Méthodes neuronales de Monte-Carlo pour la génération automatique de résumés de textes

Mémoire

Mathieu Godbout

Sous la direction de:

Luc Lamontagne, codirecteur de recherche
Audrey Durand, codirectrice de recherche

Résumé

<Texte du résumé en français. Obligatoire.>

Abstract

<Text of English abstract. Optional, but recommended.>

Table des matières

Liste des tableaux

Liste des figures

<Dédicace si désiré>

"<Texte de l'épigraphe>"

<Source ou auteur>

Remerciements

<Texte des remerciements en prose.>

Avant-propos

<Texte de l'avant-propos. Obligatoire dans une thèse ou un mémoire par articles.>

Introduction

Question: À quel point est-ce que l'introduction doit déboucher exactement sur mon mémoire ? J'ai l'impression qu'à la fin de mon intro, le lecteur devrait être en contexte par rapport au titre, i.e. il devrait savoir qu'on va parler de (1) résumés, (2) méthodes de MC et (3) que tout ça va être fait avec des NN.

Ère du big data : on a une quantité immense de données, notamment textuelles.

Succès des dernières décennies nous montrent le potentiel qui peut se cacher derrière une utilisation judicieuse de ces données.

Or, il n'est pas toujours possible d'incorporer toutes les données dans un processus de traitement.

Notamment, dans certains domaines, les humains sont encore requis dans la loop dans des applications de traitement de données textuelles en raison de législations ou d'assurance qualité (i.e. processus de réclamations en assurance).

Comment faire profiter du grand afflux de données dans des systèmes où l'humain (et sa capacité de traitement limitée) sont essentiels ?

Une option qui peut s'avérer facilement attrayante est l'idée de condenser un ou plusieurs documents en un texte concis contenant seulement l'information requise pour procéder à la tâche à exécuter.

On dit alors que l'on fait appel à un système de génération automatique de résumés de textes.

Comment fonctionnent ces systèmes de génération de résumés ?

L'espace des résumés possibles est très vaste ; comment faire pour en trouver un qui est suffisamment bon dans un délai de temps assez raisonnable pour l'utilisation dans un contexte du monde réel ?

Des outils probabilistes, connus sous le nom de méthodes de Monte-Carlo, ont été spécifiquement conçus pour de tels contextes.

Ils se basent sur l'aléatoire pour parcourir efficacement des espaces potentiellement très grands.

En les jumelant à des approches neuronales ayant démontré largement leur efficacité sur des données textuelles, il est naturel de s'attendre à ce que ces outils permettent d'obtenir des systèmes de génération automatique qui produisent non seulement d'excellents résumés, mais qui ne le font pas au détriment d'un temps d'entraînement déraisonnable.

Objectifs

Dans ce mémoire, on explorera d'abord comment différentes méthodes de Monte-Carlo peuvent être utilisées pour estimer des mesures liées à la qualité de résumés de textes dont le calcul est difficile à première vue.

On verra ensuite comment ces approximations obtenues par des approches Monte-Carlo peuvent être utilisées dans un système complet de génération de résumés.

Les différents systèmes explorés seront enfin comparés aux approches représentant l'état de l'art en génération de résumés en fonction de leur efficacité d'entraînement et de prédiction ainsi que selon la qualité des résumés produits.

Structure du mémoire

Ce mémoire sera divisé en trois chapitres pour les trois méthodes de Monte-Carlo explorées : échantillonnage, recherche arborescente et recherche arborescente linéaire.

Pour chacun des chapitres, on posera d'abord une fonction en lien avec la génération de résumé qui est difficile à évaluer.

On montera ensuite comment une méthode Monte-Carlo peut être utilisée pour l'approximer.

On évaluera empiriquement la rapidité de la convergence de la méthode proposée.

On proposera ensuite un modèle neuronal de génération automatique de textes utilisant la méthode MC à l'essai.

Des expériences seront ensuite effectuées pour évaluer la performance du modèle.

Les hyperparamètres de la portion MC seront choisis en fonction des tests empiriques préliminaires.

Chapitre 1

Concepts de base pertinents aux travaux

1.1 Méthodes de Monte-Carlo

Approximation statistique de procédés déterministes

1.1.1 Échantillonnage de Monte-Carlo

On a une fonction qui est faite sous la forme d'une espérance.

$$f = \mathop{\mathbb{E}}_{x \sim \Psi} [x] \,,$$

où Ψ est une processus aléatoire pour lequel on ne possède pas de forme analytique facilement calculable.

Si on a accès à Ψ, on peut y piger une suite d'échantillons $(x_1, ..., x_N)$, que l'on peut ensuite utiliser pour approximer f selon

$$\bar{f} = \frac{1}{N} \sum_{t=1}^{N} x_t.$$

Notons que $\bar{f}$ est un estimateur non-biaisé de f.

Applications variées : cas de calcul de π en tirant au hasard plusieurs fois deux nombres entre -1 et 1 (Ramaley, 1969).

1.1.2 Recherche arborescente de Monte-Carlo

On a une fonction qui affiche une structure arborescente (séquentielle avec direction) et on souhaite avoir son maximum (max de -f si min).

On doit parcourir un nombre suffisant de feuilles de l'arbre de f pour s'assurer de l'optimalité de notre réponse mais on veut aussi s'éviter d'avoir à visiter toutes les feuilles (sinon un minmax ferait le travail).

Idée : explorer les sous-arbres en fonction des feuilles vues précédemment, en priorisant les sous-arbres *payants* et ceux qui ont été moins explorés.

L'exploration est guidée par une borne supérieure sur la performance anticipée d'un sous-arbre donné.

