Gradient Descent Method

Judd Marc

March 2023

1 Introduction

Gradient descent is an iterative optimization algorithm used for minimizing the cost function in a model. The idea behind gradient descent is to update the model parameters in small steps towards the direction of steepest descent of the cost function. In other words, the algorithm aims to find the local minimum of the cost function by adjusting the model parameters in the direction of decreasing cost

Gradient Descent Method Explanation

To explain in brief about gradient descent, imagine that you are on a mountain and are blindfolded and your task is to come down from the mountain to the flat land without assistance.
The only assistance you have is a gadget which tells you the height from sea-level. What would be your approach be.
You would start to descend in some random direction and then ask the gadget what is the height now. If the gadget tells you that height and it is more than the initial height then you know you started in wrong direction.
You change the direction and repeat the process in many iterations untill you finally successfully descend down.

Well here is the analogy with machine learning terms now:
a. Size of Steps took in any direction = Learning rate
b. Gadget tells you height = Cost function
c. The direction of your steps = Gradients

Cost Function Gradients

The equation for calculating cost function and gradients are as shown below. Note: For other algorithms the cost function will be different and the gradients would have to be derived from the cost functions

Cost

$$J(\theta) = 1/2m\sum_{i=1}^{m}(h(\theta)^{(i)} - y^{(i)})^2 \tag{1}$$

Gradient

$$\frac{\partial J(\theta)}{\partial \theta_j} = 1/m\sum_{i=1}^{m}(h(\theta^{(i)} - y^{(i)}).X_j^{(i)} \tag{2}$$

Other Gradients

$$\theta_0 := \theta_0 - \alpha.(1/m.\sum_{i=1}^{m}(h(\theta^{(i)} - y^{(i)}).X_0^{(i)}) \tag{3}$$

$$\theta_1 := \theta_1 - \alpha.(1/m.\sum_{i=1}^{m}(h(\theta^{(i)} - y^{(i)}).X_1^{(i)}) \tag{4}$$

$$\theta_2 := \theta_2 - \alpha.(1/m.\sum_{i=1}^{m}(h(\theta^{(i)} - y^{(i)}).X_2^{(i)}) \tag{5}$$

$$\theta_j := \theta_j - \alpha.(1/m.\sum_{i=1}^{m}(h(\theta^{(i)} - y^{(i)}).X_0^{(i)}) \tag{6}$$

The goal of the gradient descent algorithm is to find the values of the slope m and the intercept b of the line that best fit the given data points.

The gradient descent algorithm minimizes the cost function, which is the sum of squared errors between the predicted y values and the actual y values.

The algorithm adjusts the values of m and b iterate until the cost function is minimized. The line represents the final values of m and b that give the best fit to the data points.

The purpose of plotting the line and the dotted points together is to visually assess how well the linear regression model fits the data.

If the line fits the points closely, then the model is a good fit for the data, and can be used to make predictions about new values of the independent variable.

2 Python code

```
from timeit import Timer
import numpy as np
import matplotlib.pyplot as  plt

def gradient_descent(x,y):
    m_curr=b_curr=0

    iterations = 100
    n = len(x)
    learning_rate = 0.08

    for i in range(0, iterations):
        y_predicted = m_curr * x + b_curr
        cost = (1/n) * sum([val**2 for val in (y-y_predicted)])
        md = -(2/n)*sum(x*(y-y_predicted))
        bd = -(2/n)*sum(y-y_predicted)
        m_curr = m_curr - learning_rate * md
        b_curr = b_curr - learning_rate * bd
        print ("m: {}, b: {}, cost: {} iteration: {}".format(m_curr,b_curr,cost, i))
        t = Timer("gradient_descent", "from __main__ import gradient_descent")
        print("Time taken: " +str(t.timeit(1))+ "seconds.")
    plt.plot(x, y, 'ro')
    plt.plot(x, m_curr*x+b_curr, 'b-')
    plt.grid()
    plt.show()

x = np.array([1,2,3,4,5])
y = np.array([5,7,9,11,13])
gradient_descent(x,y)
```