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.

Les humains sont encore requis dans la loop avec données textuelles pour plusieurs applications en raison de législations ou d'assurance qualité.

Une question se pose : Comment est-ce que l'on peut améliorer l'efficacité d'un système conjoint humain-machine ?

C'est pour répondre à ce besoin que plusieurs applications ajoutent maintenant un intermédiaire qui fait un résumé automatique d'un ou de plusieurs textes pour faciliter la lecture et le traitement par un humain.

Comment fonctionnent ces systèmes de génération de résumés ?

L'espace des résumés possibles est tellement vaste, quelles sont les techniques pour l'explorer de manière efficace ?

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.

Objectifs

Dans ce mémoire, on explorera d'abord comment différentes méthodes de Monte-Carlo peuvent être utilisées pour estimer des fonctions difficilement calculables de manière analytique.

On verra ensuite comment des méthodes peuvent être incorporées dans un système de génération automatique de résumé.

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 roulées pour évaluer la performance du modèle.

Les hyperparamètres de la portion MC seront settés aux valeurs suggérées par les tests empiriques préliminaires.

Chapitre 1

Prérequis

Commentaire: Tout ceci est seulement en place à titre indicatif. Toute information pertinente sera insérée ou retirée en fonction du développement des chapitres de corps du mémoire.

1.1 Méthodes de Monte-Carlo

Approximation statistique de procédés déterministes

1.1.1 Estimation de Monte-Carlo

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

En samplant directement la distribution, on peut converger rapidement à un bon estimé de la vraie valeur.

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

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 à présent, en priorisant les sous-arbres *payants* et ceux qui ont été moins explorés.

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.

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 ?

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 : abstractive et extractive.

2.1 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)

2.2 Formulation extractive

On sélectionne des phrases du ou des documents initiaux.

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.

(Dong et al., 2018; Luo et al., 2019; Zhong et al., 2020) sont des approches représentant l'état de l'art en la matière actuellement.

(Dong et al., 2018; Luo et al., 2019) emploient le même modèle 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.

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.

TODO: Mentionner que la majorité des approches extractives utilisent une heuristique pour déterminer un target *3-hot* mais qu'en mode RL on a des outils pour optimiser directement la métrique ROUGE. Seul hic : le ROUGE est mentionné après.

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), qui 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 utilise généralement

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

pour évaluer un candidat s à partir d'une cible $\hat{s}$. TODO: expliquer mérites et défauts de R1, R2 et RL et donc pourquoi leur moyenne est bonne. Range de R est [0,1]

Question: ROUGE est invariant à l'ordre des phrases. J'utilise cette propriété dans le pré-calcul des ROUGE et dans la recherche arborescente. Je devrais en parler ici ?

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.

Question: Où est-ce que je dis que j'ai précomputé tous les scores ROUGE possibles pour tous les documents ?

Chapitre 3

Échantillonnage

Commentaire: J'ai écrit ce chapitre sous la forme : fonction difficile à calculer -> méthode MC pour approximer -> algo qui se sert de la méthode. À bien y penser, ça serait bien plus naturel d'y aller avec algo à l'allure impossible -> suggestion de fix via MC -> exploration de la viabilité du fix -> implémentation système complet.

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

Concrètement, on cherche donc à approximer

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

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 a seulement besoin de ceux qui sont les plus susceptibles selon $\phi(\pi(d))$. À mentionner : la différence de probabilité entre deux résumés finit par outrepasser la différence de R entre les deux mêmes résumés, c'est aussi pour ça que ça marche.

On procède donc avec une approche nommé Échantillonnage de Monte-Carlo, où on calcule plutôt

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

avec les résumés s_t échantillonnés selon $\phi(\pi(d))$. Notons que $\tilde{f}$ est un estimateur non-biaisé de f.

