Lecture Notes: Computational Mathematics and AI

A Ten-Lecture Workshop Series

Lars Ruthotto
Departments of Mathematics and Computer Science
Emory University
lruthotto@emory.edu

December 19, 2025

Abstract

These lecture notes accompany a ten-lecture course on computational mathematics and artificial intelligence held as part of the Research at the Interface of Applied Mathematics and Machine Learning conference in Houston in December 2025. The course exposes the bidirectional exchange between these fields: how computational mathematics provides rigorous foundations, precise language, and design principles for AI, and how AI enables new capabilities for tackling previously intractable computational problems. Topics span machine learning fundamentals, optimization theory and practice, regularization techniques, generative modeling, scientific machine learning, high-dimensional PDEs, inverse problems, and AI-assisted mathematical discovery.

Contents

1 Introduction

1.1 The Bidirectional Bridge Between Fields

The bidirectional exchange between computational mathematics and artificial intelligence has strong traditions and has been rapidly accelerating recently. This workshop explores both directions of this exchange and how they create a virtuous cycle of innovation. This relationship, visualized in Figure 1, is built on strong common foundations and continues to strengthen as advances in each field enable breakthroughs in the other.

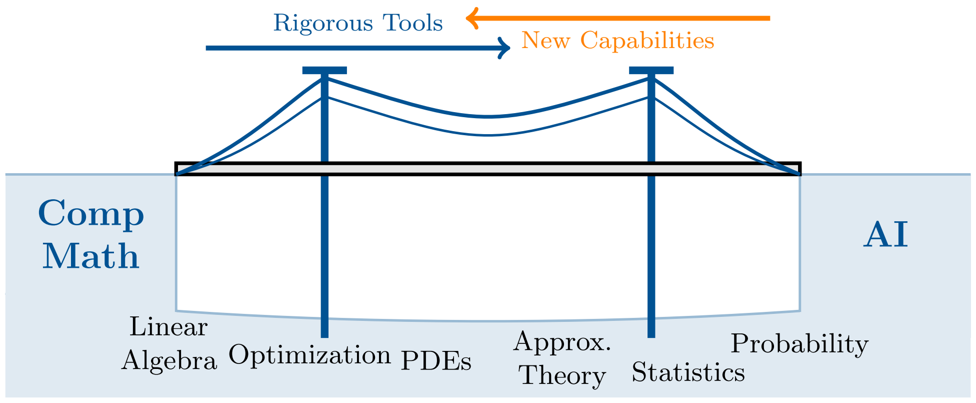

Figure 1: The bidirectional exchange between computational mathematics and artificial intelligence. Mathematical foundations, including linear algebra, optimization theory, partial differential equations, statistics, approximation theory, and probability, provide the structural support for rigorous AI development. Simultaneously, AI provides new computational capabilities and techniques for tackling previously intractable mathematical and scientific problems.

Computational Mathematics → AI Mathematics and statistics supply essential techniques, tools, and theoretical frameworks that make AI possible. They offer a *precise language and rigorous framework* for communicating, describing, analyzing, and understanding AI systems. Optimization theory and dynamical systems provide foundations for understanding training: recent perspectives view neural network training as gradient flows in high-dimensional spaces governed by partial differential equations and optimal transport. Approximation theory guides architecture design choices in depth and width, while statistical foundations enable principled approaches to generalization, uncertainty quantification, and inference. Kernel methods and random matrix theory illuminate how networks learn feature representations and generalize despite overparameterization. Mathematical insights continuously inform the foundation and design of new AI models and paradigms. As a 2025 NSF workshop on AI and Mathematics and Physical Sciences [18] emphasizes, these mathematical tools remain central to unlocking the potential of AI systems. Just as geometry reshaped physics and Maxwell's equations transformed electromagnetic theory, mathematical tools are now reshaping artificial intelligence.

AI $\rightarrow$ Computational Mathematics Artificial intelligence enables us to tackle previously intractable problems across computational mathematics. Neural networks break the curse of dimensionality for high-dimensional partial differential equations and optimal control problems where classical methods become prohibitively expensive. Data-driven approaches provide fast surrogate models and discovered closure terms for large-scale simulations, while neural operators learn solution mappings for parametric families of equations. AI accelerates outer-loop problems including uncertainty quantification, inverse problem solving with stable Bayesian inference, and optimal control. Machine learning enables discovery and optimization of numerical algorithms themselves: from learning iterative methods to discovering novel matrix multiplication schemes and proving mathematical theorems with AI-assisted systems. Examples include AlphaFold2's solution to protein structure prediction, reinforcement learning-based discovery of combinatorial counterexamples, and Lean copilot systems achieving significant automation in proof generation. Computational mathematicians are uniquely positioned to develop next-generation AI tools for scientific discovery while advancing fundamental AI research. Just as quantum mechanics reshaped physics a century ago and computational science transformed mathematics half a century ago, AI is now reshaping computational mathematics.

The Virtuous Cycle This workshop shows how computational mathematicians can not only benefit from AI but also develop tailored and often novel AI techniques that exploit problem structure (symmetries, conservation laws, multi-scale behavior). These domain-informed innovations, in turn, *advance AI itself*, creating feedback loops where insights in one direction strengthen the bridge. This creates a virtuous cycle between analyzing existing mathematical datasets and generating new ones, with computational mathematicians serving as essential connectors who deepen the understanding and capabilities flowing in both directions across the bridge.

1.2 Historical Perspective: A Timeline of Convergence

The fusion of computational mathematics and artificial intelligence has deep historical roots, with key mathematical breakthroughs consistently enabling subsequent AI advances. This convergence reflects a pattern where fundamental theoretical advances in mathematics eventually unlock new possibilities for computational systems.

The **early foundations** were laid through optimization theory and probability. In 1809, Carl Friedrich Gauss developed the method of least squares, establishing optimization-based learning long before modern computers existed. A century later, in 1922, Ronald Fisher introduced maximum likelihood estimation, providing the probabilistic framework that underpins statistical learning. In 1943, Warren McCulloch and Warren Pitts proposed the first mathematical model of a neuron, bridging neurophysiology and computation in a way that would inspire generations of researchers.

The **computational era** (1950s through 1980s) saw parallel developments: numerical methods for partial differential equations (finite differences, finite elements) alongside the emergence of scientific computing as a distinct discipline. This infrastructure (the numerical analysis, convergence theory, and computational methods) proved essential for AI development. A milestone came in 1986 when the backpropagation algorithm was popularized, finally making it computationally tractable to train multi-layer neural networks via efficient gradient computation.

The **1990s brought rigorous statistical learning theory**, which formalized what makes learning possible. Concepts like VC dimension and PAC (Probably Approximately Correct) learning provided theoretical foundations. Support vector machines elegantly bridged statistics and convex optimization, demonstrating that machine learning could be a mathematically principled discipline.

The **deep learning revolution began in 2012** with AlexNet's remarkable performance on image recognition, sparking renewed interest in neural networks. The success was driven by sheer size of the datasets, advances in parallel computing, and optimization algorithms. Computational mathematicians developed essential tools for understanding training dynamics, convergence properties, and generalization in these overparameterized systems. The introduction of transformer architectures in 2017 leveraged mathematical concepts from kernel methods and attention mechanisms, illustrating how decades of theoretical work enabled new architectural innovations.

The **period from 2017 to present** has witnessed genuine bidirectional flow. Physics-informed neural networks (PINNs) integrate differential equations directly into neural architectures, bridging classical scientific computing and deep learning. Large language models demonstrate emergent capabilities in mathematical reasoning. Critically, AI now contributes to mathematical discovery itself: Lean-based proof assistants achieve 74% automation in theorem proving, reinforcement learning discovers combinatorial counterexamples, and AlphaTensor discovers novel fast matrix multiplication algorithms. A 2024 NSF workshop on AI and the Mathematical and Physical Sciences [17] formalized this understanding, concluding that *"the link between AI and the mathematical and physical sciences is becoming increasingly inextricable."* The workshop called for sustained investment in bidirectional research, interdisciplinary communities, and workforce development.

This timeline illustrates not linear progress but an accelerating spiral: mathematical foundations enable AI advances, which provide new tools and problems for computational mathematics, strengthening the bridge in both directions.

2 Reading List

This section provides essential readings organized by lecture topic. References are extracted from individual lecture outlines and consolidated here for convenient access. Full bibliographic details are in the course bibliography. General foundational references are:

- [19]: Comprehensive modern textbook covering neural networks, optimization, and regularization
- [24]: Bridges applied mathematics and deep learning with computational perspective
- [40]: Two-part series on probabilistic machine learning and graphical models
- [22]: Foundational text on statistical learning theory and methods

2.1 Lecture 1: Machine Learning Overview

- [39]: Foundational machine learning textbook defining learning from experience
- [48]: SVD-based analysis explaining double descent phenomenon
- [58]: No Free Lunch theorems establishing fundamental limits

- [24]: Bridges applied mathematics and deep learning
- [17]: Community white paper positioning course themes

2.2 Lecture 2: Neural Network Architectures and Loss Functions

- [19]: Comprehensive textbook covering feedforward networks, CNNs, RNNs, and optimization
- [6]: Unifying framework showing CNNs, GNNs, and Transformers as geometric deep learning
- [55]: Transformer architecture based on self-attention mechanism
- [23]: Residual networks with skip connections enabling very deep architectures
- [29]: Comprehensive thesis on neural ODEs, SDEs, and CDEs

2.3 Lecture 3: Optimization for Machine Learning

- [5]: Comprehensive SIAM Review survey establishing SA vs SAA framework for large-scale optimization
- [46]: Foundational paper proving convergence of stochastic approximation methods
- [3]: JMLR survey of automatic differentiation covering forward/reverse mode AD
- [47]: Classic Nature paper introducing backpropagation algorithm
- [41]: Standard textbook on numerical optimization and second-order methods

2.4 Lecture 4: Modern Theory of Stochastic Gradient Descent

- [21]: Stability of SGD, early stopping, and implicit regularization
- [28]: Batch size effects on generalization and sharp vs. flat minima
- [11]: Edge of stability phenomenon in neural network training
- [26]: Neural tangent kernel theory and lazy training regime
- [38]: Mean-field view of two-layer networks and distributional dynamics

2.5 Lecture 5: Adaptive Optimization Methods

- [30]: Adam optimizer combining momentum and adaptive learning rates with bias correction
- [35]: AdamW decoupling weight decay from gradient-based updates
- [9]: Lion optimizer discovered via evolutionary program search
- [1]: Natural gradient framework for parameterization-invariant optimization
- [37]: K-FAC using Kronecker-factored Fisher approximations for tractable second-order methods

2.6 Lecture 6: PDE Framework for Generative Modeling

- [4]: Dynamic formulation of optimal transport via continuity equation and kinetic energy minimization
- [34]: Conditional flow matching for efficient generative modeling through supervised learning
- [43]: Optimal transport regularization for continuous normalizing flows
- [50]: Unified framework connecting score-based models to SDEs via Fokker-Planck equation
- [7]: Neural ODEs as the continuous-time foundation for normalizing flows

2.7 Lecture 7: Scientific Machine Learning for PDEs

- [45]: Foundational PINN framework for forward and inverse problems
- [36]: Universal approximation theorem for operators enabling DeepONet architecture
- [33]: Fourier Neural Operator leveraging spectral methods for parametric PDE solving
- [52]: PDEBench benchmarks revealing 2–3 order of magnitude accuracy gaps between ML and classical methods
- [31]: Documented PINN failure modes including spectral bias and gradient pathologies
- [53]: GNN-enhanced preconditioners demonstrating hybrid classical/ML approach

2.8 Lecture 8: High-Dimensional PDEs

- [20]: Deep BSDE method breaking curse of dimensionality for $d = 100$ benchmark
- [44]: Forward-backward stochastic neural networks for high-dimensional parabolic PDEs
- [25]: Hutchinson trace estimation for scaling PINNs to 100,000+ dimensions
- [32]: Neural networks for high-dimensional stochastic optimal control with PMP-informed sampling

2.9 Lecture 9: Machine Learning for Inverse Problems

- [2]: Demonstrates catastrophic instabilities of direct neural networks in medical imaging
- [10]: Diffusion posterior sampling for general inverse problems using score-based priors
- [12]: Comprehensive survey of simulation-based inference methods for likelihood-free problems
- [57]: Conditional optimal transport flows for efficient posterior approximation
- [27]: Denoising diffusion restoration models for linear inverse problems

2.10 Lecture 10: Mathematical Discovery and Verification

- [16]: AlphaTensor discovers 48-multiplication algorithm for 4×4 matrices using RL
- [42]: AlphaEvolve achieves 48 multiplications for complex-valued matrices via LLM-guided code evolution
- [14]: Lean 4 theorem prover with 210,000+ theorems in Mathlib
- [49]: AI-assisted proving achieving 74% proof step automation

3 Outline of the Ten Lectures

This section provides a overviews of each lecture, summarizing the key topics, mathematical concepts, and connections to computational mathematics.

