ECS 171 Homework2 Report

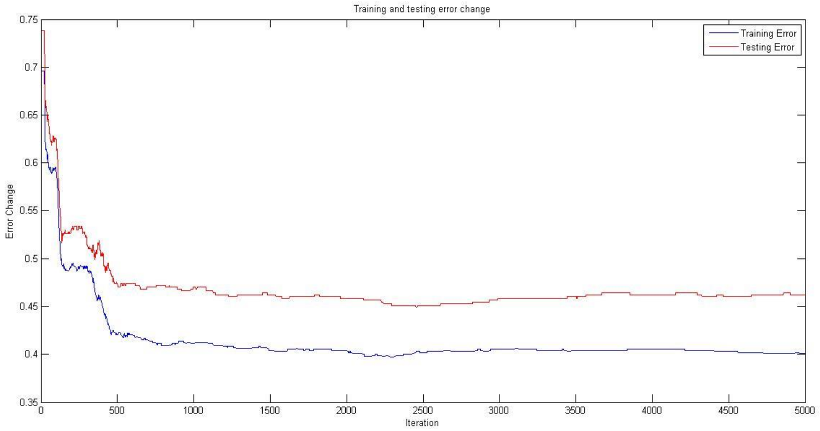
Yilun Yang
912425744

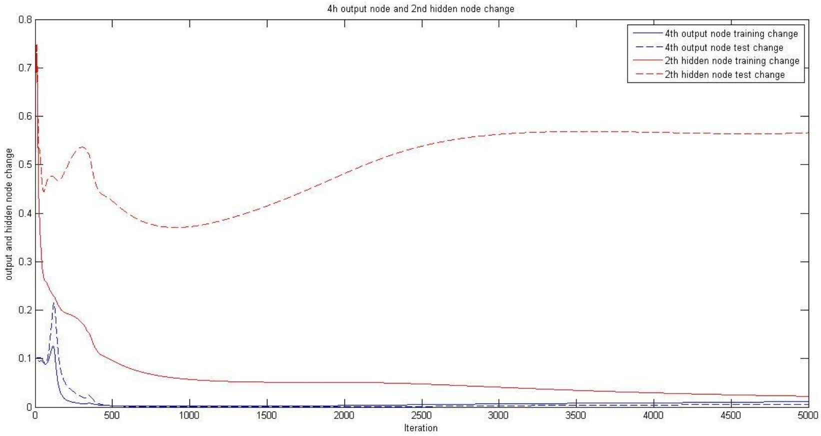
Note: 1st problem contains: Problem1.m, ANN.m, ANN_new.m
2nd problem contains: Problem2.m, ANN_all.m
3rd problem contains: Problem3.m, ANN_1st_iter, Problem3_derive
4th problem contains: Problem4.m, ANN_onelayer.m, ANN_twolayers.m, ANN_threelayers.m, Problem4_derive.
5th problem contains: Problem5.m, ANN_pred.m
6th problem contains: Problem6.m

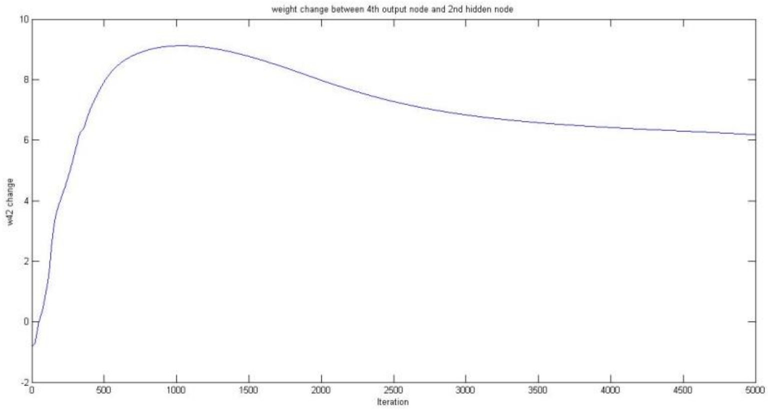
1. *Construct a 3-layer artificial neural network (ANN) and specifically a feed-forward multilayer perceptron to perform multi-class classification. The hidden layer should have 3 nodes. Split your data into a random set of 65% of the samples as the training set and the rest 35% as the testing set. Plot how the weights, error and output changes at each iteration for both the training set and the testing set.*

For the first problem, I implemented two ways to calculate, the first is to calculate and update each weight, activation function result, output, etc. element-wisely. This method is clear but very slow. Thus I tried to wrap all loops in matrixes, this vectorized method is much faster than the first one thus we will use it to solve the following problems as well.

We used 0.1 learning rate and iterate 5000 times to ensure convergence. In the following problems, I will keep learning rate the same for comparison. The images are as following.







In the first plot, our lowest training and testing error are 0.4 and 0.46 and we can see obvious drop from iter1 to iter5000.

