

Deep Learning & Neural Networks

Dr. Shahira Ahmed

DS710

Lecture 7





Deep Learning & Neural Networks DS710

Lecture 7
Exploring Activation Functions:
Key Ingredients for Deep Neural
Networks

Dr. Shahira Ahmed



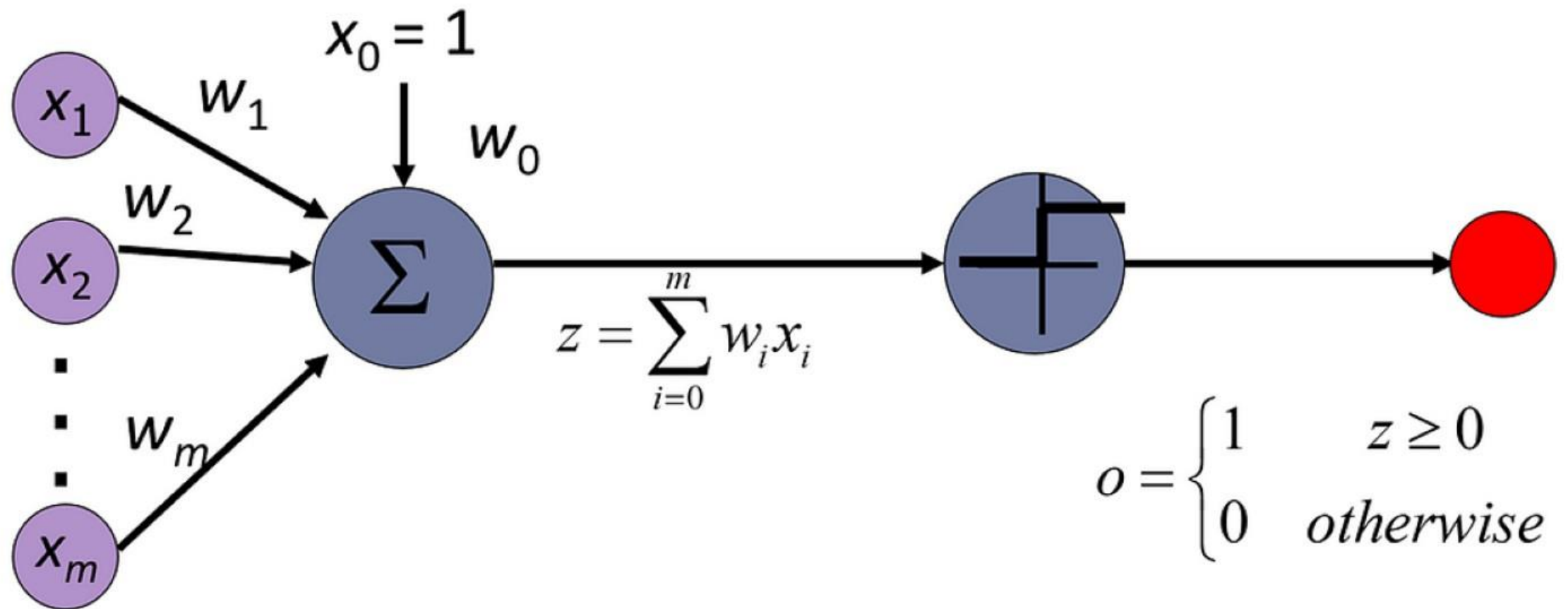
LEARNING OUTCOMES

- ◉ Define activation functions and understand their role in neural networks and deep learning models.
- ◉ Recognize and describe the characteristics of common activation functions
- ◉ Understand the advantages, limitations, and appropriate use cases for each activation function

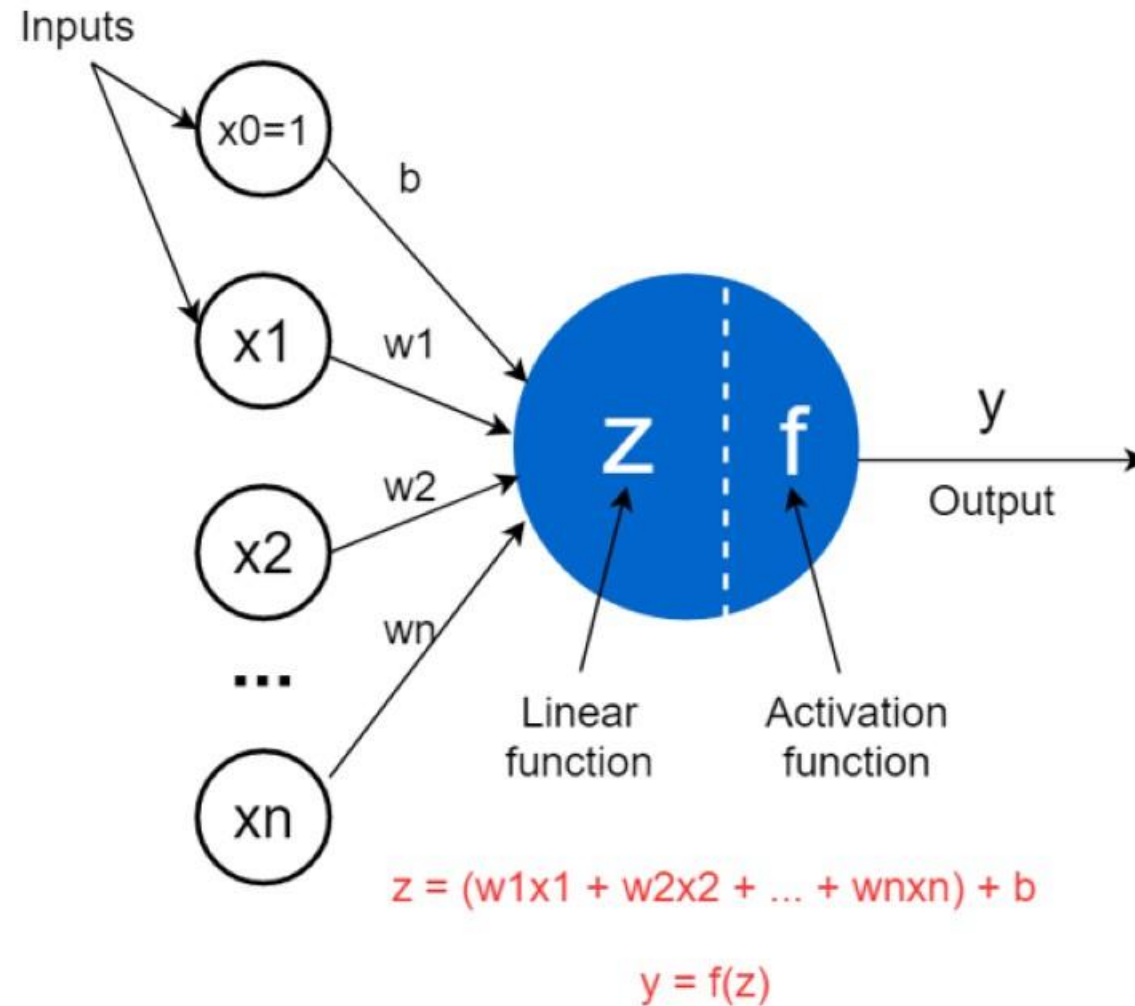


LECTURE OUTLINE

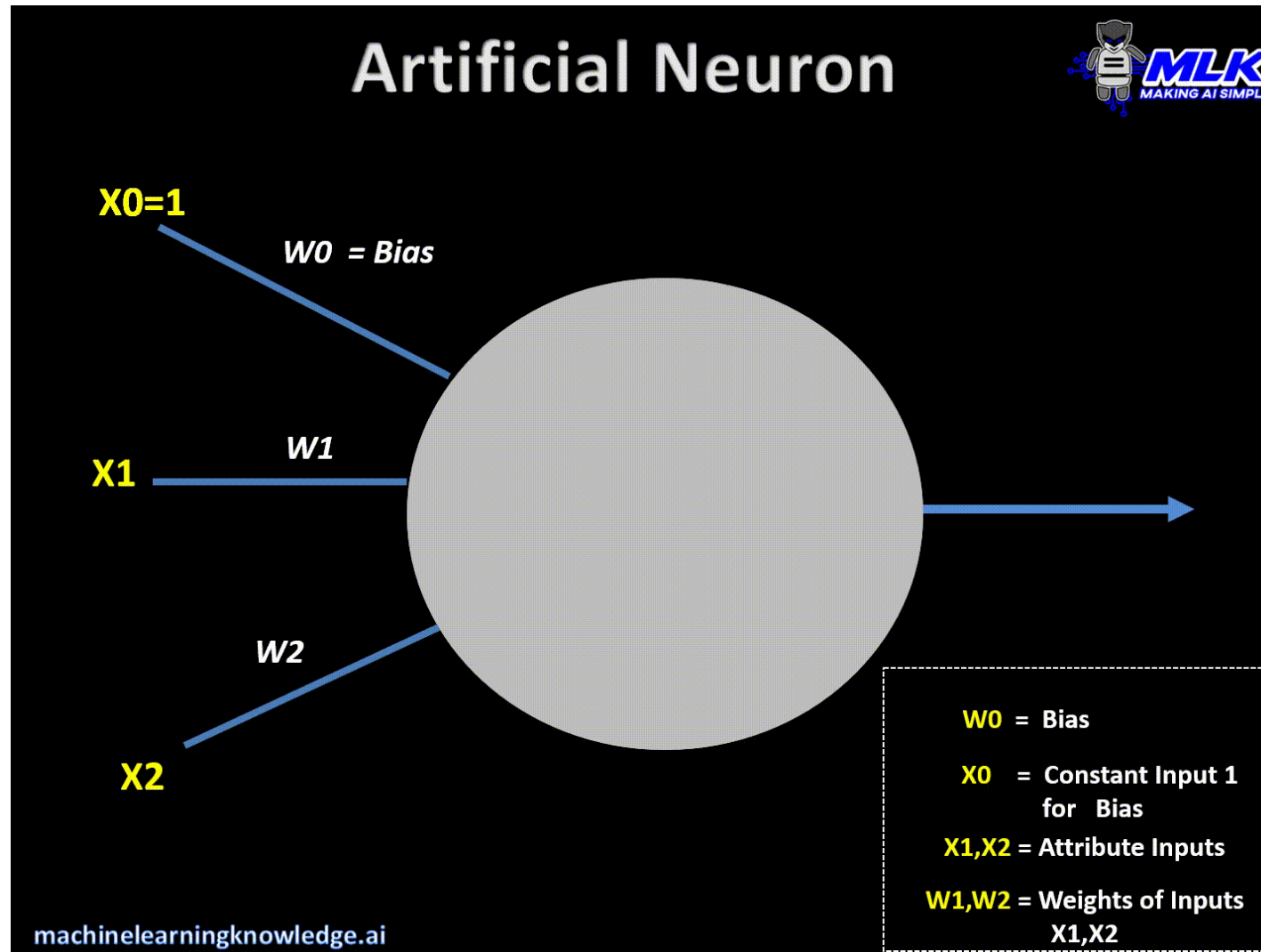
- ◉ Introduction to Activation Functions.
- ◉ Common Activation Functions.
- ◉ Choosing the right Activation Functions.
- ◉ Lab: image classification.



