Université Constantine 2
جامعة قسنطينة 2

Module : Machine Learning (ML – SDSI)

– Course 4 –

Chapter 4 : Neural Networks

Habib-Ellah GUERGOUR
Faculty of NTIC / TLSI Department
Contact: habib.guergour@univ-constantine2.dz

Université Constantine 2
جامعة قسنطينة 2

Module : Machine Learning (ML – SDSI)

– Course 4 –

Chapter 4 : Neural Networks

Habib-Ellah GUERGOUR
Faculty of NTIC / TLSI Department
Contact: habib.guergour@univ-constantine2.dz

Etudiants concernés

Faculté/Institut	Département	Niveau	Spécialité
NTIC	TLSI	M1	SDSI

Goals of the Chapter

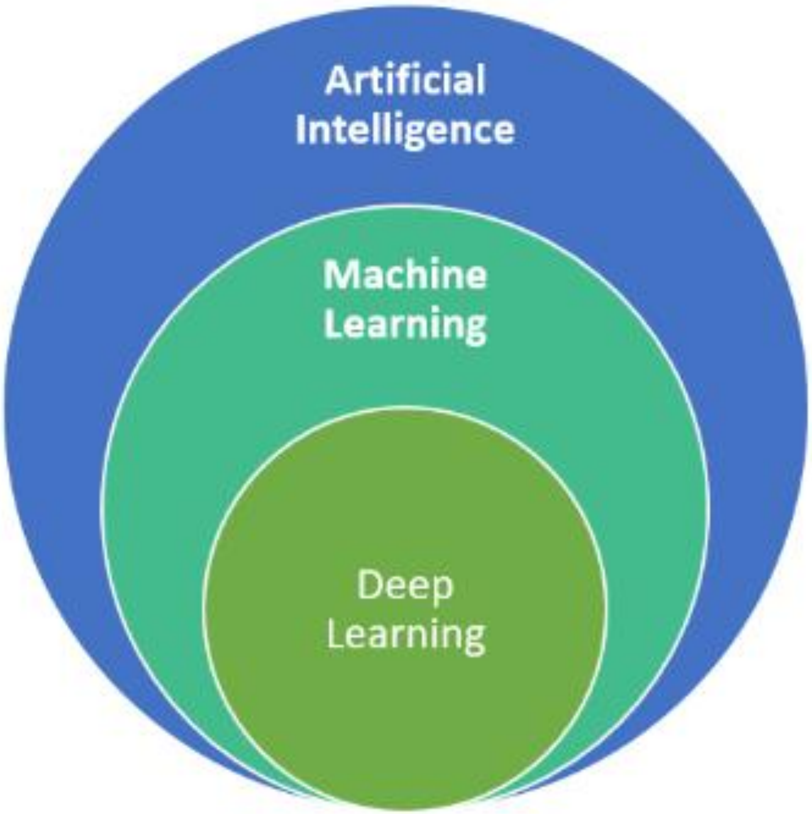
- Introduce the **basic concepts** of neural networks.
- Understand **neuron** structure, **layers**, and **activations**.
- Learn the training process: **forward pass**, **loss computation**, **backpropagation**.
- Connect neural networks to the foundations of **deep learning.**

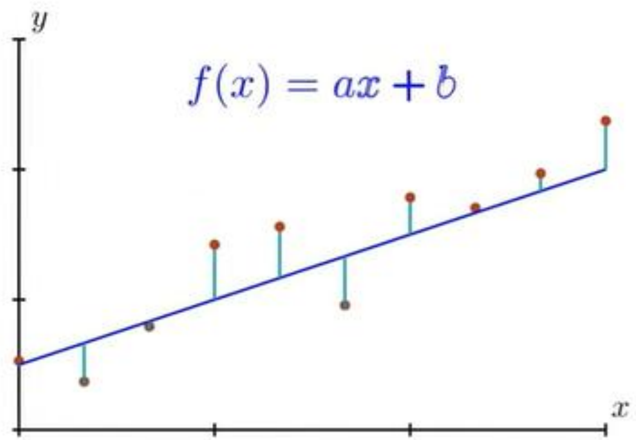
Main Titles

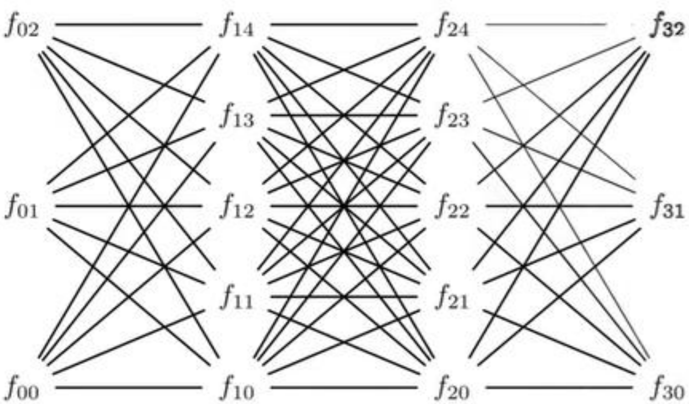
- Introduction
- Biological Roots to Deep Learning
- How to build a neural network
- Forward Propagation & Activation Functions
- Backpropagation: The Core of Learning

