

Deep Learning, Text Mining et Traitement du Langage Naturel

Rapport de projet d'option informatique

Damien DOUTEAUX, Vincent HOCQUEMILLER et Louis REDONNET

Mars
30
2017



Table des matières

■ Introduction	4
■ 1 • État de l'art.	5
1.1 Inférence.	5
1.1.1 Généralités.	5
1.1.2 Structures de données	5
1.1.3 Exemples de résultats	5
1.2 Question-réponse	5
1.2.1 Généralités.	6
1.2.2 Utilisation de LSTM	6
1.2.3 Utilisation de MemNN	6
1.2.4 Utilisation de DMN	6
1.3 Traduction	7
1.3.1 Généralités.	7
1.3.2 Méthodes à base de règles	7
1.3.3 Méthodes statistiques	7
1.3.4 Méthodes neuronales	8
1.4 Reconnaissance d'auteur.	8
1.4.1 Généralités.	9
1.4.2 Bases de données disponibles.	9
1.4.3 Techniques utilisées	9
1.5 Analyse de sentiments	10
1.5.1 Arbres récursifs	10
1.5.2 Vecteurs paragraphes	11
1.5.3 Réseaux LSTM	12
1.6 Choix d'un sujet	12
■ 2 • Base de données	13
2.1 Rencontre avec un PAR	13
2.2 Les bases de données utilisée.	13
2.2.1 Recherche de bases de données en analyse de sentiment	13
2.2.2 Choisir quelle base utiliser	14
2.3 Mise en forme des données	14
2.3.1 Des formats divers	15
2.3.2 Harmoniser les bases de données	17
2.3.3 Résultats de l'harmonisation.	18
■ 3 • Étude pratique d'un modèle	20
3.1 Motivation du choix du modèle	20
3.2 Les hyperparamètres du modèle	20
3.2.1 Hyperparamètres basiques	20
3.2.2 Hyperparamètres les plus importants	21
3.3 Représentation des mots.	21
3.3.1 Bag of Word	21
3.3.2 Les N-grams	22
3.3.3 Représentations denses.	22
3.4 Enseignements de l'apprentissage	23
3.4.1 Valeurs retenues	23
3.4.2 Méthode de choix des valeurs	23
3.4.3 Remarque sur la qualité du modèle obtenu	24

3.5	Influence des paramètres	24
3.5.1	Influence de <i>NO_CUT</i>	24
3.5.2	Influence de <i>VOCAB_SIZE</i>	25
3.5.3	Influence de <i>DROPOUT</i>	25
3.6	Détails sur la préparation des données	26
3.7	Classifications binaires ou multi-variée	28

■	Conclusion	30
---	----------------------	----

■	Références	32
---	----------------------	----

Liste des figures

1	Structure d'un modèle DMN	7
2	Utilisation de l'état interne d'une cellule LSTM pour prédire des séquences à partir de séquences [12]	8
3	Amélioration de Google Translate en utilisant des méthodes neuronales	9
4	Comparaison des différentes méthodes de traduction pour les cas « usuels »	9
5	Exemple d'arbres syntaxique pour l'analyse de sentiments	10
6	Exemples d'entraînement pour labelliser un arbre syntaxique	11
7	Utilisateur des vecteurs paragraphes en prédiction	11
8	Fonctionnement global d'un LSTM pour propager l'information retenue	12
9	Exemple de données de la base de données Amazon	15
10	Exemple de données issues de la bases de données de LMDB	16
11	Exemple de données de la base de données <i>Sentiment Analysis Dataset</i>	17
12	Un exemple de fichier de données après harmonisation des bases	18
13	Représentation de l'idée de la méthode Bag of Word	22
14	Représentation d'une phrase à l'aide de N-grams	22
15	Résultats obtenus sur l'influence de <i>NO_CUT</i>	24
16	Résultats obtenus sur l'influence de <i>NO_CUT</i> pour les grandeurs les plus importantes	25
17	Résultats obtenus sur l'influence de <i>VOCAB_SIZE</i>	26
18	Influence de <i>DROPOUT</i> pour différents réseaux	27
19	Observation de la fonction de coût en fonction de <i>DROPOUT</i> sur différents réseaux	28
20	Répartition des notes pour une des bases d'Amazon	29

Liste des tables

1	Bases de données considérées pour le projet	13
2	Comparatif des intérêts et des problèmes techniques des différentes bases de données repérées	14

Introduction

Ce compte-rendu vous présente un résumé des principaux résultats obtenus lors de ce projet d'option informatique. Le sujet du projet était d'étudier les applications possibles en *machine learning* se trouvant à la jonction entre le *text mining*, le *natural language processing* ainsi que le *deep learning*.

Pour se faire, ce projet s'est axé autour de quatre points principaux :

- ◉ *Étude de termes et applications* Cette première étape a consisté à s'approprier les termes de l'énoncé, et à regarder ensuite plus spécifiquement les différentes applications entre le *deep learning* et le *NLP*.
Si les différents termes du sujet ont été précisément détaillés lors des présentations, nous nous attacherons plus dans ce rapport sur les différentes applications, et en particulier les résultats et méthodes utilisées.
- ◉ *Construction d'une base de données* Après avoir retenu un sujet en particulier parmi ceux précédemment vus, nous avons constitué une base de données sur ce dernier. L'objectif de cette base était double, d'une part que nous puissions l'utiliser pour nos applications, mais également qu'elle soit accessible à de futurs projets.
- ◉ *Test d'un réseau LSTM* Comme nous l'expliquerons dans ce rapport, nous nous sommes attardés sur l'utilisation d'un réseau LSTM. Nous avons pour ce dernier utilisé tout d'abord un exemple jouet (présenté à l'oral), mais ce rapport va s'attarder plus sur le paramétrage de ce réseau et sur une application avec nos « vraies données ».

Nous dresserons enfin un bilan et des perspectives pour ce projet.

Il est à noter que ce rapport est vu comme un complément aux différentes présentations qui ont été réalisées (et qui sont toutes disponibles sur le répertoire Git du projet). Ainsi, certains éléments fortement détaillés dans ces présentations comme la définition des termes génériques, les explications complètes des LSTMs ou encore la présentation de l'exemple jouet ne seront pas nécessairement re-traités en détails dans ce rapport qui essayera d'aller plus loin. Au besoin, des liens seront faits vers différentes publications pour sustenter le lecteur désireux d'approfondir certains points.

Pour terminer, vous pouvez retrouver toutes nos ressources (code, présentations, bibliographie, données,...) au lien suivant, n'hésitez pas à consulter sa page d'accueil !

https://github.com/DDouteaux/Projet_option_info

1 • État de l'art

1.1. Inférence

1.1.1. Généralités

Cette application consiste à tirer des relations entre les phrases d'un corpus. Par exemple, est-ce qu'une phrase est la contradiction de la précédente. Ce concept est fondamental pour la théorie des bases de données. Plus précisément, cela consiste à déduire des contraintes qui sont sémantiquement impliquées par d'autres, grâce à l'analyse syntaxique (grammaticale) et sémantique.

Ainsi, en NLP l'inférence (NLI pour *Natural Language Inference*) consiste généralement à déterminer si une phrase implique ou contredit une autre phrase.

Cette problématique a été particulièrement travaillée à Stanford, où on retrouve de nombreux travaux [1,2].

D'autres tâches de compréhension de texte sont également liées au NLI. En effet, l'idée est d'utiliser des hypothèses de la forme *entité1* \rightarrow *relation* \rightarrow *entité2* (où *entité1* et *entité2* sont deux éléments d'un corpus), cependant, il est possible d'aller plus loin en ne considérant pas nécessairement que les entités sont des mots, mais des phrases du corpus.

La NLI est donc une étape clé de la compréhension d'un texte. Il s'agit d'un domaine très vaste aux applications variées, et ses frontières peuvent être floues. Des applications possibles du NLI sont, entre autres :

- Compréhension de contenu et dans une certaine mesure le Question Answering (voir Section 1.2).
- Résumé de texte
- Traduction automatique

Nous allons dans la suite de cette partie aborder quelques réalisations et aspects techniques sur cette application.

1.1.2. Structures de données

Pour le NLI plusieurs structures de données sont possibles. On retiendra les principales suivantes :

- 1. Des tableaux de phrases à 300 dimensions, 3 couches de traitement.
- 2. Arbres récursifs.
- 3. Tableaux d'imbrications GloVe à 100 dimensions (plus de détails vous seront fournis dans la suite du rapport).

La représentation GloVe, publiée par Stanford, est utilisée par de nombreux travaux comme représentation initiale des mots (comme par exemple pour [1,3]). Les tailles des vecteurs des mots peuvent varier, et vont généralement de 50 à 300 dimensions.

1.1.3. Exemples de résultats

Sur une étude de prédiction de relations entre des phrases (contradiction, implication), les résultats atteignent d'une précision comprise entre 65 et 80% [4].

Cependant, les différentes publications indiquent que l'utilisation de modèles à base de neurones (LSTM ou réseaux avec mémoire) fournit de meilleurs résultats. Cependant, ces modèles nécessitent plus de puissance de calcul et deviennent relativement coûteuses [3].

1.2. Question-réponse

1.2.1. Généralités

Comme nous l'avons déjà partiellement évoqué, le Q&A (procédures de réponses à des questions) est une des applications de l'inférence. Le Q&A consiste d'une part, en la compréhension de questions grâce à l'analyse syntaxique et sémantique d'une question ou requête, et d'autre part en la recherche d'une réponse.

Il s'agit généralement pour cette dernière tâche d'explorer un nombre important de documents en rapport avec la question, de les traiter pour trouver une réponse, et de fournir la réponse qui paraît la plus adéquate. La réponse est alors générée en langage naturel. Ce problème est très ambitieux, mais ses avancées dans la recherche ont des intérêts non négligeables dans le Data Mining et ses nombreuses applications.

Un exemple d'application du Q&A serait de pouvoir formuler une question à moteur de recherche et que ce dernier agrège une réponse en fonctions des documents sur le web. Cette application est à différencier d'une simple requête à une moteur de recherche à partir de mots clés, puisque l'idée serait ici de prendre en entrée une question *rédigée* et d'y répondre par des phrases elles aussi *rédigées*.

Ces question a notamment été traité dans des projets de Stanford [5]. La plupart de ces travaux, cependant, se limitent à la tâche consistant à répondre en une phrase ou un mot à des questions simples. Ils tentent généralement de comparer la performance de différents modèles, ou bien d'améliorer l'efficacité de l'un d'entre eux.

Dans la suite de cette section, nous allons nous intéresser aux différentes technologies utilisables pour cette application.

1.2.2. Utilisation de LSTM

Plusieurs méthodes sont utilisées pour traiter les problématiques de Q&A en deep learning.

Une des méthodes les plus couramment utilisées est connue sous le nom de LSTM. Dans la mesure où un exemple de LSTM sera fourni plus loin dans ce rapport, nous ne détaillons pas plus cette méthode ici.

1.2.3. Utilisation de MemNN

La méthode MemNN (*Memory Networks Implementation*) [6] utilise des réseaux de neurones comprenant des couches de mémoire. Ces dernières stockent des faits lus dans un set de données. Pour extraire les informations pertinentes à la requête, un système de poids permet de voir la similarité entre chaque élément et la requête.

Dans le détail, ces systèmes fonctionnent en trois étapes :

- ⊙ **Représentation mémoire** On convertit le vecteur en entrée grâce à une matrice interne pour le stocker sous la forme d'un état interne. Le même processus est réalisé pour les requête qui sont transformées pour être dans le même « langage » que l'état interne.
- ⊙ **Extraction de la mémoire** Des poids sont appliqués sur les mesure de similarités entre une requête et les informations stockées qui seront utilisés en prédiction.
- ⊙ **Prédiction** Les poids calculés ainsi que l'état interne de la requête sont associés et filtré en utilisant des couches neuronales linéaires et/ou *softmax* pour obtenir une sortie.

1.2.4. Utilisation de DMN

Un DMN (*Dynamic Memory Network*) fonctionne en traitant les questions et les inputs en utilisant des mémoires intermittentes (épisodiques) pour déterminer les réponses aux questions [6].

Le modèle DMN consiste en 4 modules, comme indiqué Figure 1.

Les quatre blocs présents à la Figure 1 sont alors les suivants :

- ⊙ **Module d'entrée** Il convertit l'entrée en une suite de représentations d'états cachés c_t en utilisant un RNN (réseau de neurones récurrent). La séquence d'états cachés consiste soit en un état par mot pour une représentation mot à mot; soit en un état par phrase dans le cas d'une représentation phrase à phrase.
- ⊙ **Module de question** Il produit l'état final caché $q = q_{T_Q}$ en utilisant lui aussi un RNN.

