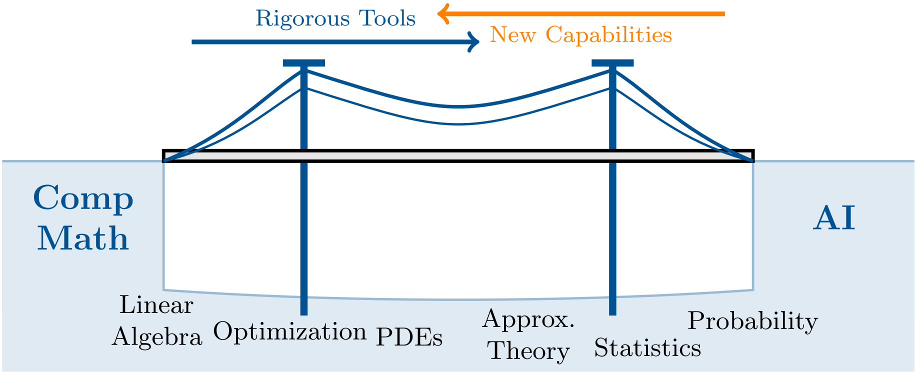
Computational Mathematics and AI

Lecture 2: Neural Network Architectures and Loss Functions

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

slido.com #CBMS25

Course Framework: The Bidirectional Exchange



slido.com
#CBMS25

- **Lecture 1:** Overview of ML paradigms and modern phenomena
- **Today (Lecture 2):** Neural network architectures and loss functions
- Forward connection to **Lecture 3:** Optimization and training

Reading List: Neural Network Architectures and Losses

Historical Context: Neural networks evolved from finite-depth perceptrons through convolutional networks to modern transformers and continuous-time architectures.

Key Readings:

1. Goodfellow et al. (2016) – *Deep Learning*, MIT Press

   Comprehensive coverage of architectures and training.
2. Bronstein et al. (2021) – Geometric Deep Learning.

   Unifying framework for CNNs, GNNs, Transformers.
3. Vaswani et al. (2017) – Attention Is All You Need.

   Transformer architecture and self-attention.
4. He et al. (2016) – Deep Residual Learning.

   Skip connections enabling very deep networks.
5. Kidger (2022) – On Neural Differential Equations.

   Comprehensive thesis/textbook on neural ODEs/SDEs/CDEs.

Lecture Outline: Universal Approximation $\rightarrow$ CNNs/GNNs/Transformers $\rightarrow$ ResNets & Neural ODEs $\rightarrow$ Loss Functions

Today's Roadmap

Goal: Design the machine learning problem

$$\mathcal{L}(\boldsymbol{\theta}) = \min_{\boldsymbol{\theta}} \mathbb{E}\left[\ell(F_{\boldsymbol{\theta}}(\mathbf{x}), \mathbf{y})\right]$$

Part 1: Connecting the Dots

- Layer connectivity patterns
- CNNs, GNNs, Transformers
- Structure $\rightarrow$ Invariance

Part 2: Going Deep

- MLPs $\rightarrow$ ResNets $\rightarrow$ Neural ODEs
- Depth as discretization parameter

Part 3: Loss Functions

- MSE (regression)
- Cross-entropy (classification)
- Examples: Autoencoders, GPT

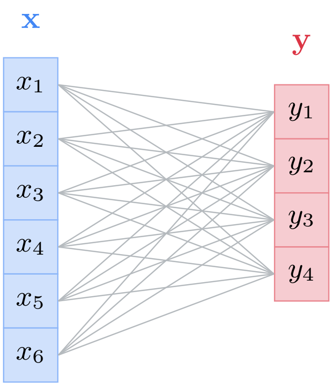
Focus: Explain common ingredients and use cases

Layer Design: Connecting the Dots

From Vectors to Structured Data

Fully-connected baseline: $\mathbf{y} = \sigma(\mathbf{W}\mathbf{x} + \mathbf{b})$

- Input $\mathbf{x} \in \mathbb{R}^n$, output $\mathbf{y} \in \mathbb{R}^m$
- $\mathbf{W}$ is dense — every input connects to every output
- No assumptions about structure in $\mathbf{x}$



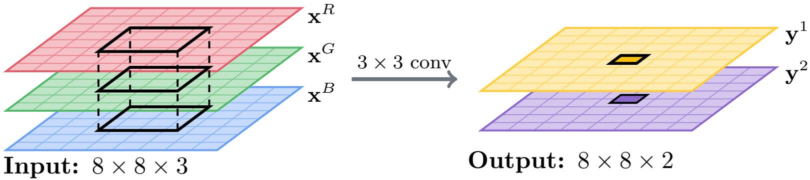
But data often has structure:

- Images: grid of pixels with RGB channels
- Sequences: ordered tokens (text, time series)
- Graphs: nodes + edges (molecules, social networks)

Limitations: fully connected, input/output size fixed

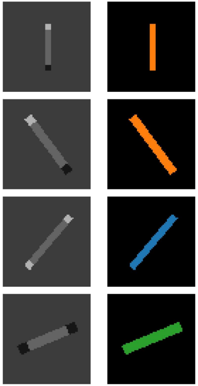
Architecture design = choosing how to connect features

