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.>

Introduction

Lors d'une entrevue en 2018, le président de la multinationale Siemens mentionnait que "Data is the oil, some say the gold, of the 21st century — the raw material that our economies, societies and democracies are increasingly being built on". Son affirmation, partagée par bon nombre d'experts dans le domaine de la technologie, décrit ce que l'on appelle communément l'ère du *Big Data*. Cette appellation vise à décrire la quantité immense de données maintenant générées et collectées par le traffic internet ou encore les appareils de l'Internet des ObjeTs (IoT), pour n'en nommer que quelques-uns.

Dans la panoplie de données à disposition se trouvent les données textuelles, issues par exemple d'interactions sur des réseaux sociaux ou encore de documents numériques. Ces données textuelles présentent une opportunité d'affaires en or à qui saura les utiliser. En effet, des traitements requérant habituellement l'avis d'un expert humain peuvent se voir automatiser par des modèles entraînés à reproduire le comportement des experts sur les données disponibles. Il suffit de penser aux cas des agents conversationnels pour le service à la clientèle ou des détecteurs automatiques de contenus toxiques sur les sites webs pour réaliser l'immense potentiel apporté par l'abondance des données textuelles.

Ce ne sont toutefois pas tous les domaines utilisant des données textuelles qui peuvent être automatisables en entièreté. Certains domaines, notamment en assurance, requièrent explicitement l'intervention d'un humain pour des raisons de législation. Comment alors faire profiter du grand afflux de données dans ces 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. Ces systèmes sont le sujet principal du présent document.

Plus particulièrement, on s'intéresse à une nouvelle approche basée sur une formulation de la génération de résumé comme un problème de bandit. Les problèmes de bandit sont des problèmes étudiés depuis le début du 20ème siècle, dont la résolution est basée sur l'obtention rapide d'une bonne solution. En voyant le génération de résumé comme un bandit, on sou-

haite bâtir des systèmes de génération de résumés qui apprendront rapidement à générer des résumés satisfaisants grâce à une exploration efficace du vaste espace de résumés possibles.

Objectifs

Concrètement, ce document a pour objectifs de répondre aux questions suivantes :

— Comment peut-on formuler la génération de résumés comme un problème de bandit multi-bras (MAB) ? Comment les algorithmes utilisés habituellement pour résoudre les MAB peuvent alors être appliqués à la génération de résumés ?

— 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 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-elles entre eux ? Et par rapport à l'état de l'art ?

On atteint les 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 débute par une introduction aux concepts de base essentiels à la compréhension du reste du document. Le document est 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 commence par définir le cadre bandit proposé. Des expériences empiriques sont effectuées pour justifier l'applicabilité du cadre au contexte de génération de résumés. On montre ensuite comment le cadre bandit peut utilisé dans un nouvel algorithme pour l'entraînement d'un système neuronal de génération automatique de résumés. Enfin, on valide le comportement de l'algorithme en l'utilisant pour entraîner plusieurs modèles de génération de résumés. Les algorithmes sont comparés en fonction de leur rapidité de convergence et de la qualité finale des modèles produits.

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 par bandits

On définit un bandit (Lattimore and Szepesvári, 2020) comme étant un apprenant qui interagit avec un environnement séquentiel. À chaque pas de temps, le bandit sélectionne une action à effectuer et reçoit une récompense de l'environnement associée à l'action. L'objectif du bandit est maximiser les récompenses obtenues sur un horizon fini n d'interactions avec l'environnement. Notons que cet objectif est différent des approches d'apprentissage automatique supervisées : au lieu de s'intéresser seulement à la performance finale de l'apprenant, on s'intéresse aussi à ce que sa performance s'améliore rapidement. En pratique, l'objectif bandit est le plus approprié 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), de la gestion de portfolios financiers (Shen et al., 2015) et de la suggestion personnalisée d'articles de journaux (Li et al., 2010).

1.1.1 Bandit multi-bras stochastique

Un bandit multi-bras stochastique (MAB) est un 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. Dans le contexte stochastique, 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 après T interactions :

$$\rho_T = T\mu^* - \sum_{t=1}^{T} X_t. \tag{1.1}$$

Ici, ρ_T représente donc la différence entre les récompenses obtenues en sélectionnant l'action optimale et les actions sélectionnées par l'apprenant. Il s'agit donc d'une mesure de la sous-optimalité de l'apprenant. En minimisant le regret cumulatif, un apprenant maximise donc les récompenses perçues et satisfait l'objectif bandit.

L'objectif de minimisation du regret cumulatif fait intervenir un principe fondamental dans la littérature bandit : le dilemme exploration-exploitation. En effet, afin de minimiser le regret, un bandit doit limiter la sous-optimalité de la récompense associée à l'action qu'il choisit en s'assurant qu'elle exploite les informations perçues par le passé mais explore aussi suffisamment les actions susceptibles d'être meilleures.

1.2 Réseaux de neurones

TODO: Cette section va être à retoucher selon les commentaires de Luc. Pas encore modifié car pas prioritaire pour le moment.

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'activations : ReLU (Xu et al., 2015) et sigmoïde.

— Les neurones sont regroupés 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). Présenter RNN (Schuster and Paliwal, 1997), puis 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. Des approches existent spécifiquement pour la génération de résumés à partir d'un ensemble de documents en entrée, mais nous nous contentons de dire que le cas multi-documents se ramène grossièrement au cas de la génération d'un seul document en considérant la concaténation de l'ensemble de documents.

1.3.1 Formulation extractive

On sélectionne des phrases du document initial.

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

Les approches de l'état de l'art sont toutes basées sur l'obtention d'une représentation vectorielle de chaque phrase. Ces représentations sont ensuite utilisées pour prédire un score associé à chaque phrase. Ces scores produisent une distribution sur les phrases, à partir de laquelle le résumé est bâti.

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. On définit ψ comme étant le processus vorace et ξ le processus stochastique.

Enfin, pour un problème de génération de résumés donnés, on fixe habituellement ϕ. 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.

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 $J(\theta)$

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

basée sur un échantillon de J. 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. Deux LSTMs consécutifs sont utilisés pour générer les représentations des phrases, un travaillant au niveau des mots et l'autre au niveau des phrases. Les scores sont obtenus via une couche pleinement connectée avec sortie réelle, partagée pour toutes les phrases. 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., 2020) 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 remplacement 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 près 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. En vue des statistiques 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) :

$$R(\hat{s}, s) := \frac{1}{3}\left[\text{ROUGE-1}(\hat{s}, s) + \text{ROUGE-2}(\hat{s}, s) + \text{ROUGE-L}(\hat{s}, s)\right]. \tag{1.4}$$

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 l'approche multitâche.

Chapitre 2

Bandit contextuel

La première approche explorée est celle qui formule la génération de résumés comme un bandit contextuel, telle qu'initialement proposée par (Dong et al., 2018). Dans ce chapitre, on présente la formulation en bandit contextuel, on présente comment elle s'applique à la génération de résumés et on valide son applicabilité sur un jeu de données de développement. On montre 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 contextuelle

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 $\mathcal{A}$ 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 $a \in \mathcal{A}$ en insérant les phrases dans l'ordre dans lequel elles apparaissent dans le document original.

2.2 Approximation de la performance par échantillonnage

Selon le formalisme de génération de résumés défini à le section 1.3.1, on peut donc dire que l'on a un bandit π qui, pour un document d en entrée, retourne une distribution $\pi(d) \in [0,1]^{|d|}$. 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). Rappelons ici que $J(\theta)$ représente la récompense espérée en pigeant un document d et en utilisant π_θ pour générer son résumé. En générant les résumés de manière stochastique, on permet à J de tenir compte de tous les résumés possibles de d pondérés par leur probabilité selon $\pi_\theta(d)$, permettant de distinguer des distributions qui accordent une plus grande probabilité aux meilleurs résumés. On emploie donc le processus ξ de génération de résumé stochastique qui pige sans répétition 3 phrases selon $\pi_\theta(d)$.

En pratique, les approximations de $\nabla J(\theta)$ générées en utilisant un seul échantillon de J sont très instables pour un processus stochastique ψ et rendent l'apprentissage difficile. Une solution simple et peu coûteuse pour stabiliser l'entraînement est de piger N résumés $\hat{s}_t$ en fonction de $\xi(\pi)$ pour bâtir un meilleur estimateur selon

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

2.2.1 Expériences

On s'intéresse à savoir combien d'échantillons sont requis pour obtenir une représentation adéquate de la valeur de $\nabla J(\theta)$. On note d'abord que, dans l'équation (2.1), la dérivée de $\xi(\pi)$ est déterministe et n'est pas sensible à la portion stochastique du processus de génération de résumés. Pour mesurer la qualité de l'approximation, il faut donc seulement s'intéresser à la différence entre la véritable valeur de $J(\theta)$ et son estimation basée sur des échantillons selon $\xi(\pi)$. Enfin, comme la pige d'un document d dans le calcul de J est uniforme, on peut simplement s'intéresser à l'estimation de J sur un seul document à la fois.

On prend donc J comme étant l'espérance de récompense de $\xi(\pi)$ sur une paire document résumé (d, s) quelconque et 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 $\bar{J_N}$, 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 l'approximation $\bar{J_N}$ se trouve dans l'algorithme 1.

Algorithme 1 Calcul de $\bar{J_N}$

Entrée: $\pi(d)$ (distribution de phrases), N (nombre d'échantillons), s (résumé cible).

1: **pour** $t = 1, ..., N$ **faire**
2: Piger $\hat{s_t} \sim \xi\left(\pi(d)\right)$
3: $\hat{J_N} = \frac{1}{N} \sum_{t=1}^{N} R(\hat{s_t}, s)$

Pré-calcul des scores R

Les méthodes présentées dans ce mémoire utilisent toutes un grand nombre de scores R dans leur procédure d'entraînement, dont le calcul est plutôt lent. Or, pour un document d donné, on sait que tous les résumés que l'on doit considérer sont l'ensemble des combinaisons des phrases ordonnées selon leur position dans le document initial. Il est donc possible de calculer le score $R(\hat{s}, s)$ de tous les résumés $\hat{s}$ dans un premier temps et de simplement aller lire la valeur dans un tableau correspondant lorsque nécessaire. On a donc, dans un premier temps, précalculé tous les scores R associés à tous les résumés pour chaque document du jeu de données.

Méthodologie

On valide la rapidité de la convergence de $\bar{J}$ vers J en les comparant directement sur un sous-ensemble de 25 000 documents d'entraînement du jeu de données CNN/DailyMail. Ce jeu de développement, qui sera réutilisé pour les chapitres suivants, se veut aussi varié que possible. Notamment, il contient 8 674 documents de 20 phrases ou moins, 10 490 documents contenant entre 20 et 35 phrases exclusivement et 5 796 documents de 35 phrases ou plus.

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 vorace $G(d)$:

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

La distribution vorace employée $G(d)$ représente un vecteur de taille $|d|$ où trois indices pigés aléatoirement ont la valeur de $1/3$ et les autres indices sont nuls. On explore τ variant de 0 à 1 en incréments de 0.01 et en repigeant les indices non-nuls de $G(d)$ à chaque fois.

