Machine Learning

Salvador Ruiz Correa

February 11, 2026

Agenda:
1 Gradient.
2 Gradient D

Gradient descent is a first-order iterative optimization algorithm that minimizes a loss function by updating model parameters in the opposite direction of the gradient—the vector of partial derivatives that indicates the direction of steepest ascent. At each iteration, parameters are adjusted proportionally to the negative gradient scaled by a learning rate, gradually descending the loss landscape toward a local minimum. The algorithm's variants—batch, stochastic, and mini-batch gradient descent—balance computational efficiency and convergence stability, while advanced optimizers such as Momentum, RMSprop, and Adam enhance basic gradient descent with adaptive learning rates and momentum terms. This fundamental algorithm underpins nearly all neural network training, from simple linear regression to large-scale deep learning models, enabling the automated discovery of complex patterns in high-dimensional parameter spaces through iterative, gradient-guided refinement.

The Gradient

Definition

Let $f : \mathbb{R}^n \to \mathbb{R}$ be a scalar-valued function that is differentiable at a point $\mathbf{x} = (x_1, x_2, \ldots, x_n) \in \mathbb{R}^n$. The **gradient** of f at $\mathbf{x}$, denoted $\nabla f(\mathbf{x})$, is a vector in $\mathbb{R}^n$ defined as:

$$\nabla f(\mathbf{x}) = \begin{bmatrix} \frac{\partial f}{\partial x_1}(\mathbf{x}) \\ \frac{\partial f}{\partial x_2}(\mathbf{x}) \\ \vdots \\ \frac{\partial f}{\partial x_n}(\mathbf{x}) \end{bmatrix} \in \mathbb{R}^n$$

Equivalently, in operator notation:

$$\nabla = \left(\frac{\partial}{\partial x_1}, \frac{\partial}{\partial x_2}, \ldots, \frac{\partial}{\partial x_n} \right)$$

Geometric Interpretation

The gradient admits three fundamental geometric interpretations:

1. Direction of Steepest Ascent

At any point $\mathbf{x}_0$, the gradient $\nabla f(\mathbf{x}_0)$ points in the **direction of the greatest rate of increase** of f. For any unit vector $\mathbf{u} \in \mathbb{R}^n$, the directional derivative satisfies:

$$D_{\mathbf{u}} f(\mathbf{x}_0) = \nabla f(\mathbf{x}_0) \cdot \mathbf{u} \leq \|\nabla f(\mathbf{x}_0)\|$$

with equality if and only if $\mathbf{u} = \nabla f(\mathbf{x}_0) / \|\nabla f(\mathbf{x}_0)\|$. Thus, the normalized gradient gives the direction of steepest ascent.

2. Magnitude as Rate of Change

The **magnitude** (or norm) of the gradient,

$$\|\nabla f(\mathbf{x}_0)\| = \sqrt{\left(\frac{\partial f}{\partial x_1}\right)^2 + \cdots + \left(\frac{\partial f}{\partial x_n}\right)^2},$$

equals the **maximum rate of change** of f at $\mathbf{x}_0$. A larger magnitude indicates a steeper slope.

3. Orthogonality to Level Sets

The gradient is **perpendicular (orthogonal)** to the level set (contour) of f passing through $\mathbf{x}_0$. If $f(\mathbf{x}) = c$ defines a level surface, then $\nabla f(\mathbf{x}_0)$ is normal to this surface at $\mathbf{x}_0$:

$$\nabla f(\mathbf{x}_0) \cdot \mathbf{v} = 0 \quad \text{for all tangent vectors } \mathbf{v} \text{ to the level set at } \mathbf{x}_0.$$

Visualization in $\mathbb{R}^2$

For a function $f : \mathbb{R}^2 \to \mathbb{R}$, the gradient $\nabla f(x, y) = (f_x, f_y)^T$ has a clear visual interpretation:

- **Arrows** point uphill, perpendicular to contour lines.
- **Arrow length** indicates slope steepness.
- **Contour density** corresponds to gradient magnitude.