3.1 Module 1: Machine Learning Crash Course

Lecture 1: Machine Learning Overview This lecture establishes machine learning as a framework for data-driven approximation, grounded in the mathematical foundations of optimization, statistics, and approximation theory. Building on centuries of work from Gauss's least squares (1809) to modern statistical learning theory, we introduce machine learning through [39]'s definition: a program learns from experience E with respect to task T and performance measure P if its performance at T improves with E. We explore the central tension between fitting training data and generalizing to unseen examples, introducing the classical bias-variance tradeoff and the surprising modern phenomenon of double descent. The lecture emphasizes the bidirectional exchange between computational mathematics and artificial intelligence, showing how mathematical tools enable rigorous understanding of machine learning while AI techniques solve challenging computational problems.

The lecture opens with a motivating discussion of the bidirectional relationship between computational mathematics and artificial intelligence, framed by the 2024 Nobel Prizes in Physics (Hopfield and Hinton for foundational neural network methods) and Chemistry (Jumper and Hassabis for AlphaFold), which demonstrate how decades of basic research in mathematics and physical sciences underpin modern AI breakthroughs. A recent workshop report "[17]" identifies mathematics and statistics as the foundational backbone of AI, highlighting four pillars: optimization theory (gradient descent, stochastic optimization, non-convex landscape analysis), statistical learning (generalization theory, PAC-Bayes, concentration inequalities), approximation theory (universal approximation, function spaces, kernel methods), and linear algebra (matrix factorizations, randomized algorithms, high-dimensional geometry). The report also identifies computational mathematics opportunities at the AI frontier, and we covered a few examples: numerical optimization for distributed and non-convex problems, numerical linear algebra with randomized algorithms and preconditioning, numerical PDEs through physics-informed neural networks and neural operators, high-dimensional problems combining Monte Carlo with deep learning, and uncertainty quantification via Bayesian methods and error analysis. This bidirectional perspectiv, where computational mathematics provides foundational tools for AI while AI enables new approaches to computational challenges, motivates the course structure across three modules: ML fundamentals (Lectures 1–3), computational mathematics for AI (Lectures 4–6), and AI for computational mathematics (Lectures 7–10).

Machine learning is fundamentally a science of approximation, where we seek functions f_θ that approximate relationships in data while balancing approximation quality against generalization to unseen examples. The [58] No Free Lunch theorem proves that without domain knowledge or prior assumptions, no learning algorithm generalizes better than any other when averaged over all possible problems, making domain expertise (and AI-literate domain experts) essential for success. Using linear least squares as an example, we illustrate the differences of minimizing the empirical risk $\frac{1}{N}\|\mathbf{X}\theta - \mathbf{y}\|_2^2$ versus the expected risk $\mathbb{E}_{(x,y)}[(F_\theta(\mathbf{x}) - \mathbf{y})^2]$. While both rely on the same normal equations, the numerical techniques and thus results differ.

We survey five major machine learning tasks that we will see in the course. Supervised learning learns mappings $F_\theta(x) \approx y$ from labeled pairs, employing function approximation and regularization theory (Lectures 2-3). Unsupervised learning discovers structure in unlabeled data using spectral methods and manifold learning. Operator learning extends beyond finite-dimensional function approximation to learn maps $F_\theta : \mathcal{X} \to \mathcal{Y}$ between infinite-dimensional function spaces, leveraging Green's functions and operator theory (Lecture 7). Reinforcement learning finds optimal policies π from states and rewards via optimal control and Hamilton-Jacobi-Bellman equations (Lectures 8 and 10), while generative modeling learns distributions p from samples using optimal transport, stochastic differential equations, and PDEs (Lecture 6).

Classical statistical learning theory characterizes generalization through model capacity measures such as Vapnik-Chervonenkis (VC) dimension, which quantifies the maximum number of points a hypothesis class can shatter. The bias-variance decomposition decomposes test error as $\mathbb{E}[(\hat{f} - y)^2] = \text{Bias}^2 + \text{Variance} + \sigma^2$, where bias measures error from model assumptions (underfitting) and variance measures sensitivity to training data (overfitting). Classical wisdom suggests choosing model capacity to balance these competing forces, visualized as a U-shaped curve where test error is minimized at an intermediate "sweet spot" capacity. For polynomial regression with degree p and n training points, classical theory recommends stopping before the interpolation threshold $p = n - 1$ to avoid overfitting.

Modern deep learning challenges classical intuition through the double descent phenomenon: test error first decreases, then spikes at the interpolation threshold where parameters equal samples ($p = n$), but surprisingly decreases again in the overparameterized regime ($p \gg n$). [48] explain this through the lens of singular value decomposition: at the interpolation threshold, small singular values σ_r amplify noise in poorly-sampled directions, while overparameterization enables gradient descent to find minimum-norm solutions that implicitly regularize. In the overparameterized regime, the minimum-norm interpolating solution $\theta_{\text{over}} = \mathbf{X}^\top(\mathbf{X}\mathbf{X}^\top)^{-1}\mathbf{y}$ exhibits better generalization than the unique interpolating solution at threshold. Regularization, whether explicit via ridge regression $\min_\theta \|\mathbf{X}\theta - \mathbf{y}\|_2^2 + \lambda\|\theta\|_2^2$ or implicit through gradient descent's bias toward low-norm solutions, stabilizes learning by preventing small singular values from dominating.

Machine learning emerges as a rigorous mathematical discipline where computational mathematics provides essential tools for understanding approximation, optimization, and generalization, while modern phenomena like double descent reveal gaps in classical theory that demand new mathematical frameworks. The course explores this bidirectional exchange: Module 2 (Lectures 4-6) investigates how computational mathematics advances AI through optimization theory, loss landscape analysis, and PDE-based generative modeling, while Module 3 (Lectures 7-10) examines how AI solves computational challenges in scientific machine learning, high-dimensional PDEs, inverse problems, and mathematical discovery.

Lecture 2: Neural Network Architectures and Loss Functions Neural network architectures constitute the fundamental building blocks for modern machine learning systems, where design choices in layer connectivity, depth, and loss functions determine what patterns can be learned from data. This lecture bridges computational mathematics and AI by showing how mathematical structures such as convolution (sparse matrices), graph Laplacians, and transformers encode inductive biases that improve generalization. We focus on three central questions: how layer structure creates invariances (width), how composition depth enables feature learning (depth), and how loss functions guide the learning process. The lecture provides the mathematical foundation for formulating the optimization problem $\min_{\boldsymbol{\theta}} \mathbb{E}[\ell(F_{\boldsymbol{\theta}}(\mathbf{x}), \mathbf{y})]$ that will be solved in Lecture 3.

Architecture design begins with choosing how to connect features; that is, selecting the connectivity pattern encoded in the weight matrix $\mathbf{W}$ for each layer $\mathbf{y} = \sigma(\mathbf{W}\mathbf{x} + \mathbf{b})$. The fully-connected baseline uses dense $\mathbf{W}$, connecting every input to every output with no structural assumptions, but real data (images, sequences, graphs) often has inherent structure that can be exploited. Convolutional neural networks (CNNs) exploit this by making $\mathbf{W}$ block-sparse with shared weights: each output channel $\mathbf{y}^j$ depends on local convolutions $C(\boldsymbol{\theta}_{jk})$ applied to input channels, yielding translation equivariance and dramatically fewer parameters than dense layers. Graph neural networks (GNNs) generalize local connectivity to arbitrary graphs through the message-passing layer $\mathbf{Y} = \sigma(\mathbf{M}\,\mathbf{X}\,\mathbf{W})$, where $\mathbf{X} \in \mathbb{R}^{N\times d}$ contains node features, $\mathbf{M} \in \mathbb{R}^{N\times N}$ is a fixed message-passing matrix (e.g., normalized adjacency $\mathbf{M} = \mathbf{D}^{-1}\mathbf{A}$ or graph Laplacian), and $\mathbf{W}$ transforms features. Since the same weights apply to all nodes, GNNs are permutation equivariant. The "missing graph problem" arises when connectivity is unknown (e.g., which words relate to which in a sentence?), motivating attention mechanisms that learn the adjacency structure from data itself. Transformers address this by projecting features into queries, keys, and values ($\mathbf{Q} = \mathbf{X}\mathbf{W}_Q$, $\mathbf{K} = \mathbf{X}\mathbf{W}_K$, $\mathbf{V} = \mathbf{X}\mathbf{W}_V$), then computing data-dependent edge weights via $\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}(\mathbf{Q}\mathbf{K}^T/\sqrt{d_k})\mathbf{V}$. This is essentially a GNN with learned (dense) adjacency plus feed-forward layers. The unifying insight from [6] is that architecture equals connectivity pattern equals encoded invariance: CNNs encode translation equivariance through sparse weight sharing on grids, GNNs encode permutation equivariance through message passing on fixed graphs, and Transformers learn flexible connectivity adapted to the data.

Network depth controls how many transformations are composed to build increasingly abstract feature representations. Multilayer perceptrons (MLPs) compose layers sequentially $\mathbf{h}_\ell = \sigma(\mathbf{W}_\ell \mathbf{h}_{\ell-1} + \mathbf{b}_{\ell-1})$, but training very deep MLPs is difficult due to vanishing or exploding gradients that grow exponentially with depth. Residual networks ([23]) solve this problem with skip connections $\mathbf{h}_{\ell+1} = \mathbf{h}_\ell + F_{\boldsymbol{\theta}_\ell}(\mathbf{h}_\ell)$ that create additive gradient paths, enabling training of 100+ layer networks by ensuring gradients flow directly through identity shortcuts. The ResNet update resembles Euler discretization of an ODE, which motivates neural ordinary differential equations that replace discrete layers with continuous dynamics $d\mathbf{h}/dt = F_{\boldsymbol{\theta}}(\mathbf{h}, t)$, where depth becomes the integration time T rather than layer count L—see [29] for a comprehensive treatment. A PDE perspective reveals that classification can be viewed as a transport equation $\partial_t u + F_{\boldsymbol{\theta}}^\top \nabla u = 0$, where features follow characteristics; adding diffusion $\partial_t u + F_{\boldsymbol{\theta}}^\top \nabla u = \frac{\sigma^2}{2}\Delta u$ yields smoother probabilistic predictions. The Feynman-Kac formula connects this advection-diffusion PDE to an expectation over stochastic trajectories $u(\mathbf{x}, t) = \mathbb{E}[u_0(\mathbf{h}_T)]$ where $d\mathbf{h} = F_{\boldsymbol{\theta}}(\mathbf{h}, t)dt + \sigma d\mathbf{W}$, motivating neural stochastic differential equations that add controlled randomness to model uncertainty and improve robustness.

The loss function $\ell(F_{\boldsymbol{\theta}}(\mathbf{x}), \mathbf{y})$ defines the learning objective through expected risk minimization $\mathcal{L}(\boldsymbol{\theta}) = \mathbb{E}[\ell(F_{\boldsymbol{\theta}}(\mathbf{X}), \mathbf{Y})]$. For regression tasks, mean squared error $\mathcal{L}_{\text{MSE}} = \mathbb{E}[\|\mathbf{Y} - F_{\boldsymbol{\theta}}(\mathbf{X})\|^2]$ is the standard choice, used in autoencoders where the goal is to reconstruct the input $\mathcal{L} = \mathbb{E}[\|\mathbf{X} - D(E(\mathbf{X}))\|^2]$ after passing through a bottleneck encoder-decoder architecture. Classification tasks typically use cross-entropy loss $\mathcal{L}_{\text{CE}} = -\mathbb{E}[\sum_{c=1}^{C} Y_c \log p_c]$ where $p_c = \text{softmax}(F_{\boldsymbol{\theta}}(\mathbf{X}))_c$, as seen in autoregressive language models like GPT that minimize $\mathcal{L}_{\text{GPT}} = -\mathbb{E}[\sum_{i=1}^{T-1} \log p(x_{i+1}|x_{1:i})]$ through next-token prediction. Other paradigms include generative adversarial networks with minimax objectives, variational autoencoders combining reconstruction and KL regularization, score-based models for generative modeling (previewing Lecture 6), and physics-informed neural networks that blend data fitting with PDE constraints (covered in Lecture 7).

