Atelier d'optimisation :

Convolutional Neural Network CNN

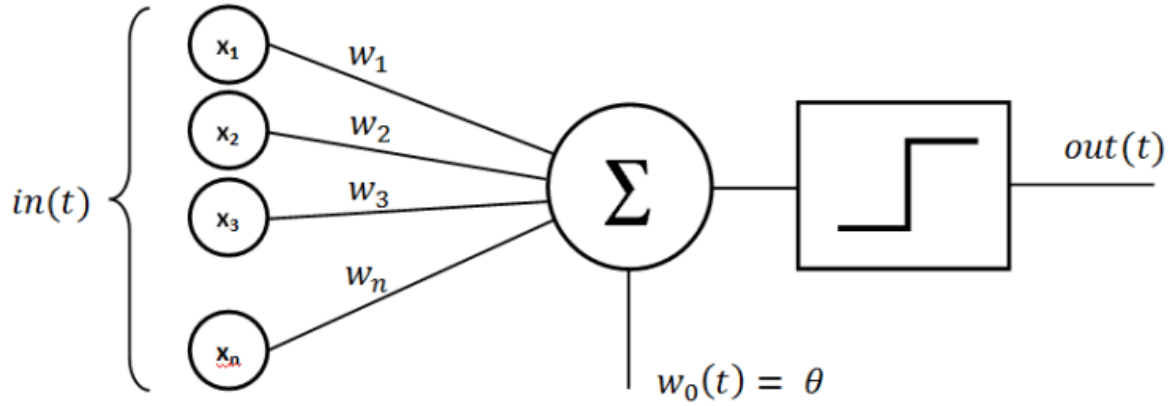
Réalisé par :

Abdelkader MILADI

INDP2 F

Le Perceptron

Le **perceptron** est un algorithme d'apprentissage supervisé de classifieurs binaires (c'est-à-dire séparant deux classes).
Il s'agit d'un neurone formel muni d'une règle d'apprentissage qui permet de déterminer automatiquement les poids synaptiques de manière à séparer un problème d'apprentissage supervisé. Si le problème est linéairement séparable, un théorème assure que la règle du perceptron permet de trouver une séparatrice entre les deux classes.



Un perceptron à *n* entrées (x1,..xn) et à une seule sortie *o* est défini par la donnée de *n* poids (ou coefficients synaptiques) (w1,..wn) et un biais (ou seuil) theta par:

$$o = f(z) = \begin{cases} 1 & \text{si} \quad \sum_{i=1}^{n} w_i x_i > \theta \\ 0 & \text{sinon} \end{cases}$$

La sortie *o* résulte alors de l'application de la fonction de Heaviside au potentiel post-synaptique :

$$z = \sum_{i=1}^{n} w_i x_i - \theta, \text{ avec:}$$

$$\forall x \in \mathbb{R},\ H(x) = \begin{cases} 0 & \text{si} \quad x < 0 \\ 1 & \text{si} \quad x \geq 0. \end{cases}$$

Cette fonction non linéaire est appelée fonction d'activation.

L'algorithme du perceptron de Rosenblatt est un cas particulier de l'algorithme du gradient stochastique utilisant comme fonction objectif

$$C(W) = \sum_{i \in \mathcal{M}} y_i \langle W, X_i \rangle, \text{ où } \mathcal{M} \text{ est l'ensemble des exemples mal classés ; et un taux d'apprentissage de } 1$$

