Activations Functions :

What is an Activation Function?

An activation function in a neural network is a mathematical function applied to each neuron’s output (also called the neuron’s activation). It determines whether a neuron should be activated or not, introducing non-linearity into the model, which allows the network to learn complex patterns.

Why Do We Need Activation Functions?

1. **Introduce Non-linearity**:
    - Without non-linearity, a neural network, no matter how many layers it has, would behave like a single-layer linear model.
    - Non-linear activation functions allow the network to learn and represent complex patterns and functions.
2. **Control the Output**:
    - Activation functions can control the range of the output values (e.g., sigmoid outputs between 0 and 1).
    - This is particularly useful for tasks like classification where outputs need to represent probabilities.
3. **Gradient Flow**:
    - Proper activation functions ensure gradients flow properly during backpropagation.
    - This helps in updating weights effectively and avoiding issues like vanishing gradients.

Good Features in Activation Functions

1. **Non-linearity**:
    - Enables the network to solve non-trivial problems.
    - Example: ReLU introduces non-linearity by zeroing out negative values.
2. **Differentiability**:
    - Essential for backpropagation to compute gradients.

- Example: Sigmoid and Tanh functions are differentiable at all points.

3. **Computational Efficiency**:
    - Should not add significant computational overhead.
    - Example: ReLU is computationally efficient as it only involves a simple threshold operation.
4. **Avoiding Vanishing/Exploding Gradients**:
    - Ensures stable training by maintaining gradient magnitudes.
    - Example: ReLU helps mitigate the vanishing gradient problem.
5. **Zero-Centered Outputs**:
    - Helps in faster convergence by making the output range symmetric around zero.
    - Example: Tanh outputs range from -1 to 1.
6. **Sparse Activation**:
    - Only a subset of neurons should be active at a time, leading to efficient computation.
    - Example: ReLU only activates neurons with positive inputs.

Bad Features in Activation Functions

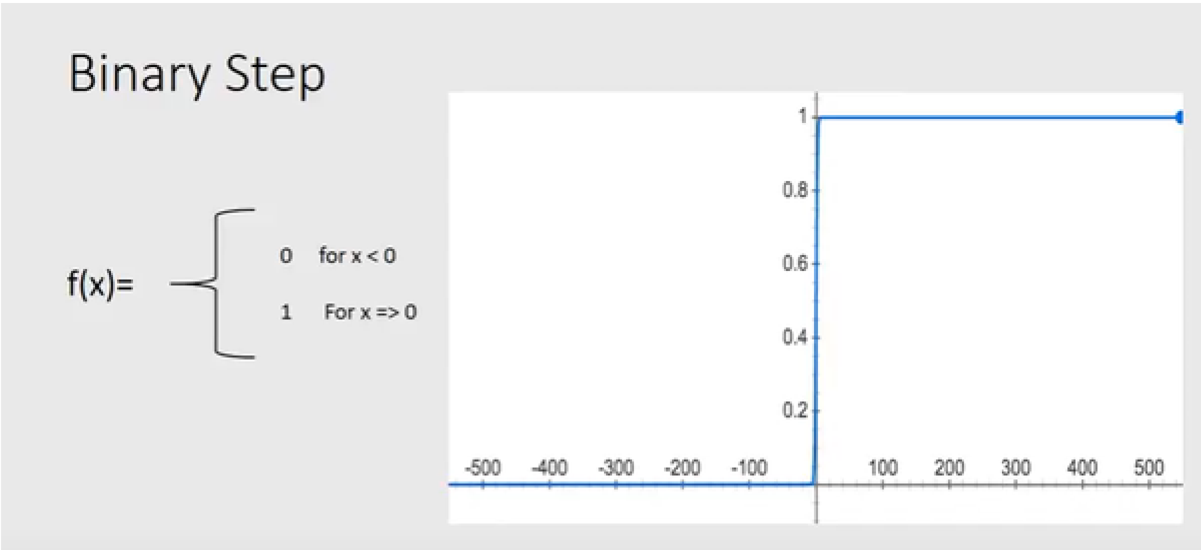
1. **Non-differentiability**:
    - Prevents the use of gradient-based optimization.
    - Example: Binary Step Function is not differentiable.
2. **Vanishing Gradients**:
    - Leads to very small gradients, making it hard for the network to learn.
    - Example: Sigmoid can cause vanishing gradients for large positive or negative inputs.
3. **Exploding Gradients**:
    - Causes gradients to become excessively large, leading to unstable training.
    - Example: Functions that lead to large outputs for large inputs can cause exploding gradients.

