

COMPTE RENDU

## **Introduction aux réseaux de neurones**

Souheib Ben Mabrouk

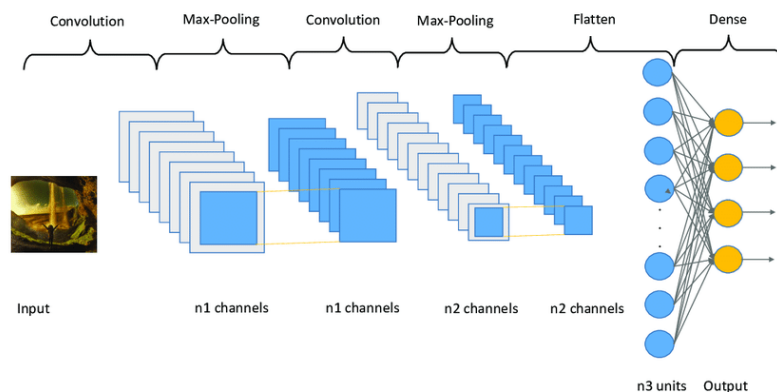
INDP2E

8 novembre 2021

# Introduction

Un réseau de neurones convolutif (ConvNet/CNN) est un algorithme d'apprentissage profond qui peut prendre une image en entrée, attribuer de l'importance (poids et biais) à divers aspects et objets de l'image et être capable de les différencier les uns des autres. [1]

Le prétraitement requis dans un ConvNet est beaucoup plus faible que dans d'autres algorithmes de classification. Alors que dans les méthodes primitives, les filtres sont conçus à la main, avec un entraînement suffisant, les ConvNet ont la capacité d'apprendre ces filtres et caractéristiques. L'architecture d'un ConvNet est analogue à celle du modèle de connectivité des neurones dans le cerveau humain et s'inspire de l'organisation du cortex visuel. Les neurones individuels ne répondent aux stimuli que dans une région restreinte du champ visuel, appelée champ de réception. Un ensemble de ces champs se chevauchent pour couvrir l'ensemble de la zone visuelle.



**Fig. 1** — Réseau de neurones convolutifs

L'objectif de ce rapport est de se familiariser avec le concept de réseaux de neurones ainsi que d'implémenter quelques fonctions basiques tout en tirant des conclusions.

Pour ce faire, j'ai choisi de travailler avec **Python 3.8** comme langage de programmation. D'autre part, j'utiliserai le système de composition de documents **LATEX** afin d'avoir un rapport bien conçu avec les formulations mathématiques nécessaires.

# Table des matières

<b>Introduction</b>	<b>1</b>
<b>1 Principe</b>	<b>4</b>
1.1 Présentation . . . . .	4
1.2 Modélisation Mathématique . . . . .	5
1.2.1 Notion de Perceptron . . . . .	5
1.2.2 Neurone sigmoïde . . . . .	7
1.3 Un réseau simple pour classer les chiffres manuscrits . . . . .	9
<b>2 Implementation Python des fonctions logiques à partir du modèle de perceptron</b>	<b>10</b>
2.1 Préparation du perceptron . . . . .	10
2.2 Implémentation de la fonction NOT . . . . .	11
2.3 Implémentation de la fonction AND . . . . .	11
2.4 Implémentation de la fonction OR . . . . .	11
2.5 Implémentation de la fonction XOR . . . . .	12
<b>Conclusion</b>	<b>14</b>

# Table des figures

1	Réseau de neurones convolutifs . . . . .	1
1.1	Exemple de modélisation d'un réseau de neurones convolutifs . . . . .	5
1.2	Un perceptron . . . . .	5
1.3	Un perceptron simplifié . . . . .	6
1.4	Apprentissage par un perceptron unique . . . . .	7
1.5	Réseau de neurones . . . . .	7
1.6	La fonction Sigmoidale . . . . .	8
1.7	classer les chiffres manuscrits . . . . .	9
2.1	Modélisation les fonctions logiques avec le modèle de perceptron . . . . .	10
2.2	Conception de la fonction Xor à partir des perceptron AND, NOT, OR et AND . . . .	12
2.3	Résultat final obtenu . . . . .	13

# Chapitre 1

## Principe

### 1.1 Présentation

Les réseaux neuronaux sont un sous-ensemble de l'apprentissage automatique, et ils sont au cœur des algorithmes d'apprentissage profond. Ils sont constitués de couches de nœuds, contenant une couche d'entrée, une ou plusieurs couches cachées et une couche de sortie.

