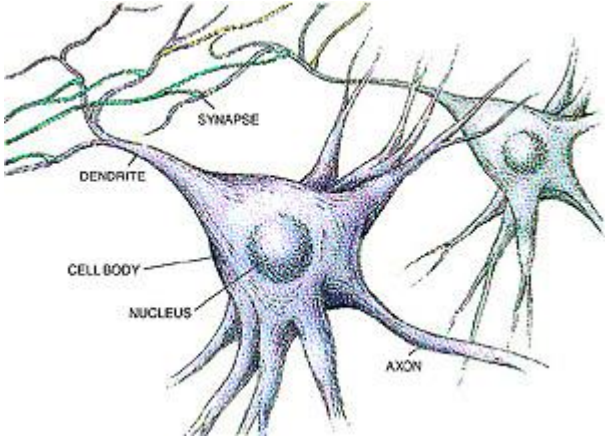
Les réseaux de neurones (RN)

Principes des RN

- Depuis longtemps, les scientifiques se sont intéressés à comprendre les mécanismes de fonctionnement du cerveau qui est un réseau de ($\sim 10^{11}$) neurones.



- Par exemple, 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

- L'ordinateur possède un seul processeur alors que le cerveau $\sim 10^{11}$ unités de traitement parallèles (c.à.d.., neurones).

- Le cerveau possède une grande connectivité entre les neurones $\sim 2 \times 10^{4}$ par neurone. Les connections sont appelées des synapses.

- Dans l'ordinateur, le processeur est actif et la mémoire est séparée du processeur et elle est passive.

- Dans le cerveau, 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.

Les réseaux de neurones (RN)

PERCEPTRON ET RÉGRESSION

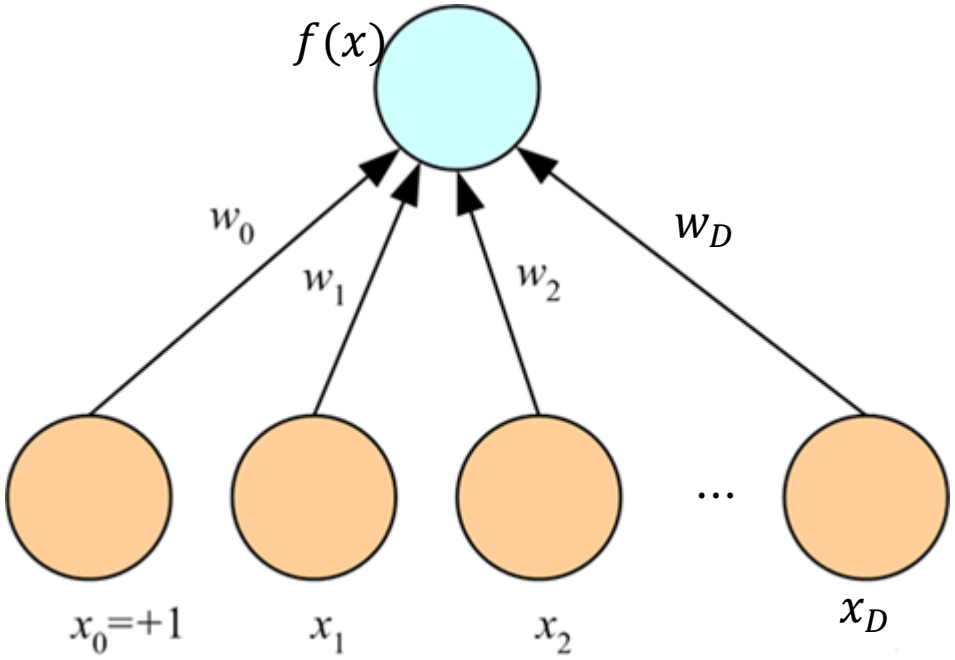
Le perceptron: élément de base

- En intelligence artificielle (IA), le but est plutôt de développer des réseaux de neurones artificiels pour construire de meilleurs systèmes.
- Le perceptron est l'élément de traitement de base d'un réseau de neurones.
- Il possède un vecteur d'entrée $x = (x_1, x_2, \dots, x_D)$, 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.

