# EP3260:Fundamentals of Machine Learning over Networks Computer Assignments 1 & 2

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 1

The linear regression optimization (unconstrained) problem is given by

$$\boldsymbol{w}^{\star} = \min_{\boldsymbol{w}\in\mathbb{R}^d} \frac{1}{N}\sum_{i=1}^{N} \left\|\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_i - y_i\right\|_2^2 + \lambda\|\boldsymbol{w}\|_2^2 \,, \tag{1}$$

where the data set $\mathcal{D} = \{\boldsymbol{x}_i, y_i\}_{i=1}^{N}$, $y_i \in \mathbb{R}$, $\boldsymbol{w}, \boldsymbol{x}_i \in \mathbb{R}^N$, and $\lambda \in \mathbb{R}$.

## 1.1 Find a closed-form solution for this problem

Let the cost function

$$f(\boldsymbol{w}) \coloneqq \frac{1}{N}\sum_{i=1}^{N} \left\|\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_i - y_i\right\|_2^2 + \lambda\|\boldsymbol{w}\|_2^2 \tag{2}$$

$$= \frac{1}{N}\left\|\boldsymbol{X}\boldsymbol{w} - \boldsymbol{y}\right\|_2^2 + \lambda\|\boldsymbol{w}\|_2^2 \,, \tag{3}$$

where the input datum matrix reads $\boldsymbol{X} \in \mathbb{R}^{N\times d}$.

To this end, taking derivative with respect to $\boldsymbol{w}$ and setting it to zero; thereby obtaining closed-form weight vector

$$\frac{\partial f}{\partial \boldsymbol{w}} = 0 \Longrightarrow \boldsymbol{X}^{\mathrm{T}}\left(\boldsymbol{X}\boldsymbol{w} - \boldsymbol{y}\right) + \lambda N\boldsymbol{w} = 0 \Longleftrightarrow \boldsymbol{w} = \left(\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X} + \lambda N\boldsymbol{I}_d\right)^{-1}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{y} \,. \tag{4}$$

## 1.2 Consider *Communities and Crime* dataset (N = 1994, d = 128) and find the optimal linear regressor from the closed-form expression

We divided the *Communities and Crime* dataset $\mathcal{D}^1 \coloneqq \left\{\boldsymbol{x}_i \in \mathbb{R}^d, y_i \in [-1, 1]\right\}_{i=1}^{N}$ in two parts, namely 1) training data set $\mathcal{D}^1_{\text{train}} \coloneqq \{\boldsymbol{x}_i, y_i\}_{i=1}^{N}$ corresponding to $80\%$ of the original dataset and 2) the remaining dataset for the cross validation, i.e., test data set $\mathcal{D}^1_{\text{test}}$.
We utilize the training dataset $\mathcal{D}^1_{\text{train}}$ to compute the weights (4). Use these weights to cross validate the performance on the test dataset $\mathcal{D}^1_{\text{test}}$.
Figure 1 depicts the cost (3) in dB against the regularization parameter $\lambda$ in (3) for both the training and test dataset corresponding to *Communities and Crime* dataset. Evidently, for this dataset, the cross validation renders similar loss as in the training. Also, the reasonable $\lambda$ from grid search seems to be between $10^{-15}$ (really small but non-zero) and $2 \times 10^{-2}$.

## 1.3 Repeat 1.2) for *Individual household electric power consumption* dataset (N = 2075259, d = 9) and observe the scalability issue of the closed-form expression

Unlike in *Communities and Crime* dataset, we had to pre-process the potential outputs for the weights computation, namely $\text{submetering\_}k$ ($k \in \{1, 2, 3\}$ such that the output is between $-1$ and 1, otherwise the cost function of the logistic regression was rendering numerical issues (e.g., $\infty$ when the output $|y|$ was more than one).

Similar to previous dataset, we divided the *Individual household electric power consumption* dataset $\mathcal{D}^2 \coloneqq \left\{\boldsymbol{x}_i \in \mathbb{R}^d, y_i \in [-1, 1]\right\}_{i=1}^{N}$ in two parts, namely 1) training data set $\mathcal{D}^2_{\text{train}} \coloneqq \{\boldsymbol{x}_i, y_i\}_{i=1}^{N}$ corresponding to $80\%$ of the original dataset and 2) the remaining dataset for the cross validation, i.e., test data set $\mathcal{D}^2_{\text{test}}$.

