Lecture 7.2: Modern architecture for deep learning

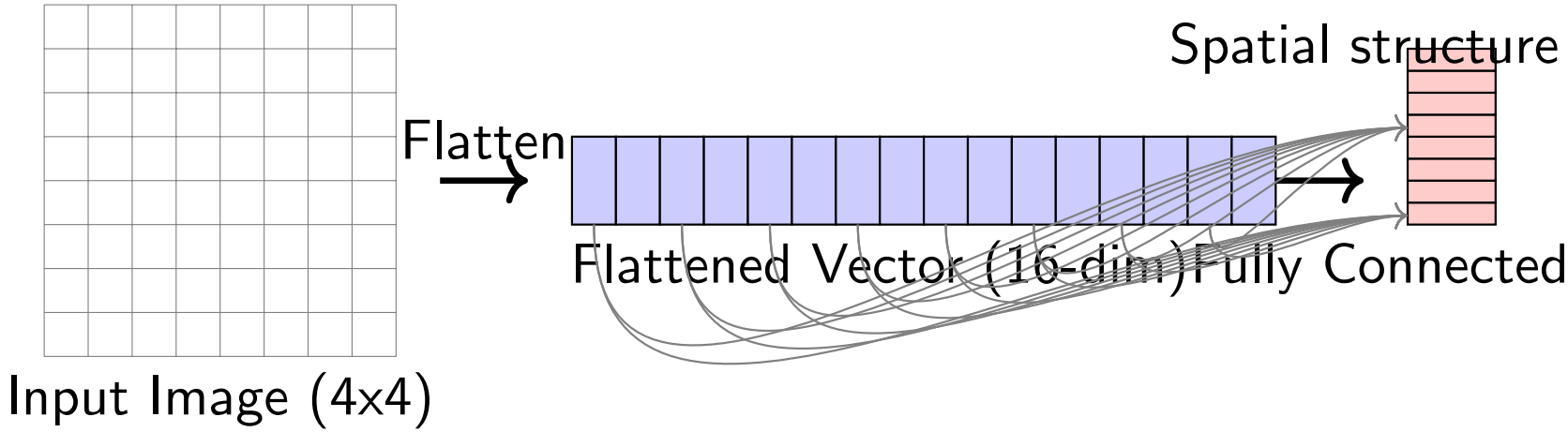
谢丹
清华大学数学系

November 23, 2025

Section 1: CNN

The Problem with Fully Connected Networks for Images

16 pixels $\rightarrow$ 16 input features

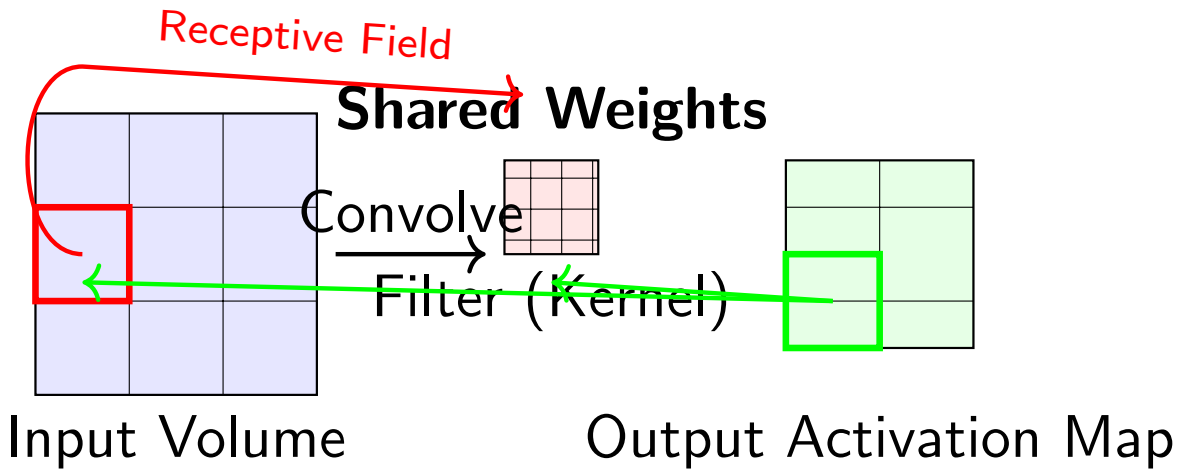


The Curse of Dimensionality and Lost Structure

A small 100x100 RGB image has $100 \times 100 \times 3 = 30,000$ input features. A single FC layer with 1,000 neurons would have 30 million parameters! This is:

- Computationally prohibitive
- Prone to overfitting
- Ignores the spatial structure of pixels

The CNN Solution: Convolution and Parameter Sharing



Key Idea: Parameter Sharing

The same filter is used across the entire image. A filter that detects a horizontal edge is useful everywhere. This drastically reduces parameters and encodes translation invariance.

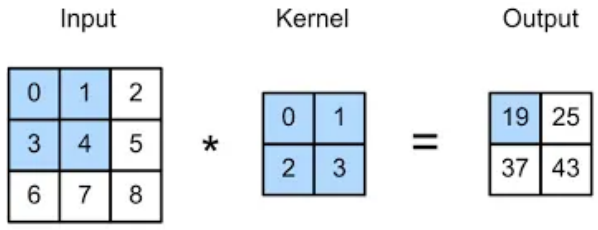


Figure: Convolution product.

