Neural Networks

Artificial Intelligence

Core Concept of AI

Artificial Intelligence

AI involves techniques that equip computers to emulate human behavior, enabling them to learn, make decisions, recognize patterns, and solve complex problems in a manner akin to human intelligence.

Machine Learning

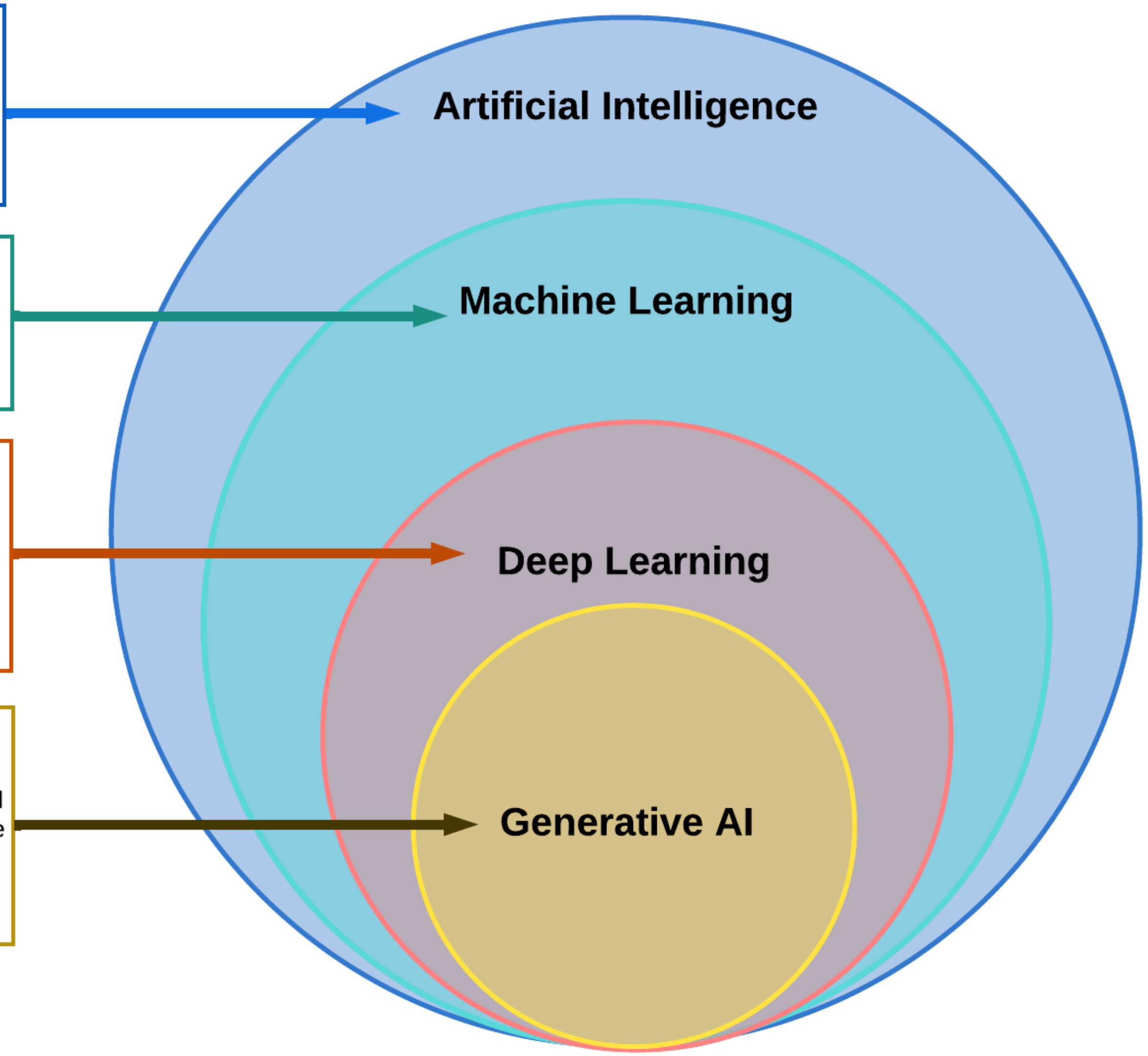
ML is a subset of AI, uses advanced algorithms to detect patterns in large data sets, allowing machines to learn and adapt. ML algorithms use supervised or unsupervised learning methods.

Deep Learning

DL is a subset of ML which uses neural networks for in-depth data processing and analytical tasks. DL leverages multiple layers of artificial neural networks to extract high-level features from raw input data, simulating the way human brains perceive and understand the world.

Generative AI

Generative AI is a subset of DL models that generates content like text, images, or code based on provided input. Trained on vast data sets, these models detect patterns and create outputs without explicit instruction, using a mix of supervised and unsupervised learning.



Architecture of a Biological Neuron

Deep Learning & AI

Neuron in Human Brain

A human neuron is an incredibly complex biological cell with numerous structures like dendrites, an axon, the soma (cell body), and synaptic terminals. The human brain is estimated to contain approximately **86 billion** neurons.

