EP3260:Fundamentals of Machine Learning over Networks Computer Assignments 3

Group 3

Notation

Upper-case letters with a double underline denotes matrices, e.g., $\boldsymbol{A}$. Lower-case letters with a single underline denotes vectors, e.g., $\boldsymbol{a}$. The i-th element of the vector $\boldsymbol{a}$ is denoted by either $a\left[i\right]$ or a_i, and element in the i-th row and j-th column of the matrix $\boldsymbol{A}$ is denoted by $\boldsymbol{A}\left[i,j\right]$. An i-th column vector of a matrix $\boldsymbol{A}$ is denoted as either $\boldsymbol{A}_i$ or $\boldsymbol{A}[:,i]$.

1 Computer Assignment 3

The optimization problem can be expressed as

$$\min_{\boldsymbol{W}_1,\boldsymbol{W}_2,\boldsymbol{w}_3} \frac{1}{N}\sum_{i=1}^{N} f_i(\boldsymbol{W}_1,\boldsymbol{W}_2,\boldsymbol{w}_3), \tag{1}$$

and

$$f_i(\boldsymbol{W}_1,\boldsymbol{W}_2,\boldsymbol{w}_3) = \left\|\boldsymbol{w}_3^{\mathrm{T}}\boldsymbol{s}(\boldsymbol{W}_2\boldsymbol{s}(\boldsymbol{W}_1\boldsymbol{x}_i)) - y_i\right\|_2^2, \tag{2}$$

where $\boldsymbol{x}_i \in \mathbb{R}^{d\times 1}$, $\boldsymbol{W}_1 \in \mathbb{R}^{d_1\times d}$, $\boldsymbol{W}_2 \in \mathbb{R}^{d_2\times d_1}$, $\boldsymbol{w}_3 \in \mathbb{R}^{d_2\times 1}$.

In the “Individual household electric power consumption” dataset, $d = 4$ is small, we let $d_2 = d_1 = d$. In the “Communities and crime” dataset, $d = 123$ is large, we let $d_1 = \lceil d/2\rceil$, and $d_2 = \lceil d_1/2\rceil$. In this way, we decrease the size of the weights from inner to outer layer.

As $\boldsymbol{s}(\boldsymbol{x}) = 1/(1+\exp(-\boldsymbol{x}))$, we have

$$\frac{\partial s}{\partial \boldsymbol{x}} = \mathrm{diag}[\boldsymbol{s}^2(\boldsymbol{x}) \circ \exp(-\boldsymbol{x})] \tag{3}$$

with $\circ$ denotes Hadamard product.

Denote

$$a = \boldsymbol{w}_3^{\mathrm{T}}\boldsymbol{s}(\boldsymbol{W}_2\boldsymbol{s}(\boldsymbol{W}_1\boldsymbol{x}_i)), \tag{4}$$
$$\boldsymbol{b} = \boldsymbol{W}_2\boldsymbol{s}(\boldsymbol{W}_1\boldsymbol{x}_i), \tag{5}$$
$$\boldsymbol{c} = \boldsymbol{W}_1\boldsymbol{x}_i, \tag{6}$$

where a is scalar, $\boldsymbol{b} \in \mathbb{R}^{d\times 1}$, and $\boldsymbol{c} \in \mathbb{R}^{d\times 1}$. Then the derivative of f_i with respective to $\boldsymbol{W}_1, \boldsymbol{W}_2, \boldsymbol{w}_3$ are:

$$\frac{\partial f_i}{\partial \boldsymbol{w}_3} = 2(a - y_i)\cdot \boldsymbol{s}(\boldsymbol{b}), \tag{7}$$
$$\frac{\partial f_i}{\partial \boldsymbol{W}_2} = 2(a - y_i)\cdot (\boldsymbol{w}_3\cdot \boldsymbol{e})\cdot \boldsymbol{1}_{d_1}^{\mathrm{T}} \circ (\boldsymbol{1}_{d_2}\cdot \boldsymbol{s}^{\mathrm{T}}(\boldsymbol{c})) \tag{8}$$
$$\frac{\partial f_i}{\partial \boldsymbol{W}_1} = 2(a - y_i)\cdot \mathrm{diag}[\boldsymbol{f}\cdot(\boldsymbol{w}_3^{\mathrm{T}} \circ \boldsymbol{e}^{\mathrm{T}})\cdot \boldsymbol{W}_2]\cdot \boldsymbol{x}_i^{\mathrm{T}} \tag{9}$$

where $\boldsymbol{1}_d$ is a colomn vector with length d,
$\boldsymbol{e} \in \mathbb{R}^{d_2\times 1}$, and $\boldsymbol{e} = (\boldsymbol{s}^2(\boldsymbol{b}) \circ \exp(-\boldsymbol{b}))$,
$\boldsymbol{f} \in \mathbb{R}^{d_1\times 1}$, and $\boldsymbol{f} = (\boldsymbol{s}^2(\boldsymbol{c}) \circ \exp(-\boldsymbol{c})$.

