Optimization for Machine Learning

M2 IASD Apprentissage

Course project - 2023/2024

Dauphine | PSL
UNIVERSITÉ PARIS

- The last version of this document can be found at: **https://www.lamsade.dauphine.fr/~croyer/ensdocs/OAA/ProjOAA.pdf**.
- Typos, questions, etc, can be sent to `clement.royer@lamsade.dauphine.fr`.
- Current version: March 6, 2024.
- **Major updates to the document**
  - 2024.03.06: Fixed typo in equations (18)-(19).

Assignment

- This project follows the template of the course notebooks by mixing optimization questions with implementation tasks.
- Students may discuss the project with their classmates, but the submitted version must be worked out, written, and submitted individually.
- Students may submit their sources in either French or English.
- Students are expected to submit sources that include:
  - Their answers to the questions (in PDF or notebook format).
  - Their implementation of the proposed algorithms in Python (in .py files or in a notebook).
  - A Python script or a notebook to run the methods and reproduce the results.
- Please send your sources to `clement.royer@lamsade.dauphine.fr` under the form of a compressed folder. Those sources must include your first and last name.
- The deadline to send the sources is **March 10, 2024 AOE** (Anywhere On Earth).

Contents

Project: Stochastic Newton and quasi-Newton methods

Introduction

This project is concerned with going beyond first-order techniques in machine learning. Algorithmic frameworks such as gradient descent and stochastic gradient are inherently first-order methods, in that they rely solely on first-order derivatives. **Second-order methods**, on the other hand, make use of higher-order information, either explicitly or implicitly. Although those techniques are widely used in scientific computing, their use in machine learning has yet to be generalized. The goal of this project is to illustrate the performance of these techniques on learning problems involving both synthetic and real data.

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, especially for the linear algebra calculation (matrix inverse, eigenvalues) but not mandatory.

Question guidelines

- A comparison between methods consists in reporting numerical results (number of iterations, final function values, convergence plots) 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)?*

1 Second-order optimization methods

The purpose of this section is to present the basic Newton and quasi-Newton methods that this project is based upon. Those methods will be implemented and validated on small-dimensional toy problems of the generic form

$$\underset{\boldsymbol{w}\in\mathbb{R}^d}{\text{minimize}}\, f(\boldsymbol{w}), \tag{1}$$

where f is a twice continuously differentiable function (which we denote by $f \in \mathcal{C}^2$).

1.1 Newton's method

Newton's method is an optimization algorithm based on using the first- and second-order derivatives of the objective function. Starting from a point $\boldsymbol{w}_0 \in \mathbb{R}^d$, the method generates a sequence $\{\boldsymbol{w}_k\}_k$ from the following recursion:

$$\boldsymbol{w}_{k+1} = \boldsymbol{w}_k - \left[\nabla^2 f(\boldsymbol{w}_k)\right]^{-1} \nabla f(\boldsymbol{w}_k). \tag{2}$$

One immediately notices that iteration (2) is not well-defined, since the Hessian matrix of f at $\boldsymbol{w}_k$ may not be invertible. On the other hand, note that when the function is strongly convex, we have $\nabla^2 f(\boldsymbol{w}) \succ \mathbf{0}$ for any $\boldsymbol{w} \in \mathbb{R}^d$, and the Newton iteration is well defined. We will come back to this issue in Section 1.2.

One of the main characteristics of Newton's method is its fast local convergence rate guarantees. When f is a strongly convex quadratic, Newton's method converges in one iteration. If f is strongly convex, it can be shown that Newton's method converges at a local quadratic rate, i.e. if $\boldsymbol{w}_0$ is close enough to the global minimum $\boldsymbol{w}^*$ of f, we have:

$$\|\boldsymbol{w}_{k+1} - \boldsymbol{w}^*\| \leq C \|\boldsymbol{w}_k - \boldsymbol{w}^*\|^2,$$

where $C > 0$. By comparison, with a standard gradient descent approach, one would obtain a result of the form $\|\boldsymbol{w}_{k+1} - \boldsymbol{w}^*\| \leq C \|\boldsymbol{w}_k - \boldsymbol{w}^*\|$, i. e. a linear rate of convergence.

