SPECTRUM OF ENGINEERING SCIENCES
ISSN (E) 3007-3138 (P) 3007-312X VOLUME 4, ISSUE 3, 2026

A MATHEMATICAL ANALYSIS OF GRADIENT DESCENT AND ITS VARIANTS IN ARTIFICIAL INTELLIGENCE

Maaz Ahmad[*1], Mubeen Yaseen[2], Syed Ale Hassan Rizvi[3], Faiza Shafqat[4], Darain Fatima[5]

[*1]*Department of Mathematics Post graduate College Mardan affiliated with Abdul wali khan University Mardan*
[2]*Department of Mathematics University of Sargodha*
[3]*Electrical and Telecommunications Engineering Ahlul Bayt International University, Tehran*
[4]*Department of Mathematics University of Sargodha*
[5]*Department of Mathematics Riphah International University Islamabad (Faisalabad Campus), Pakistan*

[*1]maaz1908078@gmail.com, [2]mubeenyaseen470@gmail.com, [3]alehassan09@gmail.com, [4]zimalahmadali4@gmail.com, [5]darainfatima51272@gmail.com

DOI: https://doi.org/10.5281/zenodo.23876196

Keywords
Artificial Intelligence; Optimization Algorithms; Gradient Descent; Stochastic Gradient Descent; Adaptive Optimization; Convergence Analysis; Machine Learning

Article History
Received: 01 January 2026
Accepted: 16 March 2026
Published: 31 March 2026

Copyright @Author
Corresponding Author: *
Maaz Ahmad

Abstract
Gradient descent is the core AI optimization algorithm : it iteratively steps along the negative gradient to find the parameters that minimize a loss function . In this paper we present a mathematical analysis of the algorithm and its main variants. The aims are to develop the theoretical background of gradient descent, to derive and compare its variants and to analyze their convergence behavior and computational efficiency. Analysis of first and second order derivatives, Hessian and convexity classification of objective functions. Formulation Batch, stochastic and mini-batch descent Momentum and Nesterov acceleration Adaptive methods: AdaGrad, RMSProp and Adam. We consider convergence in terms of smoothness, strong convexity and non-convexity and we differentiate between linear, sublinear and stochastic rates. A numerical experiment on the convex quadratic f(x,y)=x^2+4y^2 verifies the theory quantitatively: full-gradient descent achieves a gradient norm below 〖10〗^(-3) in 19 iterations at a final loss of 4.83x 〖10〗^(-8), RMSProp in 38 iterations, while stochastic descent stalls at a variance floor near 3.31x 〖10〗^(-3). The results show that no variant dominates: each trades stability, speed and adaptivity against the others, and the guaranties available under convexity degrade to mere stationarity when the landscape is non-convex. The choice of optimizer is mathematical, not a conventional, choice.

2. INTRODUCTION

2.1 Background of Optimization in Artificial Intelligence

Machine learning is, at its core, an optimization problem. Training a model means searching a parameter space $\Theta \subseteq \mathbb{R}^d$ for a vector θ that minimizes an objective function, usually the empirical loss

$$L(\theta) = \frac{1}{n}\sum_{i=1}^{n} \ell(f(x_i;\theta), y_i),$$

where f is the model, (x_i, y_i) are training examples, and ℓ measures prediction error. Linear regression, logistic classification, support vector machines and deep networks all fit this template; what differs is the geometry of the loss and the dimension of the parameter space.

The loss function is the single quantity that connects model design to learning: it converts a task ("classify these images") into a numerical surface that an algorithm can traverse. Minimizing it is therefore not a preliminary step but the act of learning itself. A model whose loss

is not reduced has not extracted structure from the data, regardless of how well it is designed.

In neural networks the stakes rise sharply. Because the objective is composed through many layers, $L(\theta)$ becomes non-convex, and d can reach millions. Optimization here is the mechanism of representation learning: the optimizer decides whether training escapes poor regions of the surface, how quickly error falls, and how much computation is consumed before a usable model emerges.

2.2 Research Problem

Despite its ubiquity, gradient-based optimization is only partially understood, and four limitations motivate this study.

First, classical methods were designed under convexity and smoothness assumptions. Batch gradient descent, Newton's method and quasi-Newton methods carry strong guarantees — linear or quadratic convergence — that do not transfer to deep networks. Newton-type updates also require Hessian information, costing $O(d^2)$ memory and $O(d^3)$ time per step, which is infeasible at modern scale.

Second, convergence is often slow. Ill-conditioning, narrow ravines, flat plateaus and saddle points force a single global step size to serve directions of very different curvature, while vanishing gradients suppress updates in early layers.

Third, non-convexity weakens what can be guaranteed. For a general non-convex L, first-order methods ensure only that $\|\nabla L(\theta)\| \leq \varepsilon$ at some iterate, and that statement is compatible with stopping at a saddle point rather than a minimum.

Fourth, computational cost is severe: an exact gradient costs $O(nd)$ per iteration, and stochastic estimates trade that cost for variance that prevents the loss from settling. These tensions — speed against accuracy, theory against practice — define the problem this paper addresses.

2.3 Research Objectives

- To analyze the mathematical foundations of gradient descent: its derivation as steepest descent, the descent lemma, and the conditions under which convergence is guaranteed.
- To examine the principal variants — batch, stochastic and mini-batch descent, momentum and Nesterov acceleration, and the adaptive methods AdaGrad, RMSProp and Adam.
- To evaluate convergence behaviour across convex, strongly convex, non-convex, and Polyak–Łojasiewicz settings, including the effect of step-size schedules and batch size.
- To compare computational efficiency and accuracy: cost per iteration, iteration count to a target loss, and final loss and generalization performance.

2.4 Research Questions

1. How does gradient descent minimize loss functions — by what mechanism, and under what conditions does the iterates' loss decrease monotonically and converge?
2. How do its variants improve optimization — through variance reduction, momentum, or adaptive per-coordinate scaling — and what is gained and lost in each case?
3. Which optimization methods perform better under different mathematical conditions, such as convex versus non-convex objectives, ill-conditioned curvature, and noisy or large-scale data?

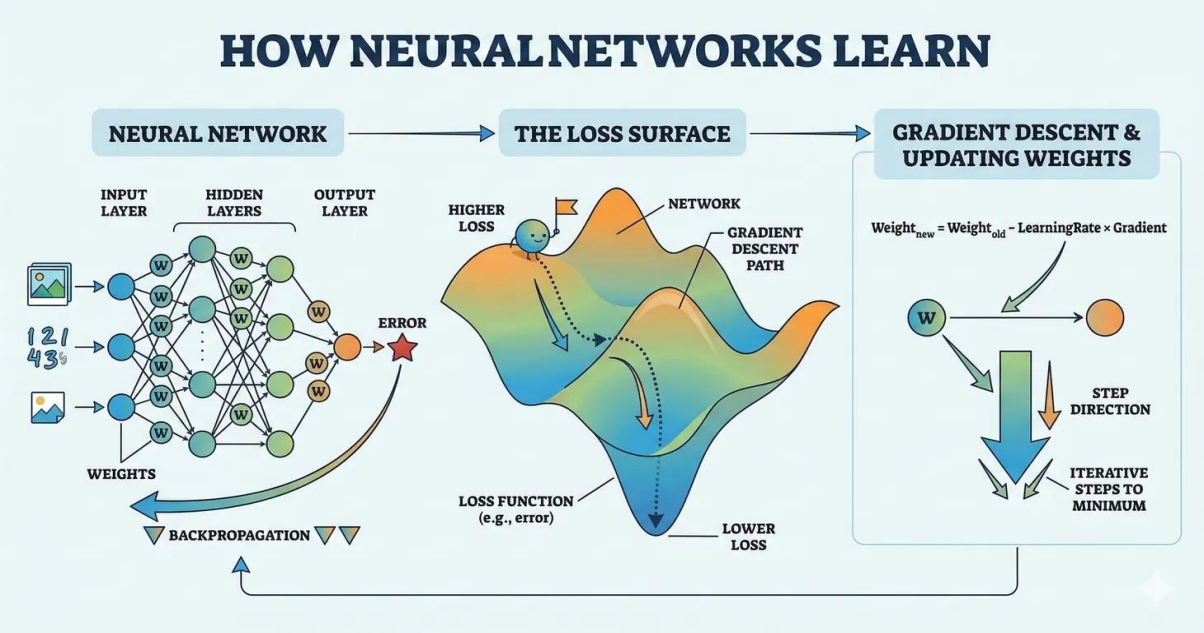


Figure 1: Role of Optimization in Artificial Intelligence.

