EE-556: Mathematics of Data: From Theory to Computation
Laboratory for Information and Inference Systems
Fall 2019

Instructor:
Prof. Volkan Cevher

Head TA:
Thomas Sanchez

Homework 3 (for Weeks 9-11)

1 Multiclass classification - 70 Points

In Lecture 2 (slide 19) you were introduced binary logistic regression [3] and learned how to derive the expression of its maximum likelihood estimator. We briefly remind you of the procedure and then guide you through extending these notions to multiple classes.

We are working with data represented by pairs $\left(\mathbf{a}_i^T, b_i\right)$, where $\mathbf{a}_i^T$ is the i^{th} row of matrix $\mathbf{A} \in \mathbb{R}^{n\times d}$ and b_i is the i^{th} entry of $\mathbf{b} \in \mathbb{R}^n$. Each b_i denotes the class label associated to sample $\mathbf{a}_i^T$ and we have a total of $C = 2$ classes, which we encode as $\{-1, 1\}$. We denote by n the number of data samples and by d the problem dimension. Lastly, we aim to model our data with a feature vector $\mathbf{x} \in \mathbb{R}^d$. For clarity, refer to the visual representations below.

$$\mathbf{A} = \begin{bmatrix} \leftarrow & \mathbf{a}_1^T & \rightarrow \\ \leftarrow & \mathbf{a}_2^T & \rightarrow \\ & \dots & \\ \leftarrow & \mathbf{a}_n^T & \rightarrow \end{bmatrix} \in \mathbb{R}^{n\times d} \qquad \mathbf{b} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix} \in \{-1,1\}^n \qquad \mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_d \end{bmatrix} \in \mathbb{R}^d \tag{1}$$

The logistic model assumes there is a linear relationship between $\mathbf{x}$ and the logarithm of the odds of $b_i = 1$ [1], given $\mathbf{a}_i$. Formally, we can write this as

$$\log \underbrace{\frac{\mathbb{P}(b_i = 1 \mid \mathbf{a}_i, \mathbf{x})}{\mathbb{P}(b_i = -1 \mid \mathbf{a}_i, \mathbf{x})}}_{\text{Odds of } b_i=1 \text{ to } b_i=-1.} = \mathbf{a}_i^T\mathbf{x} \overset{C=2}{\iff} \underbrace{\log \frac{\mathbb{P}(b_i = 1 \mid \mathbf{a}_i, \mathbf{x})}{1 - \mathbb{P}(b_i = 1 \mid \mathbf{a}_i, \mathbf{x})}}_{\substack{\text{This quantity is called a } \textbf{logit}, \\ \text{hence the name } \textbf{logistic regression}.}} = \mathbf{a}_i^T\mathbf{x} \iff \mathbb{P}(b_i = 1 \mid \mathbf{a}_i, \mathbf{x}) = \underbrace{\frac{e^{\mathbf{a}_i^T\mathbf{x}}}{1 + e^{\mathbf{a}_i^T\mathbf{x}}}}_{\substack{\text{The } \textbf{logistic} \\ \text{function.}}}$$

Assuming the data samples are statistically independent, we can write down the probability of observing vector $\mathbf{b}$ as:

$$\mathbb{P}(\mathbf{b} \mid \mathbf{A}, \mathbf{x}) = \mathbb{P}(b_1 \wedge b_2 \wedge \ldots \wedge b_n \mid \mathbf{A}, \mathbf{x}) = \prod_{i=1}^{n} \mathbb{P}(b_i \mid \mathbf{a}_i, \mathbf{x}) \tag{2}$$

Expression (2) is called the likelihood of observing $\mathbf{b}$, given data A and parameters $\mathbf{x}$. The goal of logistic regression is to find a feature vector $\hat{\mathbf{x}}_{\text{ML}}$ that maximizes the likelihood of observing $\mathbf{b}$.

We can find $\hat{\mathbf{x}}_{\text{ML}}$ using the numerical optimization methods you have seen in the lectures. The objective function is chosen as the negative logarithm of the likelihood [2], as minimizing it is equivalent to maximizing $\mathbb{P}(\mathbf{b} \mid \mathbf{A}, \mathbf{x})$. The expression of the maximum likelihood estimator is therefore given by:

$$\hat{\mathbf{x}}_{\text{ML}} \in \arg\min_{\mathbf{x}} \{-\log \mathbb{P}(\mathbf{b} \mid \mathbf{A}, \mathbf{x}) = \sum_{i=1}^{n} \log\left[1 + \exp\left(-b_i \mathbf{a}_i^T \mathbf{x}\right)\right] : \mathbf{x} \in \mathbb{R}^d\}.$$

We now extend the binary setting to $C > 2$ classes, which naturally leads to having correspondingly many feature vectors. Note that one may use the same technique as is the binary case, by setting one of the classes as the pivot (for example C) and writing the log-odds against it. This effectively leaves us with $C - 1$ feature vectors because we fix $\mathbf{x}_C$ by construction. We will take a different approach that is (arguably) more natural and easier to implement in PART II of this exercise. This approach uses C feature vectors represented as a matrix $\mathbf{X} \in \mathbb{R}^{d\times C}$. For clarity, refer to the visual representation below.

$$\mathbf{X} = \begin{bmatrix} \uparrow & \uparrow & & \uparrow \\ \mathbf{x}_1 & \mathbf{x}_2 & \ldots & \mathbf{x}_C \\ \downarrow & \downarrow & & \downarrow \end{bmatrix} \in \mathbb{R}^{d\times C} \qquad \mathbf{x}_i = \begin{bmatrix} x_{1i} \\ x_{2i} \\ \vdots \\ x_{di} \end{bmatrix} \in \mathbb{R}^d \tag{3}$$

[1] The choice of class label does not matter - we could have just as well chosen -1.
[2] Operating with sums is easier, hence the logarithm. The minus is added for making the function convex.