The central takeaway is that architecture and loss function choices directly encode our inductive biases about the problem structure, where well-designed formulations lead to better generalization through fewer parameters and stronger invariances. This lecture completes the problem formulation phase: we now understand how to design $F_{\boldsymbol{\theta}}$ (through architecture choices) and $\mathcal{L}$ (through loss functions), which together define the optimization problem $\min_{\boldsymbol{\theta}} \mathbb{E}[\ell(F_{\boldsymbol{\theta}}(\mathbf{x}), \mathbf{y})]$. This sets the foundation for Lecture 3, where we address how to actually solve this optimization problem through automatic differentiation and stochastic gradient descent.

Lecture 3: Optimization for Machine Learning Modern deep learning depends on our ability to train neural networks with millions to billions of parameters, making optimization the enabling technology for contemporary artificial intelligence. This lecture establishes the theoretical foundations and practical guidance of optimization for machine learning by examining four aspects: stochastic approximation versus sample average approximation formulations, efficient gradient computation via backpropagation, the performance of traditional second-order methods, and stochastic gradient descent as the prototype stochastic approximation algorithm.

Throughout this lecture, we develop the Peaks classification problem as a running example. This dataset consists of 1000 balanced samples from a 2D Peaks function divided into 5 classes based on level sets, which we classify using a multilayer perceptron. The visualization-friendly nature of this problem allows us to directly compare optimization methods while illustrating the transition from small networks where traditional methods excel to over-parameterized networks where stochastic gradient descent becomes a natural fit.

Stochastic approximation and sample average approximation represent two complementary perspectives on optimizing over random data, originating from [46] and [54] respectively. The SA formulation optimizes expectations via streaming independent samples using unbiased gradient estimates, making it natural for online learning scenarios, while the SAA formulation replaces expectations with sample averages over fixed datasets and benefits from rich statistical learning theory. Modern machine learning practice blends both perspectives through mini-batch SGD, which operates on SAA-style fixed datasets while using SA-style stochastic gradient approximations, thereby achieving data parallelism on modern accelerators while maintaining the variance reduction benefits of larger batches. The key insight from [5] is that this hybrid approach defines the contemporary machine learning optimization paradigm.

Backpropagation emerged as the enabling technology for deep learning by providing exact gradients in $O(p)$ operations through systematic application of the chain rule via reverse-mode automatic differentiation. [47] demonstrated that backpropagation computes exact gradients with the same asymptotic cost as a single forward pass by propagating derivatives backward through the computational graph, storing intermediate activations during the forward pass to enable efficient chain rule application during the backward pass. Modern automatic differentiation frameworks like PyTorch and JAX, surveyed comprehensively by [3], automate this process through built-in reverse-mode AD capabilities while handling complex control flow and providing optimizations such as gradient checkpointing that trade recomputation for memory by reducing storage requirements from $O(L)$ to $O(\sqrt{L})$ for networks with L layers.

Classical methods like Newton's method and Gauss-Newton achieve fast convergence and curvature adaptation through Hessian information, yet are rarely used in modern deep learning. The Gauss-Newton approximation $\nabla^2 \mathcal{L} \approx J^T H J$ emerges naturally from linearizing network predictions and taking a quadratic Taylor expansion, where H captures the Hessian of the per-sample loss with respect to predictions and depends critically on loss function structure (identity for least squares, non-trivial for softmax cross-entropy). While Gauss-Newton dominates for small networks, it faces three fundamental limitations at scale: $O(p^2)$ memory requirements (2.5 petabytes for ResNet-50's Hessian), $O(p^3)$ computational cost for matrix operations, and tension between SAA-style large batches needed for accurate curvature estimation versus SA-style small batches that preserve implicit regularization benefits of gradient noise. As detailed in [41], these challenges require structured approximations and matrix-free methods using Hessian-vector products.

Stochastic gradient descent serves as the prototype stochastic approximation method, achieving convergence through unbiased gradient estimates while maintaining $O(1)$ per-iteration cost compared to $O(n)$ for full-batch gradient descent. The Robbins-Monro convergence conditions require learning rates satisfying $\sum_{t=1}^{\infty} \eta_t = \infty$ and $\sum_{t=1}^{\infty} \eta_t^2 < \infty$, guaranteeing convergence to stationary points in non-convex settings and achieving $O(1/\sqrt{t})$ convergence rates for convex objectives, though a gap persists between theoretical guarantees (stationary points) and practical success (generalizable solutions). Mini-batch SGD balances variance reduction with computational efficiency through the relationship $\text{Var}[\text{mini-batch gradient}] = \frac{1}{b}\text{Var}[\text{single-sample gradient}]$, where batch size b is typically chosen as powers of 2 to exploit GPU parallelism. The Peaks experiment reveals a striking regime shift: in the **small regime** (width 32, leading to 261 parameters), vanilla SGD with hyperparameter tuning struggles to reach 80% training accuracy while Gauss-Newton achieves 94%. In contrast, the **lazy regime** (width 8192, over 180k parameters), vanilla SGD effortlessly achieves 99% training accuracy and 92% test accuracy. This shows that over-parameterization transforms the optimization landscape from challenging to benign.

This lecture established a fundamental dichotomy in neural network optimization: in the **small regime** where $p < n$, optimization is genuinely difficult and benefits from second-order methods like Gauss-Newton or careful hyperparameter tuning; in the **lazy regime** where $p \gg n$, vanilla SGD succeeds effortlessly because over-parameterization creates a benign loss landscape. The Peaks experiment quantified this contrast. This mirrors Lecture 1's double descent phenomenon and motivates Lecture 4's investigation into why SGD works so remarkably well through implicit regularization mechanisms, continuous-time gradient flow perspectives, and the edge-of-stability dynamics that characterize training in over-parameterized networks.

3.2 Module 2: Computational Mathematics for AI

Lecture 4: Modern Theory of Stochastic Gradient Descent Stochastic gradient descent works remarkably well for training neural networks in the lazy regime, yet classical optimization theory provides no guarantees for finding good solutions in non-convex problems. This lecture addresses the theory-practice gap by examining the mechanisms that make SGD successful: flat versus sharp minima and their connection to generalization, continuous-time dynamics that reveal hidden biases, implicit regularization from finite step sizes via backward error analysis, and benign loss landscape structure created by over-parametrization. Understanding these mechanisms bridges computational mathematics perspectives on optimization with practical deep learning success, revealing that the algorithm itself shapes which solutions are discovered.

The distinction between flat and sharp minima provides geometric intuition for why some solutions generalize better than others. At a local minimum θ^*, the Hessian eigenvalues characterize local curvature: sharp minima have large maximum eigenvalues where loss rises quickly, while flat minima have small eigenvalues where the loss surface is locally smooth. Empirical evidence from [28] suggests that flat minima correlate with better test error, as they are robust to parameter perturbations. However, as [15] demonstrated, Hessian eigenvalues are not reparameterization-invariant, motivating the use of the Fisher information metric to define Information Geometric Sharpness as a principled, reparameterization-invariant measure of curvature.

The continuous-time limit of gradient descent yields the gradient flow ODE $d\theta/dt = -\nabla L(\theta)$, which dissipates energy monotonically via $dL/dt = -\|\nabla L\|^2 \leq 0$. For linear models, gradient flow exhibits two forms of implicit regularization: early stopping at time t is equivalent to Tikhonov regularization with strength $1/t$, and the asymptotic limit selects the minimum-norm solution among all interpolators. Adding stochastic noise transforms gradient flow into Langevin dynamics $d\theta = -\nabla L(\theta)\,dt + \sqrt{\epsilon}\,dW$ with stationary Gibbs distribution $p(\theta) \propto \exp(-2L(\theta)/\epsilon)$, where temperature $\epsilon \propto \eta/b$ governs exploration of the loss landscape. The edge of stability phenomenon, where training operates at $\lambda_{\max}(H) \approx 2/\eta$, reveals that finite step size effects extend beyond what continuous-time analysis captures.

Backward error analysis reveals that discrete gradient descent with finite step size η implicitly optimizes a modified loss $L_{\text{mod}}(\theta) = L(\theta) + (\eta/4)\|\nabla L(\theta)\|^2 + O(\eta^2)$. This gradient magnitude penalty creates an implicit preference for flat minima where gradients are small throughout the basin. Larger learning rates strengthen this implicit regularization effect, explaining why moderate step sizes often generalize better than very small ones. The analysis extends to stochastic gradient descent, where the same modified loss structure combines with noise-driven exploration, creating two complementary mechanisms for preferring flat regions: the gradient penalty from finite steps and the temperature-dependent sampling from stochastic noise.

Modern neural networks with parameters p far exceeding training samples n defy classical learning theory predictions of catastrophic overfitting, instead exhibiting improved generalization through over-parametrization. The Neural Tangent Kernel (NTK) theory of [26] explains the lazy training regime: with appropriate scaling ($1/\sqrt{n}$ for two-layer networks), the Hessian of the network function vanishes as width increases while parameter changes remain bounded, ensuring that linearization around initialization remains valid. In this regime, gradient descent on neural networks becomes equivalent to kernel regression with the fixed NTK Gram matrix $K = JJ^T$, guaranteeing convergence to global minima; numerical experiments demonstrate how the NTK eigenspectrum converges as network width increases. The mean-field perspective of [38] provides a complementary view where features can evolve, with SGD dynamics converging to a Wasserstein gradient flow on the space of weight distributions, offering global convergence guarantees for two-layer networks with feature learning. Numerical experiments comparing NTK and mean-field scaling on classification tasks illustrate these theoretical predictions.

The success of SGD in neural network training emerges from the interplay of benign loss landscapes created by over-parametrization, implicit regularization from finite step sizes via backward error analysis, and noise-driven exploration via Langevin dynamics that together guide optimization toward flat, generalizable solutions. Building on Lecture 3's introduction of SA/SAA frameworks, backpropagation, and the observation that lazy training works surprisingly well, this lecture explains why these phenomena occur. This sets the foundation for Lecture 5, which explores efficient optimization methods including momentum, adaptive gradient techniques like Adam and Lion, and structured second-order approximations like K-FAC that improve upon vanilla SGD's convergence speed and robustness while preserving its implicit regularization benefits.

Lecture 5: Efficient Optimization Methods Stochastic gradient descent (SGD) has proven remarkably effective for training neural networks, but its basic formulation suffers from learning rate sensitivity, poor conditioning, and lack of momentum that slow convergence on ill-conditioned problems. This lecture explores three complementary strategies for accelerating optimization beyond vanilla SGD: momentum methods that accumulate velocity to smooth gradients and accelerate in consistent directions, adaptive gradient methods that automatically tune per-parameter learning rates from gradient history, and structured second-order approximations that exploit network architecture to make curvature information tractable. We analyze the memory-computation tradeoffs inherent to each approach and demonstrate through computational experiments on the peaks classification problem that optimizer choice depends critically on problem constraints, network architecture, and the training regime. The lecture synthesizes computational mathematics principles such as preconditioning, natural gradients, and Kronecker factorization with modern AI practices to provide a principled framework for selecting optimization algorithms. Building on the foundations from Lectures 3 and 4, we continue using the peaks classification problem as a running example.

Standard SGD achieves stationary points with implicit regularization benefits, but exhibits three key limitations that motivate more sophisticated methods: excessive sensitivity to learning rate choice where small rates yield slow convergence and large rates cause divergence, poor handling of ill-conditioning where a single global learning rate cannot effectively optimize all parameter directions when the loss landscape has vastly different curvatures, and lack of memory where each gradient step ignores historical gradient information that could accelerate convergence in consistent directions. Gauss-Newton methods from Lecture 3 address these issues through explicit curvature information but require prohibitive $O(p^2)$ memory, creating a fundamental trade-off between convergence speed and computational tractability that motivates the three approaches explored in this lecture.

