23 juin 2018

Reconnaissance de caractères

TIPE - Planchon, Croce, Durand & Marzook

Table des matières

Reconnaissance de caractères

TIPE - Planchon, Croce, Durand & Marzook

I. La reconnaissance par Intelligence artificielle

I.1. Présentation des réseaux de neurones

I.1.a Les notations

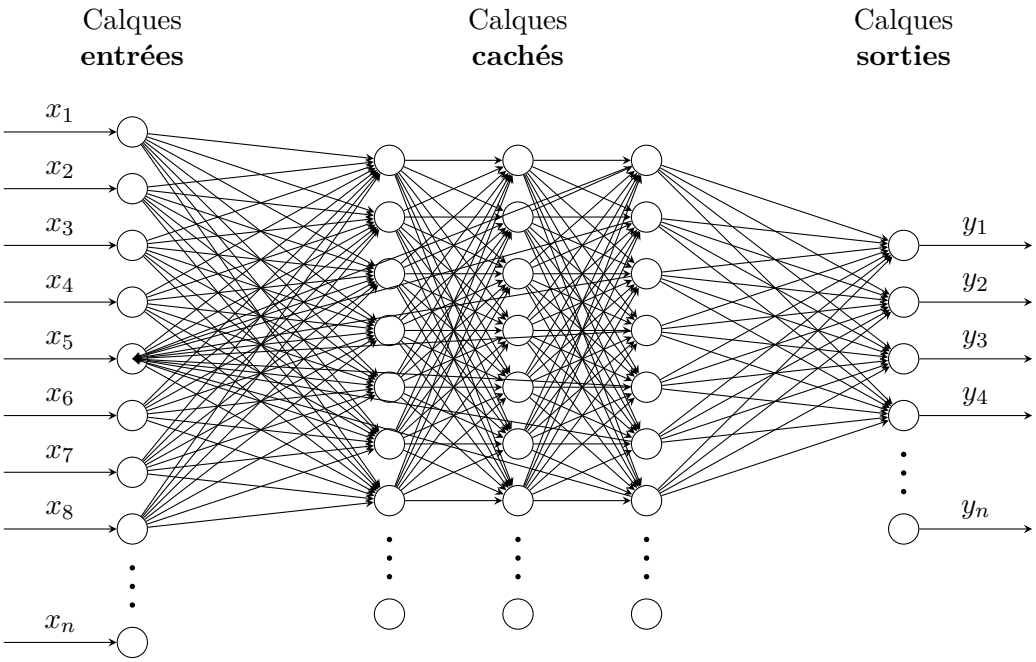

FIGURE 1 – Représentation d'un réseau de neurones

Notations utilisées :

- $x_i^{(n)}$: donnée d'entrée à la colonne n et ligne i.
- $y_i^{(n)}$: donnée de sortie à la colonne n et ligne i.
- $w_{ba}^{(n)}$: poids du neurone b vers a et colonne n.
- $b_i^{(n)}$: biai à la colonne n et ligne i.
- t_i : sortie **voulue** à la ligne i.
- $X^{(n)}$: Matrice des entrées à la colonne n.
- $Y_i^{(n)}$: Matrice sortie à la colonne n.
- $W^{(n)}$: Matrice des poids de la ligne $n-1$ à n.s
- $B^{(n)}$: Matrice des biai à la colonne n.

On note $\sigma(x) = \dfrac{1}{1+e^x}$ et $\sigma(x)' = \sigma(x)(1-\sigma(x))$
et $h(n) = \sum_n w_i^{(n)} y_i^{(n-1)}$

I.1.b Fonctionnement d'un neurone

I.1.c Idée de réseau de neurones

I.2. La méthode de rétropropagation par calcul du gradient

I.2.a Qu'est-ce que la rétropropagation

Nous avons vu que les réseaux de neurones "apprenaient", nous avons aussi montré que cet apprentissage était le coeur des réseaux de neurones et leur force. Mais nous n'avons pas encore expliqué le fonctionnement de cet apprentissage. C'est ce dont cette partie va traiter.

Pour l'instant un réseau de neurone n'est qu'une grosse fonction qui prend n vecteur x_i (par exemple tous les pixels d'une image) en entrée et s'entraîne à donner en sortie m résultats y_i (par exemple la valeur d'un chiffre).
Sauf que contrairement à des fonctions simples comme une application linéaire, qui est déterminée entièrement par l'image d'une base, on ne peut pas donner une "base d'image de caractères" à notre réseau de neurone. Le fonctionnement d'un réseau de neurone est donc calqué sur la nature, et plus précisément sur le fonctionnement du cerveau : lui travaille grâce à l'apprentissage.

Mais là encore une question se pose, comment définir le mot "apprentissage" en mathématique (et donc en informatique) ? Cette grande question a été résolue en partie en 1980 par David Rumelhart, Geoffrey Hinton, et Ronald Williams dans le papier **"Learning representation by back-propagating errors"**. En appliquant cette méthode au réseau de neurone, il sera capable de s'affiner pour avoir un pourcentage d'erreur faible, voir très faible dans les meilleurs cas.

Le but de l'algorithme est de calculer un minimum local de la fonction "réseau de neurone". C'est à dire trouver la solution la plus optimale au problème posé, dans notre cas, la reconnaissance d'image. C'est à dire trouver la meilleur combinaison de poids et de biais (étant donné que ce sont les seules variables que nous pouvons changer). Pour calculer l'efficacité de la combinaison de poids et biais, le réseau va devoir calculer son erreur

$$E = \frac{1}{2}\sum_{n}(t_i - y_i)^2$$

Le but de la méthode par rétropropagation est de faire tendre cette fonction vers un minimum local, au mieux vers 0. Il est important de noter ici, qu'il est possible que le réseau ne puisse pas trouver la solution la plus optimale. Cependant, il est certain qu'il en trouvera une qui est optimisée.