[2] Chaque nœud est relié à un autre et possède un poids et un seuil associés. Si la sortie d'un nœud individuel est supérieure à la valeur seuil spécifiée, ce nœud est activé et envoie des données à la couche suivante du réseau. Dans le cas contraire, aucune donnée n'est transmise à la couche suivante du réseau.

Il existe plusieurs types de réseaux neuronaux, qui sont utilisés pour différents cas d'utilisation et types de données. Par exemple, les réseaux neuronaux récurrents sont couramment utilisés pour le traitement du langage naturel et la reconnaissance vocale, tandis que les réseaux neuronaux convolutifs (ConvNets ou CNN) sont plus souvent utilisés pour les tâches de classification et de vision par ordinateur.

Avant l'avènement des réseaux neuronaux convolutifs, des méthodes d'extraction de caractéristiques manuelles et fastidieuses étaient utilisées pour identifier les objets dans les images. Cependant, les réseaux de neurones convolutifs offrent désormais une approche plus évolutive des tâches de classification d'images et de reconnaissance d'objets, en tirant parti des principes de l'algèbre linéaire, en particulier de la multiplication matricielle, pour identifier les motifs dans une image. Cela dit, ils peuvent être exigeants en termes de calcul, nécessitant des unités de traitement graphique (GPU) pour former les modèles.

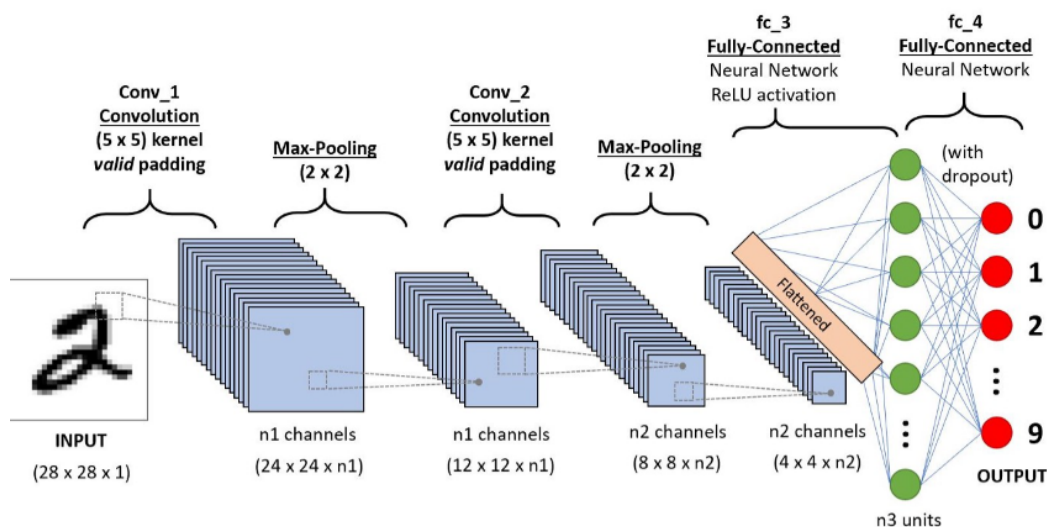


Fig. 1.1 – Exemple de modélisation d'un réseau de neurones convolutifs

## 1.2 Modélisation Mathématique

### 1.2.1 Notion de Perceptron

Malgré sa nouvelle notoriété, le domaine des réseaux neuronaux n'est pas nouveau du tout. En 1958, Frank Rosenblatt, un psychologue américain, a tenté de construire "une machine qui sent, reconnaît, se souvient et répond comme l'esprit humain" et a appelé sa machine un perceptron. Mais Rosenblatt n'a pas inventé les perceptrons de toutes pièces. . Voilà qui fait des réseaux neuronaux ou, plus précisément, du perceptron, un dinosaure dans ce monde technologique en pleine évolution. [3]

