

Master

Machine Learning

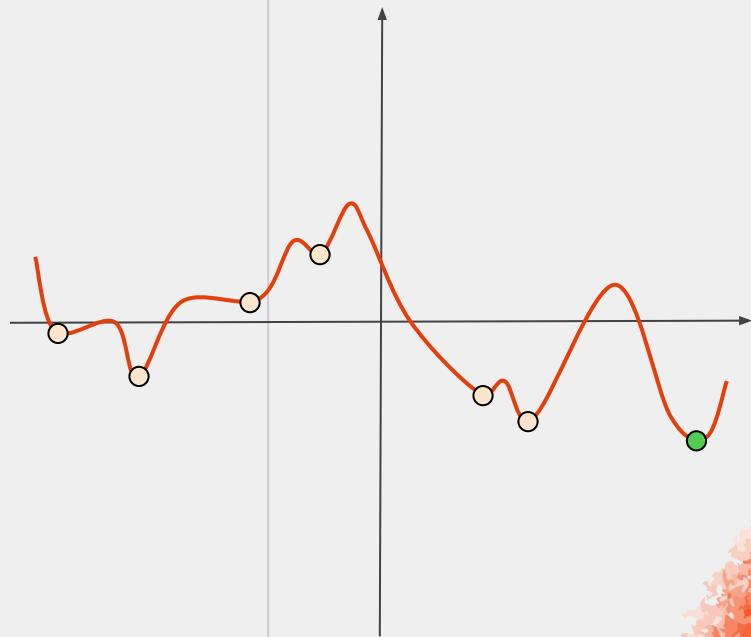
Gradient Descent

Antoine PALISSON

Introduction

Gradient descent is a first-order iterative **optimization** algorithm for **finding a local minimum** of a differentiable function. In other words, it is a process that gradually converges towards the minimum (local) of a function.

This method is commonly used in machine learning and deep learning algorithms to minimise a cost/loss function.



A function with a global minimum and many local minimums

Introduction (2)

An example

An **analogy for Gradient Descent** could be a **person who is stuck in the mountains and wants to get down to the base.**

The person can see the base from where they are, but the path down the mountain is not clear. However, they know that if they move in the direction of the steepest descent, they will eventually reach the base.


The person could start by taking a step in a random direction. They can then look around to see which direction is the steepest and take another step in that direction. They can repeat this process of taking steps in the steepest direction until they reach the base.

The person can also adjust the size of their steps to ensure they do not overshoot the base or get stuck in a local minimum. This is similar to the learning rate used in Gradient Descent, which controls the size of the step taken in the direction of the negative gradient.



Requirements

Unfortunately, the Gradient descent algorithm does not work for all functions. There are two specific requirements:

- The function must be **defined** - *A function is said to be undefined at points outside of its domain. For example, the square root function is not defined for negative numbers.*
 - The function must be **derivable** - *It is a function whose derivative exists at each point in its domain. A continuous function is not necessarily differentiable, but a differentiable function is necessarily continuous.*
- 

