Report for HW4

Q1: How does an autoencoder detect errors?

A: An autoencoder purposes to reconstruct its own inputs, so it detects errors by comparing inputs and outputs.

Q2: The network starts out with 28*28 = 784 inputs. Why do subsequent layers have fewer nodes?

A: The goal of the network is to classify the 784-pixels images into 10 classes. It's a process of information compressing or say dimensions reduction. And the number of nodes shows the amount of information or say the number of dimensions. Therefore, the number of nodes reduces in network.

Q3: Why are autoencoders trained one hidden layer at a time?

A: There is some difficulty to train neural networks with multiple hidden layers in practice. For example, once errors are backpropagated to the first few layers, they become minuscule and insignificant so that the network will almost always learn to reconstruct the average of all the training data.[*wikipedia*] By training one hidden layer at a time, stacking them and then fine-tuning, some of the difficulties can be evaded, producing a more desirable network.

Q4: How were the features in Figure 3 obtained? Compare the method of identifying features here with the method of HW1. What are a few pros and cons of these methods?

A: The features are obtained by training neural network. The weights of nodes in neural network composes the features. The pros of this method are only little supervision needed, well-adapted and suitable for multiple-layer structure. The cons of this method are training needed, yielding features incomprehensible for human and not so robust.

Q5: What does the function plotconfusion do?

A: It plot classification confusion matrix to show an evaluation of performance of the neural network. The confusion matrix shows accuracy in different situations.

Q6: What activation function is used in the hidden layers of the MATLAB tutorial?

A: Logistic-sigmoid function.

Q7: In training deep networks, the ReLU activation function is generally preferred to the

sigmoid activation function. Why might this be the case?
A: The ReLU activation function give rise to sparsity, which reduce overfitting. For deep network, the sigmoid is easy to lose gradient when be close to saturation region. The ReLU activation function also benefits computation of backpropagation.

Q8: The MATLAB demo uses a random number generator to initialize the network. Why is it not a good idea to initialize a network with all zeros? How about all ones, or some other constant value?
A: To initialize a network with a constant value makes the nodes symmetric to each other. And there is no other random process in training. As a result, every node in a layer will learn the same thing so that the network wouldn't work as desired.

Q9: Give pros and cons for both stochastic and batch gradient descent. In general, which one is faster to train in terms of number of epochs? Which one is faster in terms of number of iterations?
A: Compared to SGD, BGD is more robust to get a satisfying result. SGD can be used for online learning, BGD can't. SGD is faster in terms of number of epochs. BGD is faster in terms of number of iterations.

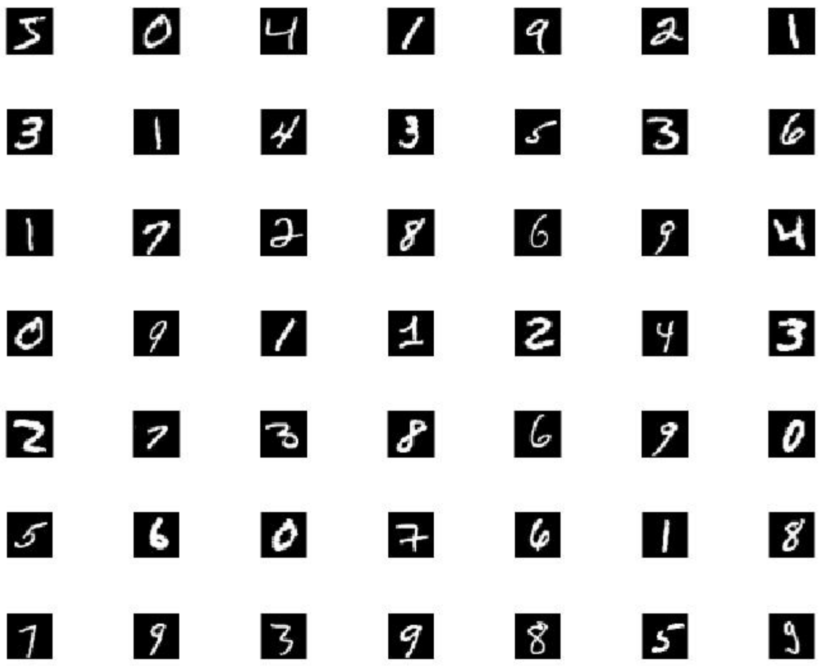
Q10: Report the impact of slightly modifying the parameters. Is the tutorial presentation robust or fragile with respect to parameter settings?
A:
Condition: I used MNIST for the question. The number of test images and train images are both 1000. The accuracy is about 78% using parameters in tutorial.
Change numbers of layers: The accuracy increases to 86% when the second layer is removed. An error occurred when stack encoders when a third layer is added. The following "a little" refers to about 5%.
Change numbers of nodes of layer 2: The accuracy changes a little when it changes to 20 or 75.
Change numbers of nodes of layer 1: The accuracy changes a little when it changes to 250 or 75. Adding nodes makes running time increase.
Change SparsityRegularizations: The accuracy changes a little when they change from 0.25 to 200.
Change SparsityProportions: The accuracy changes a little when they scale from 0.1 to 6.