Implementation 1.1 *Implement Newton's method in its basic form (2). Your code should take f, ∇f and $\nabla^2 f$ as inputs.*

Question 1 *Suppose that we apply Newton's method to the quadratic problem:*

$$\underset{\boldsymbol{w}\in\mathbb{R}^3}{\text{minimize}}\; q(\boldsymbol{w}) := 2(w_1 + w_2 + w_3 - 3)^2 + (w_1 - w_2)^2 + (w_2 - w_3)^2. \tag{3}$$

This problem is a strongly convex quadratic function, with a unique minimizer $\boldsymbol{w}^* = \begin{bmatrix}1\\1\\1\end{bmatrix}$.

a) *Write down the first iteration of Newton's method for problem (3), and show that Newton's method indeed converges in one iteration to the solution.*

b) *Run the method starting from the origin and two other starting points of your choice. Do you indeed observe convergence in one iteration?*

Question 2 *We now consider the celebrated Rosenbrock function*

$$\underset{\boldsymbol{w}\in\mathbb{R}^2}{\text{minimize}}\; 100(w_2 - w_1^2)^2 + (1 - w_1)^2. \tag{4}$$

a) *Apply your implementation of Newton's method to this problem starting from*

$$\boldsymbol{w}_{01} := \begin{bmatrix}-1.2\\1\end{bmatrix}, \quad \textit{and} \quad \boldsymbol{w}_{02} := \begin{bmatrix}0\\\frac{1}{200} + 10^{-12}\end{bmatrix}.$$

Report your results and your observations. Does the method converge?

b) *Could we run the method starting from the point $\boldsymbol{w}_{03} = \begin{bmatrix}0\\0.005\end{bmatrix}$? How does that illustrate that Newton's method is local in nature?*

1.2 A globally convergent version of Newton's method

Newton's method comes with strong local guarantees, but not global ones. For general nonconvex functions (or even convex functions that are not strongly convex), the choice of the initial point determines whether or not the method will converge, or be well-defined. *Globalization techniques* were developed to guarantee that Newton's method would converge independently of its starting point, and regardless of the convexity of the problem. In this project, we will focus on one globalization technique based on line search.

Suppose that we are at iteration k of the method: in order to apply a Newton-like iteration, one can always select a nonnegative value λ_k such that $\nabla^2 f(\boldsymbol{w}_k) + \lambda_k \boldsymbol{I}_d \succ 0$ and compute a Newton-type direction as

$$\boldsymbol{d}_k = -[\nabla^2 f(\boldsymbol{w}_k) + \lambda_k \boldsymbol{I}]^{-1} \nabla f(\boldsymbol{w}_k). \tag{5}$$

A common choice for λ_k is

$$\lambda_k = 2 \max\left\{-\lambda_{\min}(\nabla^2 f(\boldsymbol{w}_k)), 10^{-10}\right\}, \tag{6}$$

where $\lambda_{\min}(\boldsymbol{A})$ stands for the minimum eigenvalue of the matrix $\boldsymbol{A}$, and the small positive constant 10^{-10} handles the singular case.

Once this direction has been defined, a line search is performed to compute a suitable stepsize along that direction. In this project, we will consider the basic **Armijo line-search** defined below.

Definition 1.1 *Consider the function f of problem (1). Given a vector $\boldsymbol{w}_k$ and a search direction $\boldsymbol{d}_k$, the* **Armijo line-search process** *consists in computing a stepsize $\alpha_k = \theta^{j_k}$, where $\theta \in (0,1)$ and j_k is the smallest nonnegative integer such that*

$$f(\boldsymbol{w}_k + \alpha_k \boldsymbol{d}_k) < f(\boldsymbol{w}_k) + c\alpha_k \boldsymbol{d}_k^{\mathrm{T}} \nabla f(\boldsymbol{w}_k), \tag{7}$$

with $c \in \left(0, \frac{1}{2}\right)$.

