Usage des Réseaux Neuronaux Récurrents LSTM en Traitement Automatique du Langage

Bastien Liétard

Mai 2020

L'usage de plus en plus fréquent des réseaux de neurones artificiels (RNA) est un tournant récent en apprentissage automatique, en intelligence artificielle et en traitement automatique du langage (TAL) et leur usage se répand très rapidement. En TAL les données prennent souvent la forme de séquences ordonnées d'entrées (caractères, mots, etc.) et ces séquences constituent une problématique particulière. Les réseaux Long-Short Term memory (LSTM) ont été justement conçus pour la prédiction séquentielle. Dans ce document, nous présenterons d'abord les RNA et les problèmes liés aux séquences puis nous développerons une description théorique des LSTM. Enfin, nous mettrons en application un "embedder" pour essayer de comprendre la représentation du langage par les LSTM.

Mots-clés : Apprentissage automatique, Traitement Automatique de Language, *Word embedding*, Réseaux de neurones artificels, LSTM

Université de Lille

Sous la direction de Mikaëla Keller L3 MIASHS SL

Table des matières

Liste des abréviations et acronymes utilisés :

— RNA : Réseaux de Neurones Artificiels *(EN : Artificial Neural Network (ANN))*

— RNR : Réseaux de Neurones Récurrents *(EN : Recurrent Neural Network (RNN))*

— TAL : Traitement Automatique du Langage *(EN : Natural Langage Processing (NLP))*

— LSTM : *Long-Short Term Memory*

— LM : *Language Model* ; en français : Modélisation du Langage

— biLM : *Bidirectional Language Model* ; en français : Modélisation Bidirectionnelle du Langage

1 Introduction : vers les LSTM

Le Traitement Automatique du Langage (TAL), ou **Natural Langage Processing (NLP)** est un domaine qui s'est considérablement développé ces dernières années, avec des applications techniques nombreuses : traduction automatique, génération de texte, classification du discours, reconnaissance vocale, etc. En TAL comme dans l'ensemble du Machine Learning, l'enjeu est de trouver des méthodes rapides pour traiter un très grand nombre de données. Dans le cas particulier du TAL, il s'agit surtout de données textuelles (ou de signaux sonores si on travaille sur de la reconnaissance vocale par exemple). Un élément clé de ces travaux est donc la modélisation des données textuelles en objets mathématiques et par la suite le traitement de ces modélisations. L'**embedding** est une façon de réaliser cette modélisation, qui transforme l'objet textuel (souvent un mot, parfois un caractère, etc.) en vecteur de nombres réels pour le représenter, chaque coordonnée du vecteur étant un descripteur de l'objet textuel. Ensuite, le traitement de ces données est un traitement algorithmique et mathématique.

Ce travail abordera une méthode de traitement des données en TAL. Dans les techniques les plus utilisées ces dernières années en Machine Learning, les réseaux de neurones artificiels (RNA) sont parmi les plus efficaces pour des tâches variées, de classification, de prédiction, de génération, etc. En TAL en particulier, les utilisations sont nombreuses : de la classification de texte à la traduction automatique en passant par la reconnaissance vocale, etc. Cette efficacité est en partie dûe à plusieurs principes que nous allons détailler.

1.1 Les Réseaux de Neurones Artificiels (RNA)

Les Réseaux de Neurones Artificiels sont construit selon le paradygme du Neurone Formel [1] : de façon analogue aux neurones biologiques, chaque neurone artificiel (perceptron) est relié à d'autres dans le réseau (synapses). Chaque perceptron reçoit des informations en entrée (*input*), modulées par des coefficients (poids) et dispose d'une fonction d'activation qui crée la sortie du perceptron (*output*) à partir de cela. La figure 1 présente un perceptron.

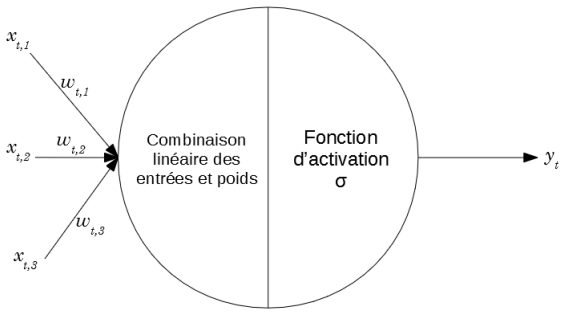

FIGURE 1 – Schema du fonctionnement d'un perceptron

Formellement : soient $x_{t,i}$ les k inputs avec $i \in [\![1; k]\!]$, soient $w_{t,i}$ avec $i \in [\![1; k]\!]$ les poids associées aux inputs, σ la fonction d'activation du perceptron et y_t l'ouput, on a :

$$y_t = \sigma(x_{t,1}.w_{t,1} + ... + x_{t,k}.w_{t,k}).$$

En posant X_t et W_t tels que $X_t = \begin{pmatrix} x_{t,1} \\ \vdots \\ x_{t,k} \end{pmatrix}$ et $W_t = \begin{pmatrix} w_{t,1} \\ \vdots \\ w_{t,k} \end{pmatrix}$, on a $y_t = \sigma(X_t.W_t)$. Notons bien que y_t est un scalaire dans le cas d'un unique perceptron mais un vecteur dès lors qu'on travaille avec un réseaux de neurones artificiels.

Un réseau de neurones artificiels est organisé en plusieurs couches de perceptrons. Chaque couche contient un certain nombre de perceptrons et chacun des n perceptrons de la couche est connecté aux m perceptrons de la couche suivante, comme l'illustre la figure 2. Un réseau de neurone suit un principe de **parallélisation des calculs**, permettant aux perceptrons de calculer leur output individuellement et simultanément (voir figure 2). La première couche du réseau qui reçoit les données est appelée **couche d'entrée** (*input layer* et celle qui est en fin de réseau et qui sort les prédictions faites par celui-ci est appelée **couche de sortie**. Les couches intermédiaires sont dites **couches cachées** (*hidden layers*). Les vecteurs de nombre qui circulent sur les connexions entre les couches cachées et les poids associés à ces connexions sont eux aussi dits "cachés" (*hidden vectors & hidden weights*)

1.2 Les réseaux de neurones récurrents

Il est parfois nécessaire de prendre en compte les résultats obtenus précédemment, lors d'étape de calcul antérieur. C'est par exemple le cas pour des tâches de générations de séquences, où il est essentiel de savoir ce qui a été précédemment généré pour calculer ce qui doit suivre. Pour cela, un réseau de neurones formels de suffit pas.

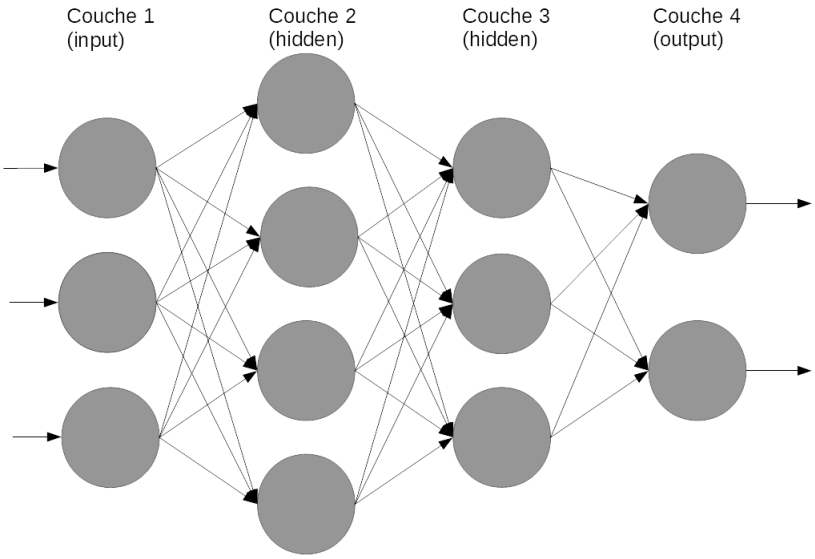

FIGURE 2 – Un réseau de neurones artificiels. La couche 1 est la **couche d'entrée**, les couches 2 et 3 sont des **couches cachées** et la couche 4 est la **couche de sortie**.

