Gradient Descent for Risk Optimization

Michael

1 Intuition Behind Gradient Descent

Although the term 'gradient descent' is very fancy, the idea and intuition behind this term is extremely simple. It is same for *convergence rate*. All you need to know is that what is the meaning of derivative.

The **derivative** of a function of a real variable measures the sensitivity to change of the function value (output value) with respect to a change in its argument (input value). Let's draw a table for this definition.

Table 1.1: Derivative

Measures	Input value	Output value
Sensitivity to	Change	Change
↓	↓	↓
	change rate	change rate
	↓	↓
	Convergence Rate	

If you got the idea of table 1.1, you can stop reading the section 1 and jump to section 2 and 3. If not, allow me to spend a little more time to explain this.

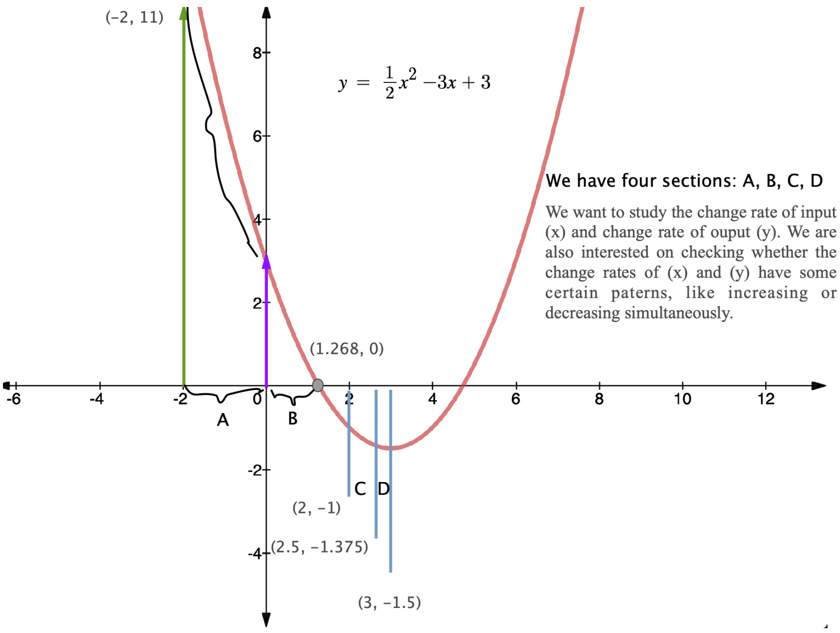


Figure 1.1: Gradient Visualisation

Again, the derivative is not just about the measurement of change of input and output, it also reflects the sensitivity of those changes. Take a look the figure 1.1, tell me where both input (x) and output (y) are changing fast, and where both input (x) and output (y) are changing slowly[1].

From the figure 1.1, we can see that in section A, both input (x) and output (y) are changing quite fast by measuring the differences. When it comes to section C and D, the changing rates for input-x and output-y are decreasing simultaneously. We also say that input x and output y have the same **convergence rate** intuitively[2]. For the **supervised learning** in *machine learning*, it starts to do optimization with the tool of convergence rate.

Now, let's check the formal definition of derivative. The slope m of the secant line is the difference between the y values of these points divided by the difference between the x values, that is,

$$m = \frac{\Delta f(x)}{\Delta x} = \frac{f(x+h) - f(x)}{(x+h) - x} = \frac{f(x+h) - f(x)}{h}$$