Under classical assumptions, it is possible to show convergence of this framework regardless of the starting point: this method is thus *globally convergent*. Under some additional conditions, it is also possible to obtain local convergence results, that are usually worse than that of Newton's method because of the use of λ_k. Still, line-search methods are widely used in nonlinear optimization, and Newton-type directions turn out to be less sensitive to the choice of stepsize than gradient-based methods.

Implementation 1.2 *Implement the globalized version of Newton's method using the direction choice (5)–(6) and the Armijo line-search procedure from Definition 1.1.*

Question 3 *Apply Newton's method with line search to the problem (4) using $c = 0.0001$, $\theta = 0.5$, and the two initial points mentioned in Question 2. Compare your results with those obtained for the basic Newton iteration.*

Question 4 *Try out a few values for c and θ (report the results of your tests). How sensitive does the method appear to be to these values?*

1.3 Quasi-Newton methods and BFGS

Second-order methods are often deemed too expensive because of their need to evaluate the Hessian matrix: this issue was addressed in the 1950s through the development of **quasi-Newton methods**, that form one of the most powerful techniques in smooth optimization.

For the generic problem (1), the k-th iteration of a quasi-Newton method typically has the following form:

$$\boldsymbol{w}_{k+1} \;=\; \boldsymbol{w}_k - \alpha_k \boldsymbol{H}_k \nabla f(\boldsymbol{w}_k), \tag{8}$$

where $\boldsymbol{H}_k$ is a symmetric, positive-definite matrix such that $\boldsymbol{H}_k^{-1} \approx \nabla^2 f(\boldsymbol{w}_k)$, and $\alpha_k > 0$ is a stepsize typically computed through a line search. The sequence of quasi-Newton matrices $\{\boldsymbol{H}_k\}_k$ is built so that every matrix is invertible, and the iteration (8) is well-defined.

Several formulas for defining $\{\boldsymbol{H}_k\}_k$ have been proposed in the literature. We focus here on the most popular of all, named BFGS after the four researchers that (independently) designed it (Broyden, Fletcher, Goldfarb and Shanno).

Definition 1.2 (BFGS Quasi-Newton update) *Start with $\boldsymbol{w}_0$ and $\boldsymbol{H}_0 = \boldsymbol{I}_d$ (the identity matrix in $\mathbb{R}^{d\times d}$). At every iteration k, compute $\boldsymbol{w}_{k+1}$ according to (8), then define*

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

Finally, update the quasi-Newton matrix as follows:

$$\boldsymbol{H}_{k+1} = \begin{cases} \left(\boldsymbol{I}_d - \frac{\boldsymbol{v}_k \boldsymbol{s}_k^{\mathrm{T}}}{\boldsymbol{s}_k^{\mathrm{T}} \boldsymbol{v}_k}\right)^{\mathrm{T}} \boldsymbol{H}_k \left(\boldsymbol{I}_d - \frac{\boldsymbol{v}_k \boldsymbol{s}_k^{\mathrm{T}}}{\boldsymbol{s}_k^{\mathrm{T}} \boldsymbol{v}_k}\right) + \frac{\boldsymbol{s}_k \boldsymbol{s}_k^{\mathrm{T}}}{\boldsymbol{s}_k^{\mathrm{T}} \boldsymbol{v}_k} & \text{if } \boldsymbol{s}_k^{\mathrm{T}} \boldsymbol{v}_k > 0 \\ \boldsymbol{H}_k & \text{otherwise.} \end{cases} \tag{9}$$

The update formula (9) not only guarantees that $\boldsymbol{H}_{k+1} \succ \boldsymbol{0}$, but it also ensures that $\boldsymbol{H}_{k+1}^{-1} \boldsymbol{s}_k = \boldsymbol{v}_k$. This latter property means that $\boldsymbol{H}_{k+1}^{-1}$ is consistent with $\nabla^2 f(\boldsymbol{w}_k)$ in the direction of the step $\boldsymbol{w}_{k+1} - \boldsymbol{w}_k$.

