Université de Technologie de Compiègne

Génie informatique

Rétropropagation

Authors :

Nacim Khalis et Antoine Collas

27 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 :

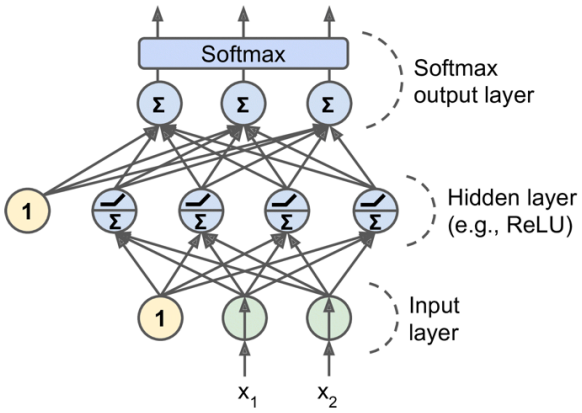

FIGURE 1 – Perceptron multi-couches tiré de 'Hands-On Machine Learning with Scikit-Learn and TensorFlow' de Aurélien Géron.

2 Rétropropagation appliquée à un réseau 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 Rétropropagation

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.

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

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 :

$$\frac{\partial E}{\partial w_{ij}^{(2)}} = -\frac{1}{m}\sum_{p=1}^{m}\sum_{k=1}^{n_{out}} y_k^{(p)} \frac{\partial log(\hat{p}_k^{(p)})}{\partial w_{ij}^{(2)}} \tag{4}$$

$$\frac{\partial log(\hat{p}_k^{(p)})}{\partial w_{ij}^{(2)}} = \sum_{q=1}^{n_{out}} \frac{\partial log(\hat{p}_k^{(p)})}{\partial (x^{(p,3)})_q}\frac{\partial (x^{(p,3)})_q}{\partial w_{ij}^{(2)}} \tag{5}$$

or si $q \neq j$

$$\frac{\partial (x^{(p,3)})_q}{\partial w_{ij}^{(2)}} = 0 \tag{6}$$

donc

$$\frac{\partial log(\hat{p}_k^{(p)})}{\partial w_{ij}^{(2)}} = \frac{\partial log(\hat{p}_k^{(p)})}{\partial (x^{(p,3)})_j} \frac{\partial (x^{(p,3)})_j}{\partial w_{ij}^{(2)}} \tag{7}$$

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.