CNNs: Block-Sparse & Weight Sharing



Convolution as block matrix:

$$\begin{bmatrix} \mathbf{y}^1 \\ \mathbf{y}^2 \end{bmatrix} = \begin{bmatrix} C(\boldsymbol{\theta}_{11}) & C(\boldsymbol{\theta}_{12}) & C(\boldsymbol{\theta}_{13}) \\ C(\boldsymbol{\theta}_{21}) & C(\boldsymbol{\theta}_{22}) & C(\boldsymbol{\theta}_{23}) \end{bmatrix} \begin{bmatrix} \mathbf{x}^R \\ \mathbf{x}^G \\ \mathbf{x}^B \end{bmatrix}$$

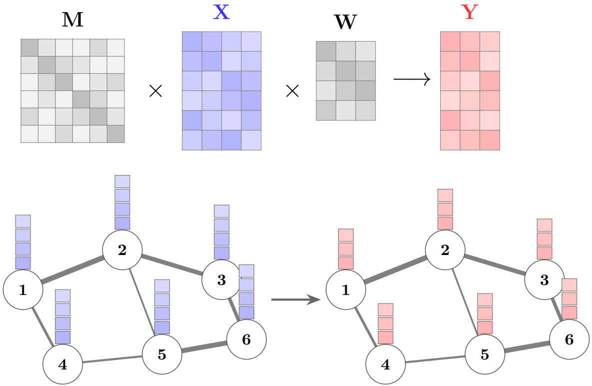


Each $C(\theta)$ is a convolution operator:

- **Sparse:** Each output pixel depends only on local neighborhood
- **Shared weights:** Same θ applied at every spatial location

CNN = block-sparse $\mathbf{W}$, weight sharing, limited field of view

GNNs: Message Passing on Graphs



GNN layer:

$$\mathbf{Y} = \sigma\left(\mathbf{M}\,\mathbf{X}\,\mathbf{W}\right)$$

- $\mathbf{X} \in \mathbb{R}^{N\times d}$: **input features** (nodal)
- $\mathbf{M} \in \mathbb{R}^{N\times N}$: **message passing matrix**
- $\mathbf{W} \in \mathbb{R}^{d\times d'}$: **feature transformation**

Example choices for $\mathbf{M}$: (fixed a-priori, not learned)

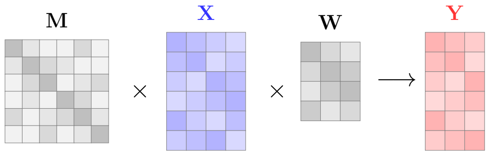
- Graph Laplacian: $\mathbf{M} = \mathbf{D}^{-1/2}(\mathbf{D} - \mathbf{A})\mathbf{D}^{-1/2}$ (diffusion on graph)
- Normalized adjacency: $\mathbf{M} = \mathbf{D}^{-1}\mathbf{A}$ (average over neighbors)

take away: GNN = message passing $\mathbf{M}$ $\times$ features $\times$ MLP

GNN Advantages and the Missing Graph Problem

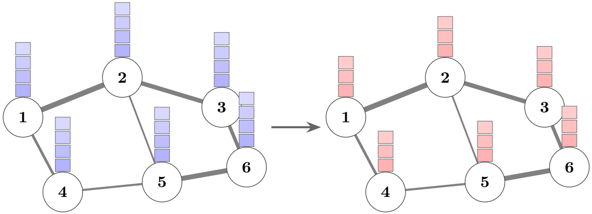
Key advantages:

- same network works for **any graph**
- $\mathbf{W}$ shared across all nodes
- Respects graph structure (permutation equivariant)



Use cases with known graph:

- molecules (atoms + bonds)
- PDE meshes (nodes + connectivity)
- social networks (people + relationships)



The missing graph problem:

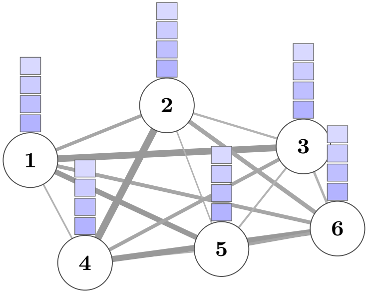
- what if connectivity is **unknown**?
- text: which words relate to which?

Solution: Learn the connectivity — this leads to **attention**

Attention: Learning Which Nodes Connect

Step 1: Project features into Query, Key, Value

$$\mathbf{Q} = \mathbf{X}\mathbf{W}_Q, \quad \mathbf{K} = \mathbf{X}\mathbf{W}_K, \quad \mathbf{V} = \mathbf{X}\mathbf{W}_V$$



- $\mathbf{Q}$ (Query): What is node i looking for?
- $\mathbf{K}$ (Key): What does node j contain?
- $\mathbf{V}$ (Value): What information does node j send?

Step 2: Compute similarity via dot product

$$\text{score}(i,j) = \frac{\mathbf{q}_i \cdot \mathbf{k}_j}{\sqrt{d_k}}$$

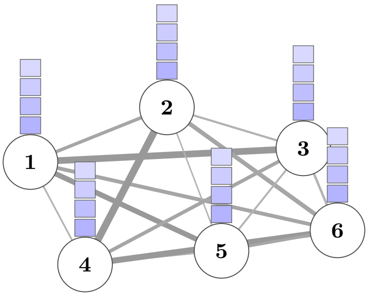
- High score $\Rightarrow$ node i should attend to node j
- Scaling by $\sqrt{d_k}$ prevents large values

key idea: Similarity = learned function of the data itself

Attention: Aggregation and the Full Formula

Step 3: Normalize scores to get attention weights

$$\alpha_{ij} = \frac{\exp(\text{score}(i,j))}{\sum_k \exp(\text{score}(i,k))} \quad \text{(softmax)}$$



