Untitled

April 26, 2016

```
In [20]: import scipy.io
         import numpy as np
         import matplotlib.pyplot as plt
         import sklearn
         import pickle
         import csv
         from sklearn import metrics
         np.random.seed(None)
```

Q1: Code.

The part below is the basic network part.

Q2: Train this multi-layer neural network on full training data using stochastic gradient descent. Predict the labels to the test data and submit your results to Kaggle. Please also report the following:

• Parameters that you tuned including learning rate, when you stopped train-ing, how you initialized the weights

Ans: I use the learning rate of 0.01. I include a save and load in the network to output the weight, and I stop training based on each response from the training set. For each epoch I monitor its accuracy. I initialize the weights with a Gaussian with mean 0 and theta 0.1

==========

• Training accuracy and validation accuracy

Ans: Training accuracy in the 50000 sample is about 96%, same for the test set.

==========

• Running-time (Total training time)

Ans: Total running time is about approximate 10 min to run through the code, for the basic network.

==========

• Plots of total training error and classification accuracy on training set vs. iteration. If you find that evaluating error takes a long time, you may compute the error or accuracy every 1000 or so iterations.

Ans: See the plot below.

==========

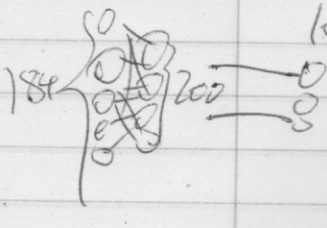
• Comment on the differences in using the two loss functions. Which performs better?

Ans: The sigmoid function give a more continuous performance. With the tanh it seems to be more jumpy on the edge values. Don't know why yet.

```
In [421]: # Don't run this code at first pass.
          # This is just for the answer purpose.
          # The accuracy is noted below, which reach 96.1% on Kaggle.
          # plt.plot(list(map(lambda k:1-k, error_lst2)))
```

```
Out[421]: [<matplotlib.lines.Line2D at 0x111357ac8>]
```

He Yucong 2536890I HW6



10.

$$L = \frac{1}{2}\sum_{i=1}^{10}(\hat{y}_i - y_i)^2$$

Thus

$$\frac{\partial L}{\partial \hat{y}_i} = \hat{y} - y$$

$$\frac{\partial \hat{y}_i}{\partial z^{(3)}} = f_3'(z^{(3)}) = S(z^{(3)})(1 - S(z^{(3)}))$$

$$\frac{\partial z^{(3)}}{\partial w_2} = a^{(2)} \text{ just the weight.}$$

$$\frac{\partial L}{\partial w_2} = \frac{\partial L}{\partial \hat{y}_i}\frac{\partial \hat{y}_i}{\partial z^{(3)}}\frac{\partial z^{(3)}}{\partial w_2} = a^{(2)T}\delta^{(3)}$$

$$= a^{(2)T}(\hat{y} - y)(S(z^{(2)})(1 - S(z^{(3)}))$$

$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial \hat{y}}\frac{\partial \hat{y}}{\partial z^{(3)}}\frac{\partial z^{(3)}}{\partial w_1}$$

$$= \delta^{(3)}\frac{\partial z^{(3)}}{\partial a^{(2)}}\frac{\partial a^{(2)}}{\partial w}$$

$$= \delta^{(3)}(W_2)^T\frac{\partial (a^{(2)})}{\partial z^{(2)}}\frac{\partial z^{(2)}}{\partial w^{(2)}}$$

$$= \delta^{3}(w_2)^T f'(z^{(2)}) \times$$

$$= (w_2^T)\,\delta^{(3)}\,S(z^{(2)})(1 - S(z^{(1)}))\times.$$

S(x) for tanh, just replace

① $f'(z) = S(z)(1 - S(z))$

with

② $\tanh'(z) = \tanh(z)(1 - 2\tanh(z$

Cross Entropy:

$$L = -\sum_{i}^{10}[y_i \ln \hat{y}_i + (1 - y_i)\ln(1 - \hat{y}_i)]$$

$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial \hat{y}}\cdot\frac{\partial \hat{y}}{\partial z^{(3)}}\frac{\partial z^{(3)}}{\partial w_1}$$

$$= \left(\frac{1-y}{1-\hat{y}} - \frac{y}{\hat{y}}\right)\cdot f'(z^{(3)})\cdot a^{(2)}$$

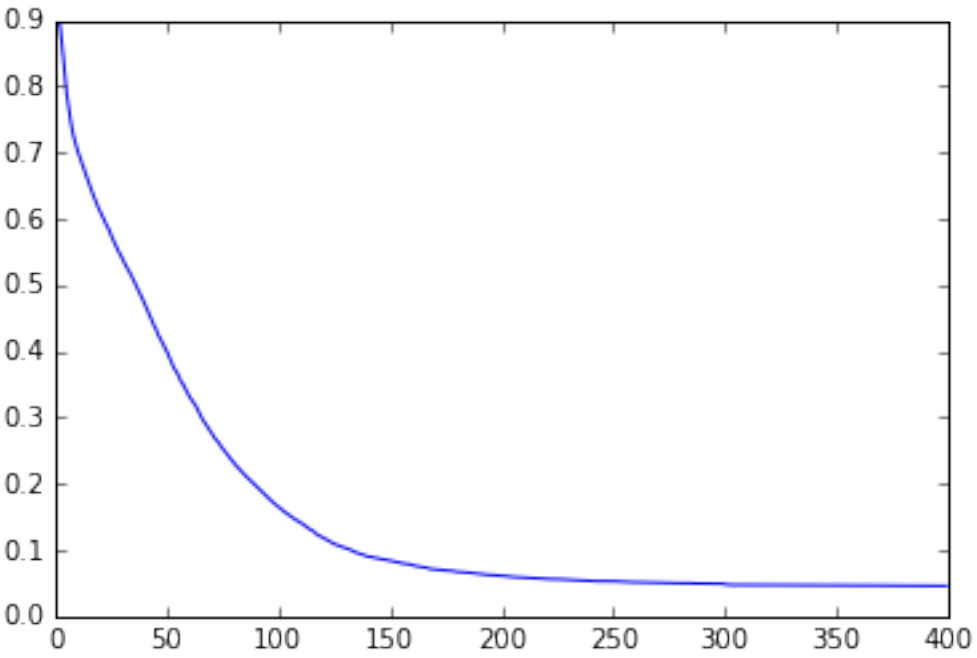
$$= (a^{(2)})^T f'(z^{(3)})\left(\frac{1-y}{1-\hat{y}} - \frac{y}{\hat{y}}\right)$$

$$\frac{\partial L}{\partial w_2} = \frac{\partial L}{\partial \hat{y}}\frac{\partial \hat{y}}{\partial z^{(3)}}\frac{\partial z^{(3)}}{\partial w^{(2)}}$$

$$= (\hat{y} - y)\frac{\partial z^{(3)}}{\partial a^{(2)}}\frac{\partial a^{(2)}}{\partial z^{(2)}}\frac{\partial z^{(2)}}{\partial w^{(2)}}$$

