Activation Functions

By Anshum Banga

This document provides a comprehensive comparison of common activation functions used in deep learning models. Each function is evaluated based on its typical range, suitability for use in hidden or output layers, advantages, and drawbacks. These functions play a crucial role in enabling neural networks to learn and make complex decisions.

Table of Contents

Overview

Activation functions introduce **non-linearity** in neural networks — a crucial step that allows them to **learn from complex data**. Without them, no matter how many layers a model has, it behaves like a plain linear regression. Activations help the network **decide when to "fire" a neuron**, just like our brain.

1. Sigmoid Function

What is Sigmoid?

Sigmoid is one of the oldest activation functions and was widely used in early neural networks. It squashes any real number input into a value between **0 and 1**. This makes it great for models where we want to predict **probabilities**.

Think of it as a **confidence meter**: no matter how large or small the input is, the output stays in the range of 0 to 1.

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

Range:

- **(0, 1)**

Use Case:

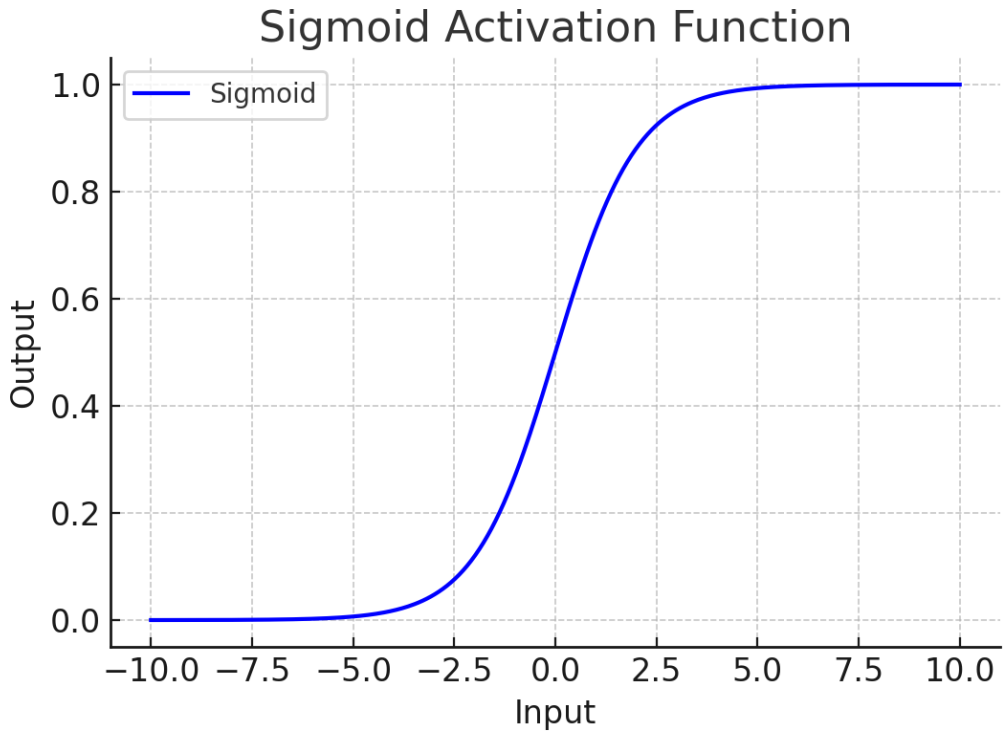
- Mostly used in the **output layer of binary classification** problems.
    - Example: Predicting whether a customer will **buy or not buy** a product.

Pros:

- Gives output in the form of **probability**
- Smooth and continuous
- Works well when the output must be between 0 and 1

Cons:

- **Vanishing gradient**: For very large or small input values, the gradient becomes very small, and the model stops learning.
- **Not zero-centered**: Can make optimization harder.



Example:

You build a neural network to predict whether someone has a disease (1) or not (0). The final layer uses a sigmoid function. The output is something like **0.92**, which you interpret as a **92% chance** that the person has the disease.

2. Tanh (Hyperbolic Tangent)

What is Tanh?

