Homework 1
IMGS 789 Deep Learning for Vision Fall 2016

Due: 9:00 PM EDT, September 30, 2016

Instructions

Your homework submission must cite any references used (including articles, books, code, websites, and personal communications). All solutions must be written in your own words, and you must program the algorithms yourself. **If you do work with others, you must list the people you worked with.** Submit your solutions as a PDF to the Dropbox Folder on MyCourses.

Your homework solution must be prepared in LaTeX and output to PDF format. I suggest using `http://overleaf.com` or BaKoMa TeX to create your document. Overleaf is free and can be accessed online.

Your programs must be written in either MATLAB or Python. The relevant code to the problem should be in the PDF you turn in. If a problem involves programming, then the code should be shown as part of the solution to that problem. One easy way to do this in LaTeX is to use the verbatim environment, i.e., \begin{verbatim} YOUR CODE \end{verbatim}

If you have forgotten your linear algebra, you may find *The Matrix Cookbook* useful, which can be readily found online. You may wish to use the program *MathType*, which can easily export equations to AMS LaTeX so that you don't have to write the equations in LaTeX directly: `http://www.dessci.com/en/products/mathtype/`

If told to implement an algorithm, don't use a toolbox, or you will receive no credit.

Problem 1 - Softmax Properties

Part 1 (7 points)

Recall the softmax function, which is the most common activation function used for the output of a neural network trained to do classification. In a vectorized form, it is given by

$$\text{softmax}(\mathbf{a}) = \frac{\exp(\mathbf{a})}{\sum_{j=1}^{K} \exp(a_j)},$$

where $\mathbf{a} \in \mathbb{R}^K$. The exp function in the numerator is applied element-wise and a_j denotes the j'th element of $\mathbf{a}$.

Show that the softmax function is invariant to constant offsets to its input, i.e.,

$$\text{softmax}(\mathbf{a} + c\mathbf{1}) = \text{softmax}(\mathbf{a}),$$

where $c \in \mathbb{R}$ is some constant and $\mathbf{1}$ denotes a column vector of 1's.

Solution:

$$\text{softmax}(\mathbf{a}) = \frac{\exp(\mathbf{a})}{\sum_{j=1}^{K} \exp(a_j)},$$

$$\text{softmax}(\mathbf{a} + \mathbf{c1}) = \frac{\exp(\mathbf{a} + \mathbf{c1})}{\sum_{j=1}^{K} \exp(a_j + c1)},$$

$$\text{softmax}(\mathbf{a} + \mathbf{c1}) = \frac{\exp(\mathbf{a})}{\sum_{j=1}^{K} \exp(a_j)},$$

Cancelling exp(c1) from both numerator and denominator

$$\text{softmax}(\mathbf{a} + \mathbf{c1}) = softmax(a)$$

Part 2 (3 points)

In practice, why is the observation that the softmax function is invariant to constant offsets to its input important when implementing it in a neural network?

Solution:

It is important to maintain numerical stability when implementing a neural network.

Problem 2 - Implementing a Softmax Classifier

For this problem, you will use the 2-dimensional Iris dataset. Download `iris-train.txt` and `iris-test.txt` from MyCourses. Each row is one data instance. The first column is the label (1, 2 or 3) and the next two columns are features.

Part 1 - Implementation & Evaluation (30 points)

Recall that a softmax classifier is a shallow one-layer neural network of the form:

$$P\left(C=k|\mathbf{x}\right)=\frac{\exp\left(\mathbf{w}_k^T\mathbf{x}\right)}{\sum_{j=1}^{K}\exp\left(\mathbf{w}_j^T\mathbf{x}\right)}$$

where $\mathbf{x}$ is the vector of inputs, K is the total number of categories, and $\mathbf{w}_k$ is the weight vector for category k.

In this problem you will implement a softmax classifier from scratch. **Do not use a toolbox.** Use the softmax (cross-entropy) loss with L_2 weight decay regularization. Your implementation should use stochastic gradient descent with mini-batches and momentum to minimize softmax (cross-entropy) loss of this single layer neural network. To make your implementation fast, do as much as possible using matrix and vector operations. This will allow your code to use your environment's BLAS. Your code should loop over epochs and mini-batches, but do not iterate over individual elements of vectors and matrices. Try to make your code as fast as possible. I suggest using profiling and timing tools to do this.