Properties

$$\begin{aligned}
\text{Linearity:} \quad & \nabla(af + bg) = a\nabla f + b\nabla g, \quad a, b \in \mathbb{R} && (1)\\
\text{Product rule:} \quad & \nabla(fg) = f\nabla g + g\nabla f && (2)\\
\text{Chain rule:} \quad & \nabla(f \circ g)(\mathbf{x}) = f'(g(\mathbf{x}))\nabla g(\mathbf{x}) && (3)\\
\text{Quotient rule:} \quad & \nabla\left(\frac{f}{g}\right) = \frac{g\nabla f - f\nabla g}{g^2}, \quad g(\mathbf{x}) \neq 0 && (4)
\end{aligned}$$

Examples

$$\begin{aligned}
f(x, y) &= x^2 + y^2 & \nabla f(x, y) &= (2x, 2y)^T\\
f(x, y) &= \sin(x)\cos(y) & \nabla f(x, y) &= (\cos(x)\cos(y), -\sin(x)\sin(y))^T\\
f(\mathbf{x}) &= \|\mathbf{x}\|^2 = \mathbf{x} \cdot \mathbf{x} & \nabla f(\mathbf{x}) &= 2\mathbf{x}\\
f(\mathbf{x}) &= \|\mathbf{x}\| = \sqrt{\mathbf{x} \cdot \mathbf{x}} & \nabla f(\mathbf{x}) &= \frac{\mathbf{x}}{\|\mathbf{x}\|}, \quad \mathbf{x} \neq \mathbf{0}
\end{aligned}$$

Critical Points

A point $\mathbf{x}_0$ is called a **critical point** if $\nabla f(\mathbf{x}_0) = \mathbf{0}$. At such points:

- f may have a **local minimum**, **local maximum**, or **saddle point**.
- The Hessian matrix $Hf(\mathbf{x}_0)$ (matrix of second partial derivatives) determines the nature of the critical point.

Applications

- **Optimization:** Gradient descent uses $-\nabla f(\mathbf{x})$ to find minima.
- **Physics:** Conservative forces satisfy $\mathbf{F} = -\nabla U$ (force is negative gradient of potential).
- **Computer vision:** Image gradients detect edges.
- **Machine learning:** Backpropagation computes gradients of loss functions w.r.t. parameters.
- **Partial differential equations:** ∇ appears in diffusion, heat, and wave equations.

The Optimization Problem in Machine Learning

At the heart of supervised machine learning lies an optimization problem. Given a dataset $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^N$ of N i.i.d. samples, we seek parameters $\theta \in \mathbb{R}^d$ that minimize the expected loss under the data distribution. Since the true distribution is unknown, we instead minimize the *empirical risk*:

$$\mathcal{L}(\theta) = \frac{1}{N} \sum_{i=1}^{N} \ell(\mathbf{x}_i, y_i; \theta) \tag{5}$$

where $\ell(\cdot)$ is a loss function measuring prediction error. This formulation encompasses linear regression (squared loss), logistic regression (cross-entropy loss), and neural networks (arbitrary differentiable loss functions).

The optimization problem is therefore:

$$\theta^* = \arg\min_{\theta \in \mathbb{R}^d} \mathcal{L}(\theta) \tag{6}$$

For most models of interest, $\mathcal{L}(\theta)$ is:

- **Non-convex** in the case of neural networks with hidden layers
- **High-dimensional** with d ranging from thousands to billions
- **Differentiable** almost everywhere (or subdifferentiable)

Gradient Descent: The Fundamental Algorithm

Geometric Intuition

Consider a differentiable function $\mathcal{L} : \mathbb{R}^d \to \mathbb{R}$. From multivariable calculus, the gradient $\nabla \mathcal{L}(\theta) \in \mathbb{R}^d$ is a vector pointing in the *direction of steepest ascent* of $\mathcal{L}$ at θ. Its negative, $-\nabla \mathcal{L}(\theta)$, points in the direction of steepest descent.