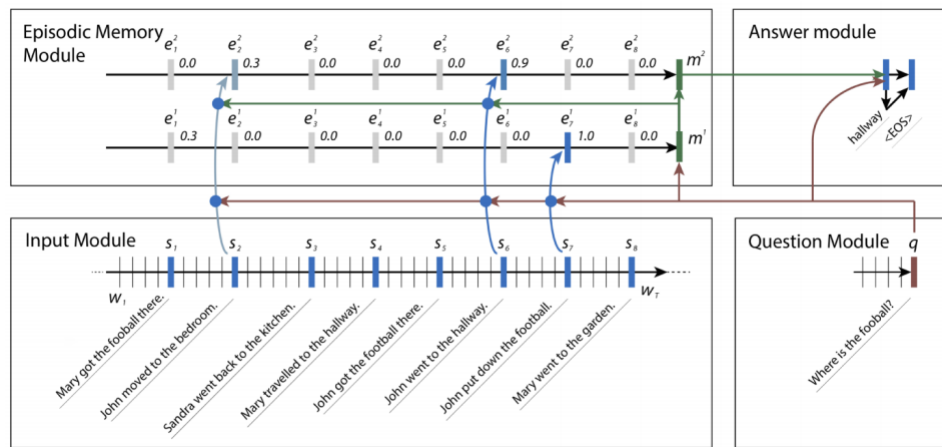


FIGURE 1 • Structure d'un modèle DMN

- Module de mémoire épisodique** Il est composé d'un épisode e_i et d'une mémoire m_i , où i désigne l'itération. Pendant chaque itération, un mécanisme génère des portes g_t pour chaque entrée c_t basée sur sa similarité avec la requête q et sa mémoire précédente $m_i - 1$. Ensuite, un RNN est utilisé pour produire un nouvel état (épisode) e_i .
 Quand à la mémoire, elle est calculée grâce à un GRU (*Gated Recurrent Unit*, une variation d'un modèle LSTM). On note ainsi $m_i = \text{GRU}(e_i, m_i - 1)$, avec $m_0 = q$.
- Module de réponse** Il utilise un module vecteur comme état initial caché. Il génère ensuite sa réponse en utilisant un GRU qu'il applique sur son état initial, la requête q et la prédiction précédente.

1.3. Traduction

1.3.1. Généralités

La traduction automatique consiste à obtenir (sans intervention humaine) une traduction dans une langue de destination à partir d'un texte dans une langue de départ. Ces possibilités ont été étudiées et mises en pratiques dès les années 1950, bien que ces systèmes n'ont pas toujours utilisé des solutions de Deep Learning.

Les solutions mises en œuvre sont aujourd'hui très diverses, et nous allons ici esquisser un tour d'horizon de celles utilisés couramment.

1.3.2. Méthodes à base de règles

Comme un certain nombre de règles strictes sont nécessaires, et structurent la manière dont la traduction humaine est effectuée, la première approche a d'abord été celle consistant à donner à la machine, de manière explicite, des règles de traduction [7].

La méthode la plus simple consiste à remplacer, dans la phrase à traduire, chaque mot par son équivalent dans le langage de destination, à l'aide d'un simple dictionnaire de traduction mot-à-mot.

Cette approche est limitée car elle considère un mot indépendamment de son contexte. Une amélioration est donc de considérer la phrase dans son ensemble, et de donner à la machine une grande quantité de règles syntaxiques, sémantiques, etc... pour établir la traduction au cas par cas.

Peu d'exemples sont alors nécessaires, mais de l'expertise humaine est alors indispensables (celle d'un linguiste en particulier). On appelle ce genre de traducteurs automatiques des systèmes experts.

1.3.3. Méthodes statistiques

Les méthodes précédentes se heurtent à la limite de la modélisation du langage mouvant, permettant des subtilités et variations infinies, par un corpus de règles fixes. Dans les années 90, suite à l'apparition d'Internet, et à la mise à disposition d'ensembles de traductions réalisées par l'homme de taille conséquente

(textes du Parlement Européen par exemple [8]), de nouvelles méthodes fondées sur les statistiques ont alors pu voir le jour [9].

Ces méthodes nécessitent une très grande quantité de textes déjà traduits. La traduction peut se faire mot à mot, ou par groupes de mots (syntaxiques ou de circonstance). On sélectionne la traduction la plus probable au regard du corpus, en prenant en compte les mots environnant la traduction. Il s'agit de l'approche qui a été utilisée par Google Traduction ces 10 dernières années [10].

1.3.4. Méthodes neuronales

La traduction par méthode neuronale (NMT pour *Neural Machine Translation*) qui nous intéresse dans le cadre de ce projet est apparue récemment, et a tout de suite donné des résultats très encourageants [11].

Elle se base sur les LSTM (*Long Short Term Memory Networks*), qui est un cas particulier des réseaux de neurones récurrents [12]. Pour ces réseaux, le réseau ne prédit pas seulement une sortie à partir d'une entrée, indépendamment de toutes les entrées précédentes, mais il peut prédire une séquence de sorties à partir d'une séquence d'entrées (séquences de tailles arbitraires), en stockant et adaptant, à chaque entrée, un état interne. Plus de détails sur ces réseaux vous seront fournis à la Section 1.5.3, vous pouvez cependant observer à la Figure 2 le fonctionnement d'une cellule LSTM (la manière de schématiser la propagation de la sortie sur les entrées).

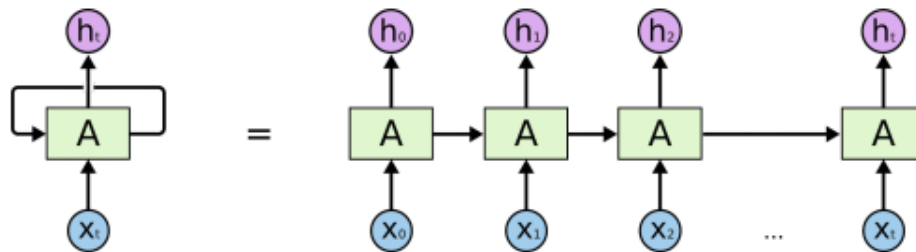


FIGURE 2 • Utilisation de l'état interne d'une cellule LSTM pour prédire des séquences à partir de séquences [12]

Cette méthode, appelée Sequence2Sequence, s'est révélée très adaptée à la traduction automatique, entre autres champs. Il suffit de donner la phrase de la langue de départ, et de donner en cible à prédire, la traduction de la phrase dans la langue d'arrivée. En pratique, les systèmes utilisés ont surtout été des systèmes « encoder-decoder » : on code la séquence de la phrase à traduire par une représentation abstraite intermédiaire grâce à un premier réseau, et on tente de prédire la traduction à partir de la représentation intermédiaire par un deuxième réseau.

Cette approche offre de nombreux avantages : le réseau n'apprend aucune règle et peut, avec une quantité suffisante de données d'apprentissage, apprendre tout seul la traduction d'un langage à un autre. De plus, la traduction est plus cohérente, et prend en compte un contexte plus large, et peut inférer, choses qui n'étaient pas possible avec les méthodes statistiques, ou les exemples deviennent trop rares à partir d'un certain nombre de mots.

En novembre 2015, Google a mis cette nouvelle méthode en production pour Google Traduction [13]. Un exemple d'amélioration de traduction grâce à cette méthode vous est fourni à la Figure 3.

Cette méthode n'est cependant pas adaptée pour tous les passages d'une langue à une autre, car elle nécessite une grande quantité de données ce qui n'est pas toujours possible. Elle atteint cependant des performances très proches de celles de l'humain pour les principales traductions envisageables, comme en témoigne le graphique de la Figure 4.

On remarque en particulier que pour les traductions de l'anglais vers l'espagnol ou le français ou inversement du français vers l'anglais, les résultats sont presque aussi bon que ce que peut faire un humain.

1.4. Reconnaissance d'auteur

1.4.1. Généralités

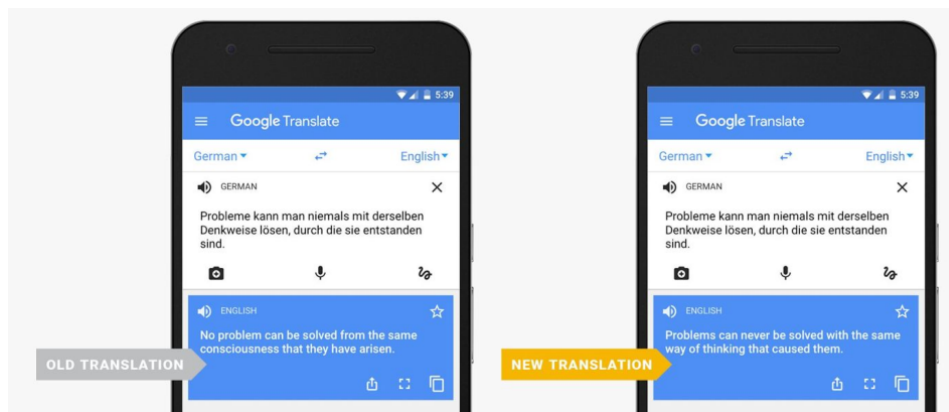


FIGURE 3 • Amélioration de Google Translate en utilisant des méthodes neuronales

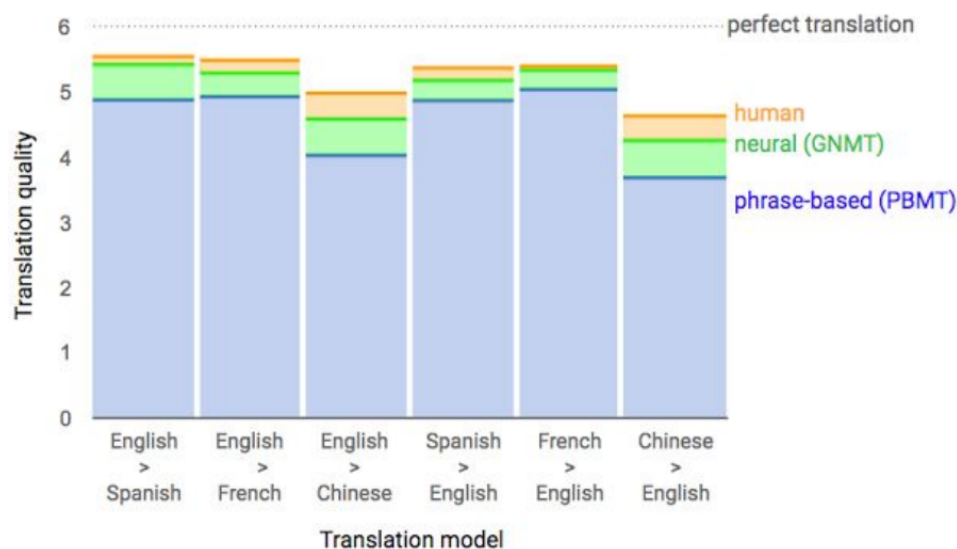


FIGURE 4 • Comparaison des différentes méthodes de traduction pour les cas « usuels »

Ce sujet correspond à une tâche permettant de reconnaître quel est l'auteur d'un document fourni en entrée du système de reconnaissance. L'idée est ainsi d'entraîner un classifieur sur un ensemble de textes (phrases, paragraphes, documents complets) dont on connaît les auteurs afin de pouvoir réaliser des vérifications ultérieurement.

L'intérêt pratique de ce système tient par exemple dans la détection de fraudes ou de plagats (ou par exemple la comparaison de travaux d'élèves pour savoir s'il s'agit d'une copie ou d'un original).

Ce sujet a été soumis à de nombreuses approches depuis une cinquantaine d'années, notamment l'utilisation de classifieurs de Bayes multivariés, de SVMs et plus récemment de RNN [14].

1.4.2. Bases de données disponibles

Différents projets ont pour but de regrouper des livres pour les rendre public, comme le projet Gutenberg [15]. Ces projets peuvent donc aisément fournir des bases de données (de contes par exemple) pour entraîner de petits réseaux.

1.4.3. Techniques utilisées

Comme toutes les applications déjà traitées, deux questions se posent :

- ◉ Quel modèle utiliser pour traiter les données ?
- ◉ Quelle représentation des données utiliser pour entraîner le modèle ?

Un des freins majeurs à l'utilisation des réseaux profonds pour le NLP étant le fait que les textes soient de tailles variables à la différence des images, la réponse à la deuxième question est primordiale pour avoir un système le plus générique possible.

Réseaux retenus Sur les différents cas d'application que nous avons pu trouver, les réseaux utilisés sont de type convolutionnels (CNN), qui utilisent également des solutions de *dropout* (ces dernières seront détaillées dans la partie consacrée à l'implémentation dans ce rapport).

De plus, un des travaux [16] mentionne l'utilisation d'une variante de la descente du gradient traditionnelle pour améliorer les résultats : AdaGrad. Cette source propose également quelques explications sur des choix d'architecture et l'utilisation des couches profondes du réseau.

Représentation des données Comme nous l'avons déjà plusieurs fois mentionné, la représentation préférentiellement utilisée pour ce projet a été la représentation GloVe, qui sera détaillé plus précisément dans l'application technique de ce projet.

Résultats obtenus Si les résultats peuvent être très bons (plus de 96% de précision [14]), ces derniers semblent être très dépendant de la taille des données en entrée ainsi que du nombre total de classe à entraîner. Il est cependant mentionné que ces résultats restent bien supérieur à une détermination au hasard, signe que cette tâche est réalisable avec du *machine learning*.

1.5. Analyse de sentiments

L'analyse de sentiment est une partie du NLP qui consiste à extraire le sentiment global d'un texte. Le sentiment global est alors vu comme un ressenti, c'est-à-dire que l'on cherche à savoir si le contenu est plutôt positif ou négatif. Ainsi, cette application (dans sa forme qui vous est ici présentée) ne cherche pas à savoir si l'auteur du texte a désiré faire passer de la joie, de la tristesse ou encore de la colère, mais plutôt si le sentiment général qui s'en dégage est positif ou négatif.

Un des aspects centraux pour l'analyse de sentiments est alors de trouver un modèle pour représenter la phrase de manière efficace. Nous allons présenter ces différents modèles dans la suite de cette partie.

1.5.1. Arbres récursifs

Dans ce modèle, la phrase est considérée comme un arbre descripteur, dont la structure est déterminée par la structure grammaticale de la phrase.

