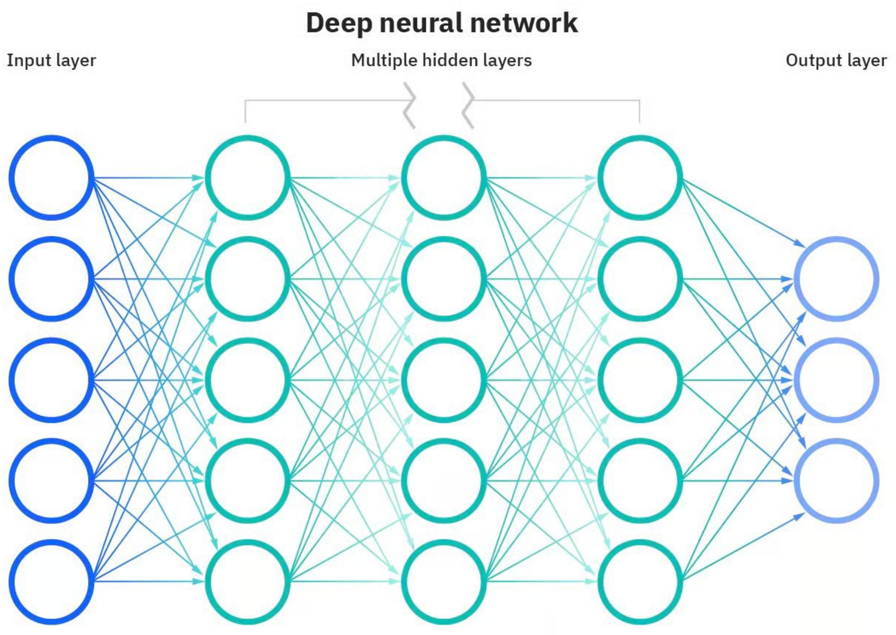
Introduction to Activation Functions in Neural Networks

Gain an understanding of common activation functions, ranging from the robustness of ReLU to the probabilistic nature of Softmax.

Updated **Jan 16, 2025** · 11 min read

Author: Bhuvnesh Verma

Systems, Not Hype



Contents

1 What Are Activation Functions? **2**

2 Why Are Activation Functions Essential? **2**

3 Types of Activation Functions **2**
- 3.1 Linear Activation 3
- 3.2 Sigmoid Activation 4
- 3.3 Tanh (Hyperbolic Tangent) Activation 5
- 3.4 ReLU (Rectified Linear Unit) Activation 6
- 3.5 Softmax Activation 7

4 Choosing the Right Activation Function **9**
- 4.1 For binary classification 9
- 4.2 For multi-class classification 9
- 4.3 If unsure 9

5 Conclusion **9**

6 FAQs **9**

1 What Are Activation Functions?

Activation functions are an integral building block of neural networks that enable them to learn complex patterns in data. They transform the input signal of a node in a neural network into an output signal that is then passed on to the next layer. Without activation functions, neural networks would be restricted to modeling only linear relationships between inputs and outputs.

Activation functions introduce non-linearities, allowing neural networks to learn highly complex mappings between inputs and outputs.

Choosing the right activation function is crucial for training neural networks that generalize well and provide accurate predictions. In this docs, we will provide an overview of the most common activation functions, their roles, and how to select suitable activation functions for different use cases.

Whether you are just starting out in deep learning or are a seasoned practitioner, understanding activation functions in depth will build your intuition and improve your application of neural networks.

2 Why Are Activation Functions Essential?

Without activation functions, neural networks would just consist of linear operations like matrix multiplication. All layers would perform linear transformations of the input, and no non-linearities would be introduced.

Most real-world data is non-linear. For example, relationships between house prices and size, income, and purchases, etc., are non-linear. If neural networks had no activation functions, they would fail to learn the complex non-linear patterns that exist in real-world data.

Activation functions enable neural networks to learn these non-linear relationships by introducing non-linear behaviors through activation functions. This greatly increases the flexibility and power of neural networks to model complex and nuanced data.

3 Types of Activation Functions

Neural networks leverage various types of activation functions to introduce non-linearities and enable learning complex patterns. Each activation function has its own unique properties and is suitable for certain use cases.

For example, the sigmoid function is ideal for binary classification problems, softmax is useful for multi-class prediction, and ReLU helps overcome the vanishing gradient problem.

Using the right activation function for the task leads to faster training and better performance.