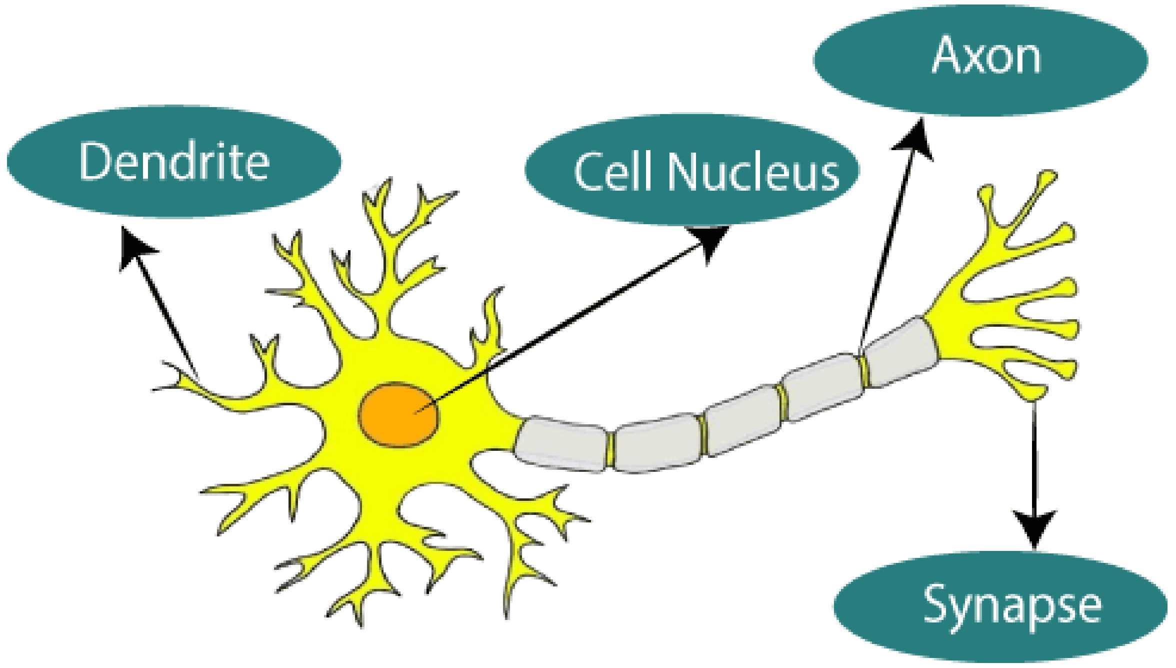


Fig: A Neuron

Types of Neural Network

1. Feedforward Neural Networks (FNN)
2. Convolutional Neural Networks (CNN)
3. Recurrent Neural Networks (RNN)
4. Generative Adversarial Networks (GAN)
5. Autoencoders
6. Transformer Networks

Architecture of an Artificial Neuron

Deep Learning & AI

Artificial Neural Networks (ANNs)

- ❖ An artificial neuron is a simple computational model that includes input weights, an activation function, and an output.
- ❖ It processes information using mathematical functions, taking numeric inputs, multiplying them by weights, summing them up, adding a bias, and passing the result through an activation function.

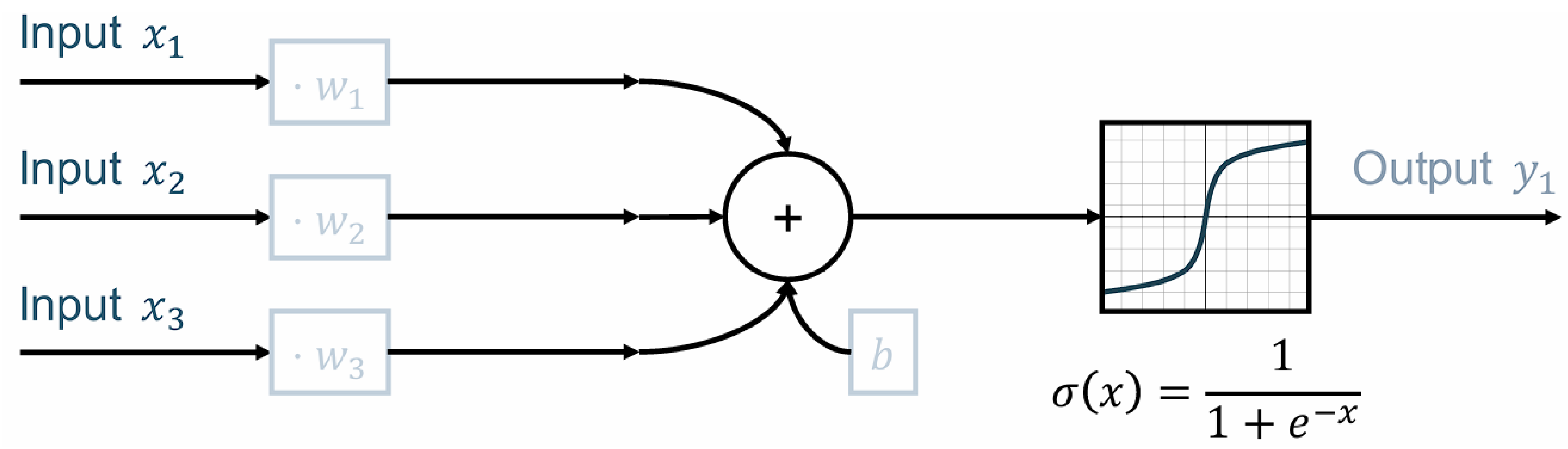


Fig: Artificial Neuron (Single Layer Perceptron)

Deep Learning & AI

Artificial Neural Networks (ANNs)

$$y_1 = \sigma(w_1 \cdot x_1 + w_2 \cdot x_2 + w_3 \cdot x_3 + b) = \sigma(\sum_i w_i \cdot x_i + b)$$