Le modèle a été en particulier mis en avant et développé par l'université de Stanford, on retrouvera donc tous les détails sur la page du projet [17] et le papier de recherche associé [18].

Nous présenterons dans la suite une synthèse de ce modèle.

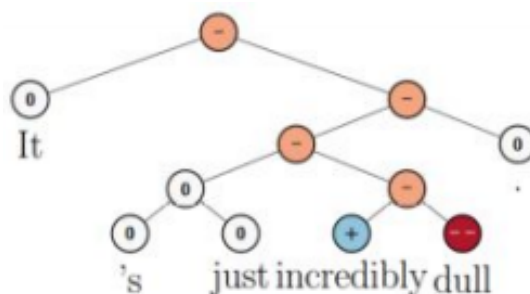


FIGURE 5 • Exemple d'arbres syntaxique pour l'analyse de sentiments

Pour appuyer notre propos, considérons un arbre syntaxique extrait des données de Stanford, proposé à la Figure 5. Sur cet arbre, chaque sous-arbre est labellisé sur une échelle de 1 à 5 (-- à ++). Chaque nœud, correspondant à un sous-arbre ou à une feuille, possède une représentation vectorielle (un simple vecteur par exemple, ou bien une paire (vecteur, matrice)).

On entraîne à la fois deux tenseurs, un qui à partir de deux nœuds, va donner le nœud supérieur, et un autre, qui à partir d'un nœud, va donner son label.

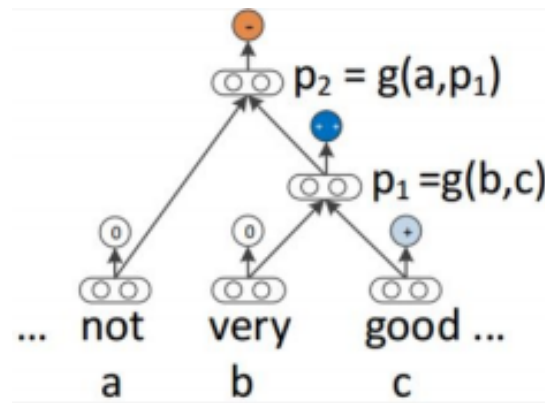


FIGURE 6 • Exemples d'entraînement pour labelliser un arbre syntaxique

Le label de la phrase complète est considéré comme le label de la racine. Pour un nouvel exemple, on transforme la phrase en arbre, et on va commencer par classifier par le modèle les feuilles ou sous-arbres élémentaires, et on va remonter la hiérarchie de l'arbre jusqu'à la racine comme à la Figure 6.

Cette méthode a montré de très bons résultats, mais elle présente des inconvénients :

- Elle reste très spécifique à l'analyse de sentiments.
- Elle nécessite une labellisation très fastidieuse des exemples d'apprentissage.

1.5.2. Vecteurs paragraphes

Une autre méthode part du principe de représenter l'information de la phrase ou du paragraphe, non comme un arbre, mais comme un vecteur. La manière la plus simple et la plus naïve consisterait à prendre la moyenne de tous les vecteurs qui constituent le paragraphe. Une manière qui se révèle plus efficace [19] consiste à chercher à prédire le vecteur du paragraphe à partir des vecteurs des mots qui le constituent, et inversement, prédire un mot à partir des vecteurs des mots de son contexte, et de son paragraphe, comme suivant la Figure 7.

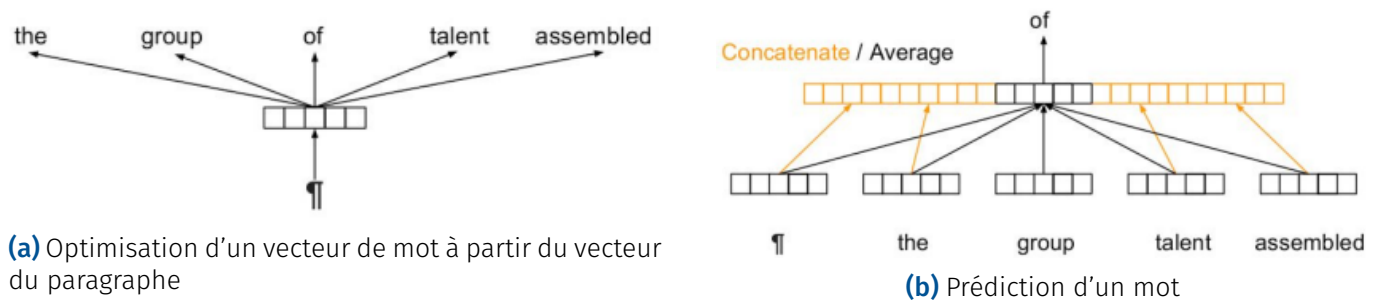


FIGURE 7 • Utilisateur des vecteurs paragraphes en prédiction

Pour la méthode représentée par la première image, une fenêtre glissante parcourt les mots du paragraphe, et on optimise la prédiction des vecteurs des mots de cette fenêtre à partir du vecteur du paragraphe, en optimisant conjointement le classifieur et les vecteurs-représentations.

Pour la méthode représentée dans la deuxième image, on concatène ou on moyenne les vecteurs des mots de la fenêtre glissante, auxquels on ajoute le vecteur du paragraphe, et on cherche à prédire le mot au centre de la fenêtre.

Ces méthodes sont des extensions de l'algorithme *Word2Vec* pour des paragraphes et non simplement des mots.

Une fois qu'on a représenté chaque paragraphe ou phrase par un vecteur, on peut appliquer une simple régression logistique pour prédire le sentiment du paragraphe.

1.5.3. Réseaux LSTM

Les réseaux LSTM sont des cas particuliers des réseaux récurrents (RNN), dont la prédiction à un instant t est prise comme entrée à l'instant $t+1$ en plus de la nouvelle entrée. Cela en fait des réseaux particulièrement adaptés aux données séquentielles, et en particulier les données textuelles, dans la mesure où ce bouclage crée un « effet mémoire » intéressant.

Très grossièrement, ces réseaux récurrents sont constitués d'une seule cellule, qui prend en entrée à la fois sa sortie à l'instant précédent, son état interne, et la nouvelle entrée. Elle va alors actualiser son état interne, et fournir une nouvelle sortie.

La particularité de cette cellule, et ce qui fait son efficacité, est qu'elle est naturellement conçue pour retenir de l'information sur des périodes arbitrairement longues. Elle arbitre à chaque itération entre la nouvelle information qu'elle peut rajouter dans son état interne, et la persévérance de l'information contenue dans l'état interne de l'ancienne itération à la nouvelle.

Un post de blog rédigé par Christopher OLAH [20] présente l'architecture complète des LSTM et leur fonctionnement détaillé. Il nous a en particulier servi pour les visuels et les explications de la présentation orale.

Nous détaillerons ici simplement l'utilisation des LSTM pour l'analyse de sentiments, tout lecteur intéressé dans les aspects théoriques pourra donc se rapporter à ce blog [20] et aux visuels de notre présentation.

Pour notre application, considérons la Figure 8.

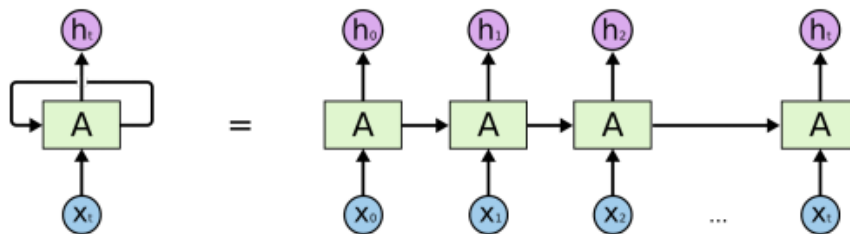


FIGURE 8 • Fonctionnement global d'un LSTM pour propager l'information retenue

En considérant que, dans la Figure 8, que les entrées x_1, \dots, x_t sont les mots de notre paragraphe, la méthode qui semble le mieux convenir pour faire de l'analyse de sentiment est de moyenner l'ensemble des états internes obtenus à chaque itération, de projeter ces états (qui sont des matrices) sur des vecteurs, puis de faire de nouveau une régression logistique. Une étude qui compare de manière exhaustive ces différentes méthodes a été proposée par de Hong JAMES [21].

L'utilisation pratique d'un réseau LSTM sera abordée en Section 3.

1.6. Choix d'un sujet

Pour la suite de ce travail, il nous a fallu choisir un sujet de test. Parmi, les différents sujet qui vous ont été présentés, le choix s'est effectué en fonction des critères suivants :

- ◉ *Difficulté technique* Cette dernière représentait soit le pas théorique à franchir, soit la difficulté pour appréhender les réseaux mis en place dans le laps de temps du projet imparti à cette tâche.
- ◉ *Intérêt du sujet* L'intérêt que portait l'ensemble de l'équipe pour le sujet à retenir.
- ◉ *Apport possible* L'objectif n'étant pas de juste refaire une expérience », mais plutôt d'essayer d'apporter une plus-value sur le sujet.
- ◉ *Ressources disponibles* Si certains modèles neuronaux peuvent sembler attrayant (ceux avec de la mémoire notamment), nous avons dû prendre en compte les capacités de calculs limitées de nos machines dans le choix du sujet (et le choix technique potentiel qui en découlait).

Si les différents détails sont fournis dans les présentations orales que nous avons faites, nous ne reprendrons ici que la conclusion générale qui a été de s'orienter vers de l'analyse de sentiments. Il est à noter que ce choix a également été fortement motivé par la faciliter de trouver un grand nombre de données libres et « réelles » en lien avec ce sujet. De plus, les modèles neuronaux utilisés (LSTM) pouvaient encore s'exécuter en temps raisonnable sur nos machines. Enfin, ce sujet a justement été l'occasion de tester l'utilisation de ces LSTMs dans ce domaine récent ou ces derniers semblaient peu utilisés.

2 • Base de données

2.1. Rencontre avec un PAR

Dans notre recherche de base de données, la première étape proposée par nos commanditaires a été la rencontre d'un PAR travaillant sur des problématiques similaires. Nous avons ainsi rencontré Thomas BLONDELLE le jeudi 2 février.

Lors de cette réunion, ce dernier nous a présenté son objectif de travail, avant que nous abordions la liste des bases de données qu'il avait déjà pu récupérer et en partie pré-traiter.

Les bases de données proposées consistaient essentiellement en les deux jeux suivants :

- ◉ **Contes** Une base de données constituées de contes libres de droits et trouvables en ligne. Cette base contenait ainsi les contes, découpés en paragraphe.
- ◉ **Wikipédia** Une base de données constituées de phrases issues de Wikipédia, avec l'avantage d'être en français.

Cependant, dans le cadre de notre application choisie, nous étions à la recherche de bases de données en lien avec l'analyse de sentiments, comme des critiques (de films de produits), ou des phrases labellisés dans cet objectif. Ainsi, les bases de données proposées par le PAR ne répondaient pas à ces attentes et n'ont donc pas été retenues pour la suite de ce travail.

2.2. Les bases de données utilisée

2.2.1. Recherche de bases de données en analyse de sentiment

Suite à la rencontre avec le PAR décrite à la Section 2.1, nous avons alors cherché des bases de données plus orientées sur l'analyse de sentiments. Nos recherches nous ont permis de dégager les bases détaillées à la Table 1.








	Nom	Quantité de données	Origine	Source
	Large Movie Review Dataset	25000 × 2	Stanford	[22]
	Rotten Tomatoes Dataset	20 399	Kaggle	[23]
	Twitter Sentiment Corpus	5500	Niek SANDERS	[24]
	Twitter Sentiment Analysis Corpus	1 578 627	Anonyme	[25]
	Sentiment Analysis Dataset	9645	Stanford	[26]
	UMICH SI650	40 000	Kaggle	[27]
	Amazon reviews	> 6 millions	Julian MC. AULEY	[28]

TABLE 1 • Bases de données considérées pour le projet

Les bases de données de la Table 1 ont été regroupées par types de données. On retrouve ainsi :

- ◉  Des bases de données liées à des critiques de films. Chaque commentaire est associé à une évaluation faite par l'utilisateur, ainsi on garantit le fait que la base soit labellisée manuellement par des humains, seul l'extraction du site a été automatisée.
- ◉  Des bases de Tweets, dans le détail la base la plus large inclus la plus petite de 5500 Tweets. La base de 5500 Tweets a été labellisée manuellement par son auteur, ce qui garantit le réalisme des annotations. La deuxième base quant à elle a été labellisé par apprentissage semi-automatique, ainsi on ne peut être sûr de la validité de tous ces labels.

- Des bases données diverses de commentaires ou de phrases diverses et labellisées. Celle issue de Kaggle provient d'un challenge de Machine Learning et a été fournie par l'université du Michigan. La base de Stanford quant à elle est un peu particulière et correspond à des arbres syntaxiques sur le modèle de celui qui vous a été présenté à la Section 1.5.1. Le découpage et la validation des données ont été réalisés avec Amazon Turk, il est difficile d'étendre facilement son volume.
-  Cette dernière base de données est la plus conséquente et a été proposée par un professeur de l'université de Californie San Diego. Elle provient d'un scrapping d'Amazon et est donc elle aussi initialement assurée d'être anotée par des humains. Il s'agit de la base la plus volumineuse que nous ayons pu récolter.

