```
import numpy as np
```

```
data = np.load("mcs_hw2_p3_data.npy")
```

```
x = data[:, :2]
y = data[:, 2]
```

```
import scipy.stats

def get_gradient_mu(beta, mu, sigma2):
    return (beta - mu) / sigma2

def get_gradient_logsigma2(beta, mu, sigma2):
    norm = np.linalg.norm(beta - mu)
    return (- 1 / sigma2 + norm * norm / (2 * sigma2 * sigma2)) * sigma2

def get_gradient_mu_t(beta, mu, sigma2, v):
    res = -(v + 2) / 2
    norm = np.linalg.norm(beta - mu)
    res *= 1 / (1 + norm * norm / (v * sigma2))
    res *= 2 * (mu - beta) / (v * sigma2)
    return res

def get_gradient_logsigma2_t(beta, mu, sigm2, v):
    res_1 = - 1 / sigma2
    norm = np.linalg.norm(beta - mu)
    res_2 = (v + 2) / 2
    res_2 *= 1 / (1 + norm * norm / (v * sigma2))
    res_2 *= norm * norm / v
    res_2 *= -1 / (sigma2 * sigma2)
    return res_1 - res_2

def sigmoid(x):
    return 1 / (1 + np.exp(-x))

def get_log_p(x, y, beta):
    res = 0.0
    res += np.sum(y * np.log(sigmoid(np.dot(x, beta))) + (1.0 - y) * np.log(1 -
sigmoid(np.dot(x, beta))))
    res += np.sum(scipy.stats.norm.logpdf(beta, np.zeros(2), np.ones(2)))
    return res

def get_log_q(mu, sigma2, beta):
    res = np.sum(scipy.stats.norm.logpdf(beta, mu, np.sqrt(sigma2)))
    return res
```

```
import concurrent.futures

def elbo(x, y, mu, sigma2):
    res = 0.0
    sample_size = 1024
    sample_beta = np.random.normal(mu, np.sqrt(sigma2), size=[sample_size,
mu.shape[0]])
    with concurrent.futures.ThreadPoolExecutor(max_workers=32) as executor:
        future_list = [executor.submit(get_log_p, x, y, beta) for beta in sample_beta]
        for future in concurrent.futures.as_completed(future_list):
            res += future.result()

    with concurrent.futures.ThreadPoolExecutor(max_workers=32) as executor:
        future_list = [executor.submit(get_log_q, mu, sigma2, beta) for beta in
sample_beta]
        for future in concurrent.futures.as_completed(future_list):
            res += future.result()

    return res / sample_size
```

```
def bbvi(x, y, mu, sigma2, lr, n_iter, m, v):
    sample_size = 4
    sample_beta = np.random.normal(mu, np.sqrt(sigma2), size=[sample_size,
mu.shape[0]])
    # update mu
    loss_mu = np.zeros(shape=[sample_size, mu.shape[0]])
    loss_logsigma2 = np.zeros(shape=[sample_size, sigma2.shape[0]])
    for i in range(sample_size):
        loss_mu[i] = get_gradient_mu(sample_beta[i], mu, sigma2)
        loss_logsigma2[i] = get_gradient_logsigma2(sample_beta[i], mu, sigma2)
        log_p = get_log_p(x, y, sample_beta[i])
        log_q = get_log_q(mu, sigma2, sample_beta[i])
        loss_mu[i] *= (log_p - log_q)
        loss_logsigma2[i] *= (log_p - log_q)
    update_mu = np.mean(loss_mu, axis=0)
    var_loss_mu = np.var(loss_mu, axis=0)
    update_logsigma2 = np.mean(loss_logsigma2, axis=0)
    var_loss_logsigma2 = np.var(loss_logsigma2, axis=0)

    grad = np.concatenate([update_mu, update_logsigma2])

    m = 0.9 * m + 0.1 * grad
    v = 0.999 * v + 0.001 * np.power(grad, 2)

    m_hat = m / (1 - np.power(0.9, n_iter))
    v_hat = v / (1 - np.power(0.999, n_iter))

    update = m_hat / (np.sqrt(v_hat) + 1e-10)

    mu += lr * update[:2]
    sigma2 = np.exp(np.log(sigma2) + lr * update[2])
    return mu, sigma2, m, v, var_loss_mu, var_loss_logsigma2
```

```
def train_bbvi(x, y, n_iter):
    mu_list = []
    sigma2_list = []
    var_list = []
    mu = np.random.normal(size=2)
    sigma2 = np.power(np.random.normal(size=1), 2)
    lr = 0.1
    m = np.zeros(shape=3)
    v = np.zeros(shape=3)
    for i in range(n_iter):
        mu, sigma2, m, v, var_mu, var_sigma = bbvi(x, y, mu, sigma2, lr, i + 1, m, v)
        mu_list.append(mu.copy())
