IFT6285 Devoir 1
Séquence de Lemmes Vers Formes de Surface

Zhibin Lu, Thomas Schweizer

March 16, 2018

1 Introduction

Dès le début, nous étions curieux d'utiliser une approche supervisée d'apprentissage par machine. Pour commencer, nous avons entraîné un modèle LSTM pour faire une traduction de séquence à séquence. Le problème principal avec cette technique était que nous utilisions un processus génératif, comme si l'on faisait de la traduction et nous obtenions souvent des phrases qui n'avaient pas la même taille et qui n'avaient plus rien avoir avec la phrase originale. Pour régler ce problème, nous avons essayé un modèle HMM. Ensuite, nous avons essayé des modèles plus simples tel qu'un bi-gramme et une approche hybride car les modèles LSTM et HMM étaient trop lents à utiliser dans le cadre de notre projet.

L'utilisation de ces différentes approches nous ont permis de comprendre leurs forces et leurs faiblesses et l'importance d'utiliser des outils et des modèles adaptés à la situation.

2 Protocole expérimental

2.1 Métriques

Nous avons pensé à utiliser BLEU [1] puis METEOR[2] car il a pour but de fixer les problèmes de BLEU. L'utilisation d'un tel algorithme semblait adapté pour tâche de traduction machine. Cependant, METEOR s'est avéré être un mauvais choix car il utilise la forme 'stem' d'un mot dans son calcul. Or, ce n'est absolument pas souhaitable dans notre cas si notre modèle oublie de traduire des lemmes vers leur forme de surface.
Concernant BLEU, il n'est pas idéal: la comparaison des n-gram est indépendante de leur position. Dans notre cas de traduction, les mots ne changent pas de place. De plus, BLEU vise plutôt à mesurer la qualité d'une traduction et notre tâche est plus 'basique'. Nous voulons seulement savoir avec quel succès nous avons transformé les lemmes vers leur forme de surface correspondante.

NIST ne convient pas non plus car, en plus d'être basé sur BLEU, il donne d'avantage d'importance à certains NGRAM que d'autres. Dans notre cas, tous les n-grammes ont le même poids. D'autres métriques comme ROUGE-L, la distance d'édition et *word error rate* ne sont pas retenues car elles ne mesurent pas exactement ce que l'on veut.

Un autre problème avec un score comme BLEU ou la distance d'édition est qu'il n'y a pas beaucoup d'intuition derrière cette métrique[3]: A quoi correspond un score de 0.34 dans notre cas? Ce n'est pas très parlant...

Ainsi, nous avons choisi d'utiliser le ratio du nombre de lemme correctement traduits sur le nombre total de lemme. Il s'agit de l'exactitude ou *accuracy*. Cette mesure nous donne une bonne idée de la qualité de notre traduction et prends en compte les mots en double.
Suivant le modèle utilisé, il se peut également que nous ayons plus ou moins de formes de surfaces que le nombre de lemme. Évidement, c'est une erreur. Dans ce cas là, nous comptons automatiquement les mots en trop ou manquants comme faux, même dans le cas d'un décalage (e.g. "is" est traduit en "he was") où la suite serait correcte. Cette approche privilégie fortement les modèles à prédire une seule forme de surface par lemme.

2.2 Données

Nous utilisons 3 sets de données d'entraînement: le fichier `train-1183.gz`, les 10 et 100 premiers fichiers de la tranche d'entraînement que nous testons contre 2 sets de tests: le fichier `test-2834.gz` et toute la tranche de test (20 fichiers).

Pour le modèle HMM et bi-gramme, nous utilisons uniquement la combinaison `train-1183.gz` : `test-2834.gz`. Pour le modèle LSTM nous utilisons `train-1154.gz` au lieu de `train-1183.gz`[1].

2.3 Hypothèses

Nous prenons le parti de travailler sur des phrases en minuscules. Ça simplifie le problème à résoudre. Corriger les majuscules dans un texte est un problème à part entière et des solutions existent déjà [4].

