Date : 20 janvier 2018
Auteur : Vincent CHAMBRIN
Fabien Rouillon

Projet Mathématiques et Applications

Deep networks for character recognition

Sommaire

Le but de ce projet est d'étudier l'utilisation de réseaux de neurones profonds pour la reconnaissance de caractères manuscrits. Différentes méthodes de classifications seront utilisées et comparées à un réseau de neurones simple et à des algorithmes n'utilisant pas de réseaux de neurones. Le langage Python sera utilisé pour l'écriture des programmes, ce dernier fournissant bon nombre de bibliothèques de *machine learning*. Pour mieux comprendre les algorithmes et idées mis en jeux, une brève introduction aux principes des réseaux de neurones sera présentée.

Mots-clés : réseaux de neurones, *machine learning*, *deep networks*, reconnaissance de caractères, Python.

Table des matières

Sommaire ii

Table des matières iii

1 Introduction 1

2 Les réseaux de neurones 2
- 2.1 Le modèle du perceptron . . . 2
- 2.2 Structure des réseaux de neurones . . . 6
- 2.3 Apprentissage par la descente de gradient . . . 7
- 2.4 L'algorithme de rétro-propagation . . . 7

3 Reconnaissance via un réseau de neurones simple 8
- 3.1 Construction du réseau de neurones . . . 8
- 3.2 Apprentissage . . . 8
- 3.3 Résultats . . . 8

4 Reconnaissance basée sur l'extraction de *features* 9
- 4.1 Binarisation de l'image . . . 9
- 4.2 Densités de pixels coloriés . . . 9
- 4.3 Nombre de croissements (*crossings*) . . . 9
- 4.4 Histogramme des projections . . . 9
- 4.5 Moments . . . 10
- 4.6 Transformée de Fourier du contour . . . 10
- 4.7 Transformée de Fourier de l'image . . . 10

5 Reconnaissance basée sur les réseaux convolutionnels 11

6 Comparaison des résultats 12

1 Introduction

La reconnaissance de caractères, ou plus généralement de textes manuscrits, est un domaine en pleine expansion. De nombreux géants de l'informatique comme Google, Apple ou Microsoft se battent pour offrir à leurs utilisateurs (et probablement à d'autres fins) le meilleur système de reconnaissance de texte. Les utilisations sont nombreuses :

— application de prises de notes ;
— saisie de formulaires ;
— saisie d'une adresse pour le système GPS de certaines voitures ;
— reconnaissance d'adresse postale sur les enveloppes ;
— reconnaissance d'un montant sur les chéques.

Dans le cadre de ce projet, on se contentera de tenter de reconnaître des chiffres. On utilisera pour cela le jeu de données MNIST, qui est un jeu de données contenant 60000 images d'apprentissage et 10000 images de test. Ces images, représentants des chiffres écrits à la main, sont en niveaux de gris et font 28 pixels par 28 pixels. Elles sont issues de deux jeux de données collectés par le NIST (United States' National Institute of Standards and Technology). A chaque image est associé un label (i.e. le chiffre représenté). Ce jeu de données est pratiquement devenu un standard dans le monde de la reconnaissance de caractères.

Ce rapport est découpé en plusieurs parties. Dans un premier temps, une introduction sur les réseaux de neurones permettra de se familiariser avec ces derniers. On sera alors en mesure d'écrire nous-même les algorithmes pour les réseaux les plus simples et nous pourrons construire un premier réseau permettant de répondre au problème de classification que l'on se pose.
Ensuite, nous pourrons mettre en oeuvre des techniques de'*machine learning* plus poussés et plus adaptés à notre domaine. On commencera pour cela par étudier certaines techniques propres au problème de reconnaissance de caractères (l'extraction de *features*). Enfin, nous nous pencherons sur l'utilisation de réseaux de neurones convolutionnels, ces derniers étant particulièrement adaptés aux problèmes de vision par ordinateur et de traitement d'image.

