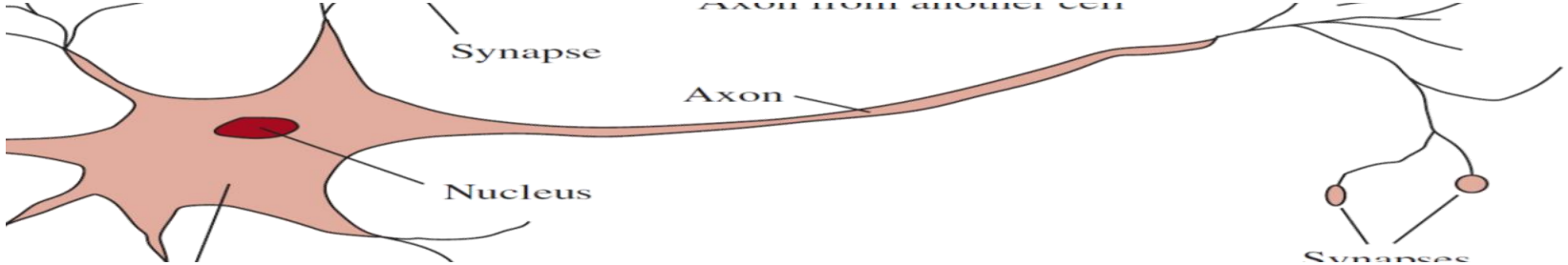
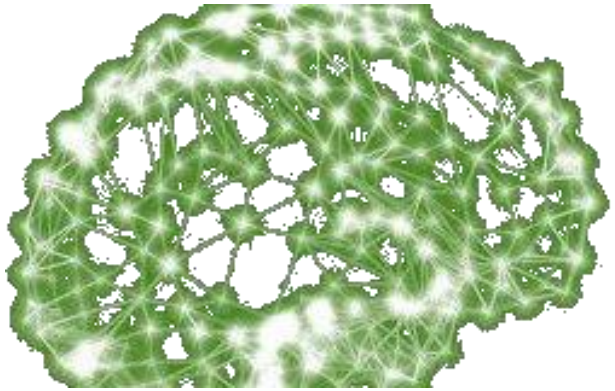
Les réseaux de neurones (RN)

Université de Mila
Master I: Apprentissage automatique
Hadjadj abdelhalim

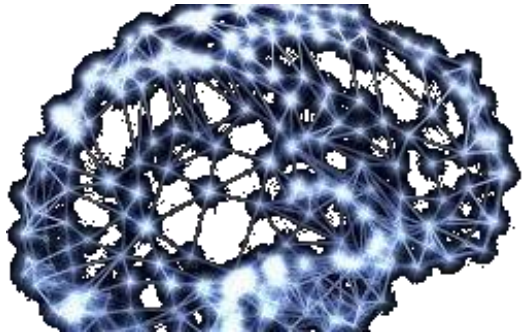


Les réseaux de neurones

- Depuis longtemps, les scientifiques se sont intéressés à comprendre les mécanismes de fonctionnement du cerveau qui est un réseau de ($\sim 10^{11}$) neurones.
- Le cerveau a des capacités pour effectuer plusieurs applications (ex. la vision, la reconnaissance de la parole, l'apprentissage du langage, etc.).



Cerveau versus ordinateur

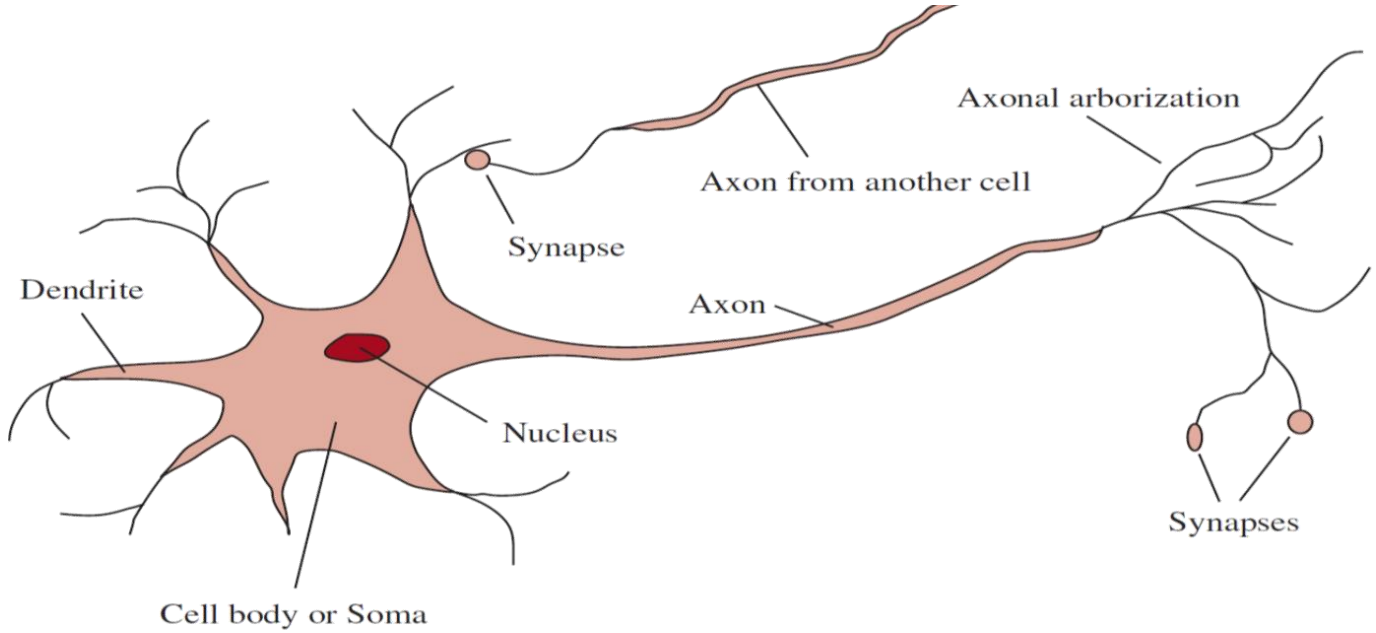


- Le cerveau ~10^{11} unités de traitement parallèles (c.à.d.., neurones).
- Le cerveau possède une grande connectivité entre les neurones ~2×10^{4} par neurone. Les connections sont appelées des synapses.
- les unités de traitement et la mémoire sont distribuées sur le réseaux. Le traitement est fait par les neurones et la mémoire est encodée dans les synapses.

