chapter-2-3

January 22, 2022

Run in Google Colab

1 Introduction to Machine Learning

1.1 Cost Functions

1.1.1 Linear Cost Function

In Machine Learning a cost function or loss function is used to represent how far away a mathematical model is from the real data. One adjusts the mathematical model, usually by varying parameters within the model, so as to minimize the cost function.

Let's take for example the simple case of a linear fitting. We want to find a relationship of the form

$$y = \theta_0 + \theta_1 x \tag{1}$$

where the θs are the parameters that we want to find to give us the best fit to the data. We call this linear function $h_\theta(x)$ to emphasize the dependence on both the variable x and the two parameters θ_0 and θ_1.

We want to measure how far away the data, the $y^{(n)}$s, are from the function $h_\theta(x)$. A common way to do this is via the quadratic *cost function*

$$J(\grave{}) = \frac{1}{2N} \sum_{n=1}^{N} \left[h_\theta \left(x^{(n)} \right) - y^{(n)} \right]^2 \tag{2}$$

This is called *Ordinary Least Squares*.

In this case, the minimum is easily find analitically, differentiate (2) with respect to both θs and set the result to zero:

$$\begin{aligned} \frac{\partial J}{\partial \theta_0} &= \sum_{n=1}^{N} \left(\theta_0 + \theta_1 x^{(n)} - y^{(n)} \right) = 0 \\ \frac{\partial J}{\partial \theta_1} &= \sum_{n=1}^{N} x^{(n)} \left(\theta_0 + \theta_1 x^{(n)} - y^{(n)} \right) = 0 \end{aligned} \tag{3}$$

The solution is trivially obtained for both θs

$$\begin{aligned}\theta_0 &= \frac{(\sum y)\left(\sum x^2\right)-(\sum x)(\sum xy)}{N(\sum x^2)(\sum x)^2} \\ \theta_1 &= \frac{N(\sum xy)-(\sum y)(\sum x)}{N(\sum x^2)(\sum x)^2}\end{aligned} \tag{4}$$

1.2 Gradient Descent

We can describe the main idea behind gradient descent as climbing down a hill until a local or global cost minimum is reached. In each iteration, we take a step in the opposite direction of the gradient, where the step size is determined by the value of the **learning rate**, as well as the slope of the gradient.

The scheme works as follow: start with an initial guess for each parameter θ_k. Then move θ_k in the direction of the slope:

$$\theta_k^{new} = \theta_k^{old} + \beta \frac{\partial J}{\partial \theta_k} \tag{5}$$

Update all θ_k simultaneously and repet until convergence. Here β is a *learning factor* that governs how far you move. if β is too small it will take a long time to converge, if too large it will overshoot and might not converge at all.

The loss function J is a function of all of the data points. In the above description of gradient descent we have used all of the data points simultaneously. This is called *batch gradient* descent. But rather than use all of the data in the parameter updating we can use a technique called *stochastic gradient descent*. This is like batch gradient descent except that you only update using *one* of the data points each time. And that data point is chosen randomly.

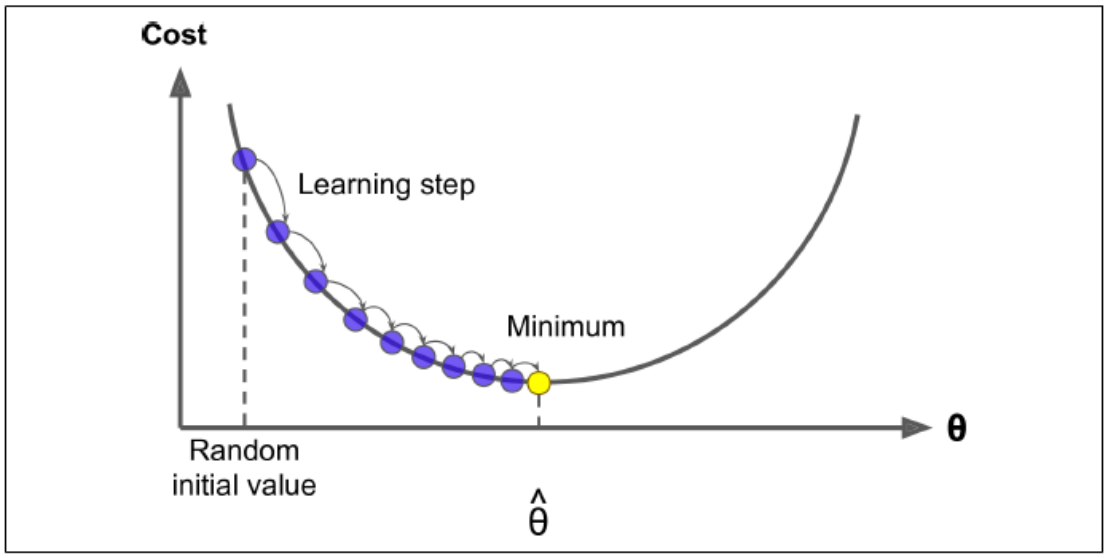
$$J(\grave{}) = \sum_{n=1}^{N} J_n(\grave{}) \tag{6}$$

Stochastic gradient descent means pick an n at random and then update according to

