Multilayer Perceptron (MLP)

Neural Network Fundamentals

A Comprehensive Introduction

Marc Reyes

Lecturer

March 7, 2025

Multilayer Perceptron (MLP)

The Building Blocks of Deep Learning

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

From Neurons to Networks

- **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.)

- **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

- **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.)

- **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

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.)

Key Components:

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

Activation Functions

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.)

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

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
  - Represents connection strengths between neurons
- $b^{[l]}$ = bias vector
  - Allows shifting the activation function

Forward Propagation (cont.)

Where:

- $g^{[l]}$ = activation function
  - Introduces non-linearity to the model
- $A^{[l]}$ = activation output
  - Result passed to the next layer

Backpropagation: Learning Process

1. **Forward pass**: Compute predictions
   - Process inputs through the network to get outputs
2. **Error calculation**: Compare with ground truth
   - Measure the difference between predictions and actual values

Backpropagation: Learning Process (cont.)

3. **Backward pass**: Compute gradients
    - Calculate how each parameter affects the error
4. **Parameter update**: Adjust weights and biases
    - Modify parameters to reduce error on future predictions

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

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

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
    - Quantifies how well the model is performing
- Different problems require different loss functions
    - Each task type has specialized error measurements
- Optimization aims to minimize loss
    - Algorithms search for parameters that reduce error

Universal Approximation Theorem

> "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."

- Theoretical foundation for MLP capabilities
    - Explains why neural networks are so powerful

Universal Approximation Theorem (cont.)

- More complex functions may require more neurons
  - Practical implementation needs sufficient capacity
- Practical limitations: training difficulty, data requirements
  - Theory doesn't guarantee efficient learning

Visualizing Decision Boundaries

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

Visualizing Decision Boundaries (cont.)

- **Complexity increases**: With more layers and neurons
  - Network capacity grows with architecture size
- **Interactive demo**: https://perceptron.marcr.xyz
  - Explore decision boundaries in real-time

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!

Contact Information

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

Resources

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