Approches par bandit pour la génération automatique de résumés de textes

Mémoire

Mathieu Godbout

Sous la direction de:

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

Résumé

Ce mémoire aborde l'utilisation des méthodes par bandit pour résoudre la problématique de l'entraînement de modèles de générations de résumés extractifs. Les modèles extractifs, qui bâtissent des résumés en sélectionnant des phrases d'un document original, sont difficiles à entraîner car le résumé cible correspondant à un document n'est habituellement pas constitué de manière extractive. C'est à cet effet que l'on propose de voir la production de résumés extractifs comme différents problèmes de bandit, lesquels sont accompagnés d'algorithmes pouvant être utilisés pour l'entraînement.

On commence ce document en présentant BanditSum, une approche tirée de la litérature et qui voit la génération des résumés d'un ensemble de documents comme un problème de bandit contextuel. Ensuite, on introduit CombiSum, un nouvel algorithme qui formule la génération du résumé d'un seul document comme un bandit combinatoire. En exploitant la formule combinatoire, CombiSum réussit à incorporer la notion du potentiel extractif de chaque phrase à son entraînement. Enfin, on propose LinCombiSum, la variante linéaire de CombiSum qui exploite les similarités entre les phrases d'un document et emploie plutôt la formulation en bandit linéaire combinatoire.

Abstract

This thesis discusses the use of bandit methods to solve the problem of training extractive abstract generation models. The extractive models, which build summaries by selecting sentences from an original document, are difficult to train because the target summary of a document is usually not built in an extractive way. It is for this purpose that we propose to see the production of extractive summaries as different bandit problems, for which there exist algorithms that can be leveraged for training summarization models.

In this paper, BanditSum is first presented, an approach drawn from the literature that sees the generation of the summaries of a set of documents as a contextual bandit problem. Next, we introduce CombiSum, a new algorithm which formulates the generation of the summary of a single document as a combinatorial bandit. By exploiting the combinatorial formulation, CombiSum manages to incorporate the notion of the extractive potential of each sentence of a document in its training. Finally, we propose LinCombiSum, the linear variant of CombiSum which exploits the similarities between sentences in a document and uses the linear combinatorial bandit formulation instead.

Table des matières

Liste des tableaux

Liste des figures

Liste des algorithmes

À Samantha.

To avoid situations in which you might make mistakes may be the biggest mistake of all.

Peter McWilliams

Remerciements

Je tiens à remercier mon directeur de recherche, Luc Lamontagne, pour ses précieux conseils tout au long de mon parcours à la maîtrise. Sa bienveillance et sa franchise m'auront permis de progresser non seulement en tant que chercheur, mais aussi en tant que personne. Un merci tout particulier à Audrey Durand, ma co-directrice, pour m'avoir initié au passionnant domaine des bandits et de l'apprentissage par renforcement. À travers nos interactions, j'ai découvert tout le plaisir qui se cache derrière le processus scientifique bien fait et je suis impatient de continuer notre collaboration. Par leurs efforts conjoints, Luc et Audrey m'ont permis de me dépasser et de bâtir mon propre bagage scientifique.

Je ne peux passer sous le silence l'incroyable communauté de recherche du Groupe de Recherche en Apprentissage Automatique de l'université Laval (GRAAL) dans laquelle j'ai eu la chance de compléter ma maîtrise. Tantôt collègues de travail, tantôt partenaires de célébrations, les personnes que j'ai rencontrées au GRAAL ont grandement contribué à faire de ma maîtrise une période de ma vie que je chérirai toujours. Merci à David, Jean-Thomas, Nicolas, Jean-Samuel, Frédérik, Gaël et tous les autres.

Un merci spécial va aussi à mes parents, Martine et Alain, qui, avec leur éternelle confiance en mes capacités et leur dévouement à mon épanouissement ont rendu possible mon cheminement scolaire et personnel. Je conserve le dernier de mes remerciements pour Samantha, sans qui je ne serais pas l'ombre de la personne que je suis aujourd'hui. Partenaire de mes joies comme de mes peines, elle a toujours su faire ressortir la meilleure version de moi-même. Par son écoute, ses questions intéressées et son soutien incomparable, elle a certainement participé à l'écriture de ce mémoire dans une bien plus grande proportion qu'elle ne peut l'imaginer.

Je remercie aussi le Conseil de Recherches en Sciences Naturelles et en Génie du Canada (CRSNG), le Fonds de recherche du Québec — Nature et Technologies (FRQ-NT) et Intact Corporation financière pour leur soutien financier ayant permis l'élaboration de ce document.

Introduction

Lors d'une entrevue en 2018, le président de la multinationale Siemens mentionnait que[1] "Les données sont le nouveau pétrole, ou or, du 21ème siècle - la matière première sur laquelle nos économies, sociétés et démocraties sont de plus en plus bâties." 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. Une variante particulièrement intéressante de la génération de résumé est la variante qui consiste à bâtir un résumé en sélectionnant des phrases du texte original. Les systèmes de génération de ces résumés, dits résumés extractifs, sont le sujet d'étude principal du présent document.

1. Traduction libre.

Plus particulièrement, on s'intéresse à la formulation de la génération de résumé comme un problème de bandit (Robbins, 1952). Les problèmes de bandit sont des problèmes d'interaction séquentielle entre un apprenant et un environnement, où l'apprenant sélectionne des actions dans le but de maximiser les récompenses qui leur sont assignées par l'environnement. La résolution de tout problème de bandit est basée sur l'identification rapide par l'agent d'actions près de l'action optimale sur l'environnement. En voyant la génération de résumé comme un bandit, on souhaite ainsi bâtir des systèmes de génération de résumés qui apprendront rapidement à générer des résumés de haute qualité grâce à une exploration efficace du vaste espace de résumés possibles.

Objectifs

Pour analyser la viabilité des approches par bandit pour l'entraînement de modèles de génération de résumés, ce document souhaite accomplir les objectifs suivants :

— Présenter BanditSum (Dong et al., 2018), une approche modélisant la génération de résumés pour un ensemble de documents comme un problème de bandit contextuel (Auer, 2002). Avec la formulation contextuelle retenue, BanditSum parvient à entraîner un système de génération de résumés en optimisant l'espérance de sa performance sur un ensemble de documents.

— Aborder la génération du résumé d'un seul document comme un problème de bandit combinatoire (Chen et al., 2013). Particulièrement, on s'intéresse à comment la formulation en bandit combinatoire peut être utilisée pour entraîner un système de génération résumés.

— Exploiter les similarités entre les phrases d'un même document pour formuler la génération du résumé d'un document comme un bandit linéaire combinatoire (Chen et al., 2018). Spécifiquement, c'est l'application de cette formulation linéaire à un algorithme d'entraînement de modèle de génération et son impact sur la rapidité de convergence auxquelles on s'intéresse.

— Comparer la performance des approches par bandit aux modèles issus de l'état de l'art et à l'heuristique de référence Lead-3 (Nallapati et al., 2017) sur le jeu de données du CNN/DailyMail (Hermann et al., 2015). Notamment, on désire savoir si les approches par bandit permettent un entraînement plus rapide en nombre de documents et produisent des modèles qui génèrent des résumés de meilleure qualité.

Structure du mémoire

Ce mémoire débute par une introduction aux concepts de base essentiels à la compréhension du reste du document au chapitre 1. S'ensuit au chapitre 2 une description détaillée de la pro-

blématique à l'étude ainsi que la présentation d'un algorithme basé sur le bandit contextuel : BanditSum. Au chapitre 3, la formulation en bandit combinatoire est employée afin d'obtenir des cibles riches pour l'entraînement de modèles de génération de résumé. Enfin, le chapitre 4 explore l'utilisation du bandit linéaire combinatoire pour générer des cibles riches encore une fois, mais qui exploitent les similarités entre les phrases d'un document pour une plus grande efficacité.

Chapitre 1

Concepts de base pertinents

Ce chapitre vise à présenter les concepts de base clés à la compréhension des travaux présentés du reste de ce document. Les concepts abordés ici sont accompagnés d'une explication aussi dénuée que possible de notions non-nécessaires aux travaux de ce mémoire, afin de garder leur présentation digeste. À cette fin, les concepts des problèmes 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.

1.1 Approches par bandit

On définit un bandit (Robbins, 1952) comme étant un problème où un apprenant interagit avec un environnement de manière séquentielle. À chaque pas de temps t, l'apprenant sélectionne une action $a_t \in \mathcal{A}$ à effectuer parmi un ensemble $\mathcal{A}$ d'actions. Pour chaque action a_t sélectionnée, une récompense associée $X_{a_t,t}$ est observée. Le but de l'apprenant est alors de maximiser les récompenses obtenues sur un horizon fini d'interactions avec l'environnement. Pour mesurer la performance d'un apprenant, on utilise une métrique appelée le regret cumulatif, qui mesure la différence de récompense entre l'action maximisant la récompense sur l'environnement et les actions choisies par l'apprenant. Formellement, les algorithmes qui s'attaquent à la résolution d'un problème de bandits ont l'objectif fondamental de minimiser le regret cumulatif.

L'objectif de minimisation du regret cumulatif fait intervenir un principe fondamental dans les problèmes où un apprenant interagit avec un environnement : le dilemme entre l'exploration et l'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. Pour ce faire, il doit s'assurer que l'action sélectionnée à chaque pas de temps **exploite** les informations perçues par le passé mais **explore** aussi suffisamment les actions susceptibles d'être meilleures.

Notons que l'objectif de minimisation du regret cumulatif est différent des approches d'ap-

prentissage 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, cet objectif 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).

On présente maintenant le bandit stochastique et sa déclinaison linéaire, qui représentent les versions du problème bandit que l'on applique tout au long du document. Pour chacune des versions, un algorithme permettant accomplissant la minimisation du regret associé aussi présenté.

1.1.1 Bandit stochastique

Un bandit stochastique (Lai and Robbins, 1985) est un problème pour lequel on a un ensemble $\mathcal{A}$ de K actions disponibles. Dans le contexte stochastique, il est supposé que les récompenses $X_{a,t}$ d'une action sont indépendantes et identiquement distribuées (i.i.d.) selon une espérance fixe et inconnue de l'apprenant μ_a, i.e. $\mu_a = \mathbb{E}[X_{a,t}]$. Dans ce contexte, le but de l'apprenant est de maximiser l'**espérance** de ses récompenses. On définit alors $a^* = \arg\max_{a \in \mathcal{A}} \mu_a$ et $\mu^* = \mu_{a^*}$, respectivement l'action et l'espérance de récompense optimales. Afin de mesurer la performance d'un apprenant sur un bandit stochastique, la métrique de performance utilisée est le pseudo-regret cumulatif après T interactions

$$R_T = T\mu^* - \sum_{t=1}^{T} \mu_{a_t}. \tag{1.1}$$

Ici, R_T représente la différence entre les récompenses attendues en sélectionnant l'action optimale et les actions sélectionnées par l'apprenant. Il s'agit donc d'une mesure de regret, mais appliquée sur l'espérance (inconnue) de récompense de chaque action, d'où le nom pseudo-regret.

1.1.2 Upper Confidence Bound (UCB)

L'algorithme UCB (Auer et al., 2002) est un algorithme de minimisation du pseudo-regret cumulatif (1.1) pour les bandits stochastiques. UCB est basé sur le principe de l'optimisme face à l'incertitude (OFUL), qui permet de résoudre de manière élégante le dilemme exploration-exploitation décrit à la section 1.1. Pour suivre le principe OFUL, UCB maintient des bornes supérieures UCB_a sur l'espérance de chaque action a. Les bornes UCB_a sont construites de sorte à être supérieures à la valeur μ_a espérée pour l'action avec une grande probabilité. Après t pas de temps effectués, la borne supérieure pour l'espérance de l'action a est définie selon

$$\text{UCB}_a(t) = \bar{x}_a(t) + \beta\sqrt{\frac{2\ln t}{n_a(t)}}, \tag{1.2}$$

où β est un hyperparamètre régulant l'exploration et

$$\bar{x}_a(t) = \frac{1}{n_a(t)}\sum_{\tau=1}^{t}\mathbb{I}\left[a_\tau = a\right]X_{a_\tau,\tau} \quad \text{et} \quad n_a(t) = \sum_{\tau=1}^{t}\mathbb{I}\left[a_\tau = a\right]$$

représentent respectivement la moyenne des récompenses reçues jusqu'au temps t pour l'action a et le nombre de fois que l'action a a été sélectionnée. Dans les équations de $\bar{x}_a(t)$ et $n_a(t)$, $\mathbb{I}$ est la fonction indicatrice retournant 1 quand son prédicat est vrai et 0 sinon. Quand une action a n'a pas encore été sélectionnée ($n_a(t) = 0$), on lui assigne $\text{UCB}_a(t) = \infty$, forçant l'apprenant à sélectionner au moins une fois chaque action avant d'exploiter les moyennes $\bar{x}_a(t)$ dans son choix d'action.

La construction des bornes supérieures UCB_a incorpore de manière explicite le dilemme entre l'exploration et l'exploitation. En effet, le terme $\bar{x}_a(t)$ à gauche présente l'estimation actuelle de la récompense associée à l'action a et permet donc d'exploiter les connaissances acquises dans les pas de temps précédant t. À droite, le terme $\sqrt{\frac{2\ln t}{n_a(t)}}$ est proportionnel à l'inverse des sélections relatives. Il s'agit donc d'un terme qui sera plus élevé pour les actions moins choisies, incitant l'exploration d'actions sous-exploitées. Ainsi, en sélectionnant l'action a_t avec la borne supérieure maximale selon les observations précédentes

$$a_t = \arg\max_{a\in\mathcal{A}} \text{UCB}_a(t-1),$$

l'algorithme UCB résout directement le dilemme exploitation-exploration. Lorsque plusieurs actions ont la même valeur de borne supérieure, les égalités sont brisées de manière aléatoire.

1.1.3 Bandit stochastique linéaire

La formulation en bandit stochastique linéaire reprend exactement le même cadre formel que le bandit stochastique. Les seules nouvelles notions sont la présence de représentations vectorielles $\tilde{a} \in \tilde{\mathcal{A}} \subseteq \mathbb{R}^n$ pour les actions du problème et l'introduction de l'hypothèse linéaire. Cette dernière suppose qu'il existe un vecteur de récompense unique ω_* permettant de relier chaque action $\tilde{a}$ à sa récompense espérée μ_a, i.e. $\langle\omega_*, \tilde{a}\rangle \approx \mu_a$ pour tout a. Sous cette formulation, le pseudo-regret cumulatif après T interactions devient

$$R_T = T\langle\omega_*, \tilde{a}^*\rangle - \sum_{t=1}^{T}\langle\omega_*,\tilde{a}_t\rangle = \sum_{t=1}^{T}\langle\omega_*, \tilde{a}^* - \tilde{a}_t\rangle, \tag{1.3}$$

pour $\tilde{a}^*$, l'action avec l'espérance μ_a la plus élevée, dite action optimale.

1.1.4 Linear Upper Confidence Bound (LinUCB)

L'algorithme LinUCB (Chu et al., 2011) représente une application du principe de l'optimisme face à l'incertitude (OFUL) sur le problème de bandit stochastique linéaire pour la minimisation du pseudo-regret cumulatif (1.3).

Rappelons que le principe OFUL consiste à avoir, pour chaque action a, un estimé optimiste UCB_a de sa valeur espérée μ_a, lequel est plus élevé que μ_a avec forte probabilité. Dans le contexte linéaire, on cherche à avoir UCB_a qui fait usage des relations linéaires entre les actions pour accélérer l'apprentissage. On présente plus bas l'intuition derrière la construction de bornes supérieures UCB_a 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 les bornes supérieures consiste à exploiter les observations faites avant le temps t pour bâtir un estimé $\hat{\omega}_t$ de ω_*. Cet estimé devrait naturellement viser à combiner aussi bien que possible les paires actions-récompenses $(\tilde{a}_\tau, X_\tau)$ observées, i.e.

$$\langle \hat{\omega}_t, \tilde{a}_\tau \rangle \approx X_\tau, \quad \text{pour} \quad \tau \in \{1, ..., t\}. \tag{1.4}$$

On peut voir (1.4) comme un problème de moindres carrés, où l'on souhaite trouver la solution

$$\hat{\omega}_t = \underset{\omega \in \mathbb{R}^n}{\arg\min} \sum_{\tau=1}^{t} \left(\langle \omega, \tilde{a}_\tau \rangle - X_\tau \right)^2. \tag{1.5}$$

Or, le problème (1.5) 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. L'estimé $\hat{\omega}_t$ est alors choisi selon la solution analytique connue du problème

$$\hat{\omega}_t = {\mathbf{V_t}}^{-1} \mathbf{b_t}, \quad \text{pour} \quad \mathbf{V_t} = (\mathbf{A_t}^\top \mathbf{A_t} + \lambda \mathbf{I}) \quad \text{et} \quad \mathbf{b_t} = \mathbf{A_t}^\top \mathbf{X_t}, \tag{1.6}$$

où $\mathbf{A_t}$ est la matrice dont les lignes sont les actions $\tilde{a}_\tau$ sélectionnées jusqu'à maintenant, $\mathbf{X_t}$ est le vecteur composé des récompenses X_τ associées et $\mathbf{I}$ est la matrice identité. En pratique, $\mathbf{V_t}$ et $\mathbf{b_t}$ peuvent tous deux être calculés de manière itérative selon

$$\mathbf{V_t} = \lambda \mathbf{I} + \sum_{\tau=1}^{t} \tilde{a}_\tau \tilde{a}_\tau^\top \quad \text{et} \quad \mathbf{b_t} = \sum_{\tau=1}^{t} \tilde{a}_\tau X_\tau. \tag{1.7}$$

Maintenant que l'on possède une bonne estimation $\hat{\omega}_t$ exploitant l'expérience récoltée, on cherche à introduire un terme garantissant une exploration satisfaisante des actions possibles. LinUCB propose à cet effet d'utiliser

$$\beta\sqrt{\tilde{a}^\top \mathbf{V_t}^{-1}\tilde{a}}, \tag{1.8}$$

un terme qui base l'exploration liée à une action a à la différence de sa représentation vectorielle $\tilde{a}$ par rapport aux actions précédentes représentées par $\mathbf{V_t}$, que l'on pondère encore une fois avec 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).

À partir de (1.6) et (1.8), on peut encore une fois bâtir une borne supérieure incorporant directement le compromis exploration-exploitation

$$\text{UCB}_a(t) = \langle \hat{\omega}_t^\top, \tilde{a}\rangle + \beta\sqrt{\tilde{a}^\top \mathbf{V_t}^{-1}\tilde{a}}, \tag{1.9}$$

à partir de laquelle LinUCB choisit la borne supérieure maximale

$$a_t = \arg\max_{a\in\mathcal{A}} \text{UCB}_a(t-1).$$

Connexions avec UCB

On prend 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 (Lattimore and Szepesvári, 2020). En effet, si l'on considère les vecteurs d'action correspondant aux vecteurs unitaires orthogonaux $\tilde{a} = e_a$ et une régularisation de Tikhonov avec $\lambda = 0$, on a que $\mathbf{V_t}$ est une matrice diagonale, où l'entrée à la position a est le nombre $n_a(t)$ de sélections de l'action a. À ce moment, le terme $e_a^\top \mathbf{V_t}^{-1} e_a$ n'est donc que $\frac{1}{n_a(t)}$ et le terme d'exploration devient

$$\beta\sqrt{\tilde{a}^\top \mathbf{V_t}^{-1}\tilde{a}} = \beta\sqrt{\frac{1}{n_a(t)}},$$

un terme proportionnel à celui utilisé dans UCB (1.2) si l'on pose $\beta' = \beta\sqrt{2\ln(t)}$.

Aussi, $\hat{\omega}_t$ devient simplement le vecteur où l'entrée a correspond à la moyenne des récompenses perçues pour l'action a. Le produit vectoriel $\langle \hat{\omega}_t, e_a\rangle$ ne fait donc qu'aller chercher la moyenne $\bar{x}_a(t)$ utilisée dans UCB. Ainsi, si l'on se positionne dans le cas où aucune information n'est partagée par les vecteurs d'action $\tilde{a}$, LinUCB est équivalent à UCB. Comme cela correspond au pire cas pour LinUCB, celui où aucune relation linéaire n'est exploitable, LinUCB obtiendra toujours une performance supérieure ou égale à UCB dans les cas où l'hypothèse

linéaire est respectée.

1.2 Réseaux de neurones

L'apprentissage supervisé est un problème dans lequel on tente d'apprendre la relation entre des données en entrée et une valeur en sortie qui leur est associée. Concrètement, on dit que l'on dispose d'un jeu de données $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^{N}$ pour lequel on souhaite apprendre une fonction f reliant chaque entrée à sa sortie, i.e. $f(x_i) \approx y_i$. Cette formulation est générale et polyvalente : elle peut être appliquée à des tâches très variées allant de la classification d'images à la traduction de textes.

En pratique, on résout habituellement un problème d'apprentissage supervisé en commençant par sélectionner une classe de fonctions paramétrées $\mathcal{F}$. On tente alors de trouver les paramètres θ^* produisant l'approximateur paramétré $f_{\theta^*} \in \mathcal{F}$ reliant le mieux chaque entrée à sa sortie pour le problème. Les réseaux de neurones (RN) (Goodfellow et al., 2016) forment l'une de ces classes de fonctions paramétrées particulièrement intéressante. Leur utilisation a récemment explosé suite à leur excellente performance empirique et leur application à vraisemblablement n'importe quel problème supervisé (Liu et al., 2017).

Les RNs ne sont utilisés dans ce document que pour permettre l'application des contributions. On se limite donc à présenter le processus de descente de gradient que les RNs utilisent ainsi que les cas d'utilisation pertinents aux travaux.

