# CS 5787 Assignment 1

**Ran Ji**
**rj369@**

# Problem 1

## Part 1

$$softmax(a+c) = \frac{exp(a+c)}{\sum_{j=1}^{K} exp(aj+c)} \tag{1}$$

$$= \frac{exp(a)*exp(c)}{\sum_{j=1}^{K} exp(aj)*exp(c)} \tag{2}$$

$$= \frac{exp(a)*exp(c)}{exp(c)*\sum_{j=1}^{K} exp(aj)} \tag{3}$$

$$= \frac{exp(a)}{\sum_{j=1}^{K} exp(aj)} \tag{4}$$

$$= softmax(a) \tag{5}$$

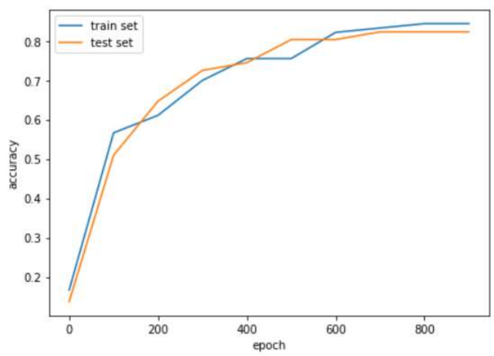
## Part 2

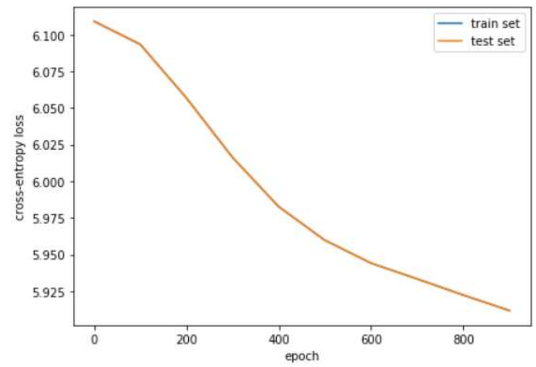
**In practice, why is the observation that the softmax function is invariant to constant offsets to its input important when implementing it in a neural network?**

As the softmax function is invariant to constant offset, so the neural network doesn't need to worry about different offsets and the same information will be passed to next layer.

# Problem 2

**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?**

A: I achieved the 0.8235 in test accuracy.

```
In [138]: predict_all_score(X_train, y_train)
Out[138]: 0.8111111111111111

In [139]: predict_all_score(X_test, y_test)
Out[139]: 0.8235294117647058
```

**What hyperparameters did you use?**

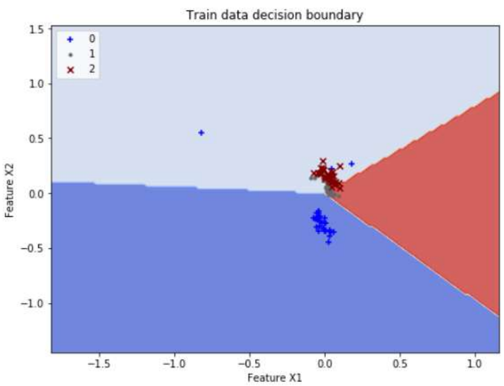
A: I used learning rate = 0.05, weight decay parameter = 0.00005, and moment parameter = 0

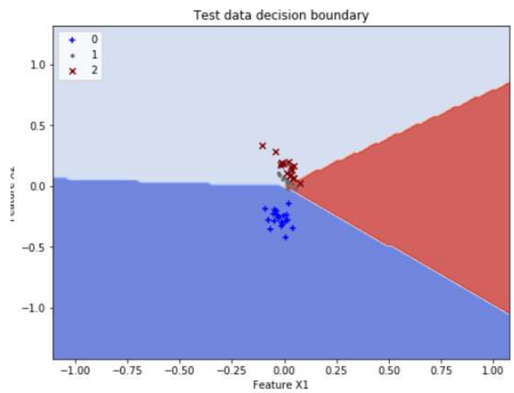
**Would early stopping have helped improve accuracy on the test data?**

A: In my case, the early stopping does not help improving accuracy on the test data as shown by the figure. The accuracy increases and coverages as number of epochs goes up.

**Part 2 - Displaying Decision Boundaries (10 points)**

**Plot the decision boundaries learned by softmax classier 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.**



## Problem 3

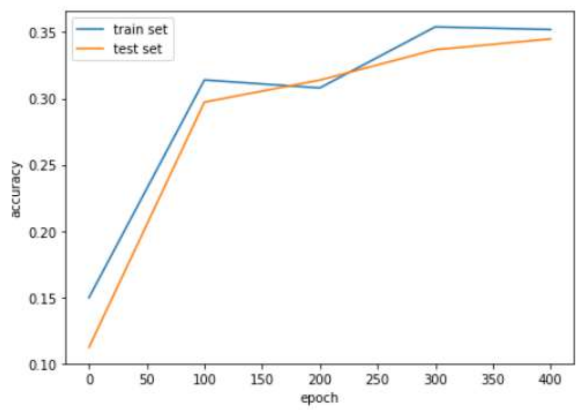
## Visualizing and Loading CIFAR-10

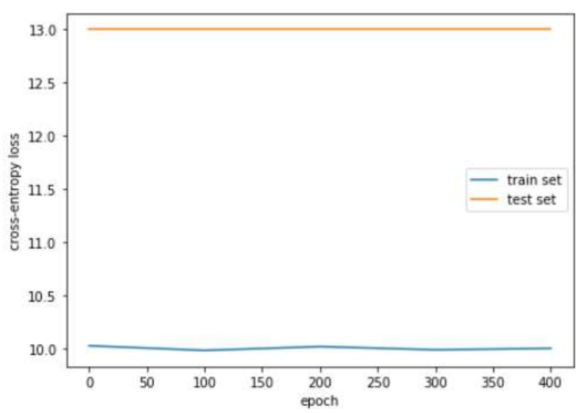
```
In [10]: rows = 3
cols = 10
fig, axes = plt.subplots(rows, cols, figsize=(12,6))

image_to_show = []
for label in range(10):
    images = find_3_image(label)
    image_to_show += images

index = -1
def getImage():
    global index
    index = index + 1
    return image_to_show[index]

for i in range(rows*cols):
    row_index = i//cols
    col_index = i%cols
    ax = axes[row_index, col_index]
    img = getImage()
    img = Image.fromarray(img, 'RGB')
    ax.imshow(img)
```

# Problem 4 - Classifying Images

```
In [74]: predict_all_score(trainFeat, trainLabels)
Out[74]: 0.3699

In [75]: predict_all_score(testFeat, testLabels)
Out[75]: 0.3642
```

**Plot the training loss as a function of training epochs.**




**What were the best hyperparameters?**

Learning rate = 0.001

weight decay parameter = 0. 000001

moment parameter = 0.0001

**Output the final test accuracy and a normalized 10 x 10 confusion matrix computed on the test partition. Make sure to label the columns and rows of the confusion matrix.**