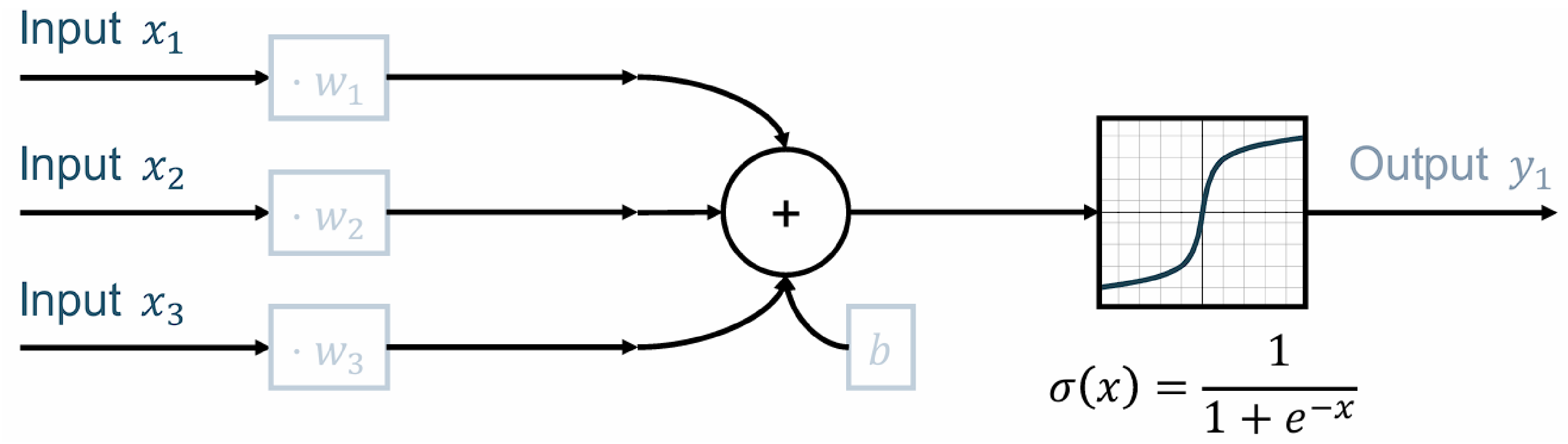
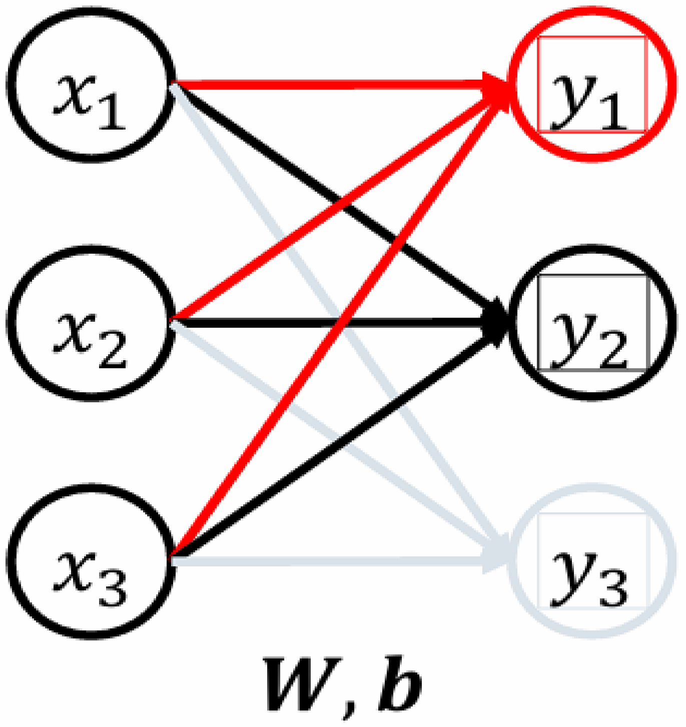


Fig: Artificial Neuron (Single Layer Perceptron)

We can combine multiple perceptrons to create a layer.



We can thus rewrite the three computations as:

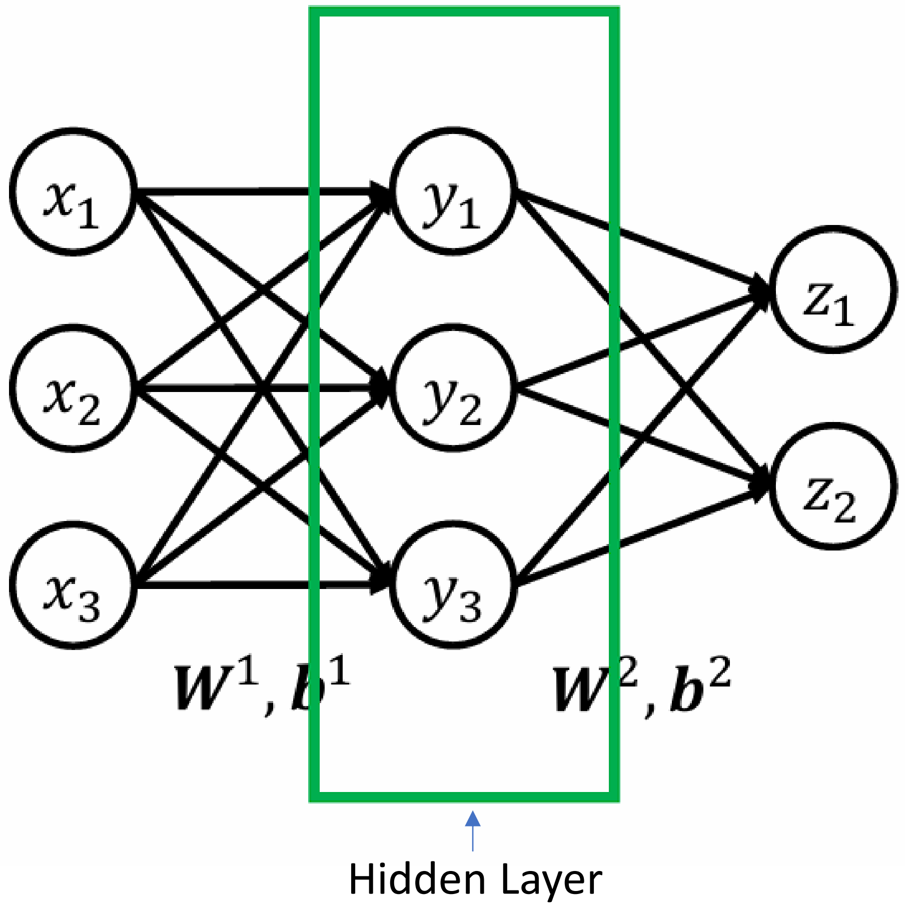
$$\begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix} = \begin{pmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \\ w_{31} & w_{32} & w_{33} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} + \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix}$$

