Approches de bandits pour la génération automatique de résumés de textes

Mémoire

Mathieu Godbout

Sous la direction de:

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

Résumé

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 nous proposons de voir la production d'un résumé extractif comme différents problèmes de bandit, lesquels sont accompagnées d'algorithmes pouvant être utilisés pour l'entraînement.

Dans ce document, nous voyons d'abord comment la génération du résumé de n'importe quel document peut être formulée comme un problème de bandit contextuel pour atteindre une excellente efficacité computationnelle. Ensuite, nous proposons de voir la génération du résumé d'un seul document comme un bandit combinatoire. Nous présentons une méthode permettant d'obtenir des cibles pour l'entraînement d'un modèle à partir de la résolution du bandit combinatoire. On présente deux algorithmes, UCB et LinUCB, qui peuvent être utilisés à cet effet. UCB, le premier algorithme présenté, se caractérise par son universalité alors que LinUCB se veut computationnellement plus efficace, au prix ajouté d'une hypothèse de l'existence d'une relation linéaire entre phrase et la qualité des résumés qu'elle produit.

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 an extractive summary as different bandit problems, for which there exist algorithms that can be used for training.

In this document, we first see how the generation of the summary of any document can be formulated as a contextual bandit problem to achieve excellent computational efficiency. Next, we propose to see the generation of the abstract of a single document like a combinatorial bandit. We present a method for obtaining targets for training a model from the resolution of the combinatorial bandit. We present two algorithms, UCB and LinUCB, which can be used for this purpose. UCB, the first algorithm presented, is characterized by its universality while LinUCB is computationally more efficient, at the added cost of an assumption of the existence a linear relationship between a sentence and the quality of abstracts it produces.

Table des matières

Liste des tableaux

Liste des figures

Liste des algorithmes

À Samantha.

"<Texte de l'épigraphe>"

<Source ou auteur>

Remerciements

<Texte des remerciements en prose.>

Introduction

Lors d'une entrevue en 2018, le président de la multinationale Siemens mentionnait que (traduction libre) "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. Ces systèmes sont le sujet d'étude principal du présent document.

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

tion de résumés qui apprendront rapidement à générer des résumés satisfaisants grâce à une exploration efficace du vaste espace de résumés possibles.

Objectifs

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

— Comment peut-on formuler la génération de résumés comme un problème de bandit stochastique ? Comment les algorithmes utilisés habituellement pour résoudre ces problèmes peuvent alors être appliqués à l'entraînement de modèles de génération de résumés ?

— Dans quelle mesure ces algorithmes de bandit sont-ils performants dans le contexte de la génération de résumés ? Sont-ils plus rapides en nombre d'échantillons requis pour une bonne performance ? Est-ce que les solutions auxquelles ils convergent sont meilleures ?

— Comment l'introduction de ces artefacts bandits a-t-elle affecté la performance ? Est-ce que les systèmes produits génèrent des résumés de meilleure qualité ? A-t-on besoin de moins d'exemples avant d'avoir un système de haute qualité ? Est-ce que l'entraînement est plus rapide à effectuer ? Avec quelles ressources de calcul ?

— Comment les performances de ces algorithmes se comparent-elles entre eux ? Et par rapport à l'état de l'art ?

On atteint les objectifs en proposant 3 formulations de bandits applicables au processus de la génération de résumés. Une des approches proposée est présente dans la littérature mais les deux autres sont des nouveautés, du mieux de notre connaissance.

Structure du mémoire

Ce mémoire débute par une introduction aux concepts de base essentiels à la compréhension du reste du document. Le document est ensuite divisé en trois chapitres pour les trois formulations de MAB explorées : bandit contextuel, bandit combinatoire et bandit combinatoire linéaire.

Pour chacun des chapitres, on commence par définir le cadre bandit proposé. Des expériences empiriques sont effectuées pour justifier l'applicabilité du cadre au contexte de génération de résumés. On montre ensuite comment le cadre bandit peut utilisé dans un nouvel algorithme pour l'entraînement d'un système neuronal de génération automatique de résumés. Enfin, on valide le comportement de l'algorithme en l'utilisant pour entraîner plusieurs modèles de génération de résumés. Les algorithmes sont comparés en fonction de leur rapidité de convergence et de la qualité finale des modèles produits.

Chapitre 1

Concepts de base pertinents aux travaux

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

On définit un bandit (Lattimore and Szepesvári, 2020) comme étant un problème où un apprenant interagit avec un environnement de manière séquentielle. À chaque pas de temps, l'apprenant sélectionne une action à effectuer et reçoit une récompense de l'environnement associée à l'action. L'objectif de l'apprenant est alors de maximiser les récompenses obtenues sur un horizon fini d'interactions avec l'environnement. Notons que cet objectif est différent des approches d'apprentissage automatique supervisées : au lieu de s'intéresser seulement à la performance finale de l'apprenant, on s'intéresse aussi à ce que sa performance s'améliore rapidement. En pratique, l'objectif bandit est le plus approprié pour des problèmes où il est crucial de ne pas commettre trop d'erreurs durant l'apprentissage. Notamment, diverses déclinaisons des bandits ont été appliquées avec succès dans les domaines des essais cliniques (Kuleshov and Precup, 2014), de la gestion de portfolios financiers (Shen et al., 2015) et de la suggestion personnalisée d'articles de journaux (Li et al., 2010).

Nous présentons ici le bandit multi-bras stochastique et sa déclinaison linéaire, qui représentent les versions du problème bandit que nous appliquons tout au long du document. Pour chacune des versions, un algorithme permettant de le résoudre est aussi présenté.

1.1.1 Bandit multi-bras stochastique

Un bandit multi-bras stochastique est un problème pour lequel on a un ensemble $\mathcal{A}$ de K actions disponibles. À chaque action a est associée une récompense $X_{a,t}$ pour le t-ème pas de temps effectué. Dans le contexte stochastique, il est supposé que les récompenses $X_{a,t}$ d'une action sont indépendantes et identiquement distribuées (i.i.d.) et qu'il existe une espérance fixe μ_a pour chaque action a, i.e. $\mu_a = \mathbb{E}[X_{a,t}]$.

Dans ce contexte, l'objectif de l'apprenant de maximiser les récompenses perçues devient celui 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. Un algorithme visant à résoudre le problème de bandit sélectionnera successivement des actions a_t dans le but de maximiser l'espérance de ses récompenses $X_{a_t,t}$. Afin de mesurer la performance d'un algorithme sur un bandit stochastique, la métrique de performance habituellement 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 donc 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 la sous-optimalité de l'apprenant. En minimisant le regret cumulatif, un apprenant maximise donc, en espérance, les récompenses perçues et résout le problème de bandit.

L'objectif de minimisation du pseudo-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.

1.1.2 Upper Confidence Bound (UCB)

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

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

où $\bar{x}_a(t)$ est la moyenne des récompenses reçues jusqu'au temps t par l'action a, $n_a(t)$ est le nombre de fois que l'action a a été sélectionnée et β est un hyperparamètre régulant l'exploration. Quand une action n'a pas encore été sélectionnée i.e. $n_a(t) = 0$, on lui assigne $\text{UCB}_a = \infty$, 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 borne supérieure ainsi définie garantit avec une forte probabilité que la valeur μ_a espérée pour l'action a soit inférieure à UCB_a au temps t.

L'équation (1.2) peut être visualisée comme résolvant directement le dilemme exploitation-exploration. En effet, le terme $\bar{x}_a(t)$ présente l'estimation actuelle de la récompense associée à l'action a et permet donc d'exploiter les connaissances acquises avant le temps t. À gauche, le terme $\sqrt{\frac{2 \ln t}{n_a(t)}}$ est proportionnel à l'inverse des visites relatives. Il s'agit donc d'un terme qui sera plus élevé pour les actions moins visitées, incitant ainsi l'exploration d'actions sous-visitées.

1.1.3 Bandit stochastique linéaire

La formulation en bandit linéaire reprend exactement le même cadre formel que la bandit stochastique. La seule nouvelle notion est 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 :

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

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 sur le problème de bandit stochastique linéaire qui vise la minimisation du pseudo-regret cumulatif (1.3).

Rappelons que ce principe 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, comme on souhaite faire usage des représentations vectorielles des actions, on cherche à avoir UCB_a qui fait usage des relations linéaires entre les actions. Nous présentons plus bas l'intuition derrière la construction de bonnes 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 une bonne borne supérieure est celle d'exploiter les observations faites aux pas de temps précédentes pour bâtir un estimé $\hat{\omega}_N$ de ω_*. Cet estimé devrait naturellement viser à combiner aussi bien que possible les paires actions-récompenses $(\tilde{a}_t, X_T)$ observées aux pas de temps précédentes, i.e.

$$\langle \hat{\omega}_N, \tilde{a}_t \rangle \approx X_t. \tag{1.4}$$

L'équation (1.4) peut être vue comme un problème de moindres carrés, où l'on souhaite trouver

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

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 $\hat{\omega}_N$ est alors choisi selon la solution analytique connue du problème

$$\hat{\omega}_N = \mathbf{V_N}^{-1} \mathbf{b_N}, \quad \text{pour} \quad \mathbf{V_N} = (\mathbf{A_N}^\top \mathbf{A_N} + \lambda \mathbf{I}) \quad \text{et} \quad \mathbf{b_N} = \mathbf{A_N}^\top \mathbf{X_N}, \tag{1.6}$$

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

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

$$\mathbf{V_N} = \lambda \mathbf{I} + \sum_{t=1}^{N} \tilde{a}_t \tilde{a}_t^\top, \qquad \mathbf{b_N} = \sum_{t=1}^{N} \tilde{a}_t X_t. \tag{1.7}$$

Maintenant que l'on possède une bonne estimation $\hat{\omega}_N$ qui exploite l'expérience des pas de temps précédentes, on cherche à introduire un terme garantissant une exploration satisfaisante des actions possibles. LinUCB propose à cet effet d'utiliser

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

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

En regroupant les termes d'exploitation et d'exploration, on obtient encore une fois une borne

supérieure résolvant de manière élégante le compromis exploration-exploitation :

$$\text{UCB}_a = \langle \omega_N^\top, \tilde{a} \rangle + \beta \sqrt{\tilde{a}^\top {\mathbf{V_N}}^{-1} \tilde{a}}, \tag{1.8}$$

où β représente encore un hyperparamètre balançant l'exploration.

Connexions avec UCB

Nous prenons un moment ici pour mettre l'emphase sur les connexions entre UCB et LinUCB, qui peut être vu comme une généralisation linéaire directe de UCB, tel que présenté dans Lattimore and Szepesvári (2020). En effet, si l'on considère les vecteurs d'action correspondant aux vecteurs unitaires, i.e. $\tilde{a} = e_a$ et une régularisation de Tikhonov avec $\lambda = 0$, on a que $\mathbf{V_N}$ est une matrice diagonale, où la a-ème entrée est le nombre de fois où l'action a a été sélectionnée. À ce moment, le terme $\tilde{a}^\top {\mathbf{V_N}}^{-1} \tilde{a}$ n'est donc que l'inverse du nombre n_a de fois que l'action a a été sélectionnée et le terme d'exploration devient

$$\beta \sqrt{\frac{1}{n_a(N)}},$$

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

Aussi, $\hat{\omega}_N$ devient simplement le vecteur où l'entrée a correspond à la moyenne des récompenses perçues pour l'action i. Le produit vectoriel $\langle \hat{\omega}_N, e_a \rangle$ ne fait donc qu'aller chercher la moyenne $\bar{x}_a$ 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. On peut donc considérer que, si l'hypothèse linéaire est respectée, LinUCB obtiendra toujours une performance supérieure ou égale à UCB.

1.2 Réseaux de neurones

L'apprentissage supervisé est un problème dans lequel on tente d'apprendre la relation se cachant 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 très 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 munie de paramètres θ. On tente alors de trouver les paramètres θ^* produisant l'approximateur paramétré f_{θ^*} 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 et leur utilisation a récemment explosé suite à leur excellente performance empirique et leur application à vraisemblablement n'importe quel problème supervisé (voir Liu et al. (2017) pour un récapitulatif détaillé).

Les RN ne sont utilisés dans ce document que pour permettre l'application de nos contributions. Nous nous limitons donc à présenter le processus de descente de gradient que les RN utilisent ainsi que les cas d'utilisation pertinents à notre étude.

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 y et $\hat{y}$, l est seulement soumise à la contrainte d'être dérivable, i.e. le gradient $\nabla_y l(\hat{y}, y)$ existe et est bien défini. Par exemple, pour des problèmes de classification binaire où les cibles y sont des nombres entre 0 et 1, une fonction de perte utilisable est l'entropie croisée binaire

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

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

$$\theta^* = \arg\min_\theta \mathcal{L}(\theta), \quad \text{où} \quad \mathcal{L}(\theta) = \mathop{\mathbb{E}}_{(x,y)\sim\mathcal{D}} l\left(f_\theta(x), y\right). \tag{1.9}$$

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

$$\bar{\mathcal{L}}(\theta) = \frac{1}{B} \sum_{i=1}^{B} l\left(f_\theta(x_i), y_i\right), \tag{1.10}$$

où $\bar{\mathcal{L}}(\theta)$ est 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 computationnelles à disposition.

Le problème d'optimisation (1.9) peut être résolu en utilisant la perte empirique définie en

(1.10) en appliquant une descente de gradient. En effet, comme nous le verrons bientôt, un réseau de neurones est entièrement dérivable par rapport à ses paramètres et on peut donc optimiser les paramètres θ selon la règle de mise à jour

$$\theta = \theta - \alpha \nabla \bar{\mathcal{L}}(\theta), \tag{1.11}$$

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 (1.11) 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 x comme entrée, et de la dernière couche c_n qui produit une valeur prédite $\hat{y}$. Enfin, notons que l'on définit les paramètres θ d'un RN comme l'ensemble des matrices $\mathbf{W}$ 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., 2015) : $\sigma(z) = \max(0, z)$. Fonction dont la sortie est toujours positive et non bornée.