# Problem 5

### What is the range of the variables?

Below is the range for each feature:

```
max value is 3.057125650919129, min value is -6.849083082782316
max value is 7.416758357286213, min value is -6.555528340337723
max value is 8.909800665090884, min value is -8.780884858044628
max value is 17.655495566858843, min value is -9.508265956216096
max value is 11.751147673557183, min value is -7.675992494779988
max value is 10.078955827995857, min value is -5.626714114321393
max value is 11.988035286371067, min value is -12.745385152626366
max value is 13.439546015479719, min value is -8.88108949454268
max value is 13.477656715664258, min value is -12.30651844082854
max value is 8.964286659489355, min value is -6.669483449570478
max value is 7.090613474622595, min value is -15.9173020921891
max value is 10.256871402741275, min value is -11.603864053505609
max value is 23.21916834016465, min value is -1.508789561254468
max value is 36.21243544046178, min value is -1.3886761916226775
max value is 27.622341191920558, min value is -1.5406665361821796
max value is 27.75638274198071, min value is -1.367072051717689
max value is 39.85768818633951, min value is -1.8888189736697154
max value is 27.664856069047893, min value is -1.5121923551980918
max value is 31.772130622781464, min value is -1.8416672507025469
max value is 29.33541309096434, min value is -1.6522304138211832
max value is 43.08267110077384, min value is -1.8107157957520523
max value is 20.567371293838274, min value is -1.8753572017985967
max value is 34.65085989710159, min value is -1.5158824100031694
max value is 61.936398065034204, min value is -1.8667518502902354
max value is 16.497829487098826, min value is -23.55236406964639
max value is 34.16368043201125, min value is -18.759232761884526
max value is 26.701295863607218, min value is -21.965449084703526
max value is 15.760863459929636, min value is -19.695013015868103
max value is 19.984411722202072, min value is -20.651214124107554
max value is 26.01622520953935, min value is -23.43217250821948
max value is 18.38171921207375, min value is -38.4337429715435
max value is 26.913289751090524, min value is -21.47552136986284
max value is 38.658430819091876, min value is -22.686796102706356
max value is 43.66878622365821, min value is -29.676686029337745
max value is 42.73969111325339, min value is -23.35009041137857
max value is 22.451164264136764, min value is -16.5294506387842
max value is 34.17133567774759, min value is -20.20458339760652
max value is 17.535609387550473, min value is -22.550542377818264
max value is 23.981276751287915, min value is -36.8607413837146
max value is 40.57390903479269, min value is -38.477315834136235
max value is 29.48359273980198, min value is -38.92737597745639
max value is 32.351904390614884, min value is -35.026191295936
max value is 17.043481982632706, min value is -66.89825253913575
max value is 47.62984167161246, min value is -28.048853701136
max value is 18.001136385216558, min value is -24.534094342025302
max value is 21.730001975546877, min value is -24.355785899112746
max value is 19.113034444078647, min value is -31.7945548721705
max value is 21.653162468970763, min value is -15.77575612909508
max value is 20.09676129348846, min value is -31.274700917425
```

```
max value is 16.361524394112234, min value is -19.422051910864024
max value is 15.17976248861029, min value is -24.331477136593932
max value is 32.374908178003125, min value is -23.62533019782441
max value is 34.03531712974816, min value is -45.735719339860665
max value is 34.72800284650435, min value is -26.374837355985363
max value is 10.03385444023951, min value is -29.17142640360085
max value is 25.203252073269628, min value is -16.825677314137348
max value is 28.331337340840626, min value is -16.821773787690855
max value is 48.550516051911494, min value is -17.66427470691089
max value is 31.864840200438696, min value is -18.104405235680094
max value is 39.698722120388, min value is -17.159926732185106
max value is 30.499482118753228, min value is -32.106106262404694
max value is 19.807472446621365, min value is -23.008332511230975
max value is 21.596164285026628, min value is -15.480212007073094
max value is 36.38139052991565, min value is -33.19882527793925
max value is 12.953710142993996, min value is -33.19510800657213
max value is 26.8789745171673, min value is -30.384861045790455
max value is 20.563017884999763, min value is -12.046487422828838
max value is 40.80568144992973, min value is -22.264477671767306
max value is 49.62664979080883, min value is -18.291660748609335
max value is 20.31257705619953, min value is -25.298460868485
max value is 15.779512097107556, min value is -28.03941230474677
max value is 20.654769525357306, min value is -30.349222727979715
max value is 13.12080070493906, min value is -51.06792320698983
max value is 26.135332370617878, min value is -28.496081579177115
max value is 20.092760735263408, min value is -21.112497939863914
max value is 28.34786936625298, min value is -27.06916534552786
max value is 37.4721516352411, min value is -36.59342209613789
max value is 23.52087258429061, min value is -14.220933006933384
max value is 24.059752970279526, min value is -41.28415783417706
max value is 29.90592306709198, min value is -38.567131237118566
max value is 25.719032946393092, min value is -14.132656043786438
max value is 25.90149780010079, min value is -24.685748553422865
max value is 26.02149503890222, min value is -15.206857931705324
max value is 17.23188754215112, min value is -33.055586355207154
max value is 16.10764598262076, min value is -21.179812516019254
max value is 31.87491321418461, min value is -27.869697708719848
max value is 16.41890153624518, min value is -24.649402777067586
max value is 34.45497067212309, min value is -18.05302343084412
max value is 39.89136634657852, min value is -40.456485512449646
max value is 27.19128707079417, min value is -14.494933136155161
```

### How might you normalize them?

I subtract each year by the mean average of the training data.

For each feature, I subtract each feature by its mean and then divide the standard derivation.

In [26]:
```python
# normalize data
# trainFeat = normalize(trainFeat, axis=1, norm='l1')
# testFeat = normalize(testFeat, axis=1, norm='l1')

trainFeat_temp = trainFeat - trainFeat.mean(axis=0)
trainFeat_temp = trainFeat_temp / np.std(trainFeat, axis=0)

testFeat_temp = testFeat - trainFeat.mean(axis=0)
testFeat_temp = testFeat_temp / np.std(trainFeat, axis=0)

trainFeat = np.append(trainFeat_temp,np.ones([len(trainFeat_temp),1]),1)
testFeat = np.append(testFeat_temp,np.ones([len(testFeat_temp),1]),1)
```

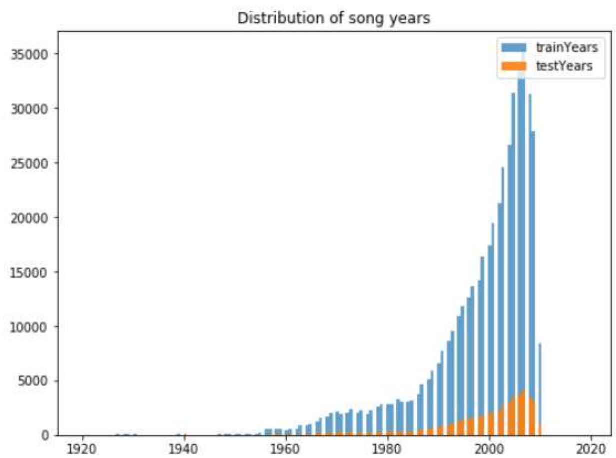
In [61]:
```python
trainYearAverage = trainYears.mean()
```

In [62]:
```python
trainYears = trainYears - trainYearAverage
testYears = testYears - trainYearAverage
```

**What years are represented in the dataset?**

```
Min year is 1922
Max year is 2011
All years are dict_keys([2001, 2007, 2008, 2002, 2004, 2003, 1999, 1992, 1997, 1987, 2000, 2005, 1996, 1998, 2009, 2006, 1993,
1991, 1933, 1930, 1935, 1995, 1941, 1990, 1943, 1994, 1974, 1976, 1975, 1970, 1971, 1981, 1989, 1969, 1972, 1973, 1983, 2010, 1
985, 1988, 1979, 1980, 1986, 1958, 1978, 1968, 1962, 1967, 1982, 1984, 1961, 1966, 1964, 1960, 1965, 1963, 1977, 1942, 1945, 19
55, 1926, 1927, 1957, 1959, 1956, 1954, 1928, 1948, 1922, 1952, 1953, 1944, 1946, 1949, 1950, 1939, 1932, 1938, 1937, 1936, 194
0, 1951, 1929, 1934, 1947, 1931, 1925, 1924, 2011])
```

**Generate a histogram of the labels in the train and test set and discuss any years or year ranges that are under/over-represented.**




**What will the test mean squared error (MSE) be if your classier always outputs the most common year in the dataset? What will the test MSE be if your classier always outputs 1998, the rounded mean of the years?**

```
In [25]: mse_mean = musicMSE(train_years_list, [1998]*len(train_years_list))
         print("MSE that always outputs 1998 (mean) is "+str(mse_mean))

         mse_most_common = musicMSE(train_years_list, [most_common_key]*len(train_years_list))
         print("MSE that always outputs the most common key "+str(most_common_key)+ " is "+str(mse_most_common))

         MSE that always outputs 1998 (mean) is 119.82739576549339
         MSE that always outputs the most common key 2007 is 193.87802179791854
```

