

Reconnaissance de caractères à l'aide de réseaux de neurones

Etienne MOUTOT

24 octobre 2014

Résumé

Un réseau de neurone est un modèle permettant de résoudre des problèmes relevant de l'intelligence artificielle. Après en avoir étudié certains aspects théoriques, ce TIPE en propose une application à la reconnaissance de caractères.

Table des matières

Reconnaissance de caractères à l'aide de réseaux de neurones	2
I Objectifs	2
II Outils	2
II.1 Réseaux de neurones	2
II.2 Python	2
III Bibliothèque générale	3
III.1 Description	3
III.2 Algorithme d'apprentissage	3
IV Programme de reconnaissance	4
V Résultats	4
V.1 Optimisations	4
V.2 Performances finales	5
Annexes	I
A Illustrations	I
B Bibliographie	VI
C Contacts	VI

I Objectifs

Le but de ce TIPE a été l'étude des réseaux de neurones formels, un modèle permettant de résoudre des problèmes relevant de l'intelligence artificielle, tels que la classification de données ou l'approximation de fonctions. Il a débouché sur une application pratique : la reconnaissance de caractères.

Il est le fruit d'un travail commun à deux personnes, mon travail ayant essentiellement porté sur l'aspect informatique des réseaux de neurones et leur utilisation dans la reconnaissance de caractères : implémentation de l'algorithme et optimisation de son fonctionnement.

Le problème de reconnaissance de caractère peut être qualifié de problème de classification. En effet, le programme de lecture doit associer une classe (un caractère) à une donnée (une image). L'objectif final du TIPE est donc de fournir un programme prenant en entrée une image représentant un caractère et retournant en sortie le caractère qu'elle représente.

Pour ce TIPE, les caractères à reconnaître sont les chiffres de 0 à 4. L'entrée est une image de 8 pixels par 8 pixels et la sortie une valeur entre 0 et 1 par caractère, qui représentera le degré de ressemblance d'après le programme (1 : c'est le caractère, 0 : ce n'est pas le caractère) (**Figure 1** : Entrée et sortie du programme).

II Outils

II.1 Réseaux de neurones

Les réseaux de neurones sont un assemblage de neurones (**Figure 2**) interconnectés les uns aux autres. L'objectif est ensuite de trouver une architecture qui produise la sortie souhaitée en fonction des entrées. Les principaux paramètres sont la "forme" du réseau et les poids des différentes liaisons.

Dans le cas de notre étude, nous nous limitons au cas du *perceptron multicouche*, qui impose que les neurones soient organisés en couches successives, entièrement reliées à la couche suivante et précédente (**Figure 3**). Le nombre de caractères étant ici assez limité, l'architecture à choisir a été déterminée de manière empirique. Le seul paramètre à déterminer reste donc l'ensemble des poids du réseau, via un algorithme dit d'**apprentissage** (voir partie III.2 : Apprentissage).

II.2 Python

- Python** a été utilisé comme langage de programmation pour plusieurs raisons :
- Il est multi-plate-forme, ce qui permet de faire fonctionner le programme de manière transparente sur Windows, GNU-Linux ou MacOS
 - La souplesse des types permet une conversion simple entre la plupart des types communs, contrairement à des langages fortement typés comme le Caml ou le C++
 - La bibliothèque standard est très fournie
 - Python dispose également d'un grand choix de bibliothèques externes, comme **numpy**, utilisé ici pour effectuer du calcul matriciel

- Enfin, python permet de programmer en *orienté-objet*, qui permet de créer une couche d'abstraction permettant la manipulation de réseaux de neurones en tant que type.

III Bibliothèque générale

Le programme se sépare en deux parties distinctes. La première est une bibliothèque indépendante permettant la manipulation des réseaux de neurones.

III.1 Description

Cette bibliothèque permet la création et la manipulation de réseaux de type perceptron multicouche. Elle est indépendante du problème de reconnaissance de caractères et peut être réutilisée pour n'importe quelle utilisation de ces réseaux.

