Momentum-based Gradient Optimizer - ML

Last Updated : 30 Sep, 2025

Momentum-based gradient optimizers are advanced techniques used to enhance the training of machine learning models. Unlike classic gradient descent, they incorporate a "momentum" term that helps the optimizer navigate the loss surface more efficiently.

This leads to faster convergence, reduced oscillations and improved performance particularly when training deep neural networks or working with large-scale datasets.

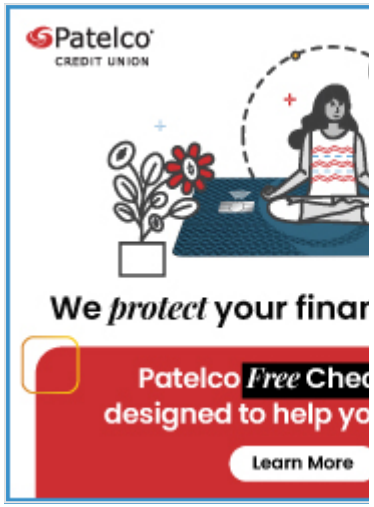
What is Momentum?

Momentum is a concept from physics where an object's motion depends not only on the current force but also on its previous velocity. In the context of gradient optimization it refers to a method that smoothens the optimization trajectory by adding a term that helps the optimizer remember the past gradients.

In mathematical terms the momentum-based gradient descent updates can be described as:

$$v_{t+1} = \beta v_t + (1 - \beta)\nabla L(w_t)$$

$$w_{t+1} = w_t - \eta v_{t+1}$$



Where:

- v_t is the velocity i.e., a running average of gradients
- β is the momentum factor, typically a value between 0 and 1 (often around 0.9)
- $\nabla L(w_t)$ is the current gradient of the loss function
- η is the learning rate

Understanding Hyperparameters:

- **Learning Rate (η)**: The learning rate determines the size of the step taken during each update. It plays a crucial role in both standard gradient descent and momentum-based optimizers.
- **Momentum Factor (β)**: This controls how much of the past gradients are remembered in the current update. A value close to 1 means the optimizer will

have more inertia while a value closer to 0 means less reliance on past gradients.

Working of the Algorithm:

1. **Velocity Update**: The velocity v_t is updated by considering both the previous velocity which represents the momentum and the current gradient. The momentum factor β controls the contribution of the previous velocity to the current update.
2. **Weight Update**: The weights are updated using the velocity v_{t+1} which is a weighted average of the past gradients and the current gradient.

Types of Momentum-Based Optimizers

There are several variations of momentum-based optimizers each with slight modifications to the basic momentum algorithm:

1. Nesterov Accelerated Gradient (NAG)

Nesterov momentum is an advanced form of momentum-based optimization. It modifies the

update rule by calculating the gradient at the upcoming position rather than the current position of the weights.

The update rule becomes:

$$v_{t+1} = \beta v_t + \nabla L(w_t - \eta \beta v_t)$$

$$w_{t+1} = w_t - \eta v_{t+1}$$

NAG is considered more efficient than classical momentum because it has a better understanding of the future trajectory, leading to even faster convergence and better performance in some cases.

2. AdaMomentum

AdaMomentum combines the concept of adaptive learning rates with momentum. It adjusts the momentum term based on the recent gradients making the optimizer more sensitive to the landscape of the loss function. This can help in fine-tuning the convergence process.

3. RMSProp (Root Mean Square Propagation)

Although not strictly a momentum-based optimizer in the traditional sense RMSProp incorporates a form of momentum by adapting the learning rate for each parameter. It's particularly effective when dealing with non-stationary objectives such as in training recurrent neural networks (RNNs).

Advantages

- **Faster Convergence**: It helps to accelerate the convergence by considering past gradients which helps the model navigate through flat regions more efficiently.
- **Reduces Oscillation**: Traditional gradient descent can oscillate when there are steep gradients in some directions and flat gradients in others. Momentum reduces this oscillation by maintaining the direction of previous updates.
- **Improved Generalization**: By smoothing the optimization process, momentum-based methods can lead to better generalization on unseen data, preventing overfitting.
- **Helps Avoid Local Minima**: The momentum term can help the optimizer escape from local minima by maintaining a strong enough "velocity" to continue moving past these suboptimal points.

Challenges and Considerations

- **Choosing Hyperparameters**: Selecting the appropriate values for the learning rate and momentum factor can be challenging. Typically a momentum factor of 0.9 is common but it may vary based on the specific problem or dataset.
- **Potential for Over-Accumulation**: If the momentum term becomes too large it can lead to the optimizer overshooting the minimum, especially in the presence of noisy gradients.
- **Initial Momentum**: When momentum is initialized it can have a significant impact on the convergence rate. Poor initialization can lead to slow or erratic optimization behavior.

Suggested Quiz 6 Questions

Which of the following is not an advantage of momentum-based optimizers?

A Faster convergence

B Reduced oscillations

C Automatic feature selection

D Ability to escape local minima

Login to View Explanation 1/6 < Previous Next >

Comment N Nikhil ... + Follow 6

Article Tags: Machine Learning AI-ML-DS

AI-ML-DS With Python

Explore

Machine Learning Basics

Python for Machine Learning

Feature Engineering

Supervised Learning

Unsupervised Learning

Model Evaluation and Tuning

Advanced Techniques

Machine Learning Practice