RDFIA TP1b : Introduction aux réseaux de neurones
Notes additionnelles

Taylor Mordan, Thomas Robert, Matthieu Cord

26 octobre 2016

Pour les TP sur les réseaux de neurones, vous devrez rendre :
— Un compte rendu à la fin de chaque séance (manuscrit ou par mail à `taylor.mordan@lip6.fr` ou `thomas.robert@lip6.fr` selon votre encadrant de TP, les compte-rendus en retard ne seront pas acceptés). Ce compte-rendu devra décrire le travail effectué pendant la séance et les résultats importants obtenus.
— Un compte rendu global une semaine après la dernière séance (par mail), dont le but est de reprendre au propre le travail effectué pendant l'intégralité des séance de TP, en prenant du recul sur ce qui a été fait. Ce compte rendu représentera la majorité de la note de TP.

Les données et une version numérique du sujet sont accessibles
à l'adresse `http://webia.lip6.fr/~robert`

1 Contexte

Dans ce TP, nous considérons un réseau de neurones à deux couches (soit avec une couche cachée) tel que celui de la Figure 1. Il est composé de :
— Une couche d'entrée x de taille n_x ;
— Une couche cachée h de taille n_h ;
— Une couche de sortie $\hat{y}$ de taille n_y.
Chaque couche $l \in \{h, y\}$ contient une matrice de poids W^l (voir Figure 1 pour les dimensions), un vecteur (colonne) de biais b^l (non représenté sur la Figure 1), ainsi qu'une fonction d'activation non linéaire σ^l (non représentée). Les fonctions choisies sont $\sigma^h = \tanh$ pour la couche cachée, et $\sigma^y = \text{SoftMax}$ pour la couche de sortie.

Pour chaque problème, nous disposons de N exemples d'apprentissage, chacun étant une paire (x, y) constituée d'une donnée d'entrée x et de la classe vérité y associée. C'est à partir de ces données que le réseau de neurones peut être appris par descente de gradient, grâce à l'algorithme de rétropropagation du gradient.

Questions de compréhension

1. Sur le schéma 1, quelles sont les tailles n_x, n_h et n_y ?
2. Comment ces tailles sont-elles choisies en pratique ?
3. À quoi servent les fonctions non linéaires pour les couches cachées ?
4. Quel est l'intérêt particulier de la fonction SoftMax pour la couche de sortie ?
5. Que représentent les vecteurs $\hat{y}$ et y ? Quelle est la différence entre ces deux quantités ?

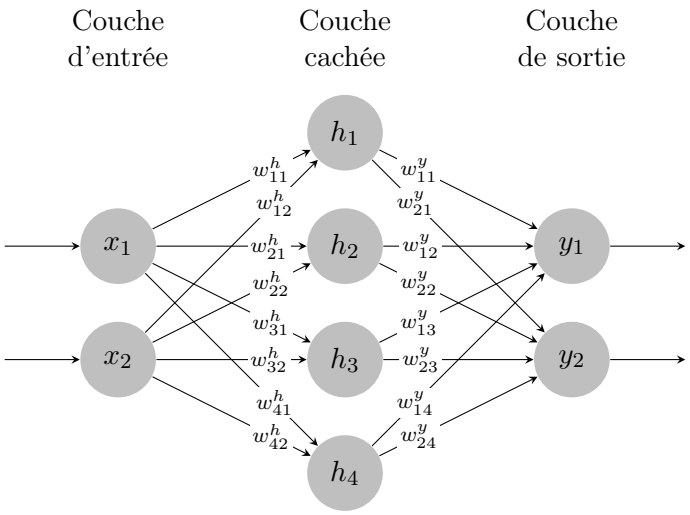

FIGURE 1 – Exemple d'architecture d'un réseau de neurones à une couche cachée.

2 Préparation : un peu de maths

L'objectif de la première séance était de trouver les équations des passes *forward* et *backward* d'un réseau de neurones à deux couches.

