Function Approximation:
From Linear Models to Neural Networks

Scientific Machine Learning

Alena Kopaničáková

Toulouse INP / ENSEEIHT

Spring 2026

Outline

Course Organization

9 Lectures (CTD)

Theory + worked examples in class

Each lecture builds on the previous: approximation → PINNs → operators → hybrid

6 Lab Sessions (TP)

Hands-on implementation in Python / PyTorch

- Code models from scratch
- Run experiments, interpret results
- Reproduce key theoretical findings

Final Exam

Balanced between:

Theory - derivations, error analysis, proofs, conceptual questions

Practice - design models, interpret results, choose architectures, pseudocode, discuss practical difficulties

What is Scientific Machine Learning?

SciML sits at the intersection of machine learning, numerical analysis, applied mathematics and optimization. Its goal: learn accurate, fast computational models of physical systems from data and physical structure.

Classical Simulation

- Physics-based, first principles
- Mesh / discretization required
- Rigorous error bounds, mature theory
- Expensive: minutes to hours per run

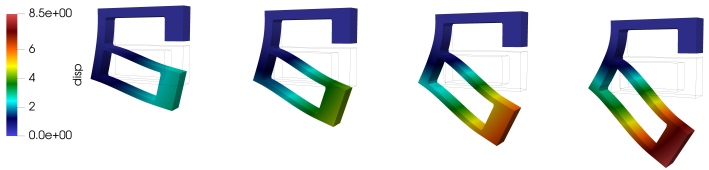
Pure Machine Learning

- Fully data-driven; no physics
- Fast inference after training
- No built-in conservation or symmetry
- Needs large labeled datasets

SciML: Combining Both Worlds

Encode physical structure *inside* ML models - faster than solvers at query time, more data-efficient and reliable than black-box ML

Parametric partial differential equations (PDEs)



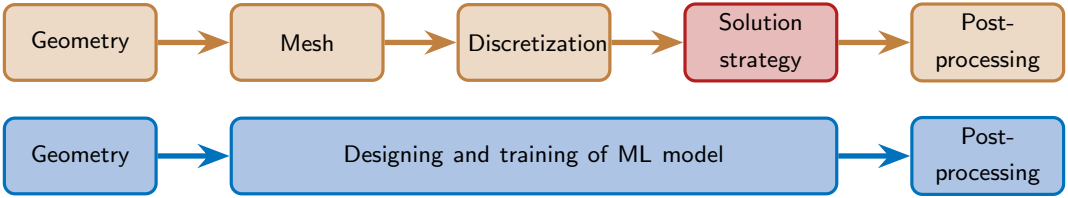
Let $\Omega \subset \mathbb{R}^d$, $\boldsymbol{A} \subset \mathbb{R}^P$, and $\mathcal{V} = \mathcal{V}(\Omega)$ be a Hilbert space. The parametric problem is given in abstract strong form as:
For a given $\boldsymbol{\alpha} \in \boldsymbol{A}$, find the solution $u(\boldsymbol{\alpha}) \in \mathcal{V}$ such that

$$\mathcal{P}(\boldsymbol{\alpha})u(\boldsymbol{\alpha}) = f(\boldsymbol{\alpha}), \quad \text{in } \mathcal{V}',$$

where $\mathcal{P}(\boldsymbol{\alpha}) : \mathcal{V} \to \mathcal{V}'$ is a differential operator and $f(\boldsymbol{\alpha})$ is a linear continuous form

- Applications areas: uncertainty quantification, design optimization, or inverse problems, ...
- Problem can be parametrized, for example, wrt. material parameters, source term, or boundary conditions, ...

Solution of a PDE



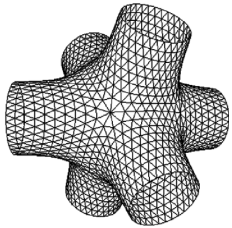
"Classical" numerical methods

- High-fidelity solutions using advanced methods from numerics, scientific computing, optimization, ...
- Capture physics on multiple scales
- Capture well the details and high-frequency features
- Encounter challenges in data integration

Scientific machine-learning (SciML)

- Exceptionally cheap low-fidelity surrogates using novel machine-learning (ML)
- Discover systems dynamics
- Capture well global trends and low-frequency features (spectral bias)
- Seamless integration of data

“Classical” numerical methods - high-fidelity solution



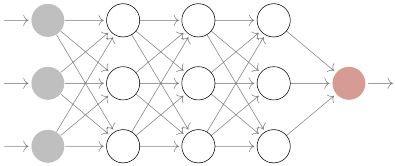
High-fidelity discretizations

- Finite elements/Finite differences/...
  - Requires meshing
- High accuracy, but also high computational cost, especially as the problem size grows
- Resolves well high-frequency features

Solution strategies

- Discretization gives rise to the linear/non-linear system of equations
- Various type of linear (from direct to iterative) and nonlinear (1st, 2nd order) solution strategies can be employed
- The most efficient and robust solvers for large-scale problems typically utilize divide and conquer approaches, e.g., multilevel, domain-decomposition, field-split, ...

Scientific machine-learning (SciML)



Key idea

Use ML to enhance the numerical simulations, e.g.,

- Process and analyze large volume of data, which can be then used for example as BC/IC
- Mesh design and adaptivity
- Approximate solution of PDEs using DNNs
- Initialization of the high-fidelity numerical solvers
- Replacing specific parts of the high-fidelity simulation codes
- Pre/post-processing, visualization and results interpretation
- ...

SciML: Approximate solution of PDEs

Physics-informed learning (PINNs)

- Focus on a single instance PDE
- Integration of the data and physics
- "Mesh-free" (complex geo, high-dim. problems)
- Discovery of physics and model parameters
- Initial related work[1,2,3]

Operator learning

- Model function-to-function relations
- Focus on a family of parametric PDEs
- Generalization across parameter spaces
- Initial related work[4,5,6]

[1] Dissanayake, Phan-Thien. "Neural-network-based approximations for solving partial differential equations." 1994.

[2] Lagaris, Isaac E., Aristidis Likas, and Dimitrios I. Fotiadis. "Artificial neural networks for solving ordinary and partial differential equations." 1998.