2.2.2. Choisir quelle base utiliser

Pour notre projet, il nous a alors fallu décider sur quels types de bases de données nous allions concentrer nos efforts. Nous avons alors dressé un tableau comparatif des caractéristiques de ces bases qui vous est proposé à la Table 2. Les critères retenus sont les suivants :

- Quantité de données** Le volume est-il réaliste vis-à-vis d'une utilisation pour du Deep Learning ou le volume de données demandé est souvent grand (**insuffisant**; suffisant; **très fourni**). Nous avons généralement considéré que nos bases de données devaient contenir au moins 10 000 données pour être d'un suffisantes.
- Qualité de l'annotation** Permet d'évaluer la fiabilité que l'on fournit envers l'annotation réalisée. La meilleure solution étant les cas où l'on est assuré que la base a été annotée (et/ou vérifiée) par des humains (**peu fiable**; **fiable**).
- Type de données** Le type de données qui est annoté, à savoir des phrases, des paragraphes,...
- Format de données** Quel est la forme des données, et dans quelle mesure leur lecture sera facilement réalisable avec des outils à notre disposition. Le jugement viendra du fait que la structure soit **courante** ou plus **exotique**.





	Nom de la base	Quantité de données	Qualité de l'annotation	Types de données	Format de données
	Large Movie Review Dataset	50 000	+	Paragraphes	Par fichier
	Rotten Tomatoes Dataset	20 399	++	Phrases (et arbres syntaxiques)	Spécifique
	Twitter Sentiment Corpus	5500	+	Tweets	API Python
	Twitter Sentiment Analysis	1 578 627	-	Tweets	CSV
	Sentiment Analyses Dataset	9645	+	Arbres syntaxiques	Spécifique
	UMICH SI650	40 000	+	Phrases	CSV
	Amazon reviews	> 6 000 000	+	Paragraphes	JSON

TABLE 2 • Comparatif des intérêts et des problèmes techniques des différentes bases de données repérées

Suite à ce comparatif, nous avons écarté les bases de données liées à Twitter. En effet, le format de texte de ces dernières (le Tweet), bien qu'intéressant, utilise une syntaxe particulière, qui n'est pas nécessairement représentative de la « vraie » syntaxe. Si son étude n'est pas dénuée d'intérêt, nous avons préféré privilégier une cohérence dans nos réalisations, en nous focalisant plus sur des phrases écrites dans une syntaxe non contrainte. De plus, le fait que la base Twitter pour l'analyse de sentiment utilisable en Deep Learning provienne d'un apprentissage semi-supervisé nous a conforté dans notre choix, dans la mesure où nous préférons privilégier des bases annotées par des personnes physiques.

Nous nous sommes ainsi orienté vers une étude sur les retours de clients d'Amazon ainsi que sur les critiques de cinéma.

2.3. Mise en forme des données

Nous avons désormais cerné l'ensemble des bases de données que nous souhaitons utiliser. La dernière étape de préparation de ces dernières était alors de les remettre en forme pour leur utilisation.

2.3.1. Des formats divers

Les bases de données retenues provenaient de sources variées, et ainsi leurs formats n'étaient pas uniformes. Pour en témoigner, nous allons fournir ci-après quelques exemples issues de ces dernières.

Retours clients d'Amazon Cette base de données est la plus conséquente que nous ayons récupérée. Il s'agit également des données les mieux structurées que nous avons) disposition, en effet, ces dernières étaient initialement proposées sous la forme de fichiers JSON. De même, chaque fichier JSON correspondait à une thématique précise (outillage, livres, CDs,...), ainsi cette base de données étaient proposées sous formes de fichiers cohérents et structurés.

À titre d'exemple, vous pourrez trouver Figure 9 le contenu d'une critique trouvée dans un des fichiers originaux de la base de données d'Amazon.

```
1 {  
2   "reviewerID": "A2IBPI20UZIROU",  
3   "asin": "1384719342",  
4   "reviewerName": "cassandra tu \"Yeah, well, that's just like, u  
5     ...\",  
6   "helpful": [0, 0],  
7   "reviewText": "Not much to write about here, but it does exactly  
8     what it's supposed to. filters out the pop sounds. now my  
9     recordings are much more crisp. it is one of the lowest prices  
10    pop filters on amazon so might as well buy it, they honestly  
11    work the same despite their pricing,",  
12  "overall": 5.0,  
13  "summary": "good",  
14  "unixReviewTime": 1393545600,  
15  "reviewTime": "02 28, 2014"  
16 }
```

FIGURE 9 • Exemple de données de la base de données Amazon

On retrouve ainsi dans cette base de données de nombreux éléments concernant l'auteur et le contexte du retour client. Dans le cadre de notre étude cependant, peu de champs nous intéresse, à savoir les suivants :

- *reviewText* Le texte du retour client sur le produit. Pour notre application, c'est à partir de ce dernier que l'on devra chercher à trouver la note attribuée.
- *overall* La note donnée par le client au produit, ie. celle que l'on va chercher à prévoir (elle reflète le sentiment du client sur le produit).
- *summary* Il s'agit d'une version condensée du champs *overall* permettant de savoir si une échelle « bon/pas bon » (binaire) le sentiment du client. En regardant la base de données, on remarque que la limite se situe aux alentours de 2,5-3, nous considérerons donc que les revues clients avec une note supérieure ou égale à 3 sont considérées comme positives.

Ces différents champs seront réutilisés dans le code pour harmoniser ces bases de données entre elles comme nous le verrons dans la suite de cette section.

Critiques de cinéma (LMDb) Cette base de données a été agrégée par Stanford dans le cadre de leurs travaux et rendue publique. Le format de cette base est cependant original dans la mesure où elle consiste en des fichiers, chacun d'entre eux contenant une critique. De plus, la note associée à la critique est elle précisée dans le nom du document.

En exemple de cette base de données (fichiers avec les notes et contenu de fichier) vous est proposé à la Figure 10.

On remarquera dans la liste des fichiers, que la note (allant de 1 à 10) est située après un « _ ». De plus, les différents exemples sont rangés dans une architecture particulière :

0_2.txt	12/04/2011 11:48	Fichier TXT	1 Ko
1_3.txt	12/04/2011 11:48	Fichier TXT	1 Ko
2_3.txt	12/04/2011 11:48	Fichier TXT	2 Ko
3_4.txt	12/04/2011 11:48	Fichier TXT	1 Ko
4_4.txt	12/04/2011 11:48	Fichier TXT	1 Ko
5_4.txt	12/04/2011 11:48	Fichier TXT	3 Ko
6_3.txt	12/04/2011 11:48	Fichier TXT	2 Ko
7_1.txt	12/04/2011 11:48	Fichier TXT	2 Ko
8_2.txt	12/04/2011 11:48	Fichier TXT	2 Ko

Once again Mr. Costner has dragged out a movie for far longer than necessary. Aside from the terrific sea rescue sequences, of which there are very few I just did not care about any of the characters. Most of us have ghosts in the closet, and Costner's character are realized early on, and then forgotten until much later, by which time I did not care.[...]

FIGURE 10 • Exemple de données issues de la bases de données de LMDB

- ◉ **Deux répertoires à la racine** Un pour les documents de test (*test*) et un pour de l'apprentissage (*train*).
- ◉ **Des sous-répertoires** Pour chacun des ensembles de tests et d'apprentissage, on retrouve un répertoire *neg* pour les exemples négatifs et un répertoire *pos* pour les exemples positifs. De plus, pour l'apprentissage, on retrouve un certains nombre de documents pour lesquels on ne dispose pas d'information (*unsup*), qui ne seront donc pas utilisés par la suite pour notre application.

On remarque en particulier que l'on dispose bien de tous les labels pour le test et l'apprentissage.

Ainsi, le traitement de cette base de données sera, de par sa structure, fondamentalement différent de celui pour Amazon.

Critiques de cinéma (Rotten Tomatoes) Cette base de données reprend l'idée de l'utilisation d'arbres syntaxiques. Ainsi, sa structure diffère des deux bases de données précédemment rencontrées, dans la mesure où chaque ligne (ou chaque fichier) n'est pas directement exploitable.

Pour mieux comprendre la structure de cette base de données, regardons plus en détails comment y est stockée la phrase suivante :

This quiet, introspective and entertaining independent is worth seeking.

Cette phrase est stockée à l'aide des lignes suivantes :

```

1 PhraseId SentenceId Phrase Sentiment
2 64 2 This quiet , introspective and entertaining independent is worth seeking . 4
3 65 2 This quiet , introspective and entertaining independent 3
4 66 2 This 2
5 67 2 quiet , introspective and entertaining independent 4
6 68 2 quiet , introspective and entertaining 3
7 69 2 quiet 2
8 70 2 , introspective and entertaining 3
9 71 2 introspective and entertaining 3
10 72 2 introspective and 3
11 73 2 introspective 2
12 74 2 and 2
13 75 2 entertaining 4
14 76 2 independent 2
15 77 2 is worth seeking . 3
16 78 2 is worth seeking 4
17 79 2 is worth 2
18 80 2 worth 2
19 81 2 seeking 2

```

On retrouve ainsi l'idée de notre arbre syntaxique, avec une phrase qui est découpée de manière hiérarchique, et où chaque tronçon est labellisé avec une valeur donnée. On retrouve ainsi la note totale de la phrase (4) qui est construite à partir des notes élémentaires de chaque tronçon.

Dans la mesure où la structure que nous avons retenu par la suite est un LSTM, nous n'utiliserons par directement ces arbres syntaxiques. Ainsi, dans cette base de données, la seule partie qui nous intéressera

sera la phrase complète. Dans le fichier fourni, il s'agit toujours de la phrase correspondant à la première occurrence du *SentenceId* de la phrase. Cette idée est utilisée dans le code pour harmoniser cette base de données aux autres.

Sentiment Analysis Dataset Cette base de données utilise également des structures à base d'arbres. Cependant, la structure est ici différente et beaucoup moins visuelle pour un humain. Un exemple vous est fourni à la Figure 11.

```
1 (2 (3 (3 Effective) (2 but)) (1 (1 too-temperid) (2 biopic)))
2 (3 (3 (2 If) (3 (2 you) (3 (2 sometimes) (2 (2 like) (3 (2 to) (3 (3 (2 go) (2 (2 to)
  (2 (2 the) (2 movies)))) (3 (2 to) (3 (2 have) (4 fun))))))))) (2 (2 ,) (2 (2
  Wasabi) (3 (3 (2 is) (2 (2 a) (2 (3 good) (2 (2 place) (2 (2 to) (2 start)))))) (2
  .))))))
```

FIGURE 11 • Exemple de données de la base de données *Sentiment Analysis Dataset*

Le sentiment total de la phrase est le premier chiffre de chaque ligne, et les parenthèses permettent de définir les différentes sous-parties de manière récursive. Cette base a été traitée (de manière un peu simpliste), malgré le fait qu'elle n'apporte que peu de nouveaux éléments en comparaison avec le volume de celle provenant d'Amazon.

2.3.2. Harmoniser les bases de données

Nous vous avons présenté ci-dessus les quatre bases de données que nous comptons harmoniser. Pour réaliser ceci, nous sommes parti de l'idée que la plupart de ces bases de données fonctionnaient par une lecture « ligne par ligne » des fichiers (mis à part LMDB). Nous avons donc mis en place une structure de lecture des fichiers ligne à ligne, qui délègue à une méthode adéquate le traitement des-dites lignes.

Structure générale du code La structure globale du code s'oriente autour des deux éléments suivants :

- ⊙ *main.cpp* Comme pour tout projet C++, ce fichier joue le rôle de chef d'orchestre entre les classes. L'idée est ici de configurer les chemins vers les bases de données brutes dans ce fichier ainsi que de déterminer quels sont les fichiers à convertir.
- ⊙ *La classe JsonWorker* Cette dernière est en charge de toutes les étapes courantes pour le traitement des fichiers, à savoir :
 - ▷ *Création du fichier de sortie* Ce dernier est créé directement s'il n'existe pas, et dans le cas où il existe déjà on demande à l'utilisateur l'autorisation de l'écraser (pour éviter les erreurs, dans la mesure où certains traitements peuvent être un peu long).
 - ▷ *Lecture ligne à ligne* Ceci est réalisé une fois que les fichiers d'entrée et de sortie ont été vérifiés. La plupart des actions sont menées directement dans la méthode *launchWorker*. Cette méthode va simplement lire le fichier ligne à ligne et déléguer le traitement de chaque ligne pour toutes les classes héritant de *JsonWorker*.
 - ▷ *Traitement des lignes* La classe *JsonWorker* propose à ses héritières d'implémenter une méthode *parseData* qui précise quel traitement appliquer aux diverses lignes pour générer notre résultat final.

On remarquera cependant que le traitement pour LMDB est un peu différent et ne reprend donc pas cette structure (le *worker* associé n'hérite pas de *JsonWorker*).

Dans la suite, nous allons préciser quel traitement sont effectués en fonction des bases de données.

Traitement des bases ligne par ligne Comme nous l'avons précisé, ce dernier est réalisé par la méthode *parseData*, qui prend en paramètre la ligne à traiter sous forme d'une *QString*.

Le traitement est alors différent pour les différentes classes suivantes :

AmazonWorker, *RottenWorker*, et *SentimentWorker*

Les détails sont suffisamment commentés dans le code pour comprendre le fonctionnement (traitement de Json, TSV, arbres). Le point important est la sortie pour chaque ligne qui est une ligne à insérer dans un fichier TSV sous la forme :

```
Comment    RatingDegrad    Rating    TextType    BDDName
```

Nous obtenons donc les éléments de base, à savoir le contenu textuel, le nom et l'origine du contenu. Concernant l'évaluation, le champs *RatingDegrad* correspond à la note « simplifiée » (content ou pas content) et le champs *Rating* à la note réel. Les seuils pour réaliser la note dégradée sont directement visibles dans le code.

