Here is an explanation of the common activation functions used in neural networks along with their **pros** and **cons**:

1. **Step Function**

The **Step Activation Function** is a simple threshold function that outputs a 0 or 1 based on whether the input is below or above a certain threshold.

Mathematical Expression:

$$f(x) = \begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x \geq 0 \end{cases}$$

Pros:

- **Simple:** The function is easy to understand and implement.
- **Binary Output:** Useful in binary classification problems (though it's not used much in modern deep learning).

Cons:

- **Non-differentiable:** The step function is not differentiable, making it unsuitable for gradient-based optimization methods like backpropagation.
- **Vanishing Gradient:** Because of the hard threshold, the gradient is zero everywhere except at the threshold, which makes learning inefficient (stagnation).
- **Lack of Learning Ability:** Cannot adjust weights efficiently during training due to its flat regions.

---

2. **Sigmoid Function**

The **Sigmoid Activation Function** squashes its input to a value between 0 and 1, which is often used in binary classification.

Mathematical Expression:

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

Pros:

- **Output Range (0, 1):** Useful for binary classification (outputs a probability-like value).
- **Differentiable:** The sigmoid function is smooth and differentiable, which helps in gradient-based optimization.

Cons:

- **Vanishing Gradient:** For very high or very low values of input, the gradient becomes very small, causing the network to stop learning effectively (problem with deep networks).
- **Not Zero-Centered:** Outputs only positive values (0 to 1), which can lead to inefficient gradient descent since the gradients always have the same sign.
- **Slow Convergence:** Due to the vanishing gradient problem, training can be slow in deeper networks.

---

3. **tanh (Hyperbolic Tangent)**

The **tanh Activation Function** is similar to the sigmoid but outputs values between -1 and 1.

Mathematical Expression:

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

Pros:

- **Zero-Centered:** The output range of **-1 to 1** helps with the learning process because the data is zero-centered.
- **Differentiable:** Like sigmoid, it's differentiable and works well for gradient-based optimization.
- **Better Gradient Flow:** Compared to sigmoid, the gradient is less likely to vanish because the output is centered around zero.

Cons:

- **Vanishing Gradient:** While tanh improves upon sigmoid by being zero-centered, it still suffers from vanishing gradients for very large positive or negative inputs.
- **Computational Cost:** The exponential operations can be computationally expensive.

---

4. **ReLU (Rectified Linear Unit)**

The **ReLU Activation Function** is one of the most commonly used activation functions, particularly in deep neural networks. It outputs the input directly if it is positive, otherwise, it outputs zero.

Mathematical Expression:

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

Pros:

- **Efficient Computation:** Simple and computationally efficient, with no exponential operations.
- **Solves Vanishing Gradient Problem:** It doesn't saturate for positive inputs, helping the model to learn faster and preventing vanishing gradients in the positive region.
- **Sparse Activation:** Most neurons are inactive (output zero), leading to a sparse representation, which can make the model easier to train and less prone to overfitting.

Cons:

- **Dying ReLU Problem:** If a neuron's weights are initialized poorly or it starts outputting zero for all inputs (for instance, for large negative values), it might never recover and will stop learning. This is known as the "Dying ReLU" problem.
- **Non-zero-centered:** Like the sigmoid, it can lead to poor convergence during training in some cases since the gradients can always have the same sign.

---

5. **Leaky ReLU**

Leaky ReLU is a modified version of ReLU that allows a small, non-zero gradient when the input is negative, thus preventing the "dying ReLU" problem.

Mathematical Expression:

$$f(x) = \{\begin{matrix} x & \text{if } x > 0 \\ \alpha x & \text{if } x \leq 0 \end{matrix}$$

where **α** (alpha) is a small constant (usually a small value like 0.01).

Pros:

- **Prevents Dying ReLU:** By allowing a small slope for negative inputs, Leaky ReLU reduces the chances of neurons "dying" and getting stuck with zero gradients.
- **Efficient:** Like ReLU, Leaky ReLU is computationally efficient.
- **Zero-Centered (in a way):** While not completely zero-centered, the small slope for negative values can help improve learning.

Cons:

- **Still Not Perfect:** While it addresses the Dying ReLU problem, it still has some issues with gradients for very negative values, though much less than ReLU.
- **Alpha Hyperparameter:** The value of **α** needs to be chosen carefully; if too large, it can cause instability in the training.

Summary Table

Activation Function	Pros	Cons
Step	Simple, Binary output	Non-differentiable, Vanishing gradient, Poor learning
Sigmoid	Output between 0 and 1, Differentiable	Vanishing gradient, Not zero-centered, Slow convergence
Tanh	Zero-centered, Better gradient flow than sigmoid	Vanishing gradient for large inputs, Computationally expensive
ReLU	Computationally efficient, No vanishing gradient for positive inputs, Sparse activation	Dying ReLU problem, Non-zero-centered
Leaky ReLU	Prevents Dying ReLU, Efficient, Somewhat zero-centered	Alpha hyperparameter needs careful selection, Not perfect solution

---

When to Use Which Activation Function?

- **Step:** Rarely used in modern deep learning; more useful in simple binary classification tasks with a decision boundary.
- **Sigmoid:** Best for binary classification output (probability prediction), though ReLU and variants are preferred for hidden layers in deep networks.
- **Tanh:** Often used in tasks where the output should be between -1 and 1, such as for regression or in specific types of RNNs.
- **ReLU:** Widely used for hidden layers, particularly in deep neural networks.
- **Leaky ReLU:** Preferred over ReLU for preventing the "dying neuron" issue, particularly in deep networks.