3.1 Expériences sur l'échantillonnage

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

On pige aléatoirement des distributions $\pi(d)$ qui sont une moyenne pondérée entre une distribution uniforme $U(d)$ et une distribution *3-hot* $G(d)$ sous la forme

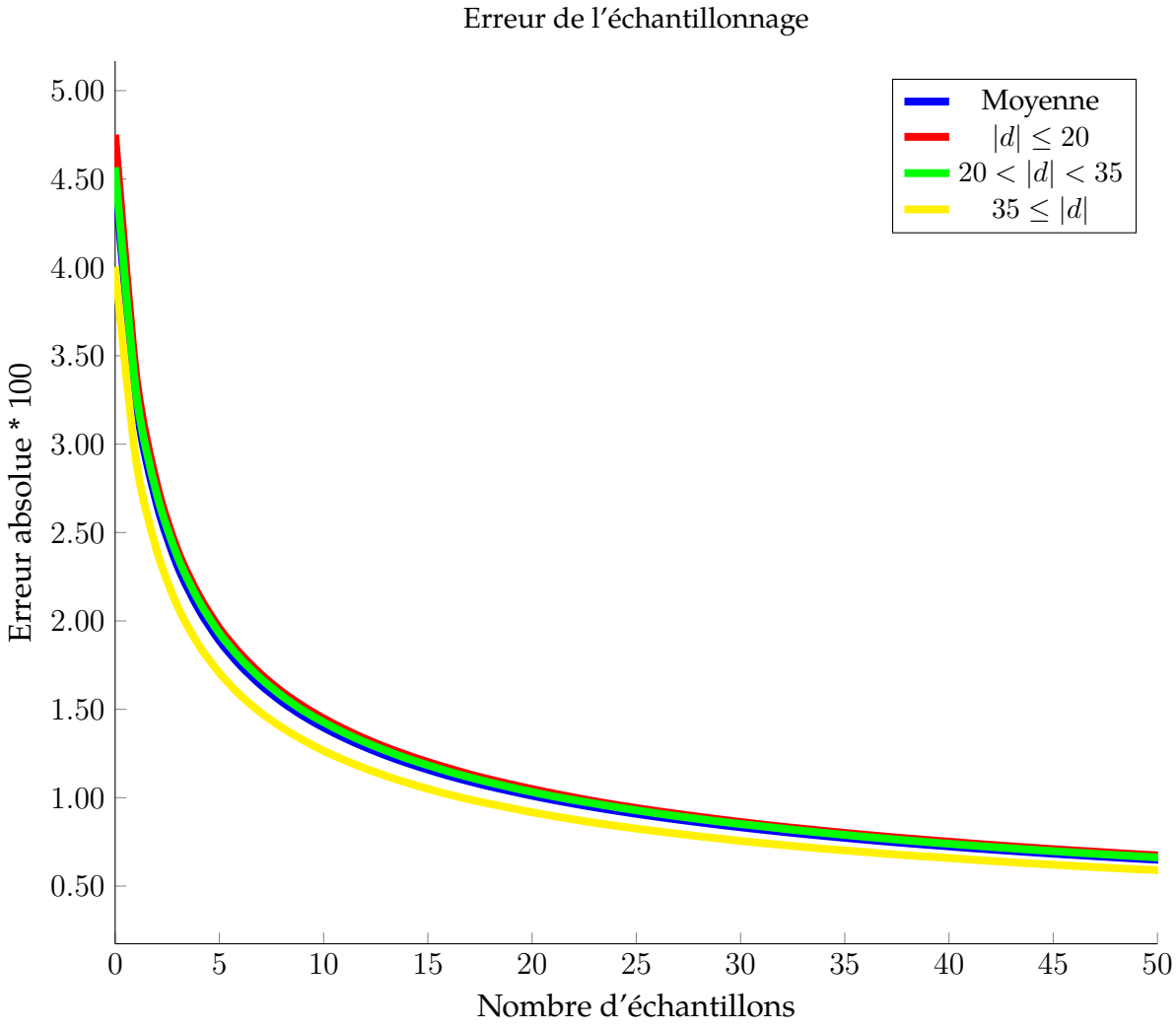
$$\pi(d) = (1 - \tau)U(d) + \tau G(d). \tag{3.3}$$

