Actes de la 6e conférence conjointe Journées d'Études sur la Parole (JEP, 31e édition),
Traitement Automatique des Langues Naturelles (TALN, 27e édition),
Rencontre des Étudiants Chercheurs en Informatique pour le Traitement Automatique des Langues (RÉCITAL, 22e édition)
Nancy, France, 08-19 juin 2020

Analyse de la régulation de la longueur dans un système neuronal de compression de phrase : une étude du modèle LenInit

François Buet
Université Paris-Saclay, CNRS, LIMSI,
Campus universitaire bât 508, Rue John von Neumann, F - 91405 Orsay cedex
buet@limsi.fr

RÉSUMÉ

La simplification de phrase vise à réduire la complexité d'une phrase tout en retenant son sens initial et sa grammaticalité. En pratique, il est souvent attendu que la phrase produite soit plus courte que la phrase d'origine, et les modèles qui intègrent un contrôle explicite de la longueur de sortie revêtent un intérêt particulier. Dans la continuité de la littérature dédiée à la compréhension du comportement des systèmes neuronaux, nous examinons dans cet article les mécanismes de régulation de longueur d'un encodeur-décodeur RNN appliqué à la compression de phrase, en étudiant spécifiquement le cas du modèle LenInit. Notre analyse met en évidence la coexistence de deux influences distinctes au cours du décodage : celle du contrôle explicite de la longueur, et celle du modèle de langue du décodeur.

ABSTRACT

Investigating Length Regulation in a Sentence Compression Neural System : a Study on the LenInit Model

Sentence simplification aims to reduce the complexity of a sentence, while retaining its original meaning and fluency. In practice, the target sentence is expected to be shorter than the source, and methods that incorporate an explicit control over output length are particularly relevant. In the wake of litterature which seeks to understand the internal behavior of neural systems, we investigate in this article the length regulation mechanisms for a RNN encoder-decoder applied to sentence compression, specifically studying the case of the LenInit model. Our analysis highlights the coexistence of two distinct influences during the decoding, from the explicit control and from the decoder language model.

MOTS-CLÉS : compression de phrase, seq2seq, longueur, contrôle explicite, probing.

KEYWORDS: sentence compression, seq2seq, length, explicit control, probing.

1 Introduction

La *simplification de phrase* peut être définie comme la tâche qui consiste à réduire la complexité d'une phrase, tout en préservant sa grammaticalité et son sens initial. La notion de complexité peut ici faire référence à plusieurs caractéristiques de la phrase (par exemple le vocabulaire, la syntaxe, les connaissances extérieures pré-supposées), et doit être considérée relativement à l'objectif de l'application. La littérature sur la simplification a développé de nombreuses approches, qui se

Volume 3 : Rencontre des Étudiants Chercheurs en Informatique pour le TAL, pages 57–70.
HAL : hal-02786187.

distinguent notamment par le type d'opérations autorisées pour la transformation de la phrase : la simplification *extractive* repose sur la suppression de mot (Knight & Marcu, 2002) ; d'autres méthodes utilisent la *paraphrase*, en accentuant les simplifications lexicales (Horn *et al.*, 2014) ou syntaxiques (Cohn & Lapata, 2008; Specia, 2010; Rush *et al.*, 2015; Takase & Okazaki, 2019).

Souvent, il est attendu que la phrase simplifiée soit plus courte que l'entrée - la tâche devient alors la *compression de phrase*. Dans bien des cas, il est intéressant que le système soit capable de contrôler la réduction de la longueur. En effet, si le but final est d'améliorer la lisibilité d'un texte pour un groupe d'utilisateurs, un contrôle lâche de la longueur est un moyen simple d'adapter le niveau de lecture. Dans le cadre du sous-titrage automatique, un contrôle plus strict est nécessaire pour respecter les contraintes de temps de lecture et de largeur du moniteur (Aziz *et al.*, 2012).

Kikuchi *et al.* (2016) ont proposé *LenInit* [1], une approche pour la compression à base d'encodeur-décodeur, qui contrôle la longueur de la phrase engendrée en introduisant dans l'état initial du décodeur un vecteur qui encode la valeur visée. Afin d'avoir une meilleure compréhension des mécanismes possibles pour la compression de phrase, et dans la continuité des travaux de *sondage* (*probing*) des modèles neuronaux (Shi *et al.*, 2016; Adi *et al.*, 2016; Conneau *et al.*, 2018), nous avons décidé d'analyser la méthode LenInit, en tentant d'estimer ses limites, et en essayant de comprendre les changements qu'elle implique par rapport au fonctionnement interne d'un RNN encodeur-décodeur classique.

Pour cette étude, nous avons créé un corpus artificiel de compression de phrase, et l'avons utilisé pour entraîner une ré-implémentation au niveau caractère de LenInit. Puis nous avons mené trois groupes d'expériences. Premièrement, nous avons effectué des mesures sur la précision du contrôle de longueur par LenInit, et sur la qualité des phrases produites. Deuxièmement, nous avons entraîné un classificateur pour prédire la longueur future (i.e. le nombre de caractères à engendrer avant *fin-de-phrase*) à partir d'un état caché du décodeur, de manière à suivre l'évolution de la représentation de la longueur au cours du décodage. Troisièmement, nous avons tracé l'évolution de la probabilité associée par le modèle aux caractères associés à la fin de phrase. Notre analyse montre que LenInit exerce un contrôle probabiliste sur la longueur, et suggère la coexistence de deux influences distinctes au cours du décodage : celle de l'objectif explicite, et celle du modèle de langue du décodeur.

2 Contexte

Nous décrivons ici le modèle et les données que nous avons utilisés pour nos expériences de compression de phrase au niveau des caractères.