De plus, un problème majeur du travail avec les RNA sur les séquences est la variabilité de la longueur de celles-ci. Si les RNA sont très performants, ils ne fonctionnent, comme beaucoup d'algorithmes de machine learning, qu'avec des entrées de taille fixe (des vecteurs de dimension connue). Or, les séquences (ex : phrases, mots) sont, elles, des ensembles d'objets (respectivement mots et caractères) de taille très variable.

Pour pallier ce problème ont été mis au point les réseaux de neurones récurrents (RNR). La différence avec les RNA classiques est que : dans un RNA, les perceptrons calculent l'output uniquement à partir de leur input ; dans un RNR, pour le calcul à une étape t, chacun des perceptrons prend également en compte ce qu'il a calculé à l'étape $t-1$. On peut alors, comme le montre la figure 3, se représenter le calcul de l'output à l'étape t par un perceptron comme une succession de calcul par t perceptrons donc chacun envoie son output à son successeur. En réalité cependant ces t calculs sont réalisées par le même perceptron en t étapes. La figure 3 montre une topologie particulière de RNR, celle dite du *many-to-many*, car il y a plusieurs inputs (une à chaque étape) pour plusieurs sorties (une à chaque étape). Il existe d'autres architectures de RNR :

— *many-to-one* : plusieurs entrées au fil des étapes mais une seule output (à la fin du calcul). Les sorties des étapes intermédiaires (sur la figure 3 : 1, 2, ... $t-1$) ne servent qu'au calcul de l'étape suivante.

— *one-to-many* : une seule entrée x lors de la première étape de calcul : pour toutes

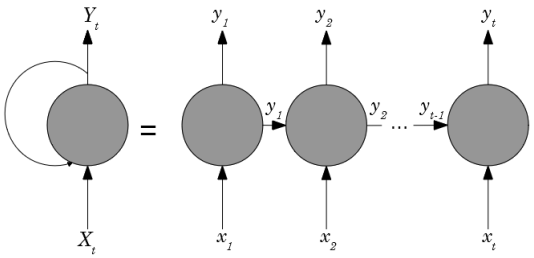

FIGURE 3 – Modélisation du comportement d'un perceptron récurrent. *Notons bien que ce ne sont pas plusieurs perceptrons, mais bien un unique perceptron à différentes étapes de son calcul.*

> les autres étapes le perceptron se sert uniquement de ce qu'il a calculé à l'étape précédente. Dans cette architecture, il y a une output à chaque étape et donc globalement plusieurs outputs (autant qu'il y a d'étapes).

Il faut noter également qu'un RNR en *many-to-many* peut ne pas avoir un nombre d'inputs égal au nombre d'outputs. C'est le cas des systèmes *encoder/decoder* sur lesquels nous reviendrons plus tard.

Pour le travail sur les séquences en TAL, un perceptron récurrent qui va devoir traiter une séquence fera autant d'étape qu'il en a besoin pour parcourir tous les éléments de la séquence et pourra ainsi résoudre le problème de variabilité de la dimension de l'input.

1.3 Les problèmes des dépendances longues

Si les RNR permettent une flexibilité sur la taille de l'entrée, taille qui n'a plus besoin d'être connue, ils sont cependant soumis à un autre problème. En théorie, un RNR n'a aucune difficulté à garder et prendre en compte une information, une input ou un calcul précédant, sur le long terme. On appelle ce besoin de garder longtemps une information en mémoire une dépendance longue. Dans la pratique cependant, les informations éloignées ont tendance à disparaître dans les calculs. Cela a été mis en évidence par Hochreiter en 1991 [2] et Bengio et al. [3] en 1994.

Il faut pour cela comprendre la méthode la plus commune d'apprentissage du modèle : la rétro-propagation du gradient. Dans un RNA, la justesse des calculs par rapport à la réalité dépend de la force des connexions synaptiques entre perceptrons, donc des poids

w_i de chacun des perceptrons du réseau. Il faut donc trouver une façon d'adapter au mieux les poids sur un ensemble d'entraînement (c'est à dire un ensemble de données pour lequel on connaît les résultats attendus ; pour de la traduction par exemple, un ensemble de phrases en français associées à leur traduction en anglais.). Sur cet ensemble d'entraînement, on doit minimiser le taux d'erreurs commises par le modèle (le RNA), ce qui revient à minimiser une fonction de coût. La méthode de rétro-propagation du gradient consiste à régler les poids des couches les plus élevées (la couche de droite sur la figure 2), de les ajuster pour minimiser le taux d'erreur, puis de faire pareil sur les couches inférieures (plus à gauche sur la figure 2) et ainsi de suite.

Cependant, Hochreiter [2] montre que dans le cas des RNA avec un nombre de couches élevé ou dans les RNR, une méthode d'apprentissage du modèle par rétro-propagation du gradient ne fonctionnerait pas à cause de la disparition du gradient ou bien une explosion de celui-ci, selon le caractère exponentiel de la fonction d'activation. Bengio et al. [3] montrent le lien entre ces problèmes et les dépendances longues dans les séquences. De fait ils montrent donc l'inefficacité des RNR "classiques" à maintenir ces dépendances longues.

En 1997, Hochreiter et Schmidhuber proposèrent une première approche d'un RNR capable de garder l'information sur le long terme : le *Long-Short Term Memory network*, ou LSTM.

2 Long-Short Term Memory et apprentissage "séquence vers séquence"

L'architecture proposée par Hochreiter et Schmidhuber en 1997 ne sera pas la seule utilisée pour les LSTM et sera par la suite modifiée et améliorée. Nous allons commencer par la présenter - notamment sa caractéristique principale, le *Constant Error Carrousel* - mais nous discuterons principalement la formulation proposée par Graves en 2013, utilisée de nombreuses fois par la suite, notamment par Sutskever, Vinyals & Le en 2014.

2.1 Le modèle LSTM dans la littérature

Dans leur article intitulé *Long Short Term Memory* de 1997 [4], Hochreiter et Schmidhuber partent des résultats développés dans la thèse du premier en 1991, à savoir la preuve

de la disparition du gradient d'erreur globale. Faisant état de ces difficultés, ils proposent une solution permettant de stabiliser les signaux d'erreurs, le *Constant Error Carrousel (CEC)*. D'abord ils connectent un unité (un perceptron) avec lui-même (faisant de lui une unité d'un RNR), puis ils posent deux conditions : d'une part, la fonction d'activation d'une unité doit être linéaire et d'autre part, l'activation de l'unité doit être constante, de sorte à ce que l'équation suivante soit vérifiée :

$$y_i(t+1) = \sigma(w_{ii} y_i(t)) = y_i(t)$$

pour $y_i(t)$ l'output de l'unité considérée à l'étape t, w_{ii} le poids associé à la connexion de récurrence de l'unité, de son output vers son input et σ la fonction d'activation (linéaire donc) de l'unité .