## Part 2 - Ridge Regression

Test music MSE is 90.59

```
def sgd(x, y_pred, y):
    global w, alpha
    n = x.shape[0]
    y = y.reshape((y.shape[0], 1))
    value = alpha * w
    value -= 2 * (x * (y - y_pred)).sum(0).reshape((91, 1))
    w -= lr * value/n
```

```
In [81]: def train(X_train2, y_train2):
             global loss_value_previous, loss_value_current, w, lr
             epoch = 10000
             batch_size = 1000
             for i in range(epoch):
                 if (i % 100 == 0):
                     lr = lr / 2
                 random_index = np.random.choice(len(X_train2), size=batch_size, replace=True)
                 X_mini_batch = X_train2[random_index]
                 y_mini_batch = y_train2[random_index]
                 pred = np.dot(X_mini_batch, w)
                 sgd(X_mini_batch, pred, y_mini_batch)
```

```
In [82]: train(trainFeat, trainYears)
```

```
In [83]: def predict(X_all):
             predict = np.dot(X_all, w)
             return predict
```

```
In [84]: musicMSE(predict(testFeat).reshape(testFeat.shape[0],).tolist(), testYears)

Out[84]: 90.59326129962352
```

## Part 3 - L1 Weight Decay

Test music MSE is 90.59, same as using L2.

```python
def sgd1(x, y_pred, y):
    global w, alpha, lr
    y = y.reshape((y.shape[0], 1))
    value = alpha * w / abs(w)
    value -= 2 * ((x * (y -y_pred)).sum(0)).reshape((91,1))
    w -= lr * value / x.shape[0]

def train1(X_train2, y_train2):
    global loss_value_previous, loss_value_current, w, lr
    epoch = 10000
    batch_size = 1000
    for i in range(epoch):
        if (i % 100 == 0):
            lr = lr / 2
        random_index = np.random.choice(len(X_train2), size=batch_size, replace=True)
        X_mini_batch = X_train2[random_index]
        y_mini_batch = y_train2[random_index]
        pred = np.dot(X_mini_batch, w)
        sgd1(X_mini_batch, pred, y_mini_batch)
```

In [95]:
```python
train1(trainFeat, trainYears)
```

In [96]:
```python
#L1 music MSE
musicMSE(predict(testFeat).reshape(testFeat.shape[0],).tolist(), testYears)
```

Out[96]:
```
90.59326129962352
```

# q2

February 19, 2019

```
In [35]: # Q2

In [ ]: import numpy as np
        from matplotlib import pyplot as plt

In [2]: X_train = 0
        y_train = 0
        X_test = 0
        y_test = 0

In [3]: f = open("./iris-train.txt", "r")
        content = f.readlines()
        f.close()

        x1_list_train = []
        x2_list_train = []
        X_list_train = []
        y_list_train = []

        for line in content:
            if line.strip() != "":
                temp = line.split()
                y = int(temp[0])
                x1 = float(temp[1])
                x2 = float(temp[2])
                y_list_train.append(y)
                X_list_train.append([x1, x2])

        for name in X_list_train:
            x1_list_train.append(name[0])
            x2_list_train.append(name[1])

In [4]: f = open("./iris-test.txt", "r")
        content = f.readlines()
        f.close()

        x1_list_test = []
        x2_list_test = []
```

```
X_list_test = []
y_list_test = []

for line in content:
    if line.strip() != "":
        temp = line.split()
        y = int(temp[0])
        x1 = float(temp[1])
        x2 = float(temp[2])
        y_list_test.append(y)
        X_list_test.append([x1, x2])

for name in X_list_test:
    x1_list_test.append(name[0])
    x2_list_test.append(name[1])
```

In [5]:
```
def norm(base_list, list_number):
    average = sum(base_list)/len(base_list)
    ret = []
    for number in list_number:
#         ret.append(number-average)
        ret.append((number-average)/(max(base_list)-min(base_list))*2)
    return ret
```

In [6]:
```
x1_list_train_copy = norm(x1_list_train, x1_list_train)
x2_list_train_copy = norm(x2_list_train, x2_list_train)
x1_list_test_copy = norm(x1_list_train, x1_list_test)
x2_list_test_copy = norm(x2_list_train, x2_list_test)
```

In [7]:
```
temp_x_train = []
for i in range(len(x1_list_train_copy)):
    temp_x_train.append([x1_list_train_copy[i], x2_list_train_copy[i], 1.0])

temp_x_test = []
for i in range(len(x1_list_test_copy)):
    temp_x_test.append([x1_list_test_copy[i], x2_list_test_copy[i], 1.0])
```

In [8]:
```
X_train = np.array(temp_x_train)
y_train = X = np.array(y_list_train)
X_test = np.array(temp_x_test)
y_test = np.array(y_list_test)
X_train_num = X_train.shape[0]
y_train_num = X_train.shape[0]
X_test_num = X_test.shape[0]
y_test_num = y_test.shape[0]
```

In [9]:
```
# above is for setup
```

In [130]:
```
w1 = np.random.random((1, 3))
w2 = np.random.random((1, 3))
```

```
w3 = np.random.random((1, 3))
v1 = 0
v2 = 0
v3 = 0
```

In [131]:
```
alpha = 0.05
r = 0.00005
lamb = 0

def weight_decay(w):
    return r*0.5*(np.dot(w,w.transpose()))

def soft_max(x_one, w):
#     return np.exp(np.dot(w, x_one))/(np.exp(np.dot(w1, x_one))+np.exp(np.dot(w2, x
    return np.exp(np.dot(x_one, w.transpose()))/(np.exp(np.dot(x_one, w1.transpose()

def one_pass_w1(x_one, y_one):
    global w1, v1
    is_w = y_one.copy()
    for i in range(is_w.shape[0]):
        if is_w[i] != 1:
            is_w[i] = 0
    is_w = is_w.reshape(is_w.shape[0], 1)
    derivatives_list = x_one*(is_w-soft_max(x_one, w1))
    derivatives = np.mean(derivatives_list, axis=0) # should be negative
    v1 = lamb*v1-alpha*derivatives
    w1 = w1-v1

def one_pass_w2(x_one, y_one):
    global w2, v2
    is_w = y_one.copy()
    for i in range(is_w.shape[0]):
        if is_w[i] != 2:
            is_w[i] = 0
    is_w = is_w.reshape(is_w.shape[0], 1)
    derivatives_list = x_one*(is_w-soft_max(x_one, w2))
    derivatives = np.mean(derivatives_list, axis=0) # should be negative
    v2 = lamb*v2-alpha*derivatives
    w2 = w2-v2

def one_pass_w3(x_one, y_one):
    global w3, v3
    is_w = y_one.copy()
    for i in range(is_w.shape[0]):
        if is_w[i] != 3:
            is_w[i] = 0
    is_w = is_w.reshape(is_w.shape[0], 1)
    derivatives_list = x_one*(is_w-soft_max(x_one, w3))
```

```
            derivatives = np.mean(derivatives_list, axis=0) # should be negative
            v3 = lamb*v3-alpha*derivatives
            w3 = w3-v3
```

```
In [132]: def run_pass(X, y):
              one_pass_w1(X, y)
              one_pass_w2(X, y)
              one_pass_w3(X, y)
```

```
In [133]: def predict(X_one):
              p1 = soft_max(X_one, w1)
              p2 = soft_max(X_one, w2)
              p3 = soft_max(X_one, w3)
              if (p1 >= p2 and p1 >= p3):
                  return 1
              if (p2 >= p1 and p2 >= p3):
                  return 2
              if (p3 >= p1 and p3 >= p2):
                  return 3

          def predict_all_score(X_all, Y_all):
              count = 0.0
              for i in range(X_all.shape[0]):
                  if (predict(X_all[i])) == Y_all[i]:
                      count = count + 1
              return count/X_all.shape[0]
```

```
In [134]: loss_train_x = []
          cross_loss_train_x = []
          loss_train_y = []
          loss_test_x = []
          cross_loss_test_x = []
          loss_test_y = []
```

```
In [135]: def cross_loss(X_all, y_all):
              loss = 0.0
              for i in range(X_all.shape[0]):
                  x_one = X_all[i]
                  current = np.asarray([soft_max(x_one, w1), soft_max(x_one, w2), soft_max(x_o
                  loss = loss + abs(current-y_all).sum()
              return np.log(loss/(X_all.shape[0]))
```

```
In [136]: def train(X_train, y_train):
              times = 1000
              batch_number = 20
              for i in range(times):
                  if (i%100 == 0):
                      print(cross_loss(X_train, y_train))
```

```
                loss_train_x.append(predict_all_score(X_train, y_train))
                cross_loss_train_x.append(cross_loss(X_train, y_train))
                loss_train_y.append(i)

                loss_test_x.append(predict_all_score(X_test, y_test))
                cross_loss_test_x.append(cross_loss(X_train, y_train))
                loss_test_y.append(i)
            batch_size = int(len(X_train)/batch_number)
            for i in range(batch_number):
                run_pass(X_train[i*batch_size:i*batch_size+batch_size], y_train[i*batch_s
```