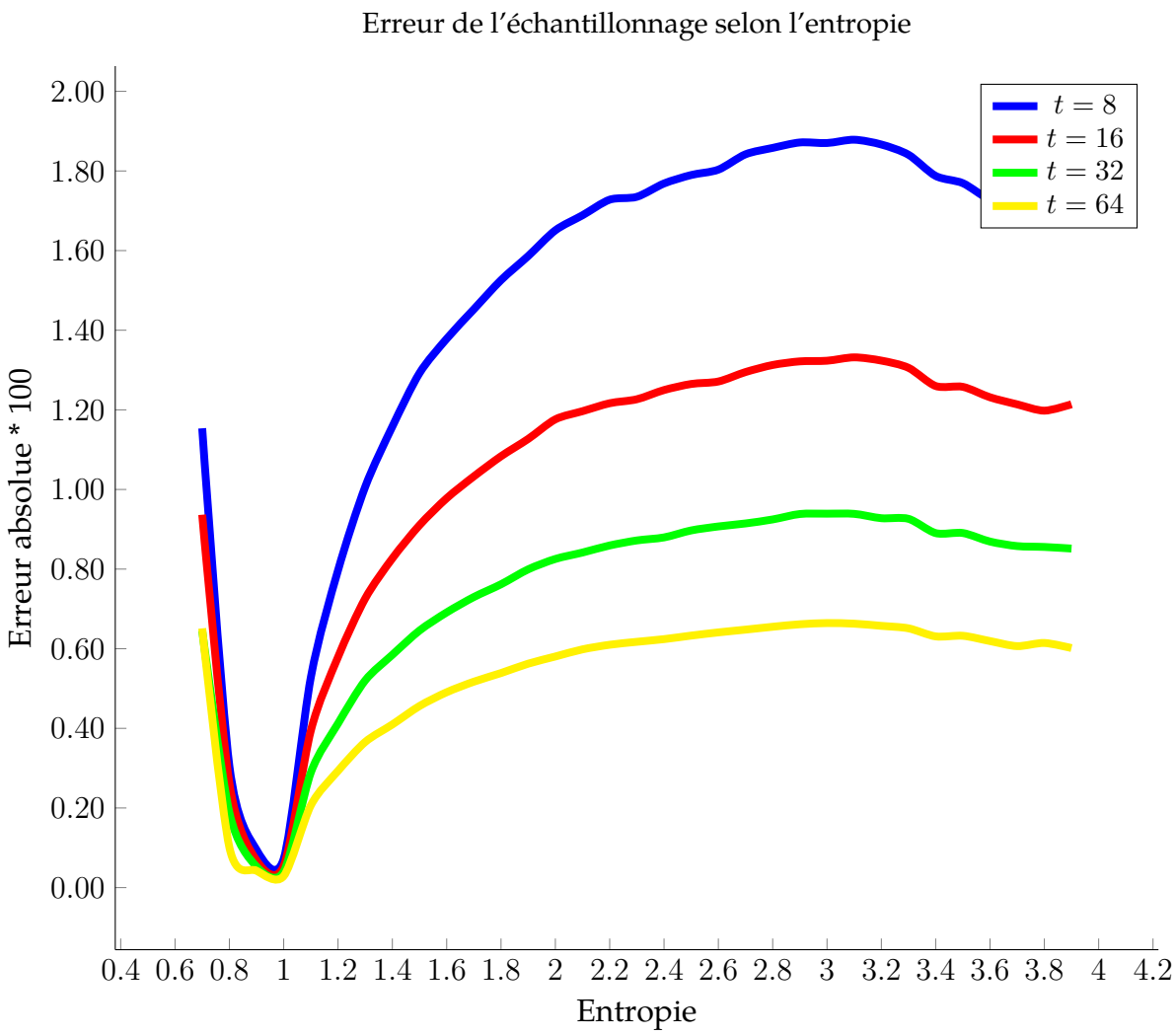
On explore $\tau \in \{\frac{n}{100} : 0 \leq n \leq 100\}$ en repigeant les 3 index non-nuls de $G(d)$ à chaque fois. On ajoute un bruit gaussien sur chaque $\pi(d)$.