Tanh is similar to sigmoid but better in many ways. It also compresses input values, but its output range is **-1 to 1**. This helps the network learn better because the values are **centered around 0**, which helps gradients flow better.

$$\tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$

Range:

- **(-1, 1)**

Use Case:

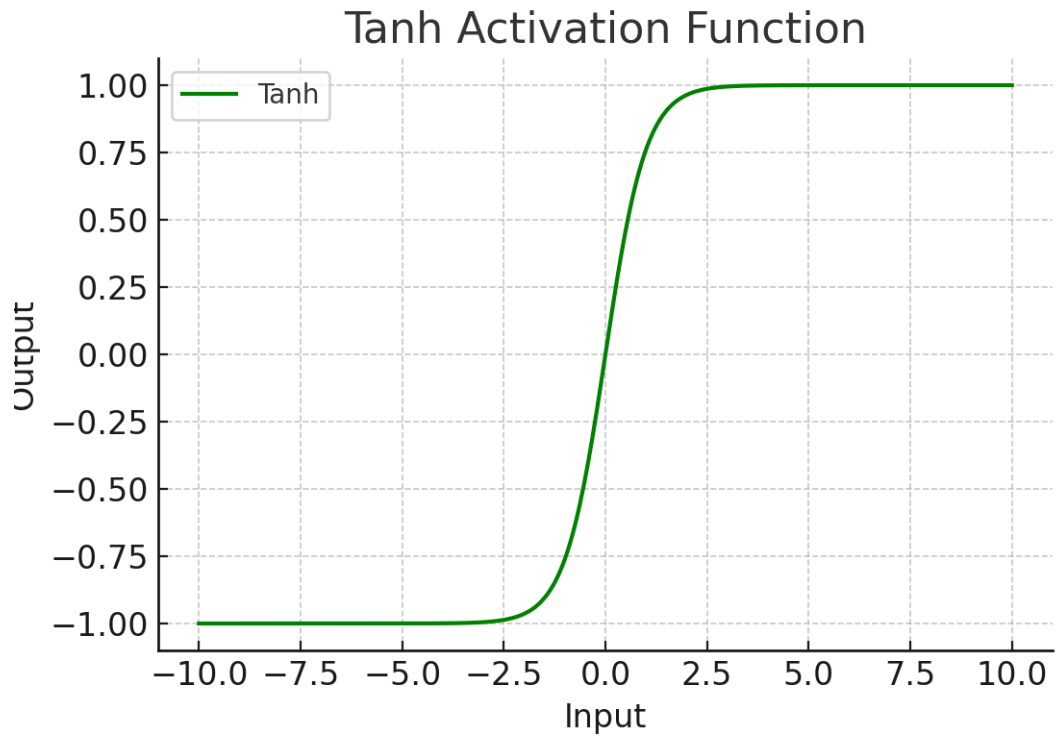
- Used in **hidden layers** to allow both positive and negative activations.

Pros:

- Zero-centered → better learning
- Steeper gradient than sigmoid

Cons:

- Still suffers from **vanishing gradient** at extremes



Example:

In a **time series model** like weather prediction, tanh helps in outputting values that oscillate — like temperature changes from hot (+) to cold (-).

3. ReLU (Rectified Linear Unit)

What is ReLU?

ReLU is the most widely used activation function in deep learning. It outputs the input directly if it's positive; otherwise, it returns 0. It's **simple**, **fast**, and works well in most scenarios.

$$f(x) = \max(0, x)$$

Range:

- **[0, ∞)**

Use Case:

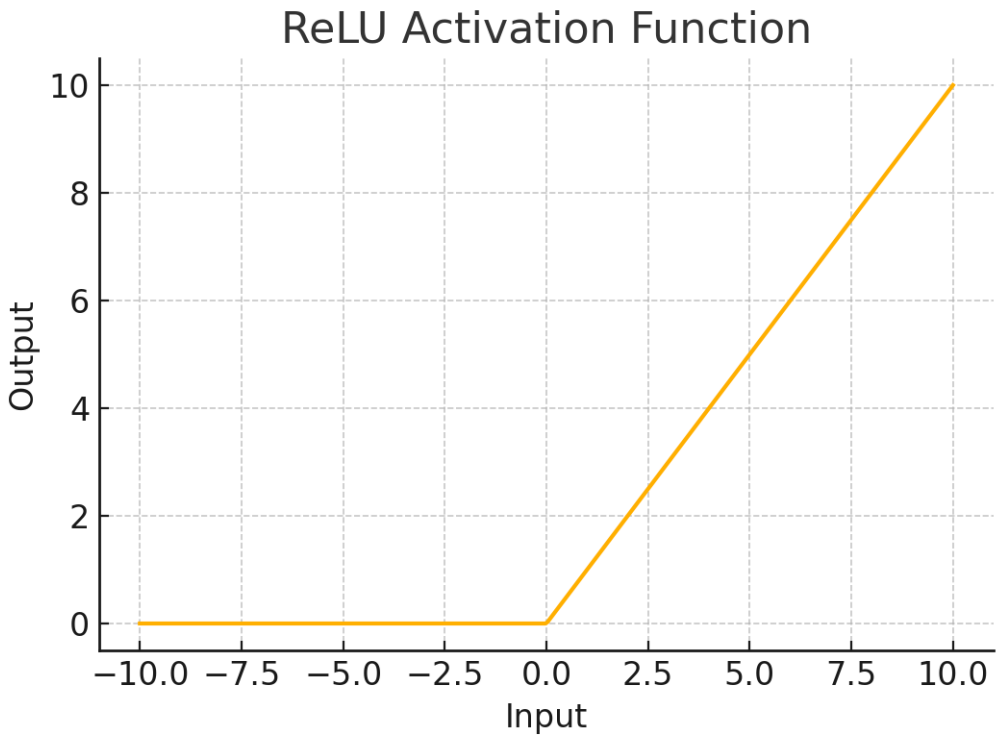
- Hidden layers of deep networks (CNNs, DNNs)

Pros:

- Very simple and fast
- Solves vanishing gradient for positive inputs
- Introduces sparsity — helps focus only on important neurons

Cons:

- **Dying ReLU problem**: Some neurons may always output 0 and stop learning



Example:

ReLU is used in models like **face recognition** or **image classification** to detect edges, curves, or facial features by filtering out unimportant signals.

4. Leaky ReLU

What is Leaky ReLU?

Leaky ReLU tries to fix the dying ReLU problem by allowing a **small negative slope** instead of outputting 0 for negative values.

$$\mathrm{f(x)} = \begin{cases} x & \text{if } x > 0 \\ \alpha x & \text{if } x \leq 0 \end{cases}, \text{where } \alpha \approx 0.01$$

Range:

- **(-∞, ∞)**

Use Case:

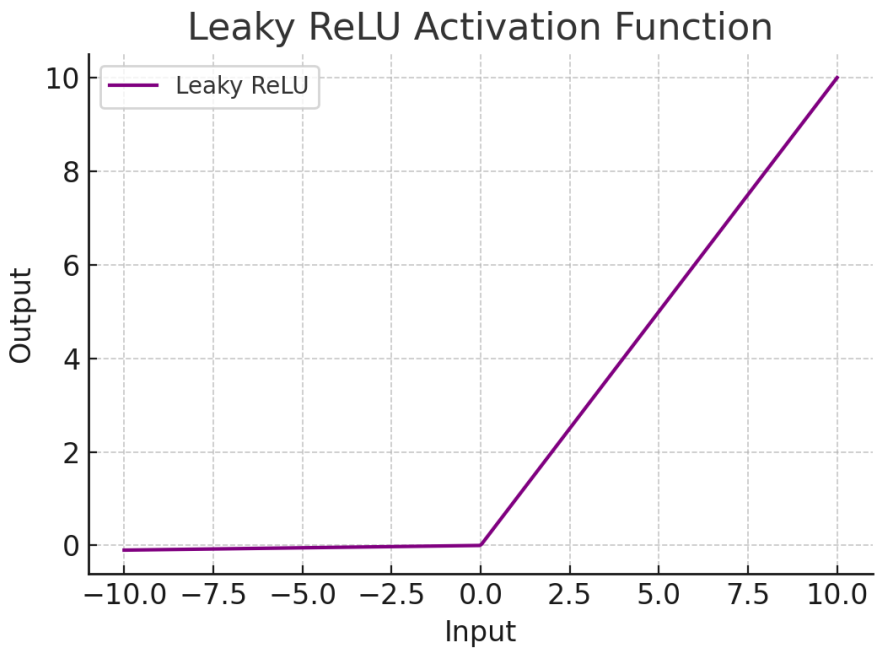
- Deep networks where some neurons might die out