Par la suite, ils ajoutent à l'unité incluant le CEC une *input gate unit* et une *output gate unit* afin de contrôler les données entrantes et sortantes du CEC. Nous allons détailler le fonctionnement de ces *gates* par la suite. Ils appellent l'unité résultante une *memory cell*. Les signaux d'erreurs capturé dans la *memory cell* lors de l'entrainement du réseau ne peuvent varier, alors l'*input (output) gate* sert à contrôler quels signaux doivent être captés (relâchés) par la cellule en les échelonnant de manière appropriée. Pour que leur CEC respecte les contraintes données, Hochreiter et Schmidhuber fixe $w_{ii} = 1.0$ et définisse σ comme la fonction identité, à savoir $\sigma(X) = X$.

En 2013, Alex Graves utilise les LSTM dans un article sur la prédiction de séquences [5]. Il ajoute à la formulation originale de Hochreiter et Schmidhuber la modification proposée par Gers et Schmidhuber en 2000 [6], à savoir l'ajout d'une *forget gate*. Les deux chercheurs constatent en effet que lors de l'apprentissage sur un flux continu de données, la *memory cell* finit par saturer et perd alors son effet de memoire selective. Si, en principe, un réseau LSTM devrait pouvoir réinitialiser seul son *cell state* (l'état de la *memory cell*), en pratique il n'y parvient pas car les valeurs d'activation précises pour réinitialiser le *cell-state* sont difficiles à apprendre. Gers et Schmidhuber proposent alors de remplacer la connexion de poids $w_{ii} = 1.0$ par une *forget gate* basée sur le même fonctionnement que les *input et output gates*.

Dans son article [5], Graves résume donc la formulation d'une unité LSTM à ces fonctions (formulation présentée graphiquement comme dans la figure 4) :

$$i_t = \sigma(W_{xi} x_t + W_{hi} h_{t-1} + W_{ci} c_{t-1} + b_i)$$
$$f_t = \sigma(W_{xf} x_t + W_{hf} h_{t-1} + W_{cf} c_{t-1} + b_f)$$

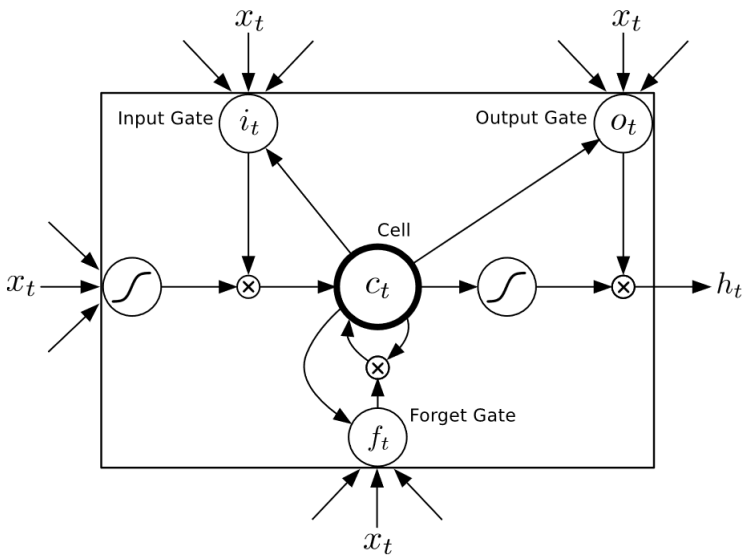

FIGURE 4 – Représentation de la *Long Short-term Memory Cell* utilisée par Alex Graves en 2013, extraite de l'article cité. Les points de rencontre avec des croix représentent des opérateurs de **multiplication**, tandis que les cercles contenant la figure d'une sigmoïde représentent des opérateurs de fonction **tangente hyperbolique**

$$
\begin{aligned}
c_t &= f_t c_{t-1} + i_t \tanh(W_{xc} x_t + W_{hc} h_{t-1} + b_c) \\
o_t &= \sigma(W_{xo} x_t + W_{ho} h_{t-1} + W_{co} c_t + b_o) \\
h_t &= o_t \tanh(c_t)
\end{aligned}
$$

où σ est la fonction logistique sigmoïde et i, f, o et c sont respectivement *l'input gate, la forget gate, l'output gate* et le *cell state*, tous de même dimension que le *hidden vector* h.

On remarque que les tois *gates* sont en réalité l'image par la fonction logistique sigmoïde de la combinaison linéaire des données entrées (x_t), de la sortie du perceptron à l'étape t-1 (h_{t-1}) et le *cell-state* (c_{t-1} pour *input* et *forget* et c_t pour l'output), chacun de ces termes étant modulé par les poids associés. À chaque fois, on ajoute un biais. On notera de plus qu'un biais b_c est également ajouté au calcul de l'actualisation du *cell-state*.

En analysant cette formulation, on comprend alors que le LSTM est bien un perceptron récurrent (car on utilise h_{t-1} à l'étape t) et ainsi doté de sa cellule mémoire construite autour du CEC (et dont l'état est contenu dans c_t), il permet la conservation d'informations sur le long terme, résolvant ainsi le problème des dépendances longues. Aussi, Pascanu et al. confirment en 2012 [7] que le LSTM permet l'évitement du problème de disparition du gradient soulevé par Hochreiter en 1991 [2]. Maintenant que nous avons analysé la

construction d'une unité LSTM, intéressons nous maintenant à son utilisation.

2.2 Quelques utilisations du modèle LSTM

Les LSTM connaîtront dans les années suivantes et notamment dans les années 2010 un grand succès dans diverses tâches, ce qui conduira à un emploi massif de ceux-ci dans l'industrie du TAL (Google, Amazon, Apple, etc.), pour des applications comme la reconnaissance de texte manuscrit, la reconnaissance vocale, la traduction automatique, etc. Cet emploi massif des LSTM s'explique par l'efficacité de ce modèle à surmonter des difficultés jusqu'alors jamais dépassées dans le travail sur les séquences en TAL. Dans cette section nous allons d'abord revenir sur les démonstrations par Hochreiter et Schmidhuber de la réussite inédite des LSTM [4], puis nous décrirons les résultats des expériences menées par Graves [5] sur la génération de texte et d'écriture manuscrite.

Dès leur origine, les LSTM ont montré leur efficacité pour des problèmes qu'aucun autre RNR n'avait réussi à résoudre jusque là. Hochreiter et Schmidhuber développent cela dans la section 5 de leur article de 1997 [4] au travers de différentes expériences pour montrer l'efficacité des LSTM face à différents problèmes, comme notamment la prise en compte des dépendances longues avec des valeurs continues ou non ou bien la préservation des relations d'ordre temporelle entre des informations. Pour la première expérience (section 5.4), Hochreiter et Schmidhuber donnent pour consigne d'effectuer la somme de deux éléments X_1 et X_2 choisis au hasard dans des séquences de taille T très grande (jusqu'à 1000 éléments par séquence), X_1 et X_2 étant reliés par une marque. La somme ciblée est obtenue avec une erreur et projetée dans l'intervalle par le calcul suivant : $0.5 + \frac{X_1+X_2}{4.0}$. Les résultats de l'expérience montrent que le réseau utilisé (3 couches de LSTM) parvient à accomplir cette tâche avec un taux d'erreur inférieur à 0.01%. De la même manière, dans la section 5.5, les deux chercheurs montrent que le réseau parvient à résoudre le même problème en ne ciblant pas cette fois la somme mais le produit : $X_1 \times X_2$. La solution visée n'est ici pas directement intégrable et échappe donc aux avantages directs qu'on pourrait attribuer à l'intégrabilité du CEC. Le réseau y parvient avec une erreur moyenne carrée inférieure à 0.026%. Enfin, dans la section 6, Hochreiter et Schmidhuber s'intéressent à une tâche de classification de séquence de caractère selon l'ordre temporel d'apparition de 2 (ou 3) caractères spécifiques, chacun étant un **X** ou un **Y**. Le réseau doit réussir à classer les séquences selon l'ordre dans

lequel apparaissent les 2 (ou 3) caractères ciblés. Pour 2 caractères spéciaux par exemple, il y a donc 4 classes : X - X, X - Y, Y - X et Y - Y. Finalement, après entraînement, le réseau y parvient avec un taux d'erreur inférieur à 0.1%. Les deux chercheurs soulignent bien qu'aucun réseau de neurones artificiels n'avait précédemment réussi ces trois types de tâches.

