Gradient Descent

-

PALISSON Antoine

Optimization Algorithm

An **optimization algorithm** is a method designed to find the best solution to a problem. For example, it can be used to minimize or maximize a function.

Objective Function

This is the heart of the optimization problem – the function that the algorithm seeks to optimize.

In a **minimization problem**, the objective function represents something to be minimized, like cost, error, or risk.

In a **maximization problem**, it might represent profit, efficiency, or likelihood.

+

Variables & Constraints

Optimization problems often involve finding the values of certain **variables that result in the optimal value of the objective function**.

These variables are usually **subject to constraints** which mainly define the feasible region in which the optimal solution must lie.

Types of Optimization

The types of optimization are classified based on different aspects such as the nature of the **objective function**, the **type of variables**, the **presence of constraints** and the **deterministic nature** of the problem.

Linear

Both the objective function and the constraints are linear.

$$L(x) = a_1x_1 + \dots + a_nx_n + b$$

VS

Non-linear

Problems with a nonlinear objective functions and/or constraints.

$$L(x) = x^2$$

Discrete

The variables in these problems take on discrete values.

VS

Continuous

The variables can take any value within a given range.

Constrained

These are constraints on the variables such as equalities or inequalities.

VS

Unconstrained

There are no constraints on the variables. The objective is simply to find the maximum/minimum.

Types of Optimization

The types of optimization are classified based on different aspects such as the nature of the **objective function**, the **type of variables**, the **presence of constraints** and the **deterministic nature** of the problem.

Global

The goal is to find the best solution over the entire set of feasible solutions.

VS

Local

The goal is to find the best solution within a limited region of the space i.e. the local optimum.

Deterministic

The outcome of the optimization is determined solely by the algorithm's input.

VS

Stochastic

There is some degree of randomness/uncertainty in the optimization.

Single-Objective

A single objective function needs to be optimized at the same time.

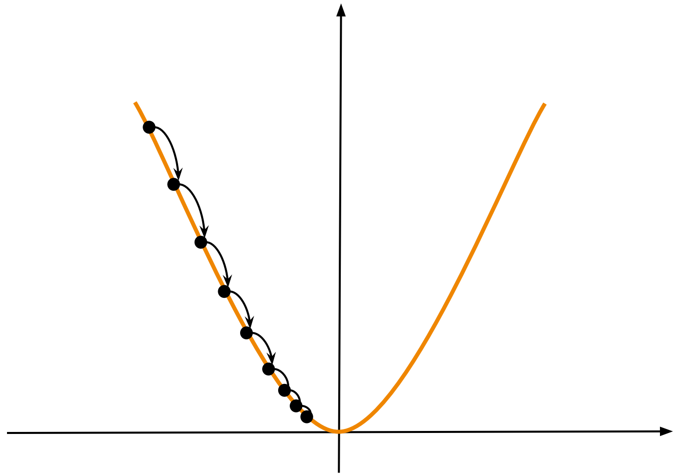
VS

Multi-Objective

Multiple objective functions are optimized at the same time and they often conflict with each other. A trade-off is generally needed.

Gradient Descent – A Simple Definition

Gradient Descent is an optimization algorithm primarily used in machine learning to find an **approximation** of a function's minimum by **iteratively** moving towards its minimum.



Gradient Descent is primarily used to optimize a **single nonlinear objective function**. It works in a **continuous solution space** where the variables can take any value within a given range. It is a **local optimization method** as it moves towards the nearest local minimum from the starting point.

Gradient Descent is generally used for **unconstrained optimization** problems – but can be extended to constrained problems.

Similarly, Gradient Descent is generally a **deterministic algorithm** – if it is run multiple times on the same problem with the same initial conditions, it will follow the same path. However, stochastic variations of Gradient Descent exist.

Gradient Descent – Analogy

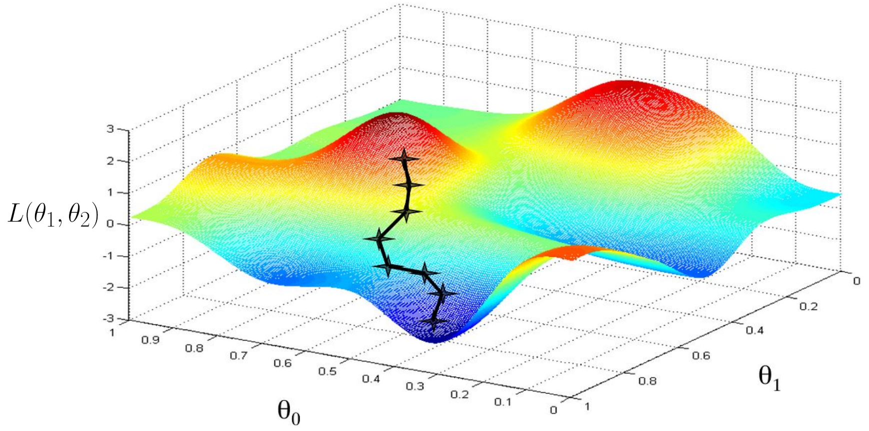
Imagine a person being blindfolded on a hilly terrain and their goal is to find the the bottom of the valley.