Les expériences consistent à calculer la valeur de Δ_N pour différentes valeurs de N allant jusqu'à 100 sur tous les documents du jeu de développement. L'approximation $\hat{J_N}$ est obtenue en suivant l'algorithme 1. On calcule analytiquement la véritable valeur de J grâce au fait que l'on possède tous les scores R de tous les résumés possibles pour les documents du jeu de données.

2.2.2 Résultats

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.

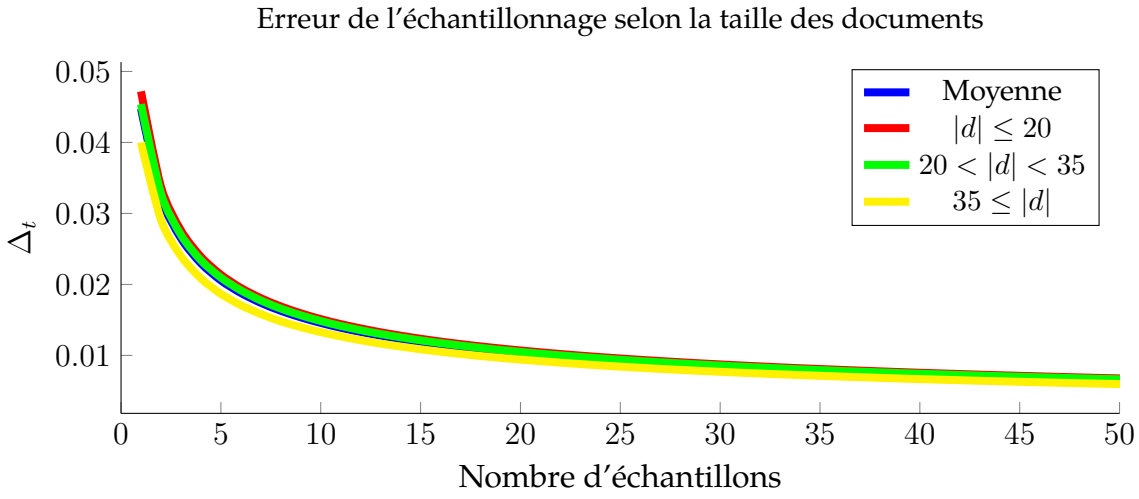

FIGURE 2.1 – Impact de la taille du document en entrée sur l'erreur d'échantillonnage.

TODO: Préciser nb de documents dans chaque bracket de nombre de phrases.

Une mesure naturelle sur la distribution des résumés est la valeur maximale de la distribution des résumés. La figure 2.2 nous montre une courbe à la tendance logarithmique, où plus la probabilité maximale 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 0.01 R est définitivement suffisante pour justifier l'utilisation de $\hat{J}_N$ au lieu de J. Aussi, les résultats indiquent que la rapidité de la convergence ralentit progressivement autour de $N = 16$. Comme il est plus coûteux en temps de calcul de faire plus d'échantillons, il apparaît naturel de considérer $N = 16$ comme un excellent compromis entre le temps de calcul et la rapidité de l'estimation. Notons que les articles de (Dong et al., 2018) et (Luo et al., 2019), qui utilisent cet échantillonnage, utilisent $N = 20$ dans leurs expériences.

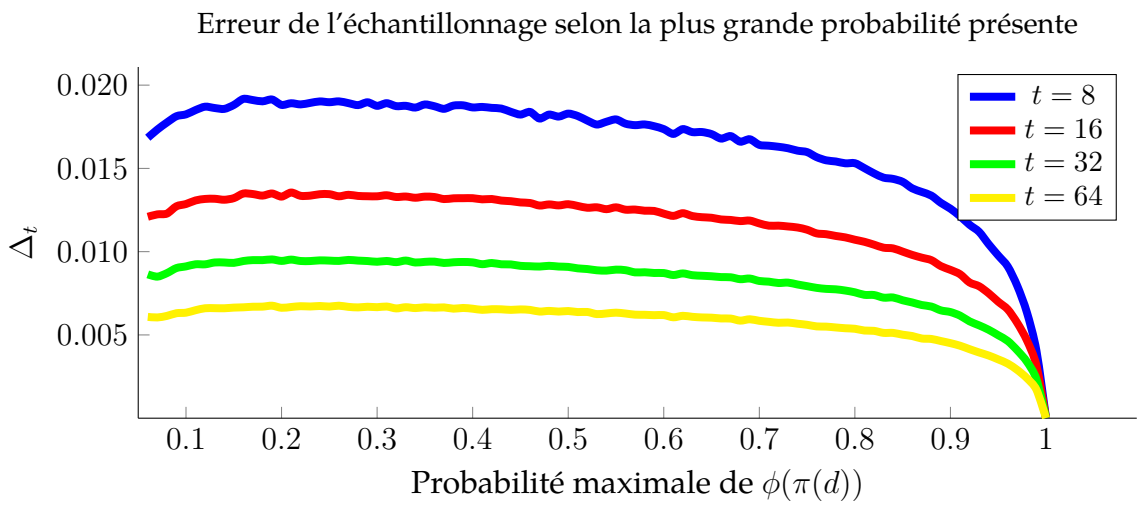

FIGURE 2.2 – Impact de la probabilité maximale de la distribution des résumés sur l'erreur d'échantillonnage.

2.3 BanditSum

Maintenant que l'on sait que l'on peut avoir des approximations très justes pour J (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_θ prenant en entrée un document et produisant une distribution sur les phrases comme un bandit contextuel, que l'on peut apprendre efficacement avec REINFORCE.

2.3.1 Description

Le système présenté est attribué à (Dong et al., 2018), qui l'ont nommé BanditSum. Dans BanditSum, on considère un réseau de neurones h_θ comme un bandit contextuel, lequel a pour objectif de maximiser le score des résumés qu'il produit pour l'ensemble des documents d d'un jeu de données.

L'architecture retenue pour h est séparée en un encodeur de phrases et une tête de prédiction d'affinités. Pour l'encodeur de phrases, un LSTM bidirectionnel à une couche prend en entrée la séquence des plongements de mots de chaque phrase et retourne une représentation de chaque phrase par rapport aux mots qu'elle contient. Ces représentations sont par la suite fournies à un LSTM bidirectionnel à deux couches qui produit une représentation $\tilde{d}_i$ de chaque phrase par rapport aux autres phrases du même document. La tête de prédiction est constituée d'un réseau pleinement connecté prenant en entrée les représentations $\tilde{d}_i$ et produisant une affinité entre 0 et 1 pour chaque phrase. Les affinités liées à chaque phrase sont enfin regroupées et divisées par leur somme pour produire la distribution $\pi_\theta(d)$ sur les phrases du document. À l'entraînement, les paramètres θ sont appris via REINFORCE en

générant les résumés avec le processus de génération stochastique ξ. À l'inférence, c'est le processus vorace ψ qui est employé, sélectionnant les 3 phrases avec la plus grande affinité.

TODO: Figure de l'architecture de BanditSum.

Banditsum incorpore aussi deux artefacts techniques pour faciliter la convergence. Ceux-ci sont énumérés plus bas et accompagnés de l'intuition justifiant leur utilisation.

D'abord, BanditSum utilise la version de REINFORCE incorporant une **baseline** $b(d)$:

$$\nabla J(\theta) = \frac{1}{N} \sum_{t=1}^{N} \left(R(\hat{s}_t, s) - b(d)\right) \nabla \ln \xi \left(\pi_\theta(\hat{s}_t)\right). \tag{2.4}$$

La baseline retenue est $b(d) = R\left(\psi(\pi), s\right)$, représentant le score associé au résumé le plus probable document d. L'introduction de b peut être vue comme assignant un scalaire positif aux résumés meilleurs que le résumé actuellement privilégié et un sclaire négatif à ceux qui sont moins bons. En pratique, l'introduction de baseline est fréquemment utilisée pour réduire la variance élevée dans l'entraînement avec REINFORCE et permettre un apprentissage plus stable.

Le second artefact employé par BanditSum est celui de l'ajout d'une exploration artificielle ϵ dans le processus de pige de résumé ξ. Afin d'assurer une exploration satisfaisante des résumés possibles d'un document, le processus de génération stochastique ξ qu'ils utilisent pige une phrase au hasard dans une proportion $\epsilon = 0.1$ du temps. L'introduction de ϵ a pour effet d'assurer que les résumés pigés selon $\xi(\pi, \epsilon)$ seront suffisamment variés pour assurer une bonne exploration de l'espace des résumés possibles lors de l'entraînement. Enfin, notons que la probabilité de générer un résumé $\hat{s}$, représentée par $\xi\left(\pi_\theta(\hat{s}_t), \epsilon\right)$ dans l'équation 2.4, s'écrit alors sous la forme close

$$\xi\left(\pi_\theta(\hat{s}_t), \epsilon\right) = \prod_{i=1}^{3} \left(\frac{\epsilon}{|d| - i + 1} + \frac{(1-\epsilon)\pi(d)_{s_i}}{\sum_{k=1}^{j-1} \pi(d)_{s_j}} \right), \tag{2.5}$$

où s_i représente l'index de la i-ème phrase du résumé $\hat{s}$. Ainsi, le gradient de (2.5), nécessaire dans la mise à jour des paramètres par REINFORCE, est facilement calculable avec les logiciels de différentiation automatique habituellement utilisés pour les réseaux de neurones.

2.3.2 Expériences

On roule notre propre implémentation de BanditSum, dont le pseudocode se trouve à l'algorithme 2, sur le jeu de données CNN/DailyMail complet. Pour les expériences, on fait varier N, pour explorer l'importance que peut avoir un meilleur estimé du gradient sur le temps de calcul vs la rapidité de convergence.

Algorithme 2 Système utilisant un bandit contextuel

Entrée: $\mathcal{D}$ (jeu de données), h_θ (modèle neuronal), N (nombre d'échantillons), α (taux d'apprentissage), ϵ (taux d'exploration).

tant que vrai **faire**
  $(d, s) \sim \mathcal{D}$ ▷ On pige un document du jeu de données
  $\pi = h_\theta(d)$
  Piger N résumés $\hat{s}_i \sim \xi(\pi, \epsilon)$
  Calculer les N scores associés $r_i = R(\hat{s}_i, s)$
  $b = R(\psi(\pi), s)$
  $\nabla = \frac{1}{N} \sum_{i=1}^{N} (r_i - b) \nabla_\theta \ln \xi\left(\pi(\hat{s}_i), \epsilon\right)$ ▷ Mise à jour REINFORCE avec baseline
  $\theta = \theta + \alpha \nabla$

Détails d'implémentation

Nous reprenons exactement les mêmes paramètres d'entraînement que (Dong et al., 2018), que nous énumérons.

Pour les LSTMs, la dimension cachée est fixée à 200. La tête de prédiction est un réseau pleinement connecté avec une seule couche cachée de taille 100, activation ReLU (Agarap, 2018) et sortie sigmoïde.