Principale variante considérée : UCT (Coquelin and Munos, 2007).

Variante linéaire : si on dispose de représentations pour les noeuds de l'arbre, on peut utiliser la représentation d'un noeud exploré pour tenter d'estimer les valeurs associées à d'autre noeuds. Principale variante considérée : linUCT (Mandai and Kaneko, 2016).

1.2 Réseaux de neurones

1.2.1 Réseaux pleinement connectés

Présentation haut niveau de leur utilité prouvée empiriquement sur à peu près n'importe quel type de problème d'apprentissage supervisé.

— Préciser architecture : neurone, activation

— Préciser fonction de perte : doit être intimement connectée au savoir préalable (prior) sur la tâche et la cible.

1.2.2 Réseaux récurrents

Cas où les données sont des séquences de tailles variables (i.e. données textuelles).

— RNN

— LSTM : pour quelle raison ?

1.2.3 Plongements de mots

On peut utiliser des réseaux pour apprendre des représentations vectorielles de mots (Mikolov et al., 2013; Pennington et al., 2014).

Ces représentations vectorielles permettent d'exprimer certaines relations linéaire entre les mots ("roi" - "homme" + "femme" = "reine").

Les plongements de mots sont à la base de l'explosion de performance dans les tâches liées aux données textuelles grâce à leur utilisation avec des réseaux de neurones.

Chapitre 2

Génération automatique de résumés de textes

Quand on résume, on veut (1) compresser un ou des textes tout en s'assurant de (2) conserver la majeure partie de l'information contenue.

On dit qu'il s'agit d'un dilemme compression-conservation.

Deux techniques principales actuellement utilisées : extractive et abstractive.

Nous nous intéresserons seulement au cas de la génération de résumés à partir d'un seul document pour lequel nous possédons un seul résumé cible.

Notons que le cas où on génère un résumé à partir de plusieurs documents se ramène naturellement au cas d'un seul document en considérant un nouveau document représentant la concaténation de tous les documents en entrée.

2.1 Formulation extractive

On sélectionne des phrases du document initial.

Formulation simple à résoudre : la nombre de résumés dépend maintenant seulement du nombre de phrases et on s'évite toute les difficultés en termes de syntaxe et de cohérence d'avoir à générer du texte.

Les approches de l'état de l'art sont toutes basées sur une approche encodeur-décodeur avec des réseaux de neurones.

Ces approches produisent en sortie une distribution sur les phrases originales.

Le résumé est ensuite bâti à partir de la distribution prédite par le modèle.

Définition formelle : On dit qu'on a un modèle de génération de résumés (π, ϕ). Ici π prend en entrée un document d et retourne $\pi(d)$, une distribution de probabilités de sélection sur les phrases de d. On a ensuite ϕ, qui est un processus de génération de résumé extractif à partir d'une distribution $\pi(d)$. Deux exemples intuitifs de ϕ sont les processus voraces et stochastiques, où on choisit les n phrases avec la plus grande probabilité ou on en pige n sans remise de $\pi(d)$, respectivement.

Note : le modèle doit fondamentalement contenir ϕ et π car l'encoder-décodeur optimal π^* dépend du processus ϕ employé pour la génération de résumés.

2.1.1 Approches supervisées

La majorité des approches extractives sont supervisées.

Comme le résumé correspondant à un document n'est habituellement pas obtenu de manière extractive (i.e. le résumé n'est pas une combinaison de phrases du document initial), les approches supervisées doivent utilisent des cibles basées sur des heuristiques pour leur entraînement.

Par exemple, (Nallapati et al., 2017) sélectionnent de manière vorace des phrases une à la fois en fonction de leur similarité à un résumé pour un document.

Après avoir sélectionné trois phrases, leur processus est arrêté et la cible pour le document est fixée à un vecteur binaire où les index des trois phrases ont une valeur de 1 et les autres index ont une valeur nulle.

Comme on est supervisé, le réseau entraîné ne peut être au mieux aussi bon que l'heuristique utilisée pour générer les cibles utilisées.

La performance est tout de même très bonne ; (Zhong et al., 2020) est le SOTA actuel en extractive.

2.1.2 Approches par renforcement

(Dong et al., 2018; Luo et al., 2019) sont des approches représentant l'état de l'art en la matière actuellement.

Au lieu d'utiliser des cibles binaires, les approches par renforcement visent à optimiser directement une mesure de la similarité entre deux résumés : le ROUGE (Lin, 2004) (plus de détails plus bas).

(Dong et al., 2018; Luo et al., 2019) emploient le même modèle de réseau de neurones. Comme encodeur, deux LSTMs consécutifs, un travaillant au niveau des mots et l'autre au niveau des phrases. Comme décodeur, une couche pleinement connectée partagée pour toutes les phrases et avec sortie réelle. Un schéma et davantage de détails sur ce modèle neuronal sont

disponibles au chapitre 3. Nous reprendrons une architecture similaire de réseau de neurones pour toutes les expérimentations.

Bémol à insérer : (Paulus et al., 2017) entraînent par renforcement sur la formulation abstractive mais finissent par ne pas utiliser le modèle avec le meilleur score ROUGE.

2.2 Formulation abstractive

On écrit un résumé *à la mitaine*.

Formulation difficile car nécessite de gérer la syntaxe et les fautes d'orthographe en plus de la gestion du dilemme compression-conversation.

Récentes percées en NLG ont beaucoup boosté les performances ici.

(Raffel et al., 2020; Dong et al., 2019; Zhang et al., 2019) sont le SOTA en abstractif.

2.3 Évaluation de la performance

Comment distinguer quantitativement deux résumés candidats s et $\hat{s}$?