Perceptron et régression

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

- w_0 est la valeur de l'intercepte, qui est le poids d'une entrée fictive $x_0 = +1$.



Perceptron et régression

- En ayant un ensemble de données $\mathcal{D}$, on peut estimer le vecteur des poids $\tilde{\mathbf{w}}$ 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 $\mathcal{D}$ 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 posant $\tilde{\mathbf{w}} = (w_0, w_1, \cdots, w_D)$ et $\tilde{x} = (x_0, x_1, \cdots, x_D)$.
- L'erreur de sortie produite par un exemple d'entrainement $\left(x^{(i)}, y^{(i)}\right), i \in \{1, \ldots, N\}$, 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 $\tilde{\mathbf{w}}$ 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.

Algorithme d'apprentissage en ligne (régression)

- De manière générale:

$$\boldsymbol{mise}\ \text{à}\ \boldsymbol{jour} = facteur \times (\boldsymbol{sortie\ désirée} - \boldsymbol{sortie\ calculée}) \times \boldsymbol{donnée}$$

- Exemple d'algorithme d'apprentissage en ligne:

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

Répéter

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

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

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

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

    Fin

  Fin

Jusqu'à convergence

Les réseaux de neurones (RN)

PERCEPTRON ET CLASSIFICATION

Perceptron et classification binaire

- Le perceptron définit un hyperplan, et de ce fait divise l'espace des données en deux parties qui forment deux classes C_1 et C_2 si elles sont séparables.

- Si nous avons besoin de la probabilité à posteriori de chaque classe, nous utiliserons une fonction sigmoïde.

- En posant $\tilde{\mathbf{w}} = (w_0, w_1, \cdots, w_D)$ et $\tilde{x} = (x_0, x_1, \cdots, x_D)$
- La probabilité à posteriori de la classe C_1 est donnée par:

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

Perceptron et classification binaire

- L'erreur associée à un exemple d'entrainement $\left(x^{(i)}, y^{(i)}\right)$ est:

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

- Par la **montée du gradient**, la mise à jour **en ligne** des $D + 1$ éléments de $\tilde{\mathbf{w}}$ sera comme suit:

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

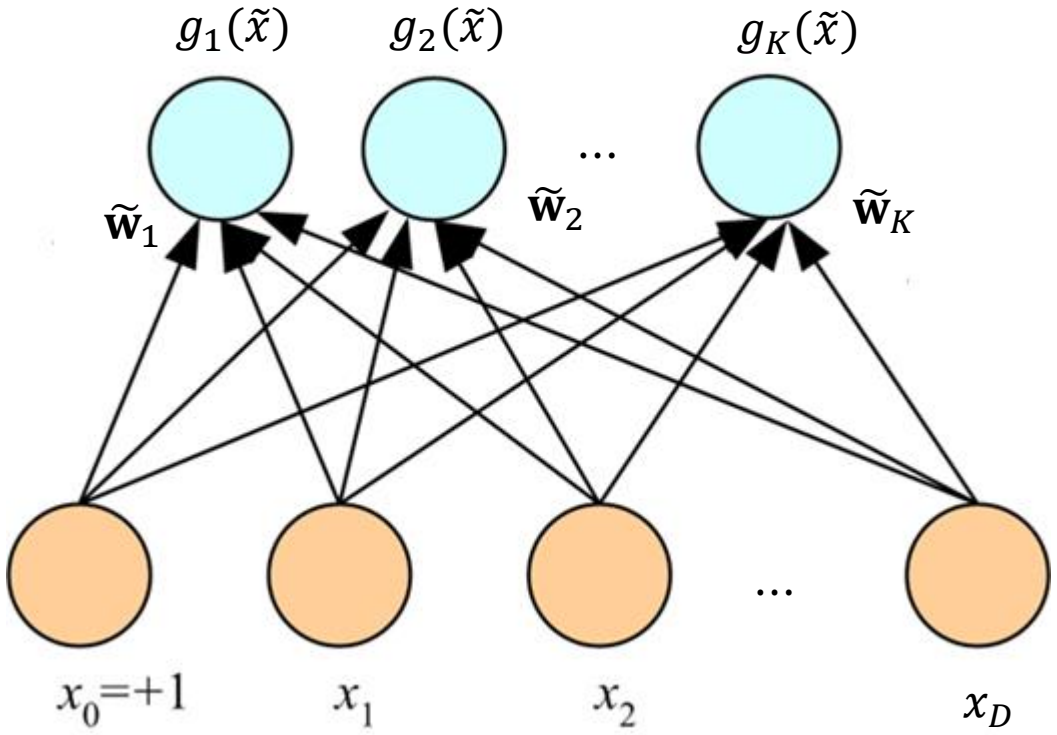
α est **le facteur d'entrainement**.