        sigma2_list.append(sigma2.copy())
        var_list.append([var_mu.copy(), var_sigma.copy()])
    return mu_list, sigma2_list, var_list
```

```
def bbvi_cv(x, y, mu, sigma2, lr, n_iter, m, v):
    sample_size = 4
    sample_beta = np.random.normal(mu, np.sqrt(sigma2), size=[sample_size,
mu.shape[0]])
    # update mu
    loss_mu = np.zeros(shape=[sample_size, mu.shape[0]])
    loss_logsigma2 = np.zeros(shape=[sample_size, sigma2.shape[0]])
    cv_mu = np.zeros(shape=[sample_size, mu.shape[0]])
    cv_sigma2 = np.zeros(shape=[sample_size, sigma2.shape[0]])
    for i in range(sample_size):
        loss_mu[i] = cv_mu[i] = get_gradient_mu(sample_beta[i], mu, sigma2)
        loss_logsigma2[i] = cv_sigma2[i] = get_gradient_logsigma2(sample_beta[i], mu,
sigma2)
        log_p = get_log_p(x, y, sample_beta[i])
        log_q = get_log_q(mu, sigma2, sample_beta[i])
        loss_mu[i] *= (log_p - log_q)
        loss_logsigma2[i] *= (log_p - log_q)

    cov_mu0 = np.cov(np.stack((cv_mu.T[0], loss_mu.T[0]), axis=0))
    a_mu0 = cov_mu0[0][1] / cov_mu0[0][0]
    cov_mu1 = np.cov(np.stack((cv_mu.T[1], loss_mu.T[1]), axis=0))
    a_mu1 = cov_mu1[0][1] / cov_mu1[0][0]
    cov_logsigma2 = np.cov(np.stack((cv_sigma2.T[0], loss_logsigma2.T[0]), axis=0))
    a_logsigma2 = cov_logsigma2[0][1] / cov_logsigma2[0][0]

    update_mu = np.mean(loss_mu, axis=0)
    update_logsigma2 = np.mean(loss_logsigma2, axis=0)
    update_h_mu = np.mean(cv_mu, axis=0) * [a_mu0, a_mu1]
    update_h_logsigma2 = np.mean(cv_sigma2, axis=0) * a_logsigma2

    var_mu = np.var(loss_mu - cv_mu * [a_mu0, a_mu1], axis=0)
    var_sigma = np.var(loss_logsigma2 - cv_sigma2 * a_logsigma2, axis=0)

    grad = np.concatenate([update_mu - update_h_mu, update_logsigma2 -
update_h_logsigma2])
```

```
    m = 0.9 * m + 0.1 * grad
    v = 0.999 * v + 0.001 * np.power(grad, 2)

    m_hat = m / (1 - np.power(0.9, n_iter))
    v_hat = v / (1 - np.power(0.999, n_iter))

    update = m_hat / (np.sqrt(v_hat) + 1e-10)

    mu += lr * update[:2]
    sigma2 = np.exp(np.log(sigma2) + lr * update[2])
    #print(mu, sigma2)
    return mu, sigma2, m, v, var_mu, var_sigma
```

```
def train_bbvi_cv(x, y, n_iter):
    mu_list = []
    sigma2_list = []
    var_list = []
    mu = np.random.normal(size=2)
    sigma2 = np.power(np.random.normal(size=1), 2)
    lr = 0.1
    m = np.zeros(shape=3)
    v = np.zeros(shape=3)
    for i in range(n_iter):
        mu, sigma2, m, v, var_mu, var_sigma = bbvi_cv(x, y, mu, sigma2, lr, i + 1, m,
v)
        mu_list.append(mu.copy())
        sigma2_list.append(sigma2.copy())
        var_list.append([var_mu.copy(), var_sigma.copy()])
    return mu_list, sigma2_list, var_list
```

```
def get_gradient_mu_rt(x, y, mu, sigma2, eps):
    beta = mu + eps * np.sqrt(sigma2)
    data_part = (y * (1 - sigmoid(np.dot(x, beta))))[:, None] * x
    data_part += ((y - 1) * sigmoid(np.dot(x, beta)))[:, None] * x
    data_part = np.sum(data_part, axis=0)
    return data_part - beta
```

```
def get_gradient_logsigma2_rt(x, y, mu, sigma2, eps):
    res = 0.0
    beta = mu + eps * np.sqrt(sigma2)
    data_part = (y * (1 - sigmoid(np.dot(x, beta))))[:, None] * x
    data_part += ((y - 1) * sigmoid(np.dot(x, beta)))[:, None] * x
    data_part = np.sum(data_part, axis=0)
    res += (data_part - beta) * eps / (2 * np.sqrt(sigma2))
    res = np.sum(res)
    res += 1 / sigma2
    return res
```

```
def bbvi_rt(x, y, mu, sigma2, lr, n_iter, m, v):
    sample_size = 4
    sample_eps = np.random.normal(size=[sample_size, mu.shape[0]])
```

```
    # update mu
    loss_mu = np.zeros(shape=[sample_size, mu.shape[0]])
    loss_logsigma2 = np.zeros(shape=[sample_size, sigma2.shape[0]])