2.1 Réimplementation de LenInit

Le cadre des systèmes *encodeur-décodeur* avec *réseaux de neurones récurrents* (*RNN*) présente un processus en deux phases pour la *transduction* de phrases. D'abord, le côté encodeur reçoit une *phrase d'entrée* x (de longueur l_x) et produit une séquence *d'états cachés* $(h_1, \ldots, h_{l_x})$. Puis, le côté décodeur reçoit les états cachés de l'encodeur et engendre une *phrase de sortie* $\hat{y}$ (de longueur $l_{\hat{y}}$). À chaque étape j de cette seconde phase, le décodeur calcule un *vecteur de contexte* c_j comme une

1. Code disponible à l'adresse https://github.com/kiyukuta/lencon.

combinaison pondérée et normalisée des états cachés de l'encodeur (ce qui est désigné par *mécanisme d'attention*) et l'utilise pour la mise à jour de son propre état caché courant s_j :

$$s_j = f(s_{j-1}, \hat{y}_{j-1}, c_j), \tag{1}$$

où $\hat{y}_{j-1}$ est le mot engendré à l'étape précédente, et f est une fonction non-linéaire. s_j est à son tour utilisé pour calculer la distribution de probabilité associée au prochain mot :

$$p_{decoder}(\hat{y}_j|\hat{y}_{[1;j-1]}, x) = \text{softmax}(g(s_j, \hat{y}_{j-1}, c_j)), \tag{2}$$

où g est une fonction non-linéaire.

Développée par Kikuchi *et al.* (2016), la méthode *LenInit* est fondée sur ce cadre. Nous l'avons réimplémentée, en l'adaptant pour la transduction au niveau des caractères.

LenInit encode la longueur de sortie voulue en multipliant sa valeur scalaire l avec un paramètre appris, le vecteur de longueur V. Cet encodage continu est ensuite introduit au sein du premier état caché du décodeur s_0, comme suit :

$$s_0 = \tanh\left(W_{init}\left[\frac{\sum_{i=1}^{l_x} h_i}{l_x}; L_{param}\right]\right), \quad L_{param} = l \times V, \tag{3}$$

où W_{init} est un paramètre appris. Pendant la période d'entraînement, l est égal à la longueur de la séquence cible de référence, l_y. Pendant la période de test, l est fixé par l'utilisateur : dans nos expériences, $l = r \times l_x$, avec r le *taux de compression* visé.

Nous avons aussi testé des variantes de LenInit, désignées comme LenInit2 et LenInit3, qui se distinguent par la façon de définir L_{param} dans l'équation (3) :

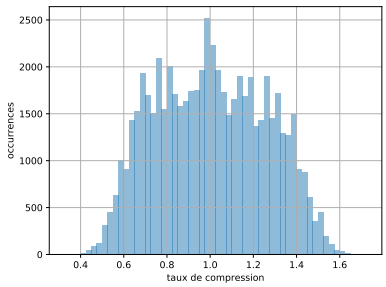
$$\text{LenInit2 :} \quad L_{param} = [l \times V; L_{plong}(l)], \quad \text{LenInit3 :} \quad L_{param} = L_{plong}(l), \tag{4}$$

où $L_{plong}(l)$ est le *plongement* associé à l par une table apprise. L'encodage par la norme de LenInit a en théorie l'avantage de pouvoir marcher pour n'importe quelle longueur, alors qu'un plongement classique (qui apprend indépendamment un vecteur pour chaque valeur) est limité par la fréquence dans les données d'entraînement de la longueur considérée. Avec ces variantes nous avons voulu vérifier le bénéfice lié à un encodage continu, et voir s'il vient au coût d'une perte sur la précision du contrôle de longueur.

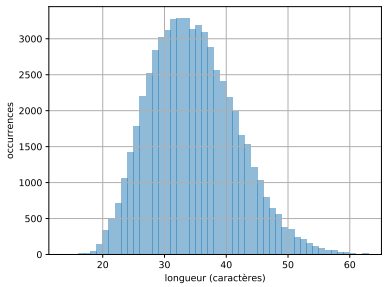
2.2 Un corpus artificiel pour la compression de phrase

Nous avons choisi de mener nos essais sur une tâche plus simple et plus contrôlée que la compression de phrase classique. Pour cela, nous avons créé des données artificielles à l'aide d'un système élémentaire de transduction, qui compresse ou décompresse une phrase source de façon extractive, respectivement en supprimant ou en répétant une partie des caractères. Afin de rendre l'apprentissage d'une telle transformation moins triviale, la décision de supprimer (resp. répéter) un caractère source x_i suit une probabilité p_{sup} (resp. p_{rep}) qui dépend de son « entropie » (autrement dit, qui dépend de sa propension à pouvoir être prédit étant donné le contexte antérieur) :

$$\begin{aligned} p_{sup}(x_i|x_{[i-n+1;i-1]};\theta) &= \max(1, \alpha \times p_\theta(x_i|x_{[i-n+1;i-1]})), \\ p_{rep}(x_i|x_{[i-n+1;i-1]};\theta) &= \max(1, \beta \times (1 - p_\theta(x_i|x_{[i-n+1;i-1]}))), \end{aligned} \tag{5}$$

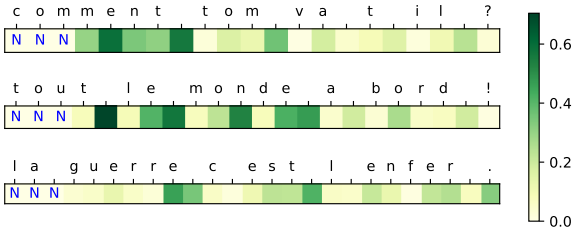


(a) Taux de compression



(b) Longueur source

FIGURE 1: Histogrammes du taux de compression et de la longueur source dans l'ensemble d'entraînement du corpus artificiel.



(a) phrases initiales (sources)

$r = 0,8$	`commttomva t il ?`
$r = 1,2$	`toutt le mmondde a borrd !`
$r = 0,6$	`la guerrc est l enfer`

(b) phrases modifiées (cibles)

FIGURE 2: Exemples de paires dans le corpus. Les phrases sources sont accompagnées d'une carte thermique indiquant pour chaque caractère (excepté les 3 premiers) la probabilité attribuée par un modèle de langue 4-gramme. Les phrases cibles sont compressées ou décompressées, selon r.

où θ est un *modèle de langue* n-gramme au niveau caractère, $p_\theta(x_i|x_{[i-n+1;i-1]})$ est la probabilité d'après p_θ de x_i sachant le contexte des $n-1$ caractères précédents, et α et β sont des facteurs d'échelle (permettant de contrôler la valeur moyenne). Ainsi, une compression devrait supprimer les caractères les plus prévisibles, et une décompression devrait répéter les moins prévisibles.

