Multi-Layer Perceptron

A Complete Guide to Understanding MLPs

Mohamed Amine

September 14, 2025

Contents

1 Introduction to Multi-Layer Perceptrons

Key Concept: A Multi-Layer Perceptron (MLP) is one of the simplest yet most fundamental types of artificial neural networks, serving as the foundation for understanding more complex deep learning architectures.

1.1 Why is it called "Multi-Layer Perceptron"?

- **Multi-layer**: Built from several layers of interconnected neurons
- **Perceptron**: Each neuron follows the perceptron concept—a mathematical function that:
  1. Takes multiple inputs
  2. Applies learned weights
  3. Adds a bias term
  4. Passes the result through an activation function

1.2 MLP Architecture

An MLP consists of three essential types of layers:

1.2.1 1. Input Layer

- Receives raw data without any processing
- Number of neurons = Number of input features
- **Example**: For a 28×28 MNIST image:

$$\text{Input neurons} = 28 \times 28 = 784 \text{ neurons}$$

1.2.2 2. Hidden Layers

- One or more fully connected (Dense) layers
- Where the actual *learning* happens
- Each neuron learns patterns by adjusting weights and biases
- Non-linear activation functions (ReLU, sigmoid, tanh) enable complex pattern recognition

1.2.3 3. Output Layer

- Produces the final prediction
- For classification: Number of neurons = Number of classes
- **MNIST example**: 10 output neurons (digits 0–9)
- Often uses softmax activation for probability distribution:

$$\text{softmax}(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

2 The Training Process: Four Essential Steps

Training Overview: Neural network training follows a systematic four-step process that repeats until the model learns effectively.

2.1 Step 1: Forward Propagation

Goal: Pass input data through the network to generate a prediction.

Process:

1. Feed input data into the network
2. For each neuron, compute the weighted sum:

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

   where w_i are weights, x_i are inputs, and b is the bias

3. Apply activation function:

$$a = f(z)$$

4. Pass result to the next layer
5. Output layer produces final prediction

2.2 Step 2: Loss Function

Goal: Measure how wrong the network's prediction is.

The loss function quantifies the difference between prediction and true label:

$$\text{Loss} = L(\text{prediction}, \text{true label})$$

Common Loss Functions:

- **Cross-entropy loss** (classification):

$$L = -\sum_{i=1}^{C} y_i \log(\hat{y}_i)$$

  where y_i is true label and $\hat{y}_i$ is predicted probability

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

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

2.3 Step 3: Backpropagation

Goal: Calculate how much each weight contributed to the error.

Using the **chain rule of calculus**, backpropagation computes:

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

These gradients indicate the "direction" each parameter should move to reduce error.

2.4 Step 4: Optimization

Goal: Update weights to reduce the loss.

Common Optimizers:

- **Stochastic Gradient Descent (SGD)**:

$$w_{\text{new}} = w_{\text{old}} - \eta \frac{\partial L}{\partial w}$$

- **Adam Optimizer**: Adaptive learning rates with momentum

where η is the learning rate.

3 Activation Functions: The Key to Non-linearity

Purpose: Activation functions determine whether a neuron should "activate" (fire) based on its input. They introduce **non-linearity**, enabling networks to learn complex patterns.

3.1 Why Activation Functions Matter

Without activation functions: The network becomes just a linear combination of inputs, regardless of the number of layers:

$$\text{Linear network output} = W_n \cdot W_{n-1} \cdot \ldots \cdot W_1 \cdot x = W_{\text{combined}} \cdot x$$

With activation functions: The network can approximate any continuous function (Universal Approximation Theorem).

3.2 Common Activation Functions

3.2.1 1. Sigmoid Function

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

Properties:

- Range: $[0, 1]$
- Smooth gradient
- Often used in output layer for binary classification
- **Drawback**: Vanishing gradient problem for deep networks

3.2.2 2. ReLU (Rectified Linear Unit)

$$f(z) = \max(0, z) = \begin{cases} z & \text{if } z > 0 \\ 0 & \text{if } z \leq 0 \end{cases}$$

Properties:

- Range: $[0, +\infty)$
- Computationally efficient
- Helps avoid vanishing gradient problem
- Most popular for hidden layers
- **Drawback**: "Dead neurons" (neurons that always output 0)

3.2.3 3. Tanh (Hyperbolic Tangent)

$$\tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}} = \frac{2}{1 + e^{-2z}} - 1$$

Properties:

- Range: $[-1, 1]$
- Zero-centered (unlike sigmoid)
- Steeper gradient than sigmoid
- Good for hidden layers when you need negative outputs

4 Key Advantages of MLPs

1. **Universal Approximation**: Can theoretically approximate any continuous function
2. **Flexibility**: Suitable for both classification and regression tasks
3. **Foundation Knowledge**: Understanding MLPs helps grasp more complex architectures
4. **Non-linear Modeling**: Can capture complex relationships in data

Summary

Multi-Layer Perceptrons provide an excellent introduction to neural networks and deep learning. The four-step training process forms the backbone of virtually all neural network training, making MLPs an essential foundation for understanding modern deep learning architectures.