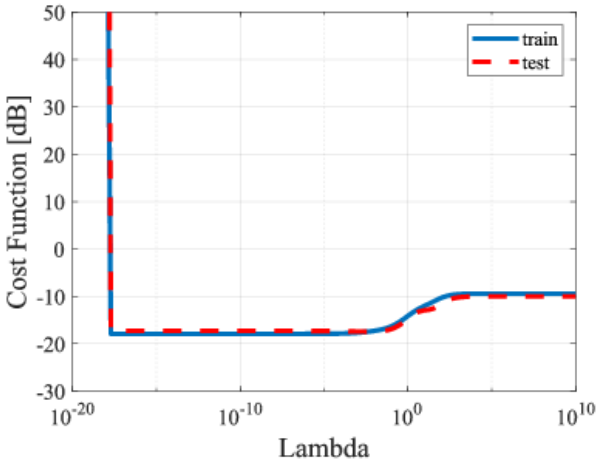


Figure 1: Loss versus $\lambda$ regularization parameter for the *Communities and Crime* dataset.

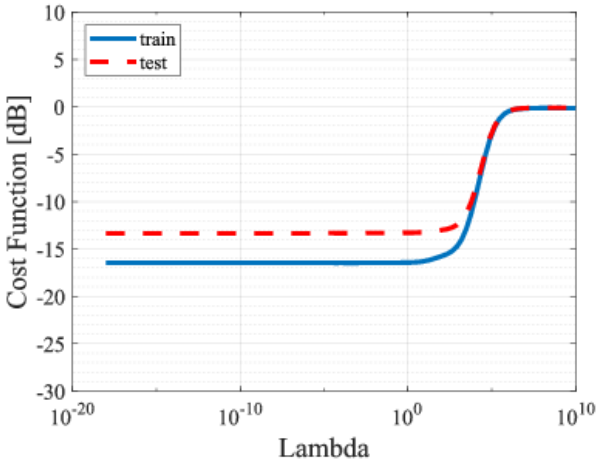


Figure 2: Loss versus $\lambda$ regularization parameter for the *Individual household electric power consumption* dataset.

Similarly, we utilized the training dataset $\mathcal{D}^2_{\text{train}}$ to obtain the weights (4). Thereby, used these weights to cross validate the performance on the test dataset $\mathcal{D}^2_{\text{test}}$.
Figure 2 illustrates the cost (3) against the regularization parameter $\lambda$ in (3) for both the training and test dataset corresponding to *Individual household electric power consumption* dataset. Evidently, for this dataset, the cross validation renders loss of around 3 dB notably when the $\lambda$ is less than 1.

## 1.4 How would you address even bigger datasets?

Although the size of the matrix to be inverted for the weights computation in (4) is $d \times d$, the major cost is in the computation of Gram matrix $\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}$ (assuming that $d \ll N$) that is $\mathcal{O}\left(dNd\right)$. If $d$ is really large than the major computational cost will be in the matrix inversion. If the batch based weights computation is a bottleneck, then one could employ online methods, e.g., GD, SGD, SAG, etc.

# 2 Computer Assignment 2

The logistic regression optimization (unconstrained) problem is given by

$$f(\boldsymbol{w}) = \frac{1}{N}\sum_{i=1}^{N} \underbrace{\log\left(1+\exp\left\{-y_i \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_i\right\}\right) + \lambda\|\boldsymbol{w}\|_2^2}_{:=f_i(\boldsymbol{w})}, \tag{5}$$

where the data set $\mathcal{D} = \{\boldsymbol{x}_i, y_i\}_{i=1}^N$, $y_i \in \mathbb{R}$, $\boldsymbol{w}, \boldsymbol{x}_i \in \mathbb{R}^N$, and $\lambda \in \mathbb{R}$.

Firstly, we compute some preliminaries, particularly, gradient and also the Hessian.

## Gradient

$$\frac{\partial f}{\partial \boldsymbol{w}} = \frac{1}{N}\sum_{i=1}^{N}\left(\frac{-y_i\boldsymbol{x}_i \exp\left\{-y_i\boldsymbol{w}^T\boldsymbol{x}_i\right\}}{1+\exp\left\{-y_i\boldsymbol{w}^T\boldsymbol{x}_i\right\}}\right) + 2\lambda\boldsymbol{w} \tag{6}$$

$$= \frac{1}{N}\sum_{i=1}^{N}\left(\frac{-y_i\boldsymbol{x}_i}{1+\exp\left\{+y_i\boldsymbol{w}^T\boldsymbol{x}_i\right\}}\right) + 2\lambda\boldsymbol{w} \tag{7}$$

$$\equiv \frac{-1}{N}\boldsymbol{X}^{\mathrm{T}}\left(\boldsymbol{I}_N + \boldsymbol{M}\right)^{-1}\boldsymbol{y} + 2\lambda\boldsymbol{w}\,, \tag{8}$$

where $\boldsymbol{X} \in \mathbb{R}^{N\times d}$ and $\boldsymbol{M} = \operatorname{diag}\left(\exp\left\{\boldsymbol{y}\circ(\boldsymbol{X}\boldsymbol{w})\right\}\right)$ with $\circ$ denotes Hadamard product.