Notre ensemble de données est constitué de 75 569 phrases de 5 à 10 mots en français, normalisées [2], provenant de Tatoeba [3]. Les phrases cibles sont un mélange de phrases compressées et décompressées. Nous avons défini les valeurs des coefficients α et β de manière à ce que le taux de compression l_y/l_x soit uniformément distribué dans $[0,5; 1,5]$, quand échantillonné sur le corpus entier (Figure 1a) :

$$\alpha = \frac{1-r}{\tilde{p}_\theta}, \quad \beta = \frac{r-1}{1-\tilde{p}_\theta}, \tag{6}$$

où r est une variable échantillonnée uniformément dans $[0,5; 1,5]$ pour chaque phrase (si $r > 1$ une décompression est opérée, sinon une compression), et $\tilde{p}_\theta$ est la probabilité moyenne attribuée par le modèle n-gramme θ aux caractères de référence sur ses données d'apprentissage. θ avait été appris au préalable sur le côté source de l'ensemble d'entraînement.

La figure 1b montre la répartition des longueurs sources de l'ensemble d'entraînement, et la figure 2 présente des exemples de phrases compressées ou décompressées de l'ensemble de développement.

2. Nous avons retiré les diacritiques, majuscules, apostrophes, tirets et virgules.
3. Tatoeba est une base de données de traduction multilingue, constituée de contributions volontaires. `https://tatoeba.org`, diffusé sous licence CC-BY 2.0 FR.

3 Expériences

3.1 Évaluation de LenInit

Précision du contrôle de longueur

Nous nous sommes attachés dans notre premier groupe d'expériences à mesurer la précision avec laquelle LenInit contrôle la longueur de la phrase engendrée. Pour cela nous avons choisi de calculer l'*erreur absolue moyenne* (EAM) et la *racine de l'erreur quadratique moyenne* (REQM) des taux de compression obtenus par rapport aux taux de compression visés, selon les formules suivantes :

$$\text{EAM} = \frac{1}{n}\sum_{i=1}^{n}|\hat{r}_i - r_i|, \quad \text{REQM} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(\hat{r}_i - r_i)^2}, \tag{7}$$

où $n = 7457$ est le nombre d'instances dans l'ensemble de développement, et $\hat{r}_i$ et r_i sont respectivement le taux de compression obtenu et le taux de compression visé pour la i-ème phrase transduite.

L'*erreur absolue* (EA) $|\hat{r} - r|$ peut aussi être vue comme la différence entre la longueur produite et la longueur visée $|l_{\hat{y}} - r \times l_x|$ rapportée à la longueur source l_x. Pour compléter nos métriques, nous avons évalué la proportion d'instances pour lesquelles l'erreur absolue est inférieure à 10%, 5%, ou bien est nulle [4] (Section 5.1).

Validité des phrases engendrées

Quoique davantage intéressés par l'aptitude de LenInit à contrôler la longueur, nous avons mis en place une évaluation pour mesurer la validité (ou grammaticalité en un sens) des phrases engendrées par rapport aux données d'entraînement. La procédure suivie pour créer une phrase dans le côté cible du corpus peut être interprétée comme un *transducteur fini pondéré* : chaque lettre d'entrée est conservée ou supprimée/doublée (selon qu'une compression ou une décompression est réalisée) en suivant des probabilités (Équation (5)) conditionnées sur le contexte des $n-1$ caractères précédents, contexte qui peut être associé à un état.

Notons T_r^{comp} les transducteurs pour la compression ($r \in [0,5;1]$), et T_r^{decomp} les transducteurs pour la décompression ($r \in [1;1,5]$). Grâce à eux, nous sommes capables de vérifier si une phrase produite par un modèle encodeur-décodeur est « correcte » par rapport à la procédure suivie pour créer le côté cible de notre corpus artificiel pour la compression de phrase. Une phrase $\hat{y} = \text{LenInit}_{r=0,6}(x)$, engendrée par LenInit à partir d'une phrase source x avec un taux de compression visé $r = 0,6$, est correcte si elle appartient à $\text{Im}(T_{r=0,6}^{comp} \circ x)$, l'ensemble des phrases cibles qui peuvent être obtenue depuis x par l'automate de compression correspondant (comme à chaque étape du décodage LenInit peut a priori engendrer n'importe quel caractère de l'alphabet, il est tout à fait possible que $\hat{y}$ ne soit pas dans le langage rationnel image du transducteur).

Nous pouvons aussi calculer la probabilité associée à une paire (x, y) formée d'une phrase source et d'une phrase cible correcte : $T_r^{comp}(x, y)$ (resp. $T_r^{decomp}(x, y)$), qui correspond à la probabilité

4. Les cas où $|l_{\hat{y}} - \lfloor r \times l_x \rfloor| = 0$.

x	`la guerre c est l enfer .`
$\hat{y} = \text{LenInit}_{r=0,6}(x)$	`la gerre c est lenfre`
$\hat{y}'$	`la gerre c est lenfe`

TABLE 1: Exemple de phrase source x, phrase prédite (produite par LenInit) $\hat{y}$, et plus proche phrase parmi celles que le transducteur de compression pourrait engendrer à partir de la source $\hat{y}'$.

cumulée de tous les chemins dans T_r^{comp} (resp. T_r^{decomp}) qui transduisent x en y. Dans notre évaluation, nous avons utilisé la probabilité logarithmique négative divisée par la longueur de la phrase source, définissant les scores :

$$S_r^{comp}(x,y) = \frac{-\log T_r^{comp}(x,y)}{l_x}, \quad S_r^{decomp}(x,y) = \frac{-\log T_r^{decomp}(x,y)}{l_x}. \tag{8}$$

Pour les cas dans lesquels une phrase prédite $\hat{y} = \text{LenInit}_r(x)$ n'est pas correcte (au sens donné ci-dessus), nous recherchons la phrase $\hat{y}'$ la plus proche (en terme de distance d'édition) dans $\text{Im}(T_r^{comp} \circ x)$ (resp. $\text{Im}(T_r^{decomp} \circ x)$), et calculons $S_r^{comp}(x, \hat{y}')$ (resp. $S_r^{decomp}(x, \hat{y}')$). Le tableau 1 donne un exemple d'un tel triplet $(x, \hat{y}, \hat{y}')$.

