Week 0d: Neural Networks
The Depth Challenge

Machine Learning for Smarter Innovation

BSc Course - Theory Foundation

October 7, 2025

Course Outline

1. Part 1: The Challenge
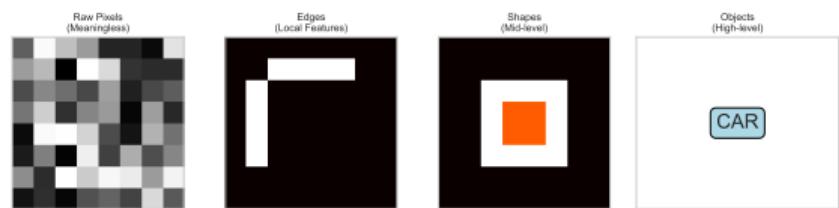
2. Part 2: Shallow MLPs

3. Part 3: Modern Architectures

4. Part 4: Synthesis

- Raw pixels are meaningless noise
- Vision builds up complexity:
  - Edges from pixel gradients
  - Shapes from edge combinations
  - Objects from shape patterns
- **Example: Cat detection**
  - Pixels $\rightarrow$ edges $\rightarrow$ whiskers $\rightarrow$ face $\rightarrow$ cat
- Traditional ML: Manual feature engineering
- Deep learning: Automatic feature hierarchy
- **Problem**: Manual features require domain expertise and fail to generalize

Raw Pixels (Meaningless)   Edges (Local Features)   Shapes (Mid-level)   Objects (High-level)

CAR

Hierarchical feature learning mirrors biological vision - complexity emerges through layered abstraction

The Perceptron (Rosenblatt 1957)
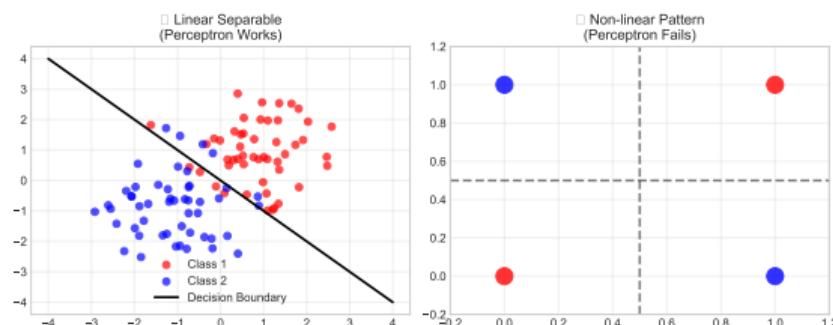
$$y = \text{sign}(w_1 x_1 + w_2 x_2 + b) \qquad (1)$$
$$= \text{sign}(\mathbf{w}^T \mathbf{x} + b) \qquad (2)$$

Learning Rule:

$$w_{new} = w_{old} + \eta(y_{true} - y_{pred})x$$

Geometric Interpretation:

- Creates a linear decision boundary
- Hyperplane in n-dimensional space
- Cannot separate non-linear patterns
- **Perceptron Convergence Theorem**: Guaranteed to converge if data is linearly separable

Single layer = single hyperplane = linear separation only - Rosenblatt's 1957 perceptron could only solve linearly separable problems
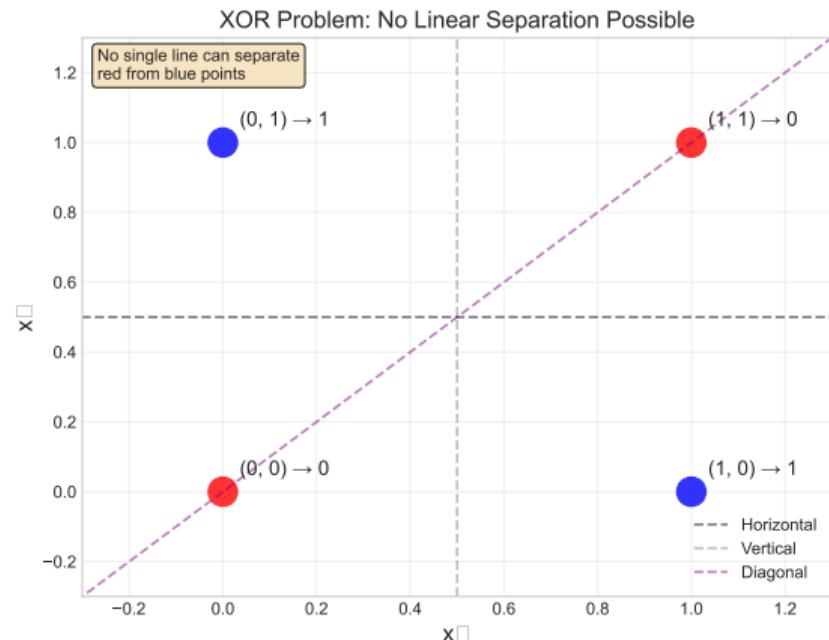
XOR Truth Table:

x_1	x_2	XOR
0	0	0
0	1	1
1	0	1
1	1	0

The Problem:

- No single line separates the classes
- XOR is the simplest non-linear problem (2 inputs, 4 points)
- Requires non-linear decision boundary
- **Minsky & Papert (1969)**: Mathematical proof that perceptrons cannot solve XOR

XOR Problem: No Linear Separation Possible

No single line can separate red from blue points

$(0, 1) \rightarrow 1$ $(1, 1) \rightarrow 0$

$(0, 0) \rightarrow 0$ $(1, 0) \rightarrow 1$

- - - Horizontal
- - - Vertical
- - - Diagonal

XOR became the symbol of perceptron limitations - Minsky & Papert's 1969 proof triggered the first AI winter

