Advanced Topics: Neural Networks Introduction

Perceptron, Multi-layer Perceptrons, and Backpropagation

Sarwan Ali

Department of Computer Science
Georgia State University

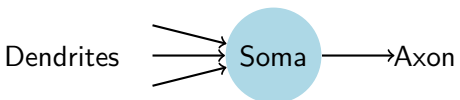

Understanding Neural Networks


Today's Learning Journey

1. Introduction to Neural Networks
2. The Perceptron
3. Multi-layer Perceptrons (MLPs)
4. Backpropagation
5. Applications and Extensions
6. Summary and Key Takeaways

What are Neural Networks?

- **Biological Inspiration:** Inspired by the human brain's network of neurons
- **Mathematical Model:** Computational model consisting of interconnected nodes (artificial neurons)
- **Learning Paradigm:** Learns patterns from data through adjusting connection weights
- **Universal Approximators:** Can approximate any continuous function given sufficient neurons



Why Neural Networks?

Advantages:

- Non-linear pattern recognition
- Parallel processing capability
- Fault tolerance
- Adaptive learning
- Handle noisy data well

Applications:

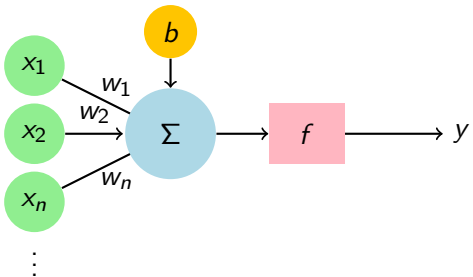
- Image recognition
- Natural language processing
- Speech recognition
- Medical diagnosis
- Financial forecasting

Key Insight

Neural networks excel at finding complex, non-linear relationships in data that traditional linear methods cannot capture.

The Perceptron: Building Block of Neural Networks

- **Definition:** Simplest form of artificial neuron (Frank Rosenblatt, 1957)
- **Binary Classifier:** Separates data into two classes using a linear decision boundary
- **Linear Model:** Computes weighted sum of inputs plus bias



Perceptron Mathematical Model

Mathematical Formulation:

$$\text{Net Input:} \quad z = \sum_{i=1}^{n} w_i x_i + b = \mathbf{w}^T \mathbf{x} + b \tag{1}$$

$$\text{Output:} \quad y = f(z) = \begin{cases} 1 & \text{if } z \geq 0 \\ 0 & \text{if } z < 0 \end{cases} \tag{2}$$

Components:

- $\mathbf{x} = [x_1, x_2, \ldots, x_n]^T$: Input vector
- $\mathbf{w} = [w_1, w_2, \ldots, w_n]^T$: Weight vector
- b: Bias term
- $f(\cdot)$: Step activation function

Geometric Interpretation

The perceptron creates a hyperplane: $\mathbf{w}^T \mathbf{x} + b = 0$ that separates the input space into two regions.

Perceptron Learning Algorithm

Goal: Find weights $\mathbf{w}$ and bias b that correctly classify training data

Algorithm Steps:

1. Initialize weights $\mathbf{w}$ and bias b randomly (or to zero)
2. For each training example $(\mathbf{x}^{(i)}, t^{(i)})$:
   - Compute output: $y^{(i)} = f(\mathbf{w}^T\mathbf{x}^{(i)} + b)$
   - Update weights if prediction is wrong:

Weight Update Rule

$$\mathbf{w} \leftarrow \mathbf{w} + \eta(t^{(i)} - y^{(i)})\mathbf{x}^{(i)} \tag{3}$$

$$b \leftarrow b + \eta(t^{(i)} - y^{(i)}) \tag{4}$$

where η is the learning rate and $t^{(i)}$ is the true label.

