
aracne

Classificazione Decimale Dewey

— 006.30151 (23.) INTELLIGENZA ARTIFICIALE. Matematica

Thema

— Soggetto: UYQ. Intelligenza artificiale

— Qualificatori: 4CT. Per l'istruzione superiore/terziaria/universitaria

GERARDO IOVANE

MATHEMATICAL METHODS FOR ARTIFICIAL INTELLIGENCE

aracne

ISBN
979-12-218-2687-6

PRIMA EDIZIONE
ROMA 30 APRILE 2026

Table of Contents

Part I – Mathematical foundations for artificial intelligence

1. The mathematical language of artificial intelligence

The development of artificial intelligence systems rests fundamentally upon a precise mathematical framework. While empirical success has driven much of the recent progress in machine learning and deep learning, the underlying mechanisms can be understood only through rigorous formalization. This chapter establishes the essential mathematical vocabulary and structures that will permeate the entire treatment of AI theory presented in this volume.

The mathematical language of AI serves multiple purposes beyond mere notational convenience. It provides a unified framework for reasoning about diverse phenomena including optimization landscapes, statistical generalization, representational capacity, and computational complexity. More importantly, formalization exposes the implicit assumptions embedded in algorithms and models, making explicit the conditions under which theoretical guarantees hold and empirical performance can be expected.

We begin by establishing notational conventions and fundamental mathematical structures. The treatment then progresses through the essential elements of multivariate calculus, culminating in a formal characterization of learning problems. Throughout, we emphasize not merely the definitions themselves but their role in enabling precise statements about learning systems.

1.1. Notation and mathematical structures

The foundation of any rigorous treatment requires careful specification of the mathematical objects under consideration and the notation used to manipulate them. We adopt conventions that balance precision with readability while remaining consistent with the contemporary literature in machine learning theory.

Let $\mathbb{N}$ denote the natural numbers, $\mathbb{Z}$ the integers, $\mathbb{R}$ the real numbers, and $\mathbb{C}$ the complex numbers. We write $\mathbb{R}^+$ for the non-negative reals. When discussing vector spaces, we typically work over $\mathbb{R}$ unless otherwise specified. For a vector space V over $\mathbb{R}$, we denote its dimension by $\dim(V)$, with the understanding that V may be infinite-dimensional in functional analysis contexts.

Vectors are represented by lowercase bold letters in prose and standard lowercase in mathematical expressions. Thus a vector $\boldsymbol{x} \in \mathbb{R}^d$ has components $x_1, \ldots, x_d$. Matrices are denoted by uppercase letters, so $A \in \mathbb{R}^{m \times n}$ has entries A_{ij} for $i = 1, \ldots, m$ and $j = 1, \ldots, n$. The transpose of a matrix A is written A^T, while A^{-1} denotes the inverse when it exists. The Moore-Penrose pseudoinverse is denoted $A^\dagger$.

Tensors, which arise naturally in deep learning contexts, are written as $T \in \mathbb{R}^{n_1 \times n_2 \times \ldots \times n_k}$. Individual entries are accessed through multi-index notation $T_{i_1 i_2 \ldots i_k}$. While general tensor operations can be formalized through the language of multilinear algebra, in practice we often work with specific contractions and reshaping operations that have direct computational implementations.

Norms provide essential tools for measuring magnitudes and distances. The Euclidean norm on $\mathbb{R}^d$ is defined as $\|\boldsymbol{x}\|_2 = (\sum_{i=1}^d x_i^2)^{1/2}$, while the L^1 norm is $\|\boldsymbol{x}\|_1 = \sum_{i=1}^d |x_i|$ and the L^∞ norm is $\|\boldsymbol{x}\|_\infty = \max_i |x_i|$. For matrices, the induced operator norm is $\|A\| = \sup\{\|A\boldsymbol{x}\|_2 : \|\boldsymbol{x}\|_2 \leq 1\}$. These norms generalize to the L^p spaces through the definition $\|f\|_p = (\int |f(x)|^p dx)^{1/p}$ for measurable functions.