Traitement de la base LMDB Pour la base de données LMDB, le fonctionnement est ici différent et ne s'inscrit pas directement dans le même cadre (bien que nous ayons fait en sorte que ce *worker* hérite également de *JsonWorker*).

À la différence de traiter les lignes une à une, nous allons ici traiter les différents répertoires de l'architecture de manière automatique et lire les fichiers un à un (voir les méthodes *launchWorker* ré-implémenter dans *LargeMovieWorker* puis l'implémentation de *parseData*).

Le reste consiste simplement à extraire la note du nom du fichier et à créer une ligne pour l'export TSV sous la même forme que celle vue auparavant.

2.3.3. Résultats de l'harmonisation

Fichiers mis à disposition Les différents fichiers traités sont mis à disposition sur un répertoire dans Google drive au lien suivant :

<https://drive.google.com/drive/folders/OB-qeHiX-LTwoWnEtTDk3ZkNFZGM>

Ces fichiers sont tous au format TSV et issus du code proposé. Nous avons ainsi pu constituer une base de données complète et harmoniser à partir des différentes bases trouvées sur Internet.

Les fichiers ont alors des contenus similaire à celui de la Figure 12.

```
1 Comment    RatingDegrad    Rating    TextType    BDDName
2 "It 's a lovely film with lovely performances by Buy and Accorsi ."
   1 3    sentence    Sentiment_Analysis_Dataset
3 "No one goes unindicted here , which is probably for the best ."    0
   2    sentence    Sentiment_Analysis_Dataset
4 "And if you 're not nearly moved to tears by a couple of scenes ,
   you 've got ice water in your veins ."    1 3    sentence
   Sentiment_Analysis_Dataset
5 "A warm , funny , engaging film ."    1 4    sentence
   Sentiment_Analysis_Dataset
```

FIGURE 12 • Un exemple de fichier de données après harmonisation des bases

Lancer les worker Pour récupérer les bases de données remaniées, deux solutions sont envisageables :

- ⊙ **Les télécharger depuis Google Drive** Si votre connexion Internet est suffisante et que vous ne désirez pas réaliser de traitement particulier des données brutes, il est possible de les télécharger depuis notre Drive. Attention cependant, le téléchargement complet de la base de données sera long, dans la mesure où cette dernière mesure un peu moins de 4Go.
- ⊙ **La régénérer à partir des bases initiales** Le code de traitement des bases vous étant fourni, il vous est possible de télécharger les bases initiales et de les traiter manuellement sur votre ordinateur. Vous pouvez dans ce cas adapter le code de pré-traitement si vous disposez de besoins particuliers. De plus, la base d'Amazon n'a pas été complètement traitée (plus de 20Go au totale, ce qui était surdimensionné pour notre application), il vous est donc possible de télécharger la partie que nous n'avons pas traité pour agrandir la base que nous vous proposons.

Afin que vous disposiez d'une autonomie suffisamment grande pour le traitement de la base sur un poste individuel, nous allons préciser rapidement les grandes parties du fichier *main.cpp* pour lancer les *worker* sur les bases de données brutes.

Chaque base de données est associée à un booléen pour lancer ou non son traitement. Ces booléens sont les premiers éléments à configurer dans le fichier *main.cpp*. On remarquera que dans le cas d'Amazon, nous proposons au total quatre booléens en fonction de la taille des fichiers à traiter (certains traitements pouvant être longs). Nous avons ainsi les déclarations suivantes :

```

1  /**
2   * Définir quels traitements sont à lancer.
3   * Pour les chemins vers les BDD, voir les différentes parties associées à
4   * chaque BDD.
5   */
6  bool launchSentiment = false;      // Toute la base Sentiment Analysis Dataset.
7  bool launchRotten = false;        // Toute la base Rotten Tomatoes.
8  bool launchLMDB = false;          // Toute la base Large Movie DataBase.
9  bool launchAmazonAll = false;     // Toute la base d'Amazon.
10 bool launchAmazonInf100 = false;   // Les éléments d'Amazon de taille < 100Mo.
11 bool launchAmazon100500 = false;   // Les éléments d'Amazon où 100Mo < taille <
    500Mo.
    bool launchAmazonSup500 = false; // Les éléments d'Amazon où taill > 500Mo.

```

Il vous suffit ensuite pour chaque base à traiter de regarder quels *workers* sont proposés et de les configurer. La plupart des *workers* (à l'exception de celui pour IMDB) ont les paramètres suivants :

- ◉ *Chemin du fichier d'entrée* Le chemin relatif vers le fichier à traiter pour cette base de données.
- ◉ *Chemin du fichier de sortie* Le chemin relatif vers le fichier qui contiendra le TSV à la sortie du traitement.
- ◉ *Nom de la base de données* Celui qui apparaîtra dans la colonne associée pour le fichier de sortie (l'idée est de savoir d'où viennent les données si on concatène les bases entre elles).

Un exemple de configuration pour le cas d'Amazon vous est proposé ci-dessous :

```

1 AmazonWorker ACellPhoneWorker(QString::fromStdString(amazonInputPath + "
    Amazon_Cell_Phones_and_Accessories.json"), QString::fromStdString(
    amazonOutputPath + "Amazon_Cell_Phones_and_Accessories.csv"), "
    Amazon_Cell_Phones_and_Accessories");
2 ACellPhoneWorker.launchWorker();

```

Une fois le *worker* configuré, il restera alors à le lancer en appelant sa méthode *launchWorker*.

Le fichier *main.cpp* contient toutes les déclarations de *workers* pour les fichiers du Drive. Si vous désirez étendre la base de données (en utilisant d'autres fichiers de la base de données d'Amazon par exemple), il vous faudra ajouter les *workers* associés dans le fichier *main.cpp* et relancer une compilation du projet.

3 • Étude pratique d'un modèle

3.1. Motivation du choix du modèle

Nous avons choisi d'étudier l'implémentation d'un réseau LSTM car ce modèle semble être de loin le plus général, et ne se limite pas aux applications de *Natural Language Processing*.

Il est ainsi utilisé en *Image Captioning*, traduction, transcription de données audio,...ie. tout type de données séquentielles.

D'autres solutions plus complexes auraient encore pu être envisageables, ces dernières seront mentionnées dans les perspectives de notre conclusion.

Nous verrons que nos choix nous ont conduit dans le cas de la base de données extraite d'IMDB à des résultats proches de ceux de l'état de l'art à savoir 85% d'exactitude sur l'ensemble de test. Nous détaillerons pour cela en Section 3.2 les hyperparamètres du modèle.

3.2. Les hyperparamètres du modèle

Pour comprendre quels étaient les hyperparamètres du modèle et leur influence, nous avons fait varier ces dernier pour déterminer leurs importances relatives et leurs valeurs optimales.

Nous allons ici vous présenter nos résultats, tant au niveau des valeurs retenues que de l'utilité des paramètres.

3.2.1. Hyperparamètres basiques

Nous commençons avec des hyperparamètres génériques, qui peuvent se retrouver au sein de diverses méthodes d'apprentissage.

Paramètre `VALID_BATCH_SIZE` Taille de l'ensemble de validation. Plus la valeur est élevée, plus l'évolution de l'exactitude est fiable, mais plus on prend du temps à chaque estimation de cette exactitude.

La valeur retenue est de `1000`.

Paramètre `SIZE_EMBEDDING` Taille du vecteur qui représente les mots. La valeur retenue est de `50`, par soucis d'avoir un modèle synthétique mais qui ne converge pas trop lentement. Il est cependant probable que les performances s'améliorent pour des vecteurs plus grands. À noter que les vecteurs utilisés sont produits par un algorithme développé par le projet *GloVe* (*Global Vectors for Word Representation*) [29].

Paramètre `VOCAB_SIZE` Taille du vocabulaire considéré. On considère comme « inconnus » tous les mots qui seront moins fréquents que le 10000^{ème} mot le plus fréquent. Ces mots auront la même représentation vectorielle.

Plus `VOCAB_SIZE` est grand, et plus on peut gagner en expressivité du modèle, mais plus on détériorera la capacité de généralisation du modèle.

Comme suggéré un peu plus haut, la valeur retenue ici est de `10000`.

Paramètre `MAXLEN` Taille maximum pour la prise en compte des séquences. Un grand `MAXLEN` donnera un modèle plus réaliste, avec un ensemble d'exemples plus grand, mais le temps de calcul augmentera.

Paramètre `NO_CUT` Si la séquence en entrée est de longueur supérieure strictement à `MAXLEN`, la question est de savoir si on la conserve en la tronquant à `MAXLEN` mots, ou si on ne la considère pas du tout dans le modèle.

Paramètre `LEARNING_RATE` Pas d'apprentissage lors de la descente de gradient. Il eu été possible de choisir un pas d'apprentissage décroissant, mais cela n'a pas été fait.

Paramètres `OUTPUT_SIZE`, `TINY`, `DO_TEST` Paramètres pas ou peu importants. Ils déterminent respectivement la taille de la sortie (ici binaire donc de taille `1`), une petite valeur pour des cas particuliers lors du

calcul de l'entropie croisée, et un booléen qui indique si on calcule ou pas la performance du modèle sur l'ensemble de test à la fin.

3.2.2. Hyperparamètres les plus importants

Suite à ces hyperparamètres génériques, nous allons désormais nous attarder sur ceux plus spécifiques au cas des LSTM.

Paramètre DROPOUT Permet de savoir si l'on applique ou non la technique de régularisation par drop-out [?]. Cette méthode réduit très fortement le risque de sur-apprentissage, mais le modèle met alors plus de temps à converger.

Paramètre BATCH_SIZE Le nombre de séquences que l'on donne au modèle entre deux descentes de gradient. En effet la descente du gradient pour les LSTM n'est pas simple et ne peut pas se faire à chaque itération, on procède donc par « cycles ».

Plus **BATCH_SIZE** est grand, et plus la convergence sera lente, mais plus on aura de chances de « descendre dans la bonne direction ». Cet hyperparamètre est à mettre en relation avec **LEARNING_RATE**.

La valeur de **BATCH_SIZE** peut également être limitée par la quantité de mémoire RAM dont on dispose.

Paramètre RNN_HIDDEN Taille de la couche cachée du LSTM. Plus elle est grande, et plus le calcul de la sortie à chaque itération prendra de temps tout comme le risque de sur-apprendre. Cependant, plus il est grand et plus on pourra apprendre de choses complexes (en supposant pour simplification que les coefficients de la couche cachée ne peuvent prendre que des valeurs binaires, alors pour une couche cachée de taille n , on pourra avoir 2^n états internes différents).

Paramètre ITERATIONS_PER_EPOCH Nombre de descentes de gradient (chacune réalisées après un nombre de **BATCH_SIZE** exemples), entre deux calculs de l'exactitude sur l'ensemble de validation.

Avoir un petit **ITERATIONS_PER_EPOCH** mène à observer plus régulièrement l'évolution du modèle et peut rassurer quant à sa convergence, mais on risque d'observer une évolution plus « aléatoire du modèle » (due à la variabilité des exemples de chaque batch de taille **BATCH_SIZE**).

Paramètre NUM_EPOCHS Nombre de fois que l'on répète les **ITERATIONS_PER_EPOCH** itérations. Ainsi au final, on donne au modèle le nombre d'exemples suivants :

$$NUM_EPOCHS \times ITERATIONS_PER_EPOCH \times BATCH_SIZE$$

Ce paramètre permet de contrôler le nombre total d'itérations qu'aura effectué notre modèle.

3.3. Représentation des mots

3.3.1. Bag of Word

L'abstraction qui a pendant très longtemps été employée est celle du *BOW* ou *Bag Of Word* (sac de mots). Elle consiste à considérer un mot comme une entité unique et atomique, et à l'identifier de manière unique comme tel. Le contexte ainsi que l'ordre des mots n'a aucune importance, ce processus est illustré à la Figure 13.

Chaque mot est donc représenté par un vecteur de la taille du vocabulaire considéré, avec des zéros partout sauf à l'endroit faisant référence au mot.

Si le mot « chien » est le mot traité en 6^{ème} position par un modèle BOW pour un vocabulaire de 10 mots, « chien » sera numériquement représenté par (0, 0, 0, 0, 0, 1, 0, 0, 0, 0).

Le principal désavantage de cette représentation est qu'elle ne peut pas transférer les connaissances qu'elle a acquise sur des mots qu'elle n'a jamais vus. De plus le mot « niche » peut signifier aussi bien « niche fiscale » que « niche du chien ».

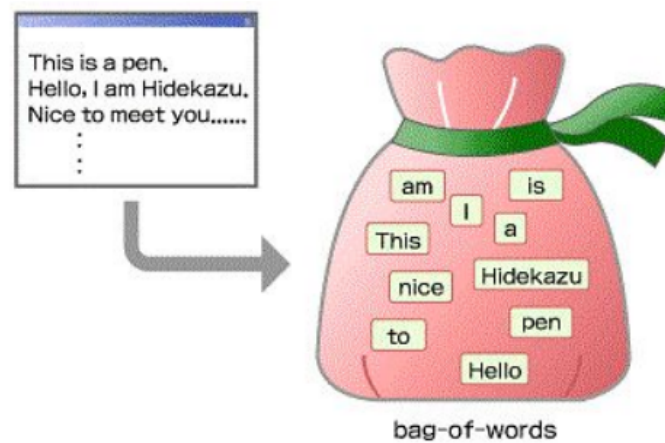


FIGURE 13 • Représentation de l'idée de la méthode Bag of Word

3.3.2. Les N-grams

Pour contrer le dernier écueil observé avec les Bag of Word, la base de la représentation a été étendue au modèle N-gram. Ici, l'élément atomique n'est plus un mot seul, mais n mots consécutifs, comme illustré à la Figure 14.