Dès lors, on comprend que les LSTM sont des RNR plus adaptés aux tâches de Traitement Automatique du Langage. En effet, pour travailler efficacement sur les éléments du langage, il faut pouvoir préserver les liens sémantiques pré-existants entre les composantes d'un énoncé par exemple et ce sur le long terme. Des données linguistiques traitées par le réseau à l'étape t doivent pouvoir être gardées jusqu'au traitement d'autres données liées aux premières à l'étape $t + n$ et pouvoir prendre en compte l'ordre dans lequel les données sont apparues au réseau. Dans la seconde moitié de son article de 2013, Alex Graves [5] utilise les LSTM pour procéder à la génération de séquences linguistiques. À l'aide d'un réseau LSTM, il entraîne son modèle sur des corpus de séquences pour répondre à 3 problèmes de TAL : la génération de texte, la génération d'écriture manuscrite et la synthèse d'écriture manuscrite. Pour ce qui est de la génération de texte, Graves entraîne d'abord son réseau sur le corpus de texte TreeBank, composé de portions du Wall Street Journal annotées selon leur structure syntaxique. Le réseau doit prédire des mots à patir d'un début de séquence et sa prédiction est comparée au mot qui suivait réellement. Graves constate alors qu'un réseau traitant des séquences de mots est légèrement plus efficace qu'un réseau traitant des séquences de caractères pour cela. Pour une seconde expérience, Graves entraîne un réseau LSTM à générer une page Wikipedia à partir de rien. Les résultats montrent qu'un réseau LSTM est capable de créer une page respectant la structure de Wikipedia et dont les phrases sont, dans l'ensemble, syntaxiquement correctes. Le texte généré est donc plutôt cohérent. Le résultat n'est pas parfait mais il montre plusieurs choses : le réseau est capable de maitriser des éléments de ponctuations de base dans un texte, et il est parvient à utiliser correctement des paires de caractères ouvrants et fermants, comme des parenthèses. L'indication d'une fermeture de parenthèse ne pouvant venir du sens du texte, c'est bien grâce à la *memory-cell* que le réseau LSTM a appris qu'une parenthèse droite suivait toujours une parenthèse gauche. Pour la prédiction d'écriture manuscrite, Graves entraîne son réseau sur un jeu de données de retranscription numérique d'écriture manuscrite. Le réseau LSTM parvient à reproduire un style manus-

crit, même s'il est difficile à lire. On distingue correctement les caractères et le texte écrit pourrait ressembler à de l'anglais réel. Pour la synthèse de texte manuscrit, le réseau était entraîné sur un jeu de données comportant des associations phrases écrites - tracés de crayon correspondants. Une fois entraîné, le réseau parvient à décrire un tracé simulant une écriture manuscrite pour une phrase donnée. En ajoutant un biais, Graves parvient à faire gagner à l'écriture manuscrite en lisibilité, au risque de paraître trop artificielle si le biais est trop important.

Parmi les autres domaines du TAL, la traduction automatique est sûrement l'un de ceux pour qui le développement des LSTM a marqué un tournant, notamment dans l'industrie. En 2016, pour l'annonce de la mise en place de la *Google Neural Machine Translation*, Google annonce que ce nouveau modèle, basé sur les LSTM, a fait baisser de 60% les erreurs de traductions [8]. Par la suite, nous allons étudier la modélisation d'un réseau LSTM pour la traduction automatique, ce qui nous permettra de nous poser plusieurs interrogations sur le fonctionnement des LSTM.

2.3 Les LSTM pour la traduction automatique : encoder et décoder

En 2014, Sutskever, Vinyals et V. Le proposent, dans leur article *Sequence to Sequence Learning with Neural Networks* [9], un modèle de réseau de neurones LSTM permettant de réaliser de la traduction automatique. Après avoir exposé l'architecture de leur modèle, Sutskever et al. comparent les performances de leur réseau aux performances de Google Translate (de l'époque, c'est-à-dire avant le déploiement de la *Google Neural Machine Translation*. Dans cette section, nous résumerons leur article et celui-ci nous amènera à quelques interrogations sur les LSTM.

Il faut d'abord comprendre que Sutskever et al. se confrontent au problème de l'apprentissage dit "séquences vers séquences". Cela consiste à disposer de données d'entrées sous forme d'une séquence et à vouloir créer en sortie une autre séquence, toutes deux de tailles variables et potentiellement différentes. Nous avons déjà décrit le problème posé par les séquences d'entrées avec les réseaux de neurones artificiels, à savoir les possibles dépendances temporelles qui existent entre les éléments de ces séquences. De fait, les chercheurs s'orientent donc naturellement vers les RNR et en particulier les LSTM, choix que les auteurs justifient par "leur capacité à apprendre avec succès à partir de données

avec des longues dépendances temporelles", comme cela a été montré précédemment [4]. Cependant, la plus grosse difficulté dont les chercheurs font état est que l'apprentissage en séquences vers séquences impliquent de travailler sur des séquences d'entrées et de sorties de tailles inconnues et potentiellement différentes. L'objectif de leur article étant de proposer un modèle de traduction réaliste, ne consistant pas en une traduction mot à mot et donc avec une taille de séquence de mot en sortie différant de la taille de la séquence d'entrée. De plus, la plupart des réseaux de neurones y compris les RNR nécessitent de connaitre la taille de la séquence d'entrée et la taille de la séquence de sortie. Les chercheurs proposent donc l'architecture d'un réseau de neurones LSTM permettant de dépasser ces obstacles.

Puisque les réseaux de neurones artificiels n'acceptent en entrée que des vecteur, les auteurs réalisent d'abord un *"one-hot encoding"* sur leurs séquences d'entrée, c'est à dire qu'ils transforment chaque mot de la séquence d'entrée en un vecteur à n dimensions, n étant la taille du lexique de départ (c'est-à-dire un ensemble des mots uniques issus de la langue source). Chacun des mots est représenté par un vecteur de $n-1$ "0" et d'un unique "1" à la position i, signifiant que ce vecteur est associé au mot à la i-ème position du lexique.

Pour effectuer la tâche Sutskever, Vinyals et V. Le n'utilisent en réalité pas un mais deux ensembles de LSTM : un premier qui lit la séquence d'entrée et calcule une représentation vectorielle de celle-ci et une seconde qui utilise cette représentation vectorielle pour calculer la séquence de sortie. Grâce à cette stratégie, plus connue sous le nom d'*encoder-decoder*, ils dépassent le problème de la différence de taille entre les séquences d'entrée et de sortie. Le premier LSTM (*encoder*) va calculer une représentation de taille fixe et connue, et le second LSTM (*decoder*) va donc prendre une entrée de taille fixe et connue. Le but de l'ensemble de leur réseau LSTM est de "calculer la probabilité conditionnelle $p(y_1, ...y_{T'}|x_1, ...x_T)$, avec $(x_1, ...x_T)$ la séquence d'entrée de taille T et $(y_1, ...y_{T'})$ la séquence de sortie de taille T'. L'*encoder* calcule donc une représentation vectorielle v à partir de $(x_1, ...x_T)$ et le *decoder* calcule $\prod_{t=1}^{T'} p(y_t|v, y_1, ..., y_{t-1})$. Chaque distribution des $p(y_t|v, y_1, ..., y_{t-1})$ est calculée à l'aide d'une application de la fonction *softmax* à l'ensemble du lexique d'arrivée pour déterminer quels mots sont les plus suceptibles de servir dans la traduction à la position t.