2 Les réseaux de neurones

2.1 Le modèle du perceptron

Définition 1 (Fonction de Heaviside)**.** *On appelle fonction de Heaviside la fonction H définie de $\mathbb{R}$ dans $\{0,1\}$ par :*

$$H(x) = \begin{cases} 1, & \textit{si } x \geq 0 \\ 0, & \textit{sinon.} \end{cases}$$

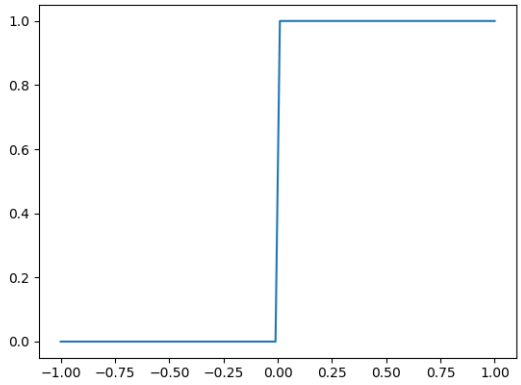

FIGURE 1 – La fonction échelon unité, ou fonction de Heaviside.

Le but du perceptron est de modéliser le comportement du neurone biologique. Ce dernier est stimulé par des signaux qui lui parviennent par ses dendrites et, si la stimulation est suffisament importante, renvoie un signal à d'autres neurones au travers de son axone.

En notant $x_1, x_2, \ldots, x_n$ ces stimulations, qui sont donc les entrées du perceptron, et $w_1, w_2, \ldots, w_n$ les pondérations associées à ces entrées, on peut simplement exprimer le comportement du perceptron par :

$$\text{sortie } = \begin{cases} 1 \text{ si } w \cdot x \geq \text{ seuil} \\ 0 \text{ si } w \cdot x < \text{ seuil.} \end{cases}$$

Par la suite, on préférera, plutôt que de considérer un seuil, considérer une quantité b que l'on appelera biais définie par $b = -\text{seuil}$, et on écrira :

$$\text{sortie } = H(w \cdot x + b)$$

Les perceptrons peuvent être utilisés pour coder des fonctions logiques. Par exemple, en prenant $w = (1,1)$ et $b = -2$, notre perceptron code un ET logique (avec la convention 0 pour FAUX et 1 pour VRAI).

De même, en prenant $w = (-2,-2)$ et $b = 3$, on code la fonction NAND (négation du ET logique). Pour s'en convaincre, on peut dresser le tableau des sorties en fonctions de toutes les entrées possibles comme cela est fait dans la table 1.

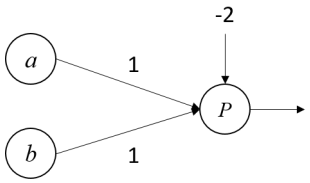

FIGURE 2 – Fonction ET logique.

x_1	x_2	$wx + b$	Sortie
0	0	3	1
0	1	1	1
1	0	1	1
0	0	-1	0

TABLE 1 – Table de vérité du perceptron.

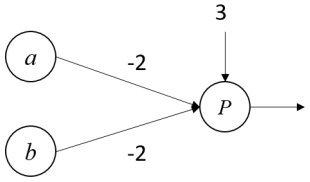

FIGURE 3 – Fonction NAND.

Comme la fonction NAND permet de coder n'importe quelle fonction logique, on en déduit qu'en utilisant la sortie de perceptrons comme entrées d'autres perceptrons, on peut construire n'importe quelle fonction logique.

Exercice : Coder la fonction OU logique pour n entrées.

Cette idée d'empiler des couches de perceptrons pour coder une fonction logique conduit naturellement à l'idée que l'on pourrait se faire d'un réseau de neurones.

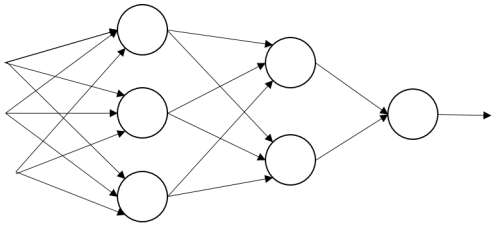

FIGURE 4 – Un réseau de perceptrons.