Without sight, they use their feet to feel the slope of the ground to estimate the steepest ascent direction at that point.

Knowing that going uphill will lead them away from the valley, they decide to walk in the opposite direction of the steepest slope.

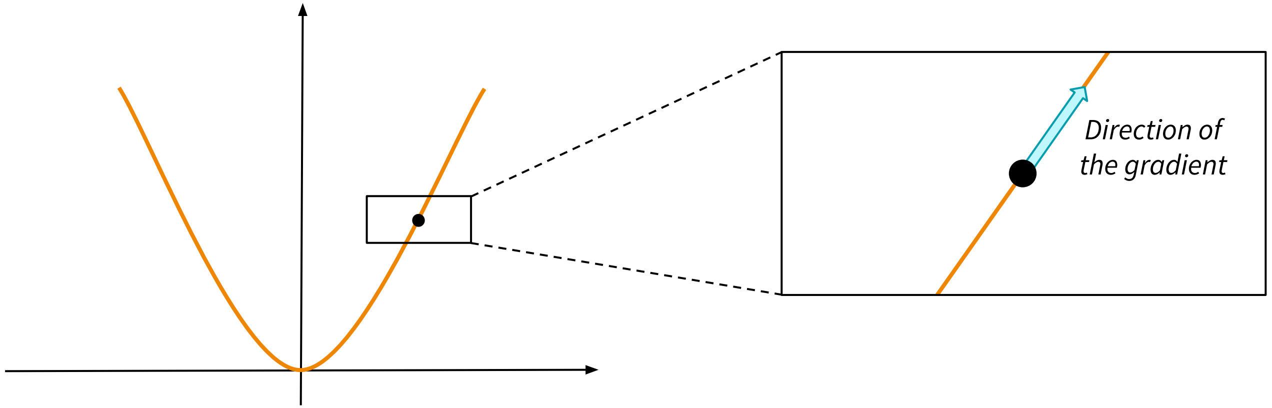
After each step, they pause to feel the slope again because the landscape might change as they move.

As they continue this process, the slope becomes gradually flatter which indicates they are nearing a low point. Eventually, they find a spot where the ground is flat in all directions i.e. the lowest point.



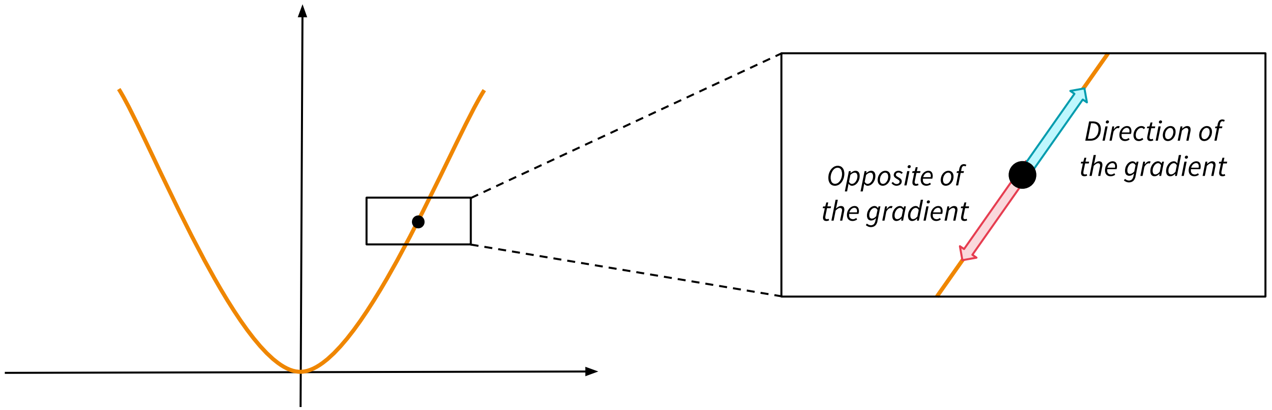
Gradient

The term **gradient** is a mathematical concept that represents the **direction and rate of the steepest increase** of a function at a given point. It tells how much the function value changes if you move a little bit in any direction from a point. The gradient itself is a vector. Its **direction** points towards where the function increases most rapidly and its **magnitude** indicates how steep the increase is.

