Approches de bandits 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

Liste des algorithmes

<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

Commentaire: Toute la portion de mise en contexte n'est pas touchée. Pas pertinent de la lire, comme elle n'incorpore pas encore les approches par bandit.

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

Commentaire: À partir d'ici c'est mis à jour et prêt à révision.

Répondre aux questions suivantes :

— Comment peut-on formuler la génération de résumés comme un problème de bandit multi-bras (MAB) ? Quels sont les algorithmes de la littérature utilisés habituellement pour résoudre ces problèmes ?
— Dans quelle mesure ces algorithmes de bandit sont-ils performants dans le contexte de résumés ? Sont-ils plus rapides en temps de calcul ? En nombre d'échantillons requis pour une bonne performance ? Est-ce que les solutions auxquelles ils convergent sont meilleures ?
— Comment peut-on intégrer ces formulations bandits dans un algorithme qui produit un système de génération de résumés ?
— Comment l'introduction de ces artefacts bandits a-t-elle affecté la performance ? Est-ce que les systèmes produits génèrent des résumés de meilleure qualité ? A-t-on besoin de moins d'exemples avant d'avoir un système de haute qualité ? Est-ce que l'entraînement est plus rapide à effectuer ? Avec quelles ressources de calcul ?
— Comment les performances de ces algorithmes se comparent-ils entre eux ? Et par rapport à l'état de l'art ?

On répond aux objectifs en proposant 3 formulations de bandits applicables au processus de la génération de résumés.

Une des approches proposée est présente dans la littérature mais les deux autres sont des nouveautés, du mieux de notre connaissance.

Structure du mémoire

Ce mémoire débutera par une introduction aux concepts de base essentiels à la compréhension du reste du document.

Le document sera ensuite divisé en trois chapitres pour les trois formulations de MAB explorées : bandit contextuel, bandit combinatoire et bandit combinatoire linéaire.

Pour chacun des chapitres, on définira le cadre MAB proposé. Des expériences empiriques seront effectuées pour justifier l'applicabilité du cadre au contexte de génération de résumés.

On proposera ensuite un modèle neuronal de génération automatique de textes utilisant le cadre proposé.

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

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

Chapitre 1

Concepts de base pertinents aux travaux

Ce chapitre vise à présenter les concepts de base clés à la compréhension des travaux présentés dans le reste du document. Les concepts abordés ici sont accompagnés d'une explication aussi dénuée de notions non-nécessaires que possible aux travaux de ce mémoire, afin de garder leur présentation aussi digeste que possible. À cette fin, les concepts des algorithmes de bandits, des réseaux de neurones et de la génération automatique de résumés, tous essentiels à la compréhension de ce document, sont donc présentés dans ce chapitre. À la fin de ce chapitre, le lecteur trouvera ainsi une présentation formelle du problème exploré dans ce document, sous toutes les contraintes et hypothèses retenues.

1.1 Approches de bandits

Formulation générale Multi-Armed Bandit (MAB) stochastique : Problème pour lequel on a un ensemble $\mathcal{A}$ de k actions disponibles (bras). À chaque action a est associée une récompense $X_{a,t}$ pour la t-ème fois qu'elle est choisie. Les récompenses ne sont pas nécessairement identiques pour tout t, mais on suppose qu'il existe μ_a tel que $\mu_a = \mathbb{E}[X_{a,t}]$.

On définit alors l'action optimale $a^* = \arg\max_{a \in \mathcal{A}} \mu_a$. Conjointement, on a aussi $\mu^* = \mu_{a^*}$, l'espérance du bras optimal. L'objectif d'un algorithme visant à résoudre le MAB est de sélectionner successivement des actions a_t et de maximiser les récompenses X_t perçues. Pour normaliser la formulation à des récompenses d'amplitude variée, la métrique de performance utilisée est le regret cumulatif

$$\rho_n = n\mu^* - \sum_{t=1}^{n} X_t, \tag{1.1}$$

qui représente la différence entre la récompense qui aurait été reçue en prenant l'action optimale et celle reçue par l'action sélectionnée. Les algorithmes sont donc généralement formulés de sorte à minimiser le regret cumulatif.

Il est important de comprendre que l'objectif de minimisation du regret cumulatif ne consiste pas exactement à trouver l'action optimale. En effet, comme le regret cumulatif inclut les erreurs commises depuis le début de l'apprentissage, un bon algorithme au sens de la minimisation du regret cumulatif doit à la fois converger à une action presque optimale, mais il doit aussi le faire en mettant de côté rapidement les actions les plus sous-optimales.

En pratique, la formulation MAB est donc la plus appropriée pour des problèmes où il est crucial de ne pas commettre trop d'erreurs durant l'apprentissage. Notamment, diverses déclinaisons des bandits ont été appliquées avec succès dans les domaines des essais cliniques (Kuleshov and Precup, 2014) et de la gestion de portfolios financiers (Shen et al., 2015).

1.2 Réseaux de neurones

Classe de fonctions h_θ paramétrées par des poids $\theta \in \mathbb{R}^n$ qui peuvent être appliquées pour apprendre virtuellement n'importe quelle fonction f à l'étude. Pour apprendre, les réseaux de neurones utilisent des approches de descente de gradient sur une fonction de perte $\mathcal{L}(\theta)$. Le fonction de perte, dont la forme dépend de celle de la fonction à l'étude, sert de guide d'apprentissage au réseau. Essentiellement, pour des paramètres θ et θ' pour lesquels on a $\mathcal{L}(\theta) < \mathcal{L}(\theta')$, on a que les paramètres θ produisent une approximation h_θ plus près de la fonction cible f. Éventuellement, l'apprentissage d'un réseau de neurones a pour objectif ultime de trouver les paramètres optimaux

$$\theta^* = \arg\min_{\theta \in \mathbb{R}^n} \mathcal{L}(\theta).$$

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.

— Unité de base : le neurone. Neurone prend entrée un vecteur de caractéristiques numériques. Le neurone fait une somme pondérée sur les caractéristiques en entrée et applique une fonction d'activation pour obtenir une couche de non-linéarité.

— Fonction d'activation : ReLU (Xu et al., 2015).

— Les neurones sont regroupées ensemble en couches, où un vecteur de caractéristiques est passé à travers une séquence de couches, où la sortie d'une couche représente l'entrée de la prochaine.

— θ est un ensemble de matrices, où chaque matrice représente les poids reliant le neurone i d'une couche à la caractéristique j en sortie de la couche précédente.

— Fonction de perte quadratique : permet une convergence rapide en pénalisant sévèrement les erreurs graves et peu les erreurs très faibles.

1.2.2 Réseaux récurrents

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

Juste présenter le LSTM (Gers, 1999). Architecture qui incorpore les notions de mémoire à court et à long termes pour exploiter les relations séquentielles courtes et longues présentes dans une même séquence en entrée. Important à noter : pour une séquence $[x_1, ..., x_n]$ en entrée, un LSTM produit une séquence $[\hat{y_1}, ..., \hat{y_n}]$ de vecteurs $\hat{y_i}$ pour chacun des éléments.

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.

1.3 Génération automatique de résumés

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 d pour lequel nous possédons un seul résumé cible s.

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.

1.3.1 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 a un document en entrée d qui contient $|d|$ phrases $[d_1, ..., d_{|d|}]$ et pour lequel on a un résumé cible s. On dit qu'on a un modèle de génération de résumés à deux composantes (π, ϕ). Ici, π prend en entrée un document d et retourne $\pi(d) \in [0, 1]^{|d|}$, 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 $\pi(d)$. Concrètement, en définissant $\mathscr{P}(d)$ comme étant le power set de l'ensemble des phrases de d (i.e. $\mathscr{P}(d)$ contient tous les résumés extractifs possibles de d), on a la signature $\phi(\pi(d)) \in \mathscr{P}(d)$. 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.

Enfin, pour un problème de génération de résumés donnés, on fixe habituellement ϕ pour tous les documents. L'objectif d'apprentissage est alors de trouver des paramètres θ qui permettent de maximiser la qualité des résumés produits par le modèle (π_θ, ϕ) par rapport à des paires document-résumé cible (d, s) contenues dans un jeu de données $\mathcal{D}$. La forme habituellement retenue pour π_θ est celle d'un réseau de neurones récurrent LSTM.

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: Figure tikz qui prend pas mal une page complète décrivant le processus de résumé extractif selon la formalisation donnée. Les étapes sont :

- Document d en entrée, visuellement séparé pour voir les phrases.
- Application de π_θ sur le document : flèche indiquant explicitement que l'entrée est le document d et la sortie est une distribution.
- Application de ϕ sur la distribution des phrases : flèche indiquant explicitement que l'entrée est une distribution sur les phrases et la sortie est un résumé.
- Évaluation de la qualité : Comparaison entre le résumé produit et le résumé cible s.