$$= (\hat{y} - y)(w_2)^T(f'(z^{(2)}))\times.$$

if $f'(z)$ = sigmoid, then $f'(z)^3 = (\hat{y})(1-\hat{y})$ cancel out.




```
In [30]: def testPrint(content, arg):
             print(content+"{0}".format(arg))

         def sigmoid(z):
             # Apply sigmoid activation function
             # Support
             return 1/(1 + np.exp(-z))

         def sigmoid_deriv(x):
             return sigmoid(x)*(1-sigmoid(x))

         class QuadraticCost:
             """ Cost functions for quadratic cost. """

             @staticmethod
             def fn(activations, targets):
                 """ Evaluate quadratic cost. """
                 return 0.5 * (activations - targets)**2

             @staticmethod
             def fn_deriv(activations, targets):
                 """ Evaluate derivative of quadratic cost. """
                 return activations - targets

             @staticmethod
             def delta(inputs, activations, targets):
                 """ Compute the delta error at the output layer for the quadratic cost. """
                 return (activations - targets)*sigmoid_deriv(inputs)
```

```
class CrossEntropyCost:
    """ Cost functions for cross entropy cost. """

    @staticmethod
    def fn(activations, targets):
        """ Evaluate cross entropy cost. """
        # The np.nan_to_num function ensures that np.log(0) evaluates to 0 instead of nan.
        return -np.nan_to_num(targets*np.log(activations) + \
                                                (1-targets)*np.log(1 - activations))
    @staticmethod
    def fn_deriv(activations, targets):
        """ Evaluate the derivative of the cross entropy cost. """
        return -np.nan_to_num(targets/activations - (1-targets)/(1-activations))

    @staticmethod
    def delta(inputs, activations, targets):
        """ Compute the delta error at the output layer for the cross entropy cost. """
        return (activations-targets)

class Network(object):

    """
    Synapse: take a value and multiply by its weight.
    Neuron: z = x1 + x2 + x3...
            a = 1/(1 + e^-z)
    """
    def __init__(self, cost=CrossEntropyCost):
        self.inputLayerSize = 784
        self.outputLayerSize = 10
        self.hiddenLayerSize = 200

        self.num_of_layers = 1

        self.cost = cost

        self.W1 = 0.1 * np.random.randn(self.inputLayerSize, self.hiddenLayerSize)
        self.W2 = 0.1 * np.random.randn(self.hiddenLayerSize, self.outputLayerSize)
        self.bias2 = 0.1 * np.random.randn(self.hiddenLayerSize)
        self.bias3 = 0.1 * np.random.randn(self.outputLayerSize)

    def forward(self, X):
        """
        Propagate multiple input through matrix
        Do matrix multiplication by input layer and weight on synapse.

        X * W(1) = Z(2)
        Z is row(example)(784) * col(hidden)(200)

        a(2) = s(Z(2))
        a is 200 x 200

        z(3) = a(2)W(2)
        z(3) is 200 * 10
```

```
        y = f(z(3))
        """
        self.z2 = np.dot(X, self.W1) + self.bias2
        self.a2 = sigmoid(self.z2)
        self.z3 = np.dot(self.a2, self.W2) + self.bias3
        yHat = sigmoid(self.z3)
        return yHat

    def cost_function_prime(self, X, y):
        self.yHat = self.forward(X)

        #delta3 = np.multiply(-(y - self.yHat), self.sigmoid_deriv(self.z3))
        delta3 = (self.cost).delta(self.z3, self.yHat, y)
        dJdW2 = np.dot(self.a2.T, delta3)

        delta2 = np.multiply(np.dot(delta3, self.W2.T), sigmoid_deriv(self.z2))
        dJdW1 = np.dot(X.T, delta2)

        # deltas are for bias term.
        return delta2, delta3, dJdW1, dJdW2

    def train_mini_batch(self, X, y, rate, L2):
        """ Train the network on a mini-batch """
        n = len(y)
        self.delta2, self.delta3, self.dJdW1, self.dJdW2 = self.cost_function_prime(X, y)
        self.bias2 -= (rate)*np.mean(self.delta2, axis=0)
        self.bias3 -= (rate)*np.mean(self.delta3, axis=0)
        self.W1 -= (rate/n)*self.dJdW1 - rate*L2*self.W1
        self.W2 -= (rate/n)*self.dJdW2 - rate*L2*self.W2

    def stochastic_gradient_descent(self, X, y, number_of_epochs, mini_batch_size, \
                                            rate = 1, L2 = 0.1, test_X = None, test_y = None, ls
        """ Train the network using the stochastic gradient descent method. """
        for epoch in range(number_of_epochs):
            indexes = np.random.shuffle(np.arange(len(y)))
            X_train = X[indexes]
            y_train = y[indexes]
            batches = [(X[x:x+mini_batch_size], y[x:x+mini_batch_size]) \
                        for x in np.arange(0, len(y), mini_batch_size)]

            for batch in batches:
                self.train_mini_batch( batch[0], batch[1], rate, L2)

            if test_X != None and test_y != None:
                result = self.evaluate(test_X, test_y)
                print("Epoch {0}: {1} / {2}".format(epoch, result, \
                                                    len(test_y)))
                lst.append(float(result) / len(test_y))
        return lst

    def evaluate(self, test_X, test_y):
        """ Evaluate performance by counting how many examples in test_data are correctly
            evaluated. """
        count = 0
```

```
        for i in range(len(test_y)):
            answer = np.argmax(test_y[i])
            prediction = np.argmax(self.forward(test_X[i]))
            count = count + 1 if (answer - prediction) == 0 else count
        return count

    def save(self, filename):
        """ Save neural network (weights) to a file. """
        with open(filename, 'wb') as f:
            pickle.dump({'b2':self.bias2, 'b3':self.bias3, 'W1':self.W1, 'W2':self.W2}, f )

    def load(self, filename):
        """ Load neural network (weights) from a file. """
        with open(filename, 'rb') as f:
            data = pickle.load(f)

        # Set biases and weights
        self.W1 = data['W1']
        self.W2 = data['W2']
        self.bias2 = data['b2']
        self.bias3 = data['b3']

    """Numerical checking part. """
    def getParams(self):
        params = np.concatenate((self.W1.ravel(), self.W2.ravel()))
        return params

    def setParams(self, params):
        # Set W1 and W2 using single parameter vector:
        W1_start = 0
        W1_end = self.hiddenLayerSize * self.inputLayerSize

        self.W1 = np.reshape(params[W1_start:W1_end], (self.inputLayerSize, self.hiddenLayerSi
        W2_end = W1_end + self.hiddenLayerSize * self.outputLayerSize
        self.W2 = np.reshape(params[W1_end:W2_end], (self.hiddenLayerSize, self.outputLayerSiz

    def computeGradients(self, X, y):
        delta2, delta3, dJdW1, dJdW2 = self.cost_function_prime(X, y)
        testPrint("dJdW1", dJdW1.shape)
        testPrint("dJdW2", dJdW2.shape)
        return np.concatenate((dJdW1.ravel(), dJdW2.ravel()))

"""Method for numerical check."""
def computeNumericalGradient(N, X, y, indexes):
    paramsInitial = N.getParams()
    numgrad = np.zeros(paramsInitial.shape)
    perturb = np.zeros(paramsInitial.shape)

    e = 1e-4
    for p in indexes:
        # Set perturbation error
        perturb[p] = e
        N.setParams(paramsInitial + perturb)
        yHat = N.forward(X)
```

```
            loss2 = sum((N.cost).fn(yHat, y))

            N.setParams(paramsInitial - perturb)
            yHat = N.forward(X)
            loss1 = sum((N.cost).fn(yHat, y))

            d = (loss2 - loss1) / (2 * e)
            testPrint("d", sum(d))

            numgrad[p] = sum(d)
            perturb[p] = 0
        N.setParams(paramsInitial)
        return numgrad
```

In [4]:
```
%matplotlib inline
train_data = scipy.io.loadmat("./dataset/train.mat")
test_data = scipy.io.loadmat("./dataset/test.mat")
# _mat = sklearn.preprocessing.scale(_mat)
# test_mat = _mat[5172:]
# _mat = _mat[0:5172]

# print(test_mat.shape)
# print(_mat.shape)
# _labels = spamData['training_labels'].reshape(5172)
# #
# v_index = list(set(np.random.randint(5172, size=1724)))
# t_index = list(set(range(5172)) - set(v_index))
# v_mat = _mat[v_index]
# v_labels = _labels[v_index]
# t_mat = _mat[t_index]
# t_labels = _labels[t_index]
```