To arrive at our probabilistic model, we start from the following equations:

$$
\begin{array}{lll}
\log\left(\mathbb{P}(b_i = 1 \mid \mathbf{a}_i, \mathbf{X})\, S\right) = \mathbf{a}_i^T \mathbf{x}_1 & \Longrightarrow \log \mathbb{P}(b_i = 1 \mid \mathbf{a}_i, \mathbf{X}) = \mathbf{a}_i^T \mathbf{x}_1 - \log S & \Longrightarrow \mathbb{P}(b_i = 1 \mid \mathbf{a}_i, \mathbf{X}) = \dfrac{e^{\mathbf{a}_i^T \mathbf{x}_1}}{S} \\
\log\left(\mathbb{P}(b_i = 2 \mid \mathbf{a}_i, \mathbf{X})\, S\right) = \mathbf{a}_i^T \mathbf{x}_2 & \Longrightarrow \log \mathbb{P}(b_i = 2 \mid \mathbf{a}_i, \mathbf{X}) = \mathbf{a}_i^T \mathbf{x}_2 - \log S & \Longrightarrow \mathbb{P}(b_i = 2 \mid \mathbf{a}_i, \mathbf{X}) = \dfrac{e^{\mathbf{a}_i^T \mathbf{x}_2}}{S} \\
\vdots & \vdots & \vdots \\
\log\left(\mathbb{P}(b_i = C \mid \mathbf{a}_i, \mathbf{X})\, S\right) = \mathbf{a}_i^T \mathbf{x}_C & \Longrightarrow \log \mathbb{P}(b_i = C \mid \mathbf{a}_i, \mathbf{X}) = \mathbf{a}_i^T \mathbf{x}_C - \log S & \Longrightarrow \mathbb{P}(b_i = C \mid \mathbf{a}_i, \mathbf{X}) = \dfrac{e^{\mathbf{a}_i^T \mathbf{x}_C}}{S}
\end{array}
$$

Different from the binary case, the probability in the denominator is replaced by a factor S that acts as a normalizing constant. We set $S = \sum_{k=1}^{C} e^{\mathbf{a}_i^T \mathbf{x}_k}$ in order for the probabilities to sum to 1. We therefore have:

$$
\mathbb{P}(b_i = j \mid \mathbf{a}_i, \mathbf{X}) = \frac{e^{\mathbf{a}_i^T \mathbf{x}_j}}{\sum_{k=1}^{C} e^{\mathbf{a}_i^T \mathbf{x}_k}}
$$

PART I - THEORY (35 POINTS)

EXERCISE 1.I.1: (7 POINTS) - *Following the same reasoning as for the binary case, show that the maximum likelihood estimator for multinomial logistic regression problem has the following expression:*

$$
\hat{\mathbf{X}}_{ML} \in \arg\min_{\mathbf{X}} \left\{ f(\mathbf{X}) \mid f : \mathbb{R}^{d\times C} \to \mathbb{R},\ f(\mathbf{X}) = \sum_{i=1}^{n} \left(\log \sum_{k=1}^{C} e^{\mathbf{a}_i^T \mathbf{x}_k} - \mathbf{a}_i^T \mathbf{x}_{b_i} \right),\ b_i \in \{1, 2 \ldots C\} \right\}. \tag{4}
$$

EXERCISE 1.I.2: (7 POINTS) - *Show that the gradient with respect to X of the loss function f from Exercise I.1 has the following matrix form (this will come in useful in the second part of the exercise where you have to implement the algorithms):*

$$
\nabla f(\mathbf{X}) = \mathbf{A}^T \left(\mathbf{Z} \exp(\mathbf{A}\mathbf{X}) - \mathbf{Y} \right),
$$

where $\mathbf{Z}$ *is an* $n \times n$ *matrix with* $\mathbf{Z}_{ij} = \begin{cases} \frac{1}{\sum_{k=1}^{C} e^{\mathbf{a}_i^T \mathbf{x}_k}}, & \text{if } i = j \\ 0, & \text{othewrise} \end{cases}$ *, Y is an* $n \times C$ *matrix whose rows* $\mathbf{y}_i^T$ *are the one-hot-encodings* [3] *corresponding to data* b_i *and* exp *is applied entry-wise.*

HINT: *It is helpful to first compute the gradient of f with respect to a single entry* x_{ij} *of* $\mathbf{X}$.

EXERCISE 1.I.3: (SELF-STUDY, 0 POINTS) - *In this exercise, we are interested in showing that f is convex. We provide you with the expression of the Hessian of f below* [4]:

$$
\nabla^2 f(\mathbf{X}) = \sum_{i=1}^{n} \boldsymbol{\Sigma}_i \otimes \mathbf{a}_i \mathbf{a}_i^T, \tag{5}
$$

where $\otimes$ *is the Kronecker product, and* $\boldsymbol{\Sigma}_i$ *is given by:*

$$
\boldsymbol{\Sigma}_i = \begin{bmatrix} \sigma_{i1}(1 - \sigma_{i1}) & -\sigma_{i1}\sigma_2 & \ldots & -\sigma_{i1}\sigma_{iC} \\ -\sigma_{i2}\sigma_{i1} & \sigma_{i2}(1 - \sigma_{i2}) & \ldots & -\sigma_{i2}\sigma_{iC} \\ \vdots & \vdots & \vdots & \vdots \\ -\sigma_{iC}\sigma_{i1} & -\sigma_{iC}\sigma_{i2} & \ldots & \sigma_{iC}(1 - \sigma_{iC}) \end{bmatrix}, \quad \text{with} \quad \sigma_{ij} = \frac{e^{\mathbf{a}_i^T \mathbf{x}_j}}{\sum_{k=1}^{C} e^{\mathbf{a}_i^T \mathbf{x}_k}}. \tag{6}
$$

Knowing that a function of multiple variables is convex iff its Hessian is PSD, use the expression of $\nabla^2 f$ *above to prove that f is convex.*

HINT:

- *Start by showing that both* $\boldsymbol{\Sigma}_i \succcurlyeq 0$ *and* $\mathbf{a}_i \mathbf{a}_i^T \succcurlyeq 0,\ \forall i$. *You can use the following:* $\mathbf{D} \succcurlyeq 0 \iff \mathbf{z}^T \mathbf{D} \mathbf{z} \geq 0,\ \forall \mathbf{z}$.

[3]The one-hot-encoding of class $k \leq C$ is given by vector $y \in \mathbb{R}^C$, with $y_k = 1$ and $y_j = 0,\ j \neq k$. Precisely, if $b_i = 2$, then y_i has 1 on position 2 and 0 everywhere else.
[4]The interested student can refer to the Appendix for details of the derivation.

- *Then apply the following result [9]:* $\mathbf{D}, \boldsymbol{E} \succcurlyeq 0 \implies \mathbf{D} \otimes \boldsymbol{E} \succcurlyeq 0$.
- *Finally, argue that given two sum-compatible matrices* $\mathbf{D}, \boldsymbol{E} \succcurlyeq 0$*, you have* $\mathbf{D} + \boldsymbol{E} \succcurlyeq 0$.