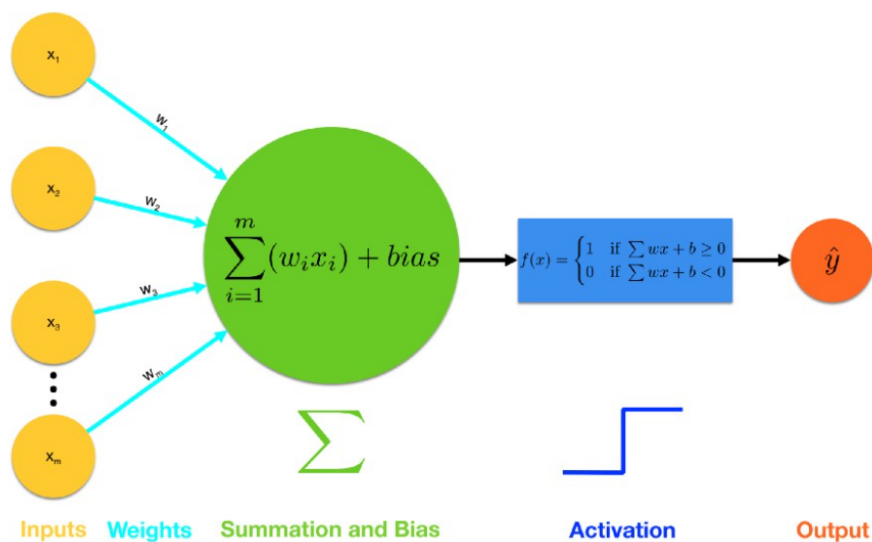


Fig. 1.2 – Un perceptron

En effet. Les perceptrons et autres réseaux neuronaux sont inspirés de véritables neurones dans notre cerveau. Ils ne sont qu'inspirés et ne fonctionnent pas exactement comme les vrais neurones.

La procédure de traitement des données par un perceptron est la suivante :

**1-** Sur le côté gauche, vous avez des neurones (petits cercles) de  $x$  avec les indices 1, 2, ... ,  $m$  portant des données d'entrée.

**2-** Nous multiplions chacune des entrées par un poids  $w$ , également étiqueté avec les indices 1, 2, ...,  $m$ , le long de la flèche vers le grand cercle du milieu. Ainsi,  $w_1 * x_1$ ,  $w_2 * x_2$ ,  $w_3 * x_3$  et ainsi de suite.

**3-** Une fois que toutes les entrées sont multipliées par un poids, nous les additionnons toutes et ajoutons un autre nombre prédéterminé appelé biais.

**4-** Ensuite, nous poussons le résultat vers la droite. Maintenant, nous avons cette fonction d'étape dans le rectangle. Ce que cela signifie, c'est que si le résultat de l'étape 3 est un nombre égal ou supérieur à 0, alors nous obtenons 1 en sortie, sinon si le résultat est inférieur à 0, nous obtenons 0 en sortie.

**5-** La sortie est soit 1 soit 0.

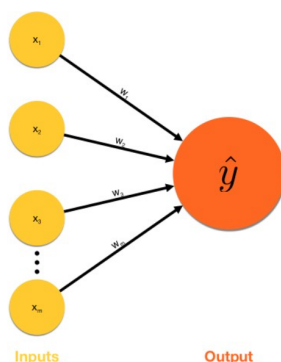
**Remarque :**

Si on déplace le biais vers le côté droit de l'équation dans la fonction d'activation comme :

$$\sum_i (w_i x_i) \geq -b$$

alors ce  $-b$  est appelé une valeur seuil. Ainsi, si la somme des entrées et des poids est supérieure ou égale au seuil, alors l'activation déclenche un 1. Sinon, le résultat de l'activation est 0.

