

```

1 import numpy as np
2 import time
3 np.random.seed(1234)
4
5 n, d = 100000, 100
6 X = np.random.normal(size=(n,d))
7 beta_0 = np.random.normal(size=d)
8
9 def get_prob(X, beta):
10     return 1.0 / (1.0 + np.exp(- np.matmul(X, beta)))
11
12 def gen_label(X, beta):
13     return np.random.binomial(1, get_prob(X, beta))
14
15 Y_gt = gen_label(X, beta_0)
16
17 def get_gradient(X, Y_gt, Y_pred, batch_size):
18     grad_beta = np.dot(np.transpose(X), Y_gt - Y_pred)
19     return grad_beta / batch_size
20
21 def get_likelihood(X, Y_gt, prob):
22     res = 1.0
23     for i in range(X.shape[0]):
24         if Y_gt[i] == 1.0:
25             res *= prob[i]
26         else:
27             res *= 1.0 - prob[i]
28     return res
29
30 def lr_scheduler(lr_init, step, decay):
31     warm_up_step = 100.0
32     lr_decay = 1e-6
33     if step <= warm_up_step:
34         return lr_init * step / warm_up_step
35     if not decay:
36         return lr_init
37     return np.power(1 - lr_decay, step - 100) * lr_init
38
39 def get_batch(X, Y_gt, start_ele, batch_size):
40     if start_ele + batch_size >= n:
41         X_batch = np.concatenate((X[start_ele: ], X[ :start_ele + batch_size - n]))
42         Y_gt_batch = np.concatenate((Y_gt[start_ele: ], Y_gt[ :start_ele + batch_size - n]))
43     else:
44         X_batch = X[start_ele: (start_ele + batch_size)]
45         Y_gt_batch = Y_gt[start_ele: (start_ele + batch_size)]
46     return X_batch, Y_gt_batch
47
48 # batch size = n
49 def gd(X, Y_gt, lr_init, d, beta_est):
50     start = time.time()
51     loss = []
52     beta = np.zeros(shape=d)
53     step = 1
54     while True:
55         Y_pred = get_prob(X, beta)
56         grad_beta = get_gradient(X, Y_gt, Y_pred, X.shape[0])
57         lr = lr_scheduler(lr_init, step, 1)
58         beta += lr * grad_beta
59         step += 1
60         loss_this = np.sum(np.absolute(beta_est - beta))
61         loss.append(loss_this)
62         if loss_this < 1e-3 or step > 1e4:
63             print("vanilla GD ended with L1 diff as: ", np.sum(np.absolute(beta_est - beta)))
64             print("Total time:", time.time() - start, "total steps:", step)
65             break;
66     return loss
67
68 def nag(X, Y_gt, lr_init, d, beta_est):
69     start = time.time()
70     loss = []
71     beta = np.zeros(shape=d)
72     step = 1

