**1. Common Activation Functions and Their Uses**

| **Activation Function** | **Description** | **Use Cases** | **Pros** | **Cons** |
| --- | --- | --- | --- | --- |
| **ReLU (Rectified Linear Unit)** | Outputs 0 for negatives, x for positives. | Most hidden layers in deep networks. | Efficient, prevents vanishing gradients. | Can cause "dying neurons" (outputs stuck at 0). |
| **Leaky ReLU** | Allows small gradients for negatives. | GANs, deep networks prone to dying neurons. | Prevents dying neurons, better gradient flow. | Introduces a tunable hyperparameter (alpha). |
| **Sigmoid** | Outputs values between 0 and 1. | Binary classification, final layer for probabilities. | Smooth gradient, interpretable output. | Vanishing gradient problem, saturates for large inputs. |
| **Tanh (Hyperbolic Tangent)** | Outputs values between -1 and 1. | Regression, when outputs can be negative. | Centered around 0, better for gradient flow than sigmoid. | Similar vanishing gradient issues as sigmoid. |
| **Softmax** | Converts logits to probabilities summing to 1. | Multi-class classification (final layer). | Probabilistic interpretation. | Computationally expensive for many classes. |
| **ELU (Exponential Linear Unit)** | Similar to ReLU but smoother for negatives. | Deep networks needing faster convergence. | Avoids dying neurons, smoother gradient flow. | Slightly more expensive computationally. |
| **GELU (Gaussian Error Linear Unit)** | Scaled sigmoid-like activation. | Transformers, NLP, cutting-edge architectures. | Smooth, robust gradients. | Relatively newer, less studied for all tasks. |
| **Linear** | Outputs the input directly. | Final layers in regression tasks. | Direct output, no distortion. | No non-linearity, not suitable for hidden layers. |

**2. General Guidelines for Choosing Activation Functions**

1. **For Hidden Layers:**
   * **ReLU** is often the default choice because of its simplicity and efficiency.
   * Use **LeakyReLU** or **ELU** if ReLU causes "dying neurons."
   * Consider **GELU** for modern architectures like Transformers.
2. **For Output Layers:**
   * **Sigmoid:** Binary classification (e.g., spam detection).
   * **Softmax:** Multi-class classification (e.g., digit classification).
   * **Linear:** Regression tasks (e.g., predicting continuous values like house prices).
3. **For GANs:**
   * **Generator Hidden Layers:** Use **LeakyReLU** for better gradient flow.
   * **Output Layer:** Use **Tanh** if your output needs to scale between -1 and 1.
4. **For NLP and Transformers:**
   * Use **GELU** or **ReLU** for better gradient smoothness and performance.
5. **Task-Specific Needs:**
   * Consider the range of your outputs:
     + For normalized outputs (0 to 1): Use **Sigmoid** or **Softmax.**
     + For outputs centered around 0: Use **Tanh.**

**3. Practical Considerations**

* **Avoid Vanishing Gradient Problems:**
  + Use ReLU variants (ReLU, LeakyReLU, ELU) for deeper networks.
  + Avoid sigmoid and tanh in hidden layers of very deep networks.
* **Test and Experiment:**
  + There’s no one-size-fits-all activation function. Experiment with a few options if you're unsure.
* **Consider Computational Cost:**
  + Some functions (like Softmax or GELU) are computationally heavier. Evaluate the trade-offs for your application.

**4. When to Change Activation Functions**

* If the model is **not learning**:  
  Replace Sigmoid/Tanh with ReLU or LeakyReLU to combat vanishing gradients.
* If the model is **overfitting**:  
  Combine with dropout or try smoother activations like ELU.
* If the model's **predictions are biased**:  
  Ensure the activation matches the output range (e.g., using Sigmoid for probabilities).

1. Small Batch Sizes?

* Use **Layer Normalization** or **Group Normalization**.

1. Image Generation Tasks?

* Use **Instance Normalization**.

1. General Convolutional Networks?

* Try **Group Normalization** or BatchNorm alternatives.

1. Architectural Simplicity?

* Use **Weight Normalization** or skip normalization.