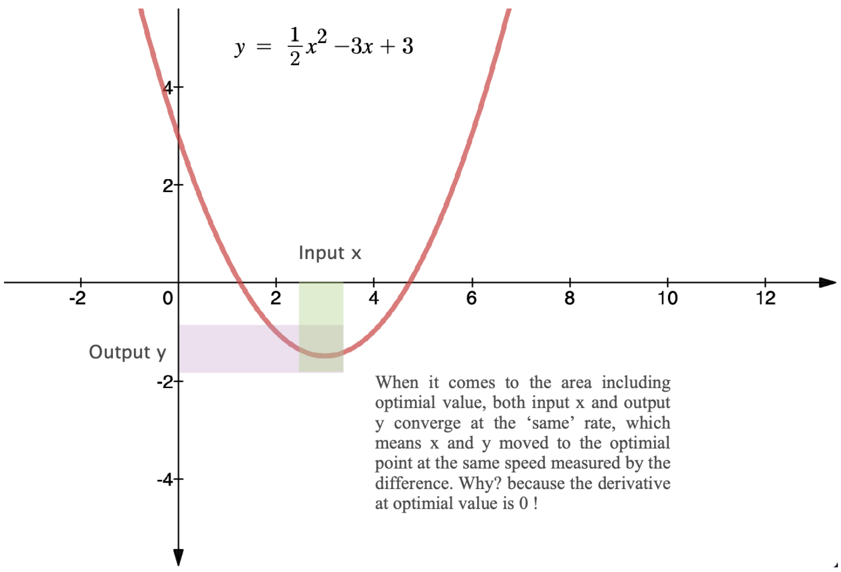
Geometrically, the limit of the secant lines is the tangent line. Therefore, the limit of the difference quotient as h approaches zero, if it exists, should represent the slope of the tangent line to $(x, f(x))$. This limit is defined to be the derivative of the function f at x:

$$f'(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

With the riview of defintion of derivative, we are ready to apply the so called **gradient descent** algorithm to find the minimum of a function.

Definition 1.1. Gradient descent is a first-order iterative optimization algorithm for finding the minimum of a function.

Let's disentangle the convergence rate for the function $y = \frac{1}{2}x^2 - 3x + 3$ in figure 1.1 by presenting the following figure and table:



[1]If we assume that speed of changing is measured by the difference between two values.
[2]This is not formal mathematical definition.

Table 1.2: Show the gradient descent step by step

Section	Points	Change of input(x)	Change of ouput (y)	Convergence ratio
A	$(-2, 11)$			
	$(0, 0)$	2	11	$11/2 = 5.5$
B	$(0, 0)$			
	(2,-1)	2	1	$1/2 = 0.5$
C	(2,-1)			
	(2.5,-1.375)	0.5	0.375	$0.375/0.5 = 0.75$
D	(2.5,-1.375)			
	(3,-1.5)	0.5	0.125	$0.125/0.5 = 0.25$
		...		
	(2.9,-1.49499)			
	(2.95,-1.49875)	0.05	0.00376	$0.00376/0.05 = 0.075$
		...		
	(2.98,-1.499799)			
	(2.99,-1.499950)	0.01	0.000151	$0.00376/0.05 = 0.00151$
we have learning rate α to scale the convergence rate $= 1$				

Markup: it's all about simultaneous moving of input (x) and (y).

2 A Mathematical example

Now, we will employ the gradient descent[3] to find the minimum value for the function in section 1:

$$f(x) = \frac{1}{2}x^2 - 3x + 3$$

The intuition of using gradient descent is that we are trying to iterate the change rate of input x by tracing the change rate of output y. To do this, we need the help from the derivative of the function. Here is the Python code:

```python
# Gradient Descent: mathematical example
# @ Michael


# define function
def qudraticfun(x):
    y = 1/2 * x**2 - 3 * x + 3
    return(y)


# define the derivative
def qudraticder(x):
    yprime = x - 3
    return(yprime)


# find the minium numerically
```

[3]Review the definition, it's an algorithm.

```
update_x = 0  # most time we start it from zero
alpha = 0.01  # learning rate
tolerate_rule = 1  # initialize the tolerate rule for stopping iterating
max_iters = 10000  # set the maximum iterative number

i = 0  # interation counting index
while tolerate_rule >= 0.00001 and i <= max_iters:
    start_x = update_x  # set the starting value
    update_x = start_x - alpha * qudraticder(start_x)
    tolerate_rule = abs(update_x - start_x)

print("The minimum value is", qudraticfun(update_x),
      "when x is equal to", update_x)

# The minimum value is -1.4999995136117308 when x is equal to
  2.999013705653870
```

Listing 1: Gradient Descent Math Example

3 A Machine Learning Example

For most students who have done econometric 101 or statistics 101, least squares estimation (LSE) and maximum likelihood estimation (MLE) are not unusual. Later, we can show that LSE and MLE are just one of special cases among gradient descent methods. In the section, we will apply gradient descent for the linear regression in one variable. Then the multivariable cases will be given. All examples are from Ng (2014), unless otherwise stated.