Let's look at some of the common activation functions:

3.1 Linear Activation

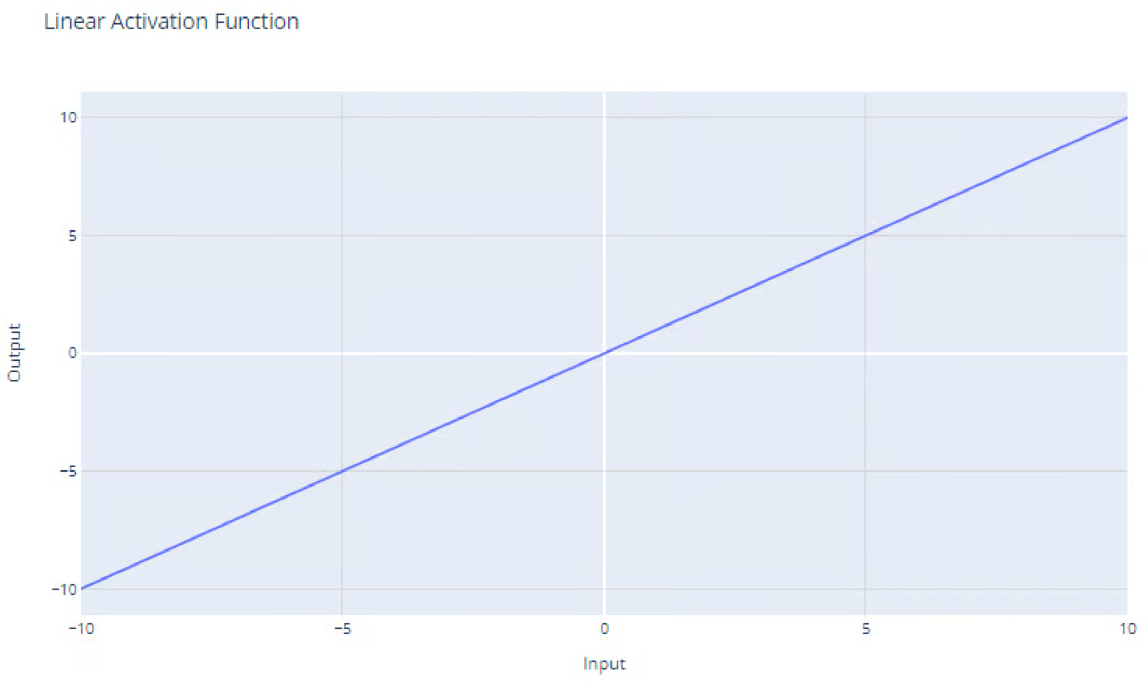


Figure 1: Linear activation function.

The linear activation function is the simplest activation function, defined as:

$$f(x) = x$$

It simply returns the input x as the output. Graphically, it looks like a straight line with a slope of 1.

The main use case of the linear activation function is in the output layer of a neural network used for regression. For regression problems where we want to predict a numerical value, using a linear activation function in the output layer ensures the neural network outputs a numerical value. The linear activation function does not squash or transform the output, so the actual predicted value is returned.

However, the linear activation function is rarely used in hidden layers of neural networks. This is because it does not provide any non-linearity. The whole point of hidden layers is to learn non-linear combinations of the input features. Using a linear activation throughout would restrict the model to just learning linear transformations of the input.

