18 JUNE 2018 / MACHINE LEARNING

How to understand Gradient Descent, the most popular ML algorithm

by Keshav Dhandhania

Gradient Descent is one of the most popular and widely used algorithms for training machine

Machine learning models typically have parameters (weights and biases) and a cost function to evaluate how good a particular set of parameters are. Many machine learning problems reduce to finding a set of weights for the model which minimizes the cost function.

For example, if the prediction is *p*, the target is *t*, and our error metric is squared error, then the cost function $J(W) = (p - t)^2$.

Note that the predicted value *p* depends on the input *X* as well as the machine learning model and (current) values of the parameters *W*. During training, our aim is to find a set of values for *W* such that $(p - t)^2$ is small. This means our prediction *p* will be close to the target *t*.

Gradient descent is an iterative method. We start with some set of values for our model parameters (weights and biases), and improve them slowly.

To improve a given set of weights, we try to get a sense of the value of the cost function for weights similar to the current weights (by calculating the gradient). Then we move in the direction which reduces the cost function.

By repeating this step thousands of times, we'll continually minimize our cost function.

Pseudocode for Gradient Descent

Gradient descent is used to minimize a cost function *J(W)* parameterized by a model

The gradient (or derivative) tells us the incline or slope of the cost function. Hence, to minimize the cost function, we move in the direction opposite to the gradient.

1. **Initialize** the weights W randomly.
2. **Calculate the gradients** G of cost function w.r.t parameters. This is done using partial differentiation: $G = \partial J(W)/\partial W$. The value of the gradient G depends on the inputs, the current values of the model parameters, and the cost function. You might need to revisit the topic of differentiation if you are calculating the gradient by hand.
3. **Update the weights** by an amount proportional to G, i.e. $W = W - \eta G$
4. Repeat until the cost $J(w)$ stops reducing, or some other pre-defined **termination criteria** is met.

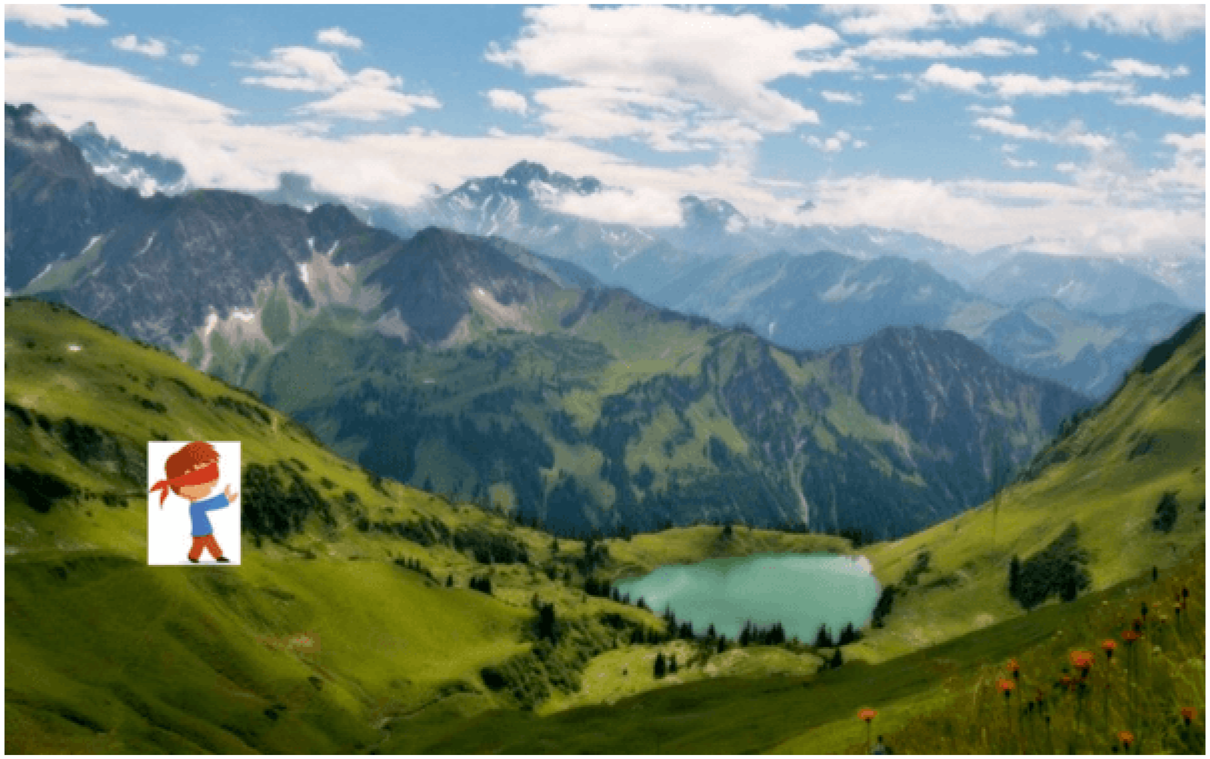
In step 3, η is the **learning rate** which determines the size of the steps we take to reach a minimum. We need to be very careful about this parameter. High values of η may overshoot the minimum, and very low values will reach the minimum very slowly.

