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

8 juin 2018

1 Réseau de neurones : perceptron

L'objectif est de programmer un perceptron afin de classer un ensemble de bouteilles de vins en 3 ensembles (dataset disponible ici). Dans la suite de ce document nous souhaitons entrainer un perceptron ayant une couche cachée sur cet ensemble de données.

L'architecture de notre perceptron est la suivante :

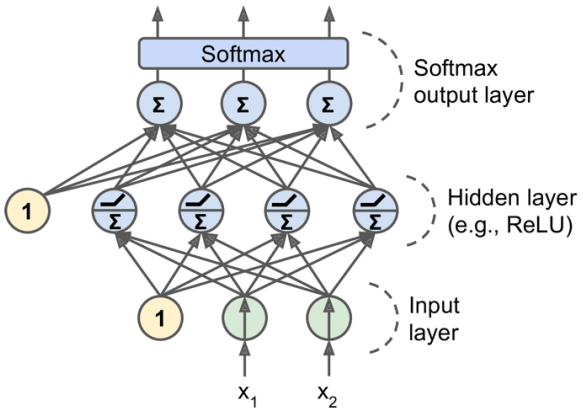

FIGURE 1 – Perceptron multi-couches tiré de [2].

Le réseau est composé d'une couche d'entrée avec 13 neurones (les bouteilles de vins ont 13 caractéristiques et donc sont représentées par des vecteurs de 13 dimensions), une couche cachée avec un nombre de neurones à régler (un hyperparamètre de notre réseau de neurones) et de 3 sorties (une sortie par classe à prédire). La fonction d'activation est une fonction RELU : $RELU(x) = max(0, x)$.

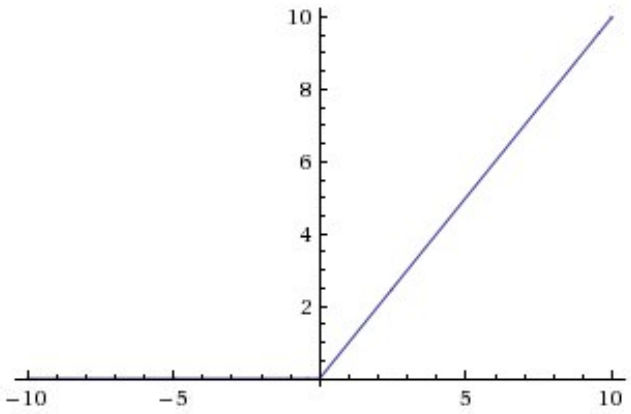

FIGURE 2 – Fonction RELU.

Le choix de cette fonction d'activation est motivé par les raisons suivantes (tirées de [1]) :

— Rapide à calculer.

— Biomimétisme. Tout d'abord La fréquence d'impulsions d'un neurone biologique en fonction de l'intensité reçue peut être assimilée à une fonction RELU. En effet, la fréquence d'impulsions d'un neurone biologique est une fonction de l'intensité reçue en entrée. En dessous d'un certains

seuil le neurone biologique ne renvoie aucune impulsion. Au dessus de ce seuil la fréquence d'impulsions croit linéairement par rapport à l'intensité reçue (jusqu'à $1 \times 10^{-9} A$). Cette linéarité est visible dans le carré rouge de la figure 3. De plus, le réseau est épars. En, effet si la combinaison linéaire de ce que reçoit un neurone est négative alors le neurone ne renvoie rien. La figure 4 illustre ce phénomène. Dans le cerveau humain seuls 1 à 4% des neurones sont actifs en même temps. [1] indique que 83% des neurones d'un réseau entrainé sur le MNIST renvoient 0.

— La variation du nombre de neurones actifs permet à un modèle de contrôler la dimensionnalité du réseau.

