Fifteenth Session

Alireza Moradi

Artificial Neural Networks

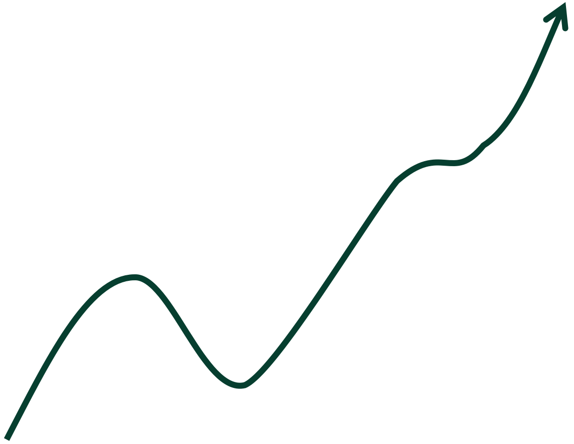
Artificial Neural Networks History

- **Neural networks** became popular in the 1980s.

- Then along came **SVMs**, **Random Forests** and **Boosting** in the 1990s, and Neural Networks took a back seat.

- Re-emerged around 2010 as **Deep Learning**. By 2020s very dominant and successful.

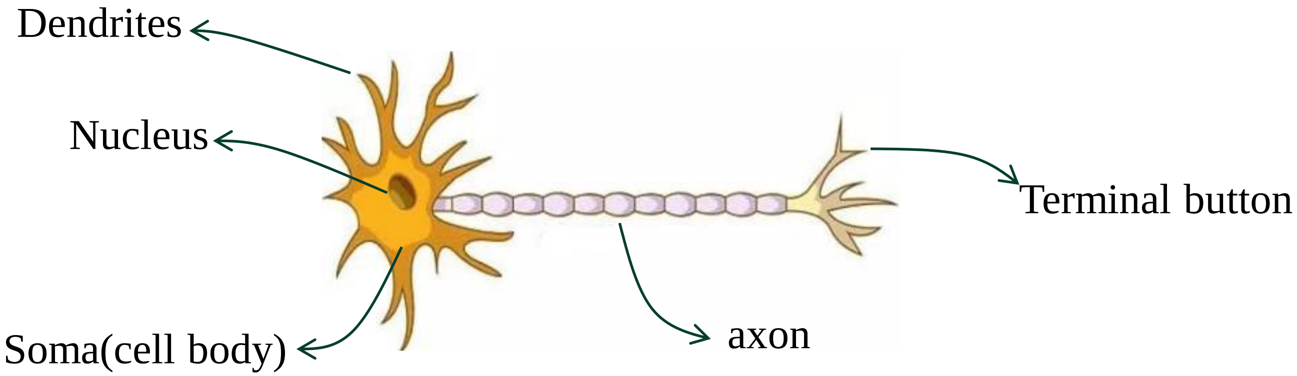
- Part of success due to vast improvements in computing power, larger training sets, and software: TensorFlow and PyTorch.



Artificial Neural Networks (ANNs)

- **Artificial neural networks** are inspired by the structure and function of the biological brain, mimicking how **neurons** process information.

- Our brains are composed of approximately **86 billion neurons**, each connected to about **10,000 other neurons**.

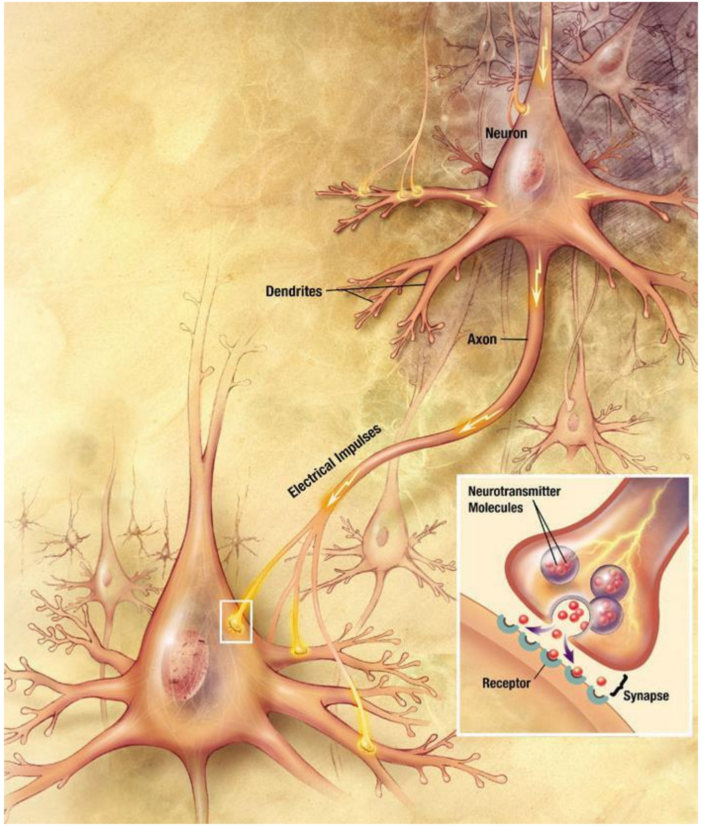


Artificial Neural Networks (ANNs)

- Each neuron receives **electrochemical inputs** from other neurons at their dendrites.

- If these electrical inputs are sufficiently powerful to activate the neuron, then the activated neuron transmits the signal along its axon, passing it along to the dendrites of other neurons.

- These attached neurons may also fire, thus continuing the process of passing the message along.

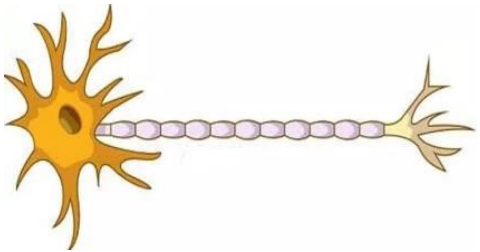


Artificial neuron

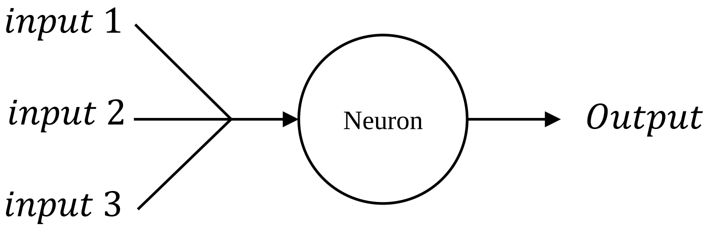
- The key takeaway here is that a **neuron firing** is a binary operation — the neuron either fires or it doesn't fire. There are no different "grades" of firing.

- Simply put, a neuron will only **fire** if the total signal received at the soma exceeds a given threshold.

