EECE 576L

Advanced Topics in Deep Learning

Lecture 1: Introduction

Renjie Liao

University of British Columbia

2026/27 Winter Session, Term 1 (Fall 2026)

Outline

- **Course information**
- History and applications
- Basic deep learning models
  - MLPs: dense layers and nonlinearities
  - CNNs: local filters and spatial sharing
  - RNNs: recurrence and temporal sharing
- Learning from data
- Data structure vs. model inductive bias
- Research practice

Course Overview

- Graduate study of geometric deep learning, generative models, and reasoning
- Connect mathematical ideas to modern architectures and research papers
- Prerequisites: calculus, linear algebra, probability, and basic deep learning
- Programming: Python and experience with PyTorch or a similar framework
- Research practice: critical reading, reproducible experiments, and clear writing

Course Information

- EECE 576L: Advanced Topics in Deep Learning
- Lectures: Tuesday, 4:00–7:00 p.m.
- UBCV, Hector J. MacLeod Building (MCLD), Floor 2, Room 2012
- Office hour: Tuesday, 1:00–2:00 p.m., KAIS 3047 (Ohm)
- Instructor: Renjie Liao, renjie.liao@ubc.ca
- TA: Yuanpei Gao, yuanpeig@student.ubc.ca

Course Communication and Calendar

- Website: EECE 576L syllabus and assessment guidelines
- Piazza: piazza.com/ubc.ca/winterterm12026/eece576l
- Canvas: announcements, submissions, grades, and confirmed deadlines
- First class: September 15. No class November 10 (midterm break).
- Last meeting and project presentations: December 1

Assessment

- 30% Paper presentation
- 10% Project proposal
- 10% Project presentation
- 10% Presentation peer review
- 40% Final project report and code
- Participation: up to 3 extra percentage points

Paper Presentations

- 25 papers, normally one presenter per paper
- If enrollment exceeds 25, two students may jointly present a paper
- Both presenters contribute and receive individual grades
- 20–25 min presentation + remaining 5–10 min Q&A (30 min per paper)
- Five papers per meeting: 150 min + 10 min for transitions and overflow
- A separate 20 min break after the second presentation
- For long papers, agree a focused reading scope with the instructor

Paper presentation guidelines

Course Project

- Work individually or in groups of up to four students
- A larger group is expected to complete more work
- Proposal: 2–4 pages. Final report: 6–8 pages, plus code
- Use the NeurIPS report format. Deadlines will be announced on Canvas.
- Full marks require a novel contribution, supported by careful evidence

Project scope, deliverables, and policies

AI Tools and Research Integrity

- AI tools may assist with reading, writing, slides, and code
- Disclose the tools used and explain their contribution
- Cite the original papers, code, data, and other sources
- Check generated citations and technical claims against the originals
- You must be able to explain your methods, code, and results

Planning a Feasible Research Project

- Start with a precise question and a strong, affordable baseline
- List the compute and datasets you can actually access
- Run a small pilot before committing to a large experiment
- Use pretrained models, smaller datasets, or efficient adaptation when useful
- Keep checkpoints and a fallback plan. Free GPU availability can vary.

Course Roadmap

- Geometric deep learning: sets, sequences, graphs, and symmetry
- Language models and post-training: autoregression, preferences, and reasoning RL
- Generative models: diffusion, score-based methods, and flows
- Embodied models: vision-language-action and world action models
- Research papers and a project connect these ideas to current work

Outline

- Course information
- **History and applications**
- Basic deep learning models
  - MLPs: dense layers and nonlinearities
  - CNNs: local filters and spatial sharing
  - RNNs: recurrence and temporal sharing
- Learning from data
- Data structure vs. model inductive bias
- Research practice

What is Deep Learning?

- Deep learning uses multi-layer neural networks to learn representations from data.
- Data: labeled and unlabeled examples, including text, images, audio, and video
- Models: neural networks such as CNNs and transformers
- Learning: back-propagation computes gradients. Optimizers such as SGD and AdamW use these gradients to update parameters.

History of Deep Learning: Early Ideas

- Artificial neurons (McCulloch and Pitts, 1943)
- Hebbian learning: cells that fire together wire together (Hebb, 1949)
- Perceptron (Rosenblatt, 1958)
- Orientation-selective receptive fields and cortical orientation columns (Hubel and Wiesel, 1959–1962)
- Neocognitron, an early convolutional architecture (Fukushima, 1979/1980)

History of Deep Learning: Learning Algorithms

- Hopfield networks (Hopfield, 1982)
- Boltzmann machines (Hinton and Sejnowski, 1983)
- Back-propagation: Linnainmaa (1970), Werbos (1974), and Rumelhart, Hinton and Williams (1986)
- Back-propagation for handwritten recognition with CNNs (LeCun et al., 1989)
- Long short-term memory (Hochreiter and Schmidhuber, 1997)

Rumelhart, Hinton & Williams (1986), Learning representations by back-propagating errors

History of Deep Learning: Major Breakthroughs

- Deep belief networks (Hinton et al., 2006)
- Deep speech recognition (Dahl et al., 2011/2012)
- Computer vision: AlexNet (2012) and ResNet (2015/2016)
- Games: DQN for Atari (Mnih et al., 2015) and AlphaGo (2016)
- Language: Seq2seq (2014), Transformers (2017), and GPT-3 (2020)
- Protein structure: AlphaFold2 at CASP14 (2020), published in 2021

History of Deep Learning: Recent Breakthroughs

- Diffusion for image generation: DDPM (2020) and latent diffusion (2022)
- Conversational, instruction-following models: ChatGPT (2022)
- Biomolecular interactions: AlphaFold 3 (2024)
- Reasoning: DeepSeek-R1 and Gemini Deep Think's IMO gold-medal performance (2025)

DeepSeek-AI et al. (2025), DeepSeek-R1

A Researcher's Perspective

"The future depends on some graduate student who is deeply suspicious of everything I have said."

---- Geoffrey Hinton

Are you suspicious of today's AI?

Applications of Deep Learning

Multimodal Assistants: Gemini and OpenAI

Gemini Live: visual guidance (2026)

OpenAI: camera and voice (2024)

Play OpenAI demo: Rock Paper Scissors

Demo: Google / Android, Gemini Live (September 2026)
Video and image: OpenAI, GPT-4o (May 2024)

Applications of Deep Learning

Interactive World Models (Project Genie, 2026)

Generate and explore worlds from text and images.

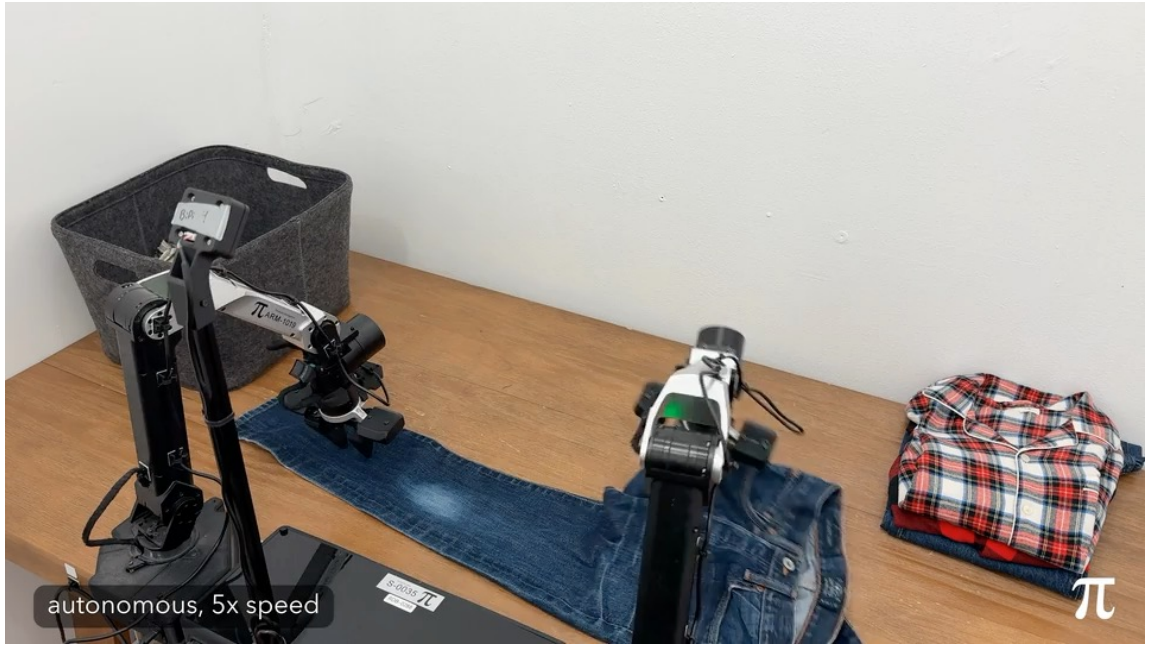
The model predicts what happens as the user moves.

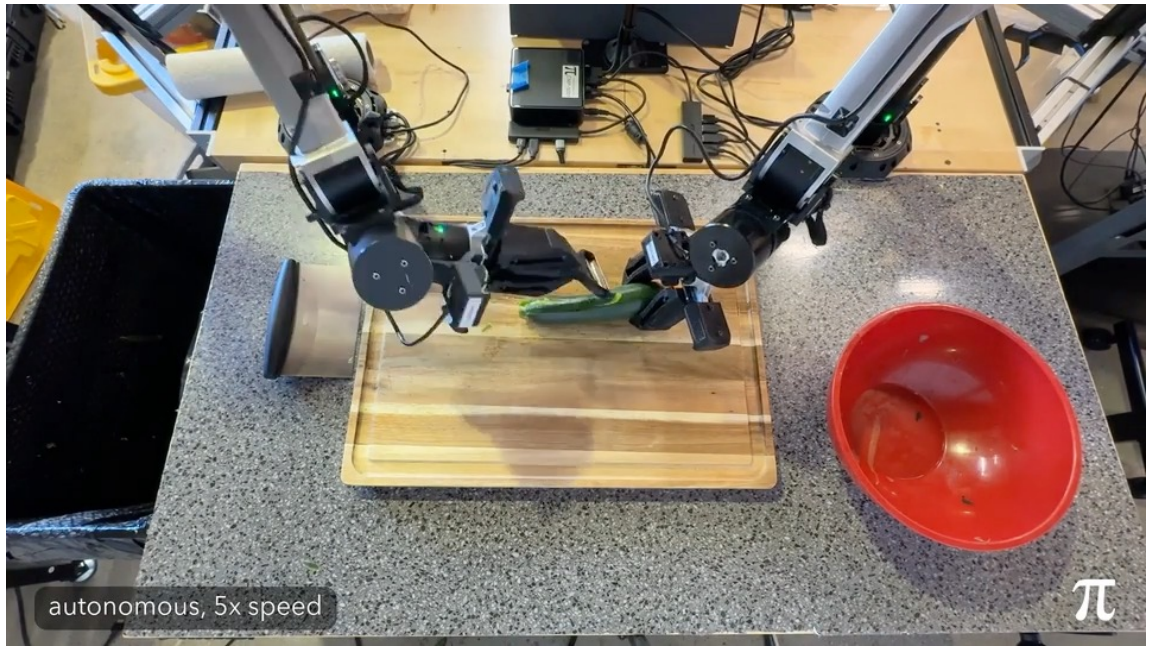
Experimental worlds can still be physically inconsistent.

Demo: Google DeepMind / Google Labs, Project Genie (2026), powered by Genie 3 (2025)

Applications of Deep Learning

Robotics (Physical Intelligence $\pi_{0.7}$, 2026)





One vision-language-action model controls diverse manipulation tasks.

Click either image to play its video.

Video: Physical Intelligence, $\pi_{0.7}$ (April 2026), folding jeans
Video: Physical Intelligence, $\pi_{0.7}$ (April 2026), peeling zucchini

Applications of Deep Learning

Self-Driving Cars (Waymo, 2026)

Perception and planning in London traffic.

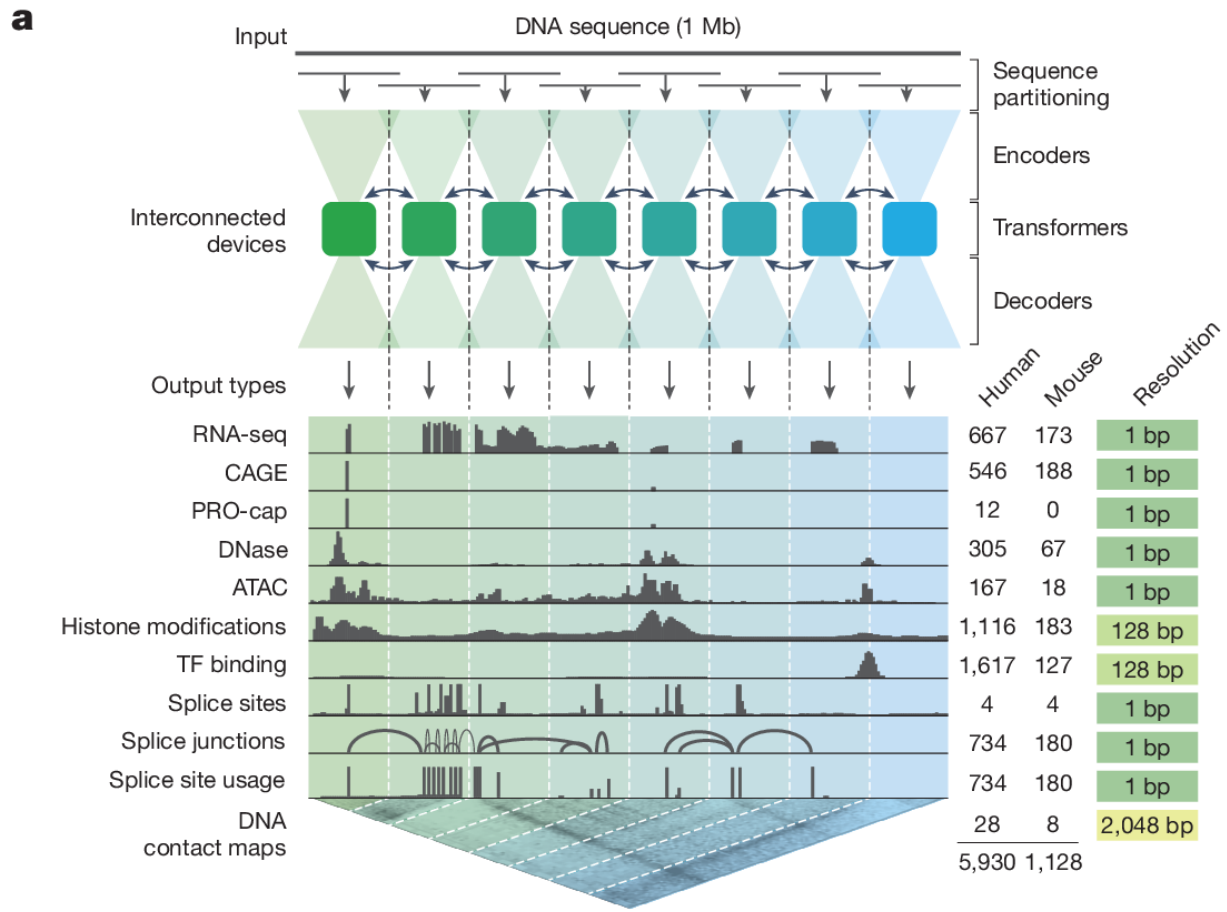
Testing with trained specialists behind the wheel.