Momentum methods accumulate gradient history in a velocity vector to accelerate optimization in consistent directions while damping oscillations in inconsistent ones. The heavy ball method updates parameters via $v_{t+1} = \beta v_t + \nabla L(\theta_t)$ and $\theta_{t+1} = \theta_t - \eta v_{t+1}$, where the momentum coefficient $\beta \in [0, 1)$ (typically 0.9) controls memory, achieving provably optimal $O(\sqrt{\kappa})$ iteration complexity for convex quadratics compared to gradient descent's $O(\kappa)$ dependence on the condition number. Nesterov's accelerated gradient refines this by computing gradients at the look-ahead position $\tilde{\theta}_t = \theta_t + \beta(\theta_t - \theta_{t-1})$ rather than the current position, achieving the theoretically optimal $O(1/t^2)$ convergence rate for smooth convex functions.

Adaptive gradient methods address SGD's global learning rate limitation by maintaining per-parameter learning rates that automatically rescale based on gradient history, trading $O(2p)$ memory for remarkable robustness across hyperparameter choices. Adam ([30]) combines three design principles: exponential moving averages of gradients (first moment $m_t = \beta_1 m_{t-1} + (1-\beta_1)g_t$) for momentum, exponential moving averages of squared gradients (second moment $v_t = \beta_2 v_{t-1} + (1-\beta_2)g_t^2$) for adaptive scaling, and bias correction $\hat{m}_t = m_t/(1 - \beta_1^t)$ to account for zero initialization, with update rule $\theta_t = \theta_{t-1} - \eta\hat{m}_t/(\sqrt{\hat{v}_t} + \epsilon)$ using default hyperparameters that work remarkably well without tuning ($\eta = 10^{-3}$, $\beta_1 = 0.9$, $\beta_2 = 0.999$). AdamW ([35]) decouples weight decay from the loss gradient by applying regularization after adaptive scaling, fixing Adam's interference between adaptive learning rates and regularization strength, and has become the dominant choice for transformer and language model training. Lion ([9]), discovered via evolutionary program search, uses sign-based updates $\theta_t = \theta_{t-1} - \eta \cdot \text{sign}(c_t)$ with only $O(p)$ memory, providing a memory-efficient alternative that achieves competitive performance on large-scale models while maintaining scale invariance and robustness.

Extending Lecture 4's continuous-time analysis to Adam reveals that in its deterministic limit, Adam converges to zeros of the *Adam vector field* rather than zeros of the gradient, explaining why Adam and SGD find different solutions even on the same problem. Furthermore, Adam's adaptive scaling dampens the heavy-tailed noise structure observed in SGD, resulting in longer escape times from sharp minima and potentially explaining the generalization gap between Adam and SGD on vision tasks. Classical preconditioning with the full Hessian or Fisher information matrix would provide optimal convergence but requires infeasible $O(p^2)$ memory. Natural gradient descent ([1]) provides a parameterization-invariant optimization framework using the Fisher metric, which is equivalent to Gauss-Newton for least-squares losses, and so it remains computationally intractable for large networks without approximations. K-FAC ([37]) makes second-order methods tractable by exploiting neural network layer structure through Kronecker-factored approximations, factoring each layer's Fisher block as a Kronecker product of activation and error correlations, which dramatically reduces both memory and computational cost; however, implementation complexity and the overhead of matrix inversions have limited its adoption compared to first-order adaptive methods.

We summarized the optimization trilogy. Lecture 3 introduced the SA versus SAA framework, backpropagation's $O(p)$ gradient computation, and SGD as the prototype scalable optimizer; Lecture 4 explained why SGD works through convergence theory, implicit regularization mechanisms, continuous-time perspectives, and benign landscape structure from over-parametrization; this lecture demonstrates how to improve upon vanilla SGD by trading modest memory overhead ($O(p)$ for momentum, $O(2p)$ for Adam) for faster convergence and greater robustness, or by exploiting problem structure (K-FAC) when computational budgets permit. Four key insights emerge: over-parametrization creates favorable landscapes where first-order methods suffice, optimization algorithms implicitly regularize by selecting which minimum is found, fundamental trade-offs exist between memory, compute, and tuning effort, and significant theory-practice gaps remain regarding finite-width networks and generalization mechanisms.

Lecture 6: PDE Framework for Generative Modeling This lecture unifies modern generative AI methods through the lens of partial differential equations, demonstrating how state-of-the-art models like diffusion and flow matching arise from classical PDEs. Generative modeling reduces to the problem of matching distributions: finding a generator $g_\theta : \mathbb{R}^q \to \mathbb{R}^n$ that transforms a simple latent distribution (e.g., Gaussian noise) into a complex data distribution. While classical approaches like GANs, VAEs, and normalizing flows can be effective, we focus on today's revalend continuous-time models and cast them in a PDE framework to reveal their common structure and links to dynamic optimal transport. The central insight is that computational mathematics principles (method of characteristics, optimal transport, superposition) explain modern AI systems that generate realistic images and video.

Continuous normalizing flows (CNFs) [7] transform the continuity equation $\frac{\partial \rho_t}{\partial t} + \nabla \cdot (\rho_t v_t) = 0$ into a tractable generative model using the method of characteristics. By parameterizing the velocity field $v_t(x,t)$ as a neural network $v_\theta(x,t)$, the PDE perspective yields a system of ordinary differential equations (ODEs) whose particle trajectories $\frac{dx}{dt} = v_\theta(x,t)$ automatically satisfy the transport equation. This approach enables exact likelihood computation via the instantaneous change-of-variables formula $\log p_\theta(x) = \log p_0(x(1)) + \int_0^1 \nabla \cdot v_\theta(x(t),t)dt$, but requires expensive ODE solving and trace computation at every training step. Moreover, maximum likelihood training does not penalize complex, high-energy trajectories, leading to inefficient sampling that requires many function evaluations.

Optimal transport theory provides structure to CNFs by penalizing the kinetic energy of trajectories through the Benamou-Brenier formulation, which minimizes $\int_0^1 \int \frac{1}{2} \|v_t(x)\|^2 \rho_t(x)\, dx\, dt$ subject to the continuity equation. When the velocity field minimizes transport cost, it must be conservative ($v_t = -\nabla \Phi_t$), enabling more efficient computation since $\nabla \cdot v_t = -\Delta \Phi_t$ (a Laplacian rather than a general trace). The OT-Flow method [43] exploits this structure by directly learning the value function $\Phi_\theta(x,t)$, yielding uniqueness and sample paths with smaller kinetic energy, which reduces computational costs in sampling compared to vanilla CNFs. However, the maximum likelihood framework can struggle when data is supported on a lower-dimensional manifold and still requires time integration during training, making it less scalable than modern alternatives for high-dimensional imaging applications.

Flow matching [34] circumvents the computational bottlenecks of CNF and OT-Flow by constructing feasible probability paths through superposition rather than time integration. For each pair of data point $x_0 \sim p_{\mathcal{X}}$ and noise sample $x_1 \sim p_{\mathcal{Z}}$, we construct an explicit conditional path $\psi_t(x_0, x_1) = (1-t)x_0 + tx_1$ with conditional velocity $u_t(x|x_0, x_1) = x_1 - x_0$. By linearity of the continuity equation, the marginal density $\rho_t(x) = \mathbb{E}_{x_0,x_1}[\delta(x - \psi_t(x_0, x_1))]$ satisfies the PDE, and the marginal velocity is the conditional expectation $v_t(x) = \mathbb{E}[u_t(x|x_0, x_1) \mid \psi_t(x_0, x_1) = x]$. This expectation can be computed via regression: minimizing $\mathbb{E}_{t,x_0,x_1}[\|v_\theta(\psi_t(x_0, x_1), t) - u_t(x|x_0, x_1)\|^2]$ yields a supervised learning objective with no ODE solves or trace computations during training, enabling the scalability behind models like Stable Diffusion 3, Sora, and AlphaFold 3.

Score-based diffusion provides a stochastic alternative through the Fokker-Planck equation $\frac{\partial p_t}{\partial t} + \nabla \cdot (p_t v_t) = \frac{g^2(t)}{2}\Delta p_t$, where the diffusion term asymptotically maps data to a Gaussian distribution [50]. The score function $s_t(x) := \nabla_x \log p_t(x)$ transforms the second-order PDE into an effective first-order transport equation with velocity $v_t - \frac{g^2(t)}{2}s_t$, enabling deterministic sampling via the probability flow ODE. Following the flow matching strategy, we construct conditional Gaussian paths $p_t(x|x_0) = \mathcal{N}(x; \alpha_t x_0, \sigma_t^2 I)$ with analytically known conditional scores $s_t(x|x_0) = -\epsilon/\sigma_t$ (where $x = \alpha_t x_0 + \sigma_t \epsilon$). The marginal score is recovered through regression: minimizing $\mathbb{E}_{t,x_0,\epsilon}[\|s_\theta(x_t, t) - s_t(x_t|x_0)\|^2]$ gives a simple supervised learning objective.

The PDE framework reveals that state-of-the-art generative models share a common computational mathematics foundation, differing primarily in their trade-offs between optimality and computational feasibility. Flow matching and score-based diffusion both achieve scalable training by constructing feasible solutions to transport PDEs through superposition and regression, avoiding the optimization over velocity fields required by CNF and OT-Flow. While optimal transport provides theoretical guarantees (unique maps, minimal energy paths), practical success in high-dimensional imaging comes from feasible, efficient algorithms that exploit linearity and conditional construction. Building on the optimization techniques from Lecture 5, this lecture demonstrates how continuous-time perspectives and PDE formulations provide principled foundations for modern generative AI. The concepts introduced here connect forward to Lecture 7, where we examine scientific machine learning and physics-informed neural networks that incorporate domain-specific PDE constraints, and to Lecture 8, where we consider high-dimensional stochastic control problems.

3.3 Module 3: AI for Computational Mathematics

Lecture 7: Scientific Machine Learning for PDEs Developing neural network algorithms to improve numerical techniques for partial differential equations is a major research area of scientific machine learning. The goal of this lecture is to discuss different approaches how ML augments classical solvers. This lecture examines the computational bottlenecks of classical PDE solvers (finite element, finite volume, spectral methods) and evaluates whether ML methods (physics-informed neural networks (PINNs), neural operators, and hybrid approaches) can provide measurable improvements. Unlike generative AI where perceptual quality suffices, most scientific ML applications require high accuracy for safety-critical applications, making rigorous benchmarking essential. Recently, benchmarks such as PDEBench [52] revealed significant accuracy gaps between ML and classical methods, and have helped steer the field into a more rigorous direction that emphasizes discussion of computational costs and comparison to state-of-the-art methods.

Classical PDE solvers represent decades of mathematical engineering, achieving 10^{-6} to 10^{-12} accuracy with convergence guarantees and conservation law preservation through methods like finite element (FEM), finite volume (FV), and spectral discretizations. However, these methods encounter computational bottlenecks in specific scenarios: optimization studies requiring 10,000+ solves (e.g., airfoil optimization), inverse problems, and multi-scale phenomena where hand-crafted closures prove inadequate (e.g., turbulence modeling). ML's opportunity lies not in general replacement but in targeting these specific gaps where classical methods struggle.

The theoretical justification for neural PDE solvers rests on two pillars: the universal approximation theorem [13] showing that neural networks can represent any continuous function $u(x,t)$, and the operator approximation theorem [8] proving that networks can approximate nonlinear operators $G : V \to W$ between function spaces. Automatic differentiation provides the technical innovation enabling these theories, allowing exact computation of PDE residuals like $u_t - \alpha u_{xx}$ through computational graphs without finite difference approximations. However, the critical caveat is existence of approximations does not imply they can be learned efficiently. The required network width may be exponential, and finding good weights remains a non-convex optimization challenge. The fundamental distinction between PINNs (learning one solution $u(x,t)$ via physics-informed optimization) and neural operators (learning the operator G mapping inputs to solutions via supervised learning on thousands of examples) determines their respective computational economics and use cases.