Biological neuron



Artificial Neuron: Perceptron

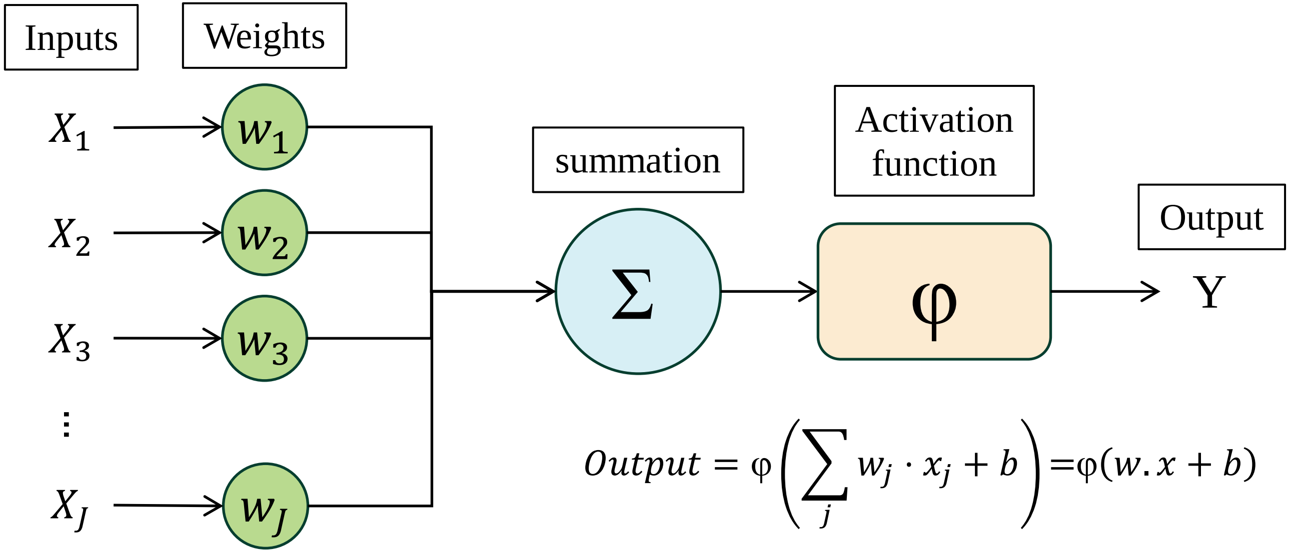


- A **Perceptron** is a simplified mathematical model of a Biological neuron.

Artificial neuron

- An **artificial neuron** or **neural node** is a mathematical model.

- An **artificial neuron** takes inputs, applies weights to these inputs, sums them up, applies a bias, and uses an activation function to produce an output.



$$Output = \varphi\left(\sum_j w_j \cdot x_j + b\right) = \varphi(w.x + b)$$

Artificial neuron

- Recall logistic regression:

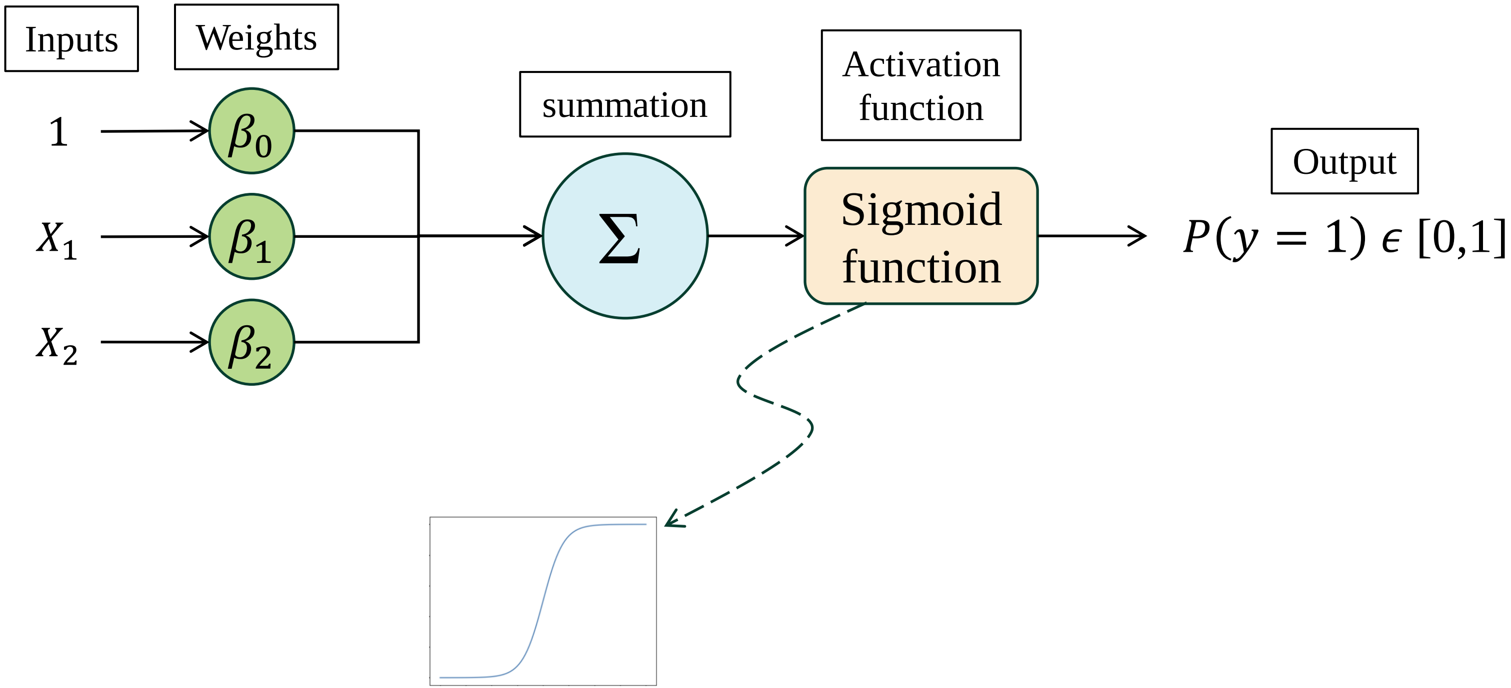
Output Inputs

$$P(X) = \frac{e^{\beta_0+\beta_1.x_1+\beta_2.x_2}}{1 + e^{\beta_0+\beta_1.x_1+\beta_2.x_2}}$$

$z = \beta_0 + \beta_1.x_1 + \beta_2.x_2$ → Sigmoid function → $P(y = 1) \in [0,1]$

Artificial neuron

- Basically, we can think of logistic regression as a one-layer neural network.



A Single Neuron; Example

- Suppose we want to classify handwritten digits as 0 or 1:

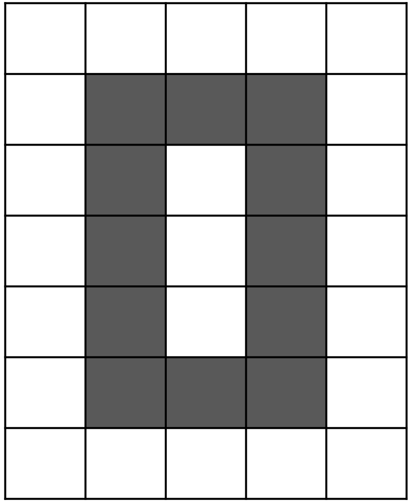


Image of digit 0

0	0	0	0	0
0	**1**	**1**	**1**	0
0	**1**	0	**1**	0
0	**1**	0	**1**	0
0	**1**	0	**1**	0
0	**1**	**1**	**1**	0
0	0	0	0	0

Image representation

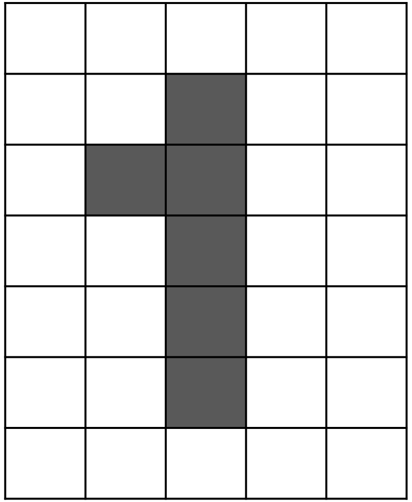


Image of digit 1

0	0	0	0	0
0	0	**1**	0	0
0	**1**	**1**	0	0
0	0	**1**	0	0
0	0	**1**	0	0
0	0	**1**	0	0
0	0	0	0	0

