Yong-Chan Shin CS189 HW6

Problem 1. Let input layer : $x_1, \dots x_{784}; x_{785}=1$ — V is 200×785
hidden layer : $h_1, \dots h_{200}; h_{201}=1$ — W is 26×201
output layer : $z_1, \dots z_{26}$

$$\begin{cases} V = V - \epsilon \nabla_V J \\ W = W - \epsilon \nabla_W J \end{cases} \quad \text{where } J(z) = \sum_i L(z_i, y_i) \quad (n=786)$$
$$= \left(-\sum_i [y_i \ln z_i + (1-y_i)\ln(1-z_i)]\right)$$

According to the architecture,

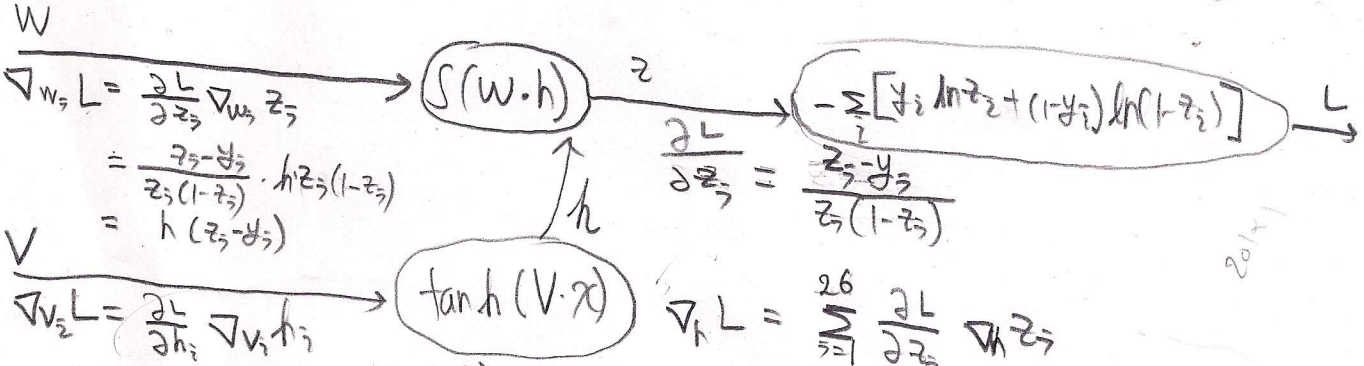
$\therefore h = \tanh(V\cdot x)$, $z = S(W\cdot h) = S(W\cdot \tanh(V\cdot x))$

We know that $\tanh'(\gamma) = \operatorname{sech}^2(\gamma)$, $S'(\gamma) = S(\gamma)(1-S(\gamma))$

$h_i = \tanh(V_i \cdot x)$, so $\nabla_{V_i} h_i = \tanh'(V_i\cdot x) = x\cdot \operatorname{sech}^2(V_i\cdot x) = x\cdot(1-h_i^2)$

$z_j = S(W_j \cdot h)$, so $\nabla_{W_j} z_j = S'(W_j\cdot h) = h\cdot z_j(1-z_j)$

$\nabla_h z_j = S'(W_j\cdot h)\cdot \frac{\partial W_j\cdot h}{\partial h} = W_j \cdot z_j(1-z_j)$



$$\nabla_h L = \sum_{j=1}^{26} \frac{\partial L}{\partial z_j}\nabla_h z_j = \sum_{j=1}^{26} \frac{z_j - y_j}{z_j(1-z_j)} W_j \cdot z_j(1-z_j) = \sum_{j=1}^{26} (z_j - y_j) W_j = W^T\cdot(z-y)$$