2.4 Baseline

Notre baseline consiste à utiliser la séquence de lemmes comme forme de surface et comparer par rapport à la phrase originale. Cette baseline a pour avantage d'être facile à produire et d'être pertinente car elle est déjà assez proche de la phrase originale.

|Phrases|	Fichier(s)	Exactitude [%]
6 434	`test-2834.gz`	83.30
236 404	Tout le corpus de test	82.76
8 624 162	300 fichiers du corpus d'entraînement	82.39

Le calcul de la baseline sur un échantillon des données d'entraînement nous permet de vérifier que le corpus de test est bien représenté dans le corpus d'entraînement au regard de notre métrique.
Il semble également qu'un fichier de test représente bien le corpus de test.

3 Approche neuronale (LSTM)

A l'origine, nous voulions utiliser une approche par réseau de neurones. Pour ce faire, nous avons commencé par suivre l'exemple donné dans l'énoncé du devoir[2]. L'exemple original est pour traduire des phrases anglaises courtes en français. Dans notre cas, nous "traduisons" une phrase lemmatisée vers sa forme de surface.

3.1 Entraînement

Pour nos données, nous transformons chaque phrase en une matrice dont chaque élément e_{ij} =(indice du caractère dans la phrase, indice des caractères candidats) comme dans l'exemple.

Pour entraîner ce modèle nous utilisons Google Colaboratory. Malheureusement, il y a une limite et nous pouvons entraîner uniquement 4800 phrases et utiliser 1201 phrases pour la validation.

Meta-paramètres

- **batch size**: 64
- **epochs**: 100
- **latent dimension**: 256

3.2 Discussion & Résultats

Après 10 heures d'entraînement, nous avons effectué la prédication sur les données de test [3].

[1] Le plus pertinent aurait été d'utiliser le même fichier que HMM mais nous nous en sommes aperçus trop tard...
[2] https://blog.keras.io/a-ten-minute-introduction-to-sequence-to-sequence-learning-in-keras.html
[3] Voir chapitre 2

Comme on peut s'y attendre [5] avec un set de données si petit, nos résultats ne sont pas bons. Nous obtenons une exactitude de 7.74%... La figure 1 contient quelques exemples de traductions.

```
predict:  it has a settlement in the american league with a record of 72 wins and 70 losses .
surface:  it has two locations , including the main facility on west chestnut street in washington , as well as i

predict:  the main tailer was a senate of emberom , as a structures and service and in survic , song , the state
surface:  the main restaurant boasts old wooden booths , a dining counter , as well a large storefront , showing

predict:  the temple mount , all of the band , and the second to be population of the season .
surface:  the trademark menu item is an albert 's frank with chili , mustard and onions , as well as fries with ç

predict:  the former was released in 1929 , and was also a man and evertan and in the carol , stringul to enders
surface:  the food is inexpensive , with the hot beef dinner being the most expensive menu item at $ 5 .

predict:  shortly after the station , the season , the season , the most of recording and the second game to the
surface:  shorty 's main location sells 600-700 hot dogs a day .

predict:  the club is located in the town of again .
surface:  the chili is especially favored by customers .
```

Figure 1: Examples de prédictions faite par le modèle LSTM.

Bien que le réseau manque clairement de données d'entraînement, ces résultats nous permettent d'identifier deux problèmes intrinsèques à cette approche.

1. le LSTM demande un temps d'entraînement par rapport au résultat obtenu; C'est beaucoup d'efforts pour un résultat si en deça de la baseline.
2. un processus génératif n'est pas la bonne façon d'aborder notre cas particulier de traduction: nous savons déjà que la phrase cible comportera le même nombre de mots que la phrase d'entrée. Le problème est en fait de retrouver la bonne forme de surface pour chaque lemme.

4 Approche HMM

Le but est de corriger les deux problèmes observés dans l'approche précédente. En effet, avec un modèle HMM, nous sommes garantis d'avoir un nombre de mots prédits et de lemme égaux pour une même phrase.

Nous entraînons le modèle HMM fourni par le paquet *hmmlearn*[6] avec nos données (mode fermé). *textacy*[7] et *spacy*[8] nous servent à générer les n-grammes et préparer les données d'entraînement (mode ouvert).