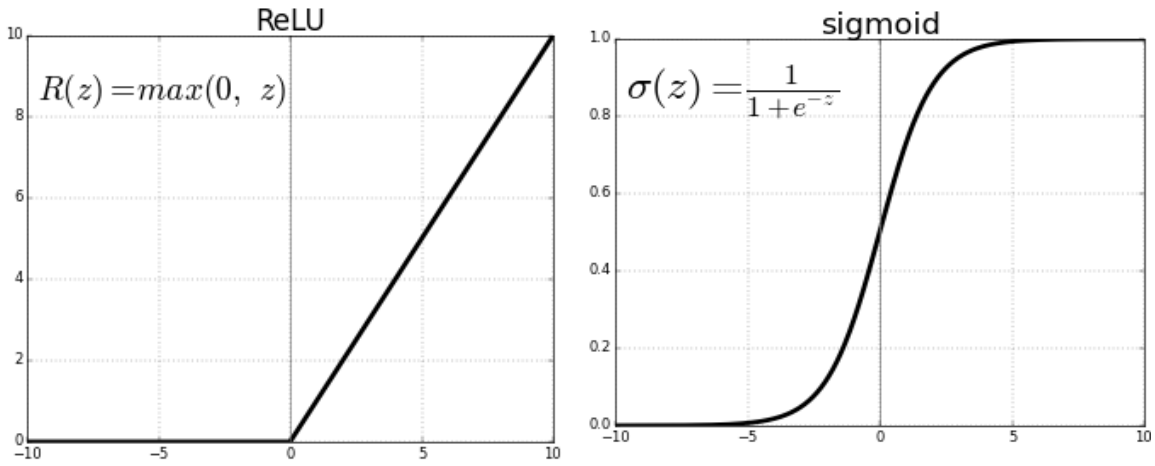
Fonction d'activation

Dans le domaine des réseaux de neurones artificiels, la fonction d'activation est une fonction mathématique appliquée à un signal en sortie d'un neurone artificiel.

Le terme de "fonction d'activation" vient de l'équivalent biologique "potentiel d'activation", seuil de stimulation qui, une fois atteint entraîne une réponse du neurone. La fonction d'activation est souvent une fonction non linéaire.

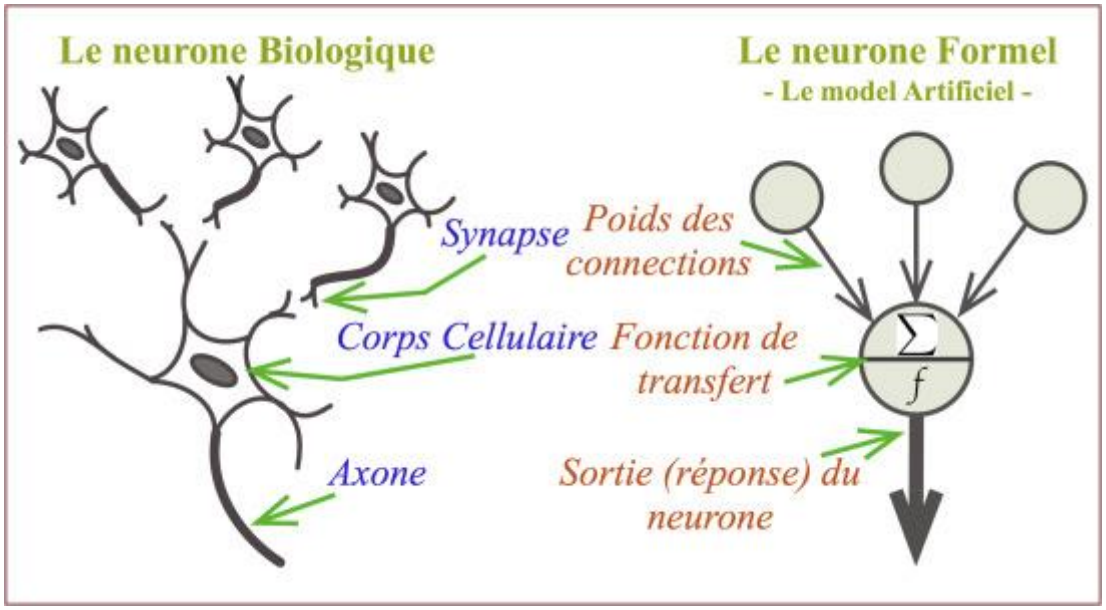
Un exemple de fonction d'activation est la fonction de **Heaviside**, qui renvoie tout le temps 1 si le signal en entrée est positif, ou 0 s'il est négatif.

Exemple de fonction d'activation :