— [1] indique avoir obtenu de meilleurs performances sur 4 jeux de données différents en utilisant un RELU à la place d'une tangente hyperbolique

— Une dernière raison concerne le problème de disparition de gradients. Ce problème ne concerne que les réseaux très profonds, c'est à dire ayant un nombre important de couches cachées. Il ne concerne donc pas notre réseau mais il est intéressant de l'expliquer. Le problème des premiers réseaux de neurones est qu'ils utilisaient la fonction d'activation sigmoïde. Or, $\frac{\partial sigmode(x)}{\partial x} \in]0, \frac{1}{4}]$. Donc en appliquant la règle de la chaine nous allons multiplier successivement des nombre appartenant à $]0, \frac{1}{4}]$. Donc les dérivées partielles vont décroitre exponentiellement au fur et à mesure de la rétropropagation. L'entrainement des premières couches va être très lent voir totalement bloqué. La fonction RELU n'a pas ce problème puisque les dérivées partielles de RELU calculées en passant par les chemins bleus de la figure 4 sont égales à 1. Cela permet de ni augmenter ni diminuer exponentiellement la rétropropagation de l'erreur.

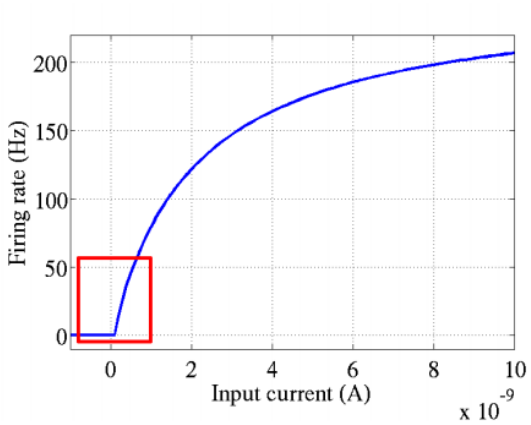

FIGURE 3 – Fréquence d'impulsions en fonction de l'intensité d'un neurone biologique (figure tirée de [1]).

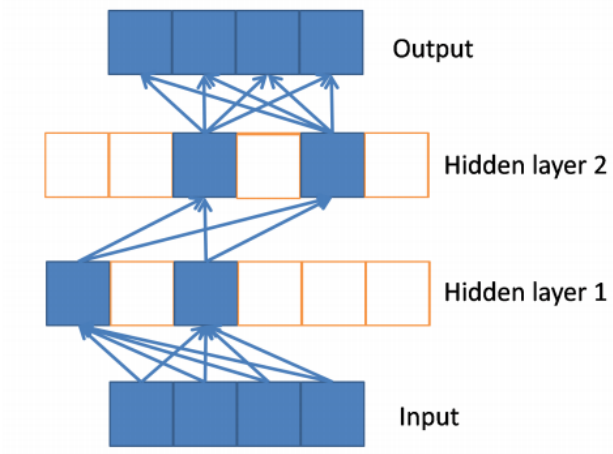

FIGURE 4 – Réseau (perceptron avec deux couches cachées) utilisant la fonction d'activation RELU (figure tirée de [1]). Les cases blanches représentent les neurones dont le RELU renvoie 0. Les cases bleues représentent les neurones dont le RELU renvoie une valeur non nulle (strictement positive).

La fonction d'erreur est l'entropie croisée qui donne une mesure d'erreur pour un problème de classification. $E = -\frac{1}{m}\sum_{i=1}^{m}\sum_{k=1}^{K}(y_k^{(i)} \times ln(\hat{p}_k^{(i)}))$

$y_k^{(i)}$ 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.

Si le réseau a trouvé une probabilité de 1 pour la classe correspondant à l'individu alors l'erreur est nulle. Sinon l'erreur croit exponentiellement quand la probabilité tend vers 0. L'entropie croisée pénalise donc très fortement les grosses erreurs et faiblement les petites erreurs.