Play full video

Source and video: Waymo, London autonomous testing (April 2026).

Applications of Deep Learning

Genome Interpretation (AlphaGenome, 2026)



Predict gene activity and other molecular signals from one million DNA letters.

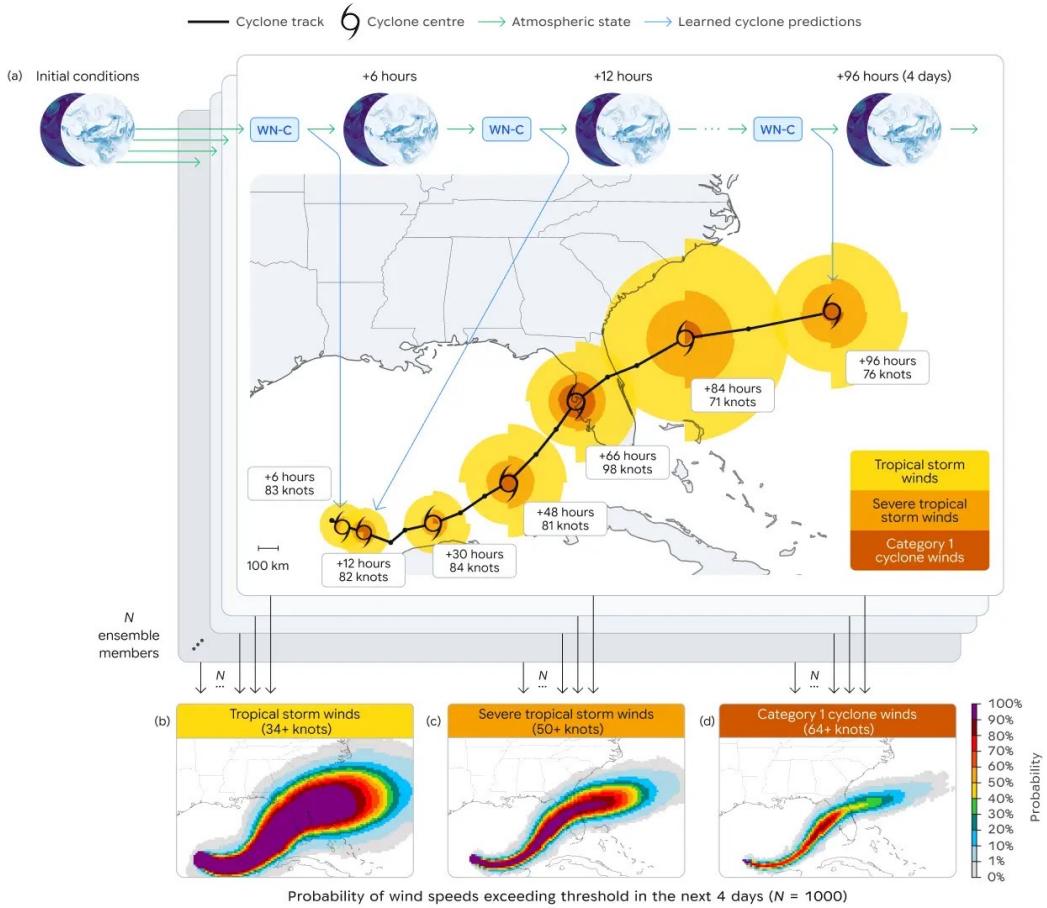
AlphaGenome Atlas (Sept. 2026) precomputes predicted effects for 9 billion single-letter variants.

Research predictions help prioritize biological experiments.

Figure: Avsec et al., Nature (2026), Fig. 1a (cropped), CC BY 4.0
Atlas update: Google DeepMind (8 September 2026)

Applications of Deep Learning

Weather Forecasting (WeatherNext Cyclones, 2026)



Predict cyclone paths, intensity and wind structure together.

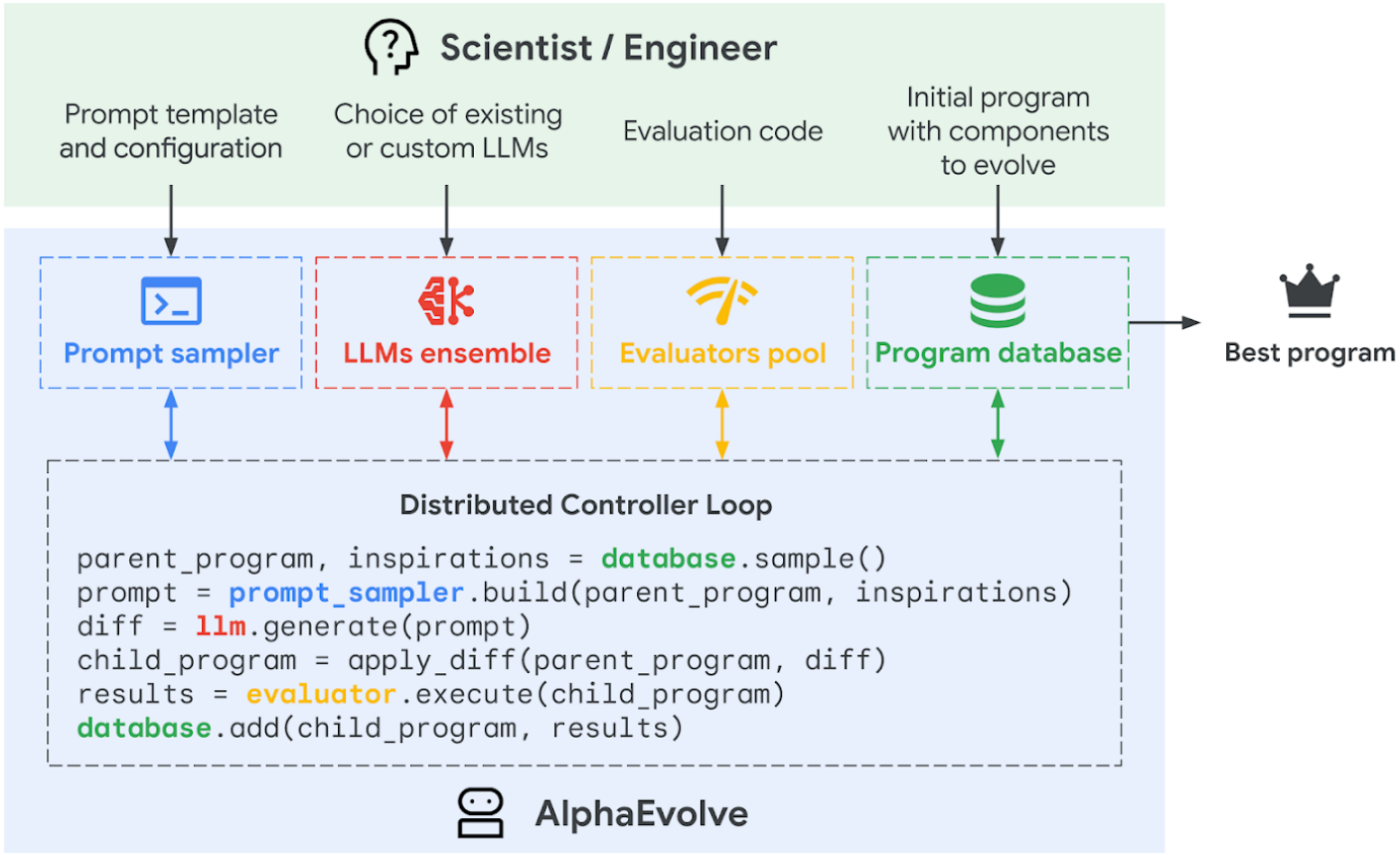
Generate many possible forecasts to estimate uncertainty.

These forecasts provide guidance for human forecasters.

Figure: Google DeepMind / WeatherNext team (August 2026). Alet et al., Nature

Applications of Deep Learning

Algorithm Discovery (AlphaEvolve, 2026)



Language models propose programs. Automated tests guide the search.

2026 applications include DNA sequencing and power-grid optimization.

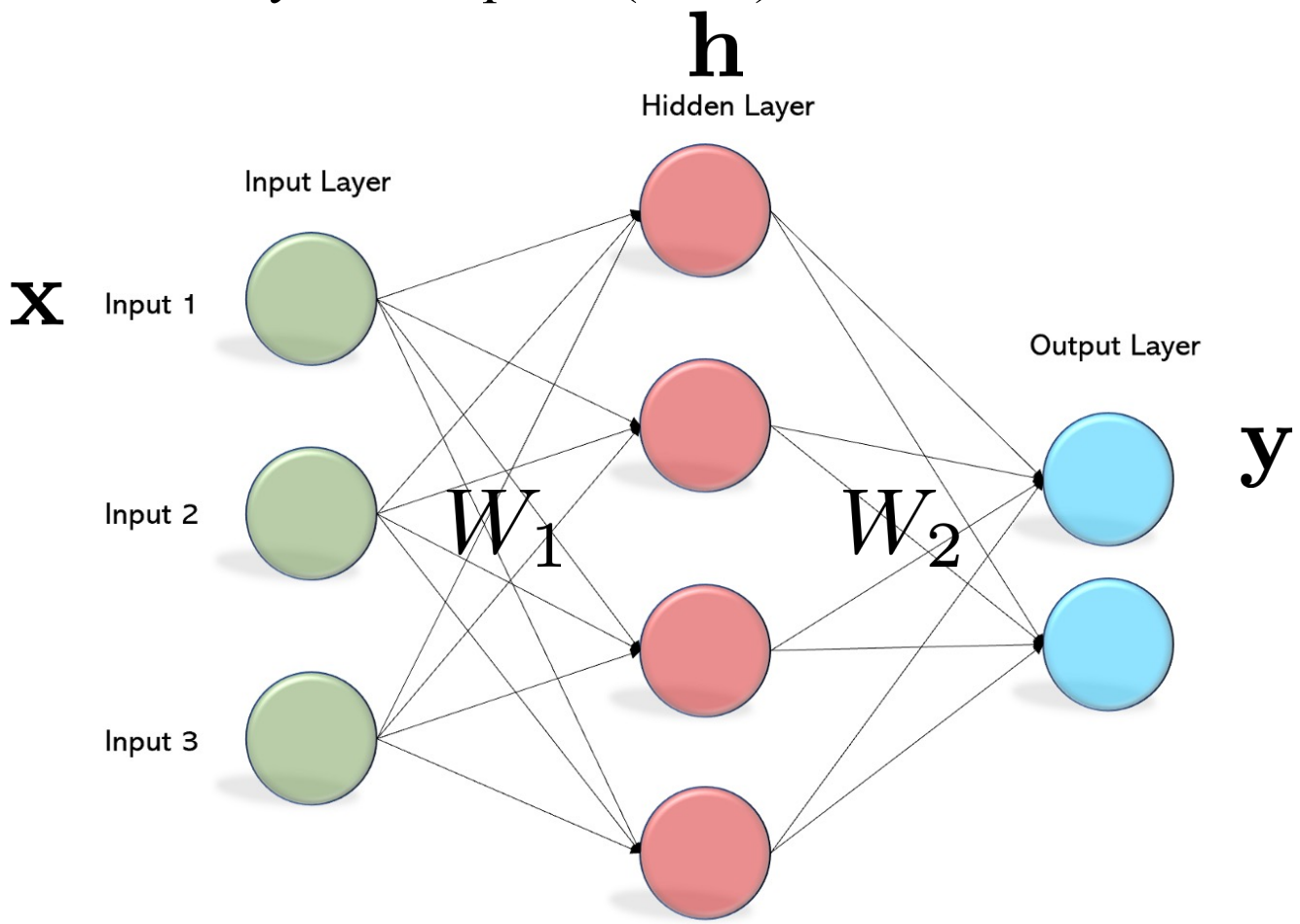
Diagram: Google DeepMind, AlphaEvolve (May 2025)
Applications update: Google DeepMind (May 2026)

Outline

- Course information
- History and applications
- Basic deep learning models
  - **MLPs: dense layers and nonlinearities**
  - CNNs: local filters and spatial sharing
  - RNNs: recurrence and temporal sharing
- Learning from data
- Data structure vs. model inductive bias
- Research practice

Basic Deep Learning Models

Multi-Layer Perceptron (MLP)



$$\mathbf{h} = \sigma(W_1\mathbf{x})$$

$$\mathbf{y} = W_2\mathbf{h}$$

ReLU: $\sigma(\mathbf{h}) = \max(\mathbf{h}, 0)$

Sigmoid: $\sigma(\mathbf{h}) = \dfrac{1}{1+\exp(-\mathbf{h})}$

Tanh, Softplus, ELU, …

Image Credit: https://becominghuman.ai/multi-layer-perceptron-mlp-models-on-real-world-banking-data-f6dd3d7e998f

MLP: Forward Computation

Column-vector convention: input $x \in \mathbb{R}^d$, hidden width m, C output classes

$$a_1 = W_1 x + b_1, \qquad h = \mathrm{ReLU}(a_1)$$

$$z = W_2 h + b_2, \qquad p = \mathrm{softmax}(z)$$

$$W_1 \in \mathbb{R}^{m \times d}, \quad b_1 \in \mathbb{R}^m; \qquad W_2 \in \mathbb{R}^{C \times m}, \quad b_2 \in \mathbb{R}^C$$

$\mathrm{ReLU}(a) = \max(0, a)$, applied elementwise

Without nonlinearities, stacked affine layers reduce to one affine map

Outline

- Course information
- History and applications
- Basic deep learning models
  - MLPs: dense layers and nonlinearities
  - **CNNs: local filters and spatial sharing**
  - RNNs: recurrence and temporal sharing
- Learning from data
- Data structure vs. model inductive bias
- Research practice

Basic Deep Learning Models

Convolutional Neural Network (CNN)