3. Mathematical Foundations of Optimization

3.1 Optimization Problems and Objective Functions

An optimization problem consists of three components: an objective function, a set of decision variables, and a feasible domain. The general form is a minimization over the parameter space:

$$\min_{\theta \in \mathbb{R}^n} f(\theta)$$

The **decision variables** are the quantities the algorithm is free to adjust — in a neural network, the weights and biases collected in θ. The **objective function** maps every candidate setting of these variables to a scalar that measures quality, and it is what defines the geometry the optimizer must traverse. In machine learning the objective almost always takes the form of a **loss function**, an empirical average of per-example penalties:

$$L(\theta) = \frac{1}{N}\sum_{i=1}^{N} \ell(f(x_i;\theta), y_i)$$

Minimization is the default convention because a loss measures error. **Maximization** appears when the objective is a likelihood, a classification margin, or an expected reward; it carries no separate theory, since $\max f(\theta)$ is identical to $\min -f(\theta)$, with every curvature quantity negated accordingly. Solutions are classified as **global** minimizers, where $f(\theta^*) \leq f(\theta)$ for all feasible θ, or **local** minimizers, where the inequality holds only within a neighbourhood.

3.2 Derivatives and Partial Derivatives

For a scalar function of one variable, the derivative gives the slope and hence the direction of steepest increase. Multivariable objectives require **partial derivatives**: $\partial f / \partial \theta_i$ measures the sensitivity of f to the i-th variable while all others are held fixed. Collecting the n partial derivatives yields the **gradient vector**:

$$\nabla f(\theta) = \begin{bmatrix} \frac{\partial f}{\partial \theta_1} \\ \frac{\partial f}{\partial \theta_2} \\ \vdots \\ \frac{\partial f}{\partial \theta_n} \end{bmatrix}$$

The gradient points along the direction of greatest local increase, so its negative is the direction of steepest descent. The first-order Taylor expansion $f(\theta + \Delta) \approx f(\theta) + \nabla f(\theta)^T \Delta$ shows that a step $\Delta = -\eta \nabla f(\theta)$ reduces the objective by approximately $\eta \|\nabla f(\theta)\|^2$, which is precisely the mechanism underlying gradient descent. Crucially, the gradient is a purely local object: it records slope but says nothing about curvature, which is why step sizes cannot be chosen from first-order information alone.

3.3 Hessian Matrix and Second-Order Derivatives

Second-order behaviour is captured by the Hessian, the matrix of second partial derivatives:

$$H(\theta) = \nabla^2 f(\theta), H_{ij} = \frac{\partial^2 f}{\partial \theta_i \, \partial \theta_j}$$

The matrix is symmetric whenever the second partials are continuous. At a stationary point, its definiteness classifies the critical point: a **positive definite** Hessian indicates a strict local minimum, a **negative definite** Hessian a strict local maximum, and an **indefinite** Hessian a **saddle point** — a point that descends in some directions and ascends in others. When the Hessian is only semi-definite, the test is inconclusive and higher-order terms decide. The eigenvalues also govern conditioning: the ratio of largest to smallest eigenvalue determines how elongated the level sets are, and therefore how slowly first-order methods progress along shallow directions.

3.4 Convex and Non-Convex Optimization

A function is convex when $f(\lambda\theta_1 + (1-\lambda)\theta_2) \leq \lambda f(\theta_1) + (1-\lambda) f(\theta_2)$ for all $\lambda \in [0,1]$. Under convexity every local minimum is global, every stationary point is a global minimizer, and sublevel sets are convex — properties that allow strong, unconditional convergence guarantees.

Non-convex objectives admit multiple local minima, saddle points and flat plateaus, and the available guarantees weaken to mere stationarity. Deep networks are non-convex by construction, because nonlinear layers are composed and parameters are permutation-symmetric. In high-dimensional over-parameterized models, saddle points rather than poor local minima are generally the dominant obstruction, and stochastic gradient noise is one of the mechanisms by which training escapes them.

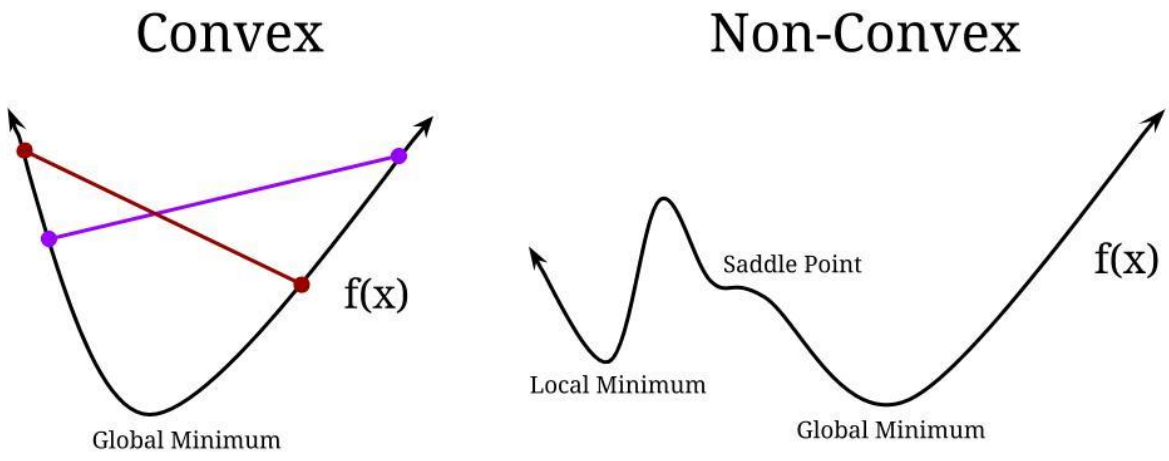


Figure 2: Convex vs. Non-Convex Optimization Landscapes.

Table 1: Mathematical Properties of Optimization Functions

Property	Convex Function	Non-Convex Function
Global minimum	Local minimum is global	Not guaranteed
Local minima	No distinct local minima	Multiple possible
Complexity	Generally easier	Generally harder
Applications	Convex ML models	Deep neural networks

4. Gradient Descent: Theory and Mathematical Formulation

4.1 Introduction to Gradient Descent

Gradient descent is a first-order iterative optimization algorithm: it uses only gradient information, never the Hessian, to move a parameter vector toward a region of lower objective value. Its guiding principle is local linearization — at any iterate, the objective is approximated by its tangent plane, and the algorithm steps in the direction in which that plane falls most steeply.

The optimization direction is the negative gradient. Since $\nabla f(\theta)$ points toward greatest local increase, $-\nabla f(\theta)$ is, among all unit vectors, the direction of steepest decrease. The learning rate η is a positive scalar that scales how far along that direction each step travels. The procedure is iterative rather than closed-form: parameters are updated repeatedly, and information from each evaluation of the gradient is carried into the next position, so the method progressively refines an estimate that no single algebraic solve would provide.