Or in a more simplified form:

$$y = \sigma(W \cdot x + b)$$

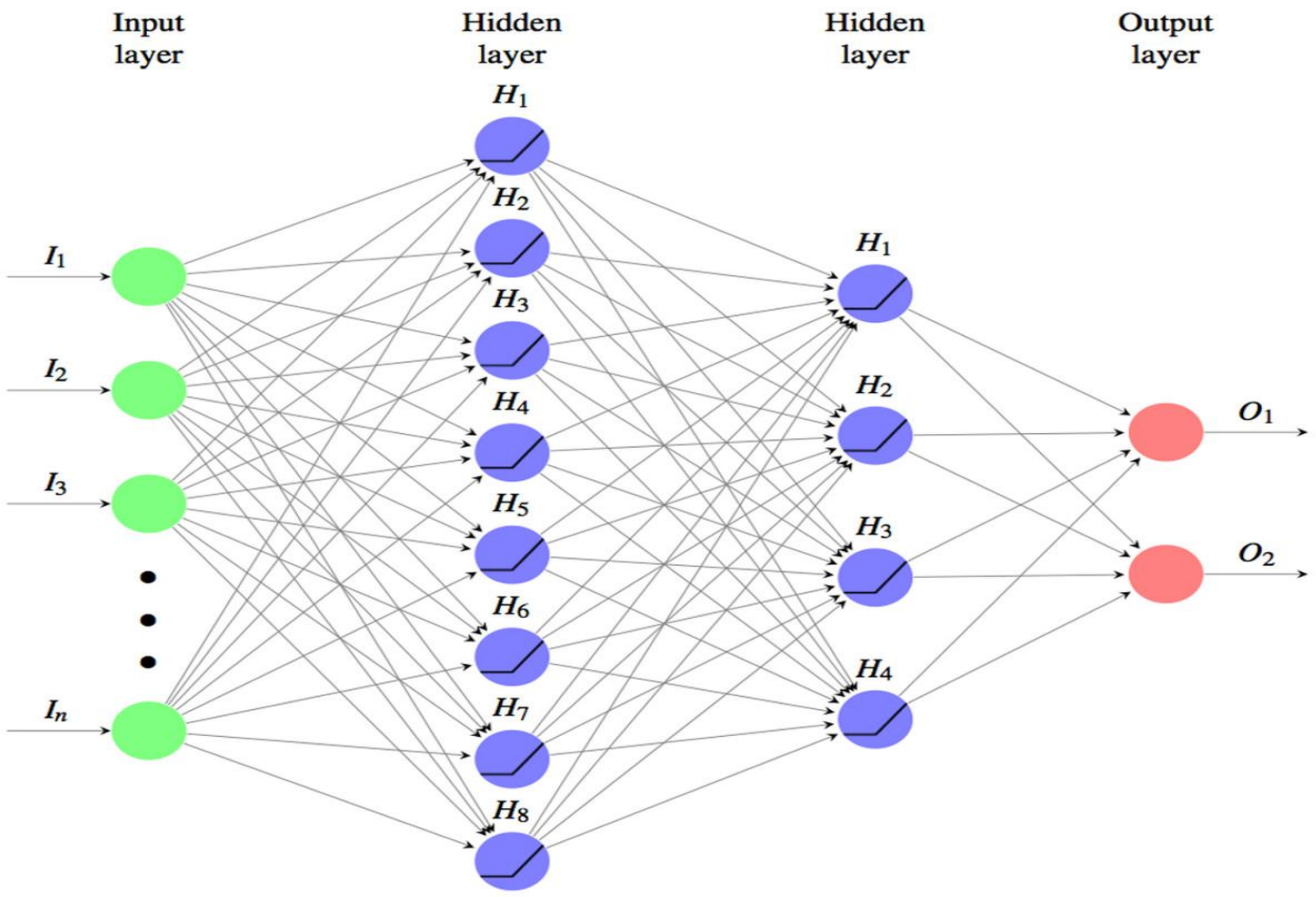
We call “hidden layer” any layer in between the input and the output layers.

