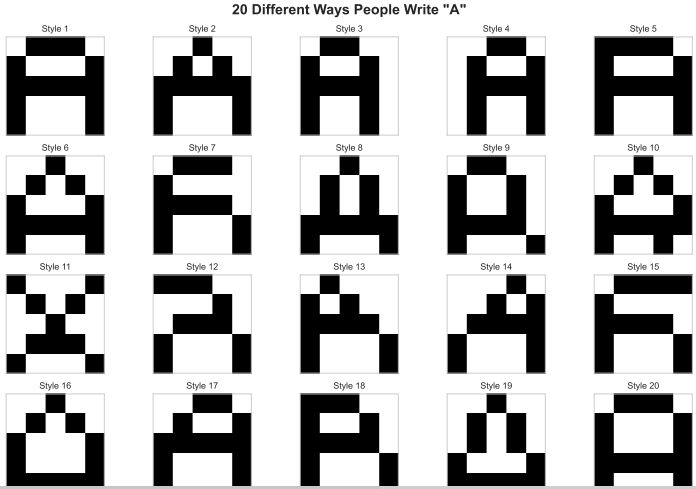
Teaching Computers to Read Your Handwriting

How Machines Learn Without Being Programmed

NLP Course 2025

From impossible rules to learning machines: A journey anyone can understand

Can you recognize all these A's?



You can see they're all "A" because:

- You learned from examples
- You recognize patterns
- You don't need exact rules

But a computer...

- Needs exact instructions
- Can't "just see" patterns
- Fails with variations

Why Can't We Just Program It?

Let's try to write rules for recognizing "A"

Attempt 1: Describe the shape

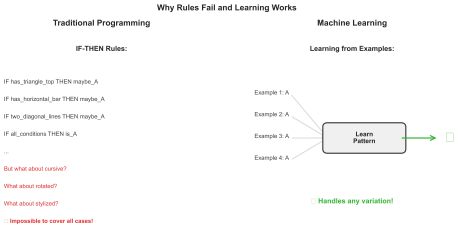
- Two diagonal lines meeting at top
- Horizontal line in middle
- Forms triangle shape

Problem: What about curved A's? Stylized A's?

Attempt 2: List all variations

- If pixels match pattern 1, or
- If pixels match pattern 2, or
- If pixels match pattern 3...

Problem: Infinite variations!



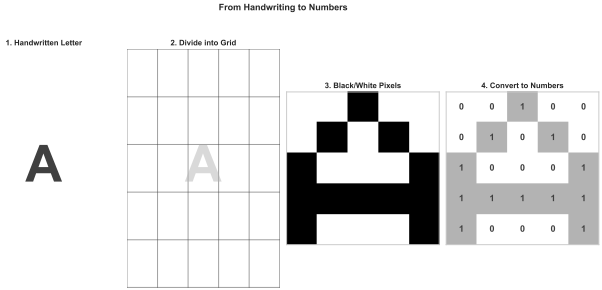
The Insight:

- Rules can't capture all variations
- Every person writes differently
- Context and style change
- We need learning, not programming

Traditional programming: Tell computer HOW. Machine learning: Show computer EXAMPLES

Everything Must Become Numbers

Computers only understand numbers, not shapes



The Transformation:

1. Start with handwritten letter
2. Divide into grid (pixels)
3. Black = 1, White = 0
4. Now it's just numbers!

Example for 5×5 grid:

- Total: 25 numbers
- Each is 0 or 1
- Patterns in the numbers = shapes
- Computer can now process it

A 28×28 image (like MNIST) becomes 784 numbers - that's our input

How did YOU learn to read?

A Child Learning:

1. Sees many examples of "A"
2. Makes mistakes
3. Gets corrected
4. Adjusts understanding
5. Tries again
6. Eventually gets it right

No Explicit Rules!

- Nobody lists all ways to write "A"
- Child discovers patterns
- Builds internal model
- Generalizes to new examples

This idea, born in 1957, would change everything

What if computers could learn this way?