Entraîner et nourrir le réseau qu'une fois sera loin de suffir à faire fonctionner l'algorithme de rétropropagation. Il va falloir répéter cette étape de nombreuses fois. C'est pourquoi la phase d'entraînement et de "nourrissage" du réseau de neurone est séparée en epoch (base de donnée entière), batch (partie d'un epoch) et itérations.

Nous devons séparer l'étape du nourrissage pour, dans un premier temps optimiser le temps de calcul nécessaire, mais aussi pour éviter de "gaver" le réseau. C'est à dire trop le nourrir, menant vers le gavage, c'est à dire un réseau qui est incapable de minimiser sa fonction et donc qui ne convergera jamais vers une solution.

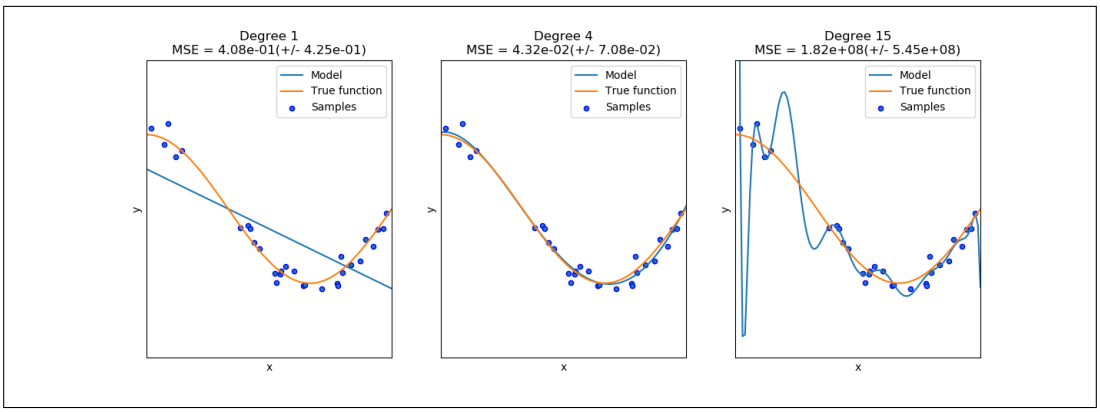

FIGURE 2 – Exemple de nourrissage de réseaux

On voit clairement que les images de gauche et de droite font le travail demandé, mais celui de gauche n'est pas précis du tout, et celui de droite l'est beaucoup trop. Le réseau de droite sera surement très lent à entrainer car il est trop précis. Aussi il pourra devenir moins précis pour des cas un peu spéciaux.

Ainsi on ne nourrit pas le réseau avec un batch de la taille de l'epoch, mais plutôt avec plusieurs batch, eux-même ayant plusieurs itérations sur le réseau. Malheureusement, la meilleur découpe des batch et le nombre d'itération optimale n'est pas obtenable pour l'instant par calcul (peut-être prochainement avec un réseau de neurone ?), il faut donc essayer. A chaque fois qu'un batch a été traité par le réseau de neurone, on calcul l'erreur moyenne de chaque batch. C'est cette erreur moyenne qui est utilisée pour le calcul du gradient.

I.2.b Les mathématiques de la rétropropagation

Le calcul du gradient est le coeur de la rétropropagation, et donc le coeur des réseaux de neurones. Optimiser l'erreur revient à trouver un minimum. C'est à dire trouver un endroit où la fonction "dérivée" s'annule. Or nous n'avons pas de forme dérivable de cette fameuse fonction. Pour optimiser le réseau il suffit donc d'utiliser la méthode des infi-décimaux. C'est à dire calculer ∇E (gradient de E). Voici la méthode :

— On calcul l'erreur $e_i^n = \sigma'(h_i^n)[t_i - y_i]$

— On propage l'erreur calculé vers l'arrière : $e_j^{(n-1)} = g'^{(n-1)}(h_j^{(n-1)}) \sum_i w_{ij} e_i^{(n)}$,

avec $e_j^{(n)} = \sum_i [t_i - y_i] \frac{\partial y_i}{\partial h_j^{(n)}}$

— Notre but est de minimiser $E = \frac{1}{2} \sum_i (t_i - y_i)^2$,

c'est à dire $\frac{\partial E}{w_{ab}^{(l)}} = \sum_k \frac{\partial x_k^{(n-1)}}{w_{ab}^{(l)}} \sum_i w_{ik}^{(n)} e_i^{(n)}$

Une fois avoir calculé les erreurs, on met à jour les poids et les biais de neurones :

$$w_{ij}^{(n)} = w_{ij}^{(n)} + \lambda e_i^{(n)} x_j^{(n-1)}$$

Le gradient calculé, il suffit de mettre à jour les poids et les biais avec leur valeur respective (calculée avec $-\nabla E$).

Cette méthode est récursive. On ne peut pas calculer les poids du neurones à la ligne n si on a pas déjà calculé les poids à la ligne $n+1$. D'où cette idée de back propagation. Il faut aller de l'erreur vers les poids de l'entrée. Pour résumer le fonctionnement d'un réseau de neurone par CNN, voici l'algo en pseudo code :

Pseudo Code - 1: CNN

```
1 Initialiser les poids
2 repeter
3         calculer le "feed-forward"
4         calculer l'erreur
5         calculer le gradient
6         mettre a jour les poids
7 jusqu'a precision > precision voulue
```

I.3. Application de la technique à la reconnaissance de caractère

I.3.a Utilisation de la BDD MNIST sur les algorithmes

I.3.b Créations de nos propres bases