1.1 Results with data sets

Table 1: Some Hyperparameters

Method	step-size	Nr of Epochs (outer iterations)	Epoch Length (inner iterations)	Mini-batch-size
GD	fixed $\{10^{-2}, 10^{-1}\}$	500	–	–
pGD	fixed $\{10^{-2}, 10^{-1}\}$	500	–	–
SGD	fixed $\{10^{-2}, 10^{-1}\}$	500	–	10
SVRG	fixed $\{10^{-2}, 10^{-1}\}$	500	20	10
BCD	fixed $\{10^{-2}, 10^{-1}\}$	500	–	–

We detail in Table 1 the hyper-parameters that we can tune for each algorithm. The results for the two datasets are shown in Figure 1 and Figure 2. In Fig. 1 where the step size equals 0.1 and is pretty

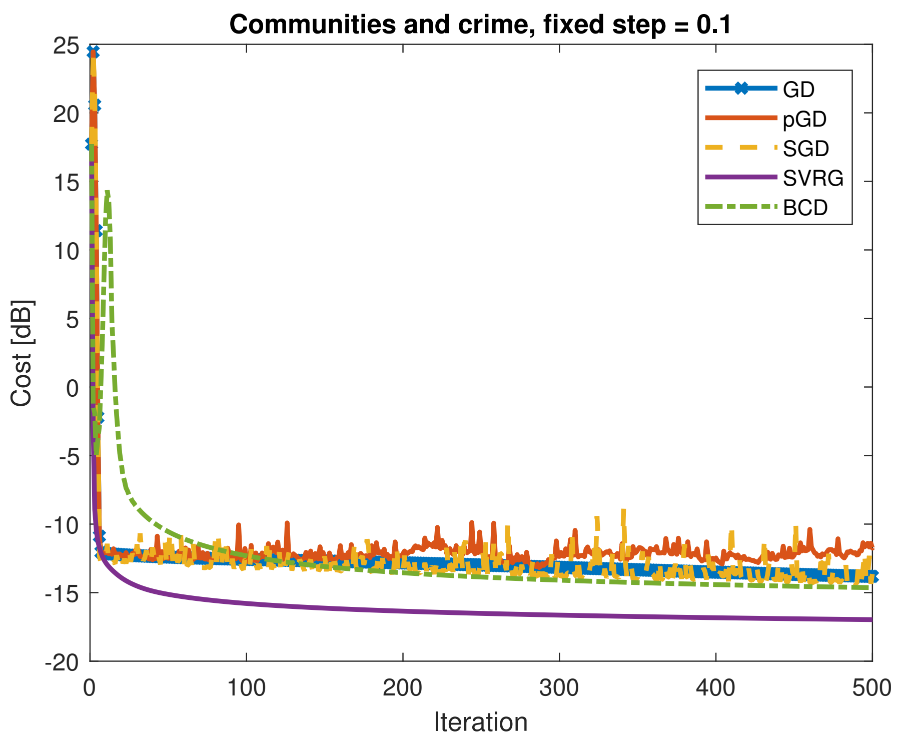

Figure 1: Cost [dB] versus iterations for various methods, namely GD, pGD, sGD, SVRG, and BCD, with fixed step-size, batch-size, and inner iterations (valid for SVRG) for "Communities and crime" dataset.