- L'ordinateur possède un seul processeur
- le processeur est actif et la mémoire est séparée du processeur et elle est passive.

Les réseaux de neurones

- Les réseaux de neurones artificiels sont des modèles mathématiques inspirés de la biologie.
- La brique de base de ces réseaux, le neurone artificiel, était issu au départ d'une volonté de modélisation du fonctionnement d'un neurone biologique.



Principes des Réseaux de Neurones (RN)

1. Un réseau de neurones est un modèle informatique inspiré du fonctionnement du cerveau.

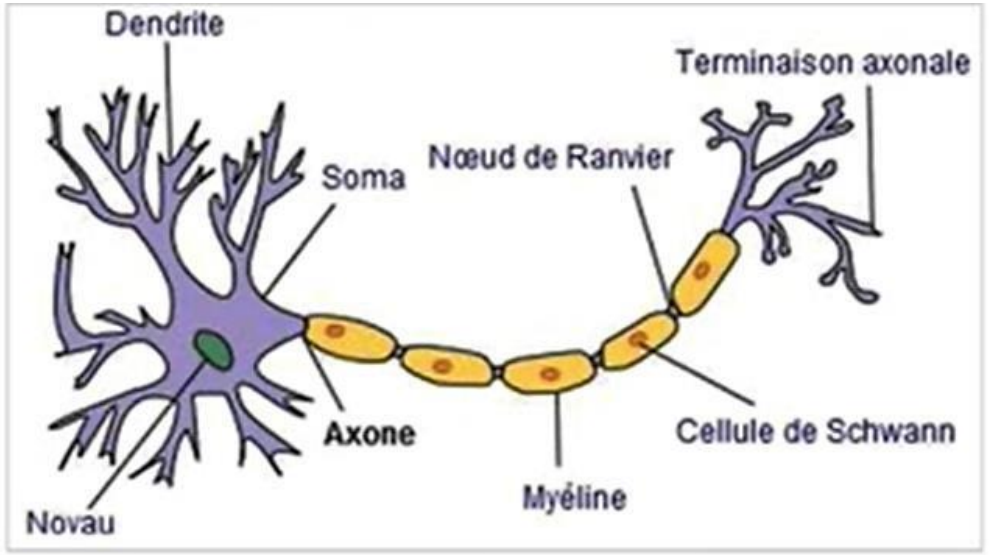
- Il est composé de **neurones artificiels**, organisés en **couches** (entrée, cachée(s), sortie).
- Chaque neurone effectue une opération simple : une somme pondérée suivie d'une fonction d'activation.

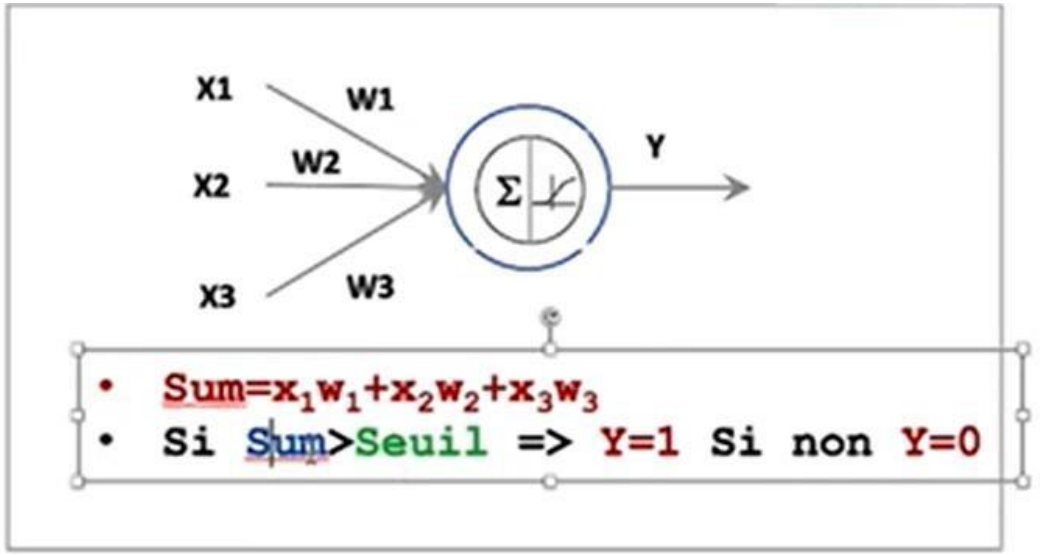
Exemple : Une entrée x, un poids w, une sortie

$y=f(w_x+b)$

Le perceptron

- Le perceptron est l'élément de traitement de base d'un réseau de neurones, conçu en 1958 par Rosenblatt.
- Il possède un vecteur d'entrée $x = (x_1, x_2, \ldots, x_D)$, où $X_d \in \mathbb{R}$, qui peut venir de l'environnement (entrée) ou bien d'autres perceptrons.
- Pour chaque entrée x_d, on associe un poids (synaptique) w_d.
- La sortie y est une somme pondérée des entrées.