Introduction

Machine Learning vs Deep Learning

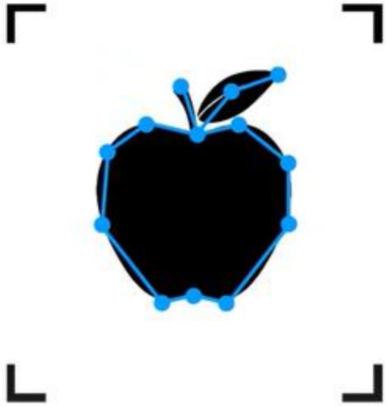


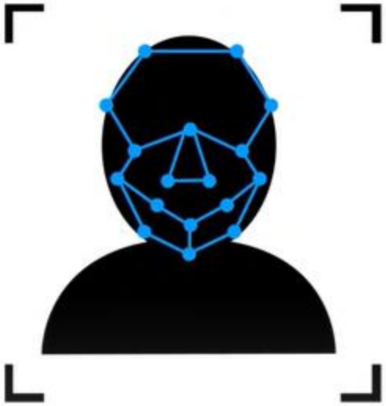




Introduction

Applications

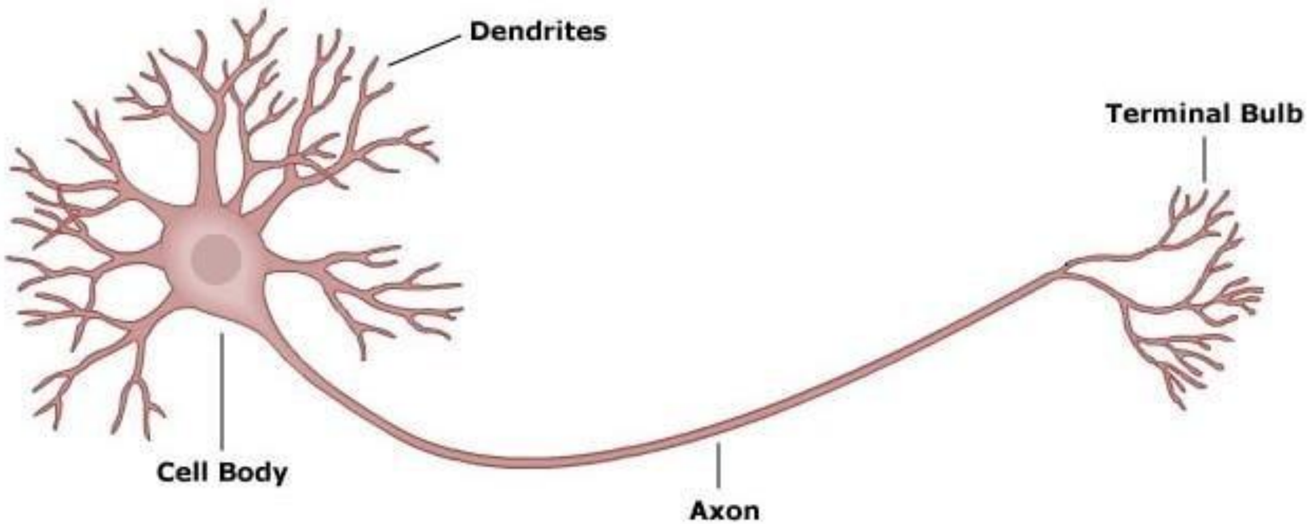




Biological Roots to Deep Learning

Biological Inspiration

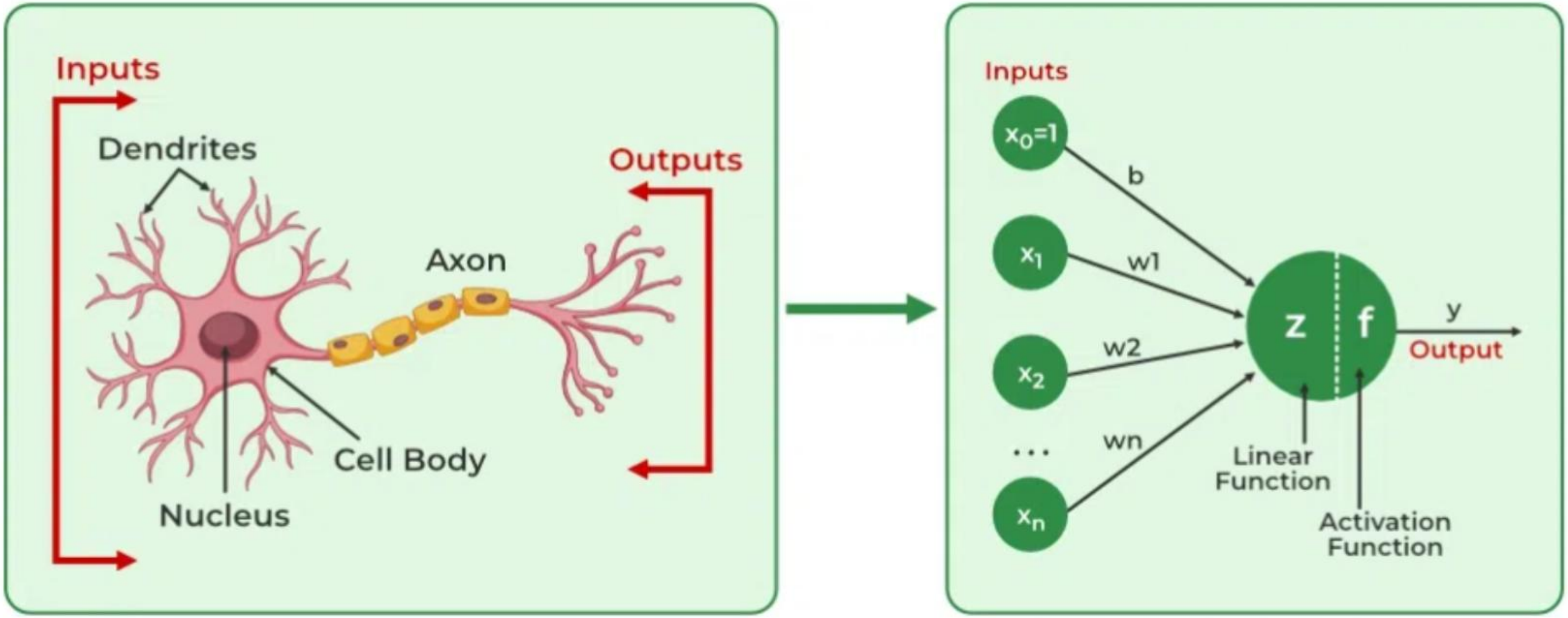
- Neurons are excitable cells connected to each other and whose role is to transmit information in our nervous system



Biological Roots to Deep Learning

From Neurons to Deep Learning

- Brain neurons → Artificial neurons (McCulloch-Pitts, 1943).
  - Binary threshold: Fires if weighted sum ≥ threshold.
  - No learning—just fixed logic.



The Perceptron and Its Limitations

Rosenblatt's Perceptron (1958):

- Learns weights from data (unlike fixed McCulloch-Pitts).
- Rule: Adjust weights to minimize classification errors.

Formula:

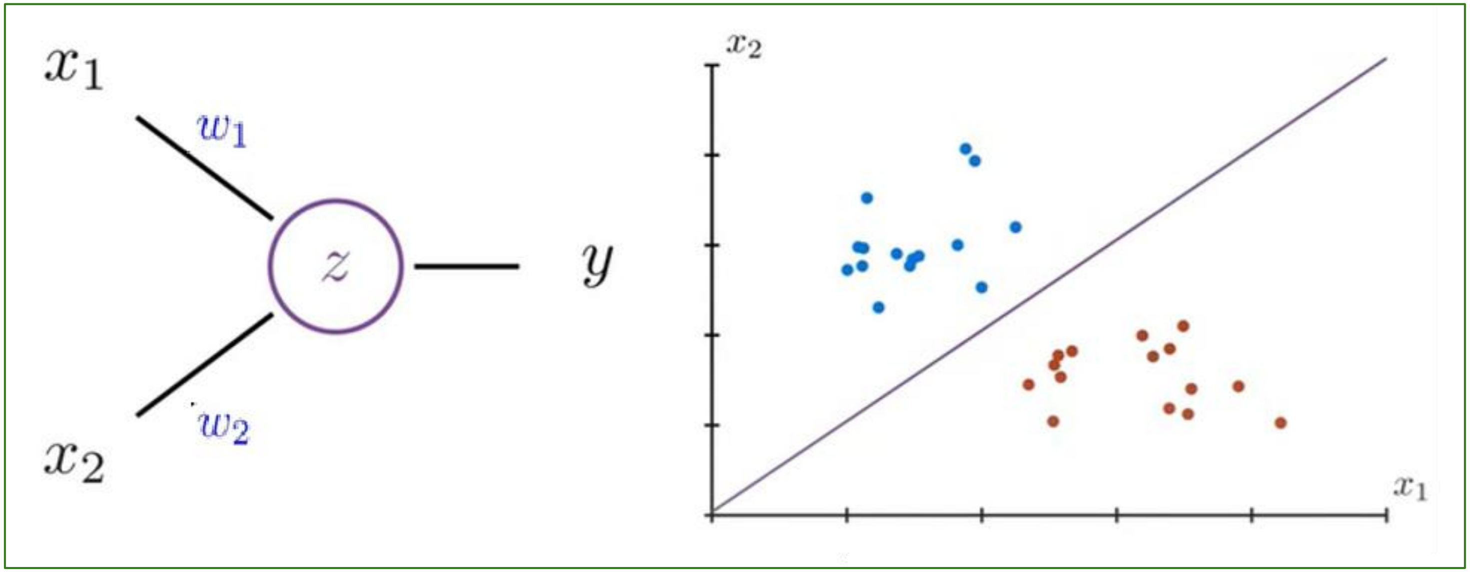
$$y = \begin{cases} 1 & \text{if } \sum w_i x_i + b \geq 0, \\ 0 & \text{otherwise.} \end{cases}$$

- Limitation: Only works for linearly separable data (XOR problem).

The Perceptron and Its Limitations

Rosenblatt's Perceptron (1958):

- Example : The model preforms well when the data is linearly separable.

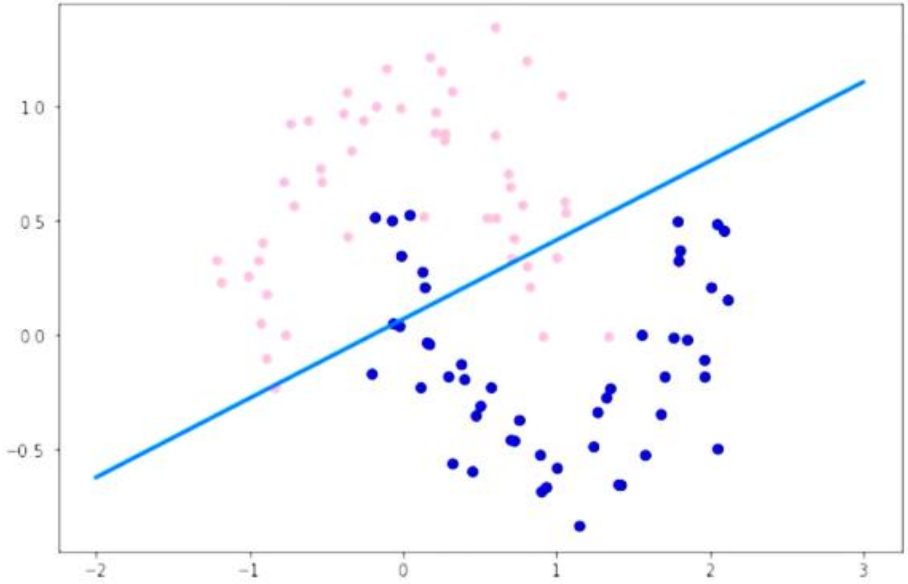


$$z(x_1, x_2) = w_1 x_1 + w_2 x_2 + b$$

The Perceptron and Its Limitations

Rosenblatt’s Perceptron (1958):

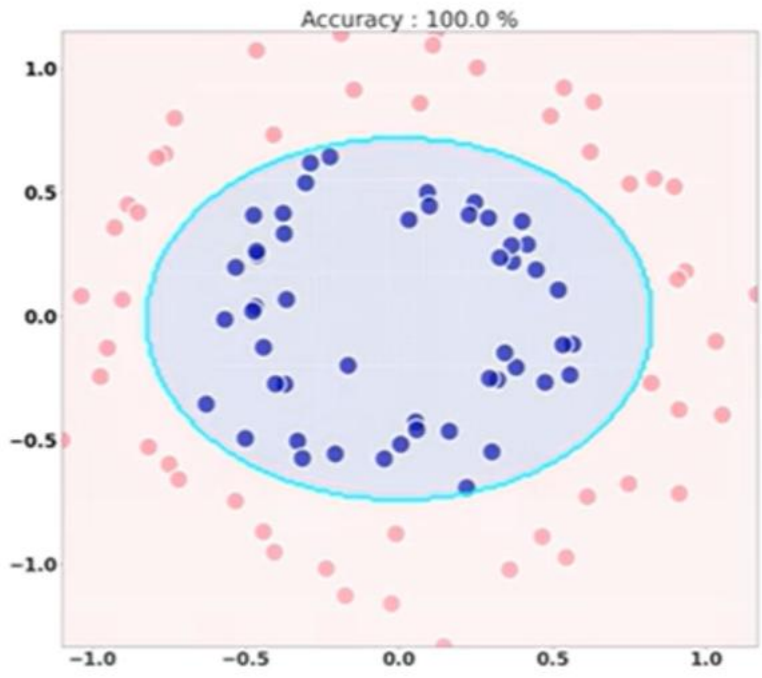
- Example : The model is not at all capable of working with nonlinear data
- Is there a solution?