4.1 Entraînement

Nous pré-traitons chaque phrase comme une séquence, chaque type de surface (mot de surface) est un état caché, chaque type de lemme (lemme) est un état visible.

Nous utilisons l'une des trois propriétés d'un HMM: En donnant les probabilités à priori d'états cachés, les probabilités de transition entre les états cachés et les probabilité d'émission entre états cachés et états visibles. On fait le décodage directement d'un séquence d'états cachés (phrase de surface) la plus probable en donnant un séquence d'observation (phrase lemmatisée) avec l'algorithme de Viterbi.

En utilisant un-gramme et bi-grammes, on calcule les probabilités à priori des états cachés:

$$p(c_t|BOS) = \frac{p(c_t, BOS)}{p(BOS)} \cong \frac{\#(c_t, BOS)}{\#(BOS)}$$

Où BOS est le marqueur de début de phrase et c_t le t_{eme} type de surface.

Pour les probabilités de transition entre les états cachés:

$$p\left(c_{t-1} \to c_t\right) = p(c_t|c_{t-1}) = \frac{p(c_{t-1}, c_t)}{p(c_{t-1})} \approx \frac{\#(c_{t-1}, c_t)}{\#(c_{t-1})}$$

Pour les probabilité d'émission:

$$p(c_t \to v_t) = p(v_t|c_t) = \frac{p(c_t, v_t)}{p(c_t)} \cong \frac{\#(c_t, v_t)}{\#(c_t)}$$

Où v_t est le type de lemme respectif à c_t.

4.2 Problèmes & solutions

Le principal problème de ce modèle est qu'il ne peut pas prédire certains mots: aucune probabilité d'émission leur est associée. Vous pouvez en voir un exemple sur le tableau ci-dessous:

SRC	`the food be inexpensive , with the hot beef dinner be the most expensive menu item at $ 5 .`
REF	`the food is inexpensive , with the hot beef dinner being the most expensive menu item at $ 5 .`
PRD	`the food is celebratory , with the hot ! ! ! ! ! ! ! ! ! ! ! !`

4.2.1 Lissage

Pour résoudre ce problème, nous avons appliqué un filtre de Laplace modifié[4] afin de garantir que les trois matrices possèdent une probabilité pour chaque mot:

$$M = M + 1.0/n$$
$$M = M/2.0$$

Où M est la matrice de probabilité. Les opérations + et / sont faites sur chaque élément de la matrice.

Le lissage résout ce problème avec succès, comme le montre l'exemple suivant:

SRC	`the food be inexpensive , with the hot beef dinner be the most expensive menu item at $ 5 .`
REF	`the food is inexpensive , with the hot beef dinner being the most expensive menu item at $ 5 .`
PRD	`the food is inexpensive , with the hot beef dinner be the most expensive menu item at $ 5 .`

Nous avons un autre problème majeur: la prédiction prend beaucoup de temps: plus de 9 minutes par phrase! Clairement, il y a un problème dans notre modèle. Nous pensons que c'est du au fait que nous avons trop d'états cachés (30 139), que les phrases sont longues en moyenne(max 120 mots!) et que la complexité de l'algorithme de Viterbi est $O(T * (n)^2)$ (n est le nombre d'état cachés).

4.2.2 Regroupement des mots communs

Pour tenter de contourner ce problème, nous avons essayé de réduire la taille des phrases en regroupant les suites de mots communs entre la phrase lemmatisée et la phrase originale. Ces regroupements sont utilisés pour entraîner le HMM. Voir l'exemple suivant:

La fin de la phrase est omise car la chaîne de mots commençant à "à" est la même. "je" n'est pas absorbé par le début de la phrase car il est est placé juste avant un mot qui change.

SRC	`le mercredi, je vais à la piscine`
REF	`le mercredi, je aller à la piscine`
SRC'	`le je vais à`
REF'	`le je aller à`

Avec cette modification, le nombre d'état cachés tombe à 16'200 et réduit la longueur moyenne des phrases[5]. La prédiction est 12 fois plus rapide: une phrase est prédite en 42 secondes.

