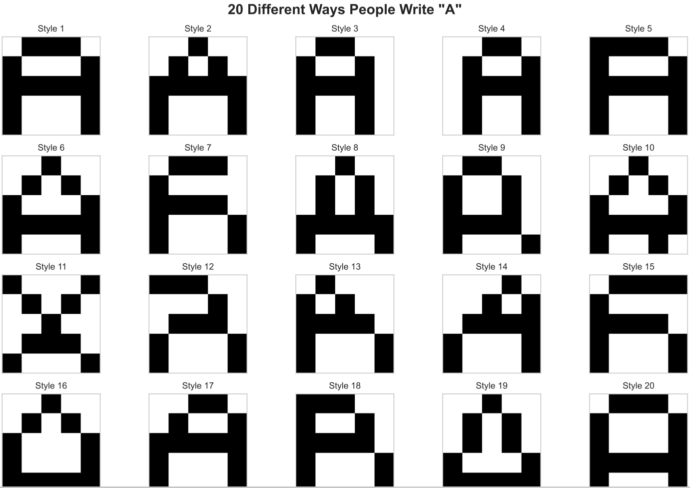
Teaching Computers to Recognize Your Handwriting

A Complete Journey from Simple Rules to Deep Learning

NLP Course 2025

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

Everyone writes the letter "A" differently



You recognize them all because:

- You learned from examples
- You see the pattern, not exact shape
- Your brain generalizes

But a computer sees:

- Just pixels (dots)
- No inherent meaning
- Needs exact instructions

Computers Need Rules, But Writing is Infinite

The Variety Problem:

- Print vs cursive
- Size variations
- Rotation angles
- Thickness differences
- Personal style
- Speed of writing
- Writing instrument

The Numbers:

- 7 billion people
- Each writes uniquely
- Changes with mood/age
- = Infinite variations!

Central Question:
How can we possibly program rules for infinite variations?

Spoiler: We can't. We need a different approach entirely.

Why IF-THEN Rules Don't Work

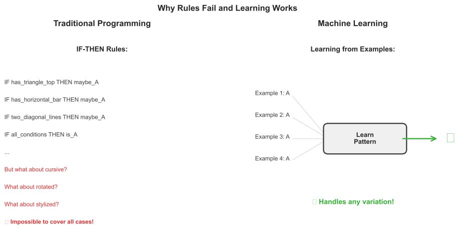
Attempt at Programming "A" Recognition:

```
def recognize_A(pixels):
    if has_triangle_top(pixels):
        if has_horizontal_bar(pixels):
            if two_diagonal_lines(pixels):
                return "It's an A!"

    # But wait...
    if cursive_style(pixels):
        # Different rules!

    if child_handwriting(pixels):
        # More different rules!

    # This goes on forever...
```



Why This Fails:

- Can't anticipate all styles
- Rules conflict with each other
- Exceptions have exceptions
- Impossible to maintain!

After decades of trying, programmers gave up on rule-based recognition

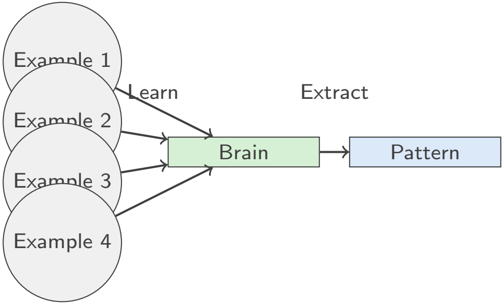
Children Don't Use Rules

A Child Learning to Read:

1. Sees letter "A" in a book
2. Parent says "That's an A"
3. Sees different "A" styles
4. Makes mistakes
5. Gets corrected
6. Brain adjusts understanding
7. Eventually recognizes any "A"

No rules memorized!
Pattern discovered naturally!

The Learning Process:



Key Insight:
Learning from examples works better than following rules!

This biological inspiration would revolutionize computing

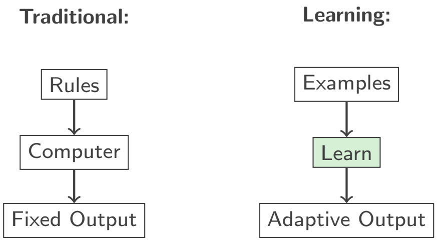
What if Computers Could Learn Like Children?

