

Teaching Machines to See Patterns

A Neural Networks Primer: Why We Needed Each Piece of the Puzzle

NLP Course 2025

From the 1950s mail sorting crisis to ChatGPT: How humanity taught machines to think

1950s: The Mail Sorting Crisis

The Challenge:

- 150 million letters per day
- Hand-written addresses
- Human sorters: slow, expensive, error-prone
- Traditional programming: useless

Why Traditional Code Failed:

- Can't write rules for every handwriting style
- Too many variations of each letter
- Context matters: "l" vs "l" vs "1"
- This wasn't computation—it was **pattern recognition**

This problem would take 40 years to solve properly

Why Can't We Just Write Rules?

Problem: Recognize the Letter "A"

Traditional Approach (Failed):

```
if (has_triangle_top AND  
    has_horizontal_bar AND  
    two_diagonal_lines) {  
    return "A"  
}
```

But what about...

- Handwritten A's?
- Different fonts?
- Rotated A's?
- Partial A's?

The Challenge: Infinite Variations of "A"

Four standard capital letter 'A' characters, each consisting of a triangle at the top, a horizontal bar across the middle, and two diagonal lines connecting them.Four variations of the letter 'A': a standard capital 'A', a lowercase 'a', a rotated 'A', and a partial 'A'.

Just for the letter "A", we'd need thousands of rules!

The breakthrough: What if machines could learn patterns like children do?

The Birth of Computational Neuroscience

The Revolutionary Paper:

- "A Logical Calculus of Ideas Immanent in Nervous Activity"
- First mathematical model of neurons
- Proved: Networks can compute ANY logical function
- Inspired von Neumann's computer architecture

Key Insight:

- Neurons = Logic gates
- Brain = Computing machine
- Thinking = Computation

The Model:

- Binary neurons (0 or 1)
- Threshold activation
- Fixed connections
- No learning yet!

Historical Impact:

- Founded field of neural networks
- Influenced cybernetics movement
- Set stage for AI research
- "The brain is a computer" metaphor

14 years later, Rosenblatt would add the missing piece: learning

1957: The First Learning Machine - The Perceptron

Frank Rosenblatt's Radical Idea: Neurons That Learn

Beyond McCulloch-Pitts:

- Adjustable weights (not fixed!)
- Learning from mistakes
- Physical machine built (Mark I)
- Could recognize simple patterns

The Hardware:

- 400 photocells (20×20 "retina")
- 512 motor-driven potentiometers
- Weights adjusted by electric motors
- Took 5 minutes to learn patterns

Mathematical Model:

- Inputs: x_1, x_2, \dots, x_n
- Weights: w_1, w_2, \dots, w_n
- Sum: $z = \sum_{i=1}^n w_i x_i + b$
- Output: $y = \begin{cases} 1 & \text{if } z > 0 \\ 0 & \text{if } z \leq 0 \end{cases}$

Learning Rule: If wrong: $w_i = w_i + \eta \cdot \text{error} \cdot x_i$

The New York Times, 1958: "The Navy revealed the embryo of an electronic computer that will be able to walk, talk, see, write, reproduce itself and be conscious of its existence."

Making It Concrete: Teaching OR Logic

Problem: Learn OR function (output 1 if ANY input is 1)

Training Data:

x_1	x_2	Output
0	0	0
0	1	1
1	0	1
1	1	1

The Perceptron:

$$z = w_1 \cdot x_1 + w_2 \cdot x_2 + b$$

$$\text{output} = \begin{cases} 1 & \text{if } z > 0 \\ 0 & \text{if } z \leq 0 \end{cases}$$

Learning Process:

- ➊ Start with random weights
- ➋ For each example:
 - Calculate output
 - If wrong: adjust weights
 - If correct: keep weights
- ➌ Repeat until all correct

Final Solution: $w_1 = 1, w_2 = 1, b = -0.5$

Success! But this was just the beginning...

Understanding the Notation

Breaking Down the Math Symbols

Inputs and Weights:

- x_i = input value (what we see)
- w_i = weight (importance/strength)
- b = bias (threshold adjuster)

The Computation:

$$z = \sum_{i=1}^n w_i x_i + b$$

This means:

- Multiply each input by its weight
- Add them all up
- Add the bias

This simple math would evolve into deep learning

Real Example:

Should I go outside?

