```python
import numpy as np
import time
np.random.seed(1234)

n, d = 100000, 100
X = np.random.normal(size=(n,d))
beta_0 = np.random.normal(size=d)

def get_prob(X, beta):
    return 1.0 / (1.0 + np.exp(- np.matmul(X, beta)))

def gen_label(X, beta):
    return np.random.binomial(1, get_prob(X, beta))

Y_gt = gen_label(X, beta_0)

def get_gradient(X, Y_gt, Y_pred, batch_size):
    grad_beta = np.dot(np.transpose(X), Y_gt - Y_pred)
    return grad_beta / batch_size

def get_likelihood(X, Y_gt, prob):
    res = 1.0
    for i in range(X.shape[0]):
        if Y_gt[i] == 1.0:
            res *= prob[i]
        else:
            res *= 1.0 - prob[i]
    return res

def lr_scheduler(lr_init, step, decay):
    warm_up_step = 100.0
    lr_decay = 1e-6
    if step <= warm_up_step:
        return lr_init * step / warm_up_step
    if not decay:
        return lr_init
    return np.power(1 - lr_decay, step - 100) * lr_init

def get_batch(X, Y_gt, start_ele, batch_size):
    if start_ele + batch_size >= n:
        X_batch = np.concatenate((X[start_ele: ], X[ :start_ele + batch_size - n]))
        Y_gt_batch = np.concatenate((Y_gt[start_ele: ], Y_gt[ :start_ele + batch_size - n]))
    else:
        X_batch = X[start_ele: (start_ele + batch_size)]
        Y_gt_batch = Y_gt[start_ele: (start_ele + batch_size)]
    return X_batch, Y_gt_batch

# batch size = n
def gd(X, Y_gt, lr_init, d, beta_est):
    start = time.time()
    loss = []
    beta = np.zeros(shape=d)
    step = 1
    while True:
        Y_pred = get_prob(X, beta)
        grad_beta = get_gradient(X, Y_gt, Y_pred, X.shape[0])
        lr = lr_scheduler(lr_init, step, 1)
        beta += lr * grad_beta
        step += 1
        loss_this = np.sum(np.absolute(beta_est - beta))
        loss.append(loss_this)
        if loss_this < 1e-3 or step > 1e4:
            print("vanilla GD ended with L_1 diff as: ", np.sum(np.absolute(beta_est - beta)))
            print("Total time:", time.time() - start, "total steps:", step)
            break;
    return loss

def nag(X, Y_gt, lr_init, d, beta_est):
    start = time.time()
    loss = []
    beta = np.zeros(shape=d)
    step = 1
```

```
    beta_tmp = beta
    while True:
        y_beta = beta + ((step - 2.0) / (step + 1.0)) * (beta - beta_tmp)
        Y_pred = get_prob(X, y_beta)
        grad_beta = get_gradient(X, Y_gt, Y_pred, X.shape[0])
        lr =  lr_scheduler(lr_init, step, 1)
        beta_tmp = beta
        beta = y_beta + lr * grad_beta
        step += 1
        loss_this = np.sum(np.absolute(beta_est - beta))
        loss.append(loss_this)
        if loss_this < 1e-3 or step > 1e4:
            print("NAG ended with L_1 diff as: ", np.sum(np.absolute(beta_est - beta)))
            print("Total time:", time.time() - start, "total steps:", step)
            break;
    return loss, beta

def adagrad(X, Y_gt, lr_init, d, eps, batch_size, beta_est):
    start = time.time()
    loss = []
    beta = np.zeros(shape=d)
    step = 1
    g_beta = np.zeros(shape=d)
    while True:
        start_ele = ((step - 1) * batch_size) % n
        X_batch, Y_gt_batch = get_batch(X, Y_gt, start_ele, batch_size)
        Y_pred = get_prob(X_batch, beta)
        grad_beta = get_gradient(X_batch, Y_gt_batch, Y_pred, batch_size)
        g_beta += np.square(grad_beta)
        lr =  lr_scheduler(lr_init, step, 0)
        beta += lr * np.multiply((1.0 / np.sqrt(g_beta + eps)), grad_beta)
        step += 1
        loss.append(np.sum(np.absolute(beta_est - beta)))
        if step > 1e5:
            print("AdaGrad ended with L_1 diff as: ", np.sum(np.absolute(beta_est - beta)))
            print("Total time:", time.time() - start)
            break;
    return loss

def rmsprop(X, Y_gt, lr_init, d, eps, batch_size,beta_est):
    start = time.time()
    loss = []
    beta = np.zeros(shape=d)
    step = 1
    g_beta = np.zeros(shape=d)
    while True:
        start_ele = ((step - 1) * batch_size) % n
        X_batch, Y_gt_batch = get_batch(X, Y_gt, start_ele, batch_size)
        Y_pred = get_prob(X_batch, beta)
        grad_beta = get_gradient(X_batch, Y_gt_batch, Y_pred, batch_size)
        g_beta = 0.9 * g_beta + 0.1 * np.square(grad_beta)
        lr =  lr_scheduler(lr_init, step, 0)
        beta += lr * np.multiply((1.0 / np.sqrt(g_beta + eps)), grad_beta)
        step += 1
        loss.append(np.sum(np.absolute(beta_est - beta)))
        if step > 1e5:
            print("RMSprop ended with L_1 diff as: ", np.sum(np.absolute(beta_est - beta)))
            print("Total time:", time.time() - start)
            break;
    return loss

def sgd(X, Y_gt, lr_init, d, batch_size, beta_est):
    start = time.time()
    loss = []
    beta = np.zeros(shape=d)
    step = 1
    while True:
        start_ele = ((step - 1) * batch_size) % n
        X_batch, Y_gt_batch = get_batch(X, Y_gt, start_ele, batch_size)
        Y_pred = get_prob(X_batch, beta)
        grad_beta = get_gradient(X_batch, Y_gt_batch, Y_pred, batch_size)
        lr =  lr_scheduler(lr_init, step, 1)
        beta += lr * grad_beta
        step += 1
```

```
        loss.append(np.sum(np.absolute(beta_est - beta)))
        if step > 1e5:
            print("SGD ended with L_1 diff as: ", np.sum(np.absolute(beta_est - beta)))
            print("Total time:", time.time() - start)
            break;
    return loss

def adam(X, Y_gt, lr_init, d, b_1, b_2, eps, batch_size, beta_est):
    start = time.time()
    loss = []
    beta = np.zeros(shape=d)
    step = 1
    m_beta = np.zeros(shape=d)
    v_beta = np.zeros(shape=d)
    while True:
        start_ele = ((step - 1) * batch_size) % n
        X_batch, Y_gt_batch = get_batch(X, Y_gt, start_ele, batch_size)
        Y_pred = get_prob(X_batch, beta)
        grad_beta = get_gradient(X_batch, Y_gt_batch, Y_pred, batch_size)
        lr =  lr_scheduler(lr_init, step, 0)
        m_beta = b_1 * m_beta + (1 - b_1) * grad_beta
        v_beta = b_2 * v_beta + (1 - b_2) * np.square(grad_beta)
        beta += lr * (m_beta / (1.0 - np.power(b_1, step)))
                / np.sqrt(eps + v_beta / (1.0 - np.power(b_2, step)))
        step += 1
        loss.append(np.sum(np.absolute(beta_est - beta)))
        if step > 1e5:
            print("Adam ended with L_1 diff as: ", np.sum(np.absolute(beta_est - beta)))
            print("Total time:", time.time() - start)
            break;
    return loss

```