DL Optimization

How would you define optimization?

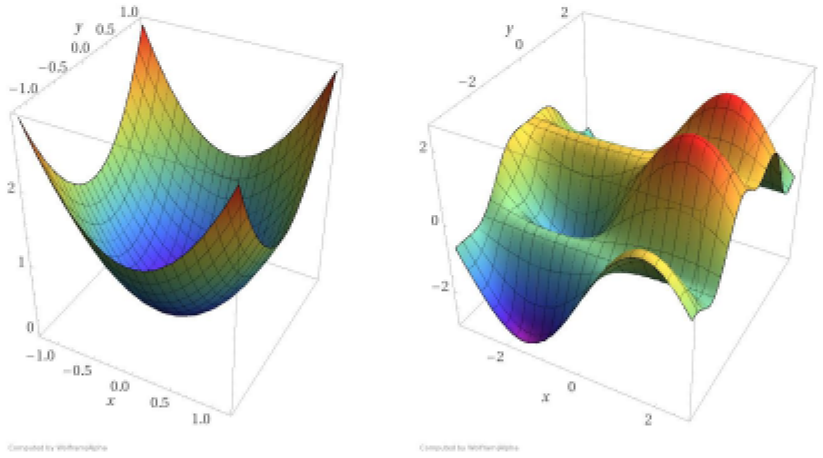
Iterative training of the model leads to a maximum and minimum function evaluation, which is known as optimization. Improved results are one of the most significant phenomena in Machine Learning.

What purpose do we serve by optimising our machine learning models?

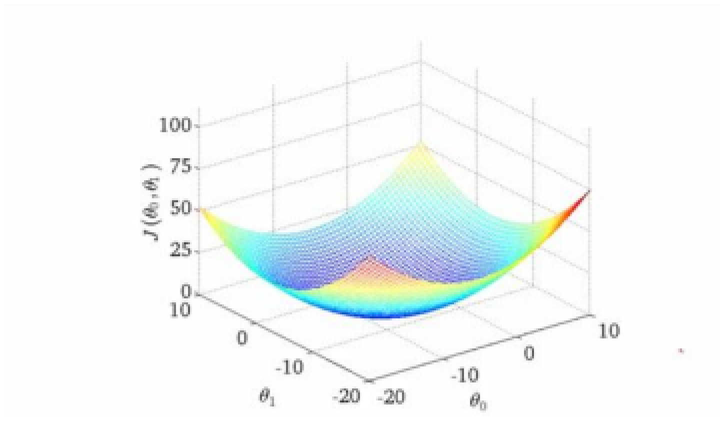
Every iteration is compared to the previous one, and the hyperparameters are changed step by step until the desired results are achieved. We develop a model that is more accurate while also having a lower error rate. We can improve a model in a variety of ways.

Loss Surfaces

- For the most part, machine learning entails attempting to locate the most cost-effective locations on a loss surface.
- The loss surface represents the loss function in relation to a hypothesis parameter.
- When we train a model, what we're really doing is trying to reduce the size of a loss function. The value of this loss function tells us how far our network's performance on a particular dataset is from perfect.



Assume our network has only two parameters for the purpose of simplicity. If we're lucky, the loss function's curve will resemble this:



Our objective is to determine the weight at which the least amount of weight is lost. The loss function's minima will be found at this point.

Gradient Descent

During the process of developing a machine learning model, an optimization approach called gradient descent is employed. As a result, it uses a convex function and iteratively lowers the parameters until the function is reduced to a local minimum.

So, what exactly is the gradient descendant system?

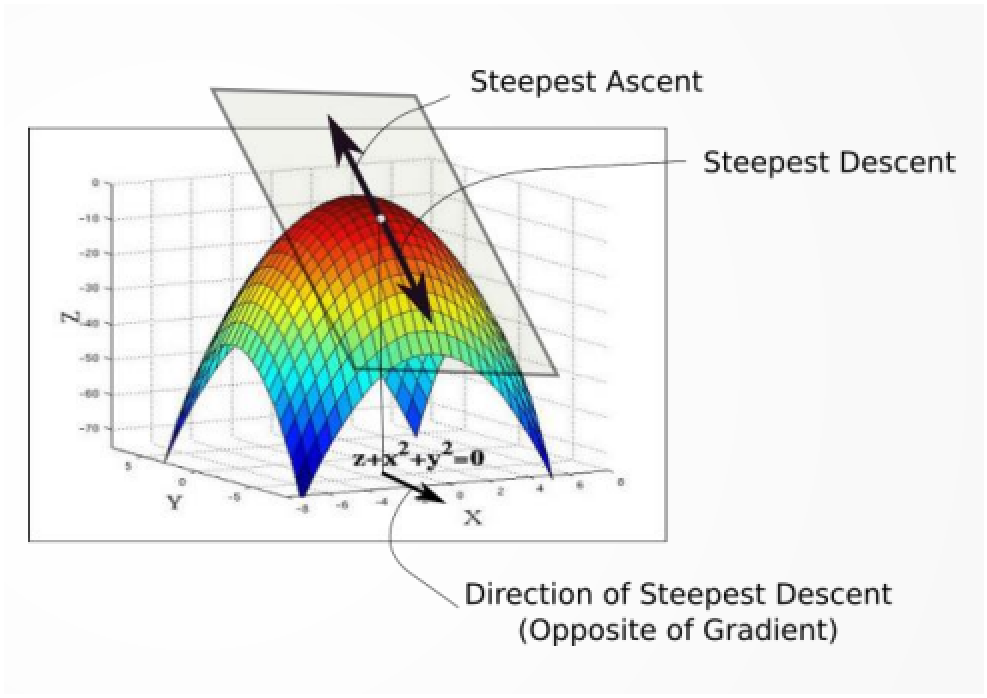
In order to determine a local minimum of any differentiable function, Gradient Descent is used as an optimization procedure. Progressive descent is used to identify the parameters (coefficients) of a function whose values minimise a cost function.

