Neural Networks: Complete Summary

From Zero to Understanding in 10 Slides

NLP Course 2025

1. The Problem & Motivation

Why Traditional Programming Fails
Traditional code: IF-THEN-ELSE rules
But how to code:

- Handwritten digit recognition?
- Spam detection?
- Chess at grandmaster level?

1959 Mail Sorting Crisis

- U.S. Postal Service: millions of ZIP codes
- Every person writes differently
- Traditional OCR failed

Paradigm Shift
Instead of programming rules, let computers *learn patterns from examples*!

Historical Timeline

- 1943: McCulloch-Pitts neuron
- 1958: Perceptron (first learning)
- 1969: Limitations proved (AI Winter)
- 1986: Backpropagation rediscovered
- 1998: LeNet reads bank checks
- 2012: AlexNet (deep learning revolution)

Key: Neural networks excel at pattern recognition where rules are unclear

2. The Neuron: Building Block

Mathematical Definition

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

Components:

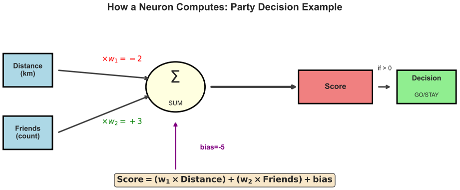
- x_i = inputs (data)
- w_i = weights (importance)
- b = bias (baseline)
- z = output (score)

Party Decision Example

Alex's formula:

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

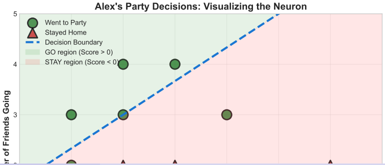
If Score $> 0 \rightarrow$ GO, else STAY



Geometric Interpretation

Decision boundary: line where Score $= 0$

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



3. Activation Functions: The Secret to Power

The Linearity Problem
Without activation: multiple neurons = another linear function!

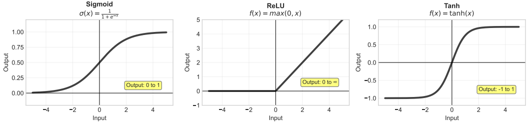
$$z_2 = w_1(w_2x + b_2) + b_1 = (w_1w_2)x + (w_1b_2 + b_1)$$

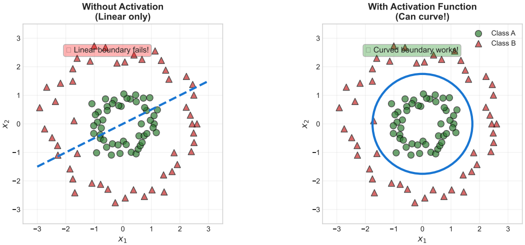
Solution: Add non-linear activation

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

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!

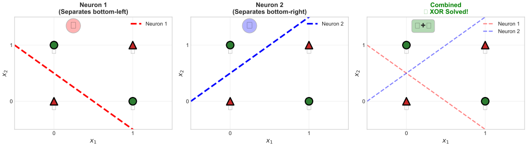
XOR Problem
Output 1 if inputs different, 0 if same

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!*

Geometric Proof

- (0,1) and (1,0) must be on one side
- (0,0) and (1,1) on the other
- No straight line separates opposite corners!



Historical Impact (1969)
Minsky & Papert proved single-layer networks cannot solve XOR
$\Rightarrow$ **First AI Winter**
Funding dried up for decades

Limitation: Single neurons only solve *linearly separable* problems

5. Hidden Layers: The Breakthrough

The Solution

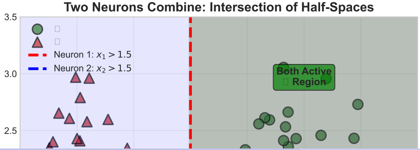
Use TWO neurons in hidden layer, combine outputs!

Architecture

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

Geometric Intuition

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



Forward Pass Example

Given weights:

- Hidden 1: $w = [1, 1], b = -0.5$
- Hidden 2: $w = [1, 1], b = -1.5$
- Output: $w = [1, -1], b = 0$

For input $(1, 0)$:

$$\begin{aligned} h_1 &= \sigma(1 \cdot 1 + 1 \cdot 0 - 0.5) \\ &= \sigma(0.5) \approx 0.62 \\ h_2 &= \sigma(1 \cdot 1 + 1 \cdot 0 - 1.5) \\ &= \sigma(-0.5) \approx 0.38 \\ y &= \sigma(1 \cdot 0.62 - 1 \cdot 0.38) \\ &= \sigma(0.24) \approx 0.56 \text{ (close to 1!)} \end{aligned}$$

Why it works: Each neuron learns different feature. Enough neurons $\rightarrow$ any boundary!