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

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

Le choix de la fonction d'activation utilisée à chaque couche dépend du comportement souhaité. Notamment, 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 des séquences. 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 $\mathbf{W}$ et 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 de la séquence x en entrée.

En pratique, les réseaux récurrents peuvent être rendus 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 RNN, en utilisant les sorties o 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 RNN empilés. Une illustration contenant un réseau récurrent bidirectionnel à deux couches est disponible à la figure 2.1.

Plus récemment, la plupart des bons résultats empiriques obtenus avec des réseaux récurrents ont été obtenus en utilisant une variante plus complexe que celle présentée : le LSTM (Gers, 1999). Les LSTM 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., 2016).

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 (plongements) de mots.

Les plongements ainsi obtenus incorporent des relations présentes entre les mots qu'ils représentent. Par exemple, si on définit $p(m)$ comme le plongement associé au mot m, Mikolov

et al. (2013) démontrent que la relation

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

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 RN 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 que l'on souhaite aussi compressée que possible mais qui doit aussi conserver la majeure partie de l'information contenue.

Pour notre part, nous nous intéressons seulement au cas de la génération de résumés à partir d'un seul document d pour lequel nous possédons un seul résumé cible s. Des approches existent spécifiquement pour la génération de résumés à partir d'un ensemble de documents en entrée, mais nous nous contentons de dire que le cas multi-documents se ramène grossièrement au cas de la génération d'un seul document en considérant la concaténation de l'ensemble de documents.

Sous l'optique qui nous intéresse, la litérature peut actuellement être séparée en deux formulations distinctes : extractive et abstractive. Nous présentons chacune de ces méthodes, apportant une attention particulière à la formulation extractive qui fait l'objet d'une investigation détaillée dans ce document. Nous en profitons 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) éviter toutes les difficultés en termes de syntaxe et de cohérence liées à avoir à générer des phrases 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 qui contient $|d|$ phrases $[d_1, ..., d_{|d|}]$ et pour lequel on a un 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 $\pi(d) \in [0,1]^{|d|}$, un vecteur où chaque index représente à quel point la phrase associée 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 process général de génération de résumé extractif décrit 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) par rapport à 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 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 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) représente actuellement l'état de l'art extractif sur le jeu de données CNN/DailyMail (Hermann et al., 2015) en utilisant seulement les cibles binaires. 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.

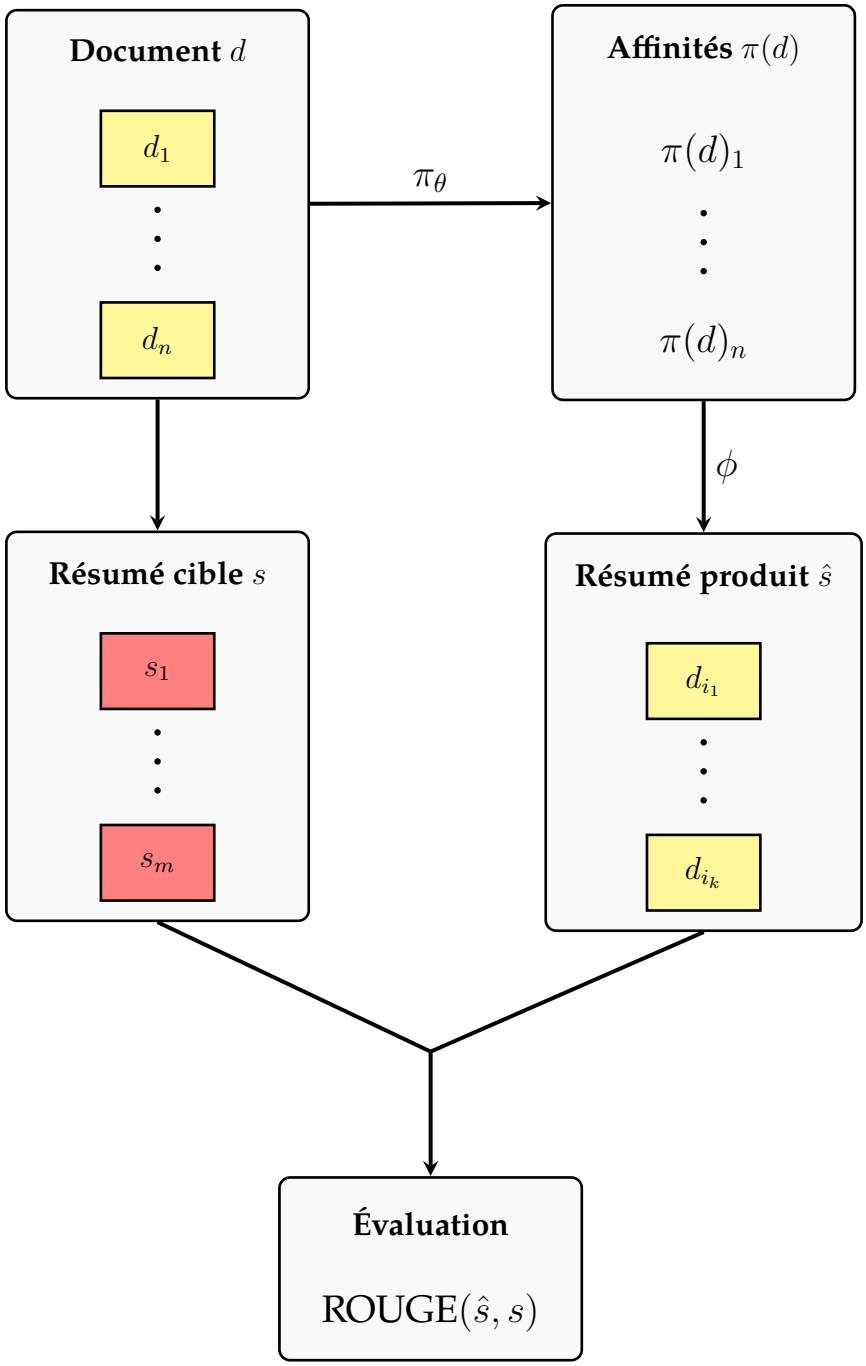

FIGURE 1.1 – Schéma des étapes de la génération de résumé extractif pour un document d et son résumé cible s. La couleur différente utilisée pour les phrases du résumé cible sert à indiquer qu'elles ne sont habituellement pas des phrases du document en entrée.

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 dis-

pose 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

$$J(\theta) = \mathop{\mathbb{E}}_{(d,s)\sim\mathcal{D}} \mathop{\mathbb{E}}_{s\sim\phi(\pi_\theta(d))} \left[G(\hat{s}, s)\right], \tag{1.12}$$

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

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

$$\nabla J(\theta) = G(\hat{s}, s)\nabla \ln \phi\left(\pi_\theta(\hat{s})\right), \tag{1.13}$$

basée sur un échantillon de J. Le théorème du gradient de politique (Sutton et al., 1999) affirme que, en espérance, l'approximation 1.13 correspond au véritable gradient 1.12 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 computationnel grandement inférieur aux approches de l'état de l'art.

1.3.2 Formulation abstractive

Sous cette formulation, on génère le résumé d'un texte en écrivant le résumé mot à mot. Les résumés abstractifs peuvent donc potentiellement reproduire de manière parfaite le résumé cible associé à un 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, Kryscinski et al. (2020) ont même découvert que les modèles abstractifs peuvent être sujets à la génération d'énoncés factuellement incorrects.

Comme nous nous intéressons aux méthodes extractives pour le reste du document, nous nous contentons 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., 2019) 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 cible s et un résumé généré $\hat{s}$. Il en existe en fait plusieurs, notamment :

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

— ROUGE-WE (Ng and Abrecht, 2015) : métrique similaire à ROUGE-n, mais qui utilise le *soft-matching* basé sur la similarité cosine entre les plongements de mots au lieu 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. Notons aussi que toutes les métriques énoncées plus haut ne sont pas dérivables et ne peuvent donc pas être utilisées directement comme fonction de perte d'un réseau de neurones.

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.

Peyrard (2019) démontre que, bien que ces métriques soient généralement corrélées avec l'avis d'un expert humain, elles peuvent difficilement être considérées comme un remplacement de celui-ci. En effet, comme les métriques sont toutes similairement corrélées avec le jugement humain mais qu'elles ne sont que faiblement corrélées entre elles, aucune des métriques ne peut être utilisée à elle seule comme un remplacement de l'avis humain. En utilisant une moyenne de plusieurs métriques, il est possible d'alléger quelque peu ce problème. Notons toutefois que même l'utilisation d'une moyenne ne suffit pas à régler le problème : Paulus et al. (2017) entraînent plusieurs modèles abstractifs mais finissent par déployer un modèle autre que celui atteignant la meilleure moyenne de score ROUGE-1, ROUGE-2 et ROUGE-L.

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 de notre 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 retenue

L'objectif fondamental de ce document est de proposer et d'analyser différents algorithmes d'entraînement 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.

Nous commençons donc en présentant et motivant la problématique de génération de résumé à l'étude, avec toutes les contraintes et hypothèses retenues. On décrit ensuite un premier algorithme qui agira à titre de référence pour les prochains chapitres : BanditSum (Dong et al., 2018) et son approche basée sur un bandit contextuel. Des expériences empiriques sont menées pour certifier l'applicabilité de la formulation contextuelle à la génération de résumés. Enfin, on présente comment BanditSum incorpore l'algorithme REINFORCE (Williams, 1992) pour obtenir une performance représentant l'état de l'art en termes d'efficacité computationnelle.

2.1 Description de la problématique

L'enjeu principal auquel on s'intéresse est celui de l'entraînement de modèles de générations de résumés. Plus particulièrement, on désire trouver, pour un modèle π_θ doté de paramètres θ, quels sont les paramètres qui permettent de maximiser la qualité des résumés produits. Pour nos travaux, il est choisi de s'intéresser exclusivement à la génération automatique de résumés extractifs (1.3.1) à partir d'un seul document en entrée. La formulation extractive est retenue en raison de la facilité avec laquelle les approches par bandit peuvent y être utilisées. À partir de maintenant, comme nous considérons exclusivement les résumés extractifs, nous utiliserons interchangeablement résumé et résumé extractif.

Or, comme le résumé cible s associé à un document d est rarement constitué de phrases de d, il n'est pas trivial de définir une procédure d'apprentissage dans le contexte extractif. Ce

sont précisément ces procédures d'apprentissage, leurs intuitions et leur comparaison qui font l'intérêt de notre étude.

On se dote d'un cadre uniforme 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 Dong et al. (2018), que nous choisissons comme base de référence.

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.

Nous emploierons aussi plusieurs des contraintes retenues par les approches de l'état-de-l'art sur le jeu de données du CNN/DailyMail. Notamment, à la manière de BanditSum, nous considérerons seulement les résumés de 3 phrases en sortie. Cette contrainte, utilisée fréquemment sur le jeu de données du CNN/DailyMail, est basée sur le fait que la moyenne du nombre de phrases contenues dans les résumés cibles est près de 3. Aussi, une référence souvent utilisée pour ce jeu de données est l'heuristique Lead-3 (Nallapati et al., 2017) qui consiste à générer le résumé d'un document en retenant les 3 premières phrases. Les résultats obtenus par Lead-3 sont très bons aux yeux des métriques automatiques, justifiant encore une fois qu'avec 3 phrases on peut générer de bons résumés des documents du jeu de données. En vue des statistiques du jeu de données, nous considérerons aussi la régularisation 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.

Jeu de développement

Afin de mener des expériences empiriques plus poussées sur l'applicabilité des différentes approches par bandit proposées, nous créons un jeu de données 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.

2.1.2 Métrique d'évaluation

Pour l'évaluation de la performance, 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, les métriques ROUGE (Lin, 2004) 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) est motivé par deux facteurs principaux. D'abord, ROUGE jouit d'une diversité de signal naturelle comme il s'agit de la moyenne de trois métriques automatiques. C'est cette diversité de signal qui permet d'avoir une évaluation automatique se rapprochant le plus possible de l'évaluation humaine.

Aussi, 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. Nous exploitons abondamment cette propriété car elle nous 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 extractif, 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 d, on sait que tous les résumés que l'on doit considérer sont les combinaisons de 3 phrases, que l'on ordonne selon leur position initiale dans d. Il est donc possible de calculer et entreposer le score $\text{ROUGE}(\hat{s}, s)$ 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 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 composée de deux couches reliées par une activation ReLU et avec sortie sigmoïde. À l'inférence, un résumé est produit en sélectionnant les 3 phrases avec la plus grande affinité. Nous reprendrons exactement la même architecture de réseau de neurones pour toutes les expérimentations de ce document.

Détails expérimentaux

Nous reprenons des détails d'implémentation presque identiques à BanditSum. La dimension cachée des LSTM 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 seront 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 seront menées sur une carte graphique Tesla T4, dotée d'une mémoire virtuelle de 16 GB.

En fixant de cette manière l'architecture neuronale ainsi que les hyperparamètres qui y sont liés, nous pourrons attribuer toute différence de performance entre deux modèles aux algorithmes utilisés pour les entraîner.

2.2 Formulation contextuelle

Nous commençons ici par décrire la formulation en bandit contextuel utilisée par notre base de référence BanditSum.

Un bandit contextuel est un cas particulier du problème de bandit multi-bras stochastique présenté en 1.1. Dans cette formulation, à chaque pas de temps t l'apprenant perçoit un contexte $c_t \in \mathcal{C}$, pour $\mathcal{C}$ un ensemble de contextes. Ce contexte c_t permet à l'agent de guider le choix de son action a_t, à laquelle une récompense $X_{a_t,t} = r(c_t, a_t)$ dépendante du contexte 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 agent est dépendant d'informations reçues en entrée.