In [5]:
```
print(sklearn.__version__)
```

```
0.17.1
```

In [6]:
```
from sklearn import preprocessing
train_images = train_data['train_images'].transpose(2, 0, 1) # Correct rotation for the data. (
test_images = test_data['test_images'] # Correct rotation for the data. (10000, 28, 28)
# _mat = np.hstack((np.vstack((train_images, test_images)).reshape(70000, 784), np.ones((70000,
_mat = np.vstack((train_images, test_images)).reshape(70000, 784)
_mat = sklearn.preprocessing.scale(_mat)
train_mat = _mat[:60000]
test_mat = _mat[60000:]
_labels = list(train_data['train_labels'].reshape(60000))
_labels = np.array([[1 if i == _labels[j] else 0 for i in range(10)] for j in range(60000)])
print(_labels.shape)

v_index = list(np.random.choice(np.arange(60000), size=10000, replace=False))
t_index = list(set(range(60000)) - set(v_index))
v_mat = train_mat[v_index]
v_labels = _labels[v_index]
t_mat = train_mat[t_index]
t_labels = _labels[t_index]
```

```
/Users/yuconghe/anaconda3/lib/python3.5/site-packages/sklearn/utils/validation.py:420: DataConversionWa
  warnings.warn(msg, DataConversionWarning)

(60000, 10)
```

```
In [7]: import pickle
        """
        After some training we got test1 and test2 as two weight documents.
        """
```

```
Out[7]: '\nAfter some training we got test1 and test2 as two weight documents.\n'
```

```
In [413]: NN1 = Network()
          np.random.seed(None)
          error_lst = NN1.stochastic_gradient_descent(t_mat, t_labels, 300, 10, 0.01, 0.001/len(t_labels
                                                test_X = v_mat, test_y = v_labels)
```

```
/Users/yuconghe/anaconda3/lib/python3.5/site-packages/ipykernel/__main__.py:124: FutureWarning: comparis

Epoch 0: 1024 / 10000
Epoch 1: 1024 / 10000
Epoch 2: 1081 / 10000
Epoch 3: 1399 / 10000
Epoch 4: 1781 / 10000
Epoch 5: 2117 / 10000
Epoch 6: 2337 / 10000
Epoch 7: 2572 / 10000
Epoch 8: 2744 / 10000
Epoch 9: 2871 / 10000
Epoch 10: 2986 / 10000
Epoch 11: 3084 / 10000
Epoch 12: 3178 / 10000
Epoch 13: 3284 / 10000
Epoch 14: 3379 / 10000
Epoch 15: 3485 / 10000
Epoch 16: 3562 / 10000
Epoch 17: 3647 / 10000
Epoch 18: 3740 / 10000
Epoch 19: 3821 / 10000
Epoch 20: 3889 / 10000
Epoch 21: 3962 / 10000
Epoch 22: 4036 / 10000
Epoch 23: 4108 / 10000
Epoch 24: 4167 / 10000
Epoch 25: 4258 / 10000
Epoch 26: 4334 / 10000
Epoch 27: 4400 / 10000
Epoch 28: 4478 / 10000
Epoch 29: 4540 / 10000
Epoch 30: 4604 / 10000
Epoch 31: 4676 / 10000
Epoch 32: 4736 / 10000
Epoch 33: 4805 / 10000
Epoch 34: 4859 / 10000
Epoch 35: 4928 / 10000
```

```
Epoch 36: 4998 / 10000
Epoch 37: 5064 / 10000
Epoch 38: 5140 / 10000
Epoch 39: 5213 / 10000
Epoch 40: 5288 / 10000
Epoch 41: 5361 / 10000
Epoch 42: 5443 / 10000
Epoch 43: 5520 / 10000
Epoch 44: 5603 / 10000
Epoch 45: 5665 / 10000
Epoch 46: 5755 / 10000
Epoch 47: 5812 / 10000
Epoch 48: 5884 / 10000
Epoch 49: 5949 / 10000
Epoch 50: 6012 / 10000
Epoch 51: 6094 / 10000
Epoch 52: 6175 / 10000
Epoch 53: 6249 / 10000
Epoch 54: 6304 / 10000
Epoch 55: 6357 / 10000
Epoch 56: 6443 / 10000
Epoch 57: 6489 / 10000
Epoch 58: 6557 / 10000
Epoch 59: 6627 / 10000
Epoch 60: 6685 / 10000
Epoch 61: 6733 / 10000
Epoch 62: 6786 / 10000
Epoch 63: 6831 / 10000
Epoch 64: 6905 / 10000
Epoch 65: 6974 / 10000
Epoch 66: 7034 / 10000
Epoch 67: 7081 / 10000
Epoch 68: 7135 / 10000
Epoch 69: 7190 / 10000
Epoch 70: 7236 / 10000
Epoch 71: 7294 / 10000
Epoch 72: 7338 / 10000
Epoch 73: 7373 / 10000
Epoch 74: 7429 / 10000
Epoch 75: 7474 / 10000
Epoch 76: 7514 / 10000
Epoch 77: 7561 / 10000
Epoch 78: 7603 / 10000
Epoch 79: 7646 / 10000
Epoch 80: 7684 / 10000
Epoch 81: 7728 / 10000
Epoch 82: 7766 / 10000
Epoch 83: 7807 / 10000
Epoch 84: 7842 / 10000
Epoch 85: 7875 / 10000
Epoch 86: 7908 / 10000
Epoch 87: 7941 / 10000
Epoch 88: 7974 / 10000
Epoch 89: 8003 / 10000
```

```
Epoch 90: 8035 / 10000
Epoch 91: 8072 / 10000
Epoch 92: 8110 / 10000
Epoch 93: 8142 / 10000
Epoch 94: 8177 / 10000
Epoch 95: 8202 / 10000
Epoch 96: 8236 / 10000
Epoch 97: 8267 / 10000
Epoch 98: 8298 / 10000
Epoch 99: 8327 / 10000
Epoch 100: 8352 / 10000
Epoch 101: 8377 / 10000
Epoch 102: 8408 / 10000
Epoch 103: 8432 / 10000
Epoch 104: 8455 / 10000
Epoch 105: 8482 / 10000
Epoch 106: 8503 / 10000
Epoch 107: 8527 / 10000
Epoch 108: 8553 / 10000
Epoch 109: 8567 / 10000
Epoch 110: 8592 / 10000
Epoch 111: 8611 / 10000
Epoch 112: 8639 / 10000
Epoch 113: 8664 / 10000
Epoch 114: 8688 / 10000
Epoch 115: 8704 / 10000
Epoch 116: 8728 / 10000
Epoch 117: 8762 / 10000
Epoch 118: 8779 / 10000
Epoch 119: 8795 / 10000
Epoch 120: 8817 / 10000
Epoch 121: 8834 / 10000
Epoch 122: 8853 / 10000
Epoch 123: 8874 / 10000
Epoch 124: 8894 / 10000
Epoch 125: 8909 / 10000
Epoch 126: 8922 / 10000
Epoch 127: 8937 / 10000
Epoch 128: 8951 / 10000
Epoch 129: 8958 / 10000
Epoch 130: 8971 / 10000
Epoch 131: 8982 / 10000
Epoch 132: 8993 / 10000
Epoch 133: 9011 / 10000
Epoch 134: 9029 / 10000
Epoch 135: 9039 / 10000
Epoch 136: 9051 / 10000
Epoch 137: 9064 / 10000
Epoch 138: 9074 / 10000
Epoch 139: 9085 / 10000
Epoch 140: 9098 / 10000
Epoch 141: 9100 / 10000
Epoch 142: 9109 / 10000
Epoch 143: 9116 / 10000
```

```
Epoch 144: 9122 / 10000
Epoch 145: 9127 / 10000
Epoch 146: 9135 / 10000
Epoch 147: 9138 / 10000
Epoch 148: 9142 / 10000
Epoch 149: 9148 / 10000
Epoch 150: 9160 / 10000
Epoch 151: 9167 / 10000
Epoch 152: 9176 / 10000
Epoch 153: 9182 / 10000
Epoch 154: 9187 / 10000
Epoch 155: 9199 / 10000
Epoch 156: 9203 / 10000
Epoch 157: 9203 / 10000
Epoch 158: 9209 / 10000
Epoch 159: 9216 / 10000
Epoch 160: 9228 / 10000
Epoch 161: 9238 / 10000
Epoch 162: 9243 / 10000
Epoch 163: 9251 / 10000
Epoch 164: 9255 / 10000
Epoch 165: 9262 / 10000
Epoch 166: 9265 / 10000
Epoch 167: 9275 / 10000
Epoch 168: 9285 / 10000
Epoch 169: 9287 / 10000
Epoch 170: 9290 / 10000
Epoch 171: 9296 / 10000
Epoch 172: 9296 / 10000
Epoch 173: 9301 / 10000
Epoch 174: 9302 / 10000
Epoch 175: 9308 / 10000
Epoch 176: 9310 / 10000
Epoch 177: 9312 / 10000
Epoch 178: 9315 / 10000
Epoch 179: 9325 / 10000
Epoch 180: 9329 / 10000
Epoch 181: 9330 / 10000
Epoch 182: 9331 / 10000
Epoch 183: 9336 / 10000
Epoch 184: 9337 / 10000
Epoch 185: 9341 / 10000
Epoch 186: 9347 / 10000
Epoch 187: 9348 / 10000
Epoch 188: 9355 / 10000
Epoch 189: 9357 / 10000
Epoch 190: 9360 / 10000
Epoch 191: 9365 / 10000
Epoch 192: 9370 / 10000
Epoch 193: 9374 / 10000
Epoch 194: 9375 / 10000
Epoch 195: 9375 / 10000
Epoch 196: 9376 / 10000
Epoch 197: 9380 / 10000
```

```
Epoch 198: 9385 / 10000
Epoch 199: 9387 / 10000
Epoch 200: 9390 / 10000
Epoch 201: 9390 / 10000
Epoch 202: 9395 / 10000
Epoch 203: 9400 / 10000
Epoch 204: 9402 / 10000
Epoch 205: 9406 / 10000
Epoch 206: 9407 / 10000
Epoch 207: 9408 / 10000
Epoch 208: 9407 / 10000
Epoch 209: 9410 / 10000
Epoch 210: 9414 / 10000
Epoch 211: 9416 / 10000
Epoch 212: 9420 / 10000
Epoch 213: 9421 / 10000
Epoch 214: 9421 / 10000
Epoch 215: 9425 / 10000
Epoch 216: 9426 / 10000
Epoch 217: 9428 / 10000
Epoch 218: 9431 / 10000
Epoch 219: 9433 / 10000
Epoch 220: 9436 / 10000
Epoch 221: 9438 / 10000
Epoch 222: 9439 / 10000
Epoch 223: 9438 / 10000
Epoch 224: 9439 / 10000
Epoch 225: 9439 / 10000
Epoch 226: 9441 / 10000
Epoch 227: 9443 / 10000
Epoch 228: 9445 / 10000
Epoch 229: 9446 / 10000
Epoch 230: 9447 / 10000
Epoch 231: 9448 / 10000
Epoch 232: 9451 / 10000
Epoch 233: 9451 / 10000
Epoch 234: 9453 / 10000
Epoch 235: 9455 / 10000
Epoch 236: 9458 / 10000
Epoch 237: 9462 / 10000
Epoch 238: 9466 / 10000
Epoch 239: 9467 / 10000
Epoch 240: 9468 / 10000
Epoch 241: 9470 / 10000
Epoch 242: 9471 / 10000
Epoch 243: 9471 / 10000
Epoch 244: 9472 / 10000
Epoch 245: 9472 / 10000
Epoch 246: 9470 / 10000
Epoch 247: 9470 / 10000
Epoch 248: 9470 / 10000
Epoch 249: 9471 / 10000
Epoch 250: 9474 / 10000
Epoch 251: 9473 / 10000
```

```
Epoch 252: 9475 / 10000
Epoch 253: 9477 / 10000
Epoch 254: 9478 / 10000
Epoch 255: 9481 / 10000
Epoch 256: 9483 / 10000
Epoch 257: 9483 / 10000
Epoch 258: 9484 / 10000
Epoch 259: 9485 / 10000
Epoch 260: 9484 / 10000
Epoch 261: 9485 / 10000
Epoch 262: 9486 / 10000
Epoch 263: 9486 / 10000
Epoch 264: 9485 / 10000
Epoch 265: 9487 / 10000
Epoch 266: 9488 / 10000
Epoch 267: 9489 / 10000
Epoch 268: 9490 / 10000
Epoch 269: 9491 / 10000
Epoch 270: 9494 / 10000
Epoch 271: 9494 / 10000
Epoch 272: 9494 / 10000
Epoch 273: 9495 / 10000
Epoch 274: 9495 / 10000
Epoch 275: 9495 / 10000
Epoch 276: 9496 / 10000
Epoch 277: 9497 / 10000
Epoch 278: 9496 / 10000
Epoch 279: 9497 / 10000
Epoch 280: 9498 / 10000
Epoch 281: 9500 / 10000
Epoch 282: 9499 / 10000
Epoch 283: 9500 / 10000
Epoch 284: 9499 / 10000
Epoch 285: 9501 / 10000
Epoch 286: 9502 / 10000
Epoch 287: 9503 / 10000
Epoch 288: 9504 / 10000
Epoch 289: 9504 / 10000
Epoch 290: 9504 / 10000
Epoch 291: 9506 / 10000
Epoch 292: 9506 / 10000
Epoch 293: 9508 / 10000
Epoch 294: 9508 / 10000
Epoch 295: 9508 / 10000
Epoch 296: 9508 / 10000
Epoch 297: 9509 / 10000
Epoch 298: 9509 / 10000
Epoch 299: 9509 / 10000
```

```
In [415]: error_lst2 = NN1.stochastic_gradient_descent(t_mat, t_labels, 50, 10, 0.005, 0.001/len(t_label
                                                    test_X = v_mat, test_y = v_labels)
```