4. **Dead Neurons**:
    - Neurons that never activate and thus do not contribute to learning.
    - Example: ReLU can cause dead neurons when inputs are negative.
5. **Not Zero-Centered**:
    - Slows down convergence as gradients may consistently move in a certain direction.
    - Example: Sigmoid outputs are not zero-centered (range from 0 to 1).

Activation Functions in Neural Networks: Detailed Analysis

1. Binary Step Function

- **Formula**:



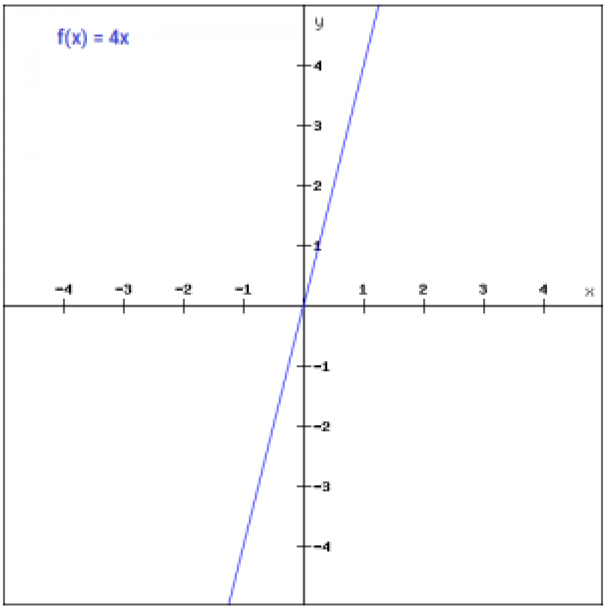
- **Range**: {0,1}

- **Usage**: Rarely used in practice due to its simplicity and limitations. Sometimes used in perceptrons for binary classification.
- **Advantages**: Simple implementation.
- **Disadvantages**:

- Not useful for multi-class classification.
- Gradient is zero everywhere, hindering backpropagation.
- Not differentiable, making it unsuitable for gradient-based optimization.

2. Linear Function

- **Formula**:
-

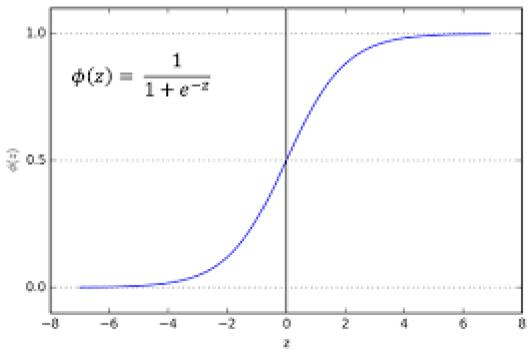


- **Range**: $(-\infty,\infty)$

- **Usage**: Generally used in simple regression tasks or in the final layer of a neural network for regression problems.
- **Advantages**:
  - Simple and interpretable.
  - No vanishing gradient problem.
- **Disadvantages**:
  - Gradient is constant and does not depend on x.
  - Poor at capturing complex patterns.
  - The network may not learn non-linear boundaries.

