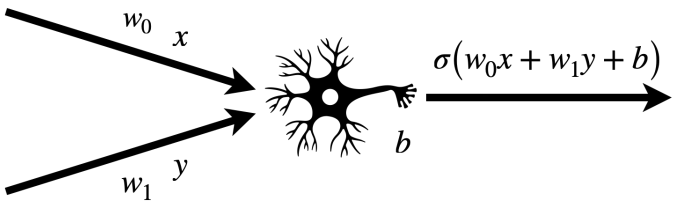
Perceptron

Le perceptron est l'un des premiers modèles de réseau de neurones artificiels. Il a été proposé à la fin des années 50 par le chercheur américain Frank Rosenblatt pour simuler la manière dont un neurone biologique traite l'information.

En informatique, un neurone est modélisé comme une unité de calcul simple : il reçoit plusieurs signaux en entrée, transmis par des « synapses ». Chaque synapse est associé à un poids, qui traduit l'importance de l'information qu'elle transporte. Le neurone additionne ensuite l'ensemble des signaux pondérés, puis applique une fonction d'activation qui détermine la sortie, souvent un nombre flottant dans l'intervalle $[0, 1]$.

On s'intéresse à un réseau de neurone où chaque neurone sera représenté comme ci-dessous.



On y trouve :

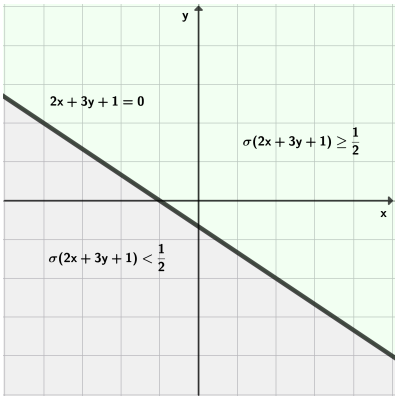
- l'intensité des signaux x et y ;
- les poids des synapses w_0 et w_1 ;
- le biais du neurone b ;
- la fonction d'activation la plus utilisée, la sigmoïde σ définie par

$$\forall x \in \mathbb{R}, \quad \sigma(x) = \frac{1}{1 + e^{-x}} \quad \text{et} \quad \sigma'(x) = \sigma(x)(1 - \sigma(x))$$

On considère un exemple avec $w_0 = 2$, $w_1 = 3$ et $b = 1$. Le plan (x, y) est alors séparé en deux régions par la droite d'équation :

$$2x + 3y + 1 = 0$$

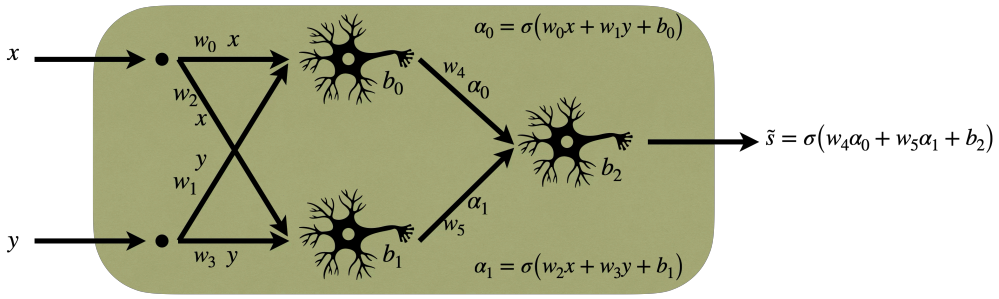
chaque région correspondant à une décision différente du perceptron selon la valeur du signal calculée avec σ.



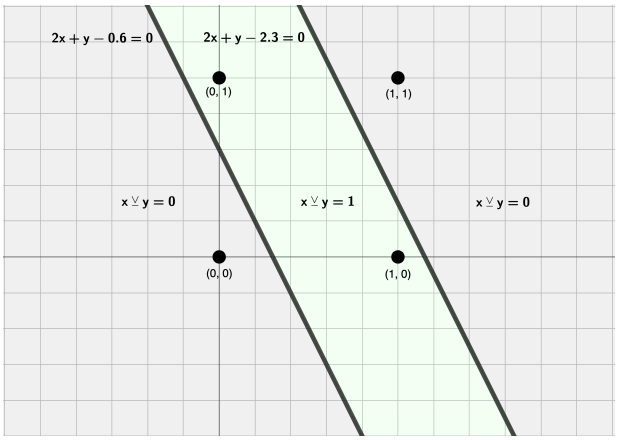
L'objectif de ce sujet est de simuler la fonction logique « ou-exclusive », noté $\veebar$, dont la table de vérité est donnée ci-dessous. On représente une valeur `True` par 1 et une valeur `False` par 0.

x	y	$x \veebar y$
0	0	0
0	1	1
1	0	1
1	1	0

On souhaite donc séparer le plan (x, y) en deux régions, l'une correspondant à la sortie 1 lorsque les entrées x et y sont différentes, et l'autre correspondant à la sortie 0 lorsque les deux entrées sont identiques. Cette séparation est impossible avec un perceptron simple. On s'intéresse alors au perceptron affine avec trois neurones.



Le perceptron devra apprendre à tracer une frontière de décision qui distingue les points $(0, 1)$ et $(1, 0)$, pour lesquels la sortie vaut 1, des points $(0, 0)$ et $(1, 1)$, pour lesquels la sortie vaut 0. Un exemple de cette séparation est illustré dans la figure ci-dessous.