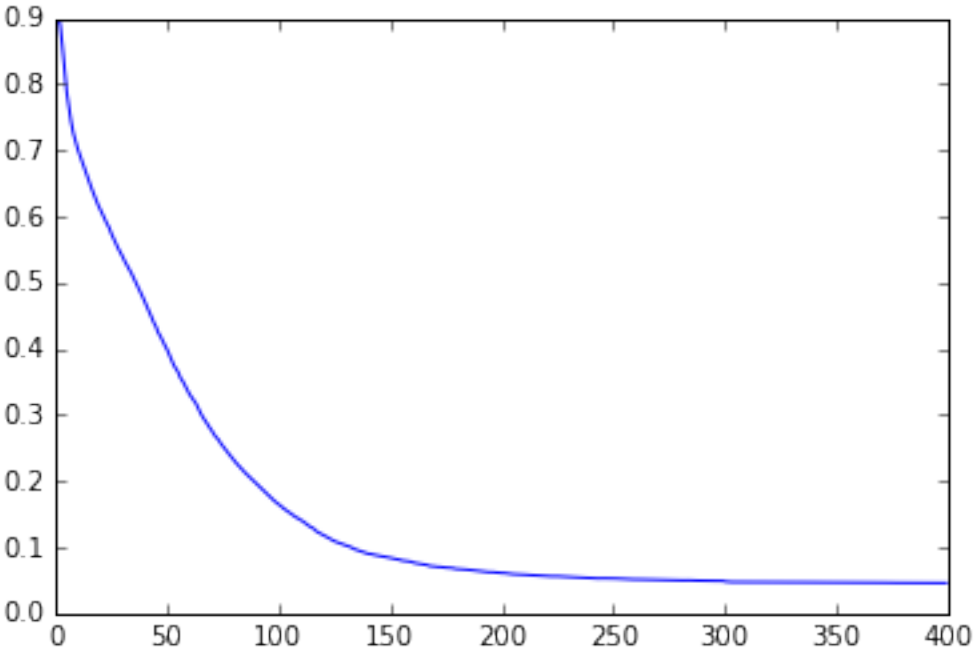
```
/Users/yuconghe/anaconda3/lib/python3.5/site-packages/ipykernel/__main__.py:124: FutureWarning: comparis
```

```
Epoch 0: 9531 / 10000
Epoch 1: 9531 / 10000
```

```
Epoch 2: 9531 / 10000
Epoch 3: 9530 / 10000
Epoch 4: 9530 / 10000
Epoch 5: 9532 / 10000
Epoch 6: 9532 / 10000
Epoch 7: 9532 / 10000
Epoch 8: 9532 / 10000
Epoch 9: 9532 / 10000
Epoch 10: 9532 / 10000
Epoch 11: 9532 / 10000
Epoch 12: 9532 / 10000
Epoch 13: 9533 / 10000
Epoch 14: 9534 / 10000
Epoch 15: 9534 / 10000
Epoch 16: 9534 / 10000
Epoch 17: 9534 / 10000
Epoch 18: 9534 / 10000
Epoch 19: 9534 / 10000
Epoch 20: 9534 / 10000
Epoch 21: 9534 / 10000
Epoch 22: 9535 / 10000
Epoch 23: 9535 / 10000
Epoch 24: 9535 / 10000
Epoch 25: 9536 / 10000
Epoch 26: 9536 / 10000
Epoch 27: 9536 / 10000
Epoch 28: 9536 / 10000
Epoch 29: 9536 / 10000
Epoch 30: 9536 / 10000
Epoch 31: 9536 / 10000
Epoch 32: 9536 / 10000
Epoch 33: 9537 / 10000
Epoch 34: 9538 / 10000
Epoch 35: 9538 / 10000
Epoch 36: 9538 / 10000
Epoch 37: 9537 / 10000
Epoch 38: 9538 / 10000
Epoch 39: 9538 / 10000
Epoch 40: 9538 / 10000
Epoch 41: 9538 / 10000
Epoch 42: 9538 / 10000
Epoch 43: 9538 / 10000
Epoch 44: 9538 / 10000
Epoch 45: 9538 / 10000
Epoch 46: 9539 / 10000
Epoch 47: 9539 / 10000
Epoch 48: 9539 / 10000
Epoch 49: 9539 / 10000
```

```
In [424]: plt.plot(list(map(lambda k:1-k, error_lst2)))
          NN1.save("test2")
          NN.load("FinalCell96Ver1")
          yHat = NN.forward(v_mat)
          # predicted = np.argmax(yHat, axis = 1)
```

```
# expected = np.argmax(v_labels, axis = 1)
# s = metrics.accuracy_score(expected, predicted)
# m = metrics.confusion_matrix(expected, predicted)
# print(s)
# print(m)
prediction = np.argmax(NN.forward(test_mat), axis=1)
import csv
with open('predicted_digies.csv', 'w') as csvfile:
    writer = csv.writer(csvfile, lineterminator = '\n')
    for i, content in enumerate(prediction):
        writer.writerow([i+1, content])
# NN1.stochastic_gradient_descent(t_mat, t_labels, 250, 100, 0.05, 0.005, \
#                                 test_X = v_mat, test_y = v_labels)
```



In [303]:
```
"""Testing numerical Gradient."""
NN = Network()
# yHat = NN.forward(t_mat)
grad = NN.computeGradients(t_mat, t_labels)
indexes = np.random.randint(low=30000, high=150000, size=10)
numgrad = computeNumericalGradient(NN, t_mat, t_labels, indexes)
print(grad[indexes].shape)
print(numgrad[indexes].shape)
print(grad[indexes])
print(numgrad[indexes])
```

```
/Users/yuconghe/anaconda3/lib/python3.5/site-packages/ipykernel/__main__.py:7: RuntimeWarning: overflow e

(50826, 200) (50826, 10) (784, 200) (200, 10)
dJdW1(784, 200)
dJdW2(200, 10)
```

```
d-14.590899957056536
d2.5801350147958146
d-12.756509640894365
d0.10754162076409557
d1.2244919344084337
d-13.09003203004977
d-0.6733428722327517
d12.701527900844667
d5.067685081030504
d16.610999125532544
(10,)
(10,)
[-14.59089841   2.58013447 -12.75650932   0.1075408    1.22449189
 -13.09003461  -0.67334359  12.70152759   5.0676839   16.61099764]
[-14.59089996   2.58013501 -12.75650964   0.10754162   1.22449193
 -13.09003203  -0.67334287  12.7015279    5.06768508  16.61099913]
```

We can see that both grad and numgrad are the same. Here we randomly choose some axis and compute their direction.

```
In [285]: """Testing Box."""
          NN = Network()
          # yHat = NN.forward(t_mat)
          # dJdW1, dJdW2 = NN.cost_function_prime(t_mat, t_labels)
          # cost = NN.cost.fn(yHat, t_labels)
          # print(sum(cost))
          # scalar = 0.001
          # predicted = np.argmax(yHat, axis = 1)
          # expected = np.argmax(t_labels, axis = 1)
          # s = metrics.accuracy_score(expected, predicted)
          # m = metrics.confusion_matrix(expected, predicted)
          # print(s)
          # print(m)
          # NN.W1 = NN.W1 + scalar * dJdW1
          # NN.W2 = NN.W2 + scalar * dJdW2
          # yHat = NN.forward(t_mat)
          # cost = NN.cost.fn(yHat, t_labels)
          # print(sum(cost))
          # predicted = np.argmax(yHat, axis = 1)
          # expected = np.argmax(t_labels, axis = 1)
          # s = metrics.accuracy_score(expected, predicted)
          # m = metrics.confusion_matrix(expected, predicted)
          # print(s)
          # print(m)
```