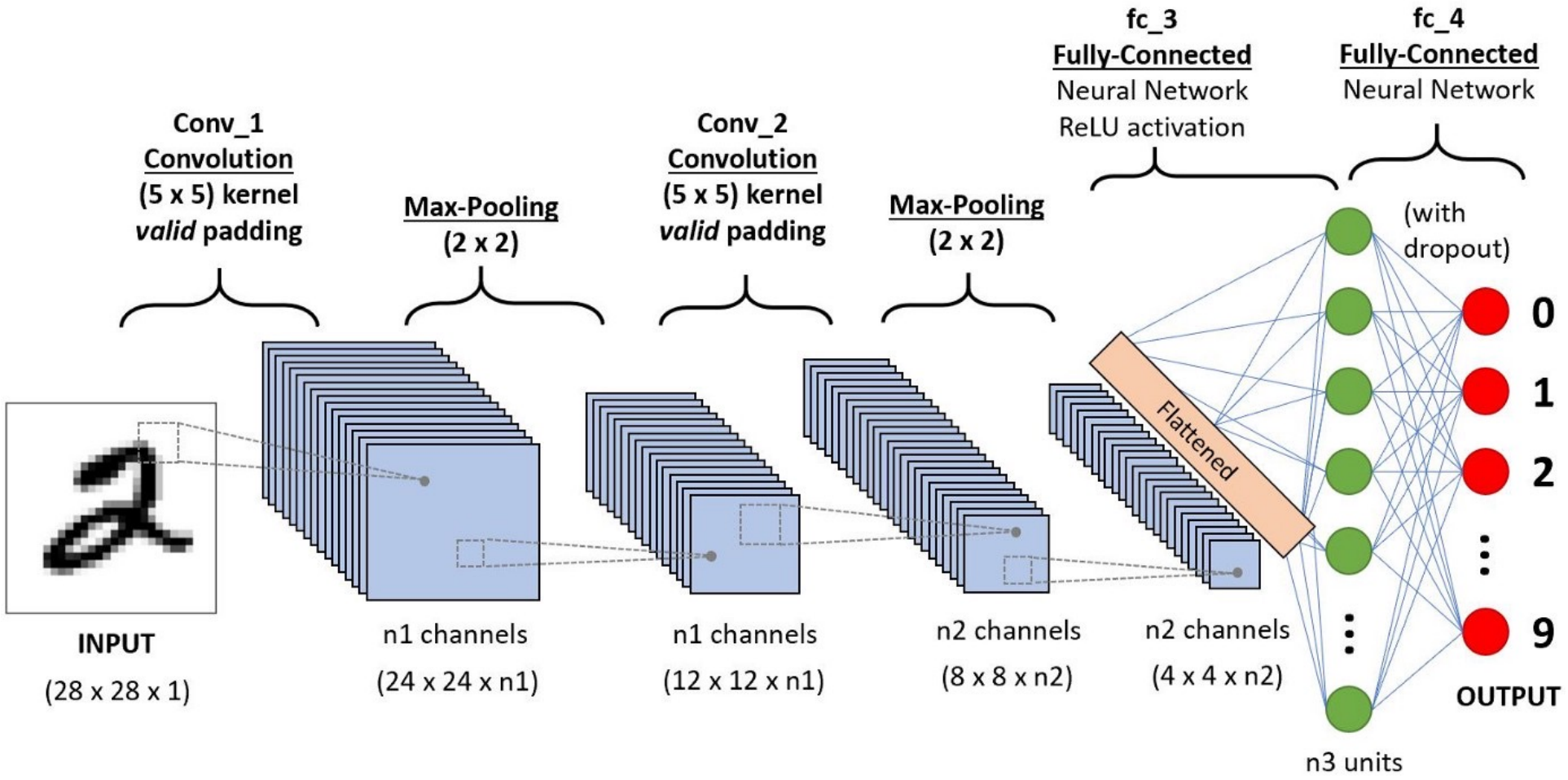
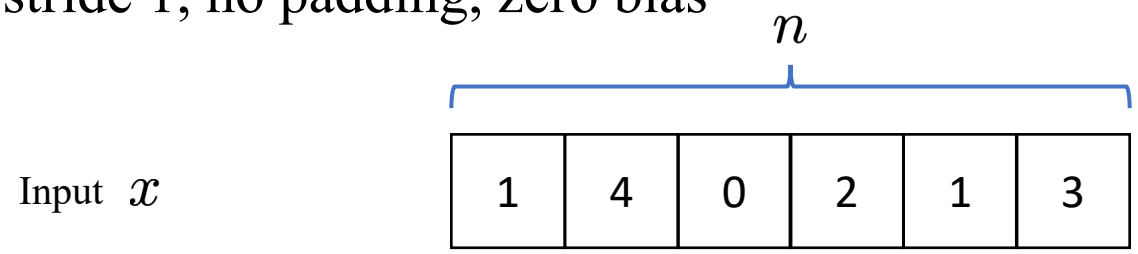