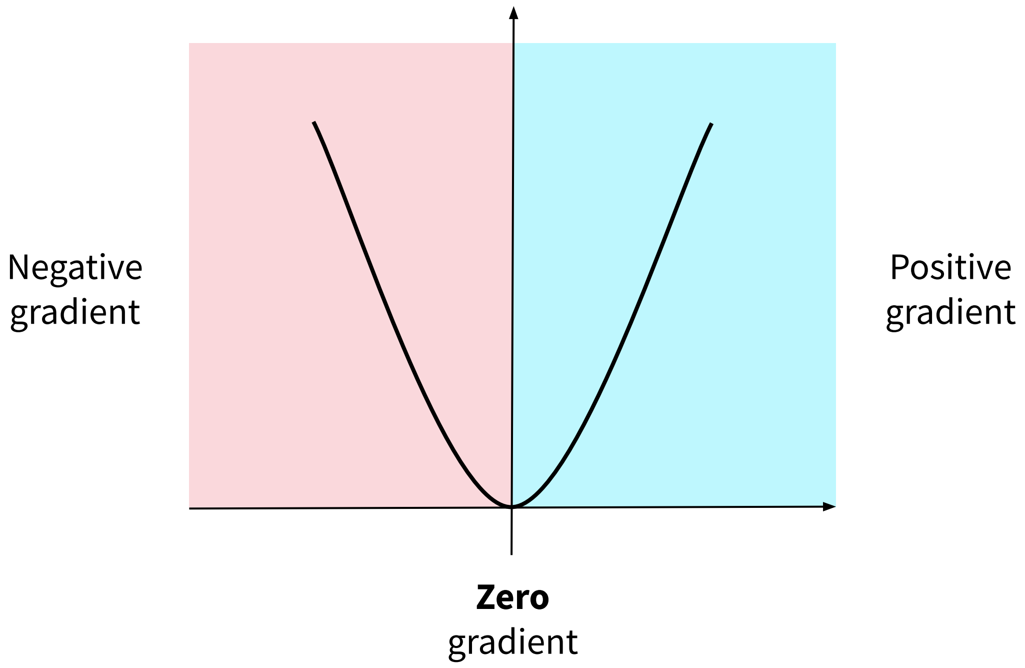


Descent

The term **descent** refers to the process of moving downwards towards the minimum of the function being optimized. As the gradient points in the direction of the steepest ascent – the direction in which the function increases most rapidly – the opposite direction points towards the **steepest descent**. Thus, going in the opposite direction of the gradient is akin to moving towards the minimum of the function.



Convergence



The process of descent continues until the algorithm reaches a point where the gradient is zero – or very close to zero. Further descent won't significantly minimize the function further.

This is known as **convergence**. The point reached is considered an optimal solution for the problem.

Gradient Vector

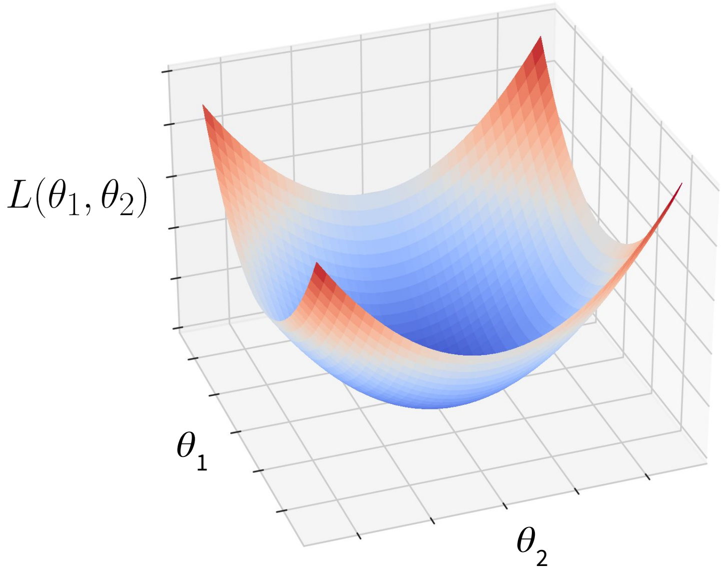
Mathematically, the gradient of a function is a vector consisting of the partial derivatives relative to each variable.

$$\triangledown L\left(\Theta\right) = \begin{bmatrix} \frac{\partial L(\Theta)}{\partial \theta_1} \\ \dots \\ \frac{\partial L(\Theta)}{\partial \theta_n} \end{bmatrix}$$

with $\frac{\partial L(\Theta)}{\partial \theta_1}$ the partial derivative of L with respect to θ_1

with $\frac{\partial L(\Theta)}{\partial \theta_2}$ the partial derivative of L with respect to θ_2

