**Reducing loss**

 **An iterative approach**

In this method we have to start with a wild guess in the equation: and wait for the model to give loss, then we will try another guess and wait for the model to give loss and so on, until we get best possible outcome.

The "Compute Loss" part of the diagram is the loss function that the model will use. Suppose we use the squared loss function. The loss function takes in two input values:

 y’: The model's prediction for features *x*

 y: The correct label corresponding to features *x*.

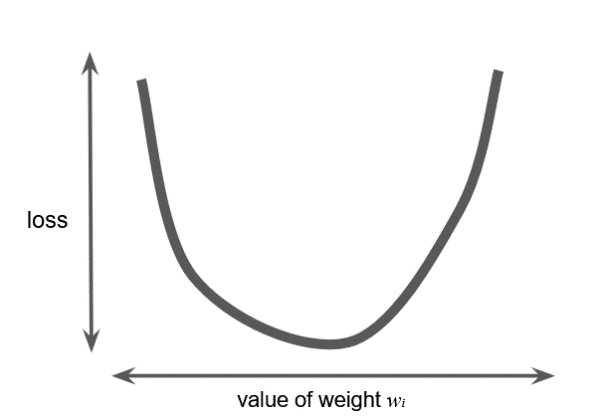
**Key point:**

A Machine Learning model is trained by starting with an initial guess for the weights and bias and iteratively adjusting those guesses until learning the weights and bias with the lowest possible loss.

 **Gradient Descent**

Suppose we had the time and the computing resources to calculate the loss for all possible values of . For the kind of regression problems we've been examining, the resulting plot of loss vs.

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.

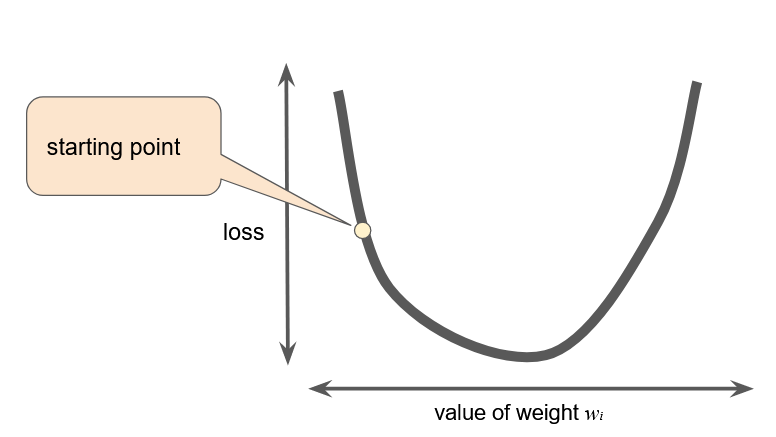
Calculating the loss function for every conceivable value of

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

. The starting point doesn't matter much; therefore, many algorithms simply set

to 0 or pick a random value. The following figure shows that we've picked a starting point slightly greater than 0:



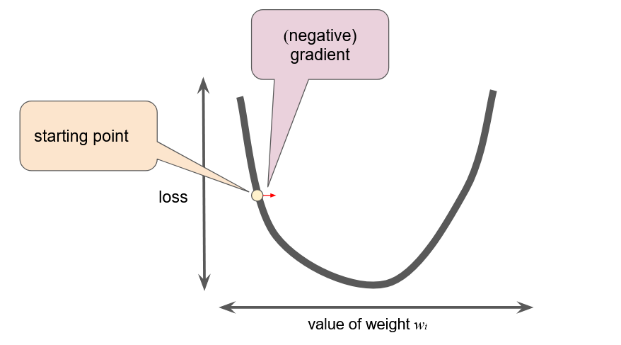
**Figure 3: A starting point for gradient descent.**

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.

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

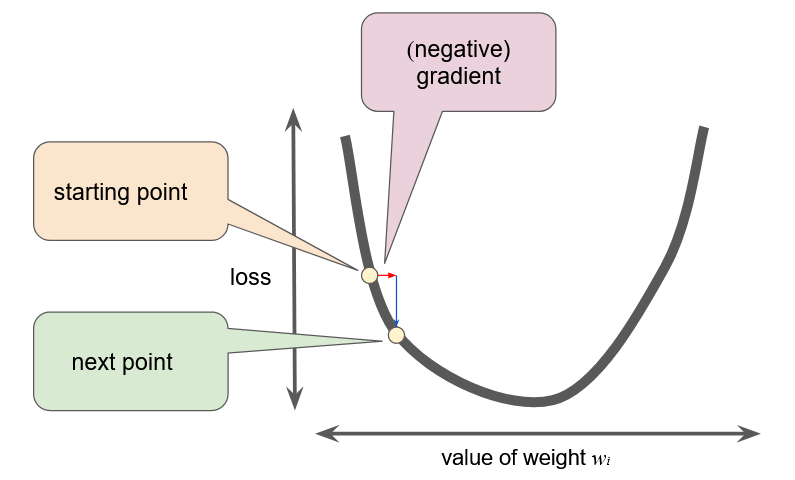
* a direction
* a magnitude

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.

**Stochastic gradient descent** (**SGD**) takes this idea to the extreme--it uses only a single example (a batch size of 1) per iteration. Given enough iterations, SGD works but is very noisy. The term "stochastic" indicates that the one example comprising each batch is chosen at random.

**Mini-batch stochastic gradient descent** (**mini-batch SGD**) is a compromise between full-batch iteration and SGD. A mini-batch is typically between 10 and 1,000 examples, chosen at random. Mini-batch SGD reduces the amount of noise in SGD but is still more efficient than full-batch.