In the case of a strongly convex function f, it can be shown that quasi-Newton methods such as BFGS possess a *local superlinear rate*. That is, for $\boldsymbol{w}_0$ sufficiently close to the optimum $\boldsymbol{w}^*$, the iterates satisfy the relation

$$\|\boldsymbol{w}_{k+1} - \boldsymbol{w}^*\| \leq C \|\boldsymbol{w}_k - \boldsymbol{w}^*\|^{1+t},$$

where $t \in (0, 1)$, $C > 0$ and $\boldsymbol{w}^*$ is the global minimum of the function. One thus sees that the convergence guarantees are not as good as those of Newton's method: however, they are still (locally) better than that of gradient descent, which is remarkable as a quasi-Newton approach never accesses the Hessian matrix.

Implementation 1.3 *Implement a quasi-Newton method with the BFGS update (9) and Armijo line search (7).*

Question 5 *Compare the quasi-Newton method from Implementation 1.3 and the line-search Newton method from Implementation 1.1 on problems (3) and (4). Use the same starting points and the same values for c and θ in the comparison. Illustrate the comparison in terms of*

- *Iterations;*
- *Gradient and Hessian evaluations, assuming the cost of a Hessian matrix is d times that of a gradient vector;*
- *Function evaluations.*

1.4 Limited-memory BFGS

The standard, recursive formula for BFGS defines $\boldsymbol{H}_{k+1}$ as a function of $\boldsymbol{H}_0$ and the pairs $(\boldsymbol{s}_0, \boldsymbol{v}_0)$, $(\boldsymbol{s}_1, \boldsymbol{v}_1), \ldots, (\boldsymbol{s}_k, \boldsymbol{v}_k)$. 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 $\boldsymbol{H}_{k+1}$ according to the most recent displacements, by computing $\boldsymbol{H}_{k+1}$ based on the latest m pairs $\{(\boldsymbol{s}_{k-i}, \boldsymbol{v}_{k-i})\}_{i=0}^{\max\{0,m-1\}}$ (and a given matrix $\boldsymbol{H}_0$, which we will choose to be the identity). This process is detailed in Algorithm 1.

Implementation 1.4 *Implement a quasi-Newton method with the L-BFGS update described by Algorithm 1 and Armijo line search (7).*

Question 6 *Run BFGS and three variants of L-BFGS respectively using $m = 0$, $m = 1$ and $m = 5$ on the Rosenbrock function (4), and compare the results. What metric could show the computational interest of limited memory quasi-Newton? Do you observe this improvement here?*

Algorithm 1: L-BFGS update

Inputs: $k \in \mathbb{N}$, $\boldsymbol{w}_k, \boldsymbol{w}_{k+1} \in \mathbb{R}^d$, $m \in \mathbb{N}$, $\{\boldsymbol{s}_{k-i}, \boldsymbol{v}_{k-i}\}_{i=1}^{\max\{0,m-1\}}$.
Compute $\boldsymbol{s}_k = \boldsymbol{w}_{k+1} - \boldsymbol{w}_k$ and $\boldsymbol{v}_k = \nabla f(\boldsymbol{w}_{k+1}) - \nabla f(\boldsymbol{w}_k)$.
Set $\boldsymbol{H}_{k+1} = \boldsymbol{I}_d$.
for $i = \max\{0, m-1\}, \ldots, 0$ **do**
    If $\boldsymbol{s}_{k-i}^{\mathrm{T}} \boldsymbol{v}_{k-i} > 0$, set

