

# What is gradient descent?

Gradient descent is an optimization algorithm which is commonly-used to train [machine learning](#) models and [neural networks](#). Training data helps these models learn over time, and the cost function within gradient descent specifically acts as a barometer, gauging its accuracy with each iteration of parameter updates. Until the function is close to or equal to zero, the model will continue to adjust its parameters to yield the smallest possible error.

Once machine learning models are optimized for accuracy, they can be powerful tools for artificial intelligence (AI) and computer science applications.

## How does gradient descent work?

Before we dive into gradient descent, it may help to review some concepts from linear regression. You may recall the following formula for the slope of a line, which is  $y = mx + b$ , where  $m$  represents the slope and  $b$  is the intercept on the y-axis.

You may also recall plotting a scatterplot in statistics and finding the line of best fit, which required calculating the error between the actual output and the predicted output ( $\hat{y}$ ) using the mean squared error formula. The gradient descent algorithm behaves similarly, but it is based on a convex function.

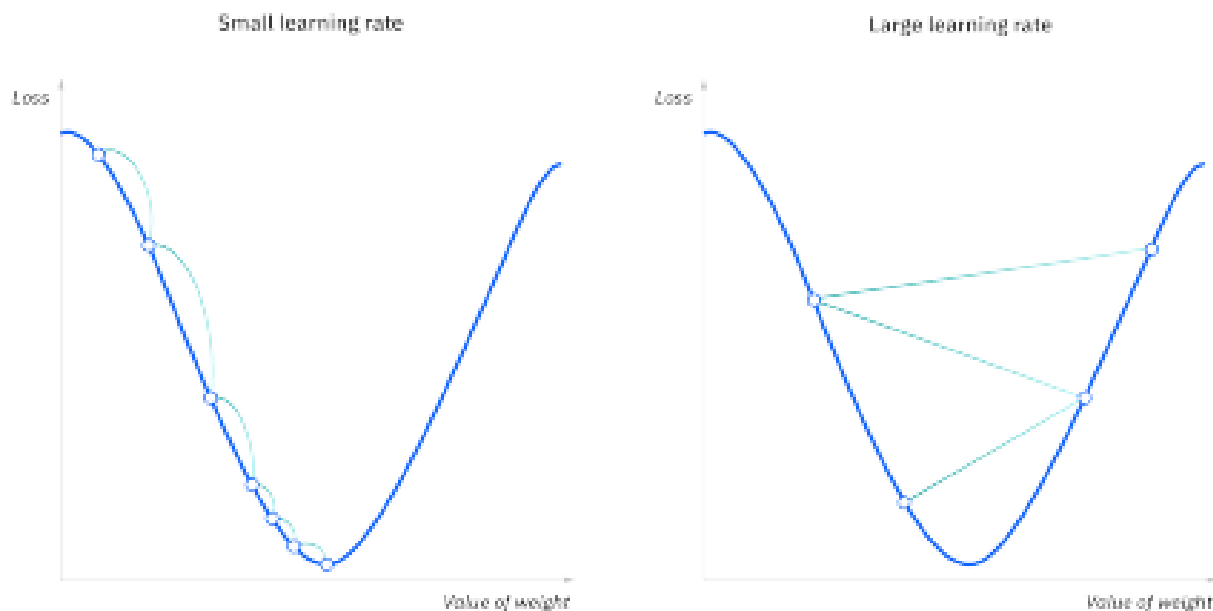
The starting point is just an arbitrary point for us to evaluate the performance. From that starting point, we will find the derivative (or slope), and from there, we can use a tangent line to observe the steepness of the slope. The slope will inform the updates to the parameters—i.e. the weights and bias. The slope at the starting point will be steeper, but as new parameters are generated, the steepness should gradually reduce until it reaches the lowest point on the curve, known as the point of convergence.

