Université J.F. CHAMPOLLION Albi

Réseaux de neurones artificiels

Licence d'informatique S6

Thierry Montaut

2022

Réseaux de neurones artificiels

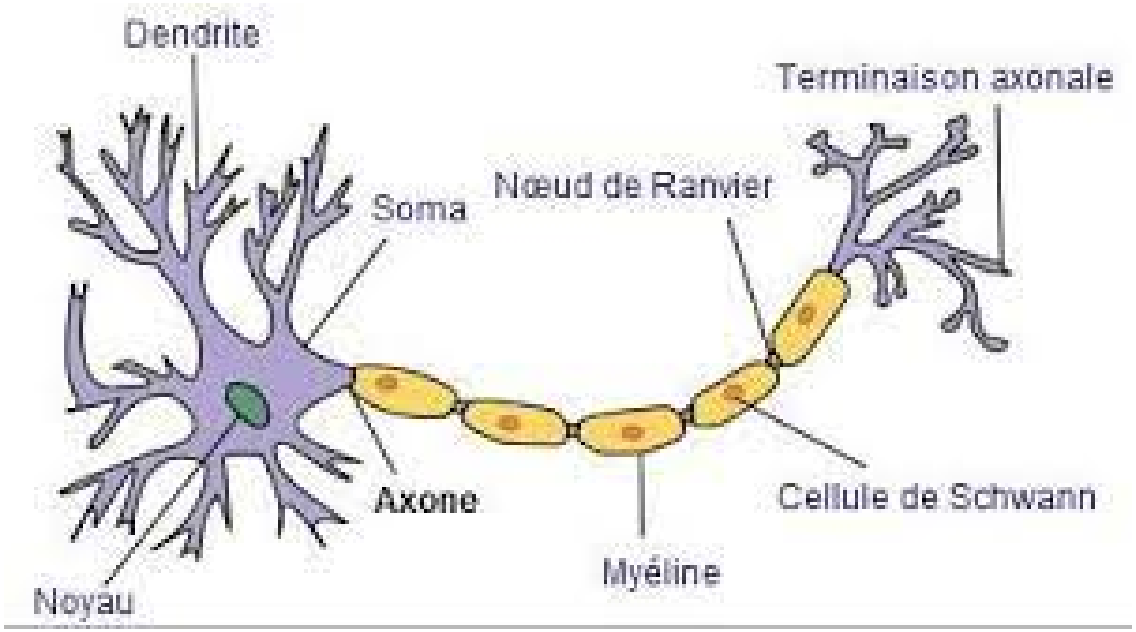
- Les récents succès de l'intelligence artificielle sont nombreux à reposer sur les réseaux de neurones et le deep learning est souvent synonyme d'IA pour le grand public.
- Les réseaux de neurones artificiels ne sont pourtant rien d'autre qu'un des nombreux modèles d'apprentissage supervisé.
- Sa particularité est de permettre de construire facilement des modèles très flexibles.
- Dans ce chapitre, nous aborderons les principes de base d'une classe particulière de réseaux de neurones artificiels, les perceptrons mono puis multi-couche, et de leur entraînement.

Le perceptron

- L'histoire des réseaux de neurones artificiels remonte aux années 1950 et aux efforts de psychologues et de neurologues comme McCulloch, Pitts et Rosenblatt pour comprendre le cerveau humain.
- Initialement, ils ont été conçus dans le but de modéliser mathématiquement le traitement de l'information par les réseaux de neurones biologiques qui se trouvent dans le cortex des mammifères.
- De nos jours, leur réalisme biologique importe peu et c'est leur efficacité à modéliser des relations complexes et non linéaires qui fait leur succès.
- Le premier réseau de neurones artificiels est le perceptron (Rosenblatt, 1957). Loin d'être profond, il comporte une seule couche et a une capacité de modélisation limitée.