for $i = 1, 2, \dots 200$

for $j = 1, 2, \dots 26$

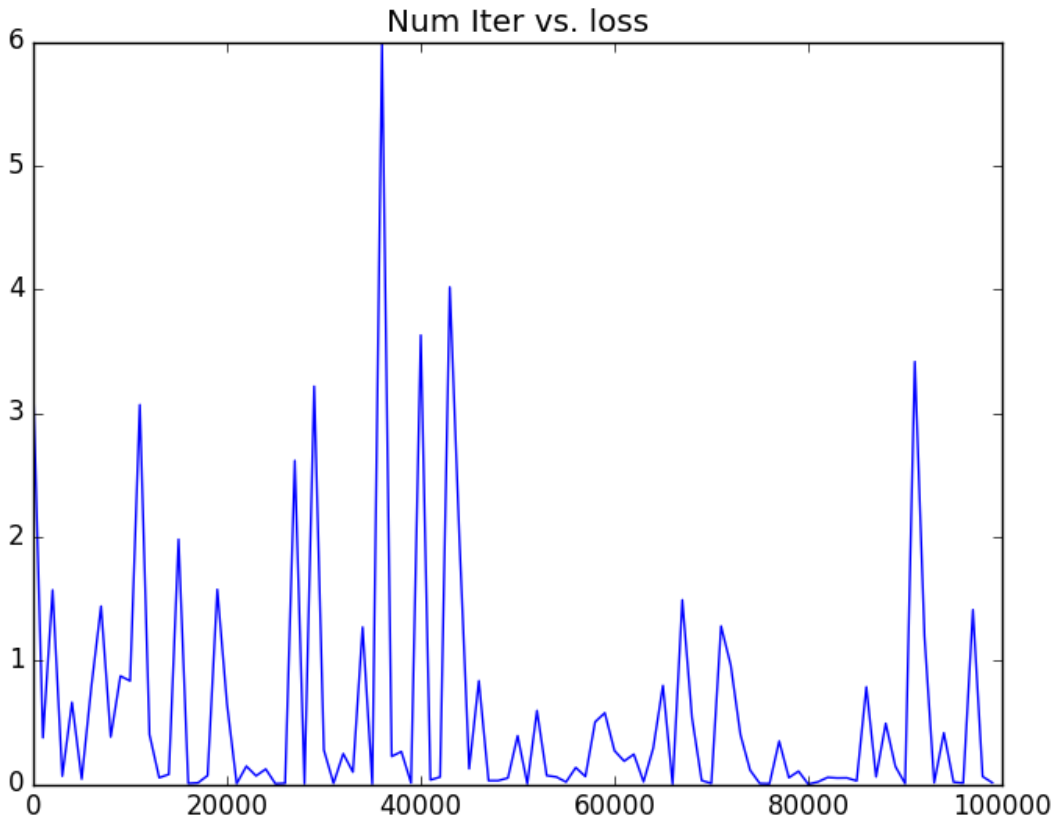
From $\nabla_{W_j} L = h(z_j - y_j)$, $\nabla_W L = (z-y)\cdot h^T$

So $V_i \leftarrow V_i - \epsilon\cdot x\cdot(W^T_i\cdot(z-y))\cdot(1-h_i)^2$ for $i = 1, 2, \dots 200$

$W \leftarrow W - \epsilon\cdot(z-y)\cdot h^T$

Problem2:

1. Modify the hidden layer size, learning rate for V and W, decay rate for the learning rate. For the Kaggle one which reached 87% accuracy used 300 hidden layer size, learning rate 0.01 for both, and 0.95 decay rate for every 1000 iteration.
2. Training accuracy: `0.8905849358974359`
3. Validation accuracy: `0.8754807692307692`
4. Plot:

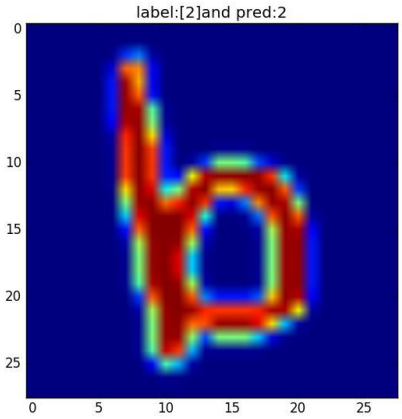


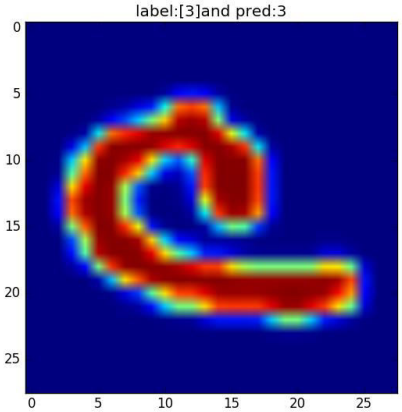
   The numbers of iterations are multiples of 1000. The loss is not monotonically decreasing but it tends to decrease overall.

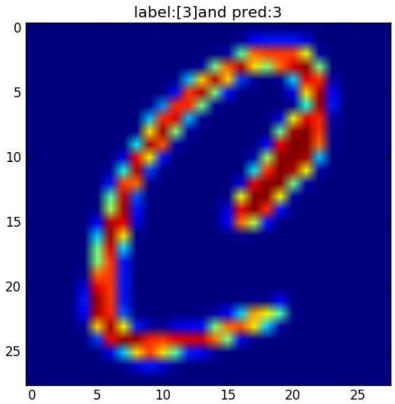
5. Kaggle score: 0.87712, Display Name: YONG-CHAN_SHIN

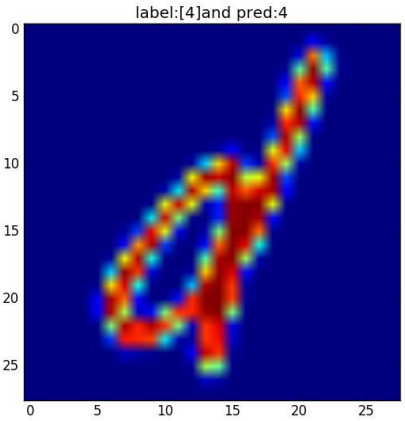
Problem3:

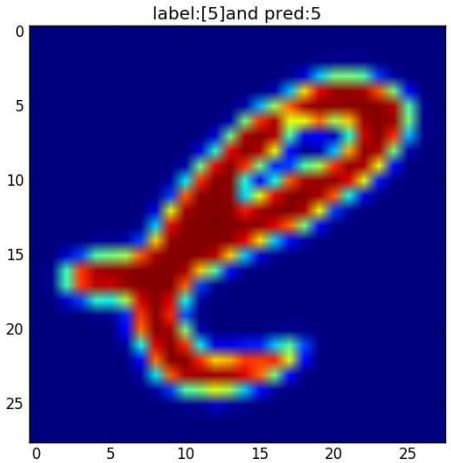
Correct predictions:



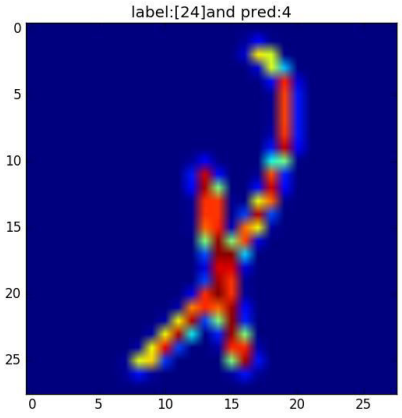


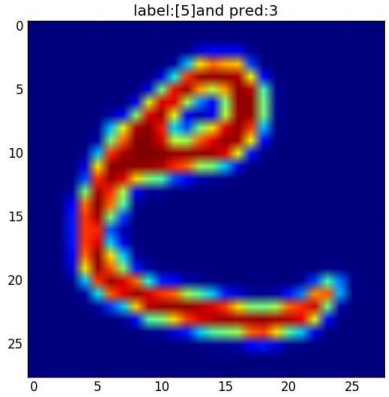


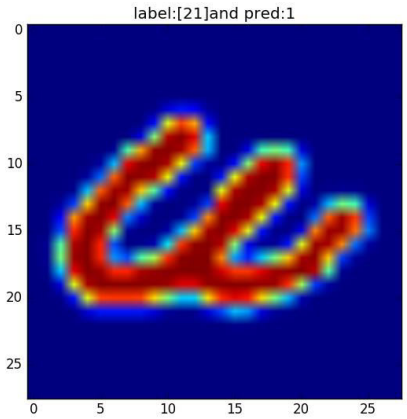


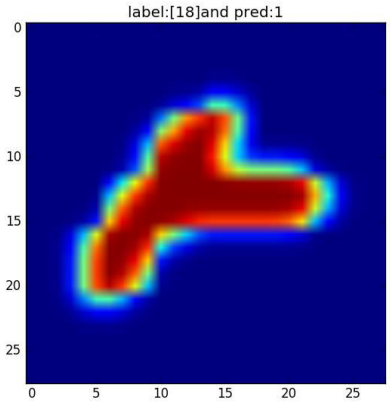


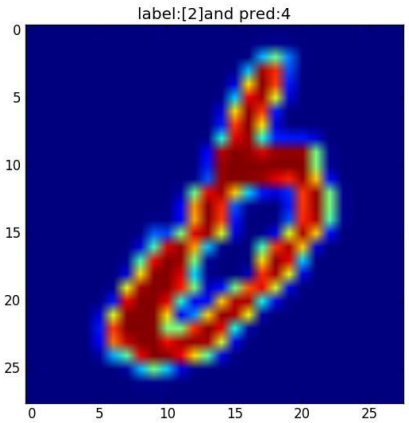
Wrong predictions:











Problem4:

1. Use different number of hidden layer units (200 to 300).
2. I used same learning rates for different layers, but I made those to decay by 0.95 for every 1000 iterations.
3. Used ReLU and softmax output units
4. Initialize the weights with mean 0, standard deviation 0.01