Mathematical Foundation of Convolutional Networks I

The building blocks of CNNs expressed mathematically

1. Discrete Convolution Operation

The core operation in a CNN is the discrete convolution between an input volume and a set of learnable filters:

$$(\mathbf{I} * \mathbf{K})_{i,j} = \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} \mathbf{I}_{i+m,j+n} \cdot \mathbf{K}_{m,n}$$

Where:

- $\mathbf{I}$ is the input volume (e.g., an image)
- $\mathbf{K}$ is the filter/kernel
- $M \times N$ is the size of the filter

2. Multi-Channel Convolution

Mathematical Foundation of Convolutional Networks II

The building blocks of CNNs expressed mathematically

For RGB images or feature maps from previous layers:

$$\mathbf{Z}_{i,j,k} = \sum_{c=0}^{C-1} \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} \mathbf{X}_{i+m,j+n,c} \cdot \mathbf{W}_{m,n,c,k} + b_k$$

Where:

- $\mathbf{X}$ is the input volume of size (H, W, C)
- $\mathbf{W}$ is the filter bank of size (M, N, C, K)
- b_k is the bias term for the k-th filter
- $\mathbf{Z}$ is the output volume of size (H', W', K)

Managing Output Size: Padding and Strides I

Challenge: Convolution layers naturally reduce the spatial size (height/width) of the feature map.

Solution 1: Padding

- Add a border of zeros (or other values) around the input image.
- Allows the kernel to process the edges of the original input more thoroughly.
- **Result:** The output feature map can be the same size as the input.

Solution 2: Strided Convolutions

- The kernel is moved with a step size $S > 1$ (the *stride*).
- This aggressively downsamples the input, reducing the output size.

Managing Output Size: Padding and Strides II

- The output size for a square input and kernel is given by:

$$O = \frac{W - K + 2P}{S} + 1$$

O: Output size W: Input size
K: Kernel size P: Padding size
S: Stride

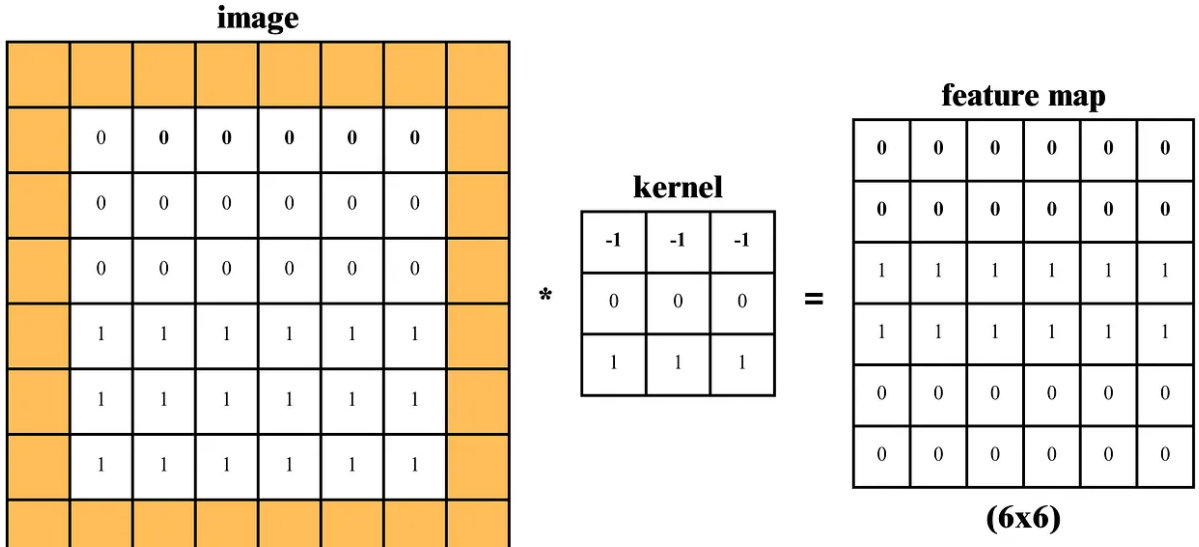


Figure: Padding.

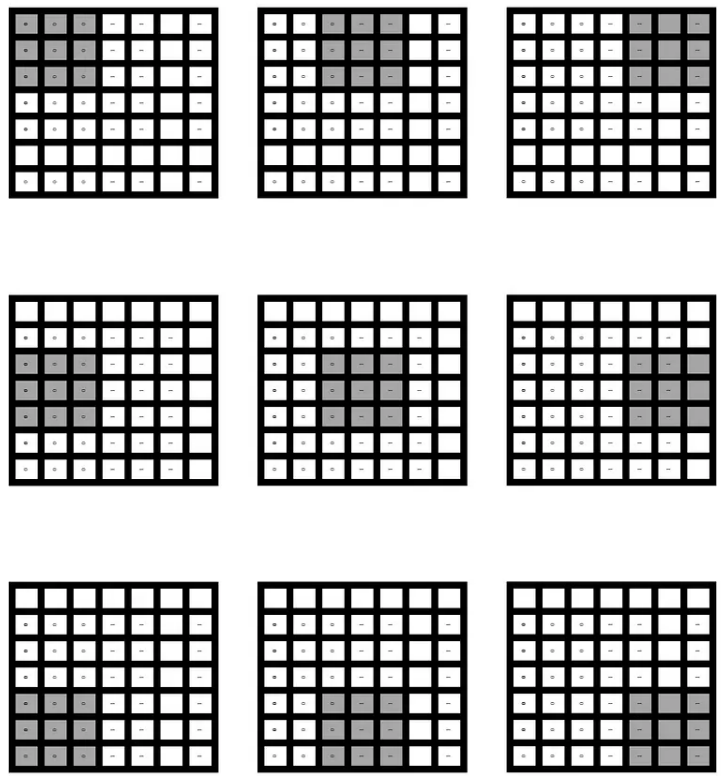


Figure: Stride.

