Université de Technologie de Compiègne

Génie informatique

Entrainement d'un réseau de neurones et rétropropagation dans le cadre d'un problème de classification.

Authors :

Nacim Khalis et Antoine Collas

31 mars 2018

1 Réseau de neurones : perceptron multicouche

L'objectif est de programmer un perceptron multicouche afin de classer un ensemble de bouteilles de vins en 3 ensembles.

L'architecture de notre perceptron multicouche est la suivante :

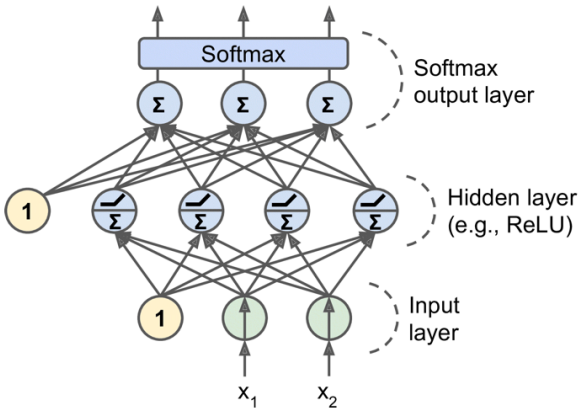

FIGURE 1 – Perceptron multi-couches tiré de 'Hands-On Machine Learning with Scikit-Learn and TensorFlow' de Aurélien Géron.

2 Entrainement d'un réseau de neurones de classification

2.1 Notation

m est le nombre d'éléments dans notre ensemble d'entrainement.

K est le nombre de classes (et donc de sorties de notre réseau de neurones).

n_{in} est le nombre de neurones de la couche d'entrée (sans le biais). Donc n_{in} correspond à la dimension de nos données d'entrée.

n_{hidden} est le nombre de neurones de la couche cachée (sans le biais).

n_{out} est le nombre de neurones de la couche de sortie. Donc n_{out} correspond au nombre de classes de notre problème.

$b_i^{(l)}$ est le biais de la couche l allant vers le ième neurone de la couche $l+1$.

$w_{ij}^{(l)}$ est le poids entre le neurone i de la couche l et le neurone j de la couche $l+1$.

$x_i^{(l)}$ est la sortie du neurone i de la couche l. Donc $x_i^{(2)} = relu(\sum_{k=1}^{n_{in}} w_{ki}^{(1)} x_k^{(1)} + b_i^{(1)})$

$relu(x) = max(0, x)$ est la fonction d'activation de la couche cachée.

$softmax(x)_j = \frac{e^{x_j}}{\sum_{k=1}^{n} e^{x_k}}$ est la fonction de sortie permettant d'obtenir la probabilité pour chaque classe.

$E = -\frac{1}{m}\sum_{i=1}^{m}\sum_{k=1}^{K}(y_k^{(i)} \times log(\hat{p}_k^{(i)}))$ est l'erreur appelée entropie croisée. $y_k^{(i)}$ vaut 1 si le ième élément de l'échantillon appartient à la classe k, 0 sinon. $\hat{p}_k^{(i)}$ est la kème sortie du perceptron pour le ième élément de l'échantillon.

2.2 Calcul des dérivées partielles de la fonction de coût

L'apprentissage d'un réseau de neurones est executé à l'aide d'une descente de gradient sur la fonction de coût. Les paramètres de notre réseau sont les poids $w_{ij}^{(l)}$ et les biais $b_j^{(l)}$. Nous cherchons donc à calculer les dérivées partielles de la fonction de coût par rapport à ces différents paramètres. Nous commençons par montrer comment calculer $\frac{\partial E}{\partial w_{ij}^{(2)}}$ $\forall (i,j) \in \{1,...,n_{hidden}\} \times \{1,...,n_{out}\}$.

