IFT6285 – Projet 2
De l'ordre dans les mots

Maxime MONRAT
Juan Felipe DURAN
Nathan MIGEON

Université de Montréal

Automne 2021

1 Introduction

L'objectif de ce projet est d'étudier le problème de réordonner un sac de mots. Dans la communauté du TALN, on appelle cela « *bag generation* » ou bien « *shake-and-bake generation* », ou plus récemment *linéarisation de phrase.* C'est un problème qui fait partie d'un domaine de recherche très actif de nos jours, notamment pour les tâches de traduction automatique.

En conséquence, plusieurs méthodes et idées ont été essayées dans la littérature, et notre plan est d'observer et analyser des modèles ainsi que l'impact de leurs méta-paramètres afin d'estimer celui qui fonctionne le mieux. Nous avons accès à 99 tranches du 1B-word corpus. Ces données seront utilisées pour entraîner nos modèles. De plus, nous avons les corpus de développement: News, Hans, et Euro. Chaque corpus contient 1000 phrases désorganisées et 1000 phrases ordonnées.

Nous explorons dans ce travail des solutions basées sur trois modèles de langue: un modèle N-gramme, un modèle neuronal de type séquence-à-séquence dont l'encodeur est un RNN LSTM, un dont l'encodeur est un BiLSTM, et un dernier dont l'encodeur est un CNN.

2 Historique

Jusqu'à récemment, la solution proposée au problème de linéarisation était d'utiliser les propriétés syntaxiques de langue pour retrouver l'ordre le plus probable.

Aujourd'hui les travaux de Schmaltz et al. (2016) ainsi que Vainio et al. (2018) ont démontré que les modèles de langues sont en réalité capables d'obtenir de bons résultats en l'absence de données de syntaxe: si les modèles n-grammes sont efficaces, les modèles neuronaux récurrents de type séquence-à-séquence semblent capables de pousser plus loin les résultats sur ces tâches de classification

3 Description du problème

La tâche de linéarisation d'un sac de mot est en fait une version du problème du commis voyageur : il s'agit de trouver la permutation k la plus probable pour les n mots d'une phrase.

$$argmax_k \prod_n P(x_n | x_{n-1} \dots x_0)$$

Ce problème étant d'ordre $O(k!)$, il s'agit de trouver un moyen de déterminer la permutation *k* la plus probable en tout en réduisant l'ordre de traitement

Une des façons de réduire l'ordre du traitement est d'utiliser un algorithme de recherche en faisceau: à l'aide d'un modèle de langue *M*, on évalue les *k* mots les plus probables parmi l'ensemble $V = [x_0, x_1, \dots, x_{n-1}, x_n]$ des *n* mots du sac de mots donné le mot déterminé précédent, puis on répète cette opération pour chacun des candidats trouvés. Cette méthode permet alors de réduire l'ordre du problème à $O(k^n)$.

L'algorithme proposé par Vainio et al. (2018) vient, pour chaque candidat regarder les *k* candidats les plus probables suivants et n'ajoute aux branches possibles que la suite la plus probables parmi ces k^2 candidats. En répétant ce traitement sur chaque mot, l'ordre est réduit à $O(nk^2)$.

4 Description des approches

4.1 Modèle N-Gramme à minimum de perplexité

Notre première approche utilise des modèles 2-gramme et 3-gramme avec un lissage Kneser-Ney, entraîné avec la librairie KenLM sur les 99 tranches du corpus d'entraînement. Nous nous sommes servis de ce modèle afin d'évaluer la perplexité des phrases candidates.

L'algorithme en faisceau utilisé dans cette approche est celui fourni par Schmatz et al. (2016) dans leurs travaux autour de la linéarisation de sacs de mots sans données syntaxiques.

4.2 Modèles Séquence - à - Séquence

Nous avons exploré dans un second temps les modèles séquence-à-séquence. Ces modèles sont composés de deux parties: un encodeur, dont le but est d'extraire et compresser des caractéristiques des données en entrée, et un décodeur qui génère les données de sortie à partir de ces caractéristiques.

En amont de notre encodeur, nous plongeons les mots dans un espace vectoriel de dimension 50 à 300, en utilisant les plongements de mots pré-entraînés GloVE.

Encodeur LSTM

Le premier encodeur est un LSTM simple qui ne retourne pas de séquence: il ne fait que transmettre ces états internes au décodeur.

