The Gradient Descent Algorithm

StartOnAI Machine Learning Tutorial 3

This is a tutorial which helps to break down the Gradient Descent Algorithm.

What is the Gradient Descent algorithm?

Note: This tutorial assumes that the reader understands what a cost function is, which is key to understanding how gradient descent works. In simple terms, a cost function determines how well a particular machine learning model performs on a set of data. If this concept is not clear, this link should clear up any doubts. Click Me!(Link to Cost Function Video: Not Created by Me!)

In the vast field of machine learning, there is a plethora of algorithms that all carry out different tasks by adjusting themselves to perform better as they encounter more data. An essential ingredient in the recipe for almost every one of these machine learning algorithms is the process of optimization. One optimization algorithm in particular, which can be used with any machine learning algorithm and is very easy to grasp, is known as Gradient Descent. The heart of the algorithm lies in the "gradient", which is a function that represents the slope of the tangent of the graph of the cost function. In other words, it is a derivative of the cost function which signifies its steepness. It may not completely make sense as to how this "gradient" would be useful to us in terms of optimization and minimizing the cost function, but one visualization might help with this. A great way to think about gradient descent is imagining yourself at the peak of a hill. You want to reach the bottom of the hill, but you are blindfolded and can only use your sense of touch to navigate. In order to do this, you must base your movements on the steepness of the hill until you eventually reach the bottom. This is exactly what gradient descent does, except the steepness of the hill is the gradient and the bottom of the hill is a local minimum of the cost function. Using the gradient, initial parameters can be continuously changed accordingly until a minimum is reached.

How does it physically work?

Essentially, what gradient descent is trying to do in mathematical terms is finding parameters for a function that minimizes the cost function. *[Note: Parameters are just another way of saying coefficients.]* Gradient descent is an iterative algorithm, meaning that it executes several steps in order to come to the solution. In each one of these steps, gradient descent should change

the values for the parameters so that their cost becomes lower. But how should the algorithm know where to make these steps? And how large these steps should be?

The first step of the gradient descent process is choosing some initial parameters for the function. These parameters will have a specific cost which determines how well they perform for the given data. In order to calculate these costs, the values of the parameters are plugged into a cost function. After this, the most important step of gradient descent occurs. This is when the derivative of the cost is calculated. *[If you don't know what a derivative is. Go to this link, Click me!]* Knowing the derivative or the slope of the tangent line to the graph at a specific point allows the algorithm to know how to change the parameters so that their cost is lower. However, there is another factor that determines how much these coefficients change with every iteration. The ***learning rate*** is a number that is multiplied with the derivative in order to determine how much the parameters should be altered. The size of this learning rate is something that should be taken into consideration because a learning rate that is too large might cause gradient descent to overshoot a minimum and might even cause it to diverge or fail to converge. On the other hand, a learning rate that is too small will make gradient descent a very slow process. [The learning rate does not need to be changed in between because gradient descent takes smaller steps by default] After all of these steps are repeated continuously, gradient descent will (hopefully!) result in the cost of the parameters becoming close to or at 0.

Implementing Gradient Descent for Linear Regression

We will now write code in python for gradient descent to find the solution to a linear regression problem.

```
import numpy as np
```

We start by importing the numPy library, which provides us with various tools to work with arrays.

```
def gradient_descent(x,y):
m_curr = b_curr = 0
iterations = 10000
n = len(x)
```

```
learning_rate = 0.08
```

Next, we start the gradient descent function by initializing some parameters at 0 as explained in the last section. We then determine the amount of iterations that gradient descent will go through in order to come to a solution and a learning rate which will determine how large its steps will be. The variable "n" will be used later in order to determine derivatives and costs.

```
for i in range(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))
```

This part of the code is where the meat of the algorithm occurs. The number of times that the body of the for-loop will occur is determined by our initial value for "iterations". After the predicted value is determined by the parameters, their cost is also calculated. Next, the derivative of the cost is calculated and then multiplied by the learning rate in order to determine how the parameters should be tweaked. The new values of the parameters, their cost, and the iteration number is then printed. Printing these allows us to see whether the cost is actually decreasing and we can then make adjustments accordingly. If the cost is actually increasing when we run the program, we know that the learning rate is probably too high. We can try different learning rates until we find a range that does not result in the process of gradient descent being too slow or cause it to never find the solution.

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

Remember how we imported the numPy library at the beginning of the tutorial? Well, we can now use it to create two numPy arrays. These arrays represent our two vectors, which contain our x-values and y-values.

```
gradient_descent(x,y)
```

Now, all we have to do is call our gradient descent function with two parameters being the numPy arrays we created. The algorithm will go through ten-thousand iterations, as we specified at the beginning. If the cost seems to be increasing, we can tweak the learning rate accordingly as it might be too large. If the cost is decreasing, but it doesn't reach as low as we want it to, we can increase the number of iterations.

---

5. More Resources

A Brief Explanation and Code
Understanding the Mathematics Behind Gradient Descent
Simple Understanding of the Gradient Descent Algorithm
Some Helpful Diagrams
An In-Depth Explanation

6. References

Codebasics. "Codebasics/Py." GitHub, github.com/codebasics/py/blob/master/ML/3_gradient_descent/gradient_descent.py.