Pour un lot de données entré dans le réseau nous avons les équations suivantes en partant de l'erreur :

$$E = -\frac{1}{m}\sum_{p=1}^{m}\sum_{k=1}^{K}(y_k^{(p)} \times log(\hat{p}_k^{(p)})) \tag{1}$$

avec

$$\hat{p}_k^{(p)} = softmax(x^{(p,3)})_k \tag{2}$$

où $x^{(p,3)}$ est la sortie de la dernière couche avant le softmax pour la pème donnée d'entrainement.

$$\frac{\partial E}{\partial w_{ij}^{(2)}} = -\frac{1}{m}\sum_{p=1}^{m}\sum_{k=1}^{n_{out}} y_k^{(p)} \frac{\partial log(\hat{p}_k^{(p)})}{\partial w_{ij}^{(2)}} \tag{3}$$

Règle de la chaine : si $y = f(u)$ et $u = g(x) = (g_1(x),...,g_m(x))$ alors $\frac{\partial y}{\partial x_i} = \sum_{l=1}^{m} \frac{\partial y}{\partial u_l}\frac{\partial u_l}{\partial x_i}$

Donc en appliquant la règle de la chaine :

$$\frac{\partial log(\hat{p}_k^{(p)})}{\partial w_{ij}^{(2)}} = \sum_{q=1}^{n_{out}} \frac{\partial log(\hat{p}_k^{(p)})}{\partial (x^{(p,3)})_q} \frac{\partial (x^{(p,3)})_q}{\partial w_{ij}^{(2)}} \tag{4}$$

or si $q \neq j$

$$\frac{\partial (x^{(p,3)})_q}{\partial w_{ij}^{(2)}} = 0 \tag{5}$$

donc

$$\frac{\partial log(\hat{p}_k^{(p)})}{\partial w_{ij}^{(2)}} = \frac{\partial log(\hat{p}_k^{(p)})}{\partial (x^{(p,3)})_j} \frac{\partial (x^{(p,3)})_j}{\partial w_{ij}^{(2)}} \tag{6}$$

$$log(\hat{p}_k^{(p)}) = \frac{ln(\hat{p}_k^{(p)})}{ln(10)} = \frac{(x^{(p,3)})_k - ln(\sum_{q=1}^{n_{out}} e^{(x^{(p,3)})_q})}{ln(10)} \tag{7}$$

Si $k \neq j$

$$\frac{\partial log(\hat{p}_k^{(p)})}{\partial (x^{(p,3)})_j} = -\frac{1}{ln(10)} \frac{e^{(x^{(p,3)})_j}}{\sum_{q=1}^{n_{out}} e^{(x^{(p,3)})_q}} = -\frac{softmax(x^{(p,3)})_j}{ln(10)} \tag{8}$$

Sinon (dans le cas où $k = j$) :

$$\frac{\partial log(\hat{p}_k^{(p)})}{\partial (x^{(p,3)})_j} = \frac{1}{ln(10)} \left[1 - softmax(x^{(p,3)})_j \right] \tag{9}$$

Donc $\forall k$:

$$\frac{\partial log(\hat{p}_k^{(p)})}{\partial (x^{(p,3)})_j} = \frac{1}{ln(10)} \left[\delta_{kj} - softmax(x^{(p,3)})_j \right] \tag{10}$$

où $\delta_{kj} = 1$ si $k = j$, 0 sinon.

De plus,

$$(x^{(p,3)})_j = \sum_{i=1}^{n_{hidden}} w_{ij}^{(2)} (x^{(p,2)})_i + b_j^{(2)} \tag{11}$$

Donc,

$$\frac{\partial (x^{(p,3)})_q}{\partial w_{ij}^{(2)}} = (x^{(p,2)})_i \tag{12}$$

$$\frac{\partial log(\hat{p}_k^{(p)})}{\partial w_{ij}^{(2)}} = \frac{1}{ln(10)} \left[\delta_{kj} - softmax(x^{(p,3)})_j \right] (x^{(p,2)})_i \tag{13}$$