Jusqu'à présent, nous nous sommes restreint à des entrées binaires (0 ou 1) codant des valeurs logiques. Ceci n'est pas nécessaire : les entrées d'un perceptron peuvent très bien être des réels. Plaçons nous, pour les exemples qui vont suivres, dans $\mathbb{R}^2$. Pour chaque point (x, y) du plan, on donnera x comme première entrée d'un perceptron et y comme seconde entrée. Le perceptron ayant pour paramètre $w = (\alpha_x, \alpha_y)$ et b va alors séparer l'espace en deux plans par la droite d'équation $\alpha_y y + \alpha_x x + b = 0$. Les points tels que $\alpha_y y + \alpha_x x + b \geq 0$ auront pour sortie 1, et les autres auront pour sortie 0.

Ceci est illustré par la figure 5 dans laquelle la partie verte correspond à une sortie positive du perceptron

dès lors que $y \geq x$.

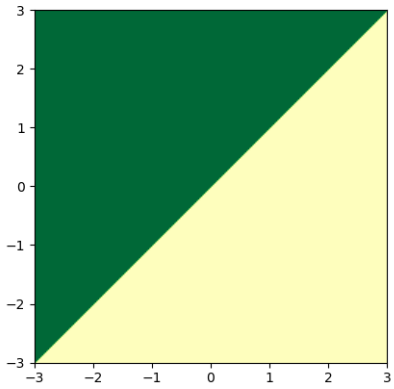

FIGURE 5 – Découpage du plan en deux par un perceptron.

En utilisant plusieurs perceptron, on va pouvoir effectuer plusieurs sous-découpage de l'espace, et en utilisant un ET logique dans une dernière couche on va pouvoir récuperer la partie de l'espace correspondant à l'intersection de toutes les parties correspondants aux sorties positives des perceptrons de la couche précédente.

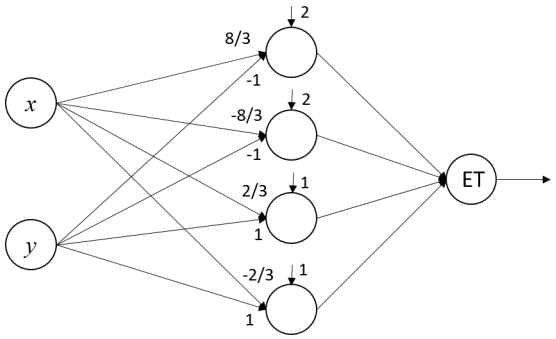

FIGURE 6 – Un premier réseau de perceptrons.

Cette approche a ses limites car elle ne permet d'obtenir qu'une partie convexe du plan. Pour obtenir des parties plus complexes (e.g. non convexes, ou non fortement connexes), on peut utiliser d'autres fonctions logiques, comme le OU.

Il n'est pas difficile d'imaginer que l'on puisse étendre ce que l'on vient de faire en 2 dimensions à un nombre plus grand de dimensions ; et avec un non plus une seule sortie mais plusieurs (correspondant par exemple chacune à une classe, à un chiffre que l'on souhaite reonnaître). On pourrait donc en théorie construire un réseau de neurones avec des perceptrons qui serait capable de reconnaître des chiffres. L'un des problèmes de cette approche est qu'il faudrait déjà connaître les coefficients (ou paramètres) de l'ensemble des perceptrons du réseau. On pourrait alors avoir envie d'écrire un algorithme permettant de construire un tel réseau. On peut imaginer plusieurs principes pour cet algorithme :‘

1. tester tous les coefficients possibles pour les perceptrons ;
2. partir d'une configuration aléatoire et modifier petit à petit les coefficients jusqu'à obtenir un résultat satisfaisant.

