**A Study on Gradient Descent**

**Introduction:**

Gradient Descent is an optimization algorithm for finding a local minimum of a differentiable function. Gradient descent is simply used in machine learning to find the values of a function's parameters (coefficients) that minimize a cost function as far as possible.

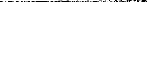
Gradient descent is by far the most popular optimization strategy used in machine learning and deep learning now. It is used when training data models, can be combined with every algorithm and is easy to understand and implement. Everyone working with machine learning should understand its concept. We'll walk through how gradient descent works, what types of it are used today, and its advantages and trade-offs.

**What is a Gradient?**

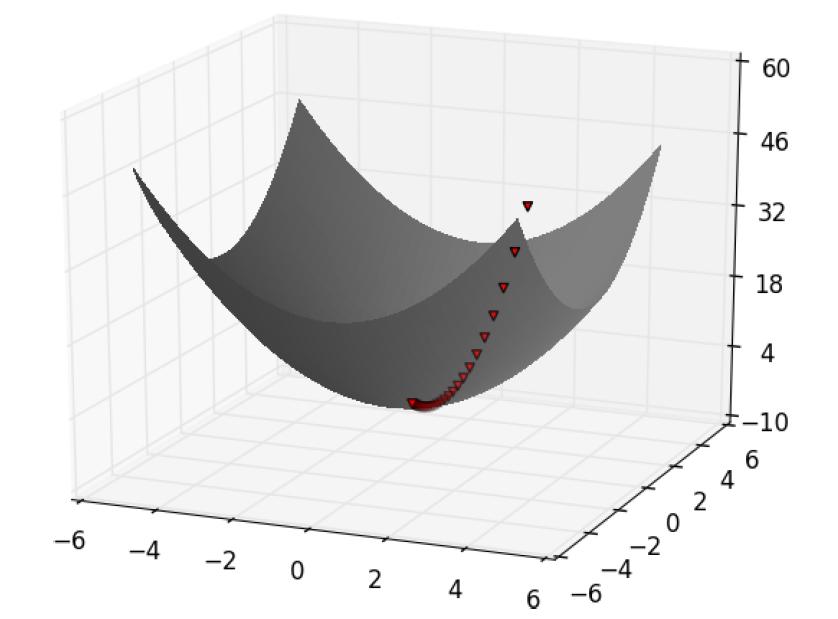
A gradient measures the change in all weights with regards to the change in error. A gradient can also be thought of as the slope of a function. The higher the gradient, the steeper the slope and the faster a model can learn. But if the slope is zero, the model stops learning. In mathematical terms, a gradient is a partial derivative with respect to its inputs.

***"A gradient measures how much the output of a function changes if you change the inputs a little bit." — Lex Fridman (MIT)***

In the case of a univariate function, it is simply the first derivative at a selected point. In the case of a multivariate function, it is a vector of derivatives in each main direction (along variable axes). Because we are interested only in a slope along one axis, and we don’t care about others these derivatives are called partial derivatives.

A gradient for an n-dimensional function f(x) at a given point p is defined as follows:

Gradient descent is a common optimization algorithm used in machine learning to estimate the model parameters. As per calculus, this is a pure partial derivative and gives the input direction in which the function most quickly increases.



Fundamentally, to maximize a function, the algorithm picks a random starting point, measures the gradient, takes a small step in the direction of the gradient, and repeats with the new starting point. Similarly, the function is minimized by taking small steps in the opposite direction. We calculate the cost function based on its initial values, and the parameter estimations are refined over various steps such that the cost function implies a minimum value ultimately.

**Function requirements:**

Gradient descent algorithm does not work for all functions. There are two specific requirements. A function must be:

* differentiable
* convex

Differentiable:

If a function is differentiable, it has a derivative for each point in its domain — not all functions meet these criteria. First, let’s see some examples of functions meeting this criterion:

Differentiable functions:

Chart, line chart

Description automatically generatedChart, line chart

Description automatically generatedChart, line chart

Description automatically generated

*f(x) = x2 f(x) = 2sin(x) f(x) = x3-2x*

*= 2x = 2cos(x) = 3x2-2*

Non-differentiable functions:Chart, line chart

Description automatically generatedChart, line chart

Description automatically generatedChart, line chart

Description automatically generated

f(x) = f(x) = f(x) =

Typical non-differentiable functions have a step, a cusp, or a discontinuity.

Convex:

For a univariate function, this means that the line segment connecting two function’s points lies on or above its curve (it does not cross it). If it does, it means that it has a local minimum which is not a global one.