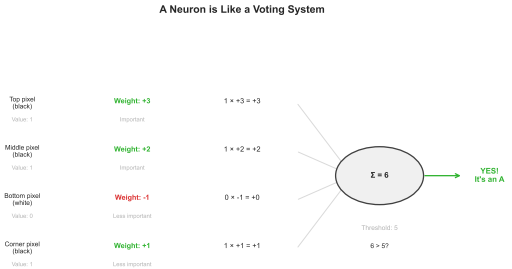
The Big Idea:

Instead of programming rules,
show many examples and let
the computer figure out patterns

Requirements:

- Way to process examples
- Method to measure mistakes
- Mechanism to improve
- Patience for many iterations

A Neuron is a Decision Maker



Think of it like voting:

- Inputs = votes from different sources
- Weights = how much each vote counts

Real Example: Is this pixel pattern an "A"?

Pixel values (inputs):

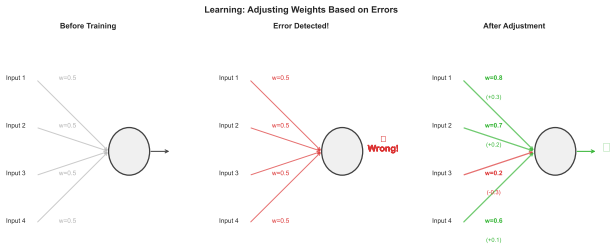
- Top center: 1 (black)
- Middle left: 1 (black)
- Middle right: 1 (black)
- Bottom: 0 (white)

Importance (weights):

- Top center: $+3$ (very important!)
- Middle left: $+2$ (important)
- Middle right: $+2$ (important)
- Bottom: -1 (should be white)

Total: $13 + 12 + 12 + 0(-1) = 7$
Decision: 7 ¿ 5? Yes! Probably an "A"

What Happens When the Neuron is Wrong?



The Learning Process:

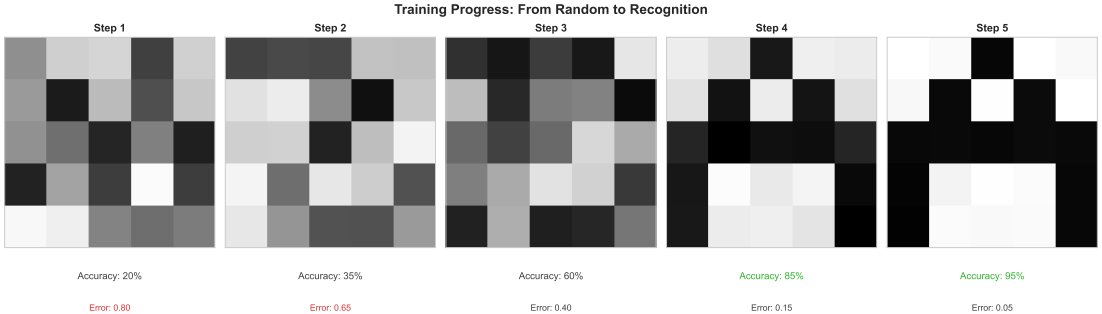
1. Neuron sees input
2. Makes prediction
3. Gets told correct answer
4. Calculates error
5. Adjusts weights

Adjustment Rules:

- Wrong + input was 1: increase weight
- Wrong + input was 0: decrease weight
- Correct: keep weight
- Bigger error = bigger change

After many examples, weights converge to useful values

Learning in Action: 5 Training Steps



Each mistake makes the neuron better at recognizing the pattern

Victory! Teaching OR Logic

Can a neuron learn: "Output 1 if ANY input is 1"?

Training Data:

Input 1	Input 2	Output
0	0	0
0	1	1
1	0	1
1	1	1

After Training:

- Weight 1: 1.0
- Weight 2: 1.0
- Threshold: 0.5

Testing Our Trained Neuron:

Example 1: (0, 1)

- Sum: $01 + 11 = 1$
- Is 1 ¿ 0.5? Yes! Output: 1

Example 2: (0, 0)

- Sum: $01 + 01 = 0$
- Is 0 ¿ 0.5? No! Output: 0

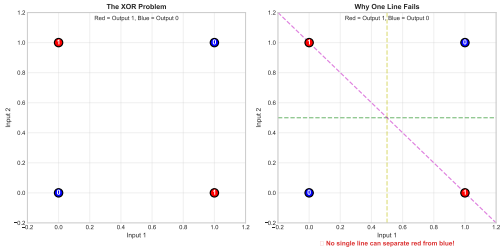
Perfect! The neuron learned OR

This same principle scales to recognizing handwritten letters

The XOR Problem: Impossible for One Neuron

XOR: Output 1 if inputs are DIFFERENT

Input 1	Input 2	Output
0	0	0
0	1	1
1	0	1
1	1	0



Why It's Impossible:

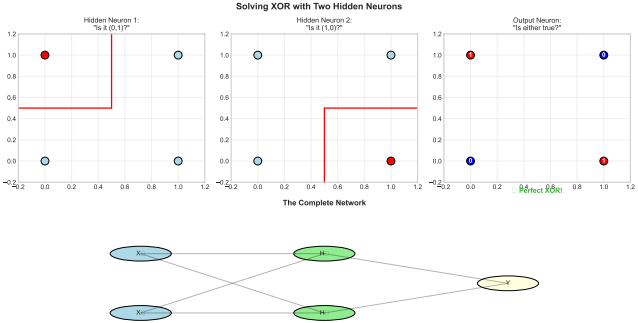
- Need to separate red from blue
- One line can't do it!
- Single neuron = single line
- Some patterns need combinations

The Revelation:

- Not all patterns are linearly separable
- Real-world problems are complex
- We need multiple neurons!

This discovery in 1969 nearly killed neural network research

Two Simple Questions Solve XOR



Break It Down:

Hidden Neuron 1: *"Is it (0,1)?"*

- Learns to detect top-left corner
- Outputs 1 only for (0,1)

Hidden Neuron 2: *"Is it (1,0)?"*

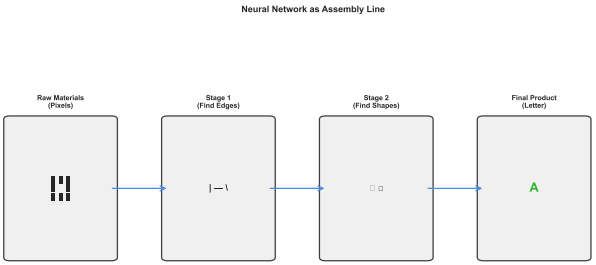
- Learns to detect bottom-right
- Outputs 1 only for (1,0)

Output Neuron: *"Is either true?"*

- Combines both answers
- Implements OR on hidden outputs
- Solves XOR perfectly!

Complex patterns = combinations of simple patterns

The Assembly Line of Pattern Recognition



Layer by Layer:

Input Layer:

- Raw pixel values
- No processing yet
- Just passes data forward

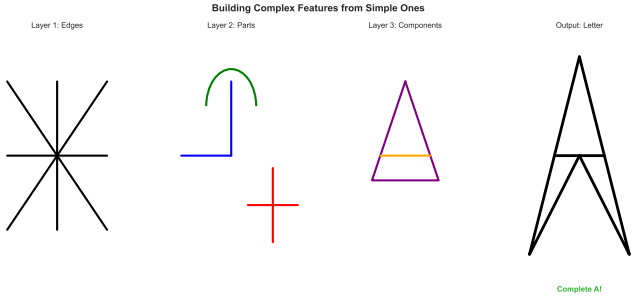
Hidden Layer:

- Detects simple features
- Each neuron finds one pattern
- Outputs are feature strengths
- We don't specify what to find!

Output Layer:

- Combines features
- Makes final decision
- "Given these features, what is it?"

How Deep Networks Recognize Letters



Layer 1: Edges

- Horizontal lines
- Vertical lines
- Diagonal lines
- Curves

Layer 2: Parts

- Corners
- Intersections
- Loops
- Line endings

Layer 3: Components

- Triangle tops
- Cross bars
- Letter segments

Output: Letters

- Complete A, B, C

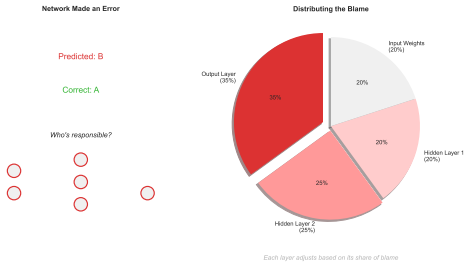
When Wrong, Which Weights Should Change?

The Challenge:

- Network predicts "B"
- Correct answer was "A"
- Many neurons involved
- Many weights contributed
- Who caused the error?