[3] Raissi et al. "Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations." 2017.

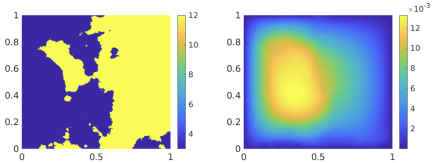
[4] Chen, Chen. "Universal approximation to nonlinear operators by neural networks with arbitrary activation functions and its application to dynamical systems." 1995.

[5] Lu et al. "Deeponet: Learning nonlinear operators for identifying differential equations based on the universal approximation theorem of operators." 2019.

[6] Li, Zongyi, et al. "Fourier neural operator for parametric partial differential equations." 2020.

Example 1: Darcy PDEs

$$-\nabla(a\nabla u) = f, \quad v \in \Omega$$
$$v = 0, \quad v \in \partial\Omega.$$



(A) Input a (B) Output v

Quantities of interest are:

- u is temperature or pressure
- a is conductance or permeability
- f is source

Examples of many-query problem

Classical solvers are accurate but slow. One high-fidelity run: minutes to hours.

Settings that demand many runs

- **Design optimization:** Evaluate drag/lift for $\sim 10^4$ geometries
- **Uncertainty quantification:** Propagate a distribution over uncertain inputs
- **Real-time control:** Need $u(\mu)$ at actuator rate, not solver rate
- **Inverse problems:** Forward map queried $\sim 10^6\times$ inside MCMC

The Bottleneck

Many-query setting: 10^3-10^6 solves required. At hours per run - **infeasible**

ML Solution: Amortize the Cost

Train surrogate $\mathcal{G}_\theta \approx \mathcal{G}^\dagger$ *once* (offline). Query in milliseconds (online). Cost shared over all future queries

Three SciML Paradigms

Physics-Informed Learning (PINNs)

Embed the PDE residual as a soft penalty in the training loss $\Longrightarrow$ **no mesh, data optional**

Neural Operators (NO)

Learn $\mathcal{G}_\theta \approx \mathcal{G}^\dagger$ from input-output pairs (a_n, u_n) produced by a classical solver $\Longrightarrow$ **no PDE required at inference**

Hybrid Methods

Neural network components embedded inside a classical numerical solver $\Longrightarrow$ Retains stability guarantees; network corrects solver errors

This Course at a Glance

Module 1: Function Approximation

Hypothesis class, bias-variance trade-off, linear/Fourier/NN models, UAT, SGD

Module 2: Physics-Informed NNs

PINN loss, collocation, convergence, failure modes, extensions, inverse problems, equation discovery

Module 3: Operator Learning

DeepONet, Fourier Neural Operator, operator approximation theory

Module 4: Hybrid Methods

ML-accelerated solvers, learned preconditioners, neural closures

The Supervised Learning Problem

Setting: Dataset of N labeled examples

$$\mathcal{D} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^{N}, \quad \boldsymbol{x}_i \in \mathbb{R}^d,\ y_i \in \mathbb{R},$$

generated by an unknown target $f^\star : \mathbb{R}^d \to \mathbb{R}$ with noise:

$$y_i = f^\star(\boldsymbol{x}_i) + \varepsilon_i, \qquad \varepsilon_i \overset{\text{i.i.d.}}{\sim} \mathcal{N}(0, \sigma^2)$$

Goal: Construct $\hat{f} : \mathbb{R}^d \to \mathbb{R}$ that *generalizes* to unseen inputs

Context: Scientific Machine Learning

- Surrogate models for parametric PDEs
- Data-driven closure terms
- Operator learning architectures (FNO, DeepONet)

Hypothesis Class

The central design choice in supervised learning:

Definition: Hypothesis Class

A parameterized family of candidate functions:

$$\widehat{\mathcal{F}} = \left\{ f_{\boldsymbol{\theta}} : \mathbb{R}^d \to \mathbb{R} \;\middle|\; \boldsymbol{\theta} \in \Theta \subseteq \mathbb{R}^p \right\}$$

Small $\widehat{\mathcal{F}}$ (few parameters p)

- Easy to optimize
- Generalizes from limited data
- May not approximate $f^\star$ well

$\Rightarrow$ **High bias / underfitting**

Large $\widehat{\mathcal{F}}$ (many parameters p)

- Approximates $f^\star$ closely
- Sensitive to training data
- Harder to optimize

$\Rightarrow$ **High variance / overfitting**

Empirical and Population Risk

We measure quality with the **squared error loss**

Population Risk (True Objective)

$$\mathcal{R}(\hat{f}) = \mathbb{E}_{\boldsymbol{x}}\left[\left|\hat{f}(\boldsymbol{x}) - f^{\star}(\boldsymbol{x})\right|^2\right]$$

Cannot be computed directly - requires the distribution of inputs

Empirical Risk (Training Loss)

$$\widehat{\mathcal{R}}(\hat{f}) = \frac{1}{N}\sum_{i=1}^{N}\left|\hat{f}(\boldsymbol{x}_i) - y_i\right|^2$$

Feasible proxy - what we actually minimize

Key challenge: the *generalization gap* $\mathcal{R}(\hat{f}) - \widehat{\mathcal{R}}(\hat{f})$

A model can achieve $\widehat{\mathcal{R}} = 0$ while $\mathcal{R}$ remains large: **overfitting**

Bias-Variance-Noise Decomposition

Fix any point $\boldsymbol{x}$, the expected squared prediction error decomposes:

Proposition (BVN Decomposition)

$$\mathbb{E}_{\mathcal{D}}\left[\left|\hat{f}(\boldsymbol{x}) - y_{\text{new}}\right|^2\right] = \underbrace{\left(\mathbb{E}_{\mathcal{D}}[\hat{f}(\boldsymbol{x})] - f^\star(\boldsymbol{x})\right)^2}_{\text{Bias}^2} + \underbrace{\text{Var}_{\mathcal{D}}\left[\hat{f}(\boldsymbol{x})\right]}_{\text{Variance}} + \underbrace{\sigma^2}_{\text{Noise}}$$