En génération de résumés, cela correspond à l'hypothèse facilement vérifiable que ce n'est pas nécessairement une bonne stratégie de toujours sélectionner les mêmes index de phrases d'un

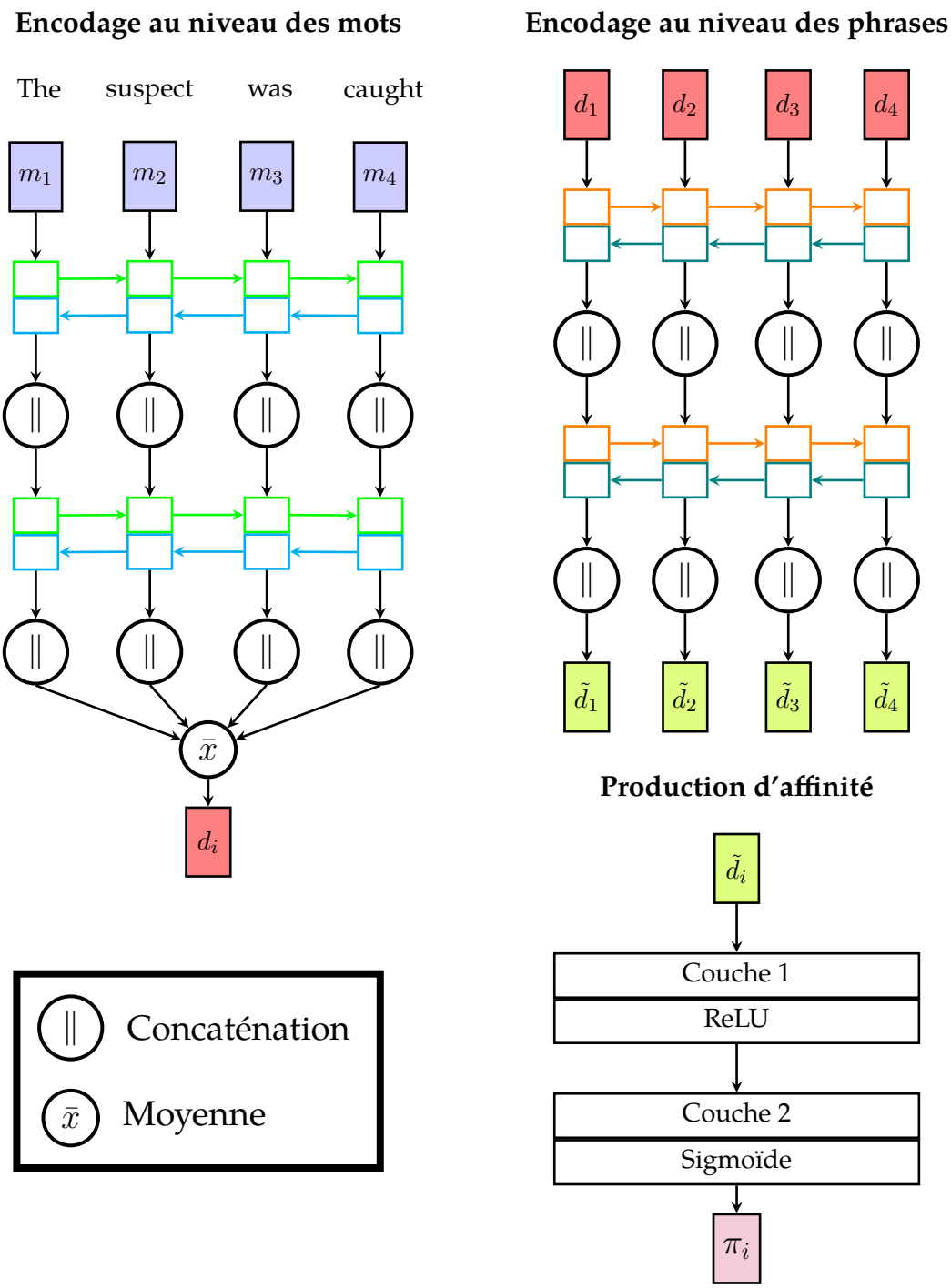

FIGURE 2.1 – Schéma décrivant l'architecture neuronale employée par BanditSum et reprise pour les expérimentations tout au long du document.

document pour bâtir son résumé. Concrètement, pour une paire document-résumé (d, s), on définit le contexte comme étant le document $(c = d)$. En posant $\mathcal{S}_d$ comme étant l'ensemble des groupes de 3 phrases de d, on a que les actions qui s'offrent à l'apprenant sont simplement

$a \in \mathcal{S}_d$. Ainsi, la récompense perçue représente naturellement la valeur de ROUGE associée ROUGE$(\hat{s}, s)$, où $\hat{s}$ est le résumé extractif issue du groupe de 3 phrases représenté par a. Rappelons ici que l'on ne s'intéresse pas à l'ordre des 3 phrases d'une action car, tel que mentionné à la section 2.1.2, le score utilisé est presque totalement invariant à l'ordre et on ordonne les phrases dans $\hat{s}$ selon leur ordre d'apparition dans d.

L'introduction de la notion de contexte dans le problème de bandit est cruciale 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 choisis afin de maximiser la performance. Dans cette optique, la prochaine section présente une technique permettant une maximisation efficace de la performance avec une grande efficacité computationnelle.

2.3 Approximation de la performance par échantillonnage

Selon le formalisme de génération de résumés défini à la section 1.3.1, on dit que l'on a un apprenant π qui, pour un document d en entrée, retourne des affinités $\pi(d) \in [0,1]^{|d|}$. 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 $J(\theta)$, la récompense espérée en générant un résumé à partir π_θ.

Dans ce contexte, il est alors naturel de vouloir utiliser un processus stochastique pour la génération de résumé, 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 π_θ. Toutefois, si l'on génère les résumés en pigeant sans remise dans la distribution engendrée 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 ψ de génération de résumé stochastique qui pige sans répétition 3 phrases selon $\pi(d)$.

En pratique, les approximations de $\nabla J(\theta)$ générées en utilisant un seul échantillon de J sont très instables pour un processus stochastique ξ, rendant l'ascension de gradient (1.13) très instable du même coup. Une solution simple et peu coûteuse pour stabiliser l'entraînement est alors de piger N résumés $\hat{s_n}$ en fonction de $\xi(\pi)$ pour bâtir un meilleur estimateur selon

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

où $\xi\left(\pi_\theta(\hat{s}_n, d)\right)$ est la probabilité de produire le résumé $\hat{s}_n$ a partir de la distribution $p(d)$

induite par $\pi_\theta(d)$.

2.3.1 Expériences

On s'intéresse à savoir combien d'échantillons sont requis pour obtenir une représentation adéquate de la valeur de $\nabla J(\theta)$. On note d'abord que, dans l'équation (2.2), la dérivée de $\xi(\pi)$ est déterministe et n'est pas sensible à la portion stochastique du processus de génération de résumés. Pour mesurer la qualité de l'approximation, il faut donc seulement s'intéresser à la différence entre la véritable valeur de $J(\theta)$ et son estimation basée sur des échantillons selon $\xi(\pi)$. Enfin, comme la pige d'un document d dans le calcul de J est uniforme, on peut simplement s'intéresser à l'estimation de J sur un seul document à la fois.

On prend donc J comme étant l'espérance de récompense de $\xi(\pi)$ sur une paire document-résumé (d, s) quelconque et on pose $\bar{J}_N(\theta) = \frac{1}{N}\sum_{i=1}^{N} \text{ROUGE}(\hat{s}_i, s)$, son approximation par échantillonnage. La qualité de l'échantillonnage dépend de la proximité entre J et $\bar{J_N}$, que nous nommons l'erreur d'échantillonnage Δ_N pour

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

Le pseudocode permettant de générer l'approximation $\bar{J}_N$ se trouve dans l'algorithme 1.

Algorithme 1 Calcul de $\bar{J}_N$

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

1: **pour** $i = 1, ..., N$ **faire**
2: Piger $\hat{s}_i \sim \xi\left(p(d)\right)$
3: $\bar{J}_N = \frac{1}{N}\sum_{i=1}^{N} \text{ROUGE}(\hat{s}_i, s)$

Méthodologie

On valide la rapidité de la convergence de $\bar{J}$ vers J en les comparant directement sur notre ensemble de développement décrit en 2.1.1. Comme le calcul de $\bar{J}$ 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 $G(d)$:

$$p(d) = \tau U(d) + (1 - \tau)G(d). \tag{2.4}$$

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 distri-

butions vorace et uniforme selon un paramètre τ, les distributions générées $p(d)$ peuvent être arbitrairement plus faciles ou difficiles à estimer. En effet, plus τ est élevé, plus $p(d)$ serait près d'une distribution vorace, pour laquelle la pige stochastique est toujours identique et l'approximation ne requiert donc qu'un seul échantillon.

Les expériences consistent à calculer la valeur de Δ_N pour N allant jusqu'à 100 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 $\hat{J}_N$ sont obtenues en suivant l'algorithme 1. On calcule analytiquement la véritable valeur de J grâce au fait que l'on possède les scores ROUGE de tous les résumés extractifs de chacun des documents du jeu de développement.

2.3.2 Résultats

Les résultats obtenus sont rapportés dans les figures 2.2 et 2.3. 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.

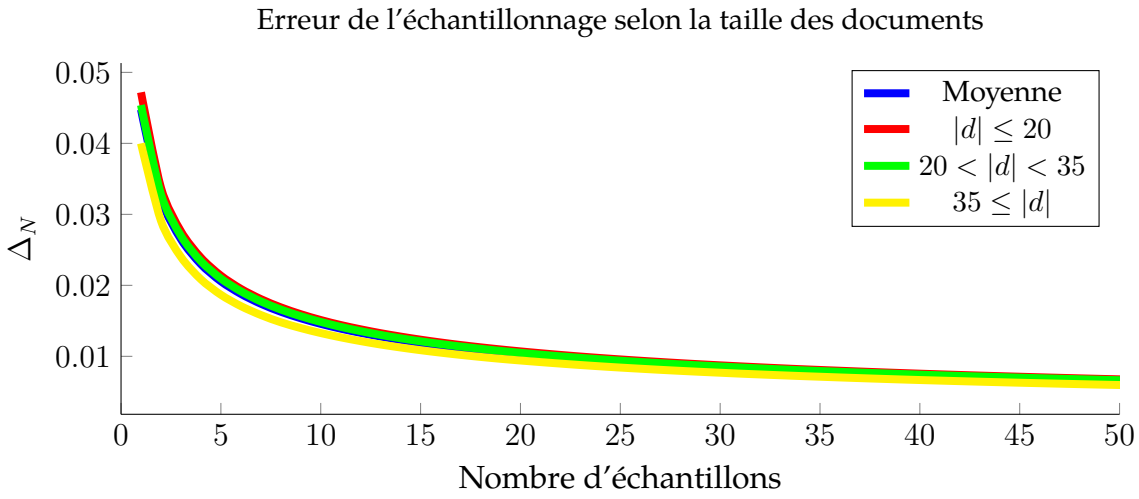

FIGURE 2.2 – Impact de la taille du document en entrée sur l'erreur d'échantillonnage définie par (2.3).

Une mesure naturelle sur la distribution des résumés est la valeur maximale de la distribution des résumés. La figure 2.3 nous montre une courbe à la tendance logarithmique, où plus la probabilité maximale de la distribution augmente, plus rapide est la convergence de notre processus d'échantillonnage.

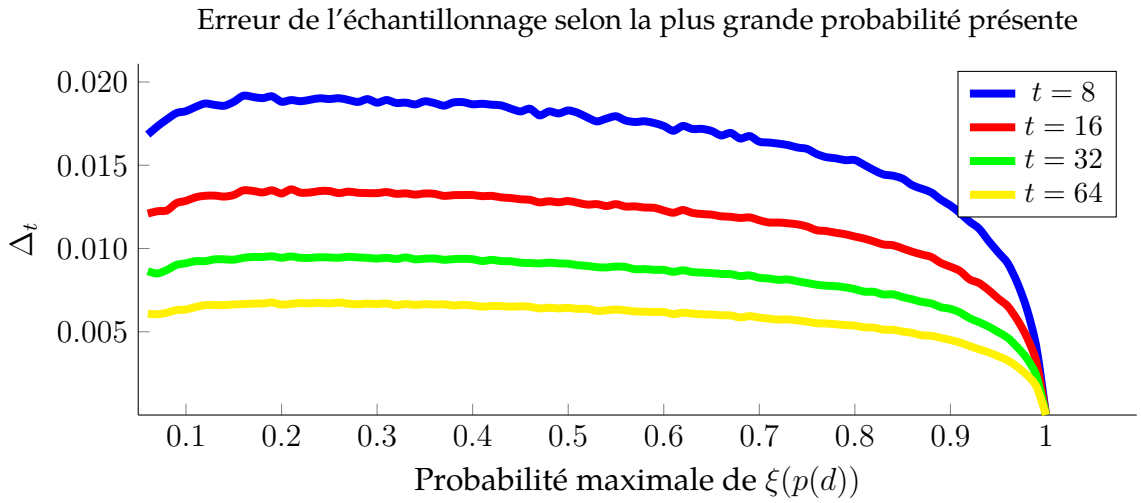

FIGURE 2.3 – Impact de la probabilité maximale de la distribution des résumés sur l'erreur d'échantillonnage définie par (2.3).

Globalement, on remarque aussi que la convergence est rapide. Une différence de moins de 0.01 ROUGE est définitivement suffisante pour justifier l'utilisation de $\bar{J}_N$ au lieu de J. Aussi, les résultats indiquent que la rapidité de la convergence ralentit progressivement autour de $N = 16$. Comme il est plus coûteux en temps de calcul de faire plus d'échantillons, il apparaît naturel de considérer $N = 16$ comme un excellent compromis entre le temps de calcul et la rapidité de l'estimation. Notons que les articles de Dong et al. (2018) et Luo et al. (2019), qui utilisent cet échantillonnage dans leur implémentation de REINFORCE, prennent $N = 20$ dans leurs expériences.