3.2 Prédiction de longueur à partir des états cachés

Notre deuxième groupe d'expériences s'attache à comprendre comment la longueur est représentée dans les états cachés du décodeur, et comment la contrainte impliquée par LenInit agit pendant le décodage. Shi *et al.* (2016); Adi *et al.* (2016) ont déjà montré qu'une information de longueur se trouve dans ce genre de représentations dans les cas de l'auto-encodage et de la traduction. Néanmoins nos essais se placent dans un cadre de compression de phrase avec un taux visé variable ; dans ce cas, la longueur cible ne peut être déterminée par une relation constante à partir de la longueur source, et le décodeur doit intégrer des données extérieures.

En utilisant le modèle LenInit (Section 2.1) entraîné sur notre corpus de compression (Section 2.2), nous avons conçu une tâche de classification qui prédit – étant donné un état caché échantillonné à une certaine étape j du décodage – la longueur de la séquence restant à produire. Les classes de sortie correspondent à des valeurs de longueur, entre 0 et 149 (ce qui est supérieur à la plus grande longueur enregistrée dans le corpus).

Nous avons créé un ensemble de données pour cette tâche, en échantillonnant aléatoirement une fraction (1%) de tous les états cachés produits pendant le décodage (effectué pour divers objectifs de taux de compression) de phrases issues de l'ensemble d'entraînement du corpus de compression [5]. À chacun de ces états cachés a été associé le nombre de caractères de sortie engendrés après qu'il a été échantillonné (c'est-à-dire entre son étape j et la fin du décodage de sa phrase). Ainsi, nous avons obtenu des classes représentées selon leurs proportions naturelles.

5. LenInit a été utilisé pour transduire les mêmes phrases sur lesquelles il avait été entraîné.

3.3 Évolution de la probabilité de génération du caractère de fin de phrase

Notre troisième groupe d'expériences suit l'évolution des probabilités respectivement associées au symbole *fin-de-phrase* et à certaines marques de ponctuation (à savoir « . », « ! » et « ? ») au cours du décodage. Des essais semblables ont été menés par Shi *et al.* (2016), sans trouver de progression régulière. Nous souhaitons vérifier si le cadre de la compression de phrase amène un changement à ce niveau.

Nous utilisons la même configuration que précédemment pour le modèle encodeur-décodeur (LenInit), que nous exécutons sur la partie développement du corpus de compression (Section 2.2). Les probabilités sont extraites des distributions sur le vocabulaire [6] créées à chaque étape du décodage.

4 Implémentation

Modèle de langue n-gramme au niveau caractère

Pour pouvoir produire le corpus expérimental de compression (Section 2.2), nous avons implémenté un modèle de langue 4-gramme au niveau caractère sous la forme d'un *perceptron multicouche* (Bengio *et al.*, 2003) avec une seule couche cachée (de dimension 128). Il a été entraîné pendant 4 époques sur 60 418 phrases en français de Tatoeba, en utilisant *Adam* (Kingma & Ba, 2015) avec son paramétrage standard : $\alpha = 0,0005$, $\beta_1 = 0,9$, $\beta_2 = 0,999$, $eps = 10^{-8}$.

LenInit et variantes

Nous avons implémenté le modèle LenInit (Section 2.1) comme un bi-*GRU* (Cho *et al.*, 2014) (dimension de plongement = 20, dimension cachée = 300, dimension de L_{param} = 300), entraîné pendant 1 époque sur 60 418 phrases en français de Tatoeba, en utilisant Adam (paramétrage standard).

Les variantes LenInit2 et LenInit3 ont les mêmes caractéristiques (en particulier, le nombre de paramètres dans L_{param} est le même).

Évaluation des phrases engendrées

Les transducteurs finis pondérés utilisés pour évaluer la validité des phrases engendrées ont été implémentés à l'aide de la librairie *Pynini* (Gorman, 2016).

Classification de la longueur future

Le classificateur pour la longueur future est un perceptron multicouche avec une seule couche cachée (de dimension 300, égale à la dimension d'entrée) activée par la fonction *ReLU*. Il a été entraîné pendant 5 époques sur 17 112 vecteurs échantillonnés, en utilisant Adam (paramétrage standard).

6. Pour notre modèle, le vocabulaire est l'ensemble des caractères utilisés

Les vecteurs des états cachés de l'ensemble de données pour la classification ont été produits en décodant des phrases de l'ensemble d'entraînement du corpus de compression : pour ces transductions nous avons utilisé LenInit, et pour chacune d'elles nous avons pris un taux de compression objectif uniformément échantillonné dans $[0,5; 1,5]$. Cette répartition aléatoire a été choisie afin de limiter l'introduction de biais dans la tâche.