In most of the conditions above, the accuracy increase to about 84%.

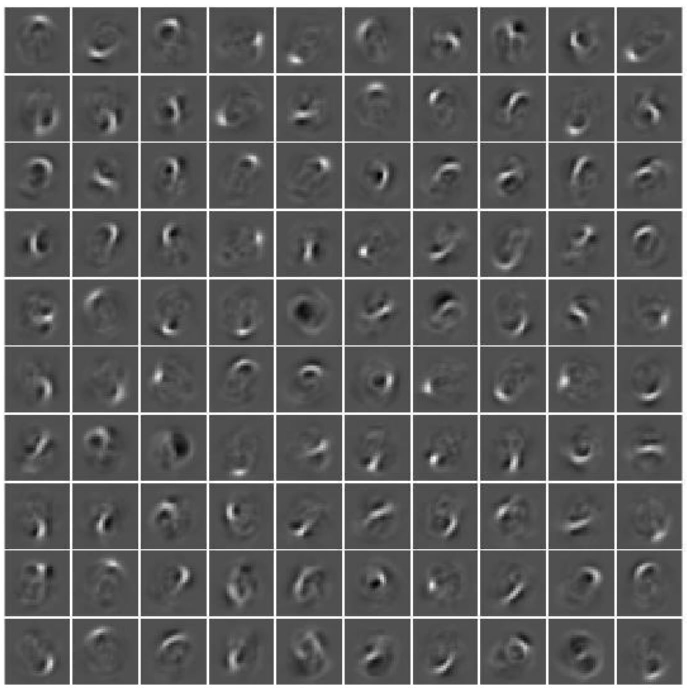
The tutorial presentation is robust with respect to parameter settings. I think this may due to the fine-tuning process. In some conditions, the accuracy is quite low before fine-tuning, which become almost as high as original accuracy after fine-tuning.

