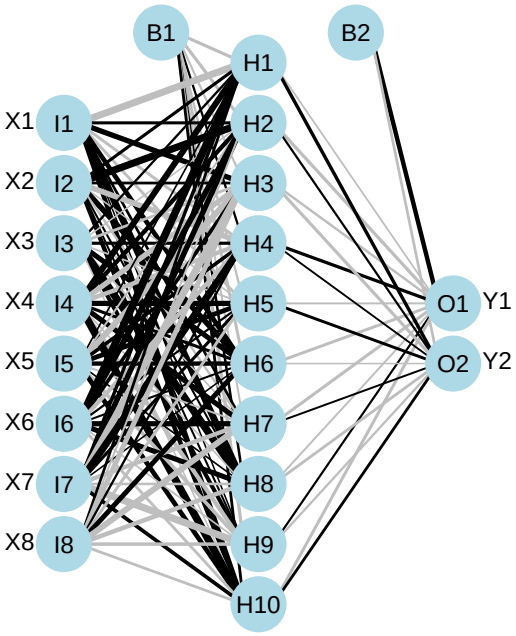
Université de Caen

Introduction aux réseaux de neurones

Arthur Delcroix - Marie Paccalet-Magat - Christophe Chesneau

https://chesneau.users.lmno.cnrs.fr/



Caen, le 14 Janvier 2024

Table des matières

~ Note ~

Ce document propose une introduction aux réseaux de neurones.

On y aborde principalement le perceptron simple.

Les logiciel utilisé sont Python et R.

N'hésitez pas à me contacter pour tout commentaire :

`christophe.chesneau@gmail.com`

Bonne lecture !

1 Introduction

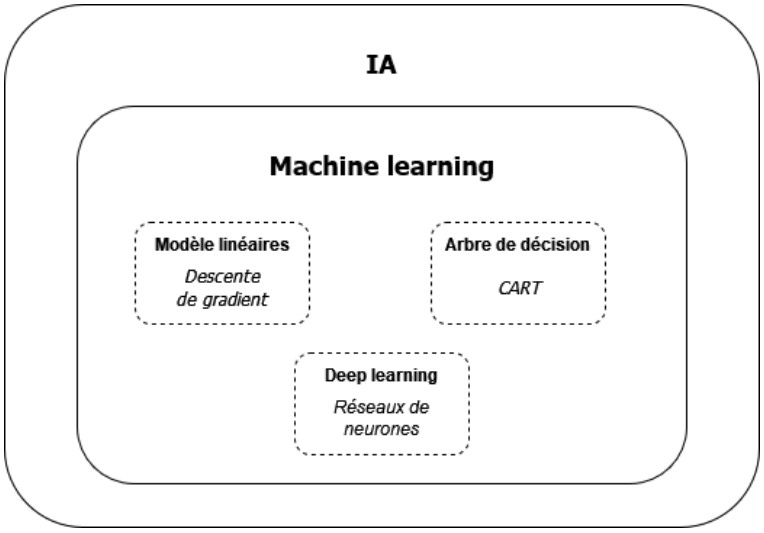
Le machine learning est une branche de l'Intelligence Artificielle qui consiste à programmer des machines afin qu'elles apprennent à accomplir des tâches, en analysant et en s'adaptant à partir d'exemples concrets (données).

En machine learning, pour l'apprentissage supervisé, la manière de procéder est toujours la même :

— On développe un modèle

— On utilise un algorithme d'optimisation qui minimise l'erreur entre le modèle et les données. C'est la phase apprentissage

— On cherche une classification pour de nouvelles données



La principale différence entre le Deep Learning et les autres méthodes de Machine Learning réside dans la nature de leurs modèles. Dans les méthodes plus traditionnelles, le modèle est généralement représenté par une fonction unique, tandis que dans le Deep Learning, le modèle est constitué d'un réseau de fonctions inter connectées, appelées neurones, qui travaillent de concert pour traiter les données et effectuer des prédictions. Inspirés par le fonctionnement du cerveau humain, ces réseaux sont caractérisés par leur capacité à apprendre des représentations hiérarchiques des données à différents niveaux d'abstraction. Contrairement aux méthodes classiques, qui nécessitent souvent une ingénierie de caractéristiques manuelle, le Deep Learning peut apprendre à détecter automatiquement des caractéristiques pertinentes à partir des données brutes, ce qui le rend particulièrement

efficace pour des tâches complexes et des volumes de données massifs.

`https://culturesciencesphysique.ens-lyon.fr/ressource/IA-apprentissage-Rousseau.xml#avantages-inconvenients`

Usage

— Données compliquées : Si vous avez des données comme des images, du texte ou du son qui sont un peu compliquées à comprendre, le Deep Learning pourrait être utile.

— Beaucoup de données : Si vous avez beaucoup de données, le Deep Learning peut vous aider à en tirer le meilleur parti et à obtenir de bonnes prédictions.

— Problèmes difficiles : Si vous avez un problème difficile à résoudre, le Deep Learning pourrait être une bonne option car il peut apprendre des modèles complexes.

Avantages

— Peut apprendre automatiquement : Avec le Deep Learning, vous n'avez pas toujours besoin de dire au modèle ce qu'il doit chercher. Il peut souvent le découvrir tout seul à partir des données.

— Performances élevées : Le Deep Learning peut donner de très bons résultats dans de nombreuses situations, surtout lorsque les problèmes sont complexes.

— Peut s'adapter facilement : Vous pouvez ajuster un modèle de Deep Learning pour différentes tâches en changeant juste un peu sa structure et en lui donnant de nouvelles données à apprendre.

Inconvénients

— Besoin de beaucoup de données : Le Deep Learning a besoin de beaucoup de données pour bien fonctionner, ce qui peut être un problème si vous n'en avez pas assez.

— Besoin de beaucoup de calcul : Pour entraîner un modèle de Deep Learning, vous avez souvent besoin d'un ordinateur très puissant, ce qui peut coûter cher.

— Pas toujours facile à comprendre : Parfois, les modèles de Deep Learning peuvent être difficiles à comprendre, ce qui peut être un problème si vous avez besoin de savoir comment ils prennent

leurs décisions.

Le Deep Learning peut être une bonne option pour des problèmes difficiles avec beaucoup de données, mais il peut être coûteux et parfois difficile à comprendre. Il vaut mieux peser le pour et le contre avant de décider de l'utiliser.

2 Rappels mathématiques : norme, produit scalaire, hyperplan, Descente de gradient

2.1 Norme

Définition

Une norme sur $\mathbb{R}^n$ est une application $N : \mathbb{R}^n \longrightarrow \mathbb{R}$ telle que :

1. N est à valeur postives ou nulles : $\forall x \in \mathbb{R}^n \quad N(x) \geq 0$
2. $\forall x \in \mathbb{R}^n \quad N(x) = 0 \Rightarrow x = 0_{\mathbb{R}^n}$
3. $\forall (\alpha, x) \in \mathbb{R} \times \mathbb{R}^n \quad N(\alpha x) = |\alpha| N(x) \quad$ (**homogénéité**)
4. $\forall (x, y) \in \mathbb{R}^{n2} \quad N(x + y) \leq N(x) + N(y) \quad$ (**inégalité trirangulaire**)

Notation

En général, on note une norme sur $\mathbb{R}^n$ à l'aide de doubles barres $\|.\| : \mathbb{R}^n \longrightarrow \mathbb{R}$ et on note alors $\|x\|$ la norme d'un vecteur x de $\mathbb{R}^n$.

La norme euclidienne

Soit $d \in \mathbb{N}^*$. On note $N_2 : \mathbb{R}^n \longrightarrow \mathbb{R}$ l'application définie pour tout $d-$uplets de scalaires $x = (x_1, \ldots, x_d)$ par :

$$N_2(x) = \sqrt{\sum_{k=1}^{n} |x_k|^2}$$

L'application N_2 est une norme sur $\mathbb{R}^d$ et est appelée **norme euclidienne**. On notera $N_2(x) = \|x\|_2$

2.2 Produit scalaire

Définition

Un produit scalaire sur $\mathbb{R}^n$ est une application $\langle ., . \rangle : \mathbb{R}^n \times \mathbb{R}^n \longrightarrow \mathbb{R}$ telle que :

1. $\forall (x, y) \in \mathbb{R}^{n2} \quad \langle x, y \rangle = \langle y, x \rangle \quad$ (**symétrie**)

2. $\forall (x, x', y, y') \in \mathbb{R}^{n4} \quad \forall (\lambda, \mu) \in \mathbb{R}$
   $\langle \lambda x + \mu x', y \rangle = \lambda \langle x, y \rangle + \mu \langle x', y \rangle$
   et $\langle x, \lambda y + \mu y' \rangle = \lambda \langle x, y \rangle + \mu \langle x, y' \rangle$ (**bilinéarité**)
3. $\forall x \in \mathbb{R}^n \quad \langle x, x \rangle \geq 0$ (**positivité**)
4. $\forall x \in \mathbb{R}^n \quad \langle x, x \rangle = 0 \Leftrightarrow x = 0_{\mathbb{R}^n}$

Inégalité de Cauchy-Schwarz

Soit $\langle ., . \rangle : \mathbb{R}^n \times \mathbb{R}^n \longrightarrow \mathbb{R}$ un produit scalaire sur $\mathbb{R}^n$. Alors :

$$\forall (x, y) \in \mathbb{R}^{n2} |\langle x, y \rangle| \leq \sqrt{\langle x, x \rangle} \sqrt{\langle y, y \rangle}$$

De plus, pour tout $(x, y) \in \mathbb{R}^{\times 2}$, il y a égalité $|\langle x, y \rangle| = \sqrt{\langle x, x \rangle} \sqrt{\langle y, y \rangle}$ si et seulement si x et y sont linéairement dépendants.