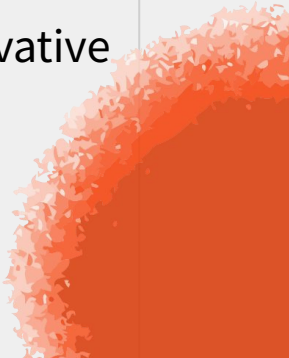


Convexity

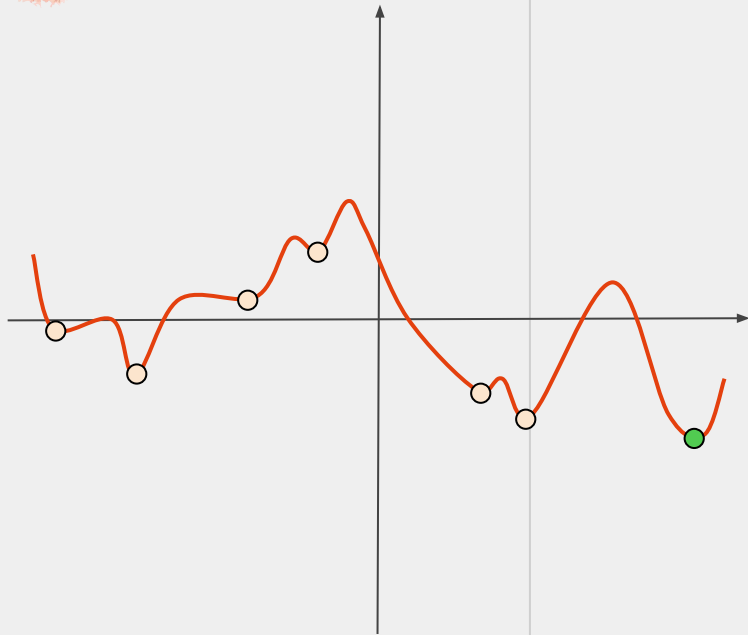
Technically, **convexity** is not a requirement for the gradient descent algorithm. However, if the function is not convex, a local minimum could be found instead of the global minimum.

In simple terms, a convex function refers to a function whose graph is **shaped like a cup**. Some of the most known convex functions are the quadratic function and the exponential function.

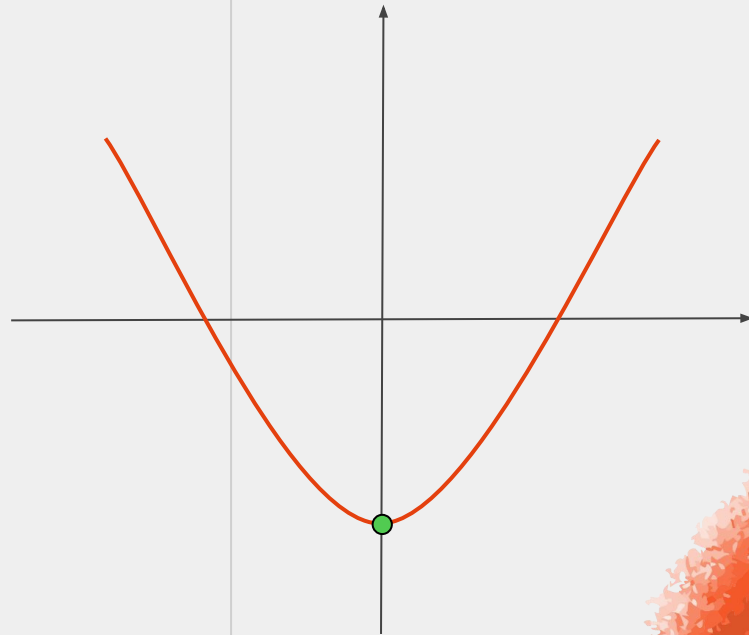
Fortunately, there is an easy way to check for convexity : the second derivative of a convex function is strictly higher than 0.



Convexity (2)