Image Credit: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

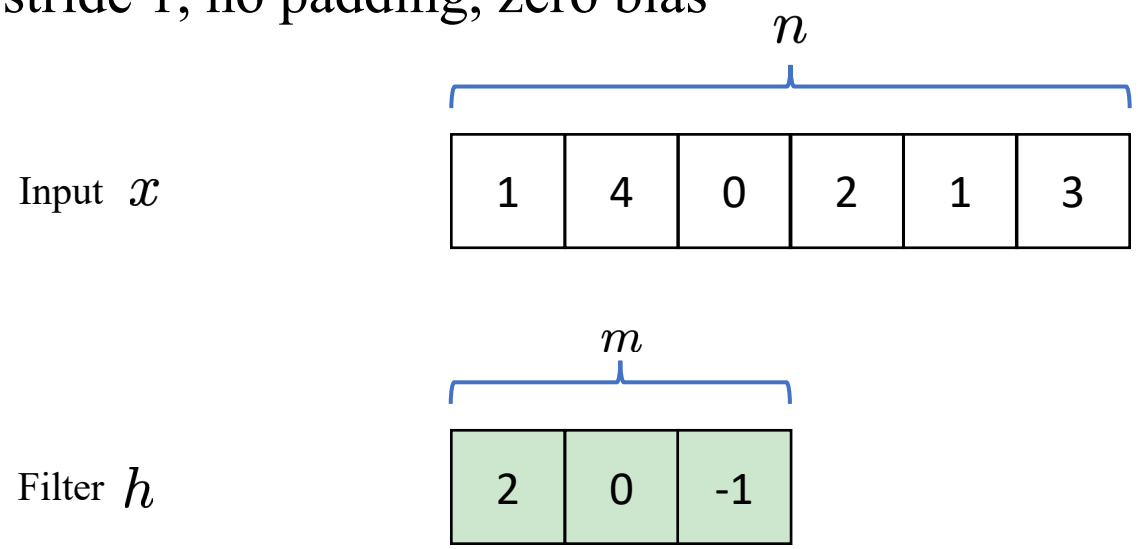
CNN: A Sliding Filter

Cross-correlation: stride 1, no padding, zero bias



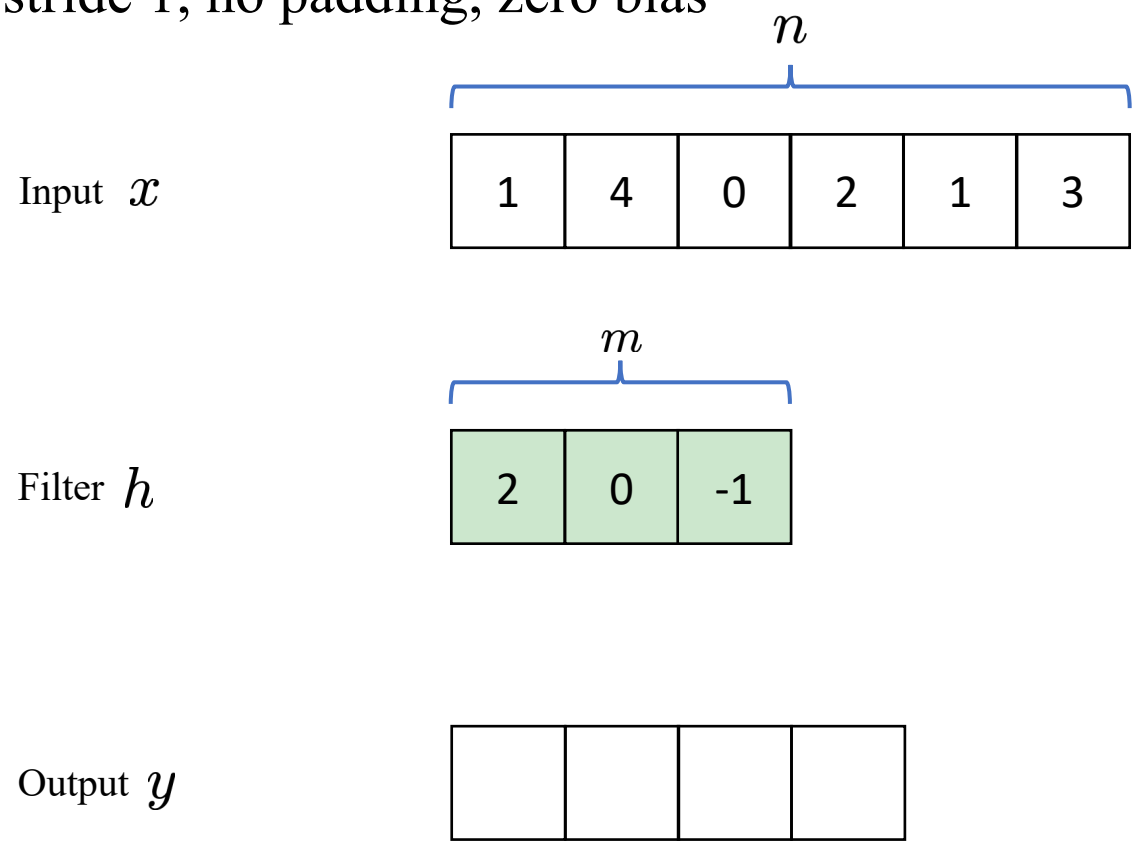
CNN: A Sliding Filter

Cross-correlation: stride 1, no padding, zero bias



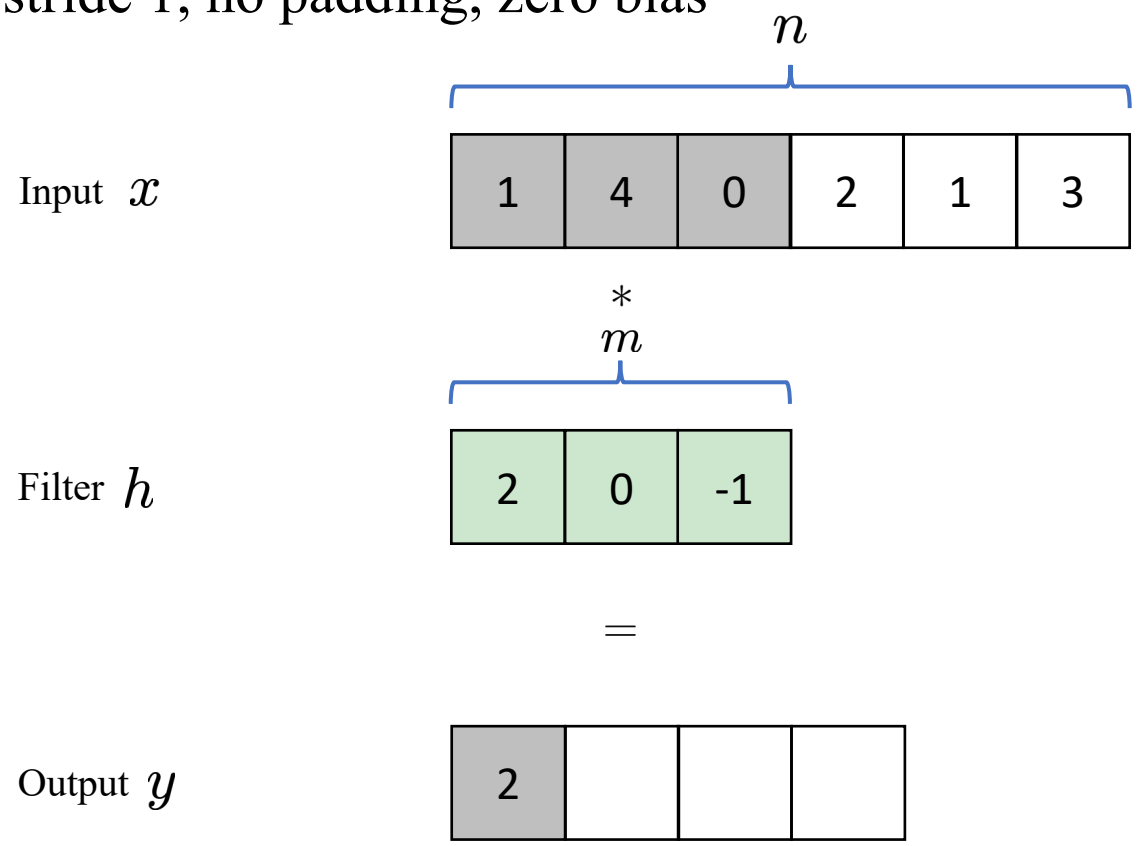
CNN: A Sliding Filter

Cross-correlation: stride 1, no padding, zero bias



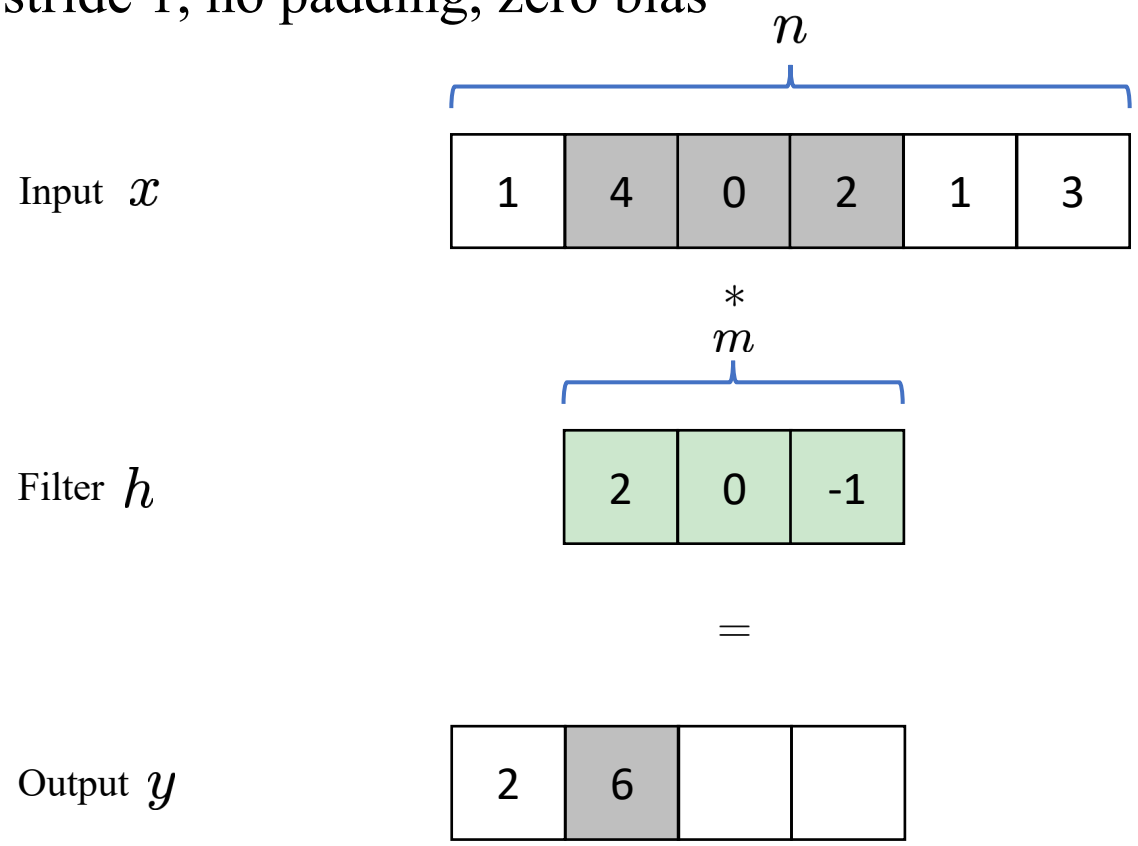
CNN: A Sliding Filter

Cross-correlation: stride 1, no padding, zero bias



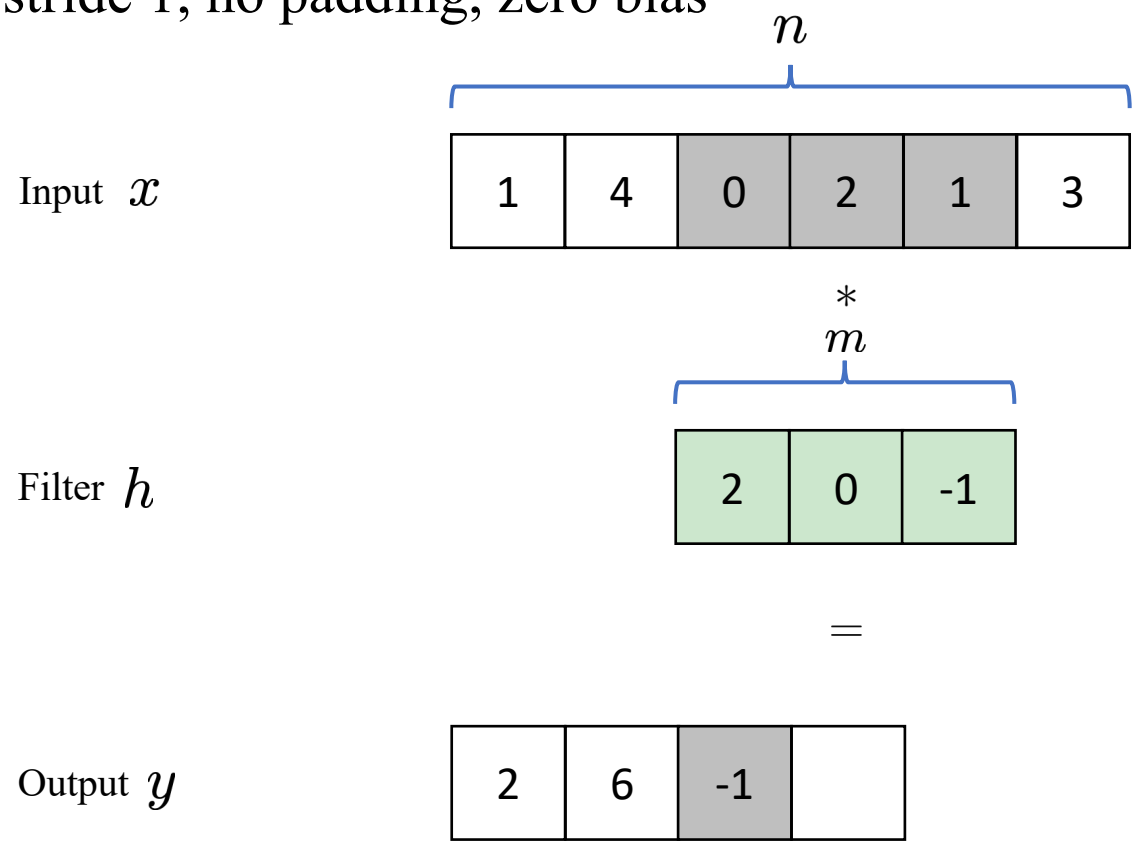
CNN: A Sliding Filter

Cross-correlation: stride 1, no padding, zero bias



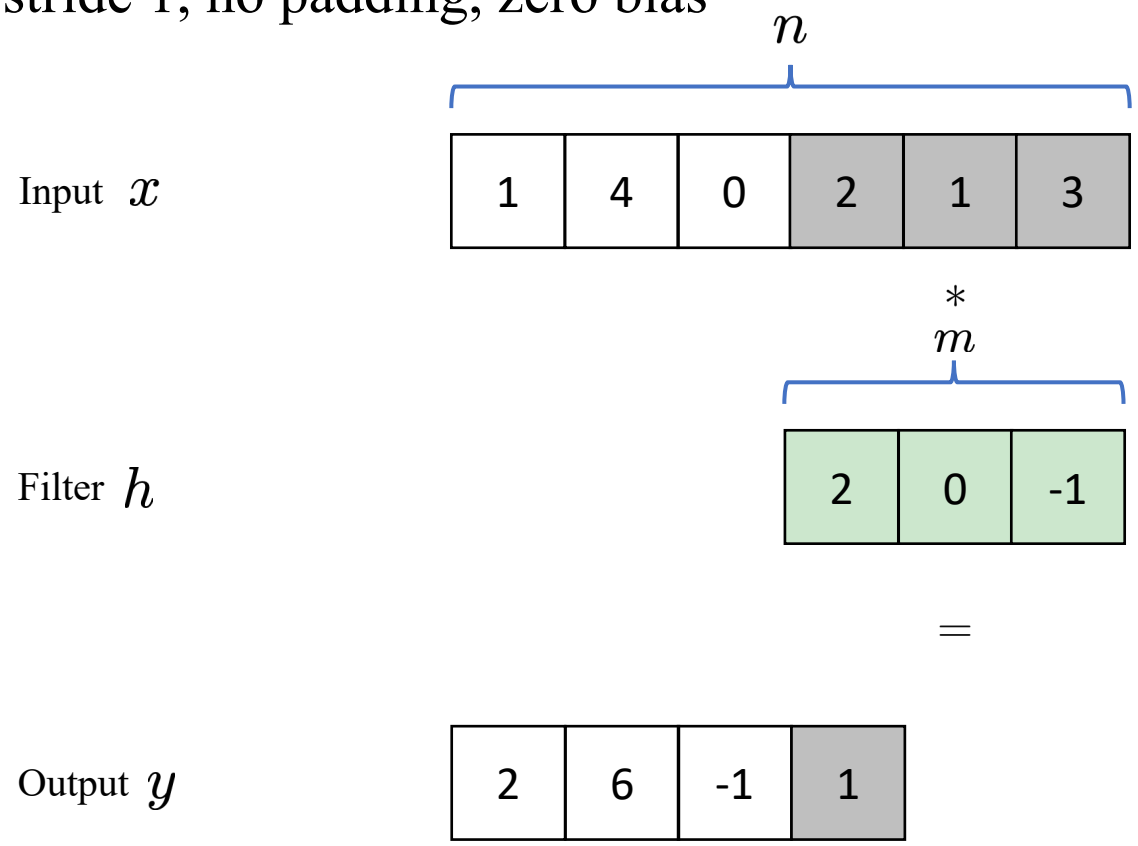
CNN: A Sliding Filter

Cross-correlation: stride 1, no padding, zero bias



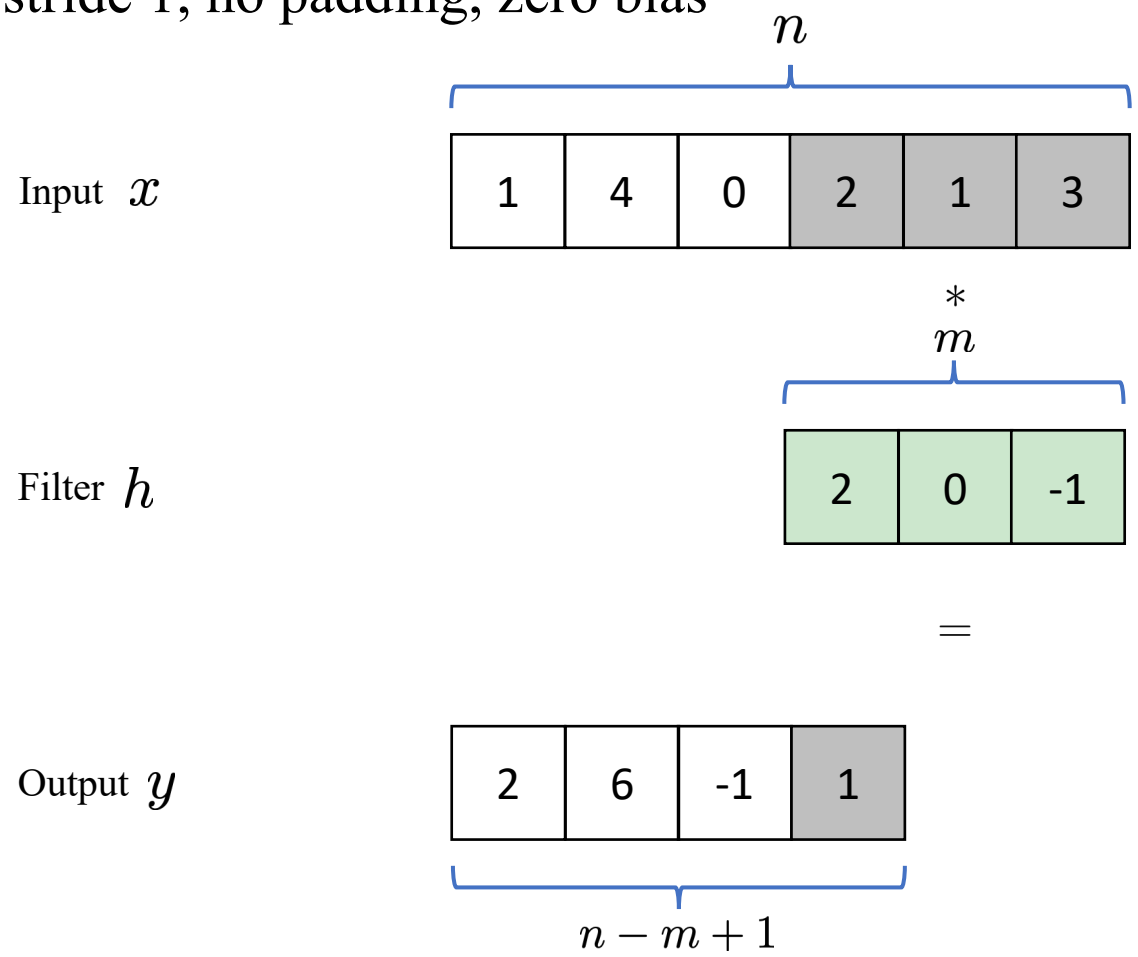
CNN: A Sliding Filter

Cross-correlation: stride 1, no padding, zero bias



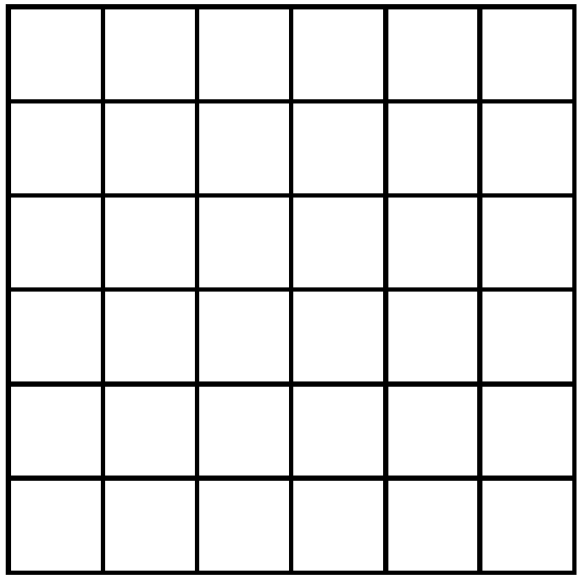
CNN: A Sliding Filter

Cross-correlation: stride 1, no padding, zero bias



CNN: Two Spatial Dimensions

A shared filter slides across height and width



$$y_{i,j} = \sum_{m=1}^{K} \sum_{n=1}^{K} W_{m,n}\, x_{i+m-1,j+n-1}$$

- Each output is one local dot product.
- The animation uses stride 1 and no padding.
- The same filter weights apply at every location.

Animation: valid cross-correlation. DL libraries call this operation convolution.