Norme associée au produit scalaire

Soit $\langle ., . \rangle : \mathbb{R}^n \times \mathbb{R}^n \longrightarrow \mathbb{R}$ un produit scalaire sur $\mathbb{R}^n$. Alors l'application $N : \mathbb{R}^n \longrightarrow \mathbb{R}$ définie pour tout $x \in \mathbb{R}^n$ par $N(x) = \sqrt{\langle x, x \rangle}$. Une telle norme est dite euclidienne

2.3 Hyperplan

2.4 Descente de Gradient

Idée générale

On cherche à trouver le minimum d'une fonction $\begin{array}{ccccc} f & : & \mathbb{R}^n & \to & \mathbb{R} \\ & & X & \mapsto & f(X) \end{array}$
De manière générale, il n' existe pas de solution analytique, on utilise donc un algorithme itératif qui permet de trouver un minimum local, la descente de gradient.

La descente de gradient est une méthode d'optimisation qui peut être appliquée à des fonctions de diverses formes, qu'elles soient linéaires, non linéaires, convexes ou non convexes. L'important est que la fonction soit dérivable par rapport aux paramètres que l'on souhaite optimiser. L'algorithme utilise les gradients de la fonction pour guider la recherche vers les valeurs qui minimisent celle-ci.

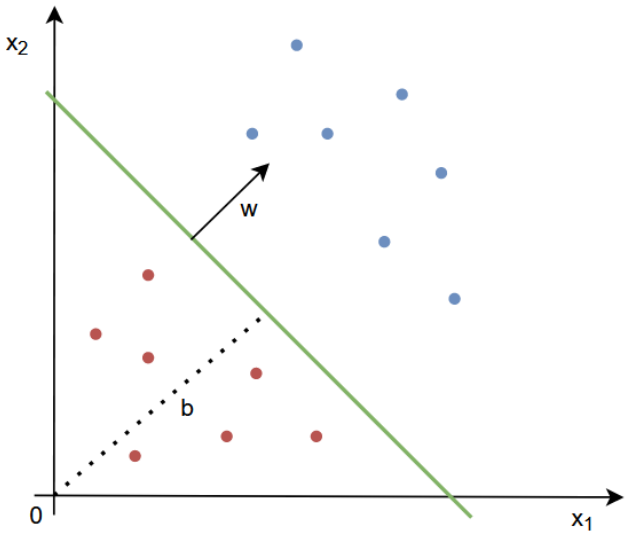


FIGURE 1 – Hyperplan en 2D : $w_1x_1 + w_2x_2 + b = 0$

La descente de gradient consiste à ajuster itérativement les paramètres d'un vecteur w dans la direction qui réduit le plus la fonction f. Pour cela, on calcule le gradient Δf, qui est un vecteur qui indique la direction dans laquelle la fonction augmente le plus rapidement à partir du point actuel. On ajuste donc les valeurs de w pour aller à l'opposé de cette direction. On obtient la forme itérative $\mathrm{w}^{n+1} \leftarrow \mathrm{w}^n - \eta \times \Delta \mathrm{w}^n$ où η est le taux d'apprentissage qui correspond à l'importance des pas effectués. C'est un élément crucial de l'algorithme de descente de gradient, s'il est trop faible l'algorithme mettra trop de temps à converger, si il est trop grand, cela risque d'affecter la correction de celui-ci. Usuellement $\eta \in [0.05, 0.15]$

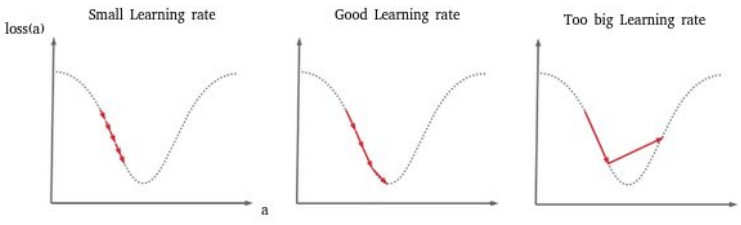


FIGURE 2 – Initialisation

autres méthodes

Il existe d'autres méthodes pour trouver le minimum d'une fonction. Lorsqu'il n'y a pas trop de données, on peut utiliser toutes ces données pour calculer le gradient utile à l'algorithme précédent. Cependant, On utilise souvent un très grand nombre de donnée pour entraîner un réseau de neurone et cela rend ce calcul trop long. C'est pourquoi on utilise plus souvent l'algorithme de descente de gradient stochastique qui utilise le même principe que le premier mais en ne prenant en compte dans son calcul de gradient qu'une partie des données appelé "batch". Cela introduit nécessairement un bruit du fait de ne pas perdre en compte toutes les données mais il converge bien plus rapidement vers une solution. Il existe aussi bien d'autres méthodes, telles que :

- Descente de Gradient avec Moment
- Méthode de Newton
- Nesterov Accelerated Gradient (NAG)
- Optimisation par essaims de particules

etc...

En savoir plus sur la descente de gradient : https ://exo7math.github.io/deepmath-exo7/descente/descente.pdf

3 Perceptron simple

3.1 Introduction

3.1.1 Contexte

On considère une population divisée en 2 groupes G_1, G_2 linéairement séparables. Ces groupes sont distinguables suivant les valeurs de p caractères $X_1, ..., X_p$, sans que l'on ait connaissance des valeurs de $X_1, ..., X_p$ les distinguant.
Soit Y un caractère qualitatif nominal binaire égal au groupe dans lequel un individu appartient.

Pour n individus $\omega_1, ..., \omega_n$ de cette population, on dispose des valeurs de $Y, X_1, ..., X_p$.
Elles sont généralement présentées sous la forme d'un tableau :

	Y	X_1	$\cdots$	X_p
ω_1	y_1	$x_{1,1}$	$\cdots$	$x_{1,p}$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
ω_n	y_n	$x_{n,1}$	$\cdots$	$x_{n,p}$

Ces valeurs constituent les données.

3.1.2 Objectif

On dispose de données linéairement séparables.
À partir de ces données, on veut créer un modèle capable de prédire à quelle classe va appartenir un nouvel individu ω_* en se basant sur cette frontière linéaire (hyperplan) appellée frontière de décision.
Si un individu est au dessus de la droite il appartiendra au groupe $Y = 1$ sinon, il appartiendra au groupe $Y = 0$.
L'objectif est donc de trouver l'équation de cette frontière de décision.

Reformulation mathématique :

Soit l'equation de la frontière de décision d :

$\forall\, x \in \mathbb{R}^p \;;\; b, \mathrm{w}_1, \ldots, \mathrm{w}_n \in \mathbb{R},$

$$d : \mathrm{w}_1 x_1 + \mathrm{w}_2 x_2 + \ldots \mathrm{w}_p x_p + b = 0$$

$$d : \langle x, \mathrm{w} \rangle + b = 0$$

Où $x = (x_{i,1}, \ldots, x_{i,p}) \;\; i = 1 \ldots n$ et $\mathrm{w} = (\mathrm{w}_1, \ldots, \mathrm{w}_n) \;\; b \in \mathbb{R}$

où $\mathrm{w}_1, \mathrm{w}_2, \ldots, \mathrm{w}_p$ et b sont les paramètres à estimer.

formulation mathématique du modèle ici

3.1.3 Approche de Rosenblatt

En 1957, Frank Rosenblatt a inventé le tout premier neurone artificiel capable d'apprentissage, le perceptron.