Train your classifier on the Iris dataset for 1000 epochs. Hand tune the hyperparameters (i.e., learning rate, mini-batch size, momentum rate, and L_2 weight decay factor) to achieve the best possible training accuracy. During a training epoch, your code should compute the mean per-class accuracy for the training data and the loss. After each epoch, compute the mean per-class accuracy for the testing data and the loss as well. **The test data should not be used for updating the weights.**

After you have tuned the hyperparameters, generate two plots next to each other. The one on the left should show the cross-entropy loss during training for both the train and test sets as a function of the number of training epochs. The plot on the right should show the mean per-class accuracy as a function of the number of training epochs on both the train set and the test set.

What is the best test accuracy your model achieved? What hyperparameters did you use? Would early stopping have helped improve accuracy on the test data?

Solution:

Accuracy obtained = 80%

Learning rate = 0.005

Momentum = 0.05

L2 = 0.001

batch size = 10

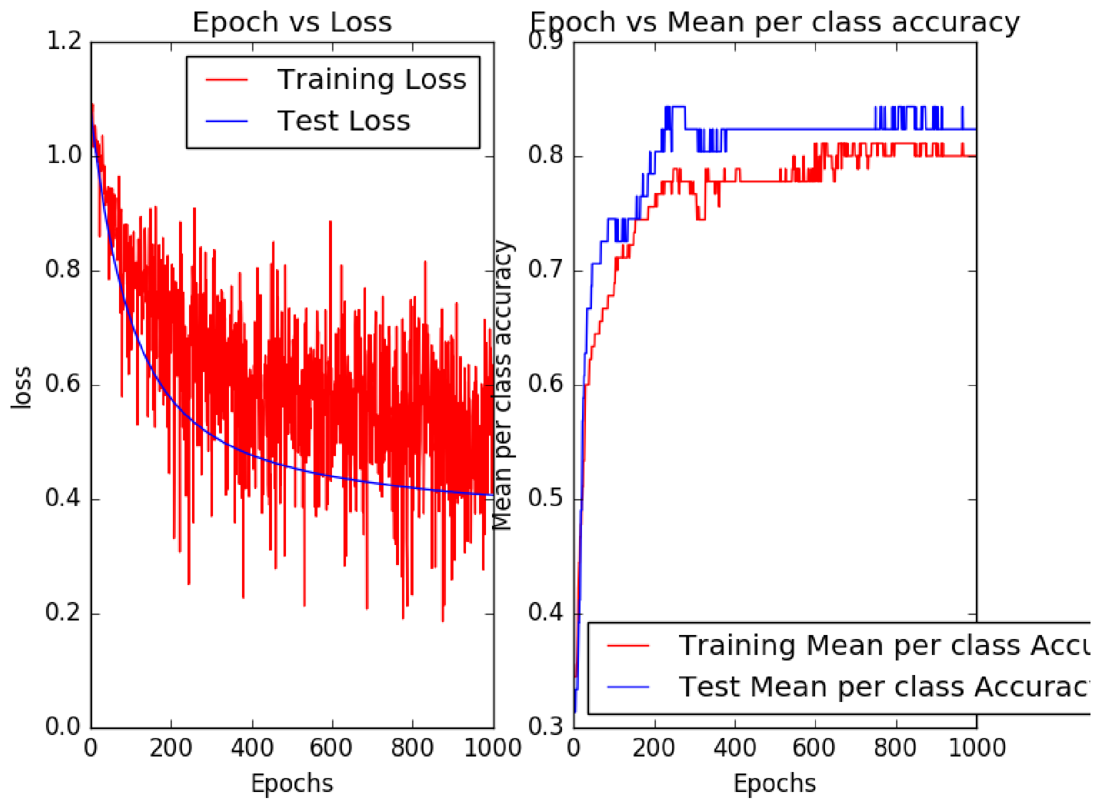
Early stopping could have been better.

```
import matplotlib.pyplot as plt
import numpy as np
import random
import time
import pdb;

train_file = "./iris-train.txt" # "/Users/sanjanakapistalam/Desktop/Deeplearni
test_file = "./iris-test.txt"
train_rows = open(train_file).read().splitlines()
test_rows = open(test_file).read().splitlines()

D = 2 #Dimensionality
K = 3 #No of classes/categories

#Randomly initializing parameters
W = 0.01 * np.random.randn(D,K) #Weights
dW = 0
alpha = 0.05
s_size = 1
L2 = 0.001
lr = 0.005
bsize = 10
all_tr_loss = []
all_tst_loss = []
train_res = []
test_res = []
for num in xrange(1000):
    np.array(random.shuffle(train_rows)) #Shuffling the train text file for ea
    splitlines = [x.strip().split(' ') for x in train_rows]
    train_cls = [x[0] for x in splitlines] #All classes
    arr_traincls = np.array(train_cls,dtype = np.uint8) #As an array
```

```
    trainf = [(x[1],x[2]) for x in splitlines] #All features
    arr_trainf = np.array(trainf,dtype=np.float32) #As an array
    np.array(random.shuffle(test_rows)) #Shuffling the test text file for each
    splitlines_test = [x.strip().split(' ') for x in test_rows]
    test_cls = [x[0] for x in splitlines_test]
    arr_testcls = np.array(test_cls,dtype = np.uint8)
    testf = [(x[1],x[2]) for x in splitlines_test]
    arr_testf = np.array(testf,dtype=np.float32)

    for ix in xrange(bsize-1):
        int_tr_cls = np.zeros((bsize,K),dtype = np.uint8)
        int_tst_cls = np.zeros((bsize,K),dtype = np.uint8)