Intuition : on se base sur les n-grammes d'un résumé cible. Plus le overlap est grand entre les n-grammes d'un candidat et ceux de la cible, plus le résumé a conservé l'information recherchée.

Un problème : si on se fie juste au rappel sur les n-grammes , le résumé optimal sera de conserver tout le texte original. Pour incorporer la portion compression du dilemme fondamental de la génération de résumé, on peut utiliser une métrique plus appropriée comme le score F1 pour ajouter une pénalité sur les n-grammes présents dans le candidat pas dans la cible.

Les considérations précédentes ont mené à la famille de métriques ROUGE (Lin, 2004). Le score ROUGE correspond à assigner une valeur numérique à un résumé candidat à partir de son F1-Score sur une tâche de classification sur les n-grammes d'un résumé cible. En pratique, on utilise généralement $n = 1{,}2$ et on utilise aussi la plus longue sous-séquence, nommée ROUGE-L. On pose généralement

$$R(s, \hat{s}) := \frac{1}{3} R_1(s, \hat{s}) + R_2(s, \hat{s}) + R_L(s, \hat{s}). \tag{2.1}$$

On définit enfin la similarité entre un résumé candidat s à partir d'un résumé cible $\hat{s}$ comme étant $R(s, \hat{s})$.

Une propriété intéressante du score ROUGE est son invariance à l'ordre des phrases.

TODO: expliquer mérites et défauts de R1, R2 et RL et donc pourquoi leur moyenne est bonne. Range de R est [0,1]

Benchmark classique : CNN/DailyMail (See et al., 2017).

À ajouter :

— N articles (n train/valid/test)

— Moyennes mots, phrases texte/abstract

En fonction des stats, les modèles entraînés sur ce jeu de données produisent pour la plupart des résumés de 3 phrases du document original.

C'est ce jeu de données qu'on utilisera pour tous les tests.

2.3.1 Pré-calcul des scores ROUGE

Les méthodes présentées dans les prochains chapitres utilisent toutes un grand nombre de scores R dans leur procédure d'entraînement.

Or, comme le calcul du ROUGE entre deux résumés est lent. Aussi, il est possible de calculer, pour un document d donné, tous les résumés de 3 phrases possibles.

On a donc, dans un premier temps, précalculé tous les scores ROUGE associés à tous les résumés pour chaque document du jeu de données.

Commentaire: Ce chapitre me semble un peu décousu, j'ai l'impression que plusieurs de mes sections ont des dépendances circulaires les unes envers les autres.

Chapitre 3

Échantillonnage

Idée : si on possède une distribution sur les phrases, on souhaiterait savoir quelle est la performance qui lui est associée (processus de génération stochastique seulement).

Il serait alors possible de bâtir un système qui apprend à prédire de bonnes distributions sur les phrases en maximisant directement la performance reliée à une distribution.

Concrètement, si on souhaite approximer la performance d'une distribution π jumelée à un processus de génération de résumé ϕ, on cherche donc à approximer, pour un document et son résumé cible $(d, \hat{s})$,

$$f = \mathop{\mathbb{E}}_{s \sim \phi(\pi(d))} \left[R(s, \hat{s}) \right], \tag{3.1}$$

Cette fonction ne peut pas être calculée analytiquement dans un temps raisonnable car le nombre de résumés possibles à considérer est trop grand (exponentiel en nombre de phrases).

Or, pour estimer f, on n'a pas **besoin** de vérifier tous les résumés possibles : on peut seulement se limiter à piger à quelques reprises de la distribution de résumés générée par $\phi(\pi(d))$.

On procède donc par un échantillonnage de Monte-Carlo (1.1.1), où on calcule plutôt

$$\bar{f} = \frac{1}{N} \sum_{t=1}^{N} R(s_t, \hat{s}), \tag{3.2}$$

avec les résumés s_t échantillonnés selon $\phi(\pi(d))$.

3.1 Expériences sur l'échantillonnage

On valide la rapidité de la convergence de $\bar{f}$ vers f en les comparant directement sur un sous-ensemble du jeu de données CNN/DailyMail.

Pour les tests, on prend 10 000 documents d'entraînement du jeu de données.

On pige aléatoirement des distributions $\pi(d)$ sur les phrases de chacun des documents. Pour assurer une bonne variété dans les distributions explorées, on choisit des distributions représentant une moyenne pondérée entre une distribution uniforme $U(d)$ et une distribution *3-hot* $G(d)$:

$$\pi(d) = (1 - \tau)U(d) + \tau G(d). \tag{3.3}$$

On explore $\tau \in \{\frac{n}{100} : 0 \leq n \leq 100\}$ en repigeant les 3 index non-nuls de $G(d)$ à chaque fois. On ajoute un bruit gaussien sur chaque $\pi(d)$.

On calcule analytiquement f grâce au fait que l'on possède tous les scores ROUGE de tous les résumés possibles pour les documents du jeu de données. On pige 1000 échantillons de $\phi(\pi(d))$, où ϕ est le processus de pige sans remise de 3 phrases.

On pige enfin 1000 résumés de $\phi(\pi(d))$, nous donnant des estimés $\bar{f}_t$ pour t allant de 1 à 1000.

La mesure de qualité à laquelle on s'intéresse ici est l'erreur d'échantillonnage, que l'on définit naturellement comme la différence absolue entre f et $\bar{f}_t$: $\left|f - \bar{f}_t\right|$.

3.2 Résultats échantillonnage

Commentaire: Pour les tableaux, je veux surtout votre input sur (1) la pertinence de chacun, (2) la manière de les présenter (j'ai les données brutes, je peux facilement réorganiser comment je présente les donnnées) et (3) s'il y a des expériences supplémentaires que vous jugez pertinentes.