The structure of a perceptron

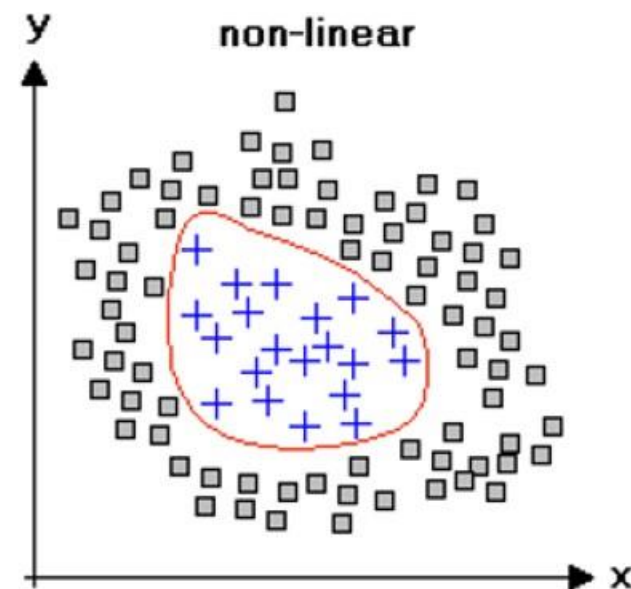
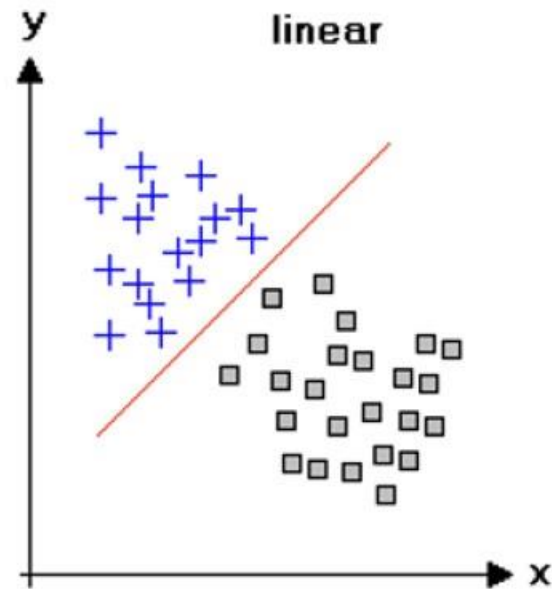


Introduction to Activation Functions



- Activation functions play a fundamental role in introducing **non-linearity** and enabling **complex mappings** within neural networks.
- Understanding activation functions is essential for designing effective and powerful deep learning models.
- **Without activation functions**, a neural network would only be able to **learn linear relationships** between inputs and outputs.

Linear vs non-Linear



Linear vs non-Linear



Linear function



Non-linear function

Why Activation Function?

- In a neural network, each neuron performs a **linear transformation** on its input by multiplying the input values with corresponding weights and summing them up. This step is represented by the equation:

$$Z = w_1 * x_1 + w_2 * x_2 + \dots + w_n * x_n + b$$

- where z is the **weighted sum** of inputs $x_1, x_2, \dots, x_n$ with corresponding weights $w_1, w_2, \dots, w_n$, and b is a bias term.

Why Activation Function?

- If we consider a network with **multiple layers**, each layer's output would also be a **linear** transformation of the previous layer's output.
- The overall network would still be a composition of linear transformations.
- In this case, no matter how many layers we have, the neural network would essentially be **equivalent to a single-layer** neural network.

Why Activation Function?

- keep the weighted sum of inputs in a **certain range**
 - **Sigmoid**: between **0 and 1** (both inclusive)
 - **Tanh**: between **-1 and +1** (both inclusive)
 - **ReLU**: between **0 and positive infinity** (zero inclusive)
 - **ReLU6**: between **0 and 6** (both inclusive)
 - **Binary step**: **either 0 or 1**

Why Activation Function?

- Task-specific requirements:
 - In **multi-label classification problems**, where an input can belong to multiple classes simultaneously, the **sigmoid activation** function is commonly used in the output layer to produce independent probabilities for each class.
 - for tasks such as image segmentation, where the goal is to assign a label to each pixel in an image, the **softmax** activation function is often used to produce a probability distribution over the different classes for each pixel.
 - **Regression** Task uses **identity** function

Why Activation Function?

1. To introduce **non-linearity** to neural networks
 - Enable Complex Mapping. making it capable of handling tasks such as image classification, natural language processing, and time series analysis.
2. To determine the neural network's **output**
3. **Task-specific requirements** (multilabel-multiclass-Regression.....)

- 1. Definition:** Vanishing gradients refer to the phenomenon where the **gradients** calculated during backpropagation **become extremely small** as they propagate from the output layers to the earlier layers of the network.
- 2. Causes:** Vanishing gradients often occur when using activation functions that **squash large input ranges into a small output range**, such as the **sigmoid** or hyperbolic tangent (**tanh**) functions. As the gradients are **repeatedly multiplied** during backpropagation, they can exponentially decrease, resulting **in nearly zero gradients** for the earlier layers.

3. Consequences: When gradients become vanishingly small, it becomes difficult for the network to update the weights of the earlier layers effectively. This leads to slower convergence.

4. Mitigation: Several techniques can help alleviate the vanishing gradient problem, such as using **activation functions** like the rectified linear unit (**ReLU**) or variants like Leaky ReLU and Parametric ReLU. Additionally, **normalization** techniques like batch normalization and layer normalization can help stabilize the gradients.

1. **Definition:** Exploding gradients refer to the opposite problem, where the gradients during backpropagation become **extremely large**.
2. **Causes:** Exploding gradients often occur when gradients are **repeatedly multiplied during backpropagation**, causing them to **grow exponentially**. This can happen when the **weights** in the network are **initialized too large** or during the training process when the **learning rate is set too high**.

3. Consequences: When gradients become extremely large, they can cause **unstable** updates to the network's weights, leading to erratic behavior and poor convergence.

4. Mitigation: To address the exploding gradient problem, gradient clipping can be applied, where the gradients are rescaled if they exceed a certain threshold. Another approach is to use weight initialization techniques that ensure the weights are initialized within a suitable range. Additionally, reducing the learning rate or using adaptive optimization algorithms like Adam can help prevent the gradients from exploding.

1. **Non-Linearity** – Activation function should be able to add nonlinearity in neural network especially in the neurons of hidden layers.
2. **Differentiability**: Differentiability is an important characteristic for activation functions, especially when using **gradient-based optimization** algorithms for training neural networks. Activation functions need to be differentiable to allow for efficient computation of gradients during **backpropagation**.