Nous pourrions parfois simplifier notre perceptron comme suit. Ce type de perceptron dont nous venons de parler est également un perceptron à une seule couche puisque nous traitons les entrées directement en une sortie sans aucune autre couche de neurones au milieu.



**Fig. 1.3** – Un perceptron simplifié

Le principe de d'apprentissage par le perceptron consiste à ajuster les poids du modèle comme suit :

$$f(x) = \begin{cases} 1 & \text{if } \sum w_i x_i + b \geq 0 \\ 0 & \text{if } \sum w_i x_i + b < 0 \end{cases}$$

Learning algorithm:  $w_i = w_i + (y - \hat{y})x_i$

Fig. 1.4 – Apprentissage par un perceptron unique

On se trouve donc en plein du cours d'optimisation pour bien comprendre le modèle de perceptron, il ne s'agit que de résoudre des problèmes d'optimisation en agissant sur les poids.

Un réseau de Neurones est un ensemble de perceptrons permettant de réaliser une fonctionnalité donnée.

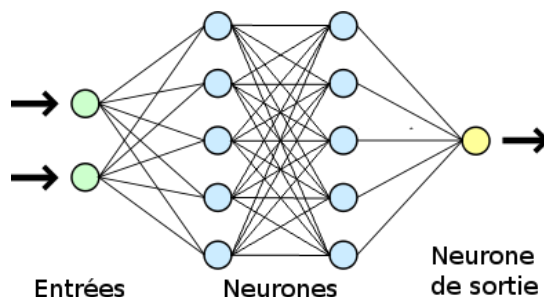


Fig. 1.5 – Réseau de neurones

## 1.2.2 Neurone sigmoïde

### Propriétés importantes de la fonction d'activation

Les propriétés de la fonction d'activation influent en effet sur celle du neurone formel et il est donc important de bien choisir celle-ci pour obtenir un modèle utile en pratique.

Quand les neurones sont combinés en un réseau de neurones formels, il est important par exemple que la fonction d'activation de certains d'entre eux ne soit pas un polynôme sous réserve de limiter la puissance de calcul du réseau obtenu. Un cas caricatural de puissance limitée correspond à l'utilisation d'une fonction d'activation linéaire, comme la fonction identité : dans une telle situation le calcul global réalisé par le réseau est lui aussi linéaire et il est donc parfaitement inutile d'utiliser plusieurs neurones, un seul donnant des résultats strictement équivalents.



Il est aussi utile en pratique que la fonction d'activation présente une certaine forme de régularité. Pour calculer le gradient de l'erreur commise par un réseau de neurones, lors de son apprentissage, il faut que la fonction d'activation soit dérivable. Pour calculer la matrice hessienne de l'erreur, ce qui est utile pour certaines analyses d'erreur, il faut que la fonction d'activation soit dérivable deux fois. Comme elles comportent généralement des points singuliers, les fonctions linéaires par morceaux sont relativement peu utilisées en pratique

### La fonction sigmoïde

La fonction sigmoïde (aussi appelée fonction logistique ou fonction en S) est définie par :

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

Elle possède plusieurs propriétés qui la rendent intéressante comme fonction d'activation.

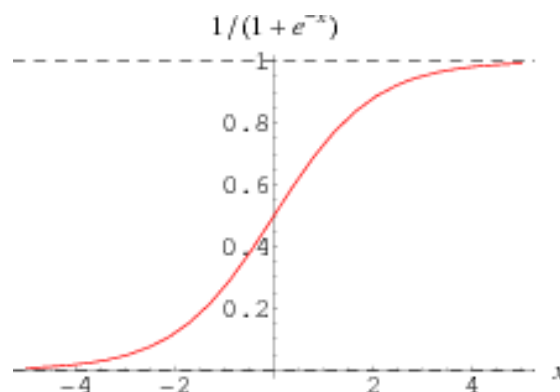
Elle n'est pas polynômiale. Elle est indéfiniment continûment dérivable et une propriété simple permet d'accélérer le calcul de sa dérivée. Ceci réduit le temps calcul nécessaire à l'apprentissage d'un réseau de neurones.

