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 `theta`.
3. Plot `theta` vs `J(theta)`. When do we reach the minimum?
4. Create a function `dJ_dtheta(x_i)` which is computing the gradient $\frac{\partial J}{\partial \theta}$. What is the value of `dJ_theta(0)` ?
5. Create a function `gradient_descent(theta_0, lr, nb_iters, df_dx, f)` which returns the solution of $\underset{\theta}{\operatorname{argmin}} f(\theta)$. `x_0, lr, nb_iters, df_dx, f` 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 = -7, 0.1, 100` while debugging.
6. 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).
7. 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?
8. 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 ($y_i = wx_i + b$) by closed-form and GD. The loss function is defined on a data point by $l(\hat{y}_i, y_i) = (\hat{y}_i - y_i)^2$ where the prediction is given by $\hat{y}_i = \hat{w}x + \hat{b}$. The optimization problem we want to solve is $\min_{w,b} \sum_{i=1}^{N} l(\hat{y}_i, y_i)$. 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 `x` vs `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 `x` vs `y`.
3. Estimate parameters of the simple linear regression by closed-form. 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_i,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(x_i, y_i, 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_i, y_i, w, b)` which is computing $\frac{\partial l}{\partial w}$. What the values of `dl_dw(2, 1)` and `dl_dw(2, 2)` ?
10. Create a similar function `dl_db(x_i, y_i, w, b)` for computing the gradient $\frac{\partial l}{\partial b}$. What the value of `dl_db(4, -1)` and `dl_db(3, 3)` ?
11. Implement a function `update_w_and_b(x_i, y_i, 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 leanring 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?

Exercise 2: Linear Regression on Boston Dataset

The Boston dataset is composed of 506 instances and 14 columns (13 features and 1 target variable). The target variable corresponds to the median value of owner-occupied homes in $1000s. We are going to use this real dataset for moving to linear regression (i.e. more than one features for explaining the target variable)s.

1. Load the Boston dataset. We consider that the first 300 examples are our training set, the next 100 are our validation set and the remaining examples are the test set.
2. What is the range of the target variable? Describe the target variable?
3. Find three different ways of normalizing the target variable and write the their associated functions.
4. Right now we will be using only the first features called `CRIM` for modelling the target variable. Plot `CRIM` vs `target` with and without normalizing your data. What do you observe?
5. Use `LinearRegression()` for modelling `target` with `CRIM` on the training set and compute the predicted values for the validation set.
6. Implement a function which is computing the Root-Mean-Square-Error (RMSE). What is the RMSE on the training set and on the validation set?
7. Improve your performance on the validation by using normalization technics and regularization tricks. Do not forget to plot your fitted line vs the ground-truth target for the validation set to check how good you are.
8. Add an another variable for modelling `target` (e.g. `RAD`). Do you get a better score on the validation set?
9. Repeat the process with the other variables. What is the best model you have found?
10. You have probably overfitted on the validation set. Implement a 5-cross-fold validation to make sure you are selecting the best model.

Exercise 3: Simple Logistic Regression with Gradient Descent

Without using `scikit_learn` implement a simple logistic regression algorithm with Gradient Descent using the Sigmoid function. Reminder: we are using the following modelling - $P(y_i = 1|x_i; w, b) = p_{w,b}(x_i) = \frac{1}{1+e^{(-wx_i+b)}}$ The first thing to do is to compute by hand the gradients for w and b according to the loss function. In our case the loss function is equal to: $J(w, b) = -\frac{1}{N}\sum_{i=1}^{N} y_i log(p_{w,b}(x_i)) + (1 - y_i)log(1 - p_{w,b}(x_i))$