Bias^2

-systematic error

-zero only if $f^\star \in \widehat{\mathcal{F}}$

-*decreases* as $\widehat{\mathcal{F}}$ grows

Variance

-sensitivity to training data

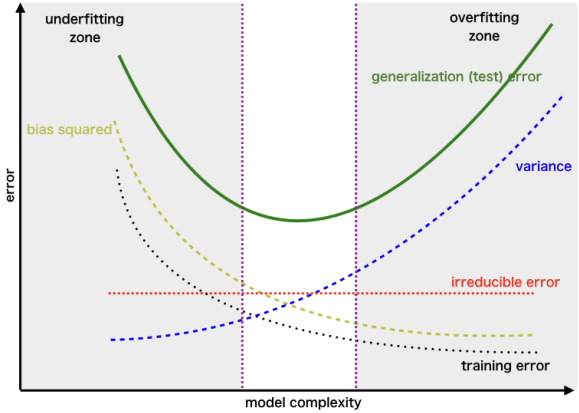
-*increases* as $\widehat{\mathcal{F}}$ grows (typically)

Noise σ^2

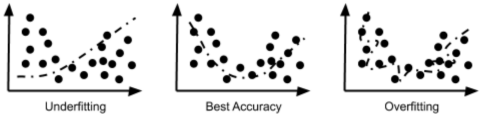
-irreducible

-no estimator can beat this floor

The Bias-Variance Trade-off Illustrated



- **Underfitting:** model too simple, high bias, misses structure
- **Best fit:** optimal complexity, best generalization
- **Overfitting:** model too complex, high variance, fits noise



Three-Term Error Decomposition

A complementary, *algorithmic* decomposition of the generalization error:

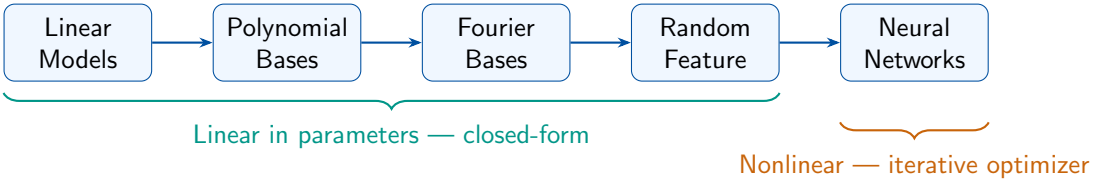
Theorem

$$\mathcal{R}(\hat{f}) \leq \underbrace{\mathcal{R}(f_{\mathcal{G}}^{\star})}_{\text{approximation error}} + \underbrace{2 \sup_{\hat{g} \in \widehat{\mathcal{F}}} \left| \mathcal{R}(\hat{g}) - \widehat{\mathcal{R}}(\hat{g}) \right|}_{\text{generalization gap}} + \underbrace{\widehat{\mathcal{R}}(\hat{f}) - \widehat{\mathcal{R}}(f_{T}^{\star})}_{\text{optimization error}}$$

Term	Meaning	Cause	Remedy
Approximation	Can $\widehat{\mathcal{F}}$ represent $f^{\star}$?	$\widehat{\mathcal{F}}$ too small	Enlarge model
Generalization	Training $\approx$ Testing?	N too small	More data; regularize
Optimization	Minimized $\widehat{\mathcal{R}}$ well?	Poor optimizer	More epochs; better schedule

A Progressive Hierarchy of Models

We progressively enlarge $\widehat{\mathcal{F}}$ from linear functions to neural networks:



For each model family we derive:

1. The model and its training procedure
2. The three error terms: approximation, generalization gap, optimization

Linear Models: Model and Normal Equations

Hypothesis class: $\widehat{\mathcal{F}} = \{\boldsymbol{x} \mapsto \boldsymbol{\theta}^\top \boldsymbol{x} \mid \boldsymbol{\theta} \in \mathbb{R}^d\}, \quad p = d$ parameters.

Empirical risk: With design matrix $\boldsymbol{X} \in \mathbb{R}^{N \times d}$, target $\boldsymbol{y} \in \mathbb{R}^N$:

$$\widehat{\mathcal{R}}(\boldsymbol{\theta}) = \frac{1}{N} \|\boldsymbol{X}\boldsymbol{\theta} - \boldsymbol{y}\|_2^2 \quad \text{(Convex quadratic)}$$

Normal Equations

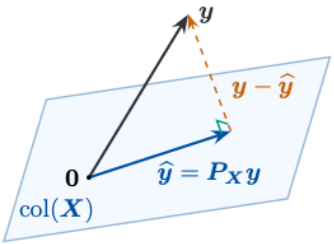
$$\boldsymbol{X}^\top \boldsymbol{X}\, \boldsymbol{\theta}^\star = \boldsymbol{X}^\top \boldsymbol{y} \qquad \Rightarrow \qquad \boldsymbol{\theta}^\star = (\boldsymbol{X}^\top \boldsymbol{X})^{-1} \boldsymbol{X}^\top \boldsymbol{y},$$

when $\boldsymbol{X}$ has full column rank ($N \geq d$)

Linear Models: Geometric Interpretation

Orthogonal Projection

Predictions $\widehat{\boldsymbol{y}} = \boldsymbol{P_X y} = \boldsymbol{X}(\boldsymbol{X}^\top \boldsymbol{X})^{-1}\boldsymbol{X}^\top \boldsymbol{y}$ are the **orthogonal projection** of $\boldsymbol{y}$ onto $\mathrm{col}(\boldsymbol{X})$. Residual $\boldsymbol{y} - \widehat{\boldsymbol{y}}$ is orthogonal to every column of $\boldsymbol{X}$.



- $\widehat{\boldsymbol{y}}$ is the **closest point** in $\mathrm{col}(\boldsymbol{X})$ to $\boldsymbol{y}$
- Residual $\boldsymbol{y} - \widehat{\boldsymbol{y}} \perp \mathrm{col}(\boldsymbol{X})$
- Least-squares finds the best approximation of in col(X)

Underdetermined case ($N < d$)

$\boldsymbol{X}^\top \boldsymbol{X}$ singular: use the **Moore-Penrose pseudoinverse** $\boldsymbol{\theta}^+ = \boldsymbol{X}^+ \boldsymbol{y}$ (minimum-norm solution).

