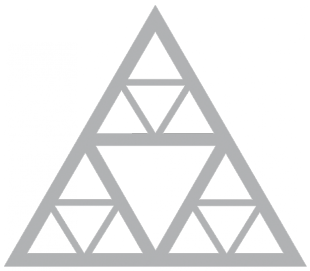

École des Ponts

ParisTech

Département IMI

Février 2018 - Mai 2018

Projet de Deep Learning

Reconnaissance de Caractères dans des Manuscrits

Guillaume Desforges et Théo Viel,

Encadrés par Mathieu Aubry, Imagine, LIGM, École des Ponts

Table des matières

Première partie
Introduction

L'objectif de ce projet est de tester différentes méthodes de reconnaissance de texte pour des manuscrits anciens écrits en vieux français. Ces manuscrits sont souvent très difficiles à lire, même pour des initiés. Il s'agit donc d'automatiser une tâche déjà dure pour l'homme.

Le problème de reconnaissance de caractères (OCR) est une application classique des réseaux de neurones. C'est donc ce genre de technique que nous explorerons.

Nous commencerons par utiliser des moteurs de reconnaissance existants, à savoir `ocropy` et `Tesseract`. La deuxième partie du projet consistera soit à essayer d'améliorer le fonctionnement de l'un d'eux, soit à implémenter notre propre réseau de neurones.

A Base théoriques et état de l'art

Insert ~~meme~~ relevant info here.

B Reconnaissance avec ocropy

B.1 Présentation

GitHub du projet ocropy

> OCRopus is a collection of document analysis programs, not a turn-key OCR system. In order to apply it to your documents, you may need to do some image preprocessing, and possibly also train new models.
>
> In addition to the recognition scripts themselves, there are a number of scripts for ground truth editing and correction, measuring error rates, determining confusion matrices, etc. OCRopus commands will generally print a stack trace along with an error message ; this is not generally indicative of a problem (in a future release, we'll suppress the stack trace by default since it seems to confuse too many users).

B.2 Entraînement

Le principe d'entraînement, adapté aux manuscrits, est le suivant :

1. On récupère un ensemble de photos de pages d'un livre ;
2. On les binarise, c'est-à-dire que l'on met tous les pixels soit à 1 soit à 0 ;
3. On en extrait les lignes ;

4. On annote un maximum données ;
5. On sélectionne une partie des données annotées pour l'entraînement (90% des données annotées par exemple), et on garde le reste pour la validation ;
6. On entraîne le réseau de neurones sur les données d'entraînement ;
7. On valide sur les données de validation en regardant l'erreur.

B.3 Premiers essais

B.3.1 Préparation des données

Afin de nous familiariser avec cet outil, nous avons commencé par binariser et extraire des lignes d'un manuscrit pris sur le site http ://www.e-codices.unifr.ch nous-même. Ce premier test nous a permis de bien comprendre les premiers enjeux. En effet, arrivés à l'étape d'annotation des données, nous avons été confrontés à la difficulté que représente la traduction de ces manuscrits. Le processus est fastidieux puisque les mots mais aussi les caractères ont changé depuis.

Par la suite, nous avons abandonné ces données au profit de données annotées nous ayant été mises à disposition par l'ENC.

Nous avons ensuite travaillé sur un manuscrit de la *Chanson d'Otinel* écrit en anglo-normand au XIIIème siècle.

Il est d'une longueur de 46 pages, d'environ 30 lignes chacune. Nous avons pris environ 1300 lignes pour l'entraînement et 100 pour la validation.

B.3.2 Commandes basiques

La commande naïve à exécuter pour entraîner un modèle est :
`ocropus-rtrain -o <ModelName> train/*/*.bin.png`
On peut également visualiser les étapes avec la commande :
`ocropus-rtrain -o <ModelName> -d 1 train/*/*.bin.png`

A chaque itération, on a un output de la forme suivante :

```
4004 26.45 (493, 48) train/0033/010016.bin.png
   TRU: u'e l g\u0363nt \u1ebdfern v g\u0131\u017fent l\u0131 larun'
   ALN: u'e l g\u0363nt \u1ebdfern v g\u0131~ent l\u0131 larun'
   OUT: u'e lgc ern o g\u0131~ent l\u0131 larun'
```

Figure 1 – Output d'une itération de l'entraînement

Sur la première ligne se trouve d'abord le numéro de l'itération, puis quelques propriétés de l'image étudiée. L'information se contenant dans le lignes suivantes :
- `TRU` est la solution recherchée
- `ALN` est une version alignée de `TRU`
- `OUT` est le résultat trouvé par le réseau de neurones

Un réseau de neurones idéal aurait les 3 lignes identiques.