4. Universal Approximation Theorem

**Theoretical Foundation (Cybenko 1989, Hornik 1991):**
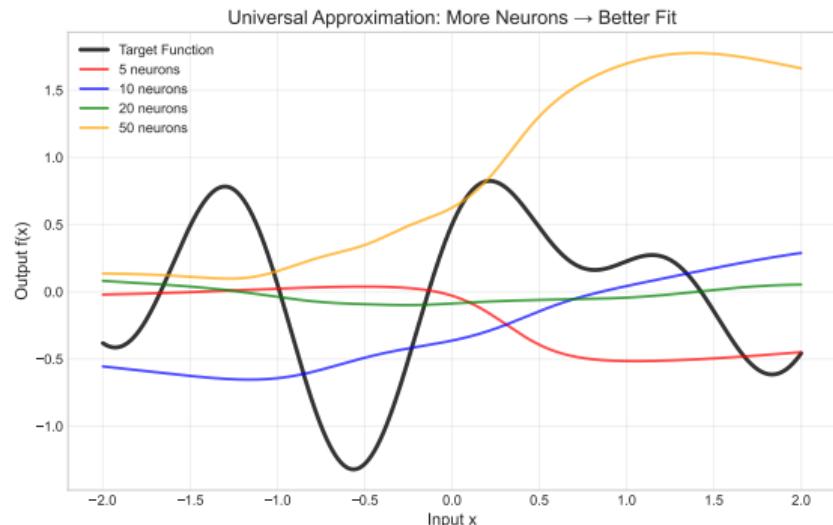
- Any continuous function can be approximated
- Single hidden layer with enough neurons
- Activation: sigmoid, tanh, ReLU
- Arbitrarily small error possible

Mathematical Statement: For any $\epsilon > 0$ and continuous f on compact set K (bounded and closed), there exists network N such that:

$$\sup_{x \in K} |f(x) - N(x)| < \epsilon$$

Theory-Practice Gap:

- May need exponentially many neurons for single layer

Universal Approximation: More Neurons → Better Fit
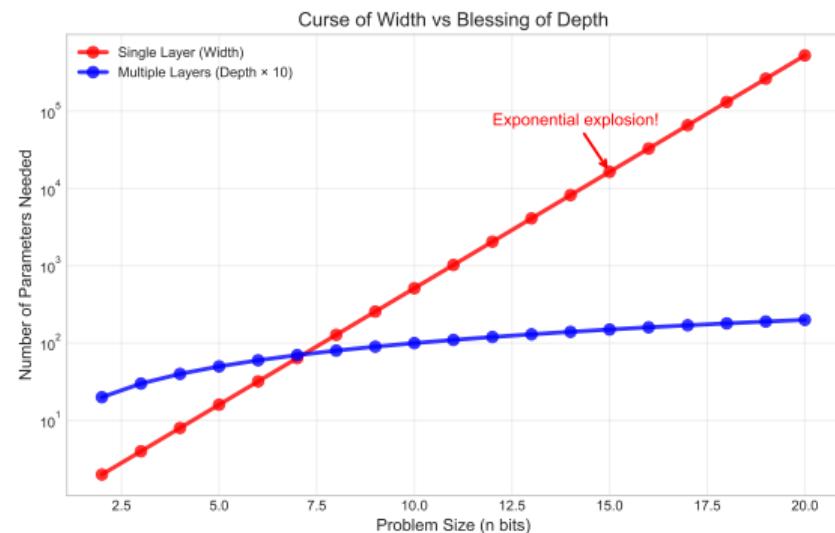
Theory says possible but doesn't tell us how many neurons - this gap motivates depth over width

5. Quantify: How Many Neurons/Layers Needed?

Practical Reality:

- Theory: Single layer sufficient
- Practice: Exponentially many neurons
- **Example 1:** Parity function on n bits
  - 1 layer: 2^{n-1} neurons needed
  - 2 layers: $O(n)$ neurons sufficient
- **Example 2:** Checkerboard pattern
  - 1 layer: $O(2^{\sqrt{n}})$ neurons
  - 2 layers: $O(n)$ neurons
- Curse of width vs. blessing of depth
- **General principle:** Depth allows exponentially more efficient representation

Curse of Width vs Blessing of Depth

Depth provides exponential expressivity advantage - this is why 'deep' learning succeeded where shallow networks failed
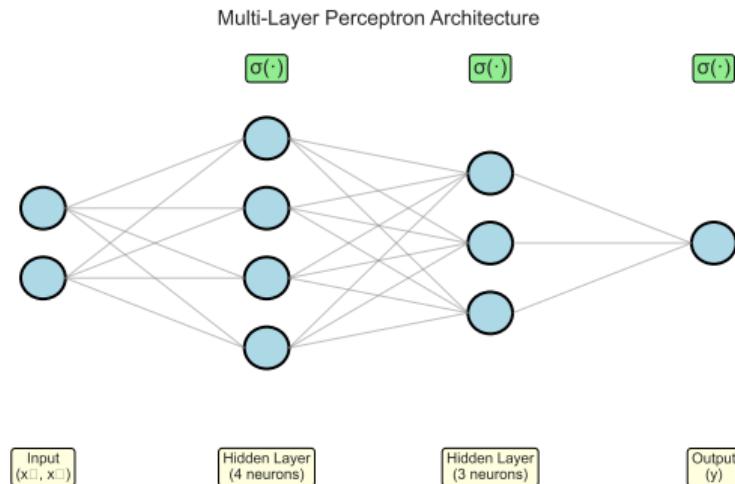
Multi-Layer Perceptron (MLP):

$$\mathbf{h} = \sigma(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1) \quad (3)$$

$$y = \sigma(\mathbf{w}_2^T\mathbf{h} + b_2) \quad (4)$$

Multi-Layer Perceptron Architecture

Key Innovation:

- Hidden layer creates feature combinations
- Hidden layer creates new representation where linear separation becomes possible
- Non-linear activation σ (sigmoid, tanh, ReLU)
- Each neuron = learned feature detector
- Output combines these features
- **Training**: Backpropagation (Rumelhart et al., 1986) computes gradients via chain rule