Factor	Value	Weight
Sunny?	1	+2
Raining?	0	-3
Weekend?	1	+1

$$z = (1 \times 2) + (0 \times -3) + (1 \times 1) = 3$$

Decision: $z > 0$, so YES!

1969: The Crisis - XOR Problem

Minsky & Papert's Devastating Discovery

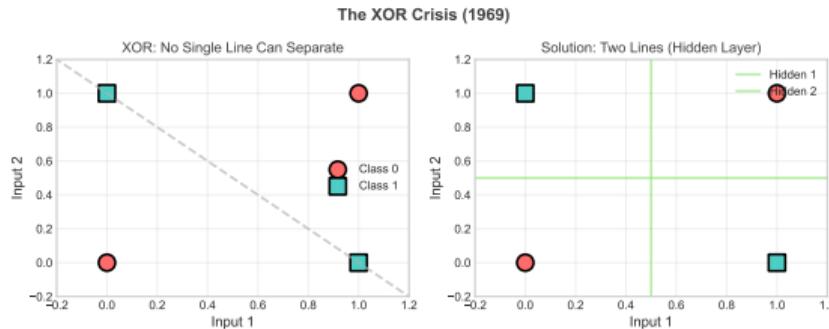
XOR (Exclusive OR):

x_1	x_2	Output
0	0	0
0	1	1
1	0	1
1	1	0

The Problem:

- Can't draw a single line to separate
- Perceptron only learns linear boundaries
- Real-world problems are non-linear!

The field would be dormant for over a decade...



Impact:

- Funding dried up
- "AI Winter" begins
- Neural networks abandoned

Act II: The Journey Back

1980s: The Hidden Layer Revolution

The Insight:

- Stack multiple layers!
- First layer: detect simple features
- Hidden layer: combine features
- Output layer: final decision

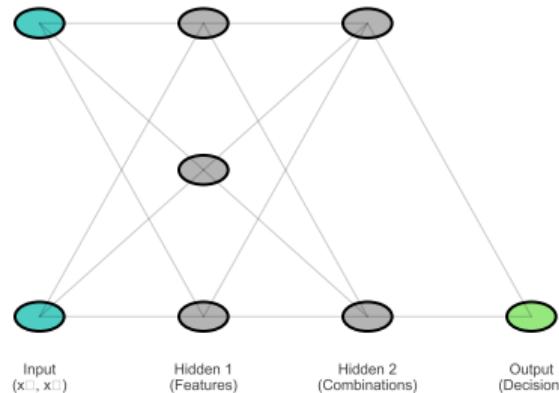
Solving XOR:

- Hidden neuron 1: Is it (0,1)?
- Hidden neuron 2: Is it (1,0)?
- Output: OR of hidden neurons

Multi-Layer Network: Solving Complex Problems

Information Flow →

Each connection has a weight (w), each node has a bias (b)



New Architecture:

- Input layer: raw data
- Hidden layer(s): feature extraction

The Credit Assignment Problem: Who's to Blame?

The Challenge:

- Network makes error at output
- Many neurons contributed
- Which weights should change?
- By how much?

The Solution: Chain Rule

- Calculate error at output
- Propagate error backwards
- Each layer gets its "share of blame"
- Adjust weights proportionally

Mathematical Insight:

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial \text{out}_j} \cdot \frac{\partial \text{out}_j}{\partial \text{net}_j} \cdot \frac{\partial \text{net}_j}{\partial w_{ij}}$$

In Simple Terms:

- ➊ How wrong were we? (Error)
- ➋ How sensitive is error to this weight?
- ➌ Adjust weight in opposite direction
- ➍ Repeat for all weights, back to front

This algorithm is still the foundation of all deep learning today

1987: NetTalk - Networks Learn to Speak

Sejnowski & Rosenberg's Speaking Network

The Challenge:

- English pronunciation is irregular
- "though" vs "through" vs "tough"
- Rule-based systems failed
- Can a network learn from examples?

The Architecture:

- Input: 7 letters (context window)
- Hidden: 80 neurons
- Output: 26 phonemes
- 18,000 total weights

The Results:

- Started: Random babbling
- After 10 epochs: Consonants/vowels
- After 30 epochs: Simple words
- After 50 epochs: 95% correct!