        start = (ix*bsize)
        stop = ((ix+1)*10)
        stop = min(stop,arr_trainf.size)
        #import pdb;pdb.set_trace()
        tr_scores = np.dot(arr_trainf[start:stop,[0,1]],W)
        for ind,i in enumerate(arr_traincls[start:stop]):
            int_tr_cls[ind,i-1] = 1

        t_cls = int_tr_cls
        tr_scores -= np.max(tr_scores)
        out = np.exp(tr_scores)
        sum_  = out.sum(axis=1)
        probs = out / sum_[:, np.newaxis]
        #import pdb;pdb.set_trace()
        tr_loss = -np.log( np.max(probs,0.0000001) ) * t_cls
        diff_loss = -np.dot(arr_trainf[start:stop].T,t_cls-probs)

        rloss = 0.5*L2*np.sum(W*W)
        tloss = (np.sum(tr_loss)/bsize) + rloss

        dW = (alpha*dW) + (lr*diff_loss) #(L2*W) + (diff_loss)
        W  = W - dW
    all_tr_loss.append(tloss)
    int_tst_cls = np.zeros((len(arr_testf),K),dtype = np.uint8)
    tst_scores = np.dot(arr_testf[:,[0,1]],W)
    for index,val in enumerate(arr_testcls):
        int_tst_cls[index,val-1] = 1
    tst_cls = int_tst_cls
    tst_scores -= np.max(tst_scores)
```

```
    tst_out = np.exp(tst_scores)
    tst_sum_ = tst_out.sum(axis=1)
    tst_probs = tst_out / tst_sum_[:,np.newaxis]
    tst_loss = -np.log( np.max(tst_probs,0.0000001) ) * tst_cls
    t_tst_loss = (np.sum(tst_loss)/len(arr_testf)) + rloss
    all_tst_loss.append(t_tst_loss)
    if 0%10 == 0:
      print "iteration %d: train loss %f" % (num,tloss)
      print "iteration %d: test loss %f" % (num,t_tst_loss)
    import pdb;pdb.set_trace()
    scores_tr = np.dot(arr_trainf[:,[0,1]],W)
    pred_cls_tr= np.argmax(scores_tr,axis=1)+1
    train_acc = np.mean(pred_cls_tr == arr_traincls)
    train_res.append(train_acc)
    print 'training accuracy: %.2f' % (train_acc) #(np.mean(pred_cls == arr_tr
    scores_tst = np.dot(arr_testf[:,[0,1]],W)
    pred_cls_tst = np.argmax(scores_tst,axis=1)+1
    test_acc = np.mean(pred_cls_tst == arr_testcls)
    import pdb;pdb.set_trace()
    test_res.append(test_acc)
    print 'test accuracy: %.2f' % (test_acc)
    time.sleep(0.1)
#-----------------Decision Boundaries(2b)--------
#import pdb;pdb.set_trace()
up_W = W
h = .02
X = arr_trainf[:,:2]
Y = arr_traincls
x_min, x_max = X[:, 0].min() - .5, X[:, 0].max() + .5
y_min, y_max = X[:, 1].min() - .5, X[:, 1].max() + .5
Data = arr_trainf[:,:2]
xx, yy = np.meshgrid(np.arange(x_min, x_max, h),np.arange(y_min, y_max, h))
arr = np.array([xx.ravel(),yy.ravel()])
Score = np.dot(arr.T,up_W)
Score -= np.max(Score)
out1 = np.exp(Score)
sum1_  = out1.sum(axis=1)
prob = out1 / sum1_[:, np.newaxis]
Z = np.argmax(prob,axis=1)+1
Z = Z.reshape(xx.shape)
plt.figure(1,figsize=(4,3))
```