Linear Models: Ridge Regularization

Limitations of plain linear models:

- High bias: cannot capture any nonlinear structure
- Irrelevant features: All d input dimensions contribute with fixed linear weights
- Underdetermined ($N < d$): infinitely many solutions
- Ill-conditioned: small eigenvalues of $\boldsymbol{X}^\top \boldsymbol{X}$ amplify noise variance

Ridge Regression (Tikhonov Regularization)

$$\boldsymbol{\theta}_\lambda^\star = (\boldsymbol{X}^\top \boldsymbol{X} + \lambda \boldsymbol{I})^{-1} \boldsymbol{X}^\top \boldsymbol{y}, \qquad \lambda > 0$$

Linear Models: Error Analysis Summary

Error term	Value	Comment
Approximation	> 0 if $f^\star \notin \widehat{\mathcal{F}}$	Fixed by model family; *does not* improve with more data
Generalization gap	$\mathcal{O}(d/N)$	Shrinks with data; controlled by Ridge ($d \to d_{\text{eff}}$)
Optimization	$= 0$	Exact closed-form via normal equations

Key Insight

Linear models achieve the **best possible optimization error** (zero) but pay a **high approximation cost** for any nonlinear $f^\star$. As we enlarge $\widehat{\mathcal{F}}$: approximation error $\downarrow$ but optimization error $\uparrow$

Polynomial Basis Models: Idea

Key idea: apply a nonlinear *feature map* $\phi : \mathbb{R}^d \to \mathbb{R}^M$, then fit linearly.

For $d = 1$ and degree m: $\quad \phi_k(x) = x^k, \quad k = 0, \ldots, m$:

$$\hat{f}(x) = \boldsymbol{\theta}^\top \boldsymbol{\phi}(x) = \sum_{k=0}^{m} \theta_k \, x^k, \qquad p = m + 1.$$

Design matrix (Vandermonde):

$$\boldsymbol{\Phi} = \begin{pmatrix} 1 & x_1 & \cdots & x_1^m \\ \vdots & & & \vdots \\ 1 & x_N & \cdots & x_N^m \end{pmatrix} \in \mathbb{R}^{N \times M}$$

Linear in Parameters

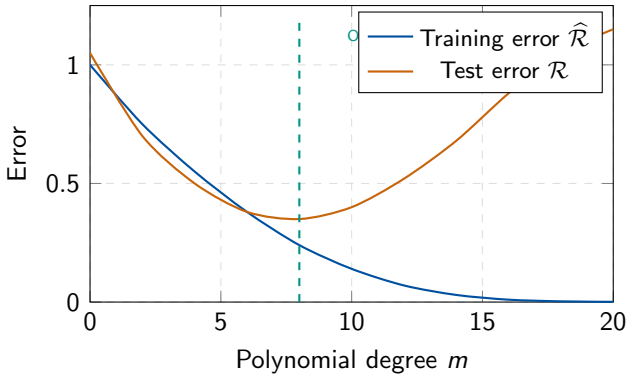
Despite x^k nonlinearity, model is **linear in $\boldsymbol{\theta}$**.

$\Rightarrow$ Same normal equations:

$\boldsymbol{\Phi}^\top \boldsymbol{\Phi} \, \boldsymbol{\theta}^\star = \boldsymbol{\Phi}^\top \boldsymbol{y}$

Closed-form solution available.

Polynomial Models: Bias-Variance Trade-off



As degree m grows:

- **Bias** $\downarrow$: with $N = m + 1$, exact interpolation possible
- **Variance** $\uparrow$: Runge's phenomenon - wild oscillations
- **Conditioning** $\downarrow$: $\kappa(\mathbf{\Phi}^\top \mathbf{\Phi})$ grows rapidly

Common Pitfall

Minimizing training error always favors larger m since $\widehat{\mathcal{R}} \to 0$ as $m \to N$.

Use cross-validation!

Polynomial Models: Error Analysis

Three Error Terms (Degree-m Polynomial, $f^\star \in C^s$)

- **Approximation:** $\mathcal{O}(m^{-2s})$ (Weierstrass: can drive to zero by increasing m)
- **Generalization gap:** $\mathcal{O}((m+1)/N)$, (increases linearly with m)
- **Optimization:** $= 0$ (exact via normal equations)

Optimal degree (balance approximation vs. generalization):

$$m^\star \asymp N^{\frac{1}{2s+1}}, \qquad \mathcal{R}(\hat{f}_{m^\star}) = \mathcal{O}\left(N^{-\frac{2s}{2s+1}}\right)$$

Smoother $f^\star$ (larger s) $\Rightarrow$ faster convergence

Curse of Dimensionality

- for $f^\star \in C^s([0,1]^d)$ with $d > 1$: $\mathcal{R}(\hat{f}) = \mathcal{O}\left(N^{-\frac{2s}{2s+d}}\right)$
- for $s = 2$, $d = 10$: exponent ≈ 0.29 vs. 0.80 in 1D

Fourier Basis Models

For $f^\star \in L^2([0, 2\pi])$ periodic, the Fourier basis provides sharper guarantees

Fourier Hypothesis Class (Bandwidth K)

$$\widehat{\mathcal{F}}_K = \left\{ x \mapsto \sum_{|k| \le K} \theta_k \, e^{ikx} \;\middle|\; \boldsymbol{\theta} \in \mathbb{C}^{2K+1} \right\}$$

$M = 2K + 1$ parameters; **still linear in $\boldsymbol{\theta}$** $\Rightarrow$ closed-form solution

Special case: equispaced nodes ($x_i = 2\pi(i-1)/N$, $N = M$):

$$\boldsymbol{\Phi}^{\mathsf{H}}\boldsymbol{\Phi} = N\boldsymbol{I} \;\Rightarrow\; \boldsymbol{\theta}^\star = \tfrac{1}{N}\boldsymbol{\Phi}^{\mathsf{H}}\boldsymbol{y} \quad \text{(DFT, computable in } \mathcal{O}(N \log N) \text{ via FFT)}$$

Matrix is unitary: **perfectly conditioned** (no Vandermonde ill-conditioning)