PINNs train neural networks to minimize PDE residuals, boundary conditions, and sparse data simultaneously through composite loss functions $L = \lambda_r L_{\text{PDE}} + \lambda_b L_{\text{BC}} + \lambda_d L_{\text{data}}$, offering mesh-free, dimension-agnostic approaches with seamless data fusion. Rigorous benchmarking by [52] and [56] revealed that for 2-3D forward problems, PINNs are both slower and less accurate than classical methods, with documented failure modes including spectral bias (networks struggle with high frequencies and shocks), gradient pathologies (PDE, BC, and data loss terms operate at vastly different scales), and extreme sensitivity to initialization requiring problem-specific tuning. However, PINNs excel in their niche: inverse problems where sparse measurements combine with known physics to infer unknown PDE parameters, achieving 5-10% parameter error where classical methods require dense measurements, and data assimilation tasks where approximate models fuse naturally with experimental corrections.

Neural operators learn mappings from initial conditions or parameters to PDE solutions through supervised training on thousands of solved instances, enabling millisecond inference after expensive upfront training. [33]'s Fourier Neural Operator and [36]'s DeepONet architecture achieve speedup at inference but with significant training costs and orders of magnitude accuracy loss, creating a fundamental tradeoff between speed and precision. Neural operators excel in four sweet spots where classical methods prove impractical: outer loop problems like parametric optimization and uncertainty quantification that require thousands of PDE solves and real-time control requiring millisecond latency and can tolerate low-accuracy. However, their application is less promising in safety-critical validation requiring 10^{-6} accuracy.

Hybrid methods enhance classical solvers at specific bottlenecks rather than replacing them, preserving convergence guarantees, stability analysis, and conservation laws while achieving measurable speedups. A concrete example is GNN-enhanced preconditioners [53], which learn corrections to incomplete Cholesky factorizations using graph neural networks: starting from a standard IC(0) preconditioner, the GNN adds a learned correction that preserves SPD structure while reducing condition numbers by up to 79% (iteration counts from 95 to 52 on 2D diffusion problems). This approach exemplifies the hybrid philosophy: classical numerical methods provide the foundation and guarantees, while ML components target specific computational bottlenecks where learning can help.

The key insight from this lecture is that while universal approximation theory proves neural PDE solvers are theoretically possible and automatic differentiation makes them technically feasible, rigorous benchmarking determines they are practically viable only for specific niches: inverse problems (PINNs), parametric studies requiring thousands of solves (neural operators), and targeted enhancements (hybrid methods). This lecture sets the foundation for Lecture 8, where we examine high-dimensional PDEs ($d > 6$) where the curse of dimensionality makes classical grid methods infeasible and ML becomes essential rather than optional, and for Lecture 9, which expands on the inverse problem capabilities introduced here through the Bayesian framework and diffusion-based posterior sampling.

Lecture 8: High-Dimensional PDEs High-dimensional partial differential equations arise naturally in optimal control, mean field games, and computational finance, but traditional grid-based numerical methods fail beyond three or four dimensions due to the exponential scaling of discretization: the curse of dimensionality. This lecture focuses on semilinear parabolic PDEs and their concrete instantiation as Hamilton-Jacobi-Bellman (HJB) equations in stochastic optimal control, demonstrating how neural network methods combined with the forward-backward stochastic differential equation (FBSDE) reformulation can break the curse of dimensionality in hundreds of dimensions. However, neural approximation alone is insufficient: after deriving the backward SDE under both random-walk and optimal-control-guided dynamics, we demonstrate that sampling strategy is another critical ingredient, and Pontryagin's Maximum Principle (PMP) provides the key insight for concentrating computational effort where the solution matters most.

Throughout this lecture, we use a $d = 100$ dimensional optimal control benchmark as a running example, contrasting random-walk sampling against PMP-informed sampling. This benchmark, also used by [20], features a target state far from the origin in 100-dimensional space, illustrating precisely why naive Monte Carlo methods fail in high dimensions: random trajectories rarely reach the relevant regions of state space where the value function must be learned accurately.

The lecture begins with a stochastic optimal control problem: minimize expected cost subject to controlled stochastic dynamics $dX_t = 2a_t\,dt + \sqrt{2}\,dW_t$, where a_t is the control, $L(x, a) = \|a\|^2$ is the running cost, and $g(x)$ is the terminal cost. The value function satisfies the Hamilton-Jacobi-Bellman (HJB) equation $\partial_t u + \Delta u - \|\nabla u\|^2 = 0$, a semilinear parabolic PDE arising throughout optimal control, financial mathematics, and reaction-diffusion systems. This HJB equation admits an analytical solution via the Hopf-Cole transform (showing $v = e^{-u}$ solves the heat equation), providing ground truth for validating neural methods in $d = 100$ dimensions where grid-based methods are completely intractable.

Physics-informed neural networks (PINNs) from Lecture 7 can be extended to high dimensions by addressing the computational bottleneck of Hessian computation: the trace term $\mathrm{tr}(\sigma\sigma^\top \nabla^2 u)$ requires $O(d^2)$ operations to compute the full Hessian matrix. The Hutchinson trace estimator provides an elegant solution: for any matrix A and random vector v with $\mathbb{E}[vv^\top] = I$, we have $\mathrm{tr}(A) = \mathbb{E}[v^\top A v]$, which can be computed via Hessian-vector products in $O(1)$ memory using Taylor-mode automatic differentiation. This reduces computational complexity from $O(d^2)$ to $O(1)$, enabling PINNs to scale to $d > 1000$ dimensions, but the fundamental limitation remains: uniform random sampling of collocation points does not beat the curse of dimensionality for problems where the solution structure concentrates in specific regions of state space.

The forward-backward stochastic differential equation (FBSDE) formulation connects the semilinear parabolic PDE to a system of SDEs: a forward SDE generating sample paths and a backward SDE whose solution $Y_t = u(t, X_t)$ and scaled gradient $Z_t = \sigma^\top \nabla u$ evolve along these paths. Applying Itô's lemma to the value function along an uncontrolled forward SDE $dX_t = \sqrt{2}\,dW_t$ reveals that $du = +\|\nabla u\|^2\,dt + Z_t^\top dW_t$—the drift is *positive*, meaning the value increases along random-walk trajectories as they wander into high-cost regions. Deep BSDE [20] learns separate networks Z_k for each time step to match terminal conditions, while FBSNN [44] learns a single network $u_\theta(t,x)$ enforcing BSDE residuals; both avoid spatial grids but share a critical weakness: random-walk sampling fails when the target region lies far from the initial state distribution.

Numerical experiments on the 100D HJB benchmark reveal the critical role of sampling strategy: with centered targets at the origin, both methods (PINNs, FBSNN) achieve less than 1% suboptimality because random sampling naturally covers the relevant region. However, shifting the target to $x_{\text{target}} = (3,\ldots,3)^\top$ (distance $\|x_{\text{target}}\| = 30$ in 100D) causes catastrophic failure: PINNs achieve 175% suboptimality and FBSNN achieves 131% suboptimality despite good loss convergence, because random walks with typical distance $\sqrt{2d} \approx 14$ rarely reach the target region. This diagnostic experiment demonstrates that neural networks alone do not break the curse of dimensionality: sampling is essential.

Pontryagin's Maximum Principle (PMP) provides the key insight for identifying relevant parts of the state space: for the HJB equation, the optimal control is $a^*(t,x) = -\nabla u(t,x)$, so the controlled forward SDE becomes $dX_t = -2\nabla u_\theta(t,X_t)\,dt + \sqrt{2}\,dW_t$. Unlike random walks, this PMP-guided drift uses the current network estimate to steer sample trajectories toward low-cost regions, creating a feedback loop where improved value function estimates produce better sampling, which in turn yields better training data. Applying Itô's lemma to $u(t,X_t)$ along these *controlled* trajectories yields a different backward SDE: $du = -\|\nabla u\|^2\,dt + Z_t^\top dW_t$, where the drift is now *negative* meaning the value decreases along optimal paths as cost is accumulated. This contrast between the two backward SDEs (positive drift for random walks versus negative drift for controlled dynamics) illuminates why random-sampling methods fail: they train the network on trajectories where value increases, while the optimal policy requires learning where value decreases. Neural SOC achieves less than 1% suboptimality on the shifted-target benchmark where random-sampling methods fail catastrophically, demonstrating that three ingredients are essential: FBSDE reformulation, neural network approximation, and PMP-informed sampling.

The shift from spatial discretization to function approximation enables solving high-dimensional PDEs through three essential components: neural networks for universal approximation, FBSDE systems to derive loss functions, and sampling strategies informed by optimal control theory that concentrate computational effort on relevant regions of state space. The lecture concludes with three research directions that extend PMP-informed neural methods to broader problem classes. First, the connection to reinforcement learning: when the dynamics model is unknown or complex, actor-critic methods can learn control policies from observations, but exploiting known objective structure through HJB-informed approaches can dramatically improve sample efficiency compared to pure RL. Second, connections to global optimization: the Moreau envelope of a non-convex function satisfies a Burgers-type HJB equation, and adding viscosity enables Cole-Hopf-based sampling strategies similar to those developed for stochastic control. Third, mean field games extend single-agent optimal control to populations of interacting agents, coupling an HJB equation (backward in time) with a Fokker-Planck equation (forward in time) for the population density; neural parameterizations of both the value function and density enable high-dimensional solutions with applications to crowd dynamics, traffic, finance, and multi-agent reinforcement learning.

Lecture 9: Machine Learning for Inverse Problems Inverse problems require inferring unknown parameters $\mathbf{x} \in \mathbb{R}^n$ from observed data $\mathbf{y} = A(\mathbf{x}) + \epsilon \in \mathbb{R}^m$, where the forward operator A maps parameters to observations. Unlike forward problems, which are typically well-posed and tractable, inverse problems are mathematically challenging as they commonly violate Hadamard's conditions of existence, uniqueness, and stability. This lecture demonstrates why direct neural network approaches fail on inverse problems and how modern AI methods, specifically Bayesian frameworks with diffusion priors and simulation-based inference, transform inverse problem solving from seeking a single unstable solution to characterizing posterior distributions that quantify uncertainty. The central thesis is that uncertainty quantification is not a sign of failure but a mathematical necessity, and generative AI from Lecture 6 becomes the principled tool for posterior sampling.

Direct approaches that train neural networks $\mathbf{x} = G_\theta(\mathbf{y})$ to map observations to parameters inherit the mathematical ill-posedness of the inverse problem. [2] demonstrated catastrophically unstable behavior: networks trained on clean or low-noise data hallucinate nonexistent features (tumors in medical imaging) and fail to detect real structures when test-time noise levels slightly exceed training distributions. Performance degrades exponentially as noise increases from $\sigma = 0.01$ to $\sigma = 0.10$, with reconstruction errors growing by factors of 10-20 while forward problem accuracy remains stable. This is not overfitting or a training failure: it reflects the fundamental mathematical fact that no amount of data or regularization can stabilize an inherently ill-posed inverse mapping.

The Bayesian formulation $\pi(\mathbf{x}|\mathbf{y}) \propto p(\mathbf{y}|\mathbf{x})\pi(\mathbf{x})$ transforms inverse problems from optimization to inference, where the posterior distribution encodes all plausible solutions consistent with observations and prior knowledge. Classical regularization methods like Tikhonov ($\min_{\mathbf{x}}\{\|A(\mathbf{x}) - \mathbf{y}\|^2 + \alpha\|\mathbf{x}\|_2^2\}$) correspond to maximum a posteriori (MAP) estimation but provide only point estimates without uncertainty. Full posterior sampling reveals the solution space's geometry: high-uncertainty regions indicate where data are insufficient, while concentrated posterior mass identifies well-constrained parameters. The key computational challenge is drawing samples $\mathbf{x}^{(i)} \sim \pi(\mathbf{x}|\mathbf{y})$, which requires either evaluating the likelihood $p(\mathbf{y}|\mathbf{x})$ (often intractable) or leveraging learned generative priors.