Below is the upgraded Neural Network which uses convoluted layers and different tuning values. Code below this point is largely experimental, and not meant to be graded.

```
In [8]: def testPrint(content, arg):
            print(content+"{0}".format(arg))

        def sigmoid(z):
            # Apply sigmoid activation function
            # Support
            return 1/(1 + np.exp(-z))
```

```
def sigmoid_deriv(x):
    return sigmoid(x)*(1-sigmoid(x))

class QuadraticCost:
    """ Cost functions for quadratic cost. """

    @staticmethod
    def fn(activations, targets):
        """ Evaluate quadratic cost. """
        return 0.5 * (activations - targets)**2

    @staticmethod
    def fn_deriv(activations, targets):
        """ Evaluate derivative of quadratic cost. """
        return activations - targets

    @staticmethod
    def delta(inputs, activations, targets):
        """ Compute the delta error at the output layer for the quadratic cost. """
        return (activations - targets)*sigmoid_deriv(inputs)

class CrossEntropyCost:
    """ Cost functions for cross entropy cost. """

    @staticmethod
    def fn(activations, targets):
        """ Evaluate cross entropy cost. """
        # The np.nan_to_num function ensures that np.log(0) evaluates to 0 instead of nan.
        return -np.nan_to_num(targets*np.log(activations) + \
                                                (1-targets)*np.log(1 - activations))
    @staticmethod
    def fn_deriv(activations, targets):
        """ Evaluate the derivative of the cross entropy cost. """
        return -np.nan_to_num(targets/activations - (1-targets)/(1-activations))

    @staticmethod
    def delta(inputs, activations, targets):
        """ Compute the delta error at the output layer for the cross entropy cost. """
        return (activations-targets)

class ComplexNetwork(object):

    """
    Synapse: take a value and multiply by its weight.
    Neuron: z = x1 + x2 + x3...
            a = 1/(1 + e^-z)
    """
    def __init__(self, cost=QuadraticCost):
        self.inputLayerSize = 784
        self.outputLayerSize = 10
        self.hiddenLayer1Size = 1200
        self.hiddenLayer2Size = 1200
```

```
        self.num_of_layers = 2

        self.cost = cost

        self.W1 = 0.1 * np.random.randn(self.inputLayerSize, self.hiddenLayer1Size)
        self.W2 = 0.1 * np.random.randn(self.hiddenLayer1Size, self.hiddenLayer2Size)
        self.W3 = 0.1 * np.random.randn(self.hiddenLayer2Size, self.outputLayerSize)
        self.bias2 = 0.1 * np.random.randn(self.hiddenLayer1Size)
        self.bias3 = 0.1 * np.random.randn(self.hiddenLayer2Size)
        self.bias4 = 0.1 * np.random.randn(self.outputLayerSize)

    def forward(self, X, dropout=False):
        """
        Propagate multiple input through matrix
        Do matrix multiplication by input layer and weight on synapse.

        X * W(1) = Z(2)
        Z is row(example)(784) * col(hidden)(200)

        a(2) = s(Z(2))
        a is 200 x 200

        z(3) = a(2)W(2)
        z(3) is 200 * 10

        y = f(z(3))
        """
        self.z2 = np.dot(X, self.W1) + self.bias2
        self.a2 = sigmoid(self.z2)
        if dropout:
            m2 = np.random.binomial(1, 0.5, size=z2.shape)
        else:
            m2 = 1
        self.a2 *= m2
        self.z3 = np.dot(self.a2, self.W2) + self.bias3
        self.a3 = sigmoid(self.z3)
        if dropout:
            m3 = np.random.binomial(1, 0.5, size=z3.shape)
        else:
            m3 = 1
        self.z4 = np.dot(self.a3, self.W3) + self.bias4
        yHat = sigmoid(self.z4)
        return yHat,m2,m3

    def cost_function_prime(self, X, y, m2=1, m3=1):
        self.yHat,m2,m3 = self.forward(X)

        #delta3 = np.multiply(-(y - self.yHat), self.sigmoid_deriv(self.z3))
        delta4 = (self.cost).delta(self.z4, self.yHat, y)
        dJdW3 = np.dot(self.a3.T, delta4)

        delta3 = np.multiply(np.dot(delta4, self.W3.T), sigmoid_deriv(self.z3)) * m3
        dJdW2 = np.dot(self.a2.T, delta3)
```

```
        delta2 = np.multiply(np.dot(delta3, self.W2.T), sigmoid_deriv(self.z2)) * m2
        dJdW1 = np.dot(X.T, delta2)

        # deltas are for bias term.
        return delta2, delta3, delta4, dJdW1, dJdW2, dJdW3

    def train_mini_batch(self, X, y, rate, L2, momentum):
        """ Train the network on a mini-batch """
        n = len(y)
        self.delta2, self.delta3, self.delta4, self.dJdW1, self.dJdW2, self.dJdW3 = self.cost_fu
        self.bias2 -= (rate)*np.mean(self.delta2, axis=0)
        self.bias3 -= (rate)*np.mean(self.delta3, axis=0)
        self.bias4 -= (rate)*np.mean(self.delta4, axis=0)
        self.W1 -= (rate/n)*self.dJdW1*(1-momentum) - rate*L2*self.W1*momentum
        self.W2 -= (rate/n)*self.dJdW2*(1-momentum) - rate*L2*self.W2*momentum
        self.W3 -= (rate/n)*self.dJdW3*(1-momentum) - rate*L2*self.W3*momentum

    def calculate_momentum(self, iteration, pi=0.5, pf=0.99, T=500.0):
        if iteration < T:
            return iteration/T*pi + (1-iteration/T)*pf
        else:
            return pf

    def stochastic_gradient_descent(self, X, y, number_of_epochs, mini_batch_size, \
                                            rate = 10, L2 = 0.1, test_X = None, test_y = None, t=
        """ Train the network using the stochastic gradient descent method. """
        for epoch in range(number_of_epochs):
            rate *= 0.998
            indexes = np.random.shuffle(np.arange(len(y)))
            X_train = X[indexes]
            y_train = y[indexes]
            batches = [(X[x:x+mini_batch_size], y[x:x+mini_batch_size]) \
                        for x in np.arange(0, len(y), mini_batch_size)]

            momentum = self.calculate_momentum(epoch+t)
            for batch in batches:
                self.train_mini_batch( batch[0], batch[1], rate, L2, momentum)

            if test_X != None and test_y != None:
                result = self.evaluate(test_X, test_y)
                print("Epoch {0}: {1} / {2}".format(epoch, result, \
                                                    len(test_y)))
            if epoch % 10 == 0:
                self.save(filename)
        return rate

    def evaluate(self, test_X, test_y):
        """ Evaluate performance by counting how many examples in test_data are correctly
            evaluated. """
        count = 0
        for i in range(len(test_y)):
            answer = np.argmax(test_y[i])
            prediction = np.argmax(self.forward(test_X[i])[0])
            count = count + 1 if (answer - prediction) == 0 else count
```

```
        return count

    def save(self, filename):
        """ Save neural network (weights) to a file. """
        with open(filename, 'wb') as f:
            pickle.dump({'b2':self.bias2, 'b3':self.bias3, 'b4':self.bias4, 'W1':self.W1, 'W2':s

    def load(self, filename):
        """ Load neural network (weights) from a file. """
        with open(filename, 'rb') as f:
            data = pickle.load(f)

        # Set biases and weights
        self.W1 = data['W1']
        self.W2 = data['W2']
        self.W3 = data['W3']
        self.bias2 = data['b2']
        self.bias3 = data['b3']
        self.bias4 = data['b4']

#     """Numerical checking part. """
#     def getParams(self):
#         params = np.concatenate((self.W1.ravel(), self.W2.ravel()))
#         return params

#     def setParams(self, params):
#         # Set W1 and W2 using single parameter vector:
#         W1_start = 0
#         W1_end = self.hiddenLayerSize * self.inputLayerSize

#         self.W1 = np.reshape(params[W1_start:W1_end], (self.inputLayerSize, self.hiddenLayerS
#         W2_end = W1_end + self.hiddenLayerSize * self.outputLayerSize
#         self.W2 = np.reshape(params[W1_end:W2_end], (self.hiddenLayerSize, self.outputLayerSi

#     def computeGradients(self, X, y):
#         delta2, delta3, delta4, dJdW1, dJdW2, dJdW3 = self.cost_function_prime(X, y)
#         return np.concatenate((dJdW1.ravel(), dJdW2.ravel(), dJdW3.ravel()))

# """Method for numerical check."""
# def computeNumericalGradient(N, X, y, indexes):
#     paramsInitial = N.getParams()
#     numgrad = np.zeros(paramsInitial.shape)
#     perturb = np.zeros(paramsInitial.shape)

#     e = 1e-4
#     for p in indexes:
#         # Set perturbation error
#         perturb[p] = e
#         N.setParams(paramsInitial + perturb)
#         yHat = N.forward(X)
#         loss2 = sum((N.cost).fn(yHat, y))

#         N.setParams(paramsInitial - perturb)
#         yHat = N.forward(X)
```

```
        #         loss1 = sum((N.cost).fn(yHat, y))

        #         d = (loss2 - loss1) / (2 * e)
        #         testPrint("d", sum(d))

        #         numgrad[p] = sum(d)
        #         perturb[p] = 0
        #     N.setParams(paramsInitial)
        #     return numgrad
```

