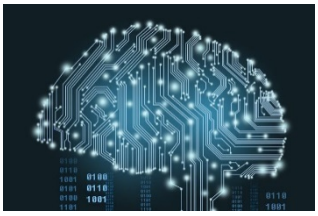
Neurone à plusieurs entrées

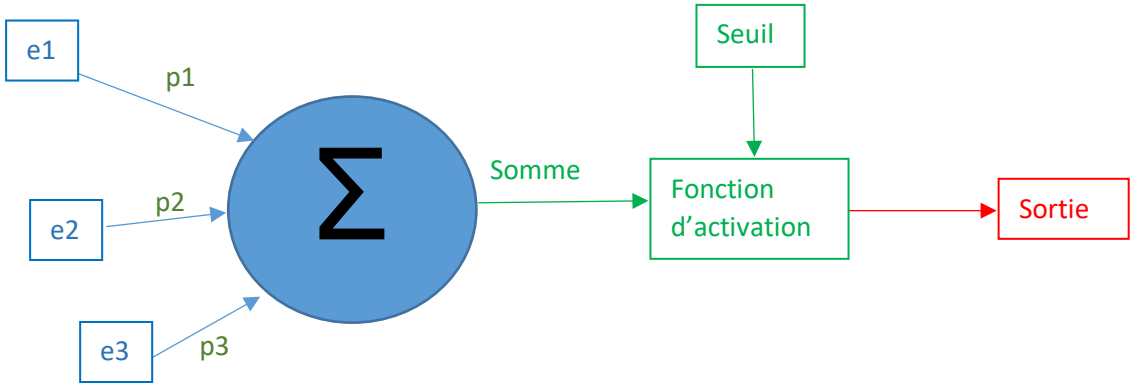


Classe de neurone à 3 entrées

Tu as vu précédemment qu'il était possible de modéliser avec une programmation orientée objet un neurone (voir activité mon premier neurone).

Simplement dans la définition de la classe de ce neurone, les attributs étaient fixes, ici par exemple, on a fixé que 3 poids synaptiques, donc un neurone à uniquement 3 synapses, donc 3 entrées.

Neurone précédent à 3 entrées (e1, e2, e3) :



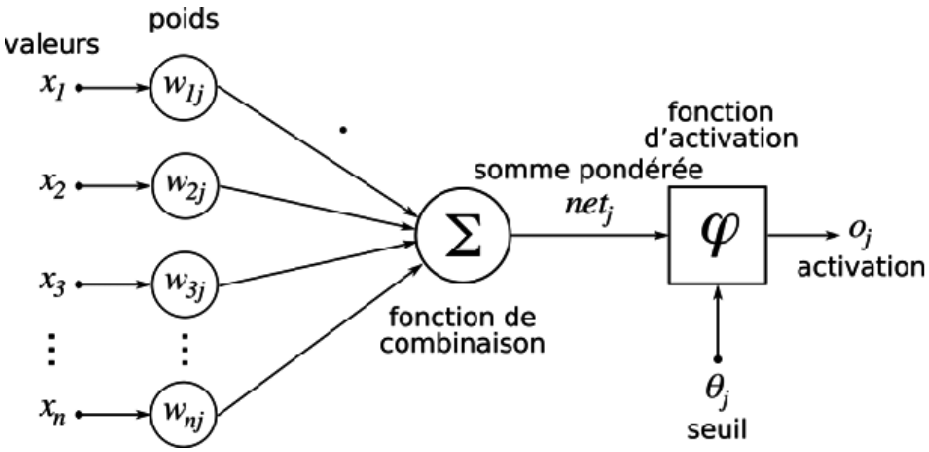
$$Somme = e1 \times p1 + e2 \times p2 + e3 \times p3$$

Mais maintenant, comment faire si l'on veut avoir un neurone à 6, 10 entrées ?

Il faudrait alors modifier à chaque fois le code de la classe d'un neurone, pas très pratique.

Nouvelle classe de neurone pour n d'entrées

Nous voulons maintenant créer une classe neurone qui soit flexible et qui accepte un nombre quelconque d'entrées et de poids synaptiques.



Utilisation des listes ou tableau

Nous allons utiliser alors la notion de liste ou de tableau qui nous permettra la souplesse voulue.

Une liste, ou un tableau, est un ensemble de données accessibles via un indice qui commence toujours à 0.

Exemple de tableau qui donnerait les poids synaptiques p1=0.2, p2=0.5, p3=0.4

```
poids_synaptiques = [0.2, 0.5, 0.4]
print("p1={0} p2={1} p3={2}".format(poids_synaptiques[0], poids_synaptiques[1], poids_synaptiques[2]))
```

```
>>>
p1=0.2 p2=0.5 p3=0.4
>>>
```

Ici nous avons créer un tableau à 3 poids synaptiques, mais maintenant il est facile d'en ajouter ou d'en enlever de la liste.

