch_size=137, iteration=10000, others are as the default.

In [67]:
```
network(input_layer=68, hidden_layer= 25, output_layer=1, batch_size=137, lr =
ts(2)
ts,groundtruth)
```

```
10000
Epoch: 0 Error 62.58270
Epoch: 1000 Error 3.85194
Epoch: 2000 Error 1.14044
Epoch: 3000 Error 0.62496
Epoch: 4000 Error 0.41843
Epoch: 5000 Error 0.30955
Epoch: 6000 Error 0.24220
Epoch: 7000 Error 0.19709
Epoch: 8000 Error 0.16503
Epoch: 9000 Error 0.14128
```

In [68]:
```
test_sites = []
with open('../data/rap1-lieb-test.txt','r') as F:
    lines = F.readlines()
    for line in lines:
        line = line.split('\n')[0]
        test_sites.append(line)
```

In [69]:
```
test_sites_encoded = encoder(test_sites)
```

In [71]:
```
with open('../test_scores.txt','w') as F:
    index = 0
    for seq_encode in test_sites_encoded:

        F.write(test_sites[index]+'\t')

        F.write(str(TFs_NN.test(seq_encode)[0][0])+'\n')
        index = index +1
```

In [ ]:
```

```