Perceptron et multiple classes

- Pour un nombre de classes $K > 2$, il faudra définir K perceptrons, chacun avec un vecteur $\tilde{\mathbf{w}}_k$, $k \in \{1, \ldots, K\}$. La probabilité à posteriori de chaque classe est définie par la fonction suivante (fonction Softmax):

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

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



Perceptron et multiple classes

- Soit les vecteurs $\{\widetilde{\mathbf{w}}_1, \widetilde{\mathbf{w}}_2, \ldots, \widetilde{\mathbf{w}}_K\}$.

- Pour K classes $C_1, \ldots, C_K$, l'étiquette du i-ième exemple sera définie par $y^{(i)} = \left({y_1}^{(i)}, {y_2}^{(i)}, \ldots, {y_K}^{(i)}\right)$, où ${y_k}^{(i)} = 1$ si la donné est de la classe C_k.

- L'erreur associée à un exemple d'entraiment $\left(x^{(i)}, y^{(i)}\right)$ est:

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

- Par la montée du gradient, la mise à jour en ligne des $D + 1$ éléments du vecteur $\widetilde{\mathbf{w}}_k$ sera comme suit:

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

Perceptron et classification

- De manière générale:

$$\boldsymbol{mise}\ \text{à}\ \boldsymbol{jour} = facteur \times (\boldsymbol{sortie\ désirée} - \boldsymbol{sortie\ calculée}) \times \boldsymbol{donnée}$$

- 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

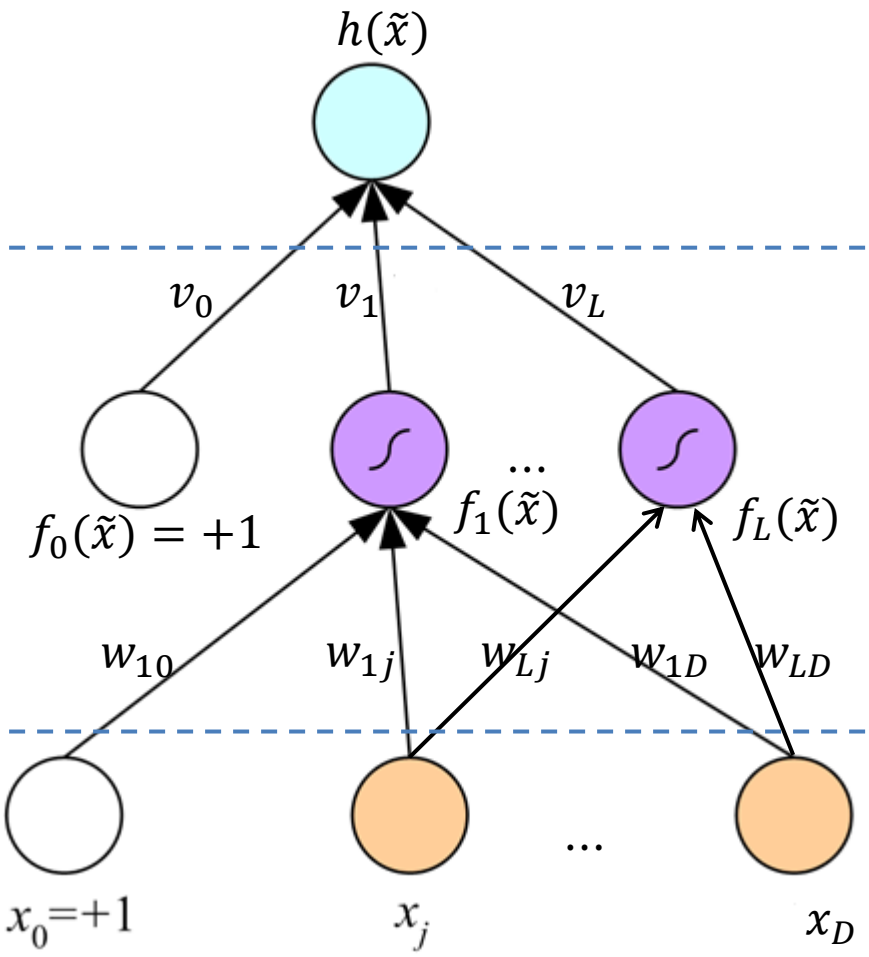
Les réseaux de neurones (RN)