CNN: Resolution and Parameters

For one spatial axis, dilation 1, symmetric padding p, and stride s:

$$L_{\text{out}} = \left\lfloor \frac{L_{\text{in}} + 2p - k}{s} \right\rfloor + 1$$

A 2D layer has C_{in} input channels and C_{out} output channels

number of parameters $= k_h k_w C_{\text{in}} C_{\text{out}} + C_{\text{out}}$

The last term counts biases. Parameter count does not depend on image size

Example: 3×3 filters, 3 input channels, 16 output channels $\rightarrow 448$ parameters

PyTorch Conv2d: cross-correlation, shapes, and channel conventions

CNN: Matrix Multiplication View I

1D cross-correlation as matrix multiplication

$$y = h * x = \begin{bmatrix} h_{\lfloor m/2 \rfloor+1} & h_{\lfloor m/2 \rfloor+2} & \cdots & h_m & 0 & \cdots & \cdots & \cdots & \cdots & 0 \\ h_{\lfloor m/2 \rfloor} & h_{\lfloor m/2 \rfloor+1} & \cdots & h_{m-1} & h_m & 0 & \cdots & \cdots & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ h_1 & h_2 & \cdots & \cdots & \cdots & \cdots & h_m & 0 & \cdots & 0 \\ 0 & h_1 & h_2 & \cdots & \cdots & \cdots & \cdots & h_m & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \cdots & \cdots & \cdots & 0 & h_1 & h_2 & \cdots & h_{\lfloor m/2 \rfloor+2} \\ 0 & 0 & \cdots & \cdots & & \cdots\cdots & 0 & h_1 & \cdots & h_{\lfloor m/2 \rfloor+1} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{bmatrix}$$

Zero padding keeps the output length equal to the input length

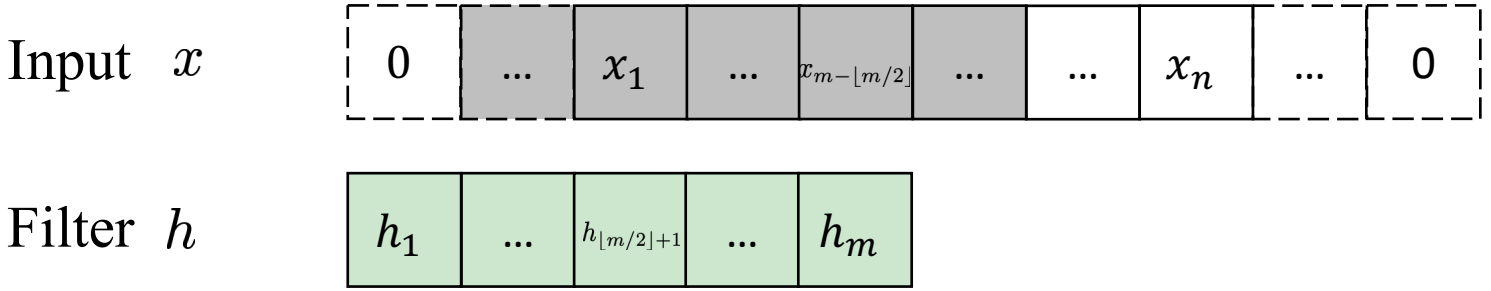
Odd filter length, stride 1, zero padding. DL libraries call this operation convolution.

CNN: Matrix Multiplication View I

1D cross-correlation as matrix multiplication

$$y = h * x = \begin{bmatrix} h_{\lfloor m/2 \rfloor+1} & h_{\lfloor m/2 \rfloor+2} & \cdots & h_m & 0 & \cdots & \cdots & \cdots & \cdots & 0 \\ h_{\lfloor m/2 \rfloor} & h_{\lfloor m/2 \rfloor+1} & \cdots & h_{m-1} & h_m & 0 & \cdots & \cdots & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ h_1 & h_2 & \cdots & \cdots & \cdots & \cdots & h_m & 0 & \cdots & 0 \\ 0 & h_1 & h_2 & \cdots & \cdots & \cdots & \cdots & h_m & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \cdots & \cdots & \cdots & 0 & h_1 & h_2 & \cdots & h_{\lfloor m/2 \rfloor+2} \\ 0 & 0 & \cdots & \cdots & & \cdots\cdots & 0 & h_1 & \cdots & h_{\lfloor m/2 \rfloor+1} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{bmatrix}$$

Zero padding keeps the output length equal to the input length



Omit boundary outputs to recover the preceding unpadded example

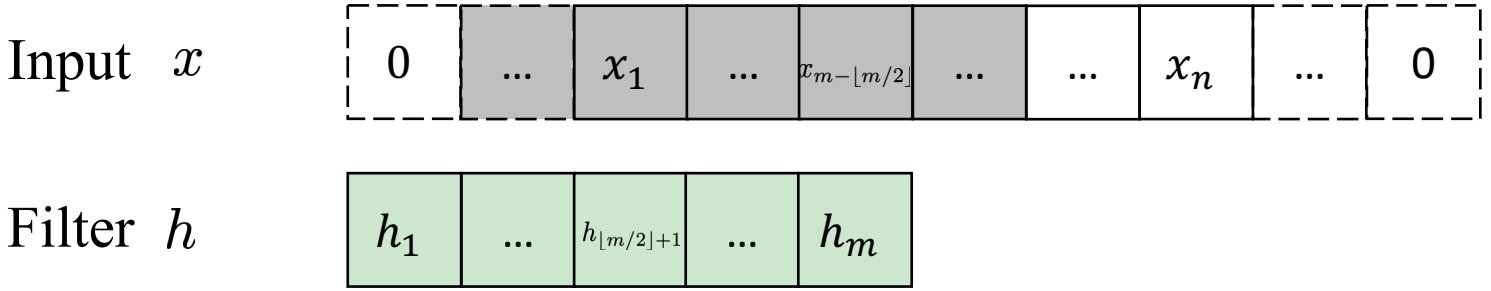
Odd filter length, stride 1, zero padding. DL libraries call this operation convolution.

CNN: Matrix Multiplication View I

1D cross-correlation as matrix multiplication

$$y = h * x = \begin{bmatrix} h_{\lfloor m/2 \rfloor+1} & h_{\lfloor m/2 \rfloor+2} & \cdots & h_m & 0 & \cdots & \cdots & \cdots & \cdots & 0 \\ h_{\lfloor m/2 \rfloor} & h_{\lfloor m/2 \rfloor+1} & \cdots & h_{m-1} & h_m & 0 & \cdots & \cdots & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ h_1 & h_2 & \cdots & \cdots & \cdots & \cdots & h_m & 0 & \cdots & 0 \\ 0 & h_1 & h_2 & \cdots & \cdots & \cdots & \cdots & h_m & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \cdots & \cdots & \cdots & 0 & h_1 & h_2 & \cdots & h_{\lfloor m/2 \rfloor+2} \\ 0 & 0 & \cdots & \cdots & & \cdots\cdots & 0 & h_1 & \cdots & h_{\lfloor m/2 \rfloor+1} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{bmatrix}$$

Zero padding keeps the output length equal to the input length



Omit boundary outputs to recover the preceding unpadded example

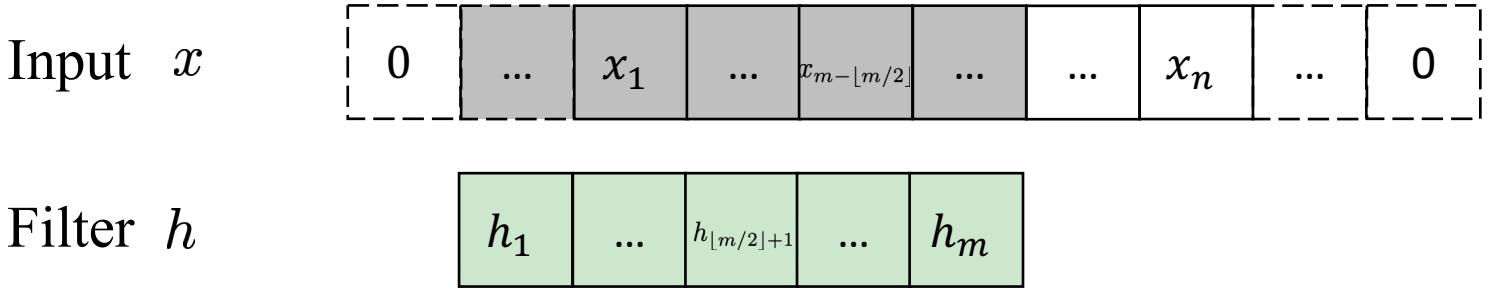
Odd filter length, stride 1, zero padding. DL libraries call this operation convolution.

CNN: Matrix Multiplication View I

1D cross-correlation as matrix multiplication

$$y = h * x = \begin{bmatrix} h_{\lfloor m/2 \rfloor+1} & h_{\lfloor m/2 \rfloor+2} & \cdots & h_m & 0 & \cdots & \cdots & \cdots & \cdots & 0 \\ h_{\lfloor m/2 \rfloor} & h_{\lfloor m/2 \rfloor+1} & \cdots & h_{m-1} & h_m & 0 & \cdots & \cdots & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ h_1 & h_2 & \cdots & \cdots & \cdots & \cdots & h_m & 0 & \cdots & 0 \\ 0 & h_1 & h_2 & \cdots & \cdots & \cdots & \cdots & h_m & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \cdots & \cdots & \cdots & 0 & h_1 & h_2 & \cdots & h_{\lfloor m/2 \rfloor+2} \\ 0 & 0 & \cdots & \cdots & & \cdots\cdots & 0 & h_1 & \cdots & h_{\lfloor m/2 \rfloor+1} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{bmatrix}$$

Zero padding keeps the output length equal to the input length



Omit boundary outputs to recover the preceding unpadded example

Odd filter length, stride 1, zero padding. DL libraries call this operation convolution.

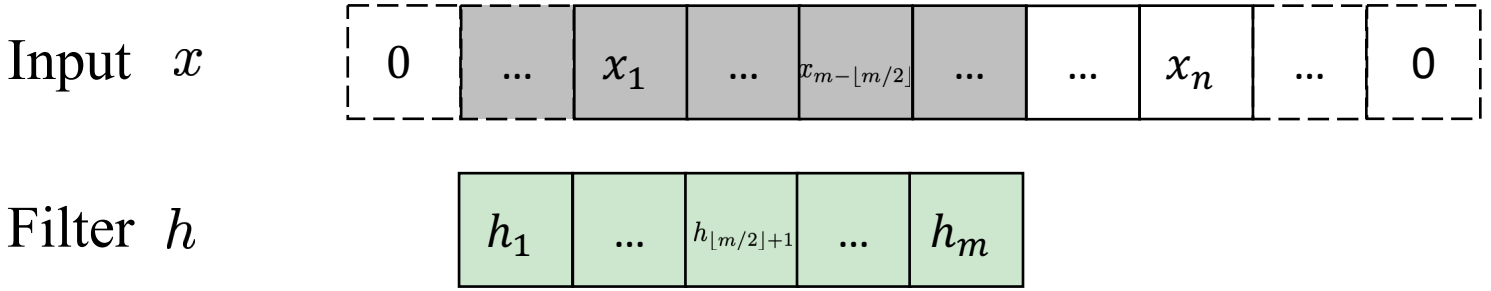
CNN: Matrix Multiplication View I

1D cross-correlation as matrix multiplication

Shared filter weights form a sparse Toeplitz matrix

$$y = h * x = \begin{bmatrix} h_{\lfloor m/2 \rfloor+1} & h_{\lfloor m/2 \rfloor+2} & \cdots & h_m & 0 & \cdots & \cdots & \cdots & \cdots & 0 \\ h_{\lfloor m/2 \rfloor} & h_{\lfloor m/2 \rfloor+1} & \cdots & h_{m-1} & h_m & 0 & \cdots & \cdots & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ h_1 & h_2 & \cdots & \cdots & \cdots & \cdots & h_m & 0 & \cdots & 0 \\ 0 & h_1 & h_2 & \cdots & \cdots & \cdots & \cdots & h_m & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \cdots & \cdots & \cdots & 0 & h_1 & h_2 & \cdots & h_{\lfloor m/2 \rfloor+2} \\ 0 & 0 & \cdots & \cdots & & \cdots\cdots & 0 & h_1 & \cdots & h_{\lfloor m/2 \rfloor+1} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{bmatrix}$$

Zero padding keeps the output length equal to the input length



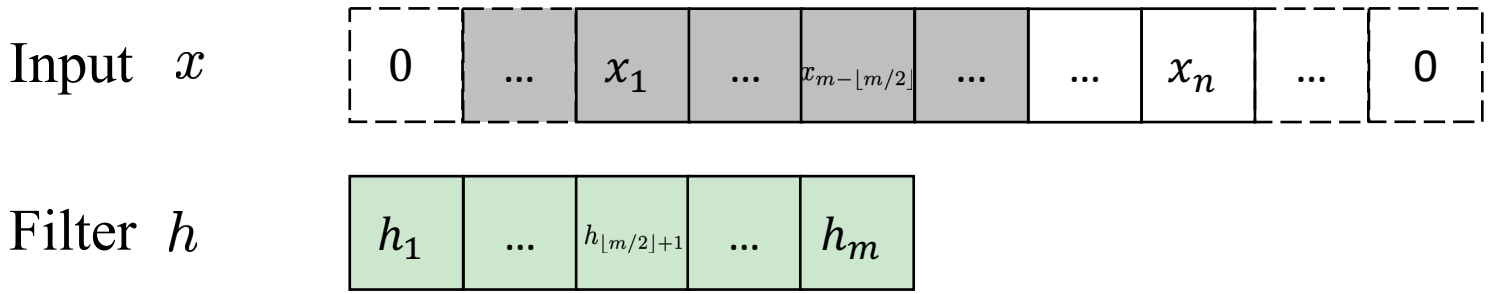
Omit boundary outputs to recover the preceding unpadded example

Odd filter length, stride 1, zero padding. DL libraries call this operation convolution.

CNN: Matrix Multiplication View II

1D cross-correlation as matrix multiplication

$$y^\top = (h * x)^\top = \begin{bmatrix} h_m & h_{m-1} & \cdots & h_3 & h_2 & h_1 \end{bmatrix} \begin{bmatrix} x_{m-\lfloor m/2 \rfloor} & x_{m-\lfloor m/2 \rfloor+1} & \cdots & x_m & x_{m+1} & \cdots & 0 & 0 \\ \vdots & \vdots & \cdots & x_{m-1} & x_m & \cdots & \vdots & \vdots \\ x_1 & x_2 & \cdots & \vdots & x_{m-1} & \cdots & x_n & 0 \\ 0 & x_1 & \cdots & \vdots & \vdots & \cdots & x_{n-1} & x_n \\ \vdots & 0 & \cdots & \vdots & \vdots & \cdots & \vdots & \vdots \\ 0 & 0 & \cdots & x_1 & x_2 & \cdots & x_{n-\lfloor m/2 \rfloor+1} & x_{n-\lfloor m/2 \rfloor} \end{bmatrix}$$

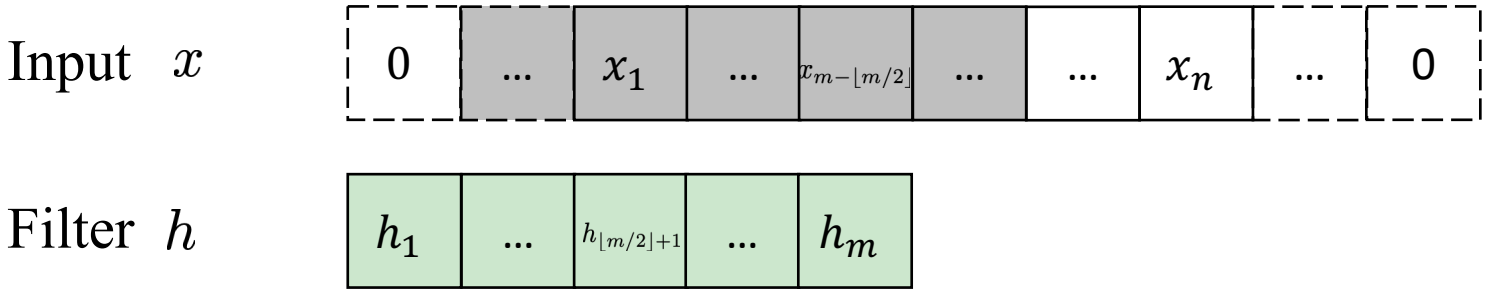


Odd filter length, stride 1, zero padding. DL libraries call this operation convolution.