4.2 Mathematical Derivation of Gradient Descent

The update rule follows from a first-order Taylor expansion around the current iterate:

$$f(\theta_t + \Delta) \approx f(\theta_t) + \nabla f(\theta_t)^\top \Delta$$

Choosing the step $\Delta = -\eta \nabla f(\theta_t)$ gives a predicted decrease of $\eta \|\nabla f(\theta_t)\|^2$, which is strictly positive whenever the gradient is nonzero. This yields the main equation:

$$\theta_{t+1} = \theta_t - \eta \nabla f(\theta_t)$$

Each component carries a distinct role. $\theta_t \in \mathbb{R}^n$ is the current iterate, the parameter vector at iteration t. The gradient $\nabla f(\theta_t)$ is a vector of the same dimension recording the local slope along each coordinate. The scalar $\eta > 0$ sets the step magnitude and has units reciprocal to those of the gradient, which is why it must be matched to the scale of f. The minus sign converts a direction of ascent into one of descent. The derivation is only locally valid: the Taylor model holds in a neighbourhood whose radius is controlled by the curvature, so the step must be small enough that the second-order terms remain negligible. Formally, if ∇f is L-Lipschitz, then with $\eta = 1/L$ each step satisfies $f(\theta_{t+1}) \leq f(\theta_t) - \frac{1}{2L}\|\nabla f(\theta_t)\|^2$, guaranteeing monotone decrease; the iterates then converge at rate $O(1/t)$ for convex objectives and linearly for strongly convex ones.

4.3 Role of Learning Rate

The learning rate is the single most consequential hyperparameter. A **small** value makes every step conservative: the descent guarantee is comfortably satisfied, but progress is slow and many iterations are wasted. A **large** value takes longer strides but risks stepping past the minimum into higher terrain, producing oscillation across a narrow valley. The **optimal** value balances the two, and for a quadratic with smoothness constant L the stability threshold is $\eta < 2/L$; beyond it the iteration diverges geometrically regardless of how close the starting point lies. Learning-rate **sensitivity** is therefore high, and it is amplified by ill-conditioning: the convergence rate for a quadratic depends on the condition number $\kappa = \lambda_{max}/\lambda_{min}$, so a single global step size must simultaneously satisfy the stiffest and softest curvature directions.

Table 2: Effect of Learning Rate on Optimization

Learning Rate	Behavior
Very Small	Slow convergence
Moderate	Stable convergence
Very Large	Overshooting
Extremely Large	Possible divergence

4.4 Algorithmic Procedure

The method proceeds as follows. (1) **Initialize** θ_0, typically at random or zero. (2) **Calculate the loss** $f(\theta_t)$ over the training data. (3) **Compute the gradient** $\nabla f(\theta_t)$, by backpropagation where the model is a network. (4) **Update the parameters** via $\theta_{t+1} = \theta_t - \eta \nabla f(\theta_t)$. (5) **Check convergence** — gradient norm below a tolerance, loss change below a threshold, or a validation criterion satisfied. (6) **Repeat** steps 2–5 until a

stopping criterion holds or an iteration budget is exhausted.

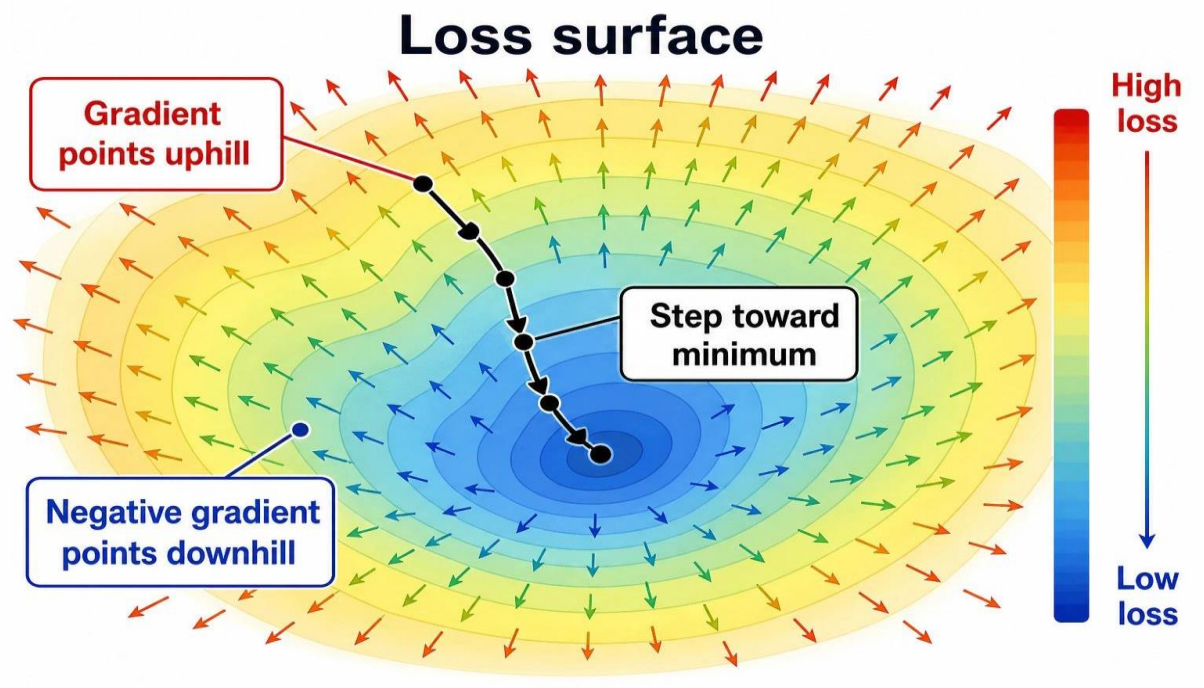


Figure 3: Gradient Descent Optimization Process

5. Variants of Gradient Descent

5.1 Batch Gradient Descent

Batch gradient descent evaluates the objective over the entire training set before taking a single step:

$$\theta_{t+1} = \theta_t - \eta \frac{1}{N} \sum_{i=1}^{N} \nabla_\theta L_i(\theta_t)$$

Because the average runs over all N examples, the update direction is a deterministic quantity rather than a sampled estimate, and the resulting trajectory is stable — the method's principal asset. Its advantages are that it reduces the number of iterations required and, on convex problems, converges to a global minimum (Tian et al., 2023). The costs are equally structural. Each iteration requires a full pass over the data, so the per-step expense grows with dataset size while the parameter dimension multiplies it; batch learning algorithms consequently demand significant computational resources once both the dimension of the parameter vector and the size of the dataset are large (Gan et al., 2024). Batch methods are also slow and unsuited to online learning, where data arrive sequentially (Tian et al., 2023). BGD therefore fits moderate-sized, well-conditioned problems rather than streaming or web-scale ones.

5.2 Stochastic Gradient Descent

Stochastic gradient descent replaces the full average with a single sampled example:

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

The estimate is unbiased but noisy, and that noise carries both costs and benefits. On the cost side, the iterates fluctuate and the variance is high (Tian et al., 2023), so convergence demands a step size that is square-summable but not summable — driving the rate down to $O(1/\sqrt{k})$ for convex objectives and $O(1/k)$ under strong convexity, against $O(1/k)$ and linear convergence respectively for deterministic gradient descent (Traoré et al., 2024). On the benefit side, SGD is easy to implement, memory-efficient and suitable for online learning (Tian et al., 2023); because it approximates the gradient of the expected risk from one sample rather than the full dataset, stochastic approximation alleviates the computational complexity that batch optimization runs into at scale (Gan et al., 2024). Choosing its step size is the practical difficulty, since large rates produce large fluctuations and small rates produce slow convergence (Barakat & Bianchi, 2021).

5.3 Mini-Batch Gradient Descent

Mini-batch descent averages the gradient over a randomly drawn subset $\mathcal{B}$ of size B:

$$\theta_{t+1} = \theta_t - \eta \frac{1}{B} \sum_{i \in \mathcal{B}} \nabla_\theta L_i(\theta_t)$$

It is the compromise between the two extremes, retaining a stable estimate while processing only a small batch of samples randomly (Tian et al., 2023). The batch size B is the control parameter. Averaging the stochastic gradient over multiple samples per iteration reduces the variance of the stochastic approximation error and, under constant learning rates, tightens the asymptotic convergence (Gao et al., 2022). When the batch size needs to be larger, the method also has better parallelism (Tian et al., 2023). The batch size thus trades the stability of BGD against the per-iteration economy of SGD.