On a en effet

$$\frac{d}{dx} f_{sig}(x) = f_{sig}(x) (1 - f_{sig}(x)) .$$

On peut donc calculer la dérivée de cette fonction en un point de façon très efficace à partir de sa valeur en ce point.

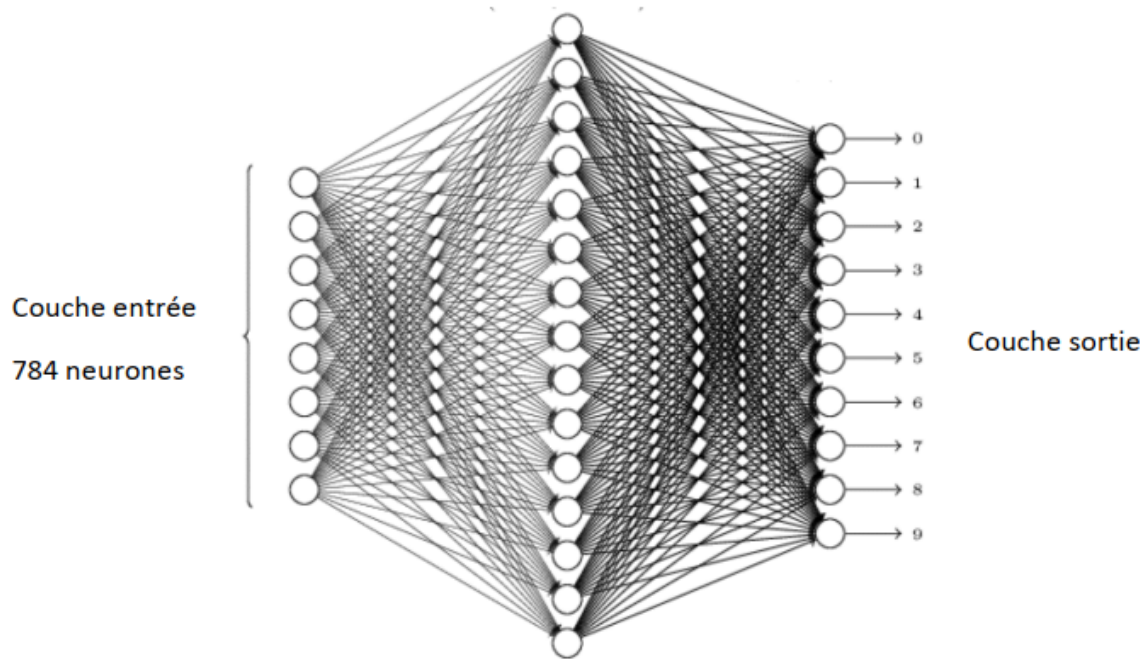
De plus, la fonction sigmoïde renvoie des valeurs dans l'intervalle  $[0,1]$ , ce qui permet d'interpréter la sortie du neurone comme une probabilité.



**Fig. 1.6** – La fonction Sigmoïde

### 1.3 Un réseau simple pour classer les chiffres manuscrits

Après avoir défini les réseaux de neurones, revenons à la reconnaissance de l'écriture manuscrite. Chaque neurone dans la première couche cachée reçoit toutes les entrées de la première couche. Pour la couche de sortie, nous aurons maintenant 10 neurones plutôt qu'un seul, avec des connexions complètes entre elle et la couche cachée, comme avant. Chacun des dix neurones de sortie est affecté à l'étiquette d'une classe : le premier est pour le chiffre 0, le second pour 1 et ainsi de suite. Après le passage par le réseau de neurones, la sortie indiquerait le chiffre écrit dans l'image d'entrée.

