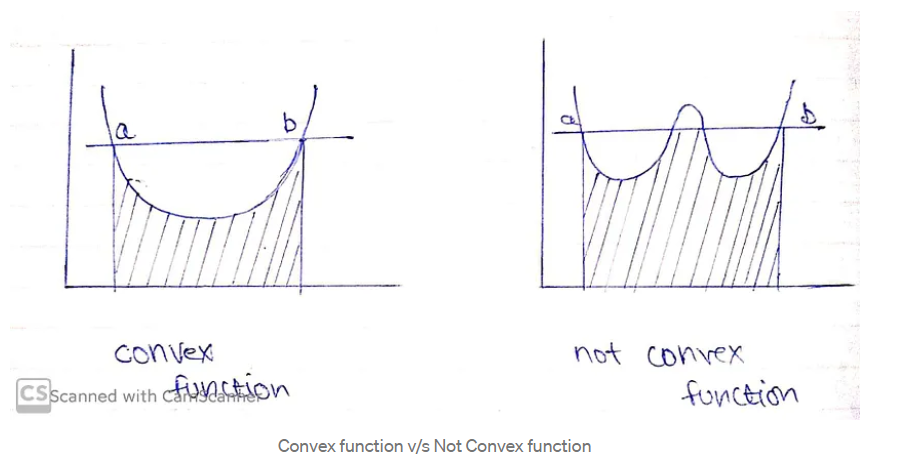
1. Gradient Descent:

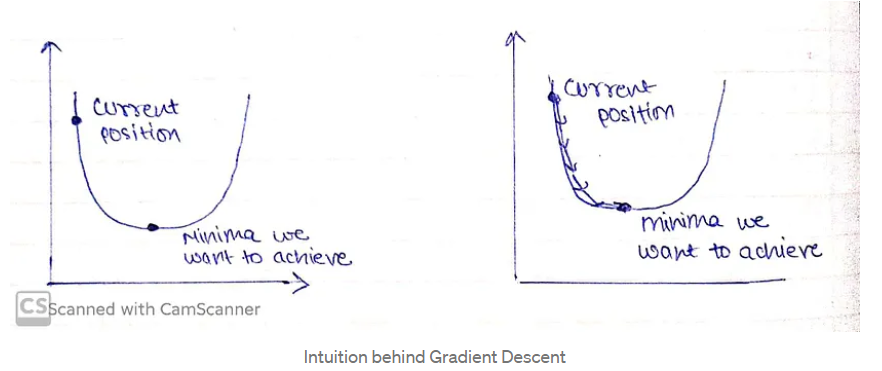
*Gradient Descent* is an **optimization algorithm** which is commonly-used to train [**machine learning**](https://www.ibm.com/topics/machine-learning)**models and**[**neural networks**](https://www.ibm.com/topics/neural-networks).  Training data helps these models learn over time, and the **cost function** 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.

In machine/deep learning terminology, it’s the task of minimizing the cost/loss function J(w) parameterized by the model’s parameters w ∈ R^d.  
Optimization algorithms (in the case of minimization) have one of the following goals:

1. Find the global minimum of the objective function. This is feasible if the objective function is convex, i.e. any local minimum is a global minimum.
2. Find the lowest possible value of the objective function within its neighborhood. That’s usually the case if the objective function is not convex as the case in most deep learning problems.



Let’s take a look at what we want to achieve.

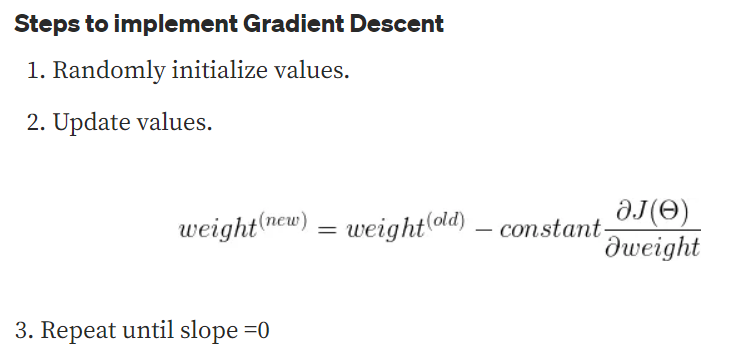


For ease, let’s take a simple linear model.

Error = Y(Predicted)-Y(Actual)

A machine learning model always wants low error with maximum accuracy, in order to decrease error we will intuit our algorithm that you’re doing something wrong that is needed to be rectified, that would be done through Gradient Descent.