- ❖ X: Input Layer
- ❖ Y: Hidden Layer
- ❖ Z: Output Layer



Deep Learning & AI

Artificial Neural Networks (ANNs)



Artificial Neural Networks (ANNs)

We can chain multiple layers, with each output being the input of the next:

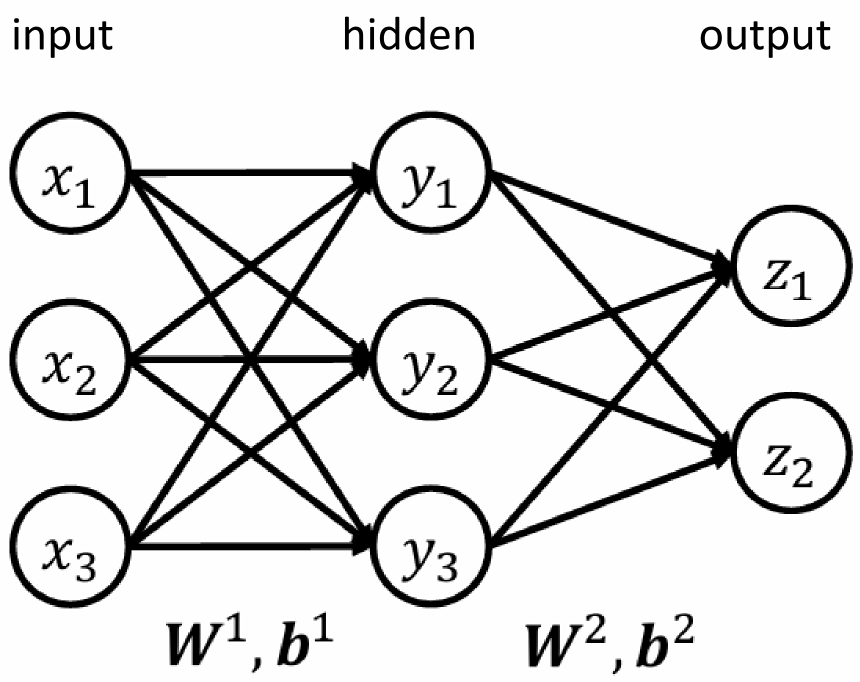
$$y = \sigma(W^1 \cdot x + b^1)$$

$$z = \sigma(W^2 \cdot y + b^2)$$

Combining these leads us to:

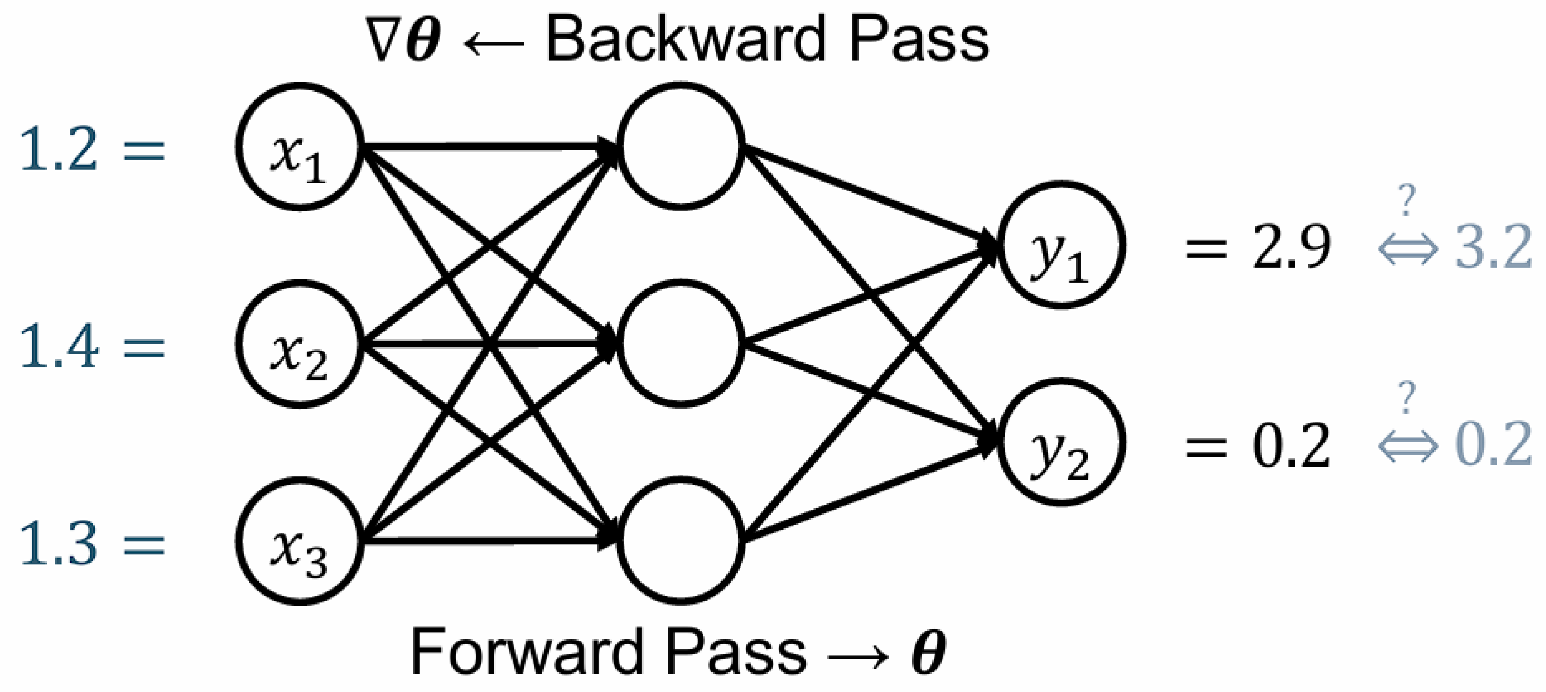
$$z = \sigma(W^2 \cdot \underbrace{\sigma(W^1 \cdot x + b^1)}_{y} + b^2)$$