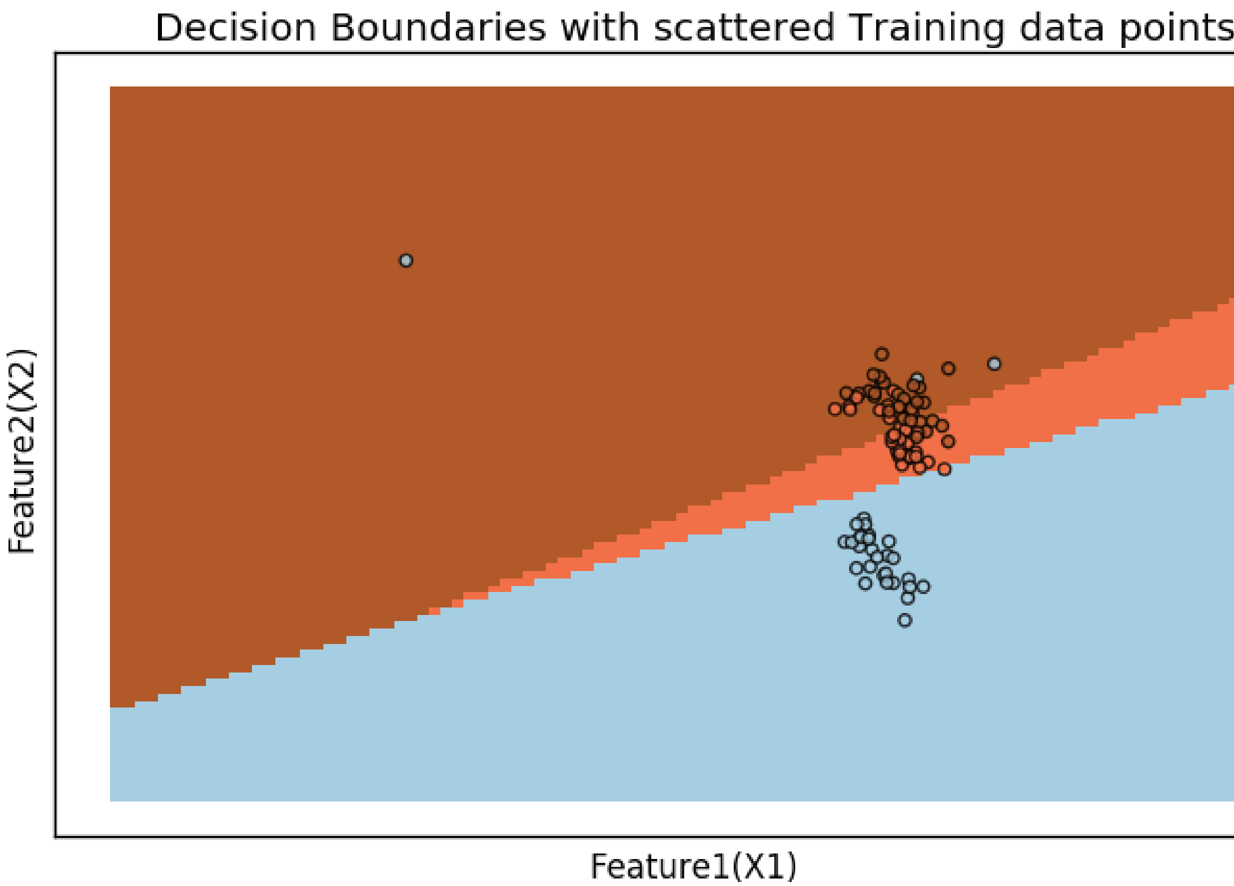
```
plt.pcolormesh(xx,yy,Z,cmap=plt.cm.Paired)
plt.scatter(X[:,0],X[:,1],c=Y,edgecolors='k',alpha=0.8,cmap=plt.cm.Paired)
plt.xlabel('Feature1(X1)')
plt.ylabel('Feature2(X2)')
plt.title('Decision Boundaries with scattered Training data points')
plt.xlim(xx.min(), xx.max())
plt.ylim(yy.min(), yy.max())
plt.xticks(())
plt.yticks(())
plt.axis("tight")
plt.show()

#-------loss plots (2a)------------

plt.subplot(1,2,1)
list = range(1000)
#import pdb;pdb.set_trace()
plt.plot(list,all_tr_loss,'-',color='r',label='Training Loss')
plt.plot(list,all_tst_loss,'-',color='b',label='Test Loss')
plt.title('Epoch vs Loss')
plt.xlabel('Epochs')
plt.ylabel('loss')
plt.legend(loc='best')
plt.subplot(1,2,2)
plt.plot(list,train_res,'-',color='r',label='Training Mean per class Accuracy'
plt.plot(list,test_res,'-',color='b',label='Test Mean per class Accuracy')
plt.title('Epoch vs Mean per class accuracy')
plt.xlabel('Epochs')
plt.ylabel('Mean per class accuracy')
plt.legend(loc='best')
plt.show()
```