EXERCISE 1.I.4: (8 POINTS) - *We now seek to estimate the Lipschitz constant L of ∇f, as we will need it for PART II of this exercise. We know that L is such that $\lambda_{max}(\nabla^2 f) \leq L < \infty$, where $\lambda_{max}(\nabla^2 f)$ is the largest eigenvalue of $\nabla^2 f(X)$.*

a) *Knowing that $\mathbf{a}_i\mathbf{a}_i^T$ is a rank-1 matrix, show that $\lambda_{max}(\mathbf{a}_i\mathbf{a}_i^T) = \|\mathbf{a}_i\|_2^2$.*

b) *Starting from* (5)*, show that we can choose $L = \frac{\|\mathbf{A}\|_F^2}{2}$*

HINT: *You can use the following statements:*

- *For* $\mathbf{D}, \boldsymbol{E} \succcurlyeq 0$*, we have* $\lambda_{max}(\mathbf{D} \otimes \boldsymbol{E}) = \lambda_{max}(\mathbf{D})\lambda_{max}(\boldsymbol{E})$.
- *Given* $\mathbf{D} \succcurlyeq 0$*, we have* $\lambda_{max}(d) \leq \max_i \sum_j \left|\mathbf{D}_{ij}\right|$ *(the top eigenvalue is upper-bounded by the maximum ℓ_1 norm of rows).*
- $1 - \sigma_{ij} = \sum_{\substack{k=1 \\ k\neq j}}^{C} \sigma_{ik}$. *This comes from having a probability distribution over the C classes.*

EXERCISE 1.I.5: (8 POINTS) - *Sometimes we wish to enforce certain properties onto our solution (e.g. sparsity). This can be achieved by adding a regularizer to our objective:*

$$F(\mathbf{X}) = f(\mathbf{X}) + \lambda g(\mathbf{X}),$$

As we saw in Lecture 9, such a function can be minimized using proximal gradient algorithms, provided that g is 'proximable' (i.e., its proximal operator is efficient to evaluate). We recall the proximal operator of g as the solution to the following convex problem:

$$\text{prox}_g(\mathbf{z}) := \arg\min_{\mathbf{y}\in\mathbb{R}^d}\{g(\mathbf{y}) + \frac{1}{2}\|\mathbf{y} - \mathbf{z}\|_2^2\}.$$

Given $g : \mathbb{R}^d \to \mathbb{R},\ g(\mathbf{x}) := \|\mathbf{x}\|_1$, show that its proximal function can be written as

$$\text{prox}_{\lambda g}(\mathbf{z}) = \max(|\mathbf{z}| - \lambda, 0) \circ \text{sign}(\mathbf{z}),\ \mathbf{z} \in \mathbb{R}^d \tag{7}$$

where the operators max, sign *and* $|\cdot|$ *are applied coordinate-wise to the vector* $\mathbf{z}$ *and* $\circ$ *stands for* $(\mathbf{x} \circ \mathbf{y})_i = x_i y_i$. *Such a regularizer imposes sparsity on the solutions.*

HINT: *Show that relation* (7) *holds for $z \in \mathbb{R}$. Then use the fact that $\|\mathbf{y}\|_1 + \frac{1}{2}\|\mathbf{y} - \mathbf{z}\|_2^2 = \sum_{i=1}^{d}|y_i| + \frac{1}{2}(y_i - z_i)^2$ to conclude that the operators can be applied coordinate-wise.*

EXERCISE 1.I.6: (5 POINTS) - *Given $g(\mathbf{x}) := \frac{1}{2}\|\mathbf{x}\|_2^2$, show that its proximal function can be written as*

$$\text{prox}_{\lambda g}(\mathbf{z}) = \frac{\mathbf{z}}{1 + \lambda}. \tag{8}$$

Such a regularizer bounds the growth of X's components, thus helping with numerical stability.

PART II - HANDWRITTEN DIGIT CLASSIFICATION (35 POINTS)

In this section you will implement algorithms based on your theoretical derivations from **PART I** and test them on the well-known MNIST dataset [4] consisting of 28×28 grayscale images of handwritten digits. Figure 1 below depicts a subset of samples pertaining to each of the 10 classes.

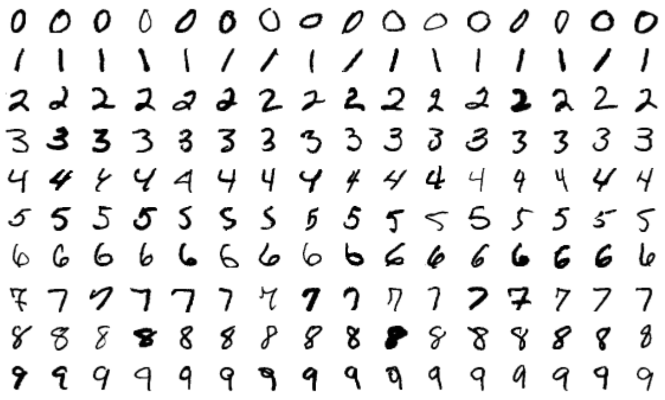

Figure 1: MNIST selected digit images.
Source: `https://wikipedia.org/wiki/MNIST_database`

The problem constants become $C = 10$, $d = 28 \times 28 = 784$ and we choose $n = 5000$. The rows $\mathbf{a}_i^T$ of our data matrix $\mathbf{A}$ are the vectorized[5] representations of the digit images. Finally, the labels b_i are integers of the set $\{0, 1, \dots 9\}$. The data is preprocessed and given to you in compressed format in the file `mnist_train_test_split.npz`. The code for loading it is already provided in the `main.py` file.

For this exercise you will implement algorithms which solve the following regularized versions of the multiclass Logistic Regression:

- **ℓ_1 regularized (or sparse) Logistic Regression:**

$$F(\mathbf{X}) = f(\mathbf{X}) + \lambda \sum_{k=1}^{C} \|\mathbf{x}_k\|_1 \, . \tag{9}$$

- **ℓ_2 regularized Logistic Regression:**

$$F(\mathbf{X}) = f(\mathbf{X}) + \frac{\lambda}{2} \sum_{k=1}^{C} \|\mathbf{x}_k\|_2^2 \, . \tag{10}$$

Using the information in PART I of this exercise, as well as the contents of Lecture 9 you are asked to fill in the code gaps, run the experiments and interpret the results.

