```
import numpy as np
import os
import matplotlib.pyplot as plt
import pdb
import matplotlib as mpl

mpl.rcParams['figure.dpi'] = 300

def Relu(x, require_grad=True):
    mask = 1.0*(x>=0)
    val = mask*x
    if(require_grad):
         return val, mask
    else:
        return val

def gen_gt(x):
    '''
    x: input of shape (b,d)
    '''
    return np.sum(Relu(x, require_grad=False),axis=1)

def h(W,x, require_grad=True):
    '''
    W: weight matrix of shape (d,m)
    x: input of shape (b,d)
    '''
    d, m = W.shape
    b = x.shape[0]
    W_T = np.repeat(np.expand_dims(W.T,axis=0), b, axis=0) # b,m,d
    X = np.repeat(np.expand_dims(x,axis=1), m, axis=1) # b,m,d

    if require_grad:
        val, mask = Relu(np.sum(W_T*X, axis=2), require_grad=True)
        MASK = np.repeat(np.expand_dims(mask,2), d, axis=2)
        grad_h_W = MASK*X
        return np.sum(val, axis=1), grad_h_W
    else:
        val= Relu(np.sum(W_T*X, axis=2), require_grad=False)
        return np.sum(val, axis=1)


def SGD_step(W, X, Y, lr):
    '''
    W: (d,m)
    X: (b,d)
    Y: (b,)
    '''
    pred, grad_h_W = h(W, X, require_grad=True)
    diff = Y - pred
    loss_f = np.mean(diff**2)

    grad_W = np.mean(-2*np.expand_dims(diff, axis=(1,2))*grad_h_W, axis=0)
    W = W - lr*grad_W.T

    return W, loss_f

def plot_prop(x_label, datas, labels, prop_name, figname, X=None):
    fig = plt.figure(figsize=(10,7))
    if(X==None):
        for data, label in zip(datas, labels):
            plt.plot(data, label=label)
    else:
        for data, label in zip(datas, labels):
            plt.plot(X, data, label=label)

    plt.xlabel(x_label)
    plt.ylabel(prop_name)
    plt.legend()
```

```
    plt.savefig("{}.png".format(figname))
    plt.close()

def train(d, m, b, lr, X_test=None, Y_test=None, eps=1e-3):
    print("d: {}, m: {}, b: {}".format(d,m,b))

    mean_w = np.zeros(shape=(d,))
    std_w = (1.0/d)*np.ones(shape=(d,))

    W = np.array([np.random.normal(mean_w, std_w) for i in range(m)]).T

    num_iters = 200
    f_arr = []
    f_old = 99999999
    count = 0
    for iter in range(num_iters):
        X = np.array([np.random.normal(np.zeros(shape=(d,)), np.ones(shape=(d,))) for
j in range(b)])
        Y = gen_gt(X)
        W, f = SGD_step(W, X, Y, lr)
        print("iter: [{}/{}], f: {}".format(iter, num_iters, f))
        pred_test = h(W, X_test, require_grad=False)
        err_test = Y_test - pred_test
        f_test = np.mean(err_test**2)
        f_arr.append(f_test*1.0)
        if(abs(f-f_old)<eps):
            break
        else:
            f_old = f*1.0
            count += 1

    return np.array(f_arr), count

def train_epochs(X, Y, d, m, b, lr, eps=1e-3):
    print("d: {}, m: {}, b: {}".format(d,m,b))

    mean_w = np.zeros(shape=(d,))
    std_w = (1.0/d)*np.ones(shape=(d,))

    W = np.array([np.random.normal(mean_w, std_w) for i in range(m)]).T

    num_iters = 200
    f_arr = []
    f_old = 99999999
    count = 0

    ids = np.arange(X.shape[0])
    for epoch in range(num_epochs):
        np.random.shuffle(ids)
        f_epoch = 0.0
        for i in range(0, X.shape[0], b):
            batch_ids = ids[i:np.amin(i+b, X.shape[0])]
            X_batch = X[batch_ids]
            Y_batch = Y[batch_ids]
            W, f = SGD_step(W, X_batch, Y_batch, lr)
            print("iter: [{}/{}], f: {}".format(iter, num_iters, f))
            f_epoch += f*1.0
            if(abs(f-f_old)<eps):
                break
            else:
                f_old = f*1.0
                count += 1
        f_arr.append(f_epoch*1.0)

    return np.array(f_arr), count

def main():
    d = 20
    m_start = 20
```

```
    m_end = 201
    m_step = 10
    b = 32
    lr = 1e-3
    datas = []
    labels = []
    counts = []
    f_min_arr = []
    Ms = []
    save_path = os.path.join(os.getcwd(), "d_{}_b_{}_lr_{}".format(d,b,lr))
    if not os.path.exists(save_path):
        os.mkdir(save_path)
    mean_x = np.zeros((d,))
    std_x = np.ones((d,))

    num_test_samples = 1000
    X_test = np.array([np.random.normal(mean_x, std_x) for i in range(num_test_samples
)])
    Y_test = gen_gt(X_test)

    for m in range(m_start,m_end, m_step):
        f_arr, count = train(d, m, b, lr, X_test, Y_test)
        f_min_arr.append(np.amin(f_arr))
        datas.append(f_arr)
        labels.append("m = {}".format(m))
        Ms.append(m)
        counts.append(count)
    plot_prop("num_iters", [datas[0], datas[-1]], ["proper", "over"], "f", os.path.joi
n(save_path,"part_1"))
    plot_prop("num_iters", datas, labels, "f", os.path.join(save_path, "bonus_f"))
    plot_prop("m", [counts], ["num_iters"], "num_iters", os.path.join(save_path, "bonu
s_iters"), Ms)
    plot_prop("m", [f_min_arr], ["f_min"], "f_min", os.path.join(save_path, "bonus_f_m
in"), Ms)
    # plot_prop(f_arr, "f", figname="q1_d_{}_m_{}_b_{}".format(d,m,b))
main()
```