3. Sigmoid Activation Function

- **Formula**:



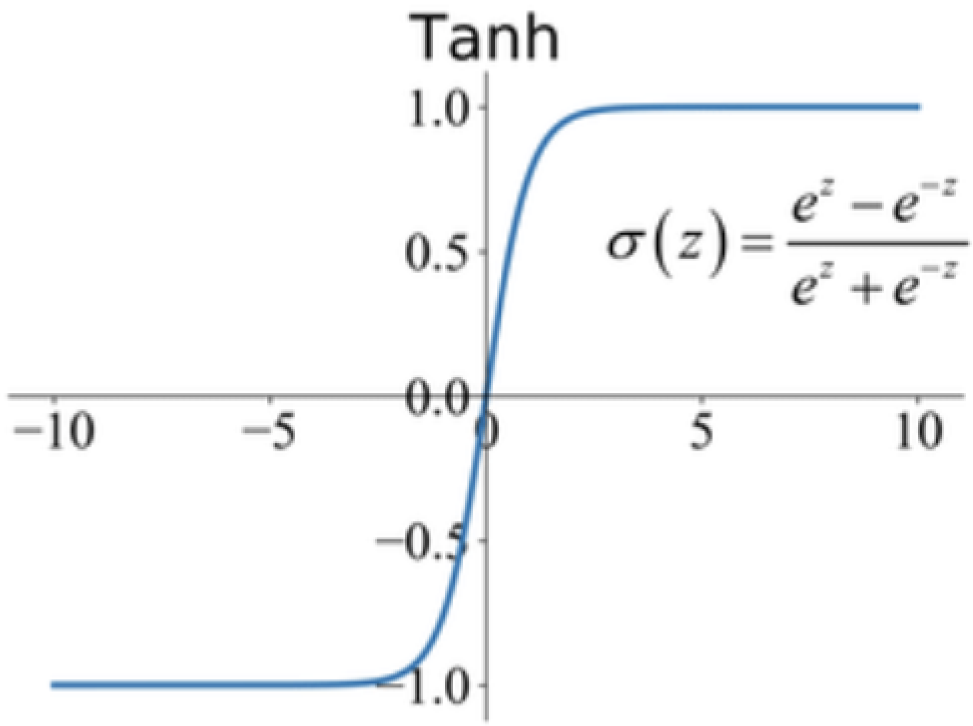
- **Range**: (0,1)

  (0,1)(0, 1)

- **Usage**: Commonly used in binary classification problems and in the output layer of binary classifiers.
- **Advantages**:
  - Smooth and differentiable.
  - Output can be interpreted as probabilities.
- **Disadvantages**:
  - Vanishing gradient problem for large positive or negative values.
  - Output is not zero-centered, which can slow down convergence.

4. Tanh (Hyperbolic Tangent) Function

- **Formula**:

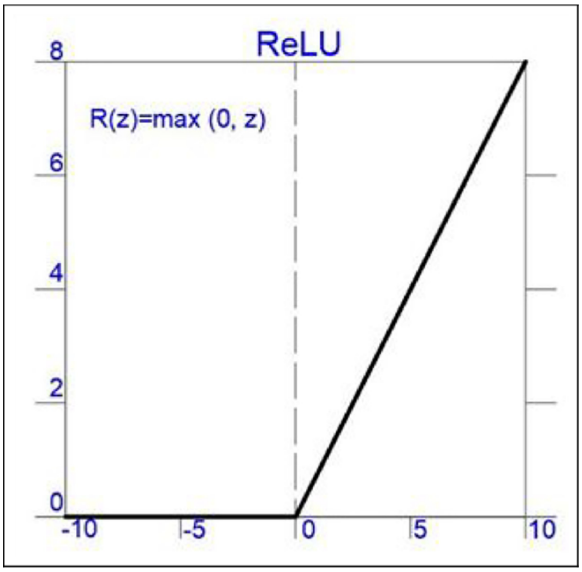


- **Range**: (−1,1)

  (−1,1)(-1, 1)
- **Usage**: Often preferred over sigmoid in hidden layers due to its zero-centered output.
- **Advantages**:
  - Zero-centered output.
  - Smooth and differentiable.
- **Disadvantages**:
  - Vanishing gradient problem similar to sigmoid.
  - Computationally more expensive than ReLU.