EXERCISE 1.II.1: (20 POINTS) - *In the file named `algorithms.py` you can find the incomplete methods whose code you need to fill in.*

a) *In the file `operators.py` fill in the methods `l1_prox` and `l2_prox` with the proximal operator expressions from PART I adapted to problems* (9) *and* (10)*, respectively. Also fill in the `gradfx` with the expression of ∇f from PART I.*

b) *Using the information in Lecture 9 fill in the codes of methods `ista`, `fista` (adapted to also handle restarting) and `gradient_scheme_restart_condition` from the file `algorithms.py`. In the latter method, you need to implement the so-called 'gradient restart condition', described in depth in papers such as [2, 7]. Formally, the criterion is:*

$$\langle \mathbf{Y}^t - \mathbf{X}^{t+1}, \mathbf{X}^{t+1} - \mathbf{X}^t \rangle > 0,$$

where t is the iteration index. The term $\mathbf{Y}^t - \mathbf{Y}^{t+1}$ can be seen as a gradient, while $\mathbf{X}^{t+1} - \mathbf{X}^t$ is the descent direction of the momentum term. Overall this criterion resets the momentum of FISTA to 0 when it goes in a bad descent direction. Note that we are working with matrices, so we need to adapt the inner product correspondingly: $\langle \mathbf{D}, \boldsymbol{E} \rangle = Tr(\mathbf{D}^T \boldsymbol{E})$.

NOTE: *The $\mathbf{Y}$ in the gradient restart condition refers to the FISTA iterate.* ***Do not*** *confuse it with the one-hot-encoding matrix from* (1).

c) *Using Lecture 9 as guideline, fill in the method `prox_sg` form the file `algorithm.py` and `stocgradfx` from `operators.py`. The latter method needs to return the minibatch stochastic gradient of f. It is not difficult to show that given m_s stochastic samples of $\mathbf{A}$ i.e., rows of $\mathbf{A}$ chosen uniformly at random, then*

$$\tilde{\nabla}_{m_s} f(\mathbf{X}) = \frac{n}{m_s} \sum_{i=1}^{m_s} \mathbf{a}_i \left(\frac{e^{\mathbf{a}_i^T X}}{\sum_{k=1}^{C} e^{\mathbf{a}_i^T \mathbf{x}_k}} - \mathbf{y}_i^T \right)$$

[5]A vectorized representation of a matrix $Y \in \mathbb{R}^{n \times d}$ is given by a vector $y \in \mathbb{R}^{nd}$ obtained by arranging the columns of Y on top of one another.

is an unbiased stochastic gradient of $\nabla f(\mathbf{X})$. The minibatch size and learning rate decrease function are given to you in the `params` variable of the `main` method - please make sure to use those.

***Note:** For the `prox_sg` method you need to record convergence with respect to the ergodic iterate $\hat{\mathbf{X}}^t = \frac{\sum_{j=1}^{t} \gamma_j \mathbf{X}^j}{\sum_{j=1}^{t} \gamma_j}$, where γ_j is the learning rate at step j.*

EXERCISE 1.II.2: (10 POINTS) - *Running the script provided in `main.py` will produce plots of the convergence for your implementation of the 4 methods above.*

a) Insert the convergence plots into your report.

b) Run the deterministic methods for 2000 *iterations and comment on how the convergence rates observed in your plots compare to the theoretical ones (refer to the lecture notes). Are the practical rates better, worse or as expected?*

c) For the deterministic methods, compare the convergence speed of FISTA relative to ISTA, and of FISTA-RESTART with respect to FISTA. Explain your observations.

d) Run Prox-SG for 50000 *iterations. Comment on the accuracy and convergence speed of deterministic methods versus Prox-SG. Note that Prox-SG is plotted with respect to the number of epochs (full passes through the data), where an epoch is* comparable *to a full gradient. Explain your observations.*

e) In the `main.py` file, you have calls to the method `plot_digit_features`. This method takes as input the feature matrices $\mathbf{X}$ *obtained from running FISTA with restart for* 2000 *iterations for both problems* (9) *and* (10)*. Insert the plots produced by calling these methods into the report. Do you observe any similarity between the feature vectors and their corresponding classes?*

EXERCISE 1.II.3: (5 POINTS) - *In the file called `train_and_test_mnist_nn.py` you are provided with the code for training and evaluating a fully-connected three-layer neural net with ReLU activation functions. Identical to Logistic Regression, the network is trained on a dataset with* 5000 *MNIST images. We now compare the test-set accuracies corresponding to the solutions obtained by the neural net, ℓ_1 and ℓ_2 regularized Logistic Regression. These accuracies are computed for a test set of* 10000 *MNIST images.*

- *Run `train_and_test_mnist_nn.py` until completion and insert the final test set accuracy printed on the screen.*
- *In the `main.py` template you are provided with code* [6] *that reports the test-set accuracy of the solution obtained by FISTA with gradient scheme restart for the ℓ_1 and ℓ_2 regularized problems. Run the method for* 2000 *iterations and insert the two accuracies in your report.*
- *Is there a difference between the accuracy of the neural net solution and those obtained by FISTA with gradient scheme restart? Does the Logistic regression model have the same expressive power as the neural net? Why / why not?*

  HINT: *Think of linear versus nonlinear models.*

2 IMAGE RECONSTRUCTION - 30 POINTS

In this exercise an *image* should be understood as the matrix equivalent of a digital grayscale image, where an entry represents the intensity of the corresponding pixel.

2.1 Wavelets

A widely used multi-scale localized representation in signal and image processing is the wavelet transform.[7] This representation is constructed so that piecewise polynomial signals have sparse wavelet expansions [5]. Since many real-world signals can be composed by piecewise smooth parts, it follows that they have sparse or compressible wavelet expansions.

Scale. In wavelet analysis, we frequently refer to a particular scale of analysis for a signal. In particular, we consider dyadic scaling, such that the supports from one scale to the next are halved along each dimension. For images, which can be represented with matrices, we can imagine the highest scale as consisting of each pixel. At other scales, each region correspond to the union of four neighboring regions at the next higher scale, as illustrated in Figure 2.