1.2.1 Descente de gradient

Pour trouver de bons paramètres θ, les réseaux de neurones emploient une fonction de perte l qui mesure la proximité entre une valeur prédite $\hat{y} = f_\theta(x)$ et la cible y pour un exemple (x,y) du jeu de données. Hors mis la nécessité de représenter la proximité entre $\hat{y}$ et y, l est seulement soumise à la contrainte d'être dérivable, i.e. le gradient $\nabla_{\hat{y}} l(\hat{y}, y)$ doit être bien défini. Par exemple, pour des problèmes de classification binaire où les cibles y et les valeurs prédites $\hat{y}$ sont des nombres entre 0 et 1, une fonction de perte utilisable est l'entropie croisée binaire (Rubinstein and Kroese, 2013)

$$l(\hat{y}, y) = y \ln(\hat{y}) + (1 - y) \ln(1 - \hat{y}).$$

L'objectif de l'entraînement d'un RN est alors de trouver la paramétrisation θ^* minimisant la perte espérée, c'est à dire obtenir

$$\theta^* = \arg\min_\theta \mathcal{L}(\theta), \quad \text{où} \quad \mathcal{L}(\theta) = \mathop{\mathbb{E}}_{(x,y)\sim\mathcal{D}} l\left(f_\theta(x), y\right). \tag{1.10}$$

On nomme $\mathcal{L}$ la perte espérée, en raison de l'espérance faite sur les exemples du jeu de données. En pratique, avec n'importe quel jeu de données de taille raisonnable, cette espérance ne peut toutefois pas être calculée efficacement et on utilise plutôt son estimation basée sur une *minibatch* de B exemples pigés aléatoirement

$$\bar{\mathcal{L}}(\theta) = \frac{1}{B}\sum_{i=1}^{B} l\left(f_\theta(x_i), y_i\right), \quad \text{pour} \quad \{(x_i, y_i)\}_{i=1}^{B} \sim \mathcal{D}^B, \tag{1.11}$$

appelée la perte empirique. Naturellement, plus la taille B de la *minibatch* est importante, meilleure est l'estimation de la perte espérée et c'est pour cela que l'on choisit habituellement B aussi grand que possible selon les ressources de calcul à disposition.

Le problème d'optimisation (1.10) peut être résolu en utilisant la perte empirique (1.11) en appliquant une descente de gradient. En effet, comme on le verra bientôt, un réseau de neurones est entièrement dérivable par rapport à ses paramètres θ et on peut donc optimiser les paramètres selon la règle de mise à jour

$$\theta = \theta - \alpha \nabla \bar{\mathcal{L}}(\theta), \tag{1.12}$$

où α est un hyperparamètre appelé taux d'apprentissage. En pratique, on utilise des optimiseurs, notamment SGD (Robbins and Monro, 1951) et Adam (Kingma and Ba, 2014), qui implémentent des variantes de l'équation (1.12) pour trouver les paramètres θ^*.

1.2.2 Réseaux pleinement connectés

Un réseau pleinement connecté est la forme la plus simple de réseau de neurones. Au niveau le plus fondamental, un réseau pleinement connecté est composé d'un ensemble de couches c_i consécutives. Chaque couche est constituée d'une matrice $\mathbf{W}_i$ de poids et d'une fonction non-linéaire d'activation σ_i. Pour un vecteur h_i en entrée, une couche c_i produit en sortie un autre vecteur h_{i+1} qui sert d'entrée à la prochaine couche selon

$$h_{i+1} = \sigma_i\left(\mathbf{W}_i h_i\right),$$

où $\mathbf{W}_i h_i$ est un produit matriciel et σ_i est appliquée sur toutes les composantes du vecteur $\mathbf{W}_i h_i$. Les seuls cas particuliers sont ceux de la première couche c_1, qui prend le vecteur $h_1 = x$ comme entrée, et de la dernière couche c_n qui génère la valeur prédite $h_{n+1} = \hat{y}$. Enfin, on définit les paramètres θ d'un RN comme l'ensemble des matrices $\mathbf{W}_i$ qui le composent.

Pour être admissible, une fonction d'activation σ doit seulement être non-linéaire et dérivable. Cela fait en sorte que les fonctions d'activation sont nombreuses dans la litérature. Celles qui

sont utilisées dans ce document sont :

— ReLU (Xu et al., 2015a) : $\sigma(z) = \max(0, z)$, dont la sortie est toujours positive et non bornée.

— tanh : $\sigma(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$, dont la sortie est entre -1 et 1.

— sigmoïde : $\sigma(z) = \frac{1}{1+e^{-z}}$, dont la sortie est entre 0 et 1.

Le choix de la fonction d'activation utilisée à chaque couche dépend du comportement souhaité. On utilise habituellement les fonctions ReLU et tanh pour les transitions entre deux couches consécutives alors que la fonction sigmoïde est habituellement employée en sortie quand les cibles sont entre 0 et 1.

1.2.3 Réseaux récurrents

Les réseaux pleinement connectés présentés à la section 1.2.2 ne permettent pas d'exploiter la nature séquentielle de certains problèmes comme ceux basés sur des données textuelles. En effet, pour ces problèmes où l'entrée x et la sortie y associée sont des séquences $[x_1, ..., x_n]$ et $[y_1, ..., y_n]$, les réseaux pleinement connectés ne peuvent pas utiliser les potentielles relations entre les éléments d'une même séquence. C'est à cet effet qu'ont été proposés les réseaux récurrents (RNN) (Schuster and Paliwal, 1997).

Sous sa forme la plus simple, un RNN est représenté par trois matrices de poids $\mathbf{U}$, $\mathbf{V}$ et $\mathbf{W}$ accompagnées de deux fonctions d'activation σ_h et σ_o. Pour chaque élément x_i et représentation cachée h_i en entrées, le RNN produit un vecteur en sortie o_i et une représentation cachée h_{i+1} qui sera utilisée pour le prochain élément selon

$$h_{i+1} = \sigma_h\left(\mathbf{U}x_i + \mathbf{V}h_i\right) \quad \text{et} \quad o_i = \sigma_o\left(\mathbf{W}h_i\right).$$

L'idée d'un réseau récurrent est donc de générer une représentation cachée h qui est utilisée et incrémentée à chaque nouvel élément x_i de la séquence x en entrée.

En pratique, les réseaux récurrents peuvent être bidirectionnels en incorporant trois nouvelles matrices $\mathbf{U}'$, $\mathbf{V}'$ et $\mathbf{W}'$ utilisées pour parcourir la séquence x dans le sens contraire (de x_n à x_1). Il est aussi possible d'empiler plusieurs RNNs, en utilisant les sorties o_i produites par un RNN comme séquence en entrée à un prochain RNN. On dit alors que l'on a un réseau récurrent à n couches, pour le nombre n de RNNs empilés. Une illustration contenant un réseau récurrent bidirectionnel à deux couches est disponible à la figure 2.1.

Plus récemment, plusieurs des bons résultats empiriques obtenus avec des réseaux récurrents ont été obtenus en utilisant une variante plus complexe que celle présentée : le LSTM (Hochreiter and Schmidhuber, 1997). Les LSTMs incorporent explicitement dans leur architecture une représentation cachée plus riche permettant une meilleure circulation de l'information dans une séquence et un entraînement plus facile. Ils ont notamment été appliqués avec

succès sur la traduction de phrase (Wu et al., 2016) et la génération de description d'image (Xu et al., 2015b).

1.2.4 Plongements de mots

Un cas d'utilisation particulièrement intéressant des réseaux de neurones est celui de la génération de plongements de mots (Mikolov et al., 2013; Pennington et al., 2014; Joulin et al., 2016). En entraînant un RN à prédire les mots dans une phrase, il est possible d'obtenir des représentations vectorielles de mots, communément appelés plongements de mots.

Les plongements incorporent des relations présentes entre les mots qu'ils représentent. Par exemple, si l'on définit $p(m)$ comme le plongement associé au mot m, Mikolov et al. (2013) démontrent que la relation

$$p(\text{"King"}) - p(\text{"Man"}) + p(\text{"Woman"}) = p(\text{"Queen"})$$

est respectée par les plongements de mots générés par leur technique. La richesse des relations incorporées dans les plongements de mots est un des principaux facteurs de l'explosion récente de la performance des RNs sur les tâches de traitement de la langue naturelle (Almeida and Xexéo, 2019).

1.3 Génération automatique de résumés

Le sujet principal d'étude de ce document est la génération automatique de résumés de textes. Formellement, un résumé est une version allégée d'un ou de plusieurs documents originaux. Intuitivement, on souhaite qu'un résumé soit aussi court que possible mais qu'il conserve la majeure partie de l'information originale présente.

Pour notre part, on s'intéresse seulement au cas de la génération de résumés à partir d'un seul document d pour lequel on possède un seul résumé cible s. Des approches existent spécifiquement pour la génération d'un résumé à partir d'un ensemble de documents en entrée, mais on se contente de dire que ca cas (multi-documents) se ramène grossièrement au cas de la génération d'un seul document en considérant la concaténation de l'ensemble de documents.

Sous l'optique à laquelle on s'intéresse, la litérature peut actuellement être séparée en deux formulations distinctes : extractive et abstractive. On présente chacune de ces méthodes, apportant une attention particulière à la formulation extractive qui fait l'objet d'une investigation détaillée dans ce document. On en profite aussi pour bien définir comment il est possible d'évaluer la performance d'un système de génération de résumés en fin de section.

1.3.1 Formulation extractive

Dans la formulation extractive, un résumé est généré en sélectionnant des phrases du document initial. Cette formulation est intéressante car elle permet de (1) réduire drastiquement le nombre de résumés possibles, le faisant seulement dépendre du nombre de phrases d'un document, et (2) d'éviter toutes les difficultés en termes de syntaxe et de cohérence liées à avoir à générer du texte de manière automatique. Actuellement, les approches extractives de l'état de l'art sont toutes basées sur l'obtention d'affinités pour chaque phrase, indiquant si elle devrait être inclue ou non dans un résumé extractif.

Pour formaliser un peu les choses, on dit que l'on dispose de paires (d,s), représentant un document $d = [d_1, ..., d_{|d|}]$ de $|d|$ phrases et son résumé cible s. Un modèle de génération de résumés extractifs est un système à deux composantes (π, ϕ). Ici, π prend en entrée un document d et retourne des affinités $\pi(d) \in [0,1]^{|d|}$ représentant à quel point chaque phrase devrait faire partie d'un résumé extractif. On a ensuite ϕ, qui est un processus de génération de résumé extractif à partir des affinités $\pi(d)$. Formellement, en définissant $\mathscr{P}(d)$ comme étant l'ensemble puissance 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 selon $\pi(d)$, respectivement. Pour référence future, posons ψ comme étant le processus vorace et ξ le processus stochastique. Une illustration du processus général de génération de résumé extractif présenté est disponible à la figure 1.1.

Enfin, pour un problème de génération de résumés donné, 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) à partir des paires document-résumé cibles (d, s) contenues dans un jeu de données $\mathcal{D}$.

Approches supervisées

La majorité des approches extractives de l'état de l'art sont entraînées de manière supervisée à partir de cibles pour les affinités d'un document. Or, comme le résumé s correspondant à un document d n'est habituellement pas obtenu de manière extractive (i.e. le résumé n'est pas une combinaison de phrases du document initial), il n'est pas trivial d'obtenir des cibles pour les affinités. En pratique, les approches supervisées doivent utiliser 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 représentent les phrases choisies par un oracle sélectionnant de manière vorace les phrases $d_i \in d$ 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 d'un document d est fixée à un vecteur binaire

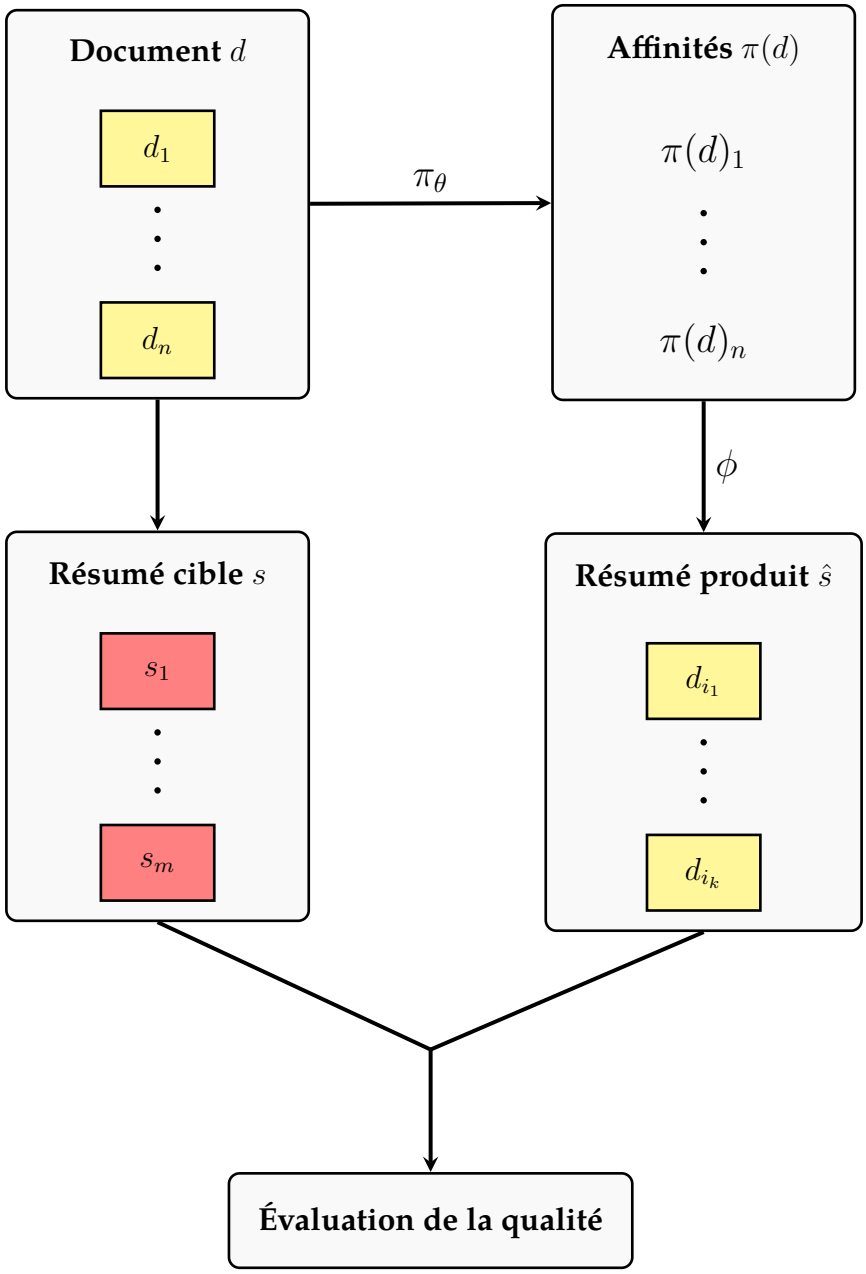

FIGURE 1.1 – Schéma des étapes de la génération de résumé extractif pour un document d et son résumé cible s. La couleur différente utilisée pour les phrases du résumé cible sert à indiquer qu'elles ne sont habituellement pas des phrases du document en entrée.

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 : Liu and Lapata (2019) utilisent seulement les cibles binaires et représentent actuellement l'état de l'art ex-

tractif sur le populaire jeu de données du CNN/DailyMail (Hermann et al., 2015). Notons toutefois qu'un réseau entraîné de manière supervisée à imiter un oracle ne peut pas obtenir une performance supérieure à celle de l'oracle. Ainsi, les approches supervisées sont plus difficilement applicables à des contextes où un bon oracle n'est pas disponible.

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 $G(\hat{s}, s)$ 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

$$J(\theta) = \mathop{\mathbb{E}}_{(d,s)\sim\mathcal{D}} \mathop{\mathbb{E}}_{\hat{s}\sim\phi(\pi_\theta(d))} \left[G(\hat{s}, s)\right], \tag{1.13}$$

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[1]. 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 $J(\theta)$ basée sur un seul échantillon de J

$$\nabla J(\theta) = G(\hat{s}, s)\nabla \ln \phi(\hat{s}|\pi_\theta, d), \quad d \sim \mathcal{D}, \hat{s} \sim \phi(\pi_\theta(d)) \tag{1.14}$$

où $\phi(\hat{s}|\pi_\theta, d)$ représente la probabilité de générer le résumé $\hat{s}$ à partir de $\pi_\theta(d)$ et ϕ.

Le théorème du gradient de politique (Sutton et al., 2000) affirme que, en espérance, l'approximation (1.14) correspond au gradient de (1.13) 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 très près de l'état de l'art pour la génération automatique de résumés à un coût de calcul nettement inférieur.

1.3.2 Formulation abstractive

Sous cette formulation, on génère le résumé d'un texte un mot à la fois. Les résumés abstractifs peuvent donc potentiellement reproduire de manière parfaite le résumé cible associé à un

1. De manière analogue à la descente de gradient présentée à la section 1.2.1 pour la minimisation de fonction, l'ascension de gradient permet de maximiser une fonction.

document. Cette performance potentielle vient toutefois à un coût : la formulation abstractive est naturellement plus difficile car elle nécessite de gérer la syntaxe et les fautes d'orthographe en plus de la tâche de trouver les informations pertinentes du document. De surcroît, les modèles abstractifs peuvent aussi être sujets à la génération d'énoncés factuellement incorrects (Kryscinski et al., 2020) .

Comme on s'intéresse aux méthodes extractives pour le reste du document, on se contente de référer le lecteur aux approches abstractives représentant l'état de l'art (Dou et al., 2020; Raffel et al., 2020; Dong et al., 2019; Zhang et al., 2020). Au coeur de toutes ces approches se trouvent les architectures BERT (Devlin et al., 2018) et Transformer (Vaswani et al., 2017), auxquelles sont dues les percées récentes dans le domaine de la génération de texte.

1.3.3 Évaluation de la performance

À la section 1.3.1, on prenait pour acquis qu'une fonction G existait pour mesurer la proximité entre un résumé généré $\hat{s}$ et un résumé cible 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) de $\hat{s}$ et s. Par exemple, ROUGE-1 représente le chevauchement entre les unigrammes et ROUGE-2 celui entre les bigrammes ;

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

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

L'intuition ici est simple : plus le chevauchement est grand entre les n-grammes d'un résumé généré et ceux du résumé cible, plus le résumé généré a conservé l'information recherchée.

Or, si on se fie seulement au rappel sur les n-grammes, le résumé optimal sera de conserver tout le texte original. Pour pénaliser les résumés trop peu concis, on peut utiliser une métrique plus appropriée que le rappel comme le score F1 pour pénaliser les n-grammes présents dans le résumé généré mais pas dans la cible. Pour tout le document, tous les scores basés sur ROUGE rapportés seront donc basés sur la métrique F1.

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 (Peyrard, 2019). 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 l'utilisation d'une moyenne ne suffit pas à régler le problème en entier. Par exemple, Paulus

et al. (2017) entraînent plusieurs modèles abstractifs mais finissent par déployer un modèle n'atteignant pas la meilleure moyenne de score ROUGE-1, ROUGE-2 et ROUGE-L après une évaluation de la qualité des modèles par des humains.

Enfin, il est aussi important de mentionner que toutes les métriques énoncées plus haut ne sont pas dérivables. Elles ne peuvent donc pas être utilisées directement comme fonctions de perte pour l'entraînement d'un réseau de neurones.

Évaluation de la performance à partir d'un jeu de données

On termine en mentionnant que, dans le contexte où l'on apprend à partir d'un jeu de données $\mathcal{D} = \{(d_i, s_i)\}_{i=1}^{N}$, il est important de faire l'évaluation sur un jeu de données différent de celui utilisé pour l'entraînement. En effet, comme on s'intéresse à la performance d'un modèle de génération de résumés sur n'importe quel document en entrée, il est nécessaire de valider la performance du modèle sur des documents qui n'ont pas encore été vus par le modèle.

À cet effet, on sépare habituellement le jeu de données $\mathcal{D}$ en trois sous-ensembles : un ensemble d'entraînement, un ensemble de validation et un ensemble de test. L'ensemble d'entraînement est naturellement utilisé pour l'apprentissage du modèle alors que l'ensemble de validation est utilisé pour estimer la performance sur des nouveaux documents pendant l'entraînement. À la fin de l'entraînement, le modèle retenu est celui ayant la meilleure performance sur le jeu de validation. Enfin, l'ensemble de test est utilisé pour estimer la performance réelle du modèle retenu.

Chapitre 2

Problématique

L'objectif fondamental de ce mémoire est de proposer et d'analyser différents algorithmes basés sur les bandits pour l'apprentissage de systèmes de génération de résumés. Pour atteindre cet objectif, il est essentiel de se doter d'un cadre expérimental qui sera réutilisé pour tous les algorithmes testés afin de bien pouvoir attribuer les variations de performance aux algorithmes à l'essai. On commence donc en présentant et motivant la problématique de génération de résumés à l'étude, avec toutes les contraintes et hypothèses retenues.

On présente ensuite BanditSum (Dong et al., 2018), la première approche par bandit de ce document. On voit comment BanditSum aborde la génération de tous les documents d'un ensemble comme un bandit contextuel (Auer, 2002). On décrit par la suite comment l'algorithme REINFORCE (Williams, 1992) peut être utilisé pour résoudre le bandit contextuel proposé et on justifie son applicabilité à partir d'un ensemble de documents de développement. Enfin, on met BanditSum à l'essai sur le jeu de données du CNN/DailyMail (Hermann et al., 2015) et on fait valoir que la performance obtenue représente l'état de l'art en termes d'efficacité computationnelle.

2.1 Description

La problématique à laquelle on s'intéresse est de trouver des approches par bandit qui permettent l'entraînement de modèles de génération de résumés. Pour les travaux de ce document, il est choisi de s'intéresser exclusivement aux résumés extractifs (voir section 1.3.1), qui sont bâtis en sélectionnant des phrases d'un document. C'est la formulation extractive qui est retenue en raison de la facilité avec laquelle la génération de ce type de résumé peut être vue comme un problème de bandit.

Comme le résumé cible s associé à un document d est rarement constitué intégralement de phrases prélevées de d, il n'est pas trivial de définir une procédure d'apprentissage dans le

contexte extractif. En voyant la génération de résumés [1] comme un problème de bandit, on peut faire appel aux algorithmes connus de résolution des bandits pour optimiser les résumés produits par un modèle. Pour la problématique à l'étude, on considère deux niveaux d'application des bandits : sur un ensemble de documents et sur un seul document à la fois. Au niveau d'un ensemble de documents, on parlera d'un objectif d'optimisation de la performance espérée du modèle alors qu'au niveau d'un document, on parlera plutôt d'un objectif de production de cibles de qualité.