- α_{ij} = how much node i attends to node j
- Weights sum to 1: $\sum_j \alpha_{ij} = 1$

Matrix form: the attention formula

$$\boxed{\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_k}}\right)\mathbf{V}}$$

Attention = learned adjacency matrix applied to values

The Transformer Block

Self-attention layer: $\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_k}}\right)\mathbf{V}$

This is a GNN layer with **data-dependent** connectivity (dense, learned)

Full transformer block:

1. Self-attention: $\mathbf{Z} = \text{Attention}(\mathbf{X}\mathbf{W}_Q, \mathbf{X}\mathbf{W}_K, \mathbf{X}\mathbf{W}_V)$
2. Residual + LayerNorm: $\mathbf{H}' = \text{LayerNorm}(\mathbf{X} + \mathbf{Z})$
3. Feed-forward (per node): $\mathbf{H}'' = \text{FFN}(\mathbf{H}')$
4. Residual + LayerNorm: $\mathbf{Y} = \text{LayerNorm}(\mathbf{H}' + \mathbf{H}'')$

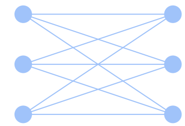
Extensions:

- **Multi-head:** multiple attention patterns in parallel
- **Causal masking:** prevent attending to future (autoregressive)

Transformer = GNN with learned (dense) adjacency + FFN

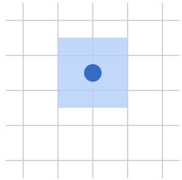
Σ: Layer Design Summary

Fully-connected layer



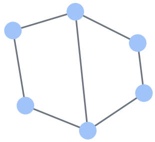
- no structural assumptions on data
- fixed input/output dimensions

CNN (Convolutional Neural Network)



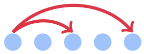
- local receptive fields (sparse connectivity)
- weight sharing across spatial locations
- translation equivariant

GNN (Graph Neural Network)



- message passing on arbitrary graphs
- same weights for all nodes (permutation equivariant)
- works on variable-size inputs

Transformer

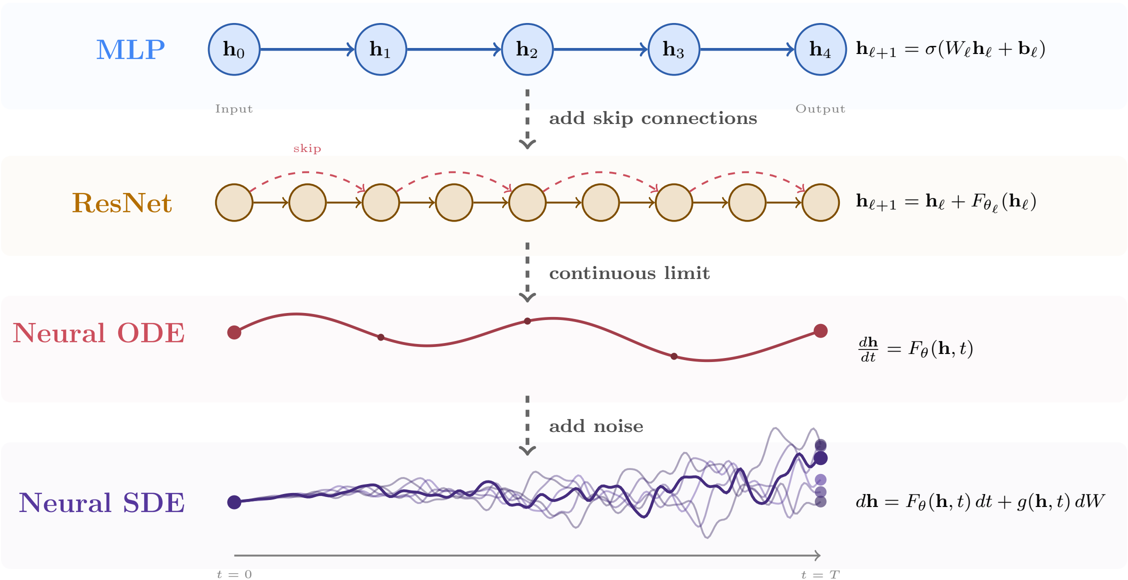


- learned connectivity via attention
- data-dependent weights (dense, adaptive)
- permutation equivariant (with positional encoding)

take away: Architecture = connectivity pattern = encoded invariance

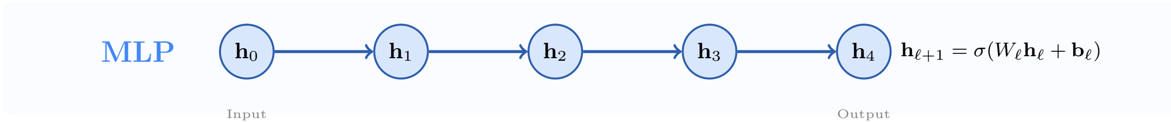
Depth

From Layers to Networks: Going Deep



Theory: width suffices vs. Practice: the deeper, the better

Multilayer Perceptrons: Foundation



Composition structure:

$$\mathbf{h}_0 = \mathbf{x}, \quad f_\theta(\mathbf{x}) = \mathbf{h}_L, \quad \mathbf{h}_\ell = \sigma(\mathbf{W}_\ell \mathbf{h}_{\ell-1} + \mathbf{b}_{\ell-1}), \quad \ell = 1, \ldots, L$$

- Depth = number of layers L (discrete, finite)
- Each layer: Affine transformation + nonlinearity