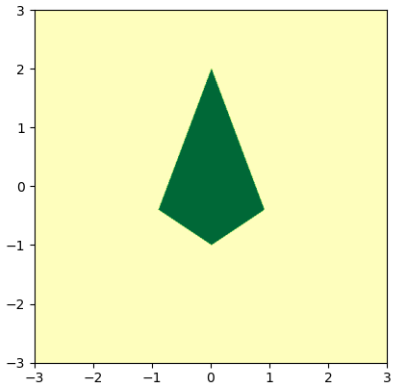

FIGURE 7 – Découpage du plan par un réseau de perceptrons.

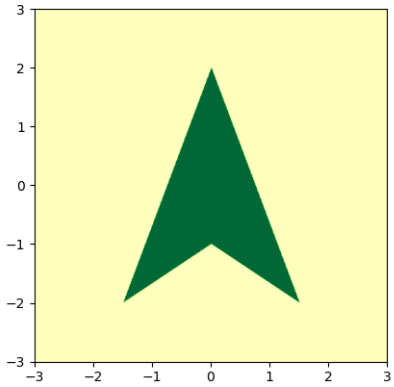

FIGURE 8 – Un découpage non convexe du plan avec un réseau de perceptrons à deux couches.

La première idée est tout simplement irréalisable car il y a une infinité de valeurs possibles pour chaque coefficient du réseau. On pourrait penser à prendre une approche probabiliste en testant un certains nombres de réseau et en espérant tomber sur un bon ; ce qui paraît néanmoins très improbable ! La deuxième idée n'est également pas envisageable car elle souffre d'un problème difficile à gérer. La discontinuité de la fonction de Heaviside fait qu'un petit changement de coefficient peut faire passer la sortie d'un neurone de 0 à 1 ce qui peut correspondre à un changement significatif de l'entrée des neurones de la couche suivante. Il est donc compliqué d'évaluer l'impact d'un changement à une couche donnée sur les couches qui suivent.

L'idée qui va être exploré dans les paragraphes suivants est d'utiliser une autre fonction, appelée fonction d'activation, que la fonction échelon d'Heaviside ; et de faire en sorte que cette fonction d'activation soit dérivable (et donc continue) pour pouvoir utiliser des résultats d'analyse et pouvoir ainsi estimer l'impact du changement d'un coefficient. On sera alors en mesure de modifier un réseau généré aléatoirement pour le faire résoudre notre problème de classification (on parlera d'apprentissage).

2.2 Structure des réseaux de neurones

Définition 2 (Fonction sigmoïde)**.** *On appelle fonction sigmoïde la fonction définie de* $\mathbb{R}$ *dans* $[0,1]$ *par :*

$$\sigma(x) = \frac{1}{1+e^{-x}}$$

On notera que $\lim_{x\to+\infty} \sigma(x) = 1$ et $\lim_{x\to-\infty} \sigma(x) = 0$. Ainsi, asymptotiquement, la fonction sigmoïde à le même comportement que la fonction de Heaviside.

Entre les deux, on à un comportement qui diffère fortement du perceptron avec notamment la non discontinuité en 0 et $\sigma(0) = \frac{1}{2}$. Dans le cadre de notre utilisation pour la classification on aura tendance à considérer que la sortie du neurone est à VRAI dès qu'elle sera supérieure à une certaine valeur (typiquement $\frac{1}{2}$).

On a également quelques propriétés intéressantes de symétrie. En effet,

$$\sigma'(x) = \frac{e^{-x}}{1+e^{-x}} = \sigma(x)(1-\sigma(x))$$

$$\sigma'(x) - \sigma'(-x) = \frac{e^{-x}(e^{2x}+2e^{x}+1) - e^{x}(e^{-2x}+2e^{-x}+1)}{(1+e^{-x})^2(1+e^{x})^2} = 0$$

ce qui signifie que la sigmoïde admet pour centre de symétrie le point $(0, \frac{1}{2})$.

Cette fonction a de plus l'avantage d'être dérivable en tout point (et de dérivée non nulle), ce qui nous servira plus tard dans le cadre de l'apprentissage.

