Machine Learning : Réseaux de neurones

D. Cornu

Automne 2019

Le cerveau comme inspiration

≪ There is a fantastic existence proof that learning is possible, which is the bag of water and electricity (together with a few trace chemicals) sitting between your ears [...] wich is the squishy thing that your skull protects ≫

** Stephen Marsland*

Le cerveau comme inspiration

≪ There is a fantastic existence proof that learning is possible, which is the bag of water and electricity (together with a few trace chemicals) sitting between your ears [...] wich is the squishy thing that your skull protects ≫

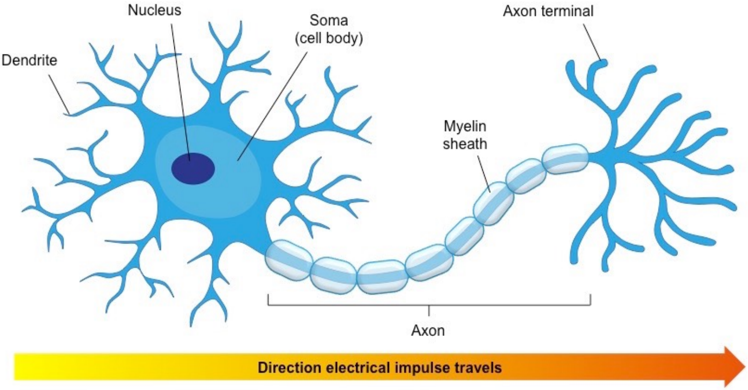
** Stephen Marsland*

Le cerveau fait exactement ce qui nous intéresse en science :

- Traiter des données bruitées ou incohérentes
- Fonctionner avec un nombre important de dimensions
- Donner un résultat le plus souvent correct
- Trouver une réponse en un temps très court
- Rester robuste malgré la perte de neurones avec l'âge

Neurone

Brique élémentaire (10^{11} dans le cerveau) $=$ une base pour reproduire l'apprentissage



Fait la somme de signaux d'entrée. Si celle-ci est suffisante il envoie un signal le long de son axone.

Réseau de neurones

Une Synapse est une connexion entre deux neurone (10^{14}).
Neurone = Unité de calcul simple "Tire" ou "Ne tire pas" (1 ou 0)

→ **Calculateur massivement parrallèle de 10^{11} unités**

Une Synapse est une connexion entre deux neurone (10^{14}).
Neurone = Unité de calcul simple "Tire" ou "Ne tire pas" (1 ou 0)

$\rightarrow$ **Calculateur massivement parrallèle de 10^{11} unités**

Outils de l'apprentissage :

Les synapses représentent la **force** de la connexion entre deux neurones. Apprentissage = modification de ces liaisons
$\rightarrow$ **plasticité**.

La **loi de Hebb** définit une règle d'apprentissage : la connexion entre deux neurones se renforce lorsqu'ils tirent au même moment
$\rightarrow$ **conditionnement**.

Neurone Artificiel

Il est la brique élémentaire des réseaux qui seront construits.

Il est basé sur un **modèle mathématique** inspiré du neurone biologique.

C'est le modèle de McCulloch and Pitts.

Neurone Artificiel

Il est la brique élémentaire des réseaux qui seront construits.
Il est basé sur un **modèle mathématique** inspiré du neurone biologique.
C'est le modèle de McCulloch and Pitts.

Il est constitué :

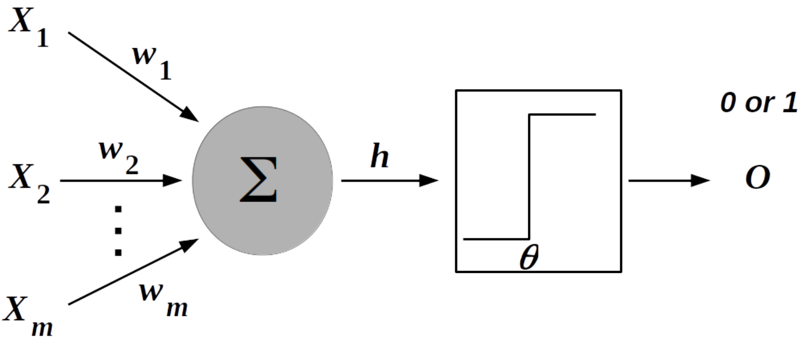
- D'un vecteur d'entrée X_i qui représente les différentes dimensions d'un même objet.
- D'un ensemble de poids w_i qui lient les entrées au neurone.
- D'une fonction de somme $h = \sum ...$ qui définie comment ces poids doivent êtres associés aux entrées correspondantes et envoyés vers le neurone.
- D'une fonction d'activation $g(h)$, qui définie si le neurone doit passer dans un état actif "1" ou non "0" en fonction du résultat de la somme précédente.