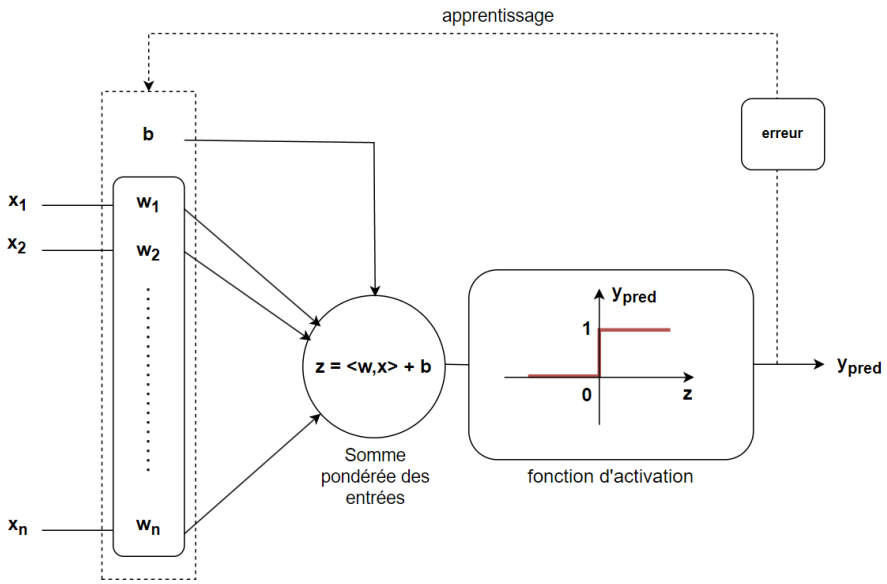


FIGURE 3 – Perceptron de Rosenblatt

Le perceptron s'active quand la somme pondéré de ses entrées dépasse un certain seuil $-b$ modélisé par la fonction d'activation Heaviside. Le perceptron dispose également d'un algorithme d'apprentissage bio-inspiré s'appuyant sur la théorie du neuro-psychologue Donald Hebb.

Théorie de Hebb :

Lorsque deux neurones sont excités conjointement, alors ils renforcent leur lien synaptique.

En s'inspirant de cette théorie, Frank Rosenblatt a imaginé les deux formules itératives suivantes :

$$\mathrm{w}_{k+1} = \mathrm{w}_k + \eta\,(y - y_{pred})\,x_i \tag{1}$$

$$b_{k+1} = b_k + \eta\,(y - y_{pred})\,R^2 \tag{2}$$

Lorsqu 'une entrée active la mauvaise sortie c'est à dire, lorsque $y - ypred \neq 0$ les poids synaptiques sont mis à jours pour que l'entrée s'active avec la sortie attendue. Cette mise à jour est en fait le renforcement synaptique entre l'entrée et la sortie juste. η est la vitesse d'apprentissage. R^2 permet de bouger le biais à l'echelle des données.

Algorithme d'apprentissage du perceptron :

On pose $R = \max_{i=1,\dots,n}(\|x_i\|)$

1. Initialiser aléatoirement les paramètres w et b de l'hyperplan
2. Faire passer les observations unes à unes
   - Calculer l'erreur de prédiction de l'observation $(y - y_{pred})$
   - Mettre à jour les paramètres w et b de l'hyperplan

$$\mathrm{w}_{k+1} = \mathrm{w}_k + \eta\,(y - y_{pred})\,x_i \tag{3}$$

$$b_{k+1} = b_k + \eta\,(y - y_{pred})\,R^2 \tag{4}$$

3. Jusqu'à convergence du processus

Grâce au terme d'erreur de prédiction $(y - y_{pred})$, les formules itératives (3) et (4) ne changent les paramètres que si l'observation est du mauvais côté de la frontière de décision, c'est à dire si elle

est mal classée.

Si une observation est mal classée, $y - y_{pred} = 1$ ou $y - y_{pred} = -1$, les paramètres sont modifiés pour que la droite se "décale" dans le but que l'observation se retrouve du bon côté. Cela traduit le renforcement synaptique entre l'entrée et la sortie attendue.

Par exemple, si notre modèle, à une itération k donne en sortie $y_{pred} = 1$ pour une observation x_i alors que $y = 0$ était attendu, l'algorithme va renforcer le lien synaptique entre l'entrée x_i et la bonne sortie $y = 0$, en réduisant le poids synaptique de cette entrée.

$$w_{k+1} = w_k - \eta\, x_i$$

$$b_{k+1} = b_k - \eta\, R^2$$

Ce qui donne graphiquement (x_i le point entouré) :

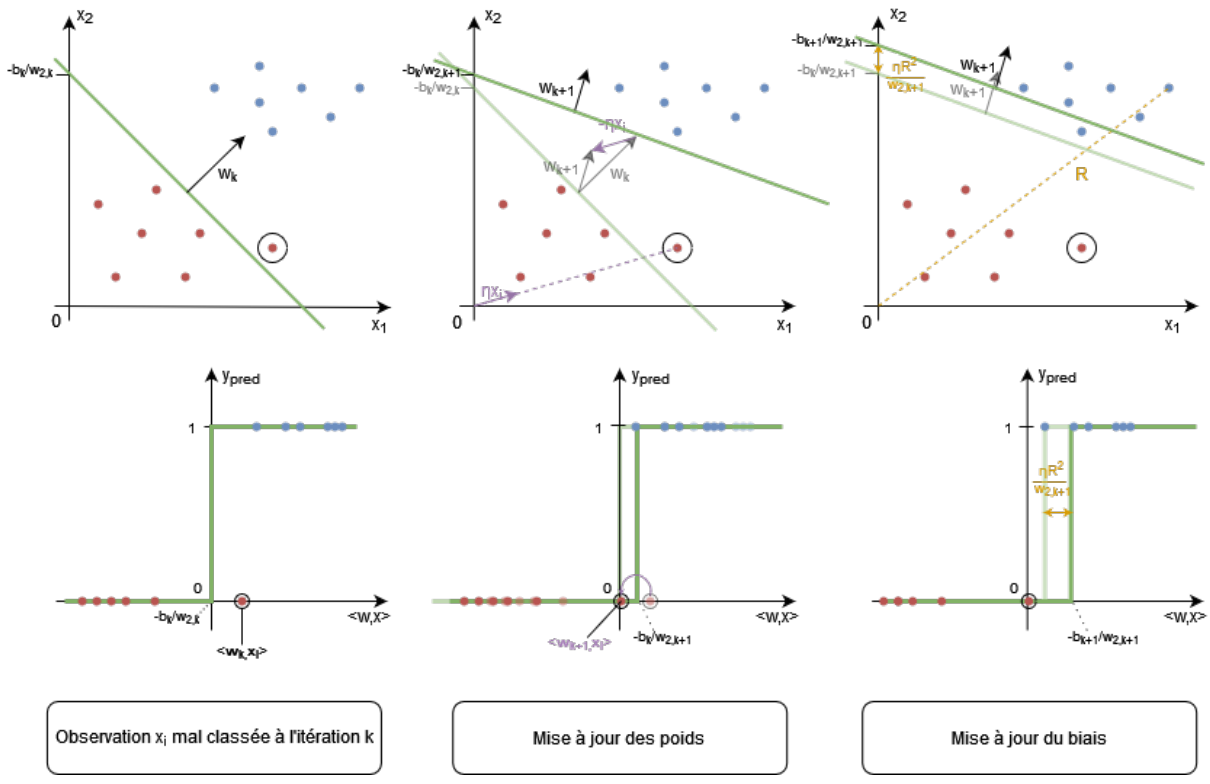


FIGURE 4 – Observation x_i à l'itération k de l'algorithme d'apprentissage du perceptron

3.2 Exemple 1

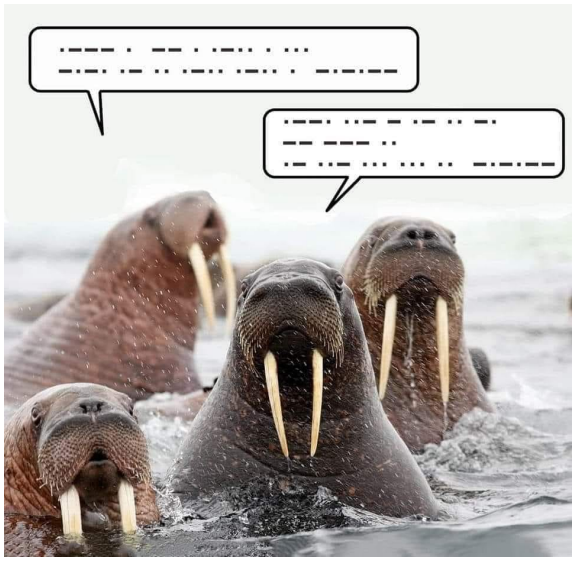


FIGURE 5 – Étape 1

On dispose du poids x_1 (en kg) et de la taille x_2 (en m) d'une population de 10 morses du pacifique dont 5 males (Y = 0) et 5 femelles (Y = 1).