Ce qui donne finalement :

$$\frac{\partial E}{\partial w_{ij}^{(2)}} = -\frac{1}{m \times ln(10)} \sum_{p=1}^{m} \sum_{k=1}^{n_{out}} y_k^{(p)} \left[\delta_{kj} - softmax(x^{(p,3)})_j \right] (x^{(p,2)})_i \tag{14}$$

Cette formule peut être interprétée de la manière suivante :

— Si la classe à prédire pour la p^{eme} donnée entrée dans le réseau est j alors le poids $w_{ij}^{(2)}$ est augmenté en fonction de la performance de la prédiction et de $(x^{(p,2)})_i$. Si la prédiction est déjà très bonne, $w_{ij}^{(2)}$ n'est est que légèrement augmenté sinon il est formetement augmenté pour augmenter la probabilité sortante du réseau.

— Si la classe à prédire est $k \neq j$ alors il y une diminution du poids en fonction de la probabilité sortante et de $(x^{(p,2)})_i$. Si la probabilité sortante est faible, le poids n'est que légèrement diminué. Si elle est élevée (alors qu'elle devrait être faible), le poids est fortement diminué.

Nous appliquons le même procédé pour calculer $\frac{\partial E}{\partial w_{ij}^{(1)}}$

$$\frac{\partial E}{\partial w_{ij}^{(1)}} = -\frac{1}{m}\sum_{p=1}^{m}\sum_{k=1}^{n_{out}} y_k^{(p)} \frac{\partial log(\hat{p}_k^{(p)})}{\partial w_{ij}^{(1)}} \tag{15}$$

$$\frac{\partial log(\hat{p}_k^{(p)})}{\partial w_{ij}^{(1)}} = \frac{\partial log(\hat{p}_k^{(p)})}{\partial (x^{(p,2)})_j} \frac{\partial (x^{(p,2)})_j}{\partial w_{ij}^{(1)}} \tag{16}$$

$$\frac{\partial log(\hat{p}_k^{(p)})}{\partial (x^{(p,2)})_j} = \sum_{q=1}^{n_{out}} \frac{\partial log(\hat{p}_k^{(p)})}{\partial (x^{(p,3)})_q} \frac{\partial (x^{(p,3)})_q}{\partial (x^{(p,2)})_j} \tag{17}$$

$$\frac{\partial (x^{(p,3)})_k}{\partial (x^{(p,2)})_j} = w_{jk}^{(2)} \tag{18}$$

D'après les équations (10), (17), (18) :

$$\frac{\partial log(\hat{p}_k^{(p)})}{\partial (x^{(p,2)})_j} = \frac{1}{ln(10)} \sum_{q=1}^{n_{out}} \Big[\delta_{kq} - softmax(x^{(p,3)})_q\Big] w_{jk}^{(2)} \tag{19}$$

$$(x^{(p,2)})_j = relu\Big[\sum_{k=1}^{n_{in}} w_{kj}^{(1)} (x^{(p,1)})_k + b_j^{(1)}\Big] \tag{20}$$

Si $\sum_{k=1}^{n_{in}} w_{kj}^{(1)} (x^{(p,1)})_k + b_j^{(1)} > 0$:

$$\frac{\partial (x^{(p,2)})_j}{\partial w_{ij}^{(1)}} = \frac{\partial relu\Big[\sum_{k=1}^{n_{in}} w_{kj}^{(1)} (x^{(p,1)})_k + b_j^{(1)}\Big]}{\partial w_{ij}^{(1)}} = \frac{\partial \Big[\sum_{k=1}^{n_{in}} w_{kj}^{(1)} (x^{(p,1)})_k + b_j^{(1)}\Big]}{\partial w_{ij}^{(1)}} = (x^{(p,1)})_i \tag{21}$$

Si $\sum_{k=1}^{n_{in}} w_{kj}^{(1)} (x^{(p,1)})_k + b_j^{(1)} < 0$:

$$\frac{\partial (x^{(p,2)})_j}{\partial w_{ij}^{(1)}} = \frac{\partial relu\Big[\sum_{k=1}^{n_{in}} w_{kj}^{(1)} (x^{(p,1)})_k + b_j^{(1)}\Big]}{\partial w_{ij}^{(1)}} = 0 \tag{22}$$

Donc :

$$\frac{\partial (x^{(p,2)})_j}{\partial w_{ij}^{(1)}} = H\Big[\sum_{k=1}^{n_{in}} w_{kj}^{(1)} (x^{(p,1)})_k + b_j^{(1)}\Big] (x^{(p,1)})_i \tag{23}$$

où $H(x) = 1$ si $x > 0$, et 0 si $x < 0$.