Neurone Artificiel

Il est la brique élémentaire des réseaux qui seront construits.

Il est basé sur un **modèle mathématique** inspiré du neurone biologique.

C'est le modèle de McCulloch and Pitts.



$$h = \sum_{i=1}^{m} X_i \omega_i \qquad O = g(h) = \begin{cases} 1 & \text{if} \quad h > \theta \\ 0 & \text{if} \quad h \leq \theta \end{cases}$$

Apprentissage supervisé $\rightarrow$ faire retrouver à un réseau la sortie attendue pour une entrée donnée.
Quel intérêt ? La sortie correcte est déja connue à l'avance ...

$\rightarrow$ **Généralisation :** Retrouver des "motifs" dans les données, et prédire les futures entrées dont le résultat est inconnu.

Comment changer les paramètres pour qu'un neurone apprenne ?

Apprentissage supervisé $\rightarrow$ faire retrouver à un réseau la sortie attendue pour une entrée donnée.
Quel intérêt ? La sortie correcte est déja connue à l'avance ...

$\rightarrow$ **Généralisation :** Retrouver des "motifs" dans les données, et prédire les futures entrées dont le résultat est inconnu.

Comment changer les paramètres pour qu'un neurone apprenne ?

Entrée et Sortie fixées, l'apprentissage repose donc uniquement sur les **poids W_i et la limite d'activation θ**

Apprentissage

Il sagit d'un modèle **Supervisé**, chaque vecteur d'entrée est donc associé à une réponse attendue : la target t.

Supposons que l'on présente un vecteur d'entrée au neurone et que celui-ci ne donne pas la réponse attendue.
Il y a m poids w_i (avec i de 1 à m) qui sont connectés à ce neurones, correspondant à chaque noeud d'entrée.

Comment modifier les poids ? :

- Si le neurone tire alors qu'il ne devrait pas, les poids sont trop grands
- Si le neurone ne tire pas alors qu'il devrait, les poids sont trop petits

On calcule une erreur : $y_k - t_k$
La différence entre la sortie et la cible pour le neurone k

Cette diférence est multipliée par la valeur de chaque entrée et utilisée pour **changer le poid associé** $\Delta w_{ij} = -\eta\,(y_k - t_k) \times x_i$

Taux d'apprentissage

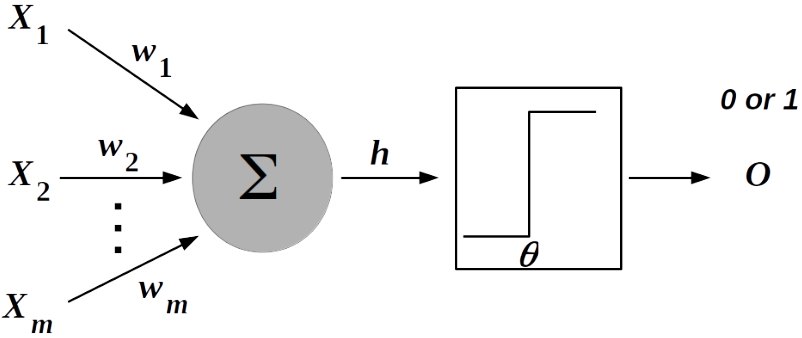
On défini un **taux d'apprentissage** η qui permet de quantifier la vitesse à laquelle l'algorithme va modifier les poids.

- **Une valeur trop importante de ce taux rends l'algorithme instable.**
- **Une valeur trop faible ralentie l'apprentissage.**

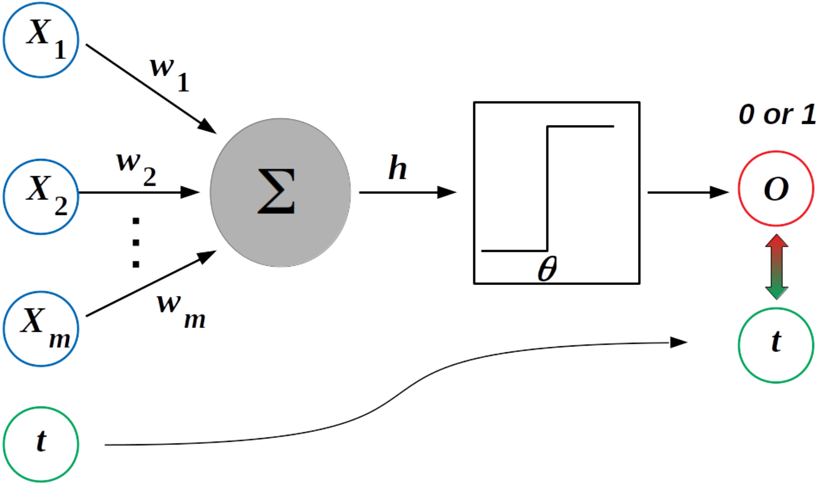
Une valeur typique se situe entre $0.1 < \eta < 0.4$ ce qui permet de rendre l'algorithme plus stable mais aussi plus résistant aux erreurs (bruit).