Pros:

- Prevents dying neurons
- Keeps a small gradient for negative inputs

Cons:

- Gradient for negative values is very small — may still slow down learning



Example:

Used in **GANs** (image generators) or **deep classification models** where normal ReLU causes some features to be ignored.

5. PReLU (Parametric ReLU)

What is PReLU?

PReLU is an advanced version of Leaky ReLU. Instead of using a fixed small value (like 0.01), it lets the model **learn** the best negative slope value during training.

$$f(x) = \max(\alpha x, x), \quad \alpha \text{ is learned}$$

Use Case:

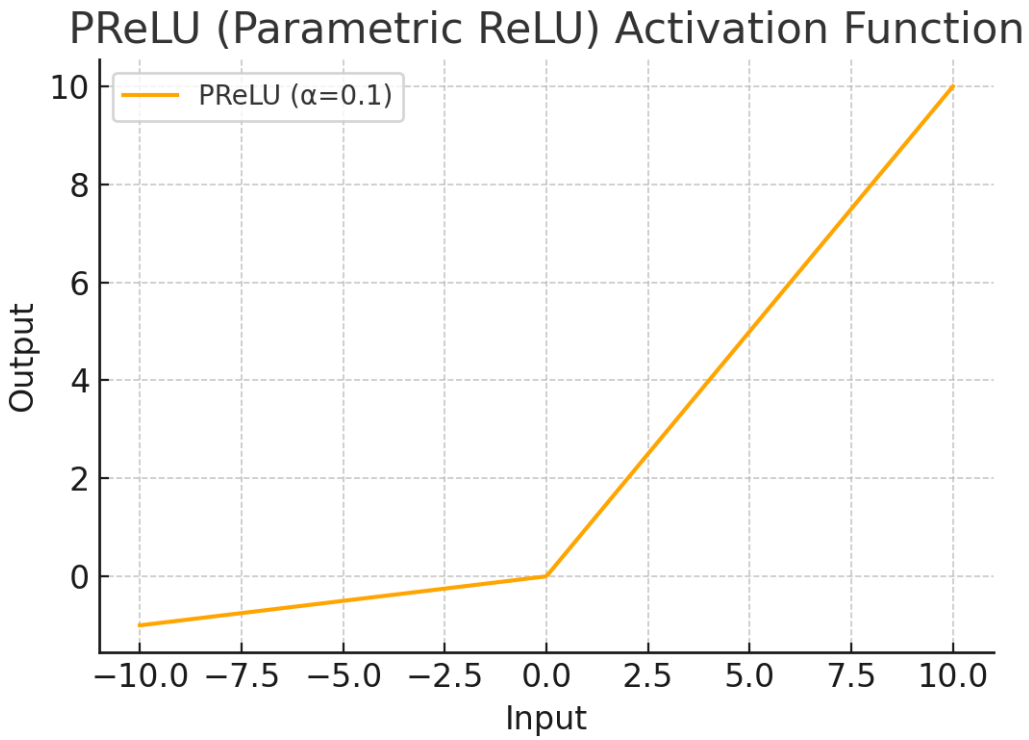
- When you want the network to adapt better during training

Pros:

- Learns better negative slopes
- More flexible than ReLU and Leaky ReLU

Cons:

- Adds more parameters → can overfit



Example:

Used in **object detection systems** or **medical imaging models** where subtle patterns are important and flexibility helps.

6. ELU (Exponential Linear Unit)

What is ELU?