Image representation

A Single Neuron; Example

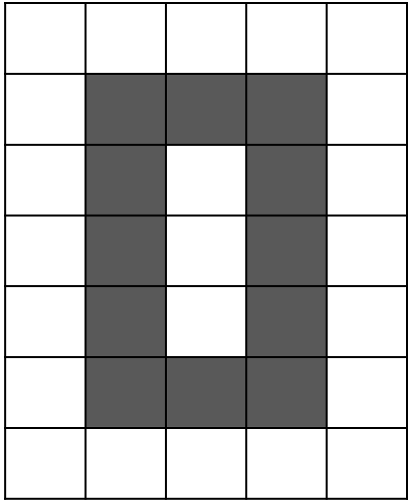


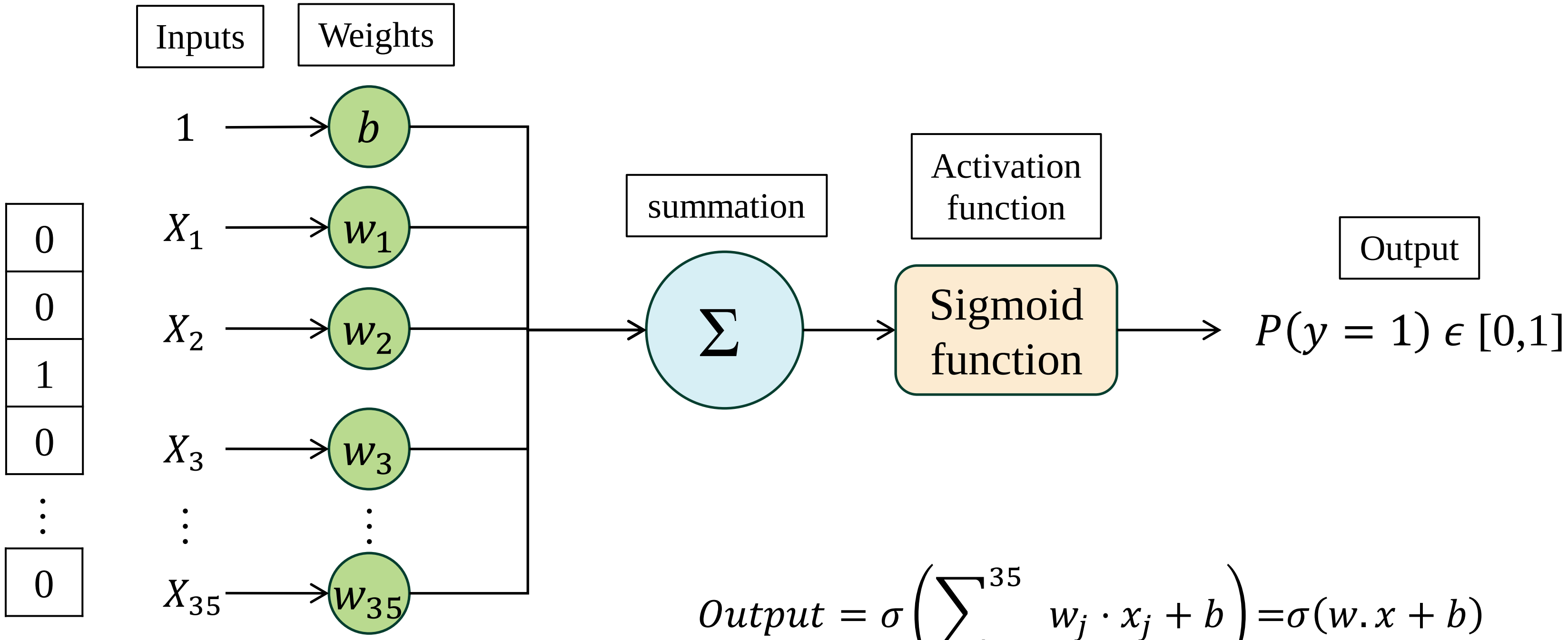
Image of digit 0

0	0	0	0	0
0	**1**	**1**	**1**	0
0	**1**	0	**1**	0
0	**1**	0	**1**	0
0	**1**	0	**1**	0
0	**1**	**1**	**1**	0
0	0	0	0	0

Image representation

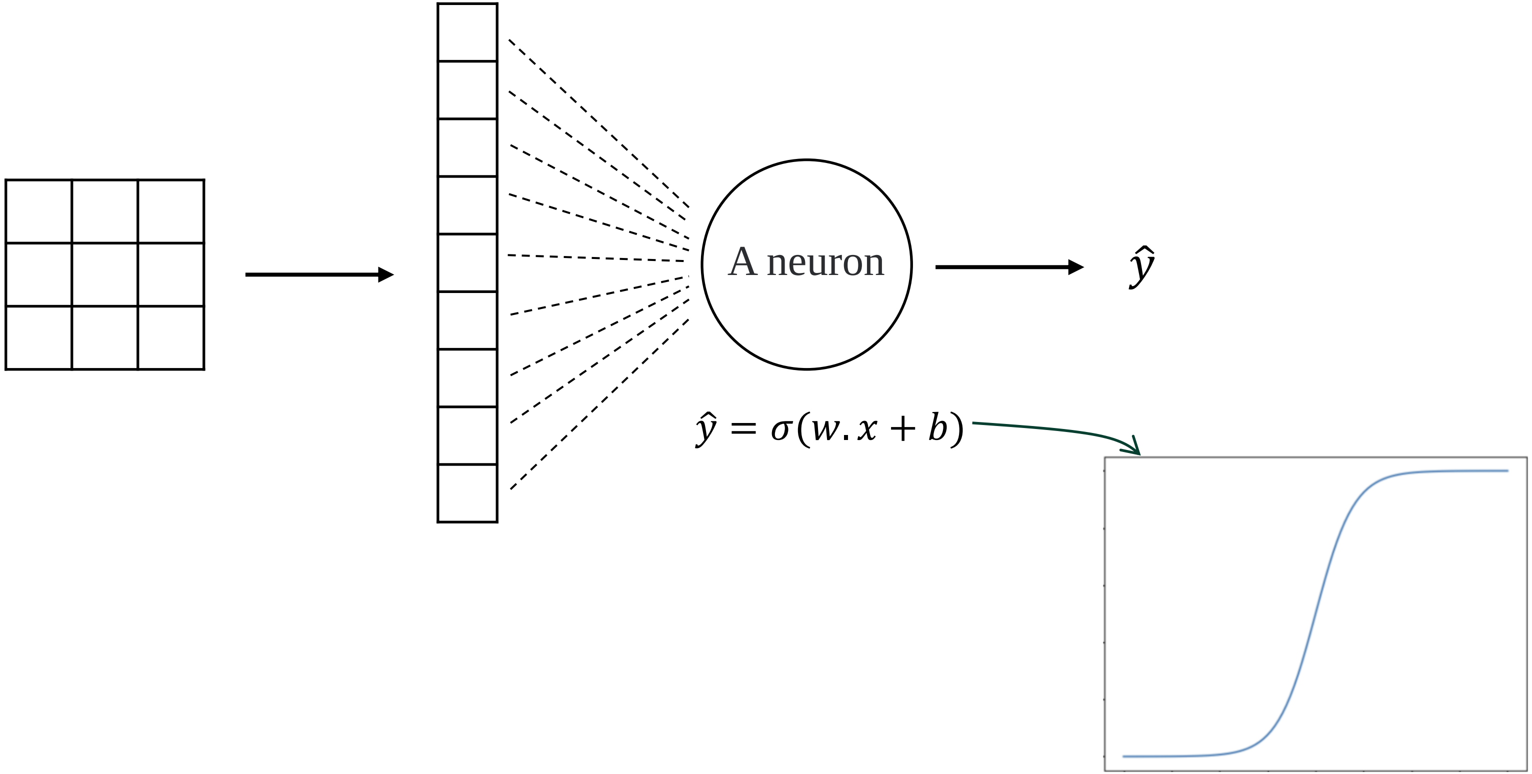
$$\Rightarrow \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 1 \\ 0 \\ 0 \\ 1 \\ \vdots \\ 0 \end{bmatrix}$$