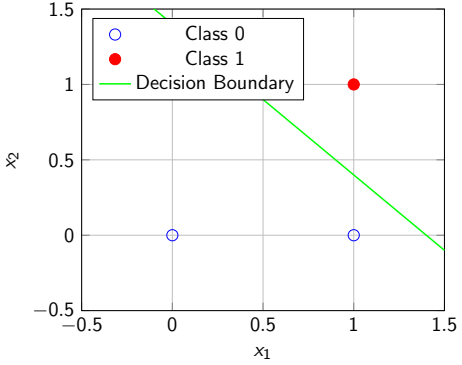
Repeat until convergence or maximum iterations reached

Perceptron Example: AND Gate

Truth Table for AND Gate:

x_1	x_2	y
0	0	0
0	1	0
1	0	0
1	1	1

Solution: $w_1 = 0.5, w_2 = 0.5, b = -0.7$

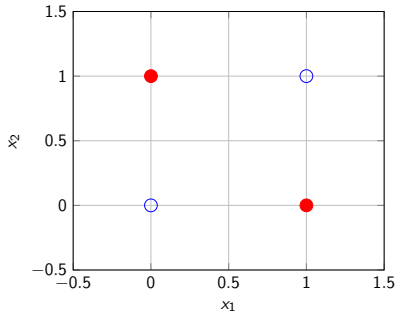


Perceptron Limitations

The XOR Problem: Perceptron cannot solve linearly non-separable problems

XOR Truth Table:

x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	0



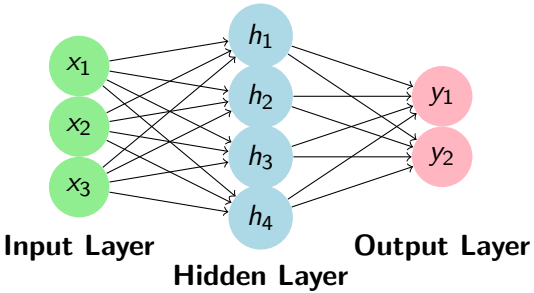
Key Limitation

Linear Separability: Single perceptron can only learn linearly separable functions. No single line can separate the XOR classes!

Solution: Use multiple perceptrons in layers → Multi-layer Perceptrons (MLPs)

Multi-layer Perceptrons: Beyond Linear Separability

Key Idea: Stack multiple layers of perceptrons to learn non-linear functions



Architecture Components:

- **Input Layer:** Receives input features
- **Hidden Layer(s):** Intermediate processing layers
- **Output Layer:** Produces final predictions

MLP Mathematical Formulation

Forward Pass Computation:

For a 3-layer MLP:

$$\text{Hidden Layer:} \quad \mathbf{h} = f(\mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)}) \tag{5}$$

$$\text{Output Layer:} \quad \mathbf{y} = f(\mathbf{W}^{(2)}\mathbf{h} + \mathbf{b}^{(2)}) \tag{6}$$

Notation:

- $\mathbf{W}^{(l)}$: Weight matrix for layer l
- $\mathbf{b}^{(l)}$: Bias vector for layer l
- $f(\cdot)$: Activation function (non-linear)

Universal Approximation Theorem

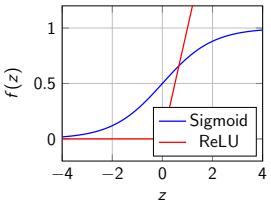
An MLP with at least one hidden layer and sufficient neurons can approximate any continuous function to arbitrary precision.

Activation Functions

Why Non-linear Activations? Without non-linearity, multiple layers collapse to a single linear transformation.

Common Activation Functions:

- **Sigmoid:** $\sigma(z) = \frac{1}{1+e^{-z}}$
- **Tanh:** $\tanh(z) = \frac{e^{z}-e^{-z}}{e^{z}+e^{-z}}$
- **ReLU:** $\text{ReLU}(z) = \max(0, z)$
- **Softmax:** $\text{softmax}(z_i) = \frac{e^{z_i}}{\sum_j e^{z_j}}$

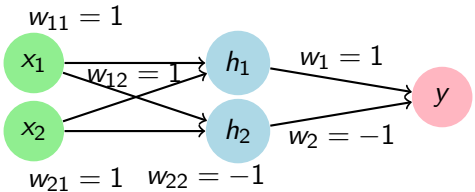


Activation Function Properties

Good activation functions are: non-linear, differentiable, computationally efficient, and avoid vanishing gradients.

