MINI PROJET : LE PERCEPTRON

Description :

Ce projet peut être traité seul ou par un groupe de deux personnes.

Il sera rendu sur MBN.

Le nom du fichier sera du type "Nom1-Nom2.py" et les deux noms des élèves devront figurer au début du fichier.

Ce projet porte sur l'Intelligence Artificielle et plus précisément sur le perceptron, ancêtre du neurone, constituant de base des Réseau de neruones (Deep Learning).

- On fournit à la machine des données, on choisit un modèle, on utilise un algorithme d'optimisation pour ajuster le modèle et minimiser les erreurs.

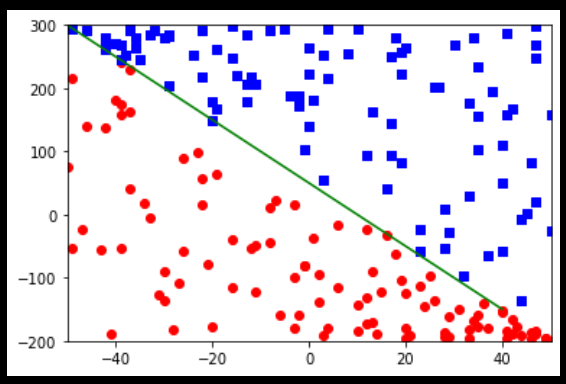
- Un neurone apprend tout seul, mais le Data Scientist lui fournit des hyperparamètres qui permettent de régler l'apprentissage :
    - Le pas d'apprentissage α.
    - Le biais θ.

- Le but du projet est de créer un perceptron qui nous permettra de répondre à différentes questions :
    - Séparer linéairement deux classes.
    - Estimer si une couleur est **rouge**.
    - Savoir si une personne survit au naufrage du Titanic.

Production : Classification simple

Présentation

Le but est de, modestement, apprendre à notre perceptron à déterminer la classe d'un point de coordonnées (e_1, e_2), c'est-à-dire savoir si un point est rouge (0) ou bleu (1), donc si ce point est au-dessus ou en-dessous de la droite verte.



On donne le programme Python *Fabrication_2_classes.py* qui permet de fabriquer le Dataset $((x, y), couleur)$, qui permet d'arriver à la figure ci-dessus.

```
# Spécification de la droite (pente,ordonnée à l'origine)
a, b = -5, 50
# valeur maximale de x et y
x_min, x_max = -50, 50
```

```
# Fabrication des points
NB_POINTS = 100
```

Les nombres a et b permettent de régler le coefficient directeur et l'ordonnée à l'origine de la droite.

Ce programme génère deux fichiers *rougebleu_train.csv* et *rougebleu_test.csv* qui contiennent les id, les coordonnées et la couleur des NB_POINTS points choisis.

Visualisation avec la bibliothèque *pandas*

	x	y	coul
0	39	-146	r
1	6	-128	r
2	20	-133	r
3	41	184	b
4	28	-106	r
..	..	...	...
195	-45	275	b
196	14	173	b
197	-23	60	r
198	-6	283	b
199	-6	101	b

Visualisation du fichier *.csv*

```
,x,y,coul
0,39,-146,r
1,6,-128,r
2,20,-133,r
3,41,184,b
4,28,-106,r
5,1,-7,r
6,-34,229,b
7,-23,277,b
8,18,-179,r
```

Chargement de la base

- Créer une fonction ***chargement_base(****nom_fichier****)*** qui prend en paramètre un nom de fichier *nom_fichier* et retourne un dictionnaire *jeu_tests*.

  Le dictionnaire aura :
  - Pour clé : les id des données.
  - Pour valeurs : une liste comprenant un tuple des $entrées$ (x, y) et la sortie connue (target) y.

  $$jeu\_tests = \{id\_data1: [(x, y), y],$$
  $$id\_data2 : [(x, y), y],$$
  $$\ldots\}$$

  Le séparateur dans le *fichier.csv* est la virgule.

- Cette fonction sera une ***méthode*** de la classe Perceptron de la $2^{ème}$ partie.