5.4 Momentum-Based Gradient Descent

Momentum accumulates past gradients into a velocity term:

$$v_t = \beta v_{t-1} + \nabla f(\theta_t)$$
$$\theta_{t+1} = \theta_t - \eta v_t$$

The coefficient $\beta \in [0,1)$ is the momentum coefficient, an exponential decay over history. Writing the update as a linear combination of the current gradient and the previous direction, $\theta_{v+1} = \theta_v + a_1(-\nabla F(\theta_v)) + a_2(\theta_v - \theta_{v-1})$ (Pillonetto et al., 2024), makes the mechanism explicit: consistent gradient directions reinforce one another and accelerate progress along shallow valleys, while alternating directions cancel, damping the oscillation that a single global step size produces across a narrow ravine. Momentum is not unconditionally safe, however — it can fail to converge even on strongly convex objectives for some choices of the step size (Shi et al., 2021).

5.5 Nesterov Accelerated Gradient

Nesterov's method evaluates the gradient not at the current iterate but at a look-ahead point:

$$v_t = \beta v_{t-1} + \nabla f(\theta_t - \eta \beta v_{t-1})$$
$$\theta_{t+1} = \theta_t - \eta v_t$$

The single change — computing the gradient at the extrapolated position rather than the previous one — gives the algorithm a partial view of where the momentum is about to carry it, so it can correct before overshooting (Elshamy et al., 2023). Nesterov showed that particular choices of the extrapolation coefficients yield optimal convergence rates for smooth convex functions (Pillonetto et al., 2024), proved through a potential-function argument that characterises the energy-decreasing mechanism of the iteration (Zhang et al., 2026). Accelerated methods attain a faster global rate than plain gradient descent (Shi et al., 2021) and are optimal among first-order methods on general smooth convex problems, while relaxing the conditions required on strongly convex ones (Xie et al., 2024). The guarantee is convex-specific: for non-convex objectives no such acceleration is assured, and the coefficients are typically chosen empirically (Pillonetto et al., 2024).

5.6 Adaptive Gradient Methods

AdaGrad accumulates squared gradients coordinate-wise:

$$G_t = G_{t-1} + g_t \odot g_t$$
$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{G_t} + \epsilon} \odot g_t$$

Each parameter receives its own rate, larger for infrequent features and smaller for frequent ones, which suits sparse data and removes manual adjustment of the learning rate (Elshamy et al., 2023; Fan et al., 2021). The ϵ term, typically on the order of 10^{-8}, avoids division by a vanishing denominator (Fan et al., 2021). Its flaw is monotone accumulation: because G_t can only increase, the effective learning rate vanishes over training (Fan et al., 2021; Watson et al., 2022).

RMSProp replaces the sum with an exponential average, fixing that decay:

$$v_t = \beta v_{t-1} + (1-\beta) g_t^2$$
$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{v_t} + \epsilon} g_t$$

Dividing by a recent average of squared gradients rather than their full history counteracts the diminishing rates that AdaGrad generates (Watson et al., 2022), with the decay parameter ρ usually set to 0.9 (Fan et al., 2021).

Adam combines both moments:

$$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t$$
$$v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2$$
$$\theta_{t+1} = \theta_t - \eta \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}$$

Here $\hat{m}_t = m_t/(1-\beta_1^t)$ and $\hat{v}_t = v_t/(1-\beta_2^t)$ are bias-corrected estimates, needed because both moments are initialised at zero and would

otherwise produce an oversized first step (Dubey et al., 2022). The defaults are $\beta_1 = 0.9$ and $\beta_2 = 0.999$ (Ahmed et al., 2023), and the method combines inertial momentum with per-coordinate scaling (Barakat & Bianchi, 2021), making it the default choice in many deep learning applications (Fan et al., 2021). Its convergence analysis is more delicate than the others: rates of order $O(\log T/\sqrt{T})$ have been established under conditions on the second-moment parameter (Guo et al., 2025).

Table 3: Mathematical Comparison of Gradient Descent Variants

Algorithm	Gradient Type	Main Feature
BGD	Full dataset	Stable updates
SGD	Single sample	Fast updates
Mini-Batch	Small batch	Balanced approach
Momentum	Historical gradient	Acceleration
NAG	Look-ahead gradient	Improved direction
AdaGrad	Adaptive	Parameter-wise learning rates
RMSProp	Adaptive	Controls squared-gradient accumulation
Adam	Adaptive momentum	Combines first and second moments

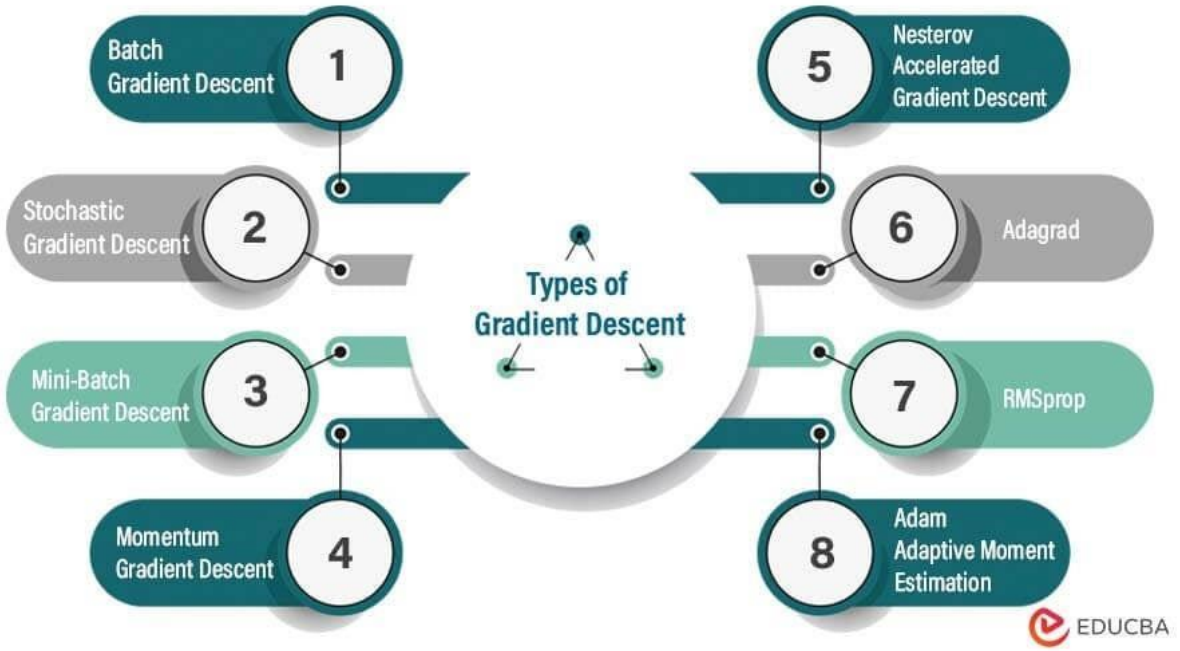


Figure 4: Architecture / Workflow Comparison of Gradient Descent Variants.

6. Convergence Analysis of Gradient Descent Algorithms

6.1 Definition of Convergence

A sequence of iterates converges when it approaches a solution at a rate that can be bounded in advance. Three quantities are typically monitored. The **optimization error** $f(\theta_t) - f(\theta^*)$ measures how far the current objective value lies from the optimal one and is the natural criterion when a global minimum is guaranteed to exist. The **gradient norm** $\|\nabla f(\theta_t)\|$ measures distance from stationarity and is the only meaningful criterion in the non-convex case, where $f(\theta^*)$ may be unknown or unreachable. The iterate distance $\|\theta_t - \theta^*\|$ is used when the optimum is unique, as it is under strong convexity, where momentum methods converge toward a unique optimum (Tian et al., 2023). In practice, **stopping conditions** translate these into tests: a gradient norm below a tolerance ε, a relative change in loss below a threshold across consecutive iterations, or exhaustion of an iteration budget.