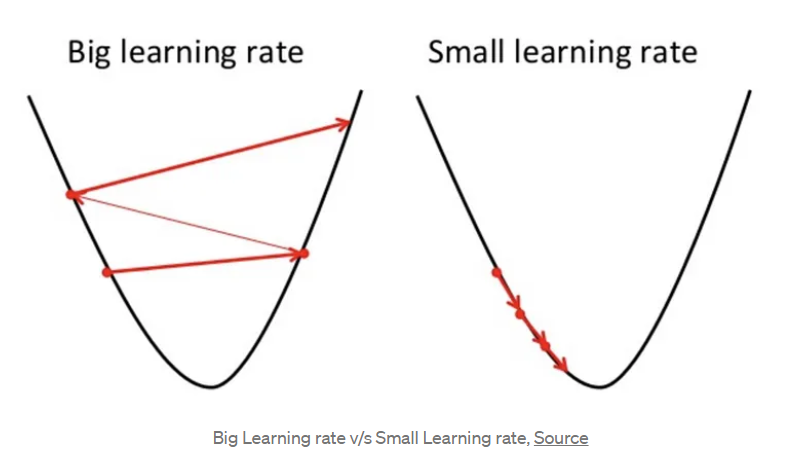
We need to minimize our error, in order to get pointer to minima we need to walk some steps that are known as alpha(learning rate).



Learning rate (λ) is one such **hyper-parameter** that defines the **adjustment in the weights of our network with respect to the loss gradient descent**. It determines how fast or slow we will move towards the optimal weights

The Gradient Descent Algorithm estimates the weights of the model in many iterations by minimizing a cost function at every step.

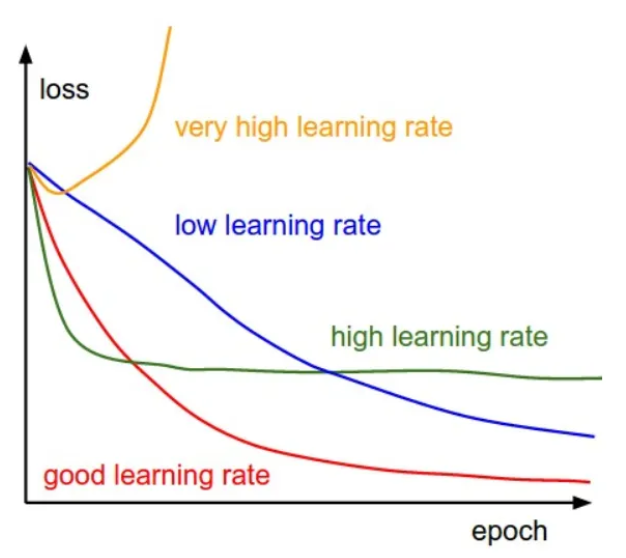
Learning rate must be chosen wisely as:  
1. if it is too small, then the model will take some time to learn.  
2. if it is too large, model will converge as our pointer will shoot and we’ll not be able to get to minima.



A good way to make sure the gradient descent algorithm runs properly is by plotting the cost function as the optimization runs. Put the number of iterations on the x-axis and the value of the cost function on the y-axis. This helps you see the value of your cost function after each iteration of gradient descent, and provides a way to easily spot how appropriate your learning rate is.

If the gradient descent algorithm is working properly, the cost function should decrease after every iteration.

When gradient descent can’t decrease the cost-function anymore and remains more or less on the same level, it has converged. The number of iterations gradient descent needs to converge can sometimes vary a lot. It can take 50 iterations, 60,000 or maybe even 3 million.



Gradient Descent with different learning rates

There are three main types of Gradient Descent:

1. Batch Gradient Descent

2. Stochastic Gradient Descent

3. Mini-batch Gradient Descent

**2. Stochastic Gradient Descent**

There are a few downsides of the gradient descent algorithm. We need to take a closer look at the amount of computation we make for each iteration of the algorithm.

Assuming we have 10,000 data points and 10 features. The sum of squared residuals consists of as many terms as there are data points, so 10000 terms in our case. We need to compute the derivative of this function with respect to each of the features, so in effect we will be doing 10000 \* 10 = 100,000 computations per iteration. It is common to take 1000 iterations, in effect we have 100,000 \* 1000 = 100000000 computations to complete the algorithm. That is pretty much an overhead and hence gradient descent is slow on huge data.

This is where we consider using Stochastic Gradient Descent. The word ‘*stochastic*’ means a system or process linked with a random probability. Hence, in Stochastic Gradient Descent, a few samples are selected randomly instead of the whole data set for each iteration.

Stochastic Gradient Descent (SGD) is a variant of the Gradient Descent algorithm used for optimizing machine learning models. In this variant, only one random training example is used to calculate the gradient and update the parameters at each iteration. Here are some of the advantages and disadvantages of using SGD:

***Advantages****:*

**+ Speed:**SGD is faster than other variants of Gradient Descent such as Batch Gradient Descent and Mini-Batch Gradient Descent since it uses only one example to update the parameters.

**+ Memory Efficiency:** Since SGD updates the parameters for each training example one at a time, it is memory-efficient and can handle large datasets that cannot fit into memory.

**+ Avoidance of Local Minima:** Due to the noisy updates in SGD, it has the ability to escape from local minima and converge to a global minimum.

***Disadvantages:***

**+ Noisy updates:** The updates in SGD are noisy and have a high variance, which can make the optimization process less stable and lead to oscillations around the minimum.