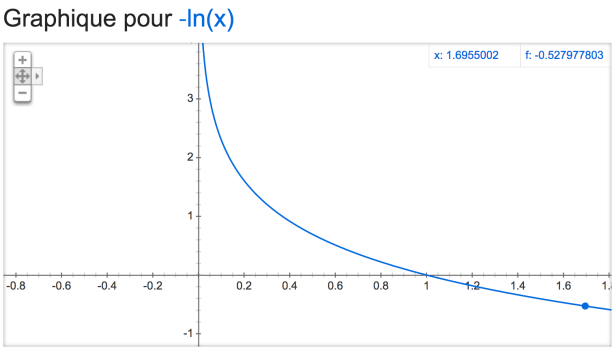

FIGURE 5 – f(x)=-ln(x)

2 Entrainement d'un réseau de neurones de classification

Dans cette partie l'objectif est d'entrainer notre réseau de neurones. L'entrainement d'un réseau de neurones consiste à régler les paramètres (poids des connexions) afin de minimiser la fonction d'erreur (l'entropie croisée dans notre cas). Cette minimisation est réalisée par une descente de gradient : le

gradient de la fonction d'erreur est calculé puis les poids sont mis à jour et on recommence jusqu'à atteindre un minimum. La mise à jour est réalisée de la manière suivante :

$$W := W - \eta \frac{\partial E}{\partial W}$$

Nous avons donc besoin de calculer les dérivées partielles de l'erreur par rapport à chacun des poids.

2.1 Notation

Nous détaillons l'ensemble des notations pour notre perceptron ayant une couche cachée.

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). K=3 dans notre cas.

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. $n_{in} = 13$ dans notre cas.

n_{hidden} est le nombre de neurones de la couche cachée (sans le biais). n_{hidden} est un hyperparamètre (donc réglable avant l'entrainement).

$b_i^{(l)}$ 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 $l+1$.

$(x^{(p,l)})_i$ est la sortie du neurone i de la couche l pour le p^{eme} individu d'entrainement. Donc $(x^{(p,2)})_i = relu(\sum_{k=1}^{n_{in}} w_{ki}^{(1)}(x^{(p,1)})_k + b_i^{(1)})$

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

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

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 $\frac{\partial E}{\partial w_{ij}^{(2)}}$ $\forall (i,j) \in \{1, ..., n_{hidden}\} \times \{1, ..., K\}$.

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

l'erreur :

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

avec

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

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

$$\forall (i,j) \in \{1,...,n_{hidden}\} \times \{1,...,K\} \quad \frac{\partial E}{\partial w_{ij}^{(2)}} = -\frac{1}{m}\sum_{p=1}^{m}\sum_{k=1}^{K} y_k^{(p)} \frac{\partial ln(\hat{p}_k^{(p)})}{\partial w_{ij}^{(2)}}$$

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 :

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

$$ln(\hat{p}_k^{(p)}) = (x^{(p,3)})_k - ln(\sum_{q=1}^{K} e^{(x^{(p,3)})_q})$$

Si $k \neq j$

$$\frac{\partial ln(\hat{p}_k^{(p)})}{\partial (x^{(p,3)})_j} = -\frac{e^{(x^{(p,3)})_j}}{\sum_{q=1}^{K} e^{(x^{(p,3)})_q}} = -softmax(x^{(p,3)})_j$$

Sinon (dans le cas où $k = j$) :

$$\frac{\partial ln(\hat{p}_k^{(p)})}{\partial (x^{(p,3)})_j} = 1 - softmax(x^{(p,3)})_j$$

Donc $\forall k$:

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

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

De plus,

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

Donc,

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

$$\frac{\partial ln(\hat{p}_k^{(p)})}{\partial w_{ij}^{(2)}} = \Big[\delta_{kj} - softmax(x^{(p,3)})_j\Big](x^{(p,2)})_i$$

Ce qui donne finalement :

$$\frac{\partial E}{\partial w_{ij}^{(2)}} = -\frac{1}{m}\sum_{p=1}^{m}\sum_{k=1}^{K} y_k^{(p)}\Big[\delta_{kj} - softmax(x^{(p,3)})_j\Big](x^{(p,2)})_i$$

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 modifié 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 modifié sinon il est fortement modifié 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 modifié. Si elle est élevée (alors qu'elle devrait être faible), le poids est fortement modifié.