Réseau neuronal convolutif

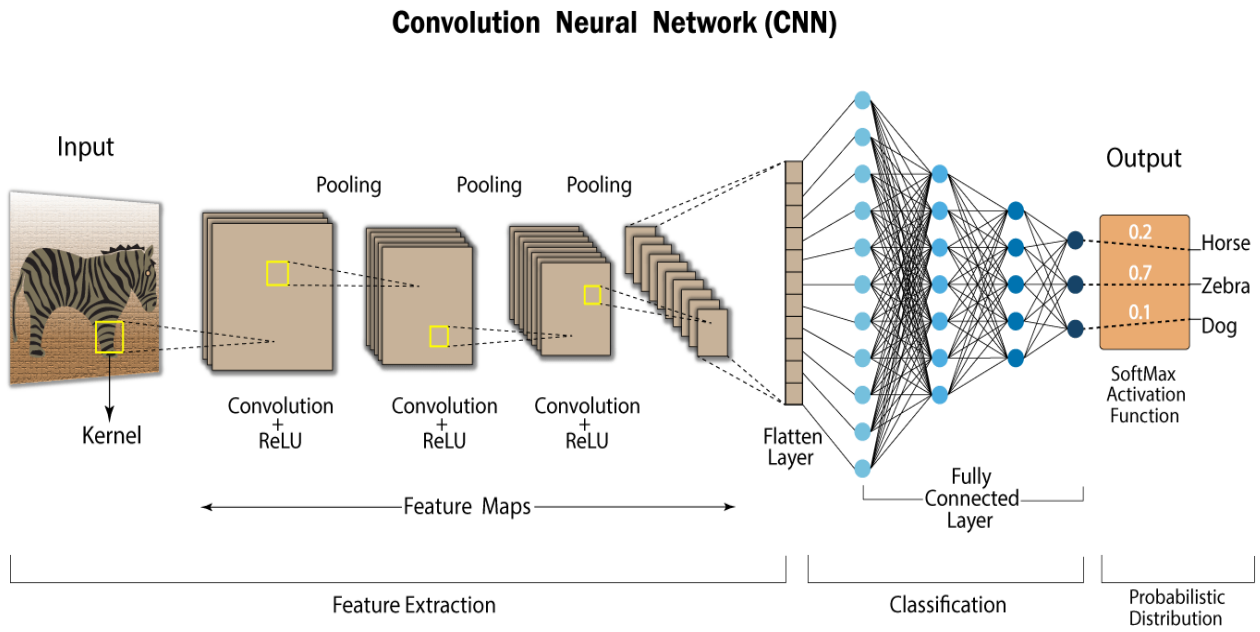
En apprentissage automatique, un réseau de neurones convolutifs ou réseau de neurones à convolution (en anglais CNN ou ConvNet pour convolutional neural networks) est un type de réseau de neurones artificiels acycliques, dans lequel le motif de connexion entre les neurones est inspiré par le cortex visuel des animaux.



Les neurones de cette région du cerveau sont arrangés de sorte qu'ils correspondent à des régions qui se chevauchent lors du pavage du champ visuel. Leur fonctionnement est inspiré par les processus biologiques, ils consistent en un empilage multicouche de perceptrons, dont le but est de prétraiter de petites quantités d'informations.

Au cours des dernières décennies, le Deep Learning s'est avéré être un outil très puissant en raison de sa capacité à gérer de grandes quantités de données. L'intérêt d'utiliser des couches cachées a dépassé les techniques traditionnelles, notamment en reconnaissance de formes. L'un des réseaux de neurones profonds les plus populaires est les réseaux de neurones convolutifs.

Exemple d'un réseau neuronal convolutif :



Avantages et Inconvénients

• Avantages :

Les principaux points forts des CNN sont de fournir un réseau dense efficace qui effectue efficacement la prédiction ou l'identification, etc. La puissance de CNN est de détecter des caractéristiques distinctes des images toute seules, sans aucune intervention humaine réelle.

• Inconvénients :

Malgré la puissance et la complexité des ressources des CNN, ils fournissent des résultats approfondis. À la base de tout cela, il s'agit simplement de reconnaître des motifs et des détails si infimes et discrets qu'ils passent inaperçus à l'oeil humain. Mais lorsqu'il s'agit de comprendre le contenu d'une image, cela échoue.

Implémentation du perceptron

```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
```

```
def RELU(x):
    if x<0:
        return 0
    else:
        return 1
```

```
training_set = [((0, 0), 0), ((0, 1), 1), ((1, 0), 1), ((1, 1), 1)]


plt.figure(0)
```

```
<Figure size 432x288 with 0 Axes>
<Figure size 432x288 with 0 Axes>
```

```
x1 = [training_set[i][0][0] for i in range(4)]
x2 = [training_set[i][0][1] for i in range(4)]
y = [training_set[i][1] for i in range(4)]
```

```
x1
```

```
[0, 0, 1, 1]
```

```
x2
```

```
[0, 1, 0, 1]
```