Les résultats obtenus sont rapportés dans les figures 3.1, 3.2 et 3.3.

Tout d'abord, sur la figure 3.1, on remarque que le nombre de phrases dans un document n'est pas un facteur déterminant sur la rapidité de convergence de l'échantillonnage de Monte-Carlo. Ce résultat n'est pas surprenant : on peut s'attendre à ce que la difficulté de convergence soit davantage qualifiée par une mesure sur la distribution des résumés.

TODO: Rapporter nombre de phrases selon la taille (figure 3.1).

Une première mesure naturelle sur la distribution des résumés est l'entropie, définie pour une distribution X quelconque comme :

$$H(X) = -\sum_{i=1}^{n} X_i \log X_i.$$

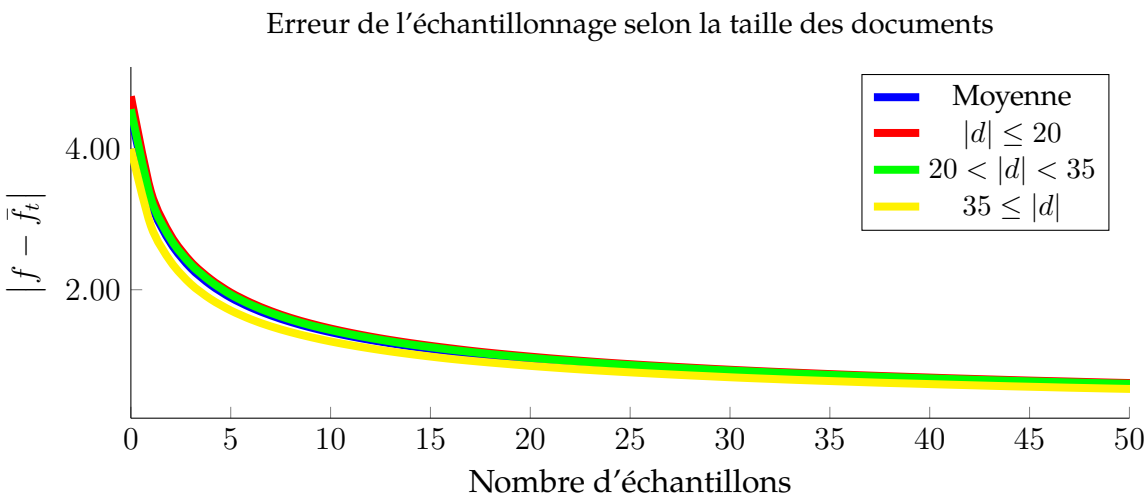


FIGURE 3.1 – Impact de la taille du document en entrée sur l'erreur d'échantillonnage.

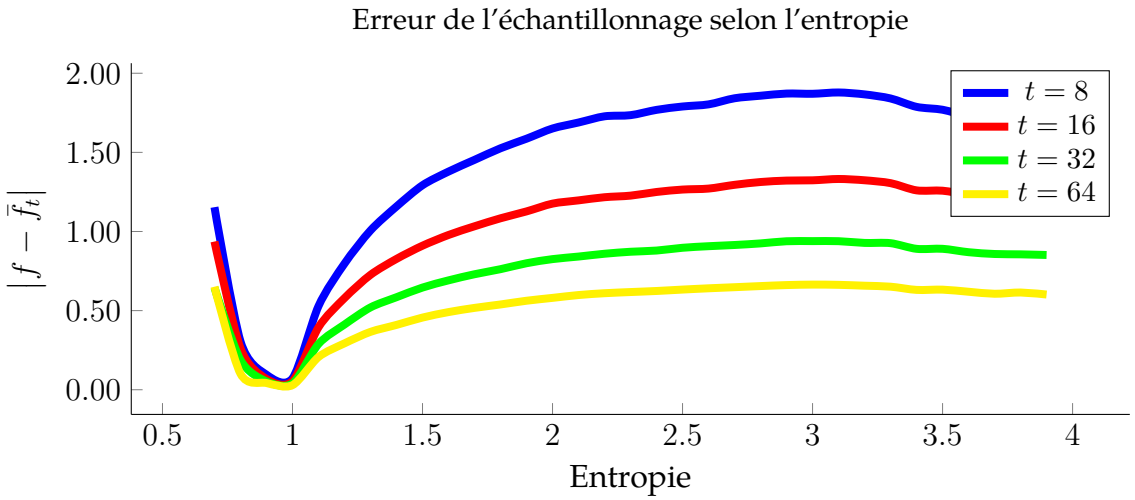


FIGURE 3.2 – Impact de l'entropie de la distribution des résumés sur l'erreur d'échantillonnage.

Important à savoir : entropie est 0 pour un one-hot, plus la distribution est près d'une distribution uniforme, plus elle sera grande.

Analyse de (3.2) : Commentaire: Je viens de réaliser que mon calcul d'entropie est fait sur la distribution des phrases et non pas sur la distribution des résumés. Je vais updater le graphe ASAP. Je vais updater l'analyse à ce moment.