Challenge: Vanishing/exploding gradients

- Deep MLPs ($L > 20$) hard to train
- Gradient magnitude decays/explodes exponentially with depth
- Motivates residual connections

Universal Approximation: Width Suffices (in Theory)

Theorem (Cybenko (1989), Pinkus (1999))

Let σ be a non-polynomial continuous activation. Single-layer networks

$$f_{\boldsymbol{\theta}}(\mathbf{x}) = \mathbf{W}_2\,\sigma(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1) + \mathbf{b}_2, \quad \mathbf{W}_1 \in \mathbb{R}^{w\times n},\ \mathbf{W}_2 \in \mathbb{R}^{1\times w}$$

*are **dense** in $C([0,1]^n)$ (continuous functions on the unit cube).*

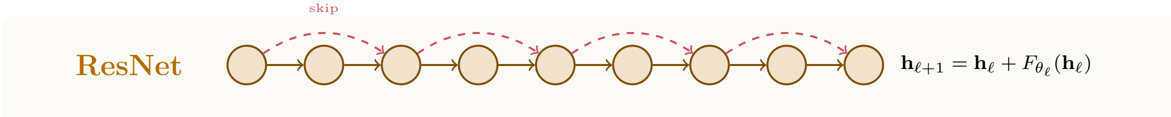
Interpretation: one hidden layer with enough width w approximates any continuous function

Why depth matters in practice:

- UAT is an **existence result** — says nothing about efficiency
- required width w may grow **exponentially** with input dimension n
- deep networks can be **exponentially more efficient** Telgarsky 2016
- optimization: deep narrow networks often easier to train

Theory: width suffices | Practice: depth wins

Residual Networks (ResNets): Skip Connections



Innovation: Identity shortcut paths

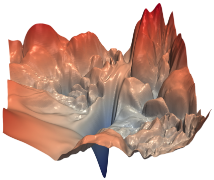
$$\mathbf{h}_{\ell+1} = \mathbf{h}_\ell + F_{\boldsymbol{\theta}_\ell}(\mathbf{h}_\ell)$$

- F: Residual block (typically 2-3 conv layers)
- Identity path: $\mathbf{h}_\ell$ propagates directly
- Residual block learns perturbation

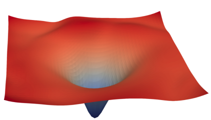
Why this matters:

- Gradient flow: $\frac{\partial \mathcal{L}}{\partial \mathbf{h}_\ell}$ has additive path through identities
- Enables training 100+ layer networks (vs. $\sim$20 without)
- ImageNet breakthrough (He et al., 2015)

impact on loss function (Li et al. 2018)



56-layer network (no ResNet)



56-layer ResNet

Skip connections improve gradient flow in deep networks

Continuous-Time Deep Learning

ResNets: Connection to Discretization

Observation: ResNet update resembles Euler discretization E 2017; Haber and Ruthotto 2017

$$\mathbf{h}_{\ell+1} = \mathbf{h}_\ell + F_{\boldsymbol{\theta}_\ell}(\mathbf{h}_\ell) \quad \longleftrightarrow \quad \mathbf{x}_{n+1} = \mathbf{x}_n + \Delta t \, F_{\boldsymbol{\theta}_n}(\mathbf{x}_n)$$

- Layer index $\ell \leftrightarrow$ time step t_n
- Residual block $F \leftrightarrow$ vector field $F_{\boldsymbol{\theta}_n}$
- Similar structure to ResNet González-García et al. 1998

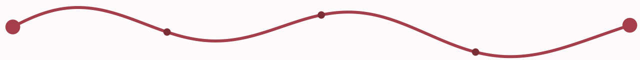
Question: What if we let $\Delta t \to 0$?

- Discrete layers $\to$ continuous time
- Finite compositions $\to$ differential equation
- This motivates Neural ODEs

key insight: ResNet = discretization of continuous transformation

Neural Ordinary Differential Equations (Neural ODEs)

Neural ODE



$\frac{d\mathbf{h}}{dt} = F_\theta(\mathbf{h}, t)$

Define $f_\theta(\mathbf{x}) = \mathbf{h}(T)$ where $\mathbf{h}$ solves the ODE

$$\frac{d\mathbf{h}}{dt} = F_{\boldsymbol{\theta}}(\mathbf{h}(t), t), \quad t \in (0, T] \quad \mathbf{h}(0) = \mathbf{x}$$

- Depth parameter: T (integration time) not L (layer count)
- Continuous trajectory $\mathbf{h}(t)$ instead of discrete layers

Some advantages:

- Can leverage powerful ODE solvers (adaptive time stepping)
- Analyze stability, stiffness, long-term behavior, develop dynamics
- Derive PDE / control interpretations

Depth becomes a continuous parameter

Neural ODEs: Memory Costs

Claim: Neural ODEs have $O(1)$ memory cost via adjoint equation

$$\frac{d}{dt}\begin{bmatrix}\mathbf{h}\\ \mathbf{a}\\ \mathbf{g}\end{bmatrix} = \begin{bmatrix}F_{\boldsymbol{\theta}}(\mathbf{h},t)\\ -\mathbf{a}^\top \frac{\partial F_{\boldsymbol{\theta}}}{\partial \mathbf{h}}(\mathbf{h},t)\\ \mathbf{a}^\top \frac{\partial F_{\boldsymbol{\theta}}}{\partial \boldsymbol{\theta}}(\mathbf{h},t)\end{bmatrix}, \quad t \in (T,0], \quad \begin{bmatrix}\mathbf{h}(T)\\ \mathbf{a}(T)\\ \mathbf{g}(T)\end{bmatrix} = \begin{bmatrix}\mathbf{y}\\ \nabla_{\mathbf{h}(T)}\mathcal{L}\\ 0\end{bmatrix}$$

Then $\mathbf{g}(0) = \nabla_{\boldsymbol{\theta}}\mathcal{L}$ (Gradient of loss $\mathcal{L}$ w.r.t. parameters)

Reality: Only valid for backward-stable networks!