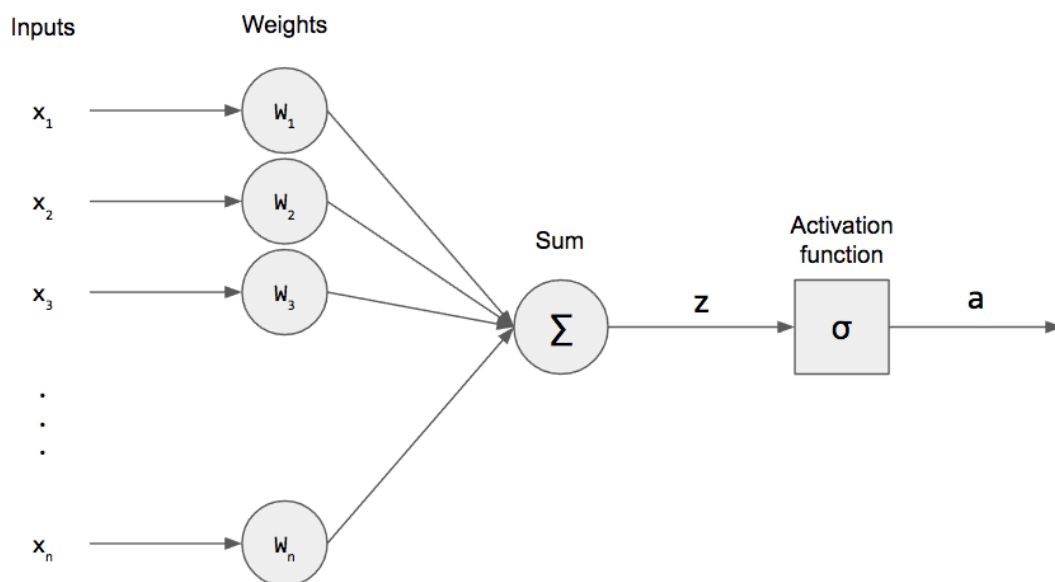


**Fig. 1.7** — classer les chiffres manuscrits

# Chapitre 2

## Implementation Python des fonctions logiques à partir du modèle de perceptron

Dans cette Partie on va essayé de modéliser les fonctions logiques avec des neurones.



**Fig. 2.1** — Modélisation les fonctions logiques avec le modèle de perceptron

### 2.1 Préparation du perceptron

On commence par importer les bibliothèques dont on aura besoin d'utiliser :

```
import numpy as np
```

```
def unitStep(v):  
    if v >= 0:  
        return 1
```

```
        else:
            return 0

def perceptronModel(x, w, b):
    v = np.dot(w, x) + b
    y = unitStep(v)
    return y
```

## 2.2 Implémentation de la fonction NOT

On crée la porte Not avec  $w_{NOT} = -1$  et  $b_{NOT} = 0.5$

```
def NOT_logicFunction(x):
    wNOT = -1
    bNOT = 0.5
    return perceptronModel(x, wNOT, bNOT)
```

## 2.3 Implémentation de la fonction AND

On crée la porte AND avec  $w_1 = w_{AND1} = 1$ ,  $w_2 = w_{AND2} = 1$ ,  $b_{AND} = -1.5$

```
def AND_logicFunction(x):
    w = np.array([1, 1])
    bAND = -1.5
    return perceptronModel(x, w, bAND)
```

## 2.4 Implémentation de la fonction OR

On crée la porte OR avec  $w_1 = 1$ ,  $w_2 = 1$ ,  $b_{OR} = -0.5$

```
def OR_logicFunction(x):
    w = np.array([1, 1])
    bOR = -0.5
    return perceptronModel(x, w, bOR)
```

## 2.5 Implémentation de la fonction XOR

On crée la porte XOr avec appel des autres portes comme suit :