Part 2 - Displaying Decision Boundaries (10 points)

Plot the decision boundaries learned by softmax classifier on the Iris dataset, just like we saw in class. On top of the decision boundaries, generate a scatter plot of the training data. Make sure to label the categories.

Solution:



Problem 3 - Classifying Images

The CIFAR-10 dataset contains 60,000 RGB images from 10 categories. Download it from here: `https://www.cs.toronto.edu/~kriz/cifar.html`
Read the documentation.

Part 1 (10 points)

Using the first CIFAR-10 training batch file, display the first three images from each of the 10 categories as a 3×10 image array. The images are stored as rows, and you will need to reshape them into $32 \times 32 \times 3$ images.

Solution:

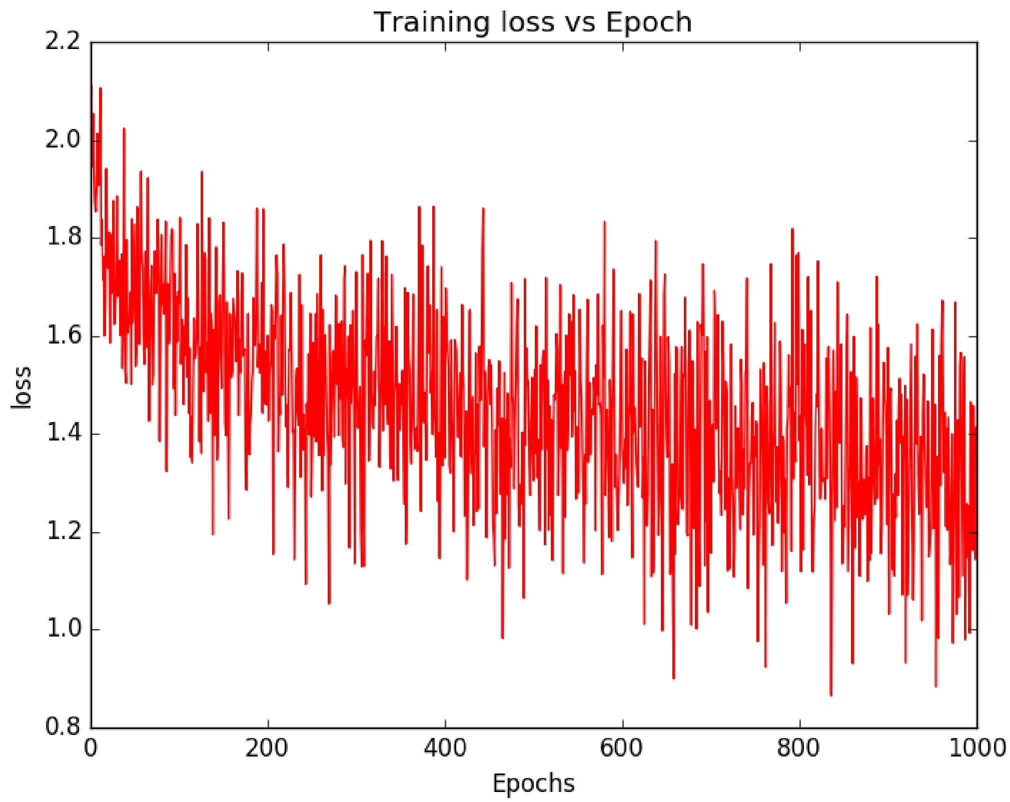
```python
import matplotlib.pyplot as plt
import numpy as np
import cPickle
import sys
file = "./cifar-10-batches-py/data_batch_1"
f_open = open(file,'rb')
dict = cPickle.load(f_open)
img_data = dict['data']
classes = dict['labels']
img_cls = np.array(classes)
imgs = []
for i in range(3):
    for ind,val in enumerate(np.unique(classes)):
        cls_ind = np.where(img_cls == ind)
        im = img_data[cls_ind][i].reshape(3,32,32).transpose(1,2,0)
        imgs.append(im)
for img in range(30):
    plt.subplot(3,10,img+1)
    #import pdb;pdb.set_trace()
    plt.imshow(imgs[img])
    plt.axis('off')
plt.tight_layout()
plt.show()
```

Part 2 (20 points)

Using the softmax classifier you implemented, train the model on CIFAR-10's training partitions. To do this, you will need to treat each image as a vector. You will need to tweak the hyperparmaters you used earlier.

Plot the training loss as a function of training epochs. Try to minimize the error as much as possible. What were the best hyperparmeters? Output the final test accuracy and a normalized 10×10 confusion matrix computed on the test partition. Make sure to label the columns and rows of the confusion matrix.

Solution:



Softmax Classifier Code Appendix

```
import matplotlib.pyplot as plt
import numpy as np
import cPickle
import sys
import random
import pandas as pd


file1 = "./cifar-10-batches-py/data_batch_1"
f1_open = open(file1,'rb')
dict1 = cPickle.load(f1_open)
data1 = dict1['data']
classes1 = dict1['labels']
```

```
imgs_cls1 = np.array(classes1)

#-----file2-----
file2 = "./cifar-10-batches-py/data_batch_2"
f2_open = open(file2,'rb')
dict2 = cPickle.load(f2_open)
data2 = dict1['data']
classes2 = dict2['labels']
imgs_cls2 = np.array(classes2)

#-----file3------
file3 = "./cifar-10-batches-py/data_batch_3"
f3_open = open(file3,'rb')
dict3 = cPickle.load(f3_open)
data3 = dict3['data']
classes3 = dict3['labels']
imgs_cls3 = np.array(classes3)

#-----file4------
file4 = "./cifar-10-batches-py/data_batch_4"
f4_open = open(file4,'rb')
dict4 = cPickle.load(f4_open)
data4 = dict4['data']
classes4 = dict4['labels']
imgs_cls4 = np.array(classes4)

#------file5------
file5 = "./cifar-10-batches-py/data_batch_5"
f5_open = open(file5,'rb')
dict5 = cPickle.load(f5_open)
data5 = dict5['data']
classes5 = dict5['labels']
imgs_cls5 = np.array(classes5)

#----------------Test file ---------
test_file = "./cifar-10-batches-py/test_batch"
tf_open = open(test_file,'rb')
test_dict = cPickle.load(tf_open)
test_data = test_dict['data']
test_cls = test_dict['labels']
tst_img_cls = np.array(test_cls)
```

```

#———Implementing Softmax————
D = 3072 #Dimensionality
K = 10 #No of classes/categories

#———Randomly initializing parameters
W = 0.01 * np.random.randn(D,K) #Weights
dW = 0
alpha = 0.9
s_size = 1
L2 = 0.0005#0.00001
lr = 0.0001#1e-05,0.001,0.0005
bsize = 100
all_tr_loss = []
all_tst_loss = []
train_res = []
pred_cls = []
all_images = np.vstack((data1,data2,data3,data4,data5))
all_classes = np.hstack((imgs_cls1,imgs_cls2,imgs_cls3,imgs_cls4,imgs_cls5))
for num in xrange(1000):
    all_list = list(zip(all_images,all_classes))
    random.shuffle(all_list)
    all_images,all_classes = zip(*all_list)
    all_images = np.array(all_images)
    all_classes = np.array(all_classes)
    test_list = list(zip(test_data,test_cls))
    random.shuffle(test_list)