A Single Neuron; Example



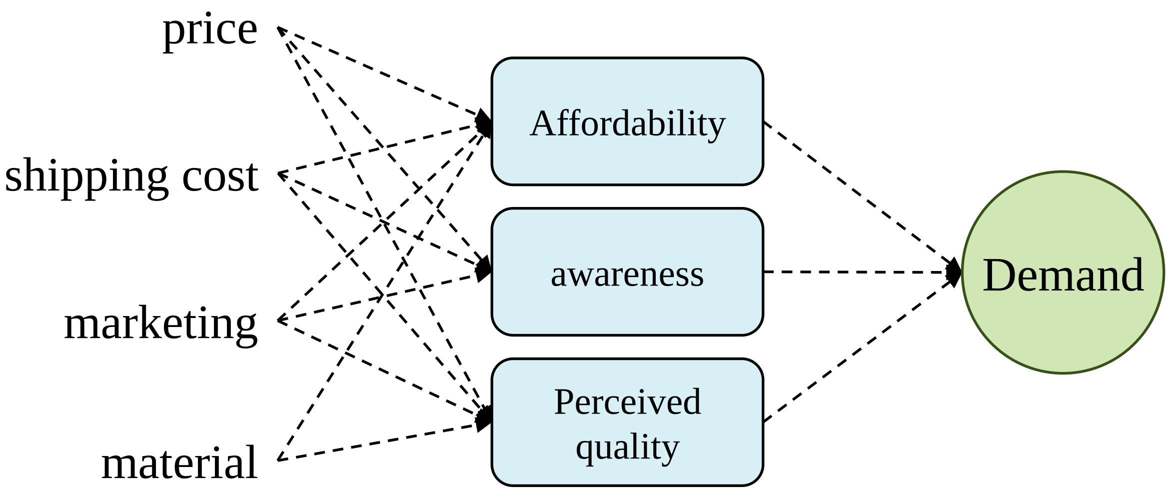
$$Output = \sigma\left(\sum_{j=1}^{35} w_j \cdot x_j + b\right) = \sigma(w.x + b)$$

A Single Neuron; Summary



ANNs; network of neurons

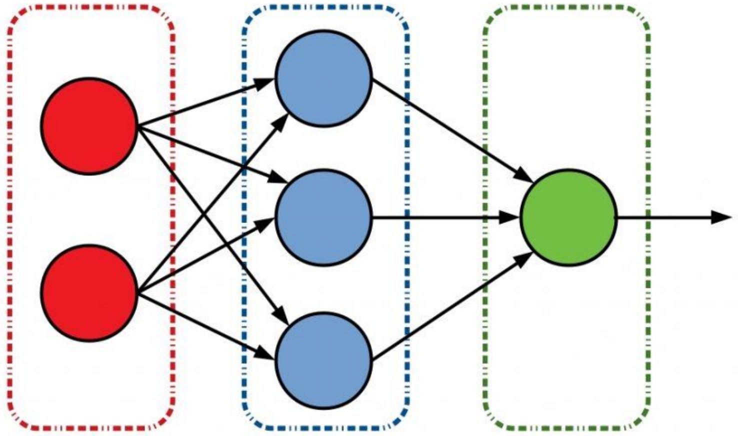
- ✓ Suppose we want to predict demand for different products. How we do this prediction?



ANNs; layers

- ANNs (Artificial Neural Networks) consists of 3 types of **layers of nodes**, or artificial neurons:

1. **Input layer**: Data enters through the input layer.
2. **Hidden layers**: Hidden layers process and transport data to other layers.
3. **Output layer**: The result or prediction is made in the output layer.

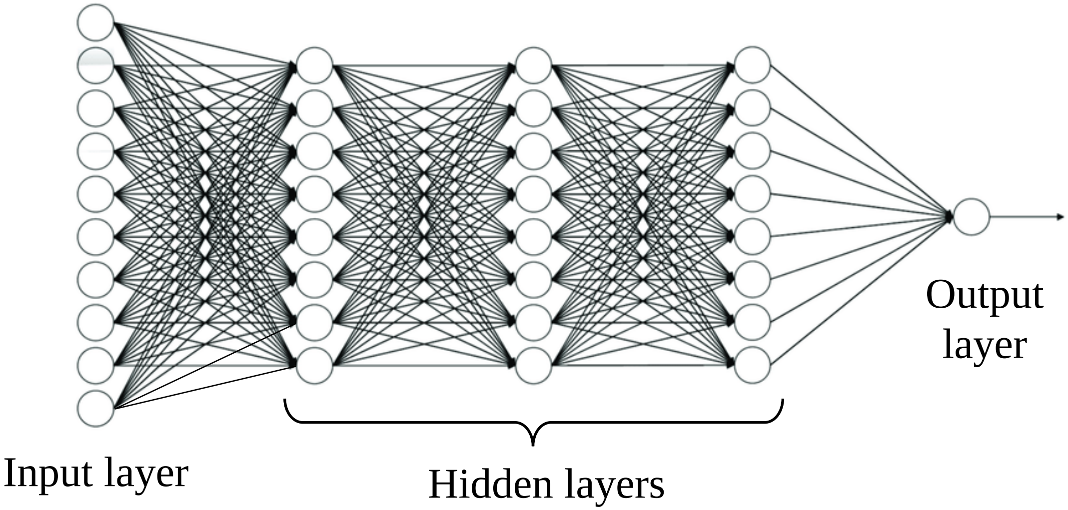


- Hidden Layers can be one or more.

Multilayer Perceptron

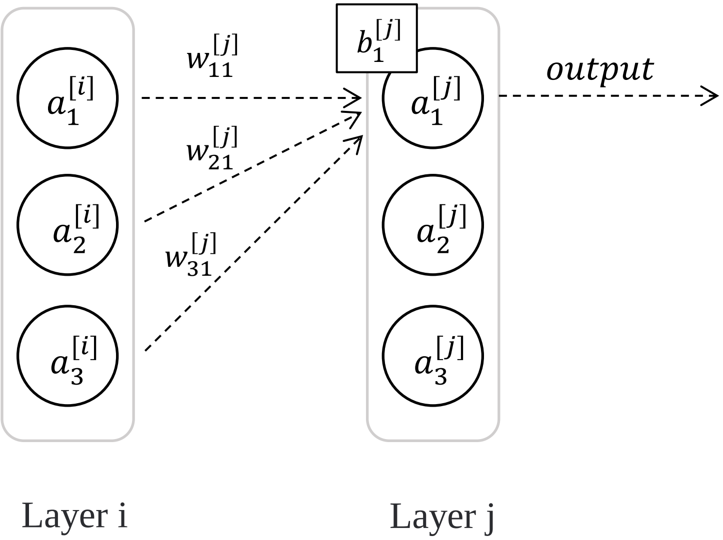
- A fully connected multi-layer neural network is called a **Multilayer Perceptron (MLP)**.

- Recall that the **perceptron** is a mathematical model of a biological neuron.



ANNs; Notation

$$\boxed{w, b, z, g}$$



$$z_1^{[j]} = \sum_k w_{k1}^{[j]} . a_k^{[i]} + b_1^{[j]} = w_1^{[j]} . a^{[i]} + b_1^{[j]}$$

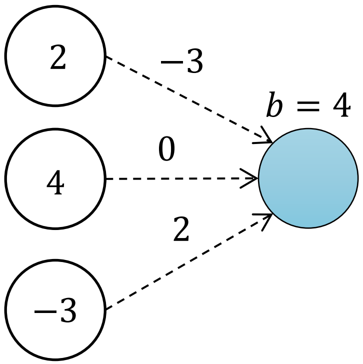
$$z_1^{[j]} = w_{11}^{[j]} . a_1^{[i]} + w_{21}^{[j]} . a_2^{[i]} + w_{31}^{[j]} . a_3^{[i]} + b_1^{[j]}$$