The Perceptron and Its Limitations

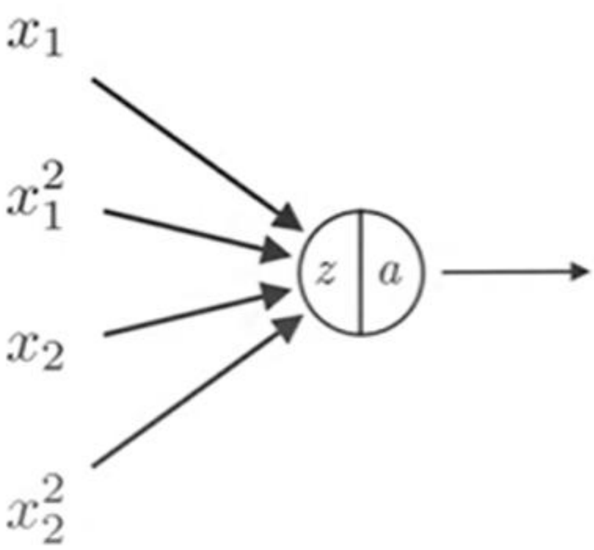
Rosenblatt's Perceptron (1958):

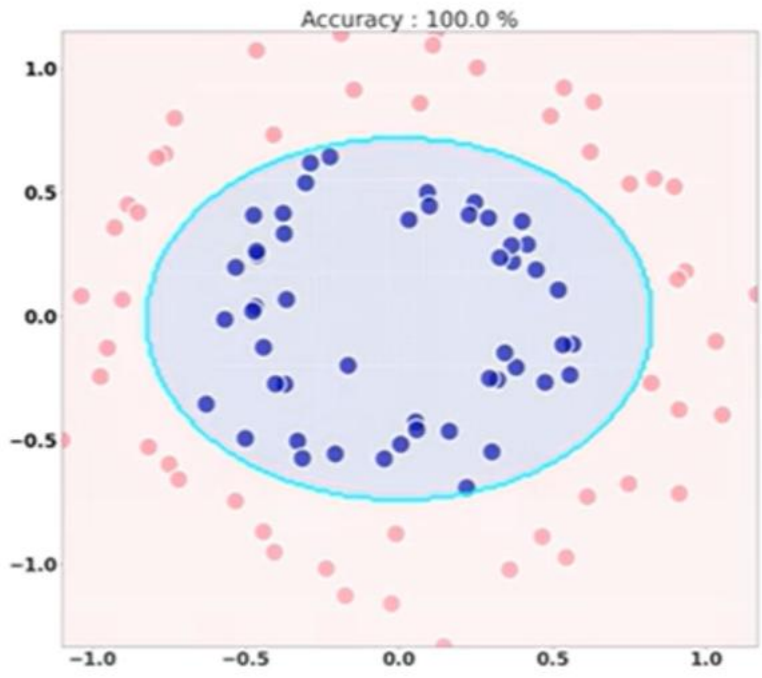
- Example : The model is not at all capable of working with nonlinear data.
- Is there a solution?



Rosenblatt's Perceptron (1958):

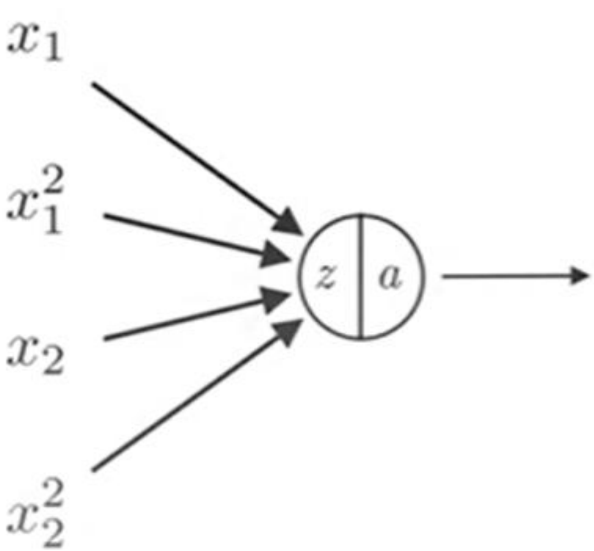
- Example : The model is not at all capable of working with nonlinear data.
- Is there a solution?

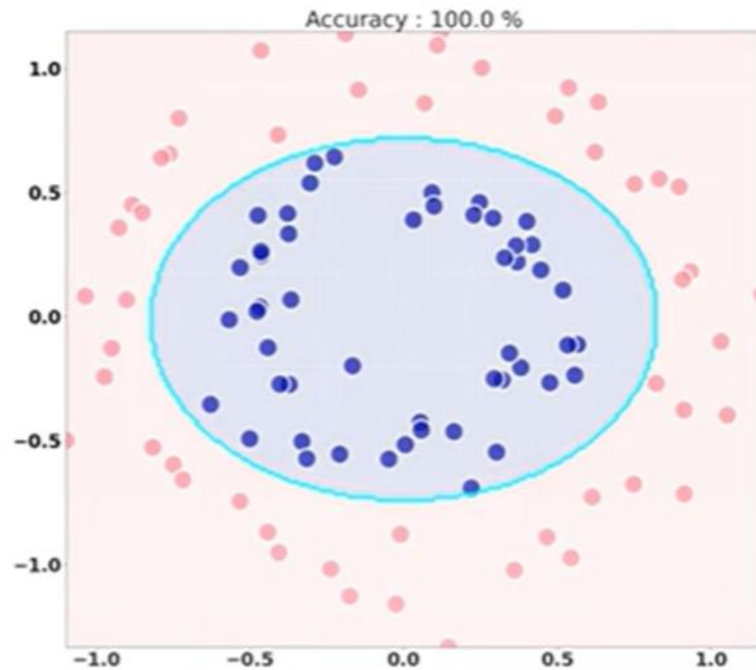




Rosenblatt's Perceptron (1958):

- Example : The model is not at all capable of working with nonlinear data.
- Feature Engineering

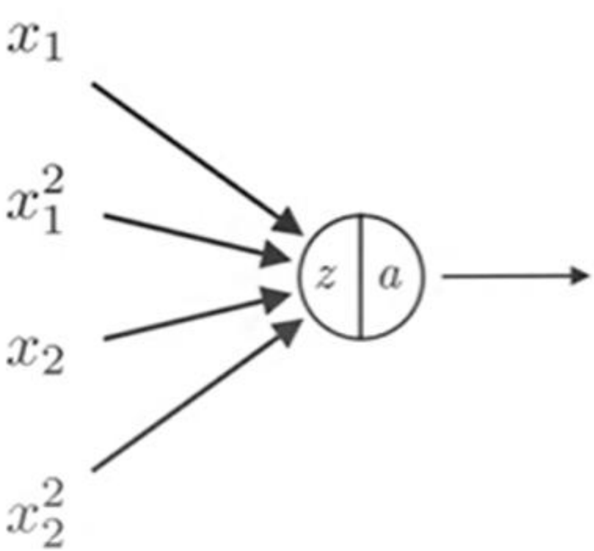


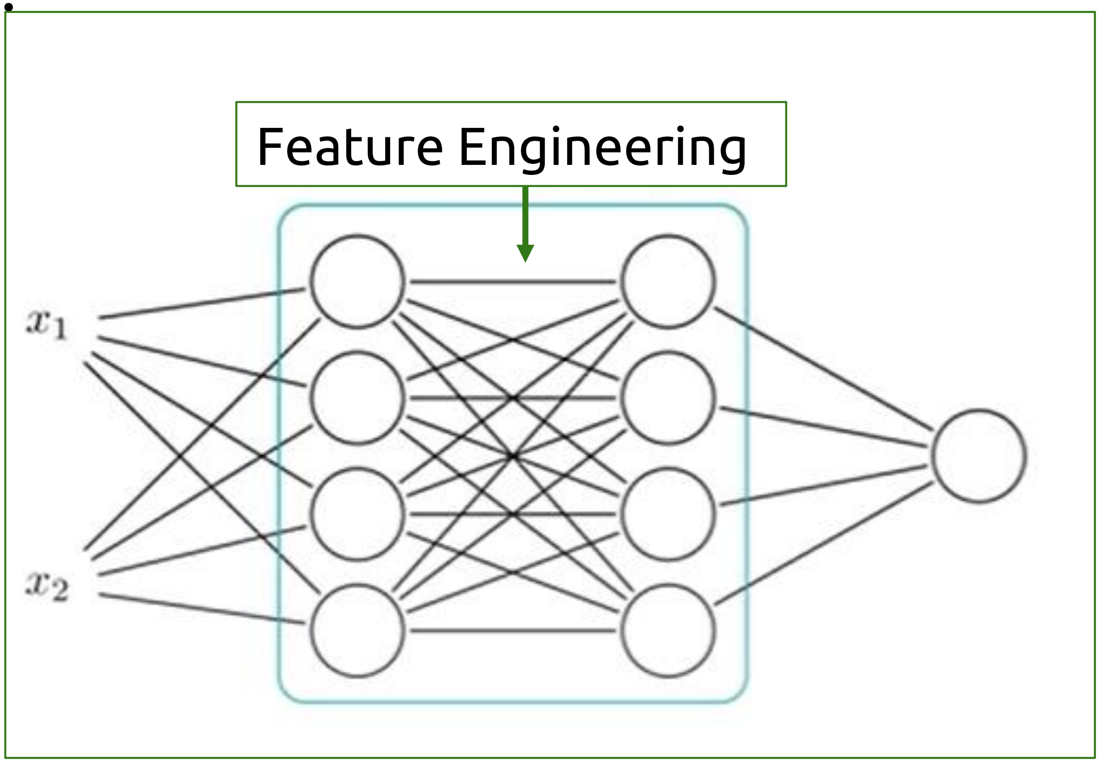


The Perceptron and Its Limitations

Rosenblatt's Perceptron (1958):

- Example : The model is not at all capable of working with nonlinear data.
- Feature Engineering