Les plongements de mots utilisés sont les plongements GLoVe (Pennington et al., 2014) de taille 100 pré-entraînés sur la langue anglaise. L'optimiseur employé est Adam (Kingma and Ba, 2014), pour lequel on fixe $\beta = [0, 0.999]$. On utilise un taux d'apprentissage $\alpha = 5e^{-5}$ et une pénalité sur la norme des poids θ de $1e^{-6}$. Un *gradient clipping* de 1 est aussi appliqué. TODO: GPU et CPU utilisés.

2.3.3 Résultats

Rapporter graphe de l'évolution de R en validation selon le nombre de documents vus. En validation et en test, on calcule le vrai ROUGE pour éviter toute possibilité d'erreur introduite dans le pré-calcul. On garde quand même l'heuristique que l'ordre des phrases du résumé est le même que celui dans le document, car c'est l'ordre qui risque le moins d'introduire des incohérences syntaxiques (références mal-définies). Rapporter une courbe pour chaque N parmi $\{8,16,32\}$, qui sont les valeurs les plus pertinentes selon les expés plus haut. Pour chacune des valeurs de N à l'essai, on roule l'algorithme à 5 reprises en faisant varier le générateur de nombres aléatoires et on rapporte la moyenne et déviation standard. Rapporter la courbe que BanditSum dit obtenir pour $N = 20$. Rouler 4 epochs, selon ce que BS rapporte comme meilleur modèle.

Dans un tableau, rapporter performance en test de Banditsum (nous) vs Banditsum (paper) et SOTA extractif. Rapporter ROUGE-1, ROUGE-2, ROUGE-L et leur moyenne.

Dans un autre tableau, rapporter temps d'entraînement et architecture de calcul utilisée pour

montrer l'impact de N sur le temps de calcul. Discussion sur non-nécessité de vrai gradient de J pour l'apprentissage, mais que ça stabilise naturellement.

2.4 Conclusion

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

Chapitre 3

Bandit combinatoire

Ce chapitre et le prochain représentent des contributions entièrement nouvelles au domaine de la génération de résumés. L'idée principale qui se répète dans les deux chapitres est celle d'utiliser les approches par bandits pour générer des cibles plus riches pour la génération de résumés que les cibles binaires habituellement employées dans les formulations supervisées.

Dans ce chapitre, on présente la formulation en bandit combinatoire et comment elle s'applique à la génération du résumé d'un document. On décrit ensuite l'algorithme UCB (Auer et al., 2002) qui minimise le regret du bandit combinatoire proposé et comment il peut être employé pour générer des cibles pour un système de génération de résumés. L'applicabilité des cibles proposées est par la suite validée en montrant la progression de leur qualité sur des documents issus d'un jeu de données de développement. Enfin, on présente comment ces cibles peuvent être intégrées dans un système complet de génération de résumés, que l'on compare aux performances de l'état-de-l'art et l'approche par bandit contextuel présentée au chapitre précédent.

3.1 Formulation combinatoire

Le bandit combinatoire est une approche par bandit qui vise à étendre la formulation MAB à des environnements où plusieurs actions peuvent être sélectionnées en même temps par l'apprenant. Nous nous intéresserons seulement à la variante dite multitâche (Lattimore and Szepesvári, 2020), où il y a un nombre fixé m d'actions sélectionnées à chaque pas de temps, créant une *super-action* $\mathcal{M} = \{a_1, ..., a_m\}$ pour laquelle l'environnement retourne une récompense $X_{\mathcal{M},t}$. L'objectif demeure de minimiser le regret (1.1), mais celui-ci est désormais calculé entre le *super-bras* optimal $\mathcal{M}^*$ et les *super-bras* tirés $\mathcal{M}_t$.

Tout l'intérêt de cette formulation vient de l'éventuelle relation entre les *super-actions*. Bien qu'il serait possible de formuler le bandit multitâche comme un MAB où l'ensemble des actions possibles est l'ensemble des *super-actions* $\mathcal{M}$, cette formulation risque d'être très peu

efficace. En effet, comme les actions a_i sont partagées entre les divers *super-actions*, il est possible d'utiliser la récompense associée à une *super-action* pour déduire de l'information sur la récompense associée aux actions qui la composent. Cette information sur les actions a_i peut alors être utilisée pour guider le choix des prochaines *super-actions* et accélérer l'apprentissage.

La génération du résumé d'un document d peut être vue comme un bandit combinatoire où les actions de base sont les phrases d_i et les *super-bras* $\mathcal{M}$ sont les résumés de 3 phrases possibles. En tirant un *super-bras* $\mathcal{M}_{\hat{s}}$ associé au résumé $\hat{s}$, le bandit perçoit $R(\hat{s}, s)$ comme récompense et peut mettre à jour ses croyances sur les phrases d_i de $\hat{s}$.

Notons d'abord que, comme on ne tient pas compte de l'ordre dans nos expériences, il est naturel de considérer que chaque phrase d'un résumé contribue à part égale au score observé. Implicitement, cela revient à faire l'hypothèse que la récompense associée à un résumé $\mathcal{M}$ est la moyenne de la récompense associable à chaque phrase. Or, comme le score R est basé sur un score F1 sur le résumé complet, cette hypothèse n'est pas strictement vraie mais elle est intuitivement valide. En effet, les phrases qui résument bien le document généreront des résumés aux scores plus élevés et donc leur présence à elle seule doit contribuer à augmenter le score associé à un résumé.

Aussi, contrairement à la formulation contextuelle du chapitre 2, la formulation combinatoire présentée ici s'applique à la génération du résumé d'un seul document d. Ainsi, au lieu d'attaquer le problème large de la génération de n'importe quel document, on pose la génération du résumé de chaque document comme une instance de bandit séparée. On ne peut donc pas utiliser directement cette formulation combinatoire de pair avec un algorithme comme REINFORCE pour apprendre un système complet de génération de résumés. La prochaine section décrit comment c'est plutôt la solution du bandit combinatoire qui pourrait être utilisée comme cible dans un contexte d'apprentissage supervisé.

3.2 Upper Confidence Bound (UCB)

L'algorithme UCB (Auer et al., 2002) est un algorithme de minimisation du regret pour les bandits stochastiques qui peut facilement être généralisé au cas combinatoire. UCB est basé sur le principe de l'optimisme en cas d'incertitude, qui permet de résoudre de manière élégante le dilemme exploration-exploitation décrit à la section 1.1. Pour une action i au temps t, l'algorithme définit une borne supérieure

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

où $\bar{x_i}(t)$ est la moyenne des récompenses reçues jusqu'au temps t par l'action i, $n_i(t)$ est le

nombre de fois que l'action i a été sélectionnée et β est un hyperparamètre régulant l'exploration. La borne supérieure ainsi définie garantit avec une forte probabilité que la valeur μ_i espérée pour l'action i soit inférieure à UCB_i au temps t.

L'équation (3.1) peut être visualisée comme résolvant directement le dilemme exploitation-exploration. En effet, le terme $\bar{x_i}(t)$ présente l'estimation actuelle de la récompense associée au bras i et permet donc d'exploiter les connaissances acquises avant le temps t. À gauche, le terme $\sqrt{\frac{2 \ln t}{n_i(t)}}$ est proportionnel à l'inverse des visites relatives du bras i. Il s'agit donc d'un terme qui sera plus élevé pour les actions moins visitées, incitant ainsi l'exploration d'actions sous-visitées.

3.2.1 Application au bandit combinatoire

Bien que UCB est conçu pour les problèmes de bandits stochastiques, l'algorithme peut être naturellement étendu au problème de bandit combinatoire. En effet, il suffit alors de choisir à chaque ronde le super-bras maximisant la borne supérieure définie par UCB. Dans notre cas, cela revient à sélectionner le super-bras composé des trois actions avec la borne supérieure maximale, i.e.

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

pour lequel le score $R(\mathcal{M}_t, s)$, associé au résumé $\hat{s}_t$ correspondant aux indices de $\mathcal{M}$, est perçu.

Une question demeure toutefois : comment peut-on utiliser UCB pour générer des cibles pour la génération du résumé d'un document ? Nous proposons naturellement d'utiliser les quantités $\bar{x_i}(T)$ et $n_i(T)$ générées par UCB après T rondes pour chaque phrase i.

Intuitivement, $\hat{x}_i$ et $\hat{n}_i$ peuvent être vus comme des estimations de la solution optimale. Comme la récompense associée à un bras i au temps t dépend des autres bras présents dans le super-bras $\mathcal{M}_t$ et que UCB identifierait éventuellement le super-bras optimal et qu'avec un nombre infini de samples, la distribution induite par $\hat{n}_i$ tendrait vers le vecteur binaire représentant le résumé optimal. Similairement, les $\hat{x}_i$ convergeraient éventuellement à R^*, le meilleur score de résumé extractif d'un document, pour les phrases faisant partie du résumé optimal.

Mentionnons aussi que, par sa structure inhérente, la formulation extractive de la génération de résumés donne lieu à bon nombre de phrases aux scores similaires (quelques mots correspondant au résumé cible) et peu d'excellentes phrases, difficiles à discerner. En minimisant le regret, UCB va donc naturellement obtenir plus d'échantillons de résumés contenant les excellentes phrases. Or, comme les excellentes phrases sont limitées, il faut échantillonner plus de résumés les contenant avant d'obtenir une bonne estimation de leur valeur, contrai-

rement aux phrases insatisfaisantes dont la valeur est rapidement estimée après quelques échantillons. Il est donc justifié d'utiliser les quantités mesurées par UCB pour obtenir de l'information sur toutes les phrases d'un document, pas seulement pour trouver le résumé optimal.

3.2.2 Expériences

Si l'intuition de l'utilisation des quantités calculées par UCB comme cibles est naturelle pour le cas où le nombre de rondes effectué est immense, il demeure néanmoins important de vérifier comment ces cibles sont adéquates sur un nombre de rondes plus restreint. Mais, comment savoir quand on approche de ce stade d'optimalité et que l'on peut considérer les cibles comme adéquates ? Réponse : quand le score du résumé le plus prometteur selon UCB, nommons le R_t, commence à converger vers le score R^* du résumé optimal. En mesurant la sous-optimalité $\Delta_t = R^* - R_t$ des cibles générées par UCB, on peut donc savoir à partir de quel moment les cibles seraient satisfaisantes pour être utilisées en entraînement.

On reprend donc le jeu de 25 000 documents pour mener des expériences sur la rapidité de l'obtention de cibles satisfaisante par UCB. Les résultats des figures 3.1, 3.2 et 3.3 sont obtenus [1] en calculant les Δ_t à partir de l'algorithme 3.

Algorithme 3 UCB combinatoire pour génération de résumé

Entrée: d (document), s (résumé cible), β (paramètre d'exploration), T (nombre de rondes).

Initialiser $\bar{x}_i = 0$ et $n_i = 0$ pour $i = 1, ..., |d|$
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 $\text{UCB}_i = \infty$
  $\mathcal{M}_t = 3\text{-arg max UCB}_i$
  $r_t = R(\mathcal{M}_t, s)$
  Mettre à jour $\bar{x}_i$ et n_i pour $i \in \mathcal{M}_t$

3.2.3 Résultats

La figure 3.1 présente l'importance du critère de sélection utilisé pour trouver le résumé optimal selon UCB au temps t. Deux approches sont 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 selon 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

1. Les résultats sont en fait obtenus en utilisant la version de UCB avec connaissances a priori uniformes de la section suivante. Les deux versions sont fondamentalement équivalentes et on utilise le prior uniforme seulement pour garder des valeurs de β cohérentes entre les figures.