Exercice 1 : Compréhension du projet

1) Renommer le fichier `Nom_Perceptron.py` en utilisant ton propre nom. Pour le moment, le résultat du programme n'est pas correcte, mais il doit s'exécuter sans erreur. Ouvrir le fichier et l'exécuter.

2) Le fichier `Nom_Perceptron.py` contient des variables globales et des fonctions prédéfinies :

- `w` est la liste des poids des synapses, `b` est la liste des biais des neurones, `alpha` est la liste des sorties intérmédiaires des neurones 0 et 1, `s_tilde` est la sortie finale du réseau ;
- la fonction `fonction_logique` code la fonction à apprendre avec le perceptron, dans notre cas, c'est la fonction logique ou-exclusive ;
- les fonctions `sigmoide` et `dsigmoide` correspondent respectivement à σ et σ'.

Observer le contenu du fichier et faire le lien avec les paramètres du perceptron.

Exercice 2 : Propagation

La fonction `propagation` permet de calculer la sortie du réseau de neurones à partir des valeurs d'entrée x et y. Elle simule le passage de l'information de la couche d'entrée vers la couche de sortie, en appliquant successivement les poids et les biais associés à chaque neurone, puis la fonction d'activation sigmoïde.

1) Calculer les sorties des neurones 0 et 1.

2) Calculer la sortie du réseau, c'est-à-dire la sortie du neurone 2.

Exercice 3 : Rétropropagation

La fonction `retropropagation` effectue la phase d'apprentissage du perceptron par rétropropagation de l'erreur. Elle met ainsi en œuvre le principe fondamental de l'*apprentissage supervisé* : à chaque exemple, elle ajuste les poids et les biais du réseau à partir de l'erreur observée entre la sortie attendue s et la sortie calculée $\tilde{s}$.

Concrètement, on commence par calculer l'erreur $e = s - \tilde{s}$. On met ensuite à jour les poids et le biais du neurone 2, puis ceux des neurones 0 et 1.

$$\begin{array}{rclrclrcl} \delta_2 & = & e \times \sigma'(\tilde{s}) & \delta_0 & = & \delta_2 \times w_4 \times \sigma'(\alpha_0) & \delta_1 & = & \delta_2 \times w_5 \times \sigma'(\alpha_1) \\ w_4 & = & w_4 + \delta_2 \times \alpha_0 & w_0 & = & w_0 + \delta_0 \times x & w_2 & = & w_2 + \delta_1 \times x \\ w_5 & = & w_5 + \delta_2 \times \alpha_0 & w_1 & = & w_1 + \delta_0 \times y & w_3 & = & w_3 + \delta_1 \times y \\ b_2 & = & b_2 + \delta_2 & b_0 & = & b_0 + \delta_0 & b_1 & = & b_1 + \delta_1 \end{array}$$

1) Calculer l'erreur.

2) Calculer δ_2.

3) Mettre à jour w_4, w_5 et b_2.

4) Calculer δ_0 et δ_1.

5) Mettre à jour w_0, w_1, b_0 ainsi que w_2, w_3 et b_1.

Exercice 4 : Apprentissage

La fonction `apprentissage` effectue une époque complète d'apprentissage, c'est-à-dire un passage sur l'ensemble des exemples d'entraînement, durant lequel le réseau ajuste ses poids et ses biais à partir de chacun d'eux.

L'ordre des quatre exemples $(0,0)$, $(0,1)$, $(1,0)$ et $(1,1)$ est mélangé à chaque appel afin de limiter les effets de répétition et éviter que le réseau n'apprenne une séquence fixe, ce qui favorise une meilleure généralisation de l'apprentissage.

1) Pour chacun des exemples, récupérer les entrées et calculer la sortie attendue.

2) Pour chacun des exemples, réaliser la propagation et la rétropropagation.

Exercice 5 : Expérience

La fonction `experience` initialise les poids et biais du réseau avec des valeurs aléatoires entre -1 et 1, puis lance successivement plusieurs phases d'apprentissage supervisé sur la fonction logique ou-exclusive, afin d'ajuster progressivement les paramètres du perceptron.

1) Initialiser les poids et les biais du réseau avec des valeurs aléatoires entre -1 et 1, à l'aide de la bibliothèque `random`.

2) Lancer l'apprentissage autant de fois que souhaité.

Exercice 6 : Test

Une fois que la phase d'apprentissage est réalisée, on teste le réseau sur les quatre combinaisons possibles des données de l'entrée afin de vérifier sa capacité à reproduire correctement la fonction logique visée.

1) Pour chacune des données, prédire le résultat : la prédiction vaut 1 si la sortie du réseau est supérieure ou égale à 0.5, 0 dans le cas contraire.

2) Si la prédiction est égale à la sortie attendue, incrémenter la variable `correct`.

Un exemple d'exécution du programme conduit aux valeurs suivantes pour les paramètres du perceptron. Et le graphique, obtenu à partir de ces paramètres, représente les droites de décision associées. Il montre que le plan (x, y) est correctement séparé en deux régions : la fonction logique renvoie 1 lorsque x et y sont différentes, et 0 lorsqu'elles sont identiques.

$$b_0 = 2.829 \quad b_1 = -7.582 \quad b_2 = 5.207$$

$$w_0 = -6.996 \quad w_1 = -7.023 \quad w_2 = 4.881 \quad w_3 = 4.885 \quad w_4 = -10.292 \quad w_5 = -10.721$$