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. La prochaine section présente notre base de référence, BanditSum, dont l'apprentissage est fait selon l'ascension de gradient présentée.

2.4 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 sont quant à eux appris via une ascension de gradient basée sur 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* $b(d)$:

$$\nabla J(\theta) = \frac{1}{N} \sum_{n=1}^{N} \big[\text{ROUGE}(\hat{s}_n, s) - b(d)\big] \nabla \ln \xi\big(\pi_\theta(\hat{s}_n)\big). \tag{2.5}$$

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 de *baseline* est fréquemment utilisée pour réduire la variance élevée dans l'entraînement avec REINFORCE et permettre un apprentissage plus stable.

Le second artefact employé par BanditSum est l'ajout d'une exploration artificielle ϵ dans le processus de pige de résumé ξ. Afin d'assurer une exploration satisfaisante des résumés possibles d'un document, le processus de génération stochastique ξ qu'ils utilisent pige une phrase au hasard dans une proportion $\epsilon = 0.1$ du temps. L'introduction de ϵ a pour effet d'assurer que les résumés pigés selon $\xi(\pi, \epsilon)$ seront suffisamment variés pour assurer une bonne exploration de l'espace des résumés possibles lors de l'entraînement. Enfin, notons que la probabilité de générer un résumé $\hat{s}$, représentée par $\xi\left(\pi_\theta(\hat{s})\right)$ dans l'équation 2.5, s'écrit alors sous la forme close

$$\xi\left(\pi_\theta(\hat{s}), \epsilon\right) = \prod_{i=1}^{3} \left(\frac{\epsilon}{|d| - i + 1} + \frac{(1-\epsilon)\pi(d)_{s_i}}{\sum_{j=1}^{i-1} \pi(d)_{s_j}} \right), \tag{2.6}$$

où s_i représente l'index de la i-ème phrase du résumé produit $\hat{s}$. Ainsi, le gradient de (2.6), 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.

Enfin, dans l'article original de Dong et al. (2018), il est recommandé de faire une mise à jour des poids à partir d'une *minibatch* contenant seulement 1 article. Nous avons expérimenté avec différentes tailles B de *minibatch*, où la mise à jour des paramètres θ est désormais faite selon

$$\nabla J(\theta) = \frac{1}{B} \sum_{i=1}^{B} \frac{1}{N} \sum_{n=1}^{N} \big[\text{ROUGE}(\hat{s}_{i_n}, s_i) - b(d_i)\big] \nabla \ln \xi\big(\pi_\theta(\hat{s}_{i_n}, \epsilon)\big). \tag{2.7}$$

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

2.4.1 Expériences

On roule notre propre implémentation de BanditSum, dont le pseudocode se trouve à l'algorithme 2. On effectue un entraînement sur le jeu de données CNN/DailyMail en parcourant dans son entièreté le jeu de 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.

Algorithme 2 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 vrai **faire**
  batch $\sim \mathcal{D}^B$ ▷ On pige la minibatch du jeu de données
  $\nabla = \mathbf{0}$
  pour tout $(d,s) \in$ batch **faire**
    Piger N résumés $\hat{s}_n \sim \xi(\pi_\theta(d), \epsilon)$
    Calculer les N scores associés $r_n = \text{ROUGE}(\hat{s}_n, s)$
    $b = \text{ROUGE}(\psi(\pi), s)$
    $\nabla = \nabla + \frac{1}{N} \sum_{i=1}^{N} (r_i - b) \nabla_\theta \ln \xi\left(\pi(\hat{s}_i), \epsilon\right)$ ▷ (2.7)
  $\theta = \theta + \alpha \frac{1}{B} \nabla$

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 (2.7), en se basant sur nos expériences empiriques sur l'échantillonnage de la section 2.3. Étant donné le facteur aléatoire présent dans l'entraînement, nous effectuons 5 entraînements distincts pour chaque N et présentons la moyenne des résultats obtenus.

2.4.2 Résultats

Les résultats obtenus sont présentés dans la figure 2.4 et le tableau 2.1.

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 verte semble indiquer une en-

traînement plus stable en utilisant $N = 32$, car les variations de performance sont moins nombreuses et plus douces.

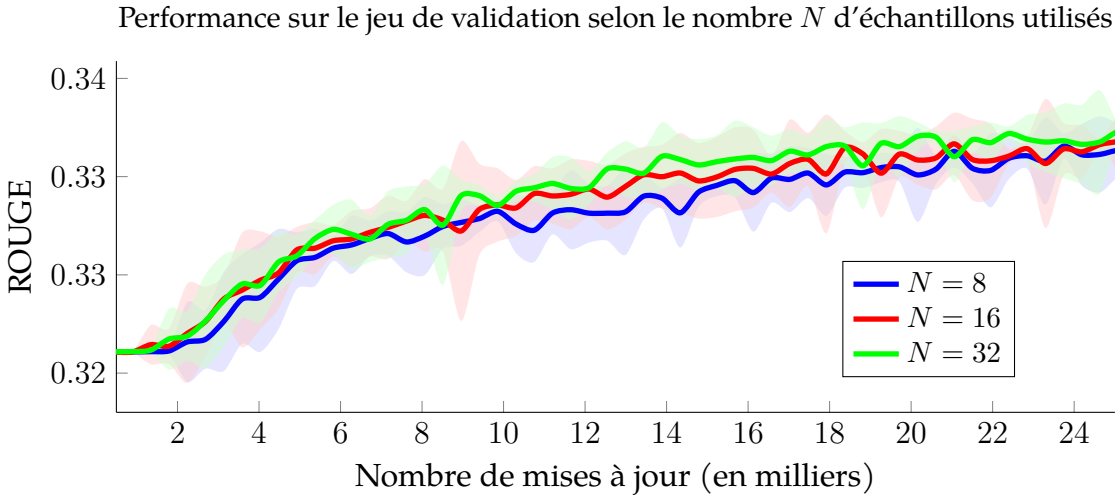

FIGURE 2.4 – Courbe d'apprentissage de BanditSum selon le nombre N d'échantillons utilisés. Un intervalle de confiance à 95 % calculé à partir de 5 entraînements distincts est rapporté pour chaque N.

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

Nous présentons d'abord les bases de référence Oracle et Lead-3 (1.3.1), représentant respectivement une estimation de la borne supérieure sur la performance de la formulation extractive et une heuristique simple mais très performante sur le jeu de données CNN/DailyMail. On rapporte ensuite la performance de trois algorithmes présentés dans la litérature : BertSumExt (Liu and Lapata, 2019), Refresh (Narayan et al., 2018) et BanditSum (Dong et al., 2018), notre base de référence. 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. Enfin, la performance en test observée avec notre propre implémentation de BanditSum est rapportée au bas du tableau. Le modèle utilisé pour l'évaluation est celui avec la meilleure performance sur le jeu de validation pour chaque entraînement.

Une première observation intéressante qui peut être tirée du tableau 2.1 est que nous parvenons à reproduire de manière convaincante les résultats rapportés par Dong et al. (2018).

Modèle	ROUGE	Taille (MB)	Mises à jour (en milliers)	Taille de minibatch
Oracle†	44.2	-	-	-
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 les bases de références, les modèles extractifs de l'état de l'art et BanditSum sur le jeu de test du CNN/DailyMail. Les † indiquent qu'il s'agit de notre implémentation de BanditSum et de notre propre recalcul des bases de références. 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.

En effet, bien que nous ne testons pas directement avec le nombre d'échantillons $N = 20$ utilisé par BanditSum, les entraînements avec 16 et 32 échantillons de notre implémentation obtiennent des performances sans différence statistique significative par rapport à l'article original. Soulignons aussi que cette reproduction des résultats est aussi 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 parcourait 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 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, nous considérons que BanditSum représente l'état de l'art en termes d'efficacité computationnelle.

2.5 Conclusion

Nous avons 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 génération de résumés extractifs. Pour permettre une comparaison rigoureuse entre les divers algorithmes à l'essai, nous nous sommes ensuite dotés d'un cadre expérimental uniformisé.

Après, nous avons présenté notre premier algorithme utilisant les bandits et qui nous servira de base de référence : BanditSum. Nous avons détaillé la formulation en bandit contextuel au coeur de BanditSum, dont nous avons établi empiriquement la validité sur le cadre expérimental uniformisé. Enfin, on a décortiqué l'algorithme BanditSum, que l'on a mis à l'épreuve sur le cadre expérimental.

Maintenant, les deux prochains chapitres décrivent deux algorithmes qui représentent des contributions entièrement nouvelles au domaine de la génération de résumés. En contraste à l'approche présentée dans ce chapitre, ces algorithmes visualisent la génération d'un seul résumé comme un problème de bandit. En résolvant le problème associé à un document, ces algorithmes génèrent des cibles utilisables pour l'entraînement et qui sont plus riches que les cibles binaires issues d'un oracle habituellement employées dans les formulations supervisées.

Chapitre 3

Bandit combinatoire

Un problème fondamental de l'apprentissage de systèmes de génération de résumés extractifs est l'indisponibilité de cibles naturelles. À cet effet, une façon de faire attrayante est de calculer un oracle (1.3.1) qui génère d'excellents résumés extractifs que l'on utilise comme cibles pour un entraînement. Habituellement, ces cibles sont binaires et représentent l'inclusion ou non d'une phrase dans le résumé produit par l'oracle. L'utilisation de cibles binaires n'est pas sans faute. Si 3 phrases peuvent constituer le résumé extractif optimal selon un oracle, il est bien possible que certaines autres phrases produisent tout de même d'excellents résumés et se voient assigner une cible nulle.

Nous proposons une approche basée sur un problème de bandit combinatoire permettant de générer des cibles plus riches que les cibles binaires. En pénalisant moins drastiquement les phrases qui ne font pas partie du résumé extractif optimal mais qui génèrent d'excellents résumés extractifs, nos cibles permettent intuitivement de quantifier la *qualité extractive* d'une phrase.

Dans ce chapitre, on présente comment la génération du résumé d'un document peut être interprétée un problème de bandit combinatoire (Cesa-Bianchi and Lugosi, 2012). On monte ensuite comment l'algorithme Upper Confidence Bound (UCB) (Auer et al., 2002) peut être utilisé pour résoudre ce type de problème et comment il peut être employé pour générer des cibles pour un système de génération de résumés. L'applicabilité des cibles proposées est par la suite validée empiriquement à l'aide d'un jeu de données de développement. Enfin, 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. On met CombiSum à l'épreuve sur notre cadre expérimental uniformisé pour juger de son efficacité.

3.1 Formulation combinatoire

Le bandit combinatoire est une approche par bandit qui vise à étendre la formulation multi-bras à des environnements où plusieurs actions $a \in \mathcal{A}$ peuvent être sélectionnées en même temps par l'apprenant. Nous nous intéressons seulement à la variante dite multitâche (Lattimore and Szepesvári, 2020), où il y a un nombre fixé m d'actions sélectionnées à chaque pas de temps t, créant une *super-action* $\mathcal{M} = \{a_1, ..., a_m\}$ pour laquelle l'environnement 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 la *super-action* optimale $\mathcal{M}^*$ et les *super-actions* choisies $\mathcal{M}_t$.

Tout l'intérêt de cette formulation vient de l'éventuelle relation entre les *super-actions*. Bien qu'il serait possible de formuler le bandit multitâche comme un bandit multi-bras 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

La génération du résumé d'un document d peut être vue comme un bandit combinatoire où les actions de base sont les phrases d_i et les *super-actions* $\mathcal{M}$ sont les résumés de 3 phrases possibles. En choisissant la *super-action* $\mathcal{M}_{\hat{s}}$ associée au résumé $\hat{s}$, l'apprenant reçoit ROUGE$(\mathcal{M}_{\hat{s}}, s)$ comme récompense et peut mettre à jour ses croyances sur les phrases d_i de $\hat{s}$. Pour alléger la notation, nous dirons désormais que la *super-action* $\mathcal{M} = \{a_{i_1}, a_{i_2}, a_{i_3}\}$ est un résumé extractif où l'action a_i correspond à la phrase d_i.

Notons d'abord que, comme on ne tient pas compte de l'ordre dans le cadre expérimental uniformisé décrit à la section 2.1, 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 complet, cette hypothèse n'est pas strictement vraie mais elle est intuitivement valide. En effet, les phrases qui résument bien le document généreront des résumés aux scores plus élevés et donc leur présence à elle seule doit contribuer à augmenter le score associé à un résumé.

Maintenant que l'on a établi le cadre théorique entourant le problème de bandit combinatoire, une question demeure : comment peut-on le résoudre ? La prochaine section répond à cette question en démontrant comment l'algorithme UCB peut être légèrement modifié pour être appliqué au bandit combinatoire.

3.1.2 UCB pour bandit combinatoire

Tel que décrit à la section 1.1.2, Upper Confidence Bound (UCB) est une approche permettant de minimiser le pseudo-regret cumulatif pour les problèmes de bandit stochastique. Pour ce faire, UCB maintient des bornes supérieures $\text{UCB}_a(t)$ pour chaque action $a \in \mathcal{A}$ et chaque pas de temps t

$$\text{UCB}_a(t) = \bar{x}_a(t) + \beta\sqrt{\frac{2\ln t}{n_a(t)}}, \tag{3.1}$$

où $\bar{x}_a(t)$ est la moyenne des récompenses reçues jusqu'au temps t par l'action a, $n_a(t)$ est le nombre de fois que l'action a a été sélectionnée et β est un hyperparamètre régulant l'exploration. Quand une action n'a pas encore été sélectionnée i.e. $n_a(t) = 0$, on lui assigne $\text{UCB}_a = \infty$, 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.

Bien que UCB est conçu pour les problèmes de bandits stochastiques, l'approche peut être naturellement étendue au problème de bandit combinatoire. En effet, il suffit alors de choisir à chaque pas de temps t la *super-action* maximisant la borne supérieure définie par UCB sur les actions $a \in \mathcal{A}$. Dans notre cas, cela revient à sélectionner la *super-action* composée des trois actions avec la borne supérieure maximale, i.e.