The inner product on $\mathbb{R}^d$ is written $\langle \boldsymbol{x}, \boldsymbol{y} \rangle = \sum_{i=1}^d x_i y_i = \boldsymbol{x}^T \boldsymbol{y}$. This induces the Euclidean norm through $\|\boldsymbol{x}\|_2 = \sqrt{\langle \boldsymbol{x}, \boldsymbol{x} \rangle}$ and satisfies the Cauchy-Schwarz inequality $|\langle \boldsymbol{x}, \boldsymbol{y} \rangle| \leq \|\boldsymbol{x}\|_2 \|\boldsymbol{y}\|_2$, with equality if and only if $\boldsymbol{x}$ and $\boldsymbol{y}$ are linearly dependent. The Cauchy-Schwarz inequality has profound implications throughout learning theory,

appearing in analyses of kernel methods, attention mechanisms, and generalization bounds.

Beyond finite-dimensional Euclidean spaces, we encounter various function spaces. The space $C^k(\Omega)$ consists of k-times continuously differentiable functions on a domain $\Omega \subset \mathbb{R}^d$. When Ω is compact, $C^k(\Omega)$ becomes a Banach space under appropriate norms. The Lebesgue spaces $L^p(\Omega)$ contain measurable functions with finite p-norm, forming the natural setting for statistical learning theory where we integrate with respect to probability measures.

Asymptotic notation provides a language for discussing algorithm complexity and approximation quality. We write $f(n) = O(g(n))$ if there exist constants C and n_0 such that $|f(n)| \leq C|g(n)|$ for all $n \geq n_0$. The notation $f(n) = \Omega(g(n))$ indicates that $g(n) = O(f(n))$, while $f(n) = \Theta(g(n))$ means $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$ simultaneously. These asymptotic relationships are essential when analyzing sample complexity, computational cost, and approximation error.

Probability theory introduces additional notation. A probability space is denoted $(\Omega, \mathcal{F}, P)$ where Ω is the sample space, $\mathcal{F}$ a σ-algebra of events, and P a probability measure. Random variables $X: \Omega \to \mathbb{R}$ have probability distributions characterized by cumulative distribution functions or probability density functions $p(x)$. The expectation of X under P is written $\mathbb{E}[X]$ or $\mathbb{E}_P[X]$ when the measure must be specified. Variance is $\mathrm{Var}(X) = \mathbb{E}[(X - \mathbb{E}[X])^2]$. Independence of random variables X and Y is denoted $X \perp\!\!\!\perp Y$, while conditional independence given Z is written $X \perp\!\!\!\perp Y \mid Z$.

The notation $:=$ indicates definition rather than equality. Thus $\mu := \mathbb{E}[X]$ defines μ as the expected value of X. Logical quantifiers $\forall$ (for all) and $\exists$ (there exists) allow concise statement of propositions. The symbol $\Rightarrow$ denotes implication while $\Leftrightarrow$ indicates logical equivalence. Set inclusion is written $\subset$ for strict inclusion and $\subseteq$ for inclusion allowing equality.

1.2. Functions, composition, and transformations

Functions form the primary objects of study in machine learning, serving simultaneously as hypotheses to be learned, transformations to be optimized, and mappings describing data generation processes. A function $f: X \to Y$ maps elements from domain X to codomain Y. When $X \subset \mathbb{R}^d$ and $Y \subset \mathbb{R}^m$, we write $f(\boldsymbol{x}) \in \mathbb{R}^m$ for $\boldsymbol{x} \in \mathbb{R}^d$, with component functions $f_i: \mathbb{R}^d \to \mathbb{R}$ for $i = 1, \ldots, m$.

Function composition lies at the heart of deep learning architectures. Given $f: X \to Y$ and $g: Y \to Z$, their composition is $(g \circ f)(\boldsymbol{x}) = g(f(\boldsymbol{x}))$. Deep neural networks implement compositions of the form $f = f_L \circ f_{L-1} \circ \ldots \circ f_1$ where each layer f_ℓ represents an affine transformation followed by a nonlinear activation. The expressiveness of such compositions derives from the universal approximation properties we shall examine in Chapter 12.

Certain function properties recur throughout learning theory and optimization. A function $f: \mathbb{R}^d \to \mathbb{R}$ is convex if for all $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^d$ and $\lambda \in [0,1]$, we have $f(\lambda \boldsymbol{x} + (1-\lambda)\boldsymbol{y}) \leq \lambda f(\boldsymbol{x}) + (1-\lambda) f(\boldsymbol{y})$. Convexity ensures that local minima are global minima, a property exploited in classical machine learning but largely absent from deep learning. Nonetheless, understanding convex optimization provides essential intuition for more general non-convex problems.