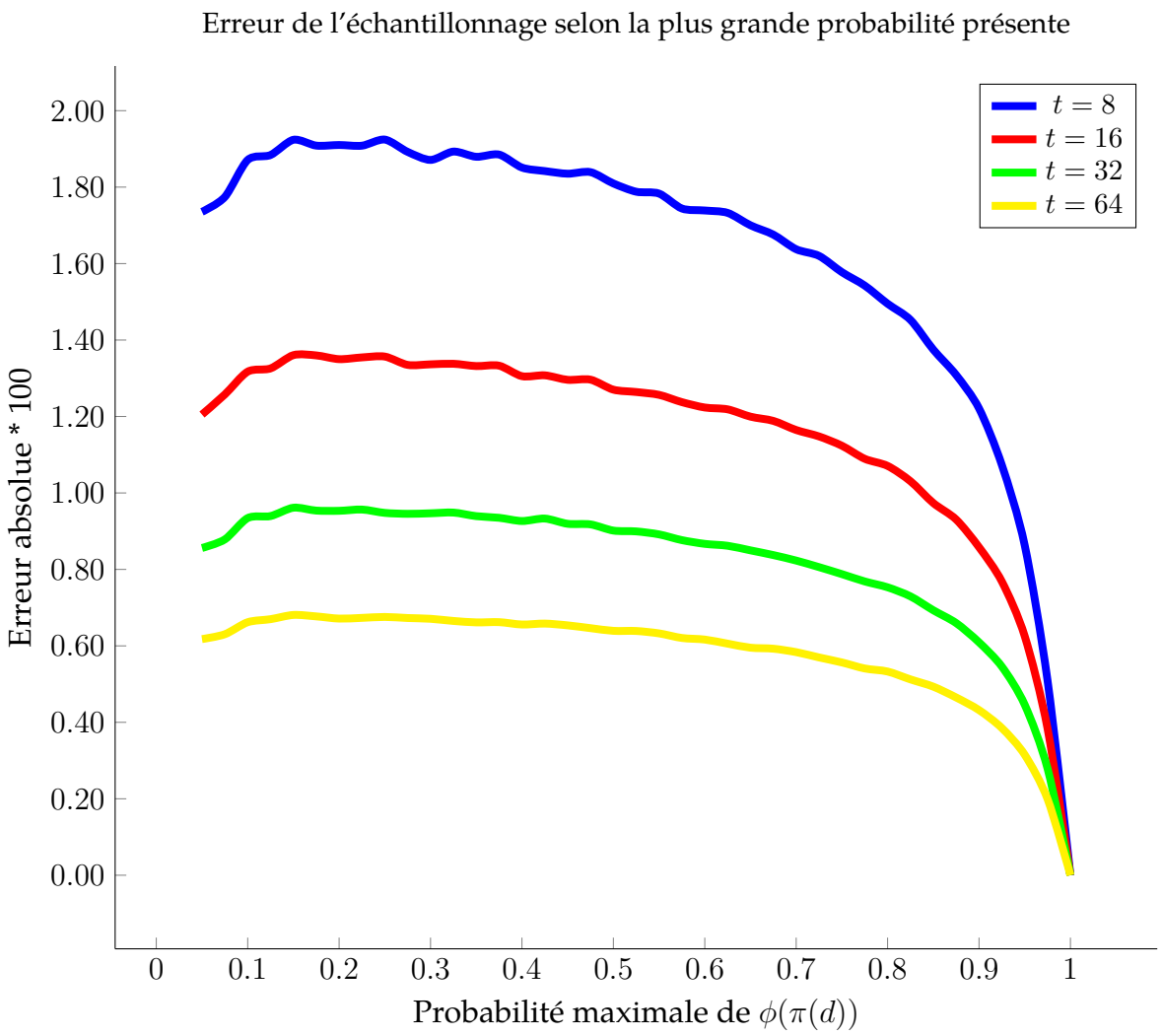
On calcule analytiquement f grâce au fait que l'on possède tous les scores ROUGE de tous les résumés possibles pour les documents du jeu de données. On pige 1000 échantillons de $\phi(\pi(d))$, où ϕ est le processus de pige sans remise de 3 phrases. On calcule ainsi $\tilde{f}_t$ pour t jusqu'à 1000.

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.


Erreur de l'échantillonnage
Moyenne
$|d| \leq 20$
$20 < |d| < 35$
$35 \leq |d|$
Erreur absolue * 100
5.00
4.50
4.00
3.50
3.00
2.50
2.00
1.50
1.00
0.50
0
5
10
15
20
25
30
35
40
45
50
Nombre d'échantillons


Erreur de l'échantillonnage selon l'entropie
Erreur absolue * 100
Entropie
$t = 8$
$t = 16$
$t = 32$
$t = 64$
2.00
1.80
1.60
1.40
1.20
1.00
0.80
0.60
0.40
0.20
0.00
0.4 0.6 0.8 1 1.2 1.4 1.6 1.8 2 2.2 2.4 2.6 2.8 3 3.2 3.4 3.6 3.8 4 4.2



Analyse :

— Absolument aucun impact de la taille des documents

— Bizarre le creux à 1 pour l'entropie.