Mathematical Formulation (Continued) I

Activation, pooling, and the complete forward pass

3. Activation Function

After convolution, we apply a non-linear activation function:

$$\mathbf{A}_{:,:,k} = f(\mathbf{Z}_{:,:,k})$$

Common choices:

- ReLU: $f(x) = \max(0, x)$
- Sigmoid: $f(x) = \frac{1}{1+e^{-x}}$
- Tanh: $f(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$

4. Pooling Operation

Mathematical Formulation (Continued) II

Activation, pooling, and the complete forward pass

Pooling reduces spatial dimensions while retaining important features:

$$\mathbf{P}_{i,j,k} = \text{pool}(\mathbf{A}_{s \cdot i : s \cdot i + p, s \cdot j : s \cdot j + p, k})$$

Where:

- pool is typically $\max$ or average
- p is the pool size
- s is the stride

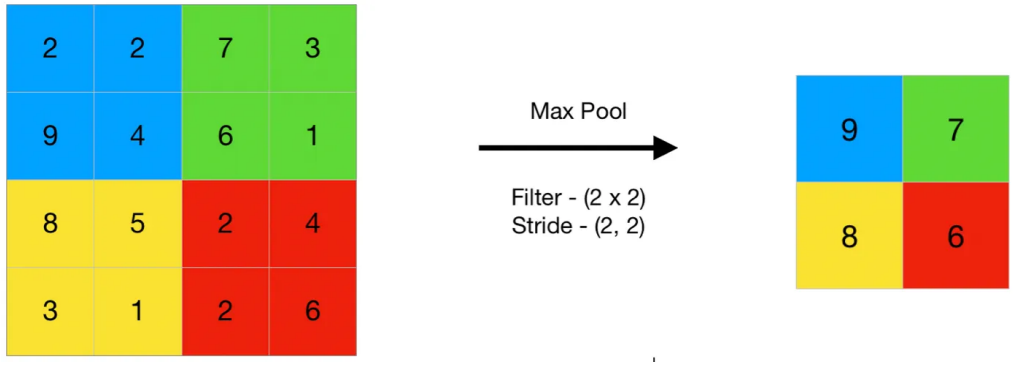


Figure: Pooling.

Why CNNs Work: Summary of Key Properties

Key Architectural Features

- Local Connectivity: Neurons are connected only to a local region (receptive field).
- Parameter Sharing: The same weights are used for different parts of the input.
- Pooling: Provides spatial invariance and reduces dimensionality.

Benefits

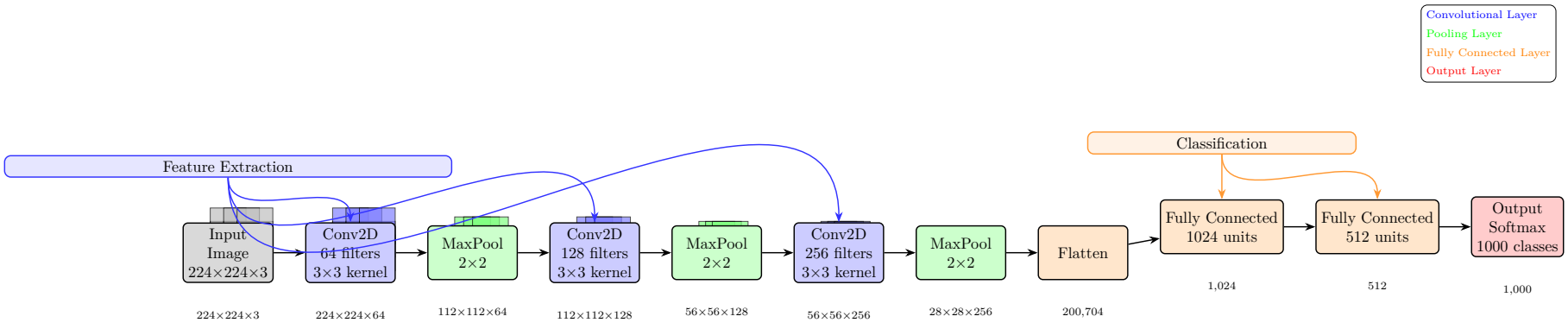
- Parameter Efficiency: Drastically fewer parameters than FC networks.
- Translation Invariance: Robust to shifts of the object in the image.
- Hierarchical Learning: Learns features from simple to complex.

