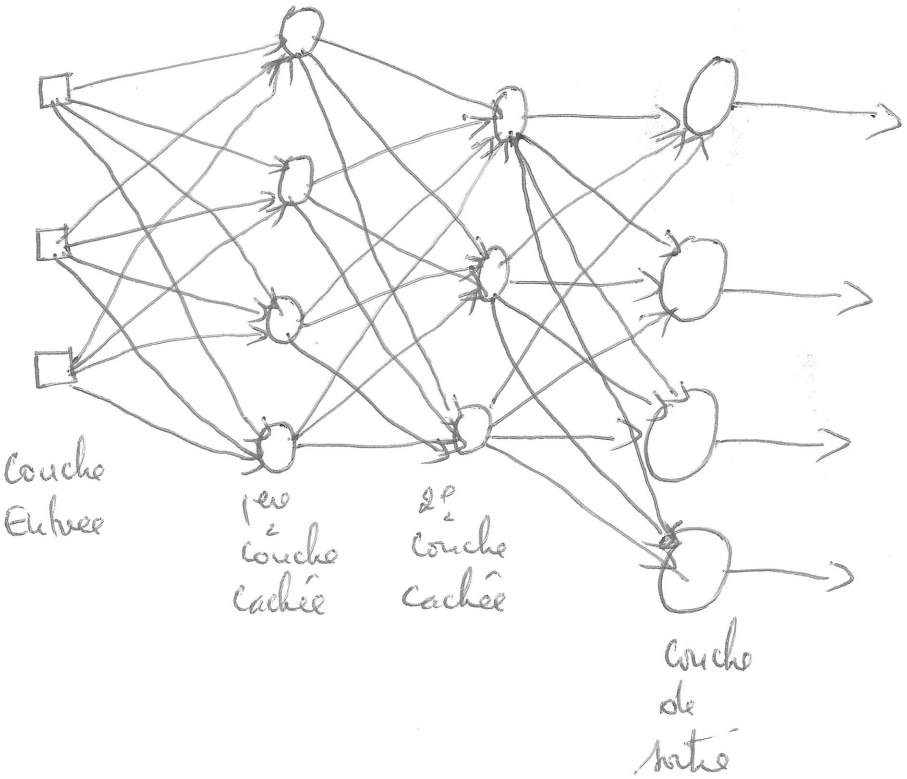
Le perceptron multicouche.
PMC

Introduction

Le PMC est un réseau orienté de neurones artificiels organisés en couches et où l'information circule dans un seul sens, de la couche d'entrée vers la couche de sortie. La figure suivante présente un exemple de PMC



D'une manière générale, un PMC peut posséder un nombre de couches quelconques et un nombre de neurones quelconque par couche.

Les principales caractéristiques d'un PMC sont :

- Au niveau de chaque neurone, y a une fonction dérivable
- On retrouve une ou plusieurs couches cachées en plus de la couche d'entrée et de la couche de sortie
- Un neurone est connecté à tous les neurones de la couche adjacente.

Apprentissage

la méthode d'apprentissage utilisée est l'algorithme de rétropropagation. Il opère en deux phases :

a- la phase forward : dans cette phase, les poids synaptiques sont fixés et le signal d'entrée vers la sortie est propagée couche par couche.

b- la phase backward : dans cette phase, le signal d'erreur en comparant les sorties désirées à celles obtenues est propagé à travers le réseau, couche par couche de la sortie vers l'entrée. les poids sont ensuite modifiés.

Algorithme Backpropagation

Soit le couple $(\vec{x}(n), \vec{d}(n))$ représentant la $n^{ième}$ donnée d'entrainement du réseau où :

$\vec{x}(n) = (x_1(n), \dots, x_p(n))$ et $\vec{d}(n) = (d_1(n), \dots, d_q(n))$ représentent les p entrées et les q sorties desirées.

l'algorithme de rétropropagation consiste à mesurer l'écart entre la sortie desirée et la sortie obtenue.

$$\vec{y}(n) = (y_1(n), \dots, y_q(n))$$

puis propager cette erreur à travers le réseau en allant des sorties vers les entrées.

1- Cas de la couche de sortie :

Soit l'erreur observée e_j pour le neurone de sortie j alors $\boxed{e_j(n) = d_j(n) - y_j(n)}$

où $d_j(n)$ correspond au vecteur de la sortie désirée du neurone j

y_j : correspond à la sortie réelle (obtenue).

la variable n représente la donnée d'entrainement c'est à dire le couple < vecteur d'entrée, vecteur de sortie désirée >

l'objectif de cet algorithme est de modifier les poids des connexions du réseau de telle sorte à minimiser la fonction coût qui correspond dans le cas à la somme des erreurs quadratiques observées sur l'ensemble des neurones de sortie C

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

la sortie du neurone j, $y_j(n)$ est donnée par l'equation suivante :

$$y_j(n) = \sigma\left(\sum_{i=0}^{r} w_{ji}(n)\, y_i(n) \right)$$

σ : fonction d'activation.

w_{ji} : Correspond au poid de la connexion du neurone i de la couche précédente avec le neurone j de la couche courante

Pour corriger l'erreur observée, il faut modifier le poid $w_{ji}(n)$ dans le sens opposé du gradient

$$\frac{\partial E(n)}{\partial w_{ji}(n)}$$ de l'erreur.

