Le perceptron multicouche et son algorithme de rétropropagation des erreurs

Marc Parizeau
Département de génie électrique et de génie informatique
Université Laval

10 septembre 2004

Le perceptron multicouche est un réseau orienté de neurones artificiels organisé en couches et où l'information voyage dans un seul sens, de la couche d'entrée vers la couche de sortie. La figure 1 donne l'exemple d'un réseau contenant une couche d'entrée, deux couches cachées et une couche de sortie. La couche d'entrée représente toujours une couche virtuelle associée aux entrées du système. Elle ne contient aucun neurone. Les couches suivantes sont des couches de neurones. Dans l'exemple illustré, il y a 3 entrées, 4 neurones sur la première couche cachée, trois neurones sur la deuxième et quatre neurones sur la couche de sortie. Les sorties des neurones de la dernière couche correspondent toujours aux sorties du système. Dans le cas général, un perceptron multicouche peut posséder un nombre de couches quelconque et un nombre de neurones (ou d'entrées) par couche également quelconque.

Les neurones sont reliés entre eux par des connexions pondérées. Ce sont les poids de ces connexions qui gouvernent le fonctionnement du réseau et "programment" une application de l'espace des entrées vers l'espace des sorties à l'aide d'une transformation non linéaire. La création d'un perceptron multicouche pour résoudre un problème donné passe donc par l'inférence de la meilleure application possible telle que définie par un ensemble de données d'apprentissage constituées de paires de vecteurs d'entrées et de sorties désirées. Cette inférence peut se faire, entre autre, par l'algorithme dit de rétropropagation.

1 Algorithme de rétropropagation

Soit le couple $(\vec{x}(n), \vec{d}(n))$ désignant la n^e donnée d'entraînement du réseau où :

$$\vec{x}(n) = < x_1(n), \ldots, x_p(n) > \text{ et } \vec{d}(n) = < d_1(n), \ldots, d_q(n) > \quad (1)$$

