Reminder: the course material is at : `https://gitlab.inria.fr/flandes/fpml`

1 Regressions

1.1 Linear Regression

1.1.1 The simplest case: 1−dimensional linear regression

You should already know the framework of linear regression, in the case where $x \in \mathbb{R}$ (and the ground truth is also $t_n \in \mathbb{R}$). Typically, we want to minimize the *Mean Squared Error*, i.e. the cost function is:

$$J(\Theta, X, T) = \frac{1}{N}\sum_n^N (f_\Theta(x_n) - t_n)^2 \tag{1}$$

$$\text{with the model:} \qquad f_\Theta(x) = \theta_0 + \theta_1.x \tag{2}$$

There are only 2 real parameters : $\Theta = (\theta_0, \theta_1)$. The data are the tuples: $(X, T) = (x_n, t_n), n \in [1, N]$. We denote X the set of the x_n and T the set of the t_n.

1. Compute $\vec{\nabla}_\Theta J$. You may first compute $\frac{\partial J}{\partial \theta_0}$, then compute $\frac{\partial J}{\partial \theta_1}$.

CORRECTION Using $\vec{\nabla}_x (u(x))^2 = 2u(x)\vec{\nabla}_x u(x)$, we can do it fast:

$$\vec{\nabla}_\theta J(\theta, X) = \frac{1}{N}\sum_n^N \vec{\nabla}_\theta (f_\theta(x_n) - t_n)^2 \tag{3}$$

$$= \frac{2}{N}\sum_n^N (f_\theta(x_n) - t_n)\vec{\nabla}_\theta (f_\theta(x_n) - t_n) \tag{4}$$

$$= \frac{2}{N}\sum_n^N (f_\theta(x_n) - t_n)\begin{pmatrix} \frac{\partial}{\partial \theta_0}(\theta_0 + \theta_1 x_n - t_n) \\ \frac{\partial}{\partial \theta_1}(\theta_0 + \theta_1 x_n - t_n) \end{pmatrix} \tag{5}$$

$$= \frac{2}{N}\sum_n^N (f_\theta(x_n) - t_n)\begin{pmatrix} 1 \\ x_n \end{pmatrix} \tag{6}$$

$$= \frac{2}{N}\sum_n^N (\theta_0 + \theta_1 x_n - t_n)\begin{pmatrix} 1 \\ x_n \end{pmatrix} \tag{7}$$

$$\tag{8}$$

In the slow way, we can simply do the 2 cases, computing first $\frac{\partial J}{\partial \theta_0}$, then $\frac{\partial J}{\partial \theta_1}$, and writing in the end that $\vec{\nabla}_\theta J(\theta, X) = \begin{pmatrix} \frac{\partial J}{\partial \theta_0} \\ \frac{\partial J}{\partial \theta_1} \end{pmatrix}$. It goes:

$$\partial_{\theta_0} \frac{1}{N}\sum_n^N \vec{(} f_\theta(x_n) - t_n)^2 = \partial_{\theta_0} \frac{1}{N}\sum_n^N \vec{(}\theta_0 + \theta_1 x_n - t_n)^2 \tag{9}$$

$$= 2\frac{1}{N}\sum_n^N \vec{(}\theta_0 + \theta_1 x_n - t_n)\partial_{\theta_0}(\theta_0 + \theta_1 x_n - t_n) \tag{10}$$

$$= 2\frac{1}{N}\sum_n^N \vec{(}\theta_0 + \theta_1 x_n - t_n)1 \tag{11}$$

$$\partial_{\theta_1} \frac{1}{N}\sum_n^N \vec{(} f_\theta(x_n) - t_n)^2 = \partial_{\theta_1} \frac{1}{N}\sum_n^N \vec{(}\theta_0 + \theta_1 x_n - t_n)^2 \tag{12}$$

$$= 2\frac{1}{N}\sum_n^N \vec{(}\theta_0 + \theta_1 x_n - t_n)\partial_{\theta_1}(\theta_0 + \theta_1 x_n - t_n) \tag{13}$$

$$= 2\frac{1}{N}\sum_n^N \vec{(}\theta_0 + \theta_1 x_n - t_n)x_n \tag{14}$$

2. Derive the update rule of the parameters, assuming we perform a Gradient Descent with learning rate η.

3. Write down on *a piece of paper*[1] (not code directly) the pseudo code of the learning algorithm (initialization, updates, stopping criterion, error computation).

4. Ok, now, you may start TP1.1-LinReg-1D.ipynb

5. What is the time complexity of this algorithm ? And space complexity ?

1.1.2 The "trick of the auxiliary ones" & Matrix formulation

We did ex. 1.1.1 in a rather manual, fastidious way.

But there is a very general trick which allows to re-cast the offset (the "b" in $ax+b$) into the other parameters, making equations –and code– much simpler. The idea is the following:

$$b + ax = \begin{pmatrix} b \\ a \end{pmatrix} \cdot \begin{pmatrix} 1 \\ x \end{pmatrix} \tag{15}$$

$$= \vec{\theta} \cdot \vec{x}' \tag{16}$$

Where $\vec{x}' = \begin{pmatrix} 1 \\ x \end{pmatrix}$ is the augmented x (with auxiliary ones), and you can guess that $\vec{\theta} = \begin{pmatrix} b \\ a \end{pmatrix}$. And more generally, for a vector $\vec{x} \in \mathbb{R}^d$,

$$b + \vec{a} \cdot \vec{x} = \begin{pmatrix} b \\ a_1 \\ a_2 \\ \dots \\ a_d \end{pmatrix} \cdot \begin{pmatrix} 1 \\ x_1 \\ x_2 \\ \dots \\ x_d \end{pmatrix} \tag{17}$$

$$= \vec{\theta} \cdot \vec{x}' \tag{18}$$

Where $\vec{x}' = \begin{pmatrix} 1 \\ \vec{x} \end{pmatrix}$ is the augmented x (with auxiliary ones), and you can guess that $\vec{\theta} = \begin{pmatrix} b \\ \vec{a} \end{pmatrix}$.