Definition 0.1 (Gradient)**.** For a differentiable function $\mathcal{L} : \mathbb{R}^d \to \mathbb{R}$, the gradient at θ is:

$$\nabla \mathcal{L}(\theta) = \begin{bmatrix} \frac{\partial \mathcal{L}}{\partial \theta_1}(\theta) \\ \frac{\partial \mathcal{L}}{\partial \theta_2}(\theta) \\ \vdots \\ \frac{\partial \mathcal{L}}{\partial \theta_d}(\theta) \end{bmatrix} \in \mathbb{R}^d \tag{7}$$

The geometric interpretation is fundamental:

- The gradient vector is orthogonal to the level sets of $\mathcal{L}$
- Its magnitude $\|\nabla \mathcal{L}(\theta)\|$ quantifies the local steepness
- Moving in the direction $-\nabla \mathcal{L}(\theta)$ maximally reduces $\mathcal{L}$

Derivation via Taylor Expansion

Consider a small perturbation $\Delta\theta$. The first-order Taylor approximation gives:

$$\mathcal{L}(\theta + \Delta\theta) \approx \mathcal{L}(\theta) + \nabla\mathcal{L}(\theta)^\top \Delta\theta \tag{8}$$

We seek $\Delta\theta$ that minimizes this approximation subject to $\|\Delta\theta\| \leq \eta$. By Cauchy-Schwarz inequality, the minimum occurs when:

$$\Delta\theta = -\eta \frac{\nabla\mathcal{L}(\theta)}{\|\nabla\mathcal{L}(\theta)\|} \tag{9}$$

Absorbing the norm into the step size yields the gradient descent update:

$$\boxed{\theta_{t+1} = \theta_t - \eta_t \nabla\mathcal{L}(\theta_t)} \tag{10}$$

where $\eta_t > 0$ is the *learning rate* (step size).

Algorithm: Batch Gradient Descent

Algorithm 1:

Require: Initial parameters θ_0, learning rate schedule $\{\eta_t\}_{t=0}^{\infty}$

1: **while** not converged **do**
2: Compute gradient on full dataset: $g_t = \frac{1}{N}\sum_{i=1}^{N} \nabla\ell(\mathbf{x}_i, y_i; \theta_t)$
3: Update parameters: $\theta_{t+1} = \theta_t - \eta_t g_t$
4: $t \leftarrow t + 1$
5: **end whilereturn** θ_t

Convergence Analysis for Convex Functions

Theorem 0.1 (Convergence of Gradient Descent)**.** Assume $\mathcal{L}$ is convex and L-smooth ($\|\nabla\mathcal{L}(\theta) - \nabla\mathcal{L}(\vartheta)\| \leq L\|\theta - \vartheta\|$). With constant step size $\eta = 1/L$, gradient descent satisfies:

$$\mathcal{L}(\theta_T) - \mathcal{L}(\theta^*) \leq \frac{L\|\theta_0 - \theta^*\|^2}{2T} \tag{11}$$

This implies a convergence rate of $\mathcal{O}(1/T)$.

For strongly convex functions (μ-strongly convex), the rate improves to linear (geometric) convergence:

$$\|\theta_T - \theta^*\|^2 \leq \left(1 - \frac{\mu}{L}\right)^T \|\theta_0 - \theta^*\|^2 \tag{12}$$

Limitations of Batch Gradient Descent

Despite its theoretical elegance, batch gradient descent suffers from critical practical limitations:

1. **Computational cost**: Each iteration requires processing all N samples. When $N \gg 10^6$, this becomes prohibitively expensive.
2. **Redundancy**: Large datasets contain substantial redundancy; computing the exact gradient on all samples is wasteful.

3. **Poor local minima**: For non-convex functions, the deterministic nature can lead to convergence to sharp local minima that generalize poorly.
4. **Online learning**: Cannot accommodate streaming data where the full dataset is unavailable.
These limitations motivated the development of stochastic variants.

Stochastic Gradient Descent (SGD)

Core Idea: Unbiased Gradient Estimation

Instead of computing the exact gradient on all N samples, SGD estimates the gradient using a *single randomly sampled* data point:

$$g_t(\theta_t) = \nabla \ell(\mathbf{x}_{i_t}, y_{i_t}; \theta_t), \quad i_t \sim \text{Uniform}\{1, \ldots, N\} \tag{13}$$

The key insight is that this estimate is *unbiased*:

$$\mathbb{E}_{i_t}[g_t(\theta_t)] = \frac{1}{N} \sum_{i=1}^{N} \nabla \ell(\mathbf{x}_i, y_i; \theta_t) = \nabla \mathcal{L}(\theta_t) \tag{14}$$