Elle définit une classe `Reseau` représentant un perceptron multicouche. Cette classe possède des méthodes permettant de :

- Créer un réseau en choisissant le nombre d'entrées, et la composition de chaque couche (nombre de neurones, fonction d'activation) (**Figure 4**)
- Plusieurs algorithmes d'apprentissages basés sur l'algorithme de rétro-propagation du gradient (**Figure 5**)
- Consultation de la sortie du réseau en fonction d'une entrée (**Figure 6**)
- Sauvegarde ou chargement du réseau dans un fichier texte à l'aide du module `pickle` (**Figure 7**)

III.2 Algorithme d'apprentissage

Pour une structure de réseau fixée, il s'agit de trouver les poids correspondant à l'utilisation que l'on souhaite faire du réseau. Pour cela, on dispose d'une base d'apprentissage contenant un échantillon d'images et leur sortie "idéale" (**Figure 8**), et on souhaite que les poids du réseau soient définis de sorte que la sortie soit la plus proche possible de cet idéal pour tous les exemples de la base.

Les poids initiaux sont choisis aléatoirement.

On dispose ensuite de l'algorithme de rétro-propagation du gradient, qui consiste à ajouter un Δw itérativement aux poids w afin de faire diminuer l'erreur, où l'erreur est la différence entre la sortie et la sortie attendue.

$$w' = w + \Delta w(e, \eta)$$

avec $e = ||\text{sortie}_{\text{attendue}} - \text{sortie}_{\text{effective}}||^2$ et η un "petit" pas arbitraire.

La **Figure 9** montre un graphique de l'erreur en fonction du nombre d'itérations de l'algorithme. L'erreur sur la base (courbes pointillés) diminue de façon régulière jusqu'au seuil de 0.1.

Ce qui est remarquable est la diminution de l'erreur sur les caractères inconnus au réseau (non présents dans la base), ce qui permet au réseau d'avoir un bon taux de reconnaissance même sur ces caractères "non prévus".

IV Programme de reconnaissance

La bibliothèque est ensuite utilisée pour créer un réseau de neurones reconnaissant les caractères. Il dispose de 64 entrées (une par pixel) et 5 sorties (une par caractère) (**Figure 3**).

Le programme crée d'abord le réseau correspondant, puis réalise un apprentissage à l'aide d'une base d'environ 150 caractères. C'est cette base qui a limité le nombre de caractères à reconnaître, car il faut un nombre important d'exemples pour que l'apprentissage soit efficace, qui doivent être créés à la main.

Une fois l'apprentissage réalisé, on l'*extrapole* en lui présentant des images non présentes dans la base mais présentant des similarités en espérant que le réseau conserve un "bon" fonctionnement. Pour évaluer le taux de reconnaissance sur des caractères inconnus, un ensemble de 50 images différentes est utilisée.

Après quelques essais à échelle réelle, le temps d'apprentissage du réseau était très long (de l'ordre d'une heure pour atteindre une erreur moyenne à 0.1). Pour améliorer ceci, j'ai fait le choix d'utiliser l'architecture multi-cœur des processeurs récents. Le réseau a donc été scindé en 5 réseaux plus petits (un par chiffre) (**Figure 10**), dont les apprentissages peuvent être réalisés en parallèle, en utilisant plusieurs cœurs au lieu d'un seul. Le temps d'apprentissage est donc divisé par plus de 5, étant donné que chaque réseau est également plus petit que l'unique "gros". Cette modification utilise le module `multiprocessing` de la bibliothèque standard, afin de créer les différents processus et de manipuler les réseaux au sein du type `Queue`, utile pour maîtriser les accès (éventuellement simultanés) des processus aux différents réseaux.

V Résultats

Une fois le programme écrit, son fonctionnement a révélé quelques défaut qu'il a été possible de corriger.