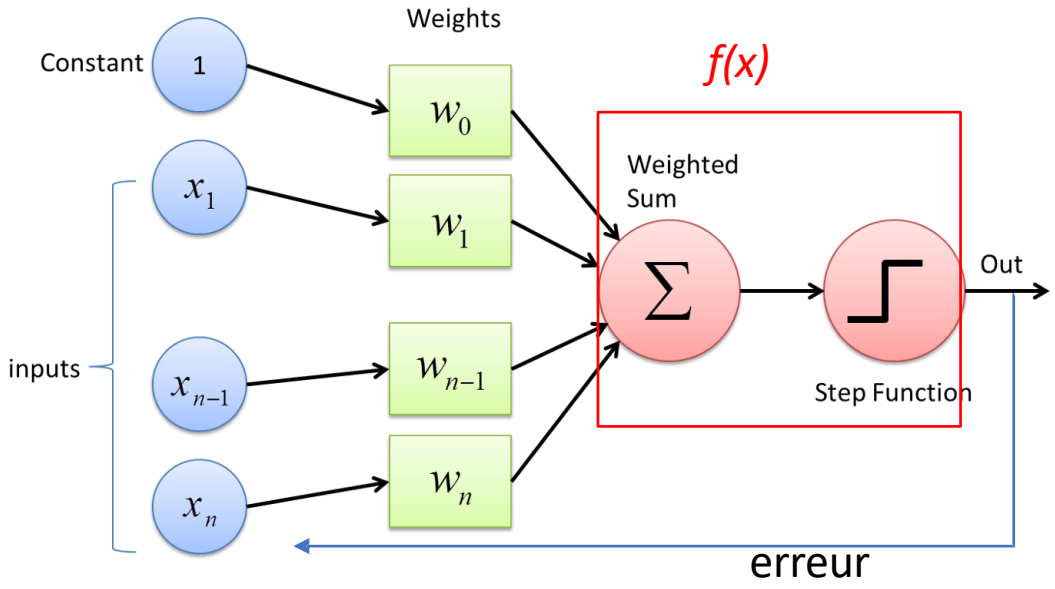




Le perceptron

- La sortie y est une somme pondérée des entrées.
- w_0 est la valeur de l'intercepte, qui est le poids d'une entrée fictive x_0 = +1.

$$y = f(x) = w_0 + \sum_{d=1}^{D} w_d x_d$$



Perceptron et régression

- En ayant un ensemble de données *D*, on peut estimer le vecteur des poids $\boldsymbol{w}$ d'une manière hors ligne.
- On place ensuite ces poids dans le perceptron pour calculer la sortie de nouvelles entrées.

- On peut aussi utiliser un entrainement en ligne où l'ensemble *D* n'est pas disponible au complet au départ.

« **Les données arrivent l'une après l'autre** »

- Dans l'entrainement en ligne, la fonction d'erreur peut être définie pour une donnée à la fois.

Perceptron et régression(en ligne)

- En posant $\tilde{\mathbf{w}}$: $(w_0, w_1, \cdots, w_D)$ et $x = (x_0, x_1, \cdots, x_D)$.
- L'erreur de sortie produite par un exemple d'entrainement $(x^{(i)}, y^{(i)}, i \in \{1, \ldots, N\}$, sera définie comme suit:

$$E^{(i)}(\tilde{\mathbf{w}}) = \frac{1}{2}\left(y^{(i)} - f\left(x^{(i)}\right)\right)^2$$

- Par montée du gradient, la mise à jour en ligne des D +1 éléments de sera

$$\Delta\tilde{\mathbf{w}}_d = \alpha\left(y^{(i)} - f\left(x^{(i)}\right)\right)\tilde{x}_d{}^{(i)}; \quad d = 0, \ldots, D.$$

α est le facteur d'entrainement.

Perceptron et régression(en ligne)

- De manière générale:
- $\boldsymbol{mise\ à\ jour} = facteur \times (\boldsymbol{sortie\ désirée - sortie\ calculée}) \times \boldsymbol{donnée}$
- algorithme d'apprentissage en ligne:

$\widetilde{\mathbf{w}}_d \leftarrow rand(-0.01, +0.01); d = 1, \dots, D.$

Répéter

  Pour chaque donnée $\left(x^{(i)}, y^{(i)}\right), i = 1, \dots, N;$

    Calculer $f\left(x^{(i)}\right);$

    Pour chaque dimension $d = 1, \dots, D;$

      $\widetilde{\mathbf{w}}_d = \widetilde{\mathbf{w}}_d + \alpha\left(y^{(i)} - f\left(x^{(i)}\right)\right) \tilde{x}_d{}^{(i)}$

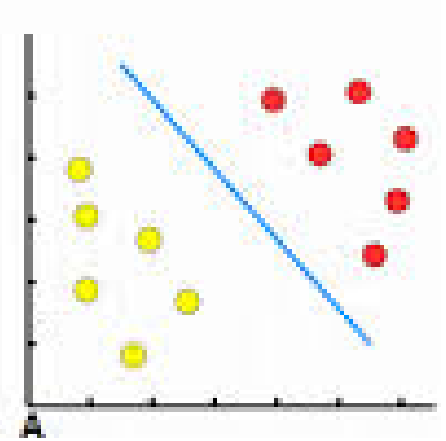
    Fin

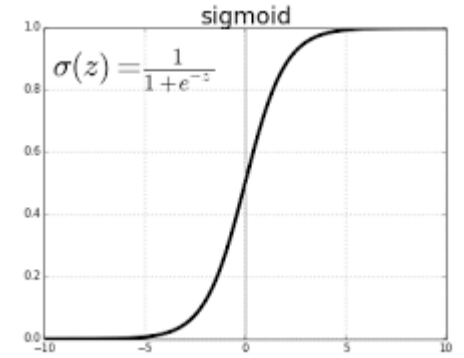
  Fin

Jusqu'à convergence

Perceptron et classification binaire

- Le perceptron permet de définir un hyperplan qui sépare l'espace des données en deux régions distinctes, correspondant aux deux classes C_1 *et* C_2, dans le cas où ces classes sont linéairement séparables.
- Lorsqu'on souhaite obtenir une estimation probabiliste de l'appartenance à une classe, on remplace la sortie du perceptron par une fonction d'activation sigmoïde.
- La probabilité a posteriori d'appartenir à la classe C_1 est alors donnée par :