In [137]:
```
train(X_train, y_train)
```

```
6.109226819165089
6.093510357104801
6.056933338973956
6.016329591168826
5.982721858130556
5.959801510446369
5.944183875639426
5.9334013455228956
5.922379382834747
5.911800123084676
```

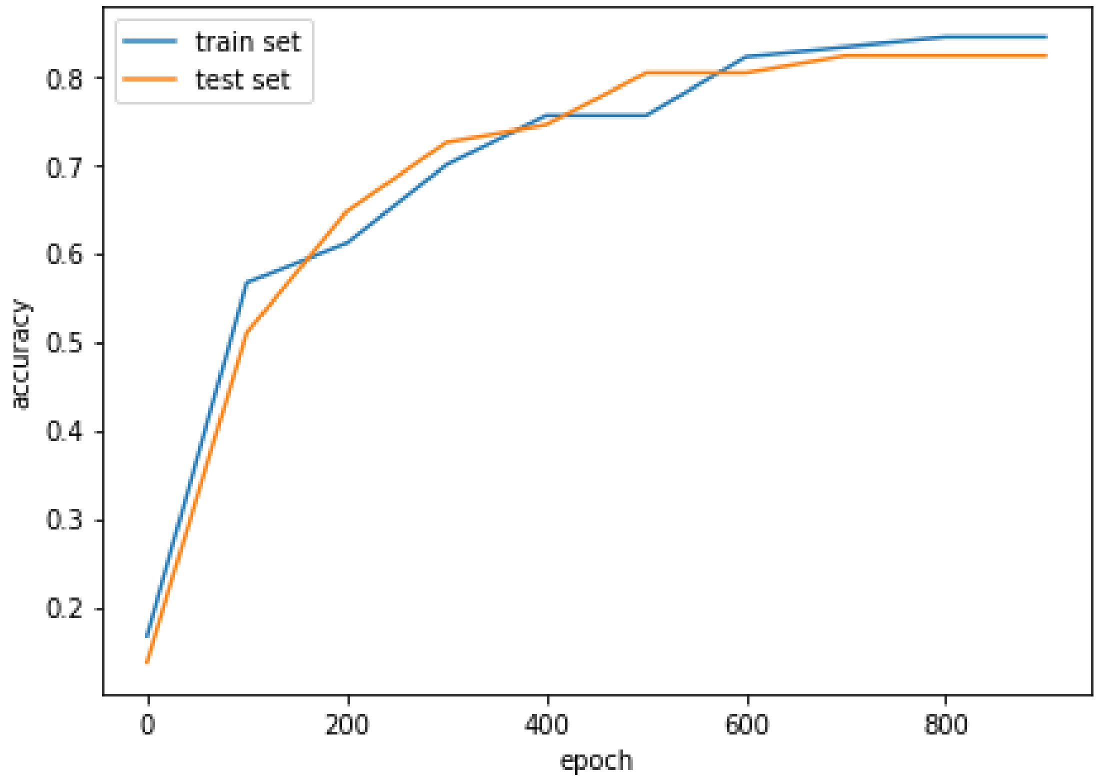
In [138]:
```
predict_all_score(X_train, y_train)
```

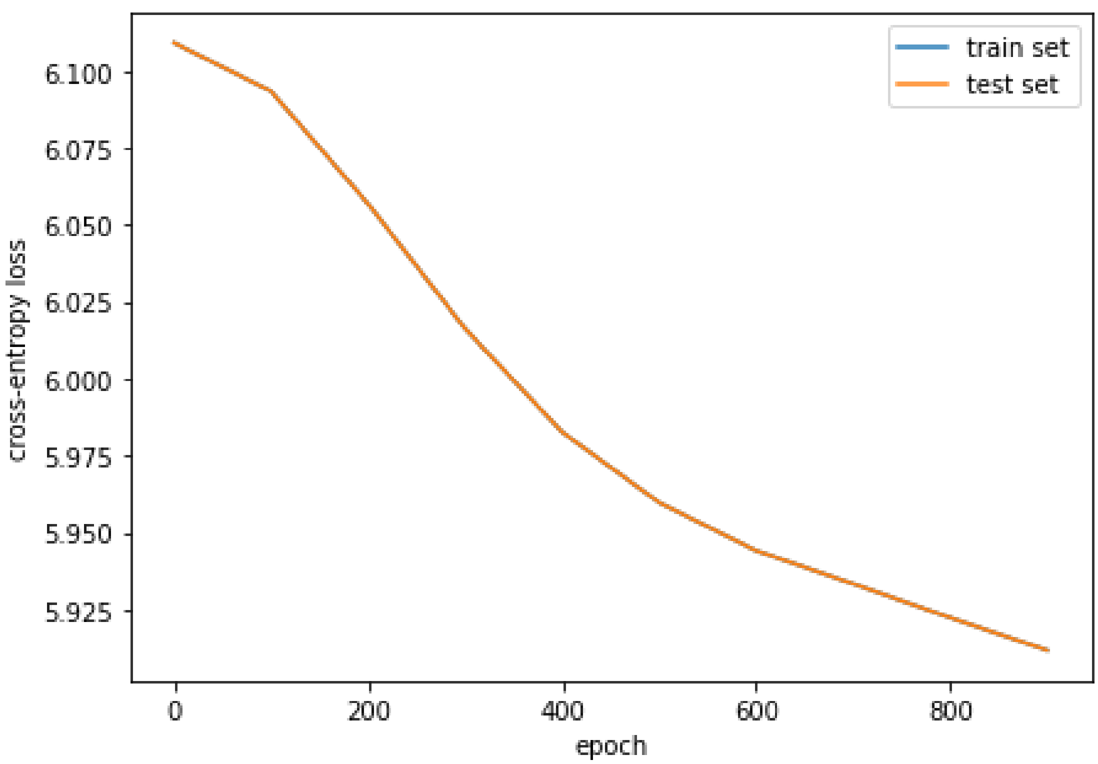
Out[138]: 0.8111111111111111

In [139]:
```
predict_all_score(X_test, y_test)
```

Out[139]: 0.8235294117647058

In [142]:
```
plt.subplots(figsize=(7,5))
plt.plot(loss_train_y, loss_train_x, label="train set")
plt.plot(loss_test_y, loss_test_x, label="test set")
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend()
plt.show()

plt.subplots(figsize=(7,5))
plt.plot(loss_train_y, cross_loss_train_x, label="train set")
plt.plot(loss_test_y, cross_loss_test_x, label="test set")
plt.ylabel('cross-entropy loss')
plt.xlabel('epoch')
plt.legend()
plt.show()
```


train set
test set
0.8
0.7
0.6
0.5
0.4
0.3
0.2
accuracy
0
200
400
600
800
epoch


train set
test set
6.100
6.075
6.050
6.025
6.000
5.975
5.950
5.925
cross-entropy loss
0
200
400
600
800
epoch