In [46]:
```
# np.random.seed(None)
# val = 0
# NN_lst = []
v_index = list(np.random.choice(np.arange(60000), size=10000, replace=False))
t_index = list(set(range(60000)) - set(v_index))
v_mat = train_mat[v_index]
v_labels = _labels[v_index]
t_mat = train_mat[t_index]
t_labels = _labels[t_index]
rate = 10
val = 0
for i in range(3, 5):
    CNN = ComplexNetwork()
    file_name = "test2_CNN_" + str(i)
    rate = CNN.stochastic_gradient_descent(t_mat, t_labels, 300, 100, rate, 0.01/len(t_labels)
                                           test_X = v_mat, test_y = v_labels, t=val, filename = file_na
    CNN.save(file_name)
    val += 100
```

```
/Users/yuconghe/anaconda3/lib/python3.5/site-packages/ipykernel/__main__.py:151: FutureWarning: comparisc

Epoch 0: 1009 / 10000
Epoch 1: 1009 / 10000
Epoch 2: 1009 / 10000
Epoch 3: 1009 / 10000
Epoch 4: 1015 / 10000
Epoch 5: 1033 / 10000
Epoch 6: 1070 / 10000
Epoch 7: 1115 / 10000
Epoch 8: 1164 / 10000
Epoch 9: 1195 / 10000
Epoch 10: 1231 / 10000
Epoch 11: 1277 / 10000
Epoch 12: 1326 / 10000
Epoch 13: 1368 / 10000
Epoch 14: 1409 / 10000
Epoch 15: 1455 / 10000
Epoch 16: 1517 / 10000
Epoch 17: 1558 / 10000
Epoch 18: 1616 / 10000
Epoch 19: 1671 / 10000
Epoch 20: 1723 / 10000
Epoch 21: 1779 / 10000
Epoch 22: 1827 / 10000
```

```
Epoch 23: 1883 / 10000
Epoch 24: 1920 / 10000
Epoch 25: 1962 / 10000
Epoch 26: 2003 / 10000
Epoch 27: 2037 / 10000
Epoch 28: 2074 / 10000
Epoch 29: 2124 / 10000
Epoch 30: 2168 / 10000
Epoch 31: 2204 / 10000
Epoch 32: 2230 / 10000
Epoch 33: 2271 / 10000
Epoch 34: 2304 / 10000
Epoch 35: 2321 / 10000
Epoch 36: 2354 / 10000
Epoch 37: 2387 / 10000
Epoch 38: 2411 / 10000
Epoch 39: 2438 / 10000
Epoch 40: 2455 / 10000
Epoch 41: 2478 / 10000
Epoch 42: 2503 / 10000
Epoch 43: 2510 / 10000
Epoch 44: 2534 / 10000
Epoch 45: 2557 / 10000
Epoch 46: 2586 / 10000
Epoch 47: 2606 / 10000
Epoch 48: 2619 / 10000
Epoch 49: 2641 / 10000
Epoch 50: 2664 / 10000
Epoch 51: 2692 / 10000
Epoch 52: 2708 / 10000
Epoch 53: 2729 / 10000
Epoch 54: 2753 / 10000
Epoch 55: 2780 / 10000
Epoch 56: 2810 / 10000
Epoch 57: 2833 / 10000
Epoch 58: 2854 / 10000
Epoch 59: 2881 / 10000
Epoch 60: 2907 / 10000
Epoch 61: 2925 / 10000
Epoch 62: 2948 / 10000
Epoch 63: 2961 / 10000
Epoch 64: 2991 / 10000
Epoch 65: 3010 / 10000
Epoch 66: 3038 / 10000
Epoch 67: 3070 / 10000
Epoch 68: 3090 / 10000
Epoch 69: 3123 / 10000
Epoch 70: 3142 / 10000
Epoch 71: 3166 / 10000
Epoch 72: 3188 / 10000
Epoch 73: 3222 / 10000
Epoch 74: 3247 / 10000
Epoch 75: 3272 / 10000
Epoch 76: 3299 / 10000
```

```
Epoch 77: 3329 / 10000
Epoch 78: 3371 / 10000
Epoch 79: 3418 / 10000
Epoch 80: 3456 / 10000
Epoch 81: 3494 / 10000
Epoch 82: 3523 / 10000
Epoch 83: 3554 / 10000
Epoch 84: 3582 / 10000
Epoch 85: 3612 / 10000
Epoch 86: 3645 / 10000
Epoch 87: 3681 / 10000
Epoch 88: 3713 / 10000
Epoch 89: 3751 / 10000
Epoch 90: 3791 / 10000
Epoch 91: 3841 / 10000
Epoch 92: 3883 / 10000
Epoch 93: 3932 / 10000
Epoch 94: 3980 / 10000
Epoch 95: 4026 / 10000
Epoch 96: 4052 / 10000
Epoch 97: 4084 / 10000
Epoch 98: 4117 / 10000
Epoch 99: 4148 / 10000
Epoch 100: 4173 / 10000
Epoch 101: 4211 / 10000
Epoch 102: 4253 / 10000
Epoch 103: 4295 / 10000
Epoch 104: 4338 / 10000
Epoch 105: 4391 / 10000
Epoch 106: 4452 / 10000
Epoch 107: 4531 / 10000
Epoch 108: 4577 / 10000
Epoch 109: 4652 / 10000
Epoch 110: 4712 / 10000
Epoch 111: 4772 / 10000
Epoch 112: 4833 / 10000
Epoch 113: 4893 / 10000
Epoch 114: 4942 / 10000
Epoch 115: 4993 / 10000
Epoch 116: 5047 / 10000
Epoch 117: 5104 / 10000
Epoch 118: 5167 / 10000
Epoch 119: 5237 / 10000
Epoch 120: 5235 / 10000
Epoch 121: 5117 / 10000
Epoch 122: 5215 / 10000
Epoch 123: 5239 / 10000
Epoch 124: 5332 / 10000
Epoch 125: 5468 / 10000
Epoch 126: 5549 / 10000
Epoch 127: 5681 / 10000
Epoch 128: 5808 / 10000
Epoch 129: 5881 / 10000
Epoch 130: 5988 / 10000
```

```
Epoch 131: 6021 / 10000
Epoch 132: 6028 / 10000
Epoch 133: 6073 / 10000
Epoch 134: 6118 / 10000
Epoch 135: 6150 / 10000
Epoch 136: 6207 / 10000
Epoch 137: 6272 / 10000
Epoch 138: 6278 / 10000
Epoch 139: 6361 / 10000
Epoch 140: 6405 / 10000
Epoch 141: 6441 / 10000
Epoch 142: 6516 / 10000
Epoch 143: 6530 / 10000
Epoch 144: 6598 / 10000
Epoch 145: 6679 / 10000
Epoch 146: 6719 / 10000
Epoch 147: 6745 / 10000
Epoch 148: 6669 / 10000
Epoch 149: 6771 / 10000
Epoch 150: 6824 / 10000
Epoch 151: 6874 / 10000
Epoch 152: 6931 / 10000
Epoch 153: 6950 / 10000
Epoch 154: 6922 / 10000
Epoch 155: 7061 / 10000
Epoch 156: 7059 / 10000
Epoch 157: 7152 / 10000
Epoch 158: 7139 / 10000
Epoch 159: 7230 / 10000
Epoch 160: 7262 / 10000
Epoch 161: 7295 / 10000
Epoch 162: 7330 / 10000
Epoch 163: 7353 / 10000
Epoch 164: 7373 / 10000
Epoch 165: 7400 / 10000
Epoch 166: 7422 / 10000
Epoch 167: 7356 / 10000
Epoch 168: 7301 / 10000
Epoch 169: 7466 / 10000
Epoch 170: 7548 / 10000
Epoch 171: 7528 / 10000
Epoch 172: 6639 / 10000
Epoch 173: 7644 / 10000
Epoch 174: 7300 / 10000
Epoch 175: 7703 / 10000
Epoch 176: 7718 / 10000
Epoch 177: 7774 / 10000
Epoch 178: 7807 / 10000
Epoch 179: 7822 / 10000
Epoch 180: 7867 / 10000
Epoch 181: 7924 / 10000
Epoch 182: 7513 / 10000
Epoch 183: 7925 / 10000
Epoch 184: 7958 / 10000
```

```
Epoch 185: 8034 / 10000
Epoch 186: 8055 / 10000
Epoch 187: 7900 / 10000
Epoch 188: 8082 / 10000
Epoch 189: 8121 / 10000
Epoch 190: 8136 / 10000
Epoch 191: 8188 / 10000
Epoch 192: 8214 / 10000
Epoch 193: 8199 / 10000
Epoch 194: 8247 / 10000
Epoch 195: 8265 / 10000
Epoch 196: 8253 / 10000
Epoch 197: 8353 / 10000
Epoch 198: 8272 / 10000
Epoch 199: 8388 / 10000
Epoch 200: 8413 / 10000
Epoch 201: 8456 / 10000
Epoch 202: 8477 / 10000
Epoch 203: 8504 / 10000
Epoch 204: 8518 / 10000
Epoch 205: 8537 / 10000