ELU behaves like ReLU for positive values but uses an **exponential function** for negative values. It makes the network faster and more accurate by pushing mean activations closer to zero.

$$f(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha(e^x - 1) & \text{if } x \leq 0 \end{cases}$$

Use Case:

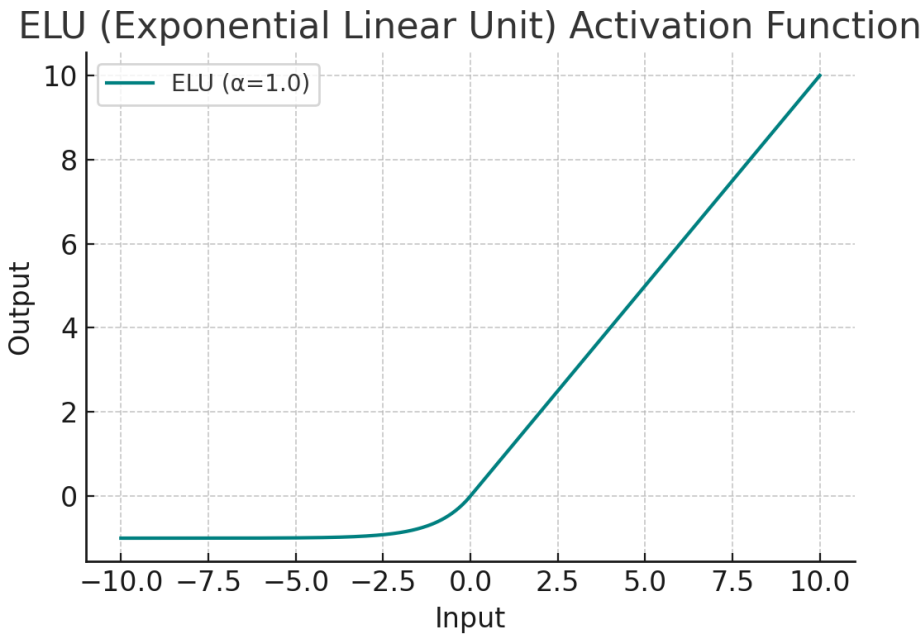
- Deep models that need better performance than ReLU

Pros:

- Zero-centered
- Smooth gradient
- Helps model learn faster

Cons:

- A bit slower to compute than ReLU



Example:

Great for **speech recognition models** or **biometric scanners** where smoother output matters.

7. SELU (Scaled ELU)

What is SELU?

SELU is a self-normalizing activation function. It keeps the outputs of neurons **automatically scaled** during training, helping the model stay stable.

$$\text{SELU}(x) = \lambda \cdot f(x), \quad \text{f(x) is ELU}$$

Use Case:

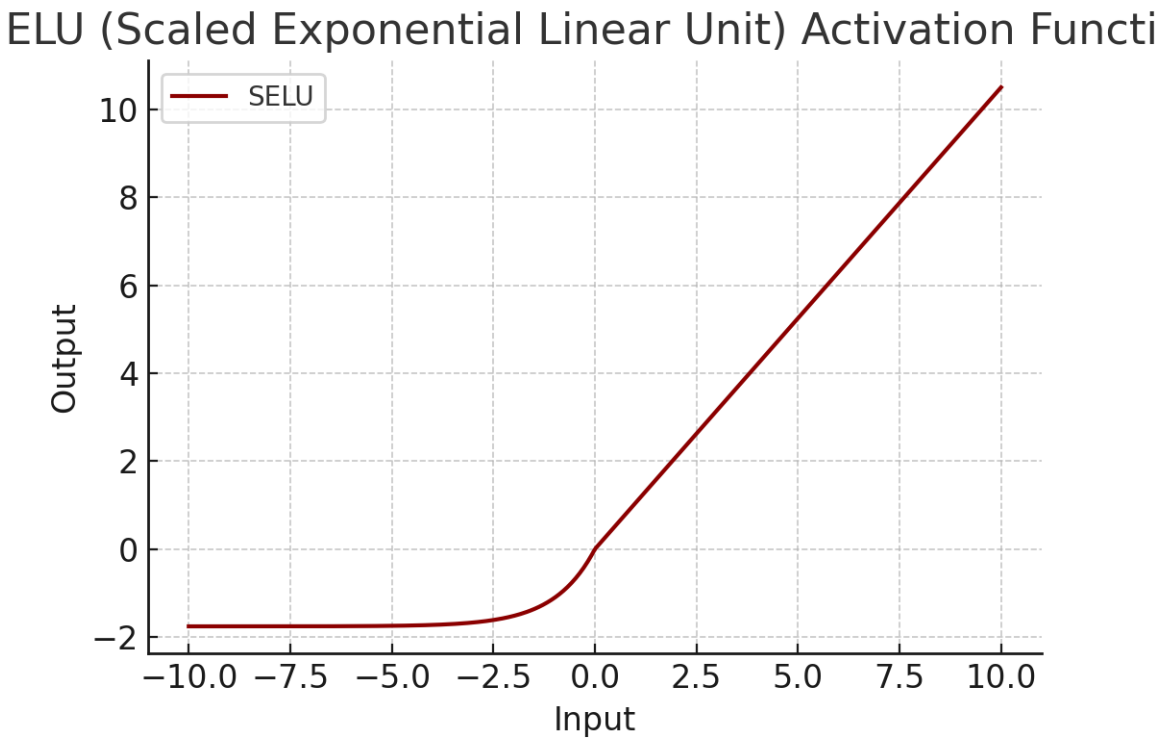
- Used in **Self-Normalizing Neural Networks (SNN)**

Pros:

- Keeps mean and variance stable
- Good for very deep networks

Cons:

- Requires specific weight initialization and input normalization



Example:

Perfect for **deep numeric forecasting models** (e.g., energy consumption prediction).

8. Softmax

What is Softmax?

Softmax turns raw scores (logits) into **probabilities**. It's used when we have **more than two classes** to choose from.

$$\text{softmax}(x_i) = \frac{e^{x_i}}{\sum_j e^{x_j}}$$

Range:

- (0, 1) — and all outputs **sum to 1**

Use Case:

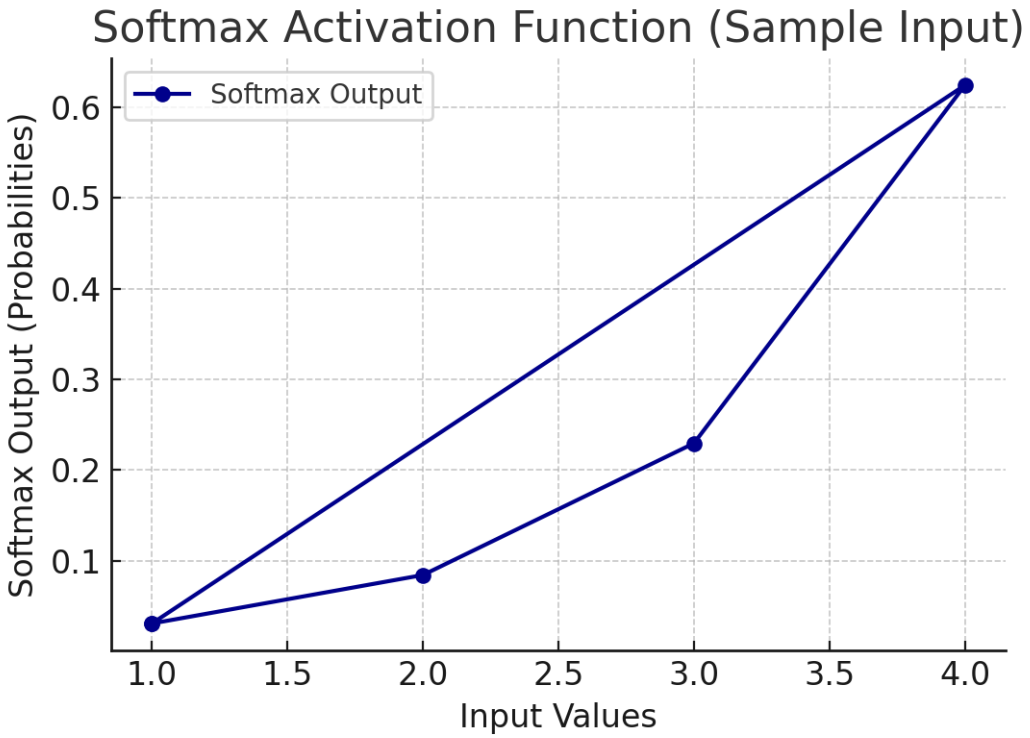
- Final layer in **multi-class classification** problems