3.2 Sigmoid Activation

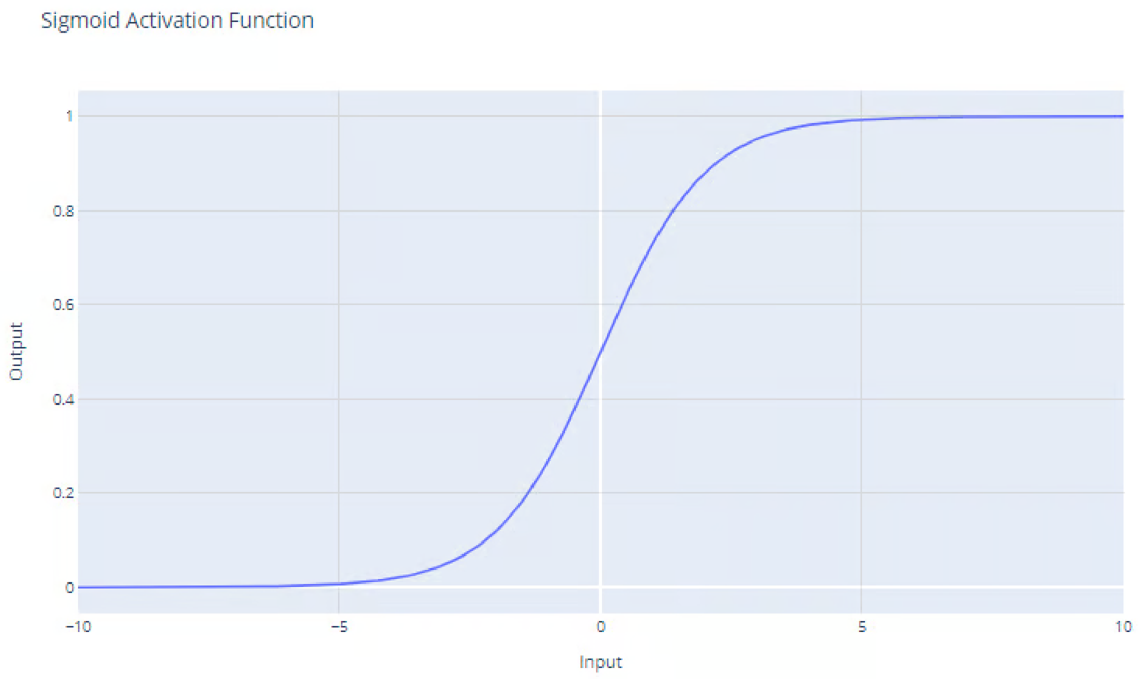


Figure 2: Sigmoid activation function.

The sigmoid activation function, often represented as $\sigma(x)$, is a smooth, continuously differentiable function that is historically important in the development of neural networks. The sigmoid activation function has the mathematical form:

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

takes a real-valued input and squashes it to a value between 0 and 1. The sigmoid function has an "S"-shaped curve that asymptotes to 0 for large negative numbers and 1 for large positive numbers. The outputs can be easily interpreted as probabilities, which makes it natural for binary classification problems.

Sigmoid units were popular in early neural networks since the gradient is strongest when the unit's output is near 0.5, allowing efficient backpropagation training. However, sigmoid units suffer from the "vanishing gradient" problem that hampers learning in deep neural networks.

As the input values become significantly positive or negative, the function saturates at 0 or 1, with an extremely flat slope. In these regions, the gradient is very close to zero. This results in very small changes in the weights during backpropagation, particularly for neurons in the earlier layers of deep networks, which makes learning painfully slow or even halts it. This is referred to as the vanishing gradient problem in neural networks.

The main use case of the sigmoid function is as the activation for the **output layer of binary classification models**. It squashes the output to a probability value between 0 and 1, which can be interpreted as the probability of the input belonging to a particular class.Use in the final layer when *predicting a probability for two classes* (e.g., Is it a cat or not?). Avoid in hidden layers of deep networks.