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) précalculent des cibles binaires $y_d \in \{0,1\}^{|d|}$ utilisées comme cibles pour chaque document d. Leurs cibles binaires sont obtenues en sélectionnant de manière vorace les phrases d_i permettant de générer un résumé extractif aussi près que possible du résumé cible s. Après avoir sélectionné trois phrases, leur processus est arrêté et la cible y_d est fixée à un vecteur binaire où les index des trois phrases retenues ont une valeur de 1 et les autres index ont une valeur nulle.

L'emploi de cibles binaires permet un entraînement supervisé très efficace. Le réseau entraîné sur ces cibles ne peut toutefois pas obtenir une performance supérieure à celle de l'heuristique employée. Il s'agit du principal facteur limitant de la performance de ces approches supervisées.

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

Approches par renforcement

Au lieu d'utiliser des cibles binaires, les approches par renforcement visent à optimiser directement une mesure numérique de la similarité entre deux résumés. Disons que l'on dispose d'une métrique numérique de performance $R(\hat{s}, s)$ permettant de quantifier la proximité entre un résumé produit $\hat{s}$ et un résumé cible s. Il est alors possible de définir la fonction objective à maximiser

$$J(\theta) = \mathop{\mathbb{E}}_{(d,s)\sim\mathcal{D}} \mathop{\mathbb{E}}_{s\sim\phi(\pi_\theta(d))} \left[R(\hat{s}, s)\right], \tag{1.2}$$

représentant l'espérance de performance des résumés produits avec la paramétrisation θ. Comme on recherche θ^* maximisant $J(\theta)$, il vient naturellement en tête de faire appel à des méthodes d'ascension de gradient. Or, J n'est pas calculable analytiquement en pratique car les deux espérances (sur tous les documents et sur tous les résumés générables) sont habituellement intractables.

L'algorithme REINFORCE (Williams, 1992) propose une solution élégante à ce problème, faisant une mise à jour des poids θ selon l'approximation du gradient de $J(\theta)$

$$\nabla J(\theta) = R(\hat{s}, s)\nabla \ln \pi_\theta(\hat{s}). \tag{1.3}$$

Le théorème du gradient de politique (Sutton et al., 1999) affirme que, en espérance, l'approximation 1.3 correspond au véritable gradient 1.2 et peut donc être utilisée pour une ascension de gradient. Les approches par renforcement basées sur REINFORCE (Dong et al.,

2018; Luo et al., 2019) parviennent actuellement à atteindre des performances au niveau de l'état de l'art pour la génération automatique de résumés.

Au niveau architectural, (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 2. Nous reprendrons une architecture similaire de réseau de neurones pour toutes les expérimentations.

1.3.2 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.

1.3.3 Évaluation de la performance

À la section 1.3.1, on prenait pour acquis qu'une fonction R existait pour distinguer quantitativement deux résumés candidats s et $\hat{s}$. Il en existe en fait plusieurs, notamment :

— ROUGE-n (Lin, 2004) : métrique basée sur le chevauchement entre les séquences de n mots (n-grammes) s et $\hat{s}$;

— ROUGE-L (Lin, 2004) : métrique mesurant la plus longue sous-séquence commune entre s et $\hat{s}$;

— ROUGE-WE (Ng and Abrecht, 2015) : métrique similaire à ROUGE-n, mais qui utilise le *soft-matching* basé sur la similarité cosine entre les plongements de mots au lieu du *matching* exact.

Intuition : 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.

(Peyrard, 2019) démontre que, bien que ces métriques soient généralement corrélées avec l'avis d'un expert humain, elles peuvent difficilement être considérées comme un rempla-

cement de celui-ci. En effet, comme les métriques sont toutes similairement corrélées avec le jugement humain mais qu'elles ne sont que faiblement corrélées entre elles, aucune des métriques ne peut être utilisée à elle seule comme un remplacement de l'avis humain. En utilisant une moyenne de plusieurs métriques, il est possible d'alléger quelque peu ce problème. Notons toutefois que même l'utilisation d'une moyenne ne suffit pas à régler le problème : (Paulus et al., 2017) entraînent par renforcement sur la formulation abstractive mais finissent par ne pas utiliser le modèle avec le meilleure score employé (moyenne de ROUGE-1, ROUGE-2 et ROUGE-L).

1.4 Formulation du problème retenue

Pour les travaux présentés, il est choisi de s'intéresser exclusivement à la génération automatique de résumés à partir d'un document en entrée. Aussi, la formulation extractive est retenue en raison de la facilité avec laquelle les approches par bandit peuvent y être utilisées. Pour évaluer la performance des approches proposées, le jeu de données du CNN/DailyMail (Nallapati et al., 2017), référence dans le milieu de la génération de résumés, sera utilisé. Celui-ci est composé de plus de 300 000 articles de journaux et leurs résumés écrits par un expert humain.

Nous emploierons aussi plusieurs des contraintes retenues par les approches de l'état-de-l'art sur le jeu de données du CNN/DailyMail. Notamment, comme (Dong et al., 2018), nous considérerons seulement les résumés de 3 phrases en sortie. Cette contrainte, utilisée fréquemment sur le jeu de données du CNN/DailyMail, est basée sur le fait que la moyenne du nombre de phrases contenues dans les résumés cibles est de 3. Aussi, une référence souvent utilisée pour ce jeu de données est l'heuristique *Lead-3* (Nallapati et al., 2017) qui consiste à générer le résumé d'un document en retenant les 3 premières phrases. Les résultats obtenus par *Lead-3* sont très bons aux yeux des métriques automatiques, justifiant encore une fois qu'avec 3 phrases on peut générer de bons résumés des documents du jeu de données. Nous considérerons aussi la régularisation du jeu de données qui consiste à conserver seulement les documents d'au moins 3 phrases, limiter la taille des documents à 50 phrases au maximum et celle des phrases à 80 mots. En cas d'excès de mots ou de phrases, l'excédent est simplement retiré.

Pour l'évaluation, on retient la métrique de similarité entre deux résumés utilisée par la plupart des travaux employant l'apprentissage par renforcement (Dong et al., 2018; Luo et al., 2019) :

$$R(\hat{s}, s) := \frac{1}{3} \left[R_1(\hat{s}, s) + R_2(\hat{s}, s) + R_L(\hat{s}, s) \right]. \tag{1.4}$$

Ce choix de R est motivé par :

— Diversité de signal, comme expliqué plus haut.

— R est **presque** invariante à l'ordre des phrases dans un résumé (seulement ROUGE-2 n'est pas invariant, mais seulement au niveau de la jonction entre deux phrases). Cette propriété permet d'éviter l'explosion combinatoire induite par la considération de l'ordre des phrases dans les approches par bandits.

Chapitre 2

Bandit contextuel

La première approche explorée est celle qui formule la génération de résumés comme un bandit contextuel, initialement proposée par (Dong et al., 2018) Dans ce chapitre, on présentera la formulation en bandit contextuel et on évaluera son applicabilité en fonction de ses hypothèses sur un jeu de données de développement. On montrera ensuite comment la formulation en bandit contextuel peut naturellement être utilisée avec l'algorithme REINFORCE Williams (1992) pour obtenir une performance représentant l'état de l'art.

2.1 Formulation

Un bandit contextuel est un problème de bandit où un contexte c_t est perçu avant de prendre une action a_t et de percevoir une récompense X_t possiblement dépendante du contexte c_t. La présence de ce contexte permet d'apprendre à prédire des actions différentes pour des entrées différentes. En génération de résumés, cela correspond à l'hypothèse facilement vérifiable que ce n'est pas nécessairement une bonne stratégie de toujours sélectionner les mêmes index de phrases d'un document pour bâtir son résumé.

Concrètement, pour une paire document-résumé (d, s), on définit le contexte comme étant le document $(c = d)$, les actions sont les groupes non-ordonnés de 3 phrases possibles à partir de d et les récompenses représentent la valeur de $R(\hat{s}, s)$, où $\hat{s}$ est le résumé bâti à partir de l'action retenue. On ne s'intéresse pas à l'ordre des 3 phrases d'une action car, tel que mentionné à la section 1.3.3, le score R utilisé est presque totalement invariant à l'ordre. Ainsi, on choisit de générer le résumé $\hat{s}$ en fonction de a en insérant les phrases dans l'ordre dans lequel elles apparaissent dans le document original. Selon le formalisme de génération de résumés défini à le section 1.3.1, on se retrouve donc avec un bandit π qui, pour un document d en entrée, retourne une distribution $\pi(d) \in [0,1]^{|d|}$. On choisit aussi le processus de génération de résumés ϕ stochastique, qui pige 3 phrases sans remise selon $\pi(d)$.