Solving XOR with MLP

Architecture: 2 inputs $\rightarrow$ 2 hidden units $\rightarrow$ 1 output



Solution Strategy:

- h_1 learns to detect OR pattern: $h_1 = \sigma(x_1 + x_2 - 0.5)$
- h_2 learns to detect AND pattern: $h_2 = \sigma(x_1 - x_2 + 0.5)$
- Output combines: $y = \sigma(h_1 - h_2 - 0.5) = \text{OR AND NOT AND} = \text{XOR}$

The Learning Challenge: Backpropagation

Problem: How do we train MLPs? The perceptron learning rule doesn't work for hidden layers!

Solution: Backpropagation Algorithm (Rumelhart, Hinton, Williams, 1986)

Key Ideas:

- Use gradient descent to minimize error
- Apply chain rule to compute gradients
- Propagate errors backward through the network
- Update weights layer by layer

Backpropagation Overview

Forward Pass: Compute predictions from inputs to outputs
Backward Pass: Compute gradients from outputs to inputs
Weight Update: Adjust weights using computed gradients

Loss Functions

We need to define what "error" means to minimize it

Common Loss Functions:

- **Mean Squared Error (Regression):**

$$L(\mathbf{y}, \hat{\mathbf{y}}) = \frac{1}{2} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 \tag{7}$$

- **Cross-entropy (Classification):**

$$L(\mathbf{y}, \hat{\mathbf{y}}) = -\sum_{i=1}^{n} y_i \log(\hat{y}_i) \tag{8}$$

Loss Function Properties

Good loss functions are: differentiable, convex (when possible), and provide meaningful gradients for learning.

Backpropagation Algorithm: Mathematical Derivation

Goal: Compute $\frac{\partial L}{\partial w_{ij}}$ for all weights using chain rule

Chain Rule Application:

$$\frac{\partial L}{\partial w_{ij}^{(l)}} = \frac{\partial L}{\partial z_j^{(l)}} \cdot \frac{\partial z_j^{(l)}}{\partial w_{ij}^{(l)}} \tag{9}$$

where $z_j^{(l)} = \sum_i w_{ij}^{(l)} a_i^{(l-1)} + b_j^{(l)}$ is the pre-activation

Define Local Gradient (Delta):

$$\delta_j^{(l)} = \frac{\partial L}{\partial z_j^{(l)}} \tag{10}$$

Then:

$$\frac{\partial L}{\partial w_{ij}^{(l)}} = \delta_j^{(l)} \cdot a_i^{(l-1)} \tag{11}$$

$$\frac{\partial L}{\partial b_j^{(l)}} = \delta_j^{(l)} \tag{12}$$

Computing Deltas: Forward and Backward

Output Layer Delta:

$$\delta_j^{(L)} = \frac{\partial L}{\partial a_j^{(L)}} \cdot \frac{\partial a_j^{(L)}}{\partial z_j^{(L)}} \tag{13}$$

For MSE loss and sigmoid activation:

$$\delta_j^{(L)} = (a_j^{(L)} - y_j) \cdot a_j^{(L)} \cdot (1 - a_j^{(L)}) \tag{14}$$

Hidden Layer Delta (Backpropagation):

$$\delta_i^{(l)} = \left(\sum_j w_{ij}^{(l+1)} \delta_j^{(l+1)} \right) \cdot f'(z_i^{(l)}) \tag{15}$$

Key Insight

Errors propagate backward: hidden layer errors are weighted sums of errors from the next layer, scaled by activation derivatives.