Définition 3 (Modélisation d'un neurone)**.** *En notant* b *le biais du neurone,* w *son vecteur des poids et* σ *sa fonction d'activation, on peut écrire la sortie* y *d'un neurone en fonction de son entrée* x *de la manière suivante :*

$$y = \sigma(w \cdot x + b)$$

Définition 4 (Vectorisation d'une fonction)**.** *Soit* f *une fonction de* $\mathbb{R} \mapsto \mathbb{R}$*. On pose pour tout vecteur* x *de* $\mathbb{R}^n$ *:*

$$f(x) = \begin{pmatrix} f(x_1) \\ f(x_2) \\ \vdots \\ f(x_n) \end{pmatrix}$$

On dit que l'on vectorise la fonction f*.*

Achtung ! Par abus de notation, on utilise encore le nom de f pour désigner cette fonction, mais il s'agit bien d'une fonction différente puisqu'elle est à valeurs de $\mathbb{R}^n$ dans $\mathbb{R}^n$. Cet abus nous est néanmoins utile pour définir simplement les couches d'un réseau de neurones.

On considère dans les définitions suivantes une couche d'un réseau de neurones constituée de n neurones. On note w_i le vecteur des poids associé à chaque neurone. On a alors la définition suivante :

Définition 5 (Matrice des poids)**.** *En notant* w_i *les vecteurs des poids de* n *neurones, on définit par blocs la matrice* w *des poids :*

$$w = \begin{pmatrix} {w_1}^T \\ {w_2}^T \\ \vdots \\ {w_n}^T \end{pmatrix}$$

On définit de manière analogue un vecteur des biais.

Définition 6 (Vecteur des biais)**.** *En notant b_i les biais de n neurones, on définit le vecteur des biais d'une couche d'un réseau de neurones par :*

$$b = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix}$$

On a alors la proposition suivante :

Proposition 2.2.1 (Ecriture matricielle d'une couche)**.** *Soit une couche d'un réseau constituée de n neurones et ayant pour matrice des poids w et pour vecteur des biais b. Si les neurones ont tous la même fonction d'activation σ, on peut écrire les sorties des neurones sous la forme d'un vecteur y vérifiant :*

$$y = \sigma(w \cdot x + b)$$

où les y_i correspondent à la sortie du neurones i.

On notera que dans cette proposition, on utilise la version vectorisée de σ. Une écriture équivalente est :

$$\begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix} = \begin{pmatrix} \sigma(w_1 \cdot x + b_1) \\ \sigma(w_2 \cdot x + b_2) \\ \vdots \\ \sigma(w_n \cdot x + b_n) \end{pmatrix}$$

En fait, cette écriture est identique à celle d'un unique neurone ! Elle est également intéressante d'un point de vue computationnelle : au lieu de calculer une à une les sorties des neurones, il est plus efficace de faire un calcul matriciel (les bibliothèques étant en générales optimisées pour cela). Cela permet également d'avoir un code plus simple (avec notamment moins de boucles).

2.3 Apprentissage par la descente de gradient

2.4 L'algorithme de rétro-propagation

3 Reconnaissance via un réseau de neurones simple

Dans ce chapitre, on se propose d'utiliser ce que l'on a vu précédement pour construire un réseau de neurones permettant de répondre à notre problème de classification.

3.1 Construction du réseau de neurones

3.2 Apprentissage

3.3 Résultats

4 Reconnaissance basée sur l'extraction de *features*

Si l'utilisation des données brutes comme entrée des réseaux de neurones permet déjà d'obtenir des résultats encourageant, il est possible de faire mieux en effectuant un pré-traitement des données. Il s'agit d'extraire des données brutes (i.e. les pixels de l'image) des caractéristiques (ou *features* en anglais) qui permettent de décrire l'image dans un format plus adapté pour une machine.

Ces *features* peuvent être regroupés en deux catégories :

1. les *features* statistiques, qui vont s'intéresser à des densité de pixels, des extremums et autres transformées mathématiques ;
2. les *features* structurelles, qui s'intéressent aux traits (strokes), aux courbes, aux nombres de bifurcations, etc. . . , ces dernières sont plus intuitives pour l'humain.