Enfin, c'est aussi pertinent de s'intéresser à l'impact qu'a la valeur maximale de la distribution. La figure (3.3) nous montre une courbe à la tendance logarithmique, où plus la grande probabilité de la distribution augmente, plus rapide est la convergence de notre processus

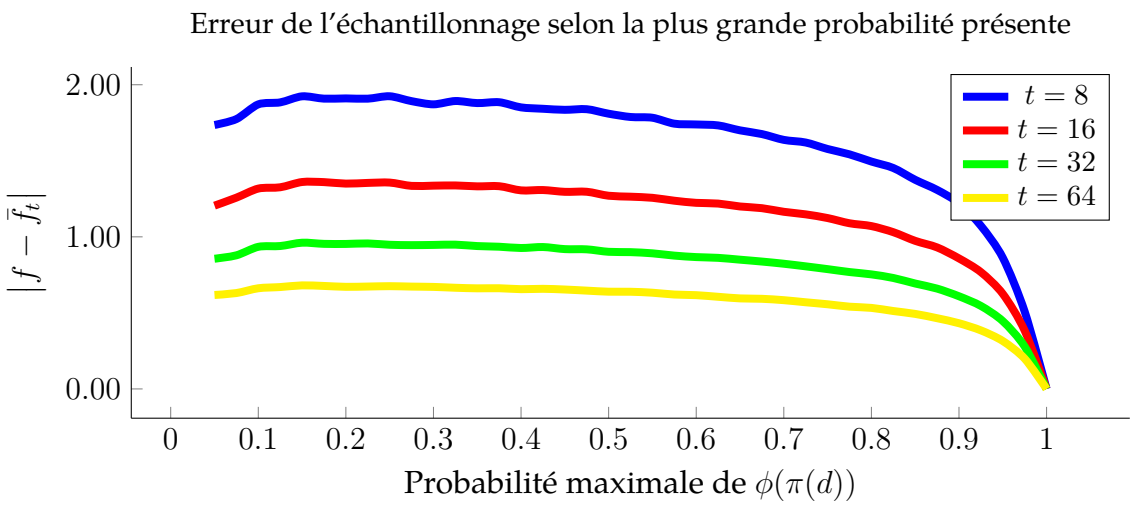


FIGURE 3.3 – Impact de la probabilité maximale de la distribution des résumés sur l'erreur d'échantillonnage.

d'échantillonnage.

Globalement, on remarque aussi que la convergence est rapide. Une différence de moins de 1 R est définitivement suffisante pour faire un apprentissage. Les résultats indiquent de faire 16 échantillonnages semble être un bon compromis performance-temps d'exécution. C'est d'ailleurs cette valeur qui est utilisée dans les articles de (Dong et al., 2018) et (Luo et al., 2019).

3.3 Intégration dans un modèle complet

Pour un modèle avec des paramètres θ, on souhaite maximiser la fonction objectif

$$J(\theta) = f = \mathop{\mathbb{E}}_{s \sim \phi(\pi(d))} [R(s, \hat{s})] . \tag{3.4}$$

Comme on a vu qu'on peut approximer f efficacement avec $\bar{f}_t$, on peut se limiter à utiliser ce dernier. On peut alors utiliser l'ascension de gradient et mettre à jour les poids θ selon la règle

$$\theta = \theta + \nabla_\theta \frac{1}{N} \sum_{t=1}^{N} R(s_t, \hat{s}).$$

C'est exactement ce que fait BanditSUM (Dong et al., 2018).

Question: Ici, je reprends la logique derrière les approches policy gradient (RENFORCE notamment) en RL. J'ai décidé que ce serait pas souhaitable de m'étaler sur quoique ce soit RL dans mon mémoire alors je sais pas à quel point je devrais prendre un moment pour mentionner que j'emprunte cette logique là à des travaux antérieurs.

BanditSUM introduit une baseline pour stabiliser les gradients de J et prend un document à la fois, en posant $N = 16$.

3.3.1 Expériences

Commentaire: Dernière rencontre j'avais parlé de considérer un autre processus de génération de résumés que le sampling qui arrête à 3, mais après réflexion je pense pas que ça va apporter quelque chose tant que ça et que ça va juste alourdir la lecture. Je vais le garder pour une conclusion avec les avenues possibles pour les futurs travaux.

TODO: Détails du processus d'entraînement complet BanditSUM

3.3.2 Résultats

Commentaire: Les expériences pour BanditSUM sont en train de rouler. Je pense surtout rapporter la courbe de performance et le temps d'exécution. Peut-être aussi rapporter comme plafond l'état de l'art ?

3.4 Conclusion

TODO: à faire une fois les résultats de BanditSUM en main

Chapitre 4

Recherche arborescente

4.1 Recherche du résumé optimal

On cherche à calculer

$$f = \arg\max R(s, \hat{(s)}) \tag{4.1}$$

À cause de la taille déraisonnable de l'espace des résumés, on ne peut pas calculer directement.

4.2 Calcul d'un résumé presque-optimal

Arbre de résumé.

Algorithme UCT pour parcours d'arbre.

Expériences sur grand nombre de documents, potentiellement plusieurs formulations envisageables, importance des hyperparamètres, taille des documents.

4.3 Intégration

UCT génère une distribution cible sur les phrases.

La distribution est apprise par MSE et l'inférence est faite de manière vorace.