$$\boldsymbol{H}_{k+1} = \left(\boldsymbol{I}_d - \frac{\boldsymbol{v}_{k-i}\boldsymbol{s}_{k-i}^{\mathrm{T}}}{\boldsymbol{s}_{k-i}^{\mathrm{T}}\boldsymbol{v}_{k-i}}\right)^{\mathrm{T}} \boldsymbol{H}_{k+1} \left(\boldsymbol{I}_d - \frac{\boldsymbol{v}_{k-i}\boldsymbol{s}_{k-i}^{\mathrm{T}}}{\boldsymbol{s}_{k-i}^{\mathrm{T}}\boldsymbol{v}_{k-i}}\right) + \frac{\boldsymbol{s}_{k-i}\boldsymbol{s}_{k-i}^{\mathrm{T}}}{\boldsymbol{s}_{k-i}^{\mathrm{T}}\boldsymbol{v}_{k-i}}.$$

end

2 Stochastic second-order methods

In this section, we describe practical variants of second-order methods that have been tailored to machine learning formulations. For the purpose of this project, we will follow the lectures and focus on finite-sum problems of the form

$$\underset{\boldsymbol{w}\in\mathbb{R}^d}{\text{minimize}}\, f(\boldsymbol{w}) := \frac{1}{n}\sum_{i=1}^{n} f_i(\boldsymbol{w}), \tag{10}$$

where every f_i is $\mathcal{C}^2$ and depends on one data point. Our canonical example will be the logistic regression problem used in the course notebooks, which is strongly convex.

Exemple 2.1 *Given a dataset $\{(\boldsymbol{x}_i, y_i)\}_{i=1}^n$ with $\boldsymbol{x}_i \in \mathbb{R}^d$ and $y_i \in \{-1, 1\}$, we consider the problem*

$$\underset{\boldsymbol{w}\in\mathbb{R}^d}{\textit{minimize}}\, f(\boldsymbol{w}) := \frac{1}{n}\sum_{i=1}^{n} f_i(\boldsymbol{w}), \qquad f_i(\boldsymbol{w}) = \ln(1+\exp(-y_i \boldsymbol{x}_i^T \boldsymbol{w})) + \frac{\lambda}{2}\|\boldsymbol{w}\|^2. \tag{11}$$

Every function f_i is $\mathcal{C}^2$, and we have

$$\nabla f_i(\boldsymbol{w}) = -\frac{y_i}{1+\exp(y_i \boldsymbol{x}_i^T \boldsymbol{w})}\boldsymbol{x}_i + \lambda \boldsymbol{w}. \tag{12}$$

as well as

$$\nabla^2 f_i(\boldsymbol{w}) := \frac{\exp(y_i \boldsymbol{w}^{\mathrm{T}} \boldsymbol{x}_i)}{(1+\exp(y_i \boldsymbol{w}^{\mathrm{T}} \boldsymbol{x}_i))^2}\boldsymbol{x}_i \boldsymbol{x}_i^{\mathrm{T}} + \lambda \boldsymbol{I}_d. \tag{13}$$

2.1 Subsampling Newton methods

In subsampling Newton methods, we construct stochastic gradient and Hessian estimates from batches of data points, akin to a batch stochastic gradient approach. At every iteration k of the algorithm, given the current iterate $\boldsymbol{w}_k$, one computes subsampling derivatives as

$$\nabla f_{\mathcal{S}_k}(\boldsymbol{w}_k) = \frac{1}{|\mathcal{S}_k|}\sum_{i\in\mathcal{S}_k} \nabla f_i(\boldsymbol{w}_k), \quad \nabla^2 f_{\mathcal{S}_k^H}(\boldsymbol{w}_k) = \frac{1}{|\mathcal{S}_k^H|}\sum_{i\in\mathcal{S}_k^H} \nabla^2 f_i(\boldsymbol{w}_k), \tag{14}$$

where $\mathcal{S}_k$ and $\mathcal{S}_k^H$ are sets of random indices drawn in $\{1, \ldots, n\}$. The subsampling iteration then reads

$$\boldsymbol{w}_{k+1} = \boldsymbol{w}_k - \alpha_k \boldsymbol{d}_k, \quad \boldsymbol{d}_k = \left[\nabla^2 f_{\mathcal{S}_k^H}(\boldsymbol{w}_k)\right]^{-1} \nabla f_{\mathcal{S}_k}(\boldsymbol{w}_k), \tag{15}$$

where $\alpha_k > 0$ is a positive stepsize, that can either be chosen as constant or through a line search. For the latter, we must adapt the line-search condition to avoid using the entire dataset at every line-search iteration. In the case of the Armijo line search described in Definition 1.1, a natural choice consists in using the same sample for the gradient and the line search.

