Labs
Machine Learning Course
Fall 2019

EPFL
School of Computer and Communication Sciences
Martin Jaggi & Rüdiger Urbanke
www.epfl.ch/labs/mlo/machine-learning-cs-433

Problem Set 2, Sept 26, 2019 (Linear Regression and Gradient Descent)

Goals. The goal of this week's lab is to

- Implement grid search, gradient descent and stochastic gradient descent.
- Learn to debug your implementations.
- Learn to visualize results.
- Understand advantages and disadvantages of these algorithms.
- Study the effect of outliers using MSE and MAE cost functions.

Setup, data and sample code. Obtain the folder `labs/ex02` of the course github repository

github.com/epfml/ML_course

We will use the dataset `height_weight_genders.csv` in this exercise, and we have provided sample code templates that already contain useful snippets of code required for this exercise.

You will be working in the notebook `ex02.ipynb` for all exercises of this week, by filling in the corresponding functions. The notebook already provides a lot of template code, as well as code to load the data, and normalize the features, visualize the results.

Additionally, please also take a look at the files `helpers.py` and `plots.py`, and make sure you understand them.

1 Computing the Cost Function

In this exercise, we will focus on simple linear regression which takes the following form,

$$y_n \approx f(x_{n1}) = w_0 + w_1 x_{n1}. \tag{1}$$

We will use height as the input variable x_{n1} and weight as the output variable y_n. The coefficients w_0 and w_1 are also called *model parameters*. We will use a mean-square-error (MSE) function defined as follows,

$$\mathcal{L}(w_0, w_1) = \frac{1}{2N}\sum_{n=1}^{N}\big(y_n - f(x_{n1})\big)^2 = \frac{1}{2N}\sum_{n=1}^{N}\big(y_n - w_0 - w_1 x_{n1}\big)^2. \tag{2}$$

Our goal is to find $w_0^\star$ and $w_1^\star$ that minimize this *cost*.

Let us start by the array data type in *NumPy*. We store all the (y_n, x_{n1}) pairs in a vector and a matrix as shown below.

$$\boldsymbol{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix} \qquad \widetilde{\boldsymbol{X}} = \begin{bmatrix} 1 & x_{11} \\ 1 & x_{21} \\ \vdots & \vdots \\ 1 & x_{N1} \end{bmatrix} \tag{3}$$

Exercise 1:

To understand this data format, answer the following warmup questions:

- What does each *column* of $\tilde{\boldsymbol{X}}$ represent? The first column is for the constant w0 in the predicted value The second column is all the height data
- What does each *row* of $\tilde{\boldsymbol{X}}$ represent? Each row is a height data point
- Why do we have 1's in $\tilde{\boldsymbol{X}}$? For the constant w0 in the predicted value
- If we have heights and weights of 3 people, what would be the size of $\boldsymbol{y}$ and $\tilde{\boldsymbol{X}}$? What would $\tilde{\boldsymbol{X}}_{32}$ represent? (3,) and (3,2), it is the height of the third person
- In `helpers.py`, we have already provided code to form arrays for $\boldsymbol{y}$ and $\tilde{\boldsymbol{X}}$. Have a look at the code, and make sure you understand how they are constructed.
- Check if the sizes of the variables make sense (use functions `shape`).

a) Now we will compute the MSE. Let us introduce the vector notation $\boldsymbol{e} = \boldsymbol{y} - \tilde{\boldsymbol{X}}\boldsymbol{w}$, for given model parameters $\boldsymbol{w} = [w_0, w_1]^\top$. Prove that the MSE can also be rewritten in terms of the vector $\boldsymbol{e}$, as

$$\mathcal{L}(\boldsymbol{w}) = \ldots \tag{4}$$

b) Complete the implementation of the notebook function `compute_loss(y, tx, w)`. You can start by setting $\boldsymbol{w} = [1, 2]^\top$, and test your function.

2 Grid Search

Now we are ready to implement our first optimization algorithm: Grid Search. Revise the lecture notes.

Exercise 2:

a) Fill in the notebook function `grid_search(y, tx, w0, w1)` to implement grid search. You will have to write one for-loop per dimension, and compute the cost function for each setting of w_0 and w_1. Once you have all values of cost function stored in the variable `loss`, the code finds an approximate minimum (as discussed in the class).

The code should print the obtained minimum value of the cost function along with the found $w_0^\star$ and $w_1^\star$. It should also show a contour plot and the plot of the fit, as shown in Figure 1.