A **non-convex** function

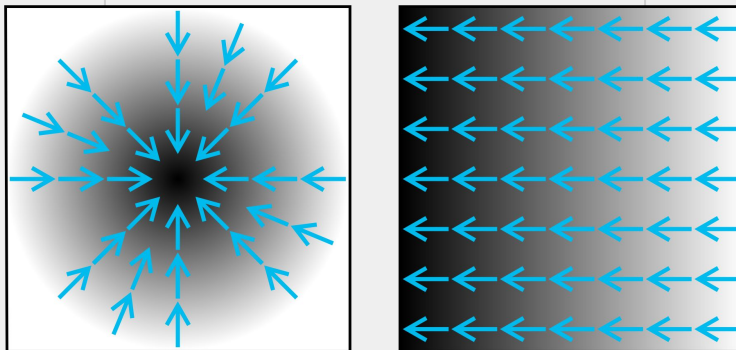


A **convex** function

Gradient

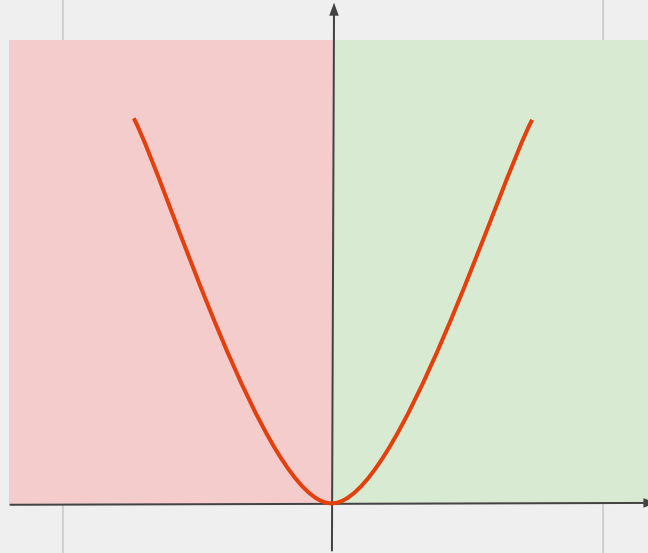
The gradient denotes the **direction and the rate of the greatest change** of a scalar function.

Thus, the gradient is positive whenever the function is increasing and negative whenever the function is decreasing.



Gradient (2)

Negative
gradient

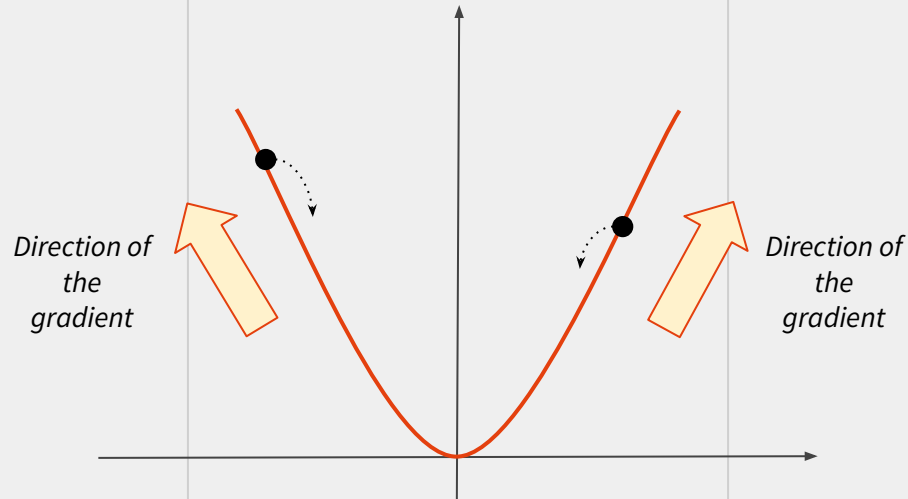


Positive
gradient

Null
gradient

Gradient (3)

The goal of the Gradient Descent is to **minimize the loss function** using its gradient. The method is named Gradient descent because it will go at the **opposite of the gradient** at each iteration.



Gradient (4)

Mathematically, the gradient is simply the **first derivative** at a selected point (in the case of a univariate function) or it is a **vector of derivatives** in each main direction (in the case of a multivariate function).

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

For a function with two variables x and y , the equation becomes

$$\nabla L(\theta_1, \theta_2) = \begin{bmatrix} \frac{\partial L(\theta_1, \theta_2)}{\partial \theta_1} \\ \frac{\partial L(\theta_1, \theta_2)}{\partial \theta_2} \end{bmatrix}$$

Gradient (4)

An example

Model: Linear regression

$$y_p = \theta X + b$$

Loss: Mean Squared Error

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

Gradient: Partial Derivatives

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

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

Algorithm

The gradient is a vector that points in the direction of the steepest increase of the loss function. By subtracting this vector from the current values, we move in the direction of the steepest decrease of the loss function, which is the direction of the minimum.

$$\Theta_{t+1} = \Theta_t - \eta \nabla L(\Theta_t)$$

The next values
matrix

The current
value matrix

The learning
rate

The gradient
matrix

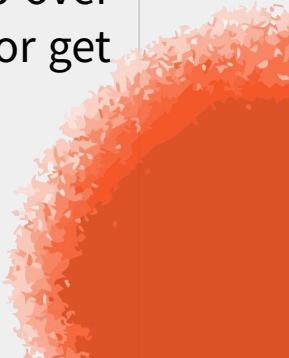


Learning Rate

The **learning rate** is a tuning parameter in an optimization algorithm that determines the step size at each iteration. In other words, it determines the **speed** at which the gradient descent goes.