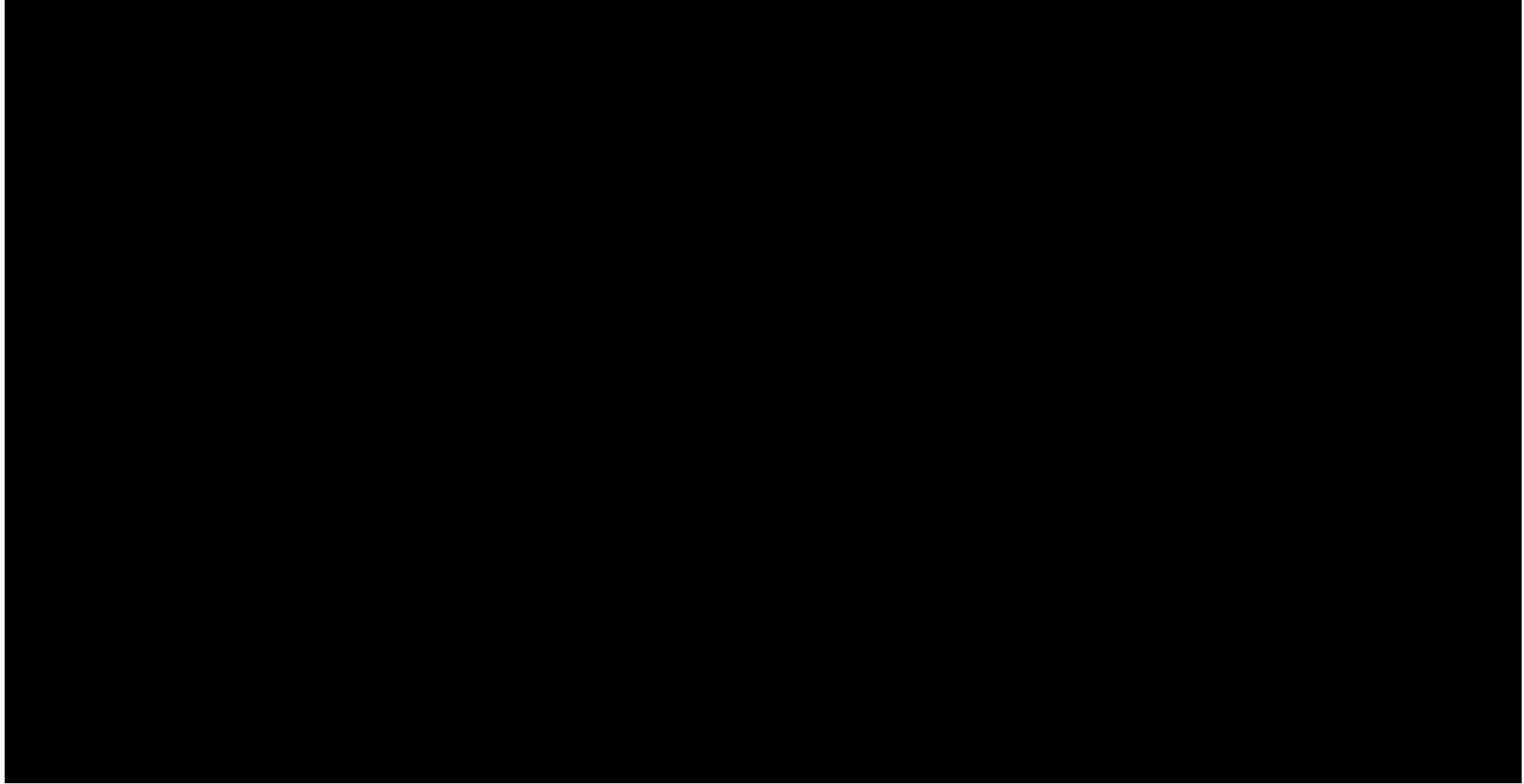
3. Computational efficiency: Activation functions can differ in terms of their computational efficiency. Some functions, like the ReLU function, are computationally efficient to compute and can accelerate training compared to other functions.

4. Vanishing and exploding gradients: Activation functions can have an impact on the occurrence of vanishing or exploding gradients, which can affect the stability and convergence of the neural network during training. Some functions, like the sigmoid function, are prone to the vanishing gradient problem.

(1)-Step Function - Characteristics:

- The step function, also known as the **Heaviside function**
- If the input is greater than or equal to a 0, the output is 1.
- If the input is less than 0, the output is 0.

Step Function



<https://machinelearningknowledge.ai/activation-functions-neural-network/>

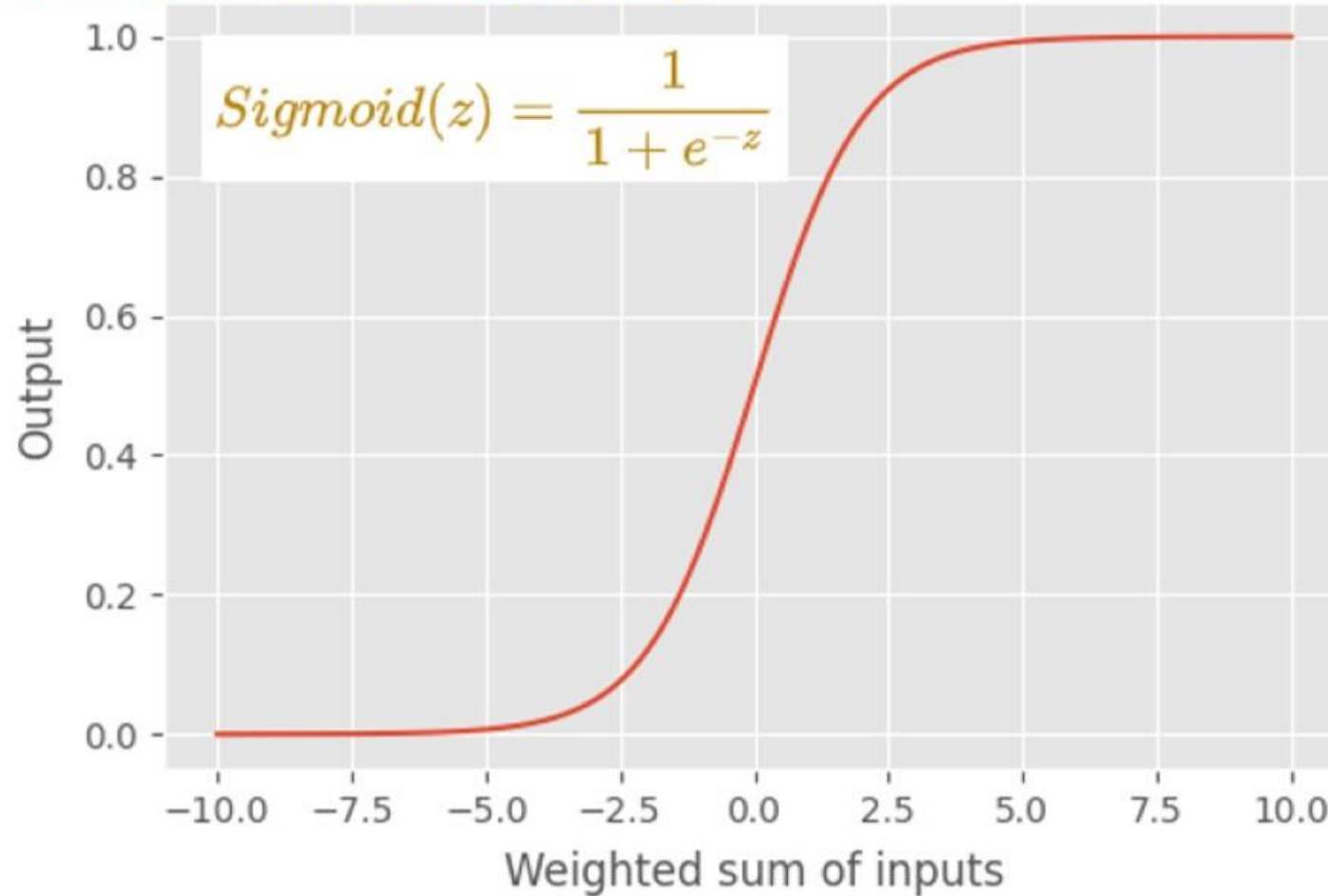
Step Function – use cases

- Step function has **historical** importance as it was used in McCulloch Pitts Neuron & Perceptron that were the earliest models of an artificial neuron.

Step Function –Limitation

- Lack of flexibility: Step functions have fixed **thresholds** or switching points where the output changes. Once defined, these thresholds cannot be easily adjusted or adapted.
- The step function is (**not continuous**) not **smooth** and has a **sudden change** at the threshold value.
- It is **non-differentiable**.