— Semblerait que $t = 16$ est un bon compromis pour être rapide mais pas trop faire d'erreurs. C'est ce que BanditSUM ont aussi.

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, on peut se limiter à faire une ascension de gradient sur l'estimateur non-biaisé

$$\tilde{J}(\theta) = \frac{1}{N} \sum_{t=1}^{N} R(s_t, \hat{s}). \tag{3.5}$$

On peut alors utiliser l'ascension de gradient et mettre à jour les poids θ selon la règle

$$\theta = \theta + \nabla_\theta \tilde{J}(\theta).$$

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 stabilier les gradients de J et prend un document à la fois, en posant $N = 16$.

TODO: Détails du processus d'entraînement complet BanditSUM

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.

Commentaire: Les expériences pour BanditSUM sont en train de rouler. Je pense surtout rapporter la courbe de performance. 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

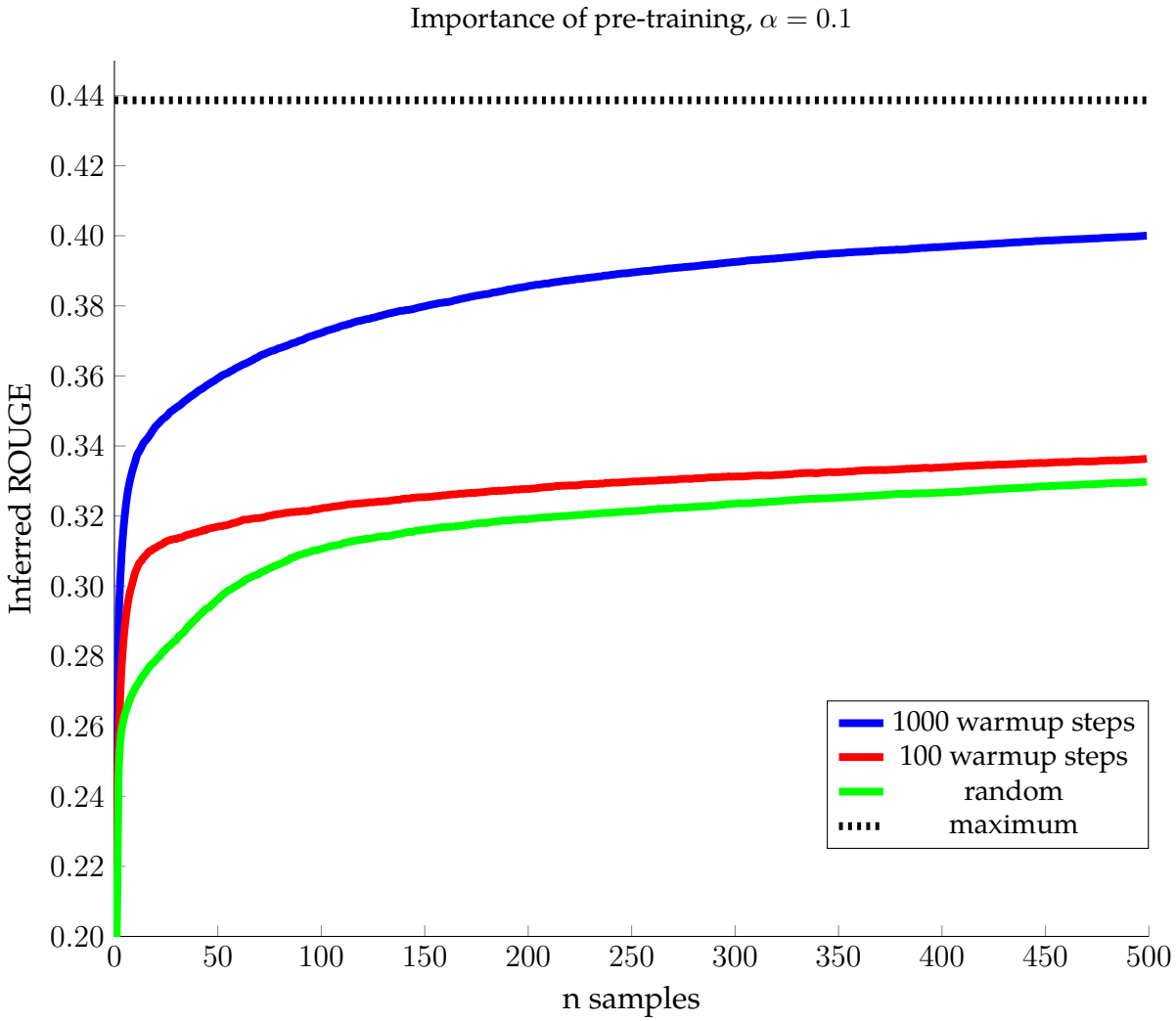
On a des représentations vectorielles $\psi(s)$ des noeuds de l'arbre de résumé. Les représentations sont issues de l'encodeur. On cherche à trouver le mapping linéaire θ minimisant