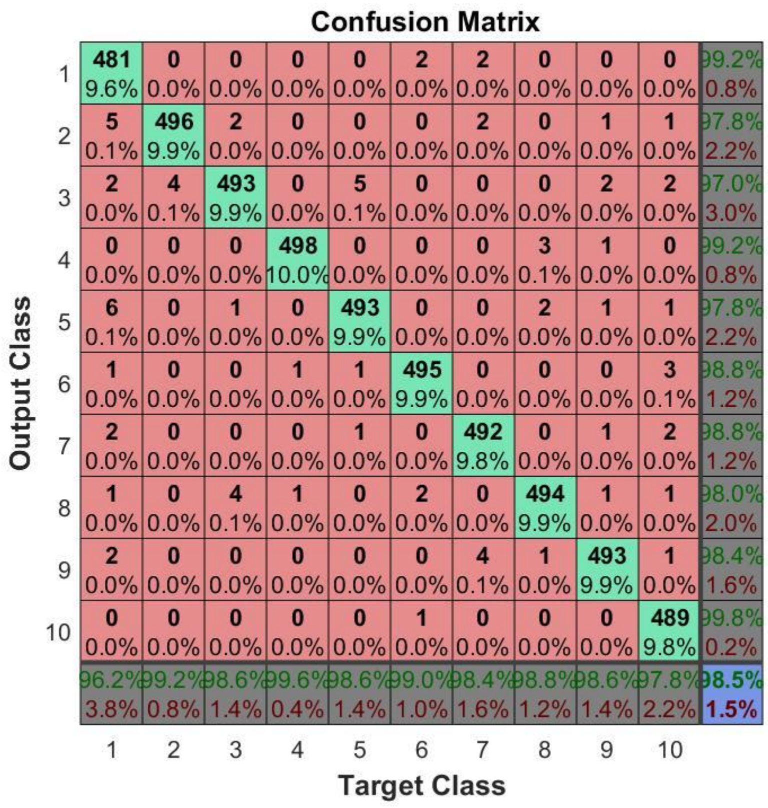
Extra Credit

1. Here are a few “real” images of digits.



2. Results from synthetic images:

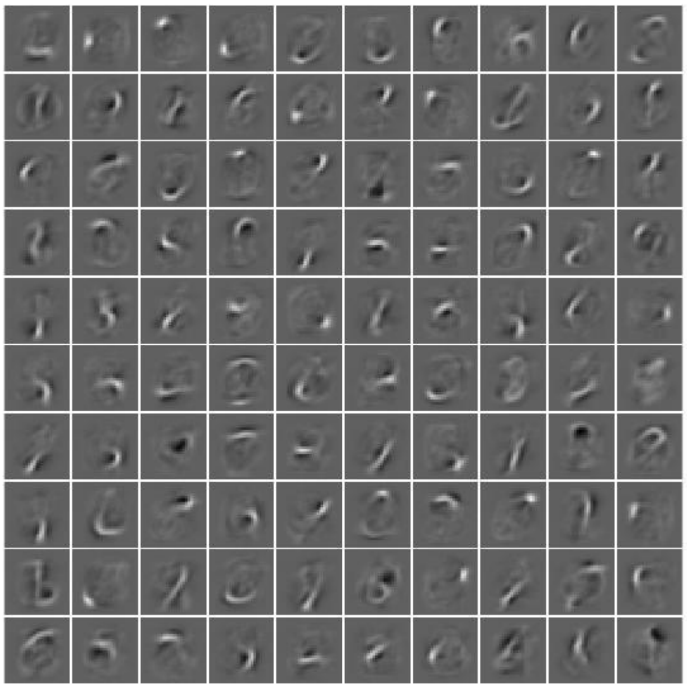


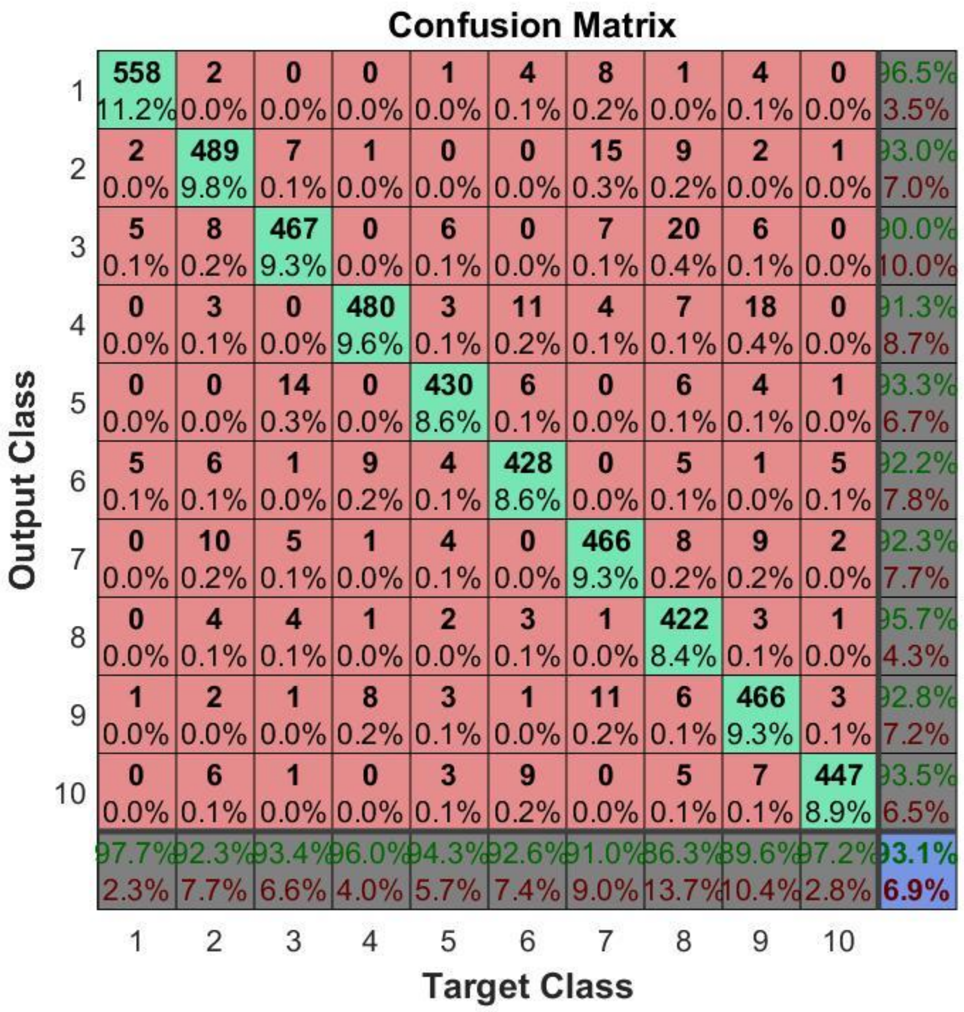


Confusion Matrix

Output Class \ Target Class	1	2	3	4	5	6	7	8	9	10	
1	**481** 9.6%	**0** 0.0%	**0** 0.0%	**0** 0.0%	**0** 0.0%	**2** 0.0%	**2** 0.0%	**0** 0.0%	**0** 0.0%	**0** 0.0%	99.2% 0.8%
2	**5** 0.1%	**496** 9.9%	**2** 0.0%	**0** 0.0%	**0** 0.0%	**0** 0.0%	**2** 0.0%	**0** 0.0%	**1** 0.0%	**1** 0.0%	97.8% 2.2%
3	**2** 0.0%	**4** 0.1%	**493** 9.9%	**0** 0.0%	**5** 0.1%	**0** 0.0%	**0** 0.0%	**0** 0.0%	**2** 0.0%	**2** 0.0%	97.0% 3.0%
4	**0** 0.0%	**0** 0.0%	**0** 0.0%	**498** 10.0%	**0** 0.0%	**0** 0.0%	**0** 0.0%	**3** 0.1%	**1** 0.0%	**0** 0.0%	99.2% 0.8%
