IASD Project: Optimization for Machine Learning [*]

Deadline: December 19, 2025 AOE.

Introduction

This course project aims at revisiting the course's notebooks by replacing the use of gradients and derivatives with *zeroth-order approximations*. Those bear resemblance with the finite difference techniques described in the course's second session, but we focus here and randomized techniques that provide good approximations in a probabilistic sense.

Guidelines

- Students may discuss the project with their classmates, but the submitted version must be worked out, written, and submitted individually.
- Any Python library can be used in the project as long as they do not provide direct answers to the questions. For instance, if a question consists in implementing Algorithm A, then students are expected to code A themselves instead of using a routine from an existing toolbox.
- NumPy structures and numerical procedures from the `scipy` library (except optimization ones!) are recommended, but not mandatory.
- 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.

[*] Version of December 5, 2025. The last version of this document can be found at: https://www.lamsade.dauphine.fr/∼croyer/ensdocs/OML/ProjectOML.pdf.
Typos, questions, etc, can be sent to `clement.royer@lamsade.dauphine.fr`

- 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)?*

An MNIST-based regression problem

The MNIST dataset [1] is one of the most classical datasets in machine learning. It consists of handwritten digits (from 0 to 9) given as pixel images in $\mathbb{R}^{28\times 28}$. There are numerous ways to obtain this dataset. An old-fashioned one would consist in downloading it from the libsvm repository:

https://www.csie.ntu.edu.tw/ cjlin/libsvmtools/datasets/multiclass.html#mnist

The dataset can also be directly imported in Python, e.g. through `PyTorch`.

Question 0 *Import the MNIST dataset and build your own dataset by making the following changes:*

i) Select two digit classes c_1 and c_2 such that $\{c_1, c_2\} \neq \{0, 1\}$. [1]

ii) Make sure the labels correspond to 0 for one class and 1 for the other class.

iii) Consider the images as vectors.

Given a subset of two classes in MNIST $\{(\boldsymbol{x}_i, y_i)\}_{i=1}^n$ with $\boldsymbol{x}_i \in \mathbb{R}^d$, $d = 28^2 = 784$ and $y_i \in \{0, +1\}$. we consider a classification problem of the form

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

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

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

Question 1 *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).*

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

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

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

[1] This requirement is merely to promote different answers among students. This still leaves you 44 digit pairs to choose from!

First-order algorithms

Question 2 *Adapt the code of gradient descent provided during the lab sessions (or use your own implementation) to run it on problem* (1).

i) What convergence rate is expected for gradient descent on this problem? Do you observe this rate empirically?

ii) Can you find a good constant value for the stepsize?

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

i) Are your results consistent with what the theory predicts?

ii) Can you find a good constant stepsize choice for stochastic gradient?

iii) What appears to be the best value for the batch size on this problem?

Question 4 *Adapt the code of Adagrad provided during the lab sessions (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 3 on problem* (1)*?*

Question 5 *Consider the best method from Question 4, and apply it to problem* (3) *using proximal gradient (consider the implementation used as illustration in class, or implement your own). Can you find a value for λ that yields a good yet sparse solution vector?*

Zeroth-order algorithms

In this final section, we consider optimization techniques that do not rely on exact gradient calculations (let alone differentiation). Instead, one computes steps by querying only function values. The classical iteration of such a method has the form

$$\boldsymbol{w}_{k+1} = \boldsymbol{w}_k - \alpha_k \frac{f(\boldsymbol{w}_k + h\boldsymbol{u}_k) - f(\boldsymbol{w}_k)}{h_k} \boldsymbol{u}_k, \tag{4}$$

where $\alpha_k > 0$ is a stepsize, $h > 0$ plays the role of a finite-difference parameter and $\boldsymbol{u}_k \sim \mathcal{N}(0, \boldsymbol{I}_d)$ is a Gaussian random vector. This approach was popularized by Nesterov [2][2] and algorithms of the form (4) are nowadays referred to as *zeroth-order optimization methods*.

Question 6 *Implement an algorithm based on iteration 4 and validate your implementation on the linear regression problem from the notebooks.*

i) Try different values for h in order to find one that leads to convergence.

ii) Try different choices for α_k to obtain the best possible performance.

[2]The paper remained a technical report and an updated version was published in 2017 with an additional co-author [3]. Understanding these papers is not necessary for this project.

It is worth comparing the proposed method with both exact gradient descent (using exact derivatives) and a finite-difference version of the method

$$\boldsymbol{w}_{k+1} = \boldsymbol{w}_k - \alpha_k \boldsymbol{g}_k, \qquad \boldsymbol{g}_k = \left[\frac{f(\boldsymbol{w}_k + h\boldsymbol{e}_j) - f(\boldsymbol{w}_k)}{h}\right]_{j=1,\dots,d}. \tag{5}$$

Question 7 *Compare the performance of your algorithm with that of gradient descent and the finite-difference method* (5) *on problem* (1).

i) How do the variants behave with constant stepsize? Decreasing stepsizes?

ii) Propose a unit of comparison (other than the number of iterations) for all three algorithms. Plot the behavior of the methods for a fixed budget of the cost you propose.

In a stochastic setting, there are two ways to implement iteration 4. The first considers that a batch of indices $\mathcal{S}_k$ is given at every iteration, and performs

$$\boldsymbol{w}_{k+1} = \boldsymbol{w}_k - \alpha_k \frac{f_{\mathcal{S}_k}(\boldsymbol{w}_k + h\boldsymbol{u}_k) - f_{\mathcal{S}_k}(\boldsymbol{w}_k)}{h_k}\boldsymbol{u}_k, \tag{6}$$

where

$$\forall \mathcal{S} \subset \{1,\dots,n\}^m, f_{\mathcal{S}}(\boldsymbol{w}) = \frac{1}{m}\sum_{i\in\mathcal{S}} f_i(\boldsymbol{w}).$$

The second approach considers that the same sample cannot be queried twice in a row, yielding the iteration

$$\boldsymbol{w}_{k+1} = \boldsymbol{w}_k - \alpha_k \frac{f_{\mathcal{S}_k^+}(\boldsymbol{w}_k + h\boldsymbol{u}_k) - f_{\mathcal{S}_k}(\boldsymbol{w}_k)}{h_k}\boldsymbol{u}_k, \tag{7}$$

where $\mathcal{S}_k$ and $\mathcal{S}_k^+$ are random batches of same size.

Question 8 *Adapt the code from the previous questions to allow for iterations* (6) *and* (7).

i) Compare the variants with batch size 1 *with the deterministic method* (4). *Do you recover observations from the first-order part of this project?*

ii) Assuming samples are drawn uniformly at random, find a good value for the batch size in both (6) *and* (7).

Question 9 *Propose an adaptation of your best stochastic zeroth-order variant to the proximal setting. Compare the resulting method with the algorithm from Question 5 on problem* (3) *with exact gradients and finite-difference gradients.*

References

[1] Y. Le Cun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proc. of the IEEE*, 86(11):2278–2324, 1998.

[2] Yu. Nesterov. Random gradient-free minimization of convex functions. Technical Report 2011/1, CORE, Université Catholique de Louvain, 2011.

[3] Yu. Nesterov and V. Spokoiny. Random gradient-free minimization of convex functions. *Found. Comput. Math.*, 17:527–566, 2017.