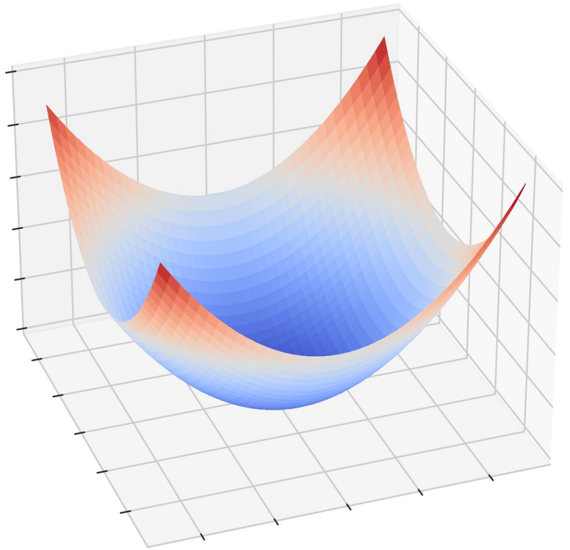
etc …

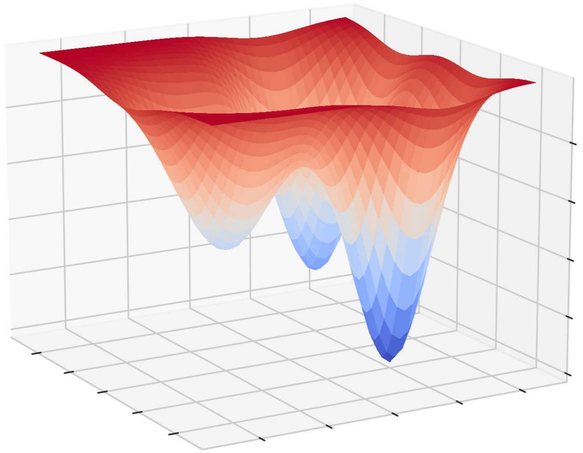


Objective Function

The **objective function** – also known as the cost function or loss function – is the function that is being minimized or maximized. The function must be **differentiable** because Gradient Descent relies on computing partial derivatives. The gradient cannot be computed without differentiability. The function should be **convex** – a function whose graph is shaped like a cup. It's not mandatory but having a function with a single minimum helps Gradient Descent to find the global minimum.

*A **convex** function*

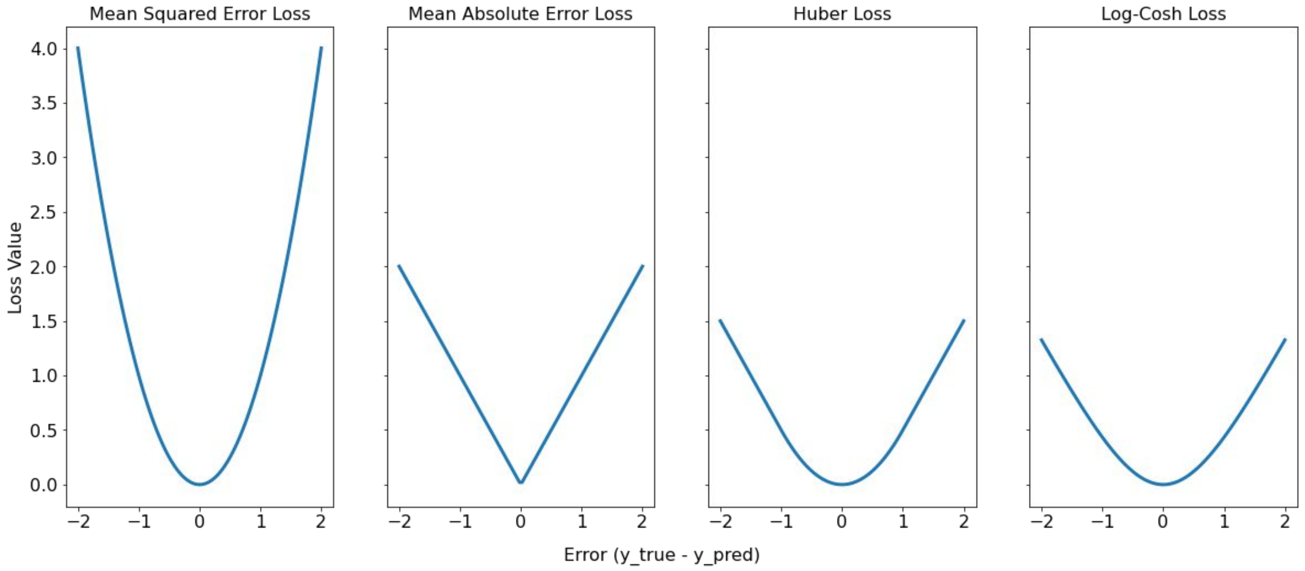




*A **non-convex** function*

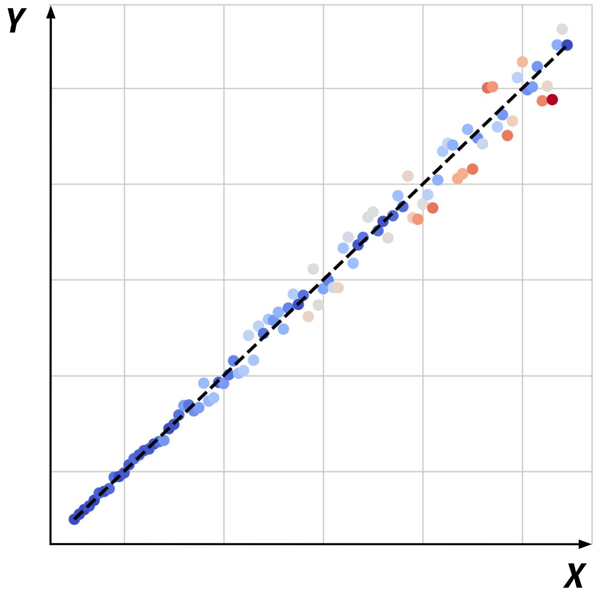
Objective Function

In the context of Machine Learning, an objective function is used to evaluate the “predictive” performances of a machine learning model. The purpose of this function is to guide the training of the model by quantifying how “off” the model's predictions are from the actual results. It generally gives a numerical value that represents the **error associated with the model's predictions** – the lower, the better.



