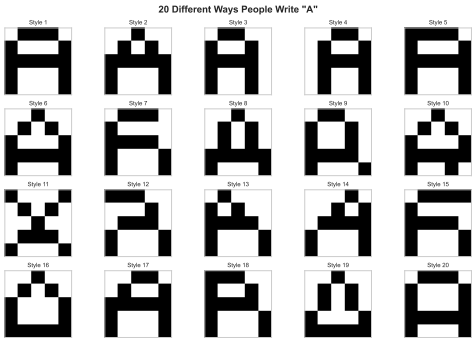
Neural Networks: Complete Guide

From Handwriting Recognition to Modern AI in 10 Slides

NLP Course 2025

1. The Handwriting Challenge

Everyone Writes Differently



You recognize them all because:

- You learned from examples
- You see patterns, not exact shapes
- Your brain generalizes

But computers see:

- Just pixels (numbers)

Why Traditional Programming Fails

Can't write rules for infinite variations:

- 7 billion people write uniquely
- Print vs cursive
- Size, rotation, thickness
- Personal style changes
- = Impossible to program!

Historical Timeline

- 1943: McCulloch-Pitts neuron
- 1958: Perceptron learns
- 1969: XOR problem (AI Winter)
- 1986: Backpropagation
- 1998: LeNet, 2012: AlexNet
- 2022: ChatGPT

Solution: Let computers learn patterns from examples!

2. The Neuron: A Weighted Voter

Think of a Traffic Light

- Input 1: Cars waiting? (w=0.5)
- Input 2: Pedestrian? (w=0.8)
- Input 3: Time? (w=0.3)
- Add weighted inputs
- Decision: Change or not

Neuron is the Same:

- Input 1: Pixel 1 value
- Input 2: Pixel 2 value
- Input 3: Pixel 3 value
- Each has importance (weight)
- Add them up
- Decision: Is it an "A"?

A neuron is just a weighted voter - nothing magical!

Mathematical Definition

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

Components:

- x_i = inputs (pixel values)
- w_i = weights (importance)
- b = bias (baseline threshold)
- z = output score

Example: Party Decision

$$\text{Score} = -2 \cdot \text{Distance} + 3 \cdot \text{Friends} - 5$$

If Score $> 0 \rightarrow$ GO, else STAY

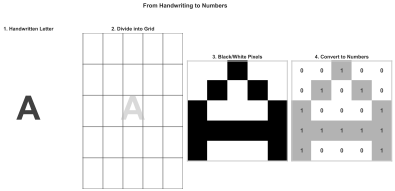
Geometric Interpretation

Decision boundary: line where Score $= 0$

$$-2d + 3f - 5 = 0 \quad \Rightarrow \quad f = \frac{2d + 5}{3}$$

3. From Images to Numbers

The Transformation Process



Steps:

1. Scan letter
2. Grid: 28x28 pixels
3. Black=1, White=0
4. Total: 784 numbers!

Why This Works:

- Math only works on numbers
- 784 inputs → neuron processes
- Same representation for all letters
- Computer can now "see"

Real Example: Letter "A"

Grid (5x5 simplified):

$$\begin{bmatrix} 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 & 1 \end{bmatrix}$$

Flattened: [0,1,1,1,0,0,1,0,1,0,1,1,1,1,1,...]

This is the Input:

$$x = [x_1, x_2, ..., x_{784}]$$

Each pixel becomes one input to the neuron!

The Calculation:

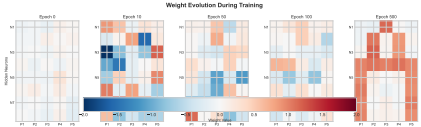
$$z = w_1 x_1 + w_2 x_2 + ... + w_{784} x_{784} + b$$

784 multiplications, 784 additions, 1 bias

Everything must become numbers for computers!

4. How Networks Learn: Adjusting Weights

Learning = Adjusting Weights



The Learning Process:

- Start: Random weights
- See example: “This is A”
- Neuron: 0.2 (wrong!)
- Adjust important weights
- Repeat 1000s of times
- Weights organize!

The Rule:

- Pixel=1, wrong? → Increase
- Pixel=0, wrong? → Decrease
- Correct? → Keep

Training Progress: 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 Happens:

- Weights organize
- Important pixels → high weights
- Unimportant → low weights
- Pattern emerges

5. Activation Functions: The Secret to Power

The Linearity Problem

Without activation, multiple neurons = one neuron!

$$z_2 = w_1(w_2x + b_2) + b_1 = (w_1w_2)x + (w_1b_2 + b_1)$$

This is just another linear function!

Solution: Add Non-linearity

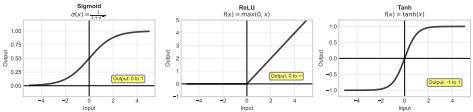
$$a = f(z) = f\left(\sum_i w_i x_i + b\right)$$

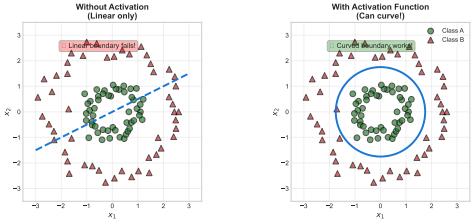
Common Activations

- **Sigmoid:** $\sigma(z) = \frac{1}{1+e^{-z}}$ (0 to 1)
- **ReLU:** $\max(0, z)$ (modern standard)
- **Tanh:** $\frac{e^z - e^{-z}}{e^z + e^{-z}}$ (-1 to 1)
- **Leaky ReLU:** $\max(0.01z, z)$ (prevents dying)

Critical: Without activation, 100 layers = 1 neuron!