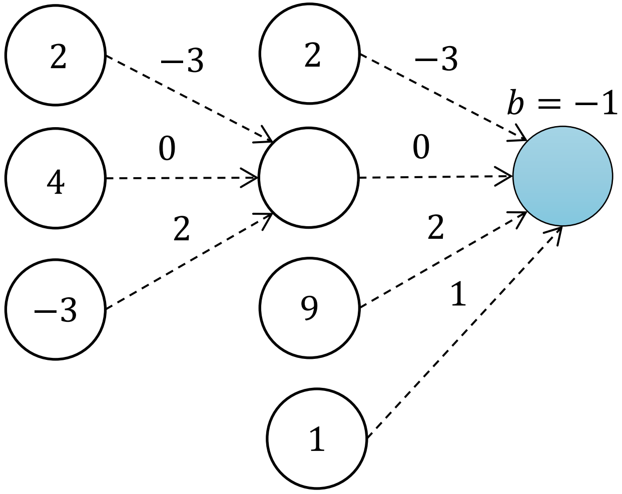
$$a_1^{[j]} = g(z_1^{[j]}) = \sigma(z_1^{[j]})$$

ANNs; Notation

- ✓ Calculate output using sigmoid as g(z):

$$g(z) = \frac{1}{1 + e^{-z}}$$

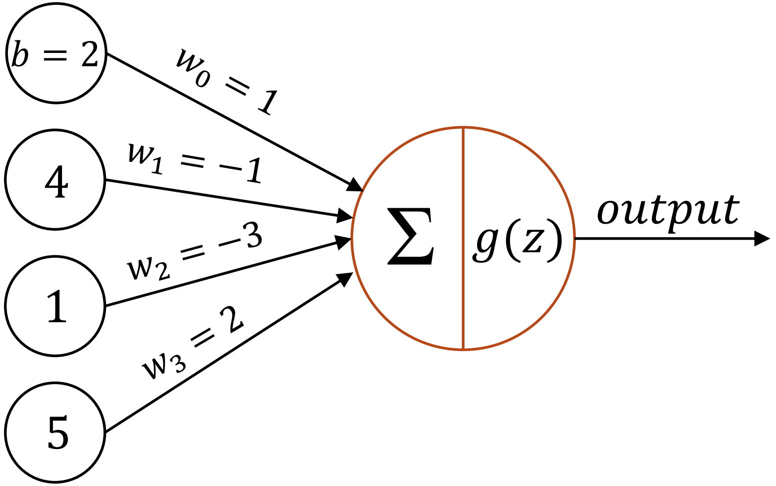




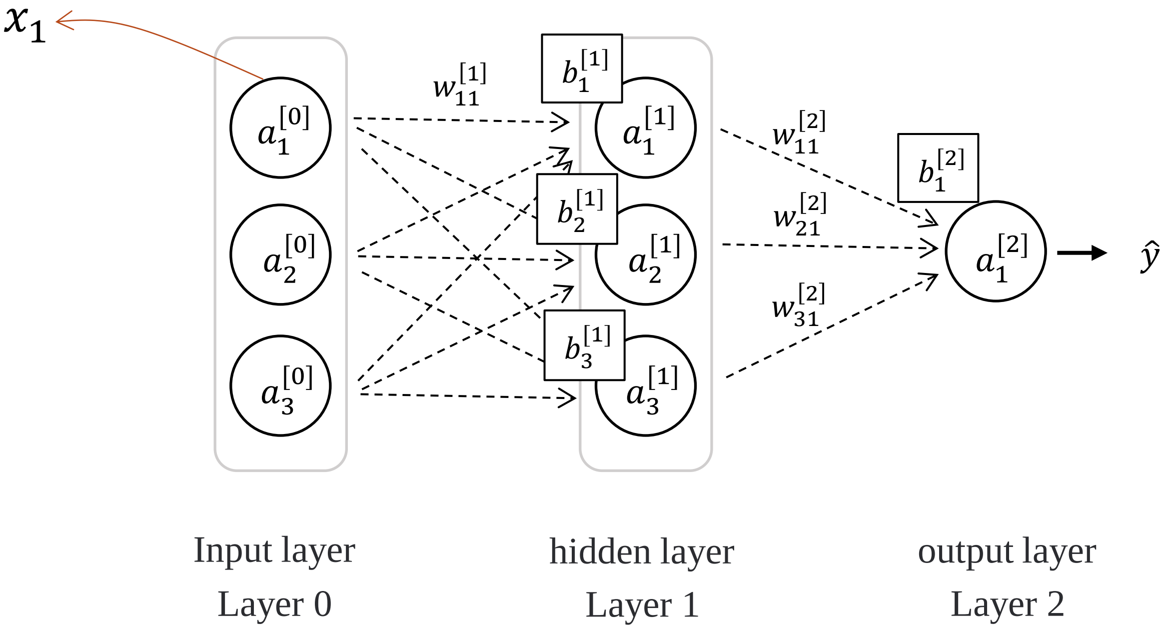
ANNs; Notation

- ✓ Calculate output using sigmoid as g(z):

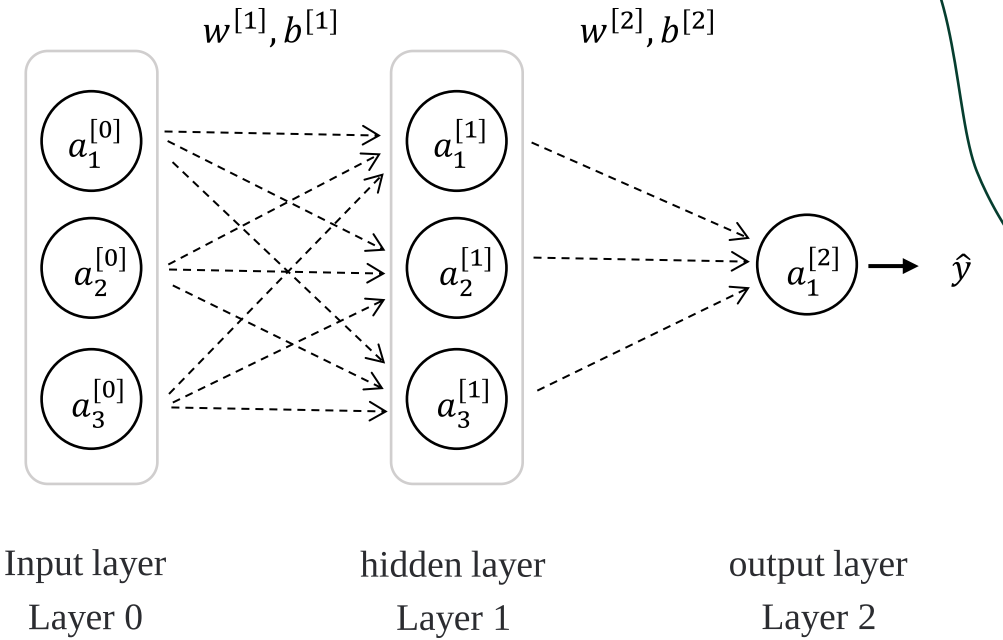
$$g(z) = \frac{1}{1 + e^{-z}}$$



ANNs; Notation



ANNs; Notation

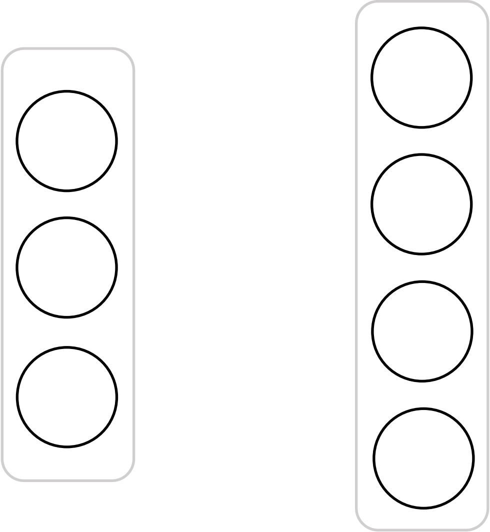


$$w^{[1]} = \begin{bmatrix} w_{11}^{[1]} & w_{21}^{[1]} & w_{31}^{[1]} \\ w_{12}^{[1]} & w_{22}^{[1]} & w_{32}^{[1]} \\ w_{13}^{[1]} & w_{23}^{[1]} & w_{33}^{[1]} \end{bmatrix}$$

$$b^{[1]} = \begin{bmatrix} b_1^{[1]} \\ b_2^{[1]} \\ b_3^{[1]} \end{bmatrix}$$

ANNs; Notation

- ✓ What is the w and b in this setting?