Afin de lancer la validation d'un modèle et d'avoir le pourcentage d'erreur, on utilise les commandes suivantes :
- `ocropus-rpred -m <ModelName> test/*/*.bin.png`
- `ocropus-errs test/*/*.gt.txt` permet d'évaluer la précision du modèle.
- `ocropus-econf test/*/*.gt.txt` donne les erreurs les plus fréquentes. On peut ajouter `-C2` pour une contextualisation des erreurs.

Sur ce premier entraînement sans préparations, on obtient une erreur de l'ordre de 27% au bout de 12 000 itérations ce qui est extrêmement mauvais. En effet, l'ensemble des caractères sortant de l'ordinaire ne sont pas reconnus.

B.3.3 Entraînement avec les caractères spéciaux

Afin d'intégrer les caractères spéciaux à la résolution, la méthode est de les ajouter à la liste des caractères, stockée dans `char.py`. C'est ce que nous avons fait, mais cela n'a rien changé. Nous avons compris par la suite que nos modifications effectuées étaient correctes, mais qu'il faut relancer la commande d'installation pour les prendre en compte ...
`sudo python setup.py install`

Ceci nous a permis d'obtenir une première courbe d'erreur satisfaisante. Le tracé est réalisé en analysant les modèles sauvegardés toutes les 1000 itérations, et en inscrivant les résultats dans des documents `.txt`. Un *parsing* avec python permet de récupérer les informations utiles de ceux-ci. Le temps nécessaire pour effectuer 1000 itérations est d'environ 11 minutes sur l'ordinateur que nous avons utilisé pour notre installation `ocropy`. C'est un portable MSI doté d'une NVIDIA GeForce GTX 1060, le GPU étant le plus important pour entraîner un réseau de neurones.

Le meilleur modèle est obtenu au bout de 50000 itérations, et atteint une erreur de 9.811%. Ces résultats sont corrects aux vues de la complexité des données, mais nous pensons pouvoir faire mieux avec un peu de pré-traitement et en modifiant certains paramètres.

Statistics of the model	
errors	329
missing	35
total	3547
err	9.275 %
errnomiss	8.289 %

Table 1 – Output de la commande `ocropus-errs`

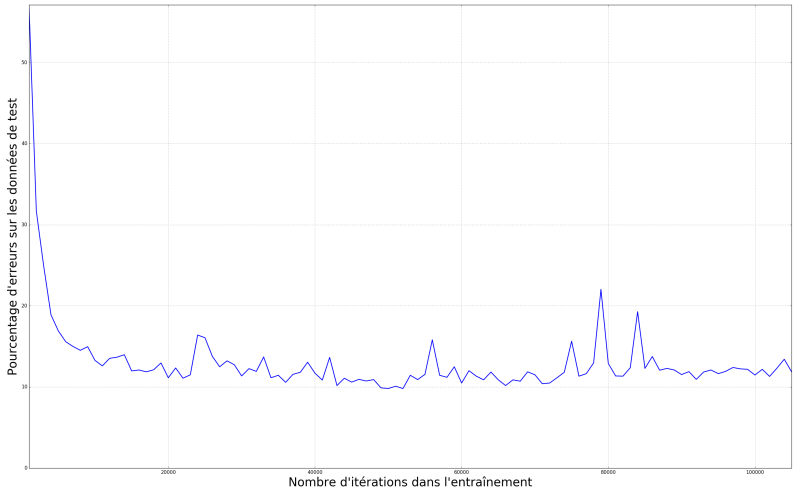

Figure 2 – Évolution de l'erreur sur les données comportant des caractères spéciaux.