Traditional Programming:

- Human writes rules
- Computer follows rules
- Fails on new situations
- Needs constant updates

Machine Learning:

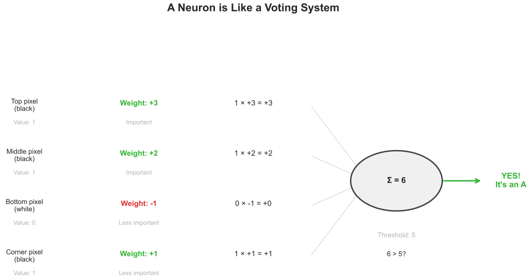
- Show many examples
- Computer finds patterns
- Generalizes to new cases
- Improves with more data



The Revolution:
Instead of programming HOW,
we show examples of WHAT

This idea, proposed in 1957, would take 60 years to fully realize

Think of a Neuron Like a Traffic Light



Traffic Light Decision:

- Input 1: Cars waiting?
- Input 2: Pedestrian button?
- Input 3: Time since last change?
- Decision: Change light or not

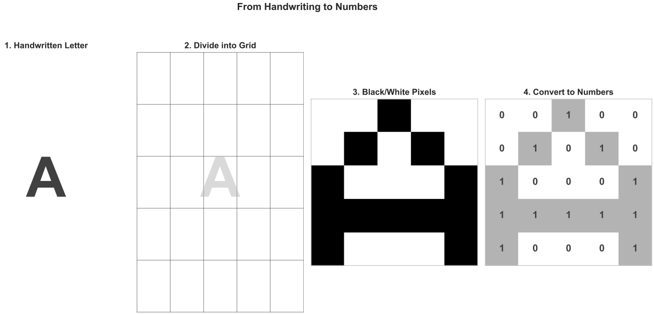
Neuron Decision:

- Input 1: Pixel 1 value
- Input 2: Pixel 2 value
- Input 3: Pixel 3 value
- Decision: Is it an "A" or not

The Process:

1. Receive inputs
2. Weight their importance

From Handwriting to Numbers



The Transformation:

1. Scan the letter
2. Divide into grid (pixels)
3. Black pixel = 1
4. White pixel = 0
5. Now we have numbers!

Example (5×5 grid):

- Center top: 1 (black)
- Sides: mostly 0 (white)
- Total: 25 numbers
- Computer can process!

Everything in AI starts with converting the real world into numbers

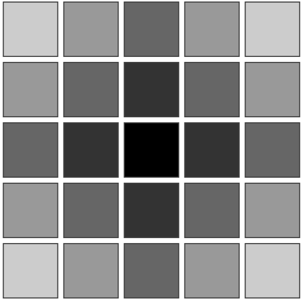
Not All Pixels Are Equally Important

Voting Analogy:

- Expert vote: Weight = 10
- Regular vote: Weight = 1
- Novice vote: Weight = 0.1

For Letter "A":

- Top center pixel: High weight
- Middle sides: High weight
- Bottom corners: Low weight
- Very bottom: Negative weight



Weight Importance Map

Learning = Finding Right Weights:

- Start with random weights
- See examples
- Adjust weights

Simple Math: Multiply and Add

Concrete Example:

Pixel	**Value**	**Weight**	**Result**
Top center	1	$\times 3$	$= 3$
Left middle	1	$\times 2$	$= 2$
Right middle	1	$\times 2$	$= 2$
Bottom center	0	$\times$(-1)	$= 0$
		Total:	**7**

It's just like calculating a bill:

- 1 pizza $\times$ \$10 $=$ \$10
- 2 sodas $\times$ \$3 $=$ \$6
- Total $=$ \$16

This simple operation, repeated millions of times, powers all of AI

What the Neuron Does:

1. Takes each input (0 or 1)
2. Multiplies by its weight
3. Adds all results
4. Gets a total score

The Formula:
Total $=$ (Input1 $\times$ Weight1) $+$
(Input2 $\times$ Weight2) $+$ ...

That's it! No complex math!

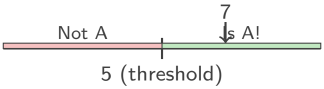
Slide 10: Making the Decision

Is the Total High Enough?

Setting a Threshold:

- Total score: 7
- Threshold: 5
- Is 7 ¿ 5? Yes!
- Decision: "It's an A!"