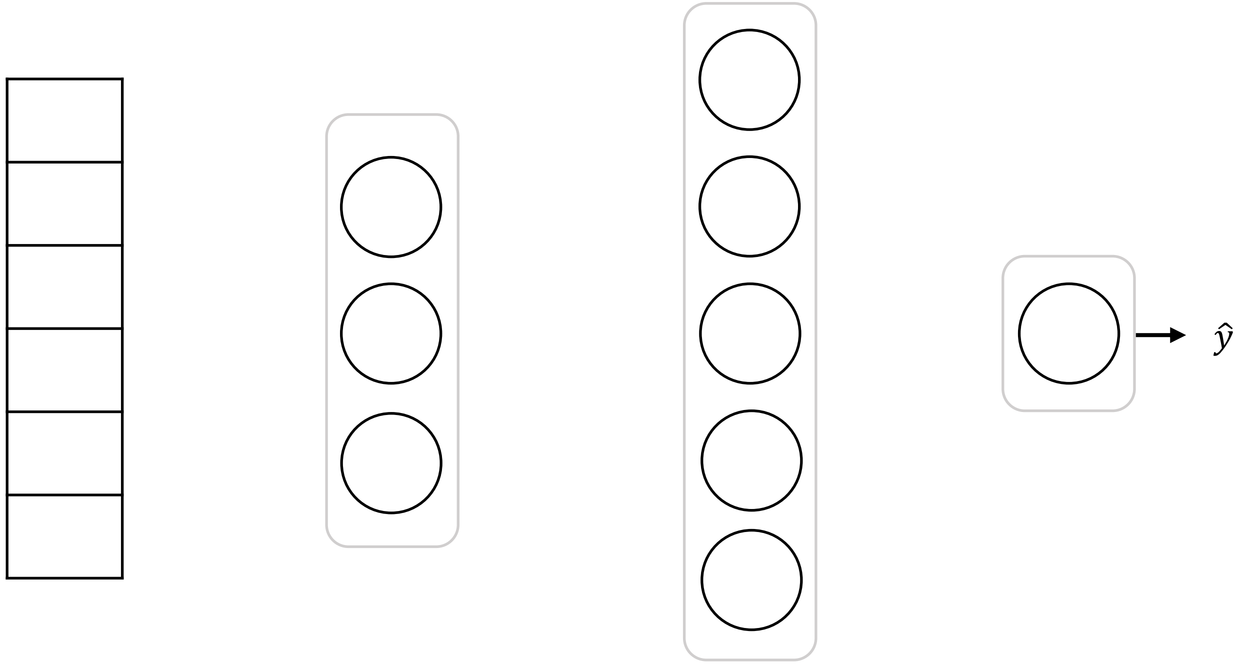


Layer j

$$\begin{cases} w^{[j]} = \begin{bmatrix} \quad \end{bmatrix} \\ b^{[j]} = \begin{bmatrix} \quad \end{bmatrix} \end{cases}$$

ANNs; Notation

- ✓ Example:



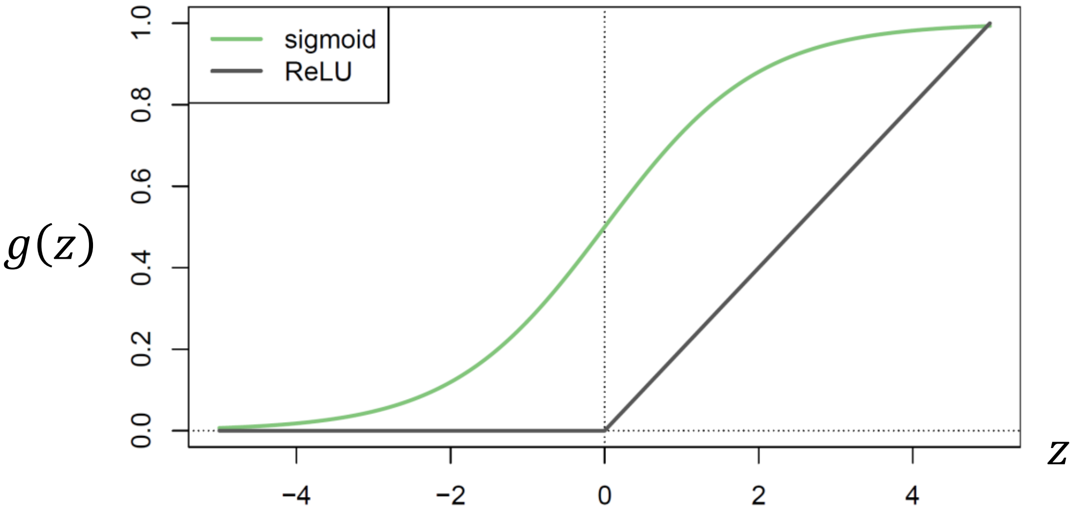
Activation functions

- An **activation function** is a mathematical function that is applied to the inputs of a neuron in a neural network.

- Specifically, $g(z)$ is called the **activation function**. Popular activation functions are the **sigmoid** and rectified linear(**ReLU**), shown in figure.

- **Activation functions** are nonlinear transformations of linear combinations of the inputs.

$$z \Rightarrow g(z) \Rightarrow a$$



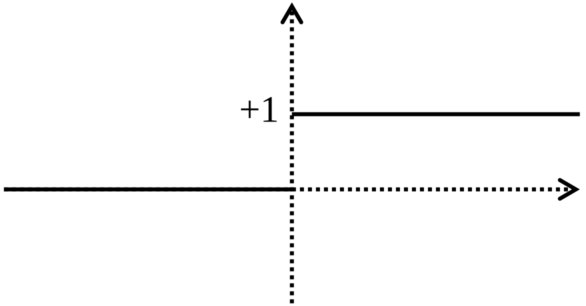
Activation functions

- Activation functions introduce **non-linearity** to the network, allowing it to model complex relationships between inputs and outputs.

- The purpose of an **activation function** is to determine the output or activation level of a neuron based on its input.

- In a simple setting, an activation function decides whether a neuron should be activated or not. (like biological neuron)

- Neural networks leverage various types of activation functions. Each activation function has its own unique properties and is suitable for certain use cases.



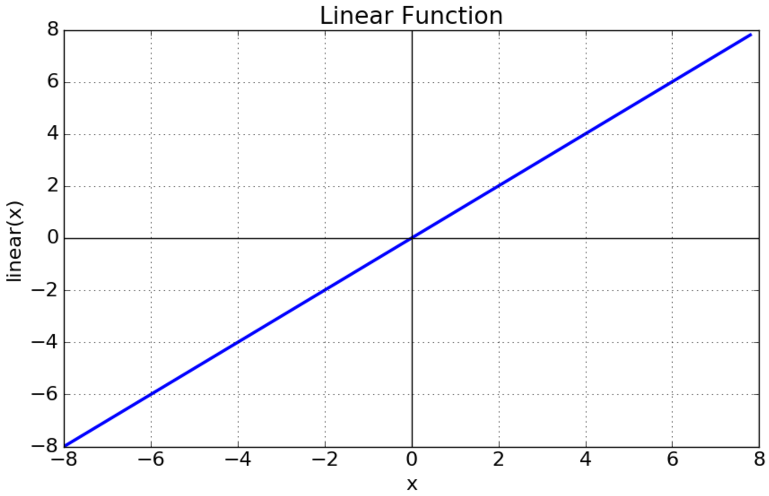
Activation functions; Linear

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

$$g(z) = z$$

- The main use case of the linear activation function is in the output layer of a neural network used for regression.

- However, the linear activation function is rarely used in hidden layers of neural networks, because it does not provide any non-linearity.