Sobolev Spaces and Approximation Rate

Definition: Sobolev Space H^s

$$H^s([0,2\pi]) = \left\{ f \in L^2 \;\middle|\; \|f\|_{H^s}^2 = \sum_{k\in\mathbb{Z}} (1+|k|^2)^s \, |c_k|^2 < \infty \right\}$$

Larger $s \Rightarrow$ faster decay of Fourier coefficients $c_k \Rightarrow$ smoother f

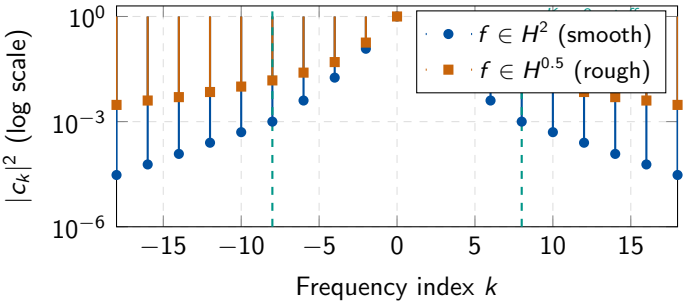
Theorem: Fourier Approximation Error (Jackson-type)

If $f \in H^s$, the truncated Fourier approximation satisfies:

$$\left\| f - \hat{f} \right\|_{L^2}^2 = \sum_{|k|>K} |c_k|^2 \;\leq\; \frac{\|f\|_{H^s}^2}{(1+K^2)^s} = \mathcal{O}(K^{-2s})$$

Key idea (low-pass filtering): Fourier truncation discards all energy at frequencies $|k| > K$. Approximation error = energy in the discarded high-frequency modes.

Fourier Models: Frequency Spectrum



- Smooth $f \in H^2$: rapid coefficient decay - most energy in low modes
- Rough $f \in H^{0.5}$: slow decay - significant energy beyond cutoff

$\Longrightarrow$ Approximation error is the area under curve beyond $K = 8$

Fourier Models: Error Analysis

Error term	Scaling	Comment
Approximation	$\mathcal{O}(K^{-2s})$	Tail energy; geometric for analytic $f^\star$
Generalization	$\mathcal{O}(M/N)$, $M = 2K+1$	Same as polynomial with $p = M$
Optimization	$= 0$	DFT/FFT; perfectly conditioned
Optimal $K^\star$	$N^{1/(2s+1)}$	Same formula as polynomial
Optimal rate	$N^{-2s/(2s+1)}$	Minimax-optimal for $f^\star \in H^s$

Advantage over Polynomial Models

Same trade-off, but optimization is **perfectly conditioned** (DFT is unitary) and solvable in $\mathcal{O}(N \log N)$ via FFT - no Vandermonde issues.

Random Feature Models

Idea: Randomize the feature map; keep it frozen during training

Sample m features: $(\boldsymbol{w}_j, b_j) \overset{\text{i.i.d.}}{\sim} q(\boldsymbol{w}, b)$, freeze them

Random Feature Hypothesis Class

$$\widehat{\mathcal{F}}_m = \left\{ \boldsymbol{x} \mapsto \sum_{j=1}^{m} \theta_j \, \sigma(\boldsymbol{w}_j^\top \boldsymbol{x} + b_j) \;\middle|\; \boldsymbol{\theta} \in \mathbb{R}^m \right\}$$

Only $\boldsymbol{\theta}$ is trained $\implies$ **linear in $\boldsymbol{\theta}$** $\Rightarrow$ closed-form solution via normal equations

Standard examples:

- $\sigma = \cos$, $\boldsymbol{w}_j \sim \mathcal{N}(0, \boldsymbol{I})$, $b_j \sim \mathrm{Uniform}[0, 2\pi]$ $\quad \Rightarrow$ **RBF kernel**
- $\sigma = \mathrm{ReLU}$, $\boldsymbol{w}_j \sim \mathrm{Uniform}(\mathbb{S}^{d-1})$ $\quad \Rightarrow$ **Arc-cosine kernel**

Random Features: Connection to Kernel Methods

Theorem (Rahimi & Recht, 2007)

As $m \to \infty$, the empirical kernel converges almost surely:

$$\frac{1}{m}\phi(\boldsymbol{x})^\top \phi(\boldsymbol{x}') \xrightarrow{a.s.} K(\boldsymbol{x}, \boldsymbol{x}') = \mathbb{E}_q\Big[\sigma(\boldsymbol{w}^\top \boldsymbol{x} + b)\,\sigma(\boldsymbol{w}^\top \boldsymbol{x}' + b)\Big]$$

Random features **approximate a kernel machine** with kernel K.
Approximation error: $\mathcal{O}(1/\sqrt{m})$ uniformly over compact sets.

Two Key Points

Infinite-width limit: As $m \to \infty$, the model converges to a **Gaussian Process** with covariance $K(\boldsymbol{x}, \boldsymbol{x}')$.

Choice of q encodes the kernel: Choosing q is the analogue of choosing the right basis in polynomial/Fourier models. A fully trained neural network does this *adaptively*.

Random Feature Models: Error Analysis

Error term	Scaling	Comment
Approximation	$\mathcal{O}(m^{-1})$	Monte Carlo rate; dimension-free
Generalization	$\mathcal{O}(m/N)$	m trainable weights
Optimization	$= 0$	Normal equations (Ridge)
Optimal $m^\star$	$\sqrt{N}$	From $m^{-1} \asymp m/N$
Optimal rate	$N^{-1/2}$	Dimension-free

Comparison with Polynomial/Fourier

Slower rate: $N^{-1/2}$ vs. $N^{-2s/(2s+1)}$ - does not exploit smoothness.
Dimension-free: no curse of dimensionality - crucial for high-dimensional PDE inputs.

