Piscine Machine Learning | Jour 3

Création d’un neurone et d’un réseau de neurone

Nom de repo: ml_d03

Droits: florian.bacho@epitech.eu, voravo_d, girard_z

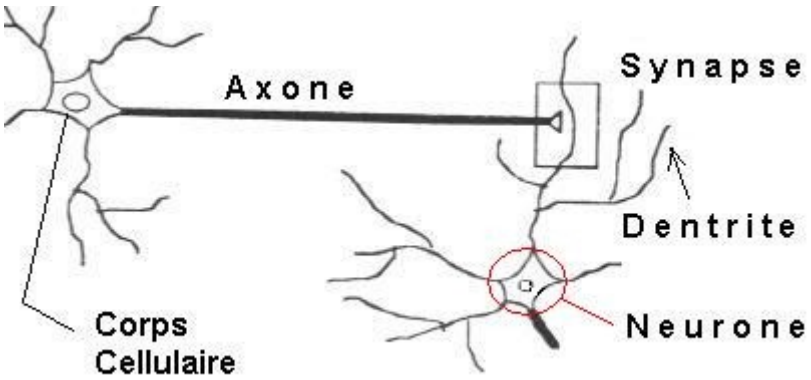
Langage : Python

Introduction

« Les neurones reçoivent les signaux (impulsions électriques) par des extensions très ramifiées de leur corps cellulaire (les dendrites) et envoient l'information par de longs prolongements (les axones). Les impulsions électriques sont régénérées pendant le parcours le long de l'axone. La durée de chaque impulsion est de l'ordre d'1 ms et son amplitude d'environ 100 mV.

Les contacts entre deux neurones, de l'axone à une dendrite, se font par l'intermédiaire des synapses. Lorsqu'une impulsion électrique atteint la terminaison d'un axone, des neuromédiateurs sont libérés et se lient à des récepteurs post-synaptiques présents sur les dendrites. L'effet peut être excitateur ou inhibiteur.

Chaque neurone intègre en permanence jusqu'à un millier de signaux synaptiques. Ces signaux n'opèrent pas de manière linéaire : il y a un effet de seuil. »



Notre objectif aujourd’hui est de créer un neurone puis de faire des assemblements de neurones afin de créer un réseau de neurones.

Chaque neurone possède k entrées que l’on note x_i. À chaque entrée est associé un « poids synaptique » noté w_i, qui est simplement un coefficient donnant plus ou moins d’importance à une information.

Un neurone artificiel va effectuer une somme pondérée de ses entrées au lieu de les considérer séparément. Cette somme pondérée se nomme « pré-activation ». La formule est la suivante :

$$sp=(\sum_{i=1}^{k} w_i x_i)+w_0$$

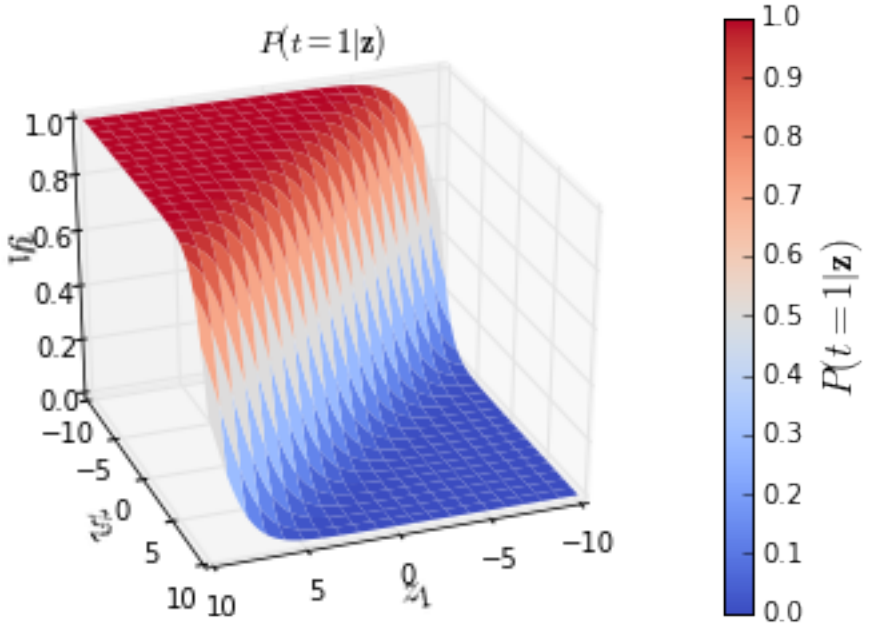
où w0 est un poids supplémentaire nommé « biais », lié à une entrée toujours égale à 1. En d'autres termes, cette valeur correspond à une constante.

Ce poids est l'équivalent de c dans la formule :

$$f(x,y)=ax+by+c$$

Chaque neurone possède une sortie qui va correspondre à l'activation du neurone (0 ou 1).

