Sciences-U **Introduction to Machine Learning**
2018 – 2019 Fabien Baradel

Gradient Descent and Machine Learing

Machine Learning (ML) approaches need to estimate parameters using an optimization technics. Gradient Descent (GD) algorithm is the most widely used optimization technic nowadays for solving ML problems. Today we will discover how does it work and how is it used in practice.

Exercise 1: Understanding Gradient Descent

What is the purpose of Gradient Descent? GD is the easiest approach for solving optimization problem. The only one hypothesis needed is to be able to compute gradients of the parameters we want to optimize. We want to find the value θ^* which is given the minimal value $J(\theta)$ where in our case $J(\theta) = (\theta + 1)^2$. This could be defined by the following optimization problem $\theta^* = \underset{\theta}{\operatorname{argmin}} J(\theta)$. For the function $(\theta + 1)^2$ we can compute by hand the solution i.e $\theta^* = -1$. However in many real world scenarios this solution cannot be found analytically and we need to estimate this quantity using optimization technic such as the Gradient Descent algorithm.

1. Create a function `J(theta_i)` which is implementing the function $J(\theta)$ described above What is the value of `J(4)` ?
2. Generate data point between -10 and 10 with a step of 0.01 between each data point and store it into a list called `list_theta`. What is the length of `list_theta`? Note: Use `np.arange()` and then `list()` to make sure you are creating a list.
3. Compute all the values $J(\theta)$ using values from the list `theta`. Store the values into a list called `list_J_theta`.
4. Plot `list_theta` vs `list_J_theta`. When do we reach the minimum?
5. Create a function `dJ_dtheta(x_i)` which is computing the gradient $\frac{\partial J}{\partial \theta}(x_i)$. What is the value of `dJ_theta(0)` ?
6. Create a function `gradient_descent(theta_0, lr, nb_iters, dJ_dtheta, J)` which returns the solution of $\underset{\theta}{\operatorname{argmin}} f(\theta)$. `theta_0, lr, nb_iters, dJ_dtheta, J` correspond respectively to the initial value of θ, the learning rate, the number of iterations allowed for solving the problem, the gradient of θ and the function $J(.)$. What is the solution $\hat{\theta}$ found by gradient descent to our problem? Note: assume that `x_0, lr, nb_iters, dJ_dtheta, J = -7, 0.1, 100, dJ_dtheta, J` while debugging.
7. Update your function `gradient_descent` for printing the optimization path (i.e. print the line between θ_t and θ_{t+1} and saving the figure at the end of the optimization process).
8. Assuming that you are setting `nb_iters` equals to 100 what is the solution $\hat{\theta}$ when varying `nb_iters` in `[-8, -1, 7]` and `lr` in `[0.1, 0.01, 0.001, -0.01, 0.8, 1.01]`. Does the estimated solution always the same? What are pros and cons about these hyperparameters and GD in general?
9. Now assume the function $J(\theta) = \sin(2\theta + 1)$, what is the solution $\hat{\theta}$ given by GD?

Exercise 2: Simple Linear Regression with Gradient Descent

We want to estimate parameters of a simple linear regression (i.e. $y_i = wx_i + b$) by closed-form and GD. The loss function on teh full dataset is defined by $J(w, b) = \sum_{i=1}^{N}((wx_i + b) - y_i)^2$. The optimization problem we want to solve is $\min_{w,b} J(w, b)$. We assume the following data generation process: $Y \sim wX + b + \epsilon$ where $X \sim \mathcal{U}[20; 40]$ and $\epsilon \sim \mathcal{N}(0, 1)$.

1. Generate N data points (x_i, y_i) using the data generation process given above and store them into `list_x` vs `list_y`. Note: $N = 100$ and start by first generating $\{x_i\}_1^N$ and then $\{y_i\}_1^N$. Do not forget to add noise!
2. Plot the data points `list_x` vs `list_y`.

3. Estimate parameters of the simple linear regression by closed-form. Note: first create `X` (ones + `list_x`) and `Y` and then do the matrix multiplications step by step. Note: store these values into `w_cf` and `b_cf`.

4. Modify the distribution of the noise (e.g. $\epsilon \sim \mathcal{N}(0, 5)$). How does it impact the estimated parameters?

5. Remove the noise from the data generation process. What are the values $\hat{w}$ and $\hat{b}$? And why?

6. Create a function `predict(x,w,b)` which is returning the estimated value $\hat{y}_i$ for a data point x_i. What the value of `predict(5,2,-1)` ?

7. Plot the fitted line on the data points.

8. Create a function `loss(list_x, list_y, w, b)` which is computing the regression loss on the full dataset. What is the value of `loss([1], [3], 1, 2)` ?

9. Create a function `dl_dw(x, y, w, b)` which is computing $\frac{\partial l}{\partial w}(w, b)$ on the single point (x, y). What the values of `dl_dw(2, 1, 0, 0)` and `dl_dw(2, 2, 0, 0)` ?

10. Create a similar function `dl_db(x, y, w, b)` for computing the gradient $\frac{\partial l}{\partial b}$ on a single data point (x, y). What the value of `dl_db(4, -1, 0, 0)` and `dl_db(3, 3, 0, 0)` ?

11. Implement a function `update_w_and_b(list_x, list_x, w, b, lr)` which is updating `w` and `b` according to the gradient compute on the full data points. What is the output of the following command of `update_w_and_b([0], [3], 5, 3, 0.1)` and why?

12. Estimate parameters of the simple linear regression by gradient descent from a random initialization of w and b and with a learning rate equals to 0.001. Note: store these values into `w_gd` and `b_gd`.

13. Plot the fitted line on the data points and compared against the one estimated by closed-form.

14. In real world scenarios it takes a lot of time for computing gradients on the full dataset (e.g. millions of examples). So we estimate gradients from a sample of the full data. This technic is called Stochastic Gradient Descent. Modify your algorithm according to these approach. How are your estimated parameters different compared to the Gradient Descent technic?

15. Repeat the Stochastic Gradient Descent algorithm. Do you find the same estimated parameters with the same hyperparameters? Why?

16. Use the library `scikit-learn` for solving this problem using `LinearRegression()`. Do you find the same estimated parameters?