The Wavelet transform. The wavelet transform offers a multi-scale decomposition of an image into coefficients related to different dyadic regions and orientations. In essence, each wavelet coefficient is given by the scalar product of the image with a wavelet function which is concentrated approximately on some dyadic square and has a given orientation (vertical, horizontal or diagonal). Wavelets

[6]The section which calls method `compute_accuracy`.

[7]This section is an adaption from *Signal Dictionaries and Representations* by M. Watkin (see `http://bit.ly/1wXDDbG`).

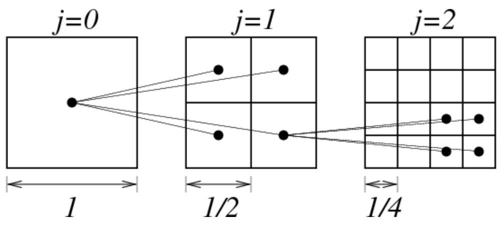

Figure 2: Dyadic partitioning of the unit square at scales $j = 0, 1, 2$. The partitioning induces a coarse-to-fine parent/child relationship that can be modeled using a tree structure.

are essentially bandpass functions that detect abrupt changes in a signal. The scale of a wavelet, which controls its support both in time and in frequency, also controls its sensitivity to changes in the signal.

An important property of the wavelet functions is that they form an orthonormal basis $\mathbf{W}$. Formally, this means $\mathbf{W}\mathbf{W}^T = \mathbf{W}^T\mathbf{W} = \mathbf{I}$, where $\mathbf{I}$ is the identity operator. In the discrete case, $\mathbf{W}$ is just an orthonormal matrix.

Implementation. We provide the codes that efficiently compute the 2D Wavelet transform $\mathbf{W}$ and its inverse $\mathbf{W}^{-1} = \mathbf{W}^T$. Operators $\mathbf{W}$ and $\mathbf{W}^T$ receive a vectorized image as input and outputs a vectorized transform. We base our implementation on the `PyWavelets` library[8]. You can find a usage example in the file `example_wavelet.py`. The interaction with the `pywt` library happens through the class `RepresentationOperator` available in the `utilities` file.

- The method `W(im)` computes the transformation from an image `im` to its wavelet counterpart `im_wav`.
- The method `WT(im_wav)` computes the transformation from wavelet coefficients `im_wav` back to the image domain `im`.

Remark 1. Should the provided codes not work after you installed the required packages, directly contact one of the teaching assistants.

2.2 Total Variation

The Total Variation (TV) was proposed by Rudin, Osher and Fatemi [8] as a regularizer for solving inverse problems. There is compelling empirical evidence showing that this regularization does not blur the sharp edges of images and as a consequence, significant research efforts went into developing efficient algorithms for computing the TV proximal operator. In this exercise we use the approach developed by Chambolle [1].

Computing the TV of an image relies on the discrete gradient operator $\nabla\mathbf{x} \in \mathbb{R}^{(m\times m)\times 2}$, formally given by $(\nabla\mathbf{x})_{i,j} = \left((\nabla\mathbf{x})^1_{i,j}, (\nabla\mathbf{x})^2_{i,j}\right)$ with

$$(\nabla\mathbf{x})^1_{i,j} = \begin{cases} \mathbf{x}_{i+1,j} - \mathbf{x}_{i,j} & \text{if } i < m, \\ 0 & \text{if } i = m, \end{cases}$$

$$(\nabla\mathbf{x})^2_{i,j} = \begin{cases} \mathbf{x}_{i,j+1} - \mathbf{x}_{i,j} & \text{if } j < m, \\ 0 & \text{if } i = m. \end{cases}$$

Then, the discrete Total Variation (TV) is defined as

$$\|\mathbf{x}\|_{\mathrm{TV}} := \begin{cases} \sum_{1\le i,j\le m} \|(\nabla\mathbf{x})_{i,j}\|_2 & \text{called } \textit{isotropic} \\ \sum_{1\le i,j\le m} |(\nabla\mathbf{x})^1_{i,j}| + |(\nabla\mathbf{x})^2_{i,j}| & \text{called } \textit{anisotropic} \end{cases}$$

Even though the isotropic case presents a slightly more difficult computational challenge, it has the advantage of being unaffected by the direction of the edges in images.

The TV-norm can be seen as the ℓ_1-norm of the absolute values of the gradients evaluated at each pixel. Analogous to the case of ℓ_1-norm, minimizing the TV-norm subject to data fidelity leads to sparse gradients, which implies an image with many flat regions and few sharp transitions; see Figure **??**.

[8]`https://pywavelets.readthedocs.io/en/latest/index.html`

2.3 Image in-painting

Image in-painting consists in reconstructing the missing parts of an image. By exploiting the structures such as sparsity over the wavelet coefficients, it is possible to in-paint images with a large portion of their pixels missing. In this part of the homework, we are going to study the convergence of different methods related to FISTA, as well as different restart strategies. We consider a subsampled image $\mathbf{b} = \mathbf{P}_\Omega \mathbf{x}$, where $\mathbf{P}_\Omega \in \mathbb{R}^{n \times p}$ is an operator that selects only few, $n \ll p := m^2$, pixels from the vectorized image $\mathbf{x} \in \mathbb{R}^p$. We can try to reconstruct the original image $\mathbf{x}$ by solving the following problems.

$$\min_{\boldsymbol{\alpha} \in \mathbb{R}^p} \underbrace{\frac{1}{2}\|\mathbf{b} - \mathbf{P}_\Omega \mathbf{W}^T \boldsymbol{\alpha}\|_2^2}_{f_{\ell_1}(\boldsymbol{\alpha})} + \underbrace{\lambda_{\ell_1}\|\boldsymbol{\alpha}\|_1}_{g_{\ell_1}(\boldsymbol{\alpha})}, \tag{11}$$

$$\min_{\mathbf{x} \in \mathbb{R}^p} \underbrace{(1/2)\|\mathbf{b} - \mathbf{P}_\Omega \mathbf{x}\|_2^2}_{f_{\mathrm{TV}}(\mathbf{x})} + \underbrace{\lambda_{\mathrm{TV}}\|\mathbf{x}\|_{\mathrm{TV}}}_{g_{\mathrm{TV}}(\mathbf{x})}. \tag{12}$$

Here, the operator $\mathbf{W}^T$ is vectorized to be dimension compatible with $\mathbf{P}_\Omega$. An example of image in-painting with this model is given in Figure 3.