Example – Linear Regression with MSE

$$y_p = \theta X + b$$

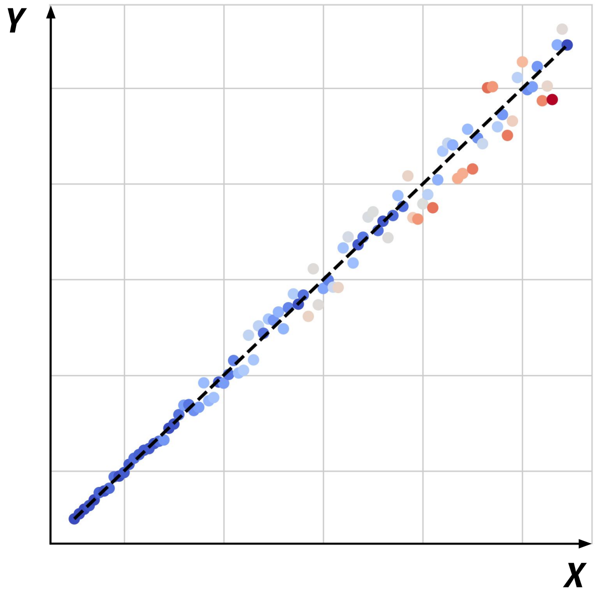


Mean Squared Error

$$\begin{aligned} L(\theta, b) &= \frac{1}{n} \sum (y_{p,i} - y_i)^2 \\ &= \frac{1}{n} \sum (\theta X_i + b - y_i)^2 \end{aligned}$$

Example – Linear Regression with MSE

$$y_p = \theta X + b$$



Gradient Calculation

$$\nabla L(\theta, b) = \begin{bmatrix} \frac{\partial L(\theta,b)}{\partial \theta} \\ \frac{\partial L(\theta,b)}{\partial b} \end{bmatrix}$$

$$\frac{\partial L(\theta, b)}{\partial \theta} = \frac{1}{n} \sum 2(\theta X_i + b - y_i) X_i$$

$$\frac{\partial L(\theta, b)}{\partial b} = \frac{1}{n} \sum 2(\theta X_i + b - y_i)$$

Gradient Descent Formula

$$\Theta_{\text{new}} = \Theta_{\text{old}} - \eta \nabla L\left(\Theta_{\text{old}}\right)$$

The **updated value** of the function's **variables**.

The **previous value** of the function's **variables**.

The **learning rate**.

The **gradient** of the objective function.

Remember, the gradient points towards the steepest ascent. Thus, you need to go in the direction of the steepest descent to minimize the function. That's why the formula updates the variables (parameters) by moving them in the opposite direction of the gradient (the **–** sign).

An Iterative Process

Gradient Descent is a **step-by-step process**. It is a local property that gives information about the slope of the function of the current position giving the current parameter values. A step-by-step process is necessary to reevaluate the gradient at each new position because it can change significantly even with small movements in the parameter space.

By updating the parameters incrementally, Gradient Descent allows for greater control over the optimization process to balance between the speed of convergence and the risk of overshooting the minimum or getting stuck in a local minimum.

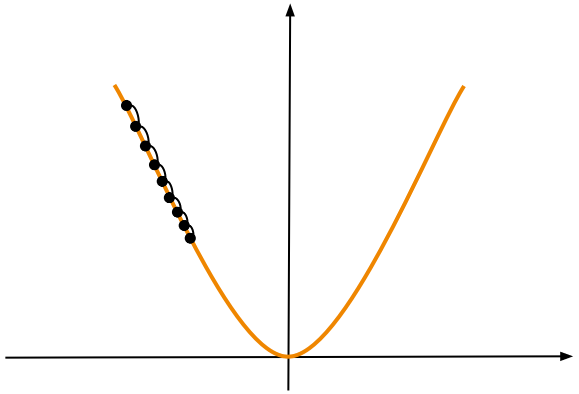
Finally, computing the exact minimum analytically is often computationally infeasible in practical scenarios using large datasets or complex models such as deep neural networks. An iterative process breaks down the problem into smaller and more manageable steps which makes it computationally more efficient.

$$\Theta_{t+1} = \Theta_t - \eta \bigtriangledown L\left(\Theta_t\right)$$

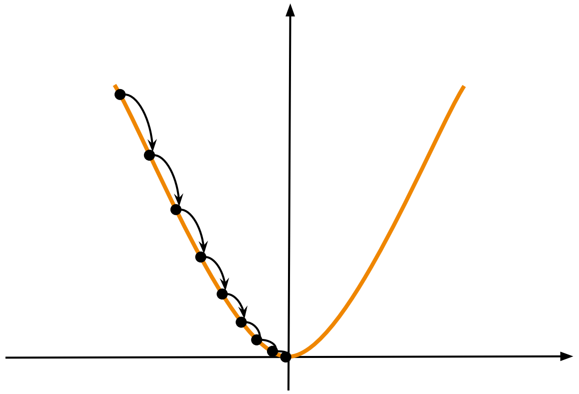
with θ_{t+1} the next value of the parameter and θ_t the previous value.