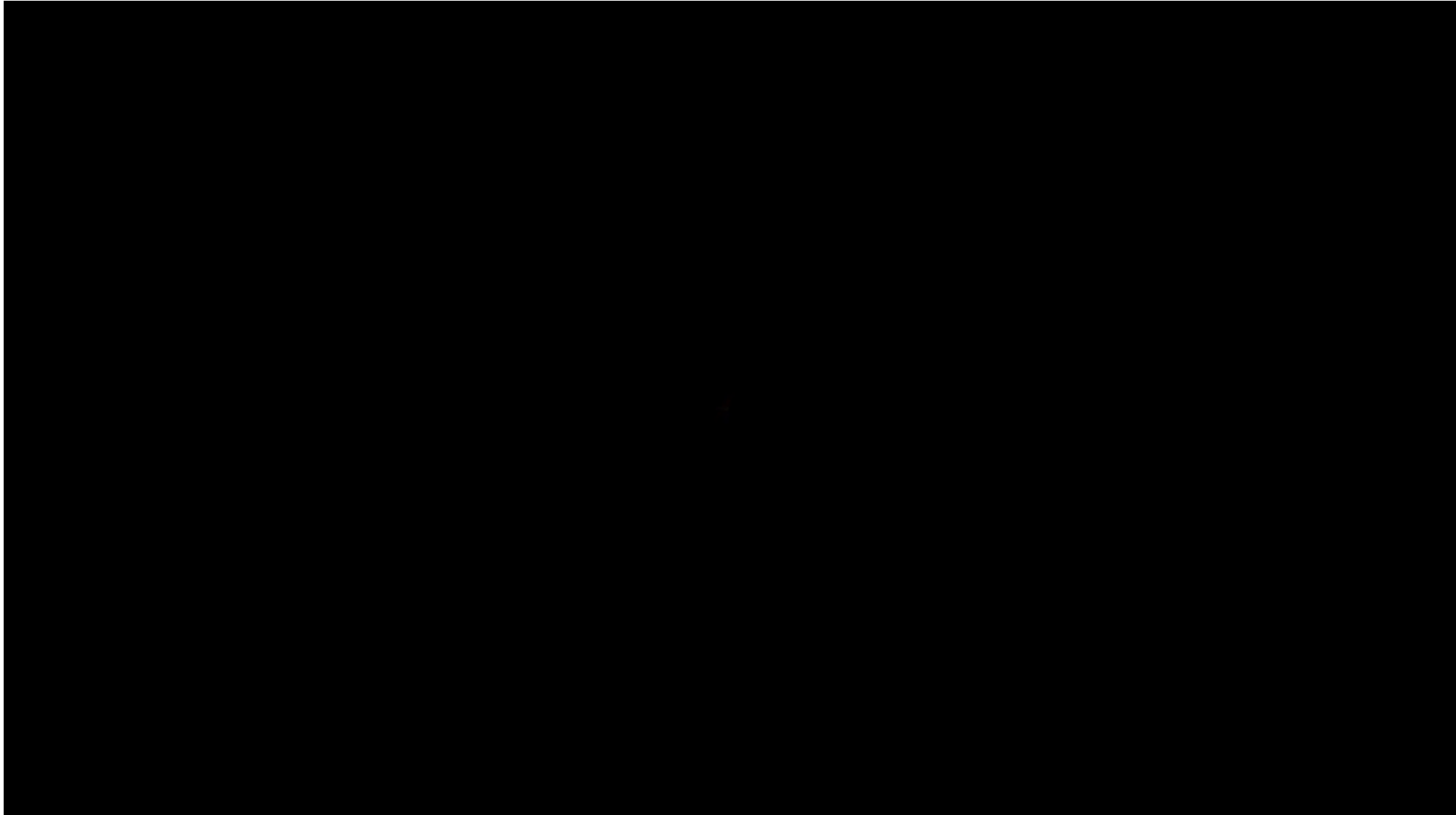
(2)- Sigmoid Function: Characteristics



(2)-Sigmoid Function: Characteristics

- This is also called the **logistic** function used in **logistic regression models**.
- The sigmoid function has an **s-shaped graph**.
- Clearly, this is a **non-linear function**.
- The sigmoid function converts its input into a probability value between 0 and 1.
 - It converts large negative values towards 0 and large positive values towards 1.
 - It returns 0.5 for the input 0. The value 0.5 is known as the threshold value which can decide that a given input belongs to what type of two classes.

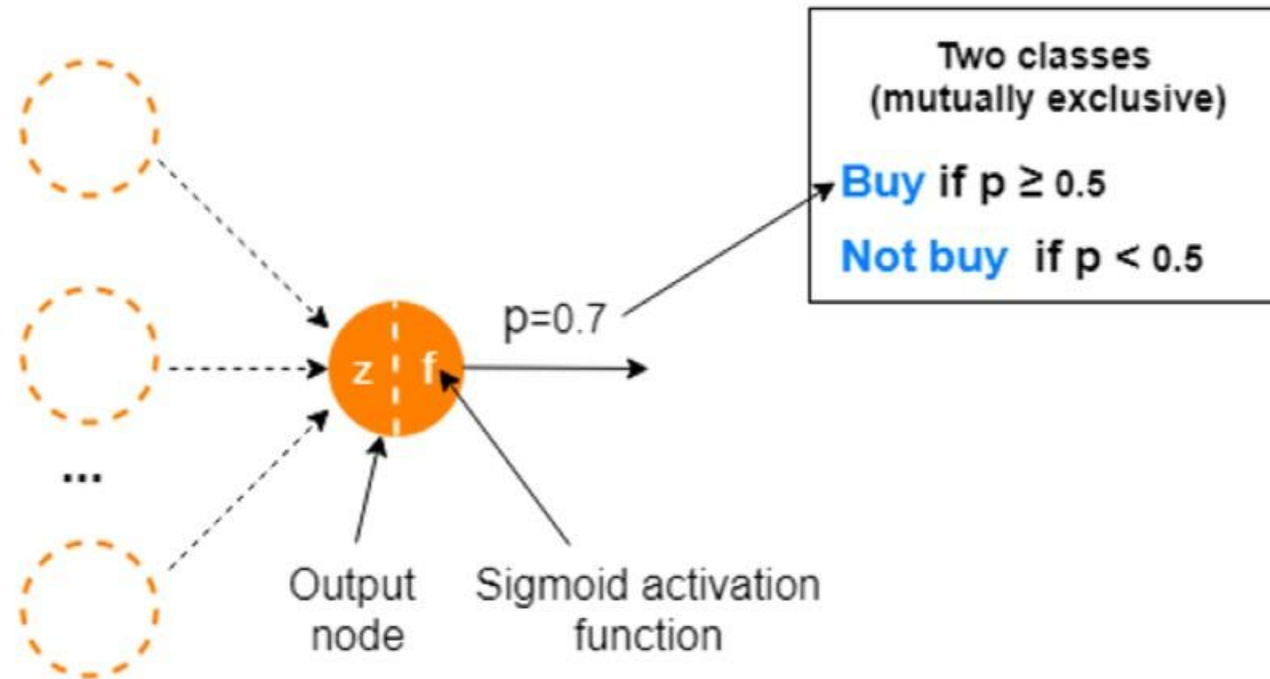
Sigmoid activation function



Sigmoid activation function

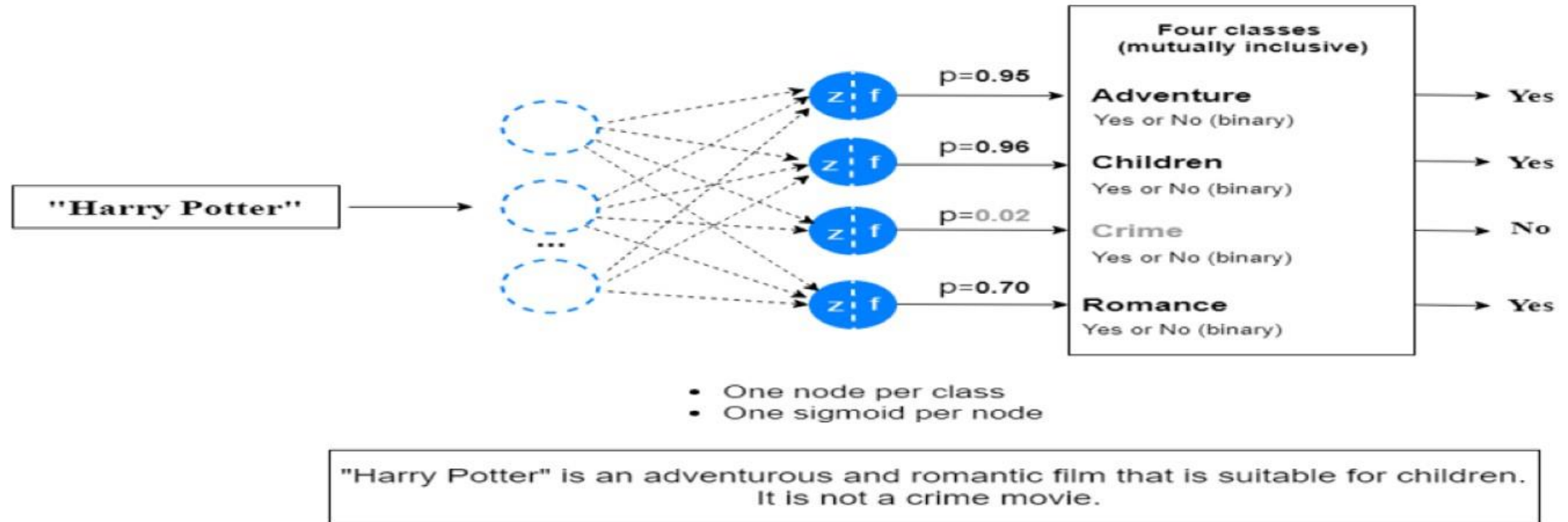
Sigmoid Function: use cases:

Binary classifier with sigmoid activation function



Sigmoid Function: use cases:

Multilabel classification model with sigmoid activation function



Sigmoid Function - use cases:

- The sigmoid function **was** used as an activation function for the hidden layers in MLPs, CNNs and RNNs.
- The sigmoid function is still used in **RNNs**.
- The sigmoid function must be used in the output layer when we build a **binary classifier** in which the output is interpreted as a class label depending on the probability value of input returned by the function.
- The sigmoid function is used when we build a **multilabel classification** model in which each mutually inclusive class has two outcomes.