Definition 2.1 (Subsampling Armijo line search) *Consider the finite-sum optimization problem (10). Given a vector $\boldsymbol{w}_k$, a search direction $\boldsymbol{d}_k$ and a set $\mathcal{S}_k$ of random indices in $\{1, \dots, n\}$, the* **subsampling Armijo line-search process** *consists in computing a stepsize $\alpha_k = \theta^{j_k}$, where $\theta \in (0, 1)$ and j_k is the smallest nonnegative integer such that*

$$f_{\mathcal{S}_k}(\boldsymbol{w}_k + \alpha_k \boldsymbol{d}_k) < f_{\mathcal{S}_k}(\boldsymbol{w}_k) + c\alpha_k \boldsymbol{d}_k^{\mathrm{T}} \nabla f_{\mathcal{S}_k}(\boldsymbol{w}_k), \tag{16}$$

for some $c \in \left(0, \frac{1}{2}\right)$.

Implementation 2.1 *Implement a subsampling Newton method with the following requirements:*

- *The method should take $|\mathcal{S}_k|$ and $|\mathcal{S}_k^H|$ as inputs (for simplicity, we only consider constant sample sizes).*
- *The method should be able to use two different stepsize approaches:*
  - *Constant stepsize $\alpha_k = \alpha > 0$ (provided by the user);*
  - *Stepsize computed via the subsampling line-search technique described in Definition 2.1.*

Question 7 *Using the same (synthetic) dataset than in lab 4 of the course (on stochastic gradient methods), compare the subsampling Newton method with the standard Newton method with the following settings:*

- *Use the appropriate version of Armijo line search for each method (use the same c and θ);*
- *Use different sizes for $|\mathcal{S}_k|$ and $|\mathcal{S}_k^H|$, including the configuration $|\mathcal{S}_k| = |\mathcal{S}_k^H| = 1$.*

Use the notion of epoch for the comparison.

Question 8 *Fix sample sizes for the gradient and Hessian. We consider two possible stepsize choices*

a) $\alpha_k = \frac{\bar{\alpha}}{L}$, *where* $L = \frac{4\|\boldsymbol{X}\boldsymbol{X}^{\mathrm{T}}\|}{n} + \lambda$ *is the Lipschitz constant of* ∇f;

b) $\alpha_k = \frac{\bar{\alpha}}{L_{\mathcal{S}_k}}$, *where*

$$L_{\mathcal{S}_k} = \frac{4\left\|\sum_{i \in \mathcal{S}_k} \boldsymbol{x}_i \boldsymbol{x}_i^{\mathrm{T}}\right\|}{|\mathcal{S}_k|} + \lambda$$

is the Lipschitz constant of $\nabla f_{\mathcal{S}_k}$.

Compare the two choices for different values of $\bar{\alpha}$ and a fixed number of epochs.

2.2 Stochastic quasi-Newton methods

The implementation of stochastic quasi-Newton methods easily follows from the material above, and is closer in spirit to that of stochastic gradient methods. At iteration k, the gradient vectors $\nabla f(\boldsymbol{w}_{k+1})$ and $\nabla f(\boldsymbol{w}_k)$ used in Definition 1.2 are replaced by $\nabla f_{\mathcal{S}_k}(\boldsymbol{w}_k)$ and $\nabla f_{\mathcal{S}_k}(\boldsymbol{w}_{k+1})$, respectively.

