

Gradient Descent and Simple Linear Regression

Logic and Python tutorial

Sophie Marchand

May 2020

1 Shortcut

Gradient descent is an optimization algorithm employed to find the parameters \mathbf{v} of a function f minimizing a cost function C . Its iterative procedure is as follow:

1. Initialize parameters to random small values
2. Calculate the cost function C over the all training data
3. Compute the update of the parameters \mathbf{v} with $\mathbf{v} - \eta \nabla C(\mathbf{v})$ with η the learning rate
4. Repeat the steps 2 and 3 until reaching "good" enough parameters

This procedure is for one pass of the batch gradient descent. For the stochastic one, step 1 includes a randomization of the training data and step 2 is performed over one instance selected according to the algorithm iteration. Then, the steps 2 and 3 are performed for each randomized training input.

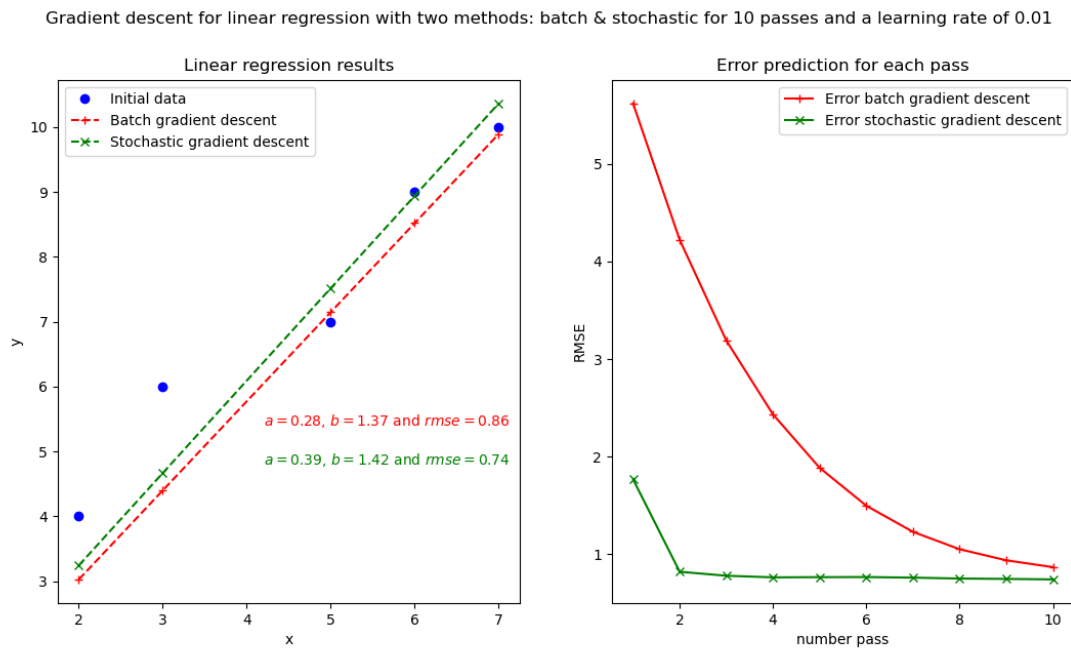


Figure 1: Output of the tutorial in section 3

2 Logic details

We show here the logic behind the gradient descent applied to a simple linear regression problem. The objective of this regression is to find the optimal slope b and intercept a which verify $y = a + b \times x$ while minimizing the prediction error for a set of n points (x, y) . In this work, the error function chosen is the Sum of Squared Residuals (SSR) defined by equation (1) where \mathbf{v} is the vector of coefficients $\begin{pmatrix} a \\ b \end{pmatrix}$ and y_{pred} the predicted output variable.

$$SSR(\mathbf{v}) = \frac{1}{2n} \sum_{i=1}^n (y_{pred}(\mathbf{v}, i) - y_i)^2 \quad (1)$$

Using the Taylor series expansion on $SSR(\mathbf{v})$, we obtain the equation (2). Then, replacing ϵ by $-\eta \nabla SSR(\mathbf{v})$ with η a small positive value called learning rate, we have the expression (3).

$$SSR(\mathbf{v} + \epsilon) \approx SSR(\mathbf{v}) + \epsilon^T \nabla SSR(\mathbf{v}) \quad (2)$$

$$SSR(\mathbf{v} - \eta \nabla SSR(\mathbf{v})) \approx SSR(\mathbf{v}) - \eta \nabla SSR(\mathbf{v})^2 \leq SSR(\mathbf{v}) \quad (3)$$

We deduce from the previous expression that updating \mathbf{v} by $\mathbf{v} - \eta \nabla SSR(\mathbf{v})$ may reduce the value of $SSR(\mathbf{v})$. This is the logic adopted by the gradient descent method consisting in the following steps:

1. Initiate the values of \mathbf{v} to zero or small random values

[Stochastic gradient descent] Randomized the training data order giving the order array r

2. Compute the prediction error $(y_{pred}(\mathbf{v}, i) - y_i)$ for:

[Batch gradient descent] all training data \mathbf{i} before calculating the update

[Stochastic gradient descent] each training data instance \mathbf{i} and calculate the update immediately

3. Compute the update of \mathbf{v} with:

[Batch gradient descent] $i \subset [1, n]$

$$\begin{cases} a := a - \eta \frac{\partial SSR(\mathbf{v}, i)}{\partial a} = a - \frac{\eta}{n} \sum_{i=1}^n (y_{pred}(\mathbf{v}, i) - y_i) \\ b := b - \eta \frac{\partial SSR(\mathbf{v}, i)}{\partial b} = b - \frac{\eta}{n} \sum_{i=1}^n (y_{pred}(\mathbf{v}, i) - y_i) x_i \end{cases} \quad (4)$$