Activation functions; Sigmoid

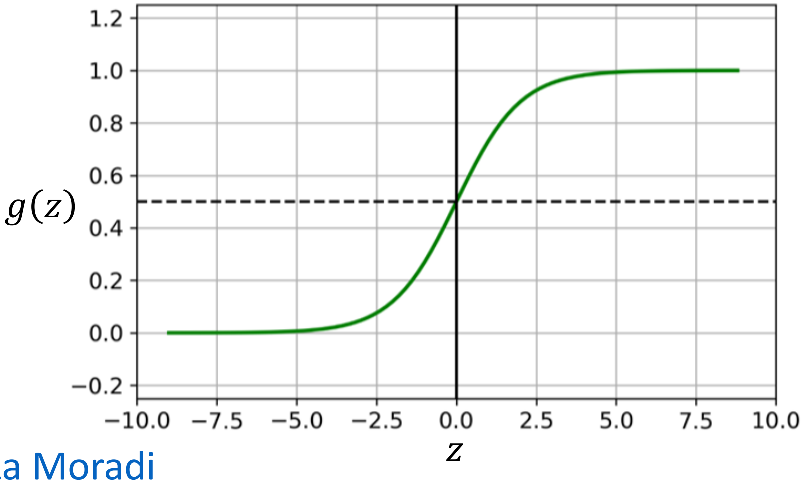
- The **sigmoid** activation function, often represented as $\boldsymbol{\sigma(z)}$, is a smooth, continuously differentiable function.

- The sigmoid activation function has the mathematical form:

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

- The outputs can be easily interpreted as probabilities, which makes it natural for binary classification problems.

- It is also used in hidden layers.



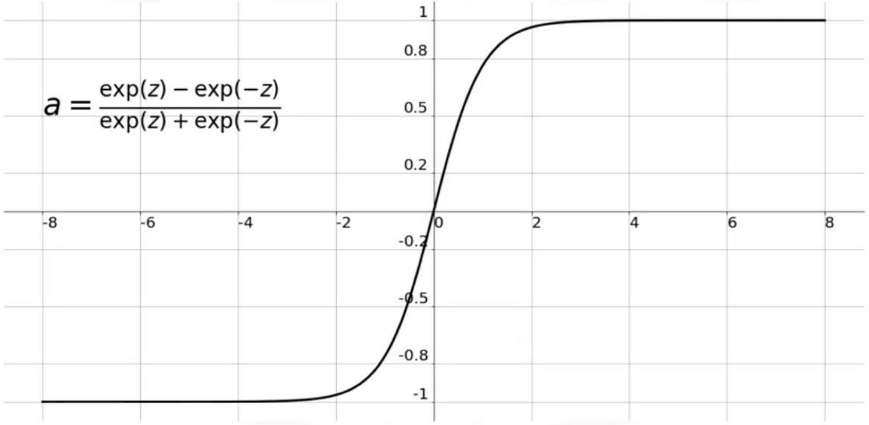
Activation functions; Tanh

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

$$g(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$$

- The tanh function is frequently used in the hidden layers of a neural network.

✓ What is the difference between Tanh and sigmoid?



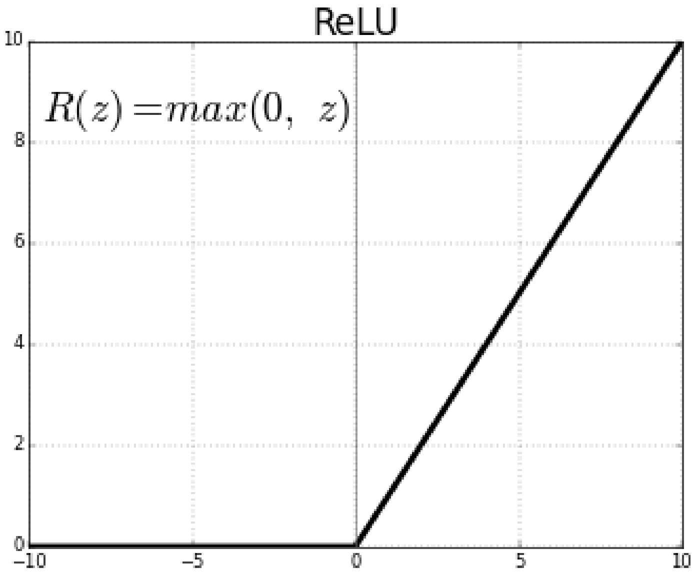
Activation functions; ReLU

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

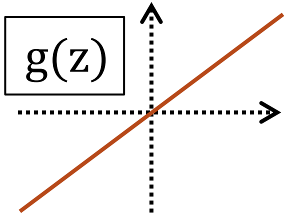
$$g(z) = \max(z, 0)$$

- The **ReLU** is the most frequent activation function used in hidden layers.

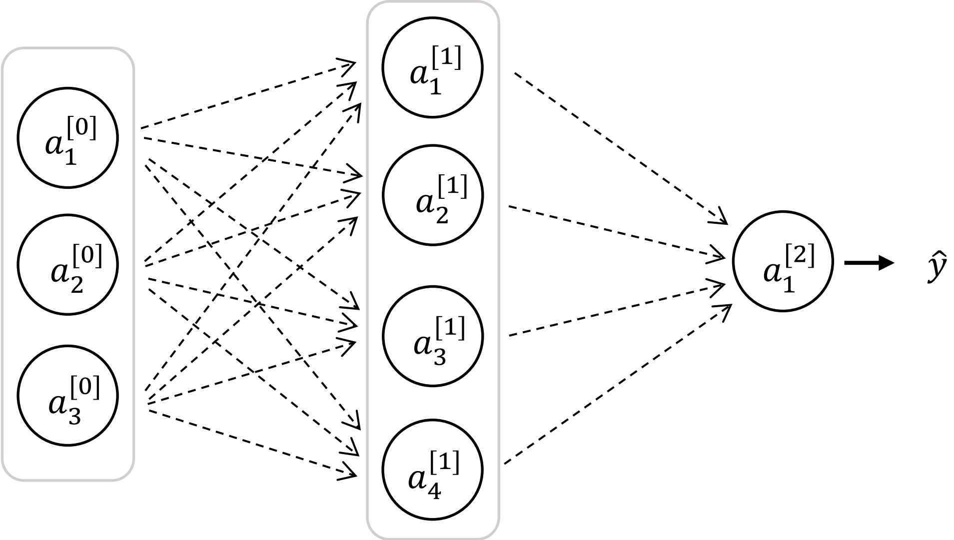
- The **gradient** of ReLU is either zero (for negative inputs) or one (for positive inputs), making it easier for the gradient to flow during backpropagation and enabling more efficient training.



Activation functions



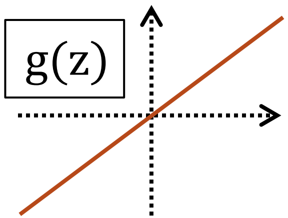
✓ What happens if we use all activation functions linear?

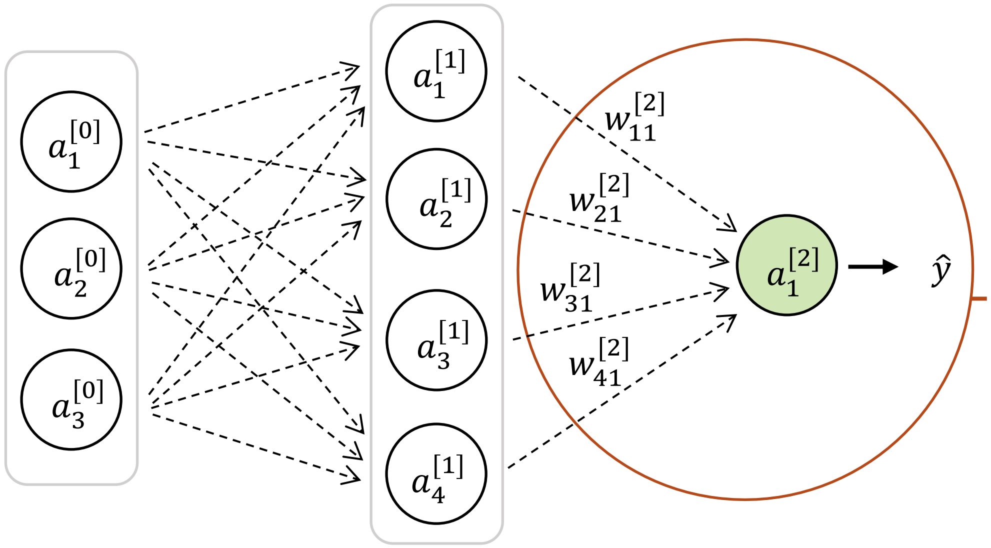


- Let's test this: [link]

Activation functions

✓ What happens if we use all activation functions linear?