B.4 Amélioration des modèles

B.4.1 Premières normalisation

Nous avons donc opté pour une nouvelle méthode, consistant à repérer manuellement les caractères utilisés avec les fichiers textes contenant l'information de référence. On utilise ajoute à la commande d'entraînement la commande suivante :

`-c train/*/*.gt.txt test/*/*.gt.txt`

Ceci indiquant qu'on va chercher les symboles dans les *ground truth* disponibles. Les trois commandes décrites précédemment nous permettent de remarquer qu'au bout de 5000 itérations, l'erreur est descendue à 20% seulement. La matrice de confusion nous a permis de voir que le caractère ſ n'était pas reconnu. Ceci étant du à une normalisation de l'unicode effectuée par la l'appel `-c`.
Pour résoudre ce problème nous avons implémenté un script `Python` remplaçant les caractères de nos données par leur normalisation *NFC*, et ainsi assurer la symétrie entre les données pouvant être apprises et la solution. Cette normalisation est utilisée par `ocropy` pour traiter les données lors de l'appel de la fonction `normalize_text` du module `common.py`. Cependant, la normalisation du caractère ſ étant le *s*, déjà présent dans les données, nous l'avons remplacé par un *Z*.

B.4.2 Analyse des premiers modèles normalisés

La figure 3 montre les améliorations obtenues grâce à ces premières normalisations.

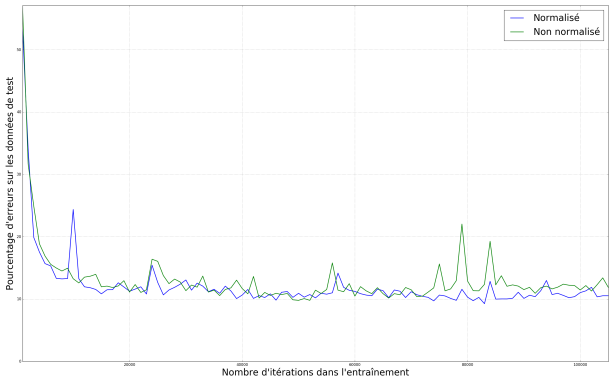

Figure 3 – Évolution de l'erreur sur les données normalisées.

La meilleure précision est obtenue pour 83000 itérations, on s'attarde donc un peu plus sur ce modèle particulier pour comprendre les défauts de l'apprentissage. La table 2 résume les statistiques importantes de ce modèle.

Statistics of the model	
errors	329
missing	35
total	3547
err	9.275 %
errnomiss	8.289 %

Table 2 – Résumé de la commande `ocropus-errs`

Ce qui nous intéresse plus est contenu dans la table 3 et explique les défauts du réseau de neurones.
On distingue 3 types d'erreurs :

1. Les erreurs sur les espaces ;
2. Les erreurs sur les caractères superscripts ;
3. Les erreurs sur les lettres semblables.

Le premier peut sans doute être éliminé avec un peu de traitement des données, le second également mais causerait sans doute de la perte de sens. Le troisième est plus caractéristique de l'écriture puisque les lettres *u, n, m, i et l* se ressemblent souvent.

Most frequent confusions					
Without context			With context		
#	output	truth	#	ouput	truth
28		_	2	a δ u	a_δ u
12	_	´	2	uɪ_t	uɪ´ t
10	_		2	ɪez	ɪe-
6	_	a	1	e u la lt	e n la_lt
4	u	n	1	uɪ_ b	uí b
4	ɪ	_	1	Ztɪ	Ztr
4	t	_	1	aɪ_nZ	aínZ
4	ɪ	r	1	eg ler	egɪ_er
3	ɪ	u	1	O_gɪ	O gɪ
3	z	_	1	e uZ	e_uZ

Table 3 – Résumé de la commande `ocropus-econf`

B.4.3 Amélioration de la normalisation

C Modification des paramètres d'entraînement

C.1 Commandes utiles

Pour modifier l'apprentissage, on peut ajouter une des lignes suivantes à la fin de la commande pour lancer l'entraînement, afin d'en modifier les propriétés.
`--ntrain <Number>` permet de modifier le nombre d'itérations réalisées dans l'entraînement. Plus ce nombre est grand, meilleur sera le résultat du réseau. Par défaut il vaut 10^6, mais en général nous nous arrêtons vers 10^5 itérations.
`--savefreq <Number>` permet de dire le nombre d'itérations entre chaque sauvegarde du modèle. Ceci nous permettra dans la suite de tracer les courbes d'erreur. Le paramètre par défaut est 1000 et est un bon compromis entre la précision des courbes et le temps passé à tester les modèles.
`--lrate <Number>` permet de modifier la vitesse d'apprentissage.
(*learning rate*). Un de nos premiers objectifs sera de trouver une bonne valeur de ce paramètre.
`--hiddensize <Number>` permet de modifier un des paramètres de la taille des matrices de notre réseau de neurones. Nous nous intéresserons également à son optimisation.