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

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

$$\frac{\partial ln(\hat{p}_k^{(p)})}{\partial w_{ij}^{(1)}} = \frac{\partial ln(\hat{p}_k^{(p)})}{\partial (x^{(p,2)})_j}\frac{\partial (x^{(p,2)})_j}{\partial w_{ij}^{(1)}} \tag{3}$$

$$\frac{\partial ln(\hat{p}_k^{(p)})}{\partial (x^{(p,2)})_j} = \sum_{q=1}^{K}\frac{\partial ln(\hat{p}_k^{(p)})}{\partial (x^{(p,3)})_q}\frac{\partial (x^{(p,3)})_q}{\partial (x^{(p,2)})_j} \tag{4}$$

$$\frac{\partial (x^{(p,3)})_k}{\partial (x^{(p,2)})_j} = w_{jk}^{(2)} \tag{5}$$

D'après les équations (1), (4), (5) :

$$\frac{\partial ln(\hat{p}_k^{(p)})}{\partial (x^{(p,2)})_j} = \sum_{q=1}^{K}\Big[\delta_{kq} - softmax(x^{(p,3)})_q\Big]w_{jk}^{(2)} \tag{6}$$

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

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

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

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

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

Donc :

$$\frac{\partial (x^{(p,2)})_j}{\partial w_{ij}^{(1)}} = H\left[\sum_{k=1}^{n_{in}} w_{kj}^{(1)}(x^{(p,1)})_k + b_j^{(1)}\right](x^{(p,1)})_i \tag{7}$$

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

D'après les équations (3), (6), (7) :

$$\frac{\partial ln(\hat{p}_k^{(p)})}{\partial w_{ij}^{(1)}} = H\left[\sum_{q=1}^{n_{in}} w_{qj}^{(1)}(x^{(p,1)})_q + b_j^{(1)}\right](x^{(p,1)})_i \sum_{q=1}^{K}\left[\left[\delta_{kq} - softmax(x^{(p,3)})_q\right] w_{jq}^{(2)}\right] \tag{8}$$

D'après les équations (2) et (8) :

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

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 $\sum_{q=1}^{K}\left[\left[\delta_{kq} - softmax(x^{(p,3)})_q\right] w_{jq}^{(2)}\right]$)

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 :

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

En appliquant la règle de la chaine :

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

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

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

D'après les deux équations précédentes :

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

Donc

$$\frac{\partial E}{\partial w_{ij}^{(1)}} = \frac{1}{m} \sum_{p=1}^{m} \sum_{k=1}^{K} \Big(\frac{\partial E^p}{\partial (x^{(p,3)})_k} w_{jk}^{(2)} \Big) H\big[\sum_{q=1}^{n_{in}} w_{qj}^{(1)} (x^{(p,1)})_q + b_j^{(1)}\big] (x^{(p,1)})_i \tag{9}$$

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 (rétropropagation) puis à faire une descente de gradients. Les poids sont initialisés avec une loi normale d'espérance 0 et de variance 1.

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

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

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

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

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

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

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

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

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

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 $j \in \{1, ..., K\}$
            calcul de $\frac{\partial E^p}{\partial (x^{(p,3)})_j}$
        pour chaque $(i, j) \in \{1, ..., n_{hidden}\} \times \{1, ..., K\}$
            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 $j \in \{1, ..., K\}$
            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 $(i, j) \in \{1, ..., n_{hidden}\} \times \{1, ..., K\}$

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 $j \in \{1,...,K\}$

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)

4 Implémentation

Nous avons implémentaté ce perceptron à une couche et l'avons entrainé sur le jeu de données indiqué en première partie. Les 3 classes ayant à peu près le même effectif nous avons calculé le nombre d'erreurs divisé par l'effectif total sur un ensemble de test représentant 30% des données (le reste servant à l'entrainement).

Nombre de neurones dans la couche cachée	Pourcentage d'erreur
5	39%
10	6%
30	6%
50	2%
100	0%
1000	0%

Références

[1] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In Geoffrey Gordon, David Dunson, and Miroslav Dudík, editors, *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, volume 15 of *Proceedings of Machine Learning Research*, pages 315–323, Fort Lauderdale, FL, USA, 11–13 Apr 2011. PMLR.

[2] Aurélien Géron. *Hands-On Machine Learning with Scikit-Learn and TensorFlow.* O'Reilly Media, 2017.