Management Sciences Topics: Convex Optimization
Final Project

1 Problem setup

We need to solve the optimization problem of a one-hidden-layer neural network

$$\min_{x_k\in\mathbb{R}^d, y_k\in\mathbb{R}, z\in\mathbb{R}^K, w\in\mathbb{R}} \frac{1}{n}\sum_{i=1}^{n}\mathcal{L}\left(b_i w + b_i \sum_{k=1}^{K}\sigma(a_i^\top x_k + y_k)z_k\right), \tag{1.1}$$

where K is the number of neurons, $a_i \in \mathbb{R}^d$ is a data point, $b_i \in \{-1, 1\}$ is the class label of a_i, $\sigma(z) = \max(z, 0)$ or $\frac{\exp(z)}{1+\exp(z)}$, $\mathcal{L}(z) = \max(1 - z, 0)$ or $\log(1 + \exp(-z))$.

2 Stochastic subgradient method

We first consider the subgradient with respect to each variables.

First, define the variables

$$X = [x_1, x_2, \dots, x_K] \in \mathbb{R}^{d\times K}, \quad Y = [y_1, y_2, \dots, y_k]^\top \in \mathbb{R}^{K\times 1}, \quad Z = [z_1, z_2, \dots, z_k]^\top \in \mathbb{R}^{K\times 1}. \tag{2.1}$$

Then a forward pass through the network can be written as

$$\begin{aligned} A_1 &= AX \oplus Y^\top, \\ A_2 &= \sigma(A_1)Z \oplus w \\ f &= \frac{1}{n}1^\top L(b \odot A_2). \end{aligned} \tag{2.2}$$

By chain rule, the subgradients of f with respect to each variable are

$$\begin{aligned} \frac{\partial f}{\partial A_2} &= \frac{1}{n}L'(b\odot A_2)\odot b \\ \frac{\partial f}{\partial w} &= \frac{\partial f}{\partial A_2}\frac{\partial A_2}{\partial w} = \mathrm{rowsum}(\frac{\partial f}{\partial A_2}\odot 1) = \left(\frac{\partial f}{\partial A_2}\right)^\top 1, \\ \frac{\partial f}{\partial Z} &= \frac{\partial f}{\partial A_2}\frac{\partial A_2}{\partial Z} = \sigma(A_1)^\top \frac{\partial f}{\partial A_2}, \\ \frac{\partial f}{\partial A_1} &= \frac{\partial f}{\partial A_2}\frac{\partial A_2}{\partial A_1} = \frac{\partial f}{\partial A_2}Z^\top \odot \sigma'(A_1), \\ \frac{\partial f}{\partial Y} &= \frac{\partial f}{\partial A_1}\frac{\partial A_1}{\partial Y} = \left(\frac{\partial f}{\partial A_1}\right)^\top 1, \\ \frac{\partial f}{\partial X} &= \frac{\partial f}{\partial A_1}\frac{\partial A_1}{\partial X} = A^\top \frac{\partial f}{\partial A_1}. \end{aligned} \tag{2.3}$$

The stochastic subgradients can be chosen to be the subgradient when input is a minibatch of the whole dataset, i.e.

$$G(x, \xi_i) = \partial_x f(x; A_{\xi_i}, b_{\xi_i}) \tag{2.4}$$

for each variable x, where ξ_i is a uniformly sample index set for each i. We can control the size of each ξ_i to vary from online learning to full-batch learning.

3 Accelerated proximal gradient method

In order to use APG for this problem, we need to choose

$$\sigma(z) = \frac{\exp(z)}{1+\exp(z)}, \quad \mathcal{L}(z) = \log(1+\exp(-z)) \tag{3.1}$$

to guarantee the objective function is smooth. Notice

$$\mathcal{L}'(z) = -\frac{1}{1+\exp(z)}, \tag{3.2}$$

and

$$|\mathcal{L}''(z)| = \left|\frac{e^z}{(1+e^z)^2}\right| \leq \frac{1}{4}, \tag{3.3}$$

by Lagrange mean value theorem, we know the Lipschitz constant for $\mathcal{L}'$ is $L = \frac{1}{4}$.

Now let's consider the Lipschitz constant for each derivatives.