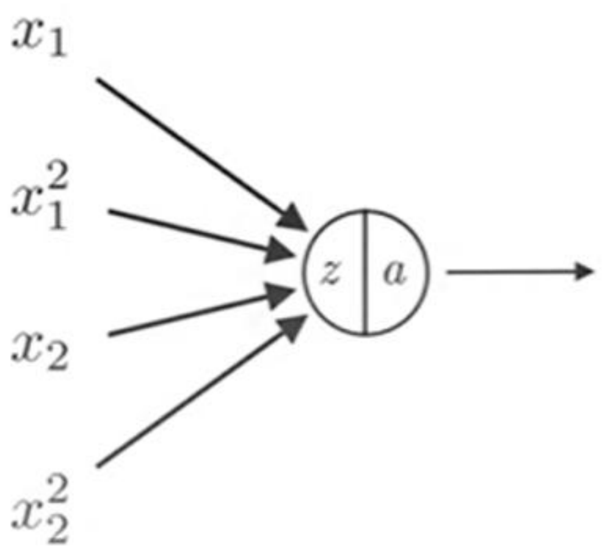


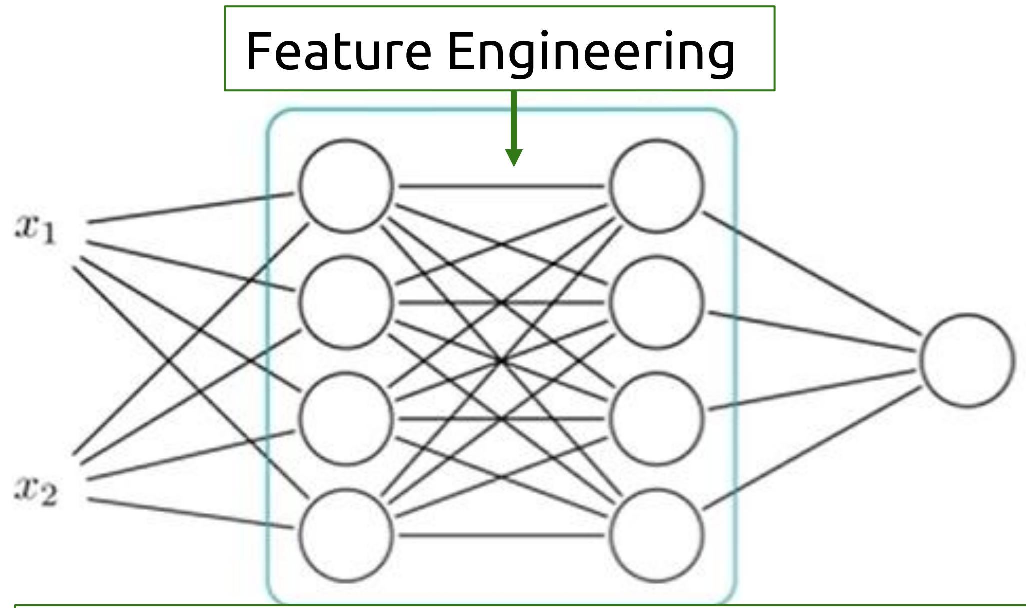


The Perceptron and Its Limitations

Rosenblatt's Perceptron (1958):

- Example : The model is not at all capable of working with nonlinear data.
- Feature Engineering





Developed later (Geoffrey Hinton 1986)…

Biological Roots to Deep Learning

From Perceptron to Multi-Layer Networks

Problem:

- Single-layer perceptrons fail on non-linear problems (e.g., XOR).

Solution:

- Add hidden layers → Multi-Layer Perceptron (MLP).

From Perceptrons to Multi-Layer Networks

Problem:

- Single-layer perceptrons fail on non-linear problems (e.g., XOR).

Solution:

- Add hidden layers → Multi-Layer Perceptron (MLP).

How Do MLP (or Neural Networks) Learn?

Biological Roots to Deep Learning

How Do Neural Networks Learn?

Key Steps:

- **Forward Pass**: Compute predictions.

$$\boxed{\text{Output} = f(\mathbf{W}\mathbf{x} + \mathbf{b})}$$

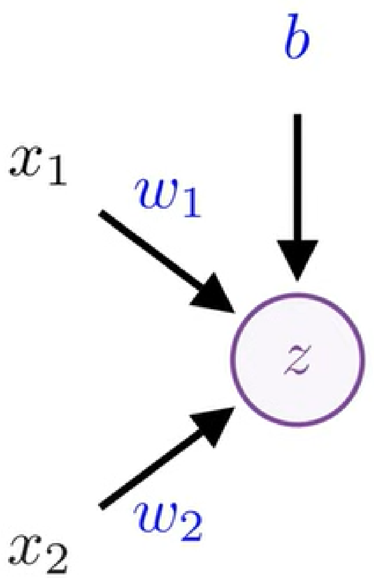
- **Loss Function**: Measure error (e.g., Mean Squared Error).
- **Backpropagation:**
  - Adjust weights using gradient descent.
  - Not in early models (McCulloch-Pitts/Rosenblatt).

How to build a neural network?

How to build a neural network?

Example

- Linear Model

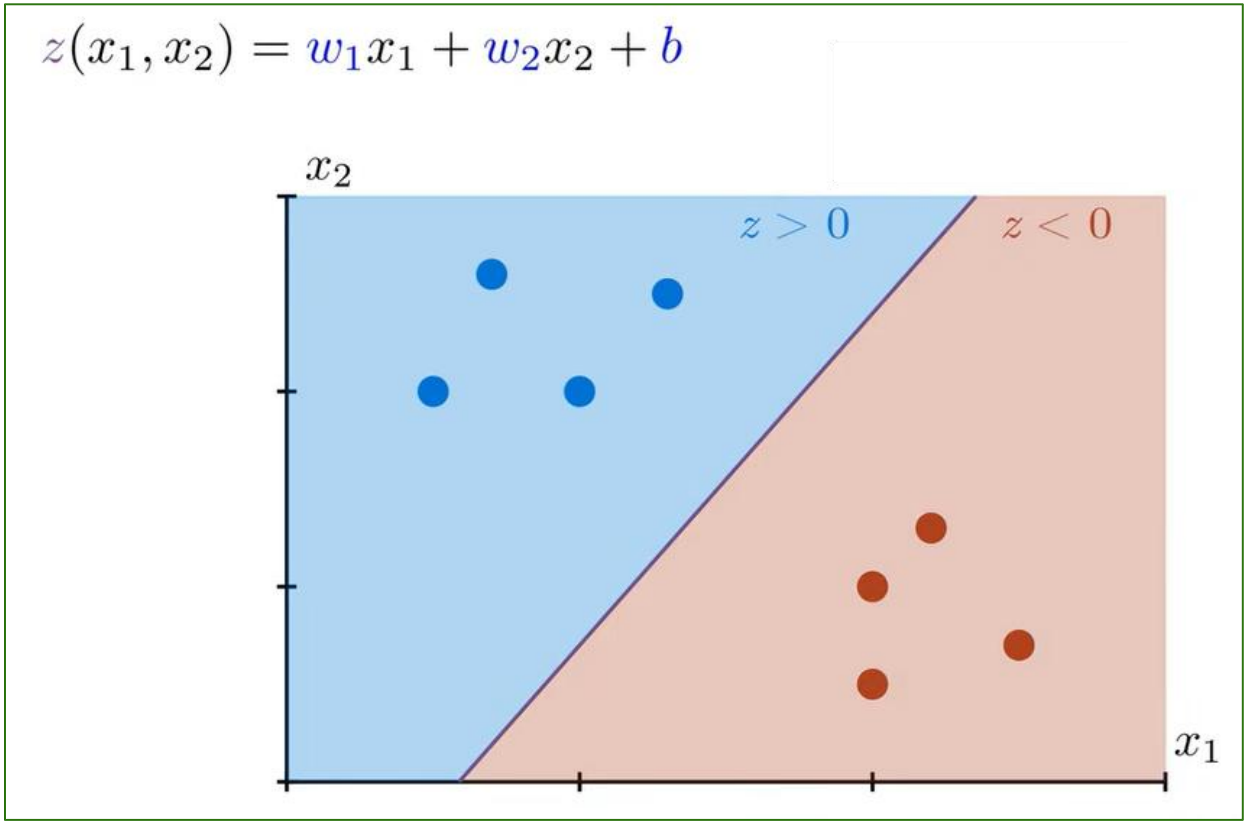


$$z(x_1, x_2) = w_1 x_1 + w_2 x_2 + b$$

How to build a neural network?