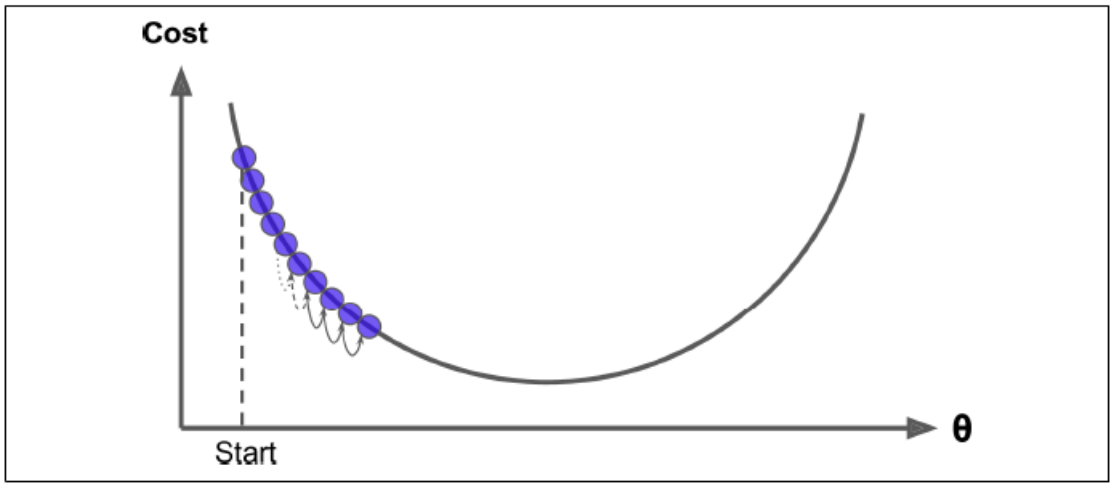
$$\theta_k^{new} = \theta_k^{old} + \beta \frac{\partial J_n}{\partial \theta_k} \tag{7}$$

Repeat, picking another data point at random, etc.

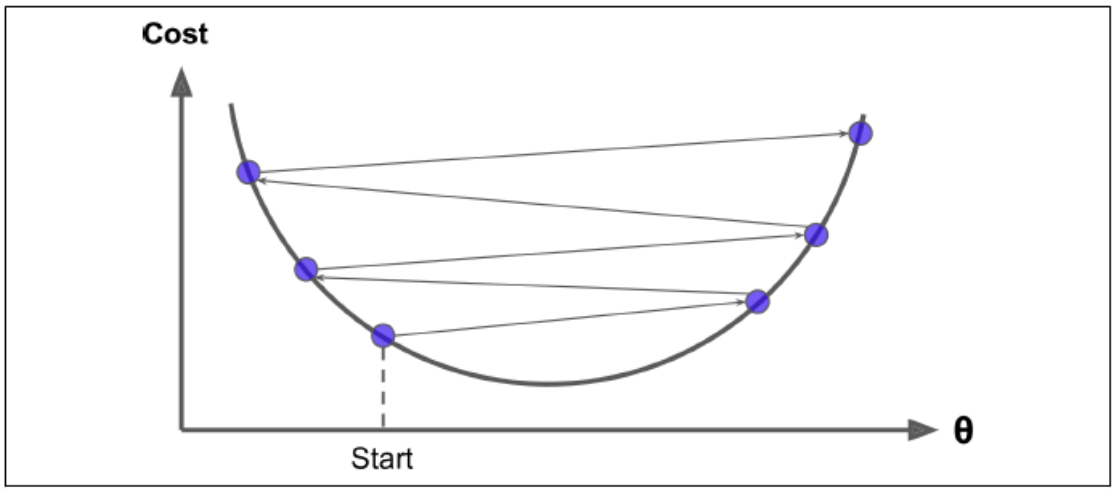
An important parameter in Gradient Descent is the size of the steps, determined by the **learning rate** hyperparameter.



If the learning rate is too small, then the algorithm will have to go through many iterations to converge, which will take a long time…



… on the other hand, if the learning rate is too high, you might jump across the valley. This might make the algorithm diverge failing to find a good solution.



Gradient descent is one of the many algorithms that benefit from feature scaling.

1.3 Stochastic Gradient Descent

In the previous section, we learned how to minimize a cost function by taking a step in the opposite direction of a cost gradient that is calculated from the whole training dataset; this is why this approach is sometimes also referred to as batch gradient descent. Now imagine that we have a very large dataset with millions of data points, which is not uncommon in many machine learning applications. Running batch gradient descent can be computationally quite costly in such scenarios, since we need to reevaluate the whole training dataset each time that we take one step toward the global minimum.

A popular alternative to the batch gradient descent algorithm is stochastic gradient descent (SGD), which is sometimes also called iterative or online gradient descent. Instead of updating the weights based on the sum of the accumulated errors over all training examples, we update the weights incrementally for each training example:

$$\eta \left(y^{(i)} - \phi \left(z^{(i)} \right) \right) \mathbf{x}^{(i)}$$

Although SGD can be considered as an approximation of gradient descent, it typically reaches convergence much faster because of the more frequent weight updates. Since each gradient is calculated based on a single training example, the error surface is noisier than in gradient descent, which can also have the advantage that SGD can escape shallow local minima more readily if we are working with nonlinear cost functions.