CNN: Matrix Multiplication View II

1D cross-correlation as matrix multiplication

$$y^\top = (h * x)^\top = \begin{bmatrix} h_m & h_{m-1} & \cdots & h_3 & h_2 & h_1 \end{bmatrix} \begin{bmatrix} x_{m-\lfloor m/2 \rfloor} & x_{m-\lfloor m/2 \rfloor+1} & \cdots & x_m & x_{m+1} & \cdots & 0 & 0 \\ \vdots & \vdots & \cdots & x_{m-1} & x_m & \cdots & \vdots & \vdots \\ x_1 & x_2 & \cdots & \vdots & x_{m-1} & \cdots & x_n & 0 \\ 0 & x_1 & \cdots & \vdots & \vdots & \cdots & x_{n-1} & x_n \\ \vdots & 0 & \cdots & \vdots & \vdots & \cdots & \vdots & \vdots \\ 0 & 0 & \cdots & x_1 & x_2 & \cdots & x_{n-\lfloor m/2 \rfloor+1} & x_{n-\lfloor m/2 \rfloor} \end{bmatrix}$$

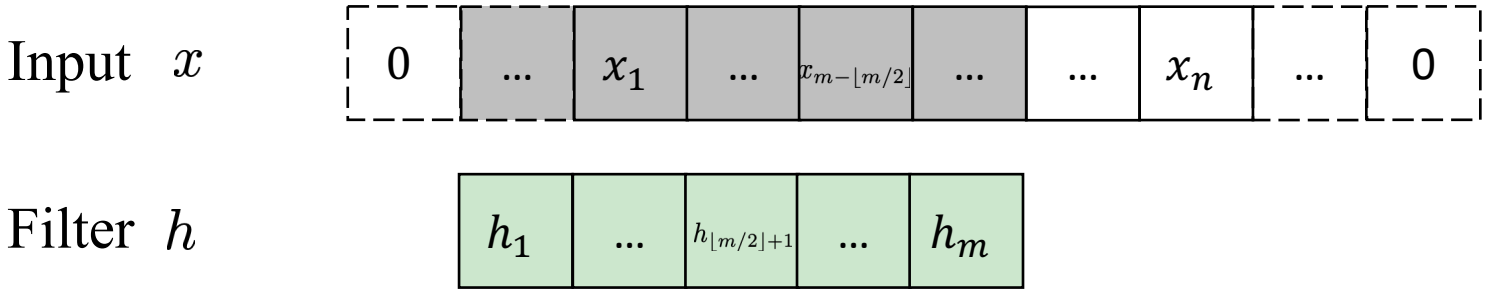


Odd filter length, stride 1, zero padding. DL libraries call this operation convolution.

CNN: Matrix Multiplication View II

1D cross-correlation as matrix multiplication

$$y^\top = (h * x)^\top = \begin{bmatrix} h_m & h_{m-1} & \cdots & h_3 & h_2 & h_1 \end{bmatrix} \begin{bmatrix} x_{m-\lfloor m/2 \rfloor} & x_{m-\lfloor m/2 \rfloor+1} & \cdots & x_m & x_{m+1} & \cdots & 0 & 0 \\ \vdots & \vdots & \cdots & x_{m-1} & x_m & \cdots & \vdots & \vdots \\ x_1 & x_2 & \cdots & \vdots & x_{m-1} & \cdots & x_n & 0 \\ 0 & x_1 & \cdots & \vdots & \vdots & \cdots & x_{n-1} & x_n \\ \vdots & 0 & \cdots & \vdots & \vdots & \cdots & \vdots & \vdots \\ 0 & 0 & \cdots & x_1 & x_2 & \cdots & x_{n-\lfloor m/2 \rfloor+1} & x_{n-\lfloor m/2 \rfloor} \end{bmatrix}$$



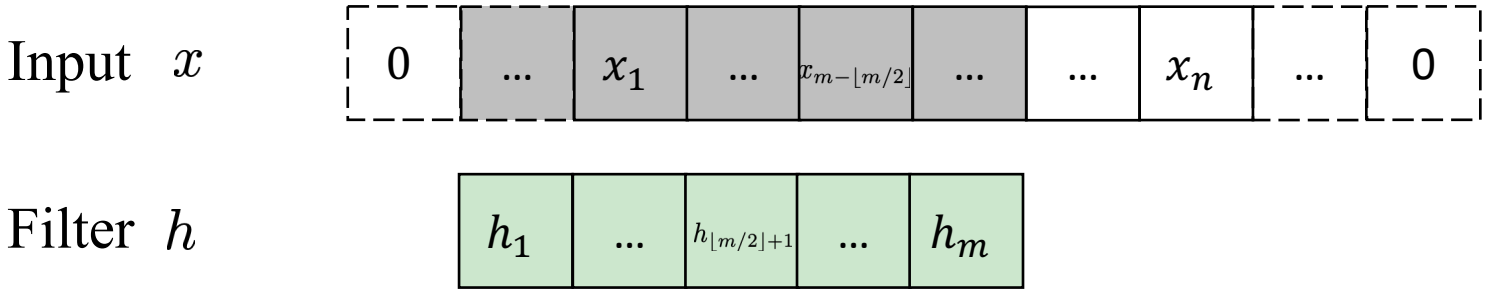
Odd filter length, stride 1, zero padding. DL libraries call this operation convolution.

CNN: Matrix Multiplication View II

1D cross-correlation as matrix multiplication

Input patches form a Toeplitz matrix in this ordering

$$y^\top = (h * x)^\top = \begin{bmatrix} h_m & h_{m-1} & \cdots & h_3 & h_2 & h_1 \end{bmatrix} \begin{bmatrix} x_{m-\lfloor m/2 \rfloor} & x_{m-\lfloor m/2 \rfloor+1} & \cdots & x_m & x_{m+1} & \cdots & 0 & 0 \\ \vdots & \vdots & \cdots & x_{m-1} & x_m & \cdots & \vdots & \vdots \\ x_1 & x_2 & \cdots & \vdots & x_{m-1} & \cdots & x_n & 0 \\ 0 & x_1 & \cdots & \vdots & \vdots & \cdots & x_{n-1} & x_n \\ \vdots & 0 & \cdots & \vdots & \vdots & \cdots & \vdots & \vdots \\ 0 & 0 & \cdots & x_1 & x_2 & \cdots & x_{n-\lfloor m/2 \rfloor+1} & x_{n-\lfloor m/2 \rfloor} \end{bmatrix}$$



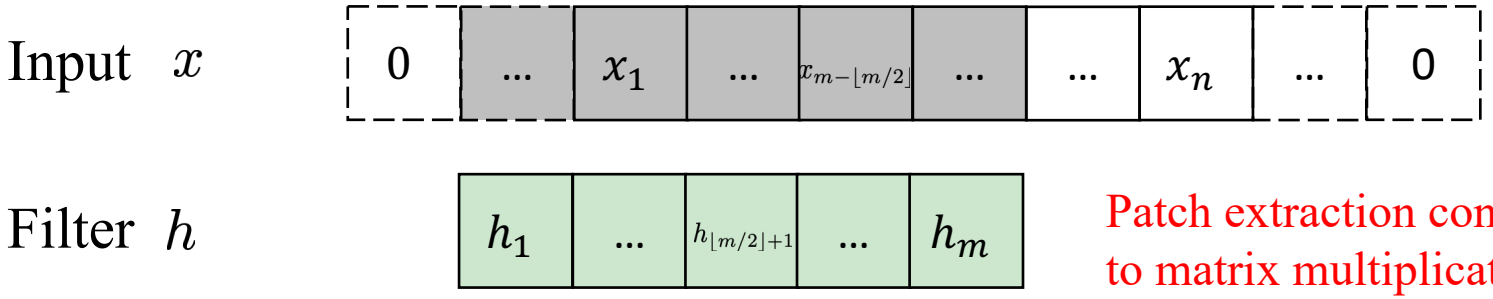
Odd filter length, stride 1, zero padding. DL libraries call this operation convolution.

CNN: Matrix Multiplication View II

1D cross-correlation as matrix multiplication

Input patches form a Toeplitz matrix in this ordering

$$y^\top = (h * x)^\top = \begin{bmatrix} h_m & h_{m-1} & \cdots & h_3 & h_2 & h_1 \end{bmatrix} \begin{bmatrix} x_{m-\lfloor m/2 \rfloor} & x_{m-\lfloor m/2 \rfloor+1} & \cdots & x_m & x_{m+1} & \cdots & 0 & 0 \\ \vdots & \vdots & \cdots & x_{m-1} & x_m & \cdots & \vdots & \vdots \\ x_1 & x_2 & \cdots & \vdots & x_{m-1} & \cdots & x_n & 0 \\ 0 & x_1 & \cdots & \vdots & \vdots & \cdots & x_{n-1} & x_n \\ \vdots & 0 & \cdots & \vdots & \vdots & \cdots & \vdots & \vdots \\ 0 & 0 & \cdots & x_1 & x_2 & \cdots & x_{n-\lfloor m/2 \rfloor+1} & x_{n-\lfloor m/2 \rfloor} \end{bmatrix}$$



Patch extraction connects convolution to matrix multiplication

Odd filter length, stride 1, zero padding. DL libraries call this operation convolution.

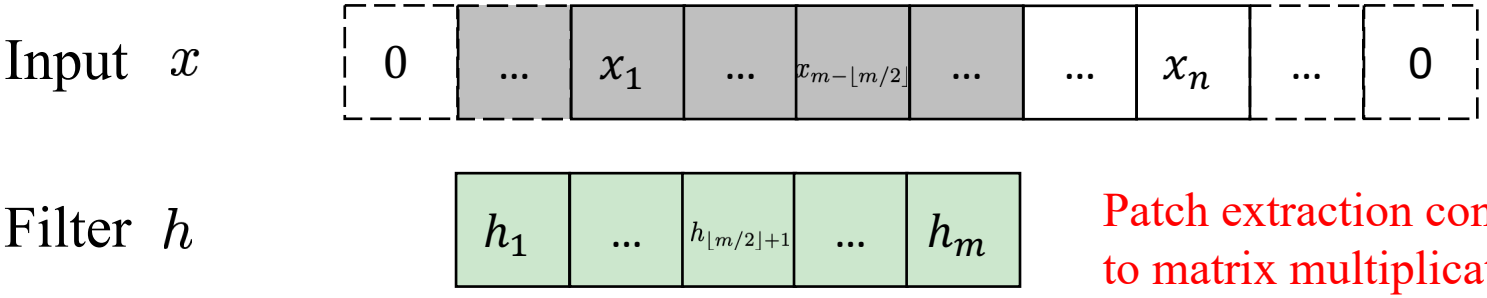
CNN: Matrix Multiplication View II

1D cross-correlation as matrix multiplication

Input patches form a Toeplitz matrix in this ordering

$$y^\top = (h * x)^\top = \begin{bmatrix} h_m & h_{m-1} & \cdots & h_3 & h_2 & h_1 \end{bmatrix} \begin{bmatrix} x_{m-\lfloor m/2 \rfloor} & x_{m-\lfloor m/2 \rfloor+1} & \cdots & x_m & x_{m+1} & \cdots & 0 & 0 \\ \vdots & \vdots & \cdots & x_{m-1} & x_m & \cdots & \vdots & \vdots \\ x_1 & x_2 & \cdots & \vdots & x_{m-1} & \cdots & x_n & 0 \\ 0 & x_1 & \cdots & \vdots & \vdots & \cdots & x_{n-1} & x_n \\ \vdots & 0 & \cdots & \vdots & \vdots & \cdots & \vdots & \vdots \\ 0 & 0 & \cdots & x_1 & x_2 & \cdots & x_{n-\lfloor m/2 \rfloor+1} & x_{n-\lfloor m/2 \rfloor} \end{bmatrix}$$

The same construction extends to 2D



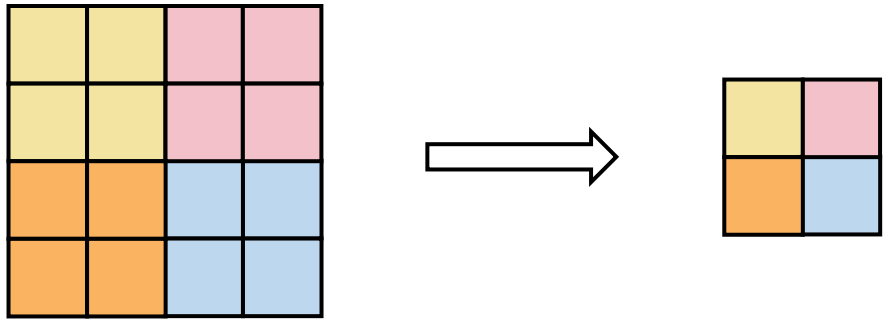
Patch extraction connects convolution to matrix multiplication

Odd filter length, stride 1, zero padding. DL libraries call this operation convolution.

CNN: Pooling

Convolutional Neural Network (CNN)

2 × 2 pooling, stride 2 (max or average)

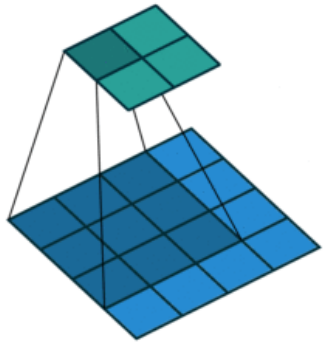


Each output aggregates one input window. The colors indicate the pooling regions.

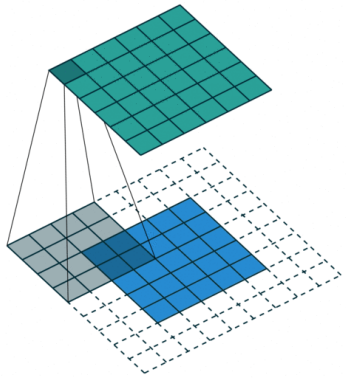
Convolution Variants

Blue: input. Cyan: output. Stride 1 unless stated otherwise.