- Adjoint method: Solve backward ODE for gradients
- **Reversing state equation requires stability** - not always guaranteed
- Generic networks: Backward ODE often numerically unstable

Checkpointing: Standard approach for generic networks

- Store select checkpoints, recompute intermediate values
- Common in computational mathematics for large-scale problems

Optimize-Discretize vs. Discretize-Optimize

$$\min_{\boldsymbol{\theta}} \mathbb{E}\left[\ell(f_{\boldsymbol{\theta}}(\mathbf{y},t),c)\right] + \frac{\alpha}{2}\|\boldsymbol{\theta}\|_2^2,$$

where $\mathbf{y}$ is approximately equal to $\mathbf{h}(T)$ given by

$$\mathbf{h}'(t) = F_{\boldsymbol{\theta}}(\mathbf{h}(t),t), \quad t \in (0,T], \quad \mathbf{h}(0) = \mathbf{h}_0.$$

$O \to D$: Optimize-Discretize (Neural ODE)

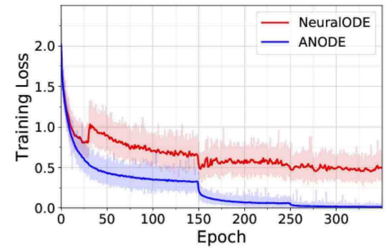
1. keep $\boldsymbol{\theta}, \mathbf{h}$ continuous in time
2. Euler-Lagrange-Equations $\rightsquigarrow$ adjoint equation
3. use adaptive time integrators in optimization

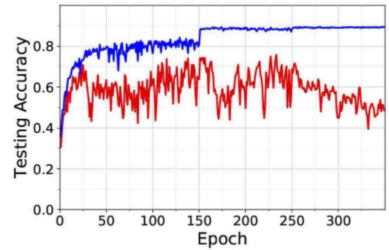
$D \to O$: Discretize-Optimize (ANODE)

1. discretize $\boldsymbol{\theta}, \mathbf{h}$ in time (could use different grids)
2. differentiate discrete problem $\rightsquigarrow$ backpropagation
3. keep discretization fixed during optimization

My advice: use $D \to O$ (accurate gradients, fixed cost per iter, convergence)

Example (image classification)





more examples in (Gholami et al. 2019; Onken and Ruthotto 2020)

Beyond First-Order ODEs: Different ODE Types

$$\frac{d^2\mathbf{h}}{dt^2} = F_{\boldsymbol{\theta}}\left(\mathbf{h}, \frac{d\mathbf{h}}{dt}, t, \boldsymbol{\theta}\right) \quad \rightarrow \quad \mathbf{h}_{n+1} = 2\mathbf{h}_n - \mathbf{h}_{n-1} + (\Delta t)^2 F_{\boldsymbol{\theta}}(\mathbf{h}_n, t_n)$$

Hyperbolic system:

- Wave propagation, conservation laws
- Forward backward stable for $F = -\mathbf{W}^\top \sigma(\mathbf{W}\mathbf{x} + \mathbf{b})$ (weight tying!)
- Reversible

$$\mathbf{h}_{n-1} = 2\mathbf{h}_n - \mathbf{h}_{n+1} + (\Delta t)^2 F_{\boldsymbol{\theta}}(\mathbf{h}_n, t_n)$$

Hamiltonian Networks: Split features $\mathbf{h} = (\mathbf{h}_1, \mathbf{h}_2)$, alternating updates

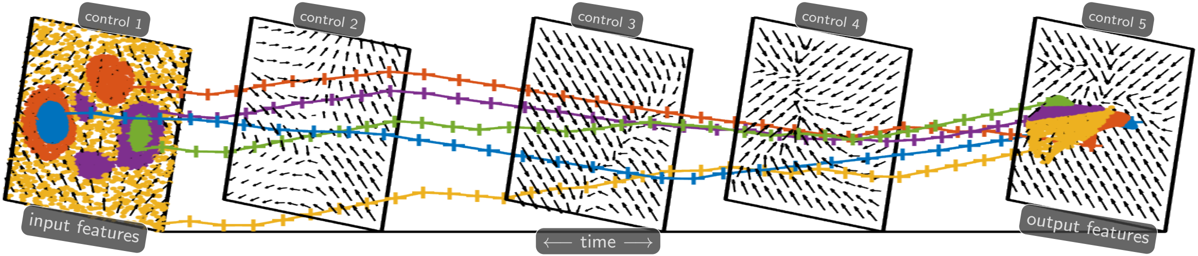
$$\mathbf{h}_1^{n+1} = \mathbf{h}_1^n + \Delta t\, \mathbf{W}^\top \sigma(\mathbf{W}\mathbf{h}_2^n + \mathbf{b}), \quad \mathbf{h}_2^{n+1} = \mathbf{h}_2^n - \Delta t\, \mathbf{W}^\top \sigma(\mathbf{W}\mathbf{h}_1^{n+1} + \mathbf{b})$$

Advantage: Reversible, volume-preserving, stable

Results: accurate image classification with 1202-layer network Chang et al. 2018

These architectures can be trained safely with $O(1)$ memory!