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

où $\text{UCB}_a(t)$ est défini selon (3.1).

La prochaine section décrit le premier volet de notre contribution : l'utilisation de UCB pour la génération de cibles extractives d'un document.

3.2 Génération de cibles par UCB

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

Sur le jeu de données CNN/DailyMail (Hermann et al., 2015), les cibles habituellement retenues sont produites par l'oracle proposé par Nallapati et al. (2017) et décrit à la section 1.3.1.

1. 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.

Mentionnons simplement que 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 ROUGE. Comme l'ordre des phrases n'impacte presque pas le score ROUGE (2.1.2), le résumé choisi par l'oracle peut être vu comme un résumé de 3 phrases presque optimal 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.

$$y_i = \mathbb{I}\big[d_i \in \dot{s}\big], \tag{3.3}$$

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

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 $\dot{s}'$ peuvent obtenir un score ROUGE très similaire. Or, les phrases de $\dot{s}'$ qui ne font pas partie de $\dot{s}$ se verront attribuer une cible de 0, alors qu'elles présentent tout de même une excellente *extractivité*.

3.2.1 Motivation

Nous proposons un nouveau processus de génération de cibles permettant d'alléger ce problème en quantifiant directement dans quelle mesure une phrase est susceptible de générer des bons résumés extractifs. Pour un document d, le processus exécute T pas de temps UCB (3.2) sur le bandit combinatoire représentant la génération du résumé de d. En minimisant le pseudo-regret, UCB identifie graduellement les bons résumés et, plus particulièrement, maintient une moyenne $\bar{x}_a$ des scores ROUGE reçus en sélectionnant la phrase a.

Étant donné un nombre de pas temps T infini, UCB identifierait éventuellement le résumé $\mathcal{M}^*$ optimal. À ce moment, les moyennes $\bar{x}_a$ convergeraient éventuellement à R^*, le meilleur score de résumé extractif d'un document, pour les phrases a faisant partie du résumé optimal. Or, tout l'intérêt repose dans les moyennes $\bar{x}_a$ pour les phrases ne faisant pas partie du résumé optimal. Comme UCB va progressivement choisir de meilleurs résumés en s'assurant de bien explorer l'espace des résumés possibles, la moyenne $\bar{x}_a$ associée à une phrase a pourra servir d'estimateur de sa qualité extractive.

Intuitivement, 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, difficiles à discerner. En minimisant le regret, UCB va donc naturellement obtenir plus d'échantillons de résumés contenant les excellentes phrases. Or, comme les excellentes phrases sont limitées, il faut échantillonner plus de résumés les contenant avant d'obtenir une bonne estimation de leur valeur, contrairement aux phrases insatisfaisantes dont la valeur est rapidement estimée après quelques échantillons. Il

est donc justifié d'utiliser les quantités mesurées par UCB pour obtenir de l'information sur toutes les phrases d'un document, pas seulement pour trouver le résumé optimal.

Concrètement, nous proposons d'utiliser les moyennes $\bar{x}_a$ produites par un UCB 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 proportions originales entre les moyennes. Les $\bar{x}'_a$ sont ensuite utilisé pour produire les cibles y_a selon

$$y_a = 10^{-10(1-\bar{x}'_a)}. \tag{3.4}$$

L'intuition derrière l'équation (3.4) est qu'une phrase avec une moyenne 10 % inférieure à une autre 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 meilleure phrase selon UCB.

3.2.2 Expériences

Si l'intuition de l'utilisation des quantités calculées par UCB comme cibles est naturelle pour le cas où le nombre de pas de temps T effectué est immense, il demeure néanmoins important de vérifier comment ces cibles sont adéquates sur un nombre de pas de temps plus restreint. Mais, comment savoir quand on approche de ce stade d'optimalité et que l'on peut considérer les cibles comme adéquates ? Réponse : quand le score du résumé le plus prometteur selon les moyennes $\bar{x}_a$, nommons le R_t, commence à converger vers le score R^* du résumé extractif optimal. En mesurant la sous-optimalité $\Delta_t = R^* - R_t$ des cibles générées par UCB, on peut donc savoir à partir de quel moment les cibles seraient satisfaisantes pour être utilisées en entraînement.

On reprend donc le jeu de développement présenté à la section 2.1.1 pour mener des expériences sur la rapidité de l'obtention de cibles satisfaisante par UCB. Les résultats des figures 3.1 et 3.2 sont obtenus en calculant les Δ_t à partir de l'algorithme 3. Notons que, pour mettre UCB à l'échelle de chaque document d, l'exploration que l'on utilise est plutôt guidée par $\beta' = \frac{\beta}{|d|}$. Il est aussi 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.

Algorithme 3 UCB combinatoire 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**
    $\text{UCB}_a = \bar{x}_a + \frac{\beta}{|d|}\sqrt{\frac{2 \ln t}{n_a}}$
  $\mathcal{M}_t = \text{3-arg max UCB}_a$
  $r_t = \text{ROUGE}(\mathcal{M}_t, s)$
  Mettre à jour $\bar{x}_a$ et n_a pour $a \in \mathcal{M}_t$

3.2.3 Résultats

La figure 3.1 présente comment l'hyperparamètre β influence la convergence de UCB. On remarque que $\beta = 10$ (courbe verte sur la figure) semble être le bon compromis entre exploration et exploitation, réussissant à converger à $\Delta_t \approx 0.05$ après 250 pas de temps de UCB.

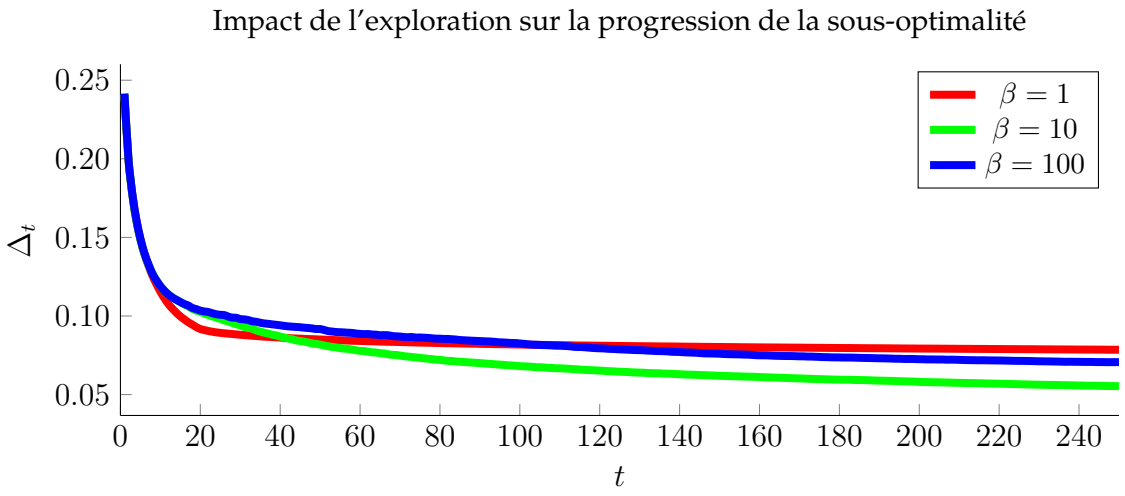

FIGURE 3.1 – Impact du paramètre d'exploration β utilisé par UCB sur la convergence. Δ_t représente la différence entre le score ROUGE du meilleur résumé selon UCB et celui généré par l'oracle.

La figure 3.2 évalue dans quelle mesure la taille des documents influe sur la sous-optimalité des cibles générées par UCB. On remarque d'abord que $\beta = 10$ (courbe verte) est la meilleure configuration pour toutes les tailles de document. Il est donc adéquat de prendre la même valeur de β pour tous les documents.

Aussi, on remarque que l'apprentissage converge plus tôt pour les documents qui sont plus

courts. Il serait donc pertinent de faire croître le nombre de pas de temps de UCB 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.

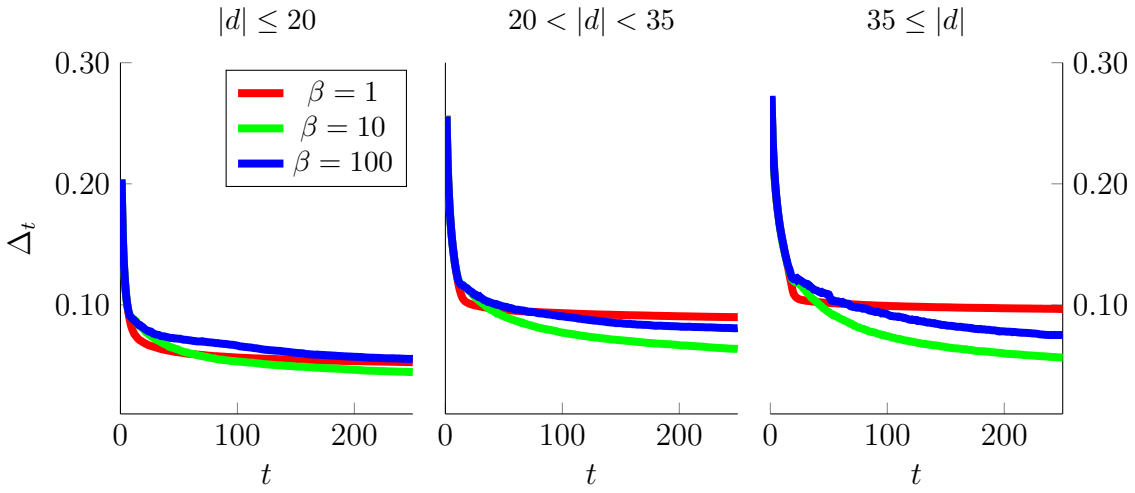

FIGURE 3.2 – Impact de la taille du document sur la convergence de UCB. Δ_t représente la différence entre le score ROUGE du meilleur résumé selon UCB et celui généré par l'oracle.

3.3 CombiSum

On propose maintenant un algorithme basé sur les cibles générées par UCB : CombiSum. Conformément au cadre expérimental uniformisé présenté à la section 2.1, CombiSum réutilise exactement la même architecture neuronale que BanditSum. 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 $\hat{y} = \pi_\theta(d)$:

$$l(\hat{y}, y) = \frac{1}{|y|} \sum_{i=1}^{|y|} \big[y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i) \big]. \tag{3.5}$$

En se basant sur les résultats de la section précédente, on explore deux fonctions $T(d)$ déterminant le nombre T de pas de temps utilisés pour générer les cibles UCB 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 il sera possible de valider laquelle des deux fonctions permet le meilleur entraînement.

3.3.1 Expériences

Une description détaillée de CombiSum est fournie à l'algorithme 4. On effectue un entraînement sur le jeu de données CNN/DailyMail en parcourant dans son entièreté le jeu de 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.

Algorithme 4 CombiSum

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

tant que vrai **faire**
  batch $\sim \mathcal{D}^B$ ▷ On pige la minibatch du jeu de données
  $\nabla = \mathbf{0}$
  pour tout $(d,s) \in$ batch **faire**
    $\hat{y} = \pi_\theta(d)$
    Exécuter $T(d)$ pas de temps de UCB et obtenir $\bar{x}$.
    Générer les cibles y à partir de $\bar{x}$ ▷ Selon (3.4)
    $\nabla = \nabla + \nabla_\theta l(\hat{y}, y)$ ▷ Selon (3.5)
  $\theta = \theta - \alpha\nabla$

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 binaire générées par l'oracle. Étant donné le facteur aléatoire présent dans l'entraînement, nous effectuons 5 entraînements distincts pour chaque T et présentons la moyenne des résultats obtenus.

3.3.2 Résultats

Les résultats obtenus sont présentés dans la figure 3.3 et le tableau 3.1. Ici, UCB linéaire et fixe font référence respectivement à l'utilisation à l'utilisation T_l et T_f pour déterminer le nombre de pas de temps effectués par UCB.

Tout d'abord, la figure 3.3 illustre l'évolution de la performance sur le jeu de validation des modèles selon les cibles utilisées. On constate d'abord que la performance en validation avec les cibles UCB 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 UCB.

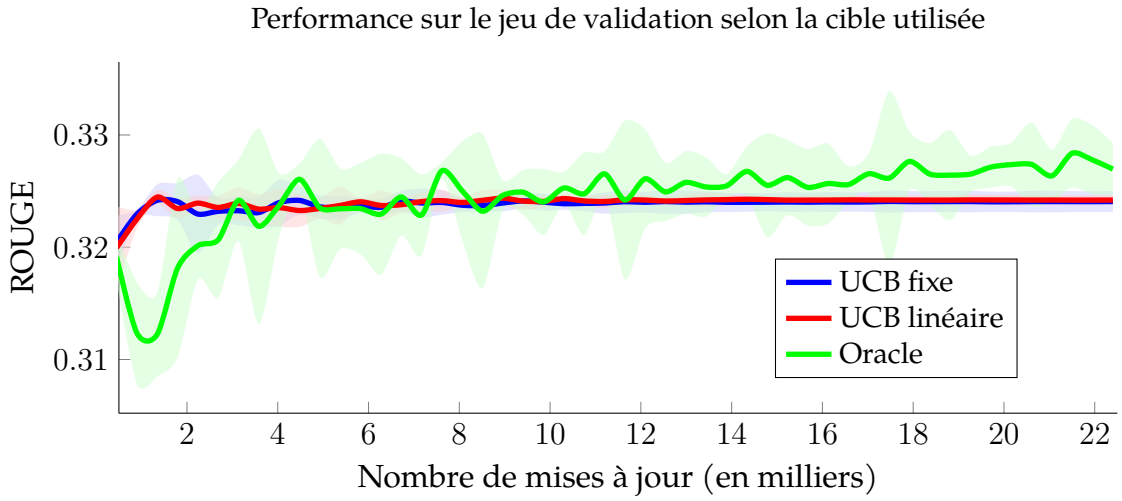

FIGURE 3.3 – Courbe d'apprentissage de CombiSum selon la cible UCB utilisée. Un intervalle de confiance à 95 % calculé à partir de 5 entraînements distincts est rapporté pour chaque cible.

