Day-2: Understanding Activation Functions in Deep Learning

---

1. Introduction

Activation functions are the heart of neural networks.
They determine whether a neuron should activate (fire) or remain inactive, thus shaping how information flows through the network.

Each neuron performs two major operations:

1. **Weighted Summation**
   The neuron computes a linear combination of inputs and weights:
   z=w0+w1x1+w2x2+…+wnxnz = w_0 + w_1x_1 + w_2x_2 + \ldots + w_nx_nz=w0+w1x1+w2x2+…+wnxn
2. **Activation**
   The computed value zzz is passed through an activation function f(z)f(z)f(z), which transforms it into the neuron's output signal yyy:
   y=f(z)y = f(z)y=f(z)

Without activation functions, even a multi-layered neural network would behave like a **single linear transformation**, incapable of capturing non-linear patterns in data.

---

2. Purpose of Activation Functions

1. **Introduce Non-Linearity**
   They enable neural networks to model complex relationships beyond linear regression or logistic models.

2. **Control the Flow of Information**
   Activation functions decide which neurons contribute to the output, acting like *decision gates*.

3. **Stabilize Learning**
   Proper choice of activation helps gradients flow effectively during backpropagation, preventing issues like vanishing or exploding gradients.

---

3. Types of Activation Functions

We can broadly categorize them into two groups:

- **Linear Activation Functions**
- **Non-Linear Activation Functions**

3.1 Linear (Identity) Activation

Formula:

f(x)=xf(x) = xf(x)=x

Range: (-∞, ∞)

Use Case:
Primarily used in output layers for regression tasks.

Limitation:
All layers remain linear; stacking layers does not increase model complexity.

3.2 Step Function (Binary Threshold Neuron)

Formula:

f(x)={1,if x≥00,otherwisef(x) = \begin{cases} 1, & \text{if } x \geq 0 \\ 0, & \text{otherwise} \end{cases}f(x)={1,0,if x≥0otherwise

Range: {0, 1}

Use Case:
Early Perceptron models (e.g., McCulloch–Pitts neuron).

Limitation:
Non-differentiable and unsuitable for gradient-based optimization.

3.3 Sigmoid Function

Formula:

σ(x)=11+e−x\sigma(x) = \frac{1}{1 + e^{-x}}σ(x)=1+e−x1

Range: (0, 1)

Derivative:

σ′(x)=σ(x)(1−σ(x))\sigma'(x) = \sigma(x)(1 - \sigma(x))σ′(x)=σ(x)(1−σ(x))

Use Case:
Binary classification; outputs interpreted as probabilities.

Pros:

- Smooth and differentiable
- Maps input to a bounded range

Cons:

- Causes **vanishing gradients** for large |x|
- Output not zero-centered

3.4 Hyperbolic Tangent (Tanh)

Formula:

tanh(x)=ex−e−xex+e−x\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}tanh(x)=ex+e−xex−e−x

Range: (-1, 1)

Derivative:

tanh′(x)=1−tanh2(x)\tanh'(x) = 1 - \tanh^2(x)tanh′(x)=1−tanh2(x)

Pros:

- Zero-centered outputs (helps optimization)
- Steeper gradient than Sigmoid

Cons:

- Still suffers from vanishing gradients for large |x|

Use Case:
Hidden layers in shallow neural networks.

3.5 ReLU (Rectified Linear Unit)

Formula:

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

Range: [0, ∞)

Derivative:

f′(x)={1,x>00,x≤0f'(x) = \begin{cases} 1, & x > 0 \\ 0, & x \le 0 \end{cases}f′(x)={1,0,x>0x≤0

Pros:

- Computationally efficient
- Reduces vanishing gradient issue
- Sparse activation → faster convergence

Cons:

- **Dead ReLU Problem:** Neurons can get stuck outputting zero if weights don’t update.

Use Case:
Default activation for hidden layers in most CNNs and MLPs.

