Le Perceptron

Introduction

Le perceptron, inventée en **1957** par **Frank Rosenblatt**, est un neurone formel associé à une méthode d'apprentissage supervisé. Le neurone formel, parfois nommé neurone de **McCulloch-Pitts**, développé par ces derniers en **1943**, est une représentation mathématique et informatique d'un neurone biologique.

Le perceptron est qualifiée de méthode d'apprentissage supervisé, en raison de l'utilisation d'une base de données annotée (X,Y), c'est-à-dire que chaque exemple de cette dernière est clairement identifié par l'intermédiaire d'une classe (numérotation, dénomination). Contrairement à l'apprentissage non supervisé où seules les données font partie de la base.

Modèle Mathématiques du perceptron

Dans le contexte du perceptron, notre base de données est constitué de n exemples, dont chaque exemple possède p caractéristiques. Ainsi, nous pouvons écrire (X,Y) $\in \mathbb{M}_{n,p}(\mathbb{R}) \times \mathbb{M}_{n,1}(\mathbb{R})$, avec n, p $\in \mathbb{N}^*$. Les lignes de X représentent les exemples, et les colonnes représentent les caractéristiques :

$$X = \begin{pmatrix} x_1^{[1]} & \cdots\cdots & x_1^{[j]} & \cdots\cdots & x_1^{[p]} \\ \vdots & \ddots & \vdots & \cdots & \vdots \\ x_i^{[1]} & \cdots\cdots & x_i^{[j]} & \cdots\cdots & x_i^{[p]} \\ \vdots & \cdots & \vdots & \ddots & \vdots \\ x_n^{[1]} & \cdots\cdots & x_n^{[j]} & \cdots\cdots & x_n^{[p]} \end{pmatrix}, Y = \begin{pmatrix} y_1 \\ \vdots \\ y_i \\ \vdots \\ y_n \end{pmatrix}$$

Chaque donnée en entrée est associée à un **poids** (noté w_j, $\forall$ j $\in$ [|1 ;p|]). Il s'agit d'une somme pondérée des x_j auquel nous ajoutons un **biais** noté b. Maintenant, nous pouvons voir cela comme un **modèle linéaire**.

$\forall$ x $\in$ X (x $\in \mathbb{M}_{1,p}$), vecteur ligne de X, un exemple de la base de données.

$$Z_x : \left\{ \begin{array}{rcl} \mathbb{M}_{p,1}(\mathbb{R}) \times \mathbb{R} & \longrightarrow & \mathbb{R} \\ (W, b) & \longmapsto & x \cdot W + b \end{array} \right\}$$

$$Z_x(W, b) = x \cdot W + b = \left(x^{[1]} \cdots x^{[p]}\right) \begin{pmatrix} w_1 \\ \vdots \\ w_p \end{pmatrix} + b$$

$$Z_x(W, b) = (\sum_{j=1}^{p} w_j x^{[j]}) + b$$

Nous appliquons ensuite à la sortie de la fonction Z_x, une **fonction d'activation**, notée A.

$$A : \left\{ \begin{array}{rcl} A(z) & \longrightarrow & A(z) = y_p \end{array} \right\}$$

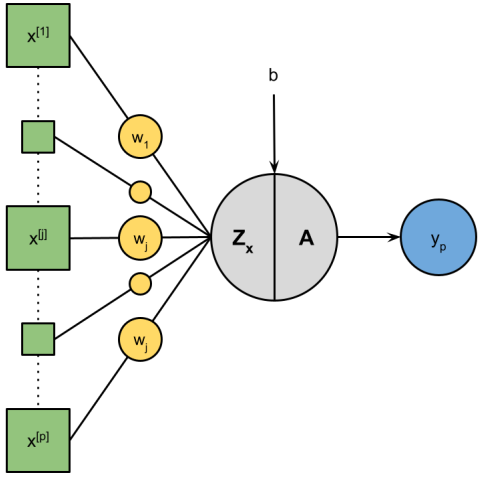

FIGURE 1 – Schéma du fonctionnement du perceptron

Remarque : Dans la suite, nous noterons Z la fonction Z_x.