Learning Rate

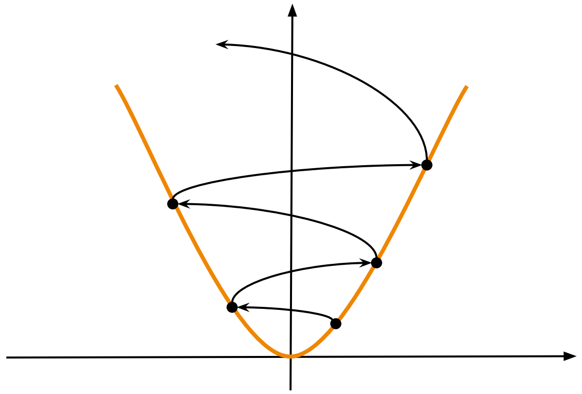
The **learning rate** is a value that determines the size of the steps that the optimization algorithm takes towards the minimum of the objective function. It controls how much the parameters of the model are adjusted during each iteration of the algorithm.



*A learning rate that's **too low** can make the convergence process extremely slow. The algorithm may take too much time or get stuck in a local minimum.*



*The **“perfect” learning rate** is often determined experimentally and can vary widely between different types of problems and datasets.*



*A learning rate that's **too high** can cause the algorithm to overshoot the minimum by diverging or oscillating around it without ever settling.*

Gradient Descent Variants – Batch

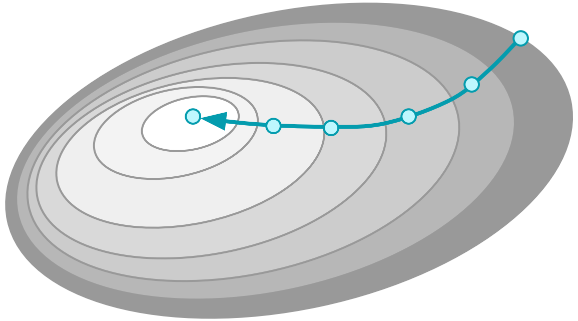
The Gradient Algorithm can be modified by adjusting the number of data samples used to calculate the gradient during each iteration. Such changes don't affect the core Gradient Descent algorithm concept but may still strongly affect its efficiency and convergence.

Batch Gradient Descent

It **uses the entire dataset** to compute the gradient of the objective function at each iteration.

It provides consistent gradient updates leading to a **very stable convergence**.

However, each iteration can be **computationally expensive** because it uses all data samples especially with large datasets.



Gradient Descent Variants – Stochastic

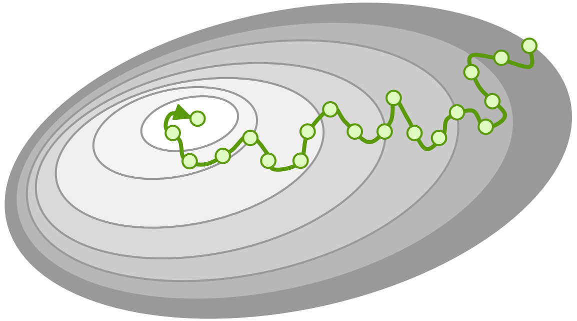
The Gradient Algorithm can be modified by adjusting the number of data samples used to calculate the gradient during each iteration. Such changes don't affect the core Gradient Descent algorithm concept but may still strongly affect its efficiency and convergence.

Stochastic Gradient Descent

It uses a **single random data sample** at each iteration to compute the gradient.

Thus, **each iteration is fast** which makes SGD suitable for large datasets and online learning.

Additionally, it results in high variance in the gradient updates which can help **escape local minima** but can also cause the algorithm to **bounce around the minimum** without exactly converging to it.



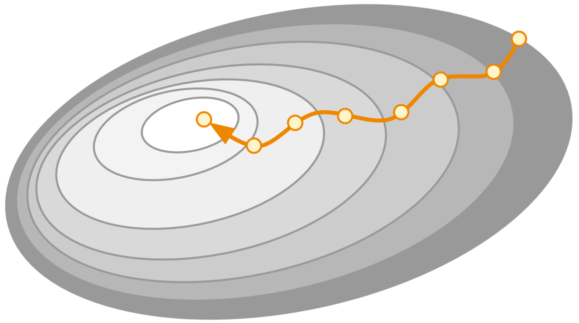
Gradient Descent Variants – Mini-Batch

The Gradient Algorithm can be modified by adjusting the number of data samples used to calculate the gradient during each iteration. Such changes don't affect the core Gradient Descent algorithm concept but may still strongly affect its efficiency and convergence.