2.2 Approximation de la performance par échantillonnage

Si l'on considère un bandit π_θ doté d'une certaine paramétrisation θ, l'algorithme REINFORCE (1.3.1) peut être utilisé pour faire une ascension de gradient pour la fonction de récompense $J(\theta)$ selon (1.3). En pratique, les approximations de $\nabla J(\theta)$ générées en utilisant un seul échantillon de J sont très instables pour un processus stochastique ϕ et rendent l'apprentissage difficile (TODO: source). Une solution simple et peu coûteuse pour stabiliser l'entraînement est de piger plusieurs résumés $\hat{s}_t$ pour bâtir un meilleur estimateur selon

$$\nabla J(\theta) = \frac{1}{N} \sum_{t=1}^{N} R(\hat{s}_t, s) \nabla \ln \pi_\theta(\hat{s}_t). \tag{2.1}$$

2.2.1 Expériences

Dans l'équation (2.1), la dérivée de π est bien définie et n'est pas sensible à la portion stochastique du processus de génération.

Pour justifier l'utilisation de l'échantillonnage, il faut donc seulement s'intéresser à la différence entre l'approximation de la fonction de récompense et sa véritable valeur pour un document donné. On prend J défini selon (1.2), on pose $\bar{J}_N(\theta) = \frac{1}{N} \sum_{t=1}^{N} R(\hat{s}_t, s)$, son approximation par échantillonnage. La qualité de l'échantillonnage dépend de la proximité entre J et $\hat{J}_N$, que nous nommons l'erreur d'échantillonnage Δ_N pour

$$\Delta_N = \left| J(\theta) - \bar{J}_N(\theta) \right|. \tag{2.2}$$

Le pseudocode permettant de générer les échantillonnages se trouve dans l'algorithme 2. Pour approximer $\hat{J}_N$, il suffit de prendre la moyenne de la liste d'échantillons retournés.

Algorithme 1 Échantillonnage de J

Entrée: π (distribution de phrases), ϕ (processus de génération de résumés), N (nombre d'échantillons), s (résumé cible).

1: **procédure** ECHANTILLONNAGE(π, ϕ, N, s)
2: **pour** $i = 1, ..., N$ **faire**
3: Piger $\hat{s}_i \sim \phi(\pi)$
4: Recevoir la récompense $r_i = R(\hat{s}_i, s)$
5: **retourner** $\{r_i\}_{i=1}^{N}, \{s_i\}_{i=1}^{N}$

Pré-calcul des scores ROUGE

Les méthodes présentées dans ce mémoire utilisent toutes un grand nombre de scores R dans leur procédure d'entraînement.

Or, le calcul du ROUGE entre deux résumés est lent et 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. Pour un groupe de 3 phrases, R est seulement calculé pour la version où l'ordre des phrases dans les résumé est le même que celui dans le document original.

Méthodologie

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

Pour les tests, on prend 25 000 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* $G(d)$:

$$\pi(d) = (1 - \tau)U(d) + \tau G(d). \tag{2.3}$$

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 $\pi(d)$.

Les expériences consistent à calculer la valeur de Δ_N pour différentes valeurs de N allant jusqu'à 100. L'approximation $\hat{J_N}$ est obtenue en suivant l'algorithme 2. On calcule analytiquement J grâce au fait que l'on possède tous les scores ROUGE de tous les résumés possibles pour les documents du jeu de données.

2.2.2 Résultats

TODO: Ajouter 95 conf int sur les graphes ? Probablement juste sur 2.2

Les résultats obtenus sont rapportés dans les figures 2.1 et 2.2.

Tout d'abord, sur la figure 2.1, on remarque que le nombre de phrases dans un document n'est pas un facteur déterminant sur la rapidité de convergence de l'approximation par échantillonnage. 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: Préciser nb de documents dans chaque bracket de nombre de phrases.

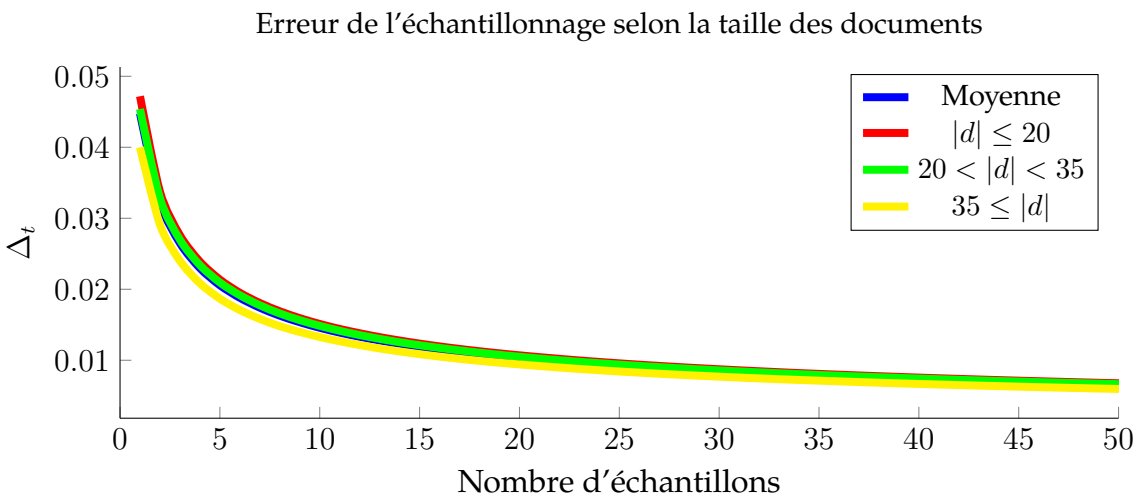

FIGURE 2.1 – Impact de la taille du document en entrée sur l'erreur d'échantillonnage.

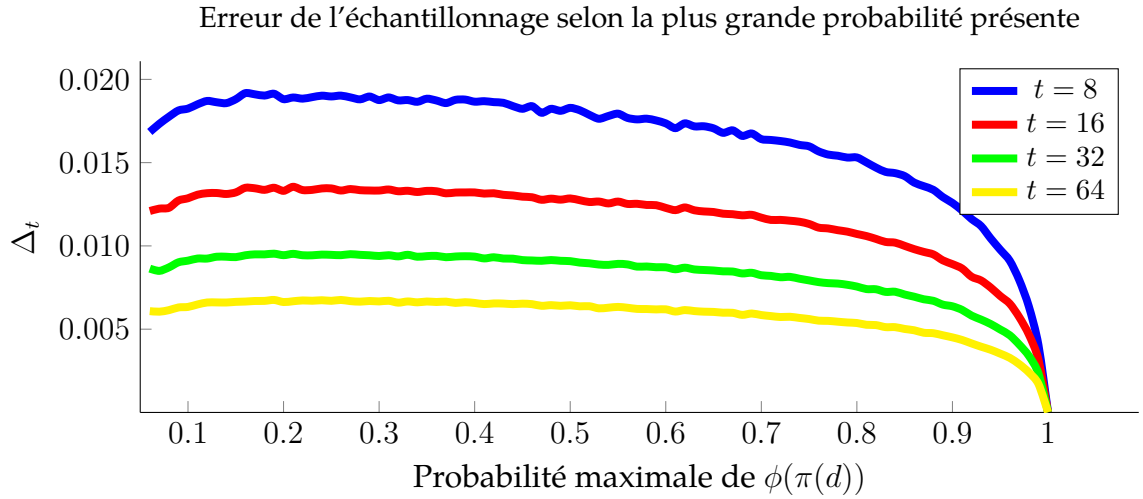

FIGURE 2.2 – Impact de la probabilité maximale de la distribution des résumés sur l'erreur d'échantillonnage.

Une mesure naturelle sur la distribution des résumés est la valeur maximale de la distribution. La figure (2.2) 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 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 que piger 16 échantillons 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).

2.3 BanditSum

Maintenant que l'on sait que l'on peut avoir des approximations très justes pour J - et donc $\nabla_\theta J$, on peut utiliser l'algorithme REINFORCE pour apprendre les paramètres θ d'un système complet. En effet, on peut considérer un réseau de neurones $h_t heta$ prenant en entrée un document et produisant une distribution sur les phrases comme un bandit contextuel, que l'on peut apprendre efficacement avec REINFORCE.

On prend le même réseau que BanditSum. TODO: Figure de l'architecture de BanditSum.

Quelques détails techniques en pratique pour faciliter l'entraînement :

— Ajout d'une baseline pour limiter la variance. La baseline est le R obtenu par le processus de sélection vorace sur π.

— Ajout d'une probabilité ϵ_t de piger aléatoirement au ϕ stochastique utilisé. Ce ϵ_t force le système à explorer. On le fait décroître vers une valeur près de 0 avec le temps pour que le modèle puisse exploiter ses apprentissages quand il devient bon.

2.3.1 Expériences

On roule notre propre implémentation de BanditSum sur le jeu de données. Le pseudocode se trouve plus bas, à la seule exception près qu'on omet le ϵ pour éviter d'alourdir l'algorithme inutilement.

