✅ **Graphs Generated!**  
The activation function plots for **Sigmoid**, **Tanh**, **ReLU**, **Leaky ReLU**, and **ELU** have been generated above.

**🎯 Now Let’s Explain Each Activation Function in Detail**

**1️⃣ Sigmoid Activation Function**

* **Formula:** (\sigma(x) = \frac{1}{1 + e^{-x}})
* **Range:** **(0, 1)**
* **Use Case:** Best suited for **binary classification** tasks (e.g., last layer of Logistic Regression, last neuron in binary classification neural networks).
* **Pros:** Smooth gradient, interpretable as probability.
* **Cons:** Causes **vanishing gradient problem**, not zero-centered.

**2️⃣ Tanh Activation Function**

* **Formula:** (\tanh(x) = \frac{2}{1 + e^{-2x}} - 1)
* **Range:** **(-1, 1)**
* **Use Case:** Hidden layers in shallow networks where normalized outputs are beneficial.
* **Pros:** Zero-centered, stronger gradient than Sigmoid.
* **Cons:** Still prone to vanishing gradient for large values.

**3️⃣ ReLU (Rectified Linear Unit)**

* **Formula:** (f(x) = \max(0, x))
* **Range:** **[0, ∞)**
* **Use Case:** **Default choice** for hidden layers in deep neural networks, CNNs, etc.
* **Pros:** Computationally efficient, **solves vanishing gradient problem**.
* **Cons:** **Dead ReLU problem** (neurons can die and never activate again).

**4️⃣ Leaky ReLU**

* **Formula:**  
  [  
  f(x) =  
  \begin{cases}  
  x & \text{if } x > 0 \  
  \alpha x & \text{if } x \leq 0  
  \end{cases}  
  ]  
  With (\alpha \approx 0.01)
* **Range:** **(-∞, ∞)**
* **Use Case:** Deep networks to prevent dead neurons.
* **Pros:** Fixes the **dead ReLU** issue.
* **Cons:** Slight computational overhead compared to ReLU.

**5️⃣ ELU (Exponential Linear Unit)**

* **Formula:**  
  [  
  f(x) =  
  \begin{cases}  
  x & \text{if } x > 0 \  
  \alpha(e^x - 1) & \text{if } x \leq 0  
  \end{cases}  
  ]
* **Range:** **(-α, ∞)** (default α = 1 → **(-1, ∞)**)
* **Use Case:** Stable deep models, especially where **mean activations near 0** help faster learning.
* **Pros:** Smooth negative values, better convergence than ReLU.
* **Cons:** Slightly slower due to exponent computation.

**🔥 Softmax Activation Function (For Multi-Class Classification)**

* **Formula:**  
  [  
  \text{softmax}(x\_i) = \frac{e^{x\_i}}{\sum\_{j=1}^{n} e^{x\_j}}  
  ]
* **Range:** **(0, 1)** but sums to **1 across all classes**
* **Use Case:** **Last layer** in multi-class classification networks (e.g., image recognition with >2 classes).
* **Note:** It is not visualized easily as a 1D curve because it takes a **vector input**, not scalar.

**🧠 Quick Comparison Table**

| **Activation** | **Range** | **Best For** | **Vanishing Gradient?** | **Notes** |
| --- | --- | --- | --- | --- |
| **Sigmoid** | (0, 1) | Binary classification output | Yes | Probabilistic output |
| **Tanh** | (-1, 1) | Hidden layers (shallow nets) | Yes | Zero-centered |
| **ReLU** | [0, ∞) | Deep CNNs/ANNs | No (mostly) | Fast but dead neurons |
| **Leaky ReLU** | (-∞, ∞) | Deep nets to fix ReLU | No | Negative small slope |
| **ELU** | (-α, ∞) | Deep nets with faster learning | No | Smooth negative region |
| **Softmax** | (0,1), sum=1 | Last layer (multi-class) | N/A | Probability distribution |

**🎓 Recommendation Summary**

| **Task Type** | **Recommended Activation** |
| --- | --- |
| Hidden layers (deep learning) | **ReLU** / **Leaky ReLU** / **ELU** |
| Binary classification (output) | **Sigmoid** |
| Multi-class classification (output) | **Softmax** |
| RNNs / LSTM internal gates | **Tanh** + **Sigmoid** combos |

Would you like me to:

* ✅ Add **Softmax 2D visualization**?
* ✅ Provide **math derivation and gradient formulas**?
* ✅ Export all plots and explanation as a **PDF / PPT for study notes**?

Just tell me **"Make PDF"** or **"Add softmax graph"**, and I’ll do it!