```

```

73     beta_tmp = beta
74     while True:
75         y_beta = beta + ((step - 2.0) / (step + 1.0)) * (beta - beta_tmp)
76         Y_pred = get_prob(X, y_beta)
77         grad_beta = get_gradient(X, Y_gt, Y_pred, X.shape[0])
78         lr = lr_scheduler(lr_init, step, 1)
79         beta_tmp = beta
80         beta = y_beta + lr * grad_beta
81         step += 1
82         loss_this = np.sum(np.absolute(beta_est - beta))
83         loss.append(loss_this)
84         if loss_this < 1e-3 or step > 1e4:
85             print("NAG ended with L1 diff as: ", np.sum(np.absolute(beta_est - beta)))
86             print("Total time:", time.time() - start, "total steps:", step)
87             break;
88     return loss, beta
89
90 def adagrad(X, Y_gt, lr_init, d, eps, batch_size, beta_est):
91     start = time.time()
92     loss = []
93     beta = np.zeros(shape=d)
94     step = 1
95     g_beta = np.zeros(shape=d)
96     while True:
97         start_ele = ((step - 1) * batch_size) % n
98         X_batch, Y_gt_batch = get_batch(X, Y_gt, start_ele, batch_size)
99         Y_pred = get_prob(X_batch, beta)
100        grad_beta = get_gradient(X_batch, Y_gt_batch, Y_pred, batch_size)
101        g_beta += np.square(grad_beta)
102        lr = lr_scheduler(lr_init, step, 0)
103        beta += lr * np.multiply((1.0 / np.sqrt(g_beta + eps)), grad_beta)
104        step += 1
105        loss.append(np.sum(np.absolute(beta_est - beta)))
106        if step > 1e5:
107            print("AdaGrad ended with L1 diff as: ", np.sum(np.absolute(beta_est - beta)))
108            print("Total time:", time.time() - start)
109            break;
110    return loss
111
112 def rmsprop(X, Y_gt, lr_init, d, eps, batch_size, beta_est):
113     start = time.time()
114     loss = []
115     beta = np.zeros(shape=d)
116     step = 1
117     g_beta = np.zeros(shape=d)
118     while True:
119         start_ele = ((step - 1) * batch_size) % n
120         X_batch, Y_gt_batch = get_batch(X, Y_gt, start_ele, batch_size)
121         Y_pred = get_prob(X_batch, beta)
122         grad_beta = get_gradient(X_batch, Y_gt_batch, Y_pred, batch_size)
123         g_beta = 0.9 * g_beta + 0.1 * np.square(grad_beta)
124         lr = lr_scheduler(lr_init, step, 0)
125         beta += lr * np.multiply((1.0 / np.sqrt(g_beta + eps)), grad_beta)
126         step += 1
127         loss.append(np.sum(np.absolute(beta_est - beta)))
128         if step > 1e5:
129             print("RMSprop ended with L1 diff as: ", np.sum(np.absolute(beta_est - beta)))
130             print("Total time:", time.time() - start)
131             break;
132     return loss
133
134 def sgd(X, Y_gt, lr_init, d, batch_size, beta_est):
135     start = time.time()
136     loss = []
137     beta = np.zeros(shape=d)
138     step = 1
139     while True:
140         start_ele = ((step - 1) * batch_size) % n
141         X_batch, Y_gt_batch = get_batch(X, Y_gt, start_ele, batch_size)
142         Y_pred = get_prob(X_batch, beta)
143         grad_beta = get_gradient(X_batch, Y_gt_batch, Y_pred, batch_size)
144         lr = lr_scheduler(lr_init, step, 1)
145         beta += lr * grad_beta
146         step += 1

```

```

147     loss.append(np.sum(np.absolute(beta_est - beta)))
148     if step > 1e5:
149         print("SGD ended with L_1 diff as: ", np.sum(np.absolute(beta_est - beta)))
150         print("Total time:", time.time() - start)
151         break;
152     return loss
153
154 def adam(X, Y_gt, lr_init, d, b_1, b_2, eps, batch_size, beta_est):
155     start = time.time()
156     loss = []
157     beta = np.zeros(shape=d)
158     step = 1
159     m_beta = np.zeros(shape=d)
160     v_beta = np.zeros(shape=d)
161     while True:
162         start_ele = ((step - 1) * batch_size) % n
163         X_batch, Y_gt_batch = get_batch(X, Y_gt, start_ele, batch_size)
164         Y_pred = get_prob(X_batch, beta)
165         grad_beta = get_gradient(X_batch, Y_gt_batch, Y_pred, batch_size)
166         lr = lr_scheduler(lr_init, step, 0)
167         m_beta = b_1 * m_beta + (1 - b_1) * grad_beta
168         v_beta = b_2 * v_beta + (1 - b_2) * np.square(grad_beta)
169         beta += lr * (m_beta / (1.0 - np.power(b_1, step)))
170             / np.sqrt(eps + v_beta / (1.0 - np.power(b_2, step)))
171         step += 1
172         loss.append(np.sum(np.absolute(beta_est - beta)))
173         if step > 1e5:
174             print("Adam ended with L_1 diff as: ", np.sum(np.absolute(beta_est - beta)))
175             print("Total time:", time.time() - start)
176             break;
177     return loss
178

```