Chapitre 5

Recherche arborescente linéaire

5.1 Estimation du score de tous les résumés

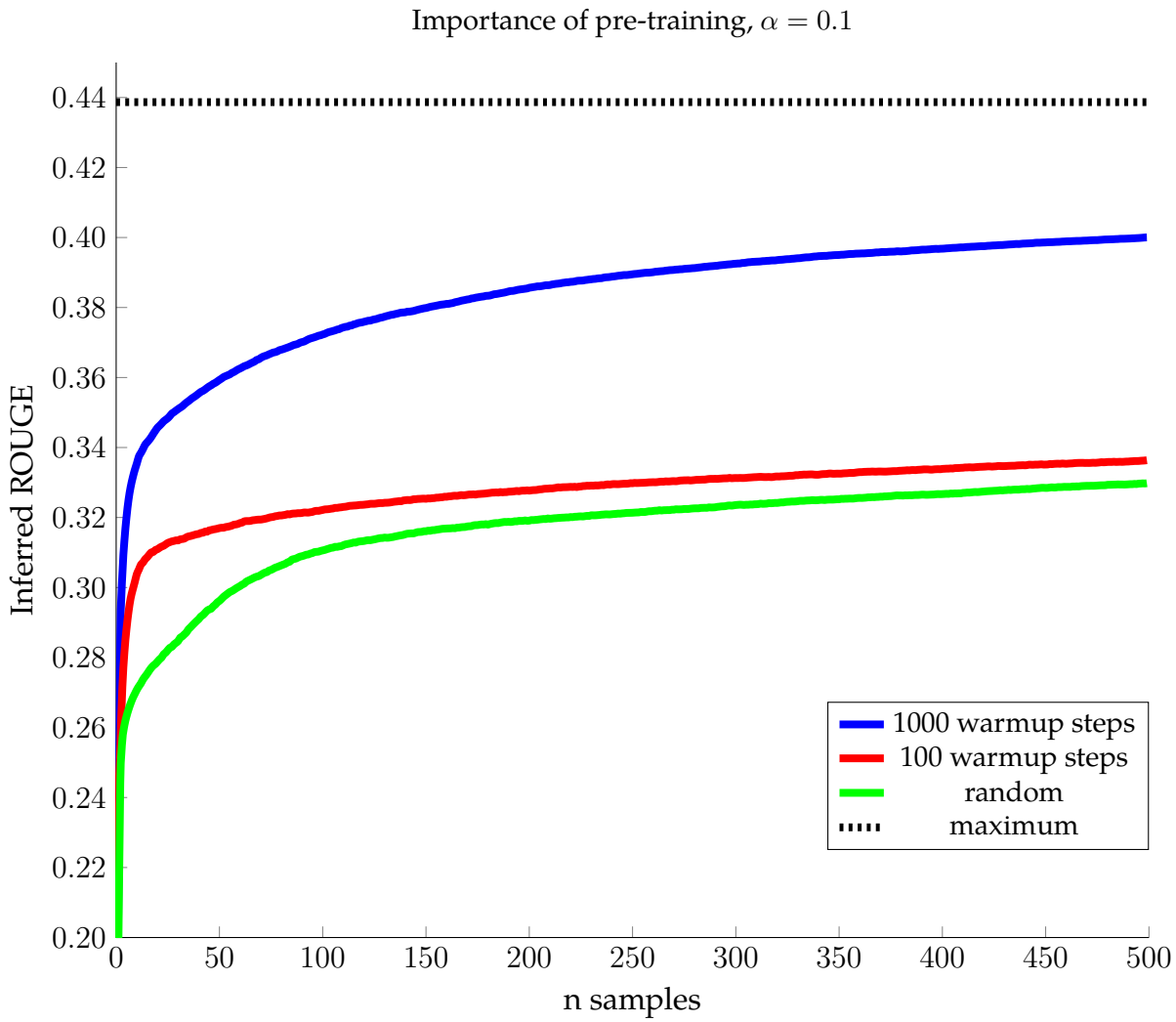
On a des représentations vectorielles $\psi(s)$ des noeuds de l'arbre de résumé. Les représentations sont issues de l'encodeur. On cherche à trouver le mapping linéaire θ minimisant

$$\Big(R(s,\hat{(}s)) - \langle\theta, \psi(s)\rangle\Big)^2 \tag{5.1}$$

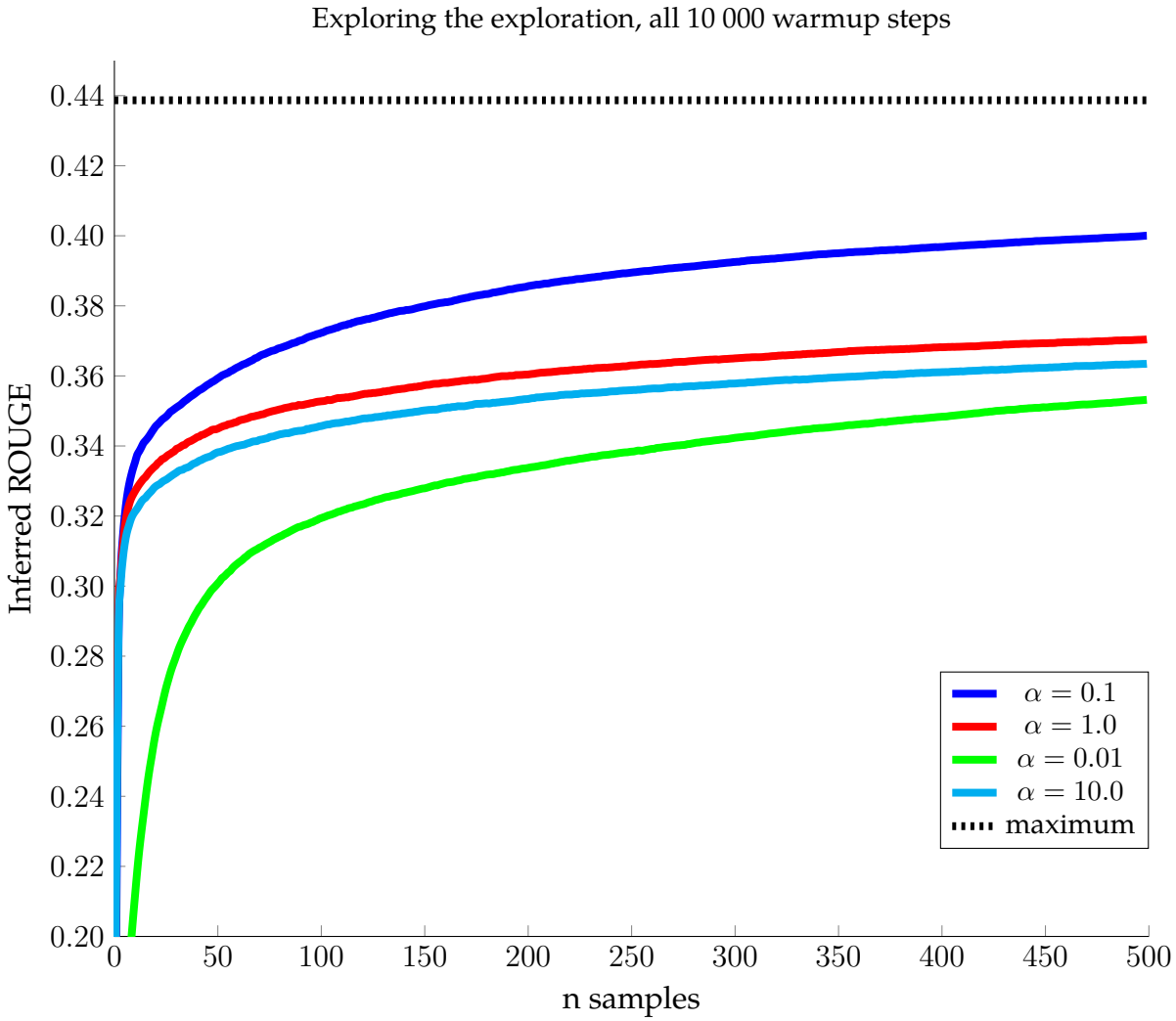
5.2 Recherche d'une approximation du score de tous les résumés