Hidden layer transforms input space to make linear separation possible - learned features replace manual engineering
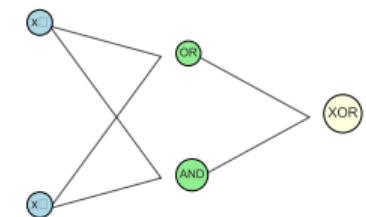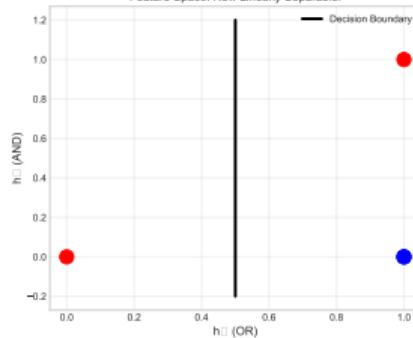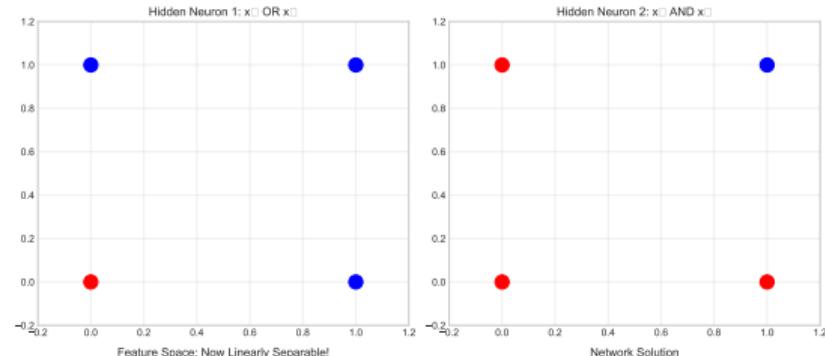
Network Architecture:

- Input: x_1, x_2
- Hidden: 2 neurons with sigmoid
- Output: 1 neuron with sigmoid

Solution Strategy:

- h_1: Detects x_1 OR x_2
- h_2: Detects x_1 AND x_2
- Output: h_1 AND NOT h_2

Actual Weights:

$$h_1 = \sigma(20x_1 + 20x_2 - 10) \tag{5}$$

$$h_2 = \sigma(20x_1 + 20x_2 - 30) \tag{6}$$

What We Gained:

- Non-linear decision boundaries
- Universal approximation
- Automatic feature learning
- Backpropagation training
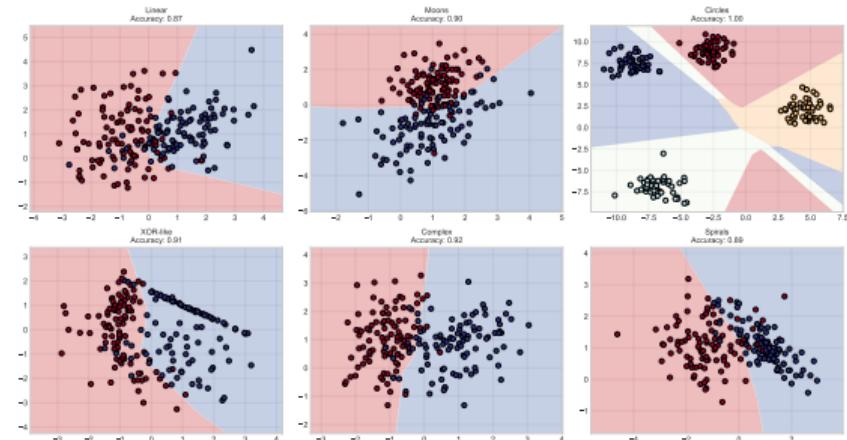
Applications Unlocked:

- Image classification (MNIST 1998: 99.2%)
- Function approximation
- Pattern recognition
- Control systems

Historical Impact:

- 1986-2006: MLPs dominated pattern recognition tasks
- Universal approximation unlocked ~10× more problem classes

MLPs solved the non-linearity problem and enabled the neural network renaissance
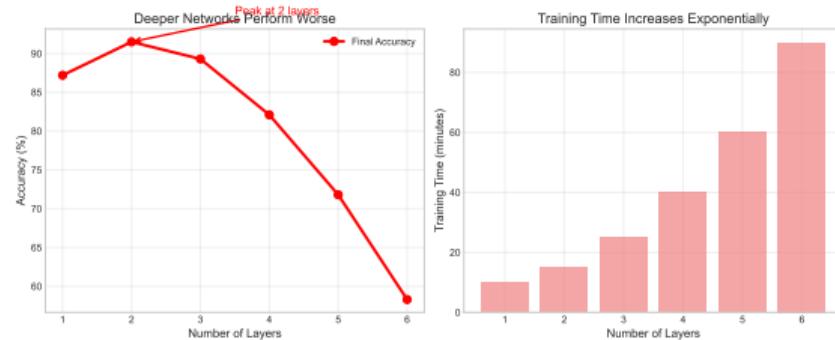
9. FAILURE PATTERN: Vanishing Gradients in Deep Networks

The Problem with Depth:

Layers	Final Accuracy	Training Time
1	87.2%	10 min
2	91.5%	15 min
3	89.3%	25 min
4	82.1%	40 min
5	71.8%	60 min
6	58.3%	90 min

Observation: Deeper networks perform worse, not better!

Real data from 1990s experiments - deeper meant worse performance

Chain Rule in Deep Networks:

$$\frac{\partial L}{\partial W_1} = \frac{\partial L}{\partial y} \cdot \frac{\partial y}{\partial h_n} \cdot \ldots \cdot \frac{\partial h_2}{\partial h_1} \cdot \frac{\partial h_1}{\partial W_1} \tag{8}$$

Sigmoid Derivative:

$$\sigma'(x) = \sigma(x)(1 - \sigma(x)) \tag{9}$$
$$\leq 0.25 \tag{10}$$
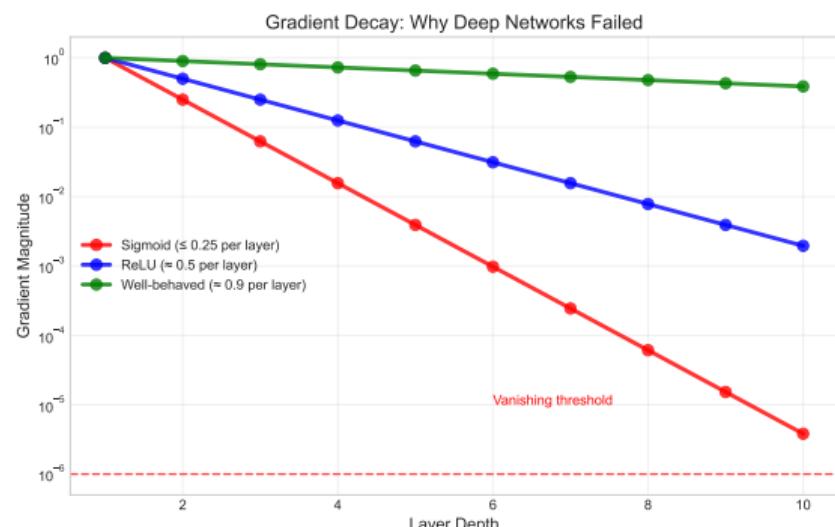
The Problem: Each layer multiplies by ≤ 0.25