5. ReLU (Rectified Linear Unit)

- **Formula**:



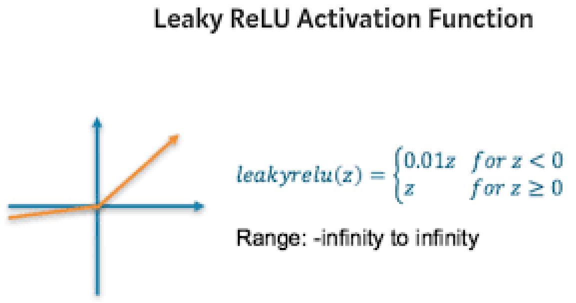
- **Range**: [0,∞)

  [0,∞)[0, \infty)
- **Usage**: Widely used in hidden layers of neural networks.
- **Advantages**:
  - Computationally efficient.
  - Helps mitigate the vanishing gradient problem.
  - Sparse activation (neurons are activated only when necessary).
- **Disadvantages**:
  - Can create "dead neurons" (neurons that never activate if x<0 consistently).

    x<0x < 0

6. Leaky ReLU

- **Formula**:
-



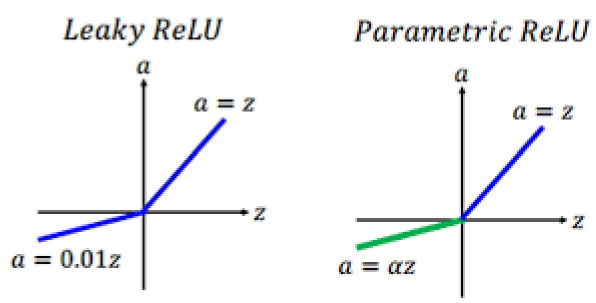
- **Range**: (−∞,∞)

  (−∞,∞)
- **Usage**: Used to address the dead neuron problem in ReLU.
- **Advantages**:
  - Mitigates the dead neuron problem by allowing a small gradient when x<0.

    x<0x < 0
- **Disadvantages**:
  - The choice of the leakage factor (0.01) may not be optimal for all tasks.

7. Parametric ReLU

- **Formula**:



- **Range**: (−∞,∞)

  (−∞,∞)
- **Usage**: Used to address dead neurons with a learnable parameter a.

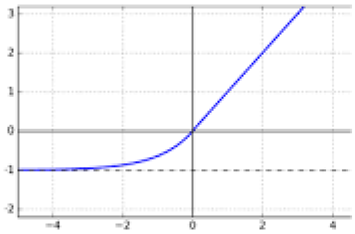
  aa

- **Advantages**:
    - Allows the network to learn the optimal value of a for better performance.

      aa
- **Disadvantages**:
    - More complex than ReLU and Leaky ReLU.

8. Exponential Linear Unit (ELU)

- **Formula**:



$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$

- **Range**: (−∞,∞) for x<0 and [0,∞) for x≥0

  (−∞,∞)

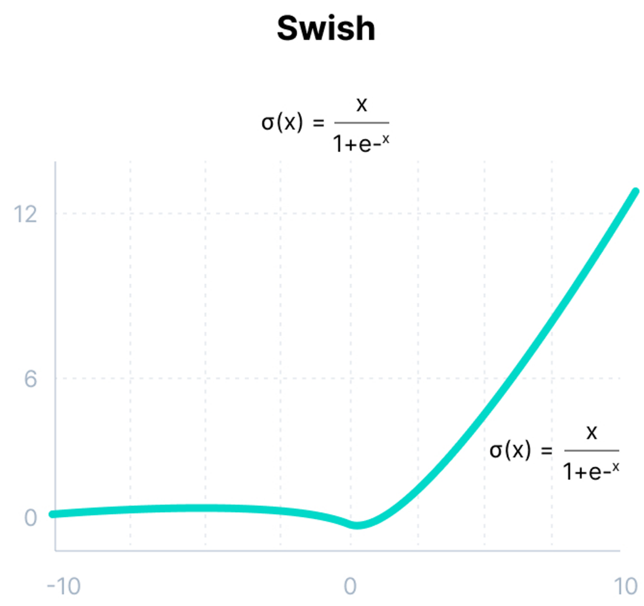
  x<0x < 0

- **Usage**: Used to avoid dead neurons and to have smooth gradients for negative inputs.
- **Advantages**:
    - Helps mitigate the vanishing gradient problem.
    - The negative saturation (when x<0) helps regularize the network.

      x<0x < 0
- **Disadvantages**:
    - More computationally intensive due to the exponential function.