Algorithme 2 Système utilisant un bandit contextuel

Entrée: $\mathcal{D}$ (jeu de données), h_θ (modèle neuronal), ϕ (processus de génération de résumés stochastique), ϕ' (processus de génération de résumés baseline), N (nombre d'échantillons), α (taux d'apprentissage).

tant que vrai **faire**
  $(d, s) \sim \mathcal{D}$ ▷ On pige un document du jeu de données
  $\pi = h_\theta(d)$
  $\mathbf{r}, \mathbf{s} = \text{Echantillonnage}(\pi, \phi, N, s)$
  $s' = \phi'(\pi)$
  $b = R(s', s)$ ▷ Valeur de baseline
  $\theta = \theta + \alpha \frac{1}{N} \sum_{i=1}^{N} (\mathbf{r}_i - b) \nabla_\theta \ln \pi(\mathbf{s}_i)$

Détails techniques : GLoVe taille 100 comme embeddings, params d'optimiseur, GPU et CPU utilisés pour mes expés, decaying de ϵ retenu.

Pour les expériences, on roule BanditSum, en faisant varier N, pour explorer l'importance que peut avoir un meilleur estimé du gradient sur le temps de calcul vs la rapidité de convergence.

2.3.2 Résultats

Rapporter graphe de l'évolution de R en validation selon le nombre de documents vus. Rapporter une courbe pour chaque N parmi $\{8,16,32\}$, qui sont les valeurs les plus pertinentes selon les expés plus haut.

Dans un tableau, rapporter performance en test de Banditsum (nous) vs Banditsum (paper) et SOTA extractif. Rapporter $R-1, R-2, R-L$ et le R utilisé à l'entraînement.

2.4 Conclusion

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

Chapitre 3

Bandit combinatoire

Les deux prochaines approches sont nouvelles. L'idée principale est celle d'utiliser la puissance et rapidité de convergence des approches supervisées en retirant la limite de performance imposée par l'heuristique des génération de cibles binaires.

Dans ce chapitre, on définit comment un bandit combinatoire peut être utilisé pour générer des cibles au cours de l'entraînement. On valide ensuite que cette alternative est viable en effectuant des tests sur le jeu de données de développement. Enfin, on présente comment ces cibles générées par bandit combinatoire peuvent être intégrées dans un système complet de génération de résumés. Le système entier est comparé aux performances du SOTA et l'approche par bandit contextuel.

3.1 Formulation

Un bandit combinatoire peut être vu comme un méta-problème de bandit où l'espace d'actions $\mathcal{A}'$ devient le power set des bras d'un MAB $\mathscr{P}(\mathcal{A})$. Formellement, on considère la possibilité de tirer un *super-bras* $\mathcal{M} = [a_i, ..., a_n]$. Les super-bras ne sont pas nécessairement de taille fixe. Dans cette formulation, une récompense est perçue pour le super-bras tiré. Aucune hypothèse n'est faite sur le nature de la relation entre la récompense associable à chaque bras et la récompense reçue pour le super-bras.

Pour nos besoin, on considère seulement les super-bras composées de 3 phrases. Comme on considère R comme invariante à l'ordre, l'ordre n'est pas important non plus dans les super-bras. On fait une hypothèse sur les récompenses associées à un super-bras : on dit que la récompense du super-bras est la moyenne des récompenses associables à tous les bras.

Commentaire: Il faut que j'y réfléchisse, mais j'ai l'impression que je me perds inutilement dans les formulations. Je présente le bandit contextuel à la manière de Banditsum, avec la portion sampling exclue du problème de bandit, un peu comme si ça faisait partie des dynamiques de l'environnement. Mais quand on y pense, la seule différence avec le bandit com-

binatoire que je présente ici revient essentiellement à dire que j'incorpore le processus de génération de résumés dans le bandit. Mais comme il est fixé à sélectionner 3 phrases et qu'il sert essentiellement juste à donner des estimés de valeur de chaque phrase, je pense que je suis juste en train d'alourdir le contenu pour rien. Je vais réfléchir à ce qui serait une approche plus claire et naturelle et qui expliquerait bien mes 3 chapitres.

3.2 Upper Confidence Bound (UCB)

L'objectif est de générer des cibles : on s'intéresse donc essentiellement à la solution du problème de bandit combinatoire. À cette fin, un algorithme fort utilisé en pratique est UCB (Auer et al., 2002). UCB jongle entre l'exploration d'actions moins visitées et l'exploitation d'actions ayant été payantes en maintenant une borne supérieure sur l'espérance μ_i de chaque bras. Pour un bras i au temps t, la borne supérieure est définie comme

$$\text{UCB}_i(t) = \bar{x_i}(t) + \beta\sqrt{\frac{2\ln t}{n_i(t)}}, \tag{3.1}$$

où $\bar{x_i}(t)$ est la moyenne des récompenses reçues jusqu'au temps t quand l'action i faisait partie des actions du super-bras tiré, β est un hyperparamètre d'exploration, t est le nombre de pas effectués sur le problème de bandit et $n_i(t)$ est le nombre de fois que l'action i a été sélectionnée jusqu'à présent.

Bien que UCB est conçu pour les problèmes de bandit stochastique, il peut être naturellement étendu au problème de bandit combinatoire à taille fixe comme nous avons. En effet, il suffit alors de choisir à chaque ronde le super-bras maximisant la borne supérieure définie par UCB. Dans notre cas, comme on ne tient pas compte de l'ordre dans le super-bras, il suffit donc de sélectionner le super-bras composé des trois actions avec la borne supérieure maximale, i.e.

$$\mathcal{M}_t = \text{3-}\arg\max_{i \in \mathcal{A}} \text{UCB}_i(t).$$

Formellement, on dira qu'on a un ϕ trivial, qui ne fait que sélectionner les phrases de d correspondant aux index de $\mathcal{M}$. Commentaire: C'est pas naturel ça, c'est en train de devenir juste encombrant cette notation là.

Première intuition clé : les réseaux de neurones sont capables de prédire des cibles denses sans problème (TODO: source ou exemple empirique). On n'est donc pas obligés de se limiter à utiliser un algorithme de bandit pour trouver le super-bras optimal et retomber sur les vecteurs binaires utilisés habituellement en supervisé. On peut demander de prédire les $\bar{x_i}(N)$ ou encore la distribution représentée par les $n_i(N)$. On va explorer les deux dans nos expériences.

Deuxième intuition : les $\hat{x}_i$ et $\hat{n}_i$ générés par UCB sont des estimés de la politique optimale. Comme la récompense associée à un bras i au temps t dépend des autres bras présents dans le super-bras $\mathcal{M}_t$, on a que éventuellement UCB identifierait le super-bras optimal et qu'avec un nombre infini de samples, la distribution induite par $\hat{n}_i$ tenderait vers le vecteur binaire représentant le résumé. Similairement, les $\hat{x}_i$ convergeraient éventuellement à R^*, le meilleur score de résumé extractif d'un document, pour les phrases faisant partie du bras optimal. Comment savoir quand on approche de ce stade ? Quand le score du résumé le plus prometteur selon UCB, nommons le R_t, commence à converger vers R^*. On peut donc étudier l'évolution $\Delta_t = R^* - R_t$ pour savoir à quel moment les cibles produites par UCB seraient satisfaisantes.

3.2.1 Expériences

On reprend le jeu de 25 000 documents pour faire des tests. Les résultats des figures **??**, 3.2 et 3.3 sont obtenus en calculant les Δ_t à partir de l'algorithme 3.

Algorithme 3 Génération de la cible pour un document d avec UCB

Entrée: d (document), ϕ (processus de génération de résumés), β (paramètre d'exploration), s (résumé cible), T (nombre de rondes de UCB).

procédure CIBLE_UCB(d, ϕ, β, s, T)
  pour $i = 1, ..., |d|$ **faire**
    $\bar{x}_i = 0$
    $n_i = 0$
  pour $t = 1, ..., T$ **faire**
    pour $i = 1, ..., |d|$ **faire**
      $\text{UCB}_i = \bar{x}_i + \beta\sqrt{\frac{2 \ln t}{n_i}}$ ▷ Si $n_i = 0$, on pose UCB_i à une grande constante.
    $\mathcal{M} = 3\text{-arg max UCB}_i$
    $\hat{s} = \phi(\mathcal{M})$
    $r_t = R(\hat{s}, s)$
    pour tout $i \in \mathcal{M}$ **faire**
      $\bar{x}_i = r + \frac{n_i}{n_i+1}\bar{x}_i$ ▷ Mise à jour incrémentale de la moyenne
      $n_i = n_i + 1$
  retourner $\{\bar{x}_i\}_{i=1}^{|d|}, \{n_i\}_{i=1}^{|d|}$

TODO: Les figures ont été générées avec un $3t$ au lieu de t dans UCB. Je dois les rerouler (devrait pas changer grand chose). Je dois aussi rajouter les intervalles de confiance.

