(https://google.com/racialequity?authuser=0)

Reducing Loss: Gradient Descent

ıted Time: 10 minutes

The iterative approach diagram (Figure 1
(https://developers.google.com/machine-learning/crash-course/reducing-loss/an-iterative-approach?authuser=0#ml-block-diagram)
) contained a green hand-wavy box entitled "Compute parameter updates." We'll now replace that algorithmic fairy dust with something more substantial.

Suppose we had the time and the computing resources to calculate the loss for all possible values of w_1. For the kind of regression problems we've been examining, the resulting plot of loss vs. w_1 will always be convex. In other words, the plot will always be bowl-shaped, kind of like this:

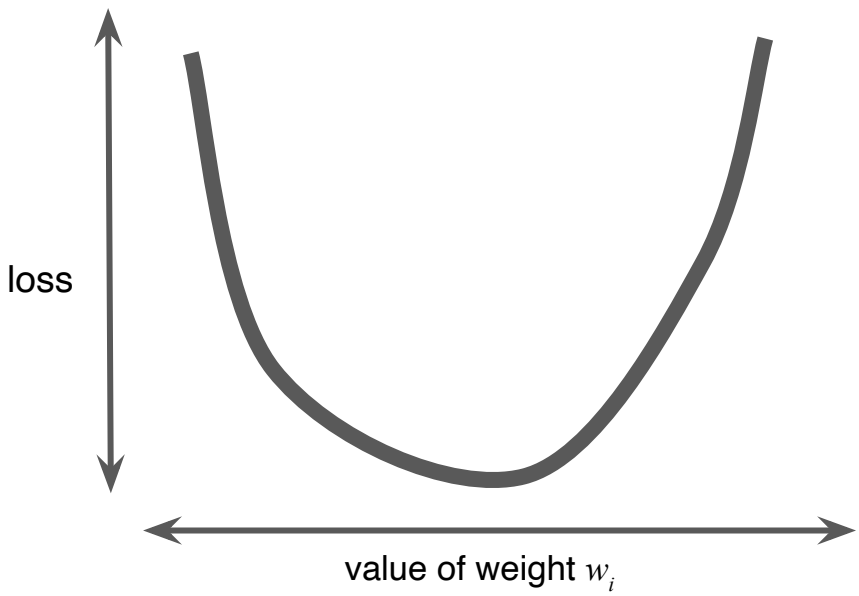


Figure 2. Regression problems yield convex loss vs. weight plots.

Convex problems have only one minimum; that is, only one place where the slope is exactly 0. That minimum is where the loss function converges.

Calculating the loss function for every conceivable value of w_1 over the entire data set would be an inefficient way of finding the convergence point. Let's examine a better mechanism—very popular in machine learning—called **gradient descent**.

The first stage in gradient descent is to pick a starting value (a starting point) for w_1. The starting point doesn't matter much; therefore, many algorithms simply set w_1 to 0 or pick a random value. The following figure shows that we've picked a starting point slightly greater than 0:

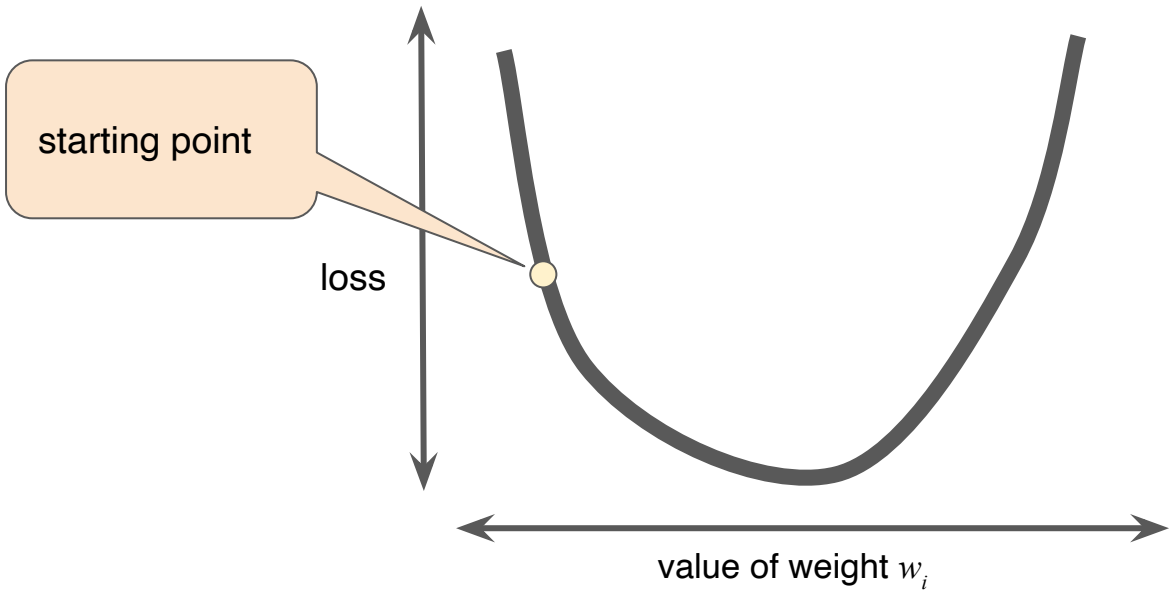


Figure 3. A starting point for gradient descent.

The gradient descent algorithm then calculates the gradient of the loss curve at the starting point. Here in Figure 3, the gradient of the loss is equal to the derivative (https://wikipedia.org/wiki/Differential_calculus#The_derivative) (slope) of the curve, and tells you which way is "warmer" or "colder." When there are multiple weights, the **gradient** is a vector of partial derivatives with respect to the weights.

+ **Click the plus icon to learn more about partial derivatives and gradients.**

The math around machine learning is fascinating and we're delighted that you clicked the link to learn more. Please note, however, that TensorFlow handles all the

gradient computations for you, so you don't actually have to understand the calculus provided here.

Partial derivatives

A **multivariable function** is a function with more than one argument, such as:

$$f(x, y) = e^{2y} \sin(x)$$

The **partial derivative f with respect to x**, denoted as follows:

$$\frac{\partial f}{\partial x}$$

is the derivative of f considered as a function of x alone. To find the following:

$$\frac{\partial f}{\partial x}$$

you must hold y constant (so f is now a function of one variable x), and take the regular derivative of f with respect to x. For example, when y is fixed at 1, the preceding function becomes:

$$f(x) = e^2 \sin(x)$$

This is just a function of one variable x, whose derivative is:

$$e^2 \cos(x)$$

In general, thinking of y as fixed, the partial derivative of f with respect to x is calculated as follows:

$$\frac{\partial f}{\partial x}(x, y) = e^{2y} \cos(x)$$

Similarly, if we hold x fixed instead, the partial derivative of f with respect to y is:

$$\frac{\partial f}{\partial y}(x, y) = 2e^{2y} \sin(x)$$

Intuitively, a partial derivative tells you how much the function changes when you perturb one variable a bit. In the preceding example:

$$\frac{\partial f}{\partial x}(0, 1) = e^2 \approx 7.4$$

So when you start at $(0, 1)$, hold y constant, and move x a little, f changes by about 7.4 times the amount that you changed x.

In machine learning, partial derivatives are mostly used in conjunction with the gradient of a function.

Gradients

The **gradient** of a function, denoted as follows, is the vector of partial derivatives with respect to all of the independent variables:

$$\nabla f$$

For instance, if:

$$f(x, y) = e^{2y} \sin(x)$$

then:

$$\nabla f(x, y) = \left(\frac{\partial f}{\partial x}(x, y), \frac{\partial f}{\partial y}(x, y) \right) = (e^{2y} \cos(x), 2e^{2y} \sin(x))$$

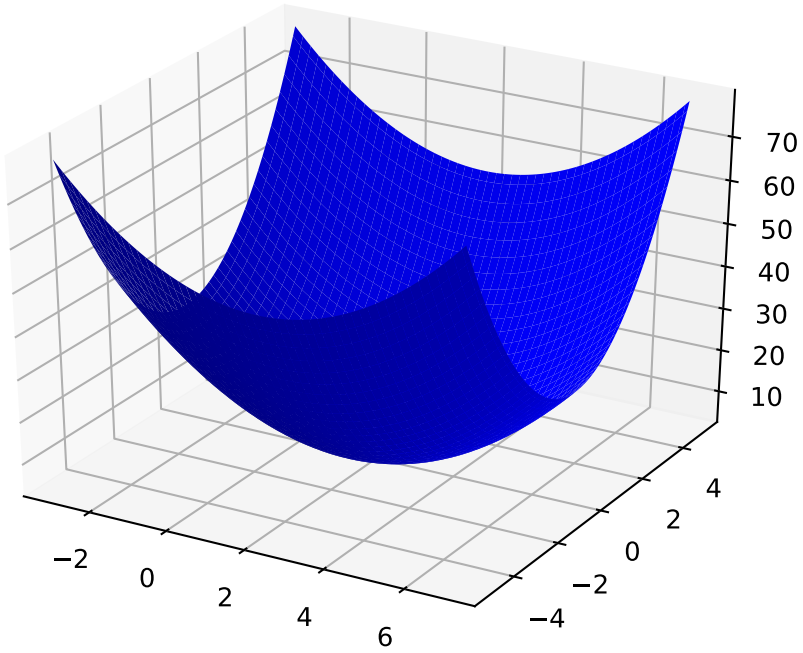
Note the following:

∇f	Points in the direction of greatest increase of the function.
$-\nabla f$	Points in the direction of greatest decrease of the function.

The number of dimensions in the vector is equal to the number of variables in the formula for f; in other words, the vector falls within the domain space of the function. For instance, the graph of the following function $f(x, y)$:

$$f(x, y) = 4 + (x - 2)^2 + 2y^2$$

when viewed in three dimensions with $z = f(x, y)$ looks like a valley with a minimum at $(2, 0, 4)$:



The gradient of $f(x, y)$ is a two-dimensional vector that tells you in which (x, y) direction to move for the maximum increase in height. Thus, the negative of the gradient moves you in the direction of maximum decrease in height. In other words, the negative of the gradient vector points into the valley.

In machine learning, gradients are used in gradient descent. We often have a loss function of many variables that we are trying to minimize, and we try to do this by following the negative of the gradient of the function.

Note that a gradient is a vector, so it has both of the following characteristics:

- a direction
- a magnitude

The gradient always points in the direction of steepest increase in the loss function. The gradient descent algorithm takes a step in the direction of the negative gradient in order to reduce loss as quickly as possible.

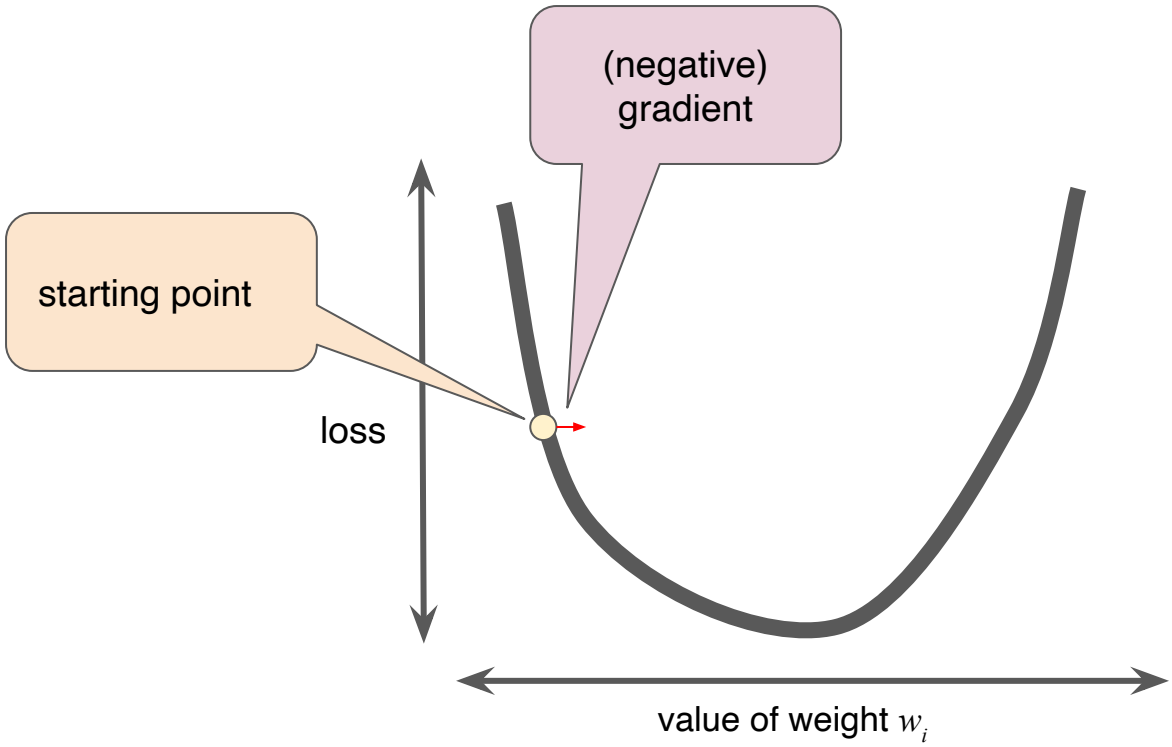


Figure 4. Gradient descent relies on negative gradients.