Common Applications

- Image Classification
- Object Detection
- Semantic Segmentation
- Image Generation (GANs)
- Medical Image Analysis
- Video Analysis

Landmark Architectures

- LeNet-5 (1998)
- AlexNet (2012)
- VGGNet (2014)
- GoogLeNet (2015)
- ResNet (2015)


Convolutional Layer
Pooling Layer
Fully Connected Layer
Output Layer
Feature Extraction
Classification
Input Image 224×224×3
Conv2D 64 filters 3×3 kernel
MaxPool 2×2
Conv2D 128 filters 3×3 kernel
MaxPool 2×2
Conv2D 256 filters 3×3 kernel
MaxPool 2×2
Flatten
Fully Connected 1024 units
Fully Connected 512 units
Output Softmax 1000 classes
224×224×3
224×224×64
112×112×64
112×112×128
56×56×128
56×56×256
28×28×256
200,704
1,024
512
1,000

Parameter Counting Formula

The total number of parameters in a convolutional layer is:

$$\text{Total Parameters} = (K_w \times K_h \times C_{in} + 1) \times C_{out}$$

Where:

- K_w: Filter width
- K_h: Filter height
- C_{in}: Input channels
- C_{out}: Output channels (number of filters)
- $+1$: Bias term for each filter

Example Calculation

Given Parameters

- Input: $32 \times 32 \times 3$ (RGB image)
- Filters: 64 filters of size 3×3
- Stride: 1, Padding: same

Calculation

$$\begin{aligned} \text{Weights per filter} &= 3 \times 3 \times 3 = 27 \\ \text{Total weights} &= 64 \times 27 = 1,728 \\ \text{Biases} &= 64 \\ \text{Total parameters} &= 1,728 + 64 = 1,792 \end{aligned}$$

Comparison: Different Layer Configurations

Configuration	Filters	Kernel Size	Parameters
Small	32	3×3	896
Medium	64	3×3	1,792
Large	128	3×3	3,584
Very Large	256	3×3	7,168

Table: Parameter counts for different configurations

Example using pytorch!

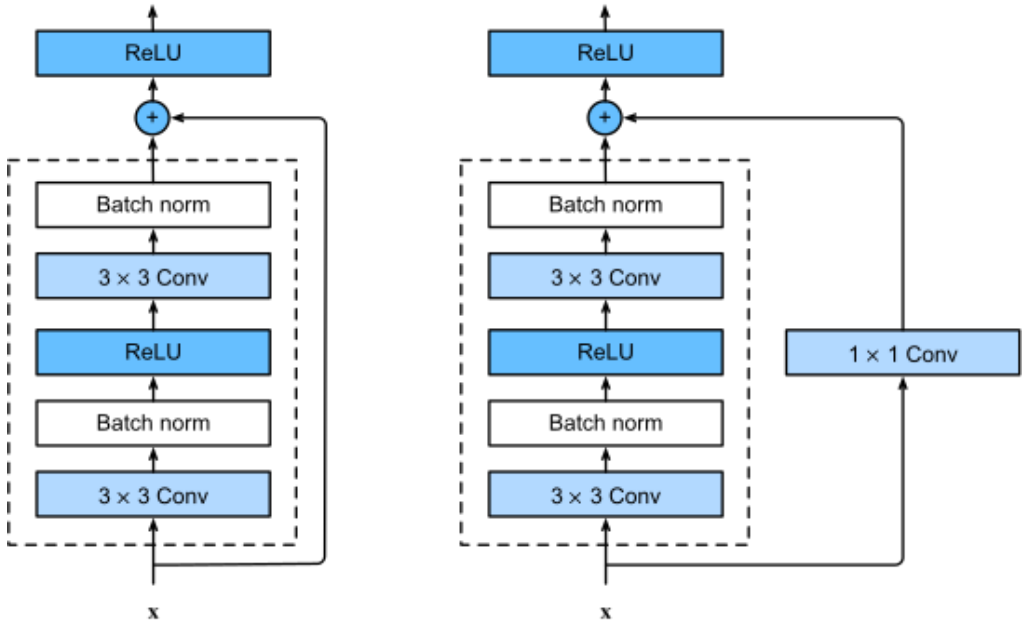


Figure: Resnet-block.

Section 2: Recurrant neural network

Introduction: The Problem with Standard NNs

- Standard Feedforward Neural Networks are **stateless**.
- They map an input to an output, assuming inputs are **independent**.
- This is a poor fit for **sequential data**:
  - Time Series (Stock prices, weather)
  - Natural Language (Sentences, paragraphs)
  - Video and Audio
- For these tasks, the **order and context** of previous inputs are crucial for predicting the next output.

Key Question

How can a network remember what it saw in the past to inform its present decision?

What are RNNs?

- **Neural networks with memory** for sequential data
- **Recurrent connections** allow information persistence
- **Parameter sharing** across time steps
- Suitable for time-series, text, speech, and other sequential data

