CS289–Spring 2017 — Homework 6 Solutions

Shuhui Huang, SID 3032129712

1. Problem 1: Derivations

We have two weight matrices.
V is a 785*200$(n_{in}+1)*n_{hid}$ matrix that connects the input units to the hidden units. The last column of V is made up of bias terms, which are multiplied by 1 instead of an input value.
W is a 201*26$(n_{hid}+1)*n_{out}$ matrix that connects the hidden units to the output units. The last column is bias terms.

Let X be the input layer, h be the hidden layer, Z be the output layer, y is the ground truth label So we have :
$h_j = tanh(\sum_i V_{ij}X_i)$, $Z_k = sigmoid(\sum_j W_{jk}h_j)$

As we use the cross-entropy loss function:
$L(z,y) = -\sum_{j=1}^{26} y_j InZ_j + (1-y_j)In(1-Z_j)$
So $\frac{\partial L}{\partial Z_k} = -(\frac{y_k}{Z_k} - \frac{1-y_k}{1-Z_k}) = \frac{Z_k - y_k}{Z_k(1-Z_k)}$
$\frac{\partial L}{\partial W_{jk}} = \frac{\partial L}{\partial Z_k}\frac{\partial Z_k}{\partial W_{jk}} = \frac{Z_k-y_k}{Z_k(1-Z_k)}Z_k(1-Z_k)h_j = (Z_k - y_k)h_j$
$\frac{\partial L}{\partial V_{ij}} = (\sum_k \frac{\partial L}{\partial Z_k}\frac{\partial Z_k}{\partial h_j})\frac{\partial h_j}{\partial V_{ij}} = (\sum_k (Z_k - y_k)W_{jk})(1-h_j^2)X_i$

$\therefore$ we have:
$\frac{\partial L}{\partial W} = h(Z-y)^T$
$\frac{\partial L}{\partial V} = X[W(Z-y)(1-h^2)]^T$

Which is the partial derivatives of L with respect to V and W we want to get.

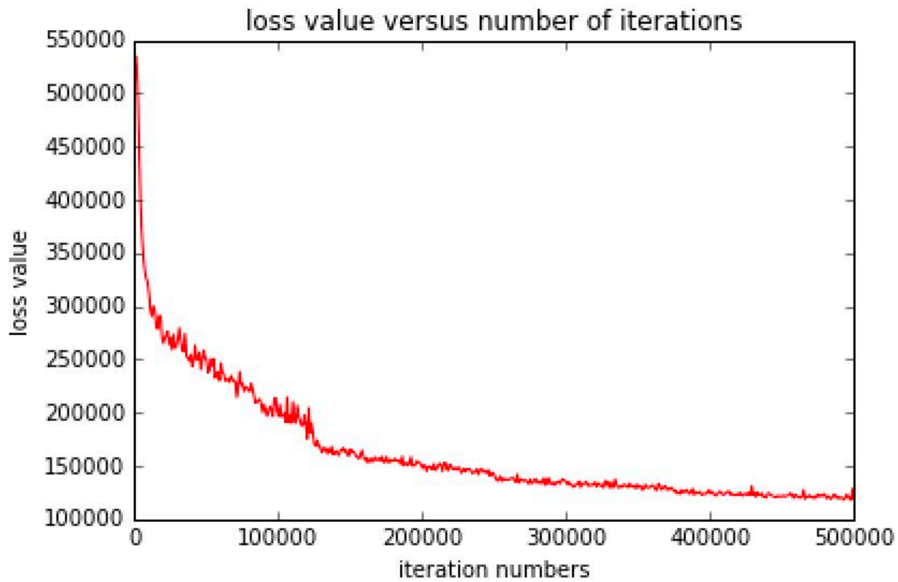
2.Problem 2: Implementation

(a) Any hyperparameters that you tuned.
I first used learning rate=0.001, then tuned it to 0.05,max iteration=500000, loss function is cross entropy error,and hidden layer have 200 units.

(b) my training accuracy.
0.8823

(c) my validation accuracy:
0.8656

(d) A plot of the loss value versus the number of iterations. You may sample (i.e., compute the loss every 1000 iterations).