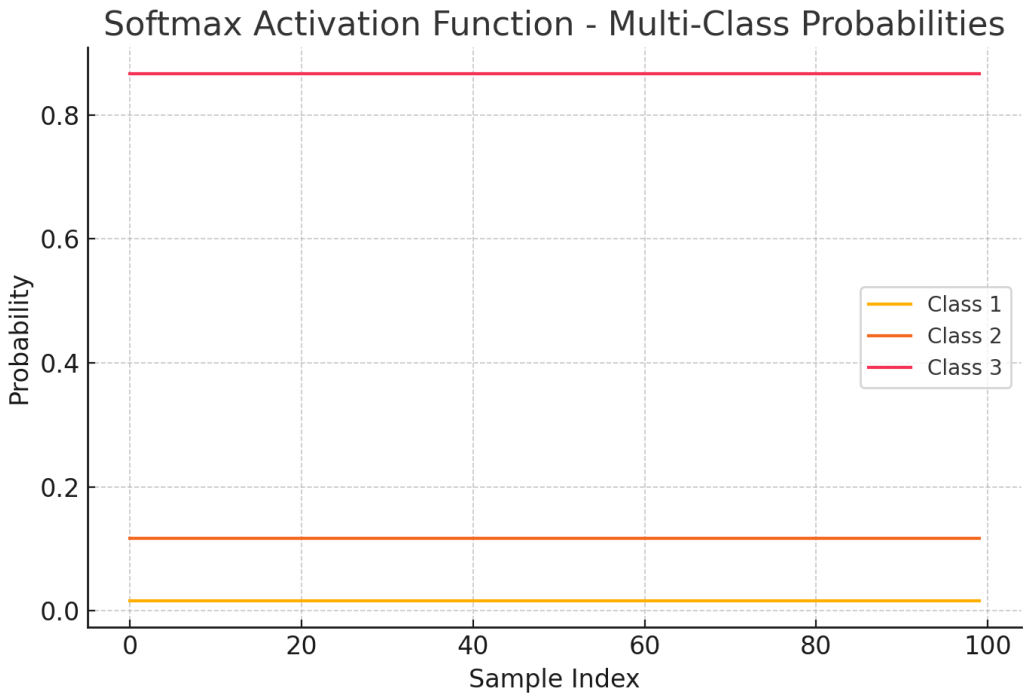
Pros:

- Easy to interpret outputs
- Helps pick the most likely class

Cons:

- Not used in hidden layers





Example:

Used in models like **digit classification** from images (0–9). It gives the probability for each digit, and we pick the highest one.

9. Swish (by Google)

What is Swish?

Swish is a newer activation function developed by Google. It's a smooth and flexible version of ReLU that often performs better in deep networks.

$$f(x) = x \cdot \sigma(x)$$

Use Case:

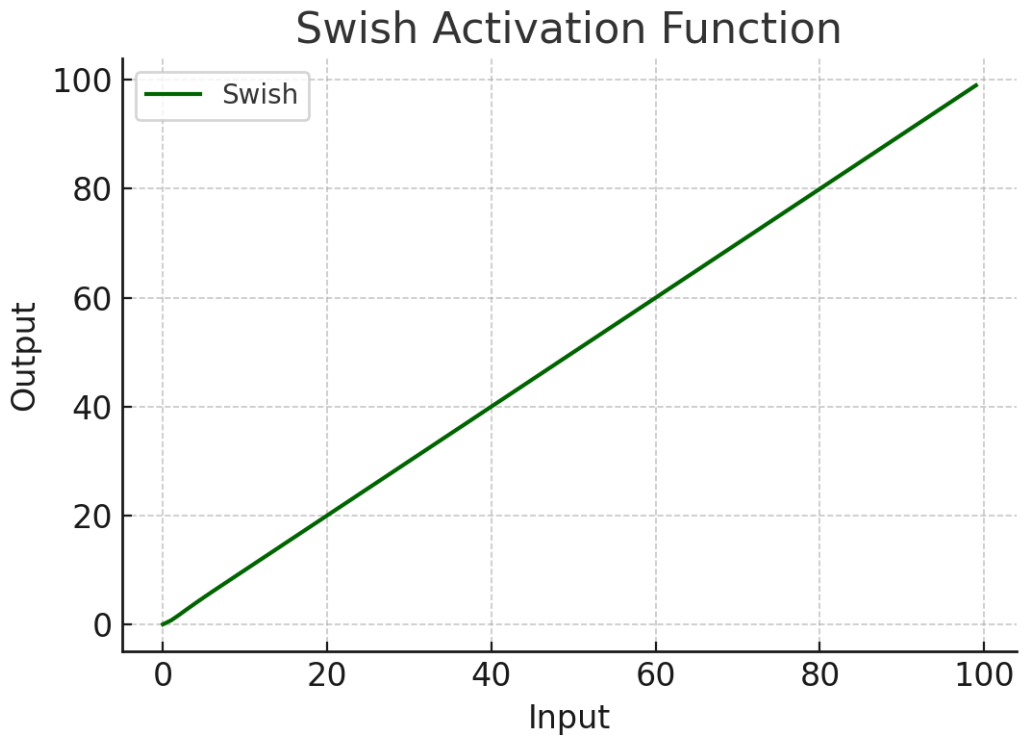
- Deep networks (like EfficientNet) for high-performance tasks

Pros:

- Smooth and non-linear
- Can outperform ReLU in large models

Cons:

- Slower to compute than ReLU



Example:

Used in **image enhancement models** or **language translation tools** where small accuracy boosts are worth the extra computing.

Summary Table

Activation	Range	Use Case	Hidden Layer	Output Layer	✅ Pros	❌ Cons
Sigmoid	(0, 1)	Binary classification (output)	❌	✔	Probabilities, simple	Vanishing gradient, not zero-centered
Tanh	(-1, 1)	Hidden layers with centered data	✔	❌	Zero-centered, stronger gradient	Still vanishing gradient
ReLU	[0, ∞)	Default in hidden layers	✔	❌	Fast, sparse, efficient	Dying neurons
Leaky ReLU	(-∞, ∞)	Robust deep learning	✔	❌	Fixes dying neurons	Gradient still small
PReLU	(-∞, ∞)	Adaptive activation	✔	❌	Learns slope α	Overfitting, more parameters
ELU	(-∞, ∞)	Deep stable learning	✔	❌	Smooth, zero-centered	Slower, needs tuning
SELU	(-∞, ∞)	Self-normalizing networks	✔ (conditional)	❌	Stabilizes activations	Architecture-specific
Softmax	(0, 1)	Multi-class output (only)	❌	✔	Probabilities, interpretable	Not for hidden layers
Swish	(-∞, ∞)	Modern deep networks	✔	❌	Smooth, high accuracy in deep nets	Slower than ReLU

Note:

If your neural network is solving a **regression problem** (i.e., predicting **continuous numeric values**), the best choice for the **output activation function** is usually:

No Activation Function (Linear Activation)

- **Formula**:

$$f(x) = x$$

- In practice, this means you **don't apply any activation** on the output layer — it directly returns the weighted sum of inputs.
- The output is **not squashed** like it would be with Sigmoid, Tanh, or Softmax.

Why this is best for regression?

- **Regression tasks** need unrestricted output values (e.g., price, age, temperature).
- If you use Sigmoid (0 to 1) or Tanh (-1 to 1), your model will **struggle to predict values outside that range**.
- A linear output gives your model the **freedom to learn any real value** depending on the data.