Puisqu'il y a r neurones sur la couche précédent la couche de sortie, il y a aussi r poids à modifier.

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

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

La variation du poids est exprimé par :

$$\Delta w_{ji}(n) = -\eta \frac{\partial E(n)}{\partial w_{ji}(n)} \quad (1)$$

avec η : taux d'apprentissage compris entre 0 et

Calculons maintenant les différentes parties

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

$$= \frac{1}{2} \frac{\partial e_j^2(n)}{\partial e_j(n)} = e_j(n)$$

$$\Rightarrow \boxed{\frac{\partial E(n)}{\partial e_j(n)} = e_j(n)}$$

(b) $$\frac{\partial y_j(n)}{\partial v_j(n)} = \frac{\partial \left(\frac{1}{1+e^{-v_j(n)}} \right)}{\partial v_j(n)} = \frac{e^{-v_j(n)}}{\left(1+e^{-v_j(n)}\right)^2}$$

$$= y_j(n)\left(1 - y_j(n)\right)$$

Dans le cas d'une fonction d'activation sous forme Sigmoïde.

(c) $\dfrac{\partial y_j(n)}{\partial w_{ji}(n)} \cdot \dfrac{\partial v_j(n)}{\partial w_{ji}(n)} = \dfrac{\partial \sum_{k=0}^{m} w_{jk}(n)\, y_k(n)}{\partial w_{ji}(n)}$

$$= \frac{\partial w_{ji}(n) y_i(n)}{\partial w_{ji}(n)}$$

$$\boxed{= y_i(n)}$$

Finalement

$$\frac{\partial E(n)}{\partial w_{ji}(n)} = -e_j(n)\, y_j(n)\left(1 - y_j(n)\right) \cdot y_i(n)$$

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

avec $\delta_j(n) = e_j(n) \cdot \left(y_j(n)\right)\left(1 - y_j(n)\right)$.

qui correspond au gradient local.

2- Cas d'une couche cachée

On considère les neurones de la dernière couche cachée, les autres sont traités de la même façon.

- la variable n désignera toujours la donnée d'entrainement. Cad (vecteur- entrée- vecteur sortie donnée)
- l'objectif sera toujours d'adapter les poids de telle sorte à minimiser l'erreur.
- les indices i et j désigneront respectivement Un neurone de la couche précédente et un neurone de la couche courante
- l'indice k servira à designer un neurone sur la couche suivante.

Soit la formule suivante :

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

Par rapport aux neurones, seul le terme

$$\frac{\partial E(n)}{\partial y_j(n)} \text{ change}$$

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

$$\frac{\partial E(n)}{\partial y_j(n)} = \sum_{k \in C} \left[e_k(n) \cdot \frac{\partial e_k(n)}{\partial y_j(n)} \right]$$

$$= \sum_{k \in C} e_k(n) \cdot \left(\frac{\partial e_k(n)}{\partial v_k(n)} \cdot \frac{\partial v_k(n)}{\partial y_j(n)} \right)$$

$$= \sum_{k \in C} e_k(n) \cdot \frac{\partial [d_k(n) - \sigma(v_k(n)]}{\partial v_k(n)} \cdot \frac{\partial \sum w_{kl} \cdot y_l(n)}{\partial y_j(n)}$$

$$= \sum_{k \in C} e_k(n) \cdot \left(- y_k(n)[1 - y_k(n)] \cdot w_{kl} \right].$$

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

Finalement $\delta_j(n) = y_j(n)\left(1 - y_j(n)\right) \sum_{k \in C} \delta_k(n) w_{kj}(n)$

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

$$= \eta \delta_j y_i(n)$$

Synthèse Algorithme Back-propagation

- Initialiser les poids à de petites valeurs
- Pour chaque données d'entraînement n.
- 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

$$w_{ji}(n) = w_{ji}(n-1) + \Delta w_{ji}(n)$$

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

Avec

$$\delta_j(n) = \begin{cases} e_j(n)\, y_j(n)\,[1 - y_j(n)] & \text{si } j \in \text{Couche de sortie} \\ y_j(n)(1 - y_j(n)) \sum_k \delta_k(n) w_{kj}(n) & \text{si } j \in \text{Couche cachée.} \end{cases}$$

- avec $0 \leq \eta \leq 1$. est le taux d'apprentissage
- $y_i(n)$ est la sortie du neurone i de la couche précédente si celle-ci existe sinon c'est l'entrée de i.

- Répéter les étapes 3 et 4 jusqu'au nombre maximum d'itérations ou la racine de l'erreur quadratique est inférieure à un certain seuil.

Une autre variante de l'algorithme de rétropropagation

Dans l'algorithme de rétropropagation, l'equation

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

est appelée la règle de delta. L'équation suivante,

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

avec $0 \leq \alpha \leq 1$ est un paramètre appelé momentum est appelée règle du delta generalisé. Elle décrit une autre variante de l'algorithme.