Algorithm 2: Descent (SGD

Require: Initial parameters θ_0, learning rate schedule $\{\eta_t\}_{t=0}^{\infty}$

1: **while** not converged **do**
2: Randomly sample index $i_t \sim \text{Uniform}\{1, \ldots, N\}$
3: Compute stochastic gradient: $g_t = \nabla \ell(\mathbf{x}_{i_t}, y_{i_t}; \theta_t)$
4: Update: $\theta_{t+1} = \theta_t - \eta_t g_t$
5: $t \leftarrow t + 1$
6: **end whilereturn** θ_t

Convergence Properties

The introduction of stochasticity fundamentally changes the optimization dynamics. The update becomes:

$$\theta_{t+1} = \theta_t - \eta_t \left(\nabla \mathcal{L}(\theta_t) + \xi_t \right) \tag{15}$$

where $\xi_t = g_t - \nabla \mathcal{L}(\theta_t)$ is a zero-mean noise vector with covariance $\Sigma(\theta_t) = \text{Cov}[\nabla \ell(\mathbf{x}_i, y_i; \theta_t)]$.

Theorem 0.2 (Convergence of SGD with Diminishing Step Sizes)**.** Assume $\mathcal{L}$ is convex and L-smooth, and the stochastic gradients have bounded variance: $\mathbb{E}[\|g_t - \nabla \mathcal{L}(\theta_t)\|^2] \leq \sigma^2$. With step sizes $\eta_t = \mathcal{O}(1/t)$, SGD satisfies:

$$\mathbb{E}[\mathcal{L}(\bar{\theta}_T)] - \mathcal{L}(\theta^*) \leq \mathcal{O}\left(\frac{1}{\sqrt{T}} \right) \tag{16}$$

where $\bar{\theta}_T = \frac{1}{T} \sum_{t=1}^{T} \theta_t$.

This $\mathcal{O}(1/\sqrt{T})$ rate is *slower* than batch gradient descent's $\mathcal{O}(1/T)$ rate in terms of iterations, but *faster* in terms of computational cost. Each SGD iteration costs $\mathcal{O}(1)$ data accesses versus $\mathcal{O}(N)$ for batch GD.

The Trade-off: Noise vs. Speed

The crucial trade-off can be summarized as:

- **Batch GD**: More accurate gradient estimates, faster convergence per iteration, but prohibitively expensive per iteration for large N.
- **SGD**: Noisy gradient estimates, slower convergence per iteration, but extremely cheap per iteration.

For large datasets, SGD reaches a reasonable solution with far less total computation. Specifically, to achieve ϵ-optimality:

- Batch GD: $\mathcal{O}(N \log(1/\epsilon))$ operations
- SGD: $\mathcal{O}(1/\epsilon)$ operations (independent of N!)

The Role of the Learning Rate

The learning rate η_t is critical for SGD convergence. Common schedules include:

$$\text{Constant:} \quad \eta_t = \eta_0 \tag{17}$$

$$\text{Step decay:} \quad \eta_t = \eta_0 \cdot \gamma^{\lfloor t/T_0 \rfloor} \tag{18}$$

$$\text{Exponential decay:} \quad \eta_t = \eta_0 e^{-\lambda t} \tag{19}$$

$$\text{Polynomial decay:} \quad \eta_t = \eta_0 (1 + \alpha t)^{-\beta}, \quad \beta \in (0.5, 1] \tag{20}$$

$$\text{Inverse time:} \quad \eta_t = \frac{\eta_0}{1 + \alpha t} \tag{21}$$

The theoretical condition for almost sure convergence is:

$$\sum_{t=1}^{\infty} \eta_t = \infty \quad \text{and} \quad \sum_{t=1}^{\infty} \eta_t^2 < \infty \tag{22}$$

This ensures the algorithm can travel arbitrarily far while eventually dampening the noise.

Mini-Batch Gradient Descent

The Compromise

Mini-batch gradient descent interpolates between batch GD and SGD by using m samples per iteration ($1 < m \ll N$):

$$g_t = \frac{1}{m} \sum_{i \in \mathcal{B}_t} \nabla \ell(\mathbf{x}_i, y_i; \theta_t) \tag{23}$$

where $\mathcal{B}_t \subset \{1, \ldots, N\}$ is a randomly selected *mini-batch* of size m.