V.1 Optimisations

- Le choix de l'architecture précise du réseau a été faite de manière empirique en essayant les diverses possibilités et en comparant leurs performances
- Le choix du pas η pour l'algorithme de rétro-propagation est difficile car avec un pas trop grand, l'algorithme ne converge pas vers une erreur assez faible, mais avec un pas trop petit il est très lent.
Le compromis retenu a été d'implémenter un pas variable : grand au début, il diminue lorsque la précision devient nécessaire
- Il arrive que dans certains cas, l'erreur ré-augmente et se mette à diverger (pour des problèmes de sur-apprentissage notamment), dans ce cas une "sécurité" a été ajoutée : lorsque l'erreur augmente continuellement pendant un trop grand nombre d'itérations (10 par exemple), l'algorithme s'arrête même si le seuil n'est pas atteint.

V.2 Performances finales

- La **Figure 11** montre des exemples d'entrées et sorties pour des images non présentes dans la base.
- Le temps de convergence total est en moyenne de l'ordre de 5 minutes pour un seuil de 0.1 et 1 minute pour 0.3.
- Pour un seuil de 0.1 et des réseaux de 5 et 1 neurones (**Figure 10**), le taux de reconnaissance est d'environ 80% sur des caractères inconnus.

La documentation complète est disponible à l'adresse suivante :

<http://tipe.etiennemoutot.com/2014/doc/html/>

L'ensemble du code est disponible à l'adresse :

<https://bitbucket.org/coma94/neural-network/src>

Annexes

A Illustrations

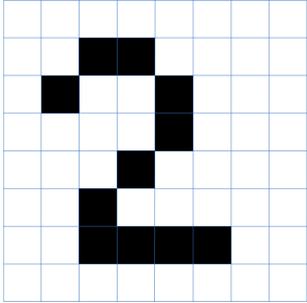
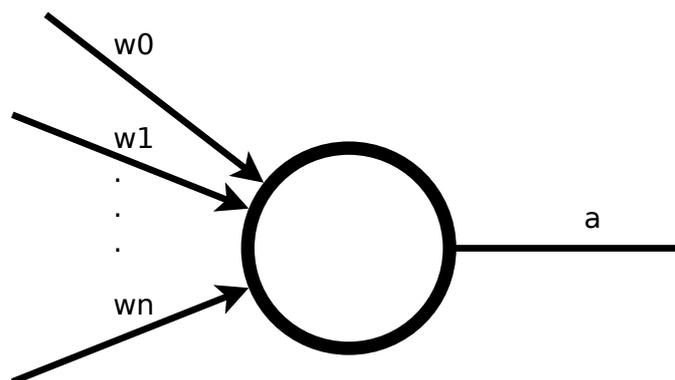
Image (entrée)	Sortie idéale	Sortie
	0 : 0	0 : 0.59
	1 : 0	1 : 0.40
	2 : 1	2 : 0.73
	3 : 0	3 : 0.14
	4 : 0	4 : 0.01