Données

	x_1	x_2	Y
ω_1	1757.6	2.72	0
ω_2	514.9	2.86	1
ω_3	849.2	2.7	1
ω_4	972.3	3.38	0
ω_5	718.8	2.19	1
ω_6	1725.0	3.5	0
ω_7	1284.1	3.4	0
ω_8	640.9	2.38	1
ω_9	1044.4	3.03	0
ω_{10}	781.9	2.81	1

Données centrées réduites

	x_1	x_2	Y
ω_1	1.773	-0.428	0
ω_2	-1.251	-0.089	1
ω_3	-0.437	-0.476	1
ω_4	-0.138	1.167	0
ω_5	-0.755	-1.708	1
ω_6	1.694	1.457	0
ω_7	0.621	1.215	0
ω_8	-0.944	-1.249	1
ω_9	0.038	0.321	0
ω_{10}	-0.601	-0.21	1

On souhaite entraîner un perceptron pour pouvoir, à partir d'un nouveau poids et d'une nouvelle

taille de morse, déterminer son sexe.

Cela revient a trouver trouver une droite qui sépare au mieux nos deux classes.

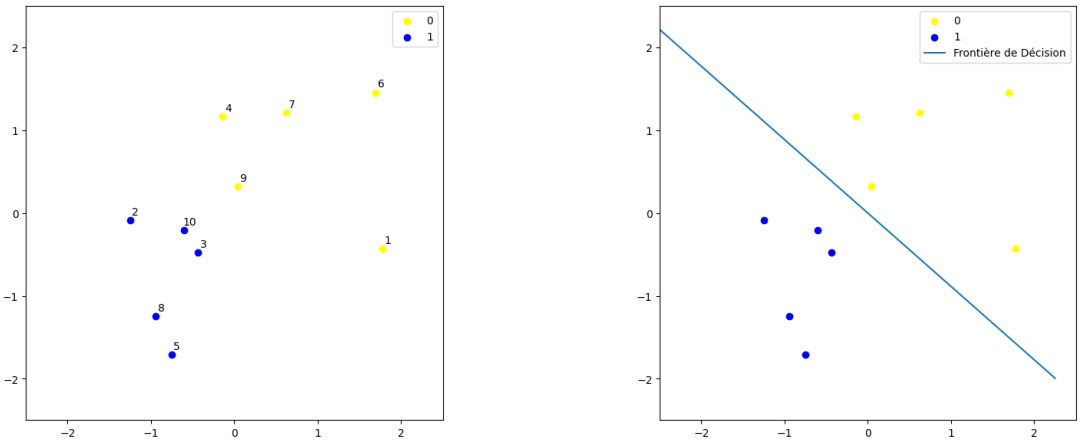


FIGURE 6 – Objectif

Si l'individu est au dessus de la droite c'est un male, sinon, c'est une femelle.

le classifieur sera alors la fonction suivante :

$$f(x) = \begin{cases} 1 & \text{si} \quad \langle x, \mathrm{w} \rangle + b \geq 0 \\ 0 & \text{sinon} \end{cases}$$

Principe :

Algorithme itératif

Pour chaque exemple mal classé, on corrige l'hyperplan pour que l'exemple se retrouve du côté correct de celui-ci.

À chaque itération, et pour chaque individu, on modifie les valeurs de w et de b afin d'améliorer l'hyperplan. On répète cela jusqu'à ce qu'il n'y ai plus d'erreur, ou on choisi un nombre d'itération arbitraire maximum.

Il y a 3 valeurs à initialiser pour cet algorithme, le poids w, le biais b et le taux d'apprentissage t. Le poids va permettre de déterminer le coefficient directeur de l'hyperplan, le biais son ordonnée à l'origine et enfin le taux d'apprentissage permet de déterminer l'importance des ajustements lors de la mise à jour de l'hyperplan. Si sa valeur est trop petite, l'algorithme mettra longtemps à converger. Si sa valeur est trop grande, l'algorithme ne convergera pas et oscillera entre plusieurs solutions.

Pour notre exemple, nous allons choisir : $\mathrm{w}^0 = \begin{pmatrix} \mathrm{w}_1^0 \\ \mathrm{w}_2^0 \end{pmatrix} = \begin{pmatrix} 0 \\ 0.2 \end{pmatrix}$, $b^0 = 1$, $t = 0.1$

En outre, nous allons séparer nos données en données d'entrainement et donnée de test. Les première permettant de faire la phase apprentissage du perceptron et les secondes permettant de vérifier la validité du modèle. ainsi les données d'entraînement seront $\{\omega_1, .., \omega_8\}$ et les données tests seront $\{\omega_9, \omega_{10}\}$

Pour $\mathbb{R}^2$, on a une équation de droite :

$$y = -\frac{\mathrm{w}_1}{\mathrm{w}_2} \times x \; - \; \frac{b}{\mathrm{w}_1}$$

Ici, l'équation de droite est $y = -5$.

On obtient ainsi le graphique suivant :

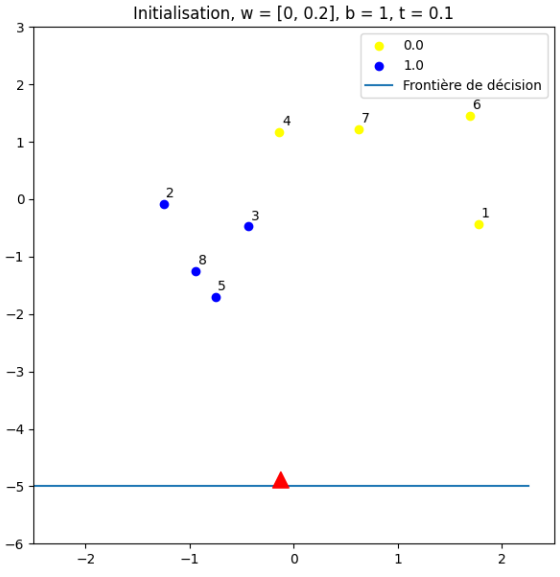


FIGURE 7 – Initialisation

Procédons étape par étape :

Étape 1

Nous sommes à la première étape, on cherche à savoir si ω_1 est bien placé par rapport à la droite. On a $\langle \omega_1, \mathrm{w}^0 \rangle + b^0 = 1.773 \times 0 + (-0.428 \times 0.2) + 1 = 0.914 > 0$ Or ce qui donne $y_{pred}(\omega_1 = 1\omega_1$ appartient à la Classe 0, on devrait donc avoir un nombre négatif. Ainsi, il faut modifier la droite. On obtient pour w^1 :

$$\mathrm{w}^1 \leftarrow \mathrm{w}^0 + t \times (y_1 - y_{pred_1}) \times x_1$$

$$\mathrm{w}^1 \leftarrow \begin{pmatrix} 0 \\ 0.2 \end{pmatrix} + 0.1 \times (0 - 1) \times \begin{pmatrix} 1.773 \\ -0.428 \end{pmatrix} = \begin{pmatrix} -0.177 \\ 0.242 \end{pmatrix}$$

Et pour b^1 :

$$b^1 \leftarrow b^0 + t \times (y_1 - y_{pred_1}) \times x_1$$

$$b^1 \leftarrow 1 + 0.1 \times (0 - 1) \times R^2$$

Avec $R^2 = max_{1 \leq i \leq n} ||x_i|| = 4.99$

$$b^1 = 0.516$$

Ce qui donne pour nouvelle équation de droite : $y = 0.73x - 2.12$

Ce qui donne le graphe suivant :

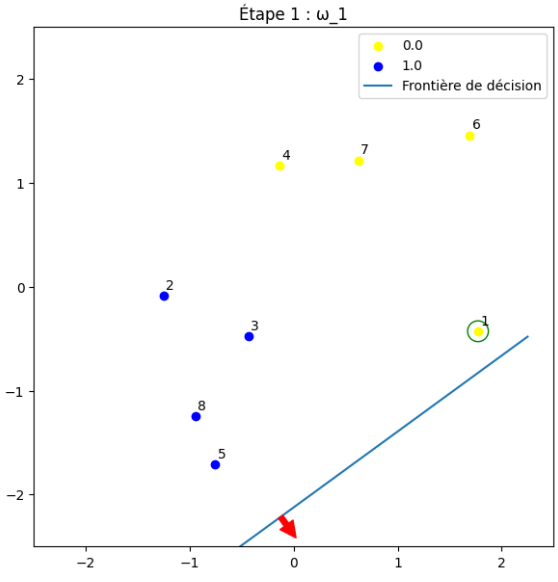


FIGURE 8 – Étape 1

Étape 2

On vérifie si ω_2 est bien placé :