9. Swish

- **Formula**:



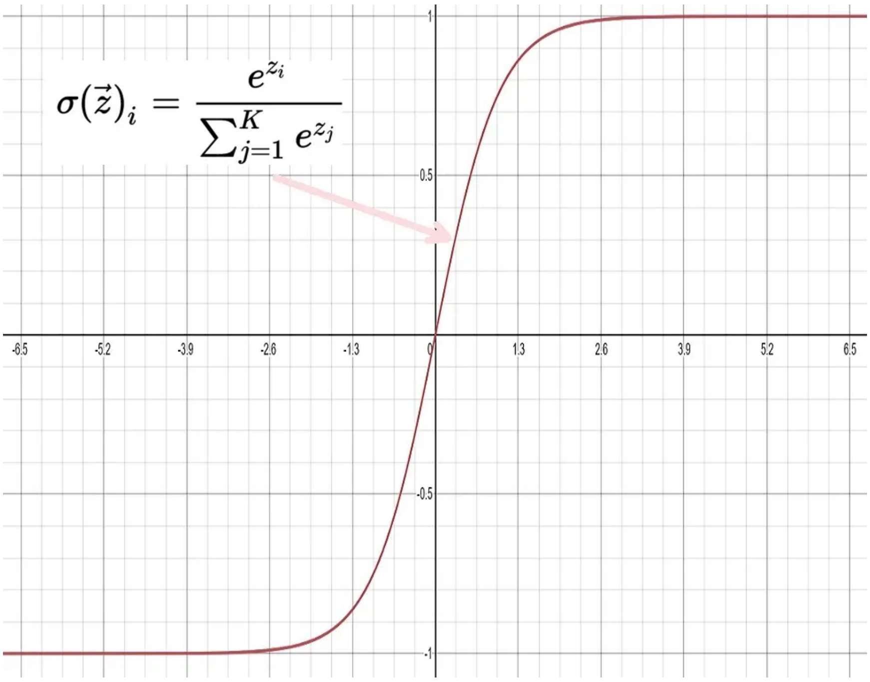
- **Range**: $(-\infty,\infty)$

  $(-\infty,\infty$

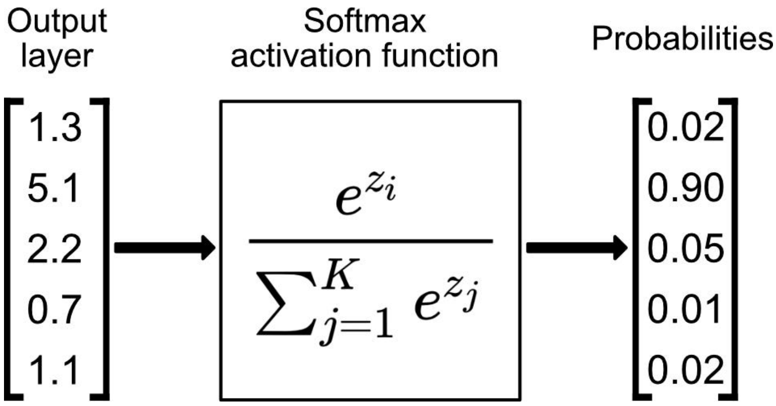
- **Usage**: Used in deeper neural networks and has shown to outperform ReLU in certain tasks.
- **Advantages**:
  - Smooth and differentiable.
  - No dead neuron problem.
  - Better performance in deep networks.
- **Disadvantages**:
  - More computationally intensive than ReLU.
  - Non-monotonic, which can complicate understanding of the activation's effect.

10. Softmax

- **Formula**:



-



-
- **Range**: [0,1] for each output neuron, and the sum of all outputs is 1.