Gradient descent employs mathematics to iteratively alter the parameter values until the provided cost function is minimised. This is where you define the initial parameter values. Gradients are necessary to fully grasp this notion.

To begin, what exactly is a Gradient?

In order to calculate a gradient, all weights are compared to their error. Gradients can also be viewed as the slope of a function, which is another way of expressing them. More gradient means steeper slope and more rapid learning for a model. A slope of zero means that the model has reached its learning limit. As a result of the gradient, the function's output is a partial derivative of the inputs.

A gradient is a derivative in machine learning of a function with several input variables. The gradient, mathematically known as the slope of a function, measures the change in all weights in relation to the error change.



Gradient Descent in Action

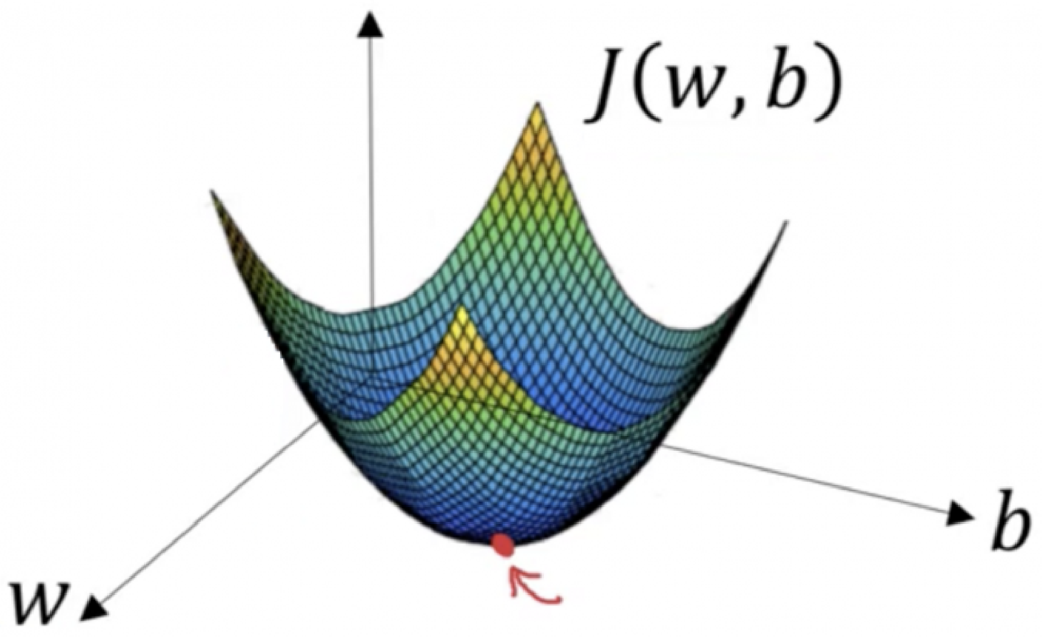
Gradient descent can be compared to going down a valley rather than ascending up a hill. We can use this as an analogy because we are using a minimal-function minimization approach.

Gradient descent can be described mathematically as follows: b indicates our climber's next position, whereas a represents his current position. Gradient descent's minimization phase is denoted by the minus sign. The gradient term (f(a)) merely indicates the direction of the steepest drop, whereas the gamma in the centre represents a waiting factor.

$$b = a - \gamma \nabla f(a)$$

As a result, this algorithm essentially informs us where to go next based on the sharpest descent. To make the point crystal clear, let's take a look at yet another illustration.