Epoch 206: 8565 / 10000
Epoch 207: 8572 / 10000
Epoch 208: 8528 / 10000
Epoch 209: 8578 / 10000
Epoch 210: 8584 / 10000
Epoch 211: 8646 / 10000
Epoch 212: 8608 / 10000
Epoch 213: 8703 / 10000
Epoch 214: 8726 / 10000
Epoch 215: 8730 / 10000
Epoch 216: 8756 / 10000
Epoch 217: 8756 / 10000
Epoch 218: 8771 / 10000
Epoch 219: 8783 / 10000
Epoch 220: 8803 / 10000
Epoch 221: 8808 / 10000
Epoch 222: 8827 / 10000
Epoch 223: 8832 / 10000
Epoch 224: 8856 / 10000
Epoch 225: 8877 / 10000
Epoch 226: 8886 / 10000
Epoch 227: 8910 / 10000
Epoch 228: 8924 / 10000
Epoch 229: 8935 / 10000
Epoch 230: 8944 / 10000
Epoch 231: 8954 / 10000
Epoch 232: 8969 / 10000
Epoch 233: 8977 / 10000
Epoch 234: 8989 / 10000
Epoch 235: 8996 / 10000
Epoch 236: 9004 / 10000
Epoch 237: 9010 / 10000
Epoch 238: 9018 / 10000
```

```
Epoch 239: 9028 / 10000
Epoch 240: 9041 / 10000
Epoch 241: 9049 / 10000
Epoch 242: 9028 / 10000
Epoch 243: 9045 / 10000
Epoch 244: 9043 / 10000
Epoch 245: 9063 / 10000
Epoch 246: 9074 / 10000
Epoch 247: 9072 / 10000
Epoch 248: 9097 / 10000
Epoch 249: 9109 / 10000
Epoch 250: 9111 / 10000
Epoch 251: 9119 / 10000
Epoch 252: 9126 / 10000
Epoch 253: 9137 / 10000
Epoch 254: 9144 / 10000
Epoch 255: 9160 / 10000
Epoch 256: 9164 / 10000
Epoch 257: 9165 / 10000
Epoch 258: 9169 / 10000
Epoch 259: 9181 / 10000
Epoch 260: 9193 / 10000
Epoch 261: 9201 / 10000
Epoch 262: 9206 / 10000
Epoch 263: 9208 / 10000
Epoch 264: 9215 / 10000
Epoch 265: 9229 / 10000
Epoch 266: 9235 / 10000
Epoch 267: 9239 / 10000
Epoch 268: 9235 / 10000
Epoch 269: 9252 / 10000
Epoch 270: 9253 / 10000
Epoch 271: 9255 / 10000
Epoch 272: 9263 / 10000
Epoch 273: 9271 / 10000
Epoch 274: 9276 / 10000
Epoch 275: 9285 / 10000
Epoch 276: 9287 / 10000
Epoch 277: 9291 / 10000
Epoch 278: 9296 / 10000
Epoch 279: 9297 / 10000
Epoch 280: 9302 / 10000
Epoch 281: 9317 / 10000
Epoch 282: 9319 / 10000
Epoch 283: 9323 / 10000
Epoch 284: 9327 / 10000
Epoch 285: 9334 / 10000
Epoch 286: 9338 / 10000
Epoch 287: 9342 / 10000
Epoch 288: 9346 / 10000
Epoch 289: 9352 / 10000
Epoch 290: 9354 / 10000
Epoch 291: 9359 / 10000
Epoch 292: 9366 / 10000
```

```
Epoch 293: 9369 / 10000
Epoch 294: 9367 / 10000
Epoch 295: 9372 / 10000
Epoch 296: 9374 / 10000
Epoch 297: 9384 / 10000
Epoch 298: 9387 / 10000
Epoch 299: 9395 / 10000
Epoch 0: 1012 / 10000
Epoch 1: 1012 / 10000
Epoch 2: 1012 / 10000
Epoch 3: 1012 / 10000
Epoch 4: 1012 / 10000
Epoch 5: 1012 / 10000
Epoch 6: 1012 / 10000
Epoch 7: 1012 / 10000
Epoch 8: 1012 / 10000
Epoch 9: 1012 / 10000
Epoch 10: 1012 / 10000
Epoch 11: 1012 / 10000
Epoch 12: 1012 / 10000
Epoch 13: 1012 / 10000
Epoch 14: 1012 / 10000
Epoch 15: 1012 / 10000
Epoch 16: 1012 / 10000
Epoch 17: 1012 / 10000
Epoch 18: 1012 / 10000
Epoch 19: 1012 / 10000
Epoch 20: 1013 / 10000
Epoch 21: 1013 / 10000
Epoch 22: 1013 / 10000
Epoch 23: 1013 / 10000
Epoch 24: 1013 / 10000
Epoch 25: 1013 / 10000
Epoch 26: 1013 / 10000
Epoch 27: 1012 / 10000
Epoch 28: 1012 / 10000
Epoch 29: 1012 / 10000
Epoch 30: 1012 / 10000
Epoch 31: 1012 / 10000
Epoch 32: 1012 / 10000
Epoch 33: 1012 / 10000
Epoch 34: 1012 / 10000
Epoch 35: 1012 / 10000
Epoch 36: 1012 / 10000
Epoch 37: 1012 / 10000
Epoch 38: 1012 / 10000
Epoch 39: 1012 / 10000
Epoch 40: 1012 / 10000
Epoch 41: 1012 / 10000
Epoch 42: 1012 / 10000
Epoch 43: 1012 / 10000
Epoch 44: 1012 / 10000
Epoch 45: 1012 / 10000
Epoch 46: 1012 / 10000
```

```
Epoch 47: 1012 / 10000
Epoch 48: 1012 / 10000
Epoch 49: 1012 / 10000
Epoch 50: 1012 / 10000
Epoch 51: 1012 / 10000
Epoch 52: 1012 / 10000
Epoch 53: 1012 / 10000
Epoch 54: 1012 / 10000
Epoch 55: 1012 / 10000
Epoch 56: 1012 / 10000
Epoch 57: 1012 / 10000
Epoch 58: 1012 / 10000
Epoch 59: 1012 / 10000
Epoch 60: 1012 / 10000
Epoch 61: 1012 / 10000
Epoch 62: 1012 / 10000
Epoch 63: 1012 / 10000
Epoch 64: 1012 / 10000
Epoch 65: 1012 / 10000
Epoch 66: 1012 / 10000
Epoch 67: 1012 / 10000
Epoch 68: 1012 / 10000
Epoch 69: 1012 / 10000
Epoch 70: 1012 / 10000
Epoch 71: 1012 / 10000
Epoch 72: 1012 / 10000
Epoch 73: 1013 / 10000
Epoch 74: 1012 / 10000
Epoch 75: 1012 / 10000
Epoch 76: 1012 / 10000
Epoch 77: 1012 / 10000
Epoch 78: 1012 / 10000
Epoch 79: 1012 / 10000
Epoch 80: 1012 / 10000
Epoch 81: 1012 / 10000
Epoch 82: 1012 / 10000
Epoch 83: 1012 / 10000
Epoch 84: 1012 / 10000
Epoch 85: 1012 / 10000
Epoch 86: 1012 / 10000
Epoch 87: 1012 / 10000
Epoch 88: 1012 / 10000
```

```
---------------------------------------------------------------------------

KeyboardInterrupt                         Traceback (most recent call last)

<ipython-input-46-883043768f70> in <module>()
     13     CNN = ComplexNetwork()
     14     file_name = "test2_CNN_" + str(i)
---> 15     rate = CNN.stochastic_gradient_descent(t_mat, t_labels, 300, 100, rate, 0.01/len(t_labels
     16     CNN.save(file_name)
     17     val += 100
```

```
        <ipython-input-8-a739399ef718> in stochastic_gradient_descent(self, X, y, number_of_epochs, mini
        147             momentum = self.calculate_momentum(epoch+t)
        148             for batch in batches:
    --> 149                 self.train_mini_batch( batch[0], batch[1], rate, L2, momentum)
        150
        151             if test_X != None and test_y != None:


        <ipython-input-8-a739399ef718> in train_mini_batch(self, X, y, rate, L2, momentum)
        127         self.bias4 -= (rate)*np.mean(self.delta4, axis=0)
        128         self.W1 -= (rate/n)*self.dJdW1*(1-momentum) - rate*L2*self.W1*momentum
    --> 129         self.W2 -= (rate/n)*self.dJdW2*(1-momentum) - rate*L2*self.W2*momentum
        130         self.W3 -= (rate/n)*self.dJdW3*(1-momentum) - rate*L2*self.W3*momentum
        131


        KeyboardInterrupt:
```