Réseaux de neurones formels

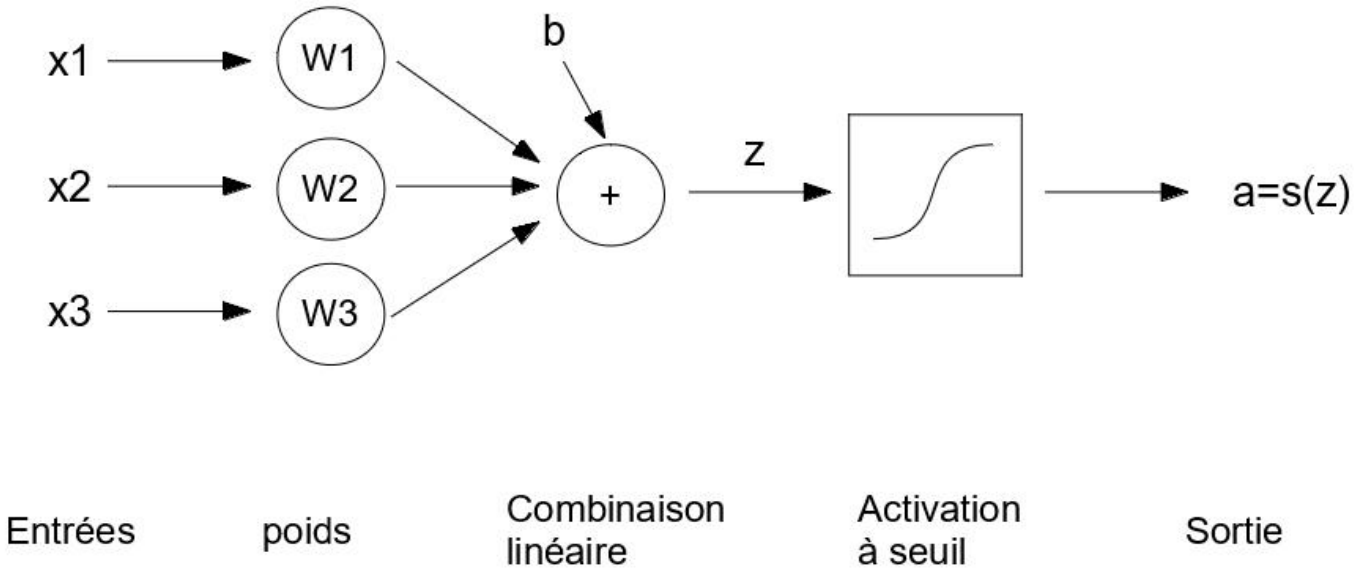
- Le neurone formel s'inspire directement du neurone biologique :



- Les études en neurophysiologie montrent que les dendrites correspondent aux entrées du neurone et l'axone à sa sortie. Le cerveau est ainsi constitué d'un réseau sophistiqué de neurones connectés les uns aux autres par le biais de dendrites et d'axones, d'entrées et de sorties.

Le neurone formel binaire

- Le modèle de neurone formel reproduit le comportement essentiel du neurone biologique : sa capacité à s'activer au delà d'un certain seuil :



Le neurone formel binaire

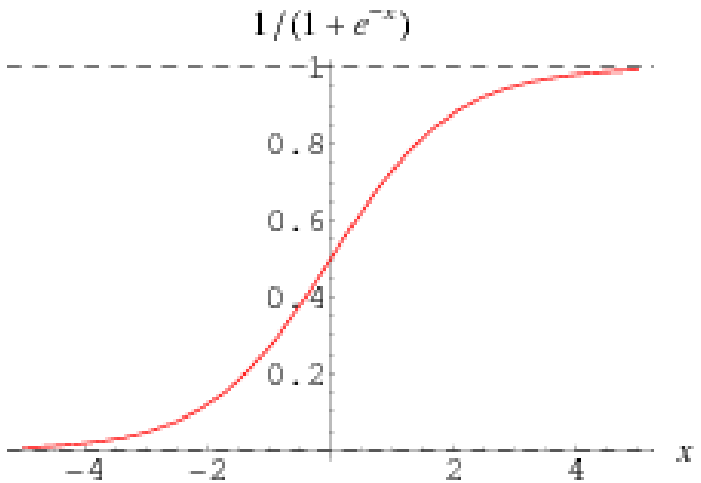
- Le modèle initial de McCulloch et Pitts est binaire : Les entrées x_i valent 0 ou 1 et sont pondérées par des poids $w_i \in [0,1]$. Si la combinaison linéaire $\sum_{i=1}^{n} w_i x_i + b$ est supérieure à un seuil donné, la sortie y vaut 1 (le neurone est activé), sinon elle vaut 0.
- Par exemple : Si l'entrée vaut (1,0,1), que les poids valent respectivement 0.5, 1 et 0.2, la combinaison vaudra 0.7. Si le seuil est fixé à 0.6, le neurone sera activé et sa sortie vaudra 1.