Mathematically, for two points x₁, x₂ laying on the function’s curve this condition is expressed as:

***f(λx1 + (1- λ)x2) <= λf(x1) + (1- λ)f(x2)***

where λ denotes a point’s location on a section line and its value must be between 0 (left point) and 1 (right point), e.g., λ=0.5 means a location in the middle. Chart, line chart

Description automatically generated

It is also possible to use quasi-convex functions with a gradient descent algorithm. However, often they have so-called saddle points (called also minimax points) where the algorithm can get stuck. An example of a quasi-convex function is:

f(x) = x4 - 2x3 + 2

*= 4x3-6x2 = x2(4x-6)*

We see that the first derivative equals zero at x=0 and x=1.5. These places are the function’s extreme points (minimum or maximum).

Second derivative of the function:



The value of this expression is zero for x=0 and x=1. These locations are called an inflexion point — a place where the curvature changes sign — meaning it changes from convex to concave or vice-versa.

* for x<0: function is convex
* for 0<x<1: function is concave
* for x>1: function is convex again

Now we see that point x=0 has both first and second derivative equal to zero meaning this is a saddle point and point x=1.5 is a global minimum.

Let’s look at the graph of this function. As calculated before a saddle point is at x=0 and minimum at x=1.5.

Chart, line chart

Description automatically generated

*Semi-convex function with a saddle point*

For multivariate functions the most appropriate way to check whether a point is a saddle point is to calculate a Hessian matrix.

**Gradient Descent Algorithm**

Gradient Descent Algorithm iteratively calculates the next point using gradient at the current position, then scales it (by a learning rate) and subtracts the obtained value from the current position (makes a step). It subtracts the value because we want to minimise the function (to maximise it would be adding). This process can be written as:



There’s an important parameter η which scales the gradient and thus controls the step size. In machine learning, it is called **learning rate** and has a strong influence on performance.

The smaller learning rate the longer GD converges, or may reach maximum iteration before reaching the optimum point

If the learning rate is too big the algorithm may not converge to the optimal point (jump around) or even to diverge completely.

In summary, Gradient Descent method’s steps are:

1. choose a starting point (initialisation)
2. calculate gradient at this point
3. make a scaled step in the opposite direction to the gradient (objective: minimise)
4. repeat points 2 and 3 until one of the criteria is met:

* maximum number of iterations reached
* step size is smaller than the tolerance.

**Primary Types of Gradient Descent**

1. **Batch Gradient Descent (BGD)**

Batch gradient descent calculates the error for each example in the training dataset but only updates the model after all training examples have been evaluated. It is excellent for convex or relatively smooth error manifolds. It is computationally costly, but it scales well with the number of features. The following are crucial for batch gradient descent:

* It is very slow and computationally expensive.
* It is never suggested to use the vast training dataset.
* Here, convergence is slow.
* It computes the gradient using the whole training dataset.

**Stochastic Gradient Descent (SGD)**

There is a situation in batch gradient descent mentioned below:

Imagine that there is a dataset with 5 million examples. So, to take one step, the model will calculate the gradients of all 5 million examples. This type of situation happens in the batch gradient descent.

The above situation is inefficient, so stochastic gradient descent is in the picture to handle this problem. Essentially, it calculates the error and updates the model for each example in the training dataset.

Stochastic gradient descent has the following points:

* It computes gradients using a single training sample.
* It is suggested to use with large training samples.
* It is faster and less computationally expensive in comparison with Batch Gradient Descent.
* It converges much faster.
* It can shallow local minima more easily.

**Mini-batch Gradient Descent (MBGD)**

Mini-batch gradient descent splits the training dataset into small batches, and these batches are used to calculate model error and update model coefficients. It can be used for smoother curves.

The mini-batch gradient descent has the following essential points:

* It can be used when the dataset is large.
* It converges directly to minima.
* It is faster to larger datasets.
* Since in SGD, only one example is used simultaneously, vectorized implementation cannot be implemented.
* It can slow down the computations — to stop this problem, a mixture of batch gradient descent and SGD is used.

Gradient descent is a first-order iterative optimization algorithm for obtaining a local minimum of a differentiable function. It bases on a convex function and tweaks its parameters iteratively to minimize a given function to its local minimum.

In terms of types of gradient descent, batch gradient descent (BGD) is the most common form of gradient descent used in machine learning. Some day-to-day algorithms with coefficients that can be optimized using gradient descent methods are Linear Regression and Logistic Regression.

The loss function explains how great the model will perform, presenting the current set of parameters (weights and biases), and gradient descent is used to find the best set of parameters.

**Python implementation of Gradient Descent Algorithm:**

**Gradient Descent Algorithm & Back Propagation:**