On a $\langle \omega_2, \mathrm{w}^1 \rangle + b^1 = -1.251 \times -0.177 + (-0.089 \times 0.242) + 0.031 = 0.715 > 0$, on a donc $y_{pred_2} = 1$ or ω_2 appartient à la classe 1, $y_{pred_2} = y_2$ la droite ne doit donc pas être modifiée. $\mathrm{w}^2 = \mathrm{w}^1, b^2 = b^1$

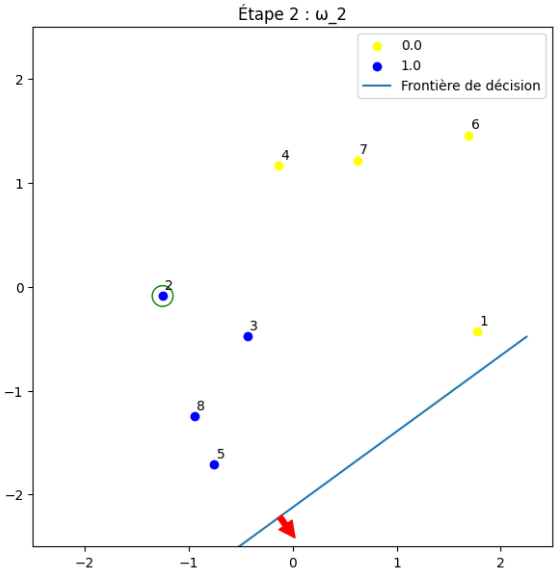


FIGURE 9 – Étape 2

Étape 3

On vérifie si ω_3 est bien placé :

On a $\langle \omega_3, \mathrm{w}^2 \rangle + b^2 = -0.437 \times -0.177 + (-0.476 \times 0.242) + 0.031 = 0.478 > 0$, $y_{pred_3} = 1$ or ω_3 appartient à la classe 1, la droite ne doit donc toujours pas être modifiée. $\mathrm{w}^3 = \mathrm{w}^2, b^3 = b^2$

Étape 4

$\langle \omega_4, \mathrm{w}^3 \rangle + b^3 = -0.138 \times -0.177 + (1.167 \times 0.242) + 0.031 = 0.477 > 0 \implies y_{pred_4} = 1$, or ω_3 appartient à la classe 0, il faut donc modifier la droite :

$$\mathrm{w}^4 \leftarrow \begin{pmatrix} -0.177 \\ 0.242 \end{pmatrix} + 0.1 \times (0-1) \times \begin{pmatrix} -0.138 \\ 1.167 \end{pmatrix} = \begin{pmatrix} -0.164 \\ 0.126 \end{pmatrix}$$

$$b^4 \leftarrow 0.516 + 0.1 \times (0-1) \times R^2 = 0.0311$$

On obtient ainsi la nouvelle équation de droite : $y = 1.297 \times x - 0.247$

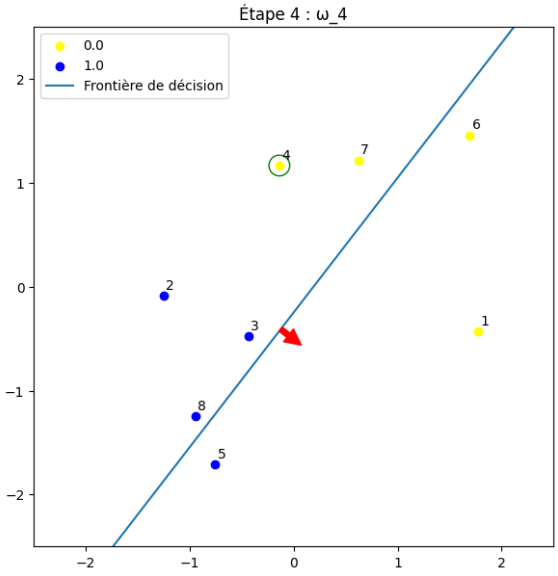


FIGURE 10 – Étape 4

Étape 5

$\langle \omega_5, \mathrm{w}^4 \rangle + b^4 = -0.061 \times -0.164 + (-1.708 \times 0.126) + 0.031 = 0.823 < 0 \implies y_{pred_5} = 0$, or ω_4 appartient à la classe 1, $y_{pred_5} \neq y_5$ on doit donc ajuster la droite :

$$\mathrm{w}^5 \leftarrow \begin{pmatrix} -0.164 \\ 0.126 \end{pmatrix} + 0.1 \times (1-0) \times \begin{pmatrix} -0.755 \\ -1.708 \end{pmatrix} = \begin{pmatrix} -0.239 \\ -0.0447 \end{pmatrix}$$

$$b^5 \leftarrow 0.0311 + 0.1 \times (1-0) \times R^2 = 0.516$$

On obtient ainsi la nouvelle équation de droite : $y = -5.347 \times x + 11.53$

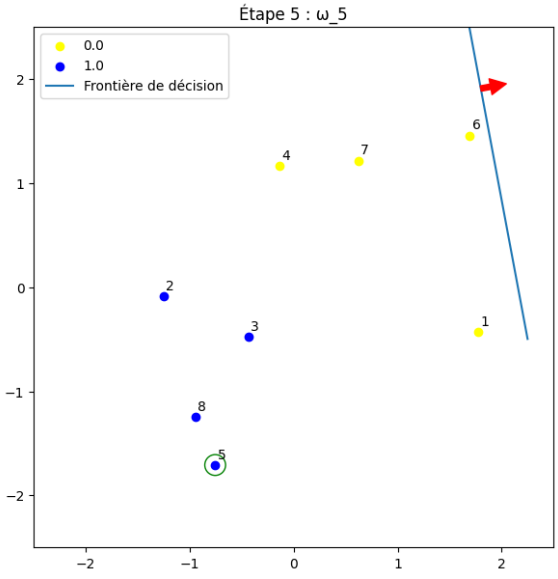


FIGURE 11 – Étape 5

Étape 6

$\langle \omega_5, \mathrm{w}^5 \rangle + b^5 = 1.694 \times -0.239 + (1.457 \times -0.0447) + 0.515 = 0.0455 > 0, y_{pred_6} = 1$, or ω_5 appartient à la classe 0, $y_{pred_6} \neq y_6$ on ajuste les paramètres du perceptron :

$$\mathrm{w}^6 \leftarrow \begin{pmatrix} -0.239 \\ -0.0447 \end{pmatrix} + 0.1 \times -1 \times \begin{pmatrix} 1.694 \\ 1.457 \end{pmatrix} = \begin{pmatrix} -0.408 \\ -0.190 \end{pmatrix}$$

$$b^6 \leftarrow 0.515 + 0.1 \times -1 \times R^2 = 0.0311$$

On obtient ainsi la nouvelle équation de droite : $y = -2.144 \times x + 0.163$

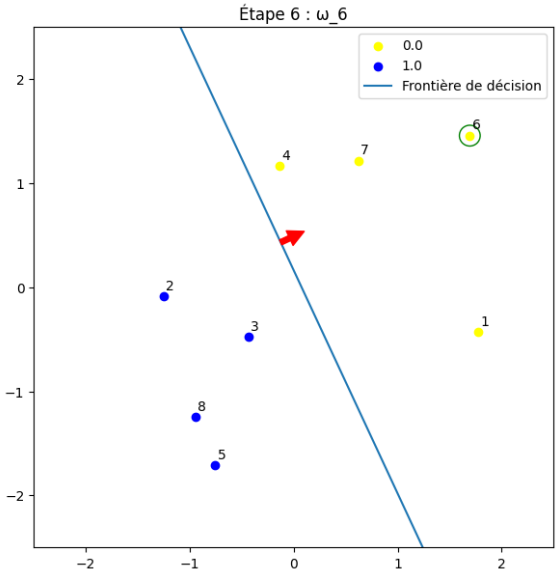


FIGURE 12 – Étape 6

La frontière de décision sépare à présent bien toutes les données d'entraînement, vérifions maintenant si elle sépare bien les données test. Vérifions le point 9 : $\langle \omega_5, \mathrm{w}^5 \rangle + b^5 = 0.038 \times -0.4084 + (0.321 \times -0.1904) + 0.0311 = -0.0317 < 0$. On obtient donc un label $y_{pred_9} = 0 = y_9$. Le perceptron donne le bon label pour la donnée test ω_9 De même, le point 10 est bien placé : $\langle \omega_5, \mathrm{w}^5 \rangle + b^5 = -0.601 \times -0.4084 + (-0.21 \times -0.1904) + 0.0311 = 0.317 > 0$. On obtient donc un label $y_{pred_{10}} = 1 = y_{10}$.