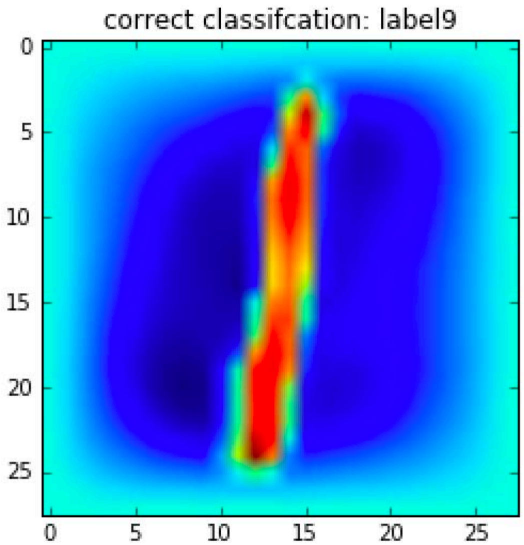


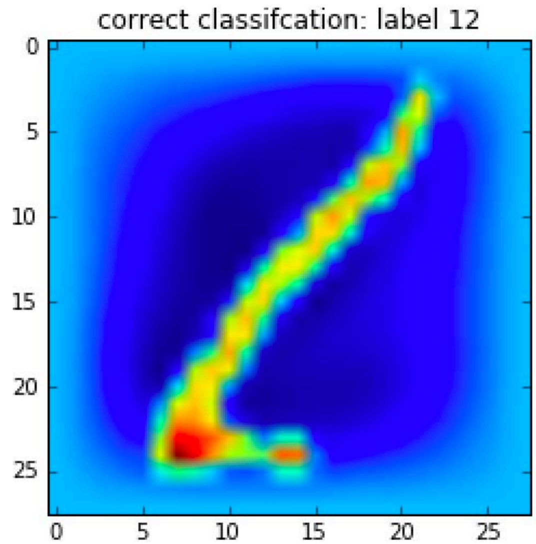
(e) my Kaggle score: 0.86115
display name:Shuhui Huang

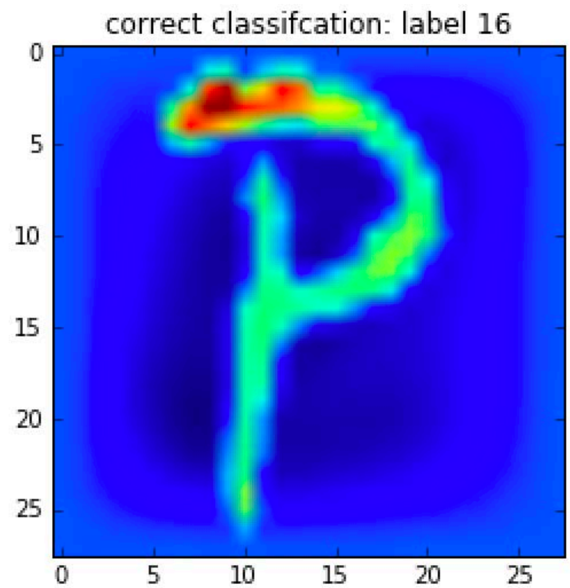
(f) My code is attached as an appendix at the end.