Pour palier le problème de la taille inconnue des entrées et des sorties, Sutskever et

al. terminent chaque séquence par le sigle "<EOS>" (pour *"End of Sequence"*). Ainsi l'*encoder* calcule la représentation vectorielle v jusqu'à ce qu'il rencontre ce sigle, puis c'est le *decoder* qui prend le relais et calcule la probabilités pour les mots de servir à la traduction. Au sortir de la softmax, le modèle garde donc les traductions qui maximisent la probabilité. A l'aide d'un algorithme de *left-to-right beam search*, les chercheurs déterminent plusieurs hypothèses partielles de traductions complétées à chaque étape temporelle, c'est à dire chaque fois que le *decoder* sort une nouvelle *output* y_t. Dès lors que le sigle <EOS> est proposé au bout d'une hypothèse partielle, celle ci est considérée comme complète.

Pour montrer l'efficacité de leur système, ils le confrontent à un système de traduction statistique (SMT, *statistical machine translation*) qui servira de référence lors d'une tâche de traduction de l'anglais vers le français, à partir du corpus *WMT'14 English to French*. Ils comparent la qualité des traductions de leur système et du système de référence avec le BLEU score. Pour cette expérience, leur réseau est composé de 4 couches de 1000 unités LSTM. Ainsi le *word embedding* (c'est à dire la représentation vectorielle v de la séquence) est de taille 1000. Le lexique anglais (source) contient 160'000 mots, le lexique français (cible) en contient 80'000. Les sorties de la dernière couche de LSTM sont passées dans une fonction softmax pour calculer les probabilités. Les résultats de leur expérience montre que le réseau ainsi créé, bien qu'il n'établisse pas un record de performance dans le domaine, est plus efficace que le système de référence (avec un *BLEU score* de 34.8 contre 33.3 pour le SMT). Les auteurs notent d'ailleurs que c'est la première fois qu'un système de traduction utilisant les réseaux de neurones artificiels est plus efficace qu'un SMT. Les auteurs remarquent également que leur modèle est d'autant plus efficace s'ils renversent les séquences d'entrées (c'est à dire, pour une séquence d'entrée *'ABC'*, de fournir en entrée du réseau la séquence *'CBA'*). Ils supposent, sans pouvoir le montrer, que c'est parce que cela réduit l'éloignement temporel entre le mot où A est intégré par l'*encoder* et le moment où sa traduction est générée par le *decoder* et donc cela réduirait les problèmes de dépendances longues lors de l'entraînement du réseau par descente de gradient stochastique.

Cet article fournit un bon exemple d'usage des LSTM dans le travail sur la langue et notamment pour l'apprentissage en "séquence vers séquence". On comprend aussi que, pour réaliser une telle tâche, il est utile de créer un *word embedding*, c'est à dire une repré-

sentation vectorielle de taille suffisamment réduite des mots (pour éviter des problèmes comme le "Fléau de Dimension", expliqué par exemple dans l'introduction de Bengio et al. [10]). Simplement, il est beaucoup plus facile pour un réseau de neurones de travailler avec un embedding de taille 1000 qu'avec des vecteurs de taille 160'000. On peut alors se poser plusieurs questions, quant à l'embedding réalisé par le modèle de Sutskever et al. et de manière plus général, avec les représentations vectorielles de langage avec les LSTM. Notamment, à quel point et avec quelle précision les LSTM permettent-ils de modéliser le langage ? Peut-on avec les LSTM modéliser le langage dans un nombre suffisamment réduit de dimension et d'une telle manière que l'on puisse préserver dans cet embedding des petites nuances de la langue ?

Par la suite, nous allons discuter un des modèles de création de *word embeddings* les plus performants : ELMo, développé par des membres du *Allen Institute for AI*.

2.4 ELMo : des embeddings avec les LSTM

En 2018, Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee et Luke Zettlemoyer signent un article intitulé *Deep contextualized word representations* [11] dans lequel ils étudient une méthode de représentation des mots de la langue en *Deep Learning*. C'est dans ce cadre qu'ils proposent leur méthode de *word embedding* qu'ils nomment **ELMo** pour "Embeddings from Language Models".

Peters et al. rappellent tout d'abord l'enjeu d'une telle représentation des mots : en effet, les travaux comme ceux de Mikolov et al. [12] ont affirmé l'importance de la prise en compte du contexte linguistique lors de la création du *word embedding* et ont également montré le rôle clé d'embeddings judisciemment pré-entrainés dans les Modèles de Langage (LM) ; l'idée de ces LM étant de fournir une représentation numérique du langage naturel permettant de saisir au mieux les composantes de celui-ci. De ce fait, Peters et al. expliquent que la création d'un *word embedding* idéal se confronte à deux défis majeurs : représenter correctement les mots dans les caractéristiques syntaxiques et sémantiques, mais également saisir les variations de ces caractéristiques selon le contexte qui entoure le mot. Pour traiter ces deux problématiques, les auteurs proposent un *word embedding* qui calcule les représentations des mots comme des fonctions de la phrase entière dans laquelle le mot se situe et pas uniquement comme une fonction du mot lui même. Ainsi, on peut qualifier leurs représentations des mots de représentations "contextualisées". Pour

les calculer, ELMo utilisent un réseau de neurones récurrents LSTM pour laquelle les *outputs* de chacune des couches cachées servent à calculer la réprésentation finale en sortie de l'*output layer*.

Cependant, ELMo ne calcule pas le LM de la même manière que peut le faire, par exemple, l'*encoder* LSTM de Sutskever et al. [9]. Le réseau LSTM de ces derniers calculait la probabilité d'un *tokens* (ici les mots) t_k en fonction des t_i avec $i = 1, ..., k-1$, ce qui en fait donc un *Forward Language Model*, "*forward*" ("vers l'avant" en anglais) car les *tokens* sont lus et interprétés du plus ancien au plus récent, du premier au dernier. Par opposition, on peut définir un "Backward Language Model" comme un LM lit les tokens du plus récent au plus ancien, c'est-à-dire que pour calculer la probabilité du token t_k on prend en compte l'historique des t_i avec $i = k+1, ..., N$. Pour finir, on définit un *Bidirectional Language Model* (biLM) comme un LM qui prend en compte le contexte *passé* et *futur* d'un mot pour en calculer sa représentation. C'est cette stratégie que Peters et al. adoptent pour construire les ELMo embeddings.