Backpropagation Algorithm Steps

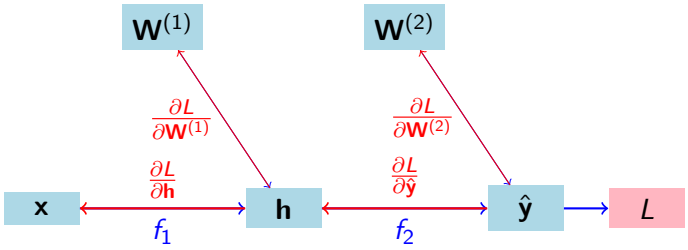
Training Algorithm:

1. **Initialize:** Randomly initialize all weights and biases
2. **For each training example or batch:**
   1. **Forward Pass:**
      - Compute activations for all layers: $a^{(l)} = f(\mathbf{W}^{(l)} a^{(l-1)} + \mathbf{b}^{(l)})$
   2. **Backward Pass:**
      - Compute output layer deltas: $\delta^{(L)} = \nabla_{a^{(L)}} L \odot f'(z^{(L)})$
      - Backpropagate deltas: $\delta^{(l)} = ((\mathbf{W}^{(l+1)})^T \delta^{(l+1)}) \odot f'(z^{(l)})$
   3. **Weight Update:**
      - Update weights: $\mathbf{W}^{(l)} \leftarrow \mathbf{W}^{(l)} - \eta \delta^{(l)} (a^{(l-1)})^T$
      - Update biases: $\mathbf{b}^{(l)} \leftarrow \mathbf{b}^{(l)} - \eta \delta^{(l)}$
3. **Repeat** until convergence or maximum iterations

where η is the learning rate and $\odot$ denotes element-wise multiplication.

Backpropagation: Computational Graph View

Visual representation of gradient flow:



Key Principle: Gradients flow backward through the computational graph, following the chain rule at each node.

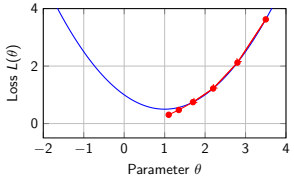
Gradient Descent Optimization

Gradient Descent: Iterative optimization algorithm to minimize loss function
Update Rule:

$$\theta_{t+1} = \theta_t - \eta \nabla_\theta L(\theta_t) \tag{16}$$

where θ represents all parameters (weights and biases)
Variants:

- **Batch Gradient Descent:** Uses entire dataset for each update
- **Stochastic Gradient Descent (SGD):** Uses single example for each update
- **Mini-batch Gradient Descent:** Uses small batch of examples



Training Considerations and Best Practices

Hyperparameter Selection:

- **Learning Rate (η):** Too high $\rightarrow$ divergence, too low $\rightarrow$ slow convergence
- **Network Architecture:** Number of layers and neurons per layer
- **Batch Size:** Balance between computation efficiency and gradient accuracy
- **Epochs:** Number of complete passes through training data

Common Problems:

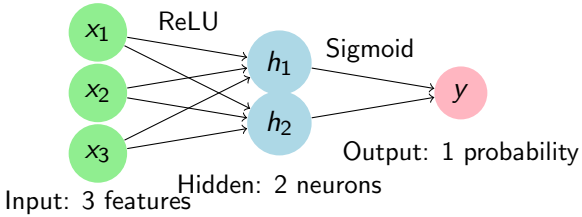
- **Vanishing Gradients:** Gradients become very small in deep networks
- **Exploding Gradients:** Gradients become very large, causing instability
- **Overfitting:** Model memorizes training data, poor generalization
- **Local Minima:** Optimization gets stuck in suboptimal solutions

Best Practice

Use techniques like weight initialization, regularization, and adaptive learning rates to improve training stability and performance.