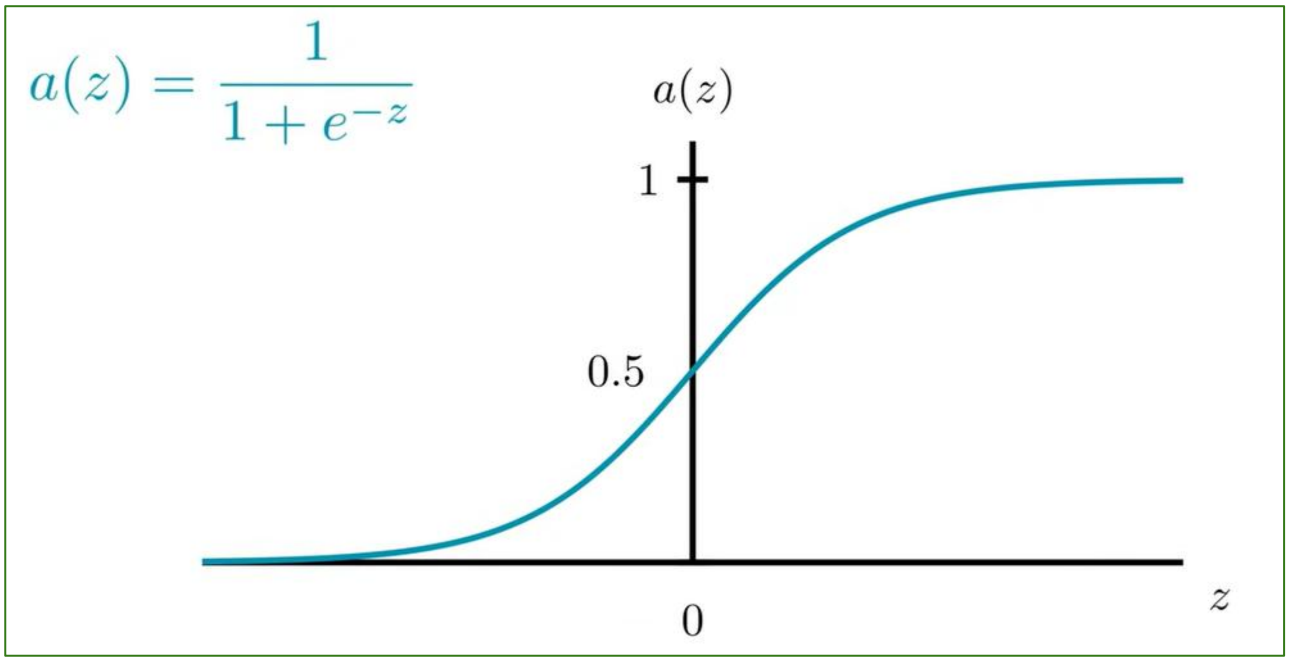
Example

- Decision Boundary



Example

- Sigmoid function



How to build a neural network?

Example

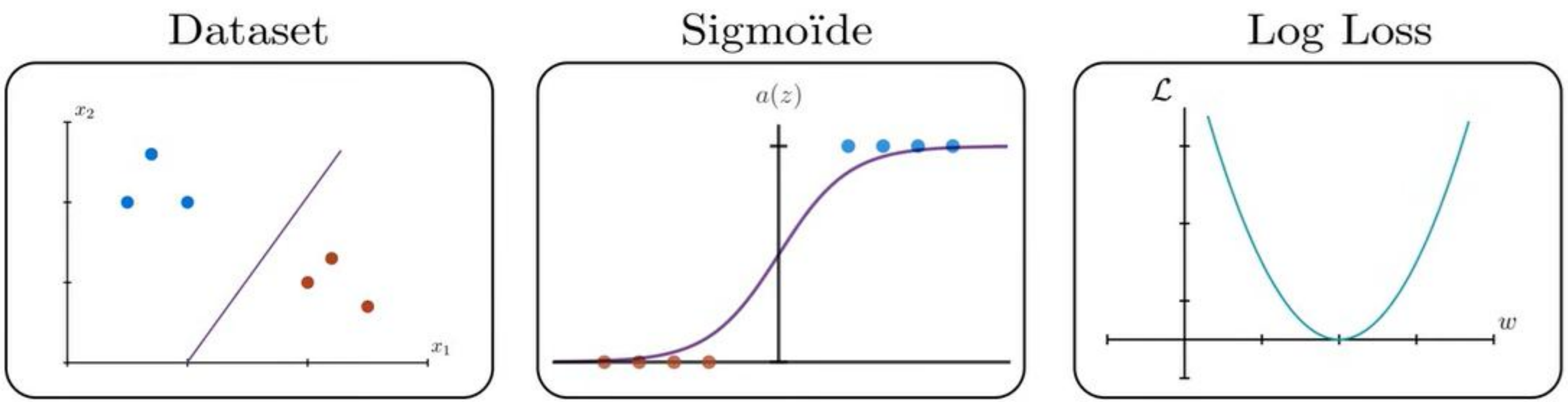
- Summarize

$$z = w_1 x_1 + w_2 x_2 + b$$

$$a = \frac{1}{1 + e^{-z}}$$

$$\mathcal{L} = -\frac{1}{m} \sum_{i=1}^{m} y_i log(a_i) + (1 - y_i) log(1 - a_i)$$

$$W = W - \alpha \frac{\partial \mathcal{L}}{\partial W}$$



Notation

- How to build a neural network

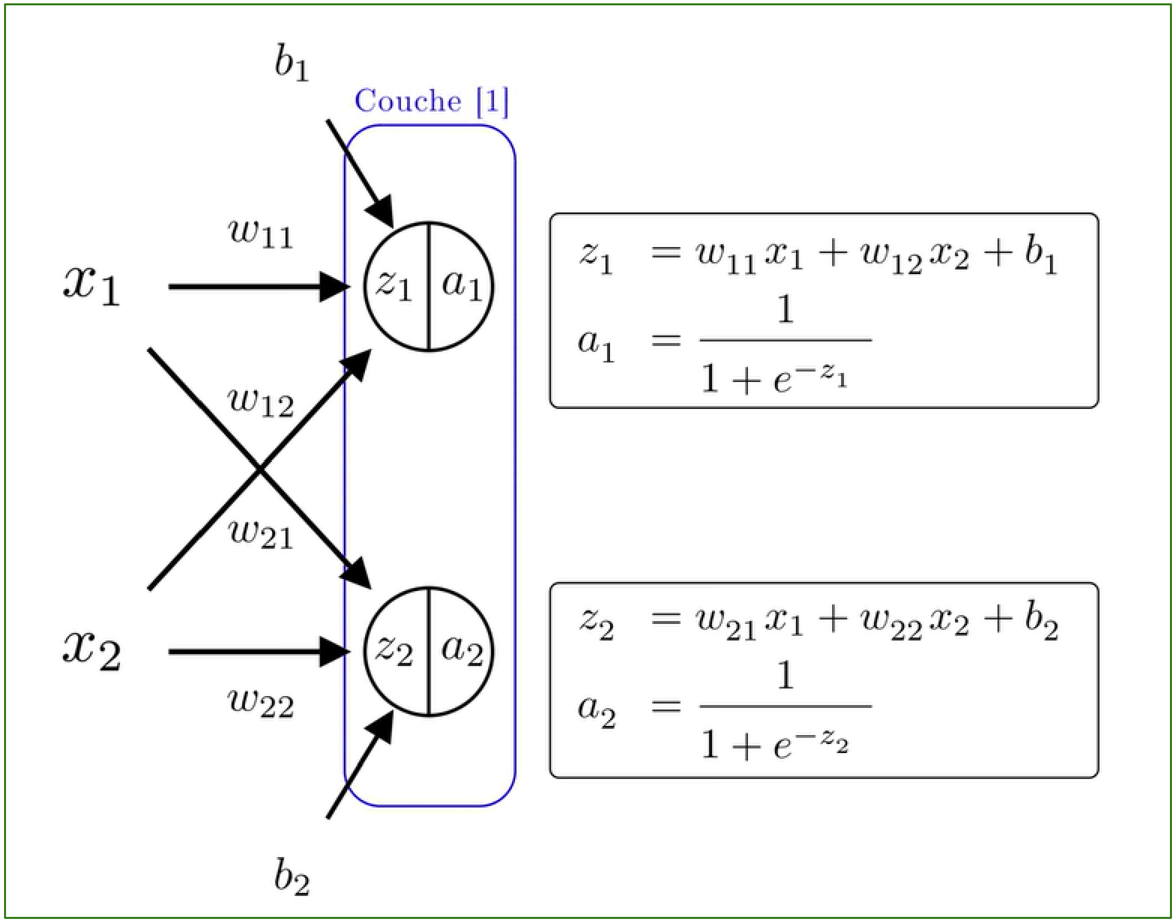
$$\frac{\partial \mathcal{L}}{\partial w_1} = \frac{1}{m}\sum_{i=1}^{m}(a_i - y_i)x_1$$

$$\frac{\partial \mathcal{L}}{\partial w_2} = \frac{1}{m}\sum(a_i - y_i)x_2$$

$$\frac{\partial \mathcal{L}}{\partial b} = \frac{1}{m}\sum_{i=1}^{m}(a_i - y_i)$$

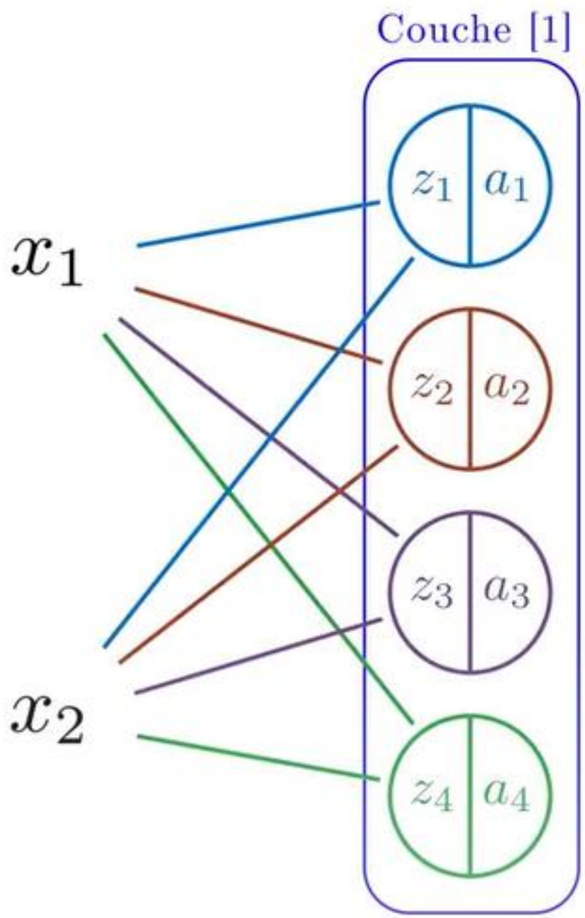
Notation

- How to build a neural network



Notation

- How to build a neural network



$$\begin{cases} z_1 = w_{11}x_1 + w_{12}x_2 + b_1 \\ a_1 = \dfrac{1}{1+e^{-z_1}} \end{cases}$$