3.2.2 Résultats

La figure **??** présente l'importance du critère de sélection utilisé pour trouver le résumé optimal selon UCB au temps t. Deux approches envisageables : considérer les phrases i où la moyenne des récompenses reçues $\bar{x}_i$ est maximale ou encore celles où le nombre de sélections n_i est maximal. Les résultats rapportés donnent un net avantage aux résumés sélectionnés se-

lon la moyenne des récompenses perçues. C'est normal : les n_i prennent beaucoup plus de temps à distinguer les bonnes phrases car ils sont issus d'une distribution uniforme pour la sélection des $|d|$ premières phrases. Remarquons aussi que les deux approches convergent éventuellement au même résumé optimal, comme c'était à prévoir. Enfin, la conclusion à tirer est que les $\bar{x}_i$ forment de meilleurs cibles que les n_i.

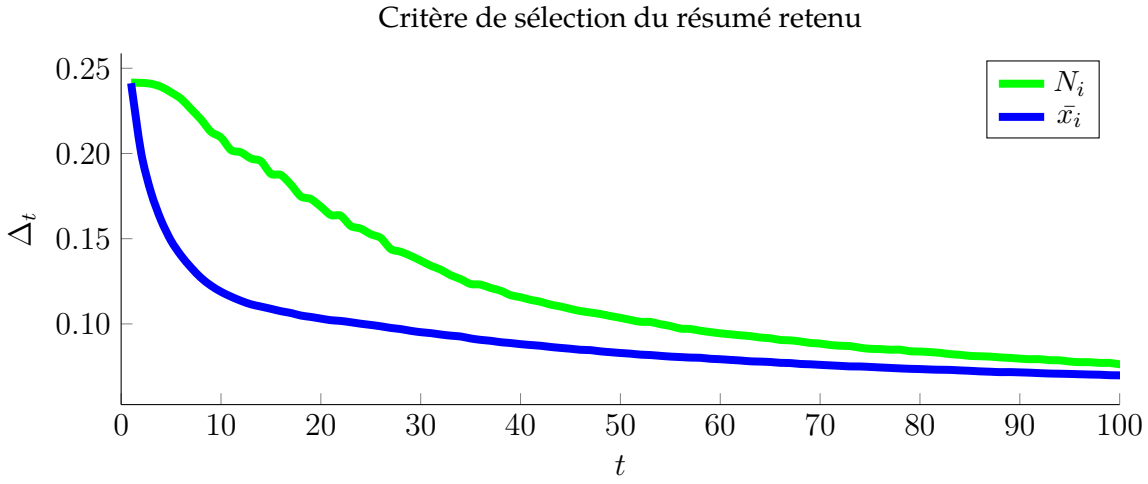

FIGURE 3.1 – Impact du critère de sélection utilisé pour choisir le résumé retenu au temps t.

La figure 3.2 présente comment le paramètre β influe sur la convergence de UCB. On remarque que $\beta = 10$ semble être le bon compromis entre exploration et exploitation, réussissant à converger à $\Delta_t \approx 0.05$ après 200 rondes de UCB.

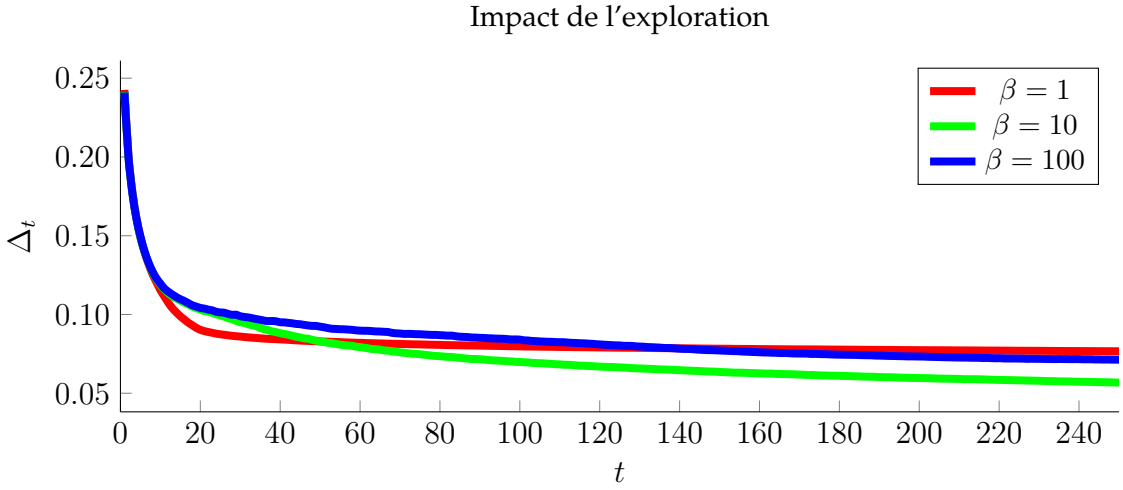

FIGURE 3.2 – Impact du paramètre d'exploration β utilisé par UCB sur la convergence.

La figure 3.3 évalue dans quelle mesure la taille des documents influe sur la qualité du pro-

cessus de génération de cible présenté. On remarque d'abord que $\beta = 10$ est la meilleure configuration pour toutes les tailles de document. Il est donc adéquat de prendre la même valeur de β pour tous les documents. Aussi, on remarque que l'apprentissage stagne plus tôt pour les documents qui sont plus courts. Il serait donc pertinent de faire croître le nombre de rondes de UCB en fonction du nombre de phrases dans un document. Un bon point de départ serait de prendre $t(|d|) = 4|d| + 50$. Cette fonction assure un minimum de 50 rondes et va augmenter de manière linéaire le nombre de rondes jusqu'à 250 pour les documents de 50 phrases, la taille maximale allouée par notre régularisation.

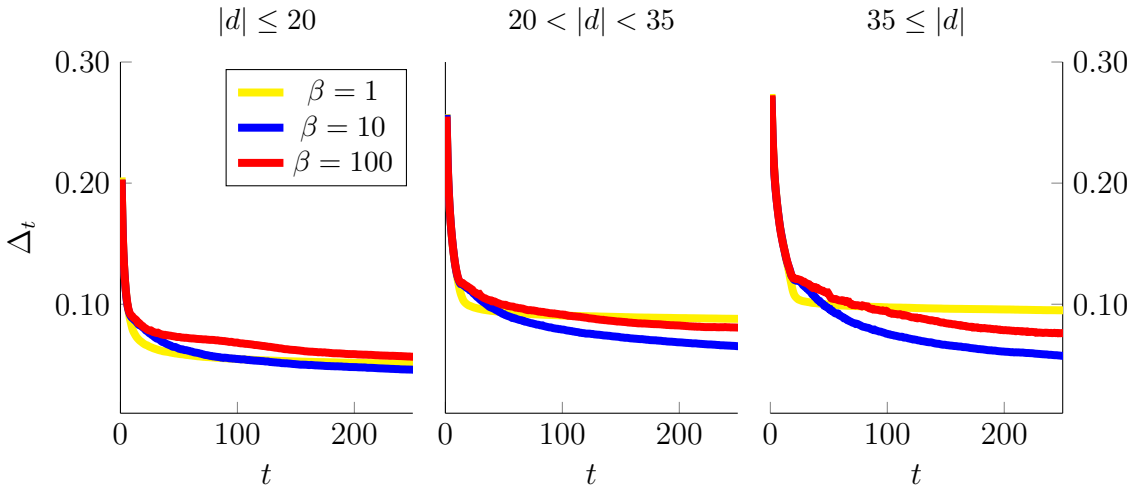

FIGURE 3.3 – Impact de la taille du document sur la convergence.

3.3 UCB avec connaissances a priori

Si on se fie exclusivement à la règle UCB déterminée plus haut, on pourrait simplement utiliser UCB comme heuristique pour générer des cibles fixes pour l'apprentissage supervisé, comme c'est habituellement fait pour les vecteurs binaires. Or, il est intéressant de considérer la possibilité d'introduire une connaissance a priori dans le calcul de UCB. En effet, si on possède une certaine information sur la distribution optimale d'un document, on peut l'utiliser pour accélérer la convergence de l'algorithme en privilégiant certaines actions. Une manière assez directe d'introduire une connaissance a prior P_i sur une action i dans le calcul de UCB est de prendre

$$\text{UCB}_i(t) = \bar{x}_i(t) + \beta P_i \sqrt{\frac{2 \ln t}{n_i(t)}}. \tag{3.2}$$

Il est alors possible de modifier naturellement l'algorithme 3 pour y incorporer la nouvelle règle incorporant les connaissances a priori.