Like a Scale:



The threshold determines how confident the neuron needs to be

Different Thresholds:

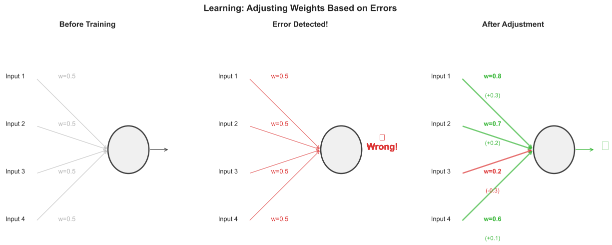
- Low threshold: More "yes"
- High threshold: More "no"
- Just right: Good accuracy

Real Examples:

- Email spam filter: Threshold = 0.8
- Medical diagnosis: Threshold = 0.95
- Face unlock: Threshold = 0.7

Threshold is also learned!

Wrong Answer? Adjust the Weights!



The Learning Rule:

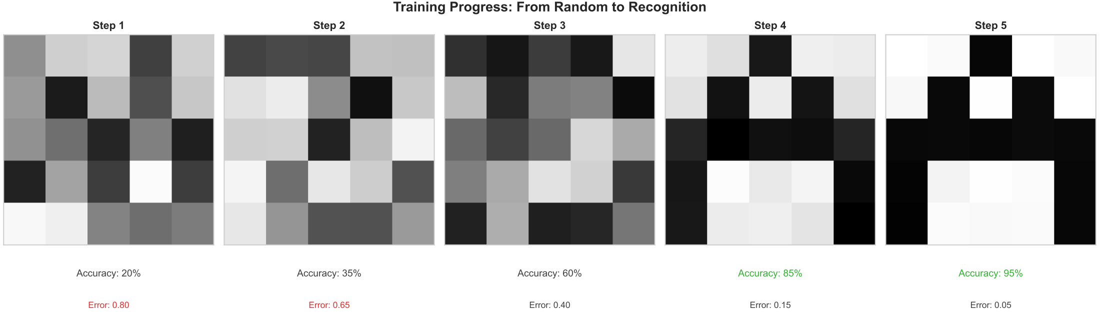
1. Show an example
2. Neuron makes prediction
3. Check if correct
4. If wrong: adjust weights
5. Repeat many times

How to Adjust:

- Pixel was 1, answer wrong?
  → Increase its weight
- Pixel was 0, answer wrong?
  → Decrease its weight
- Answer correct?
  → Keep weights same

This is exactly how children learn - through trial and error

5 Training Steps with Real Numbers



Step by Step:

- Step 1: Random weights, 20% correct
- Step 2: Small adjustments, 35% correct
- Step 3: Pattern emerging, 60% correct
- Step 4: Almost there, 85% correct
- Step 5: Trained! 95% correct

What's Happening:

- Weights organize themselves
- Important pixels get high weights
- Unimportant get low weights
- Pattern detector emerges
- No programming required!

Slide 13: Victory! The OR Problem

Proof That Learning Works

The OR Problem:

"Output 1 if ANY input is 1"

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 Neuron:

- Test (0,0): $0 \times 1 + 0 \times 1 = 0$
  0 ¡ 0.5 $\rightarrow$ Output 0
- Test (0,1): $0 \times 1 + 1 \times 1 = 1$
  1 ¿ 0.5 $\rightarrow$ Output 1
- Test (1,0): $1 \times 1 + 0 \times 1 = 1$
  1 ¿ 0.5 $\rightarrow$ Output 1
- Test (1,1): $1 \times 1 + 1 \times 1 = 2$
  2 ¿ 0.5 $\rightarrow$ Output 1

100% Accuracy! Learning Works!

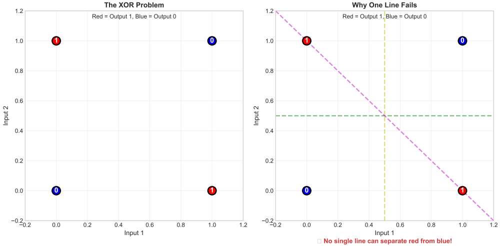
This simple success gave researchers hope that complex problems could be solved

The Impossible Problem

XOR (Exclusive OR):

"Output 1 if inputs are DIFFERENT"

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



The Problem:

- Can't separate with one line
- Single neuron = single line
- Mathematically impossible!