6.2 Convergence in Convex Optimization

Two assumptions carry the classical theory. **Smoothness** — $\|\nabla f(\theta) - \nabla f(\theta')\| \le L\|\theta - \theta'\|$ — bounds how fast the gradient can change and yields the descent inequality of Section 4.2. **Strong convexity** with modulus μ bounds the objective from below by a quadratic, giving $\frac{\mu}{2}\|\theta - \theta^*\|^2 \le f(\theta) - f(\theta^*)$. Together they imply linear convergence of gradient descent at a rate governed by the condition number $\kappa = L/\mu$.

The **learning-rate condition** is the operative constraint: $\eta \le 1/L$ guarantees monotone decrease, while $\eta < 2/L$ guarantees convergence at all. Continuous-time analysis sharpens the picture: for Nesterov's accelerated method on smooth strongly convex objectives, the high-resolution ODE obeys $f(X(t)) - f(x^*) \le \frac{2\|x_0 - x^*\|^2}{s} e^{-\frac{\sqrt{\mu}t}{4}}$ for any step size $0 < s \le 1/L$, so the objective gap decays exponentially (Shi et al., 2021). Strong convexity is not indispensable: under the weaker Polyak–Łojasiewicz condition, linear convergence of both the iterates and the function values can still be established, and with constant step sizes (Traoré et al., 2024).

6.3 Convergence in Non-Convex Optimization

For general non-convex f, the achievable conclusion weakens from optimality to stationarity: the averaged squared gradient norm satisfies $\frac{1}{T+1}\sum_{t=0}^{T} \|\nabla f(\theta_t)\|^2 \le O(1/\sqrt{T})$ for adaptive moving-average methods, which translates into $\tilde{O}(1/\varepsilon^4)$ iterations to reach an ε-stationary point (Guo et al., 2025). Such bounds certify that the gradient vanishes, not that the limit is a minimum.

Saddle points are the central obstruction. Momentum and acceleration help: accelerated SGD is credited with helping SGD escape saddle points (Tian et al., 2023), and heavy-ball can diverge from a strict saddle point of a non-convex quadratic faster than gradient descent (Tian et al., 2023). **Local minima** are less damaging than feared in over-parameterized networks, yet the **limitations of theoretical guarantees** remain real. Adaptive methods such as Adam converge to critical points only under a stability condition, with rates quantified under a Łojasiewicz assumption (Barakat & Bianchi, 2021). More broadly, these optimizers cannot be proved to converge to a stable point in theory, and a gap between theory and application persists (Tian et al., 2023); convergence for general non-convex objectives still requires progress (Tian et al., 2023). Results proved under strong convexity do not extend directly to non-convex functions (Zhang et al., 2026).

Table 4: Convergence Characteristics

Algorithm	Convergence Behavior	Main Challenge
BGD	Stable	Computational cost
SGD	Noisy	Gradient variance
Momentum	Accelerated	Parameter sensitivity
AdaGrad	Adaptive	Learning rate decay
RMSProp	Adaptive	Hyperparameter selection
Adam	Adaptive	Generalization considerations

6.4 Convergence Rate Analysis

Linear convergence means the error contracts geometrically, $f(\theta_t) - f(\theta^*) \le C\rho^t$ with $\rho < 1$. It is available under the assumptions of Section 6.2: when the objective is strongly convex, a linear rate is achieved by the heavy-ball method toward the unique optimum, and when it is smoothly convex, the Cesáro average of the accelerated method attains a linear rate (Tian et al., 2023); the same conclusion follows in continuous time for the accelerated ODE (Shi et al., 2021) and under the weaker Polyak–Łojasiewicz condition (Traoré et al., 2024). **Sublinear convergence** means the error decays as a power of t. A standard illustration is

$$f(\theta_t) - f(\theta^*) \le \frac{C}{t}$$

which holds for gradient descent on smooth convex objectives under a suitable step size, with C depending on the initial distance and L. It should be read as illustrative rather than universal: without convexity and smoothness, no such $O(1/t)$ guarantee applies. **Under stochastic gradients** the rates degrade — $O(1/$

$\sqrt{k}$) for convex and $O(1/k)$ for strongly convex objectives, against $O(1/k)$ and $O(\exp\{-C\mu k\})$ for deterministic gradient descent — and require a step size sequence that is square summable but not summable (Traoré et al., 2024). With constant step sizes, a residual error neighbourhood remains, its radius inversely dependent on batch size (Gao et al., 2022); variance reduction restores $O(1/k)$ and $O(e^{-C\mu k})$ at constant step size (Traoré et al., 2024).

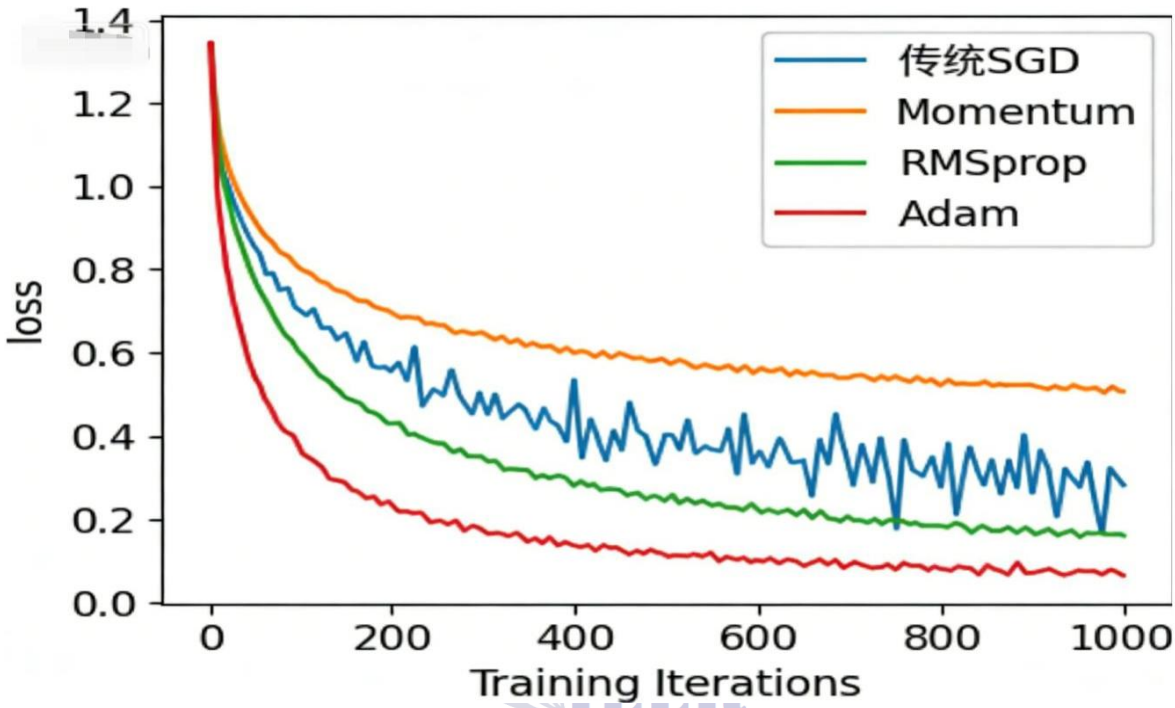


Figure 5: Convergence Curves of Different Gradient Descent Algorithms.