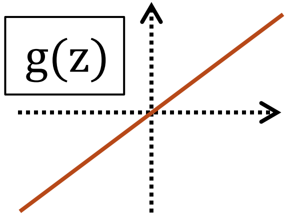


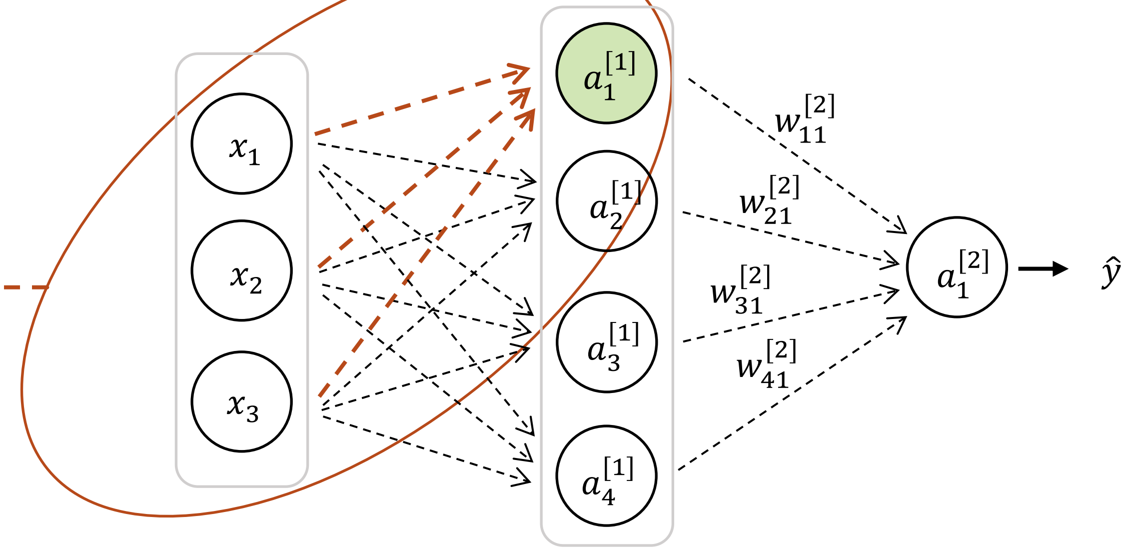


$$Output = a_1^{[2]} = g\left(w^{[2]}.a^{[1]}\right) = \sum_{j=1}^{4} w_{j1}^{[2]}.a_j^{[1]}$$

Activation functions

✓ What happens if we use all activation functions linear?

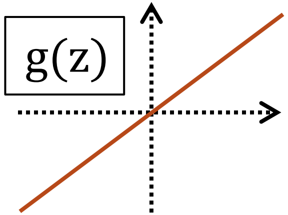


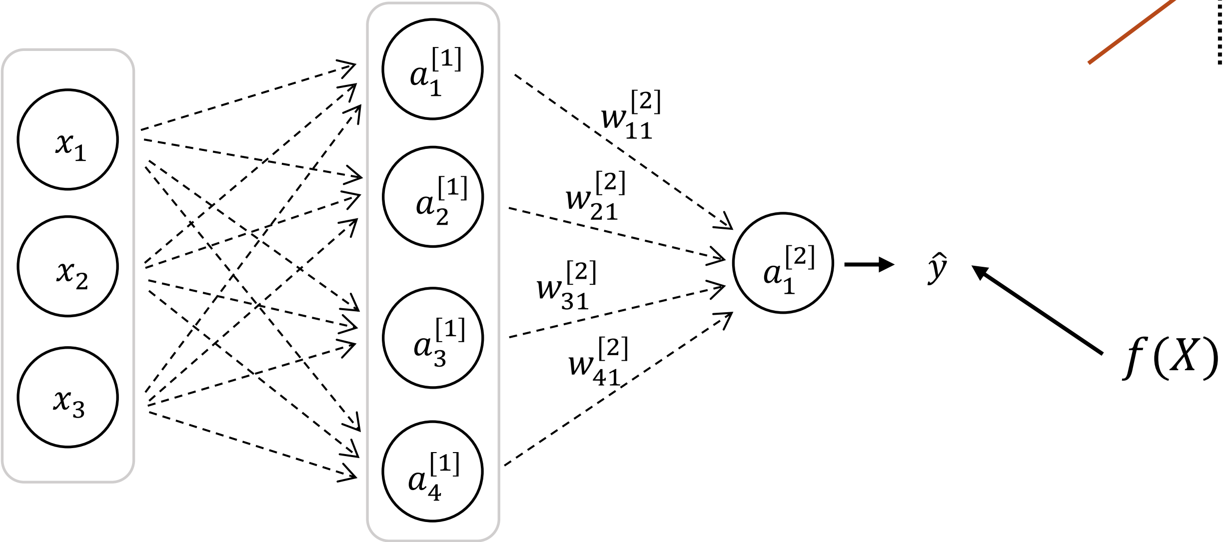


$$a_1^{[1]} = \mathrm{g}\left(w_1^{[1]}.X\right) = \sum_{i=1}^{3} w_{i1}^{[1]}.x_i$$

Activation functions

✓ What happens if we use all activation functions linear?



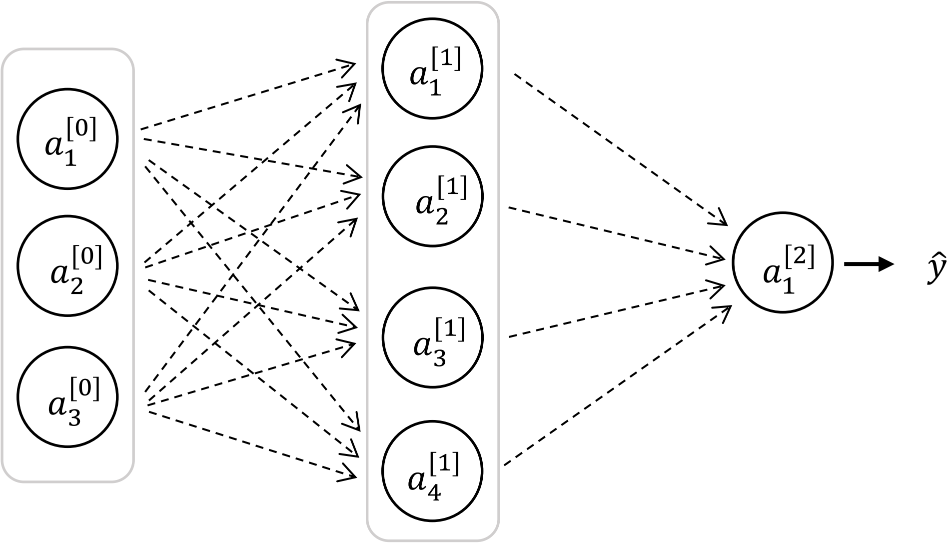


Linear model!

$$f(X) = a_1^{[2]} = \sum_{j=1}^{4} w_j^{[2]} . (\sum_{i=1}^{3} w_{ij}^{[1]} . x_i) = \boxed{w'_1 . x_1 + w'_2 . x_2 + w'_3 . x_3}$$

Activation functions

✓ What happens if we use all activation functions linear?



- **Activation functions** in hidden layers are typically **nonlinear**, otherwise the model collapses to a linear model.