Le tableau 3.1 présente la performance obtenue par 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
UCB fixe	40.04 ± 0.11	$\mathbf{18.18 \pm 0.14}$	36.46 ± 0.15	$\mathbf{31.56 \pm 0.13}$
UCB linéaire	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 – Performance sur le jeu de test.

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. Nous élaborons davantage sur les potentielles justifications de cette performance et discutons d'une piste de solution envisageable au chapitre 5.

3.4 Conclusion

Dans ce chapitre, nous avons proposé de nouvelles cibles plus riches pour l'entraînement d'un modèle de génération de résumés. Nos nouvelles cibles sont basées sur une version de UCB adaptée au bandit combinatoire, dont nous avons validé l'applicabilité sur notre jeu de développement. Enfin, nous avons présenté comment les cibles peuvent être utilisées dans un algorithme que nous avons nommé CombiSum.

Le prochain chapitre présente une autre formulation bandit pouvant être utilisée pour générer des cibles riches : le bandit combinatoire linéaire. En mode linéaire, le bandit combinatoire peut être résolu encore plus rapidement et le prochain chapitre représente donc une version potentiellement plus efficace des cibles présentées dans ce chapitre.

Chapitre 4

Bandit combinatoire linéaire

L'approche combinatoire pure présentée au chapitre précédent présente un inconvénient notoire : elle n'utilise pas de relation de similarité entre les phrases pour propager de l'information entre deux phrases similaires. Il est possible de faire cela sous la formulation du bandit combinatoire linéaire, où on possède des représentations vectorielles de chacune des actions et on fait l'hypothèse qu'il existe un vecteur propre au problème qui relie les représentations de phrases à leur récompense espérée.

Ce chapitre suit une structure très similaire au chapitre précédent. D'abord, on présente comment la génération du résumé d'un document peut être interprétée un problème de bandit combinatoire (Cesa-Bianchi and Lugosi, 2012) linéaire. On montre ensuite comment l'algorithme Linear Upper Confidence Bound (LinUCB) (Chu et al., 2011) peut être utilisé pour résoudre ce type de problème et comment il peut être employé pour générer des cibles pour un système de génération de résumés. L'applicabilité des cibles proposées est par la suite validée empiriquement à l'aide d'un jeu de données de développement. Enfin, 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 LinCombiSum. On met LinCombiSum à l'épreuve sur notre cadre expérimental uniformisé pour juger de son efficacité.

4.1 Formulation combinatoire linéaire

La formulation combinatoire linéaire n'est rien de plus que l'application du principe combinatoire (3.1) au bandit stochastique linéaire (1.1.3). Pour garder la présentation digeste, nous nous contentons 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. Il est supposé qu'il existe un vecteur de récompense ω_* reliant un 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 similairement à la variante multitâche du bandit combinatoire, où l'agent sélectionne m actions à chaque pas de temps t. Les vecteurs d'actions $\tilde{a}$ sélectionnés sont additionnés pour former une *super-action* $\tilde{\mathcal{M}} = \sum_{i=1}^{m} \tilde{a}_i$ pour laquelle l'environnement retourne une récompense $X_{\tilde{\mathcal{M}},t}$. 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* choisies $\tilde{\mathcal{M}}_t$.

4.1.1 Représentations vectorielles de phrases

Les représentations vectorielles utilisées pour les phrases sont basées sur l'observation que les calculs de ROUGE sont essentiellement basés sur les comptes de n-grammes . On choisit donc de générer, pour chaque phrase d_i d'un document d, 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, un vecteur parcimonieux $\tilde{d}_i$ de taille N, pour le nombre N de n-grammes dans le document, représentant le nombre d'occurrences de chaque n-gramme dans la phrase. Dans nos expériences, on limite l'exploration des n-grammes aux unigrammes, afin de garder une taille raisonnable pour les vecteurs parcimonieux générés.

On bâtit ensuite une matrice parcimonieuse P dont les rangées sont les vecteurs parcimonieux de chaque phrase du document. La dimension de la matrice P est ensuite réduite via une analyse sémantique latente (Kolda and O'Leary, 1998), applicable naturellement dans notre cas de matrice parcimonieuse de grande taille. Le processus retourne 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_i et, comme mentionné à la section 1.1.4, LinUCB serait équivalent à UCB. Comme pour les scores ROUGE, les vecteurs $\tilde{a}$ associés à un document peuvent être pré-calculés pour ne pas avoir à les recalculer inutilement à chaque fois que l'on traite un document.

Maintenant que l'on a bien défini le problème de bandit combinatoire linéaire, regardons comment LinUCB peut être utilisé pour le résoudre à la prochaine section.

4.1.2 LinUCB pour bandit combinatoire linéaire

Tel que décrit à la section 1.1.4, Linear Upper Confidence Bound (LinUCB) est une approche permettant de minimiser le pseudo-regret cumulatif pour les problèmes de bandit stochastique linéaire. Pour ce faire, LinUCB maintient des bornes supérieures $\text{UCB}_a(t)$ pour chaque action $\tilde{a} \in \tilde{\mathcal{A}}$ et chaque pas de temps t

$$\text{UCB}_a(t) = \langle \omega_t^\top, \tilde{a} \rangle + \beta \sqrt{\tilde{a}^\top \mathbf{V_t}^{-1} \tilde{a}}, \tag{4.1}$$

pour

$$\hat{\omega}_t = \mathbf{V_t}^{-1} \mathbf{b_t}, \quad \text{pour} \quad \mathbf{V_t} = \mathbf{I} + \sum_{\tau=1}^{t} \tilde{a}_\tau \tilde{a}_\tau^\top, \quad \mathbf{b_t} = \sum_{\tau=1}^{t} \tilde{a}_\tau X_\tau.$$

Ici, $\tilde{a}_\tau$ et X_τ représentent respectivement l'action prise et sa récompense associée perçue au pas de temps numéro τ alors que $\hat{\omega}_t$ représente la meilleure estimation de ω_* après t pas de temps.

LinUCB peut encore une fois être étendu au problème de bandit combinatoire. En effet, il suffit de choisir à chaque pas de temps t la *super-action* maximisant la borne supérieure définie par UCB sur les actions $\tilde{a} \in \tilde{\mathcal{A}}$. Dans notre cas, cela revient à sélectionner la *super-action* qui additionne les trois actions avec la borne supérieure maximale, i.e.

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

où $\text{UCB}_a(t)$ est défini selon (4.1).

La prochaine section décrit le second volet de notre contribution : l'utilisation de LinUCB pour la génération de cibles extractives d'un document.

4.2 Génération de cibles par LinUCB

La génération de cibles avec LinUCB est presque identique à celle avec UCB présentée à la section 3.2. Dans notre cas, comme LinUCB 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 $\hat{x}_a$ pour générer les cibles. 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 $x_{\tilde{a}} \in [0, 1]$ en conservant proportions originales entre les moyennes. Les $x_{\tilde{a}}$ sont ensuite utilisés pour produire les cibles y_a selon

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

4.2.1 Expériences

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

utilisé pour les expériences se trouve à l'algorithme 5. Notons que, pour mettre LinUCB à l'échelle de chaque document d, l'exploration que l'on utilise est plutôt guidée par $\beta' = \frac{\beta}{|d|}$.

Algorithme 5 LinUCB combinatoire pour génération de résumé

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

$\mathbf{V} = \mathbf{I}$
$\mathbf{b} = \mathbf{0}$
pour $t = 1, ..., T$ **faire**
    $\omega_t = \mathbf{V}^{-1}\mathbf{b}$
    pour $\tilde{a} \in \tilde{\mathcal{A}}$ **faire**
        $\text{UCB}_a = \langle \omega_t^\top, \tilde{a} \rangle + \frac{\beta}{|d|}\sqrt{\tilde{a}^\top \mathbf{V}^{-1}\tilde{a}}$
    $\mathcal{M}_t = 3\text{-arg max UCB}_a$
    $X_t = \text{ROUGE}(\mathcal{M}_t, s)$
    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 actuelle diffère légèrement du pseudocode présenté en utilisant la nature de la matrice $\mathbf{V}$ pour éviter d'avoir à calculer son inverse à chaque ronde et être computationnellement plus efficace. Les détails techniques de cette manipulation sont disponibles dans l'annexe A pour le lecteur intéressé.

4.2.2 Résultats

Les résultats obtenus sont présentés dans les figures 4.1 et 4.2. 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 LinUCB. On remarque d'abord que, comme il fallait s'y attendre, LinUCB converge nettement plus rapidement à des cibles de bonne qualité que UCB. En effet, l'erreur Δ_t de près 0.05 observée avec UCB après 250 pas de temps est plutôt observable après 50 pas de temps de LinUCB. On remarque aussi que $\beta = 10^8$ (courbe bleue sur la figure) semble être un bon choix, réussissant à converger légèrement plus rapidement que $\beta = 10^7$.

La figure 4.2 évalue dans quelle mesure la taille des documents influe sur la sous-optimalité des cibles générées par UCB. La valeur $\beta = 10^8$ (courbe bleue) semble encore une fois opti-

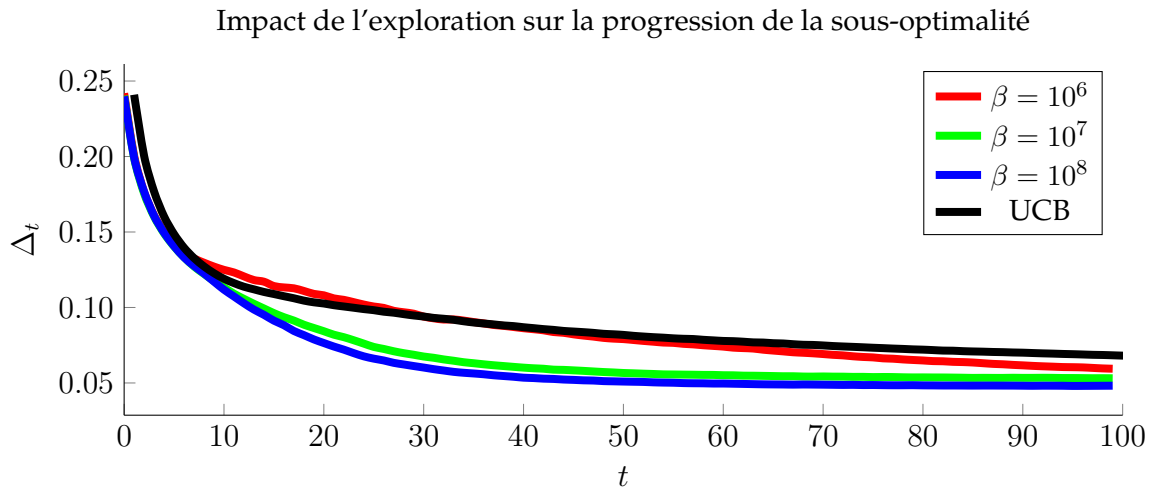

FIGURE 4.1 – Impact du paramètre d'exploration β utilisé par LinUCB sur la convergence. Δ_t représente la différence entre le score ROUGE du meilleur résumé selon LinUCB et celui généré par l'oracle.

male et ce, pour toutes les tailles de document. Similairement à ce qui avait été observé pour les cibles générées par UCB, il semble que de rouler LinUCB avec plus de pas de temps est bénéfique pour les documents plus longs.

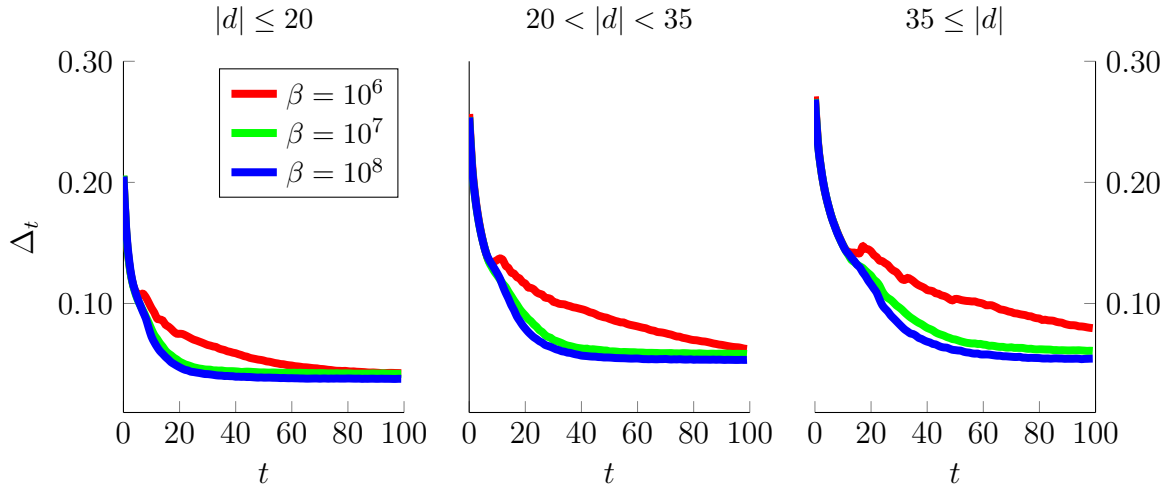

FIGURE 4.2 – Impact de la taille du document sur la convergence de LinUCB. Δ_t représente la différence entre le score ROUGE du meilleur résumé selon UCB et celui généré par l'oracle.

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 LinUCB au lieu de UCB 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 $\hat{y} = \pi_\theta(d)$:

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

En se basant sur les résultats de la section précédente, on explore deux fonctions $T(d)$ déterminant le nombre T de pas de temps utilisés pour générer les cibles UCB 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.