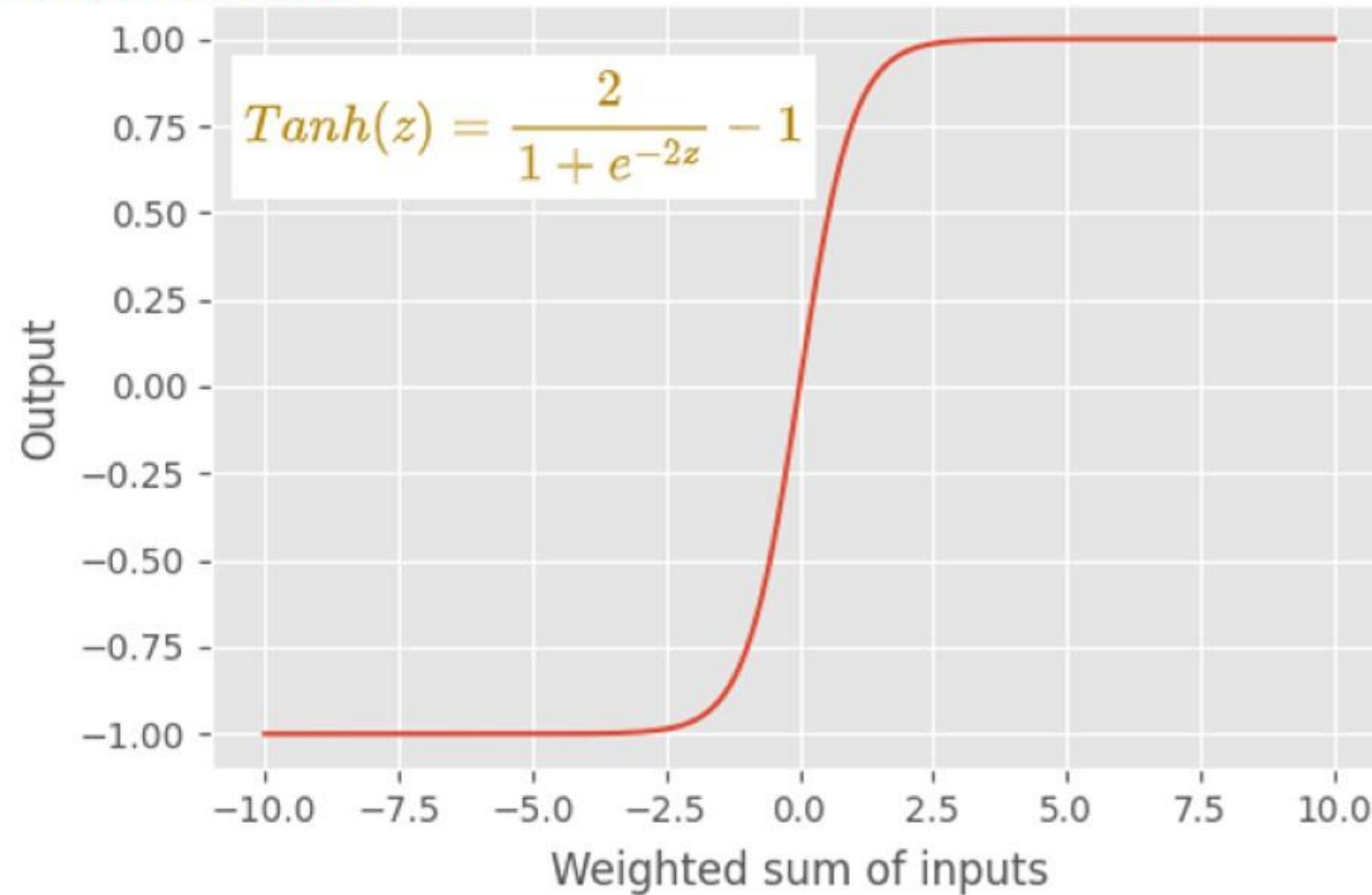
Sigmoid Function - use cases:

- The sigmoid function can also be used for other tasks, such as image processing, speech recognition, and natural language processing
 - In **image processing**, the sigmoid function is used to enhance the contrast of an image.
 - In **speech recognition**, the sigmoid function is used to model the probability of a particular sound given an input signal.
 - In **natural language processing**, the sigmoid function is used to model the probability of a particular word given a sequence of words.

Sigmoid Function - Limitation:

- The sigmoid function has the **vanishing gradient** problem. The sigmoid function has slow convergence.
- The sigmoid function is **not zero-centered**, which means that its output is always positive. This can cause issues in the gradient update of the weights during backpropagation since the positive gradients will only update the weights in a specific direction
- Output range limitation: The output of the sigmoid function is constrained **between 0 and 1**.

(3) Tanh Activation Function



Tanh Activation Function - characteristics

- The output of the tanh (tangent hyperbolic) function always ranges **between -1 and +1**.
- Like the sigmoid function, it has an s-shaped graph. This is also **a non-linear** function.
- One advantage of using the tanh function over the sigmoid function is that the tanh function is **zero centered**. This makes the optimization process much easier.

Common Activation Functions



<https://machinelearningknowledge.ai/activation-functions-neural-network/>

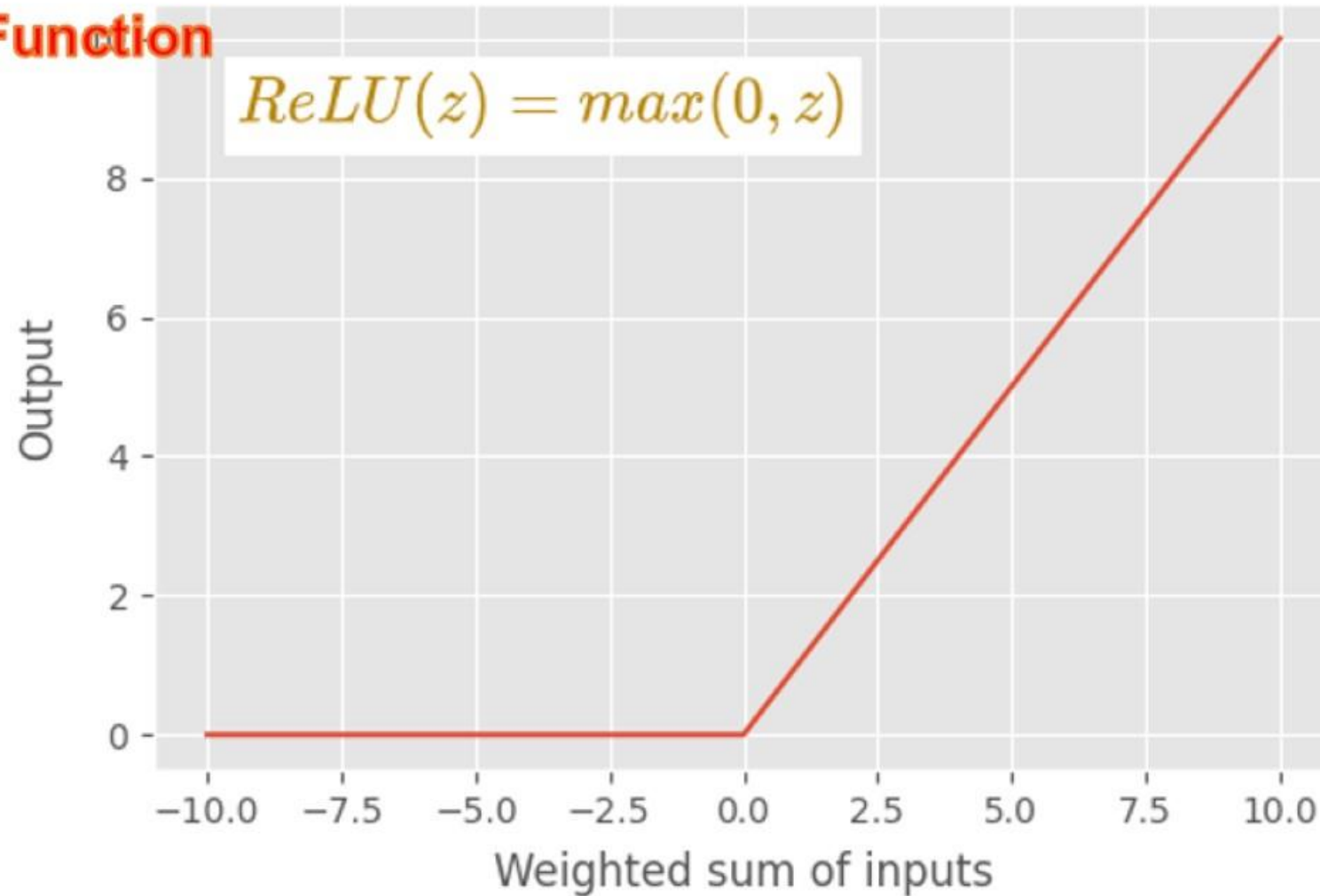
Tanh Activation Function – use cases:

- Hidden layers: The tanh activation function is **used in hidden layers** of neural networks. It can help introduce non-linearity and enable the network to learn complex representations and feature combinations but lead to **vanishing gradient** problem
- Output layer for classification: The tanh function can be used in the output layer of a neural network for **binary classification** problems, where the goal is to predict a binary outcome (e.g., positive or negative)