Neural Networks: Architecture

Shallow Network (1 hidden layer, width m)

$$\hat{f}_{\boldsymbol{\theta}}(\boldsymbol{x}) = \boldsymbol{a}^{\top}\sigma\Big(\boldsymbol{W}^{(1)}\boldsymbol{x} + \boldsymbol{b}^{(1)}\Big), \qquad p = m(d+2)$$

Deep Feedforward Network (L hidden layers)

$$\boldsymbol{h}^{(\ell)} = \sigma\Big(\boldsymbol{W}^{(\ell)}\boldsymbol{h}^{(\ell-1)} + \boldsymbol{b}^{(\ell)}\Big), \quad \ell = 1, \ldots, L \qquad \hat{f}_{\boldsymbol{\theta}}(\boldsymbol{x}) = \boldsymbol{W}^{(L+1)}\boldsymbol{h}^{(L)} + \boldsymbol{b}^{(L+1)}$$

Depth L; **width** $m = \max_{\ell} m_{\ell}$; no activation on the output layer.

Critical requirement: activation σ must be **non-polynomial**.

- ReLU, GELU, tanh, sigmoid, sine: ✓ (universal approximation valid)
- Polynomial or linear: × (collapses to a fixed-degree polynomial - no gain from depth)

Neural Networks vs. Random Features

The *only* difference: in a neural network, the inner-layer weights are also trained.

Property	**Random Features**	**Deep Neural Network**
Inner weights $(\boldsymbol{W}, \boldsymbol{b})$	**Fixed** (random draw from q)	**Trained**
Nonlinear in $(\boldsymbol{W}, \boldsymbol{b})$?	No	Yes
Hypothesis class	Random subspace of $\mathcal{H}_K$	All of $\widehat{\mathcal{F}}_{m,L}$
Closed-form solution?	**Yes** (normal equations)	No
Optimization landscape	**Convex**	Non-convex
Feature adaptation	None (fixed at draw)	**Full** (learned by gradient descent)
Universal approximation	In $\mathcal{H}_K$ (as $m \to \infty$)	**Yes** ($C(K)$, any m large enough)

The Key Trade-off

Training all weights breaks convexity but enables **adaptive feature learning**: The network discovers the right features for the problem automatically

Universal Approximation Theorem

Theorem (Cybenko 1989; Hornik 1991)

Let σ be a continuous non-polynomial activation, $K \subset \mathbb{R}^d$ compact. For any $f \in C(K)$ and $\varepsilon > 0$, there exists a single-hidden-layer network

$$\hat{f}_{\theta}(\boldsymbol{x}) = \boldsymbol{a}^{\top}\sigma(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}) \quad \text{such that} \quad \sup_{\boldsymbol{x}\in K} \left|\hat{f}_{\theta}(\boldsymbol{x}) - f(\boldsymbol{x})\right| < \varepsilon$$

Note: UAT gives **no rate** of convergence, does not say how to find the parameters, and does not say how large m must be.

Depth Separation

Theorem (Telgarsky 2016)

There exist functions computable by a depth-k ReLU network of width w that **cannot** be approximated to fixed accuracy ε by any depth-2 network unless its width is **exponential** in k.

Concretely: for the sawtooth function of period 2^{-k},
depth-k network needs $\mathcal{O}(k)$ neurons,
depth-2 approximation needs $\Omega(2^{k/2})$ neurons.

Key idea: depth enables composition

- Layer 1: sawtooth of period $1/2$
- Layer i: double the frequency by composing with itself
- After k layers: period 2^{-k} with $\mathcal{O}(k)$ neurons
- Flat network must represent 2^{k-1} oscillations independently $\Rightarrow$ exponential width

Practical takeaway

Deep networks are **exponentially more efficient** than shallow ones for hierarchically structured functions.

Curse of Dimensionality

UAT was proved in 1D. What happens for $f : [0,1]^d \to \mathbb{R}$?

Extending the bump construction to d dimensions:

- Partition $[0,1]^d$ into a grid: n divisions per axis $\Rightarrow$ n^d hypercubes
- Each d-dimensional bump requires $\mathcal{O}(d)$ neurons
- Total: $\mathcal{O}(d \cdot n^d)$ neurons; setting $n \sim 1/\varepsilon$ gives neurons needed $\sim \mathcal{O}\left(\frac{d}{\varepsilon^d}\right)$

d	$\varepsilon = 0.1$	$\varepsilon = 0.01$
1	10	10^2
10	10^{10}	10^{20}
100	10^{100}	10^{200}
784	10^{784}	10^{1568}

Curse of dimensionality

For worst-case functions, the exponential cost is **unavoidable** (Novak & Woźniakowski).

But: real data has structure!

- MNIST digits: ∼10–20 true DOF
- Hierarchical composition $\Rightarrow$ depth helps
- CNNs exploit translation symmetry

Escaping the Curse: Barron & Yarotsky

Barron's Theorem (1993) - dimension-free rate

If f belongs to *Barron's class* (bounded Fourier moment C_f), then a single-hidden-layer sigmoid network with N neurons satisfies

$$\mathrm{MSE} \;\le\; \mathcal{O}\!\left(\frac{C_f^2}{N}\right), \qquad \textbf{independent of } d.$$

Price: Barron's class is smaller than $C(K)$; C_f can itself be large.

Yarotsky's Theorem (2017) - tight bound for deep ReLU

For $f \in W^{s,\infty}([0,1]^d)$, $\|f\|_{W^{s,\infty}} \le 1$, there exists a ReLU network with

$$\text{depth} \le C_{s,d} \log \tfrac{1}{\varepsilon}, \qquad \#\text{nonzero weights} \le C_{s,d}\, \varepsilon^{-d/s} \log \tfrac{1}{\varepsilon},$$

such that $\|\hat{f} - f\|_{L^\infty} \le \varepsilon$. The weight count is **optimal** (up to log).

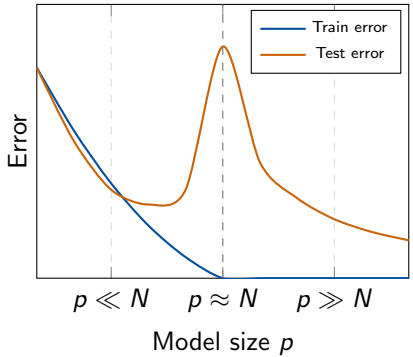
Bias-Variance Trade-off for Neural Networks

Classical regime ($p \ll N$): same U-curve as before

- Larger $p \Rightarrow$ lower bias, higher variance
- Optimal $p^\star$ balances the two

Double descent (Belkin et al. 2019):

1. Classical U-curve (underparameterized)
2. **Peak** at interpolation threshold $p \approx N$
3. **Second descent** as $p \gg N$



Why does it generalize?