Mini-batch Gradient Descent

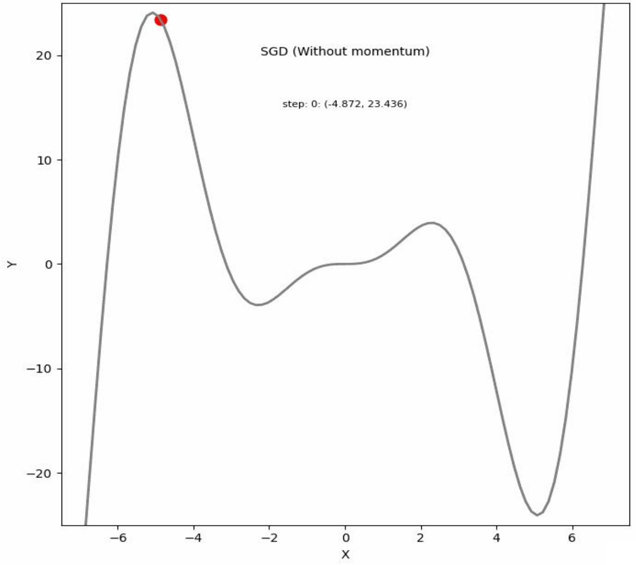
It **uses a subset of the dataset** to compute the gradient for each iteration.

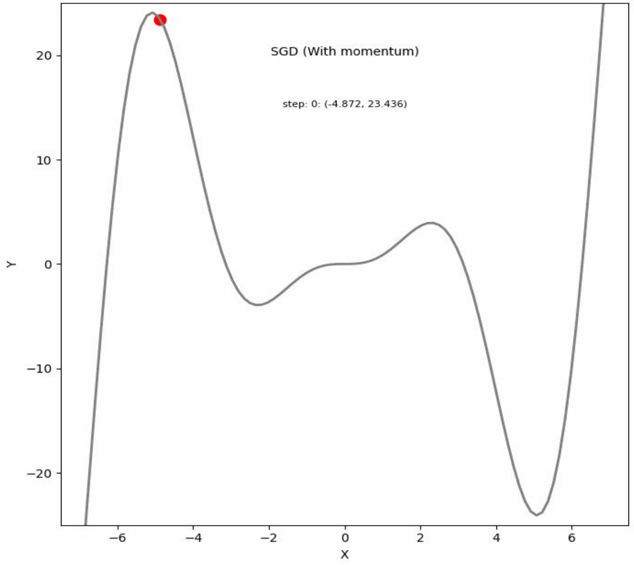
It is a balance between the computational efficiency of Stochastic GD and the stable gradient updates of Batch GD.



Momentum

The **momentum** is a method used to accelerate convergence, help to avoid getting trapped in local minima and stabilize the update especially with noisy gradients (irregular data, complex models …). Inspired by physics, the momentum method adds a fraction of the previous update vector to the current update. It's like a ball rolling downhill, gaining speed in directions with consistent gradients and reducing oscillations in directions where the gradient changes frequently.





Source: gbhat.com

Momentum

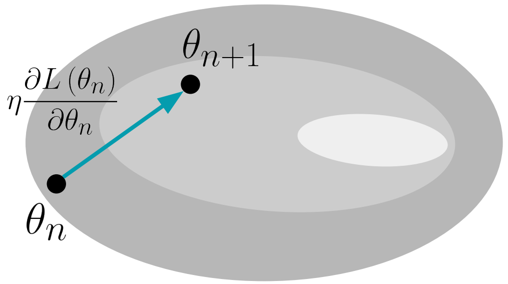
***Z**$_n$ is the momentum term at step n*

$$\boxed{z_n} = \boxed{\beta z_{n-1}} - \eta \frac{\partial L(\theta_n)}{\partial \theta_n}$$

$$\theta_{n+1} = \theta_n + z_n$$

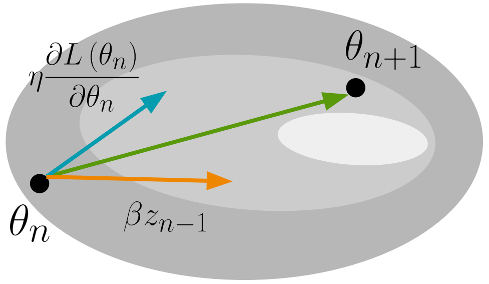
***β** is the momentum coefficient*

without *momentum*



This side shows the update of the parameters purely based on the gradient of the loss function at the current step.

with *momentum*



*This side shows the update of the parameters based on both the gradient of the loss function at the current step (shown in **blue**) and the momentum term from the previous update (shown in **orange**).*