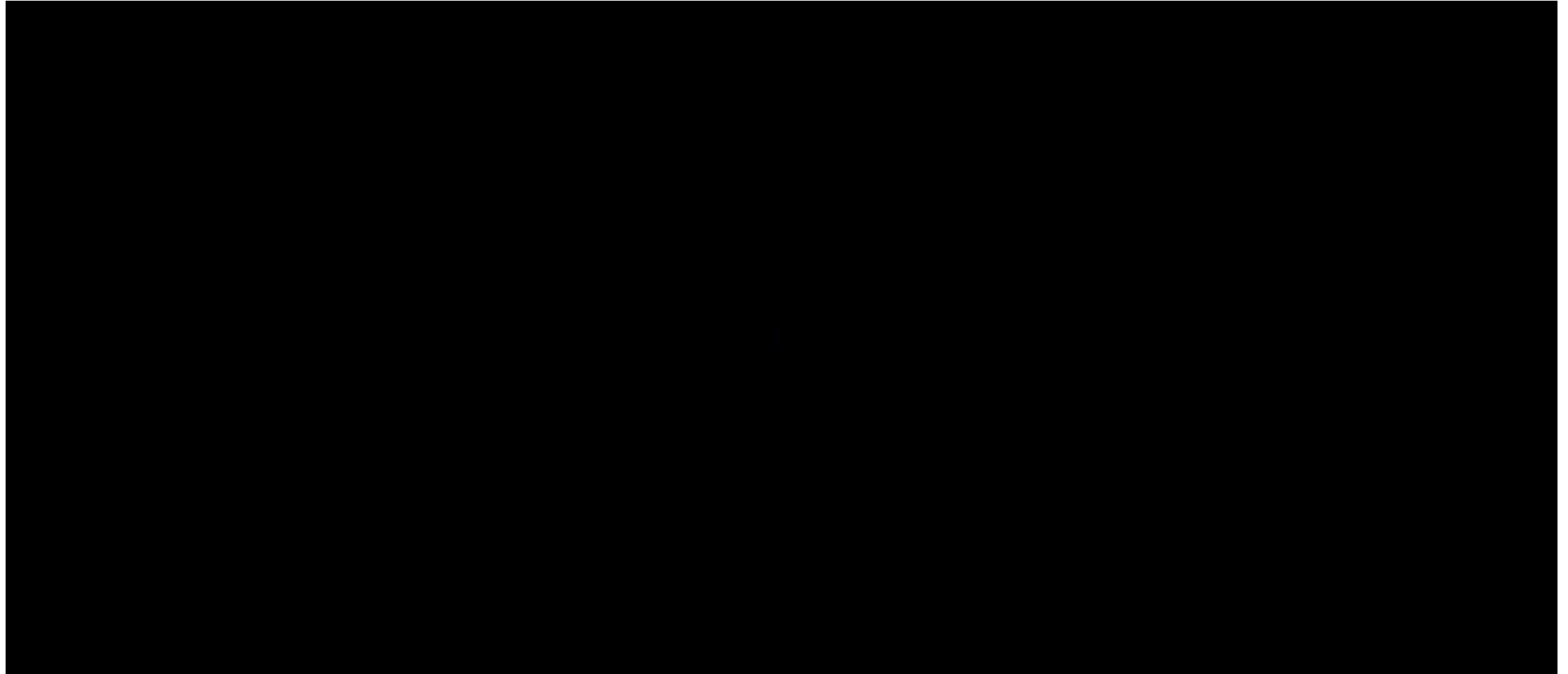
Tanh Activation Function – limitation:

- Vanishing gradient problem: Although the tanh function mitigates the **vanishing gradient** problem compared to the sigmoid function, it can still suffer from this issue, particularly for large inputs.
- **Saturation**: the tanh function saturates at the extremes of its range (-1 and 1). This saturation can lead to **slower learning** and challenges in updating the network weights properly.
- Limited **output range**: The output of the tanh function is limited to the range between -1,1
- **Computational cost**: The tanh function involves exponentiation and division operations, which can be computationally more expensive compared to the **ReLU** activation function and its variants.

(4)-ReLU Function



Common Activation Functions



<https://machinelearningknowledge.ai/activation-functions-neural-network/>

ReLU Activation Function – Characteristics:

- The ReLU (Rectified Linear Unit) activation function is a great alternative to both sigmoid and tanh activation functions.
- Inventing ReLU is one of the most important breakthroughs made in deep learning.
- This function **does not have the vanishing gradient** problem.
- ReLU is a simple function to compute since it involves a comparison and a maximum operation. This simplicity makes it **computationally efficient**, especially when dealing with large-scale neural networks and deep learning models..

ReLU Activation Function – Characteristics:

- The ReLU function is made up of two linear components. Because of that, the ReLU function is a piecewise linear function. In fact, the ReLU function is a **non-linear** function.
- The output of the ReLU function can **range** from 0 to positive infinity($0, +\infty$).
- The **convergence is faster** than sigmoid and tanh functions. This is because the ReLU function has a fixed derivate (slope) for one linear component and a zero derivative for the other linear component. Therefore, the learning process is **much faster** with the ReLU function. **Calculations** can be performed **much faster** with ReLU because no exponential terms are included in the function.

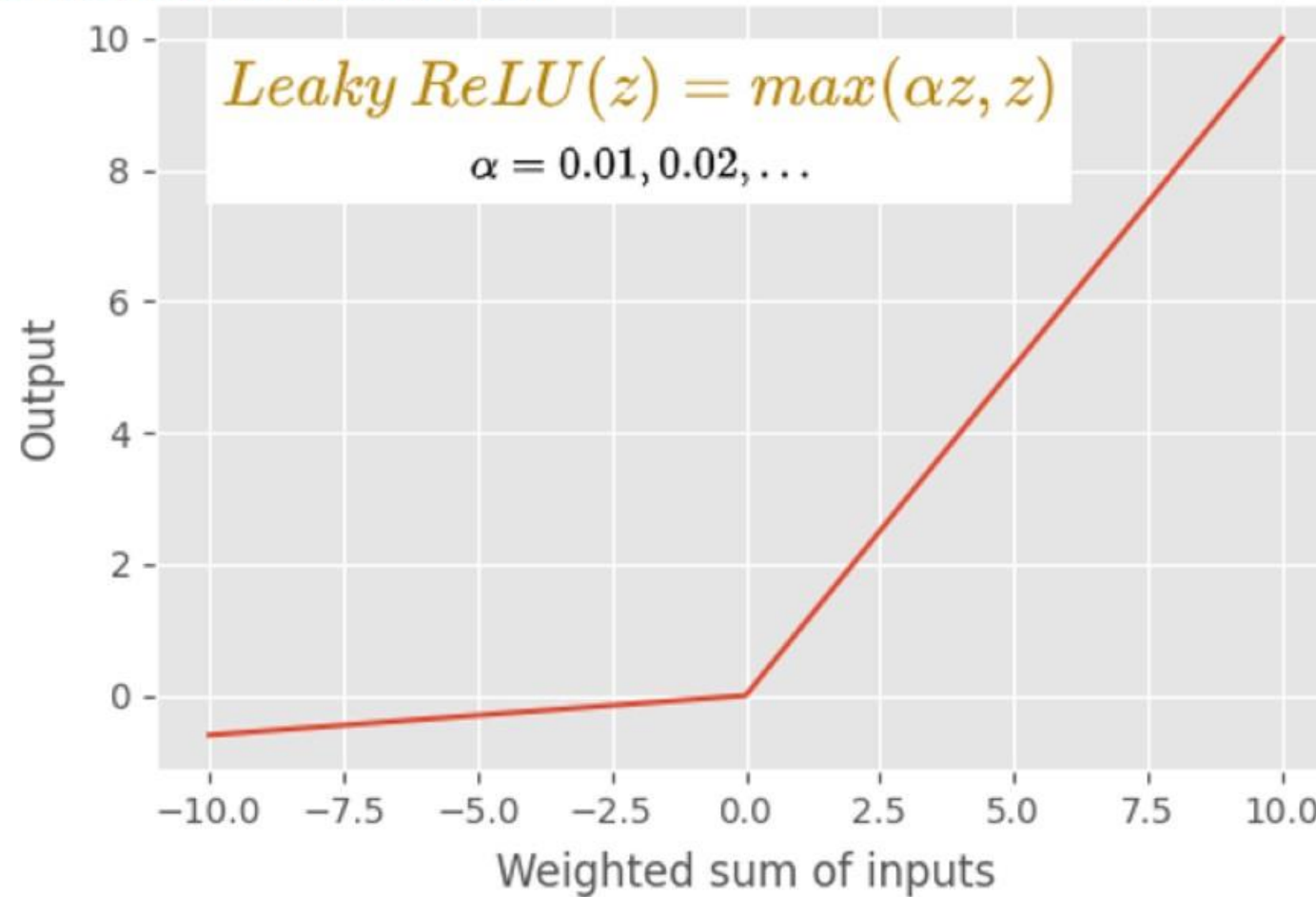
ReLU Activation Function – use cases :

- The ReLU function is the **default activation function** for hidden layers in modern MLP and CNN neural network models.
- We do **not usually** use the ReLU function in the hidden layers of **RNN** models. Instead, we use the sigmoid or tanh function there.
- We **do not use** the ReLU function in the **output layer**.

