## Gradient Descent (GD) and GD with Momentum

### 1. **Basic Gradient Descent (GD):**

In traditional gradient descent, the update rule for parameters $\theta$ is:

$$\theta_{t+1} = \theta_t - \eta \nabla_\theta J(\theta_t)$$

Where:

- $\theta_t$ is the current value of the parameters at step $t$,
- $\eta$ is the learning rate,
- $\nabla_\theta J(\theta_t)$ is the gradient of the cost function $J(\theta)$ with respect to the parameters $\theta$ at step $t$.

In this approach, we update the parameters directly using the gradient at each step.

### 2. **Gradient Descent with Momentum:**

Momentum aims to accelerate gradient descent by adding a fraction of the previous update to the current update. This helps the optimizer maintain consistent directions of movement while reducing oscillations. The update rule with momentum is:

$$v_{t+1} = \beta v_t + (1 - \beta) \nabla_\theta J(\theta_t)$$

$$\theta_{t+1} = \theta_t - \eta v_{t+1}$$

Where:

- $v_t$ is the velocity (the exponentially weighted moving average of the gradients),
- $\beta$ is the momentum term (typically set between 0 and 1, e.g., $\beta = 0.9$),
- $\nabla_\theta J(\theta_t)$ is the gradient at step $t$,
- $\eta$ is the learning rate.

Here, the parameter update is influenced not only by the current gradient but also by a moving average of the previous gradients, helping smooth out oscillations.

**Mathematical Interpretation of Momentum:**

Momentum introduces a moving average of the past gradients into the current update:

$$v_{t+1} = \beta v_t + (1 - \beta)\nabla_\theta J(\theta_t)$$

- $\beta v_t$ represents the contribution from the previous gradient updates (momentum from the past).
- $(1 - \beta)\nabla_\theta J(\theta_t)$ is the current gradient, scaled by $1 - \beta$.

The idea is that this moving average helps the optimizer "build up speed" in directions where the gradient is consistent, and it reduces oscillations in directions where the gradient might fluctuate.

## Pros and Cons of Gradient Descent (GD) and GD with Momentum

**Gradient Descent (GD) without Momentum:**

- **Pros:**
  - **Simplicity**: The basic form of GD is easy to implement and understand.
  - **Guaranteed Convergence**: If the learning rate is appropriately tuned, GD guarantees convergence to a local minimum, especially in convex optimization problems.
  - **Stability**: The updates are straightforward and don’t oscillate around the optimal solution.
- **Cons:**
  - **Slow Convergence**: It may take a long time to converge, especially for ill-conditioned problems (where the gradient behaves differently in different directions).
  - **Sensitive to Learning Rate**: The learning rate needs to be tuned carefully. Too large a learning rate can cause the algorithm to diverge, and too small a learning rate can cause slow convergence.

- **Oscillations**: When the cost function is noisy or has sharp curvatures, GD can oscillate or make slow progress, especially in directions of steep gradients.

**Gradient Descent with Momentum:**

- **Pros:**
  - **Faster Convergence**: Momentum allows the optimizer to accumulate velocity in favorable directions, leading to faster convergence, especially in problems with steep gradients or many local minima.
  - **Reduced Oscillations**: Momentum dampens oscillations in directions where the gradient changes frequently, leading to more stable updates.
  - **Escaping Local Minima**: Momentum helps escape shallow local minima by allowing the optimizer to "roll downhill" more easily.
- **Cons:**
  - **Possibly Overshooting**: The accumulated velocity can cause the optimizer to overshoot the minimum, especially when the learning rate is too high.
  - **Tuning $\beta$ and $\eta$**: The momentum term $\beta$ and the learning rate $\eta$ need to be carefully tuned. Improper tuning can lead to instability or slow convergence.
  - **May Oscillate Before Settling**: In certain cases, momentum can cause the optimization to oscillate around the global minimum before finally converging, especially if the momentum term $\beta$ is too large.

## How Momentum Helps with EWMA of Gradients:

Momentum essentially computes an **Exponentially Weighted Moving Average (EWMA)** of the gradients:

$$v_{t+1} = \beta v_t + (1 - \beta)\nabla_\theta J(\theta_t)$$

This equation is similar to the formula for EWMA, where:

- $v_t$ is the exponentially weighted moving average of the gradients (analogous to the "velocity"),

- $\beta$ is the momentum term that controls the decay of previous gradients' influence,
- $\nabla_\theta J(\theta_t)$ is the current gradient.

The EWMA of the gradients helps the optimizer to "remember" previous gradients, which stabilizes the parameter updates and improves convergence.

## Summary:

- **Without Momentum**: Gradient Descent directly follows the gradients, and it can be slow, especially for noisy or poorly conditioned problems.
- **With Momentum**: It smoothens the updates using a weighted average of previous gradients (EWMA), leading to faster convergence, reduced oscillations, and better performance in challenging optimization landscapes.