Practical Implementation Example

Simple 2-layer MLP for Binary Classification:



Forward Pass:

$$\mathbf{h} = \text{ReLU}(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1) \tag{17}$$

$$y = \sigma(\mathbf{w}_2^T\mathbf{h} + b_2) \tag{18}$$

Loss: Binary cross-entropy: $L = -[t\log(y) + (1-t)\log(1-y)]$

Real-World Applications

Computer Vision:

- Image classification and object detection
- Medical image analysis
- Autonomous vehicle perception

Natural Language Processing:

- Sentiment analysis and text classification
- Machine translation
- Chatbots and language models

Other Domains:

- Financial forecasting and fraud detection
- Recommendation systems
- Drug discovery and bioinformatics
- Game playing (AlphaGo, chess engines)

Success Story

Deep neural networks have achieved human-level performance in image recognition, game playing, and many other tasks previously thought impossible for machines.

Modern Extensions and Architectures

Beyond Basic MLPs:

- **Convolutional Neural Networks (CNNs):** For image processing
- **Recurrent Neural Networks (RNNs):** For sequential data
- **Long Short-Term Memory (LSTM):** For long-term dependencies
- **Transformers:** For attention-based processing
- **Generative Adversarial Networks (GANs):** For data generation

Advanced Techniques:

- Dropout and batch normalization
- Residual connections and skip connections
- Attention mechanisms
- Transfer learning and pre-trained models

Evolution

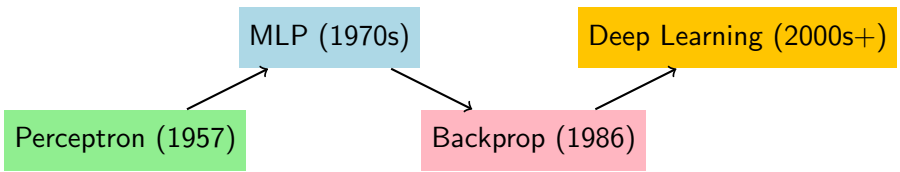
Neural networks have evolved from simple perceptrons to complex architectures with billions of parameters, enabling breakthrough applications in AI.

Summary: From Perceptron to Deep Learning

Key Concepts Covered:

1. **Perceptron:** Linear binary classifier, foundation of neural networks
2. **Multi-layer Perceptrons:** Non-linear function approximators
3. **Activation Functions:** Enable non-linear transformations
4. **Backpropagation:** Efficient algorithm for training MLPs
5. **Gradient Descent:** Optimization technique for parameter updates

Progression Timeline:



Key Mathematical Insights

Universal Approximation:

"Any continuous function can be approximated by a neural network with sufficient hidden units"

Gradient-Based Learning:

$$\frac{\partial L}{\partial w} = \frac{\partial L}{\partial y} \cdot \frac{\partial y}{\partial z} \cdot \frac{\partial z}{\partial w} \tag{19}$$

Non-linearity is Crucial:

- Without activation functions: $f(f(x)) = f^2(x)$ (still linear)
- With activation functions: $\sigma(W_2\sigma(W_1x))$ (non-linear)

Deep Learning Foundation

The principles learned here form the mathematical foundation for all modern deep learning architectures and applications.

Next Steps and Further Learning

Immediate Next Topics:

- Regularization techniques (dropout, L1/L2 regularization)
- Advanced optimization algorithms (Adam, RMSprop)
- Convolutional Neural Networks for computer vision
- Recurrent Neural Networks for sequence modeling

Practical Implementation:

- Implement basic MLP from scratch in Python/NumPy
- Use frameworks like TensorFlow, PyTorch, or Keras
- Experiment with different architectures and hyperparameters
- Apply to real datasets and problems

Resources for Further Study:

- "Deep Learning" by Goodfellow, Bengio, and Courville
- Online courses (Coursera, edX, fast.ai)
- Research papers and conferences (NeurIPS, ICML, ICLR)

Thank You!

Questions and Discussion

Neural Networks: The Foundation of Modern AI

Contact: sali85@student.gsu.edu