A popular choice for the termination criteria is that the cost $J(w)$ stops reducing on a validation dataset.

Intuition for Gradient Descent

One of the simplest strategies you can use, is to feel the ground in every direction, and take a step in the direction where the ground is descending the fastest.

If you keep repeating this process, you might end up at the lake, or even better, somewhere in the huge valley.



Source: Andrej Karpathy's Stanford Course Lecture 3 analogous to trying to reach lower altitudes.

You are blind folded, since we don't have the luxury of evaluating (or 'seeing') the value of the function for every possible set of parameters.

Feeling the slope of the terrain around you is analogous to calculating the gradient, and taking a step is analogous to one iteration of update to the parameters.

By the way — as a small aside — this tutorial is part of the free Data Science Course and free Machine Learning Course on Commonlounge. The courses include many hands-on assignments and projects. If you're interested in learning Data Science / ML, definitely recommend checking it out.

Variants of Gradient Descent

There are multiple variants of gradient descent, depending on how much of the data is being used to calculate the gradient.

The main reason for these variations is computational efficiency. A dataset may have millions of data points, and calculating the gradient over the entire dataset can be computationally expensive.

- **Batch gradient descent** computes the gradient of the cost function w.r.t to parameter dataset to perform one parameter update, batch gradient descent can be very slow.

- **Stochastic gradient descent (SGD)** computes the gradient for each update using a **single training data point** *x_i* (chosen at random). The idea is that the gradient calculated this way is a stochastic approximation to the gradient calculated using the entire training data. Each update is now much faster to calculate than in batch gradient descent, and over many updates, we will head in the same general direction.
- In **mini-batch gradient descent**, we calculate the gradient for each small mini-batch of training data. That is, we first divide the training data into small batches (say *M* samples per batch). We perform one update per mini-batch. *M* is usually in the range 30–500, depending on the problem. Usually mini-batch GD is used because computing infrastructure — compilers, CPUs, GPUs — are often optimized for performing vector additions and vector multiplications.

Of these, SGD and mini-batch GD are most popular.

In a typical scenario, we do several passes over the training data before the termination criteria is met. Each pass is called an *epoch*. Also, note that since the update step is much more computationally efficient in SGD and mini-batch GD, we typically perform 100s-1000s of updates in between checks for termination criteria being met.

Choosing the learning rate

Typically, the value of the learning rate is chosen manually. We usually start with a small very slowly (increase learning rate) or is exploding / being erratic (decrease learning rate).

Although manually choosing a learning rate is still the most common practice, several methods such as Adam optimizer, AdaGrad and RMSProp have been proposed to automatically choose a suitable learning rate.

Co-authored by Keshav Dhandhania and Savan Visalpara.

Originally published as part of the free Machine Learning Course and free Data Science Course on www.commonlounge.com.

Countinue reading about

Machine Learning

How to create a "fashion police" with React Native and off-the-shelf AI

Multi-Class classification with Sci-kit learn & XGBoost: A case study using Brainwave data

An overview of Principal Component Analysis

See all 219 posts →

MICROSERVICES

These are the most effective microservice testing strategies, according to the experts

A YEAR AGO

JAVASCRIPT

JavaScript code cleanup: how you can refactor to use Classes

A YEAR AGO