- Each layer has its own set of parameters (weights $\boldsymbol{W}^i$ and bias $\boldsymbol{b}^i$)
- The underlying computation is a matrix multiplication described by
  $$\boldsymbol{y}^{i+1} = \sigma(\boldsymbol{W}^i \cdot \boldsymbol{y}^i + \boldsymbol{b}^i)$$


input
hidden
output
x_1
x_2
x_3
y_1
y_2
y_3
z_1
z_2
$\boldsymbol{W}^1, \boldsymbol{b}^1$
$\boldsymbol{W}^2, \boldsymbol{b}^2$

Deep Learning & AI

Artificial Neural Networks (ANNs)



$\nabla\boldsymbol{\theta} \leftarrow$ Backward Pass
$1.2 =$
$1.4 =$
$1.3 =$
x_1
x_2
x_3
y_1
y_2
$= 2.9 \overset{?}{\Leftrightarrow} 3.2$
$= 0.2 \overset{?}{\Leftrightarrow} 0.2$
Forward Pass $\rightarrow \boldsymbol{\theta}$

Forward Pass:

- The input values ($x_1 = 1.2, x_2 = 1.4, x_3 = 1.3$) are fed into the network.
- These inputs are processed by the hidden layer(s) through a series of weighted sums followed by activation functions (not shown in detail in the image).
- The resulting values are then passed to the output layer, producing the outputs ($y_1 = 2.9$ and $y_2 = 0.2$).

The forward pass is essentially the computation that happens when the network makes predictions. It starts from the input layer and propagates through the hidden layers and finally to the output layer.

Backward Pass:

- This phase is about learning from errors and updating the weights (θ) in the network.
- The true target values for the outputs are compared with the predicted values from the forward pass (3.2 for y_1 and 0.2 for y_2).
- The differences between the predicted values and true values ($2.9 - 3.2 = -0.3$ for y_1 and $0.2 - 0.2 = 0$ for y_2) represent the errors.
- These errors are then used to calculate the gradients of the loss function with respect to the weights ($\nabla\theta$).

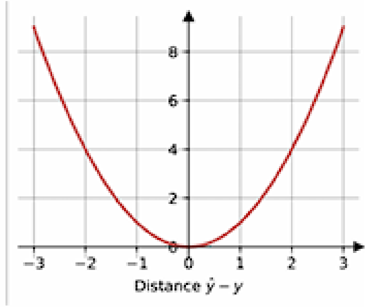
The backward pass involves calculating the gradient of the loss function with respect to each weight in the network (this gradient tells us how to change the weights to decrease the error). Then, the weights are updated typically using an optimization algorithm like gradient descent.

The loss function is a comparison metric between the predicted outputs $\hat{y}_i$ and the expected outputs y_i.

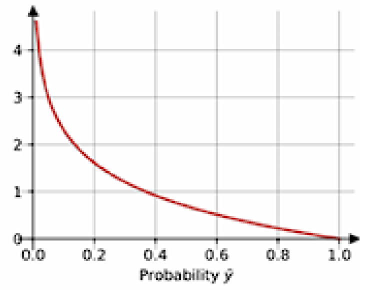
The choice of the loss function usually depends on the type of problem:

- For regression, a common metric is the mean squared error
- For classification, a common metric is the cross entropy

$$MSE(\hat{\mathbf{y}}, \mathbf{y}) = \frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2$$



$$CE(\hat{\mathbf{y}}, \mathbf{y}) = -\sum_{i} y_i \log(\hat{y}_j)$$



Activation Function in Deep Learning

- ReLU
- Leaky ReLU
- ELU
- Swish
- TanH
- Linear
- Sigmoid
- SoftMax

Activation Functions

Why are activation functions important?

- ❖ **Non-Linearity:** Without non-linear activation functions, a neural network would essentially become a linear regression model, incapable of handling the complexities of most real-world data, which often are non-linear.
- ❖ **Control of Signal Flow:** Activation functions decide whether a neuron should be activated or not, effectively allowing the network to make complex decisions and learn from the data.
- ❖ **Learning Complex Patterns:** The introduction of non-linear activation functions allows the neural network to learn complex patterns in the data by stacking layers of neurons, each possibly using different activation functions.
- ❖ **Backpropagation:** Activation functions that are differentiable play a vital role in backpropagation, the training process for neural networks. The derivative of the activation function is used to update the weights of the neurons.

Activation Functions

ReLU (Rectified Linear Unit):

- **Formula:** $\mathrm{ReLU}(x) = \max(0, x)$
- **Description:** ReLU is widely used because it helps mitigate the vanishing gradient problem, allowing models to learn faster and more effectively. It is simple and computationally efficient.

Activation Functions

ReLU (Rectified Linear Unit):

- **Formula:** $\mathrm{ReLU}(x) = \max(0, x)$
- **Description:** ReLU is widely used because it helps mitigate the vanishing gradient problem, allowing models to learn faster and more effectively. It is simple and computationally efficient.

Variants: Due to some of the **limitations** of the standard ReLU, several variants have been proposed:

- **Leaky ReLU:** It allows a small, non-zero, constant gradient ϵ when the unit is not active and x is less than zero ($f(x) = \epsilon x$ for $x < 0$).
- **Parametric ReLU (PReLU):** It generalizes the leaky ReLU by allowing the gradient during the non-active phase to be learned during training.
- **Exponential Linear Unit (ELU):** It also allows for a small gradient when x is negative, but instead of being linear, the negative part is an exponential decay.