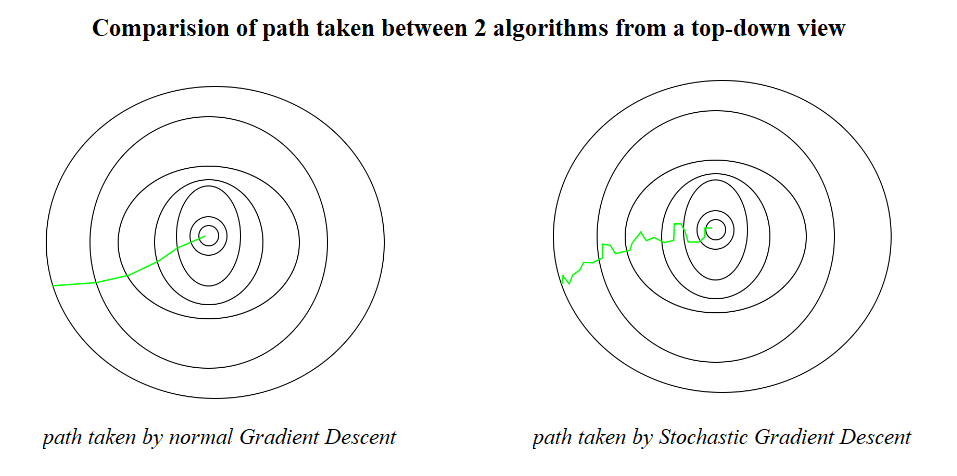
**+ Slow Convergence:** SGD may require more iterations to converge to the minimum since it updates the parameters for each training example one at a time.

**+ Sensitivity to Learning Rate:**The choice of learning rate can be critical in SGD since using a high learning rate can cause the algorithm to overshoot the minimum, while a low learning rate can make the algorithm converge slowly.

**+ Less Accurate:** Due to the noisy updates, SGD may not converge to the exact global minimum and can result in a suboptimal solution. This can be mitigated by using techniques such as learning rate scheduling and momentum-based updates

In SGD, we find out the gradient of the cost function of a single example at each iteration instead of the sum of the gradient of the cost function of all the examples.

Since only one sample from the dataset is chosen at random for each iteration, the *path taken by the algorithm to reach the minima is usually noisier* than your typical Gradient Descent algorithm. But that doesn’t matter all that much because the path taken by the algorithm does not matter, as long as we reach the minima and with a significantly shorter training time.



As you can see, both algorithms reached the minima but with different shapes of path. For all we care is the duration for it to reach the destination, not the path it took.

One thing to be noted is that, as SGD is generally noisier than typical Gradient Descent, it usually took a higher number of iterations to reach the minima, because of its randomness in its descent. Even though it requires a higher number of iterations to reach the minima than typical Gradient Descent, it is still computationally much less expensive than typical Gradient Descent.

**3. Adaptive Learning Rate**

Before the adaptive learning rate methods were introduced, the gradient descent algorithms including**Batch Gradient Descent** (BGD), **Stochastic Gradient Descent** (SGD) and **mini-Batch Gradient Descent** (mini-BGD, the mixture of BGD and SGD) were state-of-the-art.

The performance of the model on the training dataset can be monitored by the learning algorithm and the learning rate can be adjusted in response.

This is called an adaptive learning rate.

Adaptive learning rate methods are an optimization of gradient descent methods with the goal of minimizing the objective function of a network by using the gradient of the function and the parameters of the network.

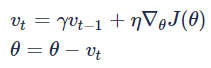
Adaptive learning rates can accelerate training and alleviate some of the pressure of choosing a learning rate and learning rate schedule.

An adaptive learning rate computed at every step is designed to accomplish several goals:

* Remove the hyperparameter, thus relieving us from the need to optimize it
* Speed up the learning process when the loss function hits a plateau
* Solve the problem of exploding gradients

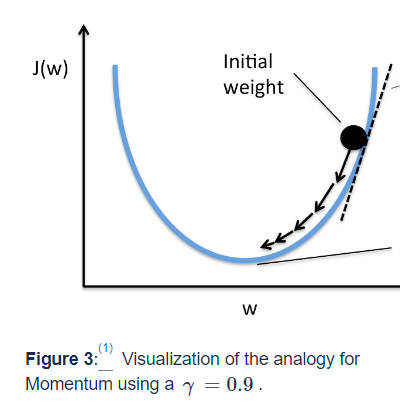
As an improvement to traditional gradient descent algorithms, the adaptive gradient descent optimization algorithms or adaptive learning rate methods can be utilized. Several versions of these algorithms are described below.

**Momentum** can be seen as an evolution of the SGD.



While SGD has problems with data having steep curves in one direction of the gradient, Momentum circumvents that by adding the update vector of the time step before multiplying it with a  γ, usually around 0.9 . As an analogy, one can think of a ball rolling down the gradient, gathering momentum (hence the name), while still being affected by the wind resistance (0< γ < 1).

*Momentum can accelerate training and learning rate schedules can help to converge the optimization process.*



1. <https://builtin.com/data-science/gradient-descent>

2. <https://www.geeksforgeeks.org/ml-stochastic-gradient-descent-sgd/>

3. <https://wiki.tum.de/display/lfdv/Adaptive+Learning+Rate+Method#AdaptiveLearningRateMethod-AdaptiveLearningRateMethod>