[4] A cause de la façon dont spacy gère les n-grammes.
[5] la phrase la plus longue a 59 mots

4.3 Discussion & résultats

Variante	Exactitude [%]
Lissage	62.55
+Regroupement	88.49

Table 1: Résultats pour l'approche HMM.

En plus de réduire le temps de prédiction, le regroupement des mots nous permet de gagner 26% de performances! Cette amélioration s'explique par le fait qu'il y a moins de mots à prédire où l'on risque de faire une erreur et que le modèle enregistre des bi-grammes plus pertinents car on enlève des mots "redondants".
Évidemment, le problème que l'on tente de résoudre avec ce travail pratique est modifié car on simplifie aussi les données de test à prédire. Cette solution pourrait néanmoins être valide si l'on possède un moyen de reconstruire la phrase après la prédiction.

En conclusion, cette approche résout en partie le problème de la lenteur d'entraînement mais expose les limites d'un modèle HMM: la complexité induite par les états cachés et la distribution de la masse des probabilités. Ce n'est pas un modèle "clé en main"; il est nécéssaire de l'adapter et d'essayer plusieurs paramètres différents.

5 Approche bi-gramme

Jusqu'à présent, nos modèles étaient compliqués. Est-ce que l'on peut obtenir de bon résultats avec un modèle plus simple ? Dans ce but, nous avons développé un modèle bi-gramme. En effet, comme on utilise des bi-grammes dans notre modèle de Markov, il serait intéressant de voir les résultats que nous obtiendrions si on supprime la composante HMM et que l'on garde juste un simple modèle bi-gramme pour notre tâche.

Nous utilisons les paquets *textacy*[7] et *spacy*[8] pour générer les n-grammes et préparer les données d'entraînement (mode ouvert).

5.1 Entraînement

Notre modèle est construit de la manière suivante: On associe toutes les paires de lemme $lemm(w_{t-1}, w_t)$ à un mot de surface $surf(w_t)$ selon les phrases d'entraînement. Ensuite, par fréquence relative, on garde la pair associée la plus fréquemment. Par exemple, si on a associé $[(you\ be) \rightarrow are]$ 3 fois et $[(you\ be) \rightarrow is]$ 1 fois, on gardera $[(you\ be) \rightarrow are]$. Si un lemme n'apparaît dans aucun bi-gramme, on retourne le lemme comme mot de surface.

Nos utilisons une chaîne de Markov et Bayes:

$$p\left(s_t|l_{t-1}, l_t\right) = \frac{p(l_{t-1}, l_t|s_t)p(s_t)}{p(l_{t-1}, l_t)} \approx \frac{\#((l_{t-1}, l_t) \rightarrow s_t)}{\#(l_{t-1}, l_t)}$$

$$predict(s|l_{t-1}, l_t) = argmax_i(\#((l_{t-1}, l_t) \rightarrow s_i))$$

5.2 Résultats

Avec ce modèle, nous obtenons en un temps raisonnable (quelques minutes) une exactitude de 89.66%. Nous dépassons enfin la baseline.

Nous avons également essayé d'utiliser un parseur[6] et de faire des règles au dessus du modèle bi-gramme. L'idée est de corriger certaines erreurs de grammaires que fait notre modèle. Nous avons défini quelque règles simples. Voir la table 2.

Nos règles n'ont pas beaucoup d'effet, nous avons seulement un gain d'environ 0.4%.

[6] Le parseur est fourni par un module tierce et a été entraîné en mode ouvert.

si sujet est "nsubj" et "NN", donc changer le verb → singulier
si sujet est "nsubj" et "NNS", donc changer le verb → pluriel

Table 2: Exemple de règles.

Variante	Exactitude [%]
Bi-gramme	89.66
+Règles	90.08

Table 3: Résultats pour l'approche bi-gramme.

5.3 Discussion

L'avantage de cette approche est qu'elle est simple à comprendre, plutôt efficace et donne des résultats sans avoir à configurer le modèle comme pour les LSTM et HMM.