Monotonicity constrains function behavior along orderings. A function $f: \mathbb{R} \to \mathbb{R}$ is monotone increasing if $x \leq y$ implies $f(x) \leq f(y)$. Many activation functions used in neural networks, such as the rectified linear unit (ReLU) $\sigma(z) = \max(0, z)$, are monotone increasing. Monotonicity interacts with optimization and expressiveness in subtle ways, as we shall explore when discussing initialization and training dynamics.

Lipschitz continuity provides a quantitative form of continuity controlling the rate of change. A function $f: X \to Y$ between metric spaces is L-Lipschitz continuous if for all $x, x' \in X$, we have $d_Y(f(x), f(x')) \leq L d_X(x, x')$. The smallest such L is the Lipschitz constant of f. When $X = Y = \mathbb{R}^d$ with Euclidean metric, this becomes $\|f(\boldsymbol{x}) - f(\boldsymbol{y})\|_2 \leq L \|\boldsymbol{x} - \boldsymbol{y}\|_2$.

Lipschitz continuity has profound implications for learning theory. It ensures that small perturbations to inputs produce bounded changes in outputs, a form of stability

essential for generalization. Neural networks with Lipschitz-constrained weights exhibit improved robustness to adversarial perturbations. Moreover, Lipschitz constants appear in generalization bounds through Rademacher complexity and related measures of function class capacity.

A function $f: \mathbb{R}^d \to \mathbb{R}^m$ is differentiable at $\boldsymbol{x}$ if there exists a linear map $Df(\boldsymbol{x}): \mathbb{R}^d \to \mathbb{R}^m$ such that

$$\lim_{\boldsymbol{h}\to 0} \frac{\|f(\boldsymbol{x}+\boldsymbol{h}) - f(\boldsymbol{x}) - Df(\boldsymbol{x})\boldsymbol{h}\|}{\|\boldsymbol{h}\|} = 0.$$

The linear map $Df(\boldsymbol{x})$ is the derivative or differential of f at $\boldsymbol{x}$. When $m = 1$, the derivative is identified with the gradient $\nabla f(\boldsymbol{x}) \in \mathbb{R}^d$ through $Df(\boldsymbol{x})\boldsymbol{h} = \langle \nabla f(\boldsymbol{x}), \boldsymbol{h} \rangle$. For vector-valued functions, the derivative is represented by the Jacobian matrix $J_f(\boldsymbol{x}) \in \mathbb{R}^{m\times d}$ with entries $(J_f)_{ij} = \partial f_i / \partial x_j$.

Differentiability extends naturally to normed vector spaces. A map $f: V \to W$ between normed spaces is Fréchet differentiable at $x \in V$ if there exists a bounded linear operator $Df(x): V \to W$ such that

$$\lim_{\|\boldsymbol{h}\|_V \to 0} \frac{\|f(x+\boldsymbol{h}) - f(x) - Df(x)\boldsymbol{h}\|_W}{\|\boldsymbol{h}\|_V} = 0.$$

This framework encompasses both finite-dimensional calculus and infinite-dimensional functional analysis, providing a unified language for discussing neural networks as functions on finite-dimensional parameter spaces and as elements of infinite-dimensional function classes.

Higher-order derivatives capture curvature information essential for optimization. The Hessian of a twice-differentiable function $f: \mathbb{R}^d \to \mathbb{R}$ is the matrix of second partial derivatives $H_f(\boldsymbol{x}) \in \mathbb{R}^{d\times d}$ with entries $H_{ij} = \partial^2 f / (\partial x_i \, \partial x_j)$. When f is twice continuously differentiable, Schwarz's theorem ensures that the Hessian is symmetric. The Hessian's eigenvalues determine local curvature properties, with positive definiteness characterizing strict local minima and negative definiteness characterizing strict local maxima.

1.3. Multivariate calculus and computational graphs

The calculus of vector-valued functions provides the foundation for understanding gradient-based optimization, the workhorse of modern machine learning. We develop the essential results carefully, as their computational implementation through automatic differentiation powers all contemporary deep learning systems.