D'après les équations (16), (19), (23) :

$$\frac{\partial log(\hat{p}_k^{(p)})}{\partial w_{ij}^{(1)}} = \frac{1}{ln(10)} H\Big[\sum_{q=1}^{n_{in}} w_{qj}^{(1)} (x^{(p,1)})_q + b_j^{(1)}\Big] (x^{(p,1)})_i \sum_{q=1}^{n_{out}} \Big[\big[\delta_{kq} - softmax(x^{(p,3)})_q\big] w_{jq}^{(2)}\Big] \tag{24}$$

D'après les équations (15) et (24) :

$$\frac{\partial E}{\partial w_{ij}^{(1)}} = -\frac{1}{m \times ln(10)} \sum_{k=1}^{n_{out}} \sum_{p=1}^{m} \left[y_k^{(p)} \sum_{q=1}^{n_{out}} \left[[\delta_{kq} - softmax(x^{(p,3)})_q] w_{jq}^{(2)} \right] H\big[\sum_{q=1}^{n_{in}} w_{qj}^{(1)} (x^{(p,1)})_q + b_j^{(1)} \big] (x^{(p,1)})_i \right] \tag{25}$$

Interprétation : d'après la formule précédente, le changement de $w_{ij}^{(1)}$ est déterminé de la façon suivante :

- nous réunissons tous les éléments d'entrainement par classe (interprétation de la double somme et de $y_k^{(p)}$)
- pour chaque élément de la classe nous calculons la responsabilité de $w_{ij}^{(1)}$ dans l'erreur finale : si pour l'élément choisi, la somme pondérée est négative alors le poids n'est pas modifié (pas de responsabilité dans l'erreur, représenté par H dans l'équation)
- sinon la modification est proportionnelle à $(x^{(p,1)})_i$ (la variable d'entrée pour le p^{eme} élément d'entrainement)
- la modification est aussi proportionnelle aux erreurs de toutes les sorties (les couches sont totalement connectées ce qui est visible dans $\sum_{q=1}^{n_{out}} \left[[\delta_{kq} - softmax(x^{(p,3)})_q] w_{jq}^{(2)} \right]$)

3 Rétropropagation

3.1 Calcul des gradients à l'aide de la rétropropagation

Nous avons montré dans la section précédente comment calculer les dérivées partielles de la fonction de coût. Cependant les calculs deviennent rapidement assez complexes et peu efficaces en terme de performances. Il va être très compliqué d'ajouter des couches cachées supplémentaires à notre réseau. Nous allons détailler l'algorithme de rétropropagation du gradient qui permet de calculer les gradients successivements en appliquant astucieusement la règle de la chaîne. L'algorithme de rétropropagation consiste à calculer $\frac{\partial E}{\partial w_{ij}^{(1)}}$ en réutilisant une partie des calculs faits pour $\frac{\partial E}{\partial w_{ij}^{(2)}}$.