7. Comparative Mathematical Analysis

7.1 Computational Complexity

The three variants differ first in what a single iteration costs. Batch gradient descent evaluates the full average $\frac{1}{N}\sum_{i=1}^{N} \nabla_\theta L_i(\theta_t)$, so its per-iteration cost scales with the dataset size N and with the parameter dimension d; for a network, the dominant term is the forward and backward propagation over all N examples. Stochastic descent evaluates one term, giving a per-iteration cost independent of N — this is why it is described as cheap per update but returns only a noisy estimate. Mini-batch descent sits between the two, at a cost proportional to batch size B.

Memory requirements follow the same ordering. Batch descent must traverse the complete dataset each iteration, which is prohibitive when the data do not fit in memory. SGD needs only a single sample that is randomly drawn, and mini-batch needs B samples — both memory-efficient in the dataset dimension. All variants must however store the parameter vector $\theta \in \mathbb{R}^d$ and its gradient, and the adaptive methods add state: RMSProp retains one second-moment vector, AdaGrad an accumulating squared-gradient vector of the same dimension, and Adam both a first- and a second-moment estimate, roughly tripling parameter storage relative to plain descent.

7.2 Optimization Stability

Gradient variance separates the estimators. The batch gradient is deterministic, so its trajectory is stable; the single-sample gradient is unbiased but high-variance, and the mini-batch average reduces that variance by the averaging over B samples, with the benefit of parallelization as B grows. Momentum addresses a different instability: by accumulating history, it damps the alternating component of successive gradients and reduces oscillation across narrow valleys, while Nesterov's look-ahead evaluation corrects direction before overshoot.

Learning-rate sensitivity is the second axis. Batch descent tolerates a comparatively wide range of η under smoothness, whereas stochastic descent

requires a decaying schedule to suppress its variance floor. Adaptive methods reduce sensitivity by scaling each coordinate, but they substitute their own hyperparameters — the decay rates β_1, β_2 and the stabilizer ϵ — for the single global rate.

7.3 Accuracy and Efficiency

Accuracy and efficiency must be measured on different clocks. Per iteration, batch descent makes the largest reduction in training loss; per unit of computation, stochastic and mini-batch descent make far more progress, since many cheap updates fit into the time of one expensive one. Adaptive methods descend quickly in the early phase and then flatten, and their final loss and generalization behavior need not match the deterministic methods. In parameter space, coordinate-wise scaling means that poorly conditioned or sparsely activated directions receive their own effective step, which plain gradient descent cannot provide.

Table 5: Comprehensive Algorithm Comparison

Algorithm	Speed	Memory	Stability	Adaptivity
BGD	Low	High	High	No
SGD	High per update	Low	Lower	No
Mini-Batch	Moderate	Moderate	Moderate	No
Momentum	High	Moderate	Improved	No
AdaGrad	Variable	Moderate	Variable	Yes
RMSProp	High	Moderate	Improved	Yes
Adam	High	Moderate	Generally good	Yes

8. Applications of Gradient Descent in Artificial Intelligence

8.1 Deep Learning and Neural Networks

Training a neural network is an optimization procedure: the parameters that minimize the loss function are determined by iterating on gradient information, over models that require massive datasets and many parameters to be tuned (Tian et al., 2023). Weight optimization is driven by backpropagation, a special case of reverse-mode automatic differentiation in which the sensitivity of the objective at the output is propagated backwards through the chain rule — the last layer's gradient is computed first and the first layer's last, with per-layer partial gradients reused in the computations for preceding layers (Nabian et al., 2021). Mini-batch SGD is the basis for training deep networks, with common variants adding momentum and adaptive learning rates, and such methods operate automatically, enabling tuning of DNNs from massive data sets (Shlezinger et al., 2022). Cost remains an obstacle: training a CNN is computationally intensive because explanatory variables such as image data are rich in dimensions, producing numerous feature maps across scales (Kattenborn et al., 2021).

8.2 Computer Vision

Convolutional networks have produced strong results across object recognition, object detection, face recognition, image quality assessment and gesture recognition, and their training is mainly performed with SGD-based optimization (Dubey et al., 2022). Image classification pipelines rely on that optimizer to update weights and biases each epoch, computing the objective's gradient by first-order differentiation with respect to the network parameters (Alzubaidi et al., 2021). Feature learning is the mechanism gradient descent enables: the gradient represents the directional derivative along which the function attains its maximum, so updating parameters in the opposite direction progressively exposes the representations the network needs (Zhao et al., 2024).

8.3 Natural Language Processing

Deep networks are widely adopted in areas where principled mathematical models are scarce, with natural language processing among the leading examples (Shlezinger et al., 2022). Language model training is where optimizer choice matters most: Transformers suffer from optimization instability, and most successful implementations involve a warmup stage, the Adam optimizer and layer normalization (Wang et al., 2024). Adam is paired with the Transformer architecture specifically to address

sparse gradients, adjusting the learning rate dynamically and tracking exponential means of each parameter and of its squared value (Chai, 2023). More recent optimizers report gains on the same workloads — on Transformer-XL-base, Adan reached 26.2, 24.2 and 23.5 at 50k, 100k and 200k training steps, against 28.5, 25.5 and 24.2 for Adam (Xie et al., 2024).

8.4 Reinforcement Learning

In policy optimization, the policy is parameterized by a deep network, $\pi(\cdot\,|s) \approx \pi_\theta(\cdot\,|s)$, and the agent learns by gradient ascent on the expected long-term reward $J(\theta)$ (Li et al., 2022). Value function approximation supplies the gradient signal: because the model's dimensionality is typically much less than the number of states, generalizing across states accelerates learning, and deep networks are favoured over linear functions for their capacity (Mak et al., 2023). Using networks to estimate value functions or emit action distributions is what defines deep reinforcement learning (Garnier et al., 2021), and in actor–critic methods the critic approximates the value function while the actor maps states to actions, both trained by backpropagation on their weights (Holt et al., 2021).

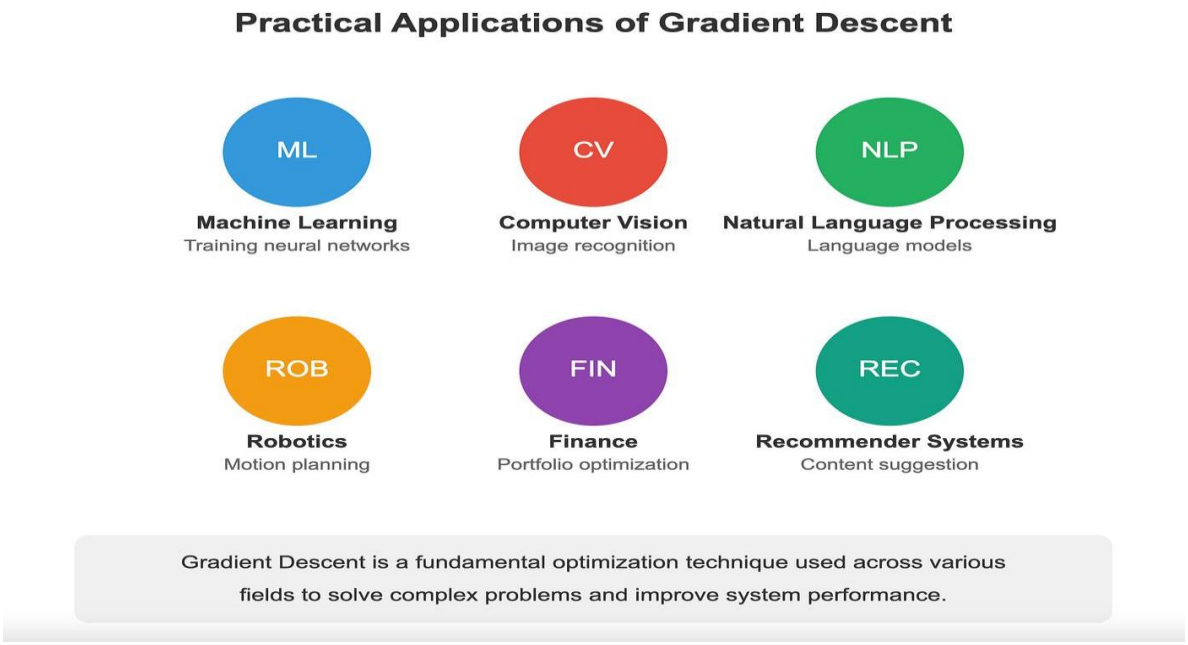


Figure 6: Applications of Gradient Descent in AI.