Implementation 2.2 *Implement stochastic variants of BFGS and L-BFGS using the modified Armijo line search described in Definition 2.1, with the following requirements:*

- *Both methods should take $|\mathcal{S}_k|$ as an input;*
- *The same stepsize strategies than in Implementation 2.1 should be included.*

Question 9 *Using the same (synthetic) dataset for logistic regression than in Question 7, compare the stochastic BFGS and L-BFGS methods with their counterparts from Section 1, using the appropriate Armijo line-search technique for each method and a fixed epoch budget. Use different values for the sample size $|\mathcal{S}_k|$, including $|\mathcal{S}_k| = 1$.*

3 Binary classification on real-world data

In this final part, we will apply our techniques to a classification problem based on real-world data.

3.1 Dataset

We consider data under the form $\{(\boldsymbol{x}_i, y_i)\}_{i=1}^n$ with $\boldsymbol{x}_i \in \mathbb{R}^d$ and $y_i \in \{0, 1\}$. To this end, we will rely on datasets from the libsvm repository, that can be downloaded from

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

Recommended datasets are `a9a`,`covtype.binary`,`ijcnn1`,`mushrooms`. To load the dataset in Python, students may use the routine

`sklearn.datasets.load_svmlight_file`

from the scikit-learn library.

Remark 3.1 *For the purpose of this project, we do not require to split the data between a training set and a testing set. If both sets appear in the data, the testing set may be ignored for this section.*

Implementation 3.1 *Select a dataset from the libsvm repository. The dataset should have at least 20 features and 1,000 training samples.*

3.2 Optimization problem

Given our dataset, we formulate the following finite-sum optimization problem

$$\underset{\boldsymbol{w}\in\mathbb{R}^d}{\text{minimize}}\, g(\boldsymbol{w}) := \frac{1}{n}\sum_{i=1}^n g_i(\boldsymbol{w}), \quad g_i(\boldsymbol{w}) := \left(y_i - \frac{1}{1+\exp(-\boldsymbol{x}_i^{\mathrm{T}}\boldsymbol{w})}\right)^2. \tag{17}$$

This problem is nonconvex in general. The function $t \mapsto \frac{1}{1+\exp(-t)}$ is called the sigmoid function. For any $i = 1, \ldots, n$, the function g_i is $\mathcal{C}^2$: its derivatives are given by

$$\nabla g_i(\boldsymbol{w}) = -\frac{2\exp(\boldsymbol{x}_i^{\mathrm{T}}\boldsymbol{w})\left(\exp(\boldsymbol{x}_i^{\mathrm{T}}\boldsymbol{w})(y_i-1)+y_i\right)}{(1+\exp(\boldsymbol{x}_i^{\mathrm{T}}\boldsymbol{w}))^3}\boldsymbol{x}_i \tag{18}$$

and

$$\nabla^2 g_i(\boldsymbol{w}) = \frac{2\exp(\boldsymbol{x}_i^{\mathrm{T}}\boldsymbol{w})\left(\exp(2\boldsymbol{x}_i^{\mathrm{T}}\boldsymbol{w})(y_i-1)+2\exp(\boldsymbol{x}_i^{\mathrm{T}}\boldsymbol{w})-y_i\right)}{(1+\exp(\boldsymbol{x}_i^{\mathrm{T}}\boldsymbol{w}))^4}\boldsymbol{x}_i\boldsymbol{x}_i^{\mathrm{T}}. \tag{19}$$

Implementation 3.2 *Given your dataset, implement the associated codes for $g_{\mathcal{S}}$, $\nabla g_{\mathcal{S}}$ and $\nabla^2 g_{\mathcal{S}}$, where $\mathcal{S}$ is a set of random indices in $\{1, \ldots, n\}$.*

3.3 Comparison of the algorithms

Our goal is now to find a suitable method for this problem.

Question 10 *Compare the performance of the subsampling Newton method from Section 2.1 and that of the stochastic quasi-Newton method of Section 2.2 with a (batch) stochastic gradient approach. You may reuse the stochastic gradient implementation from the course notebooks.*

Extra credit (optional): Choose a dataset with a testing set and compare the testing loss versus the training loss.