$$P(C_1|\mathbf{x}) = \quad g(\tilde{x}) = \frac{1}{1 + \exp(-\tilde{\mathbf{w}}^T \tilde{x})}$$

Perceptron et classification binaire

- L'erreur associée à un exemple d'entraînement (x_i,y_i) est définie par la fonction de perte suivante :

$$E^{(i)}(\tilde{\mathbf{w}}) = -y^{(i)} \ln\left(g\left(\tilde{x}^{(i)}\right)\right) - \left(1 - y^{(i)}\right) \ln\left(1 - g\left(\tilde{x}^{(i)}\right)\right)$$

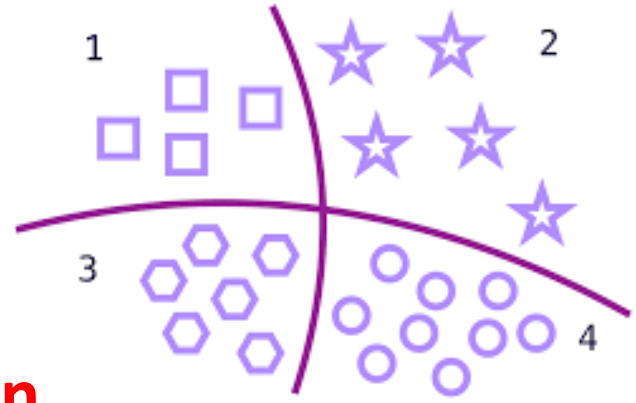
- où g(xi) représente la sortie du modèle
- Pour minimiser cette erreur, on applique une mise à jour des poids par la méthode de la montée du gradient. La mise à jour en ligne des D+1 composantes du vecteur de poids w se fait selon la règle suivante

$$\Delta\tilde{\mathbf{w}}_d = \alpha\left(y^{(i)} - g\left(\tilde{x}^{(i)}\right)\right) \tilde{x}_d{}^{(i)} \qquad d = 0, \ldots, D.$$

- où α est le taux d'apprentissage (ou facteur d'entraînement).

Perceptron et multiple classes

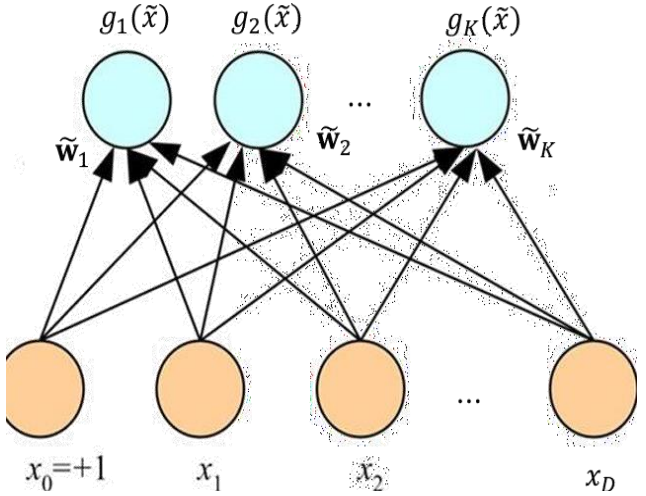
- Lorsqu'on souhaite traiter un problème de classification comportant K>2 classes, on associe **un perceptron à chaque classe**. Ainsi, on définit K vecteurs de poids w_k, pour chaque classe k∈{1,…,K}



- Pour obtenir une sortie probabiliste, on utilise la **fonction Softmax**, qui permet de calculer la **probabilité a posteriori** que l'entrée appartienne à chaque classe :

$$g_k(\tilde{x}) = \frac{\exp\left(-\tilde{\mathbf{w}}_k{}^T \tilde{x}\right)}{\sum_j \exp\left(-\tilde{\mathbf{w}}_j{}^T \tilde{x}\right)}$$

$$g_k(\tilde{x}) \in [0,1]$$



Perceptron et multiple classes

- Soient les vecteurs de poids $w_1, w_2, \ldots, w_K$ associés aux *K* classes $C_1, \ldots, C_K$.
- Pour chaque exemple d'entraînement i, l'étiquette réelle est représentée par un vecteur $y^{(i)}=(y^i_1, y^i_2, \ldots, y^i_K)$ où :

$$y_i^k = \begin{cases} 1 & \text{si l'exemple appartient à la classe } C_k \\ 0 & \text{sinon} \end{cases}$$

- La fonction de coût associée à un exemple d'entraînement *i* est définie par l'entropie croisée multi-classes:

$$E^{(i)} = -\sum_{k=1}^{K} y_k{}^{(i)} \, ln\left(g_k(\tilde{x}^{(i)})\right)$$

- Par **montée du gradient** , la mise à jour en ligne des composantes du vecteur w_k pour chaque classe k est :

$$\Delta \tilde{\mathbf{w}}_{kd} = \alpha\left(y_k{}^{(i)} - g_k\left(\tilde{x}^{(i)}\right)\right) \tilde{x}_d{}^{(i)}; \qquad d = 0, \ldots, D.$$