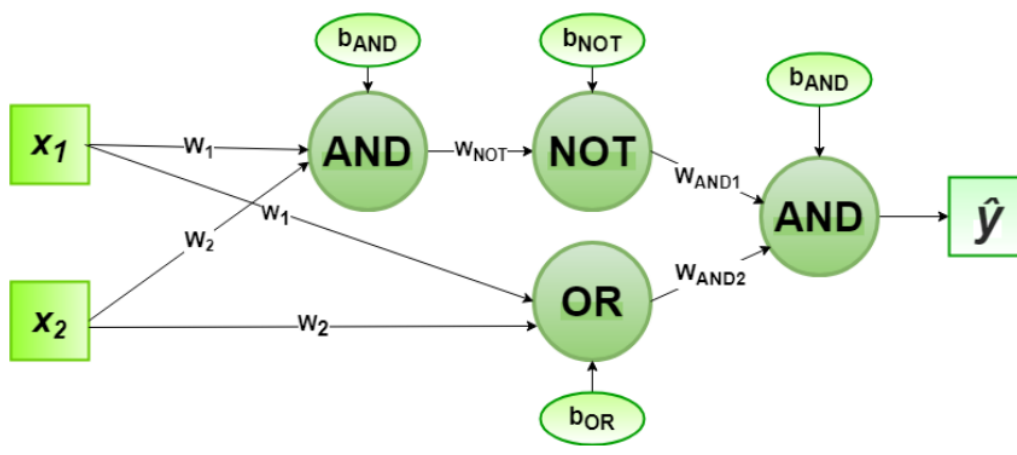


Fig. 2.2 – Conception de la fonction Xor à partir des perceptron AND, NOT, OR et AND

```

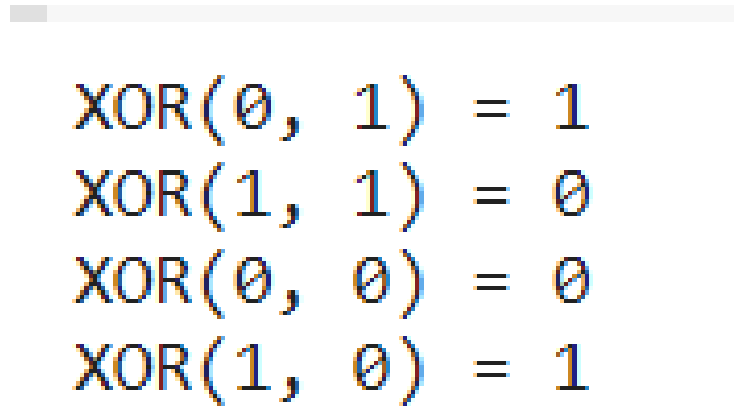
def XOR_logicFunction(x):
    y1 = AND_logicFunction(x)
    y2 = OR_logicFunction(x)
    y3 = NOT_logicFunction(y1)
    final_x = np.array([y2, y3])
    finalOutput = AND_logicFunction(final_x)
    return finalOutput

test1 = np.array([0, 1])
test2 = np.array([1, 1])
test3 = np.array([0, 0])
test4 = np.array([1, 0])

print("XOR({}, {}) = {}".format(0, 1, XOR_logicFunction(test1)))
print("XOR({}, {}) = {}".format(1, 1, XOR_logicFunction(test2)))
print("XOR({}, {}) = {}".format(0, 0, XOR_logicFunction(test3)))
print("XOR({}, {}) = {}".format(1, 0, XOR_logicFunction(test4)))

```

Interprétation :


$$\begin{array}{lcl} \text{XOR}(0, 1) & = & 1 \\ \text{XOR}(1, 1) & = & 0 \\ \text{XOR}(0, 0) & = & 0 \\ \text{XOR}(1, 0) & = & 1 \end{array}$$

**Fig. 2.3** – Résultat final obtenu

- On remarque que la porte XOr est bien conçue par l'utilisation des des perceptron AND, NOT, OR et AND.

# Conclusion

J'ai réussi dans ce travail à se familiariser avec les fondements mathématiques des réseaux de neurones et de l'explorer dans le cadre des exemples basiques de portes logiques.

En effet, le travail de documentation que j'ai fait m'a permis de faire face à la problématique de conception réseaux de neurones des présentée dans le cadre de l'atelier de l'optimisation et d'implémenter l'étude théorique en code exécutable.

Souheib Ben Mabrouk.

# Bibliographie

- [1] A comprehensive guide to convolutional neural networks. <https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53>.
- [2] Convolutional neural network. <https://datascientest.com/convolutional-neural-network>.
- [3] Introducing deep learning and neural networks — deep learning for rookies (1). <https://towardsdatascience.com/introducing-deep-learning-and-neural-networks-deep-learning-for-rookies-1-bd68f9cf588>