GD selects the **minimum-norm interpolant** - implicit regularization acts as Tikhonov penalty

Error Decomposition for Neural Networks

$$\underbrace{\mathcal{R}(\hat{f})}_{\text{gen. error}} \leq \underbrace{\mathcal{R}(f_G^\star)}_{\text{approx. error}} + \underbrace{2 \sup_{\hat{g}} \left| \mathcal{R}(\hat{g}) - \widehat{\mathcal{R}}(\hat{g}) \right|}_{\text{gen. gap}} + \underbrace{\widehat{\mathcal{R}}(\hat{f}) - \widehat{\mathcal{R}}(f_T^\star)}_{\text{optim. error}}$$

Approximation error (Yarotsky)

$$\mathcal{R}(f_G^\star) = \mathcal{O}\Big(P^{-2s/d} \log^{2s/d} P\Big)$$

P nonzero weights, $f^\star \in W^{s,\infty}([0,1]^d)$.
Optimal up to log; depth $\mathcal{O}(\log P)$ suffices.

Optimization error

> 0 in general (non-convex)

Generalization gap (Rademacher)

For **regression**: use Rademacher complexity

$$\widehat{\mathcal{R}}_N(\widehat{\mathcal{F}}) = \mathbb{E}_{\boldsymbol{\varepsilon}}\Big[\sup_f \tfrac{1}{N} \textstyle\sum_i \varepsilon_i f(\boldsymbol{x}_i)\Big]$$

With prob. $\geq 1 - \delta$:

$$\sup_f |\mathcal{R}(f) - \widehat{\mathcal{R}}(f)| \leq 8B\, \widehat{\mathcal{R}}_N + 3\sqrt{\tfrac{\log(2/\delta)}{2N}}$$

For ReLU nets with P weights:

$$\text{gen. gap} \lesssim \mathcal{O}\sqrt{\tfrac{P \log P}{N}}$$

Training Neural Networks: Non-Convexity and SGD

The empirical risk $\widehat{\mathcal{R}}(\boldsymbol{\theta})$ is **non-convex** in $\boldsymbol{\theta}$ $\Longrightarrow$ no closed-form minimizer

Stochastic Gradient Descent (SGD)

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \eta_t \nabla_{\boldsymbol{\theta}} \widehat{\mathcal{R}}_{\mathcal{B}_t}(\boldsymbol{\theta}^{(t)})$$

$\mathcal{B}_t \subset \{1, \ldots, N\}$: mini-batch; η_t: learning rate schedule.

Gradient computation:

Backpropagation = chain rule layer by layer

Cost $\mathcal{O}(p)$ per step - same as forward pass

Practical tricks:

- **Adam/AdaGrad:** adaptive steps-sizes per-parameter
- **Cosine / warm-up** schedules
- **Gradient clipping**

Backpropagation

Goal: Compute $\nabla_{\boldsymbol{\theta}} \widehat{\mathcal{R}}(\boldsymbol{\theta})$ in $\mathcal{O}(p)$ via the chain rule

Two-Pass Algorithm

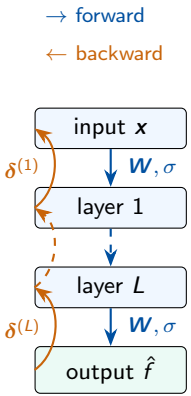
Forward: Cache $\boldsymbol{z}^{(\ell)}, \boldsymbol{h}^{(\ell)}$ for every layer

$$\boldsymbol{z}^{(\ell)} = \boldsymbol{W}^{(\ell)} \boldsymbol{h}^{(\ell-1)} + \boldsymbol{b}^{(\ell)}, \quad \boldsymbol{h}^{(\ell)} = \sigma(\boldsymbol{z}^{(\ell)})$$

Backward: Propagate error signal $\boldsymbol{\delta}$ from output to input

$$\boldsymbol{\delta}^{(\ell)} = \left(\boldsymbol{W}^{(\ell+1)}\right)^{\top} \boldsymbol{\delta}^{(\ell+1)} \odot \sigma'(\boldsymbol{z}^{(\ell)})$$

Grad: $\nabla_{\boldsymbol{W}^{(\ell)}} \widehat{\mathcal{R}} = \boldsymbol{\delta}^{(\ell)} \left(\boldsymbol{h}^{(\ell-1)}\right)^{\top}$



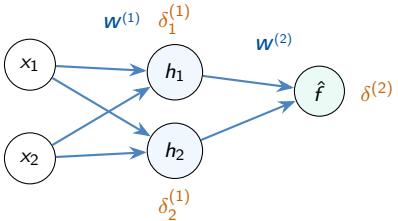
Cost **same order** as a forward pass $\Rightarrow$ scalable to millions of parameters

Backpropagation: 2-Layer Worked Example

Network: $\hat{f}(\boldsymbol{x}) = \boldsymbol{W}^{(2)}\sigma(\boldsymbol{W}^{(1)}\boldsymbol{x} + \boldsymbol{b}^{(1)}) + b^{(2)}, \quad \ell = \frac{1}{2}(\hat{f}(\boldsymbol{x}) - y)^2$

Forward pass (cache everything):

$$\boldsymbol{z}^{(1)} = \boldsymbol{W}^{(1)}\boldsymbol{x} + \boldsymbol{b}^{(1)} \in \mathbb{R}^m$$
$$\boldsymbol{h}^{(1)} = \sigma(\boldsymbol{z}^{(1)}) \in \mathbb{R}^m$$
$$z^{(2)} = \boldsymbol{W}^{(2)}\boldsymbol{h}^{(1)} + b^{(2)} \in \mathbb{R}$$
$$\ell = \tfrac{1}{2}(z^{(2)} - y)^2$$



Backward pass (output → input):

$$\delta^{(2)} = \frac{\partial \ell}{\partial z^{(2)}} = \tfrac{1}{2} \cdot 2(z^{(2)} - y) = z^{(2)} - y \quad (\tfrac{1}{2} \text{ cancels})$$

$$\boldsymbol{\delta}^{(1)} = \frac{\partial \ell}{\partial \boldsymbol{z}^{(1)}} = \left(\boldsymbol{W}^{(2)}\right)^\top \delta^{(2)} \odot \sigma'(\boldsymbol{z}^{(1)})$$