3.3 Tanh (Hyperbolic Tangent) Activation

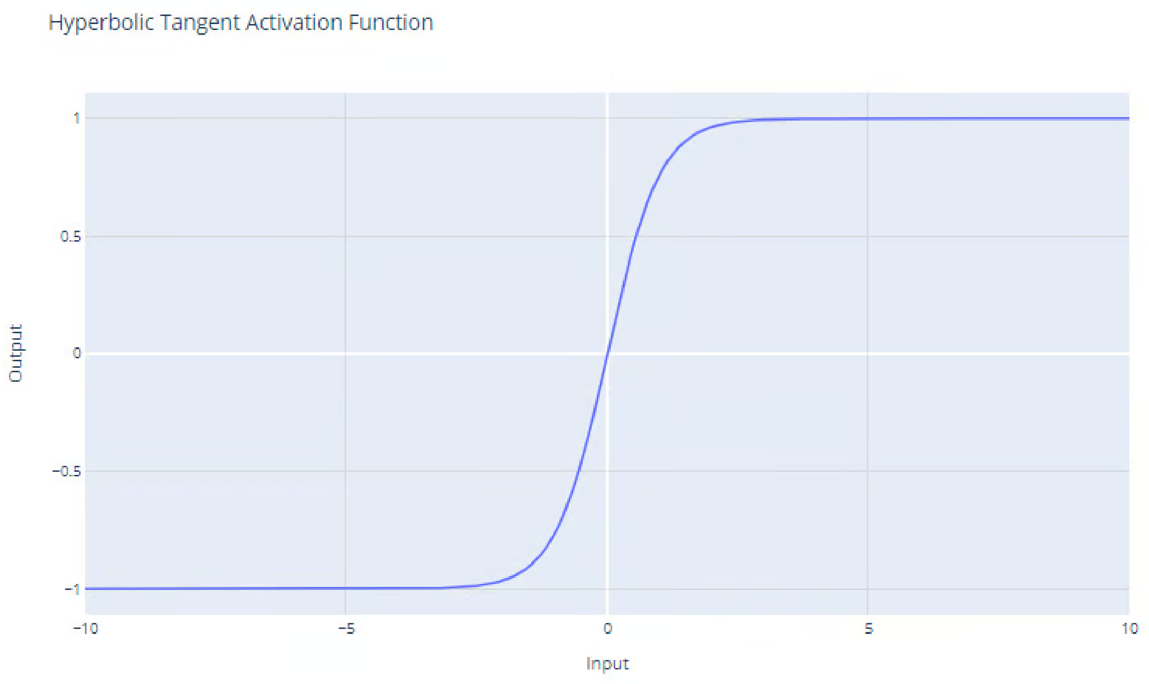


Figure 3: Tanh activation function

The tanh (hyperbolic tangent) activation function is defined as:

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

The tanh function outputs values in the range of -1 to +1. This means that it can deal with negative values more effectively than the sigmoid function, which has a range of 0 to 1.

Unlike the sigmoid function, *tanh is zero-centered*, which means that its output is symmetric around the origin of the coordinate system. This is often considered an advantage because it can help the *learning algorithm converge faster*.

Because the output of tanh ranges between -1 and +1, it has stronger gradients than the sigmoid function. Stronger gradients often result in faster learning and convergence during training because they tend to be more resilient against the problem of vanishing gradients when compared to the gradients of the sigmoid function.

Despite these advantages, the tanh function still suffers from the ”vanishing gradient problem”. During backpropagation, the gradients of the tanh function can become very small (close to zero). This issue is particularly problematic for deep networks with many layers; the gradients of the loss function may become too small to make significant changes in the weights during training as they propagate back to the initial layers. This can drastically slow down the training process and can lead to poor convergence properties.

Tanh is generally preferred over Sigmoid for hidden layers because it is zero-centered. When data is normalized to a mean of zero, Tanh allows for more efficient training and faster convergence.

While Tanh is the better default choice, the final decision should always be validated through experimentation based on your specific model’s performance.

3.4 ReLU (Rectified Linear Unit) Activation

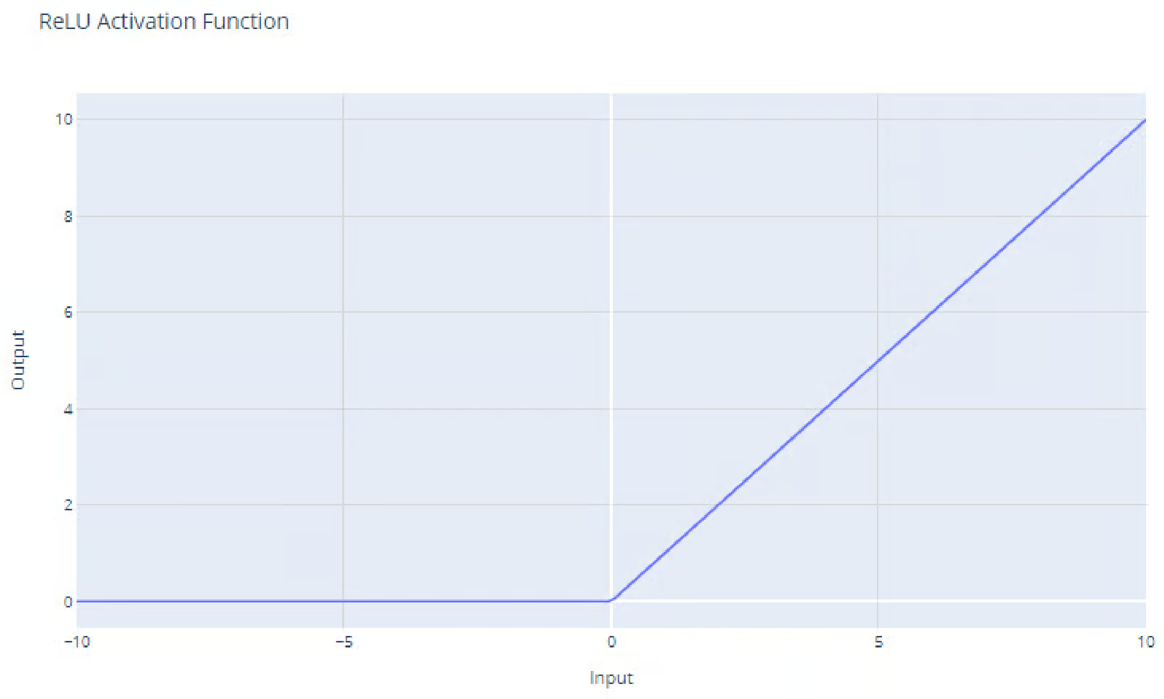


Figure 4: ReLU activation function

The Rectified Linear Unit (ReLU) activation function has the form:

$$\text{ReLU}(x) = \max(0, x)$$

It thresholds the input at zero, returning 0 for negative values and the input itself for positive values.

For inputs greater than 0, ReLU acts as a linear function with a gradient of 1. This means that it does not alter the scale of positive inputs and allows the gradient to pass through unchanged during backpropagation. *This property is critical in mitigating the vanishing gradient problem.*

Even though ReLU is linear for half of its input space, it is technically a non-linear function because it has a non-differentiable point at $x = 0$, where it abruptly changes from x. This non-linearity allows neural networks to learn complex patterns.

Since ReLU outputs zero for all negative inputs, it naturally leads to sparse activations; at any time, only a subset of neurons are activated, leading to more efficient computation.

The ReLU function is computationally inexpensive because it involves simple thresholding at zero. This allows networks to scale to many layers without a significant increase in computational burden, compared to more complex functions like tanh or sigmoid.

3.5 Softmax Activation

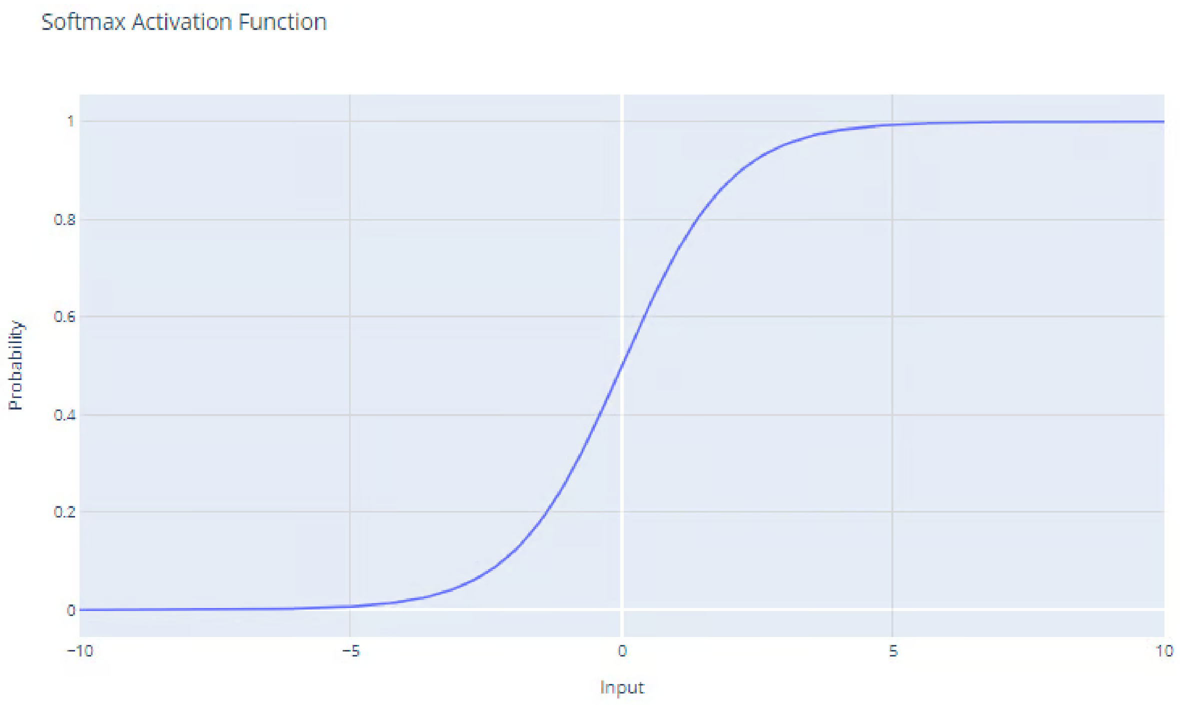


Figure 5: Softmax activation function

The softmax activation function, also known as the **normalized exponential function**, is particularly useful within the context of multi-class classification problems. This function operates on a vector, often referred to as the logits, which represents the raw predictions or scores for each class computed by the previous layers of a neural network.