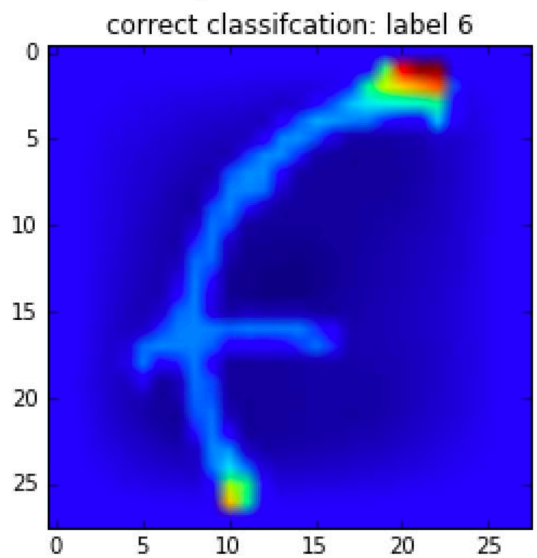
3.Problem 3: Visualization

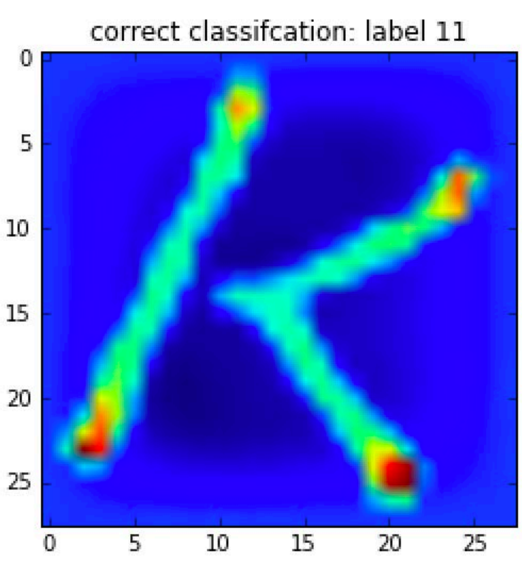
5 digits (and their labels) from the validation set that your neural network correctly classifies:



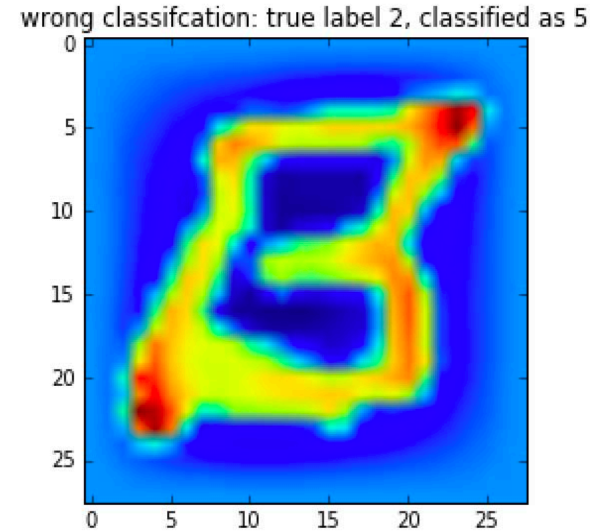


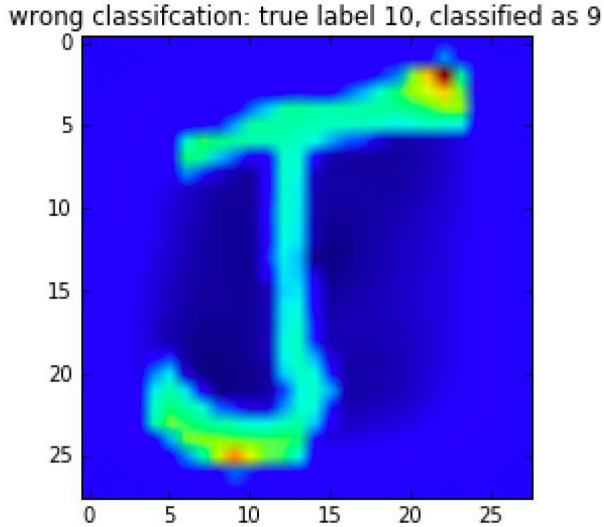


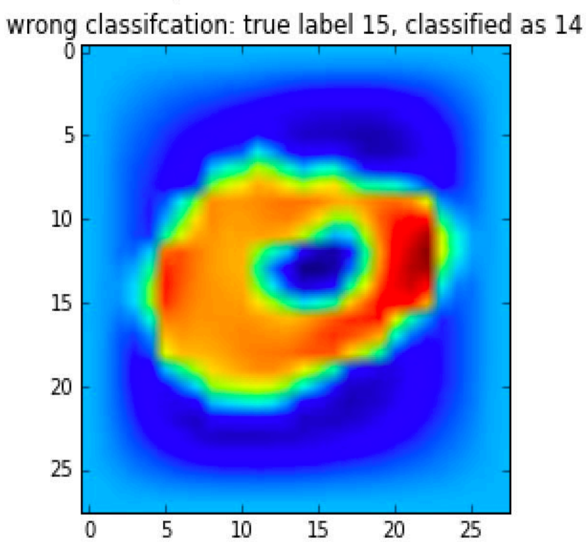


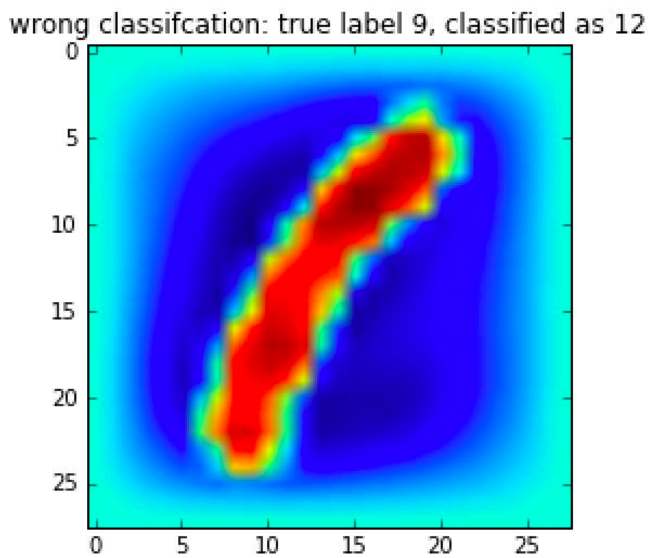


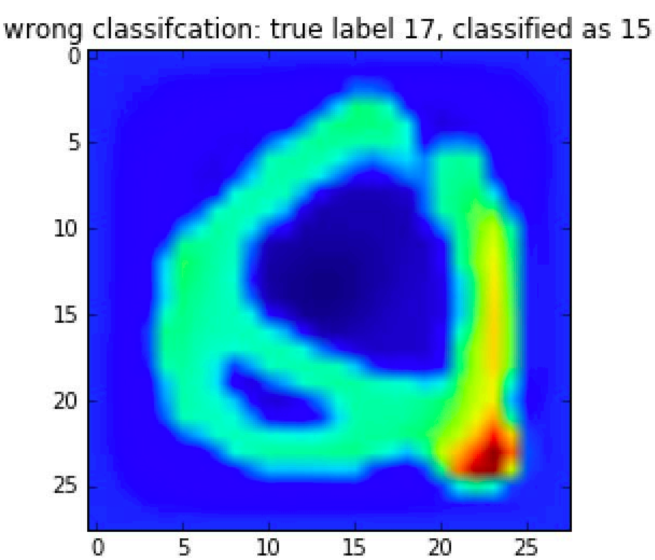
5 digits (and their labels) from the validation set that my neural network does not correctly classify:











4.Problem 4: Bells and Whistles

I changed the number of hidden layer units from 200 to 500, and also changed the initialization of weights from a Gaussian distribution with mean 0 and variance $\sigma^2 \propto 1/\eta$, where η is the fan-in of the neuron the weight is an input to.

In addition, I also use different learning rates for different layers, and have those rates decay over time. learning rate=learning rate $*\frac{k}{k+1}$, where k is $\frac{iteration\ numbers}{sample\ numbers}$.

My code has been attached in the appendix.

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Tue Mar 28 10:20:16 2017

@author: huangshuhui
"""
#######problem2######
import numpy as np
import scipy.io
import random
from sklearn.preprocessing import StandardScaler
from scipy.special import expit
import matplotlib.pyplot as plt
import math

#load the data#
data_raw=scipy.io.loadmat('/Users/huangshuhui/Google
Drive/study/cs289/hw2017/hw6/hw6_data_dist/letters_data.mat')
test_x=data_raw['test_x']
train_x=data_raw['train_x']
train_label=data_raw['train_y']

#normalize  and shuffle the data#
scaler = StandardScaler()
scaler.fit(train_x)
train_x = scaler.transform(train_x)
test_x = scaler.transform(test_x)
x_y = list(zip(train_x, train_label))
random.shuffle(x_y)
train_x = np.array([e[0] for e in x_y])
train_label = np.ravel([e[1] for e in x_y])

train_y = np.zeros(shape=(train_x.shape[0],27))
for i in range(len(train_y)):
   train_y[i][train_label[i]]=1

#the original neural network#
class NN:
   def __init__(self,learning_rate,max_iteration,cost_function="cross-
entropy",regularizor=0,n_hid=200):
      self.learning_rate = learning_rate
      self.max_iteration = max_iteration
      self.cost_function = cost_function
```