    for i in range(sample_size):
        loss_mu[i] = get_gradient_mu_rt(x, y, mu, sigma2, sample_eps[i])
        loss_logsigma2[i] = get_gradient_logsigma2_rt(x, y, mu, sigma2, sample_eps[i])
    update_mu = np.mean(loss_mu, axis=0)
    update_logsigma2 = np.mean(loss_logsigma2, axis=0)

    var_loss_mu = np.var(loss_mu, axis=0)
    var_loss_logsigma2 = np.var(loss_logsigma2, axis=0)

    grad = np.concatenate([update_mu, update_logsigma2])
    m = 0.9 * m + 0.1 * grad
    v = 0.999 * v + 0.001 * np.power(grad, 2)

    m_hat = m / (1 - np.power(0.9, n_iter))
    v_hat = v / (1 - np.power(0.999, n_iter))

    update = m_hat / (np.sqrt(v_hat) + 1e-10)

    mu += lr * update[:2]
    sigma2 = np.exp(np.log(sigma2) + lr * update[2])
    #print(mu, sigma2)
    return mu, sigma2, m, v, var_loss_mu, var_loss_logsigma2
```

```
def train_bbvi_rt(x, y, n_iter):
    mu_list = []
    sigma2_list = []
    var_list = []
    mu = np.random.normal(size=2)
    sigma2 = np.power(np.random.normal(size=1), 2)
    lr = 0.1
    m = np.zeros(shape=3)
    v = np.zeros(shape=3)
    for i in range(n_iter):
        mu, sigma2, m, v, var_mu, var_sigma = bbvi_rt(x, y, mu, sigma2, lr, i + 1, m,
v)
        mu_list.append(mu.copy())
        sigma2_list.append(sigma2.copy())
        var_list.append([var_mu.copy(), var_sigma.copy()])
    return mu_list, sigma2_list, var_list
```

```
def get_batch(X, Y, batch_size):
    n = X.shape[0]
    start_ele = np.random.randint(0, n)
    if start_ele + batch_size >= n:
        X_batch = np.concatenate((X[start_ele: ], X[ :start_ele + batch_size - n]))
        Y_batch = np.concatenate((Y[start_ele: ], Y[ :start_ele + batch_size - n]))
    else:
        X_batch = X[start_ele: (start_ele + batch_size)]
        Y_batch = Y[start_ele: (start_ele + batch_size)]
    return X_batch, Y_batch
```

```
def train_bbvi_batch(x, y, n_iter, batch_size):
    mu_list = []
    sigma2_list = []
    var_list = []
    mu = np.random.normal(size=2)
    sigma2 = np.power(np.random.normal(size=1), 2)
    lr = 0.1
    m = np.zeros(shape=3)
    v = np.zeros(shape=3)
    for i in range(n_iter):
        x_b, y_b = get_batch(x, y, batch_size)
        mu, sigma2, m, v = bbvi(x, y, mu, sigma2, lr, i + 1, m, v)
        mu_list.append(mu.copy())
        sigma2_list.append(sigma2.copy())
    return mu_list, sigma2_list
```

```
def train_bbvi_rt_batch(x, y, n_iter, batch_size):
    mu_list = []
    sigma2_list = []
    var_list = []
    mu = np.random.normal(size=2)
    sigma2 = np.power(np.random.normal(size=1), 2)
    lr = 0.1
    m = np.zeros(shape=3)
    v = np.zeros(shape=3)
    for i in range(n_iter):
        x_b, y_b = get_batch(x, y, batch_size)
        mu, sigma2, m, v, var_mu, var_sigma = bbvi_rt(x, y, mu, sigma2, lr, i + 1, m,
v)
        mu_list.append(mu.copy())
        sigma2_list.append(sigma2.copy())
        var_list.append([var_mu.copy(), var_sigma.copy()])
    return mu_list, sigma2_list, var_list
```

```
def train_bbvi_cv_batch(x, y, n_iter, batch_size):
    mu_list = []
    sigma2_list = []
    mu = np.random.normal(size=2)
    sigma2 = np.power(np.random.normal(size=1), 2)
    lr = 0.1
```

```
    m = np.zeros(shape=3)
    v = np.zeros(shape=3)
    for i in range(n_iter):
        x_b, y_b = get_batch(x, y, batch_size)
        mu, sigma2, m, v = bbvi_cv(x, y, mu, sigma2, lr, i + 1, m, v)
        mu_list.append(mu.copy())
        sigma2_list.append(sigma2.copy())
    return mu_list, sigma2_list
```

```
mu_bbvi, sigma2_bbvi, var_bbvi = train_bbvi(x, y, 1000)
```

```
mu_bbvi_cv, sigma2_bbvi_cv, var_bbvi_cv = train_bbvi_cv(x, y, 1000)
```

```
mu_bbvi_rt, sigma2_bbvi_rt, var_bbvi_rt = train_bbvi_rt(x, y, 1000)
```