Forward Les éléments de la couche cachée s'obtiennent par

$$h_i = \sigma^h\left(\tilde{h}_i\right) = \tanh\left(\sum_j W^h_{ij} x_j + b^h_i\right) \tag{1}$$

et ceux de la couche de sortie par

$$\hat{y}_i = \sigma^y\left(\tilde{y}_i\right) = \text{SoftMax}\left(\sum_j W^y_{ij} h_j + b^y_i\right). \tag{2}$$

Il est possible de réécrire ces équations sous forme vectorielle pour une implémentation plus efficace sous MATLAB. Nous noterons x, $\tilde{h}$, h, $\tilde{y}$ et $\hat{y}$ les vecteurs colonnes correspondants aux éléments des équations précédentes. Les deux équations de la passe *forward* deviennent donc :

$$\begin{cases} h = \sigma^h\left(\tilde{h}\right) = \tanh\left(W^h x + b^h\right) \ ; & (3) \\ \hat{y} = \sigma^y\left(\tilde{y}\right) = \text{SoftMax}\left(W^y h + b^y\right). & (4) \end{cases}$$

Backward L'objectif de la seconde passe est d'obtenir les gradients de l'erreur par rapport à chaque paramètre (poids ou biais) du réseau, cela pour chaque couche successivement en commençant par la fin du réseau. Le signal d'erreur pour la dernière couche est

$$\delta^y_i = \hat{y}_i - y_i \tag{5}$$

et les gradients de cette même couche s'obtiennent avec les formules

$$\begin{cases} \dfrac{\partial \ell}{\partial W^y_{ij}} = \delta^y_i h_j \ ; & (6) \\ \dfrac{\partial \ell}{\partial b^y_i} = \delta^y_i. & (7) \end{cases}$$

Pour la couche précédente, le signal d'erreur est propagé par la formule

$$\delta_i^h = \left(\sum_j \delta_j^y W_{ji}^y \right) \tanh' \left(\tilde{h}_i \right). \tag{8}$$

Les gradients de la couche cachée s'obtiennent comme précédemment par

$$\begin{cases} \dfrac{\partial \ell}{\partial W_{ij}^h} = \delta_i^h x_j \ ; & (9) \\ \dfrac{\partial \ell}{\partial b_i^h} = \delta_i^h. & (10) \end{cases}$$

Encore une fois, il est plus efficace d'implémenter ces équations sous forme vectorielle, avec δ^h et δ^y les vecteurs colonnes correspondants aux signaux d'erreurs et y celui des classes vérités :

$$\begin{cases} \delta^y = \hat{y} - y \ ; & (11) \\ \dfrac{\partial \ell}{\partial W^y} = \delta^y h^T \ ; & (12) \\ \dfrac{\partial \ell}{\partial b^y} = \delta^y \ ; & (13) \\ \delta^h = \left[(W^y)^T \delta^y \right] \odot \tanh' \left(\tilde{h} \right) \ ; & (14) \\ \dfrac{\partial \ell}{\partial W^h} = \delta^h x^T \ ; & (15) \\ \dfrac{\partial \ell}{\partial b^h} = \delta^h. & (16) \end{cases}$$

Descente de gradient Pour chaque vecteur ou matrice de poids du réseau nommé génériquement W et b ci-dessous, on applique une descente de gradient pour un ensemble d'exemples (X, Y) contenant n exemples $(x^{(i)}, y^{(i)}), i = 1..n$, suivant la formule :

$$\begin{cases} b \leftarrow b - \dfrac{\eta}{n} \dfrac{\partial \mathcal{L}(X,Y)}{\partial b} = b - \dfrac{\eta}{n} \displaystyle\sum_{i=1}^{n} \dfrac{\partial \ell(x^{(i)}, y^{(i)})}{\partial b} \ ; & (17) \\ W \leftarrow W - \dfrac{\eta}{n} \dfrac{\partial \mathcal{L}(X,Y)}{\partial W} = W - \dfrac{\eta}{n} \displaystyle\sum_{i=1}^{n} \dfrac{\partial \ell(x^{(i)}, y^{(i)})}{\partial W}. & (18) \end{cases}$$

3 Implémentation

Une fois les équations du réseau de neurones connues, l'objectif est d'implémenter le réseau sous MATLAB. Pour cela nous considérerons une optimisation par descente de gradients stochastique utilisant des mini-batchs. Les vecteurs x, $\tilde{h}$, h, $\tilde{y}$, $\hat{y}$ et y sont donc regroupés par paquets de n ($1 \leq n \leq N$) exemples dans les matrices X, $\tilde{H}$, H, $\tilde{Y}$, $\hat{Y}$ et Y de tailles $n \times n_l$ pour $l \in \{x, h, y\}$. À chaque fois, les matrices contiennent n exemples transposés en vecteurs lignes.

Il y a deux façons d'implémenter le réseau par mini-batchs : en utilisant une boucle sur les n exemples (ce qui est lent sous MATLAB) ou en écrivant les calculs sous forme matricielle (ce qui est conseillé mais qui demande de trouver les nouvelles équations).

Il est demandé d'écrire les fonctions `initMLP`, `forward`, `loss_accuracy` et `backward`, puis de les appliquer à un problème jouet de classification (voir le sujet de la semaine dernière).