In the second plot, I picked the last observation from training set and testing set to observe their 4^{th} output node change and 2^{nd} hidden node change. We can see two blue lines (output node) both converge to 0. This does make sense because we know neither the final observation in training data nor testing data belong to 4^{th} class thus a good method should predict this probability as zero. The second hidden node in training set converges to zero as well. This means the second hidden node has little effect on the output values. On the contrary, for testing set, the node value is far from zero all the time, we may conclude it has much more effect on the output values.

For the final plot, the weight between 4^{th} output node and 2^{nd} hidden node grows up first and drops a bit then. Since w42 is not zero all the time, we may conclude the 4^{th} output code is always activated by the 2^{nd} hidden node.

2. *Now re-train the ANN with all your data (all 1484 samples). What is your training error? Provide the final activation function equations after training.*

To make the problems comparable, I always set seed for sampling and use learning rate 0.1 and iteration time 5000 for the problems unless something special happens.

After training the whole dataset, we calculated the training error as 0.3996 which is lower than what we got in the first problem (0.4).

I also output all the weights into a struct thus all the activation functions can be written out directly. They are:

$$z_1^{(3)} = -0.1502 + 0.9501a_1^{(2)} -4.1108a_2^{(2)} -1.5927a_3^{(2)}$$

$$z_2^{(3)} = -3.4954 + 2.7155a_1^{(2)} -2.2443a_2^{(2)} +1.5131a_3^{(2)}$$

$$z_3^{(3)} = -0.9722-3.9091\ a_1^{(2)} -1.2660a_2^{(2)} + 2.4482a_3^{(2)}$$

$$z_4^{(3)} = -7.8201+4.8895\ a_1^{(2)} +5.3387a_2^{(2)} -0.4923a_3^{(2)}$$

$$z_5^{(3)} = -2.4821-2.1063a_1^{(2)} +1.8325a_2^{(2)} -1.7412a_3^{(2)}$$

$$z_6^{(3)} = -3.6730-10.5220a_1^{(2)} +4.5198a_2^{(2)} -0.3342a_3^{(2)}$$

$$z_7^{(3)} = 0.4076-24.1784a_1^{(2)} -1.7614a_2^{(2)} -3.6873a_3^{(2)}$$

$$z_8^{(3)} = -3.4842+0.4483a_1^{(2)} +0.1765a_2^{(2)} -3.7078a_3^{(2)}$$

$$z_9^{(3)} = -3.9824-1.2446a_1^{(2)} +0.8511a_2^{(2)} +0.5002a_3^{(2)}$$

$$z_{10}^{(3)} = -2.7858-3.0130a_1^{(2)} -0.9900a_2^{(2)} -5.9634a_3^{(2)}$$

$$z_1^{(2)} = 13.7184-11.9108a_1^{(1)} -8.6410a_2^{(1)} -8.4169a_3^{(1)} -12.9767a_4^{(1)} +7.0200a_5^{(1)} -4.2753a_6^{(1)} -3.3248a_7^{(1)} +14.3444a_8^{(1)}$$

$$z_2^{(2)} = 16.7461-0.7601a_1^{(1)} +5.2919a_2^{(1)} -44.8134a_3^{(1)} -0.2336a_4^{(1)} +1.4455a_5^{(1)} +4.5778a_6^{(1)} -1.1960a_7^{(1)} -0.1159a_8^{(1)}$$

$$z_3^{(2)} = -7.4332-7.7925a_1^{(1)} -1.7200a_2^{(1)} +10.2545a_3^{(1)} +17.8761a_4^{(1)} -10.2463a_5^{(1)} +2.7729a_6^{(1)} -4.4224a_7^{(1)} +35.2272a_8^{(1)}$$

$$g(z) = \frac{1}{1+\exp(-z)}$$

Of course, after we calculate all the $z_j^{(l)}$ s, we need to put them into g(z) function to get the corresponding $a_j^{(l)}$.

3. *For the ANN that you have built calculate the first round of weight update with back-propagation with paper and pencil for all weights but for only the first sample. Provide both calculations made by hand and corresponding output from the program that shows that both are in agreement.*

We first calculate all the first round of weight update for the first sample of yeast dataset with program:

Our original randomized weights are:

Input layer to hidden layer: each column represents all weights to the corresponding hidden node.

0.3155	0.4008	0.6544
0.7690	-0.6046	-0.8902
0.1778	-0.6700	0.6786
0.6395	-0.8824	-0.8991
-0.5456	0.5225	0.5855
-0.6845	-0.3599	-0.7245
0.2231	-0.5273	-0.5578
0.3048	-0.2826	0.9964
0.2064	0.0721	-0.7771

Hidden layer to output layer: each column represents all weights to the corresponding output node.