Many scientific inverse problems involve black-box simulators (e.g., climate models, agent-based systems, complex PDEs) where we can generate observations $\mathbf{y} = A(\mathbf{x}) + \epsilon$ but cannot evaluate the likelihood $p(\mathbf{y}|\mathbf{x})$ analytically. Simulation-based inference (SBI) addresses this likelihood-free setting by learning surrogate posteriors or likelihood functions from simulator runs. [12] surveys methods including sequential neural likelihood estimation, which trains autoregressive flows to approximate $p(\mathbf{y}|\mathbf{x})$ from $(\mathbf{x}, \mathbf{y})$ pairs generated by the simulator, and neural posterior estimation, which directly learns $\pi(\mathbf{x}|\mathbf{y})$ using conditional normalizing flows. [57] introduces COT-Flow, which frames posterior approximation as conditional optimal transport and achieves orders-of-magnitude speedups over MCMC by learning the posterior-to-prior transport map via flow matching, building directly on the OT-Flow framework from Lecture 6.

Diffusion models trained on high-dimensional data (images, scientific fields) provide powerful learned priors $\pi(\mathbf{x})$ that capture natural data distributions without explicit likelihood formulas. For inverse problems, [10] proposes Diffusion Posterior Sampling (DPS), which samples from $\pi(\mathbf{x}|\mathbf{y})$ by modifying the reverse diffusion process to incorporate the measurement likelihood $p(\mathbf{y}|\mathbf{x})$ at each denoising step. The algorithm alternates between score-based denoising (using the learned prior) and data consistency projection (enforcing $A(\mathbf{x}) \approx \mathbf{y}$), enabling exact posterior sampling for linear operators and approximate sampling for nonlinear forward models. [27]'s Denoising Diffusion Restoration Models (DDRM) exploit singular value decomposition for linear problems, achieving state-of-the-art performance on MRI reconstruction, computed tomography, and super-resolution by leveraging pretrained diffusion models as plug-and-play priors without task-specific retraining.

The lecture synthesizes several cross-cutting themes: inverse problems appear ubiquitously across scientific domains (medical imaging, climate science, geophysics), all sharing the fundamental challenge of ill-posedness; uncertainty quantification is information, not failure, revealing which parameters are constrained by data versus prior assumptions; and the generative models from Lecture 6 (flows and diffusion) directly enable principled Bayesian inference here in Lecture 9. The mathematical instability that dooms direct neural network approaches becomes manageable through posterior sampling, which separates epistemic uncertainty (reducible by collecting more data) from aleatoric uncertainty (irreducible measurement noise). This bidirectional connection—computational mathematics providing the theoretical foundation for understanding inverse problem structure, and AI supplying scalable posterior samplers—exemplifies the course's central theme.

Machine learning transforms inverse problem solving by shifting focus from unstable point estimates to full posterior distributions that faithfully represent solution uncertainty. Direct neural network approaches fail because they inherit mathematical ill-posedness, while Bayesian methods with AI-powered priors (diffusion models, normalizing flows) provide stable, principled inference frameworks.

Lecture 10: Mathematical Discovery and Verification This lecture explores how artificial intelligence has evolved from solving mathematical problems to discovering and proving new mathematics itself, establishing AI as a creative mathematical tool rather than merely a computational one. The lecture presents a unified framework bridging two complementary paradigms: discovery systems that search vast algorithmic spaces to propose novel solutions, and theorem proving systems that provide formal correctness guarantees through proof assistants. By examining how AI broke a 50-year-old record in matrix multiplication and how large language models assist formal theorem proving, we demonstrate that computational mathematics and AI are fully intertwined at the research frontier. This convergence exemplifies the bidirectional CompMath $\leftrightarrow$ AI relationship that has structured the entire course, showing how mathematical structure guides AI systems while AI advances computational mathematics.

To illustrate the discovery part of this lecture, we consider the 4×4 matrix multiplication problem as a running example. This seemingly simple question—finding the minimum number of scalar multiplications needed to compute a 4×4 matrix product—remained unsolved for over 50 years until AlphaTensor (2022) discovered algorithms with 48 multiplications (standard arithmetic) and 47 multiplications (modular arithmetic), while AlphaEvolve (2024) achieved 48 multiplications for complex-valued matrices, all beating the previous best of 49. The case study illustrates key themes including discrete optimization in combinatorial spaces, tensor decomposition as mathematical structure, and the discovery-verification gap.

Algorithm discovery is formulated as discrete optimization: find $a^* = \arg\min_{a\in\mathcal{A}} f(a)$ where $\mathcal{A}$ is the space of all valid algorithms and f measures cost. The fundamental challenge is that $\mathcal{A}$ is discrete, combinatorial, and astronomically large (approximately 10^{50} candidates for 4×4 matrix multiplication), prohibiting exhaustive search and eliminating gradient-based methods. [42] addresses this by using large language models as informed proposal distributions $q_{\text{LLM}}(a'|a)$ that concentrate probability mass on structurally valid algorithmic modifications rather than random perturbations. This approach reduces sample complexity from millions to approximately 150 evaluations through quality-diversity search that maintains a Pareto front of non-dominated solutions across multiple objectives.

Matrix multiplication serves as the canonical example of tensor decomposition: the operation $C = AB$ defines a trilinear form with tensor rank R equal to the minimum number of scalar multiplications required. [51] showed in 1969 that 2×2 matrices require only 7 multiplications (tensor rank $R = 7$) rather than the naive 8, yielding recursive complexity $O(n^{2.807})$ instead of $O(n^3)$. For the practically important 4×4 case, the best known algorithm used 49 multiplications for over 50 years until [16] formulated tensor decomposition as a single-player game and used reinforcement learning (AlphaTensor) to discover algorithms with 48 multiplications (standard) and 47 multiplications (modular arithmetic). [42] subsequently achieved 48 multiplications for complex-valued matrices by representing algorithms as executable code and evolving them using LLM-guided mutations, demonstrating complementary strengths: RL discovers from scratch via tensor structure while LLM evolution refines existing solutions through syntactic transformations.

Proof assistants like Lean 4 provide absolute certainty by converting informal mathematics into machine-checkable formal proofs, but formalization faces a severe bottleneck: the expertise barrier, library navigation difficulty (which of 210,000+ theorems in Mathlib are relevant?), and the granularity gap where informal "obvious" steps expand to 50+ formal tactics. [49] addresses this through AI assistance, achieving 74% automation of proof steps (versus 40% baseline) via three mechanisms: tactic suggestion based on the current proof state, automated multi-step proof search combining neural guidance with symbolic reasoning, and premise selection using learned mathematical relevance patterns. The result is a human-AI collaboration pattern where humans provide overall proof strategy and mathematical insight while AI handles boilerplate tactics, library search, and routine sub-goal completion.

The future vision connects discovery and verification through four emerging mechanisms: discovery-to-verification pipelines where AlphaEvolve candidates are auto-translated to Lean specifications for human-assisted proof; verified code evolution that restricts search to provably correct algorithm spaces; proof-guided search using partial proof progress as reward signals; and formal-informal translation where LLMs bridge natural mathematical language and formal Lean syntax. Currently, these capabilities exist separately—discovery systems find novel algorithms and verification systems formalize known mathematics—but integration remains an open problem. The medium-term outlook (5-10 years) anticipates automated verification of AI discoveries and proof-guided evolution systems that learn to search toward provable algorithms.

The lecture demonstrates that AI has transcended its role as a tool for solving mathematics to become an active participant in mathematical discovery and verification, exemplifying the course's central thesis that computational mathematics and AI advance together. As the final lecture, it completes the bidirectional arc: we began with machine learning as function approximation and neural networks as computational tools (Lectures 1–3), progressed through how mathematical theory informs AI (Lectures 4–6), explored how AI solves computational mathematics problems (Lectures 7–9), and conclude here with AI discovering new mathematics itself while formal methods provide rigorous certification, closing the loop to show that progress in one field necessarily advances the other.

4 Basic Definitions and Mathematical Background

This section establishes notation and recalls essential mathematical concepts used throughout the lectures. All notation below is extracted from the ten lecture outlines and organized systematically.

4.1 Notation

Sets and Spaces

- $\mathbb{R}^n$ or $\mathbb{R}^n$: n-dimensional Euclidean space
- $\mathbb{R}^{m\times n}$: Space of $m \times n$ real matrices
- $\mathcal{X}$: Input/feature space; also infinite-dimensional function space (domain) for operator learning

- $\mathcal{Y}$: Output/label space; also infinite-dimensional function space (codomain) for operator learning
- Θ or parameter space: Space of model parameters
- $\mathcal{N}(i)$: Local neighborhood of neuron i (CNN connectivity)
- $V \to W$: Function space mapping from space V to space W
- A: Algorithm space (discrete, combinatorial space of valid algorithms)

Functions and Operators

- f: Generic function (true underlying function)
- $f_{\boldsymbol{\theta}} : \mathcal{X} \to \mathcal{Y}$: Parametric function with parameters $\boldsymbol{\theta} \in \Theta$
- $\hat{f}$: Estimated/learned function
- F: Forward operator (inverse problems); also general operator map; context-dependent usage
- $F_{\boldsymbol{\theta}}(\mathbf{x})$: Neural network function output (full network with capital F)
- $f_{\boldsymbol{\theta}}(h)$: Individual layer or residual block output (lowercase f)
- $F(\mathbf{h}_\ell, \boldsymbol{\theta}_\ell)$: Residual block: $F(\mathbf{h}_\ell, \boldsymbol{\theta}_\ell) = \mathbf{h}_\ell + f_{\boldsymbol{\theta}}(\mathbf{h}_\ell)$
- G: General operator between function spaces; also inverse mapping $G_{\boldsymbol{\theta}}(y)$
- $\{f_{\boldsymbol{\theta}} : \boldsymbol{\theta} \in \Theta\}$: Hypothesis class
- $u(x,t)$: Solution to PDE depending on space x and time t
- $\Phi(x,t)$: Value function in optimal control and HJB equations
- π: Policy function (reinforcement learning)

Probability and Statistics

- $\mathbb{E}[X]$ or $\mathbb{E}[X]$: Expectation of random variable X
- $\mathcal{D}$: Data distribution
- p or p_t: Probability density (time-dependent)
- $p_{\mathcal{X}}$: Data distribution to match (generative modeling)
- $p_{\mathcal{Z}}$: Latent/noise distribution (typically Gaussian)
- p_c: Class probability (softmax output)
- $p(y|x)$: Likelihood function
- $\pi(x)$: Prior distribution (Bayesian inference)
- $\pi(x|y)$: Posterior distribution given observations
- ρ_t: Population density (time-dependent)

- σ^2: Variance (noise variance, gradient variance)
- $\{(x_i, y_i)\}_{i=1}^n$: Training dataset with n samples
- $\hat{R}(\theta)$: Empirical risk (training loss)
- $R(\theta)$: True risk (expected loss)

Optimization and Gradients

- ∇f or ∇f: Gradient of function f
- $\nabla^2 f$ or H or H_f: Hessian matrix of function f
- $\nabla_x \Phi$: Gradient with respect to x
- $\Delta\Phi$: Laplacian operator (sum of second derivatives)
- $\partial_t \Phi$: Partial derivative with respect to time
- $\arg\min$: Argument that minimizes a function
- η or η_t: Learning rate/step size at iteration t
- α_k: Step size at iteration k
- λ: Regularization parameter
- κ: Condition number (ratio of largest to smallest eigenvalues)

Loss Functions

- $\ell(\cdot, \cdot)$: Per-sample loss function on single example
- $\ell(f_{\boldsymbol{\theta}}(\mathbf{x}_i), \mathbf{y}_i)$: Loss on single training sample
- $\ell(x, u)$: Running cost/loss function (optimal control)
- $L(\boldsymbol{\theta})$ or $L_n(\boldsymbol{\theta})$: Batch/empirical loss over N samples: $L_N(\boldsymbol{\theta}) = \frac{1}{N}\sum_{i=1}^{N} \ell(f_{\boldsymbol{\theta}}(\mathbf{x}_i), \mathbf{y}_i)$
- $\mathcal{L}(\boldsymbol{\theta})$: Expected risk/population loss (theoretical objective)
- $\mathcal{L}_{\text{MSE}}$: Mean squared error expected loss (regression)
- $\mathcal{L}_{\text{CE}}$: Cross-entropy expected loss (classification)
- $\mathcal{L}_{\text{GPT}}$: GPT objective expected loss (next-token prediction)
- $\mathcal{L}(\boldsymbol{\theta}, \lambda)$: Regularized expected loss: $\mathcal{L}(\boldsymbol{\theta}, \lambda) = \mathcal{L}(\boldsymbol{\theta}) + \lambda R(\boldsymbol{\theta})$
- $L_{\text{mod}}(\boldsymbol{\theta})$: Modified batch loss (backward error analysis)
- L_{PDE}: PDE residual batch loss
- L_{BC}: Boundary condition batch loss
- L_{data}: Data fitting batch loss