Let $f: \mathbb{R}^d \to \mathbb{R}$ be a differentiable scalar function. The gradient $\nabla f(\boldsymbol{x}) \in \mathbb{R}^d$ is the vector of partial derivatives:

$$\nabla f(\boldsymbol{x}) = (\partial f / \partial x_1, \dots, \partial f / \partial x_d)^T.$$

The gradient points in the direction of steepest ascent at $\boldsymbol{x}$, and its negative indicates the direction of steepest descent. This geometric interpretation underlies gradient descent optimization: $\boldsymbol{x}_{t+1} = \boldsymbol{x}_t - \eta_t \nabla f(\boldsymbol{x}_t)$, where $\eta_t > 0$ is the learning rate or step size.

For a vector-valued function $f: \mathbb{R}^d \to \mathbb{R}^m$ with components $f_1, \dots, f_m$, the Jacobian matrix assembles the gradients of the components:

$$J_f(\boldsymbol{x}) = [\nabla f_1(\boldsymbol{x})^T, \dots, \nabla f_m(\boldsymbol{x})^T]^T = [\partial f_i / \partial x_j]_{i,j}$$

The Jacobian captures how perturbations to the input propagate through the function. Indeed, for small $\boldsymbol{h} \in \mathbb{R}^d$, we have the first-order approximation $f(\boldsymbol{x} + \boldsymbol{h}) \approx f(\boldsymbol{x}) + J_f(\boldsymbol{x})\boldsymbol{h}$. This linear approximation forms the basis for understanding how errors propagate through computational graphs.

The chain rule extends differentiation to composed functions, enabling backpropagation through deep networks. Consider functions $f: \mathbb{R}^d \to \mathbb{R}^m$ and $g: \mathbb{R}^m \to \mathbb{R}^n$. Their composition $h = g \circ f$ satisfies

$$J_h(\boldsymbol{x}) = J_g(f(\boldsymbol{x}))J_f(\boldsymbol{x}).$$

In the scalar case where $g: \mathbb{R}^m \to \mathbb{R}$, this becomes

$$\nabla(g \circ f)(\boldsymbol{x}) = J_f(\boldsymbol{x})^T \nabla g(f(\boldsymbol{x})).$$

This formulation reveals the fundamental structure of backpropagation: gradients propagate backward through transposed Jacobians. Given a loss $L = g(f(\boldsymbol{x}))$ and the gradient $\nabla_f L$ with respect to the intermediate representation $f(\boldsymbol{x})$, we compute the

gradient with respect to the input as $\nabla_x L = J_f(\boldsymbol{x})^T \nabla_f L$. This vector-Jacobian product constitutes the basic operation of reverse-mode automatic differentiation.

Theorem 1.1 (Multivariate Chain Rule). Let $f: \mathbb{R}^d \to \mathbb{R}^m$ and $g: \mathbb{R}^m \to \mathbb{R}^n$ be differentiable functions. Then $h := g \circ f: \mathbb{R}^d \to \mathbb{R}^n$ is differentiable with Jacobian

$$J_h(\boldsymbol{x}) = J_g(f(\boldsymbol{x}))J_f(\boldsymbol{x})$$

for all $\boldsymbol{x} \in \mathbb{R}^d$. Equivalently, in component form,

$$\partial h_i / \partial x_j = \sum_{k=1}^{m} (\partial g_i / \partial y_k)|_{y=f(x)} \cdot \partial f_k / \partial x_j$$

for $i = 1, \ldots, n$ and $j = 1, \ldots, d$.

Proof. Fix $\boldsymbol{x} \in \mathbb{R}^d$ and let $\boldsymbol{y} = f(\boldsymbol{x}) \in \mathbb{R}^m$. By differentiability of f at $\boldsymbol{x}$, for any $\varepsilon > 0$ there exists $\delta_1 > 0$ such that for $\|\boldsymbol{h}\| < \delta_1$,

$\|f(\boldsymbol{x} + \boldsymbol{h}) - f(\boldsymbol{x}) - J_f(\boldsymbol{x})\boldsymbol{h}\| \leq \varepsilon\|\boldsymbol{h}\|$.

Similarly, by differentiability of g at $\boldsymbol{y}$, there exists $\delta_2 > 0$ such that for $\|\boldsymbol{k}\| < \delta_2$,

$\|g(\boldsymbol{y} + \boldsymbol{k}) - g(\boldsymbol{y}) - J_g(\boldsymbol{y})\boldsymbol{k}\| \leq \varepsilon\|\boldsymbol{k}\|$.