- 5 layers: $(0.25)^5 = 0.001$
- 10 layers: $(0.25)^{10} = 0.000001$

Comparison:

- Tanh: $\tanh'(x) = 1 - \tanh^2(x) \leq 1$, still problematic
- **Solution preview**: ReLU has constant gradient of 1 for positive inputs

Gradients vanish exponentially - early layers learn nothing

Gradient Decay: Why Deep Networks Failed

- Sigmoid (≤ 0.25 per layer)
- ReLU (≈ 0.5 per layer)
- Well-behaved (≈ 0.9 per layer)

Vanishing threshold

Mathematical Analysis:

For L-layer network with sigmoid activations:

$$\left| \frac{\partial L}{\partial W_1} \right| \leq C \cdot (0.25)^{L-1}$$

Consequences:
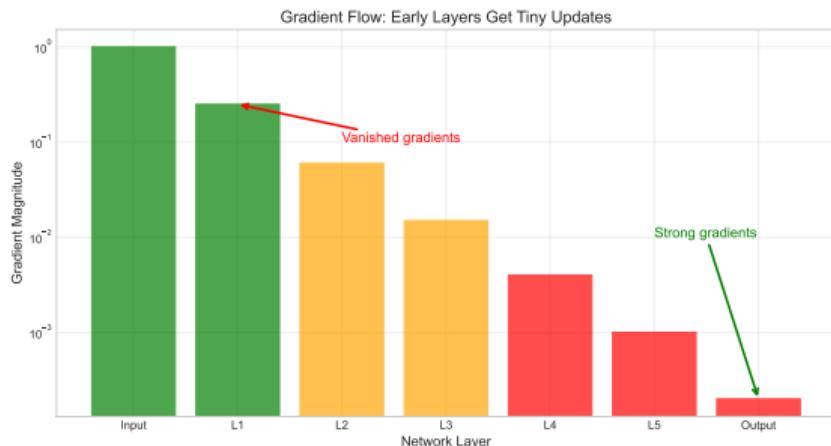
- Early layers: Tiny gradients
- Late layers: Large gradients
- Gradient mismatch problem
- Training becomes impossible

Historical Impact:

- 1990s: "Neural networks don't scale"
- SVMs and ensemble methods dominated
- Deep learning winter until 2006

Solutions (Part 3 Preview):
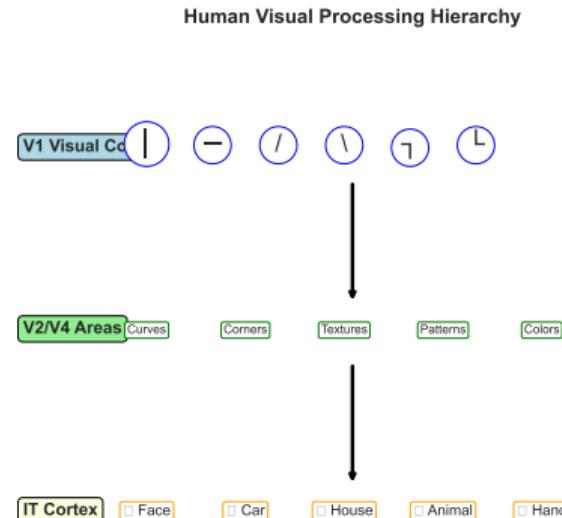
- ReLU activation (constant gradient)
- Batch normalization (stabilize distributions)
- Skip connections (ResNet shortcut paths)
- Better initialization (Xavier, He)

Gradient Flow: Early Layers Get Tiny Updates

How Humans See (Hubel & Wiesel 1959):

- **Level 1**: Edge detection (V1 cortex)
  - Horizontal, vertical, diagonal lines
  - Local contrast detection
- **Level 2**: Texture & shape (V2, V4)
  - Curves, corners, textures
  - Spatial relationships
- **Level 3**: Objects (IT cortex)
  - Faces, cars, animals
  - Invariant recognition
- **Inspiration**: Fukushima's Neocognitron (1980) - precursor to modern CNNs

Human Visual Processing Hierarchy

Hubel & Wiesel (1959 Nobel Prize): Discovered hierarchical processing in cat visual cortex - inspired modern neural network architectures
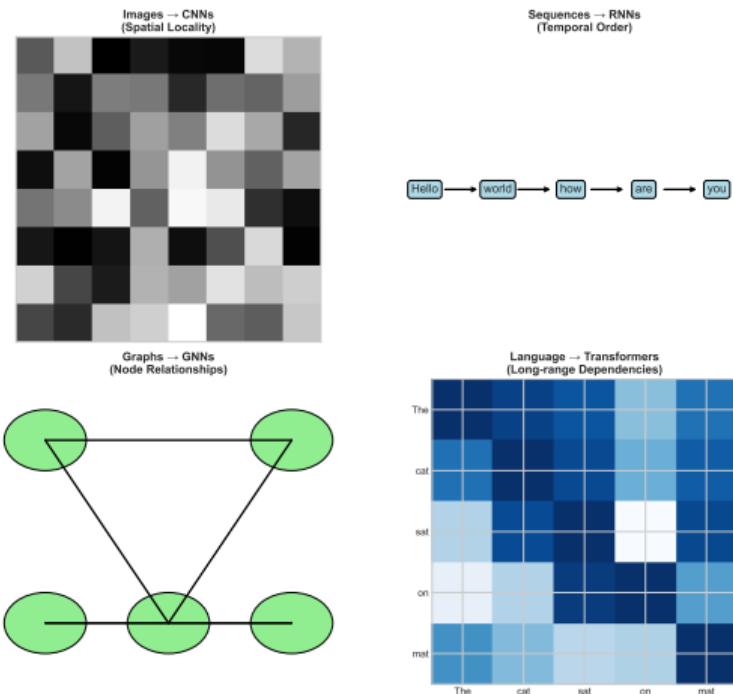
The Key Insight:

- Problem: Generic MLPs ignore data structure
- Solution: Architecture matches inductive bias
- **No Free Lunch Theorem**: No single architecture optimal for all problems
- Architecture choice encodes prior knowledge about problem structure

Examples:

- **Images**: Spatial locality -¿ CNNs
- **Sequences**: Temporal order -¿ RNNs
- **Graphs**: Node relationships -¿ GNNs
- **Language**: Long-range dependencies -¿ Transformers

Images → CNNs
(Spatial Locality)

Sequences → RNNs
(Temporal Order)

Hello → world → how → are → you

Graphs → GNNs
(Node Relationships)

Language → Transformers
(Long-range Dependencies)

Right architecture = built-in prior knowledge about the problem domain
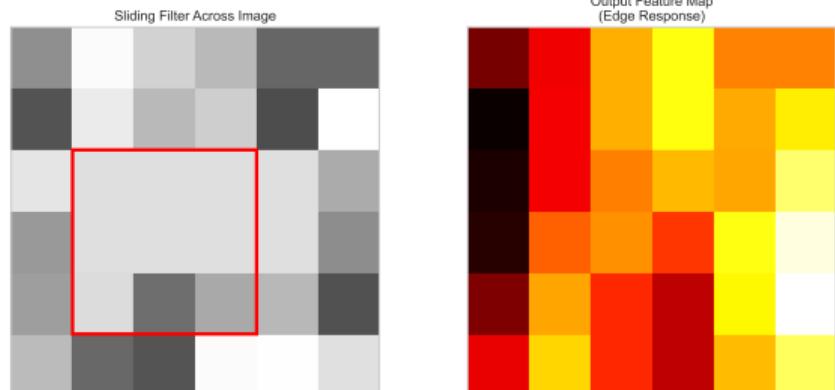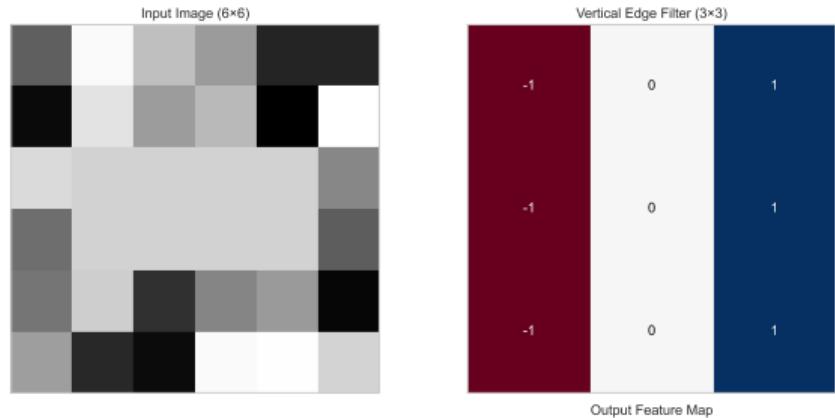
Convolution Intuition:

- Take a small "template" (3x3 filter)
- Slide it across the entire image
- At each position: compute similarity
- Result: "Where is this pattern?"

Example Filters:

- Edge detector: $\begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}$

- Blur: $\frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$

Input Image (6×6)

Vertical Edge Filter (3×3)

Sliding Filter Across Image

Output Feature Map (Edge Response)

Convolution = template matching with learnable templates

15. Geometric Intuition: Filters Detect Edges/Textures

What Filters Learn:

Layer 1: Low-level features

- Edges, corners, blobs
- Oriented lines at different angles
- Color gradients

Layer 2: Mid-level features

- Textures, patterns
- Simple shapes
- Motifs and repeating elements
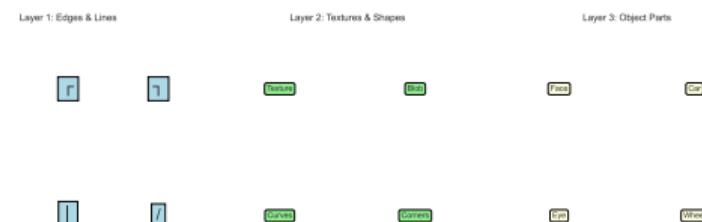
Layer 3+: High-level features

- Object parts (eyes, wheels)
- Complex patterns

Visualization Technique:

- Apply filter to white noise to see what pattern it detects

Each layer builds more complex features from simpler ones

16. CNN Architecture Details

Key Components:

1. Convolutional Layers

- Multiple filters per layer
- Shared weights across spatial locations
- Parameter sharing: Same filter applied to all spatial locations - reduces parameters by 1000x

2. Pooling Layers

- Downsampling (max, average)
- Translation invariance
- Computational efficiency

3. Fully Connected

- Final classification
- Combines all learned features

Example Parameters:

- AlexNet: 60M parameters
- VGG: 138M, ResNet: 25M

CNN = Feature extraction (conv+pool) + Classification (FC)
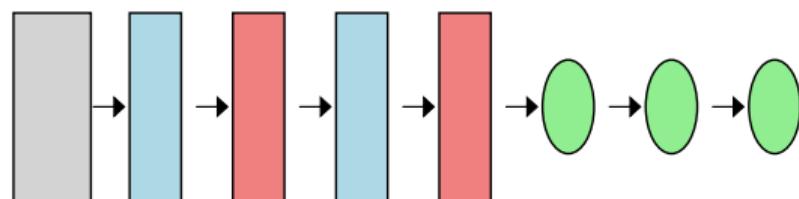
CNN Architecture: Feature Extraction → Classification

| Input 32×32×3 | Conv 28×28×6 | Pool 14×14×6 | Conv 10×10×16 | Pool 5×5×16 | FC 120 | FC 84 | Output 10 |

Example Calculation:

Input (3x3): $\begin{bmatrix} 1 & 2 & 1 \\ 0 & 1 & 2 \\ 1 & 0 & 1 \end{bmatrix}$

Filter (3x3): $\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$

Convolution (element-wise multiply + sum):
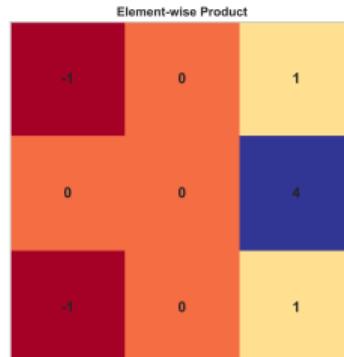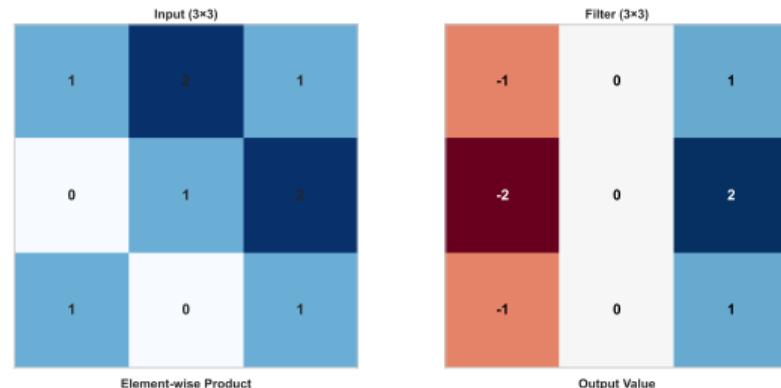
$$= (-1)(1) + (0)(2) + (1)(1) + \qquad (11)$$
$$(-2)(0) + (0)(1) + (2)(2) + \qquad (12)$$
$$(-1)(1) + (0)(0) + (1)(1) \qquad (13)$$
$$= -1 + 0 + 1 - 0 + 0 + 4 - 1 + 0 + 1 \qquad (14)$$
$$= 4 \qquad (15)$$

Input (3×3)

Filter (3×3)

Element-wise Product

Output Value

Sum = 4

High response (4) means vertical edge detected at this location

Recurrent Neural Networks:

$$h_t = \tanh(W_{hh} h_{t-1} + W_{xh} x_t + b_h) \qquad (16)$$

$$y_t = W_{hy} h_t + b_y \qquad (17)$$
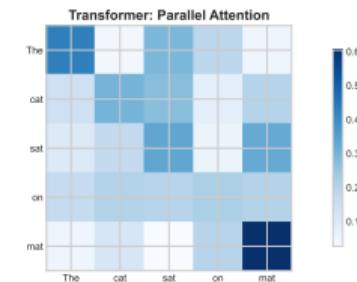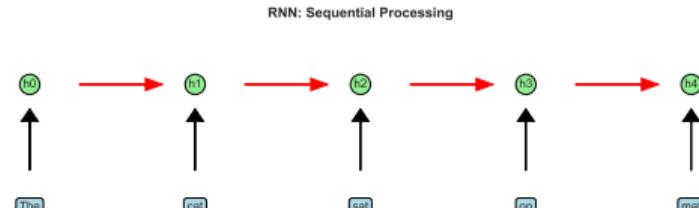
- Hidden state carries memory
- Sequential processing ($O(n)$ path length)
- Good for: Time series, NLP
- Problem: Vanishing gradients over time

Transformers ("Attention Is All You Need" 2017): Attention Mechanism:

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

- Query: What I'm looking for
- Key: What I have to offer
- Value: What I'll return if matched

RNN: Sequential Processing

Transformer: Parallel Attention

Key Advantages:
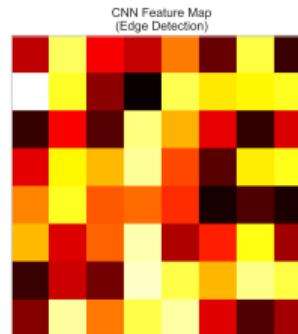
- Self-attention: $O(1)$ path length
- Parallel processing (no sequential bottleneck)
- Long-range dependencies without vanishing gradients
- State-of-the-art for language (GPT, BERT)

Transformers enable O(1) path length vs O(n) for RNNs - revolutionized NLP and beyond
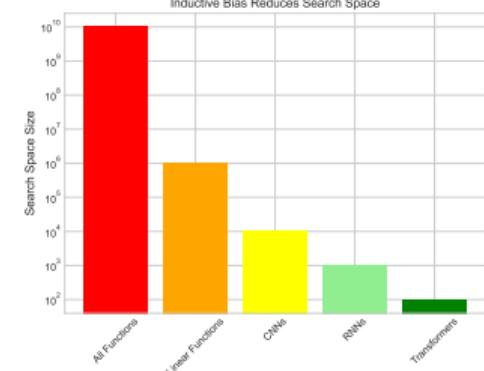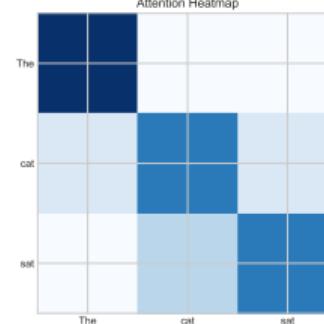
CNN Feature Maps:
- Each filter produces a feature map
- Bright areas = high activation
- Shows what the network "sees"
- Layer 1: Edges and textures
- Layer N: Complex patterns

Transformer Attention:
- Attention weights as heatmaps
- Shows which words influence others
- Different heads learn different patterns
- Interpretable relationships
- **Technique**: Grad-CAM visualizes which regions influenced decision
- Attention weights reveal learned linguistic structure (syntax trees emerge naturally)

CNN Feature Map (Edge Detection)

Multiple Feature Maps (Different Filters)

Attention Heatmap

Inductive Bias Reduces Search Space

Visualization reveals the internal representations learned by neural networks

The Core Principle:
Without Structure:
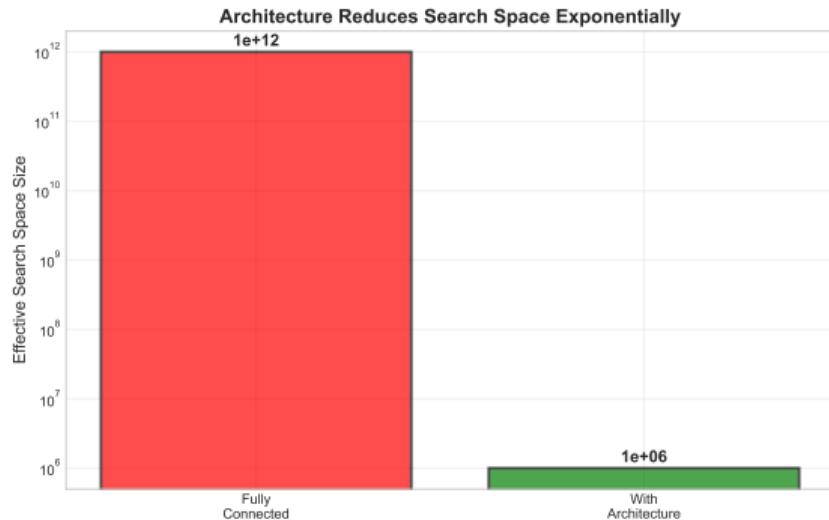
- Search space: All possible functions
- Size: Exponential in parameters
- Sample complexity: Intractable

