SY19
Réseaux de neurones: définition et apprentissage

T. Denœux

1 Introduction

Nous avons vu dans le chapitre précédent comment construire des fonctions de prédiction *linéaires* en discrimination et en régression, par minimisation du risque empirique. Cependant, dans certains cas, la relation entre la variable à expliquer et les variables explicatives peut être fortement non linéaire, et une approximation linéaire peut avoir des performances médiocres. Une approche peut alors consister à définir des *fonctions de prédiction linéaires généralisées* de la forme :

$$g(\mathbf{x};\mathbf{w}) = w_0 + \sum_{\ell=1}^{q} w_\ell \phi_\ell(\mathbf{x}), \tag{1}$$

où les ϕ_ℓ sont des fonctions, linéaires ou non, de $\mathbb{R}^p$ dans $\mathbb{R}$. Par exemple, une fonction de prédiction quadratique a la forme suivante :

$$g(\mathbf{x},\mathbf{w}) = w_0 + \sum_{j=1}^{p} w_j x_j + \sum_{j'\geq j} w_{jj'} x_j x_{j'}.$$

Cette approche consiste donc à changer d'espace de représentation, puis à construire un classifieur linéaire dans le nouvel espace.

2 Perceptrons multi-couches

2.1 Neurone formel

Plutôt que de fixer à l'avance les fonctions ϕ_ℓ, une approche particulièrement intéressante consiste à les choisir parmi un ensemble paramétré de fonctions de base. Par exemple, on peut poser :

$$\phi_\ell(\mathbf{x}) = \varphi\left(\sum_{j=0}^{p} v_{\ell j} x_j\right) = \varphi(\mathbf{v}_\ell'\mathbf{x}), \tag{2}$$

où $\mathbf{x} = (1, x_1, \ldots, x_p)'$ est le vecteur d'entrée augmenté, $\mathbf{v}_\ell = (v_{\ell 0}, v_{\ell 1}, \ldots, v_{\ell p})'$ un vecteur de poids, et φ une fonction non linéaire appelée *fonction d'activation*. La fonction φ est dite *unipolaire* si elle est à valeur dans $[0,1]$, comme par exemple la fonction logistique :

$$\varphi(u) = \frac{1}{1+e^{-u}}$$

et elle est dite *bipolaire* si elle est à valeurs dans $[-1, 1]$, comme la fonction tangente hyperbolique :

$$\varphi(u) = \tanh(u) = \frac{e^u - e^{-u}}{e^u + e^{-u}}.$$

Nous supposerons par la suite que φ est unipolaire.

L'équation (2) a été proposée comme modèle de fonction de transfert d'un neurone biologique, les x_j désignant les signaux d'entrée du neurone, $s_\ell = \phi_\ell(\mathbf{x})$ la sortie et $v_{\ell j}$ le poids synaptique associé à la connexion correspondant à l'entrée j. La connexion est excitatrice si $v_{\ell j} > 0$ et inhibitrice si $v_{\ell j} < 0$. La fonction d'activation φ correspond à un effet de seuil : le neurone « déclenche » (émet un influx nerveux) si la somme des entrées dépasse un seuil. Par référence à ce modèle biologique, la fonction (2) est quelquefois appelée *neurone formel*.

2.2 Réseau de neurones multi-couches

En appliquant le principe des fonctions discriminantes linéaires généralisées (1) avec les neurones formels (2), on obtient la fonction de prédiction suivante :

$$g(\mathbf{x}; \mathbf{v}, \mathbf{w}) = \sum_{\ell=0}^{q} w_\ell s_\ell = \mathbf{w}'\mathbf{s}$$

avec $\mathbf{w} = (w_0, \ldots, w_q)'$ et $\mathbf{s} = (1, \phi_1(\mathbf{x}), \ldots, \phi_q(\mathbf{x}))'$. Une forme plus générale consiste à appliquer à la sortie une fonction d'activation ψ, ce qui donne :

$$d = \psi(\mathbf{w}'\mathbf{s}) = \psi\left(w_0 + \sum_{\ell=1}^{q} w_\ell \varphi\left(\sum_{j=0}^{p} v_{\ell j} x_j\right)\right), \tag{3}$$

où d est la sortie du réseau. Typiquement, on choisit $\psi = \varphi$ en discrimination (sorties entre 0 et 1), et ψ égale à la fonction identité en régression.