PERCEPTRON MULTICOUCHES (PM)

Perceptron multicouches (PM)

- Le perceptron uni-couche peut approximer uniquement les fonctions linéaires.
- Cependant, il est incapable d'approximer avec des fonctions non-linéaires.
- En utilisant des couches intermédiaires (cachées) entre les entrées et les sorties, on peut faire la régression et la classification non-linéaires.
- En utilisant 2 couches, par exemple, les sorties de la première couche seront les entrées de la seconde couche.

PM pour la régression non linéaire



Couche de Sorties

Couche cachée

Couche d'entrées

PM pour la régression non linéaire

- Pour chaque fonction $f_l(\tilde{x})$, $l \in \{1, \ldots, L\}$, on aura:

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

- Les sorties $h(\tilde{x})$ 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 $f_0(\tilde{x}) = 1$.

PM pour la régression non linéaire

- On peut remarquer que la première couche (cachée) opère une transformation non-linéaire aux variables d'entrées, d'un espace de dimension D à un espace de dimension L.

- La seconde couche implémente une fonction linéaire à partir des sorties de la première couche.

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

Les réseaux de neurones (RN)

ALGORITHME DE PROPAGATION EN ARRIÈRE

Algorithme de propagation en arrière

- L'entrainement d'un perceptron multicouches est similaire à celui d'un perceptron simple. La différence réside seulement dans les fonctions non-linéaires de la couche cachée.

- Les poids v_l peuvent être mis à jour de la même manière que dans le perceptron simple en ayant les entrées $f_l(x)$. En définit l'erreur pour l'exemple $(x^{(i)}, y^{(i)})$ par:

$$E^{(i)} = \frac{1}{2}\left(y^{(i)} - h^{(i)}\right)^2 \qquad \text{où: } h^{(i)} = h\left(\tilde{x}^{(i)}\right)$$

- Un poids v_l va être mis à jour comme suit:

$$\Delta v_l = \alpha\left(y^{(i)} - h^{(i)}\right) f_l{}^{(i)}; \;\; l = 0, .., L. \qquad \text{où: } f_l{}^{(i)} = f_l\left(\tilde{x}^{(i)}\right)$$

α est le facteur d'entrainement.

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:

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

Le rôle de la règle de chaine est de simplifier les calculs des dérivées.