Why It Mattered:

- Proved backprop works on real problems
- Learned complex, irregular mappings
- No rules programmed!
- Sounded like a child learning to read

The network literally learned English pronunciation overnight

Cybenko's Theorem: Networks Can Learn ANY Function

The Theorem: "A feedforward network with:

- One hidden layer
- Finite neurons
- Sigmoid activation

can approximate ANY continuous function to arbitrary accuracy"

What This Means:

- Neural networks are universal
- Can solve any pattern recognition
- Just need enough neurons
- Mathematics guarantees it!

The Catch:

- Doesn't say HOW MANY neurons
- Doesn't say HOW to find weights
- Might need exponentially many
- Training might take forever

Historical Impact:

- Ended theoretical doubts
- Justified deep learning research
- Shifted focus to practical training
- "We know it's possible, now make it work"

This theorem convinced skeptics that neural networks were worth pursuing

Why Linear Doesn't Work: Activation Functions

The Need for Non-Linearity

Problem with Linear:

- Stack of linear layers = still linear!
- $f(g(x)) = (wx + b_1)w' + b_2 = w'wx + \dots$
- Can't learn complex patterns

Solution: Activation Functions

- Add non-linearity after each layer
- Allows learning complex boundaries
- Different functions for different needs

Common Activation Functions:

- **Sigmoid:** $\sigma(x) = \frac{1}{1+e^{-x}}$
 - Smooth, outputs 0-1
 - Good for probabilities
- **ReLU:** $f(x) = \max(0, x)$
 - Simple, fast
 - Solves vanishing gradient
- **Tanh:** $\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$
 - Outputs -1 to 1
 - Zero-centered

ReLU's simplicity revolutionized deep learning in 2011

Visualizing Learning: 2D Classification

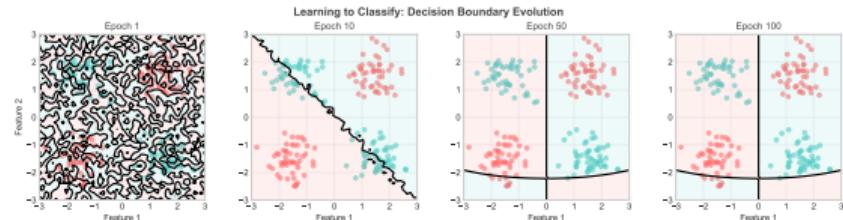
Teaching a Network to Separate Red from Blue Points

The Setup:

- Input: (x, y) coordinates
- Output: Red or Blue class
- Network: $2 \rightarrow 4 \rightarrow 2$ neurons

Training Process:

- ① Epoch 1: Random boundary
- ② Epoch 10: Rough separation
- ③ Epoch 50: Good boundary
- ④ Epoch 100: Perfect fit



What Each Layer Learns:

- Layer 1: Simple boundaries
- Hidden: Combine boundaries
- Output: Final decision

This same principle scales to millions of parameters

1998-2012: From Digits to ImageNet

1998 - LeNet: First Success

- Yann LeCun's CNN for digits
- 32×32 pixels → 10 classes
- 60,000 parameters
- Banks adopt for check reading

Key Innovation: Convolutions

- Share weights across image
- Detect features anywhere
- Build complexity layer by layer

2012 - AlexNet: The Revolution

- 1000 ImageNet classes
- 60 million parameters
- GPUs enable training
- Error rate: 26% → 16%

What Changed:

- Big Data (millions of images)
- GPU computing (100x faster)
- ReLU activation
- Dropout regularization

This victory ended the second AI winter permanently

The Convolution Innovation: See Like Humans Do

How We Actually Recognize Objects

Human Vision Process:

- ① Detect edges
- ② Find shapes
- ③ Identify parts
- ④ Recognize object

CNN Mimics This:

- Layer 1: Edge detectors
- Layer 2: Corner/curve detectors
- Layer 3: Part detectors
- Layer 4: Object detectors

This is why CNNs dominate computer vision



Key Insight:

- A "wheel detector" works anywhere in image
- Share the same detector across positions
- Reduces parameters dramatically
- Makes network translation-invariant

The Mathematics of Learning: Gradient Descent

Finding the Best Weights: Like Hiking Down a Mountain

What is Gradient Descent?

Gradient Descent is an optimization algorithm used to find the minimum of a function.