This Discovery (1969):

- Killed neural network research
- "AI Winter" began
- Funding disappeared
- 15 years of abandonment

This limitation seemed to prove that neural networks were a dead end

Slide 15: Why One Neuron Fails

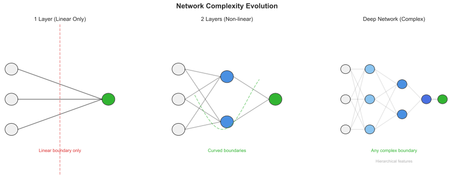
Some Patterns Need Combinations

What One Neuron Can Do:

- Draw one straight line
- Separate into two groups
- Simple decisions only
- Linear patterns

Real World Needs:

- Complex boundaries
- Multiple criteria
- Hierarchical features
- Non-linear patterns



The Solution:

- Use multiple neurons
- Arrange in layers
- Combine simple decisions
- Create complex boundaries

Nature uses billions of neurons - why did we think one would suffice?

Two Simple Questions Solve XOR

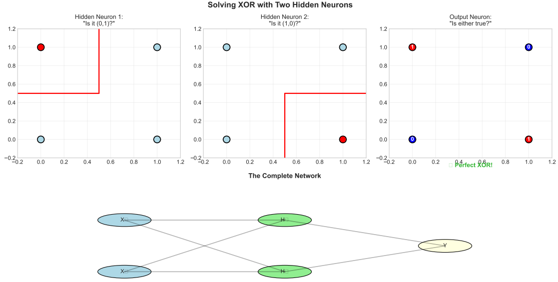
Original Problem:
"Is input pattern (0,1) OR (1,0)?"

Break It Down:

1. Question 1: "Is it (0,1)?"
2. Question 2: "Is it (1,0)?"
3. Combine: "Is either true?"

Each Question is Simple:

- Can be solved by one neuron
- Just needs right weights
- Linear separation works

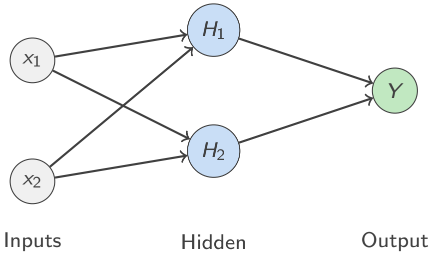


The Magic:
Combining two simple linear decisions creates one complex non-linear decision!

This breakthrough saved neural networks from obscurity

Teamwork Makes Complex Patterns Possible

The Network Structure:



How It Solves XOR:

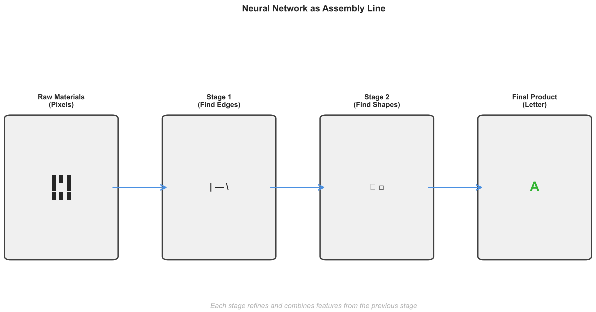
1. H_1 detects pattern (0,1)
2. H_2 detects pattern (1,0)
3. Output combines: H_1 OR H_2

Example: Input (0,1)

- H_1: "Yes, it's (0,1)!" $\rightarrow$ 1
- H_2: "No, not (1,0)" $\rightarrow$ 0
- Output: "1 OR 0 = 1"

Hidden neurons create internal representations that make problems solvable

Hidden Layers Discover Their Own Features



Why "Hidden"?

- We don't tell them what to find
- They discover useful features
- Different every time you train
- Internal representation emerges

Like a Factory:

1. Raw materials (pixels) enter
2. Workers (hidden neurons) process
3. Each finds something useful
4. Assembly (output) combines results
5. Final product emerges

The network decides what features are useful - we just provide examples

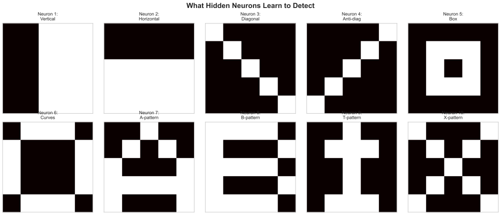
Each Hidden Neuron Finds One Pattern

In Handwriting Recognition:

- Neuron 1: Vertical lines
- Neuron 2: Horizontal lines
- Neuron 3: Diagonal lines
- Neuron 4: Curves
- Neuron 5: Intersections
- Neuron 6: Loops

