# Multilayer Perceptron (MLP)

## Neural Network Fundamentals

### A Comprehensive Introduction

**Marc Reyes**

*Lecturer*

*March 7, 2025*

# Multilayer Perceptron (MLP)

## The Building Blocks of Deep Learning

```
graph TD
    A[Input Layer] --> B[Hidden Layer]
    B --> C[Output Layer]
```

- Foundation of modern neural networks
- Versatile architecture for diverse problems
- Combines simplicity with powerful learning capabilities

# From Neurons to Networks

```
flowchart TD
    D[Dendrites] --> S[Cell Body]
    S --> A[Axon]
    A --> O[Synapse]
```

- **Biological inspiration**: Mimics brain's neural structure
  - Neurons receive, process, and transmit information
- **Artificial neuron**: Weighted sum + activation function
  - Processes inputs through mathematical operations

# From Neurons to Networks (cont.)

```
flowchart TD
    D[Dendrites] --> S[Cell Body]
    S --> A[Axon]
    A --> O[Synapse]
```

- **Network topology**: Input layer → Hidden layers → Output layer
  - Organized structure for information processing
- **Information flow**: Forward propagation for predictions
  - Data travels from input to output through the network

# The Perceptron Journey

```
gantt
    dateFormat  YYYY
    title Perceptron Journey
    section Milestones
    Rosenblatt's Perceptron       :a1, 1958, 1y
    Minsky & Papert (XOR issue)    :a2, 1969, 1y
    Backpropagation introduced     :a3, 1986, 1y
    Modern Neural Networks         :a4, 2025, 1y
```

- **1958**: Rosenblatt's single-layer perceptron
    - First implementation of a neural learning algorithm
- **1969**: Minsky & Papert expose limitations (XOR problem)
    - Demonstrated that single-layer networks couldn’t solve nonlinear problems

# The Perceptron Journey (cont.)

```
gantt
    dateFormat  YYYY
    title Perceptron Journey
    section Milestones
    Rosenblatt's Perceptron       :a1, 1958, 1y
    Minsky & Papert (XOR issue)    :a2, 1969, 1y
    Backpropagation introduced     :a3, 1986, 1y
    Modern Neural Networks         :a4, 2025, 1y
```

- **1986**: Rumelhart, Hinton & Williams introduce backpropagation
    - Breakthrough algorithm enabling training of multi-layer networks
- **Today**: Foundation for advanced architectures (CNNs, RNNs, Transformers)
    - Core concepts extended to specialized network designs

# MLP Architecture

```
graph TD
    I[Input Layer] --> H[Hidden Layer]
    H --> O[Output Layer]
    I -.->|Weights| H
    H -.->|Weights| O
```

## Key Components:

- **Input layer**: Raw data reception
  - Receives and standardizes input features
- **Hidden layers**: Feature extraction and transformation
  - Learns hierarchical representations of data
- **Output layer**: Final prediction/classification
  - Produces the network's answer to the given problem

# MLP Architecture (cont.)

```
graph TD
    I[Input Layer] --> H[Hidden Layer]
    H --> O[Output Layer]
    I -.->|Weights| H
    H -.->|Weights| O
```

## Key Components (continued):

- **Weights & biases**: Learnable parameters
    - Adjusted during training to minimize error
- **Activation functions**: Introduce non-linearity
    - Enable the network to learn complex patterns

# Activation Functions

```
flowchart TD
    AF[Activation Functions]
    AF --> SIG[Sigmoid]
    AF --> TANH[Tanh]
    AF --> RELU[ReLU]
    AF --> LRELU[Leaky ReLU]
```

| Function | Formula | Characteristics |
| --- | --- | --- |
| Sigmoid | $\sigma(x) = \frac{1}{1+e^{-x}}$ | Output range [0,1], vanishing gradient |
| Tanh | $\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$ | Output range [-1,1], zero-centered |

# Activation Functions (cont.)

```
flowchart TD
    AF[Activation Functions]
    AF --> SIG[Sigmoid]
    AF --> TANH[Tanh]
    AF --> RELU[ReLU]
    AF --> LRELU[Leaky ReLU]
```

| Function | Formula | Characteristics |
|---|---|---|
| ReLU | $f(x) = \max(0, x)$ | Computationally efficient, sparse activation |
| Leaky ReLU | $f(x) = \max(0.01x, x)$ | Prevents dying ReLU problem |

# Forward Propagation

```
graph LR
    IN[Input]
    WB[Weights & Biases]
    ACT[Activation]
    OUT[Output]
    IN --> WB
    WB --> ACT
    ACT --> OUT
```