Set $\boldsymbol{k} = f(\boldsymbol{x} + \boldsymbol{h}) - f(\boldsymbol{x})$. For sufficiently small $\boldsymbol{h}$, we have

$\|\boldsymbol{k}\| \leq \|J_f(\boldsymbol{x})\boldsymbol{h}\| + \varepsilon\|\boldsymbol{h}\| \leq (\|J_f(\boldsymbol{x})\| + \varepsilon)\|\boldsymbol{h}\|$.

Thus for $\|\boldsymbol{h}\|$ sufficiently small, $\|\boldsymbol{k}\| < \delta_2$. We then compute

$$h(\boldsymbol{x} + \boldsymbol{h}) - h(\boldsymbol{x}) = g(f(\boldsymbol{x} + \boldsymbol{h})) - g(f(\boldsymbol{x})) = g(\boldsymbol{y} + \boldsymbol{k}) - g(\boldsymbol{y}) = J_g(\boldsymbol{y})\boldsymbol{k} + r(\boldsymbol{k})$$

where $\|r(\boldsymbol{k})\| \leq \varepsilon\|\boldsymbol{k}\|$. Substituting $\boldsymbol{k} = J_f(\boldsymbol{x})\boldsymbol{h} + s(\boldsymbol{h})$ with $\|s(\boldsymbol{h})\| \leq \varepsilon\|\boldsymbol{h}\|$,

$h(\boldsymbol{x} + \boldsymbol{h}) - h(\boldsymbol{x}) = J_g(\boldsymbol{y})J_f(\boldsymbol{x})\boldsymbol{h} + J_g(\boldsymbol{y})s(\boldsymbol{h}) + r(\boldsymbol{k})$.

The error terms satisfy

$$\|J_g(\boldsymbol{y})s(\boldsymbol{h}) + r(\boldsymbol{k})\| \leq \|J_g(\boldsymbol{y})\|\|s(\boldsymbol{h})\| + \|r(\boldsymbol{k})\| \leq (\|J_g(\boldsymbol{y})\| + 1)\varepsilon\|\boldsymbol{h}\| + \varepsilon\|\boldsymbol{k}\|$$

$\leq (\|J_g(\boldsymbol{y})\| + 1 + \varepsilon(\|J_f(\boldsymbol{x})\| + \varepsilon))\varepsilon\|\boldsymbol{h}\|$.

Since ε was arbitrary, we conclude that h is differentiable at $\boldsymbol{x}$ with derivative $J_h(\boldsymbol{x}) = J_g(f(\boldsymbol{x}))J_f(\boldsymbol{x})$.

The chain rule extends to arbitrary compositions. For $h = f_L \circ f_{L-1} \circ \ldots \circ f_1$, we have

$$J_h(\boldsymbol{x}) = J_{f_L}(z_{L-1})J_{f_{L-1}}(z_{L-2}) \ldots J_{f_1}(\boldsymbol{x})$$

where $z_\ell = (f_\ell \circ \ldots \circ f_1)(\boldsymbol{x})$ represents the intermediate activations. This telescoping product reveals both the power and the challenge of deep learning. Each Jacobian modulates the gradient signal, potentially causing exponential decay (vanishing gradients) or exponential growth (exploding gradients) as gradients propagate through many layers.

Taylor's theorem provides higher-order approximations essential for understanding optimization landscapes. For $f: \mathbb{R}^d \to \mathbb{R}$ twice continuously differentiable, we have

$$f(\boldsymbol{x} + \boldsymbol{h}) = f(\boldsymbol{x}) + \langle \nabla f(\boldsymbol{x}), \boldsymbol{h} \rangle + \frac{1}{2}\langle \boldsymbol{h}, H_f(\boldsymbol{x})\boldsymbol{h} \rangle + o(\|\boldsymbol{h}\|^2)$$

as $\boldsymbol{h} \to 0$. The quadratic term involving the Hessian $H_f(\boldsymbol{x})$ captures local curvature. When $H_f(\boldsymbol{x})$ is positive definite, $\boldsymbol{x}$ is a strict local minimum. The condition number $\kappa(H_f(\boldsymbol{x})) = \lambda_{\max}/\lambda_{\min}$, the ratio of largest to smallest eigenvalue, governs the difficulty of optimization near $\boldsymbol{x}$.