6. Backpropagation: How Networks Learn

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

The Algorithm (4 steps)

1. Forward Pass

$$z^{[l]} = W^{[l]}a^{[l-1]} + b^{[l]}, \quad a^{[l]} = f(z^{[l]})$$

2. Compute Error

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

3. Backward Pass (chain rule)

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

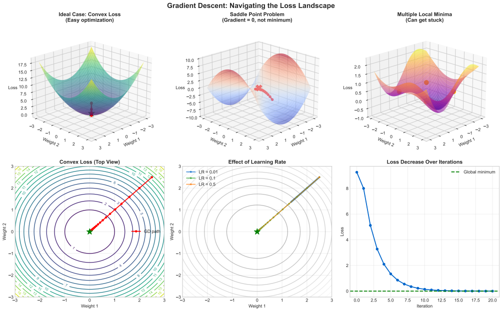
4. Update Weights

$$w \leftarrow w - \eta \frac{\partial L}{\partial w}$$

where η = learning rate (step size)

Gradient Descent Intuition
Hiking in fog to reach valley:

1. Feel slope under feet (gradient)
2. Take step downhill (update)
3. Repeat until can't go lower (converge)



Historical Note
Invented 1970s, famous 1986
(Rumelhart/Hinton/Williams)

7. Universal Approximation Theorem

Cybenko's Theorem (1989)
Network with:

- One hidden layer
- Finite neurons
- Sigmoid activation

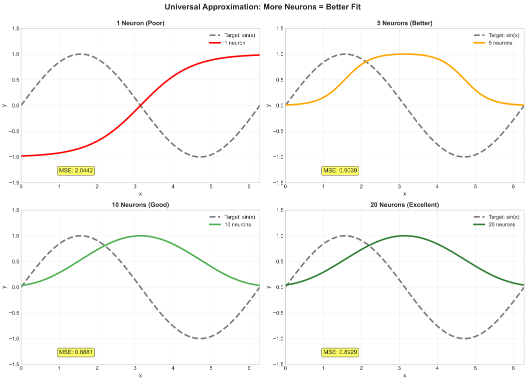
can approximate *any* continuous function to *any* accuracy!

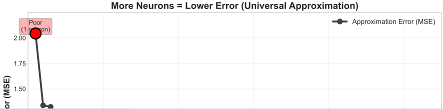
How It Works
Each sigmoid = smooth step function
Position steps at different locations/heights → build any curve:

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

Caveats

- Guarantees existence, not efficient learning
- May need exponential neurons
- Deep networks often better than wide





8. From Theory to Modern Practice

Key Breakthroughs Timeline

- 1998 LeNet-5: First CNN, read checks
- 2012 AlexNet: ImageNet 26% → 16%
- 2015 ResNet: Skip connections, 152 layers
- 2017 Transformers: Attention revolutionized NLP
- 2022 ChatGPT: LLMs mainstream

Why 2012 Was Different

1. **ReLU**: Solved vanishing gradients
2. **Dropout**: Prevents overfitting
3. **GPUs**: 50x faster training
4. **Big Data**: ImageNet 14M images
5. **Batch Norm**: Stabilizes layers

Modern Architectures

- **CNNs:** Images (ResNet, EfficientNet)
- **RNNs:** Sequences (LSTM, GRU)
- **Transformers:** Everything (BERT, GPT, ViT)

Real-World Applications Today

- Medical diagnosis
- Autonomous vehicles
- Drug discovery
- Language translation
- Code generation
- Art generation
- Speech recognition
- Recommendation systems
- Protein folding
- Climate modeling
- Fraud detection
- Robotics

Key Innovation: Transfer Learning

Pre-train large model → Fine-tune for specific task
Examples: BERT, GPT-3, DALL-E, AlphaFold

Today: Neural networks power most modern AI systems

9. Building Your First Network

7-Step Process

1. Define Problem

- Classification vs Regression?
- Input/output sizes?
- Target accuracy?

2. Prepare Data

- Split: 70% train, 15% val, 15% test
- Normalize: [0,1] or mean=0, std=1
- Augment: flip, rotate, paraphrase

3. Design Architecture

- Start simple: 1-2 hidden, 32-128 neurons
- ReLU hidden, sigmoid/softmax output
- Add dropout (0.2-0.5)

4. Hyperparameters

- Learning rate: 0.001 (most critical!)
- Batch size: 32-256
- Optimizer: Adam
- Loss: CrossEntropy/MSE

5. Train

Forward → Loss → Backward → Update

6. Debug Common Issues

Symptom	Solution
Loss not decreasing	Try 10x higher/lower LR
Train good, val bad	Dropout, more data
Loss = NaN	Lower LR, clip gradients

7. Evaluate

- Never touch test until final!
- Multiple metrics (Accuracy, F1, Precision)
- Visualize: confusion matrix, learning curves

Best Practices

- Start simple, add complexity
- Log everything
- Save checkpoints
- Monitor training (TensorBoard)

10. Summary: The Complete Picture

Essential Formulas

Neuron	$z = \sum_i w_i x_i + b$
Sigmoid	$\sigma(z) = 1/(1 + e^{-z})$
ReLU	$\max(0, z)$
Forward	$a^{[l]} = f(W^{[l]} a^{[l-1]} + b^{[l]})$
Loss	$L = \frac{1}{n} \sum (y_{pred} - y_{true})^2$
Gradient	$w \leftarrow w - \eta \frac{\partial L}{\partial w}$

Logical Flow

Problem → Neuron → Activation → XOR Crisis → Hidden Layers → Backprop → Theory → Practice → Applications

Key Concepts Checklist

- Neuron = weighted sum
- Weights control importance
- Activation adds non-linearity
- Single neuron = linear
- Hidden layers = non-linear
- XOR impossible alone
- Backprop assigns credit
- Gradient descent optimizes

What's Next

Implement:

- Code from scratch (NumPy)
- Use frameworks (PyTorch/TensorFlow)

Learn:

- Fast.ai, CS231n, Coursera
- Papers: LeNet, AlexNet, ResNet, Attention

Build:

- Image classifier
- Text generator
- Game AI

Resources

- Book: Deep Learning (Goodfellow et al.)
- Viz: playground.tensorflow.org
- Code: github.com/pytorch/examples
- Papers: arxiv-sanity.com

You now understand the fundamentals powering modern AI!