For each layer ( l ):

$$Z^{[l]} = W^{[l]} \cdot A^{[l-1]} + b^{[l]}$$
$$A^{[l]} = g^{[l]}(Z^{[l]})$$

Where:

- (W^{[l]}) = weights matrix

# Forward Propagation (cont.)

```
graph LR
    IN[Input]
    WB[Weights & Biases]
    ACT[Activation]
    OUT[Output]
    IN --> WB
    WB --> ACT
    ACT --> OUT
```

Where:

- (g^{[l]}) = activation function
- (A^{[l]}) = activation output

# Backpropagation: Learning Process

```
flowchart TD
    FP[Forward Pass]
    EC[Error Calculation]
    BP[Backward Pass]
    UP[Update Parameters]
    FP --> EC
    EC --> BP
    BP --> UP
```

1. **Forward pass**: Compute predictions
   - Process inputs through the network
2. **Error calculation**: Compare with ground truth

# Backpropagation: Learning Process (cont.)

```
flowchart TD
    FP[Forward Pass]
    EC[Error Calculation]
    BP[Backward Pass]
    UP[Update Parameters]
    FP --> EC
    EC --> BP
    BP --> UP
```

3. **Backward pass**: Compute gradients
4. **Parameter update**: Adjust weights and biases

$$W^{[l]} = W^{[l]} - \alpha \frac{\partial J}{\partial W^{[l]}}$$

$$b^{[l]} = b^{[l]} - \alpha \frac{\partial J}{\partial b^{[l]}}$$

# Loss Functions

| Task | Loss Function | Formula |
|---|---|---|
| Regression | Mean Squared Error | $\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$ |
| Binary Classification | Binary Cross-Entropy | $-\frac{1}{n}\sum_{i=1}^{n}[y_i\log(\hat{y}_i) + (1-y_i)\log(1-\hat{y}_i)]$ |

# Loss Functions (cont.)

| Task | Loss Function | Formula |
|---|---|---|
| Multi-class Classification | Categorical Cross-Entropy | $(-\frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{m}y_{ij}\log(\hat{y}_{ij}))$ |

- Loss guides the learning process
- Different tasks use specialized error measurements
- Optimization aims to minimize loss

# Universal Approximation Theorem

```
graph LR
    IN[Input]
    HL[Hidden Layer]
    OUT[Output]
    IN --> HL
    HL --> OUT
    HL --- NOTE[Approximates any continuous function]
```

> "A feedforward network with a single hidden layer containing a finite number of neurons can approximate any continuous function, under mild assumptions on the activation function."

- The theoretical foundation for MLP capabilities

# Universal Approximation Theorem (cont.)

```
graph LR
    IN[Input]
    HL[Hidden Layer]
    OUT[Output]
    IN --> HL
    HL --> OUT
    HL --- NOTE[Approximates any continuous function]
```

- More complex functions may require more neurons
- Practical implementations must balance capacity and training challenges

# Visualizing Decision Boundaries

```
graph LR
    IN[Input]
    WB[Weights & Biases]
    ACT[Activation]
    OUT[Output]
    IN --> WB
    WB --> ACT
    ACT --> OUT
```

- **Linear boundaries**: Single-layer perceptrons
    - Separate data with straight lines
- **Non-linear boundaries**: MLPs with hidden layers
    - Can form complex separation surfaces

# Visualizing Decision Boundaries (cont.)

```
graph LR
    IN[Input]
    WB[Weights & Biases]
    ACT[Activation]
    OUT[Output]
    IN --> WB
    WB --> ACT
    ACT --> OUT
```

- **Complexity increases** with deeper architectures
- Explore an interactive demo at perceptron.marcr.xyz

# Quick Quiz: Test Your Knowledge!

## Which of these problems can a single-layer perceptron solve?

A) XOR problem

B) Linear classification

C) Image recognition

D) All of the above

*Use the poll feature to submit your answer!*

# Practical Implementation Challenges

## Poll: What's your biggest challenge with neural networks?

- [ ] Understanding the math
- [ ] Choosing the right architecture
- [ ] Overfitting/underfitting
- [ ] Computational resources
- [ ] Interpreting results

*Share your thoughts in the chat!*

# Practical Implementation

# Thank You!

```
graph TD
    A[Input Layer] --> B[Hidden Layer]
    B --> C[Output Layer]
```

## Contact Information

- Email: hi@marcr.xyz
- Website: marcr.xyz

## Resources

- Interactive Demo: perceptron.marcr.xyz
- Slides: github.com/mabreyes/dlsu-lecture-slides