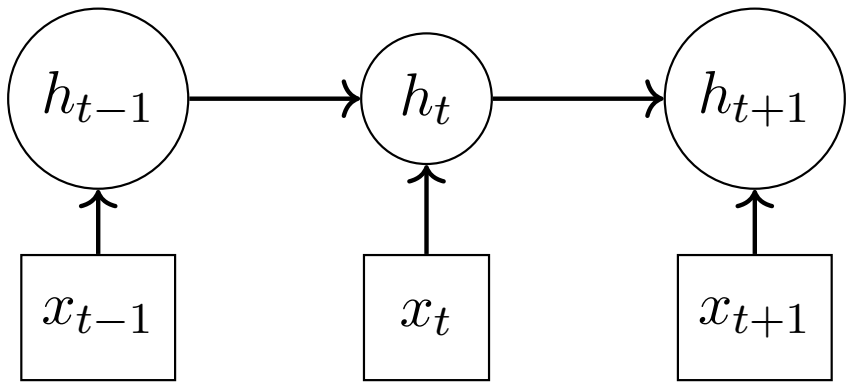


Figure: RNN unrolled through time

RNN Architecture

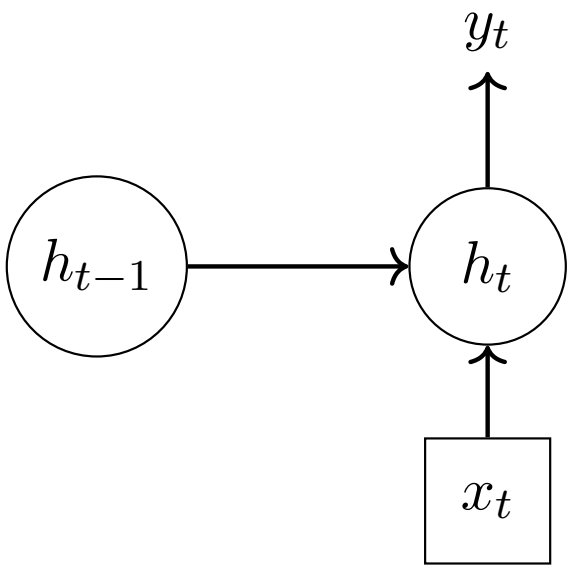


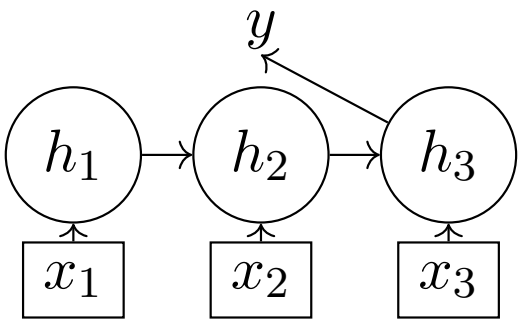
Figure: RNN cell with recurrence

Mathematical Formulation

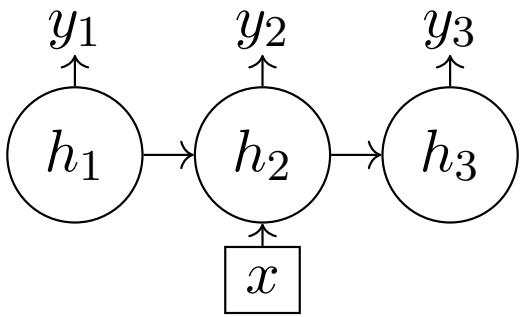
$$h_t = \sigma(W_{hh}h_{t-1} + W_{xh}x_t + b_h)$$
$$y_t = W_{hy}h_t + b_y$$

- h_t: Hidden state at time t
- x_t: Input at time t
- W: Weight matrices
- b: Bias terms
- σ: Activation function