Fonction d'activation

Le terme "fonction d'activation" est inspiré du concept biologique de "potentiel d'activation", qui correspond au seuil de stimulation nécessaire pour déclencher une réponse du neurone. Dans le neurone formel de **McCulloch-Pitts**, la fonction d'activation utilisée est Heaviside/Marche. Cependant, cette fonction n'est pas utile dans le cas du perceptron. Les fonctions d'activations doivent suivre des caractéristiques afin de coïncider avec des problèmes d'optimisations et d'apprentissages. Les principales caractéristiques dans le cas du perceptron sont :

— Non linéaire : soient E et F deux $\mathbb{K}$-espaces vectoriels. On appelle application linéaire entre E et F, une application $f : \left\{ \begin{array}{ccc} E & \longrightarrow & F \\ x & \longmapsto & f(x) \end{array} \right\}$ telle que

$$\forall\, x, y \in E,\ \forall\, \lambda, \mu \in \mathbb{K},\ f(\lambda x + \mu y) = \lambda f(x) + \mu f(y)$$

Ainsi, par contraposée :

$$\exists\, x, y \in E,\ \exists\, \lambda, \mu \in \mathbb{K},\ f(\lambda x + \mu y) \neq \lambda f(x) + \mu f(y)$$

— Différentiable : cette caractéristique permet d'utiliser la méthode d'apprentissage par excellence en Deep Learning, la **descente de gradient**.

— Monotone : la fonction est croissante ou décroissante sur un intervalle I.

Dans le cas de modèles bien plus complexes comme des réseaux de neurones (feed-forward), réseaux convolutifs, Transformers... d'autres caractéristiques sont à prendre en compte, mais elles dépendent considérablement du contexte dans lequel ces fonctions sont utilisées. Pour les réseaux de neurones feed-forward, des éléments tels que la profondeur du réseau, le nombre de neurones par couche et les fonctions d'activation utilisées (par exemple, ReLU, Sigmoid, Tanh) sont cruciaux, car ils influencent la capacité d'apprentissage et la performance du modèle.

Les fonctions d'activations présentes dans le répertoire GitHub :

Identité : $\left\{ \begin{array}{ccc} \mathbb{R} & \longrightarrow & \mathbb{R} \\ Id(x) & \longmapsto & x \end{array} \right\}$

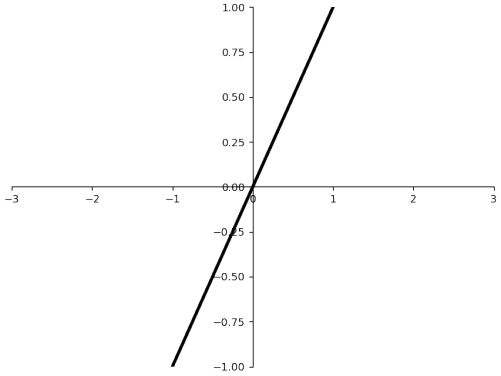

FIGURE 2 – Fonction Identité

$$\forall\, x \in \mathbb{R}\ \ H(x) = \left\{ \begin{array}{lll} 0 & \text{si} & x < 0 \\ 1 & & \text{sinon} \end{array} \right.$$

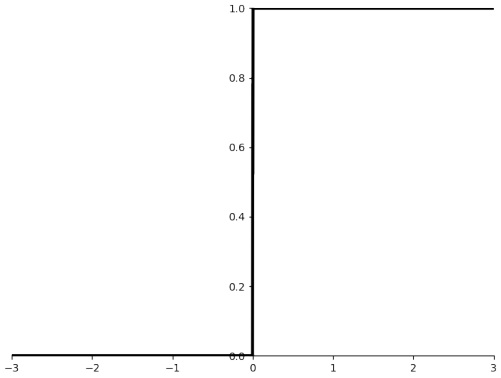

FIGURE 3 – Fonction Heaviside/Marche

Sigmoïde : $\left\{ \begin{array}{ccc} \mathbb{R} & \longrightarrow &]0\,;1[\\ x & \longmapsto & \frac{1}{1+e^{-x}} \end{array} \right\}$