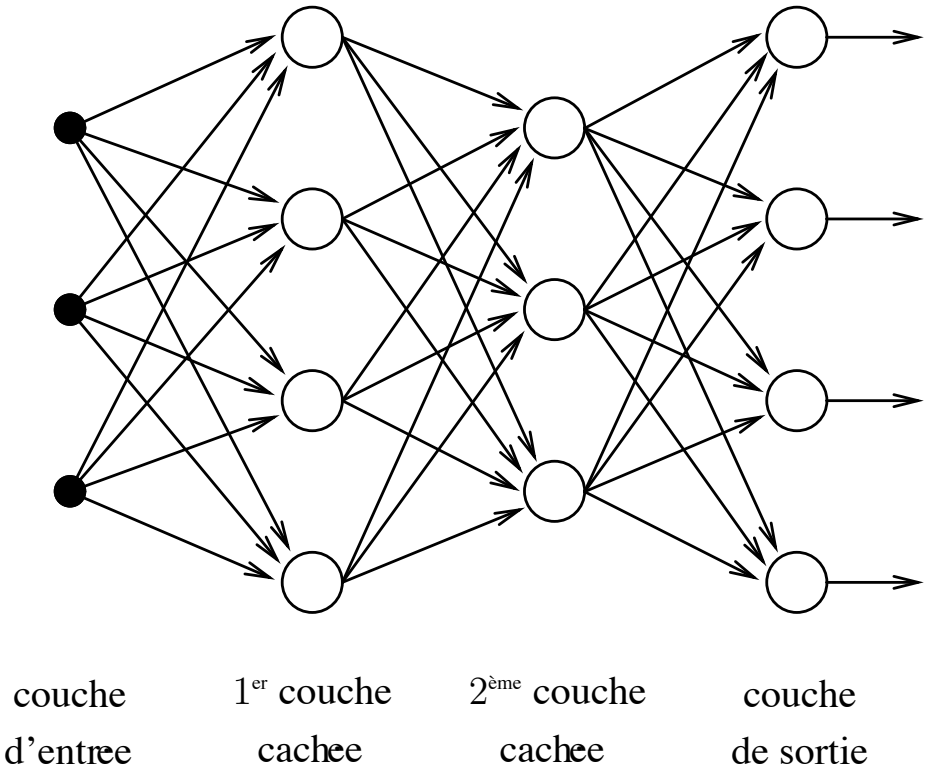

FIG. 1 – *Exemple d'un réseau de type perceptron multicouche.*

correspondent respectivement aux p entrées et aux q sorties désirées du système. L'algorithme de rétropropagation consiste alors à mesurer l'erreur entre les sorties désirées $\vec{d}(n)$ et les sorties observées $\vec{y}(n)$:

$$\vec{y}(n) = < y_1(n), \ldots, y_q(n) > \tag{2}$$

résultant de la propagation vers l'avant des entrées $\vec{x}(n)$, et à rétropropager cette erreur à travers les couches du réseau en allant des sorties vers les entrées.

1.1 Cas de la couche de sortie

L'algorithme de rétropropagation procède à l'adaptation des poids neurone par neurone en commençant par la couche de sortie. Soit l'erreur observée $e_j(n)$ pour le neurone de sortie j et la donnée d'entraînement n :

$$e_j(n) = d_j(n) - y_j(n) \tag{3}$$

où $d_j(n)$ correspond à la sortie désirée du neurone j et $y_j(n)$ à sa sortie observée.

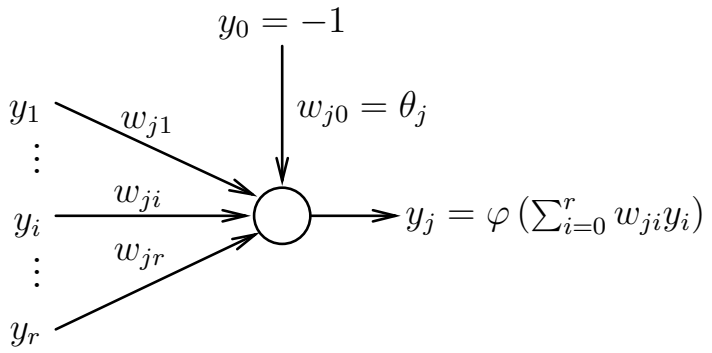

FIG. 2 – *Modèle du neurone j.*

– La variable n représentera toujours la donnée d'entraînement c'est-à-dire le couple contenant un vecteur d'entrées et un vecteur de sorties désirées.
– L'objectif de l'algorithme est d'adapter les poids des connexions du réseau de manière à minimiser la somme des erreurs sur tous les neurones de sortie.
– L'indice j représentera toujours le neurone pour lequel on veut adapter les poids.

Soit $E(n)$ la somme des erreurs quadratiques observées sur l'ensemble C des neurones de sortie :

$$E(n) = \frac{1}{2} \sum_{j \in C} e_j^2(n) \tag{4}$$

La sortie $y_j(n)$ du neurone j est définie par :

$$y_j(n) = \varphi[v_j(n)] = \varphi\left[\sum_{i=0}^{r} w_{ji}(n) y_i(n)\right] \tag{5}$$

où $\varphi[.]$ est la fonction d'activation du neurone, $v_j(n)$ est la somme pondérée des entrées du neurone j, $w_{ji}(n)$ est le poids de la connexion entre le neurone i de la couche précédente et le neurone j de la couche courante, et $y_i(n)$ est la sortie du neurone i. On suppose ici que la couche précédente contient r neurones numérotés de 1 à r, que le poids $w_{j0}(n)$ correspond au biais du neurone j et que l'entrée $y_0(n) = -1$. La figure 2 illustre l'équation 5.

– L'indice i représentera toujours un neurone sur la couche précédente par rapport au neurone j ; on suppose par ailleurs que cette couche contient r neurones.