On peut les représenter graphiquement pour vérifier ces résultats : Le perceptron a en effet bien

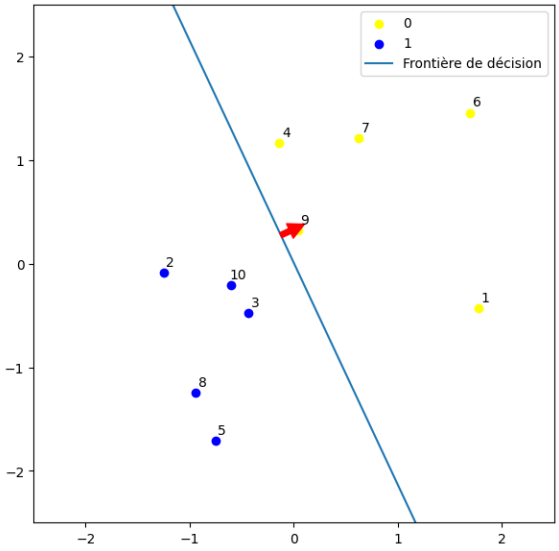


FIGURE 13 – Étape 6

classé les données, on a un taux d'erreur avec cette initialisation de 0 %.

Cependant, ici la frontière de décision obtenue après convergence de l'algorithme dépend des poids initiaux. Usuellement, ceux-ci sont initialisés de manière aléatoire, ce qui donnerait donc une frontière de décision après convergence différente à chaque fois que l'on relance le programme. Par exemple, si on initialise différemment, on obtient cette convergence :

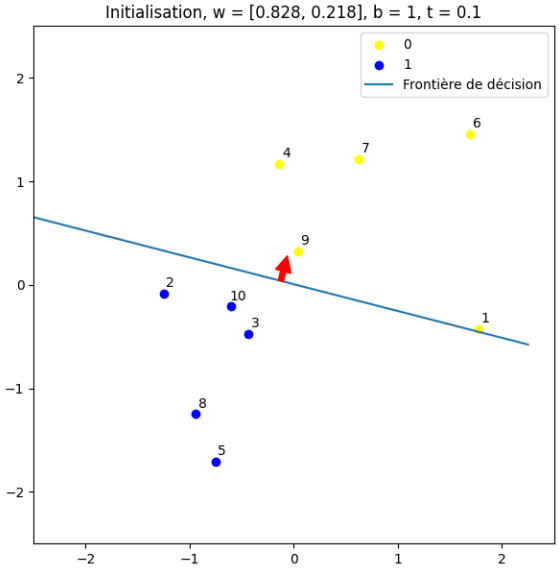


FIGURE 14 – Initialisation aléatoire

insérer un lien vers notebook ici ou après la conclu

3.3 Conclusion Perceptron historique

+ Converge toujours si données linéairement séparables et eta pas trop grand

+ simple

- solution non optimale, dépend de l'ordre de parcours des données (pas assez flexible Heaviside)

- sensible aux données aberrante

Fonctionne pour des données linéairement séparable comme prévu

Converge très rapidement (faible complexité algorithmique)

limite :

- Peu sûr de la qualité de la solution trouvé

(l'algorithme s'arrete une fois que tout les points sont classés correctement, on obtient donc une solution mais pas la solution optimale, ce qui est mauvais pour classer de nouvelles données) - Si une erreur / donnée aberrante -> converge pas / mauvaise classif

3.4 Le perceptron avec une autre fonction d'activation

3.4.1 Une autre manière d'optimiser

Nous avons utilisé la fonction d'activation Marche ou step/Heaviside en anglais, cependant ce n'est pas la plus utilisée d'une part, parce qu'elle n'est pas dérivable et d'autre part parce qu'elle est trop rigide, ne montrant pas si l'erreur commise est grande ou petite. C'est pour cela que l'on va utiliser la sigmoïde, qui est définit par $a(z) = \frac{1}{1+e^{-z}}$ qui donne une valeur entre 0 et 1.

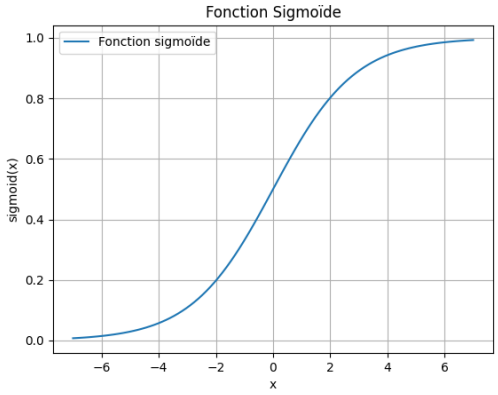


FIGURE 15 – Sigmoïde

Ainsi, intuitivement, si on a des individus avec les classes ici en rouge et les valeurs données par le perceptron en bleu, on a une valeur renvoyée par le perceptron fausse si la classe de l'individu est 1 mais que le perceptron donne une valeur plus proche de 0 et inversement.

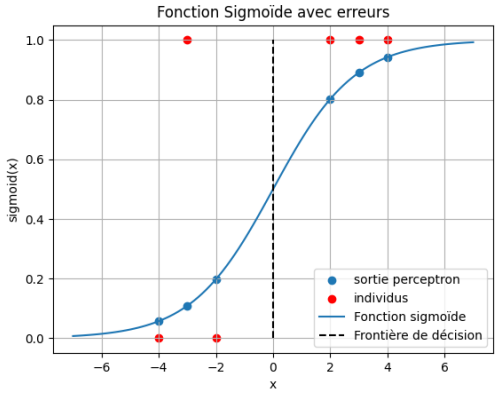


FIGURE 16 – Sigmoïde avec individus

Comme par exemple sur ce graphique l'individu en 2^e position.

Cependant, on a maintenant une manière de quantifier l'importance de l'erreur même lorsque le perceptron donne une valeur proche de la classe. La valeur transmise à la fonction d'activation est pour un individu ω_i, $z_i = \sum_{j=1}^{p} \mathrm{w}_j \times x_{i,j}$.

Cette valeur est dans l'intervalle $[0, 1]$

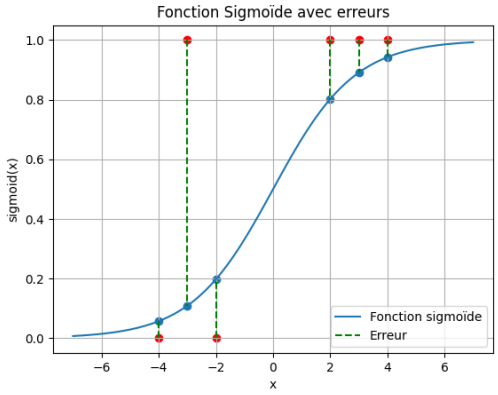


FIGURE 17 – Sigmoïde avec erreurs

On peut considérer que le valeur renvoyée par la sigmoïde correspond à la probabilité qu'a un individu d'appartenir à une classe d'après le perceptron. $P(\hat{Y}_i = y_i) = a(z_i)^{y_i} \times (1 - a(z_i))^{1-y_i}$. Ce qui correspond à $P(\hat{Y}_i = 0) = a(z_i)$ et $P(\hat{Y}_i = 1) = 1 - a(z_i)$

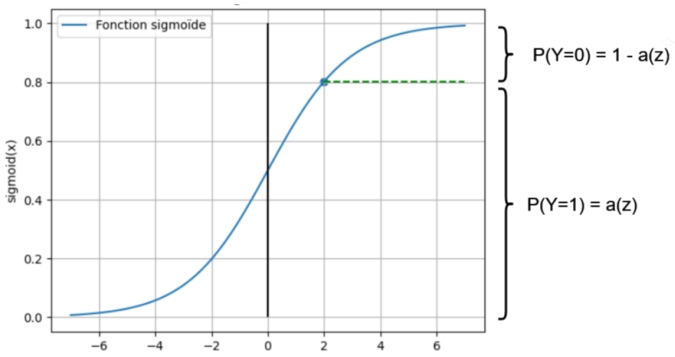


FIGURE 18 – Sigmoïde Bernoulli

On reconnaît ici une probabilité de Bernoulli, avec $p = a(z_i)$. On va donc chercher à minimiser l'erreur du modèle en modifiant les paramètres de p que sont les poids w et le biais b ($z_i = \langle x_i, w \rangle + b$). Pour rendre compte de la fiabilité du modèle, on calcule la logvraisemblance entre y et $\hat{y}$. Ce qui correspond pour une variable suivant une loi de Bernoulli à $LL(y, \hat{y}) = y \times ln(\hat{y}) + (1 - y) \times ln(1 - \hat{y})$