To determine the next point along the loss function curve, the gradient descent algorithm adds some fraction of the gradient's magnitude to the starting point as shown in the following figure:

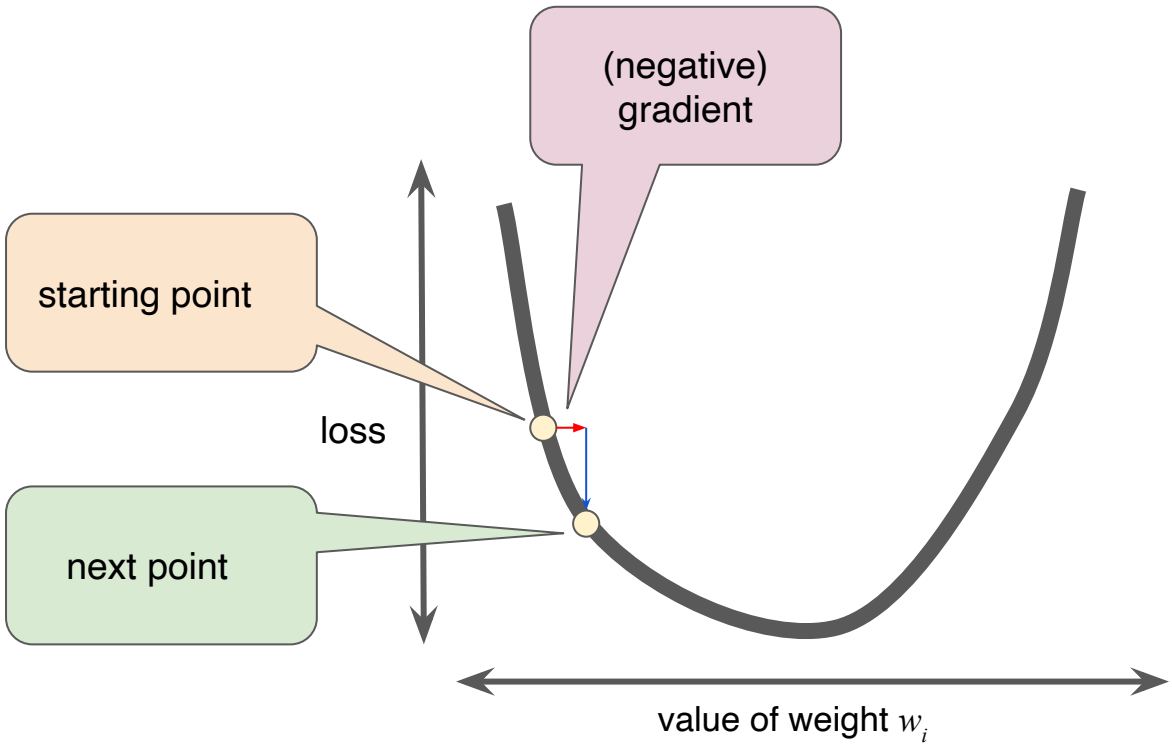


Figure 5. A gradient step moves us to the next point on the loss curve.

The gradient descent then repeats this process, edging ever closer to the minimum.

When performing gradient descent, we generalize the above process to tune all the model parameters *aneously*. For example, to find the optimal values of both w_1 and the bias b, we calculate the gradients wit :t to both w_1 and b. Next, we modify the values of w_1 and b based on their respective gradients. Then we these steps until we reach minimum loss.

erms

adient descent
://developers.google.com/machine-learning/glossary?ser=0#gradient_descent)

- step (https://developers.google.com/machine-learning/glossary?authuser=0#step)

Help Center (https://support.google.com/machinelearningeducation?authuser=0)

Previous
← An Iterative Approach (https://developers.google.com/machine-learning/crash-course/reducing-loss/an-iterative-approach?authuser=0)

Next
Learning Rate → (https://developers.google.com/machine-learning/crash-course/reducing-loss/learning-rate?authuser=0)