Activation Functions

This means that the output is the input itself if the input is greater than zero, and zero if the input is zero or negative. Let's compute the ReLU activation for the following input values: $-3, -1, 0, 2, 4$.

1. For $x = -3$:

$f(-3) = \max(0, -3) = 0$

2. For $x = -1$:

$f(-1) = \max(0, -1) = 0$

3. For $x = 0$:

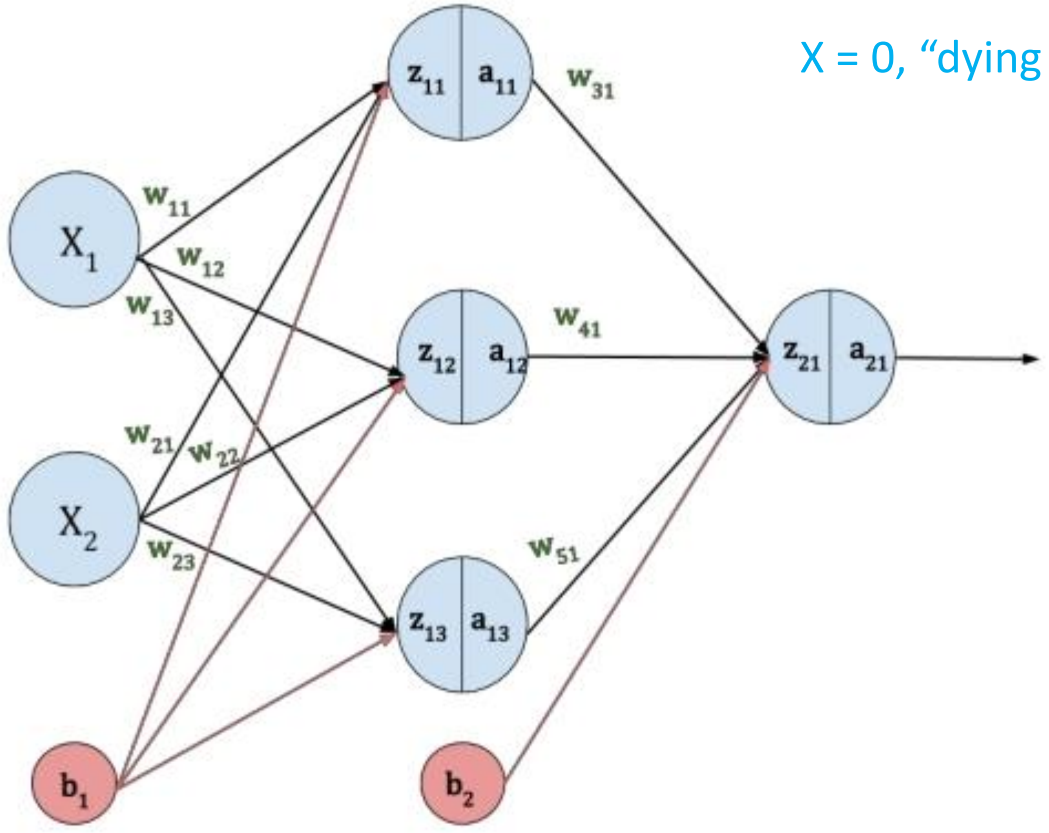
$f(0) = \max(0, 0) = 0$

4. For $x = 2$:

$f(2) = \max(0, 2) = 2$

5. For $x = 4$:

$f(4) = \max(0, 4) = 4$



X = 0, "dying ReLU"

Activation Functions

Leaky ReLU Function:

- **Function:** $f(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha x & \text{if } x \leq 0 \end{cases}$
  - Here, α is a small positive coefficient, such as 0.01.
- **Output:** Positive inputs yield positive outputs (just like ReLU), but negative inputs result in small negative outputs instead of zeros because of the multiplication by the small coefficient α.

For example, if $\alpha = 0.01$ and $x = -5$, then:

- $f(x) = 0.01 \times -5 = -0.05$

Activation Functions

Key Differences

- **Output Range**:
  - **ReLU**: $[0, \infty)$
  - **Leaky ReLU**: $(-\infty, \infty)$, although negative values are very small
- **Gradient**:
  - **ReLU**: The gradient is 1 for positive inputs and 0 for negative inputs.
  - **Leaky ReLU**: The gradient is 1 for positive inputs and α (a small value) for negative inputs, which helps to keep information flowing through the network even when the input is negative.

Activation Functions

ELU (Exponential Linear Unit):

- **Formula:** $\mathrm{ELU}(x) = \begin{cases} x & \text{if } x \geq 0 \\ \alpha(e^{x} - 1) & \text{if } x < 0 \end{cases}$
- **Description:** ELU is similar to ReLU but includes a nonzero gradient for negative values, which helps reduce the vanishing gradient effect, leading to better performance and faster learning.

ELU: Outputs a smooth, non-linear response for negative inputs, with the output curving towards $-\alpha$ as x decreases.

Leaky ReLU: Provides a simple, linear response for negative inputs, proportionally scaled down by α.