Le choix réside dans la qualité attendue des données d'apprentissage.

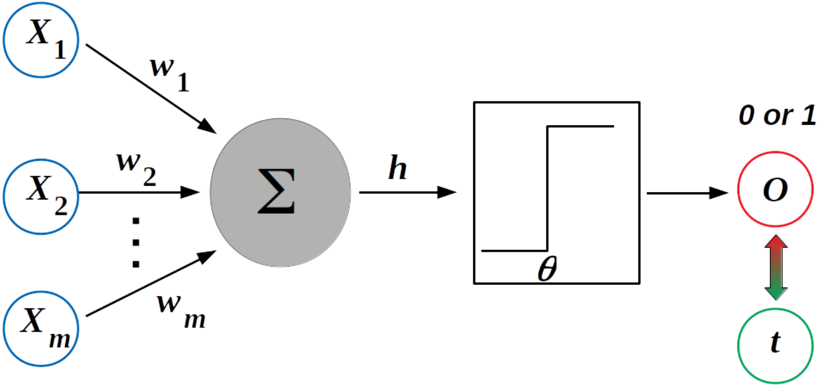
Apprentissage illustré



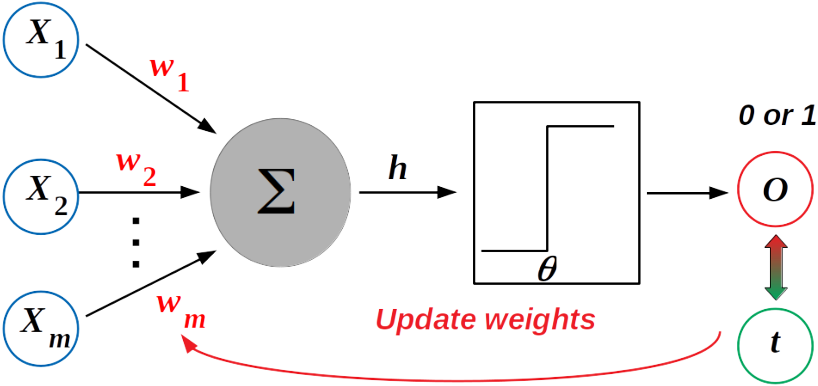
Apprentissage illustré



Apprentissage illustré



Apprentissage illustré

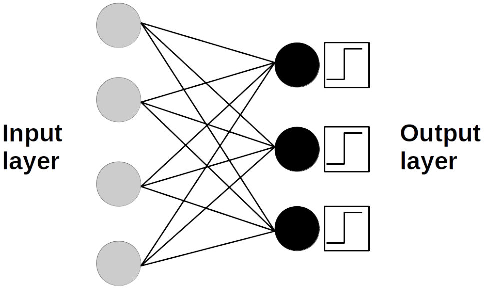


$$\omega_i \leftarrow \omega_i - \eta\,(o - t) \times X_i$$

Le Perceptron

Un seul neurone est dans la majorité des cas insuffisant pour représenter correctement un problème. La façon la plus simple d'en ajouter est de le faire sur une "couche/layer". Ces neurones restent indépendants, chacun représentant une séparation linéaire dans l'espace des paramètres d'entrée, mais l'ensemble peut servir à encoder une information.

- Les neurones sont indépendants entre eux
- Les poids sont séparés pour chaque neurone
- Le nombre de dimensions en entrée et le nombre de neurones (sorties) sont indépendants
- Les dimensions d'entrée et de sortie sont imposées par les données
- La sortie est un shéma de 0 et de 1 qui encode l'information voulue



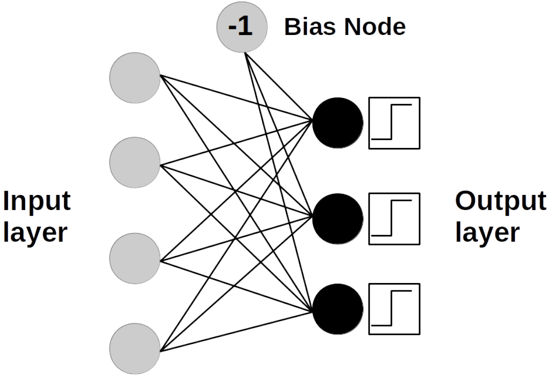
Vecteur d'entrée connecté de manière pondérée à des neurones de McCulloch