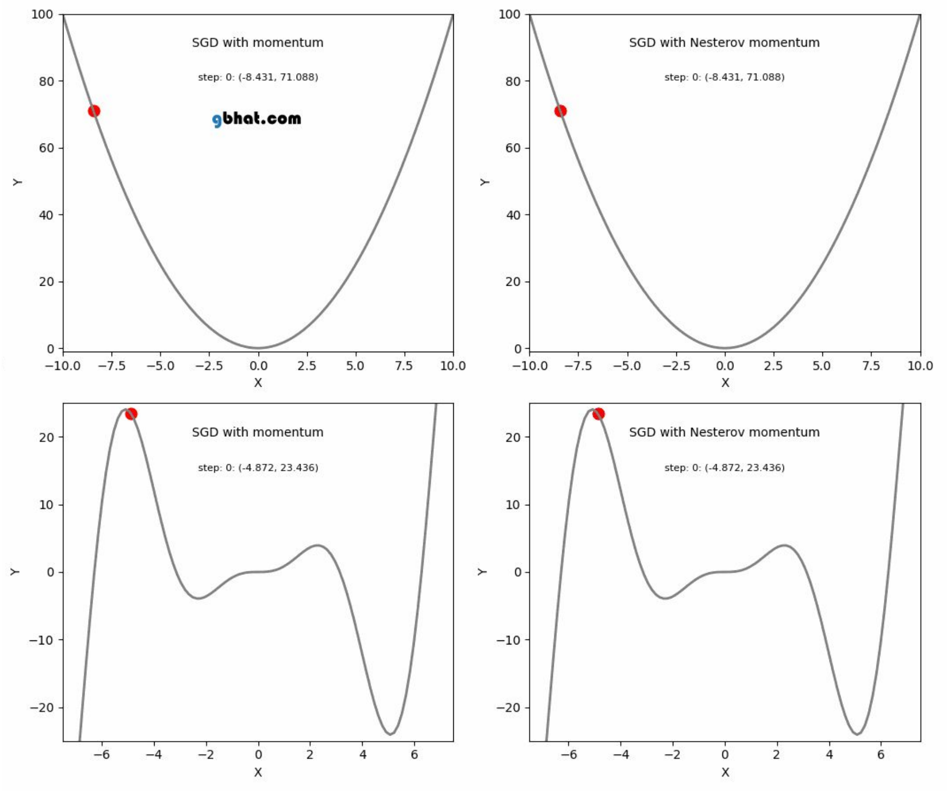
Nesterov Accelerated Gradient

Source: gbhat.com

The **Nesterov Accelerated Gradient** is an improvement on the traditional momentum method.

While both methods aim to accelerate the gradient descent process by incorporating the concept of momentum, NAG introduces a more refined approach by first making a big jump in the direction of the previous accumulated momentum and then measuring the gradient.

By anticipating the future position of the parameters, NAG often leads to faster convergence and reduces the risk of overshooting the minimum than traditional momentum.



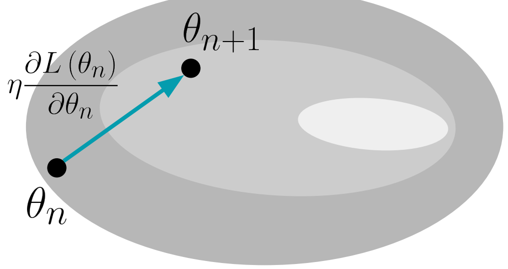
Nesterov Accelerated Gradient

$$z_n = \beta z_{n-1} - \eta \frac{\partial L\left(\boxed{\theta_n + \beta z_{n-1}}\right)}{\partial \theta_n}$$

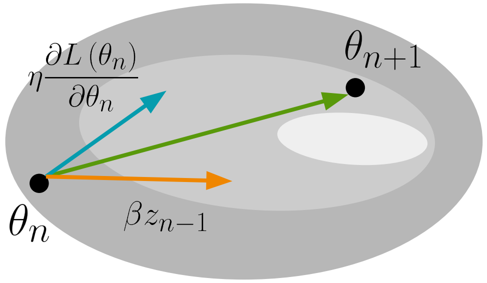
$$\theta_{n+1} = \theta_n + z_n$$

θ_n + βZ_{n-1} *is the "lookahead" step that anticipates the future position of the parameters after the momentum*

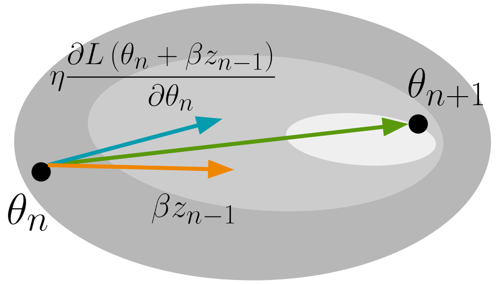
Gradient Descent



Momentum



Nesterov Accelerated Gradient


$\eta\frac{\partial L(\theta_n + \beta z_{n-1})}{\partial \theta_n}$
θ_{n+1}
θ_n
βz_{n-1}