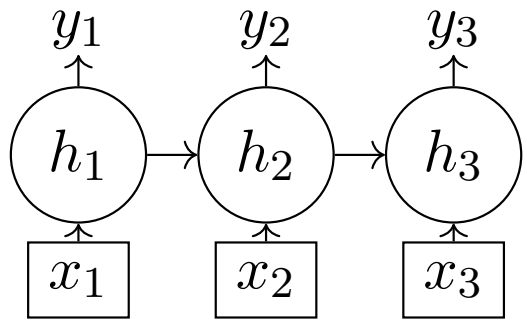
RNN Variants



One-to-Many



Many-to-One



Many-to-Many

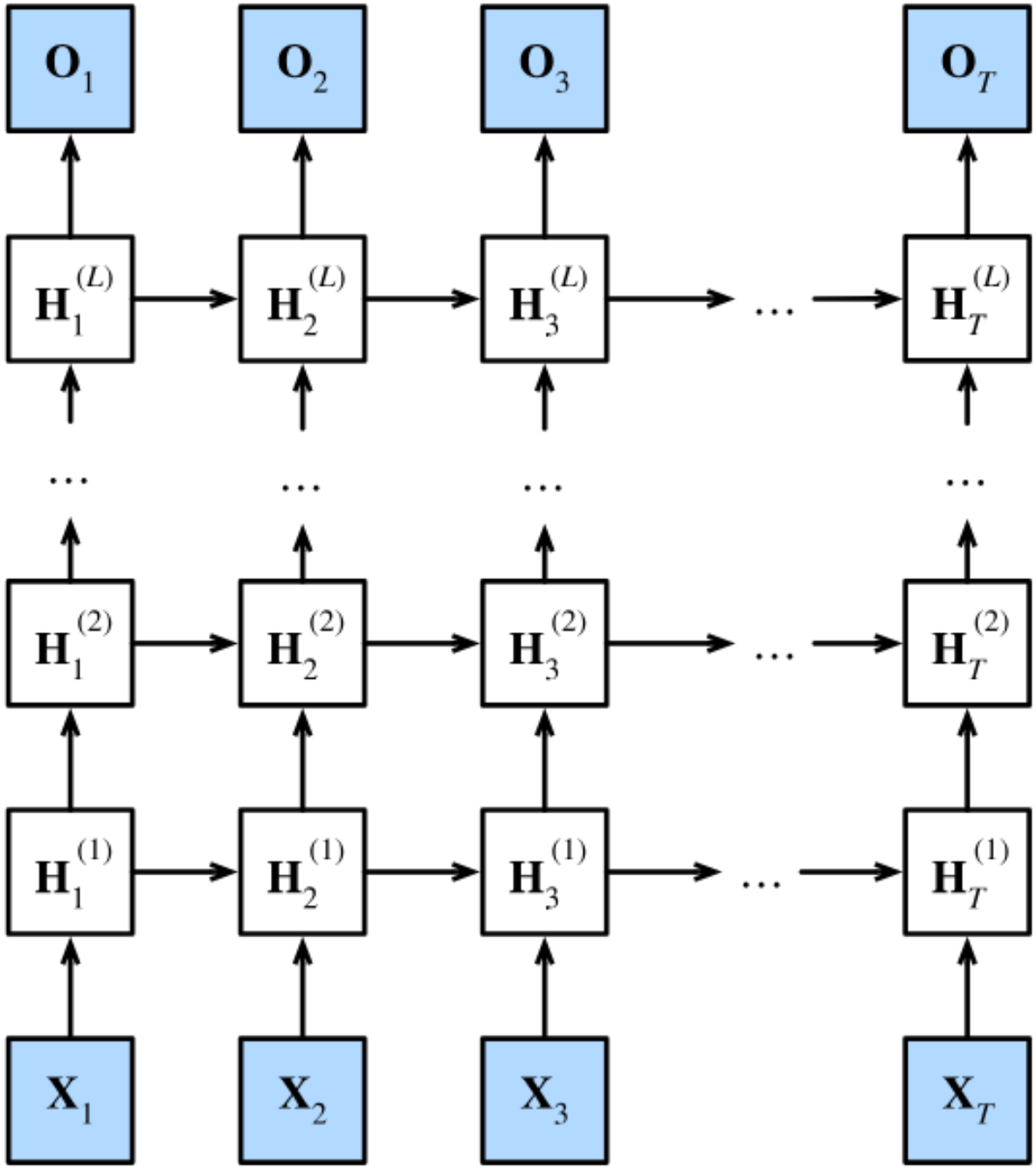


Figure: Deep RNN

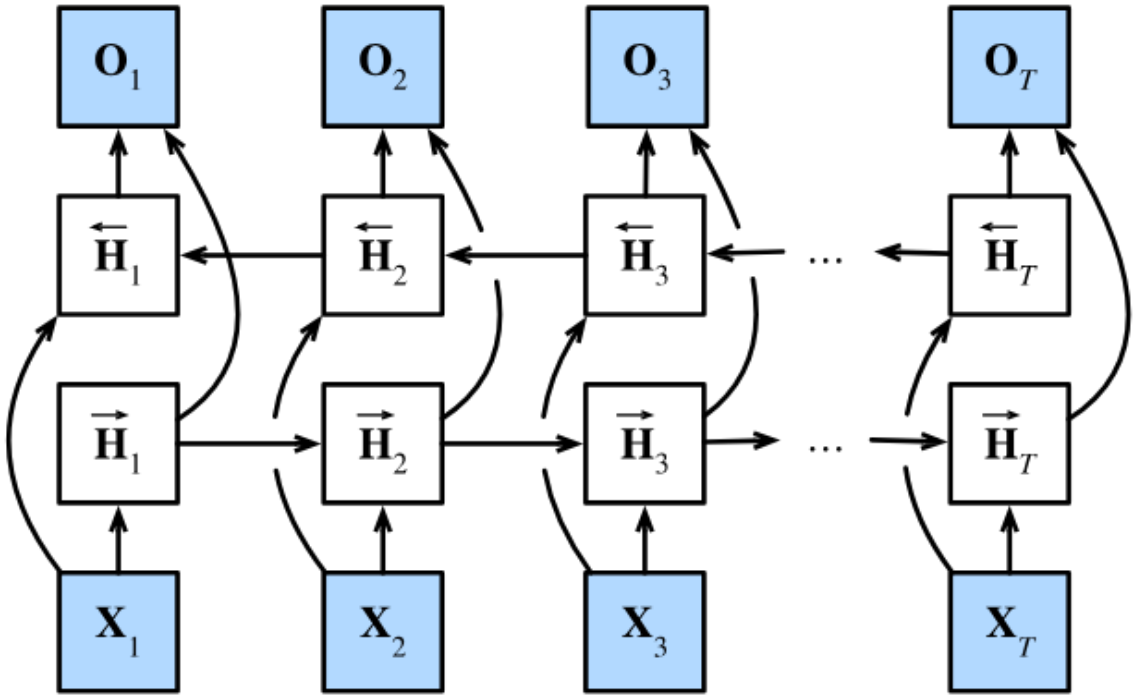


Figure: Bidirectional RNN

Challenges and Limitations

Vanishing/Exploding Gradients

- Gradients shrink/explode exponentially
- Difficulty learning long-range dependencies
- Limited memory span

$$\frac{\partial h_t}{\partial h_k} = \prod_{j=k+1}^{t} \frac{\partial h_j}{\partial h_{j-1}}$$

$$= \prod_{j=k+1}^{t} W_{hh}^{\top} \cdot \text{diag}(\sigma'(W_{hh} h_{j-1} + \cdots))$$

Solutions

- LSTM networks
- GRU networks
- Gradient clipping
- Better initialization

Solutions: LSTM & GRU

Gated Cells to control information flow.