Like a Team Project:

- Project fails
- Multiple people worked on it
- Need to find what went wrong
- Everyone must improve their part

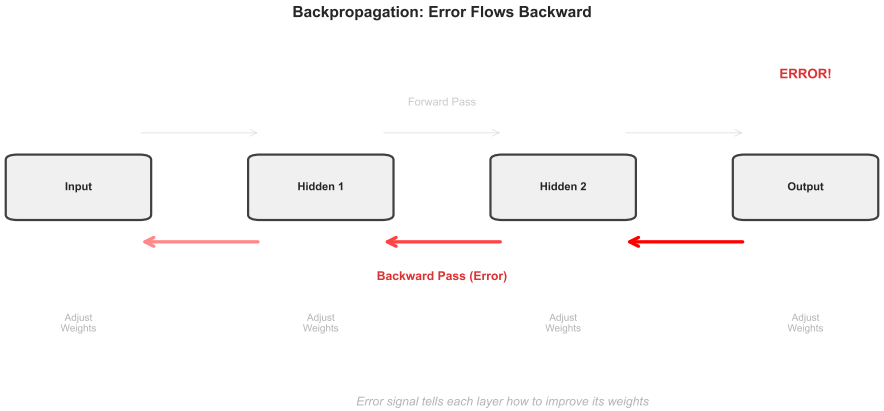


The Solution: Backpropagation

1. Calculate error at output
2. Pass blame backward
3. Each layer gets its share
4. Adjust weights accordingly

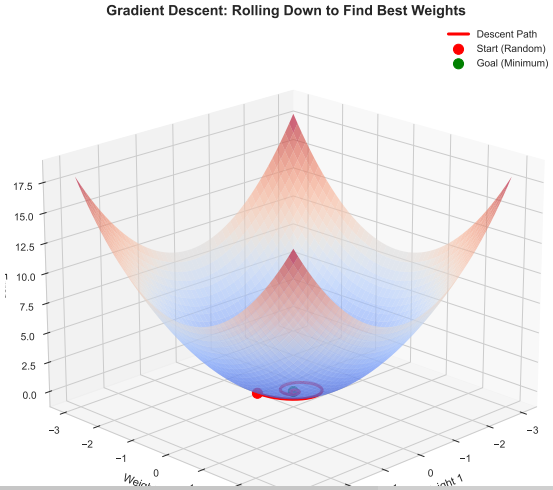
Backpropagation = systematically distributing blame for mistakes

Backpropagation: Error Flows Backward



Error signal flows backward, telling each weight how to improve

Gradient Descent: Nature's Way to Find Minimum



The Landscape:

- Height = error amount
- Position = weight values
- Goal = find lowest point
- Path = gradient descent

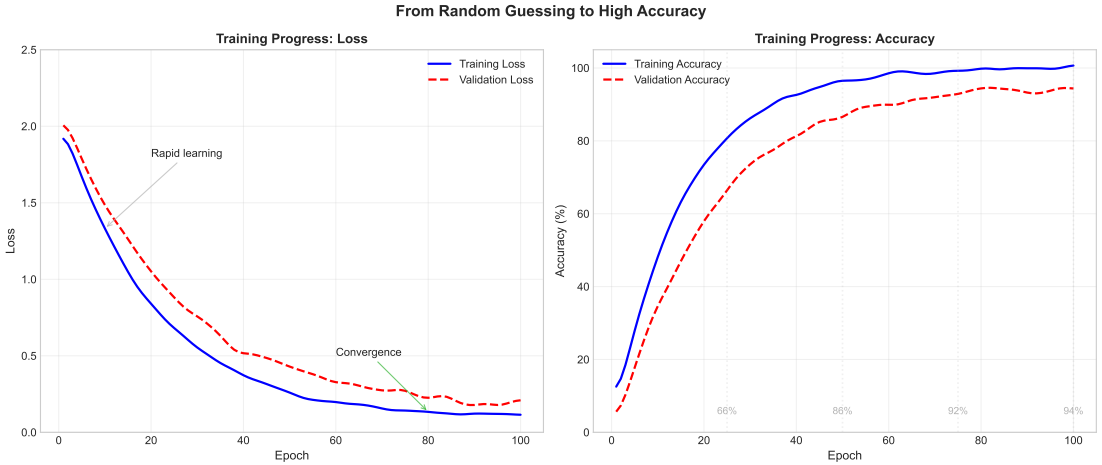
The Algorithm:

1. Start somewhere random
2. Find downhill direction
3. Take small step down
4. Repeat until flat

Learning Rate:

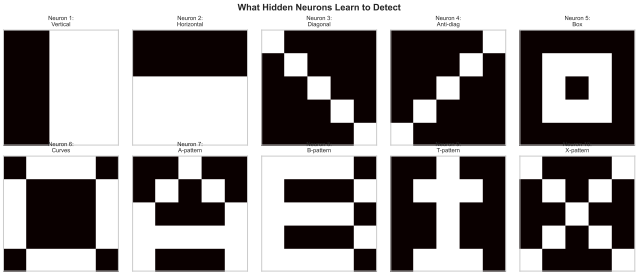
- Too big: overshoot valley

Training in Action: From Random to Recognition



Each epoch (pass through data) improves recognition accuracy

What Do Hidden Neurons Actually Learn?



Visualizing Weights:

- Each hidden neuron develops preferences
- Bright = this pixel is important
- Dark = this pixel should be off
- Patterns emerge automatically!

Discovered Features:

- Neuron 1: Vertical strokes
- Neuron 2: Diagonal lines
- Neuron 3: Curves
- Neuron 4: Intersections
- Neuron 5: Loops

Nobody programmed these!
Network discovered them from data

These feature detectors combine to recognize any handwriting style

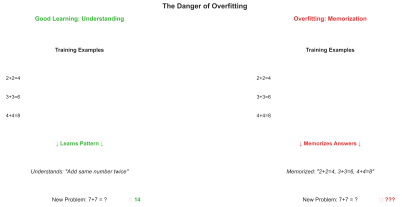
When Learning Goes Too Far

Like a Student Who:

- Memorizes practice problems
- Gets 100% on homework
- Fails the real exam
- Didn't understand concepts

Network Overfitting:

- Perfect on training data
- Terrible on new examples
- Memorized, not learned
- Too complex for the pattern



Solutions:

- More training data
- Simpler network
- Stop training earlier
- Dropout: randomly ignore neurons

The goal is generalization, not memorization

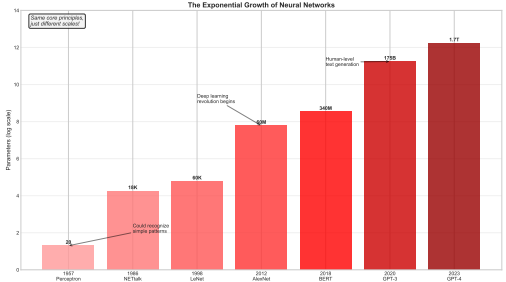
2012: The Year Everything Changed

Before 2012:

- MNIST digits: 99% accuracy
- Simple patterns only
- Shallow networks (2-3 layers)
- Limited to small images

AlexNet Changes Everything:

- 1000 object categories
- Real-world photos
- 8 layers deep
- 60 million parameters
- GPU acceleration

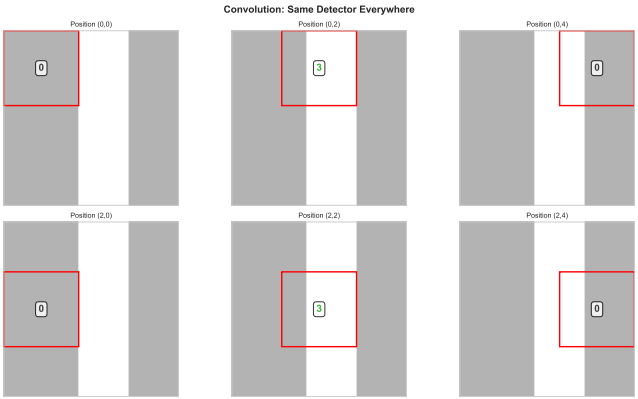


The Same Principles:

- Neurons compute weighted sums
- Layers build features
- Backprop adjusts weights
- Just MUCH bigger scale

From 784 inputs (MNIST) to 150,528 inputs (ImageNet) - same math!