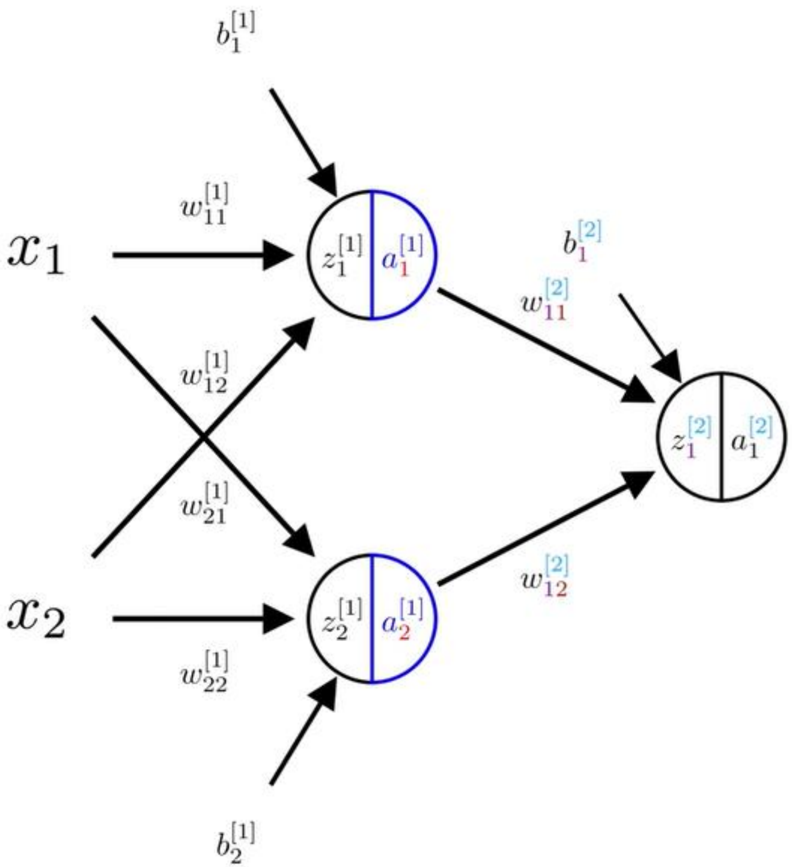
$$\begin{cases} z_2 = w_{21}x_1 + w_{21}x_2 + b_2 \\ a_2 = \dfrac{1}{1+e^{-z_2}} \end{cases}$$

$$\begin{cases} z_3 = w_{31}x_1 + w_{32}x_2 + b_3 \\ a_3 = \dfrac{1}{1+e^{-z_3}} \end{cases}$$

$$\begin{cases} z_4 = w_{41}x_1 + w_{42}x_2 + b_4 \\ a_4 = \dfrac{1}{1+e^{-z_4}} \end{cases}$$

Notation

- How to build a neural network of 2 layers

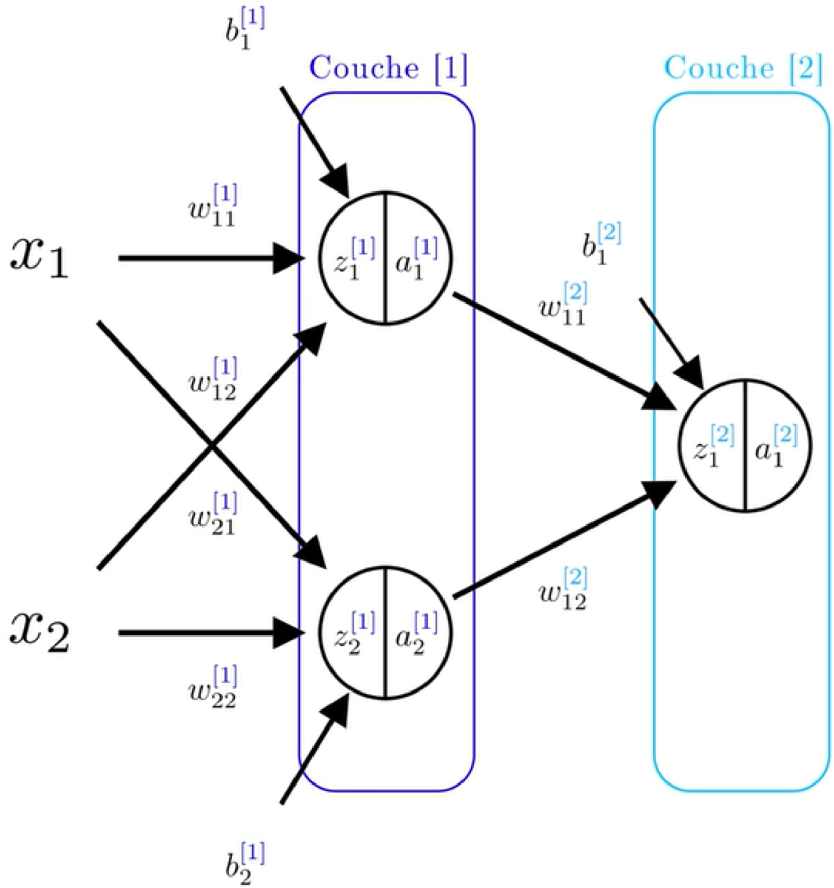


$$z_1^{[2]} = w_{11}^{[2]} a_1^{[1]} + w_{12}^{[2]} a_2^{[1]} + b_1^{[2]}$$

$$a_1^{[2]} = \frac{1}{1 + e^{-z_1^{[2]}}}$$

Notation

- How to build a neural network of 2 layers



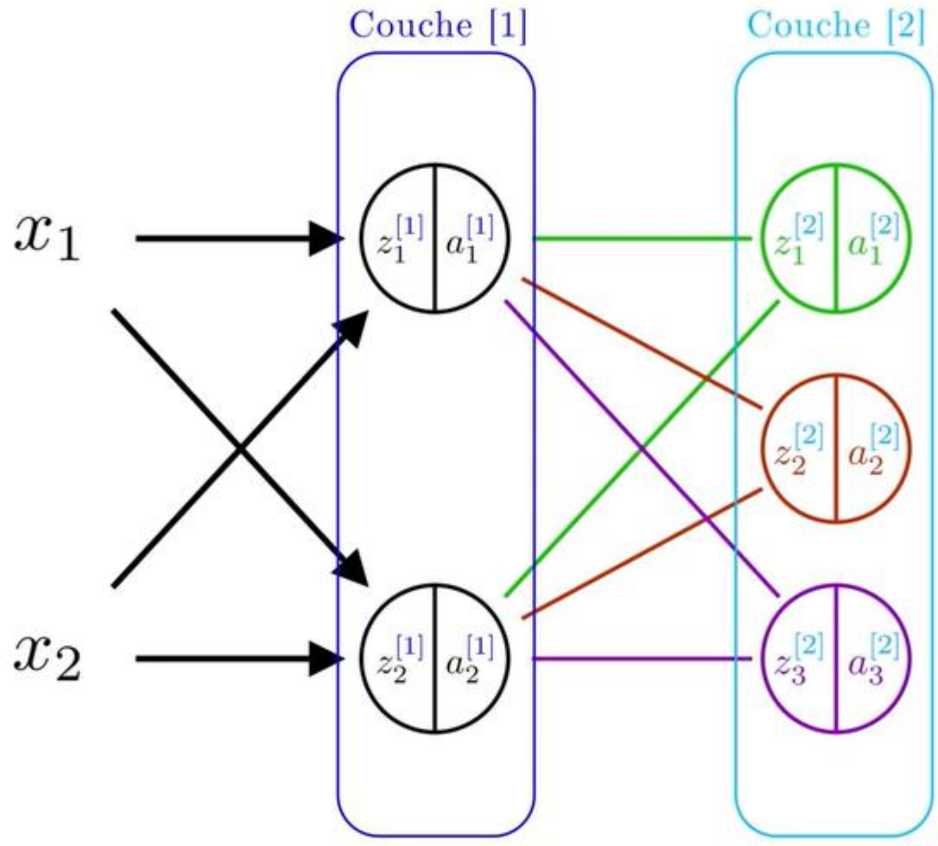
$$z_1^{[1]} = w_{11}^{[1]} x_1 + w_{12}^{[1]} x_2 + b_1^{[1]}$$
$$a_1^{[1]} = \frac{1}{1 + e^{-z_1^{[1]}}}$$

$$z_2^{[1]} = w_{21}^{[1]} x_1 + w_{22}^{[1]} x_2 + b_2^{[1]}$$
$$a_2^{[1]} = \frac{1}{1 + e^{-z_2^{[1]}}}$$

$$z_1^{[2]} = w_{11}^{[2]} a_1^{[1]} + w_{12}^{[2]} a_2^{[1]} + b_1^{[2]}$$
$$a_1^{[2]} = \frac{1}{1 + e^{-z_1^{[2]}}}$$