éventuellement au même résumé optimal, comme c'était à prévoir. Enfin, la conclusion retenue est que les $\bar{x}_i$ forment de meilleurs cibles que les n_i. Pour les prochaines figures, les résultats rapportés seront donc seulement ceux obtenus selon le critère de sélection basés sur $\bar{x}_i$.

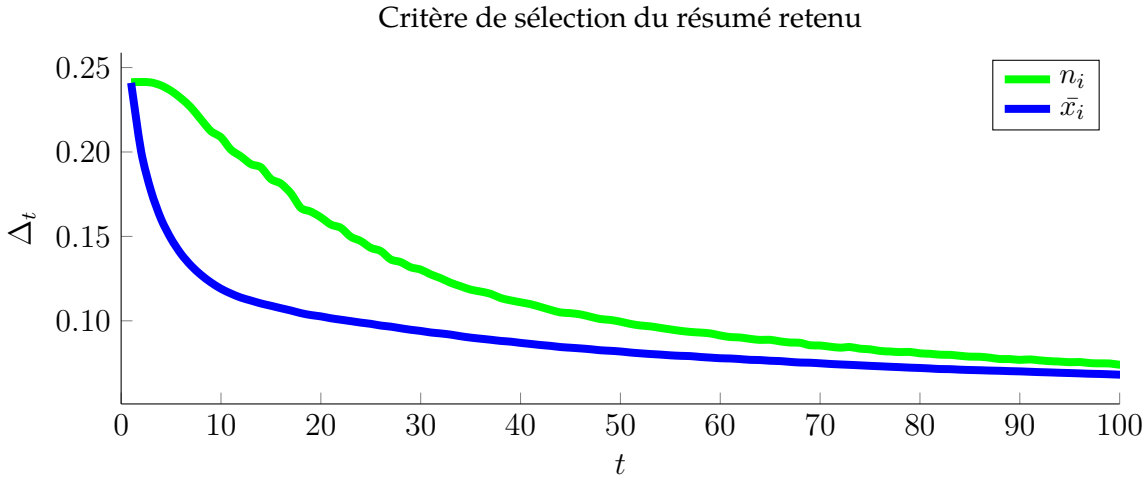

FIGURE 3.1 – Impact du critère de sélection utilisé pour choisir le résumé retenu au temps t.

La figure 3.2 présente comment l'hyperparamètre β influence la convergence de UCB. On remarque que $\beta = 10$ (courbe verte sur la figure) semble être le bon compromis entre exploration et exploitation, réussissant à converger à $\Delta_t \approx 0.05$ après 250 rondes de UCB.

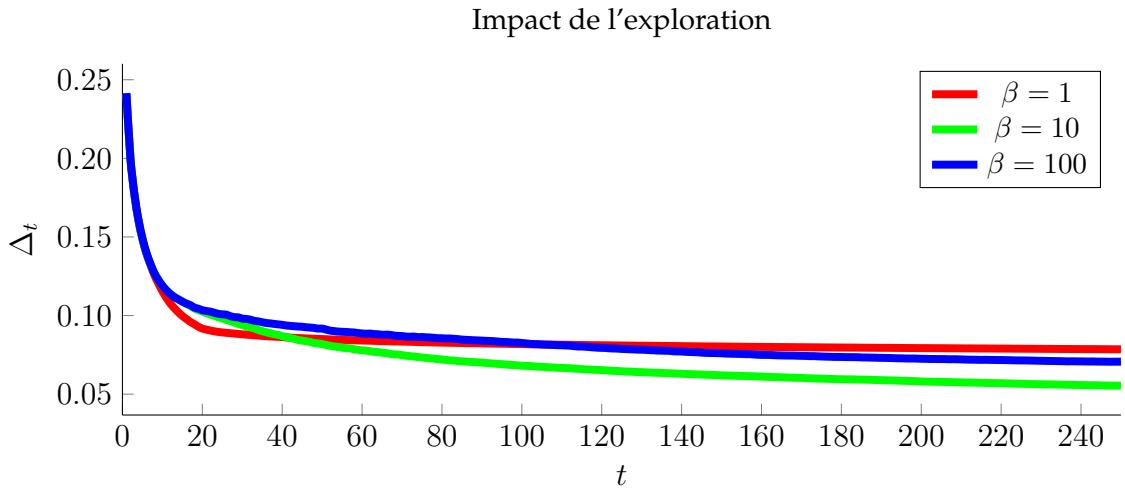

FIGURE 3.2 – Impact du paramètre d'exploration β utilisé par UCB sur la convergence.

La figure 3.3 évalue dans quelle mesure la taille des documents influe sur la sous-optimalité des cibles générées par UCB. 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 converge 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. À cette fin, on remarque que la performance stagne autour de 100 rondes pour les documents courts mais continue à augmenter jusqu'à 250 rondes pour les longs documents. Un bon point de départ serait donc de prendre $t(|d|) = 4|d| + 50$ pour atteindre s'assurer d'effectuer toujours au moins 50 rondes de UCB et progressivement augmenter le nombre de rondes jusqu'à 250 pour les documents de 50 phrases, la taille maximale allouée par notre régularisation.

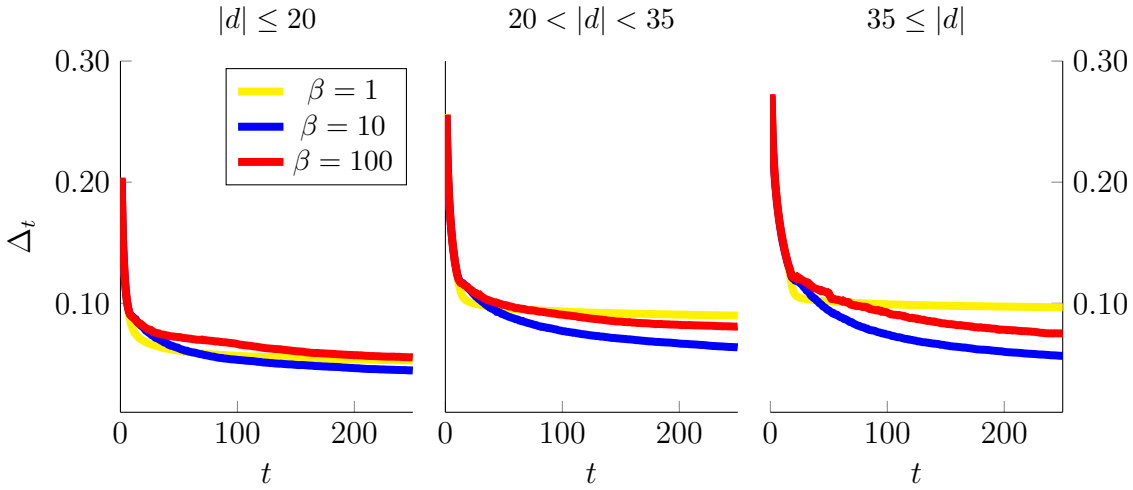

FIGURE 3.3 – Impact de la taille du document sur la convergence.

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 avec les vecteurs binaires. À la manière de (Silver et al., 2016), il est intéressant de considérer la possibilité d'introduire une connaissance a priori (prior) dans le calcul de UCB. En effet, si l'on possède une certaine intuition 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 priori 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.

L'introduction de P_i aura pour effet de forcer UCB à concentrer son échantillonnage autour des phrases où P_i est plus élevé. Intuitivement, cela peut éviter d'explorer des phrases que l'on sait peu prometteuses et forcer UCB à se concentrer sur quelques phrases d'intérêt.

3.3.1 Expériences

On s'intéresse à voir comment l'introduction de connaissances a prior permet d'améliorer la convergence de UCB. Sur un même document, on évalue l'impact d'un prior basé sur (1) le meilleur résumé, (2) le résumé médian et (3) le pire résumé. Pour chacun des résumés retenus, on bâtit une distribution a prior $P(d)$ sur les phrases basée sur une moyenne pondérée entre une distribution uniforme $U(d)$ et la distribution vorace $G(d)$ associée au résumé :

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

où on explore $\tau \in \{0.0, 0.1, 0.2, 0.3\}$. On évalue, pour chacun des priors générés, l'évolution Δ_t en fonction du temps, comme à la section précédente. L'expérience est répétée pour tous les 25 000 documents du jeu de développement.

3.3.2 Résultats

Les résultats des expériences sont rapportés dans les figures 3.4 et 3.5.

Sur la figure 3.4, commençons par noter que le cas $\tau = 0.0$ correspond exactement aux expériences sans prior de la section précédente. On remarque ainsi que l'introduction d'un prior accélère l'apprentissage, mais seulement dans le cas où le prior est assez près d'une distribution uniforme pour permettre une exploration suffisante. Notons aussi que ces résultats sont obtenus à partir du résumé médian et qu'ils représentent donc le cas où la connaissance a prior n'est pas particulièrement bonne. Comment alors expliquer le gain de performance ? On fait l'hypothèse que le prior permet en pratique de diminuer la taille du problème de bandit combinatoire à résoudre. En effet, en forçant UCB à se concentrer sur certaines phrases, le prior permet intuitivement à UCB de passer de la recherche difficile du résumé optimal à la recherche du résumé optimal contenant une phrase identifiée comme pertinente par le prior. Asymptotiquement, l'introduction de prior sera donc néfaste sur la convergence mais, en pratique, elle peut accélérer l'apprentissage de manière appréciable.

La figure 3.5 présente l'impact de la qualité du prior sur l'évolution de Δ_t. On y remarque, sans surprise, que le prior basé sur le meilleur résumé converge nettement plus rapidement. La portion intéressante se situe dans le constat que les priors basés sur le résumé médian et le pire résumé se comportent de manière similaires, dépassant tous deux la performance obtenue avec une connaissance a priori uniforme. La performance surprenante du mauvais prior, celui basé sur le pire résumé, est probablement explicable encore une fois par la diminution

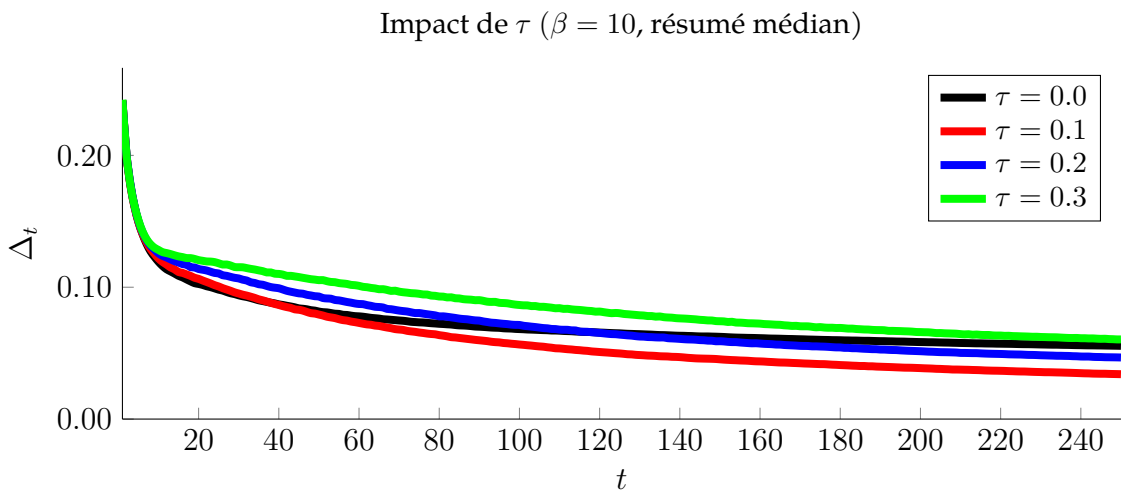

FIGURE 3.4 – Impact de la similarité de distributions a priori avec une distribution uniforme, selon le nombre t de rondes effectuées.

de la taille du problème à résoudre. En effet, même si le résumé est mauvais, cela n'écarte pas qu'une phrase décente puisse s'y trouver et être utilisée pour converger rapidement à un résumé de qualité satisfaisante. En perspective, on peut donc dire que le processus UCB avec prior présenté semble appliquer un raffinement du prior.