Pour balayer une liste complète, il suffit alors de réaliser une boucle itérative bornée :

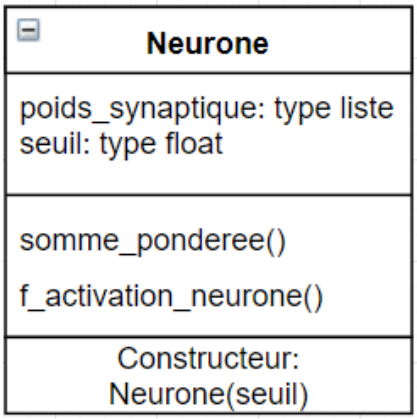
```
poids_synaptiques = [0.2, 0.5, 0.4]

for indice_tableau in range(len(poids_synaptiques)):
    print("p{0}={1}".format(indice_tableau, poids_synaptiques[indice_tableau]))
```

```
>>>
p0=0.2
p1=0.5
p2=0.4
>>>
```

On obtient bien la même chose qu'avant, mais de façon bien plus souple, car on peut modifier le nombre d'entrées sans changer de code dans la classe.

Modification du diagramme de classe Neurone



Modification du code de la classe du neurone.

Modification des attributs de la classe

Ton ancienne classe comprenait 4 attributs :

3 attributs de poids (p1, p2, p3) et 1 attribut seuil.

Il suffit alors de supprimer les 3 attributs de poids et de les remplacer par un seul attribut poids de type liste (vide).

Ainsi l'attribut poids pourra être maintenant définit comme une liste du type [0.2, 0.5, 0.6] qui dans cet exemple permet de définir 3 poids. Mais maintenant nous ne sommes plus limités, et surtout on ne modifie plus le code (les attributs) de la classe Neurone.

Modification de la somme pondérée

Tu dois maintenant réaliser une boucle itérative bornée sur le nombre d'entrées ou de poids

Pour trouver le nombre d'entrées ou de poids synaptiques (informations identiques), il suffit de trouver la taille de la liste avec la fonction len.

Exemple :

```
>>> a=[0.2,0.3,0.5,0.8]
>>> print(len(a))
4
```

Fonction d'activation du neurone

Il suffit juste de mettre en paramètre la liste des entrées au lieu des entrée (e1, e2, e3). Le neurone aura ainsi la possibilité d'avoir n entrées.

Instanciation d'un neurone pour essais.

L'instanciation ne change pas.

Il faudra juste choisir les poids synaptiques, soit en les fixant dans une liste, soit en les recherchant par une méthode d'apprentissage.

Création d'un test automatique pour validation du neurone.

Pour pouvoir tester un neurone qui a n entrées sur notre cas qui est une image à n pixels, il faut créer les 2^n Images différentes qui ont chacune n pixels. Cela est un cas particulier bien évidement.

Pour créer ces images automatiquement, il suffit de réaliser une boucle de 0 à n-1 et transformer l'indice de boucle en binaire avec la fonction bin.

```
>>> print(bin(5))
0b101
```

Cette donnée est de type str

```
>>> print(type(bin(5)))
<class 'str'>
```

Il suffit alors de supprimer les caractères '0b' devant avec :

```
>>> print(bin(5)[2:])
101
```

Qui permet de prendre toutes les données de la chaine de caractère (qui est une liste de string en fait) à partir de l'indice 2.

Mais la fonction d'activation du neurone admet en argument une liste d'entiers [1, 0, 1]

Il faudra donc convertir la chaine de caractères (ici '101') en une liste d'entiers [1, 0, 1].

Mais attention il faudra aussi ajouter si nécessaire des 0 devant pour avoir une image sur n pixels tout le temps.

Exemple : si on travaille sur des images à 4 pixels. Pour le codage de 5 on a vu que l'on avait '101'. Comme il y a 4 pixels, il faut ajouter un 0 pour avoir la liste [0, 1, 0, 1]

On pourra créer une fonction particulière format_bin(image_brute, nombre_pixels).

Cette fonction renverra une liste de nombre entiers (0 ou 1).à partir de l'image brute.

Dans notre exemple avec 5

Image_brute='101'

Nombre_pixels = 4

Format_bit('101',4) = [0, 1, 0, 1]

Pour créer une liste on pourra utiliser la fonction append(liste) qui permet d'ajouter une donnée à la fin d'une liste donnée.

On pourra commencer par ajouter les 0 et ensuite la valeur binaire dans deux boucles itératives bornées.

Exemple de résultats de tests pour n=3 comme dans la précédente activité

```
#on instancie l'objet neurone1 avec le seuil=0.7
neurone1=neurone(0.7)
#on fixe les attributs du neurone avec ces poids dans une liste
neurone1.poids_synaptique = [-3, -3, 1]
```

```
>>>
image d'entrée= [0, 0, 0] sortie= 0
image d'entrée= [0, 0, 1] sortie= 1
image d'entrée= [0, 1, 0] sortie= 0
image d'entrée= [0, 1, 1] sortie= 0
image d'entrée= [1, 0, 0] sortie= 0
image d'entrée= [1, 0, 1] sortie= 0
image d'entrée= [1, 1, 0] sortie= 0
image d'entrée= [1, 1, 1] sortie= 0
>>>
```