    test_data,test_cls = zip(*test_list)
    test_data = np.array(test_data)
    test_cls = np.array(test_cls)
    for imgs in xrange(bsize-1):
        class_arr = np.zeros((bsize,K),dtype = np.uint8)
        start = (imgs*bsize)
        stop = ((imgs+1)*bsize)
        stop = min(stop,all_images.size)
        X = all_images[start:stop]
        X = X/255 #normalizing

        score = np.dot(X[:,:],W)
        for ind,i in enumerate(all_classes[start:stop]):
            class_arr[ind,i-1] = 1
```

```
        #import pdb;pdb.set_trace()
        all_cls = class_arr
        score -= np.max(score)
        out = np.exp(score)
        sum_  = out.sum(axis=1)
        probs = out / sum_[:, np.newaxis]
        tr_loss = -np.log( np.max(probs,0.0000001) ) * all_cls
        diff_loss = -np.dot(X.T, all_cls-probs)
        rloss = 0.5*L2*np.sum(W*W)
        tloss = (np.sum(tr_loss)/bsize) + rloss

        dW = (alpha*dW) + (lr*diff_loss)#Implementing momentum
        W  = W - dW#Updating initialized weights
    all_tr_loss.append(tloss)
    if 0%100 == 0:
        print "iteration %d: train loss %f" % (num, tloss)

    scores_tr = np.dot(all_images,W)
    pred_cls_tr= np.argmax(scores_tr,axis=1)+1
    train_acc = np.mean(pred_cls_tr == all_classes)#all_classes[img_arr][start
    train_res.append(train_acc)
    print 'training accuracy: %.2f' % (train_acc)

    scores_tst = np.dot((test_data[:,:])/255,W)
    pred_cls_tst = np.argmax(scores_tst,axis=1)+1
    pred_cls.append(pred_cls_tst)
    test_acc = np.mean(pred_cls_tst == tst_img_cls)
    print 'test accuracy: %.2f' % (test_acc)
#-------Loss plot----------
list = range(1000)
plt.figure()
plt.plot(list,all_tr_loss,'-',color='r',label='Training Loss')
plt.xlabel('Epochs')
plt.ylabel('loss')
plt.title('Training loss vs Epoch')
plt.show()
#-------------Confusion matrix
#import pdb;pdb.set_trace()
act_cls = test_cls
y_actu = pd.Series(act_cls, name='Actual')
y_pred = pd.Series(pred_cls[0][:].tolist(), name='Predicted')
```

```
df_confusion = pd.crosstab(y_actu, y_pred)
df_conf_norm = df_confusion / df_confusion.sum(axis=1)
plt.matshow(df_conf_norm)
plt.colorbar()
plt.show()
```