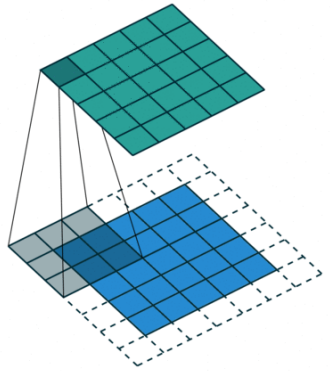
Valid



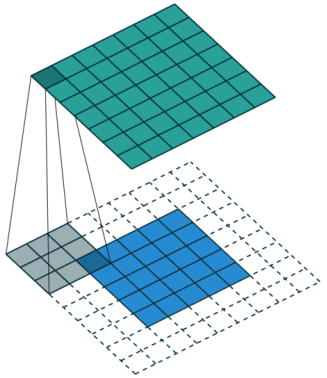
Arbitrary padding



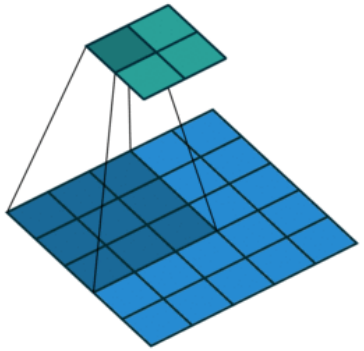
Same padding



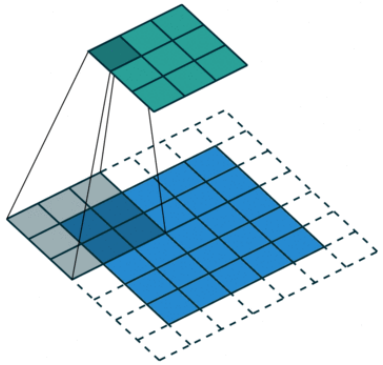
Full padding



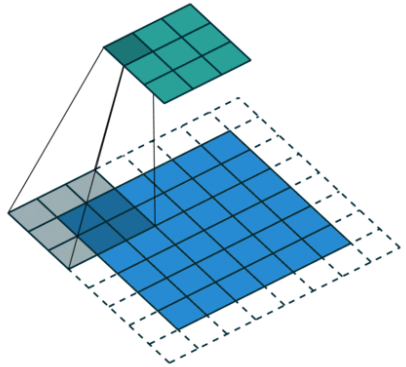
Stride 2



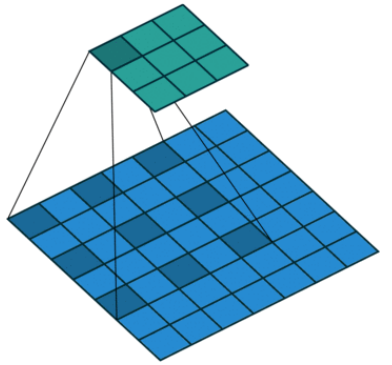
Stride 2 + padding



Stride 2 (remainder)



Dilation 2

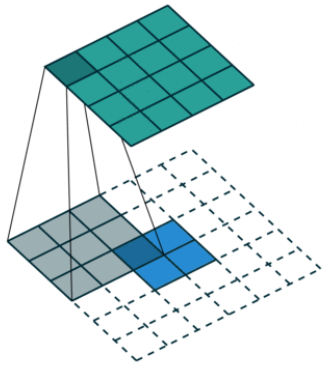


https://github.com/vdumoulin/conv_arithmetic/blob/master/README.md

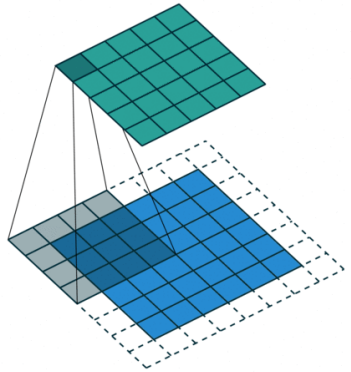
Transposed Convolution

Cases refer to the forward convolution. Blue: input. Cyan: output.

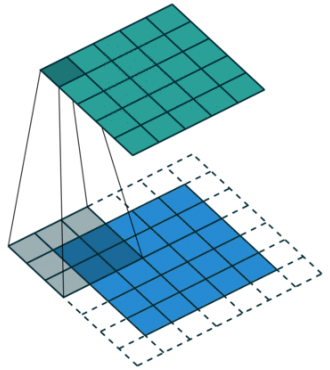
Valid



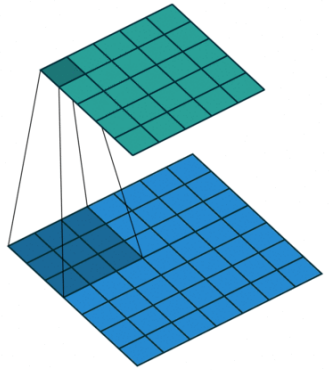
Arbitrary padding



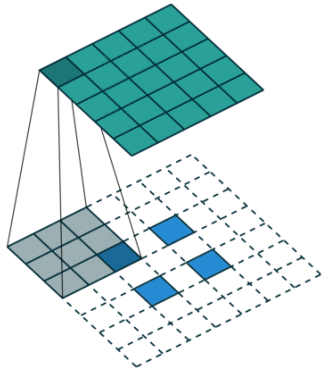
Same padding



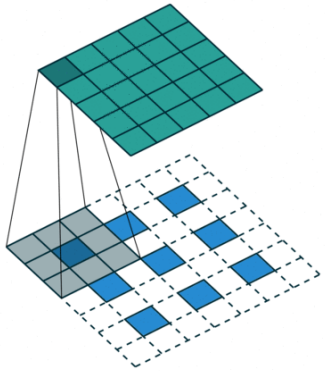
Full padding



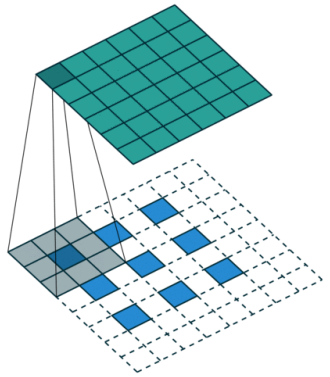
Stride 2



Stride 2 + padding



Stride 2 (remainder)



Transpose the connectivity.

Generally not an inverse.

https://github.com/vdumoulin/conv_arithmetic/blob/master/README.md

Convolution and Inductive Bias

- Local connectivity focuses computation on nearby spatial relationships
- Weight sharing applies the same filter across spatial locations
- Convolution is translation equivariant under appropriate boundary assumptions
- Stride, pooling, padding, and the output head affect the transformation behavior
- CNNs can process varying spatial sizes. A dense head may impose a size constraint.

Bronstein et al. (2021), Geometric Deep Learning

Outline

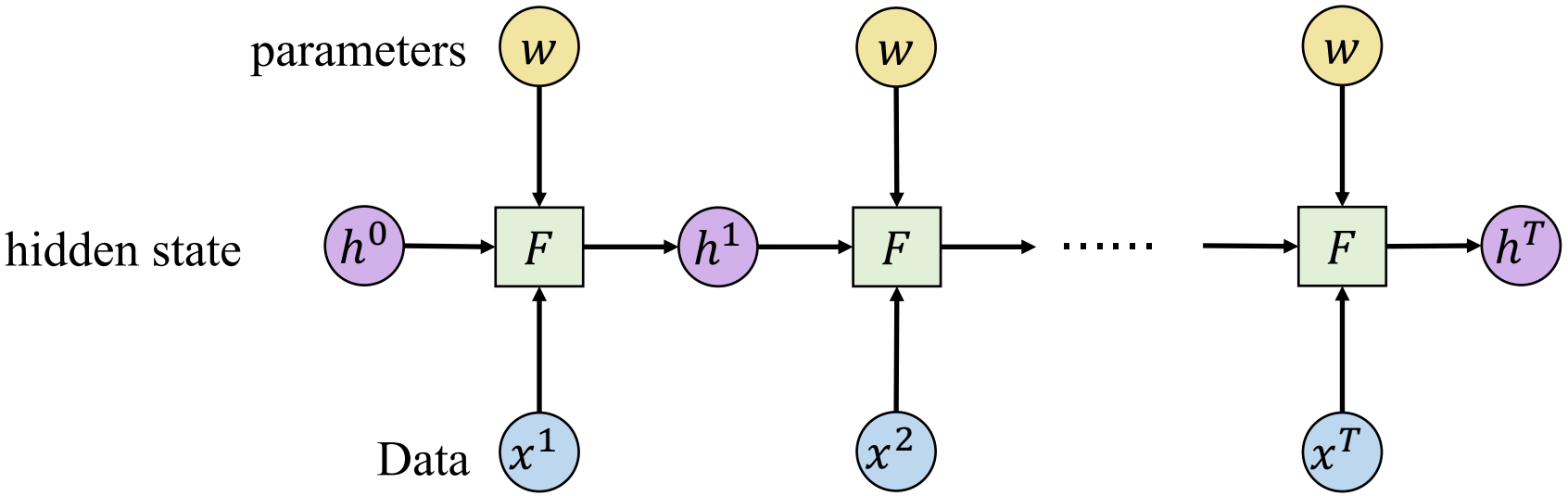
- Course information
- History and applications
- Basic deep learning models
  - MLPs: dense layers and nonlinearities
  - CNNs: local filters and spatial sharing
  - **RNNs: recurrence and temporal sharing**
- Learning from data
- Data structure vs. model inductive bias
- Research practice

RNN: Shared Computation over Time

Recurrent Neural Network (RNN)

Same neural network gets reused many times!

$$\mathbf{h}^t = F(\mathbf{x}^t, \mathbf{h}^{t-1}, W)$$



F could be any neural network!

RNN: State Update and Readout

Input $x_t \in \mathbb{R}^d$, hidden state $h_t \in \mathbb{R}^m$, and initial state $h_0 = 0$

$$a_t = W_x x_t + W_h h_{t-1} + b_h$$

$$h_t = \varphi(a_t), \qquad z_t = W_y h_t + b_y$$

$$W_x \in \mathbb{R}^{m \times d}, \qquad W_h \in \mathbb{R}^{m \times m}, \qquad W_y \in \mathbb{R}^{C \times m}$$

The same weights and biases apply at every time step

Read out at every step for sequence labeling, or at the final step for a class

Shared parameters support variable sequence length; sequence order still matters

Outline

- Course information
- History and applications
- Basic deep learning models
  - MLPs: dense layers and nonlinearities
  - CNNs: local filters and spatial sharing
  - RNNs: recurrence and temporal sharing
- **Learning from data**
- Data structure vs. model inductive bias
- Research practice

Learning from Data: Loss Functions

Each architecture defines a predictor f_θ; training minimizes empirical risk

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

Classification: predicted probabilities p and one-hot target y

$$\ell(p, y) = -\sum_{c=1}^{C} y_c \log p_c$$

Regression: real-valued prediction $\hat{y}$ and target y

$$\ell(\hat{y}, y) = \frac{1}{d_{\text{out}}} \|\hat{y} - y\|_2^2$$

Training an MLP: Forward Pass

Multi-Layer Perceptron (MLP)

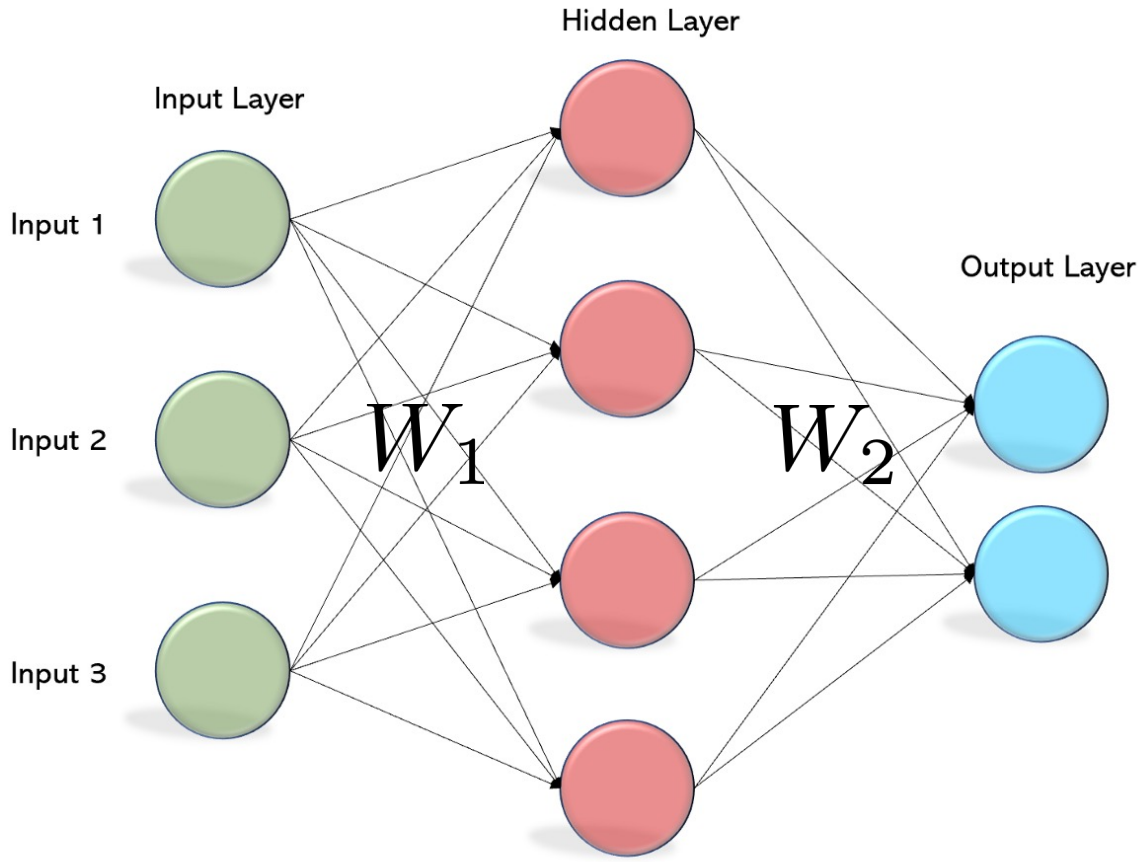


Image Credit: https://becominghuman.ai/multi-layer-perceptron-mlp-models-on-real-world-banking-data-f6dd3d7e998f

Training an MLP: Back-propagation

Backward pass:
compute gradients

$$\frac{\partial \ell}{\partial \operatorname{vec}(W_1)} = \left(\frac{\partial h}{\partial \operatorname{vec}(W_1)}\right)^\intercal \left(\frac{\partial y}{\partial h}\right)^\intercal \frac{\partial \ell}{\partial y}$$