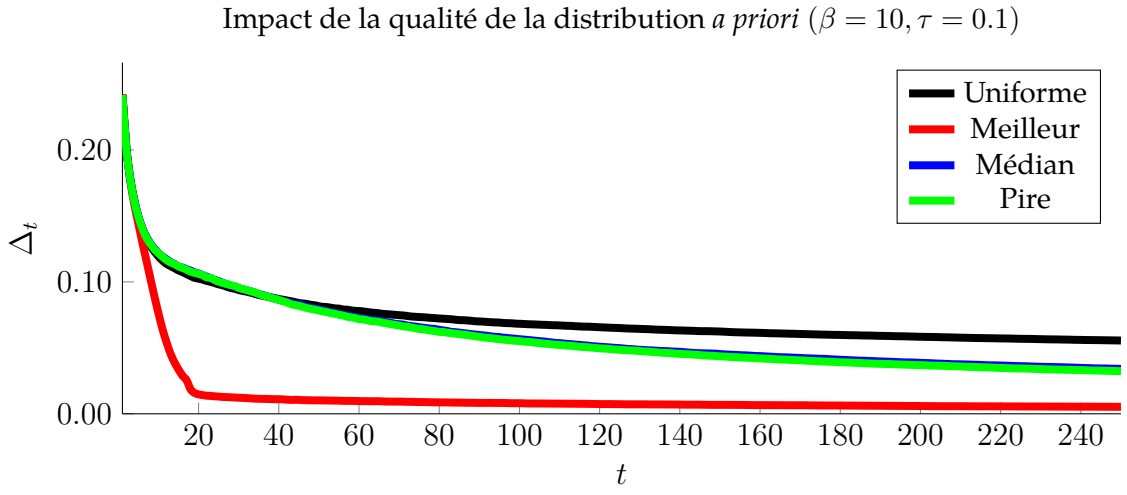

FIGURE 3.5 – Impact de la similarité de distributions à priori avec une distribution uniforme, selon le nombre t de rondes effectuées.

3.4 CombiSum

On propose maintenant un système de génération de résumés complet nommé CombiSum. CombiSum réutilise exactement la même architecture que BanditSum, mais fait son apprentissage à partir des cibles générées par UCB plutôt que par REINFORCE. La procédure d'entraînement correspond alors davantage aux approches supervisées, où l'on pige une minibatch de documents, on obtient les cibles associées que l'on entraîne un réseau de neurones à prédire. Dans notre cas, les cibles sont donc les valeurs $\bar{x_i}$ retournées par UCB, que l'on peut naturellement apprendre via une fonction de perte quadratique. À l'inférence, CombiSum génère le résumé en regroupant les 3 phrases pour lesquelles sa prédiction est maximale.

Inspirés par les bons résultats obtenus avec une connaissance a priori, on considère aussi l'utilisation de priors basés sur la prédiction de notre réseau. Pour ce faire, on applique une fonction softmax σ de température τ aux affinités x_i prédites par le réseau :

$$\sigma(x, \tau)_i = \frac{e^{x_i/\tau}}{\sum_{j=1}^{|d|} e^{x_j/\tau}}.$$

Plus la température τ augmente, plus la distribution générée par σ sera près de la distribution uniforme. On propose donc de faire décroître τ progressivement jusqu'à 1, permettant d'avoir un prior presque uniforme au début de l'apprentissage où le réseau est encore mauvais et progressivement accorder plus d'importance au prior au fur et à mesure que le réseau s'améliore. On garde toujours une température d'au moins 1 pour assurer que le prior permettra toujours une exploration satisfaisante.

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 6.

Algorithme 4 CombiSum

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), τ (température).

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**
    $x = h_\theta(d)$
    $P = \sigma(x, \tau)$
    Générer les cibles $\hat{x}$ après T rondes de UCB avec prior P.
    $\nabla = \nabla + \nabla_\theta \|x - \hat{x}\|^2$
  $\theta = \theta - \alpha\nabla$

Pour les expériences, on teste l'impact de l'utilisation ou non de priors. On explore aussi de faire varier le nombre de rondes de UCB utilisées en fonction du nombre de phrases ou de le fixer à $T = 150$. Cela fait donc un total de 4 configurations, que l'on roulera chacune 5 fois pour estimer la moyenne et la variance.

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.

Idem au chapitre précédent.

3.5 Conclusion

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

Chapitre 4

Bandit combinatoire linéaire

L'approche combinatoire pure présentée au chapitre précédent présente un inconvénient notoire : elle n'utilise pas de relation de similarité entre les phrases pour propager de l'information entre deux phrases similaires. 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 récompense espérée.

Dans ce chapitre, on présente la formulation en bandit combinatoire linéaire et comment elle peut s'appliquer au processus de génération du résumé d'un document. On aborde l'algorithme LinUCB (Chu et al., 2011) qui minimise le regret sur le bandit combinatoire linéaire et on suggère comment il peut aussi être utilisé pour générer des cibles pour un système de génération de résumés. On valide encore une fois l'applicabilité des cibles proposées à partir d'un jeu de données de développement. Enfin, on présente comment ces cibles peuvent être intégrées dans un système complet de génération de résumés, que l'on compare aux performances de l'état-de-l'art et aux approches par bandits présentées précédemment.

4.1 Formulation combinatoire linéaire

De manière analogue à comment nous avons présenté la formulation combinatoire, débutons par considérer le cas multi-bras linéaire avant d'incorporer la portion combinatoire. La formulation en bandit linéaire reprend exactement le même cadre formel que la bandit mutli-bras stochastique vu à la section 1.1. 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 suppose que, pour les vecteurs $\tilde{a}_i$, tous soumis à $\|\tilde{a}_i\| \leq 1$, il existe un vecteur unique ω_* avec $\|\omega_*\| \leq 1$ de récompense permettant de relier chaque à sa récompense espérée μ_i, i.e. $\langle \omega_*, \tilde{a}_i \rangle \approx \mu_i$ pour tout i. Lorsque l'on se place dans la version combinatoire du bandit linéaire, l'hypothèse linéaire est alors simplement appliquée à des représentations vectorielles des *super-bras* $\tilde{\mathcal{M}}$.

4.2 Linear Upper Confidence Bound (LinUCB)

L'algorithme LinUCB (Chu et al., 2011) représente une application du principe de l'optimisme face à l'incertitude sur le problème de bandit linéaire qui vise encore une fois la minimisation du regret cumulatif. Rappelons que ce principe consiste à avoir, pour chaque action i, un estimé optimiste UCB_i de sa valeur espérée μ_i, lequel est plus élevé que μ_i avec forte probabilité. Dans le contexte linéaire, comme on souhaite faire usage des représentations vectorielles des actions, on cherche à avoir UCB_i qui fait usage des relations linéaires entre les actions. Nous présentons plus bas l'intuition derrière la construction de bonnes bornes supérieures UCB_i en omettant certains détails techniques pour alléger la lecture. Le lecteur plus curieux pourra se référer à (Chu et al., 2011) ou (Abbasi-Yadkori et al., 2011) pour des descriptions techniques complètes.

Une première intuition clé pour bâtir une bonne borne supérieure est celle d'exploiter les observations faites aux rondes précédentes pour bâtir un estimé $\hat{\omega}_N$ de ω_*. Cet estimé devrait naturellement viser à combiner aussi bien que possible les paires bras-récompenses $(\tilde{a}_{i_t}, X_T)$ observées aux rondes précédentes, i.e.

$$\langle \hat{\omega}_N, \tilde{a}_{i_t} \rangle \approx X_t. \tag{4.1}$$

L'équation (4.1) peut être vue comme un problème de moindres carrés, où l'on souhaite trouver

$$\hat{\omega}_N = \underset{\omega \in \mathbb{R}^n}{\arg\min} \sum_{t=1}^{N-1} \left(\langle \omega, \tilde{a}_{i_t} \rangle - X_t \right)^2. \tag{4.2}$$

Or, le problème (4.2) n'admet pas nécessairement de solution exacte ou unique. LinUCB emploie donc la régularisation de Tikhonov (Tikhonov, 1963) avec $\lambda = 1$ pour garantir une solution et son unicité. Le centre $\hat{\omega}_N$ est alors choisi selon la solution analytique connue du problème

$$\hat{\omega}_N = {\mathbf{V_N}}^{-1} \mathbf{b_N}, \quad \text{pour} \quad \mathbf{V_N} = (\mathbf{A_N}^\top \mathbf{A_N} + \lambda \mathbf{I}) \quad \text{et} \quad \mathbf{b_N} = \mathbf{A_N}^\top \mathbf{X_N}, \tag{4.3}$$

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

En pratique, $\mathbf{V_N}$ et $\mathbf{b_N}$ peuvent tous deux être calculés de manière itérative selon

$$\mathbf{V_N} = \lambda \mathbf{I} + \sum_{t=1}^{N} \tilde{a}_{i_t} \tilde{a}_{i_t}^\top, \qquad \mathbf{b_N} = \sum_{t=1}^{N} \tilde{a}_{i_t} X_t. \tag{4.4}$$

Maintenant que l'on possède une bonne estimation $\hat{\omega}_N$ qui exploite l'expérience des rondes précédentes, on cherche à introduire un terme garantissant une exploration satisfaisante des

actions possibles. LinUCB propose à cet effet d'utiliser

$$\sqrt{\beta \tilde{a}_i^\top {\mathbf{V_N}}^{-1} \tilde{a}_i},$$

un terme qui base l'exploration liée à une action i à sa différence par rapport aux actions précédentes représentées par $\mathcal{A}_\mathcal{N}$ et un hyperparamètre d'exploration β. Les détails techniques complets permettant d'arriver au terme pour l'exploration sont donnés dans (Abbasi-Yadkori et al., 2011).