L'équation (3) définit une famille de fonctions indicées par les paramètres $v_{\ell j}$ et w_ℓ, appelée *perceptron multi-couches* (PMC). Elle peut être vue comme la fonction de transfert d'un réseau de neurones à trois couches : une *couche d'entrée* de $p+1$ neurones, une *couche cachée* de $q+1$ neurones, et une *couche de sortie* composée d'un seul neurone. Chaque neurone caché (excepté celui d'indice 0) et le neurone de sortie sont complètement connectés à tous les neurones de la couche précédente.

En discrimination, on associe à un PMC une fonction de décision δ définie par

$$\delta(\mathbf{x}) = \begin{cases} 1 & \text{si } s \geq 0 \\ 0 & \text{sinon.} \end{cases}$$

Ce modèle se généralise facilement au cas de c classes, $c \geq 2$: dans ce cas, il suffit de définir c neurones de sortie. La sortie du neurone k est alors

$$d_k = \varphi\left(w_{k0} + \sum_{l=1}^{q} w_{k\ell} s_\ell\right) = \varphi(\mathbf{w}_k'\mathbf{s}),$$

$\mathbf{w}_k$ étant le vecteur des poids associés aux connexions entrantes du neurone de sortie k. La règle de décision correspondante est alors

$$\delta(\mathbf{x}) = k \text{ si } d_k \geq d_\ell, \quad \forall \ell.$$

On peut également généraliser l'architecture du PMC en ajoutant autant de couches cachées que l'on veut. Cependant, on se limite souvent à une seule couche cachée, ce que nous suposerons par la suite.

3 Apprentissage

Etant donné un PMC, le problème d'apprentissage consiste à déterminer les poids $v_{\ell j}$ et $w_{l\ell}$ de manière à optimiser les performances du classifieur.

3.1 Critère d'erreur

Soit $\mathcal{L} = \{(\mathbf{x}_1, \mathbf{y}_1), \ldots, (\mathbf{x}_n, \mathbf{y}_n)\}$ un ensemble d'apprentissage composé de n exemples $(\mathbf{x}_i, \mathbf{y}_i)$, $\mathbf{x}_i$ étant le vecteur d'entrée pour l'exemple i et $\mathbf{y}_i$ le vecteur des sorties désirées pour l'exemple i. Dans le case de la discrimination en c classes, on choisit souvent le codage suivant :

$$y_{ik} = \begin{cases} 1 & \text{si l'example } i \text{ appartient à la classe } k \\ 0 & \text{sinon.} \end{cases}$$

Dans le cas $c = 2$, il suffit de conserver la première composante de $\mathbf{y}_i$, puisque $y_{i2} = 1 - y_{i1}$. Nous supposerons par la suite, pour simplifier les notations, l'existence d'une seule sortie.

Soit la fonction de risque empirique suivante (appelée fonction d'erreur quadratique) :

$$\widehat{R} = \frac{1}{n} \sum_{i=1}^{n} \widehat{R}_i$$

avec

$$\widehat{R}_i = (d_i - y_i)^2$$

d_i étant la sortie du réseau pour l'exemple i.

Nous avons vu que minimiser ce critère revient asymptotiquement à approcher la fonction de régression $g^*(\mathbf{x}) = \mathbb{E}(Y|\mathbf{x})$. Par ailleurs, en discrimination, $\widehat{R}$ peut être vu comme une approximation du taux d'erreur empirique (ou taux d'erreur apparent) : en effet, si les sorties d_i sont proches de 0 ou de 1, $\widehat{R}_i$ est proche de 0 lorsque l'exemple i est bien classé, et proche de 1 sinon. L'avantage du critère $\widehat{R}$ par rapport au taux d'erreur est le fait qu'il soit dérivable par rapport aux poids, ce qui permet d'utiliser un algorithme d'optimisation locale, comme nous allons le voir dans la section suivante.