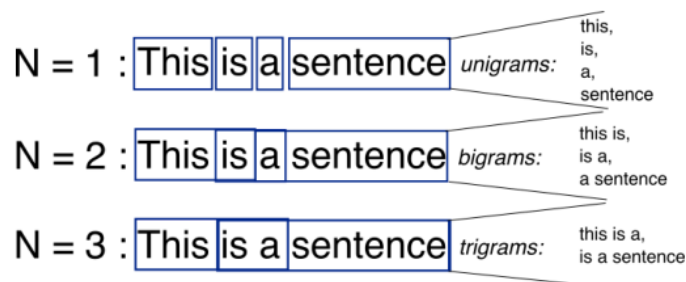


FIGURE 14 • Représentation d'une phrase à l'aide de N-grams

Cette représentation a le mérite de préciser le contexte immédiat d'un mot, mais on ne résout pas les problèmes principaux :

- ◉ **Incapacité à généraliser** Pour $N=4$, *LE CHIEN A MANGÉ* n'a rien à voir avec *LE CHAT A MANGÉ*.
- ◉ **Taille des vecteurs** Ces derniers ont des tailles très (trop) importantes, d'autant plus grande que le vocabulaire, et iN , sont grands. On retombe alors dans le problème de la « *malédiction de la dimension* ».

3.3.3. Représentations denses

Pour résoudre ces problèmes, il était nécessaire d'apprendre des représentations denses, et continues. Ces représentations correspondent à des vecteurs de taille limitée, et composés de réels et non de zéros et de uns. On pourrait ainsi généraliser tout en gardant une taille de représentation correcte. C'est ce qu'on appelle les vecteurs sémantiques.

Pour cela, deux grands modèles de vecteurs sémantiques sont apparus [30], l'un fondé sur le dénombrement du contexte et l'autre sur la prédiction du contexte.

Vecteurs de dénombrement du contexte Pour ces vecteurs, on peut citer la méthode GloVe (Global Vector for Word Representation) [29]. Résumée sommairement, cette méthode utilise une matrice de co-occurrences, modifiée par *Tf-Idf*, et réduite par *SVD*, à partir de corpus énormes, jusqu'à des représentations de longueur réduite (50, 100, voire 500).

Vecteurs de prédiction du contexte Nous nous attarderons ici aux méthodes Word2Vec [31]. Plus précisément, il existe deux grands types d'algorithmes.

Pour ces deux algorithmes, les mots sont d'abord instanciés comme des vecteurs réels aléatoires de taille fixée. Ces vecteurs sont considérés comme des variables d'un modèle à optimiser. Deux méthodes sont alors possibles :

- ◉ **Méthode Skip-Gram** Elle va tenter, à partir de la représentation d'un mot, de prévoir son contexte immédiat (celui contenu dans une fenêtre prédéfinie, par exemple les deux mots précédents et les deux mots suivants) à l'aide d'un classifieur qui sera optimisé par le modèle. Les représentations vectorielles des mots sont optimisées en même temps que le classifieur
- ◉ **Méthode C-BOW (Continuous-Bag Of Word)** Cette méthode fonctionne de la même manière que Skip-Gram, mais va chercher à prédire un mot à partir de son contexte.

Lien avec le modèle à représenter Ces deux manières d'optimiser les vecteurs sont décorréliées d'un modèle qui utiliseraient ultérieurement ces représentations.

Le projet GloVe par exemple, a pour objectif, en prenant en compte de très gros corpus, d'arriver à des représentations des mots qui sont assez générales pour être utilisées indépendamment d'un contexte spécifique.

Une autre approche peut consister en l'initialisation aléatoire des vecteurs, et à les considérer comme des variables à part entières du modèle, qui seront optimisées par descente du gradient en même temps que le modèle, de la même manière que les réseaux convolutifs extraient dans un premier temps les descripteurs primaires d'une image. Cette approche peut s'avérer plus performante pour une application au vocabulaire très spécifique, mais le risque est qu'un mot de l'ensemble de test qui n'aura jamais été vu dans la phase d'apprentissage aura un vecteur qui ne porte aucune information sémantique. Une solution peut être alors de considérer l'ensemble du vocabulaire disponible (apprentissage et test), d'optimiser des représentations vectorielles par la méthode Word2Vec, et ensuite d'optimiser le modèle sur ces représentations considérées comme des constantes.

3.4. Enseignements de l'apprentissage

3.4.1. Valeurs retenues

Les paramètres qui nous ont semblé optimaux sont :

```
1 LEARNING_RATE = 0.003      SIZE_EMBEDDING = 50
2 DROPOUT = 0                BATCH_SIZE = 15
3 VALID_BATCH_SIZE = 100     RNN_HIDDEN = 200
4 NO_CUT = 1                 ITERATIONS_PER_EPOCH = 20
5 MAXLEN = 500               VOCAB_SIZE = 5000
6 NUM_EPOCHS = 100
```

Une règle de bon sens dont nous nous sommes rendus compte est que, dans le cas d'IMDB, la grandeur :

$$NUM_EPOCHS \times ITERATIONS_PER_EPOCH \times BATCH_SIZE$$

devait dépasser le nombre total d'exemples disponibles (soit environs 20000, pour voir une convergence).

3.4.2. Méthode de choix des valeurs

Il est important de noter que certaines valeurs de paramètres ont été guidées plus par le temps de convergence attendu, que par l'optimisation des résultats. Si nous avons à notre disposition autant de temps et de puissance de calcul que désiré, alors l'idée serait :

- ◉ **BATCH_SIZE** L'augmenter jusqu'à la taille de l'ensemble d'apprentissage complet.
- ◉ **NUM_EPOCHS** L'augmenter jusqu'à s'assurer que le modèle a bien convergé et stagne en performance.

Pour des raisons de praticité et de temps de calcul, nous avons cependant dû limiter ces paramètres.

Pour d'autres paramètres plus spécifiques aux données et au modèle, nous avons pu explorer leur influence respective.

3.4.3. Remarque sur la qualité du modèle obtenu

Pour terminer, on remarquera que la fonction de coût d'un modèle neuronal n'est pas convexe. Ainsi, nous ne pouvons être certains d'avoir obtenu le meilleur modèle pour un jeu de paramètre donné.

Il aurait été judicieux et plus pertinent de faire pour chaque jeu de paramètres, plusieurs simulations pour ne conserver que la meilleure, ou observer les résultats moyens. Encore une fois notre temps et capacité de calcul se sont avérés limitants, et nous n'avons réalisé chaque optimisation du modèle qu'une fois.

3.5. Influence des paramètres

Nous allons passer en revue les principaux paramètres pouvant influencer sur le modèles, à savoir :

NO_CUT

VOCAB_SIZE

DROPOUT

Pour chaque situation, nous regarderons l'influence sur les performances du modèles, ainsi qu'en les croisant avec certains paramètres.

3.5.1. Influence de *NO_CUT*

On rappelle que si *NO_CUT* vaut 1, on ne conserve pas les séquences de longueurs supérieures à *MAXLEN*, sinon, on les conserve mais en les tronquant à la longueur *MAXLEN*.

Les résultats de nos différents essais sont alors résumé à la Figure 15.

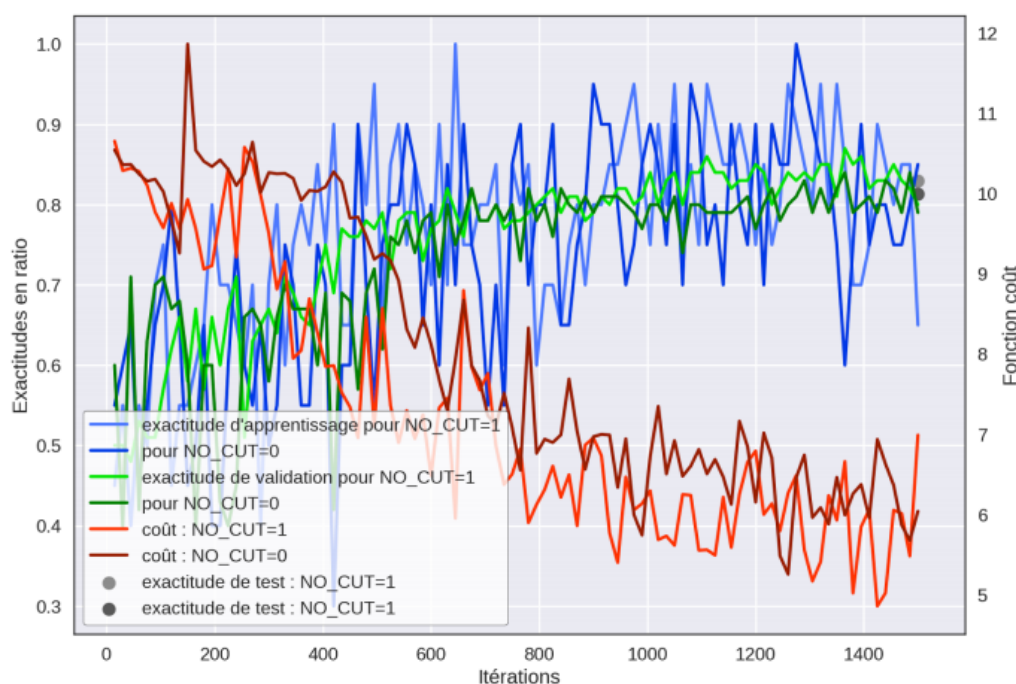


FIGURE 15 • Résultats obtenus sur l'influence de *NO_CUT*

Si la Figure 15 est chargée, il suffit d'avoir en tête que les métriques considérées sont au nombre de quatre :

- ⊙ *Exactitude d'apprentissage* Évaluée sur les batchs de taille *BATCH_SIZE*. Représentée en **bleu**.
- ⊙ *Exactitude de validation* Évaluée sur un ensemble de taille *VALIDATION_SIZE*. Représentée en **vert**.
- ⊙ *Fonction de coût* Ces dernières sont représentées en **rouges**.
- ⊙ *Exactitude de test* Ces dernières sont représentées en **gris**.

De plus, les courbes qui se rapportent au modèle avec $NO_CUT=1$ sont en clair, celles pour $NO_CUT=0$ en foncé.

On peut déjà voir que la taille de l'ensemble considéré joue beaucoup dans la variabilité des exactitudes. Les batches d'apprentissage de taille $BATCH_SIZE \sim 20$ donnent des exactitudes très variables, alors que pour l'ensemble de validation, de taille 100 , on a des exactitudes beaucoup moins variables.

L'exactitude de test, effectuée sur 8000 exemples de la base de test, est la métrique censée être la plus fiable pour évaluer la qualité d'un modèle.

Pour observer les différences entre les courbes claires et foncées, on peut enlever de la figure les exactitudes d'apprentissage, qui n'apportent pas d'information particulière par rapport aux autres. On obtient alors la Figure 16.

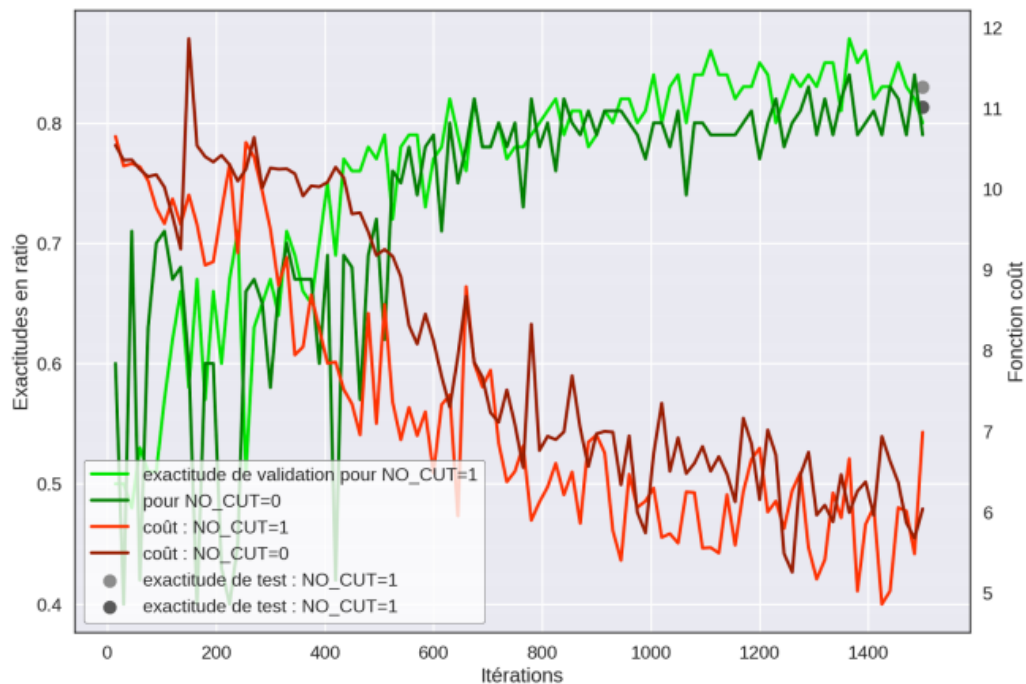


FIGURE 16 • Résultats obtenus sur l'influence de NO_CUT pour les grandeurs les plus importantes

Nous pouvons alors observer que la fonction de coût pour $NO_CUT=1$ est majoritairement inférieure à celle pour $NO_CUT=0$, et de même l'exactitude de validation et de test sont supérieures pour $NO_CUT=1$.