Dans les sous-sections suivantes, nous travaillons avec les données obtenues avec la première normalisation.

C.2 Modification de la *learning rate*

On rappelle que la *learning rate*, vitesse d'apprentissage, ou vitesse d'entraînement, est un hyper-paramètre qui caractérise le pas dans la descente du gradient stochastique présente dans le réseau de neurone. Les courbes figure 4

comparent les modèles pour la valeur par défaut de la vitesse d'apprentissage, 10^{-3}, et pour une valeur de 10^{-4}.

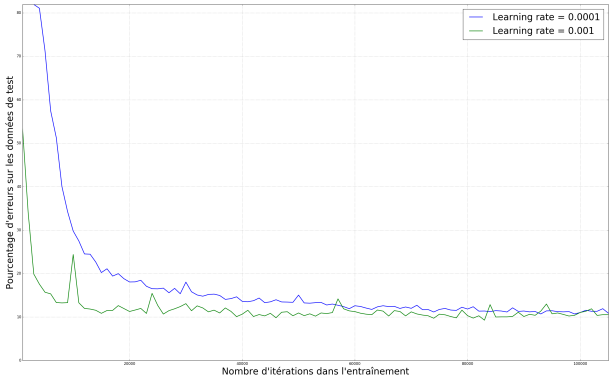

Figure 4 – Évolution de l'erreur pour deux vitesses d'apprentissage différentes.

On remarque que la courbe au *learning rate* plus faible est plus lente à converger, mais plus régulière. En raison de cette stabilité, le meilleur modèle, obtenu pour 133000 itérations a un taux d'erreur de 9.949% pour la *learning rate* la plus faible, contre 9.275% au bout de 83000 itérations dans l'autre cas. C'est pourquoi nous garderons une *learning rate* de 10^{-3} dans la suite.

C.3 Modification de la taille du réseau

Le paramètre sur lequel on joue ici est le paramètre définissant le nombre de *hidden state*. Ce sont les variables que le *LSTM* utilise pour garder une trace du temps. C'est la partie récurrente du réseau. L'influence de ce paramètre est présentée figure 5.

On remarque que les résultats sont meilleurs quand on augmente ce paramètre. La convergence est un peu plus lente en nombre d'itérations, cependant, nous sommes passé de 12 à 18 minutes de temps de calcul nécessaire pour effectuer 1000 itérations. Le meilleur modèle que nous avons obtenu a une erreur de 7.976%, atteinte pour 95000 itérations, soit plus d'un jour d'entraînement.

D Commandes bash utilisées

```
ocropus-rpred -m test3-00005000.pyrnn test/*/*.bin.png
ocropus-errs test/*/*.gt.txt

ocropus-rtrain -o <modelname> <training-dir>/*/*.bin
ocropus-rtrain -o myModel -d 1 train/*/*.bin.png
```

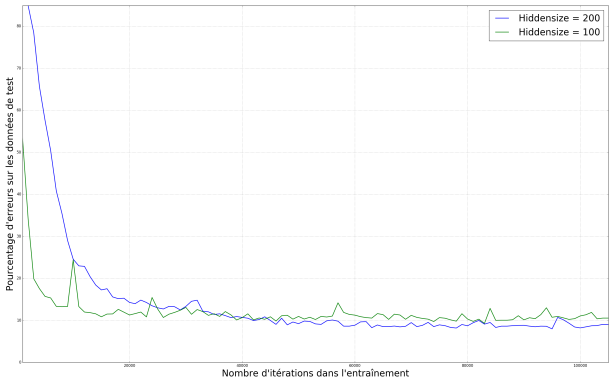

Figure 5 – Évolution de l'erreur pour deux tailles de réseau différentes.

```
for i in test8/*.pyrnn.gz; do
    echo "\$i" >> results.txt
    ocropus-rpred -m \$i test/*/*.bin.png
    ocropus-errs test/*/*.gt.txt >> results.txt
    echo "\$i" >> index.txt
done

sudo python setup.py install

ocropus-rtrain -o test8 train2/*/*.bin.png --load test8-00086000.pyrnn

ocropus-rtrain -c train/*/*.gt.txt test/*/*.gt.txt -o test7 train/*/*.bi

ocropus-rpred -m test8-00050000.pyrnn test/*/*.bin.png
```