Le neurone formel numérique

On peut améliorer le comportement du neurone formel en remplaçant la simple fonction à seuil par une fonction numérique $a(\sum_{i=1}^{n} w_i x_i + b)$, appelée fonction d'activation, qui donnera toutes les valeurs réelles entre 0 et 1, transformant la sortie en une probabilité que y soit égal à 1. Les réseaux de neurones peuvent en utiliser plusieurs mais la plus répandue est la fonction sigmoïde

$$y = a(z) = \frac{1}{1 + e^{-z}}$$

:



Neurone multi-classe

Dans le cas d'un problème de classification multi-classe, on modifiera l'architecture du perceptron de sorte à n'avoir non plus 1 mais p neurones dans la couche de sortie, où p est le nombre de classes.

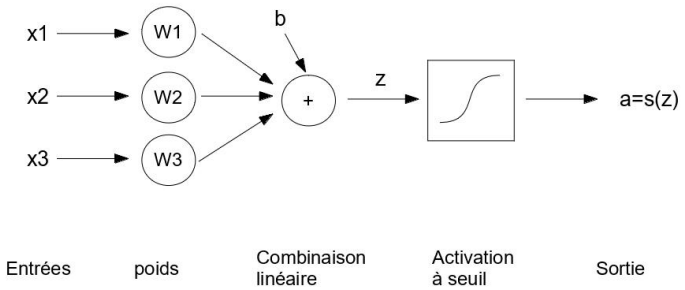
Propagation avant (forward propagation)

étant donné un perceptron et un jeu de données d'entrée

$$X = \begin{pmatrix} x_1^{(1)} & x_1^{(2)} & \dots & x_1^{(m)} \\ \vdots & \vdots & & \vdots \\ x_n^{(1)} & x_n^{(2)} & \dots & x_n^{(m)} \end{pmatrix}$$

On cherche à déterminer les valeurs de sortie du perceptron.

Propagation avant (forward propagation)



On calcule successivement :

$$z^{(k)} = \sum_{i=1}^{n} w_i x_i^{(k)} + b$$

$$a^{(k)} = \frac{1}{1 + e^{-z^{(k)}}}$$

Propagation avant - Vectorisation

En posant : $W = \begin{pmatrix} w_1, & w_2, & \dots & , w_n \end{pmatrix}$ et $B = (b)$
On obtient les écritures matricielles :

$$Z = WX + B$$

$$A = \frac{1}{1 + e^{-Z}}$$

Qui pourront être accélérés et parallélisés par Numpy.

Entraînement

- Pour entraîner un perceptron, nous allons chercher dans un premier temps à minimiser l'erreur $a - y$ commise (nous chercherons dans un deuxième temps à maximiser la vraisemblance des données.)
- Nous allons supposer que les observations $(x^{(k)}, y^{(k)})$ ne sont pas disponibles simultanément, mais qu'elles sont observées séquentiellement. (A l'image des réseaux de neurones biologiques qui s'adaptent constamment en fonction des signaux qu'ils reçoivent.)
- Nous allons donc utiliser un algorithme d'entraînement incrémental, qui s'adapte à des observations arrivant les unes après les autres.
- Nous utilisons pour cela un algorithme du gradient en ajustant les paramètres w, pour chaque nouvelle observation, dans la direction opposée au gradient de l'erreur d'observation.