Vectors (bold lowercase)

- $\mathbf{x}$ or $\mathbf{x}$: Input vector; parameter vector in inverse problems
- $\mathbf{y}$ or $\mathbf{y}$: Output/target vector; observation vector in inverse problems
- $\boldsymbol{\theta}$ or $\boldsymbol{\theta}$: Parameter vector (always boldface)
- $\boldsymbol{\theta}_0$: Initial parameters
- $\mathbf{h}$ or $\mathbf{h}^{(\ell)}$: Hidden layer activations at layer ℓ
- $\mathbf{b}$ or $\mathbf{b}^{(\ell)}$: Bias vector at layer ℓ
- v_t: Velocity vector (momentum methods); also second moment in Adam; also velocity field in CNF
- m_t: First moment (exponential moving average of gradients)
- g_t: Gradient at time t

Matrices (bold uppercase)

- $\mathbf{X}$: Data matrix/input features
- $\mathbf{Y}$: Target matrix (multi-output)
- $\mathbf{W}$ or $\mathbf{W}^{(\ell)}$: Weight matrix at layer ℓ
- $\mathbf{A}$: Adjacency matrix (graph structure)
- $\mathbf{A}_{\text{attn}}$: Attention adjacency matrix (Transformers)
- $\tilde{\mathbf{A}}$: Normalized adjacency matrix (GCN)
- $\mathbf{L}$: Graph Laplacian ($L = D - A$)
- $\mathbf{D}$: Degree matrix (graph diagonal degree matrix)
- $\tilde{\mathbf{D}}$: Normalized degree matrix (GCN)
- $\mathbf{H}$ or $\mathbf{H}^{(\ell)}$: Hidden layer matrix at layer ℓ
- $\mathbf{Q}$: Query matrix (Transformer attention)
- $\mathbf{K}$: Key matrix (Transformer attention)
- $\mathbf{V}$: Value matrix (Transformer attention)
- H: Hessian matrix of per-sample loss; also Fisher block for weights
- J: Jacobian matrix
- $\mathbf{F}(\boldsymbol{\theta})$: Fisher information matrix
- $\mathbf{F}_W$: Fisher block for weight matrix (K-FAC approximation $\mathbf{F}_W \approx A \otimes S$)
- A: Activation covariance matrix $\mathbb{E}[hh^T]$
- S: Error covariance matrix $\mathbb{E}[\delta\delta^T]$
- I: Identity matrix

Norms and Distances

- $\|\cdot\|$: Generic norm
- $\|\cdot\|_2$: Euclidean (L2) norm
- $\|\cdot\|_2^2$: Squared L2 norm
- $\|\cdot\|_F$: Frobenius norm $\sqrt{\sum_{i,j} a_{ij}^2} = \sqrt{\mathrm{tr}(A^T A)}$
- $\|\cdot\|_\infty$: Maximum norm (L-infinity)
- $\|A\|_2$: Spectral norm (operator norm) $= \sigma_1(A)$ (largest singular value)

Time and Indexing

- t: Time variable (continuous or discrete iteration index)
- k: Iteration index
- ℓ or L: Layer index; also total number of layers
- i: Sample/token index
- c or C: Class index; also total number of classes
- N: Number of training samples (uppercase)
- n: Number of features/input dimension (lowercase)
- p: Number of parameters (lowercase)
- d: Dimension parameter (problem dimension)
- d_k: Key/query dimension (Transformer attention)
- b: Batch size (mini-batch size)
- M: Grid points per dimension; also time steps (*Note: context-dependent*)
- q: Latent dimension
- n_{in}: Input dimension to a layer
- n_{out}: Output dimension from a layer
- r: Singular value index

Stochastic Processes and SDEs

- dW or dW_t: Wiener process/Brownian motion increment
- X_t: State trajectory at time t
- Y_t: Value function along trajectory
- Z_t: Gradient term in backward SDE
- u_t: Control input at time t
- ξ: Random sample/mini-batch (stochastic optimization)
- ϵ or ε: Noise variable; temperature parameter (Langevin); numerical stability constant (Adam)
- σ: Diffusion coefficient; noise level; gradient variance (*Note: context-dependent*)

Special Functions and Operators

- $\sigma(\cdot)$: Activation function (nonlinearity)
- $\text{softmax}(\cdot)$: Softmax function (classification output)
- $\text{sign}(c_t)$: Sign function (Lion optimizer)
- $E(\cdot)$: Encoder function (autoencoder)
- $D(\cdot)$: Decoder function (autoencoder)
- $\delta(\cdot)$: Dirac delta function
- $\otimes$ or $\otimes$: Kronecker product operator
- $K(\mathbf{x}, \mathbf{x}')$: Neural tangent kernel

Continuous Normalizing Flows and Diffusion

- $v_t(x, t)$: Velocity field (time-dependent vector field)
- $x(t)$: Trajectory at time t
- $\psi_t(x_0, x_1)$: Conditional path $(1 - t)x_0 + tx_1$
- $u_t(x|x_0, x_1)$: Conditional velocity $x_1 - x_0$
- $s_t(x)$ or $\nabla_x \log p_t(x)$: Score function (gradient of log-density)
- α_t: Mean schedule (time-dependent scaling)
- σ_t: Variance schedule (time-dependent noise level)
- $g(t)$: Diffusion coefficient (noise schedule)

Complexity Notation

- $O(\cdot)$: Big-O notation (asymptotic complexity)
- $O(1/\sqrt{t})$: Convex SGD convergence rate
- $O(1/t^2)$: Nesterov optimal convergence rate
- $O(\kappa)$: Convergence complexity (condition number dependent)
- $O(\sqrt{\kappa})$: Momentum method convergence rate
- $O(p)$: Linear complexity in parameters
- $O(2p)$: Double parameter memory (Adam)
- $O(p^2)$: Quadratic complexity (Hessian memory)
- $O(p^3)$: Cubic complexity (matrix inversion)
- $O(M^d)$: Exponential in dimension (curse of dimensionality)
- $O(N^{-1/2})$: Monte Carlo convergence rate

4.2 Notational Conventions Adopted

The following conventions are used throughout these notes for consistency and clarity:

Loss Functions

- $\ell(\cdot,\cdot)$: Per-sample loss on individual examples
- $L(\boldsymbol{\theta})$ or $L_N(\boldsymbol{\theta})$: Batch/empirical loss over N training samples
- $\mathcal{L}(\boldsymbol{\theta})$: Expected/population loss (theoretical objective)
- Regularized loss: $\mathcal{L}(\boldsymbol{\theta}, \lambda) = \mathcal{L}(\boldsymbol{\theta}) + \lambda R(\boldsymbol{\theta})$

Parameters

- Parameters are always boldface: $\boldsymbol{\theta}$ or $\boldsymbol{\theta}$
- Neural network: $F_{\boldsymbol{\theta}}(\mathbf{x})$ for full network output; $f_{\boldsymbol{\theta}}(h)$ for individual layers
- Residual block: $F(\mathbf{h}_\ell, \boldsymbol{\theta}_\ell) = \mathbf{h}_\ell + f_{\boldsymbol{\theta}}(\mathbf{h}_\ell)$

Dimensions

- N: Number of training samples (uppercase)
- n: Number of features/input dimension (lowercase)
- p: Number of parameters (lowercase)
- Layer indices: $h^{(\ell)}$ (superscript in parentheses) for layer ℓ

Matrices and Vectors

- All vectors and matrices use boldface: $\mathbf{x}$, $\boldsymbol{\theta}$, $\mathbf{F}$, $\mathbf{H}$, etc.
- Fisher information matrix: $\mathbf{F}(\boldsymbol{\theta})$
- Hessian: H or $\nabla^2 f$ (non-bold, context-dependent meaning)

Context-Dependent Symbols The following symbols intentionally have multiple meanings based on context (this is standard mathematical notation):

- v_t: Velocity (momentum methods), second moment (Adam), velocity field (continuous normalizing flows)
- ϵ: Temperature (Langevin dynamics), numerical stability constant (Adam), noise variable (diffusion models)
- F: Forward operator (inverse problems), Fisher information, general operator; context clarifies meaning
- M: Grid points, time steps; context determines meaning
- σ: Gradient variance, diffusion coefficient, noise level; context determines meaning
- H: Hessian, Fisher block, hidden layer matrix; context clarifies meaning

Operator Notation

- Gradient: ∇ or ∇ used interchangeably (equivalent)
- Integration with implicit dependencies: Often $\mathbf{A}$, $\mathbf{b}$, etc. appear without explicit parameter dependence when context is clear

4.3 Linear Algebra Essentials

Definition 4.1 (Singular Value Decomposition)**.** For any matrix $A \in \mathbb{R}^{m\times n}$, the singular value decomposition (SVD) is

$$A = U\Sigma V^T$$

where $U \in \mathbb{R}^{m\times m}$ and $V \in \mathbb{R}^{n\times n}$ are orthogonal matrices, and $\Sigma \in \mathbb{R}^{m\times n}$ is diagonal with non-negative entries $\sigma_1 \geq \sigma_2 \geq \cdots \geq 0$ (the singular values).

Definition 4.2 (Matrix Norms)**.** For $A \in \mathbb{R}^{m\times n}$:

- Frobenius norm: $\|A\|_F = \sqrt{\sum_{i,j} a_{ij}^2} = \sqrt{\text{tr}(A^T A)}$
- Spectral norm (operator norm): $\|A\|_2 = \sigma_1(A)$ (largest singular value)

4.4 Optimization Foundations

Definition 4.3 (Gradient and Gradient Descent)**.** For a differentiable function $f : \mathbb{R}^n \to \mathbb{R}$, the gradient descent update is

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \alpha_k \nabla f(\boldsymbol{\theta}_k)$$

where $\alpha_k > 0$ is the step size (learning rate) at iteration k.

Definition 4.4 (Convexity)**.** A function $f : \mathbb{R}^n \to \mathbb{R}$ is convex if for all $x, y \in \mathbb{R}^n$ and $\lambda \in [0, 1]$:

$$f(\lambda x + (1 - \lambda)y) \leq \lambda f(x) + (1 - \lambda)f(y)$$

For differentiable f, this is equivalent to: $f(y) \geq f(x) + \nabla f(x)^T (y - x)$ for all x, y.

Definition 4.5 (Smoothness)**.** A function $f : \mathbb{R}^n \to \mathbb{R}$ is L-smooth if its gradient is L-Lipschitz continuous:

$$\|\nabla f(x) - \nabla f(y)\| \leq L\|x - y\|, \quad \forall x, y \in \mathbb{R}^n$$

4.5 Probability and Statistics

Definition 4.6 (Risk Minimization)**.** Given a loss function $\mathcal{L}$ and data distribution $\mathcal{D}$, the goal is to minimize the expected risk:

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

In practice, we minimize the empirical risk over training data $\{(x_i, y_i)\}_{i=1}^N$:

$$\hat{R}(\boldsymbol{\theta}) = \frac{1}{N} \sum_{i=1}^{N} \mathcal{L}(f_{\boldsymbol{\theta}}(x_i), y_i)$$

Definition 4.7 (Bias-Variance Decomposition)**.** For regression with squared loss, the expected prediction error decomposes as:

$$\mathbb{E}[(y - \hat{f}(x))^2] = \text{Bias}^2 + \text{Variance} + \text{Irreducible Error}$$

where the bias and variance measure the model's systematic error and sensitivity to training data, respectively.

4.6 Neural Network Fundamentals

Definition 4.8 (Feedforward Neural Network)**.** A feedforward neural network with L layers is a composition of affine transformations and nonlinear activations:

$$f_{\boldsymbol{\theta}}(x) = W_L \sigma(W_{L-1}\sigma(\cdots \sigma(W_1 x + b_1) \cdots) + b_{L-1}) + b_L$$

where $\boldsymbol{\theta} = \{W_1, b_1, \ldots, W_L, b_L\}$ are the weights and biases, and σ is an activation function.