FIGURE 1 – Entrée et sortie du programme



$$a = f \left(\sum_i w_i x_i \right)$$

FIGURE 2 – Neurone et sortie associée

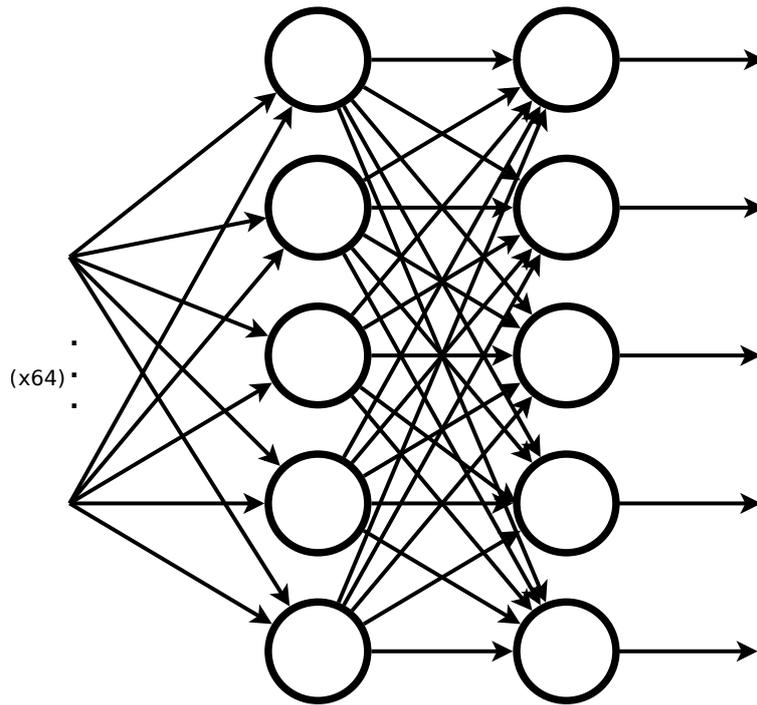


FIGURE 3 – Perceptron multicouche

```
r = p.Reseau(64, [5,1], [p.tanh, p.id])
```

FIGURE 4 – Création d'un réseau r

```
r.learn1(base, ecart=0.1, pas=0.01) # Pas fixe
r.learn12(base, ecart=0.1, div=2) # Pas variable
```

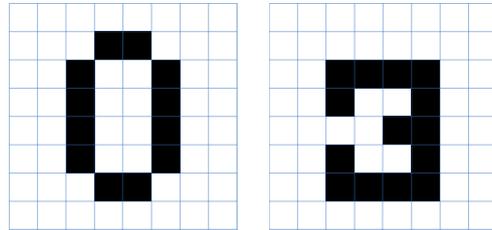
FIGURE 5 – Apprentissage de la base jusqu'à une erreur de 0.1

```
sortie = r.a( entree )
```

FIGURE 6 – Consultation de la sortie du réseau

```
r.save("reseau.txt")  
r.load("reseau.txt")
```

FIGURE 7 – Sauvegarde et chargement d'un réseau



0	:	1.00	0	:	0.00
1	:	0.00	1	:	0.00
2	:	0.00	2	:	0.00
3	:	0.00	3	:	1.00
4	:	0.00	4	:	0.00

FIGURE 8 – Éléments de la base d'apprentissage

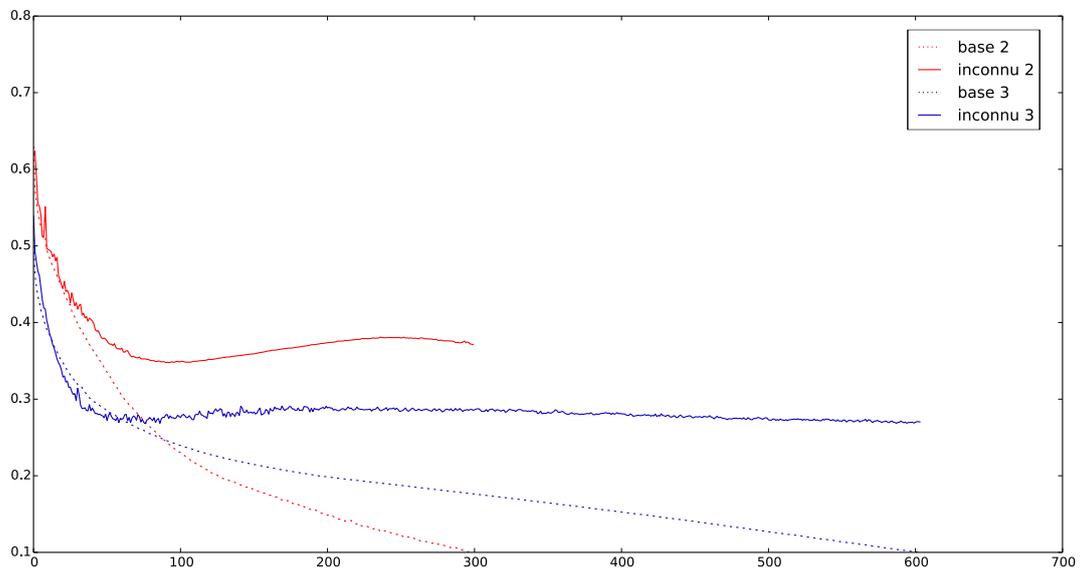


FIGURE 9 – Graphique de l'erreur au cours des itérations de l'algorithme d'apprentissage