Commentaire: Juste un copier-coller d'un document que j'avais préparé à la fin de l'été.

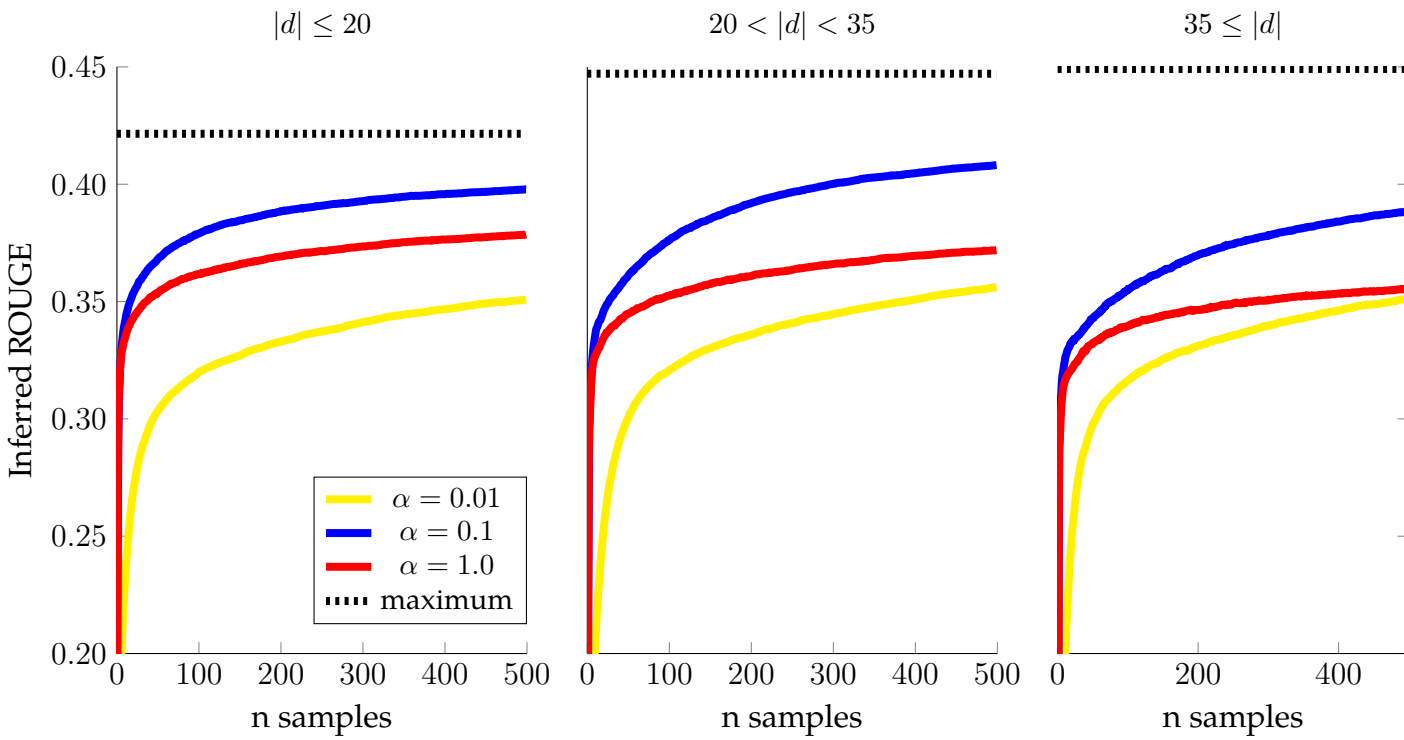
The experiments were done using 25 000 random training documents from the CNN/DailyMail dataset. For the pre-training, a fully connected layer took as input 3 sentence embeddings and was tasked with predicting their relative ROUGE-{1,2,L} scores. For each document in the pre-training phase, 250 summaries of 3 sentences were randomly sampled and their ROUGE scores were used as targets. The pre-training batch size was of 64.