Cela nous amène à penser que considérer les séquences trompées n'apporte pas grand-chose, d'autant plus que pour $MAXLEN=500$, elles ne représentent plus qu'autour de 10% des séquences.

3.5.2. Influence de $VOCAB_SIZE$

Comme pour le cas précédent, les résultats sont résumés à la Figure 17.

On remarque que le modèle le plus performant ici est le modèle pour $VOCAB_SIZE=5000$. Il a la fonction de coût qui prend les valeurs les plus basses, et l'exactitude de test la plus grande.

Il est difficile de voir si le modèle avec $VOCAB_SIZE=5000$ obtient de meilleurs résultats qu'avec 1000 ou 10000. En effet, la valeur de 5000 est d'une part un compromis entre un biais trop important pour 1000, et un sur-apprentissage pour 10000. Mais également, on remarque que la variabilité intrinsèque du modèle indépendamment de ce paramètre, est trop grande pour pouvoir tirer des conclusions à partir d'un essai.

3.5.3. Influence de $DROPOUT$

Intérêt du drop-out Les réseaux neuronaux sont sensibles au sur-apprentissage, et donc il est important d'employer des techniques de régularisation. Une technique qui s'est révélée très efficace est celle du drop-out.

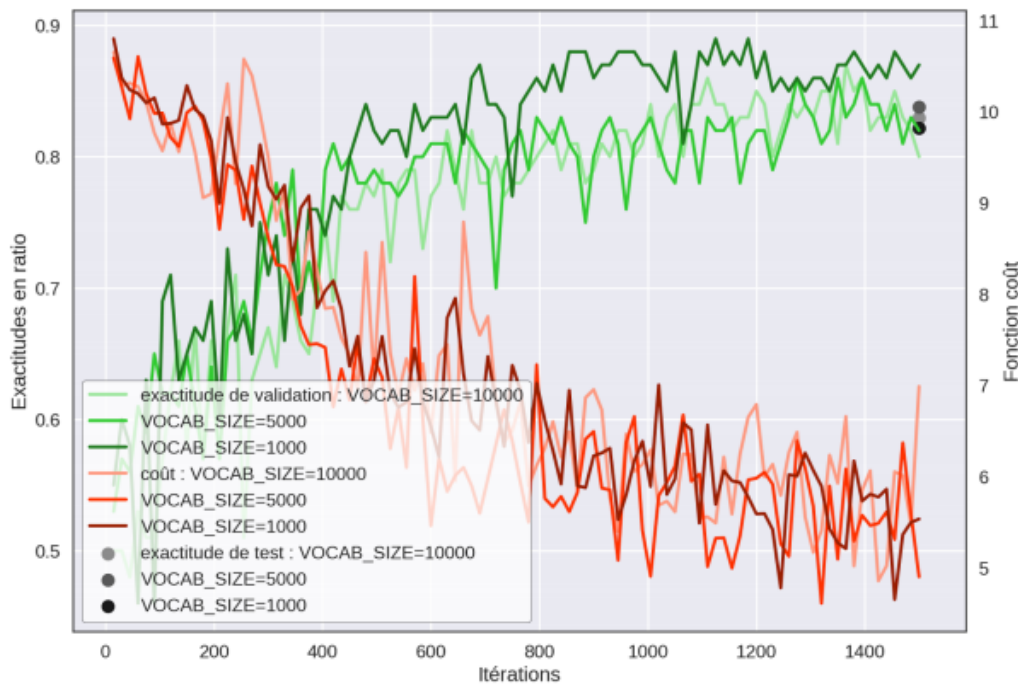


FIGURE 17 • Résultats obtenus sur l'influence de *VOCAB_SIZE*

Fonctionnement du drop-out Cette méthode revient à procéder aléatoirement, on ne laisse ainsi passer qu'une proportion définie de sorties du modèle, et on ne laisse entrer qu'une proportion des entrées du modèle; les autres cellules sont mises à zéro. Cela force ainsi le réseau à ne pas spécialiser une cellule de calcul que pour un seul usage, mais à multiplier les cellules qui produiront les mêmes sorties, ceci dans le but d'avoir un modèle plus robuste.

Cependant, cette technique *dropout* oblige à avoir un nombre de cellules considérées beaucoup plus grand.

Essais sur l'influence de DROPOUT Ainsi nous avons d'abord utilisé un modèle avec *DROPOUT*=0.5, et *RNN_HIDDEN* inchangé par rapport au modèle sans *drop-out*, à 200.

Dans un second temps, nous avons augmenté *RNN_HIDDEN*, en lui faisant prendre les valeurs 400, 500 et 600. Les résultats sont présentés à la Figure 18.

On voit que la technique du *drop-out* ne permet pas d'atteindre des performances similaires aux modèles sans *drop-out* avec le même nombre d'itérations. Cependant, les modèles avec régularisation par *drop-out* prennent toujours plus de temps à converger.

Les performances des modèles avec *drop-out* ont donc des performances dégradées mais similaires, avec une exactitude de test autour de 0,78.

On remarque alors que pour le modèle avec *RNN_HIDDEN* = 600, les performances s'effondrent. Pour comprendre ce phénomène, nous pouvons observer l'évolution des différentes fonctions de coût sur la Figure 19.

On remarque que contrairement aux autres fonctions de coût, celle pour *drop-out* et *RNN_HIDDEN*=600 ne converge jamais vraiment, elle stagne. Cela peut être dû au trop grand nombre de paramètres à optimiser, ou à une valeur de *LEARNING_RATE* trop grande.

Une étape supplémentaire qui aurait pu être testée serait d'augmenter *BATCH_SIZE*, pour avoir une descente de gradient plus lente mais moins stochastique. On pourra aussi augmenter le nombre d'itérations.

3.6. Détails sur la préparation des données

Nous avons produit un petit script qui prépare la base Amazon pour les mettre en entrée du modèle, de façon à pouvoir montrer les différentes étapes nécessaires.

Ces différentes étapes vous sont présentées dans l'ordre dans la suite de cette section.

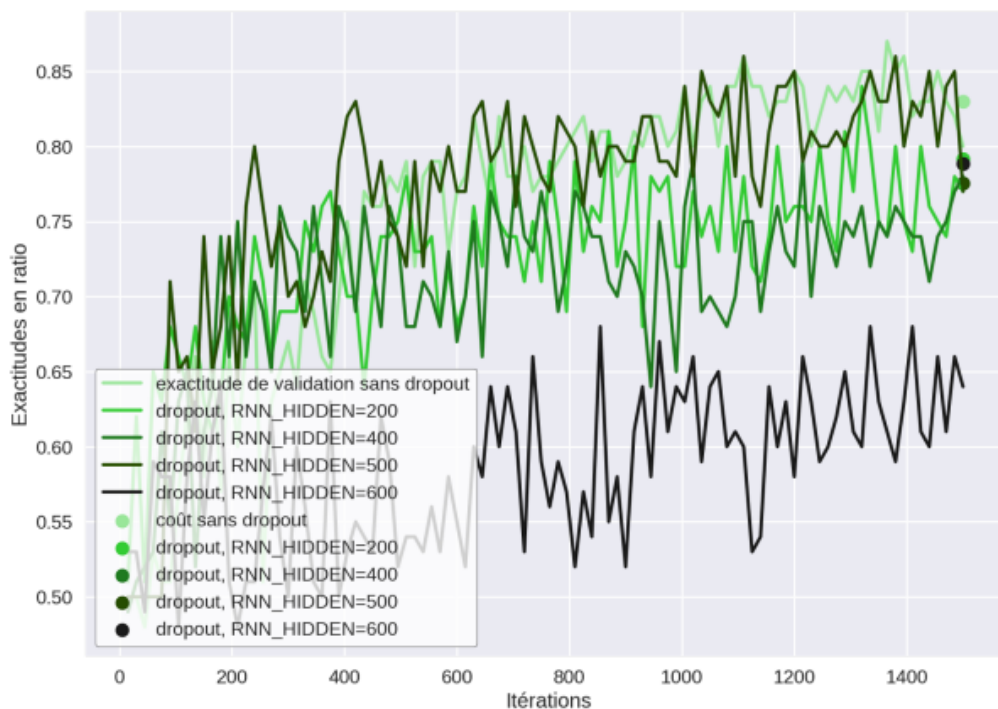


FIGURE 18 • Influence de *DROPOUT* pour différents réseaux

Tokenization des données Étape assez courante. Les signes de ponctuation seront considéré par le modèle comme des mots « normaux ».

Comptage des occurrences On compte le nombre d'occurrences de chaque mot. On crée ainsi un dictionnaire qui associe chaque mot avec son rang de fréquence (on commence à 2 : l'identifiant du mot le plus fréquent est donc de 2, pour le second plus fréquent il est de 3, etc...). Ensuite, on remplace dans les données chaque mot par son identifiant. On garde également pour des besoins ultérieurs d'interprétation le dictionnaire $\{mots:id\}$ et la liste des mots avec leurs nombres d'occurrences associés.

Donner une forme vectorielle aux mots Cela peut se faire en créant une table qui associe chaque mot avec sa représentation vectorielle. Dans notre cas, cette représentation était tirée des vecteurs donnés par le projet *GloVe*, et la table consistait en un dictionnaire.

La construction de ce dictionnaire se fait aussi simplement à parcourir toutes les lignes du fichier *.txt* téléchargé à partir du site internet du projet *GloVe*. Dans notre cas, on utilise celui créé à partir du contenu de Wikipedia en 2014. Chaque ligne est constitué d'une paire (*mot*, *représentation vectorielle du mot*).

À chaque fois que ce mot correspond à un mot de notre vocabulaire, on ajoute la paire clé-valeur *mot*-(*vecteur associé au mot*) au dictionnaire.

Attention, il n'est pas impossible (cela ne s'est cependant pas présenté ici) que l'accès aux vecteurs de ce dictionnaire, lors de la création du batch d'exemples, soit l'étape limitante au niveau de la vitesse de calcul lors de l'optimisation du réseau neuronal.

Utilisation de la fonction *embedding_lookup* Une autre possibilité pour la représentation vectorielle des mots consiste à utiliser la fonction *embedding_lookup* à partir d'une table d'*embedding* déjà initialisée.

Dans ce cas, ces représentations vectorielles seront considérées comme des paramètres optimisables du modèle, et seront d'abord initialisées aléatoirement, puis optimisées pour maximiser le pouvoir prédictif du modèle.

Alors pour chaque batch, si on cherche à obtenir les *embeddings* pour les identifiants contenus dans *tf*, on utilisera le code suivant :

```
1 embedding = tf.get_variable(name="embedding", shape=[EMBEDDING_SIZE,HIDDEN_SIZE],
2     initializer=tf.random_uniform_initializer(-1.0,1.0))
3 embedded_tokens = tf.nn.embedding_lookup(embedding, self.batch_input)
```

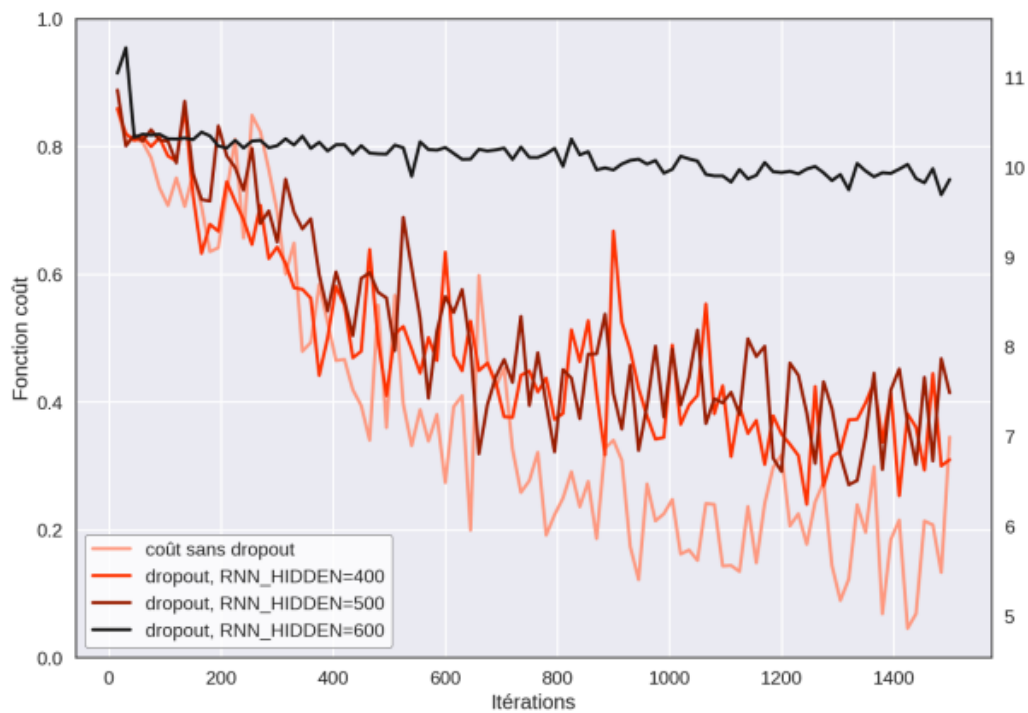


FIGURE 19 • Observation de la fonction de coût en fonction de *DROPOUT* sur différents réseaux

Le tenseur `embedded_tokens` est alors mis en entrée du modèle (la cellule LSTM) de la même manière que nous l'avons fait dans le cas de la base de données IMDB, avec le tenseur `x` construit explicitement par la fonction `generate_batch`. Cette représentation vectorielle n'a pas été implémentée dans un cas pratique.