## Hessian

$$\frac{\partial^2 f}{\partial \boldsymbol{w}^2} = \frac{1}{N}\boldsymbol{X}^{\mathrm{T}}\left(\boldsymbol{M}^2 + 2\boldsymbol{M} + \boldsymbol{I}_N\right)^{-1}\boldsymbol{Y}^2\boldsymbol{M}\boldsymbol{X} + 2\lambda\boldsymbol{I}_d\,, \tag{9}$$

where $\boldsymbol{Y} = \operatorname{diag}(y_1, \ldots, y_N)$.

The Hessian can be utilized to approximate the initial step-size, e.g., for GD.

## 2.1 Solve the optimization problem using GD, stochastic GD, SVRG, and SAG

Figure 3 illustrates the performance of all 4 algorithms with appropriate hyperparameters from the considered set of hyperparameters as given in Table 1. Since the step-size was fixed in this result, we observed that the GD and SGD were slowly decreasing with the increasing iterations. Also, SVRG converged much faster than the other methods but the convergence time was very slow due to inner iterations. However, SAG diverges after $958$ iterations– the reason is due to the fixed step-size.

## 2.2 Tune a bit hyper-parameters (including $\lambda$)

For the fixed hyperparameters scenario, we tried to perform coarse grid-search for the considered set of parameters given in Table 1 in order to find suitable fixed parameters. However, a better set of parameters can be found by performing fine grid-search, but at the expense of longer simulation time. Further, one could employ adaptive schemes, in particular for the step-size.

For different values of $\lambda$, we noticed an improvement in the convergence rate of the algorithms. However, for fixed step size we need to keep in mind that if we increase $\lambda$ we need to decrease the step size accordingly. This happens because as $\lambda$ increases the fixed step size to fulfil the smooth and strongly convex condition decreases.

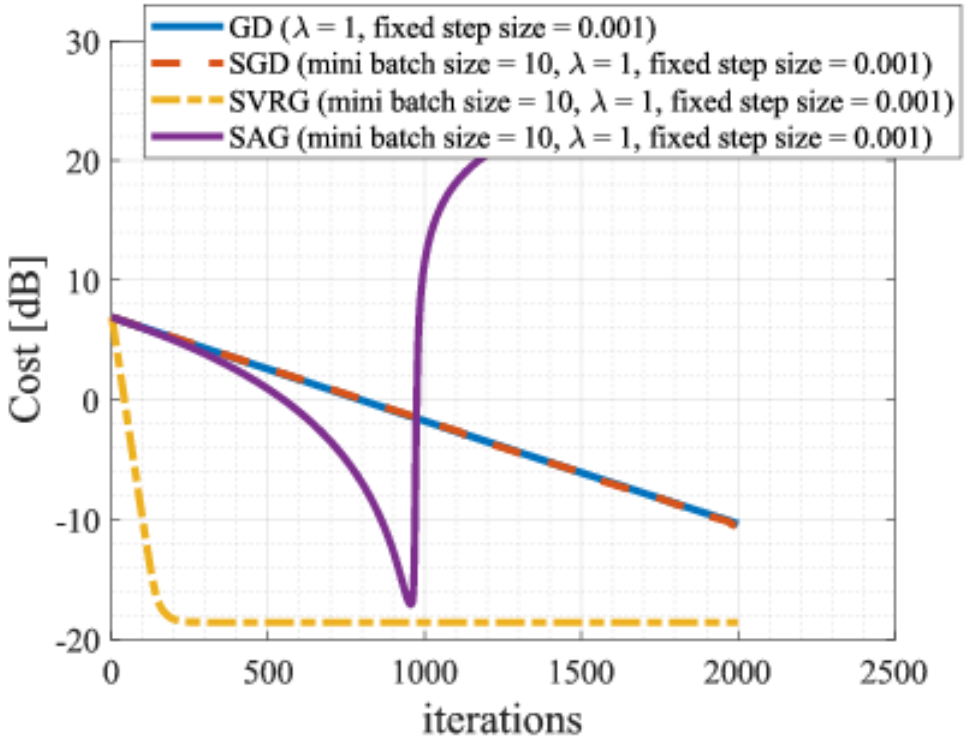


Figure 3: Cost [dB] versus iterations for various methods, namely GD, SGD, SVRG, and SAG, with fixed $\lambda$, step-size, batch-size, and inner iterations (valid for SVRG).