Long Short-Term Memory (LSTM)

- **Input, Forget, Output Gates** decide what information to keep, forget, and output.
- Maintains a separate **cell state** (C_t) for long-term memory.
- Powerful but computationally complex.

Gated Recurrent Unit (GRU)

- **Update & Reset Gates**.
- Simpler, faster to train than LSTM.
- Merges cell state and hidden state.
- Often performs on par with LSTM.

LSTM Core Concept

Key Idea

An LSTM introduces a **cell state** (C_t) that runs through the entire sequence, allowing information to persist. It uses **gates** to regulate the flow of information.

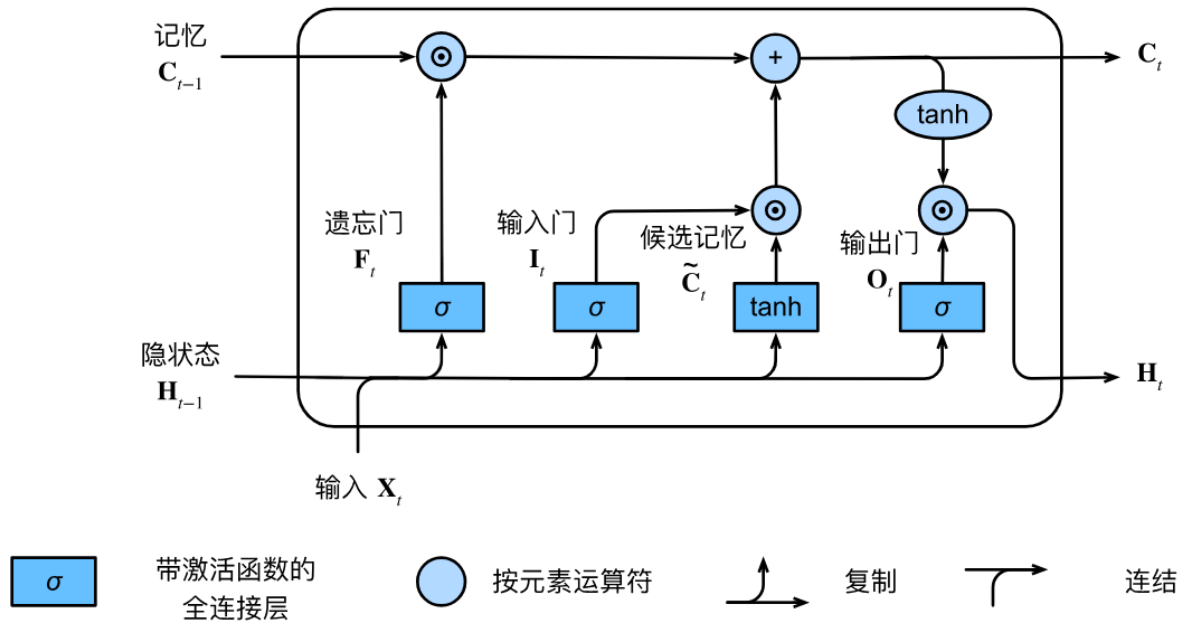


Figure: Structure of an LSTM unit.

LSTM Gates: The Mathematical Details

At each timestep t, given input x_t and previous hidden state h_{t-1}:

Forget Gate	**"What to discard from the cell state?"**
	$f_t = \sigma(\, W_f \cdot [h_{t-1}, x_t] + b_f \,)$
Input Gate	**"What new information to store?"**
	$i_t = \sigma(\, W_i \cdot [h_{t-1}, x_t] + b_i \,)$
	$\tilde{C}_t = \tanh(\, W_C \cdot [h_{t-1}, x_t] + b_C \,)$
Update Cell State	**Combine forget and input decisions.**
	$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$
Output Gate	**"What part of the cell state to output?"**
	$o_t = \sigma(\, W_o \cdot [h_{t-1}, x_t] + b_o \,)$
	$h_t = o_t * \tanh(C_t)$

Where: σ = Sigmoid, $\tanh$ = Hyperbolic Tangent, $*$ = Element-wise multiplication, $[a, b]$ = Vector concatenation.

Summary of Parameters

- **Weights**: Four sets of weight matrices W_f, W_i, W_C, W_o.
- **Biases**: Four bias vectors b_f, b_i, b_C, b_o.
- The gates allow the LSTM to **learn long-term dependencies** by carefully regulating the cell state.
- The gradients flowing back through the cell state are protected from vanishing/exploding by the additive update $C_t = \ldots + i_t * \tilde{C}_t$.

What is GRU?