3.2 Calcul du gradient

L'algorithme de *rétropropagation du gradient* permet le calcul des dérivées de $\widehat{R}$ par rapport aux poids, en remontant de la couche de sortie vers la couche d'entrée.

Calculons tout d'abord les dérivées par rapport aux poids de la couche de sortie. Comme

$$\frac{\partial \widehat{R}}{\partial w_\ell} = \frac{1}{n} \sum_{i=1}^{n} \frac{\partial \widehat{R}_i}{\partial w_\ell},$$

il suffit de calculer les dérivées de $\widehat{R}_i$. On a :

$$\frac{\partial \widehat{R}_i}{\partial w_\ell} = \frac{\partial \widehat{R}_i}{\partial d_i}\frac{\partial d_i}{\partial w_\ell} = 2(d_i - y_i)\psi'(\mathbf{w}'\mathbf{s}_i)s_{i\ell}.$$

Si ψ est la fonction logistique, $\psi'(u) = \psi(u)(1-\psi(u))$. On a alors

$$\frac{\partial \widehat{R}_i}{\partial w_\ell} = 2(d_i - y_i)d_i(1-d_i)s_{i\ell} = \Delta_i s_{i\ell},$$

avec $\Delta_i = 2(d_i - y_i)d_i(1-d_i)$.

Pour calculer la dérivée de $\widehat{R}_i$ par rapport à $v_{\ell j}$, on écrit :

$$\frac{\partial \widehat{R}_i}{\partial v_{\ell j}} = \frac{\partial \widehat{R}_i}{\partial s_{i\ell}}\frac{\partial s_{i\ell}}{\partial v_{\ell j}}.$$

Comme

$$\frac{\partial \widehat{R}_i}{\partial s_{i\ell}} = \frac{\partial \widehat{R}_i}{\partial d_i}\frac{\partial d_i}{\partial d_{i\ell}} = 2(d_i - y_i)d_i(1-d_i)w_\ell$$

et

$$\frac{\partial s_{i\ell}}{\partial v_{\ell j}} = \varphi'(\mathbf{v}'_\ell \mathbf{x}_i)x_{ij} = s_{i\ell}(1-s_{i\ell})x_{ij},$$

on a donc :

$$\frac{\partial \widehat{R}_i}{\partial v_{\ell j}} = \Delta_i w_\ell s_{i\ell}(1-s_{i\ell})x_{ij} = \Delta_{i\ell}x_{ij},$$

avec

$$\Delta_{i\ell} = s_{i\ell}(1-s_{i\ell})\Delta_i w_\ell.$$

L'algorithme de rétropropagation du gradient peut donc être récapitulé de la manière suivante :

for $\ell = 0 : q$ **do**
  $\frac{\partial \widehat{R}}{\partial \mathbf{w}_\ell} \leftarrow 0$
  for $j = 0 : p$ **do**
    $\frac{\partial \widehat{R}}{\partial v_{\ell j}} \leftarrow 0$
  end for
end for
for $i = 1 : n$ **do**
  $\Delta_i \leftarrow 2(d_i - y_i)d_i(1-d_i)$
  for $\ell = 0 : q$ **do**
    $\frac{\partial \widehat{R}_i}{\partial w_\ell} \leftarrow \Delta_i s_{i\ell}$
    $\frac{\partial \widehat{R}}{\partial w_\ell} \leftarrow \frac{\partial \widehat{R}}{\partial w_\ell} + \frac{\partial \widehat{R}_i}{\partial w_\ell}$
  end for
  for $\ell = 1 : q$ **do**
    $\Delta_{i\ell} \leftarrow s_{i\ell}(1-s_{i\ell})\Delta_i w_\ell$
    for $j = 0 : p$ **do**
      $\frac{\partial \widehat{R}_i}{\partial v_{\ell j}} \leftarrow \Delta_{i\ell}x_{ij}$
      $\frac{\partial \widehat{R}}{\partial v_{\ell j}} \leftarrow \frac{\partial \widehat{R}}{\partial v_{\ell j}} + \frac{\partial \widehat{R}_i}{\partial v_{\ell j}}$
    end for