Soit E^p l'erreur pour la p^{eme} donnée :

$$E^p = -\sum_{k=1}^{K} (y_k^{(p)} \times log(\hat{p}_k^{(p)})) \tag{26}$$

En appliquant la règle de la chaine :

$$\begin{aligned} \frac{\partial E^p}{\partial w_{ij}^{(1)}} &= \frac{\partial E^p}{\partial (x^{(p,2)})_j} \frac{\partial (x^{(p,2)})_j}{\partial (\sum_{k=1}^{n_{in}} w_{kj}^{(1)} (x^{(p,1)})_k + b_j^{(1)})} \frac{\partial (\sum_{k=1}^{n_{in}} w_{kj}^{(1)} (x^{(p,1)})_k + b_j^{(1)})}{\partial w_{ij}^{(1)}} \\ &= \frac{\partial E^p}{\partial (x^{(p,2)})_j} H\big[\sum_{q=1}^{n_{in}} w_{qj}^{(1)} (x^{(p,1)})_q + b_j^{(1)} \big] (x^{(p,1)})_i \end{aligned} \tag{27}$$

Il reste à calculer $\frac{\partial E^p}{\partial (x^{(p,2)})_j}$:

$$\frac{\partial E^p}{\partial (x^{(p,2)})_j} = \sum_{k=1}^{K} \Big(\frac{\partial E^p}{\partial (x^{(p,3)})_k} \frac{\partial (x^{(p,3)})_k}{\partial (x^{(p,2)})_j} \Big) = \sum_{k=1}^{K} \Big(\frac{\partial E^p}{\partial (x^{(p,3)})_k} w_{jk}^{(2)} \Big) \tag{28}$$

D'après les deux équations précédentes :

$$\frac{\partial E^p}{\partial w_{ij}^{(1)}} = \sum_{k=1}^{K} \Big(\frac{\partial E^p}{\partial (x^{(p,3)})_k} w_{jk}^{(2)} \Big) H\big[\sum_{q=1}^{n_{in}} w_{qj}^{(1)} (x^{(p,1)})_q + b_j^{(1)} \big] (x^{(p,1)})_i \tag{29}$$

Donc

$$\frac{\partial E}{\partial w_{ij}^{(1)}} = \frac{1}{m} \sum_{p=1}^{m} \sum_{k=1}^{K} \Big(\frac{\partial E^p}{\partial (x^{(p,3)})_k} w_{jk}^{(2)} \Big) H\big[\sum_{q=1}^{n_{in}} w_{qj}^{(1)} (x^{(p,1)})_q + b_j^{(1)} \big] (x^{(p,1)})_i \tag{30}$$

Si $\frac{\partial E}{\partial w_{ij}^{(2)}}$ a déjà été calculé alors $\frac{\partial E}{\partial w_{ij}^{(1)}}$ est calculable. En effet, les $\frac{\partial E^p}{\partial (x^{(p,3)})_k}$ sont calculés pour $\frac{\partial E}{\partial w_{ij}^{(2)}}$. Les gradients d'une couche l sont sonc calculés en fonction de ce qui a déjà été calculé pour la couche $l+1$.

3.2 Algorithme d'apprentissage d'un réseau de neurones

L'apprentissage du réseau de neurones consiste à calculer les gradients de la fonction erreur par rapport aux poids du réseau à l'aide des équations suivantes (rétroprpagation) puis à faire une descente de gradients.

$$\frac{\partial E^p}{\partial (x^{(p,3)})_j} = -\sum_{k=1}^{n_{out}} y_k^{(p)} \frac{\partial log(\hat{p}_k^{(p)})}{\partial (x^{(p,3)})_j} = -\frac{1}{ln(10)} \sum_{k=1}^{n_{out}} y_k^{(p)} \Big[\delta_{kj} - softmax(x^{(p,3)})_j \Big] \tag{31}$$

$$\frac{\partial E^p}{\partial w_{ij}^{(2)}} = \frac{\partial E^p}{\partial (x^{(p,3)})_j} \frac{\partial (x^{(p,3)})_j}{\partial w_{ij}^{(2)}} = \frac{\partial E^p}{\partial (x^{(p,3)})_j} (x^{(p,2)})_i \tag{32}$$

$$\frac{\partial E}{\partial w_{ij}^{(2)}} = \frac{1}{m} \sum_{p=1}^{m} \frac{\partial E^p}{\partial w_{ij}^{(2)}} \tag{33}$$