Perceptron et multiple classes

- Exemple d'algorithme d'apprentissage en ligne:

$\widetilde{\mathbf{w}}_{kd} \leftarrow rand(-0.01, +0.01);\ k = 1, \dots, K; d = 1, \dots, D.$

Répéter

Pour chaque donnée $\left(x^{(i)}, y^{(i)}\right), i = 1, \dots, N;$

Calculer $g_k\left(\tilde{x}^{(i)}\right), k = 1, \dots, K;$

Pour chaque dimension $d = 1, \dots, D;$

$$\widetilde{\mathbf{w}}_{kd} = \widetilde{\mathbf{w}}_{kd} + \alpha\left(y_k{}^{(i)} - g_k\left(\tilde{x}^{(i)}\right)\right) \tilde{x}_d{}^{(i)}$$

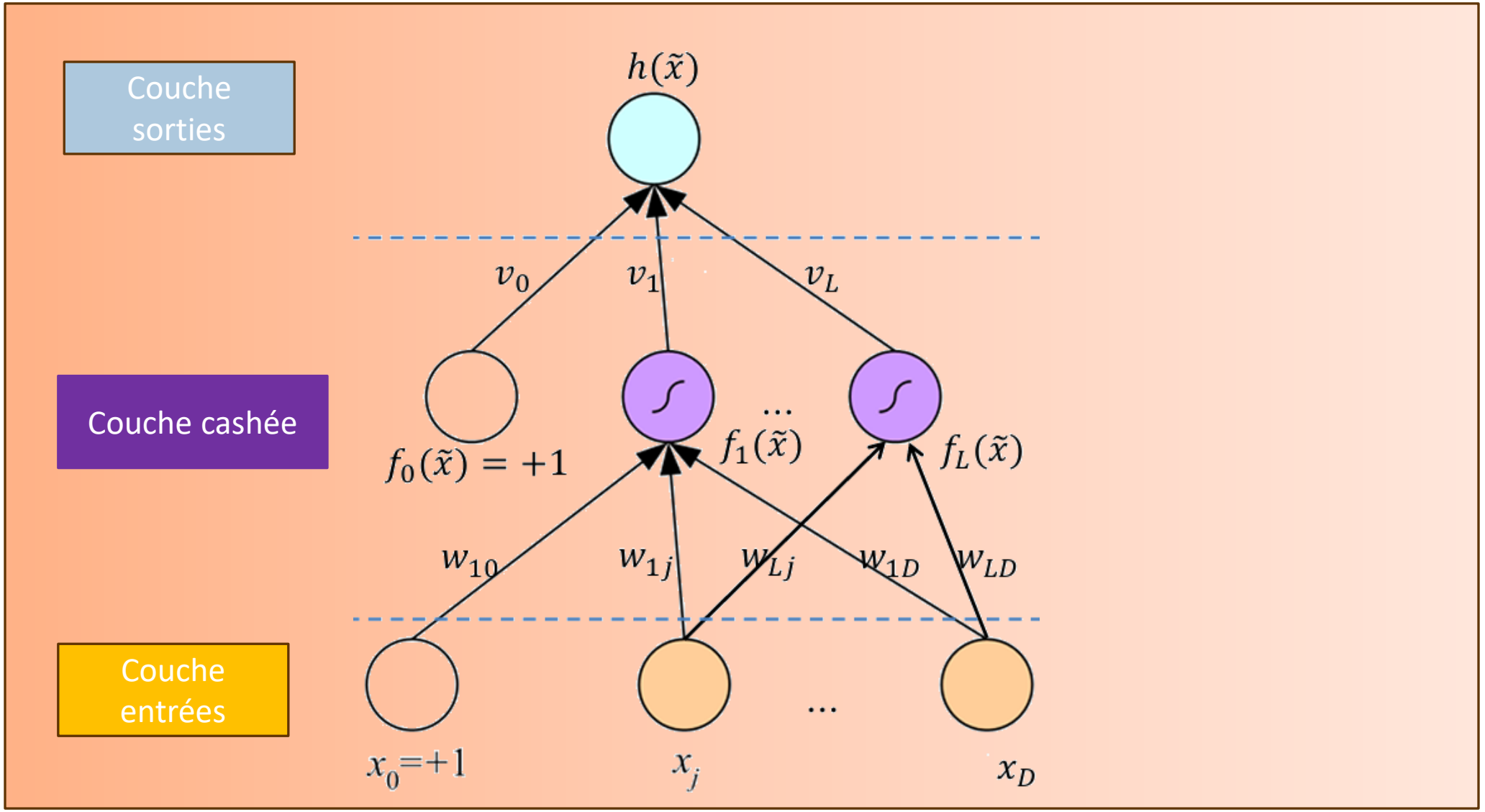
Fin

Fin

Jusqu'à convergence

Perceptron multicouches (PM)

- **Le perceptron à une seule couche** ne peut modéliser que des **relations linéaires**.
- Il n'est donc **pas adapté** à l'approximation de **fonctions non linéaires**.
- Pour remédier à cela, on peut introduire des **couches cachées** entre la couche d'entrée et la couche de sortie, ce qui permet de traiter des tâches de **classification** ou de **régression non linéaires**.
- Par exemple, avec **deux couches**, les **sorties de la première couche** servent d'**entrées à la seconde**, rendant le modèle plus expressif.