EXERCISE 2.1: (5 POINTS) -

a) *Write the gradient of* $f(\boldsymbol{\alpha})$ *in* (11) *and* $f(\mathbf{x})$ *in* (12).

b) *Find the Lipschitz constant of* $\nabla_\alpha f(\boldsymbol{\alpha})$ *for* (11) *and* $\nabla_{\mathbf{x}} f(\mathbf{x})$ *in* (12) *analytically.*

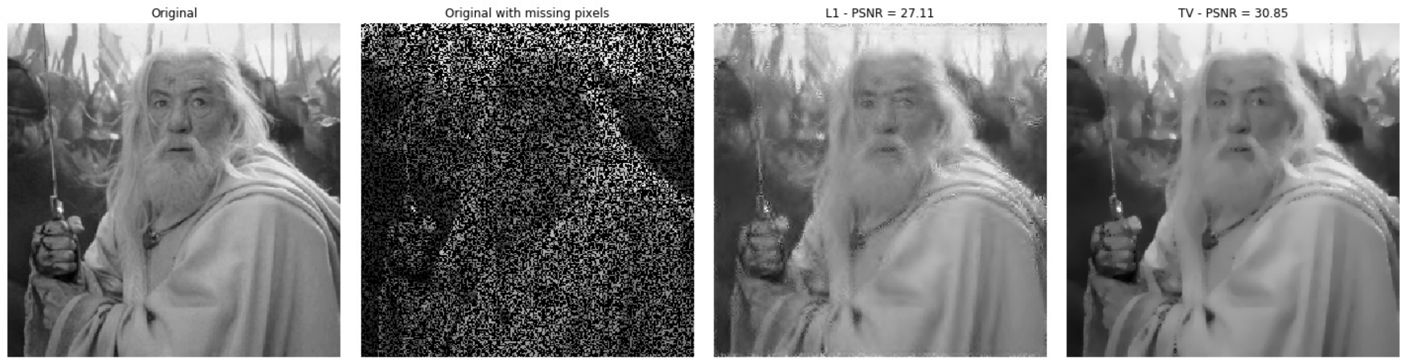

Figure 3: An example of image in-painting with the three models described in this homework. The regularization parameter has been set to 0.01 for all methods.

EXERCISE 2.2: (10 POINTS) - *Adapt the FISTA algorithm from Exercise II.1 to solve the problems* (11) *and* (12).

a) *Take a* 256×256 *natural image or better, a picture of you, and subsample* 40% *of its pixels uniformly at random (i.e. remove* 60% *of the pixels).*

b) *Perform a parameter sweep over* λ_1 *and* λ_{TV}. *You should choose a meaningful range for the parameters, one that allows you to observe the behavior of the PSNR as a function of the regularization parameters. Use* 200 *iterations for your algorithm.*

c) *Plot the obtained PSNR against the regularization parameter values.*

d) *Provide your codes with comments, and include the reconstructed images.*

EXERCISE 2.3: (10 POINTS) - *You will now study the convergence to* $\mathbf{x}^*$, *the optimal solution of problem* (11). *We use use* $\lambda = 0.01$ *and again uniformly subsample* 40% *of the pixels.*

a) *Adapt (or reuse) the ISTA variations you implemented in Exercise II.1:*

- *ISTA*
- *FISTA*
- *FISTA with gradient scheme restart (see Exercise 1.II.1)*

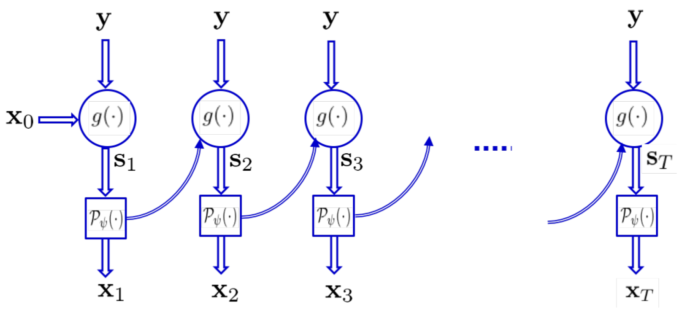

Figure 4: Illustration of the unrolled proximal method. The function $g(\cdot)$ represents the gradient step $\mathbf{x}^k - \alpha_k \nabla f(\mathbf{x}^k)$, $\mathbf{y}$ represents the measurements $\mathbf{b}$, while the function $\mathcal{P}_\psi(\cdot)$ (corresponding to g_θ in text) is the learned proximal step. The values $\mathbf{x}_i$ are then the results of the different iterations. Taken from [6].

b) *The optimal solution $\mathbf{x}^*$ of problem (11) and the corresponding optimal value $F^* := f(\mathbf{x}^*) + g(\mathbf{x}^*)$ are not known a priori. You need to estimate them by running FISTA with gradient scheme restart for 5000 iterations and use the relative error $rerr^k < 10^{-15}$ () as the stopping criterion. The error at iteration k is defined as*

$$err^k = \|\mathbf{y}^{k+1} - \mathbf{y}^k\|_2, \qquad rerr^k := \frac{err^k}{err^0}.$$

Write the value that you obtained for F^* in the report.

c) *Study the convergence of ISTA, FISTA, FISTA with gradient scheme restart over 2000 iterations. Adapt the tolerance criterion to stop iterating when $\frac{|F(\mathbf{x}^k)-F^*|}{F^*} < 10^{-15}$. Plot a graph $\log\left(\frac{|F(\mathbf{x}^k)-F^*|}{F^*}\right)$ against k (cf. the examples in Lecture 10) and comment on the different rates and behaviors observed.*

Hint: *You can use the function* `semilogy` *for the plot.*

d) *Replace then the F^* with $F^\natural := F(\mathbf{x}^\natural)$, where $\mathbf{x}^\natural$ is your ground-truth image. Plot a graph $\log\left(|F(\mathbf{x}^k) - F^\natural|/F^\natural\right)$ against k using the same algorithms as in 3. for 1000 iterations. Comment on the differences.*

e) *Provide your codes with comments, as well as both convergence plots with discussion on the results.*

Comparison with algorithm unrolling. In recent years, the interest has grown for deep learning method that learn to imitate and improve onto algorithms like ISTA of FISTA for composite convex minimization. Given un update rule for a composite minimization problem of form of (12), i.e. $\mathbf{x}^{k+1} = prox_g(\mathbf{x}^k - \alpha_k \nabla f(\mathbf{x}^k))$, the approach of [6] proposes to learn the proximal operator with a neural network.