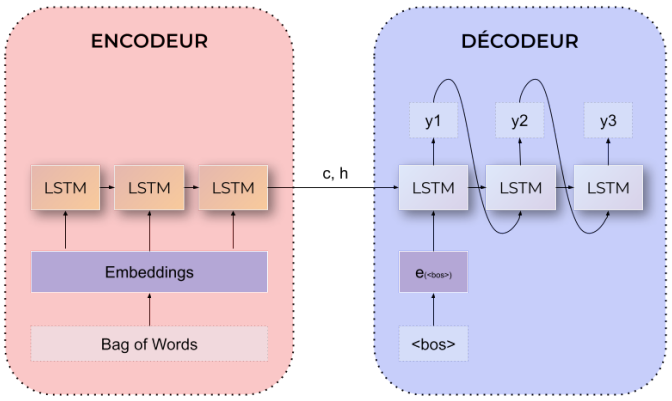

Figure 1 : Architecture du modèle séquentiel avec un encodeur LSTM

Encodeur BiLSTM

Le second encodeur testé est un encodeur BiLSTM. Cet encodeur vient transmettre au décodeur la concaténation de ses quatre états cachés (il s'agit en réalité de deux LSTM évoluant en parallèle et en sens inverse l'un de l'autre)

Encodeur CNN

Cet encodeur utilise un réseau neuronal à convolutions plutôt qu'un modèle récurrent. Ce modèle est inspiré des travaux de Vainio et al. (2018) en reprenant la même architecture:

$$Input \rightarrow Conv1D_{1024}, ReLU \rightarrow Conv1D_{512}, ReLU$$

$$\rightarrow MaxPool1D \rightarrow Conv1D_{512}, ReLU \rightarrow MaxPool1D$$

$$\rightarrow fully\ connected \rightarrow Output$$

Les états des cellules du décodeur sont ensuite initialisés avec les sorties de la dernière couche entièrement connectée.

Décodeur LSTM

Les trois architectures essayées utilisent toutes le même décodeur: il s'agit d'un LSTM simple dont le but est d'évaluer selon le mot en entrée le mot suivant le plus probable. Lors de l'entraînement nous utilisons la technique du "*Teacher Forcing"*, c'est-à-dire que le décodeur utilise à chaque étape le mot de référence qui *devrait* être prédit plutôt que le mot précédemment trouvé.

En plus des données d'entrée et de sorties, les cellules LSTM possèdent un état caché et un état de cellule. Ces états viennent activer les différentes portes (*forget gate, update gate, output gate*), créant des liens entre les échantillons. Ce sont ces états qui sont transmis par l'encodeur.

La sortie de notre décodeur est une activation softmax qui parcourt l'ensemble du vocabulaire. En inférence, nous ne sélectionnons seulement les mots présents dans la phrase d'entrée et qui n'ont pas été trouvés, plutôt que d'effectuer un *argmax* sur l'ensemble du vocabulaire

5 Expérimentations

5.1 Données

Nos modèles n-grammes ont été entraînés sur les 99 tranches du corpus 1BW. À cause d'un manque de mémoire, nos modèles séquentiels n'ont quant à eux pu être entraînés que sur 50 000 phrases de ce corpus, ce qui est très peu.

Plutôt que de créer notre propre dictionnaire nous nous sommes servis du vocabulaire du fichier de plongements GloVE afin de convertir chaque phrase en entrée en séquence d'entiers. Nous avons ajouté au dictionnaire quatre types :

- <*unk*> pour les mots inconnus, <*bos*> et <*eos*> pour les débuts et fin de phrases, dont les plongements vectoriels sont initialisés aléatoirement,
- <*pad*> pour compléter la phrase jusqu'à la taille maximale acceptée (afin que toutes les séquences d'entrée aient la même taille), dont le plongement est initialisé à 0.

5.2 Métriques d'évaluation et méta-paramètres

Lors de la recherche en faisceau nous sélectionnons les meilleures *k* phrases candidates en prenant le logarithme des probabilités, afin de réduire les erreurs d'approximation :

$$argmax_k \sum_n log\Big(P(x_n | x_{n-1} \dots x_0)\Big)$$

Nous évaluons les résultats sur les corpus de test en utilisant le score BLEU tel que défini par Liu et al. (2015).

Nos modèles séquentiels ont été entraînés sur 50 000 phrases du corpus d'entraînement par batches de 10 échantillons pendant 30 epochs, à l'aide d'une carte graphique NVidia RTX3080. Pendant toute la durée de l'entraînement nous avons utilisé un taux d'apprentissage de 0.001 et une optimisation ADAM

6 Résultats

6.1 Modèles N-Grammes

Nous remarquons que le meilleur modèle est le 3-gram avec un beam size de 64, qui nous retourne des scores BLEU aux alentours de 40 pour les trois corpus de test.

En ce qui concerne ses limitations, nous avons considéré et même entraîné un modèle 4-gramme, mais à cause de sa taille et son besoin de mémoire (11Go de RAM), il était impossible pour nous de faire des analyses avec ce modèle

Une limitation importante du modèle n-gramme est qu'il n'est basé que sur des méthodes probabilistes. En conséquence, cela ne tient pas compte du contexte des phrases ou même de si elles font réellement du sens, ce qui est particulièrement visible lorsqu'on n'utilise qu'un seul faisceau (*greedy search*). Par ailleurs, en théorie, un plus grand "beam size" devrait permettre d'avoir de meilleurs résultats, mais le temps d'entraînement devient irréaliste.