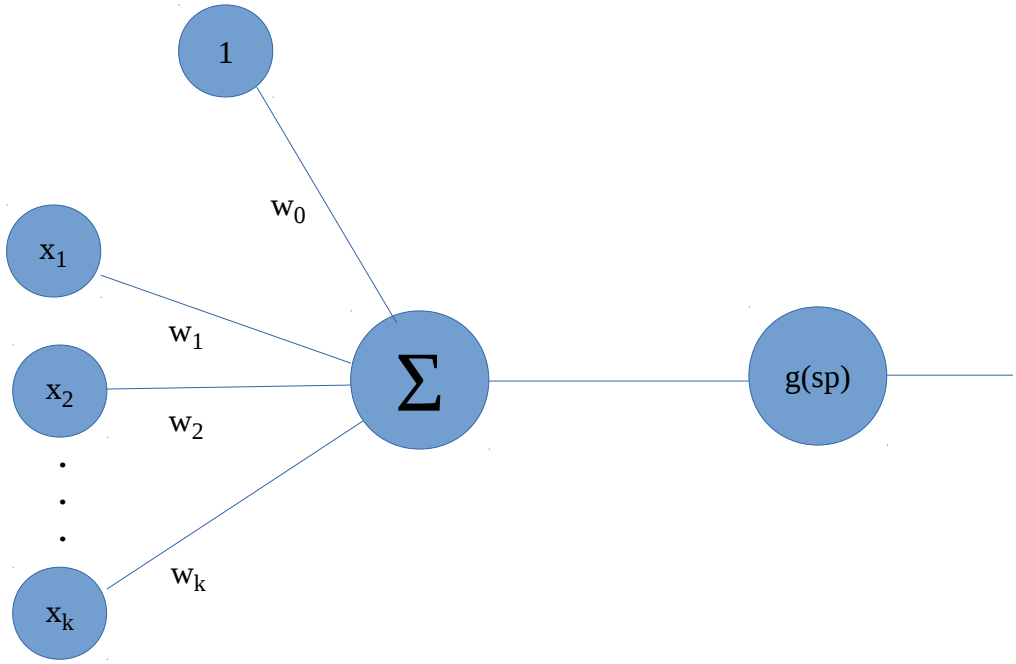
Souvenez-vous, lors du jour 1 nous avons vu la fonction logistique ou Sigmoïde qui donne une valeur comprise entre 0 et 1. Cette fonction a aussi la particularité de séparer l'espace en 2.



> **!** A noter qu'il existe d'autres fonctions d'activation possibles. Cependant, nous allons nous concentrer uniquement sur la fonction logistique pour le moment.

Ainsi, la pré-activation doit être passée dans la fonction d'activation (ici la Logistique) pour obtenir l'activation finale a de notre neurone:

$$a=Logistic(sp)$$



Où g(x) = Logistic(x)

Schéma d'un neurone artificiel

Ex01 :

Fichier: Neuron.py

Créez une class *Neuron*.

Son constructeur doit prendre en paramètre le nombre d'entrées du neurone et initialiser les poids synaptiques aléatoirement (vous verrez pourquoi plus tard !). Initialisez-les sur un intervalle relativement faible ([-0.5, 0.5] / [-2, 2] … il n'y a pas de règle universelle pour l'initialisation).

Cette class doit contenir une fonction *activate* avec le prototype suivant:

```
def activate(self, entries)
```

Où entries est une list de float

Ex02 :

Fichier: or.py

Créez un neurone à 2 entrées et trouvez les bons poids synaptiques pour qu'il donne une approximation de la porte OR.

Affichez vos résultats sur MatPlotLib.

Ex03 :

Fichier: and.py

Créez un neurone à 2 entrées et trouvez les bons poids synaptiques pour qu'il donne une approximation de la porte AND.

Affichez vos résultats sur MatPlotLib.

Ex04 :

Fichier: xor.py

Créez un neurone à 2 entrées et essayez de trouvez les bons poids synaptiques pour qu'il donne une approximation de la porte XOR.

Affichez vos résultats sur MatPlotLib.

> ! Ne pas y passer plus de 10 minutes ! Vraiment.