end for
end for

La complexité du calcul du gradient de $\widehat{R}$ est la même que celle du calcul de $\widehat{R}$, soit $\mathcal{O}(nN)$, N étant le nombre de poids du réseau.

3.3 Minimisation de l'erreur

La minimisation de $\widehat{R}$ par rapport aux poids est un problème d'optimisation non linéaire sans contrainte. L'algorithme le plus simple pour résoudre un tel problème est l'algorithme de descente de gradient à pas constant étudié dans le chapitre précédent. Un inconvénient de cette méthode est la sensibilité au choix du pas d'apprentissage η : s'il est trop petit, la convergence est très lente et, s'il est trop grand, l'algorithme peut diverger.

La méthode de Newton peut être utilisée, mais elle nécessite le calcul des dérivées secondes de l'erreur, ce qui induit un temps de calcul important si la dimension de $\mathbf{w}$ (le vecteur des poids du réseau) est grande. Il est possible d'approcher la matrice hessienne par une matrice diagonale (méthode « quasi-Newton »). Une approche plus simple consiste à modifier η au cours de l'apprentissage afin de s'adapter localement à la surface d'erreur. Nous présentons ci-dessous deux méthodes.

Méthode 1 : On augmente η quand l'erreur a diminué à l'itération précédente, sinon on diminue η :

if $\widehat{R}(k) < \widehat{R}(k-1)$ **then**
  $\eta(k) \leftarrow a\eta(k-1)$ avec $a > 1$
  $\mathbf{w}(k+1) \leftarrow \mathbf{w}(k) - \eta(k)\frac{\partial \widehat{R}}{\partial \mathbf{w}}(k)$
else
  $\eta(k) \leftarrow b\eta(k-1)$ avec $b < 1$
  $\mathbf{w}(k+1) \leftarrow \mathbf{w}(k-1) - \eta(k)\frac{\partial \widehat{R}}{\partial \mathbf{w}}(k-1)$
end if

Si l'erreur a augmenté entre les itérations $k-1$ et k, on repart donc de la meilleure solution trouvée (c'est-à-dire $\mathbf{w}(k-1)$).

Méthode 2 : On associe un pas η_j à chaque poids w_j, et on adapte chaque η_j individuellement en considérant le signe de la dérivée de $\widehat{R}$ par rapport à w_j entre les itérations $k-1$ et k :

if $\widehat{R}(k) < \widehat{R}(k-1)$ **then**
  for $j = 1 : N$ **do**
    if $\frac{\partial \widehat{R}}{\partial w_j}(k-1) \cdot \frac{\partial \widehat{R}}{\partial w_j}(k) > 0$ **then**
      $\eta_j(k) \leftarrow a\eta_j(k-1)$ avec $a > 1$
    else
      $\eta_j(k) \leftarrow b\eta_j(k-1)$ avec $b < 1$
    end if
    $w_j(k+1) \leftarrow w_j(k) - \eta_j(k)\frac{\partial \widehat{R}}{\partial w_j}(k)$
  end for
else

for $j = 1 : N$ **do**
$\eta_j(k) \leftarrow c\eta_j(k-1)$ avec $c < 1$
$w_j(k+1) \leftarrow w_j(k-1) - \eta_j(k)\frac{\partial \widehat{R}}{\partial w_j}(k-1)$
end for
end if

On prend typiquement $a = 1.2$, $b = 0.8$ et $c = 0.5$.