It starts at a random point on the surface and iteratively moves in the direction of the steepest descent until it reaches the minimum.

Common applications include:

- Minimizing error in linear regression

- Maximizing probability in logistic regression

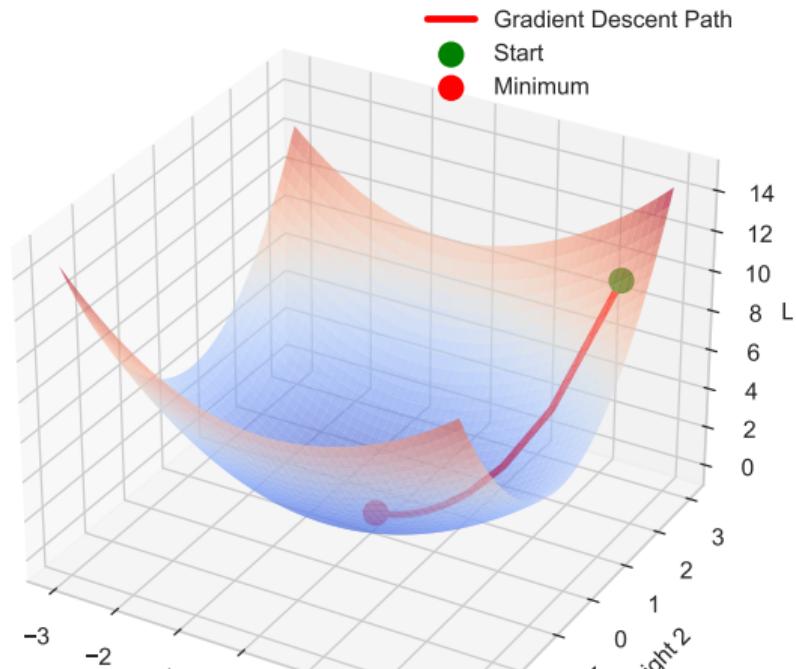
The Optimization Problem:

- Millions of weights to adjust
- Each affects the error
- Need to find best combination

Gradient Descent:

- ① Calculate error (loss)
- ② Find slope (gradient) for each weight
- ③ Step downhill: $w = w - \alpha \cdot \nabla L$
- ④ Repeat until bottom

Gradient Descent: Finding the Lowest Point



Types of Learning: Different Problems, Different Approaches

Supervised Learning:

- Have input-output pairs
- Learn mapping function
- Examples: Classification, Regression

Unsupervised Learning:

- Only have inputs
- Find patterns/structure
- Examples: Clustering, Compression

Reinforcement Learning:

- Learn through trial/error
- Maximize reward signal
- Examples: Games, Robotics

Self-Supervised (Modern):

- Create labels from data itself
- Predict next word, masked words
- Examples: GPT, BERT

Self-supervised learning powers all modern language models

The Overfitting Problem: When Learning Goes Too Far

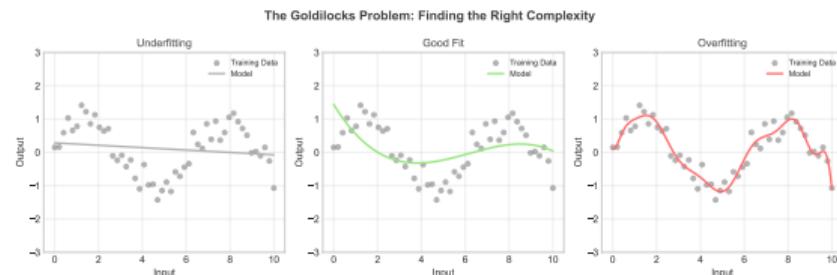
Memorization vs. Understanding

The Problem:

- Network memorizes training data
- Fails on new, unseen data
- Like student memorizing answers

Signs of Overfitting:

- Training accuracy: 99%
- Test accuracy: 60%
- Complex decision boundaries
- High variance



Solutions:

- **More data:** Can't memorize everything
- **Dropout:** Randomly disable neurons
- **Regularization:** Penalize complexity
- **Early stopping:** Stop before overfitting

"With four parameters I can fit an elephant, with five I can make him wiggle his trunk" - von Neumann

2014-Present: Networks That Changed the World

The Depth Revolution:

- 2014 - VGGNet: 19 layers
- 2015 - ResNet: 152 layers
- 2017 - Transformers: Attention
- 2020 - GPT-3: 175B parameters

Why Depth Matters:

- Each layer = abstraction level
- Deep = complex reasoning
- Hierarchical feature learning

Real-World Impact:

- **Vision:** Self-driving cars
- **Language:** Google Translate
- **Speech:** Siri, Alexa
- **Medicine:** Disease diagnosis
- **Science:** Protein folding

The Scale:

- Billions of parameters
- Trained on internet-scale data
- Months of GPU time
- Emergent abilities appear

We went from recognizing digits to passing the bar exam in 25 years

2015: ResNet - The Skip Connection Revolution

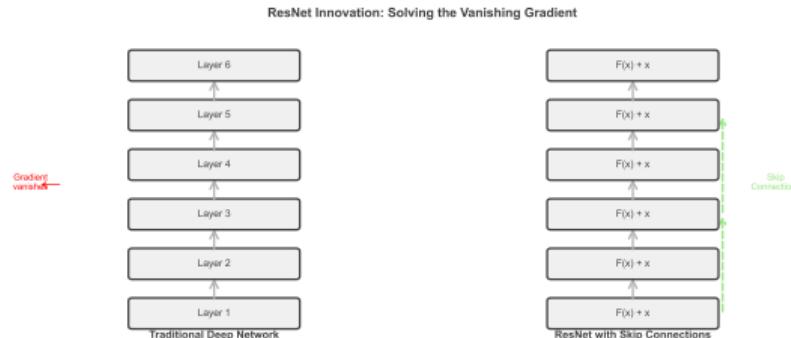
Problem: Networks Couldn't Get Deeper

The Vanishing Gradient:

- Gradients multiply through layers
- Become exponentially small
- Deep layers stop learning
- 20 layers was the limit

The Breakthrough: Skip Connections

- Add input directly to output
- $F(x) + x$ instead of just $F(x)$
- Gradients flow directly backward
- Can train 1000+ layers!



Why It Works:

- Learn residual (difference) only
- Identity mapping is easy default
- Gradients have direct path
- Each layer refines previous result

This simple trick enabled the deep learning revolution

Batch Normalization: Keeping Networks Stable

The Internal Covariate Shift Problem

The Issue:

- Each layer's input distribution changes
- As previous layers update
- Makes learning unstable
- Requires tiny learning rates

The Solution:

- Normalize inputs to each layer
- Mean = 0, Variance = 1
- Learn scale and shift parameters
- Apply during training and testing

BatchNorm Algorithm:

$$\mu_B = \frac{1}{m} \sum_{i=1}^m x_i$$

$$\sigma_B^2 = \frac{1}{m} \sum_{i=1}^m (x_i - \mu_B)^2$$

$$\hat{x}_i = \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}$$

$$y_i = \gamma \hat{x}_i + \beta$$

Benefits:

- 10x faster training
- Higher learning rates OK
- Less sensitive to initialization
- Acts as regularization

Most Network Weights Don't Matter!

The Discovery:

- Networks contain "winning tickets"
- Subnetworks that train well alone
- 90-95% of weights can be removed
- Performance stays the same!

The Hypothesis: "Dense networks succeed because they contain sparse subnetworks that are capable of training effectively"

A 1 billion parameter model might only need 50 million

Implications:

- We massively overparameterize
- Training finds the needle in haystack
- Future: Train small from start?
- Mobile deployment possible

Why It Matters:

- Explains why big networks train better
- Pruning after training works
- Efficiency revolution starting
- Changes how we think about learning

The Right Architecture for the Right Problem

What Are Inductive Biases?

- Assumptions built into architecture
- Guide learning toward solutions
- Trade flexibility for efficiency
- "Priors" about the problem

Examples:

- **CNN:** Spatial locality matters
- **RNN:** Order/time matters
- **GNN:** Graph structure matters
- **Transformer:** All positions can interact

Why They Matter:

- Reduce search space
- Faster convergence
- Better generalization
- Less data needed

The Tradeoff:

- Right bias = 10x better
- Wrong bias = 10x worse
- General architectures = safe but slow
- Specialized = fast but limited

Choosing the right inductive bias is still an art

Capabilities That Appear Suddenly with Scale

The Phenomenon:

- Small models: Can't do task at all
- Medium models: Still can't
- Large models: Suddenly can!
- No gradual improvement