A PDE Perspective of Continuous-Time Learning



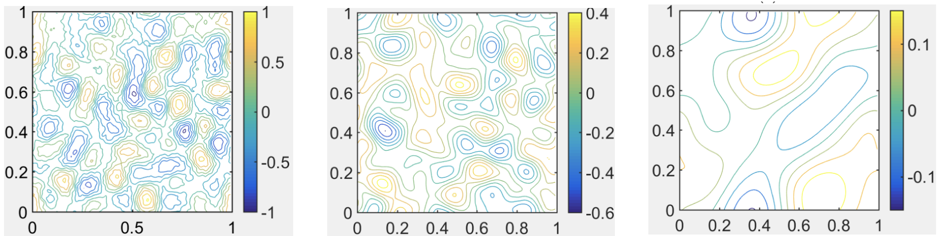
Supervised Deep Learning as PDE-Constrained Optimization

Find network parameters $\boldsymbol{\theta}$ and classification weights $\mathbf{W}$, $\boldsymbol{\mu}$:

$$\begin{aligned} \min_{\boldsymbol{\theta},\mathbf{W},\boldsymbol{\mu}} \quad & \text{loss}[u(\mathbf{x},1),\mathbf{y}] \\ \text{s.t.} \quad & \partial_t u + F_{\boldsymbol{\theta}}(\mathbf{x},t)^\top \nabla u = 0 \\ & u(\mathbf{x},0) = \mathbf{W}\mathbf{x} + \boldsymbol{\mu} \end{aligned}$$

Classification involves transport PDE, add diffusion for robustness Wang et al. 2018

Adding Diffusion for Robustness (Wang et al. 2018)



(a) No diffusion (b) Medium diffusion (c) High diffusion

Transport PDE (deterministic):

$$\partial_t u + F_{\boldsymbol{\theta}}^\top \nabla u = 0$$

- Features follow characteristics
- Single deterministic prediction

Advection-Diffusion (stochastic):

$$\partial_t u + F_{\boldsymbol{\theta}}^\top \nabla u = \frac{\sigma^2}{2} \Delta u$$

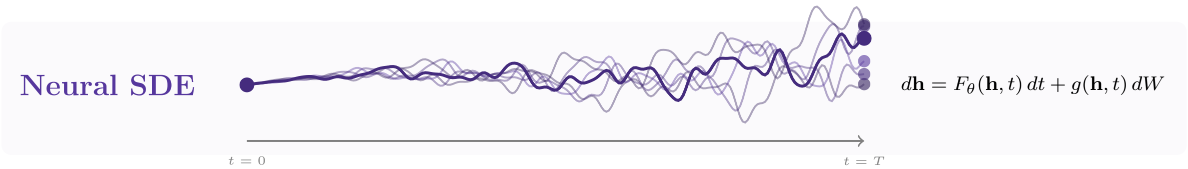
- Features spread/diffuse
- Probabilistic predictions

Feynman-Kac Formula: PDE solution $\leftrightarrow$ SDE expectation

$$u(\mathbf{x}, t) = \mathbb{E}\left[u_0(\mathbf{h}_T)\right], \quad d\mathbf{h} = F_{\boldsymbol{\theta}}(\mathbf{h}, t)\, dt + \sigma\, d\mathbf{W}$$

Key: Noise + averaging $\rightarrow$ smoother probabilistic classification

Neural Stochastic Differential Equations (Neural SDEs)



$$d\mathbf{h} = F_{\boldsymbol{\theta}}(\mathbf{h}, t, \boldsymbol{\theta})\,dt + g(\mathbf{h}, t, \boldsymbol{\theta})\,d\mathbf{W}$$

- f: Deterministic drift (same as Neural ODE)
- g: Stochastic diffusion (controls randomness)
- $d\mathbf{W}$: Brownian motion (random noise)

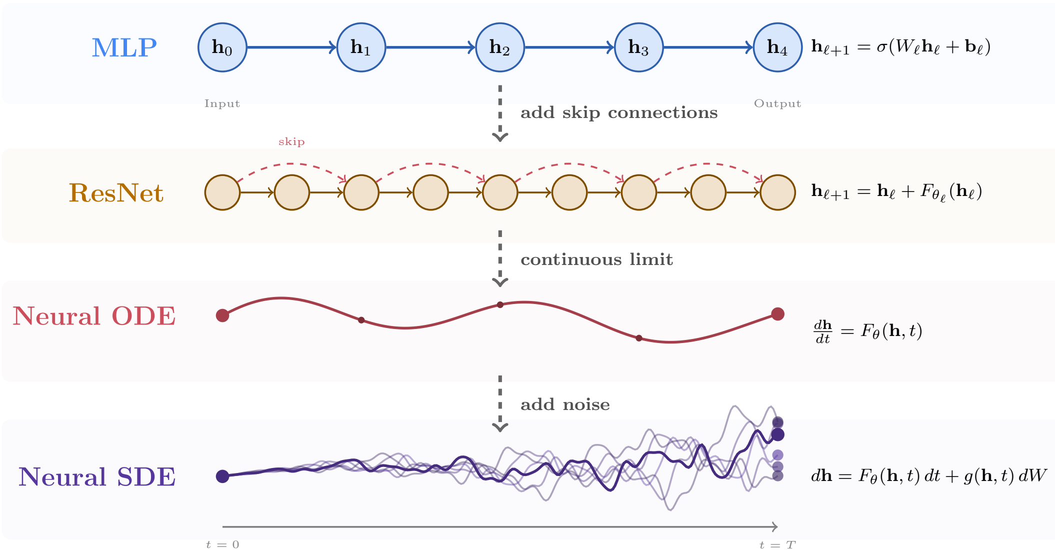
Generalizes Neural ODE:

- $g = 0$ is deterministic ODE, $g > 0 \to$ stochastic trajectories
- Can model uncertainty in dynamics

Applications:

- Financial markets (stock price dynamics), physical systems
- Stochastic optimal control
- Generative modeling (see Lecture 6)

Σ: Features and Depth



take away: Continuous formulations offer flexibility and interpretability

Loss Functions

Some Common Loss Examples

Regression: Mean Squared Error

$$\mathcal{L}_{\text{MSE}} = \mathbb{E}\left[\|\mathbf{y} - f_{\boldsymbol{\theta}}(\mathbf{x})\|^2\right]$$

Empirical approximation:

$$\hat{\mathcal{L}}_{\text{MSE}} = \frac{1}{n}\sum_{i=1}^{n} \|\mathbf{y}_i - f_{\boldsymbol{\theta}}(\mathbf{x}_i)\|^2$$

Example: Autoencoder

$$\mathcal{L} = \mathbb{E}\left[\|\mathbf{x} - D(E(\mathbf{x}))\|^2\right]$$

Classification: Cross-Entropy

$$\mathcal{L}_{\text{CE}} = -\mathbb{E}\left[\mathbf{y}^\top \log(\mathbf{p}(f_{\boldsymbol{\theta}}(\mathbf{x})))\right]$$

where $\mathbf{p}(\mathbf{f}) = \text{softmax}(\mathbf{f})$

Example: Next-token (GPT)

$$\mathcal{L}_{\text{GPT}} = -\mathbb{E}\left[\sum_i \log p(x_{i+1}|x_{1:i})\right]$$

Autoregressive generation

In practice: minimize expected loss or empirical loss

Cross-Entropy as Log-Sum-Exp

Setup: Network outputs logits $\mathbf{f} = f_{\boldsymbol{\theta}}(\mathbf{x}) \in \mathbb{R}^K$ for K classes

Softmax defines class probabilities ($\mathbf{e}$ = all-ones vector):

$$\mathbf{p}(\mathbf{f}) = \frac{\exp(\mathbf{f})}{\mathbf{e}^\top \exp(\mathbf{f})} \in \mathbb{R}^K$$

Cross-entropy loss for sample $(\mathbf{x}, \mathbf{y})$ with one-hot label $\mathbf{y} \in \mathbb{R}^K$:

$$\ell(\mathbf{f}, \mathbf{y}) = -\mathbf{y}^\top \log \mathbf{p}(\mathbf{f}) = -\mathbf{y}^\top \mathbf{f} + \underbrace{\log\left(\mathbf{e}^\top \exp(\mathbf{f})\right)}_{\text{LSE}(\mathbf{f})}$$

Numerical stability: Subtract $m = \max_j f_j$ before computing (LSE is shift-invariant)

Hessian (useful for Gauss-Newton, next lecture):

$$\nabla^2_{\mathbf{f}} \ell = \nabla^2 \text{LSE}(\mathbf{f}) = \text{diag}(\mathbf{p}) - \mathbf{p}\mathbf{p}^\top \succeq 0$$

Key insight: LSE$(\mathbf{f})$ is convex, $-\mathbf{y}^\top \mathbf{f}$ is linear $\Rightarrow \ell(\mathbf{f}, \mathbf{y})$ is convex in $\mathbf{f} = f_{\boldsymbol{\theta}}(\mathbf{x})$

Other Loss Functions: Brief Mentions

Generative Adversarial Networks (GANs):

- $\min_G \max_D \mathbb{E}_{\mathbf{x}}[\log D(\mathbf{x})] + \mathbb{E}_{\mathbf{z}}[\log(1 - D(G(\mathbf{z})))]$
- Minimax game: Generator vs. Discriminator

Variational Autoencoders (VAEs):

- $\mathcal{L} = \mathbb{E}[\|\mathbf{x} - D(E(\mathbf{x}))\|^2] + \mathsf{KL}(q(\mathbf{z}|\mathbf{x})\|p(\mathbf{z}))$
- Reconstruction + regularization to prior

Score-Based / Denoising:

- $\mathcal{L} = \mathbb{E}_{t,\mathbf{x}}\left[\|s_{\boldsymbol{\theta}}(\mathbf{x}_t, t) - \nabla \log p_t(\mathbf{x}_t)\|^2\right]$
- Train score function for generative modeling (Lecture 6)

Physics-Informed Neural Networks (PINNs):

- $\mathcal{L} = \lambda_{\text{data}}\mathbb{E}[\mathcal{L}_{\text{data}}] + \lambda_{\text{physics}}\mathbb{E}[\mathcal{L}_{\text{PDE}}]$
- Blend data fitting with PDE constraints (Lectures 7-8)

Not an exhaustive list. Loss function choice depends on application.