Le coeur de ce document repose dans l'analyse de procédures d'apprentissage pour ces deux objectifs. On se dote d'un cadre uniformisé pour comparer sur un pied d'égalité les différents algorithmes explorés. On fixe donc un jeu de données, une métrique d'évaluation et un modèle de réseau de neurones, tous largement inspirés de BanditSum, la première approche par bandit proposée dans la litérature pour l'entraînement de modèles de génération de résumés.

2.1.1 Jeu de données

Pour évaluer la performance des approches proposées, le jeu de données du CNN/DailyMail (Hermann et al., 2015), référence classique 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, séparés en 287 113 exemples d'entraînement, 13 368 exemples de validation et 11 490 exemples de test. Afin de mener des expériences empiriques plus poussées sur l'applicabilité des différentes approches par bandit proposées, on utilise aussi un sous-ensemble du jeu d'entraînement comme jeu de développement. Ce jeu de développement est composé de 25 000 exemples pigés aléatoirement du jeu d'entraînement afin d'être 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.

De plus, 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 à partir de ses 3 premières phrases. En raison de sa bonne performance sur le jeu de données, Lead-3 est généralement considérée comme une référence plancher pour la performance des modèles de génération de résumés. En effet, comme l'heuristique Lead-3 ne requiert aucun temps d'entraînement ou ressource de calcul dédiée, n'importe quel modèle entraîné qui ne dépasse pas sa performance est considéré comme peu pertinent.

On emploie aussi plusieurs des contraintes retenues par les approches de l'état de l'art sur le jeu de données du CNN/DailyMail. Notamment, on considère exclusivement 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

1. À partir de maintenant, comme on considère exclusivement les résumés extractifs, on utilisera résumé et résumé extractif de manière interchangeable.

les résumés cibles est près de 3. De surcroît, 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. Selon les statistiques du jeu de données, on considère aussi la régularisation 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.

2.1.2 Métrique d'évaluation

Pour l'évaluation de la performance, on emploie les métriques ROUGE (Lin, 2004) présentées à la section 1.3.3. On retient la métrique de similarité entre deux résumés utilisée par la plupart des travaux employant l'apprentissage par renforcement (Paulus et al., 2017; Dong et al., 2018; Luo et al., 2019)

$$\text{ROUGE}(\hat{s}, s) := \frac{1}{3}\left[\text{ROUGE-1}(\hat{s}, s) + \text{ROUGE-2}(\hat{s}, s) + \text{ROUGE-L}(\hat{s}, s)\right]. \tag{2.1}$$

Ici, ROUGE-1 et ROUGE-2 représentent respectivement le chevauchement au niveau des unigrammes et des bigrammes entre $\hat{s}$ et s alors que ROUGE-L mesure leur plus longue sous-séquence commune. Toutes les métriques utilisées sont basées sur le score F1 afin de pénaliser les résumés trop longs, tel que mentionné à la section 1.3.3.

Notons que le choix de (2.1) comme métrique d'évaluation est motivé par deux facteurs principaux. D'abord, ROUGE [2] jouit d'une diversité de signal naturelle comme il s'agit de la moyenne de trois métriques différentes. C'est cette diversité de signal qui permet d'avoir une évaluation automatique se rapprochant le plus possible de l'évaluation humaine. Aussi, le score ROUGE a la propriété intéressante d'être presque invariant à l'ordre des phrases dans un résumé : seul le score ROUGE-2 est modifié au niveau de la jonction entre deux phrases quand l'ordre des phrases varie. On exploite abondamment cette propriété car elle permet d'éviter l'explosion combinatoire induite par la considération de l'ordre des phrases dans un résumé. Notamment, pour un groupe de 3 phrases composant un résumé, on assigne toujours le score ROUGE associé au résumé où les phrases apparaîssent dans le même ordre que dans le document original.

Pré-calcul des scores ROUGE

Les méthodes présentées dans ce document font toutes appel au calcul de plusieurs scores ROUGE dans leur procédure d'entraînement. Comme les calculs de ROUGE sont plutôt lents et que l'on souhaite éviter qu'ils deviennent le principal fardeau computationnel, on effectue le pré-calcul des scores pour tous les documents du jeu d'entraînement. Pour un document

2. Ici et dans le reste du document, ROUGE représente la métrique définie par l'équation (2.1).

d, on sait que tous les résumés que l'on doit considérer sont les combinaisons de 3 phrases ordonnées selon leur position initiale dans d. Il est donc possible de calculer et entreposer le score ROUGE$(\hat{s}, s)$ de tous les résumés $\hat{s}$ d'un document et de simplement aller lire la valeur correspondante lorsque nécessaire dans l'entraînement.

2.1.3 Architecture neuronale

Au niveau architectural, on reprend exactement le modèle neuronal de BanditSum, pour lequel une illustration est disponible à la figure 2.1. Le modèle obtient d'abord une représentation de chaque phrase en prenant la moyenne des sorties d'un LSTM bidirectionnel à deux couches appliqué sur les plongements de mots. Ces représentations sont ensuite passées à un second LSTM bidirectionnel à deux couches, lequel produit des représentations de chaque phrase qui tiennent compte des autres phrases du document. Les affinités sont finalement obtenues via une tête de prédiction constituée d'un réseau pleinement connecté composé de deux couches reliées par une activation ReLU et avec sortie sigmoïde. Ici, la fonction sigmoïde est utilisée car elle permet d'avoir une sortie entre 0 et 1 que l'on interprète comme l'affinité prédite par le modèle pour l'inclusion d'une phrase dans le résumé. À l'inférence, un résumé est produit en sélectionnant les 3 phrases avec la plus grande affinité. On utilisera exactement la même architecture de réseau de neurones pour toutes les expérimentations de ce document.

Détails expérimentaux

On reprend des détails d'implémentation identiques à BanditSum afin de faciliter la comparaison avec les autres approches par bandit. La dimension cachée des LSTMs est fixée à 200 et celle de la tête de prédiction est de 100. Les plongements de mots utilisés sont les plongements GLoVe (Pennington et al., 2014) de taille 100 pré-entraînés sur la langue anglaise. On retient seulement les plongements pour les mots présents au moins une fois dans le jeu d'entraînement. Pour tous les mots pour lesquels on ne dispose pas de plongement (mots hors vocabulaire), on utilise un même plongement aléatoire partagé. 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.

Tous les entraînements sont faits en itérant 5 fois sur l'entièreté du jeu d'entraînement. La taille de *minibatch* utilisée pour la mise à jour des paramètres est $B = 64$. Toutes les expérimentations sont menées sur une carte graphique Tesla T4, dotée d'une mémoire virtuelle de 16 GB.

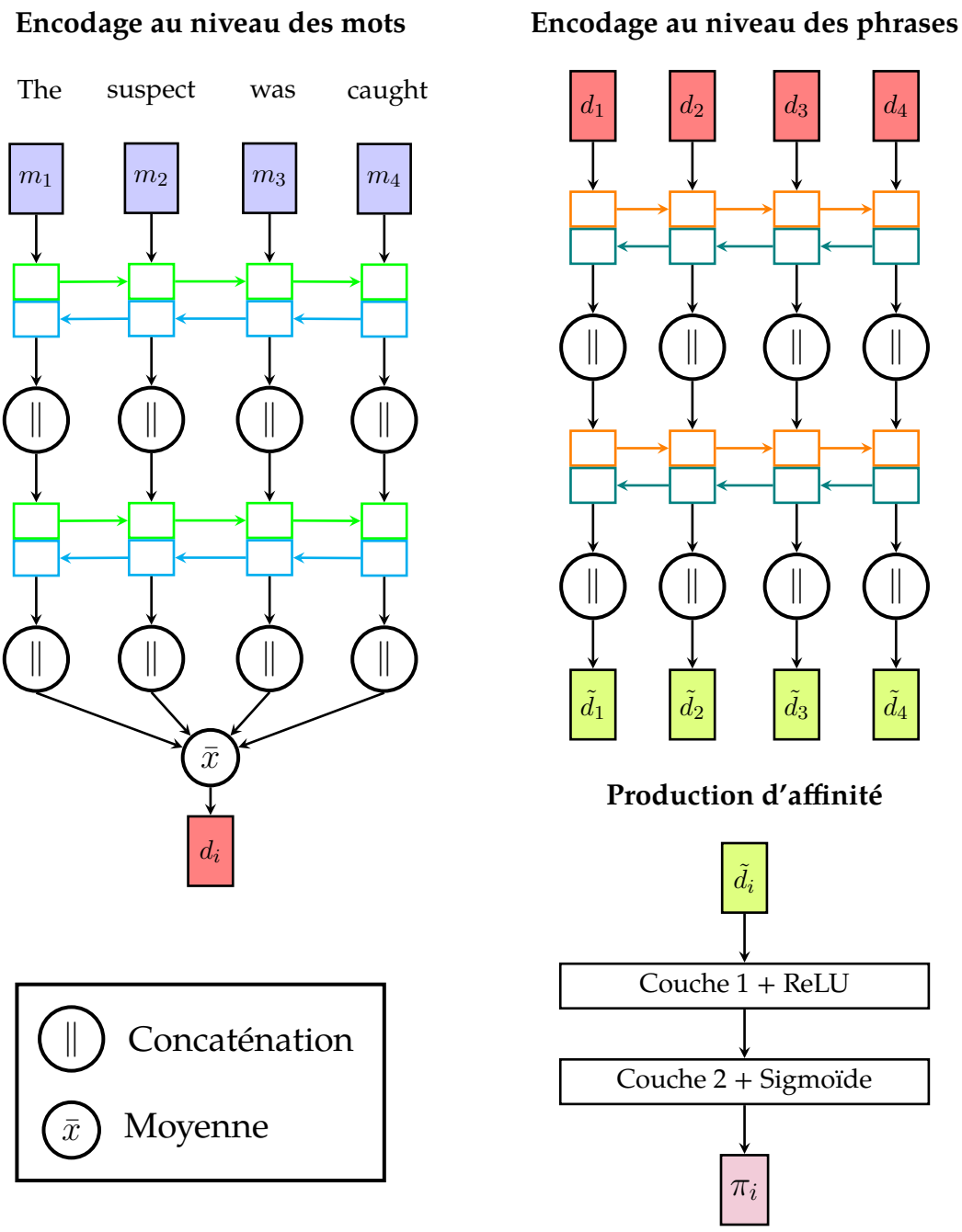

FIGURE 2.1 – Schéma décrivant l'architecture neuronale employée par BanditSum et reprise pour les expérimentations tout au long du document. On présente tout le parcours interne des deux LSTMs bidirectionnels à deux couches. Les représentations de phrases à partir de leurs mots sont en rouge et les représentations incorporant les autres phrases sont en vert pâle.

2.2 Bandit contextuel

Un bandit contextuel est une généralisation du problème de bandit stochastique présenté en 1.1.1. Dans cette formulation, à chaque pas de temps t l'apprenant perçoit un contexte $c_t \in \mathcal{C}$, pour $\mathcal{C}$ l'ensemble des contextes possibles. Ce contexte c_t permet à l'apprenant de guider le choix de son action a_t, à laquelle une récompense dépendante du contexte $X_{a_t,t} = r(c_t, a_t)+\xi_t$, où ξ_t est un bruit (e.g. supposé gaussien), est associée. Comme il n'existe pas de contraintes sur le contexte utilisé, cette formulation est très flexible et peut être utilisée pour n'importe quelle tâche où le comportement optimal d'un apprenant est dépendant d'informations reçues en entrée.

2.2.1 Application à la génération de résumés

La formulation contextuelle se prête naturellement à la génération de résumés pour un ensemble de documents que représente notre problématique (voir la section 2.1). Pour une paire document-résumé (d, s) d'un jeu de données, il suffit de définir le contexte comme étant le document $(c = d)$. En génération de résumés, l'utilisation d'un document comme contexte 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. En posant $\mathcal{S}_d$ comme étant l'ensemble des résumés de 3 phrases de d, on a que les actions qui s'offrent à l'apprenant sont simplement les résumés $\hat{s} \in \mathcal{S}_d$. Ainsi, la récompense X_t perçue au temps t représente naturellement la valeur de ROUGE associée ROUGE$(\hat{s}_t, s)$ au résumé $\hat{s}_t$ sélectionné par l'apprenant au temps t.

L'introduction de la notion de contexte est cruciale dans le cas de la génération de résumés, car elle permet d'avoir un seul apprenant dont la tâche est de produire un résumé pour n'importe quel document d en entrée. Cet apprenant peut alors être n'importe quelle fonction paramétrée, un réseau de neurones par exemple, dont les paramètres θ sont appris dans l'objectif de maximisation de la performance espérée sur tous les documents.

2.2.2 REINFORCE pour maximisation de la performance espérée

Si l'on considère un apprenant π_θ doté d'une certaine paramétrisation θ, l'algorithme REINFORCE (1.3.1) peut être utilisé pour maximiser $J(\theta)$ (1.13), la récompense espérée en générant un résumé à partir de π_θ. Selon le formalisme de génération de résumés défini à la section 1.3.1, l'apprenant π retourne des affinités de sélection $\pi(d) \in [0,1]^{|d|}$ pour chaque phrase d'un document d.

Dans ce contexte, il est alors naturel de vouloir utiliser un processus stochastique pour la génération de résumés, car cela permet une expressivité bien plus élevée pour J. En effet, en sélectionnant les résumés de manière vorace (i.e. selon les 3 affinités maximales de $\pi(d)$), $J(\theta)$ devient une simple espérance sur la pige d'un document d, une mesure peu révélatrice

de la performance globale de l'apprenant. Toutefois, si l'on génère les résumés en pigeant sans remise dans la distribution $p(d)$ induite par $\pi(d)$, on permet à $J(\theta)$ de tenir compte de tous les résumés possibles de d, pondérés par leur probabilité de pige, permettant de distinguer des distributions qui accordent une plus grande probabilité aux meilleurs résumés. Pour toutes les expériences de ce chapitre, on emploie donc le processus ξ de génération stochastique de résumé qui pige 3 phrases sans répétition selon la distribution induite par $\pi(d)$.

En pratique, les approximations de $\nabla J(\theta)$ nécessaires à la mise à jour de REINFORCE générées en utilisant un seul échantillon de J sont très instables pour un processus stochastique comme ξ, rendant l'ascension de gradient (1.14) difficile. Une solution simple et peu coûteuse pour stabiliser l'entraînement est alors de piger N résumés $\hat{s}_n$ en fonction de ξ et π_θ pour bâtir un meilleur estimateur selon

$$\nabla J(\theta) = \frac{1}{N} \sum_{n=1}^{N} \text{ROUGE}(\hat{s}_n, s) \nabla \ln \xi \left(\hat{s}_n | \pi_\theta, d\right). \tag{2.2}$$

Ici, $\xi \left(\hat{s}_n | \pi_\theta, d\right)$ est la probabilité de produire le résumé $\hat{s}_n$ a partir de la distribution induite par $\pi_\theta(d)$ et du processus de génération stochastique ξ et ROUGE($\hat{s}_n$, s) est la similarité entre un résumé généré $\hat{s}_n$ et sa cible s.

2.2.3 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 (2.2), la dérivée de $\ln \xi(\pi)$ 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 exacte de récompense de ξ et de π sur une paire document-résumé (d, s) quelconque et on pose $\bar{J}_N(\theta) = \frac{1}{N} \sum_{n=1}^{N} \text{ROUGE}(\hat{s}_n, s)$, son approximation à partir de N échantillons. La qualité de l'échantillonnage dépend de la proximité entre J et $\bar{J}_N$, que l'on nomme l'erreur d'échantillonnage Δ_N

$$\Delta_N = \left| J(\theta) - \bar{J}_N(\theta) \right|. \tag{2.3}$$

Méthodologie

On valide la rapidité de la convergence de $\bar{J}_N$ vers J en les comparant directement sur notre ensemble de développement (2.1.1). Comme le calcul de $\bar{J}_N$ requiert une distribution pour

un document, on génère artificiellement des distributions $p(d)$ sur les phrases d'un document d. Pour assurer une bonne variété dans les distributions générées, on choisit des distributions représentant une somme pondérée entre une distribution uniforme $U(d)$ et une distribution vorace $G(d)$:

$$p(d) = \tau U(d) + (1 - \tau)G(d).$$

La distribution vorace employée $G(d)$ représente un vecteur de taille $|d|$ où trois indices pigés aléatoirement ont la valeur de $\frac{1}{3}$ et les autres indices sont nuls. En jumelant ainsi des distributions vorace et uniforme selon un paramètre τ, les distributions générées $p(d)$ peuvent être arbitrairement plus faciles ou difficiles à estimer. Plus τ est élevé, plus $p(d)$ sera près d'une distribution vorace, pour laquelle la pige stochastique est toujours identique et l'approximation ne requiert qu'un seul échantillon.

Les expériences consistent à calculer la valeur de Δ_N pour N allant jusqu'à 50 sur tous les documents du jeu de développement. Pour chaque document, on génère 100 distributions $p(d)$ en faisant varier τ de 0 à 1 en incréments de 0.01 et en repigeant les indices non-nuls de $G(d)$ à chaque fois. Pour chaque distribution artificielle $p(d)$, les approximations $\bar{J}_N$ sont calculées. On calcule la véritable valeur de

$$J = \mathop{\mathbb{E}}_{\hat{s} \sim \xi(p(d))} [\text{ROUGE}(\hat{s}, s)]$$

grâce au fait que l'on possède les scores ROUGE de tous les résumés extractifs de chacun des documents.

2.2.4 Résultats

Les résultats obtenus sont rapportés dans les figures 2.2 et 2.3 qui présentent respectivement l'impact de la taille du document et de la distribution $p(d)$ employée sur l'évolution de l'erreur d'échantillonnage Δ_N. 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.

Tout d'abord, sur la figure 2.2, 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.

Une mesure naturelle sur la distribution des résumés est la valeur maximale de la distribution des résumés $\xi\left(p(d)\right)$. La figure 2.3 montre une évolution à la tendance logarithmique pour

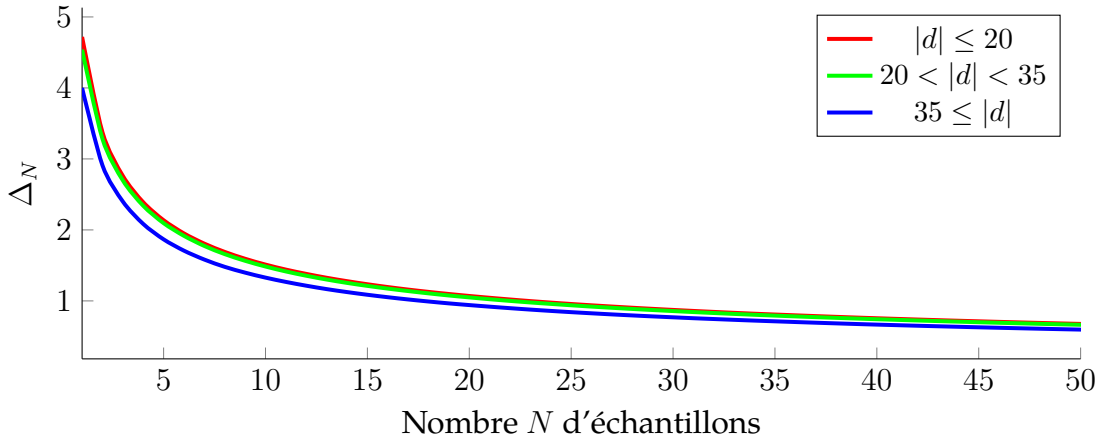

FIGURE 2.2 – Impact de la taille du document en entrée sur l'erreur d'échantillonnage (2.3) (plus bas est meilleur).

chaque nombre d'échantillons N considérés, où plus la probabilité maximale de la distribution augmente, plus rapide est la convergence de notre processus d'échantillonnage.

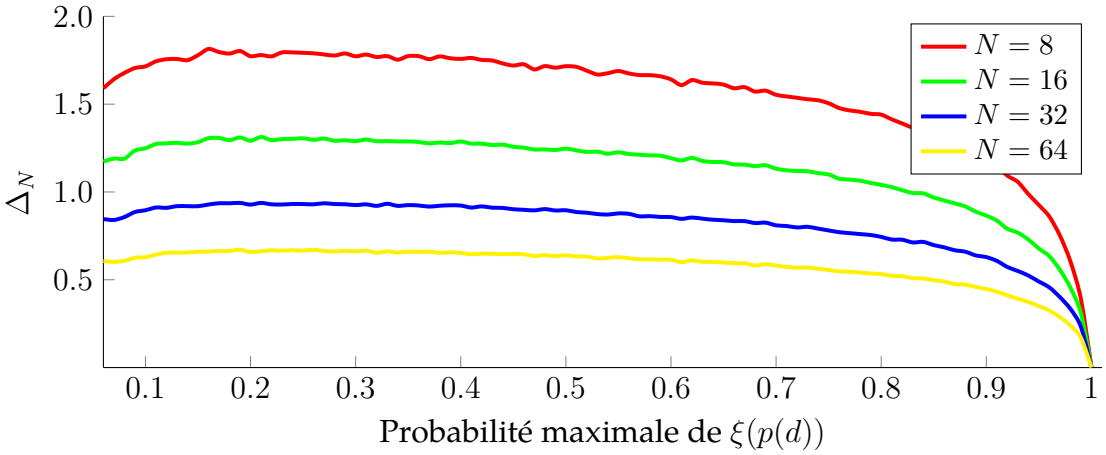

FIGURE 2.3 – Impact de la probabilité maximale de la distribution des résumés sur l'erreur d'échantillonnage (2.3) (plus bas est meilleur).

Globalement, on remarque aussi que la convergence est très rapide. Une différence de moins de 1 point ROUGE est définitivement suffisante pour justifier l'utilisation de $\bar{J}_N$ au lieu de J dans la mise à jour du gradient et elle est toujours obtenue par $N = 32$. Comme il est plus coûteux en temps de calcul de faire plus d'échantillons, il apparaît naturel de considérer aussi des valeurs de N inférieures pour faire un compromis entre le temps de calcul et la rapidité de calcul de l'estimation $\bar{J}_N$. Notons que les articles qui utilisent cet échantillonnage dans leur implémentation de REINFORCE (Dong et al., 2018; Luo et al., 2019) utilisent $N = 20$ dans

leurs expériences. Nos expériences apportent ainsi une justification empirique rigoureuse à ce choix fait à la suite d'expérimentations en pratique.