Let's say you have a machine learning problem and you want to train your algorithm with gradient descent to minimise your cost-function J(w, b) and reach its local minimum by adjusting its parameters (w and b). These parameters (w and b) are shown on horizontal axes whereas the cost function J(w, b) is shown vertically. Gradient descent can be thought of as a convex function because of its slope.



We wish to determine w and b such that the cost function has a minimum value (marked with the red arrow). We initialise w and b with some random numbers to get the search started. This is where gradient descent begins (around the top of our figure), and it proceeds in the steepest downward direction (i.e. from the top to the bottom of our illustration) in order to arrive at the smallest possible cost function.

Challenges with Gradient Descent

- Local Minima
- Global Minima
- Saddle Point
- Plateau

Types of Gradient Descent

Gradient descent can be divided into three categories based on how much data they use:

Batch gradient descent

Known as batch gradient descent, this technique calculates the error for each example in the training dataset, but the model is not changed until all of the instances have been reviewed. An epoch is a term used to describe this period of time in the training process.

As a computationally efficient algorithm, batch gradient descent also yields steady errors and convergence. The stable error gradient has certain drawbacks, such as causing the model to reach a state of convergence that isn't its best. To make this work, the algorithm necessitates the availability of the whole training dataset in memory.

Stochastic gradient descent

Stochastic gradient descent (SGD) on the other hand adjusts the parameters for each training example in the dataset one at a time. SGD may be faster than batch gradient descent depending on the problem. One benefit is that the regular updates give us a good idea of how quickly things are improving.

The frequent updates, on the other hand, are more computationally expensive. Because of this, the error rate may hop around instead of steadily decreasing, resulting in noisy gradients.

Mini-batch gradient descent

This approach, which combines the principles of SGD and batch gradient descent, is the go-to one. It does nothing more complicated than dividing the training dataset into smaller groups and running an update on each one. Stochastic gradient descent is more resilient, but batch gradient descent is more efficient, thus this is a good compromise.

In machine learning, mini-batch sizes typically range from 50 to 256, but there is no hard and fast rule because it varies depending on the application. Gradient descent is the most frequent sort of gradient learning algorithm when training a neural network.

More challenges

- Choosing a proper learning rate can be difficult.
- Additionally, the same learning rate applies to all parameter updates. If our data is sparse and our features have very different frequencies, we might not want to update all of them to the same extent, but perform a larger update for rarely occurring features.
- Another key challenge of minimizing highly non-convex error functions common for neural networks is avoiding getting trapped in their numerous suboptimal local minima.
- Dauphin et al. [3] argue that the difficulty arises in fact not from local minima but from saddle points, i.e. points where one dimension slopes up and another slopes down. These saddle points are usually surrounded by a plateau of the same error, which makes it notoriously hard for SGD to escape, as the gradient is close to zero in all dimensions.

Momentum

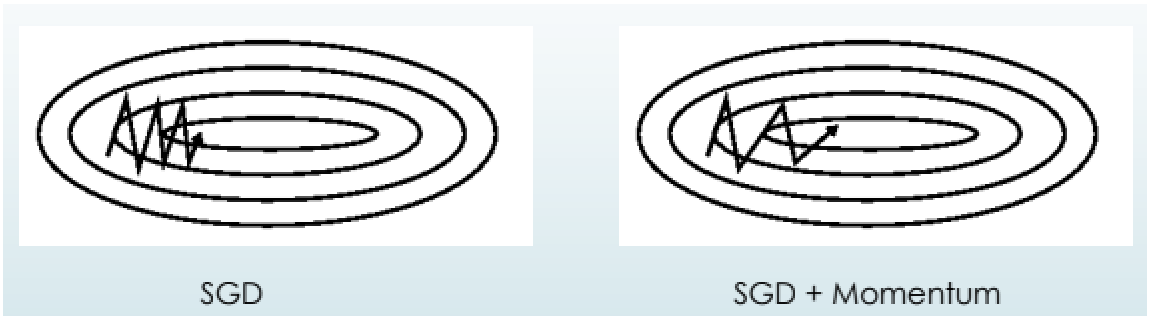
- Commonly referred as as SGD Momentum
- Here's a well-known example of how momentum works: Imagining a graded drop is like watching someone stroll down a mountain. When descending, he chooses the most difficult road; his progress is slow but sure. Momentum is like a big ball of steel rolling down a steep hill at a constant speed for a long time. As a result of the additional inertia, we are both smoothed out and accelerated, barreling through narrow troughs, minor humps, and local minima.

$$V_t = \beta V_{t-1} + \alpha \nabla_w L(W, X, y)$$
$$W = W - V_t$$

Problem with Momentum

- The danger is that we'll get too much momentum and blow past the zero-point

- When Yuri Nesterov discovered this issue in 1983, he wrote a paper that provided a solution.


SGD
SGD + Momentum