Why do we say "ones" and not just "a one" ? Because the idea is to augment the entire raw data X with a "1" for each data point, and then use this "new data" as your effective data (and for this new data, you don't need an offset –the "b term– in your equations). For a data set X:

$$X \in \mathbb{R}^{N,d} = \begin{pmatrix} \vec{x}_1^T \\ \vec{x}_2^T \\ \dots \\ \vec{x}_N^T \end{pmatrix} = \begin{pmatrix} x_{11} & x_{12} & \dots & x_{1d} \\ x_{21} & x_{22} & \dots & x_{2d} \\ \dots & \dots & \dots & \dots \\ x_{N1} & x_{N2} & \dots & x_{Nd} \end{pmatrix} \tag{19}$$

The new data set X' with the auxiliary ones is:

$$X' \in \mathbb{R}^{N,d+1} = \begin{pmatrix} \vec{x}'_1 \\ \vec{x}'_2 \\ \dots \\ \vec{x}'_N \end{pmatrix} = \begin{pmatrix} 1 & x_{11} & x_{12} & \dots & x_{1d} \\ 1 & x_{21} & x_{22} & \dots & x_{2d} \\ \dots & \dots & \dots & \dots & \dots \\ 1 & x_{N1} & x_{N2} & \dots & x_{Nd} \end{pmatrix} \tag{20}$$

Where we omit some of the transpose operators $(.^T)$ for readability.

Questions:

1. Re-write the solution of ex. 1.1.1 using this trick. In particular, re-write the gradient of J and the update of weights for the gradient descent, referring only to the new vectors $\vec{x}'$ instead of the original $\vec{x}$.

CORRECTION

$$\nabla_\theta J(\theta, X) = \frac{2}{N} \sum_n^N (\theta_0 + \theta_1 x_n - t_n) \begin{pmatrix} 1 \\ x_n \end{pmatrix} \tag{21}$$

$$= \frac{2}{N} \sum_n^N (\vec{\theta} \cdot \vec{x}'_n - t_n) \vec{x}'_n \tag{22}$$

2. Use this elegant formulation to rewrite the update step without using any explicit sum (instead, making it a purely algebraic computation, i.e. with matrices and vectors only). To help you, here is an example: If I have the sum $S = \sum_n^N p_n q_n$, I can rewrite it by defining hte appropriate vctors $\vec{p} \in \mathbb{R}^N, \vec{q} \in \mathbb{R}^N$, so that $S = \vec{p} \cdot \vec{q}$. If each $p_n \in \mathbb{R}$, but e.g. $q_n \in \mathbb{R}^k$, then I will go from $\vec{S} = \sum_n p_n \vec{q}_n \in \mathbb{R}^k$ to $\vec{S} = P \cdot Q \in \mathbb{R}^k$, where $P \in \mathbb{R}^N, Q \in \mathbb{R}^{N,k}$.

[1] It doesn't have to be litterally paper, you can use your tablet and stylus if you prefer.

CORRECTION - the model f is : $f_\theta(x_n) = \theta_0 + \theta_1 x_n$. *- the loss function that we were told in the Lecture is the squared Loss, i.e.* $J = \frac{1}{N}\sum_n^N (f_\theta(x_n) - t_n)^2$ *- We want to compute the gradient* $\nabla_\theta J(\theta, X)$ *- We do it (see your lecture notes for lots of details):*

$$\nabla_\theta J(\theta, X) = \frac{2}{N}\sum_n^N (\vec{\theta}\cdot\vec{x}'_n - t_n)\vec{x}'_n \tag{23}$$

$$= \frac{2}{N}(X'\theta - T)X \tag{24}$$

$$= \frac{2}{N}(X'_{N,d+1}\theta_{d+1} - T_N)X_{N,d+1} \tag{25}$$

Where, in the last line, we put the dimensions of each array as subscript, to make it clear that all our operations are legal matrix multiplications. Note how here, $X'_{N,d+1}\theta_{d+1} \neq \theta_{d+1}X'_{N,d+1}$ *(the second term is an illegal operation).*
Another re-writing could be : $\nabla_\theta J(\theta, X) = \frac{2}{N}(X')^T_{d+1,N}(\theta_{d+1}(X')^T_{d+1,N} - T_N)$. *The only difference being in the choice of whether* $\nabla_\theta J(\theta, X)$ *is a row-vector or a column-vector (i.e. size* (N, d) *or* $(1, d)$

3. Ideally, you may even derive the gradient directly from the matrix notation (but you need to be sure of the rules to be applied.. they are not always trivial).

Note: in python, vectorial/matricial operations using numpy are about 100 to 200 times faster than simple python loops. So a purely matricial formulation will be way faster that one with a loop !

Be careful that in python, you may have to explicit the shape of vectors like $T \in \mathbb{R}^N$ as being matrices of size $T \in \mathbb{R}^{N,1}$, which is a different kind of object. In pen and paper exercises, we typically do not make any difference between vectors $\in \mathbb{R}^N$ and single-column matrices $\in \mathbb{R}^{N,1}$.