How to Handle Large Images Efficiently



The Problem:

- Full connection: too many weights
- 1000×1000 image = 1M inputs!
- Would need billions of weights

The Solution: Convolution

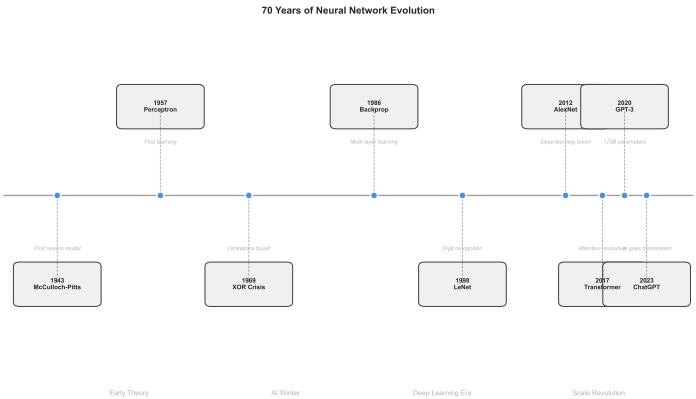
- Small detector (e.g., 3×3)
- Slide across entire image
- Same detector everywhere
- Share weights = fewer parameters

Why It Works:

- Edges are edges anywhere
- Reuse same feature detector
- Translation invariant

A 3×3 detector has only 9 weights but can find patterns anywhere in the image

From Perceptron to GPT: 70 Years of Growth



The Growth:

- 1957: 20 weights
- 1989: 10,000 weights
- 2012: 60 million
- 2020: 175 billion
- 2023: 1.7 trillion

But Still:

- Same neurons (weighted sum)
- Same learning (gradient descent)
- Same layers (feature hierarchy)
- Just massively parallel

The principles you learned today power ChatGPT - just at incredible scale

From Theory to Practice

```python
import numpy as np

# Your first neural network!
class SimpleNetwork:
    def __init__(self):
        # 784 inputs (28x28 pixels)
        # 128 hidden neurons
        # 10 outputs (digits 0-9)
        self.weights1 = np.random.randn(784, 128) * 0.01
        self.weights2 = np.random.randn(128, 10) * 0.01

    def forward(self, pixels):
        # Layer 1: pixels -> hidden
        hidden = np.maximum(0, pixels @ self.weights1)
        # Layer 2: hidden -> output
        output = hidden @ self.weights2
        return output

    def train_step(self, pixels, correct_digit):
        # Forward pass
        output = self.forward(pixels)
        # Calculate error
        error = output - correct_digit
        # Backprop (simplified)
        # ... gradient calculations ...
        # Update weights
        self.weights2 -= learning_rate * gradients2
        self.weights1 -= learning_rate * gradients1
```

This simple network can read handwriting better than many humans

What This Does:

1. Takes 784 pixel values
2. Passes through hidden layer
3. Outputs 10 scores (one per digit)
4. Highest score = prediction

Training Loop:

1. Show image
2. Get prediction
3. Calculate error
4. Update weights
5. Repeat 10,000 times
6. 97% accuracy!

You now understand every line!

When Things Go Wrong (They Will!)

"My network won't learn!"

- Learning rate too high/low
- Weights initialized poorly
- Data not normalized
- Wrong activation function

"Loss is NaN!"

- Numbers got too big
- Gradient explosion
- Fix: smaller learning rate
- Fix: gradient clipping

"Perfect training, bad testing!"

- Classic overfitting
- Network memorized data
- Fix: more data
- Fix: simpler network
- Fix: dropout

Debug Strategy:

1. Start with tiny network
2. Overfit single example first
3. Gradually add complexity
4. Monitor everything

"If it's not working, it's probably the learning rate" - Everyone who's trained networks

Your Journey Through Neural Networks

The Core Concepts:

- Rules can't capture all patterns
- Neurons: weighted sum + decision
- Learning: adjust weights from errors
- Layers: build complex from simple
- Backprop: distribute blame backward
- Training: gradient descent

The Bigger Picture:

- Same math from 1957 to GPT
- Scale changes everything
- Principles remain constant

You Can Now Understand:

- How computers read handwriting
- Why deep learning works
- What training a network means
- How ChatGPT learns patterns
- Why AI seems "intelligent"

Next Steps:

- RNNs: Networks with memory
- Attention: Focus mechanisms
- Transformers: Modern architecture
- Applications: NLP, vision, more

You've mastered the foundation that powers all modern AI

You Now Understand Neural Networks!

From recognizing handwritten letters
to understanding the principles behind ChatGPT

The same neurons that learned to read your handwriting
can learn to write poetry, translate languages,
diagnose diseases, and drive cars

It's all weighted sums and gradient descent

Next Week: Teaching Networks to Remember
Recurrent Neural Networks and Sequential Processing

"The question is not whether machines can learn, but what they cannot learn"