**Optimization**

Explore the different variations of the gradient descent algorithm.

**We'll cover the following**

* [Gradient descent](https://www.educative.io/module/page/qjv3oKCzn0m9nxLwv/10370001/5200716586549248/6384046380154880#Gradient-descent)
* [Batch Gradient Descent](https://www.educative.io/module/page/qjv3oKCzn0m9nxLwv/10370001/5200716586549248/6384046380154880#Batch-Gradient-Descent)
* [Stochastic Gradient Descent](https://www.educative.io/module/page/qjv3oKCzn0m9nxLwv/10370001/5200716586549248/6384046380154880#Stochastic-Gradient-Descent)
* [Mini-batch Stochastic Gradient Descent](https://www.educative.io/module/page/qjv3oKCzn0m9nxLwv/10370001/5200716586549248/6384046380154880#Mini-batch-Stochastic-Gradient-Descent)

Optimization is the selection of the best element (with regard to some criterion) from a set of available alternatives.

In the simplest case, an optimization problem consists of maximizing or minimizing a real function by systematically choosing input values from within an allowed set and computing the value of the function.

In the case of machine learning, **optimization refers to minimizing the loss function by systematically updating the network weights**. Mathematically, this is expressed as:

�′=��������(�)*w*′=*argminw*​*C*(*w*)

given a loss function �*C* and weights �*w*.

Intuitively, it can be thought of as descending a high-dimensional landscape. If we could project it in 2D plot, the height of the landscape would be the value of the loss function, and the horizontal axis would be the values of our weights w. Ultimately, the goal is to reach the bottom of the landscape by iteratively exploring the space around us.

**Gradient descent**

Gradient descent is based on the basic idea of following the local slope of our landscape. We essentially introduce physics and the law of gravity in the mix. Calculus provides us with an elegant way to calculate the slope of the landscape, which is the derivative of the function at this point (also known as the gradient) with respect to the weights.

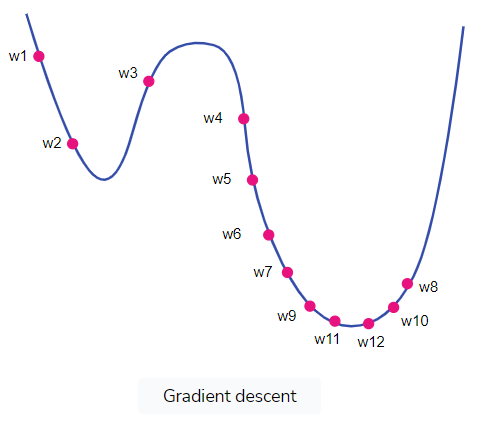
�=�−�⋅∇��(�)*w*=*w*−*λ*⋅∇*w*​*C*(*w*)

Learning rate (�*λ*) is a constant value that determines the step size at each iteration, while moving toward a minimum of a loss function.

Algorithmically, this can be expressed in Python as below:

for t in range(steps):  
  dw = gradient(loss, data, w)  
  w = w - learning\_rate \*dw

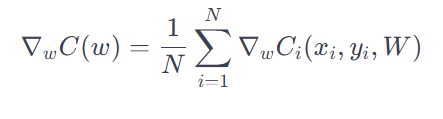
Visually, we can imagine the following diagram which corresponds to a 2d space:



In practice, there are 3 main variants of gradient descent when it comes to deep learning.

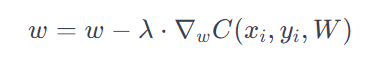
## Batch Gradient Descent

The equation and code presented above actually referred to batch gradient descent. In this variant, **we calculate the gradient for the entire dataset on each training step before updating the weights**.



Since we take the sum of the loss of all individual training examples, our computation quickly becomes very expensive. Therefore, it is impractical for large datasets.

**Stochastic Gradient Descent**

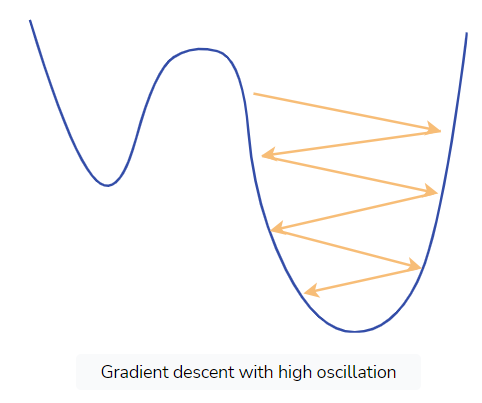
Stochastic Gradient Descent (SGD) was introduced to address this exact issue. **Instead of calculating the gradient over all training examples and updating the weights, SGD updates the weights for each training example ��,��*xi*​,*y****i*​  


In Python, this can be represented as follows:

for t in range(steps):  
  for example in data:  
    dw = gradient(loss, example, w)  
    w = w - learning\_rate \*dw

As a result, SGD is much faster and more computationally efficient, but it has noise in the estimation of the gradient. Since it updates the weight frequently, it can lead to big oscillations and that makes the training process highly unstable.

We continuously walk in a zig-zag fashion down the landscape, which keeps overshooting and missing our minimum. However, we can easily get away from local minimums for the same reason, and keep searching for a better one.



## Mini-batch Stochastic Gradient Descent

Mini-batch SGD sits right in the middle of the two previous ideas and combines the best of both worlds. **It randomly selects** �*n* **training examples — the so-called mini-batch — from the whole dataset and computes the gradients only from them**. It essentially tries to approximate Batch Gradient Descent by sampling only a subset of the data. Mathematically:



In practice, mini-batch SGD is the most frequently used variation because it is both computationally cheap and results in more robust convergence.

for t in range(steps):  
  for mini\_batch in get\_batches(data, batch\_size):  
    dw = gradient(loss, mini\_batch, w)  
    w = w - learning\_rate \*dw

Note that in the bibliography and frameworks, the term SGD often refers to mini-batch SGD. From now on, we will use the same terminology.