Theorem 1.2 (Second-Order Taylor Expansion). Let $f: \mathbb{R}^d \to \mathbb{R}$ be twice continuously differentiable on an open convex set $U \subset \mathbb{R}^d$. Then for any $\boldsymbol{x}, \boldsymbol{x} + \boldsymbol{h} \in U$,

$$f(\boldsymbol{x} + \boldsymbol{h}) = f(\boldsymbol{x}) + \langle \nabla f(\boldsymbol{x}), \boldsymbol{h} \rangle + \frac{1}{2}\langle \boldsymbol{h}, H_f(\boldsymbol{x} + \theta\boldsymbol{h})\boldsymbol{h} \rangle$$

for some $\theta \in (0,1)$, where H_f denotes the Hessian matrix of f.

This result follows from repeated application of the mean value theorem along the line segment $[\boldsymbol{x}, \boldsymbol{x} + \boldsymbol{h}]$. The first-order approximation $f(\boldsymbol{x} + \boldsymbol{h}) \approx f(\boldsymbol{x}) + \langle \nabla f(\boldsymbol{x}), \boldsymbol{h} \rangle$ underlies gradient-based optimization, while the second-order term reveals how curvature affects convergence rates. Newton's method exploits second-order information through the update $\boldsymbol{x}_{t+1} = \boldsymbol{x}_t - H_f(\boldsymbol{x}_t)^{-1}\nabla f(\boldsymbol{x}_t)$, achieving quadratic convergence near minima but at the cost of computing and inverting the Hessian.

Modern deep learning typically operates with first-order methods, computing only gradients rather than full Hessians. Stochastic gradient descent and its variants (momentum, Adam, RMSprop) use gradient estimates from minibatches to perform scalable optimization. The success of these methods despite ignoring second-order information reflects both the structure of neural network loss landscapes and implicit regularization through architectural choices and training procedures.

Computational graphs formalize the structure of composed functions as directed acyclic graphs where nodes represent operations and edges represent data flow. Each node corresponds to a function $f_i: \mathbb{R}^{n_i} \to \mathbb{R}^{m_i}$, and edges indicate which outputs feed into which inputs. The overall computation $h: \mathbb{R}^d \to \mathbb{R}$ can be expressed as a composition of these primitives.

Automatic differentiation exploits the chain rule to compute gradients efficiently through computational graphs. Forward-mode differentiation computes Jacobian-vector products by propagating perturbations forward through the graph, while reverse-mode (backpropagation) computes vector-Jacobian products by propagating gradient information backward. For functions $\mathbb{R}^d \to \mathbb{R}$, reverse-mode requires only $O(1)$ backward passes regardless of d, making it optimal for the typical machine learning setting where d is large (millions or billions of parameters) but the output is a scalar loss.

The graph structure imposes an evaluation order respecting dependencies. A topological sort provides a valid sequence for forward evaluation. The reverse of this sequence gives a valid order for backpropagation. Modern automatic differentiation frameworks construct these computational graphs either statically (define-then-run) or dynamically (define-by-run), with the latter offering greater flexibility for control flow at some computational overhead.

1.4. The formal structure of learning problems

Having established the mathematical language, we can now formalize what constitutes a learning problem and situate optimization within this framework. This formulation provides the conceptual scaffolding for the entire treatment of machine learning to follow.

A learning problem begins with a data-generating distribution D defined over a space $X \times Y$, where X is the input or feature space and Y the output or label space. In supervised learning, each sample $(\boldsymbol{x}, y) \sim D$ represents an input-output pair. The

distribution D is typically unknown, reflecting our epistemic position: we observe only finite samples from D, not D itself.

Given a hypothesis space H consisting of functions $h: X \to Y$, we seek to find $h^* \in H$ that minimizes the risk or expected loss

$$R(h) := \mathbb{E}_{(x,y)\sim D}[L(h(\boldsymbol{x}), y)]$$

where $L: Y \times Y \to \mathbb{R}^+$ is a loss function quantifying the cost of predicting $h(\boldsymbol{x})$ when the true label is y. Common losses include squared error $L(\hat{y}, y) = (\hat{y} - y)^2$ for regression and cross-entropy $L(\hat{y}, y) = -\sum_k y_k \log(\hat{y}_k)$ for classification with probabilistic predictions.