En regroupant les termes d'exploitation et d'exploration, on obtient encore une fois une borne supérieure résolvant de manière élégante le compromis exploration-exploitation :

$$\text{UCB}_i = \langle \omega_N^\top, a_i \rangle + \sqrt{\beta \tilde{a}_i^\top {\mathbf{V_N}}^{-1} \tilde{a}_i}, \tag{4.5}$$

où β représente encore un hyperparamètre balançant l'exploration.

Connexions avec UCB

Nous prenons un moment ici pour mettre l'emphase sur les connexions entre UCB et LinUCB, qui peut être vu comme une généralisation linéaire directe de UCB. En effet, si l'on considère les vecteurs d'action correspondant aux vecteurs unitaires, i.e. $\tilde{a}_i = e_i$, et une régularisation de Tikhonov avec $\lambda = 0$, on a que $\mathbf{V_N}$ est une matrice diagonale, où la i-ème entrée est le nombre de fois où l'action i a été sélectionnée. À ce moment, le terme $\tilde{a}_i^\top {\mathbf{V_N}}^{-1} \tilde{a}_i$ n'est donc que l'inverse du nombre n_i de fois que l'action i a été sélectionnée et le terme d'exploration devient

$$\sqrt{\frac{\beta}{n_i(N)}},$$

un terme proportionnel à celui présent dans UCB si l'on pose $\beta = 2\log(N)$.

Aussi, $\hat{\omega}_N$ devient simplement le vecteur où l'entrée i correspond à la moyenne des récompenses perçues pour l'action i. Le produit vectoriel $\langle \hat{\omega}_N, e_i \rangle$ ne fait donc qu'aller chercher la moyenne $\bar{x}_i$ utilisée dans UCB.

Ainsi, si l'on se positionne dans le cas où aucune information n'est partagée par les vecteurs d'action $\tilde{a}_i$, LinUCB est équivalent à UCB. On peut donc considérer que, si l'hypothèse linéaire est respectée, LinUCB obtiendra toujours une performance supérieure ou égale à UCB.

4.2.1 Représentations vectorielles de phrases

Les représentations vectorielles utilisées pour les phrases sont basées sur l'observation que les calculs de ROUGE sont essentiellement basés sur les comptes de n-grammes . On choisit donc de générer, pour chaque phrase, un vecteur parcimonieux représentant le nombre de

fois qu'un n-gramme donné y est présent. Pour que les relations entre les phrases soient incorporées dans ces vecteurs, on associe à chaque n-gramme du document un index unique i. On obtient donc, pour chaque phrase, un vecteur parcimonieux de taille N, pour le nombre N de n-grammes dans le document, où l'indice i correspond au nombre d'occurrences du n-gramme i dans la phrase. Dans nos expériences, on limite l'exploration des n-grammes aux unigrammes, afin de garder une taille raisonnable pour les vecteurs parcimonieux générés.

On bâtit ensuite une matrice parcimonieuse M dont les rangées sont les vecteurs parcimonieux de chaque phrase du document. La dimension de la matrice M est ensuite réduite via une analyse sémantique latente (Kolda and O'Leary, 1998), applicable naturellement dans notre cas de matrice parcimonieuse de grande taille. Le processus retourne $|d|$ vecteurs de taille $|d|$, que l'on normalise pour avoir une norme unitaire et qu'on utilise pour les représentations vectorielles $\tilde{a}_i$. On note ici que, si aucun n-gramme n'était partagé entre les phrases, leur représentation vectorielle $\tilde{a}_i$ serait alors simplement e_i et, comme mentionné plus haut, LinUCB serait équivalent à UCB.

Comme pour les scores R, les vecteurs $\tilde{a}_i$ associés à un document peuvent être pré-calculés pour ne pas avoir à les recalculer inutilement à chaque fois que l'on traite un document.

4.2.2 Application à la génération de résumés

Pour appliquer LinUCB au contexte combinatoire, il suffit de disposer de représentations vectorielles $\tilde{\mathcal{M}}$ des super-bras. Dans le contexte de la génération de résumés, on se base encore une fois sur la contribution égale de chacune des phrases d'un résumé pour obtenir les représentations vectorielles des résumés. Pour ce faire, on prend la représentation vectorielle de chacune des phrases et que la représentation d'un résumé de 3 phrases est alors représentée par la moyenne des phrases qu'il contient

$$\tilde{\mathcal{M}} = \frac{1}{3} \sum_{i \in \mathcal{M}} \tilde{a}_i. \tag{4.6}$$

Ainsi, à chaque ronde t, la version combinatoire de LinUCB sélectionnera

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

pour UCB_i tel que défini à l'équation (4.5).

L'objectif est encore une fois d'utiliser LinUCB pour générer des cibles. Dans notre cas, comme LinUCB ne garantit plus l'exploration de chacune des actions, on utilise alors les approximations fournies par $\langle \omega_N, \tilde{a}_i \rangle$ comme cibles au lieu de $\hat{x}_i$. On mesure la qualité de la cible générée par LinUCB par le score $R_t = R(\mathcal{M}, s)$ du meilleur résumé $\mathcal{M}$ pris en sélectionnant les 3 actions dont la valeur $\langle \omega_t, \tilde{a}_i \rangle$ est maximale.

4.2.3 Introduction de connaissances a priori

On note aussi que, comme pour le bandit combinatoire pur, des connaissance a priori P peuvent être insérées dans le calcul de borne supérieure de LinUCB, qui devient alors

$$\text{UCB}_i = \langle \omega_N^\top, a_i \rangle + P_i \sqrt{\beta \tilde{a}_i^\top \mathbf{V_N}^{-1} \tilde{a}_i}. \tag{4.7}$$

4.2.4 Expériences

Similairement au chapitre sur la formulation combinatoire pure, on mesure la sous-optimalité $\Delta_t = R^* - R_t$ des cibles générées par LinUCB. On reprend donc le jeu de développement de 25 000 documents pour mener des expériences sur la qualité des cibles générées par LinUCB.

Les figures 4.1 et 4.2 sont obtenues avec un prior uniforme alors que les figures 4.3 et 4.4 expérimentent avec les mêmes configurations de priors décrite à la section 3.3.1. Le pseudocode permettant de recréer les expériences se trouve à l'algorithme 5.

Notre implémentation actuelle diffère légèrement du pseudocode présenté en utilisant la nature de la matrice $\mathbf{V}$ pour éviter d'avoir à calculer son inverse à chaque ronde et être computationnellement plus efficace. Les détails techniques de cette manipulation sont disponibles dans l'annexe A pour le lecteur intéressé.

Algorithme 5 LinUCB combinatoire pour génération de résumé

Entrée: d (document), s (résumé cible), β (paramètre d'exploration), T (nombre de rondes de UCB), $\tilde{a}_i$ (représentations des phrases).

$\mathbf{V} = \mathbf{I}$
$\mathbf{b} = \mathbf{0}$
pour $t = 1, ..., T$ **faire**
  $\omega_t = \mathbf{V}^{-1}\mathbf{b}$
  pour $i = 1,..., |d|$ **faire**
    $\text{UCB}_i = \langle \omega_t^\top, \tilde{a}_i \rangle + \sqrt{\beta \tilde{a}_i^\top \mathbf{V}^{-1} \tilde{a}_i}$
  $\mathcal{M}_t = 3\text{-arg max UCB}_i$
  $X_t = R(\mathcal{M}_t, s)$
  pour tout $i \in \mathcal{M}_t$ **faire**
    $\mathbf{V} = \mathbf{V} + \tilde{a}_i \tilde{a}_i^\top$
    $\mathbf{b} = \mathbf{b} + \tilde{a}_i X_t$

4.2.5 Résultats

La figure 4.1 présente comment l'hyperparamètre β influence la convergence de LinUCB. On remarque d'abord que, comme il fallait s'y attendre, LinUCB converge permet d'obtenir nettement plus rapidement des cibles de bonne qualité de UCB. En effet, l'erreur Δ_t de près 0.05 observée avec UCB après 250 rondes est plutôt observable après 50 rondes de LinUCB.

On remarque aussi $\beta = 10^8$ (courbe bleue sur la figure) semble être un bon choix, réussissant à converger légèrement plus rapidement que $\beta = 10^7$.

Commentaire: Honnêtement ici j'ai essayé des valeurs de β entre $10^5 et 10^1 0$ et le constat est que la convergence est exécrable en bas de 10^7 et similaire après. C'est difficle de rapporter plus de valeurs de β car elles sont soit aberrantes soit elles se confondent les unes avec les autres.

Il est à noter que les différences rapportées entre les différentes courbes sur les figures ne sont pas statistiquement significatives. Pour éviter d'encombrer inutilement les figures, on rapporte alors seulement les moyennes observées. Les versions incorporant la déviation standard des différentes courbes se trouvent à l'annexe B.

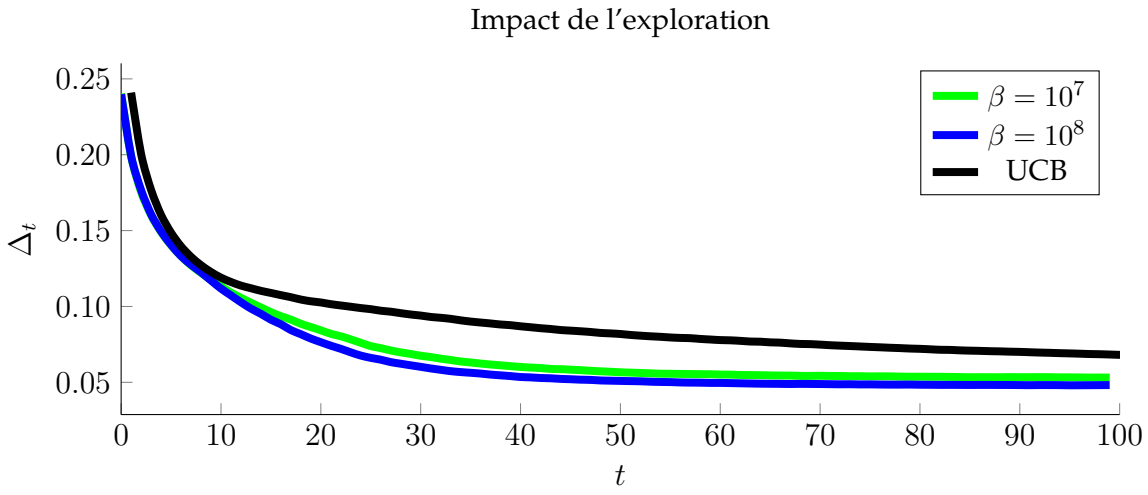

FIGURE 4.1 – Impact du paramètre d'exploration β utilisé par LinUCB sur la convergence.

La figure 4.2 évalue dans quelle mesure la taille des documents influe sur la sous-optimalité des cibles générées par UCB. La valeur $\beta = 10^8$ (courbe bleue) semble encore une fois optimale et ce, pour toutes les tailles de document. Similairement à ce qui avait été observé pour les cibles générées par UCB, il semble que le rouler LinUCB avec plus de rondes est bénéfique pour les documents plus longs. Notamment, il semble qu'un minimum de 20 rondes soit raisonnable et que les documents plus longs pourraient bénéficier d'aller jusqu'à 100 rondes.