Couche sorties
Couche cashée
Couche entrées
$h(\tilde{x})$
v_0
v_1
v_L
$f_0(\tilde{x}) = +1$
$f_1(\tilde{x})$
...
$f_L(\tilde{x})$
w_{10}
w_{1j}
w_{Lj}
w_{1D}
w_{LD}
x_0=+1
x_j
...
x_D

Perceptron Multicouche (PM) pour la Régression Non Linéaire

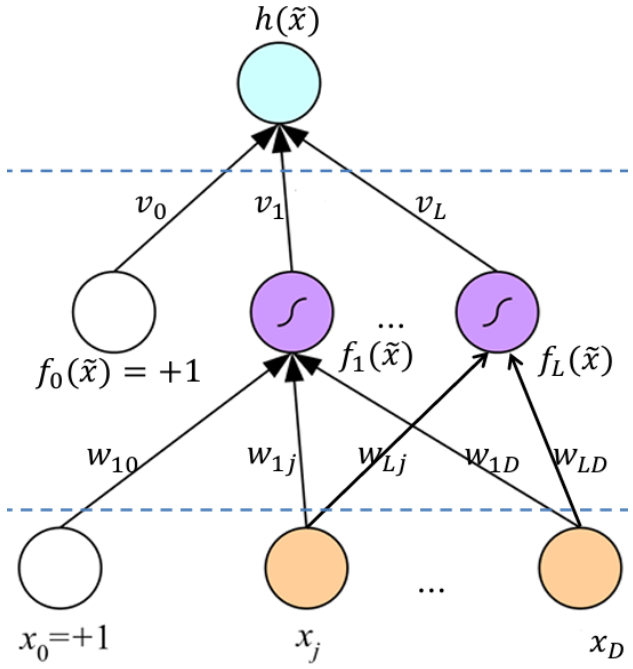
- Pour chaque fonction d'activation **non linéaire** $f_l(x)$, avec $l \in \{1,\ldots,L\}$ on définit :

$$f_l(\tilde{x}) = \frac{1}{1 + \exp[-(\sum_{d=0}^{D} w_{ld}\tilde{x}_d)]}$$

- Les sorties *h(x)* sont des perceptrons dans la seconde couche prenant les sorties de la première couche comme entrées.

$$h(\tilde{x}) = \sum_{l=0}^{L} v_l f_l(\tilde{x})$$

- Noter qu'on a ajouté aussi une entrée fictive $f_0(x) = 1$.



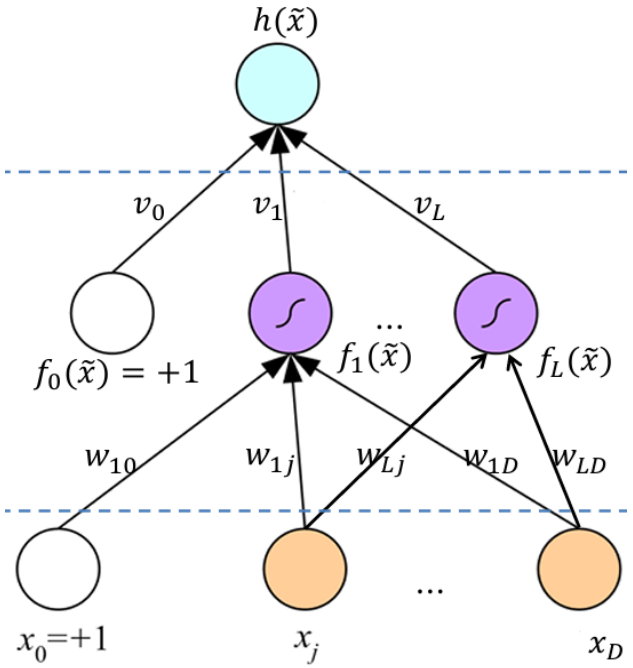
Perceptron Multicouche (PM) pour la Régression Non Linéaire

- **Première couche cachée**

Elle effectue une **transformation non linéaire** de l'espace d'entrée de dimension D vers un nouvel espace de dimension L, grâce à l'application des fonctions sigmoïdes.
Cela permet de **capturer des relations complexes et non linéaires** entre les variables.

- **Seconde couche (sortie)**

Elle applique une **fonction linéaire** sur les activations issues de la couche précédente.
Cette combinaison permet d'approximer des fonctions non linéaires globales.



Perceptron Multicouche (PM) pour la Régression Non Linéaire

On peut ajouter d'autres couches intermédiaires pour implémenter des fonctions très complexes. Seulement, le réseaux deviendra compliqué pour l'entrainement et l'interprétation.

université de Mila
Master I: Apprentissage automatique
Hadjadj abdelhalim

ALGORITHME DE PROPAGATION EN ARRIÈRE

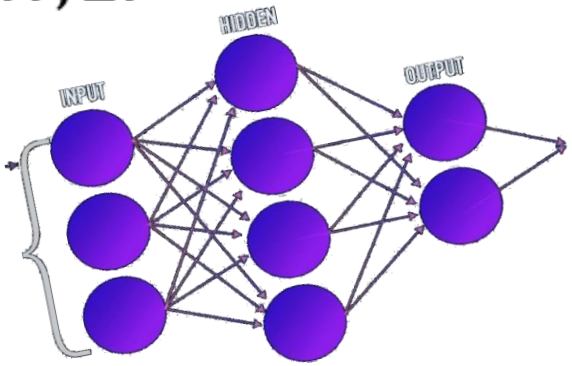
L'entraînement d'un **Perceptron Multicouche (MLP)** suit une logique similaire à celle d'un **Perceptron Simple**, mais avec des différences clés dues aux **fonctions d'activation non linéaires** dans les couches cachées.

