```
import scipy.io as io
import scipy.special as ss
import numpy as np
import sklearn.preprocessing
import sklearn.model_selection
import matplotlib.pyplot as plt

file_handle = io.loadmat('imagenet_data.mat')
# features and labels
features = file_handle['features']
labels = file_handle['labels']
labels = np.reshape(labels, [1000,1])


# normalize all data points
column_sum = [0] * 2048
for features_ in features:
    column_sum += features_
norm_mean = column_sum/1000
norm_features = []
for norm_ in features:
    diff = norm_ - norm_mean
    row_norm = np.linalg.norm(diff)
    norm_features.append(diff/row_norm)

# superscript features
super_features = []
for i in range(len(features)):
    super_features.append(np.append(features[i], np.array([1]),))

# random select test and training samples
def softmax_loss_naive(W, X, y, reg):

    # Initialize the loss and gradient to zero.
    loss = 0.0
    dW = np.zeros_like(W)
    l_w = np.zeros_like(W)

    # Get shapes
    num_classes = W.shape[0]
    num_train = X.shape[1]

    for i in range(num_train):
        # Compute vector of scores
```

```
        f_i = W.dot(X[:, i]) # in R^{num_classes}

        # Normalization trick to avoid numerical instability, per
http://cs231n.github.io/linear-classify/#softmax
        log_c = np.max(f_i)
        f_i -= log_c

        # Compute loss (and add to it, divided later)
        # L_i = - f(x_i)_{y_i} + log \sum_j e^{f(x_i)_j}
        sum_i = 0.0
        for f_i_j in f_i:
            sum_i += np.exp(f_i_j)
        loss += -f_i[y[i]] + np.log(sum_i)

        # Compute gradient
        # dw_j = 1/num_train * \sum_i[x_i * (p(y_i = j)-Ind{y_i =
j} )]
        # Here we are computing the contribution to the inner sum for
a given i.
        for j in range(num_classes):
            p = np.exp(f_i[j])/sum_i
            dW[j, :] += (p-(j == y[i])) * X[:, i]
    # Compute average
    loss /= num_train
    dW /= num_train

    # Regularization
    loss += 0.5 * reg * np.linalg.norm(W, ord='fro')**2
    dW += reg*W
    return loss, dW

tot_error = 0
for in_ in range(10):
    # random select test and training samples
    X_train, X_test, y_train, y_test =
sklearn.model_selection.train_test_split(super_features, labels,
test_size=100)
    # initialize w
    w = np.ones((10, 2049))


    X_train = np.transpose(X_train)
    y_train = y_train.astype(int)
    iteration = 0
```

```
    while iteration < 500:
        l, dw = softmax_loss_naive(w, X_train, y_train, 0.1)
        w -= 0.01 * dw
        #print(0.01*dw)
        iteration += 1
    # Test
    X_test = np.transpose(X_test)
    z = np.matmul(w, X_test)
    sf = ss.softmax(np.transpose(z))
    #print(np.shape(sf))
    error = 0
    for i, sf_ in enumerate(sf):
        max_prob = max(sf_)
        for j in range(len(sf_)):
            if sf_[j] == max_prob:
                max_index = j
        #print(max_index, int(y_test[i]))

        if max_index != int(y_test[i]):
            error += 1
    tot_error += error
    print('trial{} error rate is {}'.format(in_, error/len(y_test)))
print('average error {}'.format(tot_error/10/len(y_test)))

### C
tot_iteration = 0
for i in range(1, 2):
    X_train, X_test, y_train, y_test = \
        sklearn.model_selection.train_test_split(super_features,
labels, test_size=100)
    # initialize w
    w = np.ones((10, 2049))
    X_train = np.transpose(X_train)
    y_train = y_train.astype(int)
    iteration = 0
    gradient_descent_lst = []
    while True:
        l, dw= softmax_loss_naive(w, X_train, y_train, 0.1)
        w -= 0.01 * dw
        fro_squared = np.linalg.norm(dw, ord='fro')**2
        #print(fro_squared/(1+abs(l)))
```

```
        if iteration <= 150:
            gradient_descent_lst.append(fro_squared/(1+abs(l)))
        if fro_squared/(1+abs(l)) <= 0.01:
            break
        iteration += 1

    tot_iteration += iteration
    #print('trail{}, iteration took {}'.format(i, iteration))

print('average iteration took {}'.format(tot_iteration/1))

### D
# heavy ball
tot_iteration = 0
for i in range(1, 2):
    beta = 0.01
    X_train, X_test, y_train, y_test = \
        sklearn.model_selection.train_test_split(super_features,
labels, test_size=100)
    iteration = 0
    w = np.ones((10, 2049))
    X_train = np.transpose(X_train)
    y_train = y_train.astype(int)
    heavy_ball_lst = []
    while True:
        l, dw = softmax_loss_naive(w, X_train, y_train, 0.1)
        if iteration >= 1:
            diff = dw - d_w
            d_w = dw
        else:
            d_w = dw
            diff = 0
        w -= 0.01 * dw - beta * diff
        fro_squared = np.linalg.norm(dw, ord='fro') ** 2
        #print(fro_squared / (1 + abs(l)))
        if iteration <= 150:
            heavy_ball_lst.append(fro_squared/(1+abs(l)))
        if fro_squared / (1 + abs(l)) <= 0.01:
            break
        iteration += 1
    tot_iteration += iteration
    # print(iteration)
print('average iteration took {}'.format(tot_iteration/1))
```

```
plt.figure()
t = list(range(1, 152))
plt.plot(t, heavy_ball_lst)
plt.plot(t, gradient_descent_lst)
plt.legend(('heavy ball', 'gradient_descent'))
plt.show()
```