[Stochastic gradient descent] $i = r[j]$ with j the iteration of the gradient descent

$$\begin{cases} a := a - \eta \frac{\partial SSR(\mathbf{v}, r[j])}{\partial a} = a - \eta (y_{pred}(\mathbf{v}, r[j]) - y_{r[j]}) \\ b := b - \eta \frac{\partial SSR(\mathbf{v}, r[j])}{\partial b} = b - \eta (y_{pred}(\mathbf{v}, r[j]) - y_{r[j]}) x_{r[j]} \end{cases} \quad (5)$$

4. Repeat the steps 2 and 3 until reaching "good" enough coefficients. The performance threshold th_p could be defined as value on the Root Mean Square Error ($RMSE$) such that we should verify:

$$RMSE = \sqrt{\frac{\sum_{i=0}^n (y_i^{pred} - y_i)^2}{n}} < th_p \quad (6)$$

Remarks: the stochastic gradient descent is preferred to the batch one for large datasets. To note also that stochastic gradient descent will require a small number of passes through the dataset to reach "good" enough coefficients typically between 1-to-10 passes.

3 Python tutorial

The code source displayed below can be found on [GitHub](#) under `Python_GradientDescentLinearRegression.py`

```
1 """
2 Tutorial Gradient Descent for Linear Regression
3
4 Author: Sophie Marchand
5 """
6 import matplotlib.pyplot as plt
7 import numpy as np
8
9
10 # Function initialization parameters, update parameters and compute error
11 def init_parameters():
12     return 0, 0
13
14
15 def randomized_training_data(x, y):
16     index_array = np.arange(len(x))
17     np.random.shuffle(index_array)
18     return x[index_array], y[index_array]
19
20
21 def update_parameters_batch(a, b, x, y, learning_rate):
22     a_update = a - learning_rate * sum((a + b * x) - y) / len(x)
23     b_update = b - learning_rate * sum(((a + b * x) - y) * x) / len(x)
24     return a_update, b_update
25
26
27 def update_parameters_stochastic(a, b, x, y, learning_rate, iteration):
28     x_it = x[iteration]
29     y_it = y[iteration]
30     a_update = a - learning_rate * ((a + b * x_it) - y_it)
31     b_update = b - learning_rate * ((a + b * x_it) - y_it) * x_it
32     return a_update, b_update
33
34
35 def compute_error_rmse(a, b, x, y):
36     return np.sqrt(sum(np.square((a + b * x) - y)) / len(x))
37
38
39 # Create a set of points (x,y) and set a learning rate
40 x = np.array([2, 3, 5, 6, 7])
41 y = np.array([4, 6, 7, 9, 10])
42 learning_rate = 0.01
43 number_batch = 10
44 rmse_batch = []
45 rmse_stochastic = []
46
47 # Workflow batch gradient descent to estimate (a,b) parameters of y = a + b*x
48 a, b = init_parameters()
49 for batch in range(number_batch):
50     a_update, b_update = update_parameters_batch(a, b, x, y, learning_rate)
51     rmse_batch.append(compute_error_rmse(a_update, b_update, x, y))
52     a, b = a_update, b_update
53 a_batch, b_batch = a, b
54
55 # Workflow stochastic gradient descent to estimate (a,b) parameters of y = a + b*x
56 a, b = init_parameters()
57 for batch in range(number_batch):
58     x_rand, y_rand = randomized_training_data(x, y)
59     for iteration in range(len(x)):
60         a_update, b_update = update_parameters_stochastic(a, b, x_rand, y_rand,
61                                                         learning_rate, iteration)
62         a, b = a_update, b_update
63     rmse_stochastic.append(compute_error_rmse(a, b, x, y))
64 a_stochastic, b_stochastic = a, b
```

```

64
65 # Print results
66 y_batch = a_batch + b_batch * x
67 y_stochastic = a_stochastic + b_stochastic * x
68
69 fig, (ax1, ax2) = plt.subplots(1, 2)
70 fig.suptitle('Gradient descent for linear regression with two methods: batch & stochastic
    for '
71             + str(number_batch) + ' passes and a learning rate of ' + str(learning_rate),
    fontsize=12)
72 ax1.plot(x, y, 'bo', label='Initial data')
73 ax1.plot(x, y_batch, linestyle='--', marker='+',
74         color='r', label='Batch gradient descent')
75 ax1.plot(x, y_stochastic, linestyle='--', marker='x',
76         color='g', label='Stochastic gradient descent')
77 ax1.set(xlabel='x', ylabel='y',
78         title='Linear regression results')
79 ax1.legend()
80 ax1.text(4.2, 5.4, r'$a=$' + str(a_batch)[:4] +
81         ', $b=$' + str(b_batch)[:4] + ' and $rmse=$' + str(rmse_batch[-1])[:4], fontsize
    =10, color='r')
82 ax1.text(4.2, 4.8, r'$a=$' + str(a_stochastic)[:4] + ', $b=$' + str(b_stochastic)[:4] +
83         ' and $rmse=$' + str(rmse_stochastic[-1])[:4], fontsize=10, color='g')
84 ax2.plot(range(1, len(rmse_batch)+1), rmse_batch, 'r+-', label='Error batch gradient descent
    ')
85 ax2.plot(range(1, len(rmse_stochastic)+1), rmse_stochastic, 'gx-', label='Error stochastic
    gradient descent')
86 ax2.set(xlabel='number pass', ylabel='RMSE',
87         title='Error prediction for each pass')
88 ax2.legend()
89 plt.show()

```