On peut considérer l'erreur pour une observation à partir de la fonction perte correspondant à

l'opposé de cette vraisemblance : $\mathcal{L}(y, \hat{y}) = -[y \times ln(\hat{y}) + (1-y) \times ln(1-\hat{y})]$. Qui dans notre cas se traduit pour un individu par $\mathcal{L}(y_i, a(z_i)) = -[y_i \times ln(a(z_i)) + (1-y_i) \times log(1-a(z_i))]$. On a donc notre fonction coût rendant compte de l'erreur moyenne sur tous les individus :

$$\mathcal{L} = -\frac{1}{n}\sum_{i=1}^{n} y_i \times ln(a(z_i)) + (1-y_i) \times ln(1-a(z_i))$$

Maintenant que l'on a un objet permettant de rendre compte de la qualité de notre perceptron, on souhaiterais l'améliorer ce qui revient à minimiser $\mathcal{L}$. On utilise l'algorithme de descente de gradient pour trouver ce minimum en modifiant les valeurs des poids w_i. Comme vu précédemment, on trouve le minimum de manière itérative :

$\mathrm{w}_i \leftarrow \mathrm{w}_i - \eta \times \nabla\mathcal{L}(\mathrm{w}_i)$

On souhaite donc appliquer l'algorithme de descente du gradient à notre log vraisemblance : On cherche donc le gradient de $\mathcal{L}$

$$\nabla\mathcal{L} = \begin{pmatrix} \frac{\partial\mathcal{L}}{\partial b} \\ \frac{\partial\mathcal{L}}{\partial \mathrm{w}_1} \\ \frac{\partial\mathcal{L}}{\partial \mathrm{w}_2} \\ \dots \\ \frac{\partial\mathcal{L}}{\partial \mathrm{w}_m} \end{pmatrix}$$

Il nous reste maintenant à calculer ces valeurs, on utilise la règle de la chaine :

$$\frac{\partial\mathcal{L}}{\partial \mathrm{w}_j} = \frac{\partial\mathcal{L}}{\partial a(z_i)} \times \frac{\partial a(z_i)}{\partial z_i} \times \frac{\partial z_i}{\partial \mathrm{w}_j}$$

$$\frac{\partial\mathcal{L}}{\partial a(z_i)} = \frac{\partial}{\partial a(z_i)}\left(-\frac{1}{n}\sum_{i=1}^{n} y_i \times ln(a(z_i)) + (1-y_i) \times ln(1-a(z_i))\right) = -\frac{1}{n}\sum_{i=1}^{n} \frac{y_i}{a(z_i)} - \frac{1-y_i}{1-a(z_i)}$$

$$\frac{\partial a(z_i)}{\partial z_i} = \frac{\partial}{\partial z_i}\frac{1}{1+e^{-z_i}} = \frac{e^{-z_i}}{(1+e^{-z_i})^2}$$

$$\frac{\partial z_i}{\partial \mathrm{w}_j} = \frac{\partial}{\partial \mathrm{w}_j}(\sum_{k=1}^{p} \mathrm{w}_k \times x_{i,k}) = x_{i,j}$$

et ainsi, en remarquant que

$$\frac{e^{-z_i}}{(1+e^{-z_i})^2} = \frac{1}{1+e^{-z_i}} \times \frac{e^{-z_i}}{(1+e^{-z_i})} = a(z_i)(1-a(z_i))$$

On obtient

$$\frac{\partial \mathcal{L}}{\partial \mathrm{w}_j} = -\frac{1}{n}\sum_{i=1}^{n}(\frac{y_i}{a(z_i)} - \frac{1-y_i}{1-a(z_i)}) \times a(z_i)(1-a(z_i)) \times x_{i,j}$$

$$= -\frac{1}{n}\sum_{i=1}^{n}\big(y_i \times (1-a(z_i)) - (1-y_i)\big) \times a(z_i) \times x_{i,j}$$

Ce qui donne :

$$\frac{\partial \mathcal{L}}{\partial \mathrm{w}_j} = \frac{1}{n}\sum_{i=1}^{n}(a(z_i)-y_i) \times x_{i,j}$$

De la même manière, pour le biais, on obtient

$$\frac{\partial \mathcal{L}}{\partial b} = \frac{1}{n}\sum_{i=1}^{n}(a(z_i)-y_i)$$

.

On obtient ainsi la forme itérative de l'algorithme de descente du gradient sous forme vectorielle :

$$\mathrm{w} \leftarrow \mathrm{w} - \eta \times \nabla \mathcal{L}$$

Soit :

$$\forall j : \mathrm{w}_j \leftarrow \mathrm{w}_j - \eta \times \frac{1}{n}\sum_{i=1}^{n}(a(z_i)-y_i)x_{i,j}$$

$$b \leftarrow b - \eta \times \frac{1}{n}\sum_{i=1}^{n}(a(z_i)-y_i)$$

Une fois que l'on considère avoir converger à l'aide de cet algorithme de descente de gradient et obtenu un perceptron optimisée pour sa tâche de classification, on obtient pour un nouvel individu $y_{pred}(x_{n+1}) = \begin{cases} 1 & \text{si} \qquad \langle x_{n+1}, w\rangle + b > 0.5 \\ 0 & \text{sinon} \end{cases}$

Regardons cela sur l'exemple vu en 2.5

3.4.2 Exemple des morses avec une fonction d'activation sigmoïde

Nous allons utiliser pour cet approche le même exemple, les données sont celles de la taille (x_1) et le poids (x_2) de morses et on souhaite déterminer entraîner le modèle pour pouvoir déterminer le sexe d'un nouvel individu.

Tout d'abord, on effectue la partie apprentissage du perceptron à l'aide de l'algorithme itératif basé sur la descente de gradient vu précédemment :

	x_1	x_2	Y
ω_1	1.773	-0.428	0
ω_2	-1.251	-0.089	1
ω_3	-0.437	-0.476	1
ω_4	-0.138	1.167	0
ω_5	-0.755	-1.708	1
ω_6	1.694	1.457	0
ω_7	0.621	1.215	0
ω_8	-0.944	-1.249	1
ω_9	0.038	0.321	0
ω_{10}	-0.601	-0.21	1

Algorithm 1 Algorithme du perceptron simple avec sigmoïde

Require: x : n données d'apprentissage, y : les classes, η : le taux d'apprentissage, N_{max} : nombre d'itérations.

Ensure: w_k, b_k : *les poids et le biais.*

Initialisation
$\mathrm{w}_0 = (0, 0.2)$ *poids*
$b_0 = 1$ *biais*
$k = 0$ *itérations*

while $k < N_{max}$ **do**

$a \leftarrow sigmoid(\langle x, \mathrm{w} \rangle + b)$
$\nabla \mathrm{w} \leftarrow \frac{1}{n} \sum_{i=1}^{n} (a_i - y_i) * x_i$
$\nabla b \leftarrow \frac{1}{n} \sum_{i=1}^{n} (a_i - y_i)$
$\mathrm{w}_{k+1} \leftarrow \mathrm{w}_k + \eta * \nabla \mathrm{w}$
$b_{k+1} \leftarrow b_k + \eta * \nabla b$
$k \leftarrow k + 1$

end while

return (w_k, b_k) *paramètres de l'hyperplan*

Regardons ce qu'il se passe sur les premiers points. On représente ces données sur (x_1, x_2) :

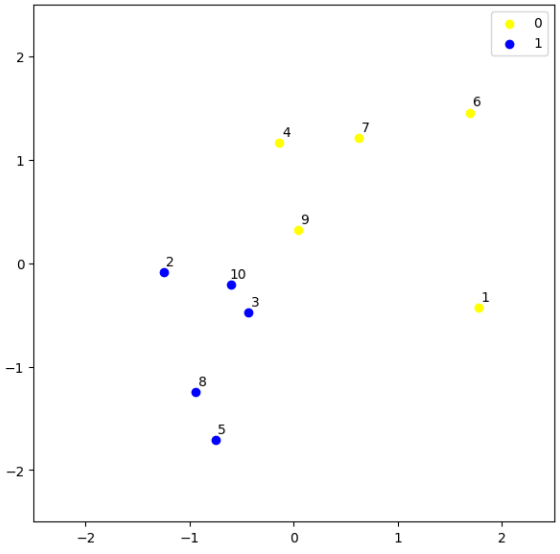


FIGURE 19 – Représentation données Morse

Une époque (ou epoch) correspond à une étape durant laquelle les données d'entraînement vont "passer" en entrée dans le réseau. Regardons ce que fais l'algorithme sur quelques epoch (ou itéra-

tions) :

On initialise les poids et le biais aux valeurs : $w^0 = \begin{pmatrix} w_1^0 \\ w_2^0 \end{pmatrix} = \begin{pmatrix} 0.13 \\ 0.36 \end{pmatrix}$, $b^0 = 0$, $t = 0.5$

