Algorithmes des Réseaux de Neurones

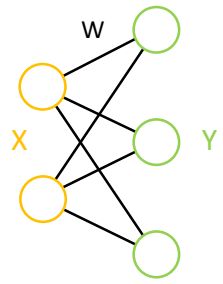
Objectif : Découvrir les algorithmes qui permettent le fonctionnement des réseaux de neurones sur un ordinateur (en Python)

Fonctionnement avec des matrices :

Les poids d'un réseau de neurones peuvent être représentés par des matrices et peuvent être utilisés pour calculer la valeur de sortie à partir de valeurs en entrée, grâce au produit matriciel.

Pour une couche de 3 neurones avec 2 entrées :

- Entrées : $X = [X_1 \quad X_2]$
- Poids : $W = \begin{bmatrix} W_{11} & W_{21} & W_{31} \\ W_{12} & W_{22} & W_{32} \end{bmatrix}$ (chaque colonne = poids du neurone)



La sortie de la couche doit ainsi être :

$$Y = [X_1 \cdot W_{11} + X_2 \cdot W_{12} \quad X_1 \cdot W_{21} + X_2 \cdot W_{22} \quad X_1 \cdot W_{31} + X_2 \cdot W_{32}]$$

Ce résultat peut être obtenu avec le produit matriciel entre X et W :

$$Y = X \cdot W$$

Les matrices permettent ainsi d'avoir une représentation simple des valeurs d'un réseau de neurones, mais aussi de pouvoir exécuter des calculs (en respectant les lois mathématiques déjà établies).

Initialisation : Création d'un Réseau de neurones

Un réseau de neurones peut s'écrire comme un *Objet* avec des listes qui contiennent les valeurs des poids, et des variables qui jouent le rôle de fonctions vectorisées (: la valeur de chaque élément d'une matrice est traitée individuellement) :

```
class Network:

    def __init__(self, layers, f): # layers: liste (ex:[2,3,1]), f: String ('sigmoid', 'tanh', 'relu', ...)

        # Crée les matrices des poids et des bias pour chaque couche :
        self.weights = [0.01*np.random.randn(x, y) for x, y in zip(layers[:-1], layers[1:])]
        self.biases = [0.01*np.random.randn(1, y) for y in layers[1:]]

        # Fonction d'activation et sa dérivée pour le gradient :
        self.f = activation[f]                # activation et activationPrime sont des dictionnaires
        self.fPrime = activationPrime[f]      # qui contiennent des fonctions vectorisées numpy
```

Propagation : Calculer la valeur de sortie d'un réseau à partir de valeurs X en entrées.

Cette fonction parcourt chaque couche en récupérant les valeurs des poids et des bias correspondant pour pouvoir appliquer l'étape de propagation pour chaque couche et ainsi pour tout le réseau :

```
def forward(self, X):
        Y = X  # Valeurs en entrée de la couche
        for w,b in zip(self.weights, self.biases):
            Y = Y.dot(w) + b  # Produit matriciel + bias
            Y = self.f(Y)     # Fonction d'activation

        return Y
```

(bias : entrée unitaire (=1) supplémentaire qui permet d'ajouter un poids pour chaque neurones pour leur donner un degré de liberté constant, comme le 'b' d'une fonction affine : $f(x) = ax + b$)

Notes :

- Utilisation de la librairie Numpy pour le calcul matriciel
- Algorithmes du coût et de la rétropropagation disponibles sur le site : machinelearning.ddns.net