$$\frac{\partial E^p}{\partial b_j^{(2)}} = \frac{\partial E^p}{\partial (x^{(p,3)})_j} \frac{\partial (x^{(p,3)})_j}{\partial b_j^{(2)}} = \frac{\partial E^p}{\partial (x^{(p,3)})_j} \tag{34}$$

$$\frac{\partial E}{\partial b_j^{(2)}} = \frac{1}{m} \sum_{p=1}^{m} \frac{\partial E^p}{\partial b_j^{(2)}} \tag{35}$$

$$\frac{\partial E^p}{\partial w_{ij}^{(1)}} = \sum_{k=1}^{K} \Big(\frac{\partial E^p}{\partial (x^{(p,3)})_k} w_{jk}^{(2)} \Big) H\big[\sum_{q=1}^{n_{in}} w_{qj}^{(1)} (x^{(p,1)})_q + b_j^{(1)} \big] (x^{(p,1)})_i \tag{36}$$

$$\frac{\partial E}{\partial w_{ij}^{(1)}} = \frac{1}{m}\sum_{p=1}^{m}\frac{\partial E^p}{\partial w_{ij}^{(1)}} \tag{37}$$

$$\frac{\partial E^p}{\partial b_j^{(1)}} = \sum_{k=1}^{K}\Big(\frac{\partial E^p}{\partial (x^{(p,3)})_k} w_{jk}^{(2)}\Big) H\big[\sum_{q=1}^{n_{in}} w_{qj}^{(1)}(x^{(p,1)})_q + b_j^{(1)}\big] \tag{38}$$

$$\frac{\partial E}{\partial b_j^{(1)}} = \frac{1}{m}\sum_{p=1}^{m}\frac{\partial E^p}{\partial b_j^{(1)}} \tag{39}$$

initialisation aleatoire de tous les poids
faire
    pour chaque exemple d'entrainement
        prediction: calcul de $x^{(p,2)}$, $x^{(p,3)}$, et $softmax(x^{(p,3)})$
        pour chaque $j \in \{1, ..., n_{out}\}$
            calcul de $\frac{\partial E^p}{\partial (x^{(p,3)})_j}$
        pour chaque $(i,j) \in \{1, ..., n_{hidden}\} \times \{1, ..., n_{out}\}$
            calcul de $\frac{\partial E^p}{\partial w_{ij}^{(2)}}$
        pour chaque $(i,j) \in \{1, ..., n_{in}\} \times \{1, ..., n_{hidden}\}$
            calcul de $\frac{\partial E^p}{\partial w_{ij}^{(1)}}$
        pour chaque $j \in \{1, ..., n_{out}\}$
            calcul de $\frac{\partial E^p}{\partial b_j^{(2)}}$
        pour chaque $j \in \{1, ..., n_{hidden}\}$
            calcul de $\frac{\partial E^p}{\partial b_j^{(1)}}$
    pour chaque $(i,j) \in \{1, ..., n_{hidden}\} \times \{1, ..., n_{out}\}$
        calcul de $\frac{\partial E}{\partial w_{ij}^{(2)}}$ (moyenne de $\frac{\partial E^p}{\partial w_{ij}^{(2)}}$)
    pour chaque $(i,j) \in \{1, ..., n_{in}\} \times \{1, ..., n_{hidden}\}$
        calcul de $\frac{\partial E}{\partial w_{ij}^{(1)}}$ (moyenne de $\frac{\partial E^p}{\partial w_{ij}^{(1)}}$)
    pour chaque $j \in \{1, ..., n_{out}\}$
        calcul de $\frac{\partial E}{\partial b_j^{(2)}}$
    pour chaque $j \in \{1, ..., n_{hidden}\}$
        calcul de $\frac{\partial E}{\partial b_j^{(1)}}$
    Mise a jour des poids: $W = W - \eta\frac{\partial E}{\partial W}$ (descente de gradients)
    Mise a jour des biais: $b = b - \eta\frac{\partial E}{\partial b}$ (descente de gradients)
    jusqu'a ce que tous les exemples d'entrainement soient tries correctement (ou tout autre critere d'arret)