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

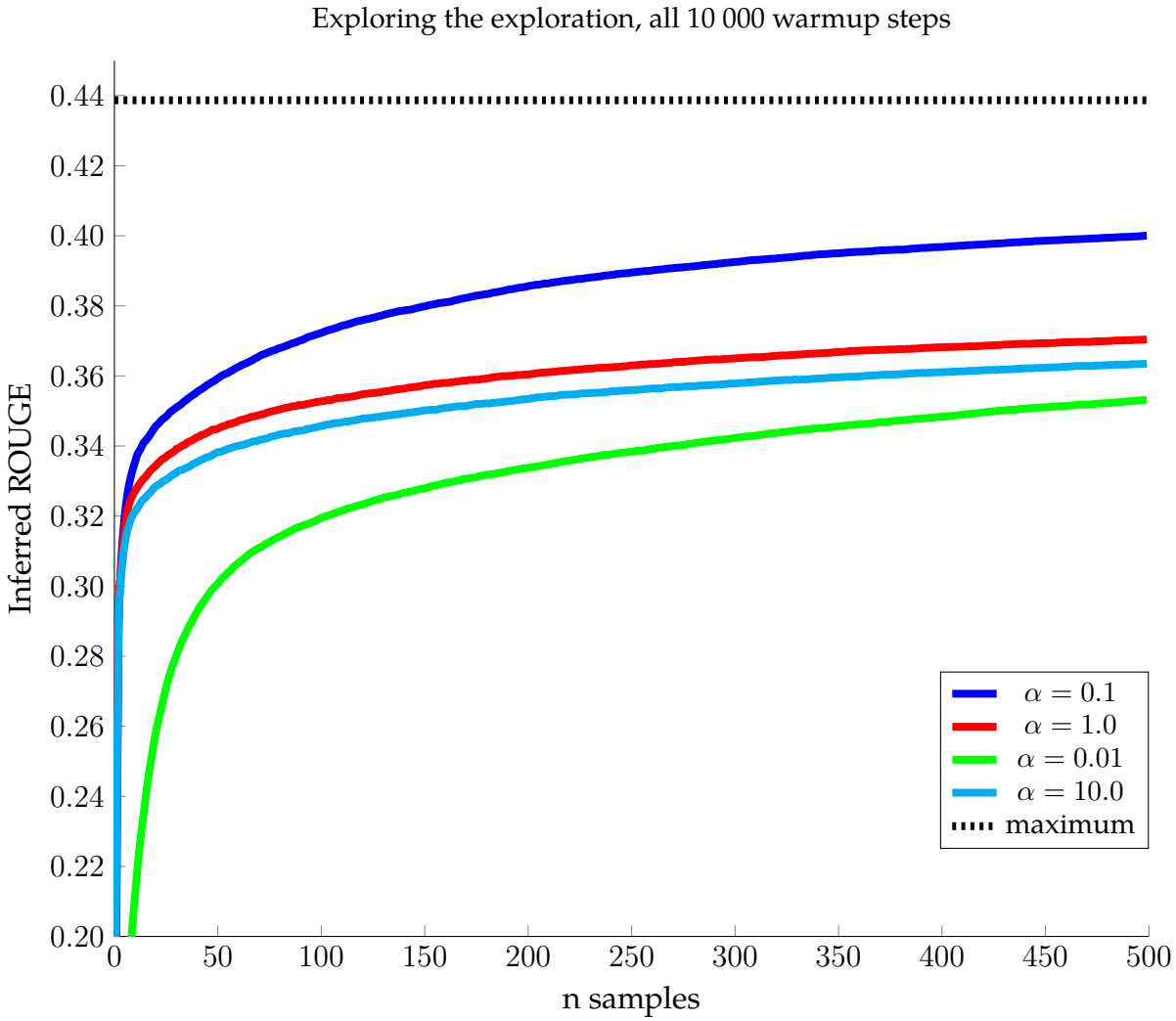
5.2 Recherche d'une approximation du score de tous les résumés