For input vector x with elements $x_1, x_2, \ldots, x_C$, the softmax function is defined as:

$$\text{softmax}(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{C} e^{x_j}}$$

The output of the softmax function is a probability distribution that sums up to one. Each element of the output represents the probability that the input belongs to a particular class.

The use of the exponential function ensures that all output values are non-negative. This is crucial because probabilities cannot be negative.

Softmax amplifies differences in the input vector. Even small differences in the input values can lead to substantial differences in the output probabilities, with the highest input value(s) tending to dominate in the resulting probability distribution.

Softmax is typically used in the output layer of a neural network when the task involves classifying an input into one of several (more than two) possible categories (multi-class classification).

The probabilities produced by the softmax function can be interpreted as confidence scores for each class, providing insight into the model's certainty about its predictions.

Because softmax amplifies differences, it can be sensitive to outliers or extreme values. For example, if the input vector has a very large value, softmax can "squash" the probabilities of other classes, leading to an overconfident model.

Name	**Equation**	**Range**	**Description**	**Best Practice Use**
Sigmoid	$\sigma(x) = \frac{1}{1+e^{-x}}$	$(0, 1)$	S-shaped curve. Suffers from vanishing gradients in deep networks.	**Output Layer:** Binary classification. Avoid in hidden layers.
Tanh	$\tanh(x) = \frac{2}{1+e^{-2x}} - 1$	$(-1, 1)$	Zero-centered version of Sigmoid. Faster training than Sigmoid.	**Hidden Layers:** Legacy/RNNs. Largely replaced by ReLU.
ReLU	$f(x) = \max(0, x)$	$[0, \infty)$	Computationally fast; avoids vanishing gradients for positive inputs.	**Hidden Layers:** Default choice for most CNNs and FNNs.
Leaky ReLU	$f(x) = \begin{cases} x & \text{if } x \geq 0 \\ \alpha x & \text{otherwise} \end{cases}$	$(-\infty, \infty)$	Prevents "Dying ReLU" by allowing a small negative gradient.	**Hidden Layers:** Use if ReLU neurons are "dying" during training.
Softmax	$\frac{e^{x_i}}{\sum_j e^{x_j}}$	$(0, 1)$	Normalizes a vector into a probability distribution (sums to 1).	**Output Layer:** Multi-class classification.
GELU	$x \cdot \Phi(x)$	$(-\infty, \infty)$	Weights inputs by their percentile; standard in Transformers.	**Transformers:** BERT, GPT, and modern NLP models.
Swish	$f(x) = x \cdot \sigma(x)$	$(-\infty, \infty)$	Smooth and non-monotonic; often outperforms ReLU in deep models.	**Hidden Layers:** Very deep networks (e.g., EfficientNet).

4 Choosing the Right Activation Function

The choice of activation function depends on the type of problem you are trying to solve. Here are some guidelines:

4.1 For binary classification

Use the sigmoid activation function in the output layer. It will squash outputs between 0 and 1, representing probabilities for the two classes.

4.2 For multi-class classification

Use the softmax activation function in the output layer. It will output probability distributions over all classes.

4.3 If unsure

Use the ReLU activation function in the hidden layers. ReLU is the most common default activation function and usually a good choice.

5 Conclusion

Activation functions are fundamental components of neural networks that enable them to learn complex, non-linear patterns in data. Each activation function has its strengths and weaknesses, making them suitable for different scenarios:

- **Sigmoid**: Best for binary classification output layers
- **Tanh**: Good for hidden layers, especially with normalized data
- **ReLU**: Default choice for hidden layers in most networks
- **Softmax**: Essential for multi-class classification output layers
- **Linear**: Suitable for regression output layers

Understanding these functions' mathematical properties, advantages, and limitations will help us design more effective neural networks and troubleshoot training issues.

6 FAQs

- **What happens if I don't use any activation function?** The network becomes essentially a linear regression model, unable to learn complex patterns.
- **Can I mix different activation functions in the same network?** Yes, it's common to use ReLU in hidden layers and sigmoid/softmax in the output layer.
- **Why is ReLU so popular despite its simplicity?** It helps mitigate the vanishing gradient problem, is computationally efficient, and often leads to faster convergence.
- **When should I avoid using sigmoid?** Avoid sigmoid in deep networks with many hidden layers due to the vanishing gradient problem.