5 Résultats

5.1 Compression/décompression avec LenInit

Avant de réaliser nos expériences autour des mécanismes de détermination de la longueur, nous avons testé LenInit sur le corpus artificiel de compression de phrase. Le tableau 2 présente quelques exemples de phrases issues de l'ensemble de développement, transduites avec différents taux de compression visés. Afin de vérifier la capacité du modèle à contrôler la longueur, nous avons transduit l'ensemble de développement selon plusieurs modalités :

1. en échantillonnant le taux visé r uniformément dans $[0,5; 1,5]$, ce qui permet de vérifier que la distribution des taux de compression obtenus (Figure 3a) est conforme à celle présente dans l'ensemble d'entraînement (Figure 1) (avec néanmoins une densité plus importante autour de 1, qui semble témoigner d'une tendance à reproduire la phrase source) ;
2. en fixant r à des valeurs présentes dans les données d'apprentissage ($0,6$, $1,0$ et $1,4$), ce qui permet d'observer des distributions de taux de compression effectivement centrées autour d'une valeur (Figure 3b) (notons toutefois que dans le cas $r = 1,4$, la gaussienne obtenue est davantage décalée par rapport au taux visé, probablement à cause de la difficulté pour l'encodeur-décodeur à manipuler les longues séquences) ;
3. en fixant r à des valeurs hors domaine (en dessous – $0,0$, $0,2$, $0,4$, et au dessus – $1,6$, $1,8$, $2,0$), ce qui montre (dans ce contexte au moins) l'incapacité de LenInit à généraliser son action pour des taux de compressions non-vus dans les données d'entraînement (Figures 3c, 3d).

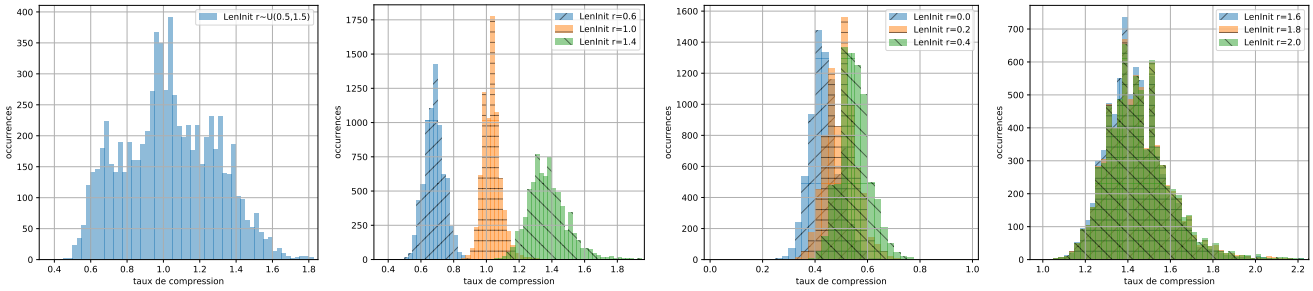
Ce dernier point n'est pas intuitif, dans la mesure où LenInit encode la valeur absolue de la longueur visée, et non le taux de compression (Équation (3)). Une valeur de r en dehors de $[0,5; 1,5]$ peut, selon la longueur de la phrase source, correspondre à une longueur cible (objectif) rencontrée lors de l'apprentissage. Cela laisse penser que le modèle de langue contenu dans le décodeur pourrait s'opposer et prévaloir face aux mécanismes de contrôle de longueur.

Le tableau 3 donne des mesures portant d'une part sur la précision du contrôle de longueur, et d'autre part sur la validité des phrases engendrées. Pour ce qui concerne la précision, LenInit2 et LenInit3 opèrent mieux que LenInit : les valeurs EAM et REQM montrent que les écarts aux valeurs de longueur visées sont moins grands dans l'ensemble, et les proportions d'instances proches des objectifs (EA=0, EA<5%, EA<10%) sont plus importantes. Il apparaît donc qu'un encodage par plongement permet plus de précision dans le contrôle de longueur qu'un encodage par la norme (qui en outre, comme précisé ci-dessus, ne garantit pas la fonctionnalité pour des valeurs hors domaine). Toutefois, l'encodage mixte de LenInit2 se montre plus efficace que le plongement pur de LenInit3, ce qui suggère que l'encodage continu apporte tout de même un bénéfice pour les valeurs peu fréquentes.

Nous observons également que LenInit réussit mieux à conserver la longueur ($r = 1$) qu'à compresser ou décompresser. Il est intéressant de noter que seulement 8,6% des phrases sont correctement recopiées par $\text{LenInit}_{r=1,0}$, quoique 25,1% des sorties soient de longueur exactes.

Source	`comment tom va t il ?`
LenInit$_{r=0,6}$	`commnt m a il ?`
LenInit$_{r=1,0}$	`comment tom va ti l ?`
LenInit$_{r=1,4}$	`commment tomm va ti l ??`
Source	`tout le monde a bord !`
LenInit$_{r=0,6}$	`tout lmode bor !`
LenInit$_{r=1,0}$	`tout le moodne a boord !!`
LenInit$_{r=1,4}$	`tout le moondde aa boord !!`
Source	`la guerre c est l enfer .`
LenInit$_{r=0,6}$	`la gerre c est lenfre`
LenInit$_{r=1,0}$	`la gurrre r c est l enfre .`
LenInit$_{r=1,4}$	`la guurerre c eesst l enfefre ..`

TABLE 2: Exemples de phrases transduites par LenInit (taux $0,6$, $1,0$ et $1,4$).



(a) $r \sim \mathcal{U}(0,5;1,5)$ (b) $r = 0,6;1,0;1,4$ (c) $r = 0,0;0,2;0,4$ (d) $r = 1,6;1,8;2,0$

FIGURE 3: Histogrammes des taux de compression dans l'ensemble de développement, transduit par LenInit selon différentes modalités. Les distributions se recouvrent pour les essais hors domaine.

Modèle	EAM	REQM	EA=0	EA<5%	EA<10%	corr.	S corr.	S inco.	S glob.
				$r \sim \mathcal{U}(0,5;1,5)$					
LenInit	8,4%	0,108	9,6%	35%	67%	16,4%	0,35	0,44	0,42
LenInit2	6,1%	0,086	18,5%	53%	82%	22,2%	0,39	0,47	0,45
LenInit3	7,7%	0,119	14,7%	45%	74%	19,8%	0,36	0,45	0,43
Oracle (T)	6,1%	0,080	17,0%	51%	80%	100%	0,37	-	0,37
				r fixé					
LenInit$_{r=0,6}$	8,6%	0,102	6,0%	29%	64%	15,5%	0,52	0,56	0,56
LenInit$_{r=1,0}$	4,7%	0,071	25,1%	62%	89%	8,6%	0	-	-
LenInit$_{r=1,4}$	9,9%	0,131	10,3%	31%	59%	6,1%	0,54	0,55	0,55
RNN$_{r=0,6}$	10,9%	0,195	7,30%	29%	58%	16,4%	0,38	0,47	0,45

TABLE 3: Mesures sur la précision du contrôle de longueur et sur la validité des phrases produites (Section 3.1). corr., S corr., S inco., S glob. indiquent respectivement la proportion de phrases correctes, et le score moyen attribué aux phrases correctes, incorrectes, et à l'ensemble des phrases. L'oracle est le transducteur fini pondéré qui a engendré le corpus.