Mots sans représentation vectorielle Tous les mots qui n'ont pas de représentation vectorielle (par exemple les mots qui n'étaient pas présents dans le fichier `.txt` de *GloVe*, ou ceux dont la fréquence est trop faible) sont remplacés par le token `UNK`, pour *unknown*. Il convient donc, dans notre cas où les vecteurs étaient fournis par *GloVe*, de donner une représentation vectorielle « custom » pour `UNK`. Nous avons choisi le vecteur constant multiplié par `0.1`, mais cela a peut-être pu être un choix perfectible et peut-être qu'un vecteur plus « identifiable » par le réseau comme $(1, 0, 0, \dots)$ aurait été plus judicieux.

Le token `UNK` a pour identifiant le nombre `1`.

Traitement des vecteurs restants En pratique, on décide d'une taille de vocabulaire (`VOCAB_SIZE`) et tous les vecteurs dont l'identifiant est supérieur à `VOCAB_SIZE` sont aussi remplacés par `UNK`.

3.7. Classifications binaires ou multi-variée

Notre modèle pour la base IMDB avait été optimisé avec des labels binaires. Pour la base de données Amazon, qui comporte des commentaires labellisés de 1 à 5 comme le nombre d'étoiles associés au commentaire, nous avons pris le parti de ne pas binariser nous-mêmes ces labels. En effet, il n'était pas évident de fixer un réel seuil. De manière générale, doit-on classer les commentaires notés 3 comme positifs (ie. avec ceux notés 4 et 5) ou alors comme négatifs (ie. avec ceux notés 1 et 2).

Les problèmes qui se posent alors sont les suivants :

- ⦿ **Favoriser l'équilibre des classes** Si on veut équilibrer les deux classes, il faut mettre 3 en négatif. Mais alors le choix n'est plus arbitraire, et de plus les classes seront toujours déséquilibrées.
- ⦿ **Échantillonner les avis positifs** Ainsi, la question se pose de savoir s'il ne faudrait pas conserver en négatif uniquement les commentaires notés 0 et 1 et échantillonner le même nombre de commentaires dans les notes 4 et 5 pour avoir des classes bien équilibrées et distinctes.

Pour mieux comprendre le problème, la répartition des labels de la plus petite des bases de commentaires d'Amazon, extraite sur les produits dans la catégorie « *Instruments de Musique* » vous est proposée Figure 20.

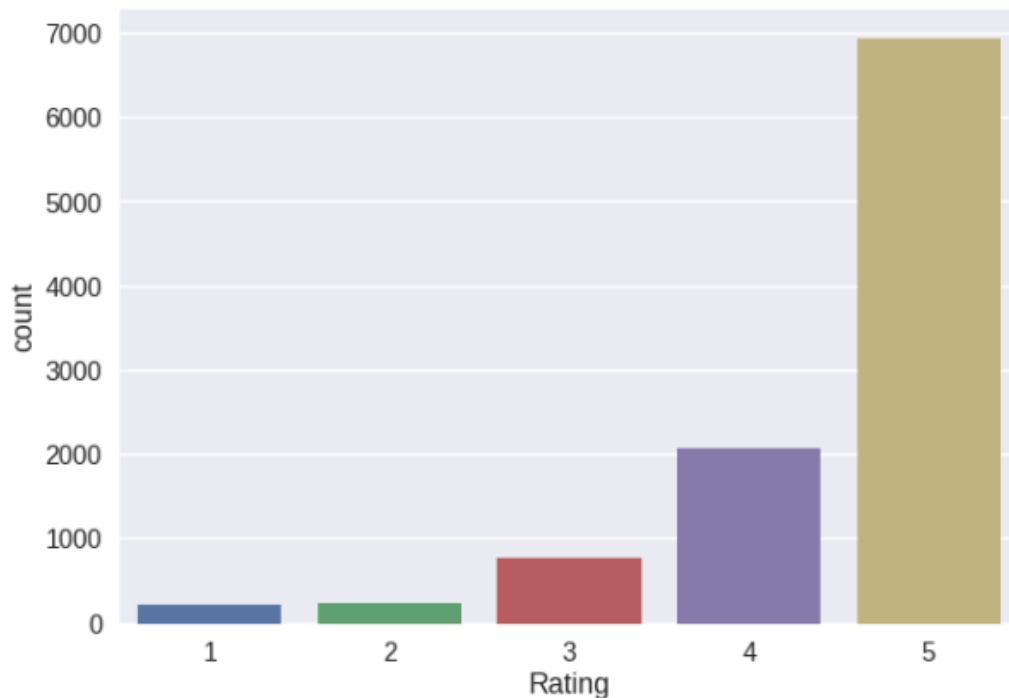


FIGURE 20 • Répartition des notes pour une des bases d'Amazon

Lorsque la base est plus conséquente (et souvent moins déséquilibrée, on peut en effet penser que les instruments de musique vendus, produit de luxe, sont souvent de bonne facture et que les critiques sont plus souvent bonnes que mauvaises), on peut avoir assez de commentaires notés 0 ou 1 pour constituer la base de commentaires négatifs, et sélectionner aléatoirement le même nombre dans les commentaires notés 5 pour constituer la base de commentaires positifs.

Quoiqu'il en soit, on remarque que si l'on ne veut pas se retrouver dans une situation où les retours négatifs soient presque des *outliers*, il y a une nécessité de bien vérifier la structure des données avant de les utiliser. Ainsi, les seuils fournis pour la préparation des données de la Section 2 sont indicatifs, mais il pourrait être intéressant de les faire varier en poussant plus loin l'analyse menée précédemment.

Conclusion

En conclusion de ce projet, nous pouvons donc avancer que nous avons réussi à remplir nos objectifs initiaux, à savoir :

- ⊙ Dresser un inventaire des différents champs d'applications du *deep learning* pour le NLP.
- ⊙ Constituer une base de données *réutilisable* en lien avec le sujet choisi. Comme nous l'avons vu, la base constituée représente plus de 4Go, et le code associé peut permettre de l'étendre au delà de 20Go.
- ⊙ Faire tourner un réseau de neurones sur nos données pour obtenir des résultats. Nous avons ainsi implémenté un réseau LSTM que nous avons initialement testé avec des données jouets avant de compléter le travail avec des données réelles (qui constituent un sous-ensemble de la base de données agrégée).

Cependant, ce travail dispose encore de nombreuses possibilités pour le future. Nous citons ainsi un certains nombre d'axes de recherche qu'il serait intéressant de fouiller en continuité de ce projet :

- ⊙ En conservant notre réseau, pousser plus loin l'optimisation des hyperparamètres et utiliser plus de données pour l'apprentissage (nous n'avons utiliser que 250Mo de données...). En particulier, il serait intéressant de fixer précisément les seuils pour la binarisation des sentiments associés aux documents.
- ⊙ Tester d'autres types de réseaux. Comme nous l'avons dit, l'objectif dans le traitement du texte est d'utiliser des réseaux avec des capacités de mémoire. Nous avons ainsi cité les LSTM, les MemNN et les DMN. Il est ainsi possible de tester d'autres solutions que les LSTM. En particulier, un type de réseau nommé NTM (*Neural Turing Machine*) semble prometteur dans ce domaine, avec sa capacité d'utiliser *en permanence* une mémoire tampon externe au réseau [32].

Ainsi, ce projet qui avait pour but de fournir un premier aperçu des possibilités semble avoir porté ses fruits. Dans une optique de continuité, tous nos éléments sont disponible sur Git comme nous l'avons mentionné en introduction.

Références

- [1] A. JIANG et C. ASAWA. Dynamic inference : Using dynamic memory networks for question answering. <https://cs224d.stanford.edu/reports/allan.pdf>, Consulté le 14 avril 2017.
- [2] C DE LICHY. Natural language inference, sentence representation and attention mechanism. <https://cs224d.stanford.edu/reports/Lichy.pdf>, Consulté le 14 avril 2017.
- [3] C. BILLOVITS et M ERIC. Using contextual information for neural natural language inference. <https://cs224d.stanford.edu/reports/billovits.pdf>, Consulté le 14 avril 2017.
- [4] R. GUPTA et N DESSAI. Extension to tree-recursive neural networks for natural language inference. <https://cs224d.stanford.edu/reports/GuptaDesai.pdf>, Consulté le 14 avril 2017.
- [5] Divers. Cs224d : Deep learning for natural language processing. https://cs224d.stanford.edu/reports_2016.html, Consulté le 14 avril 2017.
- [6] Q LIN et H. XIONG. Dynamic memory network on natural language question-answering. <https://cs224d.stanford.edu/reports/qian.pdf>, Consulté le 14 avril 2017.
- [7] Communauté Wikipédia. Rule-based machine translation. https://en.wikipedia.org/wiki/Rule-based_machine_translation, Consulté le 14 avril 2017.
- [8] Parlement européen. Textes adoptés par le parlement. <http://www.europarl.europa.eu/plenary/fr/texts-adopted.html>, Consulté le 14 avril 2017.
- [9] Communauté Wikipédia. Statistical machine translation. https://en.wikipedia.org/wiki/Statistical_machine_translation, Consulté le 14 avril 2017.
- [10] Q. V. LE et M. SCHUSTER. A neural network for machine translation at production. <https://research.googleblog.com/2016/09/a-neural-network-for-machine.html>, Consulté le 14 avril 2017.
- [11] O. VINYALS et Q. V. LE I. SUTSKEVER. Sequence to sequence learning with neural networks. <https://papers.nips.cc/paper/5346-sequence-to-sequence-learning-with-neural-networks.pdf>, Consulté le 14 avril 2017.
- [12] C. OLAH. Understanding lstm networks. <http://colah.github.io/posts/2015-08-Understanding-LSTMs/>, Consulté le 14 avril 2017.
- [13] B. TUROVSKY. Found in translation : more accurate, fluent sentences in google translate. <https://blog.google/products/translate/found-translation-more-accurate-fluent-sentences-google-translate>, Consulté le 14 avril 2017.
- [14] L. YAO et D. LIU. Wallace : Author detection via recurrent neural networks. <https://cs224d.stanford.edu/reports/YaoLeon.pdf>, Consulté le 14 avril 2017.
- [15] Project Gutenberg. Free ebooks by project gutenberg. <https://www.gutenberg.org>, Consulté le 14 avril 2017.
- [16] D. RHODES. Author attribution with cnn's. <https://cs224d.stanford.edu/reports/RhodesDylan.pdf>, Consulté le 14 avril 2017.
- [17] A. PERELYGIN et al. R. SOCHER. Deeply moving : Deep learning for sentiment analysis. <https://nlp.stanford.edu/sentiment/>, Consulté le 14 avril 2017.
- [18] A. PERELYGIN et al. R. SOCHER. Recursive deep models for semantic compositionality over a sentiment treebank. https://nlp.stanford.edu/~socherr/EMNLP2013_RNTN.pdf, Consulté le 14 avril 2017.
- [19] Q. LE et T. MIKOLOV. Distributed representations of sentences and documents. <https://arxiv.org/pdf/1405.4053.pdf>, Consulté le 14 avril 2017.
- [20] C. OLAH. Understanding lstm networks. <https://colah.github.io/posts/2015-08-Understanding-LSTMs/>, Consulté le 14 avril 2017.
- [21] J. HONG et M. FANG. Sentiment analysis with deeply learned distributed representations of variable length texts. <https://cs224d.stanford.edu/reports/HongJames.pdf>, Consulté le 14 avril 2017.

- [22] Équipes de Stanford. Large movie review dataset. http://ai.stanford.edu/~amaas/data/sentiment/aclImdb_v1.tar.gz, Consulté le 14 avril 2017.
- [23] Challenge Kaggle. Sentiment analysis on movie review. <https://www.kaggle.com/c/sentiment-analysis-on-movie-reviews/data>, Consulté le 14 avril 2017.
- [24] N. SANDERS. Twitter sentiment corpus. <http://www.sananalytics.com/lab/twitter-sentiment/>, Consulté le 14 avril 2017.
- [25] Particulier. Twitter sentiment analysis training corpus. <http://thinknook.com/twitter-sentiment-analysis-training-corpus-dataset-2012-09-22/>, Consulté le 14 avril 2017.
- [26] Équipes de Stanford. Sentiment treebank. <http://nlp.stanford.edu/sentiment/treebank.html>, Consulté le 14 avril 2017.
- [27] Challenge Kaggle. Umich si650 - sentiment classification. <https://inclass.kaggle.com/c/si650winter11/data>, Consulté le 14 avril 2017.
- [28] J. MC AULEY. Amazon product data. <http://jmcauley.ucsd.edu/data/amazon/>, Consulté le 14 avril 2017.
- [29] R. SOCHER et C. D. MANNING J. PENNINGTON. Glove : Global vectors for word representation. <https://nlp.stanford.edu/projects/glove/>, Consulté le 14 avril 2017.
- [30] G. DINU et G. KRUSZEWSKI M. BARONI. Don't count, predict! a systematic comparison of context-counting vs. context-predicting semantic vectors. <http://www.aclweb.org/anthology/P14-1023>, Consulté le 14 avril 2017.
- [31] I. SUTSKEVER et al. T. MIKOLOV. Distributed representations of words and phrases and their compositionality. <https://papers.nips.cc/paper/5021-distributed-representations-of-words-and-phrases-and-their-compositionality.pdf>, Consulté le 14 avril 2017.
- [32] G. WAYNE et I. DANIHELKA A. GRAVES. Neural turing machines. <https://arxiv.org/pdf/1410.5401v2.pdf>, Consulté le 14 avril 2017.