Notation

- How to build a neural network of 2 layers



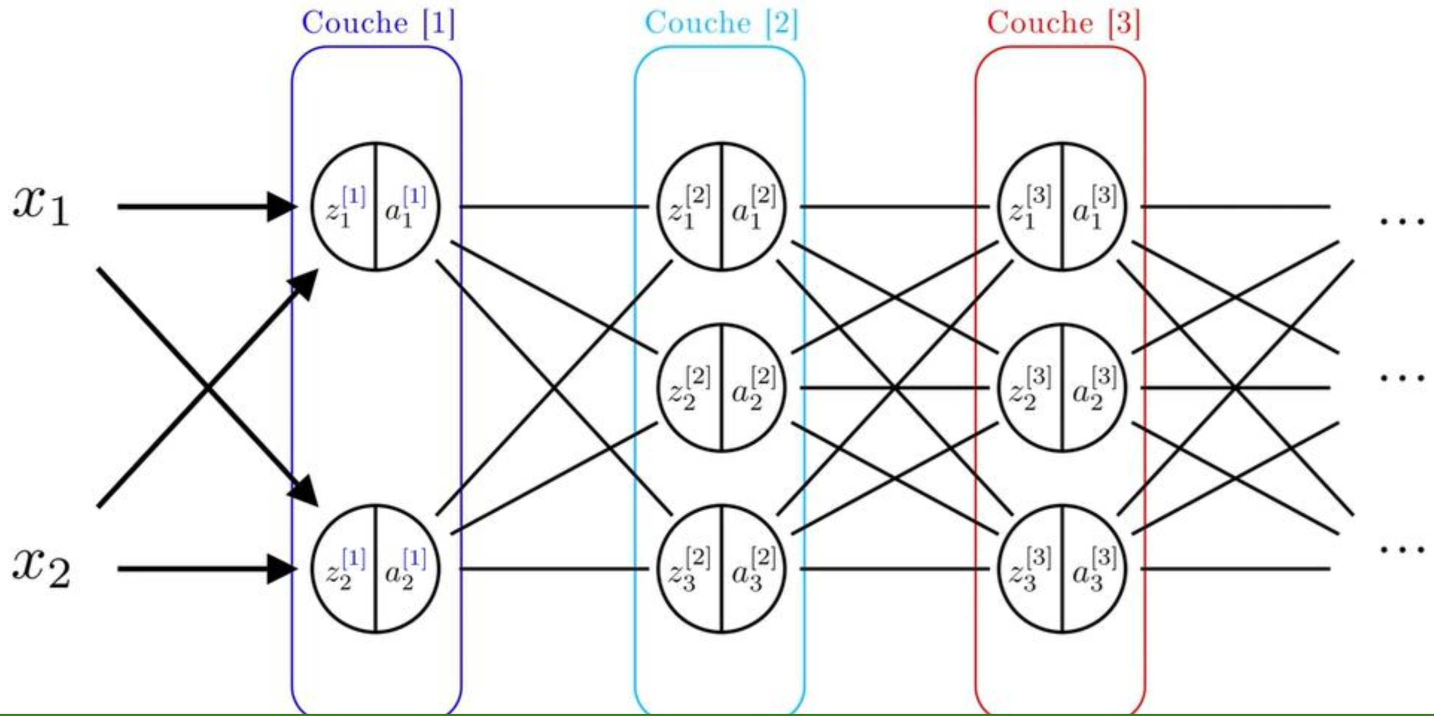
$$\begin{cases} z_1^{[2]} = w_{11}^{[2]} a_1^{[1]} + w_{12}^{[2]} a_2^{[1]} + b_1^{[2]} \\ a_1^{[2]} = \dfrac{1}{1+e^{-z_1^{[2]}}} \end{cases}$$

$$\begin{cases} z_2^{[2]} = w_{21}^{[2]} a_1^{[1]} + w_{22}^{[2]} a_2^{[1]} + b_2^{[2]} \\ a_2^{[2]} = \dfrac{1}{1+e^{-z_2^{[2]}}} \end{cases}$$

$$\begin{cases} z_3^{[2]} = w_{31}^{[2]} a_1^{[1]} + w_{32}^{[2]} a_2^{[1]} + b_3^{[2]} \\ a_3^{[2]} = \dfrac{1}{1+e^{-z_3^{[2]}}} \end{cases}$$

Notation

- How to build a neural network of N layers



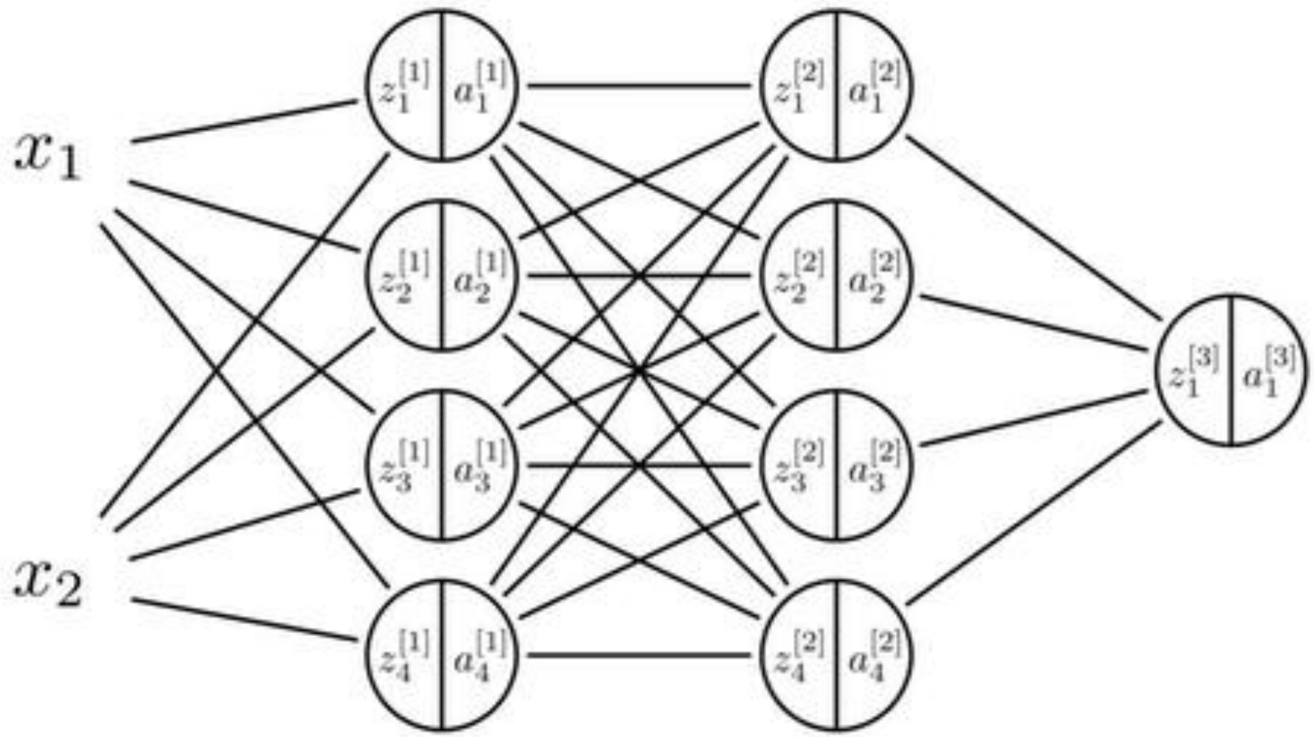
Plus le réseau est profond, plus il est capable d'apprendre des choses compliquées.
Mais cela rend aussi l'apprentissage plus long.

How to build a neural network?

Notation

- Vectorization

Pour implémenter de tels modèles, il n'est pas pratique d'écrire les équations de chaque neurone... l'idéal est donc de ces équations.



Notation

- Vectorization

$$z_1^{[1]} = w_{11}^{[1]} x_1 + w_{12}^{[1]} x_2 + b_1^{[1]} \qquad z_1^{[2]} = w_{11}^{[1]} a_1^{[1]} + w_{12}^{[2]} a_2^{[1]} + w_{13}^{[2]} a_3^{[1]} + w_{14}^{[2]} a_4^{[1]} + b_1^{[2]}$$

$$z_2^{[1]} = w_{21}^{[1]} x_1 + w_{22}^{[1]} x_2 + b_2^{[1]} \qquad z_2^{[2]} = w_{21}^{[1]} a_1^{[1]} + w_{22}^{[2]} a_2^{[1]} + w_{23}^{[2]} a_3^{[1]} + w_{24}^{[2]} a_4^{[1]} + b_2^{[2]}$$

$$z_3^{[1]} = w_{31}^{[1]} x_1 + w_{32}^{[1]} x_2 + b_3^{[1]} \qquad z_3^{[2]} = w_{31}^{[1]} a_1^{[1]} + w_{32}^{[2]} a_2^{[1]} + w_{33}^{[2]} a_3^{[1]} + w_{34}^{[2]} a_4^{[1]} + b_3^{[2]}$$

$$z_4^{[1]} = w_{41}^{[1]} x_1 + w_{42}^{[1]} x_2 + b_4^{[1]} \qquad z_4^{[2]} = w_{41}^{[1]} a_1^{[1]} + w_{42}^{[2]} a_2^{[1]} + w_{43}^{[2]} a_3^{[1]} + w_{44}^{[2]} a_4^{[1]} + b_4^{[2]}$$

$$z_1^{[3]} = w_{11}^{[3]} a_1^{[2]} + w_{12}^{[3]} a_2^{[2]} + w_{13}^{[3]} a_3^{[2]} + w_{14}^{[3]} a_4^{[2]} + b_1^{[3]}$$