They fix the number of iterations that they will use to T and alternate between the inner updates $(\mathbf{x}^k - \alpha_k \nabla f(\mathbf{x}^k))$ and the proximal operator parametrized by a neural network (we will call it g_θ), as shown on figure 4. On Figure 4, one starts from the undersampled image $\mathbf{x}_0$ and iterate T times according to the rule $\mathbf{x}^{k+1} = prox_{g_\theta}(\mathbf{x}^k - \alpha_k \nabla f(\mathbf{x}^k))$. The network g_θ is then trained to minimize $\mathbb{E}_{\mathbf{x}^\natural}[\|\mathbf{x}_T - \mathbf{x}^\natural\|]$, i.e. to minimize the average ℓ_2 distance between natural images $\mathbf{x}^\natural$ and reconstructed images $\mathbf{x}_T$, starting from the measured data $\mathbf{x}_0 = \mathbf{b} = \mathbf{P}_\Omega \mathbf{x}^\natural$. Their result show a drastic improvement upon some classically used methods such as Wavelet minimization (Problem (11)).
We have implemented and pretrained the method for inpainting images of size 256×256 and we would like you to explore compare your implementations to the method. An example on how to use it is provided in the script `example_unrolled_nn.py`, which implements $T = 5$ unrolled step with an operator parametrized by a simple residual network.

Exercise 2.4: (5 Points) - *Compare the reconstruction performance and speed of reconstruction for FISTA with gradient scheme restart for ℓ_1 and TV minimization.*

a) *Compare the performance after 500 reconstruction steps using ℓ_1 and TV minimization against the unrolled method and report the PSNR and reconstruction time, as well as the images reconstructed and error maps ($|\mathbf{x} - \hat{\mathbf{x}}|$).*

b) *Compare the performance after $T = 5$ iterations of all three methods, and report the PSNR and reconstruction time as well as the images reconstructed and error maps ($|\mathbf{x} - \hat{\mathbf{x}}|$). Comment on your results.*

c) Comment on the results. Which image appears the best to you, and why? Identify a tradeoff between the unrolled method and the classical iterative approaches.

N.B. If your image yields poor reconstruction results with the unrolled CNN, try running it with of the provided images, such as `gandalf.jpg`*.*

3 Guidelines for the preparation and the submission of the homework

Work on your own. Do not copy or distribute your codes to other students in the class. Do not reuse any other code related to this homework. Here are few warnings and suggestions for you to prepare and submit your homework.

- This homework is due at 4:00PM, 6 December, 2019.
- Submit your work before the due date. Late submissions are not allowed and you will get 0 point from this homework if you submit it after the deadline.
- Questions of 0 point are for self study. You do not need to answer them in the report.
- Your final report should include detailed answers and it needs to be submitted in PDF format.
- The PDF file can be a scan or a photo. Make sure that it is eligible.
- The results of your simulations should be presented in the final report with clear explanation and comparison evaluation.
- We provide some Python scripts that you can use to implement the algorithms, but you can implement them from scratch using any other convenient programming tool (in this case, you should also write the codes to time your algorithm and to evaluate their efficiency by plotting necessary graphs).
- Even if you use the Python scripts that we provide, you are responsible for the entire code you submit. Apart from completing the missing parts in the scripts, you might need to change some written parts and parameters as well, depending on your implementations.
- The codes should be well-documented and should work properly. Make sure that your code runs without error. If the code you submit does not run, you will not be able to get any credits from the related exercises.
- Compress your codes and your final report into a single ZIP file, name it as `ee556hw3_NameSurname.zip`, and submit it through the moodle page of the course.

References

[1] CHAMBOLLE, A. An algorithm for total variation minimization and applications. *J. Math. Imaging Vis. 20* (2004), 89–97.

[2] GISELSSON, P., AND BOYD, S. Monotonicity and restart in fast gradient methods. In *Decision and Control (CDC), 2014 IEEE 53rd Annual Conference on* (2014), IEEE, pp. 5058–5063.

[3] JORDAN, M. I., ET AL. Why the logistic function? a tutorial discussion on probabilities and neural networks, 1995.

[4] LECUN, Y., AND CORTES, C. MNIST handwritten digit database.

[5] MALLAT, S. *A wavelet tour of signal processing*. Academic press, 1999.

[6] MARDANI, M., SUN, Q., DONOHO, D., PAPYAN, V., MONAJEMI, H., VASANAWALA, S., AND PAULY, J. Neural proximal gradient descent for compressive imaging. In *Advances in Neural Information Processing Systems* (2018), pp. 9573–9583.

[7] O'DONOGHUE, B., AND CANDES, E. Adaptive restart for accelerated gradient schemes. *Foundations of computational mathematics 15*, 3 (2015), 715–732.

[8] RUDIN, L., OSHER, S., AND FATEMI, E. Nonlinear total variation based noise removal algorithms. *Physica D: Nonlinear Phenomena 60*, 1-4 (1992), 259–268.

[9] SCHACKE, K. On the kronecker product.

APPENDIX

A Deriving the Hessian of multinomial logistic function

Differentiating expression (1) a second time with respect to **X** gives us a 3D tensor as the Hessian, which is cumbersome to work with. Luckily, we can avoid this by differentiating f with respect to a vectorized version of **X**, thus imposing a regular matrix form on $\nabla^2 f$. Our Hessian will now belong to $\mathbb{R}^{dC\times dC}$. The vectorized **X** and $\nabla_{X_{vec}} f$ corresponding to it are given below:

$$\mathbf{X}_{vec} = \begin{bmatrix} x_{11} \\ \vdots \\ x_{d1} \\ \vdots \\ x_{1C} \\ \vdots \\ x_{dC} \end{bmatrix} \in \mathbb{R}^{dc} \qquad \nabla_{\mathbf{X}_{vec}} f(\mathbf{X}) = \sum_{i=1}^{n} \left(\frac{1}{\sum\limits_{k=1}^{C} e^{\mathbf{a}_i^T \mathbf{x}_k}} \begin{bmatrix} a_{i1} e^{\mathbf{a}_i^T \mathbf{x}_1} \\ \vdots \\ a_{id} e^{\mathbf{a}_i^T \mathbf{x}_1} \\ \vdots \\ a_{i1} e^{\mathbf{a}_i^T \mathbf{x}_C} \\ \vdots \\ a_{id} e^{\mathbf{a}_i^T \mathbf{x}_C} \end{bmatrix} - \begin{bmatrix} a_{i1}\, \mathbf{1}_1(b_i) \\ \vdots \\ a_{id}\, \mathbf{1}_1(b_i) \\ \vdots \\ a_{i1}\, \mathbf{1}_C(b_i) \\ \vdots \\ a_{id}\, \mathbf{1}_C(b_i) \end{bmatrix} \right) = \sum_{i=1}^{n} \left(\begin{bmatrix} \frac{e^{\mathbf{a}_i^T \mathbf{x}_1}}{\sum_{k=1}^{C} e^{\mathbf{a}_i^T \mathbf{x}_k}} \\ \vdots \\ \frac{e^{\mathbf{a}_i^T \mathbf{x}_C}}{\sum_{k=1}^{C} e^{\mathbf{a}_i^T \mathbf{x}_k}} \end{bmatrix} \otimes \begin{bmatrix} a_{i1} \\ \vdots \\ a_{id} \end{bmatrix} - \mathbf{y}_i \otimes \begin{bmatrix} a_{i1} \\ \vdots \\ a_{id} \end{bmatrix} \right) = \sum_{i=1}^{n} \left(\begin{bmatrix} \frac{e^{\mathbf{a}_i^T \mathbf{x}_1}}{\sum_{k=1}^{C} e^{\mathbf{a}_i^T \mathbf{x}_k}} \\ \vdots \\ \frac{e^{\mathbf{a}_i^T \mathbf{x}_C}}{\sum_{k=1}^{C} e^{\mathbf{a}_i^T \mathbf{x}_k}} \end{bmatrix} \otimes \mathbf{a}_i - \mathbf{y}_i \otimes \mathbf{a}_i \right), \tag{13}$$

where $\mathbf{1}_j(b_i) = \begin{cases} 1, \text{ if } b_i = j \\ 0, \text{ otherwise} \end{cases}$ is the indicator function. We can now differentiate $\nabla_{\mathbf{X}_{vec}} f(\mathbf{X})$ with respect to $\mathbf{X}_{vec}$ to obtain the Hessian. Note that the second term within the sum does not depend on $\mathbf{X}_{vec}$ and will therefore not appear in $\nabla^2_{\mathbf{X}_{vec}} f(\mathbf{X})$. We make a further simplification and only consider one data point, as the final expression will be the summation over the data. To ease notation, we drop the i-index from $\mathbf{a}_i$ and denote as $\sigma_j := \frac{e^{\mathbf{a}^T \mathbf{x}_j}}{\sum_{k=1}^{C} e^{\mathbf{a}^T \mathbf{x}_k}}$. Finally, we observe that $\frac{\partial \sigma_j}{\partial x_{lj}} = a_l \sigma_j (1-\sigma_j)$ and $\frac{\partial \sigma_j}{\partial x_{lp}} = -a_l \sigma_j \sigma_p$.

Using the new conventions and differentiating a second time with respect to X_{vec}, we have:

$$\frac{\partial \begin{bmatrix} \frac{e^{\mathbf{a}^T \mathbf{x}_1}}{\sum_{k=1}^{C} e^{\mathbf{a}^T \mathbf{x}_k}} \\ \vdots \\ \frac{e^{\mathbf{a}^T \mathbf{x}_C}}{\sum_{k=1}^{C} e^{\mathbf{a}^T \mathbf{x}_k}} \end{bmatrix} \otimes \mathbf{a}}{\partial X_{vec}} = \frac{\partial \begin{bmatrix} \sigma_1 \\ \vdots \\ \sigma_C \end{bmatrix} \otimes \mathbf{a}}{\partial X_{vec}} = \frac{\partial \begin{bmatrix} \sigma_1 \\ \vdots \\ \sigma_C \end{bmatrix}}{\partial X_{vec}} \otimes \mathbf{a} \tag{14}$$

We address the first term of the Kronecker product separately:

$$\frac{\partial \begin{bmatrix} \sigma_1 \\ \vdots \\ \sigma_C \end{bmatrix}}{\partial X_{vec}} = \begin{bmatrix} a_1\sigma_1(1-\sigma_1) & a_2\sigma_1(1-\sigma_1) & \dots a_d\sigma_1(1-\sigma_1) & -a_1\sigma_1\sigma_2 & -a_2\sigma_1\sigma_2 & \dots - a_d\sigma_1\sigma_2 & \dots - a_d\sigma_1\sigma_C \\ -a_1\sigma_2\sigma_1 & -a_2\sigma_2\sigma_1 & \dots - a_d\sigma_2\sigma_1 & a_1\sigma_2(1-\sigma_2) & a_2\sigma_2(1-\sigma_2) & \dots a_d\sigma_2(1-\sigma_2) & \dots - a_d\sigma_2\sigma_C \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ -a_1\sigma_C\sigma_1 & -a_2\sigma_C\sigma_1 & \dots - a_d\sigma_C\sigma_1 & -a_1\sigma_C\sigma_2 & -a_2\sigma_C\sigma_2 & \dots - a_d\sigma_C\sigma_2 & \dots a_d\sigma_C(1-\sigma_C) \end{bmatrix} \tag{15}$$

$$= \underbrace{\begin{bmatrix} \sigma_1(1-\sigma_1) & -\sigma_1\sigma_2 & -\sigma_1\sigma_3 & \dots & -\sigma_1\sigma_C \\ -\sigma_2\sigma_1 & \sigma_2(1-\sigma_2) & -\sigma_2\sigma_3 & \dots & -\sigma_2\sigma_C \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ -\sigma_C\sigma_1 & -\sigma_C\sigma_2 & -\sigma_C\sigma_3 & \dots & \sigma_C(1-\sigma_C) \end{bmatrix}}_{\Sigma} \otimes \mathbf{a}^T \tag{16}$$

Finally we have:

$$\frac{\partial \begin{bmatrix} \sigma_1 \\ \vdots \\ \sigma_C \end{bmatrix} \otimes \mathbf{a}}{\partial X_{vec}} = \left(\Sigma \otimes \mathbf{a}^T\right) \otimes \mathbf{a} \tag{17}$$

$$= \Sigma \otimes \left(\mathbf{a}^T \otimes \mathbf{a}\right) \tag{18}$$

$$= \Sigma \otimes \mathbf{a}\mathbf{a}^T \tag{19}$$

$$\tag{20}$$

A simple summation over the data retrieves the expression of the Hessian given in (5).