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

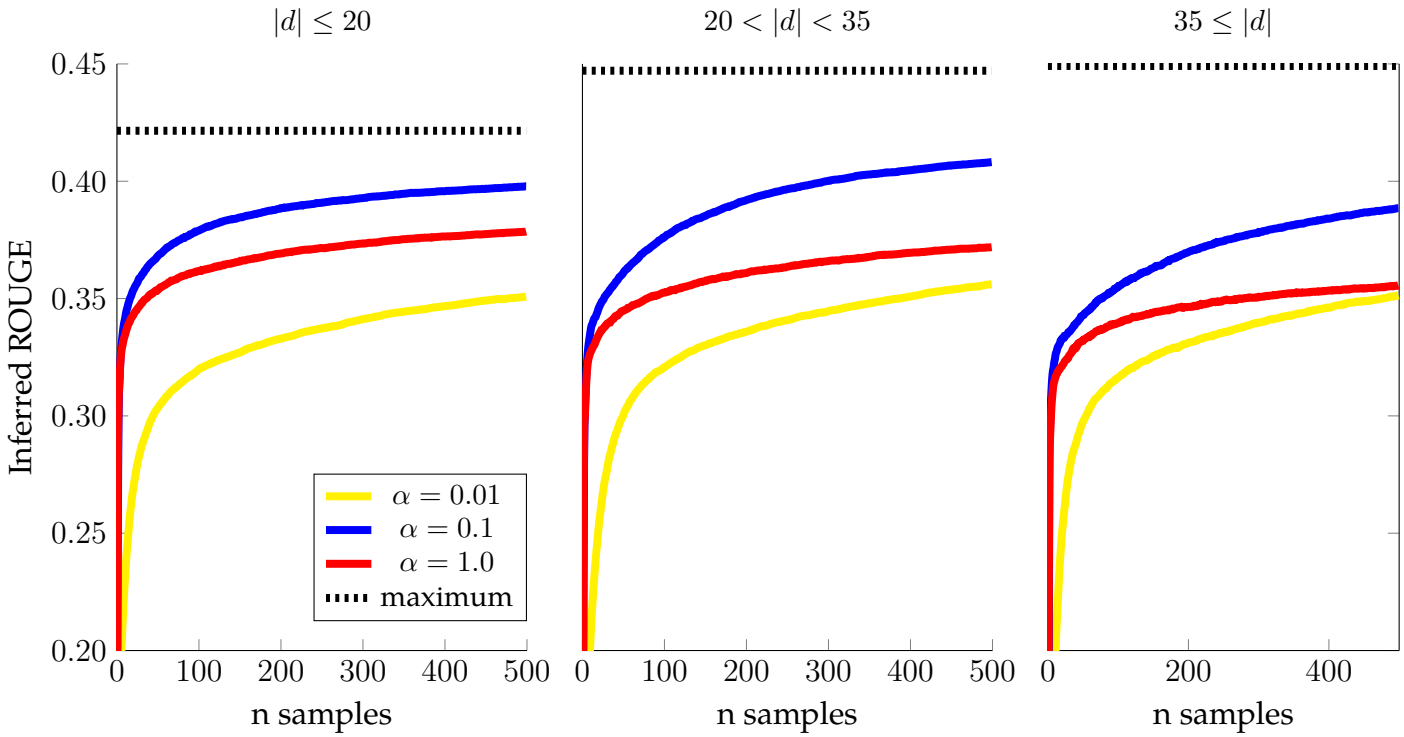
The experiments were done using 25 000 random training documents from the CNN/DailyMail dataset. For the pre-training, a fully connected layer took as input 3 sentence embeddings and was tasked with predicting their relative ROUGE-{1,2,L} scores. For each document in the pre-training phase, 250 summaries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

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.

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.

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 Lin-

guistics. doi : 10.18653/v1/2020.acl-main.552. URL https://www.aclweb.org/anthology/2020.acl-main.552.