Notation

- Vectorization

$$z_1^{[1]} = w_{11}^{[1]}x_1 + w_{12}^{[1]}x_2 + b_1^{[1]} \qquad z_1^{[2]} = w_{11}^{[1]}a_1^{[1]} + w_{12}^{[2]}a_2^{[1]} + w_{13}^{[2]}a_3^{[1]} + w_{14}^{[2]}a_4^{[1]} + b_1^{[2]}$$

$$z_2^{[1]} = w_{21}^{[1]}x_1 + w_{22}^{[1]}x_2 + b_2^{[1]} \qquad z_2^{[2]} = w_{21}^{[1]}a_1^{[1]} + w_{22}^{[2]}a_2^{[1]} + w_{23}^{[2]}a_3^{[1]} + w_{24}^{[2]}a_4^{[1]} + b_2^{[2]}$$

$$\boxed{Z^{[1]} = W^{[1]} \cdot X + b^{[1]}} \qquad z_3^{[2]} = w_{31}^{[1]} \quad \boxed{Z^{[2]} = W^{[2]} \cdot A^{[1]} + b^{[2]}} \quad b_3^{[2]}$$

$$z_4^{[1]} = w_{41}^{[1]}x_1 + w_{42}^{[1]}x_2 + b_4^{[1]} \qquad z_4^{[2]} = w_{41}^{[1]}a_1^{[1]} + w_{42}^{[2]}a_2^{[1]} + w_{43}^{[2]}a_3^{[1]} + w_{44}^{[2]}a_4^{[1]} + b_4^{[2]}$$

$$z_1^{[3]} = w \quad \boxed{Z^{[3]} = W^{[3]} \cdot A^{[2]} + b^{[3]}} \quad + b_1^{[3]}$$

Forward Propagation & Activation Functions

Forward Propagation

- Forward propagation is the process by which input data passes through the network layer by layer to produce a prediction/output.
- General formula for one neuron:

$$z = \sum_{i=1}^{n} w_i x_i + b = \mathbf{w}^T \mathbf{x} + b$$

$$a = f(z)$$

Where:

- x = input vector
- w = weight vector
- b = bias
- f = activation function
- a = output of the neuron

Forward Propagation – Layer by Layer

- For layer l:

$$\mathbf{z}^{[l]} = \mathbf{W}^{[l]}\mathbf{a}^{[l-1]} + \mathbf{b}^{[l]}$$

$$\mathbf{a}^{[l]} = f(\mathbf{z}^{[l]})$$

- Repeat this computation through each layer until the output layer.
- Final output: $\hat{y} = \mathbf{a}^{[L]}$

Activation Functions

- Without non-linear activation functions, the whole network acts like a linear model:

composition of linear functions is still linear

- Activation functions allow the model to capture non-linear patterns.

Common Activation Functions

Function	Formula	Range
Sigmoid	$\sigma(z) = \frac{1}{1+e^{-z}}$	(0, 1)
Tanh	$\tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$	(−1, 1)
ReLU	$\text{ReLU}(z) = \max(0, z)$	[0, ∞)

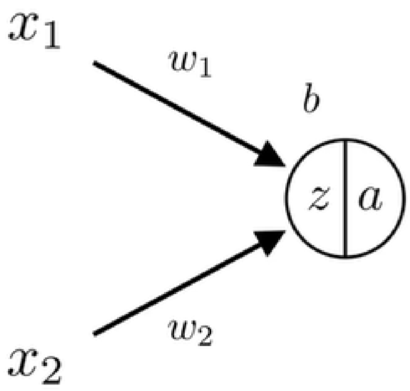
Choosing the Right Activation Function

- **Sigmoid**: good for output layers in binary classification, but can cause vanishing gradients.
- **Tanh**: better than sigmoid in hidden layers, but still suffers from gradient issues.
- **ReLU**: standard for hidden layers in deep networks.
- Others (brief mention): Leaky ReLU, ELU, Softmax (for multi-class outputs)

Backpropagation: The Core of Learning

Backpropagation: The Core of Learning

Recall: Training of an Artificial Neuron

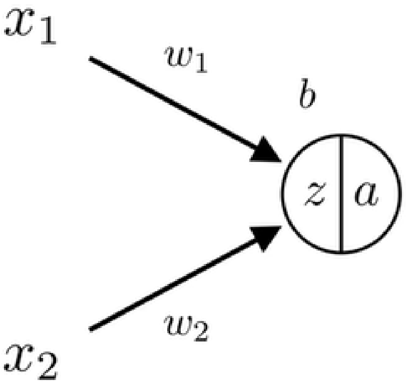


- **1. Define a Cost Function**

$$\mathcal{L} = -\frac{1}{m}\sum y \times \log(A) + (1-y) \times \log(1-A)$$

binary cross-entropy loss (also called log loss).

Recall: Training of an Artificial Neuron



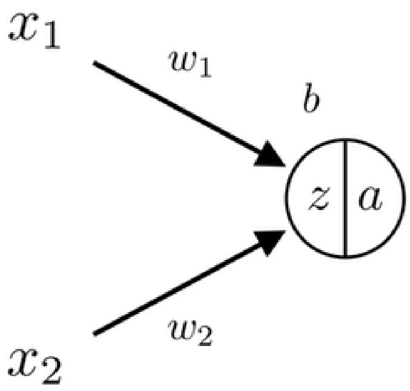
- **2. Compute the Partial Derivatives**

$$\frac{\partial \mathcal{L}}{\partial W} = \frac{1}{m} X^T \cdot (A - y) \qquad \frac{\partial \mathcal{L}}{\partial b} = \frac{1}{m} \sum (A - y)$$

These are the gradients of the loss function with respect to the parameters W and b.

Backpropagation: The Core of Learning

Recall: Training of an Artificial Neuron



- **3. Update Parameters W and b (Gradient Descent Step)**

$$W = W - \alpha \frac{\partial \mathcal{L}}{\partial W} \qquad b = b - \alpha \frac{\partial \mathcal{L}}{\partial b}$$

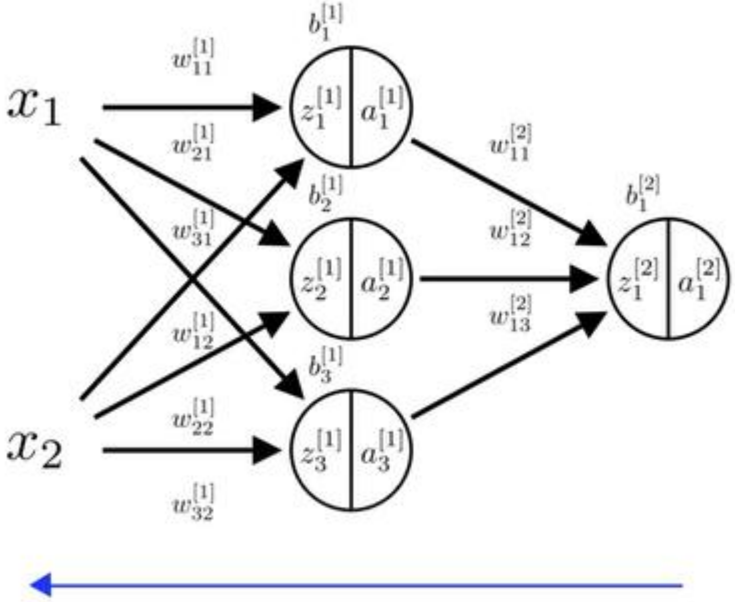
This is the gradient descent update rule, where α is the learning rate.

Backpropagation: The Core of Learning

Backpropagation

it consists in tracing back how the Cost Function evolves from the last layer of the network all the way to the very first.

La Back-Propagation consiste à revenir en arrière pour comprendre comment la fonction coût évolue depuis la dernière couche du réseau jusqu'à la toute première.



Backpropagation (Example 2 layers)

$$\boxed{Z^{[1]} = W^{[1]} \cdot X + b^{[1]}}$$

$$A^{[1]} = \frac{1}{1 + e^{-Z^{[1]}}}$$

$$Z^{[2]} = W^{[2]} \cdot A^{[1]} + b^{[2]}$$

$$A^{[2]} = \frac{1}{1 + e^{-Z^{[2]}}}$$

$$\mathcal{L} = -\frac{1}{m} \sum y \times log(A^{[2]}) + (1 - y) \times log(1 - A^{[2]})$$

Backpropagation: The Core of Learning

Steps:

1- Compute gradients w.r.t weights and biases

Each parameter $w_{ij}^{[C]}$ connects neuron j (previous layer) to neuron i (current layer):

$$\frac{\partial \mathcal{L}}{\partial w_{ij}^{[C]}} = \delta_i^{[C]} \cdot a_j^{[C-1]}$$

$$\frac{\partial \mathcal{L}}{\partial b_i^{[C]}} = \delta_i^{[C]}$$

matrix form

$$\boxed{\begin{aligned} \frac{\partial \mathcal{L}}{\partial W^{[C]}} &= \delta^{[C]} \cdot (a^{[C-1]})^T \\ \frac{\partial \mathcal{L}}{\partial b^{[C]}} &= \delta^{[C]} \end{aligned}}$$

Backpropagation: The Core of Learning

Steps:

2- Backpropagate the error to the previous layer

$$\boxed{\delta^{[C-1]} = \left((W^{[C]})^T \delta^{[C]}\right) \circ \sigma'(z^{[C-1]})}$$

Where:

- $\circ$ is element-wise multiplication (Hadamard product),
- $\sigma'(z) = a^{[C-1]} \circ (1 - a^{[C-1]})$ for sigmoid.

Backpropagation: The Core of Learning

Steps:

3. Update Parameters

Using learning rate α:

$$\boxed{\begin{gathered} W^{[C]} := W^{[C]} - \alpha \cdot \frac{\partial \mathcal{L}}{\partial W^{[C]}} \\ b^{[C]} := b^{[C]} - \alpha \cdot \frac{\partial \mathcal{L}}{\partial b^{[C]}} \end{gathered}}$$

Thank you for your attention…