Ils commencent avec un simple réseau LSTM à L couches. Soit une séquence de N tokens $(t_1, t_2, ...t_N)$. Dans le cas d'un *Forward LM*, le réseau calcule à chaque couche $j \in [\![1, L]\!]$ une représentation vectorielle dépendante du contexte passé, qu'ils notent $\overrightarrow{\mathbf{h}}_{k,j}^{LM}$. De la même manière, un *Backward LM* calcule à chaque couche $j \in [\![1, L]\!]$ la représentation dépendante du contexte futur $\overleftarrow{\mathbf{h}}_{k,j}^{LM}$. Ils définissent donc leur biLM comme ceci : pour un token t_k, ELMo calcule un ensemble de $2L+1$ représentations noté R_k et définit comme $R_k = \{\mathbf{x}_k^{LM}, \overrightarrow{\mathbf{h}}_{k,j}^{LM}, \overleftarrow{\mathbf{h}}_{k,j}^{LM} | j = 1, ..., L\}$ avec $\mathbf{x}_k^{LM}$ un embedding "primaire" des tokens à la couche d'entrée du réseau LSTM (cette représentation naïve est indépendante du contexte du token). La représentations ELMo de t_k contient toutes ces représentations selon les couches, c'est à dire :

$$R_k = \{\mathbf{h}_{k,j}^{LM} | j = 0, ..., L\} \quad \text{et} \quad \mathbf{ELMo}_k = E(R_k, \Theta_e)$$

avec $\mathbf{h}_{k,0}^{LM} = \mathbf{x}_k^{LM}$, $\mathbf{h}_{k,j}^{LM} = [\overrightarrow{\mathbf{h}}_{k,j}^{LM}, \overleftarrow{\mathbf{h}}_{k,j}^{LM}]$ et Θ_e les paramètres (poids) appris lors du pré-entraînement de ELMo. Peters et al. expliquent que chacune des couches permet de capter des informations différentes et que les couches les plus basses peuvent s'avérer intéressantes pour des tâches comme l'étiquetage morpho-syntaxique. Les dernières couches elles sont celles qui captent le mieux les nuances sémantiques et seraient plus à même de restituer les cas de polysémie. Les auteurs réalisent ensuite plusieurs expériences montrant qu'en utilisant ELMo comme *word embedding* devant des modèles de référence pour résoudre des

tâches, comme répondre à des questions, de l'étiquetage sémantique, ou bien déterminer la véracité d'une hypothèse d'après des prémices textuels donnés. Dans tous ces tests, l'utilisation des représentations ELMo comme *embedding* augmente le taux de réussite du modèle. Les configurations pré-entraînées de ELMo sont disponibles dans la librairie Python **AllenNLP** créée par les membres de *AllenAI*. L'architecture utilisée dans l'article est composée de 2 couches de biLSTM de 4096 unités, chacune des couches de LSTM projetant dans 1024 dimensions (512 pour le *Forward LSTM* et 512 pour le *Backward LSTM*).

ELMo fournit une représentation vectorielle des mots permettant selon les auteurs d'améliorer l'efficacité sur de nombreuses tâches de TAL. Les performances des embeddings ELMo seront parmi les meilleures pendant un moment, mais seront dépassées la même année par BERT [13], un autre biLM permettant des d'*embeddings* encore meilleurs (cette nouvelle méthode n'étant pas basée sur les LSTM, nous ne développerons pas ici son fonctionnement). Dans la prochaine section, puisqu'ELMo a réalisé de bonnes performances parmi les *word embeddings* utilisant les LSTM, nous allons nous en servir pour mener plusieurs expériences de Traitement Automatique du Langage et nous allons notamment utiliser cette représentation des mots pour vérifier une hypothèse de sémantique distributionnelle.

3 Expériences de TAL avec les LSTM

Dans cette section, nous nous intéresserons à l'utilisation d'embeddings de mot créés avec des LSTM pour tenter de vérifier les hypothèses formulées par Bert Capelle, Pascal Denis et Mikaela Keller dans leur article intitulé *Facing the facts of fake : a distributional semantics and corpus annotation approch*. Au travers de plusieurs expériences nous confronterons leurs hypothèses à des embeddings obtenus sur un corpus. Enfin, nous analyseront des résultats de ces expériences et nous tirerons des conclusions sur les hypothèses linguistiques avancées.

3.1 Contexte : le caractère privatif de "fake"

Dans leur article de 2018, Bert Capelle, Pascal Denis et Mikaela Keller s'intéressent aux utilisations du mot "*fake*". [14] En linguistique, *fake* est souvent pris comme exemple

d'une catégorie particulière d'adjectifs, les adjectifs privatifs. Dans le cas de la plupart des adjectifs, les entités désignées par le couple *(Adj+Nom)* sont des instances de l'ensemble des signifiés de *Nom* (*e.g.* Une voiture bleue est toujours une voiture). Par opposition à cela, on appelle "privatifs" les adjectifs pour lesquels la paire *(Adj+Nom)* n'est plus une instance de *Nom*. Ainsi un "faux enlèvement" n'est pas un enlèvement. La première hypothèse que les auteurs font sur les adjectifs privatifs est que ceux-ci modifient davantage les noms que les adjectifs normaux et les éloignent plus de la sémantique initiale du nom. D'autre part, les auteurs notent qu'il suffit d'étudier un ensemble suffisamment large de combinaisons (*fake* + noun) pour se rendre compte que *fake* n'a pas toujours cet usage privatif. En effet, une *"fake video"* est toujours une véritable vidéo (enregistrement ou ensemble d'images animées sur un support électronique), seulement on lui retire sa propriété d'être d'origine authentique (montage, etc.). D'autres associations, comme *"fake news"* sont plus ambiguës car on peut d'une part considérer que ce n'est pas une *news* ou bien considérer que c'en est toujours une, en fonction de l'importance qu'on accorde au critère de véracité dans la définition d'une *news* (est-ce une condition nécessaire ?). Les auteurs distinguent donc plusieurs usages de *fake* qu'ils regroupent en 3 catégories (privatif, non-privatif et ambivalent) et l'hypothèse qu'ils avancent est que le caractère privatif de *fake* dépend surtout des propriétés sémantiques intrinsèques au nom auquel il est associé. Enfin, les chercheurs veulent également vérifier si la différence distributionnelle entre deux usages différents de *fake*, c'est-à-dire si par exemple *fake gun* (usage privatif) est plus éloigné de *gun* que *fake article* (usage non-privatif) ne l'est de *article*.

Par la suite, les auteurs vont réaliser plusieurs expériences de TAL pour essayer de vérifier leurs hypothèses et vont pour cela utiliser des *word embeddings*, ceux ci étant calculés grâce au modèle utilisé dans le projet Mangoes [1], développé par l'INRIA. Ils travaillent sur un corpus de phrases issues de l'entièreté de la version anglophone du site Wikipedia telle qu'elle était en 2013. Pour vérifier leur première hypothèse, ils ont donc calculé la distance sémantique qu'il y avait entre une paire (Adjectif + Nom) et le Nom seul, en distinguant les cas où l'adjectif était ou n'était pas privatif. Ils ont pour cela eu recours au calcul de la similarité cosinus pour déterminer la proximité entre deux vecteurs représentatifs des paires. Les résultats de cette première expérience ne montre pas de différence notable entre la distribution des combinaisons (Adj+Nom) autour de

1. https ://gitlab.inria.fr/magnet/mangoes/wikis/resources

(Nom) selon le caractère privatif de l'adjectif. Les auteurs ne purent donc pas confirmer leur hypothèse de différentiation des paires avec des adjectifs privatifs de celles avec des adjectifs non-privatifs.

Dans une seconde expérience, Capelle et al. vérifient si le caractère privatif ou non de *fake* est due à la distribution sémantique du nom auquel il est associé. Pour cela, ils réalisent une projection des embeddings dans un espace en deux dimensions et regroupent ainsi les noms selon leurs proximités et liens sémantiques. En associant à chaque nom l'interprétation qu'on peut faire en terme de caractère privatif de *fake* lorsqu'il est associé au nom, on remarque que les différents usages de *fake* sont regroupés dans la représentation, indiquant que ce sont bien les caractéristiques sémantiques communes des groupes de noms qui déterminent si l'association avec *fake* se fera de manière privative ou non.