Visual Comparison





Why ReLU Dominates:

- Simple: max(0, z)
- Fast to compute
- No vanishing gradient

6. The XOR Crisis and Hidden Layers

XOR Problem
Output 1 if inputs different, 0 if same

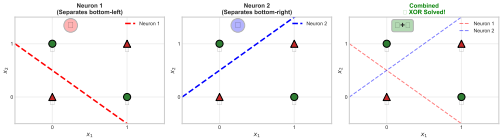
x_1	x_2	Output
0	0	0
0	1	1
1	0	1
1	1	0

Challenge: Draw ONE line separating 1's from 0's
Result: Impossible!

Historical Impact (1969)
Minsky & Papert proved single-layer networks can't solve XOR
⇒ **First AI Winter** - funding dried up for decades

Single neurons only solve linearly separable problems

The Solution: Hidden Layers



How It Works:

- Hidden neuron 1: Separates (0,0) from others
- Hidden neuron 2: Separates (1,1) from others
- Output neuron: Finds intersection
- Only (0,1) and (1,0) satisfy both!

Architecture:

- Input layer: 2 neurons (x_1, x_2)
- Hidden layer: 2 neurons (two boundaries)
- Output layer: 1 neuron (combines)

Hidden layers enable networks to solve any problem!

7. Measuring Error: Loss Functions

How Do We Know If Network Is Wrong?
Concrete Example:
Predicting house price:

- Network predicts: \$300,000
- Actual price: \$400,000
- Error = \$400k - \$300k = \$100k

The Problem:

- Positive (+\$100k) and negative (-\$100k) cancel out
- We care about magnitude, not direction
- Solution: Square the error!

Why Square It?

- Always positive: $(\$100k)^2 = 10B\$$
- Big mistakes hurt more: $(\$200k)^2 = 40B\$$
- Math works nicely for optimization

The Loss Function (MSE)
For one example:

$$\text{Error} = (\text{pred} - \text{actual})^2$$

For all examples:

$$L = \frac{1}{n}\sum_{i=1}^{n}(y_{\text{pred}}^{(i)} - y_{\text{true}}^{(i)})^2$$

Real Numbers:

Predicted	Actual	Error^2
0.3	1.0	0.49
0.8	1.0	0.04
0.1	0.0	0.01
	Average:	0.18

Goal of training: Make this loss as small as possible!

8. Backpropagation: How Networks Learn

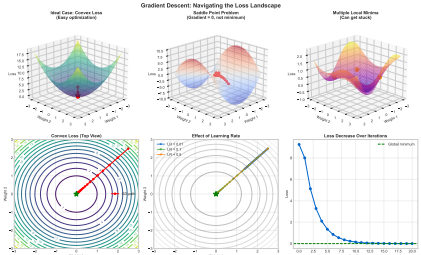
The Credit Assignment Problem
Given output error, which weights to adjust by how much?

Algorithm (4 Steps)
1. **Forward:** $z = Wa + b$, $a = f(z)$
2. **Loss:** $L = \frac{1}{2}(y_{pred} - y_{true})^2$
3. **Backward (Chain):**

$$\frac{\partial L}{\partial w} = \frac{\partial L}{\partial a} \cdot \frac{\partial a}{\partial z} \cdot \frac{\partial z}{\partial w}$$

4. **Update:** $w \leftarrow w - \eta \partial L / \partial w$ (η=0.001)

Gradient Descent Intuition



Hiking in fog:

1. Feel slope (gradient)
2. Step downhill (update)
3. Repeat to valley (converge)

History

- 1970s: Werbos invented
- 1986: Rumelhart, Hinton, Williams
- Foundation of modern DL

9. Emergent Features and Theoretical Power

Features Emerge
Training for "A":

Week 1: Random (10% acc)
Week 2: Edges, regions (40%)
Week 3: Diagonals, intersections! (90%)

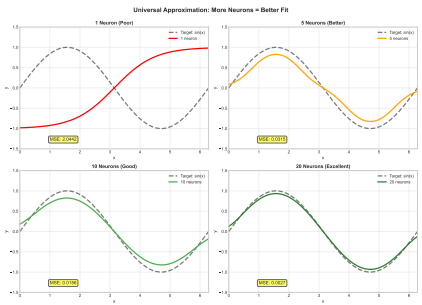
Why Matters:

- No feature engineering
- Adapts to any problem
- Finds optimal representation
- Different runs, same accuracy!

Universal Approximation Theorem
Cybenko (1989): Network with one hidden layer can approximate *any* continuous function to *any* accuracy!

How: Position steps at different locations:

$$f(x) \approx \sum_{i=1}^{n} a_i \sigma(w_i x + b_i)$$



10. From Theory to Modern AI: Complete Summary

Key Breakthroughs

- 1998 LeNet, 2012 AlexNet
- 2015 ResNet (152 layers)
- 2017 Transformers
- 2020 GPT-3 (175B params)
- 2022 ChatGPT

Applications

- Medical diagnosis
- Self-driving
- Drug discovery
- Translation
- Code/Art gen
- Speech
- Protein folding
- Climate

Formulas

Neuron	$\sum w_i x_i + b$
Sigmoid	$1/(1+e^{-z})$
ReLU	$\max(0, z)$
Loss	$\frac{1}{n}\sum(y_p - y_t)^2$
Update	$w - \eta \partial L/\partial w$

Path

Problem → Numbers → Neuron → Activation → XOR → Hidden → Loss → Backprop → Theory → Practice

Key Insights

- Neurons = voters
- Learning = weights (20%→95%)
- Activation = non-linearity
- Hidden = any problem
- Backprop = assign credit
- Features emerge

Next

- Code (NumPy/PyTorch)
- Build classifier/generator
- Learn: Fast.ai, CS231n
- Viz: playground.tensorflow

You now understand the fundamentals powering modern AI!