IASD/MASH Project:
Optimization for Machine Learning [*]

Introduction

This course project aims at revisiting the course's notebooks by considering a more advanced method than gradient descent, namely L-BFGS, that was used in several notebooks to obtain a good estimate for the actual solution. It is also meant to go beyond the synthetic datasets used in the lab sessions.

Implementation guidelines

- All Python libraries are allowed, but the project should **not rely on existing implementations of optimization methods**.
- NumPy structures and numerical procedures from the `scipy` library (except optimization ones!) are recommended, but not mandatory.

Question guidelines

- A comparison between methods consists in reporting numerical results (final function values, convergence plots) given a budget (number of iterations, epochs,...) and commenting on them. *Is there a clear winner in the comparison? Are there results that are surprising to you?*
- When details of the implementation are left open, you should choose settings that allow for fast convergence, or that look informative to you. Your comments should reflect these choices.
- The goal of testing several values of an hyperparameter (e.g. a constant step size) is to assess the robustness of a given method with respect to this hyperparameter. *Are the results sensitive to changes in the value of the hyperparameter? Can you identify regimes of values that yield similar results (such as the large batch and mini-batch regimes for the batch size in stochastic gradient)?*

[*] Version of December 5, 2024. The last version of this document can be found at: **https://www.lamsade.dauphine.fr/∼croyer/ensdocs/OIM/ProjectOIM.pdf**.
Typos, questions, etc, can be sent to `clement.royer@lamsade.dauphine.fr`

Problem data

Question 1 *Select a training dataset for binary classification of the form* $\{(\boldsymbol{a}_i, y_i)\}_{i=1}^n$ *with* $\boldsymbol{a}_i \in \mathbb{R}^d$ *and* $y_i \in \{0,1\}$*. Comment briefly on this dataset: what are the problem dimensions, what does the correlation matrix look like, etc.*

Examples of such datasets can be downloaded from the libsvm repository:

https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/

Recommended datasets are `a9a`,`covtype.binary`,`ijcnn1`,`mushrooms`. More broadly, it is strongly advised to select a dataset with at least 20 features and 1000 data points.

Optimization problems

In this project, we consider a binary classification problem based on the dataset from Question 1, of the form

$$\operatorname*{minimize}_{\boldsymbol{x}\in\mathbb{R}^d} f(\boldsymbol{x}) := \frac{1}{n}\sum_{i=1}^n f_i(\boldsymbol{x}), \quad f_i(\boldsymbol{x}) := \left(y_i - \frac{1}{1+\exp(-\boldsymbol{a}_i^{\mathrm{T}}\boldsymbol{x})}\right)^2. \tag{1}$$

This problem is nonconvex in general. The function $t \mapsto \frac{1}{1+\exp(-t)}$ is called the sigmoid function. For any $i = 1, \dots, n$, the gradient of f_i is given by

$$\nabla f_i(\boldsymbol{x}) = -\frac{2\exp(\boldsymbol{a}_i^{\mathrm{T}}\boldsymbol{x})\left(\exp(\boldsymbol{a}_i^{\mathrm{T}}\boldsymbol{x})(y_i-1)+y_i\right)}{(1+\exp(\boldsymbol{a}_i^{\mathrm{T}}\boldsymbol{x}))^3}\boldsymbol{a}_i. \tag{2}$$

Question 2 *Given a data point* $(\boldsymbol{a}_i, y_i)$ *from your dataset, use the* `Autograd` *framework described in the second lab session to implement a code for the function* f_i *that enables to compute* $\nabla f_i(\boldsymbol{x})$ *for any* $\boldsymbol{x}$ *through automatic differentiation. Validate your implementation using the explicit formula* (2).

In the rest of the project, the use of `Autograd` is optional, and can be replaced by an explicit code for the derivative.

Regularized version We will also be interested in a regularized version of problem (1), of the form

$$\operatorname*{minimize}_{\boldsymbol{x}\in\mathbb{R}^d} f(\boldsymbol{x}) + \lambda\|\boldsymbol{x}\|_1, \tag{3}$$

where $\lambda > 0$ and $\|\boldsymbol{x}\|_1 = \sum_{j=1}^d |x_i|$.