Similar to finding the line of best fit in linear regression, the goal of gradient descent is to minimize the cost function, or the error between predicted and actual  $y$ . In order to do this, it requires two data points—a direction and a learning rate. These factors determine the partial derivative calculations of future

iterations, allowing it to gradually arrive at the local or global minimum (i.e. point of convergence).

**Learning rate** (also referred to as step size or the alpha) is the size of the steps that are taken to reach the minimum. This is typically a small value, and it is evaluated and updated based on the behavior of the cost function. High learning rates result in larger steps but risks overshooting the minimum. Conversely, a low learning rate has small step sizes. While it has the advantage of more precision, the number of iterations compromises overall efficiency as this takes more time and computations to reach the minimum.

**The cost (or loss) function** measures the difference, or error, between actual  $y$  and predicted  $y$  at its current position. This improves the machine learning model's efficacy by providing feedback to the model so that it can adjust the parameters to minimize the error and find the local or global minimum. It continuously iterates, moving along the direction of steepest descent (or the negative gradient) until the cost function is close to or at zero. At this point, the model will stop learning. Additionally, while the terms, cost function and loss function, are considered synonymous, there is a slight difference between them. It's worth noting that a loss function refers to the error of one training example, while a cost function calculates the average error across an entire training set.



## Types of gradient descent

There are three types of gradient descent learning algorithms: batch gradient descent, stochastic gradient descent and mini-batch gradient descent.

## Batch gradient descent

Batch gradient descent sums the error for each point in a training set, updating the model only after all training examples have been evaluated. This process is referred to as a training epoch.

While this batching provides computation efficiency, it can still have a long processing time for large training datasets as it still needs to store all of the data into memory. Batch gradient descent also usually produces a stable error gradient and convergence, but sometimes that convergence point isn't the most ideal, finding the local minimum versus the global one.

## Stochastic gradient descent

Stochastic gradient descent (SGD) runs a training epoch for each example within the dataset and it updates each training example's parameters one at a time. Since you only need to hold one training example, they are easier to store in memory. While these frequent updates can offer more detail and speed, it can result in losses in computational efficiency when compared to batch gradient descent. Its frequent updates can result in noisy gradients, but this can also be helpful in escaping the local minimum and finding the global one.

## Mini-batch gradient descent

Mini-batch gradient descent combines concepts from both batch gradient descent and stochastic gradient descent. It splits the training dataset into small batch sizes and performs updates on each of those batches. This approach strikes a balance between the computational efficiency of batch gradient descent and the speed of stochastic gradient descent.

# Challenges with gradient descent

While gradient descent is the most common approach for optimization problems, it does come with its own set of challenges. Some of them include:

## Local minima and saddle points

For convex problems, gradient descent can find the global minimum with ease, but as nonconvex problems emerge, gradient descent can struggle to find the global minimum, where the model achieves the best results.

Recall that when the slope of the cost function is at or close to zero, the model stops learning. A few scenarios beyond the global minimum can also yield this slope, which are local minima and saddle points. Local minima mimic the shape of a global minimum, where the slope of the cost function increases on either side of the current point. However, with saddle points, the negative gradient only exists on one side of the point, reaching a local maximum on one side and a local minimum on the other. Its name inspired by that of a horse's saddle.

Noisy gradients can help the gradient escape local minimums and saddle points.

## Vanishing and Exploding Gradients

In deeper neural networks, particular [recurrent neural networks](#), we can also encounter two other problems when the model is trained with gradient descent and backpropagation.

**Vanishing gradients:** This occurs when the gradient is too small. As we move backwards during backpropagation, the gradient continues to become smaller, causing the earlier layers in the network to learn more slowly than later layers. When this happens, the weight parameters update until they become insignificant—i.e. 0—resulting in an algorithm that is no longer learning.

**Exploding gradients:** This happens when the gradient is too large, creating an unstable model. In this case, the model weights will grow too large, and they will eventually be represented as NaN. One solution to this issue is to leverage a dimensionality

reduction technique, which can help to minimize complexity within the model.