Les scores de validité des phrases sont comparables, entre les variantes de LenInit et l'oracle (le transducteur utilisé pour générer le corpus).

À titre de comparaison nous avons aussi entraîné un encodeur-décodeur RNN classique sur un corpus comparable mais ne contenant que des phrases compressées pour $r = 0,6$. Les scores de validité indiquent que le RNN a appris un modèle de langue adapté, et nous notons une précision un peu moins importante que celle de $\mathrm{LenInit}_{r=0,6}$.

5.2 Prédiction de la longueur future

Nous avons testé notre classificateur sur l'ensemble de développement du corpus de prédiction de la longueur future (Section 3.2), qui contient $n = 2169$ états cachés échantillonnés. La figure 4a montre la distribution de la différence entre la longueur prédite par le modèle et la longueur de référence (récupérée expérimentalement lors de la création de l'ensemble de données). Cette distribution est approximativement centrée autour de 0 et nous avons calculé les valeurs EAM et REQM comme suit :

$$\text{EAM} = \frac{1}{n}\sum_{i=1}^{n}|\hat{l}_i - l_i| = 3,08, \quad \text{REQM} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(\hat{l}_i - l_i)^2} = 4,94, \tag{9}$$

où $\hat{l}_i$ et l_i sont respectivement la longueur prédite et la longueur de référence pour le i-ème échantillon.

Notons que ce résultat confirme l'existence dans un état caché du décodeur de données qui représentent spécifiquement la longueur à venir, et qui ne peuvent dériver directement d'un reliquat d'information relatif à la phrase d'entrée. En effet, puisque le modèle LenInit qui a engendré les états cachés avait reçu des objectifs choisis aléatoirement (uniformément dans $[0,5; 1,5]$), le classificateur ne devrait pas avoir pu établir de corrélation systématique entre la longueur d'entrée et la longueur de sortie.

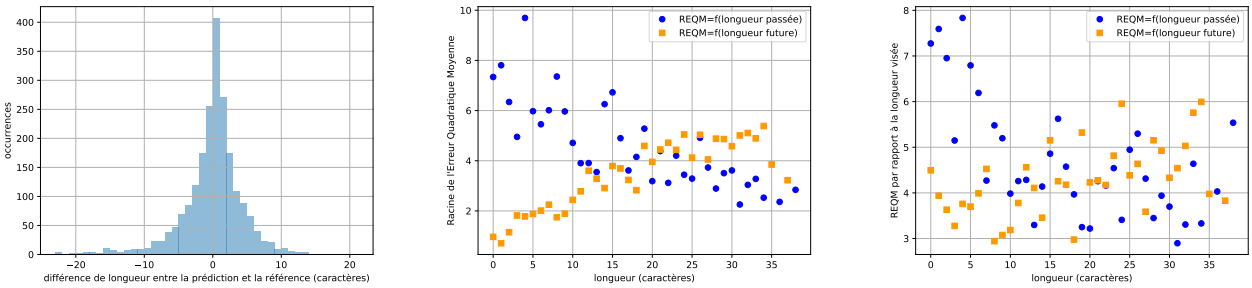
La figure 4b montre l'évolution de la mesure REQM en fonction de la position de l'état caché dans la séquence de sortie. L'indicateur d'erreur est calculé selon :

$$\text{REQM}(k) = \sqrt{\frac{1}{|I_k|}\sum_{i\in I_k}(\hat{l}_i - l_i)^2}, \tag{10}$$

où I_k est l'ensemble d'indices qui décrit les instances de l'ensemble de développement qui ont été échantillonnées à la position k dans leur phrase de sortie ; les deux séries sur la figure se distinguent en mesurant k soit depuis le début de la phrase, soit avant sa fin. La tendance suggère que la précision de la prédiction augmente au fil du décodage.

Similairement, la figure 4c montre l'évolution au cours du décodage d'une mesure de type REQM, cette fois calculée par rapport à la longueur visée : dans l'équation (10), l_i ne correspond plus à la longueur future de référence (celle obtenue en pratique), mais à la longueur future visée (celle qu'il faudrait produire pour atteindre l'objectif de compression donné par r). Les profils sont moins nets dans ce cas, mais nous constatons que l'erreur en fin de phrase est plus importante (ce qui découle logiquement de l'imprécision du contrôle exercé par LenInit), et que l'erreur en début de phrase est étonnamment haute, étant donné que la longueur visée est intégrée dans le premier état caché.

Ces observations peuvent être interprétées comme des signes que l'objectif de longueur plongé dans l'état initial coexiste durant le décodage avec un autre mécanisme de contrôle de la longueur, propre au décodeur, qui est influencé par les caractères progressivement engendrés.