- 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

- Lorsque nous avons deux classes C_1 et C_2, la variable de sortie va être une fonction sigmoïde, comme suit:

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

- 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:

$$\Delta v_l = \alpha\left(y^{(i)} - g^{(i)}\right) f_l{}^{(i)}$$

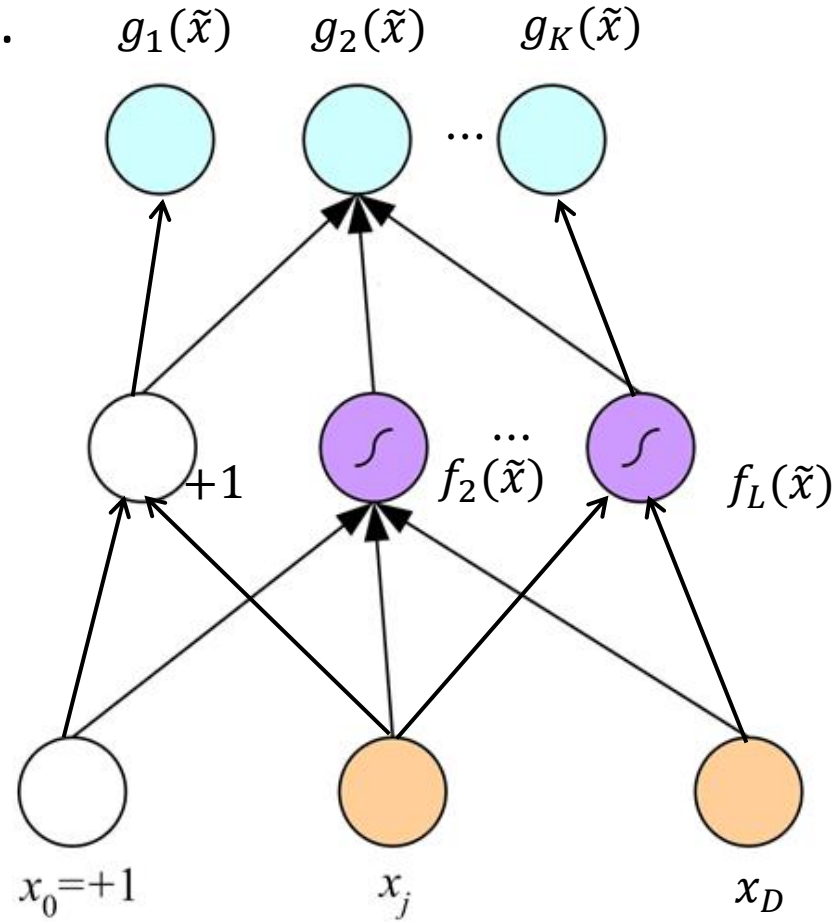
$$\Delta w_{ld} = \alpha \left(y^{(i)} - g^{(i)}\right) \; v_l \; f_l{}^{(i)}(1 - f_l{}^{(i)})\tilde{x}_d{}^{(i)}$$

- À noter que même 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

- Lorsque nous avons plusieurs classes $\{C_1, \dots, C_K\}$ où $K > 2$, on aura K sorties. 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

- L'étiquette du i-ième exemple sera définie par $y^{(i)} = \left({y_1}^{(i)}, {y_2}^{(i)}, \dots, {y_K}^{(i)}\right)$, où ${y_k}^{(i)} = 1$ si l'exemple est de classe C_k.

- On aura aussi L vecteurs $\tilde{\mathbf{w}}_l$ et K vecteurs $\tilde{\mathbf{v}}_k$ à apprendre où : $l \in \{1, \dots, L\}$ et $k \in \{1, \dots, K\}$.

- 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

Références

1. M. S. Allili. Techniques d'apprentissage automatique (Cours de 2e cycle). Université du Québec en Outaouais (UQO), Québec, Canada. Hivers 2015.
2. S. Rogers et M Girolami. A first Course in machine learning, CRC press, 2012.
3. C. Bishop. Pattern Recognition and Machine learning. Springer 2006.
4. R. Duda, P. Storck et D. Hart. Pattern Classification. Prentice Hall, 2002.