3.3.1 Résultats

Les expériences ont été reprises dans le même cadre qu'à la section 3.2.2. Commentaire: Les graphes ici sont à retravailler. Je veux prouver la propriété d'amélioration de connaissances a prior que je suppose du processus UCB. Je devrais donc établir que la convergence est (1) plus rapide avec un prior informatif qu'avec le prior uniforme et (2) définir à quel point un prior est trop contraignant pour être utilisé (i.e. quand il est trop près d'être greedy et bloque l'exploration).

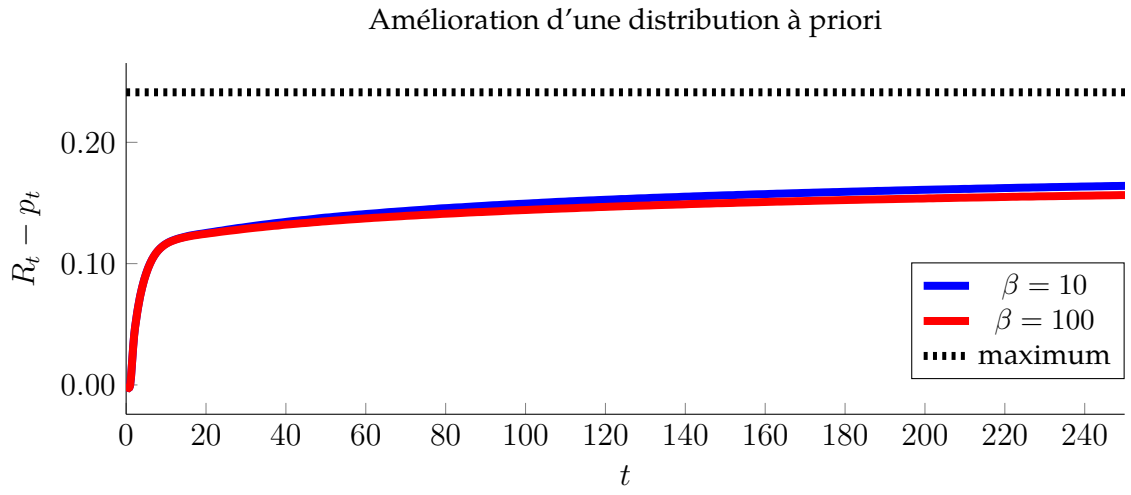

FIGURE 3.4 – Évolution de l'amélioration de distributions à priori selon le nombre t de rondes effectuées.

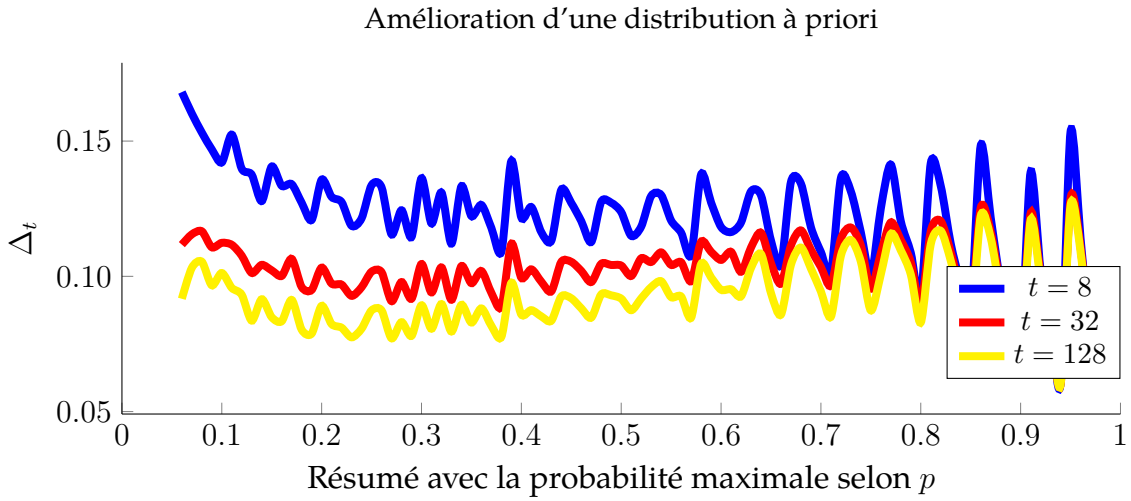

FIGURE 3.5 – Évolution de l'amélioration de distributions à priori selon le nombre t de rondes effectuées.

3.4 CombiSum

Dans un système complet, on veut piger une batch de documents, obtenir leurs cibles $\bar{\mathbf{x}}_\mathbf{d}$ et apprendre une réseau de neurones qui prédit la valeur associable à chacune des phrases. Le réseau est donc apprenable avec une perte MSE standard sur les cibles générées par UCB.

Le plan est d'utiliser les valeurs prédites pour générer des priors qui aideront à accélérer le processus UCB. Comme je soupçonne que les priors devront être relativement près de la distribution uniforme, je ne fais pas de softmax, je prends juste la fraction des valeurs prédites comme prior. Probablement possible d'explorer plutôt un softmax avec température qui décroît au courant de l'apprentissage.

Aucune modification à l'architecture de BanditSum : on garde des sigmoid comme sortie pour le MLP passé sur chaque sentence embedding.

3.4.1 Expériences

On roule le système sur le jeu de données CNN/DailyMail. Le pseudocode décrivant le système se trouve dans l'algorithme 4.

Algorithme 4 Système utilisant un bandit combinatoire

Entrée: $\mathcal{D}$ (jeu de données), h_θ (modèle neuronal), T (nombre de rondes UCB), α (taux d'apprentissage), β (taux d'exploration UCB), B (taille de minibatch).

tant que vrai **faire**
  batch $\sim \mathcal{D}^B$ ▷ On pige la minibatch du jeu de données
  $\nabla = \mathbf{0}$
  pour tout $(d,s) \in$ batch **faire**
    $\hat{\mathbf{x}} = h_\theta(d)$
    pour $i = 1,...,|d|$ **faire**
      $\mathbf{P}_i = \frac{\hat{\mathbf{x}}_\mathbf{i}}{\|\hat{\mathbf{x}}\|}$
    $\mathbf{x} =$ Cible_UCB(d, ϕ, β, s, T, $\mathbf{P}$) ▷ On passe le prior à Cible_UCB
    $\nabla = \nabla + \nabla_\theta \|\mathbf{x} - \hat{\mathbf{x}}\|^2$
  $\theta = \theta - \alpha\nabla$

Pour les expériences, on fait une gridsearch sur le fait d'utiliser un nombre fixe $T = 150$ et le T linéaire selon le nombre de phrases et sur l'utilisation ou non de priors. Ça nous fait 4 configurations au total

3.4.2 Résultats

Rapporter graphe de l'évolution de R en validation selon le nombre de documents vus. Rapporter une courbe pour chacune des configurations.

Dans un tableau, rapporter performance en test de Combisum vs Banditsum (nous) et SOTA extractif. Rapporter $R-1, R-2, R-L$ et le R utilisé à l'entraînement.

3.5 Conclusion

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

Chapitre 4

Bandit combinatoire linéaire

Un des défauts de la formulation précédente est que les récompenses perçues pour une phrase ne sont pas propagées aux autres phrases et il faut donc minimalement explorer chacune des phrases quelques fois avant de déceler la moindre tendance. Il est possible de faire cela sous la formulation du bandit combinatoire linéaire, où on possède des représentations vectorielles de chacune des actions et on fait l'hypothèse qu'il existe un vecteur propre au problème qui relie les représentations de phrases à leur valeur en espérance.

Dans ce chapitre, on définit comment un bandit combinatoire linéaire peut être utilisé pour générer des cibles au cours de l'entraînement et éventuellement être plus efficace que sa version sans représentations vectorielles. On valide ensuite que cette alternative est viable en effectuant des tests sur le jeu de données de développement. Enfin, on présente comment ces cibles générées par bandit combinatoire linéaire peuvent être intégrées dans un système complet de génération de résumés. Le système entier est comparé aux performances du SOTA et les approches par bandit contextuel et combinatoire présentées précédemment.

4.1 Formulation

La formulation en bandit combinatoire linéaire reprend exactement le même cadre formel que la bandit combinatoire vu à la section 3.1. La seule nouvelle notion est la présence de représentations vectorielles $\tilde{a}_i \in \mathbb{R}^n$ pour les actions du problème et l'introduction de l'hypothèse linéaire. Cette dernière mentionne qu'il existe un vecteur unique ω_* tel que $\langle \omega_*^\intercal, a_i \rangle \approx \mu_i$ pour tout i.

Dans notre cas, plusieurs choix sont possibles pour le choix des représentations vectorielles de chaque phrase. Nous faisons le choix de prendre le plongement de phrase généré par le LSTM au niveau des phrases dans le réseau de neurones de génération de résumés habituel. La justification de ce choix deviendra claire lorsque l'on présentera le système complet utilisant le bandit combinatoire linéaire à la section 4.3.

