# Activation Function

## Sigmoid Activation Function

**Name:** Sigmoid (Logistic Function)  
**Formula:**

**Range:** (0, 1)  
**When to Use:**

* Best for **binary classification** (output layer).
* When you need output as a **probability**.

**Effects / Characteristics:**

* Smooth and differentiable.
* **Squashes** input into (0,1) range.

**Typical use**

* **Output layer** for binary classification **only** if you want a single probability output (paired with binary crossentropy). e.g. last layer: Dense(1, activation='sigmoid').
* Rarely used in hidden layers for modern deep networks.

**Example**  
Binary classifier with single output neuron.

### Tanh (Hyperbolic Tangent) Activation Function

**Name:** Hyperbolic Tangent (tanh)  
**Formula:**

**Range:** (-1, 1)  
**When to Use:**

* Hidden layers when data is **centered around zero**.
* Preferred over sigmoid in hidden layers (historically).

**Effects / Characteristics:**

* Zero-centered output → helps faster convergence.
* Still suffers from **vanishing gradients** for large |x|.

### ReLU (Rectified Linear Unit)

**Name:** ReLU  
**Formula:**

**Range:** [0, ∞)

**Shape / Intuition:** Keeps positive inputs unchanged, zeroes negative inputs. Simple and effective.  
**When to Use:**

* **Most commonly used** in hidden layers of CNNs and DNNs.
* When you need **fast training and sparse activation**.
* **Default hidden-layer activation** for many networks (CNNs, dense nets). e.g. Dense(128, activation='relu').

**When not to use**

* On outputs that need probabilities or negative outputs. For those, use sigmoid/tanh/softmax as appropriate.

**Effects / Characteristics:**

* Computationally efficient.
* Avoids vanishing gradients for positive x.
* Causes **dead neurons** (when x < 0, gradient = 0 forever).

### Leaky ReLU

**Name:** Leaky ReLU  
**Formula:**

(typically α = 0.01)

**Range:** (-∞, ∞)  
**When to Use:**

* When you want to avoid **dead neurons** of ReLU.

**Effects / Characteristics:**

* Allows a **small gradient** for negative values.
* Performs better in practice than ReLU for some datasets.

**Typical use**

* **Hidden layers** when you worry about dying ReLUs or when plain ReLU underperforms. Popular in some CNN architectures and GANs.

### Softmax

**Name:** Softmax  
**Formula:**

**Range:** (0, 1), and outputs sum to 1  
**When to Use:**

* **Output layer for multi-class classification**.

**Typical use**

* **Output layer** for multi-class classification: Dense(num\_classes, activation='softmax').

**Effects / Characteristics:**

* Converts logits into **probability distribution**.
* Sensitive to large input values (can cause overflow).

# Loss Function

A **loss function** (or **cost function**) measures **how far the model’s predictions are from the actual truth (labels)**.

**Goal:**

**Why We Use Loss Functions**

Without a loss function, the model wouldn’t know what to improve.

Loss functions:

* Quantify prediction errors.
* Guide gradient descent updates.
* Determine **what the model learns** (classification, regression, etc.).
* Influence **training stability and speed**.

## Types of Loss Functions

They depend on your **task type**:

* Classification
* Regression

## A. Loss Functions for Classification

**Binary Cross-Entropy (Log Loss)**

Used for **binary classification** (0/1 labels).

**Formula:**

**Intuition:**

* Penalizes confident wrong predictions heavily.
* Works with **sigmoid** output (since it outputs probabilities 0–1).

**Example use:**

model.compile(optimizer='adam', loss='binary\_crossentropy', metrics=['accuracy'])

**Example problem:**  
Spam (1) vs. Not spam (0)

**Categorical Cross-Entropy**

Used for **multi-class classification** (mutually exclusive classes).

**Formula:**

**Key:**

* Works with **Softmax** activation (outputs probability distribution).
* Label format: **one-hot encoded** (e.g., [0, 0, 1, 0]).

**Example:**

model.compile(loss='categorical\_crossentropy', optimizer='adam', metrics=['accuracy'])

**Example problem:**  
Classify image as {cat, dog, horse} (only one label).

**Sparse Categorical Cross-Entropy**

Same as above, but target labels are **integers** (not one-hot vectors).

**Example:**

model.compile(loss='sparse\_categorical\_crossentropy', optimizer='adam')

**Why useful:** avoids manual one-hot encoding.

## Loss Functions for Regression

**Mean Squared Error (MSE)**

**Meaning:** Penalizes large errors quadratically → good for continuous value prediction.

**Example:**

model.compile(loss='mse', optimizer='adam')

**Use for:** Predicting house prices, stock prices, etc.

**Mean Absolute Error (MAE)**

**Meaning:** Measures absolute difference — less sensitive to outliers than MSE.

**When to use:**  
When outliers exist or you care about median-like behavior.

**Example:**

model.compile(loss='mae', optimizer='adam')

# Optimizers

An optimizer is an algorithm that adjusts the weights and biases of a neural network to minimize the loss function.

Why Do We Use Optimizers?

Neural networks learn by:

1. Making predictions
2. Measuring the error (loss)
3. Using backpropagation to compute gradients of that loss with respect to every weight
4. The optimizer updates each weight in the direction that reduces the loss.

## Most Commonly Used

1. Gradient Descent

2. Adam