Maintenant que l'on sait que l'on peut avoir des approximations très justes pour J (donc $\nabla_\theta J$), on vient d'établir que l'estimateur par échantillonnage de l'équation (2.2) peut être utilisé sans problème avec l'algorithme REINFORCE. Ainsi, on dispose d'un algorithme d'ascension de gradient pour les paramètres θ d'un apprenant π_θ sur le problème de bandit contextuel de la génération de résumés.

2.3 BanditSum

BanditSum est un algorithme dans lequel on considère un apprenant neuronal π_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 π_θ est présentée à la section 2.1.3. Les paramètres θ de l'apprenant sont appris via une ascension de gradient à partir de l'équation (2.2). 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* $b(d)$:

$$\nabla J(\theta) = \frac{1}{N} \sum_{n=1}^{N} \big[\text{ROUGE}(\hat{s}_n, s) - b(d)\big] \nabla \ln \xi\big(\hat{s}_n | \pi_\theta, d\big).$$

La *baseline* utilisée par BanditSum est $b(d) = \text{ROUGE}\left(\psi(\pi), s\right)$, représentant le score associé au résumé le plus probable du document d selon π. L'introduction de b peut être vue comme assignant un scalaire positif aux résumés meilleurs que le résumé actuellement privilégié et un scalaire négatif à ceux qui sont moins bons. En pratique, l'introduction d'une *baseline* est fréquemment utilisée pour réduire la variance élevée dans l'entraînement avec REINFORCE et faciliter l'apprentissage.

Le second artefact employé par BanditSum est 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 utilisé 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 permettre une bonne exploration de l'espace des résumés possibles lors de l'entraînement. Enfin, notons que la probabilité de générer un résumé $\hat{s}$ d'un document d, représentée par $\xi\left(\hat{s} | \pi_\theta, d\right)$ dans l'équation 2.3, s'écrit alors sous la forme close

$$\xi\left(\hat{s} | \pi_\theta, d, \epsilon\right) = \prod_{i=1}^{3} \left(\frac{\epsilon}{|d| - i + 1} + \frac{(1-\epsilon)\pi(d)_{\hat{s}_i}}{\sum_s \pi(d)_s - \sum_{j=1}^{i-1} \pi(d)_{\hat{s}_j}} \right), \tag{2.4}$$

où $\hat{s}_i$ représente l'index de la i-ème phrase du résumé produit $\hat{s}$. Ainsi, le gradient de (2.4), 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 (Abadi et al., 2016; Paszke et al., 2019).

Enfin, dans l'article original de BanditSum, il est recommandé de faire une mise à jour des poids à partir d'une *minibatch* contenant seulement 1 article. On a expérimenté avec différentes tailles B de *minibatch*, où la mise à jour des paramètres θ est désormais faite selon

$$\nabla J(\theta) = \frac{1}{B}\sum_{i=1}^{B}\frac{1}{N}\sum_{n=1}^{N}\big[\text{ROUGE}(\hat{s}_{i,n}, s_i) - b(d_i)\big]\nabla \ln \xi\big(\hat{s}_{i,n}|\pi_\theta, d_i, \epsilon\big). \tag{2.5}$$

Nos essais n'ont pas démontré de gain de performance notoire pour la version avec $B = 1$ suggérée et on utilisera donc $B = 64$ dans nos expériences, afin de profiter pleinement de la capacité de parallélisation des réseaux de neurones.

Le pseudocode détaillant BanditSum se trouve à l'algorithme 1. Pour chaque paire document-résumé (d,s) de chaque *minibatch* d'exemples du jeu de données, BanditSum commence par piger N résumés selon $\xi(\pi_\theta(d), \epsilon)$ et obtenir leurs scores ROUGE r_n associés. La baseline $b(d) = \text{ROUGE}(\psi(\pi), s)$ est alors calculée comme étant le score associé au résumé le plus probable du document d selon π_θ. Les poids θ sont enfin mis à jour selon (2.5).

Algorithme 1 BanditSum

Entrée: $\mathcal{D}$ (jeu de données), N (nombre d'échantillons), α (taux d'apprentissage), ϵ (taux d'exploration), B (taille de *minibatch*).

tant que critère d'arrêt non atteint **faire**
  batch $\sim \mathcal{D}^B$ ▷ On pige la minibatch du jeu de données
  $\nabla = \mathbf{0}$
  pour tout $(d,s) \in$ batch **faire**
    Piger N résumés $\hat{s}_n \sim \xi(\pi_\theta(d), \epsilon)$
    Calculer les N scores associés $r_n = \text{ROUGE}(\hat{s}_n, s)$
    $b = \text{ROUGE}(\psi(\pi), s)$
    $\nabla = \nabla + \frac{1}{N}\sum_{n=1}^{N}(r_n - b)\nabla_\theta \ln \xi\left(\hat{s}_n|\pi_\theta, d, \epsilon\right)$ ▷ (2.5)
  $\theta = \theta + \alpha\frac{1}{B}\nabla$

2.3.1 Expériences

On effectue un entraînement sur le jeu de données CNN/DailyMail en parcourant dans son entièreté le jeu d'entraînement à 5 reprises et en utilisant des *minibatches* de taille $B = 64$. Notre implémentation est faite selon les détails expérimentaux décrits à la section 2.1.3.

Pour les expériences, on s'intéresse à l'importance que peut avoir un meilleur estimé du gradient sur la convergence et la qualité du modèle produit. On expérimente donc avec les nombres d'échantillons $N \in \{8, 16, 32\}$ pour la mise à jour des poids selon (2.5), en se basant sur nos expériences empiriques sur l'échantillonnage de la section 2.2.4. Étant donné le facteur aléatoire présent dans l'entraînement, on effectue 5 entraînements distincts pour chaque N et on rapporte la moyenne des résultats obtenus.

On compare les modèles obtenus par notre implémentation à l'heuristique de référence Lead-3 (2.1.1) ainsi qu'aux performances rapportées de trois algorithmes présentés dans la littérature : BertSumExt (Liu and Lapata, 2019), Refresh (Narayan et al., 2018) et BanditSum (Dong et al., 2018). La comparaison est faite sur les bases de l'efficacité computationnelle, que l'on établit à partir de la performance des modèles sur le jeu de test, leur taille ainsi que le nombre d'entraînement utilisés pour leur entraînement. Notons que BertSumExt représente actuellement l'état de l'art sur le jeu de données du CNN/DailyMail et que les architectures neuronales obtenant des performances similaires ont toutes une taille comparables. BertSumExt et Refresh sont présentés en raison de la disponibilité de la configuration expérimentale utilisée et de leur considération exclusive de résumés extractifs de 3 phrases, comme dans le cadre uniformisé.

2.3.2 Résultats

On présente à la figure 2.4 l'évolution du score ROUGE moyen obtenu en validation par notre implémentation de BanditSum pour différentes valeurs de N. Le tableau 2.1 présente quant à lui une comparaison entre la performance sur le jeu de test et la procédure d'entraînement de divers algorithmes de l'état de l'art et notre implémentation de BanditSum.

Tout d'abord, la figure 2.4 illustre l'évolution de la performance sur le jeu de validation des modèles selon le nombre d'échantillons N utilisé. On constate que, sur le jeu de validation du moins, les différentes valeurs de N testées mènent toutes à des performances similaires, sans différence statistiquement significative. Aussi, la courbe bleue semble indiquer un entraînement plus stable en utilisant $N = 32$, car les variations de performance sont moins nombreuses et plus douces.

Le tableau 2.1 présente une comparaison de la performance obtenue par divers modèles sur le jeu de test en fonction de leur efficacité computationnelle. Pour chacun des modèles, on indique le score ROUGE obtenu sur le jeu de test, la taille du modèle (incluant les plongements de mots), le nombre de mises à jour des poids effectuées et la taille de la *minibatch* utilisée pour chaque mise à jour.

Une première observation intéressante qui peut être tirée du tableau 2.1 est que l'on parvient à reproduire de manière convaincante les résultats rapportés par l'article original BanditSum. En effet, bien que l'on ne teste pas directement avec le nombre d'échantillons $N = 20$ utilisé

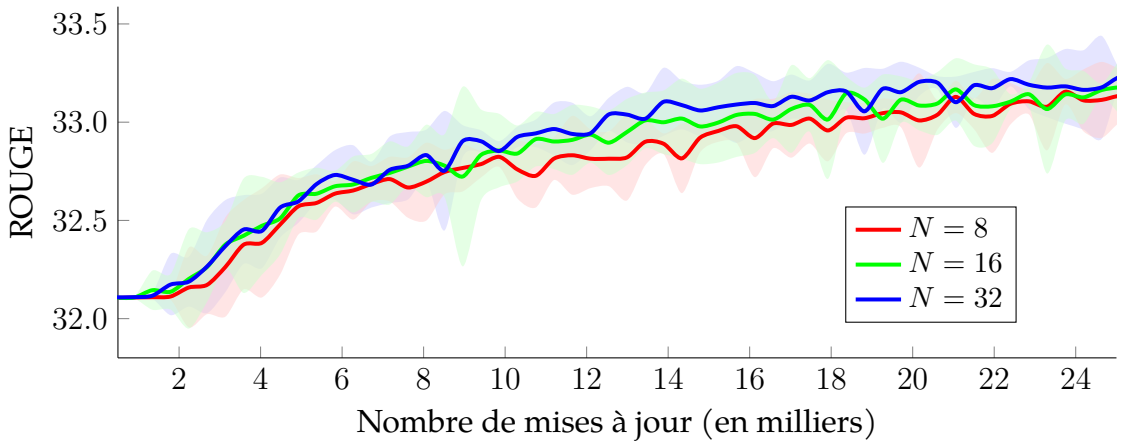

FIGURE 2.4 – Évolution du score ROUGE (plus élevé est meilleur) de BanditSum sur le jeu de validation selon le nombre N d'échantillons utilisés. Un intervalle de confiance à 95 % calculé à partir de 5 entraînements distincts est rapporté pour chaque N.

Modèle	**ROUGE**	**Taille (MB)**	**Mises à jour (en milliers)**	**Taille de minibatch**
Lead-3†	31.16	-	-	-
BertSumExt (Liu and Lapata, 2019)	**34.7**	1900	50	6000
Refresh (Narayan et al., 2018)	31.6	1500	300	20
BanditSum (Dong et al., 2018)	32.6	**80**	1 150	1
BanditSum† ($N = 8$)	32.51 ± 0.06	**80**	**25**	**64**
BanditSum† ($N = 16$)	32.58 ± 0.04	**80**	**25**	**64**
BanditSum† ($N = 32$)	32.58 ± 0.09	**80**	**25**	**64**

Tableau 2.1 – Comparaison entre Lead-3, les modèles extractifs de l'état de l'art et BanditSum sur le jeu de test du CNN/DailyMail. On rapporte le score ROUGE (plus élevé est meilleur) sur le jeu de test ainsi que les ressources de calcul nécessaire à l'entraînement pour chaque modèle. Les † indiquent qu'il s'agit de notre implémentation de BanditSum et de notre propre recalcul de Lead-3. Pour notre implémentation de BanditSum, un intervalle de confiance à 95 % obtenu à partir de 5 entraînements distincts est rapporté sur la valeur de ROUGE.

dans l'article, les entraînements avec 16 et 32 échantillons de notre implémentation obtiennent des performances sans différence statistiquement significative par rapport à l'article original. Soulignons aussi que cette reproduction des résultats est obtenue en dépit de l'augmentation de taille de minibatch B de 1 à 64 dans notre implémentation. Cette dernière nuance est

importante car, en utilisant $B = 64$, notre modèle parcourt le jeu de données en entier environ 25 fois plus rapidement.

Le tableau 2.1 témoigne aussi de l'efficacité computationnelle indéniable de BanditSum. En effet, le modèle neuronal de BanditSum est près de 20 fois plus petit que celui de Refresh mais obtient tout de même une meilleure performance en test. De surcroît, en considérant le nombre d'exemples utilisés pour l'entraînement (nombre de mises à jours multiplié par la taille de la *minibatch*), BanditSum est aussi particulièrement efficace dans son apprentissage. Ainsi, comme les 2 points ROUGE que BertSumExt réussit à obtenir de plus que BanditSum se font au prix d'un modèle 25 fois plus gros et de 100 fois plus d'exemples vus en entraînement, on considère que BanditSum représente l'état de l'art en termes d'efficacité computationnelle.

2.4 Conclusion

On a débuté ce chapitre en définissant clairement la problématique qui fait l'étude de ce document : l'utilisation des bandits pour des algorithmes d'entraînement de modèles de génération de résumés extractifs. Pour permettre une comparaison rigoureuse entre les divers algorithmes à l'essai, on s'est ensuite doté d'un cadre expérimental uniformisé. Dans le cadre uniformisé, on s'intéresse au jeu de données CNN/DailyMail, duquel on extirpe un jeu de développement de 25 000 documents d'entraînement pour des expériences empiriques dédiées à la validation de l'applicabilité des formulations bandits. On considère exclusivement la génération de résumés extractifs de 3 phrases et on reprend le modèle neuronal de BanditSum. Le cadre uniformisé retenu est tiré en grande partie de BanditSum, car il s'agit de la première approche par bandit pour l'entraînement de modèles de génération de résumés dans la litérature.

Après, on a présenté BanditSum, qui utilise la formulation en bandit contextuel de la tâche de génération de résumés pour l'optimisation de la performance sur un ensemble de documents. On a détaillé la formulation en bandit contextuel au coeur de BanditSum et validé l'utilisation de l'échantillonnage pour l'estimation du gradient nécessaire à l'apprentissage du bandit par REINFORCE. Notamment, nos expériences ont permis de justifier rigoureusement le choix de BanditSum d'effectuer 20 échantillons par document pour l'estimation du gradient de REINFORCE. Enfin, on a décortiqué l'algorithme BanditSum, que l'on a mis à l'épreuve sur le cadre expérimental uniformisé. Les résultats obtenus ont montré qu'avec un modèle 25 fois plus petit et moins de 10 % du nombre de documents vus, BanditSum génère des modèles seulement légèrement inférieurs à l'état de l'art. On fait valoir que cela permet d'établir la supériorité de BanditSum en termes de ressources de calcul requises par rapport aux autres algorithmes de l'état de l'art.

Chapitre 3

Bandit combinatoire

Dans ce chapitre, on présente comment la génération du résumé d'un document peut être interprétée comme un problème de bandit combinatoire (Chen et al., 2013). On montre ensuite comment l'algorithme Combinatorial Upper Confidence Bound (CUCB) (Chen et al., 2013) peut être utilisé pour résoudre ce type de problème. Avec le jeu de développement, on établit empiriquement l'applicabilité de CUCB pour l'identification du meilleur résumé d'un document. On s'intéresse alors à savoir comment la formulation combinatoire et sa résolution par CUCB peuvent être employées pour générer des cibles d'entraînement pour les phrases d'un document. À cette fin, on introduit le concept de potentiel extractif d'une phrase et on montre comment CUCB permet de générer des bonnes estimations du potentiel de chaque phrase. On présente une procédure de génération de cibles d'entraînement basées sur le potentiel extractif des phrases que l'on compare à l'approche utilisant les cibles binaires habituelles basées sur un oracle (Nallapati et al., 2017). L'applicabilité de la procédure de génération de cibles proposée est validée empiriquement à partir du jeu de données de développement. On présente comment ces cibles peuvent être utilisées pour l'apprentissage de systèmes de génération de résumés en proposant un nouvel algorithme que l'on nomme CombiSum. Enfin, on met CombiSum à l'épreuve sur notre cadre expérimental uniformisé en comparant sa performance à celle obtenue à partir des cibles issues d'un oracle pour juger de son efficacité.

3.1 Bandit combinatoire

Le bandit combinatoire est un problème de bandit qui vise à étendre la formulation stochastique à des environnements où plusieurs actions $a \in \mathcal{A}$ peuvent être sélectionnées en même temps par l'apprenant. On s'intéresse seulement à la variante dite multitâche [1] (Lattimore and Szepesvári, 2020), où il y a un nombre fixé m d'actions sélectionnées à chaque pas de temps t. Les actions sélectionnées créent une *super-action* $\mathcal{M} = \{a_1, ..., a_m\}$ pour laquelle l'environne-

1. On fera un léger abus de langage dans le reste du document, utilisant le terme bandit combinatoire pour définir le bandit combinatoire multitâche.

ment retourne une récompense $X_{\mathcal{M},t}$. L'objectif demeure la minimisation du pseudo-regret cumulatif (1.1), mais celui-ci est désormais calculé entre l'espérance de la *super-action* optimale $\mu^* = \mu_{\mathcal{M}^*}$ et celles des *super-actions* $\mathcal{M}_t$ choisies

$$R_T = T\mu^* - \sum_{t=1}^{T} \mu_{\mathcal{M}_t}. \tag{3.1}$$

Tout l'intérêt de cette formulation vient de l'éventuelle relation entre les *super-actions*. Bien qu'il serait possible de formuler le bandit combinatoire comme un bandit stochastique 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 $a \in \mathcal{A}$ sont partagées entre les diverses *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 peut alors être utilisée pour guider le choix des prochaines *super-actions* et accélérer l'apprentissage.

3.1.1 Application à la génération de résumé

En contraste au bandit contextuel présenté à la section 2.2.1, la formulation combinatoire est appliquée à la génération du résumé extractif d'un seul document. Dans notre cadre uniformisé (2.1), la génération du résumé d'un document d peut être vue comme un bandit combinatoire où les actions de base $a \in \mathcal{A}$ sont les phrases $d_i \in d$ et les *super-actions* $\mathcal{M}$ sont les résumés de 3 phrases possibles pour d. En choisissant la *super-action* $\mathcal{M} = \{a_{i_1}, a_{i_2}, a_{i_3}\}$ associée au résumé $\hat{s} = \{d_{i_1}, d_{i_2}, d_{i_3}\}$, l'apprenant reçoit la récompense ROUGE$(\mathcal{M}, s)$ et peut mettre à jour ses statistiques sur les phrases d_i de $\hat{s}$. Pour alléger la notation, on dira désormais que la *super-action* $\mathcal{M} = \{a_{i_1}, a_{i_2}, a_{i_3}\}$ est un résumé où l'action a_i correspond à la phrase d_i et que l'ensemble des actions possibles est l'ensemble des phrases d'un document, i.e. $\mathcal{A} = d$.

Notons aussi que, comme on ne tient pas compte de l'ordre dans le cadre expérimental uniformisé, 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 ROUGE est basé sur un score F1 sur le résumé $\mathcal{M}$ complet, cette hypothèse n'est pas strictement vraie, mais demeure 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.

3.1.2 UCB pour bandit combinatoire (CUCB)

Tel que décrit à la section 1.1.2, Upper Confidence Bound (UCB) (Auer et al., 2002) est un algorithme permettant de minimiser le pseudo-regret cumulatif pour les problèmes de bandit

stochastique. Bien que UCB est conçu pour les problèmes de bandit stochastique, l'approche peut être naturellement étendue aux problèmes de bandit combinatoire avec l'algorithme Combinatorial Upper Confidence Bound (CUCB). CUCB permet de minimiser le pseudo-regret cumulatif de n'importe quel bandit combinatoire, mais on présente ici seulement son application à la formulation multitâche, qui est la formulation présentée à la section 3.1.1 pour représenter la génération du résumé extractif d'un document.

Pour étendre UCB au bandit multitâche, CUCB modifie d'abord la moyenne empirique $\bar{x}_a(t)$ et le nombre de visites $n_a(t)$ maintenues pour chaque action a après t pas de temps. Ces statistiques sont modifiées indiquer maintenant la présence de l'action dans les *super-actions* sélectionnées selon

$$\bar{x}_a(t) = \frac{1}{n_a(t)} \sum_{\tau=1}^{t} \mathbb{I}\left[a \in \mathcal{M}_\tau\right] X_{\mathcal{M}_\tau,\tau} \quad \text{et} \quad n_a(t) = \sum_{\tau=1}^{t} \mathbb{I}\left[a \in \mathcal{M}_\tau\right],$$

où $\mathbb{I}$ est la fonction indicatrice retournant 1 quand son prédicat est vrai et 0 sinon.

À chaque pas de temps t, CUCB fait appel à un oracle pour sélectionner la *super-action* $\mathcal{M}_t$ la plus prometteuse selon la borne supérieure UCB (1.2) de chacune des actions $a \in \mathcal{A}$. L'oracle utilisé est requis d'être probablement approximativement correct dans son estimation de la *super-action* optimale. Dans le contexte multitâche et selon l'hypothèse faite à la section 3.1.1 que chaque phrase d'un résumé contribue de manière égale au score du résumé, il est naturel de considérer l'oracle qui sélectionne directement la *super-action* contenant les 3 actions avec la borne UCB maximale. Bien qu'il n'est pas forcément toujours optimal de sélectionner les 3 phrases les plus prometteuses individuellement pour bâtir un résumé, cette approche sera presque optimale la plupart du temps. Cela revient à sélectionner au temps t la *super-action* composée des trois actions avec la borne supérieure maximale, i.e.

$$\mathcal{M}_t = \text{3-}\arg\max_{a \in \mathcal{A}} \text{UCB}_a(t-1). \tag{3.2}$$

En sélectionnant les *super-actions* selon (3.2), CUCB accomplit l'objectif de minimisation du pseudo-regret cumulatif (3.1). Une vue d'ensemble de l'application de CUCB au bandit combinatoire représentant la génération du résumé d'un document se trouve à l'algorithme 2. Notons que, pour mettre CUCB à l'échelle de chaque document d, l'exploration que l'on utilise est plutôt guidée par $\beta' = \frac{\beta}{|d|}$.