Columns 1 through 4

-0.1280	-0.1593	-0.4007	-0.7308
-0.9481	-0.3393	-0.4663	0.0272
0.0993	-0.5907	0.2423	-0.6311
-0.1294	0.2385	0.0583	0.5707

Columns 5 through 8

0.7080	0.0105	-0.7457	-0.5594
-0.0115	-0.8694	0.1935	-0.3003
0.6931	-0.1438	-0.5480	-0.0644
-0.8407	-0.8069	-0.7861	-0.5965

Columns 9 through 10

0.2808	0.5873
-0.0339	0.1600
0.0105	-0.6754
-0.2262	0.4015

After the update, we get the new weights:

Input layer to hidden layer:

0.3166	0.4042	0.6579
0.7696	-0.6026	-0.8882
0.1785	-0.6679	0.6808
0.6401	-0.8809	-0.8974
-0.5454	0.5230	0.5860
-0.6839	-0.3582	-0.7227
0.2231	-0.5273	-0.5578
0.3054	-0.2810	0.9981
0.2067	0.0728	-0.7764

```
Columns 1 through 4

 -0.1343   -0.1686   -0.3859   -0.7391
 -0.9527   -0.3461   -0.4557    0.0212
  0.0977   -0.5932    0.2462   -0.6333
 -0.1327    0.2335    0.0662    0.5662

Columns 5 through 8

  0.6935    0.0057   -0.7499   -0.5640
 -0.0220   -0.8728    0.1904   -0.3037
  0.6892   -0.1450   -0.5491   -0.0657
 -0.8485   -0.8095   -0.7884   -0.5990

Columns 9 through 10

  0.2675    0.5725
 -0.0434    0.1493
  0.0069   -0.6794
 -0.2334    0.3935
```

Next, we will calculate all weight with paper, details are omitted. After our calculation and comparison, we find all weights match each other in two forms.

4. *Increase the number of hidden layers from 1 to 2 and then to 3. Then increase the number of hidden nodes per layer from 3 to 6, then to 9 and finally to 12. Create a 3x4 matrix with the number of hidden layers as rows and the number of hidden nodes per layer as columns, with each element (cell) of the matrix representing the testing set error for that specific combination of layers/nodes. What is the optimal configuration? What you find the relationship between these attributes (number of layers, number of nodes) and the generalization error (i.e. error in testing data) to be?*

   I derived the weight update rule for two hidden layer and three hidden layer model on paper.

   We first show the testing error of all pairs:

	3 nodes	6 nodes	9 nodes	12 nodes
One layer	0.4624	0.4374	0.4200	0.4297
Two layers	0.4509	0.4566	0.4528	0.4566
Three layers	0.4913	0.4644	0.4855	0.4566

After comparison, one layer with 9 nodes is the best model and we will use this model to predict new sample in problem5.

For the one layer models, testing error first decrease with the increase of nodes then increase at node 12. Small number of nodes is not enough to describe the dataset while too many nodes may cause over fitting. Errors for two layers and three layers are quite random and don't have obvious trends. On the other hand, if we compare the results row by row, errors tend to become bigger in most cases with the increase of layer number. Thus, one layer is already enough for this dataset.

There are some others things I want to point out, two layer and three models are very sensitive to learning rate and initial weight range and may converge very slow (small learning rate), never converge at all (saturated due to bad weight range) or very unstable (big learning rate) if we pick the inappropriate learning rate and weights. I tried many combinations of them and found some interesting things. First, if learning rate is less than 0.05, it would take huge amount of time to converge while if bigger than 0.6, the error becomes very unstable (sometimes more than 0.7, sometimes between 0.6 and 0.7 depending on iteration time and hidden node number). Besides, if we sample weights from [0, 1], the three layer model would NEVER converge. A more appropriate range is [-1, 1], from the table, we can see the model converge pretty well with range [-1, 1].

5. *Which class does the following sample belong to?*
   We used the one layer, 9 hidden nodes model to predict, probability for each class are 0.3955, 0.5450, 0.0097, 0.0217, 0.0000, 0.0000, 0.0000, 0.0127, 0.0000, 0.0001, thus the sample should be classified into the second class which is NUC. However, the probability for class CYT is very high as well.

6. *Can you come up with a quantitative measure of uncertainty for each classification? What is the uncertainty for the unknown sample of the previous question? Justify your assumptions and method?*

   First, if we are 100% certain of the classification, then the prediction result should have a class probability as 1 and other nine classes as 0. So uncertainty comes from the gap between prediction probability and 0 or 1. We can write the formula as following:

$$score = \sum_{k=1}^{10} | a_k^{(l)} - s_k |, \qquad s_k = \begin{cases} 0 & \text{if } a_k^{(l)} \neq \max(a_i^{(l)}), i = 1, 2...10 \\ 1 & \text{if } a_k^{(l)} = \max(a_i^{(l)}), i = 1, 2...10 \end{cases}$$

   So the uncertainty score for our new data should be 0.7517 which is calculated in program.

   Next, we should think what the max value of uncertainty score is. We here assume the biggest probability of 10 classes is at least 0.5 while other nine probabilities are always no more than 0.5. Otherwise, we should think this model as inappropriate because the classifier doesn't distinguish each class from others. Then we need to retrain other ANN models or try other type of models instead of calculating uncertainty.

   Anyway, under this condition, the biggest uncertainty score should be 5, which is reached if all the probabilities are 0.5. In this situation, we don't have any information on the dataset. Now we can calculate the uncertainty rate as

   Rate = score/5 = 15.03%
   In other words, we are 15% unsure if the classification is correct.