ReLU Activation Function – Limitation

- **Dead neurons:** If the input to a ReLU neuron is negative, the output is zero and the gradient is also zero. This means that the neuron stops learning and becomes inactive.
- However, ReLU can also cause **exploding gradient problem**, which occurs when the gradients become very large and the network becomes unstable. This can happen if the weights are too large or the network is too deep. To prevent this, we can use gradient clipping, which sets a maximum threshold for the gradients

(5) – Leaky ReLU Activation Function



Common Activation Functions



<https://towardsdatascience.com/how-to-choose-the-right-activation-function-for-neural-networks-394110e619c>

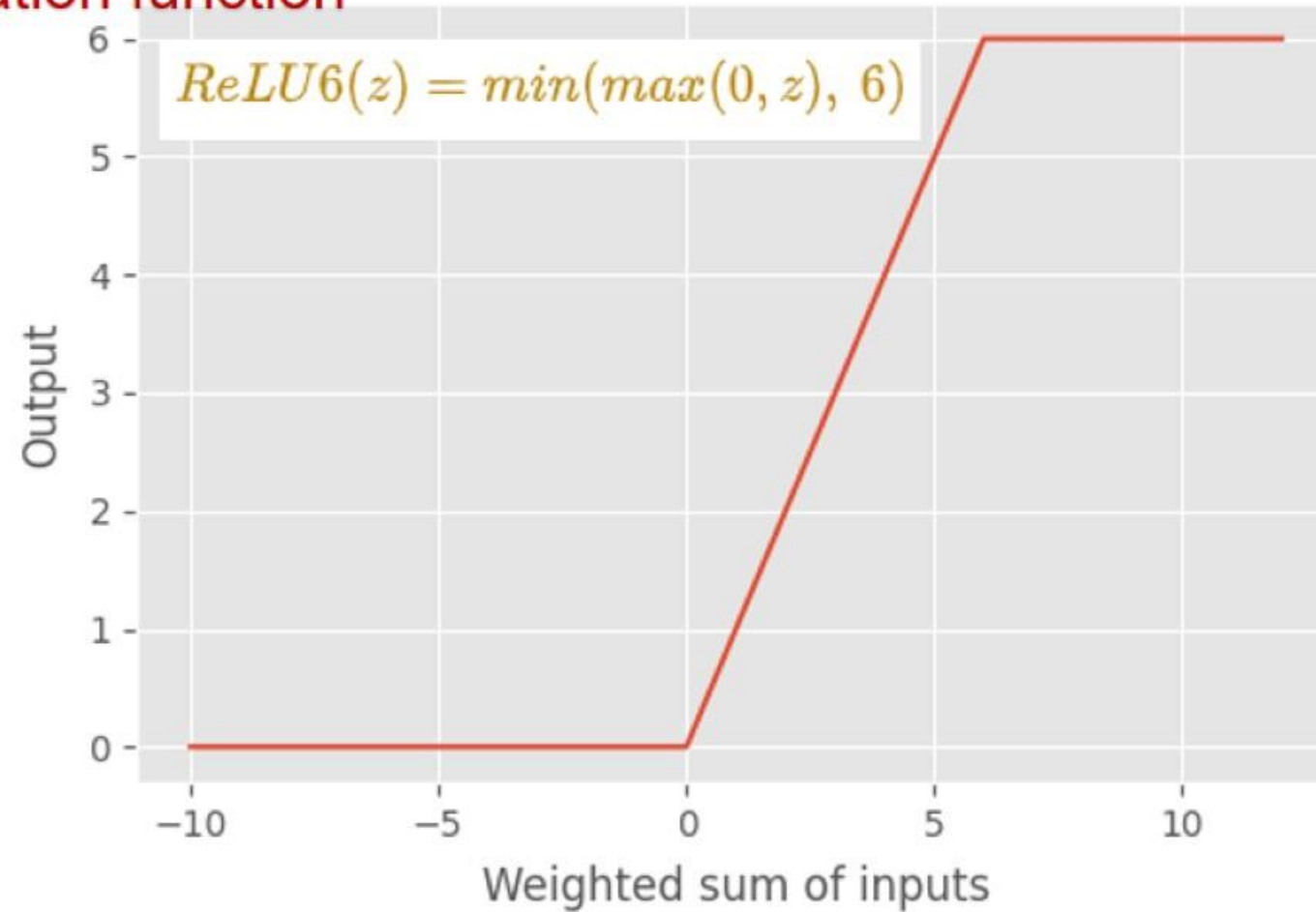
Leaky ReLU Activation Function – Characteristics

- The leaky ReLU activation function is a **modified version of the default ReLU** function.
- Leaky ReLU activation function **does not have the vanishing gradient problem**.
- If the input value is **greater than 0**, the leaky ReLU function **outputs the input** as it is like the default ReLU function does.
- if the input is **less than 0**, the leaky ReLU function **outputs a small negative value** defined by αz (where α is a small constant value, usually 0.01 and z is the input value).
- It does not have any linear component with zero derivatives (slopes). Therefore, it can avoid the dying ReLU problem.
- The learning process with leaky ReLU is **faster than** the default ReLU.

Leaky ReLU Activation Function – Limitation:

- **Extra hyperparameter:** Leaky ReLU introduces another hyperparameter, the slope of the negative part, which has to be tuned before training. This can increase the complexity and risk of overfitting.
- **Computational inefficiency:** Leaky ReLU can be **slower** to compute than ReLU, as it requires additional calculation for the negative inputs. This can affect the training speed, especially for **large datasets**.

ReLU6 activation function



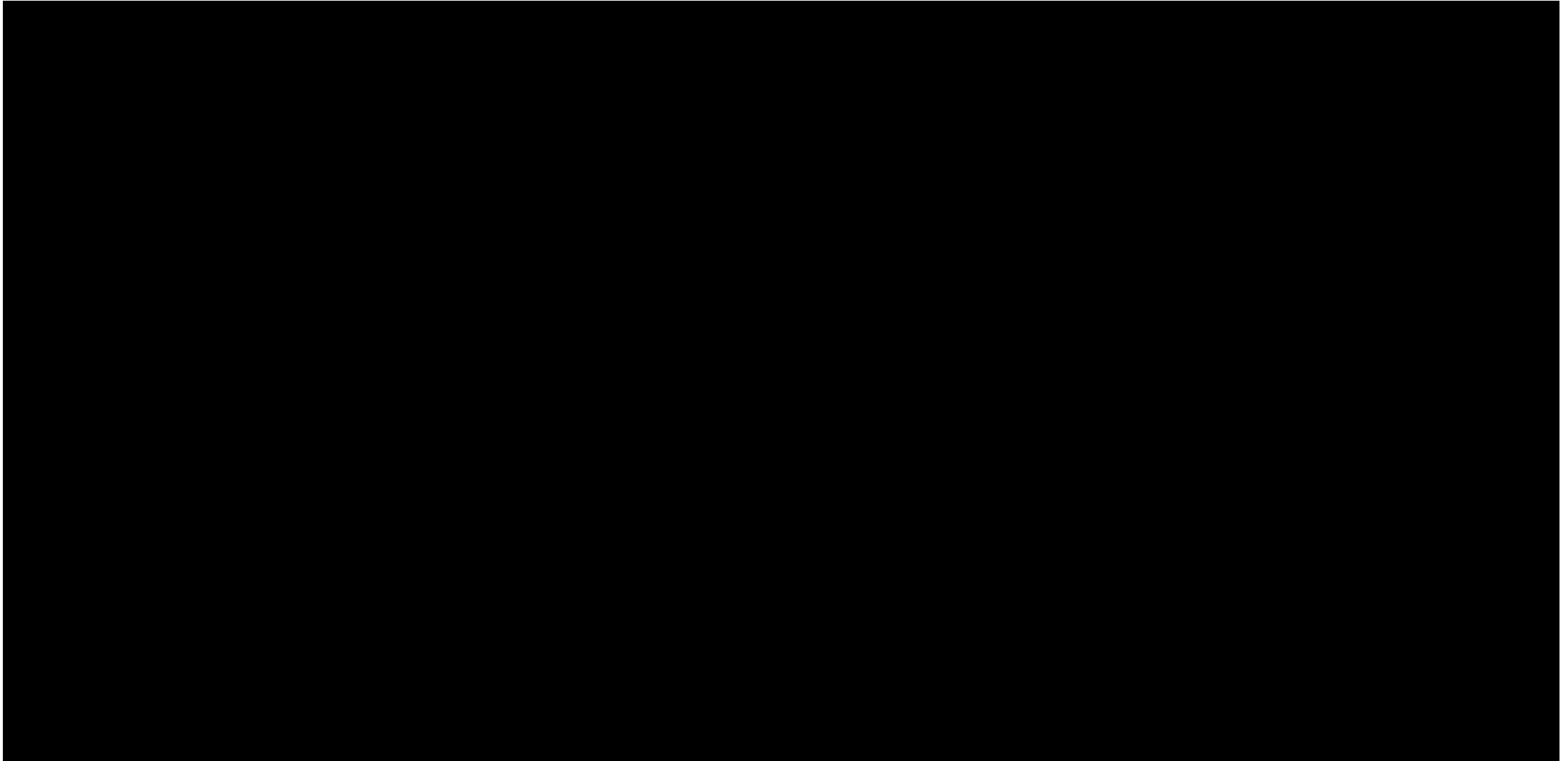
ReLU6 Activation Function – Limitation:

- The main difference between ReLU and ReLU6 is that ReLU allows very high values on the positive side while **ReLU6 restricts to the value 6 on the positive side**. Any input value which is 6 or greater than 6 will be restricted to the value 6 (hence the name).