Enfin dans une dernière expérience, ils s'intéressent à la distance entre (*fake*+nom) et le nom, selon l'usage privatif ou non de l'adjectif. Ils utilisent encore une fois la similarité cosinus et la comparaison de ces similarités selon la catégorie d'usage de *fake* parvient à noter une différence selon les catégories : les embeddings sont légèrement plus concentrés autour de la moyenne pour les usages de non-privatifs de *fake* que pour les usages privatifs. Cette différence se fait cependant sur un nombre trop réduit de données pour savoir si cela est significatif ou non.

Par la suite, en nous inspirant de leur travail et de leurs hypothèses, nous allons mener quelques expérimentations pour vérifier de telles hypothèses mais avec un embedding différent, celui que nous avons présenter précédemment, ELMo.

3.2 Expérimentations

Pour travailler de manière analogue à Bert Capelle, Pascal Denis et Mikaela Keller, nous allons étudier le même corpus source : English Wikipedia 2013. Aussi nous sélectionnons un ensemble de 9 noms pour lesquels nous allons étudier l'association avec *fake*. Cet ensemble de noms couvre 5 des 7 catégories précises d'usage énoncées par les auteurs

et couvre bien les 3 catégories principales :

— *beard, blood, company, death* et *gun* pour l'usage privatif

— *ID* et *passport* pour l'usage ambivalent

— *article* et *interview* pour l'usage non-privatif

A l'instar des auteurs, nous ne regarderons que les 3 catégories principales et non la classification en 7 groupes de propriétés sémantiques. Pour chacun des noms, nous avons sélectionné dans le corpus un certains nombres de phrases qui contenait le nom, précédé *fake* d'une part et seul d'autre part. Pour chacun des noms sans *fake*, nous avons collecté 10'000 phrases qui le contenait, à l'exception de *beard* (3500 phrases), *ID* (8690) et *passport* (8370) pour lesquels nous n'avons pas trouvé assez d'occurences. Les occurences de *fake*+Nom étant plus rare, nous avons en moyenne collecté pour chaque association un nombre de phrases aux environs de 72. Après avoir nettoyé les données des résidus du formatage spécifique à la plateforme Wikipedia, nous avons calculé les représentations ELMo. Il faut noter cependant qu'une même phrase peut contenir plusieurs occurrences de *noun* ou de *(fake+noun)* donc le nombre final d'embeddings obtenus pour chaque nom peut être sensiblement plus élevé que le nombre de phrases collectées.

Pour le calcul des embeddings, nous avons utilisé le package *AllenNLP* distribué pour Python. Il contient notamment la classe ElmoEmbedder qui permet de calculer les embeddings selon la configuration choisie de ELMo, parmi plusieurs disponibles. Nous avons choisi de prendre la configuration identique à celle présentée dans Peters et al. [11], à ceci près que cette nouvelle version a été pré-entraînée sur un plus grand nombre de données dont l'origine n'était pas seulement le corpus Penn TreeBank mais également Wikipedia. En plus d'avoir obtenu de meilleurs résultats dans les tests effectués par ses auteurs, cette configuration se rapproche davantage des données sur lesquelles nous voulons travailler, elles aussi extraites de Wikipedia.

Notre approche diffère de Capelle et al. sur un point notable : alors que ces derniers ont considéré l'embedding du bigramme (*fake*+nom), nous avons calculé les embeddings de *fake* et du nom séparément. Ce choix est motivé par l'idée que le caractère privatif ou non de *fake* serait porté par *fake* lui même et que ce dernier modifie ensuite le nom. Ainsi il est intéressant d'étudier les deux séparément. On ne considère alors plus (*fake*+nom) comme une construction idiomatique indépendante mais comme une simple combinaison de deux mots.

Ainsi la première moitié de nos expériences s'intéresse aux embeddings du *nom* et la seconde moitié aux embeddings de *fake*. La première expérience cherche à montrer si la présence de *fake* devant le nom a une influence sur la dispersion du point moyen. En d'autres termes, on cherche à voir vérifier que la présence de *fake* a un effet de sélection sémantique sur le nom et qu'il restreint ses usages à des cas plus limités. Pour cela on calcule pour chaque embedding du nom N la similarité cosinus entre lui et le point moyen des embeddings de N, d'un coté dans les cas où il est précédé de *fake* et d'un autre coté dans les cas où il est seul. Puis on compare, grâce à un test statistique de Student d'égalité des moyennes, la différence des moyennes entre les similarités avec et les similarités sans *fake*.

Dans une seconde expérience, on veut savoir si le caractère privatif de *fake* a un impact sur la similarité pour un nom N entre les points moyens des embeddings du nom seul et des embeddings de N lorsqu'il est précédé de *fake*. En reprenant l'hypothèse linguistique de Capelle et al., on s'attend à ce que la similarité entre la représentation moyenne de N et celle de *fake*+N soit plus grande dans le cas d'un usage non-privatif de *fake* que dans un usage privatif et que la similarité dans l'usage ambivalent soit intermédiaire.

Dans une troisième expérience et pour conclure l'étude des embeddings du nom, on ne compare pas cette fois les embeddings moyens mais la différence de dispersion des embeddings de N entre le cas où *fake* est présent et le cas où il est absent et cela pour les différentes catégories de privativité. La dispersion est évaluée grâce à la distribution des similarités cosinus au sein d'une série entre un embedding et le point moyen de la série : plus les similarités sont élevées, moins la dispersion est forte. Après avoir montré pour chaque catégorie que la présence de *fake* fait varier la dispersion autour du point moyen, on calcule pour chaque catégorie le rapport entre la dispersion moyenne dans le cas où l'adjectif est présent avec la dispersion en son absence. On cherche à montrer une distinction des catégories d'usage de *fake* selon ce rapport des dispersions moyennes et observer si la privativité de *fake* augmente (ou diminue) ce coefficient multiplicateur.

Pour la suite de nos expériences, on s'intéresse aux embeddings du *fake* lui même et non pas ceux du nom comme précédemment. Les expériences 4 et 5 visent à comparer les positions relatives des nuages de points des embeddings de *fake* correspondants aux différents usages de l'adjectif. Pour la quatrième expérience, on représente les nuages de points comme inclus dans des boules ouvertes centrées sur le barycentre des embeddings

de la catégorie et de rayon la dispersion moyenne des embeddings de cette catégorie autour de ce point moyen. Puis on cherche si ces boules ouvertes possèdent des intersections. Si ce n'était pas le cas alors les embeddings de *fake* seraient significativement distincts selon la catégorie.

L'expérience 5 est une comparaison des 1024 coordonnées de deux séries d'embeddings, chacune correspondant à un usage différent de *fake*. Pour chacune des 1024 dimensions des vecteurs, on compare les deux distributions avec un test de Student d'égalité des moyennes pour trouver les dimensions dans lesquelles on pourrait observer des différences significatives de moyenne entre les deux séries. On s'attend à ce que l'usage privatif et l'usage non-privatif se distingue dans davantage de dimensions entre eux qu'avec l'usage ambivalent.

L'expérience 6 revient sur les méthodologies analogues à la première moitié de nos expériences puisqu'elle consiste en une comparaison de dispersion des embeddings de *fake* selon son caractère privatif. On compare ensuite les catégories 2 à 2 pour voir entre lesquelles on peut observer une différence notable de dispersion. De la même façon que dans l'expérience 3, on s'attend à pouvoir construire une relation d'ordre entre les dispersions en fonction du degré de privativité de *fake*.