La figure 4.3 présente comment la similarité entre le prior utilisé et la distribution uniforme sur les phrases (paramétrée par τ) impacte la qualité des cibles générées. Les résultats démontrent que l'introduction de priors ne permet absolument pas de générer de meilleures cibles, car le prior uniforme (ligne noire) obtient des résultats considérablement meilleurs.

Il est difficile de s'expliquer comment l'introduction de priors pouvait être si bénéfique à la convergence de UCB mais ne fait que nuire à LinUCB. On émet l'hypothèse que cette diffé-

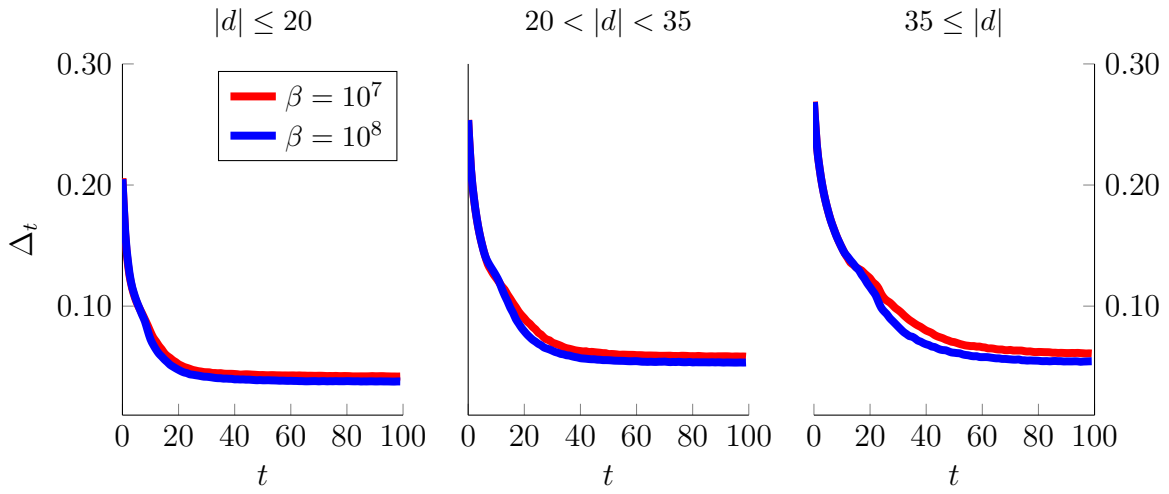

FIGURE 4.2 – Impact de la taille du document sur la convergence.

rence est dûe à la nature de l'exploitation dans LinUCB. En effet, comme LinUCB bâtit un estimé $\hat{\omega}_N$ utilisé pour estimer les valeurs de toutes les actions à partir des rondes précédentes, le fait d'introduire un prior peut faire en sorte que $\hat{\omega}_N$ produit un piètre estimé pour les phrases avec un prior faible. Or, comme un prior faible leur est associé, le terme d'exploration ne permet pas de les visiter suffisamment souvent pour générer des cibles de meilleure qualité.

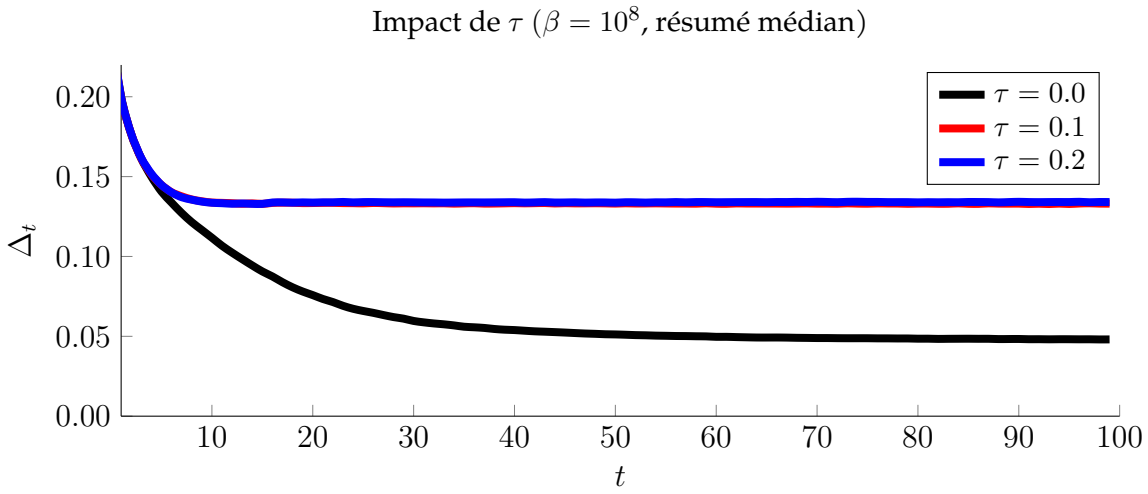

FIGURE 4.3 – Impact de la similarité de distributions a priori avec une distribution uniforme, selon le nombre t de rondes effectuées.

La figure 3.5 présente l'impact de la qualité du prior sur l'évolution de Δ_t. Sans grande surprise, les priors basés sur le pire résumé et le résumé médian génèrent de piètre cibles alors que les priors basés sur le meilleur résumé convergent extrêmement rapidement.

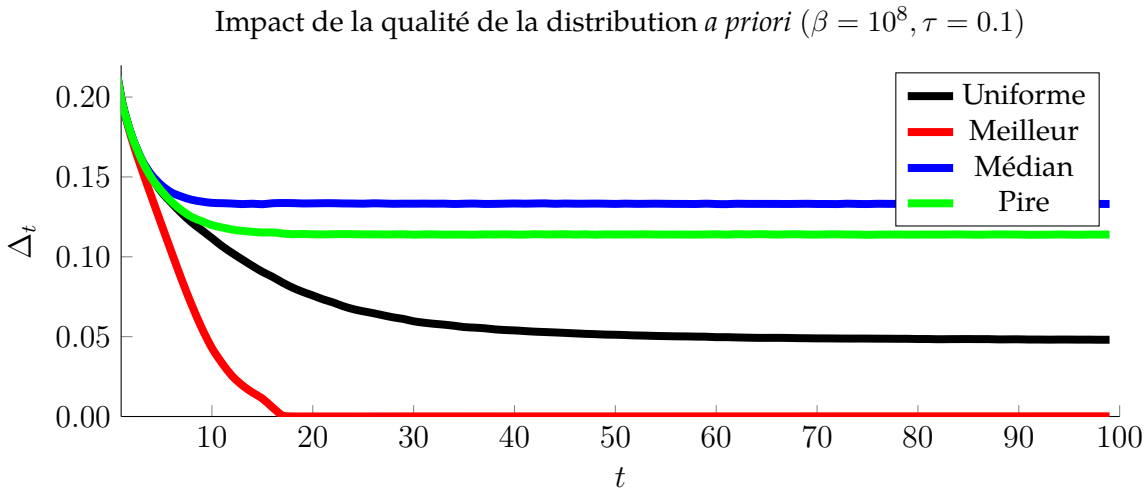

FIGURE 4.4 – Impact de la similarité de distributions à priori avec une distribution uniforme, selon le nombre t de rondes effectuées.

4.3 LinCombiSum

On propose maintenant un système de génération de résumés complet nommé LinCombiSum, qui se veut identique à CombiSum (3.4) mais qui utilise LinUCB au lieu de UCB pour générer ses cibles. Aussi, comme les expériences ont démontré que l'introduction de connaissances a priori ne permet pas d'améliorer les cibles générées par LinUCB, on considère seulement les cibles avec prior uniforme. Pour LinCombiSum, les cibles supervisées utilisées sont donc les valeurs $\langle \hat{\omega}, \tilde{a}_i \rangle$ retournées par LinUCB, que l'on peut naturellement apprendre via une fonction de perte quadratique. À l'inférence, le résumé est encore une fois généré en regroupant les 3 phrases pour lesquelles la prédiction de LinCombiSum est maximale.

4.3.1 Expériences

On roule le système sur le jeu de données CNN/DailyMail. Le pseudocode décrivant le système se trouve dans l'algorithme 6.

Pour les expériences, on teste de faire varier le nombre de rondes de LinUCB utilisées en fonction du nombre de phrases ou de le fixer à $T = 50$ selon les expériences de LinUCB.

4.3.2 Résultats

TODO: Idem à CombiSum

4.4 Conclusion

TODO: Idem à CombiSum

Algorithme 6 LinCombiSum

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**
    $x = h_\theta(d)$
    Obtenir les représentations $\tilde{a}_i$ des phrases de d
    Générer les cibles $\hat{x}_i = \langle \hat{\omega}_T, \tilde{a}_i \rangle$ après T rondes de LinUCB
    $\nabla = \nabla + \nabla_\theta \|x - \hat{x}\|^2$
  $\theta = \theta - \alpha\nabla$

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.

Annexe A

Gains computationnels LinUCB

Cet annexe vise à décrire comment il est possible d'alléger le coût computationnel lié à LinUCB en exploitant la nature de la matrice $\mathbf{V}_t$ construite selon (4.4). Commençons d'abord par énoncer la formule de Sherman-Morrison (Sherman and Morrison, 1950) :

Formule de Sherman-Morrison Soit $\mathbf{M} \in \mathbb{R}^{n \times n}$ une matrice carrée inversible et $u, v \in \mathbb{R}^n$ deux vecteurs. Dans ce cas, $\mathbf{M} + uv^\top$ est inversible si et seulement si $1 + v^\top \mathbf{M} u \neq 0$. On a alors

$$\left(\mathbf{M} + uv^\top\right)^{-1} = \mathbf{M}^{-1} - \frac{\mathbf{M}^{-1} uv^\top \mathbf{M}^{-1}}{1 + v^\top \mathbf{M} u}. \tag{A.1}$$

Pour le calcul de $\mathbf{V}_t$ dans LinUCB, la formule s'applique directement et donne

$$\mathbf{V}_t^{-1} = \left(\mathbf{V}_{t-1} + \tilde{a}_{i_t}^\top \tilde{a}_{i_t}\right)^{-1} = \mathbf{V}_{t-1}^{-1} - \frac{\mathbf{V}_{t-1}^{-1} \tilde{a}_{i_t} \tilde{a}_{i_t}^\top \mathbf{V}_{t-1}^{-1}}{1 + \tilde{a}_{i_t}^\top \mathbf{V}_{t-1} \tilde{a}_{i_t}}, \tag{A.2}$$

un calcul bien plus efficace que d'inverser la matrice $\mathbf{V}_t$ car il ne requiert que des produits matriciels et vectoriels.

On note aussi qu'une légère optimisation peut aussi être faite en constatant que le produit $\mathbf{V}_{t-1}^{-1} \tilde{a}_{i_t}$ est utilisé à plusieurs reprises. On peut donc entreposer le produit en posant

$$W = \mathbf{V}_{t-1}^{-1} \tilde{a}_{i_t}.$$

Enfin, comme la matrice $\mathbf{V}$ sera toujours symétrique car elle est la somme de matrices symétriques, on aura aussi $\mathbf{V}^{-1}$ symétrique. Cette symétrie peut aussi être exploitée pour donner

$$\mathbf{V}_t^{-1} = \mathbf{V}_{t-1}^{-1} - \frac{WW^\top}{1 + \tilde{a}_{i_t}^\top W}, \tag{A.3}$$

qui est la version utilisée dans notre implémentation de LinUCB.