Activation Functions

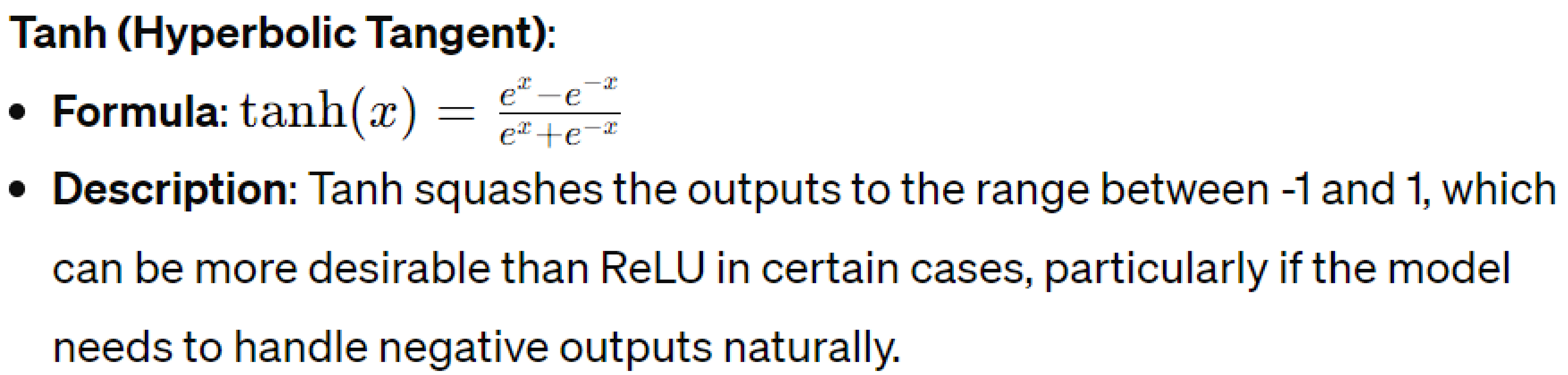
Swish:

- **Formula:** $\text{Swish}(x) = x \cdot \sigma(x)$ $\longrightarrow$ $f(x) = x \cdot \frac{1}{1+e^{-x}}$
- **Description:** A newer activation function that combines aspects of ReLU and sigmoid functions. It has been found to sometimes outperform ReLU in deeper networks.

Swish allows small negative values when the input is negative, it can maintain activation dynamics across a broader range of values, potentially capturing more complex patterns in the data.

Swish is computationally more expensive than ‘ReLU’ due to the calculation of the sigmoid function. This might be a consideration when designing systems where computational efficiency is critical.

Activation Functions



Tanh (Hyperbolic Tangent):

- **Formula:** $\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$
- **Description:** Tanh squashes the outputs to the range between -1 and 1, which can be more desirable than ReLU in certain cases, particularly if the model needs to handle negative outputs naturally.

- **Usage in Neural Networks:** It was particularly popular in the hidden layers of traditional neural networks and is still widely used in the **gates of LSTM** (Long Short-Term Memory) and **GRU** (Gated Recurrent Unit) cells in recurrent neural networks, where the regulation of information flow benefits from the symmetric properties of the function.

- **Disadvantages:** Vanishing Gradients Problem & More Computational Complexity than ReLU.

Activation Functions

Sigmoid: The sigmoid function, also known as the logistic function, is a classical activation function used in neural networks, particularly in the output layer of binary classification models.

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

Disadvantages: Vanishing Gradient Problem, Non-zero-centered Output, Computational Complexity is High.

Activation Functions

Softmax Function: The Softmax function is defined as follows for a vector **z** of raw class scores from the final layer of a model:

$$\text{softmax}(\mathbf{z})_i = \frac{e^{z_i}}{\sum_j e^{z_j}}$$

where:

- z_i is the score for class i,
- e^{z_i} is the exponential function applied to z_i,
- the denominator is the sum of exponential scores for all classes in the vector $\mathbf{z}$.

Activation Functions

$$\text{softmax}(\mathbf{z})_i = \frac{e^{z_i}}{\sum_j e^{z_j}}$$

Let’s assume-

- Score for 1st Class: 2.0
- Score for 2nd Class: 1.0
- Score for 3rd Class: 0.5

- $e^{2.0} \approx 7.389$
- $e^{1.0} \approx 2.718$
- $e^{0.5} \approx 1.649$

Now, computing the probabilities:

- **Probability for Class 1:** $\frac{7.389}{11.756} \approx 0.628$
- **Probability for Class 2:** $\frac{2.718}{11.756} \approx 0.231$
- **Probability for Class 3:** $\frac{1.649}{11.756} \approx 0.140$

These probabilities sum to approximately 1.0, as expected.

Sum of exponentiated scores: $7.389 + 2.718 + 1.649 \approx 11.756$

Activation Functions

In regression problems where the goal is to predict a **continuous output** value, the activation function used in the output layer is typically a **linear activation function** or no activation function at all. The equation for the linear activation function, commonly used in the output layer of regression problems, is simply:

$$f(x) = x$$

Properties:

- **Input**: x (input value)
- **Output**: x (output is the same as the input)

This signifies that the output of the activation function is directly equal to the input value (x). It doesn't introduce any modification to the weighted sum of the inputs.

Activation Functions

Activation Function	Formula	Typical Usage
ReLU	$\mathrm{ReLU}(x) = \max(0, x)$	Hidden layers (common for general purposes)
Sigmoid	$\sigma(x) = \frac{1}{1+e^{-x}}$	Output layer (binary classification)
Tanh	$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$	Hidden layers (range between -1 and 1)
Softmax	$\mathrm{softmax}(x_i) = \frac{e^{x_i}}{\sum_j e^{x_j}}$	Output layer (multi-class classification)
Leaky ReLU	$\mathrm{LeakyReLU}(x) = \max(\alpha x, x)$	Hidden layers (variation of ReLU)

Thank you!