3.2 Génération de cibles

Pour entraîner le réseau de neurones décrit à la section 2.1.3, il est possible d'employer des cibles extractives y associées à chaque document d. Intuitivement, ces cibles doivent repré-

Algorithme 2 CUCB pour génération de résumé

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

Initialiser $\bar{x}_a = 0$ et $n_a = 0$ pour $a \in d$
pour $t = 1, ..., T$ **faire**
  pour $a \in d$ **faire**
    $\text{UCB}_a = \bar{x}_a + \frac{\beta}{|d|}\sqrt{\frac{2 \ln t}{n_a}}$ ▷ $\text{UCB}_a = \infty$ si $n(a) = 0$
  $\mathcal{M}_t = 3\text{-arg max UCB}_a$ ▷ Selon (3.2)
  $X_t = \text{ROUGE}(\mathcal{M}_t, s)$
  Mettre à jour $\bar{x}_a$ et n_a pour $a \in \mathcal{M}_t$

senter, dans quelle mesure chacune des phrases d_i correspond à une bonne phrase d'un point de vue extractif. Naturellement, on prend [2] $y_i \in [0, 1]$, où un score près de 1 indique que la phrase d_i permet de générer des résumés extractifs de haute qualité.

3.2.1 Utilisation d'un oracle

Sur le jeu de données du CNN/DailyMail (Hermann et al., 2015), les cibles habituellement retenues sont produites par un oracle tel que décrit à la section 1.3.1. Pour un document d et son résumé cible s, l'oracle est une heuristique qui génère un résumé extractif $\dot{s}$ de 3 phrases en sélectionnant une à une les phrases d'un document maximisant le score $\text{ROUGE}(\dot{s}, s)$. Comme l'ordre des phrases n'impacte presque pas le score ROUGE (2.1.2), la performance du résumé choisi par l'oracle peut être considérée comme très près du véritable résumé extractif optimal $s^* = \arg\max_{\hat{s}} \text{ROUGE}(\hat{s}, s)$ d'un document. Ainsi, Nallapati et al. (2017) proposent d'utiliser les cibles binaires indiquant la présence ou non d'une phrase dans le résumé $\dot{s}$ produit par l'oracle, i.e.

$$y_a = \mathbb{I}\big[a \in \dot{s}\big], \quad \text{où} \quad \text{ROUGE}(\dot{s}, s) \approx R^* \tag{3.3}$$

pour $R^* = \text{ROUGE}(s^*, s)$ le score ROUGE du résumé optimal et $\mathbb{I}$ la fonction indicatrice.

En raison de leur nature binaire, les cibles obtenues selon (3.3) peuvent potentiellement être mal spécifiées. En effet, si le résumé $\dot{s}$ produit par l'oracle est presque optimal au sens extractif, cela n'empêche pas que d'autres résumés $\hat{s}'$ peuvent obtenir un score ROUGE très similaire. Or, les phrases de $\hat{s}'$ qui ne font pas partie de $\dot{s}$ se verront attribuer une cible de 0, alors qu'elles représentent tout de même des phrases pouvant générer des résumés presque optimaux. Ce problème d'identification de phrases alternatives presque optimales peut rendre l'entraînement de systèmes de générations de résumés plus difficile.

2. Ici y_i représente la cible associée à la phrase d_i. On utilisera aussi parfois y_a pour représenter la cible associée à la phrase a.

3.2.2 Potentiel extractif d'une phrase

On propose d'adresser directement le problème de l'identification de phrases alternatives en quantifiant directement dans quelle mesure une phrase est susceptible de générer de bons résumés extractifs. Formellement, pour un document d et son résumé cible s, on définit le potentiel extractif v_a d'une phrase a comme

$$v_a := \frac{1}{|\mathcal{S}_d(a)|} \sum_{\hat{s} \in \mathcal{S}_d(a)} \text{ROUGE}(\hat{s}, s), \tag{3.4}$$

où $\mathcal{S}_d(a)$ est l'ensemble des résumés de 3 phrases de d incluant a. Ainsi défini, le potentiel extractif v_a d'une phrase représente donc la moyenne des scores générés par les résumés contenant a. Il est alors naturel d'utiliser des cibles y proportionnelles aux potentiels extractifs des phrases d'un document, i.e. $y_a \propto v_a$, pour incorporer de l'information sur la sous-optimalité des phrases.

Une alternative à (3.4) pour définir le potentiel extractif v_a aurait été de plutôt considérer le score maximal d'un résumé contenant a. Nous avons choisi d'utiliser une moyenne car la moyenne incorpore aussi les mauvais résumés produits par une phrase a. Prenons le cas de deux phrases a et a' similaires et générant toutes deux d'excellents résumés. Le score maximal pour chacune des phrases sera très élevé, mais le score associé à un résumé contenant à la fois a et a' sera plus faible en raison de la répétition d'information. Il n'est donc pas souhaitable d'associer simultanément aux deux phrases un potentiel (une cible) élevé, car cela risquerait seulement d'inciter les résumés contenant a et a'. En utilisant la moyenne, nos potentiels extractifs v_a contournent le problème en considérant aussi les mauvais résumés contenant a, comme ceux incluant des phrases répétitives.

3.2.3 Utilisation de CUCB

L'équation (3.4) ne peut toutefois pas être calculée efficacement en pratique, car elle nécessite de tester tous les résumés possibles d'un document. Elle n'est pas non plus approximée efficacement par pige aléatoire en raison de la nature de la distribution des scores ROUGE des résumés. En effet, la structure inhérente de la génération de résumés extractifs donne lieu à bon nombre de phrases aux potentiels extractifs similaires (quelques mots correspondant au résumé cible) et peu d'excellentes phrases, requérant l'exploration des résumés presque optimaux pour être discernées.

Pour approximer les valeurs de v_a, il est donc nécessaire d'explorer de bons résumés pour distinguer les phrases au potentiel extractif élevé. Nous proposons l'utilisation de CUCB (3.2) à cet effet. Pour un document d, on exécute T pas de temps CUCB sur le bandit combinatoire représentant la génération du résumé de d. En minimisant le pseudo-regret cumulatif (3.1),

CUCB identifie graduellement les bons résumés et parvient à trouver les phrases au potentiel extractif élevé. Parallèlement, CUCB maintient aussi une moyenne empirique $\bar{x}_a(T)$ [3] des scores ROUGE reçus en sélectionnant la phrase a, laquelle peut donc servir d'approximation de v_a.

Concrètement, on propose donc d'utiliser les moyennes $\bar{x}_a$ produites par CUCB comme approximations des potentiels extractifs v_a pour générer les cibles y_a selon le processus suivant. On commence par appliquer une mise à l'échelle min-max des moyennes $\bar{x}_a$. Cette mise à l'échelle produit des moyennes normalisées $\bar{x}'_a \in [0, 1]$ en conservant les proportions originales entre les moyennes. Les $\bar{x}'_a$ sont ensuite utilisées pour produire les cibles y_a selon

$$y_a = 10^{-10(1-\bar{x}'_a)}. \tag{3.5}$$

L'intuition derrière l'équation (3.5) est qu'une phrase avec un potentiel extractif 10 % inférieur à celui de la meilleure phrase devrait se voir attribuer une cible 10 fois moins élevée. Cette mise à l'échelle peut sembler drastique mais il faut se souvenir que l'on ne souhaite accorder d'importance qu'aux phrases qui produisent d'excellents résumés. Enfin, grâce à la mise à l'échelle min-max préalable, une cible de 1 sera toujours associée pour la phrase au meilleur potentiel extractif selon CUCB.

3.2.4 Expériences

On souhaite maintenant vérifier dans quelle mesure CUCB est applicable au processus de génération de résumé d'un document et quelle est son efficacité en fonction du nombre de pas de temps T effectués. On s'intéresse donc à savoir à quel moment le score du résumé le plus prometteur selon les moyennes $\bar{x}_a(T)$, nommons le Y_T, se rapproche du score Y^* du résumé extractif optimal. En mesurant la sous-optimalité $\Delta_T = Y^* - Y_T$ de CUCB, on peut savoir à partir de quel moment les moyennes empiriques obtenues seraient satisfaisantes pour générer des cibles de bonne qualité. On reprend donc le jeu de développement présenté à la section 2.1.1 pour mener des expériences sur la rapidité de l'identification de résumés presque optimaux par CUCB. En fonction des résultats sur la convergence de CUCB, on présente aussi un aperçu des cibles générées par CUCB selon (3.5) et leur comparaison aux cibles issues de l'oracle (3.3) sur le jeu d'entraînement du CNN/DailyMail.

3.2.5 Résultats

Les figures 3.1 et 3.2 présentent respectivement l'impact du paramètre d'exploration β et de la taille des documents sur l'évolution de la sous-optimalité Δ_T des meilleurs résumés selon CUCB. Il est à noter que les différences rapportées entre les différentes courbes sur les figures

3. Pour alléger la notation, on utilisera $\bar{x}_a$ à l'avenir lorsqu'il s'agit clairement des moyennes empiriques après T pas de temps.

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.

La figure 3.1 présente comment l'hyperparamètre β influence la convergence de CUCB. On remarque que $\beta = 10$ (courbe verte sur la figure) semble être le bon compromis entre exploration et exploitation, réussissant à converger à $\Delta_T \approx 7$ après 100 pas de temps de CUCB.

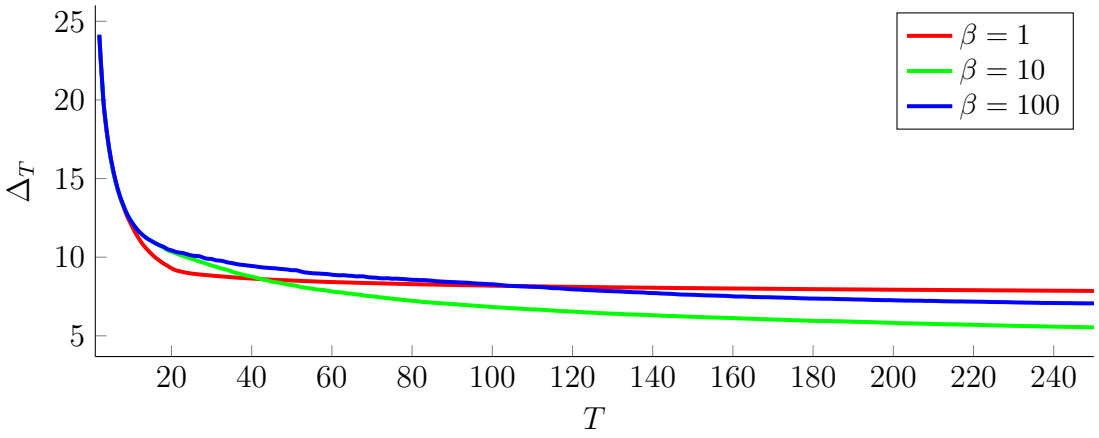

FIGURE 3.1 – Impact du paramètre d'exploration β utilisé par CUCB sur la convergence. Δ_T (plus bas est meilleur) représente la différence entre le score ROUGE du meilleur résumé extractif d'un document et celui suggéré CUCB après T pas de temps.

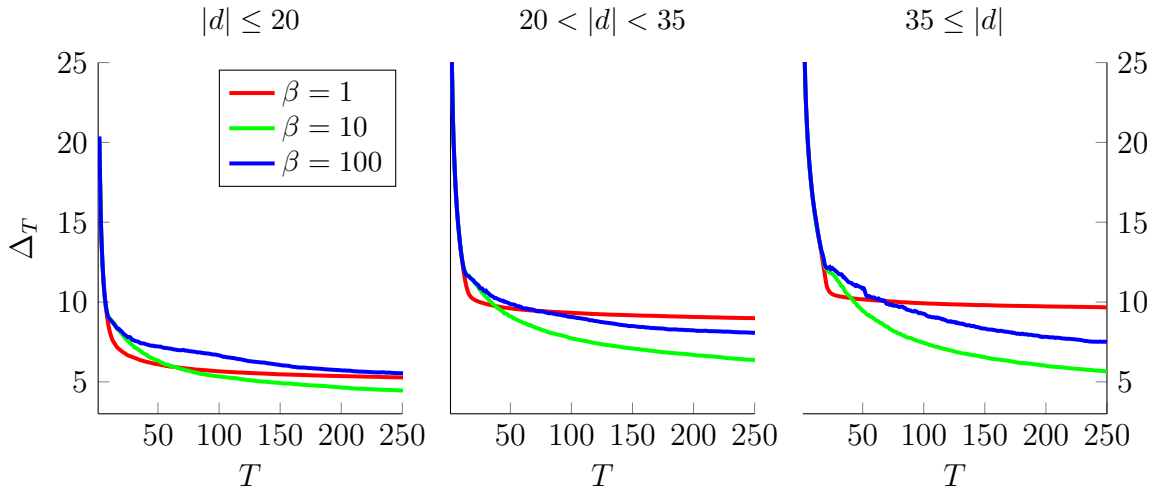

FIGURE 3.2 – Impact de la taille du document sur la convergence de CUCB. Δ_T (plus bas est meilleur) représente la différence entre le score ROUGE du meilleur résumé extractif d'un document et celui suggéré CUCB après T pas de temps.

La figure 3.2 évalue dans quelle mesure la taille des documents influe sur la sous-optimalité des cibles générées par CUCB. On remarque d'abord que $\beta = 10$ (courbe verte) est encore

une fois la meilleure configuration, peu importe la taille des documents. 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 pas de temps de CUCB en fonction du nombre de phrases dans un document. À cette fin, on remarque que la performance stagne autour de 100 pas de temps pour les documents courts mais continue à augmenter jusqu'à 250 pas de temps pour les longs documents.

On explore donc deux fonctions $T(d)$ déterminant le nombre T de pas de temps utilisés pour générer les cibles CUCB pour un document d. Premièrement, on considère une version fixe $T_f(d) = 100$, correspondant à une bonne estimation pour toutes les tailles de document. On considère aussi une version $T_l(d) = 2|d| + 50$ augmentant linéairement selon le nombre de phrases d'un document. Comme notre régularisation limite les documents à 50 phrases ou moins, $T_l(d)$ varie entre 50 et 150, avec une moyenne (sur les tailles de document) à 100. Ainsi, T_f et T_l utiliseront le même nombre de pas de temps en moyenne et leur comparaison est équitable.

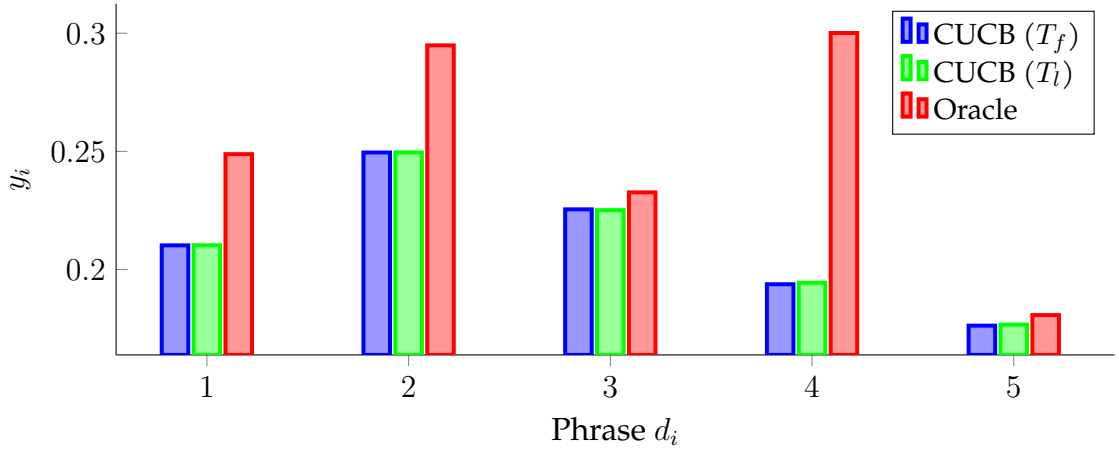

FIGURE 3.3 – Distribution moyenne des cibles y produites par CUCB sur les phrases d'un document du jeu d'entraînement du jeu de données CNN/DailyMail.

Pour mettre à l'échelle les cibles générées par l'oracle et celles générées par CUCB dans la figure 3.3, les cibles CUCB de chaque document sont normalisées pour avoir une somme de 3, comme les cibles binaires. Le premier constat est le très faible impact de l'utilisation de T_f ou T_l sur les cibles générées par CUCB. Aussi, comme il fallait s'y attendre, les cibles générées par CUCB sont assez différentes de celles générées par l'oracle. Notons simplement que la différence entre les cibles peut être attribuée à deux facteurs distincts. Premièrement, CUCB ne converge pas, la plupart du temps, au résumé optimal d'un document. Ainsi, les phrases auxquelles CUCB attribuera une cible de 1 seront, la plupart du temps, différentes de celles déterminées par l'oracle. L'autre facteur important de différence entre les cibles est l'incorporation de scores non-nuls pour les phrases sous-optimales par CUCB.

3.3 CombiSum

On propose maintenant un algorithme basé sur les cibles générées par CUCB : CombiSum. Conformément au cadre expérimental uniformisé présenté à la section 2.1, CombiSum réutilise exactement la même architecture neuronale que BanditSum. Comme les méthodes de l'état de l'art sur l'apprentissage par cibles (Liu and Lapata, 2019), la mise à jour des paramètres est faite selon la perte basée sur l'entropie croisée binaire entre une cible y et le vecteur d'affinités produit $\hat{y} = \pi_\theta(d)$:

$$l(\hat{y}, y) = \frac{1}{|y|} \sum_{i=1}^{|y|} \big[y_i \ln(\hat{y}_i) + (1 - y_i) \ln(1 - \hat{y}_i) \big]. \tag{3.6}$$

Une description détaillée de CombiSum est fournie à l'algorithme 3. Pour chaque document d d'une *minibatch* d'exemples, CombiSum exécute $T(d)$ pas de temps de CUCB, obtient une cible y selon (3.5) et met à jour les poids θ du modèle neuronal selon (3.6).

Algorithme 3 CombiSum

Entrée: $\mathcal{D}$ (jeu de données), T (fonction pour le nombre de pas de temps), α (taux d'apprentissage), β (taux d'exploration CUCB), B (taille de minibatch).

tant que critère d'arrêt non atteint **faire**
  batch $\sim \mathcal{D}^B$ ▷ On pige la minibatch du jeu de données
  $\nabla = \mathbf{0}$
  pour tout $(d,s) \in$ batch **faire**
    $\hat{y} = \pi_\theta(d)$
    Exécuter $T(d)$ pas de temps de CUCB et obtenir $\bar{x}$.
    Générer les cibles y à partir de $\bar{x}$ ▷ Selon (3.5)
    $\nabla = \nabla + \nabla_\theta l(\hat{y}, y)$ ▷ Selon (3.6)
  $\theta = \theta - \alpha\nabla$

3.3.1 Expériences

On effectue un entraînement sur le jeu de données du CNN/DailyMail en parcourant dans son entièreté le jeu d'entraînement à 5 reprises et en utilisant des *minibatches* de taille $B = 64$. Notre implémentation est faite selon les détails expérimentaux décrits à la section 2.1.3. En se basant sur les résultats empiriques présentés à la section 3.2.5, on choisit de fixer $\beta = 10$ pour toutes nos expériences.

Pour les expériences, on s'intéresse à l'impact de la fonction T utilisée. On expérimente donc avec $T = T_f$ et $T = T_l$, tel que décrits à la section 3.2.5. En guise de référence, on entraîne aussi un modèle en utilisant les cibles binaires générées par l'oracle selon (3.3). Étant donné

le facteur aléatoire présent dans l'entraînement, on effectue 5 entraînements distincts pour chaque cible utilisée et présentons la moyenne des résultats obtenus.

3.3.2 Résultats

On présente à la figure 3.4 l'évolution du score ROUGE moyen obtenu en validation par CombiSum selon les cibles employées. Le tableau 3.1 présente quant à lui une comparaison entre le score ROUGE sur le jeu de test des modèles entraînés par les différentes cibles.

On constate à la figure 3.4 que la performance en validation avec les cibles CUCB atteint rapidement un plateau à partir duquel elle ne varie plus. Ce plateau n'affecte toutefois pas l'entraînement avec les cibles binaires, avec lesquelles l'apprentissage continue de varier jusqu'à éventuellement dépasser légèrement la performance des cibles CUCB.

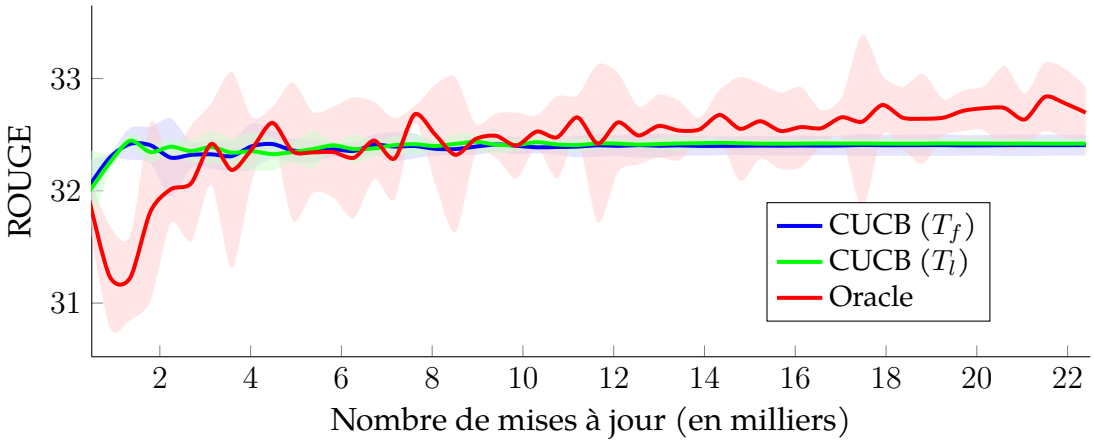

FIGURE 3.4 – Score ROUGE (plus élevé est meilleur) moyen de CombiSum en validation selon la cible utilisée. Un intervalle de confiance à 95 % calculé à partir de 5 entraînements distincts est rapporté pour chaque cible.

Le tableau 3.1 présente la performance obtenue par CombiSum avec les différentes cibles et par la référence Lead-3 sur le jeu de test. Pour chaque entraînement effectué, la performance en test rapportée est celle obtenue avec les paramètres θ permettant d'obtenir la meilleure performance en validation.