Nous allons dans ce chapitre présenter quelques *features* qui peuvent être utilisées dans le cadre de la reconnaissance de caractères. Puis nous les utiliserons comme entrées de différents algorithmes de *machine learning*.

4.1 Binarisation de l'image

Les images du jeu de données sont en niveaux de gris. C'est à dire que chaque pixel est représenté par un entier entre 0 et 255 (du blanc au noir). Certains algorithmes ne prennent pas en compte le niveau de coloration du pixel et s'intéresse juste au fait que le pixel soit noir ou blanc. Il est donc utile, dans ce cas de binariser l'image (i.e. de passer à la convention 0 pour un pixel ne contenant pas d'encre et 1 pour un pixel en contenant). On choisit donc arbitrairement un seuil à partir duquel on considère que le pixel est colorié.

```
def binarize(img, treshold = 200):
    w = len(img)
    h = len(img[0])
    ret = img.copy()
    for x in range(w):
        for y in range(h):
            ret[y][x] = 1 if img[y][x] >= treshold else 0
    return ret
```

Bien évidemment, le choix du seuil peut avoir un impact sur le calcul des *features* si l'on choisit un seuil très élevé, beaucoup de pixels seront considérés comme vide. A contrario, si le seuil est très faible, on considéra qu'un pixel est colorié dès qu'il y aura un peu d'encre dessus, ce qui peut également poser des problèmes.

4.2 Densités de pixels coloriés

4.3 Nombre de croissements (*crossings*)

On prend deux points à l'extrémité de l'image et on compte le nombre d'alternance entre les groupes de pixels vides et les groupes de pixels contenant de l'encre. Cette méthode n'est à priori pas sensible à l'épaisseur du trait.

4.4 Histogramme des projections

On compte pour chaque ligne (resp. chaque colonne) le nombre de pixels allumés sur la ligne (resp. colonne). En faisant ça sur l'ensemble de l'image, on obtient deux histogrammes. Cette technique peut aussi être utilisé

pour segmenter des lignes et caractères isolés. Cette technique peut être sensible à l'épaisseur du trait. Pour palier ce problème, on peut renormaliser chaque histogramme en divisant chaque valeur par le total.

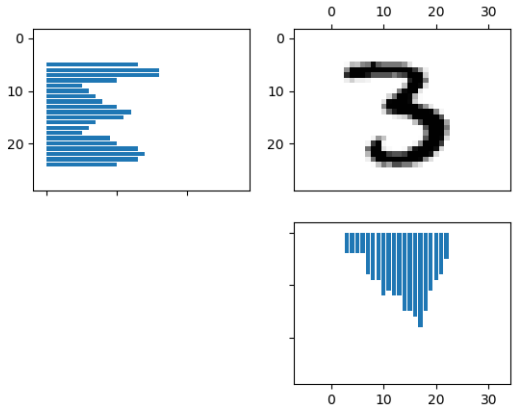

FIGURE 9 – Histogramme des projections du chiffre 3.

4.5 Moments

Dans tout ce qui suit, on note p_{xy} la valeur du pixel (x, y).

On définit le moment d'ordre $(p + q)$ par :

$$m_{pq} = \sum_x \sum_y x^p y^q p_{xy}$$

En pratique, on préférera utiliser des moments centrés (car invariant par translation de l'image).

$$\mu_{pq} = \sum_x \sum_y (x - \overline{x})^p (y - \overline{y})^q p_{xy}$$

avec

$$\overline{x} = \frac{m_{10}}{m_{00}} \qquad \overline{y} = \frac{m_{01}}{m_{00}}$$

4.6 Transformée de Fourier du contour

4.7 Transformée de Fourier de l'image

5 Reconnaissance basée sur les réseaux convolutionnels

6 Comparaison des résultats

TODO : on pourra dans cette partie comparer les résultats avec des méthodes qui ne sont pas basés sur des réseaux de neurones.