First-order algorithms

Question 3 *Adapt the code of gradient descent provided during the lab sessions (or use your own implementation) to run it on problem* (1)*.*

- *What convergence rate is expected for gradient descent on this problem? Do you observe this rate empirically?*
- *Can you find a good constant value for the stepsize?*

Question 4 *Adapt the code of batch stochastic gradient provided during the third lab session (or use your own implementation) to compare gradient descent and stochastic gradient on problem* (1)*.*

- *Are your results consistent with what the theory predicts?*
- *Can you find a good constant stepsize choice for stochastic gradient? What appears to be the best value for the batch size on this problem?*

Question 5 *Adapt the code of Adagrad provided during the third lab session (or use your own implementation) to include that method in the comparison. How does this method compare to the best stochastic gradient method from Question 4 on problem* (1)*?*

Question 6 *Consider the best method from Question 5, and apply it to problem* (3) *using the proximal gradient approach from the fourth lab session. Can you find a value for λ that yields a good yet sparse solution vector?*

Quasi-Newton techniques

Question 7 *Implement the BFGS method, and compare its performance with that of gradient descent on problem* (1)*.*

Question 8 *Adapt the code from Question 7 so that the algorithm uses a stochastic gradient (obtained using a batch of indices) in lieu of the true gradient. Compare the resulting methods with their (batch) stochastic gradient counterparts.*

Question 9 *Compare your best stochastic BFGS variant with Adagrad using the following metrics:*

- *Number of accesses to data points.*
- *Number of d-dimensional vectors updated at every iteration.*

Question 10 *Generalize the approach used in Question 6 to BFGS techniques. How does the resulting methods compare to that used in Question 6 on the regularized problem* (3) *for a fixed value of λ?*

Question 11 *Implement the L-BFGS variant (see below) for $m = 0, 5, 10$, and compare it with the deterministic Question 7.*

Question 12 *Replace BFGS by L-BFGS in one of the questions 8–10, and answer the question.*

Remark: The (L-)BFGS codes are not meant to be identical or competitive to that of `scipy` or `PyTorch`, especially in terms of implementation features. Simplicity is advised!

Background: (L-)BFGS techniques

Consider a generic problem of the form

$$\operatorname*{minimize}_{\boldsymbol{x}\in\mathbb{R}^d} f(\boldsymbol{x}) = \frac{1}{n}\sum_{i=1}^{n} f_i, \tag{4}$$

where every f_i is a continuously differentiable function (hence $f \in \mathcal{C}^1$).

The k-th iteration of a **quasi-Newton method** for problem (4) has the form

$$\boldsymbol{x}_{k+1} \;=\; \boldsymbol{x}_k - \alpha_k \boldsymbol{H}_k \nabla f(\boldsymbol{x}_k), \tag{5}$$

where $\boldsymbol{H}_k$ is a symmetric, positive-definite matrix and $\alpha_k > 0$ is a given step size. The sequence of quasi-Newton matrices $\{\boldsymbol{H}_k\}_k$ is built so that every matrix is invertible, and the iteration (5) is well-defined. The most popular formula is the **BFGS** quasi-Newton update: starting from $\boldsymbol{x}_0$ and $\boldsymbol{H}_0 = \boldsymbol{I}_d$ (the identity matrix in $\mathbb{R}^{d\times d}$), at every iteration k, compute $\boldsymbol{x}_{k+1}$ according to (5), then define

$$\boldsymbol{s}_k = \boldsymbol{x}_{k+1} - \boldsymbol{x}_k \quad \text{and} \quad \boldsymbol{v}_k = \nabla f(\boldsymbol{x}_{k+1}) - \nabla f(\boldsymbol{x}_k).$$

Finally, update the quasi-Newton matrix as follows:

$$\boldsymbol{H}_{k+1} \;=\; \begin{cases} \left(\boldsymbol{I}_d - \frac{\boldsymbol{v}_k\boldsymbol{s}_k^{\mathrm{T}}}{\boldsymbol{s}_k^{\mathrm{T}}\boldsymbol{v}_k}\right)^{\mathrm{T}} \boldsymbol{H}_k \left(\boldsymbol{I}_d - \frac{\boldsymbol{v}_k\boldsymbol{s}_k^{\mathrm{T}}}{\boldsymbol{s}_k^{\mathrm{T}}\boldsymbol{v}_k}\right) + \frac{\boldsymbol{s}_k\boldsymbol{s}_k^{\mathrm{T}}}{\boldsymbol{s}_k^{\mathrm{T}}\boldsymbol{v}_k} & \text{if } \boldsymbol{s}_k^{\mathrm{T}}\boldsymbol{v}_k > 0 \\ \boldsymbol{H}_k & \text{otherwise.} \end{cases} \tag{6}$$