Combining Features:

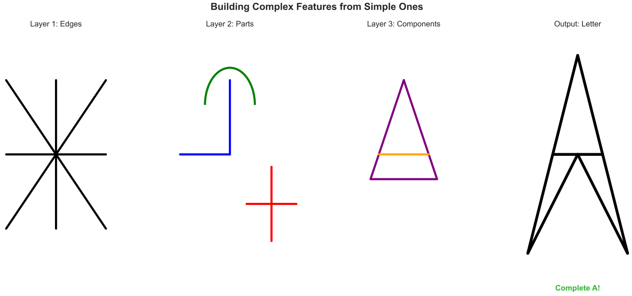
- "A" = diagonals + horizontal
- "B" = vertical + curves
- "C" = curve only
- "D" = vertical + curve



Important:
Nobody programmed these features!
Network discovered them from data.

Feature detection is automatic - the network finds what's useful

Each Layer Refines the Previous



Layer by Layer:

1. **Input:** Raw pixels
2. **Layer 1:** Find edges
3. **Layer 2:** Combine edges
4. **Layer 3:** Find shapes
5. **Output:** Identify letter

Progressive Refinement:

- Simple $\rightarrow$ Complex
- Local $\rightarrow$ Global
- Concrete $\rightarrow$ Abstract
- Parts $\rightarrow$ Whole

Just like a factory assembly line, each stage adds value

We Don't Control What They Learn

What We Control:

- Number of hidden neurons
- Number of layers
- Learning rate
- Training examples

What We Don't Control:

- What each neuron detects
- How features combine
- Internal representations
- Discovery process

It's Like Teaching:

- Show a child many dogs
- They learn "dogness"
- Can't explain exactly how
- But recognition works!

Hidden layers are where
the "intelligence" emerges
without explicit programming

This emergent behavior is what makes neural networks powerful and mysterious

The Network Discovers What Matters

Training for "A" Recognition:

Week 1: Random features

- Neuron 1: Random noise
- Neuron 2: Random pixels
- Accuracy: 10%

Week 2: Basic patterns

- Neuron 1: Some edges
- Neuron 2: Dark regions
- Accuracy: 40%

Week 3: Useful features

- Neuron 1: Diagonal lines!
- Neuron 2: Intersections!
- Accuracy: 90%

Different Training Runs:

- Same data
- Same architecture
- Different features emerge!
- But same accuracy

Why This Matters:

- No feature engineering
- Adapts to any problem
- Finds optimal representation
- Works for any pattern type

The network finds its own way to solve the problem!

This self-organization is the key to deep learning's success

How Layers Build Understanding

1-Layer Network:

- Can learn: OR, AND
- Can't learn: XOR
- Linear boundaries only
- Simple patterns

2-Layer Network:

- Can learn: XOR
- Curved boundaries
- Combined features
- Most practical problems

Deep Networks (many layers):

- Complex hierarchies
- Abstract concepts
- Subtle patterns

1 Layer:

Linear only

2 Layers:

Curved boundaries

Deep:

Any boundary

The Universal Approximation Theorem (1989)

The Theorem (simplified):

"A network with one hidden layer and enough neurons can approximate ANY continuous function to any desired accuracy"

What This Means:

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

The Catch:

- "Enough" might be millions
- Training might take forever
- Shallow networks need width
- Deep networks more efficient

Practical Impact:

- Ended theoretical doubts
- Justified continued research
- Led to deep learning
- Changed everything

This theorem proved neural networks weren't limited - we just needed to scale them

How Do We Train Multiple Layers?

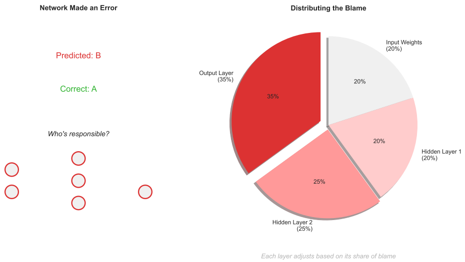
The Problem:

- Output layer: We know the error
- Hidden layers: What's their error?
- Can't measure directly
- Need to distribute blame

Credit Assignment:

- Network predicts "B"
- Correct answer: "A"
- Many neurons involved
- Who's responsible?

This problem stumped researchers for years until backpropagation



The Solution: Backpropagation
Coming next: How to teach entire networks!

When Wrong, Who's Responsible?

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"