Pour corriger l'erreur observée, il s'agit de modifier le poids $w_{ji}(n)$ dans le sens opposé au gradient $\frac{\partial E(n)}{\partial w_{ji}(n)}$ de l'erreur (voir figure 3).

– Cette dérivée partielle représente un facteur de sensibilité : si je change un peu $w_{ji}(n)$, est-ce que ça change beaucoup $E(n)$? Si oui, alors je vais changer beaucoup $w_{ji}(n)$ dans le sens inverse de cette dérivée car cela devrait me rapprocher beaucoup du minimum local. Sinon, je dois changer seulement un peu $w_{ji}(n)$ pour corriger l'erreur car je suis tout près de ce minimum !

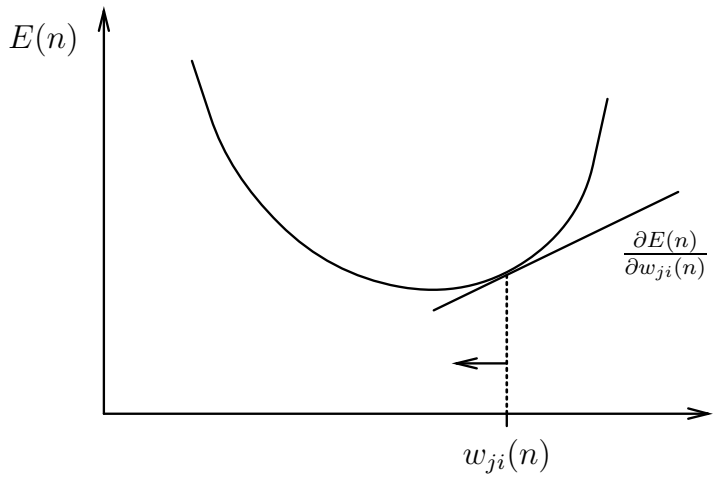

FIG. 3 – *Gradient de l'erreur totale.*

– Puisqu'il y a r neurones sur la couche précédant la couche de sortie, il y a aussi r poids à adapter, et il importe donc de remarquer que la courbe de la figure 3 correspond en fait à une hyper-surface de $r + 1$ dimensions !

Par la règle de chaînage des dérivées partielles, qui nous dit que $\frac{\partial f(y)}{\partial x} = \frac{\partial f(y)}{\partial y} \cdot \frac{\partial y}{\partial x}$, on obtient :

$$\frac{\partial E(n)}{\partial w_{ji}(n)} = \frac{\partial E(n)}{\partial e_j(n)} \cdot \frac{\partial e_j(n)}{\partial y_j(n)} \cdot \frac{\partial y_j(n)}{\partial v_j(n)} \cdot \frac{\partial v_j(n)}{\partial w_{ji}(n)}, \tag{6}$$

et on exprime la variation de poids $\Delta w_{ji}(n)$ sous la forme suivante :

$$\Delta w_{ji}(n) = -\eta \frac{\partial E(n)}{\partial w_{ji}(n)} \tag{7}$$

avec $0 \leq \eta \leq 1$ représentant un *taux d'apprentissage* ou *gain* de l'algorithme.

Évaluons maintenant chacun des termes du gradient.

$$\begin{aligned} \frac{\partial E(n)}{\partial e_j(n)} &= \frac{\partial \left[\frac{1}{2} \sum_{k \in C} e_k^2(n) \right]}{\partial e_j(n)} \\ &= \frac{1}{2} \cdot \frac{\partial e_j^2(n)}{\partial e_j(n)} \\ &= e_j(n) \end{aligned}$$

$$\begin{aligned}\frac{\partial e_j(n)}{\partial y_j(n)} &= \frac{\partial[d_j(n) - y_j(n)]}{\partial y_j(n)} \\ &= -1\end{aligned}$$

$$\begin{aligned}\frac{\partial y_j(n)}{\partial v_j(n)} &= \frac{\partial\left[\frac{1}{1+e^{-v_j(n)}}\right]}{\partial v_j(n)} \\ &= \frac{e^{-v_j(n)}}{\left[1+e^{-v_j(n)}\right]^2} \\ &= y_j(n)\left[\frac{e^{-v_j(n)}}{1+e^{-v_j(n)}}\right] \\ &= y_j(n)\left[\frac{e^{-v_j(n)}+1}{1+e^{-v_j(n)}} - \frac{1}{1+e^{-v_j(n)}}\right] \\ &= y_j(n)[1-y_j(n)]\end{aligned}$$