- **Similarités avec le Perceptron Simple**
  - **Mise à jour des poids** : Comme dans le Perceptron Simple, les poids sont ajustés pour minimiser l'erreur de prédiction.

$$E^{(i)} = \frac{1}{2}\left(y^{(i)} - h^{(i)}\right)^2$$

  - **Règle de descente de gradient** : On utilise toujours une **règle de mise à jour** basée sur le gradient de l'erreur.

$$\Delta v_l = \alpha\left(y^{(i)} - h^{(i)}\right) f_l{}^{(i)}; \;\; l = 0, \ldots, L.$$

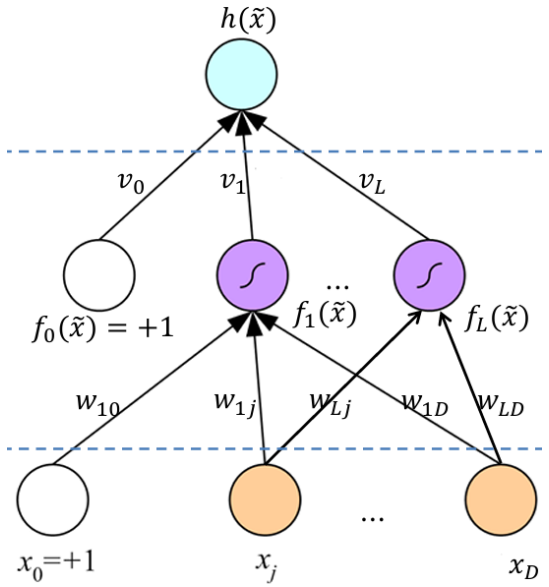


ALGORITHME DE PROPAGATION EN ARRIÈRE

- Les poids w_{ld} peuvent être mis à jour par la règle de chaine (backpropagation). On aura alors d'une part:

$$\frac{\partial E^{(i)}}{\partial w_{ld}} = \frac{\partial E^{(i)}}{\partial h}\frac{\partial h^{(i)}}{\partial f_l}\frac{\partial f_l{}^{(i)}}{\partial w_{ld}}$$

- La mise à jour du poids w_{ld} sera alors:



$$\Delta w_{ld} = \alpha \frac{\partial E^{(i)}}{\partial w_{ld}}$$

$$= \alpha \underbrace{\frac{\partial E^{(i)}}{\partial h}} \times \underbrace{\frac{\partial h^{(i)}}{\partial f_l}} \times \underbrace{\frac{\partial f_l{}^{(i)}}{\partial w_{ld}}}$$

$$= \alpha \left(y^{(i)} - h^{(i)}\right) \quad v_l \quad f_l{}^{(i)}(1 - f_l{}^{(i)})\tilde{x}_d{}^{(i)}$$

ALGORITHME DE PROPAGATION EN ARRIÈRE

$$\Delta w_{ld} = \alpha \frac{\partial E^{(i)}}{\partial w_{ld}} = \alpha \frac{\partial E^{(i)}}{\partial h} \times \frac{\partial h^{(i)}}{\partial f_l} \times \frac{\partial {f_l}^{(i)}}{\partial w_{ld}}$$

$$= \alpha \left(y^{(i)} - h^{(i)}\right) v_l {f_l}^{(i)} (1 - {f_l}^{(i)}) {\tilde{x}_d}^{(i)}$$

$$\frac{\partial E^{(i)}}{\partial h} = \frac{\partial \left[\frac{1}{2}\left(y^{(i)} - h^{(i)}\right)^2\right]}{\partial h} = \left(y^{(i)} - h^{(i)}\right)$$

$$\frac{\partial h^{(i)}}{\partial f_l} = \frac{\partial \sum_{l=0}^{L} v_l f_l(\tilde{x})}{\partial f_l} = v_l$$

$$\frac{\partial {f_l}^{(i)}}{\partial w_{ld}} = \frac{\partial \frac{1}{1 + \exp[-(\sum_{d=0}^{D} w_{ld} \tilde{x}_d)]}}{\partial w_{ld}} = {f_l}^{(i)} (1 - {f_l}^{(i)}) {\tilde{x}_d}^{(i)}$$

ALGORITHME DE PROPAGATION EN ARRIÈRE

- Le produit $\left(y^{(i)} - h^{(i)}\right) v_l$ agit comme un terme d'erreur de l'unité f_l. Elle a été propagée en arrière de l'erreur $E^{(i)}$ au unité cachées.

- Ce terme désigne l'erreur de sortie pondérée par la responsabilité de l'unité cachée manifestée par le poids v_l.

- Le terme ${f_l}^{(i)}(1 - {f_l}^{(i)}){\tilde{x}_d}^{(i)}$ est le produit de la dérivée de la fonction sigmoïde ${f_l}^{(i)}$ et la dérivée de la somme pondérée par les poids w_{ld}.

- On peut mettre à jour les poids des deux couches de manière alternative pour avoir une bonne convergence.

PM et classification binaire

- Avec deux classes $C1$ et $C2$, la variable de sortie va être une fonction sigmoïde:

$$g(\tilde{x}) = \frac{1}{1 + \exp[-(\sum_{l=0}^{L} v_l f_l(\tilde{x}))]}$$

- L'erreur de la classification pour un exemple d'apprentissage (x^i ,y^i) sera donnée par