4.3.1 Expériences

Une description détaillée de LinCombiSum est fournie à l'algorithme 6. On effectue un entraînement sur le jeu de données CNN/DailyMail en parcourant dans son entièreté le jeu de 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.

Algorithme 6 LinCombiSum

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

tant que vrai **faire**
  batch $\sim \mathcal{D}^B$ ▷ On pige la minibatch du jeu de données
  $\nabla = \mathbf{0}$
  pour tout $(d,s) \in$ batch **faire**
    $\hat{y} = \pi_\theta(d)$
    Obtenir les représentations $\tilde{a}$ des phrases de d
    Exécuter $T(d)$ pas de temps de UCB et obtenir $\langle \hat{\omega}_T, \tilde{a}_i \rangle$
    Générer les cibles y à partir de $\langle \hat{\omega}_T, \tilde{a}_i \rangle$ ▷ Selon (4.3)
    $\nabla = \nabla + \nabla_\theta l(\hat{y}, y)$ ▷ Selon (4.4)
  $\theta = \theta - \alpha \nabla$

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 binaire générées par l'oracle telles que décrites à la section 3.2. Étant donné le facteur aléatoire présent dans l'entraînement, nous effectuons 5 entraînements distincts pour chaque T et présentons la moyenne des résultats obtenus.

4.3.2 Résultats

Les résultats obtenus sont présentés dans la figure 4.3 et le tableau 4.1. Ici, UCB linéaire et fixe font référence respectivement à l'utilisation à l'utilisation T_l et T_f pour déterminer le nombre de pas de temps effectués par UCB.

Tout d'abord, la figure 4.3 illustre l'évolution de la performance sur le jeu de validation des modèles selon les cibles utilisées. On constate d'abord que la performance en validation avec les cibles UCB 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 LinUCB.

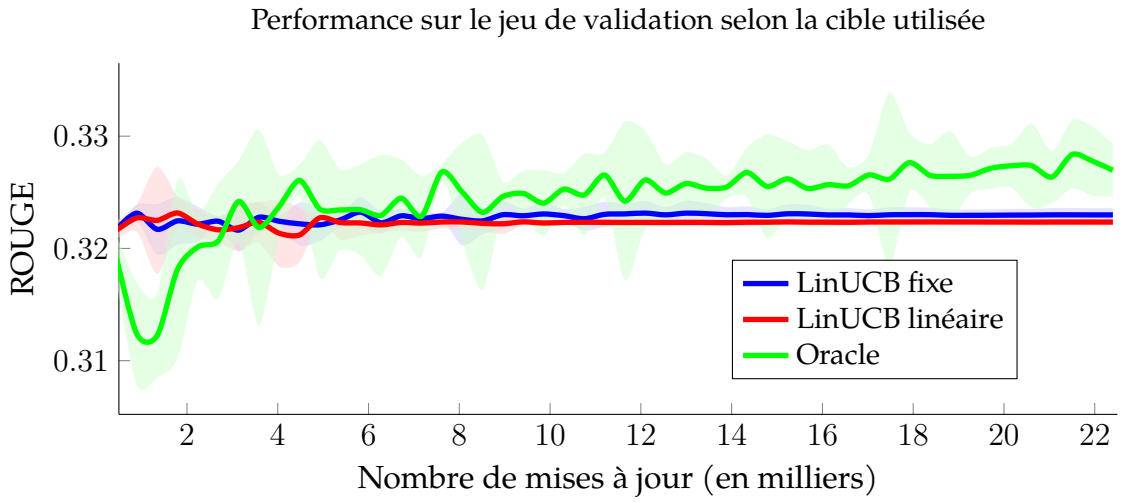

FIGURE 4.3 – Courbe d'apprentissage de LinCombiSum selon la cible utilisée. Un intervalle de confiance à 95 % calculé à partir de 5 entraînements distincts est rapporté pour chaque cible.

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.

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.

Cible utilisée	ROUGE-1	ROUGE-2	ROUGE-L	ROUGE
Lead-3	39.59	17.68	36.21	31.16
LinUCB fixe	39.91 ± 0.04	$\mathbf{18.11 \pm 0.05}$	36.33 ± 0.06	31.45 ± 0.05
LinUCB linéaire	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 – Performance sur le jeu de test.

En somme, les résultats obtenus par LinCombiSum sont décevants, ne se distinguant que très légèrement de la référence Lead-3. Nous élaborons davantage sur les potentielles justifications de cette performance et discutons d'une piste de solution envisageable au chapitre 5.

4.4 Conclusion

Dans ce chapitre, nous avons proposé de nouvelles cibles plus riches pour l'entraînement d'un modèle de génération de résumés. Nos nouvelles cibles sont basées sur une version de LinUCB adaptée au bandit combinatoire, dont nous avons validé l'applicabilité sur notre jeu de développement. Enfin, nous avons présenté comment les cibles peuvent être utilisées dans un algorithme que nous avons nommé CombiSum.

Dans le prochain chapitre, on compare de manière exhaustive les cibles générées par UCB et LinUCB ainsi que les modèles qu'elles génèrent.

Chapitre 5

Impact des cibles extractives

Les nouvelles cibles riches proposées dans les chapitres 3 et 4 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.

Dans ce chapitre, nous commençons par analyser en profondeur les différences entre les cibles que nous proposons et les cibles binaires traditionnelles. On présente ensuite comment les cibles employées influencent l'entraînement, en termes de modèle produit et de perte empirique.

5.1 Différences entre les cibles considérées

On rapporte à la figure 5.1 la distribution moyenne des cibles évaluées dans ce mémoire pour le jeu d'entraînement. Pour les cibles riches basées sur UCB et LinUCB, on présente seulement la version basée sur une augmentation linéaire du nombre de pas de temps T effectués, car les résultats empiriques n'ont pas démontré de différence significative avec la version fixe. Notons aussi que, pour que les cibles soient toutes à la même échelle, on normalise les cibles y issues de UCB et LinUCB pour que leur somme soit égale à 3 comme les cibles binaires.

La figure 5.1 témoigne de l'emphase mise par les cibles sur les phrases se situant au début d'un document. En moyenne, les cibles riches issues de UCB et LinUCB accordent une importance plus faible aux premières phrases que les cibles générées par l'oracle.

5.2 Différences à l'entraînement

La figure 5.2 montre que la perte empirique progresse de manière similaire peu importe la cible.

La figure 5.3 montre clairement que l'entraînement avec les cibles issues de UCB et LinUCB

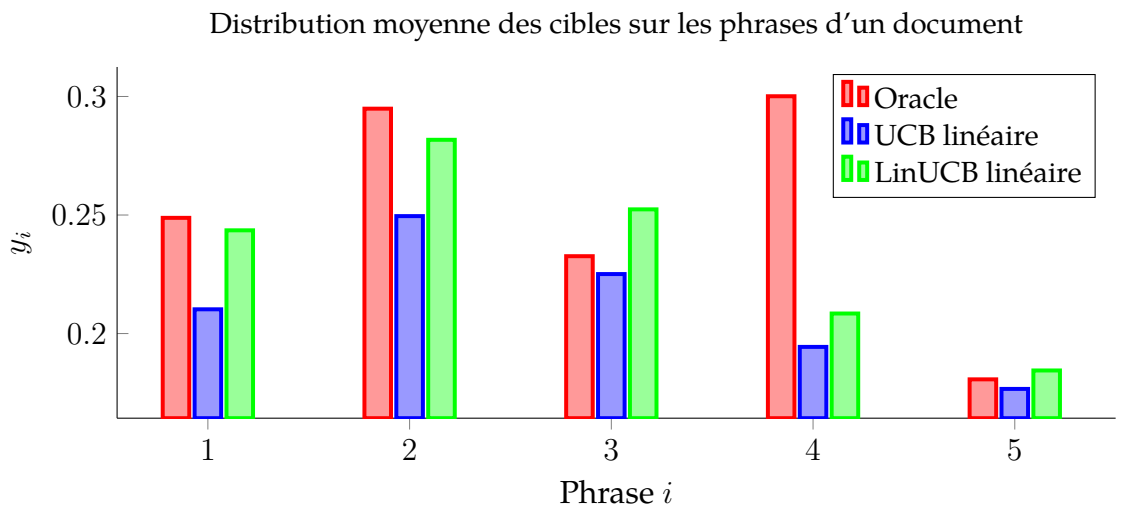

FIGURE 5.1 – Distribution moyenne des cibles sur les phrases d'un document sur du jeu d'entraînement du jeu de données CNN/DailyMail.

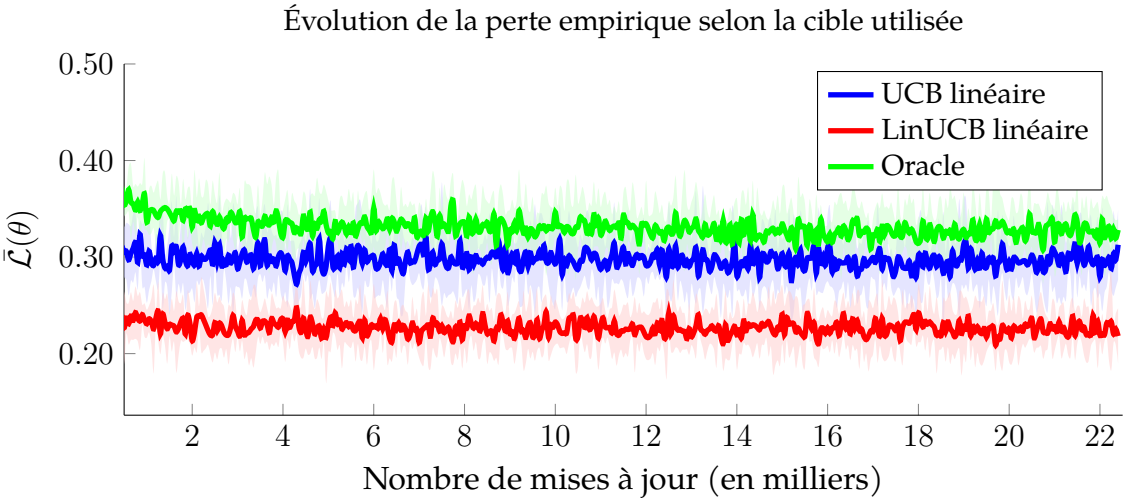

FIGURE 5.2 – Courbe d'évolution de la perte empirique selon la cible utilisée. Un intervalle de confiance à 95 % calculé à partir de 5 entraînements distincts est rapporté pour chaque cible.

converge très fortement à une distribution favorisant les premières phrases d'un document. Nous faisons l'hypothèse qu'une première cause de ce comportement est l'attractivité du minimum local qui consiste à prédire les phrases seulement en fonction de leur index. Or, les modèles entraînés 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 que nous. La seule différence entre nos expériences et BertSumExt est dans le modèle neuronal : on utilise un modèle léger basé sur deux LSTM alors qu'ils

utilisent un modèle BERT, un ordre de magnitude plus gros. Notre hypothèse est donc que les mauvaises performances obtenues avec nos cibles sont dues à un manque d'expressivité du réseau de neurones utilisé. Ainsi, en utilisant nos cibles avec le modèle BertSumExt, il n'y a pas de réelle raison pour que la performance ne soit pas au minimum similaire.

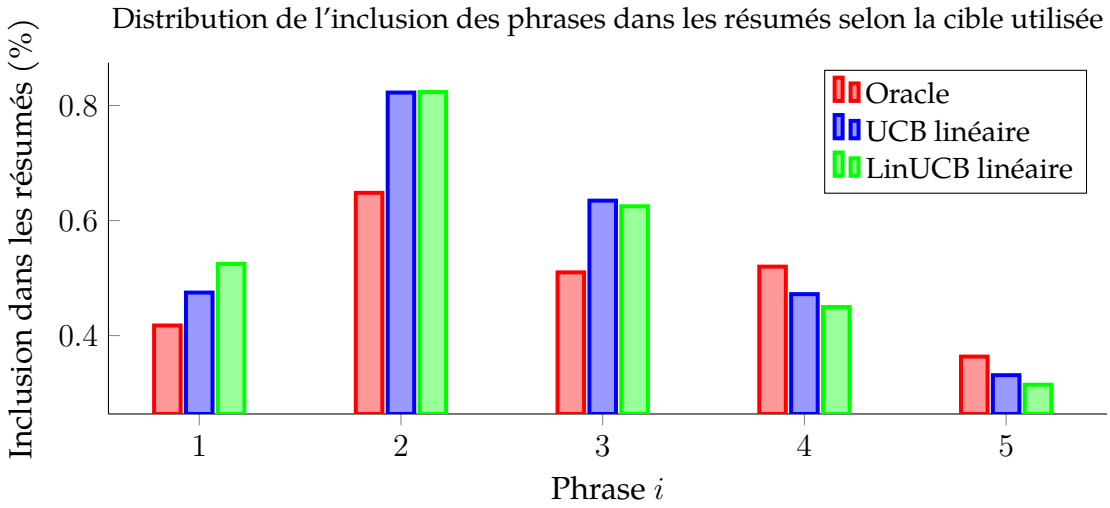

FIGURE 5.3 – Distribution moyenne des phrases retenues sur le jeu de test jeu de données CNN/DailyMail.

5.3 Conclusion

Cibles générées par UCB et LinUCB ont moins de variance (comme attendu). Les modèles entraînés prédisent dans une très grande proportion des résumés issus du début du document.

Conclusion

Dans ce mémoire, nous avons abordé la problématique de l'entraînement de modèles de génération de résumés de documents. Nous avons considéré les résumés dits extractifs, qui sont bâtis en sélectionnant des phrases du document initial. Pour mener nos expérimentations, nous nous sommes intéressés 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. Nous avons commencé par présenter BanditSum (Dong et al., 2018), qui voit la génération du résumé de n'importe quel document comme un bandit contextuel. BanditSum représente l'état de l'art en termes d'efficacité computationnelle, atteignant une performance inférieure de seulement 2 points ROUGE à l'état de l'art tout en ayant un entraînement plusieurs ordres de magnitude moins coûteux computationnellement.