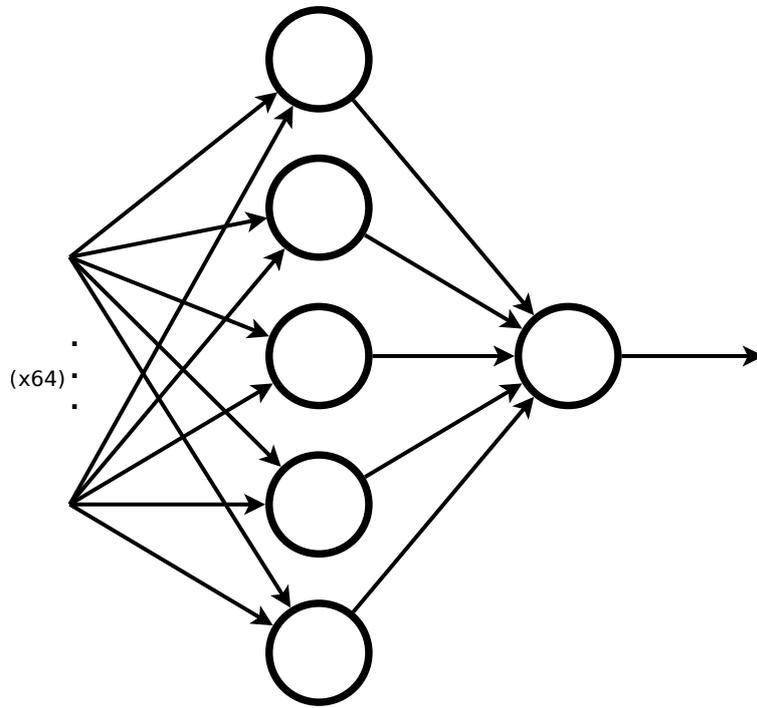


FIGURE 10 – Nouveau réseau (×5)

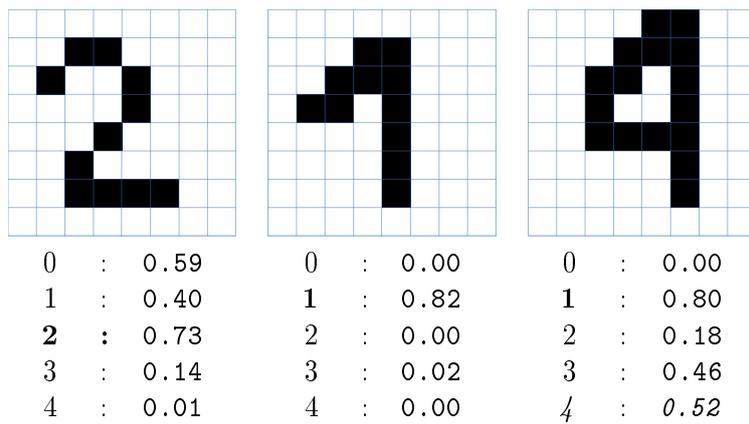


FIGURE 11 – Exemples d'entrées et sorties

B Bibliographie

- "Zythom", *Réseaux de neurones 1*, mars 2013.
<http://zythom.blogspot.fr/2013/03/reseaux-de-neurones-1.html>
- Marc Parizeau, *Réseaux de neurones*, 2004.
- Claude Touzet *Les réseaux de neurones artificiels - Introduction au connexionnisme - Cours, exercices et travaux pratiques*, juillet 1992.
- G. Dreyfus *Les réseaux de neurones*, septembre 1998.
http://www.neurones.espci.fr/Articles_PS/GAMI.pdf

C Contacts

- "Zythom", ancien chercheur dans le domaine des réseaux de neurones formels.