# q34

February 19, 2019

```
In [1]: # Q3 & Q4
```

```
In [ ]: %matplotlib inline

        import pickle
        import numpy as np
        import cv2
        from matplotlib import pyplot as plt
        from PIL import Image
```

```
In [2]: NUM = 10000
```

```
In [3]: def unpickle(file):
            with open(file, 'rb') as fo:
                dict = pickle.load(fo, encoding='bytes')
            return dict
```

```
In [4]: '''
        b'batch_label'
        21
        b'labels'
        10000
        b'data'
        10000
        b'filenames'
        10000
        '''
```

```
Out[4]: "\nb'batch_label'\n21\nb'labels'\n10000\nb'data'\n10000\nb'filenames'\n10000\n"
```

```
In [5]: def loadImageToRgbList(path):
            image_list_all = unpickle(path)[b'data']
            ret = []
            for i in range(NUM):
                image_list = image_list_all[i]
                r = image_list[:32*32]
                g = image_list[32*32:32*32*2]
                b = image_list[32*32*2:]
```

```
            image = []
            for ii in range(32*32):
                image.append([r[ii], g[ii], b[ii]])
            ret.append(image)
        return ret
```

In [6]:
```
def loadLabelToArray(path):
    return unpickle(path)[b'labels'][:NUM]
```

In [7]:
```
trainFeat = np.asarray(loadImageToRgbList("./cifar-10-batches-py/data_batch_1"))
trainLabels = np.asarray(loadLabelToArray("./cifar-10-batches-py/data_batch_1"))

trainFeat2 = np.asarray(loadImageToRgbList("./cifar-10-batches-py/data_batch_2"))
trainLabels2 = np.asarray(loadLabelToArray("./cifar-10-batches-py/data_batch_2"))

trainFeat3 = np.asarray(loadImageToRgbList("./cifar-10-batches-py/data_batch_3"))
trainLabels3 = np.asarray(loadLabelToArray("./cifar-10-batches-py/data_batch_3"))

trainFeat4 = np.asarray(loadImageToRgbList("./cifar-10-batches-py/data_batch_4"))
trainLabels4 = np.asarray(loadLabelToArray("./cifar-10-batches-py/data_batch_4"))

trainFeat5 = np.asarray(loadImageToRgbList("./cifar-10-batches-py/data_batch_5"))
trainLabels5 = np.asarray(loadLabelToArray("./cifar-10-batches-py/data_batch_5"))

testFeat = np.asarray(loadImageToRgbList("./cifar-10-batches-py/test_batch"))
testLabels = np.asarray(loadLabelToArray("./cifar-10-batches-py/test_batch"))
```

In [8]:
```
trainFeat = trainFeat.reshape(NUM, 32, 32, 3)
testFeat = testFeat.reshape(NUM, 32, 32, 3)
```

In [9]:
```
def find_3_image(label):
    count = 3
    ret = []
    for i in range(NUM):
        if (count == 0):
            break
        if (trainLabels[i] == label):
            ret.append(trainFeat[i])
            count = count - 1
    return ret
```

In [10]:
```
rows = 3
cols = 10
fig, axes = plt.subplots(rows, cols, figsize=(12,6))

image_to_show = []
for label in range(10):
    images = find_3_image(label)
    image_to_show += images
```

```
index = -1
def getImage():
    global index
    index = index + 1
    return image_to_show[index]

for i in range(rows*cols):
    row_index = i//cols
    col_index = i%cols
    ax = axes[row_index, col_index]
    img = getImage()
    img = Image.fromarray(img, 'RGB')
    ax.imshow(img)
```

```
In [11]: trainFeat = trainFeat.reshape(NUM, 32*32*3)
         trainFeat2 = trainFeat2.reshape(NUM, 32*32*3)
         trainFeat3 = trainFeat3.reshape(NUM, 32*32*3)
         trainFeat4 = trainFeat4.reshape(NUM, 32*32*3)
         trainFeat5 = trainFeat5.reshape(NUM, 32*32*3)

         testFeat = testFeat.reshape(NUM, 32*32*3)
```

```
In [12]: mean = lambda x: x/10000
         vfunc = np.vectorize(mean)

         trainFeat = vfunc(trainFeat)
         trainFeat2 = vfunc(trainFeat2)
         trainFeat3 = vfunc(trainFeat3)
         trainFeat4 = vfunc(trainFeat4)
```

```
trainFeat5 = vfunc(trainFeat5)

testFeat = vfunc(testFeat)
```

```
In [13]: X_train = np.stack((trainFeat, trainFeat2, trainFeat3, trainFeat4, trainFeat5))
         y_train = np.stack((trainLabels, trainLabels2, trainLabels3, trainLabels4, trainLabel
```

```
In [14]: X_train = X_train.reshape(NUM*5, 3072)
         y_train = y_train.reshape(NUM*5)
```

```
In [15]: # above is for load data
```

```
In [16]: w0 = np.random.random((1, 32*32*3))
         w1 = np.random.random((1, 32*32*3))
         w2 = np.random.random((1, 32*32*3))
         w3 = np.random.random((1, 32*32*3))
         w4 = np.random.random((1, 32*32*3))
         w5 = np.random.random((1, 32*32*3))
         w6 = np.random.random((1, 32*32*3))
         w7 = np.random.random((1, 32*32*3))
         w8 = np.random.random((1, 32*32*3))
         w9 = np.random.random((1, 32*32*3))
         v0 = 0
         v1 = 0
         v2 = 0
         v3 = 0
         v4 = 0
         v5 = 0
         v6 = 0
         v7 = 0
         v8 = 0
         v9 = 0
```

```
In [17]: alpha = 10 # due to a 1/10000 re-scale in input, otherwise learning rate should be 0.
         r = 0.000001
         lamb = 0.0001

         def weight_decay(w):
             return r*0.5*(np.dot(w,w.transpose()))

         def soft_max(x_one, w):
             return np.exp(np.dot(x_one, w.transpose())) / (
             np.exp(np.dot(x_one, w0.transpose())) +
             np.exp(np.dot(x_one, w1.transpose())) +
             np.exp(np.dot(x_one, w2.transpose())) +
             np.exp(np.dot(x_one, w3.transpose())) +
             np.exp(np.dot(x_one, w4.transpose())) +
             np.exp(np.dot(x_one, w5.transpose())) +
             np.exp(np.dot(x_one, w6.transpose())) +
```

```
            np.exp(np.dot(x_one, w7.transpose())) +
            np.exp(np.dot(x_one, w8.transpose())) +
            np.exp(np.dot(x_one, w9.transpose()))) + weight_decay(w)
```

```
In [18]: loss_train_x = []
         cross_loss_train_x = []
         loss_train_y = []
         loss_test_x = []
         cross_loss_test_x = []
         loss_test_y = []
```

```
In [19]: def one_pass_w0(x_one, y_one):
             global w0, v0
             is_w = y_one.copy()
             for i in range(is_w.shape[0]):
                 if is_w[i]!=0:
                     is_w[i] = 0
                 else:
                     is_w[i] = 1
             is_w = is_w.reshape(is_w.shape[0], 1)
             derivatives_list = x_one*(is_w-soft_max(x_one, w0))
             derivatives = np.mean(derivatives_list, axis=0) # should be negative
             v0 = lamb*v0-alpha*derivatives
             w0 = w0-v0
         #     print(w0)

         def one_pass_w1(x_one, y_one):
             global w1, v1
             is_w = y_one.copy()
             for i in range(is_w.shape[0]):
                 if is_w[i]!=1:
                     is_w[i] = 0
                 else:
                     is_w[i] = 1
             is_w = is_w.reshape(is_w.shape[0], 1)
             derivatives_list = x_one*(is_w-soft_max(x_one, w1))
             derivatives = np.mean(derivatives_list, axis=0) # should be negative
             v1 = lamb*v1-alpha*derivatives
             w1 = w1-v1

         def one_pass_w2(x_one, y_one):
             global w2, v2
             is_w = y_one.copy()
             for i in range(is_w.shape[0]):
                 if is_w[i]!=2:
                     is_w[i] = 0
                 else:
                     is_w[i] = 1
```

```
    is_w = is_w.reshape(is_w.shape[0], 1)
    derivatives_list = x_one*(is_w-soft_max(x_one, w2))
    derivatives = np.mean(derivatives_list, axis=0) # should be negative
    v2 = lamb*v2-alpha*derivatives
    w2 = w2-v2

def one_pass_w3(x_one, y_one):
    global w3, v3
    is_w = y_one.copy()
    for i in range(is_w.shape[0]):
        if is_w[i]!=3:
            is_w[i] = 0
        else:
            is_w[i] = 1
    is_w = is_w.reshape(is_w.shape[0], 1)
    derivatives_list = x_one*(is_w-soft_max(x_one, w3))
    derivatives = np.mean(derivatives_list, axis=0) # should be negative
    v3 = lamb*v3-alpha*derivatives
    w3 = w3-v3

def one_pass_w4(x_one, y_one):
    global w4, v4
    is_w = y_one.copy()
    for i in range(is_w.shape[0]):
        if is_w[i]!=4:
            is_w[i] = 0
        else:
            is_w[i] = 1
    is_w = is_w.reshape(is_w.shape[0], 1)
    derivatives_list = x_one*(is_w-soft_max(x_one, w4))
    derivatives = np.mean(derivatives_list, axis=0) # should be negative
    v4 = lamb*v4-alpha*derivatives
    w4 = w4-v4

def one_pass_w5(x_one, y_one):
    global w5, v5
    is_w = y_one.copy()
    for i in range(is_w.shape[0]):
        if is_w[i]!=5:
            is_w[i] = 0
        else:
            is_w[i] = 1
    is_w = is_w.reshape(is_w.shape[0], 1)
    derivatives_list = x_one*(is_w-soft_max(x_one, w5))
    derivatives = np.mean(derivatives_list, axis=0) # should be negative
    v5 = lamb*v5-alpha*derivatives
    w5 = w5-v5
```

```
def one_pass_w6(x_one, y_one):
    global w6, v6
    is_w = y_one.copy()
    for i in range(is_w.shape[0]):
        if is_w[i]!=6:
            is_w[i] = 0
        else:
            is_w[i] = 1
    is_w = is_w.reshape(is_w.shape[0], 1)
    derivatives_list = x_one*(is_w-soft_max(x_one, w6))
    derivatives = np.mean(derivatives_list, axis=0) # should be negative
    v6 = lamb*v6-alpha*derivatives
    w6 = w6-v6

def one_pass_w7(x_one, y_one):
    global w7, v7
    is_w = y_one.copy()
    for i in range(is_w.shape[0]):
        if is_w[i]!=7:
            is_w[i] = 0
        else:
            is_w[i] = 1
    is_w = is_w.reshape(is_w.shape[0], 1)
    derivatives_list = x_one*(is_w-soft_max(x_one, w7))
    derivatives = np.mean(derivatives_list, axis=0) # should be negative
    v7 = lamb*v7-alpha*derivatives
    w7 = w7-v7

def one_pass_w8(x_one, y_one):
    global w8, v8
    is_w = y_one.copy()
    for i in range(is_w.shape[0]):
        if is_w[i]!=8:
            is_w[i] = 0
        else:
            is_w[i] = 1
    is_w = is_w.reshape(is_w.shape[0], 1)
    derivatives_list = x_one*(is_w-soft_max(x_one, w8))
    derivatives = np.mean(derivatives_list, axis=0) # should be negative
    v8 = lamb*v8-alpha*derivatives
    w8 = w8-v8

def one_pass_w9(x_one, y_one):
    global w9, v9
    is_w = y_one.copy()
    for i in range(is_w.shape[0]):
        if is_w[i]!=9:
            is_w[i] = 0
```

```
            else:
                is_w[i] = 1
        is_w = is_w.reshape(is_w.shape[0], 1)
        derivatives_list = x_one*(is_w-soft_max(x_one, w9))
        derivatives = np.mean(derivatives_list, axis=0) # should be negative
        v9 = lamb*v9-alpha*derivatives
        w9 = w9-v9
```