For ∂_{A_2},

$$\begin{aligned}\left|\frac{\partial f}{\partial A_2^1} - \frac{\partial f}{\partial A_2^2}\right| &= \frac{1}{n}|\mathcal{L}'(b \odot A_2^1) \odot b - \mathcal{L}'(b \odot A_2^2) \odot b| = \frac{1}{n}|(\mathcal{L}'(b \odot A_2^1) - \mathcal{L}'(b \odot A_2^2)) \odot b| \\ &= \frac{1}{n}|\mathcal{L}''(\xi)(b \odot (A_2^1 - A_2^2)) \odot b| \\ &\leq \frac{1}{4n}|A_2^1 - A_2^2|.\end{aligned} \tag{3.4}$$

Then the Lipschitz constant for $\frac{\partial f}{\partial A_2}$ is $L_{A_2} = \frac{1}{4n}$.

For $\frac{\partial f}{\partial w}$,

$$\begin{aligned}\left|\frac{\partial f}{\partial w^1} - \frac{\partial f}{\partial w^2}\right| &= \left(\frac{\partial f}{\partial A_2^1} - \frac{\partial f}{\partial A_2^2}\right)^\top 1 \\ &\leq \frac{1}{4n}|w_1 - w_2|n = \frac{1}{4}|w_1 - w_2|.\end{aligned} \tag{3.5}$$

Then the Lipschitz constant for ∂_w is $L_w = \frac{1}{4}$.

For $\frac{\partial f}{\partial Z}$, we first notice $\sigma(x) \in (0,1)$. Then

$$\begin{aligned}\left|\frac{\partial f}{\partial Z^1} - \frac{\partial f}{\partial Z^2}\right| &= \left|\sigma(A_1)^\top \left(\frac{\partial f}{\partial A_2^1} - \frac{\partial f}{\partial A_2^2}\right)\right| \leq \frac{1}{4}\left|\sigma(A_1)^\top (A_2^1 - A_2^2)\right| \\ &= \frac{1}{4}|\sigma(A_1)^\top \sigma(A_1)(Z_1 - Z_2)| \\ &\leq \frac{1}{4}\|\sigma(A_1)^\top \sigma(A_1)\|_2 |Z_1 - Z_2| \\ &\leq \frac{1}{4}\|\sigma(A_1)^\top \sigma(A_1)\|_F |Z_1 - Z_2| \\ &\leq \frac{1}{4}\sqrt{k^2 n^4}|Z_1 - Z_2| = \frac{1}{4}kn^2|Z_1 - Z_2|.\end{aligned} \tag{3.6}$$

Then $L_z = \frac{1}{4}kn^2$ can be an upper bound for the Lipschitz constant for ∂_z.

The lipschitz constant for $\frac{\partial f}{\partial y}$ and $\frac{\partial f}{\partial x}$ are too complex to solve.

4 Proximal Gradient method with line search

In order to use PG for this problem, we need to choose

$$\sigma(z) = \frac{\exp(z)}{1+\exp(z)}, \quad \mathcal{L}(z) = \log(1+\exp(-z)) \tag{4.1}$$

to guarantee the objective function is smooth.

Since we use an iterative way to update the variables, the value of x will be updated from x_0 to x_1 before we start to update y, z, w. So there are two schemes in updating:

1. Use x_0 to update y, and use x_0, y_0 to update z, and use x_0, y_0, z_0 to update w.
2. Use x_1 to update y. In this case, we need to recompute the objective function and the derivatives using x_1. For y, z, w, we also use this scheme.

We test both schemes in the experiments.

5 Experiments

For the experiments, we test the one-hidden-layer neural network on the datasets `rcv1.binary` and `covtype` from libsvm library. We use the same random seed across the experiments to get the same random initialization for a fair comparison. For all following experiments, we compute the objective value, subgradient and out-of-sample accuracy on the averaged variable after each iteration.

We compute the prediction accuracy by

$$Acc = \frac{1}{n}\sum_{i=1}^{n} 1_{\mathrm{sign}(A_{2i})=\mathrm{sign}(b_i)} \times 100\%. \tag{5.1}$$

We compute 2-norm of ∂x by first flatten it to a vector, and then compute the vector 2-norm instead of matrix 2-norm.

5.1 rcv1.binary

For `rcv1.binary`, we use only the `rcv1_train.binary` from libsvm library. We randomly sample 20% of the dataset, i.e., 4048 samples to form the test set, and use the rest part as the training set.

Considering memory issues (i.e., when we run APG with $K = 200$ iterations, the memory occupation is over 32GB for this dataset since we need to keep track of both variables and their subgradients), for PG and APG, we run the algorithm with $K = 100$ iterations. For SSG, we run $K = 600$ iterations with minibatch size $N = 2699$ in each iteration. The the total number of training samples for all three algorithms are the same.