Analysis The pretraining works as intended. After 100 warmup steps, the score prediction head has merely learned which range is adequate for ROUGE-{1,2,L} After 1000 warmup steps, it has however begun to better correlate embeddings to their scores, enabling a much improved MCTS exploration. Less massive batches could be interesting to explore (i.e. instead of sampling 250 summaries per document, move to a more reasonable 16 summaries in a more classical online learning fashion).



Analysis There seems to be a sole winner of $\alpha = 0.1$ as the best exploration parameter for our experiments. It seems this parameter is absolutely crucial as it really influences the performance in both the earlier and later stages of the MCTS.



Analysis Let's first note that there are 8674, 10490 and 5796 documents for each of the respective graphs. The longer the document, the highest is its maximum performing summary. It seems like a single value of α is still the way to go, no matter the document length. Smaller documents are easier to solve for the MCTS, much likely due to the exponentially smaller tree to explore. The convergence is not only better but also faster for smaller documents. This implies that one should attempt to scale the number of samples made by the MCTS with the number of sentences in a document. A linear scaling seems legitimate, starting at 200 samples for documents of 10 sentences and ending at 500 for documents of 50 sentences i.e. $n = \lceil 7.5|d| \rceil + 125$.

Conclusion

— **Pre-training formulation seems to be adequate.** The number of 1000 warmup steps is to be fixed and the number of sampled summaries per document should be tried at the lower value of 16.

— **Should explore values of** $\alpha \in [0.05, 0.5]$.

— Should try scaling number of MCTS samples per document with document length via $n = \lceil 7.5|d| \rceil + 125$.

5.3 Intégration

Le linUCT génère des targets θ, qu'on apprend par perte cosine. À l'inférence, on utilise la linéarité du produit vectoriel et le fait que nos représentations de résumés sont la somme des représentations des phrases pour prédire les 3 phrases dont la représentation a le produit vectoriel maximal avec la mapping prédit.

Bibliographie

P.-A. Coquelin and R. Munos. Bandit algorithms for tree search. In *Proceedings of the Twenty-Third Conference on Uncertainty in Artificial Intelligence*, UAI'07, page 67–74, Arlington, Virginia, USA, 2007. AUAI Press. ISBN 0974903930.

L. Dong, N. Yang, W. Wang, F. Wei, X. Liu, Y. Wang, J. Gao, M. Zhou, and H.-W. Hon. Unified language model pre-training for natural language understanding and generation. In *33rd Conference on Neural Information Processing Systems (NeurIPS 2019)*, 2019.

Y. Dong, Y. Shen, E. Crawford, H. van Hoof, and J. C. K. Cheung. Banditsum : Extractive summarization as a contextual bandit. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 3739–3748, 2018.

C.-Y. Lin. ROUGE : A package for automatic evaluation of summaries. In *Text Summarization Branches Out*, pages 74–81, Barcelona, Spain, July 2004. Association for Computational Linguistics. URL https://www.aclweb.org/anthology/W04-1013.

L. Luo, X. Ao, Y. Song, F. Pan, M. Yang, and Q. He. Reading like HER : Human reading inspired extractive summarization. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 3033–3043, Hong Kong, China, Nov. 2019. Association for Computational Linguistics. doi : 10.18653/v1/D19-1300. URL https://www.aclweb.org/anthology/D19-1300.

Y. Mandai and T. Kaneko. Improved linuct and its evaluation on incremental random-feature tree. In *2016 IEEE Conference on Computational Intelligence and Games (CIG)*, pages 1–8, 2016. doi : 10.1109/CIG.2016.7860440.

T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. In C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 26, pages 3111–3119. Curran Associates, Inc., 2013. URL https://proceedings.neurips.cc/paper/2013/file/9aa42b31882ec039965f3c4923ce901b-Paper.pdf.

R. Nallapati, F. Zhai, and B. Zhou. Summarunner : A recurrent neural network based sequence model for extractive summarization of documents. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*, AAAI'17, page 3075–3081. AAAI Press, 2017.

R. Paulus, C. Xiong, and R. Socher. A deep reinforced model for abstractive summarization. *CoRR*, abs/1705.04304, 2017. URL http://arxiv.org/abs/1705.04304.

J. Pennington, R. Socher, and C. D. Manning. Glove : Global vectors for word representation. In *EMNLP*, volume 14, pages 1532–1543, 2014.

C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21(140) :1–67, 2020. URL http://jmlr.org/papers/v21/20-074.html.

J. F. Ramaley. Buffon's noodle problem. *The American Mathematical Monthly*, 76(8) :916–918, 1969. doi : 10.1080/00029890.1969.12000364. URL https://doi.org/10.1080/00029890.1969.12000364.

A. See, P. J. Liu, and C. D. Manning. Get to the point : Summarization with pointer-generator networks. *CoRR*, abs/1704.04368, 2017. URL http://arxiv.org/abs/1704.04368.

J. Zhang, Y. Zhao, M. Saleh, and P. J. Liu. Pegasus : Pre-training with extracted gap-sentences for abstractive summarization, 2019.

M. Zhong, P. Liu, Y. Chen, D. Wang, X. Qiu, and X. Huang. Extractive summarization as text matching. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 6197–6208, Online, July 2020. Association for Computational Linguistics. doi : 10.18653/v1/2020.acl-main.552. URL https://www.aclweb.org/anthology/2020.acl-main.552.