Theorem 4.9 (Universal Approximation)**.** *A feedforward neural network with a single hidden layer containing a finite number of neurons can approximate any continuous function on a compact subset of $\mathbb{R}^n$ to arbitrary accuracy, provided the activation function is non-polynomial.*

5 Acknowledgments

This conference is supported under the NSF CBMS Award Number 2430460 and by the Department of Mathematics at the University of Houston and the course is supported in part by the NSF award DMS-2038118.

References

[1] S.-I. Amari. Natural gradient works efficiently in learning. *Neural Computation*, 10(2):251–276, 1998.

[2] V. Antun, F. Renna, C. Poon, B. Adcock, and A. C. Hansen. On instabilities of deep learning in image reconstruction and the potential costs of ai. *Proceedings of the National Academy of Sciences*, 117(48):30088–30095, 2020.

[3] A. G. Baydin, B. A. Pearlmutter, A. A. Radul, and J. M. Siskind. Automatic differentiation in machine learning: A survey. *Journal of Machine Learning Research*, 18(153):1–43, 2018.

[4] J.-D. Benamou and Y. Brenier. A computational fluid mechanics solution to the monge-kantorovich mass transfer problem. *Numerische Mathematik*, 84(3):375–393, 2000.

[5] L. Bottou, F. E. Curtis, and J. Nocedal. Optimization methods for large-scale machine learning. *SIAM Review*, 60(2):223–311, 2018.

[6] M. M. Bronstein, J. Bruna, T. Cohen, and P. Veličković. Geometric deep learning: Grids, groups, graphs, geodesics, and gauges, 2021.

[7] R. T. Q. Chen, Y. Rubanova, J. Bettencourt, and D. K. Duvenaud. Neural ordinary differential equations. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 31, 2018.

[8] T. Chen and H. Chen. Universal approximation to nonlinear operators by neural networks with arbitrary activation functions and its application to dynamical systems. *IEEE Transactions on Neural Networks*, 6(4):911–917, 1995.

[9] X. Chen, C. Liang, D. Huang, E. Real, K. Wang, Y. Liu, H. Pham, X. Dong, T. Luong, C.-J. Hsieh, Y. Lu, and Q. V. Le. Symbolic discovery of optimization algorithms. *arXiv preprint arXiv:2302.06675*, 2023.

[10] H. Chung, J. Kim, M. T. McCann, M. L. Klasky, and J. C. Ye. Diffusion posterior sampling for general noisy inverse problems. In *International Conference on Learning Representations (ICLR)*, 2023.

[11] J. Cohen, S. Kaur, Y. Li, J. Z. Kolter, and A. Talwalkar. Gradient descent on neural networks typically occurs at the edge of stability. In *International Conference on Learning Representations (ICLR)*, 2021.

[12] K. Cranmer, J. Brehmer, and G. Louppe. The frontier of simulation-based inference. *Proceedings of the National Academy of Sciences*, 117(48):30055–30062, 2020.

[13] G. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems*, 2(4):303–314, 1989.

[14] L. de Moura and S. Ullrich. The Lean 4 theorem prover and programming language. In *International Conference on Automated Deduction (CADE)*, pages 625–635. Springer, 2021.

[15] L. Dinh, R. Pascanu, S. Bengio, and Y. Bengio. Sharp minima can generalize for deep nets. In *International Conference on Machine Learning (ICML)*, 2017.

[16] A. Fawzi, M. Balog, A. Huang, T. Hubert, B. Romera-Paredes, M. Barekatain, A. Novikov, F. J. R. Ruiz, J. Schrittwieser, G. Swirszcz, et al. Discovering faster matrix multiplication algorithms with reinforcement learning. *Nature*, 610(7930):47–53, 2022.

[17] A. Ferguson, M. LaFleur, L. Ruthotto, J. Thaler, et al. The future of artificial intelligence and the mathematical and physical sciences (ai+mps), 2025. NSF Workshop White Paper.

[18] A. Ferguson, M. LaFleur, L. Ruthotto, J. Thaler, Y.-S. Ting, P. Tiwary, S. Villar, E. P. Alves, J. Avigad, S. Billinge, C. Bilodeau, K. Brown, E. Candes, A. Chattopadhyay, B. Cheng, J. Clausen, C. Coley, A. Connolly, F. Daum, S. Dong, C. X. Du, C. Dvorkin, C. Fanelli, E. B. Ford, L. M. Frutos, N. García Trillos, C. Garraffo, R. Ghrist, R. Gomez-Bombarelli, G. Guadagni, S. Guggilam, S. Gukov, J. B. Gutiérrez, S. Habib, J. Hachmann, B. Hanin, P. Harris, M. Holland, E. Holm, H.-Y. Huang, S.-C. Hsu, N. Jackson, O. Isayev, H. Ji, A. Katsaggelos, J. Kepner, Y. Kevrekidis, M. Kuchera, J. N. Kutz, B. Lalic, A. Lee, M. LeBlanc, J. Lim, R. Lindsey, Y. Liu, P. Y. Lu, S. Malik, V. Mandic, V. Manian, E. P. Mazi, P. Mehta, P. Melchior, B. Ménard, J. Ngadiuba, S. Offner, E. Olivetti, S. P. Ong, C. Rackauckas, P. Rigollet, C. Risko, P. Romero, G. Rotskoff, B. Savoie, U. Seljak, D. Shih, G. Shiu, D. Shlyakhtenko, E. Silverstein, T. Sparks, T. Strohmer, C. Stubbs, S. Thomas, S. Vaikuntanathan, R. Vidal, F. Villaescusa-Navarro, G. Voth, B. Wandelt, R. Ward, M. Weber, R. Wechsler, S. Whitelam, O. Wiest, M. Williams, Z. Yang, Y. G. Yingling, B. Yu, S. Yue, A. Zabludoff, H. Zhao, and T. Zhang. The future of artificial intelligence and the mathematical and physical sciences (ai+mps), 2025.

[19] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016.

[20] J. Han, A. Jentzen, and W. E. Solving high-dimensional partial differential equations using deep learning. *Proceedings of the National Academy of Sciences*, 115(34):8505–8510, 2018.

[21] M. Hardt, B. Recht, and Y. Singer. Train faster, generalize better: Stability of stochastic gradient descent. In *International Conference on Machine Learning (ICML)*, pages 1225–1234, 2016.

[22] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer, 2nd edition, 2009.

[23] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

[24] C. F. Higham and D. J. Higham. Deep learning: An introduction for applied mathematicians. *SIAM Review*, 61(4):860–891, 2019.

[25] Z. Hu, Z. Shi, G. E. Karniadakis, and K. Kawaguchi. Hutchinson trace estimation for high-dimensional and high-order physics-informed neural networks. *Computer Methods in Applied Mechanics and Engineering*, 424:116883, 2024.

[26] A. Jacot, F. Gabriel, and C. Hongler. Neural tangent kernel: Convergence and generalization in neural networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 31, 2018.

[27] B. Kawar, M. Elad, S. Ermon, and J. Song. Denoising diffusion restoration models. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 35, pages 23593–23606, 2022.

[28] N. S. Keskar, D. Mudigere, J. Nocedal, M. Smelyanskiy, and P. T. P. Tang. On large-batch training for deep learning: Generalization gap and sharp minima. In *International Conference on Learning Representations (ICLR)*, 2017.

[29] Patrick Kidger. On neural differential equations, 2022.

[30] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2015.

[31] A. S. Krishnapriyan, A. Gholami, S. Zhe, R. M. Kirby, and M. W. Mahoney. Characterizing possible failure modes in physics-informed neural networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 34, 2021.

[32] X. Li, D. Verma, and L. Ruthotto. A neural network approach for stochastic optimal control. *SIAM Journal on Scientific Computing*, 46(5):A3094–A3117, 2024.

[33] Z. Li, N. Kovachki, K. Azizzadenesheli, B. Liu, K. Bhattacharya, A. Stuart, and A. Anandkumar. Fourier neural operator for parametric partial differential equations. In *International Conference on Learning Representations (ICLR)*, 2021.

[34] Y. Lipman, R. T. Q. Chen, H. Ben-Hamu, M. Nickel, and M. Le. Flow matching for generative modeling. In *International Conference on Learning Representations (ICLR)*, 2023.

[35] I. Loshchilov and F. Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations (ICLR)*, 2019.

[36] L. Lu, P. Jin, G. Pang, Z. Zhang, and G. E. Karniadakis. Learning nonlinear operators via DeepONet based on the universal approximation theorem of operators. *Nature Machine Intelligence*, 3(3):218–229, 2021.

[37] J. Martens and R. Grosse. Optimizing neural networks with kronecker-factored approximate curvature. In *International Conference on Machine Learning (ICML)*, pages 2408–2417, 2015.

[38] S. Mei, A. Montanari, and P.-M. Nguyen. A mean field view of the landscape of two-layer neural networks. *Proceedings of the National Academy of Sciences*, 115(33):E7665–E7671, 2018.

[39] T. M. Mitchell. *Machine Learning.* McGraw-Hill, 1997.

[40] K. P. Murphy. *Probabilistic Machine Learning: An Introduction.* MIT Press, 2022.

[41] J. Nocedal and S. J. Wright. *Numerical Optimization.* Springer, 2nd edition, 2006.

[42] A. Novikov, A. Fawzi, et al. AlphaEvolve: A coding agent for scientific and algorithmic discovery. *Google DeepMind Technical Report*, 2025.

[43] Derek Onken, Samy Wu Fung, Xingjian Li, and Lars Ruthotto. OT-flow: Fast and accurate continuous normalizing flows via optimal transport. *Proceedings of the AAAI Conference on Artificial Intelligence*, 35(10):9223–9232, 2021.

[44] M. Raissi. Forward-backward stochastic neural networks: Deep learning of high-dimensional partial differential equations. *arXiv preprint arXiv:1804.07010*, 2018.

[45] M. Raissi, P. Perdikaris, and G. E. Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, 378:686–707, 2019.

[46] H. Robbins and S. Monro. A stochastic approximation method. *The Annals of Mathematical Statistics*, 22(3):400–407, 1951.

[47] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, 1986.

[48] R. Schaeffer, M. Khona, Z. Robertson, A. Boopathy, and I. R. Fiete. Double descent demystified: Identifying, interpreting & ablating the sources of a deep learning puzzle. *arXiv preprint arXiv:2303.14151*, 2023.

[49] P. Song, K. Yang, and A. Anandkumar. Lean Copilot: Large language models as copilots for theorem proving in Lean. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024.

[50] Y. Song, J. Sohl-Dickstein, D. P. Kingma, A. Kumar, S. Ermon, and B. Poole. Score-based generative modeling through stochastic differential equations. In *International Conference on Learning Representations (ICLR)*, 2021.

[51] V. Strassen. Gaussian elimination is not optimal. *Numerische Mathematik*, 13(4):354–356, 1969.

[52] M. Takamoto, T. Praditia, R. Leiteritz, D. MacKinlay, F. Alesiani, D. Pflüger, and M. Niepert. PDEBench: An extensive benchmark for scientific machine learning. In *NeurIPS Datasets and Benchmarks*, 2022.

[53] V. Trifonov, A. Rudikov, O. Iliev, Y. M. Laevsky, I. Oseledets, and E. Muravleva. Learning from linear algebra: A graph neural network approach to preconditioner design for conjugate gradient solvers. *arXiv preprint arXiv:2405.15557*, 2024.

[54] V. N. Vapnik. *Statistical Learning Theory.* Wiley-Interscience, 1998.

[55] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 30, 2017.

[56] S. Wang, Y. Teng, and P. Perdikaris. Understanding and mitigating gradient flow pathologies in physics-informed neural networks. *SIAM Journal on Scientific Computing*, 43(5):A3055–A3081, 2021.

[57] Z. O. Wang, R. Baptista, Y. Marzouk, L. Ruthotto, and D. Verma. Efficient neural network approaches for conditional optimal transport with applications in bayesian inference. *arXiv preprint arXiv:2310.16975*, 2023.

[58] D. H. Wolpert and W. G. Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82, 1997.