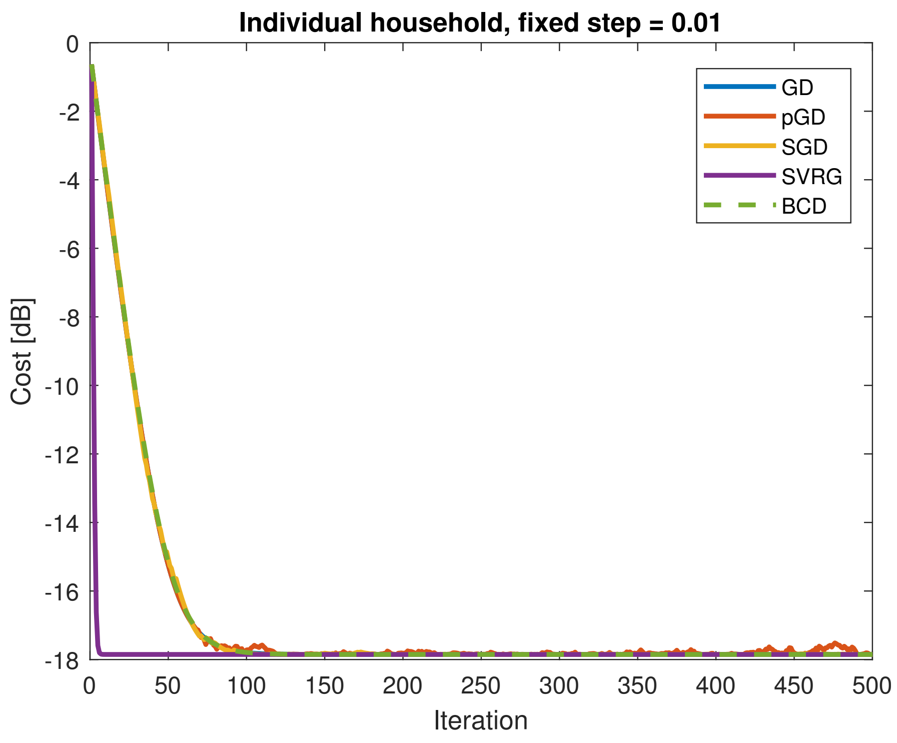

Figure 2: Cost [dB] versus iterations for various methods, namely GD, pGD, sGD, SVRG, and BCD, with fixed step-size, batch-size, and inner iterations (valid for SVRG) for "Individual household" dataset.

big, GD, pGD, sGD and SVRG converge with only few steps. While BCD takes more steps, it achieves lower cost compared to GD, pGD, sGD. As the norm of the first order gradients of the weights drops, noise is added to pGD, and its weights follow some random direction. As sGD is also naturally has the random property, its cost also goes through some fluctuation.

In Figure 2, it takes SVRG only a few iterations to converge. GD, psG, sGD and BCD takes around 100 iterations to converge, and they achieve similar minimum cost compared with SVRG.

1.2 Compare these solvers in terms complexity of hyper-parameter tunning, convergence time, convergence rate (in terms of # outer-loop iterations), and memory requirement

Regarding the convergence time, we fixed the number of outer loop iterations so that we could evaluate all the algorithms with the same outer number of iterations, and using the setup from Figure 1. From Table 2, we notice that SGD is the fastest algorithms, while GD, pGD, BCD have similar speed, and SVRG take the longest time. This behaviour is expected because SGD has cheaper iterations than all the others. For GD, pGD, BCD, they all calculate the full gradients. For SVRG, recall that we run the inner loop with T epochs, so we notice that running 20 epochs really increases the running time.

Regarding the convergence rate, we compare the # outer-loop iterations of the solvers, and SVRG has the fastest rate, while the other four has similar fat rate. For BCD, from Lecture 4, slide 17, if each coordinate is L_g-smooth, then the convergence rate id of $\mathcal{O}\left(dL_g/\epsilon_g\right)$, thus when we decrease the size of the weights from the inner layer to outer layer in the “Communities and crime” dataset, it is effective to improve the convergence rate.

Regarding the memory requirement, we include it in Table 2 along with the theoretical requirements based on gradient. GD, pGD, and BCD have the same memory requirement, and sGD has the smallest memory requirement.

Table 2: Complexity, convergence, and memory requirement Analysis

Method	Convergence time (2000 fixed iterations)	Convergence rate	Memory requirement
GD	158s	fast	1 memory of 3 gradients
pGD	169s	fast	1 memory of 3 gradients
SGD	3.8s	fast	1 memory of 3 gradient batches
SVRG	214s	fastest	1 memory of 6 two gradient batches
BCD	161s	$\mathcal{O}\left(dL_g/\epsilon_g\right)$	1 memory of 3 gradients

README: How to run the MATLAB script

For both computer assignments, we have main script $\mathrm{main\_ca3.m}$. One could simply run them with default set of parameters.