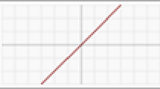
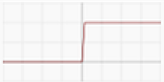
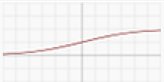
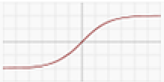
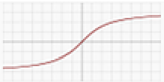
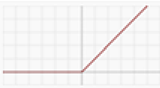
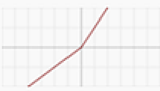
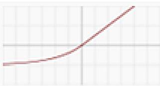
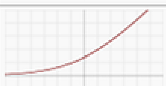
  [0,1][0, 1]
- **Usage**: Used in the output layer for multi-class classification problems.
- **Advantages**:
  - Converts logits to probabilities, making them interpretable.
  - Ensures that outputs sum to 1, representing a valid probability distribution.
- **Disadvantages**:

- Computationally intensive due to exponentiation and normalization.
- Can suffer from the vanishing gradient problem if logits are large.

Summary of Usage and Recommendations

- **Binary Step Function**: Simple tasks with binary outputs, though rarely used in practice due to its limitations.
- **Linear Function**: Regression problems, particularly in the output layer.
- **Sigmoid Function**: Binary classification, particularly in the output layer.
- **Tanh Function**: Preferred over sigmoid for hidden layers due to zero-centered output.
- **ReLU Function**: Default choice for hidden layers in most networks.
- **Leaky ReLU**: Used when encountering dead neurons with ReLU.
- **Parametric ReLU**: Advanced variant of Leaky ReLU when even finer control over negative slopes is needed.
- **ELU**: Used when the benefits of ReLU are desired, but with smoother negative saturation to avoid dead neurons.
- **Swish**: Used in deeper networks for better performance, though computationally more demanding.
- **Softmax**: Used in the output layer for multi-class classification to convert logits to probabilities.

- **Binary Classification**: Sigmoid (output layer), Tanh/ReLU (hidden layers).
- **Multi-class Classification**: Softmax (output layer), ReLU (hidden layers).
- **Regression**: Linear (output layer), ReLU (hidden layers).
- **General Use**: ReLU for hidden layers due to efficiency and effectiveness.

Name	Plot	Equation	Derivative
Identity		$f(x) = x$	$f'(x) = 1$
Binary step		$f(x) = \begin{cases} 0 & \text{for} \quad x < 0 \\ 1 & \text{for} \quad x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for} \quad x \neq 0 \\ ? & \text{for} \quad x = 0 \end{cases}$
Logistic (a.k.a Soft step)		$f(x) = \frac{1}{1 + e^{-x}}$	$f'(x) = f(x)(1 - f(x))$
TanH		$f(x) = \tanh(x) = \frac{2}{1 + e^{-2x}} - 1$	$f'(x) = 1 - f(x)^2$
ArcTan		$f(x) = \tan^{-1}(x)$	$f'(x) = \frac{1}{x^2 + 1}$
Rectified Linear Unit (ReLU)		$f(x) = \begin{cases} 0 & \text{for} \quad x < 0 \\ x & \text{for} \quad x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for} \quad x < 0 \\ 1 & \text{for} \quad x \geq 0 \end{cases}$
Parameteric Rectified Linear Unit (PReLU)[2]		$f(x) = \begin{cases} \alpha x & \text{for} \quad x < 0 \\ x & \text{for} \quad x \geq 0 \end{cases}$	$f'(x) = \begin{cases} \alpha & \text{for} \quad x < 0 \\ 1 & \text{for} \quad x \geq 0 \end{cases}$
Exponential Linear Unit (ELU)[3]		$f(x) = \begin{cases} \alpha(e^x - 1) & \text{for} \quad x < 0 \\ x & \text{for} \quad x \geq 0 \end{cases}$	$f'(x) = \begin{cases} f(x) + \alpha & \text{for} \quad x < 0 \\ 1 & \text{for} \quad x \geq 0 \end{cases}$
SoftPlus		$f(x) = \log_e(1 + e^x)$	$f'(x) = \frac{1}{1 + e^{-x}}$