Table 1: Some Hyperparameters

| Method | Lambda $\lambda$ | step-size | Nr of Epochs (outer iterations) | Epoch Length (inner iterations) | Mini-batch-size |
|---|---|---|---|---|---|
| GD | $\{0, 10^{-4}, 10^{-2}, 1\}$ | fixed $\{10^{-5}, 10^{-4}, 10^{-3}\}$ | 2000 | – | – |
| SGD | $\{0, 10^{-4}, 10^{-2}, 1\}$ | fixed $\{10^{-5}, 10^{-4}, 10^{-3}\}$ | 2000 | – | 10 |
| SVRG | $\{0, 10^{-4}, 10^{-2}, 1\}$ | fixed $\{10^{-5}, 10^{-4}, 10^{-3}\}$ | 2000 | 20 | 10 |
| SAG | $\{0, 10^{-4}, 10^{-2}, 1\}$ | fixed $\{10^{-5}, 10^{-4}, 10^{-3}\}$ | 2000 | – | 10 |

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

We detail in Table 1 the hyper-parameters that we can tune for each algorithm. For a fixed step size, the convergence of GD and SGD is slow as we can see in Figure 3. However, the fixed step size does not impact much the other two algorithms, SVRG and SAG. We implemented the adaptive step size in the code and run some tests and the conclusion was that it speeds up the convergence of both algorithms.

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. From Table 2, we notice that SAG and SGD are the fastest algorithms, while GD and SVRG take more time than both. This behaviour is expected because SGD has cheaper iterations than all the others. For SVRG, recall that we run the inner loop with $T$ epochs, so we notice that running 20 epochs really increases the running time. Using small number of epochs such as 5, the convergence degrades and the running time improves, which is already expected because the convergence rate of SVRG is linear for sufficiently large $T$. For GD, we expect a high running time due to the need of evaluating and using the total batch size of the gradients.

Regarding the convergence rate, we include in Table 2 the theoretical convergence rate assuming that the function is smooth and strongly convex, and with $0 < \rho < 1$. For the GD, we have a linear rate of convergence while assuming both smooth and strongly convex (see Lecture 2, slide 19). For SGD, the convergence rate depends on the number of iterations $k$, and it is sublinear. Theoretically, SGD is slower

than GD but in our simulations they behave very similar in terms of convergence rate. For SVRG, the convergence rate depends on the number of epochs $T$, and for $T$ sufficiently high the convergence rate is linear [8]. From our simulations, $T$ is sufficiently high and SVRG converges much faster than all the others, including GD and SAG that also have linear rate. For SAG, the convergence rate is very similar to SVRG and also linear, but it has no epoch $T$ [7]. From our simulations, SAG converges faster than GD and SGD but it is still slower than SVRG.

Regarding the memory requirement, we include it in Table 2 along with the theoretical requirements based on gradient. Notice that SAG requires more memory than all the others, but only requires one gradient mini batch. The remaining algorithms require only one memory, but GD is the only one that requires the total gradient batch. Notice that although SAG is the one that requires more memory in theory, in practice it runs quickly and much faster than GD or SVRG.

Table 2: Complexity, convergence, and memory requirement Analysis

| **Method** | **Convergence time (2000 fixed iterations)** | **Convergence rate** | **Memory requirement** |
|---|---|---|---|
| GD | 490s | $\mathcal{O}\left(\rho^k \cdot\right)$ | 1 memory of 1 gradient vector |
| SGD | 60s | $\mathcal{O}\left(1/k \cdot\right)$ | 1 memory of 1 gradient batch vector |
| SVRG | 1600s | $\mathcal{O}\left(\rho^k \cdot\right)$ | 1 memory of 2 two gradient batch vector |
| SAG | 80s | $\mathcal{O}\left(\rho^k \cdot\right)$ | M memory of 1 two gradient batch vector |

# README: How to run the MATLAB script

For both computer assignments, we have main scripts $\mathrm{main\_ca1.m}$ and $\mathrm{main\_ca2.m}$. One could simply run them with default set of parameters.

# References

[1] Bubeck, Sbastien, ”Convex optimization: Algorithms and complexity,” Foundations and Trends in Machine Learning, vol. 8, no.3–4, pp. 231–357, 2015.

[2] L. Bottou, F. Curtis, J. Norcedal, ”Optimization Methods for Large-Scale Machine Learning,” SIAM Rev., 60, no. 2, pp. 223-311, 2018.

[3] T. Hastie, R. Tibshirani, and J. Friedman, ”The Elements of Statistical Learning: Data Mining, Inference and Prediction,” Second edition, Springer, 2009.

[4] C. D. Meyer, ”Matrix Analysis and Applied Linear Algebra,” SIAM, 2000.

[5] J. R. Magnus and H. Neudecker, ”Matrix differential calculus with applications in statistics and econometrics,” 2nd Edition, Wiley, UK, 1999.

[6] K. P. Murphy, ”Machine learning - a probabilistic perspective,” MIT Press, 2012.

[7] M. Schmidt,N. Le Roux, F. Bach, ”Minimizing finite sums with the stochastic average gradient,” Mathematical Programming, 2017.

[8] R. Johnson and T. Zhang, ”Accelerating stochastic gradient descent using predictive variance reduction,” NIPS, 2013.