Summary

Lecture 2: Setting up the Learning Problem

$$\mathcal{L}(\boldsymbol{\theta}) = \min_{\boldsymbol{\theta}} \mathbb{E}\left[\ell(f_{\boldsymbol{\theta}}(\mathbf{x}), \mathbf{y})\right]$$

Architecture Choices - Building $f_{\boldsymbol{\theta}}$

Part 1: Layer Structure

- **CNN**: sparse + shared $\mathbf{W}$ (translation)
- **GNN**: graph Laplacian $\mathbf{L}$ (permutation)
- **Transformer**: learned $\mathbf{A}_{\text{attn}}$ (flexible)

Part 2: Depth/Stacking

- **MLP**: finite layers L
- **ResNet**: $\mathbf{h}_{\ell+1} = \mathbf{h}_\ell + F(\mathbf{h}_\ell)$
- **Neural ODE**: $d\mathbf{h}/dt = f(\mathbf{h}, t)$
- **Neural SDE**: add $g(\mathbf{h}, t)d\mathbf{W}$

Loss Function Choices

Expected Loss Minimization

- MSE: $\mathbb{E}[\|\mathbf{Y} - F_{\boldsymbol{\theta}}(\mathbf{X})\|^2]$
- CE: $-\mathbb{E}[\sum_c Y_c \log p_c]$

Examples

- Autoencoder: reconstruction
- GPT: next-token prediction

Other Paradigms

- GANs, VAEs, score-based
- Physics-informed (PINNs)

design choices in F_θ and $\mathcal{L}$ determine what we can learn

Σ: Neural Networks - Looking Forward

Key Insights from Today

1. **Architecture = Encoding Structure**
   - GNN unifies CNNs and Transformers
   - Structure $\rightarrow$ Invariance $\rightarrow$ Generalization
2. **Depth = Discretization Parameter**
   - ResNet $\rightarrow$ Neural ODE $\rightarrow$ Neural SDE
3. **Expected Loss Minimization**
   - All paradigms: $\min_{\boldsymbol{\theta}} \mathbb{E}[\mathcal{L}]$

slido.com

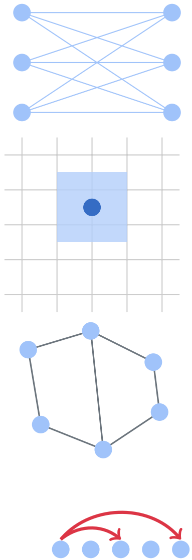
#CBMS25

Bridge to Lecture 3

Today: Design $\min_{\boldsymbol{\theta}} \mathbb{E}[\mathcal{L}(F_{\boldsymbol{\theta}}(\mathbf{X}))]$

Next: How to solve it?

- Automatic differentiation: $\nabla_{\boldsymbol{\theta}}\mathcal{L}$
- Optimization: SGD, Adam



References I

- Bronstein, M. M., J. Bruna, T. Cohen, and P. Veličković (2021). *Geometric Deep Learning: Grids, Groups, Graphs, Geodesics, and Gauges*. arXiv: 2104.13478.
- Chang, B., L. Meng, E. Haber, L. Ruthotto, D. Begert, and E. Holtham (2018). "Reversible Architectures for Arbitrarily Deep Residual Neural Networks". In: *AAAI Conference on Artificial Intelligence*, pp. 2811–2818.
- Cybenko, G. (1989). "Approximation by Superpositions of a Sigmoidal Function". In: *Mathematics of Control, Signals and Systems* 2.4, pp. 303–314.
- E, W. (2017). "A Proposal on Machine Learning via Dynamical Systems". In: *Communications in Mathematics and Statistics* 5.1, pp. 1–11.
- Gholami, A., K. Keutzer, and G. Biros (2019). "ANODE: Unconditionally Accurate Memory-Efficient Gradients for Neural ODEs". In: *International Joint Conference on Artificial Intelligence (IJCAI)*.
- González-García, R., R. Rico-Martínez, and I. G. Kevrekidis (1998). "Identification of Distributed Parameter Systems: A Neural Net Based Approach". In: *Computers & Chemical Engineering* 22, S965–S968.
- Goodfellow, I., Y. Bengio, and A. Courville (2016). *Deep Learning*. MIT Press.

References II

- Haber, E. and L. Ruthotto (2017). "Stable Architectures for Deep Neural Networks". In: *Inverse Problems* 34.1, p. 014004.
- He, K., X. Zhang, S. Ren, and J. Sun (2016). "Deep Residual Learning for Image Recognition". In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778.
- Kidger, Patrick (2022). *On Neural Differential Equations*. arXiv: 2202.02435 [cs.LG]. URL: https://arxiv.org/abs/2202.02435.
- Li, H., Z. Xu, G. Taylor, and Tom Goldstein (2018). "Visualizing the Loss Landscape of Neural Nets". In: *Advances in Neural Information Processing Systems*.
- Onken, D. and L. Ruthotto (2020). "Discretize-Optimize vs. Optimize-Discretize for Time-Series Regression and Continuous Normalizing Flows". In: *arXiv preprint arXiv:2005.13420*.
- Pinkus, Allan (1999). "Approximation Theory of the MLP Model in Neural Networks". In: *Acta Numerica* 8, pp. 143–195.
- Telgarsky, Matus (2016). "Benefits of Depth in Neural Networks". In: *Conference on Learning Theory (COLT)*, pp. 1517–1539.

References III

Vaswani, A., N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin (2017). "Attention Is All You Need". In: *Advances in Neural Information Processing Systems (NeurIPS)*. Vol. 30.

Wang, B., B. Yuan, Z. Shi, and S. J. Osher (2018). *ResNets Ensemble via the Feynman-Kac Formalism to Improve Natural and Robust Accuracies*. arXiv: 1811.10745.