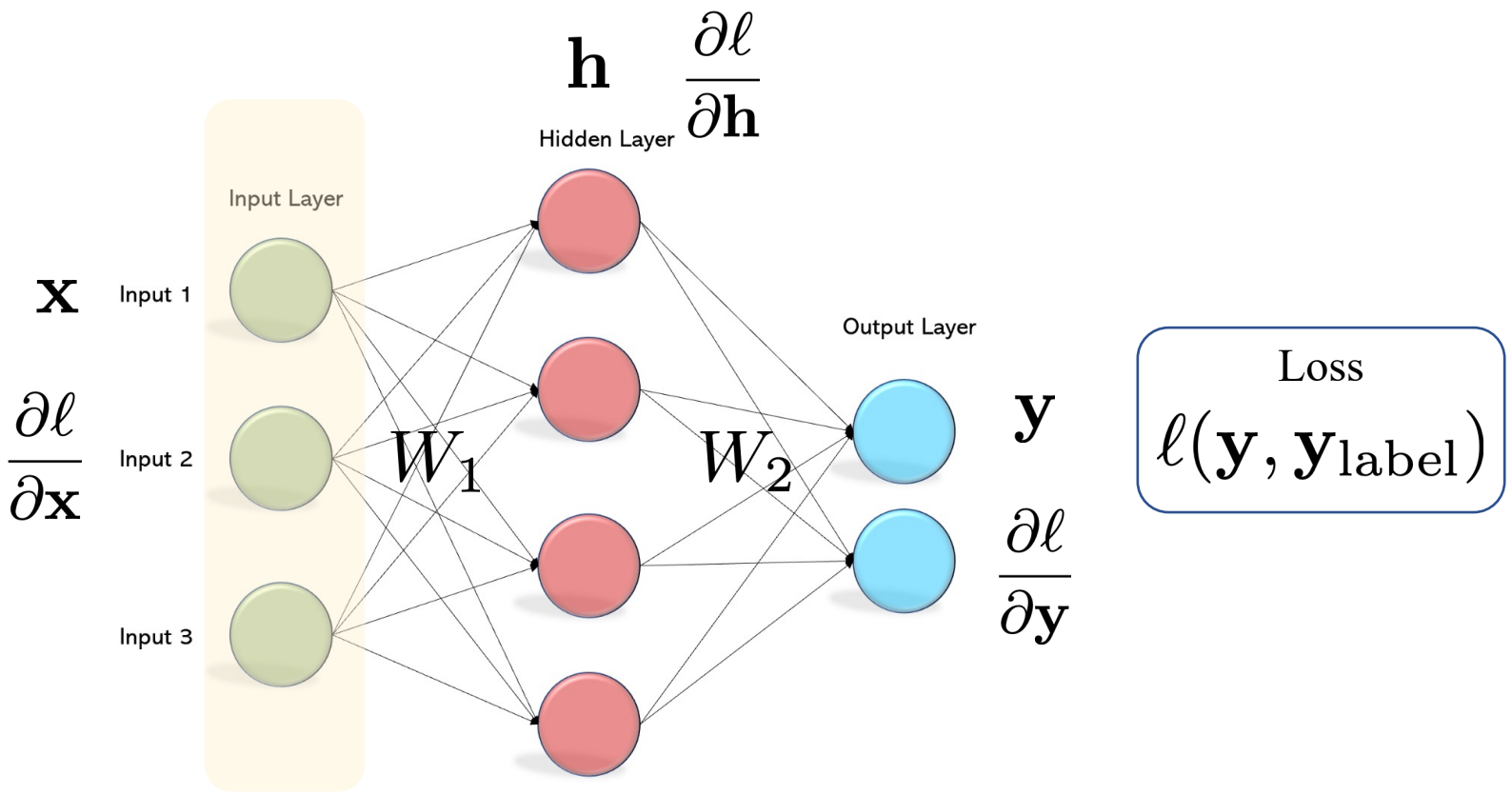


Image Credit: https://becominghuman.ai/multi-layer-perceptron-mlp-models-on-real-world-banking-data-f6dd3d7e998f

MLP: Gradients for Classification

One example: $h = \mathrm{ReLU}(a_1),\ z = W_2 h + b_2,\ p = \mathrm{softmax}(z)$, one-hot target y

$$\ell = -\sum_{c=1}^{C} y_c \log p_c, \qquad \delta_2 = p - y$$

$$\nabla_{W_2} \ell = \delta_2 h^\top, \qquad \nabla_{b_2} \ell = \delta_2$$

$$\delta_1 = (W_2^\top \delta_2) \odot \mathbf{1}[a_1 > 0]$$

$$\nabla_{W_1} \ell = \delta_1 x^\top, \qquad \nabla_{b_1} \ell = \delta_1$$

Gradients are outer products. Minibatch gradients average these example gradients.

RNN: Back-propagation through Time

Tanh states and a softmax classification loss at each time step

$$p_t = \mathrm{softmax}(z_t), \qquad \mathcal{L} = \sum_{t=1}^{T} \ell_t$$

$$\delta_t = \frac{\partial \mathcal{L}}{\partial a_t}, \qquad \delta_{T+1} = 0$$

$$\delta_t = \left(W_y^{\mathsf{T}}(p_t - y_t) + W_h^{\mathsf{T}}\delta_{t+1}\right) \odot \left(1 - h_t \odot h_t\right)$$

$$\nabla_{W_h}\mathcal{L} = \sum_{t=1}^{T} \delta_t h_{t-1}^{\mathsf{T}}$$

Shared weights collect gradients from every step; long chains can shrink or grow them

Gradients and Parameter Updates

Choose a minibatch B and compute its mean loss

$$\mathcal{L}_B(\theta) = \frac{1}{|B|} \sum_{i \in B} \ell\big(f_\theta(x_i), y_i\big)$$

Back-propagation computes the gradient of this loss

$$g_t = \nabla_\theta \mathcal{L}_B(\theta_t), \qquad \theta_{t+1} = \theta_t - \eta_t g_t$$

The second equation is a plain SGD update with learning rate η_t

Other optimizers, such as AdamW, use the same gradients differently

Loshchilov & Hutter (2019), Decoupled Weight Decay Regularization

Outline

- Course information
- History and applications
- Basic deep learning models
  - MLPs: dense layers and nonlinearities
  - CNNs: local filters and spatial sharing
  - RNNs: recurrence and temporal sharing
- Learning from data
- **Data structure vs. model inductive bias**
- Research practice

Inductive Bias in Model Design

- Inductive bias: assumptions that guide generalization beyond the training data
- The model's structure encodes some of these assumptions
- CNNs: local filters and weight sharing favor translation equivariance
- RNNs: the same state-update rule applies at every time step
- Two views: encode known structure, or learn task structure from data at scale

Exact CNN equivariance depends on stride and boundary handling; subsampling requires care.

Invariance and Equivariance

Invariance: transforming the input leaves the prediction unchanged

$$f(Tx) = f(x)$$

Example: translating an object should preserve its class label

Equivariance: the output transforms in a corresponding way

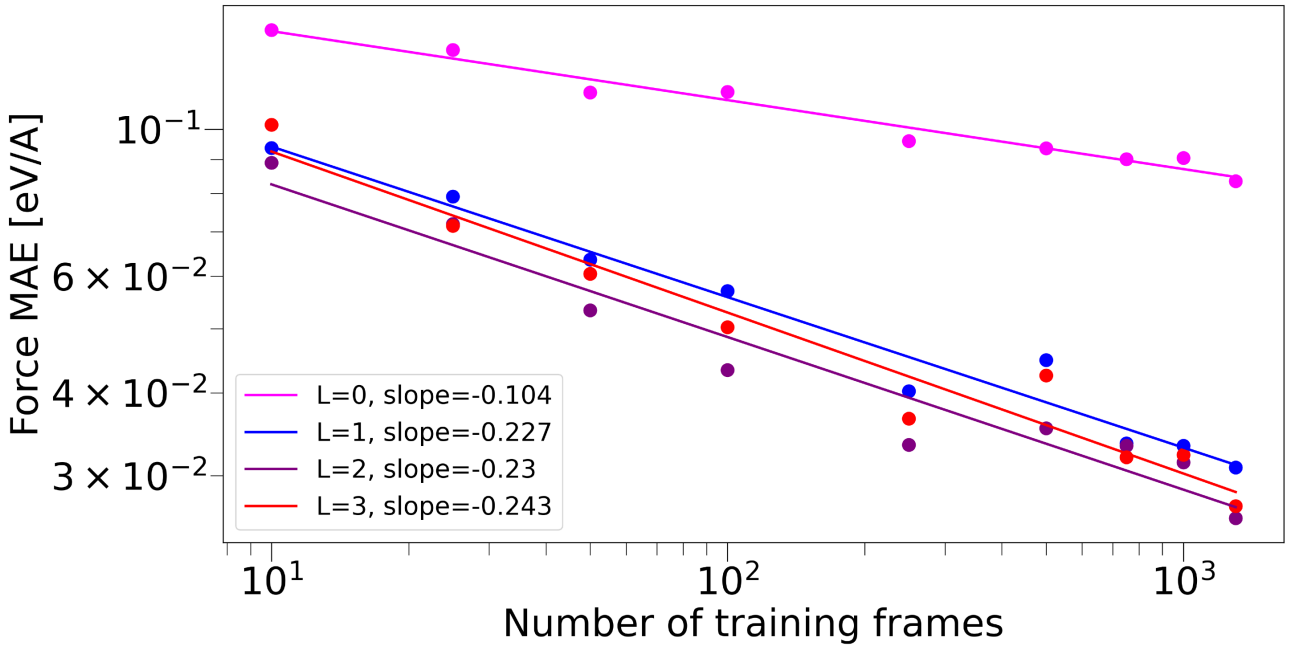
$$f(Tx) = T'f(x)$$

Example: translating an image should shift its segmentation mask

View 1: Build in Task-specific Structure

- NequIP encodes rotational symmetry in molecular energies and forces.
- Equivariant features improve data efficiency on this water benchmark.
- The architecture enforces symmetry even before training.

NequIP: force prediction



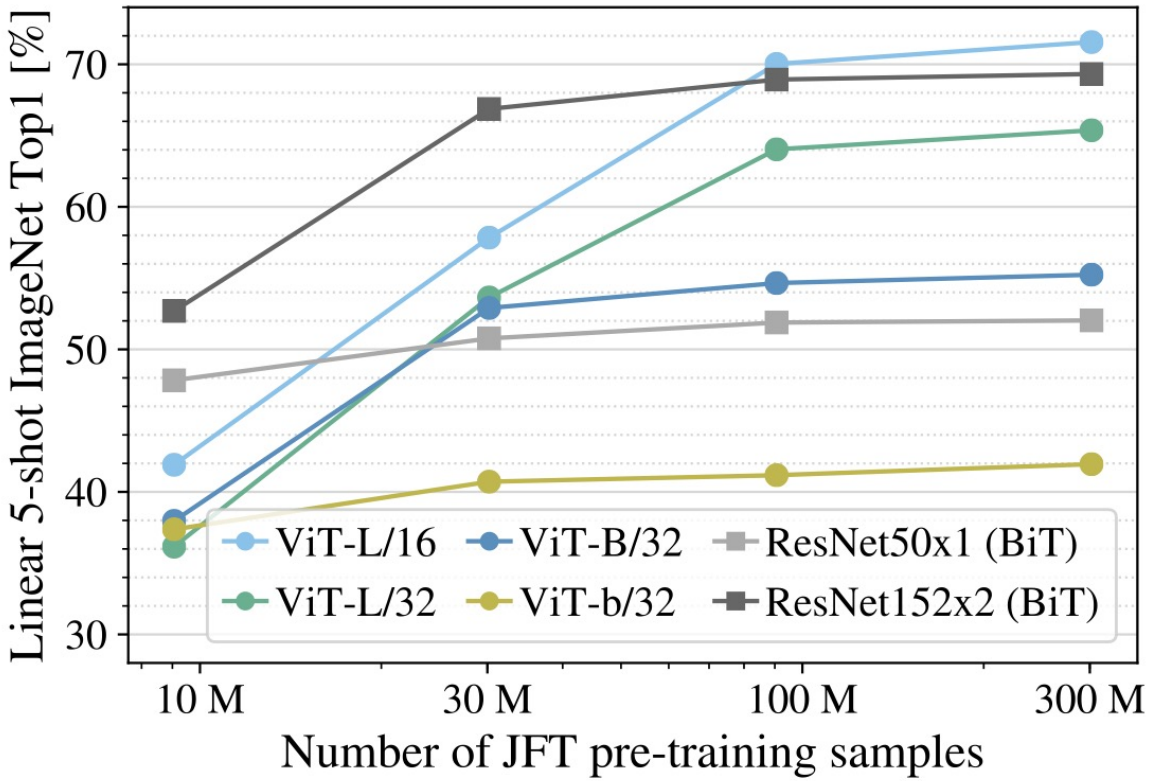
Magenta: scalar-only features.
Other curves: equivariant tensor features.

Batzner et al., NequIP (2022), arXiv v3, Fig. 7

View 2: Learn Structure at Scale

- ViT applies a Transformer to image patches with fewer vision-specific assumptions.
- Large-scale pre-training can compensate for that weaker prior.
- Patches, positional information, and shared attention still impose biases.

ViT: pre-training scale and transfer



Dosovitskiy et al., Vision Transformer, ICLR (2021), Fig. 4

Two Views on Inductive Bias

Task-aligned model structure

- Can learn efficiently from limited data when the prior is appropriate
- Can guarantee chosen symmetries by construction
- A wrong assumption restricts valid solutions

General architecture + large-scale data

- Flexible features and broad reuse across tasks
- Can require substantial pre-training data and compute
- Task-specific symmetries are not automatically guaranteed

Compare both at a stated data and compute budget. Hybrid designs combine them.

Outline

- Course information
- History and applications
- Basic deep learning models
  - MLPs: dense layers and nonlinearities
  - CNNs: local filters and spatial sharing
  - RNNs: recurrence and temporal sharing
- Learning from data
- Data structure vs. model inductive bias
- **Research practice**

Reading Deep Learning Research

Research venues

- General ML: ICLR, NeurIPS, and ICML.
- Vision and language: CVPR, ICCV, ACL, COLM, and EMNLP.
- Robotics, embodied AI, and physical AI: ICRA, IROS, RSS, and CoRL.

Tips for Reading Papers: Think Like a Reviewer

- Before accepting the solution, ask how you would solve the problem.
- Challenge the claims: are the assumptions justified and the evidence convincing?
- Look for missing baselines, alternative explanations, and counterexamples.
- Write your own verdict: strengths, weaknesses, questions, and a decisive next test.

What Counts as Convincing Evidence?

- Define the task, data split, metrics, and intended use
- Compare strong baselines under a clear compute and data budget
- Use ablations to test which design choices actually matter
- Inspect uncertainty, failure cases, and out-of-distribution behavior
- Separate benchmark gains from evidence for the claimed mechanism
- Open-source code, configurations, and practical training tricks or guidelines

Lecture Summary

- MLPs, CNNs, and RNNs combine learnable transformations and nonlinearities
- The architecture determines connectivity and parameter sharing
- A loss defines the training signal; back-propagation computes its gradients
- The optimizer uses those gradients to update parameters
- After the break: sets, sequences, and models that respect their structure

Questions?