5	**6** 0.1%	**0** 0.0%	**1** 0.0%	**0** 0.0%	**493** 9.9%	**0** 0.0%	**0** 0.0%	**2** 0.0%	**1** 0.0%	**1** 0.0%	97.8% 2.2%
6	**1** 0.0%	**0** 0.0%	**0** 0.0%	**1** 0.0%	**1** 0.0%	**495** 9.9%	**0** 0.0%	**0** 0.0%	**0** 0.0%	**3** 0.1%	98.8% 1.2%
7	**2** 0.0%	**0** 0.0%	**0** 0.0%	**0** 0.0%	**1** 0.0%	**0** 0.0%	**492** 9.8%	**0** 0.0%	**1** 0.0%	**2** 0.0%	98.8% 1.2%
8	**1** 0.0%	**0** 0.0%	**4** 0.1%	**1** 0.0%	**0** 0.0%	**2** 0.0%	**0** 0.0%	**494** 9.9%	**1** 0.0%	**1** 0.0%	98.0% 2.0%
9	**2** 0.0%	**0** 0.0%	**0** 0.0%	**0** 0.0%	**0** 0.0%	**0** 0.0%	**4** 0.1%	**1** 0.0%	**493** 9.9%	**1** 0.0%	98.4% 1.6%
10	**0** 0.0%	**0** 0.0%	**0** 0.0%	**0** 0.0%	**0** 0.0%	**1** 0.0%	**0** 0.0%	**0** 0.0%	**0** 0.0%	**489** 9.8%	99.8% 0.2%
	96.2% 3.8%	99.2% 0.8%	98.6% 1.4%	99.6% 0.4%	98.6% 1.4%	99.0% 1.0%	98.4% 1.6%	98.8% 1.2%	98.6% 1.4%	97.8% 2.2%	**98.5% 1.5%**