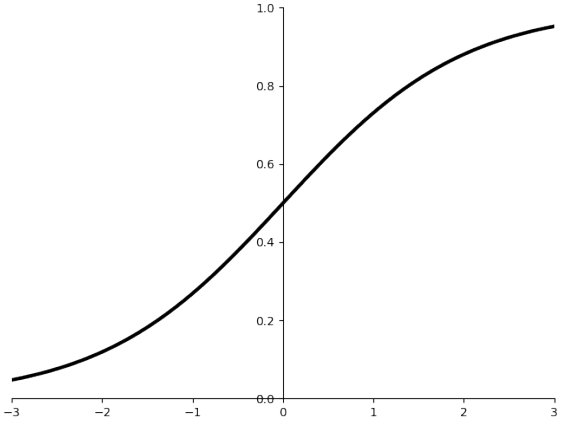

FIGURE 4 – Fonction Sigmoïde

ArcTan : $\left\{ \begin{array}{ccc} \mathbb{R} & \longrightarrow &]-\frac{\pi}{2}\,;\frac{\pi}{2}[\\ x & \longmapsto & tan^{-1}(x) \end{array} \right\}$

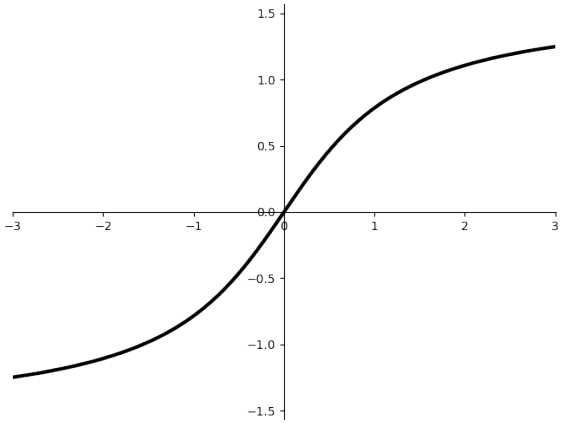

FIGURE 6 – Fonction ArcTan

Tangente Hyperbolique : $\left\{ \begin{array}{ccc} \mathbb{R} & \longrightarrow &]-1\,;1[\\ x & \longmapsto & \frac{e^x-e^{-x}}{e^x+e^{-x}} \end{array} \right\}$

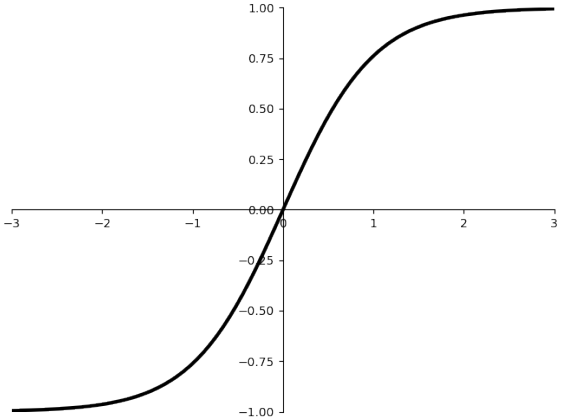

FIGURE 5 – Fonction Tangente Hyperbolique

Sigmum : $\left\{ \begin{array}{ccc} \mathbb{R} & \longrightarrow &]-1\,;1[\\ x & \longmapsto & \frac{x}{1+|x|} \end{array} \right\}$