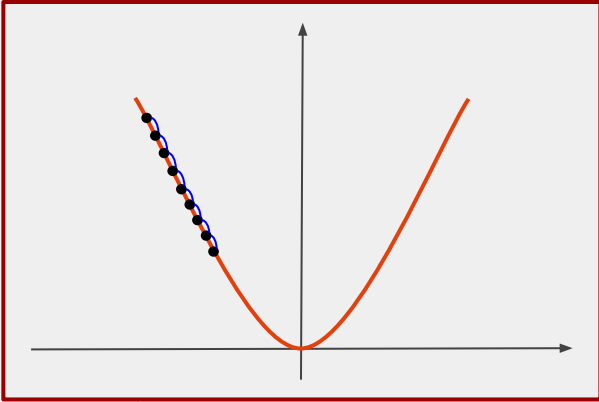
Setting the learning rate for an optimization algorithm always results in a trade-off between the rate of **convergence** and **overshooting**.

Indeed, a too high learning rate will make the optimization process jump over minima but a too low learning rate will either take too long to converge or get stuck in an undesirable local minimum.

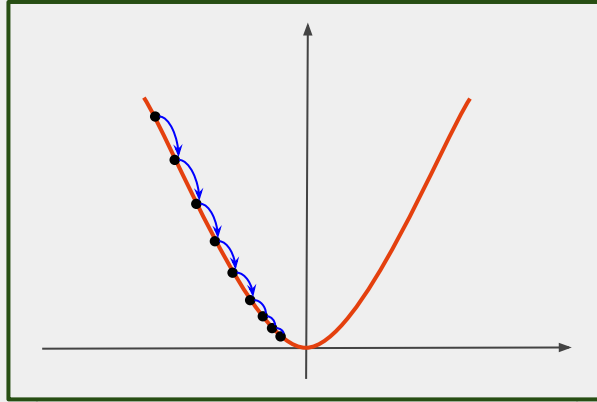


Learning Rate (2)

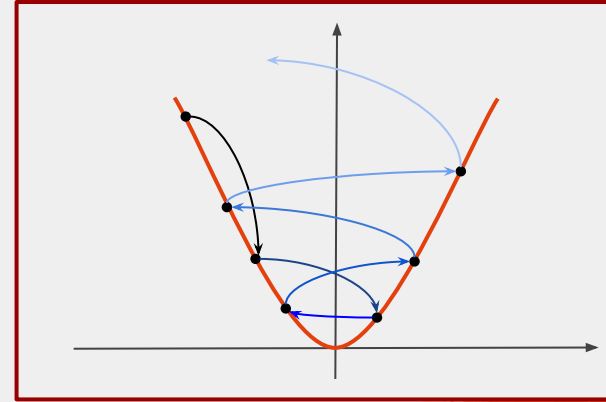
Too small



Good learning rate



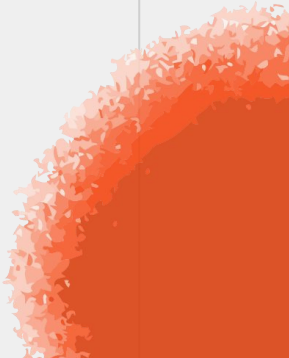
Too big





Gradient Descent Variants

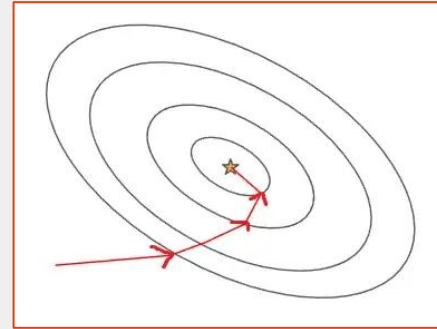
There are three main variants of the Gradient descent method:

- **Batch** Gradient Descent
 - **Mini-Batch** Gradient Descent
 - **Stochastic** Gradient Descent
- 

Batch Gradient Descent

The **Batch Gradient Descent** is the most simple usage of the Gradient Descent algorithm. It performs parameter updates at the end of each epoch. In other words, all the dataset samples are considered at each iteration of the GD algorithm.