Et finalement :

$$\begin{aligned}\frac{\partial v_j(n)}{\partial w_{ji}(n)} &= \frac{\partial\left[\sum_{l=0}^{r} w_{jl}(n)y_l(n)\right]}{\partial w_{ji}(n)} \\ &= \frac{\partial[w_{ji}(n)y_i(n)]}{\partial w_{ji}(n)} \\ &= y_i(n)\end{aligned}$$

Nous obtenons donc :

$$\frac{\partial E(n)}{\partial w_{ji}(n)} = -e_j(n)y_j(n)[1-y_j(n)]y_i(n) \tag{8}$$

et la règle dite du “delta” pour la couche de sortie s’exprime par :

$$\Delta w_{ji}(n) = -\eta\frac{\partial E(n)}{\partial w_{ji}(n)} = \eta\delta_j(n)y_i(n) \tag{9}$$

avec :

$$\delta_j(n) = e_j(n)y_j(n)[1-y_j(n)] \tag{10}$$

qui correspond à ce qu’on appelle le “gradient local”.

- Jusqu’ici, nous avons traité seulement le cas de la couche de sortie ! Il reste maintenant à faire l’adaptation des poids sur les couches cachées...
- Mais le problème est qu’on ne dispose plus de l’erreur observée ! ?

1.2 Cas d'une couche cachée

Considérons maintenant le cas des neurones sur la dernière couche cachée (le cas des autres couches cachées est semblable).

- La variable n désignera toujours la donnée d'entraînement c'est-à-dire un couple de vecteurs d'entrées et de sorties désirées.
- L'objectif sera toujours d'adapter les poids de la couche courante en minimisant la somme des erreurs sur les neurones de la couche de sortie.
- Les indices i et j désigneront respectivement (comme précédemment) un neurone sur la couche précédente et un neurone sur la couche courante.
- L'indice k servira maintenant à désigner un neurone sur la couche suivante.

Reprenons l'expression de la dérivée partielle de l'erreur totale $E(n)$ par rapport à w_{ji} mais en ne dérivant plus par rapport à l'erreur $e_j(n)$ car celle-ci est maintenant inconnue :

$$\frac{\partial E(n)}{\partial w_{ji}(n)} = \frac{\partial E(n)}{\partial y_j(n)} \cdot \frac{\partial y_j(n)}{\partial v_j(n)} \cdot \frac{\partial v_j(n)}{\partial w_{ji}(n)} \tag{11}$$

Par rapport aux résultats obtenus pour la couche de sortie, les deux derniers termes de cette équation restent inchangés, seul le premier terme requiert d'être évalué :

$$\frac{\partial E(n)}{\partial y_j(n)} = \frac{\partial \left[\frac{1}{2} \sum_{k \in C} e_k^2(n) \right]}{\partial y_j(n)} \tag{12}$$

Notre problème ici, contrairement au cas des neurones de la couche de sortie, est que tous les $e_k(n)$ dans la somme ci-dessus dépendent de $y_j(n)$. On ne peut donc pas se débarrasser de cette somme ! Néanmoins, nous pouvons écrire :