Nous avons ensuite présenté comment la génération du résumé d'un document peut être vue comme un bandit combinatoire (Cesa-Bianchi and Lugosi, 2012). En modifiant légèrement l'algorithme UCB (Auer et al., 2002), nous avons démontré qu'il est possible d'apprendre de manière efficace à résoudre le problème de la génération de résumé d'un document. Nous avons alors introduit le premier volet de notre contribution en proposant des cibles d'entraînement basées sur l'exécution de UCB. Les cibles proposées sont utilisées par un nouvel algorithme d'entraînement nommé CombiSum.

Pour poursuivre, on a démontré que l'on pouvait encore accélérer la convergence sur le problème de bandit combinatoire en utilisant les relations linéaires entre les phrases. À cet effet, on fait appel à une version modifiée de l'algorithm LinUCB (Li et al., 2010) pour résoudre le nouveau problème. Nous avons montré comment des cibles d'entraînement peuvent aussi être extraites de l'éxécution de LinUCB, ce qui représente le second volet de notre contribution. Les cibles proposées sont utilisées par un nouvel algorithme d'entraînement nommé LinCombiSum.

Pour finir, on a évalué en détails comment les cibles obtenues avec UCB et LinUCB influent sur la convergence des procédures d'entraînement de génération de résumés. Pour ce faire,

on les a comparées avec les cibles d'entraînement binaires issues d'un oracle, représentant les cibles utilisées par les modèles de l'état de l'art. Notre analyse a permis d'identifier que les mauvaises performances obtenus avec les cibles que nous proposons sont probablement explicables par la faible expressivité de l'architecture neuronale retenue.

Travaux futurs

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

Premièrement, les approches abordées considèrent toujours que les résumés extractifs d'un document contiennent un nombre fixe de phrases. Il serait intéressant d'incorporer des formulations plus souples comme Luo et al. (2019) qui permettent de sélectionner un nombre variable de phrases d'un document.

Deuxièmement, on n'a utilisé aucun algorithme de bandit combinatoire dédié, on s'est plutôt contenté d'utiliser des modifications aux algorithmes UCB et LinUCB. Il serait intéressant de voir la différence de performance quand on utilise les algorithmes dédiés spécifiquement à cette tâche comme Combes et al. (2015) par exemple.

Troisièmement, il serait intéressant 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, peut-être qu'avec un modèle plus expressif, les cibles proposées pourraient être apprises plus efficacement et mener à des modèles rivalisant avec l'état de l'art.

Annexe A

Gains computationnels LinUCB

TODO: Uniformisation notation vecteurs d'action

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

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

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

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

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

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

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

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

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

qui est la version utilisée dans notre implémentation de LinUCB.

Annexe B

Graphiques avec variance

TODO: Tous les graphiques manquants

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

Graphiques du chapitre 4

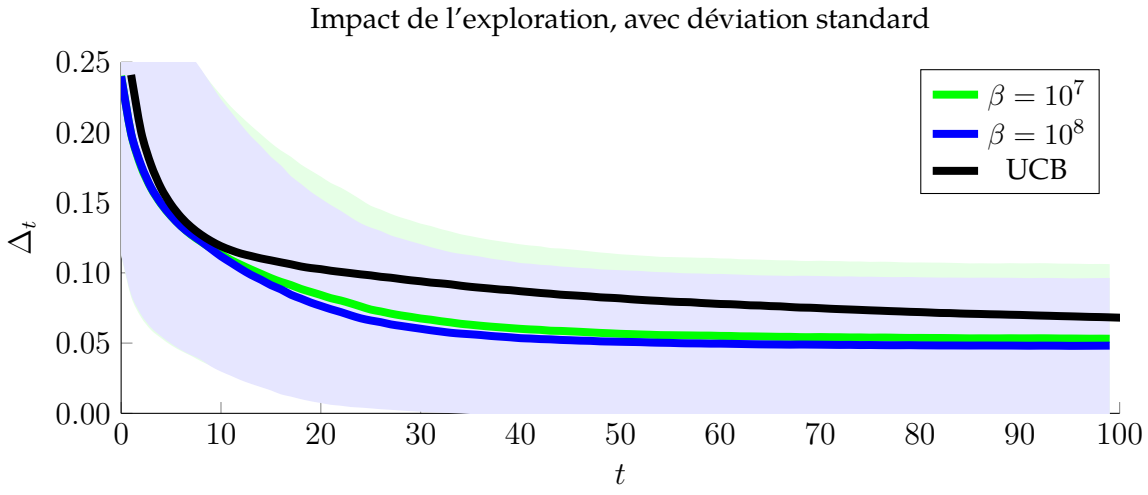

FIGURE B.1 – Impact du paramètre d'exploration β utilisé par LinUCB sur la convergence.

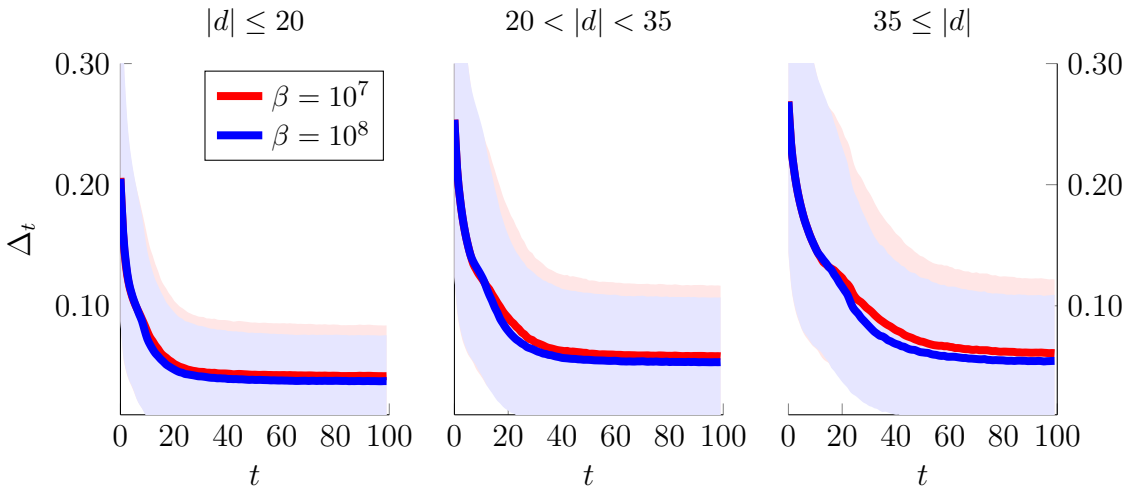

FIGURE B.2 – Impact de la taille du document sur la convergence, avec variance.

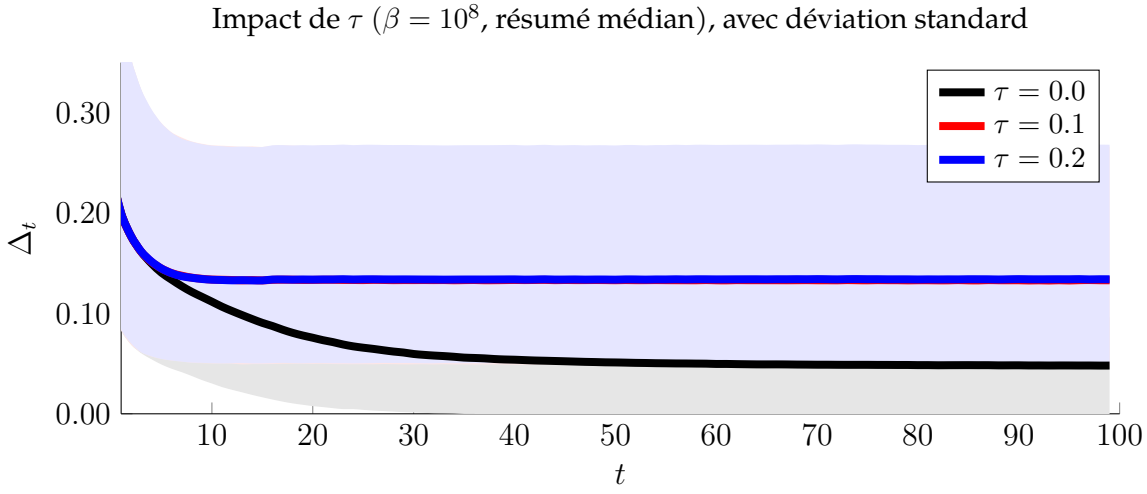

FIGURE B.3 – Impact de la similarité de distributions a priori avec une distribution uniforme, selon le nombre t de pas de temps effectuées.

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

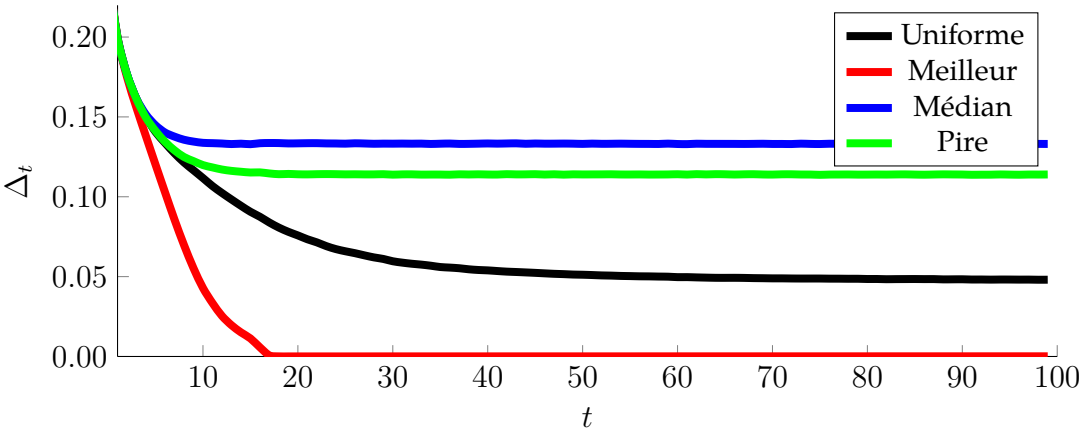

FIGURE B.4 – Impact de la similarité de distributions à priori avec une distribution uniforme, selon le nombre t de pas de temps effectuées.

Bibliographie

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

F. Almeida and G. Xexéo. Word embeddings : A survey, 2019.

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

N. Cesa-Bianchi and G. Lugosi. Combinatorial bandits. *Journal of Computer and System Sciences*, 78(5), 2012. ISSN 0022-0000. doi : https://doi.org/10.1016/j.jcss.2012.01.001. URL http://www.sciencedirect.com/science/article/pii/S0022000012000219. JCSS Special Issue : Cloud Computing 2011.

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

R. Combes, M. S. Talebi, A. Proutiere, and M. Lelarge. Combinatorial bandits revisited. In *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'15, page 2116–2124, Cambridge, MA, USA, 2015. MIT Press.

J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. BERT : Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics : Human Language Technologies, Volume 1* (*Long and Short Papers*), Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. doi : 10.18653/v1/N19-1423. URL https://www.aclweb.org/anthology/N19-1423.

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

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

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

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

I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. The MIT Press, 2016. ISBN 0262035618.

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

A. Joulin, E. Grave, P. Bojanowski, and T. Mikolov. Bag of tricks for efficient text classification, 2016.

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

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

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*), 2020.

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

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

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

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

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, 2017. ISSN 0925-2312. doi : https://doi.org/10.1016/j.neucom.2016.12.038. URL http://www.sciencedirect.com/science/article/pii/S0925231216315533.

Y. Liu and M. Lapata. Text summarization with pretrained encoders. 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*), 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*), Hong Kong, China, Nov. 2019. Association for Computational Linguistics. doi : 10.18653/v1/D19-1300. URL https://www.aclweb.org/anthology/D19-1300.

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

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

S. Narayan, S. B. Cohen, and M. Lapata. Ranking sentences for extractive summarization with reinforcement learning. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics : Human Language Technologies, Volume 1* (*Long Papers*), New Orleans, Louisiana, June 2018. Association for Computational Linguistics. doi : 10.18653/v1/N18-1158. URL https://www.aclweb.org/anthology/N18-1158.

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

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

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

M. Peyrard. Studying summarization evaluation metrics in the appropriate scoring range. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, Florence, Italy, July 2019. Association for Computational Linguistics. doi : 10.18653/v1/P19-1502.

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

H. Robbins. Some aspects of the sequential design of experiments. *Bulletin of the American Mathematical Society*, 58(5) :527–535, 1952. URL http://www.projecteuclid.org/DPubS/Repository/1.0/Disseminate?view=body&id=pdf_1&handle=euclid.bams/1183517370.

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

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

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

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

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

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

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*, 2017.

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

Y. Wu, M. Schuster, Z. Chen, Q. V. Le, M. Norouzi, W. Macherey, M. Krikun, Y. Cao, Q. Gao, K. Macherey, J. Klingner, A. Shah, M. Johnson, X. Liu, Łukasz Kaiser, S. Gouws, Y. Kato, T. Kudo, H. Kazawa, K. Stevens, G. Kurian, N. Patil, W. Wang, C. Young, J. Smith, J. Riesa,

A. Rudnick, O. Vinyals, G. Corrado, M. Hughes, and J. Dean. Google's neural machine translation system : Bridging the gap between human and machine translation, 2016.

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

K. Xu, J. Ba, R. Kiros, K. Cho, A. Courville, R. Salakhutdinov, R. Zemel, and Y. Bengio. Show, attend and tell : Neural image caption generation with visual attention, 2016.

J. Zhang, Y. Zhao, M. Saleh, and P. Liu. PEGASUS : Pre-training with extracted gap-sentences for abstractive summarization. In H. D. III and A. Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, Virtual, 13–18 Jul 2020. PMLR. URL http://proceedings.mlr.press/v119/zhang20ae.html.