3.6 Leaky ReLU

Formula:

f(x)={x,x>0αx,x≤0f(x) = \begin{cases} x, & x > 0 \\ \alpha x, & x \le 0 \end{cases}f(x)={x,αx,x>0x≤0

Typically, α=0.01\alpha = 0.01α=0.01

Range: (-∞, ∞)

Pros:

- Fixes “dead ReLU” issue
- Allows small gradients for negative inputs

Use Case:
An improved version of ReLU in deep CNNs.

3.7 Parametric ReLU (PReLU)

Formula:

f(x)={x,x>0ax,x≤0f(x) = \begin{cases} x, & x > 0 \\ a x, & x \le 0 \end{cases}f(x)={x,ax,x>0x≤0

where aaa is **learnable** during training.

Pros:

- Learns optimal negative slope
- Provides flexibility across layers

3.8 Exponential Linear Unit (ELU)

Formula:

f(x)={x,x>0α(ex−1),x≤0f(x) = \begin{cases} x, & x > 0 \\ \alpha (e^x - 1), & x \le 0 \end{cases}f(x)={x,α(ex−1),x>0x≤0

Pros:

- Produces negative outputs (mean activations closer to zero)
- Reduces bias shift

Cons:

- More computationally expensive due to exponential term

3.9 Softmax Function

Formula:

f(xi)=exi∑j=1nexjf(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}}f(xi)=∑j=1nexjexi

Range: (0, 1)

Use Case:
Used in output layers for **multi-class classification**.

Properties:

- Converts raw scores (logits) into class probabilities
- Ensures sum of outputs = 1

---

4. Comparison of Activation Functions

Function	Formula	Range	Derivative	Pros	Cons	Typical Use
Step	1 if $x \geq 0$ else 0	$\{0,1\}$	—	Simple, intuitive	Non-differentiable	Binary threshold models
Sigmoid	$1/(1+e^{-x})$	$(0,1)$	$\sigma(1-\sigma)$	Smooth, probabilistic	Vanishing gradient	Binary classification
Tanh	$(e^{x}-e^{-x})/(e^{x}+e^{-x})$	$(-1,1)$	$1-\tanh^2(x)$	Zero-centered	Still saturates	Hidden layers
ReLU	$\max(0,x)$	$[0,\infty)$	1 or 0	Fast, sparse	Dead neurons	Hidden layers (CNNs)
Leaky ReLU	$\max(\alpha x,x)$	$(-\infty,\infty)$	α or 1	Avoids dead ReLU	Needs α tuning	Deep CNNs
ELU	x if $x>0$ else $\alpha(e^{x}-1)$	$(-\alpha,\infty)$	1 or $f(x)+\alpha$	Zero-centered	Expensive	Deep nets
Softmax	$e^{(x_i)}/\Sigma e^{(x_j)}$	$(0,1)$	complex	Probabilistic output	Sensitive to large logits	Output layer (multi-class)

---

5. Choosing the Right Activation Function

Task Type	Recommended Activations
Binary Classification	Sigmoid, Tanh
Multi-Class Classification	Softmax
Regression	Linear
Hidden Layers (General)	ReLU, Leaky ReLU, ELU
Deep CNNs	ReLU / Leaky ReLU
RNNs / LSTMs	Tanh, Sigmoid

---

6. Common Pitfalls

1. **Vanishing Gradients** — In Sigmoid/Tanh networks, gradients shrink as layers increase.
   → *Use ReLU or its variants.*

2. **Dead Neurons** — ReLU neurons stuck at 0.
   → *Switch to Leaky ReLU or PReLU.*

3. **Exploding Gradients** — When weights grow uncontrollably.
   → *Use gradient clipping or normalization layers.*

---

7. Summary

- Activation functions introduce **non-linearity**, allowing neural networks to learn complex mappings.

- The **choice of activation** affects convergence speed, stability, and final accuracy.

- **ReLU and its variants** dominate modern architectures due to simplicity and efficiency.

- **Sigmoid and Softmax** remain essential in output layers for classification tasks.

Together, these functions form the backbone of Deep Learning — transforming raw inputs into intelligent representations.