With Architecture:

- Built-in assumptions about data
- Drastically reduced search space
- Faster learning, better generalization

Examples:

- CNNs assume translation invariance
- RNNs assume sequential dependence
- Transformers assume attention patterns
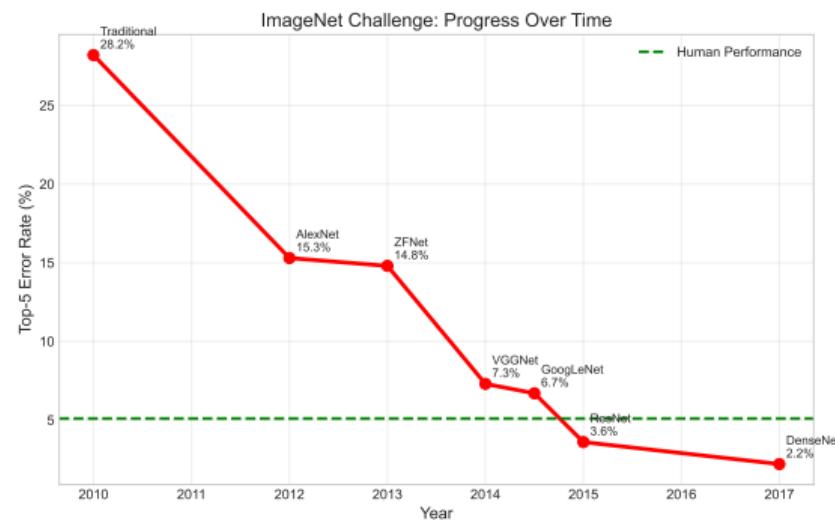
Architecture = built-in prior knowledge that guides learning

Architecture Reduces Search Space Exponentially

Architecture = built-in prior knowledge that guides learning

ImageNet Challenge Results:

Year	Model	Top-5 Error
2010	Traditional CV	28.2%
2012	AlexNet (CNN)	15.3%
2013	ZFNet	14.8%
2014	VGGNet	7.3%
2014	GoogLeNet	6.7%
2015	ResNet	3.6%
2017	DenseNet	2.2%
-	Human Performance	5.1%

ImageNet Challenge: Progress Over Time

ResNet (2015): First to surpass human performance (3.6% vs 5.1%) - CNNs achieved superhuman vision in just 5 years

22. Solutions to Vanishing Gradients

1. ReLU Activation:

$$\text{ReLU}(x) = \max(0, x) \qquad (18)$$

$$\frac{d}{dx}\text{ReLU}(x) = \begin{cases} 1 & x > 0 \\ 0 & x \leq 0 \end{cases} \qquad (19)$$

Comparison:

- Sigmoid: $\sigma'(x) \leq 0.25$ (vanishes)
- ReLU: gradient $= 1$ (constant for $x > 0$)
- Enables training $100+$ layer networks

2. Other Solutions:

- Batch Normalization (2015): Stabilize distributions
- Skip connections (ResNet 2015): Shortcut paths
- Xavier/He initialization: Preserve variance

Gradient Flow Comparison:
Sigmoid Network (10 layers):

- Layer 10 gradient: 1.0
- Layer 5 gradient: 0.001
- Layer 1 gradient: 0.000001

ReLU Network (10 layers):

- Layer 10 gradient: 1.0
- Layer 5 gradient: 1.0
- Layer 1 gradient: 1.0

Result:

- ReLU: All layers learn at similar rates
- Deep networks become trainable
- 2012 breakthrough: AlexNet (8 layers with ReLU)

ReLU's constant gradient solved the vanishing gradient problem - enabling the deep learning revolution

Key Milestones:
2012 - AlexNet:

- CNNs + ImageNet breakthrough
- 8-layer network, ReLU activation
- GPU acceleration

2014 - Sequence-to-Sequence:

- RNNs for machine translation
- Encoder-decoder architecture

2015 - ResNet:
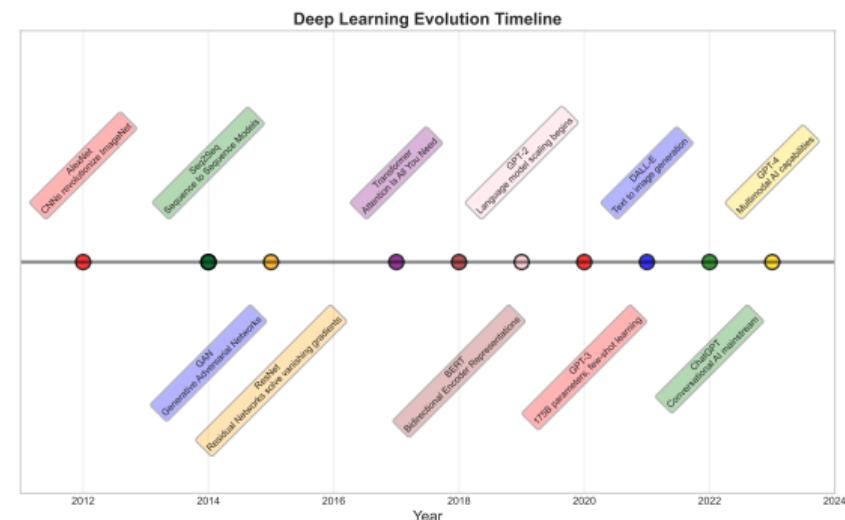
- 152 layers with skip connections
- First to surpass human performance on ImageNet
- Solved vanishing gradient problem

2017 - Transformers:

- "Attention Is All You Need"
- Self-attention mechanism
- Foundation for GPT, BERT

2020 - GPT-3:

- 175 billion parameters

Deep Learning Evolution Timeline

Universal Design Principles:

1. Locality:

- Nearby elements are related
- CNNs: Spatial locality
- RNNs: Temporal locality

2. Hierarchy:

- Build complexity gradually
- Low-level -¿ High-level features
- Mirrors human cognition

3. Invariance:

- Robust to irrelevant changes
- Translation, rotation, scale
- Attention: Permutation invariance

4. Efficiency:
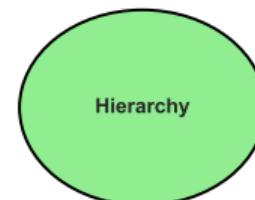
- Parameter sharing
- Computational optimization
- Memory constraints

Good architectures encode the right inductive biases for the domain
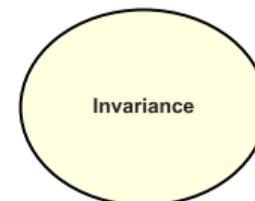
Architecture Design Principles
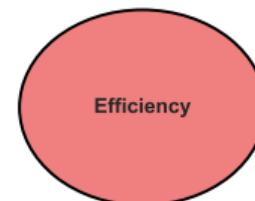
Locality

Nearby elements
are related

Hierarchy

Build complexity
gradually

Invariance

Robust to
irrelevant changes

Efficiency

Parameter sharing
& optimization

Decision Criteria:
Use CNNs when:

- Data has spatial structure (images, video)
- Translation invariance needed
- Local patterns matter
- Examples: Vision, medical imaging

Use RNNs when:

- Sequential dependencies
- Variable-length sequences
- Real-time processing needed
- Examples: Speech, time series

Use Transformers when:

- Long-range dependencies critical
- Parallel processing available
- Sufficient compute budget
- Examples: NLP, multimodal AI

Use Generic MLPs when:

- Tabular data (no structure)
- Small datasets ($<$10k examples)
- Interpretability required
- Examples: Finance, healthcare

Key Rule:
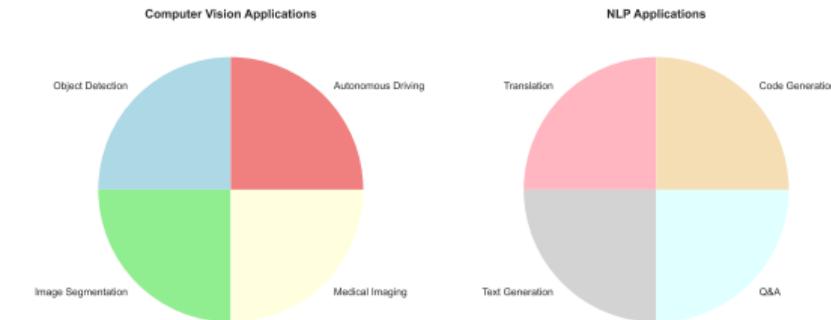
- Match architecture to data structure
- Start simple, add complexity if needed
- More parameters $\neq$ better performance

Architecture choice depends on data structure, computational budget, and domain requirements
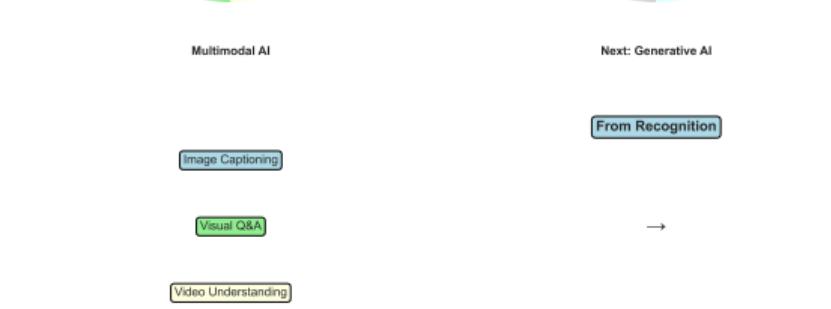
Computer Vision:

- Object detection (YOLO, R-CNN)
- Image segmentation
- Medical imaging diagnosis
- Autonomous driving

Natural Language Processing:

- Machine translation approaching human quality
- Text generation (GPT family)
- Question answering
- Code generation (GitHub Copilot)

Multimodal AI:

- Image captioning
- Visual question answering
- Video understanding
- Robotics integration

Neural networks now match or exceed human performance in many domains

Computer Vision Applications

- Object Detection
- Autonomous Driving
- Image Segmentation
- Medical Imaging

NLP Applications

- Translation
- Code Generation
- Text Generation
- Q&A

Multimodal AI

- Image Captioning
- Visual Q&A
- Video Understanding

Next: Generative AI

From Recognition

$\rightarrow$

To Generation

Pitfall 1: Wrong Architecture
- Using MLP for images (ignores structure)
- Solution: Match architecture to data

Pitfall 2: Too Deep Too Soon
- 100 layers without skip connections
- Solution: Start shallow, add depth incrementally

Pitfall 3: Poor Initialization
- All weights $= 0$ (symmetry breaking fails)
- Solution: Xavier/He initialization

Pitfall 4: Ignoring Overfitting
- Training accuracy 99%, test 60%
- Solution: Dropout, regularization, augmentation

Pitfall 5: Wrong Learning Rate
- Too high: Divergence, too low: No learning
- Solution: LR schedules, adaptive optimizers

Pitfall 6: Insufficient Data
- Deep networks need 1000s of examples
- Solution: Transfer learning, data augmentation

Most neural network failures stem from architecture mismatch, poor initialization, or insufficient data

26. Summary & Preview: Generative AI

What We Learned:
- Perceptrons: Linear limitations
- MLPs: Non-linear but shallow
- Deep networks: Vanishing gradients
- Modern architectures: Structured solutions

Key Insights:
- Architecture matters more than size
- Three breakthroughs: ReLU activation, specialized architectures, skip connections
- Depth provides exponential expressivity advantage

Next: Generative AI
- From classification (is this a cat?) to generation (draw me a cat)
- VAEs, GANs, Diffusion models
- Large language models
- Applications in innovation

Neural networks: From solving XOR to generating Shakespeare