Unlike the mathematical example in section 2, we do not have the given function to optimize. Therefore, we need define the function for doing optimization. In machine learning, this function is called the **cost function** or **loss function**. In machine learning, we still have *input* and *output* like in the following table:

Table 3.1: Input and Output of Machine Learning

Input	Function	Output
Training data X	Regression or Classification	Estimated coefficients θ
Matrix form		Vector or matrix
Measure	convergence rate between θ and loss function	

We will use `Ex1data.csv` to run one variable regression. The first column is the population of a city and the second column is the profit of a food truck in that city. A negative value for profit indicates a loss. The first column refers to the population size in 10,000s and the second column refers to the profit in $10,000s.

Row Index	Population	Profit
0	6.1101	17.5920
1	5.5277	9.1302
2	8.5186	13.6620
3	7.0032	11.8540
4	5.8598	6.8233
$\vdots$		

Now, we must decide how we are going to represent the regression function f. As an initial choice, let's say we decide to approximate y as a linear function of x:

$$f(x) = \theta_0 + \theta_1 x \tag{3.1}$$

Here, the θ_i's are the **parameters** (also called **weights**). In this function, we set $x_0 = 1$ to make the intercept become θ_0. Okay, how to we find the optimal parameters θ? We need fist set the cost (loss) functions, and then choose θ to minimize the cost (loss) functions.

There are several loss (cost) functions we can use. Here is the list:

(i) Squared Error Loss: $\frac{1}{2}[y - f(x)]^2$

(ii) Absolute Error Loss: $|y - f(x)|$

(iii) Huber's Loss: $\begin{cases} \frac{1}{2}[y - f(x)]^2 & if|y - f(x)| \leq \delta \\ \delta[|y - f(x)| - \frac{1}{2}\delta] & otherwise \end{cases}$

Let's use the squared error loss function, which we can write it as

$$J(\theta) = \frac{1}{2}\sum_{i=1}^{n}[f(x^i) - y^i]^2 \tag{3.2}$$

where n is the length of dataset or size of vector y (or x). Our task is to minimize the equation (3.2). If you have read the section 2 carefully, then you will realise that the first step is to write down it's derivative:

$$\begin{aligned} \frac{\partial}{\partial\theta_j}J(\theta) &= \frac{\partial}{\partial\theta_j}\frac{1}{2}\sum_{i=1}^{n}[f(x^i) - y^i]^2 \\ &= 1 \cdot \frac{1}{2}\sum_{i=1}^{n}[f(x) - y]\frac{\partial}{\partial\theta_j}[f(x) - y] && \text{(chain rule)} \end{aligned}$$

Now, substitute equation (3.1) into the above general case, we can have:

$$\frac{\partial}{\partial\theta_0}J(\theta) = [f(x) - y]\theta_0 \tag{3.3}$$

$$\frac{\partial}{\partial\theta_1}J(\theta) = [f(x) - y]\theta_1 \tag{3.4}$$

Then we can apply the gradient descent with derivative in equation (3.3) and (3.4) to update the parameters:

$$\theta_j = \theta_j - \alpha \sum_{i=1}^{n} [f(x) - y]\theta_j \quad \text{(for every j)} \tag{3.5}$$