Un premier constat qui se pose est que, peu importe la cible utilisée, les modèles produits se comportent de manière extrêmement similaire sur le jeu de test. Aussi, la performance obtenue n'est que très légèrement supérieure à Lead-3. En somme, les résultats obtenus par CombiSum sont décevants, ne se distinguant que très légèrement de la référence Lead-3. On élabore davantage sur les potentielles justifications de cette performance et on discute d'une piste de solution envisageable à la section 4.4.

Cible utilisée	ROUGE-1	ROUGE-2	ROUGE-L	ROUGE
Lead-3	39.59	17.68	36.21	31.16
CUCB (T_f)	40.04 ± 0.11	$\mathbf{18.18 \pm 0.14}$	36.46 ± 0.15	$\mathbf{31.56 \pm 0.13}$
CUCB (T_l)	40.07 ± 0.09	$\mathbf{18.21 \pm 0.10}$	36.49 ± 0.10	$\mathbf{31.59 \pm 0.10}$
Oracle	$\mathbf{40.47 \pm 0.27}$	$\mathbf{18.21 \pm 0.15}$	$\mathbf{36.93 \pm 0.24}$	$\mathbf{31.87 \pm 0.22}$

Tableau 3.1 – Score ROUGE (plus élevé est meilleur) de CombiSum sur le jeu de test selon la cible utilisée

3.4 Conclusion

On a débuté ce chapitre en présentant le bandit combinatoire et son application au problème de la génération du résumé d'un document. On a ensuite introduit CUCB, un algorithme permettant de résoudre le bandit combinatoire proposé. On a introduit la notion de potentiel extractif d'une phrase, présentée comme le score moyen des résumés extractifs incluant la phrase. On a proposé l'utilisation de CUCB pour estimer de manière efficace le potentiel extractif d'une phrase et générer des cibles d'entraînement. L'applicabilité de CUCB à l'estimation du potentiel extractif d'une phrase a été validée empiriquement sur le jeu de développement, où on a démontré que CUCB identifie, en moyenne, un résumé inférieur d'environ 7 points ROUGE au résumé optimal après avoir observé les scores associés à 100 résumés. Les cibles proposées sont intuitivement plus riches que les cibles binaires habituelles générées par un oracle, car elles permettent d'incorporer de l'information sur les phrases qui ne font pas partie du résumé optimal d'un document. Enfin, on a présenté CombiSum, un algorithme d'entraînement de modèles de génération de résumés qui incorpore les cibles générées par CUCB. CombiSum a été mis à l'essai sur le jeu de données du CNN/DailyMail, où on a comparé sa performance à celle obtenue en utilisant les cibles binaires issues d'un oracle. Que l'on utilise les cibles issues de CUCB ou celles de l'oracle, les modèles produits obtiennent tous une performance similaire et seulement légèrement supérieure à l'heuristique de référence Lead-3.

En somme, l'application de la formulation en bandit combinatoire au problème de la génération du résumé d'un document permet de générer des cibles incorporant la notion de potentiel extractif propre à chaque phrase. Il n'est toutefois pas encore clair que ces cibles soient meilleures que les cibles binaires habituelles basées sur un oracle du point de vue de la performance des modèles produits.

Chapitre 4

Bandit linéaire combinatoire

L'approche combinatoire pure présentée au chapitre 3 comporte une amélioration notoire et facilement accessible : l'utilisation des relations de similarité entre les phrases d'un même document. Dans ce chapitre, on présente comment la formulation en bandit linéaire combinatoire (Chen et al., 2018) peut être appliquée à la génération du résumé d'un document et permettre d'exploiter ces similarités entre les phrases. Pour ce faire, on propose une procédure permettant d'obtenir des représentations vectorielles de chaque phrase et résumé. On introduit ensuite LinCUCB, la version linéaire de Combinatorial Upper Confidence Bound (CUCB) (Chen et al., 2013), qui permet de résoudre le bandit linéaire combinatoire de manière plus efficace grâce aux représentations vectorielles. On utilise le jeu de développement pour prouver l'applicabilité et l'efficacité computationnelle de LinCUCB. Similairement à ce qui a été fait avec CUCB, on montre comment LinCUCB peut être utilisé pour produire des cibles d'entraînement pour un document basées sur le potentiel extractif de chacune de ses phrases. On présente enfin LinCombiSum, un algorithme basé sur les cibles produites à partir de LinCUCB pour l'entraînement de modèles de génération de résumés. On met LinCombiSum à l'épreuve sur notre cadre expérimental uniformisé en comparant sa performance à celle obtenue avec les cibles binaires issues d'un oracle pour juger de la qualité des cibles proposées. On termine en analysant de manière approfondie l'impact des cibles utilisées sur la rapidité de convergence et la nature des modèles produits par CombiSum (chapitre 3) et LinCombiSum.

4.1 Bandit linéaire combinatoire

Le bandit linéaire combinatoire n'est rien de plus que l'application du principe combinatoire au bandit stochastique linéaire (1.1.3). Comme on a déjà présenté l'application du principe combinatoire au problème de bandit stochastique à la section 3.1, on se contente ici de rapporter les différences clés introduites par la formulation linéaire et les modifications aux équations pertinentes.

Pour un problème de bandit linéaire, on dit que l'on dispose de représentations vectorielles $\tilde{a} \in \tilde{\mathcal{A}} \subseteq \mathbb{R}^n$ des actions qui peuvent être sélectionnées par l'apprenant. L'hypothèse linéaire est faite : on suppose qu'il existe un vecteur de récompense ω_* reliant une action $\tilde{a}$ à son espérance de récompense μ_a, i.e. $\langle \omega_*, \tilde{a} \rangle \approx \mu_a$ pour tout a.

Cette formulation se généralise à la variante multitâche (Lattimore and Szepesvári, 2020) du bandit combinatoire, où l'apprenant sélectionne plutôt des *super-actions* $\mathcal{M} = \{a_{i_1}, ..., a_{i_m}\}$ de m actions. À chaque pas de temps t, l'environnement produit une récompense $X_{\tilde{\mathcal{M}},t}$ pour toutes les *super-actions* $\mathcal{M}$. Dans ce contexte, l'hypothèse linéaire revient maintenant à supposer que l'on dispose de représentations vectorielles $\tilde{\mathcal{M}} \in \mathbb{R}^n$ pour lesquelles il existe un vecteur de récompense ω_* reliant chaque *super-action* à sa récompense espérée $\mu_{\mathcal{M}}$, i.e. $\langle \omega_*, \tilde{\mathcal{M}} \rangle \approx \mu_{\mathcal{M}}$.

L'objectif demeure la minimisation du pseudo-regret cumulatif (1.3), mais celui-ci est désormais calculé entre la *super-action* optimale $\tilde{\mathcal{M}}^*$ et les *super-actions* $\tilde{\mathcal{M}}_t$ choisies

$$R_T = \sum_{t=1}^{T} \langle \omega_*, \tilde{\mathcal{M}}^* - \tilde{\mathcal{M}}_t \rangle. \tag{4.1}$$

Tout l'intérêt de la formulation linéaire combinatoire vient de l'exploitation des représentations vectorielles $\tilde{\mathcal{M}}$. Si l'hypothèse linéaire est respectée, les similarités entre les *super-actions* peuvent être exploitées pour extraire de l'information sur toutes les *super-actions* $\mathcal{M}$ à partir de la *super-action* $\mathcal{M}_t$ sélectionnée au temps t. Par exemple, pour deux *super-actions* dotées de représentations vectorielles $\tilde{\mathcal{M}}$ et $\tilde{\mathcal{M}}'$ similaires, l'hypothèse linéaire indique que les deux *super-actions* devraient avoir une espérance de récompense similaire. On peut donc considérer qu'une récompense reçue en sélectionnant $\mathcal{M}$ permet de donner un bon aperçu de ce que la *super-action* $\mathcal{M}'$ semblable aurait pu recevoir et exploiter toutes les similarités entre les *super-actions* pour accélérer la résolution du problème. Intuitivement, cela revient à dire qu'il devrait être possible de résoudre la version linéaire du bandit combinatoire en un nombre de pas de temps moins importants grâce à l'exploitation de l'hypothèse linéaire.

4.1.1 Application à la génération de résumé

Comme présenté précédemment à la section 3.1.1, la génération du résumé d'un document d peut être vue comme un bandit combinatoire où les actions $a \in \mathcal{A}$ sont les phrases $d_i \in d$ du document et les *super-actions* $\mathcal{M} = \{a_{i_1},...,a_{i_3}\}$ sont les résumés non-ordonnés de 3 phrases possibles. Dans le contexte linéaire considéré dans ce chapitre, on cherche aussi à obtenir des représentation vectorielles $\tilde{a}$ des phrases et $\tilde{\mathcal{M}}$ des résumés.

Représentations vectorielles des phrases

On prend appui sur la nature du score ROUGE (section 2.1.2) utilisé comme récompense pour bâtir les représentations vectorielles des phrases. Comme les calculs de ROUGE sont basés sur les comptes de n-grammes entre les phrases d'un résumé produit et d'un résumé cible, il est naturel de baser la construction du vecteur d'une phrase sur les n-grammes qui la composent. Concrètement, on choisit de générer, pour chaque phrase a d'un document d, sa représentation en sac de mots. Le sac de mots d'une phrase est 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. On obtient donc, pour chaque phrase a, un vecteur parcimonieux $\tilde{a}$ de taille N représentant le nombre d'occurrences de chaque n-gramme dans la phrase, pour N le nombre de n-grammes différents dans le document. Dans nos expériences, on limite l'exploration des n-grammes aux unigrammes, afin de garder une taille raisonnable pour les vecteurs parcimonieux générés. Bien qu'elle ne permet pas d'incorporer toute l'information requise pour le calcul du ROUGE, cette représentation basée uniquement sur les unigrammes permet d'avoir une approximation suffisante pour nos besoins.

On bâtit ensuite une matrice parcimonieuse dont les rangées sont les vecteurs parcimonieux de chaque phrase du document. La dimension de la matrice est ensuite réduite via une analyse sémantique latente (LSA) (Kolda and O'leary, 1998), applicable naturellement dans notre cas de matrice parcimonieuse de grande taille. Le processus retourne un vecteur de taille $|d|$ pour chaque phrase, que l'on normalise pour avoir une norme unitaire et qu'on utilise pour les représentations vectorielles $\tilde{a}$. On note ici que, si aucun n-gramme n'était partagé entre les phrases, leurs représentations vectorielles $\tilde{a}$ seraient les vecteurs unitaires orthogonaux e_a et, comme mentionné à la section 1.1.4, les vecteurs ne permettraient pas d'accélérer l'apprentissage.

Représentations vectorielles de résumés

Maintenant que l'on dispose de représentations $\tilde{a}$ pour chaque phrase a, on s'intéresse à la manière de les regrouper pour obtenir une représentation $\tilde{\mathcal{M}}$ d'un résumé $\mathcal{M}$. On reprend ici l'hypothèse présentée à la section 3.1.1 selon laquelle chaque phrase d'un résumé contribue à part égale au score du résumé complet. Cela revient à dire que le score associé à un résumé n'est que la moyenne des scores associables à chacune des phrases qu'il contient. Dans le contexte de représentations vectorielles, on pose alors naturellement la représentation d'un résumé comme la moyenne de la représentation de ses phrases

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

Pour un document d et son résumé cible s, l'hypothèse linéaire devient alors l'existence d'un vecteur ω_* reliant chaque résumé $\mathcal{M}$ à son score ROUGE$(\mathcal{M}, s)$ correspondant, i.e. $\langle \omega_*, \tilde{\mathcal{M}} \rangle \approx$ ROUGE$(\mathcal{M}, s)$.

4.1.2 LinUCB pour bandit linéaire combinatoire (LinCUCB)

Selon le même argument qu'à la section 3.1.2 où on appliquait UCB au problème combinatoire pour trouver l'algorithme CUCB, LinUCB peut être adapté au problème multitâche linéaire en sélectionnant

$$\mathcal{M}_t = \text{3-}\arg\max_{\tilde{a} \in \tilde{\mathcal{A}}} \text{UCB}_a(t-1), \tag{4.2}$$

où $\text{UCB}_a(t-1)$ représente la borne supérieure associée à l'action a par LinUCB selon (1.9).

De manière analogue, on dira que l'apprenant sélectionnant les *super-actions* d'un bandit linéaire combinatoire selon (4.2) correspond à un algorithme que l'on choisit de nommer LinCUCB. Le pseudocode de LinCUCB se trouve à l'algorithme 4. Notons que, pour mettre LinCUCB à l'échelle de chaque document d, l'exploration que l'on utilise est plutôt guidée par $\beta' = \frac{\beta}{|d|}$.

Algorithme 4 LinCUCB pour génération de résumé

Entrée: d (document), s (résumé cible), β (paramètre d'exploration), T (nombre de pas de temps), $\tilde{\mathcal{A}}$ (représentations des phrases).

$\mathbf{V} = \mathbf{I}$
$\mathbf{b} = \mathbf{0}$
pour $t = 1, ..., T$ **faire**
  $\hat{\omega}_t = \mathbf{V}^{-1}\mathbf{b}$
  pour $\tilde{a} \in \tilde{\mathcal{A}}$ **faire**
    $\text{UCB}_a = \langle \hat{\omega}_t^\top, \tilde{a} \rangle + \frac{\beta}{|d|}\sqrt{\tilde{a}^\top \mathbf{V}^{-1} \tilde{a}}$
  $\mathcal{M}_t = \text{3-}\arg\max \text{UCB}_a$ ▷ Selon (4.2)
  $X_t = \text{ROUGE}(\mathcal{M}_t, s)$
  pour tout $a \in \mathcal{M}_t$ **faire**
    $\mathbf{V} = \mathbf{V} + \tilde{a}\tilde{a}^\top$
    $\mathbf{b} = \mathbf{b} + \tilde{a}X_t$

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

4.2 Génération de cibles par LinCUCB

La génération de cibles avec LinCUCB est presque identique à celle avec CUCB présentée à la section 3.2. Dans notre cas, comme LinCUCB ne garantit plus l'exploration de chacune des actions, on utilise alors les approximations fournies par $\langle \omega_T, \tilde{a} \rangle$ au lieu des moyennes empiriques $\bar{x}_a(T)$ comme estimations du potentiel extractif de chaque phrase. Une mise à l'échelle min-max est encore une fois utilisée pour normaliser les valeurs estimées $\langle \omega_T, \tilde{a} \rangle$. Cette mise à l'échelle produit des estimés normalisés $x_{\tilde{a}} \in [0, 1]$ conservant les proportions originales entre les moyennes. Les $x_{\tilde{a}}$ sont ensuite utilisés pour produire les cibles y_a selon

$$y_a = 10^{-10(1-x_{\tilde{a}})}, \tag{4.3}$$

où l'intuition est encore qu'une phrase avec un potentiel extractif 10 % inférieur à celui de la meilleure phrase devrait se voir attribuer une cible 10 fois moins élevée. Enfin, grâce à la mise à l'échelle min-max préalable, une cible de 1 sera toujours associée pour la phrase au meilleur potentiel extractif selon LinCUCB.

4.2.1 Expériences

Similairement à ce qui a été fait à la section 3.2.4, on désire valider l'applicabilité de LinCUCB à la formulation linéaire combinatoire présentée à partir des documents du jeu de développement. On définit Y_T comme le score du résumé le plus prometteur selon LinCUCB et Y^* le score du résumé extractif optimal d'un document. On mesure l'évolution de la sous-optimalité $\Delta_T = Y^* - Y_T$ des meilleurs résumés selon LinCUCB pour établir la rapidité de la convergence. Comme l'intérêt est d'établir la plus grande efficacité de LinCUCB en nombre de pas de temps T requis pour la convergence, on compare la performance de LinCUCB à celle de CUCB, qui n'utilise aucune représentation vectorielle. En fonction des résultats sur la convergence de LinCUCB, on présente aussi un aperçu des cibles générées par LinCUCB selon (4.3) et leur comparaison aux cibles issues de l'oracle (3.3) sur le jeu d'entraînement du CNN/DailyMail.

4.2.2 Résultats

Les figures 4.1 et 4.2 présentent respectivement l'impact du paramètre d'exploration β et de la taille des documents sur l'évolution de la sous-optimalité Δ_T des meilleurs résumés selon LinCUCB. 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.

La figure 4.1 présente comment l'hyperparamètre β influence la convergence de LinCUCB.

On remarque d'abord que, comme il fallait s'y attendre, LinCUCB converge nettement plus rapidement que CUCB. En effet, alors que CUCB atteint une sous-optimalité $\Delta_T \approx 7$ après 100 pas de temps, LinCUCB obtient atteint $\Delta_T \approx 5$ apres seulement 50 pas de temps. On remarque aussi que $\beta = 10^8$ (courbe bleue sur la figure) semble être un bon choix pour l'exploration, réussissant à converger légèrement plus rapidement que $\beta = 10^7$.

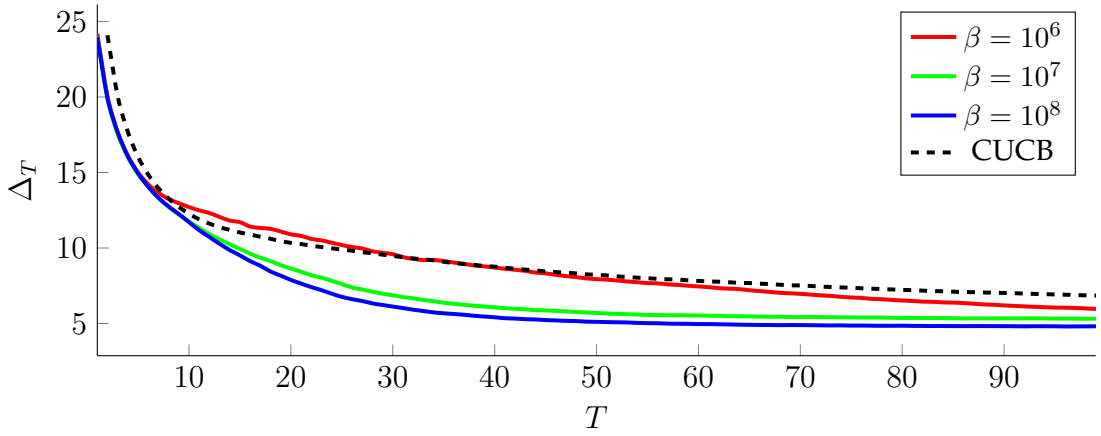

FIGURE 4.1 – Impact du paramètre d'exploration β utilisé par LinCUCB sur la convergence. Δ_T (plus bas est meilleur) représente la différence entre le score ROUGE du meilleur résumé extractif d'un document et celui suggéré CUCB après T pas de temps.

À la figure 3.2, on constate que 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 CUCB, il semble que l'exécution de LinUCB avec plus de pas de temps est bénéfique pour les documents plus longs.

En se basant sur ces résultats, on explore deux fonctions $T(d)$ déterminant le nombre T de pas de temps utilisés pour générer les cibles LinCUCB pour un document d. Premièrement, on considère une version fixe $T_f(d) = 50$, correspondant à une bonne estimation pour toutes les tailles de document. On considère aussi une version $T_l(d) = |d| + 25$ augmentant linéairement selon le nombre de phrases d'un document. Comme notre régularisation limite les documents à 50 phrases ou moins, $T_l(d)$ varie entre 25 et 75, avec une moyenne (sur les tailles de document) à 50. Ainsi, T_f et T_l utiliseront le même nombre de pas de temps en moyenne et il sera possible de valider laquelle des deux fonctions permet le meilleur entraînement.

Pour mettre à l'échelle les cibles générées par l'oracle et celles générées par LinCUCB dans la figure 4.3, les cibles LinCUCB de chaque document sont normalisées pour avoir une somme de 3, comme les cibles binaires. Le premier constat est le faible impact de l'utilisation de T_f ou T_l sur les cibles générées par LinCUCB. On note toutefois que l'impact est plus élevé que pour les cibles générées pas CUCB affichées à la figure 3.3.

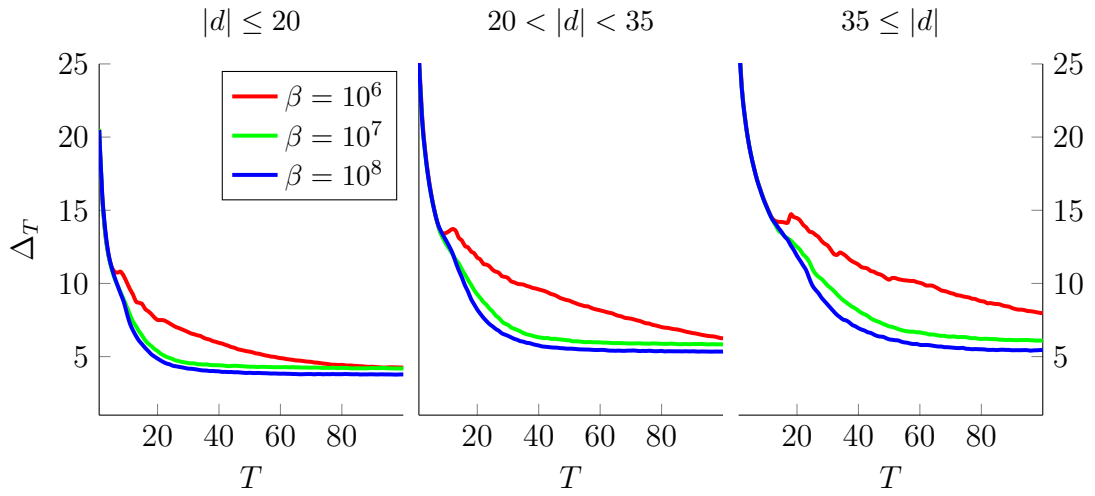

FIGURE 4.2 – Impact de la taille du document sur la convergence de LinCUCB. Δ_T (plus bas est meilleur) représente la différence entre le score ROUGE du meilleur résumé extractif d'un document et celui suggéré CUCB après T pas de temps.