Algorithm 3: Descent

Require: Initial parameters θ_0, learning rate η, batch size m

1: **while** not converged **do**
2: Randomly sample mini-batch $\mathcal{B}_t$ of size m
3: $g_t = \frac{1}{m} \sum_{i \in \mathcal{B}_t} \nabla \ell(\mathbf{x}_i, y_i; \theta_t)$
4: $\theta_{t+1} = \theta_t - \eta_t g_t$
5: $t \leftarrow t + 1$
6: **end whilereturn** θ_t

Variance Reduction

The variance of the mini-batch gradient estimator is:

$$\text{Var}[g_t] = \frac{1}{m}\text{Var}[\nabla \ell(\mathbf{x}_i, y_i; \theta_t)] \tag{24}$$

This decays linearly with batch size m. Thus, mini-batching provides a direct trade-off between computational cost per iteration and gradient estimate quality.

Computational Efficiency

Modern hardware (GPUs, TPUs) provides massive parallelism for matrix operations. Mini-batching enables:

- **Vectorization**: Computing m gradients simultaneously is often m times faster than computing them sequentially.
- **Memory hierarchy**: Optimal batch sizes (typically 32-512) exploit GPU memory bandwidth.
- **Distributed training**: Mini-batches can be sharded across multiple devices.

Batch Size Dynamics

Research has revealed a rich phenomenology:

- **Small batches** ($m \sim 1 - 32$): High noise, better generalization, slower convergence
- **Medium batches** ($m \sim 32 - 512$): Optimal trade-off for most tasks
- **Large batches** ($m \sim 1024+$): Stable gradients, faster distributed training, but often poorer generalization

The generalization gap—where large batches yield worse test performance—remains an active research area, hypothesized to relate to the noise scale:

$$\text{Noise scale} \approx \frac{\eta}{m}\text{Var}[\nabla \ell] \tag{25}$$

Unified View and Comparison

Algorithmic Family

All three algorithms can be expressed in a unified framework:

$$\theta_{t+1} = \theta_t - \eta_t \cdot \frac{1}{|\mathcal{B}_t|} \sum_{i \in \mathcal{B}_t} \nabla \ell(\mathbf{x}_i, y_i; \theta_t) \tag{26}$$

where:

- **Batch GD**: $\mathcal{B}_t = \{1, \ldots, N\}$ (deterministic, full dataset)
- **SGD**: $|\mathcal{B}_t| = 1$ (single sample)
- **Mini-batch**: $1 < |\mathcal{B}_t| < N$ (small subset)

Comparative Analysis

Convergence Rate in Wall-Clock Time

The relevant metric for practitioners is not iterations but *wall-clock time*. Let $T_{\text{iter}}(m)$ be the time per iteration with batch size m. The time to reach ϵ-optimality is:

Property	Batch GD	SGD	Mini-Batch
Gradient variance	0	High	Moderate
Per-iteration cost	$\mathcal{O}(Nd)$	$\mathcal{O}(d)$	$\mathcal{O}(md)$
Convergence rate	$\mathcal{O}(1/T)$	$\mathcal{O}(1/\sqrt{T})$	$\mathcal{O}(1/\sqrt{T})$
Data passes	Full passes	Single samples	Batches
Parallelization	Poor	Poor	Excellent
Escape saddle points	Difficult	Easy	Moderate

Table 1: Comp
scent variants
dimension, m

$$T_{\text{total}}(\epsilon) = \frac{\text{Iterations to } \epsilon}{\text{Time per iteration}} = \mathcal{O}\left(\frac{1}{\epsilon^2} \cdot T_{\text{iter}}(m)\right) \tag{27}$$

Since $T_{\text{iter}}(m)$ scales sublinearly in m due to parallelization, the optimal batch size balances reduced iterations against increased per-iteration cost.

Practical Considerations

Gradient Clipping

For deep networks, exploding gradients can destabilize training. Gradient clipping bounds the gradient norm:

$$g_t \leftarrow \begin{cases} g_t & \text{if } \|g_t\| \leq C \\ \frac{C}{\|g_t\|} g_t & \text{otherwise} \end{cases} \tag{28}$$

This ensures each update step is bounded, providing numerical stability.