Limited-memory variants The standard, recursive formula for BFGS defines $\boldsymbol{H}_k$ as a function of $\boldsymbol{H}_0$ and the pairs $(\boldsymbol{s}_0, \boldsymbol{v}_0)$, $(\boldsymbol{s}_1, \boldsymbol{v}_1)$,...,$(\boldsymbol{s}_{k-1}, \boldsymbol{v}_{k-1})$. Such a matrix is likely to become dense as the iteration unfolds, while the information going back several iterations will become less and less relevant. The **limited-memory** variant of the BFGS quasi-Newton update computes $\boldsymbol{H}_k$ according to the most recent displacements, i.e. based on the latest m pairs $\{(\boldsymbol{s}_{k-\ell}, \boldsymbol{v}_{k-\ell})\}_{\ell=1}^{\min\{k,m\}}$[1] and a given initial matrix, which we will choose to be the identity for simplicity. An efficient implementation avoids forming the matrix $\boldsymbol{H}_k$ and instead decomposes the product $\boldsymbol{H}_k \nabla f(\boldsymbol{x}_k)$ into vector operations. The overall process for an iteration is detailed in Algorithm 1, with the assumption that iteration 0 is a classical gradient step.

Algorithm 1: L-BFGS iteration

```
1  **Inputs**: $k \geq 1$, $\boldsymbol{x}_k \in \mathbb{R}^d$, $m \in \mathbb{N}$, $\{\boldsymbol{s}_{k-\ell}, \boldsymbol{v}_{k-\ell}\}_{\ell=1}^{\min\{k,m\}}$.
2  Set $\boldsymbol{q}_k = \nabla f(\boldsymbol{x}_k)$.
3  **for** $\ell = 1, \dots, \min\{k, m\}$ **do**
4      **if** $\boldsymbol{s}_{k-\ell}^{\mathrm{T}} \boldsymbol{v}_{k-\ell} > 0$ **then**
5          $\gamma_\ell = \frac{\boldsymbol{s}_{k-\ell}^{\mathrm{T}} \boldsymbol{q}_k}{\boldsymbol{v}_{k-\ell}^{\mathrm{T}} \boldsymbol{s}_{k-\ell}}$
6          $\boldsymbol{q}_k \leftarrow \boldsymbol{q}_k - \gamma_\ell \boldsymbol{v}_{k-\ell}$
7      **end**
8  **end**
9  **for** $\ell = \min\{k, m\}, \dots, 1$ **do**
10     **if** $\boldsymbol{s}_{k-\ell}^{\mathrm{T}} \boldsymbol{v}_{k-\ell} > 0$ **then**
11         $\delta \leftarrow \frac{\boldsymbol{v}_{k-\ell}^{\mathrm{T}} \boldsymbol{q}_k}{\boldsymbol{v}_{k-\ell}^{\mathrm{T}} \boldsymbol{s}_{k-\ell}}$
12         $\boldsymbol{q}_k \leftarrow \boldsymbol{q}_k + (\gamma_\ell - \delta) \boldsymbol{s}_{k-\ell}$
13     **end**
14 **end**
15 Set $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \alpha_k \boldsymbol{q}_k$ with $\alpha_k > 0$.
16 Compute $\boldsymbol{s}_k = \boldsymbol{x}_{k+1} - \boldsymbol{x}_k$ and $\boldsymbol{v}_k = \nabla f(\boldsymbol{x}_{k+1}) - \nabla f(\boldsymbol{x}_k)$.
17 If $k > m$, discard $\{\boldsymbol{s}_{k-m}, \boldsymbol{v}_{k-m}\}$.
18 Add $\{\boldsymbol{s}_k, \boldsymbol{v}_k\}$ to the memory.
```

[1] For $k \leq m$, the memory consists of all pairs $(\boldsymbol{s}_{k-\ell}, \boldsymbol{v}_{k-\ell})$ available.