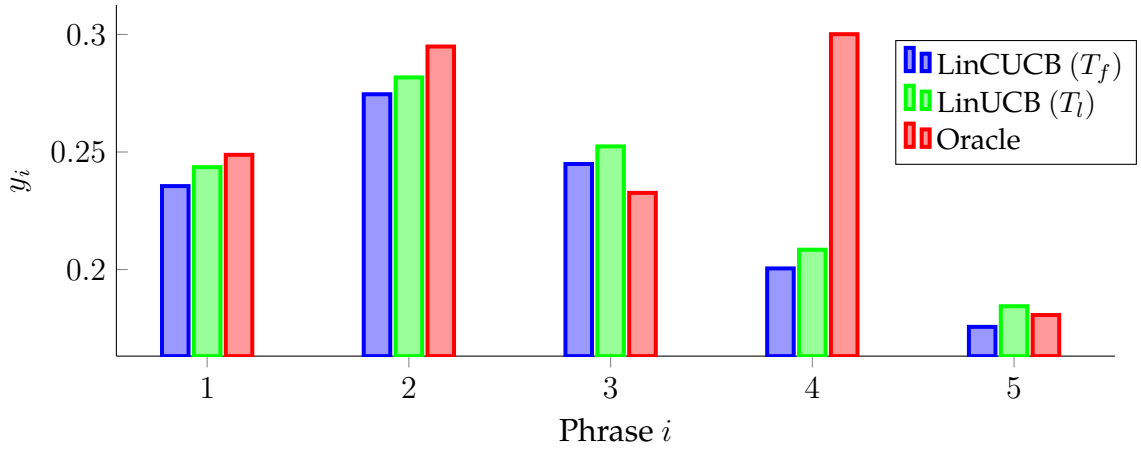

FIGURE 4.3 – Distribution moyenne des cibles y produites par LinCUCB sur les phrases d'un document du jeu d'entraînement du jeu de données CNN/DailyMail.

En contraste à ce qui avait été observé pour les cibles CUCB, les cibles LinCUCB sont très près de celles générées par l'oracle, la seule différence notoire étant dans la distribution de la 4ème phrase. En vue de la convergence rapide de LinCUCB, on fait l'hypothèse que LinCUCB parvient à identifier plus souvent le résumé généré par l'oracle et que les variations sont principalement dûes à la nature plus riche des cibles proposées.

4.3 LinCombiSum

On propose maintenant un système de génération de résumés complet nommé LinCombiSum, qui se veut identique à CombiSum (3.3) mais qui utilise LinCUCB au lieu de CUCB

pour générer ses cibles selon l'équation 4.3. Conformément aux méthodes de l'état de l'art sur l'apprentissage de cibles (Liu and Lapata, 2019), la mise à jour des paramètres est faite selon la perte basée sur l'entropie croisée binaire entre une cible y et un vecteur d'affinités produit $\hat{y} = \pi_\theta(d)$:

$$l(\hat{y}, y) = \frac{1}{|y|} \sum_{i=1}^{|y|} \big[y_i \ln(\hat{y}_i) + (1 - y_i) \ln(1 - \hat{y}_i) \big]. \tag{4.4}$$

Une description détaillée de LinCombiSum est fournie à l'algorithme 5. Pour chaque document d d'une *minibatch* d'exemples, LinCombiSum exécute $T(d)$ pas de temps de LinCUCB, obtient une cible y selon (4.3) et met à jour les poids θ du modèle neuronal selon (4.4).

Algorithme 5 LinCombiSum

Entrée: $\mathcal{D}$ (jeu de données), T (fonction pour le nombre de pas de temps), α (taux d'apprentissage), β (taux d'exploration LinCUCB), B (taille de minibatch).

tant que critère d'arrêt non atteint **faire**
  batch $\sim \mathcal{D}^B$ ▷ On pige la minibatch du jeu de données
  $\nabla = \mathbf{0}$
  pour tout $(d,s) \in$ batch **faire**
    $\hat{y} = \pi_\theta(d)$
    Obtenir les représentations $\tilde{a}$ des phrases de d ▷ Section (4.1.1)
    Exécuter $T(d)$ pas de temps de LinCUCB et obtenir $\langle \hat{\omega}_T, \tilde{a} \rangle$
    Générer les cibles y à partir de $\langle \hat{\omega}_T, \tilde{a} \rangle$ ▷ Selon (4.3)
    $\nabla = \nabla + \nabla_\theta l(\hat{y}, y)$ ▷ Selon (4.4)
  $\theta = \theta - \alpha \nabla$

4.3.1 Expériences

On effectue un entraînement sur le jeu de données CNN/DailyMail en parcourant dans son entièreté le jeu d'entraînement à 5 reprises et en utilisant des *minibatches* de taille $B = 64$. Notre implémentation est faite selon les détails expérimentaux décrits à la section 2.1.3. En se basant sur les résultats empiriques présentés à la section 4.2.2, on choisit de fixer $\beta = 10^8$ pour toutes nos expériences.

Pour les expériences, on s'intéresse à l'impact de la fonction T utilisée. On expérimente donc avec $T = T_f$ et $T = T_l$, tel que décrits plus haut. En guise de référence, on entraîne aussi un modèle en utilisant les cibles binaires générées par l'oracle telles que décrites à la section 3.2.1. Étant donné le facteur aléatoire présent dans l'entraînement, on effectue 5 entraînements distincts pour chaque T et présentons la moyenne des résultats obtenus.

4.3.2 Résultats

On présente à la figure 4.4 l'évolution du score ROUGE moyen obtenu en validation par LinCombiSum en fonction de la méthode de génération des cibles employée. Une comparaison entre le score ROUGE sur le jeu de test des modèles entraînés par les différentes cibles se trouve au tableau 4.1.

Tout d'abord, la figure 4.4 illustre l'évolution de la performance sur le jeu de validation des modèles selon les cibles utilisées. Les constats sont les mêmes que pour les cibles CUCB : la performance stagne rapidement pour les cibles LinCUCB et est éventuellement dépassée par celle obtenue avec les cibles issues de l'oracle.

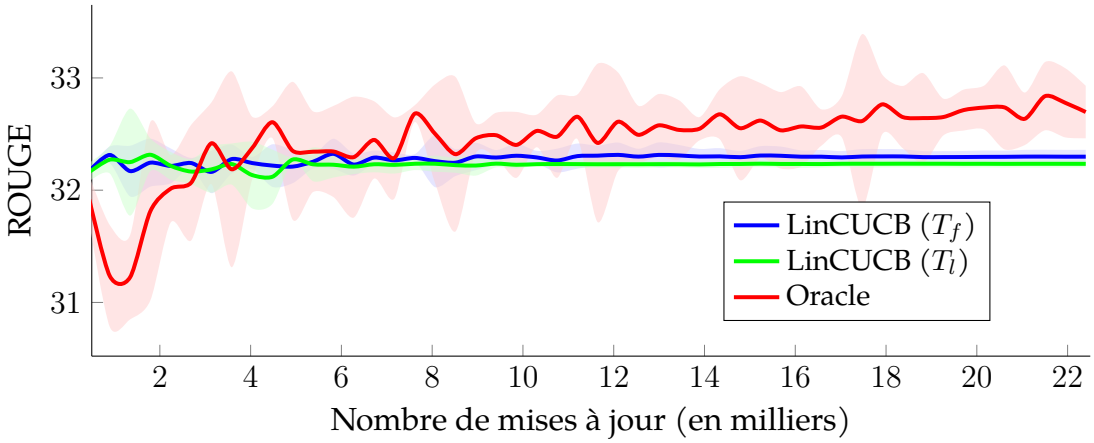

FIGURE 4.4 – Score ROUGE (plus élevé est meilleur) moyen de LinCombiSum en validation selon la cible utilisée. Un intervalle de confiance à 95 % calculé à partir de 5 entraînements distincts est rapporté pour chaque cible.

Le tableau 4.1 présente la performance obtenue par nos modèles et par la référence Lead-3 sur le jeu de test. Pour chaque entraînement effectué, la performance en test rapportée est celle obtenue avec les paramètres θ permettant d'obtenir la meilleure performance en validation.

Cible utilisée	**ROUGE-1**	**ROUGE-2**	**ROUGE-L**	**ROUGE**
Lead-3	39.59	17.68	36.21	31.16
LinCUCB (T_f)	39.91 ± 0.04	$\mathbf{18.11 \pm 0.05}$	36.33 ± 0.06	31.45 ± 0.05
LinCUCB (T_l)	39.84 ± 0.06	$\mathbf{18.03 \pm 0.04}$	36.25 ± 0.06	31.37 ± 0.06
Oracle	$\mathbf{40.47 \pm 0.27}$	$\mathbf{18.21 \pm 0.15}$	$\mathbf{36.93 \pm 0.24}$	$\mathbf{31.87 \pm 0.22}$

Tableau 4.1 – Score ROUGE (plus élevé est meilleur) de LinCombiSum sur le jeu de test selon la cible utilisée

Un premier constat qui se pose est que, peu importe la cible utilisée, les modèles générés par LinCombiSum se comportent de manière extrêmement similaire sur le jeu de test. Aussi, la performance obtenue n'est que très légèrement supérieure à Lead-3. En somme, les résultats obtenus par LinCombiSum sont décevants, ne se distinguant que très légèrement de la référence Lead-3. On élabore davantage sur les potentielles justifications de cette performance et discutons d'une piste de solution envisageable à la prochaine section.

4.4 Impact des cibles sur la convergence

Les nouvelles cibles riches basées sur CUCB et LinCUCB n'ont pas permis d'obtenir des résultats dépassant la référence représentée par les cibles binaires d'un oracle. En effet, peu importe les cibles utilisées, les modèles entraînés semblaient toujours converger à des modèles dont les prédictions étaient extrêmement près de celles l'heuristique Lead-3 (voir tableaux 3.1 et 4.1). On analyse de manière plus approfondie le comportement de nos algorithmes CombiSum et LinCombiSum dans cette section afin de tenter de comprendre à quoi ces résultats décevants peuvent être attribuables. Comme aucune différence statistiquement significative n'avait été observée entre les versions linéaire et fixe des cibles proposées, on limite notre analyse au comportement de la version linéaire de nos cibles.

La figure 4.5 montre d'abord l'évolution de la perte empirique (1.11) à l'entraînement selon la cible utilisée. On voit que les modèles entraînés convergent tous très rapidement à une perte empirique stable, qui est maintenue sur l'ensemble de la procédure d'entraînement. Cela témoigne de l'incapacité du modèle neuronal à apprendre autant les cibles issues de l'oracle que celles basées sur CUCB et LinCUCB.

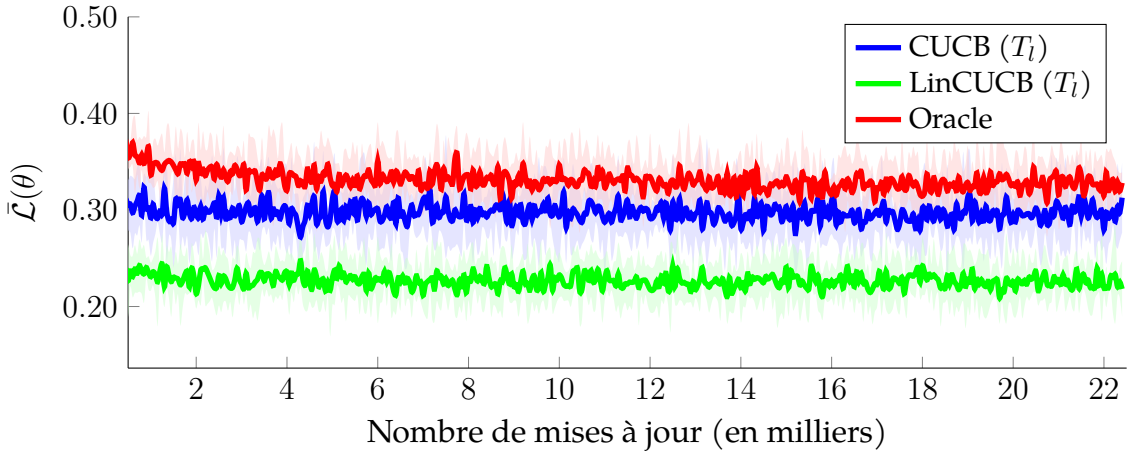

FIGURE 4.5 – Courbe d'évolution de la perte empirique selon la cible utilisée. Un intervalle de confiance à 95 % calculé à partir de 5 entraînements distincts est rapporté pour chaque cible.

La figure 4.6 présente le pourcentage d'inclusion des phrases sur le jeu de test en fonction

des cibles utilisées à l'entraînement. On y voit que la ressemblance de la performance à celle l'heuristique Lead-3 n'est pas le fruit du hasard : peu importe la cible, les modèles entraînés sélectionnent dans une très grande proportion les premières phrases d'un document.

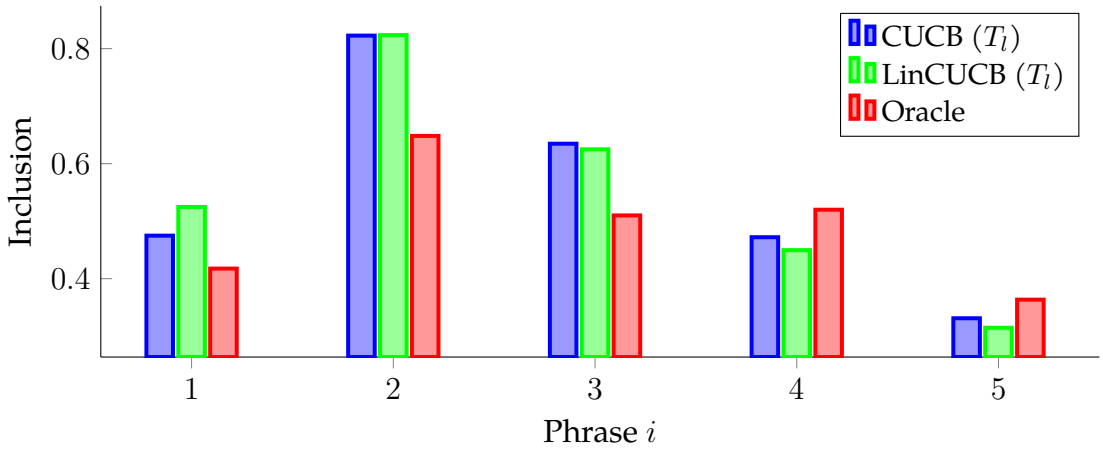

FIGURE 4.6 – Distribution moyenne des phrases retenues sur le jeu de test du jeu de données de CNN/DailyMail.

On fait l'hypothèse qu'une cause plausible de ce comportement est l'attractivité du minimum local qui consiste à prédire les phrases seulement en fonction de leur index. Comme les résumés d'articles de journaux suivent une structure en pyramide inverse où l'information pertinente contenue dans une phrase diminue alors que sa position dans le document augmente, le modèle utilisant toujours les premières phrases d'un document pour générer son résumé sera toujours bon sur ce type de document (Kryściński et al., 2019).

Or, les modèles que l'on entraîne avec les cibles binaires de l'oracle souffrent aussi (dans une moindre mesure) de ce problème. Cela fait contraste avec la performance rapportée par BertSumExt (Liu and Lapata, 2019) qui représente l'état de l'art extractif et qui utilise exactement la même procédure d'entraînement. La seule différence entre nos expériences et BertSumExt est dans le modèle neuronal : on utilise un modèle léger basé sur deux LSTMs (Hochreiter and Schmidhuber, 1997) alors qu'ils utilisent un modèle BERT (Devlin et al., 2018), un ordre de magnitude plus gros. Cela suggère que les mauvaises performances obtenues avec nos cibles sont dues à un manque d'expressivité du réseau de neurones utilisé. Ainsi, en utilisant les cibles basées sur CUCB et LinCUCB avec le modèle utilisé par BertSumExt, il n'y a pas de raison évidente pour que la performance ne soit pas au minimum similaire.

4.5 Conclusion

On a commencé ce chapitre par la présentation du bandit linéaire combinatoire, une formulation permettant d'exploiter la similarité entre les phrases dans le problème de la génération du résumé d'un document. On a proposé des représentations vectorielles de phrases et des

résumés fondées sur les n-grammes et utilisables dans la formulation linéaire. On a présenté LinCUCB, un algorithme qui résout le bandit linéaire combinatoire proposé et validé qu'il permet d'identifier des résumés de qualité comparable à CUCB en deux fois moins de pas de temps sur le jeu de développement. On a ensuite utilisé LinCUCB pour générer des cibles d'entraînement basées sur le potentiel extractif des phrases, que l'on a incorporées à un nouvel algorithme intitulé LinCombiSum. L'application au cadre expérimental uniforme a permis d'établir que, similairement à CombiSum et les cibles binaires générées par un oracle, LinCombiSum produit des modèles dont la performance est seulement marginalement plus élevée que celle de l'heuristique Lead-3. On a investigué davantage la convergence avec CombiSum, LinCombiSum et les cibles générées par un oracle pour établir que la piètre performance est probablement attribuable à la nature de la distribution de l'information dans les documents considérés et un manque d'expressivité dans le modèle neuronal employé.

En somme, ce chapitre a établi comment la génération du résumé d'un document peut être vue comme un bandit linéaire combinatoire, permettant d'utiliser les similarités entre les phrases d'un document pour accélérer la résolution du problème. Les cibles issues de la résolution du bandit linéaire combinatoire par LinCUCB n'ont toutefois pas permis d'atteindre des performances convaincantes sur le jeu de données du CNN/DailyMail, mais il semble que l'utilisation d'un modèle neuronal plus expressif devrait permettre d'exploiter la richesse des cibles proposées.

Conclusion

Dans ce mémoire, on a abordé la problématique de l'entraînement de modèles de génération de résumés de documents. On a considéré les résumés dits extractifs, qui sont bâtis en sélectionnant des phrases du document initial. Pour mener nos expérimentations, on s'est intéressé au jeu de données du CNN/DailyMail (Hermann et al., 2015), regroupant plus de 300 000 articles de journaux et leurs résumés. Notre intérêt était de démontrer comment les approches par bandit (Robbins, 1952) peuvent être mises à profit dans des algorithmes d'entraînement pour la génération de résumés.

Au chapitre 2, on a d'abord détaillé la problématique à l'étude ainsi que le cadre uniformisé utilisé tout au long du document pour comparer la performance des algorithmes considérés. Le cadre uniformisé retenu est de considérer seulement les résumés extractifs de 3 phrases et d'utiliser le même réseau de neurones que BanditSum (Dong et al., 2018). On a aussi mis sur pied un jeu de développement constitué de 25 000 documents sur le jeu de données du CNN/DailyMail qui est utilisé pour déterminer l'applicabilité des formulations bandits considérées. Ensuite, on a présenté BanditSum en détails, qui voit la génération des résumés sur un jeu de données en entier comme un bandit contextuel. BanditSum exploite la formulation contextuelle en utilisant l'algorithme REINFORCE (Williams, 1992) pour entraîner un système de génération de résumés optimisant la qualité des résumés générés en espérance. À partir du jeu de développement, on a établi de manière rigoureuse que le nombre d'échantillons $N = 20$ utilisé par BanditSum pour REINFORCE est approprié. Enfin, on a présenté qu'avec un modèle contenant 25 fois moins de paramètres et avec un entraînement sur 10 fois moins de documents, BanditSum obtient une performance seulement légèrement inférieure aux meilleurs algorithmes de l'état de l'art, établissant sa supériorité en matière de temps de calcul.

Au chapitre 3, on s'est intéressé à un second niveau d'application pour le problème de bandit : l'application à la génération du résumé d'un seul document. On a présenté comment la génération du résumé d'un document peut être vue comme un problème de bandit combinatoire (Chen et al., 2013). On a ensuite introduit la notion de potentiel extractif d'une phrase, utilisable comme cible quantifiable pour l'affinité d'un système de génération de résumés à inclure une phrase dans le résumé d'un document. On a détaillé l'algorithme CUCB (Chen

et al., 2013) permettant de résoudre le bandit combinatoire proposé et pouvant être utilisé pour estimer le potentiel extractif de chaque phrase d'un document. Les cibles basées sur le potentiel extractif des phrases sont utilisées par un nouvel algorithme, nommé CombiSum, que l'on a mis à l'essai sur le jeu de données CNN/DailyMail. Nos résultats ont montré que les cibles proposées ne permettent pas un meilleur entraînement que les cibles basées sur un oracle (Nallapati et al., 2017), car les modèles produits convergent tous vers une performance très similaire à l'heuristique de référence Lead-3 (Nallapati et al., 2017).

Au chapitre 4, on a montré comment il est possible d'incorporer les similarités entre les phrases d'un document pour voir la génération du résumé d'un document comme un bandit linéaire combinatoire (Chen et al., 2018). On a présenté l'algorithme LinCUCB, une variante de CUCB qui permet d'exploiter l'hypothèse linéaire pour résoudre le bandit linéaire combinatoire à un coût de calcul amoindri. On a utilisé LinCUCB pour générer des cibles basées sur le potentiel extractif des phrases d'un document dans un processus plus efficace en termes de coût de calcul que celui basé sur CUCB. L'algorithme LinCombiSum, utilisant les cibles obtenues à partir de LinCUCB, a été présenté et mis à l'épreuve sur le jeu de données du CNN/DailyMail. Les modèles produits par LinCombiSum obtiennent encore une fois une performance très similaire à l'heuristique Lead-3. On a analysé plus en détails la convergence de CombiSum, LinCombiSum et un modèle entraîné selon les cibles issues d'un oracle et prouvé que les modèles produits à partir de toutes les cibles génèrent leurs résumés d'une manière très similaire à l'heuristique Lead-3. On a émis l'hypothèse que cette convergence est dûe à la nature de la distribution de l'information contenue dans les articles de journaux ainsi qu'à l'expressivité limitée du modèle neuronal employé.

En somme, les travaux présentés ont permis d'établir comment les formulations en bandit contextuel, combinatoire et linéaire combinatoire peuvent être utilisées pour donner naissance à des approches par bandit pour l'entraînement de modèle de génération résumés. L'utilisation des bandits aura permis aux algorithmes abordés une efficacité accrue, mais surtout une façon directe d'incorporer la différence de qualité entre deux résumés dans la procédure d'entraînement.

Contributions

Les contributions principales des travaux présentés sont les suivantes :

— **Algorithme d'entraînement CombiSum.** En décrivant le problème de la génération du résumé extractif d'un document comme un bandit combinatoire, CombiSum permet un changement de perspective important sur la production de cibles d'entraînement pour un document. En effet, la formulation combinatoire alloue l'incorporation d'information relative aux résumés non-optimaux d'un document dans la génération de cibles,

en contraste avec les cibles binaires actuelles basées sur le meilleur résumé selon un oracle.