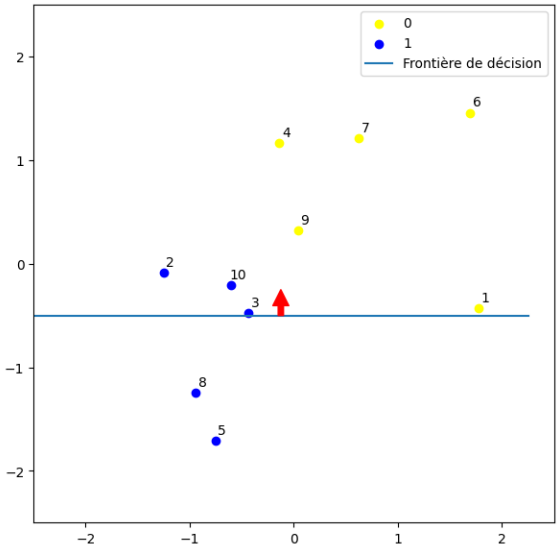


FIGURE 20 – Donnée avec perceptron initialisé

Pour ω_1, nous allons calculer le gradient : On commence par calculer les valeurs du perceptron initialisé ainsi pour les individus :

$a(z_1) = a(w_1^0 * x_{1,1} + w_2^0/x1, 2) = a(0.13 * 1.773 + 0.36 * -0.428) = a(0.076) = \frac{1}{1+e^{-0.076}} = 0.519$

$a(z_2) = a(w_1^0 * x_{2,1} + w_2^0/x2, 2) = a(0.13 * -1.251 + 0.36 * -0.089) = a(-0.178) = \frac{1}{1+e^{-(-0.178)}} = 0.456$

...

On peut maintenant calculer la différence avec la valeur de y pour ces points :

$a(z_1) - y_1 = 0.4786 - 0 = 0.519$

$a(z_2) - y_2 = 0.4956 - 1 = -0.544$

...

On peut à présent calculer le gradient de notre modèle :

$\nabla \mathrm{w}_1^0 = \frac{1}{n} \sum_{i=1}^{n} (a(z_i) - y_i) * x_{i,1}$

$= \frac{1}{n} * (0.519 \times 1.773 + (-0.544) \times -1.251 + ...) = 0.476$

et

$\nabla \mathrm{w}_2^0 = \frac{1}{n} \sum_{i=1}^{n} (a(z_i) - y_i) \times x_{i,2}$

$= \frac{1}{n} * (0.519 \times -0.428 + (-0.544) \times -0.089 + ...) = 0.448$

et enfin

$\nabla b^0 = \frac{1}{n} \sum_{i=1}^{n} (a(z_i) - y_i)$

$= \frac{1}{n} * (0.519 + (-0.544) + ...) = 2.3.10^{-5}$

On obtient donc pour nos nouvelles valeurs de poids :

$$\begin{pmatrix} b^1 \\ \mathrm{w}_1^1 \\ \mathrm{w}_2^1 \end{pmatrix} = \begin{pmatrix} b^0 \\ \mathrm{w}_1^0 \\ \mathrm{w}_2^0 \end{pmatrix} - \eta \times \begin{pmatrix} \nabla b^0 \\ \nabla \mathrm{w}_1^0 \\ \nabla \mathrm{w}_2^0 \end{pmatrix} = \begin{pmatrix} 0 \\ 0.13 \\ 0.38 \end{pmatrix} - 0.5 \times \begin{pmatrix} 2.3.10^{-5} \\ 0.476 \\ 0.448 \end{pmatrix}$$

$$\begin{pmatrix} b^1 \\ \mathrm{w}_1^1 \\ \mathrm{w}_2^1 \end{pmatrix} = \begin{pmatrix} 0 \\ -0.108 \\ 0.136 \end{pmatrix}$$

On obtient ainsi la nouvelle frontière de décision de la même manière que précédemment, $x_2 = -\frac{\mathrm{w}_1^1}{\mathrm{w}_2^1} \times x_1 - b$ soit $x_2 = 0.794 \times x_1$ ce qui donne le graphique suivant :

On peut voir l'évolution de la courbe au fil des epoch :

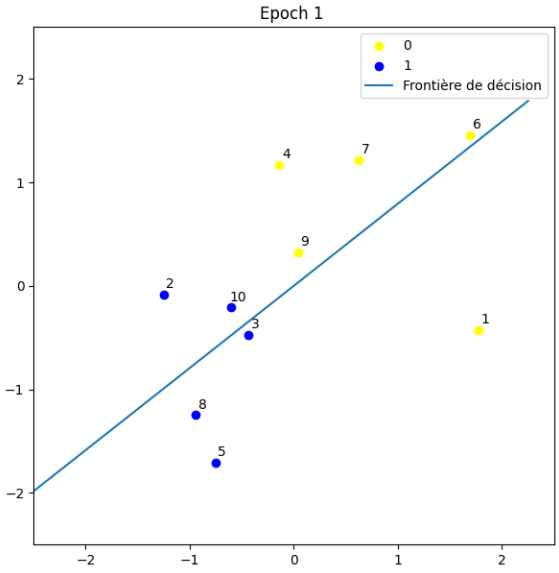
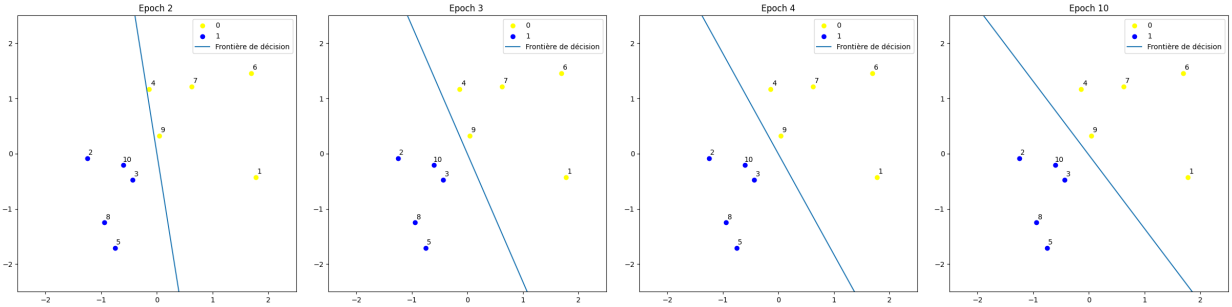


FIGURE 21 – Algo sigmoïde après 1 epoch



Epoch 2 Epoch 3 Epoch 4 Epoch 10

FIGURE 22 – Évolution de la frontière de décision

Qui converge vers la solution optimale :

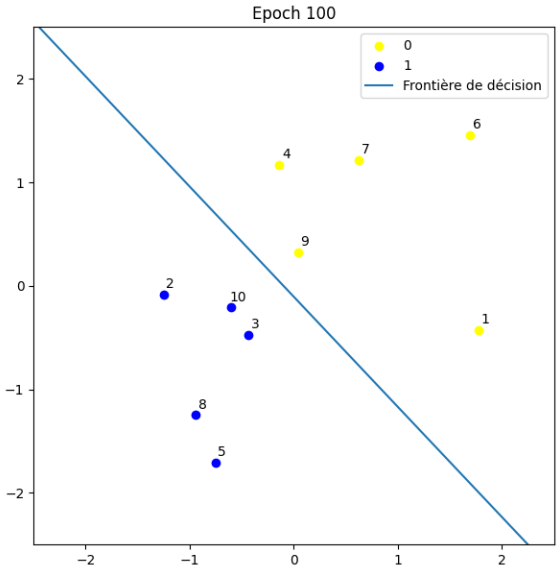
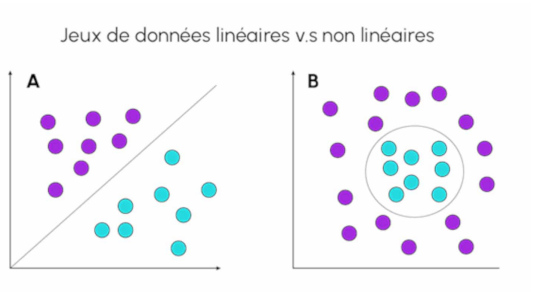


FIGURE 23 – Algo sigmoïde après 100 epoch

insérer un lien vers les notebook

3.4.3 Conclusion

On a réussi à créer un modèle qui permet de séparer linéairement des données de manière efficace et fiable. Cependant, cet outil n'est utilisable que sur des données séparable linéairement, mais qu'en est-il des données qui ne le sont pas.



Pour ces données, un perceptron seul ne sera pas capable de trouver une frontière de décision convenable, c'est pourquoi on utilise plusieurs perceptrons connectés les uns aux autres, appelé perceptron multicouche, un réseau de neurones.