```
    self.regularizor = regularizor
    self.n_hid = n_hid

def train(self,train_x,train_y):
    #### initialize all weights randomly ####
    n_in = train_x.shape[1]
    n_hid = self.n_hid
    n_out = train_y.shape[1]

    self.V = 0.01*np.random.randn(n_in+1,n_hid)
    self.W = 0.01*np.random.randn(n_hid+1,n_out)
    self.iteration_nums = []
    self.error_ls = []
    self.accuracy_ls = []

    #### iteration ####
    t=0
    while(t<self.max_iteration):
    # while(t<self.max_iteration):
        t+=1
        ### pick a random point and add bias term ###
        rand_index = np.random.randint(0,len(train_y))
        x = np.append(train_x[rand_index],1)
        y = train_y[rand_index]
        x = np.array([x]).T
        y = np.array([y]).T
        ### forward pass ###
        z = np.tanh((self.V.T).dot(x))
        z = np.vstack((z,[1]))
        h = expit((self.W.T).dot(z))
        ### backward pass ###
        if self.cost_function == "cross-entropy":
            delta_V,delta_W = self.Cross_entropy_delta_W(z,h,y,x,self.W)
        ### gradient descent update ###
        self.V = (1-self.regularizor)*self.V - self.learning_rate*delta_V
        self.W = (1-self.regularizor)*self.W - self.learning_rate*delta_W


        ### total training error and classification accuracy vs. iteration ###
        if t%1000==0:
            self.iteration_nums.append(t)
            if self.cost_function=="cross-entropy":
                error = self.cross_entropy_error(train_x,train_y,self.V,self.W)
                predict_labels = self.predict(train_x)
```

```
            true_labels = train_y.argmax(axis=1)
            accuracy = sum(predict_labels==true_labels)/len(true_labels)
            print(t," : ",error,accuracy)
            self.error_ls.append(error)
            self.accuracy_ls.append(accuracy)

    def Cross_entropy_delta_W(self,z,h,y,x,W):
        delta_W = z.dot((h-y).T)
        delta_V = x.dot(((W[0:-1].dot(h-y))*(1-(z[0:-1])**2)).T)
        return delta_V,delta_W

    def cross_entropy_error(self,train_x,train_y,V,W):
        ### calculate total cross entropy error ###
        X = np.column_stack((train_x, np.ones(train_x.shape[0])))
        Z = np.tanh(X.dot(V))
        Z = np.column_stack((Z, np.ones(Z.shape[0])))
        H = 1/(np.exp(-Z.dot(W))+1)
        Cross_entropy_error = np.sum(-(train_y*np.log(H)+(1-train_y)*np.log(1-H)))
        return Cross_entropy_error

    def predict(self,test_x):
        X = np.column_stack((test_x, np.ones(test_x.shape[0])))
        Z = np.tanh(X.dot(self.V))
        Z = np.column_stack((Z, np.ones(Z.shape[0])))
        H = Z.dot(self.W)
        predicted_labels = H.argmax(axis=1)
        return predicted_labels

#test the trainging and validation accuracy#
sub_train_x = train_x[0:100000]
sub_train_y = train_y[0:100000]
sub_validate_x = train_x[100000:120000]
sub_validate_y = train_y[100000:120000]
test_NN = NN(learning_rate=0.01,max_iteration=600000,cost_function="cross-
entropy",regularizor=0,n_hid=500)
test_NN.train(sub_train_x,sub_train_y)
predict_y = test_NN.predict(sub_validate_x)
true_y = sub_validate_y.argmax(axis=1)
validate_accuracy = sum(predict_y==true_y)/len(true_y)
validate_accuracy

#plot iteration and loss#
cross_NN = NN(learning_rate=0.001,max_iteration=500000,cost_function="cross-
entropy",regularizor=0,n_hid=200)
```

```
cross_NN.train(train_x,train_y)
plt.plot(cross_NN.iteration_nums, cross_NN.error_ls, 'r-')
plt.xlabel('iteration numbers')
plt.ylabel('loss value')
plt.title('loss value versus number of iterations')
```

```
#######problem3######

##Visualization
correct=0
wrong=0
i=0
label1,label2,label3=[],[],[]
validate_correct=[]
validate_wrong=[]
while correct<5:
   if predict_y[i]==true_y[i] and true_y[i] not in label1:
      correct+=1
      label1.append(true_y[i])
      validate_correct.append(sub_validate_x[i])
   i+=1

i=0
while wrong<5:
   if predict_y[i]!=true_y[i]and true_y[i] not in label2:
      wrong+=1
      label2.append(true_y[i])
      label3.append(predict_y[i])
      validate_wrong.append(sub_validate_x[i])
   i+=1

a=validate_wrong[4]
plt.imshow(a.reshape(28, 28))
plt.title('wrong classifcation: true label 17, classified as 15' )
```