Examples:

- 3-digit arithmetic ($\sim 10B$ params)
- Chain-of-thought reasoning ($\sim 50B$)
- Code generation ($\sim 20B$)
- Multilingual translation ($\sim 100B$)

Why It Happens:

- Complex patterns need capacity
- Phase transitions in learning
- Composition of simpler abilities
- "Grokking" - sudden understanding

Implications:

- We can't predict what's next
- Scaling might unlock AGI
- Or hit fundamental limits
- Active area of research

GPT-3 showed abilities nobody expected or programmed

Neural Network Architectures: Right Tool for Right Job

Feedforward Networks:

- Information flows forward only
- Fixed-size input and output
- Good for: Classification, regression

Convolutional (CNN):

- Spatial feature detection
- Translation invariance
- Good for: Images, video

Recurrent (RNN):

- Process sequences
- Maintain memory/state
- Good for: Text, time-series

Transformer:

- Attention mechanism
- Parallel processing
- Good for: Language, everything else

Each architecture encodes different assumptions about the data

Modern Training: Standing on Shoulders of Giants

Transfer Learning:

- Start with pre-trained network
- Fine-tune on your task
- 100x less data needed
- Days → Hours training

Data Augmentation:

- Create variations of training data
- Rotations, crops, color shifts
- Prevents overfitting
- Free performance boost

Advanced Optimizers:

- **SGD:** Basic gradient descent
- **Momentum:** Remember past gradients
- **Adam:** Adaptive learning rates
- **AdamW:** With weight decay

Mixed Precision:

- Use 16-bit floats where possible
- Keep 32-bit for critical ops
- 2-3x speedup
- Same accuracy

These techniques make deep learning practical for everyone

Why Deep Learning Exploded Now: The Perfect Storm

1. Data Explosion:

- Internet = infinite training data
- ImageNet: 14M labeled images
- Common Crawl: 300TB of text
- YouTube: 500 hours/minute

2. Hardware Revolution:

- GPUs: 100x faster than CPUs
- TPUs: Built for neural nets
- Cloud computing: Rent supercomputers
- Mobile chips with NPUs

3. Algorithm Breakthroughs:

- ReLU activation (2011)
- Batch normalization (2015)
- Skip connections (2015)
- Attention mechanism (2017)

4. Open Source Culture:

- TensorFlow, PyTorch free
- Pre-trained models shared
- Papers with code
- Collaborative research

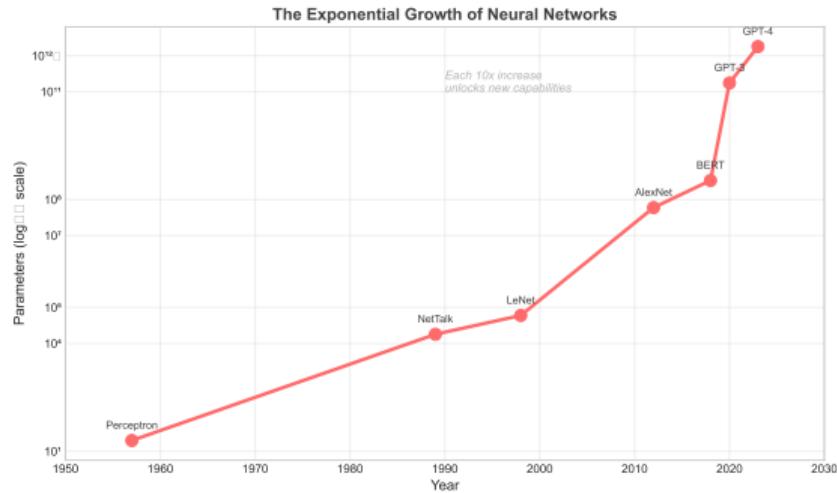
The same ideas from 1980s finally had the resources to work

Understanding Scale: From Perceptron to GPT-4

The Exponential Growth of Neural Networks

Parameter Growth:

- 1957 Perceptron: 20 weights
- 1987 NetTalk: 18,000
- 1998 LeNet: 60,000
- 2012 AlexNet: 60 million
- 2018 BERT: 340 million
- 2020 GPT-3: 175 billion
- 2023 GPT-4: 1.8 trillion



What Scale Brings:

- Emergent abilities
- Zero-shot learning
- Multi-task capability

From Theory to Practice: Your First Network

Building a Digit Classifier in 10 Lines

PyTorch Implementation:

```
import torch
import torch.nn as nn

class SimpleNet(nn.Module):
    def __init__(self):
        super().__init__()
        self.fc1 = nn.Linear(784, 128)
        self.fc2 = nn.Linear(128, 10)

    def forward(self, x):
        x = torch.relu(self.fc1(x))
        return self.fc2(x)

# Train
model = SimpleNet()
optimizer = torch.optim.Adam(model.parameters())
criterion = nn.CrossEntropyLoss()
```

This simple network achieves 97% accuracy on MNIST

What This Does:

- Input: 28×28 pixel image
- Hidden: 128 neurons
- Output: 10 digit classes
- Activation: ReLU
- Training: Adam optimizer

Training Loop:

- Forward pass
- Calculate loss
- Backward pass
- Update weights
- Repeat

When Things Go Wrong (They Always Do)

Gradient Issues:

- **Exploding:** Gradients \rightarrow infinity
 - Solution: Gradient clipping
- **Vanishing:** Gradients \rightarrow 0
 - Solution: Better initialization, ReLU
- **Dead ReLU:** Neurons never activate
 - Solution: LeakyReLU, smaller learning rate

Debugging Tools:

- TensorBoard: Visualize training
- Gradient histograms
- Activation distributions
- Weight evolution plots

Common Failure Modes:

- Loss not decreasing: Learning rate
- Loss NaN: Numerical instability
- Oscillating loss: LR too high
- Plateau: Local minimum or LR too small

Sanity Checks:

- ➊ Overfit single batch first
- ➋ Check gradient flow
- ➌ Visualize first layer filters
- ➍ Plot loss curves
- ➎ Test on toy problem

"If it's not working, it's always the learning rate" - Andrej Karpathy

Common Pitfalls: Learn from Others' Mistakes

Data Problems:

- Not enough data
- Unbalanced classes
- Data leakage
- No validation set

Architecture Issues:

- Too deep without skip connections
- Wrong activation functions
- Incorrect output layer
- Bad initialization

Training Mistakes:

- Learning rate too high/low
- No normalization
- Overfitting ignored
- Wrong loss function

Debugging Tips:

- Start simple, add complexity
- Overfit single batch first
- Monitor gradients
- Visualize predictions

"It's not working" usually means one of these issues

The Future: What's Next?

Current Frontiers:

- Multimodal models (text+image+audio)
- Efficient models for phones
- Neuromorphic hardware
- Quantum neural networks

Unsolved Problems:

- True reasoning ability
- Learning from few examples
- Explaining decisions
- Energy efficiency

Next Breakthroughs?

- Models that update continuously
- Networks that program themselves
- Biological-digital hybrids
- AGI (Artificial General Intelligence)?

Your Role:

- This field is 70 years young
- Major breakthroughs every 2-3 years
- Anyone can contribute
- The best is yet to come

"We're still in the steam engine era of AI" - Geoffrey Hinton

The Journey So Far

Core Concepts:

- ① **Neurons:** $y = f(\sum w_i x_i + b)$
- ② **Learning:** Adjust weights to minimize error
- ③ **Depth:** Each layer adds abstraction
- ④ **Backpropagation:** Distribute error backwards
- ⑤ **Non-linearity:** Enables complex functions

Historical Lessons:

- ① Every limitation spawned innovation
- ② Simple ideas + scale = revolution
- ③ Biology inspires but doesn't limit
- ④ Persistence pays (40-year problem!)
- ⑤ We're just getting started

Remember: Neural networks are just functions that learn from examples

Next: RNNs - Teaching networks to remember

From Here to RNNs and Beyond

What You Now Understand:

- Why traditional programming failed for pattern recognition
- How neurons compute: inputs → weights → sum → activation → output
- Why we need multiple layers and non-linearity
- How networks learn through backpropagation
- The historical journey from McCulloch-Pitts to GPT-4

Next Steps:

- Week 3: RNNs - Adding memory for sequences
- Week 4: Seq2Seq - Teaching translation
- Week 5: Transformers - The attention revolution
- Week 6: Pre-trained models - Standing on giants

"The question is not whether machines can think, but whether humans do" - B.F. Skinner