Code:

```
import scipy.io as scio
import matplotlib.pyplot as plt
from sklearn.utils import shuffle
import numpy as np
import pandas as pd


def compute_score(labels,pred):
    error_count = 0
    for tup in zip(pred,labels):
        if tup[0]!=tup[1]:
            error_count+=1
    return float(len(labels)-error_count)/float(len(labels))


class NeuralNetwork:
    def __init__(self,inputLayerSize,hiddenLayerSize,outputLayerSize,eps_v,eps_w):
        self.inputLayerSize = inputLayerSize
        self.hiddenLayerSize = hiddenLayerSize
        self.outputLayerSize = outputLayerSize
        self.eps_v = eps_v
        self.eps_w = eps_w
        # V is (200, 785), W is (26, 201)
        self.J = 0


        self.V = np.random.normal(0,0.01,(self.hiddenLayerSize,self.inputLayerSize + 1)) #200x785
```

```
        self.W = np.random.normal(0,0.01,(self.outputLayerSize,self.hiddenLayerSize + 1)) #26x201


    def forward_propagate(self, X):
        self.V_X = np.dot(self.V, X.reshape(X.shape[0], 1))
        self.V_X = np.ravel(self.V_X)
        #self.h = np.tanh(self.V_X)
        self.h = self.ReLU(self.V_X).T
        self.h = np.append(self.h, 1)
        self.z = np.dot(self.W, self.h.reshape(self.h.shape[0], 1))
        self.z = np.ravel(self.z)
        #yHat = self.sigmoid(self.z)
        yHat = self.softmax(self.z)
        return yHat


    def ReLU(self,z):
        return np.asarray([max(z_i,0) for z_i in z])


    def ReLU_prime(self,z):
        ret = []
        for z_i in z:
            if z_i<0:
                ret.append(0)
            else:
                ret.append(1)
```

```
        return np.asarray(ret)


def softmax(self,z):
        tot = sum([np.exp(z_i) for z_i in z])
        return np.exp(z)/tot


def tanh_prime(self, z):
        return 1 - np.tanh(z)**2


def sigmoid(self, z):
        return 1/(1+np.exp(-z))


def cross_entropy(self, X, y):
        self.yHat = self.forward_propagate(X)
        loss = sum(-(np.multiply(y,np.log(self.yHat)) + np.multiply(1.0-y, np.log(1.0-self.yHat))))
        self.J = loss
        return loss


def cross_entropy_prime(self, X, y):
        self.yHat = self.forward_propagate(X)
        #print("yhat shape",self.yHat.shape)
        delta = self.yHat-y
        #print("delat shape:",delta.shape)
        #print("h shape:",self.h.shape)
```

```
        dJdW = np.dot(delta.reshape(delta.shape[0],1),self.h.reshape(1,self.h.shape[0]))
        #print("dJdW shape",dJdW.shape)
        #delta2 = np.multiply(np.delete(np.dot(self.W.T, delta.T), 200), self.tanh_prime(self.V_X).T)
        delta2 = np.multiply(np.delete(np.dot(self.W.T, delta.T), 200), self.ReLU_prime(self.V_X).T)
        #print("delta2 shape",delta2.shape)
        delta2 = np.ravel(delta2)
        dJdV = np.dot(delta2.reshape(delta2.shape[0], 1),X.reshape(1,X.shape[0],))
        #print("dJdV shape",dJdV.shape)
        return dJdV, dJdW

    def gradients(self, X, y):
        dJdV, dJdW = self.cross_entropy_prime(X, y)
        self.cross_entropy(X, y)
        return dJdV, dJdW

    def train(self, X, y):
        gradient_V, gradient_W = self.gradients(X, y)
        self.W -= self.eps_w * gradient_W
        self.V -= self.eps_v * gradient_V

    def predict(self, X):
        return self.forward_propagate(X)

def prob2():
```

```
data = scio.loadmat('hw6_data_dist/letters_data.mat')
inputLayerSize = 784  # +1 represents bias.
hiddenLayerSize = 300  # Activation function will return 1x200 vector. Make sure to append the bias.
outputLayerSize = 26
eps_v = 0.01
eps_w = 0.01
NN = NeuralNetwork(inputLayerSize,hiddenLayerSize,outputLayerSize,eps_v,eps_w)

train_set = data['train_x']
train_labels = data['train_y']
train_set,train_labels = shuffle(train_set,train_labels)
t_set = train_set[train_set.shape[0]//5:,:]
v_set = train_set[:train_set.shape[0]//5,:]
t_labels = train_labels[train_labels.shape[0]//5:]
v_labels = train_labels[:train_labels.shape[0]//5]

losses = []
num_iter = []