4.2 Linear Upper Confidence Bound (LinUCB)

Pour la version linéaire du problème MAB, l'algorithme LinUCB (Chu et al., 2011) représente l'état de l'art. Cet algorithme est basé sur le même principe d'optimisme face à l'incertain que sa version sans représentations d'action, UCB. Ici, au lieu de maintenir une borne supérieure sur les valeurs possibles de μ_i, LinUCB maintient un ellipsoïde de confiance autour de son estimé $\hat{\omega}_t$. Cet ellipsoïde exploite encore le principe d'optimisme face à l'incertain car il est conçu pour avoir une forte probabilité de contenir le vrai ω_*.

Pour bâtir l'ellipsoïde à chaque ronde, LinUCB utilise d'abord les représentations des actions choisies $\tilde{a}_{i_t}$ et la récompense X_t associée pour fixer le centre à la meilleure estimation possible. L'ellipsoïde est simplement bâtie en résolvant le problème de moindres carrés

$$\omega_t = \underset{\omega \in \mathbb{R}^n}{\arg\min} \sum_{n=1}^{t-1} \left(\langle \omega^\intercal, a_{i_t} \rangle - X_t \right)^2 . \tag{4.1}$$

Comme le problème n'admet pas nécessairement de solution exacte, LinUCB emploie la régularisation de Tikhonov (Tikhonov, 1963) avec $\lambda = 1$. Le centre ω_t est alors choisi selon la solution analytique connue du problème

$$\omega_t = (\mathbf{A}^\intercal \mathbf{A} + \lambda \mathbf{I})^{-1} \mathbf{A}^\intercal \mathbf{X}, \tag{4.2}$$

où $\mathbf{A}$ est la matrice dont les lignes sont les actions $\tilde{a}_{i_t}$ sélectionnées, $\mathbf{I}$ est la matrice identité et $\mathbf{X}$ est le vecteur composé des récompenses X_t.

Enfin, à partir du centre ω_t calculé selon (4.2) et de la matrice $\mathbf{A}$, on se retrouve avec le borne supérieure de chaque action i

$$\text{UCB}_i = \langle \omega_t^\intercal, a_i \rangle + \beta \sqrt{a_i^\intercal \mathbf{A}^{-1} a_i}, \tag{4.3}$$

où β représente encore un hyperparamètre balançant l'exploration. Notons la différence avec UCB pour gérer l'exploration : celle-ci n'est plus gérée selon les nombre de visites à un bras mais bien en fonction de la similarité entre le bras i et les bras tentés dans le passé.

Application au contexte combinatoire et à la génération de résumés : on a pour chaque phrase une représentation $\tilde{a}_i$ produite par notre modèle. Toutes les représentations sont produites par le LSTM qui travaille au niveau des phrases. On considère ainsi la représentation vectorielle d'un super-bras $\mathcal{M}$ comme étant

$$\tilde{\mathcal{M}} = \sum_{i \in \mathcal{M}} \tilde{a}_i. \tag{4.4}$$

Encore une fois, comme on ne tient pas compte de l'ordre dans le super-bras, on peut sélectionner le super-bras optimal en sélectionnant simplement les 3 phrases dont la borne supérieure UCB_i est maximale.

Enfin, on utilise LinUCB pour générer une cible encore une fois. Cette fois-ci, on s'intéresse à prédire ω_T, le meilleur estimé pour ω_*.

Note : on peut aussi considérer les connaissances a priori pour cette formulation. TODO: Il faudra faire les expériences pour les priors aussi.

4.2.1 Expériences

On reprend le jeu de 25 000 documents pour faire des tests. On pose encore une fois $\Delta_t = R^* - R_t$, la différence entre le résumé optimal d'un document et le résumé suggéré par LinUCB au temps t. Les résultats des figures 4.1, 4.2 et 4.3 sont obtenus en calculant les Δ_t à partir de l'algorithme 5. Le pré-entraînement fait pour obtenir des bonnes représentations de phrases $\tilde{a}_i$ est une tâche de génération de prédiction de scores R où les représentations de 3 phrases d'un résumé sont fournies à un réseau de neurones à une seule couche qui doit prédire les 3 métriques ROUGE associées au résumé.

Algorithme 5 Génération de la cible pour un document d avec LinUCB

Entrée: d (document), ϕ (processus de génération de résumés), β (paramètre d'exploration), s (résumé cible), T (nombre de rondes de UCB), $\tilde{a}$ (matrice des représentations d'actions).

procédure CIBLE_LINUCB(d, β, s, T, $\tilde{a}$)
  $A = \mathbf{I}$
  $b = \mathbf{0}$
  pour $t = 1, ..., T$ **faire**
    $\omega_t = A^{-1}b$
    pour $i = 1,..., |d|$ **faire**
      $\text{UCB}_i = \langle \omega_t^\intercal, \tilde{a}_i \rangle + \beta\sqrt{\tilde{a}_i^\intercal \mathbf{A}^{-1} \tilde{a}_i}$
    $\mathcal{M} = 3\text{-arg max UCB}_i$
    $\hat{s} = \phi(\mathcal{M})$
    $X_t = R(\hat{s}, s)$
    pour tout $i \in \mathcal{M}$ **faire**
      $A = A + \tilde{a}_i \tilde{a}_i^\intercal$
      $b = b + X_t \tilde{a}_i$
retourner ω_T

Commentaire: Dans mon code j'utilise plusieurs théorèmes, notamment sur l'inversion de matrices qui sont des sommes. Je ne crois pas que c'est nécessaire d'en parler ici, comme ce sont seulement des règles qui permettent d'alléger le coût computationnel.

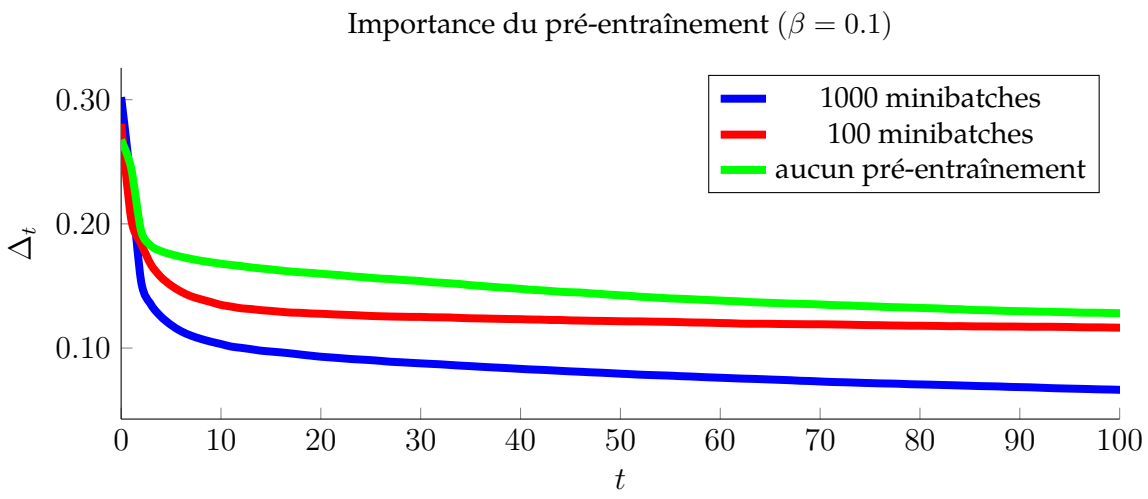

FIGURE 4.1 – Impact du pré-entraînement utilisé sur la convergence de LinUCB.

4.2.2 Résultats

Commentaire: Les résultats présentés sont à corriger. Ils ne sont pas issus de LinUCB, c'est LinUCT, qui incorporait une structure d'arbre dans la mise à jour après chaque récompense perçue. Aussi, j'ai décidé de ne plus incorporer d'embeddings du document, commme ça se justifiait difficilement théoriquement avec le cadre LinUCB.

Commentaire: Je skip donc pas mal les commentaires ici car l'analyse sera probablement légèrement modifiée.

La figure 4.1 illustre que le pré-entraînement est essentiel pour obtenir des performances de bonne qualité avec LinUCB. En effet, le processus converge à de meilleures cibles avec un pré-entraînement de 1000 minibatches et il le fait plus rapidement.

4.3 LinCombiSum

À l'inférence, on prédit le super-bras pour lequel on a le produit maximal entre le ω prédit et les représentations des 3 bras. Le LinUCB génère des targets ω qu'on apprend par perte cosine. On prend la perte cosine pcq on ne s'intéresse pas à l'amplitude du vecteur. On s'en sert essentiellement comme discriminant pour déterminer quelles phrases choisir.