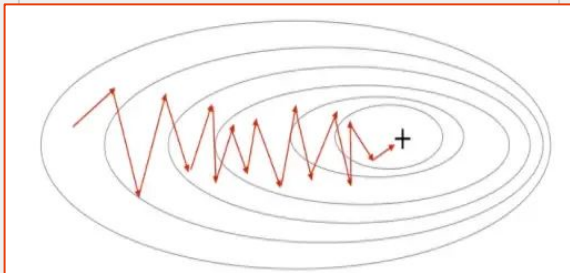
This leads to a **very stable convergence** and a more **direct path** towards the minimum. It is also computationally **efficient** because updates are only required after the run of an epoch i.e. after calculating all the errors. On the other hand, it can **converge at local minima and saddle points**. Moreover, the efficient comes at a cost of a **slower learning speed**.



Stochastic Gradient Descent

The **Stochastic Gradient Descent** is the opposite of the Batch Gradient Descent method. It performs the parameter updates at the end of each step. In other words, the algorithm only see the dataset samples one by one.

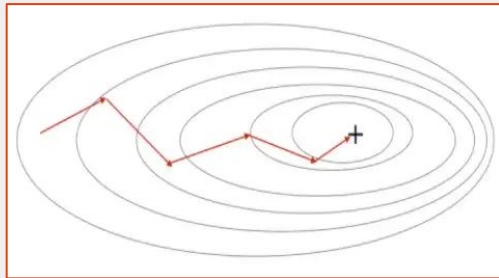
This leads to a **very unstable convergence**. However, it may reach near the minimum (and begin to oscillate around it) faster than Batch Gradient Descent on a large dataset.



Mini-Batch Gradient Descent

The **Mini-Batch Gradient Descent** stands in between the Batch and the stochastic methods as it will select a number of samples between 1 and the length of the dataset.

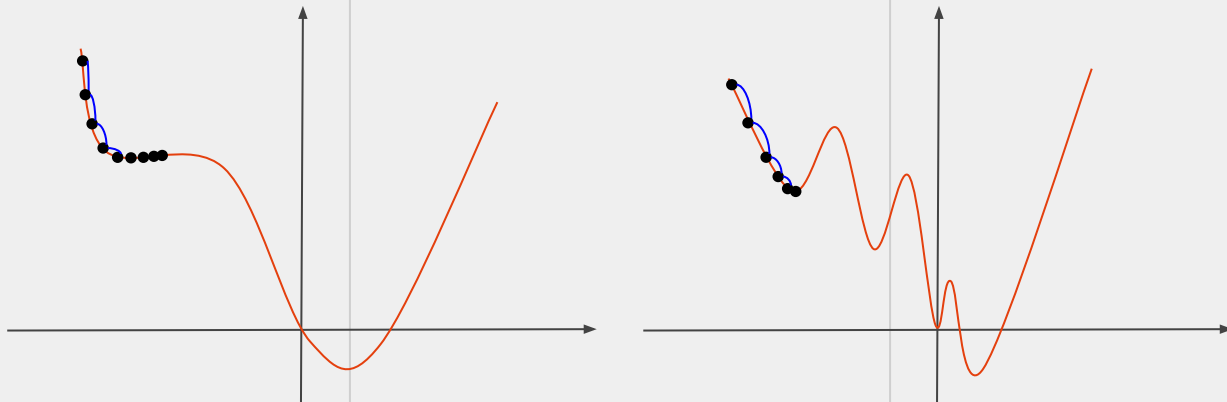
This method has all the benefits from the two other methods. However, it also introduces a new hyperparameter: the number samples (or **batch** size).