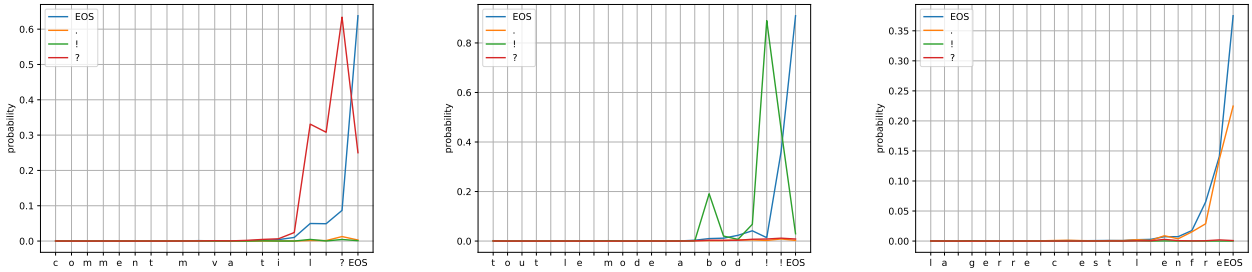


(a) Différence par rapport à la longueur de référence

(b) REQM par rapport à la longueur de référence

(c) REQM par rapport à la longueur visée

FIGURE 4: (a) Histogramme de la différence $(\hat{l}_i - l_i)$, sur l'ensemble de développement. (b), (c) Évolution au cours du décodage de l'erreur de la longueur prédite par rapport à la longueur de référence et par rapport à la longueur visée. Les marqueurs carrés correspondent à l'erreur en fonction du nombre d'étapes avant la génération de *fin-de-phrase*. Les marqueurs ronds correspondent à l'erreur en fonction du nombre d'étapes après *début-de-phrase*.



(a) Source :
`comment tom va t il ?`

(b) Source :
`tout le monde a bord !`

(c) Source :
`la guerre c est l enfer .`

FIGURE 5: Évolution au cours du décodage de la probabilité attribuée par $\text{LenInit}_{r=0,8}$ aux caractères « . », « ! », « ? » et *fin-de-phrase*, pour trois phrases exemples.

5.3 Évolution de la probabilité des caractères de fin de phrase

Après avoir analysé l'information de longueur présente dans les états cachés, nous nous sommes penchés sur ce que le modèle de langue du décodeur en faisait. La figure 5 donne l'évolution de la probabilité attribuée par le modèle LenInit (appliqué avec un taux de compression visé $r = 0,8$) aux caractères « . », « ! », « ? » et *fin-de-phrase*, pendant le décodage de phrases provenant de Tatoeba (présentes dans l'ensemble de développement du corpus de compression). Après examen d'un large éventail de tels graphes, il apparaît que la hausse qui provoque la génération de *fin-de-phrase* est généralement très abrupte, et qu'elle succède la plupart du temps à la génération d'une marque de ponctuation, qui elle même n'a pas eu lieu après une montée régulière ou étagée de probabilité. Ainsi, contrairement aux mesures de la Section 5.2, la probabilité des caractères de fin de phrase semble être un signal final à travers lequel il est difficile de suivre la progression du processus de contrôle de la longueur.

6 Travaux connexes

La simplification de phrase et la compression de phrase sont des domaines bien étudiés du traitement automatique des langues, qui ont bénéficié des avancées majeures apportées par la traduction automatique aux méthodes de transduction. Aux approches pionnières fondées sur les règles (Cohn & Lapata, 2008) ont succédé - dans la suite de *Moses* (Koehn *et al.*, 2007) - des modèles statistiques comme ceux de Specia (2010); Wubben *et al.* (2012). Puis, grâce au cadre des réseaux neuronaux récurrents (Sutskever *et al.*, 2014; Bahdanau *et al.*, 2015), Rush *et al.* (2015) ont implémenté pour la tâche un système encodeur-décodeur avec attention. Enfin plus récemment, des travaux tels que ceux de Zhao *et al.* (2018); Takase & Okazaki (2019) ont adapté l'architecture *Transformer*.

Quoique souvent comparée à la traduction, la simplification de phrase peut également être décrite du point de vue du transfert de style, dans la mesure où elle vise à reformuler la même signification avec un tour moins complexe (et plus concis parfois). La littérature sur le transfert de style propose généralement de travailler avec des représentations dans un espace continu, et d'y séparer le contenu sémantique des dimensions stylistiques. Ainsi Hu *et al.* (2017) utilisent un *auto-encodeur variationnel* avec des représentations *démêlées* pour engendrer des phrases dont les attributs (parmi lesquels la longueur) sont contrôlés. La dissociation entre la représentation style et celle du fond sémantique est réalisée dans ce cas par apprentissage adverse. En comparaison, le modèle LenInit de Kikuchi *et al.* (2016) utilise un paramètre pour encoder la longueur voulue et le concatène à la représentation de la phrase initiale, sans employer de mécanisme pour assurer que celle-ci ne contienne pas d'information de longueur. Cela rejoint sur le principe certaines méthodes qui contrôlent la catégorie de la phrase produite en ajoutant un symbole spécifique à la fin de la phrase d'entrée (pour la formalité (Sennrich *et al.*, 2016), ou le domaine pour (Kobus *et al.*, 2017)).

Nous suivons dans cet article les études de sondage des représentations, qui ont détecté certaines informations de surfaces (longueur entre autres) dans les plongements de phrases (Adi *et al.*, 2016; Conneau *et al.*, 2018). Nos expériences sur la prédiction de longueur future et le suivi de probabilité au cours du décodage sont en partie similaires à celles de Shi *et al.* (2016) (dans le cadre de la compression de phrase toutefois).

7 Conclusion

Nous avons réimplémenté la méthode LenInit, et avons étudié son efficacité et ses mécanismes de régulation de longueur pour une tâche de compression/décompression sur un corpus artificiel. Il semble que deux influences s'opposent au cours du décodage : d'une part l'objectif explicite de longueur, et d'autre part la contrainte de produire une phrase grammaticalement correcte exercée par le modèle de langue du décodeur. Nous avons pu déceler ces influences dans l'information contenue dans les états cachés, mais sans encore comprendre exactement comment elles agissent. Si LenInit exerce bien un contrôle sur la longueur, celui-ci n'est pas étroit mais plutôt probabiliste (d'après des expériences où pourtant la consigne pourrait toujours être précisément respectée), et peut être influencé par le choix d'encodage. À l'avenir, il pourrait être intéressant de tester si la contrôlabilité dépend de certaines caractéristiques de la phrase d'entrée, ou si la distribution des types d'opérations choisis par le modèle change pendant le décodage.