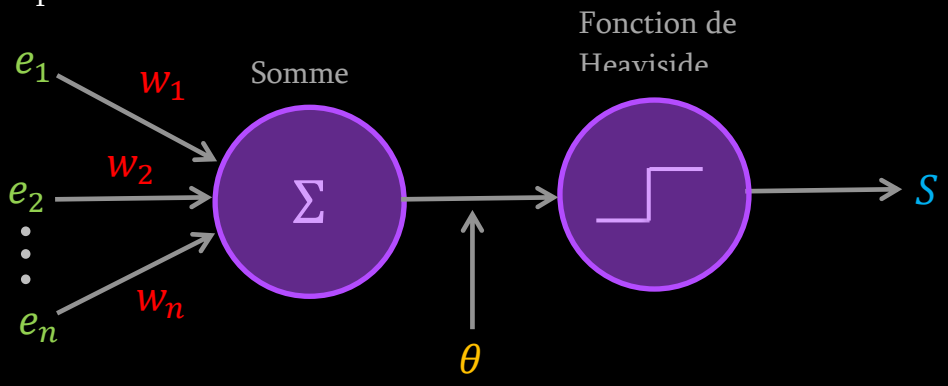
Le perceptron simple (ou monocouche) :

Définition

- Le premier modèle de neurone artificiel, nommé perceptron, a été inventé en 1957 par Frank Rosenblatt : c'est un algorithme d'apprentissage supervisé binaire, donc séparant deux classes.

- Ce neurone possède :
  - Des entrées (features) représentées par n nombres réels : e_1, e_2,…, e_n.
  - Des poids w_1, w_2,…, w_n représentés par des nombres réels qui s'appliquent à chaque entrée : c'est l'état du neurone.
  - Une sortie représentée par un entier S (ou $\hat{y}$) qui vaut 0 ou 1.
    Si la sortie vaut 1, on dit que le neurone est activé.
  - Un biais θ, que l'on fixera à 1, qui est le seuil de l'activation du neurone.

- Il se modélise par le schéma suivant :



Condition d'activation : $S = \begin{cases} 1 & si\ e_1.w_1 + e_2.w_2 + \cdots + e_n.w_n \geq \theta \\ 0 & sinon \end{cases}$

Principe de fonctionnement

- Le perceptron simple (ou monocouche) fonctionne sur un modèle basique de type classifieur supervisé linéaire (qui sépare deux classes grâce à une droite, un plan, un hyperplan).

- Comme il n'y a que deux valeurs possibles en sortie, notre perceptron ne pourra répondre que par « oui », « non » ou « bleu », « rouge » ou …

- Mathématiquement, si on se limite à 2 paramètres x, y (on se place dans le plan), cela revient à positionner le point de coordonnées (e_1, e_2) de part ou d'autre de la droite d'équation (frontière de décision) :

$$w_1 x + w_2 y = \theta \ ou\ y = -\frac{w_1}{w_2}x + \frac{\theta}{w_2}$$

Phase d'apprentissage : algorithme d'optimisation

- Au départ, on ne connait pas la frontière de décision qui répond au problème car on ne sait pas quel état (w_1, w_2) convient : il va donc falloir lui apprendre.

- On part d'un état initial du perceptron où on donne une valeur à chaque poids w_i, par exemple $(w_1, w_2, \dots, w_n) = (1,1, \dots,1)$.

- On fait ensuite évoluer son état jusqu'à trouver la valeur de $(w_1, w_2, \dots, w_n)$ qui répond au problème.

- Une étape d'entrainement se déroule de la manière suivante :
    - On donne des entrées $(e_1, e_2, .., e_n)$ ou features et l'objectif ou target ($y = 0$ ou $y = 1$) qui est la sortie attendue si le perceptron est parfaitement paramétré.
    - On calcule alors la sortie $S = 0$ ou $S = 1$ que renvoie le perceptron selon l'entrée $(e_1, e_2, .., e_n)$ dans son état actuel $(w_1, w_2, \dots, w_n)$.

- Plusieurs cas sont possibles :
    - Si l'objectif (target) y est égal à la sortie S alors le perceptron fonctionne bien pour cette entrée (feature), on ne change pas l'état du neurone.

    - Si la sortie calculée est $S = 0$ alors que l'objectif est $y = 1$, on change l'état actuel du perceptron $(w_1, w_2, \dots, w_n)$ en un nouvel état $(w_1', w_2', \dots, w_n')$ selon les formules suivantes :

$$w_1' = w_1 + \alpha e_1$$
$$w_2' = w_2 + \alpha e_2$$
$$\dots$$
$$w_n' = w_n + \alpha e_n$$

    - Si la sortie calculée est $S = 1$ alors que l'objectif est $y = 0$, on change l'état actuel du perceptron $(w_1, w_2, \dots, w_n)$ en un nouvel état $(w_1', w_2', \dots, w_n')$ selon les formules suivantes :

$$w_1' = w_1 - \alpha e_1$$
$$w_2' = w_2 - \alpha e_2$$
$$\dots$$
$$w_n' = w_n - \alpha e_n$$

- Le taux d'apprentissage α ($alpha$) est un réel petit qui est un hyperparamètre. On commencera, par exemple, avec $\alpha = 0.1$.