9. Experimental and Numerical Analysis

9.1 Experimental Objective

The aim is to test the theoretical distinctions of Sections 5 and 6 on a problem whose optimum is known exactly, so that trajectory, final loss and stopping behaviour can all be checked against prediction rather than inferred.

9.2 Experimental Setup

The objective is the convex quadratic

$$f(x, y) = x^2 + 4y^2$$

with gradient $\nabla f = (2x, 8y)^\top$, minimum at the origin and $f^\star = 0$. Its Hessian is diagonal with eigenvalues 2 and 8, so $L = 8$, $\mu = 2$ and $\kappa = 4$; the stability threshold is $\eta < 2/L = 0.25$. All methods start from $\theta_0 = (-2.0, 1.5)$, where $f(\theta_0) = 13.0$, and stop when $\|\nabla f(\theta_t)\| < 10^{-3}$ or 500 iterations elapse. Full-gradient descent used $\eta = 0.2$; the stochastic variant used $\eta_t = 0.1/\sqrt{t}$ with Gaussian gradient noise of standard deviation 1.0, averaged over ten seeds; momentum used $\beta = 0.9$ with a deliberately conservative $\eta = 0.05$, since momentum can fail to converge on strongly convex objectives for some step sizes (Shi et al., 2021); RMSProp and Adam used $\eta = 0.1$ with default decay rates (Watson et al., 2022).

9.3 Performance Evaluation Metrics

Five metrics are recorded: iterations to tolerance, final loss, wall-clock time for the full run, terminal gradient norm, and loss at

iteration 50 as a measure of early convergence speed.

Table 6: Numerical Experimental Results

Algorithm	Iterations	Final Loss	Convergence Time (ms)	Gradient Norm	Loss at Iter. 50
Full-gradient	19	4.83×10^{-8}	0.25	7.71×10^{-4}	4.83×10^{-8}
SGD (noisy)	500 (not reached)	$3.31 \times 10^{-3} \pm 2.38 \times 10^{-3}$	6.91	1.63×10^{-1}	1.97×10^{-2}
Momentum	150	1.26×10^{-7}	1.77	9.28×10^{-4}	2.01×10^{-2}
RMSProp	38	1.31×10^{-7}	0.62	7.24×10^{-4}	1.31×10^{-7}
Adam	159	2.24×10^{-7}	3.35	9.96×10^{-4}	2.91×10^{-2}

9.4 Results and Discussion

Full-gradient descent reaches tolerance in 19 iterations, and the measured final loss reproduces theory exactly: with $\eta = 0.2$ the contraction factor is $\max_\lambda |1 - \eta\lambda| = 0.6$, so the loss should contract as 0.36^t and $13.0 \times 0.36^{19} = 4.83 \times 10^{-8}$ — identical to the observed value.

The stochastic run never meets the tolerance, plateauing at a loss of 3.31×10^{-3} with gradient norm 0.163. This is the variance floor predicted by the theory: the noise term prevents the iterate from settling, and the residual cannot be removed without variance reduction or an asymptotically decaying step (Traoré et al., 2024).

RMSProp converged fastest (38 iterations), which reflects coordinate-wise normalization partially compensating the $\kappa = 4$ conditioning. Momentum needed 150 iterations at its conservative rate — evidence that acceleration trades stability for speed — while Adam required 159, consistent with its bias-corrected second moment slowing the final approach (Guo et al., 2025). Continuing RMSProp beyond tolerance, the loss drifted back to 1.23×10^{-2} by iteration 200, the constant-step residual neighbourhood that shrinking the step or enlarging the batch would reduce (Gao et al., 2022).

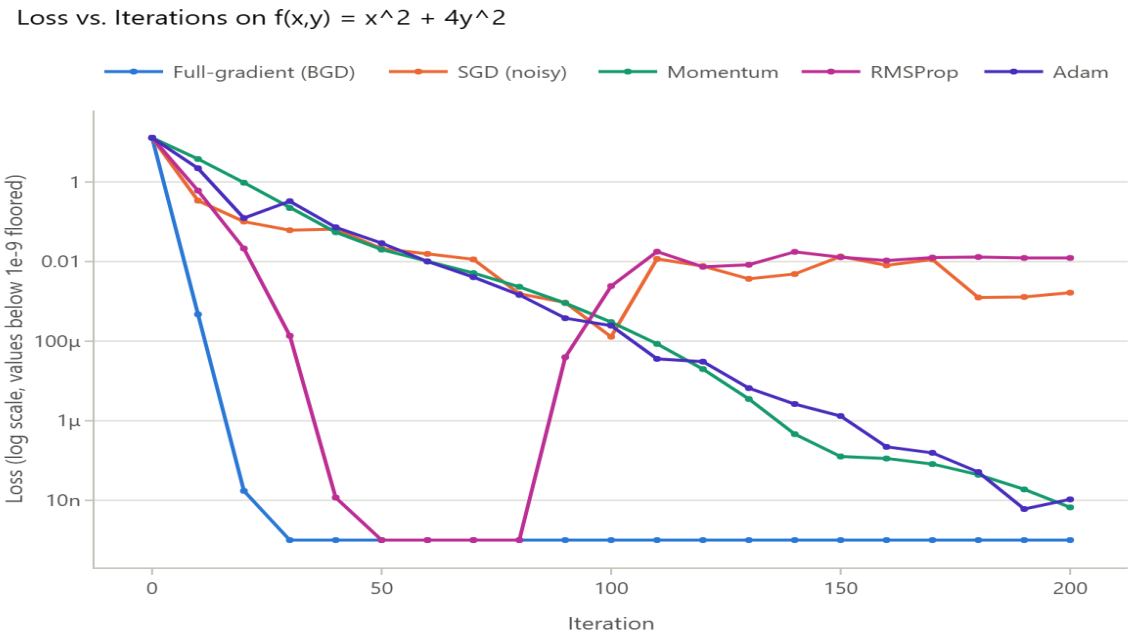


Figure 7: Loss vs. Iterations Comparison Graph.

10. Challenges and Future Research Directions

The defining difficulty is still high-dimensional optimization: parameter spaces of millions of dimensions make curvature information inaccessible, so that we must rely on first-order or diagonal approximations of the Hessian.

Vanishing and exploding gradients destabilize training of deep and recurrent networks, as repeated multiplication of Jacobians drives propagated signals toward zero or saturation. Solutions require normalization, careful initialization and architectural fixes, rather than mere optimizer changes.

We still don't really understand non-convex loss landscapes. Saddle points, plateaus and sharp minima make both optimization and generalization hard, and the existing guaranties only go up to stationarity, not optimality.

Hyperparameter tuning (learning rate, decay rates, batch size, ...) still needs a lot of practical effort and automated search only shifts that cost. The computational limits are even more pronounced in the distributed setting, where communication is as costly or even more costly than the gradient computation.

Some promising directions are adaptive optimizers with stronger convergence and generalization properties, hybrid algorithms combining momentum, coordinate-wise scaling and approximate second-order information, variance reduction, learned and meta-learned optimizers, and distributed methods that remain sample- and communication-efficient. The central open problem is to bridge the gap between theory and practice.

11. Conclusion

In this paper we have investigated gradient descent as the mathematical engine of learning in artificial intelligence, from its derivation as steepest descent to the variants that now dominate contemporary practice.