for i in range(t_set.shape[0]):
    X = np.append(t_set[i], 1)
    X = (X - np.mean(X))/np.std(X)
    y = np.array([0 for _ in range(26)])
    yLabel = int(t_labels[i])-1
```

```
        y[yLabel] += 1
        NN.train(X, y)
        if i%1000==0:
            NN.eps_v *= 0.95
            NN.eps_w *= 0.95
            losses.append(NN.J)
            num_iter.append(i)
plt.title("Num Iter vs. loss")
plt.plot(num_iter,losses)
plt.show()

v_pred=[]
for i in range(v_set.shape[0]):
        X = np.append(v_set[i], 1) # (785,1) np array
        X = (X - np.mean(X))/np.std(X)
        #label = v_labels[i]
        output = np.argmax(NN.predict(X))+1
        v_pred.append(output)
v_score = compute_score(v_labels,v_pred)

t_pred=[]
for i in range(t_set.shape[0]):
        X = np.append(t_set[i], 1) # (785,1) np array
        X = (X - np.mean(X))/np.std(X)
```

```
        #label = v_labels[i]
        output = np.argmax(NN.predict(X))+1
        t_pred.append(output)
    t_score = compute_score(t_labels,t_pred)

    print("v_score: ",v_score)
    print("t_score: ",t_score)

    visualize(NN,v_set,v_labels)

def kaggle():
    data = scio.loadmat('hw6_data_dist/letters_data.mat')
    inputLayerSize = 784   # +1 represents bias.
    hiddenLayerSize = 300   # Activation function will return 1x200 vector. Make sure to append the bias.
    outputLayerSize = 26
    eps_v = 0.01
    eps_w = 0.01
    NN = NeuralNetwork(inputLayerSize,hiddenLayerSize,outputLayerSize,eps_v,eps_w)
    t0 = time.time()

    train_set = data['train_x']
    train_labels = data['train_y']
    test_set = data['test_x']
```

```
train_set,train_labels = shuffle(train_set,train_labels)
t_set = train_set
t_labels = train_labels

for i in range(t_set.shape[0]):
    X = np.append(t_set[i], 1)
    X = (X - np.mean(X))/np.std(X)
    y = np.array([0 for _ in range(26)])
    yLabel = int(t_labels[i])-1
    y[yLabel] += 1
    NN.train(X, y)
    if i%1000==0:
        NN.eps_v *= 0.95
        NN.eps_w *= 0.95

pred=[]
for i in range(test_set.shape[0]):
    X = np.append(test_set[i], 1) # (785,1) np array
    X = (X - np.mean(X))/np.std(X)
    #label = v_labels[i]
    output = np.argmax(NN.predict(X))+1
    pred.append(output)

table = {"Category":pred,"Id":np.arange(1,len(pred)+1)}
```

```
    output = pd.DataFrame(data=table)
    output.to_csv("kaggle_ycls_hw6.csv", index=False)
    print("csv created")


def visualize(NN,v_set,v_labels):
    pred=[]
    for i in range(v_set.shape[0]):
        #i = indices[j]
        X = np.append(v_set[i], 1) # (785,1) np array
        X = (X - np.mean(X))/np.std(X)
        #label = v_labels[i]
        output = np.argmax(NN.predict(X))+1
        pred.append(output)
    correct_count = 0
    incorrect_count = 0
    for i in range(len(pred)):
        if (pred[i]==1 and pred[i]==v_labels[i]) or (pred[i]==2 and pred[i]==v_labels[i]) or (pred[i]==3 and pred[i]==v_labels[i]) or (pred[i]==4 and pred[i]==v_labels[i]) or (pred[i]==5 and pred[i]==v_labels[i]): #correctly classify 'A','B','C','D', or 'E'
            if correct_count<5:
                print("correct")
                plt.figure()
                plt.imshow(v_set[i].reshape((28,28)))
                plt.title("label:"+str(v_labels[i])+"and pred:"+str(pred[i]))
```

```
                plt.show()
                correct_count+=1
        elif (pred[i]==1 and pred[i]!=v_labels[i]) or (pred[i]==2 and pred[i]!=v_labels[i]) or (pred[i]==3 and pred[i]!=v_labels[i]) or (pred[i]==4 and pred[i]!=v_labels[i]) or (pred[i]==5 and pred[i]!=v_labels[i]): #incorrectly classify 'A','B','C','D', or 'E'
            if incorrect_count<5:
                print("incorrect")
                plt.figure()
                plt.imshow(v_set[i].reshape((28,28)))
                plt.title("label:"+str(v_labels[i])+"and pred:"+str(pred[i]))
                plt.show()
                incorrect_count+=1
        if correct_count+incorrect_count>=10:
            break
#kaggle()
prob2()
```