Does the equation (3.5) looks familiar to you? It should be if you have done the python code in listing 1 (page 4). The following is the regression code with gradient descent.

```python
# Gradient descent: regression example
# @ Michael

import pandas as pd
import numpy as np
import os
import matplotlib.pyplot as plt


# check the working directory
os.getcwd()
os.chdir('/Users/Michael/Documents/MachineLearning/GraidentDescent')

# read the dataset
ex1data = pd.read_csv('Ex1data.csv', names=['Population', 'Profit'])

# explore the dataset
ex1data.head()
ex1data.columns
ex1data.shape  # (97, 2)

fig, ax = plt.subplots(figsize=(6, 5))
ax.scatter(ex1data.Population, ex1data.Profit, marker='x')
ax.set(xlabel='Population of City in 10,000s',
       ylabel='Profit in $10,000s',
       title='Plot of X and Y')
fig.show()


# run regression: use population to predict profit

# set x
datamatrix = np.asmatrix(ex1data['Population']).transpose()
datamatrix.shape
input_x = np.hstack([np.ones(ex1data.shape[0]).reshape(-1, 1), datamatrix
  ])
input_x.shape  # check the matrix shape, (97, 2)
input_y = np.asmatrix(ex1data.Profit).transpose()
input_y.shape


# define the loss function
def SquareLoss(x, y, theta):
    """
    A square error loss function to compare the loss (cost) error
    Input: x - matrix n by m
           y - vector n by 1
```

```
            theta - vector m by 1, order matters considering the intercept
    Output: the sum of squared error
    """
    n = x.shape[0]
    fx = x @ theta  # matrix (dot) production
    loss = 1/2 * 1/n * np.sum(np.square(fx - y))  # use average with 1/n

    return(loss)


# set intial theta value
theta_initial = np.array([0, 0]).reshape(-1, 1)
# test SquareLoss function
SquareLoss(input_x, input_y, theta_initial)  # 32.072733877455676


# define the gradient descent function
def GradientDescent(x, y, theta, alpha, tolerate, maxiterate):
    i = 0  # set the iteration counting index
    tolerate_rule = 1  # set the initial tolerate rate
    n = x.shape[0]
    current_theta = theta
    cost_vector = np.empty([0, 1])

    # iterate
    while tolerate_rule >= tolerate and i <= maxiterate:
        sl = np.array(SquareLoss(x, y, current_theta)).reshape([1, 1])
        cost_vector = np.append(cost_vector, sl, axis=0)  # store cost function
        fx = x @ current_theta
        update_theta = current_theta - alpha * (1/n) * x.transpose() @ (fx - y)
        tolerate_rule = np.min(np.abs(update_theta - current_theta))
        i += 1
        current_theta = update_theta

    return(current_theta, cost_vector)


theta_initial = np.array([0, 0]).reshape(-1, 1)  # give initial value
alpha = 0.01  # learning rate
tolerate = 0.00001  # tolerate rates
maxiter1 = 1500
coefficents1, lossvalues1 = GradientDescent(input_x, input_y,
                                            theta_initial, alpha,
                                            tolerate, maxiter1)

print("The estimated coefficients are", coefficents1)
# The estimated coefficients are [[-3.63077001] [ 1.16641043]]
lossvalues1.shape
# iteration stops because function reaches to maxiter, (1501, 1)
plt.plot(lossvalues1[1:])

# we can set maxiter = 3000 to see what's going on
theta_initial = np.array([0, 0]).reshape(-1, 1)  # give initial value
alpha = 0.01  # learning rate
tolerate = 0.00001  # tolerate rates
maxiter2 = 3000
```

```python
coefficents2, lossvalues2 = GradientDescent(input_x, input_y,
                                            theta_initial, alpha,
                                            tolerate, maxiter2)

print("The estimated coefficients are", coefficents2)
# The estimated coefficients are [[-3.84072806][ 1.18750299]]


lossvalues2.shape  # (2372, 1), iteration stops because of tolerate

# plot the cost function
fig, ax = plt.subplots(figsize=(6, 5))
ax.plot(lossvalues2[1:])
ax.set(title='Plot of Loss Function', xlabel='Iteration',
       ylabel='Loss')
fig.show()


# plot the regression line
xdomain = np.linspace(5, 25)
yfit = coefficents2[0] + coefficents2[1] * xdomain

fig, ax = plt.subplots(figsize=(6, 5), sharex=True)
ax.scatter(ex1data.Population, ex1data.Profit, marker='x',
           label='Raw data')
ax.plot(xdomain.reshape(-1, 1), yfit.reshape(-1, 1), 'r',
        label='Linear regression (Gradient descent)')
ax.set(xlabel='Population of City in 10,000s',
       ylabel='Profit in $10,000s',
       title='Plot of X and Y with fitted regression')
ax.legend(loc=4)
fig.show()


# show the dynamics of loss function in gradient descent
# I will add it later
# End of code
```

Listing 2: Gradient Descent Regression

References

Ng, A. (2014). Cs229 machine learning: Lecture notes.