Results from “real” images:





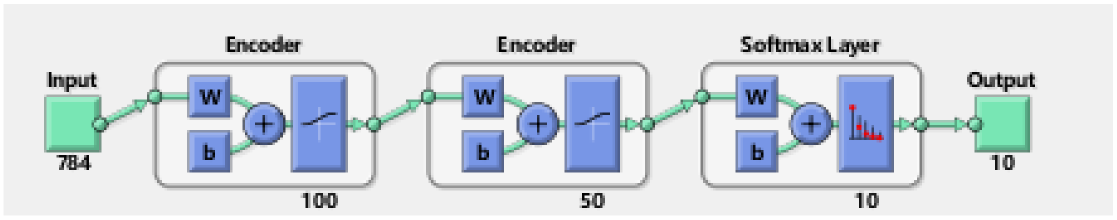
Confusion Matrix

Output Class \ Target Class	1	2	3	4	5	6	7	8	9	10	
1	558 11.2%	2 0.0%	0 0.0%	0 0.0%	1 0.0%	4 0.1%	8 0.2%	1 0.0%	4 0.1%	0 0.0%	96.5% 3.5%
2	2 0.0%	489 9.8%	7 0.1%	1 0.0%	0 0.0%	0 0.0%	15 0.3%	9 0.2%	2 0.0%	1 0.0%	93.0% 7.0%
3	5 0.1%	8 0.2%	467 9.3%	0 0.0%	6 0.1%	0 0.0%	7 0.1%	20 0.4%	6 0.1%	0 0.0%	90.0% 10.0%
4	0 0.0%	3 0.1%	0 0.0%	480 9.6%	3 0.1%	11 0.2%	4 0.1%	7 0.1%	18 0.4%	0 0.0%	91.3% 8.7%
5	0 0.0%	0 0.0%	14 0.3%	0 0.0%	430 8.6%	6 0.1%	0 0.0%	6 0.1%	4 0.1%	1 0.0%	93.3% 6.7%
6	5 0.1%	6 0.1%	1 0.0%	9 0.2%	4 0.1%	428 8.6%	0 0.0%	5 0.1%	1 0.0%	5 0.1%	92.2% 7.8%
7	0 0.0%	10 0.2%	5 0.1%	1 0.0%	4 0.1%	0 0.0%	466 9.3%	8 0.2%	9 0.2%	2 0.0%	92.3% 7.7%
8	0 0.0%	4 0.1%	4 0.1%	1 0.0%	2 0.0%	3 0.1%	1 0.0%	422 8.4%	3 0.1%	1 0.0%	95.7% 4.3%
9	1 0.0%	2 0.0%	1 0.0%	8 0.2%	3 0.1%	1 0.0%	11 0.2%	6 0.1%	466 9.3%	3 0.1%	92.8% 7.2%
10	0 0.0%	6 0.1%	1 0.0%	0 0.0%	3 0.1%	9 0.2%	0 0.0%	5 0.1%	7 0.1%	447 8.9%	93.5% 6.5%
	97.7% 2.3%	92.3% 7.7%	93.4% 6.6%	96.0% 4.0%	94.3% 5.7%	92.6% 7.4%	91.0% 9.0%	86.3% 13.7%	89.6% 10.4%	97.2% 2.8%	93.1% 6.9%

The features are similar in appearance. The performance for “real” images is worse than that for synthetic images.

3. The network architecture, parameters settings, and number of training and test images used are all the same with MATLAB tutorial.

   Network architecture:



parameters settings:

```
autoenc1 = trainAutoencoder(xTrainImages,hiddenSize1, ...
    'MaxEpochs',400, ...
    'L2WeightRegularization',0.004, ...
    'SparsityRegularization',4, ...
    'SparsityProportion',0.15, ...
    'ScaleData', false);

autoenc2 = trainAutoencoder(feat1,hiddenSize2, ...
    'MaxEpochs',100, ...
    'L2WeightRegularization',0.002, ...
    'SparsityRegularization',4, ...
    'SparsityProportion',0.1, ...
    'ScaleData', false);

softnet = trainSoftmaxLayer(feat2,tTrain,'MaxEpochs',400);
```

Number of training and test images used: 5000