The minimizer $h^* = \text{argmin}_{h\in H} R(h)$, when it exists, is called the optimal hypothesis or Bayes optimal predictor within H. This represents the best possible performance achievable by the hypothesis class under the given loss. However, $R(h)$ cannot be computed directly without knowing D. This fundamental limitation drives the central strategy of empirical risk minimization.

Observing a training sample $S = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^n$ drawn i.i.d. from D, we define the empirical risk

$$R_{\text{emp}}(h; S) := \frac{1}{n}\sum_{i=1}^{n} L(h(\boldsymbol{x}_i), y_i).$$

This empirical average provides an estimate of the true risk $R(h)$. The law of large numbers ensures that $R_{\text{emp}}(h; S) \to R(h)$ as $n \to \infty$ for any fixed h, provided appropriate regularity conditions hold. Empirical risk minimization seeks

$$\hat{h}_S = \text{argmin}_{h\in H} R_{\text{emp}}(h; S),$$

selecting the hypothesis that performs best on the training data.

The fundamental question of learning theory concerns generalization: does $\hat{h}_S$, which minimizes empirical risk, achieve near-optimal true risk? Formally, we want to bound the generalization gap $R(\hat{h}_S) - R(h^*)$.This gap decomposes as $R(\hat{h}_S) - R(h^*) = [R(\hat{h}_S) - R_{\text{emp}}(\hat{h}_S; S)] + [R_{\text{emp}}(\hat{h}_S; S) - R_{\text{emp}}(h^*; S)] + [R_{\text{emp}}(h^*; S) - R(h^*)]$. The second term is non-positive by definition of $\hat{h}_S$. The third term concentrates

around zero by the law of large numbers. The first term, measuring how empirical and true risk diverge for the selected hypothesis, requires more sophisticated analysis involving the complexity of H.

Capacity measures like VC dimension, Rademacher complexity, and covering numbers quantify how large H is in a sense relevant for generalization. Roughly, more complex hypothesis classes can fit training data better (lower empirical risk) but suffer larger generalization gaps for fixed sample size n. This creates the fundamental trade-off between expressiveness and generalization captured by the bias-variance decomposition and related frameworks.

Optimization algorithms implement empirical risk minimization approximately. Gradient descent and its stochastic variants iterate toward local minima of R_{emp}. When H is parameterized as $H = \{h_\theta : \theta \in \Theta \subset \mathbb{R}^p\}$, the empirical risk becomes a function of parameters:

$$J(\theta) := R_{\text{emp}}(h_\theta; S) = \frac{1}{n}\sum_{i=1}^{n} L(h_\theta(\boldsymbol{x}_i), y_i).$$

Gradient-based optimization seeks to minimize J through updates $\theta_{t+1} = \theta_t - \eta_t \nabla_\theta J(\theta_t)$. In stochastic gradient descent, gradients are estimated from random minibatches rather than the full dataset, providing computational efficiency and implicit regularization benefits.

Deep learning introduces additional structure: the hypothesis h_θ is a composition $h_\theta = f_L \circ \ldots \circ f_1$ of parameterized layers. Each layer f_ℓ has parameters θ_ℓ, and the overall parameter vector θ concatenates all layer parameters. Backpropagation computes $\nabla_\theta J$ efficiently by applying the chain rule layer by layer in reverse topological order. The gradient with respect to layer ℓ parameters depends on the gradient from layer $\ell + 1$, creating a recursive structure that mirrors the forward computation.

Constraints and regularization modify the basic optimization problem. Ridge regression adds a penalty $\lambda\|\theta\|^2$ to the empirical risk, encouraging simpler models through parameter shrinkage. Constrained optimization enforces explicit requirements such as $\|\theta\|_2 \leq B$, maintaining parameters within a bounded set. These techniques

implement Occam's razor: among hypotheses fitting the data equally well, prefer the simplest.

The learning problem admits various generalizations. In unsupervised learning, Y is absent and we seek structure in the distribution of X alone, such as clustering or density estimation. Reinforcement learning replaces the fixed distribution D with an interactive environment, and the goal becomes maximizing cumulative reward through sequential decision-making. Multi-task learning shares representations across related problems, while meta-learning optimizes for rapid adaptation to new tasks.