In [20]:
```
def run_pass(X, y):
    one_pass_w0(X, y)
    one_pass_w1(X, y)
    one_pass_w2(X, y)
    one_pass_w3(X, y)
    one_pass_w4(X, y)
    one_pass_w5(X, y)
    one_pass_w6(X, y)
    one_pass_w7(X, y)
    one_pass_w8(X, y)
    one_pass_w9(X, y)
```

In [21]:
```
def predict(X_one):
    p0 = soft_max(X_one, w0)
    p1 = soft_max(X_one, w1)
    p2 = soft_max(X_one, w2)
    p3 = soft_max(X_one, w3)
    p4 = soft_max(X_one, w4)
    p5 = soft_max(X_one, w5)
    p6 = soft_max(X_one, w6)
    p7 = soft_max(X_one, w7)
    p8 = soft_max(X_one, w8)
    p9 = soft_max(X_one, w9)

    results = [p0, p1, p2, p3, p4, p5, p6, p7, p8, p9]
    max_p = max(results)
    ret = results.index(max_p)
    return ret

def predict_all_score(X_all, Y_all):
    count = 0.0
    for i in range(X_all.shape[0]):
        if (predict(X_all[i])) == Y_all[i]:
            count = count + 1
    return count/X_all.shape[0]
```

In [22]:
```
def cross_loss(X_all, y_all):
    current = np.asarray([soft_max(X_all, w0), soft_max(X_all, w1), soft_max(X_all, w
                         soft_max(X_all, w4), soft_max(X_all, w5), soft_max(X_all
                          soft_max(X_all, w8), soft_max(X_all, w9)
```

```
                                ])
        loss = abs(current-y_all).sum()
        return np.log(loss/(X_all.shape[0]))
```

In [23]:
```
def train(X_train2, y_train2):
    epoch = 500
    batch_size = 500
    for i in range(epoch):
        print("running " + str(i))
        random_index = np.random.choice(len(X_train2), size=batch_size, replace=True)
        X_mini_batch = X_train2[random_index]
        y_mini_batch = y_train2[random_index]
        run_pass(X_mini_batch, y_mini_batch)


        if (i%100 == 0):
            loss_train_x.append(predict_all_score(X_mini_batch, y_mini_batch))
            cross_loss_train_x.append(cross_loss(X_mini_batch, y_mini_batch))
            loss_train_y.append(i)

            loss_test_x.append(predict_all_score(testFeat, testLabels))
            cross_loss_test_x.append(cross_loss(testFeat, testLabels))
            loss_test_y.append(i)
```