- Softmax function comes from the family of **Sigmoid functions** only.
- Softmax function **squashes the incoming signals** for **multiple classes** in a probability distribution. The sum of this probability distribution would obviously be **1**.

$$\text{Softmax}(z_i) = \frac{e^{z_i}}{\sum_{j=1}^K e^{z_j}}$$

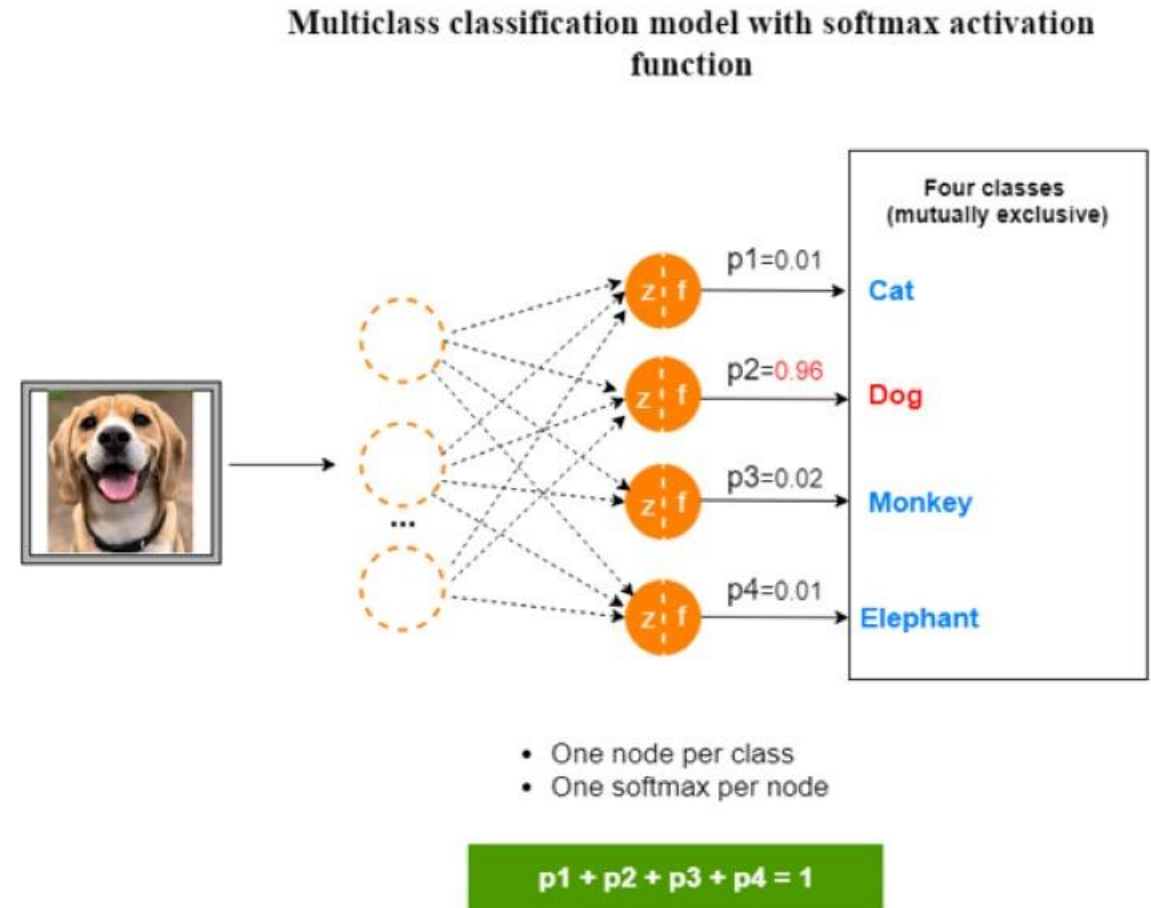
Softmax Function



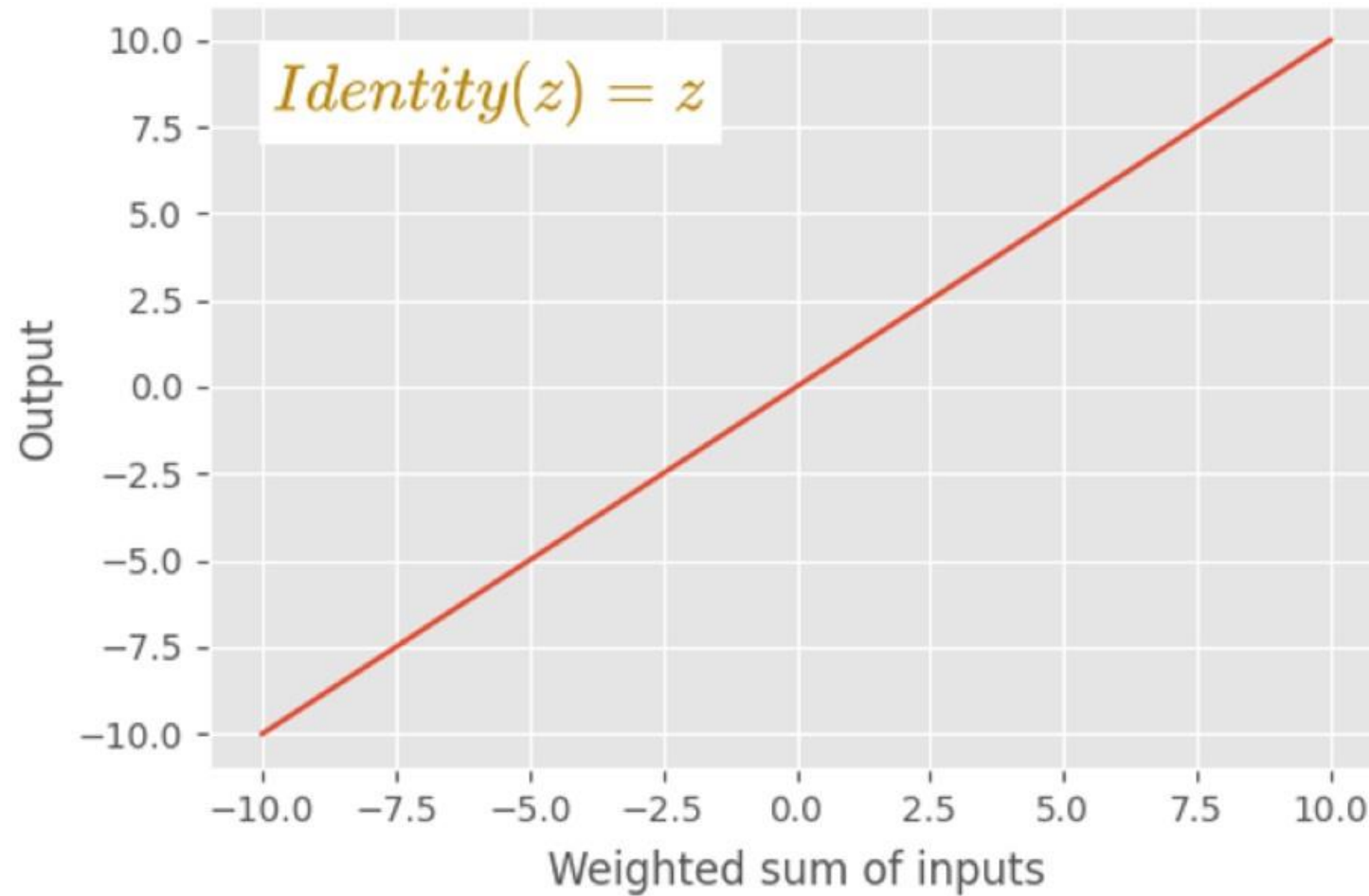
<https://towardsdatascience.com/how-to-choose-the-right-activation-function-for-neural-networks-394110e619c>

Softmax Function

- Usage:
 - We must use the softmax function in the output layer of a multiclass classification problem.
 - We never use the softmax function in the hidden layers.



Identity activation function



- This function outputs the input value as it is. **No changes** are made to the input.
- This function is only used in the output layer of a neural network model that solves a **regression problem**.
- It cannot be used in training of deep learning

Choose the Right Activation Function

- Choosing the right activation function is the **main challenge**.
- No activation function is required in the input layer nodes of a neural network. So, you don't need to worry about activation functions when you define the input layer.

Choose the Right Activation Function

- The output layer activation function depends on the type of problem that we want to solve.
 - In a **regression** problem, we use the linear (**identity**) activation function.
 - In a **binary classifier**, we use the **sigmoid** activation function.
 - In a **multiclass classification** problem, we use the **softmax** activation function.
 - In a **multilabel classification** problem, we use the **sigmoid** activation function.

Choose the Right Activation Function

- We should use a non-linear activation function in hidden layers. For hidden layers, the most used activation function is the rectified linear unit (ReLU). Start with **the ReLU activation** function and if you have **a dying ReLU** problem, try leaky ReLU.
- In MLP and CNN neural network models, ReLU is the **default activation** function for hidden layers.

LAB



LECTURE SUMMARY

- ◉ Activation functions play a crucial role in deep learning by introducing non-linearity to neural networks.
- ◉ ReLU (Rectified Linear Unit) is a widely used activation function due to its simplicity and efficiency.



LECTURE SUMMARY

- ◉ Leaky ReLU are variants of ReLU that address the "dying ReLU" problem
- ◉ Sigmoid and tanh activation.
- ◉ Identity, softmax
- ◉ Lab: Image Classification



Thank You