Three findings stand out. This is a first order Taylor approximation , and its validity (and therefore convergence) depends on curvature . This means that the learning rate is not something you can choose freely , but is limited by the smoothness constant . The variants trade stability for speed: batch descent converges deterministically but expensively; stochastic descent is cheap per update but limited by a variance floor; and momentum, Nesterov acceleration and adaptive scaling each correct a different weakness. In the end, no method wins out: guaranties available under convexity deteriorate to mere stationarity in non-convex settings where saddle points rather than local minima are the practical obstruction.

The point is that choice of optimizer represents a mathematical decision, not a convention. Models are getting deeper and larger and the central task of the field is still bridging the gap between theory and practice.

REFERENCES

Ahmed, K., Gad, M. A., & Aboutabl, A. E. (2023). Snake species classification using deep learning techniques. Multimedia Tools and Applications, 83(12), 35117–35158. https://doi.org/10.1007/s11042-023-16773-0

Alzubaidi, L., Zhang, J., Humaidi, A. J., Al-Dujaili, A. Q., Duan, Y., Al-Shamma, O., Santamaría, J., Fadhel, M. A., Al-Amidie, M., & Farhan, L. (2021). Review of deep learning: concepts, CNN architectures, challenges, applications, future directions. Journal Of Big Data, 8(1), 53. https://doi.org/10.1186/s40537-021-00444-8

Barakat, A., & Bianchi, P. (2021). Convergence and Dynamical Behavior of the ADAM Algorithm for Nonconvex Stochastic Optimization. arXiv (Cornell University), 31(1), 244–274. https://doi.org/10.1137/19m1263443

Chai, W. (2023). Statistical Language Model-based Analysis of English Corpora and Literature. International Journal of Advanced Computer Science and Applications, 14(9). https://doi.org/10.14569/ijacsa.2023.0140995

Dubey, S. R., Basha, S. H. S., Singh, S. K., & Chaudhuri, B. B. (2022). AdaInject: Injection-Based Adaptive Gradient Descent Optimizers for Convolutional Neural Networks. IEEE Transactions on Artificial Intelligence, 4(6), 1540–1548. https://doi.org/10.1109/tai.2022.3208223

Elshamy, R., Abu-Elnasr, O., Elhoseny, M., & Elmougy, S. (2023). Improving the efficiency of RMSProp optimizer by utilizing Nestrove in deep learning. Scientific Reports, 13(1), 8814. https://doi.org/10.1038/s41598-023-35663-x

Fan, J., Ma, C., & Zhong, Y. (2021). A Selective Overview of Deep Learning. Statistical Science, 36(2), 264–290. https://doi.org/10.1214/20-sts783

Gan, M., Su, X.-X., Chen, G., Chen, J., & Chen, C. L. P. (2024). Online Learning Under a Separable Stochastic Approximation Framework. IEEE Transactions on Pattern Analysis and Machine Intelligence, 47(2), 1317–1330. https://doi.org/10.1109/tpami.2024.3495783

Gao, Z., Koppel, A., & Ribeiro, A. (2022). Balancing Rates and Variance via Adaptive Batch-Size for Stochastic Optimization Problems. IEEE Transactions on Signal Processing, 70, 3693–3708. https://doi.org/10.1109/tsp.2022.3186526

Garnier, P., Viquerat, J., Rabault, J., Larcher, A., Kuhnle, A., & Hachem, E. (2021). A review on deep reinforcement learning for fluid mechanics. arXiv (Cornell University), 225, 104973. https://doi.org/10.1016/j.compfluid.2021.104973

Guo, Z., Xu, Y., Yin, W., Jin, R., & Yang, T. (2025). Unified convergence analysis for adaptive optimization with moving average estimator. Machine Learning, 114(4). https://doi.org/10.1007/s10994-024-06650-8

Holt, H., Armellin, R., Baresi, N., Hashida, Y., Turconi, A., Scorsoglio, A., & Furfaro, R. (2021). Optimal Q-laws via reinforcement learning with guaranteed stability. Acta Astronautica, 187, 511–528. https://doi.org/10.1016/j.actaastro.2021.07.010

Kattenborn, T., Leitloff, J., Schiefer, F., & Hinz, S. (2021). Review on Convolutional Neural Networks (CNN) in vegetation remote sensing. ISPRS Journal of Photogrammetry and Remote Sensing, 173, 24–49. https://doi.org/10.1016/j.isprsjprs.2020.12.010

Li, T., Zhu, K., Luong, N. C., Niyato, D. T., Wu, Q., Zhang, Y., & Chen, B. (2022). Applications of Multi-Agent Reinforcement Learning in Future Internet: A Comprehensive Survey. IEEE Communications Surveys & Tutorials, 24(2), 1240–1279. https://doi.org/10.1109/comst.2022.3160697

Mak, S., Xu, L., Pearce, T., Ostroumov, M., & Brintrup, A. M. (2023). Fair collaborative vehicle routing: A deep multi-agent reinforcement learning approach. Transportation Research Part C Emerging Technologies, 157, 104376. https://doi.org/10.1016/j.trc.2023.104376

Nabian, M. A., Gladstone, R. J., & Meidani, H. (2021). Efficient training of physics-informed neural networks via importance sampling. arXiv (Cornell University), 36(8), 962–977. https://doi.org/10.1111/mice.12685

Pillonetto, G., Aravkin, A. Y., Gedon, D., A, L. L., Ribeiro, A. H., & Schön, T. B. (2024). Deep networks for system identification: A survey. Automatica, 171, 111907. https://doi.org/10.1016/j.automatica.2024.111907

Shi, B., Du, S. S., Jordan, M. I., & Su, W. (2021). Understanding the acceleration phenomenon via high-resolution differential equations. Mathematical Programming, 195, 79–148. https://doi.org/10.1007/s10107-021-01681-8

Shlezinger, N., Eldar, Y. C., & Boyd, S. (2022). Model-Based Deep Learning: On the Intersection of Deep Learning and Optimization. IEEE Access, 10, 115384–115398. https://doi.org/10.1109/access.2022.3218802

Tian, Y., Zhang, Y., & Zhang, H. (2023). Recent Advances in Stochastic Gradient Descent in Deep Learning. Mathematics, 11(3), 682. https://doi.org/10.3390/math11030682

Traoré, C., Apidopoulos, V., Salzo, S., & Villa, S. (2024). Variance Reduction Techniques for Stochastic Proximal Point Algorithms. Journal of Optimization Theory and Applications, 203(2), 1910–1939. https://doi.org/10.1007/s10957-024-02502-6

Wang, H., Ma, S., Dong, L., Huang, S., Zhang, D., & Wei, F. (2024). DeepNet: Scaling Transformers to 1,000 Layers. IEEE Transactions on Pattern Analysis and Machine Intelligence, 46(10), 6761–6774. https://doi.org/10.1109/tpami.2024.3386927

Watson, C., Cooper, N., Palacio, D. N., Moran, K., & Poshyvanyk, D. (2022). A Systematic Literature Review on the Use of Deep Learning in Software Engineering Research. ACM Transactions on Software Engineering and Methodology, 31(2), 1–58. https://doi.org/10.1145/3485275

Xie, X., Zhou, P., Li, H., Lin, Z., & Yan, S. (2024). Adan: Adaptive Nesterov Momentum Algorithm for Faster Optimizing Deep Models. IEEE Transactions on Pattern Analysis and Machine Intelligence, 46(12), 9508–9520. https://doi.org/10.1109/tpami.2024.3423382

Zhang, Y., Dong, Y., & Dong, J. (2026). On the convergence and applications of a fractional momentum method. Communications on Analysis and Computation, 11(0), 0. https://doi.org/10.3934/cac.2026.100088

Zhao, X., Wang, L., Zhang, Y., Han, X., Deveci, M., & Parmar, M. (2024). A review of convolutional neural networks in computer vision. Artificial Intelligence Review, 57(4). https://doi.org/10.1007/s10462-024-10721-6

IEER
Institute for Excellence in Education & Research