- On répète ces étapes avec plusieurs couples ($entrée\ (e_1, e_2, .., e_n)$, $objectif\ y$). Géométriquement, chaque apprentissage déplace la frontière de décision pour répondre au mieux au problème.

Mise en œuvre Python du perceptron

On cherche à programmer le perceptron en programmation objet (POO).

- On fait le choix de modéliser l'état et les entrées du perceptron par des listes.
- Créer une classe **Perceptron** ayant comme attributs passés en paramètres du constructeur :
  - *n* : nombres d'entrées (int)
  - *biais* : biais (float)
  - *alpha* : taux d'apprentissage ou learning rate (float)
- Ajouter un attribut *etat*, liste de *n* éléments initialisés à 1, qui représentera l'état donc les poids du perceptron dans l'état final.
- Activation du perceptron :
  - Créer la méthode ***activation(entrees,poids)*** qui selon les *poids* $[w_1, w_2, \dots, w_n]$ et les valeurs *entrees* $[e_1, e_2, .., e_n]$ renvoie $S = 1$ en cas d'activation et $S = 0$ sinon.
- Apprentissage du perceptron :
  - Créer la méthode ***apprentissage(entrees,poids,objectif)*** qui renvoie le nouvel état $[w'_1, w'_2, .., w'_n]$ modifié du perceptron après le résultat de l'activation, les *entrees* et l'*objectif* étant donnés.
  - On peut simplifier les étapes en renvoyant le nouvel état en calculant :

$$w'_1 = w_1 + signe \times alpha \times e_1$$
$$w'_2 = w_2 + signe \times alpha \times e_2$$
$$\dots$$
$$w'_n = w_n + signe \times alpha \times e_n$$

Avec :

- $signe = 0$ si la sortie S est égale à l'objectif y.
- $signe = +1$ si la sortie $S = 0$ et l'objectif $y = 1$.
- $signe = -1$ si la sortie $S = 1$ et l'objectif $y = 0$.

La méthode ***apprentissage( )*** fait donc appel à la méthode ***activation( )*** pour récupérer la valeur de S.

- Entrainement du perceptron :
  - Créer la méthode ***entrainement(jeu_tests)*** qui renvoie l'état final $[w_1, w_2, \dots, w_n]$ du perceptron entrainé par itération du dictionnaire *jeu_tests*.
  - Le *jeu_tests* sera donné sous la forme d'un dictionnaire dont les valeurs sont des tuples dont les éléments correspondent aux *entrées* $(e_1, e_2, .., e_n)$ et leurs objectifs y associés :

$$jeu\_tests = \{id\_data1: [(e_1, e_2, .., e_n), y],$$
$$id\_data2: [(e_1, e_2, .., e_n), y],$$
$$\dots\}$$

La méthode ***entrainement( )*** fait donc appel à la méthode ***apprentissage( )*** pour chaque donnée entrées $(e_1, e_2, .., e_n)$ et objectif y du jeu de tests.

- Interrogation du perceptron :
    - Créer la méthode ***reponse(entrees)*** qui prend en paramètre les *entrees* $[e_1, e_2, .., e_n]$ et renvoie $S = 1$ en cas de réponse positive et $S = 0$ sinon en se servant de la méthode ***activation(etat)*** une fois le perceptron entrainé.

- Performance du perceptron :
    - Créer la méthode ***metrique(jeu_tests)*** qui prend en paramètre le *jeu_tests* et renvoie l'exactitude (accuracy) du perceptron c'est-à-dire le pourcentage de prédictions correctes.

Test du perceptron

- Créer une instance perceptron de la classe **Perceptron**, l'entrainer avec le train set, dictionnaire fabriqué à partir du fichier *rougebleu_train.csv*.

- Tester la performance du modèle avec le test set, dictionnaire fabriqué à partir du fichier *rougebleu_test.csv*.

- Tester le perceptron avec un point appartenant à la catégorie bleue et un point appartenant à la catégorie rouge.

- Tester l'influence des hyperparamètres :
    - Pas d'apprentissage α.
    - Biais θ.

- Tester l'influence du nombre de points NB_POINTS du Dataset.