MLP

PROJET MATH - DEEP 2023

Nous allons programmer avec numpy un petit perceptron multicouche dont le but est de faire de la classification à partir d'un jeu de données très simple, voir ci-dessous un exemple de tel jeu de données.

On dispose de dix points dans le carré $[0,1] \times [0,1]$. Ce sont les données qui vont servir pour l'apprentissage. Elles sont classées en deux classes : les bons sont les ronds verts, les méchants les triangles bleus (c'est comme ça la vie, il y a les bons et les méchants...). Le type de réseau que l'on considère ici, le perceptron multicouche, nécessite d'avoir de telles données annotées.

Le but du réseau est de nous fournir, à partir de cet ensemble de données annotées, une frontière de décision entre le bien et le mal... Une fois connue cette frontière, nous pourrons classer les nouvelles données apparaissant en bonnes ou méchantes... Voici la frontière que l'on obtient sur cet exemple (Fig. 1).

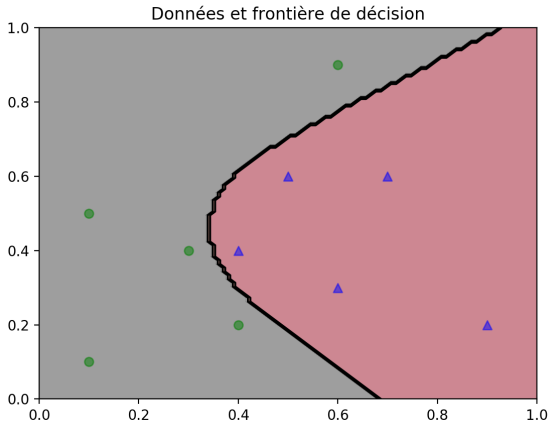

FIGURE 1. Les données et la frontière de décision

La fonction *Loss* à minimiser est définie comme suit. Notons x^i, $i = 1, 2..., 10$ les données. On pose

$$y(x^i) = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \tag{1}$$

si x^i est bon et

$$y(x^i) = \begin{pmatrix} 0 \\ 1 \end{pmatrix} \tag{2}$$

si x^i est méchant. On considère alors le *Loss* suivant

$$Loss\,(\lambda) = \frac{1}{10}\sum_{i=1}^{10} \frac{1}{2}\|y(x^i) - F(x^i)\|_2^2 \tag{3}$$

où λ est un point de l'espace des paramètres servant à construire la fonction F associée au MLP. Après minimisation, la frontière de décision est donnée par $F_1(x) > F_2(x)$ avec $F(x) = (F_1(x), F_2(x))$.

On adopte les notations suivantes pour les calculs. On note $x \in \mathbb{R}^{n_1}$ les données d'entrée (pour nous $n_1 = 2$). On note également $a_j^{(l)}$ la sortie du j-ème neurone du layer l. On a donc

$$a^{(1)} = x \in \mathbb{R}^{n_1} \tag{4}$$

$$a^{(l)} = \sigma\left(W^{(l)}a^{(l-1)} + b^{(l)}\right) \in \mathbb{R}^{n_l} \tag{5}$$

pour $l = 2, 3, ..., L$ (pour nous, $n_L = 2$). On aura aussi besoin de l'entrée du j-ème neurone du layer l (avant activation)

$$z_j^{(l)} = \left(W^{(l)}a^{(l-1)} + b^{(l)}\right)_j \tag{6}$$

Si l'on fait le choix d'une descente de gradient stochastique, notre problème se résume à calculer le gradient des fonctions $Loss_i$ avec

$$Loss_i(\lambda) = \frac{1}{2}\|y(x^i) - a^{(L),i}\|_2^2 \tag{7}$$

où $a^{(L),i}$ désigne $F(x^i)$, vue comme fonctions des poids et des biais (qui interviennent dans la sortie $a^{(L)}$ du réseau). Dans la suite pour simplifier, on note C une telle fonction.

La propagation de l'information dans le réseau est donnée par

$$a^{(l)} = \sigma\left(z^{(l)}\right) \tag{8}$$

pour $l = 2, 3, ..., L$. On a choisi comme fonction d'activation la sigmoïde. On note $\delta^{(l)} \in \mathbb{R}^{n_l}$ le vecteur de composantes

$$\delta_j^{(l)} = \frac{\partial C}{\partial z_j^{(l)}} \tag{9}$$

pour $j = 1, 2, ..., n_l$ et $l = 2, 3, ..., L$. Pour faire les calculs on peut utiliser le produit de Hadamard : si x et y sont deux vecteurs de $\mathbb{R}^n$, le produit de Hadamard $x \circ y$ de x et y est le vecteur de $\mathbb{R}^n$ de composantes $(x \circ y)_i = x_i y_i$, pour tout $i = 1, 2, ..., n$.

Le résultat suivant illustre le mécanisme dit de backpropagation.

Exercice 1. *On a*

$$\delta^{(L)} = \sigma'\left(z^{(L)}\right) \circ \left(a^{(L)} - y\right) \tag{10}$$

$$\delta^{(l)} = \sigma'\left(z^{(l)}\right) \circ \left(W^{(l+1)}\right)^T \delta^{(l+1)} \tag{11}$$

pour $2 \leq l \leq L-1$ *;*

$$\frac{\partial C}{\partial b_j^{(l)}} = \delta_j^{(l)} \tag{12}$$

pour $2 \leq l \leq L$ *;*

$$\frac{\partial C}{\partial w_{jk}^{(l)}} = \delta_j^{(l)} a_k^{(l-1)} \tag{13}$$

pour $2 \leq l \leq L$*, en notant* $w_{jk}^{(l)}$ *les coefficients de la matrice* $W^{(l)}$*.*

On peut encore condenser un peu l'écriture en introduisant la matrice $D^{(l)}$ de taille $n_l \times n_l$ diagonale dont les termes diagonaux sont

$$D_{ii}^{(l)} = \sigma'\left(z_i^{(l)}\right)$$

On a alors

$$\delta^{(L)} = D^{(L)}\left(a^{(L)} - y\right)$$

et

$$\delta^{(l)} = D^{(l)}\left(W^{(l+1)}\right)^T \delta^{(l+1)}$$

Exercice 2. *Montrer que l'on a finalement*

$$\delta^{[l]} = D^{(l)}\left(W^{(l+1)}\right)^T D^{(l+1)}\left(W^{(l+2)}\right)^T \cdots D^{(L-1)}\left(W^{(L)}\right)^T D^{(L)}\left(a^{(L)} - y\right) \tag{14}$$

Il ne faut pas oublier que le calcul de la dérivée de la sigmoïde est trivial. L'equation de l'exercice précédent est l'équation de la backpropagation. Le perceptron muti-couches est l'exemple type d'un réseau forward/backward. La passe forward permet d'évaluer la sortie $a^{(L)}$ du réseau. La passe backward permet de calculer l'erreur et la mise à jour. Ce double mécanisme doit bien apparaitre dans le programme que l'on demande de produire.

Exercice 3. *Programmer en numpy le MLP, avec une descente de gradient stochastique à pas fixe, permettant de traiter le problème de classification. On proposera un programme générique avec* $n_1 = n_L = 2$*. On choisira ensuite, et par exemple,* $L = 4$*,* $n_2 = 2$ *et* $n_3 = 3$*. Le programme montrera l'évolution de la fonction de Loss au cours des epochs.*