$$\begin{aligned}
\frac{\partial E(n)}{\partial y_j(n)} &= \sum_{k \in C} \left[e_k(n) \cdot \frac{\partial e_k(n)}{\partial y_j(n)} \right] \\
&= \sum_{k \in C} \left[e_k(n) \cdot \frac{\partial e_k(n)}{\partial v_k(n)} \cdot \frac{\partial v_k(n)}{\partial y_j(n)} \right] \\
&= \sum_{k \in C} \left[e_k(n) \cdot \frac{\partial \left[d_k(n) - \varphi(v_k(n)) \right]}{\partial v_k(n)} \cdot \frac{\partial \left[\sum_l w_{kl}(n) y_l(n) \right]}{\partial y_j(n)} \right] \\
&= \sum_{k \in C} \left[e_k(n) \cdot (-y_k(n)[1 - y_k(n)]) \cdot w_{kj} \right]
\end{aligned}$$

Et en substituant l'équation 10 on obtient :

$$\frac{\partial E(n)}{\partial y_j(n)} = - \sum_{k \in C} \delta_k(n) w_{kj}(n) \tag{13}$$

En substituant l'équation 13 dans l'équation 11, on obtient :

$$\frac{\partial E(n)}{\partial w_{ji}(n)} = -y_j(n)[1 - y_j(n)] \left[\sum_{k \in C} \delta_k(n) w_{kj}(n) \right] y_i(n) \tag{14}$$

Et :

$$\Delta w_{ji}(n) = -\eta \frac{\partial E(n)}{\partial w_{ji}(n)} = \eta \delta_j(n) y_i(n) \tag{15}$$

Avec :

$$\delta_j(n) = y_j(n)[1 - y_j(n)] \sum_{k \in C} \delta_k(n) w_{kj}(n) \tag{16}$$

On peut démontrer que les équations 15 et 16 sont valides pour toutes les couches cachées. Notez bien, cependant, que dans le cas de la première couche cachée du réseau, puisqu'il n'y a pas de couche précédente de neurones, il faut substituer la variable $y_i(n)$ par l'entrée $x_i(n)$.

1.3 Sommaire de l'algorithme (règle du "delta")

L'algorithme de rétropropagation standard se résume donc à la série d'étapes suivantes :

1. Initialiser tous les poids à de petites valeurs aléatoires dans l'intervalle $[-0.5, 0.5]$;
2. Normaliser les données d'entraînement ;
3. Permuter aléatoirement les données d'entraînement ;
4. Pour chaque donnée d'entraînement n :
   (a) Calculer les sorties observées en propageant les entrées vers l'avant ;
   (b) Ajuster les poids en rétropropageant l'erreur observée :

   $$w_{ji}(n) = w_{ji}(n-1) + \Delta w_{ji}(n) = w_{ji}(n-1) + \eta \delta_j(n) y_i(n) \tag{17}$$

   où le "gradient local" est défini par :

   $$\delta_j(n) = \begin{cases} e_j(n) y_j(n)[1 - y_j(n)] & \text{Si } j \in \text{couche de sortie} \\ y_j(n)[1 - y_j(n)] \sum_k \delta_k(n) w_{kj}(n) & \text{Si } j \in \text{couche cachée} \end{cases} \tag{18}$$

   avec $0 \leq \eta \leq 1$ représentant le taux d'apprentissage et $y_i(n)$ représentant soit la sortie du neurone i sur la couche précédente, si celui-ci existe, soit l'entrée i autrement.
5. Répéter les étape 3 et 4 jusqu'à un nombre maximum d'itérations ou jusqu'à ce que la racine de l'erreur quadratique moyenne (EQM) soit inférieure à un certain seuil.

1.4 Règle du “delta généralisé”

L’équation 17 décrit ce qu’on appelle la règle du “delta” pour l’algorithme de rétropropagation des erreurs. L’équation suivante, nommé règle du “delta généralisé”, décrit une autre variante de l’algorithme :

$$w_{ji}(n) = w_{ji}(n-1) + \eta\delta_j(n)y_i(n) + \alpha\Delta w_{ji}(n-1) \tag{19}$$

où $0 \leq \alpha \leq 1$ est un paramètre nommé *momentum* qui représente une espèce d’inertie dans le changement de poids.