Warmup and Learning Rate Scheduling

Modern practice often includes:

- **Linear warmup**: Gradually increase η from 0 to $\eta_{\max}$ over first few epochs
- **Cosine decay**: $\eta_t = \eta_{\min} + \frac{1}{2}(\eta_{\max} - \eta_{\min})(1 + \cos(\frac{t}{T}\pi))$
- **Reduce on plateau**: Decrease η when validation loss stagnates

Epoch vs. Iteration

In machine learning parlance:

- **Iteration**: One parameter update (one mini-batch)
- **Epoch**: One full pass through the dataset (N/m iterations)
  Models are typically trained for multiple epochs, repeatedly cycling through the dataset.

Beyond Classical SGD: Momentum and Acceleration

Momentum

Momentum accelerates gradient descent by accumulating a velocity vector:

$$v_{t+1} = \beta v_t + (1 - \beta) g_t \tag{29}$$

$$\theta_{t+1} = \theta_t - \eta v_{t+1} \tag{30}$$

where $\beta \in [0, 1)$ is the momentum coefficient (typically 0.9). This:

- Dampens oscillations in narrow valleys
- Accelerates convergence in consistent gradient directions
- Helps escape shallow local minima

Nesterov Accelerated Gradient

Nesterov momentum computes the gradient at the *lookahead* position:

$$\tilde{\theta}_t = \theta_t + \beta v_t \tag{31}$$

$$v_{t+1} = \beta v_t + \eta \nabla \mathcal{L}(\tilde{\theta}_t) \tag{32}$$

$$\theta_{t+1} = \theta_t - v_{t+1} \tag{33}$$

This correction term yields improved convergence rates for convex functions and practical benefits for deep learning.

Summary and Conclusion

Gradient descent and its stochastic variants form the bedrock of modern machine learning optimization. The evolution from batch to stochastic to mini-batch methods represents a fundamental shift in optimization philosophy—from precise but expensive updates to noisy but efficient ones.

Key takeaways:

1. **Batch gradient descent** provides the theoretical foundation but is computationally intractable for large datasets.
2. **Stochastic gradient descent** enables learning from massive datasets through unbiased gradient estimation, trading per-iteration accuracy for computational efficiency.
3. **Mini-batch gradient descent** offers a practical compromise, leveraging hardware parallelism while maintaining gradient diversity.
4. The **learning rate** is the most critical hyperparameter, requiring careful scheduling for convergence.
5. **Batch size** controls the variance-stochasticity trade-off, with implications for both optimization speed and generalization.

The continued development of adaptive methods (Adam, RMSprop, AdaGrad) has further improved upon vanilla SGD, but the fundamental principles remain rooted in the gradient descent framework presented here. Understanding this foundation is essential for effective machine learning practice and for developing novel optimization algorithms.

Mathematical Prerequisites

Convexity

A function $f : \mathbb{R}^d \to \mathbb{R}$ is:

- **Convex** if $f(\alpha x + (1-\alpha)y) \leq \alpha f(x) + (1-\alpha) f(y)$ for all $\alpha \in [0, 1]$.
- **Strongly convex** with parameter $\mu > 0$ if $f(y) \geq f(x) + \nabla f(x)^\top (y - x) + \frac{\mu}{2}\|y - x\|^2$.
- **L-smooth** if $\|\nabla f(x) - \nabla f(y)\| \leq L\|x - y\|$.

Probability Bounds

For SGD analysis, we frequently use:

- **Jensen's inequality**: $f(\mathbb{E}[X]) \leq \mathbb{E}[f(X)]$ for convex f
- **Markov's inequality**: $\Pr(X \geq a) \leq \mathbb{E}[X]/a$
- **Chebyshev's inequality**: $\Pr(|X - \mathbb{E}[X]| \geq a) \leq \text{Var}(X)/a^2$

Taylor's Theorem

For twice-differentiable f:

$$f(y) = f(x) + \nabla f(x)^\top (y - x) + \frac{1}{2}(y - x)^\top \nabla^2 f(\xi)(y - x) \tag{34}$$

for some ξ on the line segment $[x, y]$.