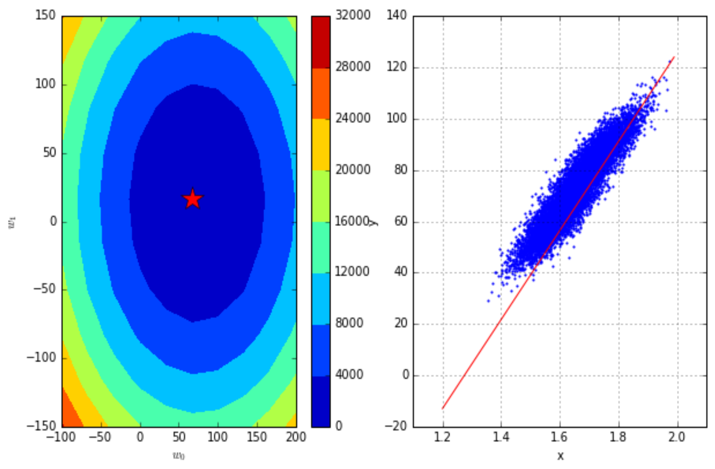

Figure 1: Grid Search Visualization

No, because the grid is not fine enough. We don't try enough parameters.
Because we only have finite number of paramters (10 for both w0 and w1)

b) Does this look like a good estimate? Why not? What is the problem? Why is the MSE plot not smooth?

Repeat the above exercise by changing the grid spacing to 10 instead of 50. Compare the new fit to the old one. It looked better.

c) Discuss with your peers:

- To obtain an accurate fit, do you need a coarse grid or a fine grid? Fine grid.
- Try different values of grid spacing. What do you observe?
- How does increasing the number of values affect the computational cost? How fast or slow does your code run? The larger the grid spacing, the better the fit.
  But we also have more computational cost.
  grid spacing = 10, time = 0.009 sec
  grid spacing = 50, time = 0.129 sec

3 Gradient Descent

In the lecture, we derived the following expressions for the gradient (the vector of partial derivatives) of the MSE for linear regression,

$$\frac{\partial \mathcal{L}(w_0, w_1)}{\partial w_0} = \frac{1}{N}\sum_{n=1}^{N}(y_n - w_0 - w_1 x_{n1}) = -\frac{1}{N}\sum_{n=1}^{N} e_n \tag{5}$$

$$\frac{\partial \mathcal{L}(w_0, w_1)}{\partial w_1} = \frac{1}{N}\sum_{i=1}^{N}(y_n - w_0 - w_1 x_{n1})x_{n1} = -\frac{1}{N}\sum_{n=1}^{N} e_n x_{n1} \tag{6}$$

Denoting the gradient by $\nabla\mathcal{L}(\boldsymbol{w})$, we can write these operations in vector form as follows,

$$\nabla\mathcal{L}(\boldsymbol{w}) := \begin{bmatrix} \frac{\partial \mathcal{L}(w_0,w_1)}{\partial w_0} & \frac{\partial \mathcal{L}(w_0,w_1)}{\partial w_1} \end{bmatrix} = -\frac{1}{N}\begin{bmatrix} \sum_{n=1}^{N} e_n \\ \sum_{n=1}^{N} e_n x_{n1} \end{bmatrix} = -\frac{1}{N}\tilde{\boldsymbol{X}}^\top \boldsymbol{e} \tag{7}$$

Exercise 3:

a) Now implement a function that computes the gradients. Implement the notebook function `compute_gradient(y, tx, w)` using Equation (7). Verify that the function returns the right values. First, manually compute the gradients for hand-picked values of $\boldsymbol{y}$, $\tilde{\boldsymbol{X}}$, and $\boldsymbol{w}$ and compare them to the output of `compute_gradient`.

b) Once you make sure that your gradient code is correct, get some intuition about the gradient values:

Compute the gradients for

- $w_0 = 100$ and $w_1 = 20$ Gradient = [26.706078 6.52028757]
- $w_0 = 50$ and $w_1 = 10$ Gradient = [-23.293922 -3.47971243]

What do the values of these gradients tell us? For example, think about the norm of this vector. In which case are they bigger? What does that mean? When the weight yield a greater loss, which means it is not a good estimate

Hint: Imagine a quadratic function and estimate its gradient near its minimum and far from it.

Hint 2: As we know from the lecture notes, the update rule for gradient descent at step t is

$$\boldsymbol{w}^{(t+1)} = \boldsymbol{w}^{(t)} - \gamma\nabla\mathcal{L}(\boldsymbol{w}^{(t)}) \tag{8}$$

where $\gamma > 0$ is the step size, and $\nabla\mathcal{L} \in \mathbb{R}^2$ is the gradient vector.

c) Fill in the notebook function `gradient_descent(y, tx, initial_w, ...)`. Run the code and visualize the iterations. Also, look at the printed messages that show $\mathcal{L}$ and values of $w_0^{(t)}$ and $w_1^{(t)}$. Take a detailed look at these plots,

- Is the cost being minimized? Yes
- Is the algorithm converging? What can be said about the convergence speed? Yes, it converge at the fifth iterations
- How good are the final values of w_1 and w_0 found? loss = 15.39

d) Now let's experiment with the value of the step size and initialization parameters and see how they influences the convergence. In theory, gradient descent converges to the optimum on convex functions, when the value of the step size is chosen appropriately.