Cost

Forward: $\mathcal{O}(md + m)$
Backward: $\mathcal{O}(md + m)$
$\Rightarrow$ gradient is **same cost** as forward pass (vs. $\mathcal{O}(p^2)$ for finite differences)

Neural Networks: Error Analysis

Error term	Scaling	Comment
Approximation	$\mathcal{O}\big((mL)^{-2s/d}\big)$	UAT: any $C(K)$ approx.; rate depends on m, L, s, d
Generalization gap	Complex	Depends on capacity (VC dim, Rademacher), not just p/N
Optimization	> 0 in general	Non-convex landscape; iterative optimizer; no global min guarantee

The Price of Expressivity

Going from random features to neural networks:

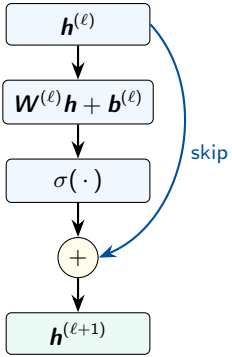
- Approximation: Dramatic improvement through adaptive feature learning
- Generalization: Richer analysis (but often works well in practice)
- Optimization: **No longer zero** - must be carefully managed

Architecture: Residual Networks (ResNet)

Residual Block (He et al. 2016)

$$\boldsymbol{h}^{(\ell+1)} = \sigma\left(\boldsymbol{W}^{(\ell)}\boldsymbol{h}^{(\ell)} + \boldsymbol{b}^{(\ell)}\right) + \boldsymbol{h}^{(\ell)}$$

- **Gradient highway:** bypasses σ, no vanishing gradients
- **Residual:** learn $\Delta \boldsymbol{h} \approx 0$ around identity
- **Neural ODE:** $\dot{\boldsymbol{h}} = F(\boldsymbol{h}, t)$ as $L \to \infty$



Architecture: CNNs - Idea and Properties

1D Convolution Layer (C_{in} channels, kernel size K)

$$h^{(\ell)}_{c',j} = \sigma\Big(\sum_c \sum_k W_{c',c,k}\, h^{(\ell-1)}_{c,j+k} + b_{c'}\Big), \qquad k \in [-K/2,\, K/2]$$

Weights $W_{c',c,k}$ **shared across all positions** j.

Key properties

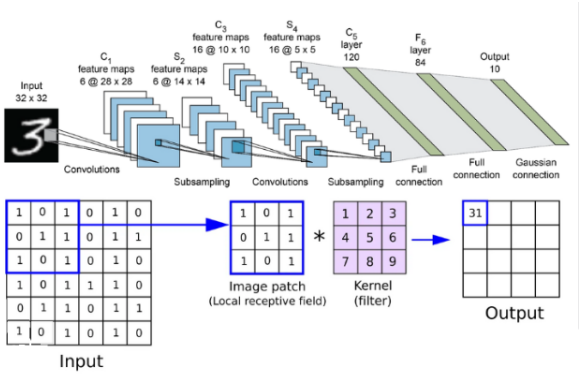
- **Parameter efficiency:** $\mathcal{O}(C_{\text{in}} C_{\text{out}} K)$ vs. $\mathcal{O}(C_{\text{in}} C_{\text{out}} n^2)$ (dense); saving factor $n^2/K \gg 1$
- **Translation equivariance:** shifting input by δ shifts output by δ; same filter detects the feature at any location
- **Locality:** output at j sees a window of size K; L layers $\Rightarrow$ receptive field $\mathcal{O}(KL)$

Limitation in SciML

Tied to a **fixed grid**: model trained at resolution n cannot evaluate at $n' \neq n$ without retraining.

$\Rightarrow$ Primary motivation for **FNO** and **DeepONet** (resolution-invariant by construction).

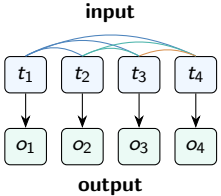
Architecture: CNNs example



Architecture: Transformer

Scaled Dot-Product Attention (Vaswani et al. 2017)

$$\mathrm{Attn}(Q, K, V) = \mathrm{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right)V$$



- **Global receptive field** from layer 1
- **Data-dependent:** weights vary with input
- **Permutation equivariant** + positional encoding

Architecture: U-Net

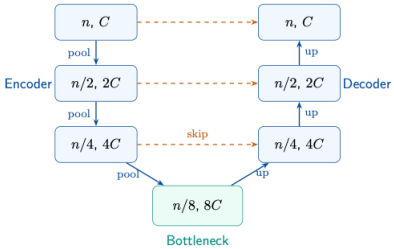
Encoder-Decoder + Skip Connections (Ronneberger et al. 2015)

Encoder: downsample, double channels

Bottleneck: coarsest scale, global structure

Decoder: upsample + concatenate encoder features

- **Multi-scale:** coarse global + fine local
- **Skip connections:** recover spatial precision
- **Cost** scales with depth, not domain size



Grand Unified Comparison

	Linear	**Polynomial**	**Fourier**	**Rand. Feat.**	**Neural Net**
Parameters	d	$m+1$	$2K+1$	m	$p \gg 1$
Approx. error	> 0 (fixed)	$\mathcal{O}(m^{-2s})$	$\mathcal{O}(K^{-2s})$	$\mathcal{O}(m^{-1})$	$\mathcal{O}((mL)^{-2s/d})$
Gen. gap	$\mathcal{O}(d/N)$	$\mathcal{O}(M/N)$	$\mathcal{O}(M/N)$	$\mathcal{O}(m/N)$	complex
Optim. error	0	0	0	0	> 0
Training	closed-form	closed-form	FFT	closed-form	SGD
Curse dimens.	mild	**yes**	**yes**	**no**	mitigated
Features	fixed linear	fixed poly	fixed Fourier	random	**adaptive**

Central Trade-off of the Course

Expressivity $\uparrow \iff$ Optimization difficulty $\uparrow \iff$ Closed-form $\rightarrow$ iterative