On doit modifier légèrement le réseau : au lieu d'avoir un MLP qui prend en entrée les sentence embeddings et traduit ça en valeur estimée, on doit avoir une tête qui prédit un ω. Deux têtes considérées :(1) un MLP qui prend en entrée le hidden state final du LSTM au niveau des phrases et sort ω. (2) Un autre LSTM qui prend en entrée toutes les représentations de phrase de retourne θ.

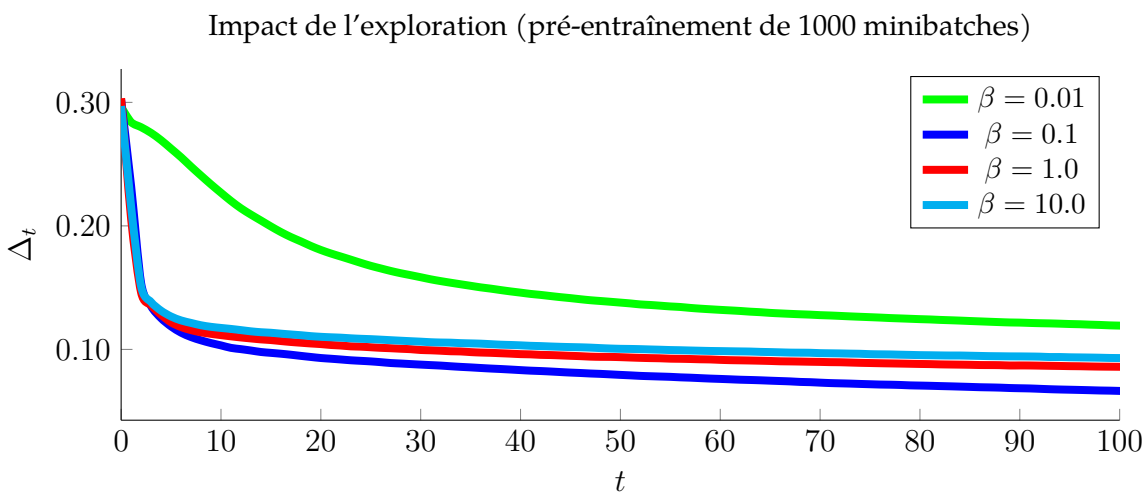

FIGURE 4.2 – Impact de l'hyperparamètre β sur la convergence de LinUCB

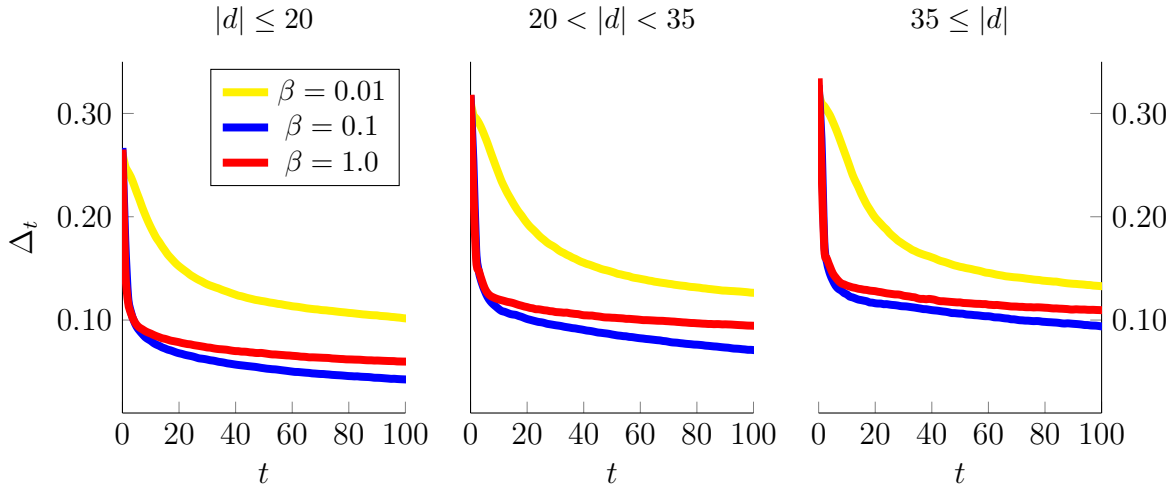

FIGURE 4.3 – Impact de la taille du document sur la convergence.

On peut encore produire les priors à partir du ω inféré et des représentations de phrases, en prenant encore une fois la fraction des scores prédits pour chaque phrase comme sa proba.

4.3.1 Expériences

Pour les expériences, on explore encore une fois un nombre T de ronde fixé et un qui scale linéairement avec le nombre de phrases. On explore aussi l'impact de la présence et de l'absence de priors.

On roule le système sur le jeu de données CNN/DailyMail. Le pseudocode décrivant le système se trouve dans l'algorithme 4.

Algorithme 6 Système utilisant un bandit combinatoire linéaire

Entrée: $\mathcal{D}$ (jeu de données), h_θ (modèle neuronal), T (nombre de rondes UCB), α (taux d'apprentissage), β (taux d'exploration LinUCB), B (taille de minibatch).

tant que vrai **faire**
  batch $\sim \mathcal{D}^B$ ▷ On pige la minibatch du jeu de données
  $\nabla = \mathbf{0}$
  pour tout $(d,s) \in$ batch **faire**
    $\hat{\omega}, \tilde{a} = h_\theta(d)$
    pour $i = 1,...,|d|$ **faire**
      $\mathbf{P}_i = \frac{\hat{\omega}\tilde{a}_i}{\|\hat{\omega}^\intercal \tilde{a}\|}$
    $\omega =$ Cible_LinUCB(d, β, s, T, $\tilde{a}$, $\mathbf{P}$) ▷ On passe le prior à Cible_UCB
    $\nabla = \nabla + \nabla_\theta \frac{\omega^\intercal \hat{\omega}}{\|\omega\|\|\hat{\omega}\|}$
  $\theta = \theta - \alpha\nabla$

4.3.2 Résultats

TODO:

4.4 Conclusion

TODO:

Conclusion

Synthèse

TODO: Comparaison des méthodes entre elles et avec l'état de l'art. Les angles à aborder sont la perfomance R, la vitesse en temps de calcul ainsi que la vitesse en nombre de documents.

Travaux futurs

On résout seulement une instance très limitée de bandit combinatoire : celle où la taille est fixée. On pourrait étudier comment des formulations plus souples comme (Luo et al., 2019) seraient envisageables avec les formulations par bandit explorées dans les deux derniers chapitres. Autre point : on n'a utilisé aucun algo de bandit combinatoire dédié, on a juste utilisé une généralisation des algos de MAB habituels pour les transformer en mode combinatoire comme on avait une formulation facile. Ce serait intéressant de voir la différence de performance quand on utilise les algorithmes dédiés spécifiquement à cette tâche.

Bibliographie

P. Auer, N. Cesa-Bianchi, and P. Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine learning*, 47(2-3) :235–256, 2002.

W. Chu, L. Li, L. Reyzin, and R. Schapire. Contextual bandits with linear payoff functions. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, pages 208–214, 2011.

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.

F. Gers. Learning to forget : continual prediction with lstm. *IET Conference Proceedings*, pages 850–855(5), January 1999. URL https://digital-library.theiet.org/content/conferences/10.1049/cp_19991218.

V. Kuleshov and D. Precup. Algorithms for multi-armed bandit problems, 2014.

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.

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.

J.-P. Ng and V. Abrecht. Better summarization evaluation with word embeddings for ROUGE. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1925–1930, Lisbon, Portugal, Sept. 2015. Association for Computational Linguistics. doi : 10.18653/v1/D15-1222. URL https://www.aclweb.org/anthology/D15-1222.

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.

M. Peyrard. Studying summarization evaluation metrics in the appropriate scoring range. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 5093–5100, Florence, Italy, July 2019. Association for Computational Linguistics. doi : 10.18653/v1/P19-1502. URL https://www.aclweb.org/anthology/P19-1502.

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.

W. Shen, J. Wang, Y.-G. Jiang, and H. Zha. Portfolio choices with orthogonal bandit learning. In *Proceedings of the 24th International Conference on Artificial Intelligence*, IJCAI'15, page 974–980. AAAI Press, 2015. ISBN 9781577357384.

R. S. Sutton, D. McAllester, S. Singh, and Y. Mansour. Policy gradient methods for reinforcement learning with function approximation. *Advances in neural information processing systems*, 12 :1057–1063, 1999.

A. N. Tikhonov. On the solution of ill-posed problems and the method of regularization. In *Doklady Akademii Nauk*, volume 151, pages 501–504. Russian Academy of Sciences, 1963.

R. J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4) :229–256, 1992.

B. Xu, N. Wang, T. Chen, and M. Li. Empirical Evaluation of Rectified Activations in Convolutional Network, Nov. 2015. URL http://arxiv.org/abs/1505.00853.

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.