Références

ADI Y., KERMANY E., BELINKOV Y., LAVI O. & GOLDBERG Y. (2016). Fine-grained analysis of sentence embeddings using auxiliary prediction tasks. *CoRR*, **abs/1608.04207**.

AZIZ W., DE SOUSA S. C. M. & SPECIA L. (2012). Cross-lingual sentence compression for subtitles. In *16th Annual Conference of the European Association for Machine Translation*, EAMT, p. 103–110, Trento, Italy.

BAHDANAU D., CHO K. & BENGIO Y. (2015). Neural machine translation by jointly learning to align and translate. In Y. BENGIO & Y. LECUN, Éds., *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*.

BENGIO Y., DUCHARME R., VINCENT P. & JANVIN C. (2003). A neural probabilistic language model. *J. Mach. Learn. Res.*, **3**, 1137–1155.

CHO K., VAN MERRIËNBOER B., GULCEHRE C., BAHDANAU D., BOUGARES F., SCHWENK H. & BENGIO Y. (2014). Learning phrase representations using RNN encoder–decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, p. 1724–1734, Doha, Qatar : Association for Computational Linguistics. DOI : 10.3115/v1/D14-1179.

COHN T. & LAPATA M. (2008). Sentence compression beyond word deletion. In *Proceedings of the 22nd International Conference on Computational Linguistics*, (COLING 2008), p. 137–144, Manchester, UK : Coling 2008 Organizing Committee.

CONNEAU A., KRUSZEWSKI G., LAMPLE G., BARRAULT L. & BARONI M. (2018). What you can cram into a single $& !#* vector : Probing sentence embeddings for linguistic properties. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1 : Long Papers)*, p. 2126–2136 : Association for Computational Linguistics.

GORMAN K. (2016). Pynini : A Python library for weighted finite-state grammar compilation. In *Proceedings of the SIGFSM Workshop on Statistical NLP and Weighted Automata*, p. 75–80, Berlin, Germany : Association for Computational Linguistics. DOI : 10.18653/v1/W16-2409.

HOCHREITER S. & SCHMIDHUBER J. (1997). Long short-term memory. *Neural Computation*, **9**(8), 1735–1780. DOI : 10.1162/neco.1997.9.8.1735.

HORN C., MANDUCA C. & KAUCHAK D. (2014). Learning a lexical simplifier using Wikipedia. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 2 : Short Papers)*, p. 458–463, Baltimore, Maryland : Association for Computational Linguistics. DOI : 10.3115/v1/P14-2075.

HU Z., YANG Z., LIANG X., SALAKHUTDINOV R. & XING E. P. (2017). Toward controlled generation of text. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ICML'17, p. 1587–1596 : JMLR.org.

KIKUCHI Y., NEUBIG G., SASANO R., TAKAMURA H. & OKUMURA M. (2016). Controlling output length in neural encoder-decoders. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, p. 1328–1338 : Association for Computational Linguistics. DOI : 10.18653/v1/D16-1140.

KINGMA D. P. & BA J. (2015). Adam : A method for stochastic optimization. In Y. BENGIO & Y. LECUN, Éds., *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*.

KNIGHT K. & MARCU D. (2002). Summarization beyond sentence extraction : A probabilistic approach to sentence compression. *Artif. Intell.*, **139**(1), 91–107. DOI : 10.1016/S0004-3702(02)00222-9.

KOBUS C., CREGO J. & SENELLART J. (2017). Domain control for neural machine translation. In *Proceedings of the International Conference Recent Advances in Natural Language Processing, RANLP 2017*, p. 372–378, Varna, Bulgaria : INCOMA Ltd. DOI : 10.26615/978-954-452-049-6_049.

KOEHN P., HOANG H., BIRCH A., CALLISON-BURCH C., FEDERICO M., BERTOLDI N., COWAN B., SHEN W., MORAN C., ZENS R., DYER C., BOJAR O., CONSTANTIN A. & HERBST E. (2007). Moses : Open source toolkit for statistical machine translation. In *Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics Companion Volume Proceedings of the Demo and Poster Sessions*, p. 177–180, Prague, Czech Republic : Association for Computational Linguistics.

MALLINSON J., SENNRICH R. & LAPATA M. (2018). Sentence compression for arbitrary languages via multilingual pivoting. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, p. 2453–2464 : Association for Computational Linguistics.

RUSH A. M., CHOPRA S. & WESTON J. (2015). A neural attention model for abstractive sentence summarization. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, p. 379–389 : Association for Computational Linguistics. DOI : 10.18653/v1/D15-1044.

SENNRICH R., HADDOW B. & BIRCH A. (2016). Controlling politeness in neural machine translation via side constraints. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics : Human Language Technologies*, p. 35–40, San Diego, California : Association for Computational Linguistics. DOI : 10.18653/v1/N16-1005.

SHI X., KNIGHT K. & YURET D. (2016). Why neural translations are the right length. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, p. 2278–2282, Austin, Texas : Association for Computational Linguistics. DOI : 10.18653/v1/D16-1248.

SPECIA L. (2010). Translating from complex to simplified sentences. *Lecture Notes in Computer Science*, **6001**, 30–39. DOI : 10.1007/978-3-642-12320-7_5.

SUTSKEVER I., VINYALS O. & LE Q. V. (2014). Sequence to sequence learning with neural networks. In Z. GHAHRAMANI, M. WELLING, C. CORTES, N. D. LAWRENCE & K. Q. WEINBERGER, Éds., *Advances in Neural Information Processing Systems 27*, p. 3104–3112. Curran Associates, Inc.

TAKASE S. & OKAZAKI N. (2019). Positional encoding to control output sequence length. *CoRR*, **abs/1904.07418**.

WUBBEN S., VAN DEN BOSCH A. & KRAHMER E. (2012). Sentence simplification by monolingual machine translation. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics (Volume 1 : Long Papers)*, p. 1015–1024 : Association for Computational Linguistics.

ZHAO S., MENG R., HE D., SAPTONO A. & PARMANTO B. (2018). Integrating transformer and paraphrase rules for sentence simplification. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, p. 3164–3173 : Association for Computational Linguistics.