Annexe B

Graphiques avec variance

Commentaire: Je présente ici seulement les graphiques avec variance pour le chapitre sur LinUCB, mais je vais éventuellement avoir tous les graphiques. La grosse variance est surtout dûe au fait que mon jeu de développement a 25 000 documents, avec chacun sa propre distribution des scores.

Graphiques du chapitre 4

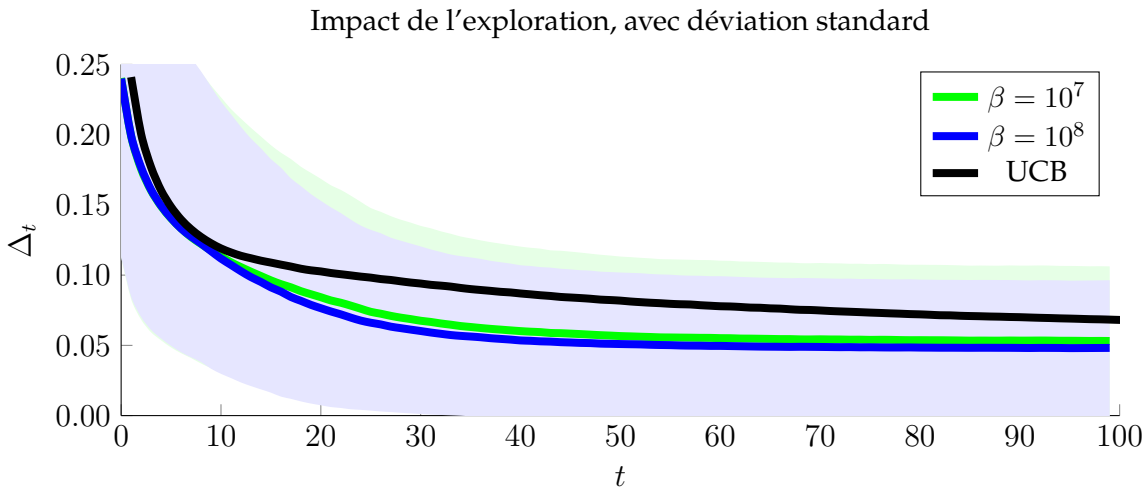

FIGURE B.1 – Impact du paramètre d'exploration β utilisé par LinUCB sur la convergence.

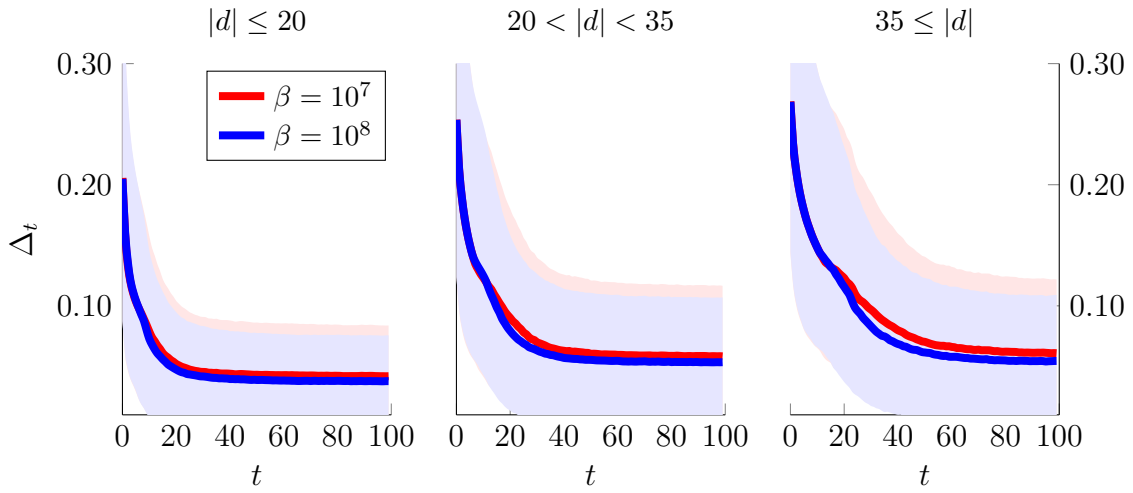

FIGURE B.2 – Impact de la taille du document sur la convergence, avec variance.

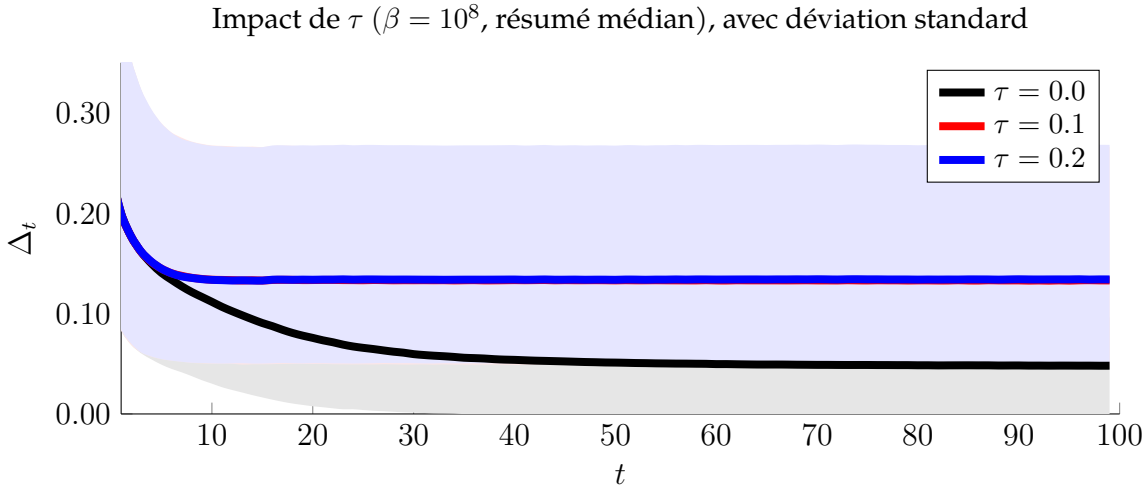

FIGURE B.3 – Impact de la similarité de distributions a priori avec une distribution uniforme, selon le nombre t de rondes effectuées.

Impact de la qualité de la distribution *a priori* ($\beta = 10^8, \tau = 0.1$), avec déviation standard

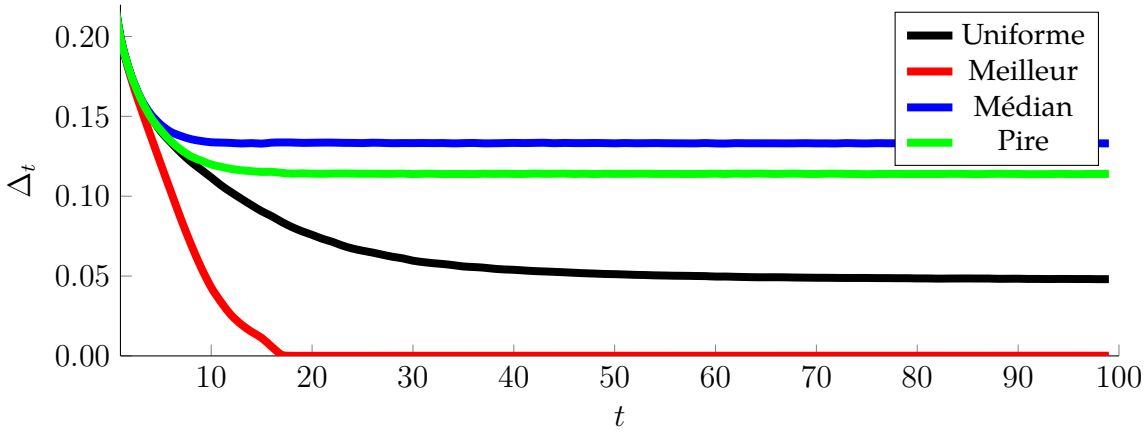

FIGURE B.4 – Impact de la similarité de distributions à priori avec une distribution uniforme, selon le nombre t de rondes effectuées.

Bibliographie

Y. Abbasi-Yadkori, D. Pál, and C. Szepesvári. Improved algorithms for linear stochastic bandits. *Advances in neural information processing systems*, 24, 2011.

A. F. Agarap. Deep learning using rectified linear units (relu), 2018. URL http://arxiv.org/abs/1803.08375.

P. Auer, N. Cesa-Bianchi, and P. Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine learning*, 47(2-3), 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*, 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*, 2018.

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

D. P. Kingma and J. Ba. Adam : A method for stochastic optimization, 2014. URL http://arxiv.org/abs/1412.6980. cite arxiv :1412.6980Comment : Published as a conference paper at the 3rd International Conference for Learning Representations, San Diego, 2015.

T. G. Kolda and D. P. O'Leary. A semidiscrete matrix decomposition for latent semantic indexing information retrieval. *ACM Trans. Inf. Syst.*, 16(4) :322–346, Oct. 1998. ISSN 1046-8188. doi : 10.1145/291128.291131. URL https://doi.org/10.1145/291128.291131.

V. Kuleshov and D. Precup. Algorithms for multi-armed bandit problems. *Journal of Machine Learning Research*, 1, 02 2014.

T. Lattimore and C. Szepesvári. *Bandit Algorithms*. Cambridge University Press, 2020. doi : 10.1017/9781108571401.

L. Li, W. Chu, J. Langford, and R. E. Schapire. A contextual-bandit approach to personalized news article recommendation. *Proceedings of the 19th international conference on World wide web - WWW '10*, 2010. doi : 10.1145/1772690.1772758. URL http://dx.doi.org/10.1145/1772690.1772758.

C.-Y. Lin. ROUGE : A package for automatic evaluation of summaries. In *Text Summarization Branches Out*, 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)*, 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. 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. 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*, 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, 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*, 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), 2020. URL http://jmlr.org/papers/v21/20-074.html.

M. Schuster and K. K. Paliwal. Bidirectional recurrent neural networks. *IEEE transactions on Signal Processing*, 45(11) :2673–2681, 1997.

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. AAAI Press, 2015. ISBN 9781577357384.

J. Sherman and W. J. Morrison. Adjustment of an inverse matrix corresponding to a change in one element of a given matrix. *The Annals of Mathematical Statistics*, 21(1) :124–127, 1950.

D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis. Mastering the game of go with deep neural networks and tree search. *Nature*, 529, 2016. URL http://www.nature.com/nature/journal/v529/n7587/full/nature16961.html.

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, 1999.

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

R. J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4), 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. Liu. PEGASUS : Pre-training with extracted gap-sentences for abstractive summarization. In H. D. III and A. Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, Virtual, 13–18 Jul 2020. PMLR. URL http://proceedings.mlr.press/v119/zhang20ae.html.

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*, 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.