Inconvenients

There are two main problems with Gradient descent:

- It may converge to a local minimum
- It slows down in a neighborhood of a saddle point



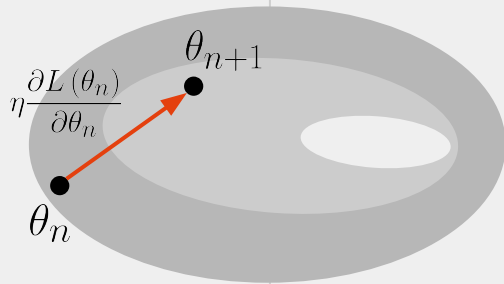
Momentum

Momentum is a technique used in the Gradient Descent algorithm to **accelerate the optimization process**, **improve the convergence rate** and lead to **better generalization performance** especially in situations where the loss function is noisy or has many local minima.

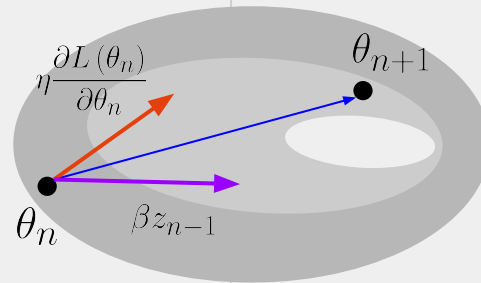
The diagram illustrates the Momentum algorithm with two equations and several annotations:

- Equation 1:**
$$z_n = \beta z_{n-1} - \eta \frac{\partial L(\theta_n)}{\partial \theta_n}$$
 - Annotations:**
 - A blue arrow points from the text "Momentum strength" to the coefficient β .
 - A green arrow points from the text "Previous Momentum vector" to the term z_{n-1} .
 - A purple arrow points from the text "Current Momentum vector" to the term z_n .
- Equation 2:**
$$\theta_{n+1} = \theta_n + z_n$$
 - Annotations:**
 - A red arrow points from the text "Next value" to the term θ_{n+1} .
 - A red arrow points from the text "Current value" to the term θ_n .

Momentum (2)



Gradient Descent



Gradient Descent
with *momentum*

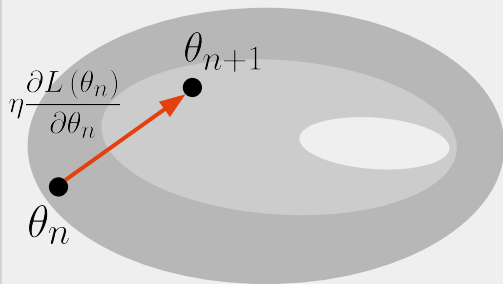
Nesterov

Another way of dealing with the local minimum and the saddle points is to use the **Nesterov Accelerated Gradient** (NAG). NAG includes a correction term that accounts for the change in the gradient caused by the momentum term at the point where the next update would be applied. This allows the algorithm to anticipate the change in the gradient caused by the momentum term and make more informed updates. In many situations, the Nesterov method is more stable than the classical momentum one.

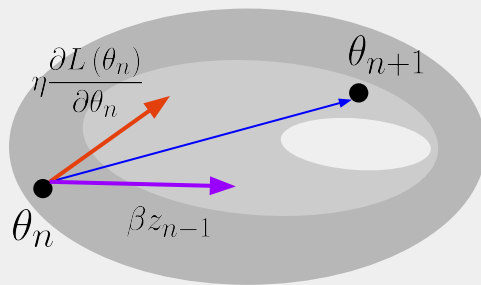
$$z_n = \beta z_{n-1} - \eta \frac{\partial L(\theta_n + \beta z_{n-1})}{\partial \theta_n}$$
$$\theta_{n+1} = \theta_n + z_n$$

Current Loss Function gradient
that takes the **current**
momentum (i.e. the previous
gradient) into account

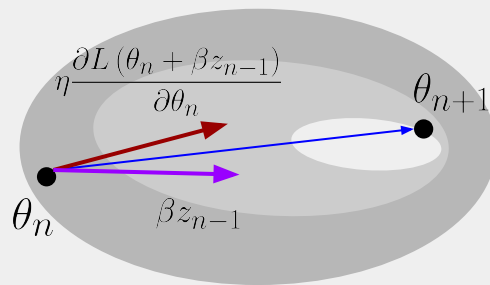
Nesterov (2)



Gradient Descent



Gradient Descent
with **momentum**



Gradient Descent with
Nesterov and **momentum**