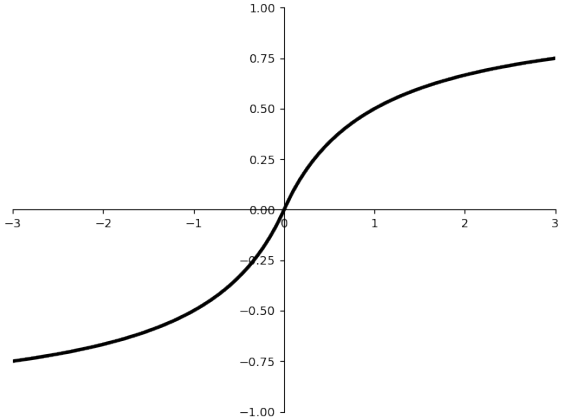

FIGURE 7 – Fonction Sigmum

Descente du Gradient

La méthode d'apprentissage du perceptron, et plus généralement en Deep Learning, repose sur la **descente de gradient**, qui appartient à la famille des algorithmes de descente. À l'origine, le perceptron de Frank Rosenblatt utilisait une méthode d'optimisation/d'apprentissage différente de la descente de gradient, un cas particulier de l'algorithme du gradient stochastique. Dans le cadre du perceptron, nous nous concentrons sur la descente de gradient à pas constant :

Algorithme : Descente de Gradient à pas constant (pour le perceptron)

$\alpha \in \mathbb{R}_+^*$

Pour tout i allant de 1 à N Faire

Calcul de $\frac{\partial f(W^i,b^i)}{\partial W^i}$ et $\frac{\partial f(W^i,b^i)}{\partial b^i}$

$$W^{i+1} = W^i - \alpha \frac{\partial f(W^i,b^i)}{\partial W^i}$$

$$b^{i+1} = b^i - \alpha \frac{\partial f(W^i,b^i)}{\partial b^i}$$

Fin Pour

À l'aide de cette méthode, nous mettons à jour les poids et le biais du perceptron.

Fonction Coût/Perte

Maintenant, nous devons définir la fonction f. Nous utilisons une fonction que l'on nomme fonction "coût" ou encore "perte". En Deep Learning, ces fonctions mesurent l'écart entre les valeurs prédites et les valeurs réelles. C'est cette fonction qui permet en partie à la descente de gradient de converger vers des minimums locaux ou globaux.

Erreur Quadratique Moyenne

On définit l'erreur quadratique moyenne (en anglais : **"Mean Squared Error"** comme étant la différence aux carrées entre les prédictions et les valeurs réelles :

$$\text{MSE} : \left\{ \begin{array}{ccc} \{0\,,1\} \times \mathbb{R} & \longrightarrow & \mathbb{R} \\ (y, y_p) & \longmapsto & (y - y_p)^2 \end{array} \right\}$$

La variable y_p représente une valeur de la composée de fonction A∘Z. Ainsi, cette composée représente la propagation des données dans le neurone.
Cette fonction coût est pratique, car y_p prend ces valeurs dans $\mathbb{R}$. Ce qui permet d'utiliser toutes les fonctions d'activations vues précédemment.

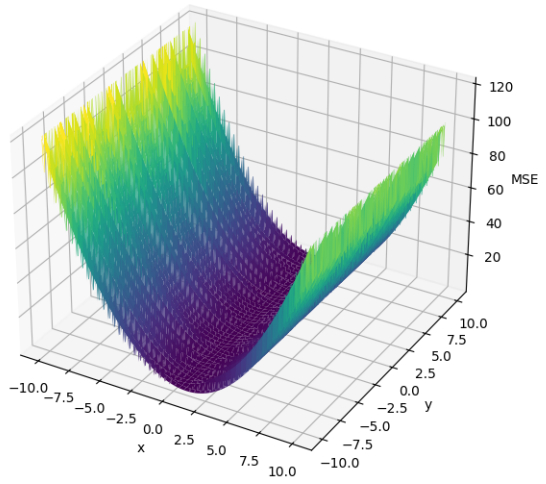

FIGURE 8 – Représentation MSE en 3D avec une génération aléatoire de y (valeurs réelles)

Remarque : À noter que cette représentation est un cas spécifique.