In [24]:
```
train(X_train, y_train)
```

```
running 0
running 1
running 2
running 3
running 4
running 5
running 6
running 7
running 8
running 9
running 10
running 11
running 12
running 13
running 14
running 15
running 16
running 17
running 18
running 19
running 20
running 21
```

```
running 22
running 23
running 24
running 25
running 26
running 27
running 28
running 29
running 30
running 31
running 32
running 33
running 34
running 35
running 36
running 37
running 38
running 39
running 40
running 41
running 42
running 43
running 44
running 45
running 46
running 47
running 48
running 49
running 50
running 51
running 52
running 53
running 54
running 55
running 56
running 57
running 58
running 59
running 60
running 61
running 62
running 63
running 64
running 65
running 66
running 67
running 68
running 69
```

```
running 70
running 71
running 72
running 73
running 74
running 75
running 76
running 77
running 78
running 79
running 80
running 81
running 82
running 83
running 84
running 85
running 86
running 87
running 88
running 89
running 90
running 91
running 92
running 93
running 94
running 95
running 96
running 97
running 98
running 99
running 100
running 101
running 102
running 103
running 104
running 105
running 106
running 107
running 108
running 109
running 110
running 111
running 112
running 113
running 114
running 115
running 116
running 117
```

```
running 118
running 119
running 120
running 121
running 122
running 123
running 124
running 125
running 126
running 127
running 128
running 129
running 130
running 131
running 132
running 133
running 134
running 135
running 136
running 137
running 138
running 139
running 140
running 141
running 142
running 143
running 144
running 145
running 146
running 147
running 148
running 149
running 150
running 151
running 152
running 153
running 154
running 155
running 156
running 157
running 158
running 159
running 160
running 161
running 162
running 163
running 164
running 165
```

```
running 166
running 167
running 168
running 169
running 170
running 171
running 172
running 173
running 174
running 175
running 176
running 177
running 178
running 179
running 180
running 181
running 182
running 183
running 184
running 185
running 186
running 187
running 188
running 189
running 190
running 191
running 192
running 193
running 194
running 195
running 196
running 197
running 198
running 199
running 200
running 201
running 202
running 203
running 204
running 205
running 206
running 207
running 208
running 209
running 210
running 211
running 212
running 213
```

```
running 214
running 215
running 216
running 217
running 218
running 219
running 220
running 221
running 222
running 223
running 224
running 225
running 226
running 227
running 228
running 229
running 230
running 231
running 232
running 233
running 234
running 235
running 236
running 237
running 238
running 239
running 240
running 241
running 242
running 243
running 244
running 245
running 246
running 247
running 248
running 249
running 250
running 251
running 252
running 253
running 254
running 255
running 256
running 257
running 258
running 259
running 260
running 261
```

```
running 262
running 263
running 264
running 265
running 266
running 267
running 268
running 269
running 270
running 271
running 272
running 273
running 274
running 275
running 276
running 277
running 278
running 279
running 280
running 281
running 282
running 283
running 284
running 285
running 286
running 287
running 288
running 289
running 290
running 291
running 292
running 293
running 294
running 295
running 296
running 297
running 298
running 299
running 300
running 301
running 302
running 303
running 304
running 305
running 306
running 307
running 308
running 309
```

```
running 310
running 311
running 312
running 313
running 314
running 315
running 316
running 317
running 318
running 319
running 320
running 321
running 322
running 323
running 324
running 325
running 326
running 327
running 328
running 329
running 330
running 331
running 332
running 333
running 334
running 335
running 336
running 337
running 338
running 339
running 340
running 341
running 342
running 343
running 344
running 345
running 346
running 347
running 348
running 349
running 350
running 351
running 352
running 353
running 354
running 355
running 356
running 357
```

```
running 358
running 359
running 360
running 361
running 362
running 363
running 364
running 365
running 366
running 367
running 368
running 369
running 370
running 371
running 372
running 373
running 374
running 375
running 376
running 377
running 378
running 379
running 380
running 381
running 382
running 383
running 384
running 385
running 386
running 387
running 388
running 389
running 390
running 391
running 392
running 393
running 394
running 395
running 396
running 397
running 398
running 399
running 400
running 401
running 402
running 403
running 404
running 405
```

```
running 406
running 407
running 408
running 409
running 410
running 411
running 412
running 413
running 414
running 415
running 416
running 417
running 418
running 419
running 420
running 421
running 422
running 423
running 424
running 425
running 426
running 427
running 428
running 429
running 430
running 431
running 432
running 433
running 434
running 435
running 436
running 437
running 438
running 439
running 440
running 441
running 442
running 443
running 444
running 445
running 446
running 447
running 448
running 449
running 450
running 451
running 452
running 453
```

```
running 454
running 455
running 456
running 457
running 458
running 459
running 460
running 461
running 462
running 463
running 464
running 465
running 466
running 467
running 468
running 469
running 470
running 471
running 472
running 473
running 474
running 475
running 476
running 477
running 478
running 479
running 480
running 481
running 482
running 483
running 484
running 485
running 486
running 487
running 488
running 489
running 490
running 491
running 492
running 493
running 494
running 495
running 496
running 497
running 498
running 499
```

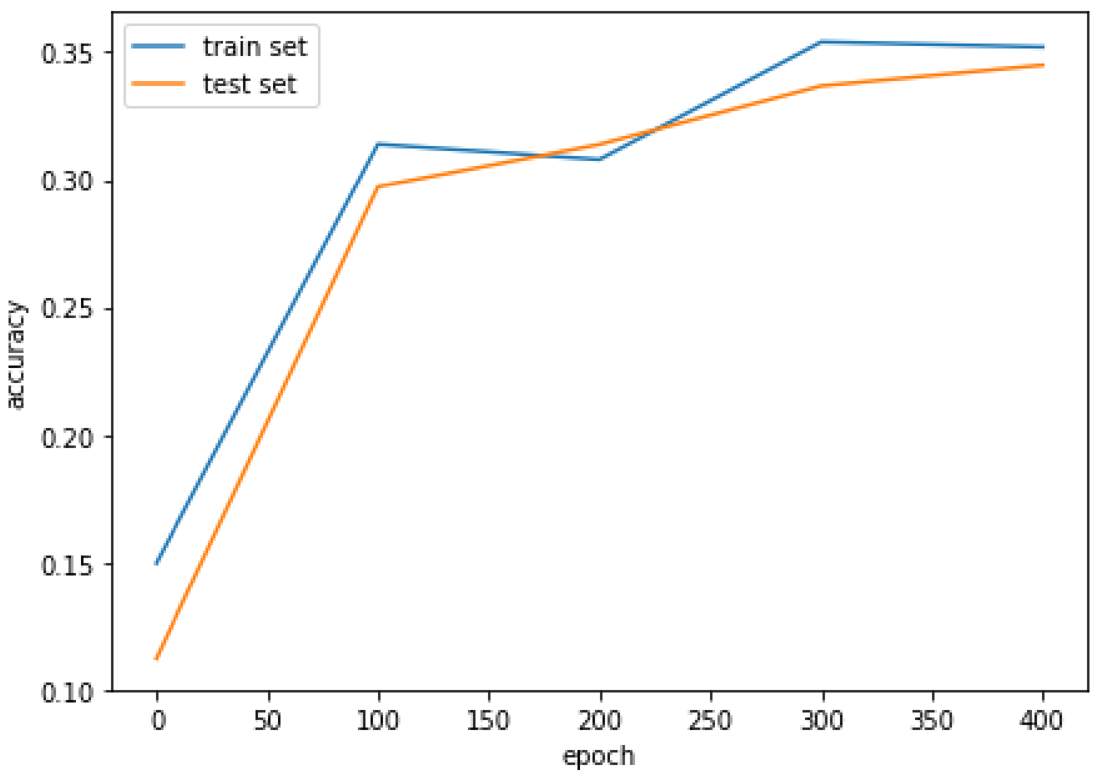
In [25]: 
```
predict_all_score(trainFeat, trainLabels)
```

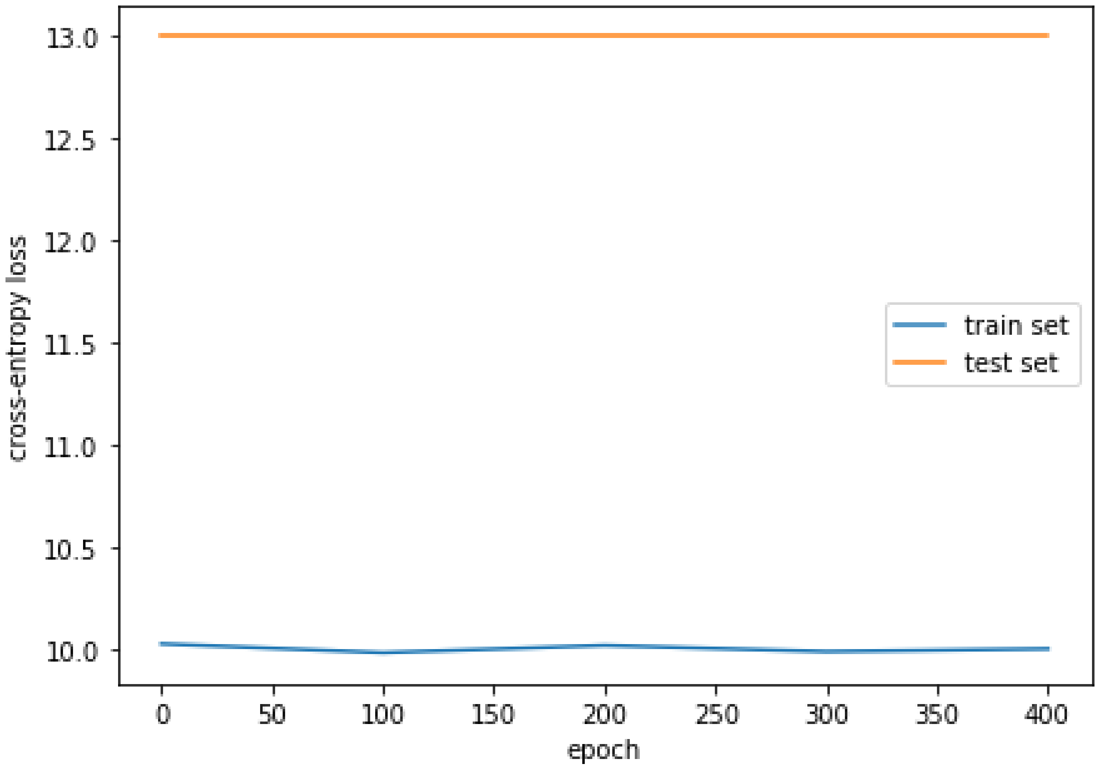
Out[25]: 0.3546

In [26]: 
```
predict_all_score(testFeat, testLabels)
```

Out[26]: 0.35

In [27]: 
```
plt.subplots(figsize=(7,5))
plt.plot(loss_train_y, loss_train_x, label="train set")
plt.plot(loss_test_y, loss_test_x, label="test set")
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend()
plt.show()

plt.subplots(figsize=(7,5))
plt.plot(loss_train_y, cross_loss_train_x, label="train set")
plt.plot(loss_test_y, cross_loss_test_x, label="test set")
plt.ylabel('cross-entropy loss')
plt.xlabel('epoch')
plt.legend()
plt.show()
```