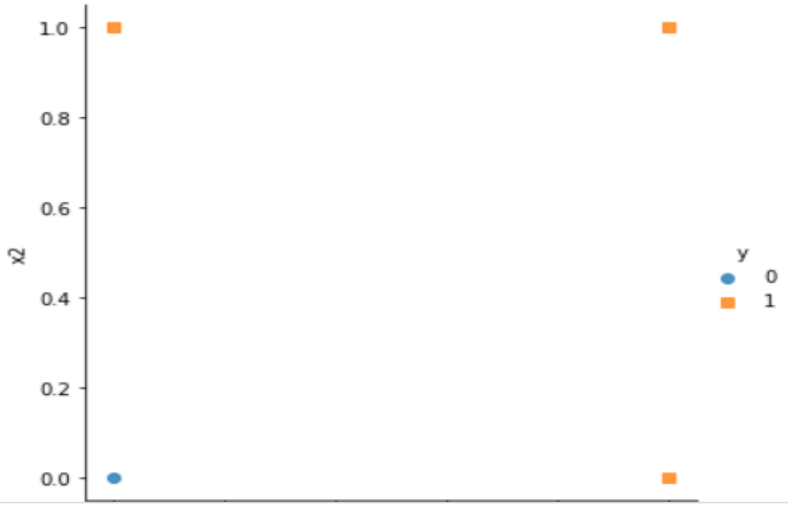
```
y
```

```
[0, 1, 1, 1]
```

```
df = pd.DataFrame(
    {'x1': x1,
     'x2': x2,
     'y': y
    })

sns.lmplot("x1", "x2", data=df, hue='y', fit_reg=False, markers=["o", "s"])
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarning:
  FutureWarning
<seaborn.axisgrid.FacetGrid at 0x7f7768b2cb50>
```




```
# parameter initialization
w = np.random.rand(2)
errors = []
eta = .5
epoch = 30
b = 0
```

```
w
```

```
array([0.10014983, 0.37067867])
```

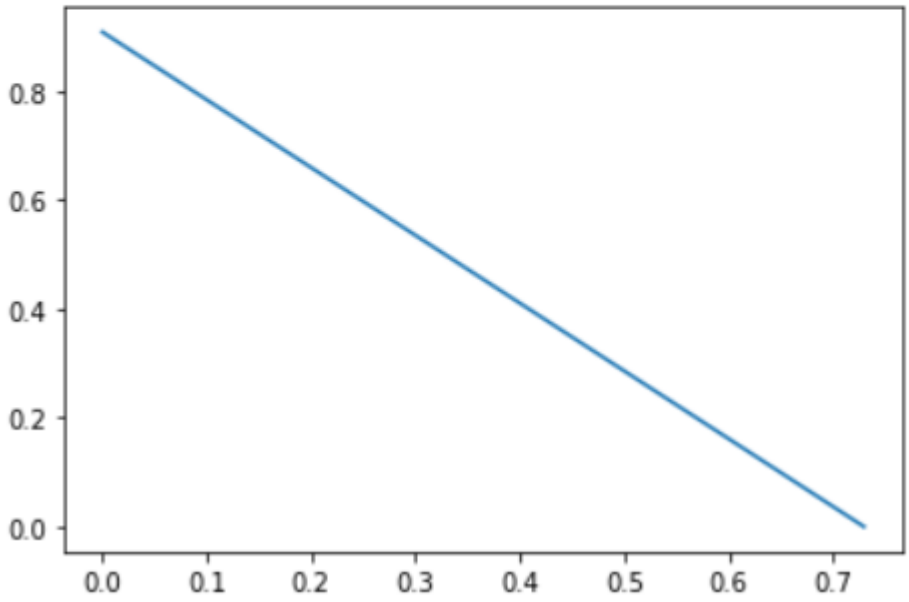
```
# Learning
for i in range(epoch):
    for x, y in training_set:
      # u = np.dot(x , w) +b
        u = sum(x*w) + b

        error = y - RELU(u)

        errors.append(error)
        for index, value in enumerate(x):
            #print(w[index])
            w[index] += eta * error * value
            b += eta*error
```

```
# final decision boundary
a = [0,-b/w[1]]
c = [-b/w[0],0]
plt.plot(a,c)
```

```
[<matplotlib.lines.Line2D at 0x7f77689d7990>]
```




```
# ploting errors
plt.figure(2)
plt.ylim([-1,1])
plt.plot(errors)
```

```
[<matplotlib.lines.Line2D at 0x7f7768befd50>]
```