- **Gated Recurrent Unit** - simplified LSTM variant
- Introduced by Cho et al. in 2014 for machine translation
- Uses **two gates** instead of LSTM's three gates
- **Computationally efficient** with comparable performance
- No separate cell state - hidden state serves dual purpose

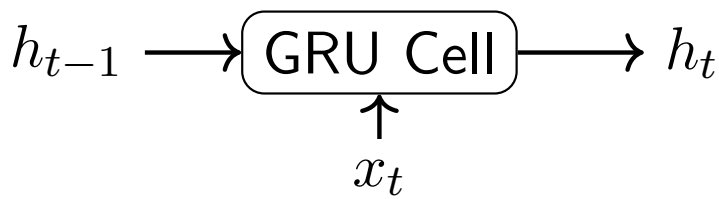


Figure: GRU cell overview

GRU Architecture: The Two Gates

Update Gate (z_t)

- Controls information flow from previous state
- Balances old vs. new information
- Similar to LSTM's input & forget gates combined

Reset Gate (r_t)

- Controls how much past information to forget
- Determines relevance of previous hidden state
- Helps capture short-term dependencies

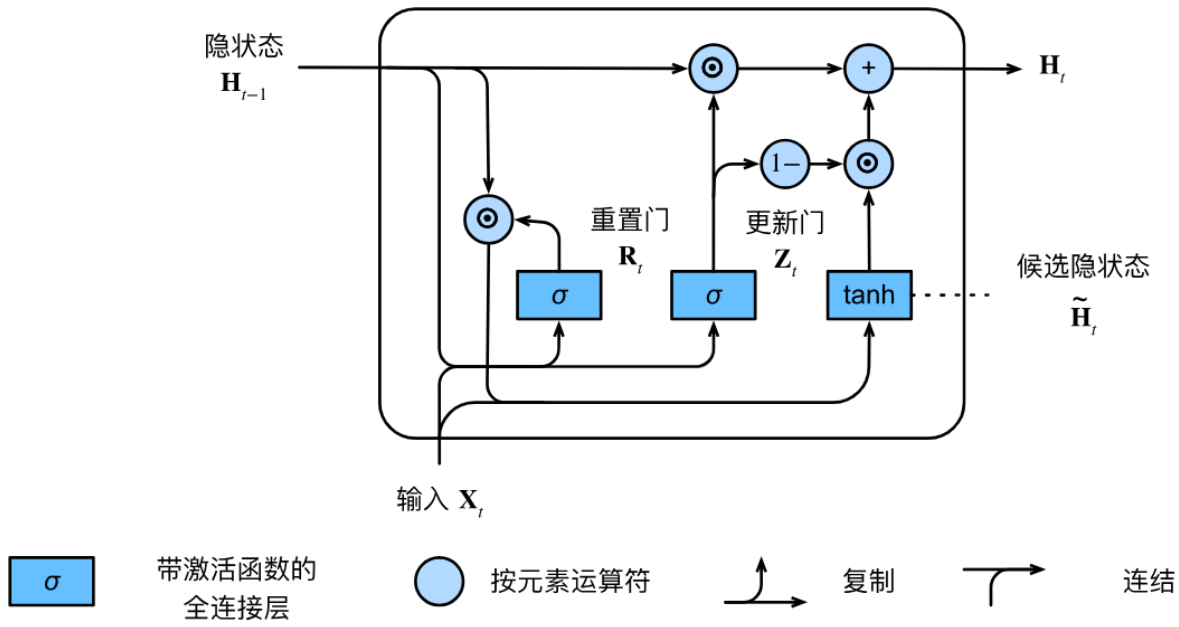


Figure: Structure of an LSTM unit: part 3

GRU Mathematical Formulation I

Gate Equations

$$\text{Update Gate: } z_t = \sigma(W_z \cdot [h_{t-1}, x_t] + b_z)$$
$$\text{Reset Gate: } r_t = \sigma(W_r \cdot [h_{t-1}, x_t] + b_r)$$

Candidate Activation

$$\tilde{h}_t = \tanh(W \cdot [r_t * h_{t-1}, x_t] + b)$$

Final Hidden State Update

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

- σ: Sigmoid activation function (outputs 0-1)

GRU Mathematical Formulation II

- $\tanh$: Hyperbolic tangent (outputs -1 to 1)
- $*$: Element-wise multiplication
- $[a, b]$: Vector concatenation

GRU vs LSTM: Key Differences

GRU Advantages

- **Fewer parameters** (2 gates vs 3 gates)
- **Faster training** and inference
- **Simpler architecture**
- Often works equally well for many tasks

When to Use GRU

- Smaller datasets
- Computational constraints
- Simpler sequence tasks
- When training speed is important

LSTM Advantages

- Separate cell state provides **better memory**
- Often better for **very long sequences**
- More explicit control over information flow
- Better for complex temporal dependencies

When to Use LSTM

- Very long-term dependencies
- Complex sequence patterns
- Large datasets with complex temporal structure

GRU Parameter Comparison

Component	GRU	LSTM
Gates	2	3
Parameters	$3 \times d_h \times (d_h + d_x)$	$4 \times d_h \times (d_h + d_x)$
Memory Mechanism	Hidden state	Cell state + Hidden state
Training Speed	Fast	Moderate
Long-term Memory	Good	Excellent

Table: Comparison of RNN variants

Where:

- d_h: Hidden state dimension
- d_x: Input dimension

Summary: Why Choose GRU?

- **Simpler than LSTM**: Fewer gates, fewer parameters
- **More powerful than simple RNN**: Can learn long-term dependencies
- **Computationally efficient**: Faster training and inference
- **Empirically effective**: Works well for many sequence tasks
- **Easy to implement**: Straightforward architecture

Key Insight

GRU achieves a sweet spot between simplicity and effectiveness by combining the forget and input gates of LSTM into a single update gate, while maintaining the ability to learn long-range dependencies.

Remember

The choice between GRU and LSTM often depends on the specific task, dataset size, and computational resources. Both are excellent choices for sequence modeling.

Two applications:

1. Classify images: treat the image as sequential data.
2. Natural language model.