13.0
12.5
12.0
11.5
11.0
10.5
10.0
cross-entropy loss
0
50
100
150
200
250
300
350
400
epoch
train set
test set

# q5

February 19, 2019

In [18]:
```
# Q5
```

In [ ]:
```
import numpy as np
from sklearn.preprocessing import normalize
```

In [19]:
```
f = open("./YearPredictionMSD.txt", "r")
content = f.readlines()
f.close()
```

In [20]:
```
train_features_list = []
train_years_list = []

START = 463714
END = len(content)
# START = 100
# END = 200

for line in content[:START]:
    items = line.split(",")
    train_years_list.append(int(items[0]))
    temp_list = list(map(float, items[1:]))
    train_features_list.append(temp_list)
```

In [21]:
```
test_features_list = []
test_years_list = []

for line in content[START:END]:
    items = line.split(",")
    test_years_list.append(int(items[0]))
    temp_list = list(map(float, items[1:]))
    test_features_list.append(temp_list)
```

In [22]:
```
trainFeat = np.asarray(train_features_list)
trainYears = np.asarray(train_years_list)
testFeat = np.asarray(test_features_list)
testYears = np.asarray(test_years_list)
```

In [23]:
```
def musicMSE(pred, gt):
    pred = np.round(pred)
```

```
        loss = (pred - gt) ** 2
        return loss.sum() / len(gt)
```

In [24]:
```
# range of the years
min_year = min(train_years_list)
max_year = max(train_years_list)
year_times_map = {}

for year in train_years_list:
    if year in year_times_map:
        year_times_map[year] += 1
    else:
        year_times_map[year] = 1

most_common_key = max(year_times_map, key=year_times_map.get)

print("Min year is "+str(min_year))
print("Max year is "+str(max_year))
print("All years are "+str(year_times_map.keys()))
```

```
Min year is 1922
Max year is 2011
All years are dict_keys([2001, 2007, 2008, 2002, 2004, 2003, 1999, 1992, 1997, 1987, 2000, 2005
```

In [25]:
```
mse_mean = musicMSE(train_years_list, [1998]*len(train_years_list))
print("MSE that always outputs 1998 (mean) is "+str(mse_mean))

mse_most_common = musicMSE(train_years_list, [most_common_key]*len(train_years_list))
print("MSE that always outputs the most common key "+str(most_common_key)+ " is "+str
```

```
MSE that always outputs 1998 (mean) is 119.82739576549339
MSE that always outputs the most common key 2007 is 193.87802179791854
```

In [26]:
```
# normalize data
# trainFeat = normalize(trainFeat, axis=1, norm='l1')
# testFeat = normalize(testFeat, axis=1, norm='l1')

trainFeat_temp = trainFeat - trainFeat.mean(axis=0)
trainFeat_temp = trainFeat_temp / np.std(trainFeat, axis=0)

testFeat_temp = testFeat - trainFeat.mean(axis=0)
testFeat_temp = testFeat_temp / np.std(trainFeat, axis=0)

trainFeat = np.append(trainFeat_temp,np.ones([len(trainFeat_temp),1]),1)
testFeat = np.append(testFeat_temp,np.ones([len(testFeat_temp),1]),1)
```

In [61]:
```
trainYearAverage = trainYears.mean()
```

In [62]:
```
trainYears = trainYears - trainYearAverage
testYears = testYears - trainYearAverage
```

In [63]:
```
print(trainFeat.shape)
print(testFeat.shape)
```

```
(463714, 91)
(51631, 91)
```

In [67]:
```
trainYears
```

Out[67]:
```
array([2.61392367, 2.61392367, 2.61392367, ..., 7.61392367, 8.61392367,
       7.61392367])
```

In [29]:
```
# above is for loading data
```

In [79]:
```
w = np.random.random((trainFeat.shape[1], 1))
lr = 0.1
alpha = 0.005
```

In [94]:
```
# sum(i-N) xi*2(wx-y) + 2*a*w
# def derv(w, x, y):
#     t1 = np.dot(x, w)
#     print(t1)
#     t15 = t1 - y.reshape((y.shape[0], 1)) # diff (90,1)
#     t2 = x * t15
#     t3 = 2*np.mean(t2, axis=0)
#     t3 = t3.reshape(((x.shape[1], 1)))
#     t4 = 2*a*w
#     return t3 + t4

# def one_pass(X_all, y_all):
#     global w
#     d = derv(w, X_all, y_all)
#     w = w - alpha*d

def sgd(x, y_pred, y):
    global w, alpha
    n = x.shape[0]
    y = y.reshape((y.shape[0], 1))
    value = alpha * w
    value -= 2 * (x * (y - y_pred)).sum(0).reshape((91, 1))
    w -= lr * value/n

def sgd1(x, y_pred, y):
    global w, alpha, lr
    y = y.reshape((y.shape[0], 1))
    value = alpha * w / abs(w)
```

```
        value -= 2 * ((x * (y -y_pred)).sum(0)).reshape((91,1))
        w -= lr * value / x.shape[0]

def train1(X_train2, y_train2):
    global loss_value_previous, loss_value_current, w, lr
    epoch = 10000
    batch_size = 1000
    for i in range(epoch):
        if (i % 100 == 0):
            lr = lr / 2
        random_index = np.random.choice(len(X_train2), size=batch_size, replace=True)
        X_mini_batch = X_train2[random_index]
        y_mini_batch = y_train2[random_index]
        pred = np.dot(X_mini_batch, w)
        sgd1(X_mini_batch, pred, y_mini_batch)

def loss(w, x, y):
    global alpha
    t1 = np.dot(x,w)-y.reshape((y.shape[0], 1))
    t2 = np.linalg.norm(t1)
    t3 = alpha*(np.linalg.norm(w))
    return t2+t3
```

In [81]:
```
def train(X_train2, y_train2):
    global loss_value_previous, loss_value_current, w, lr
    epoch = 10000
    batch_size = 1000
    for i in range(epoch):
        if (i % 100 == 0):
            lr = lr / 2
        random_index = np.random.choice(len(X_train2), size=batch_size, replace=True)
        X_mini_batch = X_train2[random_index]
        y_mini_batch = y_train2[random_index]
        pred = np.dot(X_mini_batch, w)
        sgd(X_mini_batch, pred, y_mini_batch)
```

In [82]:
```
train(trainFeat, trainYears)
```

In [83]:
```
def predict(X_all):
    predict = np.dot(X_all, w)
    return predict
```

In [84]:
```
musicMSE(predict(testFeat).reshape(testFeat.shape[0],).tolist(), testYears)
```

Out[84]:
```
90.59326129962352
```

In [95]:
```
train1(trainFeat, trainYears)
```

In [96]:
```
#L1 music MSE
musicMSE(predict(testFeat).reshape(testFeat.shape[0],).tolist(), testYears)
```

Out[96]: 90.59326129962352

In [ ]:
```
# for i in range(trainFeat.shape[1]):
#     print("max value is "+str(np.max(trainFeat[:,i])))
#     print("min value is "+str(np.min(trainFeat[:,i])))
```