Despite this diversity, the core structure remains: define a hypothesis space, specify an objective functional over that space, and employ optimization to select a good hypothesis. The mathematical language established in this chapter provides the foundation for making each of these components precise and for analyzing their interactions rigorously.

1.5. Concluding remarks

This chapter has established the mathematical vocabulary and fundamental structures that pervade artificial intelligence research. The notational system introduced here will remain consistent throughout this volume, enabling precise statements about models, algorithms, and theoretical guarantees.

We have emphasized several recurring themes. First, formalization exposes implicit assumptions and enables rigorous reasoning about learning systems. Second, the chain rule and its manifestations in backpropagation form the computational foundation of modern deep learning. Third, the statistical learning framework decomposes the problem of intelligence into manageable mathematical components: distributions, hypothesis classes, loss functions, and optimization algorithms.

The subsequent chapters build upon this foundation, developing the algebraic, probabilistic, and optimization-theoretic machinery required for a comprehensive treatment of AI. Chapter 2 examines linear algebra and numerical methods, providing tools for understanding dimensionality, spectral properties, and iterative algorithms.

Chapter 3 introduces probability and statistics, formalizing uncertainty quantification and inference. Chapter 4 develops optimization theory, analyzing the convergence and complexity of algorithms that power modern machine learning systems.

Throughout, we maintain the standard of rigor established here: precise definitions, complete proofs of fundamental results, and explicit connections between theory and practice. The goal is not merely to present algorithms but to understand why they work, when they fail, and how they can be improved. This mathematical perspective, while demanding, provides the deepest insight into the nature of learning and intelligence.

EXERCISES

1. Prove that the function $f(\boldsymbol{x}) = \|\boldsymbol{x}\|_2^2$ is convex on $\mathbb{R}^d$ by verifying the definition directly.

2. Show that if $f: \mathbb{R}^d \to \mathbb{R}$ is convex and differentiable, then $f(\boldsymbol{y}) \geq f(\boldsymbol{x}) + \langle \nabla f(\boldsymbol{x}), \boldsymbol{y} - \boldsymbol{x} \rangle$ for all $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^d$.

3. Let $f: \mathbb{R}^d \to \mathbb{R}$ be twice continuously differentiable. Prove that if $H_f(\boldsymbol{x})$ is positive definite for all $\boldsymbol{x}$, then f is strictly convex.

4. Consider the composition $h = g \circ f$ where $f: \mathbb{R} \to \mathbb{R}^2$ is defined by $f(t) = (\cos(t), \sin(t))$ and $g: \mathbb{R}^2 \to \mathbb{R}$ by $g(x_1, x_2) = x_1^2 + x_2^2$. Compute $J_h(t)$ using the chain rule and verify by direct computation.

5. Let $\sigma(z) = 1/(1 + e^{-z})$ be the logistic sigmoid. Show that σ is Lipschitz continuous and compute its Lipschitz constant.

6. Prove that for any norm $\|\cdot\|$ on $\mathbb{R}^d$, the function $f(\boldsymbol{x}) = \|\boldsymbol{x}\|$ is convex.

7. Consider the neural network layer $f(\boldsymbol{x}) = \sigma(W\boldsymbol{x} + \boldsymbol{b})$ where σ is applied element-wise. Derive the formula for the Jacobian $J_f(\boldsymbol{x})$ in terms of W and the diagonal matrix $\mathrm{Diag}(\sigma'(W\boldsymbol{x} + \boldsymbol{b}))$.

8. Show that if $f, g: \mathbb{R}^d \to \mathbb{R}$ are convex, then their pointwise maximum $h(\boldsymbol{x}) = \max(f(\boldsymbol{x}), g(\boldsymbol{x}))$ is also convex.

9. Let $f: \mathbb{R}^d \to \mathbb{R}$ be differentiable and L-Lipschitz continuous. Prove that $\|\nabla f(\boldsymbol{x})\|_2 \leq L$ for all $\boldsymbol{x}$.

10. Verify that the empirical risk $R_{\text{emp}}(h; S)$ is an unbiased estimator of the true risk $R(h)$ when samples are drawn i.i.d. from D.