La plus grosse limite de ce modèle est son implémentation. La méthode que l'on a choisie pour charger et entraîner les bi-grammes n'est pas efficace pour plusieurs fichiers de données[7]. En effet, lorsque l'on utilise une librairie tierce, il faut faire attention et être sur que c'est adapté pour résoudre notre problème.

Bien que les quelques règles que nous avons utilisées n'aient pas beaucoup de valeur ajoutée, le concept est intéressant. Pour augmenter les performances, nous pensons qu'une analyse approfondie des erreurs que fait le modèle bi-gramme est nécéssaire afin d'isoler les règles pertinentes. L'inconvénient de faire des règles basées sur les données est que c'est spécifique au langage, voir même au domaine à moins qu'elles soient générées automatiquement.

Nous pensons que nos règles ne sont pas très efficaces car nous les avons basées sur nos intuitions et ce que l'on voulait essayer plutôt que d'effectuer une analyse systématique des erreurs du modèle.

6 Approche paresseuse

Notre motivation pour ce modèle est de proposer une approche qui est efficace et est taillée pour résoudre le problème. Jusqu'à présent, nos explorations de modèles ont démontré que la meilleure façon de traduire une phrase est de produire une forme de surface par lemme. Nous savons également qu'un modèle simple comme un bi-gramme donne de bons résultats et est rapide à entraîner et utiliser comparé à un modèle complexe qui est lent et difficile à comprendre et utiliser.

Si l'on interprète les résultats de la baseline, on s'aperçoit que notre taux d'exactitude indique que près de 80% des lemmes n'ont pas besoin d'être changés[8]! Clairement, nous effectuons trop de travail actuellement dans nos modèles car nous prédisons tous les mots.

Comme il est compliqué de déterminer si un mot doit changer ou pas, notre idée alternative est d'enregistrer les formes de surface pour chaque lemme pendant l'entraînement. Lors de la prédiction, si un lemme a une seule forme du surface, utiliser celle-ci sinon utiliser un modèle de prédiction pour choisir la forme de surface la plus adaptée. Ce modèle de prédiction peut utiliser différentes *features*.

Pour tester, nous avons implémenté notre idée avec le modèle de prédiction le plus simple: un modèle de fréquence relative uni-gramme.

[7] Le temps d'exécution croît très vite après quelques fichiers!
[8] Pour la langue anglaise

6.1 Discussion & résultats

|Phrases d'entraînement|	|Phrases de test|	Exactitude [%]
15961	6434	87.51
15961	236404	86.54
1217707	6434	86.57
1217707	236404	86.50
5251515	6434	86.64
5251515	236404	86.71

Table 4: Résultats pour l'approche paresseuse.

Nos résultats sont légèrement en dessus de la baseline (environ 3%) et nous constatons qu'augmenter la taille des données rapporte des améliorations marginales. Nous pensons que c'est parce que les articles de Wikipédia sont écrits en suivant un guide de style commun.

Un autre avantage de ce modèle est son temps d'entraînement rapide comparé aux autres (quelques secondes pour 15'000 phrases et 20 minutes pour 5+ millions de phrase). Le temps de prédiction est presque instantané car il s'agit d'un accès à une hashmap pour chaque lemme.

7 Récapitulatif

Pour ce projet, l'approche qui donne les meilleures résultats par rapport à son temps d'entraînement est celle que nous avons implémentée sans utiliser de modules supplémentaires, qui a le modèle le moins complexe et qui est la plus flexible.

Approche	Exactitude [%]
Baseline	83.39
Paresseuse	87.51
Bi-gramme	90.08
HMM-R	88.49
HMM	62.55
LSTM	07.74

Table 5: Résultats pour le meilleur modèle de chaque approche, trié par vitesse d'exécution croissante. HMM-R est la variante avec le regroupement activé; nous supposons que nous pouvons reconstruire les phrases.

Malgré les propriétés miraculeuses qui sont prêtées aux réseaux de neurones, il faut une certaine expertise et beaucoup de temps et de données pour qu'ils produisent de bons résultats. Notre modèle HMM était plus rapide que le LSTM mais a quand même nécessité des modifications pour être utilisable. Il nous a permis de valider notre supposition sur la manière la plus efficace de résoudre le problème.