3.3 Résultats et conclusion des expériences

Nous allons maintenant discuter des différents résultats obtenus.

La première expérience a permis de confirmer ce que Capelle et al. avaient déjà observé : la présence de *fake* diminue bien la dispersions des embeddings autour du point moyen. Ainsi on peut en conclure que la présence de *fake* pose une contrainte sémantique sur le nom et qu'il le restreint à certains sens. Ce résultat, déjà obtenu avec un embedding différent du notre, nous assure quand à la comparabilité avec les résultats que nous obtenons avec ELMo.

L'expérience 2 en revanche n'est pas concluante. Avec des similarités moyennes – très proches – de 0.908 (privatif), 0.913 (ambivalent) et 0.894 (non-privatif), il nous est impossible d'établir une relation d'ordre selon la degré de privativité de *fake* pour la similarité entre l'embedding moyen nom avec ou sans l'adjectif devant lui. On en comprend alors qu'on peut rejeter l'hypothèse selon laquelle *fake* modifierait davantage le nom lorsqu'il est privatif que lorsqu'il ne l'est pas.

La troisième expérience n'obtient pas non plus le résultat attendu. Avec des coefficients de variations de dispersion moyenne de 1.125 (privatif), 1.140 (ambivalent) et 1.091 (non-privatif), on ne peut établir de relation d'ordre selon la privativité de *fake*. On en conclut donc que le degré de privativité de *fake* n'a pas d'influence sur la variation décrite en résultat de l'expérience 1. On ne peut pas affirmer que la privativité de *fake* augmente (ou diminue) la contrainte sémantique sur le nom.

Les expériences 4 ne parviendra pas à montrer une dissociation des nuages de points. En réalité on observe que les nuages de points sont même en grande partie confondus les uns dans les autres. Aucune des distances entre les barycentres n'est suffisantes pour les dissocier. Cependant, cette analyse se portait sur une dispersion moyenne pour chaque catégorie obtenu sur l'ensemble des 1024 dimensions des embeddings. Si on ne peut pas les distinguer globalement, on peut peut être les distinguer précisément en analysant les dimensions une à une comme dans l'expérience 5.

Celle-ci permet justement de faire cette distinction. Elle nous donne pour résultat que chacune des catégories peut se distinguer significativement des autres : ainsi l'usage privatif se distingue de l'usage ambivalent dans 629 dimensions sur les 1024 et de l'usage non-privatif dans 669 dimensions. L'usage ambivalent se distingue de l'usage non-privatif dans 624 dimensions. Comme nous l'avions pensé, il y a donc davantage de différence entre l'usage privatif et l'usage non-privatif qu'entre eux et l'usage ambivalent. Pour pousser la recherche, nous avons comparé les dimensions de distinctions de chacune des paires de catégories. Il en ressort que 239 dimensions de l'embeddings avec ELMo permettent de distinguer l'ensemble de nos 3 échantillons. Cependant ces résultats peuvent être brouillés par différents facteurs comme la petite taille de nos échantillons. Il faut donc le prendre avec précaution.

Enfin, notre dernière expérience permet de montrer que la dispersion des embeddings de *fake* dans le cas où il est privatif est significativement plus élevée non-privatif. Même si la différence qui existe entre la catégorie d'usage ambivalent et les autres n'est pas significative (ce qui est cohérent puisque l'usage ambivalent est intermédiaire aux deux catégories), on observe une relation d'ordre entre les trois. Le caractère privatif de *fake* diminue donc la similarité des embeddings à l'embedding moyen de l'adjectif et avec elle, la variété des contextes sémantiques d'utilisation.

La conclusion de ces expériences est un constat mitigé : s'il nous a été impossible de

montrer des différences d'usage privatif de *fake* par l'étude des embeddings des noms, nous avons néanmoins réussi à montrer des différences significatives entre les représentations de l'adjectif selon son caractère privatif. Ainsi, si on ne peut dire que *fake* modifie différemment le nom selon son usage, on peut à l'inverse montrer une modification de *fake* par le nom auquel il est associé, que ce soit en terme de position dans certaines dimensions qu'en terme de dispersion.

4 Conclusion générale

Pour conclure ce travail, nous rappellerons que les LSTM sont un type de réseau de neurones récurrents qui a montré son efficacité pour résoudre des problèmes liés au travail sur les séquences. En particulier, ils trouvent une application en Traitement Automatique de Langue dans divers aspects. Nous avons exploré l'utilisation des LSTM pour la construction d'un *word embedding* et utiliser celui-ci pour vérifier un ensemble d'hypothèse de sémantique distributionnelle portant sur les adjectifs dit "privatifs" et en particulier *fake*. En plus de vérifier certains des résultats obtenus par Capelle et al., nous avons pu constaté que le degré de privativité de *fake* impactait la dispersion et la position de l'ensemble des embeddings de l'adjectif constitués grâce au modèle biLSTM de ELMo. Plusieurs interrogations subsistent sur nos résultats et la taille réduite de notre jeu de données pour les embeddings de *fake* est un frein concernant la significativité de certains de nos résultats. Il pourrait aussi être intéressant de travailler avec un corpus différent pour voir si nos résultats se répètent ou si les variations observées sont propres au "dialecte Wikipedia", qui diffère de l'usage commun par son formalisme et l'absence (en théorie) d'implication du locuteur. Aussi, il serait judicieux de réaliser ce protocole expérimental avec d'autres LM que celui proposé par ELMo (notamment les plus performants comme BERT) et voir s'il est possible d'obtenir de meilleurs résultats.

5 Références

[1] Warren S. McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biology*, 52 :99–115, 1990.

[2] Sepp Hochreiter. *Untersuchungen zu dynamischen neuronalen Netzen*. PhD thesis, Technical University Munich, Institute of Computer Science, 1991.

[3] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks / a publication of the IEEE Neural Networks Council*, 5 :157–66, 02 1994.

[4] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9 :1735–80, 12 1997.

[5] Alex Graves. Generating sequences with recurrent neural networks. *arXiv : 1308.0850 [Preprint]*, 2013.

[6] Felix Gers, Jürgen Schmidhuber, and Fred Cummins. Learning to forget : Continual prediction with lstm. *Neural computation*, 12 :2451–71, 10 2000.

[7] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. *30th International Conference on Machine Learning, ICML 2013*, 11 2012.

[8] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Łukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes, and Jeffrey Dean. Google's neural machine translation system : Bridging the gap between human and machine translation. *arXix : 1609.08144 [e-print]*, 2016.

[9] Ilya Sutskever, Oriol Vinyals, and Quoc Le. Sequence to sequence learning with neural networks. *Advances in Neural Information Processing Systems*, 4, 09 2014.

[10] Y. Bengio, Réjean Ducharme, and Pascal Vincent. A neural probabilistic language model. *Journal of Machine Learning Research*, 3 :932–938, 01 2000.

[11] Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations. *arXiv : 1802.05365 [e-print]*, 2018.

[12] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. Distributed representations of words and phrases and their compositionality. *arXiv : 1310.4546 [e-print]*, 2013.

[13] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert : Pre-training of deep bidirectional transformers for language understanding. *arXiv : 1810.04805 [e-print]*, 2018.

[14] Bert Cappelle, Pascal Denis, and Mikaela Keller. Facing the facts of fake : a distributional semantics and corpus annotation approach. *Yearbook of the German Cognitive Linguistics Association*, 6(9-42), November 2018.

Annexes

L'ensemble des scripts et *notebooks* ayant servi pour la section 3 peuvent être retrouvés et téléchargés ici : `https://nextcloud.univ-lille.fr/index.php/s/cw68jGcspBD4Swz`