```
#######problem4######

#neural network(Bells and Whistles)#
class NN2:
    def __init__(self,learning_rate,max_iteration,cost_function="cross-
entropy",regularizor=0,n_hid=200):
        self.learning_rate = learning_rate
        self.max_iteration = max_iteration
        self.cost_function = cost_function
        self.regularizor = regularizor
        self.n_hid = n_hid

    def train(self,train_x,train_y):
        #### initialize all weights randomly ####
        n_in = train_x.shape[1]
        n_hid = self.n_hid
        n_out = train_y.shape[1]

        self.V =
np.vstack((math.sqrt(2.0/(n_in+n_hid))*np.random.randn(n_in,n_hid),0.01*np.ones(n_hid)))
        self.W =
np.vstack((math.sqrt(2.0/(n_hid+n_out))*np.random.randn(n_hid,n_out),0.01*np.ones(n_out))
)

        self.iteration_nums = []
        self.error_ls = []
        self.accuracy_ls = []

        #### iteration ####
        t=0
        while(t<self.max_iteration):
        # while(t<self.max_iteration):
            t+=1
            ### pick a random point and add bias term ###
            rand_index = np.random.randint(0,len(train_y))
            x = np.append(train_x[rand_index],1)
            y = train_y[rand_index]
            x = np.array([x]).T
            y = np.array([y]).T
            ### forward pass ###
            z = np.tanh((self.V.T).dot(x))
            z = np.vstack((z,[1]))
            h = expit((self.W.T).dot(z))
            ### backward pass ###
```

```
        if self.cost_function == "cross-entropy":
            delta_V,delta_W = self.Cross_entropy_delta_W(z,h,y,x,self.W)

        ### gradient descent update ###
        self.V = (1-self.regularizor)*self.V - self.learning_rate*delta_V
        self.W = (1-self.regularizor)*self.W - self.learning_rate*delta_W

        ### total training error and classification accuracy vs. iteration ###
        if t%1000==0:
            self.iteration_nums.append(t)
            if self.cost_function=="cross-entropy":
                error = self.cross_entropy_error(train_x,train_y,self.V,self.W)
                predict_labels = self.predict(train_x)
                true_labels = train_y.argmax(axis=1)
                accuracy = sum(predict_labels==true_labels)/len(true_labels)
                print(t," : ",error,accuracy)
                self.error_ls.append(error)
                self.accuracy_ls.append(accuracy)
        if t%(train_x.shape[0])==0:
            k = t/(train_x.shape[0])
            self.learning_rate = (1.0*k/(k+1))*self.learning_rate


    def Cross_entropy_delta_W(self,z,h,y,x,W):
        delta_W = z.dot((h-y).T)
        delta_V = x.dot(((W[0:-1].dot(h-y))*(1-(z[0:-1])**2)).T)
        return delta_V,delta_W

    def cross_entropy_error(self,train_x,train_y,V,W):
        ### calculate total cross entropy error ###
        X = np.column_stack((train_x, np.ones(train_x.shape[0])))
        Z = np.tanh(X.dot(V))
        Z = np.column_stack((Z, np.ones(Z.shape[0])))
        H = 1/(np.exp(-Z.dot(W))+1)
        Cross_entropy_error = np.sum(-(train_y*np.log(H)+(1-train_y)*np.log(1-H)))
        return Cross_entropy_error

    def predict(self,test_x):
        X = np.column_stack((test_x, np.ones(test_x.shape[0])))
        Z = np.tanh(X.dot(self.V))
        Z = np.column_stack((Z, np.ones(Z.shape[0])))
        H = Z.dot(self.W)
        predicted_labels = H.argmax(axis=1)
        return predicted_labels
```

```
#predict the test dataset#
final_NN = NN2(learning_rate=0.001,max_iteration=600000,cost_function="cross-
entropy",regularizor=0,n_hid=500)
final_NN.train(train_x,train_y)
test_y = final_NN.predict(test_x)
np.savetxt('letters_predict.csv', test_y, delimiter = ',')
```