- Try the following values of step size: 0.001, 0.01, 0.5, 1, 2, 2.5. What do you observe? Did the procedure converge? When step size is too small, like 0.001, and too big, like 2 or 2.5, the procedure won't converge
- Try different initializations with fixed step size $\gamma = 0.1$, for instance:
  - $w_0 = 0,\ w_1 = 0$
  - $w_0 = 100,\ w_1 = 10$
  - $w_0 = -1000,\ w_1 = 1000$

  What do you observe? Did the procedure converge?

  The third doesn't converge. It's too far away from the ideal fit

4 Stochastic Gradient Descent

Exercise 4:

Let us implement stochastic gradient descent. Recall from the lecture notes that the update rule for stochastic gradient descent on an objective function $\mathcal{L}(\boldsymbol{w}) = \frac{1}{N}\sum_{n=1}^{N}\mathcal{L}_n(\boldsymbol{w})$ at step t is

$$\boldsymbol{w}^{(t+1)} = \boldsymbol{w}^{(t)} - \gamma\,\nabla\mathcal{L}_n(\boldsymbol{w}^{(t)}) \ . \tag{9}$$

HINT: You can use the function `batch_iter()` in the file of `helpers.py` to generate mini-batch data for stochastic gradient descent.

5 Effect of Outliers and MAE Cost Function

In the course we talked about *outliers*. Outliers might occur due to measurement errors. For example, in the weight/height data, a coding mistake could introduce points whose weight is measured in pounds rather than kilograms.

Such outlier points may have a strong influence on model parameters. For example, MSE (the one you implemented above) is known to be sensitive to outliers, as discussed in the class.

Exercise 5:

Let's simulate the presence of two outliers, and their effect on linear regression under MSE cost function,

- Reload the data through function `load_data()` by setting `sub_sample=True` to keep only a few data examples.
- Plot the data. You should get a cloud of points similar, but less dense, than what you saw before with the whole dataset.
- As before, find the values of w_0, w_1 to fit a linear model (using MSE cost function), and plot the resulting f together with the data points.
- Now we will add two outliers points simulating the mistake that we entered the weights in pounds instead of kilograms. For example, you can achieve this by setting `add_outlier=True` in `load_data()`. Feel free to add more outlier points.
- Fit the model again to the augmented dataset with the outliers. Does it look like a good fit?

It is worse than the one without outliers

One way to deal with outliers is to use a more *robust* cost function, such as the Mean Absolute Error (MAE), as discussed in the class.

6 Subgradient Descent

Exercise 6:

Modify the function `compute_loss(y, tx, w)` for the Mean Absolute Error cost function.

Unfortunately, you cannot directly use gradient descent, since the MAE function is non-differentiable at several points.

a) Compute a subgradient of the MAE cost function, for every given vector $\boldsymbol{w}$.

   Hint: Use the chain rule to compute the subgradient of the absolute value function. For a function $\mathcal{L}(\boldsymbol{w}) := h(q(\boldsymbol{w}))$ *with* q *differentiable, the subgradient can be computed using* $\partial\mathcal{L}(\boldsymbol{w}) = \partial h(q(\boldsymbol{w}))\cdot\nabla q(\boldsymbol{w})$*, where each* $\partial..$ *denotes the set of all subgradient vectors.*

b) Implement subgradient descent for the MAE cost function.

   To do so, write a new function `compute_gradient(y, tx, w)` for the new MAE objective, and modify it to return a subgradient if the given $\boldsymbol{w}$ turns out to be a non-differentiable point.

   Plot the resulting model f together with the two curves obtained in the previous exercise.

   - Is the fit using MAE *better* than the one using MSE? Yes
   - Did your optimization algorithm ever encounter a non-differentiable point?

c) Implement stochastic subgradient descent (SGD) for the MAE cost function.

   How is the picture different when you compare the two algorithm variants on MAE, compared to what you have observed on MSE?

Wrap-Up

After you have finished the implementation of the above exercises the notebook `ex02.ipynb`, you can wrap up by copying your code to separate `.py` files for later re-use. For example, you'll be re-using your code from this week later on, for example for Project 1 and some of the subsequent labs.

We have provided template files for this, namely
`cost.py`, `grid_search.py`, `gradient_descent.py` and `stochastic_gradient_descent.py`,