LogLoss

La fonction coût LogLoss est définie de cette façon pour la fonction d'activation sigmoïde :
Soit L la vraisemblance de l'échantillon Y = $(y_1 \dots y_n)$. Y est une variable discrète $\Longrightarrow$ Produit de probabilité.

$$L = \prod_{i=1}^{n} P(Y = y_i)$$

On en déduit à l'aide du graphique de la fonction sigmoïde et de la frotière de décision (axe y). Les valeurs à gauche sont de classe 0 et celles de droite sont de classe 1 :

$$\forall i \in [|1\,;n]\ y_i \in \{0\,;1\}\ P(Y=0) = (1 - A_i(z)) \text{ et } P(Y=1) = A_i(z)$$

$$\Longrightarrow L = \prod_{i=1}^{n} A_i^{y_i}(1 - A_i)^{1-y_i}$$

$$\Longrightarrow ln(L) = \sum_{i=1}^{n}(ln(A_i^{y_i}) + ln((1 - A_i)^{1-y_i})$$

$$\Longrightarrow ln(L) = \sum_{i=1}^{n}(y_i ln(A_i) + (1 - y_i)ln(1 - A_i))$$

$$\text{Normalisation} \Longrightarrow LogLoss = -\frac{1}{n}\sum_{i=1}^{n}(y_i ln(A_i) + (1 - y_i)ln(1 - A_i))$$

Ainsi, la fonction LogLoss peut-être écrite de deux manières suivantes :

$$\text{LogLoss} : \left\{ \begin{array}{rcl} \mathbb{M}_{p,1}(\mathbb{R}) \times \mathbb{R} & \longrightarrow &]-1\,;1[\\ (W,b) & \longmapsto & -\frac{1}{n}\sum_{i=1}^{n}(y_i ln(A_i(W,b)) \\ & & +(1-y_i)ln(1-A_i(W,b))) \end{array} \right\}$$

$$\text{LogLoss} : \left\{ \begin{array}{rcl}]0\,;1[& \longrightarrow &]-1\,;1[\\ z & \longmapsto & -\frac{1}{n}\sum_{i=1}^{n}(y_i ln(A_i(z)) \\ & & +(1-y_i)ln(1-A_i(z))) \end{array} \right\}$$

Cette deuxième forme de la fonction LogLoss nous montre pourquoi seule la fonction d'activation sigmoïde est utilisable (elle admet des valeurs dans]0 ; 1[).

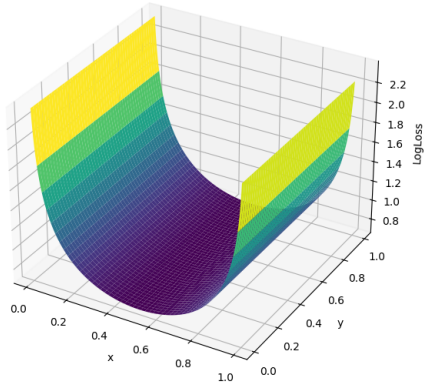

FIGURE 9 – Représentation LogLoss en 3D avec une génération aléatoire des valeurs réelles Y

Remarque : À noter que cette représentation est un cas spécifique.

Calcul des gradients

Maintenant, nous pourrions calculer les gradients des deux fonctions coûts pour chaque fonction d'activation si possible. Cependant, cela est inutile. Pourquoi ?
Dans le cas du perceptron cela peut-être fastidieux mais relativement faisable. Mais imaginons dans le cas d'un modèle plus complexe où l'unité de base est le perceptron, il s'agit de la torture. C'est pourquoi, nous utilisons la dérivation automatique. D'après Wikipédia, il s'agit d'un ensemble de techniques d'évaluation de la dérivée d'une fonction par un programme informatique. Dans l'implémentation, nous utilisons la bibliothèque jax permettant de faire cette dérivation automatique.