Finalement, il s'est avéré que les modèles les plus simples proposaient le meilleur rapport performance/temps d'exécution pour ce type de tâche. De plus, les résultats fournis par les modèles simples étaient meilleures que ceux des modèles complexes et dépassaient systématiquement la baseline.

Parmi les quatre approches, l'approche paresseuse est la plus intéressante pour de futurs travaux: La version actuelle sert d'exemple et son implémentation permet d'être modifiée facilement pour y intégrer n'importe quel modèle de prédiction.
Malheureusement, nous n'avons pas pu essayer d'y intégrer un autre modèle plus raffiné car c'est la dernière approche que nous avons étudiée. Cependant, nous sommes très intéressés de voir les performances obtenues en y intégrant un modèle de règles, analogique ou bi-gramme.

7.1 Données

Nous aurions pu d'avantage réfléchir en analysant notre domaine de données afin de trouver une approche plutôt que d'essayer directement un LSTM puis un HMM. Par exemple, en regardant la distribution des lemmes et de leurs forme(s) de surface. Un danger de cette approche est de faire un modèle lié à une

langue.
Concernant nos modèles, s'ils sont ré-entraînés sur d'autres langues, nous pensons qu'ils devraient obtenir des résultats similaires. En particulier pour les langues latines.

Globalement, nos deux approches par fréquence fonctionnent bien. Nous pensons que c'est dû, en partie, à la nature du domaine. Les articles de Wikipédia sont tous écrits (idéalement) en suivant le même style. Ainsi, nous aurons tendance à retrouver les mêmes temps, tournures de phrase, et pronoms personnels à travers le domaine de données.
Ce phénomène, nous le retrouvons dans les résultats de l'approche paresseuse: augmenter le nombre de données de test ou de prédiction ne semble pas beaucoup influencer l'exactitude.

Dernier point, nous aurions pu supprimer les marqueurs de fin de phrase (e.g. '.') des données que l'on entraînait car leur traduction est triviale et leur absence ne change pas la nature du problème.

8 Conclusion

En utilisant plusieurs approches différentes avec relativement peu de données, nous avons pu identifier quelles étaient leurs forces et leurs faiblesses et appuyer l'importance d'utiliser le bon modèle selon le contexte. Pour ce projet, une approche simple est plus profitable qu'une approche complexe.

Nos approches nous ont montré qu'il y a avait beaucoup de façons d'aborder ce problème plutôt simple et que chaque approche, possède une myriade de variations qui dépendent des meta-paramètres et des détails d'implémentation.

L'étape suivante, sera d'utiliser notre dernière approche et d'y intégrer un modèle de prédiction avec plusieurs *features*. Nous sommes très intéressés à trouver les limites de cette approche.

References

[1] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: a method for automatic evaluation of machine translation. *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 311–318, jul 2002.

[2] Satanjeev Banerjee and Alon Lavie. Meteor: An automatic metric for mt evaluation with improved correlation with human judgments. *Proceedings of the ACL Workshop on Intrinsic and Extrinsic Evaluation Measures for Machine Translation and/or Summarization*, page 65–72, jun 2005.

[3] I. Dan Melamed, Ryan Green, and Joseph P. Turian. Precision and recall of machine translation. In *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology companion volume of the Proceedings of HLT-NAACL 2003–short papers - NAACL '03*, volume 2, pages 61–63, Morristown, NJ, USA, 2003. Association for Computational Linguistics.

[4] Lucian Vlad Lita, Abe Ittycheriah, Salim Roukos, and Nanda Kambhatla. tRuEcasIng. In *Proceedings of the 41st Annual Meeting on Association for Computational Linguistics - ACL '03*, volume 1, pages 152–159, Morristown, NJ, USA, 2003. Association for Computational Linguistics.

[5] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to Sequence Learning with Neural Networks.

[6] HMM learn - https://github.com/hmmlearn/hmmlearn.

[7] textacy - https://chartbeat-labs.github.io/textacy/.

[8] spacy - https://spacy.io/.