Minimisation de l'erreur quadratique

Dans un premier temps on va chercher à minimiser l'erreur quadratique :

$$L(x^{(k)}, y^{(k)}) = \frac{1}{2}(y^{(k)} - a(x^{(k)}))^2.$$

Apprentissage et propagation arrière (version 1)

- Pour chaque nouveau couple de données $(x^{(k)}, y^{(k)})$ on ajuste les paramètres $w_1, w_2, \ldots, b$ par une descente de gradient :

$$w_i \leftarrow w_i - \eta \frac{\partial L(x^{(k)}, y^{(k)})}{\partial w_i}$$

$$b \leftarrow b - \eta \frac{\partial L(x^{(k)}, y^{(k)})}{\partial b}$$

- Il nous faut donc calculer ces dérivées partielles...

Gradient d'un neurone

En dérivant $L(x^{(k)}, y^{(k)}) = a(z(w_1, \ldots, w_n, b))$ comme une composée, on obtient successivement :

- $\dfrac{\partial \mathcal{L}}{\partial a} = y - a(x)$
- $\dfrac{\partial a}{\partial z} = a(1-a)$
- $\forall i \in [\![1, n]\!]$, $\dfrac{\partial z}{\partial w_i} = x_i$ et $\dfrac{\partial z}{\partial b} = 1$.

Gradient d'un neurone

Donc $\forall j \in [\![1, n]\!]$,

$$\frac{\partial \mathcal{L}}{\partial w_i} = (y - a(x)).a(x).(1 - a(x)).x_i$$

et

$$\frac{\partial \mathcal{L}}{\partial b} = (y - a(x)).a(x).(1 - a(x)).$$

- La valeur $a(x)$ est mémorisée de la propagation avant. On a tout intérêt à commencer par la dernière dérivée partielle et à la mémoriser pour calculer les autres.
- On peut alors mettre à jour les paramètres $(w_1, \ldots, w_n, b)$ par une descente de gradient
- On itère cette descente de gradient plusieurs fois sur l'ensemble complet des données, jusqu'à convergence.

Maximisation de la vraisemblance

- Une alternative est de chercher à maximiser la vraisemblance du modèle relativement aux données.
- Cette vraisemblance est

$$L = \prod_{k=1}^{m} P(a(x^{(k)}) = y^{(k)})$$

- Comme les probabilités sont inférieures à 1, ce produit tend rapidement vers 0 quand le nombre de données augmente, et comme d'autre part on préfère les sommes aux produits, on choisit d'utiliser le logarithme de la vraisemblance :

$$L_2 = \ln(\prod_{k=1}^{m} P(a(x^{(k)}) = y^{(k)})) = \sum_{k=1}^{m} P(a(x^{(k)}) = y^{(k)})$$

Vraisemblance

Si on suppose que la sortie $a(x^{(k)})$ suit une loi de Bernouilli de paramètre $a_k = a(x^{(k)})$: $P(a(x^{(k)}) = y^{(k)}) = a_k^y(1-a_k)^{1-y}$.
on obtient finalement (après normalisation) la vraisemblance :

$$\mathcal{L} = \frac{1}{m}\sum_{k=1}^{m} y^{(k)}\ln(a_k) + (1-y^{(k)})\ln(1-a_k).$$

Dont il faut calculer les dérivées partielles par rapport aux variables $(w_1, \ldots w_n, b)$ pour permettre une descente de gradient.

Gradient

En dérivant $\mathcal{L} = \mathcal{L}(a(z(w_1, \ldots w_n, b)))$ comme une composée, on obtient successivement :

- $\dfrac{\partial \mathcal{L}}{\partial a_i} = \dfrac{1}{m} \displaystyle\sum_{i=1}^{m} \dfrac{y_i - a_i}{ai(1 - a_i)}$.
- $\dfrac{\partial a_i}{\partial z} = a_i(1 - a_i)$
- $\forall j \in [\![1, n]\!], \quad \dfrac{\partial z}{\partial w_j} = x_j$ et $\dfrac{\partial z}{\partial b} = 1$.

Gradient

Donc $\forall j \in [\![1, n]\!]$,

$$\frac{\partial \mathcal{L}}{\partial w_j} = \frac{1}{m} \sum_{i=1}^{m} (y_i - a_i) x_j^{(i)}.$$

et

$$\frac{\partial \mathcal{L}}{\partial b} = \frac{1}{m} \sum_{i=1}^{m} (y_i - a_i).$$

On peut alors mettre à jour les paramètres $(w_1, \ldots, w_n, b)$ par une descente de gradient