— **Algorithme d'entraînement LinCombiSum.** LinCombiSum exploite la notion de similarité entre les phrases pour visualiser le problème de la génération du résumé extractif d'un document comme un bandit linéaire combinatoire. D'une manière similaire à ce qui est fait par CombiSum, LinCombiSum utilise la formulation en bandit linéaire combinatoire pour générer des cibles d'entraînement plus riches pour chaque document. La génération des cibles est toutefois obtenue à un coût de calcul moins élevé grâce à l'incorporation de similarités entre les phrases d'un même document pour faciliter la résolution du problème de bandit.

Défis et perspectives

Plusieurs modifications pourraient être apportées à nos travaux afin de les améliorer.

Un premier angle à partir duquel il serait possible d'améliorer nos travaux est celui de la relaxation des contraintes sur les résumés extractifs considérés. Notre étude s'arrête au cas où l'on génère un résumé à partir d'un groupe non-ordonné de 3 phrases. Il serait intéressant d'incorporer des formulations plus souples qui permettent de sélectionner un nombre variable de phrases d'un document. Une première piste en ce sens serait de considérer une critère d'arrêt flexible dans le processus de génération de résumés à partir d'affinités, comme le font Luo et al. (2019). On pourrait aussi considérer le cas où les phrases sont sélectionnées les unes après les autres, permettant notamment au modèle d'apprendre directement à éviter la répétition. Cette dernière approche, s'apparentant à l'apprentissage par renforcement (Sutton and Barto, 2018), est d'ailleurs déjà explorée en partie par Zhong et al. (2020).

Une seconde région où il serait possible de faire progresser notre réflexion est celle des algorithmes utilisés pour la génération de cibles. En effet, les algorithmes CUCB et LinCUCB employés sont conçus pour le bandit combinatoire général et n'exploitent donc pas directement la variante multitâche à laquelle on s'intéresse. Des algorithmes comme KMTL-UCB (Deshmukh et al., 2017) et MT-LinUCB (Soare et al., 2014), conçus spécialement pour le bandit multitâche, pourraient notamment être envisagés pour la génération de cibles d'entraînement.

Une troisième façon de bonifier nos travaux serait de voir comment les approches par bandit présentées performent pour l'entraînement de modèles bien plus imposants, comme celui utilisé par Liu and Lapata (2019). En effet, l'architecture de réseau de neurones utilisée par BanditSum et reprise dans nos travaux est très peu expressive en comparaison avec les modèles de l'état de l'art. Bien que cela permette d'établir l'efficacité computationnelle de BanditSum, on croit aussi qu'il peut s'agir de la raison pour laquelle la performance avec nos nouvelles cibles a obtenu des résultats aussi peu convaincants.

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 (1.7). 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

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

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

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

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 $\mathbf{V}_{t-1}^{-1} \tilde{a}_t$ est utilisé à plusieurs reprises. On peut donc entreposer le produit en posant

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

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

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

qui est la version utilisée dans notre implémentation de LinUCB.

Annexe B

Graphiques avec variance

On présente ici, chapitre par chapitre, les versions des graphiques incorporant les déviations standard.

Graphiques du chapitre 2

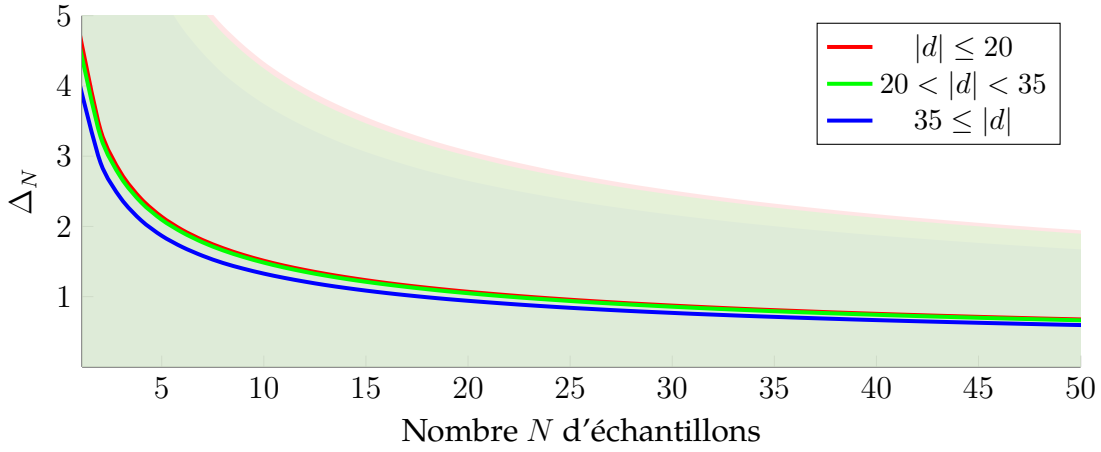

FIGURE B.1 – Impact de la taille du document en entrée sur l'erreur d'échantillonnage (2.3) (plus bas est meilleur). La zone pâle représente un intervalle de confiance de 95 % sur la valeur de chacune des courbes.

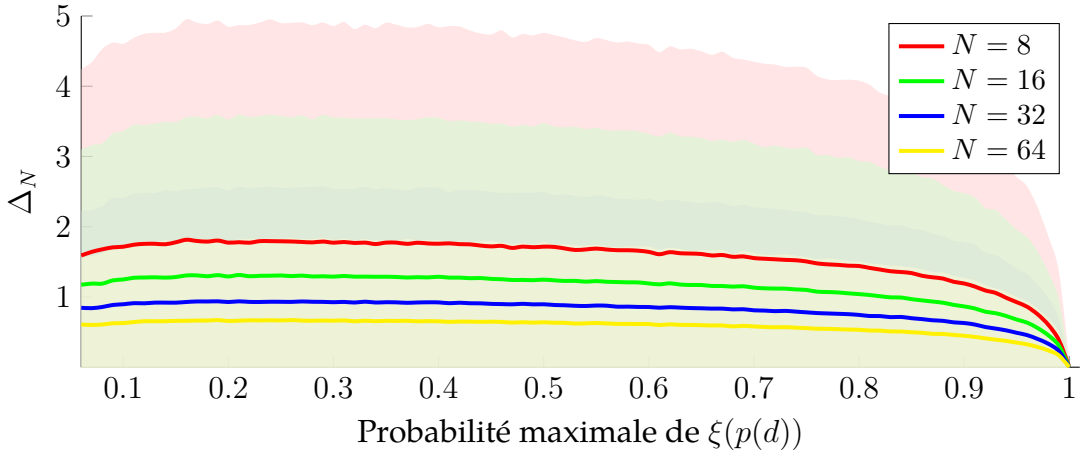

FIGURE B.2 – Impact de la probabilité maximale de la distribution des résumés sur l'erreur d'échantillonnage (2.3) (plus bas est meilleur). La zone pâle représente un intervalle de confiance de 95 % sur la valeur de chacune des courbes.

Graphiques du chapitre 3

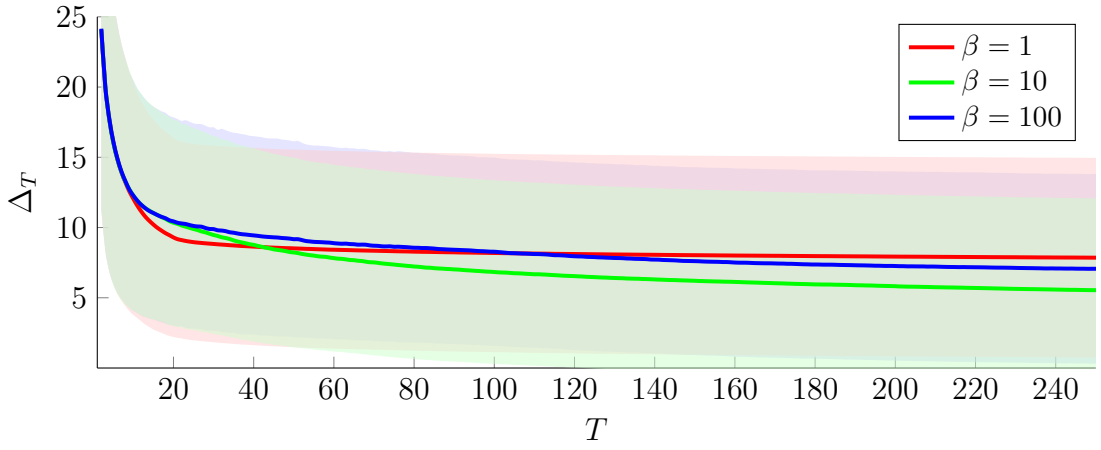

FIGURE B.3 – Impact du paramètre d'exploration β utilisé par CUCB sur la convergence. Δ_T (plus bas est meilleur) représente la différence entre le score ROUGE du meilleur résumé extractif d'un document et celui suggéré CUCB après T pas de temps. La zone pâle représente un intervalle de confiance de 95 % sur la valeur de chacune des courbes.

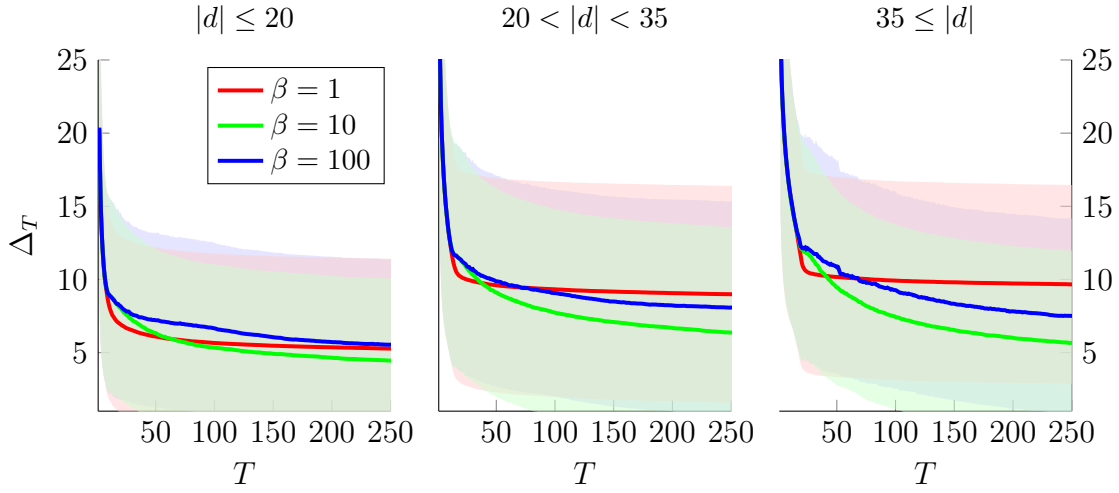

FIGURE B.4 – Impact de la taille du document sur la convergence de CUCB. Δ_T (plus bas est meilleur) représente la différence entre le score ROUGE du meilleur résumé extractif d'un document et celui suggéré CUCB après T pas de temps. La zone pâle représente un intervalle de confiance de 95 % sur la valeur de chacune des courbes.

Graphiques du chapitre 4

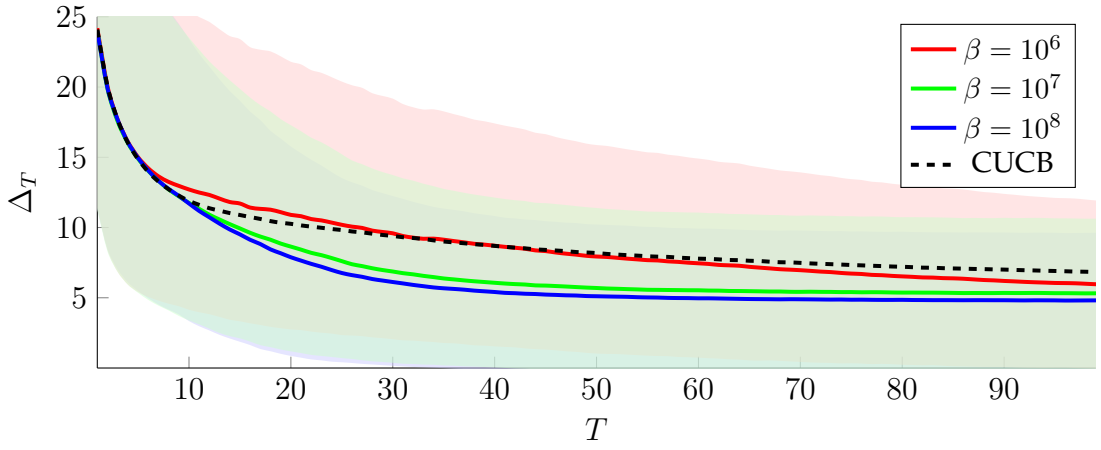

FIGURE B.5 – Impact du paramètre d'exploration β utilisé par LinCUCB sur la convergence. Δ_T (plus bas est meilleur) représente la différence entre le score ROUGE du meilleur résumé extractif d'un document et celui suggéré CUCB après T pas de temps. La zone pâle représente un intervalle de confiance de 95 % sur la valeur de chacune des courbes.

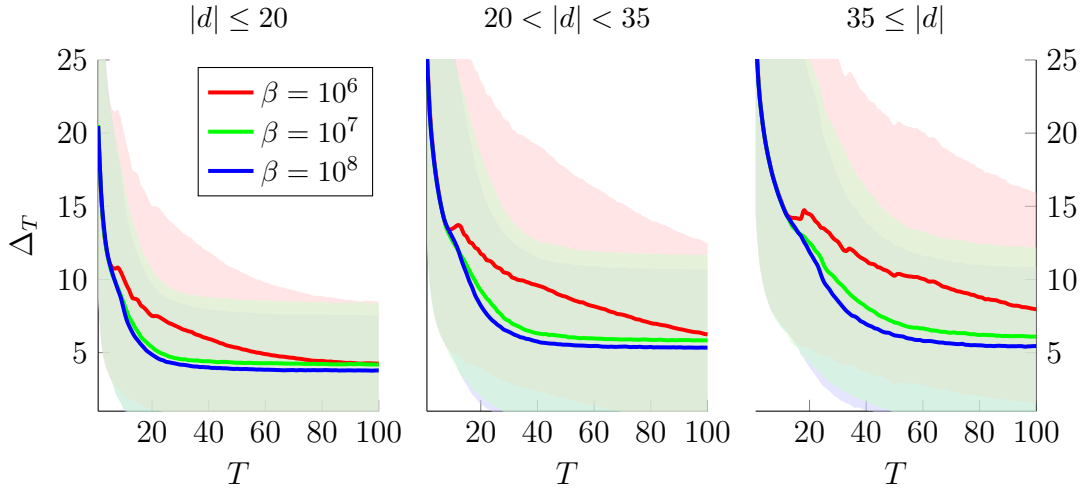

FIGURE B.6 – Impact de la taille du document sur la convergence de LinCUCB. Δ_T (plus bas est meilleur) représente la différence entre le score ROUGE du meilleur résumé extractif d'un document et celui suggéré CUCB après T pas de temps. La zone pâle représente un intervalle de confiance de 95 % sur la valeur de chacune des courbes.

Bibliographie

M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, et al. Tensorflow : A system for large-scale machine learning. In *12th {USENIX} symposium on operating systems design and implementation ({OSDI} 16)*, pages 265–283, 2016.

Y. Abbasi-Yadkori, D. Pál, and C. Szepesvári. Improved algorithms for linear stochastic bandits. *Advances in neural information processing systems*, 24 :2312–2320, 2011.

F. Almeida and G. Xexéo. Word embeddings : A survey. *arXiv preprint arXiv :1901.09069*, 2019.

P. Auer. Using confidence bounds for exploitation-exploration trade-offs. *Journal of Machine Learning Research*, 3(Nov) :397–422, 2002.

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

L. Chen, J. Xu, and Z. Lu. Contextual combinatorial multi-armed bandits with volatile arms and submodular reward. In *Advances in Neural Information Processing Systems*, pages 3247–3256, 2018.

W. Chen, Y. Wang, and Y. Yuan. Combinatorial multi-armed bandit : General framework and applications. In *International Conference on Machine Learning*, pages 151–159, 2013.

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.

A. A. Deshmukh, U. Dogan, and C. Scott. Multi-task learning for contextual bandits. In *Advances in neural information processing systems*, pages 4848–4856, 2017.

J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. Bert : Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv :1810.04805*, 2018.

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. *arXiv preprint arXiv :1905.03197*, 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.

Z.-Y. Dou, P. Liu, H. Hayashi, Z. Jiang, and G. Neubig. Gsum : A general framework for guided neural abstractive summarization. *arXiv preprint arXiv :2010.08014*, 2020.

I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio. *Deep learning*, volume 1. MIT press Cambridge, 2016.

K. M. Hermann, T. Kocisky, E. Grefenstette, L. Espeholt, W. Kay, M. Suleyman, and P. Blunsom. Teaching machines to read and comprehend. *Advances in neural information processing systems*, 28 :1693–1701, 2015.

S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8) :1735–1780, 1997.

A. Joulin, E. Grave, P. Bojanowski, and T. Mikolov. Bag of tricks for efficient text classification. *arXiv preprint arXiv :1607.01759*, 2016.

D. P. Kingma and J. Ba. Adam : A method for stochastic optimization. *arXiv preprint arXiv :1412.6980*, 2014.

T. G. Kolda and D. P. O'leary. A semidiscrete matrix decomposition for latent semantic indexing information retrieval. *ACM Transactions on Information Systems (TOIS)*, 16(4) :322–346, 1998.

W. Kryściński, N. S. Keskar, B. McCann, C. Xiong, and R. Socher. Neural text summarization : A critical evaluation. *arXiv preprint arXiv :1908.08960*, 2019.

W. Kryscinski, B. McCann, C. Xiong, and R. Socher. Evaluating the factual consistency of abstractive text summarization. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 9332–9346, 2020.

V. Kuleshov and D. Precup. Algorithms for multi-armed bandit problems. *arXiv preprint arXiv :1402.6028*, 2014.

T. L. Lai and H. Robbins. Asymptotically efficient adaptive allocation rules. *Advances in applied mathematics*, 6(1) :4–22, 1985.

T. Lattimore and C. Szepesvári. *Bandit algorithms*. Cambridge University Press, 2020.

L. Li, W. Chu, J. Langford, and R. E. Schapire. A contextual-bandit approach to personalized news article recommendation. In *Proceedings of the 19th international conference on World wide web*, pages 661–670, 2010.

C.-Y. Lin. Rouge : A package for automatic evaluation of summaries. In *Text summarization branches out*, pages 74–81, 2004.

W. Liu, Z. Wang, X. Liu, N. Zeng, Y. Liu, and F. E. Alsaadi. A survey of deep neural network architectures and their applications. *Neurocomputing*, 234 :11–26, 2017.

Y. Liu and M. Lapata. Text summarization with pretrained encoders. *arXiv preprint arXiv :1908.08345*, 2019.

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 3024–3034, 2019.

T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. *Advances in neural information processing systems*, 26 :3111–3119, 2013.

R. Nallapati, F. Zhai, and B. Zhou. Summarunner : A recurrent neural network based sequence model for extractive summarization of documents. In *Proceedings of the AAAI conference on artificial intelligence*, volume 31, 2017.

S. Narayan, S. B. Cohen, and M. Lapata. Ranking sentences for extractive summarization with reinforcement learning. *arXiv preprint arXiv :1802.08636*, 2018.

J.-P. Ng and V. Abrecht. Better summarization evaluation with word embeddings for rouge. *arXiv preprint arXiv :1508.06034*, 2015.

A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, et al. Pytorch : An imperative style, high-performance deep learning library. In *Advances in neural information processing systems*, pages 8026–8037, 2019.

R. Paulus, C. Xiong, and R. Socher. A deep reinforced model for abstractive summarization. *arXiv preprint arXiv :1705.04304*, 2017.

J. Pennington, R. Socher, and C. D. Manning. Glove : Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, 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, 2019.

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, 2020.

H. Robbins. Some aspects of the sequential design of experiments. *Bulletin of the American Mathematical Society*, 58(5) :527–535, 1952.

H. Robbins and S. Monro. A stochastic approximation method. *The annals of mathematical statistics*, pages 400–407, 1951.

R. Y. Rubinstein and D. P. Kroese. *The cross-entropy method : a unified approach to combinatorial optimization, Monte-Carlo simulation and machine learning*. Springer Science & Business Media, 2013.

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 *Twenty-fourth international joint conference on artificial intelligence*, 2015.

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.

M. Soare, O. Alsharif, A. Lazaric, and J. Pineau. Multi-task linear bandits. In *NIPS2014 Workshop on Transfer and Multi-task Learning : Theory meets Practice*, 2014.

R. S. Sutton and A. G. Barto. *Reinforcement learning : An introduction*. MIT press, 2018.

R. S. Sutton, D. A. McAllester, S. P. Singh, and Y. Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Advances in neural information processing systems*, pages 1057–1063, 2000.

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.

A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

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

Y. Wu, M. Schuster, Z. Chen, Q. V. Le, M. Norouzi, W. Macherey, M. Krikun, Y. Cao, Q. Gao, K. Macherey, et al. Google's neural machine translation system : Bridging the gap between human and machine translation. *arXiv preprint arXiv :1609.08144*, 2016.

B. Xu, N. Wang, T. Chen, and M. Li. Empirical evaluation of rectified activations in convolutional network. *arXiv preprint arXiv :1505.00853*, 2015a.

K. Xu, J. Ba, R. Kiros, K. Cho, A. Courville, R. Salakhudinov, R. Zemel, and Y. Bengio. Show, attend and tell : Neural image caption generation with visual attention. In *International conference on machine learning*, pages 2048–2057, 2015b.

J. Zhang, Y. Zhao, M. Saleh, and P. Liu. Pegasus : Pre-training with extracted gap-sentences for abstractive summarization. In *International Conference on Machine Learning*, pages 11328–11339. PMLR, 2020.

M. Zhong, P. Liu, Y. Chen, D. Wang, X. Qiu, and X. Huang. Extractive summarization as text matching. *arXiv preprint arXiv :2004.08795*, 2020.