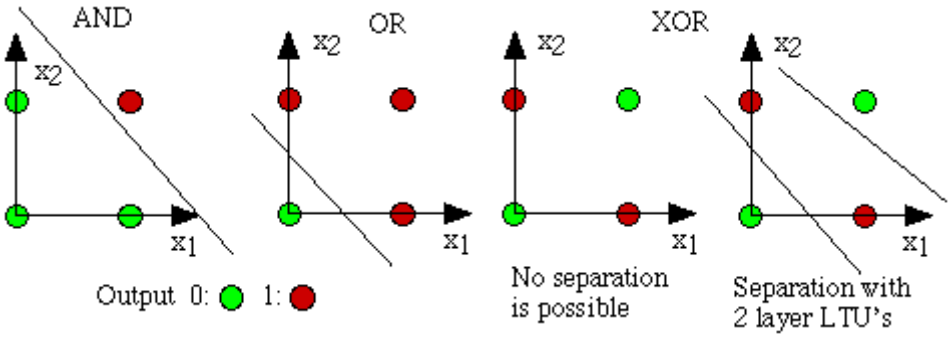
Ex05 :

Fichier: NeuralNetwork.py

Comme vous avez pu le remarquer, faire un XOR avec un neurone est impossible.

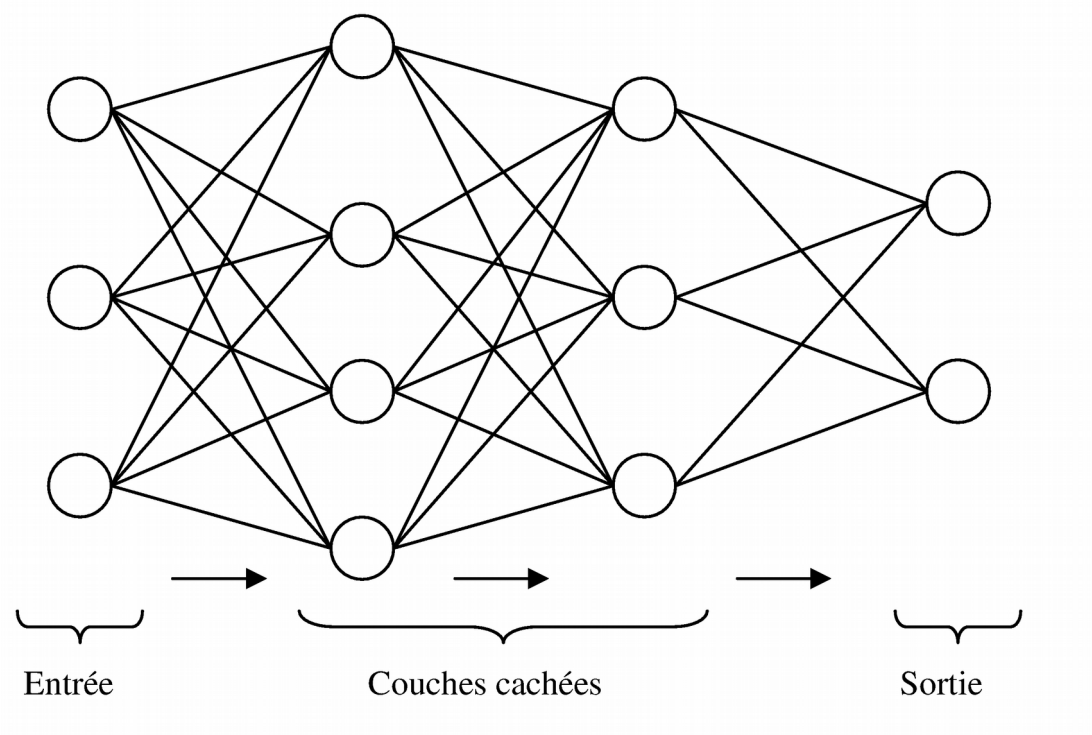
En effet, le OR et le AND est séparable par une droite. On dit que la donnée est linéairement séparable.

Cependant le XOR n’est pas linéairement séparable. Voiçi un schéma pour vous en convaincre:



Pour résoudre ce problème, on doit assembler plusieurs neurones en « couches » afin de réaliser des séparations plus complexes.

Ainsi, nous allons ajouter des couches dites « cachées » selon le schéma suivant :



Chaque neurone est relié à tous les neurones de la couche précédente. Les neurones d'entrées ne font pas de calcul. Ils sont uniquement présents pour amorcer la connection avec la première couche cachée. La couche de sortie, quand à elle, réalise la classification finale du réseau.

Créer une class *NeuralNetwork* contenant les assemblements de neurones.
Le constructeur prend en paramètre une list décrivant le nombre de neurone par couche.

Exemple: NeuralNetwork([3, 5, 2]) pour un réseau comprenant 3 entrées, 5 neurones cachés et 2 neurones de sortie.

Cette class doit contenir une fonction *activate* ayant le prototype suivant:

```
def activate(self, entries)
```

Où entries est une list de floatant.

Les neurones d'entrée prennent les valeurs du paramètre *entries* avant de propager les activations jusqu'à la couche de sortie.

Ex06 :

Fichier: xor.py

Créez un réseau de neurone comportant 2 neurones d’entrée, 2 neurones cachés et 1 neurone de sortie.
Trouver les bons poids synaptiques pour faire une approximation de la porte XOR.