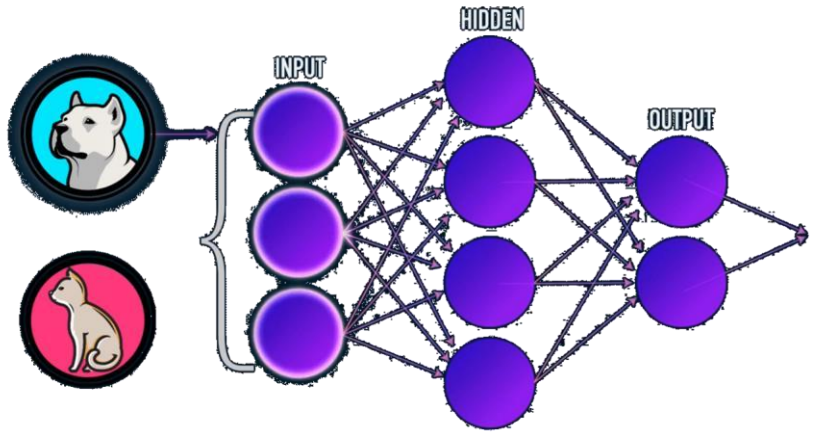
$$E^{(i)} = -y^{(i)} \ln\left(g\left(\tilde{x}^{(i)}\right)\right) - \left(1 - y^{(i)}\right) \ln\left(1 - g\left(\tilde{x}^{(i)}\right)\right)$$

PM et classification binaire

- Les mise à jour en ligne qui vont s'opérer sur les coefficients du PM sont données comme suit: $\Delta v_l = \alpha\left(y^{(i)} - g^{(i)}\right) f_l{}^{(i)}$

$$\Delta w_{ld} = \alpha \left(y^{(i)} - g^{(i)}\right) v_l \; f_l{}^{(i)}\left(1 - f_l{}^{(i)}\right)\tilde{x}_d{}^{(i)}$$

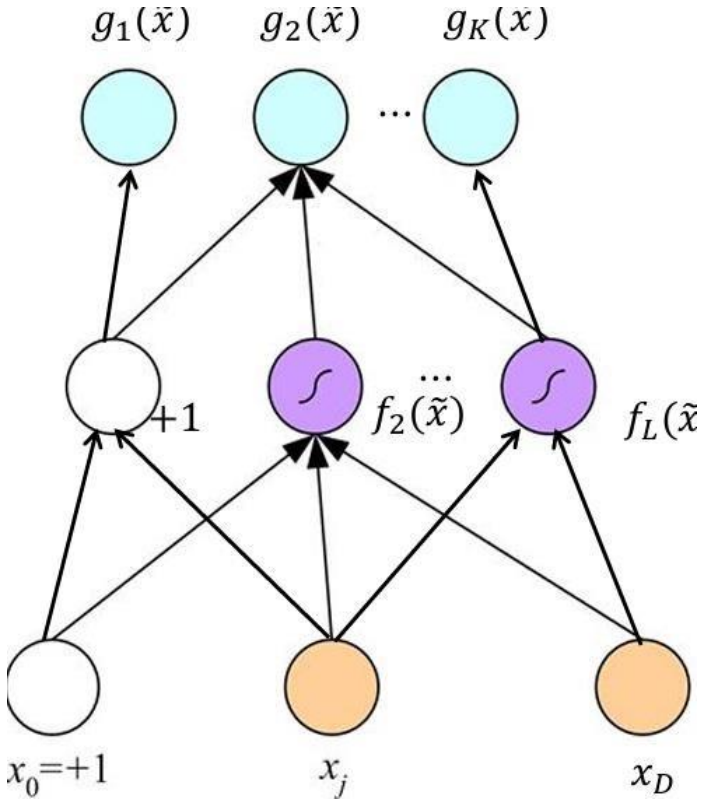
- les mises à jour de la régression et de la classification ont une ressemblance, elles diffèrent pour les sorties h^i (pour la régression) et g^i (pour la classification).



PM et multiple classes

- La probabilité à posteriori peut être calculé en utilisant la fonction Softmax.

$$g_k(\tilde{x}) = \frac{\exp\left(-\tilde{\mathbf{v}}_k{}^T f_k(\tilde{x})\right)}{\sum_l \exp\left(-\tilde{\mathbf{v}}_l{}^T f_l(\tilde{x})\right)}$$



PM et multiple classes

- La mise à jour des paramètres du PM est faite comme suit:

$$\Delta v_{kl} = \alpha\left(y_k{}^{(i)} - g_k{}^{(i)}\right) f_l{}^{(i)}$$

$$\Delta w_{ld} = \alpha \sum_{k=1}^{K} \left(y_k{}^{(i)} - g_k{}^{(i)}\right) \; v_{kl} \; f_l{}^{(i)}\left(1 - f_l{}^{(i)}\right)\tilde{x}_d{}^{(i)}$$

PM et multiple classes

Algorithme:

$\tilde{\mathbf{w}}_{ld} \leftarrow rand(-0.01, +0.01)$; $\tilde{\mathbf{v}}_{kl} \leftarrow rand(-0.01, +0.01)$;

Répéter

Pour chaque donnée $\left(x^{(i)}, y^{(i)}\right), i \in \{1, \dots, N\}$;

Calculer $f_l\left(\tilde{x}^{(i)}\right)$, pour $l \in \{1, \dots, L\}$;

Calculer $g_k\left(\tilde{x}^{(i)}\right)$, pour $k \in \{1, \dots, K\}$;

Calculer $\Delta\tilde{\mathbf{v}}_{kl}$, pour $k \in \{1, \dots, K\}, l \in \{1, \dots, L\}$;

$\tilde{\mathbf{v}}_{kl} = \tilde{\mathbf{v}}_{kl} + \Delta\tilde{\mathbf{v}}_{kl}$;

Calculer $\Delta\tilde{\mathbf{w}}_{ld}$, pour $l \in \{1, \dots, L\}, d \in \{1, \dots, D\}$;

$\tilde{\mathbf{w}}_{ld} = \tilde{\mathbf{w}}_{ld} + \Delta\tilde{\mathbf{w}}_{ld}$;

Fin

Jusqu'à convergence

Fin de chapitre 07