Optimization Algorithms in Machine Learning

> **From Gradient Descent to Adam — the science of optimization, decoded.**

📘 Overview

Optimization algorithms are at the heart of Machine Learning. They determine how a model's parameters are updated to minimize loss. This resource covers **Gradient Descent** and its key variants — **Momentum, RMSProp, Adam** — with formulas, explanations, and examples.

🎯 Why Optimization Matters

- Models are defined by parameters (weights & biases).
- Training = minimizing a **loss function** using optimization.
- Poor optimization → slow convergence or getting stuck in bad minima.
- Good optimization → faster learning, stable training, better accuracy.

⚙️ 1. Gradient Descent

Core Idea: Move parameters in the direction opposite to the gradient of the loss.

Formula:

$$\theta = \theta - \eta \cdot \nabla_\theta J(\theta)$$

- θ: parameters
- η: learning rate
- $J(\theta)$: loss function

Types:

- **Batch Gradient Descent** → full dataset each step (stable, but slow).
- **Stochastic Gradient Descent (SGD)** → one sample at a time (fast, noisy).
- **Mini-batch Gradient Descent** → balance of both (most common).

Pitfalls:

- Choosing learning rate is tricky. Too high → divergence, too low → slow.
- Can get stuck in local minima or plateaus.

⚙️ 2. Gradient Descent with Momentum

Core Idea: Add velocity to updates, like rolling a ball downhill — smooths oscillations and speeds up convergence.

Formulas:

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

\theta = \theta - v_t

- β: momentum term (0.9 typical).
- Keeps moving in the same direction if gradients are consistent.

Pros: Faster convergence, less zig-zag in ravines.

Cons: Can overshoot if momentum too high.

⚙️ 3. RMSProp (Root Mean Square Propagation)

Core Idea: Adjust learning rate per parameter by dividing by a moving average of squared gradients.

Formulas:

$$s_t = \beta s_{t-1} + (1 - \beta)(\nabla_\theta J(\theta))^2$$

$$\theta = \theta - \frac{\eta}{\sqrt{s_t + \epsilon}} \cdot \nabla_\theta J(\theta)$$

- Helps avoid vanishing/exploding steps.
- Good for recurrent networks and non-stationary problems.

Pros: Adaptive step sizes, stabilizes learning.

Cons: May still be unstable without momentum.

⚙️ 4. Adam (Adaptive Moment Estimation)

Core Idea: Combines **Momentum** + **RMSProp**. Keeps track of both first moment (mean) and second moment (variance) of gradients.

Formulas:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1)\nabla_\theta J(\theta)$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2)(\nabla_\theta J(\theta))^2$$

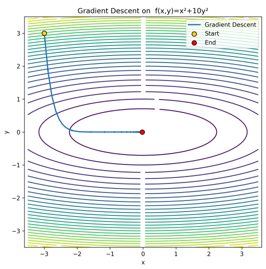
$$\theta = \theta - \frac{\eta \cdot \hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}$$

- $\hat{m}_t$, $\hat{v}_t$: bias-corrected estimates.
- Default params: $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\eta = 0.001$.

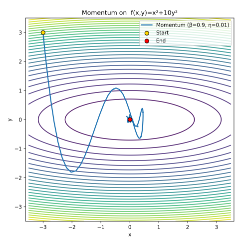
Pros: Works well out-of-the-box, robust for many ML problems.

Cons: Sometimes overfits or generalizes worse than SGD in some cases.

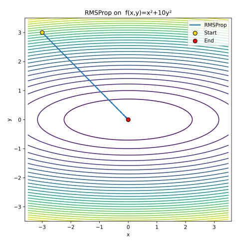
📊 Conceptual Graphs



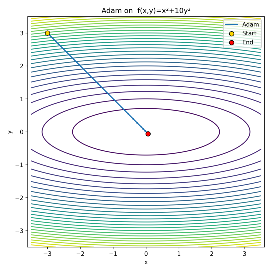
Gradient Descent: zig-zag path down a valley



Momentum: smoother, curved path like a rolling ball



RMSProp: steps adapt in x/y directions, preventing overshoot



Adam: balanced adaptive steps, often shortest path to minimum

---

🔑 Quick Comparison

Algorithm	Key Idea	Pros	Cons
Gradient Descent	Update with gradient only	Simple, widely used	Slow, sensitive to η
Momentum	Add velocity term	Faster, less oscillation	Can overshoot

Algorithm	Key Idea	Pros	Cons
RMSProp	Scale learning rate by variance	Stable, adaptive	May need tuning
Adam	Momentum + RMSProp	Fast, default choice	Sometimes overfits

🚀 Takeaway

- Start with **Adam** as baseline.
- For large-scale, try **SGD + Momentum** (often better generalization).
- Use **RMSProp** for RNNs or noisy problems.
- Always tune **learning rate** — the single most important hyperparameter.