The Perceptron

L’entrée biaisée

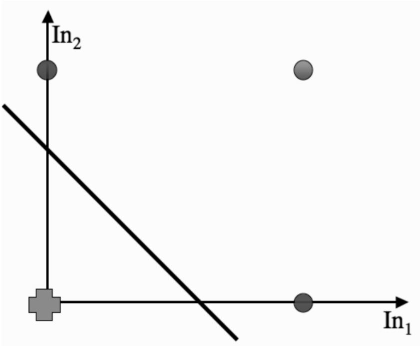
Problème :
Si toutes les entrées sont à 0 alors tous les neurones en sortie auront le même comportement... Pour contrer cet effet la limite d’activation peut etre ajusté, mais difficile à mettre en oeuvre

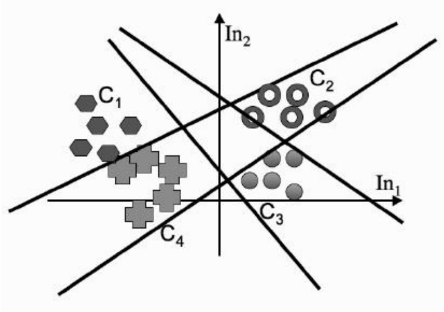
Solution :
Ajouter un noeud d’entrée qui a une valeur fixée à -1 et des poids associés. Définira le comportement pour des entrées proches de zero.



Note sur la séparabilité linéaire

Les neurones du perceptron tel que décris ici ne sont capables que de tracer des séparations linéaires entre les classes étudiées.





Il est donc nécessaire de représenter les données en entrée de manière astucieuse en ajoutant un nombre de dimensions suffisant afin de garantir leur séparabilité.

Réalisme du model ?

Un neurone réel reste assez différent :

- La somme des inputs peut etre non linéaire
- La sortie n'est pas binaire mais est une séquence d'impulsion ce qui contient de l'information supplémentaire
- La limite de tir varie au cours du temps
- L'interrogation des neurones n'est pas séquentielle (mise à jour Asynchrone)
- Les poids peuvent être négatifs ou positifs mais pas de passage de l'un à l'autre
- Les synapses peuvent revenir sur le neurone d'origine (feedback)
- Après avoir tiré, un neurone à un temps de recharge

Le model théorique est cependant suffisant pour apprendre des images, représenter des fonctions, faire du classement, ...

Perceptron : Algorithme

- Initialisation :
  - Définir tous les poids w_i avec de petites valeurs aléatoire (positives et négatives)
- Entrainement
  - pour T itérations ou jusqu'à ce que la sortie soit correcte
    - $\star$ Pour chaque vecteur d'entrée
      - · Calculer l'activation de chaque neurone j avec la fonction d'activation g :

$$y_j = g\left(\sum_{i=0}^{m} w_{ij} x_i\right) = \begin{cases} 1 & \text{if } \sum_{i=0}^{m} w_{ij} x_i > 0 \\ 0 & \text{if } \sum_{i=0}^{m} w_{ij} x_i \leq 0 \end{cases} \tag{1}$$

      - · Mettre à jour les poids individuellement avec :

$$w_{ij} \leftarrow w_{ij} - \eta\,(y_j - t_j) \cdot x_i \tag{2}$$

- Rappel
  - Calculer l'activation de chaque neurone j pour chaque vecteur

Première application : la porte OU

Un exemple très simple de l'application de cet algorithme est de tenter de lui apprendre **la porte logique "OU"**.

Le perceptron prend donc une entrée à deux dimensions et ne possède qu'un seul neurone qui doit donner la sortie attendue sous la forme d'un 0 ou d'un 1.

Codez donc un premier programme qui declare un tableau contenant les entrées possibles pour cette porte logique, et un tableau pour les cibles correspondantes.

Declarez les variables necessaire à l'algorithme, initialisez les deux poids à des petites valeurs aléatoires, et tentez de faire un entrainement sur quelques itérations. **N'oubliez pas le neurone de biais !**

Affichez la sortie, à chaque iteration pour observer la convergence.

The Pima Indian Dataset

Fait parti du UCI Machine Learning repository, très utile pour récupérer des données de test.

Donne 8 mesures éffectuées sur une population de natifs américains nommés les Pimas, et les classe selon que la personne soit atteinte de diabète ou non.

Malgré ses limitations le perceptron une fois correctement entrainé devrait être capable de trouver le résultat attendu dans 60 à 70% des cas.

Modifiez votre programme précédent pour lire les données pima. Adapatez le nombre d'entrées et les calculs effectués. Mesurez la "précision" de votre algorithme et son évolution au fil des itérations.