6.2 Modèles séquentiels

Bien qu'ils semblent être capables de bons résultats d'après l'article de Vainio et al (2019), nos modèles séquentiels ne sont malheureusement pas viables pour ce travail.

On peut identifier plusieurs causes à nos problèmes. D'abord le manque de phrases d'entraînement. La méthode d'entraînement utilisée nécessite le chargement du fichier total en mémoire. Utiliser une méthode générative permettrait probablement d'obtenir de meilleurs résultats. Ensuite, encore pour des raisons de mémoires, les tenseurs sur lesquels travaillent nos modèles sont en float32. Or les probabilités étant si faibles, et puisqu'on ne considère le logarithme de celles-ci qu'en fin de chaîne, nous perdons ici beaucoup de précision.

Nous n'avons pas non plus pu obtenir de bons résultats pour notre encodeur CNN: le temps d'entraînement et la mémoire étaient trop élevés pour pouvoir le tester (~ 3h30 / epoch).

Notre dernier problème est dû à notre algorithme de recherche. Nous n'avons pu essayer qu'un algorithme "glouton" (*greedy search*), notre recherche en faisceau n'utilisant qu'une méthode récursive simple d'ordre $O\left(k^n\right)$. Réduire cet ordre grâce à la méthode décrite par Vainio et al. (2018) nous permettrait probablement de trouver de meilleures phrases candidates.

	Hans		Euro		News	
Modèle, beam size	BLEU (%)	t (s)	BLEU (%)	t (s)	BLEU (%)	t (s)
Bigramme, 1	12.50	3.81	11.79	3.98	11.91	4.262
Bigramme, 2	27.30	21.35	26.70	22.22	26.85	22.57
Bigramme, 64	28.69	48.36	28.30	54.63	29.13	57.66
Trigramme, 1	14.94	121.99	15.36	114.70	15.22	115.01
Trigramme, 2	37.56	135.57	39.95	133.87	39.90	137.66
Trigramme, 64	39.63	158.57	42.28	165.61	**42.49**	168.63
LSTM-LSTM,1	3.37	-	4.25	-	2.18	-
BiLSTM-LSTM, 1	3.12	-	2.97	-	4.29	-
CNN-LSTM, 1	-	-	-	-	-	-

Table 1 : Score BLEU et temps de traitement lors du test de chacun de nos modèles. Les modèles sont identifiés par leur nom suivi de la taille du faisceau utilisé lors du décodage.

7 Conclusion

Alors que le problème de réorganisation d'un sac de mot semble simple dans un contexte où de plus en plus de solutions aux problèmes de traduction automatique sont apportées, on remarque que les solutions ne sont pas si évidentes à mettre en œuvre.

Dans un premier temps, nous avons utilisé des modèles n-grammes simples, puis nous avons essayé plusieurs variations de modèles neuronaux capables de vectoriser le contexte des phrases et d'extraire de l'information à un niveau plus abstrait.

Malgré nos difficultés à obtenir un score BLEU satisfaisant, nous avons pu observer les mécanismes internes des réseaux neuronaux, et confirmer, à l'aide des papiers ci-dessous, que ces modèles auraient pu répondre au problème de linéarisation non seulement avec un meilleur score BLEU, mais aussi avec beaucoup plus d'efficacité.

On notera tout de même que les modèles n-grammes sont extrêmement plus simples à mettre en œuvre et bien moins gourmands en énergie que nos gros modèles séquentiels. Ce travail vient donc mettre en lumière l'importance de considérer l'application finale d'un projet de traitement automatique des langues et la marge d'erreur acceptable : un modèle n-gramme étant bien moins coûteux qu'un modèle neuronal, est-il réellement nécessaire de toujours approcher les problèmes par une technique d'apprentissage profond?

Bibliographie

Bourdois, 2021, *Le seq2seq et le processus d'attention*

Chollet 2017, *A ten-minute introduction to sequence-to-sequence learning in Keras*

Doshi 2021, *Foundations of NLP Explained Visually: Beam Search, How It Works*

Heafield 2021, *KenLM Language Model Toolkit*

Kingma, Ba 2017, *Adam: A Method for Stochastic Optimization*

Liu et al. 2015, *Transition-Based Syntactic Linearization*

Schmatz et al. 2016 , *Word Odering Without Syntax*

Schmaltz et al. 2017, *n-gram decoder*

Vainio et al. 2019, *Sentence unscrambler: Exploring Deep Learning Models for Word Linearization*

Wong 2019, *What is Teacher Forcing?*