**Activation Functions**

An **activation function** is a mathematical function used in machine learning and deep learning models, particularly in neural networks, to determine whether a neuron should be activated or not. It introduces **non-linearity** into the model, which allows the network to learn and model complex data patterns.

Why Activation Functions Are Important:

* Non-linearity: They allow neural networks to learn and approximate non-linear functions, which is essential for solving real-world problems.
* Modeling Complex Patterns: Without activation functions, a neural network would simply be a linear function, limiting its ability to model complex data relationships.

**Activation Function in Machine Learning**

In traditional machine learning, activation functions aren't typically a core component. Models like decision trees, support vector machines (SVM), and k-nearest neighbors (KNN) don’t explicitly use activation functions because they don't operate like neural networks.

However, some machine learning models, such as **logistic regression**, do use activation functions like **Sigmoid**. In these cases, activation functions help map the output of the model to a probability or a desired range.

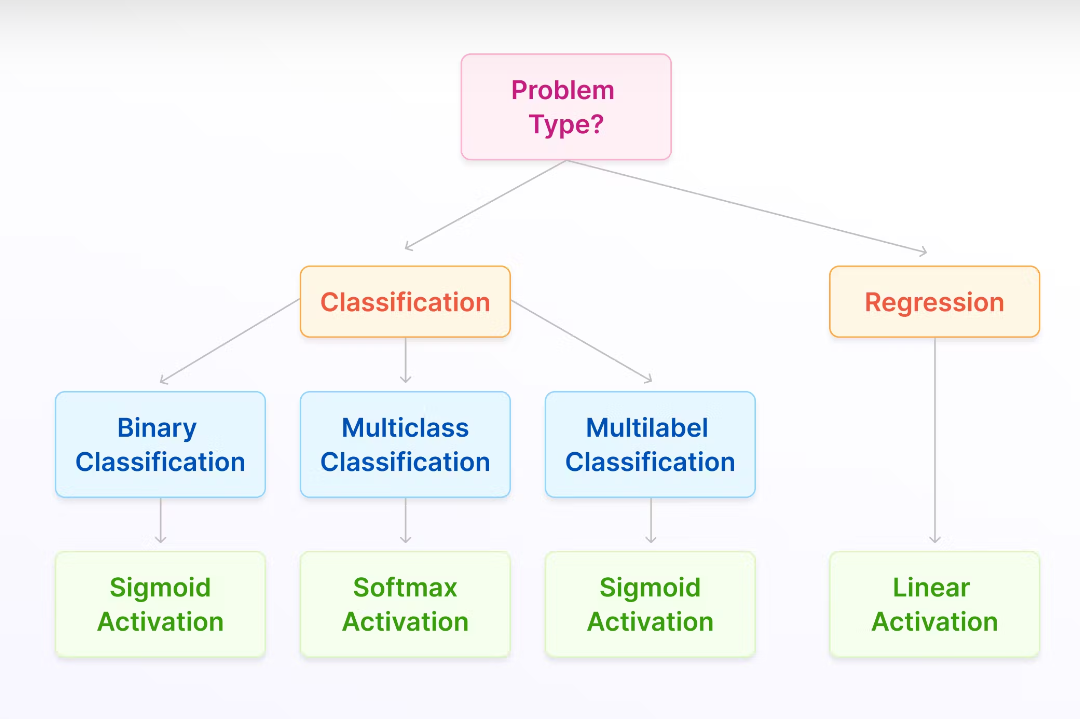
For example:

* In **logistic regression**, the **Sigmoid function** converts the output of a linear function into a probability between 0 and 1, which can be interpreted as the likelihood of a binary outcome.

**Activation Function in Deep Learning**

In deep learning, activation functions are a crucial component of neural networks. They are applied to the output of neurons in each layer of the network, enabling the model to capture non-linear relationships between the inputs and outputs.

Without activation functions, a neural network would simply be a linear combination of its inputs, limiting its ability to solve complex problems. Activation functions allow the network to model more complicated patterns by introducing non-linear transformations at each layer.

****

**TABLE of Activation Function**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Activation Function** | **Definition** | **Output Range** | **Typical Usage** | **Formula** | **Key-Benefit** |
| Linear Activation | |  | | --- | |  |  |  | | --- | | The Linear Activation function simply returns the input as the output without applying any transformation. | | −∞ to ∞ | Used in the output layer of regression tasks, where the network needs to predict continuous values. | f(x)=x | Keeps the output layer simple when the model is required to predict real-valued quantities. |
| Sigmoid | The sigmoid function is a smooth, S-shaped curve that outputs a value between 0 and 1. It's commonly used in binary classification problems. | 0 to 1 | Binary classification | σ(x)=1/1+e−x1​ | Smooth, easy to use |
| ReLU(Rectified Linear Unit) | The ReLU function outputs the input directly if it’s positive; otherwise, it outputs zero. It's the most widely used activation function in deep learning. | 0 to ∞ | Hidden layers of deep networks especially in CNN | ReLU(x)=max(0,x) | Efficient, simple |
| Tanh(Hyperbolic Tangent) | Tanh is similar to sigmoid but outputs values between -1 and 1. It’s useful when inputs need to be centered. | -1 to 1 | Zero-centered data | Tanh(x)=(2/1+e−2x)​−1 | Good for zero-centered data |
| Leaky ReLU | A variation of ReLU that allows a small, non-zero gradient when the input is negative. | Negative to positive | Used in deep networks to avoid the issue of neurons dying during training (i.e., not updating weights due to zero gradients). | Leaky ReLU(x)= max(0.01x,x) | Helps avoid vanishing gradients |
| Softmax | The softmax function converts a vector of values into a probability distribution, where the sum of all probabilities is 1. | 0 to 1 (sum = 1) | Multi-class classification | Softmax(xi​)=  e^xi/​​∑j​e^xj​​ | Outputs probabilities |
| Swish | The Swish function, developed by Google, is a smooth, non-monotonic activation function that often performs better than ReLU. | -∞ to ∞ | General deep learning tasks | x⋅1/1+e^−x1​ | **Smooth, non-linear outputs** while allowing small negative values to pass through |