In [47]:
```
CNN_3 = ComplexNetwork()
CNN_3.load("test2_CNN_3")
rate = 5.48
CNN_3.stochastic_gradient_descent(t_mat, t_labels, 300, 100, rate, 0.01/len(t_labels), \
                                        test_X = v_mat, test_y = v_labels, t=val)
```

```
/Users/yuconghe/anaconda3/lib/python3.5/site-packages/ipykernel/__main__.py:151: FutureWarning: comparis

Epoch 0: 9428 / 10000
Epoch 1: 9437 / 10000
Epoch 2: 9441 / 10000
Epoch 3: 9444 / 10000
Epoch 4: 9447 / 10000
Epoch 5: 9450 / 10000
Epoch 6: 9451 / 10000
Epoch 7: 9453 / 10000
Epoch 8: 9458 / 10000
Epoch 9: 9466 / 10000
Epoch 10: 9469 / 10000
Epoch 11: 9471 / 10000
Epoch 12: 9472 / 10000
Epoch 13: 9472 / 10000
Epoch 14: 9475 / 10000
Epoch 15: 9480 / 10000
Epoch 16: 9480 / 10000
Epoch 17: 9483 / 10000
Epoch 18: 9485 / 10000
Epoch 19: 9487 / 10000
Epoch 20: 9491 / 10000
Epoch 21: 9493 / 10000
Epoch 22: 9494 / 10000
Epoch 23: 9497 / 10000
Epoch 24: 9498 / 10000
Epoch 25: 9500 / 10000
```

```
Epoch 26: 9500 / 10000
Epoch 27: 9499 / 10000
Epoch 28: 9501 / 10000
Epoch 29: 9503 / 10000
Epoch 30: 9504 / 10000
Epoch 31: 9505 / 10000
Epoch 32: 9509 / 10000
Epoch 33: 9511 / 10000
Epoch 34: 9514 / 10000
Epoch 35: 9518 / 10000
Epoch 36: 9520 / 10000
Epoch 37: 9521 / 10000
Epoch 38: 9523 / 10000
Epoch 39: 9524 / 10000
Epoch 40: 9528 / 10000
Epoch 41: 9527 / 10000
Epoch 42: 9530 / 10000
Epoch 43: 9530 / 10000
Epoch 44: 9533 / 10000
Epoch 45: 9533 / 10000
Epoch 46: 9534 / 10000
Epoch 47: 9534 / 10000
Epoch 48: 9537 / 10000
Epoch 49: 9537 / 10000
Epoch 50: 9537 / 10000
Epoch 51: 9541 / 10000
Epoch 52: 9543 / 10000
Epoch 53: 9545 / 10000
Epoch 54: 9547 / 10000
Epoch 55: 9550 / 10000
Epoch 56: 9553 / 10000
Epoch 57: 9553 / 10000
Epoch 58: 9555 / 10000
Epoch 59: 9556 / 10000
Epoch 60: 9557 / 10000
Epoch 61: 9557 / 10000
Epoch 62: 9560 / 10000
Epoch 63: 9561 / 10000
Epoch 64: 9561 / 10000
Epoch 65: 9563 / 10000
Epoch 66: 9564 / 10000
Epoch 67: 9565 / 10000
Epoch 68: 9565 / 10000
Epoch 69: 9566 / 10000
Epoch 70: 9569 / 10000
Epoch 71: 9570 / 10000
Epoch 72: 9572 / 10000
Epoch 73: 9573 / 10000
Epoch 74: 9574 / 10000
Epoch 75: 9573 / 10000
Epoch 76: 9574 / 10000
Epoch 77: 9574 / 10000
Epoch 78: 9574 / 10000
Epoch 79: 9574 / 10000
```

```
Epoch 80: 9574 / 10000
Epoch 81: 9575 / 10000
Epoch 82: 9575 / 10000
Epoch 83: 9574 / 10000
Epoch 84: 9575 / 10000
Epoch 85: 9576 / 10000
Epoch 86: 9577 / 10000
Epoch 87: 9576 / 10000
Epoch 88: 9577 / 10000
Epoch 89: 9578 / 10000
Epoch 90: 9580 / 10000
Epoch 91: 9580 / 10000
Epoch 92: 9581 / 10000
Epoch 93: 9580 / 10000
Epoch 94: 9584 / 10000
Epoch 95: 9586 / 10000
Epoch 96: 9585 / 10000
Epoch 97: 9586 / 10000
Epoch 98: 9585 / 10000
Epoch 99: 9587 / 10000
Epoch 100: 9586 / 10000
Epoch 101: 9587 / 10000
Epoch 102: 9587 / 10000
Epoch 103: 9585 / 10000
Epoch 104: 9586 / 10000
Epoch 105: 9587 / 10000
Epoch 106: 9587 / 10000
Epoch 107: 9588 / 10000
Epoch 108: 9589 / 10000
Epoch 109: 9590 / 10000
Epoch 110: 9592 / 10000
Epoch 111: 9592 / 10000
Epoch 112: 9592 / 10000
Epoch 113: 9592 / 10000
Epoch 114: 9593 / 10000
Epoch 115: 9593 / 10000
Epoch 116: 9592 / 10000
Epoch 117: 9592 / 10000
Epoch 118: 9592 / 10000
Epoch 119: 9592 / 10000
Epoch 120: 9594 / 10000
Epoch 121: 9595 / 10000
Epoch 122: 9595 / 10000
Epoch 123: 9595 / 10000
Epoch 124: 9594 / 10000
Epoch 125: 9594 / 10000
Epoch 126: 9594 / 10000
Epoch 127: 9597 / 10000
Epoch 128: 9598 / 10000
Epoch 129: 9597 / 10000
Epoch 130: 9605 / 10000
Epoch 131: 9600 / 10000
Epoch 132: 9605 / 10000
Epoch 133: 9605 / 10000
```

```
Epoch 134: 9606 / 10000
Epoch 135: 9606 / 10000
Epoch 136: 9606 / 10000
Epoch 137: 9607 / 10000
Epoch 138: 9607 / 10000
Epoch 139: 9606 / 10000
Epoch 140: 9607 / 10000
Epoch 141: 9607 / 10000
Epoch 142: 9608 / 10000
Epoch 143: 9608 / 10000
Epoch 144: 9608 / 10000
Epoch 145: 9610 / 10000
Epoch 146: 9608 / 10000
Epoch 147: 9609 / 10000
Epoch 148: 9609 / 10000
Epoch 149: 9608 / 10000
Epoch 150: 9607 / 10000
Epoch 151: 9605 / 10000
Epoch 152: 9604 / 10000
Epoch 153: 9604 / 10000
Epoch 154: 9604 / 10000
Epoch 155: 9606 / 10000
Epoch 156: 9606 / 10000
Epoch 157: 9606 / 10000
Epoch 158: 9607 / 10000
Epoch 159: 9606 / 10000
Epoch 160: 9608 / 10000
Epoch 161: 9608 / 10000
Epoch 162: 9610 / 10000
Epoch 163: 9609 / 10000
Epoch 164: 9606 / 10000
Epoch 165: 9607 / 10000
Epoch 166: 9606 / 10000
Epoch 167: 9606 / 10000
Epoch 168: 9607 / 10000
Epoch 169: 9607 / 10000
Epoch 170: 9607 / 10000
Epoch 171: 9609 / 10000
Epoch 172: 9610 / 10000
Epoch 173: 9610 / 10000
Epoch 174: 9610 / 10000
Epoch 175: 9609 / 10000
Epoch 176: 9609 / 10000
Epoch 177: 9609 / 10000
Epoch 178: 9610 / 10000
Epoch 179: 9609 / 10000
Epoch 180: 9609 / 10000
Epoch 181: 9609 / 10000
Epoch 182: 9610 / 10000
Epoch 183: 9610 / 10000
Epoch 184: 9610 / 10000
Epoch 185: 9612 / 10000
Epoch 186: 9616 / 10000
Epoch 187: 9611 / 10000
```

```
Epoch 188: 9610 / 10000
Epoch 189: 9609 / 10000
Epoch 190: 9608 / 10000
Epoch 191: 9608 / 10000
Epoch 192: 9609 / 10000
Epoch 193: 9610 / 10000
Epoch 194: 9610 / 10000
Epoch 195: 9611 / 10000
Epoch 196: 9608 / 10000
Epoch 197: 9608 / 10000
Epoch 198: 9609 / 10000
Epoch 199: 9605 / 10000
Epoch 200: 9604 / 10000
Epoch 201: 9609 / 10000
Epoch 202: 9609 / 10000
Epoch 203: 9609 / 10000
Epoch 204: 9610 / 10000
Epoch 205: 9610 / 10000
Epoch 206: 9610 / 10000
Epoch 207: 9611 / 10000
Epoch 208: 9610 / 10000
Epoch 209: 9610 / 10000
Epoch 210: 9610 / 10000
Epoch 211: 9610 / 10000
Epoch 212: 9610 / 10000
Epoch 213: 9610 / 10000
Epoch 214: 9610 / 10000
Epoch 215: 9610 / 10000
Epoch 216: 9611 / 10000
Epoch 217: 9611 / 10000
Epoch 218: 9611 / 10000
Epoch 219: 9611 / 10000
Epoch 220: 9611 / 10000
Epoch 221: 9611 / 10000
Epoch 222: 9611 / 10000
Epoch 223: 9612 / 10000
Epoch 224: 9613 / 10000
Epoch 225: 9612 / 10000
Epoch 226: 9612 / 10000
Epoch 227: 9612 / 10000
Epoch 228: 9611 / 10000
Epoch 229: 9610 / 10000
Epoch 230: 9610 / 10000
Epoch 231: 9611 / 10000
Epoch 232: 9613 / 10000
Epoch 233: 9613 / 10000
Epoch 234: 9613 / 10000
Epoch 235: 9613 / 10000
Epoch 236: 9613 / 10000
Epoch 237: 9613 / 10000
Epoch 238: 9614 / 10000
Epoch 239: 9614 / 10000
Epoch 240: 9614 / 10000
Epoch 241: 9614 / 10000
```

```
Epoch 242: 9614 / 10000
Epoch 243: 9614 / 10000
Epoch 244: 9614 / 10000
Epoch 245: 9614 / 10000
Epoch 246: 9614 / 10000
Epoch 247: 9614 / 10000
Epoch 248: 9614 / 10000
Epoch 249: 9615 / 10000
Epoch 250: 9615 / 10000
Epoch 251: 9614 / 10000
Epoch 252: 9615 / 10000
Epoch 253: 9614 / 10000
Epoch 254: 9614 / 10000
Epoch 255: 9614 / 10000
Epoch 256: 9615 / 10000
Epoch 257: 9613 / 10000
Epoch 258: 9613 / 10000
Epoch 259: 9613 / 10000
Epoch 260: 9613 / 10000
Epoch 261: 9613 / 10000
Epoch 262: 9613 / 10000
Epoch 263: 9613 / 10000
Epoch 264: 9613 / 10000
Epoch 265: 9613 / 10000
Epoch 266: 9613 / 10000
Epoch 267: 9613 / 10000
Epoch 268: 9613 / 10000
Epoch 269: 9614 / 10000
Epoch 270: 9614 / 10000
Epoch 271: 9614 / 10000
Epoch 272: 9614 / 10000
Epoch 273: 9614 / 10000
Epoch 274: 9614 / 10000
Epoch 275: 9614 / 10000
Epoch 276: 9614 / 10000
Epoch 277: 9614 / 10000
Epoch 278: 9614 / 10000
Epoch 279: 9614 / 10000
Epoch 280: 9613 / 10000
Epoch 281: 9612 / 10000
Epoch 282: 9613 / 10000
Epoch 283: 9614 / 10000
Epoch 284: 9614 / 10000
Epoch 285: 9613 / 10000
Epoch 286: 9613 / 10000
Epoch 287: 9613 / 10000
Epoch 288: 9613 / 10000
Epoch 289: 9613 / 10000
Epoch 290: 9613 / 10000
Epoch 291: 9613 / 10000
Epoch 292: 9613 / 10000
Epoch 293: 9614 / 10000
Epoch 294: 9616 / 10000
Epoch 295: 9614 / 10000
```

```
Epoch 296: 9612 / 10000
Epoch 297: 9612 / 10000
Epoch 298: 9614 / 10000
Epoch 299: 9616 / 10000

Out[47]: 3.0056814062166457

In [33]: """ Bootstrap: trial"""
         import csv
         NN1 = Network()
         NN1.load("FinalCell96Ver1")
         prediction1 = np.argmax(NN1.forward(test_mat), axis=1)
         NN2 = ComplexNetwork()
         NN2.load("FinalCNNVer2")
         prediction2 = np.argmax(NN2.forward(test_mat)[0], axis=1)
         NN3 = ComplexNetwork()
         NN3.load("test2_CNN_0")
         prediction3 = np.argmax(NN3.forward(test_mat)[0], axis=1)
         NN4 = ComplexNetwork()
         NN4.load("test2_CNN_1")
         prediction4 = np.argmax(NN4.forward(test_mat)[0], axis=1)
         NN5 = ComplexNetwork()
         NN5.load("Final9777Ver1")
         prediction5 = np.argmax(NN5.forward(test_mat)[0], axis=1)
         NN6 = ComplexNetwork()
         NN6.load("test2_CNN_5")
         prediction6 = np.argmax(NN6.forward(test_mat)[0], axis=1)

In [34]: result = np.vstack((prediction1, prediction2, prediction3, prediction4, prediction5, predictio

In [28]: NN4 = ComplexNetwork()
         NN4.load("test2_CNN_4")
         yHat = NN4.forward(v_mat)[0]
         print(yHat)
         predicted = np.argmax(yHat, axis = 1)
         print(predicted)
         expected = np.argmax(v_labels, axis = 1)
         s = metrics.accuracy_score(expected, predicted)
         m = metrics.confusion_matrix(expected, predicted)
         print(s)
         print(m)

[[  3.04361031e-09   1.20288138e-07   7.02874034e-09 ...,   2.02279261e-04
    7.73807790e-03   3.50079807e-02]
 [  6.41950118e-13   8.69858078e-01   3.08454351e-07 ...,   1.43845660e-04
    3.89575697e-02   1.58292313e-06]
 [  1.57498381e-08   2.81276031e-07   2.85093413e-08 ...,   2.60393012e-04
    3.36546734e-09   9.99999369e-01]
 ...,
 [  9.98903269e-01   1.68685978e-08   2.78388982e-06 ...,   2.52560391e-04
    3.79162055e-05   1.59473576e-07]
 [  1.07690351e-09   1.02735978e-08   6.71610499e-06 ...,   3.31469554e-06
    3.68319325e-05   1.02405847e-03]
 [  5.12553291e-09   2.25790319e-05   7.99097454e-07 ...,   2.54440879e-06
    1.50230105e-01   5.05281522e-03]]
```

```
[4 1 9 ..., 0 4 4]
0.87
[[ 975    0    1    0    0   12   14    0    6    2]
 [   0 1035    5    4    1    2    2    3   22    2]
 [   0    0  898    9    1    6   17    6   37    2]
 [   0    1    0  857    0   93    5    4   69   27]
 [   1    0    0    0  639    1   13    1    8  296]
 [   0    0    0    0    0  819   17    0   22   27]
 [   1    0    0    0    0    0  960    0   11    0]
 [   0    1    2    0    0    0    0  585    7  460]
 [   0    1    0    0    0    1    1    0  916   69]
 [   1    0    0    1    2    2    0    1    0 1016]]
```

In [35]:
```
from scipy.stats import mode
prediction = []
print(result[1])
```

```
[6 6 6 6 6 6]
```

In [36]:
```
for i in range(10000):
    prediction.append(mode(result[i])[0][0])
```

In [37]:
```
with open('predicted_digits.csv', 'w') as csvfile:
    writer = csv.writer(csvfile, lineterminator = '\n')
    for i, content in enumerate(prediction):
        writer.writerow([i+1, content])
```

In [ ]: